Computer Graphics

Lecture3
OpenGL

Instructor: Dr. MAO Aihua

ahmao@scut.edu.cn

Announcement

- Visit discussion forum to keep up to date

http://www.pearsonhighered.com/educator/product/Computer-Graphics-with-Open-GL-4E/9780136053583.page#downlaoddiv

- Red Book is good source for Open
- http://www.opengl.org/
- http://www.opengl.org/resources/libraries/glut/
- *http://www.opengl.org/documentation/red_book_1.0/*

What is OpenGL?

- OpenGL is an open standard graphics toolkit
  - *Derived from SGI (Silicon Graphics Inc.) GL toolkit*
  - *A (low-level) Graphics rendering API*
  - *Window/operating system independent*
- Provides a range of functions for modeling, rendering and manipulating the frame buffer
- Why use it?
  - *Alternatives: Direct3D, Java3D - more complex and less well supported respectively*

OpenGL: Conventions

Functions in OpenGL start with `gl`

- Most functions just **`gl`** (e.g., **`glColor()`**)
- Functions starting with **`glu`** are utility functions (e.g., **`gluLookAt()`**)
- Functions starting with **`glx`** are for interfacing with the X Windows system (e.g., in gfx.c)

Constants: GL_2D, GL_RGB, …

Data types: GLbyte, GLfloat, …

OpenGL: Conventions

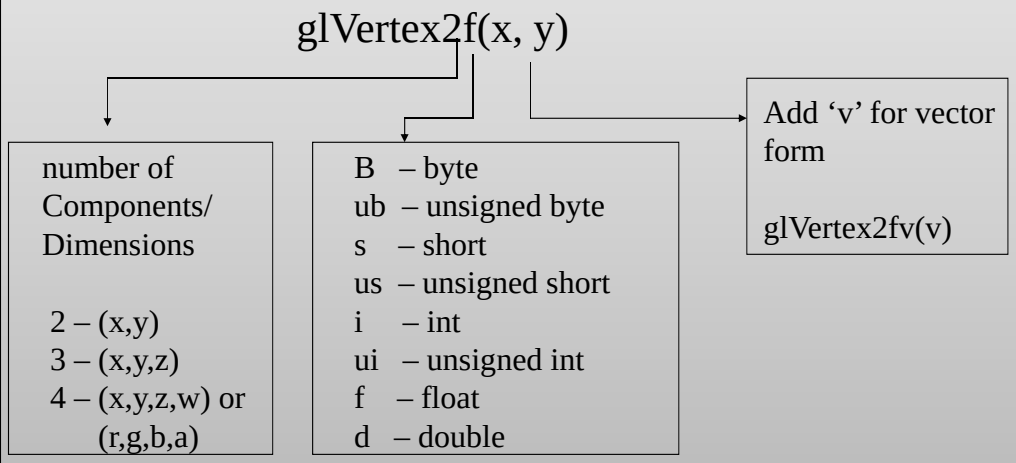
Function names indicate argument type and number

- Functions ending with **`f`** take floats
- Functions ending with **`i`** take ints
- Functions ending with **`b`** take bytes
- Functions ending with **`ub`** take unsigned bytes
- Functions that end with **`v`** take an array.

Examples

- **`glColor3f()`** takes 3 floats
- **`glColor4fv()`** takes an array of 4 floats

OpenGL: Conventions



OpenGL: Conventions

Variables written in CAPITAL letters

- Example: GLUT_SINGLE, GLUT_RGB
- usually constants
- use the bitwise or command (x | y) to combine constants

OpenGL: Conventions

OpenGL operates as an infinite loop

- Put things in the scene (points, colored lines, textured polys)
- Describe the camera (location, orientation, field of view)
- Listen for keyboard events
- Render – draw the scene

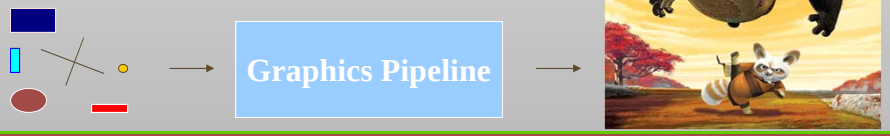
OpenGL: Conventions

Rendering

- Typically execution of OpenGL commands
- Converting geometric/mathematical object descriptions into frame buffer values

OpenGL can render:

- Geometric primitives
  - *Lines, points, polygons, etc…*
- Bitmaps and Images
  - *Images and geometry linked through texture mapping*



