Copyright ©1995 Mark J. Kilgard. All rights reserved.

PUBLISHED IN THE
NOVEMBER/DECEMBER 1995 ISSUE OF *The X Journal*.

OPENGL® AND X, COLUMN 7: USING OPEN INVENTOR WITH GLUT

Mark J. Kilgard *
Silicon Graphics Inc.
Revision : 1.2

August 26, 1995

Before turning to this column's technical content, an update on the state of the OpenGL standard is in order. SIGGRAPH '95, the world's largest computer graphics conference, was held in August at the Los Angeles Convention Center. This year's SIGGRAPH conference will be remembered as a milestone event for OpenGL. OpenGL relevant courses, announcements, and exhibits abounded:

- The two half-day courses on "OpenGL Fundamentals" and "Programming OpenGL with X" were each attended by several hundred people.
- Two other courses discussing OpenGL-related topics, VRML (the World Wide Web's Virtual Reality Modeling Language) and Open Inventor (the subject of this column's technical discussion), were also extremely well attended.
- SunSoft announced that it was adopting and licensing OpenGL.
- Microsoft was showing OpenGL running on Windows 95 and NT, including implementations optimized for accelerated graphics hardware.
- Silicon Graphics demonstrated its recently announced IMPACT graphics hardware for the Indigo^2 which is the first SGI hardware designed *specifically* for OpenGL performance.
- Addison-Wesley's booth sold out of its large stock of OpenGL books.
- Kurt Akeley, the primary architect of OpenGL, was awarded SIGGRAPH's Computer Graphics Achievement Award for his contributions to "the architecture, design, and realization of high performance 3D graphics hardware systems."
- SIGGRAPH's OpenGL special interest group had its largest attendance yet. Among the speakers, Kurt Akeley discussed the progress on OpenGL 1.1. Kurt observed that many of the extensions to OpenGL pertain to image processing and texture mapping, indicating that OpenGL is moving beyond a graphics library to become a "*visualization* library" that unites imaging and graphics.

These are just the highlights of OpenGL's presence at the show. OpenGL implementations were visible in the booths of many hardware vendors. The exhibits, events, and announcements made it clear that OpenGL is *the* standard for interactive computer graphics.

The rest of this column focuses on the relationship between OpenGL and Open Inventor and how the two graphics libraries interact. This column's example program demonstrates how to use Open Inventor with the OpenGL Utility Toolkit (or GLUT for short). For those new to this column, GLUT is a publicly available windowing library used in the examples for this column. GLUT simplifies the construction of OpenGL demos, examples, and small applications.

Procedural versus Descriptive

The OpenGL graphics system is a *procedural* standard for interactive 3D graphics. This means a sequence of OpenGL commands describe a specific procedure for drawing a 3D scene. An alternative to a procedural graphics system is a *descriptive* system. An example of a descriptive graphics system is the object-oriented Open Inventor library (or simply Inventor) developed and licensed by Silicon Graphics. Open Inventor is

*Mark graduated with B.A. in Computer Science from Rice University and is a Member of the Technical Staff at Silicon Graphics. He can be reached by electronic mail addressed to `mjk@sgi.com`

related to OpenGL since Inventor is layered on top of OpenGL.

With a procedural graphics system like OpenGL, you draw a scene by executing a series of graphics commands. For example, "enable depth buffering, set up the projection and modelview matrices, setup lighting parameters, set the viewport, clear the color and depth buffers, generate a triangle strip with these vertices, push the modelview matrix, etc." On the other hand, with a descriptive system, you specify a scene as a description of what is to be rendered. For example, "there is a light source shining down on the scene, there is a green plastic cylinder at the origin, there is a brass cube three units left of the cylinder, etc." In brief, a descriptive specification of a 3D scene is about objects, while a procedural specification is about rendering.

Because a descriptive graphics system permits a higher level description of 3D scenes than a procedural one, programmers often find descriptive graphics systems easier to use.

A descriptive graphics system does not eliminate procedural rendering. When a descriptive specification of a scene is to be rendered, a procedural specification must be generated and executed from the higher level descriptive specification. Also, an application using procedural graphics systems may still achieve better performance through efficient use of the lower level procedural interface.

