Programming Example 5

The Sample Program

This chapter contains the program listings of a simple CGE PEX 5.1 program that displays a rotating torus, with or without texture mapping. This short program shows you how to determine if CGE PEX 5.1 is present and how you can avoid queries, once you know CGE PEX 5.1 is there. It also shows you how to apply texture mapping to an object with a minimum amount of work. All CGE vendors ship the following files with their CGE PEX 5.1 products which provides you with the opportunity to experiment with the files:

- `ptorus.h` - header file for entire program. It contains definitions of common data structures and function prototypes. (See “ptorus.h - Header File for Main Program” on page 30.)
- `ptorus.c` - main program. Be sure to look at this file. (See “ptorus.c - Main Program” on page 33.)
- `view.c` - code for handling viewing calculations and setting the view table entries. Not extremely important for illustrating CGE. (See “view.c - View Computations Example” on page 45.)
- `model.c` - code for creating the model. It generates a torus and stores it into a structure for later display. A compile-time definition controls whether or not you want the PEX server to apply a texture to the torus. The code in this file also sets up the pipeline context and performs other model-specific initialization. (See “model.c - Model and Scene Generation Example” on page 50.)

ptorus.h - *Header File for Main Program*

Code Example 5-1 Header File for Main Program Example

```
*/
        CGE PEX 5.1 Simple Programming Example
        Header file
        Define common function prototypes and data structures
*/

/*
    This structure is used to describe a view using a
    camera field-of-view model.
*/

typedef struct
{
    float refx,refy,refz;
    float camx,camy,camz;
    float upx,upy,upz;
    float field_of_view;
    float front,back;
    int perspective;
} camera_arg;

/*
    This procedure sets the view table entry to
    the view described by the camera argument.
*/

extern void set_view_camera(
#if NeedFunctionPrototypes
        Display                 *dpy,
        Window                  window,
        camera_arg              *ca,
        PEXRenderer             renderer,
        PEXRendererAttributes   *rend_attr,
        int                     index
#endif
);

/*
    This procedure generates the geometric model with appropriate
    surface attributes.
```

Code Example 5-1 Header File for Main Program Example *(Continued)*

```
*/

extern void Torus(
#if NeedFunctionPrototypes
        Display                 *dpy,
        XID                     resource_id,
        PEXOCRequestType        req_type
#endif
);

/*
    This procedure creates a structure containing the geometric
    model.
*/

extern PEXStructure create_model(
#if NeedFunctionPrototypes
        Display                 *dpy
#endif
);

/*
    This procedure sets up viewing and lighting.
*/

extern void create_scene(
#if NeedFunctionPrototypes
        Display                 *dpy,
        camera_arg              *camera,
        int                     *num_lights,
        PEXLightEntry           **lights,
        PEXTableIndex           **lss
#endif
);

/*
    This procedure sets up initial attributes in the Pipeline
    Context.
*/

extern void create_initial_state(
#if NeedFunctionPrototypes
        Display                 *dpy,
        PEXPipelineContext      plc,
```

Code Example 5-1 Header File for Main Program Example *(Continued)*

```
        int                         num_lights,
        PEXTableIndex               *lss
#endif
);

/*
    This procedure sets up the Renderer and creates the useful
    associated resources (Lookup Tables and Pipeline Context).
*/

extern PEXRenderer create_renderer(
#if NeedFunctionPrototypes
        Display                     *dpy,
        Window                      window,
        PEXRendererAttributes       *rend_attrs
#endif
);
```

ptorus.c - *Main Program*

Code Example 5-2 Main Program Example

```
/*
        CGE PEX 5.1 Simple Programming Example

        Main program file
*/

#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <malloc.h>

#include <X11/Xlib.h>
#include <X11/Xatom.h>
#include <X11/Xutil.h>
#include <X11/extensions/multibuf.h>

#include <X11/PEX5/PEXlib.h>
#include <X11/PEX5/PEXExtlib.h>

#include <PEXUtCmap.h>

#include "ptorus.h"

/*
    Forward declarations of utility procedures in this file.
*/

static void redraw(
#if NeedFunctionPrototypes
        Display *dpy,
        Multibuffer *drawable,
        PEXRenderer renderer,
        PEXStructure structure,
        Multibuffer *buffers
#endif
);

static void p_animate(
#if NeedFunctionPrototypes
```

Code Example 5-2 Main Program Example *(Continued)*

```
        Display *dpy,
        PEXRenderer rid,
        PEXStructure sid
#endif
);

/*
    This data structure describes the enumerated type values this
    program requires. They are checked in the main program below.
    Note that the only check we really need to make is to look for
    the CGE Escape Identifiers. If they are present, we can be
    sure that we have a CGE server and can assume that a large
    number of features are supported.
    This program requires no features beyond what CGE PEX 5.1
    requires, so we just need to check for CGE support.
*/

static struct enum_entry {
    unsigned int        enum_type;
    unsigned int        enum_value;
    char                *message;
} enums_to_check[]={

    { PEXETEscape,              PEXExtEscapeChangePipelineContext,
        "CGE PEX 5.1 Support" },

};
#define NUM_ENUMS_TO_CHECK (sizeof(enums_to_check)/sizeof(struct
                                        enum_entry))

/*
    The main program. Takes one possible command-line argument to
    name the X server, -display <display_name>.
*/

main (argc, argv)
        int argc;
        char **argv;
{
    char                            *prog_name;
    char                            *display_name;

    int                             screen;
```