OpenGL: Conventions

OpenGL uses matrices

- Matrix describes camera type
- Matrix describes current configuration of the 3D space
  - *Explanation…*

OpenGL: Coordinate system

A metaphor for transformation

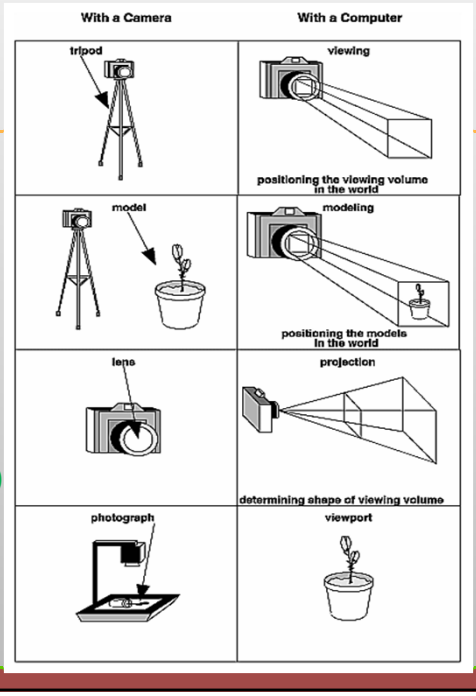
Coornidate system (discussion)

- the world coordinate

（longitude and latitude）

-the local coordinate

-the camera coordinate (OpenGL)



OpenGL: Coordinate system

OpenGL coordinate system

- right-handed （ cartesian coordinate system ）
  - *Hold out your right hand and hold your thumb, index, and middle fingers orthogonal to one another*
  - *Call your thumb the x-axis, index = y-axis, and middle = z-axis*
  - *This is the OpenGL coordinate system*
- The camera defaults to look down negative z-axis

OpenGL: Coordinate system

So…

- X-axis = thumb = 1, 0, 0
- Y-axis = index = 0, 1, 0
- Z-axis = middle = 0, 0, 1

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Camera defaults to look down negative z-axis
- Let's say we want it to look down x-axis

OpenGL: Coordinate system

Coordinate system transformation so camera looks down x-axis

- If x-axis → negative z-axis (A rotation of 90 degrees around the Y axis)
  - *x* → *-z*
  - *y* → *y*
  - *z* → *x*

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

OpenGL: Coordinate system

The $a \rightarrow i$ matrix defines the transformation

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \boldsymbol{a} & \boldsymbol{b} & \boldsymbol{c} \\ \boldsymbol{d} & \boldsymbol{e} & \boldsymbol{f} \\ \boldsymbol{g} & \boldsymbol{h} & \boldsymbol{i} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

Why store the transformation matrix and not the final desired matrix?

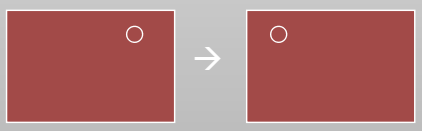
OpenGL: Coordinate system

The transformation will be applied to many points

- If the following transformation moves the axes

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \boldsymbol{a} & \boldsymbol{b} & \boldsymbol{c} \\ \boldsymbol{d} & \boldsymbol{e} & \boldsymbol{f} \\ \boldsymbol{g} & \boldsymbol{h} & \boldsymbol{i} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

- The same transformation moves vertices
  - *Example: (1, 1, -1) → (-1, 1, -1)*



$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{i} \\ \boldsymbol{j} \\ \boldsymbol{k} \end{bmatrix} = \begin{bmatrix} \boldsymbol{i}' \\ \boldsymbol{j}' \\ \boldsymbol{k}' \end{bmatrix}$$

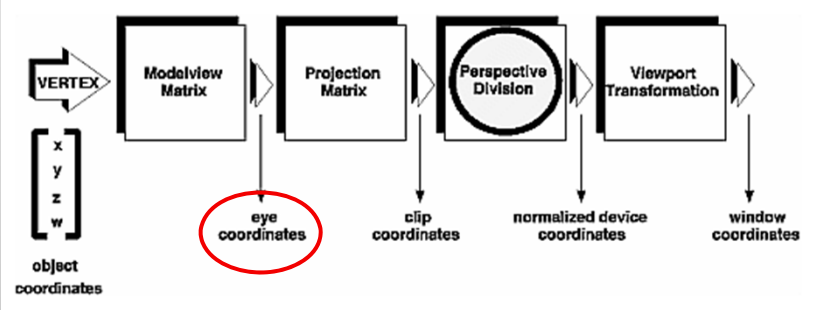
OpenGL: Coordinate system

- v' = M*v ⟶ [x,y,z,w]T

Transformation Matrix

Why we need transformation?

