Haptics
ME7960, Sect. 007
Lect. 5: OpenGL and Lab Code
Template Overview

Spring 2011
Presented by Andrew Doxon
Prof. William Provancher
University of Utah
Salt Lake City, UT USA

We would like to acknowledge the many colleagues whose course materials were borrowed and adapted in putting together this course, namely: Drs. Allison Okamura (JHU), Katherine Kuchenbecker (U Penn), Francois Conti, Federico Barbagli, and Kenneth Salisbury (Stanford), Ed Colgate (Northwestern), Hong Tan (Purdue), Blake Hannaford and Ganesh Sankaranarayanan (U. Washington), and Karon MacLean (UBC).

Today’s Class

- OpenGL Overview
- Overview of Lab code template
- Reading for next class

- Reading for Discussion Thursday, Feb. 3

OpenGL

- OpenGL is a coding interface (API) for creating interactive 2D and 3D graphics applications.
- It was first introduced in 1992 and since then has become an industry standard supported in every major OS.
- OpenGL is primarily in competition with DirectX the other leading industry standard.
- Higher level APIs such as Qt and FLTK are simply wrappers that make utilizing OpenGL and DirectX easier.
- Due to the open source nature of OpenGL it is constantly evolving. There are many free tutorials and pieces of example code available throughout the internet.
- http://nehe.gamedev.net/ provides great tutorials and examples.

What Each Part is Responsible For

genGL
- Lines and polygons, Material properties, Lighting, Renders the scene

GLUT
- Window applications, Keyboard support, Simple and easy to use, Some 3D shapes

GLU
- Higher level drawing utilities, Other 3D shapes, Texturing, NURBS handling, Tessellation
Naming conventions

- Functions each start with their parent API's name.
- There are no spaces and each word starts with a capital letter except the first.
- Predefined parameters are spelled in all caps with underscores instead of spaces.

Predefined parameter

OpenGL function

- glEnable(GL_LIGHTING);
- glutSolidSphere(1.2, 20, 20);
- gluLookAt(0.15, 0, 0, 0, 0, 0, 0, 0, 1);

GLU function

Writing an openGL program

1. Include necessary headers & libraries
   - Provide function calls
2. Initialize glut window
   - Creates the window
   - Sets your callback functions
3. Initialize the environment
   - Materials
   - Lighting
4. Start main glut loop
   - Starts message driven loop
   - Calls callback functions as necessary
5. Write each callback function
   - Idle, keyboard, display, …

Idle Callback

- When no messages are received to handle any other callbacks.

- Usually put an artificial call for a redisplay in the idle function to re-render your scene.

```c
void IdleCB(void)
{
    Sleep(10);  // Slow down redisplay calls
    glutPostRedisplay();  // Send a message telling the system to call the display callback
}
```

Keyboard Callback

- Multiple options
  - Special keys (arrow keys and F# keys)
  - Normal keys (ascii keys, abc123)
  - Keydown, Keyup (pressing and releasing keys)

- Switch statements are used for processing which key is pressed.

```c
void keySelect(unsigned char key, int x, int y);
void special_keydown(int key, int x, int y);
void special_keyup(int key, int x, int y);
```
Display Callback

- Where the graphics are defined
- OpenGL creates a stack of commands (order matters)
- Outline
  - Clear current buffer
  - Set matrix mode to GL_MODELVIEW to be safe
  - Load an identity matrix to clear previous alterations
  - Position camera
  - Draw objects
  - Swap buffers to smooth rendering
  - Check for errors

Rendering Objects

- **Always** frame your object calls with push and pop matrix commands.
- Clean up after yourself. Everything is remembered. If you are not going to set the variable every time, return it to its default state.
- Lines and polygons are drawn by vertex locations with respect to the origin.
  - These vertex lists are framed by glBegin() and glEnd()
- 3D objects (cubes, spheres, …) are drawn at the origin and are aligned with the principle axes.
  - glTranslated() and glTranslatef() move the origin
  - glRotated() and glRotatef() rotate the axes

Lines and Polygons

```c
// Green object (%RGB)
glPushMatrix();
glColor3f(0.0, 1.0, 0.0);
gLineWidth(2);
glDisable(GL_LIGHTING);
glBegin(GL_LINE_STRIP);
  double angle = 0;
  for(int i=0; i<=100; ++i)
  {
    angle = i*2*M_PI/100;
    glVertex3d(0,radius*cos(angle),radius*sin(angle));
  }
glEnd();
glEnable(GL_LIGHTING);
gLineWidth(1);
glPopMatrix();
```

3D Objects

```c
// Anything drawn after will be white (%RGB)
glPushMatrix();
glColor3f(1.0,1.0,1.0);
glTranslated(1,1,1);
glRotated(60.0,0,0,0);
glutSolidCube (1.5);
glPopMatrix();
```

- Move origin first so we get proper position
- Rotate origin 60° about X axis.
  - Remember this affects the coordinate system not the object so if it was done before the translate it would change the direction of the translate in our world space.
- Place a cube with 1.5 unit sides.
  - This cube will appear at the origin and aligned with the current axes. Thus it will be at <1,1,1> in world space and rotated 60 about its X axis.
**Haptic Rendering**

- Separate thread

```cpp
Device()->getPosition(currentPosition);
cVector3d renderedForce(0,0,0);
double gain = 600;
renderedForce = gain*(attractingPoint-currentPosition);
Device()->setForce(renderedForce);
```

**Multithreaded Applications**

Sharing variables between threads is Dangerous. Threads run asynchronously and are nondeterministic (you can’t predict when they will try and access something)

Concurrency issues are handled in 3 main ways
1. Making read/write operations atomic
2. Place locks on key variables while in use
3. Make sure variables are passed-by-value not passed-by-reference between threads.

**Template Structure**