Announcements

- Project 2 due Friday, October 8. Grading slot is now 2-4pm.
- Gradesource IDs should have been sent to everybody’s email addresses. Check Gradesource for scores. Scores may not be available until after late grading deadline.
- I need PID and email addresses from the following students, to give Gradesource access. Please see me after class, or send me email.
  - Kuttuva, Manjuladevi
  - Li, Yunjiu
  - Lorenz-Meyer, Vitus
  - Viswanathan, Ramesh
  - Ramos, Facundo
Lecture Overview

- Barycentric Coordinates
- Culling, Clipping
- Rasterization
- Visibility
Rendering Pipeline

Scene data → Modeling and Viewing Transformation → Shading → Projection → Rasterization, Visibility → Image

Lecture 2
Lectures 6-7
Lecture 3
Lecture 4 (today)
Implicit 2D Lines

- Given two 2D points \( a, b \)
- Define function \( f_{ab}(p) \) such that \( f_{ab}(p) = 0 \) if \( p \) lies on the line defined by \( a, b \)
Implicit 2D Lines

- **Point** $p$ **lies on the line**, if $p-a$ is **perpendicular** to the **normal** of the line 

$$ (a_y - b_y, b_x - a_x) $$

- **Use dot product** to determine on which side of the line $p$ lies. If $f(p) > 0$, $p$ is on **same side** as normal, if $f(p) < 0$ $p$ **lies on opposite side**. If dot product is 0, $p$ **lies on the line**.

$$ f_{ab}(p) = (a_y - b_y, b_x - a_x) \cdot (p_x - a_x, p_y - a_y) $$
Barycentric Coordinates

- Coordinates for 2D plane defined by triangle vertices \(a, b, c\)
- Any point \(p\) in the plane defined by \(a, b, c\) is
  \[p = a + \beta (b - a) + \gamma (c - a)\]
  \[= (1 - \beta - \gamma) a + \beta b + \gamma c\]
- We define \(\alpha = 1 - \beta - \gamma\)
  \[\Rightarrow p = \alpha a + \beta b + \gamma c\]
- \(\alpha, \beta, \gamma\) are called **barycentric** coordinates
- Works in 2D and in 3D
- If we imagine masses equal to \(\alpha, \beta, \gamma\) attached to the vertices of the triangle, the center of mass (the barycenter) is then \(p\). This is the origin of the term “barycentric” (introduced 1827 by Möbius)
Barycentric Coordinates

\[ \mathbf{p} = \mathbf{a} + \beta (\mathbf{b} - \mathbf{a}) + \gamma (\mathbf{c} - \mathbf{a}) \]

- \( \beta = 0 \)
- \( \beta = 1 \)
- \( \beta < 0 \)
- \( 0 < \beta < 1 \)
- \( \beta > 1 \)

- \( \mathbf{p} \) is inside the triangle if \( 0 < \alpha, \beta, \gamma < 1 \)
Barycentric Coordinates

- Problem: Given point $p$, find its barycentric coordinates
- Use equation for implicit lines
  \[
  \beta(p) = \frac{f_{ac}(p)}{f_{ac}(b)}
  \]
  \[
  \gamma(p) = \frac{f_{ab}(p)}{f_{ab}(c)}
  \]
- Division by zero if triangle is degenerate
  \[
  \alpha = 1 - \beta - \gamma
  \]
  \[
  0 < \beta < 1
  \]
Barycentric Interpolation

- Interpolate values across triangles, e.g., colors

- Linear interpolation on triangles

\[ c(p) = \alpha(p)c_a + \beta(p)c_b + \gamma(p)c_c \]
Lecture Overview

- Barycentric Coordinates
- Culling, Clipping
- Rasterization
- Visibility
Rendering Pipeline

Primitives

Modeling and Viewing
Transformation

Shading

Projection

Scan conversion, visibility

Image

Culling, Clipping
- Discard geometry that should not be drawn
Culling

- Discard geometry that does not need to be drawn as early as possible

- Two types of culling:
  - Object-level frustum culling
    - Later in class
  - Triangle culling
    - View frustum culling (clipping): outside view frustum
    - Backface culling: facing “away” from the viewer
    - Degenerate culling: area=0
Backface Culling

- Consider triangles as “one-sided”, i.e., only visible from the “front”

- Closed objects
  - If the “back” of the triangle is facing the camera, it is not visible
  - Gain efficiency by not drawing it (culling)
  - Roughly 50% of triangles in a scene are back facing
Backface Culling

- **Convention:** front side means vertices are ordered counterclockwise

- OpenGL allows one- or two-sided triangles
  - One-sided triangles:
    ```
    glEnable(GL_CULL_FACE); glCullFace(GL_BACK)
    ```
  - Two-sided triangles (no backface culling):
    ```
    glDisable(GL_CULL_FACE)
    ```
Backface Culling

- Compute triangle normal after projection (homogeneous division)
  \[ \mathbf{n} = (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0) \]

- Third component of \( \mathbf{n} \) negative: front-facing, otherwise back-facing
  - Remember: projection matrix is such that homogeneous division flips sign of third component
Degenerate Culling

