XGL Test Suite User's Guide

SunSoft, Inc.
A Sun Microsystems, Inc. Business
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

THE NETWORK IS THE COMPUTER™

Contents

Figures

Tables

Introduction 1

The Denizen Test Suite is a set of graphics verification programs used to test the accuracy of a particular XGL implementation. This chapter gives an overview of Denizen, and a description of its component parts.

The Denizen Test Suite is not intended to be a debugging tool, but instead to provide a verification tool for customers so that they can ensure the accuracy of their hardware implementation via XGL applications.

Overview of Denizen

The Denizen Test Suite is a group of shell scripts and C programs designed to use the XGL library to render objects and evaluate results. Denizen contains approximately 740 test programs that test every XGL function defined at the API and the major internal components of the XGL library. Denizen provides a script that can either run all or a select set of the test programs. Each test evaluates its results automatically. Denizen creates a log of events, errors, and failures that can be compared to previous test runs. Ports of XGL to IHV hardware should produce Denizen pass rates similar to those measured for the reference frame buffers (8- and 24-bit nonaccelerated frame buffers). The following files are included with the Denizen product:

- Binaries for tests and libdenizen
- Source code for tests
- Setup, install, and run scripts
- Inspector tool

- 8-bit and 24-bit reference images
- 8-bit and 24-bit failure logs
- Install, build, and run documentation
- Open XGL bug list

Denizen Test Types

Denizen consists of two types of test programs, sampling method (SM) tests and comparison method (CM) tests. The SM tests are programs that call the XGL library to render an object. These tests sample the pixels that are displayed, check their placement and color, and indicate pass or fail results. Some SM tests also test XGL nonrendering functionality (such as set and get functions).

The CM tests compare an image created using XGL with images that have been previously created[1]. CM tests classify the new image as either *certified* (matches a previously accepted image), or *uncertified* (does not match reference image).

1. Only 8- and 24-bit reference images are included so if the device has a different depth, you may opt not to run these tests. Also, inspector tool does not support other depths (see options listed in Chapter 3, “Options for Running Denizen”).

Denizen Directory Tree

Figure 1-1 illustrates the Denizen directory structure.

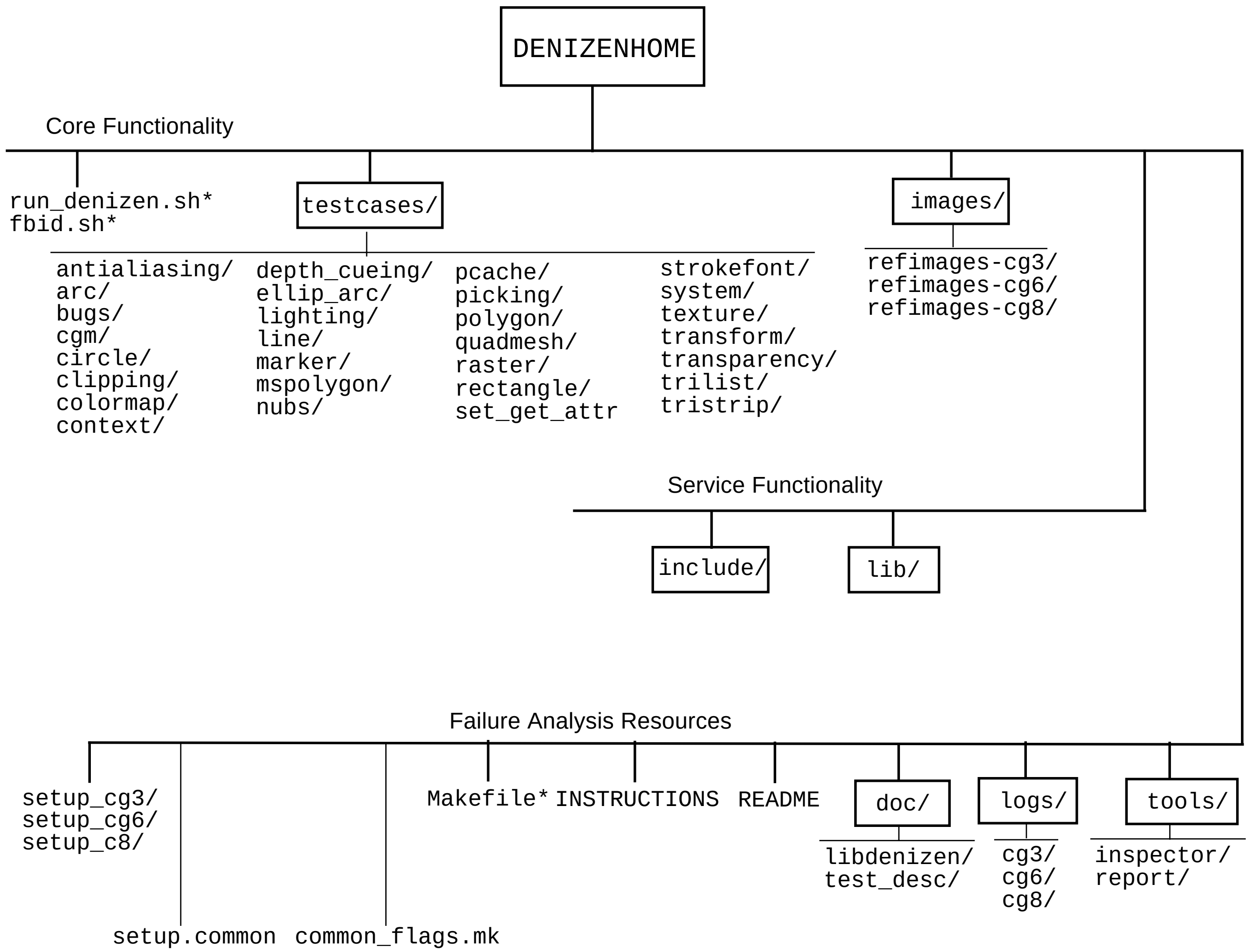

Figure 1-1 Denizen Directory Tree

Core Functionality

run_denizen.sh

Shell script that executes the Denizen test suite. This script runs the C programs in the `testcases/` subdirectories.

fbid.sh

This script is used by `denizen.sh` for configuration information.

testcases/

Directory containing the source and executable files for test programs called by `run_denizen.sh`. This directory contains a subdirectory for each test area. Within each test area subdirectory are test program files for that area, and four ASCII files listing specific tests. These ASCII files are used by the test type options of `run_denizen.sh`:

- INDEX_TESTS—A list of tests that use index color (`-index`)
- RGB_TESTS—A list of tests that use RGB color (`-rgb`)
- SM_TESTS—A list of tests that use the Sampling Method (`-sm`)
- CM_TESTS—A list of tests that use the Comparison Method (`-cm`)

images/

Directory containing the images used for CM testing. This directory contains subdirectories for each device type.

The reference images for the CM tests are in `refimages-cg3`, `refimages-cg6`, and `refimages-cg8`. If the testing device is 8-bit, set the `REFIMAGE` environment variable to *refimages-8bit*. For 24-bit devices, set the `REFIMAGE` environment variable to *refimages-24bit*.

Service Functionality

include/

Directory containing the header files used in the test programs from the `testcases/` directory.

lib/

Directory containing the verification library used by the test programs.

Failure Analysis Resources

setup_cg3, setup_cg8, setup_cg6

Basic setup for running Denizen on cg3, cg8, and cg6.

setup.common

Basic setup script (dependent upon $XGLHOME being set).

common_flags.mk

Selects the correct compiler and loader.

Makefile*

Makefile for the Denizen directory.

INSTRUCTIONS

ASCII file containing information on defining and customizing environment variables.

README

ASCII file containing information on run_denizen.sh. The same information is provided in this document; however, README is in man page style.

doc/test_desc/

Directory containing ASCII documentation for the test programs contained in testcases/. Each file represents a test area, and describes the tests available for that area. Chapters dedicated to each test area summarize this information.

logs/

Directory containing the results of the Denizen test runs on XGL. There is a log for each of the reference frame buffers.

tools/

Directory containing tools to run the comparison methodology tests.

Verifying Your Implementation 2

Verifying your XGL implementation using Denizen involves performing these steps:

1. Setting the appropriate environment variables.
2. Invoking `run_denizen.sh`, which executes Denizen and creates a verification log.
3. Comparing your results to the current XGL release test results.

This chapter shows you how to create verification logs similar to the logs stored in the `logs/` directory, and describes a method for comparing them. In addition, different options for running Denizen are discussed. Finally, several examples show you how to design your own test runs.

OpenWindows must be running before you can invoke `run_denizen.sh`.

1. **Set the environment variable `DENIZENHOME`.**
   You must use the absolute or full path name of the local directory. In this case, assume your home directory is `/home/xgl`.

```
hostname% setenv DENIZENHOME /opt/sunwddk/xgl/src/test_suite/denizen
```

2. **Change directories to the location of the Denizen files.**
   The Denizen files must either reside on your workstation, or be mounted on your system. In this case, assume that the Denizen files were installed from CD into **/opt/sunwddk/xgl/src/test_suite/denizen**.

```
hostname% cd /opt/sunwddk/xgl/src/test_suite/denizen
```

3. **Verify the installation.**
   Change the directory to $DENIZENHOME, and list the contents. You should see the following information.

```
hostname% cd $DENIZENHOME
hostname% ls *
Makefilecommon_flags.mksetup_cg3
INSTRUCTIONSrun_denizen.shsetup_cg6
READMEsetup.commonsetup_cg8

doc:
libdenizen/test_desc/

images:
refimages-cg3/
refimages-cg6/
refimages-cg8/

include:

lib:
libdenizen.so

logs:
XGL log for 8- and 24-bit reference frame buffers
```

```
testcases:

    antialiasing/      pcache/           strokefont/
    arc/               picking/          system/
    bugs/              polygon/          texture/
    cgm/               quadmesh/         transform/
    circle/            raster/           transparency/
    clipping/          rectangle/        trilist/
    colormap/          set_get_attr      tristrip/
    context/


tools:
Makefile   inspector/   report/
```

Setting Environment Variables

The Denizen Test Suite uses several environment variables. Table 2-1 lists the required environment variables and their meanings. Absolute paths must be used for the directories.

Table 2-1 Required Environment Variables

Environment Variable	Meaning
`DENIZENHOME`	The local directory where Denizen has been installed, and where testing will be performed. Note that this `$DENIZENHOME/lib` should be listed in `$LD_LIBRARY_PATH.`
`LD_LIBRARY_PATH`	A colon-separated list of directories in which to search for the XGL and OpenWindows libraries. Denizen will use the version of XGL encountered first in this list of directories.
`XGLHOME`	The directory that contains the `include` and `lib` directories of the version of XGL to be tested. Note that this `$XGLHOME/lib` should be listed in `$LD_LIBRARY_PATH`.
`OPENWINHOME`	The location of OpenWindows libraries and include files. Most OpenWindows start-up scripts set this environment variable or require that this variable be set. Note that this `$OPENWINHOME/lib` should be listed in `$LD_LIBRARY_PATH`.

Table 2-1 Required Environment Variables *(Continued)*

Environment Variable	Meaning
FB_NAME	The device being tested, for example cg3. It must also be set manually when running individual tests. An error message results if not set. Supported devices are cg3, cg8, and cg6.
REFIMAGE	The location of the approved comparison images.
CURIMAGE	The location of the nonmatching images.
DENIZEN_DESTDIR	The location for the log file if this variable is set. Otherwise, the default is your home directory.

The following information details the settings for these environment variables in our example.

```
hostname% setenv DENIZENHOME /opt/sunwddk/xgl/src/test_suite/denizen
hostname% setenv XGLHOME /opt/SUNWits/graphics-sw/xgl
hostname% setenv OPENWINHOME /usr/openwin
hostname% setenv LD_DIBRARY_PATH $XGLHOME/lib:$OPENWINHOME/lib:$DENIZENHOME/lib
hostname% setenv FB_NAME cg3
hostname% setenv REFIMAGE $DENIZENHOME/images/refimages-cg3
hostname% setenv CURIMAGE /tmp
```

Creating Verification Logs

To verify your XGL implementation, you need to create verification logs comparable to those supplied in the `logs/` directory.

```
hostname% cd $DENIZENHOME
hostname% run_denizen.sh
```

You can run Denizen from DENIZENHOME, or from a directory other than DENIZENHOME. If you choose a directory other than DENIZENHOME, use the whole path name of the script.

```
hostname% $DENIZENHOME/run_denizen.sh
```

The test programs are run in sequence. The status of each test is displayed in the window from which you started Denizen, and is also stored in the verification log. Because many of the tests read pixels from the test window to verify results, it is important that the window is not obscured. You may want to run Denizen at night, or on test machines not used for other work.

The verification log file is stored in either `$DENIZEN_DESTDIR`, or if that is not set, in your home directory. The file name is encoded with the hostname and the date the test suite was run (`run_denizen.log <hostname><date>`). This log shows the configuration that was tested, the test programs Denizen ran, and the results of each test. If you run Denizen again on the same date, the current log will be moved to a `<filename>.old` file.

Comparing Your Results

By comparing verification logs with the logs supplied in the `logs/` directory, you can see how your XGL implementation compares to the current XGL release. You may want to move your verification log to the `logs/` directory, and rename it appropriately.

```
hostname% mv run_denizen.log.hostnameSep16
logs/mylog.<fb_type>.Sep16
```

Your XGL implementation will comply with the current XGL release if the logs are similar. Discrepancies between the logs indicate differences in your implementation.

Each of the tests is described in the following chapters. More information on using Denizen is provided in Chapter 3, “Options for Running Denizen.” These chapters can help you analyze any discrepancies.

If you run the comparison methodology image tests, and the images produced by your device pipeline differ from the reference images, the images with discrepancies will be put in the directory you have set for `CURIMAGE`. You then need to use the inspector tool (discussed in the following section) to verify your images manually. The inspector tool makes it easy to view the differences between images.

Using the Inspector Tool to Analyze Images

If you choose to run the comparison methodology tests, this tool helps you manually compare a failing (nonmatching/uncertified) image to a reference image.

The environment variables `REFIMAGE` and `CURIMAGE` must be set when using the inspector tool. The inspector tool is fairly simple to use because it provides a graphical user interface.

Inspector tool lists the image files it finds in the `$CURIMAGE` directory. To view the new image with a reference image, choose the image name and press the `Load` button.

The reference image is shown in the left window, and the image under scrutiny is shown in the right window. If the discrepancies are not readily apparent, a `diff` button allows you to see the differences more easily. If the new nonmatching image is verified as acceptable, you can save this image as the new reference image for the device being tested. So for example, you might create a new `REFIMAGE` area, `$DENIZENHOME/images/refimage-<new fbname>`, move your verified images into it, and set the `REFIMAGE` environment variable to that new directory.

Options for Running Denizen 3

There are several options and arguments that you can supply to the `run_denizen.sh` script, as well as some environment variables that can help you gain more information about test failures. This chapter describes the Denizen options, arguments, and environment variables.

run_denizen.sh Options and Arguments

The `run_denizen.sh` script has the following usage:

run_denizen.sh [**-rgb** | **-index** | **-sm** | **-cm**] [*testarea1*, *testarea2*, ...]

Options

The first four options are mutually exclusive, and indicate the test type to be used. If no test type option is given, all tests for the given test areas will be run.

`-rgb`

Uses only tests listed in the `RGB_TESTS` file in the directory for each of the test areas tested. These are the tests that use the RGB color window rasters.

`-index`

Uses only tests listed in the `INDEX_TESTS` file in the directory for each of the test areas tested. These are the tests that use the index color window rasters.

-sm

Uses only tests listed in the SM_TESTS file in the directory for each of the test areas tested. These are the tests that use the sampling method (SM).

-cm

Uses only tests listed in the CM_TESTS file in the directory for each of the test areas tested. These are the tests that use the comparison method (CM).

Test Areas

You may provide the run_denizen.sh script with the test areas you want to test. If no test areas are provided, all test areas will be tested. The acceptable test area names are the names of the subdirectories in the testcases/ directory:

```
antialiasing   depth_cueing  pcache        strokefont
arc            ellip_arc     picking       system
bugs           lighting      polygon       texture
cgm            line          quadmesh      transform
circle         marker        raster        transparency
clipping       mspolygon     rectangle     trilist
colormap       nubs          set_get_attr  tristrip
context
```

Errors

The most common error encountered by Denizen users is the improper setting of environment variables. For example, if you forget to set $DENIZENHOME, you will receive a “not found” error message.

Another common error is not setting $FB_NAME. Also, remember to update $LD_LIBRARY_PATH if you change $XGLHOME.

More Environment Variables

Several additional environment variables can be used. They are listed in Table 3-1.

Table 3-1 Additional Environment Variables

Environment Variable	**Meaning**
`MAXFAIL`	A positive integer indicating the number of failures Denizen will allow before aborting the test run. If not specified, the default value is 5.
`VERBOSITY`	A number from 1 to 5 that determines the amount of status messages Denizen displays and stores in the log. The default is 1. A value of 5 will print all detailed status messages.
`FB_DEV`	The device driver of the device being tested, for example `/dev/cgthree0`. This environment variable *must* be set on two-headed systems to indicate which device driver to use. On a single-headed machine it does not have to be set.

Test Run Examples

1. **Change directories to `$DENIZENHOME`.**
   If `$DENIZENHOME` is not set, set it to the full path name of the local `denizen` directory.

```
hostname% setenv DENIZENHOME /opt/sunwddk/xgl/src/test_suite/denizen
hostname% cd $DENIZENHOME
```

2. **Set up the environment for your denizen run, and invoke `run_denizen.sh`**
   This step can be as complicated or simple as you need. Several examples are shown below.

Required environment variables:

```
hostname% setenv DENIZENHOME /opt/sunwddk/xgl/src/test_suite/denizen
hostname% setenv XGLHOME /opt/SUNWits/graphics-sw/xgl
hostname% setenv OPENWINHOME /usr/openwin
hostname% setenv LD_DIBRARY_PATH $XGLHOME/lib:$OPENWINHOME/lib:$DENIZENHOME/lib
hostname% setenv FB_NAME cg3
hostname% setenv REFIMAGE $DENIZENHOME/images/refimages-cg3
hostname% setenv CURIMAGE /tmp
```

Optional environment variables:

```
hostname% setenv MAXFAIL 20
hostname% setenv VERBOSITY 3
hostname% setenv DENIZEN_DESTDIR $DENIZENHOME
```

3. **Run Denizen**

   Here are several examples of ways to run Denizen. If you are running Denizen from a directory other than `$DENIZENHOME`, add `$DENIZENHOME/` in front of the example commands below.

   Running all Denizen tests:

```
hostname% run_denizen.sh
```

   Running tests in selected test areas:

```
hostname% run_denizen.sh transform lighting
```

   Running only the SM tests in selected test areas. If you use this option, you do not have to set the `$REFIMAGE` and `$CURIMAGE` environment variables.

```
hostname% run_denizen.sh -sm arc line
```

Running the comparison method tests only. The $REFIMAGE and $CURIMAGE environment variables must be set if you use this option.

```
hostname% run_denizen.sh -cm
```

Running an individual test. Setting $VERBOSITY to 3 and $MAXFAIL to 20 gives you more information. You must set $FB_NAME when running individual tests.

```
hostname% setenv FB_NAME cg3
hostname% cd testcases/arc
hostname% arc4
```

Test Make Example

This section describes how to build individual Denizen tests. Note that to build Denizen, you need the XGL include files that are available with the SDK. In each test area, there is a Makefile that will *make* the tests on an individual basis. If you intend to build the tests, you must modify the include Makefile, common_flags.mk, so that CC and LD point to the correct compiler and loader.

Again, $DENIZENHOME and $FB_NAME must be set. To make the individual test, change to the test area directory and type make *test_name*.

```
hostname% cd testcases/line
hostname% make line2
```

Denizen Library Functions 4

This chapter describes the functions that are used by denizen programs. Some of these functions are verification functions that check for content accuracy in the frame buffer or memory raster. The remaining functions are utility functions needed by the test programs.

Relevant Denizen Data Types

```
typedef struct dmatrix {
    Xgl_data_type data_type;
    Xgl_trans_dimension dim;
    union {
        Xgl_matrix_i2d * i2d;
        Xgl_matrix_f2d * f2d;
        Xgl_matrix_f3d * f3d;
        Xgl_matrix_d2d * d2d;
        Xgl_matrix_d3d * d3d;
    } data;
} Dtp_mtx;
```

```
typedef enum {DTP_PT_LINE, DTP_PT_LINE_ALT, DTP_PT_EDGE,
DTP_PT_EDGE_ALT, DTP_PT_ARC, DTP_PT_ELARC, DTP_PT_MARKER,
DTP_PT_STEXT, DTP_PT_INTERIOR, DTP_PT_EXTERIOR} Dtp_pt_type;
```

```
typedef enum {DTP_LINE, DTP_EDGE} Dtp_loe;

typedef struct dtp_pt{
    Dtp_pt_type pt_type;
    float x,y,z;
} Dtp_pt;

union circ_2d_arc {
        Xgl_circle_f2d *c;
        Xgl_arc_f2d *a;
};

typedef struct {
        int     c_a;
        union circ_2d_arc p;
} wch2d_gdp;

union circ_3d_arc {
        Xgl_circle_f3d *c;
        Xgl_arc_f3d *a;
};

typedef struct {
        int     c_a;
        union circ_3d_arc p;
} wch3d_gdp;
```

Commonly Used Arguments

```
Xgl_sys_state sys_st /*System state */
int (* failure) ()   /*Pointer to function for handling special
    instances of failures. For example sometimes
    a test is known to fail at a certain (x, y)
    location and the failure is a permissible
    failure. This failure may be occurring
```

```
    because of poor test design or poor
    verification function. In such a case the
    test can provide a pointer to a function
    which checks to see if the failure is one of
    the special failure instances. Then it may
    handle the failure in whatever way is
    appropriate. This can be NULL, and is in fact
    NULL for most tests./
Xgl_ctx ctx          /*XGL context */
```

Denizen Library Functions

CGM Functions

int setpointlist (

Xgl_pt_list	****pl,***	/* Destination pointlist to store data provided by other parameters passed */
Xgl_pt_f3d	****ptsf3d,***	/* Point coordinates to store inside returned pointlist */
Xgl_pt_type	***pt_type,***	/* Point type to store inside returned pointlist along with appropriate type for allocated points */
Xgl_bbox	****bbox,***	/* Bounding box to store inside pointlist */
int	***n)***	/* Number of points desired in pointlist */

pl->pt_type — pt_type which can only come from the values: `XGL_PT_I2D`, `XGL_PT_FLAG_I2D`, `XGL_PT_FLAG_F2D`, `XGL_PT_F3D`, `XGL_PT_FLAG_F3D`, and `XGL_PT_F2D`

pl->bbox — bbox
pts pointer to point type structure desired allocated for *n* number of points with the actual vertex data assigned from that contained within ptsf3d, and

As an example:

```
pl->pts.i2d[0] ... pl->pts.i2d[n-1]  ptsf3d[0].x,ptsf3d[0].y ...
ptsf3d[n-1].x,ptsf3d[n-1].y
```

This holds true when *pts* is a pointer to i2d, f2d, f3d, flag_i2d, flag_f2d and flag_f3d.

int circ2d_set_plist (

Xgl_circle_list	****pl,***	/* Destination pointlist to store data provided by other parameters passed */
Xgl_pt_f2d	****ptsf2d,***	/* pointer to list of center points for n circles */
Xgl_bbox	****bbox,***	/* bounding box to store inside pointlist and which contains all circles */
int	***n,***	/* number of circles in the list */
float	****radius,***	/* pointer to list of radius for n circles */

Return circle pointlist assembled with:

pl->num_circles	value *n*
pl->type	value `XGL_MULTICIRCLE_F2D`
pl->bbox	value *bbox*
pl->circles.f2d	*f2d* pointer to *n Xgl_circle_f2d* structures allocated dynamically within this function

pl->circles.f2d[0].center.x	ptsf2d[0].x
pl->circles.f2d[0].center.y	ptsf2d[0].y
.	.
.	.
.	.
pl->circles.f2d[n-1].center.x	ptsf2d[n-1].x
pl->circles.f2d[n-1].center.y	ptsf2d[n-1].y
pl->circles.f2d[0].center.flag	flag[0]
.	.
.	.
.	.
pl->circles.f2d[n-1].center.flag	flag[n-1]
pl->circles.f2d[0].center.radius	radius[0]
.	.

```
        .                                   .
        .                                   .
pl->circles.f2d[n-1].center.radius    radius[n-1]   /* pointer to list of flags
                                                       indicating whether
                                                       edges are desired */
```

int circi2d_set_plist (

```
Xgl_circle_list    *pl,        /* Destination pointlist to store
                                  data provided by other
                                  parameters passed */
Xgl_pt_i2d         *ptsi2d,    /* pointer to list of center points
                                  for n circles */
Xgl_bbox           *bbox,      /* bounding box to store inside
                                  pointlist and which contains all
                                  circles */
int                n,          /* number of circles in the list */
float              *radius,    /* pointer to list of radius for n
                                  circles */
int                *flag)      /* pointer to list of flags
                                  indicating whether edges are
                                  desired */
```

Return circle pointlist assembled with:

pl->num_circles	value *n*
pl->type	value `XGL_MULTICIRCLE_I2D`
pl->bbox	value *bbox*
pl->circles.i2d	*i2d* pointer to n *Xgl_circle_i2d* structures allocated dynamically within this function
pl->circles.i2d[0].center.x	ptsi2d[0].x
pl->circles.i2d[0].center.y	ptsi2d[0].y
.	.
.	.
.	.
pl->circles.i2d[n-1].center.x	ptsi2d[n-1].x
pl->circles.i2d[n-1].center.y	ptsi2d[n-1].y
pl->circles.i2d[0].center.flag	flag[0]
.	.
.	.
.	.

pl->circles.i2d[n-1].center.flag	flag[n-1]
pl->circles.i2d[0].center.radius	radius[0]
.	.
.	.
.	.
pl->circles.i2d[n-1].center.radius	radius[n-1]

int arc2d_set_plist (

Xgl_arc_list	****pl,***	/* Destination pointlist to store data provided by other parameters passed */
Xgl_pt_f2d	****ptsf2d,***	/* pointer to list of center points for *n* arcs */
Xgl_bbox	****bbox,***	/* bounding box to store inside pointlist and which contains all arcs */
int	***n,***	/* number of arcs in the list */
float	****radius,***	/* pointer to list of radius for *n* arcs */
int	****flag,***	/* pointer to list of flags indicating whether edges are desired */
float	****start,***	/* pointer to list of angles in radians from the positive x-axis in the plane of the circle to the start point of the arc for *n* arcs*/

Returns arc pointlist assembled with:

pl->num_arcs	value *n*
pl->type	value `XGL_MULTIARC_F2D`
pl->bbox	value *bbox*
pl->arcs.f2d	*f2d* pointer to *n* *Xgl_arc_f2d* structures allocated dynamically within this function
pl->arcs.f2d[0].center.x	ptsf2d[0].x
pl->arcs.f2d[0].center.y	ptsf2d[0].y
.	.
.	.
.	.

pl->arcs.f2d[n-1].center.x	ptsf2d[n-1].x
pl->arcs.f2d[n-1].center.y	ptsf2d[n-1].y
pl->arcs.f2d[0].center.flag	flag[0]
.	.
.	.
.	.
pl->arcs.f2d[n-1].center.flag	flag[n-1]
pl->arcs.f2d[0].center.radius	radius[0]
.	.
.	.
.	.
pl->arcs.f2d[n-1].center.radius	radius[n-1]
pl->arcs.f2d[0].start_angle	start[0]
.	.
.	.
.	.
pl->arcs.f2d[n-1].start_angle	start[n-1]
pl->arcs.f2d[0].stop_angle	stop_angle[0]
.	.
.	.
.	.
pl->arcs.f2d[n-1].stop_angle	stop_angle[n-1]

int arc3d_set_plist (

Xgl_arc_list	****pl,***	/* Destination pointlist to store data provided by other parameters passed */
Xgl_pt_f3d	****ptsf3d,***	/* pointer to list of center points for *n*
Xgl_bbox	****bbox,***	/* bounding box to store inside pointlist and which contains all arcs */
int	***n,***	/* number of arcs in the list */
float	****radius,***	/* pointer to list of radius for *n* arcs */
int	****flag,***	/* pointer to list of flags indicating whether edges are desired */
int	****norm_flag,***	/* pointer to list of flags indicating whether 3

		direction vectors provided have been normalized and are unit vectors */
int	****unit_flag,***	/* pointer to list of flags indicating whether the third direction vectors provided is normal to the plane in which the arc is rendered */
Xgl_pt_f3d	****dir,***	/* pointer to a list of 3 vector components from which the first two in each group represents the plane in which the arc is going to be built in model coordinates while the last vector provides the normal to this plane when the *unit_flag* indicates TRUE.
float	****start,***	/* pointer to list of angles in radians from the positive x-axis in the plane of the circle to the start point of the arc for *n* arcs */
float	****stop)***	/* pointer to list of angles in radians from the positive x-axis in the plane of the circle to the end point of the arc for *n* arcs*/

Returns arc pointlist assembled with:

pl->num_arcs	value *n*
pl->type	value `XGL_MULTIARC_F3D`
pl->bbox	value *bbox*
pl->arcs.f3d	*f3d* pointer to *n* *Xgl_arc_f3d* structures allocated dynamically within this function
pl->arcs.f3d[0].center.x	ptsf3d[0].x
pl->arcs.f3d[0].center.y	ptsf3d[0].y
.	.
.	.

```
        .                                 .
pl->arcs.f3d[n-1].center.x           ptsf3d[n-1].x
pl->arcs.f3d[n-1].center.y           ptsf3d[n-1].y
pl->arcs.f3d[0].center.flag          flag[0]
        .                                 .
        .                                 .
        .                                 .
pl->arcs.f3d[n-1].center.flag        flag[n-1]
pl->arcs.f3d[0].dir_normalized       norm_flag[0]
        .                                 .
        .                                 .
        .                                 .
pl->arcs.f3d[n-1].dir_normalized     norm_flag[n-1]
pl->arcs.f3d[0].dir_normal           unit_flag[0]
        .                                 .
        .                                 .
        .                                 .
pl->arcs.f3d[n-1].dir_normal         unit_flag[n-1]
pl->arcs.f3d[0].center.radius        radius[0]
        .                                 .
        .                                 .
        .                                 .
pl->arcs.f3d[n-1].center.radius      radius[n-1]
pl->arcs.f3d[0].dir[0]               dir[0]
pl->arcs.f3d[0].dir[1]               dir[1]
pl->arcs.f3d[0].dir[2]               dir[2]
        .                                 .
        .                                 .
        .                                 .
pl->arcs.f3d[n-1].dir[0]             dir[n*3 - 3]
pl->arcs.f3d[n-1].dir[1]             dir[n*3 - 2]
pl->arcs.f3d[n-1].dir[2]             dir[n*3 - 1]
pl->arcs.f3d[0].start_angle          start[0]
        .                                 .
        .                                 .
        .                                 .
pl->arcs.f3d[n-1].start_angle        start[n-1]
pl->arcs.f3d[0].stop_angle           stop_angle[0]
        .                                 .
```

```
        .                               .
        .                               .
pl->arcs.f3d[n-1].stop_angle         stop_angle[n-1]
```

int circ3d_set_plist (

Xgl_circle_list	****pl,***	/* Destination pointlist to store data provided by other parameters passed */
Xgl_pt_f2d	****ptsf3d,***	/* pointer to list of center points for n circles */
Xgl_bbox	****bbox,***	/* bounding box to store inside pointlist and which contains all circles */
int	***n,***	/* number of circles in the list */
float	****radius,***	/* pointer to list of radius for *n* circles */
int	****flag,***	/* pointer to list of flags indicating whether edges are desired */
int	****norm_flag,***	/* pointer to list of flags indicating whether 3 direction vectors provided have been normalized and are unit vectors */
int	****unit_flag,***	/* pointer to list of flags indicating whether the third direction vectors provided is normal to the plane in which the arc is rendered */
Xgl_pt_f3d	****dir)***	/* pointer to a list of three vector components from which the first two in each group represents the plane in which the arc is going to be built in model coordinates while the last

vector provides the normal to this plane when the *unit_flag* indicates TRUE.

Return circle pointlist assembled with:

pl->num_circles	value *n*
pl->type	value `XGL_MULTICIRCLE_F3D`
pl->bbox	value *bbox*
pl->circles.f3d	*f3d* pointer to *n Xgl_circle_f3d* structures allocated dynamically within this function
pl->circles.f3d[0].center.x	ptsf3d[0].x
pl->circles.f3d[0].center.y	ptsf3d[0].y
.	.
.	.
.	.
pl->circles.f3d[n-1].center.x	ptsf3d[n-1].x
pl->circles.f3d[n-1].center.y	ptsf3d[n-1].y
pl->circles.f3d[0].center.flag	flag[0]
.	.
.	.
.	.
pl->circles.f3d[n-1].center.flag	flag[n-1]
pl->circles.f3d[0].dir_normalized	norm_flag[0]
.	.
.	.
.	.
pl->circles.f3d[n-1].dir_normalized	norm_flag[n-1]
pl->circles.f3d[0].dir_normal	unit_flag[0]
.	.
.	.
.	.
pl->circles.f3d[n-1].dir_normal	unit_flag[n-1]
pl->circles.f3d[0].center.radius	radius[0]
.	.
.	.
.	.
pl->circles.f3d[n-1].center.radius	radius[n-1]
pl->circles.f3d[0].dir[0]	dir[0]
pl->circles.f3d[0].dir[1]	dir[1]
pl->circles.f3d[0].dir[2]	dir[2]

```
.                               .
.                               .
.                               .
pl->circles.f3d[n-1].dir[0]     dir[n*3 - 3]
pl->circles.f3d[n-1].dir[1]     dir[n*3 - 2]
pl->circles.f3d[n-1].dir[2]     dir[n*3 - 1]
```

Xgl_cmap* index16tbl (

Xgl_sys_state	***sys_state)***	/* System state */

Returns an XGL colormap (colortable) with 16 colors encoded: (0) Black, (1) Red, (2) Green, (3) Blue, (4) Yellow, (5) Cyan, (6) Magenta, (7) Orange, (8) Brown, (9) Violet, (10) Olive, (11) Grey, (12) Mint Green, (13) Light Red, (14) Light Blue, and (15) White.

Xgl_cmap* colour_table (

Xgl_sys_state	***sys_state)***	/* System state */

Returns an XGL colormap (colortable) with 8 colors encoded: (0) Black, (1) White, (2) Red, (3) Green, (4) Blue, (5) Yellow, (6) Cyan, and (7) Magenta.

int init_bounds (

Xgl_bounds_d3d	****b,***	/* pointer to a potential 3D window boundary, 3D viewport boundary or 3D bbox which is returned filled with the information provided by the remaining parameters */
double	***xmin,***	/* *xmin* boundary information for window boundary, viewport or bbox */
double	***xmax,***	/* *xmax* boundary information for window boundary, viewport or bbox */
double	***ymin,***	/* *ymin* boundary information for window

		boundary, viewport or bbox */
double	***ymax,***	/* *ymax* boundary information for window boundary, viewport or bbox */
double	***zmin,***	/* *zmin* boundary information for window boundary, viewport or bbox */
double	***zmax)***	/* *zmax* boundary information for window boundary, viewport or bbox */

Returns a pointer to a 3D boundary structure, *b*, with the information for its members from the other parameters.

int init_2dbounds (

Xgl_bounds_d2d	****b,***	/* pointer to a potential 2D window boundary, 2D viewport boundary or 2D bbox which is returned filled with the information provided by the remaining parameters */
double	***xmin,***	/* *xmin* boundary information for window boundary, viewport or bbox */
double	***xmax,***	/* *xmax* boundary information for window boundary, viewport or bbox */
double	***ymin,***	/* *ymin* boundary information for window boundary, viewport or bbox */
double	***ymax)***	/* *ymax* boundary

information for window boundary, viewport or bbox */

Returns a pointer to a 2D boundary structure, *b*, with the information for its members from the other parameters.

init_bbox3d (

Xgl_bounds_f3d	****box,***	/* pointer to bbox returned with information fully provided by other parameters.
float	***xmin,***	/* *xmin* boundary information for bbox */
float	***xmax,***	/* *xmax* boundary information for bbox */
float	***ymin,***	/* *ymin* boundary information for bbox */
float	***ymax,***	/* *ymax* boundary information for bbox */
float	***zmin,***	/* *zmin* boundary information for bbox */
float	***zmax)***	/* *zmax* boundary information for bbox */

Returns an initialized bounding box of type *f3d*, *bbox*, with coordinate information provided by the other parameters.

int is_2d_circle (

Xgl_sys_state	***sys_st,***	/* System state */
Xgl_3d_ctx	***ctx,***	/* 3D Context */
Xgl_circle_list	****pl)***	/* circle point list */

Returns 0 upon failure to substantiate the circumference of a 2D circle and 1 upon success.

```
int   is_3d_circle (
        Xgl_sys_state      sys_st,      /* System State */
        Xgl_3d_ctx         ctx,         /* 3D Context */
        Xgl_circle_list    *pl)         /* pointer to a circle list to
                                           verify appears correctly in
                                           the XGL/X window */
```

Returns 0 upon failure to substantiate the circumference of a 3D circle and 1 upon success.

```
int   is_2d_arc (
        Xgl_sys_state      sys_st,      /* System State */
        Xgl_ctx            ctx,         /* 2D Context */
        Xgl_arc_list       *pl)         /* pointer to arc list to verify
                                           on screen */
```

Returns 0 upon failure to substantiate the circumference of a 2D arc and 1 upon success.

```
int   is_3d_arc (
        Xgl_sys_state      sys_st,      /* System State */
        Xgl_ctx            ctx,         /* 3D Context */
        Xgl_arc_list       *pl)         /* pointer to arc list to verify
                                           on screen */
```

Returns 0 upon failure to substantiate the circumference of a 2D arc and 1 upon success.

```
int   is_shaded_quadmesh (
        Xgl_sys_state      sys_st,      /* System State */
        Xgl_object         ctx,         /* 2D or 3D Context */
        int                nbnds,       /* # of polygons, although
                                           assumes only 1 */
        Xgl_pt_list        *plist,      /* array of edge lists */
        Xgl_color_type     type,        /* color type either RGB or
                                           INDEX */
        Xgl_color          color)       /* interior color, not used,
                                           because all point type
```

where shading is anticipated contain color information */

XGL breaks quadmeshes into tristrips before lighting, shading, and color interpolation. As a result, *is_shaded_quadmesh()* breaks quadmeshes into tristrips composed along the diagonal from the upper-left corner to the lower-right corner and passes these two tristrips to *is_shaded_polygon* for verification. Returns 1 upon successful substantiation of quadmesh on the screen and 0 upon failure.

int wide_line (

Xgl_sys_state	***sys_st,***	/* System State */
Xgl_obj	***ctx,***	/* 2D or 3D Context */
int	***x1,***	/* 1st x endpoint in screen space */
int	***y1,***	/* 1st y endpoint in screen space */
int	***x2,***	/* 2nd x endpoint in screen space */
int	***y2,***	/* 2nd y endpoint in screen space */
int	***ew,***	/* Expected width */
Xgl_color_type	***color_type,***	/* color type RGB or INDEX */
Xgl_color	****color)***	/* expected color in RGB or INDEX format */

Returns 1 for substantiation of a line width of ew and 0 upon failure.

int is_ltype (

Xgl_sys_state	***sys_st,***	/* System State */
Xgl_object	***ctx,***	/* 2D or 3D Context */
int	***x1,***	/* 1st *x* endpoint in screen space */
int	***y1,***	/* 1st *y* endpoint in screen space */
int	***x2,***	/* 2nd *x* endpoint in screen space */
int	***y2,***	/* 2nd *y* endpoint in screen

```
                                          space */
Xgl_color        *col,            /*      Expected dash or dot color
                                          either RGB or index */
int              line_type,       /*      line types which can be
                                          among:
                                          PATDASHED,
                                          PATDOTTED,
                                          PATDASHEDDOTTED,
                                          PATDASHDOTDOT,
                                          PATLONGDASH,
                                          PATCGMDOT,
                                          PATCGMDASH,
                                          PATDASHDOT,
                                          PATCENTER or
                                          PATPHANTOM */
int              pixl,            /*      pixel length for dash or dot
                                          */
int              gapl,            /*      pixel length for gap */
int              tdd,             /*      total dash &/or dot pixel
                                          length */
Xgl_color        *gapcol)         /*      color in RGB or index for
                                          gap pixels */
```

Returns 1 on success of verifying patterned line type on screen and 0 otherwise. Only capable of verification for either horizontal or vertical patterned lines and only implemented for index.

Circle Functions

```
int    dtp_iscirc (
         Xgl_sys_state     sys_st,
         Xgl_ctx           ctx,
         Xgl_circle_list   * circle_listp,   /* Circle list to be tested */
         int               (* failure) ())
```

Checks to see if data described by *circle_listp* are indeed drawn as circles. This function just checks to see that pixels of expected color are found around the circumference of the circle and the center is of the expected color. Only some points along the circumference are checked. Works only when *circle_listp->type* is `XGL_MULTICIRCLE_F2D`, `XGL_MULTICIRCLE_F3D`, or

XGL_MULTICIRCLE_AF3D. Returns 1 if all the circles are found to be correct by the above definition. Returns 0 if any of the circles is incorrect. If pixel readback functions called by this function return -1, then this value is returned to the caller.

int dtp_isarc (

Xgl_sys_state	***sys_st,***	
Xgl_ctx	***ctx,***	
Xgl_arc_list	**** arc_listp,***	/* Arc list to be tested */
int	***(* failure)())***	

Checks to see if data described by *arc_listp* are indeed drawn as arcs. This function just checks to see that pixels of expected color are found around the circumference of the arc. It does not distinguish between open and closed arcs. Works only when *arc_listp*->*type* is XGL_MULTIARC_F2D, XGL_MULTIARC_F3D, or XGL_MULTIARC_AF3D. Returns 1 if all the arcs are found to be correct by the above definition. Returns 0 if any of the arcs is incorrect. If pixel readback functions called by this function return -1, then this value is returned to the caller.

int dtp_isellipse (

Xgl_sys_state	***sys_st,***	/* System State */
Xgl_ctx	***ctx,***	/* 3D Context */
Xgl_ell_list	****ellip_listp,***	/* Elliptical arc list to be verified on screen */
int	***(* failure) ()***	/* Pointer to function for handling special instances of failures */

Checks to see if data described by *ellip_listp* are indeed drawn as elliptical arcs. This function just checks to see that pixels of expected color are found around the circumference of the elliptical arc. It does not distinguish between open and closed elliptical arcs. Can handle elliptical arcs not parallel to the *xy* plane and back-faced elliptical arcs but only for point types XGL_MULTIELLARC_F3D and XGL_MULTIELLARC_AF3D. Returns 0 if any of the arcs is incorrect. If pixel readback functions called by this function return -1, then this value is returned to the caller.

Color Selector Functions

```
int   is_ln_selection (
          Xgl_sys_state      sys_state;     /* IN: System state */
          Xgl_3d_ctx         ctx;           /* IN: Current context */
          Xgl_bbox           *bb;           /* IN: bounding box */
          Xgl_pt_list        *pl;           /* IN: pointlist */
          int                op;            /* IN: primitive: not used */
          int                npl)           /* IN: number of pointlists in
                                               pointlist array */
```

is_ln_selection can only handle pointlists with point type XGL_PT_F3D or XGL_PT_COLOR_F3D

Returns 1 upon success of finding the correct line color rendered at the pointlist locations given the XGL_CTX_LINE_COLOR_SELECTOR and the XGL_CTX_LINE_COLOR held by the *ctx* and the vertex color information inside the pointlists.

Returns 0 upon failure.

```
int   is_marker_selection (
          Xgl_sys_state      sys_state;     /* IN: System state */
          Xgl_3d_ctx         ctx;           /* IN: Current context */
          Xgl_pt_list        *pl;           /* IN: pointlist */
          int                op;            /* IN: primitive: not used */
          int                npl;           /* IN: number of pointlists in
                                               pointlist array */
          Xgl_marker         m)             /* marker type:
                                               dot,plus,asterisk,
                                               circle,cross,square,
                                               bowtie_ne, bowtie_nw */
```

Returns 1 upon success of finding the correct marker color rendered at the pointlist locations given the XGL_CTX_MARKER_COLOR_SELECTOR, XGL_CTX_MARKER_COLOR and XGL_CTX_MARKER_SCALE_FACTOR held by the *ctx* and the vertex color information inside the pointlists.

Returns 0 upon failure.

int is_msp_selector_lighting (

Xgl_sys_state	***sys_state;***	/* IN: System state */
Xgl_3d_ctx	***ctx;***	/* IN: Current context */
Xgl_facet_list	****fl;***	/* IN: facet list */
Xgl_pt_list	***pl;***	/* IN: point list */
int	***npl;***	/* IN: number of pointlists in pointlist array */
Xgl_facet_flags	***flag;***	/* IN: facet flag data information */
int	***primname)***	/* IN: primname may only be MSP */

Returns 1 upon success of finding the correct colors at the pointlist locations for `xgl_multi_simple_polygon` given the `XGL_CTX_SURF_FRONT_COLOR_SELECTOR`, `XGL_CTX_SURF_FRONT_COLOR`, `XGL_3D_CTX_SURF_FRONT_ILLUMINATION`, `XGL_3D_CTX_SURF_FRONT_AMBIENT`, `XGL_3D_CTX_SURF_FRONT_DIFFUSE`, `XGL_3D_CTX_SURF_FRONT_DIFFUSE`, `XGL_3D_CTX_SURF_FRONT_SPECULAR_COLOR`, `XGL_3D_CTX_SURF_FRONT_SPECULAR_POWER`, `XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT`, `XGL_3D_CTX_LIGHT_NUM`, `XGL_3D_CTX_LIGHTS`, `XGL_3D_CTX_LIGHTS` and `XGL_3D_CTX_LIGHTS` held by the *ctx*, applicable normals and colors from either or both the facet list and the point list, and the applicable default normal when needed calculated as recommended by the rules prescribed by `XGL_3D_CTX_SURF_GEOM_NORMAL`.

Returns 0 upon failure.

Cannot handle facet flags which indicated the provided facet normal is inconsistent with the implicit facet normal, and `TRUE` value for `XGL_3D_CTX_SURF_FACE_DISTINGUISH`.

int is_qm_selector_lighting (

Xgl_sys_state	***sys_state;***	/* IN: System state */
Xgl_3d_ctx	***ctx;***	/* IN: Current context */
Xgl_facet_list	****fl;***	/* IN: facet list */
Xgl_pt_list	***pl;***	/* IN: point list */

int	***primname;***	/* IN: primname may only be QUADMESH */
int	***row;***	/* IN: number of rows in quad */
int	***col)***	/* IN: number of columns in quad */

Returns 1 upon success of finding the correct colors at the pointlist locations for xgl_quadrilateral_mesh given the XGL_CTX_SURF_FRONT_COLOR_SELECTOR, XGL_CTX_SURF_FRONT_COLOR, XGL_3D_CTX_SURF_FRONT_ILLUMINATION, XGL_3D_CTX_SURF_FRONT_AMBIENT, XGL_3D_CTX_SURF_FRONT_DIFFUSE, XGL_3D_CTX_SURF_FRONT_DIFFUSE, XGL_3D_CTX_SURF_FRONT_SPECULAR_COLOR, XGL_3D_CTX_SURF_FRONT_SPECULAR_POWER, XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT, XGL_3D_CTX_LIGHT_NUM, XGL_3D_CTX_LIGHTS, XGL_3D_CTX_LIGHTS and XGL_3D_CTX_LIGHTS held by the *ctx*, applicable normals and colors from either or both the facet list, and the point list and the applicable default normal when needed calculated as recommended by the rules prescribed by XGL_3D_CTX_SURF_GEOM_NORMAL.

Cannot handle TRUE value for XGL_3D_CTX_SURF_FACE_DISTINGUISH.

Returns 0 upon failure.

int is_selector_lighting (

Xgl_sys_state	***sys_state;***	/* IN: System state */
Xgl_3d_ctx	***ctx;***	/* IN: Current context */
Xgl_facet_list	****fl;***	/* IN: facet list */
Xgl_pt_list	***pl;***	/* IN: point list */
int	***primname)***	/* IN: primitive name may be either TRISTRIP or TRISTAR */

Returns 1 upon success of finding the correct colors at the pointlist locations for xgl_triangle_list and xgl_triangle_strip given the XGL_CTX_SURF_FRONT_COLOR_SELECTOR, XGL_CTX_SURF_FRONT_COLOR,

XGL_3D_CTX_SURF_FRONT_ILLUMINATION, XGL_3D_CTX_SURF_FRONT_AMBIENT, XGL_3D_CTX_SURF_FRONT_DIFFUSE, XGL_3D_CTX_SURF_FRONT_DIFFUSE, XGL_3D_CTX_SURF_FRONT_SPECULAR_COLOR, XGL_3D_CTX_SURF_FRONT_SPECULAR_POWER, XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT, XGL_3D_CTX_LIGHT_NUM, XGL_3D_CTX_LIGHTS, XGL_3D_CTX_LIGHTS and XGL_3D_CTX_LIGHTS held by the *ctx*, applicable normals and colors from either or both the facet list and the point list, and the applicable default normal when needed, calculated as recommended by the rules prescribed by XGL_3D_CTX_SURF_GEOM_NORMAL.

Cannot handle TRUE value for XGL_3D_CTX_SURF_FACE_DISTINGUISH.

Returns 0 upon failure.

Depth_Cueing Functions

apply_indexed_depth_cueing (

Xgl_3d_ctx	***ctx,***	/* IN: Current context */
Xgl_pt_f3d	****pt,***	/* IN: Vertex */
Xgl_color	****col,***	/* IN: Color for that vertex */
Xgl_color	****nc)***	/* OUT: New color, adjusted for depth cueing */

Attenuates the color (**col*), based on the z value of **pt*. The z value is assumed to be in VDC space. This function is used when the XGL_DEV_COLOR_TYPE is XGL_COLOR_INDEX. Works only when XGL_3D_CTX_DEPTH_CUE_MODE is XGL_DEPTH_CUE_LINEAR.

apply_rgbed_depth_cueing (

Xgl_3d_ctx	***ctx,***	/* IN: Current context */
Xgl_pt_f3d	****pt,***	/* IN: Vertex */
Xgl_color	****col,***	/* IN: Color for that vertex */
Xgl_color	****nc)***	/* OUT: New color, adjusted for depth cueing */

Attenuates the color (**col*), based on the z value of **pt*. The z value is assumed to be in VDC space. This function is used when the `XGL_DEV_COLOR_TYPE` is `XGL_COLOR_RGB`. Works only when `XGL_3D_CTX_DEPTH_CUE_MODE` is `XGL_DEPTH_CUE_LINEAR`.

Gen Functions

check_pixel (

Xgl_sys_state	***sys_st,***
Xgl_object	***ctx,***
Xgl_color	****col,***
Xgl_color_type	***color_type,***
int	***x,***
int	***y,***
int	***d,***
int	***should_exist)***

Checks to see if at least one pixel of the specified color either exists (*should_exist=1*) or does not exist (*should_exist=0*) within the area of dimension *d * d* centered at *x*, *y* of the device attached to the context *ctx*. This function now just calls the newer, more flexible, easier to understand function *dpix_chkreg()*. *check_pixel()* has been left in for backward compatibility.

void tbegintest (

char	****name,***	/* test name */
char	****desc)***	/* test description */

Initializes the testing environment by setting the test name and test description, which are printed out in the test suite log when the test program ends. Also sets the default values of certain testing parameters that can be set externally.

void tendtest()

Prints the message indicating if the test failed or passed.

void tfprintf (

char	****fmt,***	/* message fmt */
...)		/* variable data */

Behaves just like *printf*, except that the failure count is incremented. This failure count is used to abort the test if the number of failures for the test has gone above a certain limit. This limit can be set by setting the environment variable MAXFAIL. If the variable is not set, the value of 5 is used.

void tvprintf (

int	***verbosity,***	
char	****fmt,***	/* message format string */
...)		/* variable data */

Except for the argument verbosity, the function behaves just like *printf*. The verbosity is used to filter out messages that are too detailed (detail is defined in terms of current verbosity level). The function compares the value passed as the verbosity argument, with the current verbosity set as an environment variable. If the current verbosity is not set as an environment variable, a default value is used. The message is printed only if the verbosity argument is greater than or equal to (>=) the current verbosity.

void tabort (

char	****format,***	/* message format */
int	***va_alist)***	/* variable data */

Behaves like *printf* except that it also aborts the test.

void tcmbegintest (

char	****name,***	/* test name */
char	****desc)***	/* test description */

Behaves like *tbegintest()* except that this function is used for tests that use comparison methodology (CM). The other function is used in tests that use the sampling methodology (SM).

int dpix_chkreg (

Xgl_sys_state	***sys_st,***	
Xgl_ctx	***ctx,***	
Xgl_color	****color,***	/* color to be tested for */
int	***x,***	/* location in device
int	***y,***	coordinates of the “center” of the region of interest */

int	***w,***	/* dimensions of the region of interest */
int	***h,***	
int	***op)***	/* operation to be used for comparison of the color argument with the colors of pixels in the region of interest */

Checks the region of interest as specified. Context *ctx* must be attached to a device. The region of interest is "centered" at device coordinates (*x,y*). The dimensions of the region of interest are *width* == *w* and *height* == *h*, in device coordinates. If *op* == *DPIX_SOME*, check if *SOME* pixel in the region of interest has color *color*. If *op* == *DPIX_NONE*, check that of all the pixels in the region of interest *NONE* has color *color*. If *op* == *DPIX_EVERY*, check that *EVERY* pixel in the region of interest has color *color*.

int dpix_getreg (

Xgl_sys_state	***sys_st,***	
Xgl_ras	***ras,***	
Xgl_ctx	***ctx,***	
int	***ox,***	/* Location of "center" of region of interest */
int	***oy,***	
int	***w,***	/* Dimension of region of interest */
int	***h,***	
void	****region,***	/* Are in which pixel values read back will be stored */
Xgl_color_type	***ctype)***	

Gets pixel colors from the region of interest. The first time the function is called, it creates the space for the region. In subsequent calls the function will create more space only if needed. If the space needed is less than that which has been already allocated in a previous call, that space will be resumed.

void dtp_setup_cmap4 (

Xgl_sys_state	***sys_st,***	
Xgl_ctx	***ctx,***	
Xgl_cmap	****cmap)***	/* Pointer to colormap object created */

Creates a colormap with four entries, black, white, green and red, and returns a pointer to the colormap object created.

int dtp_pl2f2d (

Xgl_pt_list	****inpl,***	/* pointer to input point lists */
Xgl_usgn32	***num_pt_lists,***	
Xgl_pt_list	*****outpl)***	/* Output point lists */

Converts point lists with arbitrary `2D` XGL data types into point lists of `2D` float type (`XGL_PT_F2D`). This is useful for some tests where the geometry for the primitives has been specified using a point type other than `XGL_PT_F2D`, since most (not all) of the verification functions in libdenizen accept `F2D` and `F3D` point types. The function creates the space needed for the output point lists.

int dtp_pl2f3d (

Xgl_pt_list	****inpl,***
Xgl_usgn32	***num_pt_lists,***
Xgl_pt_list	*****outpl)***

Same as *dtp_plf2d()* except that this function is for 3D point types.

dtp_pt2f3d (

Xgl_pt	****inpt,***
Xgl_pt_type	***inpt_type,***
Xgl_pt_f3d	****outpt)***

Similar to *dtp_pl2f3d()* except that this is for a single point rather than a point list.

double dtp_rand (

float	***min,***
float	***max)***

Returns a random float in the closed interval [min, max].

int dtp_ismarker (

```
Xgl_sys_state     sys_st,
Xgl_ctx           ctx,
Xgl_pt_list       *pl,       /* Point list with marker data */
int               (* failure) ())
```

A simple test for correctness of markers. Checks the x, y locations (assumed to be in device coordinate space) in the point list for presence of a “point” of the right color. Thus, this function works only for markers shaped like asterisk, plus, dot, and so on, but not for circle markers, for example. The point type has to be XGL_PT_F2D or XGL_PT_F3D.

int dtp_isstext2d (

```
Xgl_sys_state     sys_st,
Xgl_ctx           ctx,
char              *str,
Xgl_pt_f2d        *pos,
int               (* failure)())
```

Text checking functions are very unsophisticated. These assume that the text is centered horizontally and vertically and just tests a small region centered at the position of text, checking for some pixels. Use a single character like “X” as the text to be drawn until these are made more sophisticated. Practical experience shows that is not as serious a flaw as it may seem. This function is only used in test programs that are testing transformations or clipping of text, rather than if text is being drawn exactly correctly. Tests for checking the correctness of text without transformations or clipping use their own internal functions for verifications.

int dtp_isstext3d (

```
Xgl_sys_state     sys_st,
Xgl_ctx           ctx,
char              *str,
Xgl_pt_f3d        *pos,
Xgl_pt_f3d        dir[],
int               (* failure)())
```

Same as *dtp_isstext2d()* except that this is for 3D text.

int Getpixarray (

Xgl_sys_state	***sys_st,***
Xgl_win_ras	***raster,***
Xgl_object	***ctx2or3d,***
int	***x,***
int	***y,***
int	***w,***
int	***h,***
void	****data,***
Xgl_color_type	***type)***

Obsolete function left in for backward compatibility. Some older tests still use this function. Now this function just calls the newer *dpix_getreg()*.

void set_rgb_color (

Xgl_color	****color,***
float	***red,***
float	***green,***
float	***blue)***

Sets the value of **color* to the r, g, b values specified in the arguments.

void set_rgb_surf_color (

Xgl_obj	***ctx,***
float	***red,***
float	***green,***
float	***blue)***

Sets the context attribute `XGL_CTX_SURF_FRONT_COLOR` to a color with the r, g, b values provided.

void set_rgb_line_color (

Xgl_obj	***ctx,***
float	***red,***
float	***green,***
float	***blue)***

Sets the context attribute `XGL_CTX_LINE_COLOR` to a color with the r, g, b values provided.

void set_rgb_back_color (

Xgl_obj	***ctx,***
float	***red,***
float	***green,***
float	***blue)***

Sets the context attribute XGL_CTX_BACKGROUND_COLOR to a color with the r, g, b values provided.

void extreme_rgb_values (

Xgl_obj	***ctx,***
float	****red,***
float	****green,***
float	****blue)***

Sets the background color to white, creates a new frame, reads back the color of the pixel at (x, y), and resets the background color.

void check_image (

char	****imagename,***	/* name of the image file (not full path) */
Display	****display,***	/* X display */
Window	***window,***	/* X window */
int	***x,***	/* origin x in device coordinates */
int	***y,***	/* origin y in device coordinates */
int	***w,***	/* width in device coordinates */
int	***h,***	/* height in device coordinates */
char	****text)***	/* image description */

Determine the paths for reference and current images with respect to the default display connection, then call *check_image_core* to do image checking. Skip this operation if the environment variable *NOCHECKPIX* is set. Find a reference image in REFIMAGE. If it exists, compare against it and take appropriate action. If it doesn't exist, find a known incorrect image in

CURIMAGE for comparison. If no image is found for comparison or all images compare with differences, save a copy in the current image directory. Print out messages reporting images matched or require manual inspection.

Image format is stored as:

<1 byte: flag image type, pseudocolor or truecolor>
<2 bytes: number of bytes of ascii text to follow>
<n bytes: ascii text>
<2 bytes: width of image>
<2 bytes: height of image>
<2 bytes: number of colormap data entries>
<n bytes: colormap data of the form
(pixel[1 byte],red[1 byte],green[1 byte],blue[1 byte]) quadruples >
<4 bytes: number of bytes of image data>
<n bytes: image data of the form (number[1 byte],pixel[1 byte]) pairs >

Lighting Functions

vector_normalize (

Xgl_pt_f3d	****v)***

Normalize the vector v in place.

apply_indexed_lighting (

Xgl_3d_ctx	***ctx***,	/* IN: Current context */
Xgl_pt_f3d	****Op***,	/* IN: Polygon vertex */
Xgl_pt_f3d	****normal***,	/* IN: Normal for that vertex */
Xgl_color	****Od_hat***,	/* IN: Color for that vertex */
Xgl_color	****color***)	/* OUT: Color adjusted for lighting */

Calculate the lit color of a vertex based on the current light status, and the original color of the object. Use the index color model.

apply_rgb_lighting (

Xgl_3d_ctx	***ctx,***	/* IN: Current context */
Xgl_pt_f3d	****Op,***	/* IN: Polygon vertex */
Xgl_pt_f3d	****normal,***	/* IN: Normal for that vertex */

```
Xgl_color        *Od,       /* IN: Color for that vertex */
Xgl_color        *color)    /* OUT: Color adjusted for
                               lighting */
```

Calculate the lit color of a vertex based on the current light status, and the original color of the object. Use the RGB color model.

Line Functions

int dtp_isline (

```
Xgl_sys_state    sys_st,
Xgl_ctx          ctx,
Xgl_usgn32       num_pt_lists,
int              loe,           /* Edge or line ? */
Xgl_pt_list      * pl,          /* Geometry data */
int              (* failure) ())
```

Multipolyline information is given in device coordinates as `XGL_PT_F2D` or `XGL_PT_F3D`. *loe* == *DTP_LINE* means that the data is for a line. *loe* == *DTP_EDGE* means the data is for an edge. This is needed because at the lowest level the point checking routine will need to get the expected color from some context attribute. This could be `XGL_CTX_LINE_COLOR` or `XGL_CTX_EDGE_COLOR`, depending on whether it is being passed a point on a line or an edge. The same applies to other attributes. If the *pl* is NULL or *num_pt_lists* is 0, -1 is returned. If the line or edge is verified to be correct then a 1 is returned, otherwise a 0 is returned. This function tests a few points along the line. The first failure causes verification conditions to become less stringent for that particular point of failure, depending on the line or edge style. For example, if it fails at a point and the style is patterned, it may look in a larger region before finally returning a 0.

int is_line (

```
Xgl_sys_state    sys_st,
Xgl_obj          ctx,
int              x1,
int              y1,
int              x2,        /* Endpoints of the line */
int              y2,
Xgl_color        *color,    /* Line color expected */
```

Xgl_color_type	***color_type,***	
int	***delta,***	/* Not every point on the line is checked. Delta controls the separation between the points checked */
int	***strict_check)***	/* However if this is 1, every point on the line is checked */

Checks points along a solid line to see if an expected color is found at each point. The separation between the points can be controlled from coarse (large delta and strict_check == 0) to very fine (strict_check == 1).

is_shaded_line (

Xgl_sys_state	***sys_st,***
Xgl_obj	***ctx,***
int	***x1,***
int	***y1,***
int	***x2,***
int	***y2,***
Xgl_color	****c1,***
Xgl_color	****c2,***
Xgl_color_type	***color_type)***

Verifies a shaded line, checking all points along the line. *c1* and *c2* are the endpoint colors that are interpolated linearly, in the manner of XGL. The geometry (endpoints) data must be in device coordinates.

is_wide_line (

Xgl_sys_state	***sys_st,***	
Xgl_obj	***ctx,***	/* IN: current context */
int	***x1,***	/* IN: 1st endpoint */
int	***y1,***	
int	***x2,*** /* IN: 2nd endpoint */	
int	***y2,***	
int	***ew,***	/* IN: expected width */
Xgl_color_type	***color_type)***	/* IN: RGB or Index */

Tests for wide lines. At all points except for the endpoints and polyline joins: a vertical line running across an x-major line encounters width pixels. Similarly, a horizontal line running across a y-major line finds the same number. The function's task is to pick an appropriate number of interior points, and check either horizontally or vertically for any given line.

void pattern_line (

Xgl_sys_state	***sys_st,***	
Xgl_2d_ctx	***ctx2d,***	
Xgl_win_ras	***ras,***	
int	***x1,***	
int	***x2,***	
int	***y1,***	/* line should be between
int	***y2,***	(x1, y1) and (x2, y2) which should be in device coordinates */
Xgl_color line	***color,***	
int	***orientation)***	/* orientation can be horizontal (0) or vertical (1) */

Draws a patterned line and then tests it for correctness. The line can only be a horizontal or vertical line.

void solid_line (

Xgl_sys_state	***sys_st,***
Xgl_2d_ctx	***ctx2d,***
Xgl_win_ras	***ras,***
int	***x1,***
int	***x2,***
int	***y1,***
int	***y2,***
Xgl_color line	***color)***

Draws a solid line and then tests it using *is_line()*.

```
void  toggle_line_type (
        Xgl_sys_state           sys_st,
        Xgl_2d_ctx              ctx2d,
        Xgl_win_ras             ras,
        int                     x1,
        int                     x2,
        int                     pattern,
        int                     direction)
```

A wrapper for *solid_line()* or *pattern_line()*. Calls these functions several times with slightly different colors and varying geometry.

Marker Functions

```
int is_multimarker (
        Xgl_sys_state           sys_st,
        Xgl_ctx                 ctx,
        Xgl_pt_list             *pl,        /* Geometry data */
        Xgl_marker              marker,
        float                   size,       /* Marker scale size */
        Xgl_color               *color)     /* Expected color */
```

Verifies that the list of markers in the pl is indeed correct. Uses the `XGL_MARKER_DESCRIPTION` attribute to get the line description of the markers.

Nurbs Functions

```
int    is_nubs (
        Xgl_sys_state           sys_st,
        Xgl_object              ctx,           /*  IN: context */
        Xgl_nu_bspline_curve    *curve,        /*  IN: ptr to a nubs curve */
        float                   u,             /*  IN: parameter value for
                                                   the curve */
        Xgl_color               *color,        /*  IN: curve color */
        Xgl_pt_f3d              *pt,           /*  OUT: curve point's
                                                   coordinates */
        int                     (* failure) ()) /* Pointer to function for
```

handling special instances of failures. For example sometimes a test is known to fail at a certain (x, y) location and the failure is a permissible failure. This failure may be occurring because of poor test design or poor verification function. In such a case, the test can provide a pointer to a function which checks to see if the failure is one of the special failure instances. Then it may handle the failure in whatever way is appropriate. This can be NULL, and is in fact NULL for most tests. */

The argument *pt* is the pointer to a *Xgl_pt_f3d* point which, upon return, will contain the coordinates of the point on the curve corresponding to the parameter *u*. Only one point on the curve, corresponding to *u*, is checked.

Polygon Functions

```
int dtp_ispg (
        Xgl_sys_state        sys_st,
        Xgl_ctx              ctx,
        Xgl_facet_type       facet_type,
        Xgl_facet            * facet,
        Xgl_usgn32           num_pt_lists,
        Xgl_pt_list          * pl,
        int                  (* failure) ())
```

A simple polygon-checking function. This one does not do scan conversion (unlike *is_polygon()*). Instead it checks points along all the boundaries to see if the expected colors are present. The data must be in device coordinates. The arguments are similar to the `xgl_polygon()`. Generally, this verification function is used in tests involving transformations or clipping.

int dtp_isrect (

Xgl_ctx	***ctx,***
Xgl_rect_list	**** rect_list,***
int	***(* failure) ())***

Verifies rectangles. This is a wrapper to *dtp_ispg()* so the caveats mentioned for that function apply here as well.

int dtp_isqm (

Xgl_sys_state	***sys_st,***
Xgl_ctx	***ctx,***
Xgl_usgn32	***rows,***
Xg;_usgn32	***cols,***
Xgl_facet_list	**** facets,***
Xgl_pt_list	**** qmpl,***
int	***(* failure) ())***

Verifies quadmeshes. This is a wrapper to *dtp_ispg()*, so the caveats mentioned for that function apply here as well.

int dtp_ists (

Xgl_sys_state	***sys_st,***
Xgl_ctx	***ctx,***
Xgl_facet_list	**** facets,***
Xgl_pt_list	**** tspl,***
int	***(* failure) ())***

Verifies tristrips. This is a wrapper to *dtp_ispg()*, so the caveats mentioned for that function apply here as well.

is_polygon (

Xgl_sys_state	***sys_st,***	
Xgl_object	***ctx,***	
int	***nbnds,***	/* IN: number of boundaries */
Xgl_pt_list	***pts[ ],***	/* IN: array of edge lists */
Xgl_color_type	***type,***	/* IN: color model for interior */
Xgl_color	****color)***	/* IN: interior color */

Scan converts the polygon described by the arguments. Checks each point along the scan line but not each scan line. Verifies roughly 10% of the scan lines. This function verifies flat shaded polygons.

is_shaded_polygon (

Xgl_sys_state	***sys_st,***	
Xgl_object	***ctx,***	
int	***nbnds,***	/* IN: number of boundaries */
Xgl_pt_list	***pts[ ],***	/* IN: array of edge lists */
Xgl_color_type	***type)***	/* IN: color model for interior */

Similar to *is_polygon()* except that the polygon can be shaded. Interpolates along y and across each scan line.

is_hollow_polygon (

Xgl_sys_state	***sys_st,***
Xgl_object	***ctx,***
Xgl_pt_list	****pl,***
Xgl_pt_f3d	****interior,***
Xgl_color_type	***type,***
Xgl_color	****col,***
int	***strict_check)***

Verifies that a flat-shaded polygon is drawn with a hollow interior. is*_line()* is called for each edge, and a sample interior point is also examined. The semantics of *strict_check* are the same as for *is_line()*.

is_hollow_shaded_polygon (

Xgl_sys_state	***sys_st,***
Xgl_object	***ctx3d,***
Xgl_pt_list	****pl,***
Xgl_pt_f3d	****interior,***
Xgl_color_type	***type)***

Similar to *is_hollow_polygon()* except that it can be shaded.

is_polygon_edge (

Xgl_sys_state	***sys_st,***
Xgl_object	***ctx,***
Xgl_pt_list	**** pl,***
Xgl_color_type	***type,***
Xgl_color	**** col)***

Checks to see if a polygon has the correct edge. This is a wrapper to *is_hollow_polygon()* with NULL for interior argument.

is_polygon_set_lskip (

int	***skip)***

is_polygon() skips the number of pixels at the start of each scan line when verifying.

is_polygon_set_rskip (

int	***skip)***

is_polygon() skips the number of pixels at the end of each scan line when verifying.

Transform Functions

int dtp_eqmtx (

Dtp_mtx	****A,***	/* matrix to be compared */
Dtp_mtx	****B)***	/* matrix to be compared */

Compares two matrixes to see if they are equal. Input matrices must have the same data type and dimension. Returns 1 if they are equal and 0 otherwise.

int dtp_eqtrans (

Xgl_trans	***A,***	/* Transform to be compared */
Xgl_trans	***B)***	/* Transform to be compared */

Compares two transforms to see if they are equal. Input transforms must be of the same data type and dimension. Returns 1 if they are equal and 0 otherwise.

int dtp_idmtx (

Xgl_trans	***trans***)	/* input transform */

Checks to see if the matrix is an identity matrix. Returns 1 if A is identity matrix and 0 otherwise.

int dtp_idtrans (

Xgl_trans	***trans)***	/* input transform */

Checks to see if the transform is an identity transform. Returns 1 if trans is identity and 0 otherwise.

dtp_matrix_multiply (

Dtp_mtx	****M,***	/* matrix in which the result of multiplying L and R (L * R) will be stored */
Dtp_mtx	****L,***	/* left matrix in the multiplication */
Dtp_mtx	****R)***	/* right matrix in the multiplication */

Multiplies matrix pointed to by L with matrix pointed to by R and returns the result in matrix pointed to by M. Space for result matrix (M) must have been allocated by the caller. L, R, and M must be of the same dimension and have the same data type.

```
dtp_transform_multiply (
        Xgl_trans        D,       /* destination transform in which
                                     the result of multiplication
                                     (L* R) will be stored */
        Xgl_trans        L,       /* left transform in the
                                     multiplication */
        Xgl_trans        R)       /* right transform in the
                                     multiplication */
```

Multiplies transform L with transform R and stores the result (L * R) into D. L, R, and D must be of the same dimension and data type.

```
int   dtp_ispt (
        Xgl_sys_state    sys_st,
        Xgl_ctx          ctx,
        float            x,       /* Coordinates of input point in
        float            y,          Model Coordinate Space */
        float            z,
        Dtp_pt_type      pt_type,
        int              (*failure) ())
```

Checks to see if point (x, y, z) is of the right color. The function internally transforms (x, y, z) to (x', y') in Device Coordinate Space. Then it checks a small region of pixels around (x', y'). The color(s) it looks for depend on the point type, that is, whether the point is on a line, an edge, a patterned line, text, interior of a surface, and so on. Returns 1 if the right color is read back in the region around (x', y') and 0 if not. It returns -1 if some lower-level pixel readback function returns an error like attempting to read back pixels from an obscured region of a window.

```
int   dtp_isptc (
        Xgl_sys_state    sys_st,
        Xgl_ctx          ctx,
        float            x,       /* Coordinates of input point in
        float            y,          Model Coordinate Space */
        float            z,
        Xgl_color        *color,  /* Check for this color at (x, y, z)
*/
        int (            *failure) ())
```

Same as *dtp_ispt()* except that it checks for the color given as input.

int dtp_modelx (

Xgl_ctx	***ctx,***	
Xgl_pt	****ptp)***	/* Pointer to input point */

Transforms the input point using the local modeling transform (XGL_CTX_LOCAL_MODEL_TRANS) and the global modeling transform (XGL_CTX_GLOBAL_MODEL_TRANS) in that order. The transformation is done in place in the input point. Returns 0 if successful and -1 if either of the input arguments is NULL.

int dtp_viewx (

Xgl_ctx	***ctx,***	
Xgl_pt	***ptp)***	/* Pointer to input point */

Same as *dtp_modelx()* except that the view transformation (XGL_CTX_VIEW_TRANS) is used instead of the modelling transforms.

void dtp_vuclipb (

Xgl_ctx	***ctx,***	
Xgl_bbox	****bounds)***	/* Output argument in which view clip bounds will be returned */

Returns the view clip bounds (XGL_CTX_VIEW_CLIP_BOUNDS) in the argument bounds. The caller must have allocated space for bounds. The dimension of the bounds is also set in *bounds>bbox_type* as XGL_BBOX_F2D or XGL_BBOX_F3D.

void dtp_vpclipb (

Xgl_ctx	***ctx,***	
Xgl_bbox	****bounds)***	/* Output argument containing view clip bounds */

Returns the view clip bounds (XGL_CTX_DC_VIEWPORT) in the argument bounds. The caller must have allocated space for bounds. The dimension of the bounds is also set in *bounds->bbox_type* as XGL_BBOX_F2D or XGL_BBOX_F3D.

int dtp_vuclipd (

Xgl_ctx	***ctx,***	
float	***x,***	/* Coordinates of input point in
float	***y,***	Device Coordinate Space */
float	***z)***	

Returns 1 if (x, y, z) is clipped by the view clip bounds, 0 if it is within the bounds, and -1 if *ctx* is NULL. The results of the clip checking also depend on the value of `XGL_CTX_CLIP_PLANES`, which determines the clip planes that are in effect. This function assumes that the VDC transform is identity.

int dtp_vpclipd (

Xgl_ctx	***ctx,***	
float	***x,***	/* Coordinates of input point in
float	***y,***	Device Coordinate Space */
float	***z)***	

Returns 1 if (x, y, z) is clipped by the viewport clip bounds, 0 if it is within the bounds, and -1 if *ctx* is NULL.

int dtp_mclipd (

Xgl_ctx	***ctx,***	
float	***x,***	/* Coordinates of input point in
float	***y,***	Model Coordinate Space */
float	***z)***	

Returns 1 if point is model clipped, that is, (x,y,z) is outside the model clip volume, and 0 if the point is not clipped. Returns -1 if *ctx* is NULL or a 2D context (since model clipping is only defined for 3D contexts).

int dtp_vdcx (

Xgl_ctx	***ctx,***	
Xgl_pt	****ptp)***	/* Input point */

Applies the VDC transform to the input point in place (that is, the transformed point is returned in *ptp*). Assumes that the view transformation is identity and works only for the default value of `XGL_CTX_VDC_ORIENTATION`. The dimension of the point and the context must match and the point type can only be `XGL_PT_F2D` or `XGL_PT_F3D`. Returns 0 if successful and -1 otherwise.

int dtp_devb (

Xgl_ctx	***ctx,***	
Xgl_usgn32	****w,***	/* Value of XGL_RAS_WIDTH */
Xgl_usgn32	****h)***	/* Value of XGL_RAS_HEIGHT */

Returns the raster width and height in w and h respectively. Space for w and h must have been allocated by the caller. Returns 0 if successful and -1 if there is no device associated with *ctx*.

int dtp_mc2dc (

Xgl_ctx	***ctx,***
Xgl_usgn32	***num_pt_lists,***
Xgl_pt_list	****pl)***

Transforms the points in the point lists in the array pointed to by *pl*. The transformation includes model, view, and VDC transformation. Returns 0 if successful and -1 if invalid point types are given as input. The only valid point types are XGL_PT_F2D and XGL_PT_F3D.

int dtp_plcopy (

Xgl_usgn32	***num_pt_lists,***	/* Number of point lists to be copied */
Xgl_pt_list	***pl[],***	/* Array of point lists to be copied */
Xgl_pt_list	*****outpl)***	/* A pointer to an array of pointers which contain pointers to copies of the elements of *pl* */

Copies the input point list array into *outpl*. Thus *pl[i]* is copied into (**outpl*)[*i*]. The function allocates the space needed for *outpl*. Returns 0 if successful and -1 if the point type of the point lists is other than XGL_PT_F2D or XGL_PT_F3D.

Antialiasing Test Descriptions 5

This chapter describes the Antialiasing test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section, “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ aa_line

Test Types:	RGB, CM
Description:	Sets the line style to antialiased and filter width. Draws 3D lines in various angles.
Attributes Tested:	`XGL_CTX_LINE_AA_BLEND_EQ` `XGL_CTX_LINE_AA_FILTER_WIDTH`
Operators Tested:	`xgl_object_set` `xgl_multipolyline`
Output:	Antialiased lines drawn like the spokes of a bicycle wheel.

aa_line_alt_patterned

Test Types:	RGB, CM
Description:	Sets the line style to alt-patterned, antialiased and filter width. Draws 3D lines in various angles for six different line patterns.

Attributes Tested:	`XGL_CTX_LINE_STYLE`
	`XGL_CTX_LINE_PATTERN`
	`XGL_CTX_LINE_AA_BLEND_EQ`
	`XGL_CTX_LINE_AA_FILTER_WIDTH`

Operators Tested:	`xgl_object_set` `xgl_multipolyline`
Output:	Antialiased alt-patterned lines drawn for six different patterns. Lines are drawn like the spokes of a bicycle wheel for each line pattern.

aa_line_alt_patterned_interp

Test Types:	RGB, CM
Description:	Sets the line style to alt-patterned, line color interpolation, antialiased, and filter width. Draws 3D lines in various angles for six different line patterns.

Attributes Tested:	`XGL_3D_CTX_LINE_COLOR_INTERP`
	`XGL_CTX_LINE_STYLE`
	`XGL_CTX_LINE_AA_BLEND_EQ`
	`XGL_CTX_LINE_AA_FILTER_WIDTH`

Operators Tested:	`xgl_object_set` `xgl_multipolyline`
Output:	Antialiased alt-patterned color interpolated lines drawn for six different patterns. Lines are drawn like the spokes of a bicycle wheel for each line pattern.

aa_line_blend_draw_mode

Test Types:	RGB, CM
Description:	Sets HLHSR, line color, and surface color. Sets non-blending, draws a polygon and two antialiased lines, then draws two non-antialiased lines. Sets blending to blended and repeats drawing the polygon and lines. Sets blending to all and repeats drawing.

Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_3D_CTX_BLEND_DRAW_MODE XGL_CTX_LINE_AA_BLEND_EQ XGL_CTX_LINE_AA_FILTER_WIDTH
Operators Tested:	xgl_object_set xgl_multipolyline xgl_polygon
Output:	A polygon and two non-antialiased lines are drawn on the top left corner, and two antialised lines on the top right corner. These are repeated on the bottom left corner. The bottom right corner is blank.

▼ aa_line_blend_eq

Test Types:	RGB, CM
Description:	For-loop the three line antialiasing blending cases: *arbitrary background*, *constant background*, and *add to background*. Draws antialiased lines in various angles. Draws a polygon. Draws antialiased and non-antialiased lines across the polygon for each blending case.
Attributes Tested:	XGL_CTX_LINE_COLOR XGL_CTX_SURF_FRONT_COLOR XGL_CTX_LINE_AA_BLEND_EQ, XGL_CTX_LINE_AA_FILTER_WIDTH
Operators Tested:	xgl_object_set xgl_multipolyline xgl_polygon
Output:	Three sets of spokes of bicycle wheels drawn with antialiased lines in the top half of the canvas and three sets of polygons, antialiased and non-antialiased lines across them in the bottom half.

▼ aa_line_interp

Test Types:	RGB, CM
Description:	Sets the vertex colors. Sets color interpolation and antialiasing blending, then draws lines in various angles.
Attributes Tested:	XGL_3D_CTX_LINE_COLOR_INTERP XGL_CTX_LINE_AA_BLEND_EQ XGL_CTX_LINE_AA_FILTER_WIDTH

Operators Tested:	`xgl_multipolyline`
Output:	Antialiased color interpolated lines drawn like the spokes of a bicycle wheel; red at the center and blue at the rim.

aa_line_patterned

Test Types:	RGB, CM
Description:	Sets the line style to patterned, antialiasing blending, and filter width. Draws 3D lines in various angles for six different line patterns.

