Graphic Pipeline

Rendering

- If we have a precise computer representation of the 3D world, how realistic are the 2D images we can generate?
- What are the best way to model 3D world?
- How to render them?

Data Representation

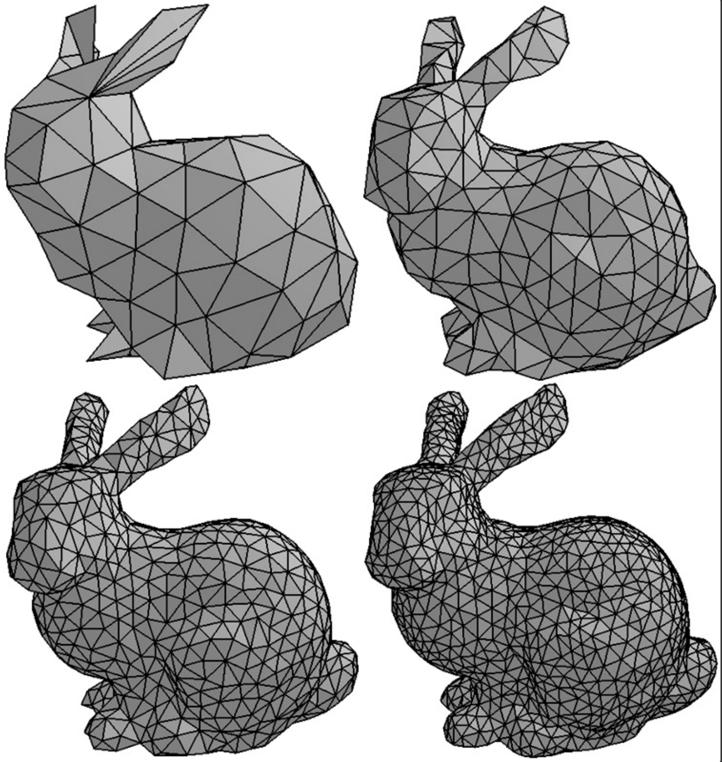
- How do we define objects
  - Primitives (triangle, polygon, surfaces)
- Polygonal model
  - Each primitive is a planar polygon
  - Object is made of a mesh of polygons

Triangular Model

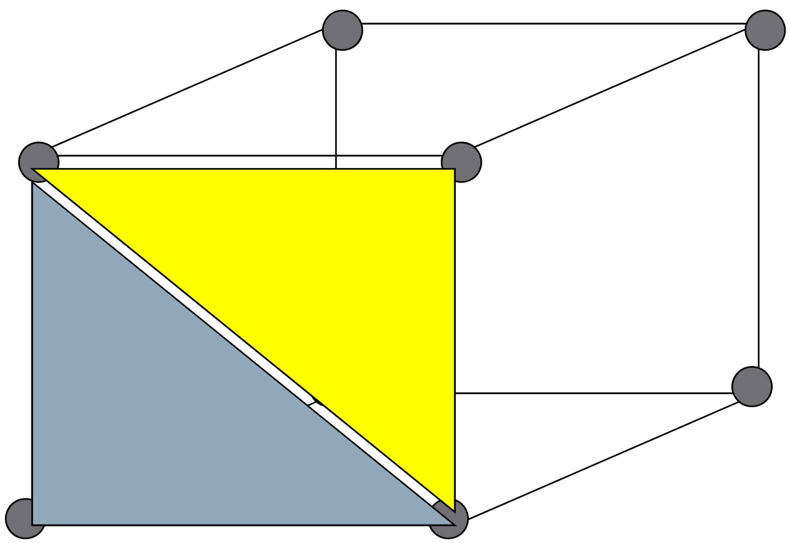
- Triangular model
  - Any surface passing through three vertices will be planar
  - Minimal planar primitives
  - No restrictions to be imposed during model building
- Piecewise Linear Representation
  - Easy to implement in hardware
  - Easy to interpolate attributes
    - Convex Linear Interpolation
    - Unique coefficients

Most Common Format

- List of vertices and attributes
  - 3D coordinates, color, texture coordinates….
  - Geometric information
    - Positions, normals, curvature
- List of triangles
  - Indices of triangles
  - Topological information
    - How are the triangles connected?



Object Representation: Example



Rendering Pipeline

- Input
  - Soup of 2D triangles in 3D space
- Output
  - 2D image from a particular view
- Why pipeline?
  - Contains different stages
  - Each triangle is sent through it in a pipeline fashion

