Open GL
A Crash Course
Part 1
What is OpenGL?

- Hardware independent graphics API (but support 3-D hardware)
- Geometric primitives (e.g. 3-D polygons, transforms, materials, rather than pixels)
- Includes GLU (higher level definitions for transforms, object, etc.)
- Platform independent (e.g. no windowing or user interface support)
- Doesn’t generally include GLUT (windowing and user I/O)
- State machine (e.g. transforms, materials, lighting, shading model, line patterns, colors, normals, shading model)
Typical Program Structure:

Main:

- Initialize Windows/Buffers (GLUT)
- Setup event handlers
- Set some initial GL states
- Store display lists
- Loop
  - Check for events
  - If window event, adjust view port, redraw
  - If user I/O event, adjust something, redraw

Redraw:

- Clear screen
- Change some states
- Render some graphics
- Change some more states
- Render some more graphics
- ...and so on
Code Fragment:

```c
main() {
    //Init windows/buffers
    //Enter main loop
}
redraw() {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLORBUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glOrtho(0., 1., 0., 1., -1., 1.);
    glBegin(GL_POLYGON);
        glVertex3f(0.25, 0.25, 0.);
        glVertex3f(0.75, 0.25, 0.);
        glVertex3f(0.75, 0.75, 0.);
        glVertex3f(0.25, 0.75, 0.);
    glEnd();
    glFlush();
}
```
Primitive Types:

```c
GLfloat p1[3] = {0, 0, 1};
GLfloat p2[3] = {1, 0, 1};
GLfloat p3[3] = {1, 1, 1};
GLfloat p4[3] = {0, 1, 1};

glBegin(GL_TRIANGLES);
    glVertex3fv(p1);
    glVertex3fv(p2);
    glVertex3fv(p3);
    glVertex3fv(p4);
glEnd();
```
Random Notes About Primitives:

- Syntax: glVertex{234}{sifd}[v]()
- GL assumes polygons are convex
- GL assumes polygons are planar
- Subdivide non-planar polygons into triangles
- Specify polygons in CCW order!
Coordinate Systems and Matrices:

Modeling Transform
- glTranslatef()
- glRotatef()
- glScalef()

Viewing Transform
- gluLookAt()

Projection Transform
- glFrustum()
- gluPerspective()
- glOrtho()

Viewport Transform
- glViewport()

Utilities
- glMatrixMode()
- glLoadIdentity()
- glMultMatrix()
- glPushMatrix()
- glPopMatrix()
Typical Matrix Use:

Projection:

- Initialize at start up
- Adjust it when the window is resized

Modelview Matrix:

Viewing:

- Projection matrix has fixed viewpoint (0,0,0)
- Initialize at startup to produce viewpoint
- Modify it when the viewpoint must change

Modeling:

- Push the matrix onto the stack
- Apply transforms to orient for object placement
- Draw an object
- Pop the matrix
- Repeat…

In general, leave the Modelview matrix as you found it after drawing something.
Some State Management:

Basic Global States:

```c
 glEnable(), glDisable()
```

such as `GL_BLEND, GL_DEPTH_TEST, GL_FOG, GL_LIGHTING, GL_LINE_STIPPLE, GL_CULL_FACE` to enable the properties defined using the following types of commands.

Buffer Clearing:

```c
 glClearColor(0.0, 0.0, 0.0)
 glClearDepth(1.0)
```

for calls to `glClear(GL_COLORBUFFER_BIT | GL_DEPTHBUFFER_BIT)`

Color:

```c
 glColor3f(1.0, 1.0, 1.0)
 glColor3fv(color_array)
 glShadeModel(GL_SMOOTH)
 glShadeModel(GL_FLAT)
```
Color (cont’d):

Can be defined per vertex. GL_SMOOTH will interpolate across polygon, but GL_FLAT picks one vertex and uses it’s properties for whole polygon.

```c
glBegin(GL_TRIANGLE);
  glColor3f(1.0, 1.0, 1.0)
  glVertex3fv(p1);
  glColor3f(0.0, 0.0, 0.0)
  glVertex3fv(p2);
  glVertex3fv(p3);
 glEnd();
```

