2. OpenGL - I

- We have used the term OpenGL many times. But what is OpenGL?

- OpenGL is a software interface to graphics hardware

- It consists of 150 distinct commands
Advantages:

- Built on top of X Windows so it inherits many good features of X Windows such as:
  - Device-independence, API to graphics hardware
  - Event driven (when the user presses a particular button of the mouse, say the left button, the event (left button pressed) and the measure (location of the cursor) are put into a queue)

- High end
  - You don’t have to write your code for most of the applications (OpenGL can do most of them for you)

- 3D-oriented
  - Developed for 3D applications, 2D is a special case of 3D (in what sense? e.g., z=0)
Things OpenGL can do:

- **Wireframe models** *(2D & 3D wireframe drawings)*
Things OpenGL can do:

- Depth-cuing effect
  - lines farther from the eye are dimmer; when we do scan conversion of a line, the intensity of a point considers the effect of depth
Things OpenGL can do:

- Anti-aliased lines
  - the intensity of a point is proportional to the areas of the pixel covered by the line polygon)
Things OpenGL can do:

- Flat-shaded vs smooth-shaded polygons
Things OpenGL can do: (conti)

- Shadows and textures (2D or 3D)
Things OpenGL can do: (conti)

- **Motion-blurred objects**
  - OpenGL has a special buffer called the **accumulation buffer**, it is used to compose the sequence of images needed to blur the moving object) (**double buffering**, good for animation)
For instance:

```c
void display_image(int i)
{
    glClear(GL_COLOR_BUFFER_BIT);  // Clear color buffer
    glClear(GL_DEPTH_BUFFER_BIT);  // Clear depth buffer
    display_image(i);
    glAccum(GL_ACCUM, 1.0/(float)num_images);
}

void main()
{
    glAccum(GL_RETURN, 1.0);
}
```
Things OpenGL can do: (conti)

- **Atmospheric effect** (fog)
  - to simulate a smoke-filled room
- **Depth-of-the-field effect**
  - Objects drawn with jittered viewing volumes into the accumulation buffer for a depth-of-the-field effect
A

add or multiply values

B

copy

Frame buffer

Accumulation buffer

Such as convolving contents of the frame buffer with a filter matrix $H$

$$b_{ij} = \sum_{k=-m}^{m} \sum_{l=-n}^{n} h_{kl} a_{i-k,j-l}$$

with e.g.

$$H = \frac{1}{5} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

or

$$H = \frac{1}{16} \begin{bmatrix} 1 & 2 & 4 & 2 \\ 2 & 4 & 2 & 2 \\ 1 & 2 & 1 & 1 \end{bmatrix}$$
### 2.2 Basic Structure of OpenGL Programs

<table>
<thead>
<tr>
<th>Initialization Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Callback Functions</strong></td>
</tr>
</tbody>
</table>

```c
void main() {
  Windows and coordinated system creation
  State Initialization
  Callback functions registration
  Infinite Event Handling Loop
}
```
Basic Structure of OpenGL Programs

- Callback functions (event handlers)
- Initialization
  - window and coordinate system creation
  - state initialization
- Registration
  - Let the system know which event handler should be invoked when an event occurs
- Infinite Event Handling Loop
  - Entering (infinite) event handling loop
Classical (X Windows based) event handling approach:

```c
void main ( ) {
    ....
    while ( 1 ) {
        XNextEvent ( display, &event );
        switch ( event.type ) {
            case  KeyPress:
                { event handler for a keyboard event } break;
            case  ButtonPress:
                { event handler for a mouse event } break;
            case  Expose:
                { event handler for an expose event } break;
                ....
        }
    }
}
```
Classical (X Windows based) event handling approach

- Event queue is maintained by the X Windows
- But handling of the events is your job
  - A statement like “case KeyPress” is like a callback function registration
- The entire structure now is replaced with one instruction:

```c
glutMainLoop()
```