to transform the vertex from the real world to the computer window



OpenGL: Coordinate system

This important matrix is stored as the MODELVIEW matrix

Note OpenGL preserves a similar matrix to describe the camera type and this is called the PROJECTION_MATRIX

Manipulating Matrix Stacks

Observation: Certain model transformations are shared among many models

We want to avoid continuously reloading the same sequence of transformations

The MODELVIEW matrix is so important OpenGL maintains **a stack** of these matrices

`glPushMatrix ()`

- push all matrices in current stack down one level and copy topmost matrix of stack

`glPopMatrix ()`

- pop the top matrix off the stack

OpenGL: Setting up Camera

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(   eyeX, eyeY, eyeZ,
             lookX, lookY, lookZ,
             upX, upY, upZ);
```

- eye[XYZ]: camera position in world coordinates
- look[XYZ]: a point centered in camera's view
- up[XYZ]: a *vector* defining the camera's vertical

Creates a matrix that transforms points in world coordinates to camera coordinates

OpenGL: Modeling Transformations

Work under the MODELVIEW matrix

glTranslate (x, y, z)

- Post-multiplies the current matrix by a matrix that moves the object by the given x-, y-, and z-values

glRotate (theta, x, y, z)

- Post-multiplies the current matrix by a matrix that rotates the object in a counterclockwise direction about the ray from the origin through the point (x, y, z)

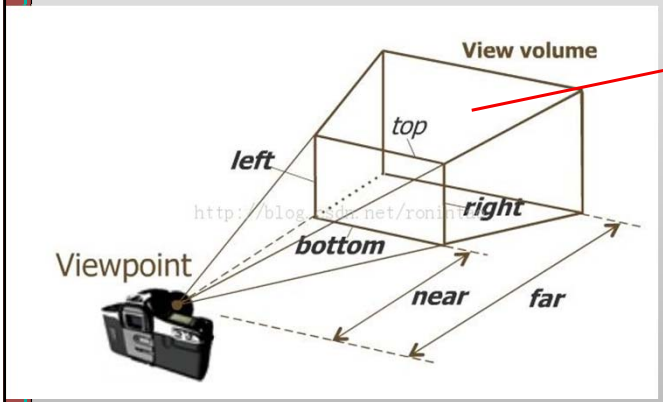
glScale (x, y, z)

- Post-multiplies the current matrix by a matrix that stretches, shrinks, or reflects an object along the axes

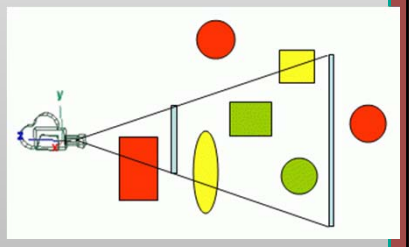
OpenGL: Perspective Projection

In OpenGL:

- Projections implemented by *projection matrix*



VIEW FRUSTUM (CONE)



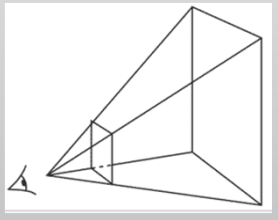
OpenGL: Perspective Projection

Set projection parameters (e.g., field of view)

Typically, we use a perspective projection

- Distant objects appear smaller than near objects
- Specifying a point at center of screen
- Defined by a *view frustum* (draw it)

Other projections: orthographic, isometric



OpenGL: Perspective Projection

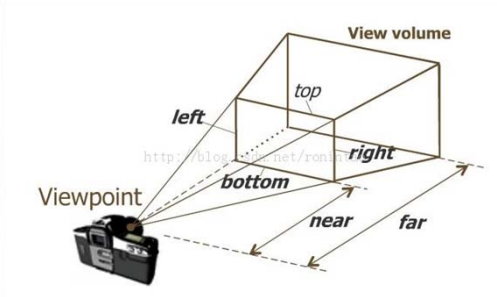
In OpenGL:

- Projections implemented by *projection matrix*
- `gluPerspective()` creates a perspective projection matrix:

