

Outline

- Graphics APIs
- What is OpenGL?
- GLUT
- A test program
- Event Callbacks

Graphics APIs

- API stands for Application Program Interface.
- It is a collection of routines that the programmer can call, along with a model of how the routines work together to produce graphics.
- As a programmer you see only the interface. You are therefore shielded from the specifics of the underlying graphics hardware or software.

Scene Graph vs. Low Level API

- Scene graph API:
 - Implemented on top of a low level API.
 - Specify rendering hints to the hardware.
 - Controls the objects in the scene through a scene graph.
- Low level API:
 - Controls the 3D rendering directly.
 - Very efficient implementation, close to hardware.

Current Graphics APIs

- Scene graph APIs:
 - Java3D
 - VRML
 - OpenInventor
 - POV-Ray
- Low level APIs:
 - OpenGL
 - Direct3D (part of Microsoft's DirectX)

Java 3D

- High level object oriented 3D Graphics API.
- Rich repository of advanced features
 - Virtual reality, haptics, stereo, etc.
- Scene graph programming model.
- Layered implementation on top of Direct3D, OpenGL
 - Transparently uses graphics hardware
- Supports scene graph loaders (VRML)
- Can be obtained at:
 - java.sun.com/products/java-media/3D/

VRML

- Virtual Reality Modeling Language.
- Simple text language for describing 3D shapes and interaction.
- VRML 97/2.0 plug-ins for WWW browsers.
- Some modeling tools (world builders) available.
- Can be obtained at:
 - www.web3d.org/vrml/vrml.htm

Direct3D

- Part of the MS DirectX Media libraries
- Controlled by Microsoft
- Designed for real-time 3D graphics
 - Games and home entertainment
- C/C++ API
- Looks procedural
- Can be obtained from the DirectX SDK at:
 - www.microsoft.com/directx

OpenGL

- Window system independent.
- Operating system independent.
- Initially developed by Silicon Graphics Inc.
- OpenGL Architecture Review Board (ARB):
 - www.opengl.org
- Scales from PC to very high-end engines.
- Intuitive, procedural interface with C bindings.
- Version 1.2 commonly available on PCs.
 - OpenGL 2.0 specification just came out.

Outline

- Graphics APIs
- What is OpenGL?
- GLUT
- A test program
- Event Callbacks

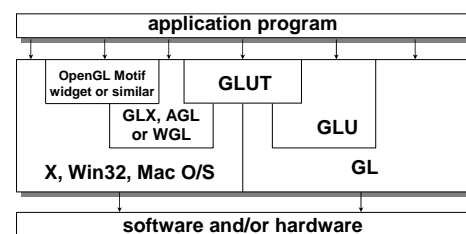
OpenGL as a Renderer

- Geometric primitives.
 - Points, lines and polygons.
- Image primitives.
 - Raster images and bitmaps.
- OpenGL calls either send input to the OpenGL machine or change its state.

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



Preliminaries

- Header Files (glut headers files in c:/glut-3.7/include)
- ```
#include <GL/glut.h> // must go first!
#include <GL/gl.h>
#include <GL/glu.h>
```
- Static Libraries (glut lib files in c:/glut-3.7)
  - Opengl32.lib
  - Glu32.lib
  - Glut32.lib
- Dynamic Link Libraries (C:\windows\system\ for Win98)
  - Opengl32.dll
  - Glu32.dll
  - Glut32.dll

## Preliminaries

- Add glut header and library folders to the project properties... See the handout.

## Outline

- Graphics APIs
- What is OpenGL?
- GLUT
- A test program
- Event Callbacks

## GLUT Basics

- GLUT handles all interactions with the windows system and user input devices (mouse, keyboard).
- General Application Structure:
  - Configure and open window.
  - Initialize OpenGL state.
  - Register input callback functions.
  - Enter event processing loop.

## A Simple Program

```
#include <GL/glut.h>

void display() {
 glClear(GL_COLOR_BUFFER_BIT);
 glBegin(GL_POLYGON);
 glVertex2f(-0.5,-0.5);
 glVertex2f(-0.5,0.5);
 glVertex2f(0.5,0.5);
 glVertex2f(0.5,-0.5);
 glEnd();
 glFlush();
}

int main(int argc, char** argv){
 glutInit(&argc, argv);
 glutCreateWindow("My First Window");
 glutDisplayFunc(display);
 glutMainLoop();
}
```

Simple

## What do we notice?

- Color
- Shape
- Position
- Try resizing the window
  - What happens?

