Announcements

- All code from book (working programs) on book website.
  - Quite useful. Take a look
Screen Coordinate System

- Screen: 2D coordinate system (WxH)
- 2D Cartesian Grid
- Origin (0,0): lower left corner (OpenGL convention)
- Horizontal axis – x
- Vertical axis – y
- Pixel positions: grid (x, y) intersections (0,0)
Screen Coordinate System

(0,0) is lower left corner of **OpenGL Window.** NOT lower left corner of entire desktop
Defining a Viewport

- Can draw to any rectangle (sub-area of screen)
- **Viewport**: Area of screen we want to draw to
- To define viewport
  
  \[
  \text{glViewport(left, bottom, width, height)}
  \]
  
  or \( \text{glViewport(V.L, V.B, V.R - V.L, V.T - V.B)} \)
  
  e.g. \( \text{glViewport(180, 260, (410 - 180), (480 - 260)}) \)
Recall: OpenGL Skeleton

```c
void main(int argc, char** argv)
{
    // First initialize toolkit, set display mode and create window
    glutInit(&argc, argv);  // initialize toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("my first attempt");
    glewInit();

    // … now register callback functions
    glutDisplayFunc(myDisplay);
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutKeyboardFunc(myKeyboard);

    myInit();
    glutMainLoop();
}
```

```c
void myDisplay(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glDrawArrays(GL_LINE_LOOP, 0, 3);
    glFlush();
}
```

Note: default viewport is entire created window
Example: Changing Viewport

How to change viewport to:
   Bottom left corner at (100,80)
   Width changes to 700, height changes to 300??

```c
void main(int argc, char** argv){
    // First initialize toolkit, set display mode and create window
    glutInit(&argc, argv);   // initialize toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("my first attempt");
    glewInit( );

    // … now register callback functions
    glutDisplayFunc(myDisplay);
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutKeyboardFunc(myKeyboard);

    myInit( );
    glutMainLoop( );
}

void mydisplay(void){
    glClear(GL_COLOR_BUFFER_BIT);
    glViewport(100, 80, 700, 300);
    glDrawArrays(GL_LINE_LOOP, 0, 3);
    glFlush( );
}
```

**Note:** Set desired viewport, then draw
Tiling: Changing Viewport in a Loop

- **Problem:** Want to tile Triangle file on screen
- **Solution:** change viewport in loop, draw tiles
Tiling Triangle Code Snippet

- Set viewport, draw into tile in a loop
- Code snippet to draw 6x6 tiles:

```c
float w, h;

w = width / 6;
h = height / 6;

for (int k=0; k<6; k++) {
    for (int m=0; m<6; m++) {
        glViewport(k * w, m * h, w, h);
        glDrawArrays(GL_LINE_LOOP, 0, NumPoints);
    }
}
```
Example: Tiling, Changing Viewport

```c
void main(int argc, char** argv){
    // First initialize toolkit, set display mode and create window
    glutInit(&argc, argv);       // initialize toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("my first attempt");
    glewInit( );

    // ... now register callback functions
    glutDisplayFunc(myDisplay);
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutKeyboardFunc(myKeyboard);

    myInit( );
    glutMainLoop( );
}

void mydisplay(void){
    glClear(GL_COLOR_BUFFER_BIT);
    float w, h;

    w = width / 6; h = height / 6;

    for (int k=0; k<6; k++) {
        for (int m=0; m<6; m++) {
            glViewport(k * w, m * h, w, h);
            glDrawArrays(GL_LINE_LOOP, 0, NumPoints);
        }
    }
    glFlush( );
}
```
World Coordinate System

- Problems with drawing in screen coordinates:
  - *(x,y) dimensions in pixels*: one mapping, inflexible
  - Not application-specific
- **World coordinate**: application-specific
- E.g: Same screen area. Change input drawing *(x,y)* range

