Graphics Rendering Pipeline

CS 211A

Pinhole Camera

- Linear geometric camera model that mimic our eye closely
- 3D to 2D image creation
- How to reconstruct 3D from 2D images?
  - Under-constrained problem
  - How do we constrain it?

The complementary view

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

Modeling

- Humans perceive objects
  - But this is too high level
- How do we define objects
  - Primitives (triangle, polygon, surfaces)
- What kind of objects can we consider?

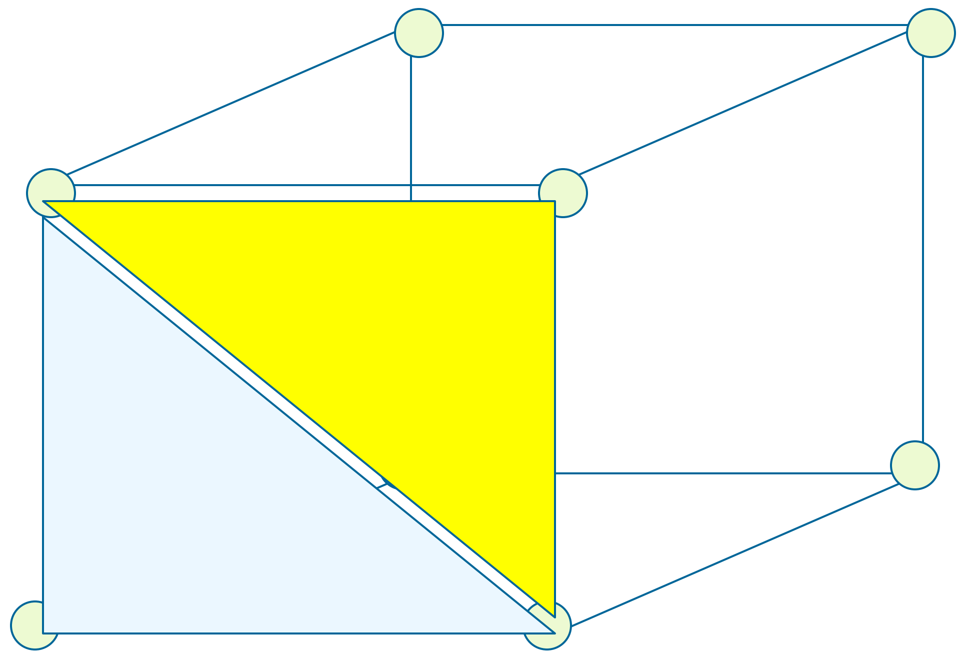
Most common

- Polygonal model
  - Each primitive is a planar polygon
  - Object is made of a mesh of polygons
- Triangular model
  - Triangle is always planar
  - Hence, less responsibility on modeling package

Most common format

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

Object Representation: Example



Topological Properties

- Manifold
  - Every edge has exactly two incident triangles
- Manifold with boundaries
  - Every edge has one or two incident triangles
- Non-manifold
  - Not with above restrictions

Topological Properties

- Euler characteristics
  - V-E+F
- Genus
  - Number of handles
  - $e = 2-2g$
- Orientability
- Dimension
  - No. of parameters that can be changed and still be on the primitive
  - Can be embedded in higher dimension space
- Cannot change these properties by changing the geometric properties

Topological Properties

- Cannot change these properties by changing the geometric properties
- Will deal with 2D orientable manifolds

Why triangles?

- 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

Why not other?

- Quadrilaterals
  - Non-unique interpolation
- Curved patches
  - Has to be rasterized for rendering
  - Can be useful for computations
  - Used for large simulation applications

Rendering Pipeline

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

Stages

- Model-View Transformation
- Projection Transformation
- Clipping and vertex interpolation of attributes
- Rasterization and pixel interpolation of attributes
- Graphics Hardware