Steps of Graphic Pipeline

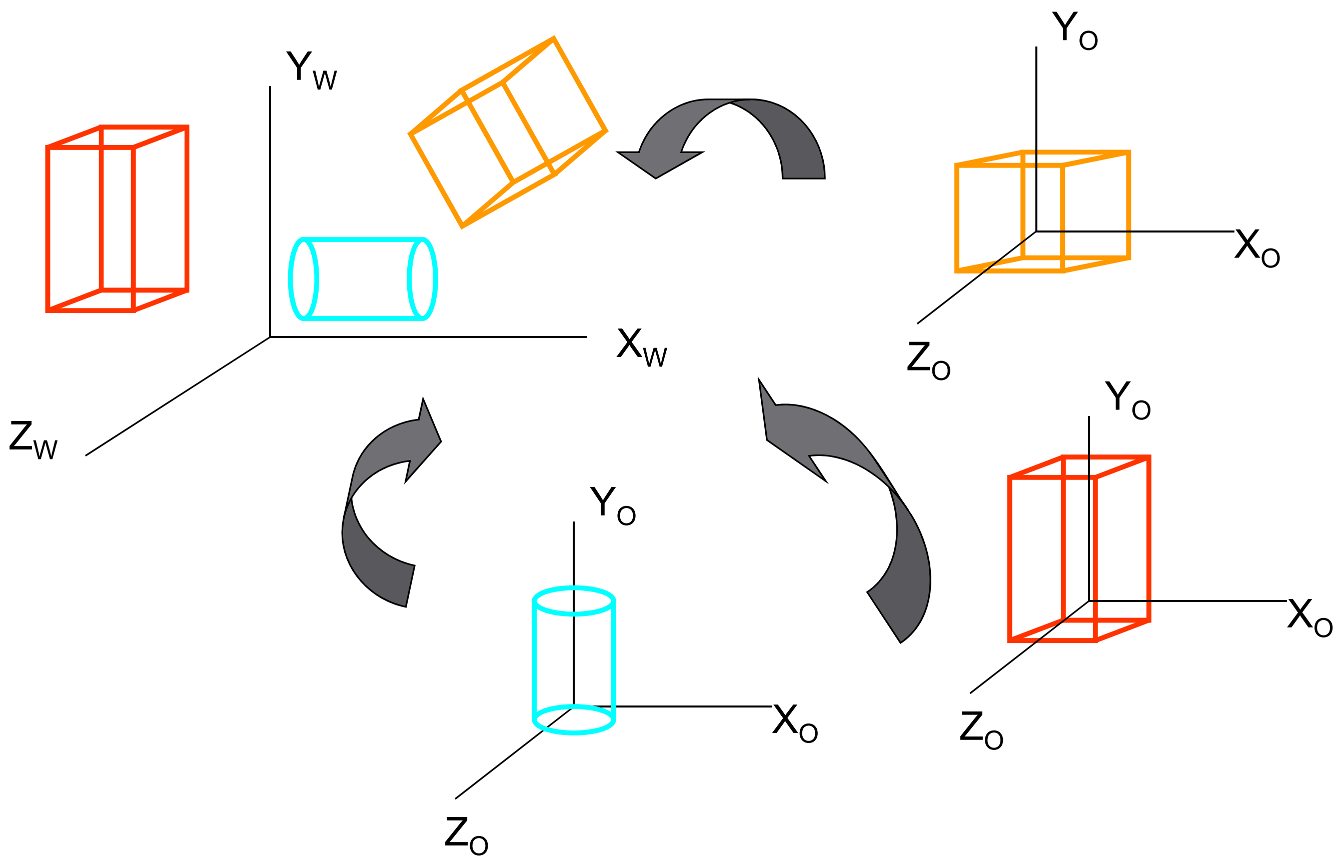
1. Geometric Transformation of Vertices
2. Clipping and Vertex Interpolation of Attributes
3. Rasterization and Pixel Interpolation of Attributes

Geometric Transformation

1. Model Transformation
2. View Transformation
3. Perspective Projection
4. Window Coordinate Transformation

Model Transformation

- World and Object Coordinates



Model Transformation

- Transforming from the object to world coordinates
  - Placing the object in the desired position, scale and orientation
- Can be done by any kind of transformations
  - Graphics hardware/library support only linear transformations like translate, rotate, scale, and shear

