CS380: Computer Graphics
Triangle Rasterization

Sung-Eui Yoon
(윤성의)

Course URL:
http://sgvr.kaist.ac.kr/~sungeui/CG/

Class Objectives (Ch. 7)

- Understand triangle rasterization using edge-equations
- Understand mechanics for parameter interpolations
- Realize benefits of incremental algorithms

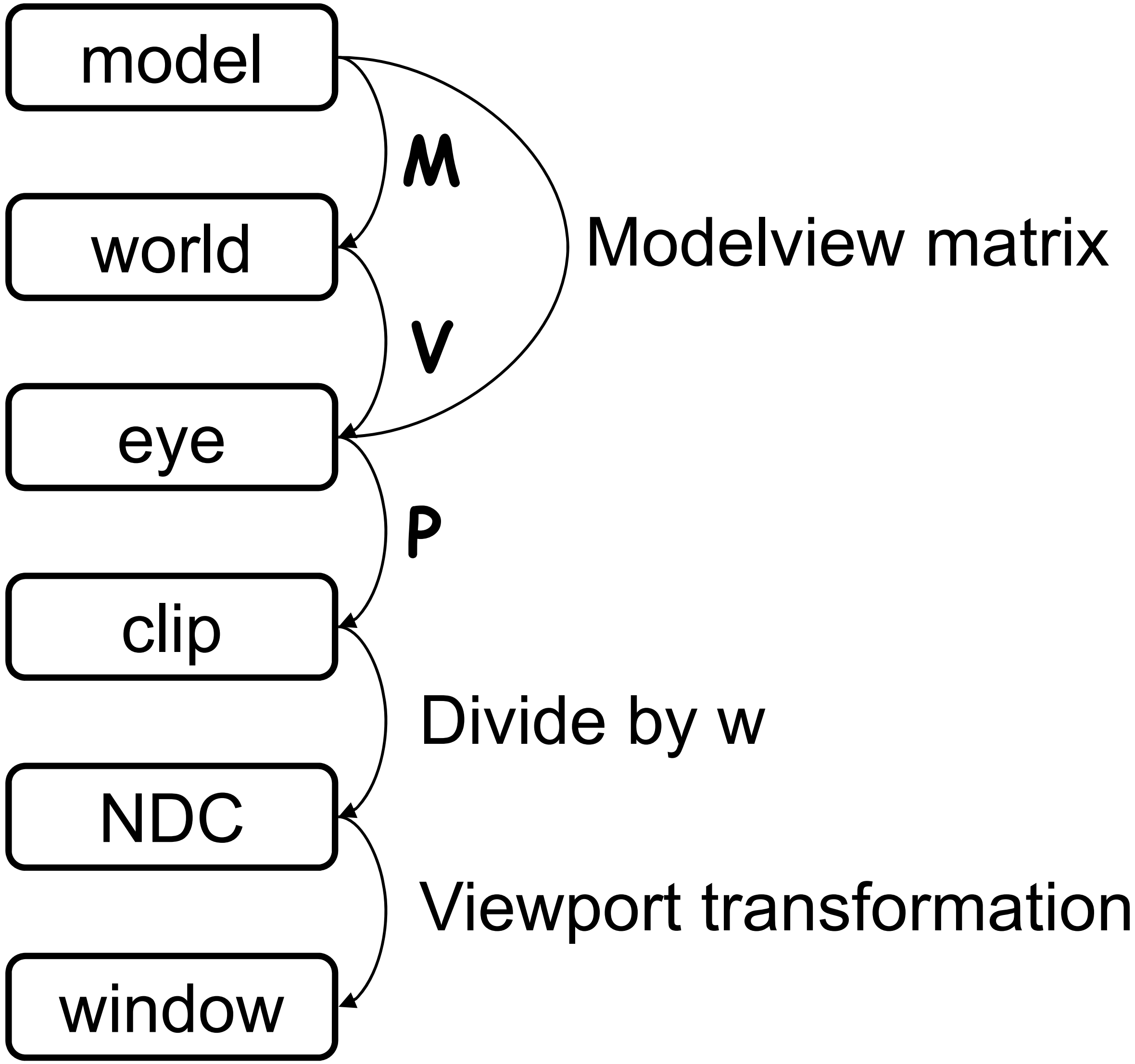
- At the last class:
  - Discussed clipping and culling methods of view-frustum, back-face, and hierarchical culling methods

Questions

- How do we apply clipping and culling when there are transparent parts are included in objects? .. virtual optical lenses to obtain realistic view?