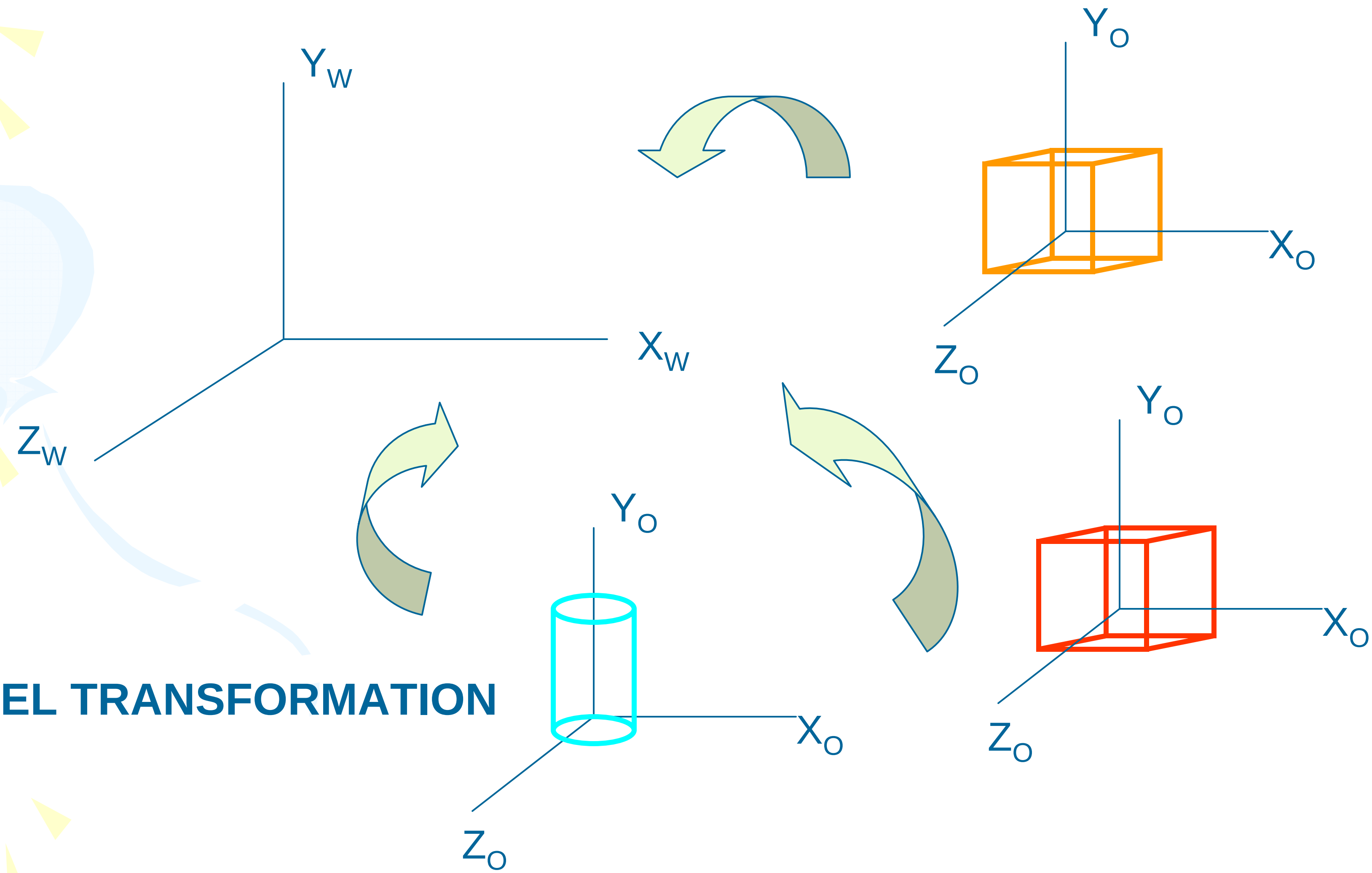
Transformations

- Consider triangles with only color attributes
  - 4D homogeneous vertices V
- Model Transformation (M – 4x4)
  - Scene Building
- View Transformation (E – 4x4)
  - Scene Viewing (Extrinsic Parameter Matrix)
- Projection Transformation (P – 4x4)
  - Scene Projection (Intrinsic Parameter Matrix)
- Complete Transformation is PEMV

