Caching Geometry 17

This chapter discusses the the display list capability provided by the XGL Pcache object and the geometry cache function provided by the XGL Gcache object. It includes information on the following topics:

- How to create and display geometry using the Pcache and Gcache objects
- Performance considerations for Pcache and Gcache use
- Gcache operators and attributes

Introduction to Geometry Caching in XGL

The XGL library provides applications with two basic buffering mechanisms to cache the geometric representation of data. One mechanism, the Pcache object, provides non-editable, non-hierarchical display list functionality for XGL. The Pcache object stores a sequence of primitives and relevant attributes for rendering at one time. Using the Pcache object, application programmers can write XGL code and tune it for the high performance that display lists can provide.

The second mechanism, the Gcache object, provides a facility to decompose complex primitives and accelerate their rendering through the use of simpler primitives better suited for a particular graphics device. The Gcache object reduces the complexity of a primitive by allowing XGL to process the primitive into many simple primitives, storing the relevant attribute values. For example, a Gcache could contain the polylines that comprise a string of stroke text, or the triangles that make up a polygon. Both Pcache and Gcache objects provide a mechanism to store graphics primitives that are rendered frequently, thus yielding higher performance.

The XGL library allows applications to choose between immediate mode, Pcache programming, or Gcache programming. It permits the application to manage its graphics geometry and to decide what information is kept in the Pcache or Gcache object without imposing constraints. Pcache or Gcache programming is slightly different than immediate mode XGL programming. Primitives are rendered into these objects rather than directly to the display device. When using a Pcache or Gcache object, the application follows these general steps:

1. Create a Context object.
2. Create a Pcache or Gcache object. If the object is a Pcache object, associate it with the Context object.
3. Render the geometry into the Pcache or Gcache.
4. Display the Pcache or Gcache.

The following sections discuss the characteristics and use of these objects in more detail.

Pcache Object

A Primitive Cache (Pcache) is an XGL object that stores a sequence of zero or more XGL primitives and/or attribute settings. Pcache contents can be stored in a device dependent format in order to maximize display speed.

The use of Pcache objects optimizes performance for most 3D graphics applications running on graphics adaptors with display list capabilities. A Pcache can be rendered with greater speed, in general, than its individual components can. Several characteristics of the Pcache contribute to its optimizing effect:

- Once a Pcache is created, it cannot be modified. This reduces overhead that would be associated with searching an editable cache, and with memory management.

- Commands are stored in a format that is optimal for the graphics hardware. Optimizing commands for the particular hardware increases performance in either a networked or a local environment.
- Values of coordinates and variables are copied into the Pcache when it is compiled. This saves on processing overhead. The more times a Pcache is displayed, the greater the performance payoff.

Using Pcache Objects With Immediate Mode Applications

Many application developers may be using display list programming models even though their code is written in immediate mode. For example, an immediate mode application may use a preformatted display list that contains all the data associated with the current model. Immediate mode applications like this, and applications rendering geometry that does not change from frame to frame, could consider using Pcache objects to gain in rendring performance.

Applications that can benefit from Pcache programming range from visual simulation to mechanical computer-aided engineering to seismic interpretation. A good example of an immediate mode application is a visual simulation application, where an entire scene is rendered per frame. In this type of application, what changes from scene to scene is the relative position of each object and the point of view. Single objects in the scene could easily be encapsulated into a Pcache. For example, in a driving simulation application using an outdoor scene, mountains could be put into one Pcache, a group of buildings in another, and each moving vehicle in its own Pcache. This could dramatically speed up the process of getting all the data transformed for use in the graphics system. Each Pcache could have many computationally intensive operations done before display time, thereby achieving a significant performance improvement.

Performance in these applications can be maximized by knowing how and when to use a Pcache. As the Pcache can be used in conjunction with immediate mode commands, there is no associated penalty in moving to the Pcache model of programming.

Pcache Performance Considerations

There are some important performance trade-offs that should be considered when using Pcache functionality. In immediate mode, geometries are rendered as they are defined. When using Pcache, geometries are first rendered into the Pcache object and later displayed using the `xgl_pcache_display()` function. There is a one-time build cost associated with the Pcache. The greater the number of times a Pcache is displayed, the lower the impact of this overhead on application performance. In this case, the optimizations described above give better performance than that available through immediate mode.