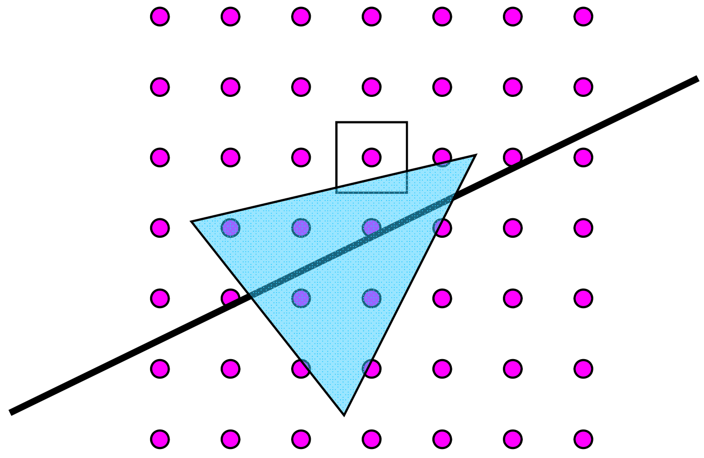
- I thought GPUs are exploited most only when extensive multi-threading is used. But up till now there doesn't seem to be any multithreading in the src codes in the lecture/homework materials.

Coordinate Systems



Primitive Rasterization

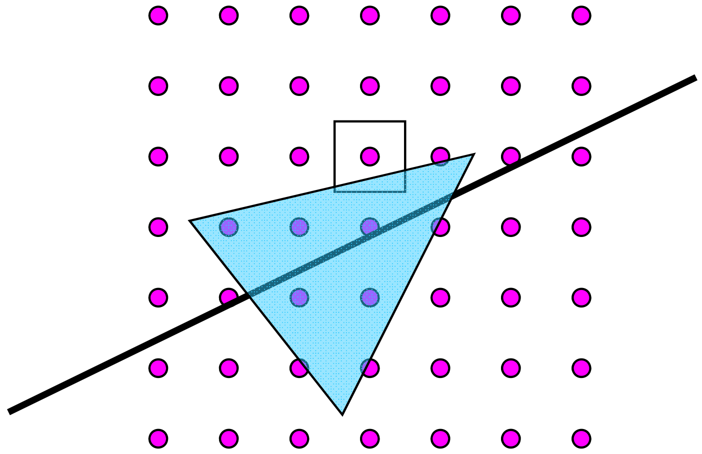
- Rasterization converts vertex representation to pixel representation



- Coverage determination
  - Computes which pixels (samples) belong to a primitive
- Parameter interpolation
  - Computes parameters at covered pixels from parameters associated with primitive vertices

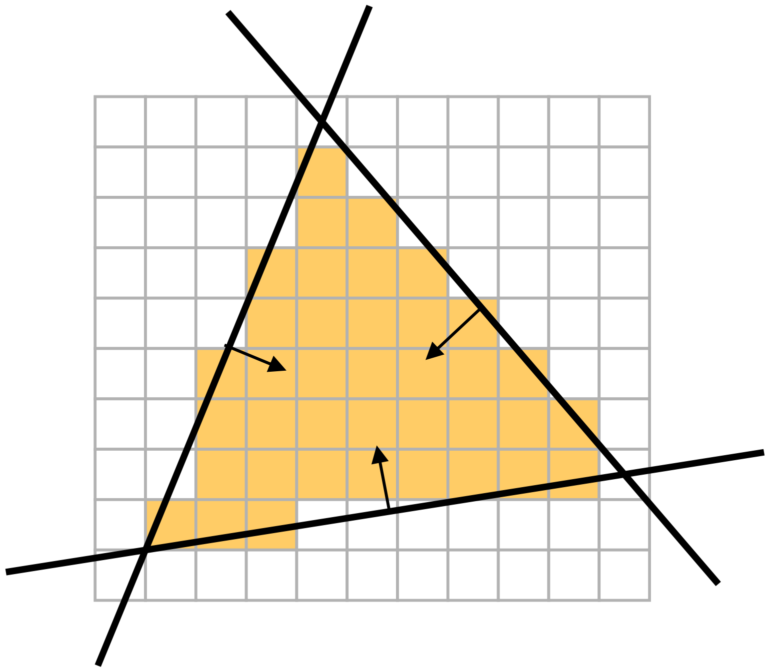
Coverage Determination

- **Coverage is a 2D sampling problem**
  - **Commonly reduced to 1D problem of checking a sample point**
- **Possible coverage criteria:**
  - **Distance of the primitive to sample point (often used with lines)**
  - **Percent coverage of a pixel (used to be popular)**
  - **Sample is inside the primitive (assuming it is closed)**