Advantages

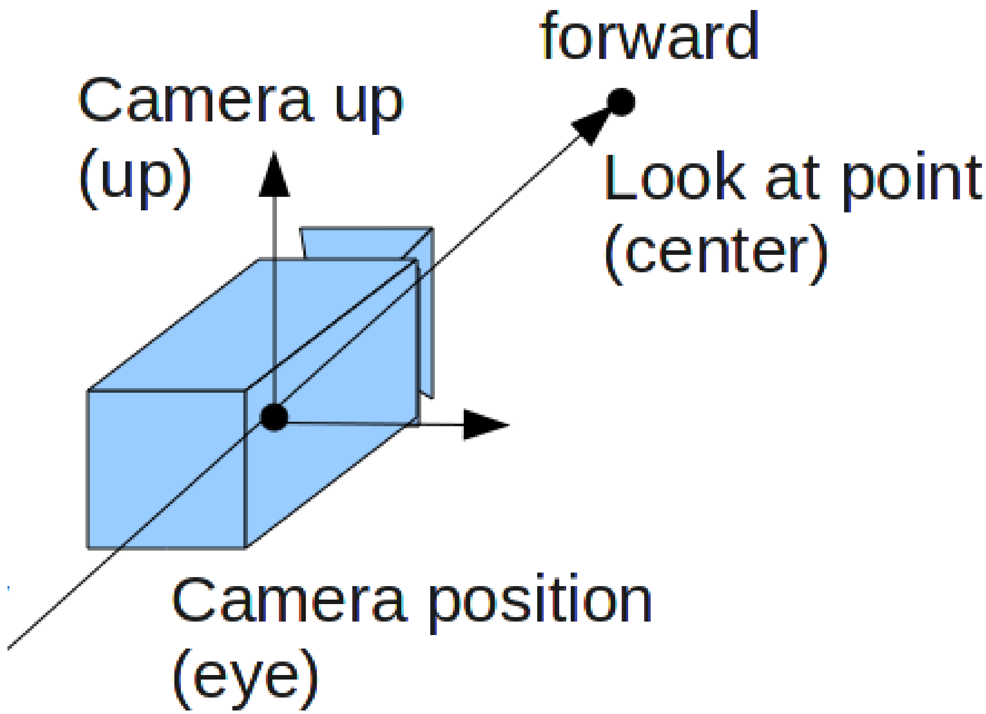
- Allows separation of concerns
  - When designing objects do not worry about scene
  - Create a library of objects
- Allows multiple instantiation by just changing the location, orientation and size of the same object

View Transformation

- Input
  - Position and orientation of eye (9 parameters)
    - View point (COP)
    - Normal to the image plane – N
    - View Up – U
- To align
  - Eye with the origin
  - Normal to the image plane with negative Z axis
  - View Up vector with positive Y axis
  - Can be achieved by rotation and translation

Default View Setup

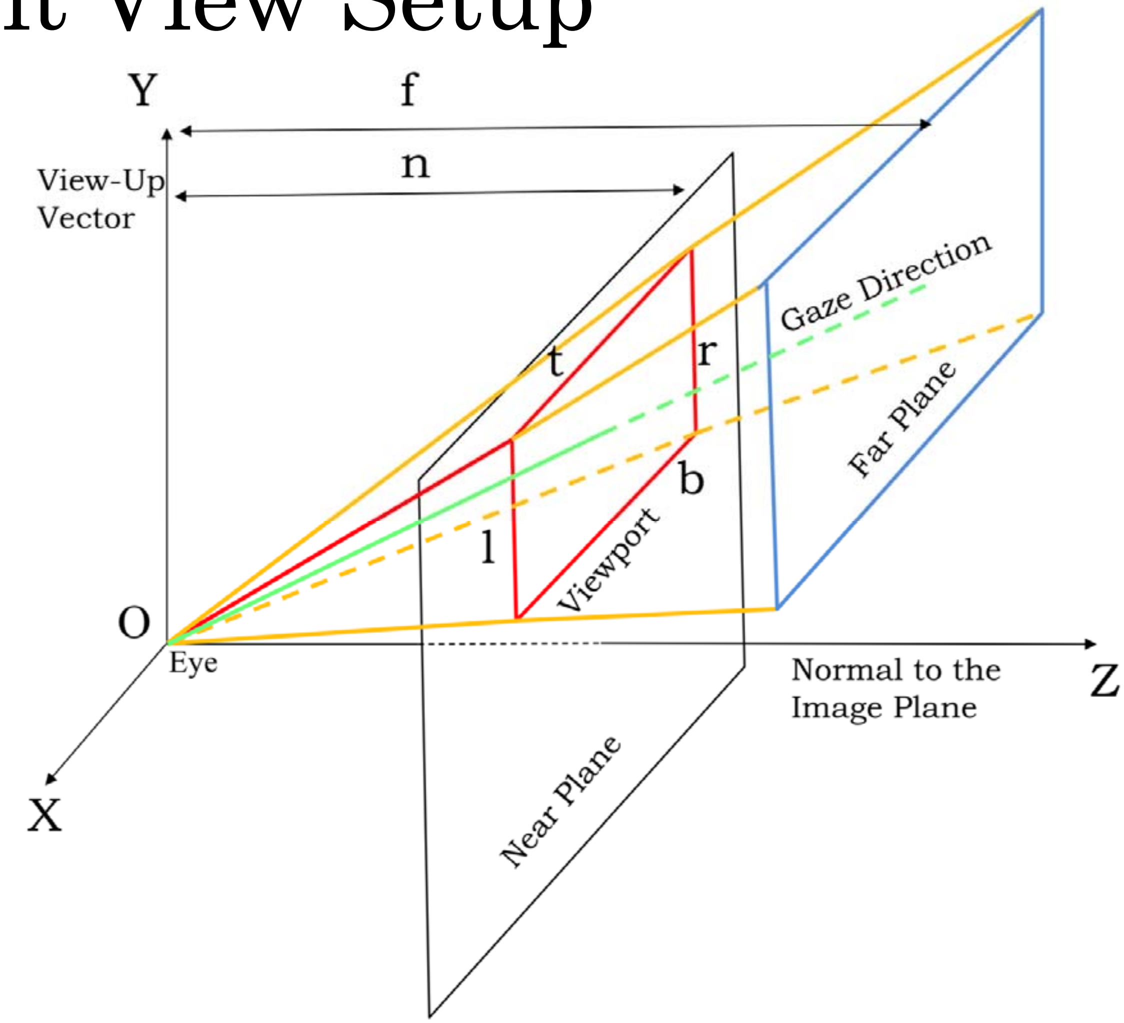
- E = (0,0,0)
- V = (0,1,0)
- N = (0,0,1)