Creating and Rendering to a Pcache

A Pcache object is created with the `xgl_object_create()` operator, using `XGL_PCACHE` as the *type* parameter value, as in the following example:

```
pcache = xgl_object_create(sys_st, XGL_PCACHE, NULL, NULL);
```

To render into the Pcache, the Pcache object must be associated with a 2D or 3D Context object using the `XGL_PCACHE_CONTEXT` attribute. When `XGL_PCACHE_CONTEXT` is set, the Pcache is emptied, even if the setting is redundant. The default value of `XGL_PCACHE_CONTEXT` is `NULL`.

```
xgl_object_set(pcache, XGL_PCACHE_CONTEXT, ctx, NULL);
```

When the Pcache and the Context are associated, the geometry is rendered using the Pcache object in place of the Context object. After the Pcache is created and bound to a Context and its device, the XGL primitives listed below may render to the Pcache:

- `xgl_multimarker()`
- `xgl_multipolyline()`
- `xgl_stroke_text()`
- `xgl_multi_simple_polygon()`
- `xgl_triangle_list()`
- `xgl_quadrilateral_mesh()`

The attributes listed in Table 17-1 can be rendered by the `xgl_object_set()` command to a Pcache object. The first group, Context line attributes, set line style and line color. Context marker attributes control the appearance of markers, and the Context surface attributes define many aspects of surface rendering. Picking may be done on the contents of the Pcache.

Table 17-1 Context Attributes for the Pcache Object

Attribute type	Attributes
Context line attributes	XGL_CTX_LINE_COLOR XGL_CTX_LINE_STYLE XGL_CTX_LINE_WIDTH_SCALE_FACTOR XGL_CTX_LINE_COLOR_SELECTOR
Context marker attribute	XGL_CTX_MARKER_COLOR_SELECTOR
Context surface attributes	XGL_3D_CTX_SURF_FACE_CULL XGL_CTX_SURF_FRONT_COLOR XGL_CTX_SURF_FRONT_COLOR_SELECTOR XGL_3D_CTX_SURF_FRONT_ILLUMINATION XGL_3D_CTX_SURF_BACK_COLOR XGL_3D_CTX_SURF_BACK_COLOR_SELECTOR XGL_3D_CTX_SURF_BACK_ILLUMINATION XGL_3D_CTX_SURF_GEOM_NORMAL
Picking attributes	XGL_CTX_PICK_ID_1 XGL_CTX_PICK_ID_2

When these attributes are applied to a Pcache, they are recorded in a device-specific format. At a later time, the `xgl_pcache_display()` operator can display its contents. Once `xgl_pcache_display()` is called, the Pcache is closed, and the attributes may no longer be changed.

An application can empty (and re-open) a Pcache by applying the operator `xgl_context_new_frame()`. A Pcache is also emptied whenever it is associated with another Context or its Context is associated with another device. It is an error to use the Pcache (except to destroy or re-associate it) if either its Context or Device has been destroyed.

Displaying a Pcache

The application displays the Pcache using the `xgl_pcache_display()` operator. This function performs four tasks:

1. It closes the Pcache.
2. It compares the Pcache and Context states.
3. If Context state and Pcache state match, it displays the Pcache.
4. It returns the result of the comparison.

```
Xgl_cache_display
xgl_pcache_display(
    Xgl_pcache      pcache,
    Xgl_boolean     test,
    Xgl_boolean     display,
    Xgl_boolean     restore);
```

When a Pcache is built, XGL may make assumptions about the state of the Context to which it is attached. At display time, the application may test these assumptions to verify that they are still valid. If so, the Pcache is rendered. If not, the application is informed that the Pcache needs to be rebuilt. The parameter `test` controls the validation of the Pcache and Context states. If `test` is `FALSE`, the Context state is not validated and the return value is `XGL_CACHE_NOT_CHECKED`. If `test` is `TRUE`, the Context state is compared with that of the Pcache. If the state is compatible, the return value is `XGL_CACHE_DISPLAY_OK`; otherwise, it is `XGL_CACHE_ATTR_STATE_DIFFERENT`.

The `display` parameter controls the display of the Pcache. If `display` is `FALSE`, the Pcache is not rendered. If `display` is `TRUE`, the Pcache is rendered only if test is `FALSE` or the Context state is compatible.

The `restore` parameter controls whether or not Context attributes can be altered by displaying the Pcache. If `restore` is `TRUE`, displaying a Pcache will not change any of the Context attributes. If `restore` is `FALSE` and the Pcache contains attributes, the attributes of the Context are changed to reflect those stored in the Pcache.