Code Example 5-2 Main Program Example *(Continued)*

```
    Display                     *dpy;
    Window                      window;
    Multibuffer                 renderBuf;
    Multibuffer                 buffers[2];
    int                         mbx_buf_count;

    PEXExtensionInfo            *ext_info;
    char                       err_string[PEXErrorStringLength];

    XVisualInfo                 vis_info;
    XStandardColormap           cmap_info;
    PEXColorApproxEntry         capx_info;

    XSizeHints                  hints;
    PEXUtVisualCriteria         criteria;
    PEXUtWindowSpecification    window_info;
    int                         crit_index;
    unsigned int                unmet;
    Atom                        prop_atom;
    int                         result;
    XColor                      returned_background;
    XEvent                      event;
    Colormap                    cmap_id;

    Atom                        xa_WM_DELETE_WINDOW;
    Atom                        xa_MOTIF_WM_MESSAGES;
    Atom                        xa_WM_PROTOCOLS;
    Atom                        protocols[2];
    char                        *title;

    PEXRendererAttributes       rend_attrs;
    PEXRenderer                 renderer;
    PEXStructure                structure;

    int                         num_lights;
    PEXLightEntry               *lights;
    PEXTableIndex               *lss;

    int                         visible;
    camera_arg                  camera;

    /*
        Get the program name for error messages.
    */
```

Code Example 5-2 Main Program Example *(Continued)*

```
    if (prog_name = strrchr(argv[0], '/'))
        prog_name++;
    else
        prog_name = argv[0];

    /*
        Parse the command line arguments.
    */
    if ((argc > 2)
            && argv[1]
            && !strcmp( "-display", argv[1] ))
        display_name = XDisplayName( argv[2] );
    else
        display_name = XDisplayName( NULL );

    /*
        Open the X connection.
    */
    if ( !(dpy = XOpenDisplay( display_name )) )
    {
       fprintf(stderr, "%s: can't open display %s -- exiting\n",
                        prog_name, display_name);
        exit (1);
    }

    screen = DefaultScreen(dpy);

    /*
        Initialize PEXlib on the connection.
    */
    if (PEXInitialize (dpy, &ext_info, PEXErrorStringLength,
                         err_string))
    {
      fprintf (stderr, "%s: PEXInitialize() failed on %s: %s\n",
                 prog_name, display_name, err_string);
        exit (1);
    }

    /*
        Select a Visual for PEX rendering and create a Window
        on the Visual.
        We're going to hint that we want Double Buffering for
        smooth animation and to use a Standard Colormap to
        promote the sharing of Colormap resources on the server.
```

Code Example 5-2 Main Program Example *(Continued)*

```
        If we don’t get either or both, we’ll stumble on.
    */

    hints.x = 128;
    hints.y = 128;
    hints.width = 256;
    hints.height = 256;
    hints.flags = (USSize|USPosition);

    criteria.hard_criteria_mask = 0;
    criteria.soft_criteria_mask = PEXUtDBCapability |
                                  PEXUtStandardColormapProperty;

    criteria.double_buffering_capability = PEXUtDBPEX;
    criteria.standard_colormap_property = True;

    window_info.attr_mask = CWEventMask;
    window_info.attrs.event_mask = ( ExposureMask        |
                                    VisibilityChangeMask |
                                    StructureNotifyMask);

    window_info.title = "PEX Torus";
    window_info.size_hints = hints;
    window_info.parent = RootWindow (dpy, screen);
    window_info.border_width = 0;
    window_info.background_color_name = "black";
    window_info.border_color_name = "white";

    result = PEXUtMakeWindowAndColormap (dpy, screen,
                                         1, &criteria,
                                        &window_info, &window,
                                        &vis_info, &cmap_info,
                                         &capx_info,
                                         &crit_index, &unmet,
                                         &prop_atom,
                                         &returned_background,
                                         &cmap_id);

    if (result != PEXUtSuccess)
    {
        static char *PEXUtFailStrings[] =
        {
            "PEXUtBadAlloc",              /* -4 maps to 0 */
            "PEXUtPEXFailure",            /* -3 maps to 1 */
```

