Installing GLUT

- In Unix machines, GLUT should already be “alive and present.”
- For Microsoft Windows, GLUT can be found at the site of OpenGL: www.opengl.org.
- However, it is also available on the course site.
- Copy glut32.dll into the C:\Windows\System directory.
- Copy the file glut32.lib into the directory where all other libraries are for the compiler.
- (usually C:\Program Files\Microsoft Visual Studio\VC98\lib).
- Copy glut.h in the directory where OpenGL header files are.
- (usually C:\Program Files\Microsoft Visual Studio\VC98\Include\GL).

Objectives

- Development of the OpenGL API
- OpenGL Architecture
  - OpenGL as a state machine
- Functions
  - Types
  - Formats
- Simple program
Early History of APIs

- IFIPS (1973) formed two committees to come up with a standard graphics API
  - Graphical Kernel System (GKS)
    - 2D but contained good workstation model
    - Core
    - Both 2D and 3D
    - GKS adopted as ISO and later ANSI standard (1980s)
- GKS not easily extended to 3D (GKS-3D)
- Far behind hardware development

SGI and GL

- Silicon Graphics (SGI) revolutionized the graphics workstation by implementing the pipeline in hardware (1982)
- To use the system, application programmers used a library called GL
- With GL, it was relatively simple to program three dimensional interactive applications

OpenGL

- The success of GL lead to OpenGL (1992), a platform-independent API that was
  - Easy to use
  - Close enough to the hardware to get excellent performance
  - Focus on rendering
  - Omitted windowing and input to avoid window system dependencies
OpenGL Evolution

• Controlled by an Architectural Review Board (ARB)
  - Members include SGI, Microsoft, Nvidia, HP, 3DLabs, IBM, ……
  - Relatively stable (present version 1.4)
    - Evolution reflects new hardware capabilities
      - 3D texture mapping and texture objects
      - Vertex programs
    - Allows for platform specific features through extensions

OpenGL Libraries

• OpenGL core library
  - OpenGL32 on Windows
  - GL on most unix/linux systems
• OpenGL Utility Library (GLU)
  - Provides functionality in OpenGL core but avoids having to rewrite code
• Links with window system
  - GLX for X window systems
  - AGL for Macintosh

GLUT

• OpenGL Utility Library (GLUT)
  - Provides functionality common to all window systems
    - Open a window
    - Get input from mouse and keyboard
    - Menus
    - Event-driven
  - Code is portable but GLUT lacks the functionality of a good toolkit for a specific platform
Software Organization

- API
  - application program
  - OpenGL Motif widget or similar
  - GLUT
  - GLU
  - GL
  - GLX, AGL or WGL
  - X, Win32, Mac O/S

OpenGL Architecture

- Immediate Mode
- Geometric pipeline
- CPU
- Display List
- Rasterization
- Per Fragment Operations
- Texture Memory
- Frame Buffer
- Pixel Operations
- Polynomial Evaluator
- Per Vertex Operations & Primitive Assembly

OpenGL Functions

- Primitives
  - Points
  - Line Segments
  - Polygons
- Attributes
- Transformations
  - Viewing
  - Modeling
- Control
- Input (GLUT)
OpenGL functions are of two types
- Primitive generating
  - Can cause output if primitive is visible
  - How vertices are processed and appearance of primitive are controlled by the state
- State changing
  - Transformation functions
  - Attribute functions

Lack of Object Orientation
OpenGL is not object oriented so that there are multiple functions for a given logical function, e.g. `glVertex3f`, `glVertex2i`, `glVertex3dv`,…..
Underlying storage mode is the same
Easy to create overloaded functions in C++ but issue is efficiency

OpenGL function format
`glVertex3f(x,y,z)`
function name
`belongsto GL library`
x,y,z are floats
`glVertex3fv(p)`
p is a pointer to an array
OpenGL #defines

- Most constants are defined in the include files gl.h, glu.h and glut.h
  - Note #include <glut.h> should automatically include the others
  - Examples
    - glBegin(GL_PLOYGON)
    - glClear(GL_COLOR_BUFFER_BIT)
- include files also define OpenGL data types: GLfloat, GLdouble, ....