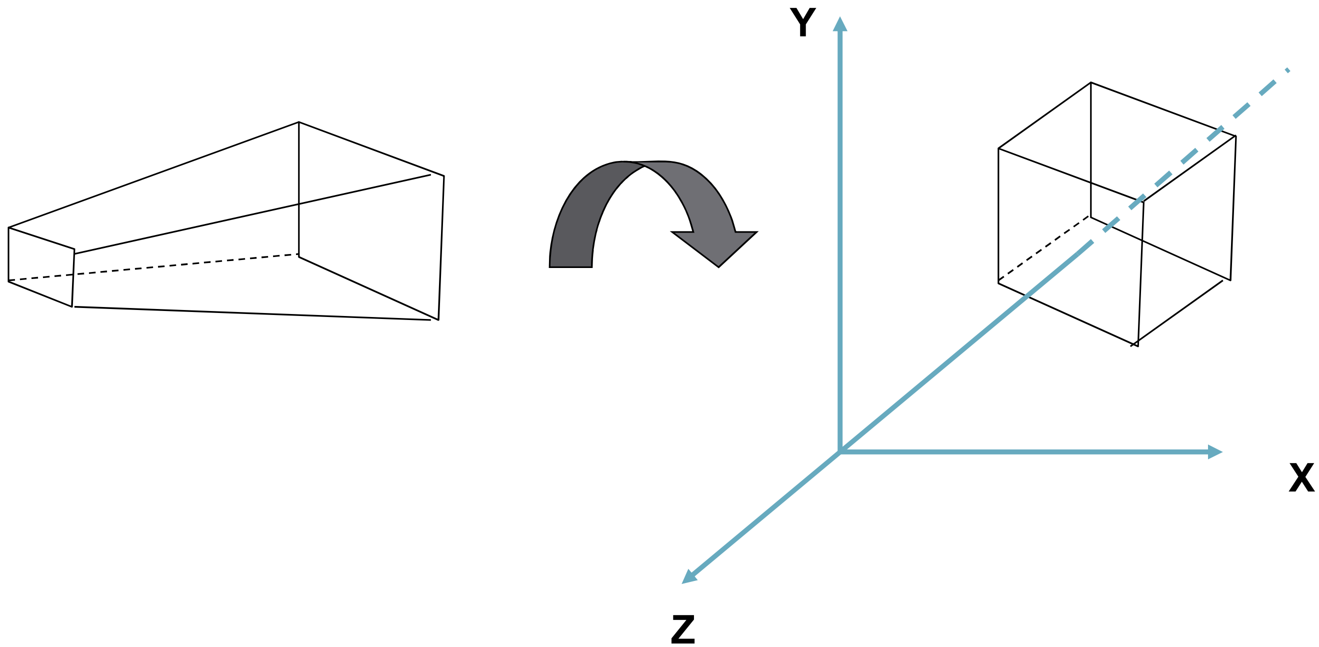
Perspective Projection Transformation

- Define the "view frustum" (6 parameters)
  - Assume origin is the view point
  - Near and far planes (planes parallel to XY plane perpendicular to the negative Z axis) [2]
  - Left, right, top, bottom rectangle defined on the near plane [4]

