Contexts 7

This chapter introduces the XGL Context object. The chapter includes information on the following topics:

- Context utility operators
- Context stacks
- Environment Context attributes

More information on the Context object is provided in Chapter 8, “Primitives and Graphics Context Attributes” on page 175.

Introduction to the XGL Context Object

The Context object is an abstraction of an idealized graphics renderer that produces images on a specified Device. The Context object controls all rendering actions directed to the associated XGL Device.

The Context object contains many operators that can be divided into two general groups: *utility operators*, discussed in this chapter, and *primitives*, discussed in Chapter 8, “Primitives and Graphics Context Attributes.” The utility operators perform tasks such as clearing the Device associated with a Context. Primitives are the basic drawing operators available to an XGL application.

Context attributes maintain the state information required for XGL rendering operations. The Context attributes, like the operators, can be divided into two subsets, *graphics Context* attributes and *environment Context* attributes. The

7

graphics Context attributes directly affect the rendering of XGL primitives (for example, line color). They constitute the Context's *graphics state*. The environment Context attributes are not directly associated with the way primitives are drawn on the Device. They constitute the Context's *environment state*. Graphics state information is saved on an internal Context stack when the `xgl_context_push()` operator is invoked. Environment state information, on the other hand, cannot be pushed onto an internal stack.

Applications render graphics through the Context to a Device. Each Context is associated with a Device object that represents a display surface for all rendering operations. Although multiple Context objects can be simultaneously attached to a Device, each Context can only be associated with one Device object at a time.

This chapter discusses utility operators for Contexts, and describes environment attributes. The following chapter discusses the XGL primitive operators and introduces the graphics attributes. More detailed discussions of many of the Context graphics attributes can be found throughout this manual in sections describing the particular objects affected by the attributes.

Creating a Context Object

A Context object is created by the `xgl_object_create()` operator, using either `XGL_2D_CTX` or `XGL_3D_CTX` as the value for the parameter *type*. 2D or 3D Context attributes can be set within the create call or with a separate `xgl_object_set()` call.

```
ctx = xgl_object_create (sys_st, XGL_3D_CTX, NULL,
                         <attribute_list>,
                         NULL);
```

For rendering to occur, the application must explicitly associate a Device object with the Context object, either at Context creation time or later, using the `XGL_CTX_DEVICE` attribute and the handle to the Device object. A Device is not associated with the Context until this relationship is established.

The application can also associate the Context object with other objects that serve as resources containing information relevant for rendering. When XGL is initialized, the Context object is associated with ten Line Pattern objects containing the predefined line patterns, a Stroke Font object specifying the

default font, eight Marker objects containing the predefined marker definitions, and several Transform objects. As the application creates new objects, it must associate these objects with the Context object. This association can be made when the Context object is created or set later with the `xgl_object_set()` operator. Table 7-1 lists the objects that can be associated with the Context object and lists the linking attributes.

Table 7-1 Objects Associated with the Context Object

User Object	Used Object	Linking Attribute
2D Context and 3D Context	Window Raster Device	XGL_CTX_DEVICE
	Memory Raster Device	XGL_CTX_DEVICE
	Stream Device	XGL_CTX_DEVICE
	Transform	XGL_CTX_GLOBAL_MODEL_TRANS XGL_CTX_LOCAL_MODEL_TRANS XGL_CTX_MODEL_TRANS XGL_CTX_VIEW_TRANS XGL_CTX_MC_TO_DC_TRANS
	Line Pattern	XGL_CTX_LINE_PATTERN XGL_CTX_EDGE_PATTERN
	Marker	XGL_CTX_MARKER
	Memory Raster	XGL_CTX_RASTER_FPAT XGL_CTX_SURF_FRONT_FPAT
	Stroke Font	XGL_CTX_SFONT_0 XGL_CTX_SFONT_1 XGL_CTX_SFONT_2 XGL_CTX_SFONT_3
3D Context only	Transform (3D)	XGL_3D_CTX_NORMAL_TRANS
	Memory Raster	XGL_3D_CTX_SURF_BACK_FPAT
	Light	XGL_3D_CTX_LIGHTS
	Data Map Texture	XGL_3D_CTX_SURF_FRONT_DMAP XGL_3D_CTX_SURF_BACK_DMAP
	Texture Map	XGL_3D_CTX_SURF_FRONT_TMAP XGL_3D_CTX_SURF_BACK_TMAP

The following code fragment assumes that an application has already created a Raster Device object, two Stroke Font objects, and a Marker object and shows a Context object being associated with these objects using `xgl_object_set()`.

```
xgl_object_set(ctx,
               XGL_CTX_DEVICE,                ras,
               XGL_CTX_SFONT_0,               strokefont,
               XGL_CTX_SFONT_1,               strokefont1,
               XGL_CTX_MARKER,                appl_marker,
               XGL_CTX_MARKER_SCALE_FACTOR,   15.0,
               NULL);
```

Context Object Operators

This section presents information on the Context object operators that are not graphics drawing primitives. Table 7-2 summarizes these operators. For more information on these operators, see the *XGL Reference Manual*.