Attributes Tested:	`XGL_CTX_LINE_COLOR`
	`XGL_CTX_LINE_STYLE`
	`XGL_CTX_LINE_AA_BLEND_EQ`
	`XGL_CTX_LINE_AA_FILTER_WIDTH`

Operators Tested:	`xgl_object_set` `xgl_multipolyline`
Output:	Antialiased patterned lines drawn for six different patterns. Lines are drawn like the spokes of a bicycle wheel for each line pattern.

aa_line_patterned_interp

Test Types:	RGB, CM
Description:	Sets the vertex colors. Sets color interpolation. Sets the line style to patterned, antialiasing blending, and filter width. Draws 3D lines in various angles for six different line patterns.

Attributes Tested:	`XGL_3D_CTX_LINE_COLOR_INTERP`
	`XGL_CTX_LINE_STYLE`
	`XGL_CTX_LINE_PATTERN`
	`XGL_CTX_LINE_AA_BLEND_EQ,`
	`XGL_CTX_LINE_AA_FILTER_WIDTH`

Operators Tested:	`xgl_object_set` `xgl_multipolyline`
Output:	Antialiased patterned color interpolated lines drawn for six different patterns. Lines are drawn like the spokes of a bicycle wheel for each line pattern.

▼ aa_marker

Test Types:	RGB, CM
Description:	Sets the antialiasing blending and filter width. Sets point colors. For loop three different marker scale factors, and eight markers to draw markers.
Attributes Tested:	`XGL_CTX_MARKER_AA_BLEND_EQ`, `XGL_CTX_MARKER_AA_FILTER_WIDTH`
Operators Tested:	`xgl_object_set` `xgl_multimarker`
Output:	Draws two sets (red and green) of eight antialised markers in different scales.

▼ aa_mspg_edge

Test Types:	RGB, CM
Description:	Sets surface transparency method, and blending equation. Sets blending mode to not blend. Draws opaque simple polygons and then draws transparent simple polygons. Sets blending mode to blend and draws opaque and transparent simple polygons. Repeats the above with edges on.
Attributes Tested:	`XGL_3D_CTX_SURF_TRANSP_METHOD` `XGL_3D_CTX_SURF_TRANSP_BLEND_EQ`, `XGL_3D_CTX_BLEND_DRAW_MODE` `XGL_3D_CTX_SURF_FRONT_TRANSP` `XGL_CTX_SURF_EDGE_FLAG` `XGL_CTX_EDGE_COLOR` `XGL_CTX_EDGE_AA_BLEND_EQ` `XGL_CTX_EDGE_AA_FILTER_WIDTH`
Operators Tested:	`xgl_object_set` `xgl_multi_simple_polygon`
Output:	Two solid blue squares in the top left portion of the canvas, two dark green blended squares in the top right, two solid blue squares in the bottom left, and two hollow and two dark green blended squares with edges on in the bottom right are drawn. Note that no edges are drawn for the squares in the bottom left.

aa_mspg_hollow

Test Types:	RGB, CM
Description:	Sets HLHSR. Sets surface fill to hollow. Sets surface antialiasing blending equation and filter width. Sets blending mode to none and draws two sets of multi-simple polygons. Sets blending mode to blended and draws polygons. Sets blending mode to blend all and draws polygons.

Attributes Tested:	`XGL_CTX_SURF_FRONT_FILL_STYLE`
	`XGL_CTX_SURF_AA_BLEND_EQ`
	`XGL_CTX_SURF_AA_FILTER_WIDTH`
	`XGL_3D_CTX_BLEND_DRAW_MODE`

Operators Tested:	`xgl_object_set` `xgl_multi_simple_polygon`
Output:	Two sets of blue and green antialiased hollow squares are drawn in the top left and bottom portions of the canvas. Nothing is drawn in the top right. The bottom right is blank.

aa_stroketext

Test Types:	RGB, CM
Description:	Sets the character height and character color. Sets the stroke text antialiasing blending equation and filter width. Draws stroke text.
Attributes Tested:	`XGL_CTX_STEXT_AA_BLEND_EQ` `XGL_CTX_STEXT_AA_FILTER_WIDTH`
Operators Tested:	`xgl_object_set` `xgl_stroke_text_3d`
Output:	Antialiased stroke text *ABCDEFGHIJKLMNOPQRSTUVWXYZ \ nabcdefghijklm nopqrstuvwxyz \ n0123456789* are drawn on the canvas.

Arc Test Descriptions 6

This chapter describes the Arc test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section, “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ arc0

Test Types:	INDEX, SM
Description:	Checks five points on an indexed 2D arc
Attributes Tested:	See Table 6-1, Column A at the end of this chapter.
Operators Tested:	`xgl_object_set` `xgl_multiarc`
Output:	Indexed 2D quarter circle.

▼ arc1

Test Types:	INDEX, SM
Description:	Checks five points on four indexed 2D arcs
Attributes Tested:	XGL_ARC_CHORD and Table 6-1, Column A at the end of this chapter
Operators Tested:	xgl_object_set xgl_multiarc
Output:	Four indexed 2D arcs. Arcs are quarter, half, three-quarter, and full circle.

▼ arc2

Test Types:	INDEX, SM
Description:	Checks for the presence of edge color at five points for each of four indexed 2D arcs
Attributes Tested:	XGL_ARC_CHORD XGL_CTX_EDGE_COLOR and Table 6-1, Column A at the end of this chapter
Operators Tested:	xgl_object_set xgl_multiarc
Output:	Several indexed 2D arcs with edges both on and off. Arcs are quarter, half, three-quarter, and full circle.

▼ arc3

Test Types:	INDEX, SM
Description:	Checks for the correct pattern and the presence of edge colors in several indexed 2D arcs with various combinations of edges and patterns.
Attributes Tested:	See Table 6-3, Column C at the end of this chapter.
Operators Tested:	xgl_object_set xgl_multiarc xgl_object_get
Output:	Several indexed 2D arcs with edges both on and off, and with three different patterns. Arcs are half a full circle.

arc4

Test Types:	INDEX, SM
Description:	Checks a few points of each of several arcs for the correct arc fill style
Attributes Tested:	See Table 6-2, Column A at the end of this chapter.
Operators Tested:	xgl_object_set xgl_multiarc xgl_object_get
Output:	An identical arc rendered three times in the lower right portion of the window raster, twice with one-quarter arc representing the fourth quadrant of a circle, and a final time representing the lower half of a circle. Arc fill styles are sector (solid) for the first arc and open (edges only) for the last two arcs.

arc5

Test Types:	INDEX, SM
Description:	Checks 17 points on each of several arcs for correct placement
Attributes Tested:	`XGL_CTX_SURF_EDGE_FLAG` and Table 6-1, Column B at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_multiarc`
	`xgl_object_get`

Output:	Several indexed 2D arcs with different data types (`I2D,F2D,B2D`), different sizes (1/4, 1/2, 3/4, full), and all sector (solid) fill style

arc6

Test Types:	RGB, SM
Description:	Checks five points on each of several different colored arcs. The colors are all colors in the color cube for 8-bit raster, and 256 random colors for other rasters.

Attributes Tested:	`XGL_ARC_CHORD`
	`XGL_CMAP_COLOR_CUBE_SIZE`
	`XGL_RAS_DEPTH`
	and Table 6-1, Column A at the end of this chapter

Operators Tested:	xgl_object_set xgl_multiarc xgl_object_get
Output:	Several quarter-circle RGB 2D arcs of various colors

▼ arc7

Test Types:	RGB, SM
Description:	RGB version of *arc1*
Attributes Tested:	XGL_ARC_CHORD and Table 6-1, Column A at the end of this chapter
Operators Tested:	xgl_object_set xgl_multiarc
Output:	Four RGB 2D arcs. Arcs are one-quarter, one-half, and three-quarter circle.

▼ arc8

Test Types:	RGB, SM
Description:	RGB version of *arc2*
Attributes Tested:	XGL_ARC_CHORD XGL_CTX_EDGE_COLOR and Table 6-1, Column A at the end of this chapter
Operators Tested:	xgl_object_set xgl_multiarc
Output:	Several RGB 2D arcs with edges both on and off. Arcs are one-quarter, one-half, and three-quarter circle.

▼ arc9

Test Types:	RGB, SM
Description:	RGB version of *arc3*
Attributes Tested:	See Table 6-3, Column C at the end of this chapter.
Operators Tested:	xgl_object_set xgl_multiarc xgl_object_get
Output:	Several RGB 2D arcs with edges both on and off, and with three different patterns. Arcs are half circle.

arc10

Test Types: RGB, SM
Description: Like *arc6*, but checks for a 9x9 solid square inside the arc instead of scattered points
Attributes Tested: XGL_ARC_SECTOR
XGL_CMAP_COLOR_CUBE_SIZE
XGL_RAS_DEPTH
and Table 6-1, Column A at the end of this chapter
Operators Tested: xgl_object_set
xgl_multiarc
xgl_object_get
Output: Several quarter-circle RGB 2D arcs of various colors

arc11

Test Types: RGB, SM
Description: RGB version of *arc4*
Attributes Tested: XGL_CTX_LINE_COLOR
and Table 6-2, Column A at the end of this chapter
Operators Tested: xgl_object_set
xgl_multiarc
xgl_object_get
Output: An identical arc rendered three times in the lower right portion of the window raster, twice with one-quarter arc representing the fourth quadrant of a circle, and a final time representing the lower half of a circle. Arc fill styles are sector (solid) for the first arc and open (edges only) for the last two arcs.

arc12

Test Types: RGB, SM
Description: RGB version of *arc5*
Attributes Tested: XGL_CTX_SURF_EDGE_FLAG
and Table 6-1, Column B at the end of this chapter
Operators Tested: xgl_object_set
xgl_multiarc
xgl_object_get

Output: Several RGB 2D arcs with different data types (I2D,F2D,B2D), different sizes (one quarter, one half, three quarter, full), and all sector (solid) fill style

▼ arc13

Test Types: INDEX, SM
Description: 3D version of *arc0*
Attributes Tested: XGL_ARC_CHORD
and Table 6-1, Column A at the end of this chapter
Operators Tested: xgl_object_set
xgl_multiarc
Output: Indexed 3D quarter circle

▼ arc14

Test Types: INDEX, SM
Description: 3D version of *arc1*
Attributes Tested: XGL_ARC_CHORD
and Table 6-1, Column A at the end of this chapter
Operators Tested: xgl_object_set
xgl_multiarc
Output: Four indexed 3D arcs. Arcs are one-quarter, one-half, and three-quarter circle.

▼ arc15

Test Types: INDEX, SM
Description: 3D version of *arc2*
Attributes Tested: XGL_ARC_CHORD
XGL_CTX_EDGE_COLOR
and Table 6-1, Column A at the end of this chapter
Operators Tested: xgl_object_set
xgl_multiarc
Output: Several indexed 3D arcs with edges both on and off. Arcs are one-quarter, one-half, and three-quarter circle.

arc16

Test Types: INDEX, SM
Description: 3D version of *arc3*
Attributes Tested: See Table 6-3, Column C at the end of this chapter.
Operators Tested: `xgl_object_set`
`xgl_multiarc`
`xgl_object_get`
Output: Several indexed 3D arcs with edges both on and off, and with three different patterns. Arcs are half circle.

arc17

Test Types: INDEX, SM
Description: 3D version of *arc4*
Attributes Tested: See Table 6-2, Column A at the end of this chapter.
Operators Tested: `xgl_object_set`
`xgl_multiarc`
`xgl_object_get`
Output: An identical arc rendered three times in the lower right portion of the window raster, twice with one-quarter arc representing the fourth quadrant of a circle, and a final time representing the lower half of a circle. Arc fill styles are sector (solid) for the first arc and open (edges only) for the last two arcs.

arc18

Test Types: RGB, SM
Description: 3D version of *arc6*
Attributes Tested: `XGL_ARC_CHORD`
`XGL_RAS_DEPTH`
and Table 6-1, Column A at the end of this chapter
Operators Tested: `xgl_object_set`
`xgl_multiarc`
`xgl_object_get`
Output: Several quarter-circle RGB 3D arcs of various colors

arc19

Test Types:	RGB, SM
Description:	3D version of *arc7*
Attributes Tested:	XGL_ARC_CHORD and Table 6-1, Column A at the end of this chapter
Operators Tested:	xgl_object_set xgl_multiarc
Output:	Four RGB 3D arcs. Arcs are one-quarter, one-half, and three-quarter circle.

arc20

Test Types:	RGB, SM
Description:	3D version of *arc8*
Attributes Tested:	XGL_ARC_CHORD XGL_CTX_EDGE_COLOR and Table 6-1, Column A at the end of this chapter
Operators Tested:	xgl_object_set xgl_multiarc
Output:	Several RGB 3D arcs with edges both on and off. Arcs are one-quarter, one-half, and three-quarter circle.

arc21

Test Types:	RGB, SM
Description:	3D version of *arc9*
Attributes Tested:	See Table 6-3, Column C at the end of this chapter.
Operators Tested:	xgl_object_set xgl_multiarc xgl_object_get
Output:	Several RGB 2D arcs with edges both on and off, and with three different patterns. Arcs are half circle.

arc22

Test Types:	RGB, SM
Description:	3D version of *arc10*

Attributes Tested:	`XGL_ARC_SECTOR`
	`XGL_RAS_DEPTH`
	and Table 6-1, Column A at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_multiarc`
	`xgl_object_get`

Output:	Several quarter-circle RGB 2D arcs of various colors

arc23

Test Types:	RGB, SM
Description:	3D version of *arc11*
Attributes Tested:	`XGL_CTX_LINE_COLOR` and Table 6-2, Column A at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_multiarc`
	`xgl_object_get`

Output:	An identical arc rendered three times in the lower right portion of the window raster, twice with one-quarter arc representing the fourth quadrant of a circle, and a final time representing the lower half of a circle. Arc fill styles are sector (solid) for the first arc and open (edges only) for the last two arcs.

arc24

Test Types:	RGB, SM
Description:	Checks 17 points on each of three arcs drawn simultaneously with nonzero starting angles
Attributes Tested:	See Table 6-1, Column B at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_multiarc`
	`xgl_object_get`

Output:	Three RGB 2D arcs with nonzero starting angle drawn simultaneously.

arc25

Test Types: RGB, SM
Description: Tests the three face-culling modes by drawing both front and back facing arcs and checking for their presence
Attributes Tested: XGL_3D_CTX_SURF_FACE_CULL
XGL_CULL_BACK
XGL_CULL_FRONT
and Table 6-2, Column B at the end of this chapter
Operators Tested: xgl_object_set
xgl_multiarc
xgl_object_get
Output: Several front-facing and back-facing arcs using the three face-culling modes. Depending on the mode, some arcs may not appear.

arc26

Test Types: RGB, SM
Description: Tests the face-distinguish and normal-flip attributes by drawing arcs using all four combinations of the above two attributes
Attributes Tested: XGL_3D_CTX_SURF_NORMAL_FLIP
and Table 6-2, Column B at the end of this chapter
Operators Tested: xgl_object_set
xgl_multiarc
xgl_object_get
Output: Several arcs with various face distinguish and normal flip attributes

arc27

Test Types: RGB, SM
Description: Draws arcs with different edge styles and checks various areas of the arcs to make sure they're drawn correctly
Attributes Tested: XGL_CTX_BACKGROUND_COLOR
XGL_CTX_EDGE_ALT_COLOR
XGL_CTX_EDGE_COLOR
XGL_CTX_EDGE_PATTERN
XGL_CTX_EDGE_STYLE

	XGL_CTX_EDGE_WIDTH_SCALE_FACTOR XGL_CTX_NURBS_CURVE_APPROX XGL_CTX_NURBS_CURVE_APPROX_VAL XGL_CTX_SURF_EDGE_FLAG XGL_CTX_SURF_FRONT_COLOR XGL_CURVE_METRIC_VDC XGL_LINE_ALT_PATTERNED
Operators Tested:	xgl_object_set xgl_multiarc xgl_object_get
Output:	Arc with edge flag off, arc with wide edge, and arc with thin alt-patterned edge

▼ arc28

Test Types:	RGB, SM
Description:	Arc HLHSR test: Draws two otherwise identical arcs of usually different depths and checks that only the one in the front shows up; tries many cases of different z values
Attributes Tested:	See Table 6-2, Column C at the end of this chapter.
Operators Tested:	xgl_object_set xgl_multiarc
Output:	A succession of arcs. If successful, only one arc displays at a time.

▼ arc29

Test Types:	RGB, SM
Description:	Arc HLHSR test: Sets Z-buffer to a certain value and draws an arc of a different z value and checks for the presence of the arc; repeats with different z values for the Z-buffer and arc
Attributes Tested:	XGL_3D_CTX_HLHSR_DATA XGL_CTX_BACKGROUND_COLOR and Table 6-2, Column C at the end of this chapter
Operators Tested:	xgl_object_set xgl_multiarc
Output:	A succession of arcs. If successful, each arc displays.

arc30

Test Types: RGB, SM
Description: Checks various areas on each of several RGB 2D arcs drawn with hollow and empty fill styles and wide and patterned edges
Attributes Tested: XGL_CTX_BACKGROUND_COLOR
XGL_CTX_EDGE_COLOR
XGL_SURF_FILL_EMPTY
XGL_SURF_FILL_HOLLOW
and Table 6-1, Column A at the end of this chapter
Operators Tested: xgl_object_set
xgl_multiarc
xgl_object_get
Output: Several RGB 2D quarter-circle arcs with hollow and empty fill styles

arc31

Test Types: RGB, SM
Description: RGB arc linear depth-cueing: Varies depth-cueing color, arc color, arc depth, and arc direction. Checks various points on the arcs for correct color.
Attributes Tested: XGL_3D_CTX_DEPTH_CUE_COLOR
and Table 6-3, Column B at the end of this chapter
Operators Tested: xgl_object_set
xgl_multiarc
xgl_object_get
Output: Several arcs with varying depth-cue attribute values and colors with arcs starting from 45 degrees and ending at 360 degrees

arc32

Test Types: RGB, SM
Description: Opaque version of *arc2*
Attributes Tested: XGL_SURF_FILL_OPAQUE_STIPPLE
XGL_SURF_FILL_SOLID
and Table 6-3, Column C at the end of this chapter

Operators Tested: xgl_object_set
xgl_multiarc
xgl_object_get

Output: Several RGB 3D half-circle arcs with edges both on and off, and with three different stipple patterns

▼ arc33

Test Types: RGB, SM

Description: Like *arc21*, but uses back facets instead of front facets

Attributes Tested: XGL_3D_CTX_SURF_BACK_FPAT
XGL_ARC_CHORD
XGL_CTX_ARC_FILL_STYLE
XGL_SURF_FILL_STIPPLE
and Table 6-3, Column A at the end of this chapter

Operators Tested: xgl_object_set
xgl_multiarc
xgl_object_get

Output: Several RGB 2D half-circle arcs with edges both on and off, and with three different patterns

▼ arc34

Test Types: RGB, SM

Description: 3D version of *arc30*

Attributes Tested: XGL_CTX_BACKGROUND_COLOR
XGL_CTX_EDGE_COLOR
XGL_SURF_FILL_EMPTY
XGL_SURF_FILL_HOLLOW
and Table 6-1, Column A at the end of this chapter

Operators Tested: xgl_object_set
xgl_multiarc
xgl_object_get

Output: Several RGB 3D 90-degree arcs with hollow and empty fill styles

arc35

Test Types:	RGB, SM
Description:	Like *arc34*, but uses back facets instead of front facets

Attributes Tested:	`XGL_SURF_FILL_EMPTY`
	`XGL_SURF_FILL_HOLLOW`
	and Table 6-3, Column A at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_multiarc`
	`xgl_object_get`

Output:	Several RGB 3D 90-degree arcs with hollow and empty fill styles

arc36

Test Types:	RGB, SM
Description:	Opaque version of *arc33*

Attributes Tested:	`XGL_ARC_CHORD`
	`XGL_SURF_FILL_SOLID`
	`XGL_SURF_FILL_OPAQUE_STIPPLE`
	`XGL_3D_CTX_SURF_BACK_FPAT`
	and Table 6-3, Column A at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_multiarc`
	`xgl_object_get`

Output:	Several RGB 2D half-circle arcs with edges both on and off, and with three different patterns

arc37

Test Types:	INDEX, SM
Description:	Indexed linear depth-cued arcs
Attributes Tested:	See Table 6-3, Column B at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_multiarc`
	`xgl_object_get`

Output:	Several indexed 3D arcs starting from 45 degrees and ending at 360 degrees

▼ arc38

Test Types: RGB, SM
Description: Checks various areas of each of several open 2D RGB arcs
Attributes Tested: XGL_ARC_OPEN
XGL_CTX_ARC_FILL_STYLE
XGL_CTX_BACKGROUND_COLOR
XGL_CTX_LINE_ALT_COLOR
XGL_CTX_LINE_COLOR
XGL_CTX_LINE_PATTERN
XGL_CTX_LINE_STYLE
XGL_CTX_LINE_WIDTH_SCALE_FACTOR
XGL_CTX_NURBS_CURVE_APPROX
XGL_CTX_NURBS_CURVE_APPROX_VAL
XGL_CURVE_METRIC_VDC
XGL_LINE_ALT_PATTERNED
XGL_LINE_PATTERNED
Operators Tested: xgl_object_set
xgl_multiarc
xgl_object_get
Output: Several open 2D half-circle RGB arcs. Arcs are wide arc, patterned arc, and alt-patterned arc.

▼ arc39

Test Types: RGB, SM
Description: Checks the color of 400 points in or near each of several arcs with 8- or 24-bit pattern fill style
Attributes Tested: See Table 6-1, Column C at the end of this chapter.
Operators Tested: xgl_object_create
xgl_object_get
xgl_object_set
xgl_multiarc
xgl_context_get_pixel
xgl_context_set_pixel
Output: Several quarter-circle arcs with different pattern fill styles

arc40

Test Types:	RGB, SM
Description:	3D version of *arc39*
Attributes Tested:	XGL_CTX_NURBS_CURVE_APPROX XGL_CTX_NURBS_CURVE_APPROX_VAL XGL_CURVE_METRIC_WC and Table 6-1, Column C at the end of this chapter
Operators Tested:	xgl_object_create xgl_object_get xgl_object_set xgl_multiarc xgl_context_get_pixel xgl_context_set_pixel
Output:	Several quarter-circle arcs with different pattern fill styles

arc41

Test Types:	RGB, SM
Description:	Backface version of *arc40*
Attributes Tested:	XGL_3D_CTX_SURF_BACK_FILL_STYLE XGL_MEM_RAS XGL_3D_CTX_SURF_BACK_FPAT XGL_3D_CTX_SURF_FACE_DISTINGUISH XGL_CTX_BACKGROUND_COLOR XGL_CTX_NURBS_CURVE_APPROX XGL_CTX_NURBS_CURVE_APPROX_VAL XGL_CURVE_METRIC_WC XGL_RAS_DEPTH XGL_RAS_HEIGHT XGL_RAS_WIDTH XGL_SURF_FILL_PATTERN
Operators Tested:	xgl_object_create xgl_object_get xgl_object_set xgl_multiarc xgl_context_get_pixel xgl_context_set_pixel
Output:	Several quarter-circle arcs with different pattern fill styles

▼ arc_annot_af3d_chord

Test Types:	INDEX, SM
Description:	Checks outline of annotation *f3d* arc list composed of four arcs using chord fill style

Attributes Tested:	`XGL_CTX_ARC_FILL_STYLE`
	`XGL_ARC_CHORD`
	`XGL_CTX_NURBS_CURVE_APPROX_VAL`

Operators Tested:	`xgl_object_set` `xgl_multiarc`
Output:	Four arcs, two to a row, (1) arc starts at 60 and ends at 315 degrees, (2) arc starts at -225 and ends at 315 degrees, (3) arc starts at 45 and ends at -90 degrees and (4) arc starts at -90 and ends at 90 degrees.

▼ arc_annot_af3d_nonid_trans

Test Types:	INDEX, SM
Description:	Check outline of annotation `f3d` arc list composed of four open arcs after a translation leaving the expected arc at 100 coordinates down (default VDC Orientation: `XGL_Y_DOWN_Z_AWAY`) from the original expected y value.

Attributes Tested:	`XGL_CTX_ARC_FILL_STYLE`
	`XGL_ARC_CHORD`
	`XGL_CTX_NURBS_CURVE_APPROX_VAL`
	`XGL_ARC_OPEN`
	`XGL_CTX_GLOBAL_MODEL_TRANS`

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_transform_translate`
	`xgl_multiarc`
	`xgl_transform_identity`

Output:	Four arcs, two to a row, (1) arc starts at 60 and ends at 315 degrees, (2) arc starts at -225 and ends at 315 degrees, (3) arc starts at 45 and ends at -90 degrees, and (4) arc starts at -90 and ends at 90 degrees.

arc_annot_af3d_open

Test Types:	INDEX, SM
Description:	Checks outline of annotation *f3d* arc list composed of four open arcs

Attributes Tested:	`XGL_CTX_ARC_FILL_STYLE`
	`XGL_ARC_OPEN`
	`XGL_CTX_NURBS_CURVE_APPROX_VAL`

Operators Tested:	`xgl_object_set` `xgl_multiarc`
Output:	Four arcs, two to a row, (1) arc starts at 60 and ends at 315 degrees, (2) arc starts at -225 and ends at 315 degrees, (3) arc starts at 45 and ends at -90 degrees, and (4) arc starts at -90 and ends at 90 degrees.

arc_annot_af3d_sector

Test Types:	INDEX, SM
Description:	Checks outline of annotation *f3d* arc list composed of four arcs using sector fill style

Attributes Tested:	`XGL_CTX_ARC_FILL_STYLE`
	`XGL_ARC_SECTOR`
	`XGL_CTX_NURBS_CURVE_APPROX_VAL`

Operators Tested:	`xgl_object_set` `xgl_multiarc`
Output:	Four arcs, 2 to a row, (1) arc starts at 60 and ends at 315 degrees, (2) arc starts at -225 and ends at 315 degrees, (3) arc starts at 45 and ends at -90 degrees, and (4) arc starts at -90 and ends at 90 degrees.

Table 6-1 Arc Attributes Tested - Set 1

Column A	Column B	Column C
XGL_CTX_ARC_FILL_STYLE	XGL_CTX_BACKGROUND_COLOR	XGL_CTX_BACKGROUND_COLOR
XGL_CTX_NURBS_CURVE_APPROX	XGL_CTX_NURBS_CURVE_APPROX	XGL_CTX_SURF_FPAT
XGL_CTX_NURBS_CURVE_APPROX _VAL	XGL_CTX_NURBS_CURVE_APPROX _VAL	XGL_CTX_SURF_FRONT_FILL_ STYLE
XGL_CTX_SURF_EDGE_FLAG	XGL_CTX_SURF_FRONT_COLOR	XGL_MEM_RAS
XGL_CTX_SURF_FRONT_COLOR	XGL_CURVE_METRIC_VDC	XGL_RAS_DEPTH
XGL_CURVE_METRIC_VDC		XGL_RAS_HEIGHT
		XGL_RAS_WIDTH
		XGL_SURF_FILL_PATTERN

Table 6-2 Arc Attributes Tested - Set 2

Column A	Column B	Column C
XGL_ARC_OPEN	XGL_3D_CTX_SURF_BACK_COLOR	XGL_3D_CTX_HLHSR_MODE
XGL_ARC_SECTOR	XGL_3D_CTX_SURF_FACE_ DISTINGUISH	XGL_CTX_NEW_FRAME_ACTION
XGL_CTX_ARC_FILL_STYLE	XGL_CURVE_METRIC_VDC	XGL_CTX_NEW_FRAME_CLEAR
XGL_CTX_BACKGROUND_COLOR	XGL_ARC_SECTOR	XGL_CTX_NEW_FRAME_HLHSR_ ACTION
XGL_CTX_NURBS_CURVE_APPROX	XGL_CTX_ARC_FILL_STYLE	XGL_CTX_NURBS_CURVE_APPROX
XGL_CTX_NURBS_CURVE_APPROX _VAL	XGL_CTX_BACKGROUND_COLOR	XGL_CTX_NURBS_CURVE_APPROX _VAL
XGL_CTX_SURF_EDGE_FLAG	XGL_CTX_NURBS_CURVE_APPROX	XGL_CTX_SURF_FRONT_COLOR
XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_NURBS_CURVE_APPROX _VAL	XGL_CURVE_METRIC_VDC
XGL_CURVE_METRIC_VDC	XGL_CTX_SURF_FRONT_COLOR	XGL_HLHSR_Z_BUFFER

Table 6-3 Arc Attributes Tested- Set 3

Column A	**Column B**	**Column C**
XGL_3D_CTX_SURF_BACK_COLR	XGL_DEPTH_CUE_LINEAR	XGL_ARC_CHORD
XGL_3D_CTX_SURF_BACK_FILL_ STYLE	XGL_3D_CTX_DEPTH_CUE_MODE	XGL_CTX_ARC_FILL_STYLE
XGL_3D_CTX_SURF_FACE_ DISTINGUISH	XGL_CTX_NURBS_CURVE_APPROX	XGL_CTX_BACKGROUND_COLOR
XGL_CTX_BACKGROUND_COLOR	XGL_CTX_NURBS_CURVE_APPROX _VAL	XGL_CTX_EDGE_COLOR
XGL_CTX_EDGE_COLOR	XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_NURBS_CURVE_APPROX
XGL_CTX_NURBS_CURVE_APPROX	XGL_CTX_VDC_MAP	XGL_CTX_NURBS_CURVE_APPROX _VAL
XGL_CTX_NURBS_CURVE_APPROX _VAL	XGL_CTX_VDC_WINDOW	XGL_CTX_SURF_EDGE_FLAG
XGL_CTX_SURF_EDGE_FLAG	XGL_CURVE_METRIC_VDC	XGL_CTX_SURF_FPAT
XGL_CURVE_METRIC_VDC	XGL_RAS_HEIGHT	XGL_CTX_SURF_FPAT_POSITION
	XGL_RAS_WIDTH	XGL_CTX_SURF_FRONT_COLOR
	XGL_VDC_MAP_ASPECT	XGL_CTX_SURF_FRONT_FILL_ STYLE
		XGL_CURVE_METRIC_VDC
		XGL_SURF_FILL_STIPPLE

Bug Test Descriptions 7

This chapter describes the escalated customer bug test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section, “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ bug1070480

Test Types:	RGB, CM
Description:	Copies 1-bit memory raster to 8-bit window raster
Attributes Tested:	`XGL_CTX_SURF_FRONT_COLOR`
Operators Tested:	`xgl_context_copy_raster`
Output:	Draws bars of rasters with different colors.

▼ bug1070568

Test Types:	RGB, CM
Description:	Spot light cone angle twice as big on GT

Attributes Tested:	`XGL_FACET_NORMAL`
Operators Tested:	xgl_triangle_strip
Output:	Draws a triangle with 2 facets; first facet as green solid, and the second as blue hollow.

▼ bug1076434

Test Types:	RGB, CM
Description:	Surface context interferes with the line color context
Attributes Tested:	`XGL_CTX_LINE_COLOR` `XGL_3D_CTX_SURF_FRONT_DIFFUSE`
Operators Tested:	`xgl_multipolyline` `xgl_object_set`
Output:	Program renders four stars using four different line colors

▼ bug1077883

Test Types:	RGB, CM
Description:	Copies 1-bit memory raster to 32-bit window raster with OR
Attributes Tested:	`XGL_CTX_ROP`
Operators Tested:	`xgl_context_copy_raster`
Output:	Red square with white cross-cursor OR-ed to raster

▼ bug1077884

Test Types:	RGB, CM
Description:	Copies 1-bit memory raster to 32-bit window raster with XOR
Attributes Tested:	`XGL_CTX_ROP`
Operators Tested:	`xgl_context_copy_raster`
Output:	Red square with white cross-cursor XOR-ed to raster shown in blue

▼ bug1078413

Test Types:	RGB, CM
Description:	XGL colormap can get lost upon repaint.

Attributes Tested:	XGL_COLOR_RGB
	XGL_2D_CTX
Operators Tested:	xgl_polygon
	xgl_object_create
Output:	Two raswins

bug1084188

Test Types:	RGB, CM
Description:	XDrawline() will not draw to XGL canvas.
Attributes Tested:	XGL_COLOR_RGB
	XGL_FACET_NONE
Operators Tested:	xgl_polygon
Output:	Line in X, polygon in XGL

bug1086156

Test Types:	RGB, CM
Description:	Cannot write to the far right and bottom pixel line
Attributes Tested:	XGL_MULTIRECT_I2D
Operators Tested:	xgl_multirectangle
Output:	White rectangle to fill the window

bug1086427

Test Types:	RGB, CM
Description:	Solid lines and Opaque fills are not XOR'd correctly.
Attributes Tested:	XGL_CTX_ROP
Operators Tested:	xgl_multi_simple_polygon
	xgl_multipolyline
Output:	A 3D polygon with a 3D polyline outline

bug1107625

Test Types:	RGB, CM
Description:	Draws a quadmesh with vertex colors, illum_none, and color selector, illum_indep.
Attributes Tested:	XGL_3D_CTX_SURF_FRONT_ILLUMINATION
	XGL_CTX_SURF_FRONT_COLOR_SELECTOR

Operators Tested:	xgl_quadrilateral_mesh
Output:	A red color quad

bug1176656

Test Types:	RGB, CM
Description:	Quadmesh XGL_FACET_COLOR interpolates incorrectly
Attributes Tested:	XGL_3D_CTX_SURF_FRONT_ILLUMINATION XGL_CTX_SURF_FRONT_COLOR_SELECTOR
Operators Tested:	xgl_quadrilateral_mesh
Output:	20x20 quadmesh with FACET_COLOR

bug1180099

Test Types:	INDEX, SM
Description:	Application SEGV when drawing large circles
Attributes Tested:	XGL_CTX_SURF_FRONT_FILL_STYLE
Operators Tested:	xgl_multicircle
Output:	Several large circles

bug1181797

Test Types:	INDEX, SM
Description:	Multiple rasters cause one cmap to core dump
Attributes Tested:	XGL_RAS_COLOR_MAP
Operators Tested:	xgl_object_create xgl_object_destroy
Output:	Multiple rasters creation and destruction

bug1182918

Test Types:	RGB, CM
Description:	One pixel line does not draw on cfb.
Attributes Tested:	XGL_LINE_SOLID
Operators Tested:	xgl_multipolyline
Output:	Forty one-pixel lines

bug1187204

Test Types:	RGB, CM
Description:	Renders wide line with round cap via both XGL and Xlib.
Attributes Tested:	XGL_CTX_LINE_CAP XGL_CTX_LINE_JOIN XGL_LPAT_DATA
Operators Tested:	xgl_multipolyline
Output:	Four wide lines, the first one via Xlib, and the remaining via XGL.

bug1187204i

Test Types:	INDEX, CM
Description:	Renders wide line with round cap via both XGL and Xlib.
Attributes Tested:	XGL_CTX_LINE_CAP XGL_CTX_LINE_JOIN XGL_LPAT_DATA
Operators Tested:	xgl_multipolyline
Output:	Four wide lines, the first one via Xlib, and the remaining via XGL.

bug1191129

Test Types:	RGB, SM
Description:	Test vertex colored, depth-cued polylines
Attributes Tested:	XGL_3D_CTX_DEPTH_CUE_MODE
Operators Tested:	xgl_multipolyline
Output:	Four red lines with vertex color

bug1194656

Test Types:	INDEX, SM
Description:	GX dump core with xgl_triangle_strip
Attributes Tested:	XGL_FACET_COLOR XGL_FACET_NONE XGL_FACET_NORMAL XGL_FACET_COLOR_NORMAL
Operators Tested:	xgl_triangle_strip
Output:	Triangles with different facet types

▼ bug1201325

Test Types:	RGB, SM
Description:	SEGV in clipTstar for `xgl_triangle_list`
Attributes Tested:	None
Operators Tested:	`xgl_triangle_list`
Output:	Clipped triangles

CGM Test Descriptions

8

This chapter describes the Computer Graphics Metafile (CGM) test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section, "Denizen Test Types" on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ cgm0

Test Types:	INDEX, SM
Description:	Tests currently implemented CGM VDC extents and attributes listed below. Bare bones case test: draws line for XGL_VDC_MAP_ASPECT. Also see if setting the two different CGM VDC extents takes effect, and if the XGL_CGM_SCALE_FACT are properly recorded in the CGM output file for XGL_CGM_METRIC.
Attributes Tested:	XGL_CGM_VDC_EXT_SHORT and XGL_CGM_VDC_EXT_LONG for XGL_CGM_CLEAR_TEXT CGM output file type, XGL_CGM_CLEAR_TEXT

	XGL_VDC_MAP_ASPECT
	XGL_CGM_SCALE_FACT
	XGL_CGM_METRIC
Operators Tested:	None
Output:	A line

▼ cgm1

Test Types:	INDEX, SM
Description:	Tests currently implemented CGM VDC extents, and attributes listed below. Verifies line types, line width as well as line colors are properly output to the CGM output file.
Attributes Tested:	XGL_CGM_VDC_EXT_SHORT and XGL_CGM_VDC_EXT_LONG for XGL_CGM_CLEAR_TEXT
Operators Tested:	None
Output:	CGM output file type

▼ cgm2

Test Types:	INDEX, SM
Description:	Tests currently implemented CGM VDC extents and attributes listed below. Verifies line types, line width as well as line colors are properly output to the CGM output file when they are pushed and popped from stack.
Attributes Tested:	XGL_CGM_VDC_EXT_SHORT and XGL_CGM_VDC_EXT_LONG for XGL_CGM_CLEAR_TEXT
Operators Tested:	None
Output:	CGM output file type

▼ cgm3

Test Types:	INDEX, SM
Description:	Tests currently implemented CGM VDC extents and attributes listed below. Verifies line and marker clipping are properly output to the CGM output file.
Attributes Tested:	XGL_CGM_VDC_EXT_SHORT and XGL_CGM_VDC_EXT_LONG for XGL_CGM_CLEAR_TEXT
Operators Tested:	None

Output: CGM output file type

▼ cgm4

Test Types:	INDEX, SM
Description:	Tests currently implemented CGM VDC extents and attributes listed below. Verifies transformed lines are properly output when pushed and popped on the stack for local, global and view transformations.
Attributes Tested:	XGL_CGM_VDC_EXT_SHORT and XGL_CGM_VDC_EXT_LONG for XGL_CGM_CLEAR_TEXT
Operators Tested:	None
Output:	CGM output file type

▼ cgm5

Test Types:	INDEX, SM
Description:	Tests currently implemented CGM VDC extents and attributes listed below. Checks that marker type, color and size are properly output to the CGM output file.
Attributes Tested:	XGL_CGM_VDC_EXT_SHORT and XGL_CGM_VDC_EXT_LONG for XGL_CGM_CLEAR_TEXT
Operators Tested:	None
Output:	CGM output file type

▼ cgm6

Test Types:	INDEX, SM
Description:	Tests currently implemented CGM VDC extents and attributes listed below. Checks that marker size, type and color are properly recorded to the CGM output file when pushed and popped on stack.
Attributes Tested:	XGL_CGM_VDC_EXT_SHORT and XGL_CGM_VDC_EXT_LONG for XGL_CGM_CLEAR_TEXT
Operators Tested:	None
Output:	CGM output file type

▼ cgm7

Test Types:	INDEX, SM
Description:	Tests currently implemented CGM VDC extents and attributes listed below. Checks that polygons interior style, interior color, edge type, edge width, edge flag are properly recorded in the CGM output file.
Attributes Tested:	`XGL_CGM_VDC_EXT_SHORT` and `XGL_CGM_VDC_EXT_LONG` for `XGL_CGM_CLEAR_TEXT`
Operators Tested:	`None`
Output:	CGM output file type

▼ cgm8

Test Types:	INDEX, SM
Description:	Tests pushing & popping of contexts for *xgl_polygon* for the attributes fill style & colour.

Attributes Tested:	`XGL_CGM_DESCRIPTION`
	`XGL_CGM_ENCODING`
	`XGL_CGM_VDC_EXTENT`
	`XGL_CGM_COLOR_MAP`

Operators Tested:	`None`
Output:	CGM output file type

▼ cgm9

Test Types:	INDEX, SM
Description:	Tests circle, circle_arc, circle_arc_close, ellipse, ellipse_arc, ellipse_arc_close

Attributes Tested:	`XGL_CGM_DESCRIPTION`
	`XGL_CGM_ENCODING`
	`XGL_CGM_VDC_EXTENT`
	`XGL_CGM_COLOR_MAP`

Operators Tested:	`xgl_multicircle` `xgl_arc`
Output:	CGM output file type

xgl_stream

Test Types:	RGB, SM
Description:	Tests XGL object XGL_STREAM
Attributes Tested:	XGL_STREAM
Operators Tested:	xgl_multirectangle xgl_multi_simple_polygon xgl_multipolyline xgl_polygon
Output:	CGM output file type

set_get_cgm_attrs

Test Types:	RGB, SM
Description:	Tests the setting and getting of CGM attributes
Attributes Tested:	XGL_CGM_DEV XGL_CGM_DESCRIPTION XGL_CGM_ENCODING XGL_CGM_PICTURE_DESCRIPTION XGL_CGM_VDC_EXTENT XGL_CGM_VDC_EXT_SHORT XGL_CGM_SCALE_MODE XGL_CGM_METRIC XGL_CGM_SCALE_FACTOR
Operators Tested:	xgl_object_create xgl_object_get xgl_object_destroy
Output:	CGM output file type

Circle Test Descriptions 9

This chapter describes the Circle test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section, “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ circle0

Test Types:	INDEX, SM
Description:	Checks five points of a 2D indexed circle
Attributes Tested:	See Table 9-1, Column A at the end of this chapter.
Operators Tested:	`xgl_multicircle` `xgl_object_set`
Output:	One 2D indexed circle

circle1

Test Types: INDEX, SM
Description: Checks five points of each of three 2D indexed circles
Attributes Tested: See Table 9-1, Column A at the end of this chapter.
Operators Tested: `xgl_multicircle`
`xgl_object_set`
Output: Three 2D indexed circles

circle2

Test Types: INDEX, SM
Description: Checks for the presence / non-presence of edges in three circles, each drawn with edges both on and off
Attributes Tested: `XGL_CTX_EDGE_COLOR`
and Table 9-1, Column A at the end of this chapter
Operators Tested: `xgl_multicircle`
`xgl_object_set`
Output: Three 2D indexed circles drawn twice, once with edges on and once with them off

circle3

Test Types: INDEX, SM
Description: Checks the patterns and the edges of six 2D indexed circles
Attributes Tested: See Table 9-1, Columns A and B at the end of this chapter.
Operators Tested: `xgl_multicircle`
`xgl_object_set`
Output: Three circles with different patterns drawn twice, once with edges on and once with them off

circle4

Test Types: RGB, SM
Description: Checks five points of each of several RGB colored circles. Colors are all colors in a color cube for 8-bit rasters and 256 random colors for other rasters.

Attributes Tested:	XGL_DEV_COLOR_MAP XGL_CMAP_COLOR_CUBE_SIZE and Table 9-1, Column A at the end of this chapter
Operators Tested:	xgl_multicircle xgl_object_set xgl_object_get
Output:	Several RGB circles of different colors

▼ circle5

Test Types:	RGB, SM
Description:	RGB version of *circle0*
Attributes Tested:	See Table 9-1, Column A at the end of this chapter.
Operators Tested:	xgl_multicircle xgl_object_set
Output:	One 2D RGB circle

▼ circle6

Test Types:	RGB, SM
Description:	RGB version of *circle1*
Attributes Tested:	See Table 9-1, Column A at the end of this chapter.
Operators Tested:	xgl_multicircle xgl_object_set
Output:	Three 2D RGB circles

▼ circle7

Test Types:	RGB, SM
Description:	RGB version of *circle2*
Attributes Tested:	XGL_CTX_EDGE_COLOR and Table 9-1, Column A at the end of this chapter
Operators Tested:	xgl_multicircle xgl_object_set
Output:	Three 2D RGB circles drawn twice, once with edges on and once with them off

circle8

Test Types: RGB, SM
Description: RGB version of *circle3*
Attributes Tested: See Table 9-1, Columns A and B at the end of this chapter.
Operators Tested: `xgl_multicircle`
`xgl_object_set`
`xgl_object_get`
Output: Three circles with different patterns drawn twice, once with edges on and once with them off

circle9

Test Types: INDEX, SM
Description: 3D version of *circle0*
Attributes Tested: See Table 9-1, Column A at the end of this chapter.
Operators Tested: `xgl_multicircle`
`xgl_object_set`
Output: One 3D indexed circle

circle10

Test Types: INDEX, SM
Description: 3D version of *circle1*
Attributes Tested: See Table 9-1, Column A at the end of this chapter.
Operators Tested: `xgl_multicircle`
`xgl_object_set`
Output: Three 3D indexed circles

circle11

Test Types: INDEX, SM
Description: 3D version of *circle2*
Attributes Tested: `XGL_CTX_EDGE_COLOR`
and Table 9-1, Column A at the end of this chapter
Operators Tested: `xgl_multicircle`
`xgl_object_set`
Output: Three 3D indexed circles drawn twice, once with edges on and once with them off

circle12

Test Types: INDEX, SM
Description: 3D version of *circle3*
Attributes Tested: See Table 9-1, Columns A and B at the end of this chapter.
Operators Tested: `xgl_multicircle`
`xgl_object_set`
`xgl_object_get`
Output: Three circles with different patterns drawn twice, once with edges on and once with them off

circle13

Test Types: RGB, SM
Description: 3D version of *circle4*
Attributes Tested: `XGL_DEV_COLOR_MAP`
`XGL_CMAP_COLOR_CUBE_SIZE`
and Table 9-1, Column A at the end of this chapter
Operators Tested: `xgl_multicircle`
`xgl_object_set`
`xgl_object_get`
Output: Several RGB circles of different colors

circle14

Test Types: RGB, SM
Description: 3D version of *circle5*
Attributes Tested: See Table 9-1, Column A at the end of this chapter.
Operators Tested: `xgl_multicircle`
`xgl_object_set`
Output: One 3D RGB circle

circle15

Test Types: RGB, SM
Description: 3D version of *circle6*
Attributes Tested: See Table 9-1, Column A at the end of this chapter.
Operators Tested: `xgl_multicircle`
`xgl_object_set`
Output: Three 3D RGB circles

circle16

Test Types:	RGB, SM
Description:	3D version of *circle7*
Attributes Tested:	`XGL_CTX_EDGE_COLOR` and Table 9-1, Column A at the end of this chapter
Operators Tested:	`xgl_multicircle` `xgl_object_set`
Output:	Three 3D RGB circles drawn twice, once with edges on and once with them off

circle17

Test Types:	RGB, SM
Description:	3D version of *circle8*
Attributes Tested:	See Table 9-1, Columns A and B at the end of this chapter

Operators Tested:	`xgl_multicircle`
	`xgl_object_set`
	`xgl_object_get`

Output:	Three circles with different patterns drawn twice, once with edges on and once with them off

circle18

Test Types:	INDEX, SM
Description:	Loops through every possible value for the plane mask, clears the plane mask by setting it to -1, and then sets it to -1^i. Sets the surface color to 0xff ^i, and then samples five points of the circle for this color.

Attributes Tested:	`XGL_CTX_PLANE_MASK`
	`XGL_CTX_SURF_EDGE_FLAG`
	`XGL_CURVE_METRIC_VDC`

Operators Tested:	`xgl_multicircle` `xgl_object_set`
Output:	Three sets of `0xff` number of circles

circle19

Test Types: INDEX, SM

Description: Four circles are rendered utilizing bounding box with non-null values, different index colors, and four different values for XGL_CTX_NURBS_CURVE_APPROX

Attributes Tested: See Table 9-1, Column C at the end of this chapter.

Operators Tested: `xgl_multicircle`
`xgl_object_set`

Output: Four different colored circles, side by side, along the top width of the window raster

circle20

Test Types: INDEX, SM

Description: Four annotation circles are rendered utilizing bounding box with non-null values, different index colors, and four different values for XGL_CTX_NURBS_CURVE_APPROX and for their radius values

Attributes Tested: See Table 9-1, Column C at the end of this chapter.

Operators Tested: `xgl_multicircle`
`xgl_object_set`

Output: Four different colored index annotation circles with different radius values

circle21

Test Types: INDEX, SM

Description: Four different colored 3D index circles rendered utilizing bound box with non-null values and different values for XGL_CTX_NURBS_CURVE_APPROX with each circle inside a different plane composed of non-normalized directional vectors

Attributes Tested: See Table 9-1, Column C at the end of this chapter.

Operators Tested: `xgl_multicircle`
`xgl_object_set`

Output: Four different colored 3D index circles with each circle inside a different plane

circle22

Test Types: INDEX, SM

Description: Seven point type I2D circles with different centers, radiuses, bounding boxes, and colors. Four translated and scaled I2D circles through the utilization of transformations for scaling and translation applied to the global model transformation.

Attributes Tested:
XGL_CTX_GLOBAL_MODEL_TRANS
XGL_TRANS_REPLACE
XGL_TRANS_POSTCONCAT
and Table 9-1, Column A at the end of this chapter

Operators Tested:
xgl_object_set
xgl_multicircle
xgl_context_push
xgl_object_destroy
xgl_object_create
xgl_transform_scale
xgl_transform_translate

Output: Seven different colored 2D index circles with varying radiuses. Four different colored 2D index circles sheared through changes to their global model coordinate system.

circle23

Test Types: RGB, CM

Description: Tests XGL_CTX_CURVE_APPROX_VALUE with xgl_multicircle()

Attributes Tested: XGL_CTX_CURVE_APPROX_VALUE

Operators Tested:
xgl_object_set
xgl_multicircle

Output: Circles with apporximation values 100.0, 50.0, 25.0 and 10.0

Table 9-1 Circle Attributes Tested

Column A	Column B	Column C
XGL_CTX_SURF_EDGE_FLAG	XGL_SURF_FILL_STIPPLE	XGL_CURVE_CONST_PARAM_ SUBDIV_BETWEEN_KNOTS
XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_SURF_FPAT_POSITION	XGL_CTX_NURBS_CURVE_ APPROX_VAL
XGL_CURVE_METRIC_VDC	XGL_CTX_SURF_FPAT	XGL_CTX_SURF_EDGE_FLAG
XGL_CTX_NURBS_CURVE_ APPROX_VAL	XGL_CTX_SURF_FRONT_FILL_ STYLE	XGL_3D_CTX_SURF_FACE_ DISTINGUISH
	XGL_MEM_RAS	XGL_CTX_SURF_FRONT_COLOR
	XGL_RAS_DEPTH	XGL_CURVE_METRIC_WC
	XGL_RAS_WIDTH	XGL_CURVE_CHORDAL_ DEVIATION_WC
	XGL_RAS_HEIGHT	XGL_CURVE_CHORDAL_ DEVIATION_VDC
	XGL_CTX_EDGE_COLOR	

Clipping Test Descriptions 10

This chapter describes the Clipping test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section, “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ clip_viewclip_pg_2d

Test Types:	INDEX, SM
Description:	Views clipping in 2D using all combinations of view clip planes of F2D solid filled polygons with and without edges, concave, and multibounded of F2D solid filled polygons with various edge styles
Attributes Tested:	XGL_CTX_VIEW_CLIP_BOUNDS XGL_CTX_EDGE_COLOR XGL_CTX_BACKGROUND_COLOR XGL_CTX_SURF_EDGE_FLAG XGL_CTX_CLIP_PLANES

	XGL_CLIP_XMIN
	XGL_CLIP_XMAX
	XGL_CLIP_YMIN
	XGL_CLIP_YMAX
	XGL_CTX_EDGE_STYLE
	XGL_LINE_PATTERNED
	XGL_CTX_EDGE_PATTERN
	XGL_LINE_ALT_PATTERNED
	XGL_LINE_SOLID
	XGL_CTX_EDGE_WIDTH_SCALE_FACTOR
Operators Tested:	xgl_polygon
	xgl_object_set
	xgl_object_get
Output:	Three side-by-side polygons that change in a variety of ways as their edge styles and clipping planes vary. Their original shapes are (1) a square with a nearly square donut chunk removed from its center, (2) an arrowhead with the tip facing the top of the window raster, and (3) a bow tie. See Table 10-1 at the end of this chapter for the 2D clipping combinations used.

▼ clip_viewclip_line_pttypes_2d

Test Types:	INDEX, SM
Description:	Views clipping in 2D of thin solid line using all the permitted point types
Attributes Tested:	XGL_CTX_VIEW_CLIP_BOUNDS
	XGL_CTX_CLIP_PLANES
	XGL_CLIP_XMIN
	XGL_CLIP_XMAX
	XGL_CLIP_YMIN
	XGL_CLIP_YMAX
	XGL_CTX_LINE_STYLE
	XGL_LINE_SOLID
Operators Tested:	xgl_multipolyline
	xgl_object_set
Output:	Renders a line with the same vector values but utilizing different point types using the clipping combinations in the order specified in the following text for 2D loop values. The original line segment looks like a lopsided

bow tie with the smaller triangle on the right side and the larger portion of the bow tie open. See Table 10-1 at the end of this chapter for the 2D clipping combinations used.

▼ clip_viewclip_line_styles_2d

Test Types:	INDEX, SM
Description:	Views clipping in 2D of F2D thin lines of all styles and wide lines
Attributes Tested:	XGL_CTX_VIEW_CLIP_BOUNDS XGL_CTX_LINE_WIDTH_SCALE_FACTOR XGL_CTX_CLIP_PLANES XGL_CLIP_XMIN XGL_CLIP_XMAX XGL_CLIP_YMIN XGL_CLIP_YMAX XGL_LINE_SOLID XGL_CTX_LINE_WIDTH_SCALE_FACTOR XGL_CTX_LINE_STYLE XGL_LINE_PATTERNED XGL_CTX_LINE_PATTERN
Operators Tested:	xgl_multipolyline xgl_object_set
Output:	Renders a line with the same vector values but utilizing different line styles using the clipping combinations in the order specified below for 2D loop values. The original line segment looks like a lopsided bow tie with the smaller triangle on the right side and the larger portion of the bow tie open. See Table 10-1 at the end of this chapter for the 2D clipping combinations used.

▼ clip_viewclip_pg_2d_1

Test Types:	INDEX, SM
Description:	Views clipping in 2D of F2D polygons with all possible fill styles
Attributes Tested:	XGL_CTX_VIEW_CLIP_BOUNDS XGL_CTX_SURF_FPAT XGL_CTX_BACKGROUND_COLOR XGL_CTX_CLIP_PLANES

	XGL_CLIP_XMIN
	XGL_CLIP_XMAX
	XGL_CLIP_YMIN
	XGL_CLIP_YMAX
	XGL_CTX_SURF_FRONT_FILL_STYLE
	XGL_SURF_FILL_SOLID
	XGL_SURF_FILL_HOLLOW
	XGL_SURF_FILL_OPAQUE_STIPPLE
	XGL_CTX_EDGE_WIDTH_SCALE_FACTOR
	XGL_SURF_FILL_EMPTY
Operators Tested:	xgl_polygon
	xgl_object_set
	xgl_object_get
Output:	Three side-by-side polygons that change in a variety of ways as their interior fill style and clipping planes vary. Their original shapes are (1) a square with a nearly square donut chunk removed from its center, (2) an arrow head with the tip facing the top of the raster, and (3) an arrowhead with a small portion of its staff and its arrow tip facing the left side of the window raster. See Table 10-1 at the end of this chapter for the 2D clipping combinations used.

▼ clip_viewclip_marker_2d

Test Types:	INDEX, SM
Description:	Views clipping in 2D of F2D multimarkers with all possible marker styles
Attributes Tested:	XGL_CTX_VIEW_CLIP_BOUNDS
	XGL_CTX_CLIP_PLANES
	XGL_CLIP_XMIN
	XGL_CLIP_XMAX
	XGL_CLIP_YMIN
	XGL_CLIP_YMAX
	XGL_CTX_MARKER
Operators Tested:	xgl_multimarker
	xgl_object_set
Output:	Five markers which form a cross that changes in a variety of ways as their clipping planes vary. See Table 10-1 for the 2D clipping combinations used.

clip_viewclip_multiarc_2d

Test Types:	INDEX, SM
Description:	Views clipping in 2D of F2D multicircles with all possible fill styles, thin and wide edge styles; of F2D closed multiarcs with all possible fill styles, thin and wide edge styles, and possible thin and wide line styles
Attributes Tested:	XGL_CTX_VIEW_CLIP_BOUNDS XGL_CTX_NURBS_CURVE_APPROX_VAL XGL_CTX_SURF_FPAT XGL_CTX_CLIP_PLANES XGL_CLIP_XMIN XGL_CLIP_XMAX XGL_CLIP_YMIN XGL_CLIP_YMAX XGL_CTX_SURF_FRONT_FILL_STYLE XGL_SURF_FILL_SOLID XGL_CTX_ARC_FILL_STYLE XGL_ARC_SECTOR XGL_ARC_CHORD XGL_SURF_FILL_HOLLOW XGL_SURF_FILL_OPAQUE_STIPPLE XGL_CTX_EDGE_COLOR XGL_CTX_SURF_EDGE_FLAG XGL_CTX_EDGE_WIDTH_SCALE_FACTOR XGL_CTX_EDGE_WIDTH_SCALE_FACTOR XGL_CTX_EDGE_STYLE XGL_LINE_SOLID XGL_LINE_PATTERNED XGL_CTX_EDGE_PATTERN XGL_LINE_ALT_PATTERNED XGL_ARC_OPEN XGL_CTX_LINE_COLOR XGL_CTX_LINE_WIDTH_SCALE_FACTOR XGL_CTX_LINE_STYLE XGL_CTX_LINE_PATTERN
Operators Tested:	xgl_multiarc xgl_object_set xgl_object_get

Output:	Four arcs that change in a variety of ways as their interior fill style and clipping planes vary. Their positions are leftmost arc, highest arc, rightmost arc, and lowest arc. The leftmost arc is directly across from the rightmost arc, and the highest arc is directly above the lowest arc. Their original unclipped appearances described clockwise are (1) an arc rendered counterclockwise from 2 p.m. to 5 p.m., (2) an arc rendered counterclockwise from 5 p.m. to 7 p.m., (3) an arc rendered counterclockwise from 6 p.m. to 12 p.m., and finally the lowest arc, (4) an arc rendered counterclockwise from 12 p.m. to 5 p.m. See Table 10-1 at the end of this chapter for the 2D clipping combinations used.

▼ clip_viewclip_multicircle_2d

Test Types:	INDEX, SM
Description:	Views clipping in 2D of F2D multicircles with all possible fill styles, thin and wide edge styles
Attributes Tested:	XGL_CTX_VIEW_CLIP_BOUNDS XGL_CTX_NURBS_CURVE_APPROX_VAL XGL_CTX_SURF_FPAT XGL_CTX_CLIP_PLANES XGL_CLIP_XMIN XGL_CLIP_XMAX XGL_CLIP_YMIN XGL_CLIP_YMAX XGL_CTX_SURF_FRONT_FILL_STYLE XGL_SURF_FILL_SOLID XGL_SURF_FILL_HOLLOW XGL_SURF_FILL_OPAQUE_STIPPLE XGL_CTX_EDGE_COLOR XGL_CTX_SURF_EDGE_FLAG XGL_LINE_PATTERNED XGL_CTX_EDGE_WIDTH_SCALE_FACTOR XGL_CTX_EDGE_STYLE XGL_LINE_SOLID XGL_CTX_EDGE_PATTERN XGL_LINE_ALT_PATTERNED

Operators Tested: xgl_multicircle
xgl_object_set
xgl_object_get

Output: Two side-by-side circles that change in a variety of ways as their interior fill style and clipping planes vary. See Table 10-1 at the end of this chapter for the 2D clipping combinations used.

▼ clip_viewclip_nurbs_curve_2d

Test Types: INDEX, SM

Description: Views clipping in 2D of F2D nurbs curves of all possible thin and wide line styles

Attributes Tested: XGL_CTX_VIEW_CLIP_BOUNDS
XGL_CTX_NURBS_CURVE_APPROX_VAL
XGL_CTX_SURF_FPAT
XGL_CTX_LINE_COLOR
XGL_CTX_LINE_WIDTH_SCALE_FACTOR
XGL_CTX_CLIP_PLANES
XGL_CLIP_XMIN
XGL_CLIP_XMAX
XGL_CLIP_YMIN
XGL_CLIP_YMAX
XGL_CTX_LINE_STYLE
XGL_LINE_SOLID
XGL_CTX_LINE_WIDTH_SCALE_FACTOR
XGL_CTX_LINE_STYLE
XGL_LINE_PATTERNED
XGL_CTX_LINE_PATTERN
XGL_LINE_ALT_PATTERNED

Operators Tested: xgl_nurbs_curve
xgl_object_set
xgl_object_get

Output: Nurbs curve in the shape of a seed that changes in a variety of ways as both of its edge styles and line styles change as well as its clipping planes. See Table 10-1 for the 2D clipping combinations used.

▼ clip_viewclip_pg_bbox_2d

Test Types:	INDEX, SM
Description:	Same as *clip_viewclip_pg_2d* but with primitives in a bounding box
Attributes Tested:	XGL_SYS_ST_ERROR_DETECTION XGL_CTX_VIEW_CLIP_BOUNDS XGL_CTX_EDGE_COLOR XGL_CTX_CLIP_PLANES XGL_CLIP_XMIN XGL_CLIP_XMAX XGL_CLIP_YMIN XGL_CLIP_YMAX XGL_CTX_SURF_EDGE_FLAG
Operators Tested:	xgl_polygon xgl_object_set
Output:	Three side-by-side polygons that change in a variety of ways as their edge styles and their clipping planes vary. Their original shapes are (1) a square with a nearly square donut chunk removed from its center, (2) an arrowhead with the tip facing the top of the window raster, and (3) a bow tie. See Table 10-1 at the end of this chapter for the 2D clipping combinations used.

▼ clip_viewclip_qm

Test Types:	INDEX, SM
Description:	Views clipping in 3D of F3D quadrilateral meshes of all possible fill styles, and thin and wide edge styles
Attributes Tested:	XGL_CTX_VIEW_CLIP_BOUNDS XGL_CTX_SURF_FPAT XGL_CTX_EDGE_COLOR XGL_CTX_SURF_EDGE_FLAG,XGL_CLIP_XMIN XGL_CTX_EDGE_WIDTH_SCALE_FACTOR XGL_CTX_CLIP_PLANES XGL_CLIP_XMAX XGL_CLIP_YMIN XGL_CLIP_YMAX XGL_CLIP_ZMIN XGL_CLIP_ZMAX

XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_CTX_EDGE_WIDTH_SCALE_FACTOR
XGL_SURF_FILL_EMPTY
XGL_SURF_FILL_SOLID
XGL_SURF_FILL_HOLLOW
XGL_SURF_FILL_OPAQUE_STIPPLE

Operators Tested: xgl_quadrilateral_mesh
xgl_object_set
xgl_object_get

Output: Two quadmesh side-by-side edges touching and the left side quadmesh is perhaps a little less than one third the side of its adjacent quadmesh. Both change in a variety of ways as their fill styles and edge style vary as well as their clipping planes. See Table 10-1 at the end of this chapter for the 3D clipping combinations used.

▼ clip_viewclip_rect_2d

Test Types: INDEX, SM

Description: Views clipping in 2D of F2D multirectangles with all possible fill styles and some edge combinations

Attributes Tested: XGL_CTX_VIEW_CLIP_BOUNDS
XGL_CTX_SURF_FPAT
XGL_CTX_BACKGROUND_COLOR
XGL_CTX_CLIP_PLANES
XGL_CLIP_XMIN
XGL_CLIP_XMAX
XGL_CLIP_YMIN
XGL_CLIP_YMAX
XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_SURF_FILL_SOLID
XGL_SURF_FILL_HOLLOW
XGL_SURF_FILL_OPAQUE_STIPPLE
XGL_CTX_EDGE_COLOR
XGL_CTX_SURF_EDGE_FLAG
XGL_CTX_EDGE_WIDTH_SCALE_FACTOR

Operators Tested: xgl_multirectangle
xgl_object_set
xgl_object_get

Output:	Three rectangles practically in a row with the last rectangle being the largest. All three rectangles change in a variety of ways as their fill styles and edge style vary as well as their clipping planes. See Table 10-1 at the end of this chapter for the 2D clipping combinations used.

▼ clip_viewclip_stext_2d

Test Types:	INDEX, SM
Description:	Views clipping in 2D of stroke text with a variety of different point types

Attributes Tested:	`XGL_CTX_VIEW_CLIP_BOUNDS`
	`XGL_CTX_STEXT_ALIGN_HORIZ`
	`XGL_STEXT_ALIGN_HORIZ_CENTER`
	`XGL_CTX_STEXT_ALIGN_VERT`
	`XGL_STEXT_ALIGN_VERT_HALF`
	`XGL_CTX_CLIP_PLANES`
	`XGL_CLIP_XMIN`
	`XGL_CLIP_XMAX`
	`XGL_CLIP_YMIN`
	`XGL_CLIP_YMAX`

Operators Tested:	`xgl_stroke_text` `xgl_object_set`
Output:	Five huge “X”s which form a cross that changes in a variety of ways as their clipping planes vary. See Table 10-1 at the end of the chapter for the 2D clipping combinations used.

▼ clip_viewclip_ts

Test Types:	INDEX, SM
Description:	Views clipping in 3D of `F3D` triangle strips of all possible fill styles, and thin and wide edge styles

Attributes Tested:	`XGL_CTX_VIEW_CLIP_BOUNDS`
	`XGL_CTX_SURF_FPAT`
	`XGL_CTX_EDGE_COLOR`
	`XGL_CTX_SURF_EDGE_FLAG`
	`XGL_CTX_EDGE_WIDTH_SCALE_FACTOR`
	`XGL_CTX_CLIP_PLANES`
	`XGL_CLIP_XMIN`

	XGL_CLIP_XMAX
	XGL_CLIP_YMIN
	XGL_CLIP_YMAX
	XGL_CLIP_ZMIN
	XGL_CLIP_ZMAX
	XGL_CTX_SURF_FRONT_FILL_STYLE
	XGL_SURF_FILL_SOLID
	XGL_SURF_FILL_EMPTY
	XGL_SURF_FILL_HOLLOW
	XGL_SURF_FILL_OPAQUE_STIPPLE
Operators Tested:	xgl_triangle_strip xgl_object_set xgl_object_get
Output:	Two tristrips connected by a vertical base which appear like two cones attached at a base that change in a variety of ways as their interior styles, their edges, and their clipping planes vary. See Table 10-1 at the end of the chapter for the 3D clipping combinations used.

▼ clip_viewclip_line_pttypes_3d

Test Types:	INDEX, SM
Description:	Views clipping in 3D of thin solid line using all the permitted point types
Attributes Tested:	XGL_CTX_VIEW_CLIP_BOUNDS XGL_CTX_CLIP_PLANES XGL_CLIP_XMIN XGL_CLIP_XMAX XGL_CLIP_YMIN XGL_CLIP_YMAX
Operators Tested:	xgl_multipolyline xgl_object_set
Output:	Same line segments rendered using all the permissible point types. The line segments correct appearance when no clipping is in effect should look like a lopsided “x” on its side with the small portion closest to the right side of the window raster. Although this is a 3D test, see Table 10-1 at the end of the chapter for the 2D clipping combinations used.

clip_viewclip_pg_3d

Test Types:	INDEX, SM
Description:	Views clipping in 3D using all combinations of view clip planes of F3D solid-filled polygons with and without edges, concave, and multibounded of F3D solid-filled polygons with various edge styles
Attributes Tested:	XGL_CTX_VIEW_CLIP_BOUNDS XGL_CTX_EDGE_COLOR XGL_CTX_BACKGROUND_COLOR XGL_CTX_SURF_EDGE_FLAG XGL_CTX_EDGE_STYLE XGL_LINE_SOLID XGL_CTX_CLIP_PLANES XGL_CLIP_XMIN XGL_CLIP_XMAX XGL_CLIP_YMIN XGL_CLIP_YMAX XGL_CLIP_ZMIN XGL_CLIP_ZMAX XGL_LINE_PATTERNED XGL_CTX_EDGE_PATTERN XGL_LINE_ALT_PATTERNED XGL_CTX_EDGE_WIDTH_SCALE_FACTOR
Operators Tested:	xgl_polygon xgl_object_set xgl_object_get
Output:	Three side by side polygons that change in a variety of ways as their edge styles and their clipping planes vary. Their original shapes are (1) a square with a nearly square donut chunk removed from its center, (2) an arrowhead with the tip facing the top of the window raster, and (3) a bow tie. See Table 10-1 at the end of this chapter for the 3D clipping combinations used.

clip_viewclip_line_styles_3d

Test Types:	INDEX, SM
Description:	Views clipping in 3D of F3D lines of all thin and wide styles (only solid style for wide lines)

Attributes Tested:	XGL_CTX_VIEW_CLIP_BOUNDS XGL_CTX_LINE_WIDTH_SCALE_FACTOR XGL_CTX_CLIP_PLANES XGL_CLIP_XMIN XGL_CLIP_XMAX XGL_CLIP_YMIN XGL_CLIP_YMAX XGL_CTX_LINE_STYLE XGL_LINE_PATTERNED XGL_CTX_LINE_PATTERN XGL_LINE_SOLID
Operators Tested:	xgl_multipolyline xgl_object_set xgl_object_get
Output:	Same line segments rendered using a variety of line types and widths. The line segments correct appearance when no clipping is in effect should look like a lopsided "x" on its side with the small portion closest to the right side of the window raster. Although this is a 3D test, see Table 10-1 at the end of the chapter for the 2D clipping combinations used.

▼ clip_viewclip_marker_3d

Test Types:	INDEX, SM
Description:	Views clipping in 3D of F3D multimarkers for a variety of marker types
Attributes Tested:	XGL_CTX_VIEW_CLIP_BOUNDS XGL_CTX_CLIP_PLANES XGL_CLIP_XMIN XGL_CLIP_XMAX XGL_CLIP_YMIN XGL_CLIP_YMAX XGL_CTX_MARKER
Operators Tested:	xgl_multimarker xgl_object_set
Output:	Five markers which form a cross that changes in a variety of ways as their clipping planes vary. Although this is a 3D test, see Table 10-1 at the end of this chapter for the 2D clipping combinations used.

▼ clip_viewclip_stext_3d

Test Types:	INDEX, SM
Description:	Views clipping in 3D of stroke text
Attributes Tested:	XGL_CTX_VIEW_CLIP_BOUNDS XGL_CTX_STEXT_ALIGN_HORIZ XGL_STEXT_ALIGN_HORIZ_CENTER XGL_CTX_STEXT_ALIGN_VERT XGL_STEXT_ALIGN_VERT_HALF XGL_CTX_CLIP_PLANES XGL_CLIP_XMIN XGL_CLIP_XMAX XGL_CLIP_YMIN XGL_CLIP_YMAX
Operators Tested:	xgl_stroke_text xgl_object_set
Output:	Five huge “X”s which form a cross that changes in a variety of ways as their clipping planes vary. Although this test is 3D, see Table 10-1 at the end of this chapter for the 2D clipping combinations used.

▼ clip_viewclip_pg_3d_1

Test Types:	INDEX, SM
Description:	Views clipping in 3D of F3D polygons of all possible fill styles with and without edges, concave, and multibounded with various edge styles
Attributes Tested:	XGL_CTX_VIEW_CLIP_BOUNDS XGL_CTX_SURF_FPAT XGL_CTX_EDGE_COLOR XGL_CTX_SURF_EDGE_FLAG XGL_CTX_EDGE_WIDTH_SCALE_FACTOR XGL_CTX_CLIP_PLANES XGL_CLIP_XMIN XGL_CLIP_XMAX XGL_CLIP_YMIN XGL_CLIP_YMAX XGL_CLIP_ZMIN XGL_CLIP_ZMAX XGL_CTX_SURF_FRONT_FILL_STYLE

	XGL_SURF_FILL_SOLID
	XGL_SURF_FILL_EMPTY
	XGL_CTX_SURF_EDGE_FLAG
	XGL_SURF_FILL_HOLLOW
	XGL_SURF_FILL_OPAQUE_STIPPLE
Operators Tested:	xgl_polygon
	xgl_object_set
Output:	Three side by side polygons that change in a variety of ways as their edge styles and their clipping planes vary. Their original shapes are (1) a square with a nearly square donut chunk removed from its center, (2) an arrowhead with the tip facing the top of the window raster, and (3) a bow tie. See Table 10-1 at the end of this chapter for the 3D clipping combinations used.

▼ clip_viewclip_nurbs_curve_3d

Test Types:	INDEX, SM
Description:	Views clipping in 3D of F3D nurbs curves of all possible thin and wide line styles
Attributes Tested:	XGL_CTX_VIEW_CLIP_BOUNDS
	XGL_CTX_NURBS_CURVE_APPROX_VAL
	XGL_CTX_LINE_COLOR
	XGL_CTX_CLIP_PLANES
	XGL_CTX_LINE_WIDTH_SCALE_FACTOR
	XGL_CLIP_XMIN
	XGL_CLIP_XMAX
	XGL_CLIP_YMIN
	XGL_CLIP_YMAX
	XGL_CLIP_ZMIN
	XGL_CLIP_ZMAX
	XGL_CTX_LINE_STYLE
	XGL_LINE_SOLID
	XGL_LINE_PATTERNED
	XGL_CTX_LINE_PATTERN
	XGL_LINE_ALT_PATTERNED
Operators Tested:	xgl_nurbs_curve
	xgl_object_set
	xgl_object_get

Output: Nurbs curve in the shape of a seed that changes in a variety of ways as both its edge style and line style vary as well as its clipping planes. See Table 10-1 at the end of this chapter for the 3D clipping combinations used.

▼ clip_viewclip_pg_bbox_3d

Test Types: INDEX, SM

Description: Similar to *clip_viewclip_pg_3d* but with primitives in a bounding box. Views clipping in 3D using all combinations of view clip planes of F3D solid-filled polygons with and without edges, concave, and multibounded F3D polygons.

Attributes Tested:
XGL_CTX_VIEW_CLIP_BOUNDS
XGL_CTX_EDGE_COLOR
XGL_CTX_BACKGROUND_COLOR
XGL_CTX_SURF_EDGE_FLAG
XGL_CTX_CLIP_PLANES
XGL_CLIP_XMIN
XGL_CLIP_XMAX
XGL_CLIP_YMIN
XGL_CLIP_YMAX
XGL_CLIP_ZMIN
XGL_CLIP_ZMAX

Operators Tested:
xgl_polygon
xgl_object_set

Output: Three side-by-side polygons that change in a variety of ways as their edge styles and their clipping planes vary. Their original shapes are (1) a square with a nearly square donut chunk removed from its center, (2) an arrowhead with the tip facing the top of the window raster, and (3) a bow tie. See Table 10-1 at the end of this chapter for the 3D clipping combinations used.

clip_viewportclip_pg_2d

Test Types:	INDEX, SM
Description:	Clipping to the bounds of XGL_CTX_DC_VIEWPORT in 2D of F2D solid-filled concave and multibounded polygons with and without thin solid edges and all combinations of XGL_CTX_VDC_MAP and XGL_CTX_VDC_ORIENTATION (clipping to DC viewport extent of F2D polygons in 2D)
Attributes Tested:	XGL_Y_UP_Z_TOWARD XGL_Y_DOWN_Z_AWAY XGL_CTX_VDC_ORIENTATION XGL_CTX_VDC_MAP XGL_VDC_MAP_OFF XGL_CTX_DC_VIEWPORT XGL_CTX_VDC_WINDOW XGL_CTX_SURF_EDGE_FLAG XGL_VDC_MAP_ALL XGL_VDC_MAP_ASPECT
Operators Tested:	xgl_polygon xgl_object_set xgl_object_get
Output:	Three side-by-side polygons whose positioning inside the window raster changes as the orientation and viewport vary. Their original shapes are (1) a long thin appearing "|", (2) a wide bodied "V", and (3) an arrowhead with a small portion of its staff with its tip facing the left side of the window raster.

clip_viewportclip_pg_3d

Test Types:	INDEX, SM
Description:	Similar to *clip_viewportclip_pg_2d* except 3D instead of 2D. Clipping to the bounds of XGL_CTX_DC_VIEWPORT in 3D of F3D solid-filled concave and multibounded polygons with and without thin solid edges using both rules for interior and all combinations of XGL_CTX_VDC_MAP and XGL_CTX_VDC_ORIENTATION. Checks that XGL_CTX_DC_VIEWPORT and VDC_WINDOW can be set properly.

Attributes Tested:	XGL_Y_UP_Z_TOWARD XGL_Y_DOWN_Z_AWAY XGL_CTX_VDC_MAP XGL_VDC_MAP_OFF XGL_CTX_DC_VIEWPORT XGL_CTX_VDC_WINDOW XGL_CTX_SURF_EDGE_FLAG XGL_VDC_MAP_ALL XGL_VDC_MAP_ASPECT
Operators Tested:	xgl_polygon xgl_object_set xgl_object_get
Output:	Three side-by-side polygons whose positioning inside the window raster changes as the orientation and viewport vary. Their original shapes are (1) a long thin appearing "|", (2) a wide bodied "V", and (3) an arrowhead with a small portion of its staff with its tip facing the left side of the window.

▼ clip_modclip_line_styles_3d

Test Types:	INDEX, SM
Description:	Model clipping in 3D of F3D lines all styles and wide lines
Attributes Tested:	XGL_3D_CTX_MODEL_CLIP_PLANE_NUM XGL_3D_CTX_MODEL_CLIP_PLANES XGL_CTX_LINE_STYLE XGL_LINE_SOLID XGL_CTX_LINE_WIDTH_SCALE_FACTOR XGL_LINE_PATTERNED XGL_CTX_LINE_PATTERN
Operators Tested:	xgl_multipolyline xgl_object_set xgl_object_get
Output:	The correct appearance is line segments that form a "v" on its side near the lower-right corner of the window raster.

▼ clip_modclip_marker_3d

Test Types:	INDEX, SM

Description:	Model clipping in 3D of all possible types of `F3D` multimarkers

Attributes Tested:	`XGL_3D_CTX_MODEL_CLIP_PLANE_NUM`
	`XGL_3D_CTX_MODEL_CLIP_PLANES`
	`XGL_CTX_MARKER`

Operators Tested:	`xgl_multimarker`
	`xgl_object_set`
	`xgl_object_get`

Output:	Although five markers are in the point list, only one should be evident on the window raster due to model clipping.

▼ clip_modclip_line_pttypes_3d

Test Types:	INDEX, SM
Description:	Model clipping in 3D of a thin solid line using all the permitted point types

Attributes Tested:	`XGL_3D_CTX_MODEL_CLIP_PLANE_NUM`
	`XGL_3D_CTX_MODEL_CLIP_PLANES`
	`XGL_CTX_LINE_STYLE`
	`XGL_LINE_SOLID`

Operators Tested:	`xgl_multipolyline` `xgl_object_set`
Output:	Renders the same line segments using four different point types. The line segments correct appearance should look like a lopsided “x” on its side with the small portion closest to the right side of the window raster

▼ clip_modclip_pg_3d

Test Types:	INDEX, SM
Description:	Model clipping in 3D with the replacement and intersection of the clip volume of `F3D` solid-filled polygons with and without edges, concave, and multibounded; of `F3D` solid-filled polygons with various edge styles

Attributes Tested:	`XGL_CTX_EDGE_COLOR`
	`XGL_CTX_BACKGROUND_COLOR`
	`XGL_CTX_SURF_EDGE_FLAG`
	`XGL_3D_CTX_MODEL_CLIP_PLANE_NUM`

	XGL_3D_CTX_MODEL_CLIP_PLANES
	XGL_LINE_ALT_PATTERNED
	XGL_CTX_EDGE_STYLE
	XGL_LINE_PATTERNED
	XGL_CTX_EDGE_PATTERN
Operators Tested:	xgl_polygon
	xgl_object_set
	xgl_object_get
Output:	Renders three polygons but the correct view on the screen should be a figure that looks like an hourglass on its side

▼ clip_modclip_pg_3d_1

Test Types:	INDEX, SM
Description:	Model clipping in 3D of F3D polygons of all possible fill styles with edges, concave, and multibounded
Attributes Tested:	XGL_CTX_SURF_FPAT
	XGL_CTX_SURF_EDGE_FLAG
	XGL_3D_CTX_MODEL_CLIP_PLANE_NUM
	XGL_3D_CTX_MODEL_CLIP_PLANES
	XGL_CTX_EDGE_COLOR
	XGL_CTX_SURF_FRONT_FILL_STYLE
	XGL_SURF_FILL_EMPTY
	XGL_SURF_FILL_HOLLOW
	XGL_SURF_FILL_OPAQUE_STIPPLE
Operators Tested:	xgl_polygon
	xgl_object_set
Output:	Renders three polygons but the correct view is a figure similar to a telephone handle and a block with two slightly curved edges

▼ clip_modclip_qm

Test Types:	INDEX, SM
Description:	Model clipping of quadmeshes (only solid filled without edges)

Attributes Tested:	XGL_CTX_SURF_EDGE_FLAG XGL_3D_CTX_MODEL_CLIP_PLANE_NUM XGL_3D_CTX_MODEL_CLIP_PLANES XGL_CTX_SURF_FRONT_FILL_STYLE XGL_SURF_FILL_SOLID
Operators Tested:	xgl_quadrilateral_mesh xgl_object_set
Output:	Renders two quadmeshes, but the correct view is two quads connected that appear as one quad with a full width but unusually low height.