Default View Setup



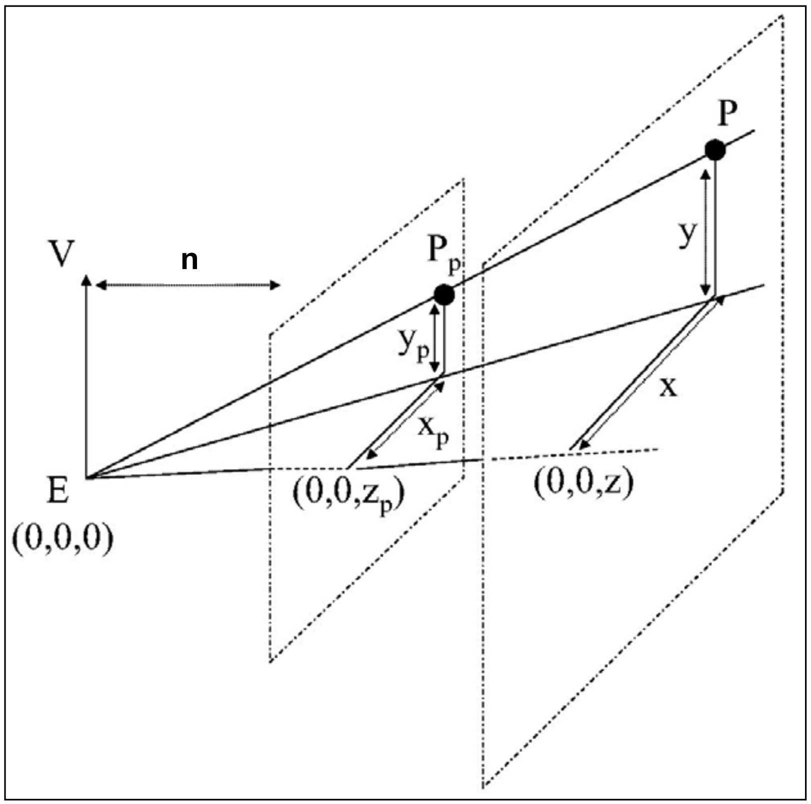
Projection Transformation

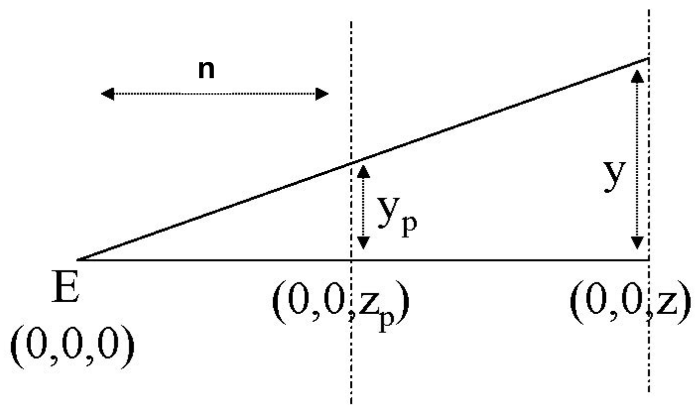


Projection Transformation

- Transforming the view frustum (along with the objects inside it) into a
  - cuboid with unit square faces on the near and far planes
  - the negative Z axis passes through the center of these two faces.
  - Projecting the objects on the near plane
- Consists of a shear, scale and perspective projection