Application program
#include <X11/Xlib.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>

void myInit (void) {
    glClearColor (1.0, 1.0, 1.0, 0.0);            // set background color
    glColor3f (0.0f, 0.0f, 0.0f);                     // set the drawing color
    glPointSize (4.0);                                   // set dot size 4 x 4
    glMatrixMode (GL_PROJECTION);      // set "camera shape"
    glLoadIdentity ();                                   // clear the matrix
    gluOrtho2D (0.0, 640.0, 0.0, 480.0);            // set the World Window
}
void myDisplay (void) {
    glClear (GL_COLOR_BUFFER_BIT);     // clear the screen
    glBegin(GL_POINTS);
    glVertex2i (100, 50);                             // draw three points
    glVertex2i (100, 130);
    glVertex2i (150, 130);
    glEnd();
    glFlush ( );                         // send all out to display
}

void myMouse (int button, int state, int x, int y) {
    switch (button) {
    case GLUT_RIGHT_BUTTON:
        if (state == GLUT_DOWN) exit (-1);
        break;
    default:
        break;
    }
}
2.3 An OpenGL Example (conti)

```c
int main (int argc, char** argv) {
    glutInit(&argc, argv);  // Initialize the toolkit
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);  // Set display mode
    glutInitWindowSize (640, 480);  // Set screen window size
    glutInitWindowPosition (100, 150);  // Set window position on screen
    glutCreateWindow (argv[0]);  // Open screen window
    myInit ();
    glutDisplayFunc( myDisplay );  // Register redraw function
    glutMouseFunc( myMouse );  // Register mouse function
    glutMainLoop();  // Go into a perpetual loop
    return 0;
}
```

To create a screen window
2.3 An OpenGL Example (conti)

- **`glColor3f()`**: set foreground color
- **World coordinate system**
  - `glMatrixMode()`
  - `glLoadIdentity()`
  - `gluOrtho2D()`
- **`myDisplay( void )`**
  - Called when the screen is redrawn (expose event)

Specify which matrix stack is the target for subsequent matrix operations

Replace current matrix with identity matrix
2.3 An OpenGL Example (conti)

- **glutCreateWindow( argv[0])**
  - Map the window and generate an “expose” event
  - argv[0] can be replaced with "Name of window“ such as glutCreateWindow ("OpenGL window")

- **myMouse**
  - State has two values: pressed or released
  - X and y stand for the location of the cursor
2.4 Include Files

- To run an OpenGL program, we need to include appropriate OpenGL libraries first.
- Related libraries:
  - OpenGL
  - OpenGL Utility Library
  - OpenGL Utility Toolkit

Graphics interface for low-level drawing commands
Provide higher-level drawing commands
Provide interface to windowing system and input devices
OpenGL

- 3D graphics format
- Include file: `<GL/gl.h>`
- GL routines use the prefix: `gl`
  
  **Example**
  
  `glClearColor (1.0, 1.0, 1.0, 0.0)`
  
  `glClear (GL_COLOR_BUFFER_BIT)`

Specify clear values for the color buffers

Clear buffer to pre-set values
OpenGL Utility Library (GLU)

- setting up matrices for viewing transformation
- performing polygon tessellation
- rendering surfaces
- include file: `<GL/glu.h>`
- GLU routines use the prefix: `glu`

- e.g. `gluOrtho2D (0.0, 640.0, 0.0, 480.0)`

Quadric surfaces and NURBS

Define a 2D orthographic projection matrix

to create texture bitmaps from a base image
OpenGL Utility Toolkit (GLUT)

- window management
- event management
- window system-independent
- include file: `<GL/glut.h`
- GLUT routines use the prefix: `glut`
  e.g. `glutInitWindowSize (640, 480)`
  `glutInit(&argc, argv)`

Appeals to C-hackers (console for printf()’s, etc)
OpenGL Utility Toolkit (GLUT)

Note that

```c
glutInit(&argc, argv)
```

// start glut library, pass any extra command
// line commands to glut.
// glutInit must be called before any other glut
// commands, both argc and argv can be null
// values.