### A Modification

```
void display() {
 glClear(GL_COLOR_BUFFER_BIT);
 glBegin(GL_POLYGON);
 glColor3f(1.0, 0.0, 0.0);
 glVertex2f(-0.5,-0.5);
 glVertex2f(-0.5,0.5);
 glVertex2f(0.5,0.5);
 glVertex2f(0.5,-0.5);
 glEnd();
 glFlush();
}
```

Simple

### What do we notice?

- Color?

### Another Modification

```
void display() {
 glClear(GL_COLOR_BUFFER_BIT);
 glBegin(GL_POLYGON);
 glColor3f(1.0, 0.0, 0.0);
 glVertex2f(-0.5,-0.5);
 glColor3f(1.0, 1.0, 1.0);
 glVertex2f(-0.5,0.5);
 glVertex2f(0.5,0.5);
 glVertex2f(0.5,-0.5);
 glEnd();
 glFlush();
}
```

Simple

### What do we notice?

- Color?
- State?
- Vertex position?

### Another Modification I

```
void display() {
 glClear(GL_COLOR_BUFFER_BIT);
 glBegin(GL_POLYGON);
 glColor3f(1.0, 0.0, 0.0); //red
 glVertex2f(-0.5,-0.5); //lower left
 glColor3f(0.0, 1.0, 0.0);
 glVertex2f(-0.5,0.5);
 glColor3f(1.0, 1.0, 1.0);
 glVertex2f(0.5,0.5);
 glVertex2f(0.5,-0.5);
 glEnd();
 glFlush();
}
```

Simple

### What do we notice?

- Color?
- Vertex position?

## Another Modification II

```
void display() {
 glClear(GL_COLOR_BUFFER_BIT);
 glBegin(GL_POLYGON);
 glColor3f(1.0, 0.0, 0.0); //red
 glVertex2f(-0.5,-0.5); //lower left
 glColor3f(0.0, 1.0, 0.0); //green
 glVertex2f(-0.5,0.5); //upper left
 glColor3f(0.0, 0.0, 1.0);
 glVertex2f(0.5,0.5);
 glColor3f(0.0, 0.0, 0.0);
 glVertex2f(0.5,-0.5);
 glEnd();
 glFlush();
}
```

Simple

## What do we notice?

- Color?
- Vertex position?

## Observations

```
void display() {
 glClear(GL_COLOR_BUFFER_BIT);
 glBegin(GL_POLYGON);
 glColor3f(1.0, 0.0, 0.0); //red
 glVertex2f(-0.5,-0.5); //lower left
 glColor3f(0.0, 1.0, 0.0); //green
 glVertex2f(-0.5,0.5); //upper left
 glColor3f(0.0, 0.0, 1.0); //blue
 glVertex2f(0.5,0.5); //upper right
 glColor3f(0.0, 0.0, 0.0); //black
 glVertex2f(0.5,-0.5); //lower right
 glEnd();
 glFlush();
}
```

## OpenGL's State Machine

- All rendering attributes are encapsulated in the OpenGL state.
  - Color
  - size
  - Rendering styles.
  - Shading and lighting.
  - Texture mapping, etc.

## OpenGL State Variables

- You can query the State Machine to determine the state of any variable.
- Any feature you enable or disable with `glEnable` / `glDisable` as well as numeric settings set with `glSet` can be queried with the many variations of `glGet`.
- Example: Setting the current drawing color.

```
GLubyte color[3];
glColor3ub(color[0],color[1],color[2]);
glColor3bv(color);
...
GLfloat colorf[3];
glGetFloatv(GL_CURRENT_COLOR,colorf);
```

## OpenGL Command Formats

`glVertex3fv( v )`

Number of components

2 - (x,y)  
3 - (x,y,z)  
4 - (x,y,z,w)

Data Type

b - byte  
ub - unsigned byte  
s - short  
us - unsigned short  
i - int  
ui - unsigned int  
f - float  
d - double

Vector

Pointer to array:  
`GLfloat point[3];`  
`glVertex3fv(point);`

Omit "v" for scalar form:

