CSE167
Introduction to Computer Graphics

Matthias Zwicker
University of California, San Diego
Spring 2007

Projects

- Grade report for project 1 at http://www.gradesource.com/reports/2652/12704/index.html
- You got secret key via email
- Project 2 due Wednesday October 17
- Go to lab session this Friday

Hint for project 2

- Explained in class how to compute camera-to-world matrix \( C \)
- In code will need world-to-camera matrix \( C^{-1} \)
- Easy to compute
  - Camera-to-world matrix can be split into two factors
    \( C = TR \)
  - Rotation matrix \( R \) uses top left 3x3 elements of \( C \)
  - Translation matrix \( T \) uses last column of \( C \)
  - Inverse
    \[ C^{-1} = R^{-1}T^{-1} \]

Today

- Review
- Drawing triangles
  - Barycentric coordinates
  - Culling, clipping
  - Rasterization
  - Visibility

Review

- Rendering pipeline
- Perspective projection

Rendering pipeline

- Simplified version
- Most operations performed by specialized hardware (GPU)
- Access to hardware through low-level 3D API (DirectX, OpenGL)
- State machine
Perspective projection

- Given 3D points (vertices) in camera coordinates, determine corresponding image coordinates
- Things farther away seem smaller
- Simplified model of human eye, or camera lens (pinhole camera)

The math: simplified case

\[ y' = \frac{y_1 d}{z_1} \]
\[ z' = d \]
**Perspective projection**

The math: simplified case

\[
y' = \frac{y}{z}d
\]

\[
z' = d
\]

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1/d & 0
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
z/d
\end{bmatrix}
= \begin{bmatrix}
x' \\
y' \\
z' \\
z/d'
\end{bmatrix}
= \begin{bmatrix}
xd/z \\
yd/z \\
z/d \\
1
\end{bmatrix}
\]

Homogeneous division

**Perspective view volume**

- Defines 3D volume that is mapped to image
- Left, right, top, bottom boundaries
- Near, far clipping planes
  - Avoid numerical problems during rendering
  - Divide by zero
  - Low precision for distant objects

**Projection matrix**

Camera coordinates

Projection matrix

Canonic view frustum

Clipping

**Perspective projection matrix**

\[P_{\text{pers}}(\text{left}, \text{right}, \text{top}, \text{bottom}, \text{near}, \text{far}) =
\begin{bmatrix}
\frac{2\text{near}}{\text{right}-\text{left}} & 0 & \frac{\text{right}+\text{left}}{\text{right}-\text{left}} & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & \frac{\text{top}-\text{bottom}}{\text{near}+\text{far}} & \frac{\text{far}+\text{near}}{\text{near}+\text{far}} \\
0 & 0 & \frac{\text{far}-\text{near}}{\text{far}+\text{near}} & 1
\end{bmatrix}
\]

**Questions?**

- Review
- Drawing triangles
  - Barycentric coordinates
  - Culling, clipping
  - Rasterization
  - Visibility

**Today**

- Review
- Drawing triangles
  - Barycentric coordinates
  - Culling, clipping
  - Rasterization
  - Visibility
Implicit 2D lines

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

\[
f_{ab}(p) = 0 \\
f_{ab}(p) \neq 0
\]

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
  \[
p = a + \beta(b - a) + \gamma(c - a) \\
  = (1 - \beta - \gamma)a + \beta b + \gamma c
\]
  where \( a = 1 - \beta - \gamma \)
- \( \alpha, \beta, \gamma \) are called barycentric coordinates
- Works in 2D and in 3D

Barycentric coordinates

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

Barycentric coordinates

- Points on triangle edges, e.g.,
  \[
p(t) = a(1 - t) + bt
  \]
- Barycentric coordinates correspond to linear interpolation weights
  \[
  \alpha(p(t)) = 1 - t \\
  \beta(p(t)) = t
  \]
- Same for other edges
- Barycentric can be used to generalize linear interpolation to triangles
Barycentric interpolation

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

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

- Linear interpolation on triangles

Barycentric coordinates

Today

- Review
- Drawing triangles
  - Barycentric coordinates
  - Culling, clipping
  - Rasterization
  - Visibility