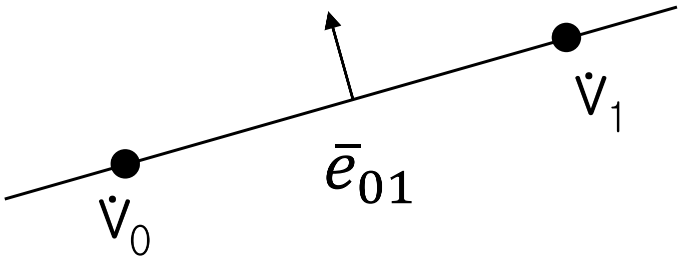
Rasterizing with Edge Equations

- Compute edge equations from vertices
- Compute interpolation equations from vertex parameters
- Traverse pixels evaluating the edge equations
- Draw pixels for which all edge equations are positive
- Interpolate parameters at pixels



Edge Equation Coefficients

- **The cross product between 2 homogeneous points generates the line between them**



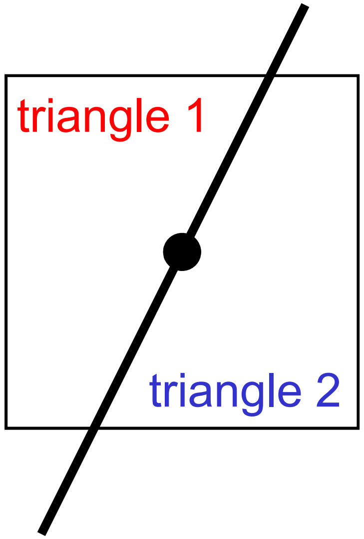
$$
\begin{aligned}
\bar{e} &= \dot{v}_0 \times \dot{v}_1 \\
&= [x_0 \quad y_0 \quad 1]^t \times [x_1 \quad y_1 \quad 1]^t \\
&= [\underset{A_{01}}{(y_0 - y_1)} \quad \underset{B_{01}}{(x_1 - x_0)} \quad \underset{C_{01}}{(x_0 y_1 - x_1 y_0)}]
\end{aligned}
$$

$$E(x, y) = \bar{e}_{01}(x, y) = A_{01}x + B_{01}y + C_{01}$$

- **A pixel at (x,y) is "inside" an edge if E(x,y)>0**

Shared Edges

- **Suppose two triangles share an edge. Which covers the pixel when the edge passes through the sample (E(x,y)=0)?**

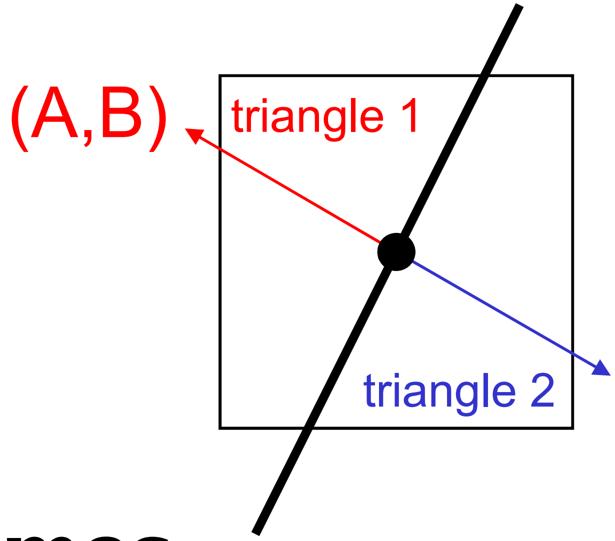


- **Both**
  - **Pixel color becomes dependent on order of triangle rendering**
  - **Creates problems when rendering transparent objects - "double hitting"**
- **Neither**
  - **Missing pixels create holes in otherwise solid surface**
- **We need a consistent tie-breaker!**

Shared Edges

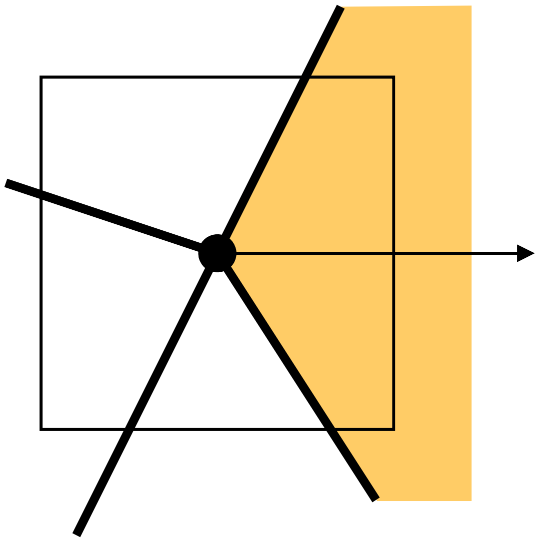
- **A common tie-breaker:**

$$\text{bool } t = \begin{cases} A > 0 & \text{if } A \neq 0 \\ B > 0 & \text{otherwise} \end{cases}$$