100 pixels = 30 miles

100 pixels = 0.25 miles
Using Window Coordinates

- Would like to:
  - Specify **source** boundaries (extents) of original drawing in world coordinates (miles, meters, etc)
  - Display **target region** in screen coordinates (pixels)

- Programming steps:
  1. Define world window (original drawing extents)
  2. Define viewport (drawing extents on screen)
  3. Map drawings within window to viewport

- Mapping called **Window-to-viewport mapping**!
World Coordinate System

- **World Window:** region of source drawing to be rendered
- Rectangle specified by world window is drawn to screen
- Defined by (left, right, bottom, top) or (W.L, W.R, W.B, W.T)
Defining World Window

- mat4 ortho = Ortho2D(left, right, bottom, top)
- Or mat4 ortho = Ortho2D(W.L, W.R, W.B, W.T)

- Ortho2D generates 4x4 matrix that scales input drawing
- Note: Need to include mat.h (contains Ortho2D)
Drawing

- After setting world window (using ortho2D) and viewport (using glviewport),
  - Draw as usual with glDrawArrays
Apply ortho( ) matrix in Vertex Shader

- **One more detail:** Need to pass ortho matrix to shader
- Multiply each vertex by ortho matrix to scale input drawing
- Need to connect *ortho* matrix to *proj* variable in shader

```
mat4 ortho = Ortho2D( W.L, W.R, W.B, W.T );

uniform mat4 Proj;
in vec4 vPosition;

void main( ){
    gl_Position = Proj * vPosition;
}
```

Call Ortho2D in Main .cpp file

In vertex shader, multiply each vertex with *proj* matrix
Apply ortho( ) matrix in Vertex Shader

1. Include mat.h from book website (ortho2D declared in mat.h)

```cpp
#include "mat.h"
```

2. Connect ortho matrix to proj variable in shader

```cpp
mat4 ortho = Ortho2D( W.L, W.R, W.B, W.T );

ProjLoc = glGetUniformLocation( program, "Proj" );
glUniformMatrix4fv( ProjLoc, 1, GL_TRUE, ortho );
```

```cpp
uniform mat4 Proj;
in vec4 vPosition;

void main( ){
    gl_Position = Proj * vPosition;
}
```

In shader, multiply each vertex with proj matrix

Call Ortho2D in Main .cpp file
Drawing Polyline Files

- May read in list of vertices defining a drawing
- **Problem:** want to draw single dino.dat on screen
- **Note:** size of input drawing may vary

Vertices (points) of Drawing read in from file (e.g. dino.dat)
**Problem:** want to draw single dino.dat on screen

**pseudocode snippet:**

```cpp
// set world window (left, right, bottom, top)
ortho = Ortho2D(0, 640.0, 0, 440.0);

//..... Pass ortho to vertex shader... then...

// now set viewport (left, bottom, width, height)
glViewport(0, 0, 64, 44);

// Draw polyline fine
drawPolylineFile(dino.dat);
```

**Question:** What if I wanted to draw the bottom quadrant of polyline?
Tiling using W-to-V Mapping

- **Problem:** Want to tile polyline file on screen
- **Solution:** W-to-V in loop, adjacent tiled viewports
Tiling Polyline Files

- Problem: want to tile `dino.dat` in 5x5 across screen
- Code snippet:

```cpp
// set world window
ortho = Ortho2D(0, 640.0, 0, 440.0);

//..... Pass ortho to vertex shader... then...

for(int i=0; i < 5; i++)
{
  for(int j = 0; j < 5; j++)
  {
    // .. now set viewport in a loop
    glViewport(i * 64, j * 44; 64, 44);
    drawPolylineFile(dino.dat);
  }
}
```
Maintaining Aspect Ratios

- Aspect ratio $R = \text{Width}/\text{Height}$
- What if window and viewport have different aspect ratios?
- Two possible cases:

  **Case a:** viewport too wide
  ![Case a: viewport too wide](image)

  **Case b:** viewport too tall
  ![Case b: viewport too tall](image)
What if Window and Viewport have different Aspect Ratios?