The application can remove the Pcache with the the `xgl_object_destroy()` function. The code fragment below shows the creation and use of a Pcache object.

```
(...parameter definition and other code omitted...)

/* Create the context object */
ctx = xgl_object_create(sys_state,
          XGL_3D_CTX, (Xgl_obj_desc *) 0,
          XGL_CTX_DEVICE, ras,
          NULL);

/* Create the Pcache object */
pcache = xgl_object_create(sys_state, XGL_PCACHE, NULL,
          NULL);

/* Associate the Pcache with the ctx */
xgl_object_set(pcache, XGL_PCACHE_CONTEXT, ctx, NULL);

xgl_context_new_frame(ctx);

create_tristrip(&pl, &fl);

/* Render the triangle strip to the Pcache */
xgl_triangle_list(pcache, &fl, &pl,
          XGL_TLIST_FLAG_TRI_STRIP);

/* Now render everything in the Pcache to the ctx */
xgl_pcache_display(pcache, FALSE, TRUE, FALSE);

/* Destroy the Pcache */
xgl_object_destroy(pcache);
```

17

Gcache Object

A Gcache object can be used to to break down complex objects into simpler forms, resulting in increased rendering performance (if the underlying accelerator is optimized for the resulting decomposed primitive). A Gcache also enables an application to instruct XGL to model clip or decompose a primitive once, and then render the result many times.

Once a Gcache object is created, an application can use the Gcache operators to store a primitive into the Gcache. For example, `xgl_gcache_polygon()`, which is used to store a polygon in a Gcache, is the Gcache counterpart of `xgl_polygon()`. The primitive can be displayed repeatedly, usually with better performance than its non-Gcache counterpart, by calling `xgl_context_display_gcache()`.

Whenever an object is stored in a Gcache, certain Context attributes (which depend on the primitive being stored) will be saved in the Gcache. If a primitive is stored in a Gcache, and any relevant context attribute changes, the properties of the cached primitive will be outdated. When this happens, the application can decide if the primitive should still be displayed. Many context attributes can be changed after a primitive has been stored in a Gcache without outdating the primitive. For example, view clipping attributes can be changed, and the primitive will still be rendered correctly.

A Gcache copies the original geometry into internal storage in Model Coordinates. For some reductions (polygon decomposition, for example), it is possible for the Gcache to reference the original geometry. This is useful for saving memory.

When To Use a Gcache

Many graphics hardware accelerators are efficient with simple geometries. For example, an accelerator may draw triangles very fast. If an application wants to render the same complex polygon several times, it is worthwhile to use a Gcache to tessellate the polygon into triangles and send the triangles to the accelerator for each rendering. If a Gcache was not used, then the polygon tessellation would have to be done each time the primitive was rendered. Additionally, normals will be calculated (assuming they were not provided by the application) and stored in the facet structure. This saves the Gcache operator from having to calculate normals every time the primitive is rendered.

For stroke text, a Gcache uses the character attributes (character spacing, height, and so on) to convert the text to a set of polylines for faster rendering.

For NURBS curves and surfaces, much of the processing time is absorbed during Gcache creation time. This substantially speeds up display of NURBS curves and surfaces.

Creating a Gcache

A Gcache object is created with the `xgl_object_create()` operator, using `XGL_GCACHE` as the *type* parameter value, as in the following example:

```
gcache = xgl_object_create(sys_st, XGL_GCACHE, NULL, NULL);
```

Rendering to a Gcache

The Gcache object has its own set of operators. These operators use the same data structures as the standard XGL drawing primitives. To render into the Gcache, the application uses a Gcache operator and specifies the Context. The Gcache primitives are:

`xgl_gcache_multi_simple_polygon()`
This operator stores multisimple polygons in a Gcache. If model clipping is turned on (that is, if `XGL_3D_CTX_MODEL_CLIP_PLANE_NUM` is greater than 0), then the polygons are stored after being model clipped. The polygons are checked to see if they are convex. If a non-convex polygon is found and `XGL_GCACHE_DO_POLYGON_DECOMP` is `TRUE`, XGL tessellates the polygon into triangles. If facet normals are not provided by the application, they are calculated and stored in the Gcache. The tessellation edges can be seen if `XGL_CTX_SURF_EDGE_FLAG` is `TRUE` and `XGL_GCACHE_SHOW_DECOMP_EDGES` is `TRUE`.

`xgl_gcache_multi_elliptical_arc()`
This operator caches the polyline representation of 3D elliptical arcs in the specified Gcache.

`xgl_gcache_multimarker()`
This operator stores markers in the specified Gcache.

`xgl_gcache_multipolyline()`

This operator stores 3D polylines and their bounding boxes in the specified Gcache.

