Chapter 3

Rendering Pipeline
OpenGL/Glut

3D Graphics
- Modeling
  - representing object properties
    - geometry: polygons, smooth surfaces etc.
    - materials: reflection models etc.
- Rendering
  - generation of images from models
    - interactive rendering
    - ray-tracing
- Animation
  - making geometric models move and deform

Rendering
- Goal
  - transform computer models into images
    - photo-realistic or not
- Interactive rendering
  - fast, but limited quality
  - roughly follows a fixed patterns of operations
    - rendering pipeline
- Offline rendering
  - ray-tracing
  - global illumination

The Rendering Pipeline
Geometry Database
Model/View Transform. "Lighting" Perspective Transform. Clipping
Scan Conversion Texturing Depth Test Blending Frame-buffer

Geometry Database
- Geometry database:
  - Application-specific data structure for holding geometric information
  - Depends on specific needs of application
    - Independent triangles, connectivity information etc.

Copyright: Alla Sheffer, UBC, 2005
Model/View Transformation

- **Modeling transformation:**
  - Map all geometric objects from a local coordinate system into a world coordinate system

- **Viewing transformation:**
  - Map all geometry from world coordinates into camera coordinates

Lighting

- **Lighting:**
  - Compute the brightness of every point based on its material properties (e.g. Lambertian diffuse) and the light position(s)
  - Computation is performed *per-vertex*

Perspective Transformation

- **Perspective transformation**
  - Projecting the geometry onto the image plane
  - Projective transformations and model/view transformations can all be expressed with 4x4 matrix operations

Clipping

- **Clipping**
  - Removal of parts of the geometry that fall outside the visible screen or window region
  - May require re-tessellation of geometry

Scan Conversion

- **Scan conversion**
  - Turn 2D drawing primitives (lines, polygons etc.) into individual pixels (discretizing/sampling)
  - Interpolate color across primitive
  - Generate discrete fragments

Texture Mapping

- **Texture mapping**
  - "gluing images onto geometry"
  - Color of every fragment is altered by looking up a new color value from an image

Copyright: Alla Sheffer, UBC, 2005
Depth Test
- Depth test:
  - Remove parts of geometry hidden behind other geometry
  - Perform on every individual fragment
  - other approaches (later)

Blending
- Blending:
  - Final image: write fragments to pixels
  - Draw from farthest to nearest
  - No blending – replace previous color
  - Blending: combine new & old values with some arithmetic operations
  - Framebuffer: video memory on graphics board that holds resulting image & used to display it

The Rendering Pipeline

OpenGL
- started in 1989 by Kurt Akeley
- based on IRIS_GL by SGI
- API to graphics hardware
- designed to exploit hardware optimized for display and manipulation of 3D graphics
- implemented on many different platforms
- low level, powerful flexible
- pipeline processing
- set state as needed
### Computer Graphics

#### Graphics State
- **setColor once, remains until overwritten**
  - `glColor3f(1.0, 1.0, 0.0)` → set color to yellow
  - `glSetClearColor(0.0, 0.0, 0.2)` → dark blue bg
- **turn on light**
  - `glEnable(LIGHT0)`
- **hidden surf.**
  - `glEnable(GL_DEPTH_TEST)`

### Rendering Pipeline

#### Geometry Pipeline
- **how to interpret geometry**
  - `glBegin(<mode of geometric primitives>)`
  - `mode = GL_TRIANGLE, GL_POLYGON, etc.`
- **feed vertices**
  - `glVertex3f(-1.0, 0.0, -1.0)`
  - `glVertex3f(1.0, 0.0, -1.0)`
  - `glVertex3f(0.0, 1.0, -1.0)`
- **done**
  - `glEnd()`

### OpenGL: Primitives
- `glPointSize(float size);`
- `glLineWidth(float width);`
- `glColor3f(float r, float g, float b);`

### OpenGL Example
- **TRIANGLE...**
  - `glColor3f(0, 1, 0);`
  - `glBegin(GL_TRIANGLES);`
  - `glVertex3f(0.0f, 0.5f, 0.0f);`
  - `glVertex3f(-0.5f, 0.5f, 0.0f);`
  - `glVertex3f(0.5f, 0.0f, 0.0f);`
  - `glEnd();`

### GLUT: OpenGL Utility Toolkit
- **The basics...**
  ```c
  int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize(640, 480);
    glutCreateWindow("openGLDemo");
    glutDisplayFunc(DrawWorld);
    glutIdleFunc(Idle);
    glutCreateWindow("openGLDemo");
    glClearColor(1,1,1);
    glutMainLoop();
    return 0;       // never reached
  }
  ```

### Event-Driven Programming
- **main loop not under your control**
  - vs. procedural
- **control flow through event callbacks**
  - redraw the window now
  - key was pressed
  - mouse moved
  - callback functions called from main loop when events occur
  - mouse/keyboard state setting vs. redrawing
OpenGL/GLUT Example

```c
void DrawWorld() {
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    glClear( GL_COLOR_BUFFER_BIT );
    angle += 0.05;
    glRotatef(angle,0,0,1);
    ...  // draw triangle
    glutSwapBuffers();
}
```

GLUT Example

```c
void Idle() {
    angle += 0.05;
    glutPostRedisplay();
}
```

GLUT Input Events

```c
// you supply these kind of functions
void reshape(int w, int h);
void keyboard(unsigned char key, int x, int y);
void mouse(int but, int state, int x, int y);

// register them with glut
glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
glutMouseFunc(mouse);
```

GLUT and GLU primitives

```c
gluSphere(...);
gluCylinder(...);
glutSolidSphere(...);
glutWireSphere(...);
glutSolidCube(...);
glutWireCube(...);
glutSolidTorus(...);
glutWireTorus(...);
glutSolidTeapot(...);
glutWireTeapot(...);
```

Depth buffer

- for visibility
- stores a z-value for every pixel
- smaller z means “closer”

```c
// allocate depth buffer
glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

// enabling the depth test
glEnable( GL_DEPTH_TEST );

// clearing the depth buffer for each frame
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
```

GLUT menus

```c
glutCreateMenu(...);
glutGetMenu(...);
glutAddMenuEntry(...);
glutAddSubMenu(...);
glutAttachMenu(...);

// Example usage
glutCreateMenu(demo_menu);
glutAddMenuEntry("quit", 1);
glutAddMenuEntry("Increase Square Size", 2);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```
Assignment 0

- Programming:
  - Experience OpenGL & GLUT
  - See “real” models – meshes in OBJ format
- Theory:
  - Basic math review
- Description:
  - [http://www.cs.ubc.ca/~cs314/Vsep2004/a0/a0.pdf](http://www.cs.ubc.ca/~cs314/Vsep2004/a0/a0.pdf)

- Deadline: Sep 23
- Basis for future assignments