Rendering pipeline

Primitives

Modeling and viewing transformation

Shading

Projection

Scan conversion, visibility

Culling, clipping
  - Discard geometry that should not be drawn

Image

Culling

- Discard geometry that does not need to be drawn as early as possible
- 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 the triangles as “one-sided”, i.e. only visible from the “front”
- If the “back” of the triangle is facing the camera, it can be culled
- Roughly 50% of triangles in a scene are back facing
**Backface culling**
- Convention: front side means vertices are ordered counterclockwise

![Diagram of backface culling](image)

- Most renderers allow one- or two-sided triangles
  - Two-sided triangles not backface culled
  - Thin objects, non-closed objects

**Backface culling**
- Compute triangle normal after projection (homogeneous division)
  \[ n = (p_1 - p_0) \times (p_2 - p_0) \]
- Third component of \( n \) negative: frontfacing, otherwise backfacing
- (Remember: 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 \( n = 0 \)

**View frustum culling, clipping**
- Triangles that intersect the faces of the view volume
  - Partly on screen, partly off
  - Do not rasterize the parts that are offscreen
- Traditional clipping
  - Split triangles that lie partly inside/outside viewing volume before homogeneous division
  - Avoid problems with division by zero

**Clipping**
- View volume defined by 6 planes
- A single triangle might intersect any of them
- Compute intersections with each plane in turn

**Clipping**
For each plane
- All 3 vertices "inside": no clipping
- All 3 vertices "outside": cull triangle
- One vertex "inside": clip
  - Create two vertices on plane, one new triangle
- Two vertices are "inside": clip:
  - Create two vertices on the plane, two new triangles
Questions?

Today

- Review
- **Drawing triangles**
- Barycentric coordinates
- Culling, clipping
- **Rasterization**
- Visibility

Rendering pipeline

Rasterization

- 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

Primitives

Modeling and viewing transformation

- Shading
- Projection
- Rasterization, visibility

Image

Rasterization

- How many pixels can a modern graphics processor draw per second?
- Rasterization is „hard-coded“, cannot be modified per software
- NVidia Geforce 8800 GTX
  - Theoretical peak: up to 14 billion pixels per second
  - 600MHz clock frequency, 24 pixels per clock
  - Multiple of what the fastest CPU could do
Rasterization

- Many different algorithms
- Old style
  - Rasterize edges first
  - Fill the spans (scan lines, scan conversion)
  - Requires clipping
  - Not preferred for hardware implementation today

Rasterization

- Many different algorithms
- Old style
  - Rasterize edges first
  - Fill the spans (scan lines, scan conversion)
- Requires clipping
- Not preferred for hardware implementation today

Today in class
- Simpler algorithm based on barycentric coordinates
- Easy to implement
- Technically, requires clipping

Rasterization

- Given vertices in pixel coordinates
  \[
  \mathbf{p}' = \mathbf{D} \mathbf{P} \mathbf{C} \mathbf{M}_p
  \]
  
  World space
  Camera space
  Clip space
  Image space

  Pixel coordinates \( \mathbf{p} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} \)

  \[
  x'/w' \quad y'/w'
  \]

Rasterization

- GPU rasteriazation today based on “homogeneous rasterization”
  
  [http://www.ece.unm.edu/course/ece595/docs/olano.pdf](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
  - Easy to implement
  - Technically, requires clipping

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
So far, we compute barycentric coordinates of many useless pixels. Improvement?

If block of pixel is outside triangle, no need to test individual pixels. Can have several levels, usually two-level. Find right granularity, size of blocks, for best performance.

Where is the center of a pixel? Depends on conventions. With our viewport transformation from last lecture:
- 800 x 600 pixels $\Rightarrow$ viewport coordinates are in $[0...800] \times [0...600]$
- Center of lower left pixel is 0.5, 0.5
- Center of upper right pixel is 799.5, 599.5

Each pixel needs to be rasterized exactly once. Result 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.

At each pixel, need to determine which triangle is visible.
**Painter’s algorithm**
- Paint from back to front
- Every new pixel always paints over previous pixel
- 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
    ```
    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

---

**Next time**
- Perspective correct interpolation
- Color