Table 7-2 Context Object Utility Operators

Operator	Description
`xgl_context_new_frame()`	Clears the Device Coordinate viewport associated with the specified Context (see Chapter 10,"View Model"). The operator can clear the Z-buffer (reset all Z values to a given value) when using a 3D Context, switch hardware display buffers in a multiple buffer configuration, or synchronize to the Device's vertical retrace.
`xgl_context_post()`	Causes any pending graphics primitives for the given Context to be posted. This operator is superseded by `xgl_context_flush()` and is available to provide backward compatibility.
`xgl_context_flush()`	Causes any pending graphics primitives for the given Context to be posted or any asynchronous processing to conclude. Depending on the value of the `flush_action` flag passed as a parameter with the call, this operator can flush the XGL system's use of an application's data or its internal buffers. It can also ensure synchronization of XGL and device processing by blocking XGL processing until the rendering device has posted its buffers.

Table 7-2 Context Object Utility Operators (Continued)

Operator	Description
`xgl_context_push()`	Saves the specified attributes from the current graphics state of the Context onto an internal stack. (Each Context has its own internal stack.) If the list of specified attributes is `NULL`, all pushable attributes associated with the Context are saved.
`xgl_context_pop()`	Restores the most recently pushed Context state from the internal stack of the given Context. All the attributes pushed by the most recent call to the `xgl_context_push()` operator are restored in the Context.
`xgl_context_set_pixel()`	Sets the color value of a pixel at a specified position in a buffer of the Raster associated with the given Context. The buffer is specified by the Context attribute `XGL_CTX_RENDER_BUFFER` and can be an image buffer or a Z-buffer.
`xgl_context_set_multi_pixel()`	Sets the color value of a list of pixels at specified positions in the specified buffer of the Raster associated with the given Context. The buffer is specified by the Context attribute `XGL_CTX_RENDER_BUFFER` and can be an image buffer or a Z-buffer.
`xgl_context_set_pixel_row()`	Sets the color value of a row of pixels in the specified buffer of the Raster associated with the given Context. The buffer is specified by the Context attribute `XGL_CTX_RENDER_BUFFER` and can be an image buffer or a Z-buffer.
`xgl_context_get_pixel()`	Gets the color value of a pixel at a specified position in a buffer of the Raster (Device) associated with the given Context. The buffer is specified by the Context attribute `XGL_CTX_RENDER_BUFFER`.
`xgl_context_copy_buffer()`	Copies a rectangular block of pixels from a buffer in the source Raster to one or more of the Raster buffers associated with the destination Context. The source buffer is specified by the attribute `XGL_RAS_SOURCE_BUFFER`. The destination buffer is specified by the Context attribute `XGL_CTX_RENDER_BUFFER`. Copying is only supported between buffers of the same type.
`xgl_image()`	Displays a block of pixels from an XGL Raster.
`xgl_context_update_model_trans()`	Used to perform fast push and pop operations involving the model transforms.

7

Context Stacks

There are two types of stacks for XGL Contexts. The first is a general stack that is manipulated with the `xgl_context_push()` and `xgl_context_pop()` operators. This type of stack is allocated in main memory, and there is generally no limit to the number of successive `xgl_context_push()` calls that an application program can make. Each `xgl_context_push()` can save the value of a single Context attribute or the entire Context state.

A list of the attributes this operator can save is located in the *XGL Reference Manual*, under the `xgl_context_push()` operator. When the application program invokes the `xgl_context_pop()` operator, the last set of values saved by the most recent call to `xgl_context_push()` is extracted from an internal stack into the Context state.

In some cases, faster push and pop operations are needed—for example, when traversing a list of structures, involving the modification of the model transforms of the Context, to animate a set of objects in an XGL Window Raster. In such cases, the `xgl_context_update_model_trans()` operator is best suited. It can push or pop the model transforms of the Context, using a different Context stack than the one used by the more general `xgl_context_push()` and `xgl_context_pop()` operators. This special stack must be initialized by the application before using `xgl_context_update_model_trans()`. The Context attribute `XGL_CTX_MODEL_TRANS_STACK_SIZE` serves that purpose. (See the *XGL Reference Manual* for more detail on how to use the `xgl_context_update_model_trans()` operator.)

Environment Context Attributes

This section provides an overview of the environment Context attributes. None of these attributes can be pushed by `xgl_context_push()`. For information on the graphics Context attributes, see "Graphics Context Attributes" on page 191."

`XGL_CTX_DEFERRAL_MODE`

Defines the *deferral mode* of a Context. It controls whether the data sent to the Device are buffered (held temporarily) or rendered immediately. The deferral modes are:

- XGL_DEFER_ASTI (At Some TIme). In this mode, XGL keeps an internal buffer of primitives, rendering them all when the buffer is full. The application must call xgl_context_post to see the buffered primitives rendered on the Device.
- XGL_DEFER_ASAP (As Soon As Possible). In this mode, XGL renders a primitive as soon as it is received.

The default value is XGL_DEFER_ASTI.

XGL_CTX_DEVICE
Defines the Device associated with a given Context. The device can be a Window Raster or Memory Raster. It is the drawing surface for all following graphics operations for that Context. The default value is NULL.

