GEL:

Graphics Expression Language

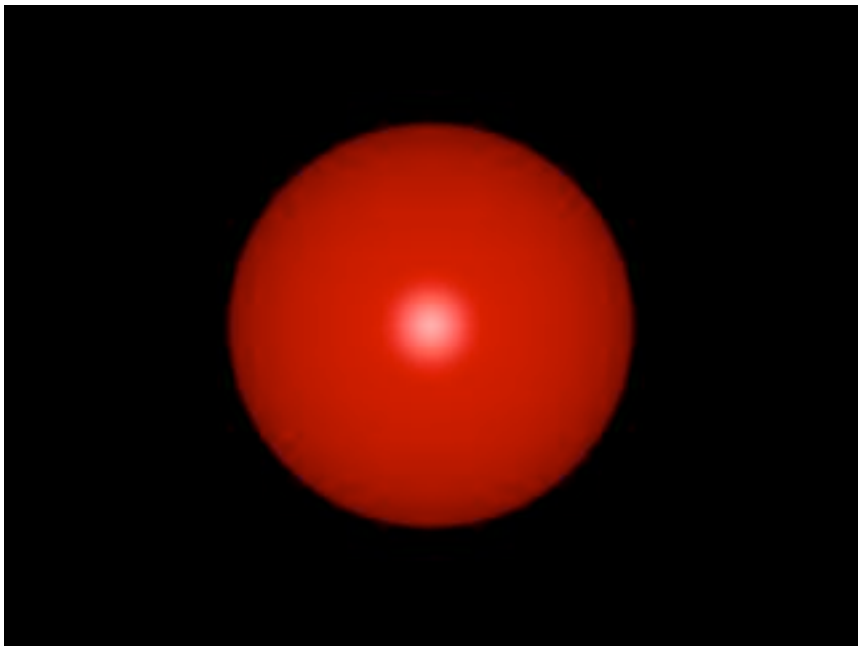
Amro Nasr

1. Introduction:

1.1 Background: Current technologies in the field of computer graphics and image synthesis provide a multitude of API's and programming languages that allow the programmer to describe rich virtual environments. Coupled with this is the divide that exists between realtime techniques used in games and other interactive environments and non-realtime methods employed in the creation of still images for such application as feature films.

1.2 Goal: The goal of GEL is to provide a level of abstraction that attempts to bind the different image synthesis methods for the realtime and non-realtime domains in a portable framework. This is accomplished by providing a unified interface that allows the programmer to write a single generic scene description that can be processed in both realtime and non-realtime based on GEL's state.

For example to describe an interactive scene that consists of a simple sphere with a camera and display window:



In OpenGL the program might look like the following:

```
#include <stdlib.h>
#include <GL/glut.h>

void init(void)
{
GLfloat ambient[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat position[] = { 0.0, 3.0, 2.0, 0.0 };
```

```
  GLfloat lmodel_ambient[] = { 0.4, 0.4, 0.4, 1.0 };
  GLfloat local_view[] = { 0.0 };

      glClearColor(0.0, 0.1, 0.1, 0.0);
      glEnable(GL_DEPTH_TEST);
      glShadeModel(GL_SMOOTH);

  glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
  glLightfv(GL_LIGHT0, GL_POSITION, position);
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
  glLightModelfv(GL_LIGHT_MODEL_LOCAL_VIEWER, local_view);

  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
}

void display(void)
{
  GLfloat no_mat[] = { 0.0, 0.0, 0.0, 1.0 };
  GLfloat mat_ambient[] = { 0.7, 0.7, 0.7, 1.0 };
  GLfloat mat_ambient_color[] = { 0.8, 0.8, 0.2, 1.0 };
  GLfloat mat_diffuse[] = { 0.1, 0.5, 0.8, 1.0 };
  GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
  GLfloat no_shininess[] = { 0.0 };
  GLfloat low_shininess[] = { 5.0 };
  GLfloat high_shininess[] = { 100.0 };
  GLfloat mat_emission[] = {0.3, 0.2, 0.2, 0.0};

  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

      glPushMatrix();
      glTranslatef (-3.75, 3.0, 0.0);
      glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);
      glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
      glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
      glMaterialfv(GL_FRONT, GL_SHININESS, no_shininess);
      glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
      glutSolidSphere(1.0, 16, 16);
      glPopMatrix();


  glFlush();
}

void reshape(int w, int h)
{
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if (w <= (h * 2))
    glOrtho (-6.0, 6.0, -3.0*((GLfloat)h*2)/(GLfloat)w,
      3.0*((GLfloat)h*2)/(GLfloat)w, -10.0, 10.0);
  else
    glOrtho (-6.0*(GLfloat)w/((GLfloat)h*2),
```

```
        6.0*(GLfloat)w/((GLfloat)h*2), -3.0, 3.0, -10.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
  }

  void keyboard(unsigned char key, int x, int y)
  {
    switch (key) {
      case 27:
        exit(0);
        break;
    }
  }

  int main(int argc, char** argv)
  {
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize (600, 450);
    glutCreateWindow(argv[0]);
    init();
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutKeyboardFunc (keyboard);
  glutMainLoop();
  return 0;
}
```

And a non-interactive version in Renderman would look like the following:

```
#Sphere1.rib
#GSO 6-22-98
Display "sphere1.tif" "file" "rgb"
Format 320 240 1
Projection "perspective" "fov" [30]
# Translate Camera (moves world origin 6 units towards +Z)
Translate 0 0 6
WorldBegin
 LightSource "ambientlight" 1 "intensity" [0.2]
    LightSource "distantlight" 2 "intensity" [1.2]"from" [0 0 -6]"to" [0 0 0]
    # Define a red plastic unit sphere at 0 0 0
    Color [1.0 0.0 0.0]
    Surface "plastic"
    Sphere 1 -1 1 360
WorldEnd
```

In GEL both scenes can be described as:

```
int globalProc() {
    mode rt          // realtime mode

    // A 320 by 240 window that is resizeable.
    window(320, 240, true);
```

```
    // A perspective camera with 30 degree FOV at position 0, 0, 1, looking down the
    z axis with y up.
    camera(<0, 0, -1>,  <0, 0, 1>, <0, 1, 0>, “perspective”);


    // Create a sphere at 1, -1, 1 with radius 1 with a plastic shader.
    sphere(<1, -1, 1>,  1, 360, “plastic”);
}
```

1.3 Method: To accomplish this GEL provides simple functions that allow for the description of 3D scenes using geometric objects and shading functions. When describing the scene GEL’s state is can be set to interactive or non-interactive mode. If GEL is in the interactive state, the scene will be rendered and displayed using an API such as OpenGL. It will automatically create the window and controls required for an interactive environment.

If the state is set to non-interactive mode, GEL will use an API such as Renderman to create a frame, or sequence of frames based on the same scene description. Thus, a single scene is implemented in both domains.

1.4 Related Works: GEL is in essence a very distilled version of much larger possibilities that can be accomplished using a 3D package such as Alias’s Maya or Discreet’s 3DStudioMax. Any of these development packages can describe complex 3D scenes that can be (in most cases) translated easily between domain applications. GEL attempts to emulate a small subset of the possibilities available with these packages in a simple script driven environment to allow the user to describe and render a scene using different methods without the need of a complex and expensive IDE.