▼ clip_modclip_qm_1

Test Types:	INDEX, SM
Description:	Model clipping of quadmeshes of all possible fill styles, and edges wide and thin
Attributes Tested:	XGL_CTX_SURF_FPAT XGL_3D_CTX_MODEL_CLIP_PLANE_NUM XGL_CTX_EDGE_COLOR XGL_CTX_SURF_EDGE_FLAG XGL_CTX_EDGE_WIDTH_SCALE_FACTOR XGL_CTX_SURF_FRONT_FILL_STYLE XGL_SURF_FILL_EMPTY XGL_SURF_FILL_SOLID XGL_SURF_FILL_HOLLOW XGL_SURF_FILL_OPAQUE_STIPPLE
Operators Tested:	xgl_quadrilateral_mesh xgl_object_set xgl_object_get
Output:	Renders two quadmeshes, but the correct view is two quads connected that appear as one quad with a full width but unusually low height.

▼ clip_modclip_stext_3d

Test Types:	INDEX, SM
Description:	Model clipping in 3D of stroke text
Attributes Tested:	XGL_3D_CTX_MODEL_CLIP_PLANE_NUM XGL_3D_CTX_MODEL_CLIP_PLANES XGL_CTX_STEXT_ALIGN_HORIZ

	XGL_STEXT_ALIGN_HORIZ_CENTER
	XGL_CTX_STEXT_ALIGN_VERT
	XGL_STEXT_ALIGN_VERT_HALF
Operators Tested:	xgl_stroke_text
	xgl_object_set
Output:	Although five huge "X"s which would normally form a cross are put on the screen for rendering, model clipping removes all but one.

▼ clip_modclip_ts

Test Types:	INDEX, SM
Description:	Model clipping of solid filled tristrips
Attributes Tested:	XGL_3D_CTX_MODEL_CLIP_PLANE_NUM
	XGL_3D_CTX_MODEL_CLIP_PLANES
Operators Tested:	xgl_triangle_strip
	xgl_object_set
Output:	Renders two tristrips and the correct view is one *quad* with a full width but unusually low height.

▼ clip_modclip_ts_1

Test Types:	INDEX, SM
Description:	Model clipping of tristrips of all possible fill styles, and thin and wide edges
Attributes Tested:	XGL_CTX_SURF_FPAT
	XGL_3D_CTX_MODEL_CLIP_PLANE_NUM
	XGL_3D_CTX_MODEL_CLIP_PLANES
	XGL_CTX_EDGE_COLOR
	XGL_CTX_SURF_EDGE_FLAG
	XGL_CTX_EDGE_WIDTH_SCALE_FACTOR
	XGL_CTX_SURF_FRONT_FILL_STYLE
	XGL_SURF_FILL_EMPTY
	XGL_SURF_FILL_SOLID
	XGL_SURF_FILL_HOLLOW
	XGL_SURF_FILL_OPAQUE_STIPPLE
Operators Tested:	xgl_triangle_strip
	xgl_object_set
	xgl_object_get

Output: Renders two tristrips and the correct view is one *quad* with a full width but unusually low height.

▼ clip_rasrect

Test Types: RGB, SM

Description: Tests DC clipping for `xgl_context_new_frame()` by specifying `XGL_RAS_RECT_NUM`.

Attributes Tested: `XGL_RAS_RECT_NUM`
`XGL_RAS_RECT_LIST`
`XGL_CTX_DEVICE`
`XGL_CTX_BACKGROUND_COLOR`

Operators Tested: `xgl_context_new_frame`
`xgl_object_set`

Output: A green right canvas, an entire black canvas, a yellow right canvas, a yellow right and a green left canvas, and an entire black canvas repeated for 2D and 3D contexts.

▼ clip_rasrect_1

Test Types: RGB, SM

Description: Tests DC clipping of a primitive by specifying `XGL_RAS_RECT_NUM`.

Attributes Tested: `XGL_RAS_RECT_NUM`
`XGL_RAS_RECT_LIST`
`XGL_CTX_DEVICE`
`XGL_CTX_SURF_FRONT_COLOR`
`XGL_3D_CTX_HLHSR_MODE`

Operators Tested: `xgl_multi_simple_polygon`
`xgl_object_set`

Output: A yellow rectangle in the left part of the canvas, a yellow rectangle in the right part of the canvas, a green rectangle on the left, and a red rectangle in the entire canvas repeated for 2D and 3D contexts, with HLHSR on for 3D context.

clip_rasrect_2

Test Types: RGB, SM
Description: Tests DC clipping of lines by specifying XGL_RAS_RECT_NUM for 2D and 3D context with HLHSR on or off.
Attributes Tested: XGL_RAS_RECT_NUM
XGL_RAS_RECT_LIST
Operators Tested: xgl_object_set
xgl_object_get
xgl_multipolyline
xgl_context_new_frame
Output: Three yellow lines, three green lines, then four red lines forming a rectangle repeated for 2D and 3D contexts, with HLHSR on for 3D context.

win_clip_cpbuf

Test Types: RGB, SM
Description: Creates one huge XGL/X window raster which entirely covers the screen. Then 15 different XGL/X window rasters which in total exhibit all possible 15 combinations of window screen clipping are individually created and destroyed. An *xgl_multipolyline* call places lines outlining the interior square boundary of the window raster and the diagonals from one upper corner to the opposite bottom lower corner is placed into a memory raster. This memory raster is then copied to an XGL/X denizen window raster already on the screen. The size, width and height and the depth of the memory raster is the same as the existing mapped XGL/X denizen window raster.
Attributes Tested: XGL_CTX_LINE_COLOR
XGL_CTX_DEVICE
XGL_DEV_COLOR_TYPE
XGL_WIN_RAS
XGL_MEM_RAS
XGL_RAS_DEPTH
XGL_RAS_WIDTH
XGL_RAS_HEIGHT

Operators Tested:	`xgl_open`
	`xgl_object_create`
	`xgl_multipolyline`
	`xgl_context_post`
	`xgl_object_destroy`
	`xgl_object_get`
	`xgl_context_copy_buffer`

Output:	The entire screen is "blanked" out with the color of an XGL/X denizen window raster and then 15 different XGL/X window rasters appear and depart. The 15 combinations of window screen XGL/X denizen window raster clipping are: (1) clip to right side of screen (2) clip to left side of screen (3) clip to top of screen (4) clip to bottom of screen (5) clip to left and right of screen (6) clip to left and top of screen (7) clip to left and bottom of screen (8) clip to right and top of screen (9) clip to right and bottom of screen (10) clip to top and bottom of screen (11) clip to left, right and top of screen (12) clip to right, top and bottom of screen (13) clip to left, top and bottom of screen (14) clip to bottom, left and right of screen (15) clip to all sides of screen The output should be exactly the same as *win_clip_ln*. There should be a minimal delay perhaps visible for the action of the copy from the memory raster to the window raster.

win_clip_ln

Test Types: RGB, SM

Description: Creates one huge XGL/X window raster which entirely covers the screen. Then 15 different XGL/X window rasters which in total exhibit all possible 15 combinations of window screen clipping are individually created and destroyed. An *xgl_multipolyline* call places lines outlining the interior square boundary of the window raster and the diagonals from one upper corner to the opposite bottom lower corner.

Attributes Tested:
`XGL_CTX_LINE_COLOR`
`XGL_CTX_DEVICE`
`XGL_DEV_COLOR_TYPE`
`XGL_WIN_RAS`

Operators Tested:
`xgl_open`
`xgl_object_create`
`xgl_multipolyline`
`xgl_context_post`
`xgl_object_destroy`
`xgl_object_get`

Output: The entire screen is "blanked" out with the color of an XGL/X denizen window raster and then 15 different XGL/X window rasters appear and depart. The 15 combinations of window screen XGL/X denizen window raster clipping are:
(1) clip to right side of screen
(2) clip to left side of screen
(3) clip to top of screen
(4) clip to bottom of screen
(5) clip to left and right of screen
(6) clip to left and top of screen
(7) clip to left and bottom of screen
(8) clip to right and top of screen
(9) clip to right and bottom of screen
(10) clip to top and bottom of screen
(11) clip to left, right and top of screen
(12) clip to right, top and bottom of screen
(13) clip to left, top and bottom of screen
(14) clip to bottom, left and right of screen
(15) clip to all sides of screen

▼ win_clip_mk

Test Types: RGB, SM

Description: Creates one huge XGL/X window raster which entirely covers the screen. Then 15 different XGL/X window rasters which in total exhibit all possible 15 combinations of window screen clipping are individually created and destroyed. An *xgl_multimarker* call places four clipped markers at the strategic screen clipping boundary. The marker color used is the default which is green.

Attributes Tested:
`XGL_CTX_MARKER_COLOR`
`XGL_CTX_DEVICE`
`XGL_DEV_COLOR_TYPE`
`XGL_WIN_RAS`
`XGL_CTX_MARKER_SCALE_FACTOR`

Operators Tested:
`xgl_open`
`xgl_object_create`
`xgl_multimarker`
`xgl_context_post`
`xgl_object_destroy`
`xgl_object_get`
`xgl_object_set`

Output: The entire screen is "blanked" out with the color of an XGL/X denizen window raster and then 15 different XGL/X window rasters appear and depart. The outer loop changes the marker rendered (1) plus, (2) asterisk, (3) circle, (4) cross, (5) square, (6) bowtie_ne and (7) bowtie_nw while the inner loop goes thru the 15 combinations of window screen XGL/X denizen window raster clipping:
(1) clip to right side of screen
(2) clip to left side of screen
(3) clip to top of screen
(4) clip to bottom of screen
(5) clip to left and right of screen
(6) clip to left and top of screen
(7) clip to left and bottom of screen
(8) clip to right and top of screen
(9) clip to right and bottom of screen
(10) clip to top and bottom of screen
(11) clip to left, right and top of screen

(12) clip to right, top and bottom of screen
(13) clip to left, top and bottom of screen
(14) clip to bottom, left and right of screen
(15) clip to all sides of screen

▼ win_clip_msp

Test Types: RGB, SM

Description: Creates one huge XGL/X window raster which entirely covers the screen. Then 15 different XGL/X window rasters which in total exhibit all possible 15 combinations of window screen clipping are individually created and destroyed. An *xgl_multi_simple_polygon* call places polygons at the strategic screen clipping boundary. The surface color used is the default which is green. The flags used to render the polygon are the combination of `XGL_FACET_FLAG_SIDES_UNSPECIFIED`, `XGL_FACET_FLAG_SHAPE_UNKNOWN`, `XGL_FACET_FLAG_FN_CONSISTENT`, `XGL_FACET_FLAG_DATA_NOT_CONTIG` and the facet type is `NONE`.

Attributes Tested:
`XGL_CTX_MARKER_COLOR`
`XGL_CTX_DEVICE`
`XGL_DEV_COLOR_TYPE`
`XGL_WIN_RAS`
`XGL_CTX_SURF_FRONT_COLOR`

Operators Tested:
`xgl_open`
`xgl_object_create`
`xgl_multi_simple_polygon`
`xgl_context_post`
`xgl_object_destroy`
`xgl_object_get`
`xgl_object_set`

Output: The entire screen is "blanked" out with the color of an XGL/X denizen window raster and then 15 different XGL/X window rasters appear and depart. The 15 combinations of window screen XGL/X denizen window raster clipping are:
(1) clip to right side of screen
(2) clip to left side of screen

(3) clip to top of screen
(4) clip to bottom of screen
(5) clip to left and right of screen
(6) clip to left and top of screen
(7) clip to left and bottom of screen
(8) clip to right and top of screen
(9) clip to right and bottom of screen
(10) clip to top and bottom of screen
(11) clip to left, right and top of screen
(12) clip to right, top and bottom of screen
(13) clip to left, top and bottom of screen
(14) clip to bottom, left and right of screen
(15) clip to all sides of screen
The green polygon is square shaped and when strategic clipping locations are spacially very distant from one another like left side of screen in combination with right side of screen, two square shaped polygons will be rendered.

▼ win_clip_pgon

Test Types: RGB, SM

Description: Creates one huge XGL/X window raster which entirely covers the screen. Then 15 different XGL/X window rasters which in total exhibit all possible 15 combinations of window screen clipping are individually created and destroyed. An *xgl_polygon* call places square polygons at the strategic screen clipping boundary. The surface color used is the default - green.

Attributes Tested:
`XGL_CTX_SURF_FRONT_COLOR`
`XGL_CTX_DEVICE`
`XGL_DEV_COLOR_TYPE`
`XGL_WIN_RAS`
`XGL_SYS_ST_ERROR_DETECTION`

Operators Tested:
`xgl_open`
`xgl_object_create`
`xgl_polygon`
`xgl_context_post`

	`xgl_object_destroy`
	`xgl_object_get`
	`xgl_object_set`

Output:	The entire screen is "blanked" out with the color of an XGL/X denizen window raster and then 15 different XGL/X window rasters appear and depart. The 15 combinations of window screen XGL/X denizen window raster clipping are: (1) clip to right side of screen (2) clip to left side of screen (3) clip to top of screen (4) clip to bottom of screen (5) clip to left and right of screen (6) clip to left and top of screen (7) clip to left and bottom of screen (8) clip to right and top of screen (9) clip to right and bottom of screen (10) clip to top and bottom of screen (11) clip to left, right and top of screen (12) clip to right, top and bottom of screen (13) clip to left, top and bottom of screen (14) clip to bottom, left and right of screen (15) clip to all sides of screen The green polygon is square shaped and when strategic clipping locations are spacially very distant from one another like left side of screen in combination with right side of screen, two square shaped polygons will be rendered.

▼ win_clip_qm

Test Types:	RGB, SM
Description:	Creates one huge XGL/X window raster which entirely covers the screen. Then 15 different XGL/X window rasters which in total exhibit all possible 15 combinations of window screen clipping are individually created and destroyed. An *xgl_quadrilateral_mesh* call places quads at strategic screen clipping boundary. The surface color used is the default which is green.

Attributes Tested: `XGL_SYS_ST_ERROR_DETECTION`
`XGL_CTX_DEVICE`
`XGL_DEV_COLOR_TYPE`
`XGL_WIN_RAS`
`XGL_CTX_SURF_FRONT_COLOR`

Operators Tested: `xgl_open`
`xgl_object_create`
`xgl_quadrilateral_mesh`
`xgl_context_post`
`xgl_object_destroy`
`xgl_object_get`
`xgl_object_set`

Output: The entire screen is "blanked" out with the color of an XGL/X denizen window raster and then 15 different XGL/X window rasters appear and depart. The 15 combinations of window screen XGL/X denizen window raster clipping are:
(1) clip to right side of screen
(2) clip to left side of screen
(3) clip to top of screen
(4) clip to bottom of screen
(5) clip to left and right of screen
(6) clip to left and top of screen
(7) clip to left and bottom of screen
(8) clip to right and top of screen
(9) clip to right and bottom of screen
(10) clip to top and bottom of screen
(11) clip to left, right and top of screen
(12) clip to right, top and bottom of screen
(13) clip to left, top and bottom of screen
(14) clip to bottom, left and right of screen
(15) clip to all sides of screen
The green quadmesh appears almost square and two may be created dependent on the strategic clipping locations spatial distance such as left side of screen in combination with right side of screen.

win_clip_tlst

Test Types: RGB, SM

Description: Creates one huge XGL/X window raster which entirely covers the screen. Then the three different mutually exclusive flag types are individually selected to do the next 15 different XGL/X window rasters which in total exhibit all possible 15 combinations of window screen clipping are individually created and destroyed. An *xgl_triangle_list* call places the chosen flag type of triangle at the strategic screen clipping boundary. The surface color used is the default which is green. The order in which the flag types are chosen are: `XGL_TLIST_FLAG_TRI_STAR`, `XGL_TLIST_FLAG_TRI_STRIP` and `XGL_TLIST_FLAG_TRI_RESTART`.

Attributes Tested:
`XGL_SYS_ST_ERROR_DETECTION`
`XGL_CTX_DEVICE`
`XGL_DEV_COLOR_TYPE`
`XGL_WIN_RAS`
`XGL_CTX_SURF_FRONT_COLOR`

Operators Tested:
`xgl_open`
`xgl_object_create`
`xgl_triangle_list`
`xgl_context_post`
`xgl_object_destroy`
`xgl_object_get`
`xgl_object_set`

Output: The entire screen is "blanked" out with the color of an XGL/X denizen window raster and then 15 different XGL/X window rasters appear and depart. The 15 combinations of window screen XGL/X denizen window raster clipping are:
(1) clip to right side of screen
(2) clip to left side of screen
(3) clip to top of screen
(4) clip to bottom of screen
(5) clip to left and right of screen
(6) clip to left and top of screen
(7) clip to left and bottom of screen
(8) clip to right and top of screen

(9) clip to right and bottom of screen
(10) clip to top and bottom of screen
(11) clip to left, right and top of screen
(12) clip to right, top and bottom of screen
(13) clip to left, top and bottom of screen
(14) clip to bottom, left and right of screen
(15) clip to all sides of screen
The shape of the triangles rendered by the *xgl_triangle_list* call may appear as the capital letter "K" shape, square or triangles. When the clipping positions are spatially quite distant such as clip to left and right side of screen, two *xgl_triangle_list* calls will place the primitive at strategic locations.

▼ win_clip_ts

Test Types: RGB, SM

Description: Creates one huge XGL/X window raster which entirely covers the screen. Then 15 different XGL/X window rasters which in total exhibit all possible 15 combinations of window screen clipping are individually created and destroyed. An *xgl_triangle_strip* call places tristrips at the strategic screen clipping boundary. The surface color used is the default which is green.

Attributes Tested:
`XGL_SYS_ST_ERROR_DETECTION`
`XGL_CTX_DEVICE`
`XGL_DEV_COLOR_TYPE`
`XGL_WIN_RAS`
`XGL_MEM_RAS`
`XGL_RAS_DEPTH`
`XGL_RAS_WIDTH`
`XGL_RAS_HEIGHT`
`XGL_CTX_SURF_FRONT_COLOR`

Operators Tested:
`xgl_open`
`xgl_object_create`
`xgl_triangle_strip`
`xgl_context_post`
`xgl_object_destroy`
`xgl_object_get`
`xgl_object_set`

Output: The entire screen is "blanked" out with the color of an XGL/X denizen window raster and then 15 different XGL/X window rasters appear and depart. The 15 combinations of window screen XGL/X denizen window raster clipping are:

(1) clip to right side of screen
(2) clip to left side of screen
(3) clip to top of screen
(4) clip to bottom of screen
(5) clip to left and right of screen
(6) clip to left and top of screen
(7) clip to left and bottom of screen
(8) clip to right and top of screen
(9) clip to right and bottom of screen
(10) clip to top and bottom of screen
(11) clip to left, right and top of screen
(12) clip to right, top and bottom of screen
(13) clip to left, top and bottom of screen
(14) clip to bottom, left and right of screen
(15) clip to all sides of screen

The green tristrip is usually two faceted and so appears as a square although it may also be a single triangle when strategic clipping locations are very far like left side of screen in combination with right side of screen.

Table 10-1 Clipping Combinations

2D Tests Loop Clipping Combinations		**3D Tests Loop Clipping Combinations**	
i	CLIP_PLANES	i	CLIP_PLANES
0	None	0	None
1	XMIN	1	XMIN
2	XMAX	2	XMAX
3	XMIN | XMAX	3	XMIN | XMAX
4	YMIN	4	YMIN
5	XMIN | YMIN	5	XMIN | YMIN
6	XMAX | YMIN	6	XMAX | YMIN
7	XMIN | XMAX | YMIN	7	XMIN | XMAX | YMIN
8	YMAX	8	YMAX
9	XMIN | YMAX	9	XMIN | YMAX
10	XMAX | YMAX	10	XMAX | YMAX
11	XMIN | XMAX | YMAX	11	XMIN | XMAX | YMAX
12	YMIN | YMAX	12	YMIN | YMAX
13	XMIN | YMIN | YMAX	13	XMIN | YMIN | YMAX
14	XMAX | YMIN | YMAX	14	XMAX | YMIN | YMAX
15	XMIN | XMAX | YMIN | YMAX	15	XMIN | XMAX | YMIN | YMAX
		16	ZMIN
		17	XMIN | ZMIN
		18	XMAX | ZMIN
		19	XMIN | XMAX | ZMIN
		20	YMIN | ZMIN
		21	XMIN | YMIN | ZMIN
		22	XMAX | YMIN | ZMIN
		23	XMIN | XMAX | YMIN | ZMIN

Table 10-1 Clipping Combinations *(Continued)*

2D Tests Loop Clipping Combinations		**3D Tests Loop Clipping Combinations**	
		24	YMAX | ZMIN
		25	XMIN | YMAX | ZMIN
		26	XMAX | YMAX | ZMIN
		27	XMIN | XMAX | YMAX | ZMIN
		28	YMIN | YMAX | ZMIN
		29	XMIN | YMIN | YMAX | ZMIN
		30	XMAX | YMIN | YMAX | ZMIN
		31	XMIN | XMAX | YMIN | YMAX | ZMIN
		32	ZMAX
		33	XMIN | ZMAX
		34	XMAX | ZMAX
		35	XMIN | XMAX | ZMAX
		36	YMIN | ZMAX
		37	XMIN | YMIN | ZMAX
		38	XMAX | YMIN | ZMAX
		39	XMIN | XMAX | YMIN | ZMAX
		40	YMAX | ZMAX
		41	XMIN | YMAX | ZMAX
		42	XMAX | YMAX | ZMAX
		43	XMIN | XMAX | YMAX | ZMAX
		44	YMIN | YMAX | ZMAX
		45	XMIN | YMIN | YMAX | ZMAX
		46	XMAX | YMIN | YMAX | ZMAX
		47	XMIN | XMAX | YMIN | YMAX | ZMAX
		48	ZMIN | ZMAX

Table 10-1 Clipping Combinations *(Continued)*

2D Tests	Loop Clipping Combinations	3D Tests	Loop Clipping Combinations
		49	XMIN | ZMIN | ZMAX
		50	XMAX | ZMIN | ZMAX
		51	XMIN | XMAX | ZMIN | ZMAX
		52	YMIN | ZMIN | ZMAX
		53	XMIN | YMIN | ZMIN | ZMAX
		54	XMAX | YMIN | ZMIN | ZMAX
		55	XMIN | XMAX | YMIN | ZMIN | ZMAX
		56	YMAX | ZMIN | ZMAX
		57	XMIN | YMAX | ZMIN | ZMAX
		58	XMAX | YMAX | ZMIN | ZMAX
		59	XMIN | XMAX | YMAX | ZMIN | ZMAX
		60	YMIN | YMAX | ZMIN | ZMAX
		61	XMIN | YMIN | YMAX | ZMIN | ZMAX
		62	XMAX | YMIN | YMAX | ZMIN | ZMAX
		63	XMIN | XMAX | YMIN | YMAX | ZMIN | ZMAX

Colormap Test Descriptions 11

This chapter describes the Colormap test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section, “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ colormap0

Test Types:	INDEX, SM
Description:	Creates a colormap, attaches to a raster and draws a circle. Sets background colors and verifies colors in map.
Attributes Tested:	See Table 11-1, Column A at the end of this chapter.
Operators Tested:	`xgl_multicircle`
Output:	First the background changes from black through grays to white, then a circle appears that changes color to white as it is continuously scaled larger.

▼ colormap1

Test Types:	INDEX, SM
Description:	Creates colormaps, attaches to a raster, and draws a circle, line, and rectangle. Interchanges maps, draws, and verifies that correct images are drawn.
Attributes Tested:	XGL_CTX_LINE_COLOR XGL_CTX_NURBS_CURVE_APPROX XGL_CTX_NURBS_CURVE_APPROX_VAL XGL_CTX_SURF_EDGE_FLAG XGL_CTX_SURF_FRONT_COLOR XGL_CURVE_METRIC_VDC XGL_PT_I2D
Operators Tested:	xgl_multicircle xgl_multipolyline xgl_multirectangle
Output:	A circle, a line, and then a rectangle. Renders first time with red background, and each element grows as its color changes. The second sequence has a blue background, and again the primitives change color as they expand.

▼ colormap2

Test Types:	INDEX, SM
Description:	Creates colormaps, attaches to a raster, and draws a circle, line, and rectangle. Verifies background colors. Clears screen and verifies the cleared image.
Attributes Tested:	XGL_CTX_LINE_COLOR XGL_DEV_COLOR_MAP XGL_MULTIRECT_I2D XGL_PT_I2D and Table 11-1, Column A at the end of this chapter
Operators Tested:	xgl_multicircle xgl_multipolyline xgl_multirectangle
Output:	First a circle, then a line, and finally a rectangle. Renders each on top of the previous. Each expands as its color changes from black to white.

▼ colormap3

Test Types:	INDEX, SM
Description:	Creates colormaps, attaches one to a raster, and draws a circle, line, and rectangle. Creates / destroys to check for memory leaks. Pushes context, destroys the colormap, and reattaches new map and looks for results. Creates another raster, switches colormaps, and verifies proper attachment. Checks compatibility of the colormap size.

Attributes Tested:	`XGL_CTX_LINE_COLOR`
	`XGL_DEV_COLOR_MAP`
	`XGL_MULTIRECT_I2D`
	`XGL_PT_I2D`
	and Table 11-1, Column A at the end of this chapter

Operators Tested:	`xgl_multicircle`
	`xgl_multipolyline`
	`xgl_multirectangle`

Output:	A circle, a line, and then a rectangle

▼ colormap4

Test Types:	INDEX, SM
Description:	Verifies that the colormap can be any size within the maximum and also checks for maximum colormap size.

Attributes Tested:	`XGL_CMAP_COLOR_TABLE_SIZE`
	`XGL_CMAP_MAX_COLOR_TABLE_SIZE`
	`XGL_COLOR_INDEX`
	`XGL_CTX_BACKGROUND_COLOR`
	`XGL_RAS_DEPTH`

Operators Tested:	`xgl_object_set` `xgl_object_get`
Output:	None

▼ colormap5

Test Types:	RGB, SM
Description:	RGB line drawing

Attributes Tested:	`XGL_COLOR_RGB`
	`XGL_CTX_BACKGROUND_COLOR`
	`XGL_CTX_LINE_COLOR`
	`XGL_PT_I2D`

Operators Tested:	`xgl_multipolyline`
Output:	Green line on a red background

▼ colormap6

Test Types:	INDEX, SM
Description:	Like *colormap0*, but uses window system colormap; tests `XGL_CMAP_NAME` and pixel mapping array

Attributes Tested:	`XGL_CMAP_NAME`
	`XGL_COLOR_INDEX`
	`XGL_WIN_RAS_DESCRIPTOR`
	and Table 11-1, Column A at the end of this chapter

Operators Tested:	`xgl_multicircle`
Output:	Renders multiple frames of an expanding circle that changes color

▼ cmap_ramp

Test Types:	CM, INDEX
Description:	Indexes the test for color ramps. Four 16-element ramps are built; first red, then green, blue, and grayscale. Then renders four polygons, one per ramp with different z (depth) values, and depth cueing is turned on.
Attributes Tested:	See Table 11-1, Column B at the end of this chapter.
Operators Tested:	`xgl_polygon`
Output:	Draws four polygons of varying shapes and colors. Polygons are depth cued to show their color ramps.

▼ xcolor_mapping

Test Types:	CM, INDEX
Description:	Creates an X colormap with 64 entries and attaches to a raster with `WIN_RAS_PIXEL_MAPPING` and `CMAP_NAME`. This is appropriate only for Pseudocolor (8-bit index) visuals, so the test aborts if it can't get the right visual.
Attributes Tested:	See Table 11-1, Column C at the end of this chapter.
Operators Tested:	`xgl_polygon`
Output:	Draws 64 polygons in eight rows of eight, using the consecutive indexes of the X colormap, used by setting Xgl_color.index(es)

▼ cmapper

Test Types:	SM, INDEX
Description:	Creates an X colormap with eight entries and attaches to an RGB raster with `WIN_RAS_PIXEL_MAPPING`, then uses it with `COLOR_MAPPER` to return the index to the color table, given an RGB style color value (xgl_color.rgb.r, for example). This is appropriate only for Pseudocolor (8-bit index) visuals, so it will exit if it can't get the right visual.

Attributes Tested:	`XGL_CMAP_COLOR_MAPPER`
	`XGL_CMAP_MAX_COLOR_TABLE_SIZE`
	`XGL_CMAP_NAME`
	`XGL_CTX_SURF_FRONT_COLOR`
	`XGL_PT_I2D`
	`XGL_WIN_RAS_PIXEL_MAPPING`

Operators Tested:	`xgl_polygon`
Output:	Draws eight rows of eight polygons using the indexes from the X colormap; our cmapper function reverses the indexes. The background is yellow with various color polygons.

Table 11-1 Colormap Attributes Tested

Column A	Column B	Column C
XGL_CMAP_COLOR_TABLE	XGL_CMAP_NAME	XGL_CMAP_MAX_COLOR_TABLE_SIZE
XGL_CMAP_COLOR_TABLE_SIZE	XGL_3D_CTX_DEPTH_CUE_MODE	XGL_CMAP_NAME
XGL_CTX_BACKGROUND_COLOR	XGL_DEPTH_CUE_LINEAR	XGL_COLOR_INDEX
XGL_CTX_NURBS_CURVE_APPROX	XGL_3D_CTX_SURF_FACE_DISTINGUISH	XGL_CTX_DEFERRAL_MODE
XGL_CTX_NURBS_CURVE_APPROX_VAL	XGL_CMAP_COLOR_TABLE	XGL_CTX_SURF_FRONT_COLOR
XGL_CTX_SURF_EDGE_FLAG	XGL_CMAP_COLOR_TABLE_SIZE	XGL_DEFER_ASAP
XGL_CTX_SURF_FRONT_COLOR	XGL_CMAP_RAMP_LIST	XGL_DEV_COLOR_MAP
XGL_CURVE_METRIC_VDC	XGL_CMAP_RAMP_NUM	XGL_FACET_NONE
XGL_MULTICIRCLE_I2D	XGL_CTX_SURF_FRONT_COLOR	XGL_PT_I2D
	XGL_CTX_VDC_MAP	XGL_DEV_COLOR_TYPE
	XGL_PT_F3D	XGL_WIN_RAS_PIXEL_MAPPING
	XGL_VDC_MAP_ASPECT	

Context Test Descriptions 12

This chapter describes the Context test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section, “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

context_2d_create

Test Types:	INDEX, SM
Description:	Tests that several context objects can be created
Attributes Tested:	`XGL_CTX_MARKER`
Operators Tested:	`xgl_multimarker`
Output:	2D marker displayed using the 19th context created

▼ context_2d_error

Test Types:	RGB or INDEX, CM
Description:	Sets erroneous values for all 2D attributes to ensure correct error detection.
Attributes Tested:	All 2D context attributes
Operators Tested:	None
Output:	Error messages

▼ context_3d_error

Test Types:	RGB or INDEX, CM
Description:	Sets erroneous values for all 3D attributes to ensure correct error detection.
Attributes Tested:	All 3D context attributes
Operators Tested:	None
Output:	Error messages

▼ context_2d_pat_line

Test Types:	INDEX, SM
Description:	Creates a 2D context, attaches to a raster and draws 2D lines using different line patterns.
Attributes Tested:	See Table 12-1, Column A at the end of this chapter.
Operators Tested:	`xgl_multipolyline`
Output:	Patterned lines are drawn.

▼ context_2d_pat_line_rgb

Test Types:	RGB, SM
Description:	Creates a 2D context, attaches to a raster and draws 2D lines using different line patterns.
Attributes Tested:	See Table 12-1, Column A at the end of this chapter.
Operators Tested:	`xgl_multipolyline`
Output:	Patterned lines are drawn.

▼ context_2d_pp

Test Types:	INDEX, SM
Description:	Creates contexts, attaches to a raster and draws a circle, line, and rectangle. Pushes/pops to check for memory leaks. Pushes context, destroys it, and checks the pop operation for results.
Attributes Tested:	See Table 12-1, Column B at the end of this chapter.

Operators Tested:	`xgl_multicircle`
	`xgl_multipolyline`
	`xgl_multirectangle`

Output:	A circle, line, and rectangle are drawn.

▼ context_2d_pp_all_attrs

Test Types:	INDEX, SM
Description:	Context pushes/pops all pushable 2D context attributes by calling `xgl_context_push()` with a NULL attribute list. Makes sure all attribute values are pushed and popped correctly using `xgl_object_get()`.
Attributes Tested:	See Table 12-2 at the end of this chapter.

Operators Tested:	`xgl_context_pop`
	`xgl_context_push`
	`xgl_multiarc`
	`xgl_multimarker`
	`xgl_multipolyline`
	`xgl_nu_bspline_curve`
	`xgl_stroke_text`

Output:	A line, marker and arc are displayed.

▼ context_2d_pp_pat_line

Test Types:	INDEX, SM
Description:	Creates contexts, attaches to a raster and draws lines. Pushes/pops/post contexts and verifies correct values recovered when popped.
Attributes Tested:	See Table 12-1, Column C at the end of this chapter.

Operators Tested:	`xgl_context_pop`
	`xgl_context_push`
	`xgl_multipolyline`

Output:	Sets a line pattern, draws a patterned line, then pushes context. Sets another line pattern, draws another line, and then pops. Draws a line again without setting another line pattern.

▼ context_2d_pp_pat_line_rgb

Test Types:	RGB, SM
Description:	Creates contexts, attaches to an RGB raster, and draws lines. Pushes/pops/posts contexts and verifies correct values recovered when popped.
Attributes Tested:	See Table 12-1, Column C at the end of this chapter.

Operators Tested:	`xgl_context_pop`
	`xgl_context_push`
	`xgl_multipolyline`

Output:	Sets a line pattern, draws a patterned line, then pushes context, sets another line pattern, draws another line, and then pops. Draws a line again without setting another line pattern.

▼ context_2d_pp_rgb

Test Types:	RGB, SM
Description:	Creates contexts, attaches to a raster and draws a circle, line, and rectangle. Pushes/pops to check for memory leaks. Pushes context, destroys it, and checks pop operation for results.
Attributes Tested:	See Table 12-1, Column B at the end of this chapter.

Operators Tested:	`xgl_context_pop`
	`xgl_context_push`
	`xgl_multicircle`
	`xgl_multipolyline`
	`xgl_multirectangle`

Output:	A circle, line and rectangle are drawn.

▼ context_2d_pu_non_null_attrs:

Test Types:	INDEX, SM

Description:	Context pushes/pops all pushable 2D context attributes by calling `xgl_context_push()` with a non-NULL attribute list. Makes sure all attribute values are pushed and popped correctly using `xgl_object_get()`.
Attributes Tested:	See Table 12-2, at the end of this chapter.

Operators Tested:	`xgl_multiarc`
	`xgl_multimarker`
	`xgl_multipolyline`
	`xgl_nu_bspline_curve`
	`xgl_stroke_text`

Output:	A line, marker and arc are displayed.

▼ context_2d_set_get_pixel

Test Types:	INDEX, SM
Description:	Creates a context, sets pixels and gets them back.
Attributes Tested:	Default
Operators Tested:	`xgl_context_get_pixel` `xgl_context_set_pixel`
Output:	A region is drawn using `xgl_context_set_pixel()`.

▼ context_2d_set_get_pixel_rgb

Test Types:	RGB, SM
Description:	Creates a context, sets pixels and gets them back.
Attributes Tested:	Default
Operators Tested:	`xgl_context_get_pixel` `xgl_context_set_pixel`
Output:	A region is drawn using `xgl_context_set_pixel()`.

▼ context_2d_simple

Test Types:	INDEX, SM
Description:	Creates contexts, attaches one to a raster and draws rectangles. Switches a context, sets/gets a pixel value and verifies correct value.
Attributes Tested:	See Table 12-4, Column A at the end of this chapter.
Operators Tested:	`xgl_context_get_pixel` `xgl_multirectangle`

Output:	A red rectangle is drawn.

▼ context_2d_simple_env_attrs

Test Types:	INDEX, SM
Description:	Checks that environmental context attributes (deferral mode, device and threshold) can be set and retrieved properly
Attributes Tested:	See Table 12-4, Column B at the end of this chapter.
Operators Tested:	`xgl_object_get` `xgl_object_set`
Output:	Nothing is displayed.

▼ context_dc_offset

Test Types:	RGB, CM
Description:	Tests the `XGL_3D_CTX_SURF_DC_OFFSET` attribute.
Attributes Tested:	`XGL_3D_CTX_SURF_DC_OFFSET`
Operators Tested:	`xgl_quadrilateral_mesh` `xgl_multipolyline`
Output:	Quadmeshed sphere and polylines

▼ context_env_attrs_rgb

Test Types:	RGB, SM
Description:	Context pushes/pops environmental attributes. Their values should not be affected by a context pop.
Attributes Tested:	See Table 12-4, Column C at the end of this chapter.
Operators Tested:	`xgl_context_pop` `xgl_context_push`
Output:	Nothing is displayed.

▼ context_gf_attrs_rgb

Test Types:	RGB, SM
Description:	Context pushes/pops graphical attributes. Makes sure their values are all pushed and popped correctly.
Attributes Tested:	See Table 12-3 at the end of this chapter.

Operators Tested:	`xgl_context_pop`
	`xgl_context_push`
	`xgl_transform_read`
	`xgl_transform_rotate`
	`xgl_transform_scale`
	`xgl_transform_translate`

Output:	Nothing is displayed.

▼ context_pp_all_attrs

Test Types:	INDEX, SM
Description:	Context pushes/pops all pushable 3D context attributes by calling `xgl_context_push()` with a NULL attribute list. Makes sure all attribute values are pushed and popped correctly using `xgl_object_get()`.
Attributes Tested:	See Table 12-2 at the end of this chapter.

Operators Tested:	`xgl_context_pop`
	`xgl_context_push`
	`xgl_multiarc`
	`xgl_multimarker`
	`xgl_multipolyline`
	`xgl_nu_bspline_curve`
	`xgl_object_get`
	`xgl_object_set`
	`xgl_stroke_text`

Output:	A line, marker, and arc are displayed.

▼ context_pu_non_null_attrs

Test Types:	INDEX, SM
Description:	Context pushes/pops all pushable 3D context attributes by calling `xgl_context_push()` with a non-NULL attribute list. Makes sure all attribute values are pushed and popped correctly using `xgl_object_get()`.
Attributes Tested:	See Table 12-2 at the end of this chapter.

Operators Tested:	`xgl_context_pop`
	`xgl_context_push`
	`xgl_multiarc`
	`xgl_multimarker`
	`xgl_multipolyline`

	xgl_nu_bspline_curve
	xgl_object_get
	xgl_object_set
	xgl_stroke_text
Output:	A line, marker and arc are displayed.

▼ context_accumulate

Test Types:	SM, RGB
Description:	Tests xgl_context_accumulate operators
Attributes Tested:	XGL_BUFFER_SEL_DISPLAY
Operators Tested:	xgl_context_accumulate
	xgl_context_clear_accumulation
Output:	Draws a polygon, accumulates draw buffer onto itself than clears buffer

▼ context_zbuf_comp_method

Test Types:	RGB, CM
Description:	Tests Z_BUFFER_COMP_METHOD attributes
Attributes Tested:	XGL_3D_CTX_Z_BUFFER_COMP_METHOD
Operators Tested:	xgl_multiarc
	xgl_multirectangle
	xgl_multicircle
	xgl_context_new_frame
Output:	A stack of red circles, blue arcs, and green rectangles

▼ context_zbuf_comp_method2

Test Types:	RGB, CM
Description:	Tests Z_BUFFER_COMP_METHOD attributes
Attributes Tested:	XGL_3D_CTX_Z_BUFFER_COMP_METHOD
Operators Tested:	xgl_multiarc
	xgl_multirectangle
	xgl_multicircle
	xgl_context_new_frame
Output:	A stack of red circles, blue arcs, and green rectangles

▼ edge_3d

Test Types:	CM, RGB
Description:	Tests EDGE_CAP, EDGE_JOIN, and EDGE_MITER_LIMIT
Attributes Tested:	XGL_CTX_EDGE_CAP XGL_CTX_EDGE_JOIN XGL_CTX_EDGE_MITER_LIMIT
Operators Tested:	xgl_multirectangle
Output:	A stack of hollow rectangles with edges on

▼ draw_front_buffer

Test Types:	SM, RGB
Description:	Tests rendering to front buffer with context attribute XGL_CTX_RENDER_BUFFER
Attributes Tested:	XGL_WIN_RAS_BUFFERS_REQUESTED XGL_WIN_RAS_BUFFERS_ALLOCATED XGL_WIN_RAS_BUF_DISPLAY XGL_WIN_RAS_BUF_DRAW XGL_CTX_RENDER_BUFFER XGL_RENDER_DISPLAY_BUFFER
Operators Tested:	xgl_context_new_frame xgl_object_get xgl_object_set xgl_multiarc xgl_multicircle xgl_multipolyline xgl_multimarker xgl_multi_simple_polygon xgl_polygon
Output:	A blue background is drawn then a red background with a blue arc appears. A blue circle on a red background, a green polyline, four yellow asterisk markers, and then three consecutive blue squares are displayed.

Table 12-1 Context Attributes Tested - Set 1

Column A	Column B	Column C
XGL_CTX_LINE_COLOR	XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_LINE_COLOR
XGL_LPAT_DATA_SIZE	XGL_CTX_NURBS_CURVE_APPROX (XGL_CURVE_METRIC_VDC)	XGL_LPAT_DATA
XGL_LPAT_DATA	XGL_CTX_NURBS_CURVE_APPROX_VAL	XGL_CTX_LINE_PATTERN
XGL_LPAT_COORD_SYS (XGL_LPAT_DC)	XGL_MULTICIRCLE_I2D	XGL_CTX_LINE_STYLE (XGL_LINE_PATTERNED)
XGL_LPAT_DATA_TYPE (XGL_DATA_INT)	XGL_MULTIRECT_I2D	
XGL_LPAT_OFFSET		
XGL_CTX_LINE_PATTERN		
XGL_CTX_LINE_STYLE		
XGL_PT_I2D		

Table 12-2 Context Attributes Tested - Set 2

XGL_CTX_DEFERRAL_MODE	XGL_CTX_RENDERING	XGL_CTX_VDC_ORIENTATION
XGL_CTX_MODEL_TRANS_STACK_ SIZE	XGL_CTX_PICK_ENABLE	XGL_CTX_PICK_APERTURE
XGL_CTX_PICK_SURF_STYLE	XGL_CTX_VIEW_MODEL_DATA_ TYPE	XGL_CTX_LOCAL_MODEL_TRANS
XGL_CTX_GLOBAL_MODEL_ TRANS	XGL_CTX_VIEW_TRANS	XGL_CTX_VIEW_CLIP_BOUNDS
XGL_CTX_VDC_WINDOW	XGL_CTX_CLIP_PLANES	XGL_CTX_VDC_MAP
XGL_CTX_THRESHOLD	XGL_CTX_PLANE_MASK	XGL_CTX_ROP
XGL_CTX_BACKGROUND_COLOR	XGL_CTX_RASTER_FILL_STYLE	XGL_CTX_RASTER_STIPPLE_ COLOR
XGL_CTX_RASTER_FPAT_POSITION	XGL_CTX_RASTER_FPAT	XGL_CTX_NEW_FRAME_ACTION
XGL_CTX_LINE_COLOR	XGL_CTX_LINE_CAP	XGL_CTX_LINE_JOIN

Table 12-2 Context Attributes Tested - Set 2 *(Continued)*

XGL_CTX_LINE_MITER_LIMIT	XGL_CTX_LINE_STYLE	XGL_CTX_LINE_PATTERN
XGL_CTX_LINE_ALT_COLOR	XGL_CTX_LINE_WIDTH_SCALE_FACTOR	XGL_CTX_LINE_COLOR_SELECTOR
XGL_CTX_MIN_TESSELLATION	XGL_CTX_MAX_TESSELLATION	XGL_CTX_NURBS_CURVE_APPROX
XGL_CTX_NURBS_CURVE_APPROX_VAL	XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_CTX_SURF_INTERIOR_RULE	XGL_CTX_SURF_FRONT_FPAT_POSITION	XGL_CTX_SURF_FRONT_FPAT
XGL_CTX_SURF_FRONT_COLOR_SELECTOR	XGL_CTX_SURF_EDGE_FLAG	XGL_CTX_EDGE_COLOR
XGL_CTX_EDGE_STYLE	XGL_CTX_EDGE_PATTERN	XGL_CTX_EDGE_ALT_COLOR
XGL_CTX_EDGE_WIDTH_SCALE_FACTOR	XGL_CTX_ARC_FILL_STYLE	XGL_CTX_MARKER_COLOR
XGL_CTX_MARKER_SCALE_FACTOR	XGL_CTX_MARKER	XGL_CTX_MARKER_COLOR_SELECTOR
XGL_CTX_SFONT_0	XGL_CTX_SFONT_1	XGL_CTX_SFONT_2
XGL_CTX_SFONT_3	XGL_CTX_STEXT_CHAR_ENCODING	XGL_CTX_STEXT_CHAR_HEIGHT
XGL_CTX_STEXT_CHAR_SPACING	XGL_CTX_STEXT_CHAR_EXPANSION_FACTOR	XGL_CTX_STEXT_PATH
XGL_CTX_STEXT_CHAR_UP_VECTOR	XGL_CTX_STEXT_CHAR_SLANT_ANGLE	XGL_CTX_STEXT_ALIGN_HORIZ
XGL_CTX_STEXT_ALIGN_VERT	XGL_CTX_STEXT_PRECISION	XGL_CTX_STEXT_COLOR
XGL_CTX_PICK_ID_1	XGL_CTX_PICK_ID_2	XGL_CTX_ATEXT_CHAR_HEIGHT
XGL_CTX_ATEXT_CHAR_UP_VECTOR	XGL_CTX_ATEXT_CHAR_SLANT_ANGLE	XGL_CTX_ATEXT_PATH
XGL_CTX_ATEXT_ALIGN_HORIZ	XGL_CTX_ATEXT_ALIGN_VERT	XGL_CTX_ATEXT_STYLE
XGL_CTX_PICK_BUFFER_SIZE	XGL_CTX_PICK_STYLE	XGL_CTX_DC_VIEWPORT

Table 12-3 Context Attributes Tested - Set 3

XGL_3D_CTX_HLHSR_DATA	XGL_3D_CTX_HLHSR_MODE	XGL_3D_CTX_LIGHT_SWITCHES
XGL_3D_CTX_LINE_COLOR_ INTERP	XGL_3D_CTX_SURF_BACK_ AMBIENT	XGL_3D_CTX_SURF_BACK_COLOR
XGL_3D_CTX_SURF_BACK_DIFFUS E	XGL_3D_CTX_SURF_BACK_FILL_ STYLE	XGL_3D_CTX_SURF_BACK_FPAT
XGL_3D_CTX_SURF_BACK_FPAT_ POSITION	XGL_3D_CTX_SURF_BACK_ ILLUMINATION	XGL_3D_CTX_SURF_BACK_LIGHT_ COMPONENT
XGL_3D_CTX_SURF_BACK_LIGHT_ TYPE	XGL_3D_CTX_SURF_BACK_ SPECULAR	XGL_3D_CTX_SURF_BACK_ SPECULAR_COLOR
XGL_3D_CTX_SURF_BACK_ SPECULAR_POWER	XGL_3D_CTX_SURF_FACE_CULL	XGL_3D_CTX_SURF_FACE_ DISTINGUISH
XGL_3D_CTX_SURF_FRONT_ AMBIENT	XGL_3D_CTX_SURF_FRONT_ DIFFUSE	XGL_3D_CTX_SURF_FRONT_ ILLUMINATION
XGL_3D_CTX_SURF_FRONT_ LIGHT_COMPONENT	XGL_3D_CTX_SURF_FRONT_ LIGHT_TYPE	XGL_3D_CTX_SURF_FRONT_ SPECULAR
XGL_3D_CTX_SURF_FRONT_ SPECULAR_COLOR	XGL_3D_CTX_SURF_FRONT_ SPECULAR_POWER	XGL_3D_CTX_SURF_GEOM_ NORMAL
XGL_3D_CTX_VIEW_CLIP_PLUS_W_ ONLY	XGL_CTX_ATEXT_CHAR_HEIGHT	XGL_CTX_ATEXT_CHAR_UP_ VECTOR
XGL_CTX_ATEXT_STYLE	XGL_CTX_ATEXT_ALIGN_HORIZ	XGL_CTX_ATEXT_ALIGN_VERT
XGL_CTX_ARC_FILL_STYLE	XGL_CTX_BACKGROUND_COLOR	XGL_CTX_CLIP_PLANES
XGL_CTX_NURBS_CURVE_APPROX	XGL_CTX_NURBS_CURVE_ APPROX_VAL	XGL_CTX_DC_VIEWPORT
XGL_CTX_EDGE_ALT_COLOR	XGL_CTX_EDGE_COLOR	XGL_CTX_EDGE_PATTERN
XGL_CTX_EDGE_STYLE	XGL_CTX_EDGE_WIDTH_SCALE_ FACTOR	XGL_CTX_GLOBAL_MODEL_ TRANS
XGL_CTX_LINE_ALT_COLOR	XGL_CTX_LINE_CAP	XGL_CTX_LINE_COLOR
XGL_CTX_LINE_JOIN	XGL_CTX_LINE_PATTERN	XGL_CTX_LINE_STYLE
XGL_CTX_LINE_WIDTH_SCALE_ FACTOR	XGL_CTX_LOCAL_MODEL_TRANS	XGL_CTX_MARKER_COLOR

Table 12-3 Context Attributes Tested - Set 3 *(Continued)*

XGL_CTX_MARKER	XGL_CTX_MARKER_SCALE_ FACTOR	XGL_CTX_MAX_TESSELLATION
XGL_CTX_LINE_MITER_LIMIT	XGL_CTX_NEW_FRAME_ACTION	XGL_CTX_PICK_ID_1
XGL_CTX_PICK_ID_2	XGL_CTX_PLANE_MASK	XGL_CTX_RASTER_FILL_STYLE
XGL_CTX_RASTER_FPAT	XGL_CTX_RASTER_FPAT_POSITION	XGL_CTX_RASTER_STIPPLE_ COLOR
XGL_CTX_ROP	XGL_CTX_STEXT_CHAR_ EXPANSION_FACTOR	XGL_CTX_STEXT_CHAR_HEIGHT
XGL_CTX_STEXT_CHAR_SLANT_ ANGLE	XGL_CTX_STEXT_CHAR_SPACING	XGL_CTX_STEXT_CHAR_UP_ VECTOR
XGL_CTX_SFONT_0	XGL_CTX_SFONT_1	XGL_CTX_SFONT_2
XGL_CTX_SFONT_3	XGL_CTX_STEXT_ALIGN_HORIZ	XGL_CTX_STEXT_ALIGN_VERT
XGL_CTX_STEXT_COLOR	XGL_CTX_STEXT_PATH	XGL_CTX_STEXT_PRECISION
XGL_CTX_SURF_EDGE_FLAG	XGL_CTX_SURF_FRONT_FPAT	XGL_CTX_SURF_FRONT_FPAT_ POSITION
XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_SURF_FRONT_FILL_ STYLE	XGL_CTX_THRESHOLD
XGL_CTX_VDC_MAP	XGL_CTX_VDC_WINDOW	XGL_CTX_VIEW_CLIP_BOUNDS
XGL_CTX_VIEW_TRANS		

Table 12-4 Context Attributes Tested - Set 4

Column A	**Column B**	**Column C**
XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_DEFERRAL_MODE	XGL_CTX_DEFERRAL_MODE
XGL_CTX_DEFERRAL_MODE (XGL_DEFER_ASAP) (XGL_DEFER_ASTI)	XGL_CTX_DEVICE	XGL_CTX_DEVICE
	XGL_CTX_THRESHOLD	XGL_CTX_RENDERING
		XGL_CTX_PICK_BUFFER_SIZE
		XGL_CTX_PICK_STYLE
		XGL_CTX_PICK_ENABLE

Table 12-4 Context Attributes Tested - Set 4

Column A	**Column B**	**Column C**
		XGL_CTX_PICK_APERTURE
		XGL_CTX_VDC_ORIENTATION
		XGL_3D_CTX_DEPTH_CUE_MODE
		XGL_3D_CTX_DEPTH_CUE_COLOR
		XGL_3D_CTX_LIGHT_NUM
		XGL_3D_CTX_LIGHTS

Depth Cueing Test Descriptions 13

This chapter describes the Depth Cueing test programs. The following is defined for each test program:

- Name of the test program
- Test types (see the section “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ dcue_fat_line

Test Types:	INDEX, SM
Description:	Draws fat lines and checks for simple depth-cueing

Attributes Tested:	`XGL_CTX_LINE_WIDTH_SCALE_FACTOR`
	`XGL_CTX_VDC_MAP (XGL_VDC_MAP_ASPECT)`
	`XGL_3D_CTX_DEPTH_CUE_MODE`
	`(XGL_DEPTH_CUE_LINEAR)`
	`XGL_3D_CTX_DEPTH_CUE_COLOR`
	`XGL_CTX_LINE_COLOR`
	`XGL_PT_F3D`

Operators Tested:	`xgl_multipolyline`

Output: Fat lines displayed in depth cue colors black and yellow

- line on the front plane
- line on the back plane
- line lies on the plane in between the front and the back, const. color
- line with z1 > z2
- line with z1 < z2
- line with multiple segments in between the planes
- line with multiple polylines in between the planes

▼ dcue_fat_line_rgb

Test Types: RGB, SM
Description: Draws fat RGB lines and checks for simple depth cue
Attributes Tested: XGL_CTX_LINE_WIDTH_SCALE_FACTOR
XGL_CTX_VDC_MAP (XGL_VDC_MAP_ASPECT)
XGL_3D_CTX_DEPTH_CUE_MODE
(XGL_DEPTH_CUE_LINEAR)
XGL_3D_CTX_DEPTH_CUE_COLOR
XGL_CTX_LINE_COLOR
XGL_PT_F3D
Operators Tested: xgl_multipolyline
Output: Fat lines displayed in depth cue colors black and yellow:

- line on the front plane
- line on the back plane
- line lies on the plane in between the front and the back, const. color
- line with z1 > z2
- line with z1 < z2
- line with multiple segments in between the planes
- line with multiple polylines in between the planes

▼ dcue_line

Test Types: INDEX, SM
Description: Draws lines and checks simple depth-cueing
Attributes Tested: XGL_CTX_VDC_MAP (XGL_VDC_MAP_ASPECT)
XGL_3D_CTX_DEPTH_CUE_MODE
(XGL_DEPTH_CUE_LINEAR)

	`XGL_3D_CTX_DEPTH_CUE_COLOR`
	`XGL_CTX_LINE_COLOR`
	`XGL_PT_F3D`
Operators Tested:	`xgl_multipolyline`
Output:	Lines displayed in depth cue colors black and yellow:

- line on the front plane
- line on the back plane
- line lies on the plane in between the front and the back, const. color
- line with z1 > z2
- line with z1 < z2
- line with multiple segments in between the planes
- line with multiple polylines in between the planes

▼ dcue_line_rgb

Test Types:	RGB, SM
Description:	Draws RGB lines and checks simple depth-cueing
Attributes Tested:	`XGL_CTX_VDC_MAP (XGL_VDC_MAP_ASPECT)`
	`XGL_3D_CTX_DEPTH_CUE_MODE (XGL_DEPTH_CUE_LINEAR)`
	`XGL_3D_CTX_DEPTH_CUE_COLOR`
	`XGL_CTX_LINE_COLOR`
	`XGL_PT_F3D`
Operators Tested:	`xgl_multipolyline`
Output:	Lines displayed in depth cue colors black and yellow:

- line on the front plane
- line on the back plane
- line lies on the plane in between the front and the back, const. color
- line with z1 > z2
- line with z1 < z2
- line with multiple segments in between the planes
- line with multiple polylines in between the planes

▼ dcue_quadmesh

Test Types:	INDEX, SM
Description:	Renders depth-cued quadmeshes on different planes with front and back facing

Attributes Tested:	XGL_CTX_VDC_MAP (XGL_VDC_MAP_ASPECT) XGL_3D_CTX_DEPTH_CUE_COLOR XGL_3D_CTX_DEPTH_CUE_MODE (XGL_DEPTH_CUE_LINEAR) XGL_3D_CTX_SURF_FACE_DISTINGUISH XGL_CTX_SURF_FRONT_FILL_STYLE XGL_CTX_SURF_FRONT_COLOR XGL_3D_CTX_SURF_BACK_COLOR XGL_PT_F3D
Operators Tested:	xgl_quadrilateral_mesh
Output:	Red quadmeshes with depth cue color black: • quadmesh on front plane • quadmesh on back plane • quadmesh lies on plane between the front and the back planes, z constant quadmesh lies between the front and the back planes, z variable • quadmesh on front plane, back-facing • quadmesh with two facets between the front and back planes

▼ dcue_quadmesh_rgb

Test Types:	RGB, SM
Description:	Renders depth-cued quadmeshes on different planes with front and back facing
Attributes Tested:	XGL_CTX_VDC_MAP (XGL_VDC_MAP_ASPECT) XGL_3D_CTX_DEPTH_CUE_COLOR XGL_3D_CTX_DEPTH_CUE_MODE (XGL_DEPTH_CUE_LINEAR) XGL_3D_CTX_SURF_FACE_DISTINGUISH XGL_CTX_SURF_FRONT_FILL_STYLE XGL_CTX_SURF_FRONT_COLOR XGL_3D_CTX_SURF_BACK_COLOR XGL_PT_F3D
Operators Tested:	xgl_quadrilateral_mesh
Output:	Red quadmeshes with depth cue color black: • quadmesh on front plane • quadmesh on back plane • quadmesh lies on plane between the front and the back planes, z constant

- quadmesh lies between the front and the back planes, z variable
- quadmesh on front plane, back-facing
- quadmesh with two facets between the front and back planes

▼ dcue_scaled_line

Test Types:	INDEX, CM
Description:	Tests scaled depth-cueing of multipolylines with some clipped. Sets variant reference planes' depths and various scale factors. Since this is an *index cmap* test for depth cueing, a color ramp is set up and 3D polyline point lists are defined with constant z and varying z values. Some of the lines are outside of the viewport to test clipping. Reference planes are default shallow, and deep (0, 1) (.4, .6) (.1, .9); scale factors are default (1.0, 0), (.4, .5), and (.1, .9).
Attributes Tested:	XGL_3D_CTX_DEPTH_CUE_MODE XGL_DEPTH_CUE_SCALED XGL_3D_CTX_DEPTH_CUE_REF_PLANES XGL_3D_CTX_DEPTH_CUE_SCALE_FACTORS
Operators Tested:	xgl_multipolyline
Output:	Nine images are compared

▼ dcue_scaled_line_rgb

Test Types:	RGB, CM
Description:	Tests scaled depth-cueing of multipolylines with some clipped. Sets variant reference planes' depths and various scale factors. Since this is an *index cmap* test for depth cueing, a color ramp is set up and 3D polyline point lists are defined with constant z and varying z values. Some of the lines are outside of the viewport to test clipping. Reference planes are default shallow, and deep (0, 1) (.4, .6) (.1, .9); scale factors are default (1.0, 0), (.4, .5), and (.1, .9).

Attributes Tested:	XGL_3D_CTX_DEPTH_CUE_MODE XGL_DEPTH_CUE_SCALED XGL_3D_CTX_DEPTH_CUE_REF_PLANES XGL_3D_CTX_DEPTH_CUE_SCALE_FACTORS
Operators Tested:	xgl_multipolyline
Output:	Nine images are compared

▼ dcue_scaled_pg

Test Types:	INDEX, CM
Description:	Tests scaled depth-cueing of general polygons. Sets variant reference planes' depths and various scale factors. Since this is an *index cmap* test for depth cueing, a color ramp is set up and 3D polygons point lists are defined with constant z and varying z values. Reference planes are default shallow, and deep (0, 1) (.4, .6) (.1, .9); scale factors are default (1.0, 0), (.4, .5), and (.1, .9).
Attributes Tested:	XGL_3D_CTX_DEPTH_CUE_MODE XGL_3D_CTX_DEPTH_CUE_REF_PLANES XGL_3D_CTX_DEPTH_CUE_SCALE_FACTORS
Operators Tested:	xgl_polygon
Output:	Nine images are compared

▼ dcue_scaled_pg_rgb

Test Types:	RGB, CM
Description:	Tests scaled depth-cueing of general polygons. Sets variant reference planes' depths and various scale factors. Since this is an *rgb* test for depth cueing, a depth cue color and a surface color are defined; as z increases, the higher percentage are defined with constant z and varying z values. Reference planes are default, shallow, and deep (0, 1) (.4, .6) (.1, .9); scale factors are default (1.0, 0), (.4, .5), and (.1, .9).
Attributes tested:	XGL_3D_CTX_DEPTH_CUE_MODE XGL_DEPTH_CUE_SCALED XGL_3D_CTX_DEPTH_CUE_REF_PLANES XGL_3D_CTX_DEPTH_CUE_SCALE_FACTORS
Operators Tested:	xgl_polygon
Output:	Nine images are compared

dcue_simple

Test Types:	INDEX, SM
Description:	Draws a line and polygon and checks simple depth-cueing
Attributes Tested:	XGL_CTX_VDC_MAP (XGL_VDC_MAP_ASPECT) XGL_3D_CTX_DEPTH_CUE_COLOR XGL_3D_CTX_DEPTH_CUE_MODE (XGL_DEPTH_CUE_LINEAR) XGL_CTX_LINE_COLOR XGL_CTX_LINE_WIDTH_SCALE_FACTOR XGL_3D_CTX_LINE_COLOR_INTERP XGL_CTX_SURF_FRONT_FILL_STYLE XGL_CTX_SURF_FRONT_COLOR XGL_PT_F3D
Operators Tested:	xgl_multipolyline xgl_polygon
Output:	A red line and a red triangle displayed

dcue_simple_rgb

Test Types:	RGB, SM
Description:	Draws a RGB line and polygon and checks simple depth-cueing
Attributes Tested:	XGL_CTX_VDC_MAP (XGL_VDC_MAP_ASPECT) XGL_3D_CTX_DEPTH_CUE_COLOR XGL_3D_CTX_DEPTH_CUE_MODE (XGL_DEPTH_CUE_LINEAR) XGL_CTX_LINE_COLOR XGL_CTX_LINE_WIDTH_SCALE_FACTOR XGL_3D_CTX_LINE_COLOR_INTERP XGL_CTX_SURF_FRONT_FILL_STYLE XGL_CTX_SURF_FRONT_COLOR XGL_PT_F3D
Operators Tested:	xgl_polygon xgl_multipolyline
Output:	A red line and a red triangle displayed

dcue_triangle

Test Types: INDEX, SM
Description: Tests depth-cued triangles on different planes with front and back facing
Attributes Tested: XGL_CTX_VDC_MAP (XGL_VDC_MAP_ASPECT)
XGL_3D_CTX_DEPTH_CUE_COLOR
XGL_3D_CTX_DEPTH_CUE_MODE
(XGL_DEPTH_CUE_LINEAR)
XGL_3D_CTX_SURF_FACE_DISTINGUISH
XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_CTX_SURF_FRONT_COLOR
XGL_3D_CTX_SURF_BACK_COLOR
XGL_PT_F3D
Operators Tested: xgl_triangle_strip
Output: Red triangles with depth cue color black:

- triangle on front plane
- triangle on back plane
- triangle lies on plane between the front and the back planes, z constant triangle lies between the front and the back planes, z variable
- triangle on front plane, back-facing
- triangle strip with two facets between the front and back planes

dcue_triangle_rgb

Test Types: RGB, SM
Description: Tests depth-cued triangles on different planes with front and back facing
Attributes Tested: XGL_CTX_VDC_MAP (XGL_VDC_MAP_ASPECT)
XGL_3D_CTX_DEPTH_CUE_COLOR
XGL_3D_CTX_DEPTH_CUE_MODE
(XGL_DEPTH_CUE_LINEAR)
XGL_3D_CTX_SURF_FACE_DISTINGUISH
XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_CTX_SURF_FRONT_COLOR
XGL_3D_CTX_SURF_BACK_COLOR
XGL_PT_F3D
Operators Tested: xgl_triangle_strip

Output: Red triangles with depth cue color black:

- triangle on front plane
- triangle on back plane
- triangle lies on plane between the front and the back planes, z constant triangle lies between the front and the back planes, z variable
- triangle on front plane, back-facing
- triangle strip with two facets between the front and back planes

Elliptical Arc Test Descriptions 14

This chapter describes the Elliptical Arc test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section, "Denizen Test Types" on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ el0

Test Types:	CM, INDEX
Description:	Renders hollow ellipses utilizing an Index Color Scheme. Ellipses are composed of 360 elliptical open arcs utilizing gcache.
Attributes Tested:	XGL_ARC_OPEN XGL_AXIS_X XGL_AXIS_Y XGL_AXIS_Z XGL_CTX_ARC_FILL_STYLE XGL_CTX_NURBS_CURVE_APPROX

	XGL_CTX_NURBS_CURVE_APPROX_VAL_
	XGL_CTX_LINE_COLOR
	XGL_CTX_SURF_EDGE_FLAG
	XGL_CURVE_METRIC_VDC
	XGL_GCACHE
Operators Tested:	xgl_object_create
	xgl_object_set
	xgl_gcache_multi_elliptical_arc
	xgl_context_display_gcache
	xgl_object_destroy
	xgl_gcache_multi_elliptical_arc
Output:	The first loop renders rotated ellipses at increments of 30 degrees through 30 degrees past a 360-rotation around their original position such that ellipses appear at rotation angles of 30, 60, 90, 120, 150, 180, 210, 240, 270, 300, 330, 360, and 390. Each time the rotated ellipses are drawn, the edge color changes per module 7. The second loop renders ellipses in the plane specified by the direction vectors, which indicate rotations around each axis (x, y, and z) in 30-degree increments from 0 - 30 degrees past a 360 rotation around their original position.

▼ el1

Test Types:	CM, INDEX
Description:	Draws solid indexed arcs not gcached with one arc per full ellipse. The front color is different than the back surface color.
Attributes Tested:	XGL_3D_CTX_SURF_BACK_COLOR
	XGL_3D_CTX_SURF_BACK_FILL_STYLE
	XGL_3D_CTX_SURF_FACE_DISTINGUISH
	XGL_ARC_SECTOR
	XGL_AXIS_X
	XGL_CTX_ARC_FILL_STYLE
	XGL_CTX_NURBS_CURVE_APPROX
	XGL_CTX_NURBS_CURVE_APPROX_VAL
	XGL_CTX_SURF_EDGE_FLAG
	XGL_CTX_SURF_FRONT_COLOR

XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_CURVE_METRIC_VDC
XGL_SURF_FILL_SOLID

Operators Tested: xgl_object_set
xgl_multi_elliptical_arc

Output: The first loop renders rotated ellipses at increments of 30–360 degrees around their original position so that ellipses appear at rotation angles of 30, 60, 90, 120, 150, 180, 210, 240, 270,300, 330, and 360. Each time the rotated ellipses are drawn, the edge color changes per module 7. The second loop renders ellipses in the plane specified by the direction vectors which indicate rotations around each axis (x, y, and z) in 45-degree increments from 0–720 degrees, and two full rotations around each axis to see if front and back surface colors change as they are rotated to the back.

▼ el2

Test Types: CM, INDEX

Description: Renders annotation *f3d* ellipses utilizing an Index Color Scheme. Full ellipses are composed of 90 elliptical open arcs of 4 degrees each utilizing gcache.

Attributes Tested: XGL_ARC_OPEN
XGL_CTX_ARC_FILL_STYLE
XGL_CTX_NURBS_CURVE_APPROX
XGL_CTX_NURBS_CURVE_APPROX_VAL
XGL_CTX_LINE_COLOR
XGL_CTX_SURF_EDGE_FLAG
XGL_CURVE_METRIC_VDC

Operators Tested: xgl_object_set
xgl_multi_elliptical_arc

Output: A loop renders annotation *f3d* ellipses with different major and minor values so the greatest width and greatest height axis of the ellipses are varied. Edge colors change per module 7.

el3

Test Types: CM, INDEX

Description: Renders full *f3d* gcached ellipses using one elliptical arc and clipped dependent on the loop variable. The color of the solid ellipse changes each time the loop variable (the clipping planes) change. The clipping plane combinations tested are (1) XMIN, (2) XMAX, (3) YMIN, (4) YMAX, (5) XMIN in combination with XMAX, and (6) YMIN in combination with YMAX.

Attributes Tested:
XGL_3D_CTX_SURF_FACE_DISTINGUISH
XGL_ARC_SECTOR
XGL_CLIP_XMAX
XGL_CLIP_XMIN
XGL_CLIP_YMAX
XGL_CLIP_YMIN
XGL_CTX_ARC_FILL_STYLE
XGL_CTX_CLIP_PLANES
XGL_CTX_NURBS_CURVE_APPROX
XGL_CTX_NURBS_CURVE_APPROX_VAL
XGL_CTX_LINE_COLOR
XGL_CTX_SURF_EDGE_FLAG
XGL_CTX_SURF_FRONT_COLOR
XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_CTX_VIEW_CLIP_BOUNDS
XGL_CURVE_METRIC_VDC
XGL_GCACHE
XGL_SURF_FILL_SOLID

Operators Used:
xgl_object_create
xgl_object_set
xgl_gcache_multi_elliptical_arc
xgl_context_display_gcache
xgl_object_destroy
xgl_context_push

Output: The first loop tests XMIN of 60 with all other bounds as normal, that is, XMAX: 600, YMIN: 0, YMAX: 600, ZMIN: 0, and finally, ZMAX: 1. The uppermost bound for the loop is 7. The center points for the seven ellipses are rendered so that we expect the minor diameter to be clipped. The expected clipped minor diameter ranges from a maximum expected visible minor diameter of 6 to a minium of 0, dependent on the value of the center of the ellipse. The center of the ellipse changes from the XMIN value of 60 to a minimum of 55 and a maximum of 61.

The second loop tests XMAX of 59 with all other bounds as normally defined. The uppermost bound for the loop is 7. The center points for the seven ellipses rendered are the same as the first loop. We expect the second loop to complete the ellipse produced by the first loop but with a different color.

The third loop tests YMIN of 40 with all other bounds as normally defined. Uppermost bound for the loop is 11. The center points for the 11 ellipses are rendered so that we expect the major diameter to be clipped. The expected clipped major diameter ranges from a maximum expected visible major diameter of 10 to a minimum of 0, dependent on the value of the center of the ellipse. The center of the ellipse changes from the XMIN value of 40 to a minimum of 30.

The fourth loop tests YMAX of 39, with all other bounds as normally defined. The uppermost bound for the loop is 11. The center points for the 11 ellipses rendered are the same as the previous loop. We expect the second loop to complete the ellipse produced by the first loop, but with a different color.

The fifth loop tests the combination of clip planes of XMIN in combination with XMAX. XMIN is set to 95, XMAX is set to 105, and all other bounds are as normally would be defined. The uppermost bound for the loop is 18. The center points for the 18 ellipses are rendered so that we expect the minor diameter to be clipped. The expected clipped minor diameter ranges from a maximum expected visible minor diameter of 10 to a minimum of 0, dependent on the value of the center of the ellipse. The center of the ellipse changes from the XMIN value of 95 to a minimum of 90 and a maximum value of 107.

The sixth and final loop tests the combination of clip planes of YMIN in combination with YMAX. YMIN is set to 60, YMAX is set to 70, and all other bounds are as normally would be defined. The uppermost bound for the loop is 26. The center points for the 26 ellipses are rendered so that we expect the major diameter to be clipped. The expected clipped major diameter ranges from a maximum expected visible major diameter of 11 to a minimum of 1, dependent on the value of the center of the ellipse. The center of the ellipse changes from the YMIN value of 60 to a minimum of 50 and a maximum value of 75.

▼ el4

Test Types: CM, INDEX

Description: Renders 3D elliptical arcs in different planes by applying a rotation around either the y- or z-axis for the first two directional vectors, which determine where the elliptical arc is located inside the plane. The rotations occur in 45 degree increments and are applied in such a way that more than two full revolutions around the axis are represented. Face distinguishing is on, the front color is yellow, and the back color is blue.

Attributes Tested:
XGL_CTX_NURBS_CURVE_APPROX
XGL_CURVE_METRIC_VDC
XGL_CTX_NURBS_CURVE_APPROX_VAL
XGL_CTX_SURF_EDGE_FLAG
XGL_3D_CTX_SURF_FACE_DISTINGUISH
XGL_CTX_ARC_FILL_STYLE
XGL_ARC_SECTOR
XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_SURF_FILL_SOLID
XGL_3D_CTX_SURF_BACK_FILL_STYLE

Operators Tested:
xgl_object_set
xgl_multi_elliptical_arc

Output: Rows of elliptical arcs in a shape similar to a peanut. The ellipses are all yellow for the rotations around the z-axis, but they change to the back color blue for some portion of the rotations around the y-axis.

el5

Test Types: CM, INDEX

Description: 3D gcached multiple clipped elliptical arcs. Clip plane combinations tested are in the order of their appearance: (1) XMIN | YMIN, (2) XMAX | YMAX, (3) XMAX | YMIN, (4) XMIN | YMAX, and finally the clipping combination (5) XMIN | YMIN | XMAX | YMAX.

Attributes Tested:
XGL_3D_CTX_SURF_FACE_DISTINGUISH
XGL_ARC_SECTOR
XGL_CLIP_XMAX
XGL_CLIP_XMIN
XGL_CLIP_YMAX
XGL_CLIP_YMIN
XGL_CTX_ARC_FILL_STYLE
XGL_CTX_CLIP_PLANES
XGL_CTX_NURBS_CURVE_APPROX
XGL_CTX_NURBS_CURVE_APPROX_VAL
XGL_CTX_LINE_COLOR
XGL_CTX_SURF_EDGE_FLAG
XGL_CTX_SURF_FRONT_COLOR
XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_CTX_VIEW_CLIP_BOUNDS
XGL_CURVE_METRIC_VDC
XGL_SURF_FILL_SOLID

Operators Tested:
xgl_object_set
xgl_multi_elliptical_arc
xgl_context_push

Output: Renders the first combination in white, the second combination in red, the third combination in green, and the fourth combination in blue. These four combinations are used to render one entire ellipse composed of the white, red, green, and blue partitions as a result of the clipping scheme. The final clipping combination renders elliptical arcs clipped on four sides, where each successive arc is rendered vertically beneath the previous arc.

Lighting Test Descriptions 15

This chapter describes the Lighting test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section, “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ light_pg_amb_facet

Test Types:	INDEX, SM
Description:	Renders polygons without edges with `color_normal_flag_f3d` vertex type, one ambient light source, various ambient coefficients, and light colors utilizing per facet illumination. Sets five different light colors for each of the ambient coefficient sets, and tries three ambient coefficients. The light colors set starts very close to an index value of 0, which is the color black, and gradually lightens to an index value of the color table size, which in this case is equivalent to the color white in five

equidistant increments. The ambient coefficient selections are equidistant increments from 0 to 1 for a total of three selections.

Attributes Tested:	See Table 15-1, Column A at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_polygon`

Output:	For values close to 0 for the ambient coefficient and the light color, the square and triangle polygons are indistinguishable from the black background. For higher values for both these values, these two polygons can be seen in dark gray or lighter gray.

▼ light_pg_amb_simple_facet

Test Types:	INDEX, SM
Description:	Renders polygons without edges with *f3d* vertex type, one ambient light source, various ambient coefficients, and light colors utilizing per facet illumination. Sets three different light colors for each of the ambient coefficient sets, and tries three ambient coefficients. The light colors set starts very close to an index value of the background color 0, which with this color table is red, and gradually lightens to an index value equivalent to the color table size which represents a white color in three equidistant increments. The ambient coefficient selections are equidistant increments from 0 to 1 for a total of three selections.
Attributes Tested:	See Table 15-1, Column B at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_polygon`

Output:	For values close to 0 for the ambient coefficient and the light color, the square and triangle polygons are indistinguishable from the red background. For higher values for both these values, these two polygons can be seen in black, gray, or white.

light_pg_amb_vtx

Test Types:	INDEX, SM
Description:	Renders polygons without edges with `color_normal_f3d` vertex type, one ambient light source, various ambient coefficients, and light colors per vertex lighting. Three different cases are tried: (1) ambient coefficient 0.0 and light color white, (2) ambient coefficient 0.0 and light color gray, and (3) ambient coefficient 1.0 and light color white. Vertex colors for the square range from black to gray. Vertex colors for the triangle range from gray to light gray.
Attributes Tested:	See Table 15-1, Column C at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_polygon`

Output:	In the first case and second case, the square polygon and the triangle polygon are indistinguishable from the black background. In the third case, the square is shaded with the left portion a darker gray than the right portion, and the triangle appears practically white.

light_pg_amb_vtx_rgb

Test Types:	RGB, SM
Description:	Renders polygons without edges with `color_normal_f3d` vertex type, one ambient light source, various ambient coefficients, and light colors per vertex lighting. Three different cases are tried: (1) ambient coefficient 0.0 and light color red, (2) ambient coefficient 0.5 and light color light blue, and (3) ambient coefficient 1.0 and light color red. Vertex colors for the square come in shades of green. Vertex colors for the triangle come in shades of blue.
Attributes Tested:	See Table 15-1, Column C at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_polygon`

Output:	In the first case, both the square and the triangle polygons are indistinguishable from the black background. In the second case, the square is a dark green and the triangle a dark blue. In the final case, the square is black on the left side and red on the right side, while the triangle is red.

light_pg_amb_facet_rgb

Test Types:	RGB, SM
Description:	Renders polygons with `color_normal_flag_i3d` vertex type, one ambient light source, various ambient coefficients, and light colors per facet lighting. All vertex colors are cyan. Five different light colors are set for each of the three ambient coefficients. The light colors set ranges from shades of green to black to shades of magenta. The ambient coefficient selections are equidistant increments from 0 to 1 for a total of three selections.
Attributes Tested:	See Table 15-1, Column A at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_polygon`

Output:	For an ambient coefficient of 0, the square and triangle polygons are blended into the black background color. For the other two ambient coefficients, the two polygons are shades of green.

light_pg_pos_facet

Test Types:	INDEX, SM
Description:	Renders polygons with `color_normal_flag_i3d` vertex type, one positional light source, and various values for attributes that affect positional lighting per facet lighting. All vertex colors are black. For each of the three different attenuation coefficients, two different light positions are set. For the six values for the light color, three different attenuation coefficients are set. Six different light colors are set for three different combinations for the diffuse reflection coefficient, the specular reflection coefficient, and the object specular exponent. The first light position is

	directly in front of the square's first vertex. The second light position is directly in front of the triangle's middle vertex. The six different light colors are incremental shades of gray to the final shade of white.
Attributes Tested:	See Table 15-2, Column A at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_polygon`

Output:	In the cases where the light color is near white and the coefficients and exponents are high values, the square and the triangle polygons appear as the color gray, which is somewhere in the range from dark gray to practically white.

▼ light_ts_amb_facet

Test Types:	INDEX, SM
Description:	Renders tristrips without edges with `color_normal_flag_i3d` vertex type, one ambient light source, various ambient coefficients and light colors, in index mode, per facet lighting. The vertex colors are shades of gray. For three different equidistant values ranging from 0 to 1.0 for the ambient coefficient, three different light colors are orange and the other two range from black to white.
Attributes Tested:	See Table 15-2, Column B at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_triangle_strip`

Output:	Where the ambient coefficient is 0 or the light color is 0, the tristrip blends into the background color of orange. As the ambient coefficient climbs to 1.0 and the light color climbs to white, the tristrip varies from black to shades of gray and white.

▼ light_ts_pttypes_pos_facet

Test Types:	INDEX, SM
Description:	Renders all non-*i3d* vertex types for tristrips without edges with one positional light source, and fixed values for attributes that affect positional lighting per facet lighting. Uses nine different point types to test the same setting for lighting conditions for a tristrip made up of three triangles. Point types in the order of their use are XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, and XGL_PT_F3H. The vertex colors used for the point types with color information are virtually white. The surface color set is black and the light color set is close to white. The light position is the third vertex of the first triangle.
Attributes Tested:	See Table 15-2, Column A at the end of this chapter.
Operators Tested:	xgl_object_set xgl_object_get xgl_triangle_strip
Output:	The composite tristrip should blend into the black background.

▼ light_ts_pos_facet

Test Types:	INDEX, SM
Description:	Renders tristrips with color_normal_flag_f3d vertex type, one positional light source, and various values for attributes that affect positional lighting per facet lighting. The vertex color used for the composite tristrip made up of three triangles is the same as the background color, which is blue. For the six different light positions tried, three different settings for the two attenuation coefficients are set. For this setting, six different light colors are set. For the three different cases that specify the diffuse reflection coefficient, specular reflection coefficient, and the object specular exponent, six different light colors are tried. The light colors are equidistant from each other and

they range from dark gray to virtually white. The light positions are all in front of the primitive. They represent positions on a 45 degree line from the left side of the primitive past the outermost right side of the primitive while passing directly in front of the common vertex to all three triangles.

Attributes Tested: See Table 15-2, Column A at the end of this chapter.

Operators Tested: `xgl_object_set` `xgl_object_get` `xgl_triangle_strip`

Output: Dependent on variables set, zero, one, two, or three triangles are displayed black on a blue background.