Code Example 5-2 Main Program Example *(Continued)*

```
            "PEXUtXFailure",                /* -2 maps to 2 */
            "PEXUtCriteriaFailure",         /* -1 maps to 3 */
            "PEXUtSuccess",                 /*  0 maps to 4 */
            "PEXUtQualifiedSuccess",        /* +1 maps to 5 */
            "PEXUtAlternativeSuccess",      /* +2 maps to 6 */
        };

        fprintf(stderr,"%s: visual selection or window creation
                 failed: %s\n", prog_name, PEXUtFailStrings
                 [ result + 4 ]);
        exit(1);
    }

    /*
       Report soft criteria failures, but don't quit.
    */
    if (unmet & PEXUtDBCapability)
        fprintf(stderr,
              "%s: could not find Visual for double buffering\n",
                 prog_name);
    if (unmet & PEXUtStandardColormapProperty)
        fprintf(stderr,
              "%s: could not find Standard Colormap Property\n",
                prog_name);

    /*
        Set up some window manager properties (see the ICCCM).
    */

    xa_WM_DELETE_WINDOW   = XInternAtom(dpy, "WM_DELETE_WINDOW",
                                    False);
   xa_MOTIF_WM_MESSAGES = XInternAtom(dpy, "_MOTIF_WM_MESSAGES",
                                    False);
    xa_WM_PROTOCOLS       = XInternAtom(dpy, "WM_PROTOCOLS",
False);
    protocols[0] = xa_WM_DELETE_WINDOW ;
    protocols[1] = xa_MOTIF_WM_MESSAGES;
    XSetWMProtocols(dpy, window, protocols, 2);

    title = "PEX Torus";
    XChangeProperty(dpy, window, XA_WM_ICON_NAME, XA_STRING,
                 8, PropModeReplace, (unsigned char *) title, 8);

    /*
```

Code Example 5-2 Main Program Example *(Continued)*

```
        Configure the window for double-buffering via MBX.
    */

    mbx_buf_count = XmbufCreateBuffers( dpy, window, 2,
                                 MultibufferUpdateActionBackground,
                                 MultibufferUpdateHintFrequent,
                                 buffers );
    if ( mbx_buf_count == 0 ) {
        fprintf(stderr,
            "%s: can't create MBX buffers on display %s, visual
             %x\n",
                prog_name, display_name, vis_info.visualid);
        exit(1);
    }

    /*
        Handle case where only one MBX buffer was created.
        This will make the swap work, even if one buffer.
    */
    if ( mbx_buf_count == 1 )
      buffers[1] = buffers[0];

      renderBuf = buffers[1];


    /*
       Check the Enum Types to be sure that CGE PEX is supported
    */
    {
        int status, i, j, found;
        int enum_types[NUM_ENUMS_TO_CHECK];
        unsigned long *enum_counts;
        PEXEnumTypeDesc *enum_values, *p_enum;

        for (i=0; i<NUM_ENUMS_TO_CHECK; i++)
            enum_types[i] = enums_to_check[i].enum_type;

        status = PEXGetEnumTypeInfo (dpy, renderBuf,
                                    NUM_ENUMS_TO_CHECK, enum_types,
                                    PEXETCounts|PEXETIndex,
                                    &enum_counts, &enum_values);
        if (!status) {
            fprintf(stderr,
             "%s: inquiry of enumerated types failed - exiting\n",
```

Code Example 5-2 Main Program Example *(Continued)*

```
                    prog_name);
                exit(1);
            }

            p_enum = enum_values;
            for (i=0; i<NUM_ENUMS_TO_CHECK; i++) {

                found = False;

                for (j=0; j<enum_counts[i]; j++) {
                    if (((unsigned short) (p_enum++)->index) ==
                    ((unsigned short) enums_to_check[i].enum_value))
 {
                        found = True;
                    }
                }

                if (!found) {
                    fprintf(stderr,
                       "%s: this program requires support for %s\n",
                        prog_name, enums_to_check[i].message);
                    exit(1);
                }
            }

            PEXFreeEnumInfo (NUM_ENUMS_TO_CHECK, enum_counts,
                                 enum_values);
        }

        /*
            Create the model and scene to be animated.
        */

        structure = create_model(dpy);
        create_scene(dpy, &camera, &num_lights, &lights, &lss);

        /*
            Create a Renderer and various lookup tables.
            Set up the view, lights, and Pipeline Context.
        */

        renderer = create_renderer(dpy, window, &rend_attrs);
```

Code Example 5-2 Main Program Example *(Continued)*

```
    set_view_camera(dpy, window, &camera, renderer, &rend_attrs,
                      1);
    PEXSetTableEntries(dpy, rend_attrs.light_table, lss[0],
                        num_lights, PEXLUTLight, lights);
    PEXSetTableEntries(dpy, rend_attrs.color_approx_table, 0, 1,
                        PEXLUTColorApprox, &capx_info);
    create_initial_state (dpy, rend_attrs.pipeline_context,
                           num_lights, lss);

    /*
        Now enter the main event-processing and animation loop.
    */

    visible = False;
    while (True)
    {

        /*
            First handle any pending events.
        */

        while (XPending(dpy))
        {
            XNextEvent( dpy, &event );

            switch ( event.type ) {
            case ConfigureNotify:
                /*
                    Adapt the view to the window configuration
                    and render the image.
                */
                set_view_camera(dpy, window, &camera, renderer,
                                  &rend_attrs, 1);
                redraw(dpy, &renderBuf, renderer, structure,
                         buffers);
                break;

            case Expose:
                /*
                    Flush remaining Expose events.
                */
                while ( XCheckTypedWindowEvent( dpy, window,
                        Expose, &event ));
                /* empty statement */
```