```
glSetMatrix(GL_PROJECTION);
glLoadIdentity(); //load an identity matrix
gluPerspective(vfov, aspect, near, far);
```

OpenGL: Perspective Projection

* Parameters to `gluPerspective()`:
  `vfov`: vertical field of view
  `aspect`: window width/height
  `near`, `far`: distance to near & far clipping planes



OpenGL: Lighting

Setup lighting, if any

Simplest option: change the current color between polygons or vertices

- glColor() sets the current color

Or OpenGL provides a simple lighting model:

- Set parameters for light(s)
  - *Intensity, position, direction & falloff (if applicable)*
- Set *material* parameters to describe how light reflects from the surface

Won't go into details now; check the red book if interested

OpenGL: Front/Back Rendering

Each polygon has two sides, front and back

OpenGL can render the two sides differently

The ordering of vertices in the list determines which is the front side:

- When looking at the *front* side, the vertices go *counterclockwise*
  - *This is basically the right-hand rule*
  - *Note that this still holds after perspective projection*

OpenGL: Double Buffering

Avoids displaying partially rendered frame buffer

OpenGL generates one raster image while another raster image is displayed on monitor

glxSwapBuffers (Display *dpy, Window, w)

glutSwapBuffers (void)

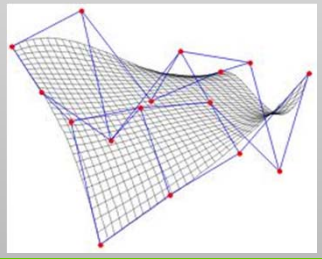
What OpenGL Can Do

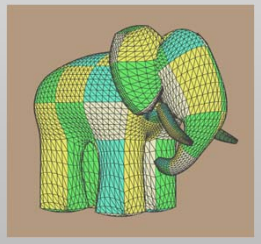
Draw geometrical graphics

Points, Lines, Polygons

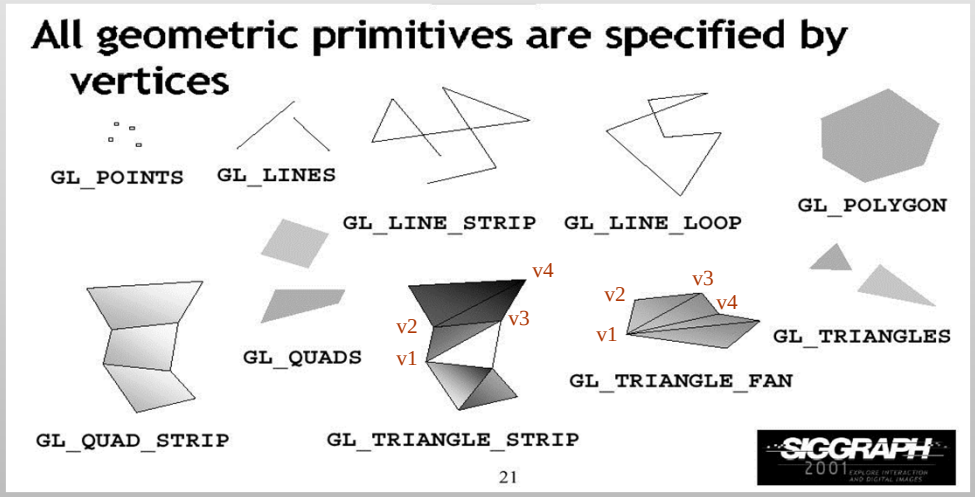
Polyhedral surfaces

Bezier surface、NURBS





Geometric Primitives



Points, Lines, Polygons

*glBegin(**mode**)* and *glEnd( )* delimit an object

mode can be one of the following:

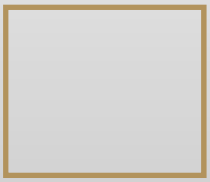
- GL_POINTS
- GL_LINES
- GL_POLYGON
- GL_LINE_STRIP
- GL_TRIANGLE_STRIP
- GL_TRIANGLES
- GL_QUADS
- GL_LINE_LOOP
- GL_QUAD_STRIP
- GL_TRIANGLE_FAN

Points

```
glBegin(GL_POINTS);
glVertex2i( 0, 0 );
glVertex2i( 0, 1 );
glVertex2i( 1, 0 );
glVertex2i( 1, 1);
glEnd( );
```

Line Loop (Polyline)