▼ light_ts_edge_pos_facet_rgb

Test Types: RGB, SM

Description: Renders all non-*i3d* vertex types for tristrips with edges, one positional light source, and fixed values for attributes that affect positional lighting per facet lighting. The point types used in the order of their appearance are `flag_f3d`, `color_flag_f3d`, `normal_flag_f3d`, and `color_normal_flag_f3d`. Each point type is used to render the same primitive, which is a tristrip composed of three triangles. The facet color set for the first triangle is pink, the second is light green, and the third is white. The light color is a shade of magenta. The light position is the common vertex to all three triangles.

Attributes Tested: `XGL_CTX_SURF_EDGE_FLAG` `XGL_CTX_EDGE_WIDTH_SCALE_FACTOR` `XGL_CTX_EDGE_COLOR` and Table 15-2, Column A at the end of this chapter

Operators Tested: `xgl_object_set` `xgl_object_get` `xgl_triangle_strip`

Output: A tristrip composed of three triangles separated by wide red edges with the bottom triangle appearing gray while the other two appear black.

light_ts_dir_facet

Test Types: INDEX, SM

Description: Renders tristrips with *f3d* vertex type and facet color normals, one directional light source, and various values for attributes that affect directional lighting per facet lighting. The facet colors are white. The light direction is varied to point at each of the three triangles that make up the tristrip. The light color is white except for the last test tried.

Attributes Tested: `XGL_LIGHT_DIRECTIONAL`
`XGL_LIGHT_DIRECTION`
and Table 15-2, Column B at the end of this chapter

Operators Tested: `xgl_object_set`
`xgl_object_get`
`xgl_triangle_strip`

Output: The first frame: light directed at the first triangle which is light gray, the second triangle is black, and the third (bottom) triangle is dark gray.
The second frame: light directed at the second triangle. The first triangle is black, the second triangle is light gray, and the third (bottom) triangle is dark gray.
The third frame: light directed at the third (bottom) triangle. The two top triangles are black and the bottom triangle is white.
The final frame: changes light color from white to gray but leaves light direction as is. The top two triangles are black, and the bottom triangle is gray. The background color is red.

light_qm_edge_spot_facet

Test Types: INDEX, SM

Description: Renders all non-*i3d* vertex types for quadmeshes with edges, one spot light source, and fixed values for attributes that affect spot lighting per facet lighting. The same two quadmeshes are rendered using these four point types in the order specified: `flag_f3d`, `color_flag_f3d`, `normal_flag_f3d`, and

	`color_normal_flag_f3d`. The light color is white. The facet colors are various shades of white. The position of the light is at the second vertex in the first quadmesh.

Attributes Tested:	`XGL_LIGHT_ENABLE_TYPE_SPOT`
	`XGL_CTX_SURF_EDGE_FLAG`
	`XGL_CTX_EDGE_WIDTH_SCALE_FACTOR`
	`XGL_CTX_EDGE_COLOR`
	`XGL_LIGHT_SPOT`
	`XGL_LIGHT_SPOT_ANGLE`
	`XGL_LIGHT_SPOT_EXPONENT`
	`XGL_LIGHT_DIRECTION`
	and Table 15-2, Column A at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_quadrilateral_mesh`

Output:	Two quadmeshes sharing the same wide gray edge. The first quadmesh shows the *artifacts* from shading and rendering quadmeshes as tristrips by its composition of one lighter gray tristrip and one darker gray tristrip. The second quadmesh is black.

▼ light_qm_pttypes_spot_facet_rgb

Test Types:	RGB, SM
Description:	Renders all non `i3d` vertex types for quadmeshes without edges, with one spot light source, and with fixed values for attributes that affect spot lighting per facet lighting. The same two quadmeshes are rendered using nine different point types in the following order: `f3d`, `color_f3d`, `normal_f3d`, `color_normal_f3d`, `flag_f3d`, `color_flag_f3d`, `normal_flag_f3d`, `color_normal_flag_f3d`, and `f3h`. The facet colors are light red and light green. The light color is close to white. The light position is the first vertex of the first quadmesh.

Attributes Tested:	`XGL_LIGHT_ENABLE_TYPE_SPOT`
	`XGL_LIGHT_SPOT`
	`XGL_LIGHT_SPOT_ANGLE`
	`XGL_LIGHT_SPOT_EXPONENT`
	`XGL_LIGHT_DIRECTION`
	and Table 15-2, Column A at the end of this chapter

Operators Tested: `xgl_object_set`
`xgl_object_get`
`xgl_quadrilateral_mesh`

Output: The left quadmesh, due to the *artifacts* from shading and rendering quadmesh as tristrips, appears as a brown tristrip and a black tristrip. The right quadmesh is olive green. The background color is black.

▼ light_qm_spot_facet_rgb

Test Types: RGB, SM

Description: Renders quadmeshes with `color_normal_flag_f3d` vertex type, one spot light source, and various values for attributes that affect spot lighting per facet lighting. The vertex color is light green. The first facet color is light red, and the second facet color is light green. For seven different values of the light position, four different light directions are tried and two different settings for the two attenuation coefficients are tried. Six different light colors are tried for the two different settings for the two attenuation coefficients. Three different combinations for the diffuse reflection coefficient, the specular reflection coefficient, and the object specular exponent are tried with the six different light colors. The light colors set range from cyan to red. The light position varies from in front of the primitive to behind the primitive, but is always located at the first vertex of the first quadmesh.

Attributes Tested: `XGL_LIGHT_ENABLE_TYPE_SPOT`
`XGL_LIGHT_SPOT`
`XGL_LIGHT_SPOT_ANGLE`
`XGL_LIGHT_SPOT_EXPONENT`
and Table 15-2, Column A at the end of this chapter

Operators Tested: `xgl_object_set`
`xgl_object_get`
`xgl_quadrilateral_mesh`

Output: The background color is black. Two quadmeshes appear on the screen and may have a variety of different colors from this assortment: (1) both blend into the background, (2) blue quadrilateral mesh next to an olive green quadrilateral mesh, (3) blue quad next to a quad with *artifacts* from its tristrips counterparts (olive green / forest green), (4) mint green quad next to dark blue quad, (5) mint green quad next to a quad with *artifacts* from its tristrips counterparts (olive green / dark blue), (6) orange quad next to a black quad, (7) tan quad next to a black quad, (8) red quad next to a brown quad, (9) red quad next to a quad with *artifacts* from its tristrips counterparts (brown / black), (10) red quad next to a quad with *artifacts* from its tristrips counterparts (dark brown / light brown), (11) blue quad next to a forest green quad, (12) mint green quad next to an olive green quad, (13) quad with *artifacts* from its tristrips counterparts (blue / purple) next to an olive green quad (14) quad with *artifacts* from its tristrips counterparts (orange / tan), (15) pink quad next to a quad with *artifacts* from its tristrips counterparts (dark brown / black).

▼ light_spg_pttypes_dir_facet

Test Types:	INDEX, SM
Description:	Renders simple polygons with some version of *f3d* vertex type, one directional light source, and various values for attributes that affect directional lighting in index mode, per facet lighting. The same two polygons are rendered with the same settings for lighting attributes but with nine different vertex types in the following order: `f3d`, `color_f3d`, `normal_f3d`, `color_normal_f3d`, `flag_f3d`, `color_flag_f3d`, `normal_flag_f3d`, `color_normal_flag_f3d`, and `f3h`. The light color is practically white. The facet colors are white. The surface color is blue, which is the same as the background color.
Attributes Tested:	See Table 15-2, Column C at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_multi_simple_polygon`

Output:	The background color is blue. A square dark gray polygon on the left side, and a lighter gray bow tie polygon on the right side of the window raster.

▼ light_spg_edge_dir_facet_rgb

Test Types: RGB, SM

Description: Renders simple polygons with edges and some version of *f3d* vertex type, one directional light source, and various values for attributes that affect directional lighting per facet lighting. The same two polygons are rendered with the same settings for lighting attributes but with nine different vertex types in the following order: `f3d`, `color_f3d`, `normal_f3d`, `color_normal_f3d`, `flag_f3d`, `color_flag_f3d`, `normal_flag_f3d`, `color_normal_flag_f3d`, and `f3h`. The light color is practically white. The facet colors are white. The surface color is black the same as the background. The edge color is red.

Attributes Tested: `XGL_CTX_SURF_EDGE_FLAG`
`XGL_CTX_EDGE_COLOR`
`XGL_CTX_EDGE_WIDTH_SCALE_FACTOR`
and Table 15-2, Column C at the end of this chapter

Operators Tested: `xgl_object_set`
`xgl_object_get`
`xgl_multi_simple_polygon`

Output: The background color is black. A square dark gray polygon with red edges on left side and lighter gray bow tie polygon with red edges on the right side of the window raster.

▼ light_spg_edge_dir_facet_rgb_norm_flip

Test Types: RGB, SM

Description: Renders simple polygons with *color_normal_flag_f3d* vertex type with one directional light source with various values for attributes that affect directional lighting in RGB mode, per facet lighting. `XGL_CTX_3D_SURF_LIGHTING_NORMAL_FLIP` is set to `FALSE`. (Directional rgb of spg).

Attributes Tested:	XGL_CTX_SURF_EDGE_FLAG XGL_CTX_EDGE_COLOR XGL_CTX_EDGE_WIDTH_SCALE_FACTOR XGL_3D_CTX_SURF_LIGHTING_NORMAL_FLIP and Table 15-2, Column C at the end of this chapter
Operators Tested:	xgl_object_set xgl_object_get xgl_multi_simple_polygon
Output:	Two squares that change color

▼ light_spg_dir_facet_rgb

Test Types:	RGB, SM
Description:	Renders simple polygons with color_normal_flag_f3d vertex type, one directional light source, and various values for attributes that affect directional lighting per facet lighting. The facet color is white and the surface color is black. The light color ranges from green to magenta in six equidistant steps. For two different light directions, six different light colors are tried. For three different combinations for the diffuse reflection coefficient, the specular reflection coefficient, and the object specular exponent, six different light colors are tried.
Attributes Tested:	See Table 15-2, Column C at the end of this chapter.
Operators Tested:	xgl_object_set xgl_object_get xgl_multi_simple_polygon
Output:	The background color is black. Two polygons appear on the screen and may have a variety of different colors from this assortment: (1) both blend into the background, (2) an olive green square next to a forest green bowtie, (3) a neon green square next to an olive green bowtie, (4) an olive green square next to an olive green bow tie, (5) a green square, (6) a purple square, (7) a dark purple square next to a burgundy bow tie, (8) a neon purple square next to a dark burgundy bow tie, (9) a forest green square next to a neon green bow tie, (10) a neon green square next to a forest green bow tie, (11) a light blue square next to an olive green bow tie, (12) a burgundy square next to a

burgundy bow tie, (13) a violet square next to a burgundy bow tie, (14) a burgundy square next to a neon purple bow tie, (15) a neon purple square next to a purple bow tie, and (16) a purple square next to a neon purple bow tie.

▼ light_many

Test Types:	RGB, SM
Description:	Tries nine lights since Sun's gs (that is, cg12) permits eight lights in hardware. Renders two tristrips with vertex colors of black and a surface color of blue. The nine lights use light colors ranging from green to magenta. Sets all nine lights off and renders three tristrips normally. Sets the nine lights on, one by one, each time rendering the three tristrips. Finally, sets the nine lights off, one by one, each time rendering the three tristrips. The variety of the nine lights set consists of three ambient, two directional, two positional, and two spot sources.

Attributes Tested:	`XGL_LIGHT_ENABLE_COMP_AMBIENT`
	`XGL_3D_CTX_SURF_FRONT_AMBIENT`
	`XGL_LIGHT_AMBIENT`
	`XGL_LIGHT_DIRECTIONAL`
	`XGL_LIGHT_DIRECTION`
	`XGL_LIGHT_POSITIONAL`
	`XGL_LIGHT_SPOT`
	`XGL_LIGHT_SPOT_ANGLE`
	`XGL_LIGHT_SPOT_EXPONENT`
	and Table 15-2, Column A at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_triangle_strip`

Output:	Turning the lights on, one by one: (1) lights all off—tristrips blend into black background, (2) add ambient light source—first tristrip is black, second is neon green, and bottom is forest green, (3) add directional light source—first tristrip is black, the other two are neon green, (4) add positional light source—no change, (5) add spot light source—no change, (6) add ambient light source—first tristrip is gray, second is mint green, and bottom is neon green, (7) add directional light source—1st

tristrip is gray, second is light gray, and bottom is mint green, (8) add positional light source—no change, (9) add spot light source—no change, (10) add ambient light source—first tristrip is purple, second is light gray, and bottom is light blue.

Turning the lights off, one by one: (1) ambient lights off—first tristrip is neon purple, second is gray, and bottom is violet, (2) directional light off—first tristrip is neon purple, second is violet, and bottom is purple, (3) positional light off—no change, (4) spot light off—no change, (5) ambient light off—first and second tristrips are neon purple, and bottom tristrip is blue, (6) directional light off—no change, (7) positional light off—no change, (8) spot light off—no change, and (9) ambient light off—tristrip composite blends into black background.

▼ light_copy

Test Types: RGB, SM

Description: Copies a light object into another light object. Tries all light types and verifies that the destination light object has the same properties as the source light object after the copying is completed. Tests that this works when the source light object and destination light object are the same.

Attributes Tested:
`XGL_LIGHT`
`XGL_LIGHT_TYPE`
`XGL_LIGHT_AMBIENT`
`XGL_LIGHT_COLOR`
`XGL_LIGHT_DIRECTIONAL`
`XGL_LIGHT_DIRECTION`
`XGL_LIGHT_POSITIONAL`
`XGL_LIGHT_POSITION`
`XGL_LIGHT_ATTENUATION_1`
`XGL_LIGHT_ATTENUATION_2`
`XGL_LIGHT_SPOT`
`XGL_LIGHT_SPOT_ANGLE`
`XGL_LIGHT_SPOT_EXPONENT`

Operators Tested:
`xgl_object_set`
`xgl_object_get`
`xgl_light_copy`

Output: No window raster is brought up and the action all appears to be *behind the scenes*.

▼ light_ts_amb_dir_facet

Test Types: INDEX, SM

Description: Sets up two lights, one ambient and the other directional. Draws the front and back facing triangles with the directional light off to check if ambient is still applied correctly. The first triangle is facing the front, and the second triangle is facing the back. The illumination is per facet.

Attributes Tested:
`XGL_3D_CTX_SURF_BACK_COLOR`
`XGL_3D_CTX_SURF_FACE_DISTINGUISH`
`XGL_CTX_SURF_FRONT_FILL_STYLE`

XGL_SURF_FILL_SOLID
XGL_3D_CTX_SURF_BACK_FILL_STYLE
XGL_3D_CTX_SURF_FRONT_AMBIENT
XGL_3D_CTX_SURF_BACK_AMBIENT
XGL_3D_CTX_SURF_BACK_DIFFUSE
XGL_3D_CTX_SURF_BACK_SPECULAR
XGL_3D_CTX_SURF_BACK_SPECULAR_POWER
XGL_3D_CTX_HLHSR_MODE
XGL_3D_CTX_SURF_BACK_SPECULAR_COLOR
XGL_3D_CTX_SURF_BACK_ILLUMINATION
XGL_HLHSR_NONE
XGL_LIGHT_AMBIENT
and Table 15-2, Column C at the end of this chapter

Operators Tested: xgl_object_set
xgl_triangle_strip

Output: Both lights on, two triangles side by side with the leftmost light gray and the second dark gray. Only the ambient light source and both triangles are black. The background color is red.

▼ light_ts_amb_dir_vtx

Test Types: INDEX, SM

Description: Sets up two lights, one ambient and the other directional. Draws the front-facing and back-facing triangles with the directional light off. The first triangle is facing the front, and the second triangle is facing the back. Illumination is per vertex.

Attributes Tested: XGL_3D_CTX_SURF_FACE_DISTINGUISH
XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_SURF_FILL_SOLID
XGL_3D_CTX_SURF_BACK_FILL_STYLE
XGL_3D_CTX_SURF_FRONT_AMBIENT
XGL_ILLUM_PER_VERTEX
XGL_LIGHT_ENABLE_COMP_AMBIENT
XGL_3D_CTX_SURF_BACK_LIGHT_COMPONENT
XGL_LIGHT_AMBIENT
and Table 15-2, Column C at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_triangle_strip`
	`xgl_object_get`

Output:	With both lights on, the first triangle is black and the second triangle is shaded gray with the lightest portion at the vertex pointing toward the bottom of the window raster. Only the ambient light source and both triangles are black. The background color is red.

▼ light_ts_modclip_amb_facet

Test Types:	INDEX, SM
Description:	Renders tristrip that is model clipped, an ambient light source, and index mode. Tries per facet lighting. For three values for the ambient reflection coefficient, try three values for the light color. A tristrip composed of three triangles is rendered.

Attributes Tested:	`XGL_3D_CTX_MODEL_CLIP_PLANE_NUM`
	`XGL_3D_CTX_MODEL_CLIP_PLANES`
	and Table 15-2, Column B at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_triangle_strip`
	`xgl_object_get`

Output:	The tristrips form a house shape. The tristrips may (1) blend into the red background, or (2) the second tristrip, (that is, right side of the *roof*) is gray while the rest of the house is black.

▼ light_marker_types_selector

Test Types:	INDEX, SM
Description:	Loops through the eight different pre-defined marker types, dot, plus, asterisk, circle, cross, square, bowtie_ne, and bowtie_nw, while changing the point type from F3D to color F3D and the color selection between context and marker color vertex.

Attributes Tested:	`XGL_CTX_MARKER_SCALE_FACTOR`
	`XGL_CTX_MARKER_COLOR_SELECTOR`
	`XGL_CTX_MARKER`

Operators Tested:	`xgl_multimarker`

Output:	Renders three markers expecting either the default context color which is green, or the vertex marker colors which are Yellow, Cyan and Magenta.

light_ln_types_selector_rgb

Test Types:	RGB, SM
Description:	Loops through point types with float 3D and color float 3D while changing the line color interpolation between true and false, and the line color selection between context and vertex attributes
Attributes Tested:	XGL_3D_CTX_LINE_COLOR_INTERP XGL_CTX_LINE_COLOR_SELECTOR
Operators Tested:	xgl_multipolyline
Output:	Renders a line segment composed of 12 points which means 11 segments expecting either the line default context color which is green or the vertex colors interpolated which are Yellow, Cyan, Magenta, Orange, Brown, Violet, Olive, Grey, Mint Green, Light Red, Light Blue and White.

light_marker_types_selector_rgb

Test Types:	RGB, SM
Description:	Loops through the seven different pre-defined marker types, plus, asterisk, circle, cross, square, bowtie_ne, and bowtie_nw, while changing the point type from F3D to color F3D and the color selection between context and marker color vertex.
Attributes Tested:	XGL_CTX_MARKER_SCALE_FACTOR XGL_CTX_MARKER_COLOR_SELECTOR XGL_CTX_MARKER
Operators Tested:	xgl_multimarker
Output:	Renders three markers expecting either the default context color which is green, or the vertex marker colors which are Yellow, Cyan and Magenta.

light_msp_types_selector_rgb

Test Types:	RGB, SM
Description:	Loops through the five different point types with F3D which include normal and color information but no flag information while changing the facet types among the four possibilities of XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_NORMAL and XGL_FACET_COLOR_NORMAL, the illumination model which could be among the four possibilies of XGL_ILLUM_NONE, XGL_ILLUM_NONE_INTERP_COLOR, XGL_ILLUM_PER_FACET and XGL_ILLUM_PER_VERTEX and the four possible candidates for surface front color selection which are XGL_SURF_COLOR_CONTEXT, XGL_SURF_COLOR_FACET, XGL_SURF_COLOR_VERTEX_ILLUM_DEP and XGL_SURF_COLOR_VERTEX_ILLUM_INDEP. Two light source illuminate the house shaped multisimple polygon. One light source is ambient and the other is directional.
Attributes Tested:	XGL_CTX_SURF_FRONT_COLOR_SELECTOR XGL_CTX_SURF_FRONT_COLOR XGL_3D_CTX_SURF_FRONT_ILLUMINATION XGL_3D_CTX_SURF_FRONT_LIGHT_TYPE XGL_3D_CTX_LIGHT_NUM XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT XGL_3D_CTX_LIGHT_SWITCHES XGL_LIGHT_TYPE XGL_3D_CTX_SURF_FRONT_SPECULAR_COLOR XGL_3D_CTX_SURF_FRONT_AMBIENT XGL_3D_CTX_SURF_FRONT_DIFFUSE XGL_3D_CTX_SURF_FRONT_SPECULAR XGL_3D_CTX_SURF_FRONT_SPECULAR_POWER
Operators Tested:	xgl_multi_simple_polygon
Output:	Renders house shaped multisimple polygon with colors which may vary from the context color to the facet colors to the interpolation of the vertex colors.

▼ light_qm_types_selector_rgb

Test Types:	RGB, SM
Description:	Loops through the five different point types with F3D which include normal and color information but no flag information while changing the facet types among the four possibilities of XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_NORMAL and XGL_FACET_COLOR_NORMAL, the illumination model which could be XGL_ILLUM_NONE_INTERP_COLOR, XGL_ILLUM_PER_FACET and XGL_ILLUM_PER_VERTEX and the four possible candidates for surface front color selection which are XGL_SURF_COLOR_CONTEXT, XGL_SURF_COLOR_FACET, XGL_SURF_COLOR_VERTEX_ILLUM_DEP and XGL_SURF_COLOR_VERTEX_ILLUM_INDEP. Two light source illuminate the quadmesh. One light source is ambient and the other is directional.
Attributes Tested:	XGL_CTX_SURF_FRONT_COLOR_SELECTOR XGL_CTX_SURF_FRONT_COLOR XGL_3D_CTX_SURF_FRONT_ILLUMINATION XGL_3D_CTX_SURF_FRONT_LIGHT_TYPE XGL_3D_CTX_LIGHT_NUM XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT XGL_3D_CTX_LIGHT_SWITCHES XGL_LIGHT_TYPE XGL_3D_CTX_SURF_FRONT_SPECULAR_COLOR XGL_3D_CTX_SURF_FRONT_AMBIENT XGL_3D_CTX_SURF_FRONT_DIFFUSE XGL_3D_CTX_SURF_FRONT_SPECULAR XGL_3D_CTX_SURF_FRONT_SPECULAR_POWER
Operators Tested:	xgl_quadrilateral_mesh
Output:	Renders quadmesh with colors which may vary from the context color to the facet colors to the interpolation of the vertex colors.

light_ts_types_selector_rgb

Test Types:	RGB, SM
Description:	Loops through the five different point types with F3D which include normal and color information but no flag information while changing the facet types among the four possibilities of XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_NORMAL and XGL_FACET_COLOR_NORMAL, the illumination model which could be among the four possibilies of XGL_ILLUM_NONE, XGL_ILLUM_NONE_INTERP_COLOR, XGL_ILLUM_PER_FACET and XGL_ILLUM_PER_VERTEX and the four possible candidates for surface front color selection which are XGL_SURF_COLOR_CONTEXT, XGL_SURF_COLOR_FACET, XGL_SURF_COLOR_VERTEX_ILLUM_DEP and XGL_SURF_COLOR_VERTEX_ILLUM_INDEP. Two light source illuminate the quadmesh. One light source is ambient and the other is directional.
Attributes Tested:	XGL_CTX_SURF_FRONT_COLOR_SELECTOR XGL_CTX_SURF_FRONT_COLOR XGL_3D_CTX_SURF_FRONT_ILLUMINATION XGL_3D_CTX_SURF_FRONT_LIGHT_TYPE XGL_3D_CTX_LIGHT_NUM XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT XGL_3D_CTX_LIGHT_SWITCHES XGL_LIGHT_TYPE XGL_3D_CTX_SURF_FRONT_SPECULAR_COLOR XGL_3D_CTX_SURF_FRONT_AMBIENT XGL_3D_CTX_SURF_FRONT_DIFFUSE XGL_3D_CTX_SURF_FRONT_SPECULAR XGL_3D_CTX_SURF_FRONT_SPECULAR_POWER
Operators Tested:	xgl_triangle_strip
Output:	Renders tristrip with colors which may vary from the context color to the facet colors to the interpolation of the vertex colors.

▼ light_tstar_types_selector_rgb

Test Types: RGB, SM

Description: Loops through the five different point types with F3D which include normal and color information but no flag information while changing the facet types among the four possibilities of XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_NORMAL and XGL_FACET_COLOR_NORMAL, the illumination model which could be among the four possibilies of XGL_ILLUM_NONE, XGL_ILLUM_NONE_INTERP_COLOR, XGL_ILLUM_PER_FACET and XGL_ILLUM_PER_VERTEX and the four possible candidates for surface front color selection which are XGL_SURF_COLOR_CONTEXT, XGL_SURF_COLOR_FACET, XGL_SURF_COLOR_VERTEX_ILLUM_DEP and XGL_SURF_COLOR_VERTEX_ILLUM_INDEP. Two light source illuminate the quadmesh. One light source is ambient and the other is directional.

Attributes Tested:
XGL_CTX_SURF_FRONT_COLOR_SELECTOR
XGL_CTX_SURF_FRONT_COLOR
XGL_3D_CTX_SURF_FRONT_ILLUMINATION
XGL_3D_CTX_SURF_FRONT_LIGHT_TYPE
XGL_3D_CTX_LIGHT_NUM
XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT
XGL_3D_CTX_LIGHT_SWITCHES
XGL_LIGHT_TYPE
XGL_3D_CTX_SURF_FRONT_SPECULAR_COLOR
XGL_3D_CTX_SURF_FRONT_AMBIENT
XGL_3D_CTX_SURF_FRONT_DIFFUSE
XGL_3D_CTX_SURF_FRONT_SPECULAR
XGL_3D_CTX_SURF_FRONT_SPECULAR_POWER

Operators Tested: xgl_triangle_strip only as tristar

Output: Renders tristar with colors which may vary from the context color to the facet colors to the interpolation of the vertex colors.

Table 15-1 Lighting Attributes Tested - Set 1

Column A	Column B	Column C
XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_SURF_FRONT_COLOR
XGL_3D_CTX_SURF_FRONT_ ILLUMINATION	XGL_3D_CTX_SURF_FRONT_ ILLUMINATION	XGL_3D_CTX_SURF_FRONT_ ILLUMINATION
XGL_ILLUM_PER_FACET	XGL_ILLUM_PER_FACET	XGL_ILLUM_PER_FACET
XGL_3D_CTX_SURF_FRONT_LIGHT _COMPONENT	XGL_3D_CTX_HLHSR_MODE	XGL_3D_CTX_SURF_FRONT_LIGHT _COMPONENT
XGL_LIGHT_ENABLE_COMP_ AMBIENT	XGL_HLHSR_Z_BUFFER	XGL_LIGHT_ENABLE_COMP_ AMBIENT
XGL_3D_CTX_SURF_FRONT_LIGHT _TYPE	XGL_3D_CTX_SURF_FRONT_LIGHT _COMPONENT	XGL_3D_CTX_SURF_FRONT_LIGHT _TYPE
XGL_LIGHT_ENABLE_TYPE_ AMBIENT	XGL_LIGHT_ENABLE_COMP_ AMBIENT	XGL_LIGHT_ENABLE_TYPE_ AMBIENT
XGL_3D_CTX_LIGHT_NUM	XGL_3D_CTX_SURF_FRONT_LIGHT _TYPE	XGL_3D_CTX_LIGHT_NUM
XGL_3D_CTX_LIGHT_SWITCHES	XGL_LIGHT_ENABLE_TYPE_ AMBIENT	XGL_3D_CTX_LIGHT_SWITCHES
XGL_3D_CTX_LIGHTS	XGL_3D_CTX_LIGHT_NUM	XGL_3D_CTX_LIGHTS
XGL_LIGHT_TYPE	XGL_3D_CTX_LIGHT_SWITCHES	XGL_LIGHT_TYPE
XGL_LIGHT_AMBIENT	XGL_3D_CTX_LIGHTS	XGL_LIGHT_AMBIENT
XGL_3D_CTX_SURF_FRONT_ AMBIENT	XGL_LIGHT_TYPE	XGL_3D_CTX_SURF_FRONT_ AMBIENT
XGL_LIGHT_COLOR	XGL_LIGHT_AMBIENT	XGL_LIGHT_COLOR

Table 15-2 Lighting Attributes Tested - Set 2

Column A	Column B	Column C
XGL_CTX_SURF_FRONT_COLOR	XGL_3D_CTX_SURF_FRONT_ ILLUMINATION	XGL_CTX_SURF_FRONT_COLOR
XGL_3D_CTX_SURF_FRONT_ ILLUMINATION	XGL_ILLUM_PER_FACET	XGL_3D_CTX_SURF_FRONT_ ILLUMINATION
XGL_ILLUM_PER_FACET	XGL_3D_CTX_SURF_FRONT_ DIFFUSE	XGL_ILLUM_PER_FACET
XGL_3D_CTX_SURF_FRONT_LIGHT _COMPONENT	XGL_3D_CTX_SURF_FRONT_ SPECULAR	XGL_3D_CTX_SURF_FRONT_LIGHT _COMPONENT
XGL_LIGHT_ENABLE_COMP_ DIFFUSE	XGL_3D_CTX_SURF_FRONT_ SPECULAR_POWER	XGL_LIGHT_ENABLE_COMP_ DIFFUSE
XGL_LIGHT_ENABLE_COMP_ SPECULAR	XGL_3D_CTX_LIGHT_NUM	XGL_LIGHT_ENABLE_COMP_ SPECULAR
XGL_3D_CTX_SURF_FRONT_LIGHT _TYPE	XGL_3D_CTX_SURF_FRONT_ SPECULAR_COLOR	XGL_3D_CTX_SURF_FRONT_LIGHT _TYPE
XGL_LIGHT_ENABLE_TYPE_ POSITIONAL	XGL_3D_CTX_LIGHT_SWITCHES	XGL_LIGHT_ENABLE_TYPE_ DIRECTIONAL
XGL_3D_CTX_LIGHT_NUM	XGL_3D_CTX_SURF_FRONT_LIGHT _COMPONENT	XGL_3D_CTX_LIGHT_NUM
XGL_3D_CTX_LIGHT_SWITCHES	XGL_LIGHT_ENABLE_COMP_ AMBIENT	XGL_3D_CTX_LIGHT_SWITCHES
XGL_3D_CTX_SURF_FRONT_ SPECULAR_COLOR	XGL_3D_CTX_SURF_FRONT_LIGHT _TYPE	XGL_3D_CTX_SURF_FRONT_ SPECULAR_COLOR
XGL_3D_CTX_LIGHTS	XGL_LIGHT_ENABLE_TYPE_ AMBIENT	XGL_3D_CTX_SURF_FRONT_ DIFFUSE
XGL_LIGHT_TYPE	XGL_3D_CTX_LIGHTS	XGL_3D_CTX_SURF_FRONT_ SPECULAR
XGL_LIGHT_POSITIONAL	XGL_LIGHT_TYPE	XGL_3D_CTX_SURF_FRONT_ SPECULAR_POWER
XGL_3D_CTX_SURF_FRONT_ DIFFUSE	XGL_LIGHT_AMBIENT	XGL_3D_CTX_LIGHTS

Table 15-2 Lighting Attributes Tested - Set 2 *(Continued)*

Column A	**Column B**	**Column C**
XGL_3D_CTX_SURF_FRONT_ SPECULAR	XGL_3D_CTX_SURF_FRONT_ AMBIENT	XGL_LIGHT_TYPE
XGL_3D_CTX_SURF_FRONT_ SPECULAR_POWER	XGL_LIGHT_COLOR	XGL_LIGHT_DIRECTIONAL
XGL_LIGHT_COLOR		XGL_LIGHT_DIRECTION
XGL_LIGHT_ATTENUATION_1		XGL_LIGHT_COLOR
XGL_LIGHT_ATTENUATION_2		
XGL_LIGHT_POSITION		

Line Test Descriptions 16

This chapter describes the Line test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section, “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ gc_line0

Test Types:	RGB, SM
Description:	Tries a gcache line with 7 points in one point list, and another in two point lists with 7 points included in each list.
Attributes Tested:	`XGL_GCACHE`

Operators Tested:	`xgl_object_create`
	`xgl_object_destroy`
	`xgl_gcache_multipolyline`
	`xgl_context_display_gcache`

Output: The first gcached line has three local maxima and two local minima. The second gcached collection of point lists has the previously rendered line plus the same line drawn translated below the previously rendered line by 100.

▼ gc_line_attr

Test Types: RGB, SM

Description: Checks that validation of attributes during gcache display is done as specified. Checks also that the attributes that are not validated do not persist.

Attributes Tested:
`XGL_3D_CTX_MODEL_CLIP_PLANE_NUM`
`XGL_CACHE_ATTR_STATE_DIFFERENT`
`XGL_CACHE_DISPLAY_OK`
`XGL_CACHE_NOT_CHECKED`
`XGL_CTX_LINE_COLOR`
`XGL_CTX_LINE_WIDTH_SCALE_FACTOR`
`XGL_GCACHE`
`XGL_GCACHE_IS_EMPTY`

Operators Tested:
`xgl_object_create`
`xgl_gcache_multipolyline`
`xgl_context_display_gcache`
`xgl_object_set`
`xgl_object_get`

Output: Two gcached collection of point lists with 7 points in each point list with each line rendered an exact copy of the previous line except the line has been translated below the previous line by 100; and each line containing three local maxima and two local minima.

▼ line0

Test Types: INDEX, SM

Description: Renders a line with a negative slope, for both *win_ras* and *mem_ras* from 20, 20 to 300, 300 utilizing point type `XGL_PT_I2D`, and a bounding box of NULL. The color map used is the default, so only two colors, black and white are available.

Attributes Tested:	XGL_CTX_LINE_COLOR XGL_MEM_RAS XGL_RAS_DEPTH XGL_RAS_WIDTH XGL_RAS_HEIGHT
Operators Tested:	xgl_object_set xgl_multipolyline
Output:	A negative sloped white line

▼ line1

Test Types:	INDEX, SM
Description:	Renders two white lines with negative slopes and one white line with a positive slope against *win_ras* and *mem_ras* utilizing point type XGL_PT_I2D. The color map used is the default, so only two colors, black and white are available.
Attributes Tested:	XGL_CTX_LINE_COLOR XGL_MEM_RAS XGL_RAS_DEPTH XGL_RAS_WIDTH XGL_RAS_HEIGHT
Operators Tested:	xgl_object_set xgl_multipolyline
Output:	Three white lines with two lines having a negative slope and one line having a positive slope. After the rendering of a single white line, the screen is cleared and another white line is drawn.

▼ line2

Test Types:	INDEX, SM
Description:	Renders six white lines against both *win_ras* and *mem_ras* with point type XGL_PT_I2D. The first three lines are solids and the last three are dashed pattern lines. The color map used is the default, so only two colors, black and white are available.
Attributes Tested:	XGL_CTX_LINE_COLOR XGL_CTX_LINE_PATTERN XGL_CTX_LINE_STYLE

	XGL_LINE_PATTERNED XGL_MEM_RAS XGL_RAS_DEPTH XGL_RAS_WIDTH XGL_RAS_HEIGHT
Operators Tested:	xgl_object_set xgl_multipolyline xgl_object_get
Output:	A line is rendered and then the screen is cleared. The first three lines rendered are solid. The next three lines rendered are dashed.

line3

Test Types: INDEX, SM

Description: Tries thin and fat shaded lines utilizing the line attribute XGL_3D_CTX_LINE_COLOR_INTERP. Loops through two settings for line width, 1.0 and 5.0, with seven variations for shading the individual segments for the line. The seven different arrangements for rendering color of the lines are:

- Same color at each end
- Different color at each end
- Reversed colors
- Same colors, first segment
- Different colors, each segment
- Multiple polylines
- Single pixel

Attributes Tested: XGL_3D_CTX_LINE_COLOR_INTERP
XGL_CTX_LINE_WIDTH_SCALE_FACTOR

Operators Tested: xgl_object_set
xgl_multipolyline

Output: Lines with arrangements mentioned above appear like:

- Red negative sloped line
- Horizontal line red, lime green, yellow, dark blue, purple light blue
- Horizontal line purple, dark blue, yellow, lime green, red
- First segment—red; second segment—red, lime green, yellow, dark blue, purple, light blue

- First segment—red, lime green, yellow, dark blue; second segment—dark blue, purple, light blue, grey
- Two horizontal lines—top: red, lime green, yellow, blue; bottom: dark blue, purple, light blue, grey
- One purple pixel

line4

Test Types: INDEX, SM

Description: Checks three cases for the correct management of HLHSR for the z value: (1) the first line is in front, (2) the second line is in front, and (3) both lines have the same z value. Tests loops through six variation of lines and they are:

- Vertical
- Horizontal
- 45 degrees
- Arbitrary slope
- Multi-segmented
- Multipolyline

Attributes Tested: XGL_3D_CTX_HLHSR_MODE
XGL_CTX_LINE_COLOR
XGL_CTX_NEW_FRAME_ACTION
XGL_CTX_NEW_FRAME_CLEAR
XGL_CTX_NEW_FRAME_HLHSR_ACTION
XGL_HLHSR_ZBUFFER

Operators Tested: xgl_object_set
xgl_multipolyline

Output: Lines appear as described above with loop 1—all lines red; loop 2—all lines green; loop 3—all lines green

line5

Test Types: RGB, SM

Description: Same test as *line3* except this test is RGB

Attributes Tested: XGL_3D_CTX_LINE_COLOR_INTERP
XGL_CTX_BACKGROUND_COLOR
XGL_CTX_LINE_WIDTH_SCALE_FACTOR

Operators Tested: xgl_object_set
xgl_multipolyline

Output: Line variations as mentioned for *line3* with the color scheme as follows:

- Red negative sloped line
- Horizontal line blue shaded to red
- Horizontal line brown shaded to blue
- Red shaded to orange shaded to yellow shaded to green shaded to light blue
- Two horizontal lines: top—red shaded to green; bottom—green shaded to blue
- One blue pixel

▼ line6

Test Types: RGB, SM
Description: Same as *line4* except this test is RGB
Attributes Tested: XGL_3D_CTX_HLHSR_MODE
XGL_CTX_LINE_COLOR
XGL_CTX_NEW_FRAME_ACTION
XGL_CTX_NEW_FRAME_CLEAR
XGL_CTX_NEW_FRAME_HLHSR_ACTION
XGL_HLHSR_ZBUFFER
Operators Tested: xgl_object_set
xgl_multipolyline
Output: Lines appear as described in *line4* with loop 1—all lines brown; loop 2—all lines green; loop 3—all lines green

▼ line7

Test Types: RGB, SM
Description: Renders three horizontal green dashed lines and three vertical green dashed lines
Attributes Tested: XGL_CTX_LINE_STYLE
XGL_CTX_LINE_PATTERN
XGL_LINE_SOLID
XGL_LINE_PATTERNED
Operators Tested: xgl_object_set
xgl_multipolyline
Output: Three horizontal green dashed lines and three vertical green dashed lines

▼ line8

Test Types:	RGB, SM
Description:	Renders light blue solid and dashed pattern horizontal lines

Attributes Tested:	`XGL_CTX_LINE_PATTERN`
	`XGL_CTX_LINE_STYLE`
	`XGL_CTX_LINE_WIDTH_SCALE_FACTOR`
	`XGL_LINE_PATTERNED`

Operators Tested:	`xgl_object_set` `xgl_multipolyline`
Output:	A light blue solid horizontal line, dashed horizontal light blue line, solid light blue line, and finally a wide dashed horizontal light blue line

▼ line9

Test Types:	RGB, SM
Description:	Renders a collection of lines utilizing the same values for x1, y1, x2 but changing y2 through each loop by adding 10 to the previous y2 value. Renders a collection of 50 random horizontal lines with random colors and random widths. Renders a collection of 50 random vertical lines with random colors and random widths.

Attributes Tested:	`XGL_CTX_LINE_STYLE`
	`XGL_CTX_LINE_WIDTH_SCALE_FACTOR`
	`XGL_LINE_SOLID`

Operators Tested:	`xgl_object_set` `xgl_multipolyline`
Output:	Renders a collection of RGB lines like the spokes of a wheel from a horizontal location to practically a 90-degree spoke. Renders a variety of different-colored wide horizontal lines and finally a variety of different-colored vertical horizontal lines.

▼ line10

Test Types:	CM, RGB

Description:	Checks RGB line-pattern balancing. Uses random colors and variable lengths. Checks horizontal and vertical cases. Does token check of wide-patterned lines.
Attributes Tested:	XGL_CTX_LINE_COLOR XGL_CTX_LINE_PATTERN XGL_CTX_LINE_STYLE XGL_CTX_LINE_WIDTH_SCALE_FACTOR XGL_LINE_PATTERNED XGL_LINE_SOLID XGL_LPAT XGL_LPAT_BALANCED_DASH XGL_LPAT_BALANCED_SEGMENT XGL_LPAT_BAL_DASH_0 XGL_LPAT_BAL_DASH_1 XGL_LPAT_COORD_SYS XGL_LPAT_DATA XGL_LPAT_DATA_SIZE XGL_LPAT_DC XGL_LPAT_OFFSET XGL_LPAT_STYLE
Operators Tested:	xgl_object_set xgl_multipolyline
Output:	Two frames of horizontal-patterned balanced lines, with the length of each successive horizontal line increased by 12 followed by two frames of vertical-patterned balanced lines, with each successive vertical line height increased by 12. The correct balancing is indicated by the gaps in each line, which when correct, form a straight undisturbed margin of background color across all the lines rendered in the frame. The final frame is a single wide-patterned line.

▼ line11

Test Types:	INDEX, SM
Description:	Checks every pixel in a horizontal and simple alternate-patterned dotted line

Attributes Tested:	XGL_CTX_LINE_ALT_COLOR XGL_CTX_LINE_COLOR XGL_CTX_LINE_PATTERN XGL_CTX_LINE_STYLE XGL_LINE_ALT_PATTERNED
Operators Tested:	xgl_object_set xgl_multipolyline
Output:	Horizontal alternate dotted line, with the even-colored dots red and the gaps green

line12

Test Types:	INDEX, SM
Description:	Sets the line color to 255. Loops through all of the possible plane mask values, 0–255. Clears the plane mask by setting it to -1. Sets the plane mask to the loop value. Checks for line color for the plane mask value of 255.
Attributes Tested:	XGL_CTX_LINE_COLOR XGL_CTX_PLANE_MASK
Operators Tested:	xgl_object_set xgl_multipolyline
Output:	Two hundred fifty five frames of a horizontal line in the line color for the loop value of 255

line13

Test Types:	INDEX, CM
Description:	Renders a trapezoid in a variety of patterned lines and checks to see that the pattern follows through properly in acute and obtuse angles. Renders a parallelogram in a variety of wide widths and checks to see that the mitered joins are properly rendered.
Attributes Tested:	XGL_CTX_LINE_COLOR XGL_CTX_LINE_PATTERN XGL_CTX_LINE_STYLE XGL_CTX_LINE_WIDTH_SCALE_FACTOR XGL_LINE_PATTERNED XGL_LINE_SOLID
Operators Tested:	xgl_object_set xgl_multipolyline

Output:	Six trapezoids, each located inside the previous one in a variety of colors and patterns. Five parallelograms in a variety of widths and colors, each located inside the previous one.

▼ line14

Test Types:	INDEX, CM
Description:	Renders a trapezoid in a variety of wide patterned lines and checks to see that both the pattern, the joins, and the width follow through the acute and obtuse angles properly.

Attributes Tested:	`XGL_CTX_LINE_COLOR`
	`XGL_CTX_LINE_PATTERN`
	`XGL_CTX_LINE_STYLE`
	`XGL_CTX_LINE_WIDTH_SCALE_FACTOR`
	`XGL_LINE_PATTERNED`
	`XGL_LINE_SOLID`

Operators Tested:	`xgl_object_set` `xgl_multipolyline`
Output:	Five trapezoids, each located inside the previous one in a variety of colors, patterns, and widths

▼ line15

Test Types:	INDEX, CM
Description:	Renders three spice jars with *3d_flag* point type by alternating the settings of edge flags on and off, varying the color, and changing the cap styles through all possibilities.

Attributes Tested:	`XGL_CAP_BUTT`
	`XGL_CAP_ROUND`
	`XGL_CAP_SQUARE`
	`XGL_CLR_EDGE_FLAG`
	`XGL_CTX_LINE_CAP`
	`XGL_CTX_LINE_COLOR`
	`XGL_CTX_LINE_STYLE`
	`XGL_CTX_LINE_WIDTH_SCALE_FACTOR`
	`XGL_LINE_SOLID`
	`XGL_SET_EDGE_FLAG`

Operators Tested: `xgl_object_set`
`xgl_multipolyline`

Output: Three spice jars outline each located within each other. The outer line color is white, the middle is red, and the inner is green. The butt styles are square, round, and butt. The on/off segments are segment, off, two segments, off, two segments, off, and last segment on. A marker indicates the commencement of the butt style.

▼ line16

Test Types: INDEX, CM

Description: Renders six spice jars with *3d_flag* point type by alternating the settings of edge flags on and off, varying the color, varying the line pattern, and changing the cap styles through all possibilities.

Attributes Tested: `XGL_CAP_BUTT`
`XGL_CAP_ROUND`
`XGL_CAP_SQUARE`
`XGL_CLR_EDGE_FLAG`
`XGL_CTX_LINE_CAP`
`XGL_CTX_LINE_COLOR`
`XGL_CTX_LINE_PATTERN`
`XGL_CTX_LINE_STYLE`
`XGL_CTX_LINE_WIDTH_SCALE_FACTOR`
`XGL_LINE_PATTERNED`
`XGL_LINE_SOLID`
`XGL_SET_EDGE_FLAG`

Operators Tested: `xgl_object_set`
`xgl_multipolyline`

Output: Six spice jars outline each located within each other. The line colors starting from the outward to innermost are white, red, lime green, dark blue, yellow, and light blue. The butt styles are square, round, and butt. The on/off segments are segment, off, two segments, off, two segments, off, and last segment on. The line pattern types from the outward are dotted, dashed, dashed-dotted, dash-dot, dash-dot-dotted, and long dashed.

line17

Test Types: INDEX, CM

Description: Renders wide single-segment lines such that they form a spoke around an imaginary center, and checks their variety of cap styles by rendering a marker at the beginning of the cap

Attributes Tested:
XGL_CAP_BUTT
XGL_CAP_ROUND
XGL_CAP_SQUARE
XGL_CTX_LINE_CAP
XGL_CTX_LINE_COLOR
XGL_CTX_LINE_WIDTH_SCALE_FACTOR

Operators Tested:
xgl_object_set
xgl_multipolyline
xgl_object_get

Output: Fourteen wide single-segment lines. Starting with the approximately 45-degree sloped, the white line in the upper-left corner of the raster is white, red, lime green, dark blue, yellow, light blue, purple, white, red, lime green, dark blue, yellow, light blue, and purple. Also starting with this white line, the cap styles are square, round, and butt and then repeat through the lines rendered.

line18

Test Types: INDEX, CM

Description: Renders wide single-segment patterned lines so that they form a spoke around an imaginary center, and checks their variety of cap styles by rendering a marker at the beginning of the cap

Attributes Tested:
XGL_CAP_BUTT
XGL_CAP_ROUND
XGL_CAP_SQUARE
XGL_CTX_LINE_CAP
XGL_CTX_LINE_COLOR
XGL_CTX_LINE_PATTERN

	XGL_CTX_LINE_STYLE XGL_CTX_LINE_WIDTH_SCALE_FACTOR XGL_LINE_PATTERNED
Operators Tested:	xgl_object_set xgl_multipolyline
Output:	Fourteen wide single-segment lines. Starting with the approximately 45-degree sloped, the white line in the upper-left corner of the raster is white, red, lime green, dark blue, yellow, light blue, purple, white, red, lime green, dark blue, yellow, light blue, and purple. Also starting with this white line, the cap styles are square and round and then repeat through the lines rendered. The pattern types repeated through this arrangement are dotted, dashed, dashed-dotted, dash-dot, dash-dot-dotted, and long dashed.

▼ line19

Test Types:	INDEX, CM
Description:	Renders three spice jars with *2d_flag* point type alternating the settings of edge flags on and off, varying the color and the width, and changing the cap styles through all possibilities against both *win_ras* and *mem_ras*
Attributes Tested:	XGL_CAP_BUTT XGL_CAP_ROUND XGL_CAP_SQUARE XGL_CLR_EDGE_FLAG XGL_CTX_LINE_CAP XGL_CTX_LINE_COLOR XGL_CTX_LINE_STYLE XGL_CTX_LINE_WIDTH_SCALE_FACTOR XGL_LINE_SOLID XGL_SET_EDGE_FLAG XGL_MEM_RAS XGL_RAS_DEPTH XGL_RAS_WIDTH XGL_RAS_HEIGHT
Operators Tested:	xgl_object_set xgl_multipolyline

Output: Three spice jars outline each located within each other. The outer line color is white, the middle is red, and the inner is green. The butt styles are square, round, and butt. The on / off segments are segment, off, two segments, off, two segments, off, and last segment on. A marker indicates the commencement of the butt style.

line20

Test Types: INDEX, CM

Description: Renders two outlines of geckos: one with *i2d_flag* point type alternating the settings of edge flags on and off with a round cap style, a width of 5.0, and a red line color. The other outline has all edges on, beveled joins, green line color, a width of 7.0, and butt cap. Markers are displayed at each of the vertexes to easily check the joins.

Attributes Tested:
`XGL_CAP_BUTT`
`XGL_CAP_ROUND`
`XGL_CAP_SQUARE`
`XGL_CLR_EDGE_FLAG`
`XGL_CTX_LINE_CAP`
`XGL_CTX_LINE_COLOR`
`XGL_CTX_LINE_JOIN`
`XGL_CTX_LINE_STYLE`
`XGL_CTX_LINE_WIDTH_SCALE_FACTOR`
`XGL_JOIN_BEVEL`
`XGL_LINE_SOLID`
`XGL_SET_EDGE_FLAG`

Operators Tested:
`xgl_object_set`
`xgl_multipolyline`

Output: Two outlines of geckos, one with red line color and alternating on / off line segments, and the other with a *blunt* join, all segments on and a green line color

line21

Test Types: INDEX, CM

Description: Sets the join miter limit so low that we expect beveled joins and observe the joins for acute and obtuse angles using *flag_pt_2d* point type

Attributes Tested: XGL_CTX_LINE_COLOR
XGL_CTX_LINE_JOIN
XGL_CTX_LINE_MITER_LIMIT
XGL_CTX_LINE_STYLE
XGL_CTX_LINE_WIDTH_SCALE_FACTOR
XGL_JOIN_BEVEL
XGL_JOIN_MITER
XGL_LINE_SOLID
XGL_SET_EDGE_FLAG

Operators Tested: xgl_object_set
xgl_multipolyline

Output: The raster is covered with two wide segmented lines in a variety of colors with acute and obtuse angles. Markers indicate the position of the vertexes.

line22

Test Types: INDEX, CM

Description: Extremely acute angles formed by a two-segmented line with an extremely high value for the miter limit to force knife edges. Widths are set alternating between 5.0 and 7.0 and the point type used is *flag_2d*.

Attributes Tested: XGL_CTX_LINE_COLOR
XGL_CTX_LINE_JOIN
XGL_CTX_LINE_MITER_LIMIT
XGL_CTX_LINE_STYLE
XGL_CTX_LINE_WIDTH_SCALE_FACTOR
XGL_JOIN_MITER
XGL_LINE_SOLID
XGL_SET_EDGE_FLAG

Operators Tested: xgl_object_set
xgl_multipolyline

Output: The raster is covered with v-shaped wide lines with knife edges in a variety of colors.

line24

Test Types: INDEX, CM

Description:	Renders a simple white index horizontal line inside a window raster whose x, y, and z values range from 0 to 1 via the setting for the window bounds under a virtual device mapping system of ASPECT
Attributes Tested:	XGL_CTX_VDC_MAP XGL_CTX_VDC_WINDOW XGL_VDC_MAP_ASPECT
Operators Tested:	xgl_object_set xgl_multipolyline
Output:	A simple white horizontal line

▼ line25

Test Types:	INDEX, CM
Description:	Tests line attributes for line width and line style inside a window raster whose x, y, and z values range from 0 to 1 via the setting for the window bounds and an orientation such that y is up and z is nearly below a virtual device mapping system of ASPECT
Attributes Tested:	XGL_CTX_BACKGROUND_COLOR XGL_CTX_LINE_COLOR XGL_CTX_LINE_PATTERN XGL_CTX_LINE_STYLE XGL_CTX_LINE_WIDTH_SCALE_FACTOR XGL_CTX_SURF_FRONT_COLOR XGL_CTX_VDC_MAP XGL_VDC_MAP_ASPECT XGL_CTX_VDC_ORIENTATION XGL_Y_UP_Z_TOWARD XGL_CTX_VDC_WINDOW XGL_LINE_PATTERNED XGL_LINE_SOLID
Operators Tested:	xgl_object_set xgl_multipolyline
Output:	A variety of patterned and colored horizontal lines forming two columns in the window raster

line26

Test Types: INDEX, CM

Description: Pushes attributes, renders, pushes new attributes, renders, pops attributes, expects the just set attributes, pops again and expects the originally set attributes. Uses this scheme to verify the push / pop scheme for line style, line width, and line color.

Attributes Tested:
XGL_CTX_BACKGROUND_COLOR
XGL_CTX_LINE_COLOR
XGL_CTX_LINE_PATTERN
XGL_CTX_LINE_STYLE
XGL_CTX_LINE_WIDTH_SCALE_FACTOR
XGL_CTX_VDC_MAP
XGL_VDC_MAP_ASPECT
XGL_CTX_VDC_ORIENTATION
XGL_Y_UP_Z_TOWARD
XGL_CTX_VDC_WINDOW
XGL_LINE_PATTERNED
XGL_LINE_SOLID

Operators Tested:
xgl_object_set
xgl_multipolyline
xgl_context_push
xgl_context_pop

Output: A variety of patterned and colored horizontal lines forming two columns in the window raster

line27

Test Types:	INDEX, CM
Description:	Loops through two settings for line width, 1.0 and 5.0, with seven colors for the individual segment for the line against both *win_ras* and *mem_ras*. The seven different arrangements for rendering color of the lines are: o Same color at each end o Different colors at each end o Reversed colors o Same color for first segment o Different colors for each segment o Multiple polylines o Single pixel

Attributes Tested:	`XGL_3D_CTX_LINE_COLOR_INTERP`
	`XGL_CTX_LINE_WIDTH_SCALE_FACTOR`
	`XGL_CTX_BACKGROUND_COLOR`
	`XGL_CMAP_COLOR_TABLE_SIZE`
	`XGL_CMAP_COLOR_TABLE`
	`XGL_MEM_RAS`
	`XGL_RAS_DEPTH`
	`XGL_RAS_WIDTH`
	`XGL_RAS_HEIGHT`

Operators Tested:	`xgl_object_set` `xgl_multipolyline`
Output:	Lines with above arrangements with color scheme: o red negative sloped line o horizontal cyan line o horizontal red line o red for the first segment, cyan for the second segment o blue for the first segment, white for the second segment o two horizontal lines; top is blue, bottom is white o like a red pixel

Marker Test Descriptions 17

This chapter describes the Marker test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section, “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ gc_marker_simple_rgb

Test Types:	RGB, SM
Description:	Draws a marker into a gcache and checks to see that it is rendered when gcache is displayed
Attributes Tested:	Uses default attributes

Operators Tested:	`xgl_object_create`
	`xgl_object_get`
	`xgl_gcache_multimarker`
	`xgl_context_display_gcache`
	`xgl_object_destroy`

Output:	Three white plus markers displayed

gc_marker_pttypes_rgb

Test Types: RGB, SM
Description: Tests different point types for gcache markers and tests one gcache marker with 25 points
Attributes Tested: XGL_CTX_MARKER
XGL_CTX_MARKER_SCALE_FACTOR
Operators Tested: xgl_object_create
xgl_object_get
xgl_gcache_multimarker
xgl_context_display_gcache
xgl_object_destroy
Output: Nine cross markers displayed four times with different point types and one 25 plus markers displayed once. Point types used are F3D and F3H.

marker_2d_default

Test Types: INDEX, SM
Description: Tests the correct rendering of 2D markers using default marker attributes
Attributes Tested: XGL_CTX_MARKER
Operators Tested: xgl_object_get
xgl_object_set
xgl_multimarker
Output: Three asterisk markers displayed with default white color

marker_attr

Test Types: INDEX, SM
Description: Tests index color multimarkers with nondefault marker attributes. Marker color, size, and description are varied.
Attributes Tested: XGL_CTX_MARKER
XGL_CTX_MARKER_SCALE_FACTOR
XGL_CTX_MARKER_COLOR
Operators Tested: xgl_object_set
xgl_multimarker
Output: Five markers displayed as dot, plus, asterisk, circle, and cross with different color and scale factors. The scale factor is in the inner loop.

▼ marker_pttypes

Test Types: INDEX, SM
Description: Tests multimarkers rendered with different point types
Attributes Tested: XGL_CTX_MARKER
XGL_CTX_MARKER_SCALE_FACTOR
Operators Tested: xgl_object_set
xgl_multimarker
Output: Nine cross markers rendered six times with a scale of 10.0 and varied point types:
XGL_PT_I2D
XGL_PT_I2H
XGL_PT_F2D
XGL_PT_F2H
XGL_PT_F3D
XGL_PT_F3H
Four hundred plus markers rendering in one multimarker call with a scale of 12.3.

▼ marker_hlhsr

Test Types: INDEX, SM
Description: Tests markers' hidden line-hidden surface removal. Draws markers at identical position but with different depth and checks that only the front marker shows up. Depth combination is varied.
Attributes Tested: XGL_3D_CTX_HLHSR_MODE
XGL_CTX_MARKER_COLOR
XGL_CTX_MARKER
XGL_CTX_MARKER_SCALE_FACTOR
Operators Tested: xgl_object_set
xgl_multimarker
Output: First frame: four circle markers in red. Second and third frame: four circle markers in green. Fourth frame: two circle markers, one in red and the other one in green.

▼ marker_2d_user

Test Types: INDEX, SM

Description:	Tests user-defined markers. Constructs a few 2D marker descriptions and renders a single marker using each of them. Checks that they're rendered correctly (checks each line segment).
Attributes Tested:	`XGL_MARKER_DESCRIPTION` `XGL_CTX_MARKER_SCALE_FACTOR`
Operators Tested:	`xgl_object_set` `xgl_multimarker`
Output:	Four user defined white markers: a rotated L shape, an asterisk, a square, and a triangle

▼ marker_2d_plane_mask

Test Types:	INDEX, SM
Description:	Tests that the `XGL_CTX_PLANE_MASK` attribute applies to 2D markers

Attributes Tested:	`XGL_CTX_MARKER`
	`XGL_CTX_MARKER_SCALE_FACTOR`
	`XGL_CTX_MARKER_COLOR`
	`XGL_CTX_PLANE_MASK`

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_multimarker`

Output:	Four circle markers displayed

▼ marker_2d_ras_op

Test Types:	INDEX, SM
Description:	Tests that the `XGL_CTX_ROP` attribute applies to 2D markers

Attributes Tested:	`XGL_CTX_MARKER`
	`XGL_CTX_MARKER_SCALE_FACTOR`
	`XGL_CTX_MARKER_COLOR`
	`XGL_CTX_ROP`

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_multimarker`

Output:	Four plus markers displayed in color: black, white, black, green, green, red, red, blue, blue, light green, light green, light blue, light blue, light red, light red, white

marker_2d_default_rgb

Test Types:	RGB, SM
Description:	Tests the correct rendering of 2D markers using default marker attributes
Attributes Tested:	XGL_CTX_MARKER
Operators Tested:	xgl_object_get xgl_object_set xgl_multimarker
Output:	Three asterisk markers displayed with default green color

marker_attr_rgb

Test Types:	RGB SM
Description:	Tests RGB multimarkers with nondefault marker attributes. Marker color, size, and description are varied.
Attributes Tested:	XGL_CTX_MARKER XGL_CTX_MARKER_SCALE_FACTOR XGL_CTX_MARKER_COLOR
Operators Tested:	xgl_object_set xgl_multimarker
Output:	Four markers displayed as dot, plus, asterisk, circle, and cross with different colors and scale factors. The scale factor is in the inner loop.

marker_pttypes_rgb

Test Types:	RGB, SM
Description:	Tests multimarkers rendered with different point types
Attributes Tested:	XGL_CTX_MARKER XGL_CTX_MARKER_SCALE_FACTOR
Operators Tested:	xgl_object_set xgl_multimarker
Output:	Nine cross markers rendered six times with a scale of 10.0 and varied point types: XGL_PT_I2D XGL_PT_I2H XGL_PT_F2D XGL_PT_F2H XGL_PT_F3D

XGL_PT_F3H
Four hundred plus markers rendering in one multimarker call with a scale of 12.3.

▼ marker_hlhsr_rgb

Test Types:	RGB, SM
Description:	Tests RGB markers' hidden line-hidden surface removal. Draws markers at identical position but with different depth and checks that only the front marker shows up. Depth combination is varied.
Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_CTX_MARKER_COLOR XGL_CTX_MARKER XGL_CTX_MARKER_SCALE_FACTOR
Operators Tested:	xgl_object_set xgl_multimarker
Output:	First frame: four circle markers in red. Second and third frame: four circle markers in green. Fourth frame: two circle markers, one in red and the other one in green.

▼ marker_2d_user_rgb:

Test Types:	RGB, SM
Description:	Tests RGB user-defined 2D markers. Constructs a few marker descriptions and renders a single marker using each of them. Checks that they're rendered correctly (checks each line segment).
Attributes Tested:	XGL_MARKER_DESCRIPTION XGL_CTX_MARKER_SCALE_FACTOR
Operators Tested:	xgl_object_set xgl_multimarker
Output:	Four user defined green markers: a rotated L shape, an asterisk, a square, and a triangle

Multisimple Polygon Test Descriptions 18

This chapter describes the Multisimple Polygon test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section, “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

Note – Each multisimple polygon test has an equivalent polygon version that is as similar as possible. Also, when gcache primitives were added to their respective areas (pre-3.0), multi* were moved to the *mspolygon* directory.

▼ multipg_simple

Test Types:	INDEX, SM
Description:	Tries two solid front facing polygons with exactly the same vertex data except their normals and their facet types differ
Attributes Tested:	`XGL_SURF_FILL_SOLID`

Operators Tested:	xgl_object_set xgl_multi_simple_polygon
Output:	First frame, a yellow square and second frame, a red square

▼ multipg_simple_rgb

Test Types:	RGB, SM
Description:	Tries two solid front facing polygons with exactly the same vertex data except their normals and their facet types differ
Attributes Tested:	XGL_CTX_SURF_EDGE_FLAG XGL_CTX_SURF_FRONT_FILL_STYLE XGL_SURF_FILL_SOLID
Operators Tested:	xgl_object_set xgl_multi_simple_polygon
Output:	First frame, a red square and second frame, a yellow square

▼ multipg0

Test Types:	RGB, SM
Description:	Linear depth-cueing of simple solid polygons with every polygon except the first back facing. All are coplanar, although some of the vertex data do not lie in the same z plane through the polygon. The expected surface color is a contribution from both the surface color and the depth cue color, with the surface color contribution proportional to the distance from the maximum window z boundary and the depth cue color directly related to the z value at the vertex. Expects the surface color to be closer to the surface color set of purple when the z value for all or most of the vertex data are closest to 0 or front facing. Expects the surface color to be closer to the depth cue color when the z value is closer to the maximum window bounds of 1.0. Expects the surface color to be an interpolation (shading) among more than

	one color dependent on the different z values for the individual vertex data. The depth cue color is varied through nine different variants.
Attributes Tested:	`XGL_3D_CTX_DEPTH_CUE_COLOR` and Table 18-2, Column A at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_multi_simple_polygon`
	`xgl_object_get`

Output:	The first polygon list has a constant z value of 0, the sole point list for which all polygons are front facing. Consequently, its surface color is always the surface color set with no contribution from the depth cue color; contains only one polygon, which is square; and is always solid unshaded purple. The second polygon list contains only one polygon, has a constant z value of the maximum z window boundary, is shaped like a house rotated so that its center passes through the line x = y and by its z-value is guaranteed to be whatever the set depth color is. The third polygon list is composed of two polygons, a triangle and a rhombus both near the top of the window, and both with a constant z value of 0.3, which makes their surface color predominantly the surface color and unshaded. The two last point lists have the variable z, so their polygons are shaded across their edge spans. The first point list contains only one polygon and is a triangle. The second point list contains four polygons with the following shapes: a triangle, a parallelogram, the front view of a house, and a diamond with its top and bottom pointed extensions clipped.

▼ multipg2

Test Types:	RGB, SM

Description:	XGL_ILLUM_NONE_INTERP_COLOR mode for multipolygons produces shaded polygons based on interpolation through scanlines from the different vertex colors set through point type *color_f3d*. Seven point lists with one polygon each with the first four polygons a square, the next a vertical bow tie, the next a horizontal bow tie, and the last a triangle with its base facing the top of the window.
Attributes Tested:	XGL_3D_CTX_SURF_FRONT_ILLUMINATION XGL_ILLUM_NONE_INTERP_COLOR
Operators Tested:	xgl_object_set xgl_multi_simple_polygon
Output:	Shaded orange to white from the left to the right side of the square. Shaded mint green to white from the top to the bottom of the square. Shaded red to white with red across the left diagonal of the square. Solid blue square. Shaded red to white vertical bow tie with the red originating from the top portion of the bow tie. Solid horizontal blue bow tie. Shaded triangle from red to purple with the red originating from the base of the triangle facing the top of the window.

▼ multipg3

Test Types:	RGB, SM
Description:	XGL_3D_CTX_SURF_GEOM_NORMAL is used for multipolygons to produce the same polygon with either the back-face color or the front-face color dependent on whether the geometry normal is set to first point or last point respectively because the z value for these points, is front and last respectively.
Attributes Tested:	XGL_3D_CTX_SURF_BACK_COLOR XGL_3D_CTX_SURF_FACE_DISTINGUISH XGL_3D_CTX_SURF_GEOM_NORMAL XGL_CTX_SURF_FRONT_COLOR XGL_GEOM_NORMAL_FIRST_POINTS
Operators Tested:	xgl_object_set xgl_multi_simple_polygon xgl_object_get

Output: When the XGL_3D_CTX_SURF_GEOM_NORMAL is set to XGL_GEOM_NORMAL_LAST_POINTS, expects the front color set for the triangle, which is red. When the XGL_3D_CTX_SURF_GEOM_NORMAL is set to XGL_GEOM_NORMAL_FIRST_POINTS, expects the back color set for the triangle, which is blue.

▼ multipg4

Test Types: INDEX, SM

Description: Linear depth-cueing of simple solid polygons with every polygon except the first back facing. All are coplanar, although some of the vertex data do not lie in the same z plane through the polygon. The expected surface color at each vertex is a portion of the surface color based on the distance of its z value from the maximum z window boundary, which is set at 1.0. When the z values for the vertex of the polygon differ, the polygon will be shaded across the edges formed by the interpolation between the vertex. Since the color table installed is a graduation from black to white in equal increments, the closer the vertex is to the z window maximum, the closer the vertex color will be to black.

Attributes Tested: See Table 18-2, Column A at the end of this chapter.

Operators Tested:
```
xgl_object_set
xgl_multi_simple_polygon
xgl_object_get
```

Output: The first polygon list has a random z value for its vertexes and its shape is a shaded triangle with one vertex replaced by a vertical edge.

The second point list is front facing with a constant z value and is thus a solid colored square.

The third point list is back facing with a constant z value of the z window maximum and consequently blends into the background color.
The next point list has a constant z and two polygons, so they are a solid colored triangle and a solid colored rhombus.

The last two point lists have a variable z, so their polygons are shaded across their edge spans. The first point list contains only one polygon and is a triangle. The second point list contains three polygons with the following shapes: a triangle, the front view of a house, and a six-sided figure.

▼ multipg_cull

Test Types:	INDEX, SM
Description:	Renders two square polygons, the first is front facing and the second is back facing. Sets front color to red and back color to green. Loops through three modes of culling, off, front and back, and expect one red square and one green square, one green square and one red square respectively.
Attributes Tested:	See Table 18-1, Column A at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_multi_simple_polygon`
	`xgl_object_get`

Output:	Two square polygons, one red and one green. One green square polygon alone and then one red square polygon alone.

▼ multipg_cull_z

Test Types:	INDEX, SM
Description:	Renders two square polygons, the first front facing and the second back facing. Sets front color to red and back color to green. Loops through three modes of culling, off, front and back, and expects one red square and one green square, one green square and one red square respectively. Also sets `XGL_3D_CTX_HLHSR_MODE` to `XGL_HLHSR_Z_BUFFER`, but since none of the vertexes of the polygons lie outside the z plane of z = origin, this should not affect the same culling as portrayed by *multipg_cull* because the default `XGL_3D_CTX_HLHSR_DATA` is the maximum depth permitted by the device attached to the context.

Attributes Tested: XGL_3D_CTX_HLHSR_MODE
XGL_CTX_NEW_FRAME_ACTION
XGL_CTX_NEW_FRAME_CLEAR
XGL_CTX_NEW_FRAME_HLHSR_ACTION
XGL_HLHSR_Z_BUFFER
and Table 18-1, Column A at the end of this chapter

Operators Tested: xgl_object_set
xgl_multi_simple_polygon
xgl_object_get

Output: Two square polygons, one red and one green. One green square polygon alone and then one red square polygon alone.

▼ multipg_cull_rgb

Test Types: RGB, SM

Description: Renders two square polygons, the first front facing and the second back facing. Sets front color to red and back color to green. Loops through three modes of culling, off, front and back, and expects one red square and one green square, one green square and one red square respectively.

Attributes Tested: See Table 18-1, Column A at the end of this chapter.

Operators Tested: xgl_object_set
xgl_multi_simple_polygon
xgl_object_get

Output: Two square polygons, one red and one green. One green square polygon alone and then one red square polygon alone.

▼ multipg_cull_z_rgb

Test Types: RGB, SM

Description: Renders two square polygons, the first front facing and the second back facing. Sets front color to red and back color to green. Loops through three modes of culling, off, front and back, and expects one red and one green square, one green square and one red square respectively.

Attributes Tested: XGL_3D_CTX_HLHSR_MODE
XGL_CTX_NEW_FRAME_ACTION
XGL_CTX_NEW_FRAME_CLEAR

	XGL_CTX_NEW_FRAME_HLHSR_ACTION
	XGL_HLHSR_Z_BUFFER
	and Table 18-1, Column A at the end of this chapter
Operators Tested:	xgl_object_set
	xgl_multi_simple_polygon
	xgl_object_get
Output:	Two square polygons, one red and one green. One green square polygon alone and then one red square polygon alone.

multipg_edge

Test Types:	RGB, SM
Description:	Loops through interior colors of a cube, setting the interior of a 2D polygon to the cube color and setting its edges to an alternate line pattern with the six dashes set to red while the alternate dash is set to white
Attributes Tested:	See Table 18-1, Column B at the end of this chapter.
Operators Tested:	xgl_object_set
	xgl_multi_simple_polygon
	xgl_object_get
	xgl_object_create
	xgl_context_get_pixel
Output:	Simple polygon with patterned edges of six dashes with two dashes of separating white color and all interior colors.

multipg_edge2

Test Types:	RGB, SM
Description:	Loops through all possible flag combinations for the edges of a four-sided square for point type *flag_i2d* with interior color green and edge color purple
Attributes Tested:	XGL_CTX_BACKGROUND_COLOR
	XGL_CTX_EDGE_COLOR
	XGL_CTX_SURF_EDGE_FLAG
	XGL_CTX_SURF_FRONT_COLOR
Operators Tested:	xgl_object_set
	xgl_multi_simple_polygon
	xgl_object_get

Output:	One huge green square with all possible combinations for purple edges on and off

▼ multipg_edge3

Test Types:	RGB, SM
Description:	Loops through interior colors of a cube, setting the interior of a 3D polygon to the cube color and setting its edges to an alternate line pattern with the six dashes set to red while the alternate dash is set to white
Attributes Tested:	See Table 18-1, Column B at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_multi_simple_polygon`
	`xgl_object_get`
	`xgl_object_create`
	`xgl_context_get_pixel`

Output:	Simple polygon with patterned edges of six dashes with two dashes of separating white color and all interior colors

▼ multipg_edge4

Test Types:	RGB, SM
Description:	Renders all possible flag combinations for the edges of a four-sided square for point type *flag_i2d* with interior color green and edge color purple

Attributes Tested:	`XGL_CTX_EDGE_COLOR`
	`XGL_CTX_SURF_EDGE_FLAG`
	`XGL_CTX_SURF_FRONT_COLOR`

Operators Tested:	`xgl_object_set` `xgl_multi_simple_polygon`
Output:	Sixteen green squares, all with different combinations for the rendering of up to four purple edges

▼ multipg_face

Test Types:	INDEX, SM

Description:	Renders three polygons, the first front facing and the other two back facing. Sets face distinguishing to true and normal flip to false and expects the polygons rendered normally. Sets both face distinguishing and normal flip to false and expects the polygons all to be front facing. Sets both face distinguishing and normal flip to true and expects the normally facing polygons to be reversed. The front fill style is solid while the back fill style is hollow.
Attributes Tested:	See Table 18-2, Column B at the end of this chapter
Operators Tested:	`xgl_object_set` `xgl_multi_simple_polygon`
Output:	First frame: red solid square, green-edged hollow square, and green-edged hollow parallelogram. Next frame: red solid square, red solid square, and red solid parallelogram. Final frame: hollow green square, red solid square, and red solid parallelogram.

▼ multipg_face_z

Test Types:	INDEX, SM
Description:	Renders three polygons, the first front facing and the other two back facing. Sets face distinguishing to true and normal flip to false and expects the polygons rendered normally. Sets both face distinguishing and normal flip to false and expects the polygons all to be front facing. Sets both face distinguishing and normal flip to true and expects the normally facing polygons to be reversed. HLHSR is also set with the default XGL_3D_CTX_HLHSR_DATA value, which is the maximum depth permitted by the device attached to the context. The front fill style is solid while the back fill style is hollow.
Attributes Tested:	See Table 18-1, Column C at the end of this chapter.
Operators Tested:	`xgl_object_set` `xgl_multi_simple_polygon`
Output:	First frame: red solid square, green-edged hollow square, and green-edged hollow parallelogram. Next frame: red solid square, red solid square, and red solid parallelogram. Final frame: hollow green square, red solid square, and red solid parallelogram.

multipg_face_rgb

Test Types:	RGB, SM
Description:	RGB version of *multipg_face*
Attributes Tested:	See Table 18-2, Column B at the end of this chapter.
Operators Tested:	`xgl_object_set` `xgl_multi_simple_polygon`
Output:	First frame: red solid square, green-edged hollow square, and green-edged hollow parallelogram. Next frame: red solid square, red solid square, and red solid parallelogram. Final frame: hollow green square, red solid square, and red solid parallelogram.

multipg_face_z_rgb

Test Types:	RGB, SM
Description:	RGB version of *multipg_face_z*
Attributes Tested:	See Table 18-1, Column C at the end of this chapter.
Operators Tested:	`xgl_object_set` `xgl_multi_simple_polygon`
Output:	First frame: red solid square, green-edged hollow square, and green-edged hollow parallelogram. Next frame: red solid square, red solid square, and red solid parallelogram. Final frame: hollow green square, red solid square, and red solid parallelogram.

multipg_fill

Test Types:	INDEX, SM
Description:	Tests various polygon fill styles for multisimple polygons
Attributes Tested:	See Table 18-2, Column C at the end of this chapter.
Operators Tested:	`xgl_object_set` `xgl_multi_simple_polygon`
Output:	Six polygon point lists displayed as hollow normal with edges, empty wide edges, and solid. The shapes inside the polygon point lists are (1) a square, (2) a square, (3) nine triangles with three to a row (4) a polygon in the shape of a letter “d” with two holes in it, the expected hole and a hole in the vertical base of the “d” (5) a square within a square, and (6) a square overlapping another square.

multipg_fill_z

Test Types: INDEX, SM

Description: Tests various HLHSR polygon fill styles for multisimple polygons. Same as *multipg_fill* but with HLHSR on and set to the default XGL_3D_CTX_HLHSR_DATA, which is the maximum depth permitted by the device attached to the context.

Attributes Tested:
XGL_3D_CTX_HLHSR_MODE
XGL_CTX_NEW_FRAME_ACTION
XGL_CTX_NEW_FRAME_CLEAR
XGL_CTX_NEW_FRAME_HLHSR_ACTION
XGL_HLHSR_Z_BUFFER
and Table 18-2, Column C at the end of this chapter

Operators Tested:
xgl_object_set
xgl_multi_simple_polygon

Output: Six polygon point lists displayed as hollow normal with edges, empty wide edges, and solid. The shapes inside the polygon point lists are (1) a square, (2) a square, (3) nine triangles with three to a row, (4) a polygon in the shape of a letter “d” with two holes in it, the expected hole and a hole in the vertical base of the “d”, (5) a square within a square, and (6) a square overlapping another square.

multipg_fill2

Test Types: INDEX, SM

Description: Tests pattern-filled multipolygons with / without edges. Patterns used include dots, crosshatch, and alternating on / off pixels (DSCREEN).

Attributes Tested: See Table 18-3, Column A at the end of this chapter.

Operators Tested:
xgl_object_set
xgl_multi_simple_polygon
xgl_object_get

Output: Two squares overlapping with the smaller square overlapping the lower right corner of the larger square filled with one of the three patterns mentioned in the *Description* section

multipg_fill_rgb

Test Types:	RGB, SM
Description:	Tests various polygon fill styles for RGB multisimple polygons
Attributes Tested:	XGL_CTX_SURF_FRONT_COLOR and Table 18-2, Column C at the end of this chapter
Operators Tested:	xgl_object_set xgl_multi_simple_polygon
Output:	Six polygon point lists displayed as hollow normal with edges, empty wide edges, and solid. The shapes inside the polygon point lists are (1) a square, (2) a square, (3) nine triangles with three to a row, (4) a polygon in the shape of a letter “d” with two holes in it, the expected hole, and a hole in the vertical base of the “d”, (5) a square within a square, and (6) a square overlapping another square.

multipg_fill_z_rgb

Test Types:	RGB, SM
Description:	Tests various HLHSR polygon fill styles for multisimple polygons. Same as *multipg_fill* but with HLHSR on and set to the default XGL_3D_CTX_HLHSR_DATA, which is the maximum depth permitted by the device attached to the context.
Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_CTX_NEW_FRAME_ACTION XGL_CTX_NEW_FRAME_CLEAR XGL_CTX_NEW_FRAME_HLHSR_ACTION XGL_CTX_SURF_FRONT_COLOR XGL_HLHSR_Z_BUFFER and Table 18-2, Column C at the end of this chapter
Operators Tested:	xgl_object_set xgl_multi_simple_polygon
Output:	Six polygon point lists displayed as hollow normal with edges, empty wide edges, and solid. The shapes inside the polygon point lists are (1) a square, (2) a square, (3) nine triangles with three to a row, (4) a polygon in the shape of

a letter “d” with two holes in it, the expected hole, and a hole in the vertical base of the “d”, (5) a square within a square, and (6) a square overlapping another square.

▼ multipg_fill4

Test Types: RGB, SM

Description: Tests pattern-filled RGB multipolygons with/without edges. Patterns used include dots, crosshatch, and alternating on/off pixels (`DSCREEN`).

Attributes Tested: See Table 18-3, Column A at the end of this chapter.

Operators Tested:
`xgl_object_set`
`xgl_multi_simple_polygon`
`xgl_object_get`

Output: Two squares overlapping with the smaller square overlapping the lower right corner of the larger square filled with one of the three patterns mentioned in the *Description* section

▼ multipg_fill5

Test Types: RGB, SM

Description: Tests pattern-filled RGB simple polygons with opaque filling style. Patterns used include dots, crosshatch, and alternating on/off pixels (`DSCREEN`). First renders the polygon with a solid fill, then sets the fill style to opaque stipple, and re-renders it with one of the patterns mentioned above.

Attributes Tested: See Table 18-3, Column B at the end of this chapter.

Operators Tested:
`xgl_object_set`
`xgl_multi_simple_polygon`

Output: Large filled square in the upper left corner alternating with a much smaller filled square in the lower right corner of the window

▼ multipg_fill6

Test Types: RGB, SM

Description:	Tests various polygon fill styles for RGB 2D multisimple polygons. Renders polygons solid, then with hollow fill style, no edges, and finally empty with wide edges.
Attributes Tested:	XGL_CTX_SURF_FRONT_COLOR and Table 18-2, Column C at the end of this chapter
Operators Tested:	xgl_object_set xgl_multi_simple_polygon
Output:	Six polygon point lists displayed as hollow normal with edges, empty wide edges, and solid. The shapes inside the polygon point lists are (1) a square, (2) a square, (3) nine triangles with three to a row, (4) a polygon in the shape of a letter "d" with two holes in it, the expected hole, and a hole in the vertical base of the "d", (5) a square within a square, and (6) a square overlapping another square.