`xgl_gcache_nurbs_curve()`

Depending on the value of the Gcache attribute `XGL_GCACHE_NURBS_CURVE_MODE`, the `xgl_gcache_nurbs_curve()` operator stores a NURBS curve in a tessellation-independent form, stores the curve in a tesselated polyline representation, or stores both forms of the curve.

If the value of `XGL_GCACHE_NURBS_CURVE_MODE` is `XGL_GCACHE_NURBS_DYNAMIC`, a tessellation-independent form of the curve is created from the NURBS input parameters, and this form is dynamically tessellated during display. If the value of `XGL_GCACHE_NURBS_CURVE_MODE` is `XGL_GCACHE_NURBS_STATIC`, the curve is tessellated at Gcache creation time, and the polyline representation is stored. If the value of `XGL_GCACHE_NURBS_CURVE_MODE` is `XGL_GCACHE_NURBS_COMBINED`, both the tessellation-independent form of the curve and the static polyline form of the NURBS curve are stored. Portions of the curve are tessellated only if it is necessary according to the current Context; otherwise, the cached tessellation is reused. The application should choose the appropriate representation mode depending on its needs. Refer to the section “Gcache Attributes” on page 448 for a detailed discussion of the `XGL_GCACHE_NURBS_CURVE_MODE` attribute and the three representation modes.

`xgl_gcache_nurbs_surface()`

Depending on the value of the Gcache attribute `XGL_GCACHE_NURBS_SURF_MODE`, the `xgl_gcache_nurbs_surface()` operator stores a NURBS surface in a tessellation-independent form, stores the surface in a tesselated polyline representation, or stores both forms of the surface. The values of the `XGL_GCACHE_NURBS_SURF_MODE` attribute are similar to those of the NURBS curve `XGL_GCACHE_NURBS_CURVE_MODE` attribute.

If the value of `XGL_GCACHE_NURBS_SURF_MODE` is `XGL_GCACHE_NURBS_DYNAMIC`, a tessellation-independent form of the surface is cached, and this form is dynamically tessellated during display. If the value is `XGL_GCACHE_NURBS_STATIC`, the surface is tessellated, and the tessellation is cached. Tessellation consists of a list of triangle lists and/or quad meshes. If isoparametric lines are drawn, they are cached as a list of polylines. If the value is XGL_GCACHE_NURBS_COMBINED, both forms are cached. Tessellation is recreated where necessary. As in the case of NURBS curves, the application should carefully choose the appropriate representation mode depending on its needs. Refer to the section "Gcache Attributes" on page 448 for a more detailed discussion of the three representation modes.

xgl_gcache_polygon()
This operator stores 3D polygons in the specified Gcache. If XGL_GCACHE_DO_POLYGON_DECOMP is TRUE, the polygon is broken down into a list of triangles and stored in the Gcache. Additionally, if a facet normal was not provided, the operator will calculate the normal to store in the Gcache.

Hollow polygons or polygons with edges turned on are drawn using the original edges. The flag XGL_GCACHE_SHOW_DECOMP_EDGES has to be set to TRUE to see the tessellation.

xgl_gcache_stroke_text()
This operator stores the polyline representation of a text string in the specified Gcache. The Gcache can contain either 2D or 3D data, and this operator accepts both 2D and 3D Contexts. The attribute XGL_GCACHE_USE_APPL_GEOM is ignored for stroke text. Note that the dimension of the Context used to write text to the Gcache must match the dimension of the Context used to display it.

xgl_gcache_triangle_list()
This operator stores a 3D triangle list in the specified Gcache. If facet normals are not provided, the operator calculates the normals for each facet to store in the Gcache.

xgl_gcache_triangle_strip()
This operator stores a 3D triangle strip in the specified Gcache. If facet normals are not provided, the operator calculates the normals for each facet to store in the Gcache.

The Context attributes saved in the Gcache for various primitives are listed in the *XGL Reference Manual* under the man page of each primitive.

Gcache Attributes

The Gcache object includes the following attributes:

XGL_GCACHE_BYPASS_MODEL_CLIP
: Controls whether model clipping is performed when a Gcache is displayed. If this attribute is FALSE (default mode), model clipping (if enabled in the context) is applied to the cached geometry.

XGL_GCACHE_DISPLAY_PRIM_TYPE
: Returns the type of primitive used when the Gcache is displayed. If XGL_GCACHE_DISPLAY_PRIM_TYPE is not the original primitive type, some sort of optimization or simplification was performed on the data by XGL. For example, a complex polygon may have been tessellated into triangles, in which case xgl_multi_simple_polygon() is used for rendering.