Code Example 5-2 Main Program Example *(Continued)*

```
                    visible = True;

                    /*
                        Adapt the view to the window configuration
                        and render the image.
                    */
                    set_view_camera(dpy, window, &camera, renderer,
                                      &rend_attrs, 1);
                    redraw(dpy, &renderBuf, renderer, structure,
                             buffers);
                    break;

                case ClientMessage:
                  if (event.xclient.message_type == xa_WM_PROTOCOLS)
                    {
                        if (event.xclient.data.l[0] ==
                             xa_WM_DELETE_WINDOW)
                        {
                            /*
                               Terminate the double-buffering on the
                                window.
                                Close the Window and the connection.
                            */

                            XmbufDestroyBuffers( dpy, window );
                            XDestroyWindow (dpy, window);
                            XCloseDisplay( dpy );
                            exit(0);
                        }
                    }
                    break;
                }
            } /* event handling */

            /*
                After handling events, render the next frame in the
                animation.
            */

            if (visible)
              redraw(dpy, &renderBuf, renderer, structure, buffers);

        } /* main loop */
```

Code Example 5-2 Main Program Example *(Continued)*

```
} /* main */

/*
    This procedure draws one frame and swaps the buffers.
*/
static void redraw(dpy, drawable, renderer, structure, buffers)
        Display *dpy;
        Multibuffer *drawable;
        PEXRenderer renderer;
        PEXStructure structure;
        Multibuffer *buffers;
{
    PEXBeginRendering(dpy, (Drawable) *drawable, renderer);
    p_animate(dpy, renderer, structure);
    PEXEndRendering(dpy, renderer, True);

    /* Swap the buffers. */
    XmbufDisplayBuffers( dpy, 1, drawable, 0, 0 );
    *drawable = ((*drawable == buffers[0]) ? buffers[1] :
buffers[0]);

    XSync(dpy, False);

} /* redraw */


/*
    This procedure generates the animation and does the rendering
    of a single frame.
*/

static float xangle = 0.01;
static float yangle = 0.1;

static void p_animate(dpy, rid, sid)
        Display *dpy;
        PEXRenderer rid;
        PEXStructure sid;
{
    PEXMatrix model_mtx, xrot_mtx, yrot_mtx;

    xangle += 0.01;
    if (xangle > 2*M_PI) xangle = 0.01;
    PEXRotate(PEXXAxis, xangle, xrot_mtx);
```

Code Example 5-2 Main Program Example *(Continued)*

```
    yangle += 0.1;
    if (yangle > 2*M_PI) yangle = 0.1;
    PEXRotate(PEXYAxis, yangle, yrot_mtx);

    PEXMatrixMult (xrot_mtx, yrot_mtx, model_mtx);

    PEXSetLocalTransform(dpy, rid, PEXOCRender, PEXReplace,
                                        model_mtx);
    PEXExecuteStructure(dpy, rid, PEXOCRender, sid);

} /* p_animate */
```

view.c - *View Computations Example*

Code Example 5-3 View Computations Example

```
/*
        CGE PEX 5.1 Simple Programming Example

        View Computations

*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#include <X11/PEX5/PEXlib.h>

#include "ptorus.h"

/*
Linear algebra utilities.
*/

void vector_subtract(c, a, b)
float *c;
float *a;
float *b;
{
    *c++ = *a++ - *b++;
    *c++ = *a++ - *b++;
    *c++ = *a++ - *b++;
}

double vector_length(v)
    register float *v;
{
    register int i;
    register double s;

    for (i = 0, s = 0.0; i < 3; i++, v++)
        s += *v * *v;
    s = sqrt(s);
    return(s);
}
```

Code Example 5-3 View Computations Example *(Continued)*

```
void vector_scale(v, s)
    register float *v, s;
{
    *v++ *= s;
    *v++ *= s;
    *v++ *= s;
}


/*
    This procedure creates a view table entry and installs it.
*/

void set_view_camera(dpy, window, ca, renderer, rend_attr, index)
    Display                     *dpy;
    Window                      window;
    camera_arg                  *ca;
    PEXRenderer                 renderer;
    PEXRendererAttributes       *rend_attr;
    int                         index;
{
    double viewDistance;

    PEXCoord            viewReferencePoint;
    PEXCoord            projectionReferencePoint;
    PEXVector           viewUpVector, viewPlaneNormal;
    PEXCoord2D          viewWindow[2];
    double              viewPlane, frontPlane, backPlane;
    PEXNPCSubVolume     viewPort;
    PEXViewEntry        view;
    int                 err, perspective;
    double              windowScale;
    double              winAspect;
    int                 winWidth, winHeight;

    XWindowAttributes   win_attrs;

    /*
     * Compute the ViewOrientationMatrix from viewReferencePoint,
      * viewPlaneNormal and viewUpVector.  This stuff is
independent
      * of the resize method.
      */
```