Model Transformation

- World Coordinates
- Object Coordinates

World and Object Coordinates



MODEL TRANSFORMATION

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

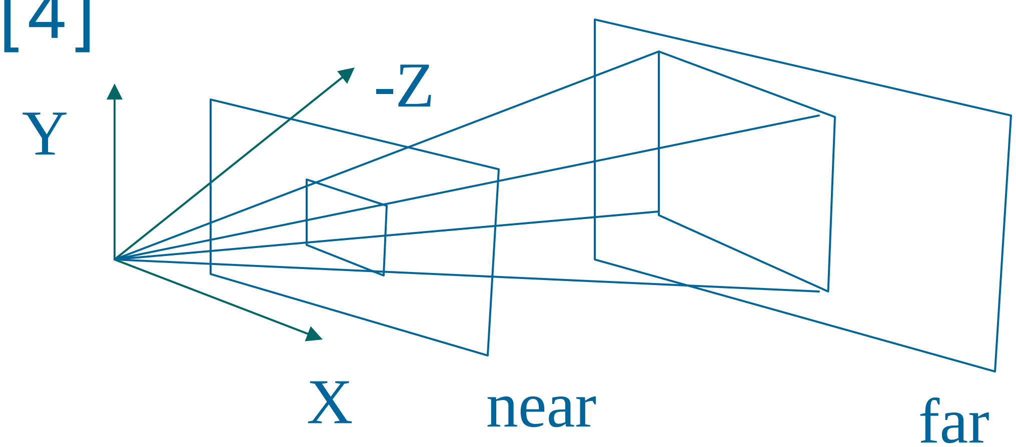
- Allows separation of concerns
  - When designing objects do not worry about scene
  - Create a library of objects
- Allows multiple instantiation by just changing the location, orientation and size of the same object

View Transformation

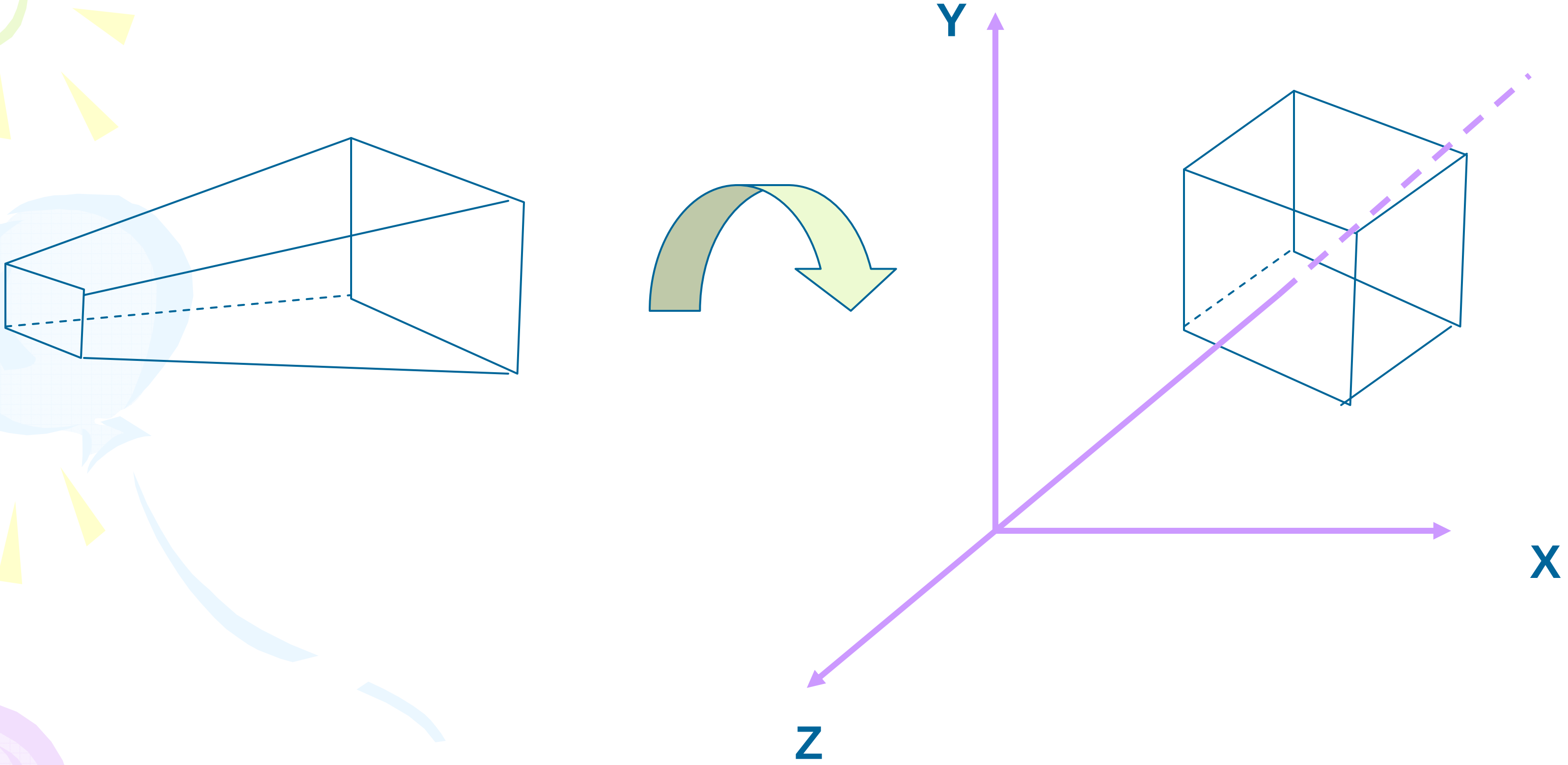
- Input
  - Position and orientation of eye (9 parameters)
    - View point (COP)
    - Normal to the image plane – N
    - View Up – U
- To align
  - Eye with the origin
  - Normal to the image plane with negative Z axis
  - View Up vector with positive Y axis
  - Can be achieved by rotation and translation