```
glBegin(GL_LINE_LOOP);
glVertex2i( 0, 0 );
glVertex2i( 0, 1 );
glVertex2i( 1, 1 );
glVertex2i( 1, 0 );
glEnd( );
```



Polygon Issues

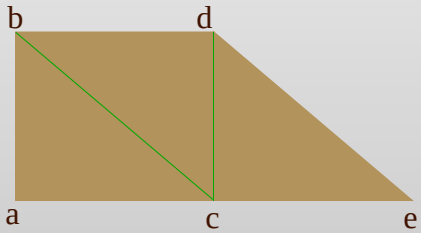
- OpenGL will only display polygons correctly that are
  - *Simple:* edges cannot cross
  - Convex: All points on line segment between two points in a polygon are also in the polygon
  - *Flat:* all vertices are in the same plane
- User program can check if above true
  - OpenGL will produce output if these conditions are violated but it may not be what is desired
- Triangles satisfy all conditions
- That's why we need triangulation algorithms!

Polygon

```
glBegin(GL_POLYGON);
glVertex2i( 0, 0 );
glVertex2i( 0, 1 );
glVertex2i( 1, 1 );
glVertex2i( 1, 0 );
glEnd( );
```

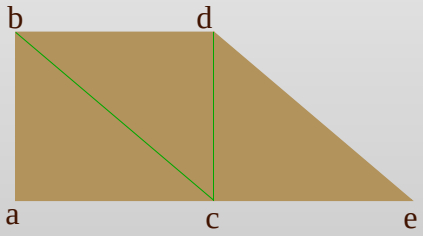
Triangles

```
glBegin(GL_TRIANGLES);
glVertex2i( 0, 0 ); // a
glVertex2i( 0, 1 ); // b
glVertex2i( 1, 0 ); // c

glVertex2i( 0, 1 ); // b
glVertex2i( 1, 0 ); // c
glVertex2i( 1, 1 ); // d

glVertex2i( 1, 0 ); // c
glVertex2i( 1, 1 ); // d
glVertex2i( 2, 0 ); // e    glEnd( );
```



Triangle Strip

