Realtime 3D Computer Graphics & Virtual Reality

OpenGL Introduction

VR-programming

- Input and display devices are the main hardware interface to users
- Immersion embeds users through the generation of live-like sensory experiences
  - But how is the programmers/designers view?

VR-programming tools

- Direct rendering and gfx packages
  - OpenGL, Direct3D, GKS (3D)
- Scene graph based tools
  - VRML, OpenGL Performer, OpenGL Optimizer, Open Inventor, PHIGS+
- VR modeling toolkits
  - AVANGO, World toolkit, Masive1-3, Dive, Lightning, game engines

A Scene Graph Language: VRML

```xml
VRML V2.0 utf8
Transform {
  translation -3 0 0
  children Shape {
    geometry Box {
      appearance Appearance {
        material Material {
          diffuseColor .8 .2 .2
        }
      }
    }
  }
}
Transform {
  translation 3 0 0
  children Shape {
    geometry Cone {
      appearance Appearance {
        material Material {
          diffuseColor .2 .2 .8
        }
      }
    }
  }
}
```

More VRML later in this course!

What is a gfx package?

- software
  - that takes user input and passes it to applications
  - that displays graphical output for applications

An Interactive Introduction to OpenGL Programming

Partly based on SIGGRAPH course notes by Dave Shreiner, Ed Angel and Vicki Shreiner
What You’ll See

- General OpenGL Introduction
- Rendering Primitives
- Rendering Modes
- Lighting
- Texture Mapping
- Additional Rendering Attributes
- Imaging

Goals

- Demonstrate enough OpenGL to write an interactive graphics program with
  - custom modeled 3D objects or imagery
  - lighting
  - texture mapping
- Introduce advanced topics for future investigation
-Generate knowledge to understand high-level scene graph based engines for VE-design

OpenGL and GLUT Overview

- What is OpenGL & what can it do for me?
- OpenGL in windowing systems
- Why GLUT
- A GLUT program template

What Is OpenGL?

- OpenGL - Open Graphics Library
- Graphics rendering API
  - high-quality color images composed of geometric and image primitives
  - window system independent
  - operating system independent
  - hardware independent layer to different acceleration designs (supporting software modes as well)

What Is OpenGL?

- Introduced 1992 by SGI
- Based on IRIS GL, an API for the SGI personal IRIS workstation and follow-ups
- Now an open standard that is widely adopted for all types of applications
- Under the supervision of the OpenGL architecture review board
OpenGL Design Goals

- SGI’s design goals for OpenGL:
  - High-performance (hardware-accelerated) graphics API
  - Some hardware independence
  - Natural, terse API with some built-in extensibility
- OpenGL has become a standard because:
  - It doesn’t try to do too much
    - Only renders the image, doesn’t manage windows, etc.
  - It does enough
    - Useful rendering effects + high performance
    - It is promoted by SGI (& Microsoft, half-heartedly)

OpenGL Architecture

OpenGL as a Renderer

- Geometric primitives
  - points, lines and polygons
- Image Primitives
  - images and bitmaps
  - separate pipeline for images and geometry
    - linked through texture mapping
- Rendering depends on state
  - colors, materials, light sources, etc.

Related APIs

- AGL, GLX, WGL
  - glue between OpenGL and windowing systems
- GLU (OpenGL Utility Library)
  - part of OpenGL
  - NURBS, tessellators, quadric shapes, etc.
- GLUT (OpenGL Utility Toolkit)
  - portable windowing API
  - not officially part of OpenGL

OpenGL and Related APIs

OpenGL: Conventions

- Functions in OpenGL start with gl
  - Most functions just gl (e.g., glColor())
  - Functions starting with glu are utility functions (e.g., gluLookAt() )
  - Functions starting with glx are for interfacing with the X Windows system (e.g., in gfx.c)
**OpenGL: Conventions**

- Variables written in CAPITAL letters
  - Example: GLUT_SINGLE, GLUT_RGB
- usually constants
- use the bitwise or command \((x \mid y)\) to combine constants

**Preliminaries**

- **Headers Files**
  - `#include <GL/gl.h>`
  - `#include <GL/glu.h>`
  - `#include <GL/glut.h>`
- **Compile with libraries**
  - `cc myapp.c -o myapp -lglut -lglu -lglut -lX11`
  - Adopt different library places using e.g. `-L/usr/...`

**Preliminaries**

- Simple make looks like
  - `CC = cc`
  - `LDLIBS = -lglut -lglu -lglut -lX11 -l/usr/...`
  - `$CC$LDLIBS -o $@`
- **Enumerated Types**
  - OpenGL defines numerous types for compatibility between different systems
  - `.GLfloat, GLint, GLenum, etc.

