**Basic Graphics Programming**

**What is OpenGL**
- A low-level graphics API for 2D and 3D interactive graphics.
- Descendent of GL (from SGI)

**OpenGL is cross-platform**
- Same code works with little/no modifications
- Implementations:
  - Windows, Mac: ships with the OS
  - Linux: Mesa, a freeware implementation.

```c
#if defined(__WIN32) || defined(linux)
#include <GL/gl.h>
#include <GL/glu.h>
#elif defined(__APPLE__)
#include <OpenGL/gl.h>
#include <GLUT/glu.h>
#include <GLUT/glut.h>
#endif
```

**How does OpenGL work**
From the programmer’s point of view:
1. Specify geometric objects
2. Describe object properties
   - Color
   - How objects reflect light
3. Define how objects should be viewed
   - where is the camera
   - what type of camera
4. Move camera or objects around for animation

**The result**
OpenGL is a state machine

State variables: color, current viewing position, line width, material properties...

These variables (the state) then apply to every subsequent drawing command

They persist until set to new values by the programmer

OpenGL Library Organization

• GL (Graphics Library): core graphics capabilities
• GLU (OpenGL Utility Library): utilities on top of GL
• GLUT (OpenGL Utility Toolkit): input and windowing

Graphics Pipeline

• Application generates stream of geometric primitives (polygons, lines)
• System draws each one into the framebuffer
• Entire scene redrawn anew every frame
• Contrast: off-line rendering (e.g., Pixar Renderman)

The pipeline is implemented by OpenGL, graphics driver and the graphics hardware

OpenGL programmer does not need to implement the pipeline.

However, pipeline is reconfigurable if needed ➔ “shaders”

Immediate-mode rendering

• Efficiently implementable in hardware (but not in software)
• Each stage can employ multiple specialized processors, working in parallel, busses between stages
• #processors per stage, bus bandwidths are fully tuned for typical graphics use
• Latency vs throughput
**Vertices**

- Vertices in world coordinates
- void glVertex3f(GLfloat x, GLfloat y, GLfloat z)
  - Vertex (x, y, z) sent down the pipeline
  - Function call returns
- Use GL for portability and consistency
- glVertex{234}{sfid}[v](TYPE coords)

**Transformer**

- Transformer in world coordinates
- Must be set before object is drawn!
  - glRotatef(45.0, 0.0, 0.0, -1.0);
  - glTranslatef(1.0, 0.0, 0.0);
- Complex [Angel Ch. 4]

**Clipper**

- Mostly automatic (must set viewport)

(a) (b)

**Projector**

- Complex transformation [Angel Ch. 5]

- Orthographic
- Perspective

**Rasterizer**

- Interesting algorithms [Angel Ch. 7]
- To window coordinates
- Antialiasing

**Primitives**

- Specified via vertices
- General schema
  - glBegin(type);
  - glVertex3f(x1, y1, z1);
  - glEnd();
- glVertex3f(xN, yN, zN);
- type determines interpretation of vertices
- Can use glVertex2f(x,y) in 2D
Example: Draw Square Outline

- Type = GL_LINE_LOOP

\[
\begin{align*}
\text{glBegin(GL\_LINE\_LOOP);} \\
\text{glVertex3f(0.0, 0.0, 0.0);} \\
\text{glVertex3f(1.0, 0.0, 0.0);} \\
\text{glVertex3f(1.0, 1.0, 0.0);} \\
\text{glVertex3f(0.0, 1.0, 0.0);} \\
\text{glEnd();}
\end{align*}
\]

- Calls to other functions are allowed between glBegin(type) and glEnd();

Points and Line Segments

Points and Line Segments

Polygons

- Polygons enclose an area

\[
\begin{align*}
\text{glBegin(GL\_POINTS);} \\
\text{glVertex3f(…);} \\
\text{…} \\
\text{glVertex3f(…);} \\
\text{glEnd();}
\end{align*}
\]

- Rendering of area (fill) depends on attributes
- All vertices must be in one plane in 3D

Polygon Restrictions

- OpenGL Polygons must be simple
- OpenGL Polygons must be convex

\[
\begin{align*}
\text{(a) simple, but not convex} \\
\text{(c) convex} \\
\text{(b) non-simple}
\end{align*}
\]

Why Polygon Restrictions?

- Non-convex and non-simple polygons are expensive to process and render
- Convexity and simplicity is expensive to test
- Behavior of OpenGL implementation on disallowed polygons is "undefined"
- Some tools in GLU for decomposing complex polygons (tessellation)
- Triangles are most efficient

Polygon Strips

- Efficiency in space and time
- Reduces visual artefacts

- Polygons have a front and a back, possibly with different attributes!
Attributes:
color, shading and reflection properties

- Part of the OpenGL state
- Set before primitives are drawn
- Remain in effect until changed!

Physics of Color

- Electromagnetic radiation
- Can see only tiny piece of the spectrum

Color Filters

- Eye can perceive only 3 basic colors
- Computer screens designed accordingly

Color Spaces

- RGB (Red, Green, Blue)
  - Convenient for display
  - Can be unintuitive (3 floats in OpenGL)
- HSV (Hue, Saturation, Value)
  - Hue: what color
  - Saturation: how far away from gray
  - Value: how bright
- Other formats for movies and printing

RGB vs HSV

Gimp Color Picker

Example: Drawing a shaded polygon

Initialization: the “main” function

```c
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize (500, 500);
    glutInitWindowPosition (100, 100);
    glutCreateWindow (argv[0]);
    init ();
    ... 
} ```
GLUT Callbacks

- Window system independent interaction
- glutMainLoop processes events

```c
void init(void)
{
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMainLoop();
    return 0;
}
```

Initializing Attributes

- Separate in "init" function

```c
void init(void)
{
    glClearColor (0.0, 0.0, 0.0, 0.0);
    /* glShadeModel (GL_FLAT); */
    glShadeModel (GL_SMOOTH);
}
```

The Display Callback

- The routine where you render the object
- Install with glutDisplayFunc(display)

```c
void display(void)
{
    glClear (GL_COLOR_BUFFER_BIT); /* clear buffer */
    setupCamera();  /* set up the camera */
    triangle();  /* draw triangle */
    glutSwapBuffers(); /* force display */
}
```

The Image

```c
glShadeModel(GL_FLAT)  glShadeModel(GL_SMOOTH)
```

Flat vs Smooth Shading

- In world coordinates; remember state!

```c
void triangle(void)
{
    glBegin (GL_TRIANGLES);
    glColor3f (1.0, 0.0, 0.0); /* red */
    glVertex2f (5.0, 5.0);
    glColor3f (0.0, 1.0, 0.0); /* green */
    glVertex2f (25.0, 5.0);
    glColor3f (0.0, 0.0, 1.0); /* blue */
    glVertex2f (5.0, 25.0);
    glEnd();
}
```
Projection

• Mapping world to screen coordinates

```c
void reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
        gluOrtho2D(0.0, 30.0, 0.0, 30.0 * (GLfloat) h/(GLfloat) w);
    else
        gluOrtho2D(0.0, 30.0 * (GLfloat) w/(GLfloat) h, 0.0, 30.0);
    glMatrixMode(GL_MODELVIEW);
}
```

Orthographic Projection

• 2D and 3D versions
• glOrtho2D(left, right, bottom, top)
• In world coordinates!

Viewport

• Determines clipping in window coordinates
• glViewport(x, y, w, h)

Summary

1. A Graphics Pipeline
2. The OpenGL API
3. Primitives: vertices, lines, polygons
4. Attributes: color
5. Example: drawing a shaded triangle