Perspective Projection

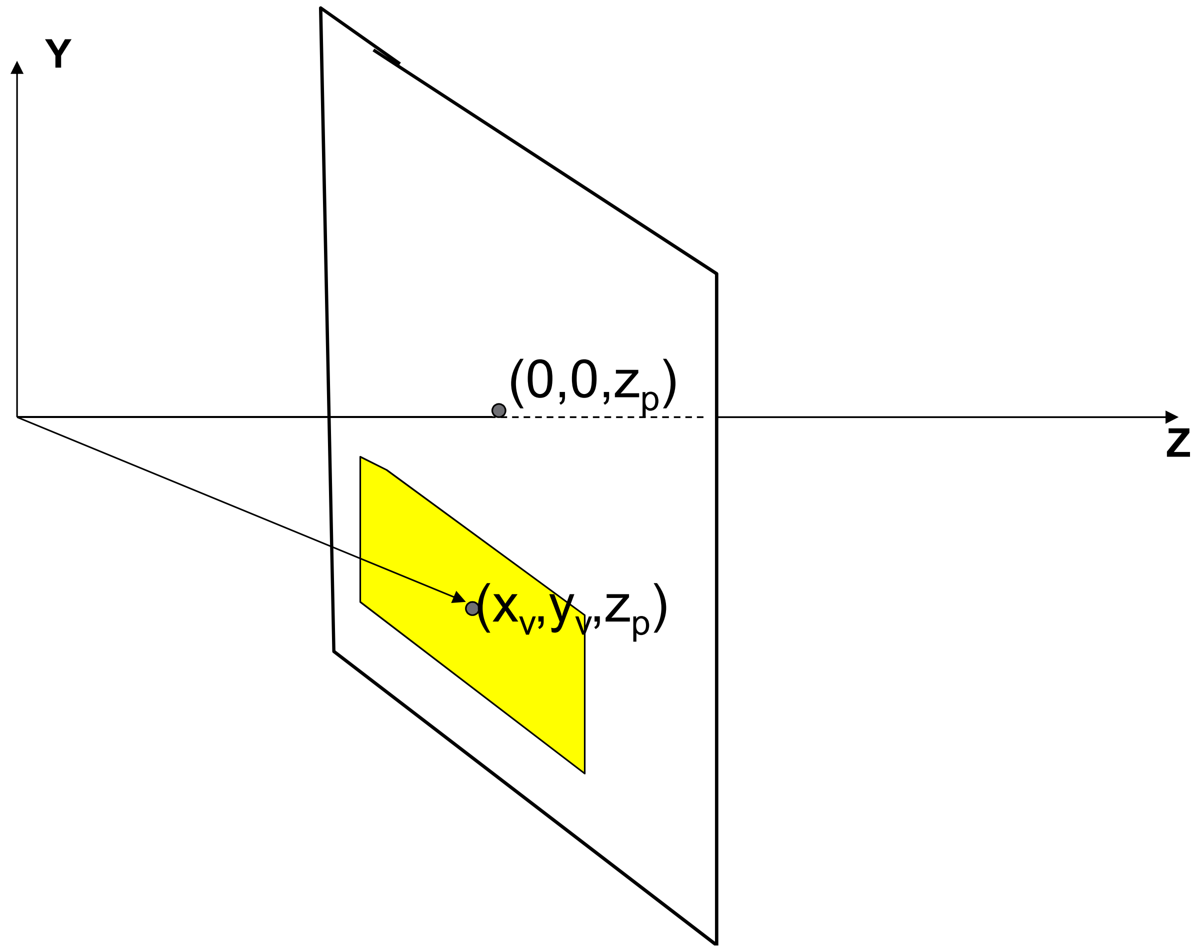




$$x_p/x = y_p/y = z_p/z$$

$$x_p = \frac{x}{\frac{z}{n}} \qquad y_p = \frac{y}{\frac{z}{n}}$$

Gaze Direction



Coincide this with N

- Shear Matrix

$$Sh(x_v/n, y_v/n) = \begin{bmatrix} 1 & 0 & x_v/n & 0 \\ 0 & 1 & y_v/n & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Can be defined by the window extents
  - l, r, t, b

$$Sh((r+l)/2n, (t+b)/2n) = \begin{bmatrix} 1 & 0 & r+l/2n & 0 \\ 0 & 1 & t+b/2n & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now normalize X and Y

- Map X and Y between -1 to +1
- Scale by 2/(r-l) and 2/(t-b)

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

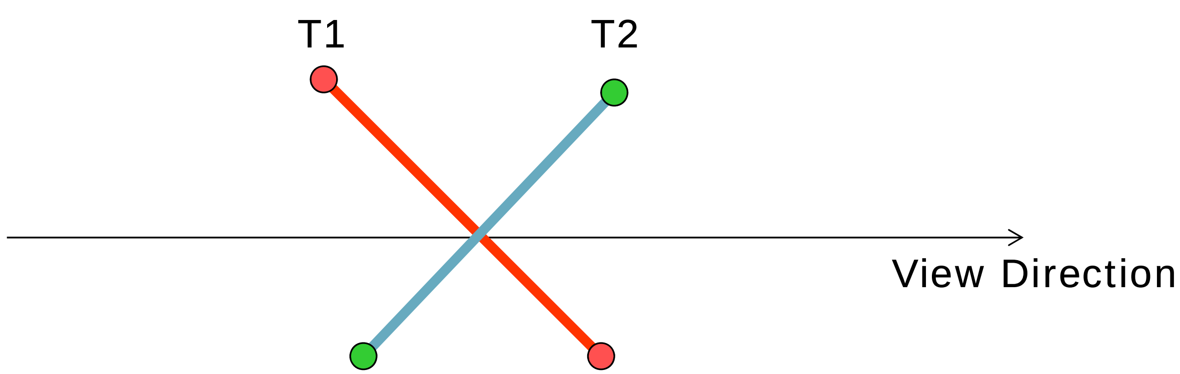
- Looks like K
  - n is focal length
  - r+l is change of center
  - r-l is inversely proportional to number of pixels

Where is the lost dimension?

- Why 4x4?
- Z should map to n always, since depth of the image is same
- But we need to resolve occlusion

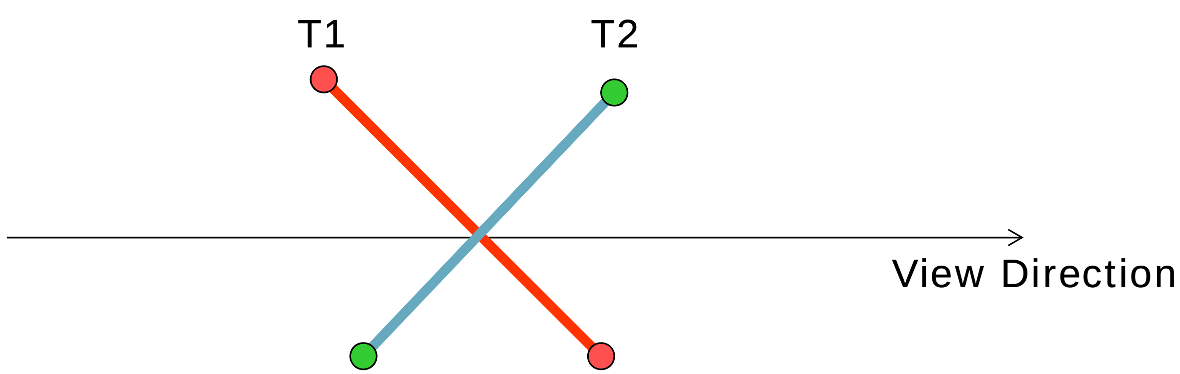
How do we use the z?