XGL_CTX_RENDER_BUFFER
Controls where primitives are rendered on the Device associated with the Context. Possible values are:

- XGL_RENDER_DRAW_BUFFER. Primitive operations are rendered on the raster's draw buffer.
- XGL_RENDER_DISPLAY_BUFFER. Primitive operations are rendered on the raster's display buffer when the raster is a double-buffered window raster.
- XGL_RENDER_Z_BUFFER. This value is used only during raster operations and is only effective when the XGL_3D_CTX_HLHSR_MODE is set to XGL_HLHSR_Z_BUFFER.

The default value is XGL_RENDER_DRAW_BUFFER. Note that it is not advisable to use this attribute for operators other than xgl_context_copy_buffer().

XGL_CTX_RENDERING_ORDER
Allows a device to set the rendering order of primitives. Applications can use this attribute to batch primitives in the most efficient order for a device. If the rendering order is XGL_RENDERING_ORDER_GIVEN, no reordering of primitives occurs. If the rendering order is XGL_RENDERING_ORDER_HLHSR, the accelerator can change the rendering order only if XGL_3D_CTX_HLHSR_MODE is set to XGL_HLHSR_Z_BUFFER. If the rendering order is set to XGL_RENDERING_ORDER_ANY, the device can reorder the rendering of the primitives. The default value is XGL_RENDERING_ORDER_GIVEN.

XGL_CTX_GEOM_DATA_IS_VOLATILE
When set to FALSE, this attribute indicates that the application program guarantees that the geometry data will not be modified or destroyed until a subsequent xgl_context_flush() call. The default value for this attribute is TRUE.

Context Attributes for Picking

The following environment Context attributes specify how picking is handled. For more information on picking, see Chapter 15, "Picking".

XGL_CTX_PICK_BUFFER_SIZE
Sets the size of the Context pick buffer. The value is the number of *Xgl_pick_info* structures that the internal pick buffer will hold. The *Xgl_pick_info* structures hold picking identifiers that identify the picked primitives. The default value is 256.

XGL_CTX_PICK_ENABLE
Turns picking on (TRUE) or off (FALSE, the default value).

XGL_CTX_PICK_STYLE
Controls how the pick events (pick identifier pairs) are stored in the Context pick buffer. If the value of this attribute is XGL_PICK_FIRST_N, the first N (where N is the size of the pick buffer) pick events are stored in the buffer, all other pick events are not buffered. If the value of this attribute is XGL_PICK_LAST_N, the events are stored in a first-in-first-out (FIFO) manner. This attribute can only be changed when the attribute XGL_CTX_PICK_ENABLE is FALSE. The default value is XGL_PICK_LAST_N.

XGL_CTX_PICK_APERTURE
Defines the area (2D Context) or the volume (3D Context) specified in Device Coordinates, which is used to test primitives for picking. Primitives within the specified coordinates are picked when the mouse button is pushed. The default value for 2D is [-1,1] × [-1,1], and for 3D is [-1,1] × [-1,1] × [-1,1].

XGL_CTX_PICK_SURF_STYLE
Controls how polygon picking occurs. If the value of this attribute is XGL_PICK_SURF_AS_SOLID, the polygon is picked as a solid polygon, regardless of the current setting of XGL_CTX_SURF_FRONT_FILL_STYLE or XGL_3D_CTX_SURF_BACK_FILL_STYLE. If this attribute is set to

XGL_PICK_SURF_AS_FILL_STYLE, the polygon is picked as it is rendered, in accordance with the XGL picking semantics. The default value is XGL_PICK_SURF_AS_SOLID.

Context Attributes Associated with the View Model

XGL_CTX_VDC_ORIENTATION

Defines the orientation of the Virtual Device Coordinate (VDC) space relative to Device Coordinate (DC) space. The coordinate system is always right-handed. (See Chapter 11, "View Model," for more information.)

XGL_CTX_VDC_ORIENTATION enables the programmer to change the orientation of the VDC space to one of the following:

- XGL_Y_DOWN_Z_AWAY
- XGL_Y_UP_Z_TOWARD.

The default value is XGL_Y_DOWN_Z_AWAY. See Figure 11-3 on page 308 for more information.

XGL_CTX_MODEL_TRANS

Returns the Transform object that represents the Model Transform, which maps geometry from model coordinates to world coordinates.

XGL_CTX_MC_TO_DC_TRANS

Returns the Transform object that represents the Model-Coordinate-to-Device-Coordinate Transform, which maps geometry from model coordinates to device coordinates.

XGL_3D_CTX_NORMAL_TRANS

Returns the Transform object that represents the Normal Transform, which transforms normal vectors from model coordinates to world coordinates.

Context Attributes Associated with Lighting

XGL_3D_CTX_LIGHT_NUM

This attribute sets the number of Lights in a 3D Context. When an application sets this attribute, XGL changes the number of Lights and their corresponding on and off switches to the value of this attribute. The default value is 0. (See Chapter 16, "Lighting, Shading, and Depth Cueing" for more information.)

`XGL_3D_CTX_LIGHTS`
This attribute specifies an array of Lights used by a given Context. The default value is `NULL`.