## CSE 167: <br> Introduction to Computer Graphics Lecture \#5: Rasterization

Jürgen P. Schulze, Ph.D.<br>University of California, San Diego Spring Quarter 2015

## Announcements

- Project 3 due this Friday at Ipm
- Grading starts at 12:I5 in CSE labs 260+270


## Lecture Overview

- Barycentric Coordinates


## Color Interpolation



Source: efg's computer lab

- 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 $\mathbf{a}, \mathbf{b}$
- Define function $f_{\mathbf{a b}}(\mathbf{p})$ such that $f_{\mathbf{a b}}(\mathbf{p})=0$ if $\mathbf{p}$ lies on the line defined by $\mathbf{a}, \mathbf{b}$



## Implicit 2D Lines

- Point $\mathbf{p}$ lies on the line, if $\mathbf{p}$-a is perpendicular to the normal $\mathbf{n}$ of the line

$$
n=\left(a_{y}-b_{y} b_{x}-a_{x}\right) \quad p=\left(p_{x}-a_{x}, p_{y}-a_{y}\right)
$$

- Use dot product to determine on which side of the line $\mathbf{p}$ lies. If $f(p)>0, p$ 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.

$$
f_{\mathbf{a b}}(\mathbf{p})=\left(a_{y}-b_{y}, b_{x}-a_{x}\right) \cdot\left(p_{x}-a_{x}, p_{y}-a_{y}\right)
$$

## Barycentric Coordinates

- Coordinates for 2D plane defined by triangle vertices $\mathbf{a}, \mathbf{b}, \mathbf{c}$
- Any point $\mathbf{p}$ in the plane defined $\mathbf{b y} \mathbf{a}, \mathbf{b}, \mathbf{c}$ is $\mathbf{p}=\mathbf{a}+\beta(\mathbf{b}-\mathbf{a})+\gamma(\mathbf{c}-\mathbf{a})$
- Solved for $\mathrm{a}, \mathrm{b}, \mathrm{c}$ :
$\mathbf{p}=(\mathrm{I}-\beta-\gamma) \mathbf{a}+\beta \mathbf{b}+\gamma \mathbf{c}$

- We define $\alpha=\mathrm{I}-\beta-\gamma$
$\Rightarrow \mathbf{p}=\alpha \mathbf{a}+\beta \mathbf{b}+\gamma \mathbf{c}$
- $\alpha, \beta, \gamma$ are called barycentric coordinates
- If we imagine masses equal to $\alpha, \beta, \gamma$ in the locations of 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 Interpolation

- Interpolate values across triangles, e.g., colors
- Done by linear interpolation on triangle:


$$
c(\mathbf{p})=\alpha(\mathbf{p}) c_{\mathbf{a}}+\beta(\mathbf{p}) c_{\mathbf{b}}+\gamma(\mathbf{p}) c_{\mathbf{c}}
$$

- Works well at common edges of neighboring triangles


## Barycentric Coordinates

## - Demo:

b http://adrianboeing.blogspot.com/2010/01/barycentric-coordinates.html


## Lecture Overview

- Rendering Pipeline


## Rendering Pipeline



- Hardware and software which draws 3D scenes on the screen
- Consists of several stages - Simplified version here
- Most operations performed by specialized hardware (GPU)
- Access to hardware through low-level 3D API (OpenGL, DirectX)
- All scene data flows through the pipeline at least once for each frame


## Rendering Pipeline

Scene data

, Textures, lights, etc.

- Geometry
, Vertices and how they are connected
- Triangles, lines, points, triangle strips
- Attributes such as color
- Specified in object coordinates
- Processed by the rendering pipeline one-by-one


Image

## Rendering Pipeline



- Transform object to camera coordinates
- Specified by

GL_MODELVIEW matrix in OpenGL

- User computes

GL_MODELVIEW matrix as discussed

$$
\mathbf{p}_{\text {camera }}=\underset{\substack{\text { MODELVIEW } \\ \text { matrix }}}{\mathbf{C}^{-1} \mathbf{M} \mathbf{p}_{\text {object }}}
$$

## Rendering Pipeline



- Look up light sources
- Compute color for each vertex


## Rendering Pipeline

Scene data
 transformation

Shading
Projection


Rasterization, visibility

Image

- Project 3D vertices to 2D image positions
- GL_PROJECTION matrix


## Rendering Pipeline

Scene data

transformation

Shading
Projection
Rasterization, visibility

Image

- Draw primitives (triangles, lines, etc.)
- Determine what is visible



## Rendering Pipeline

Scene data


- Pixel colors


## Rendering Engine



Rendering Engine:

- Additional software layer encapsulating low-level API
- Higher level functionality than OpenGL
- Platform independent
- Layered software architecture common in industry
- Game engines
, Graphics middleware


## Lecture Overview

- Rasterization
- Visibility
- Shading


## Rendering Pipeline

Primitives


- 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

- Given vertices in pixel coordinates

$$
\begin{gathered}
\mathbf{p}^{\prime}=\left|\mathbf{D P} \mathbf{C}^{-1}\right| \begin{array}{l}
\mathbf{M} \\
\text { World space }
\end{array} \\
\mathbf{p}^{\prime}=\left[\begin{array}{l}
x^{\prime} \\
y^{\prime} \\
z_{\text {Clip space }}^{\prime} \\
w^{\prime}
\end{array}\right] \quad \text { Pixel coordinates } \begin{array}{ll}
\text { Image space } & x^{\prime} / w^{\prime} \\
y^{\prime} / w^{\prime}
\end{array} \\
\end{gathered}
$$

## Rasterization

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


## Rasterization

- How many pixels can a modern graphics processor draw per second?
- NVidia GeForce GTX 780
- 160 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 rasterization today based on "Homogeneous Rasterization"


## http://www.ece.unm.edu/course/ece595/docs/olano.pdf

Olano, Marc and Trey Greer, "Triangle Scan Conversion Using 2D Homogeneous Coordinates", Proceedings of the I997 SIGGRAPH/Eurographics Workshop on Graphics Hardware (Los Angeles, CA, August 2-4, 1997),ACM SIGGRAPH, New York, 1995.

## Rasterization

- Given vertices in pixel coordinates

$$
\begin{gathered}
\mathbf{p}^{\prime}=\left|\mathbf{D P} \mathbf{C}^{-1}\right| \begin{array}{l}
\mathbf{M} \\
\text { World space }
\end{array} \\
\mathbf{p}^{\prime}=\left[\begin{array}{l}
x^{\prime} \\
y^{\prime} \\
z_{\text {Clip space }}^{\prime} \\
w^{\prime}
\end{array}\right] \quad \text { Pixel coordinates } \begin{array}{ll}
\text { Image space } & x^{\prime} / w^{\prime} \\
y^{\prime} / w^{\prime}
\end{array} \\
\end{gathered}
$$

## Rasterization

- Simple algorithm

```
    compute b.box
    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