▼ multipg_fill7

Test Types:	RGB, SM
Description:	Tests 3D pattern-filled RGB multipolygons with / without edges. Patterns used include dots, crosshatch, and alternating on / off pixels (DSCREEN).
Attributes Tested:	See Table 18-3, Column A at the end of this chapter.
Operators Tested:	xgl_object_set xgl_multi_simple_polygon
Output:	Two squares overlapping with the smaller square overlapping the lower right hand of the larger square filled with one of the three patterns mentioned in the *Description* section

▼ multipg_fill8

Test Types:	RGB, SM
Description:	Tests 3D back pattern-filled RGB simple polygons with opaque filling style. Patterns used include dots, crosshatch, and alternating on / off pixels (DSCREEN). First renders the polygon with a solid fill, then sets the fill style to opaque stipple, and re-renders it with one of the patterns mentioned above.
Attributes Tested:	See Table 18-3, Column B at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_multi_simple_polygon`
	`xgl_object_get`

Output:	Large filled square in the upper left corner alternating with a much smaller filled square in the lower right corner of the window

▼ multipg_back_fill_rgb

Test Types:	RGB, SM
Description:	Tests various polygon back fill styles for RGB multisimple polygons. Same polygon list as *multipg_fill6* but with the normals so that the polygons are back facing, that is, z component is greater than 0.0. First renders the polygons as solids, then as hollow with normal sized edge width and then as empty with wide edge width.

Attributes Tested:	`XGL_3D_CTX_SURF_FACE_DISTINGUISH`
	`XGL_3D_CTX_SURF_BACK_COLOR`
	`XGL_CTX_SURF_FRONT_FILL_STYLE`
	`XGL_SURF_FILL_EMPTY`
	`XGL_3D_CTX_SURF_BACK_FILL_STYLE`
	`XGL_SURF_FILL_SOLID`
	`XGL_SURF_FILL_HOLLOW`

Operators Tested:	`xgl_object_set` `xgl_multi_simple_polygon`
Output:	Six polygon point lists displayed as hollow normal with edges, empty wide edges, and solid. The shapes inside the polygon point lists are (1) a square, (2) a square, (3) nine triangles with three to a row, (4) a polygon in the shape of a letter “d” with two holes in it, the expected hole, and a hole in the vertical base of the “d”, (5) a square within a square, and (6) a square overlapping another square.

▼ multipg_back_fill_z_rgb

Test Types:	RGB, SM
Description:	Tests various polygon back fill styles for RGB multisimple polygons. Same polygon list as *multipg_fill6* but with the normals so that the polygons are back facing, that is, z component is greater than 0.0. First renders the polygons

	as solids, then as hollow with normal sized edge width and then as empty with wide edge width. Turns on XGL_HLHSR_Z_BUFFER assuming the default XGL_3D_CTX_HLHSR_DATA which is the maximum depth permitted by the device attached to the context.
Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_3D_CTX_SURF_BACK_COLOR XGL_3D_CTX_SURF_BACK_FILL_STYLE XGL_3D_CTX_SURF_FACE_DISTINGUISH XGL_CTX_NEW_FRAME_ACTION XGL_CTX_NEW_FRAME_CLEAR XGL_CTX_NEW_FRAME_HLHSR_ACTION XGL_HLHSR_Z_BUFFER and Table 18-2, Column C at the end of this chapter
Operators Tested:	xgl_object_set xgl_multi_simple_polygon
Output:	Six polygon point lists displayed as hollow normal with edges, empty wide edges, and solid. The shapes inside the polygon point lists are (1) a square, (2) a square, (3) nine triangles with three to a row, (4) a polygon in the shape of a letter “d” with two holes in it, the expected hole, and a hole in the vertical base of the “d”, (5) a square within a square, and (6) a square overlapping another square.

▼ multipg_fill10

Test Types:	RGB, SM
Description:	Tests 3D back pattern-filled RGB multipolygons with/without edges. Patterns used include dots, crosshatch, and alternating on/off pixels (DSCREEN).
Attributes Tested:	XGL_SURF_FILL_STIPPLE XGL_CTX_EDGE_COLOR XGL_CTX_SURF_EDGE_FLAG and Table 18-3, Column C at the end of this chapter
Operators Tested:	xgl_object_set xgl_multi_simple_polygon xgl_object_get

Output:	Two squares overlapping with the smaller square overlapping the lower right corner of the larger square filled with one of the three patterns mentioned in the *Description* section

▼ multipg_fill11

Test Types:	RGB, SM
Description:	Tests 3D back pattern-filled RGB simple polygons with opaque filling style. Patterns used include dots, crosshatch, and alternating on / off pixels (`DSCREEN`). First renders the polygon with a solid fill, then sets the fill style to opaque stipple, and re-renders it with one of the patterns mentioned above.

Attributes Tested:	`XGL_SURF_FILL_SOLID`
	`XGL_SURF_FILL_OPAQUE_STIPPLE`
	and Table 18-3, Column C at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_multi_simple_polygon`
	`xgl_object_get`

Output:	Large filled square in the upper left corner alternating with a much smaller filled square in the lower right corner of the window

▼ multipg_hlhsr

Test Types:	INDEX, SM
Description:	Renders two polygons in a point list in which each polygon is a twin of the other except for its z values. Uses facet color normal and sets the facet colors for each twin to two different colors. Expects the polygon to appear in the facet color for the polygon in front, that is, z value smallest for each member of vertex for the polygon. The polygons that are rendered in this manner are a square, vertical bow tie, horizontal bow tie, and one triangle.
Attributes Tested:	`XGL_DEV_COLOR_MAP` and Table 18-4, Column A at the end of this chapter
Operators Tested:	`xgl_object_set` `xgl_multi_simple_polygon`

Output: Polygons in either red or green, dependent on whether the first polygon is in front of its twin or the second polygon is in front of its twin. The shapes of the polygons are those mentioned in the *Description* section.

▼ multipg_hlhsr2

Test Types: RGB, SM

Description: Same as *multipg_hlhsr* except RGB. Renders two polygons in a point list in which each polygon is a twin of the other except for its z values. Uses facet color normal and sets the facet colors for each twin to two different colors. Expects the polygon to appear in the facet color for the polygon in front, that is, z value smallest for each member of vertex for the polygon. The polygons that are rendered in this manner are a square, vertical bow tie, horizontal bow tie, and one triangle.

Attributes Tested: See Table 18-4, Column A at the end of this chapter.

Operators Tested: `xgl_object_set`
`xgl_multi_simple_polygon`

Output: Polygons in either red or green, dependent on whether the first polygon is in front of its twin or the second polygon is in front of its twin. The shapes of the polygons are those mentioned in the *Description* section.

▼ multipg_hlhsr4

Test Types: RGB, SM

Description: Renders a solid planar square with hidden surface removal varying the z plane for the square and the hidden surface data for removal

Attributes Tested: `XGL_3D_CTX_HLHSR_DATA`
`XGL_CTX_BACKGROUND_COLOR`
`XGL_CTX_SURF_FRONT_COLOR`
and Table 18-4, Column A at the end of this chapter

Operators Tested: `xgl_object_set`
`xgl_multi_simple_polygon`

Output:	Expects a solid orange square for the first four frames because the z value for the primitive is greater than the hidden surface data set. Expects only background color for the final two frames because the z value for the primitive is behind the hidden surface data.

multipg_intrule

Test Types:	INDEX, SM
Description:	Tests that `xgl_multi_simple_polygon()` correctly applies `XGL_CTX_SURF_INTERIOR_RULE` when drawing both 3D and 2D polygons. Renders a polygon in the shape of a letter "d" with two holes in it, the expected hole for the letter, and an additional donut at the upper portion of the vertical base. Renders two squares. Renders one square plus the shape of a crown rotated -90 degrees and overlapping the square so that only the two triangle vertexes appear beyond the square's boundary. The same letter "d" appears only smaller and translated to a different location on the window. Now renders the same polygons in 2D rather than 3D.
Attributes Tested:	`XGL_CTX_SURF_INTERIOR_RULE`
Operators Tested:	`xgl_object_set` `xgl_multi_simple_polygon`
Output:	Shapes as mentioned in the *Description* section in the default index color map color of white

multipg_intrule_rgb

Test Types:	RGB, SM
Description:	Tests that `xgl_multi_simple_polygon()` correctly applies `XGL_CTX_SURF_INTERIOR_RULE` when drawing RGB 3D and 2D polygons. Same as *multipg_intrule* except RGB.
Attributes Tested:	`XGL_CTX_SURF_INTERIOR_RULE`
Operators Tested:	`xgl_object_set` `xgl_multi_simple_polygon`
Output:	The same shapes in the same order as mentioned in the *Description* section from *multipg_intrule* except RGB has no default color, so the front surface color is set to orange.

multipg_pttypes

Test Types:	INDEX, SM
Description:	Renders the same three triangles in a column exercising six different point types and three different facets: XGL_PT_I2D, XGL_PT_I2H, XGL_PT_F2D, XGL_PT_F2H, XGL_PT_F3D and XGL_PT_F3H. Finally tries rendering three polygons (circles), each of which has 400 vertexes using F3D point type and three different facets.
Attributes Tested:	Default attributes for xgl_multi_simple_polygon
Operators Tested:	xgl_multi_simple_polygon
Output:	Six frames of three triangles in a color with their colors from top to bottom: green, yellow, and blue. Three circles in a row with their colors from left to right: green, yellow, and blue.

multipg_pttypes2

Test Types:	RGB, SM
Description:	Exactly the same as *multipg_pttypes* except using RGB
Attributes Tested:	Default attributes for xgl_multi_simple_polygon
Operators Tested:	xgl_multi_simple_polygon
Output:	Exactly the same output as *multipg_pttypes*

gcache_multipg_cull

Test Types:	INDEX, SM
Description:	Renders two square polygons, the first front facing and the second back facing, all using gcache. Sets front color to red and back color to green. Loops through three modes of culling, off, front and back, and expects one red square and one green square, one green square and one red square respectively.
Attributes Tested:	XGL_CTX_NEW_FRAME_ACTION XGL_CTX_NEW_FRAME_CLEAR XGL_CTX_NEW_FRAME_HLHSR_ACTION XGL_GCACHE and Table 18-1, Column A at the end of this chapter

Operators Tested:	`xgl_object_create`
	`xgl_object_set`
	`xgl_gcache_multi_simple_polygon`
	`xgl_context_display_gcache`

Output:	Two square polygons, one red and one green. One green square polygon alone and then one red square polygon alone.

▼ gcache_multipg_edge4

Test Types:	RGB, SM
Description:	Renders all possible flag combinations for the edges of a four-sided square for point type *flag_f3d* with interior color green and edge color purple

Attributes Tested:	`XGL_CACHE_ATTR_STATE_DIFFERENT`
	`XGL_CTX_EDGE_COLOR`
	`XGL_CTX_SURF_EDGE_FLAG`
	`XGL_CTX_SURF_FRONT_COLOR`
	`XGL_GCACHE`
	`XGL_GCACHE_IS_EMPTY`
	`XGL_GCACHE_POLYGON_TYPE`
	`XGL_GCACHE_USE_APPL_GEOM`
	`XGL_POLYGON_NSI`

Operators Tested:	`xgl_object_create`
	`xgl_object_set`
	`xgl_gcache_multi_simple_polygon`
	`xgl_context_display_gcache`

Output:	Sixteen green squares, all with different combinations for the rendering of up to four purple edges

▼ gcache_multipg_face

Test Types:	INDEX, SM
Description:	Renders three polygons, the first front facing and the other two back facing. Sets face distinguishing to true and normal flip to false and expects the polygons rendered normally. Sets both face distinguishing and normal flip to false and expects the polygons all to be front facing. Sets

	both face distinguishing and normal flip to true and expects the normally facing polygons to be reversed. The front fill style is solid while the back fill style is hollow.
Attributes Tested:	`XGL_GCACHE` and Table 18-1, Column C at the end of this chapter

Operators Tested:	`xgl_object_create`
	`xgl_object_set,`
	`xgl_gcache_multi_simple_polygon`
	`xgl_context_display_gcache`

Output:	First frame: red solid square, green edged hollow square, and green edged hollow parallelogram. Next frame: red solid square, red solid square, and red solid parallelogram. Final frame: hollow green square, red solid square, and red solid parallelogram.

▼ gcache_multipg_face2

Test Types:	RGB, SM
Description:	Renders three polygons, the first front facing and the other two back facing. Sets face distinguishing to true and normal flip to false and expects the polygons rendered normally. Sets both face distinguishing and normal flip to false and expects the polygons all to be front facing. Sets both face distinguishing and normal flip to true and expects the normally facing polygons to be reversed. The front fill style is solid while the back fill style is hollow.
Attributes Tested:	`XGL_GCACHE` and Table 18-1, Column C at the end of this chapter

Operators Tested:	`xgl_object_create`
	`xgl_object_set`
	`xgl_gcache_multi_simple_polygon`
	`xgl_context_display_gcache`

Output:	First frame: red solid square, green edged hollow square, and green edged hollow parallelogram. Next frame: red solid square, red solid square, and red solid parallelogram. Final frame: hollow green square, red solid square, and red solid parallelogram.

gcache_multipg_fill1

Test Types:	RGB, SM
Description:	Tests various polygon fill styles for gcache multisimple polygons

Attributes Tested:	`XGL_3D_CTX_HLHSR_MODE`
	`XGL_CTX_NEW_FRAME_ACTION`
	`XGL_CTX_NEW_FRAME_CLEAR`
	`XGL_CTX_NEW_FRAME_HLHSR_ACTION`
	`XGL_GCACHE`
	`XGL_GCACHE_POLYGON_TYPE`
	`XGL_POLYGON_COMPLEX`
	and Table 18-2, Column C at the end of this chapter

Operators Tested:	`xgl_object_create`
	`xgl_object_set`
	`xgl_gcache_multi_simple_polygon`
	`xgl_context_display_gcache`

Output:	Six polygon point lists displayed as hollow normal with edges, empty wide edges and solid. The shapes inside the polygon point lists are (1) a square, (2) a square, (3) nine triangles with three to a row (4) a polygon in the shape of a letter “d” with two holes in it, the expected hole, and a hole in the vertical base of the “d”, (5) a square within a square, and (6) a square overlapping another square.

gcache_multipg_fill11

Test Types:	RGB, SM
Description:	Tests 3D back pattern-filled RGB gcache simple polygons with opaque filling style. Patterns used include dots, crosshatch, and alternating on / off pixels (`DSCREEN`). First renders the polygon with a solid fill, then sets the fill style to opaque stipple, and re-renders it with one of the patterns mentioned above.

Attributes Tested:	`XGL_GCACHE`
	`XGL_SURF_FILL_OPAQUE_STIPPLE`
	`XGL_SURF_FILL_SOLID`
	and Table 18-3, Column C at the end of this chapter

Operators Tested:	xgl_object_create xgl_object_set xgl_object_get xgl_gcache_multi_simple_polygon xgl_context_display_gcache
Output:	Large filled square in the upper left corner alternating with a much smaller filled square in the lower right corner of the window

▼ gcache_multipg_fill3

Test Types:	RGB, SM
Description:	Tests various polygon fill styles for RGB gcache multisimple polygons
Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_CTX_NEW_FRAME_ACTION XGL_CTX_NEW_FRAME_CLEAR XGL_CTX_NEW_FRAME_HLHSR_ACTION XGL_CTX_SURF_FRONT_COLOR XGL_GCACHE XGL_GCACHE_POLYGON_TYPE XGL_POLYGON_COMPLEX and Table 18-2, Column C at the end of this chapter
Operators Tested:	xgl_object_create xgl_object_set xgl_gcache_multi_simple_polygon xgl_context_display_gcache
Output:	Six polygon point lists displayed as hollow normal with edges, empty wide edges and solid. The shapes inside the polygon point lists are (1) a square, (2) a square, (3) nine triangles with three to a row, (4) a polygon in the shape of a letter “d” with two holes in it, the expected hole, and a hole in the vertical base of the “d”, (5) a square within a square, and (6) a square overlapping another square.

▼ gcache_multipg_fill9

Test Types:	RGB, SM

Description:	Tests various polygon back fill styles for RGB gcache multisimple polygons. Same polygon list as *multipg_fill6* but with the normals so that the polygons are back facing, that is, z component is greater than 0.0. First renders the polygons as solids, then as hollow with normal sized edge width, and finally as empty with wide edge width.
Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_3D_CTX_SURF_BACK_COLOR XGL_3D_CTX_SURF_BACK_FILL_STYLE XGL_3D_CTX_SURF_FACE_DISTINGUISH XGL_CTX_NEW_FRAME_ACTION XGL_CTX_NEW_FRAME_CLEAR XGL_CTX_NEW_FRAME_HLHSR_ACTION XGL_GCACHE and Table 18-2, Column C at the end of this chapter
Operators Tested:	xgl_object_create xgl_object_set xgl_gcache_multi_simple_polygon xgl_context_display_gcache
Output:	Six polygon point lists displayed as hollow normal with edges, empty wide edges, and solid. The shapes inside the polygon point lists are (1) a square, (2) a square, (3) nine triangles with three to a row, (4) a polygon in the shape of a letter “d” with two holes in it, the expected hole, and a hole in the vertical base of the “d”, (5) a square within a square, and (6) a square overlapping another square.

▼ ms_poly_sedge

Test Types:	CM, INDEX
Description:	Tests complex (3D) polygons and silhouette edges. The geometry defines a polyhedron with front- and back facing three-sided facets, then turns on XGL_3D_CTX_SURF_SILHOUETTE_EDGE_FLAG. See bug list for more information on failures.
Attributes Tested:	XGL_3D_CTX_SURF_BACK_COLOR XGL_3D_CTX_SURF_SILHOUETTE_EDGE_FLAG XGL_3D_CTX_SURF_SILHOUETTE_EDGE_FLAG XGL_CTX_EDGE_COLOR XGL_CTX_SURF_FRONT_COLOR

	XGL_CTX_VDC_MAP XGL_CTX_VDC_WINDOW XGL_RAS_HEIGHT XGL_RAS_WIDTH XGL_VDC_MAP_ASPECT and Table 18-4, Column A at the end of this chapter
Operators Tested:	xgl_object_set xgl_object_get xgl_multi_simple_polygon
Output:	Eight triangles make up the polyhedron. Triangles 1, 2, 3, and 4 fan out to the right from the pivot point, 200,200. Triangles 5, 6, 7, and 8 fan out to the left from the pivot point, 200,200. Triangles 6 and 7 overlap but 6 is back facing, so we see only triangle 7. We expect a silhouette edge at the borders of triangle 7 and triangle 8. Triangles 5 and 8 overlap but 5 is back facing and we see only triangle 8. Triangles 1 and 4 overlap but triangle 4 is back facing and we see only triangle 1. Triangles 2 and 3 are both front facing and overlap so we see only triangle 3. We expect a silhouette edge at the border of triangle 3. Silhouette edges are yellow and the front surface color is red.

▼ ms_pg_threshold

Test Types:	SM, INDEX
Description:	Tests convex and non-convex multisimple polygons with XGL_CTX_THRESHOLD and 2D context. It also sets the threshold and renders with null bbox (which should render all the shapes). This was adapted from *pg_threshold.c*.
Attributes Tested:	XGL_CTX_THRESHOLD
Operators Tested:	xgl_object_set xgl_multi_simple_polygon
Output:	Renders twelve polygons of different sizes and bboxes with threshold values set from 0 to 100 in increments of 20. Shapes are predominantly in the upper left portion of the window raster with one star shaped object further down the center and to the right. They range from stars to parallelograms.

ms_pg_facet_rgb

Test Types:	SM, RGB
Description:	RGB test for multisimple polygons with the different combinations of XGL_3D_CTX_SURF_FRONT_ILLUMINATION and XGL_CTX_SURF_FRONT_COLOR_SELECTOR. This test uses color facets with *Xgl_pt_color_normal_f3d* data. An ambient light was set in the *ctx* so you could see primitives when the illumination model was per vertex or per facet. This test uses RGB color model and sets the surface color to red, vertex colors to green, and facet colors to blue. Light is set to white. Two rows of polygons are rendered per combination (sixteen frames altogether); row 1 is tripointed, row 2 is quadpointed.
Attributes Tested:	See Table 18-4, Column B at the end of this chapter.
Operators Tested:	xgl_object_set xgl_multi_simple_polygon xgl_object_get
Output:	Regardless of the illumination—none, none_interp, per facet or per vertex—using the color selector of SURF_COLOR_CONTEXT yields the row of triangles and the row of squares in the surface color, which is red. Regardless of the illumination—none, none_interp, per facet or per vertex—using the color selector of SURF_COLOR_FACET yields the row of triangles and the row of squares in the facet color, which is blue. Regardless of the illumination—none, none_interp, per facet or per vertex—using the color selector of VERTEX_ILLUM_INDEP yields the row of triangles and the row of squares in the vertex color, which is green. Using the color selector of VERTEX_ILLUM_DEP yields the facet color, which is blue for illumination NONE and PER_FACET and the vertex color, which is green for illumination NONE_INTERP and PER_VERTEX.

ms_pg_facet_in

Test Types:	CM, INDEX

Description:	Tests the index for multisimple polygons with the different combinations of `XGL_3D_CTX_SURF_FRONT_ILLUMINATION` and `XGL_CTX_SURF_FRONT_COLOR_SELECTOR`. Uses color facets with *Xgl_pt_color_normal_f3d* data. An ambient light is set in the *ctx* so you can see primitives when the illumination model is per vertex or per facet. Uses color ramps and sets the surface color to red, vertex colors to green, and facet colors to blue. Light color is set to white (lightest gray).

Attributes Tested:	`XGL_CMAP_RAMP_LIST`
	`XGL_CMAP_RAMP_NUM`
	and Table 18-4, Column B at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_multi_simple_polygon`
	`xgl_object_get`

Output:	Regardless of the illumination—none, none_interp, per facet or per vertex—using the color selector of `SURF_COLOR_CONTEXT` yields the row of triangles and the row of squares in the surface color, which is red. Regardless of the illumination—none, none_interp, per facet or per vertex—using the color selector of `SURF_COLOR_FACET` yields the row of triangles and the row of squares in the facet color, which is blue. Regardless of the illumination—none, none_interp, per facet or per vertex—using the color selector of `VERTEX_ILLUM_INDEP` yields the row of triangles and the row of squares in the vertex color, which is green. Using the color selector of `VERTEX_ILLUM_DEP` yields the facet color, which is blue for illumination `NONE` and `PER_FACET` and the vertex color, which is green for illumination `NONE_INTERP` and `PER_VERTEX`.

▼ ms_pg_fac_in_norm

Test Types:	CM, RGB
Description:	Uses `color_normal` facets with *Xgl_pt_color_normal_f3d* data. An ambient light is set in the *ctx* so you could see primitives when the illumination model is per vertex or per facet. This is derived from *ms_pg_fac_in.c*, which uses

	facet_color facets instead of facet_normal_color. Uses color ramps and sets the surface color to red, vertex colors to green, and facet colors to blue. Light color is set to white (lightest gray). When the illumination mode is set to per_facet or per_vertex, colors are obtained from the bottom of the ramp of the color that is set.
Attributes Tested:	XGL_CMAP_RAMP_LIST XGL_CMAP_RAMP_NUM and Table 18-4, Column B at the end of this chapter
Operators Tested:	xgl_object_set xgl_multi_simple_polygon xgl_object_get
Output:	Regardless of the illumination—none, none_interp, per facet or per vertex—using the color selector of SURF_COLOR_CONTEXT yields the row of triangles and the row of squares in the surface color, which appears lime green. Regardless of the illumination—none, none_interp, per facet or per vertex — using the color selector of SURF_COLOR_FACET yields the row of triangles and the row of squares in the facet color, which appears olive green. Regardless of the illumination—none, none_interp, per facet or per vertex—using the color selector of VERTEX_ILLUM_INDEP yields the row of triangles and the row of squares in the vertex color, which appears orange. Using the color selector of VERTEX_ILLUM_DEP yields the facet color, which appears olive green for illumination NONE and PER_FACET, and the vertex color, which appears orange for illumination NONE_INTERP and PER_VERTEX. The background color is red.

▼ ms_pg_fac_rgb_norm

Test Types:	SM, RGB
Description:	RGB test for multisimple polygons with the different combinations of XGL_3D_CTX_SURF_FRONT_ILLUMINATION and XGL_CTX_SURF_FRONT_COLOR_SELECTOR. Uses color normal facets with *Xgl_pt_color_normal_f3d* data. An ambient light is set in the *ctx* so you could see primitives when the illumination model is per vertex or per facet.

This is derived from *ms_pg_facet_rgb.c*, which uses `facet_color` facets instead of `facet_normal_color`. Uses RGB color model and sets the surface color to red, vertex colors to green, facet colors to blue, and the light to white.

Attributes Tested: See Table 18-4, Column B at the end of this chapter.

Operators Tested:
`xgl_object_set`
`xgl_multi_simple_polygon`
`xgl_object_get`

Output: Regardless of the illumination—none, none_interp, per facet or per vertex—using the color selector of `SURF_COLOR_CONTEXT` yields the row of triangles and the row of squares in the surface color, which is red. Regardless of the illumination—none, none_interp, per facet or per vertex—using the color selector of `SURF_COLOR_FACET` yields the row of triangles and the row of squares in the facet color, which is blue. Regardless of the illumination—none, none_interp, per facet or per vertex—using the color selector of `VERTEX_ILLUM_INDEP` yields the row of triangles and the row of squares in the vertex color, which is green. Using the color selector of `VERTEX_ILLUM_DEP` yields the facet color, which is blue for illumination `NONE` and `PER_FACET`, and the vertex color, which is green for illumination `NONE_INTERP` and `PER_VERTEX`.

Table 18-1 Multisimple Polygon Attributes Tested - Set 1

Column A	**Column B**	**Column C**
XGL_3D_CTX_SURF_BACK_COLOR	XGL_CMAP_COLOR_CUBE_SIZE	XGL_3D_CTX_HLHSR_MODE
XGL_3D_CTX_SURF_FACE_CULL	XGL_CTX_EDGE_ALT_COLOR	XGL_3D_CTX_SURF_BACK_COLOR
XGL_3D_CTX_SURF_BACK_FILL_ STYLE	XGL_CTX_EDGE_COLOR	XGL_3D_CTX_SURF_BACK_FILL_ STYLE
XGL_3D_CTX_SURF_BACK_ ILLUMINATION	XGL_CTX_EDGE_PATTERN	XGL_3D_CTX_SURF_FACE_ DISTINGUISH
XGL_3D_CTX_SURF_FACE_ DISTINGUISH	XGL_CTX_EDGE_STYLE	XGL_3D_CTX_SURF_NORMAL_FLIP

Table 18-1 Multisimple Polygon Attributes Tested *(Continued)* - Set 1

Column A	Column B	Column C
XGL_3D_CTX_SURF_FRONT_ ILLUMINATION	XGL_CTX_SURF_EDGE_FLAG	XGL_CTX_NEW_FRAME_ACTION
XGL_CTX_BACKGROUND_COLOR	XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_NEW_FRAME_CLEAR
XGL_CTX_SURF_FRONT_COLOR	XGL_LINE_ALT_PATTERNED	XGL_CTX_NEW_FRAME_HLHSR_ ACTION
XGL_CULL_OFF	XGL_LPAT	XGL_CTX_SURF_FRONT_COLOR
XGL_CTX_SURF_FRONT_FILL_ STYLE	XGL_LPAT_DATA	XGL_CTX_SURF_FRONT_FILL_ STYLE
XGL_CULL_BACK	XGL_LPAT_DATA_SIZE	XGL_HLHSR_Z_BUFFER
XGL_CULL_FRONT	XGL_DEV_COLOR_MAP	XGL_SURF_FILL_HOLLOW
XGL_ILLUM_NONE	XGL_RAS_DEPTH	XGL_SURF_FILL_SOLID
XGL_SURF_FILL_SOLID		

Table 18-2 Multisimple Polygon Attributes Tested - Set 2

Column A	Column B	Column C
XGL_3D_CTX_DEPTH_CUE_MODE	XGL_3D_CTX_SURF_BACK_COLOR	XGL_CTX_EDGE_COLOR
XGL_3D_CTX_SURF_BACK_COLOR	XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_EDGE_WIDTH_SCALE_FACTOR
XGL_CTX_SURF_FRONT_COLOR	XGL_3D_CTX_SURF_BACK_FILL_STYLE	XGL_CTX_SURF_EDGE_FLAG
XGL_CTX_VDC_MAP	XGL_3D_CTX_SURF_FACE_DISTINGUISH	XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_CTX_VDC_WINDOW	XGL_3D_CTX_SURF_NORMAL_FLIP	XGL_SURF_FILL_EMPTY
XGL_DEPTH_CUE_LINEAR	XGL_CTX_SURF_FRONT_FILL_STYLE	XGL_SURF_FILL_HOLLOW
XGL_RAS_HEIGHT	XGL_SURF_FILL_HOLLOW	XGL_SURF_FILL_SOLID
XGL_RAS_WIDTH	XGL_SURF_FILL_SOLID	
XGL_VDC_MAP_ASPECT		

Table 18-3 Multi Simple Polygon Attributes Tested - Set 3

Column A	**Column B**	**Column C**
XGL_CTX_BACKGROUND_COLOR	XGL_CTX_BACKGROUND_COLOR	XGL_3D_CTX_SURF_BACK_COLOR
XGL_CTX_EDGE_COLOR	XGL_CTX_SURF_FPAT	XGL_3D_CTX_SURF_BACK_FILL_ STYLE
XGL_CTX_SURF_EDGE_FLAG	XGL_CTX_SURF_FPAT_POSITION	XGL_3D_CTX_SURF_BACK_FPAT
XGL_CTX_SURF_FPAT	XGL_CTX_SURF_FRONT_COLOR	XGL_3D_CTX_SURF_BACK_FPAT_ POSITION
XGL_CTX_SURF_FPAT_POSITION	XGL_SURF_FILL_SOLID	XGL_3D_CTX_SURF_FACE_ DISTINGUISH
XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_SURF_FRONT_FILL_ STYLE	XGL_CTX_BACKGROUND_COLOR
XGL_CTX_SURF_FRONT_FILL_ STYLE	XGL_SURF_FILL_OPAQUE_STIPPLE	
XGL_SURF_FILL_STIPPLE		

Table 18-4 Multi Simple Polygon Attributes Tested - Set 4

Column A	Column B
XGL_3D_CTX_HLHSR_MODE	XGL_3D_CTX_LIGHTS
XGL_3D_CTX_SURF_FRONT_ILLUMINATION	XGL_3D_CTX_LIGHT_NUM
XGL_CTX_NEW_FRAME_ACTION	XGL_3D_CTX_LIGHT_SWITCHES
XGL_CTX_NEW_FRAME_CLEAR	XGL_3D_CTX_SURF_FRONT_AMBIENT
XGL_ILLUM_NONE	XGL_3D_CTX_SURF_FRONT_ILLUMINATION
XGL_CTX_NEW_FRAME_HLHSR_ACTION	XGL_CTX_SURF_FRONT_COLOR
XGL_CTX_SURF_FRONT_FILL_STYLE	XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT
XGL_HLHSR_Z_BUFFER	XGL_CTX_SURF_FRONT_COLOR_SELECTOR
XGL_DEV_COLOR_MAP	XGL_ILLUM_NONE
XGL_SURF_FILL_SOLID	XGL_ILLUM_NONE_INTERP_COLOR
	XGL_ILLUM_PER_FACET
	XGL_ILLUM_PER_VERTEX
	XGL_LIGHT_AMBIENT
	XGL_LIGHT_COLOR
	XGL_LIGHT_ENABLE_COMP_AMBIENT
	XGL_LIGHT_TYPE
	XGL_SURF_COLOR_CONTEXT
	XGL_SURF_COLOR_FACET
	XGL_SURF_COLOR_VERTEX_ILLUM_DEP
	XGL_SURF_COLOR_VERTEX_ILLUM_INDEP

Nurbs Test Descriptions 19

This chapter describes the Nurbs test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ nubs_args

Test Types:	INDEX, SM
Description:	Varies the control points, knot vector, parameter range, and the order of curve. The first loop tests four examples of nonrational B-spline curves, while the final loop tests three examples of rational B-spline curves as the various parameters are varied.
Attributes Tested:	XGL_CURVE_CONST_PARAM_SUBDIV_BETWEEN_KNOTS and Table 19-1, Column A at the end of this chapter
Operators Tested:	xgl_context_new_frame xgl_nu_bspline_curve

Output: The nonrational B-splines look like (1) a v-shaped line, (2) a musical note on its side, (3) a portion of a circle, and (4) a tooth with an indented right edge. The rational B-spline curves look like (1) a sleigh with a portion of the top missing, (2) a tooth with an indented right edge plus two rounded curves connected at a single point, and (3) a portion of an ellipse.

▼ nubs_approx

Test Types: INDEX, SM

Description: Sets `XGL_CTX_CURVE_APPROX` to various approximation methods in computing the nurbs curves for a quadratic B-spline

Attributes Tested:
`XGL_CTX_MAX_TESSELLATION`
`XGL_CURVE_UNUSED_BETWEEN_KNOTS`
and Table 19-1, Column A at the end of this chapter

Operators Tested:
`xgl_context_new_frame`
`xgl_nu_bspline_curve`
`xgl_object_set`
`xgl_object_get`

Output: Same curve displayed with different approximations appears similar to the letter “e” rotated to point to the lower portion of the window raster

▼ nubs_attr

Test Types: INDEX, SM

Description: Tests that line attributes are also applied to nurbs curves using a quadratic B-spline.

Attributes Tested: See Table 19-1, Column B at the end of this chapter

Operators Tested:
`xgl_context_new_frame`
`xgl_nu_bspline_curve`
`xgl_object_set`

Output: Curve whose appearance looks like an ampersand on its side that varies dependent on the line pattern and width of the line

▼ nubs_pttypes

Test Types:	INDEX, SM

Description:	Renders a quadratic spline curve using different point types to ensure the B-spline curves work with different point types. Tries different point types in the following order:
	`XGL_PT_I2D`
	`XGL_PT_I2H`
	`XGL_PT_F2D`
	`XGL_PT_F2H`
	`XGL_PT_F3D`
	`XGL_PT_F3H`

Attributes Tested:	See Table 19-1, Column A at the end of this chapter.
Operators Tested:	`xgl_context_new_frame` `xgl_nu_bspline_curve`
Output:	Curves displayed alternate between a lemon on its side and an ampersand on its side

▼ nubs_hlhsr

Test Types:	INDEX, SM
Description:	Tests nurbs' hidden line-hidden surface removal. Draws the curve twice, once using a depth that is expected to be overdrawn by the more forward z-valued curve, and once using a depth that produces a more forward curve. Changes the colors for the two curves so that we can expect hidden line-hidden surface removal to display only one color. Checks that only this one color is displayed on the screen.
Attributes Tested:	`XGL_CURVE_UNUSED_BETWEEN_KNOTS` and Table 19-1, Column A at the end of this chapter

Operators Tested:	`xgl_context_new_frame`
	`xgl_nu_bspline_curve`
	`xgl_object_set`

Output:	Curves displayed alternate between three alternatives: (1) a lemon on its side, (2) an ampersand on its side, and (3) the letter "e" rotated to point to the lower left portion of the window raster

nubs0

Test Types: RGB, SM

Description: Varies the control points, knot vector, parameter range, and the order of curve in RGB. The first loop tests four examples of nonrational B-spline curves while the final loop tests three examples of rational B-spline curves, and the various parameters are varied. Same test as *nubs_args* but using an RGB raster.

Attributes Tested: `XGL_CURVE_UNUSED_BETWEEN_KNOTS`
and Table 19-1, Column A at the end of this chapter

Operators Tested: `xgl_context_new_frame`
`xgl_nu_bspline_curve`

Output: The nonrational B-splines look like (1) a v-shaped line, (2) a musical note on its side, (3) a portion of a circle, and (4) a tooth with an indented right edge plus a small written "s" on its side. The rational B-spline curves look like (1) a sleigh with a portion of the top missing, (2) a tooth with an indented right edge plus two rounded curves connected at a single point, and (3) a portion of an ellipse.

nubs1

Test Types: RGB, SM

Description: Sets `XGL_CTX_CURVE_APPROX` to various approximation methods in RGB version of *nubs_approx*

Attributes Tested: `XGL_CTX_MAX_TESSELLATION`
`XGL_CURVE_UNUSED_BETWEEN_KNOTS`
and Table 19-1, Column A at the end of this chapter

Operators Tested: `xgl_context_new_frame`
`xgl_nu_bspline_curve`
`xgl_object_get`
`xgl_object_set`

Output: Same curve displayed with different approximations appears similar to the letter "e" rotated to point to the lower portion of the window raster

nubs2

Test Types:	RGB, SM

Description:	Renders a quadratic spline curve using different point types to ensure B-spline curves work with different point types. RGB version of *nubs_pttypes*. Different point types are tried in the following order:
	`XGL_PT_I2D`
	`XGL_PT_I2H`
	`XGL_PT_F2D`
	`XGL_PT_F2H`
	`XGL_PT_F3D`
	`XGL_PT_F3H.`

Attributes Tested:	`XGL_CURVE_UNUSED_BETWEEN_KNOTS` and Table 19-1, Column A at the end of this chapter
Operators Tested:	`xgl_context_new_frame` `xgl_nu_bspline_curve`
Output:	Curves displayed alternate between a lemon on its side and an ampersand on its side

nubs3

Test Types:	RGB, SM
Description:	Tests RGB nurbs' hidden line-hidden surface removal. Draws the curve twice, once using a depth that is expected to be overdrawn by the more forward z-valued curve, and once using a depth that produces a more forward curve. Changes the colors for the two curves so that we can expect hidden line-hidden surface removal to display only one color. Checks that only this one color is displayed on the screen. Same as *nubs_hlhsr* but RGB instead of INDEX.
Attributes Tested:	`XGL_CURVE_UNUSED_BETWEEN_KNOTS` and Table 19-1, Column A at the end of this chapter

Operators Tested:	`xgl_context_new_frame`
	`xgl_nu_bspline_curve`
	`xgl_object_set`

Output:	Curves displayed alternate between three alternatives: (1) a lemon on its side, (2) an ampersand on its side, and (3) the letter "e" rotated to point to the lower left portion of the window raster

▼ nubs4

Test Types:	RGB, SM
Description:	Varies the RGB colors randomly and try settings for `XGL_CTX_MAX_TESSELATION` from 128 down to 5 in equal decrements of 1 on a quadratic B-spline curve

Attributes Tested:	`XGL_CTX_MAX_TESSELATION`
	`XGL_CURVE_METRIC_VDC`
	and Table 19-1, Column A at the end of this chapter

Operators Tested:	`xgl_context_new_frame`
	`xgl_nu_bspline_curve`
	`xgl_object_set`

Output:	The curve looks like a cross between a plump “v” seen from an angle and an ampersand on its side.

▼ nubs5

Test Types:	RGB, SM
Description:	Varies the RGB colors randomly and increases two of the knot values by increments of 0.01 to 0.25 for a quadratic B-spline curve
Attributes Tested:	`XGL_CURVE_METRIC_VDC` and Table 19-1, Column A at the end of this chapter

Operators Tested:	`xgl_context_new_frame`
	`xgl_nu_bspline_curve`
	`xgl_object_set`

Output:	Curve looks similar to an ampersand on its side

▼ gc_nubs_args

Test Types:	INDEX, SM
Description:	Varies the control points, knot vector, parameter range, and the order of curve. Then renders the curve into a gcache. The first loop tests four examples of nonrational B-spline curves while the final loop tests three examples of rational B-spline curves as the various parameters are varied. Same test as *nubs_args* but using a gcache.

Attributes Tested:	`XGL_GCACHE`
	`XGL_CURVE_UNUSED_BETWEEN_KNOTS`
	and Table 19-1, Column A at the end of this chapter

Operators Tested:	`xgl_object_create`
	`xgl_gcache_nu_bspline_curve`
	`xgl_context_new_frame`
	`xgl_context_display_gcache`

Output:	The nonrational B-splines look like (1) a v-shaped line, (2) musical note on its side, (3) a portion of a circle, and (4) a tooth with an indented right edge plus a small written “s” on its side. The rational B-spline curves look like (1) a sleigh with a portion of the top missing, (2) a tooth with an indented right edge plus two rounded curves connected at a single point, and (3) a portion of an ellipse.

▼ gc_nubs_pttypes

Test Types:	INDEX, SM

Description:	Tests that nurbs can be rendered into a gcache using different point types. Gcache version of *nubs_pttypes*. Tries different point types in the following order:
	`XGL_PT_I2D`
	`XGL_PT_I2H`
	`XGL_PT_F2D`
	`XGL_PT_F2H`
	`XGL_PT_F3D`
	`XGL_PT_F3H`

Attributes Tested:	`XGL_CURVE_UNUSED_BETWEEN_KNOTS`
	`XGL_GCACHE`
	and Table 19-1, Column A at the end of this chapter

Operators Tested:	`xgl_object_create`
	`xgl_gcache_nu_bspline_curve`
	`xgl_context_new_frame`
	`xgl_context_display_gcache`

Output:	Curves displayed alternate between a lemon on its side and an ampersand on its side

▼ gc_nubs0

Test Types:	RGB, SM
Description:	Varies the RGB control points, knot vector, parameter range, and the order of the curve. Then renders the curve into a gcache. The first loop tests four examples of

nonrational B-spline curves, while the final loop tests three examples of rational B-spline curves, and the various parameters are varied. Same test as *nubs_args* but using a gcache.

Attributes Tested: `XGL_GCACHE`
`XGL_CURVE_UNUSED_BETWEEN_KNOTS`
and Table 19-1, Column A at the end of this chapter

Operators Tested: `xgl_object_create`
`xgl_gcache_nu_bspline_curve`
`xgl_context_new_frame`
`xgl_context_display_gcache`

Output: The nonrational B-splines look like (1) a v-shaped line, (2) a musical note on its side, (3) a portion of a circle, and (4) a tooth with an indented right edge plus a small written “s” on its side. The rational B-spline curves look like (1) a sleigh with a portion of the top missing, (2) a tooth with an indented right edge plus two rounded curves connected at a single point, and (3) a portion of an ellipse.

▼ gc_nubs2

Test Types: RGB, SM

Description: Tests that RGB nurbs can be rendered into a gcache using different point types. Gcache RGB version of *nubs_pttypes*. Tries different point types in the following order:
`XGL_PT_I2D`
`XGL_PT_I2H`
`XGL_PT_F2D`
`XGL_PT_F2H`
`XGL_PT_F3D`
`XGL_PT_F3H`

Attributes Tested: `XGL_CURVE_UNUSED_BETWEEN_KNOTS`
`XGL_GCACHE`
and Table 19-1, Column A at the end of this chapter

Operators Tested: `xgl_object_create`
`xgl_gcache_nu_bspline_curve`
`xgl_context_new_frame`
`xgl_context_display_gcache`

Output: Curves displayed alternate between a lemon on its side and an ampersand on its side

nurbs0

Test Types: RGB, SM
Description: Tests that non-trimmed surface nurbs can be rendered with various surface parameters. Point type: *Xgl_pt_f3*d.
Attributes Tested:
XGL_3D_CTX_HLHSR_MODE
XGL_HLHSR_Z_BUFFER
XGL_CTX_NEW_FRAME_ACTION
XGL_CTX_NEW_FRAME_CLEAR
XGL_CTX_NEW_FRAME_HLHSR_ACTION
XGL_CTX_LINE_COLOR
XGL_CTX_EDGE_COLOR
XGL_CTX_SURF_FRONT_COLOR
XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_CTX_SURF_EDGE_FLAG
XGL_CTX_NURBS_SURF_APPROX
XGL_CTX_NURBS_SURF_APPROV_VAL_{UV}
XGL_CTX_NURBS_SURF_PARAM_STYLE
XGL_CTX_NURBS_SURF_ISO_CURVE_PLACEMENT
XGL_CTX_NURBS_SURF_ISO_CURVE_{UV}_NUM
Operators Tested:
xgl_object_create
xgl_nurbs_surface
xgl_context_new_frame
Output: A surface with and without edges and isolines.

nurbs1

Test Types: RGB, SM
Description: Tests that trimmed surface nurbs can be rendered with various surface parameters. Point type: *Xgl_pt_f3*h.
Attributes Tested:
XGL_3D_CTX_HLHSR_MODE
XGL_HLHSR_Z_BUFFER
XGL_CTX_NEW_FRAME_ACTION
XGL_CTX_NEW_FRAME_CLEAR
XGL_CTX_NEW_FRAME_HLHSR_ACTION
XGL_CTX_LINE_COLOR
XGL_CTX_EDGE_COLOR
XGL_CTX_SURF_FRONT_COLOR
XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_CTX_SURF_EDGE_FLAG

	XGL_CTX_NURBS_SURF_APPROX
	XGL_CTX_NURBS_SURF_APPROV_VAL_{UV}
	XGL_CTX_NURBS_SURF_PARAM_STYLE
	XGL_CTX_NURBS_SURF_ISO_CURVE_PLACEMENT
	XGL_CTX_NURBS_SURF_ISO_CURVE_{UV}_NUM
Operators Tested:	xgl_object_create
	xgl_nurbs_surface
	xgl_context_new_frame
Output:	A sphere with and without trimming, with and without edges and isolines.

▼ gc_nurbs0

Test Types:	RGB, SM
Description:	Tests that surface nurbs can be rendered into a gcache using different gcache modes. Gcache modes are:
	XGL_GCACHE_NURBS_DYNAMIC
	XGL_GCACHE_NURBS_STATIC
	XGL_GCACHE_NURBS_COMBINED
Attributes Tested:	XGL_3D_CTX_HLHSR_MODE
	XGL_CTX_NEW_FRAME_ACTION
	XGL_CTX_NEW_FRAME_CLEAR
	XGL_CTX_NEW_FRAME_HLHSR_ACTION
	XGL_CTX_NURBS_SURF_APPROX
	XGL_CTX_NURBS_SURF_APPROV_VAL_{UV}
	XGL_CTX_NURBS_SURF_PARAM_STYLE
	XGL_GCACHE
	XGL_GCACHE_NURBS_SURF_MODE
	XGL_HLHSR_Z_BUFFER
Operators Tested:	xgl_object_create
	xgl_gcache_nurbs_surface
	xgl_context_new_frame
	xgl_context_display_gcache
Output:	Sphere with and without trimming.

▼ nurb_silo

Test Types:	RGB, CM
Description:	Tests that surface nurbs can draw silhouettes.

Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_CTX_NEW_FRAME_ACTION XGL_CTX_NEW_FRAME_CLEAR XGL_CTX_NEW_FRAME_HLHSR_ACTION XGL_CTX_NURBS_SURF_APPROX XGL_CTX_NURBS_SURF_APPROV_VAL_{UV} XGL_CTX_NURBS_SURF_PARAM_STYLE XGL_3D_CTX_SURF_SILHOUETTE_EDGE_FLAG XGL_GCACHE,XGL_GCACHE_NURBS_SURF_MODE XGL_HLHSR_Z_BUFFER
Operators Tested:	xgl_object_create xgl_nurbs_surface xgl_context_new_frame
Output:	Draws a plane with a large bump in it.

▼ nurb_tiny

Test Types:	RGB, CM
Description:	Tests when nurbs uses a special case for simple nurbs
Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_CTX_NEW_FRAME_ACTION XGL_CTX_NEW_FRAME_CLEAR XGL_CTX_NEW_FRAME_HLHSR_ACTION XGL_CTX_NURBS_SURF_APPROX XGL_CTX_NURBS_SURF_APPROV_VAL_{UV} XGL_CTX_NURBS_SURF_PARAM_STYLE XGL_GCACHE XGL_GCACHE_NURBS_SURF_MODE XGL_HLHSR_Z_BUFFER
Operators Tested:	xgl_object_create xgl_gcache_nurbs_surface xgl_context_new_frame xgl_context_display_gcache
Output:	Flat plane on bottom and on top a plane with a bump in it

▼ nurb_sub

Test Types:	RGB, CM
Description:	Tests that surface nurbs can be subdivided

Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_CTX_NEW_FRAME_ACTION XGL_CTX_NEW_FRAME_CLEAR XGL_CTX_NEW_FRAME_HLHSR_ACTION XGL_CTX_NURBS_SURF_APPROX XGL_CTX_NURBS_SURF_APPROV_VAL_{UV} XGL_CTX_NURBS_SURF_PARAM_STYLE XGL_GCACHE XGL_GCACHE_NURBS_SURF_MODE XGL_HLHSR_Z_BUFFER
Operators Tested:	xgl_object_create xgl_gcache_nurbs_surface xgl_context_new_frame xgl_context_display_gcache
Output:	Plane with bump and tessellation outlines on

▼ nurb_high

Test Types:	RGB, CM
Description:	Tests rendering of high order surfaces
Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_CTX_NEW_FRAME_ACTION XGL_CTX_NEW_FRAME_CLEAR XGL_CTX_NEW_FRAME_HLHSR_ACTION XGL_CTX_NURBS_SURF_APPROX XGL_CTX_NURBS_SURF_APPROV_VAL_{UV} XGL_CTX_NURBS_SURF_PARAM_STYLE XGL_GCACHE XGL_GCACHE_NURBS_SURF_MODE XGL_HLHSR_Z_BUFFER
Operators Tested:	xgl_object_create xgl_gcache_nurbs_surface xgl_context_new_frame xgl_context_display_gcache
Output:	Square and error message: Error number di-130: Order is not supported

nurb_trim1

Test Types:	RGB, CM
Description:	Tests trimming nurbs
Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_CTX_NEW_FRAME_ACTION XGL_CTX_NEW_FRAME_CLEAR XGL_CTX_NEW_FRAME_HLHSR_ACTION XGL_GCACHE XGL_GCACHE_NURBS_SURF_MODE XGL_HLHSR_Z_BUFFER
Operators Tested:	xgl_object_create xgl_gcache_nurbs_surface xgl_context_new_frame xgl_context_display_gcache
Output:	Square with curve on the bottom edge

nurb_trim2

Test Types:	RGB, CM
Description:	Tests trimming nurbs
Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_CTX_NEW_FRAME_ACTION XGL_CTX_NEW_FRAME_CLEAR XGL_CTX_NEW_FRAME_HLHSR_ACTION XGL_GCACHE XGL_GCACHE_NURBS_SURF_MODE XGL_HLHSR_Z_BUFFER
Operators Tested:	xgl_object_create xgl_gcache_nurbs_surface xgl_context_new_frame xgl_context_display_gcache
Output:	Square with curve on the left edge

nurb_trim3

Test Types:	RGB, CM
Description:	Tests trimming nurbs

Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_CTX_NEW_FRAME_ACTION XGL_CTX_NEW_FRAME_CLEAR XGL_CTX_NEW_FRAME_HLHSR_ACTION XGL_GCACHE XGL_GCACHE_NURBS_SURF_MODE XGL_HLHSR_Z_BUFFER
Operators Tested:	xgl_object_create xgl_gcache_nurbs_surface xgl_context_new_frame xgl_context_display_gcache
Output:	Curves on top and bottom edge

▼ nurb_trim4

Test Types:	RGB, CM
Description:	Tests trimming nurbs
Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_CTX_NEW_FRAME_ACTION XGL_CTX_NEW_FRAME_CLEAR XGL_CTX_NEW_FRAME_HLHSR_ACTION XGL_GCACHE XGL_GCACHE_NURBS_SURF_MODE XGL_HLHSR_Z_BUFFER
Operators Tested:	xgl_object_create xgl_gcache_nurbs_surface xgl_context_new_frame xgl_context_display_gcache
Output:	Square with symmetric curve cut in the middle

Table 19-1 Nurbs Attributes Tested

Column A	Column B
XGL_3D_CTX_HLHSR_MODE	XGL_3D_CTX_HLHSR_MODE
XGL_CTX_LINE_COLOR	XGL_CTX_LINE_ALT_COLOR
XGL_CTX_NEW_FRAME_ACTION	XGL_CTX_LINE_COLOR
XGL_CTX_NEW_FRAME_CLEAR	XGL_CTX_LINE_PATTERN
XGL_CTX_NEW_FRAME_HLHSR_ ACTION	XGL_CTX_LINE_STYLE
XGL_CTX_NURBS_CURVE_APPROX	XGL_CTX_LINE_WIDTH_SCALE_ FACTOR
XGL_CTX_NURBS_CURVE_APPROX_VAL	XGL_CTX_MAX_TESSELLATION
XGL_HLHSR_Z_BUFFER	XGL_CTX_NEW_FRAME_ACTION
	XGL_CTX_NEW_FRAME_CLEAR
	XGL_CTX_NEW_FRAME_HLHSR_ ACTION
	XGL_CTX_NURBS_CURVE_APPROX
	XGL_CTX_NURBS_CURVE_APPROX_VAL
	XGL_CURVE_METRIC_WC
	XGL_HLHSR_Z_BUFFER
	XGL_LINE_ALT_PATTERNED
	XGL_LINE_SOLID

Pcache Test Descriptions 20

This chapter describes the Pcache test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section, "Denizen Test Types" on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ pcache1

Test Types:	RGB, SM
Description:	A Pcache object is created by `xgl_object_create()`. The attribute `XGL_PCACHE_CONTEXT` is used to set the context to the Pcache. The test checks the status returned by `xgl_pcache_display()` for all the combination of "test", "display" and "restore" of the function variables. `xgl_pcache_display()` is also tested for an empty Pcache. The front surface color is modified in Pcache by `xgl_object_set()` and checked if `xgl_pcache_display()` actually changes the context.

	xgl_context_new_frame() is used to empty and reopen the Pcache, and xgl_object_destroy() destroys the Pcache.
Attributes Tested:	XGL_PCACHE, XGL_PCACHE_CONTEXT XGL_CTX_SURF_FRONT_COLOR
Operators Tested:	xgl_object_create xgl_object_set xgl_object_get xgl_pcache_display xgl_context_new_frame xgl_multi_simple_polygon
Output:	One magenta square

▼ pcache2

Test Types: RGB, SM

Description:	The test creates a window raster, and a context. The primitive colors, scale factors, etc. are set in the context. The test next creates a Pcache object, then renders a multisimple polygon onto it. The Pcache is then displayed on the raster using the attributes in the context. This is repeated for triangle strip, triangle star, individual triangles, quadmesh, multipolyline, stroketext, and multimarker. All the renderings are done with the attributes stored in the context.
Attributes Tested:	XGL_CTX_BACKGROUND_COLOR XGL_CTX_SURF_FRONT_COLOR XGL_CTX_LINE_COLOR XGL_CTX_MARKER_COLOR XGL_CTX_STEXT_COLOR XGL_CTX_MARKER_SCALE_FACTOR XGL_CTX_MARKER XGL_CTX_STEXT_CHAR_HEIGHT XGL_FACET_FLAG_SHAPE_CONVEX XGL_FACET_FLAG_SIDES_ARE_3 XGL_TLIST_FLAG_TRI_STRIP XGL_TLIST_FLAG_IS_PLANAR XGL_TLIST_FLAG_FN_CONSISTENT

Operators Tested:	xgl_object_create xgl_object_set xgl_multi_simple_polygon xgl_triangle_list xgl_quadrilateral_mesh xgl_multipolyline xgl_stroke_text xgl_multimarker xgl_pcache_display
Output:	Five yellow triangles, yellow corner, yellow protruding rectangle, five triangles, another five triangles, a square, a polyline, text, and six yellow “+” markers

▼ pcache3

Test Types:	RGB, SM
Description:	Tests the general attributes supported by Pcache, i.e., the surface, line, marker, and pick attributes. A cube is rendered with `xgl_multi_simple_polygon()` to test the surface attributes. First, the default face cull is tested, then front face cull, back face cull. The front and back color selectors are next tested for the default color selector, `XGL_SURF_COLOR_CONTEXT`, and `XGL_SURF_COLOR_VERTEX_ILLUM_INDEP`. The back face is specified by changing `XGL_3D_CTX_SURF_GEOM_NORMAL` to `XGL_GEOM_NORMAL_LAST_POINTS`. The front and back illuminations are tested for `XGL_ILLUM_NONE_INTERP_COLOR`, `XGL_ILLUM_PER_FACET`, and `XGL_ILLUM_PER_VERTEX`. The line attribute test program tests for `XGL_CTX_LINE_COLOR_SELECTOR`, `XGL_CTX_LINE_WIDTH_SCALE_FACTOR`, and `XGL_CTX_LINE_STYLE`. The default color selector, `XGL_LINE_COLOR_CONTEXT` selector, and `XGL_LINE_PATTERNED` are tested. The marker attribute test program tests for default color selector, and context color selector. The pick attribute test program tests `XGL_CTX_PICK_ID_1` and `XGL_CTX_PICK_ID_2`. These attributes are changed in a single Pcache.

Attributes Tested:	XGL_3D_CTX_SURF_FACE_CULL XGL_3D_CTX_SURF_GEOM_NORMAL XGL_CTX_SURF_FRONT_COLOR_SELECTOR XGL_3D_CTX_SURF_BACK_COLOR_SELECTOR XGL_CTX_SURF_FRONT_COLOR XGL_3D_CTX_SURF_BACK_COLOR XGL_3D_CTX_SURF_FRONT_ILLUMINATION XGL_3D_CTX_SURF_BACK_ILLUMINATION XGL_CTX_LINE_COLOR_SELECTOR XGL_CTX_LINE_WIDTH_SCALE_FACTOR XGL_CTX_LINE_STYLE XGL_CTX_MARKER_COLOR_SELECTOR XGL_CTX_PICK_ID_1 XGL_CTX_PICK_ID_2
Operators Tested:	xgl_object_create xgl_object_set xgl_context_new_frame xgl_pcache_display xgl_multi_simple_polygon xgl_multipolyline xgl_multimarker xgl_pick_get_identifiers
Output:	Multi-colored cubes, lines, and “+” markers

Picking Test Descriptions 21

This chapter describes the Picking test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ pick_control

Test Types:	INDEX, SM
Description:	Tests that picking is allowed only when `XGL_CTX_PICK_ENABLE` is TRUE. The value of `XGL_CTX_RENDERING` is varied.
Attributes Tested:	See Table 21-1, Column A at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_multipolyline`
	`xgl_pick_get_identifiers`

Output:	Tests `xgl_multipolyline()` with:

- Default rendering mode true and default pick disabled (nothing should be picked)
- Default rendering mode true and pick enabled polyline should be picked)
- Rendering mode false and pick enabled (polyline should be picked)

pick_control_rgb

Test Types: RGB, SM
Description: Tests that picking is allowed only when `XGL_CTX_PICK_ENABLE` is TRUE. The value of `XGL_CTX_RENDERING` is varied.
Attributes Tested: See Table 21-1, Column A at the end of this chapter.
Operators Tested: `xgl_object_set`
`xgl_object_get`
`xgl_multipolyline`
`xgl_pick_get_identifiers`
Output: Tests `xgl_multipolyline()` with:

- Default rendering mode true and default pick disabled (nothing should be picked)
- Default rendering mode true and pick enabled (polyline should be picked)
- Rendering mode false and pick enabled (polyline should be picked)

pick_aperture

Test Types: INDEX, SM
Description: Tests that different 2D and 3D aperture sizes can be set. Also tries apertures that extend outside the DC viewport.
Attributes Tested: See Table 21-2, Column A at the end of this chapter.
Operators Tested: `xgl_object_set`
`xgl_multirectangle`
`xgl_polygon`
`xgl_pick_get_identifiers`
Output: Tests nine 2D pick apertures with `xgl_multirectangle()` (`XGL_MULTIRECT_I2D`). Tests thirteen 3D pick apertures with `xgl_polygon()` (`XGL_PT_F3D`).

pick_aperture_rgb

Test Types:	RGB, SM
Description:	Tests that different 2D and 3D aperture sizes can be set. Also tries apertures that extend outside the DC viewport.
Attributes Tested:	See Table 21-2, Column A at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_multirectangle`
	`xgl_polygon`
	`xgl_pick_get_identifiers`

Output:	Tests nine 2D pick apertures with `xgl_multirectangle()` (XGL_MULTIRECT_I2D). Tests thirteen 3D pick apertures tested with `xgl_polygon()` (XGL_PT_F3D).

pick_set_get_id

Test Types:	INDEX, SM
Description:	Tests that `xgl_object_set()` can be used to change the pick identifiers (XGL_CTX_PICK_ID_1/2). Subsequent primitives will have the same pick identifiers until the application changes any one of them. Tries pick primitives with the same/different pick ids.
Attributes Tested:	XGL_CTX_PICK_ID_2 and Table 21-2, Column A at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_triangle_strip`
	`xgl_multimarker`
	`xgl_object_get`

Output:	The following occurred: • Checks for default XGL_CTX_PICK_ID_1/2 (should be 0) and the pick aperture set • Displays 3D `xgl_multimarker()` (XGL_PT_F3D) and `xgl_triangle_strip()` (XGL_PT_F3D, XGL_FACET_COLOR_NORMAL) and verifies the display • Sets pick id1 and id2 and gets id1 and id2 and verifies • Displays the markers and the triangle strips again, and gets pick identifiers and verifies

▼ pick_2d_pp_id

Test Types:	INDEX, SM
Description:	Tests that the attributes XGL_CTX_PICK_ID_1 and XGL_CTX_PICK_ID_2 can be pushed and popped by xgl_context_push() and xgl_context_pop().
Attributes Tested:	See Table 21-1, Column B at the end of this chapter.
Operators Tested:	xgl_object_set xgl_multirectangle xgl_context_push xgl_pick_get_identifiers xgl_object_get xgl_context_pop
Output:	Pushes and pops the pick id attribute and displays xgl_multirectangle(), and gets pick id (the multirectangle should be picked)

▼ pick_set_get_id_rgb

Test Types:	RGB, SM
Description:	Tests that xgl_object_set() can be used to change the pick identifiers (XGL_CTX_PICK_ID_1/2). Subsequent primitives will have the same pick identifiers until the application changes any one of them. Tries pick primitives with the same/different pick ids.
Attributes Tested:	XGL_CTX_PICK_ID_2 and Table 21-2, Column A at the end of this chapter
Operators Tested:	xgl_object_set xgl_multimarker xgl_triangle_strip xgl_pick_get_identifiers xgl_object_get
Output:	The following occurred: • Checks for default XGL_CTX_PICK_ID_1/2 (should be 0) and the pick aperture set • Displays 3D xgl_multimarker() (XGL_PT_F3D) and xgl_triangle_strip() (XGL_PT_F3D, XGL_FACET_COLOR_NORMAL) and verifies the display • Sets pick id1 and id2 and gets id1 and id2 and verifies

- Displays the markers and triangle strips again, and gets pick identifiers and verifies

pick_2d_pp_id_rgb

Test Types: RGB, SM

Description: Tests that the attributes XGL_CTX_PICK_ID_1 and XGL_CTX_PICK_ID_2 can be pushed and popped by xgl_context_push() and xgl_context_pop()

Attributes Tested: See Table 21-1, Column B at the end of this chapter.

Operators Tested:
xgl_object_set
xgl_context_push
xgl_multirectangle
xgl_pick_get_identifiers
xgl_context_pop

Output: Pushes and pops the pick id attribute and displays xgl_multirectangle(), and gets pick id (the multirectangle should be picked)

pick_2d_buf

Test Types: INDEX, SM

Description: Tests that xgl_pick_clear() can clear the contents of the pick buffer, and that xgl_pick_get_identifiers() implicitly calls xgl_pick_clear() to clear the pick buffer

Attributes Tested: See Table 21-1, Column A at the end of this chapter.

Operators Tested:
xgl_object_set
xgl_multicircle
xgl_pick_clear
xgl_pick_get_identifiers

Output: One xgl_multicircle() is picked (XGL_MULTICIRCLE_F2D) with 10 different pick ids

▼ pick_2d_style

Test Types: INDEX, SM

Description: Tests that the value of XGL_CTX_PICK_STYLE controls the order in which the identifier pairs stored in the pick buffer are returned

Attributes Tested: XGL_CTX_PICK_STYLE
and Table 21-2, Column B at the end of this chapter

Operators Tested: xgl_object_set
xgl_multiarc
xgl_nurbs_curve
xgl_pick_get_identifiers
xgl_object_get

Output: Tests xgl_multiarc() (XGL_MULTIARC_F2D) and xgl_nurbs_curve() (XGL_PT_F2H) with different pick styles (default—XGL_PICK_FIRST_N, XGL_PICK_LAST_N)

▼ pick_2d_buf_overflow

Test Types: INDEX, SM

Description: Tests that the correct list of identifier pairs of picked primitives is returned when the pick buffer overflows

Attributes Tested: XGL_CTX_PICK_STYLE
XGL_CTX_PICK_BUFFER_SIZE
and Table 21-2, Column B at the end of this chapter

Operators Tested: xgl_object_set
xgl_multiarc
xgl_nurbs_curve
xgl_pick_get_identifiers
xgl_object_get

Output: Tests xgl_multiarc() (XGL_MULTIARC_F2D) and xgl_nurbs_curve()(XGL_PT_F2H) (total 300 primitives) with different pick buffer sizes and pick styles:

- Default buffer size of 256 with default style XGL_PICK_LAST_N
- Default buffer size of 256 with default style XGL_PICK_FIRST_N
- Buffer size of 10 with style XGL_PICK_LAST_N
- Buffer size of 10 with style XGL_PICK_FIRST_N

pick_2d_buf_size

Test Types: INDEX, SM

Description: Tests that various pick buffer sizes can be set via the context attribute XGL_CTX_PICK_BUFFER_SIZE

Attributes Tested: XGL_CTX_PICK_BUFFER_SIZE
XGL_CTX_ARC_FILL_STYLE (XGL_ARC_SECTOR)
instead of (XGL_ARC_CHORD)
and Table 21-2, Column B at the end of this chapter

Operators Tested: xgl_object_set
xgl_multiarc
xgl_nurbs_curve
xgl_pick_get_identifiers
xgl_object_get

Output: Tests xgl_multiarc() (XGL_MULTIARC_F2D) and xgl_nurbs_curve() (XGL_PT_F2H) with different pick buffer sizes: 1, 4, 50, 100, 258, 500

pick_2d_buf_rgb

Test Types: RGB, SM

Description: Tests that xgl_pick_clear() can clear the contents of the pick buffer, and that xgl_pick_get_identifiers() implicitly calls xgl_pick_clear() to clear the pick buffer

Attributes Tested: See Table 21-1, Column A at the end of this chapter.

Operators Tested: xgl_pick_clear
xgl_object_set
xgl_multicircle
xgl_pick_get_identifiers

Output: One xgl_multicircle() is picked (XGL_MULTICIRCLE_F2D) with ten different pick ids

pick_2d_style_rgb

Test Types: RGB, SM

Description: Tests that the value of XGL_CTX_PICK_STYLE controls the order in which the identifier pairs stored in the pick buffer are returned

Attributes Tested: XGL_CTX_PICK_STYLE
and Table 21-2, Column B at the end of this chapter

Operators Tested: xgl_object_set
xgl_multiarc
xgl_nurbs_curve
xgl_pick_get_identifiers
xgl_object_get

Output: Tests xgl_multiarc() (XGL_MULTIARC_F2D) and xgl_nurbs_curve() (XGL_PT_F2H) with different pick styles (default—XGL_PICK_FIRST_N, XGL_PICK_LAST_N)

pick_2d_buf_rgb_overflow

Test Types: RGB, SM

Description: Tests that the correct list of identifier pairs of picked primitives is returned when the pick buffer overflows

Attributes Tested: XGL_CTX_PICK_BUFFER_SIZE
and Table 21-2, Column B at the end of this chapter

Operators Tested: xgl_object_set
xgl_multiarc
xgl_nurbs_curve
xgl_pick_get_identifiers
xgl_object_get

Output: Tests xgl_multiarc() (XGL_MULTIARC_F2D) and xgl_nurbs_curve() (XGL_PT_F2H) (total 300 primitives) with different pick buffer sizes and pick styles:

- Default buffer size of 256 with default style XGL_PICK_LAST_N
- Default buffer size of 256 with style XGL_PICK_FIRST_N
- Buffer size of 10 with style XGL_PICK_LAST_N
- Buffer size of 10 with style XGL_PICK_FIRST_N

pick_2d_buf_size_rgb

Test Types: RGB, SM

Description: Tests that various pick buffer sizes can be set via the context attribute XGL_CTX_PICK_BUFFER_SIZE

Attributes Tested: XGL_CTX_PICK_BUFFER_SIZE
XGL_CTX_ARC_FILL_STYLE (XGL_ARC_SECTOR)
instead of (XGL_ARC_CHORD)
and Table 21-2, Column B at the end of this chapter

Operators Tested: xgl_object_set
xgl_multiarc
xgl_nurbs_curve
xgl_pick_get_identifiers
xgl_object_get

Output: Tests xgl_multiarc() (XGL_MULTIARC_F2D) and xgl_nurbs_curve() (XGL_PT_F2H) with different pick buffer sizes: 1, 4, 50, 100, 258, 500

▼ pick_rgb_primitives

Test Types: RGB, SM

Description: Tests that various primitives with default attributes can be picked

Attributes Tested: XGL_CTX_PICK_ID_1
XGL_CTX_PICK_ENABLE
XGL_CTX_PICK_APERTURE
XGL_3D_CTX_HLHSR_MODE

Operators Tested: xgl_object_set
xgl_pick_get_identifiers
xgl_multiarc
xgl_multicircle
xgl_multimarker
xgl_multipolyline
xgl_multirectangle
xgl_polygon
xgl_nurbs_curve
xgl_quadrilateral_mesh
xgl_triangle_strip
xgl_stroke_text (2D)

Output: Picks primitives xgl_multiarc(), xgl_multicircle(), xgl_multimarker(), xgl_multipolyline(), xgl_multirectangle(), xgl_polygon(), xgl_nurbs_curve(), xgl_quadrilateral_mesh(), xgl_triangle_strip(), xgl_stroke_text (2D)

pick_rgb_ndefault_primitives

Test Types: RGB, SM
Description: Tests that various primitives with nondefault attributes can be picked
Attributes Tested: See Table 21-3 at the end of this chapter.
Operators Tested: `xgl_object_set`
`xgl_pick_get_identifiers`
`xgl_multiarc`
`xgl_multicircle`
`xgl_multimarker`
`xgl_multipolyline`
`xgl_multirectangle`
`xgl_polygon`
`xgl_nurbs_curve`
`xgl_quadrilateral_mesh`
`xgl_triangle_strip`
`xgl_stroke_text` (3D)
Output: Picks primitives `xgl_multiarc()`, `xgl_multicircle()`, `xgl_multimarker()`, `xgl_multipolyline()`, `xgl_multirectangle()`, `xgl_polygon()`, `xgl_nurbs_curve()`, `xgl_quadrilateral_mesh()`, `xgl_triangle_strip()`, `xgl_stroke_text` (3D)

pick_2d_rgb_trans_clip_prim

Test Types: RGB, SM
Description: Tests whether transformed and clipped primitives can be picked in different pick apertures
Attributes Tested: See Table 21-1, Column C at the end of this chapter.
Operators Tested: `xgl_object_set`
`xgl_pick_get_identifiers`
`xgl_multirectangle`
`xgl_object_create`
`xgl_object_destroy`
`xgl_transform_scale`
`xgl_transform_rotate`
`xgl_transform_translate`

Output:	Tests nine pick apertures: picks `xgl_multirectangle()` in the first five pick apertures but not in the rest of the pick apertures

pick_primitives

Test Types:	INDEX, SM
Description:	Tests that various primitives with default attributes can be picked

Attributes Tested:	`XGL_CTX_PICK_ID_1`
	`XGL_CTX_PICK_ENABLE`
	`XGL_CTX_PICK_APERTURE`
	`XGL_3D_CTX_HLHSR_MODE`

Operators Tested:	`xgl_object_set`
	`xgl_pick_get_identifiers`
	`xgl_multiarc`
	`xgl_multicircle`
	`xgl_multimarker`
	`xgl_multipolyline`
	`xgl_multirectangle`
	`xgl_polygon`
	`xgl_nurbs_curve`
	`xgl_quadrilateral_mesh`
	`xgl_triangle_strip`
	`xgl_stroke_text` (2D)

Output:	Picks primitives `xgl_multiarc()`, `xgl_multicircle()`, `xgl_multimarker()`, `xgl_multipolyline()`, `xgl_multirectangle()`, `xgl_polygon()`, `xgl_nurbs_curve()`, `xgl_quadrilateral_mesh()`, `xgl_triangle_strip()`, `xgl_stroke_text` (2D)

pick_ndefault_primitives

Test Types:	INDEX, SM
Description:	Tests that various primitives with nondefault attributes can be picked
Attributes Tested:	See Table 21-3 at the end of this chapter.

Operators Tested:	xgl_object_set xgl_pick_get_identifiers xgl_multiarc xgl_multicircle xgl_multimarker xgl_multipolyline xgl_multirectangle xgl_polygon xgl_nurbs_curve xgl_quadrilateral_mesh xgl_triangle_strip xgl_stroke_text (3D)
Output:	Picks primitives xgl_multiarc(), xgl_multicircle(), xgl_multimarker(), xgl_multipolyline(), xgl_multirectangle(), xgl_polygon(), xgl_nurbs_curve(), xgl_quadrilateral_mesh(), xgl_triangle_strip(), xgl_stroke_text (3D)

▼ pick_2d_trans_clip_prim
pick_prims3

Test Types:	INDEX, SM
Description:	Tests whether transformed and clipped primitive can be picked in different pick apertures
Attributes Tested:	See Table 21-1, Column C at the end of this chapter.
Operators Tested:	xgl_object_set xgl_pick_get_identifiers xgl_multirectangle xgl_object_create xgl_object_destroy xgl_transform_scale xgl_transform_rotate xgl_transform_translate
Output:	Tests nine pick apertures: picks xgl_multirectangle() in the first five pick apertures but not in the rest of the pick apertures

Table 21-1 Picking Attributes Tested - Set 1

Column A	**Column B**	**Column C**
XGL_CTX_PICK_ID_1	XGL_CTX_PICK_ID_1	XGL_CTX_PICK_ID_1
XGL_CTX_PICK_ENABLE	XGL_CTX_PICK_ID_2	XGL_CTX_PICK_APERTURE
XGL_CTX_PICK_APERTURE	XGL_CTX_PICK_ENABLE	XGL_CTX_PICK_ENABLE
XGL_CTX_RENDERING	XGL_CTX_PICK_APERTURE	XGL_CTX_VIEW_TRANS
		XGL_CTX_VIEW_CLIP_BOUNDS
		XGL_CTX_CLIP_PLANES

Table 21-2 Picking Attributes Tested - Set 2

Column A	**Column B**
XGL_CTX_PICK_ID_1	XGL_CTX_PICK_ID_1
XGL_CTX_PICK_ENABLE	XGL_CTX_PICK_ID_2
XGL_CTX_PICK_APERTURE	XGL_CTX_PICK_ENABLE
XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER)	XGL_CTX_PICK_APERTURE
XGL_CTX_NEW_FRAME_ACTION (XGL_CTX_NEW_FRAME_CLEAR | (XGL_CTX_NEW_FRAME_HLHSR_ACTION)	XGL_CTX_RENDERING
XGL_CTX_RENDERING	XGL_CTX_ARC_FILL_STYLE (XGL_ARC_CHORD)
	XGL_CTX_NURBS_CURVE_APPROX
	XGL_CTX_NURBS_CURVE_APPROX_VAL

Table 21-3 Picking Attributes Tested - Set 3

XGL_CTX_PICK_ID_1	XGL_CTX_PICK_ENABLE	XGL_CTX_PICK_APERTURE
XGL_CTX_EDGE_COLOR	XGL_CTX_SURF_EDGE_FLAG	XGL_3D_CTX_HLHSR_MODE
XGL_CTX_ARC_FILL_STYLE	XGL_CTX_SURF_FRONT_FILL_ STYLE	XGL_CTX_MARKER_SCALE_ FACTOR
XGL_CTX_MARKER	XGL_CTX_MARKER_COLOR	XGL_CTX_LINE_COLOR
XGL_CTX_LINE_CAP	XGL_CTX_LINE_JOIN	XGL_CTX_LINE_PATTERN
XGL_CTX_LINE_STYLE	XGL_CTX_LINE_WIDTH_SCALE_ FACTOR	XGL_CTX_NURBS_CURVE_APPROX
XGL_CTX_NURBS_CURVE_APPROX _VAL	XGL_CTX_STEXT_CHAR_HEIGHT	XGL_CTX_STEXT_CHAR_SPACING
XGL_CTX_STEXT_CHAR_ EXPANSION_FACTOR	XGL_CTX_STEXT_PATH	XGL_CTX_STEXT_CHAR_UP_ VECTOR,
XGL_CTX_STEXT_ALIGN_HORIZ	XGL_CTX_STEXT_ALIGN_VERT	XGL_CTX_STEXT_COLOR

Polygon Test Descriptions 22

This chapter describes the Polygon test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ pg_simple

Test Types:	INDEX, SM
Description:	Draws a four-sided solid fill polygon of point type `XGL_PT_F3D` twice. The first time the polygon has a facet type of `XGL_FACET_COLOR`, and the second time it has a facet type of `XGL_FACET_NORMAL`.

Attributes Tested:	`XGL_CTX_SURF_EDGE_FLAG (FALSE)`
	`XGL_CTX_SURF_FRONT_FILL_STYLE`
	`(XGL_SURF_FILL_SOLID)`

Operators Tested:	xgl_object_get, xgl_object_set, xgl_polygon
Output:	First draws a yellow square then a red square at the same position

▼ pg_simple_rgb

Test Types:	RGB, SM
Description:	Same test as *pg_simple* except that this test is RGB
Attributes Tested:	XGL_CTX_SURF_EDGE_FLAG (FALSE) XGL_CTX_SURF_FRONT_FILL_STYLE (XGL_SURF_FILL_SOLID) XGL_CTX_SURF_FRONT_COLOR
Operators Tested:	xgl_object_get xgl_object_set xgl_polygon
Output:	First draws a red square then a cyan square at the same position

▼ pg0

Test Types:	RGB, SM
Description:	Tests linear depth-cueing of polygons: varies polygon shape and depth, and varies depth-cueing color
Attributes Tested:	XGL_3D_CTX_DEPTH_CUE_COLOR XGL_3D_CTX_DEPTH_CUE_MODE (XGL_DEPTH_CUE_LINEAR) XGL_3D_CTX_SURF_BACK_COLOR XGL_CTX_SURF_FRONT_COLOR
Operators Tested:	xgl_object_get xgl_object_set xgl_polygon
Output:	Draws each of the following five general polygons nine times (each time a different depth cue color is used): (1) A square (2) A five-sided polygon (3) A triangle and a four-sided polygon (4) A large triangle

(5) Four polygons with different shapes
Some of the rendered polygons appear shaded and some do not.

pg2

Test Types: RGB, SM

Description: Tests polygon vertex color interpolation: draws various polygons with XGL_ILLUM_NONE_INTERP_COLOR mode

Attributes Tested: XGL_3D_CTX_SURF_FRONT_ILLUMINATION (XGL_ILLUM_NONE_INTERP_COLOR)

Operators Tested: xgl_object_set
xgl_polygon

Output: The following seven polygons are rendered one after the other:
(1) A shaded square with vertical stripes
(2) A shaded square with horizontal stripes
(3) A shaded square
(4) A blue solid square
(5) A shaded vertical bow tie with horizontal stripes
(6) A blue solid horizontal bow tie
(7) A shaded triangle with horizontal stripes

pg3

Test Types: RGB, SM

Description: Tests polygon XGL_3D_CTX_SURF_GEOM_NORMAL: draws two geometrically identical triangles, one with XGL_GEOM_NORMAL_FIRST_POINTS and one with XGL_GEOM_NORMAL_LAST_POINTS, so one should be front facing and one back facing, turns on face distinguish so they appear in different colors and checks the colors

Attributes Tested: XGL_3D_CTX_SURF_BACK_COLOR
XGL_3D_CTX_SURF_FACE_DISTINGUISH
XGL_3D_CTX_SURF_GEOM_NORMAL
XGL_CTX_SURF_FRONT_COLOR

Operators Tested: xgl_object_get
xgl_object_set
xgl_polygon

Output:	First draws a red triangle then a blue triangle at the same position

▼ pg4

Test Types:	INDEX, SM
Description:	Tests indexed linear depth-cueing of polygons: varies polygon shape and depth

Attributes Tested:	`XGL_3D_CTX_DEPTH_CUE_MODE (XGL_DEPTH_CUE_LINEAR)`
	`XGL_3D_CTX_SURF_BACK_COLOR`
	`XGL_CTX_SURF_FRONT_COLOR`

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_polygon`

Output:	Draws the following five general polygons one after the other: (1) A shaded four-sided polygon (2) A gray solid square (3) A gray solid triangle and a gray solid four-sided polygon (4) A shaded large triangle (5) Three shaded polygons with different shapes

▼ pg_cull

Test Types:	INDEX, SM
Description:	Tests the three face-culling modes: draws both front- and back-facing polygons and checks that the correct ones are culled in each mode

Attributes Tested:	`XGL_3D_CTX_SURF_BACK_FILL_STYLE (XGL_SURF_FILL_SOLID)`
	`XGL_3D_CTX_SURF_FACE_CULL`
	`XGL_3D_CTX_SURF_FACE_DISTINGUISH`
	`XGL_CTX_SURF_FRONT_FILL_STYLE (XGL_SURF_FILL_SOLID)`

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_polygon`

Output:	Renders first a red square and a green square. Then the red square disappears and the green square remains on the screen. Finally, the red square is rendered again and the green square disappears.

▼ pg_cull_z

Test Types:	INDEX, SM
Description:	Tests the three face-culling modes with the z-buffer on: draws both front- and back-facing polygons and checks the correct ones are culled in each mode

Attributes Tested:	`XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER)`
	`XGL_3D_CTX_SURF_FACE_CULL`
	`XGL_3D_CTX_SURF_FACE_DISTINGUISH`
	`XGL_CTX_NEW_FRAME_ACTION`
	`(XGL_CTX_NEW_FRAME_CLEAR |`
	`XGL_CTX_NEW_FRAME_HLHSR_ACTION)`

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_polygon`

Output:	Renders first a red square and a green square. Then the red square disappears and the green square remains on the screen. Finally, the red square is rendered again and the green square disappears.