XGL_GCACHE_ORIG_PRIM_TYPE
: Returns the type of the original primitive stored in the Gcache.

XGL_GCACHE_DO_POLYGON_DECOMP
: Controls whether polygon decomposition is performed when xgl_gcache_polygon() is called. If XGL_GCACHE_DO_POLYGON_DECOMP is set to TRUE, any subsequent polygons that are cached will be stored as a decomposed list of triangles. Note that when non-planar polygons are decomposed, the results are undefined.

XGL_GCACHE_SHOW_DECOMP_EDGES
: Controls whether decomposition edges are shown when xgl_gcache_polygon() or xgl_gcache_multi_simple_polygon() is displayed. If XGL_GCACHE_SHOW_DECOMP_EDGES and XGL_CTX_SURF_EDGE_FLAG are TRUE, and the Gcache polygon or multisimple polygon was decomposed, the edges outlining the decomposition are shown.

XGL_GCACHE_POLYGON_TYPE
: Controls which decomposition algorithm is used when xgl_gcache_polygon() is called.

XGL_GCACHE_PT_LIST_LIST
: Returns the list of point lists used when the Gcache is displayed.

`XGL_GCACHE_FACET_LIST_LIST`
Returns the list of facets used when the Gcache is displayed.

`XGL_GCACHE_IS_EMPTY`
When set to `TRUE` (default mode), this attribute empties the Gcache. Nothing is drawn when an empty Gcache is displayed. Gcache resources are not freed until the Gcache object is destroyed.

`XGL_GCACHE_USE_APPL_GEOM`
When set to `TRUE`, this attribute causes the Gcache to reference the application's geometry (parameters to the Gcache operators, such as the list of points, bounding box, and facet information) in the Gcache. When set to `FALSE` (default value), the Gcache copies the applications geometry into the Gcache.

`XGL_GCACHE_NURBS_CURVE_MODE`
Controls the Gcache mode when the `xgl_gcache_nurbs_curve()` or `xgl_gcache_multi_elliptical_arc()` operators are called. This attribute can have one of three values. If the value of `XGL_GCACHE_NURBS_CURVE_MODE` is `XGL_GCACHE_NURBS_DYNAMIC`, a tessellation-independent Gcache is created from the NURBS input parameters. At each frame, the curve is tessellated according to the current Context. To obtain dynamic tessellation, the Gcache should be displayed using one of the NURBS curve dynamic approximation types. Memory consumption is minimal in this mode because the tessellation is not stored. In addition, the quality of the displayed geometry is optimal because tessellation step sizes are dynamically computed per frame.

If the value of `XGL_GCACHE_NURBS_CURVE_MODE` is `XGL_GCACHE_NURBS_STATIC`, the curve is tessellated at Gcache creation time, and the polyline representation is stored. With this mode, display on some devices may be fast, but memory consumption may be high and the quality of the rendered primitive may be affected by extreme view changes since the stored tessellation cannot be modified. In addition, the Gcache may be invalidated if any NURBS-specific attributes (such as the approximation value) changes.

If the value of `XGL_GCACHE_NURBS_CURVE_MODE` is `XGL_GCACHE_NURBS_COMBINED`, both the tessellation-independent form of the curve and the static polyline form of the NURBS curve are stored. Portions of the curve are tessellated only if this is necessary according to the current Context; otherwise, the cached tessellation is reused. This mode offers the best aspects of the static and dynamic curve representation modes; however, memory consumption may be high.

The application should carefully consider the following when choosing a representation mode:

- Memory availability: Because the static and combined Gcache modes store tessellation, they consume more memory than the dynamic mode.
- Quality of rendered geometry: The dynamic Gcache mode will consistently provide the optimal quality based on the current approximation type, whereas with the other modes, too much or too little tessellation may be displayed.
- Speed: If the hardware device has a fast CPU, or if the device can accelerate NURBS tessellation, dynamic tessellation will be comparable in speed to displaying pre-stored tessellation. Otherwise, in some cases, the application may want to use static or combined Gcache attributes to obtain better performance.
- Complexity of NURBS data and viewing requirements: If the NURBS curve consists of a lot of Bezier points and the application is planning to allow the user to zoom in and out to look closely at the geometry, use dynamic mode. If the NURBS geometry is complex but only a fixed view of it is needed, use static mode. Static mode may store a simpler form of the geometry, thus saving memory and rendering time.