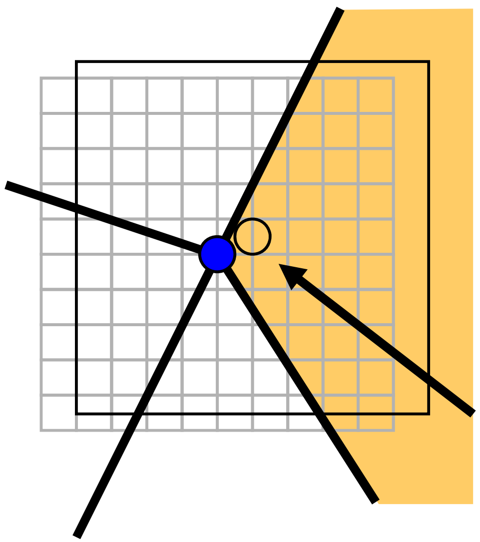


- **Coverage determination becomes**
  if(E(x,y) >0 || (E(x,y)==0 && t))
  pixel is covered

Shared Vertices



- **Use "inclusion direction" as a tie breaker**
- **Any direction can be used**



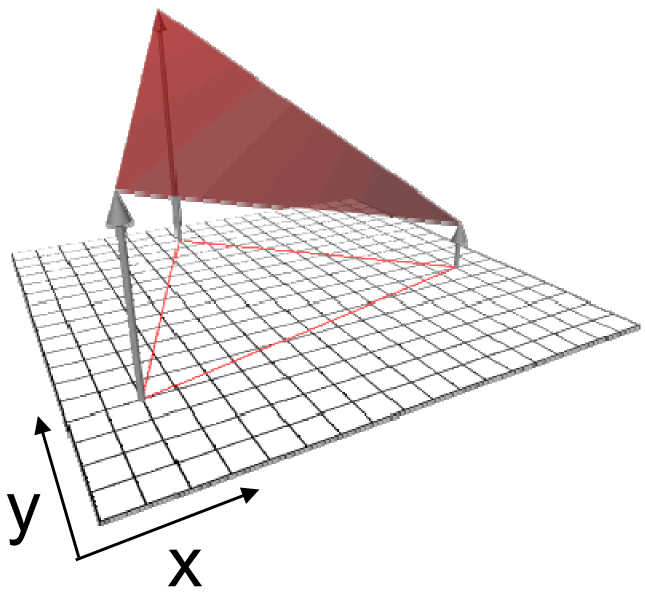
Pixel center

Snapped vertex

- **Snap vertices to subpixel grid and displace so that no vertex can be at the pixel center**

Interpolating Parameters

- **Specify a parameter, say redness (r) at each vertex of the triangle**
  - **Linear interpolation creates a planar function**



$$r(x,y) = A_r x + B_r y + C_r$$

Solving for Linear Interpolation Equations

- **Given the redness of the three vertices, we can set up the following linear system:**

$$\begin{bmatrix} r_0 & r_1 & r_2 \end{bmatrix} = \begin{bmatrix} A_r & B_r & C_r \end{bmatrix} \begin{bmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{bmatrix}$$

with the solution:

$$\begin{bmatrix} A_r & B_r & C_r \end{bmatrix} = \begin{bmatrix} r_0 & r_1 & r_2 \end{bmatrix} \frac{\begin{bmatrix} (y_1 - y_2) & (x_2 - x_1) & (x_1 y_2 - x_2 y_1) \\ (y_0 - y_2) & (x_2 - x_0) & (x_0 y_2 - x_2 y_0) \\ (y_0 - y_1) & (x_1 - x_0) & (x_0 y_1 - x_1 y_0) \end{bmatrix}}{\det \begin{bmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{bmatrix}}$$

Triangle Area

$$\text{Area} = \tfrac{1}{2}\det\begin{bmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \tfrac{1}{2}((x_1 y_2 - x_2 y_1) - (x_0 y_2 - x_2 y_0) + (x_0 y_1 - x_1 y_0))$$

$$= \tfrac{1}{2}(C_0 + C_1 + C_2) \quad \text{// they are from edge equations}$$

- **Area = 0 means that the triangle is not visible**
- **Area < 0 means the triangle is back facing:**
  - **Reject triangle if performing back-face culling**
  - **Otherwise, flip edge equations by multiplying by -1**