The efficiency of a descriptive graphics system is largely dependent on how well the descriptive system's rendering model is adapted to the lower level procedural system it is used with. In the case of Open Inventor is specifically designed to efficiently utilize OpenGL's rendering functionality. When necessary, Open Inventor still provides access to the lower level OpenGL library.

Open Inventor in Brief

Open Inventor is an object-oriented 3D toolkit. Language bindings for both C++ and ANSI C are provided (though the C++ binding is more convenient and extensible). The library itself is written in C++.

Programs use Open Inventor to manipulate *scene graphs* constructed from *node* objects. Each node is an object that holds information such as a surface material, a shape, a light, or a camera. A scene graph is a tree of nodes describing a 3D scene. An example scene graph is presented graphically in Figure 1.

The node objects are extensible, meaning specialized nodes can be subclassed from existing nodes. Advanced Inventor programmers are encouraged to develop their own nodes if necessary. Rendering the scene described by the scene graph is done by traversing the scene graph and performing the rendering action associated with each node.

Inventor metafiles provide a way to share 3D scene information between applications. An Inventor metafile encodes an Inventor scene graph in an ASCII or binary file.[1] It is easy for an Inventor program to read or write Inventor metafiles which

[1] The Inventor file format is the inspiration for the VRML data format used to interact with 3D virtual worlds via the World Wide Web.

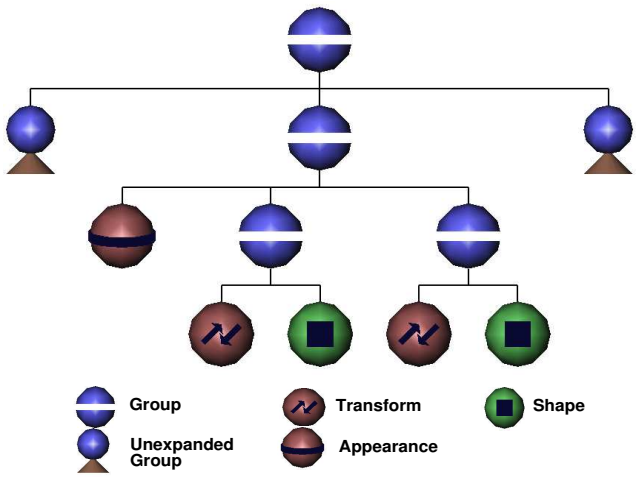

Figure 1: Graphical representation of a partial Inventor scene graph. When rendering a scene graph, Inventor generally traverses the nodes in a depth-first, left-to-right order (though specialized nodes permit more sophisticated traversal methods). To simplify the diagram, the left and right-most group nodes are not expanded to show the nodes below them.

are generally suffixed with the `.iv` extension. A portion of the metafile used in this column's example is shown in Figure 2.

The Inventor toolkit is window system independent (like OpenGL itself), but a component library helps make Inventor more useful for a specific window system. The X Window System implementation of Inventor has a component library (`libInventorXt`) that integrates Inventor with Xt/Motif. This is how most X-based Inventor programs are written. Figure 3 shows how the Inventor libraries fit in the context of related X libraries. This issue's sample program demonstrates that it is also possible to use the base Inventor library (`libInventor`) without the component library.

Open Inventor with GLUT

The `glutduck.c++` source code at the end of this article combines Open Inventor with GLUT. The program draws a duck in a pond. The duck swims a circle in the pond, either in response to user input or by enabling continuous animation of the scene. Figure 4 shows how the program appears on screen. Since GLUT does not use Xt or Motif, the `libInventorXt` library is not used; `glutduck`'s windowing support is supplied by GLUT.