Appeals to C-hackers
(console for printf()'s, etc)
Getting GLUT

Web site:
Windows: www.xmission.com/~nate/glut.html
Others: www.opengl.org/developers/documentation/glut.html

www.sourceforge.net/projects/uncpython-tools

Overview:
Appendix D of OpenGL Programming Guide
2.5 OpenGL Command Syntax

```
glVertex2i
```

- **gl**: library
- **Vertex2i**: basic command
- **2**: number of arguments
- **i**: type of argument
Constants

OpenGL defined constants begin with `GL_`, use all capital letters, and use underscores to separate words.

- `GL_COLOR_BUFFER_BIT`
- `GL_POINTS`
- `GL_LINES`
- `GL_POLYGON`
- `GL_LINE_STRIP`
- `GL_LINE_LOOP`
<table>
<thead>
<tr>
<th>Suffix</th>
<th>Data type</th>
<th>Typical C or C++ type</th>
<th>OpenGL type name</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>8-bit int</td>
<td>signed char</td>
<td>GLbyte</td>
</tr>
<tr>
<td>s</td>
<td>16-bit int</td>
<td>short</td>
<td>GLshort</td>
</tr>
<tr>
<td>i</td>
<td>32-bit int</td>
<td>Int or long</td>
<td>Glint, GLsizei</td>
</tr>
<tr>
<td>f</td>
<td>32-bit float</td>
<td>float</td>
<td>GLfloat, GLclampf</td>
</tr>
<tr>
<td>d</td>
<td>64-bit float</td>
<td>double</td>
<td>GLdouble, GLclampd</td>
</tr>
<tr>
<td>ub</td>
<td>8-bit unsigned</td>
<td>Unsigned char</td>
<td>GLubyte, GLboolean</td>
</tr>
<tr>
<td>us</td>
<td>16-bit unsigned</td>
<td>Unsigned short</td>
<td>GLushort</td>
</tr>
<tr>
<td>ui</td>
<td>32-bit unsigned</td>
<td>Unsigned int or unsigned long</td>
<td>GLuint, GLenum, GLbitfield</td>
</tr>
</tbody>
</table>
Note:

- use OpenGL defined data types throughout your application to avoid mismatched types when porting your code between different implementations.
2.6 What do they do?

void myInit (void) {
    ...
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluOrtho2D (0.0, 640.0, 0.0, 480.0);
    // Establishing a simple coordinate system
}

Specify which matrix stack is the target for subsequent matrix operations

world
What do they do? (conti)

void myDisplay (void) {
 ...
 ...

  glFlush ();
  // Force execution of the above commands
}


What do they do?  (conti)

```c
int main (int argc, char** argv) {
    ...
    glutDisplayFunc ( myDisplay );
    glutMouseFunc ( myMouse );
    glutMainLoop ( );
    // Draw the initial picture and enter
    // the perpetual event-checking loop
}
```
2.7 Interaction with the Mouse and Keyboard

Callback function registration commands:

- glutMouseFunc (myMouse)
- glutMotionFunc (myMovedMouse)
- glutKeyboardFunc (myKeyboard)

Event handler for mouse event
Event handler for motion event
Event handler for keyboard event
Callback function prototypes:

void myMouse(int button, int state, int x, int y);

void myMovedMouse(int mouseX, int mouseY);

void myKeyboard(unsigned char theKey, int mouseX, int mouseY);
class GLintPoint {
    public:
        GLint x, y; }

void myMouse (int button, int state, int x, int y) {
    switch (button) {
        case GLUT_RIGHT_BUTTON:
            if (state == GLUT_DOWN)  exit (-1);
            break;
        default:
            break;
    }
}
Generating a Curve by Dragging the Mouse (conti)

```c
void myMovedMouse(int mouseX, int mouseY) {

    GLint x = mouseX;
    GLint y = screenHeight - mouseY;
    GLint brushSize = 20;
    glColor3f (1.0, 0.0, 0.0);
    // set the drawing color to red
    glRecti (x, y, x+brushSize, y+brushSize);
    glFlush ( );
}
```
Generating a Curve by Dragging the Mouse (conti)

```c
int main (int argc, char** argv) {
    glutInit (&argc, argv);       // initialize the toolkit
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
        // set display mode
    glutInitWindowSize (screenWidth, screenHeight);
        // set screen window size
    glutInitWindowPosition (100, 150);
        // set window position on screen
    glutCreateWindow (argv[0]);   // open the screen window
    myInit ();
    glutDisplayFunc (myDisplay);  // register redraw function
    glutMouseFunc (myMouse);     // register myMouse
    glutMotionFunc (myMovedMouse);  // register myMoveMouse
    glutMainLoop();                      // go into a perpetual loop
    return 0;
}
```
3. OpenGL - II