- Perspective projection is applied on the vertices of a triangle
- Can depth be resolved in the triangle level?
  - T1 is not infront of T2 and vice versa
  - Part of T1 is in front of T2 and vice versa



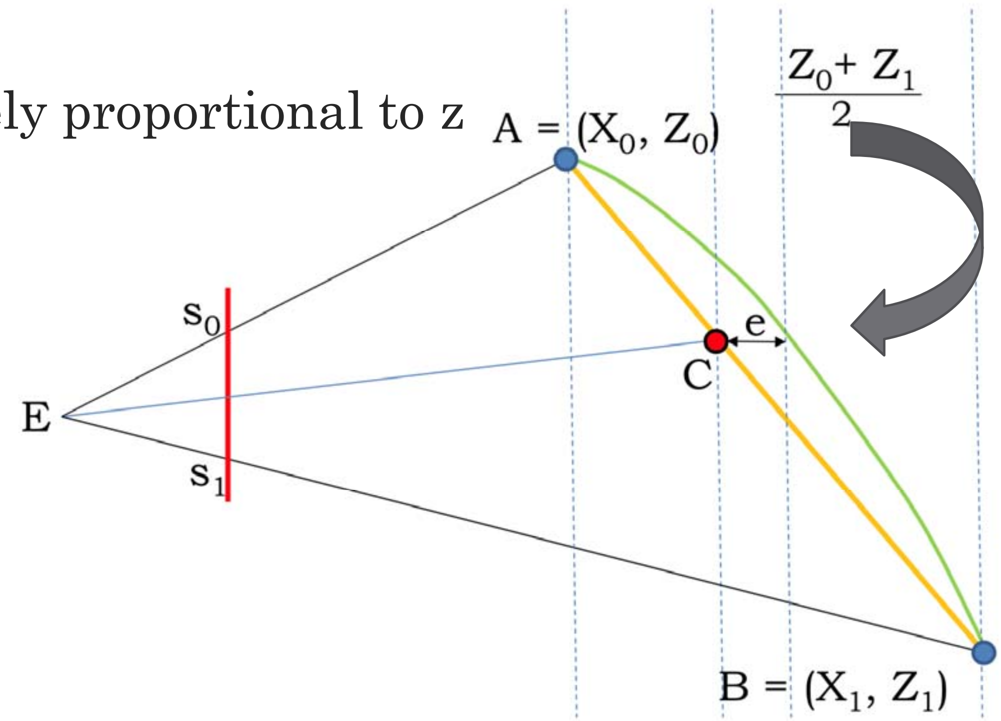
How do we use the z?

- Occlusion has to be resolved in the pixel level
- How do we find z for a point inside the triangle
  - Not its vertex
- We do not want to apply 3D to 2D xform
  - Too expensive
- Interpolate in 2D (screen space interpolation)



Screen Space Interpolation

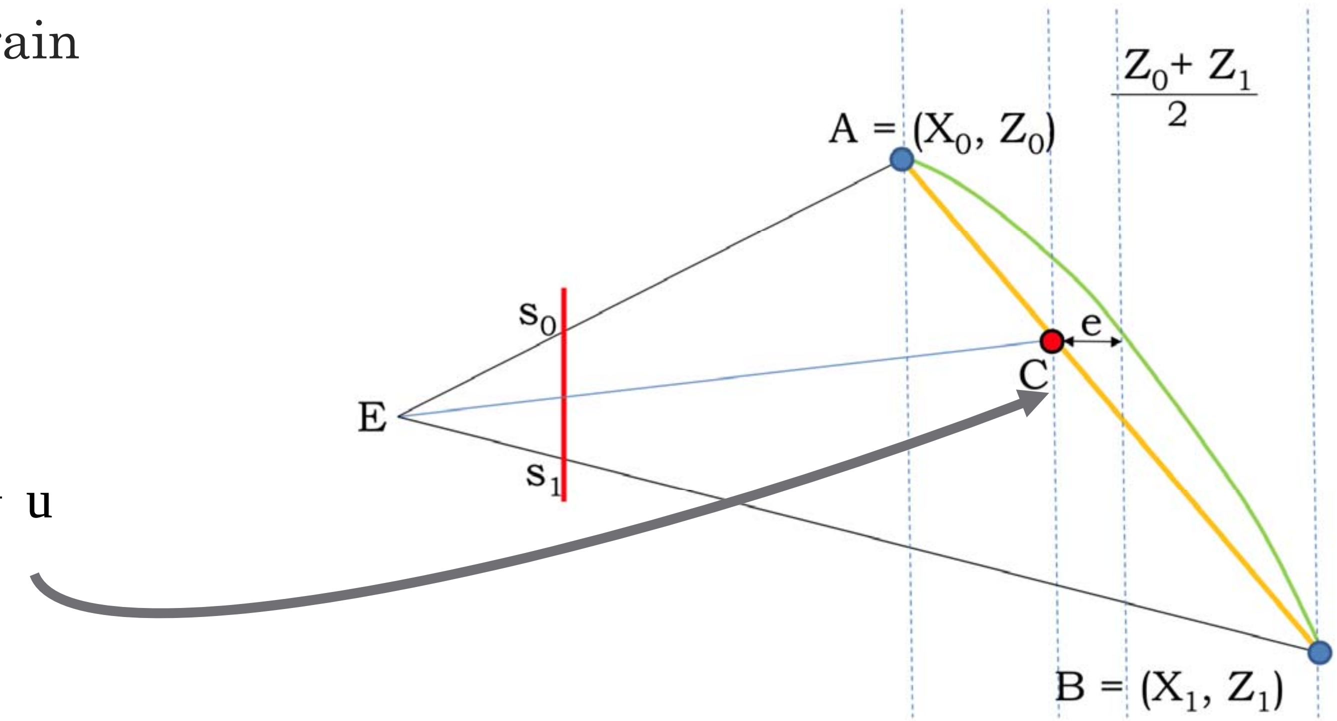
- Linear interpolation of z in screen space
- Does not work
- Why?
  - Perspective projection is inversely proportional to z
  - Over-estimates
  - Wrong occlusion resolution