Projection Transformation

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



Projection Transformation



Projection Transformation

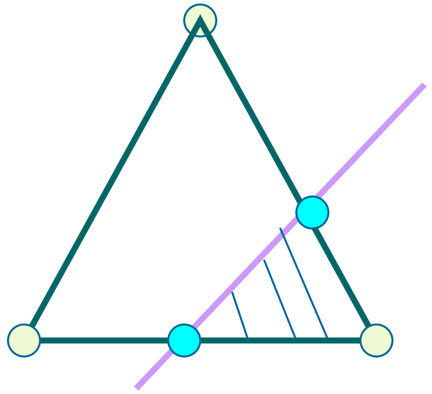
- Transforming the view frustum (along with the objects inside it) into a
  - cuboid with unit square faces on the near and far planes
  - the negative Z axis passes through the center of these two faces.
  - Projecting the objects on the near plane
- Consists of a shear, scale and perspective projection

Window Coordinate Transformation

- Scale XY coordinates of unit cuboid to reflect size of window (relative pixel coordinates)
- Translate these coordinates to the position of the window on the monitor screen to represent the absolute pixel coordinates.
- Z value is used for resolving occlusion

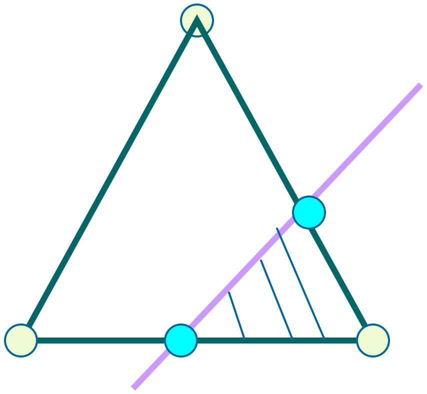
Clipping (3D or 2D)

- Removing the part of the polygon outside the view frustum
- If the polygon spans inside and outside the view frustum
  - introduce new vertices on the boundary



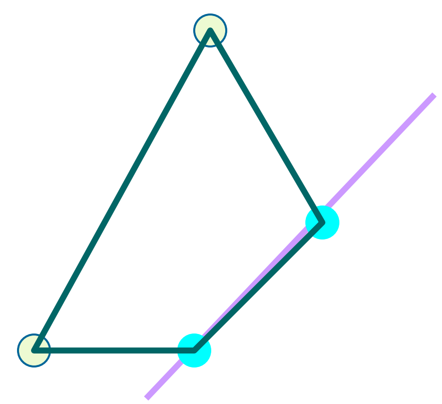
Interpolation of Attributes

- For the new vertices introduced
  - –compute all the attributes
  - –Using interpolation of the attributes of all the original vertices