A Simple Program

Generate a square on a solid background

```
#include <glut.h>

void myDisplay(){
    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){
    glutCreateWindow("simple");
    glutDisplayFunc(myDisplay);
    glutMainLoop();
}
```
Event Loop

- Note that the program defines a display callback function named `mydisplay`
  - Every glut program must have a display callback
  - The display callback is executed whenever OpenGL decides the display must be refreshed, for example when the window is opened
  - The `main` function ends with the program entering an event loop

Defaults

- `simple.c` is too simple
- Makes heavy use of state variable default values for
  - Viewing
  - Colors
  - Window parameters
- Next version will make the defaults more explicit

Compilation on Windows

- Visual C++
  - Get glut.h, glut32.lib and glut32.dll from web
  - Create a console application
  - Add opengl32.lib, glu32.lib, glut32.lib to project settings (under link tab)
- Borland C similar
Using GLUT with Microsoft Windows

Program Structure

- Most OpenGL programs have a similar structure that consists of the following functions
  - `main()`:
    - defines the callback functions
    - opens one or more windows with the required properties
    - enters event loop (last executable statement)
  - `init()` sets the state variables
    - viewing
    - Attributes
  - callbacks
    - Display function
    - Input and window functions

Simple.c revisited

- In this version, we will see the same output but have defined all the relevant state values through function calls with the default values
- In particular, we set
  - Colors
  - Viewing conditions
  - Window properties
main.c

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

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("simple");
    init();
    glutDisplayFunc(mydisplay);
    glutMainLoop();
}
```

includes gl.h

- includes gl.h
- defines window properties
- set OpenGL state
- enter event loop

init.c

```c
void init()
{
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
}
```

- black clear color
- opaque window
- fill with white
- viewing volume

GLUT functions

- `glutInit` allows application to get command line arguments and initializes system
- `glutInitDisplayMode` requests properties of the window (the rendering context)
  - RGB color
  - Single buffering
  - Properties logically ORed together
- `glutWindowSize` in pixels
- `glutWindowPosition` from top-left corner of display
- `glutCreateWindow` create window with title "simple"
- `glutDisplayFunc` display callback
- `glutMainLoop` enter infinite event loop
mydisplay.c

```c
void mydisplay()
{
    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();
}
```

OpenGL Primitives

Polygon Issues

- OpenGL will only display polygons correctly that are
  - Simple: edges cannot cross
  - Convex: All points on line segment between two points in a polygon are also in the polygon
  - Flat: all vertices are in the same plane
- User program must check if above true
- Triangles satisfy all conditions
Attributes

- Attributes are part of the OpenGL and determine the appearance of objects
  - Color (points, lines, polygons)
  - Size and width (points, lines)
  - Stipple pattern (lines, polygons)
  - Polygon mode
    - Display as filled: solid color or stipple pattern
    - Display edges

RGB color

- Each color component stored separately in the frame buffer
- Usually 8 bits per component in buffer
- Note in `glColor3f` the color values range from 0.0 (none) to 1.0 (all), while in `glColor3ub` the values range from 0 to 255

Indexed Color

- Colors are indices into tables of RGB values
- Requires less memory
  - indices usually 8 bits
  - not as important now
    - Memory inexpensive
    - Need more colors for shading
Color and State

- The color set by `glColor` becomes part of the state and will be used until changed.
  - Colors and other attributes are not part of the object but are assigned when the object is rendered.
- We can create conceptual *vertex colors* by code such as:
  ```
glColor
 glVertex
 glColor
 glVertex
  ```

Smooth Color

- Default is *smooth* shading.
  - OpenGL interpolates vertex colors across visible polygons.
- Alternative is *flat shading*.
  - Color of first vertex determines fill color.
- `glShadeModel(GL_SMOOTH)`
  - or `GL_FLAT`