COS426 Precept8

Rasterizer

Presented by: Riley Simmons-Edler

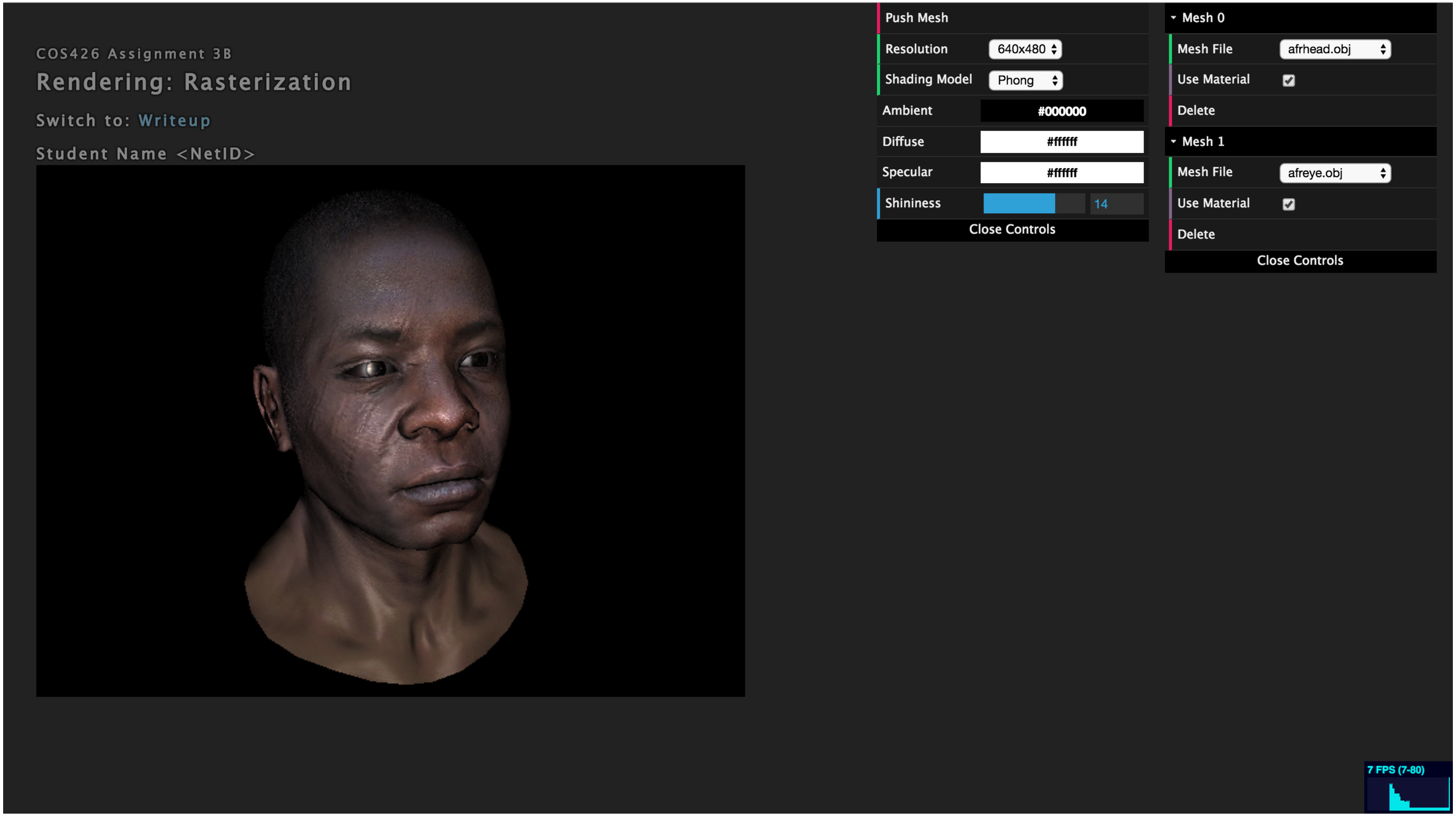
Rasterizer

- Render a lot of triangles in the image plane
  - Projection – orthogonal (naïve) or perspective
  - Which triangles are in the front? (z buffering)
  - How does the triangle react to the light? (reflection model)
  - Meshes are coarse. How to cheat our eyes? (interpolation)
  - How does the material affect the color? (texture mapping)
  - How to add fine details at low cost? (normal mapping)

Render a Pixel

- To render a pixel, we need the following ingredients.
  - normal of the pixel in the world coordinate system (interpolate using the three vertex normals and **barycentric coordinates**).
  - position of the pixel in the world coordinate system (interpolate using the three vertex positions and **barycentric coordinates**).
  - view position (where your camera/eye is, in the world coordinate system).
  - light position(s) (where the light source is, in the world coordinate system).
  - material of the pixel:
    - case 1: material is uniform (k_a, k_d, k_s, shininess).
    - case 2: texture maps. (we need uv coordinates to look up k_a, k_d, k_s, shininess of the pixel).
    - uv coordinates: (interpolate using the three uv coordnates and **barycentric coordinates**).
- All ingredients are essential for Phong reflection model.

