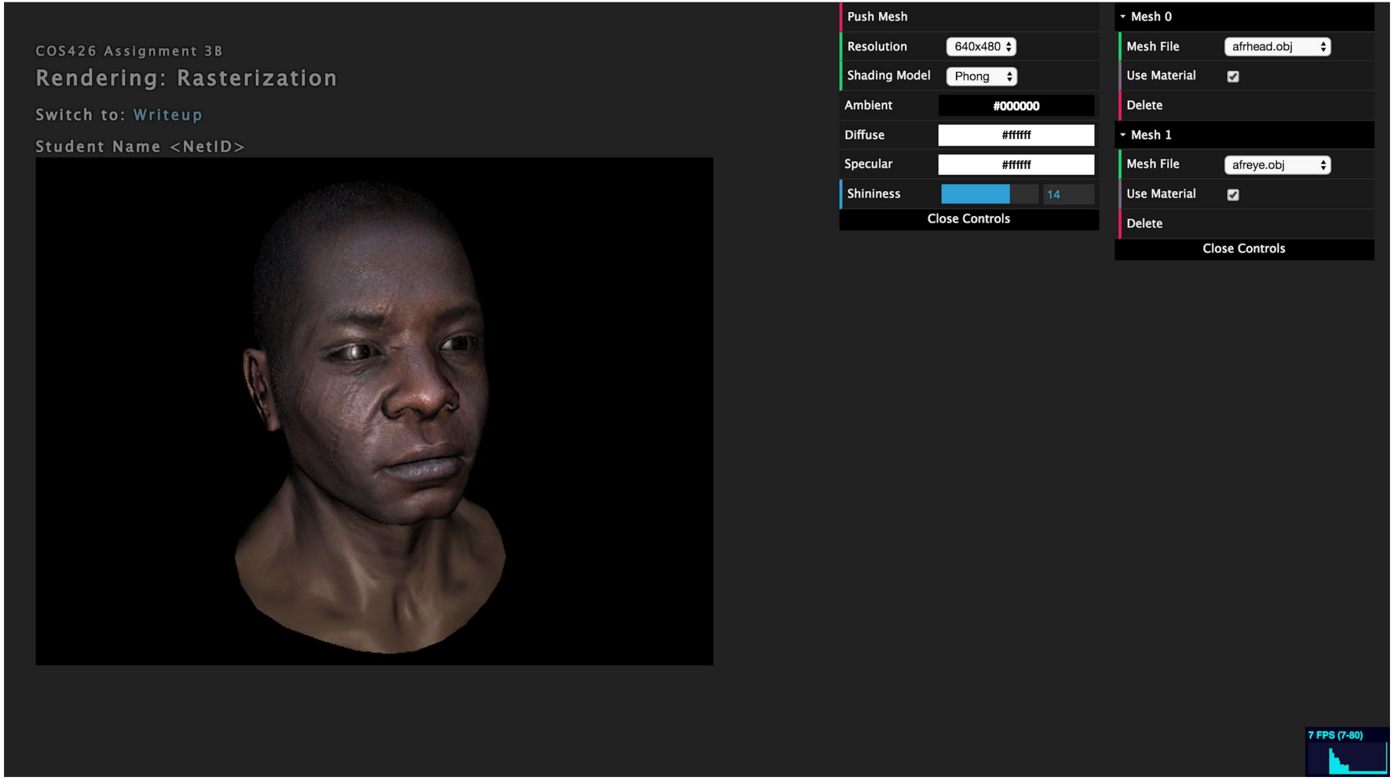
COS426 Precept

Rasterization

Presented by: Kyle Genova

GUI & Demo



COS426 Assignment 3B

Rendering: Rasterization

Switch to: Writeup

Student Name <NetID>

Resolution: 640x480

Shading Model: Phong

Ambient: #000000

Diffuse: #ffffff

Specular: #ffffff

Shininess: 14

Mesh 0

Mesh File: afrhead.obj

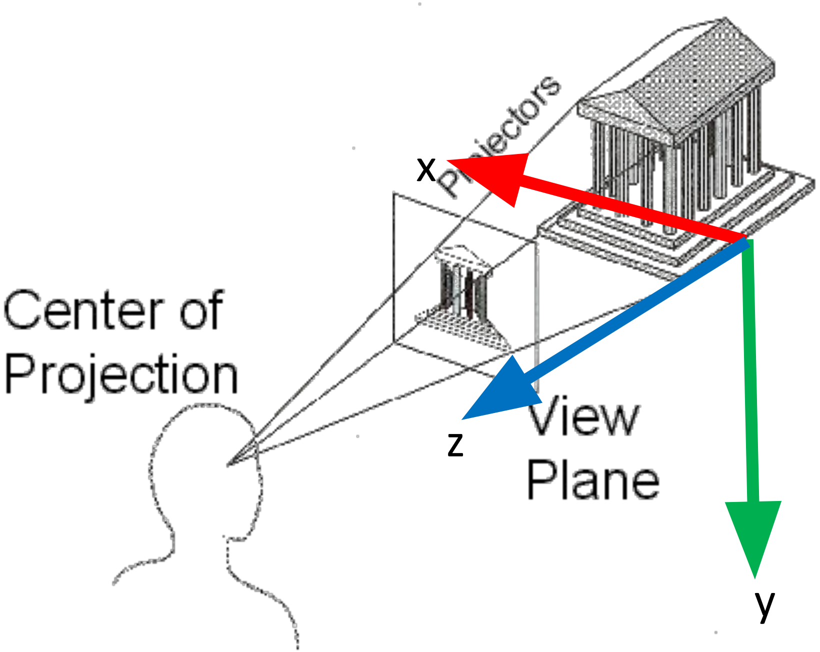
Use Material: ☑

Mesh 1

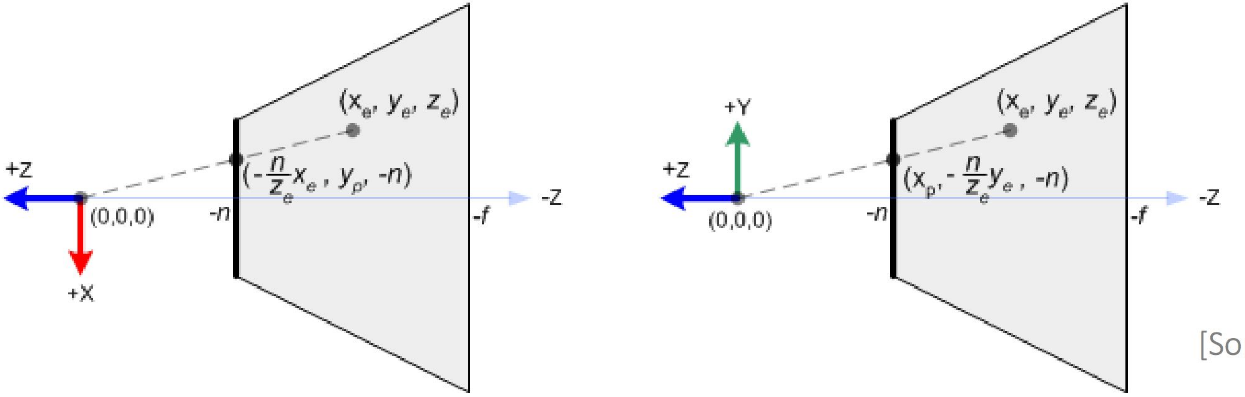
Mesh File: afreye.obj

Use Material: ☑

Perspective Projection



Near and Far Planes



[Song Ho Ahn]

n and f are usually positive values. But the near plane is located at –n and the far plane is located at –f.

Graphics Projection Transform

- Map x-component of a point from range [l,r] to range [-1, 1]
- Map y-component of a point from range [b,h] to range [-1, 1]
- Map z-component of a point from range [near, far] to range [-1, 1]
- This matrix does the transformation:

