MIDTERM 1: MISSION COMPLETE

- Avg: 78%
- Median: 80%
- Max: 97%

- Have a look on Friday
LAST BITS ON PROJECTIONS

(Switch to maya demo)
ORTHOGONALIC =
VERY FAR PERSPECTIVE (WITH VERY GOOD ZOOM)
VIEW VOLUME
ORTHOGRAPHIC PROJECTION

• specifies field-of-view, used for clipping
• restricts domain of $z$ stored for visibility test
UNDERSTANDING Z

- z axis flip changes coord system handedness
- RHS before projection (eye/view coords)
- LHS after projection (clip, norm device coords)
Map FRUSTUM to the NDCS cube. Later z will be ignored.
Now we need to scale/translate/shear -> generic transform
CPSC 314
14 – CLIPPING & RASTERIZATION

UGRAD.CS.UBC.CA/~CS314

Mikhail Bessmeltsev
THE RENDERING PIPELINE

Verteces and attributes →

Vertex Shader
- Modelview transform
- Per-vertex attributes

→

Vertex Post-Processing
- Viewport transform
- Clipping

→

Rasterization
- Scan conversion
- Interpolation

→

Fragment Shader
- Texturing/...
- Lighting/shading

→

Per-Sample Operations
- Depth test
- Blending

→

Framebuffer
CLIPPING

• We have to clip what’s outside our view volume
• Outside to the left/right, top/bottom
• More importantly, front/near:
CLIPPING

• More importantly, front/near:
CLIPPING

• More importantly, front/near:
CLIPPING

• More importantly, front/near:
CLIPPING

- Where to do it in pipeline?
CLIPPING

- Option 1: Before projection
- Option 2: After NDCS
- Option 3: In between?
CLIPPING

• Option 1: Before projection
  • Then it would have to know all the camera info
• Option 2: After NDCS
• Option 3: In between?
CLIPPING

• **Option 1: Before projection**
  • Then it would have to know all the camera info

• **Option 2: After NDCS**
  • Flip already occurred
  • Too many calculations

• **Option 3: In between?**
CLIPPING

• Option 1: Before projection
  • Then it would have to know all the camera info

• Option 2: After NDCS
  • Flip already occurred
  • Too many calculations

• Option 3: In between?
CLIPPING

• Perform clipping in clip-coordinates!
  • After projection and before dividing by w
CLIPPING

• Perform clipping in clip-coordinates!
  • After projection and before dividing by $w$

  $$-w_c < x_c < w_c$$
  $$-w_c < y_c < w_c$$
  $$-w_c < z_c < w_c$$

We have not performed any divisions =>
  no flip; efficiency
CLIPPING: UNDER THE HOOD

• Creates new vertices
• Done automatically, we won’t study the actual algorithm
CLIPPING: UNDER THE HOOD

• Creates new vertices
• Done automatically, we won’t study the actual algorithm
• Clip:
  • Points -> discard
  • Triangles -> clip
CLIPPING COORDINATES

- Eye coordinates (projected) → clip coordinates → normalized device coordinates (NDCs)

- Dividing clip coordinates \((x_c, y_c, z_c, w_c)\) by the \(w_c (w_c = w_n)\) component (the fourth component in the homogeneous coordinates) yields normalized device coordinates (NDCs).

\[
\begin{bmatrix}
  x_n w_n \\
  y_n w_n \\
  z_n w_n \\
  w_n
\end{bmatrix} = \begin{bmatrix}
  x_c \\
  y_c \\
  z_c \\
  w_c
\end{bmatrix} = \begin{bmatrix}
  s_x & 0 & -c_x & 0 \\
  0 & s_y & -c_y & 0 \\
  0 & 0 & f + n & -2fn \\
  0 & 0 & f - n & f - n \\
\end{bmatrix} \begin{bmatrix}
  x_e \\
  y_e \\
  z_e \\
  1
\end{bmatrix}
\]
• We need a transform that maps the lower left corner to \([-0.5, -0.5]^t\) and upper right corner to \([W - 0.5, H - 0.5]^t\).

• The appropriate scale and shift can be done using the viewport matrix:

\[
\begin{pmatrix}
x_w \\
y_w \\
z_w \\
1
\end{pmatrix} =
\begin{bmatrix}
W/2 & 0 & 0 & (W-1)/2 \\
0 & H/2 & 0 & (H-1)/2 \\
0 & 0 & 1/2 & 1/2 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{pmatrix}
x_n \\
y_n \\
z_n \\
1
\end{pmatrix}
\]
THE RENDERING PIPELINE