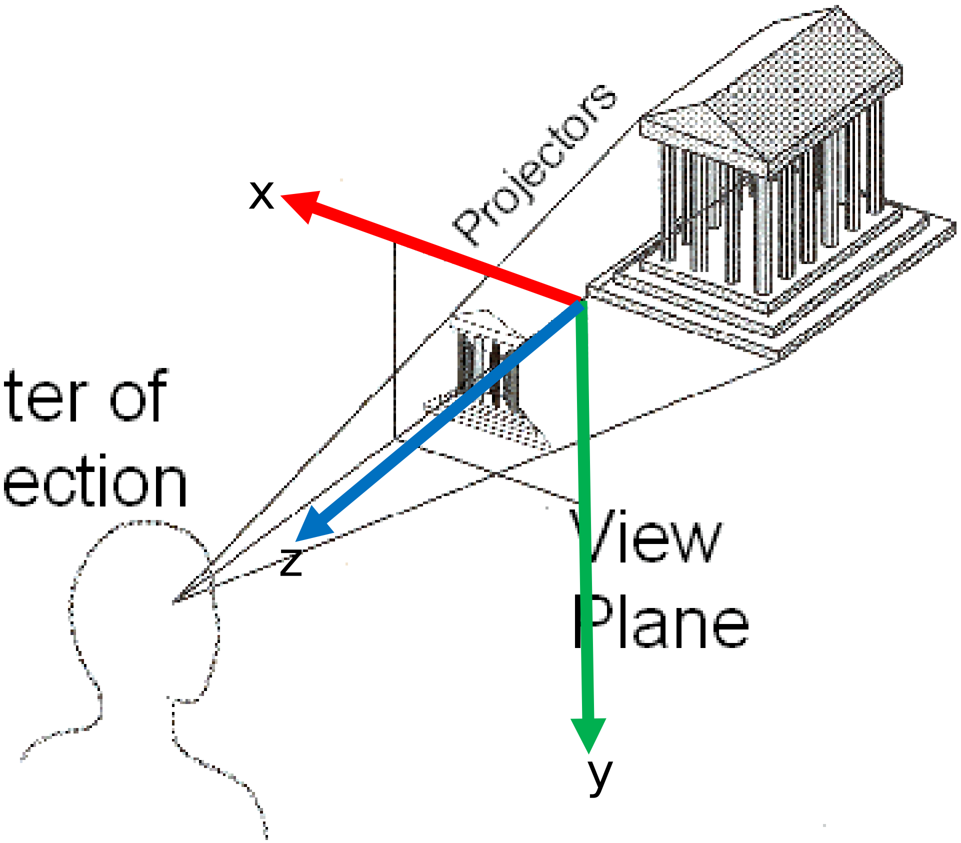
GUI & Demo



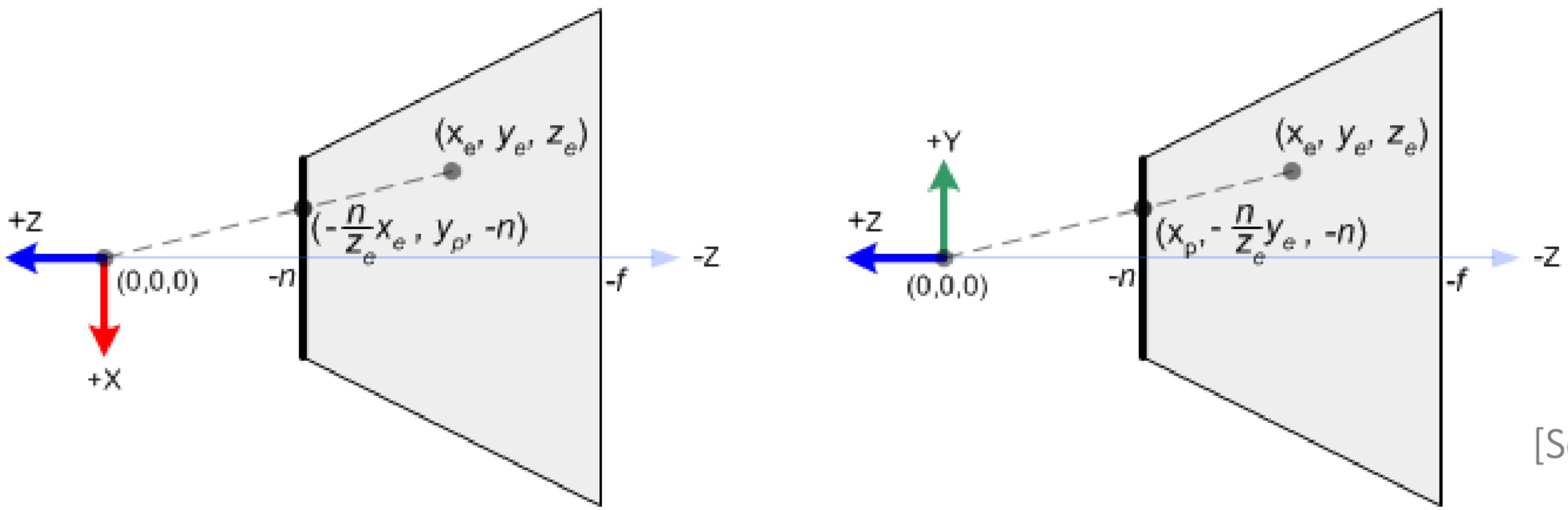
COS426 Assignment 3B
Rendering: Rasterization
Switch to: Writeup
Student Name <NetID>
Push Mesh
Resolution 640x480
Shading Model Phong
Ambient #000000
Diffuse #ffffff
Specular #ffffff
Shininess 14
Close Controls
Mesh 0
Mesh File afrhead.obj
Use Material
Delete
Mesh 1
Mesh File afreye.obj
Use Material
Delete
Close Controls
7 FPS (7-80)

Perspective Projection

objects must be on the negative z axis, otherwise cannot be seen.



Which triangles should we render? – near and far planes



[Song Ho Ahn]

1. n and f are usually positive values. But near plane locates at –n and far plane locates at –f.
2. if you z_{cam} is out side [-f, -n], skip that triangle.
3. project triangle vertices using the projection matrix.

Graphics Projection Transform

- Map x-component of a point to (-1, 1)
- Map y-component of a point to (-1, 1)
- Map z-component of a point from (near, far) to (-1, 1)
- Believe it or not, this matrix does the transformation:

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Use the Projection Matrix

- What is the fourth dimension?
  - This matrix is in homogeneous form and it should be multiplied with homogeneous coordinates: (x, y, z, 1)^T. Then you get (x’, y’, z’, w).