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

The Projection Matrix

- What is the fourth dimension?
  - This matrix is in homogeneous form and it should be multiplied with 4D homogeneous coordinates.
  - To lift a 3D nonhomogeneous coordinate, (x,y,z)^T -> (x, y, z, 1)^T. Then you get (x’, y’, z’, w) after a transformation.
  - To project a 4D homogeneous coordinate to a 3D nonhomogeneous coordinate: (x’, y’, z’, w)-> (x’/w, y’/w, z’/w)
  - if **camera space** z is outside (near, far), skip the triangle because it shouldn’t be seen.

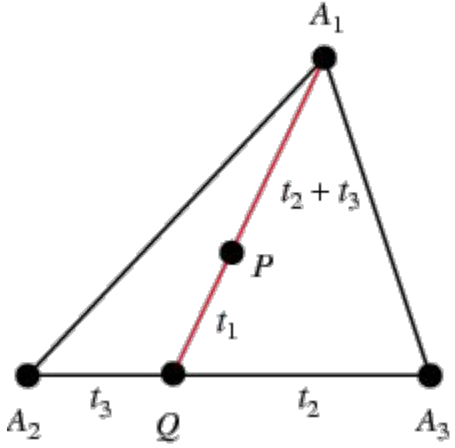
$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

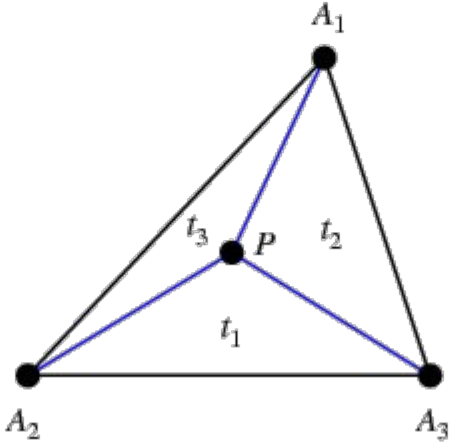
Changing Camera Pose

- This projection matrix assumes the camera is at the world origin pointing down -z. What if the camera has an arbitrary pose?
- 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 apply the inverse of [R|t].
- In the code: *viewMat := projMat * ([R|t])^-1*