The values of the `XGL_GCACHE_NURBS_CURVE_MODE` are illustrated in Figure 17-1 on page 451.

`XGL_GCACHE_NURBS_SURF_MODE`
Controls the Gcache mode when the `xgl_gcache_nurbs_surface()` operator is called. The values for this attribute are the same as for the `XGL_GCACHE_NURBS_CURVE_MODE` attribute; refer to the preceding section for a description of the three Gcache NURBS surface representation modes and for information on choosing the appropriate mode for an application.

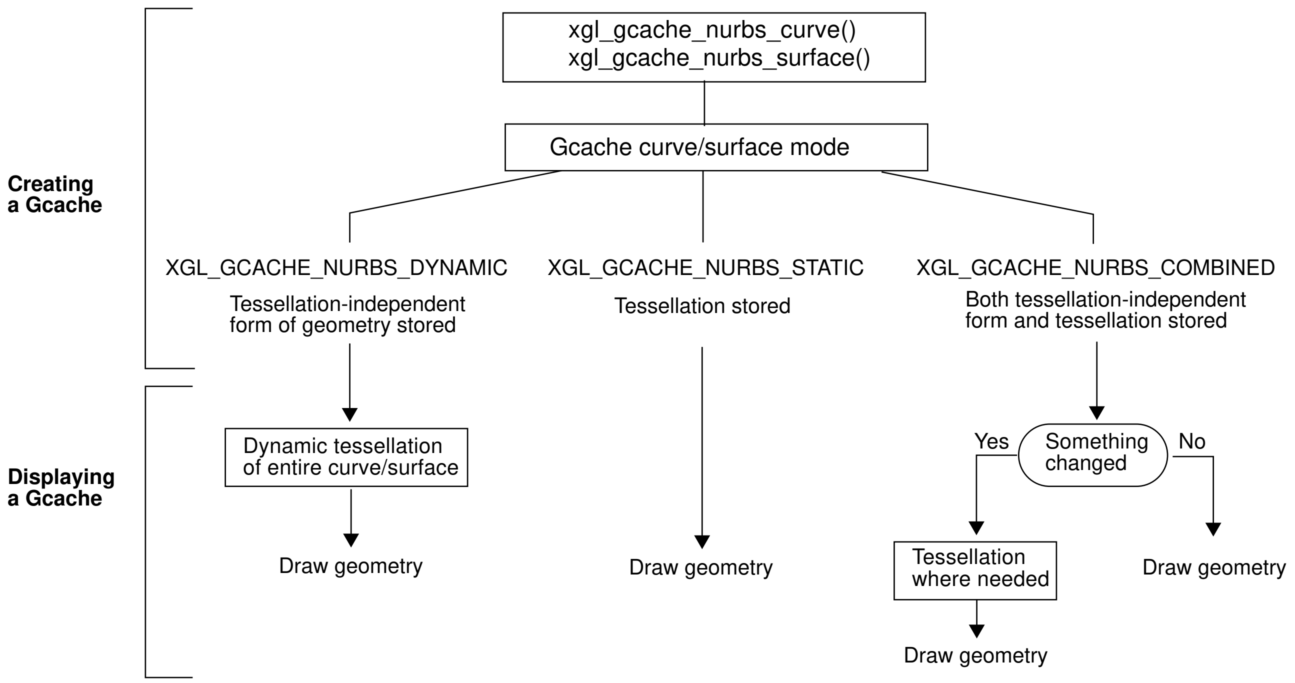

Figure 17-1 Gcache NURBS Curve and Surface Representation Modes

Displaying the Gcache

Usually an application will use one Gcache object for each primitive that needs to be cached. The primitive is rendered by calling `xgl_context_display_cache()`. This operator is used to display the Gcache on a specified device and to compare the saved state (attribute values) of the Gcache with the current state in the Context, returning the result of this comparison. This allows the application to decide if it will display the Gcache when the Gcache state and the related Context attributes differ.

The operator is defined as:

```
Xgl_cache_display
xgl_context_display_gcache(
    Xgl_ctx              ctx,
    Xgl_gcache           gcache,
    Xgl_boolean          test,
    Xgl_boolean          display);
```

The parameter `test` controls whether the state in the Gcache is validated. If `test` is `TRUE`, the saved state is compared with the current state, and `xgl_context_display_gcache()` will return either `XGL_CACHE_DISPLAY_OK` or `XGL_CACHE_ATTR_STATE_DIFFERENT`. If `test` is `FALSE`, then `xgl_context_display_gcache()` returns `XGL_GCACHE_NOT_CHECKED`.