```
glBegin(GL_TRIANGLE_STRIP);
glVertex2i( 0, 0 ); // a
glVertex2i( 0, 1 ); // b
glVertex2i( 1, 0 ); // c
glVertex2i( 1, 1 ); // d
glVertex2i( 2, 0 ); // e
glEnd( );
```



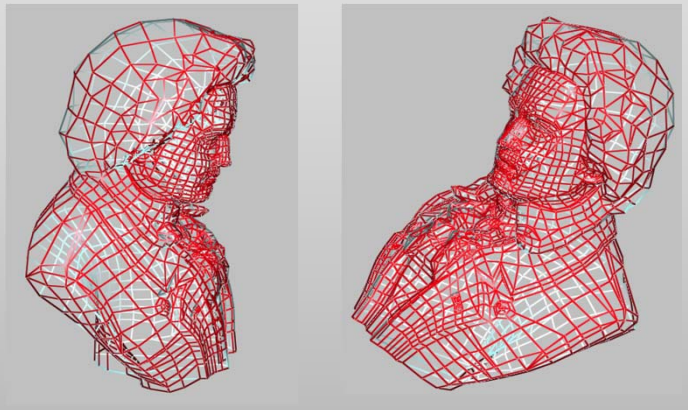
Attributes

- Point
  - Point size: ***glPointSize****(2.0);*
  - Point color: ***glColor3f*** *(0.0, 0.0, 1.0);*
- Line
  - Line width: ***glLineWidth****(2.0);*
  - Line color: ***glColor3f*** *(0.0, 0.0, 1.0);*
- Face
  - Front and/or back: ***GL_FRONT, GL_BACK, GL_FRONT_AND_BACK***
  - Face color: ***glColor3f*** *(0.0, 0.0, 1.0);*

What OpenGL Can Do

Geometrical Transformations

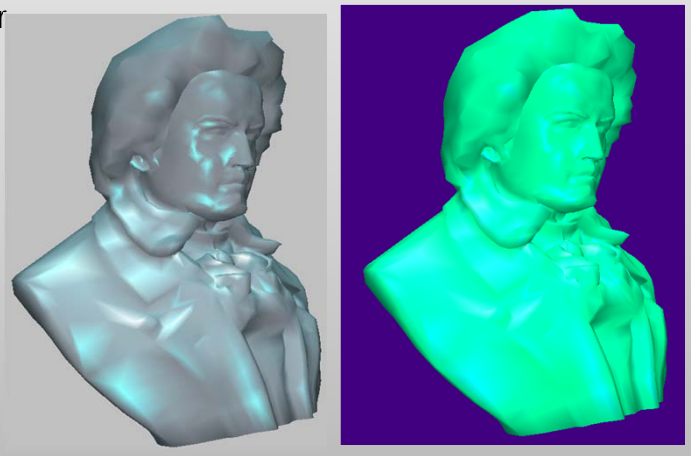
Translation/ Rotation/ Scaling



What OpenGL Can Do

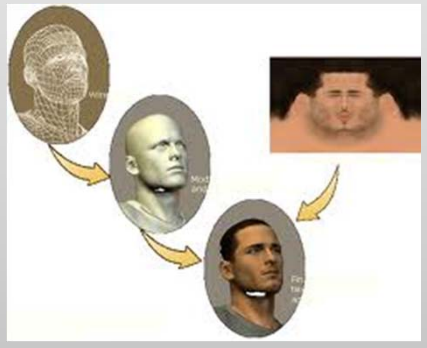
Color/ illumination

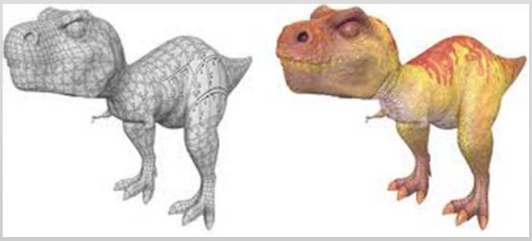
- Background color
- Light source
- Illumination model
- Material properties



What OpenGL Can Do

Texture mapping



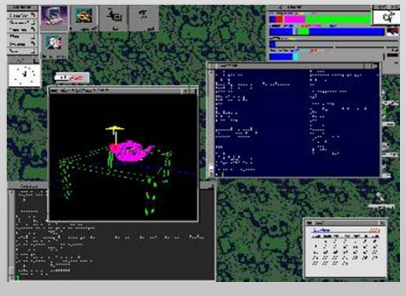


Window-based Programming

Most of the modern graphics systems are window-based

Non-window based environment

Window based environment



Window System Independent

OpenGL is window system independent

- No window management functions – create windows, resize windows, event handling, etc
- This is to ensure the application's portability
- Create some problems though – just a pure OpenGL program won't work anywhere.

More APIs are needed

X window system: GLX

Apple Macintosh: AGL

Microsoft Windows: WGL

These libraries provide complete functionality to create Graphics User Interface (GUI) such as sliders, buttons, , menus etc.

OpenGL and GLUT

GLUT (OpenGL Utility Toolkit)

- An auxiliary library
  - *A portable windowing API*
  - *Easier to show the output of your OpenGL application*
  - *We can use GLUT to interface with different window systems*
- Using OpenGL and GLUT can be ported to X windows, MS windows, and Macintosh with no effort

OpenGL and GLUT

GLUT (OpenGL Utility Toolkit)

- Handles:
  - *Window creation,*
  - *OS system calls*
    - Mouse buttons, movement, keyboard, etc…
  - *Callbacks*

GLUT Basics

Program Structure

1. Configure and open window (GLUT)
2. Initialize OpenGL (Optional)
3. Register input callback functions (GLUT)
   - Render
   - Resize
   - Input: keyboard, mouse, etc
4. Enter event processing loop (GLUT)

Sample Program

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

Void main(int argc, char** argv)
{
   int mode = GLUT_RGB|GLUT_SINGLE;
   glutInitDisplayMode(mode);
   glutInitWindowSize(500,500);
   glutCreateWindow(argv[0]);
   init();
   glutDisplayFunc(display);
   glutKeyboardFunc(key);
   glutMainLoop();
}
```

Sample Program

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

Void main(int argc, char** argv)
{
   int mode = GLUT_RGB|GLUT_SINGLE;
   glutInitDisplayMode(mode);
   glutInitWindowSize(500,500);
   glutCreateWindow("Simple");
   init();
   glutDisplayFunc(display);
   glutKeyboardFunc(key);
   glutMainLoop();
}
```

Specify the display Mode – RGB or color Index, single or double Buffer

Sample Program

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

Void main(int argc, char** argv)
{
   int mode = GLUT_RGB|GLUT_SINGLE;
   glutInitDisplayMode(mode);
   glutInitWindowSize(500,500);
   glutCreateWindow("Simple");
   init();
   glutDisplayFunc(display);
   glutKeyboardFunc(key);
   glutMainLoop();
}
```