- **R** = window aspect ratio, \( W \times H \) = viewport dimensions
- Two possible cases:
  - **Case A \((R > W/H)\)**: map window to tall viewport?

```plaintext
ortho = Ortho2D(left, right, bottom, top);
R = (right - left)/(top - bottom);
If(R > W/H)
    glViewport(0, 0, W, W/R);
```

Aspect ratio \( R \)

Viewport

Window

Orthographic projection:

- \( ortho \) = `Ortho2D(left, right, bottom, top)`;
- \( R = \frac{\text{right} - \text{left}}{\text{top} - \text{bottom}} \);
- If \((R > W/H)\)
  
  ```plaintext
  glViewport(0, 0, W, W/R);
  ```
What if Window and Viewport have different Aspect Ratios?

- **Case B \( (R < \frac{W}{H}) \):** map window to wide viewport?

```cpp
ortho = Ortho2D(left, right, bottom, top );
R = (right - left)/(top - bottom);
If(R < \frac{W}{H})
    glViewport(0, 0, H*R, H);
```
reshape( ) function that maintains aspect ratio

// Ortho2D(left, right, bottom, top )is done previously,
// probably in your draw function
// function assumes variables left, right, top and bottom
// are declared and updated globally

void myReshape(double W, double H ){
    R = (right - left)/(top - bottom);

    if(R > W/H)        // tall viewport
        glViewport(0, 0, W, W/R);
    else if(R < W/H)   // wide viewport
        glViewport(0, 0, H*R, H);
    else
        glViewport(0, 0, W, H);  // equal aspect ratios
}
Interaction
Adding Interaction

- So far, OpenGL programs just render images
- Can add user interaction
- Examples:
  - User hits ‘h’ on keyboard -> Program draws house
  - User clicks mouse left button -> Program draws table
Types of Input Devices

- **String**: produces string of characters e.g. keyboard

- **Locator**: User points to position on display. E.g mouse
Types of Input Devices

- **Valuator**: generates number between 0 and 1.0 (proportional to how much it is turned)

- **Pick**: User selects location on screen (e.g. touch screen in restaurant, ATM)
GLUT: How keyboard Interaction Works

- Example: User hits ‘h’ on keyboard -> Program draws house

1. User hits ‘h’ key

Keyboard handler Function

‘h’ key

OS

Programmer needs to write keyboard handler function
Using Keyboard Callback for Interaction

```c
void main(int argc, char** argv){
    // First initialize toolkit, set display mode and create window
    glutInit(&argc, argv);  // initialize toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("my first attempt");
    glewInit( );

    // ... now register callback functions
    glutDisplayFunc(myDisplay);
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutKeyboardFunc(myKeyboard);
    myInit( );
    glutMainLoop( );
}
```

1. Register keyboard Function

2. Implement keyboard function

```c
void myKeyboard(char key, int x, int y )
{
    // put keyboard stuff here
    ........
    switch(key){
        case 'f':
            // do stuff
            break;
        case 'k':
            // do other stuff
            break;
    }
    ........
}
```

Note: Backspace, delete, escape keys checked using their ASCII codes
glutSpecialFunc (specialKeyFcn);

......