Point, Line and Polygon Details:

```c
glPointSize(1.0)

LineWidth(1.0)

glPolygonMode(GL_FRONT, GL_FILL)

glPolygonMode(GL_BACK, GL_LINE)

glFrontFace(GL_CCW)

glCullFace(GL_BACK)
```

Normals:

```c
glNormal3f(1.0, 1.0, 1.0)
```

note: will be the normal for all vertices that follow it.
Lighting:

- Don’t use glColor3f()!!
- Let OpenGL calculate how each light strikes the surface and shades the primitive with the appropriate color
- Define object vertices with normals and material properties
- Define light sources that have color, position, direction, and some distribution

Three types of lights:

- Point source (light bulb)
- Directional light (sun at earth’s surface)
- Spot light (flashlight)
Three physical models (per type of light):

**Ambient:**
- Light so scattered that it appears to come from all directions
- Illuminates objects without regard to their orientation
- Bounces off surfaces in all directions

**Diffuse:**
- Light which comes from one direction
- Illuminates surfaces most strongly when perpendicular
- Bounces off surface in all directions

**Specular:**
- Light which comes from one direction
- Bounces off in one direction
- Effect depends on view point
Defining material properties:

- Can be defined per vertex
- State variable (like color)
- Different color for each physical model

Example:

```c
glMaterialfv(GL_FRONT, GL_AMBIENT, &ambient[0])
glMaterialfv(GL_FRONT, GL_DIFFUSE, &diffuse[0])
glMaterialfv(GL_FRONT, GL_SPECULAR, &specular[0])
glMaterialf(GL_FRONT, GL_SHININESS, shininess)  
glMaterialfv(GL_FRONT, GL_EMISSION, &emission[0])
```

Defining lighting model:

- Set the global ambient level
- Specify which side(s) of polygons to light
- Decide if viewpoint is used in specular lighting calculations

Example:

```c
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE)  
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_FALSE)  
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, &ambient[0])
```
Defining lights (common parameters):

- OpenGL guarantees at least 8 lights
- Different color for each physical model

```c
glLightfv(GL_LIGHT0, GL_AMBIENT, &ambient[0]);
glLightfv(GL_LIGHT0, GL_DIFFUSE, &diffuse[0]);
glLightfv(GL_LIGHT0, GL_SPECULAR, &specular[0]);
glLightfv(GL_LIGHT0, GL_POSITION, x,y,z,w );
```

- **GL_POSITION** values define if a light is directional or positional (point source or spot light).
  - If (x,y,z,w) has w=0 the light is directional, and (x,y,z) is its direction
  - But if w=1 the light is positional and (x,y,z) is its position

Positional Lights (Point Source):

- **SPOT_CUTOFF** is the half angle of the spot light’s cone. If it’s 180, then we have a point source since we radiate all directions.

```c
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 180.0);
```
Positional Lights (Spot Light):

- If it’s not 180, then an angle from 0-90 defines the cone of the spot light.

\[
\text{glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 30.0);}
\]

- These remaining properties control the direction, and fading from the center axis.

\[
\begin{align*}
\text{glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, &dir[0]);} \\
\text{glLightfv(GL_LIGHT0, GL_SPOT_EXPONENT, 1.0);}
\end{align*}
\]

Attenuation:

- Spot and point lights can be defined to attenuation with distance from their position.
- The factor is \(1/( k_c + k_d + k_q d^2 )\)
- It has no meaning for directional lights.

\[
\begin{align*}
\text{glLightfv(GL_LIGHT0, GL_CONSTANT_ATTENUATION, Kc);} \\
\text{glLightfv(GL_LIGHT0, GL_LINEAR_ATTENUATION, K1);} \\
\text{glLightfv(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, Kq);}
\end{align*}
\]

Enabling:

- Lighting must be turned on.

\[
\begin{align*}
\text{glEnable(GL_LIGHTING);} \\
\text{glEnable(GL_LIGHT0);} \\
\end{align*}
\]
Lighting Equation:

\[ M - \text{Material Color} \]
\[ L - \text{Light Color} \]

\[ \text{Vertex Color} = M_{\text{Emissive}} + M_{\text{Ambient}} \times L_{\text{Global Ambient}} + \sum \text{Attenuation} \times \text{Spot Light Effect} \times [M_{\text{Ambient}} \times L_{\text{Ambient}} + M_{\text{Diffuse}} \times L_{\text{Diffuse}} \times \text{Angle Effect} + M_{\text{Specular}} \times L_{\text{Specular}} \times \text{Viewer Effect}] \]
#include <math.h>  
#include <stdlib.h>  
#include <stdio.h>  
#include <GL/glut.h>  