Create a window Named "simple" with resolution 500 x 500

Sample Program

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

Void main(int argc, char** argv)
{
   int mode = GLUT_RGB|GLUT_SINGLE;
   glutInitDisplayMode(mode);
   glutInitWindowSize(500,500);
   glutCreateWindow("Simple");
   init();
   glutDisplayFunc(display);
   glutKeyboardFunc(key);
   glutMainLoop();
}
```

Your OpenGL initialization (Optional)

OpenGL Initialization

Set up whatever state you're going to use

- Don't need this much detail unless working in 3D

```
void init( void )
{  glClearColor (0.0, 0.0, 0.0, 0.0);
   glViewport(0, 0, width, height);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(-10, 10, -10, 10, -10, 20);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glEnable( GL_LIGHT0 );
   glEnable( GL_LIGHTING );
   glEnable( GL_DEPTH_TEST );}
```

Sample Program

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

Void main(int argc, char** argv)
{
   int mode = GLUT_RGB|GLUT_SINGLE;
   glutInitDisplayMode(mode);
   glutInitWindowSize(500,500);
   glutCreateWindow("Simple");
   init();
   glutDisplayFunc(display);
   glutKeyboardFunc(key);
   glutMainLoop();
}
```

← **Register your call back functions**

Callback functions?

Most of window-based programs are
event-driven

– which means do nothing until an event happens, and then execute some pre-defined functions

Events – key press, mouse button press and release, window resize, etc.

Your OpenGL program will be in infinite loop

GLUT Callback Functions

Callback function : Routine to call when an event happens

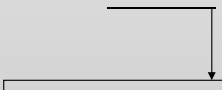
- Window resize or redraw
- User input (mouse, keyboard)
- Animation (render many frames)

"Register" callbacks with GLUT

- glutDisplayFunc(my_display_func);
- glutIdleFunc(my_idle_func);
- glutKeyboardFunc(my_key_events_func);
- glutMouseFunc (my_mouse_events_func);

glutDisplayFunc(void (*func)(void))

```
Void main(int argc, char** argv)
{
      …
   glutDisplayFunc(display);
   …
}
```



void display() – the function you provide. It contains all the OpenGL drawing function calls and will be called when pixels in the window need to be refreshed.

Rendering Callback

Callback function where all our drawing is done

Every GLUT program must have a display callback

glutDisplayFunc(***my_display_func***); /* this part is in main.c */

```
void my_display_func (void )
{
  glClear( GL_COLOR_BUFFER_BIT );
  glBegin( GL_TRIANGLE );
   glVertex3fv( v[0] );
   glVertex3fv( v[1] );
   glVertex3fv( v[2] );
  glEnd();
  glFlush();
}
```

And many more …

glutKeyboardFunc() – register the callback that will be called when a key is pressed

glutMouseFunc() – register the callback that will be called when a mouse button is pressed

glutMotionFunc() – register the callback that will be called when the mouse is in motion while a buton is pressed

glutIdleFunc() – register the callback that will be called when nothing is going on (no event)

Key Input Callbacks

Process user input

glutKeyboardFunc(my_key_events);

```
void my_key_events (char key, int x, int y )
{
  switch ( key ) {
    case ‘q’ : case ‘Q’ :
       exit ( EXIT_SUCCESS);
       break;
    case ‘r’ : case ‘R’ :
       rotate = GL_TRUE;
       break;
  }
}
```

Mouse Callback

Captures mouse press and release events

glutMouseFunc(my_mouse);

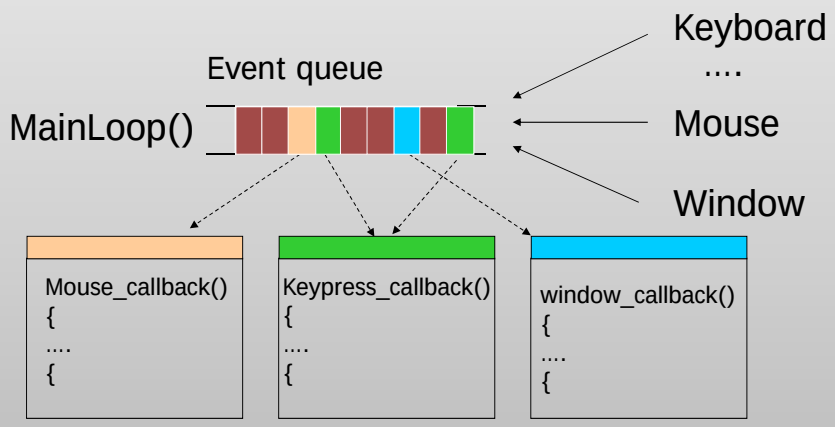
```
void myMouse(int button, int state, int x, int y)
{ if (button == GLUT_LEFT_BUTTON && state ==
  GLUT_DOWN)
    {
    …
    }
}
```