  - transform it back -> (x’/w, y’/w, z’/w)
  - if z is outside (near, far), don’t do the projection because it can’t be seen.

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Changing Camera Pose

- This projection matrix can only be directly used when the camera coordinate is perfectly aligned with the world coordinate. What if the camera is moving?
- We represent the pose of the camera in the world space as: [R|t], also in homogeneous form (4x4 matrix). [R|t] transforms a point represented in the camera coordinate system to the world coordinate system.
- But we want to transform a point in the world coordinate system to the camera coordinate system. So we simply use inv([R|t]).
- Concatenate with the previous projection matrix:
  - x inv([R|t] (given as viewMat in the code)

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Transformation

$$\begin{pmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \\ 1 \end{pmatrix} = \begin{pmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

homogeneous representation.
Why? it’s easier to concatenate square matrices.

Mixing Projection and Transformation

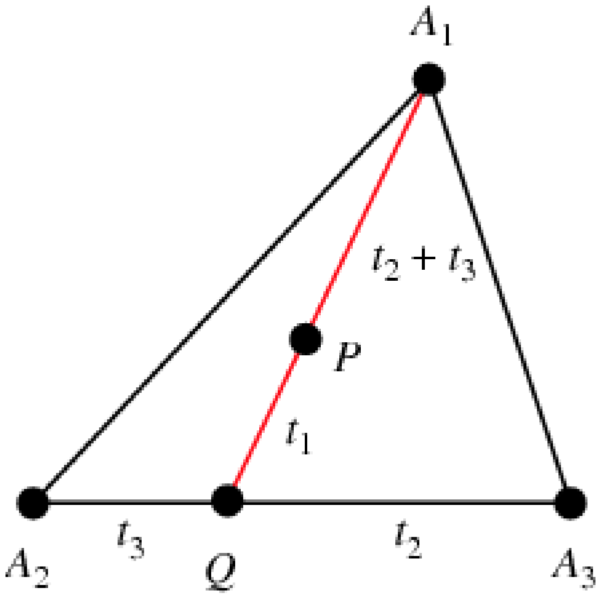
$$\begin{pmatrix} x' \\ y' \\ z' \\ w \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} * \begin{pmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