`glVertex2f( x, y )`

## Enumerated Types

- OpenGL defines numerous types for compatibility:

| OpenGL Data Type           | Internal Representation | Defined as C Type | C Literal Suffix |
|----------------------------|-------------------------|-------------------|------------------|
| GLbyte                     | 8-bit integer           | signed char       | b                |
| GLshort                    | 16-bit integer          | short             | s                |
| GLint, GLsizei             | 32-bit integer          | long              | l                |
| GLfloat, GLclampf          | 32-bit float            | float             | f                |
| GLdouble, GLclampd         | 64-bit float            | double            | d                |
| GLubyte, GLboolean         | 8-bit unsigned int      | unsigned char     | ub               |
| GLushort                   | 16-bit unsigned int     | unsigned short    | us               |
| GLuint, GLenum, GLbitfield | 32-bit unsigned int     | unsigned long     | ui               |

## GLUT Windows

- Create a top-level window with:
  - int id = glutCreateWindow(char \*name);
- Create sub-windows with:
  - glutCreateSubWindow(int id, int x, int y, int w, int h);
    - id: ID of the parent window.
    - x, y: Position relative to the parent window.
    - w, h: Width and height in pixels.
- Set the current window:
  - glutSetWindow( int id);
- Get the id of the current window:
  - int id = glutGetWindow(void);

## More Modifications

```
int main(int argc, char** argv){
 glutInit(&argc, argv);
 glutInitWindowSize(100, 200);
 glutInitWindowPosition(100, 100);
 glutCreateWindow("My First Window");
 glutDisplayFunc(display);
 glClearColor(1.0, 1.0, 1.0, 1.0);
 glutMainLoop();
}
```

Simple

## More Modifications

```
int main(int argc, char** argv){
 glutInit(&argc, argv);
 glutInitWindowSize(100, 200);
 glutInitWindowPosition(100, 100);
 glutCreateWindow("My First Window");
 glutDisplayFunc(display);
 glClearColor(1.0, 1.0, 1.0, 1.0);
 glutMainLoop();
}
```

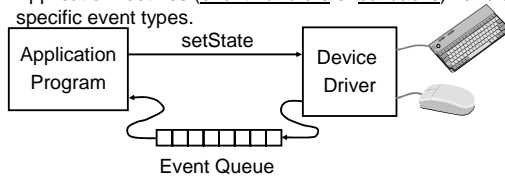
Try various values for Window size and position (including negative values)  
 Try various values for ClearColor  
 Try placing these function calls in different orders within the main loop

What do we notice?

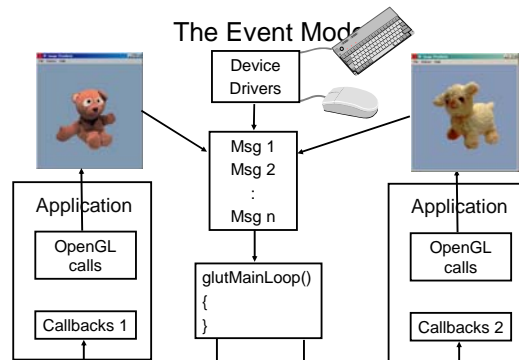
Simple

## The Event Model

- Events from the input hardware are queued into an event queue.
- Application routines (event handlers or callbacks) handle specific event types.



## The Event Model



## GLUT Callback Functions

- “Register” callbacks with GLUT:

```
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutIdleFunc(idle);
glutKeyboardFunc(keyboard);
glutMouseFunc(mouseButton);
glutMotionFunc(mouseMove);
```
- For subwindows, register the callbacks right after the call to `glutCreateSubwindow()`;

## Display Callback

- Do **all** of your drawing here:

```
glutDisplayFunc(display);

void display(void)
{
 glClear(GL_COLOR_BUFFER_BIT);
 glBegin(GL_TRIANGLE_STRIP);
 glVertex3fv(v[0]);
 glVertex3fv(v[1]);
 glVertex3fv(v[2]);
 glVertex3fv(v[3]);
 glEnd();
 glutSwapBuffers();
}
```

## Reshape Callback

- Called whenever the window is resized:

```
glutReshapeFunc(reshape);

void reshape(GLsizei w, GLsizei h)
{
 GW = w; // Set global width and height
 GH = h;
 glViewport(0, 0, w, h); // Set viewport
 glLoadIdentity();
 gluOrtho2D(-(float)w/h, (float)w/h, -1., 1.);
}
```