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

CLIPPING
- Option 1: Before projection
  - Then it would have to know all the camera info
  - Flip already occurred
  - Too many calculations
- Option 3: In between?

CLIPPING
- Option 1: Before projection
- Option 2: After NDCS
  - Flip already occurred
  - Too many calculations
- Option 3: In between?

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

UNDERSTANDING Z
- Z axis flip changes coordinate system handedness
- RHS before projection (eye/view coords)
- LHS after projection (clip, norm device coords)

CLIPPING COORDINATES
- Eye coordinates (projected) \( \rightarrow \) clip coordinates \( \rightarrow \) normalized device coordinates (NDCs)
- Dividing clip coordinates \((x', y', z', w')\) by the \(w\) component (the fourth component in the homogeneous coordinates) yields normalized device coordinates (NDCs)

CLIPPING
- We need a transform that maps the lower left corner to \([-0.5, -0.5]\) and upper right corner to \([W / 2, H / 2]\)
- The appropriate scale and shift can be done using the viewport matrix:

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
w'
\end{bmatrix} = \frac{1}{w} \begin{bmatrix} w/2 & 0 & 0 & (W-1)/2 \\ 0 & H/2 & 0 & (H-1)/2 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x \\
y \\
z \\
w \end{bmatrix}
\]

THE RENDERING PIPELINE
- Vertex Shader
  - Modelview transform
  - Per-vertex attributes
- Rasterization
  - Scan conversion
  - Interpolation
- Viewport Transform
- Vertex Post-Processing
  - Texture/.. Lighting/shading
- Clip:
  - After projection and before dividing by \(w\)
- Fragment Shader
  - Texturing/..
- Per-Sample Operations
  - Depth test
  - Blending
- Framebuffer
**Rasterization**
- This is part of the fixed function pipeline
- Input: all polygons are clipped
- Output: fragments (with varying variables interpolated)

**Path from Vertex to Pixel**
- **Clipping**
- **Viewport transform**
- **Scan conversion**
- **Fragment shader**
- **Texturing/...**
- **Lighting/shading**
- **Depth test**
- **Blending**

**How to Treat Boundary?**
- If two triangles share an edge, scan conversion should be consistent
- No pixel drawn twice
- No gaps
- Strategy ideas?

**How to Test if a Point is in a Polygon?**
- Simple
- Convex
- Simple
- Concave
- Non-
  - Simple
    - Self-intersection

**From Polygons to Triangles**
- Why? Triangles are always planar, always convex
- Simple convex polygons
- Tried 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

**How to Check if a Pixel is Inside?**
- Use implicit line equation:
  - \( A_x + B_y + C = 0 \)
  - What is geometric meaning of \( A, B, C ? \)
  - \( A, B \) is a normal (not unit!) to the line
  - \( C \) is translation of that line
- How to find \( A, B, C? \)
  - Option 1. Solve a system of 2 equations
  - Option 2. Find any normal
- Orientation?
  - Normal points in positive side

**How to Check if a Pixel is Inside?**
A point is inside if:
\[ A_i x + B_i y + C > 0, \quad i = 1, \ldots, 3 \]

**How to Treat Boundary?**
- If two triangles share an edge, scan conversion should be consistent
- No pixel drawn twice
- No gaps
- Strategy ideas?

**Naive Scan Conversion**
- Testing every pixel is suboptimal
- No pixel drawn twice
- Better ideas?

**Less Naive Scan Conversion**
- Go over each pixel in bounding rectangle
- Check if pixel is inside/outside of triangle
- Use sign of edge equations

**Scanline Idea (Simplified)**
- Basic structure of code:
  - Setup: compute edge equations, bounding box
  - \( \text{for} \) each scanline in bounding box...
  - \( \text{for} \) each pixel on scanline, evaluating edge equations and drawing the pixel if all three are positive
**ALIASING & ANTI-ALIASING**

**SIMPLER:**
How to interpolate color between two points?

\[ c(t) = c(0) \cdot (1-t) + c(1) \cdot t \]

**Linear interpolation**

**Simple generalization:**
**Bi-linear interpolation**

\[ \begin{align*}
| & a_0 & b_0 & c_0 & d_0 \\ & a_1 & b_1 & c_1 & d_1 \\ & a_2 & b_2 & c_2 & d_2 \\ \end{align*} \]

\[ P(x,y) = \frac{1}{|N|} \left( a_0 \cdot x + b_0 \cdot y + c_0 \right) + \frac{1}{|N|} \left( a_1 \cdot x + b_1 \cdot y + c_1 \right) + \frac{1}{|N|} \left( a_2 \cdot x + b_2 \cdot y + c_2 \right) \]

**VALUES IN THE INTERIOR**

**Barycentric coordinates**

**INTERPOLATION - ACCESS TRIANGLE INTERIOR**

- Interpolate between vertices:
  - \( x,y \) - colour components
  - \( u,v \) - texture coordinates
  - \( N_1, N_2, N_3 \) - surface normals
- Equivalent:
  - Barycentric coordinates
  - Bi-linear interpolation
  - Plane interpolation
**Barycentric Coordinates**

- **Area**
  \[ A = \frac{1}{2} |\mathbf{P}_1 \times \mathbf{P}_2| \]

- **Barycentric coordinates**
  \[ a_1 = \frac{A_{\mathbf{P}_2 \mathbf{P}_3}}{A}, \quad a_2 = \frac{A_{\mathbf{P}_3 \mathbf{P}_1}}{A}, \quad a_3 = \frac{A_{\mathbf{P}_1 \mathbf{P}_2}}{A} \]
  \[ P = a_1 \mathbf{P}_1 + a_2 \mathbf{P}_2 + a_3 \mathbf{P}_3 \]

**NOTE:**
- In reality, only two values are enough to encode a point in a triangle.
- We added a 3rd value – a similar idea to homogeneous coordinates!
- Those are, however, unique because of this:
  \[ a_1 + a_2 + a_3 = 1 \]