Road Map

1. Set up a **Camera** *(Tuesday)*
   - the viewing frustum has near and far clipping planes

2. Create some **Geometry** *(Tuesday & Next Week)*
   - made out of triangles

3. Place the geometry in the scene *(Tuesday)*
   - using **Transformations**
   - make sure it’s in the camera’s viewing frustum

4. SNAP THE PICTURE *(TODAY…)*
   - internal workings of OpenGL and the GPU
   - triangle vertices are **projected** onto the film plane
   - projected triangles are **rasterized** into pixels
   - this all happens in a buffer that is eventually swapped to the display
Coordinate Frames
Coordinate Frames

Object Space (model space)
- Raw values as provided by `glBufferData` to a vertex buffer object (e.g. teapot centered at origin)

World Space (scene)
- Object at final location in the environment (e.g. teapot on top of a table on the ground off to the left side of the screen)

Screen Space (film plane)
- Object splatted into pixels onto a 2D screen position

```uniform mat4 modelMatrix, viewMatrix, projectionMatrix;

- viewMatrix*modelMatrix: object to world transform
- projectionMatrix: world to screen transform
- CTM = projectionMatrix*viewMatrix*modelMatrix```
Screen Space Projection
World to Screen Transform

(x, y, z)

( x', y', h )

pinhole

film plane

world space

screen space

\[
\begin{align*}
x &= x' \\
\frac{z}{h} &= h \\
y &= y' \\
\frac{z}{h} &= h
\end{align*}
\]

\[
x' = h \frac{x}{z} \\
y' = h \frac{y}{z}
\]

glm::perspective creates a matrix to perform perspective projection to transform objects from world space to screen space (film plane)
Perspective Projection

\[ x' = h \frac{x}{z} \quad y' = h \frac{y}{z} \]

- Using **homogeneous coordinates** and setting \( w' = z \) creates a **linear** function (a 4X4 matrix multiply) for a **nonlinear** function \( 1/z \)

\[ x'w' = hx \]
\[ y'w' = hy \]
\[ z'w' = az + b \]
\[ w' = z \]

\[
\begin{bmatrix}
  x'w' \\
  y'w' \\
  z'w' \\
  w'
\end{bmatrix}
= 
\begin{bmatrix}
  h & 0 & 0 & 0 \\
  0 & h & 0 & 0 \\
  0 & 0 & a & b \\
  0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]

What are \( a \) and \( b \)?
Perspective Projection

\[ z'z = az + b \]

- Mapping \( z=n \) to \( z'=n \) and \( z=f \) to \( z'=f \) gives:
  \[
  n^2 = an + b \\
  f^2 = af + b
  \]
- Two equations, two unknowns. The solution is
  \[
  a = n + f \\
  b = -fn
  \]

- Near/far clipping planes remain unchanged
- 3D frustum in world space becomes an orthographic volume in “screen space”
- Use \( z' \) for occlusions
Rasterization
Rasterization

- Rasterize 2D triangles, after transforming the vertices to screen space
- Color the pixels inside the triangle with the RGB–color of the triangle
Normal to a Line

\[ p_0 = (x_0, y_0) \]

\[ t = p_1 - p_0 = (x_1 - x_0, y_1 - y_0) \]

\[ n = (y_1 - y_0, -(x_1 - x_0)) \text{ normalize, if desired} \]
Implicit Equation for a Line

All points $p$ on the line, have:

$$(p - p_0) \cdot n = 0$$
Implicit Equation for a Ray

- Normal points to the right of the ray direction
- “Interior” points are to the left of the ray, and have negative \((p - p_0) \cdot n\) values
Point Inside Ray Test

makeline( vert& v0, vert& v1, line& l )
{
    l.a = v1.y - v0.y;
    l.b = v0.x - v1.x;
    l.c = -(l.a * v0.x + l.b * v0.y);
}

Decide if a point(x,y) is inside a line:
\[ e = l.a \times x + l.b \times y + l.c \]
\{ e \leq 0 \text{ inside(on the left)} \}
\{ e > 0 \text{ outside(on the right)} \}
Point Inside Triangle Test

- Inside a triangle, if inside (to the left of) all 3 rays
- **Back facing triangles are not rendered**, since no points are to the left of all three rays

Counter Clockwise (Facing Camera)

Clockwise (Facing Away from Camera)
Pixel Center Inside Triangle Test

rasterize( vert v[3] )
{
    line l0, l1, l2;
makeline(v[0],v[1],l2);
makeline(v[1],v[2],l0);
makeline(v[2],v[0],l1);
for( y=0; y<YRES; y++ ) for( x=0; x<XRES; x++ ) {
    e0 = l0.a * x + l0.b * y + l0.c;
e1 = l1.a * x + l1.b * y + l1.c;
e2 = l2.a * x + l2.b * y + l2.c;
if( e0<=0 && e1<=0 && e2<=0 )
    fragment(x,y);
}
}
Indeterminate Cases

- Pixels on a shared edge between two triangles are flagged by both triangles (e=0 on both)
  - Wasted effort drawing duplicate fragments
  - Problems for transparent objects
- Not including these pixels (i.e. using e < 0) causes gaps:
Indeterminate Cases

- Don’t draw edges to the right and above each triangle/polygon (omit bold face segments and hollow points in these figures)

```cpp
int shadow( line l ) {
    return (l.a>0) || (l.a == 0 && l.b > 0);
} // normal points right || vertical normal pointing up
```