Interpolation Equation

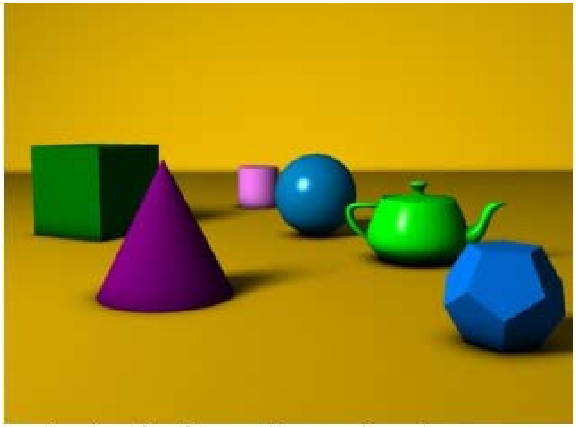
- **The parameter plane equation is just a linear combination of the edge equations**

$$[A_r \quad B_r \quad C_r] = \frac{1}{2 \cdot area}[r_0 \quad r_1 \quad r_2]\begin{bmatrix} \overline{e_0} \\ \overline{e_1} \\ \overline{e_2} \end{bmatrix}$$

$\overline{e_o}, \overline{e_1}, \overline{e_2}$ are vectors of edge equations

Z-Buffering

- When rendering multiple triangles we need to determine which triangles are visible
- Use z-buffer to resolve visibility
  - Stores the depth at each pixel
- Initialize z-buffer to 1 (far value)
  - Post-perspective z values lie between 0 and 1
- Linearly interpolate depth (z_{tri}) across triangles
- If $z_{tri}(x,y) < zBuffer[x][y]$
  write to pixel at (x,y)
  $zBuffer[x][y] = z_{tri}(x,y)$

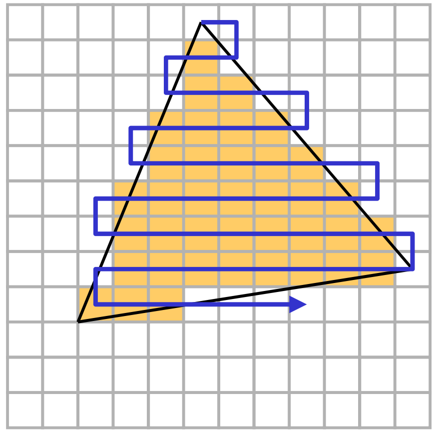

A simple three dimensional scene


Z-buffer representation

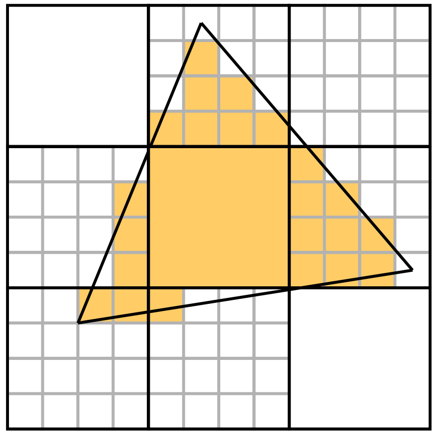
image from wikipedia.com

Traversing Pixels

- **Free to traverse pixels**
  - **Edge and interpolation equations can be computed at any point**



- **Try to minimize work**
  - **Restrict traversal to primitive bounding box**
  - **Hierarchical traversal**
    - **Knock out tiles of pixels (say 4x4) at a time**



Incremental Algorithms

- **Some computation can be saved by updating the edge and interpolation equations incrementally:**

$$
\begin{aligned}
E(x,y) &= Ax + By + C \\
E(x+\Delta, y) &= A(x+\Delta) + By + C \\
&= E(x,y) + A \cdot \Delta \\
E(x, y+\Delta) &= Ax + B(y+\Delta) + C \\
&= E(x,y) + B \cdot \Delta
\end{aligned}
$$

- **Equations can be updated with a single addition!**

Triangle Setup

- Compute edge equations
  - 3 cross products
- Compute triangle area
  - A few additions
- Cull zero area and back-facing triangles and/or flip edge equations
- Compute interpolation equations
  - Matrix/vector product per parameter