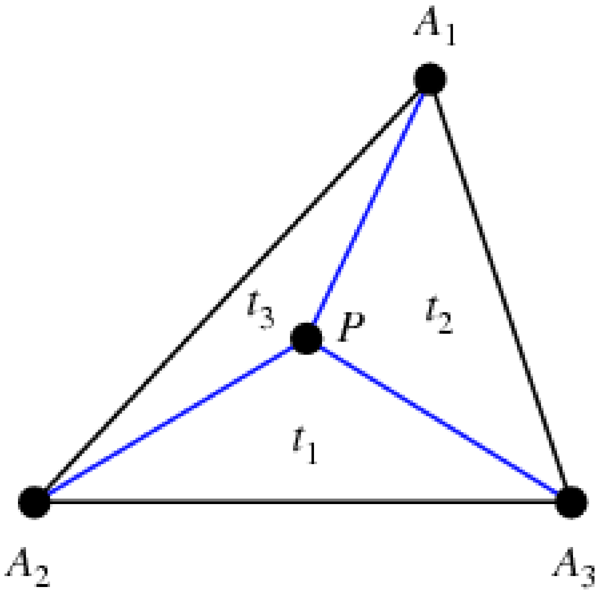
$$\begin{pmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \\ 1 \end{pmatrix}$$

$$w = -z_{cam}!$$

Barycentric Coordinates

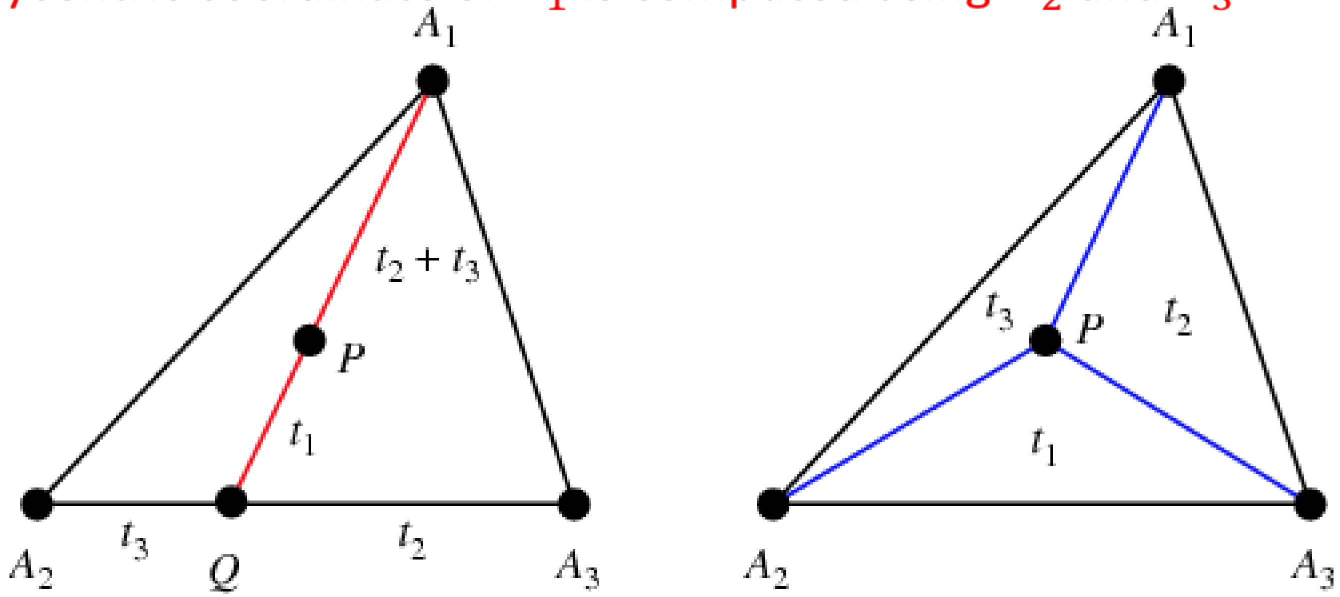
- Any point in the triangle can be represented as a linear combination of the three vertices
  - Q is a linear combination of A2 and A3
  - P is a linear combination of Q and A1





Barycentric Coordinates

- $P = \ \alpha A_1 + \beta A_2 + \gamma A_3$
- $\alpha + \beta + \gamma = 1$
- if any of $\alpha, \beta, \gamma < 0$, P is not in the triangle.
- barycentric coordinate of A_1 is computed using A_2 and A_3