Code Example 5-3 View Computations Example *(Continued)*

```
    viewReferencePoint.x = ca->refx;
    viewReferencePoint.y = ca->refy;
    viewReferencePoint.z = ca->refz;

    vector_subtract((float *) &viewPlaneNormal, &(ca->camx),
                     &(ca->refx));
    viewDistance = vector_length(&viewPlaneNormal);
    vector_scale(&viewPlaneNormal, 1.0 / viewDistance);

    viewPlane  = 0;
    backPlane  = -10000;
    frontPlane = viewDistance - 1;

    viewUpVector.x = ca->upx;
    viewUpVector.y = ca->upy;
    viewUpVector.z = ca->upz;

    err = PEXViewOrientationMatrix(&viewReferencePoint,
                                   &viewPlaneNormal,
                                   &viewUpVector,
                                   view.orientation);

    /*
       Compute the size (and shape) of the viewWindow based upon
       the camera's field of view and the aspect ratio of the
       window to which we're about to render.
    */

    XGetWindowAttributes (dpy, window, &win_attrs);

    windowScale = atan(ca->field_of_view * M_PI / 360.0)
                    * (viewDistance - viewPlane);
    winWidth    = (win_attrs.width  > 1) ? (win_attrs.width  - 1)
                    : 1;
    winHeight   = (win_attrs.height > 1) ? (win_attrs.height - 1)
                    : 1;
    winAspect   = (double) winWidth / (double) winHeight;

    viewPort.min.x = 0;    viewPort.max.x = 1;
    viewPort.min.y = 0;    viewPort.max.y = 1;
    viewPort.min.z = 0;    viewPort.max.z = 1;
    rend_attr->npc_subvolume = viewPort;
```

Code Example 5-3 View Computations Example *(Continued)*

```
    viewWindow[0].y = -windowScale;
    viewWindow[1].y =  windowScale;
    viewWindow[0].x = -windowScale;
    viewWindow[1].x =  windowScale;

    view.clip_flags = PEXClippingAll;
    perspective = True;
    projectionReferencePoint.x = 0;
    projectionReferencePoint.y = 0;
    projectionReferencePoint.z = viewDistance;

    /*
     * Compute the aspect ratio of the viewPort so we fill the
     * window. From that the ViewMappingMatrix is computed.
     */

    /*
     * This method varies only the renderer's npc_subvolume to
    * map the 0-1 range to the major dimension of the destination
     * drawable. The minor dimension is reduced by the aspect
     * ratio of the drawable and then centered.
     */

    if (winWidth > winHeight)
    {
        double aspect = (double) winHeight / (double) winWidth;

        rend_attr->npc_subvolume.min.y = (1 - aspect) / 2;
        rend_attr->npc_subvolume.max.y =
                        rend_attr->npc_subvolume.min.y + aspect;
    }
    else if (winWidth < winHeight)
    {
        rend_attr->npc_subvolume.min.x = (1 - winAspect) / 2;
        rend_attr->npc_subvolume.max.x =
                                rend_attr->npc_subvolume.min.x +
                                winAspect;
    }

    view.clip_limits = viewPort;
    PEXChangeRenderer(dpy, renderer, PEXRANPCSubVolume,
                        rend_attr);
```

Code Example 5-3 View Computations Example *(Continued)*

```
    err = PEXViewMappingMatrix(viewWindow, &viewPort,
                               perspective,
                             &projectionReferencePoint,
                             viewPlane, backPlane, frontPlane,
                             view.mapping);

    PEXSetTableEntries(dpy, rend_attr->view_table, index, 1,
                      PEXLUTView,
                     (PEXPointer) &view);

} /* set_view_camera */
```

model.c - *Model and Scene Generation Example*

Code Example 5-4 Model and Scene Generation Example

```
/*
        CGE PEX 5.1 Simple Programming Example

        Model and scene generation

*/
#define TEXTURE                                 /* Remove this line to
                                                   disable texture */
#include <stdio.h>
#include <math.h>

#include <X11/PEX5/PEXlib.h>
#include <X11/PEX5/PEXExtlib.h>

#include "ptorus.h"

/*
    This procedure normalizes a vector to a magnitude of 1.0.
*/

NormalizeVector( vector )
    PEXVector *vector;
{
    float r;

    /* Transform a vector to a unit vector. */
    r = vector->x * vector->x
        + vector->y * vector->y
        + vector->z * vector->z;

    if ( r > 0 ) {
      r = 1.0 / sqrt( r );
      vector->x *= r;
      vector->y *= r;
      vector->z *= r;
    }
} /* NormalizeVector */

/*
    This procedure generates the output commands that constitute
```

Code Example 5-4 Model and Scene Generation Example *(Continued)*

```
    the model and the closely-associated surface attributes.
*/

void Torus(dpy, resource_id, req_type)
Display *dpy;
XID resource_id;
PEXOCRequestType req_type;
{
  /*
   * Torus parameters
   */
    float xaxis                        = 10.0,
    yaxis                              = 10.0,
    zaxis                              = 10.0,
    radius                             = 4.0;
    PEXCoord center                    = {0,0,0};
    int depth                        = 4; /* quality control */

    /*
     * Misc variables
     */
    int i, j;                          /* i/phi    j/theta */
    float u, v;
    float phi, theta, step;
    float fphiSize, fthetaSize;
    int phiSize, thetaSize;
    int count;
#ifdef TEXTURE
    /* "Custom" vertex mapping with texture data */
    typedef struct _PEXExtVertexNormal {
      PEXCoord point;
      PEXVector normal;
      PEXCoord data;
    } PEXExtVertexNormal;
    PEXExtVertexNormal *torus;
    PEXExtArrayOfVertex quadmesh;
#else
    PEXVertexNormal *torus;
    PEXArrayOfVertex quadmesh;
#endif
    PEXReflectionAttributes ra;
    PEXColor color;
    PEXArrayOfFacetData facetdata;
```