Massive Models

100,000,000 primitives

<u>   1,000,000 pixels   </u>

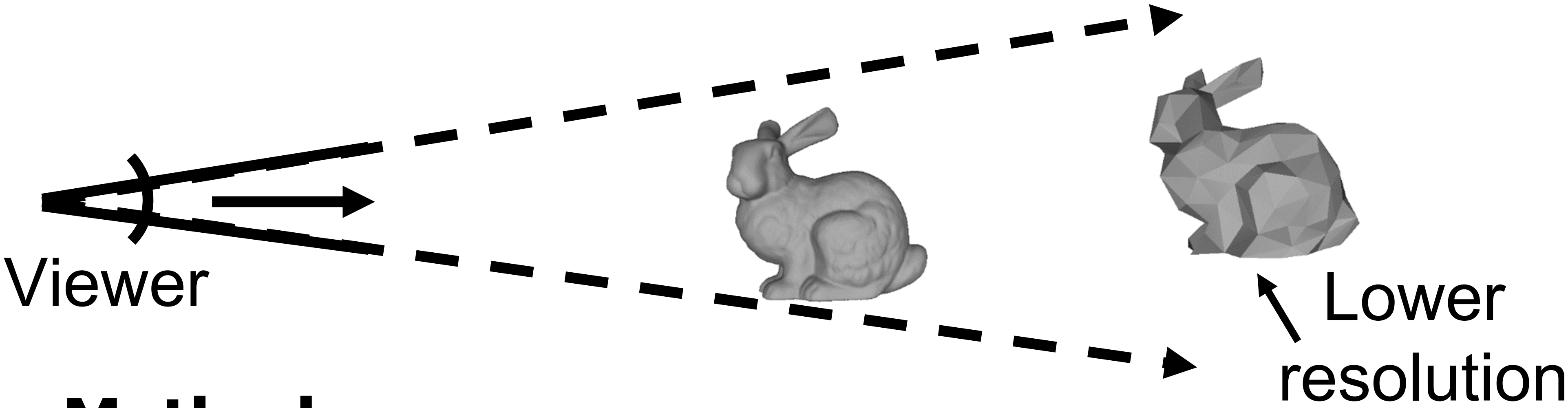
100 visible primitives/pixel

- **Cost to render a single triangle**
  - **Specify 3 vertices**
  - **Compute 3 edge equations**
  - **Evaluate equations one**

St. Mathew models consisting of about 400M triangles (Michelangelo Project)

Multi-Resolution or Levels-of-Detail (LOD) Techniques

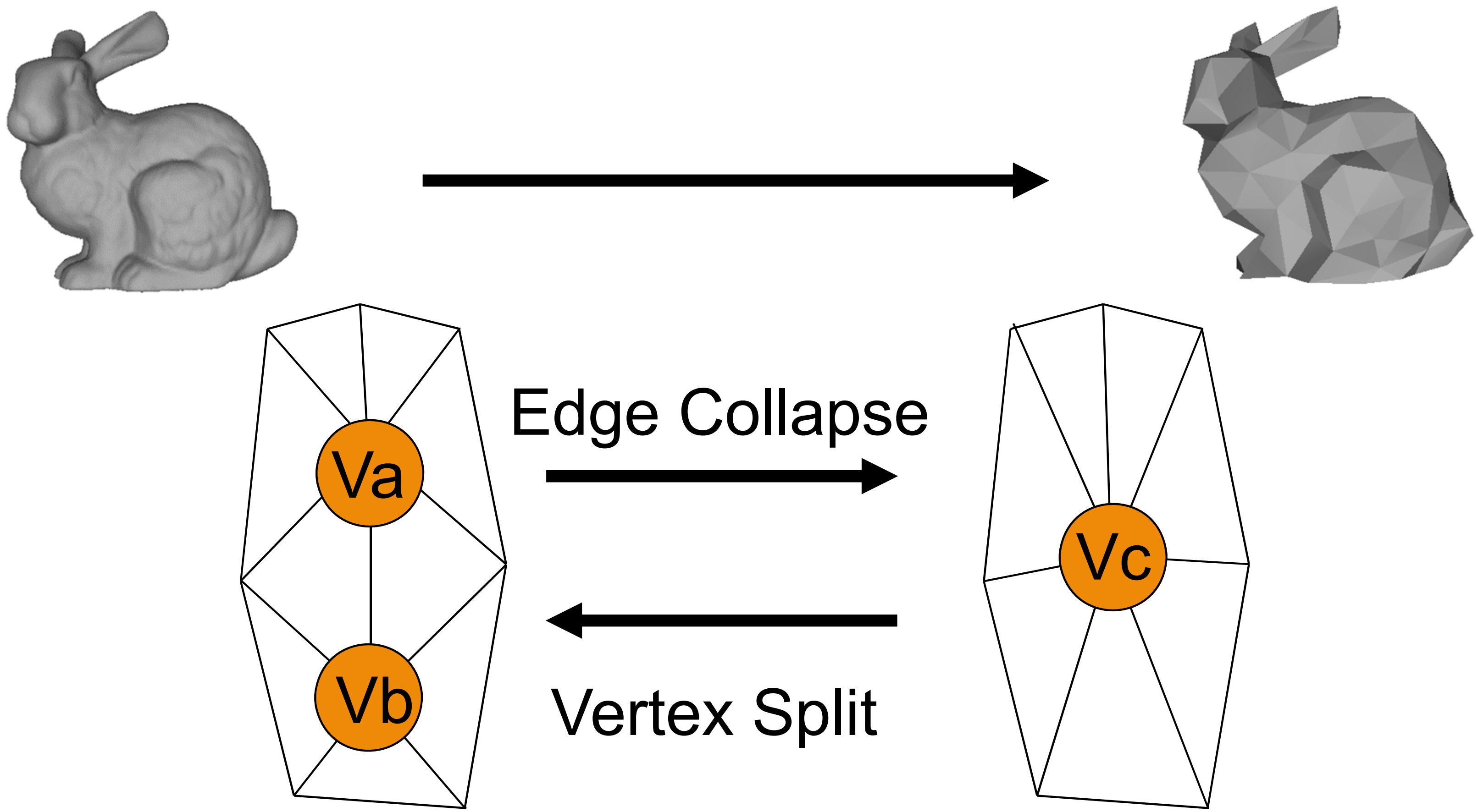
- **Basic idea**
  - **Render with fewer triangles when model is farther from viewer**