The parameter `display` controls whether the Gcache is rendered on the device associated with the Context. If `display` is `FALSE`, the Gcache is not rendered. If `display` is `TRUE`, and `test` is `FALSE`, the Gcache is rendered. If both `display` and `test` are `TRUE`, the Gcache is only rendered if the validity test is `XGL_CACHE_DISPLAY_OK`.

Gcache Example Programs

The following example programs show a non-self-intersecting polygon being rendered with and without a Gcache, and a complex polygon being rendered with and without a Gcache. To compile the programs, type `make gcache` in the example programs directory. The complete program includes the `ex_utils.c` utility file listed in Appendix B, `gcache_main.c` listed in Appendix B, and the two Gcache example programs listed in this chapter.

Gcache of Simple Polygon

This program displays a non-self-intersecting polygon. Figure 17-2 shows the program output.

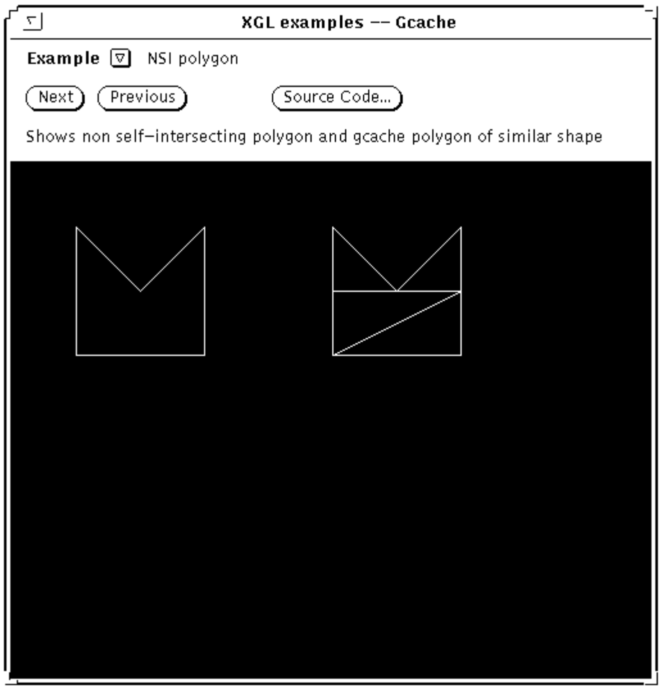

Figure 17-2 Output of `gcache_nsi_pgon.c`

Code Example 17-1 Gcache of Simple Polygon

```
/*
 * gcache_nsi_pgon.c
 */
#include "ex.h"

/*
 * Compare a non self-intersecting polygon with a gcache polygon
 * of similar shape
 */

gcache_nsi_pgon(ctx)
    Xgl_ctx    ctx;
{
    Xgl_pt_list       pl[2];
    static Xgl_pt_f3d pts_f3d[2][5] =
```

```
            {{{50, 50, 0}, {100, 100, 0}, {150, 50, 0},
            {150, 150, 0}, {50, 150, 0}}, {{250, 50, 0},
            {300, 100, 0}, {350, 50, 0}, {350, 150, 0},
            {250, 150, 0}}};
    Xgl_object          gcache;
    int                 i;
    Xgl_color           surface_color;
    Xgl_color           edge_color;

    /* Prepare data for polygon and gcache polygon */
    for (i = 0; i < 2; i++) {
        pl[i].pt_type = XGL_PT_F3D;
        pl[i].bbox = NULL;
        pl[i].num_pts = 5;
        pl[i].pts.f3d = pts_f3d[i];
    }

    xgl_context_new_frame(ctx);
    /* set surface & edge colors, turn on edges */
    surface_color = blue_color;
    edge_color = yellow_color;
    xgl_object_set(ctx,
               XGL_CTX_SURF_EDGE_FLAG, TRUE,
               XGL_CTX_EDGE_COLOR, &edge_color,
               XGL_CTX_SURF_FRONT_COLOR, &surface_color,
               NULL);



    /*
     * The polygon is drawn on the left and the gcache polygon on
     * the right
     */
    xgl_polygon(ctx, XGL_FACET_NONE, NULL, NULL, 1, &pl[0]);

    gcache = xgl_object_create(sys_st,
                 XGL_GCACHE, NULL,
                 XGL_GCACHE_DO_POLYGON_DECOMP, TRUE,
                 XGL_GCACHE_SHOW_DECOMP_EDGES, TRUE,
                 XGL_GCACHE_POLYGON_TYPE, XGL_POLYGON_NSI,
                 NULL);
    xgl_gcache_polygon(gcache, ctx, XGL_FACET_NONE, NULL, NULL, 1,
                  &pl[1]);
    xgl_context_display_gcache(ctx, gcache, FALSE, TRUE);
    xgl_context_post(ctx, TRUE);
}
```

Gcache of Complex Polygon

The following example program displays a complex self-intersecting polygon. Figure 17-3 shows the program output.