Void specialKeyFcn (Glint specialKey, GLint, xMouse, GLint yMouse)

- Example: if (specialKey == GLUT_KEY_F1) // F1 key pressed
  - GLUT_KEY_F1, GLUT_KEY_F12, .... for function keys
  - GLUT_KEY_UP, GLUT_KEY_RIGHT, .... for arrow keys keys
  - GLUT_KEY_PAGE_DOWN, GLUT_KEY_HOME, .... for page up, home keys

- Complete list of special keys designated in glut.h
GLUT: How Mouse Interaction Works

- Example: User clicks on (x,y) location in drawing window -> Program draws a line

1. User clicks on (x,y) location

Mouse handler Function

OS

Programmer needs to write keyboard handler function
Using Mouse Callback for Interaction

void main(int argc, char** argv){
    // First initialize toolkit, set display mode and create window
    glutInit(&argc, argv);   // initialize toolkit
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("my first attempt");
    glewInit( );

    // … now register callback functions
    glutDisplayFunc(myDisplay);
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutKeyboardFunc(myKeyboard);

    myInit( );
    glutMainLoop( );
}

1. Register keyboard Function

2. Implement mouse function

    void myMouse(int button, int state, int x, int y)
    {
        // put mouse stuff here
        ............
    }

Mouse Interaction

- Declare prototype
  - `mymouse(int button, int state, int x, int y)`
  - `mymovedmouse`

- Register callbacks:
  - `glutMouseFunc(mymouse)`: mouse button pressed
  - `glutMotionFunc(mymovedmouse)`: mouse moves with button pressed
  - `glutPassiveMotionFunc(mymovedmouse)`: mouse moves with no buttons pressed

- Button returned values:
  - `GLUT_LEFT_BUTTON`, `GLUT_MIDDLE_BUTTON`, `GLUT_RIGHT_BUTTON`

- State returned values:
  - `GLUT_UP`, `GLUT_DOWN`

- X,Y returned values:
  - x,y coordinates of mouse location
Mouse Interaction Example

- **Example**: draw (or select) rectangle on screen
- Each mouse click generates separate events
- Store click points in `global` or `static` variable in mouse function

```c
void myMouse(int button, int state, int x, int y)
{
    static GLintPoint corner[2];
    static int numCorners = 0; // initial value is 0
    if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        corner[numCorners].x = x;
        corner[numCorners].y = screenHeight - y; // flip y coord
        numCorners++;
    }
}
```

Screenheight is height of drawing window
if(numCorners == 2)
{
    // draw rectangle or do whatever you planned to do
    Point3 points[4] = corner[0].x, corner[0].y, //1
                     corner[1].x, corner[0].y,     //2
                     corner[1].x, corner[1].y,     //3
                     corner[0].x, corner[1].y);    //4

    glDrawArrays(GL_QUADS, 0, 4);

    numCorners == 0;
}
else if(button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
    glClear(GL_COLOR_BUFFER_BIT); // clear the window
    glFlush( );
Menus

- Adding menu that pops up on mouse click

1. Create menu using `glutCreateMenu(myMenu);`

2. Use `glutAddMenuEntry` adds entries to menu

3. Attach menu to mouse button (left, right, middle) using `glutAttachMenu`
Menus

- Example:

```c
void mymenu(int value)
{
    if(value == 1){
        glClear(GL_COLOR_BUFFER_BIT);
        glFlush();
    }
    if (value == 2) exit(0);
}
```

```c
// Shows on menu
// Checked in mymenu

// glutCreateMenu(myMenu);
// glutAddMenuEntry("Clear Screen", 1);
// glutAddMenuEntry("Exit", 2);
// glutAttachMenu(GLUT_RIGHT_BUTTON);
```

---

<table>
<thead>
<tr>
<th>Shows on menu</th>
<th>Checked in mymenu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Screen</td>
<td>1</td>
</tr>
<tr>
<td>Exit</td>
<td>2</td>
</tr>
</tbody>
</table>
GLUT Interaction using other input devices

- Tablet functions (mouse cursor must be in display window)

  ```c
  glutTabletButton (tabletFcn);
  ```

  ```c
  void tabletFcn(Glint tabletButton, Glint action, Glint xTablet, Glint yTablet)
  ```

- Spaceball functions
- Dial functions
- Picking functions: use your finger
- Menu functions: minimal pop-up windows within your drawing window
- Reference: Hearn and Baker, 3rd edition (section 20-6)
References