#ifndef min  
#define min(a,b) (((a) <= (b)) ? (a) : (b))  
#endif  

#define KEY_SPACE ' '  
#define KEY_ESCAPE '\033'  
#define KEY_a 'a'  
#define KEY_z 'z'  
#define KEY_A 'A'  
#define KEY_Z 'Z'  

#define KEY_UPARROW 101  
#define KEY_DOWNARROW 103  
#define KEY_LEFTARROW 100  
#define KEY_RIGHTARROW 102  
#define KEY_PGUP 104  
#define KEY_PGDN 105  

#define MOVE_DIST 5.0  
#define FOV 60.0  
#define WIN_SIZE 500  

//Far and near clip plane distances  
#define FAR_Z 900.0  
#define NEAR_Z 1.0  

//Position and direction of the scene's spot light  
#define L_P_X -50.0  
#define L_P_Y 50.0  
#define L_P_Z 50.0  

#define L_D_X 80.0  
#define L_D_Y -60.0  
#define L_D_Z -80.0  

GLuint listLight;                         //Handle to light's display list  
GLuint listCube;                         //Handle to cube's display list  

//Position of observer  
GLfloat positionX=0.0, positionY=0.0, positionZ=100.0;  

void Init(void);                       //Location of glut init calls, etc.  

void Keyboard(unsigned char,int,int);  //Normal key callback  
void SpecialKeys(int,int,int);         //Special key callback  
void Motion(int,int,int);               //Mouse motion callback  
void DepressedMotion(int,int,int);     //Mouse motion with button depressed  
void Mouse(int,int,int,int);            //Mouse up or down button action  
void MainMenu(int);                    //Main menu callback  
void Idle(void);                       //Idle callback  
void Draw(void);                       //Scene draw callback  

void SetupLights(void);                    //Generate light display list  
void SetupCube(void);                     //Generate cube display list
void Reshape(int, int);                //Window resize callback
void Visible(int);                    //Window visibility callback
void DrawModeMenu(int);                //DrawMode menu callback

int main( int argc, char *argv[] )
{
    glutInit( &argc, argv );            //Let glut eat any command line options (i.e. none)
    glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize(WIN_SIZE, WIN_SIZE); //Provide the window's width and height
    glutInitWindowPosition(0,0);        //Place it in the upper left corner

    glutCreateWindow( "COS426: Assignment 2" );

    Init();                              //We use this to initialize our display lists, etc.
    glutMainLoop();                      //Pass event handling over to Glut.

    return 0;
}

void Init( void )
{
    int submenu1;
    GLdouble aspect;

    glutDisplayFunc( Draw );            //Called when window is redrawn
    glutReshapeFunc( Reshape );         //Called when window is resized by the user
    glutKeyboardFunc( Keyboard );       //Called a alphanumeric or symbol is pressed
    glutMotionFunc( DepressedMotion );  //Called when special keys (such as arrows) are pressed
    glutPassiveMotionFunc( Motion );    //Called when mouse moves (regardless of button status)
    glutIdleFunc( Idle );               //Called when the visible state of the window changes

    //Glut can provide a popup menu via a mouse button.
    //glutCreateMenu creates menus, glutAddMenuEntry add items to
    //menus, and glutAddSubMenu makes a menu into a submenu entry
    //of a higher level menu.

    //First we create the submenu, returning a identifier handle
    //for this menu. glutCreateMenu takes as an argument the
    //callback function to use when an entry of the menu is
    //selected.
    submenu1 = glutCreateMenu( DrawModeMenu );

    //glutAddMenuEntry defines entries to the latest created menu.
    //The first arg is the entry label, and the second is an
    //integer passed to the menu's callback function
    glutAddMenuEntry( " Filled ", GL_FILL );
    glutAddMenuEntry( " Outline ", GL_LINE );
glutCreateMenu( MainMenu ); //Now we create the top level menu and pass in the callback function pointer
//glutAddSubMenu add the previously defined submenu
//with arguments being the submenu label and identifier
//We then add an entry as before
//The function is bound to the right mouse button.

glAddSubMenu(" Polygon mode ", submenu1 );

//Enable testing of the z buffer to properly render
// occlusions
glClearDepth(FAR_Z); //Value used when clearing z buffer. We use far clip plane.

glClearColor( 0.f, 0.f, 0.f, 0.f ); //Color to use when clearing frame buffer. We use black.