3.1 World Coordinate System, World Window, & Viewport

- Using the device coordinate system (DCS) directly is not flexible for many applications. Why?
  - Can deal with integers only
  - There is a maximum on the range of $x$ and $y$
Device-independent approach:
- Do the drawing in a World Coordinate System (WCS)
- Use world window to define the region to be shown
- Use viewport (a rectangular region of the screen window) to show the drawing
Illustration: 

World Window

Viewport

Screen Window
• Need a **window-to-viewport** mapping

• The mapping preserves **aspect ratio**
  \[ ( = \frac{width}{height} ) \]

• **clipping**: anything outside the world window should be discarded before the mapping

• **Clipping** and **mapping** are performed by OpenGL

• Example: plot \( sinc(x) = \frac{\sin(\pi x)}{\pi x} \) between \( x = -4 \) and \( x = 4 \) in the viewport \((0, 640, 0, 480)\).
Ideal condition: write the code the following way and let the system worry about the mapping (transformation)

```c
void myDisplay ( void )
{
    glBegin ( GL_LINE_STRIP );
    for (GLfloat x = -4.0 ; x < 4.0 ; x += 0.1 )
    {
        GLfloat y = sin (3.14159 * x) / (3.14159 * x);
        glVertex2f (x, y);
    }
    glEnd ( );
    glFlush ( );
}
```

How?
Window-to-Viewport Mapping: Preserving properties

\[
\begin{align*}
\frac{sx - V.l}{V.r - V.l} &= \frac{x - W.l}{W.r - W.l} \\
\frac{sy - V.b}{V.t - V.b} &= \frac{y - W.b}{W.t - W.b}
\end{align*}
\]
Hence

\[ sx = A \bullet x + C \quad sy = B \bullet y + D \]

where

\[ A = \frac{V \cdot r - V \cdot l}{W \cdot r - W \cdot l}, \quad C = V \cdot l - A \bullet W \cdot l \]

\[ B = \frac{V \cdot t - V \cdot b}{W \cdot t - W \cdot b}, \quad D = V \cdot b - B \bullet W \cdot b \]
Doing it in OpenGL:

Set Window:

\[
\begin{align*}
\text{glMatrixMode} & \left(\text{GL_PROJECTION}\right); \\
\text{glLoadIdentity} & \left(\right); \\
\text{gluOrtho2D} & \left(W\_left, W\_right, W\_bottom, W\_top\right)
\end{align*}
\]

Set Viewport:

\[
\begin{align*}
\text{glViewport} & \left(V\_left, V\_bottom, V\_width, V\_height\right)
\end{align*}
\]
Example:

```c
void myDisplay()
{
    glClear(GL_COLOR_BUFFER_BIT); // clear the screen
    //
    glMatrixMode(GL_PROJECTION);
    gluLoadIdentity();
    gluOrtho2D(-5.0, 5.0, -0.3, 1.0); // set the window
    //
    glViewport(0, 0, 640, 480); // set the viewport
    //
    glBegin(GL_LINE_STRIP);
        for (GLfloat x=-4.0; x<4.0; x += 0.1) { // draw the plot
            glVertex2f(x, sin(3.14159*x)/(3.14159*x));
        }
    glEnd();
    glFlush();
}
```
3.2 A Few Applications