Looking at the source code, you will notice a host of Inventor related header files. All the header files named `<Inventor/nodes/...>` declare C++ classes for Inventor's scene graph nodes. A second type of Inventor C++ classes are *actions*. The header files named `<Inventor/actions/...>` declare C++ classes for Inventor's action objects. Actions are applied to Inventor scene graphs to perform operations on the

```
#Inventor V2.0 ascii

Separator {
  Separator {
    Material {
      ambientColor     0.25396 0.25396 0
      diffuseColor     1 1 0
      specularColor    0.89987 0.89310 0.89682
      emissiveColor    0 0 0
      shininess        0.68108
      transparency     0
    }
    Separator {
      ShapeHints {
        vertexOrdering  COUNTERCLOCKWISE
        shapeType       SOLID
        faceType        CONVEX
      }
      NormalBinding {
        value   PER_VERTEX_INDEXED
      }
      Coordinate3 {
        point [ 0.43192 -0.48431 -0.061775,
                0.42836 -0.47830 -0.078645,
                0.38222 -0.50567 -0.078644,
```

Figure 2: First 25 lines of the `duck.iv` Open Inventor metafile containing a 3D duck model.

scene graph. The most common action is to render the scene graph into a window, but other possible actions include calculating a subgraph's bounding box, writing the scene graph to a file, or dispatching input events to the scene graph's nodes. The rendering action is optimized for OpenGL's interface. For example, Inventor automatically uses OpenGL display lists to cache the OpenGL commands generated by repeatedly rendered Inventor nodes. While nodes and actions are the most common Inventor objects, there are many other object types supported by Inventor which are beyond the scope of this article.

`glutduck`'s use of Inventor is fairly simple. The program's `duckScene` routine creates a scene graph containing the duck in the pond. While the duck model is read from the `duck.iv` Inventor file, the pond and the composition of the duck relative to the pond is done programmatically by calling the necessary Inventor routines.

Updating the scene can be performed by editing and/or manipulating the scene graph nodes. The animation in `glutduck` is done by updating the angle of the duck's rotation node within the scene graph and then redrawing the scene. Inventor also has sophisticated facilities for more automated mechanisms for controlling and updating the scene graph such as sensors, engines, draggers, and manipulators.

To redraw its window, `glutduck` does the following:

1. Uses `glClear` to clear the screen.

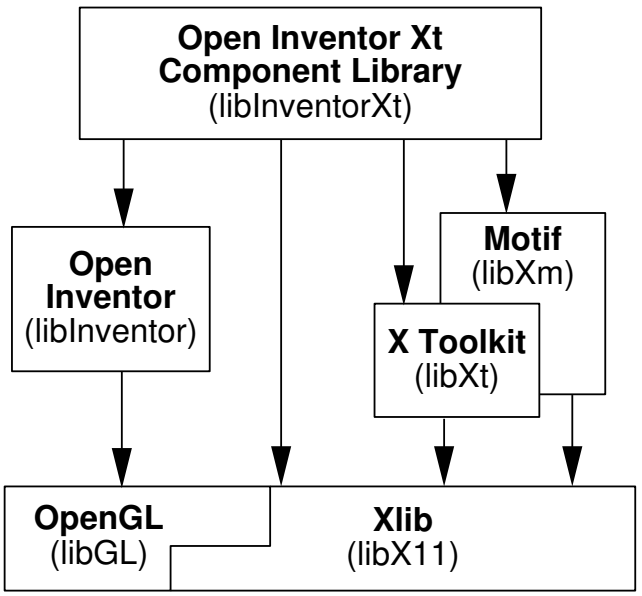

Figure 3: Relationship between Open Inventor libraries and OpenGL and X libraries.

2. Instantiates a `SoGLRenderAction` with its viewport set correctly to the size of the rendering window.

3. Applies the `SoGLRenderAction` object's action to the root of the duck pond scene graph. The action performed by `SoGLRenderAction` is to traverse the scene graph invoking each node's OpenGL render action. This process renders the duck pond.

4. The window is double buffered; to display the newly rendered scene, `glutSwapBuffers` is called to swap the window's buffers.

The `SoGLRenderAction` action causes the necessary OpenGL rendering commands to be issued. The action assumes that the OpenGL library is already bound to the correct context and window.

The user can interact with `glutduck` in several ways. A pop-up menu controls the animation of the scene. Mouse motion when the left mouse button is held down controls the duck's rotation, and hitting the space bar advances the duck's rotation in steps.