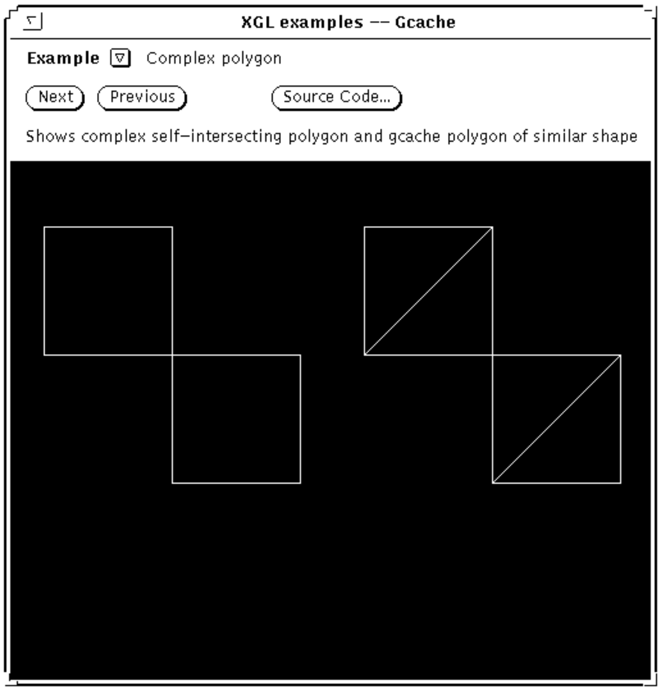

Figure 17-3 Output of `gcache_complex_pgon.c`

Code Example 17-2 Gcache of Complex Polygon

```
/*
 * gcache_complex_pgon.c
 */

#include "ex.h"

/* Compare a complex polygon with a gcache polygon of similar
 * shape */

gcache_complex_pgon(Xgl_objectctx)
```

```
{
    Xgl_pt_list          pl[2];
    static Xgl_pt_f3d    pts_f3d[2][6] =
              {{{25, 50, 0}, {125, 50, 0}, {125, 250, 0},
              {225, 250, 0}, {225, 150, 0}, {25, 150, 0}},
              {{275, 50, 0}, {375, 50, 0}, {375, 250, 0},
              {475, 250, 0}, {475, 150, 0}, {275, 150, 0}}};
    Xgl_object     gcache;
    int            i;
    Xgl_color      surface_color;
    Xgl_color      edge_color;

    /* Prepare data for polygon and gcache polygon */
    for (i = 0; i < 2; i++) {
          pl[i].pt_type = XGL_PT_F3D;
          pl[i].bbox = NULL;
          pl[i].num_pts = 6;
          pl[i].pts.f3d = pts_f3d[i];
    }

    xgl_context_new_frame(ctx);
    /* set surface & edge colors, turn on edges */
    surface_color = blue_color;
    edge_color = yellow_color;
    xgl_object_set(ctx,
              XGL_CTX_SURF_EDGE_FLAG, TRUE,
              XGL_CTX_EDGE_COLOR, &edge_color,
              XGL_CTX_SURF_FRONT_COLOR, &surface_color,
              NULL);

    /*  The polygon is drawn on the left and the gcache polygon on
     *  the right */
    xgl_polygon(ctx, XGL_FACET_NONE, NULL, NULL, 1, &pl[0]);

    gcache = xgl_object_create(sys_st,
                     XGL_GCACHE, NULL,
                     XGL_GCACHE_DO_POLYGON_DECOMP, TRUE,
                     XGL_GCACHE_SHOW_DECOMP_EDGES, TRUE,
                     XGL_GCACHE_POLYGON_TYPE,
                     XGL_POLYGON_COMPLEX,
                     NULL);
    xgl_gcache_polygon(gcache, ctx, XGL_FACET_NONE, NULL, NULL, 1,
                      &pl[1]);
    xgl_context_display_gcache(ctx, gcache, FALSE, TRUE);
    xgl_context_post(ctx, TRUE);
}
```