1. Tile the screen window
   - Use a different viewport for each instance of the pattern

```c
void myDisplay(void)
{
    glClear ( GL_COLOR_BUFFER_BIT );

    glMatrixMode ( GL_PROJECTION );
    glLoadIdentity ( ) ;
    gluOrtho2D ( -5.0, 5.0, -0.3, 1.0 );

    for (int i=0; i < 10; i++)
        for (int j=0; j < 11; j++) {
            glViewport ( i*64, j*44, 64, 44);
            glBegin ( GL_LINE_STRIP );
            for ( GLfloat x = -4.0; x < 4.0; x += 0.1 )
                glVertex2f ( x, sin(3.14159 * x) / (3.14159 * x ) );
            glEnd ( );
        }
    glFlush();
}
```
Tile the screen
2. Flip an image up side down
   - Simply flip the window up side down

```c
//*************************************************************************/
void myDisplay ( void )
{
    glClear ( GL_COLOR_BUFFER_BIT );
    //
    setWindow ( -5.0, 5.0, -0.3, 1.0 );
    //
    for ( int i=0; i < 10; i++ )
        for ( int j=0; j < 11; j++ ) {
            if ( ( i+j)%2 == 0 )
                setWindow ( -5.0, 5.0, -0.3, 1.0 );
            else
                setWindow ( -5.0, 5.0, 1.0, -0.3 );
            glViewport ( i*64, j*44, 64, 44 );
            glBegin ( GL_LINE_STRIP );
            for ( GLfloat x = -4.0; x < 4.0; x += 0.1)
                glVertex2f ( x, sin(3.14159 * x) / (3.14159 * x));
            glEnd ( );
        }
    glFlush ( );
}
```
2. Flip an image up side down
   - Simply flip the window up side down

```c
//********************************************************************************
void myDisplay ( void )
{
    glClear ( GL_COLOR_BUFFER_BIT );
    //
    setWindow ( -5.0, 5.0, -0.3, 1.0 );
    //
    for ( int i=0; i < 10; i++ )
        for ( int j=0; j < 11; j++ ) {
            if ( ( i+j)%2 == 0 )
                setWindow ( -5.0, 5.0, -0.3, 1.0 );
            else
                setWindow ( -5.0, 5.0, 1.0, -0.3 );
            glViewport ( i*64, j*44, 64, 44 );
            glBegin ( GL_LINE_STRIP );
            for ( GLfloat x = -4.0; x < 4.0; x += 0.1)
                glVertex2f ( x, sin(3.14159 * x) / (3.14159 * x));
            glEnd ( );
        }
    glFlush ( );
}
```
Flip an image up side down
3. Zooming effect
- Holding the viewport but reduce (zoom in) or increase (zoom out) the dimension of the window

```c
void myDisplay(void)
{
float cx = 0.0, cy = 0.3; // center of the window
float H, W = 5.0, aspect = 7.143;
int NumFrames = 200;

glClear(GL_COLOR_BUFFER_BIT); // clear the screen
setViewport(0, 640, 0, 480); // set the viewport
for(int frame = 0; frame < NumFrames; frame++)
{
    glClear(GL_COLOR_BUFFER_BIT); // clear the screen
    W *= 0.995; // reduce the window width
    H = W / aspect; // maintain the same aspect ratio
    setWindow(cx - W, cx + W, cy - H, cy + H);
    //set the next window
drawSincFunc ( );
}
}
```
Problems with this approach

- You get **display artifacts**, like parts of the scene visible in one state, the rest not visible or in some other state, the picture could be incomplete, etc.

How to achieve smooth animation?