Code Example 5-4 Model and Scene Generation Example *(Continued)*

```
    /*
     * Set up a green object.
     */
    color.rgb.red                        = 0.1;
    color.rgb.green                      = 0.9;
    color.rgb.blue                       = 0.1;
    ra.ambient                           = 0.2;
    ra.diffuse                           = 1;
    ra.specular                          = 1;
    ra.specular_conc                     = 15;
    ra.transmission                      = 0;
    ra.specular_color.type               = PEXColorTypeRGB;
    ra.specular_color.value.rgb.red      = 0.0402;
    ra.specular_color.value.rgb.green    = 0.4012;
    ra.specular_color.value.rgb.blue     = 0.0402;
    PEXSetSurfaceColor(dpy, resource_id, req_type,
                       PEXColorTypeRGB, &color);
    PEXSetReflectionAttributes(dpy, resource_id, req_type, &ra);
#ifdef TEXTURE
    PEXSetInteriorStyle(dpy, resource_id, req_type,
PEXExtInteriorStyleTexture);
#endif

    /*
     * Compute step size and allocate space for verticies.
     */
    step = (float)(M_PI / pow( (double)2, (double)depth ));
    fphiSize = (2 * M_PI / step) + 1.;
    fthetaSize = (2 * M_PI / step) + 1.;
    phiSize = (int)fphiSize;
    thetaSize = (int)fthetaSize;
#ifdef TEXTURE
    torus = (PEXExtVertexNormal *) malloc (phiSize * thetaSize *
                                    sizeof(PEXExtVertexNormal));
#else
    torus = (PEXVertexNormal *) malloc (phiSize * thetaSize *
                                    sizeof(PEXVertexNormal));
#endif

    /*
     * Sweep from -pi to pi along both u and v to generate
       verticies.
     */
    count = 0;
```

Code Example 5-4 Model and Scene Generation Example *(Continued)*

```
    u = -M_PI - step;
    for( i = 0; i < phiSize; i++ ) {
      u += step;
      v = -M_PI - step;
      for( j = 0; j < thetaSize; j++, count++ ) {
        v +=  step;
        /*
         * Compute vertex coordinate and normal
         */
        torus[count].point.x = xaxis *
           (radius + cos(v)) * cos(u) + center.x;
        torus[count].point.y = yaxis *
           (radius + cos(v)) * sin(u) + center.y;
        torus[count].point.z = zaxis * sin(v) + center.z;
        torus[count].normal.x = (1 / xaxis) * cos(v) * cos(u);
        torus[count].normal.y = (1 / yaxis) * cos(v) * sin(u);
        torus[count].normal.z = (1 / zaxis) * sin(v);
        NormalizeVector (&(torus[count].normal));
#ifdef TEXTURE
        torus[count].data.x = (u + M_PI) / (2 * M_PI);
        torus[count].data.y = (v + M_PI) / (2 * M_PI);
        torus[count].data.z = 0.0;
#endif
        }
    }

#ifdef TEXTURE
    quadmesh.with_fp_data = (PEXPointer)torus;
    PEXExtQuadrilateralMesh (dpy,
                             resource_id,
                             req_type,
                             PEXShapeUnknown,
                             3,
                             PEXGANone,
                             PEXGANormal | PEXExtGAData,
                             PEXColorTypeIndexed,
                             facetdata,
                             thetaSize,
                             phiSize,
                             quadmesh);
#else
    quadmesh.normal = (PEXVertexNormal *)torus;
    PEXQuadrilateralMesh (dpy,
                          resource_id,
```

Code Example 5-4 Model and Scene Generation Example *(Continued)*

```
                                req_type,
                                PEXShapeUnknown,
                                PEXGANone,
                                PEXGANormal,
                                PEXColorTypeIndexed,
                                facetdata,
                                thetaSize,
                                phiSize,
                                quadmesh);
#endif
    free(torus);
} /* Torus */

/*
     This procedure creates a structure containing the geometric
     model.
*/

PEXStructure create_model(dpy)
        Display         *dpy;
{
        PEXStructure str_return;

        /*
            Create and load the structure containing the model.
        */
        str_return = PEXCreateStructure(dpy);
        Torus (dpy, str_return, PEXOCStore);

        return str_return;

} /* create_model */

/*
    This procedure creates the light table entries and activation
    list for the scene.
*/

#define NUM_LIGHTS 5
PEXLightEntry ltbl[NUM_LIGHTS];
PEXTableIndex lndx[NUM_LIGHTS];

pex_cube_lights(dpy, num_lights, lights, lss)
        Display *dpy;
```