▼ pg_cull_rgb

Test Types:	RGB, SM
Description:	Tests the three face-culling modes in RGB: draws both front- and back-facing polygons and checks that the correct ones are culled in each mode

Attributes Tested:	`XGL_3D_CTX_SURF_BACK_FILL_STYLE`
	`(XGL_SURF_FILL_SOLID)`
	`XGL_3D_CTX_SURF_FACE_CULL`
	`XGL_3D_CTX_SURF_FACE_DISTINGUISH`
	`XGL_CTX_SURF_FRONT_FILL_STYLE`
	`(XGL_SURF_FILL_SOLID)`

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_polygon`

Output: Renders first an orange square and a green square. Then the orange square disappears and the green square remains on the screen. Finally, the orange square is rendered again and the green square disappears.

▼ pg_cull_z_rgb

Test Types: RGB, SM

Description: Tests the three face-culling modes in RGB with the z-buffer on: draws both front- and back-facing polygons and checks the correct ones are culled in each mode.

Attributes Tested:
XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER)
XGL_3D_CTX_SURF_FACE_CULL
XGL_3D_CTX_SURF_FACE_DISTINGUISH
XGL_CTX_NEW_FRAME_ACTION
(XGL_CTX_NEW_FRAME_CLEAR |
XGL_CTX_NEW_FRAME_HLHSR_ACTION)

Operators Tested:
xgl_object_get
xgl_object_set
xgl_polygon

Output: Renders first an orange square and a green square. Then the orange square disappears and the green square remains on the screen. Finally, the orange square is rendered again and the green square disappears.

▼ pg_edge

Test Types: RGB, SM

Description: Tests that alt-patterned edges for 2D RGB polygons are all drawn exactly correctly and that there are no “holes” inside the polygons. Tries many different colors for the polygon (all colors in color cube for 8-bit rasters).

Attributes Tested:
XGL_CTX_EDGE_ALT_COLOR
XGL_CTX_EDGE_COLOR
XGL_CTX_EDGE_PATTERN
XGL_CTX_EDGE_STYLE (XGL_LINE_ALT_PATTERNED)
XGL_CTX_SURF_FRONT_COLOR
XGL_LPAT_DATA
XGL_LPAT_DATA_SIZE

Operators Tested:	`xgl_context_get_pixel` `xgl_polygon`
Output:	For 8-bit rasters, a small solid square with edges is drawn many times, each time a different color in the color cube is used as the interior color. For non-8-bit rasters, nothing is rendered.

▼ pg_edge2

Test Types:	RGB, SM
Description:	Tests edge-enable flags for RGB polygons: draws a square with all sixteen combinations of edges and checks for the presence/absence of edges in each case.
Attributes Tested:	`XGL_CTX_EDGE_COLOR` `XGL_CTX_SURF_EDGE_FLAG`

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_polygon`

Output:	Draws a green square 16 times, each time with a different combination of edge flags for the four edges

▼ pg_edge3

Test Types:	RGB, SM
Description:	Tests that alt-patterned edges for 3D RGB polygons are all drawn exactly correctly and that there are no “holes” inside the polygons. Tries many different colors for the polygon (all colors in color cube for 8-bit rasters).

Attributes Tested:	`XGL_CTX_EDGE_ALT_COLOR`
	`XGL_CTX_EDGE_COLOR`
	`XGL_CTX_EDGE_PATTERN`
	`XGL_CTX_EDGE_STYLE (XGL_LINE_ALT_PATTERNED)`
	`XGL_CTX_SURF_FRONT_COLOR`
	`XGL_LPAT_DATA`
	`XGL_LPAT_DATA_SIZE`

Operators Tested:	`xgl_context_get_pixel` `xgl_polygon`

Output:	For 8-bit rasters, a small solid square with edges is drawn many times, each time a different color in the color cube is used as the interior color. For non-8-bit rasters, nothing is rendered.

▼ pg_edge4

Test Types:	RGB, CM
Description:	Tests edge-enable flags for 3D RGB polygons: cycles through various combinations of edge flag settings
Attributes Tested:	`XGL_CTX_EDGE_COLOR`
Operators Tested:	`xgl_object_set` `xgl_polygon`
Output:	Image with sixteen polygons with various edges illuminated

▼ pg_face

Test Types:	INDEX, SM
Description:	Tests that polygons are rendered correctly when `XGL_3D_CTX_SURF_FACE_DISTINGUISH` is TRUE / FALSE and `XGL_3D_CTX_SURF_NORMAL_FLIP` is TRUE / FALSE

Attributes Tested:	`XGL_3D_CTX_SURF_BACK_FILL_STYLE`
	`(XGL_SURF_FILL_HOLLOW)`
	`XGL_3D_CTX_SURF_FACE_DISTINGUISH`
	`XGL_3D_CTX_SURF_NORMAL_FLIP`
	`XGL_CTX_SURF_FRONT_FILL_STYLE`
	`(XGL_SURF_FILL_SOLID)`

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_polygon`

Output:	Draws three four-sided polygons three times. The first time, the three polygons consist of (from left to right) one red solid polygon and two green hollow polygons. The second time, the three polygons are all red solid polygons. The third time, the three polygons consist of one green hollow polygon and two red solid polygons.

pg_face_z

Test Types: INDEX, SM

Description: Tests that polygons are rendered correctly with the z-buffer on, when XGL_3D_CTX_SURF_FACE_DISTINGUISH is TRUE/FALSE and XGL_3D_CTX_SURF_NORMAL_FLIP is TRUE/FALSE

Attributes Tested:
XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER)
XGL_3D_CTX_SURF_FACE_DISTINGUISH
XGL_3D_CTX_SURF_NORMAL_FLIP
XGL_CTX_NEW_FRAME_ACTION
(XGL_CTX_NEW_FRAME_CLEAR |
XGL_CTX_NEW_FRAME_HLHSR_ACTION)

Operators Tested:
xgl_object_get
xgl_object_set
xgl_polygon

Output: Draws three four-sided polygons three times. The first time, the three polygons consist of (from left to right) one red solid polygon and two green hollow polygons. The second time, the three polygons are all red solid polygons. The third time, the three polygons consist of one green hollow polygon and two red solid polygons.

pg_face_rgb

Test Types: RGB, SM

Description: Tests that RGB polygons are rendered correctly when XGL_3D_CTX_SURF_FACE_DISTINGUISH is TRUE/FALSE and XGL_3D_CTX_SURF_NORMAL_FLIP is TRUE/FALSE

Attributes Tested:
XGL_3D_CTX_SURF_BACK_FILL_STYLE
(XGL_SURF_FILL_HOLLOW)
XGL_3D_CTX_SURF_FACE_DISTINGUISH
XGL_3D_CTX_SURF_NORMAL_FLIP
XGL_CTX_SURF_FRONT_FILL_STYLE
(XGL_SURF_FILL_SOLID)

Operators Tested:
xgl_object_get
xgl_object_set
xgl_polygon

Output: Draws three four-sided polygons three times. The first time, the three polygons consist of (from left to right) one orange solid polygon and two green hollow polygons. The second time, the three polygons are all orange solid polygons. The third time, the three polygons consist of one green hollow polygon and two orange solid polygons.

▼ pg_face_z_rgb

Test Types: RGB, SM

Description: Tests that RGB polygons are rendered correctly with the z-buffer on, when `XGL_3D_CTX_SURF_FACE_DISTINGUISH` is TRUE / FALSE and `XGL_3D_CTX_SURF_NORMAL_FLIP` is TRUE / FALSE

Attributes Tested:
`XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER)`
`XGL_3D_CTX_SURF_FACE_DISTINGUISH`
`XGL_3D_CTX_SURF_NORMAL_FLIP`
`XGL_CTX_NEW_FRAME_ACTION`
`(XGL_CTX_NEW_FRAME_CLEAR |`
`XGL_CTX_NEW_FRAME_HLHSR_ACTION)`

Operators Tested:
`xgl_object_get`
`xgl_object_set`
`xgl_polygon`

Output: Draws three four-sided polygons three times. The first time, the three polygons consist of (from left to right) one orange solid polygon and two green hollow polygons. The second time, the three polygons are all orange solid polygons. The third time, the three polygons consist of one green hollow polygon and two orange solid polygons.

▼ pg_fill

Test Types: INDEX, SM

Description: Tests various polygon fill styles: solid, hollow, and empty with edge. Draws a few polygons with each fill style and checks that they're solid / hollow polygons.

Attributes Tested: XGL_CTX_EDGE_COLOR
XGL_CTX_EDGE_WIDTH_SCALE_FACTOR
XGL_CTX_SURF_EDGE_FLAG
XGL_CTX_SURF_FRONT_FILL_STYLE

Operators Tested: xgl_object_get
xgl_object_set
xgl_polygon

Output: Renders each of the following six general polygons three times (solid, hollow, empty with edge), one after the other:
(1) A square with the facet type XGL_FACET_COLOR
(2) A square with the facet type XGL_FACET_NORMAL
(3) Nine triangles
(4) A self-intersecting polygon with 10 points
(5) Two totally overlapping polygons
(6) Two partially overlapping polygons

▼ pg_fill_z

Test Types: INDEX, SM

Description: Tests various polygon fill styles: solid, hollow, and empty with edge. Draws a few polygons with each fill style and checks that they're solid / hollow polygons. The Z-buffer is on.

Attributes Tested: XGL_3D_CTX_HLHSR_MODE
XGL_CTX_NEW_FRAME_ACTION
XGL_CTX_SURF_EDGE_FLAG
XGL_CTX_SURF_FRONT_FILL_STYLE

Operators Tested: xgl_object_get
xgl_object_set
xgl_polygon

Output: Renders each of the following six general polygons three times (solid, hollow, empty with edge), one after the other:
(1) A square with the facet type XGL_FACET_COLOR
(2) A square with the facet type XGL_FACET_NORMAL
(3) Nine triangles
(4) A self-intersecting polygon with 10 points
(5) Two totally overlapping polygons
(6) Two partially overlapping polygons

pg_fill2

Test Types:	INDEX, SM
Description:	Draws two polygons with edges on / off and with three different stipple patterns. In each case, checks for the correctness of the edges and the patterns inside.
Attributes Tested:	XGL_CTX_SURF_EDGE_FLAG XGL_CTX_SURF_FPAT XGL_CTX_SURF_FPAT_POSITION XGL_CTX_SURF_FRONT_FILL_STYLE (XGL_SURF_FILL_STIPPLE)
Operators Tested:	xgl_object_get xgl_object_set xgl_polygon
Output:	Draws two four-sided white polygons six times, one after the other, with the following stipple pattern styles and edge flags: (1) Dots stipple pattern, no edge (2) Cross hatch stipple pattern, no edge (3) Dotted screen stipple pattern, no edge (4) Dots stipple pattern with edge (5) Cross hatch stipple pattern with edge (6) Dotted screen stipple pattern with edge

pg_fill_rgb

Test Types:	RGB, SM
Description:	Tests various RGB polygon fill styles: solid, hollow, and empty with edge. Draws a few polygons with each fill style and checks that they're solid / hollow polygons.
Attributes Tested:	XGL_CTX_EDGE_COLOR XGL_CTX_EDGE_WIDTH_SCALE_FACTOR XGL_CTX_SURF_EDGE_FLAG XGL_CTX_SURF_FRONT_FILL_STYLE
Operators Tested:	xgl_object_get xgl_object_set xgl_polygon
Output:	Renders each of the following six general polygons three times (solid, hollow, empty with edge), one after the other: (1) A square with the facet type XGL_FACET_COLOR

(2) A square with the facet type XGL_FACET_NORMAL
(3) Nine triangles
(4) A self-intersecting polygon with ten points
(5) Two totally overlapping polygons
(6) Two partially overlapping polygons

▼ pg_fill_z_rgb

Test Types: RGB, SM

Description: Tests various RGB polygon fill styles: solid, hollow, and empty with edge. Draws a few polygons with each fill style and checks that they're solid / hollow polygons. The Z-buffer is on.

Attributes Tested:
XGL_3D_CTX_HLHSR_MODE
XGL_CTX_NEW_FRAME_ACTION
XGL_CTX_SURF_EDGE_FLAG
XGL_CTX_SURF_FRONT_FILL_STYLE

Operators Tested:
xgl_object_get
xgl_object_set
xgl_polygon

Output: Renders each of the following six general polygons three times (solid, hollow, empty with edge), one after the other:
(1) A square with the facet type XGL_FACET_COLOR
(2) A square with the facet type XGL_FACET_NORMAL
(3) Nine triangles
(4) A self-intersecting polygon with ten points
(5) Two totally overlapping polygons
(6) Two partially overlapping polygons

▼ pg_fill4

Test Types: RGB, SM

Description: Draws two 2D RGB polygons with edges on / off and with three different stipple patterns. In each case, checks for the correctness of the edges and the patterns inside.

Attributes Tested:
XGL_CTX_SURF_EDGE_FLAG
XGL_CTX_SURF_FPAT
XGL_CTX_SURF_FPAT_POSITION
XGL_CTX_SURF_FRONT_FILL_STYLE
(XGL_SURF_FILL_STIPPLE)

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_polygon`

Output:	Draws two four-sided orange polygons six times, one after the other, with the following stipple pattern styles and edge flags: (1) Dots stipple pattern, no edge (2) Cross hatch stipple pattern, no edge (3) Dotted screen stipple pattern, no edge (4) Dots stipple pattern with edge (5) Cross hatch stipple pattern with edge (6) Dotted screen stipple pattern with edge

▼ pg_fill5

Test Types:	RGB, SM
Description:	Draws two 2D RGB polygons with three different opaque stipple patterns. In each case, checks for the correctness of the patterns inside.

Attributes Tested:	`XGL_CTX_SURF_FPAT`
	`XGL_CTX_SURF_FPAT_POSITION`
	`XGL_CTX_SURF_FRONT_FILL_STYLE`
	`(XGL_SURF_FILL_OPAQUE_STIPPLE)`

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_polygon`

Output:	Draws two four-sided orange polygons three times, one after the other, with the following opaque stipple pattern styles, on top of a solid fill polygon drawn at the same position: (1) Dots stipple pattern (2) Cross hatch stipple pattern (3) Dotted screen stipple pattern

▼ pg_fill6

Test Types:	RGB, SM
Description:	Tests various 2D RGB polygon fill styles: solid, hollow, and empty with edge. Draws a few polygons with each fill style and checks that they're solid / hollow polygons.

Attributes Tested:	XGL_CTX_EDGE_COLOR XGL_CTX_EDGE_WIDTH_SCALE_FACTOR XGL_CTX_SURF_EDGE_FLAG XGL_CTX_SURF_FRONT_FILL_STYLE
Operators Tested:	xgl_object_get xgl_object_set xgl_polygon
Output:	Renders each of the following six general polygons three times (solid, hollow, empty with edge), one after the other: (1) A square with the facet type XGL_FACET_NORMAL (2) A square with the facet type XGL_FACET_COLOR (3) Nine triangles (4) A self-intersecting polygon with 10 points (5) Two totally overlapping polygons (6) Two partially overlapping polygons

▼ pg_fill7

Test Types:	RGB, SM
Description:	Draws two 3D RGB polygons with edges on / off and with three different stipple patterns. In each case, checks for the correctness of the edges and the patterns inside.
Attributes Tested:	XGL_CTX_SURF_EDGE_FLAG XGL_CTX_SURF_FPAT XGL_CTX_SURF_FPAT_POSITION XGL_CTX_SURF_FRONT_FILL_STYLE (XGL_SURF_FILL_STIPPLE)
Operators Tested:	xgl_object_get xgl_object_set xgl_polygon
Output:	Draws two four-sided orange polygons six times, one after the other, with the following stipple pattern styles and edge flags: (1) Dots stipple pattern, no edge (2) Cross hatch stipple pattern, no edge (3) Dotted screen stipple pattern, no edge (4) Dots stipple pattern with edge (5) Cross hatch stipple pattern with edge (6) Dotted screen stipple pattern with edge

pg_fill8

Test Types:	RGB, SM
Description:	Draws two 3D RGB polygons with three different opaque stipple patterns. In each case, checks for the correctness of the patterns inside.
Attributes Tested:	XGL_CTX_SURF_FPAT XGL_CTX_SURF_FPAT_POSITION XGL_CTX_SURF_FRONT_FILL_STYLE (XGL_SURF_FILL_OPAQUE_STIPPLE)
Operators Tested:	xgl_object_get xgl_object_set xgl_polygon
Output:	Draws two four-sided orange polygons three times, one after the other, with the following opaque stipple pattern styles, on top of a solid fill polygon drawn at the same position: (1) Dots stipple pattern (2) Cross-hatch stipple pattern (3) Dotted-screen stipple pattern

pg_back_fill_rgb

Test Types:	RGB, SM
Description:	Tests various RGB polygon back fill styles: solid, hollow, and empty with edge. Draws a few polygons with each fill style and checks that they're solid / hollow polygons.
Attributes Tested:	XGL_3D_CTX_SURF_BACK_FILL_STYLE XGL_3D_CTX_SURF_FACE_DISTINGUISH XGL_CTX_EDGE_COLOR XGL_CTX_EDGE_WIDTH_SCALE_FACTOR XGL_CTX_SURF_EDGE_FLAG
Operators Tested:	xgl_object_get xgl_object_set xgl_polygon
Output:	Renders each of the following six general polygons three times (solid, hollow, empty with edge), one after the other: (1) A square with the facet type XGL_FACET_COLOR_NORMAL (2) A square with the facet type XGL_FACET_NORMAL

(3) Nine triangles
(4) A self-intersecting polygon with ten points
(5) Two totally overlapping polygons
(6) Two partially overlapping polygons

▼ pg_back_fill_z_rgb

Test Types: RGB, SM

Description: Tests various RGB polygon back-fill styles: solid, hollow, and empty with edge. Draws a few polygons with each fill style and checks that they're solid / hollow polygons. The Z-buffer is on.

Attributes Tested: XGL_3D_CTX_HLHSR_MODE
XGL_3D_CTX_SURF_BACK_FILL_STYLE
XGL_3D_CTX_SURF_FACE_DISTINGUISH
XGL_CTX_NEW_FRAME_ACTION
XGL_CTX_SURF_EDGE_FLAG

Operators Tested: xgl_object_get
xgl_object_set
xgl_polygon

Output: Renders each of the following six general polygons three times (solid, hollow, empty with edge), one after the other:
(1) A square with the facet type XGL_FACET_COLOR_NORMAL
(2) A square with the facet type XGL_FACET_NORMAL
(3) Nine triangles
(4) A self-intersecting polygon with ten points
(5) Two totally overlapping polygons
(6) Two partially overlapping polygons

▼ pg_fill10

Test Types: RGB, SM

Description: Draws two 3D RGB polygons with edges on / off and with three different back fill stipple patterns. In each case, checks for the correctness of the edges and the patterns inside.

Attributes Tested: XGL_3D_CTX_SURF_BACK_FPAT
XGL_3D_CTX_SURF_BACK_FPAT_POSITION
XGL_3D_CTX_SURF_FACE_DISTINGUISH

Operators Tested: `xgl_object_get`
`xgl_object_set`
`xgl_polygon`

Output: Draws two four-sided orange polygons six times, one after the other, with the following stipple pattern styles and edge flags:
(1) Dots stipple pattern, no edge
(2) Cross hatch stipple pattern, no edge
(3) Dotted screen stipple pattern, no edge
(4) Dots stipple pattern with edge
(5) Cross hatch stipple pattern with edge
(6) Dotted screen stipple pattern with edge

▼ pg_fill11

Test Types: RGB, SM

Description: Draws two 3D RGB back-filled polygons with three different opaque stipple patterns. In each case, checks for the correctness of the patterns inside.

Attributes Tested: `XGL_3D_CTX_SURF_BACK_FILL_STYLE`
`(XGL_SURF_FILL_OPAQUE_STIPPLE)`
`XGL_3D_CTX_SURF_BACK_FPAT`
`XGL_3D_CTX_SURF_BACK_FPAT_POSITION`
`XGL_3D_CTX_SURF_FACE_DISTINGUISH`

Operators Tested: `xgl_object_get`
`xgl_object_set`
`xgl_polygon`

Output: Draws two four-sided orange polygons three times, one after the other, with the following opaque stipple pattern styles, on top of a solid-fill polygon drawn at the same position:
(1) Dots stipple pattern
(2) Cross hatch stipple pattern
(3) Dotted screen stipple pattern

pg_hlhsr

Test Types:	INDEX, SM
Description:	Tests the polygon hidden surface removal: draws two polygons at the same position but different depth and checks that only the front one is drawn; tries different depth combinations

Attributes Tested:	`XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER)`
	`XGL_CTX_NEW_FRAME_ACTION`
	`(XGL_CTX_NEW_FRAME_CLEAR |`
	`XGL_CTX_NEW_FRAME_HLHSR_ACTION`

Operators Tested:	`xgl_object_set`
	`xgl_polygon`
	`xgl_context_post`
	`xgl_context_new_frame`

Output:	Draws four solid polygons (a square, a vertical bow tie, a horizontal bow tie, and a triangle) three times, one after the another. The first time they are drawn in red. The second time and the third time, they are drawn in green.

pg_hlhsr_2

Test Types:	INDEX, SM
Description:	Tests the *hlhsr* polygons with data: clears the Z-buffer to a specific value and then draws a polygon with a different depth; the polygon should only show up if its depth is less than the depth of the Z-buffer; tries different depth combinations

Attributes Tested:	`XGL_3D_CTX_HLHSR_DATA`
	`XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER)`
	`XGL_CTX_NEW_FRAME_ACTION`
	`(XGL_CTX_NEW_FRAME_CLEAR |`
	`XGL_CTX_NEW_FRAME_HLHSR_ACTION`

Operators Tested:	`xgl_object_set`
	`xgl_polygon`
	`xgl_context_post`
	`xgl_context_new_frame`

Output:	Draws a red square four times

pg_hlhsr_3

Test Types: RGB, SM

Description: Tests the RGB polygon hidden surface removal: draws two polygons at the same position but different depth and checks that only the front one is drawn; tries different depth combinations

Attributes Tested: `XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER)`
`XGL_CTX_NEW_FRAME_ACTION`
`(XGL_CTX_NEW_FRAME_CLEAR |`
`XGL_CTX_NEW_FRAME_HLHSR_ACTION`

Operators Tested: `xgl_object_set`
`xgl_polygon`
`xgl_context_post`
`xgl_context_new_frame`

Output: Draws four solid polygons (a square, a vertical bow tie, a horizontal bow tie, and a triangle) three times, one after the other. The first time they are drawn in orange color. The second time and the third time, they are drawn in green.

pg_hlhsr_4

Test Types: RGB, SM

Description: Tests the *hlhsr* RGB polygons with data: clears the Z-buffer to a specific value and then draws a polygon with a different depth; the polygon should only show up if its depth is less than the depth of the Z-buffer; tries different depth combinations

Attributes Tested: `XGL_3D_CTX_HLHSR_DATA`
`XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER)`
`XGL_CTX_NEW_FRAME_ACTION`
`(XGL_CTX_NEW_FRAME_CLEAR |`
`XGL_CTX_NEW_FRAME_HLHSR_ACTION`

Operators Tested: `xgl_object_set`
`xgl_polygon`
`xgl_context_post`
`xgl_context_new_frame`

Output: Draws an orange square four times

pg_intrule

Test Types: INDEX, SM

Description: Draws various polygons with the `XGL_EVEN_ODD` interior rule using xgl_polygon(); checks their correctness

Attributes Tested: `XGL_CTX_SURF_INTERIOR_RULE` (`XGL_EVEN_ODD`)

Operators Tested:
`xgl_object_get`
`xgl_object_set`
`xgl_polygon`

Output: Draws each of the following five general polygons twice (first time using a 3D context, second time using a 2D context):

(1) A self-intersecting polygon
(2) Two totally overlapping squares
(3) Two totally overlapping squares that look the same as number 2
(4) Two partially overlapping polygons
(5) A self-intersecting polygon that looks the same as number 1

pg_intrule2

Test Types: RGB, SM

Description: Draws various RGB polygons with the `XGL_EVEN_ODD` interior rule using xgl_polygon(); checks their correctness

Attributes Tested: `XGL_CTX_SURF_INTERIOR_RULE` (`XGL_EVEN_ODD`)

Operators Tested:
`xgl_object_get`
`xgl_object_set`
`xgl_polygon`

Output: Draws each of the following five general polygons twice (first time using a 3D context, second time using a 2D context):

(1) A self-intersecting polygon
(2) Two totally overlapping squares
(3) Two totally overlapping squares that look the same as number 2
(4) Two partially overlapping polygons
(5) A self-intersecting polygon that looks the same as number 1

pg_pttypes

Test Types: INDEX, SM

Description: Tests that polygons can be rendered using different point types: draws a vertical bow tie using six different point types and checks that they're all drawn correctly; draws a 400-vertex polygon(circle) and checks that it's drawn right

Attributes Tested: None

Operators Tested: `xgl_polygon`

Output: Draws a white vertical bow tie six times at the same position. Finally, a white circle is drawn.

pg_pttypes2

Test Types: RGB, SM

Description: Tests that RGB polygons can be rendered using different point types: draws a vertical bow tie using six different point types and checks that they're all drawn correctly; draws a 400-vertex polygon(circle) and checks that its drawn right.

Attributes Tested: None

Operators Tested: `xgl_polygon`

Output: Draws an orange vertical bow tie six times at the same position. Finally, an orange circle is drawn.

pg_shade

Test Types: INDEX, SM

Description: Draws a few different polygons with an ambient light on and checks that they're shaded correctly

Attributes Tested:
`XGL_3D_CTX_LIGHTS`
`XGL_3D_CTX_LIGHT_NUM`
`XGL_3D_CTX_LIGHT_SWITCHES`
`XGL_3D_CTX_SURF_FRONT_AMBIENT`
`XGL_3D_CTX_SURF_FRONT_ILLUMINATION`
`XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT`
`XGL_LIGHT_COLOR`
`XGL_LIGHT_TYPE`

Operators Tested: `xgl_object_get`
`xgl_object_set`
`xgl_polygon`

Output: Renders the following seven polygons one after the other:
(1) A shaded square with vertical stripes
(2) A shaded square with horizontal stripes
(3) Another shaded square
(4) A blue solid square
(5) A shaded vertical bow tie with horizontal stripes
(6) A blue solid horizontal bow tie
(7) A shaded triangle with horizontal stripes

▼ pg_shade_z

Test Types: INDEX, SM

Description: Draws a few different polygons with an ambient light on and checks that they're shaded correctly. The Z-buffer is on.

Attributes Tested: `XGL_3D_CTX_HLHSR_MODE`
`XGL_3D_CTX_LIGHTS`
`XGL_3D_CTX_LIGHT_NUM`
`XGL_3D_CTX_LIGHT_SWITCHES`
`XGL_3D_CTX_SURF_FRONT_AMBIENT`
`XGL_3D_CTX_SURF_FRONT_ILLUMINATION`
`XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT`
`XGL_LIGHT_COLOR`
`XGL_LIGHT_TYPE`

Operators Tested: `xgl_object_get`
`xgl_object_set`
`xgl_polygon`

Output: Renders the following seven polygons one after the other:
(1) A shaded square with vertical stripes
(2) A shaded square with horizontal stripes
(3) Another shaded square
(4) A blue solid square
(5) A shaded vertical bow tie with horizontal stripes
(6) A blue solid horizontal bow tie
(7) A shaded triangle with horizontal stripes

pg_shade_rgb

Test Types:	RGB, SM
Description:	Draws a few different RGB polygons with an ambient light on and checks that they're shaded correctly
Attributes Tested:	XGL_3D_CTX_LIGHTS XGL_3D_CTX_LIGHT_NUM XGL_3D_CTX_LIGHT_SWITCHES XGL_3D_CTX_SURF_FRONT_AMBIENT XGL_3D_CTX_SURF_FRONT_ILLUMINATION XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT XGL_LIGHT_COLOR XGL_LIGHT_TYPE
Operators Tested:	xgl_object_get xgl_object_set xgl_polygon
Output:	The following six polygons are rendered one after the other: (1) A shaded square with vertical stripes (2) A shaded square with horizontal stripes (3) A blue solid square (4) A shaded vertical bow tie with horizontal stripes (5) A blue solid horizontal bow tie (6) A shaded triangle with horizontal stripes

pg_shade_z_rgb

Test Types:	RGB, SM
Description:	Draws a few different RGB polygons with an ambient light on and checks that they're shaded correctly. The Z-buffer is on.
Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_3D_CTX_LIGHTS XGL_3D_CTX_LIGHT_NUM XGL_3D_CTX_LIGHT_SWITCHES XGL_3D_CTX_SURF_FRONT_AMBIENT XGL_3D_CTX_SURF_FRONT_ILLUMINATION XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT XGL_LIGHT_COLOR XGL_LIGHT_TYPE

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_polygon`

Output:	Renders the following six polygons one after the other: (1) A shaded square with vertical stripes (2) A shaded square with horizontal stripes (3) A blue solid square (4) A shaded vertical bow tie with horizontal stripes (5) A blue solid horizontal bow tie (6) A shaded triangle with horizontal stripes

▼ pg_shade_hlhsr

Test Types:	INDEX, SM
Description:	Tests the shaded polygon's hidden surface removal: draws two shaded polygons at the same position but different depth and checks that only the front one is drawn; tries different depth combinations

Attributes Tested:	`XGL_3D_CTX_HLHSR_MODE`
	`XGL_3D_CTX_SURF_FRONT_AMBIENT`
	`XGL_3D_CTX_SURF_FRONT_ILLUMINATION`
	`XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT`
	`XGL_CTX_NEW_FRAME_ACTION`
	`(XGL_CTX_NEW_FRAME_CLEAR |`
	`XGL_CTX_NEW_FRAME_HLHSR_ACTION)`

Operators Tested:	`xgl_object_set`
	`xgl_polygon`
	`xgl_context_post`
	`xgl_context_new_frame`

Output:	Draws four shaded polygons (a square, a vertical bow tie, a horizontal bow tie, and a triangle) three times, one after the other

▼ pg_shade_hlhsr2

Test Types:	RGB, SM
Description:	Tests shaded RGB polygon's hidden surface removal: draws two shaded RGB polygons at the same position but different depth and checks that only the front one is drawn; tries different depth combinations

Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_3D_CTX_SURF_FRONT_AMBIENT XGL_3D_CTX_SURF_FRONT_ILLUMINATION XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT XGL_CTX_NEW_FRAME_ACTION (XGL_CTX_NEW_FRAME_CLEAR | XGL_CTX_NEW_FRAME_HLHSR_ACTION)
Operators Tested:	xgl_object_set xgl_polygon xgl_context_post xgl_context_new_frame
Output:	Draws three shaded polygons (a square, a horizontal bow tie, and a triangle) three times, one after the other

▼ gc_pg_cull

Test Types:	INDEX, SM
Description:	Tests the three face-culling modes for gcached polygon
Attributes Tested:	XGL_CTX_SURF_FRONT_FILL_STYLE XGL_3D_CTX_SURF_BACK_FILL_STYLE XGL_3D_CTX_SURF_FRONT_ILLUMINATION XGL_3D_CTX_SURF_BACK_ILLUMINATION XGL_3D_CTX_SURF_FACE_DISTINGUISH XGL_3D_CTX_SURF_FACE_CULL
Operators Tested:	xgl_object_create xgl_object_set xgl_gcache_polygon xgl_context_display_gcache
Output:	Draws two polygons, red for front facing and green for back facing. These alternate when culling is turned off and on.

▼ gc_pg_decomp

Test Types:	INDEX, SM
Description:	Tests that xgl_gcache_polygon() correctly applies XGL_CTX_SURF_INTERIOR_RULE when drawing gcached polygons. This exercises the gcached_polygon decomposing attribute.

Attributes Tested:	XGL_GCACHE_DO_POLYGON_DECOMP XGL_GCACHE_POLYGON_TYPE (XGL_POLYGON_COMPLEX) XGL_CTX_SURF_INTERIOR_RULE
Operators Tested:	xgl_object_create xgl_object_set xgl_gcache_polygon xgl_context_display_gcache
Output:	Draws various nonconvex (they all have holes) polygons, all with surface color of default white

gc_pg_decomp1

Test Types:	INDEX, SM
Description:	Tests that xgl_gcache_polygon() correctly decompose odd shaped polygons.
Attributes Tested:	XGL_GCACHE_DO_POLYGON_DECOMP
Operators Tested:	xgl_object_create xgl_object_set xgl_gcache_polygon xgl_context_display_gcache
Output:	Draws various nonconvex (they all have holes) polygons, all with blue surface

gc_pg_decomp_pttypes

Test Types:	INDEX, SM
Description:	Tests that gcached polygons can be rendered using different point types: draws a vertical bow tie using two different point types and checks that they're all drawn correctly; draws a 400-vertex polygon (circle) with XGL_GCACHE_DO_POLYGON_DECOMP set and checks that it's drawn right. Also checks XGL_GCACHE_DISPLAY_PRIM_TYPE and XGL_GCACHE_ORIG_PRIM_TYPE.
Attributes Tested:	XGL_GCACHE_DISPLAY_PRIM_TYPE XGL_GCACHE_DO_POLYGON_DECOMP XGL_GCACHE_IS_EMPTY

	`XGL_GCACHE_ORIG_PRIM_TYPE`
	`XGL_GCACHE_POLYGON_TYPE (XGL_POLYGON_NSI)`
	`XGL_GCACHE_SHOW_DECOMP_EDGES`
Operators Tested:	`xgl_object_create`
	`xgl_object_set`
	`xgl_gcache_polygon`
	`xgl_context_display_gcache`
	`xgl_object_destroy`
Output:	Draws a white vertical bow tie twice at the same position. Finally, draws a white circle.

▼ gc_pg_edge4

Test Types:	RGB, CM
Description:	3D polygon edge flags test for gcached polygons; cycles through various combinations of edge flag settings using point flag types
Attributes Tested:	XGL_CTX_EDGE_COLOR XGL_GCACHE_USE_APPL_GEOM XGL_GCACHE_POLYGON_TYPE (XGL_POLYGON_NSI)
Operators Tested:	xgl_context_display_gcache xgl_gcache_polygon
Output:	Draws image with sixteen polygons with various edges illuminated; surface color is green, edges are purple

▼ gc_pg_face

Test Types:	INDEX, SM
Description:	Tests that gcached polygons are rendered correctly when XGL_3D_CTX_SURF_FACE_DISTINGUISH is TRUE/FALSE and XGL_3D_CTX_SURF_NORMAL_FLIP is TRUE/FALSE
Attributes Tested:	XGL_CTX_SURF_FRONT_FILL_STYLE (XGL_SURF_FILL_SOLID) XGL_3D_CTX_SURF_BACK_FILL_STYLE (XGL_SURF_FILL_HOLLOW) XGL_3D_CTX_SURF_BACK_COLOR XGL_3D_CTX_SURF_NORMAL_FLIP XGL_3D_CTX_SURF_FACE_DISTINGUISH XGL_GCACHE_POLYGON_TYPE (XGL_POLYGON_NSI)
Operators Tested:	xgl_object_create xgl_object_set xgl_gcache_polygon xgl_context_display_gcache xgl_object_destroy
Output:	Draws three convex polygons, red solids and/or green hollow, both front and back facing.

▼ gc_pg_face2

Test Types: RGB, SM

Description: Tests that gcached RGB polygons are rendered correctly when XGL_3D_CTX_SURF_FACE_DISTINGUISH is TRUE/FALSE and XGL_3D_CTX_SURF_NORMAL_FLIP is TRUE/FALSE

Attributes Tested:
XGL_CTX_SURF_FRONT_FILL_STYLE (XGL_SURF_FILL_SOLID)
XGL_3D_CTX_SURF_BACK_FILL_STYLE (XGL_SURF_FILL_HOLLOW)
XGL_3D_CTX_SURF_BACK_COLOR
XGL_3D_CTX_SURF_NORMAL_FLIP
XGL_3D_CTX_SURF_FACE_DISTINGUISH
XGL_GCACHE_POLYGON_TYPE (XGL_POLYGON_NSI)

Operators Tested:
xgl_object_create
xgl_object_set
xgl_gcache_polygon
xgl_context_display_gcache
xgl_object_destroy

Output: Draws three convex polygons, red solids and/or green hollow, both front and back facing.

▼ gc_pg_fill1

Test Types: INDEX, SM

Description: Tests various gcached polygon fill styles. Uses an array of gcaches to store five point lists; since some are complex polygons, XGL_POLYGON_COMPLEX is set to true.

Attributes Tested:
XGL_3D_CTX_HLHSR_MODE
XGL_CTX_SURF_EDGE_FLAG
XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_GCACHE_IS_EMPTY
XGL_GCACHE_POLYGON_TYPE (XGL_POLYGON_COMPLEX)

Operators Tested:
xgl_object_create
xgl_object_set
xgl_gcache_polygon
xgl_context_display_gcache
xgl_object_destroy

Output: Draws various polygons, each on a rendering cycle of yellow solid, then yellow hollow, green edges, then red solid, red hollow, green edges, and so on

▼ gc_pg_fill3

Test Types: RGB, SM

Description: Tests various RGB gcached polygon fill styles; polygons are either facet normal or facet color type

Attributes Tested:
XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_CTX_SURF_FRONT_COLOR
XGL_3D_CTX_HLHSR_MODE
XGL_GCACHE_POLYGON_TYPE
(XGL_POLYGON_COMPLEX)
XGL_CTX_SURF_EDGE_FLAG
XGL_CTX_EDGE_WIDTH_SCALE_FACTOR
XGL_CTX_EDGE_COLOR

Operators Tested:
xgl_object_create
xgl_object_set
xgl_gcache_polygon
xgl_context_display_gcache
xgl_object_destroy

Output: Draws convex polygons and then overlapping: first in orange solid, then orange hollow, then green edges

▼ gc_pg_fill9

Test Types: RGB, SM

Description: Tests various RGB gcached polygon back fill styles. The polygons are of type facet color, facet normal, or facet color normal.

Attributes Tested:
XGL_CTX_EDGE_WIDTH_SCALE_FACTOR
XGL_3D_CTX_SURF_BACK_FILL_STYLE
XGL_3D_CTX_HLHSR_MODE
XGL_GCACHE_POLYGON_TYPE
(XGL_POLYGON_COMPLEX)

Operators Tested:
xgl_object_create
xgl_object_set
xgl_gcache_polygon
xgl_context_display_gcache

Output: Draws various polygons rendered first magenta solid then hollow, orange solid then hollow, edge color green, and so on

▼ gc_pg_intrule

Test Types: INDEX, SM

Description: Draws various gcached polygons with the XGL_EVEN_ODD interior rule using xgl_gcache_polygon(); checks their correctness

Attributes Tested:
XGL_3D_CTX_HLHSR_MODE
XGL_CTX_SURF_INTERIOR_RULE (XGL_EVEN_ODD)
XGL_GCACHE_POLYGON_TYPE
(XGL_POLYGON_COMPLEX)

Operators Tested:
xgl_object_create
xgl_object_set
xgl_gcache_polygon
xgl_context_display_gcache
xgl_object_destroy

Output: Draws each of the following five general polygons twice (first time with Z-buffer off, second time with Z-buffer on):
(1) A self-intersecting polygon
(2) Two totally overlapping squares
(3) Two totally overlapping squares that look the same as number 2
(4) Two partially overlapping polygons
(5) A self-intersecting polygon that looks the same as number 1

▼ gc_pg_intrule2

Test Types: RGB, SM

Description: Draws various gcached RGB polygons with the XGL_EVEN_ODD interior rule using xgl_gcache_polygon(); checks their correctness

Attributes Tested:
XGL_3D_CTX_HLHSR_MODE
XGL_CTX_SURF_INTERIOR_RULE (XGL_EVEN_ODD)
XGL_GCACHE_POLYGON_TYPE
(XGL_POLYGON_COMPLEX)

Operators Tested:	`xgl_object_create`
	`xgl_object_set`
	`xgl_gcache_polygon`
	`xgl_context_display_gcache`
	`xgl_object_destroy`

Output:	Draws each of the following five general polygons twice (first time with Z-buffer off, second time with Z-buffer on): (1) A self-intersecting polygon (2) Two totally overlapping squares (3) Two totally overlapping squares that look the same as number 2 (4) Two partially overlapping polygons (5) A self-intersecting polygon which looks the same as number 1

gc_pg_pttypes

Test Types:	INDEX, SM
Description:	Tests that gcached polygons can be rendered using different point types: draws a vertical bow tie using two different point types and checks that they're all drawn correctly; draws a 400-vertex polygon (circle) and checks that it's drawn right.
Attributes Tested:	`XGL_GCACHE_IS_EMPTY` `XGL_GCACHE_POLYGON_TYPE (XGL_POLYGON_NSI)`

Operators Tested:	`xgl_object_create`
	`xgl_object_set`
	`xgl_gcache_polygon`
	`xgl_context_display_gcache`
	`xgl_object_destroy`

Output:	Draws a white vertical bow tie twice at the same position. Finally, draws a white circle.

gc_pg_pttypes2

Test Types:	RGB, SM
Description:	Tests that gcached RGB polygons can be rendered using different point types: draws a vertical bow tie using two different point types and checks that they're all drawn correctly; draws a 400-vertex polygon (circle) and checks that it's drawn right.
Attributes Tested:	`XGL_GCACHE_IS_EMPTY` `XGL_GCACHE_POLYGON_TYPE (XGL_POLYGON_NSI)`

Operators Tested:	`xgl_object_create`
	`xgl_object_set`
	`xgl_gcache_polygon`
	`xgl_context_display_gcache`

Output:	Draws an orange vertical bow tie twice at the same position. Finally, draws an orange circle.

gc_pg_decomp_facet

Test Types:	INDEX, SM
Description:	Tests for gcaching complex (nonconvex) polygons with various facet types. The four different kinds of facets are stored with the polygon primitives in the gcache. The default surface color is blue, so red surfaces should be rendered for the facets with color information only. Every time the caches are reused, the program first explicitly sets `XGL_GCACHE_IS_EMPTY` to TRUE. Also, a check is done to see if decomposition was actually done in the cache.

Attributes Tested:	`XGL_GCACHE_DO_POLYGON_DECOMP`
	`XGL_GCACHE_DISPLAY_PRIM_TYPE`
	`XGL_GCACHE_ORIG_PRIM_TYPE`
	`XGL_GCACHE_POLYGON_TYPE`
	`(XGL_POLYGON_COMPLEX)`

Operators Tested:	`xgl_object_create`
	`xgl_object_set`
	`xgl_gcache_polygon`
	`xgl_context_display_gcache`

Output:	Nonconvex polygons, in alternating colors of blue then red

gc_pg_decomp_complex

Test Types: INDEX, SM

Description: Tests for gcached complex multiboundary polygon. Tests cache for XGL_GCACHE_USE_APPL_GEOM, as well as one polygon/cache with multiple (eight) boundaries and greater than 100 points. Index color mode used.

Attributes Tested:
XGL_GCACHE_DO_POLYGON_DECOMP
XGL_GCACHE_POLYGON_TYPE
(XGL_POLYGON_COMPLEX)
XGL_GCACHE_IS_EMPTY
XGL_GCACHE_USE_APPL_GEOM
XGL_GCACHE_DISPLAY_PRIM_TYPE
XGL_GCACHE_ORIG_PRIM_TYPE

Operators Tested:
xgl_object_create
xgl_object_set
xgl_gcache_polygon
xgl_context_display_gcache
xgl_object_destroy

Output: Draws two columns of nonconvex (horizontal zigzagged) red polygons

gc_pg_show_decomp

Test Types: CM, INDEX

Description: Shows the decomposition of a complex gcached polygon with multiple boundaries, by setting XGL_GCACHE_SHOW_DECOMP_EDGES to true. This also exercises XGL_GCACHE_USE_APPL_GEOM for mode of storage, then renders each image and uses comparison methodology for verifying correctness. Index color mode used.

Attributes Tested:
XGL_GCACHE_SHOW_DECOMP_EDGES
XGL_GCACHE_USE_APPL_GEOM

Operators Tested:
xgl_object_create
xgl_object_set
xgl_gcache_polygon
xgl_context_display_gcache
xgl_object_destroy

Output:	Nonconvex polygons with blue surfaces and red edges. On the gx, you can see the edges delineating where the tessellation took place.

▼ polygon

Test Types:	SM, INDEX
Description:	Uses various colors and hatch styles to fill four-sided polygons.
Attributes Tested:	XGL_CTX_SURF_FRONT_COLOR XGL_CTX_SURF_FRONT_FILL_STYLE XGL_CTX_SURF_FRONT_FPAT
Operators Tested:	xgl_object_set xgl_polygon
Output:	Draws solid squares of various colors and hatch-styled squares

▼ pg_threshold

Test Types:	SM, INDEX
Description:	Tests convex and nonconvex polygons and XGL_CTX_THRESHOLD. It also sets the threshold and renders with null *bbox* (which should render all the shapes).
Attributes Tested:	XGL_CTX_THRESHOLD
Operators Tested:	xgl_object_set xgl_polygon
Output:	Renders twelve polygons of different sizes and bounding boxes with threshold values set from 0 to 100 in increments of 20

Quadrilateral Mesh Test Descriptions 23

This chapter describes the Quadrilateral Mesh test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ qm_col_norm

Test Types:	INDEX, SM
Description:	Same single quadmesh rendered with all the 3D possible point types, all the possible fill styles, a single ambient light source, and all the possible illumination types. The outer loop sets the interior styles, the next loop sets the illumination types, the next loop sets the point type, and the final loop sets the facet data.
Attributes Tested:	See Table 23-1, Column A at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_quadrilateral_mesh`

Output:	When the point type includes color information and the illumination is per vertex, expects a rainbow like a candy cane for the edge for hollow or the complete interior for solid where the candy cane / rainbow is red, green, yellow, dark blue, purple, and light blue. When illumination is per vertex and the point type does not involve color but the facet type does, expects the facet color, which is green for the edge in the hollow case and the complete interior for the solid case. Otherwise when illumination is per vertex, the point type does not involve color, and the facet type is none or involves normal information only, expects the interior color set, which is red for the edge in the hollow case and the entire interior for the solid case. For the empty interior fill style, expects the background color.

▼ qm_col_norm_rgb

Test Types:	RGB, SM
Description:	Same single quadmesh rendered with all the 3D possible point types, all the possible fill styles, a single ambient light source, and all the possible illumination types. The outer loop sets the interior styles, the next loop sets the illumination types, the next loop sets the point type, and the final loop sets the facet data.
Attributes Tested:	See Table 23-1, Column A at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_quadrilateral_mesh`

Output:	When the point type includes color information and the illumination is per vertex, expects a shaded coloring that blends into green for the edge for hollow or the complete interior for solid. When illumination is per vertex and the point type does not involve color but the facet type does, expects the facet color, which is blue for the edge in the hollow case and the complete interior for the solid case. Otherwise when illumination is per vertex, the point type does not involve color, and the facet type is none or

involves normal information only, expects the interior color set, which is red for the edge in the hollow case and the entire interior for the solid case. For the empty interior fill style, expects the background color.

▼ qm_cull_rgb

Test Types:	RGB, SM
Description:	Draws RGB quad mesh composed of four facets; two on the diagonal from left to right are back facing, while the two on the opposite diagonal are front facing, with different face-culling modes (none, front, back).
Attributes Tested:	`XGL_CTX_BACKGROUND_COLOR` and Table 23-2, Column B at the end of this chapter

Operators Tested:	`xgl_object_set`
	 `xgl_object_get`
	 `xgl_quadrilateral_mesh`

Output:	When the quadmesh is front facing, that is, the z component of its normal is less than 0.0, and the cull mode is FRONT, the expected color is the background, which is black. When the quadmesh is front facing and the cull mode is anything except FRONT, the expected color is the front surface color, which is purple. When the quadmesh is not front facing, that is, the z component of its normal is greater than or equal to 0.0, and the cull mode is FRONT, the expected color is the back surface color, which is yellow. Any other cull mode for a back-facing quadmesh is expected to be the background color, which is black. The normal appearance of the quadmesh with culling set to NONE is a four-faceted quad with the left to right facets on the diagonal yellow and the opposite diagonal facets purple.

▼ qm_hlhsr2_rgb

Test Types:	RGB, SM
Description:	Draws a single quadmesh that appears in the plane defined by the vector (100,100,299, 100,300,0); checks that the portion which resides below the HLHSR_DATA point has been Z-buffer clipped out as this data point is incremented by 10 from 0 to 290
Attributes Tested:	XGL_3D_CTX_HLHSR_DATA XGL_3D_CTX_LINE_COLOR_INTERP XGL_CTX_BACKGROUND_COLOR XGL_SURF_FILL_EMPTY XGL_SURF_FILL_HOLLOW and Table 23-2, Column A at the end of this chapter
Operators Tested:	xgl_object_set xgl_object_get xgl_quadrilateral_mesh xgl_context_new_frame
Output:	The diagonal in the direction from the left to the right lies through the vertex points where the z value is greater than the HLHSR_DATA point. Because of this, as this point is increased, the red triangles on the vertex points with a 0 z component are extended toward each other.

▼ qm_hlhsr_rgb

Test Types:	RGB, SM
Description:	Renders two sets of the same quadmesh with different z values and expects to view the quadmesh with the smaller z depth. The first set of quadmeshes are set up with a front surface color of yellow, while their exact counterparts have a front surface color of light blue. The depths for the same quadmeshes with different colors are such that the first quadmeshes rendered are expected, the second quadmeshes rendered are expected and again the second quadmeshes rendered are expected. Finally, renders a quadmesh where two of the vertex are overlapping.

Attributes Tested:	XGL_3D_CTX_SURF_FRONT_ILLUMINATION XGL_ILLUM_NONE and Table 23-2, Column A at the end of this chapter
Operators Tested:	xgl_object_set xgl_quadrilateral_mesh xgl_context_new_frame
Output:	Draws two sets of quadmesh. The top one is composed of a single square, and the bottom one is composed of two squares. Their color alternates between yellow and light blue. The final frame is a single light blue quadmesh.

▼ qm_simple

Test Types:	INDEX, SM
Description:	Sets up a single ambient light source and tries SOLID, HOLLOW, and EMPTY interior styles. Tries these combinations with point types F3D accompanied by no illumination, COLOR_F3D accompanied by illumination per vertex, and F3D accompanied by illumination per facet and an edge color of blue except for EMPTY and HOLLOW where the illumination per facet is skipped.
Attributes Tested:	XGL_CTX_EDGE_COLOR XGL_CTX_SURF_EDGE_FLAG and Table 23-1, Column A at the end of this chapter
Operators Tested:	xgl_object_set xgl_quadrilateral_mesh xgl_object_get
Output:	Expects background color for EMPTY interior. Expects a red interior surface color for SOLID and a red edge color for HOLLOW and no illumination. Expects a shaded edged or interior quadmesh with rainbow colors, red, green, yellow, dark blue, purple, and light blue for illumination per vertex for HOLLOW and SOLID respectively. Expects a light blue SOLID quadmesh for illumination per facet.

qm_simple_rgb

Test Types:	RGB, SM
Description:	Sets up a single ambient light source and tries `SOLID`, `HOLLOW`, and `EMPTY` interior styles. Tries these combinations with point types `F3D` accompanied by no illumination, `COLOR_F3D` accompanied by illumination per vertex.
Attributes Tested:	See Table 23-1, Column A at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_quadrilateral_mesh`
	`xgl_object_get`

Output:	Expects background color for `EMPTY` interior. Expects a red interior surface color for `SOLID` and a red edge color for `HOLLOW` and no illumination. Expects a shaded edged or interior quadmesh with shading from red to green for illumination per vertex for `HOLLOW` and `SOLID` respectively.

qm_solid_interp

Test Types:	INDEX, SM
Description:	Tries all different point and facet types for `SOLID` interior color-interpolated quadmesh with four individual facets
Attributes Tested:	See Table 23-3, Column A at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_quadrilateral_mesh`
	`xgl_object_get`
	`xgl_context_new_frame`

Output:	For point types with color, `COLOR_F3D`, `COLOR_NORMAL_F3D`, `COLOR_FLAG_F3D`, and `COLOR_NORMAL_FLAG_F3D`, expects a shaded quadmesh with rainbow shading in candy cane fashion in red, green, yellow, dark blue, purple, light blue, and gray. For facet types without color and point types without color information, expects a solid quadmesh with the front surface color, which is red. For facet types with color information and vertex types without color information, expects the facet color, which is 1(red), 2(green), 3(yellow) and 4(blue) respectively.

▼ qm_solid_interp_rgb

Test Types:	RGB, SM
Description:	Tries all different point and facet types for `SOLID` interior color-interpolated quadmesh with four individual facets
Attributes Tested:	See Table 23-3, Column A at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_quadrilateral_mesh`
	`xgl_object_get`
	`xgl_context_new_frame`

Output:	For point types with color, `COLOR_F3D`, `COLOR_NORMAL_F3D`, `COLOR_FLAG_F3D`, and `COLOR_NORMAL_FLAG_F3D`, expects a shaded quadmesh with rainbow shading that blends red into yellow, or purple and yellow into green then light blue. For facet types without color and point types without color information, expects a solid quadmesh with the front surface color, which is red. For facet types with color information and vertex types without color information, expects the facet color, which is 1(red), 2(green), 3(yellow), and 4(blue) respectively.

▼ qm_solid_no_illum

Test Types:	INDEX, SM
Description:	Tries all different point types and facet types for solid interior with no lighting and no color interpolation for a quadmesh composite of four facets

Attributes Tested:	`XGL_CTX_BACKGROUND_COLOR`
	`XGL_CTX_SURF_FRONT_COLOR`
	`XGL_DRAW_EDGE`
	`XGL_DRAW_PREV_EDGE`

Operators Tested:	`xgl_object_set`
	`xgl_quadrilateral_mesh`
	`xgl_object_get`

Output:	Expects the surface color for facet types without color information, which would be the front surface color of 5, purple. For facet types with color information, the color is

dependent on the facet color, which for the ordered quadmesh is 1(red), 2(green), 3(yellow), and 4(blue) respectively.

▼ qm_solid_no_illum_rgb

Test Types:	RGB, SM
Description:	Tries all different point types and facet types for solid interior with no lighting and no color interpolation for a quadmesh composite of four facets
Attributes Tested:	XGL_CTX_BACKGROUND_COLOR XGL_CTX_SURF_FRONT_COLOR XGL_DRAW_EDGE XGL_DRAW_PREV_EDGE
Operators Tested:	xgl_object_set xgl_quadrilateral_mesh xgl_object_get
Output:	Expects the surface color for facet types without color information, which would be the front surface color of purple. For facet types with color information, the color is dependent on the facet color, which for the ordered quadmesh is red, green, yellow, and blue respectively.

▼ qm_solid_per_facet

Test Types:	INDEX, RGB
Description:	Tries all point and facet types for solid quadmeshes composed of four facets with illumination per facet and one light source which is ambient
Attributes Tested:	XGL_ILLUM_PER_FACET and Table 23-4, Column A at the end of this chapter
Operators Tested:	xgl_object_set xgl_quadrilateral_mesh xgl_object_get
Output:	Expects the front surface color, which is red for facet types that contain no color information. For facet types with color information, expects the facet color, which for the ordered quadmesh is 1(red), 2(green), 3(yellow), and 4(blue) respectively.

▼ qm_solid_per_facet_rgb

Test Types: RGB, SM

Description: Tries all point and facet types for solid quadmeshes composed of four facets with illumination per facet and one light source, which is ambient.

Attributes Tested: `XGL_ILLUM_PER_FACET` and Table 23-4, Column A at the end of this chapter

Operators Tested: `xgl_object_set` `xgl_quadrilateral_mesh` `xgl_object_get`

Output: Expects the front surface color which is red for facet types that contain no color information. For facet types with color information, expects the facet color, which for the ordered quadmesh is red, green, yellow, and blue respectively.

▼ qm_solid_per_vtx

Test Types: INDEX, RGB

Description: Tries all point and facet types for solid quadmeshes composed of four facets with illumination per vertex and one light source which is ambient

Attributes Tested: `XGL_ILLUM_PER_VERTEX` and Table 23-4, Column A at the end of this chapter

Operators Tested: `xgl_object_set` `xgl_quadrilateral_mesh` `xgl_object_get`

Output: When the point type involves color information, `XGL_PT_COLOR_F3D`, `XGL_PT_COLOR_NORMAL_F3D`, `XGL_PT_COLOR_FLAG_F3D`, and `XGL_PT_COLOR_NORMAL_FLAG_F3D`, expects a shaded quadmesh with rainbow colors painted in candy cane fashion composed of red, green, yellow, dark blue, purple, light blue, and gray. When both the facet type and the point type contain no color information, expects the front surface color, which is red. When just the facet type contains color information, expects the individual facet to be the facet color, which for the ordered quadmesh is red, green, yellow, and blue respectively.

qm_solid_per_vtx_rgb

Test Types:	RGB, SM
Description:	Tries all point and facet types for solid quadmeshes composed of four facets with illumination per vertex and one light source, which is ambient
Attributes Tested:	`XGL_ILLUM_PER_VERTEX` and Table 23-4, Column A at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_quadrilateral_mesh`
	`xgl_object_get`

Output:	When the point type involves color information, `XGL_PT_COLOR_F3D`, `XGL_PT_COLOR_NORMAL_F3D`, `XGL_PT_COLOR_FLAG_F3D`, and `XGL_PT_COLOR_NORMAL_FLAG_F3D`, expects a shaded quadmesh with rainbow colors that blend smoothly into one another composed of red, green, yellow, dark blue, purple, and light blue. When both the facet type and the point type contain no color information, expects the front surface color, which is red. When just the facet type contains color information, expects the individual facet to be the facet color, which for the ordered quadmesh is red, green, yellow, and blue respectively.

qm_xform_no_illum

Test Types:	INDEX, RGB
Description:	Tries all point and facet types for solid quadmeshes composed of four facets with no lighting and a nonidentity viewing transform
Attributes Tested:	See Table 23-3, Column B at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_object_create`
	`xgl_transform_write`

Output:	The transformed quadmesh should appear as a wide line in the upper-left corner of the window raster. The color of this primitive is dependent on the facet information. When the facet information contains color information,

expect the facet color, which from the visible portion of the ordered quadmesh contains only red and yellow. Otherwise expect the surface color, which is purple.

▼ qm_xform_no_illum_rgb

Test Types:	RGB, SM
Description:	Tries all point and facet types for solid quadmeshes composed of four facets with no lighting and a nonidentity viewing transform
Attributes Tested:	See Table 23-3, Column B at the end of this chapter.

Operators Tested:	`xgl_object_create`
	`xgl_object_set`
	`xgl_quadrilateral_mesh`
	`xgl_object_get`
	`xgl_transform_write`

Output:	The transformed quadmesh should appear as a wide line in the upper-left corner of the window raster. The color of this primitive is dependent on the facet information. When the facet information contains color information, expect the facet color, which from the visible portion of the ordered quadmesh contains only red and yellow. Otherwise expect the surface color, which is purple.

▼ qm_empty_interp

Test Types:	INDEX, RGB
Description:	Tries all point and facet types for empty colored interpolated quadmeshes composed of four facets

Attributes Tested:	`XGL_3D_CTX_SURF_FRONT_ILLUMINATION`
	`XGL_ILLUM_NONE_INTERP_COLOR`
	and Table 23-4, Column B at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_context_new_frame`
	`xgl_quadrilateral_mesh`

Output:	Expects an empty quadmesh composed of four facets with blue edges

qm_empty_interp_rgb

Test Types:	RGB, SM
Description:	Tries all point and facet types for empty colored interpolated quadmeshes composed of four facets

Attributes Tested:	`XGL_3D_CTX_SURF_FRONT_ILLUMINATION`
	`XGL_ILLUM_NONE_INTERP_COLOR`
	and Table 23-4, Column B at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_context_new_frame`
	`xgl_quadrilateral_mesh`

Output:	Expects an empty quadmesh composed of four facets with blue edges

qm_empty_no_illum

Test Types:	INDEX, RGB
Description:	Tries all point and facet types for empty quadmeshes composed of four facets without lighting
Attributes Tested:	See Table 23-4, Column B at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_context_new_frame`
	`xgl_quadrilateral_mesh`

Output:	Expects an empty quadmesh composed of four facets with blue edges

qm_empty_no_illum_rgb

Test Types:	RGB, SM
Description:	Tries all point and facet types for empty quadmeshes composed of four facets without lighting
Attributes Tested:	See Table 23-4, Column B at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_context_new_frame`
	`xgl_quadrilateral_mesh`

Output:	Expects an empty quadmesh composed of four facets with blue edges

qm_empty_per_facet

Test Types:	INDEX, RGB
Description:	Tries all point and facet types for empty quadmeshes composed of four facets with per-facet lighting with one ambient light source
Attributes Tested:	`XGL_ILLUM_PER_FACET` and Table 23-1, Column B at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_context_new_frame`
	`xgl_quadrilateral_mesh`

Output:	Expects an empty quadmesh composed of four facets with blue edges

qm_empty_per_facet_rgb

Test Types:	RGB, SM
Description:	Tries all point and facet types for empty quadmeshes composed of four facets with per-facet lighting with one ambient light source

Attributes Tested:	`XGL_ILLUM_PER_FACET`
	`XGL_PT_NORMAL_F3D`
	`XGL_PT_NORMAL_FLAG_F3D`
	and Table 23-1, Column B at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_context_new_frame`
	`xgl_quadrilateral_mesh`

Output:	Expects an empty quadmesh composed of four facets with blue edges

qm_empty_per_vtx

Test Types:	INDEX, RGB
Description:	Tries all point and facet types for empty quadmeshes composed of four facets with per-vertex lighting with one ambient light source
Attributes Tested:	`XGL_ILLUM_PER_VERTEX` and Table 23-1, Column B at the end of this chapter

Operators Tested:	xgl_object_set xgl_object_get xgl_context_new_frame xgl_quadrilateral_mesh
Output:	Expects an empty quadmesh composed of four facets with blue edges

▼ qm_empty_per_vtx_rgb

Test Types:	RGB, SM
Description:	Tries all point and facet types for empty quadmeshes composed of four facets with per-vertex lighting with one ambient light source
Attributes Tested:	XGL_ILLUM_PER_VERTEX and Table 23-1, Column B at the end of this chapter
Operators Tested:	xgl_object_set xgl_object_get xgl_context_new_frame xgl_quadrilateral_mesh
Output:	Expects an empty quadmesh composed of four facets with blue edges

▼ qm_hollow_interp

Test Types:	INDEX, RGB
Description:	Tries all point and facet types for hollow interpolated quadmeshes composed of four facets
Attributes Tested:	XGL_CTX_SURF_FRONT_FILL_STYLE XGL_SURF_FILL_HOLLOW and Table 23-3, Column A at the end of this chapter
Operators Tested:	xgl_object_set xgl_object_get xgl_context_new_frame xgl_quadrilateral_mesh
Output:	Expects a rainbow of candy cane colors for the edges with red, green, yellow, dark blue, purple, light blue, and gray when the point type has color information. Expects the front surface color, which is red for the edge color when both the facet data and the point type have no color

information. Expects the facet color for the individual facets, which are red, green, blue, and yellow for the edge colors when the facet data has color information.

▼ qm_hollow_interp_rgb

Test Types:	RGB, SM
Description:	Tries all point and facet types for hollow interpolated quadmeshes composed of four facets

Attributes Tested:	`XGL_CTX_SURF_FRONT_FILL_STYLE`
	`XGL_SURF_FILL_HOLLOW`
	and Table 23-3, Column A at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_context_new_frame`
	`xgl_quadrilateral_mesh`

Output:	Expects a rainbow blending for the edges with red, yellow, purple, light blue, and gray when the point type has color information. Expects the front surface color, which is red for the edge color when both the facet data and the point type have no color information. Expects the facet color of for the individual facets which are red, green, blue, and yellow for the edge colors when the facet data has color information.

▼ qm_hollow_no_illum

Test Types:	INDEX, RGB
Description:	Tries all point and facet types for hollow unlighted quadmeshes composed of four facets
Attributes Tested:	See Table 23-2, Column C at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_quadrilateral_mesh`

Output:	Expects the front surface color, which is purple for the edge color when both the facet data and the point type have no color information. Expects the facet color for the individual facets which are red, green, blue, and yellow for the edge colors when the facet data has color information.

▼ qm_hollow_no_illum_rgb

Test Types:	RGB, SM
Description:	Tries all point and facet types for hollow unlighted quadmeshes composed of four facets
Attributes Tested:	See Table 23-2, Column C at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_quadrilateral_mesh`

Output:	Expects the front surface color, which is purple for the edge color when both the facet data and the point type have no color information. Expects the facet color for the individual facets, which are red, green, blue, and yellow for the edge colors when the facet data has color information.

▼ qm_hollow_per_facet

Test Types:	INDEX, RGB
Description:	Tries all point and facet types for hollow per-facet lighted quadmeshes composed of four facets
Attributes Tested:	See Table 23-2, Column C at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_quadrilateral_mesh`

Output:	Expects the front surface color, which is red for the edge color when both the facet data and the point type have no color information. Expects the facet color for the individual facets, which are red, green, blue, and yellow for the edge colors when the facet data has color information.

▼ qm_hollow_per_facet_rgb

Test Types:	RGB, SM
Description:	Tries all point and facet types for hollow per facet lighted quadmeshes composed of four facets
Attributes Tested:	See Table 23-2, Column C at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_quadrilateral_mesh`

Output:	Expects the front surface color, which is red for the edge color when both the facet data and the point type have no color information. Expects the facet color for the individual facets, which are red, green, blue, and yellow for the edge colors when the facet data has color information.

▼ qm_hollow_per_vtx

Test Types:	INDEX, RGB
Description:	Tries all point and facet types for hollow per vertex lighted quadmeshes composed of four facets

Attributes Tested:	`XGL_CTX_SURF_FRONT_FILL_STYLE`
	`XGL_ILLUM_PER_VERTEX`
	`XGL_SURF_FILL_HOLLOW`
	and Table 23-4, Column A at the end of this chapter

Operators Tested:	`xgl_object_set`
	`xgl_object_get`
	`xgl_quadrilateral_mesh`

Output:	Expects candy cane edges of red, green, yellow, dark blue, purple, and gray when the point type contains color information. Expects the front surface color for the edge color, which is red when both the facet type and the point type contain no color information. Expects the facet color for the edge color, which is red, green, blue, and yellow for the ordered facet list of quadmeshes when only the facet data contains color information.

▼ qm_hollow_per_vtx_rgb

Test Types:	RGB, SM
Description:	Tries all point and facet types for hollow per-vertex lighted quadmeshes composed of four facets

Attributes Tested:	`XGL_CTX_SURF_FRONT_FILL_STYLE`
	`XGL_ILLUM_PER_VERTEX`
	`XGL_SURF_FILL_HOLLOW`
	and Table 23-4, Column A at the end of this chapter

Operators Tested:	xgl_object_set xgl_object_get xgl_quadrilateral_mesh
Output:	Expects a rainbow of colors in the edges of red, yellow, light blue, purple, and gray when the point type contains color information. Expects the front surface color for the edge color, which is red when both the facet type and the point type contain no color information. Expects the facet color for the edge color, which is red, green, blue, and yellow for the ordered facet list of quadmeshes when only the facet data contains color information.

▼ qm_cull

Test Types:	INDEX, SM
Description:	Draws INDEX quadmesh composed of four facets; two on the diagonal from left to right are back facing, while the two on the opposite diagonal are front facing, with different face-culling modes (none, front, back)
Attributes Tested:	See Table 23-2, Column B at the end of this chapter.
Operators Tested:	xgl_object_set xgl_quadrilateral_mesh
Output:	When the quadmesh is front facing, that is, the z component of its normal is less than 0.0, and the cull mode is FRONT, the expected color is the background, which is black. When the quadmesh is front facing and the cull mode is anything except FRONT, the expected color is the front surface color which is red. When the quadmesh is not front facing, that is, the z component of its normal is greater than or equal to 0.0, and the cull mode is FRONT, the expected color is the back surface color, which is green. Any other cull mode for a back-facing quadmesh is expected to be the background color, which is black. The normal appearance of the quadmesh with culling set to NONE is a four-faceted quad with the left to right facets on the diagonal green and the opposite diagonal facets red.

qm_hlhsr

Test Types:	INDEX, SM
Description:	Renders two sets of the same quadmesh with different z values and expects to view the quadmesh with the smaller z depth. The first set of quadmeshes are set up with a front surface color of yellow, while their exact counterparts have a front surface color of light blue. The depths for the same quadmesh with different colors are such that the first quadmeshes rendered are expected, the second quadmeshes rendered are expected and again the second quadmeshes rendered are expected. Finally, renders a quadmesh where two of the vertex are overlapping.
Attributes Tested:	XGL_3D_CTX_SURF_FRONT_ILLUMINATION XGL_ILLUM_NONE and Table 23-2, Column A at the end of this chapter
Operators Tested:	xgl_object_set xgl_quadrilateral_mesh xgl_context_new_frame
Output:	Two sets of quadmesh. The top one is composed of a single square, and the bottom one is composed of two squares. Their color alternates between red and green. The final frame is a single red quadmesh.

qm_edge_rgb

Test Types:	RGB, SM
Description:	Tests for edged quadmeshes with backface interior style set to HOLLOW with point type XGL_PT_COLOR_FLAG_F3D.
Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_3D_CTX_SURF_FRONT_ILLUMINATION XGL_3D_CTX_HLHSR_DATA XGL_CTX_DEFERRAL_MODE XGL_CTX_SURF_EDGE_FLAG XGL_CTX_EDGE_COLOR
Operators Tested:	xgl_quadrilateral_mesh
Output:	A single red edged quadmesh

Table 23-1 Quadrilateral Mesh Attributes Tested - Set 1

Column A	Column B
XGL_3D_CTX_LIGHTS	XGL_3D_CTX_HLHSR_MODE
XGL_3D_CTX_LIGHT_NUM	XGL_3D_CTX_LIGHTS
XGL_3D_CTX_LIGHT_SWITCHES	XGL_3D_CTX_LIGHT_NUM
XGL_3D_CTX_LINE_COLOR_INTERP	XGL_3D_CTX_LIGHT_SWITCHES
XGL_3D_CTX_SURF_FRONT_AMBIENT	XGL_3D_CTX_SURF_FRONT_AMBIENT
XGL_3D_CTX_SURF_FRONT_ILLUMINATION	XGL_3D_CTX_SURF_FRONT_ILLUMINATION
XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT	XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT
XGL_CTX_BACKGROUND_COLOR	XGL_CTX_BACKGROUND_COLOR
XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_EDGE_COLOR
XGL_CTX_SURF_FRONT_FILL_STYLE	XGL_CTX_NEW_FRAME_ACTION
XGL_ILLUM_NONE	XGL_CTX_NEW_FRAME_CLEAR
XGL_ILLUM_PER_FACET	XGL_CTX_NEW_FRAME_HLHSR_ACTION
XGL_ILLUM_PER_VERTEX	XGL_CTX_SURF_EDGE_FLAG
XGL_LIGHT_AMBIENT	XGL_CTX_SURF_FRONT_COLOR
XGL_LIGHT_COLOR	XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_LIGHT_ENABLE_COMP_AMBIENT	XGL_DRAW_EDGE
XGL_LIGHT_TYPE	XGL_DRAW_PREV_EDGE
XGL_SURF_FILL_EMPTY	XGL_HLHSR_Z_BUFFER
XGL_SURF_FILL_HOLLOW	XGL_LIGHT_AMBIENT
XGL_SURF_FILL_SOLID	XGL_LIGHT_COLOR
	XGL_LIGHT_ENABLE_COMP_AMBIENT
	XGL_LIGHT_TYPE
	XGL_SURF_FILL_EMPTY

Table 23-2 Quadrilateral Mesh Attributes Tested - Set 2

Column A	**Column B**	**Column C**
XGL_3D_CTX_HLHSR_MODE	XGL_3D_CTX_SURF_BACK_COLOR	XGL_CTX_BACKGROUND_COLOR
XGL_CTX_NEW_FRAME_ACTION	XGL_3D_CTX_SURF_FACE_CULL	XGL_CTX_SURF_FRONT_COLOR
XGL_CTX_NEW_FRAME_CLEAR	XGL_3D_CTX_SURF_FACE_ DISTINGUISH	XGL_CTX_SURF_FRONT_FILL_ STYLE
XGL_CTX_NEW_FRAME_HLHSR_ ACTION	XGL_CTX_SURF_FRONT_COLOR	XGL_DRAW_EDGE
XGL_CTX_SURF_FRONT_COLOR	XGL_CULL_BACK	XGL_DRAW_PREV_EDGE
XGL_CTX_SURF_FRONT_FILL_ STYLE	XGL_CULL_FRONT	XGL_SURF_FILL_HOLLOW
XGL_HLHSR_Z_BUFFER	XGL_CULL_OFF	
XGL_SURF_FILL_SOLID		

Table 23-3 Quadrilateral Mesh Attributes Tested - Set 3

Column A	**Column B**
XGL_3D_CTX_HLHSR_MODE	XGL_CTX_BACKGROUND_COLOR
XGL_3D_CTX_SURF_FRONT_ILLUMINATION	XGL_CTX_SURF_FRONT_COLOR
XGL_CTX_BACKGROUND_COLOR	XGL_CTX_VIEW_TRANS
XGL_CTX_NEW_FRAME_ACTION	XGL_DRAW_EDGE
XGL_CTX_NEW_FRAME_CLEAR	XGL_DRAW_PREV_EDGE
XGL_CTX_NEW_FRAME_HLHSR_ACTION	XGL_TRANS
XGL_CTX_SURF_FRONT_COLOR	XGL_TRANS_DATA_TYPE
XGL_DRAW_EDGE	XGL_DATA_FLT
XGL_DRAW_PREV_EDGE	XGL_TRANS_DIMENSION
XGL_HLHSR_Z_BUFFER	XGL_TRANS_3D
XGL_ILLUM_NONE_INTERP_COLOR	

Table 23-4 Quadrilateral Mesh Attributes Tested - Set 4

Column A	**Column B**
XGL_3D_CTX_LIGHTS	XGL_3D_CTX_HLHSR_MODE
XGL_3D_CTX_LIGHT_NUM	XGL_CTX_BACKGROUND_COLOR
XGL_3D_CTX_LIGHT_SWITCHES	XGL_CTX_EDGE_COLOR
XGL_3D_CTX_SURF_FRONT_AMBIENT	XGL_CTX_NEW_FRAME_ACTION
XGL_3D_CTX_SURF_FRONT_ILLUMINATION	XGL_CTX_NEW_FRAME_CLEAR
XGL_CTX_BACKGROUND_COLOR	XGL_CTX_NEW_FRAME_HLHSR_ACTION
XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT	XGL_CTX_SURF_EDGE_FLAG
XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_SURF_FRONT_COLOR
XGL_DRAW_EDGE	XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_DRAW_PREV_EDGE	XGL_DRAW_EDGE
XGL_LIGHT_AMBIENT	XGL_DRAW_PREV_EDGE
XGL_LIGHT_COLOR	XGL_HLHSR_Z_BUFFER
XGL_LIGHT_ENABLE_COMP_AMBIENT	XGL_SURF_FILL_EMPTY
XGL_LIGHT_TYPE	

Raster Test Descriptions 24

This chapter describes the Raster test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ ras_attr1

Test Types:	INDEX, SM
Description:	Creates a window raster and verifies the values of various window raster attributes
Attributes Tested:	XGL_RAS_WIDTH XGL_WIN_RAS_BUFFERS_REQUESTED XGL_WIN_RAS_BUF_DISPLAY XGL_WIN_RAS_BUF_DRAW XGL_WIN_RAS_BUF_MIN_DELAY XGL_WIN_RAS_POSITION XGL_WIN_RAS_TYPE

Operators Tested:	xgl_object_create xgl_object_get
Output:	Nothing is displayed.

▼ ras_attr2

Test Types:	INDEX, SM
Description:	Creates a memory raster and verifies the values of various memory raster attributes
Attributes Tested:	XGL_MEM_RAS XGL_MEM_RAS_IMAGE_BUFFER_ADDR XGL_RAS_DEPTH XGL_RAS_HEIGHT XGL_RAS_WIDTH
Operators Tested:	xgl_object_create xgl_object_get
Output:	Nothing is displayed.

▼ ras_copy

Test Types:	INDEX, SM
Description:	Tests that xgl_context_copy_raster() can be used to copy a block of pixels from one raster to another raster
Attributes Tested:	XGL_CTX_MARKER XGL_CTX_MARKER_COLOR XGL_CTX_MARKER_SCALE_FACTOR XGL_RAS_DEPTH XGL_RAS_HEIGHT XGL_RAS_WIDTH
Operators Tested:	xgl_context_copy_raster xgl_multimarker
Output:	A green asterisk; first in upper-left corner, then in center

▼ ras_op

Test Types:	INDEX, SM
Description:	Sets the XGL_CTX_ROP attribute to various raster operation modes and uses overlapping primitives to verify the correct raster operations are done

Attributes Tested:	XGL_CTX_LINE_COLOR XGL_CTX_LINE_WIDTH_SCALE_FACTOR XGL_CTX_ROP XGL_CTX_SURF_FRONT_COLOR XGL_RAS_DEPTH
Operators Tested:	xgl_object_set xgl_context_new_frame xgl_polygon xgl_multipolyline
Output:	A red polygon obscured by a smaller, wider polygon which is ropped in the various modes

▼ ras_copy2

Test Types:	INDEX, SM
Description:	Tests the XGL_CTX_RASTER_FILL_STYLE, XGL_CTX_RASTER_FPAT_POSITION, XGL_CTX_RASTER_FPAT, and XGL_CTX_RASTER_STIPPLE_COLOR attributes.
Attributes Tested:	XGL_CTX_BACKGROUND_COLOR XGL_CTX_RASTER_FILL_STYLE XGL_CTX_RASTER_FPAT XGL_CTX_RASTER_FPAT_POSITION XGL_CTX_RASTER_STIPPLE_COLOR XGL_RAS_DEPTH
Operators Tested:	xgl_object_set xgl_context_copy_raster xgl_polygon
Output:	Two polygons: one large red stippled, one to the right of it an olive green smaller rectangle

▼ plane_mask

Test Types:	INDEX, SM
Description:	Tests the XGL_CTX_PLANE_MASK attribute
Attributes Tested:	XGL_CTX_PLANE_MASK XGL_CTX_SURF_FRONT_COLOR XGL_DEV_COLOR_MAP XGL_RAS_DEPTH

Operators Tested:	xgl_object_set xgl_polygon
Output:	A red quad polygon that changes to green (the background color)

▼ ras0

Test Types:	INDEX, SM
Description:	Tests device attributes: XGL_DEV_CONTEXTS, XGL_DEV_CONTEXTS_NUM, and XGL_DEV_MAXIMUM_COORDINATES. The device is attached to a 2D context.
Attributes Tested:	XGL_DEV_CONTEXTS XGL_DEV_CONTEXTS_NUM XGL_DEV_MAXIMUM_COORDINATES
Operators Tested:	xgl_object_set xgl_object_get
Output:	Nothing is displayed.

▼ ras1

Test Types:	SM
Description:	Tests the device attributes XGL_DEV_CONTEXTS, XGL_DEV_CONTEXTS_NUM, and XGL_DEV_MAXIMUM_COORDINATES of a memory raster. The raster is attached to a 3D context.
Attributes Tested:	XGL_DEV_CONTEXTS XGL_DEV_CONTEXTS_NUM XGL_DEV_MAXIMUM_COORDINATES XGL_RAS_HEIGHT XGL_RAS_WIDTH
Operators Tested:	xgl_object_set xgl_object_get
Output:	Nothing is displayed.

ras_attr3

Test Types:	RGB, SM
Description:	Creates an RGB window raster and verifies the values of various window raster attributes
Attributes Tested:	XGL_WIN_RAS_BUFFERS_ALLOCATED XGL_WIN_RAS_BUFFERS_REQUESTED XGL_WIN_RAS_BUF_DISPLAY XGL_WIN_RAS_BUF_DRAW XGL_WIN_RAS_BUF_MIN_DELAY XGL_WIN_RAS_POSITION
Operators Tested:	xgl_object_create xgl_object_set xgl_object_get
Output:	Nothing is displayed.

ras_attr4

Test Types:	RGB, SM
Description:	Creates an RGB memory raster and verifies the values of various memory raster attributes
Attributes Tested:	XGL_MEM_RAS_IMAGE_BUFFER_ADDR XGL_RAS_DEPTH XGL_RAS_HEIGHT XGL_RAS_WIDTH
Operators Tested:	xgl_object_create xgl_object_get
Output:	Nothing is displayed.

ras_copy3

Test Types:	RGB, SM
Description:	Tests that `xgl_context_copy_raster()` can be used to copy a block of pixels from one RGB raster to another raster
Attributes Tested:	XGL_CTX_MARKER XGL_CTX_MARKER_COLOR XGL_CTX_MARKER_SCALE_FACTOR XGL_CTX_STEXT_CHAR_HEIGHT XGL_CTX_STEXT_COLOR

	XGL_RAS_DEPTH
	XGL_RAS_HEIGHT
	XGL_RAS_WIDTH
Operators Tested:	xgl_context_copy_raster
	xgl_multimarker
	xgl_stroke_text
Output:	A marker (green asterisk) and text that reads "XGL" with the G and L upside down. The text is red.