//Set up the projection matrix
//For simplicity, we use the gluPerspective function to
// set the projection matrix. The arguments and the
// field of view in degree, aspect ratio, and the
// distance of the near and far clipping planes

glMatrixMode( GL_PROJECTION ); //Enter project matrix mode
glLoadIdentity(); //Reset to the identity matrix
aspect = (GLdouble) WIN_SIZE/ (GLdouble) WIN_SIZE;

//Now switch to modelview matrix mode and initialize the
// state. Our projection acts as if we view the world
// from position (0,0,0), looking in the -z direction.
// So, to place ourselves at position (x,y,z), we move the
// whole world by a translation of (-x,-y,-z)

glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
glTranslatef(-positionX,-positionY,-positionZ);

//To greatly enhance performance at drawing time, we can
// precompile our calls to GL into display lists. Then,
// at draw time, we call GL to display the lists. By
// precompiling our static objects into such lists, the
// various GL calls to create the object are pre-stored
// in GL's efficient internal representation.

SetupLights(); //Store the scene's light into a list
SetupCube(); //Store the cube into a list

}
case KEY_a:
    break;

case KEY_Z:
case KEY_z:
    break;
}
}  

////////////////////////////////////////////////////////////////////
// This is the special keyboard event call back. Use it to implement
// the control required to navigate/operate our scene.
//
// Provided is a rudimentary movement system. You will modify this.
//
////////////////////////////////////////////////////////////////////

void SpecialKeys( int key, int x, int y )
{
    switch( key ){
    case KEY_UPARROW:
        positionZ-=MOVE_DIST;
        break;
    case KEY_DOWNARROW:
        positionZ+=MOVE_DIST;
        break;
    case KEY_LEFTARROW:
        positionX-=MOVE_DIST;
        break;
    case KEY_RIGHTARROW:
        positionX+=MOVE_DIST;
        break;
    case KEY_PGUP:
        positionY+=MOVE_DIST;
        break;
    case KEY_PGDN:
        positionY-=MOVE_DIST;
        break;
    }
    //Since we have moved the viewpoint, we reset the
    // modelview matrix to the new position
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    glTranslatef(-positionX,-positionY,-positionZ);
    //Then we force glut to invoke our drawing callback
    glutPostRedisplay();
}

////////////////////////////////////////////////////////////////////
// If a mouse button is currently depressed, and the mouse moves
// then this callback is invoked with the current position of the mouse
////////////////////////////////////////////////////////////////////

void DepressedMotion( int x, int y )
{ }
void Motion( int x, int y) {
}

void Mouse( int button, int state, int x, int y )
{
    switch( button ){
    case GLUT_LEFT_BUTTON:
        if( state == GLUT_DOWN ){
        }
        else if( state == GLUT_UP ){
        }
        break;
    case GLUT_MIDDLE_BUTTON:
        if( state == GLUT_DOWN ){
        }
        else if( state == GLUT_UP ){
        }
        break;
    }
    //We could put the right button here, but its already bound to the menu
}

void DrawModeMenu( int entry )
{
    glPolygonMode( GL_FRONT, entry );
glPolygonMode( GL_BACK, entry );
glutPostRedisplay();
}
/ This callback is invoked whenever an entry of the main menu is selected. entry will be the integer which we attached to the chosen menu entry in the Init() function

void MainMenu( int entry )
{
    switch( entry ){
        case KEY_ESCAPE:
            exit( 0 );
            break;
    }
}

// When glut is not busy doing anything, this call back is invoked.
// This might be a good location to place code which modifies your objects so to achieve animation.

void Idle( void )
{
    //Insert code to dynamically modify scene
    glutPostRedisplay();  //Redraw
}