- Use **double buffering**

**How?**

1. use "GLUT_DOUBLE" instead of "GLUT_SINGLE"
   
   ```c
   glutInitDisplayMode ( xxxx | GLUT_RGB );
   ```

2. Include the following instruction at the end of "myDisplay()".
   
   ```c
   glutSwapBuffers ( );
   ```
2.3 An OpenGL Example (conti)

```c
glVertex2i (150, 130);
glEnd();
glFlush(); // send all out to display
}

void myMouse (int button, int state, int x, int y) {
    switch (button) {
        case GLUT_RIGHT_BUTTON:
            if (state == GLUT_DOWN) exit (-1);
            break;
        default:
            break;
    }
}

int main (int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); // initialize the toolkit
    glutInitWindowSize (640, 480); // set display mode
    glutInitWindowSize (640, 480); // set screen window size
    glutInitWindowPosition (100, 150); // set window position on screen
    glutCreateWindow (argv[0]); // open the screen window
    myInit ();
    glutDisplayFunc( myDisplay ); // register redraw function
    glutMouseFunc( myMouse ); // register the mouse action function
    glutMainLoop(); // go into a perpetual loop
    return 0;
}
```
3. Zooming effect
- Holding the viewport but reduce (zoom in) or increase (zoom out) the dimension of the window

```c
//**************************************************************************
void myDisplay(void)
{
  float cx = 0.0, cy = 0.3; // center of the window
  float H, W = 5.0, aspect = 7.143;
  int NumFrames = 200;

glClear(GL_COLOR_BUFFER_BIT); // clear the screen
setViewport(0, 640, 0, 480); // set the viewport
for(int frame = 0; frame < NumFrames; frame++)
{
  glClear(GL_COLOR_BUFFER_BIT); // clear the screen
  W *= 0.995; // reduce the window width
  H = W / aspect; // maintain the same aspect ratio
  setWindow(cx - W, cx + W, cy - H, cy + H);
  //set the next window
drawSincFunc();
  glutSwapBuffers();
}
}
```
Line Clipping (Cohen-Sutherland algorithm)

- To avoid unnecessary computation, perform tests on trivially accepted cases and trivially rejected cases first.
- If both endpoints are inside the window, then the line segment is inside the window.
- If both endpoints are to the left ($x < x_{\text{min}}$), to the right ($x > x_{\text{min}}$), below ($y < y_{\text{min}}$), or above ($y > y_{\text{min}}$) the window, then the line segment is outside the window.
\[ X = X_{\text{min}} \quad \text{and} \quad X = X_{\text{max}} \]

\[ Y = Y_{\text{min}} \quad \text{and} \quad Y = Y_{\text{max}} \]
Defining 4-bit out code:

```
 1001   1000   1010
 0001   0000   0010
 0101   0100   0110
```
1. Compute the **out codes** for the endpoints of the line segment to be clipped

2. **Repeat** until the line segment is either **trivially accepted** or **rejected**

   2.1 [ **Trivial Acceptance Test** ]
   
   If bitwise OR of the codes is 0000 (line segment is inside the window), draw the line segment and stop.

   2.2. [ **Trivial Rejection Test** ]
   
   If bitwise AND of the codes is not 0000 (line segment is outside the window), discard the line segment and stop.
2.3. [ Subdivide the segment ]

2.3.1 Pick an endpoint whose code is non-zero (the endpoint that is outside the window)

2.3.2 Find the first non-zero bit: this corresponds to the window edge which intersects the line segment

2.3.3 Compute the intersection point and replace the outside endpoint with the intersection point
An Example

<table>
<thead>
<tr>
<th></th>
<th>1001</th>
<th>1000</th>
<th>1010</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0001</td>
<td>0000</td>
<td>0010</td>
</tr>
<tr>
<td>B</td>
<td>0101</td>
<td>0100</td>
<td>0110</td>
</tr>
</tbody>
</table>

A: 1010
B: 0101

A: 1010
& B: 0101

1111
0000

Use bit 2 of $A$ (right clipping edge) to do the subdivision
Subdivide at $C$ (Find $y$ coordinate of $C$)

$$y = m \cdot x_{\text{max}} + b$$
Example (con’t)

Use bit 4 of C (top clipping edge) to do the subdivision

Subdivide at \( D \) (need to find \( x \) coordinate of \( D \))

\[
x = \left( y_{\text{max}} - b \right)/m
\]
Example (con’t)

Use bit 1 of B (left clipping edge) to do the subdivision
Subdivide at E (need to find y coordinate of E)
\[ y = m \cdot x_{\min} + b \]
Segment $ED$ is trivially accepted
End