Sample Program

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

Void main(int argc, char** argv)
{
   int mode = GLUT_RGB|GLUT_SINGLE;
   glutInitDisplayMode(mode);
   glutInitWindowSize(500,500);
   glutCreateWindow("Simple");
   init();
   glutDisplayFunc(display);
   glutKeyboardFunc(key);
   glutMainLoop();
}
```

The program goes into a infinite loop waiting for events

Event Queue



GLUT Main Event Loop

glutMainLoop (void)

- Starts the GLUT even processing loop
- Never returns
- Calls registered function callbacks (user-defined event handlers) as appropriate
- Should be called at most once

How to install GLUT?

Download GLUT

- http://www.opengl.org/resources/libraries/glut.html

Copy the files to following folders:

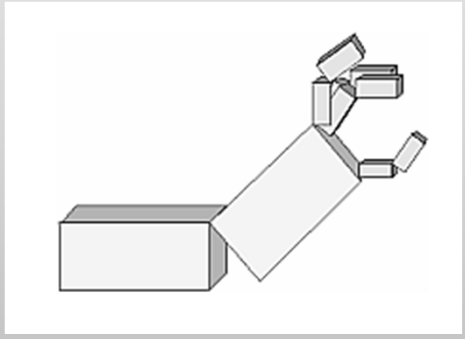
- glut.h → VC/include/gl/
- glut32.lib → VC/lib/
- glut32.dll → windows/system32/

Header Files:

- #include <GL/glut.h>
- #include <GL/gl.h>
- Include glut automatically includes other header files

Learn OpenGL by example

robot.c from the OpenGL Programming Guide



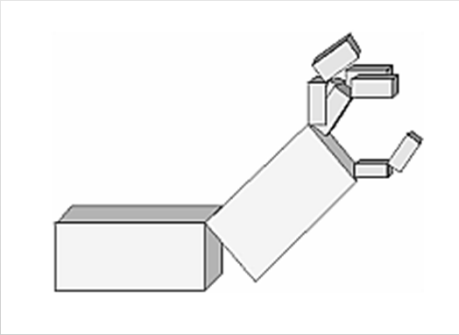
Learn OpenGL by example

Two bodies

- Upper arm
- Lower arm

Major tasks

- Position
- Orientation



Learn OpenGL by example

Headers

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

Learn OpenGL by example

```
void init(void) {
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glShadeModel (GL_FLAT);
}
```

Learn OpenGL by example

```
void display(void){
    glClear (GL_COLOR_BUFFER_BIT);
    glPushMatrix();
        glTranslatef (-1.0, 0.0, 0.0);
        glRotatef ((GLfloat) shoulder, 0.0, 0.0, 1.0);
        glTranslatef (1.0, 0.0, 0.0);
        glPushMatrix();
            glScalef (2.0, 0.4, 1.0);
            glutWireCube (1.0);
        glPopMatrix();
Continued…
```

Learn OpenGL by example

```
        glTranslatef (1.0, 0.0, 0.0);
        glRotatef ((GLfloat) elbow, 0.0, 0.0, 1.0);
        glTranslatef (1.0, 0.0, 0.0);
        glPushMatrix();
            glScalef (2.0, 0.4, 1.0);
            glutWireCube (1.0);
        glPopMatrix();
    glPopMatrix();
    glutSwapBuffers();
}
```

References

web：

- OpenGL official wbsite：http://www.opengl.org
- OpenGL Reference Mannual：http://www.opengl.org/sdk/docs/
- GLUT：http://www.opengl.org/resources/libraries/glut/
- Nate Robin: http://user.xmission.com/~nate/tutors.html
- NEHE：http://nehe.gamedev.net

specification ：

- The OpenGL Utility Toolkit (GLUT) Programming Interface (PDF/HTML) for Windows

book：

- OpenGL Programming Guide (seventh edition), Addison Wesley
- OpenGL SuperBible (third edition), Waite Group Press
- OpenGL Shading Language