// This function, called within Init(), specifies a light into a display list. During Draw() we execute the list to place the light into the scene.

void SetupLights(void){
    //First setup the light's properties
    GLfloat lm_ambient[] = {0.5, 0.5, 0.5, 1.0};
    GLfloat l0_ambient[] = {0.2, 0.2, 0.2, 1.0};
    GLfloat l0_diffuse[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat l0_specular[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat l0_position[] = {L_P_X, L_P_Y, L_P_Z, 1.0};
    GLfloat l0_spotdir[] = {L_D_X, L_D_Y, L_D_Z};

    listLight=glGenLists(1);         //glGenList(n) requests integer handle to empty display list. If n>1, then the returned handle is the integer of the first of n contiguous empty lists.
    glGenList(listLight,GL_COMPILE);    //glNewList(n,x) signifies that the following GL calls be compiled into list n. GL_COMPILE tells GL to store the //list. Alternatively, we could simultaneously store and execute (draw) the list.

    glLightModelli(GL_LIGHT_MODEL_LOCAL_VIEWER,GL_TRUE);
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT,lm_ambient);
    glLightModelli(GL_LIGHT_MODEL_TWO_SIDE,GL_FALSE);
    glLightfv(GL_LIGHT0, GL_AMBIENT, l0_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, l0_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, l0_specular);
glLightfv(GL_LIGHT0, GL_POSITION, l0_position);
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, l0_spotdir);

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);

gEndList(); //glEndList stop compiling to the current list
}  

 void SetupCube(void) {
    GLfloat mb_ambient[] = { 0.4, 0.4, 0.4, 1.0 };
    GLfloat mb_diffuse[] = { 0.6, 0.6, 0.6, 1.0 };
    GLfloat mb_specular[] = { 0.3, 0.3, 0.3, 1.0 };
    GLfloat mb_shininess = 40.0;
    GLfloat mb_emission[] = { 0.0, 0.0, 0.0, 1.0 };
    
    listCube = glGenLists(1);  //Request handle to empty list
    glNewList(listCube, GL_COMPILE);  //Begin compiling the list
    glMaterialfv(GL_FRONT, GL_AMBIENT, mb_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mb_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mb_specular);
    glMaterialf(GL_FRONT, GL_SHININESS, mb_shininess);
    glMaterialfv(GL_FRONT, GL_EMISSION, mb_emission);
    glutSolidCube(10.0);
    glEndList();  //End the list
}

 void Draw( void ) {
    //First we empty out the off screen buffer
    glClear( GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT );
    glCallList(listLight);  //Then execute the light display list
    
    glPushMatrix();
    glTranslatef(-5,10,0);
    glRotatef( -35, 0, 1, 1 );
    glCallList(listCube);
    glPopMatrix();

glPushMatrix(); // Note that we can call the cube list
    glTranslatef(5, -5, -5); // more than once to replicate it.
    glRotatef(-55, 1, 1, 0);
    glCallList(listCube);
    glPopMatrix();
    glutSwapBuffers(); // Exchange front and back buffers
}

////////////////////////////////////////////////////////////////////
// If the user resizes the window, the perspective transform maybe
// inappropriate to the new aspect ration. To repair the distortion
// we recalculate the projection matrix.
//
// You shouldn't have to play with this...
////////////////////////////////////////////////////////////////////

void Reshape( int width, int height )
{
    GLdouble aspect; // The user just changed the window size
    glViewport( 0, 0, width, height ); // We must respecify the projection to avoid distortion
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    aspect = (GLdouble) width / (GLdouble) height;
    gluPerspective(FOV, aspect, NEAR_Z, FAR_Z);
    glMatrixMode( GL_MODELVIEW ); // Now return to modelview mode.
}

////////////////////////////////////////////////////////////////////
// If the visibility of the window changes (e.g. window is minimized
// or un-minimized), then adjust the idle callback to avoid
// drawing to and invisible viewport.
//
// You shouldn't have to play with this...
////////////////////////////////////////////////////////////////////

void Visible( int state )
{
    switch( state ){
    case GLUT_VISIBLE:
        glutIdleFunc( Idle );
        break;
    case GLUT_NOT_VISIBLE:
        glutIdleFunc( 0 );
        break;
    }