Vertices and attributes

- **Vertex Shader**
  - Modelview transform
  - Per-vertex attributes

- **Rasterization**
  - Scan conversion
  - Interpolation

- **Per-Sample Operations**
  - Depth test
  - Blending

- **Fragment Shader**
  - Texturing/...
  - Lighting/shading

- **Vertex Post-Processing**
  - Viewport transform
  - Clipping

- **Framebuffer**
• This is part of the fixed function pipeline

• Input: all polygons are clipped

• Output: fragments (with \textit{varying variables} interpolated)
PATH FROM VERTEX TO PIXEL

Assembler

gl_Position
Varying variables

Rasterizer

Clipping
Divide-by-W
Culling
Viewport

Window coordinates
Varying variables

Varying variables
$z_w$
SHAPES - CURVES/SURFACES

- Mathematical representations:
  - Explicit functions
  - Parametric functions
  - Implicit functions
**SHAPES: EXPLICIT FUNCTIONS**

• Curves:  \[ y := \sin(x) \]
  
  • \( y \) is a function of \( x \):
  
  • Only works in 2D

• Surfaces:  \[ z := \sin(x) + \cos(y) \]
  
  • \( z \) is a function of \( x \) and \( y \):
  
  • Cannot define arbitrary shapes in 3D
SHAPES: PARAMETRIC FUNCTIONS

• Curves:
  • 2D: x and y are functions of a parameter value t
  • 3D: x, y, and z are functions of a parameter value t

\[ C(t) := \begin{pmatrix} \cos(t) \\ \sin(t) \\ t \end{pmatrix} \]
SHAPES: PARAMETRIC FUNCTIONS

• Surfaces:
  • Surface S is defined as a function of parameter values s, t
  • Names of parameters can be different to match intuition:

\[ S(\phi, \theta) := \begin{pmatrix} 
\cos(\phi) \cos(\theta) \\
\sin(\phi) \cos(\theta) \\
\sin(\theta) 
\end{pmatrix} \]
SHAPES: IMPLICIT

• Surface (3D) or Curve (2D) defined by zero set (roots) of function
  • E.g:

\[ S(x, y, z) : x^2 + y^2 + z^2 - 1 = 0 \]
Interactive graphics uses Polygons

- Can represent any surface with arbitrary accuracy
  - Splines, mathematical functions, ...
- Simple, regular rendering algorithms
  - embed well in hardware
POLYGONS

• Basic Types

- Simple convex
- Simple concave
- Non-simple (self-intersection)
FROM POLYGONS TO TRIANGLES

• why? triangles are always planar, always convex

• simple convex polygons
  • trivial to break into triangles

• concave or non-simple polygons
  • more effort to break into triangles
WHAT IS SCAN CONVERSION?
(A.K.A. RASTERIZATION)

• screen is discrete
one possible scan conversion
HOW TO CHECK IF A PIXEL IS INSIDE?
HOW TO CHECK IF A PIXEL IS INSIDE?

• Use implicit line equation:
  • $Ax + By + C = 0$

• How to find A,B,C?

• Orientation?
HOW TO CHECK IF A PIXEL IS INSIDE?

A point is inside $\Leftrightarrow$ $A_i x + B_i y + C > 0, i = 1, \ldots, 3$
HOW TO TREAT BOUNDARY?
HOW TO TREAT BOUNDARY?

• If two triangles share an edge, scan conversion should be consistent
  • No pixel drawn twice
  • No gaps

• Strategy ideas?
BONUS 2

• With the algorithm above, what’s the minimum number of pixels that will be drawn for the following triangle:

\[ P_1 = (0.5, 0.5) \]
\[ P_2 = (99.5, 100.5) \]
\[ P_3 = (-98.5, 100.5) \]

• Proof!

• 5 first solutions accepted; +5% to final grade
NAÏVE SCAN CONVERSION

• Testing every pixel is suboptimal
• Better ideas?
SCANLINE IDEA

• Basic structure of code:
  • Setup: compute edge equations, bounding box
  • (Outer loop) For each scanline in bounding box...
  • (Inner loop) ...check each pixel on scanline, evaluating edge equations and drawing the pixel if all three are positive
findBoundingBox(xmin, xmax, ymin, ymax);
setupEdges (a0, b0, c0, a1, b1, c1, a2, b2, c2);

for (int y = yMin; y <= yMax; y++) {
    for (int x = xMin; x <= xMax; x++) {
        float e0 = a0*x + b0*y + c0;
        float e1 = a1*x + b1*y + c1;
        float e2 = a2*x + b2*y + c2;
        if (e0 > 0 && e1 > 0 && e2 > 0)
            Image[x][y] = TriangleColor;
    }
}
TRIANGLE RASTERIZATION ISSUES

• Exactly which pixels should be lit?
• A: Those pixels inside the triangle edges
• What about pixels exactly on the edge?
TRIANGLE RASTERIZATION ISSUES

- Sliver

- Moving Slivers
HOW TO TEST IF A POINT IS IN A POLYGON?

simple convex

simple concave

non-simple (self-intersection)