**Enumerated Types**

- `GLvoid`  
- `GLuint`  
- `GLenum`  
- `GLbitfield`
- `GLushort`  
- `GLubyte`, `GLboolean`
- `GLdouble`  
- `GLfloat`  
- `GLint`, `GLsizei`
- `GLshort`  
- `GLbyte`

**OpenGL Command Formats**

```
glVertex3fv( v )
```

**Application Structure**

- Configure and open window
- Initialize OpenGL state
- Register input callback functions
  - `render`
  - `resize`
  - `input: keyboard, mouse, etc.`
- Enter event processing loop
Basic OpenGL template

```
/* simple program template for OpenGL progs */
#include <GL/glut.h>
void myDisplay()
{
    /* clear the window */
    glClear(GL_COLOR_BUFFER_BIT);
    /* draw something */
    glBegin(GL_LINES);
    glVertex2f(0.5, 0.5);
    glVertex2f(0.5, 0.5);
    glEnd();
    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutCreateWindow("basic template 1");
    glutDisplayFunc(myDisplay);
    glutMainLoop();
}
```

Sample Program

```
void main( int argc, char** argv )
{
    glutInit( argc, argv );
    int mode = GLUT_RGB|GLUT_SINGLE;
    glutInitDisplayMode( mode );
    glutCreateWindow( argv[0] );
    init();
    glutDisplayFunc( display );
    glutKeyboardFunc( key );
    glutMouseFunc( mouse );
    glutIdleFunc( idle );
    glutMainLoop();
}
```

OpenGL Initialization

- Set up whatever state you're going to use

```c
void init( void )
{
    glClearColor( 0.0, 0.0, 0.0, 1.0 );
    glColor3f( 1.0, 1.0, 1.0);
    glClearDepth( 1.0 );
    glEnable( GL_LIGHT0 );
    glEnable( GL_LIGHTING );
    glEnable( GL_DEPTH_TEST );
}
```

GLUT Callback Functions

- A callback is a routine to call when something happens
  - window resize or redraw
  - user input
  - animation

```
tag: callbacks with GLUT

void display( void )
{
    glClear( GL_COLOR_BUFFER_BIT );
    glBegin( GL_LINES );
    glVertex2f( 50.0, 50.0 );
    glVertex2f( 100.0, 100.0 );
    glVertex2f( 70.0, 10.0 );
    glVertex2f( 100.5, 70.1 );
    glEnd();
    glFlush();
}
```

GLUT Callback Functions

- "Register" callbacks with GLUT

```
void glutDisplayFunc( display );
void glutIdleFunc( idle );
void glutResizeFunc( resize );
void glutKeyboardFunc( keyboard );
void glutSpecialFunction( special )
void glutMouseFunc( mouse );
void glutMotionFunc( mouse_motion );
void glutPassiveMotionFunc( mouse_motion );
void glutEntryFunc( on_focus_change );
```
### Idle Callbacks

- Use for animation and continuous update
  ```c
  glutIdleFunc( idle );
  void idle( void )
  {
    t += dt;
    glutPostRedisplay();
  }
  ```

### "smart" update

- Requests that the display callback be executed
- Allows the implementation to be smarter in deciding when to carry out the display callback
  - As GLUT goes through the event loop, more than one event can require a redraw which should only be carried out once during the loop
  ```c
  glutPostRedisplay();
  ```

### Idle callback and smart update

- Processing an animation should be done with respect to the elapsed time
  - `t += dt`
- No hint when the update occurs
- How can we achieve a minimal simulation and frame rate using this application structure?

### User Input Callbacks

#### Process user keyboard input

```c
glutKeyboardFunc( keyboard );
void keyboard( char key, int x, int y )
{
  switch( key ) {
  case 'q' : case 'Q' :
    exit( EXIT_SUCCESS );
  break;
  case 'r' : case 'R' :
    rotate = GL_TRUE;
    break;
  }
}
```

#### Process user special keyboard input

```c
glutSpecialFunction( special );
void special( char key, int x, int y )
{
  if( key == GLUT_KEY_F1)    help();
  if( key == GLUT_KEY_UP)    up();
  if( key == GLUT_KEY_DOWN)  down();
  if( key == GLUT_KEY_LEFT) left();
  if( key == GLUT_KEY_RIGHT) right();
}
```

### User Input Callbacks

#### Process user mouse input

```c
glutMouseFunc( mouse );
void mouse( int button, int state, int x, int y )
{
  if (state == GLUT_DOWN && button == GLUT_LEFT_BUTTON)
    exit(EXIT_SUCCESS);
}
```
User Input Callbacks
- Process user mouse motion input with a pressed button
  
  ```c
  glutMotionFunc( mouse_motion );
  void mouse_motion( int x, int y )
  {
    if (first_time_called)
      glBegin();
    ...
    glEnd();
    first_time_called = GL_FALSE;
  }
  ```

User Input Callbacks
- Process user mouse motion input without a button pressed
  
  ```c
  glutPassiveMotionFunc( mouse_pmotion );
  void mouse_pmotion( int x, int y )
  {
    last_points_visited.push(pair(x,y));
    if( last_points_visited.size() > 100)
      last_points_visited.remove_last();
  }
  ```

User Input Callbacks
- Process leaving and entering the OpenGL window with the mouse
  
  ```c
  glutEntryFunc( on_focus_change );
  void on_focus_change( int state )
  {
    if (state == GLUT_ENTERED)
      beep();
    if (state == GLUT_LEFT)
      exit(EXIT_SUCCESS);
  }
  ```