CSE 167: Introduction to Computer Graphics
Lecture #5: Rasterization

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2012
Announcements

- Homework project #2 due this Friday, October 12
  - To be presented starting 1:30pm in lab 260
  - Also present late submissions for project #1
Lecture Overview

- Culling
- Rasterization
- Visibility
- Barycentric Coordinates
Culling

- **Goal:**
  Discard geometry that does not need to be drawn to speed up rendering

- **Types of culling:**
  - View frustum culling
  - Occlusion culling
  - Small object culling
  - Backface culling
  - Degenerate culling
View Frustum Culling

- Triangles outside of view frustum are off-screen
  - Done on canonical view volume

Images: SGI OpenGL Optimizer Programmer's Guide
Videos

- Rendering Optimisations - Frustum Culling
  - http://www.youtube.com/watch?v=kvVHp9wMAO8&feature=related

- View Frustum Culling Demo
  - http://www.youtube.com/watch?v=bJrYTBGpwic
Bounding Box

- How to cull objects consisting of many polygons?
- Cull bounding box
  - Rectangular box, parallel to object space coordinate planes
  - Box is smallest box containing the entire object

*Image: SGI OpenGL Optimizer Programmer's Guide*
Occlusion Culling

- Geometry hidden behind occluder cannot be seen
  - Many complex algorithms exist to identify occluded geometry

Images: SGI OpenGL Optimizer Programmer’s Guide
Video

- Umbra 3 Occlusion Culling explained
  - http://www.youtube.com/watch?v=5h4QgDBwQhc
Small Object Culling

- Object projects to less than a specified size
  - Cull objects whose screen-space bounding box is less than a threshold number of pixels
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:**
  Triangle is front facing if 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 $n=0$

Rendering Pipeline

1. Primitives
2. Modeling and Viewing Transformation
3. Shading
4. Projection
5. Scan conversion, visibility
6. Image

Culling, Clipping
- Discard geometry that will not be visible
Lecture Overview

- Culling
- Rasterization
- Visibility
- Barycentric Coordinates
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?
How many pixels can a modern graphics processor draw per second?

- **NVidia GeForce GTX 690**
  - 234 billion pixels per second
  - Multiple of what the fastest CPU could do
Rasterization

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

- Many different algorithms
- Example:
  - Rasterize edges first
  - Fill the spans (scan lines)
- Disadvantage:
  - Requires clipping

Source: http://www.arcsynthesis.org
Rasterization

- GPU rasteriazation today based on “Homogeneous Rasterization”
  
  [link](http://www.ece.unm.edu/course/ece595/docs/olano.pdf)

Rasterization

- Given vertices in pixel coordinates
  \[ p' = DPC^{-1}Mp \]
  - World space
  - Camera space
  - Clip space
  - Image space
  - Pixel coordinates: \( \frac{x'}{w'}, \frac{y'}{w'} \)
**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**

![Diagram of a triangle with bounding box]
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
Where is the center of a pixel?

- Depends on conventions
- With our viewport transformation:
  - 800 x 600 pixels $\Leftrightarrow$ 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

- Culling
- Rasterization
- Visibility
- Barycentric Coordinates
• 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

- Outdated algorithm, created when memory was expensive
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
    ```
    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
Z-Buffering

- Problem: translucent geometry
  - Storage of multiple depth and color values per pixel (not practical in real-time graphics)
  - Or back to front rendering of translucent geometry, after rendering opaque geometry
Lecture Overview

- Culling
- Rasterization
- Visibility
- Barycentric Coordinates
Color Interpolation

- What if a triangle’s vertex colors are different?
- Need to interpolate across triangle
  - How to calculate interpolation weights?
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 $\mathbf{p}$ lies on the line, if $\mathbf{p}-\mathbf{a}$ 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 $\mathbf{p}$ lies. If $f(\mathbf{p})>0$, $\mathbf{p}$ is on same side as normal, if $f(\mathbf{p})<0$ $\mathbf{p}$ is on opposite side. If dot product is 0, $\mathbf{p}$ lies on the line.

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

\[ c(p) = \alpha(p)c_a + \beta(p)c_b + \gamma(p)c_c \]
Barycentric Coordinates

- Demo Applet:
  - http://www.ccs.neu.edu/home/suhail/BaryTriangles/applet.htm