Code Example 5-4 Model and Scene Generation Example *(Continued)*

```
        int *num_lights;
        PEXLightEntry **lights;
        PEXTableIndex **lss;
{
        ltbl[0].type = PEXLightAmbient;
        ltbl[0].color.type = PEXColorTypeRGB;
        ltbl[0].color.value.rgb.red   = 0.1;
        ltbl[0].color.value.rgb.green = 0.1;
        ltbl[0].color.value.rgb.blue  = 0.1;
        lndx[0] = 1;

        ltbl[1].type = PEXLightWCPoint;
        ltbl[1].color.type = PEXColorTypeRGB;
        ltbl[1].color.value.rgb.red   = 0.7653;
        ltbl[1].color.value.rgb.green = 0.763499;
        ltbl[1].color.value.rgb.blue  = 0.7651;
        ltbl[1].point.x = -19.0;
        ltbl[1].point.y = 0.0;
        ltbl[1].point.z = 200.0;
        ltbl[1].attenuation1 = 1;
        ltbl[1].attenuation2 = 0;
        lndx[1] = 2;

        ltbl[2] = ltbl[1];
        ltbl[2].point.x = 17.0;
        ltbl[2].point.y = 200.0;
        ltbl[2].point.z = -41.0;
        lndx[2] = 3;

        ltbl[3] = ltbl[1];
        ltbl[3].point.x = 175.0;
        ltbl[3].point.y = -83.0;
        ltbl[3].point.z = -51.0;
        lndx[3] = 4;

        ltbl[4] = ltbl[1];
        ltbl[4].point.x = -181.0;
        ltbl[4].point.y = -83.0;
        ltbl[4].point.z = 0.0;
        lndx[4] = 5;

        *num_lights = NUM_LIGHTS;
        *lights = ltbl;
        *lss = lndx;
```

Code Example 5-4 Model and Scene Generation Example *(Continued)*

```
} /* pex_cube_lights */

/*
   This procedure creates the viewing and lighting for the scene.
*/

void create_scene(dpy, camera, num_lights, lights, lss)
        Display *dpy;
        camera_arg *camera;
        int *num_lights;
        PEXLightEntry **lights;
        PEXTableIndex **lss;
{
        /*
            Load the camera describing the view.
        */
        camera->refx = 0.0;
        camera->refy = 0.0;
        camera->refz = 496.0;
        camera->camx = 0.0;
        camera->camy = 0.0;
        camera->camz = 497.0;
        camera->upx = 0.0;
        camera->upy = 1.0;
        camera->upz = 0.0;
        camera->field_of_view = 18.4;
        camera->front = 0.0;
        camera->back = 9999.0;
        camera->perspective = True;

        /*
            Load the light table entries.
        */
        pex_cube_lights(dpy, num_lights, lights, lss);

} /* create_scene */

/*
    This procedure sets up a Pipeline Context with the initial
    attributes required for the scene and model rendering.
*/

void create_initial_state (dpy, plc, num_lights, lss)
```

Code Example 5-4 Model and Scene Generation Example *(Continued)*

```
        Display *dpy;
        PEXPipelineContext plc;
        int num_lights;
        PEXTableIndex *lss;
{
    PEXPCAttributes pc_attrs;
    unsigned long attr_mask[3];

    attr_mask[0] = 0;
    attr_mask[1] = 0;
    attr_mask[2] = 0;

    PEXSetPCAttributeMask(attr_mask, PEXPCCullingMode);
    pc_attrs.culling_mode = PEXBackFaces;

    PEXSetPCAttributeMask(attr_mask, PEXPCDistinguishFlag);
    pc_attrs.distinguish = False;

    PEXSetPCAttributeMask(attr_mask, PEXPCSurfaceEdgeFlag);
    pc_attrs.surface_edges = PEXOff;

    PEXSetPCAttributeMask(attr_mask, PEXPCInteriorStyle);
    pc_attrs.interior_style = PEXInteriorStyleSolid;

    PEXSetPCAttributeMask(attr_mask, PEXPCReflectionModel);
    pc_attrs.reflection_model = PEXReflectionSpecular;

    PEXSetPCAttributeMask(attr_mask, PEXPCSurfaceInterp);
    pc_attrs.surface_interp = PEXSurfaceInterpColor;

    PEXSetPCAttributeMask(attr_mask, PEXPCViewIndex);
    pc_attrs.view_index = 1;

    PEXSetPCAttributeMask(attr_mask, PEXPCLightState);
    pc_attrs.light_state.count = num_lights;
    pc_attrs.light_state.indices = lss;

    PEXChangePipelineContext (dpy, plc, attr_mask, &pc_attrs);

} /* create_initial_state */

/*
    This procedure sets up the Renderer and creates the useful
    associated resources (Lookup Tables and Pipeline Context).
```

Code Example 5-4 Model and Scene Generation Example *(Continued)*

```
*/

PEXRenderer create_renderer(dpy, window, rend_attrs)
        Display                 *dpy;
        Window                  window;
        PEXRendererAttributes   *rend_attrs;
{
    unsigned long itemMask = 0;
    unsigned long attr_mask[3];
    PEXRenderer renderer;

    itemMask |= PEXRAPipelineContext;
    attr_mask[0] = 0;
    attr_mask[1] = 0;
    attr_mask[2] = 0;
    rend_attrs->pipeline_context = PEXCreatePipelineContext(dpy,
                attr_mask, NULL);

    itemMask |= PEXRAViewTable;
    rend_attrs->view_table = PEXCreateLookupTable(dpy, window,
                PEXLUTView);

    itemMask |= PEXRALightTable;
    rend_attrs->light_table = PEXCreateLookupTable(dpy, window,
                PEXLUTLight);

    itemMask |= PEXRAColorApproxTable;
    rend_attrs->color_approx_table = PEXCreateLookupTable(dpy,
                window, PEXLUTColorApprox);

    itemMask |= PEXRABackgroundColor;
    rend_attrs->background_color.type          = PEXColorTypeRGB;
    rend_attrs->background_color.value.rgb.red   = 0.5;
    rend_attrs->background_color.value.rgb.green = 0.5;
    rend_attrs->background_color.value.rgb.blue  = 0.5;

    itemMask |= PEXRAHLHSRMode;
    rend_attrs->hlhsr_mode = PEXHLHSRZBuffer;

    itemMask |= PEXRAClearImage;
    rend_attrs->clear_image = True;

    renderer = PEXCreateRenderer(dpy, window, itemMask,
                             rend_attrs);
```