Not demonstrated in the `glutduck` program is Open Inventor's Xt-based component library. This library makes it easy to use Inventor with Motif. Some of the facilities provided by the library are:

- An `SoXtRenderArea` object that encapsulates an OpenGL drawing area widget for easy use with Inventor.
- Support for overlay planes.
- User interface components such as a directional light editor, a material editor, and a scene viewer.

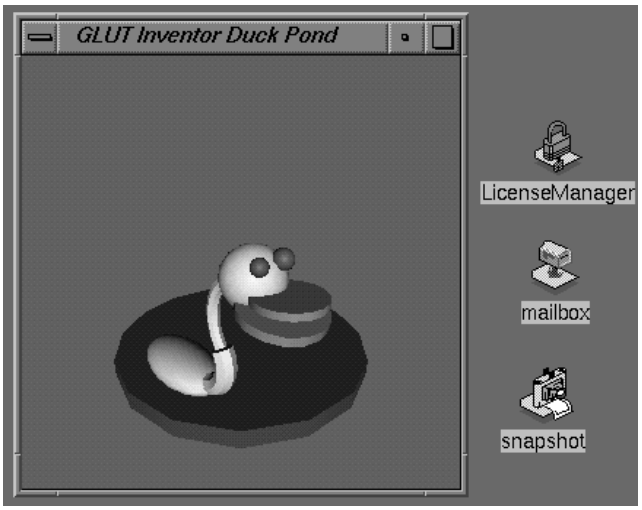

Figure 4: Screen snapshot of `glutduck` using Open Inventor.

- Clipboard support for passing 3D Inventor scene graph information between applications.

Open Inventor is a powerful tool for utilizing OpenGL. Because Inventor has a high-level, object-oriented descriptive model, Inventor simplifies developing programs that manipulate 3D data. In addition, Open Inventor is designed to efficiently generate OpenGL commands, so Inventor leverages the low-level tuning and hardware acceleration of the underlying OpenGL implementation.

Further Information

More information about programming with Open Inventor can be found in the *Inventor Mentor* [1]. The recently released GLUT 2.3 source code distribution contains the source code to `glutduck` and other GLUT examples using Open Inventor.[2] If you plan on using Open Inventor with GLUT, the *Inventor Mentor* chapter titled "Using Inventor with OpenGL" contains useful information.

Next issue will explore OpenGL's capabilities for generating smooth, curved surfaces.

[2] You can get GLUT 2.3 by anonymous ftp to `sgigate.sgi.com`; look in the `pub/opengl/xjournal/GLUT` directory.