```cpp
int inside( value e, line l ) {
    return (e == 0) ? !shadow(l) : (e < 0);
} // if e=0, don’t shade shadow line
```
Pixel Center Inside Triangle Test

rasterize( vert v[3] )
{
    line l0, l1, l2;
    makeline(v[0],v[1],l2);
    makeline(v[1],v[2],l0);
    makeline(v[2],v[0],l1);
    for( y=0; y<YRES; y++ ) for( x=0; x<XRES; x++ ) {
        e0 = l0.a * x + l0.b * y + l0.c;
        e1 = l1.a * x + l1.b * y + l1.c;
        e2 = l2.a * x + l2.b * y + l2.c;
        if( inside(e0,l0)&inside(e1,l1)&inside(e2,l2))
            fragment(x,y);
    }
}
Bounding Boxes
(Optimization!)
Bounding Box (rectangle)

- Inefficient to check every pixel on the screen
- Calculate a bounding box around the triangle, and only check pixels inside the box
- Round coordinates upward (ceil) to the nearest integer

Test points with filled circles
Don’t test hollow circles

```c
bound3( vert v[3], bbox& b )
{
    b.xmin = ceil(min(v[0].x, v[1].x, v[2].x));
    b.xmax = ceil(max(v[0].x, v[1].x, v[2].x));
    b.ymin = ceil(min(v[0].y, v[1].y, v[2].y));
    b.ymax = ceil(max(v[0].y, v[1].y, v[2].y));
}
```
rasterize( vert v[3] )
{
    bbox b; bound3(v, b);
    line l0, l1, l2;
    makeline(v[0],v[1],l2);
    makeline(v[1],v[2],l0);
    makeline(v[2],v[0],l1);
    for( y=b.ymin; y<b.ymax, y++ ) for( x=b.xmin; x<b.xmax, x++ ){
        e0 = l0.A * x + l0.B * y + l0.C;
        e2 = l2.A * x + l2.B * y + l2.C;
        if( inside(e0,l0)&inside(e1,l1)&inside(e2,l2) )
            fragment(x,y);
    }
}
Lighting and Shading
Lighting & Shading

- We ignored lights and reflective properties of objects for now
  - We’ll cover this the week after next
  - This means that you’d get a 2D splatted cartoon view of your objects
    - For example: a sphere turns into a 2D circle:

- So you’ll need a very simple light and a shader for the homework…
OpenGL
OpenGL has more than triangles...

Images

Bitmaps
OpenGl Pipeline

- **Individual Vertices**
- **Transformed Vertices**
- **Primitives**
- **Fragments**
- **Shaded Fragments**

**Commands Processor**

- **Per-vertex ops**
- **Primitive assembly**
- **Rasterization**
- **Per-fragment ops**

**Framebuffer ops**

**Display**

**Texturing**

- **Triangles, lines, points, images**
- **Pixels in the framebuffer**
Frame Buffering
Framebuffer

Example Framebuffer: 1440 x 900

The viewport is the portion of the window that can be drawn in, no pixels will appear outside the viewport.

Viewport (256x256)

Window (512 x 512)

All coordinates are integers; they refer to pixel locations in the framebuffer.
Frame Buffer Operations

Operation

- Test window ownership
- Test scissor and stencil mask
- Test alpha (transparency)
- Test depth (z-buffer)

Blending or compositing

Textured Fragments  Framebuffer Pixels
Depth Buffer (Z-Buffer)

- Initialize z-buffer to $z_{\text{max}}$
- Interpolate $z$ across the triangle
- **Draw fragment, if it’s closer**

```
if(frag.Z<Z[frag.X][frag.Y]){
}
```

Frame Buffering

Store image in a buffer to separate display refresh rate from drawing rate:

Single-buffer
- Draw into display buffer directly
- Can see picture being drawn

Double-buffer
- Display “front” buffer
- Draw into “back” buffer (can’t see drawing)
- Swap front and back (idle while waiting for swap)

Triple-buffer
- Avoid waiting for swap, keep drawing
GPU
(Machine Learning Too!)
Graphics Processing Unit

- OpenGL commands communicate with the GPU
- The GPU is designed to rapidly manipulate and alter memory to accelerate the building of images in a framebuffer
- In contrast, our ray tracer will be implemented on the CPU (although there are ray tracers that utilize the GPU, e.g. Nvidia Optix)
What’s in a GPU?

- Shader Core
- Shader Core
- Tex
- Primitive Assembly
- Rasterizer
- Framebuffer Ops
- Work Distributor
3.0 Ghz Intel Core2 Duo

Core 1

4MB L2 Cache

Core 2

2GB main memory (DDR2)

12.8 GB/sec

PCIe Bus (v1 = 4 GB/sec)

NVIDIA GeForce 8800 GTX (575 MHz)

(16 cores)

512MB video Memory (GDDR3)

12.8 GB/sec

84 GB/sec

NVIDIA GeForce 8800 GTX (575 MHz)

(16 cores)

PCIe Bus (v1 = 4 GB/sec)

NVIDIA 8800GTX

System board (Intel D975)
Xbox 360

3.2 Ghz PowerPC CPU

Core 1  Core 2  Core 3

L2 Cache

500 Mhz ATI GPU

48 3D Cores

Frame buffer

Video out

512 MB memory

controllers/ethernet/音频/DVD/等

Display (TV)
PS3

3.2 Ghz Cell

PPC Core

SPU0 SPU1 SPU2 SPU3

SPU4 SPU5 SPU6 SPU7

L2 Cache

SPU8

550 Mhz NVIDIA RSX GPU

Multiple 3D cores

Video out

256 MB Memory (XDR)

256 MB video Memory (GDDR3)

Display (TV)

controllers/ethernet/audio/DVD/etc.
Hybrid CPU–GPUs

Intel Sandybridge

Apple A5
Question 1 (short/long form)

Who is your partner?

or

Find a potential partner in the class, and send us their name/email