Code Example 5-4 Model and Scene Generation Example *(Continued)*

```
    return renderer;

} /* create_renderer */
```

Texture Mapping in this Example

The preceding code example illustrates the technique and code for activating texture mapping with a minimum amount of effort and code. Here is an explanation of the changes made to `model.c` to enable texturing with the default texture:

```
#ifdef TEXTURE
    /* "Custom" vertex mapping with texture data */
    typedef struct _PEXExtVertexNormal {
      PEXCoord point;
      PEXVector normal;
      PEXCoord data;
    } PEXExtVertexNormal;
    PEXExtVertexNormal *torus;
    PEXExtArrayOfVertex quadmesh;
#else
    PEXVertexNormal *torus;
    PEXArrayOfVertex quadmesh;
#endif
```

This program creates a torus with a QuadMesh primitive. When not texturing, the only data needed for each vertex is the vertex normal to create a smooth shading effect and so the standard PEXVertexNormal data structure is sufficient. However, when texturing, the program also provides texture parameterization data. This data describes to the PEX server what part of the texture map is applied to the object at each vertex. The extra data in this case is a PEXCoord supplied with each vertex to specify the texture coordinate to be mapped to the object at each vertex.

Since there are a wide variety of texturing methods, there are also a variety of parameterization methods that require different amounts and types of extra data per vertex. Because of the large number of possibilities, PEXlib cannot supply Vertex data structures for all possibilities. Therefore, you must create your own to match the type of texturing you intend to use. Note in the code above the new PEXExtVertexNormal data structure that allows for one texture coordinate per vertex.

```
#ifdef TEXTURE
    PEXSetInteriorStyle(dpy, resource_id, req_type,
          PEXExtInteriorStyleTexture);
#endif
```

This function call is the key to turning on texturing. It tells the PEX server to apply a texture to the object instead of using a solid color (The interior style is set to PEXInteriorStyleSolid in the Pipeline Context). Note that instead of changing the interior style attribute with an OC, you could change it instead by initializing the Pipeline Context value to PEXExtInteriorStyleTexture.

```
#ifdef TEXTURE
    torus = (PEXExtVertexNormal *) malloc (phiSize * thetaSize *
          sizeof(PEXExtVertexNormal));
#else
    torus = (PEXVertexNormal *) malloc (phiSize * thetaSize *
          sizeof(PEXVertexNormal));
#endif
```

The code above simply allocates space for the torus data. Different code is needed to handle the differing data types since the data for each vertex is of different size. A PEXExtVertexNormal structure is bigger than a PEXVertexNormal by the size of the extra texture data.

```
#ifdef TEXTURE
        torus[count].data.x = (u + M_PI) / (2 * M_PI);
        torus[count].data.y = (v + M_PI) / (2 * M_PI);
        torus[count].data.z = 0.0;
#endif
```

The code above computes the texture coordinates and stores them in the extra data for each vertex. The texture coordinates are computed so that the entire default texture wraps around the torus exactly once. The texture coordinates range from 0.0 to 1.0.

Imagine cutting the torus once through one of its cylindrical cross sections, straightening the cut torus out into a cylinder, slicing the cylinder along one of its long sides, and then unrolling the result onto a flat surface. The parameterizations the program supplies above specify that the PEX server should apply the texture to this flat surface in one complete "patch", since the texture coordinates at the edges of this flattened surface will be 0.0 along two adjacent edges and 1.0 along the opposite edges. This way, you will see the entire texture mapped exactly once around the torus, after it is conceptually rolled back up into a torus.

The code above will generate the required texture coordinate data, since u and v range from -PI to PI, resulting in texture values for x and y between 0.0 and 1.0.

```
#ifdef TEXTURE
    quadmesh.with_fp_data = (PEXPointer)torus;
    PEXExtQuadrilateralMesh (dpy,
                             resource_id,
                             req_type,
                             PEXShapeUnknown,
                             3,
                             PEXGANone,
                             PEXGANormal | PEXExtGAData,
                             PEXColorTypeIndexed,
                             facetdata,
                             thetaSize,
                             phiSize,
                             quadmesh);
#else
    quadmesh.normal = (PEXVertexNormal *)torus;
    PEXQuadrilateralMesh (dpy,
                          resource_id,
                          req_type,
                          PEXShapeUnknown,
                          PEXGANone,
                          PEXGANormal,
                          PEXColorTypeIndexed,
                          facetdata,
                          thetaSize,
```

```
                    phiSize,
                    quadmesh);
#endif
```

Finally, the program sends the QuadMesh primitive to the PEX server. The CGE version of the function call uses an extra parameter to specify how much texture data there is per vertex (the fifth parameter in the first function call above) and also turns on the flag that indicates the presence of texture data.