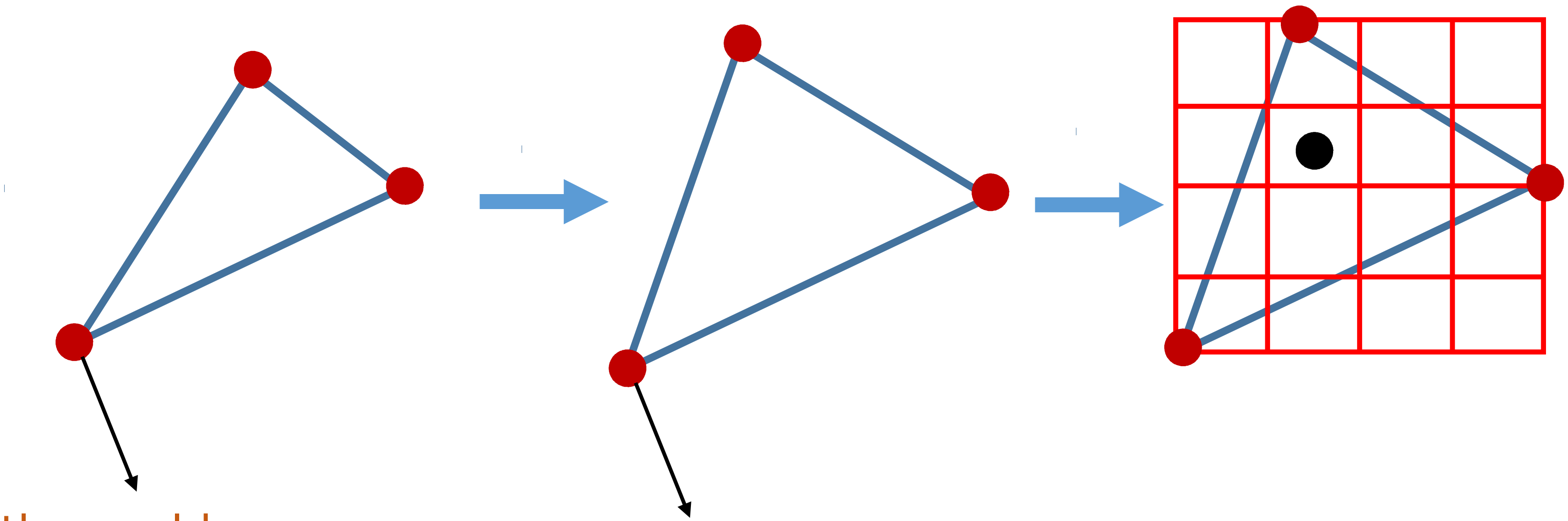


See this article for detailed computation:
https://fgiesen.wordpress.com/2013/02/06/the-barycentric-conspirac/

Use Barycentric Coordinates

- Weight average of the values on the 3 coordinates
  - Interpolate z coordinate
  - Interpolate color
  - Interpolate normal direction
  - Interpolate texture coordinates

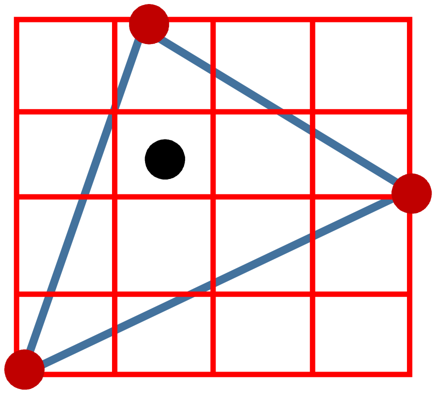
Pipeline of Rendering a Triangle



In the world coordinate system: verts[], normals[], uvs[](optional), material(optional).

In the world coordinate system: verts[], normals[], uvs[](optional), material(optional).
In the camera coordinate system: projectedVerts[].

Pipeline of Rendering a Triangle (Flat Shader)



For a pixel (x, y) in the bounding box:

1. determine whether it’s inside the triangle (**barycentric coordinates**).if not, go to the next pixel.
2. use **barycentric coordinates** to interpolate z’/w for the pixel.
3. If z’/w is not smaller(closer) than zBuffer[x][y], go to the next pixel.
4. If the pixel survives, render the pixel!

UV coordinates

- Can be computed automatically (a lot of papers). None of them is perfect.
- Done by artists.
- Specify where a triangle vertex should map to in the texture map.
- Not always available! Make sure to check whether uvs[] is defined or not.

Q&A