A glutduck.c++

```
// Based on an example from the Inventor Mentor chapter 13, example 5.

#include <stdio.h>
#include <unistd.h>
#include <GL/gl.h>
#include <GL/glut.h>
#include <Inventor/SoDB.h>
#include <Inventor/SoInput.h>
#include <Inventor/SbViewportRegion.h>
#include <Inventor/actions/SoGLRenderAction.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoCylinder.h>
#include <Inventor/nodes/SoDirectionalLight.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoPerspectiveCamera.h>
#include <Inventor/nodes/SoRotationXYZ.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/nodes/SoTranslation.h>

int spinning = 0, moving  = 0, begin;
SoSeparator *root;
SoRotationXYZ *duckRotXYZ;
float angle = 0.0;
SbViewportRegion myViewport;

void reshape(int w, int h) {
  glViewport(0, 0, w, h);
  myViewport.setWindowSize(w,h);
}

void renderScene(void) {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  SoGLRenderAction myRenderAction(myViewport);
  myRenderAction.apply(root);
}

void redraw(void) {
  renderScene();
  glutSwapBuffers();
}

int duckScene(void) {
  root = new SoSeparator;
  root->ref();

  // Add a camera and light
  SoPerspectiveCamera *myCamera = new SoPerspectiveCamera;
  myCamera->position.setValue(0., -4., 8.0);
  myCamera->heightAngle = M_PI/2.5;
  myCamera->nearDistance = 1.0;
  myCamera->farDistance = 15.0;
  root->addChild(myCamera);
  root->addChild(new SoDirectionalLight);

  // Rotate scene slightly to get better view
  SoRotationXYZ *globalRotXYZ = new SoRotationXYZ;
  globalRotXYZ->axis = SoRotationXYZ::X;
  globalRotXYZ->angle = M_PI/9;
```

```
  root->addChild(globalRotXYZ);

  // Pond group
  SoSeparator *pond = new SoSeparator;
  root->addChild(pond);
  SoMaterial *cylMaterial = new SoMaterial;
  cylMaterial->diffuseColor.setValue(0., 0.3, 0.8);
  pond->addChild(cylMaterial);
  SoTranslation *cylTranslation = new SoTranslation;
  cylTranslation->translation.setValue(0., -6.725, 0.);
  pond->addChild(cylTranslation);
  SoCylinder *myCylinder = new SoCylinder;
  myCylinder->radius.setValue(4.0);
  myCylinder->height.setValue(0.5);
  pond->addChild(myCylinder);

  // Duck group
  SoSeparator *duck = new SoSeparator;
  root->addChild(duck);

  // Read the duck object from a file and add to the group
  SoInput myInput;
  if (!myInput.openFile("duck.iv"))  {
    if (!myInput.openFile("/usr/share/src/Inventor/examples/data/duck.iv")) {
      return 1;
    }
  }
  SoSeparator *duckObject = SoDB::readAll(&myInput);
  if (duckObject == NULL) return 1;

  // Set up the duck transformations
  duckRotXYZ = new SoRotationXYZ;
  duck->addChild(duckRotXYZ);
  duckRotXYZ->angle = angle;
  duckRotXYZ->axis = SoRotationXYZ::Y;  // rotate about Y axis
  SoTransform *initialTransform = new SoTransform;
  initialTransform->translation.setValue(0., 0., 3.);
  initialTransform->scaleFactor.setValue(6., 6., 6.);
  duck->addChild(initialTransform);
  duck->addChild(duckObject);

  return 0;
}

void updateModels(void) {
  duckRotXYZ->angle = angle;
  glutPostRedisplay();
}

void animate(void) {
  angle += 0.1;
  updateModels();
}

void setAnimation(int enable) {
  if(enable) {
    spinning = 1;
    glutIdleFunc(animate);
  } else {
    spinning = 0;
```

```
    glutIdleFunc(NULL);
    glutPostRedisplay();
  }
}

void keyboard(unsigned char ch, int x, int y) {
  if(ch == ' ') {
    setAnimation(0);
    animate();
  }
}

void menuSelect(int item) {
  switch(item) {
  case 1:
    setAnimation(0);
    animate();
    break;
  case 2:
    if(!spinning) setAnimation(1);
      else setAnimation(0);
    break;
  }
}

void vis(int visible) {
  if (visible == GLUT_VISIBLE) {
    if (spinning) glutIdleFunc(animate);
  } else {
    if (spinning) glutIdleFunc(NULL);
  }
}

void mouse(int button, int state, int x, int y) {
  if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
    setAnimation(0);
    moving = 1;
    begin = x;
  }
  if (button == GLUT_LEFT_BUTTON && state == GLUT_UP) {
    moving = 0;
    glutPostRedisplay();
  }
}

void motion(int x, int y) {
  if (moving) {
    angle = angle + .01 * (x - begin);
    begin = x;
    updateModels();
  }
}

void main(int argc, char **argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH | GLUT_MULTISAMPLE);
  SoDB::init();
  if(duckScene()) {
    fprintf(stderr, "couldn't read IV file\n");
    exit(1);
```

```
  }
  glutCreateWindow("GLUT Inventor Duck Pond");
  glutReshapeFunc(reshape);
  glutDisplayFunc(redraw);
  glutCreateMenu(menuSelect);
  glutAddMenuEntry("Step", 1);
  glutAddMenuEntry("Toggle animation", 2);
  glutAttachMenu(GLUT_RIGHT_BUTTON);
  glutKeyboardFunc(keyboard);
  glutMouseFunc(mouse);
  glutMotionFunc(motion);
  glutVisibilityFunc(vis);
  glEnable(GL_DEPTH_TEST);
  glClearColor(0.132, 0.542, 0.132, 1.0);
  glutMainLoop();
}
```

References

[1] Josie Wernecke, *The Inventor Mentor*, Addison Wesley, 1993.