- Degenerate triangle has no area
  - Vertices lie in a straight line
  - Vertices at the exact same place
  - Normal $\mathbf{n}=0$
View Frustum Culling, Clipping

- Triangles that intersect the faces of the view volume
  - Partly on screen, partly off screen
  - Do not rasterize the parts that are off-screen
- Traditional clipping
  - Split triangles that lie partly inside/outside viewing volume before homogeneous division
  - Avoid problems with division by zero
- Modern GPU implementations avoid clipping
Lecture Overview

- Barycentric Coordinates
- Culling, Clipping
- Rasterization
- Visibility
Rendering Pipeline

- Primitives
- Modeling and Viewing
  - Transformation
  - Shading
  - Projection
  - Rasterization, Visibility
- Image

- Scan conversion and rasterization are synonyms
- One of the main operations performed by GPU
- Draw triangles, lines, points (squares)
- Focus on triangles in this lecture
Rasterization
Rasterization

- How many pixels can a modern graphics processor draw per second?
Rasterization

- Rasterization is „hard-coded“ in the graphics card, cannot (currently) be modified by the software
- NVidia GeForce 480 GTX
  - 33.6 billion pixels per second (GPix/s)
  - Multiple of what the fastest CPU could do
Rasterization

- Many different algorithms
- Old style
  - Rasterize edges first
Rasterization

- Many different algorithms
- Old style
  - Rasterize edges first
  - Fill the spans (scan lines, scan conversion)
Rasterization

- Many different algorithms exist
- Old style
  - Rasterize edges first
  - Fill the spans (scan lines, scan conversion)
  - Requires clipping
  - Straightforward, but not used for hardware implementation today
Rasterization

- GPU rasterization today based on “Homogeneous Rasterization”
  
  http://www.ece.unm.edu/course/ece595/docs/olano.pdf


- Does not require full clipping, does not perform homogeneous division at vertices

- Today in class
  - Simpler algorithm based on barycentric coordinates
  - More sophisticated than old style algorithm
  - Easy to implement
  - Requires clipping
Rasterization

- Given vertices in pixel coordinates

\[ p' = DPC^{-1}M_p \]

- World space
- Camera space
- Clip space
- Image space

Pixel coordinates

\[ p' = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} \]

\[ \frac{x'}{w'} \quad \frac{y'}{w'} \]
Rasterization

- **Simple algorithm**
  
  ```
  compute bbox
  clip bbox to screen limits
  for all pixels \([x,y]\) in bbox
    compute barycentric coordinates \(\alpha, \beta, \gamma\)
    if \(0<\alpha, \beta, \gamma<1\) //pixel in triangle
      \(image[x,y]=triangleColor\)
  ```

- **Bounding box clipping trivial**
Rasterization

- So far, we compute barycentric coordinates of many useless pixels
- How can this be improved?
Rasterization

Hierarchy

• If block of pixels is outside triangle, no need to test individual pixels

• Can have several levels, usually two-level

• Find right granularity and size of blocks for optimal performance
2D Triangle-Rectangle Intersection

- If one of the following tests returns true, the triangle intersects the rectangle:
  - Test if any of the triangle’s vertices are inside the rectangle (e.g., by comparing the x/y coordinates to the min/max x/y coordinates of the rectangle)
  - Test if one of the quad’s vertices is inside the triangle (e.g., using barycentric coordinates)
  - Intersect all edges of the triangle with all edges of the rectangle
Rasterization

Where is the center of a pixel?

- Depends on conventions
- With our viewport transformation:
  - 800 x 600 pixels ⇔ viewport coordinates are in [0...800]x[0...600]
  - Center of lower left pixel is 0.5, 0.5
  - Center of upper right pixel is 799.5, 599.5
Rasterization

**Shared Edges**

- Each pixel needs to be rasterized exactly once
- Resulting image is independent of drawing order
- Rule: If pixel center exactly touches an edge or vertex
  - Fill pixel only if triangle extends to the right or down
Lecture Overview

- Barycentric Coordinates
- Culling, Clipping
- Rasterization
- Visibility
• At each pixel, we need to determine which triangle is visible
Painter’s Algorithm

- Paint from back to front
- Every new pixel always paints over previous pixel in frame buffer
- Need to sort geometry according to depth
- May need to split triangles if they intersect

- Old style, before memory became cheap
Z-Buffering

- Store z-value for each pixel
- Depth test
  - During rasterization, compare stored value to new value
  - Update pixel only if new value is smaller

```c
setpixel(int x, int y, color c, float z)
if(z<zbuffer(x,y)) then
  zbuffer(x,y) = z
  color(x,y) = c
```

- z-buffer is dedicated memory reserved for GPU (graphics memory)
- Depth test is performed by GPU
Rasterization

- What if a triangle’s vertex colors are different?
- Need to interpolate across triangle
- Naïve: linear interpolation
Barycentric Interpolation

- Interpolate values across triangles, e.g., colors

\[ c(p) = \alpha(p)a_c + \beta(p)b_c + \gamma(p)c_c \]

- Linear interpolation on triangles
  - Barycentric coordinates
Next Lecture

- Perspectively Correct Interpolation
- Everything about color