▼ ras_copy4

Test Types:	RGB, SM
Description:	Tests the XGL_CTX_RASTER_FILL_STYLE, XGL_CTX_RASTER_FPAT_POSITION, XGL_CTX_RASTER_FPAT, and XGL_CTX_RASTER_STIPPLE_COLOR attributes using an RGB raster
Attributes Tested:	XGL_CTX_BACKGROUND_COLOR
	XGL_CTX_RASTER_FILL_STYLE
	XGL_CTX_RASTER_FPAT
	XGL_CTX_RASTER_FPAT_POSITION
	XGL_CTX_RASTER_STIPPLE_COLOR
	XGL_CTX_SURF_FRONT_COLOR
	XGL_RAS_DEPTH
	XGL_RAS_HEIGHT
	XGL_RAS_WIDTH
Operators Tested:	xgl_object_set
	xgl_context_copy_raster
	xgl_polygon
Output:	A larger red stipple-filled red quad with a smaller olive green polygon to its right

▼ ras3

Test Types:	INDEX, SM
Description:	Tests simple double-buffering (indexed mode, set buffer)

Attributes Tested:	XGL_WIN_RAS_BUFFERS_ALLOCATED XGL_WIN_RAS_BUFFERS_REQUESTED XGL_WIN_RAS_BUF_DISPLAY XGL_WIN_RAS_BUF_DRAW XGL_WIN_RAS_BUF_MIN_DELAY
Operators Tested:	xgl_object_set xgl_object_get xgl_context_new_frame xgl_context_set_pixel xgl_multipolyline xgl_inquire
Output:	A 100x100 window displays alternately a line and polygon. The foreground is white.

▼ ras4

Test Types:	INDEX, SM
Description:	Double-buffering: all combinations of draw and display buffers, switching using explicit buffer settings
Attributes Tested:	XGL_CTX_NEW_FRAME_ACTION (XGL_CTX_NEW_FRAME_CLEAR) XGL_DEV_COLOR_MAP XGL_WIN_RAS_BUFFERS_ALLOCATED XGL_WIN_RAS_BUFFERS_REQUESTED XGL_WIN_RAS_BUF_DISPLAY XGL_WIN_RAS_BUF_DRAW XGL_WIN_RAS_BUF_MIN_DELAY
Operators Tested:	xgl_object_set xgl_object_get xgl_context_new_frame xgl_polygon xgl_multiarc xgl_stroke_text xgl_multirectangle xgl_inquire
Output:	A quarter arc, a four-pointed object, and XGL text

ras5

Test Types:	INDEX, SM
Description:	Xlib rendering on XGL double-buffered windows
Attributes Tested:	XGL_CTX_NEW_FRAME_ACTION (XGL_CTX_NEW_FRAME_CLEAR | XGL_CTX_NEW_FRAME_SWITCH_BUFFER) XGL_WIN_RAS_BUFFERS_ALLOCATED XGL_WIN_RAS_BUFFERS_REQUESTED XGL_WIN_RAS_BUF_DISPLAY XGL_WIN_RAS_BUF_DRAW XGL_WIN_RAS_PIXEL_MAPPING
Operators Tested:	xgl_object_set xgl_object_get xgl_context_new_frame xgl_inquire
Output:	A white horizontal line

ras6

Test Types:	INDEX, SM
Description:	Tests the interleaving of XGL and Xlib rendering. This test uses Xlib to render lines on the same index raster XGL is rendering polygons. All rendering is done in white/black foreground/background for simplicity. This test exits if it determines the visual is not equal to PseudoColor (index). The Xlib primitive is checked via Xlib.
Attributes Tested:	XGL_DEV_COLOR_TYPE
Operators Tested:	xgl_context_new_frame xgl_context_post xgl_context_flush xgl_polygon
Output:	Repeated rendering of five horizontal lines, with five quad polygons to the right of the lines

ras_pix

Test Types:	INDEX, SM
Description:	Tests simple indexed `xgl_context_set_multi_pixel` as well as `xgl_context_set_pixel` rendering. Renders a diagonal line, then a rectangle.

Attributes Tested:	`XGL_CMAP_COLOR_TABLE`
	`XGL_CMAP_COLOR_TABLE_SIZE`
	`XGL_CTX_BACKGROUND_COLOR`
	`XGL_CTX_DEFERRAL_MODE (XGL_DEFER_ASAP)`
	`XGL_DEV_COLOR_MAP`
	`XGL_CTX_RENDER_BUFFER (XGL_RENDER_Z_BUFFER)`

Operators Tested:	`xgl_context_set_multi_pixel` `xgl_context_set_pixel`
Output:	A red diagonal line and a green polygon on a blue background

ras_pix_rgb

Test Types:	RGB, SM
Description:	Tests simple `xgl_context_set_multi_pixel` and `xgl_context_set_pixel` rendering with RGB color model against 2D/3D context; renders a line and a rectangle

Attributes Tested:	`XGL_CTX_BACKGROUND_COLOR`
	`XGL_CTX_DEFERRAL_MODE (XGL_DEFER_ASAP)`
	`XGL_CTX_RENDER_BUFFER (XGL_RENDER_Z_BUFFER)`

Operators Tested:	`xgl_context_set_multi_pixel`
Output:	A red diagonal line and a green polygon on a blue background, but only if you're on a RGB system; otherwise, nothing is displayed.

ras_pix_row

Test Types:	INDEX, SM
Description:	Tests `xgl_context_set_pixel_row` with different columns, rows, and lengths against 2D/3D context using an indexed color model.

Attributes Tested:	XGL_CMAP_COLOR_TABLE XGL_CMAP_COLOR_TABLE_SIZE XGL_CTX_BACKGROUND_COLOR XGL_DEV_COLOR_MAP XGL_RAS_HEIGHT XGL_RAS_WIDTH XGL_CTX_DEFERRAL_MODE (XGL_DEFER_ASAP) XGL_CTX_RENDER_BUFFER (XGL_RENDER_Z_BUFFER)
Operators Tested:	xgl_context_set_pixel_row
Output:	A dozen or so of red horizontal lines at varying lengths

▼ ras_pix_row_rgb

Test Types:	SM, RGB
Description:	Tests xgl_context_set_pixel_row with different columns, rows, and lengths against 2D/3D context using an RGB color model.
Attributes Tested:	XGL_CTX_BACKGROUND_COLOR XGL_RAS_HEIGHT XGL_RAS_WIDTH XGL_CTX_DEFERRAL_MODE (XGL_DEFER_ASAP) XGL_CTX_RENDER_BUFFER (XGL_RENDER_Z_BUFFER)
Operators Tested:	xgl_context_set_pixel_row
Output:	A dozen or so of red horizontal lines at varying lengths, but only if you're using RGB hardware; otherwise, nothing is displayed.

▼ image_tg

Test Types:	SM
Description:	Tests xgl_image operator with 3D context and HLHSR values set to Z-buffer then none. Images with 1-, 8-, and 32-bit are tested.
Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_CTX_DC_VIEWPORT XGL_CTX_VDC_MAP XGL_CTX_VDC_ORIENTATION XGL_CTX_VDC_WINDOW XGL_MEM_RAS_IMAGE_BUFFER_ADDR

	XGL_RAS_DEPTH
	XGL_RAS_HEIGHT
	XGL_RAS_WIDTH
Operators Tested:	xgl_object_set
	xgl_object_get
	xgl_image
Output:	Nine frames of two red rectangles

▼ cp_ras_multi_ctx

Test Types:	INDEX, CM
Description:	The main purpose of the test is to test multiple memory rasters and multiple contexts; stretch resources. Each memory raster has a different dimension, a different size/color line drawn into it. This test then copies these unique characteristics to different offsets within the same window raster.
Attributes Tested:	XGL_CMAP_COLOR_TABLE
	XGL_CMAP_COLOR_TABLE_SIZE
	XGL_DEV_COLOR_MAP
	XGL_RAS_DEPTH
	XGL_RAS_HEIGHT
	XGL_RAS_WIDTH
Operators Tested:	xgl_object_create
	xgl_context_set_pixel_row
	xgl_context_copy_raster
Output:	Various horizontal lines, all of different colors

▼ xgl_img_2d

Test Types:	INDEX, CM
Description:	Tests `xgl_image` in 2D, index color. This test renders polygons to four different-sized 8-bit memory rasters, then copies the memory rasters to the default depth window raster via `xgl_image` in a replrop (repeated) manner. The four memory rasters each have their own context. The final case tests window clipping of `xgl_image` where DC coordinates end; that is, only ~20 will fit, but 30 are rendered, some in coordinates outside of the window.

Attributes Tested:	XGL_CMAP_COLOR_TABLE XGL_CMAP_COLOR_TABLE_SIZE XGL_CTX_SURF_FRONT_COLOR XGL_DEV_COLOR_MAP XGL_RAS_DEPTH XGL_RAS_HEIGHT XGL_RAS_WIDTH
Operators Tested:	xgl_object_create xgl_polygon xgl_image
Output:	First row, 3 red stars; second row, 4 blue boomerangs; third row, 5 yellow quads; fourth row, 13 green triangles

▼ xgl_img_2d_32

Test Types:	RGB, CM
Description:	Tests xgl_image in 2D with RGB rasters. Draws polygons on different-sized 32-bit memory rasters, then copies the memory rasters to the default depth window raster via xgl_image in a *replrop* (repeated) manner. This was adapted from *xgl_img_2d*, but exercises RGB instead of INDEX, and 32-bit memory raster instead of 8-bit. The last test is window clipping of xgl_image where DC coordinates end; that is, only ~20 will fit, but 30 are rendered, some in coordinates outside of the window.
Attributes Tested:	XGL_CTX_SURF_FRONT_COLOR XGL_RAS_DEPTH XGL_RAS_HEIGHT XGL_RAS_WIDTH
Operators Tested:	xgl_object_create xgl_polygon xgl_image
Output:	First row, 3 red stars; second row, 4 blue boomerangs; third row, 5 yellow quads; fourth row, 13 green triangles

▼ xgl_img_3d_32

Test Types: RGB, CM

Description: Tests `xgl_image` in 3D with RGB rasters. Draws polygons on different-sized 32-bit memory rasters, then copies the memory rasters to the default depth window raster via `xgl_image` in a *replrop* (repeated) manner. This was adapted from *xgl_img_2d_32*, but exercises 3D instead of 2D, and also test `xgl_context_copy_buffer` for z.

Attributes Tested:
`XGL_CTX_RENDER_BUFFER`
`XGL_RENDER_Z_BUFFER`
`XGL_RAS_SOURCE_BUFFER`
`XGL_BUFFER_SEL_Z`

Operators Tested:
`xgl_object_create`
`xgl_context_copy_buffer`
`xgl_image`

Output: Four rows of polygonal primitives; first row is star in red, second row is boomerang in blue, third is box in yellow, fourth is triangle in green. Again, mem raster geom is 128x128, 100x100, 64x64, 32x32 respectively. Some of the xgl_image primitives should be obscured by an indigo rectangle. This rectangle is first rendered into a memroy raster and then its Z values are copied to the image raster using `xgl_context_copy_buffer`. Next the image itself is copied using copy_buffer.

▼ xgl_img_rect

Test Types: INDEX, CM

Description: Tests `xgl_image` in 2D, exercising clipping (that is, only copying regions) of memory rasters to the window raster. This creates three memory rasters, each of a different size. This test renders a polygon to each memory raster, then uses `xgl_image` to copy it back to the window raster. Only regions of the memory rasters are *replroped* (repeated) across.

Attributes Tested:
`XGL_CMAP_COLOR_TABLE`
`XGL_CMAP_COLOR_TABLE_SIZE`
`XGL_DEV_COLOR_MAP`

	XGL_RAS_DEPTH XGL_RAS_HEIGHT XGL_RAS_WIDTH
Operators Tested:	xgl_object_create xgl_polygon xgl_image
Output:	Red, blue, and yellow polygons; some three-, four-, and five-sided

▼ cp_ras_32

Test Types:	CM, RGB
Description:	Tests *copy_raster* with 32-bit RGB rasters. Draws polygons on different-sized 32-bit memory rasters, then copies the memory rasters to the default depth window raster via copy_raster in a replrop (repeated) manner. This was adapted from *xgl_img_2d_32*, but uses *copy_raster* instead of xgl_image.
Attributes Tested:	XGL_CTX_SURF_FRONT_COLOR XGL_RAS_DEPTH XGL_RAS_HEIGHT XGL_RAS_WIDTH
Operators Tested:	xgl_object_create xgl_polygon xgl_context_copy_raster
Output:	First row, 3 red stars; second row, 4 blue boomerangs; third row, 5 yellow quads; fourth row, 13 green triangles

▼ copy_buffer0

Test Types:	INDEX, CM
Description:	Tests simple copy buffer operations using a 2D context. Creates two buffers. Draws to 0th, then copies and renders to buffer 1 twice.
Attributes Tested:	XGL_CTX_RENDER_BUFFER XGL_RAS_SOURCE_BUFFER XGL_WIN_RAS_BUFFERS_ALLOCATED XGL_WIN_RAS_BUFFERS_REQUESTED XGL_WIN_RAS_BUF_DISPLAY XGL_WIN_RAS_BUF_DRAW

Operators Tested: `xgl_object_set`
`xgl_object_get`
`xgl_context_copy_buffer`
`xgl_context_flush`

Output: A diagonal line and quad polygon. The polygon is translated to the lower right.

▼ copy_buffer1

Test Types: INDEX, CM

Description: Tests simple copy buffer operations using a 3D context. Creates two buffers. Buffer 1 is the draw buffer, 0 is the display buffer. Draws, then *copy_buffer* to 0; draws again and copies again.

Attributes Tested: `XGL_CTX_RENDER_BUFFER`
`XGL_RAS_SOURCE_BUFFER`
`XGL_WIN_RAS_BUFFERS_ALLOCATED`
`XGL_WIN_RAS_BUFFERS_REQUESTED`
`XGL_WIN_RAS_BUF_DISPLAY`
`XGL_WIN_RAS_BUF_DRAW`

Operators Tested: `xgl_object_set`
`xgl_object_get`
`xgl_context_copy_buffer`
`xgl_context_flush`

Output: A white diagonal line and quad polygon. The polygon is translated to the lower right.

▼ copy_buffer2

Test Types: RGB, SM

Description: Tests copy buffer operations between two separate window rasters for both 2D and 3D contexts. Create two window rasters, both are in double buffer mode. Copy from back buffer of one raster to back buffer of the another raster and then back to front buffers. Swap the buffers of the source raster. Copy from front to back and then from front to front buffers.

Attributes Tested: `XGL_CTX_RENDER_BUFFER`
`XGL_RAS_SOURCE_BUFFER`
`XGL_WIN_RAS_BUFFERS_ALLOCATED`

	XGL_WIN_RAS_BUFFERS_REQUESTED XGL_WIN_RAS_BUF_DISPLAY XGL_WIN_RAS_BUF_DRAW
Operators Tested:	xgl_object_set xgl_object_get xgl_context_copy_buffer
Output:	Green rectangle with red star markers are drawn in the upper left corner of the destination window raster.

▼ copy_buffer3

Test Types:	RGB, SM
Description:	Tests copy buffer operations from a buffer of window rasterm to a memory raster for both 2D context. Create two window rasters, one in double buffer mode and the other in single buffer mode. Create a memory raster. Copy from the back buffer of the window raster to the memory raster. The memory raster is the copied to the second window raster for checking. Test for the front buffer to the memory raster.
Attributes Tested:	XGL_CTX_RENDER_BUFFER XGL_RAS_SOUCE_BUFFER XGL_WIN_RAS_BUFFERS_ALLOCATED XGL_WIN_RAS_BUFFERS_REQUESTED XGL_WIN_RAS_BUF_DISPLAY XGL_WIN_RAS_BUF_DRAW
Operators Tested:	xgl_object_set xgl_object_get xgl_context_copy_buffer
Output:	Green rectangle with red star markers are drawn in the upper left corner of the destination window raster.

▼ copy_buffer4

Test Types:	RGB, SM
Description:	3D version of copy_buffer3
Attributes Tested:	XGL_CTX_RENDER_BUFFER XGL_RAS_SOURCE_BUFFER XGL_WIN_RAS_BUFFERS_ALLOCATED

	XGL_WIN_RAS_BUFFERS_REQUESTED
	XGL_WIN_RAS_BUF_DISPLAY
	XGL_WIN_RAS_BUF_DRAW
Operators Tested:	xgl_object_set
	xgl_object_get
	xgl_context_copy_buffer
Output:	Green rectangle with red star markers are drawn in the upper left corner of the destination window raster.

▼ copy_buffer_ras_op

Test Types:	RGB, CM
Description:	Copies the buffer with different ROP operations
Attributes Tested:	XGL_CTX_RENDER_BUFFER
	XGL_CTX_ROP
	XGL_CTX_SURF_FRONT_COLOR
	XGL_RAS_SOURCE_BUFFER
	XGL_WIN_RAS_BUFFERS_ALLOCATED
	XGL_WIN_RAS_BUFFERS_REQUESTED
	XGL_WIN_RAS_BUF_DISPLAY
	XGL_WIN_RAS_BUF_DRAW
Operators Tested:	xgl_object_set
	xgl_object_get
	xgl_context_copy_buffer
	xgl_polygon
Output:	Quad polygons, in different ROP modes. Notes from bug 1120966: because the clut will be in different states, the values being ropped are nondeterministic for the way the test is written, so the images can change but still be correct.

▼ win_backing_store

Test Types:	INDEX, SM
Description:	Tests backing store by creating a child window to use to obscure the parent window. Polygons are rendered to the parent, the child is mapped and unmapped, and polygons are checked.

Attributes Tested:	XGL_CMAP_COLOR_TABLE XGL_CMAP_COLOR_TABLE_SIZE XGL_CTX_NEW_FRAME_ACTION (XGL_CTX_NEW_FRAME_CLEAR) XGL_DEV_COLOR_MAP XGL_WIN_RAS_BACKING_STORE
Operators Tested:	xgl_context_new_frame xgl_context_post xgl_context_flush xgl_multipolyline xgl_multi_simple_polygon
Output:	A large red polygon, which is obscured four times by a smaller window

▼ winras_resize

Test Types:	INDEX, SM
Description:	Tests window raster resize
Attributes Tested:	XGL_RAS_RECT_LIST XGL_RAS_RECT_NUM
Operators Tested:	xgl_window_raster_resize
Output:	Windows with different sizes

▼ ras_copy5

Test Types:	INDEX, SM
Description:	Tests Fb->Fb copy by rendering to the screen and then copying
Attributes Tested:	XGL_CTX_MARKER XGL_CTX_MARKER_SCALE_FACTOR XGL_CTX_MARKER_COLOR XGL_CTX_LINE_COLOR XGL_CTX_ROP
Operators Tested:	xgl_object_set xgl_context_copy_raster xgl_multimarker xgl_multipolyline
Output:	A green “*” marker is drawn, at 100,100 surrounded by a green 100×100 square and a red 102×102 square (the two squares together look yellow). The green marker and

square are then copied to 50,350. All green pixels and no red pixels should be copied. The two squares are then copied to the right to make four. Finally, an overlapping copy with XGL_CTX_ROP set to XGL_ROP_OR should result in eight squares. This process is repeated for 2D and 3D contexts.

▼ ras_copy6

Test Types:	INDEX, SM
Description:	Tests Fb->Fb overlapping copy for single / multiple clip rects
Attributes Tested:	XGL_CTX_MARKER XGL_CTX_MARKER_SCALE_FACTOR XGL_CTX_MARKER_COLOR XGL_CTX_LINE_COLOR XGL_CTX_ROP
Operators Tested:	xgl_object_set xgl_context_copy_raster xgl_multimarker xgl_multipolyline
Output:	A green “*” marker is drawn, at 100,100 surrounded by a green 100×100 square. The square is then copied in an overlapping manner. This is done four times, for overlaps on all four sides of the square. In addition, another set of copies will attempt to copy the square over the edges of the window, to assure that window clipping works properly. A sub-window will be created for the second passes of the tests such that it will force XGL to test the multiple clip rects cases. This process is repeated for 2D and 3D contexts.

▼ ras_copy7

Test Types:	INDEX, SM
Description:	Tests Mem1->Fb copy for single / multiple clip rects

Attributes Tested:	XGL_CTX_MARKER XGL_CTX_MARKER_SCALE_FACTOR XGL_CTX_MARKER_COLOR XGL_CTX_SURF_FRONT_COLOR XGL_CTX_BACKGROUND_COLOR
Operators Tested:	xgl_object_set xgl_context_copy_raster xgl_multimarker xgl_multipolyline
Output:	A “*” marker is drawn into a 1-bit memory raster. This raster is then copied to the framebuffer with xgl_context_copy_raster. The SURF_FRONT_COLOR and BACKGROUND_COLOR are changed and a series of small copies are done at 1-pixel intervals to verify that the 1-bit to N-bit copy is performed correctly for all offsets and phases. A sub-window will be created during the second passes of the tests such that it will force XGL to test the multiple clip rects cases.

▼ ras_copy7a

Test Types:	INDEX, SM
Description:	Tests Mem1->Fb copy to back buffer in double buffer mode
Attributes Tested:	XGL_CTX_MARKER XGL_CTX_MARKER_SCALE_FACTOR XGL_CTX_MARKER_COLOR XGL_CTX_SURF_FRONT_COLOR XGL_CTX_BACKGROUND_COLOR XGL_RAS_SOURCE_BUFFER XGL_WIN_RAS_BUFFERS_REQUESTED XGL_WIN_RAS_BUFFERS_ALLOCATED XGL_WIN_RAS_BUF_DRAW XGL_WIN_RAS_BUF_DISPLAY
Operators Tested:	xgl_object_set xgl_context_copy_buffer xgl_multimarker xgl_multipolyline

Output:	A double buffered version of `ras_copy7`. A '*' marker is drawn into a 1-bit memory raster. This raster is then copied to the back_buffer of the frame buffer with `xgl_context_copy_buffer()`. The `SURF_FRONT_COLOR` and `BACKGROUND_COLOR` are changed and a series of small copies are done at 1-pixel intervals to verify that the 1-bit to N-bit copy is performed correctly for all offsets and phases.

▼ ras_copy8

Test Types:	RGB, SM
Description:	Tests `xgl_context_copy_raster()` and `xgl_image()` which can be used to copy a block of pixels from a 1-bit memory raster to a window raster with multiple clip rects and stencil fill style.

Attributes Tested:	`XGL_CTX_DEFERRAL_MODE`
	`XGL_CTX_RASTER_FILL_STYLE`
	`XGL_CTX_SURF_FRONT_COLOR`
	`XGL_CTX_BACKGROUND_COLOR`
	`XGL_RAS_DEPTH`
	`XGL_RAS_WIDTH`
	`XGL_RAS_HEIGHT`
	`XGL_MEM_RAS_IMAGE_BUFFER_ADDR`

Operators Tested:	`xgl_object_set`
	`xgl_context_copy_raster`
	`xgl_image`

Output:	A 78x80 1-bit memory raster is created and then copied to the framebuffer with `xgl_context_copy_raster()` and `xgl_image()` with `XGL_RAS_FILL_COPY` and `XGL_RAS_FILL_STENCIL` raster fill styles. A sub-window will be created during the second passes of the tests such that multiple clip rects cases can also be tested.

▼ ras_copy9

Test Types:	RGB, SM
Description:	Tests that `xgl_context_copy_raster()` can be used to copy a block of pixels from a 24-bit window raster to a memory raster.

Attributes Tested: XGL_CTX_DEFERRAL_MODE
XGL_RAS_DEPTH
XGL_RAS_WIDTH
XGL_RAS_HEIGHT
Operators Tested: xgl_object_set
xgl_context_copy_raster
Output: A 600x500 rgb window is created. A marker and the text "XGL" are drawn, then copied to a memory raster. The window is then cleared, and the memory raster is copied back to the window for comparison.

▼ ras_copy10

Test Types: RGB, SM
Description: Tests that xgl_context_copy_raster() can be used to copy a block of pixels from window raster to application allocated array attatched to memory raster.
Attributes Tested: XGL_MEM_RAS_MEMORY_ADDRESS
Operators Tested: xgl_object_set
xgl_context_copy_buffer
Output: Four different colored squares - red, green, blue and white are displayed.

▼ ovl_inq

Test Types: SM, RGB
Description: Tests transparent overlay attributes
Attributes Tested: XGL_CTX_NEW_FRAME_PAINT_TYPE
XGL_CTX_PAINT_TYPE
Operators Tested: xgl_object_create
xgl_context_new_frame
xgl_inquire
Output: Text indicates whether overlay is available

▼ ras_stencil

Test Types: SM, RGB
Description: Tests stencil raster fill test and various foreground / background colors

Attributes Tested:	XGL_CTX_DEFERRAL_MODE XGL_3D_CTX_LIGHT_NUM XGL_3D_CTX_SURF_FRONT_AMBIENT XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT XGL_LIGHT_TYPE XGL_LIGHT_COLOR XGL_3D_CTX_LIGHT_SWITCHES XGL_CTX_SURF_FRONT_FILL_STYLE XGL_3D_CTX_SURF_FRONT_ILLUMINATION XGL_CTX_RASTER_FILL_STYLE XGL_CTX_SURF_FRONT_COLOR XGL_CTX_BACKGROUND_COLOR XGL_RAS_DEPTH XGL_RAS_WIDTH XGL_RAS_HEIGHT XGL_MEM_RAS_IMAGE_BUFFER_ADDR
Operators Tested:	xgl_quadrilateral_mesh xgl_image xgl_context_copy_buffer
Output:	A 1-bit "Sun #1" memory raster is drawn diagonally in various colors in front of a shaded quad.

▼ one_context_two_rasters

Test Types:	RGB, SM
Description:	Creates one context and two rasters. Test switching the rasters will not jumble up the raster attributes.
Attributes Tested:	XGL_CTX_DEVICE XGL_RAS_RECT_NUM XGL_RAS_RECT_LIST
Operators Tested:	xgl_object_set xgl_context_new_frame xgl_multi_simple_polygon
Output:	Two rasters are created with two yellow squares drawn followed by green rectangles on top of the squares on opposite sides.

Rectangle Test Descriptions 25

This chapter describes the Rectangle test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section "Denizen Test Types" on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ rect0

Test Types:	INDEX, SM
Description:	Tests simple rectangle drawing: draws a rectangle of the type `XGL_MULTIRECT_I2D` and checks five points of the rectangle
Attributes Tested:	`XGL_CTX_SURF_FRONT_COLOR`
Operators Tested:	`xgl_object_set` `xgl_multirectangle`
Output:	A white square

▼ rect1

Test Types:	INDEX, SM
Description:	Draws three rectangles of the type XGL_MULTIRECT_I2D and checks five points of each of the rectangles
Attributes Tested:	XGL_CTX_SURF_FRONT_COLOR
Operators Tested:	xgl_object_set xgl_multirectangle
Output:	Three white squares of different sizes are drawn one after the other.

▼ rect2

Test Types:	INDEX, SM
Description:	Draws three rectangles of the type XGL_MULTIRECT_I2D with edges on and checks for the presence/absence of edge color at five points on each of the rectangles.
Attributes Tested:	XGL_CTX_EDGE_COLOR XGL_CTX_SURF_EDGE_FLAG XGL_CTX_SURF_FRONT_COLOR
Operators Tested:	xgl_object_set xgl_multirectangle
Output:	Three white squares of different sizes are drawn one after the other.

▼ rect3

Test Types:	INDEX, SM
Description:	Draws three rectangles of the type XGL_MULTIRECT_I2D with edges both on and off, and with three different stipple patterns. Checks for edge presence and the patterns on the rectangles.
Attributes Tested:	XGL_CTX_SURF_EDGE_FLAG XGL_CTX_SURF_FPAT XGL_CTX_SURF_FPAT_POSITION XGL_CTX_SURF_FRONT_FILL_STYLE (XGL_SURF_FILL_STIPPLE)
Operators Tested:	xgl_object_set xgl_multirectangle

Output:	Three white squares are drawn six times, one after the other, with the following stipple pattern styles and edge flags: (1) Dots stipple pattern, no edge (2) Cross-hatch stipple pattern, no edge (3) Dotted-screen stipple pattern, no edge (4) Dots stipple pattern with edge (5) Cross-hatch stipple pattern with edge (6) Dotted-screen stipple pattern with edge

▼ rect4

Test Types:	INDEX, SM
Description:	Draws a rectangle of the type `XGL_MULTIRECT_I2D` with dual-colored edges and checks for both colors on the boundary of the rectangle.

Attributes Tested:	`XGL_CTX_EDGE_ALT_COLOR`
	`XGL_CTX_EDGE_COLOR`
	`XGL_CTX_EDGE_PATTERN`
	`XGL_CTX_EDGE_STYLE (XGL_LINE_ALT_PATTERNED)`

Operators Tested:	`xgl_object_set` `xgl_multirectangle`
Output:	A green solid square with dual-colored (white and red) edges on a white background

▼ rect5

Test Types:	RGB, SM
Description:	Tests simple RGB rectangle drawing: draws a rectangle of the type `XGL_MULTIRECT_I2D` and checks five points of the rectangle
Attributes Tested:	`XGL_CTX_SURF_FRONT_COLOR`
Operators Tested:	`xgl_object_set` `xgl_multirectangle`
Output:	A white square

▼ rect6

Test Types: RGB, SM
Description: Draws three RGB rectangles of the type `XGL_MULTIRECT_I2D` and checks five points of each of the rectangles
Attributes Tested: `XGL_CTX_SURF_FRONT_COLOR`
Operators Tested: `xgl_object_set`
`xgl_multirectangle`
Output: Three white squares of different sizes are drawn one after the other.

▼ rect7

Test Types: RGB, SM
Description: Draws three RGB rectangles of the type `XGL_MULTIRECT_I2D` with edges on and checks for the presence/absence of edge color at five points on each of the rectangles
Attributes Tested: `XGL_CTX_EDGE_COLOR`
`XGL_CTX_SURF_EDGE_FLAG`
`XGL_CTX_SURF_FRONT_COLOR`
Operators Tested: `xgl_object_set`
`xgl_multirectangle`
Output: Three white squares of different sizes with blue edges are drawn one after the other.

▼ rect8

Test Types: RGB, SM
Description: Draws three RGB rectangles of the type `XGL_MULTIRECT_I2D` with edges both on and off, and with three different stipple patterns. Checks for edge presence and the patterns on the rectangles.
Attributes Tested: `XGL_CTX_SURF_EDGE_FLAG`
`XGL_CTX_SURF_FPAT`
`XGL_CTX_SURF_FPAT_POSITION`
`XGL_CTX_SURF_FRONT_FILL_STYLE`
`(XGL_SURF_FILL_STIPPLE)`

Operators Tested: `xgl_object_set`
`xgl_multirectangle`

Output: Three white squares are drawn six times, one after the other, with the following stipple pattern styles and edge flags:
(1) Dots stipple pattern, no edge
(2) Cross-hatch stipple pattern, no edge
(3) Dotted-screen stipple pattern, no edge
(4) Dots stipple pattern with edge
(5) Cross-hatch stipple pattern with edge
(6) Dotted-screen stipple pattern with edge

▼ rect9

Test Types: RGB, SM

Description: Draws an RGB rectangle of the type `XGL_MULTIRECT_I2D` with dual-colored edges and checks for both colors on the boundary of the rectangle

Attributes Tested: `XGL_CTX_EDGE_ALT_COLOR`
`XGL_CTX_EDGE_COLOR`
`XGL_CTX_EDGE_PATTERN`
`XGL_CTX_EDGE_STYLE (XGL_LINE_ALT_PATTERNED)`

Operators Tested: `xgl_object_set`
`xgl_multirectangle`

Output: A green solid square with dual-colored (white and red) edges on a black background

▼ rect10

Test Types: RGB, SM

Description: Draws three RGB rectangles of the type `XGL_MULTIRECT_I2D` and checks five points of each of the rectangles. Tries many different colors for the polygons (all colors in color cube for 8-bit rasters).

Attributes Tested: `XGL_CMAP_COLOR_CUBE_SIZE`
`XGL_CTX_SURF_FRONT_COLOR`
`XGL_DEV_COLOR_MAP`
`XGL_RAS_DEPTH`

Operators Tested: xgl_object_get
xgl_object_set
xgl_multirectangle

Output: For 8-bit rasters, three squares of different sizes are drawn one after the other many times. Each time a different color in the color cube is used as the interior color. For non-8-bit rasters, nothing is rendered.

▼ rect11

Test Types: INDEX, SM

Description: Tests simple rectangle drawing: draws a rectangle of the type XGL_MULTIRECT_F3D and checks five points of the rectangle

Attributes Tested: XGL_CTX_SURF_FRONT_COLOR

Operators Tested: xgl_object_set
xgl_multirectangle

Output: A white square

▼ rect12

Test Types: INDEX, SM

Description: Draws three rectangles of the type XGL_MULTIRECT_F3D and checks five points of each of the rectangles

Attributes Tested: XGL_CTX_SURF_FRONT_COLOR

Operators Tested: xgl_object_set
xgl_multirectangle

Output: Three white squares of different sizes are drawn one after the other.

▼ rect13

Test Types: INDEX, SM

Description: Draws three rectangles of the type XGL_MULTIRECT_F3D with edges on and checks for the presence/absence of edge color at five points on each of the rectangles

Attributes Tested: XGL_CTX_EDGE_COLOR
XGL_CTX_SURF_EDGE_FLAG
XGL_CTX_SURF_FRONT_COLOR

Operators Tested: `xgl_object_set`
`xgl_multirectangle`

Output: Three white squares of different sizes are drawn one after the other.

▼ rect14

Test Types: INDEX, SM

Description: Draws three rectangles of the type XGL_MULTIRECT_F3D with edges both on and off, and with three different stipple patterns. Checks for edge presence and the patterns on the rectangles.

Attributes Tested: XGL_CTX_SURF_EDGE_FLAG
XGL_CTX_SURF_FPAT
XGL_CTX_SURF_FPAT_POSITION
XGL_CTX_SURF_FRONT_FILL_STYLE
(XGL_SURF_FILL_STIPPLE)

Operators Tested: `xgl_object_set`
`xgl_multirectangle`

Output: Three white squares are drawn six times, one after the other, with the following stipple pattern styles and edge flags:
(1) Dots stipple pattern, no edge
(2) Cross-hatch stipple pattern, no edge
(3) Dotted-screen stipple pattern, no edge
(4) Dots stipple pattern with edge
(5) Cross-hatch stipple pattern with edge
(6) Dotted-screen stipple pattern with edge

▼ rect15

Test Types: INDEX, SM

Description: Draws a rectangle of the type XGL_MULTIRECT_F3D with dual-colored edges and checks for both colors on the boundary of the rectangle

Attributes Tested: XGL_CTX_EDGE_ALT_COLOR
XGL_CTX_EDGE_COLOR
XGL_CTX_EDGE_PATTERN
XGL_CTX_EDGE_STYLE (XGL_LINE_ALT_PATTERNED)

Operators Tested:	xgl_object_set xgl_multirectangle
Output:	A green solid square with dual-colored (white and red) edges on a white background

▼ rect16

Test Types:	RGB, SM
Description:	Tests simple RGB rectangle drawing: draws a rectangle of the type XGL_MULTIRECT_F3D and checks five points of the rectangle
Attributes Tested:	XGL_CTX_SURF_FRONT_COLOR
Operators Tested:	xgl_object_set xgl_multirectangle
Output:	A white square

▼ rect17

Test Types:	RGB, SM
Description:	Draws three RGB rectangles of the type XGL_MULTIRECT_F3D and checks five points of each of the rectangles
Attributes Tested:	XGL_CTX_SURF_FRONT_COLOR
Operators Tested:	xgl_object_set xgl_multirectangle
Output:	Three white squares of different sizes are drawn one after the other.

▼ rect18

Test Types:	RGB, SM
Description:	Draws three RGB rectangles of the type XGL_MULTIRECT_F3D with edges on and check for the presence/absence of edge color at five points on each of the rectangles
Attributes Tested:	XGL_CTX_EDGE_COLOR XGL_CTX_SURF_EDGE_FLAG XGL_CTX_SURF_FRONT_COLOR

Operators Tested:	`xgl_object_set` `xgl_multirectangle`
Output:	Three white squares of different sizes with blue edges are drawn one after the other.

▼ rect19

Test Types:	RGB, SM
Description:	Draws three RGB rectangles of the type `XGL_MULTIRECT_F3D` with edges both on and off, and with three different stipple patterns. Checks for edge presence and the patterns on the rectangles.

Attributes Tested:	`XGL_CTX_SURF_EDGE_FLAG`
	`XGL_CTX_SURF_FPAT`
	`XGL_CTX_SURF_FPAT_POSITION`
	`XGL_CTX_SURF_FRONT_FILL_STYLE`
	`(XGL_SURF_FILL_STIPPLE)`

Operators Tested:	`xgl_object_set` `xgl_multirectangle`
Output:	Three white squares are drawn six times, one after the other, with the following stipple pattern styles and edge flags: (1) Dots stipple pattern, no edge (2) Cross-hatch stipple pattern, no edge (3) Dotted-screen stipple pattern, no edge (4) Dots stipple pattern with edge (5) Cross-hatch stipple pattern with edge (6) Dotted-screen stipple pattern with edge

▼ rect20

Test Types:	RGB, SM
Description:	Draws an RGB rectangle of the type `XGL_MULTIRECT_F3D` with dual-colored edges and checks for both colors on the boundary of the rectangle

Attributes Tested:	`XGL_CTX_EDGE_ALT_COLOR`
	`XGL_CTX_EDGE_COLOR`
	`XGL_CTX_EDGE_PATTERN`
	`XGL_CTX_EDGE_STYLE (XGL_LINE_ALT_PATTERNED)`

Operators Tested:	xgl_object_set xgl_multirectangle
Output:	A green solid square with dual-colored (white and red) edges on a black background

▼ rect21

Test Types:	RGB, SM
Description:	Draws three RGB rectangles of the type XGL_MULTIRECT_F3D and checks five points of each of the rectangles. Many different colors for the polygons (all colors in color cube for 8-bit rasters) are tried.
Attributes Tested:	XGL_CMAP_COLOR_CUBE_SIZE XGL_CTX_SURF_FRONT_COLOR XGL_DEV_COLOR_MAP XGL_RAS_DEPTH
Operators Tested:	xgl_object_get xgl_object_set xgl_multirectangle
Output:	For 8-bit rasters, three squares of different sizes are drawn one after the other many times. Each time a different color in the color cube is used as the interior color. For non-8-bit rasters, nothing is rendered.

▼ rect_annot_af3d_nonid_trans_rgb

Test Types:	RGB, SM
Description:	Draws two RGB rectangles of the type XGL_MULTIRECT_AF3D translated in y direction
Attributes Tested:	XGL_CTX_GLOBAL_MODEL_TRANS
Operators Tested:	xgl_object_get xgl_transform_translate xgl_multirectangle xgl_transform_identity
Output:	Two green rectangles with different sizes

▼ rect_annot_af3d_rgb

Test Types:	RGB, SM
Description:	Draws two RGB rectangles of the type `XGL_MULTIRECT_AF3D`
Attributes Tested:	None
Operators Tested:	`xgl_multirectangle`
Output:	Two green rectangles with different sizes

Set and Get Attribute Test Descriptions

26

This chapter describes the Set and Get attribute test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ set_get_cmap

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the color map attributes.
Attributes Tested:	XGL_CMAP_DITHER_MASK_N XGL_CMAP_DITHER_MASK
Operators Tested:	xgl_object_get xgl_object_set
Output:	None

▼ set_get_ctx1

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the environment context attributes (nonpushable)
Attributes Tested:	See Table 26-1, Column B at the end of this chapter.
Operators Tested:	`xgl_object_get` `xgl_object_set`
Output:	None

▼ set_get_ctx2

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the depth cue context attributes

Attributes Tested:	`XGL_3D_CTX_DEPTH_CUE_COLOR`
	`XGL_3D_CTX_DEPTH_CUE_INTERP`
	`XGL_3D_CTX_DEPTH_CUE_MODE`
	`XGL_3D_CTX_DEPTH_CUE_REF_PLANES`
	`XGL_3D_CTX_DEPTH_CUE_SCALE_FACTORS`
	`XGL_3D_CTX_LIGHTS`
	`XGL_3D_CTX_LIGHT_NUM`
	`XGL_CTX_VDC_ORIENTATION`

Operators Tested:	`xgl_object_get` `xgl_object_set`
Output:	None

▼ set_get_ctx3

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the graphics context attributes (view model)

Attributes Tested:	`XGL_3D_CTX_MODEL_CLIP_PLANES`
	`XGL_3D_CTX_MODEL_CLIP_PLANE_NUM`
	`XGL_3D_CTX_VIEW_CLIP_PLUS_W_ONLY`
	`XGL_CTX_CLIP_PLANES`
	`XGL_CTX_DC_VIEWPORT`
	`XGL_CTX_GLOBAL_MODEL_TRANS`
	`XGL_CTX_LOCAL_MODEL_TRANS`
	`XGL_CTX_VDC_MAP`

	XGL_CTX_VDC_WINDOW
	XGL_CTX_VIEW_CLIP_BOUNDS
	XGL_CTX_VIEW_TRANS
Operators Tested:	xgl_object_get
	xgl_object_set
Output:	None

▼ set_get_ctx4

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the graphics context attributes—general rendering (Xgl)
Attributes Tested:	XGL_3D_CTX_HLHSR_DATA
	XGL_3D_CTX_HLHSR_MODE
	XGL_CTX_BACKGROUND_COLOR
	XGL_CTX_NEW_FRAME_ACTION
	XGL_CTX_PLANE_MASK
	XGL_CTX_RASTER_FILL_STYLE
	XGL_CTX_RASTER_FPAT
	XGL_CTX_RASTER_FPAT_POSITION
	XGL_CTX_RASTER_STIPPLE_COLOR
	XGL_CTX_ROP
	XGL_CTX_THRESHOLD
Operators Tested:	xgl_object_get
	xgl_object_set
Output:	None

▼ set_get_ctx5

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the graphics context attributes—line rendering (Xgl)
Attributes Tested:	XGL_3D_CTX_LINE_COLOR_INTERP
	XGL_CTX_LINE_ALT_COLOR
	XGL_CTX_LINE_CAP
	XGL_CTX_LINE_COLOR
	XGL_CTX_LINE_COLOR_SELECTOR
	XGL_CTX_LINE_JOIN
	XGL_CTX_LINE_MITER_LIMIT

	XGL_CTX_LINE_PATTERN
	XGL_CTX_LINE_STYLE
	XGL_CTX_LINE_WIDTH_SCALE_FACTOR
Operators Tested:	xgl_object_get
	xgl_object_set
Output:	None

▼ set_get_ctx6

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the graphics context attributes: • Curve and surface maximum tessellation • Curve rendering (Xgl)
Attributes Tested:	XGL_CTX_MAX_TESSELLATION
	XGL_CTX_MIN_TESSELLATION
	XGL_CTX_NURBS_CURVE_APPROX
	XGL_CTX_NURBS_CURVE_APPROX_VAL
Operators Tested:	xgl_object_get
	xgl_object_set
Output:	None

▼ set_get_ctx7

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the graphics context attributes—surface rendering (Xgl)
Attributes Tested:	XGL_CTX_ARC_FILL_STYLE
	XGL_CTX_EDGE_ALT_COLOR
	XGL_CTX_EDGE_COLOR
	XGL_CTX_EDGE_PATTERN
	XGL_CTX_EDGE_STYLE
	XGL_CTX_EDGE_WIDTH_SCALE_FACTOR
	XGL_CTX_SURF_EDGE_FLAG
	XGL_CTX_SURF_FRONT_COLOR
	XGL_CTX_SURF_FRONT_COLOR_SELECTOR
	XGL_CTX_SURF_FRONT_FILL_STYLE
	XGL_CTX_SURF_FRONT_FPAT
	XGL_CTX_SURF_FRONT_FPAT_POSITION
	XGL_CTX_SURF_INTERIOR_RULE

Operators Tested:	xgl_object_get xgl_object_set
Output:	None

▼ set_get_ctx8

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the graphics context attributes—surface rendering (3D Xgl) [Test 1]
Attributes Tested:	XGL_3D_CTX_SURF_BACK_COLOR XGL_3D_CTX_SURF_BACK_COLOR_SELECTOR XGL_3D_CTX_SURF_BACK_FILL_STYLE XGL_3D_CTX_SURF_BACK_FPAT XGL_3D_CTX_SURF_BACK_FPAT_POSITION XGL_3D_CTX_SURF_DC_OFFSET XGL_3D_CTX_SURF_FACE_CULL XGL_3D_CTX_SURF_FACE_DISTINGUISH XGL_3D_CTX_SURF_NORMAL_FLIP XGL_3D_CTX_SURF_SILHOUETTE_EDGE_FLAG
Operators Tested:	xgl_object_get xgl_object_set
Output:	None

▼ set_get_ctx9

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the graphics context attributes—surface rendering (3D Xgl) [Test 2]
Attributes Tested:	XGL_3D_CTX_SURF_BACK_AMBIENT XGL_3D_CTX_SURF_BACK_DIFFUSE XGL_3D_CTX_SURF_BACK_ILLUMINATION XGL_3D_CTX_SURF_BACK_SPECULAR XGL_3D_CTX_SURF_BACK_SPECULAR_COLOR XGL_3D_CTX_SURF_FRONT_AMBIENT XGL_3D_CTX_SURF_FRONT_DIFFUSE XGL_3D_CTX_SURF_FRONT_ILLUMINATION XGL_3D_CTX_SURF_FRONT_SPECULAR XGL_3D_CTX_SURF_FRONT_SPECULAR_COLOR
Operators Tested:	xgl_object_get xgl_object_set

Output: None

▼ set_get_ctx10

Test Types: INDEX, SM
Description: Tests the setting and getting of the graphics context attributes—surface rendering (3D Xgl) [Test 3]
Attributes Tested: XGL_3D_CTX_LIGHT_NUM
XGL_3D_CTX_LIGHT_SWITCHES
XGL_3D_CTX_SURF_BACK_LIGHT_COMPONENT
XGL_3D_CTX_SURF_BACK_LIGHT_TYPE
XGL_3D_CTX_SURF_BACK_SPECULAR_POWER
XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT
XGL_3D_CTX_SURF_FRONT_LIGHT_TYPE
XGL_3D_CTX_SURF_FRONT_SPECULAR_POWER
XGL_3D_CTX_SURF_GEOM_NORMAL
Operators Tested: xgl_object_get
xgl_object_set
Output: None

▼ set_get_ctx11

Test Types: INDEX, SM
Description: Tests the setting and getting of the graphics context attributes—marker rendering (Xgl)
Attributes Tested: XGL_CTX_MARKER
XGL_CTX_MARKER_COLOR
XGL_CTX_MARKER_COLOR_SELECTOR
XGL_CTX_MARKER_SCALE_FACTOR
Operators Tested: xgl_object_get
xgl_object_set
Output: None

▼ set_get_ctx12

Test Types: INDEX, SM
Description: Tests the setting and getting of the graphics context attributes—stroke fonts
Attributes Tested: See Table 26-1, Column C at the end of this chapter.

Operators Tested:	xgl_object_get xgl_object_set
Output:	None

▼ set_get_ctx13

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the graphics context attributes—picking
Attributes Tested:	XGL_CTX_PICK_ID_1 XGL_CTX_PICK_ID_2
Operators Tested:	xgl_object_get xgl_object_set
Output:	None

▼ set_get_ctx14

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the graphics context attributes—annotation text
Attributes Tested:	See Table 26-1, Column A at the end of this chapter.
Operators Tested:	xgl_object_get xgl_object_set
Output:	None

▼ set_get_ctx15

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the 2D environment context attributes (nonpushable)
Attributes Tested:	See Table 26-1, Column B at the end of this chapter.
Operators Tested:	xgl_object_get xgl_object_set
Output:	None

▼ set_get_ctx16

Test Types:	INDEX, SM

Description:	Tests the setting and getting of the 2D graphics context attributes (view model)
Attributes Tested:	XGL_CTX_CLIP_PLANES XGL_CTX_DC_VIEWPORT XGL_CTX_GLOBAL_MODEL_TRANS XGL_CTX_LOCAL_MODEL_TRANS XGL_CTX_VDC_MAP XGL_CTX_VDC_WINDOW XGL_CTX_VIEW_CLIP_BOUNDS XGL_CTX_VIEW_TRANS XGL_TRANS_DIMENSION
Operators Tested:	xgl_object_get xgl_object_set
Output:	None

▼ set_get_ctx17

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the 2D graphics context attributes—general rendering (Xgl)
Attributes Tested:	XGL_CTX_BACKGROUND_COLOR XGL_CTX_NEW_FRAME_ACTION XGL_CTX_PLANE_MASK XGL_CTX_RASTER_FILL_STYLE XGL_CTX_RASTER_FPAT XGL_CTX_RASTER_FPAT_POSITION XGL_CTX_RASTER_STIPPLE_COLOR XGL_CTX_ROP XGL_CTX_THRESHOLD
Operators Tested:	xgl_object_get xgl_object_set
Output:	None

▼ set_get_ctx18

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the 2D graphics context attributes—line rendering (Xgl)

Attributes Tested:	XGL_CTX_LINE_ALT_COLOR XGL_CTX_LINE_CAP XGL_CTX_LINE_COLOR XGL_CTX_LINE_COLOR_SELECTOR XGL_CTX_LINE_JOIN XGL_CTX_LINE_MITER_LIMIT XGL_CTX_LINE_PATTERN XGL_CTX_LINE_STYLE XGL_CTX_LINE_WIDTH_SCALE_FACTOR
Operators Tested:	xgl_object_get xgl_object_set
Output:	None

▼ set_get_ctx19

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the 2D graphics context attributes: • Curve and surface maximum tessellation • Curve rendering (Xgl)
Attributes Tested:	XGL_CTX_MAX_TESSELLATION XGL_CTX_MIN_TESSELLATION XGL_CTX_NURBS_CURVE_APPROX XGL_CTX_NURBS_CURVE_APPROX_VAL
Operators Tested:	xgl_object_get xgl_object_set
Output:	None

▼ set_get_ctx20

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the 2D graphics context attributes—surface rendering (Xgl)
Attributes Tested:	XGL_CTX_ARC_FILL_STYLE XGL_CTX_EDGE_ALT_COLOR XGL_CTX_EDGE_COLOR XGL_CTX_EDGE_PATTERN XGL_CTX_EDGE_STYLE XGL_CTX_EDGE_WIDTH_SCALE_FACTOR XGL_CTX_SURF_EDGE_FLAG

	XGL_CTX_SURF_FRONT_COLOR
	XGL_CTX_SURF_FRONT_COLOR_SELECTOR
	XGL_CTX_SURF_FRONT_FILL_STYLE
	XGL_CTX_SURF_FRONT_FPAT
	XGL_CTX_SURF_FRONT_FPAT_POSITION
	XGL_CTX_SURF_INTERIOR_RULE
Operators Tested:	xgl_object_get
	xgl_object_set
Output:	None

set_get_ctx21

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the 2D graphics context attributes—marker rendering (Xgl)
Attributes Tested:	XGL_CTX_MARKER
	XGL_CTX_MARKER_COLOR
	XGL_CTX_MARKER_COLOR_SELECTOR
	XGL_CTX_MARKER_SCALE_FACTOR
Operators Tested:	xgl_object_get
	xgl_object_set
Output:	None

set_get_ctx22

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the 2D graphics context attributes—stroke fonts
Attributes Tested:	See Table 26-1 Column C at the end of this chapter.
Operators Tested:	xgl_object_get
	xgl_object_set
Output:	None

set_get_ctx23

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the 2D graphics context attributes—picking

Attributes Tested: XGL_CTX_PICK_ID_1
XGL_CTX_PICK_ID_2
Operators Tested: xgl_object_get
xgl_object_set
Output: None

▼ set_get_ctx24

Test Types: INDEX, SM
Description: Tests the setting and getting of the 2D graphics context attributes—annotation text
Attributes Tested: XGL_CTX_ATEXT_ALIGN_HORIZ
XGL_CTX_ATEXT_ALIGN_VERT
XGL_CTX_ATEXT_CHAR_HEIGHT
XGL_CTX_ATEXT_CHAR_SLANT_ANGLE
XGL_CTX_ATEXT_CHAR_UP_VECTOR
XGL_CTX_ATEXT_PATH
XGL_CTX_ATEXT_STYLE
Operators Tested: xgl_object_get
xgl_object_set
Output: None

▼ set_get_light

Test Types: INDEX, SM
Description: Tests the setting and getting of the light source attributes
Attributes Tested: XGL_LIGHT_ATTENUATION_1
XGL_LIGHT_ATTENUATION_2
XGL_LIGHT_COLOR
XGL_LIGHT_DIRECTION
XGL_LIGHT_POSITION
XGL_LIGHT_SPOT_ANGLE
XGL_LIGHT_SPOT_EXPONENT
XGL_LIGHT_TYPE
Operators Tested: xgl_object_get
xgl_object_set
Output: None

▼ set_get_lpat

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the line pattern attributes
Attributes Tested:	XGL_LPAT_BALANCED_DASH
	XGL_LPAT_COORD_SYS
	XGL_LPAT_DATA
	XGL_LPAT_DATA_SIZE
	XGL_LPAT_DATA_TYPE
	XGL_LPAT_OFFSET
	XGL_LPAT_STYLE
Operators Tested:	xgl_object_get
	xgl_object_set
Output:	None

▼ set_get_sfont

Test Types:	INDEX, SM
Description:	Tests the setting and getting of the stroke font attributes
Attributes Tested:	XGL_SFONT_COMMENT
	XGL_SFONT_DEFAULT_CHARACTER
	XGL_SFONT_IS_MONO_SPACED
	XGL_SFONT_NAME
Operators Tested:	xgl_object_get
	xgl_object_set
Output:	None

Table 26-1 Set and Get Attributes Tested

Column A	**Column B**	**Column C**
XGL_CTX_ATEXT_ALIGN_HORIZ	XGL_CTX_DEFERRAL_MODE	XGL_CTX_SFONT_0
XGL_CTX_ATEXT_ALIGN_VERT	XGL_CTX_MODEL_TRANS_STACK_SIZE	XGL_CTX_SFONT_1
XGL_CTX_ATEXT_CHAR_HEIGHT	XGL_CTX_PICK_APERTURE	XGL_CTX_SFONT_2
XGL_CTX_ATEXT_CHAR_SLANT_ANGLE	XGL_CTX_PICK_BUFFER_SIZE	XGL_CTX_SFONT_3
XGL_CTX_ATEXT_CHAR_UP_VECTOR	XGL_CTX_PICK_ENABLE	XGL_CTX_STEXT_ALIGN_HORIZ
XGL_CTX_ATEXT_PATH	XGL_CTX_PICK_STYLE	XGL_CTX_STEXT_ALIGN_VERT
XGL_CTX_ATEXT_STYLE	XGL_CTX_PICK_SURF_STYLE	XGL_CTX_STEXT_CHAR_ENCODING
	XGL_CTX_RENDERING	XGL_CTX_STEXT_CHAR_EXPANSION_FACTOR
	XGL_CTX_VDC_ORIENTATION	XGL_CTX_STEXT_CHAR_HEIGHT
	XGL_CTX_VIEW_MODEL_DATA_TYPE	XGL_CTX_STEXT_CHAR_SLANT_ANGLE
		XGL_CTX_STEXT_CHAR_SPACING
		XGL_CTX_STEXT_CHAR_UP_VECTOR
		XGL_CTX_STEXT_COLOR
		XGL_CTX_STEXT_PATH
		XGL_CTX_STEXT_PRECISION

Strokefont Test Descriptions 27

This chapter describes the Strokefont test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

Note – The testing verification for strokefonts are based on “hit and miss” targets and as such have no logical basis for derivation. As long as text appears on the screen, we conclude any failure is due to the test’s failure to accurately calculate the expected location on the window raster.

▼ sf_font

Test Types:	INDEX, SM
Description:	Tests that *xgl_stroke_font_create* can be used to create various stroke fonts and that they can be rendered by setting the `XGL_CTX_SFONT` context attribute

Attributes Tested:	XGL_CTX_SFONT XGL_CTX_SFONT_0 XGL_CTX_STEXT_CHAR_HEIGHT XGL_CTX_STEXT_COLOR XGL_SFONT
Operators Tested:	xgl_object_create xgl_object_set xgl_stroke_text_3d
Output:	Renders the letter “X” appears in succession in the following fonts: Roman_M, Roman, Roman_D, Roman_C, Roman_T, Italic_C, Italic_T, Greek_S, Greek_C, Script_S, Script_C, Cartographic, Cartographic_M, Symbol, and Miscellaneous_M.

▼ sf_attr

Test Types:	INDEX, SM
Description:	Checks that the values of various stroke font attributes, font name, font comment, monospace flag, and default character are correct for XGL 2.0 fonts
Attributes Tested:	XGL_SFONT XGL_SFONT_COMMENT XGL_SFONT_DEFAULT_CHARACTER XGL_SFONT_IS_MONO_SPACED XGL_SFONT_NAME
Operators Tested:	xgl_object_create xgl_object_get
Output:	Nothing is displayed on the screen, as values returned by get calls are compared with what is expected.

▼ sf_ctx_attr

Test Types:	INDEX, SM
Description:	Tests the various stroke fonts’ context attributes, character height, character spacing, character expansion factor, text path, character up vector, text horizontal alignment, text vertical alignment, text precision, text color, and text slant angle.
Attributes Tested:	See Table 27-1, Column A at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_stroke_text_2d`
	`xgl_object_get`

Output:	The window raster remains the black background color.

▼ sf_dir

Test Types:	INDEX, SM
Description:	Tests that various direction vectors can be used as the fourth argument to *xgl_stroke_text_3d* to specify the plane on which texts are drawn
Attributes Tested:	`XGL_CTX_STEXT_CHAR_HEIGHT` `XGL_CTX_STEXT_COLOR`
Operators Tested:	`xgl_object_set` `xgl_stroke_text_3d`
Output:	The string "HGH" in succession (1) "HGH" appears as if it were rendered horizontally and rotated around the x-axis in the clockwise direction 180 degrees (y = 1.0 where `Y_DOWN`), (2) "HGH" appears as written (y = -1.0 makes y direction up), (3) "HGH" appears as if it were written from behind the window raster starting from left to right (x = -1), (4) "HGH" appears written on a 45-degree angle and with the individual letters flipped as if they were written going down the path instead of up, but on the other side of the window raster (x=-y), (5) "HGH" written horizontally, then rotated 180 degrees counterclockwise around the y-axis (x=-1) and then rotated 180 degrees vertically clockwise around the x-axis (y=-1) with appearances of a larger character expansion due to z=1.0, (6) "HGH" written as expected but sheared to produce a taller and narrower set of these same letters z=2.0, and (7) "HGH" written perpendicular to the window raster so only three dash marks indicate where the letters exhibit (z=-1).

sf_extent

Test Types: INDEX, SM
Description: Tests that `xgl_stroke_text_extent()` returns the correct text extent
Attributes Tested: See Table 27-1, Column B at the end of this chapter.
Operators Tested: `xgl_object_set`
`xgl_stroke_text_2d`
`xgl_stroke_text_extent`
Output: Renders the letters “Xg L *” horizontally in large letters across the window raster, the letters “X g L” on the vector representing x = -1 with the text path vertical, and “X g L *” with the text path up and on the vector representing y = 1

sf_hlhsr

Test Types: INDEX, SM
Description: Tests the hidden line-hidden surface removal of stroke font texts
Attributes Tested: See Table 27-2, Column A at the end of this chapter.
Operators Tested: `xgl_object_set`
`xgl_stroke_text_3d`
Output: Renders the letters “X G L” horizontally the “normal” way and then rotated around the x-axis 180 degrees at the same location, but first with character spacing of 0.0 and then 1.0 so that only 1 “X” exists but two “G”s and two “L”s exist with the “G” of the second string coincident with the “L” of the first string

sf_ctx_attr2

Test Types: INDEX, SM
Description: RGB version of *sf_ctx_attr*
Attributes Tested: See Table 27-1, Column A at the end of this chapter.
Operators Tested: `xgl_object_set`
`xgl_stroke_text_2d`
`xgl_object_get`
`xgl_object_set`

Output: Varies character height and displays “XGL” in succession at location 100,300 with character height 5.0, 10.0, 20.5, 53.4, 100.0, and 200.0 (“L” partially clipped by right window boundary).

Varies character spacing and displays “XGL” in succession at location 250,300 with character spacing -4.3 (text appears to be written from right to left), -2.6, -1.0 (characters written on top of one another), -0.5 (characters written snuggled inside each other’s curvature), 0.0 (default), 0.5, 1.0, 2.6, and 4.3.

Varies character expansion and displays “XGL” in succession at location 400,300 with character spacing -1.0 (characters written on top of one another), -0.7, (characters expanded to fill up their location and half of their neighbors), 0.2 (characters sheared to narrow height with larger spaces between them), 0.5, 1.0 (default), 1.5 (characters expanded as though sheared to wider width), and 2.0.

Varies character text path and displays “XGL” in succession at location 250,250 with text path right (seen rotated around x-axis by 180 degrees because character up vector is y=-1.0), left (seen written from left to right and then 180-degree rotation around x-axis because character up vector is y=-1.0), up (seen written as if down and then rotated around 180-degrees from the x-axis), and down (seen written vertically down then rotated 180-degrees around the x-axis because the up vector is y=-1.0).

Varies character up vector and displays “XGL” in succession at location 250,250 with character up vector (0.0,-0.1)(text upside down from left to right), (0.0, 1.0) displayed normally, (-1.0,-1.0) displayed along the outside of the 135-degree angle, and (1.0,1.0) displayed along the inside of the 135-degree angle.

Varies character horizontal alignment and displays “XGL” in succession at location 250,250 with horizontal alignment left, center, right, and normal.

Varies character horizontal alignment normal and vertical alignment and “LOCAL TEXT” is seen in succession at location 250,250 with vertical alignment top, cap, half, base, bottom, and normal.

Varies character text precision with all other variables normal except for height, which is set to 150.0, and character up (0.0,-0.1). Displays “B G L” in succession at location 200,250. The text appears as though it was written normally but rotated around the x-axis for 180 degrees due to the character up vector. Precision is varied through stroke, character, and string. Uses the same set up for “BGL” except the precision is set at stroke and text color is varied to red, yellow, and purple.
Varies character text color and the same string appears in red, yellow, and purple.
Varies character slant angle while everything else is normal except for height of 25.0 for text “XGX” at position 250,250. Angle at -89 and 89 degrees appears as a jagged horizontal line. Angle -60 and 60 degrees slants to right and left respectively. Angle 0 is normal position and +30 and -30 slant appropriately from this norm.

▼ sf_dir2

Test Types:	RGB, SM
Description:	RGB version of *sf_dir*
Attributes Tested:	`XGL_CTX_STEXT_COLOR` `XGL_CTX_STEXT_CHAR_HEIGHT`
Operators Tested:	`xgl_object_set` `xgl_stroke_text_3d`
Output:	Same as *sf_dir*

▼ sf_extent2

Test Types:	RGB, SM
Description:	RGB version of *sf_extent*
Attributes Tested:	See Table 27-1, Column B at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_stroke_text_2d`
	`xgl_stroke_text_extent`

Output:	Same as *sf_extent*

▼ sf_font2

Test Types: RGB, SM
Description: RGB version of *sf_font*
Attributes Tested: XGL_CTX_SFONT
XGL_CTX_SFONT_0
XGL_CTX_STEXT_CHAR_HEIGHT
XGL_CTX_STEXT_COLOR
XGL_CTX_VIEW_TRANS
XGL_RAS_DEPTH
XGL_SFONT
XGL_TRANS_PRECONCAT
XGL_TRANS_REPLACE
Operators Tested: xgl_object_get
xgl_transform_translate
xgl_transform_scale
xgl_object_set
xgl_object_create
xgl_stroke_text_2d
Output: Same as *sf_font*

▼ sf_hlhsr2

Test Types: RGB, SM
Description: RGB version of *sf_hlhsr*
Attributes Tested: See Table 27-2, Column A at the end of this chapter.
Operators Tested: xgl_object_set
xgl_stroke_text_3d
Output: Same as *sf_hlhsr*

▼ sf_ctx_attr3

Test Types: RGB, SM
Description: 3D version of *sf_ctx_attr2*
Attributes Tested: See Table 27-1, Column A at the end of this chapter.
Operators Tested: xgl_object_set
xgl_stroke_text_3d
xgl_object_get
Output: Same as *sf_ctx_attr2*

sf_extent3

Test Types: RGB, SM
Description: 3D version of *sf_extent2*
Attributes Tested: See Table 27-1, Column B at the end of this chapter.
Operators Tested: `xgl_object_set`
`xgl_stroke_text_extent`
`xgl_stroke_text_3d`
Output: The text string "X g L" with the directional vector in `Y_DOWN` so that the text is rotated 180 degrees around the x-axis in lime green. The same text string is displayed on the outside edge of the 135-degree angle as if it were actually written on the inside of the edge and rotated 180 degrees around this vector. Renders the same string vertically and then rotates each character 180 degrees in its place.

sf0

Test Types: RGB, SM
Description: Checks a 20 by 20 area for the expected location of RGB text "X" which has been translated and scaled to another location by virtue of the view transformation and rendered in 256 available colors
Attributes Tested: `XGL_CTX_SFONT_0`
`XGL_CTX_STEXT_CHAR_HEIGHT`
`XGL_CTX_STEXT_COLOR`
`XGL_CTX_VIEW_TRANS`
`XGL_RAS_DEPTH`
`XGL_SFONT`
`XGL_TRANS_PRECONCAT`
`XGL_TRANS_REPLACE`
Operators Tested: `xgl_object_get`
`xgl_transform_translate`
`xgl_transform_scale`
`xgl_object_create`
`xgl_stroke_text_2d`
Output: Stroke text "X" with all many colors (all colors in color for 8-bit raster and 256 random colors otherwise)

sf2

Test Types: RGB, SM

Description: Tests all four character sets by drawing stroke texts using different character sets and checking that the right font is used

Attributes Tested:
`XGL_CTX_SFONT_0`
`XGL_CTX_SFONT_1`
`XGL_CTX_SFONT_2`
`XGL_CTX_SFONT_3`
`XGL_CTX_STEXT_CHAR_HEIGHT`
`XGL_CTX_STEXT_COLOR`
`XGL_RAS_DEPTH`
`XGL_SFONT`

Operators Tested:
`xgl_object_get`
`xgl_object_set`
`xgl_object_create`
`xgl_stroke_text_2d`

Output: Writes the letter “X” in succession using the character sets Roman_C, Italic_C, Greek_C, and Miscellaneous_M

sf4

Test Types: RGB, SM

Description: Tests linear depth-cueing of stroke text: vary text color, depth-cueing color, text direction, initial Z value, and Z slope (a line of identical characters with constantly changing depth). Checks for the right color at the center of each character.

Attributes Tested: See Table 27-2, Column B at the end of this chapter.

Operators Tested:
`xgl_object_set`
`xgl_object_get`
`xgl_stroke_text_3d`
`xgl_stroke_text_extent`

Output: Renders the text string “XXXXXXXXXX” upside down (`Y_DOWN`) in succession in the combination of colors produced by the mixture of text color, which stays at a fixed color while looping through the depth cue colors. The text colors are white, purple, and two shades of blue, while the depth colors are black, green, and red.

sf5

Test Types: INDEX, SM
Description: Indexed version of *sf4*
Attributes Tested: See Table 27-2, Column B at the end of this chapter.
Operators Tested: `xgl_object_set`
`xgl_object_get`
`xgl_stroke_text_3d`
`xgl_stroke_text_extent`
Output: Renders the text string “XXXXXXXXXX” upside down (`Y_DOWN`) in succession in the shaded combination of colors produced by the mixture of text color, which varies, and the depth cue color which is the index at the bottom of the color ramp yellow. The depth cue color is visible only when the z value for both the directional vector and the text position are close to the z maximum boundary value.

sf_extent4

Test Types: INDEX, SM
Description: Tests that `xgl_stroke_text_extent()` uses `XGL_CTX_SFONT_`* attributes, not `XGL_CTX_ANNOT_`* attributes, when calculating the text extent.
Attributes Tested: See Table 27-2, Column C at the end of this chapter.
Operators Tested: `xgl_object_set`
`xgl_stroke_text_extent`
`xgl_stroke_text_2d`
Output: Writes the text string “X g L” in succession varying the text position, the character up vector, and the character spacing and expansion, while other various unapplicable annotation attributes are set

sf_extent5

Test Types: INDEX, SM
Description: Sets `XGL_CTX_SFONT_0` to different fonts and checks that `xgl_stroke_text_extent()` returns the correct text extent
Attributes Tested: See Table 27-1, Column B at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_stroke_text_extent`
	`xgl_stroke_text_2d`
	`xgl_object_create`

Output:	Writes the letters “Xg L” in succession in Roman_c.phont, Italic_C.phont and Greek_C.phont with these characteristics: (1) normal appearance, (2) rendered along the line x = -y, and (3) rendered with the text path up and y=1 so the text is upside down in each position.

▼ sf_extent6

Test Types:	INDEX, SM
Description:	Tests that `xgl_stroke_text_extent()` returns the correct text extent for monoencoded strings
Attributes Tested:	See Table 27-1, Column B at the end of this chapter.

Operators Tested:	`xgl_object_set`
	`xgl_stroke_text_extent`
	`xgl_stroke_text_2d`
	`xgl_object_create`

Output:	Writes the letters “Xg L” in Roman_C.phont; the same text in Italic_C.phont along the line x = -y; and the same text in Greek_C.phont with the text path up and y=1 so the text is upside down in each position. Renders text concatenated with two different fonts and strings, “X g L” in Italic_C.phont and “Sun” in Greek_C.phont with the text path up and y=1 so the text is upside down in each position. Renders three different strings and three different fonts: string “Xg L” in Roman_C.phont concatenated with “Sun” in Italic_C.phont and concatenated with “text” in Greek_C.phont. The concatenated string is written along the line x = -y with the text path left.

▼ sf_plane_mask

Test Types:	INDEX, SM
Description:	Tests that the `XGL_CTX_PLANE_MASK` attribute applies to stroke font text

Attributes Tested:	XGL_CTX_PLANE_MASK XGL_CTX_STEXT_CHAR_HEIGHT XGL_CTX_STEXT_COLOR XGL_RAS_DEPTH
Operators Tested:	xgl_object_get xgl_object_set xgl_stroke_text
Output:	Renders the text "XGL" and expects the plane masked color to be 0xff ^ i (loop value and actual bits set) as the expected color for 256 loops.

▼ sf_ras_op

Test Types:	INDEX, SM
Description:	Tests that the XGL_CTX_ROP attribute applies to stroke font text
Attributes Tested:	See Table 27-1, Column C at the end of this chapter.
Operators Tested:	xgl_object_set xgl_stroke_text
Output:	Sixteen successions of the text "XGL" in the colors black, white, black, green, green, red, red, blue, blue, light green, light green, light blue, light blue, light red, light red, and white

▼ sf_mono_ctx_attr

Test Types:	INDEX, SM
Description:	Tests the various stroke fonts context attributes in indexed color mode using monoencoded strings
Attributes Tested:	See Table 27-1, Column A at the end of this chapter.
Operators Tested:	xgl_object_get xgl_object_set xgl_object_create xgl_stroke_text_2d
Output:	The window raster remains the white background color.

▼ sf_mono_ctx_attr2

Test Types:	RGB, SM

Description:	Tests the various stroke fonts context attributes in RGB mode using monoencoded strings
Attributes Tested:	See Table 27-1, Column A at the end of this chapter.

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_stroke_text_2d`

Output:	Same as *sf_ctx_attr2*

▼ sf_mono_ctx_attr3

Test Types:	RGB, SM
Description:	Tests the various stroke fonts context attributes in 3D RGB mode using monoencoded strings.
Attributes Tested:	See Table 27-1, Column A at the end of this chapter.

Operators Tested:	`xgl_object_get`
	`xgl_object_create`
	`xgl_object_set`
	`xgl_stroke_text_3d`

Output:	Same as *sf_mono_ctx_attr2* because directional vector places text as normally read

▼ sf_mono_ctx_attr4

Test Types:	RGB, SM
Description:	Tests the various stroke fonts context attributes in indexed color mode using monoencoded strings and `XGL_CHAR_ISO` character encoding
Attributes Tested:	See Table 27-1, Column A at the end of this chapter.

Operators Tested:	`xgl_object_get`
	`xgl_object_create`
	`xgl_stroke_text_2d`

Output:	Same as *sf_mono_ctx_attr2*

▼ sf_mono_ctx_attr5

Test Types:	RGB, SM
Description:	Tests the various stroke fonts context attributes in 3D RGB mode using monoencoded strings and `XGL_CHAR_ISO` character encoding.

Attributes Tested:	See Table 27-1, Column A at the end of this chapter.

Operators Tested:	`xgl_object_get`
	`xgl_object_create`
	`xgl_stroke_text_3d`

Output:	Same as *sf_mono_ctx_attr2*

▼ sf_mono_hlhsr

Test Types:	INDEX, SM
Description:	Tests the hidden line-hidden surface removal of indexed color stroke font texts using monoencoded strings

Attributes Tested:	`XGL_3D_CTX_HLHSR_MODE`
	`XGL_3D_CTX_SURF_FRONT_ILLUMINATION`
	`XGL_CHAR_ISO`
	`XGL_MULTI_STR`
	`XGL_ILLUM_NONE`
	`XGL_CTX_NEW_FRAME_ACTION`
	`XGL_SFONT`
	`XGL_CTX_NEW_FRAME_CLEAR`
	`XGL_CTX_NEW_FRAME_HLHSR_ACTION`
	`XGL_CTX_STEXT_CHAR_ENCODING`
	`XGL_CTX_STEXT_CHAR_HEIGHT`
	`XGL_CTX_STEXT_CHAR_SPACING`
	`XGL_CTX_STEXT_COLOR`
	`XGL_HLHSR_Z_BUFFER,`

Operators Tested:	`xgl_object_set`
	`xgl_object_create`
	`xgl_stroke_text_3d`

Output:	The letters “X G L” with “X” from Roman_C and “GL” from Italic_C written horizontally upside down due to directional vectors y= 0 (`Y_DOWN`), but with the first text produced with character spacing of 0.0 and then 1.0. The appearance is such that only one “X” exists but two “G”s and two “L”s exist with the “G” of the second string coincident with the “L” of the first string.

▼ sf_mono_hlhsr2

Test Types:	RGB, SM

Description: Tests the hidden line-hidden surface removal of RGB stroke font texts using monoencoded strings

Attributes Tested:
XGL_3D_CTX_HLHSR_MODE
XGL_3D_CTX_SURF_FRONT_ILLUMINATION
XGL_CHAR_ISO
XGL_SFONT
XGL_CTX_NEW_FRAME_ACTION
XGL_ILLUM_NONE
XGL_CTX_NEW_FRAME_CLEAR
XGL_CTX_NEW_FRAME_HLHSR_ACTION
XGL_CTX_STEXT_CHAR_ENCODING
XGL_CTX_STEXT_CHAR_HEIGHT
XGL_MULTI_STR
XGL_CTX_STEXT_CHAR_SPACING
XGL_CTX_STEXT_COLOR
XGL_HLHSR_Z_BUFFER

Operators Tested:
xgl_object_set
xgl_object_create
xgl_stroke_text_3d

Output: Same as *sf_mono_hlhsr*

▼ at0

Test Types: RGB, SM

Description: Tests the various annotation text context attributes in 2D RGB mode

Attributes Tested: See Table 27-1, Column A at the end of this chapter.

Operators Tested:
xgl_object_get
xgl_object_set
xgl_annotation_text

Output: Same as *sf_ctx_attr2*

▼ at1

Test Types: RGB, SM

Description: 3D version of *at0*

Attributes Tested: See Table 27-1, Column A at the end of this chapter.

Operators Tested:
xgl_object_get
xgl_object_set
xgl_annotation_text

Output: Same as *sf_ctx_attr2*

at2

Test Types: RGB, SM

Description: Tests XGL_CTX_ATEXT_STYLE: no line drawn for normal style and line drawn for line style. Renders in succession the text "" with both a nonzero text position and annotation position while changing the ANNOT_STYLE from expecting no leader line to leader line.

Attributes Tested: XGL_ATEXT_STYLE_LINE
XGL_ATEXT_STYLE_NORMAL
XGL_CTX_ATEXT_STYLE
XGL_CTX_BACKGROUND_COLOR
XGL_CTX_LINE_COLOR

Operators Tested: xgl_object_get
xgl_object_set
xgl_annotation_text

Output: Renders eight green lines from text position to text position plus annotation position

at3

Test Types: RGB, SM

Description: 3D version of *at2*

Attributes Tested: XGL_ATEXT_STYLE_LINE
XGL_ATEXT_STYLE_NORMAL
XGL_CTX_ATEXT_CHAR_HEIGHT
XGL_CTX_ATEXT_STYLE
XGL_CTX_BACKGROUND_COLOR
XGL_CTX_LINE_COLOR
XGL_CTX_STEXT_COLOR

Operators Tested: xgl_object_get
xgl_object_set
xgl_annotation_text

Output: Renders eight green lines from text position to text position plus annotation position

at6

Test Types: RGB, INDEX

Description: Tests annotation text HLHSR. Draws two strings at the same position but with different depth and checks that the front one shows up; tries various combinations of depth. The annotation position is always held at 0.

Attributes Tested:
XGL_3D_CTX_HLHSR_MODE
XGL_CTX_ATEXT_CHAR_HEIGHT
XGL_CTX_NEW_FRAME_ACTION
XGL_CTX_NEW_FRAME_CLEAR
XGL_CTX_NEW_FRAME_HLHSR_ACTION
XGL_CTX_STEXT_CHAR_SPACING
XGL_CTX_STEXT_COLOR
XGL_HLHSR_Z_BUFFER

Operators Tested:
xgl_object_set
xgl_annotation_text
xgl_object_get

Output: Draws the letters “X G L” in the default font Roman_M written twice at the same location, but first with character spacing of 0.0 and then 1.0. The appearance is such that only one “X” exists, but two “G”s and two “L”s exist with the “G” of the second string coincident with the “L” of the first string. The z value for the text positions are varied and the smaller z value governs which color the coincident “X” appears as: (1) red, (2) green, and (3) green.

at7

Test Types: RGB, SM

Description: Tests linear depth-cueing of annotation text: vary text color, depth-cueing color and text depth. Checks that the resulting character has the right color.

Attributes Tested:
XGL_3D_CTX_DEPTH_CUE_COLOR
XGL_3D_CTX_DEPTH_CUE_MODE
XGL_CTX_ATEXT_CHAR_HEIGHT
XGL_CTX_STEXT_COLOR
XGL_CTX_VDC_MAP
XGL_CTX_VDC_WINDOW

	XGL_DEPTH_CUE_LINEAR
	XGL_RAS_DEPTH
	XGL_RAS_HEIGHT
	XGL_RAS_WIDTH
	XGL_VDC_MAP_ALL
Operators Tested:	xgl_object_get
	xgl_object_set
	xgl_annotation_text
Output:	Renders the text string “X” in succession in the combination of colors produced by the mixture of text color, which stays at a fixed color while looping through the depth cue colors. The text colors are white, cyan, red, and two shades of blue while the depth colors are gray, yellow, green, and white.

▼ at8

Test Types:	INDEX, SM
Description:	Indexed version of *at7*
Attributes Tested:	XGL_3D_CTX_DEPTH_CUE_MODE
	XGL_CTX_ATEXT_CHAR_HEIGHT
	XGL_CTX_STEXT_COLOR
	XGL_CTX_VDC_MAP
	XGL_CTX_VDC_WINDOW
	XGL_DEPTH_CUE_LINEAR
	XGL_RAS_HEIGHT
	XGL_RAS_WIDTH
	XGL_VDC_MAP_ALL
Operators Tested:	xgl_object_get
	xgl_object_set
	xgl_annotation_text
Output:	Renders a succession of “X” in the upper-left corner in black, light blue, purple, light red, light purple, blue, and gray

▼ at9

Test Types:	RGB, SM

Description:	Tests scaled depth-cueing of annotation text: vary text color, depth-cueing color, reference planes, scales, and text depth. Checks that the resulting character has the right color.
Attributes Tested:	See Table 27-2, Column B at the end of this chapter.

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_annotation_text`
	`xgl_stroke_text_extent`

Output:	Renders a succession of “X” in the upper-left corner in gray, white, blue, and purple

▼ at10

Test Types:	RGB, SM
Description:	Tests that 2D RGB annotation texts can be drawn with all four character sets

Attributes Tested:	`XGL_CTX_ATEXT_CHAR_HEIGHT`
	`XGL_CTX_SFONT_0`
	`XGL_CTX_SFONT_1`
	`XGL_CTX_SFONT_2`
	`XGL_CTX_SFONT_3`
	`XGL_CTX_STEXT_COLOR`
	`XGL_RAS_DEPTH,XGL_SFONT`

Operators Tested:	`xgl_object_get`
	`xgl_annotation_text`
	`xgl_object_set`
	`xgl_object_create`

Output:	Writes the letter “X” in succession using four character sets in Roman_C, Italic_C, Greek_C, and Miscellaneous_M

▼ at11

Test Types:	INDEX, SM
Description:	Tests the hidden line-hidden surface removal of indexed color annotation texts
Attributes Tested:	See Table 27-2, Column A at the end of this chapter.
Operators Tested:	`xgl_object_set` `xgl_annotation_text`

Output: Renders the letters “X G L” with fonts from default Roman_M twice, once with character spacing of 0.0 and then 1.0. The appearance is such that only one “X” exists but two “G”s and two “L”s exist with the “G” of the second string coincident with the “L” of the first string. The z value for the text positions are varied and the smaller z value governs which color the coincident “X” appears as: (1) red, (2) green, and (3) green.

at_plane_mask

Test Types:	INDEX, SM
Description:	Tests that the `XGL_CTX_PLANE_MASK` attribute applies to annotation text

Attributes Tested:	`XGL_CTX_ATEXT_CHAR_HEIGHT`
	`XGL_CTX_PLANE_MASK`
	`XGL_CTX_STEXT_COLOR`
	`XGL_RAS_DEPTH`

Operators Tested:	`xgl_object_get`
	`xgl_annotation_text`
	`xgl_object_set`

Output:	Renders the text “XGL” and expects the plane masked color to be 0xff ^ i (loop value and actual bits set) as the expected color for 256 loops.

at_ras_op

Test Types:	INDEX, SM
Description:	Tests that the `XGL_CTX_ROP` attribute applies to annotation text
Attributes Tested:	See Table 27-1, Column C at the end of this chapter.