Correct Solution

- Interpolate 1/Z
  - Reciprocal of Z
  - Interpolate in screen space
  - Take reciprocal again

$$\frac{1}{Z_t} = \frac{1}{Z_0}(1-u) + \frac{1}{Z_1}u$$



Transforming z to 1/z

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} x_p \\ y_p \\ -z \\ 1 \end{bmatrix}$$

Instead of this ...

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} x_p \\ y_p \\ -1/z \\ 1 \end{bmatrix}$$

we would like to store 1/z for interpolation purposes

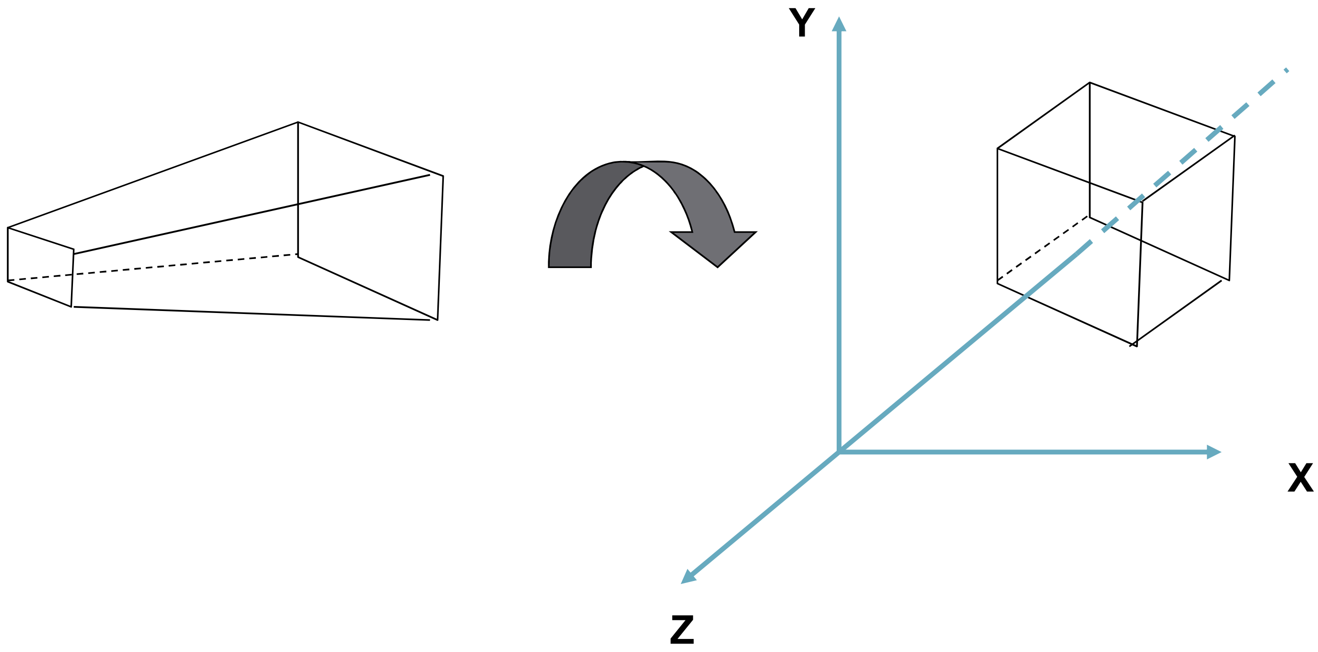
Bounding Z

- Depth of field effect
- Define a far plane - f
- Leads to culling of distant objects
  - Efficiency issues

Normalizing 1/z

- Map 1/n and 1/f to -1 and +1
  - Three steps only on z coordinates
    - Translate the center between -1/n and -1/f to origin
      - T(tz) where tz = (1/n+1/f)/2)
    - Scale it to match -1 to +1
      - S(sz) where sz = 2/(1/n-1/f))
- Whole z transform
  - (1/z + tz)sz = 1/z(2nf/f-n) + (f+n)/(f-n)

Projection Transformation



Final Matrix

- Defined only in terms of the planes of the view frustum

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