Barycentric Coordinates

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

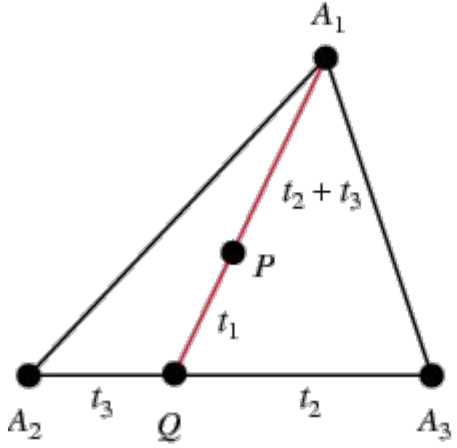


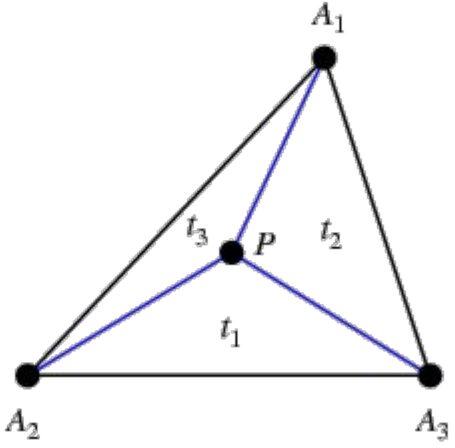


Barycentric Coordinates

See this article for an efficient 2D algorithm:

https://fgiesen.wordpress.com/2013/02/06/the-barycentric-conspirac/

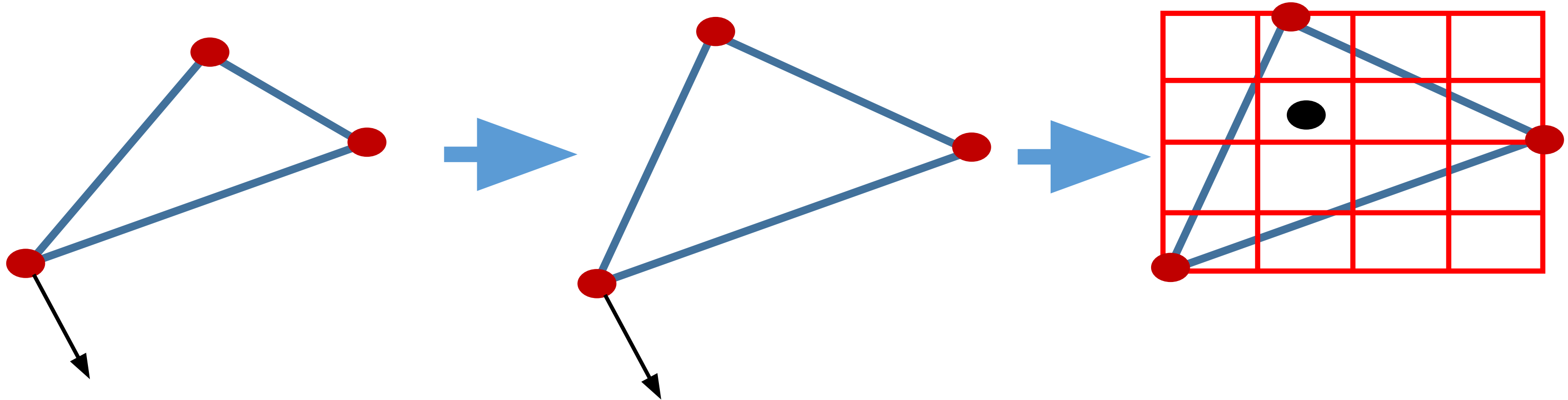




Barycentric Interpolation Uses

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

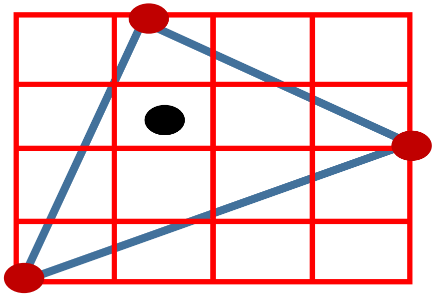
Pipeline of Rendering a Triangle



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!

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 or per-vertex (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 can also be interpolated using the three vertex uv coordinates and barycentric coordinates).

UV coordinates

- Can be computed automatically (a lot of papers). None of them is perfect.
- Usually generated with the help of 3d modelers.
- They specify the location of a vertex in the texture map.
- Not defined for all meshes! Make sure to check whether uvs[] is defined or not.