Interaction II

Objectives

Double Buffering
Selection/Picking
Example Reshape: Ortho

```c
void myReshape(int w, int h)
{
    glViewport(0,0,w,h);
    float ar = w/h;
    if( ar < startAR ) { // (w <= h ){//taller
        proj = Ortho(vl, vr, vl * (GLfloat) h / (GLfloat) w, vrt * (GLfloat) h / (GLfloat) w, 0.1,10.0);
    } else { //wider
        proj = Ortho(vb * (GLfloat) w / (GLfloat) h, vt * (GLfloat) w / (GLfloat) h, vb, vt,0.1,10.0);
    }
}
```

How would you modify to work with Perspective Projections?

OpenGL Orthogonal Viewing

**Ortho** *(left, right, bottom, top, near, far)*

*near* and *far* measured from camera
Perspective Viewing

\[ \text{perspective}(\text{fovy}, \text{aspect}, \text{near}, \text{far}) \]

often provides a better interface

\[ \text{aspect} = \frac{w}{h} \]

Perspective Viewing with Frustum

\[ \text{frustum}(\text{left}, \text{right}, \text{bottom}, \text{top}, \text{near}, \text{far}) \]

near and far are distances from cam and should always be positive!!!
Graphics System: Single Buffer

CPU

GPU

GPU Memory (dual ported)

Display Controller

Single Buffering

Example:

With Tearing

No Tearing

https://github.com/melonjs/melonJS/issues/274
Graphics System: Double Buffer

Double Buffering in OpenGL

```c
glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE);

glutSwapBuffers(); // at end of the display() function
```

Notice these are glut functions. Most window interfaces are going to provide a method for double buffering.
Selection/Picking

How can you determine when a primitive is 'picked' by the mouse?

Ray Picking
- Convert mouse coordinates to image plane coordinates (invert the screen space transform)
- Construct an “inverted” projector
- Test for intersection with scene objects

Aside: Transformations from screen to world...

- Viewport (screen units)
- View Plane (world frame)
**Color Buffer Picking**

Assign a unique ID to each ‘pickable’ element
Convert that ID to a ‘color’
Upon a mouse click, render the scene twice:
  - Render Pass 1: Unique color render
  - Render Pass 2: Normal Render
On Render Pass 1:
  - Read the Pixels under the mouse pointer
  - Convert RGB back to an ID

---

**Color Buffer Picking**

Assign a unique ID
  - Simple integer counter
Convert ID to a ‘color’
  - GPU Colorbuffer is made of 3 values – R,G,B
  - One byte for each R, G, and B
  - (24 bit color = $2^{24}$ possible colors)
API’s typically require color specification independent of this (values [0..1])
Color Buffer Picking: ID to Color

Convert ID (i) to 24-bit color

\[
\begin{align*}
\text{int } r &= (i \& 0x000000FF) \gg 0; \\
\text{int } g &= (i \& 0x0000FF00) \gg 8; \quad \text{// or divide by 256} \\
\text{int } b &= (i \& 0x00FF0000) \gg 16; \quad \text{// or divide by 256*256}
\end{align*}
\]

Initialize i to 0
Store lower order bits in \( r \)
Store next 8 bits in \( g \)
Store final 8 bits in \( b \)

Result: 3 integers \([0...255]\) for each of R, G, and B

Color Buffer Picking: Rendering

Render Pass 1: Assign the primitive a color equivalent to its unique ID as converted to R,G,B values

\[
\text{Remember to convert to } [0..1] \\
r/255.0f, g/255.0f, b/255.0f
\]

After Render Pass1, the color buffer holds all rendered objects with colors that correspond to unique IDs

DO NOT SWAP BUFFERS YET – WE DON’T WANT THE USER TO SEE THIS RENDERING!
Color Buffer Picking: Mouse Callback

Get the mouse $x,y$

Read the pixels from the framebuffer

```c
GLubyte pixel[4];
glReadPixels(x, viewport[3] - y, 1, 1, GL_RGBA, GL_UNSIGNED_BYTE, pixel);
```

Color Buffer Picking: Convert back to ID

Undo ID to Color Conversion

```c
int id =
    pixel[0] +
    pixel[1] * 256 +
    pixel[2] * 256*256;
```

Use this ID in your application to determine which ‘pickable object’ is being selected

Now, render the scene as usual for users to see!