Operators Tested:	`xgl_object_get`
	`xgl_annotation_text`
	`xgl_object_set`

Output:	Sixteen successions of the text “XGL” in the colors black, white, black, green, green, red, red, blue, blue, light green, light green, light blue, light blue, light red, light red, and white.

▼ at_mono_ctx_attr

Test Types: INDEX, SM

Description: Tests the various annotation text context attributes in 2D indexed color mode using monoencoded strings

Attributes Tested: See Table 27-1, Column A at the end of this chapter.

Operators Tested:
`xgl_object_get`
`xgl_annotation_text`
`xgl_object_set`
`xgl_object_create`

Output: Same as *sf_ctx_attr2*

▼ at_mono_ctx_attr2

Test Types: RGB, INDEX

Description: Tests the various annotation text context attributes in 2D RGB mode using monoencoded strings

Attributes Tested: See Table 27-1, Column A at the end of this chapter.

Operators Tested:
`xgl_object_get`
`xgl_annotation_text`
`xgl_object_set`
`xgl_object_create`

Output: Same as *sf_ctx_attr2*

▼ at_mono_ctx_attr3

Test Types: RGB, INDEX

Description: Tests the various annotation text attributes in 3D RGB mode using monoencoded strings

Attributes Tested: See Table 27-1, Column A at the end of this chapter.

Operators Tested:
`xgl_object_get`
`xgl_annotation_text`
`xgl_object_set`
`xgl_object_create`

Output: Same as *sf_ctx_attr2*

▼ at_mono_ctx_attr4

Test Types: INDEX, SM

Description:	Tests the various annotation text context attributes in 2D indexed color mode using monoencoded strings and XGL_CHAR_ISO character encoding
Attributes Tested:	See Table 27-1, Column A at the end of this chapter.
Operators Tested:	xgl_object_get xgl_annotation_text xgl_object_set xgl_object_create
Output:	Same as *sf_ctx_attr2*

▼ at_mono_ctx_attr5

Test Types:	RGB, SM
Description:	Tests the various annotation text attributes in 3D RGB mode using monoencoded strings and XGL_CHAR_ISO character encoding
Attributes Tested:	See Table 27-1, Column A at the end of this chapter.
Operators Tested:	xgl_object_get xgl_annotation_text xgl_object_set xgl_object_create
Output:	Same as *at_mono_ctx_attr2*

▼ at_mono_hlhsr2

Test Types:	RGB, SM
Description:	Tests the hidden line-hidden surface removal of RGB annotation text using monoencoded strings
Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_CHAR_ISO XGL_CTX_ATEXT_CHAR_HEIGHT XGL_CTX_NEW_FRAME_ACTION XGL_CTX_NEW_FRAME_CLEAR XGL_CTX_NEW_FRAME_HLHSR_ACTION XGL_CTX_STEXT_CHAR_ENCODING XGL_CTX_STEXT_CHAR_SPACING XGL_CTX_STEXT_COLOR XGL_HLHSR_Z_BUFFER XGL_MULTI_STR XGL_SFONT

Operators Tested: `xgl_annotation_text`
`xgl_object_set`

Output: Writes the letters “X G L” horizontally the “normal” way but first with character spacing of 0.0 and then 1.0 so that only one “X” exists but two “G”s and two “L”s exist with the “G” of the second string coincident with the “L” of the first string.

▼ gc_sf2

Test Types: RGB, SM

Description: Checks all four character sets (2D RGB) with gcached strokefont

Attributes Tested: `XGL_CTX_SFONT_0`
`XGL_CTX_SFONT_1`
`XGL_CTX_SFONT_2`
`XGL_CTX_SFONT_3`
`XGL_CTX_STEXT_CHAR_HEIGHT`
`XGL_CTX_STEXT_COLOR`
`XGL_GCACHE`
`XGL_RAS_DEPTH`
`XGL_SFONT`

Operators Tested: `xgl_object_create`
`xgl_object_get`
`xgl_gcache_stroke_text`
`xgl_context_display_gcache`
`xgl_object_destroy`

Output: Writes the letter “X” in succession using character sets Roman_C, Italic_C, Greek_C, and Miscellaneous_M

▼ gc_sf3

Test Types: RGB, CM

Description: Tests 3D character sets

Attributes Tested: `XGL_CTX_SFONT_0`
`XGL_CTX_SFONT_1`
`XGL_CTX_SFONT_2`
`XGL_CTX_SFONT_3`
`XGL_CTX_STEXT_CHAR_HEIGHT`
`XGL_CTX_STEXT_COLOR`

	XGL_GCACHE
	XGL_RAS_DEPTH
	XGL_SFONT
Operators Tested:	xgl_object_create
	xgl_object_get
	xgl_gcache_stroke_text
	xgl_context_display_gcache
	xgl_object_destroy
Output:	Writes the letter “X” in succession using character sets Roman_C, Italic_C, Greek_C, and Miscellaneous_M

Table 27-1 Strokefont Attributes Tested - Set 1

Column A	**Column B**	**Column C**
XGL_CTX_STEXT_ALIGN_VERT	XGL_CTX_LINE_COLOR	XGL_CTX_ROP
XGL_CTX_STEXT_CHAR_ EXPANSION_FACTOR	XGL_CTX_STEXT_ALIGN_HORIZ	XGL_CTX_STEXT_CHAR_HEIGHT
XGL_CTX_STEXT_CHAR_HEIGHT	XGL_CTX_STEXT_ALIGN_VERT	XGL_CTX_STEXT_COLOR
XGL_CTX_STEXT_CHAR_SLANT_ ANGLE	XGL_CTX_STEXT_CHAR_ EXPANSION_FACTOR	XGL_RAS_DEPTH
XGL_CTX_STEXT_CHAR_SPACING	XGL_CTX_STEXT_CHAR_HEIGHT	XGL_ROP_AND
XGL_CTX_STEXT_CHAR_UP_ VECTOR	XGL_CTX_STEXT_CHAR_SPACING	XGL_ROP_AND_INVERTED
XGL_CTX_STEXT_COLOR	XGL_CTX_STEXT_CHAR_UP_ VECTOR	XGL_ROP_AND_REVERSE
XGL_CTX_STEXT_PATH	XGL_CTX_STEXT_PATH	XGL_ROP_CLEAR
XGL_CTX_STEXT_PRECISION	XGL_STEXT_PATH_UP	XGL_ROP_COPY
XGL_STEXT_ALIGN_HORIZ_ CENTER	XGL_STEXT_ALIGN_HORIZ_ CENTER	XGL_ROP_COPY_INVERTED
XGL_STEXT_ALIGN_HORIZ_ NORMAL	XGL_STEXT_ALIGN_VERT_BOTTO M	XGL_ROP_EQUIV
XGL_STEXT_ALIGN_HORIZ_RIGHT	XGL_STEXT_ALIGN_VERT_HALF	XGL_ROP_INVERT
XGL_STEXT_ALIGN_VERT_BASE		XGL_ROP_NAND
XGL_STEXT_ALIGN_VERT_BOTTO M		XGL_ROP_NOOP
XGL_STEXT_ALIGN_VERT_CAP		XGL_ROP_NOR
XGL_STEXT_ALIGN_VERT_HALF		XGL_ROP_OR
XGL_STEXT_ALIGN_VERT_ NORMAL		XGL_ROP_OR_INVERTED
XGL_STEXT_ALIGN_VERT_TOP		XGL_ROP_OR_REVERSE
XGL_STEXT_PATH_DOWN		XGL_ROP_SET
XGL_STEXT_PATH_LEFT		XGL_ROP_XOR
XGL_STEXT_PATH_RIGHT		

Table 27-1 Strokefont Attributes Tested - Set 1 *(Continued)*

Column A	Column B	Column C
XGL_STEXT_PATH_UP		
XGL_STEXT_PRECISION_CHAR		
XGL_STEXT_PRECISION_STRING		
XGL_STEXT_PRECISION_STROKE		

Table 27-2 Strokefont Attributes Tested - Set 2

Column A	Column B	Column C
XGL_3D_CTX_HLHSR_MODE	XGL_3D_CTX_DEPTH_CUE_COLOR	XGL_CTX_ATEXT_ALIGN_HORIZ
XGL_3D_CTX_SURF_FRONT_ ILLUMINATION	XGL_3D_CTX_DEPTH_CUE_MODE	XGL_CTX_ATEXT_ALIGN_VERT
XGL_CTX_NEW_FRAME_ACTION	XGL_CTX_STEXT_CHAR_HEIGHT	XGL_CTX_ATEXT_CHAR_HEIGHT
XGL_CTX_NEW_FRAME_CLEAR	XGL_CTX_STEXT_COLOR	XGL_CTX_ATEXT_CHAR_UP_ VECTOR
XGL_CTX_NEW_FRAME_HLHSR_ ACTION	XGL_CTX_VDC_MAP	XGL_CTX_ATEXT_PATH
XGL_CTX_STEXT_CHAR_HEIGHT	XGL_CTX_VDC_WINDOW	XGL_CTX_LINE_COLOR
XGL_CTX_STEXT_CHAR_SPACING	XGL_DEPTH_CUE_LINEAR	XGL_CTX_STEXT_CHAR_ EXPANSION_FACTOR
XGL_CTX_STEXT_COLOR	XGL_RAS_HEIGHT	XGL_CTX_STEXT_CHAR_SPACING
XGL_HLHSR_Z_BUFFER	XGL_RAS_WIDTH	XGL_STEXT_ALIGN_HORIZ_ CENTER
XGL_ILLUM_NONE	XGL_VDC_MAP_ASPECT	XGL_STEXT_ALIGN_VERT_BOTTO M
		XGL_STEXT_ALIGN_VERT_HALF
		XGL_STEXT_PATH_UP

System Test Descriptions 28

This chapter describes the System test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ sys_open

Test Types:	SM
Description:	Tests that `xgl_open()` creates and initializes the system state object. Also checks for memory leak problems by calling `xgl_open()` and `xgl_close()` many times.
Attributes Tested:	`XGL_SYS_ST_ERROR_DETECTION`

Operators Tested:	`xgl_open`
	`xgl_close`
	`xgl_object_get`

Output:	None

sys_attr

Test Types: SM
Description: Verifies the default values of the various system state attributes and tries setting them to some other nondefault values. Also sets the XGL_SYS_ST_ERROR_NOTIFICATION_FUNCTION to a user_defined function, and checks that it is indeed invoked when errors occur.
Attributes Tested: XGL_SYS_ST_ERROR_DETECTION
XGL_SYS_ST_ERROR_NOTIFICATION_FUNCTION
XGL_SYS_ST_VERSION
Operators Tested: xgl_object_get
xgl_object_set
Output: None

sys_destroy

Test Types: SM, INDEX
Description: Tests that xgl_object_destroy() destroys the specified object by freeing all resources associated with the object
Attributes Tested: XGL_CMAP_COLOR_TABLE_SIZE
XGL_CMAP_COLOR_TABLE
XGL_DEV_COLOR_TYPE
XGL_DEV_COLOR_MAP
XGL_CTX_DEVICE
Operators Tested: xgl_object_create
xgl_object_destroy
Output: None

sys_create

Test Types: SM
Description: Tests that xgl_object_create() can create all types of objects and creates the right type of objects
Attributes Tested: XGL_OBJ_TYPE
Operators Tested: xgl_object_create
xgl_object_get
Output: None

▼ sys_inquire

Test Types:	SM
Description:	Tests that `xgl_inquire()` returns reasonable values
Attributes Tested:	`XGL_SYS_ST_ERROR_DETECTION`
Operators Tested:	`xgl_inquire`
Output:	None

▼ sys_obj

Test Types:	SM
Description:	Tests that `XGL_OBJ_TYPE` of all types of objects are right and that `XGL_OBJ_APPLICATION_DATA` works correctly
Attributes Tested:	`XGL_OBJ_TYPE` `XGL_OBJ_APPLICATION_DATA`

Operators Tested:	`xgl_object_create`
	`xgl_object_get`
	`xgl_object_set`

Output:	None

Texture Mapping Test Descriptions 29

This chapter describes the Texture Mapping test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section, “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ texture_mipmap

Test Types:	SM, RGB
Description:	Tests texture mipmap creation
Attributes Tested:	XGL_MIPMAP_TEXTURE
	XGL_DMAP_TEXTURE
Operators Tested:	xgl_mipmap_texture_build
Output:	Checker pattern textured polygon

▼ texture_mipmap_1

Test Types:	SM, RGB

Description:	Tests texture operation
Attributes Tested:	XGL_MIPMAP_TEXTURE XGL_DMAP_TEXTURE XGL_TEXTURE_OP_* XGL_TEXTURE_BOUNDARY_WRAP
Operators Tested:	xgl_mipmap_texture_build xgl_object_set
Output:	Checker pattern textured polygon with BOUNDARY_WRAP

texture_mipmap_2

Test Types:	SM, RGB
Description:	Tests texture operation
Attributes Tested:	XGL_MIPMAP_TEXTURE XGL_DMAP_TEXTURE XGL_TEXTURE_OP_* XGL_TEXTURE_BOUNDARY_CLAMP
Operators Tested:	xgl_mipmap_texture_build xgl_object_set
Output:	Checker pattern textured quadmesh with BOUNDARY_CLAMP

texture_mipmap_3

Test Types:	SM, RGB
Description:	Tests texture operation
Attributes Tested:	XGL_MIPMAP_TEXTURE XGL_DMAP_TEXTURE XGL_TEXTURE_OP_* XGL_TEXTURE_BOUNDARY_TRANSPARENT
Operators Tested:	xgl_mipmap_texture_build xgl_object_set
Output:	Checker pattern textured quadmesh with BOUNDARY_TRANSPARENT

texture_mipmap_4

Test Types:	SM, RGB
Description:	Tests texture operation

Attributes Tested:	XGL_MIPMAP_TEXTURE XGL_DMAP_TEXTURE XGL_TEXTURE_OP_* XGL_TEXTURE_BOUNDARY_TRANSPARENT XGL_TEXTURE_BOUNDARY_CLAMP
Operators Tested:	xgl_mipmap_texture_build xgl_object_set
Output:	Checker pattern textured quadmesh with both BOUNDARY_TRANSPARENT and BOUNDARY_CLAMP

▼ texture_mipmap_5

Test Types:	CM, RGB
Description:	Tests texture operation
Attributes Tested:	XGL_TEXTURE_INTERP_MIPMAP_TRILINEAR XGL_TEXTURE_INTERP_POINT
Operators Tested:	xgl_mipmap_texture_build xgl_object_set
Output:	Textured polygon or quadmesh

▼ texture_mipmap_6

Test Types:	CM, RGB
Description:	Tests texture operation
Attributes Tested:	XGL_RENDER_COMP_REFLECTED_COLOR XGL_TEXTURE_OP_DECAL
Operators Tested:	xgl_mipmap_texture_build xgl_object_set
Output:	Textured polygon or quadmesh

▼ texture_tmap_op_1

Test Types:	SM, RGB
Description:	Tests texture operation
Attributes Tested:	XGL_TMAP_DESCRIPTOR XGL_3D_CTX_SURF_FRONT_TMAP_SWITCHES XGL_TEXTURE_OP_* XGL_TEXTURE_BOUNDARY_WRAP

Operators Tested:	xgl_mipmap_texture_build xgl_object_set
Output:	Checker pattern textured polygon with BOUNDARY_WRAP

▼ texture_tmap_op_2

Test Types:	SM, RGB
Description:	Tests texture operation
Attributes Tested:	XGL_TMAP_DESCRIPTOR XGL_TEXTURE_OP_* XGL_TEXTURE_BOUNDARY_MIRROR
Operators Tested:	xgl_mipmap_texture_build xgl_object_set
Output:	Checker pattern textured quadmesh with BOUNDARY_MIRROR

▼ texture_tmap_op_3

Test Types:	SM, RGB
Description:	Tests texture operation
Attributes Tested:	XGL_TEXTURE_OP_* XGL_TEXTURE_BOUNDARY_TRANSPARENT
Operators Tested:	xgl_mipmap_texture_build xgl_object_set
Output:	Textured quadmesh with BOUNDARY_TRANSPARENT

▼ texture_tmap_op_4

Test Types:	SM, RGB
Description:	Tests texture operation
Attributes Tested:	XGL_TEXTURE_OP_* XGL_TEXTURE_BOUNDARY_CLAMP(_BOUNDARY)
Operators Tested:	xgl_mipmap_texture_build xgl_object_set
Output:	Textured quadmesh with both BOUNDARY_CLAMP and BOUNDARY_CLAMP_BOUNDARY

texture_tmap_mipmap_filter

Test Types: SM, RGB
Description: Texture sampling methods
Attributes Tested: `XGL_TEXTURE_INTERP_*`
Operators Tested: `xgl_mipmap_texture_build`
`xgl_object_set`
Output: Three textured triangles with different filters applied

texture_2tmap_op_1

Test Types: SM, RGB
Description: Tests texture operation
Attributes Tested: `XGL_TEXTURE_OP_*`
`XGL_TEXTURE_BOUNDARY_WRAP`
Operators Tested: `xgl_mipmap_texture_build`
`xgl_object_set`
Output: Two textures mapped onto the same quadmesh in sequence

texture_2tmap_op_2

Test Types: SM, RGB
Description: Tests texture operation
Attributes Tested: `XGL_TEXTURE_INTERP_BILINEAR`
`XGL_TMAP_COORD_SOURCE`
Operators Tested: `xgl_transform_write_specific`
`xgl_quadrilateral_mesh`
Output: Several textured spheres

texture_tmap_light_1

Test Types: SM, RGB
Description: Tests texture operation
Attributes Tested: `XGL_TEXTURE_OP_*`
`XGL_RENDER_COMP_DIFFUSE_COLOR`
Operators Tested: `xgl_mipmap_texture_build`
`xgl_object_set`
Output: A lighted and textured pentagon

texture_tmap_light_2

Test Types:	SM, RGB
Description:	Tests texture operation
Attributes Tested:	XGL_TEXTURE_OP_* XGL_RENDER_COMP_DIFFUSE_COLOR
Operators Tested:	xgl_mipmap_texture_build xgl_object_set
Output:	A lighted and textured pentagon with color_normal_facet

texture_tmap_light_3

Test Types:	SM, RGB
Description:	Tests texture operation
Attributes Tested:	XGL_TEXTURE_OP_* XGL_RENDER_COMP_REFLECTED_COLOR
Operators Tested:	xgl_mipmap_texture_build xgl_object_set
Output:	A lighted and textured pentagon

texture_tmap_light_4

Test Types:	SM, RGB
Description:	Tests texture operation
Attributes Tested:	XGL_TEXTURE_OP_* XGL_RENDER_COMP_FINAL_COLOR
Operators Tested:	xgl_mipmap_texture_build xgl_object_set
Output:	A lighted and textured pentagon

texture_tmap_light_5

Test Types:	SM, RGB
Description:	Tests texture operation
Attributes Tested:	XGL_TEXTURE_OP_* XGL_RENDER_COMP_FINAL_COLOR
Operators Tested:	xgl_multi_simple_polygon xgl_polygon
Output:	A lighted and textured pentagon

Transform Test Descriptions 30

This chapter describes the Transform test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ trans_operators_2d

Test Types:	INDEX and RGB, SM
Description:	Creates, copies, reads, writes, multiplies, inverts, transposes, and makes identity `2D INT`, `BIN` and `FLT` transform objects. Copies `2D INT` to `2D BIN` transform objects and vice versa.
Attributes Tested:	`XGL_TRANS_2D` and Table 30-1, Column A at the end of this chapter

Operators Tested:	`xgl_object_create`
	`xgl_transform_write`
	`xgl_transform_read`

	xgl_transform_copy xgl_transform_multiply xgl_transform_invert xgl_transform_transpose xgl_transform_identity xgl_object_destroy
Output:	All testing is done without the creation of a window raster, so nothing can be viewed on the monitor screen.

▼ trans_operators_3d

Test Types:	INDEX and RGB, SM
Description:	Creates, copies, reads, writes, multiplies, inverts, transposes, and makes identity 3D FLT and INT transforms (3D BIN transforms are not supported). Copies 2D FLT and INT transforms to 3D FLT and INT transforms respectively and vice versa.
Attributes Tested:	XGL_TRANS_2D XGL_TRANS_3D and Table 30-1, Column A at the end of this chapter
Operators Tested:	xgl_object_create xgl_transform_write xgl_transform_read xgl_transform_copy xgl_transform_multiply xgl_transform_transpose xgl_transform_identity xgl_object_destroy
Output:	All testing is done without the creation of a window raster, so nothing can be viewed on the monitor screen.

▼ trans_pt_ptlist_2d

Test Types:	INDEX and RGB, SM
Description:	Transforms F2D points. Transforms F2D, F2H, FLAG_F2D, and COLOR_F2D point lists using 2D FLT transform objects.
Attributes Tested:	XGL_TRANS_2D and Table 30-1, Column A at the end of this chapter

Operators Tested:	xgl_object_create xgl_transform_write xgl_transform_point xgl_transform_point_list
Output:	All testing is done without the creation of a window raster, so nothing can be viewed on the monitor screen.

▼ trans_pt_ptlist_3d

Test Types:	INDEX and RGB, SM
Description:	Transforms F3D points. Transforms F3D, F3H, FLAG_F3D, and COLOR_F3D point lists using 3D FLT transform objects.
Attributes Tested:	XGL_TRANS_3D and Table 30-1, Column A at the end of this chapter
Operators Tested:	xgl_object_create xgl_transform_write xgl_transform_point xgl_transform_point_list
Output:	All testing is done without the creation of a window raster, so nothing can be viewed on the monitor screen.

▼ trans_multiply_float

Test Types:	INDEX and RGB, SM
Description:	Multiplies all possible types of 2D (3D) transform objects (identity, translate, scale, scale-translate, rotate, 3x2(4x4), and general). These types are internal to XGL and the multiplication is different for each case for efficiency. Uses only FLT transforms.
Attributes Tested:	XGL_AXIS_X XGL_AXIS_Y XGL_AXIS_Z XGL_TRANS XGL_TRANS_2D XGL_TRANS_3D XGL_TRANS_DIMENSION XGL_TRANS_POSTCONCAT XGL_TRANS_PRECONCAT XGL_TRANS_REPLACE

Operators Tested:	xgl_object_create xgl_transform_translate xgl_transform_scale xgl_transform_rotate xgl_transform_write xgl_transform_multiply xgl_transform_read
Output:	All testing is done without the creation of a window raster, so nothing can be viewed on the monitor screen.

▼ Modeling Transformations

▼ trans_model_trans

Test Types:	INDEX and RGB, SM
Description:	Changes XGL_CTX_GLOBAL_MODEL_TRANS and LOCAL_MODEL_TRANS of a 2D context using 2D FLT, INT, and BIN transform objects. Checks that the XGL_CTX_MODEL_TRANS is correctly updated. Repeats for 3D context and transform objects.
Attributes Tested:	XGL_CTX_LOCAL_MODEL_TRANS XGL_CTX_MODEL_TRANS XGL_DATA_INT XGL_MEM_RAS XGL_TRANS_2D XGL_TRANS_3D XGL_TRANS_DBL and Table 30-1, Column B at the end of this chapter
Operators Tested:	xgl_object_create xgl_object_set xgl_object_get xgl_transform_rotate xgl_transform_translate xgl_transform_scale xgl_transform_multiply xgl_transform_read xgl_transform_copy xgl_transform_write

Output: All testing is done without the creation of a window raster, so nothing can be viewed on the monitor screen.

▼ trans_global_model_trans_2d

Test Types: INDEX, SM

Description: Tests global `2D FLT` modeling transforms (scaling, rotation, translation independent of each other) on `F2D` solid-filled polygons without edges (global `2D FLT` scaling, rotation, translation without interactions). Tries three different settings for each of the transformations, scaling, rotation, and translation. Does three rotations, one for each axis before another increment of 20 degrees is applied to the angle. Nine different angles are used in increments of 20 degrees from -20 degrees to a maximum of 170 degrees.

Attributes Tested: `XGL_TRANS_2D`
and Table 30-1, Column B at the end of this chapter

Operators Tested:
`xgl_object_create`
`xgl_polygon`
`xgl_transform_translate`
`xgl_object_set`
`xgl_transform_scale`
`xgl_transform_rotate`

Output: Draws for the default global model transformation six vertex polygons with the normal appearance of an arrowhead without the shaft, and the tip of the arrow pointing in the direction of the top of the window raster. For the small insignificant translation, the figure changes its position almost unnoticeably. For the substantial translation, the figure moves lower and to the right of its previous position. For the translation representing a shift distance of more than half the window raster width, the figure is clipped, so only a small parallelogram from the left portion of the arrow is visible.
For the small insignificant scaling, the figure is practically restored to its original position. For the equal significant scaling, the figure is substantially enlarged and appears in the middle right side of the window raster. For the double scaling for width with marginal increase for height, the

figure is enlarged and clipped approximately to the same extent as the translation example.
Rotation changes the position of the arrow polygon so that; (1) the arrow points to left of center, (2) the arrow returns to normal position but moves down the window, and (3) the arrow points to the right side of the window and moves toward the left side of the window raster. The arrow continues to face the same direction but moves more to the left with each subsequent rotation, with the arrow being clipped at the seventh and eighth loop and entirely clipped away at the ninth closing loop.

▼ trans_global_model_trans_2d_1

Test Types:	INDEX, SM
Description:	Tests global `2D FLT` modeling transforms (interaction of scaling, rotation, and translation) using all update modes on `F2D` solid-filled polygons without edges (interaction of global `2D FLT` scaling, rotation, translation, update modes). For two different translation transformations, tries three different rotation transformations in combination with three update modes, and for each of these rotation transformations, tries three scale transformations in combination with three update modes applied to the default global model transformation.

Attributes Tested:	`XGL_TRANS_2D`
	`XGL_TRANS_POSTCONCAT`
	`XGL_TRANS_PRECONCAT`
	and Table 30-1, Column B at the end of this chapter

Operators Tested:	`xgl_object_create`
	`xgl_polygon`
	`xgl_transform_translate`
	`xgl_transform_copy`
	`xgl_transform_rotate`
	`xgl_transform_scale`
	`xgl_object_set`
	`xgl_transform_multiply`

Output:	For the default global model transformation, the arrow-shaped polygon appears on the window raster. For all other changes to the global model transformation, the window raster remains the background black color.

▼ trans_global_model_trans_3d

Test Types:	INDEX, SM
Description:	Tests global `3D FLT` modeling transforms (scaling, rotation, and translation independent of each other) on `F3D` solid filled polygons without edges (global `3D FLT` scaling, rotation, and translation without interactions). Tries three different settings for each of the transformations, scaling, and translation. Does nine rotations, uses an axis, and then another increment of 20 degrees to the angle. Uses nine different angles in increments of 20 degrees from -20 degrees to a maximum of 170 degrees.
Attributes Tested:	`XGL_TRANS_3D` and Table 30-1, Column B at the end of this chapter

Operators Tested:	`xgl_polygon`
	`xgl_object_create`
	`xgl_transform_translate`
	`xgl_object_set`
	`xgl_transform_scale`
	`xgl_transform_rotate`

Output:	Draws for the default global model transformation six vertex polygons with the normal appearance of an arrowhead without the shaft, and the tip of the arrow pointing in the direction of the top of the window raster. For the small insignificant translation, the figure changes its position almost unnoticeably. For the substantial translation, the figure moves lower and to the right of its previous position. For the translation representing a shift distance of more than half the window raster width, the figure is clipped, so only a small parallelogram from the left portion of the arrow is visible. For the small insignificant scaling, the figure is practically restored to its original position. For the equal significant scaling, the figure is substantially enlarged and appears in

the middle right side of the window raster. For the double scaling for width with marginal increase for height, the figure is enlarged and clipped approximately to the same extent as the translation example. The first set of nine loops sees the position of the arrow change from its default location to movement vertically toward the top of the raster until clipping of the arrow structure takes place. The second set of nine loops sees the arrow structure move first right horizontally, and then left until clipping on the left side of the window raster takes place. The final loop sees the position change from pointing to the upper left side of the raster to swerving down and curving along a horizontal line parallel to the bottom of the window raster while pointing to the right side of the raster. Again the final positions reflect clipping of the arrow structure.

▼ trans_global_model_trans_3d_1

Test Types: INDEX, SM

Description: Tests global `3D FLT` modeling transforms (interaction of scaling, rotation, and translation) using all update modes on `F3D` solid filled polygons without edges (interaction of `3D FLT` global scaling, rotation, translation, and update modes). For two different translation transformations, tries two update modes in combination with two different rotations and two different scaling transformations.

Attributes Tested:
`XGL_TRANS_3D`
`XGL_TRANS_POSTCONCAT`
`XGL_TRANS_PRECONCAT`
and Table 30-1, Column B at the end of this chapter

Operators Tested:
`xgl_object_create`
`xgl_polygon`
`xgl_transform_translate`
`xgl_transform_copy`
`xgl_transform_rotate`
`xgl_transform_scale`
`xgl_object_set`
`xgl_transform_multiply`

Output:	First displays a polygon normally that is, six vertex arrow shaped objects with the point in the direction of the top of the window rasters. After the combined translation, rotation, and scaling transformations to the global modeling transform, the background color is all that is viewed.

▼ trans_update_model_trans

Test Types:	INDEX and RGB, SM
Description:	Tests `xgl_context_update_model_trans()`. First tests the various settings of `XGL_CTX_MODEL_TRANS_STACK_SIZE`. Then tries push and pop requests for local model trans and global model trans separately, together and finally several pushes and pops on the same stack. Tries `XGL_MTR_NEW_LEVEL`. Tries 2D and 3D transforms. The attribute values tried are `XGL_CTX_MODEL_TRANS_STACK_SIZE`: default, 0, and nonzero.

Attributes Tested:	`XGL_CTX_GLOBAL_MODEL_TRANS`
	`XGL_CTX_LOCAL_MODEL_TRANS`
	`XGL_CTX_MODEL_TRANS`
	`XGL_CTX_MODEL_TRANS_STACK_SIZE`
	`XGL_MTR_GLOBAL_TRANS`
	`XGL_MTR_LOCAL_TRANS`
	`XGL_MTR_NEW_LEVEL`
	`XGL_MTR_POP`
	`XGL_MTR_PUSH`
	`XGL_TRANS`
	`XGL_TRANS_2D`
	`XGL_TRANS_3D`
	`XGL_TRANS_DIMENSION`

Operators Tested:	`xgl_object_get`
	`xgl_object_create`
	`xgl_transform_write`
	`xgl_object_set`
	`xgl_transform_copy`
	`xgl_context_update_model_trans`

Output:	All testing is done without the creation of a window raster, so nothing can be viewed on the monitor screen.

View Transformation

trans_view_trans_3d

Test Types:	INDEX, SM
Description:	Tests nonidentity 3D FLT view transformation with default for other stages of the pipeline on F3D solid filled polygons without edges (simple 3D view transformation)
Attributes Tested:	XGL_CTX_VIEW_TRANS
Operators Tested:	xgl_object_get xgl_transform_write xgl_polygon
Output:	A small arrow-shaped polygon appears in the upper left side of the window raster.

Table 30-1 Transform Attributes Tested

Column A	Column B
XGL_DATA_FLT	XGL_AXIS_Y
XGL_TRANS	XGL_AXIS_Z
XGL_TRANS_DATA_TYPE	XGL_CTX_GLOBAL_MODEL_ TRANS
XGL_TRANS_DIMENSION	XGL_TRANS
	XGL_TRANS_DATA_TYPE
	XGL_DATA_FLT
	XGL_TRANS_DIMENSION
	XGL_TRANS_REPLACE

Transparency Test Descriptions 31

This chapter describes the Transparency test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ transp_blend_eq_mspg

Test Types:	RGB, CM
Description:	For-loop four different transparency blending equations. For each blending equation, set surface transparency method to blended, and draw opaque and transparent multi simple polygons.

Attributes Tested:	`XGL_3D_CTX_SURF_TRANSP_METHOD`
	`XGL_3D_CTX_SURF_TRANSP_BLEND_EQ`
	`XGL_3D_CTX_SURF_FRONT_TRANSP`

Operators Tested:	`xgl_object_set` `xgl_multi_simple_polygon`

Output:	Two green solid rectangles partially covering two blue solid rectangles in the top left portion of the canvas. Two dark green transparent rectangles on top of two blue solid rectangles with different darkness of green in the top right, bottom left, and bottom right portions of the canvas.

▼ transp_blend_eq_mspg_draw_unblended

Test Types:	RGB, CM
Description:	Sets HLHSR and to draw unblended. For-loop four different transparency blending equations. For each blending equation, set surface transparency method to blended, and draw opaque and transparent multi simple polygons.
Attributes Tested:	XGL_3D_CTX_BLEND_DRAW_MODE XGL_3D_CTX_SURF_TRANSP_METHOD XGL_3D_CTX_SURF_TRANSP_BLEND_EQ
Operators Tested:	xgl_object_set xgl_multi_simple_polygon
Output:	Two green solid rectangles partially covering two blue solid rectangles in the top left portion of the canvas. Two blue solid rectangles in the top right, bottom left and bottom right portions of the canvas.

▼ transp_blended_hollow_mspg

Test Types:	RGB, CM
Description:	Sets HLHSR, hollow surface fill style, blended transparency method, and arbitrary background blending equation. Draws unblended opaque and transparent multi simple polygons. Draws blended opaque and transparent multi simple polygons. Draw all polygons.
Attributes Tested:	XGL_CTX_SURF_FRONT_FILL_STYLE XGL_3D_CTX_SURF_TRANSP_METHOD XGL_3D_CTX_SURF_TRANSP_BLEND_EQ XGL_3D_CTX_BLEND_DRAW_MODE XGL_3D_CTX_SURF_FRONT_TRANSP
Operators Tested:	xgl_object_set xgl_multi_simple_polygon

Output:	Two blue hollow rectangles in the top left portion of the canvas, two green hollow rectangles in the top right, and two blue rectangles and two green rectangles in the bottom left portion of the canvas. The bottom right is blank.

▼ transp_blended_mspg

Test Types:	RGB, CM
Description:	Sets blended transparency method and arbitary background blending equation. Draws unblended opaque and transparent multi simple polygons. Draws blended opaque and transparent multi simple polygons. Sets edge flag. Draws unblended and blended opaque and transparent multi simple polygons.

Attributes Tested:	`XGL_3D_CTX_SURF_TRANSP_METHOD`
	`XGL_3D_CTX_SURF_TRANSP_BLEND_EQ`
	`XGL_3D_CTX_BLEND_DRAW_MODE`
	`XGL_3D_CTX_SURF_FRONT_TRANSP`
	`XGL_CTX_SURF_EDGE_FLAG`

Operators Tested:	`xgl_object_set` `xgl_multi_simple_polygon`
Output:	Two blue solid rectangles in the top left portion of the canvas, and in the bottom left portion of the canvas. Two dark green rectangles in the top right and two hollow (edges) rectangles and two dark green rectangles with edges in the bottom right portion of the canvas.

▼ transp_screen_door_circle

Test Types:	RGB, CM
Description:	Sets screen door transparency. Draws unblended and blended opaque and transparent circles. Sets edge flag. Draws unblended and blended opaque and transparent circles.

Attributes Tested:	`XGL_3D_CTX_SURF_TRANSP_METHOD`
	`XGL_3D_CTX_BLEND_DRAW_MODE`
	`XGL_3D_CTX_SURF_FRONT_TRANSP`
	`XGL_CTX_SURF_EDGE_FLAG`

Operators Tested:	`xgl_object_set` `xgl_multicircle`
Output:	Two green transparent circles partially covering two blue solid circles in the top left and top right portions of the canvas. Two green transparent circles with edges on partially covering two blue solid circles with edges in the bottom left and bottom right portions of the canvas.

▼ transp_screen_door_mspg

Test Types:	RGB, CM
Description:	Sets screen door transparency. Draws unblended and blended opaque and transparent multi simple polygons. Sets edge flag. Draws unblended and blended opaque and transparent multi simple polygons.

Attributes Tested:	`XGL_3D_CTX_SURF_TRANSP_METHOD`
	`XGL_3D_CTX_BLEND_DRAW_MODE`
	`XGL_3D_CTX_SURF_FRONT_TRANSP`
	`XGL_CTX_SURF_EDGE_FLAG`

Operators Tested:	`xgl_object_set` `xgl_multi_simple_polygon`
Output:	A green transparent rectangle partially covering a blue solid rectangle in the top left and top right portions of the canvas. A green transparent rectangle with edges on partially on top of a blue solid rectangle with edges in the bottom left and bottom right portions of the canvas.

▼ transp_screen_door_pg

Test Types:	RGB, CM
Description:	Sets screen door transparency. Draws unblended and blended opaque and transparent polygons. Sets edge flag. Draws unblended and blended opaque and transparent polygons.

Attributes Tested:	`XGL_3D_CTX_SURF_TRANSP_METHOD`
	`XGL_3D_CTX_BLEND_DRAW_MODE`
	`XGL_3D_CTX_SURF_FRONT_TRANSP`
	`XGL_CTX_SURF_EDGE_FLAG`

Operators Tested:	`xgl_object_set` `xgl_polygon`

Output:	Two green transparent rectangles partially on top of two blue solid rectangles in the top left and top right portions of the canvas. Two green transparent rectangles with edges on partially on top of two blue solid rectangles with edges in the bottom left and bottom right portions of the canvas.

▼ transp_screen_door_qm

Test Types:	RGB, CM
Description:	Sets screen door transparency. Draws unblended and blended opaque and transparent quadmeshes. Sets edge flag. Draws unblended and blended opaque and transparent quadmeshes.
Attributes Tested:	XGL_3D_CTX_SURF_TRANSP_METHOD XGL_3D_CTX_BLEND_DRAW_MODE XGL_3D_CTX_SURF_FRONT_TRANSP XGL_CTX_SURF_EDGE_FLAG
Operators Tested:	xgl_object_set xgl_quadrilateral_mesh
Output:	Two green transparent quadmeshes partially on top of two red solid quadmeshes in the top left and top right portions of the canvas. Two green transparent quadmeshes with edges on partially on top of two red solid quadmeshes with edges in the bottom left and bottom right portions of the canvas.

▼ transp_screen_door_rect

Test Types:	RGB, CM
Description:	Sets screen door transparency. Draws unblended and blended opaque and transparent rectangles. Sets edge flag. Draws unblended and blended opaque and transparent rectangles.
Attributes Tested:	XGL_3D_CTX_SURF_TRANSP_METHOD XGL_3D_CTX_BLEND_DRAW_MODE XGL_3D_CTX_SURF_FRONT_TRANSP XGL_CTX_SURF_EDGE_FLAG
Operators Tested:	xgl_object_set xgl_multirectangle

Output: Two green transparent rectangles partially on top of two blue solid rectangles in the top left and top right portions of the canvas. Two green transparent rectangles with edges on partially on top of two blue solid rectangles with edges in the bottom left and bottom right portions of the canvas.

▼ transp_screen_door_tl

Test Types: RGB, CM

Description: Sets screen door transparency. Draws unblended and blended opaque and transparent triangle lists. Sets edge flag. Draws unblended and blended opaque and transparent triangle lists.

Attributes Tested: XGL_3D_CTX_SURF_TRANSP_METHOD
XGL_3D_CTX_BLEND_DRAW_MODE
XGL_3D_CTX_SURF_FRONT_TRANSP
XGL_CTX_SURF_EDGE_FLAG

Operators Tested: xgl_object_set
xgl_triangle_list

Output: A green transparent rhombus partially on top of a red solid rhombus in the top left and top right portions of the canvas. A green transparent rhombus with triangle edges on partially on top of a red solid rhombus with triangle edges in the bottom left and bottom right portions of the canvas.

▼ transp_screen_door_ts

Test Types: RGB, CM

Description: Sets screen door transparency. Draws unblended and blended opaque and transparent triangle strip. Sets edge flag. Draws unblended and blended opaque and transparent triangle strip.

Attributes Tested: XGL_3D_CTX_SURF_TRANSP_METHOD
XGL_3D_CTX_BLEND_DRAW_MODE
XGL_3D_CTX_SURF_FRONT_TRANSP
XGL_CTX_SURF_EDGE_FLAG

Operators Tested: xgl_object_set
xgl_triangle_strip

Output: A green transparent triangle partially on top of a red solid triangle in the top left and top right portions of the canvas. A green transparent triangle with edges on partially on top of a red solid triangle with edges in the bottom left and bottom right portions of the canvas.

▼ transp_screen_door_values_mspg

Test Types:	RGB, CM
Description:	Sets screen door transparency. Draws four opaque polygons. Draws 16 transparent polygons with varying degrees of transparency on top of the opaque polygons.
Attributes Tested:	`XGL_3D_CTX_SURF_TRANSP_METHOD`, `XGL_3D_CTX_SURF_FRONT_TRANSP`
Operators Tested:	`xgl_object_set` `xgl_multi_simple_polygon`
Output:	Sixteen green polygons with varying degree of transparency on top of four blue solid polygons.

Triangle List Test Descriptions 32

This chapter describes the Triangle List test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section “Denizen Test Types” on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ tlist_flag1

Test Types:	RGB, SM
Description:	Tests triangle list with vertex flags and facet color
Attributes Tested:	`XGL_3D_CTX_SURF_FRONT_ILLUMINATION`

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_triangle_list`

Output:	A red trapezoid, a red triangle in the top row, a red parrallelogram, and an attached yellow rhombus in the bottom row

▼ tlist_flag2

Test Types:	RGB, SM
Description:	Tests triangle list with vertex flags. Draws an independent triangle, tstrip, tstar followed by a tstrip and facet color.
Attributes Tested:	`XGL_3D_CTX_SURF_FRONT_ILLUMINATION`

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_triangle_list`

Output:	A green independent triangle, a white tstar (looks like a parallelepiped), and two white strips in the bottom rows.

▼ tlist_flag3

Test Types:	RGB, SM
Description:	Tests triangle list with vertex winding flags.
Attributes Tested:	`XGL_3D_CTX_SURF_GEOM_NORMAL`

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_triangle_list`

Output:	Two sets of triangles and polygon are drawn on the canvas. Some triangles are missing with the changing of the winding flag.

▼ tlist_indep

Test Types:	RGB, SM
Description:	Tests the drawing of independent triangle using the `xgl_triangle_list()` test, color interpolation and the use of facet colors.
Attributes Tested:	`XGL_3D_CTX_SURF_FRONT_ILLUMINATION`

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_triangle_list`

Output:	Three rows of triangles, with three triangles per row. The first row has red triangles. The second row has interpolated vertex colors; the first triangle colors are R, W and G, the second triangle colors are Y, B, Magenta, and

the third triangle colors are Cyan, Red and White. The third row has a white, red and green triangle tlist_indep.

tlist_star

Test Types: RGB, SM

Description: Tests triangle list with global flag XGL_TLIST_FLAG_TRI_STAR. The tests also exercises vertex colors and facet colors.

Attributes Tested: XGL_3D_CTX_SURF_FRONT_ILLUMINATION

Operators Tested: xgl_object_get
xgl_object_set
xgl_triangle_list

Output: A green hexagon, an interpolated hexagon with first vertex color as white and vertex colors (Cyan, Red, Green, Blue, Yellow, Magenta), and a multi-colored hexagon (G, B, Y, Magenta, W, R). A red parrallelogram and an attached yellow rhombus in the bottom row.

tlist_star2

Test Types: RGB, SM

Description: Tests triangle list with global flag as TSTAR and facet colors and normals and back culling enabled.

Attributes Tested: XGL_3D_CTX_SURF_FRONT_ILLUMINATION
XGL_3D_CTX_SURF_FACE_CULL

Operators Tested: xgl_object_get
xgl_object_set
xgl_triangle_list

Output: A white parrallelogram

Triangle Strip Test Descriptions 33

This chapter describes the Triangle Strip test programs. The following is defined for each test program:

- Name of the test program
- Test types (See the section "Denizen Test Types" on page 2 for the different test types.)
- Description of the test program
- Attributes tested by the program
- Operators tested by the program
- Output from the test program

▼ ts_cull

Test Types:	INDEX, SM
Description:	Tests a triangle strip with three facets (triangles), two front facing and one back facing. Tries the three values of face culling. These values are none, front, and back culling. Tests loops through the three face culling modes—XGL_CULL_NONE, XGL_CULL_FRONT, and XGL_CULL_BACK.
Attributes Tested:	See Table 33-1, Column A at the end of this chapter.

Operators Tested: xgl_object_get
xgl_object_set
xgl_triangle_strip

Output: Draws a triangle strip with three triangles sharing a common vertex. The triangles on the left and right are front facing, and the one at the bottom is back facing.

▼ ts_cull_rgb

Test Types: RGB, SM

Description: Tests a triangle strip with three facets (triangles), two front facing and one back facing. Tries the three values of face culling. These values are none, front, and back culling. Tests loops through the three face culling modes—XGL_CULL_NONE, XGL_CULL_FRONT, and XGL_CULL_BACK.

Attributes Tested: See Table 33-1, Column A at the end of this chapter.

Operators Tested: xgl_object_get
xgl_object_set
xgl_triangle_strip

Output: Draws a triangle strip with three triangles sharing a common vertex. The triangles on the left and right are front facing, and the one at the bottom is back facing.

▼ ts_empty_interp

Test Types: INDEX, SM

Description: Tries all point types and facet types for a triangle strip while color interpolation is on and the fill style is empty. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.

Attributes Tested: See Table 33-1, Column B at the end of this chapter.
Operators Tested: `xgl_object_get`
`xgl_object_set`
`xgl_triangle_strip`
Output: Draws a triangle strip with two empty facet edges of the edge color. Should render two triangle edges of the edge color for each loop.

▼ ts_empty_interp_rgb

Test Types: RGB, SM
Description: Tries all point types and facet types for a triangle strip while color interpolation is on and the fill style is empty. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL) with facet type the innermost loop.
Attributes Tested: See Table 33-1, Column B at the end of this chapter.
Operators Tested: `xgl_object_get`
`xgl_object_set`
`xgl_triangle_strip`
Output: Draws a triangle strip with two empty facet edges of the edge color. Should render two triangle edges of the edge color for each loop.

▼ ts_empty_no_illum

Test Types: INDEX, SM
Description: Tries all point types and facet types for a triangle strip while illumination is off and the fill style is empty. The raster color type is INDEX. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types

	(XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL) with facet type the innermost loop.
Attributes Tested:	See Table 33-1, Column B at the end of this chapter.

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_triangle_strip`

Output:	Draws an empty triangle strip with two facets (triangles) of the edge color. Should render two edge-colored triangles for each loop.

▼ ts_empty_no_illum_rgb

Test Types:	RGB, SM
Description:	Tries all point types and facet types for a triangle strip while illumination is off and the fill style is empty. The raster color type is RGB. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL) with facet type the innermost loop.
Attributes Tested:	See Table 33-1, Column B at the end of this chapter.

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_triangle_strip`

Output: Draws an empty triangle strip with two facets (triangles) of edge color. Should render two hollow triangles for each loop. Of special interest is the color of the one shared common edge, which should be rendered with the color selected for the second triangle.

ts_empty_per_facet

Test Types: INDEX, SM

Description: Tries all point types and facet types for a triangle strip while illumination is per facet and the fill style is empty. The raster color type is INDEX. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL) with facet type the innermost loop.

Attributes Tested: See Table 33-1, Column C at the end of this chapter.

Operators Tested: xgl_object_get
xgl_object_set
xgl_triangle_strip

Output: Draws an empty triangle strip with two facets (triangles) of the edge color. Should render two edge-colored triangles with each loop.

ts_empty_per_facet_rgb

Test Types: RGB, SM

Description: Tries all point types and facet types for a triangle strip while illumination is per facet and the fill style is empty. The raster color type is RGB. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D,

	XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL) with facet type the innermost loop.
Attributes Tested:	See Table 33-1, Column C at the end of this chapter.
Operators Tested:	xgl_object_get xgl_object_set xgl_triangle_strip
Output:	Draws an empty triangle strip with two facets (triangles) of the edge color. Should render two edge-colored triangles with each loop.

▼ ts_empty_per_vtx

Test Types:	INDEX, SM
Description:	Tries all point types and facet types for a triangle strip while illumination is per vertex and the fill style is empty. The raster color type is INDEX. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL) with facet type the innermost loop.
Attributes Tested:	See Table 33-1, Column C at the end of this chapter.
Operators Tested:	xgl_object_get xgl_object_set xgl_triangle_strip
Output:	Draws a triangle strip with two empty facet edges of the edge color. Should render two triangle edges of the edge color for each loop.

▼ ts_empty_per_vtx_rgb

Test Types:	RGB, SM
Description:	Tries all point types and facet types for a triangle strip while illumination is per vertex and the fill style is empty. The raster color type is RGB. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL) with facet type the innermost loop.
Attributes Tested:	See Table 33-1, Column C at the end of this chapter.
Operators Tested:	xgl_object_get xgl_object_set xgl_triangle_strip
Output:	Draws a triangle strip with two empty facet edges of the edge color. Should render two triangle edges of the edge color with each loop.

▼ ts_gcache_col_norm

Test Types:	INDEX, SM
Description:	Tests gcache tristrips utilizing various facet types, vertex types and illumination modes. Tests loops through FILL_STYLE (SOLID, HOLLOW, EMPTY); FRONT_ILLUMINATION (NONE, PER_FACET, PER_VERTEX); vertex data (F3D, COLOR_F3D, NORMAL_F3D, COLOR_NORMAL_F3D); and facet type (FACET_NONE, FACET_COLOR, FACET_NORMAL), with facet type the innermost loop.
Attributes Tested:	See Table 33-2, Column A at the end of this chapter.

Operators Tested: `xgl_object_create`
`xgl_object_get`
`xgl_object_set`
`xgl_gcache_triangle_strip`
`xgl_context_display_gcache`

Output: Since the front surface color is green, the single triangle rendered with facet none or facet normal is green. Since the facet color and the facet normal color is red, the single rendered triangle is this color for these settings. For illumination per vertex, the color is most likely one of a rainbow from red to gray, with the intermediary colors green, yellow, dark blue, purple, and light blue.

▼ ts_gcache_col_norm_rgb

Test Types: RGB, SM

Description: Tests gcache tristrip utilizing various facet types, vertex types, and illumination modes. Tests loops through `FILL_STYLE` (`SOLID`, `HOLLOW`, `EMPTY`); `FRONT_ILLUMINATION` (`NONE`, `PER_FACET`, `PER_VERTEX`); vertex data (`F3D`, `COLOR_F3D`, `NORMAL_F3D`, `COLOR_NORMAL_F3D`); and facet type (`FACET_NONE`, `FACET_COLOR`, `FACET_NORMAL`) with facet type the innermost loop.

Attributes Tested: See Table 33-2, Column A at the end of this chapter.

Operators Tested: `xgl_object_create`
`xgl_object_get`
`xgl_object_set`
`xgl_gcache_triangle_strip`
`xgl_context_display_gcache`

Output: Since the front surface color is red, the single triangle rendered with facet none or facet normal is red. Since the facet color and the facet normal color is blue, the single rendered triangle is this color for these settings. For illumination per vertex and vertex type with color information, `COLOR_F3D` and `COLOR_NORMAL_F3D` are most likely a gradual shading from red to green.

▼ ts_gcache_cull

Test Types:	INDEX, SM
Description:	Tests face-culling modes for a gcache tristrip. The tristrip consists of three facets (triangles) with the two uppermost being front facing and the bottom being back facing, as set by their facet normals. The front surface color is red, while the back surface color is green.
Attributes Tested:	See Table 33-2, Column B at the end of this chapter.

Operators Tested:	`xgl_object_create`
	`xgl_object_get`
	`xgl_object_set xgl_gcache_triangle_strip`
	`xgl_context_display_gcache`

Output:	Culling off has three triangles, with the leftmost and rightmost triangles red, and the bottom triangle green. Culling front has only the bottom green triangle. Culling back has only the adjacent red triangles.

▼ ts_gcache_cull_rgb

Test Types:	RGB, SM
Description:	Tests face-culling modes for a gcache tristrip. The tristrip consists of three facets (triangles) with the two uppermost being front facing and the bottom being back facing, as set by their facet normals. The front surface color is red, while the back surface color is green.
Attributes Tested:	See Table 33-2, Column B at the end of this chapter.

Operators Tested:	`xgl_object_create`
	`xgl_object_get`
	`xgl_object_set`
	`xgl_gcache_triangle_strip`
	`xgl_context_display_gcache`

Output:	Culling off has three triangles, with the leftmost and rightmost triangles red, and the bottom triangle green. Culling front has only the bottom green triangle. Culling back has only the adjacent red triangles.

▼ ts_gcache_hlhsr

Test Types:	INDEX, SM
Description:	Tests hidden surface removal of a gcache tristrip. Loops through three different combinations for the depth, z value. Renders the same two point lists with the same facets list twice, differing only their depth values and changing the front surface color. Verifies pixels for the frontmost point list as the only colored pixels on the window raster. Last case renders a degenerate tristrip with two triangles overlapping within the same tristrip.

Attributes Tested:	`XGL_GCACHE`
	`XGL_GCACHE_IS_EMPTY`
	and Table 33-3, Column C at the end of this chapter

Operators Tested:	`xgl_object_create`
	`xgl_object_get`
	`xgl_object_set`
	`xgl_gcache_triangle_strip`
	`xgl_context_display_gcache`
	`xgl_object_destroy`
	`xgl_context_new_frame`

Output:	Draws one triangle plus two triangles producing a parallelogram. Both triangles are either red or green, dependent on which color is currently set to the front surface color. The last case displays only one red triangle.

▼ ts_gcache_hlhsr_rgb

Test Types:	RGB, SM
Description:	Tests hidden surface removal of a gcache tristrip. Loops through three different combinations for the depth, z value. Renders the same two point lists with the same two facets twice differing only their depth values and changing the front surface color. Verifies pixels for the frontmost point list as the only colored pixels on the window raster. The last case renders a degenerate tristrip, with two triangles overlapping within the same tristrip.

Attributes Tested:	XGL_CTX_BACKGROUND_COLOR XGL_GCACHE XGL_GCACHE_IS_EMPTY and Table 33-3, Column C at the end of this chapter
Operators Tested:	xgl_object_create xgl_object_get xgl_object_set xgl_gcache_triangle_strip xgl_context_display_gcache xgl_object_destroy xgl_context_new_frame
Output:	Draws one triangle plus two triangles producing a parallelogram. Both triangles are either peach or blue, dependent on which color is currently set to the front surface color. The last case displays only one peach triangle.

▼ ts_gcache_shade

Test Type:	INDEX, SM
Description:	Tests shaded gcache tristrip on an INDEX raster.
Attributes Tested:	XGL_3D_CTX_SURF_FRONT_ILLUMINATION
Output:	Shaded gcache tristrip

▼ ts_gcache_shade_rgb

Test Type:	RGB, SM
Description:	Tests shaded gcache tristrip on a RGB raster.
Attributes Tested:	XGL_3D_CTX_SURF_FRONT_ILLUMINATION
Output:	Shaded gcache tristrip

▼ ts_gcache_simple

Test Types:	INDEX, SM
Description:	Tests simple gcache tristrip (1 triangle) on an INDEX raster. Fill styles HOLLOW, SOLID and EMPTY are tried with and without illumination. The light type is AMBIENT and illumination is PER_VERTEX when illumination is used.

Attributes Tested:	XGL_CTX_SURF_FRONT_FILL_STYLE XGL_3D_CTX_SURF_FRONT_ILLUMINATION
Output:	Each frame shows one triangle. For each fill style the illumination is set to NONE and PER_VERTEX in successive frames before moving on to the next fill style. Fill style is set to SOLID, HOLLOW and EMPTY in that order.

▼ ts_gcache_simple_rgb

Test type:	RGB, SM
Description:	Tests one facet and various fill styles
Attributes Tested:	XGL_CTX_SURF_FRONT_FILL_STYLE XGL_3D_CTX_SURF_FRONT_ILLUMINATION
Output:	Each frame shows one triangle of various fill styles.

▼ ts_hlhsr

Test Types:	INDEX, SM
Description:	Tests solid filled tristrips hidden surface removal. Draws two triangle strips—one with one triangle and another with two triangles. Then draws the same triangles at a different depth, overlapping the first rendering. This is done for three combinations of depths (0,100), (100,0), and (100,100), with the first number in each pair being the depth used for the first rendering. Finally, a triangle strip with two triangles overlapping each other is drawn. One of the triangles has a vertex at z==150; all other vertexes are at z==0. This vertex is the lower- left corner vertex.
Attributes Tested:	See Table 33-3, Column C at the end of this chapter.
Operators Tested:	xgl_object_set xgl_triangle_strip xgl_context_new_frame
Output:	Draws one triangle plus two triangles producing a parallelogram. Both triangles are either red or green, dependent on which color is currently set to the front surface color. The last case displays only one red triangle.

▼ ts_hlhsr_rgb

Test Types: RGB, SM

Description: Draws solid filled tristrips hidden surface removal. Draws two triangle strips—one with one triangle and another with two triangles. Then draws the same triangles at a different depth, overlapping the first rendering. This is done for three combinations of depths (0,100), (100,0), and (100,100) with the first number in each pair being the depth used for the first rendering. Finally, a triangle strip with two triangles overlapping each other is drawn. One of the triangles has a vertex at z==150; all other vertexes are at z==0. This vertex is the lower-left corner vertex.

Attributes Tested: See Table 33-3, Column C at the end of this chapter.

Operators Tested: `xgl_object_set`
`xgl_triangle_strip`
`xgl_context_new_frame`

Output: Draws one triangle plus two triangles producing a parallelogram. Both triangles are either peach or blue, dependent on which color is currently set to the front surface color. The last case displays only one peach triangle.

▼ ts_hollow_interp

Test Types: INDEX, SM

Description: Tries all point types and facet types for a triangle strip while color interpolation is on and the fill style is hollow. The raster color type is `INDEX`. Tests loops through two values of `XGL_3D_CTX_HLHSR_MODE` (`XGL_HLHSR_Z_BUFFER`, and the default `value` `XGL_HLHSR_NONE`); nine point types (`XGL_PT_F3D`, `XGL_PT_COLOR_F3D`, `XGL_PT_NORMAL_F3D`, `XGL_PT_COLOR_NORMAL_F3D`, `XGL_PT_FLAG_F3D`, `XGL_PT_COLOR_FLAG_F3D`, `XGL_PT_NORMAL_FLAG_F3D`, `XGL_PT_COLOR_NORMAL_FLAG_F3D`, `XGL_PT_F3H`); and three facet types (`XGL_FACET_NONE`, `XGL_FACET_COLOR`, `XGL_FACET_COLOR_NORMAL`), with facet type the innermost loop.

Attributes Tested:	XGL_CTX_BACKGROUND_COLOR XGL_DRAW_EDGE XGL_DRAW_PREV_EDGE XGL_HLHSR_NONE XGL_SURF_FILL_HOLLOW XGL_SURF_FRONT_FILL_STYLE and Table 33-3, Column C at the end of this chapter
Operators Tested:	xgl_object_get xgl_object_set xgl_context_new_frame xgl_triangle_strip
Output:	Draws several renditions of a triangle strip with two hollow facets. Should render two triangles with each loop.

▼ ts_hollow_interp_rgb

Test Types:	RGB, SM
Description:	Tries all point types and facet types for a triangle strip while color interpolation is on and the fill style is hollow. The raster color type is RGB. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.
Attributes Tested:	XGL_CTX_BACKGROUND_COLOR XGL_DRAW_EDGE XGL_DRAW_PREV_EDGE XGL_HLHSR_NONE XGL_SURF_FILL_HOLLOW XGL_SURF_FRONT_FILL_STYLE and Table 33-3, Column C at the end of this chapter

Operators Tested: xgl_object_get
xgl_object_set
xgl_context_new_frame
xgl_triangle_strip

Output: Draws several renditions of a triangle strip with two hollow facets. Should render two triangles with each loop.

▼ ts_hollow_no_illum

Test Types: INDEX, SM

Description: Tries all point types and facet types for a triangle strip while illumination is none and the fill style is hollow. The raster color type is INDEX. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (default and XGL_HLHSR_Z_BUFFER, default is XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.

Attributes Tested: See Table 33-3, Column B at the end of this chapter.

Operators Tested: xgl_object_get
xgl_object_set
xgl_triangle_strip

Output: Draws several renditions of a triangle strip with two hollow facets. Should render two triangles with each loop. The shared edge color should be the one selected for the second triangle (the triangle on the right).

▼ ts_hollow_no_illum_rgb

Test Types: RGB, SM

Description: Tries all point types and facet types for a triangle strip while illumination is none and the fill style is hollow. The raster color type is RGB. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types

	(XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.
Attributes Tested:	See Table 33-3, Column B at the end of this chapter.
Operators Tested:	xgl_object_get xgl_object_set xgl_triangle_strip
Output:	Draws several renditions of a triangle strip with two hollow facets. Should render two triangles with each loop.

▼ ts_hollow_per_facet

Test Types:	INDEX, SM
Description:	Tries all point types and facet types for a triangle strip while illumination is per facet and the fill style is hollow. The raster color type is INDEX. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.
Attributes Tested:	XGL_CTX_SURF_FRONT_COLOR XGL_CTX_SURF_FRONT_FILL_STYLE XGL_DRAW_EDGE XGL_SURF_FILL_HOLLOW XGL_SURF_FRONT_FILL_STYLE and Table 33-1, Column C at the end of this chapter
Operators Tested:	xgl_object_get xgl_object_set xgl_triangle_strip

Output: Draws several renditions of a triangle strip with two hollow facets. Should render two triangles with each loop.

▼ ts_hollow_per_facet_rgb

Test Types: RGB, SM

Description: Tries all point types and facet types for a triangle strip while illumination is per facet and the fill style is hollow. The raster color type is RGB. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.

Attributes Tested: XGL_CTX_SURF_FRONT_COLOR
XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_DRAW_EDGE
XGL_SURF_FILL_HOLLOW
XGL_SURF_FRONT_FILL_STYLE
and Table 33-1, Column C at the end of this chapter

Operators Tested: xgl_object_get
xgl_object_set
xgl_triangle_strip

Output: Draws several renditions of a triangle strip with two hollow facets. Should render two triangles with each loop.

▼ ts_hollow_per_vtx

Test Types: INDEX, SM

Description: Tries all point types and facet types for a triangle strip while illumination is per vertex and the fill style is hollow. The raster color type is INDEX. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D,

	XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.
Attributes Tested:	XGL_CTX_SURF_FRONT_COLOR XGL_CTX_SURF_FRONT_FILL_STYLE XGL_DRAW_EDGE XGL_SURF_FILL_HOLLOW XGL_SURF_FRONT_FILL_STYLE and Table 33-1, Column C at the end of this chapter
Operators Tested:	xgl_object_get xgl_object_set xgl_triangle_strip
Output:	Draws several renditions of a triangle strip with two hollow facets. Should render two triangles with each loop.

▼ ts_hollow_per_vtx_rgb

Test Types:	RGB, SM
Description:	Tries all point types and facet types for a triangle strip while illumination is per vertex and the fill style is hollow. The raster color type is RGB. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.
Attributes Tested:	XGL_CTX_SURF_FRONT_COLOR XGL_CTX_SURF_FRONT_FILL_STYLE XGL_DRAW_EDGE

	XGL_SURF_FILL_HOLLOW
	XGL_SURF_FRONT_FILL_STYLE
	and Table 33-1, Column C at the end of this chapter
Operators Tested:	xgl_object_get
	xgl_object_set
	xgl_triangle_strip
Output:	Draws several renditions of a triangle strip with two hollow facets. Should render two triangles with each loop.

▼ ts_shade

Test Types:	INDEX, SM
Description:	Tests the shaded triangle strip with lighting type illumination per vertex. The triangle strip has three faces. The point type is `color_f3d` and the light used is an ambient source.
Attributes Tested:	XGL_3D_CTX_HLHSR_MODE
	XGL_3D_CTX_LIGHTS
	XGL_3D_CTX_LIGHT_NUM
	XGL_3D_CTX_LIGHT_SWITCHES
	XGL_3D_CTX_SURF_FRONT_AMBIENT
	XGL_3D_CTX_SURF_FRONT_ILLUMINATION
	XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT
	XGL_3D_CTX_SURF_FRONT_LIGHT_TYPE
	XGL_HLHSR_Z_BUFFER
	XGL_ILLUM_PER_VERTEX
	XGL_LIGHT_AMBIENT
	XGL_LIGHT_COLOR
	XGL_LIGHT_ENABLE_COMP_AMBIENT
	XGL_LIGHT_ENABLE_TYPE_AMBIENT
	XGL_LIGHT_TYPE
Operators Tested:	xgl_object_get
	xgl_object_set
	xgl_triangle_strip
Output:	The first triangle strip drawn has three facets with one vertex shared by all three. Then a triangle strip with only one triangle is drawn. The process is repeated with the Z-buffer on.

▼ ts_shade_rgb

Test Types:	RGB, SM
Description:	Tests the shaded triangle strip with lighting type illumination per vertex. The triangle strip has three faces. The point type is `color_f3d` and the light used is an ambient source.
Attributes Tested:	XGL_3D_CTX_HLHSR_MODE XGL_3D_CTX_LIGHTS XGL_3D_CTX_LIGHT_NUM XGL_3D_CTX_LIGHT_SWITCHES XGL_3D_CTX_SURF_FRONT_AMBIENT XGL_3D_CTX_SURF_FRONT_ILLUMINATION XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT XGL_ILLUM_PER_VERTEX XGL_3D_CTX_SURF_FRONT_LIGHT_TYPE XGL_HLHSR_Z_BUFFER XGL_LIGHT_AMBIENT XGL_LIGHT_COLOR XGL_LIGHT_ENABLE_COMP_AMBIENT XGL_LIGHT_ENABLE_TYPE_AMBIENT XGL_LIGHT_TYPE

Operators Tested:	`xgl_object_get`
	`xgl_object_set`
	`xgl_triangle_strip`

Output:	The first triangle strip drawn has three facets with one vertex shared by all three. Then a triangle strip with only one triangle is drawn. The process is repeated with the Z-buffer on.

▼ ts_simple

Test Types:	INDEX, SM
Description:	Tests a simple tristrip (one triangle). Tries with and without Z-buffer. Tries hollow, solid, and empty fill styles with per-vertex illumination, per-facet illumination, and without illumination. Various point types are tested. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_NONE, XGL_HLHSR_Z_BUFFER); three values of XGL_CTX_SURF_FRONT_FILL_STYLE

	(XGL_SURF_FILL_SOLID, XGL_SURF_FILL_HOLLOW, XGL_SURF_FILL_EMPTY); and three values of XGL_3D_CTX_SURF_FRONT_ILLUMINATION (XGL_ILLUM_NONE, XGL_ILLUM_PER_FACET and XGL_ILLUM_PER_VTX). The point types used for these three illumination modes are XGL_PT_F3D, XGL_PT_COLOR_F3D and XGL_PT_F3D respectively.
Attributes Tested:	XGL_CTX_EDGE_COLOR XGL_CTX_SURF_EDGE_FLAG XGL_FACET_NORMAL XGL_LIGHT_ENABLE_COMP_AMBIENT XGL_LIGHT_TYPE XGL_SURF_FILL_EMPTY XGL_SURF_FILL_HOLLOW XGL_SURF_FILL_SOLID and Table 33-2, Column A at the end of this chapter
Operators Tested:	xgl_object_get xgl_object_set xgl_triangle_strip
Output:	Draws a single triangle tristrip for each loop

▼ ts_simple_rgb

Test Types:	RGB, SM
Description:	Tests a simple tristrip (one triangle). Tries with and without the Z-buffer. Tries hollow, solid, and empty fill styles with per vertex illumination, per facet illumination, and without illumination. Various point types are tested. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_NONE, XGL_HLHSR_Z_BUFFER); three values of XGL_CTX_SURF_FRONT_FILL_STYLE (XGL_SURF_FILL_SOLID, XGL_SURF_FILL_HOLLOW, XGL_SURF_FILL_EMPTY); and three values of XGL_3D_CTX_SURF_FRONT_ILLUMINATION (XGL_ILLUM_NONE, XGL_ILLUM_PER_FACET, and XGL_ILLUM_PER_VTX). The point types used for these three illumination modes are XGL_PT_F3D, XGL_PT_COLOR_F3D, and XGL_PT_F3D respectively.

Attributes Tested:	XGL_CTX_EDGE_COLOR XGL_CTX_SURF_EDGE_FLAG XGL_FACET_NORMAL XGL_LIGHT_ENABLE_COMP_AMBIENT XGL_LIGHT_TYPE XGL_SURF_FILL_EMPTY XGL_SURF_FILL_HOLLOW XGL_SURF_FILL_SOLID and Table 33-2, Column A at the end of this chapter
Operators Tested:	xgl_object_get xgl_object_set xgl_triangle_strip
Output:	Draws a single triangle tristrip for each loop

▼ ts_solid_interp

Test Types:	INDEX, SM
Description:	Tries all point types and facet types for a triangle strip while color interpolation is on and the fill style is solid. The raster color type is INDEX. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.
Attributes Tested:	XGL_CTX_BACKGROUND_COLOR XGL_DRAW_EDGE XGL_DRAW_PREV_EDGE XGL_HLHSR_NONE XGL_SURF_FRONT_FILL_STYLE and Table 33-3, Column C at the end of this chapter
Operators Tested:	xgl_object_get xgl_object_set xgl_triangle_strip

Output: Draws several renditions of a triangle strip with two solid facets. Should render two triangles with each loop.

▼ ts_solid_interp_rgb

Test Types: RGB, SM

Description: Tries all point types and facet types for a triangle strip while color interpolation is on and the fill style is solid. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.

Attributes Tested:
XGL_CTX_BACKGROUND_COLOR
XGL_DRAW_EDGE
XGL_DRAW_PREV_EDGE
XGL_HLHSR_NONE
XGL_SURF_FRONT_FILL_STYLE
and Table 33-3, Column C at the end of this chapter

Operators Tested:
xgl_object_get
xgl_object_set
xgl_triangle_strip

Output: Draws several renditions of a triangle strip with two solid facets. Should render two triangles with each loop.

▼ ts_solid_no_illum

Test Types: INDEX, SM

Description: Tries all point types and facet types for a triangle strip while illumination is none and the fill style is solid. The raster color type is INDEX. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D,

XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.

Attributes Tested: See Table 33-3, Column B at the end of this chapter.

Operators Tested: xgl_object_get
xgl_object_set
xgl_triangle_strip

Output: Draws several renditions of a triangle strip with two solid facets. Should render two triangles with each loop.

▼ ts_solid_no_illum_rgb

Test Types: RGB, SM

Description: Tries all point types and facet types for a triangle strip while illumination is none and the fill style is solid. The raster color type is RGB. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.

Attributes Tested: See Table 33-3, Column B at the end of this chapter.

Operators Tested: `xgl_object_get`
`xgl_object_set`
`xgl_triangle_strip`

Output: Draws several renditions of a triangle strip with two solid facets. Should render two triangles with each loop.

▼ ts_solid_per_facet

Test Types:	INDEX, SM
Description:	Tries all point types and facet types for a triangle strip while illumination is per facet and the fill style is solid. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.
Attributes Tested:	XGL_CTX_BACKGROUND_COLOR XGL_CTX_SURF_FRONT_COLOR XGL_DRAW_EDGE XGL_SURF_FRONT_FILL_STYLE and Table 33-1, Column C at the end of this chapter
Operators Tested:	xgl_object_get xgl_object_set xgl_triangle_strip
Output:	Draws several renditions of a triangle strip with two solid facets. Should render two triangles with each loop.

▼ ts_solid_per_facet_rgb

Test Types:	RGB, SM
Description:	Tries all point types and facet types for a triangle strip while illumination is per facet and the fill style is solid. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and

	three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.
Attributes Tested:	XGL_CTX_BACKGROUND_COLOR XGL_CTX_SURF_FRONT_COLOR XGL_DRAW_EDGE XGL_SURF_FRONT_FILL_STYLE and Table 33-1, Column C at the end of this chapter
Operators Tested:	xgl_object_get xgl_object_set xgl_triangle_strip
Output:	Draws several renditions of a triangle strip with two solid facets. Should render two triangles with each loop.

▼ ts_solid_per_vtx

Test Types:	INDEX, SM
Description:	Tries all point types and facet types for a triangle strip while illumination is per vertex and the fill style is solid. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.
Attributes Tested:	XGL_DEV_COLOR_MAP XGL_CTX_BACKGROUND_COLOR XGL_CTX_SURF_FRONT_COLOR XGL_DRAW_EDGE XGL_SURF_FRONT_FILL_STYLE and Table 33-1c Column C at the end of this chapter
Operators Tested:	xgl_object_get xgl_object_set xgl_triangle_strip

Output:	Draws several renditions of a triangle strip with two solid facets. Should render two triangles with each loop.

▼ ts_solid_per_vtx_rgb

Test Types:	RGB, SM
Description:	Tries all point types and facet types for a triangle strip while illumination is per vertex and the fill style is solid. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.
Attributes Tested:	XGL_CTX_BACKGROUND_COLOR XGL_CTX_SURF_FRONT_COLOR XGL_DRAW_EDGE XGL_SURF_FRONT_FILL_STYLE and Table 33-1, Column C at the end of this chapter
Operators Tested:	xgl_object_get xgl_object_set xgl_triangle_strip
Output:	Draws several renditions of a triangle strip with two solid facets. Should render two triangles with each loop.

▼ ts_xform_no_illum

Test Types:	INDEX, SM
Description:	Tries all point types and facet types for a triangle strip with no illumination, solid fill style, and nonidentity view transform. The raster color type is INDEX. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D,

	XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.
Attributes Tested:	See Table 33-3, Column A at the end of this chapter.
Operators Tested:	xgl_object_get xgl_object_set xgl_triangle_strip xgl_object_create
Output:	Draws several renditions of a triangle strip with two solid facets. Should render two triangles with each loop.

▼ ts_xform_no_illum_rgb

Test Types:	RGB, SM
Description:	Tries all point types and facet types for a triangle strip with no illumination, solid fill style, and nonidentity view transform. The raster color type is RGB. Tests loops through two values of XGL_3D_CTX_HLHSR_MODE (XGL_HLHSR_Z_BUFFER, and the default value XGL_HLHSR_NONE); nine point types (XGL_PT_F3D, XGL_PT_COLOR_F3D, XGL_PT_NORMAL_F3D, XGL_PT_COLOR_NORMAL_F3D, XGL_PT_FLAG_F3D, XGL_PT_COLOR_FLAG_F3D, XGL_PT_NORMAL_FLAG_F3D, XGL_PT_COLOR_NORMAL_FLAG_F3D, XGL_PT_F3H); and three facet types (XGL_FACET_NONE, XGL_FACET_COLOR, XGL_FACET_COLOR_NORMAL), with facet type the innermost loop.
Attributes Tested:	See Table 33-3, Column A at the end of this chapter.
Operators Tested:	xgl_object_get xgl_object_set xgl_triangle_strip xgl_object_create
Output:	Draws several renditions of a triangle strip with two solid facets. Should render two triangles with each loop.

Table 33-1 Tristrip Attributes Tested - Set 1

Column A	Column B	Column C
XGL_3D_CTX_SURF_BACK_COLOR	XGL_3D_CTX_SURF_FRONT_ ILLUMINATION	XGL_3D_CTX_LIGHTS
XGL_3D_CTX_SURF_BACK_FILL_ STYLE	XGL_CTX_BACKGROUND_COLOR	XGL_3D_CTX_LIGHT_NUM
XGL_3D_CTX_SURF_FACE_CULL	XGL_CTX_EDGE_COLOR	XGL_3D_CTX_LIGHT_SWITCHES
XGL_3D_CTX_SURF_FACE_ DISTINGUISH	XGL_CTX_SURF_EDGE_FLAG	XGL_3D_CTX_SURF_FRONT_ AMBIENT
XGL_CTX_BACKGROUND_COLOR	XGL_CTX_SURF_FRONT_COLOR	XGL_3D_CTX_SURF_FRONT_ ILLUMINATION
XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_SURF_FRONT_FILL_ STYLE	XGL_CTX_BACKGROUND_COLOR
XGL_CTX_SURF_FRONT_FILL_ STYLE	XGL_DRAW_EDGE	XGL_3D_CTX_SURF_FRONT_LIGHT _COMPONENT
XGL_CULL_BACK	XGL_DRAW_PREV_EDGE	XGL_CTX_EDGE_COLOR
XGL_CULL_FRONT	XGL_SURF_FILL_EMPTY	XGL_CTX_SURF_EDGE_FLAG
XGL_CULL_NONE	XGL_CTX_SURF_FRONT_COLOR	XGL_DRAW_PREV_EDGE
XGL_CULL_OFF	XGL_CTX_SURF_FRONT_FILL_ STYLE	XGL_ILLUM_PER_FACET
XGL_SURF_FILL_SOLID	XGL_DRAW_EDGE	XGL_LIGHT_AMBIENT
		XGL_LIGHT_COLOR
		XGL_LIGHT_ENABLE_COMP_ AMBIENT
		XGL_LIGHT_TYPE
		XGL_SURF_FILL_EMPTY

Table 33-2 Tristrip Attributes Tested - Set 2

Column A	**Column B**
XGL_3D_CTX_HLHSR_MODE	XGL_3D_CTX_HLHSR_MODE
XGL_3D_CTX_LIGHTS	XGL_3D_CTX_SURF_BACK_COLOR
XGL_3D_CTX_LIGHT_NUM	XGL_3D_CTX_SURF_BACK_FILL_ STYLE
XGL_3D_CTX_LIGHT_SWITCHES	XGL_3D_CTX_SURF_BACK_ ILLUMINATION
XGL_3D_CTX_LINE_COLOR_INTERP	XGL_3D_CTX_SURF_FACE_CULL
XGL_3D_CTX_SURF_FRONT_AMBIENT	XGL_3D_CTX_SURF_FACE_ DISTINGUISH
XGL_3D_CTX_SURF_FRONT_ILLUMINATION	XGL_3D_CTX_SURF_FRONT_ ILLUMINATION
XGL_CTX_BACKGROUND_COLOR	XGL_CTX_BACKGROUND_COLOR
XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT	XGL_CTX_SURF_FRONT_COLOR
XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_SURF_FRONT_FILL_ STYLE
XGL_CTX_SURF_FRONT_FILL_STYLE	XGL_CULL_BACK
XGL_GCACHE	XGL_CULL_FRONT
XGL_HLHSR_Z_BUFFER	XGL_CULL_OFF
XGL_ILLUM_NONE	XGL_FACET_NORMAL
XGL_ILLUM_PER_FACET	XGL_GCACHE
XGL_ILLUM_PER_VERTEX	XGL_HLHSR_Z_BUFFER
XGL_LIGHT_AMBIENT	XGL_ILLUM_NONE
XGL_LIGHT_COLOR	XGL_SURF_FILL_SOLID
XGL_LIGHT_ENABLE_COMP_AMBIENT	
XGL_LIGHT_TYPE	
XGL_SURF_FILL_EMPTY	
XGL_SURF_FILL_HOLLOW	
XGL_SURF_FILL_SOLID	

Table 33-3 Tristrip Attributes Tested - Set 3

Column A	Column B	Column C
XGL_3D_CTX_SURF_FRONT_ILLUMINATION	XGL_3D_CTX_SURF_FRONT_ILLUMINATION	XGL_3D_CTX_HLHSR_MODE
XGL_CTX_VIEW_TRANS	XGL_CTX_BACKGROUND_COLOR	XGL_3D_CTX_SURF_FRONT_ILLUMINATION
XGL_CTX_BACKGROUND_COLOR	XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_NEW_FRAME_ACTION
XGL_CTX_SURF_FRONT_COLOR	XGL_CTX_SURF_FRONT_FILL_STYLE	XGL_CTX_NEW_FRAME_CLEAR
XGL_DRAW_EDGE	XGL_DRAW_EDGE	XGL_CTX_NEW_FRAME_HLHSR_ACTION
XGL_DRAW_PREV_EDGE	XGL_DRAW_PREV_EDGE	XGL_CTX_SURF_FRONT_COLOR
XGL_SURF_FRONT_FILL_STYLE	XGL_SURF_FILL_HOLLOW	XGL_CTX_SURF_FRONT_FILL_STYLE
XGL_TRANS		XGL_HLHSR_Z_BUFFER
XGL_TRANS_DATA_TYPE		XGL_ILLUM_NONE
XGL_DATA_FLT		XGL_SURF_FILL_SOLID
XGL_TRANS_DIMENSION		
XGL_TRANS_3D		

Index