- **Methods**
  - **Polygonal simplification**

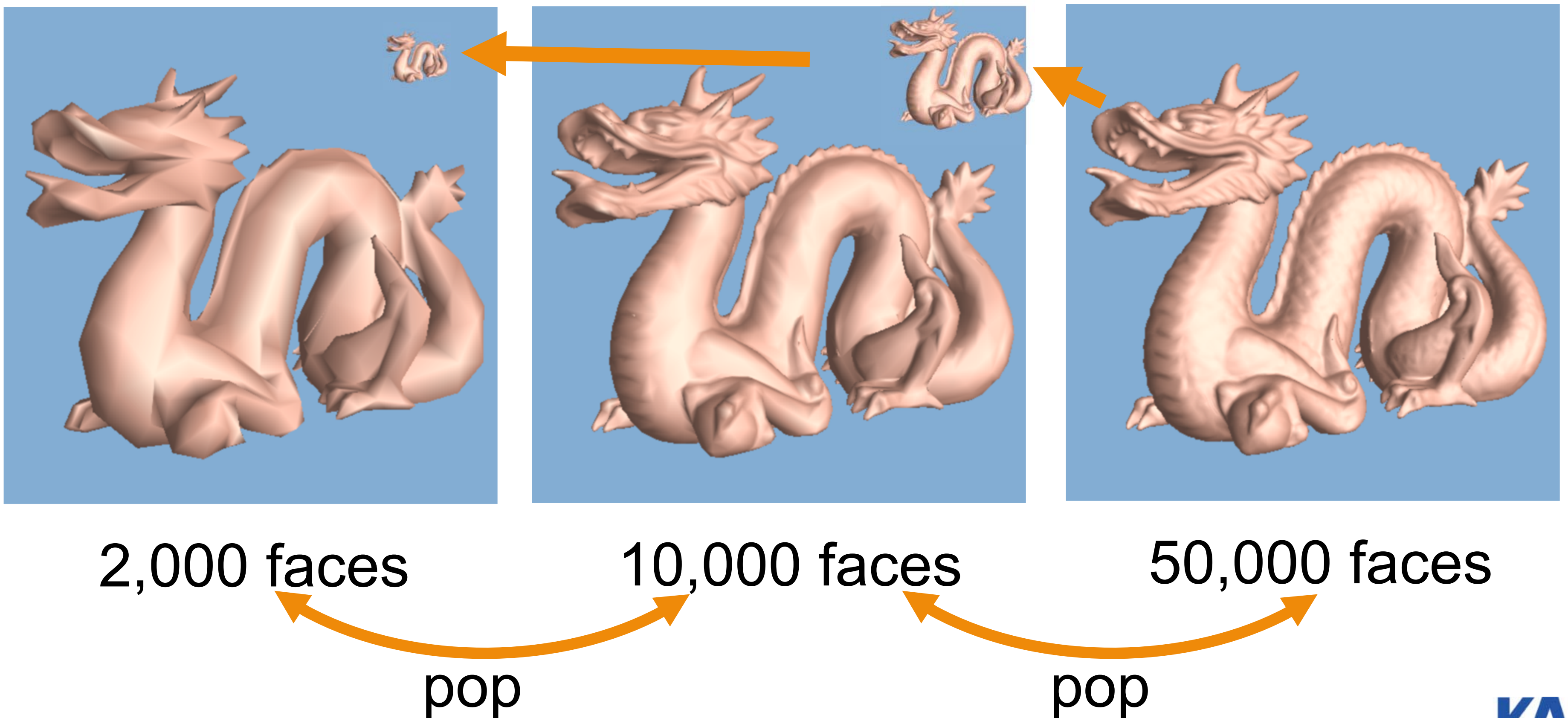
Polygonal Simplification

- **Method for reducing the polygon count of mesh**



Static LODs

- Pre-compute discrete simplified meshes
  - Switch between them at runtime
  - Has very low LOD selection overhead

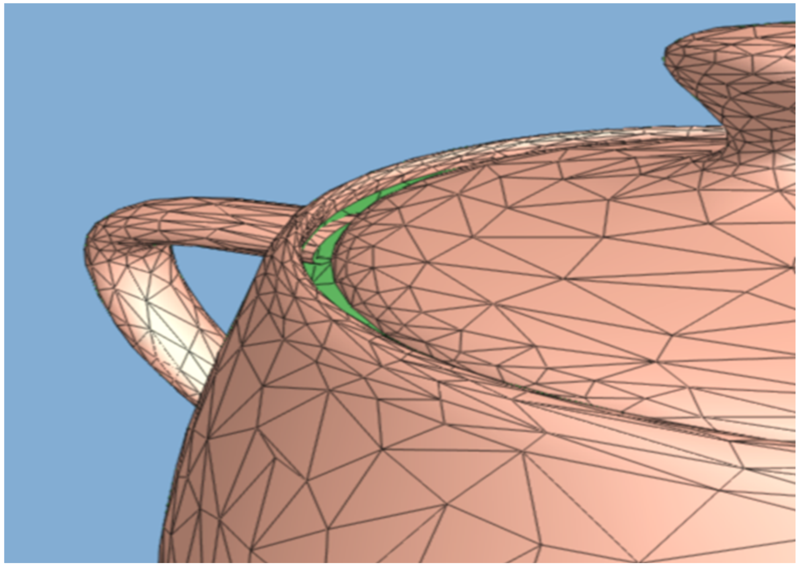


Excerpted from Hoppe's slides

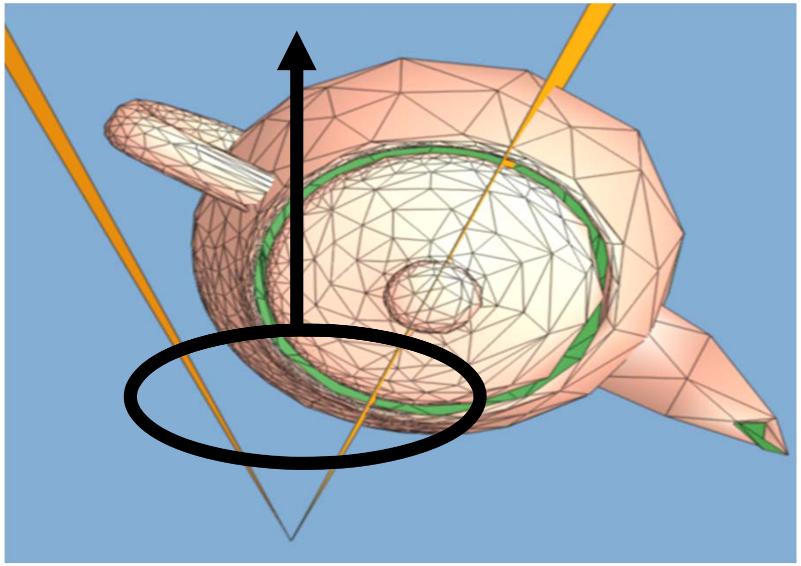
Dynamic Simplification

- **Provides smooth and varying LODs over the mesh** **[Hoppe 97]**

1st person's view



3rd person's view



Play video

KAIST

What if there are so many objects?

From "cars", a Pixar movie

Some solution: Stochastic Simplification of Aggregate Detail

Cook et al., ACM SIGGRAPH 2007

Class Objectives were:

- Understand triangle rasterization using edge-equations
- Understand mechanics for parameter interpolations
- Realize benefits of incremental algorithms

Homework

- Go over the next lecture slides before the class
- Watch 2 SIGGRAPH videos and submit your summaries before every Mon. class
  - Just one paragraph for each summary
- Submit questions two times during the whole semester

Next Time

- **Illumination and shading**
- **Texture mapping**