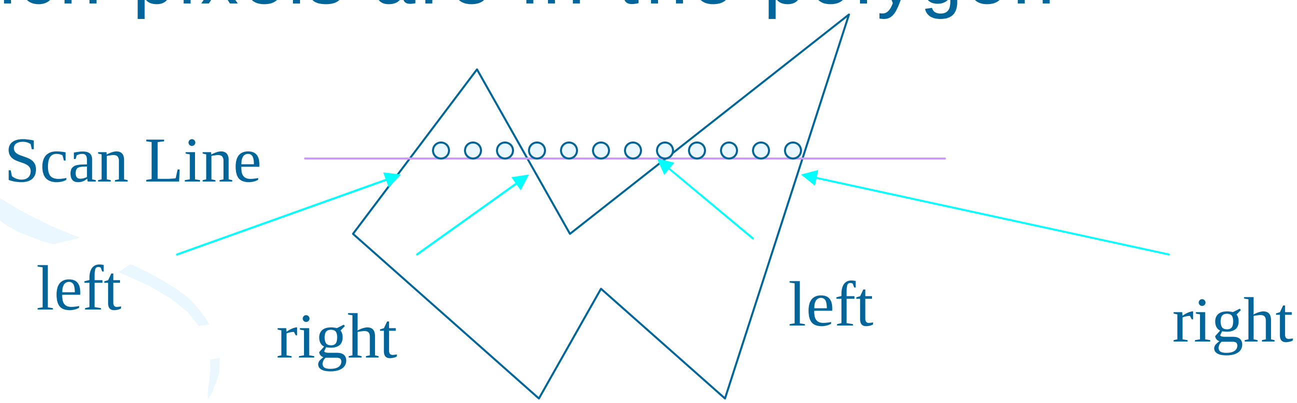
Interpolation of Attributes

- For the new vertices introduced
  - –compute all the attributes
  - –Using interpolation of the attributes of all the original vertices



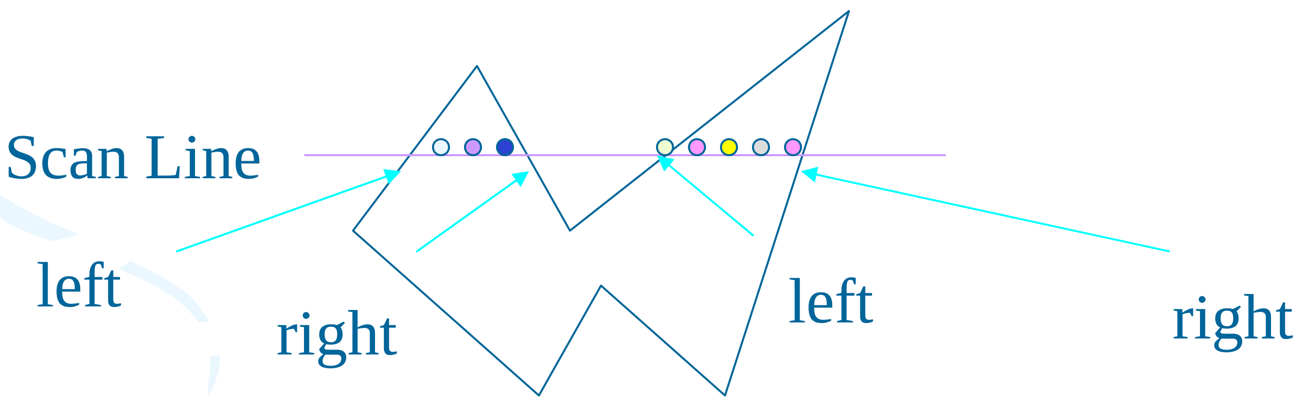
Rasterization

- Process of generating pixels in the scan (horizontal) line order (top to bottom, left to right).
  - –Which pixels are in the polygon



Interpolation of Attributes

- Interpolate the colors and other attributes at pixels from the attributes of the left and right extent of the scan line on the polygon edge.
- Also in scan line order



OpenGL

- A library that allows you to do several graphics related operations
  - E.g. All the transformations
  - Prefix gl
    - e.g. gltranslate, glscale
- Glut
  - Wrapper around gl
  - Calls gl functions
  - Prefix glu

Model Transformation

- We like to premultiply
  - A(B(CP))
- Advantage
  - Global Coordinate system
  - Intuitive for humans
- Disadvantage
  - Start with P, keep premultiplying
  - Do once for every point

OpenGL

- ((AB)C)P
- Post-multiplies
- Generates matrix once
- Then post-multiplies once with all points
- Assumes local coordinate system

Be careful

- You want to
- Rotate object
- Translate it
- Scale it
- STRP

- Write OpenGL
- Rotate object
- Translate it
- Scale it
- RTSP

WRONG!

Be careful

- You want to
- Rotate object
- Translate it
- Scale it
- STRP

- Write OpenGL
- Scale object
- Translate it
- Rotate it
- STRP

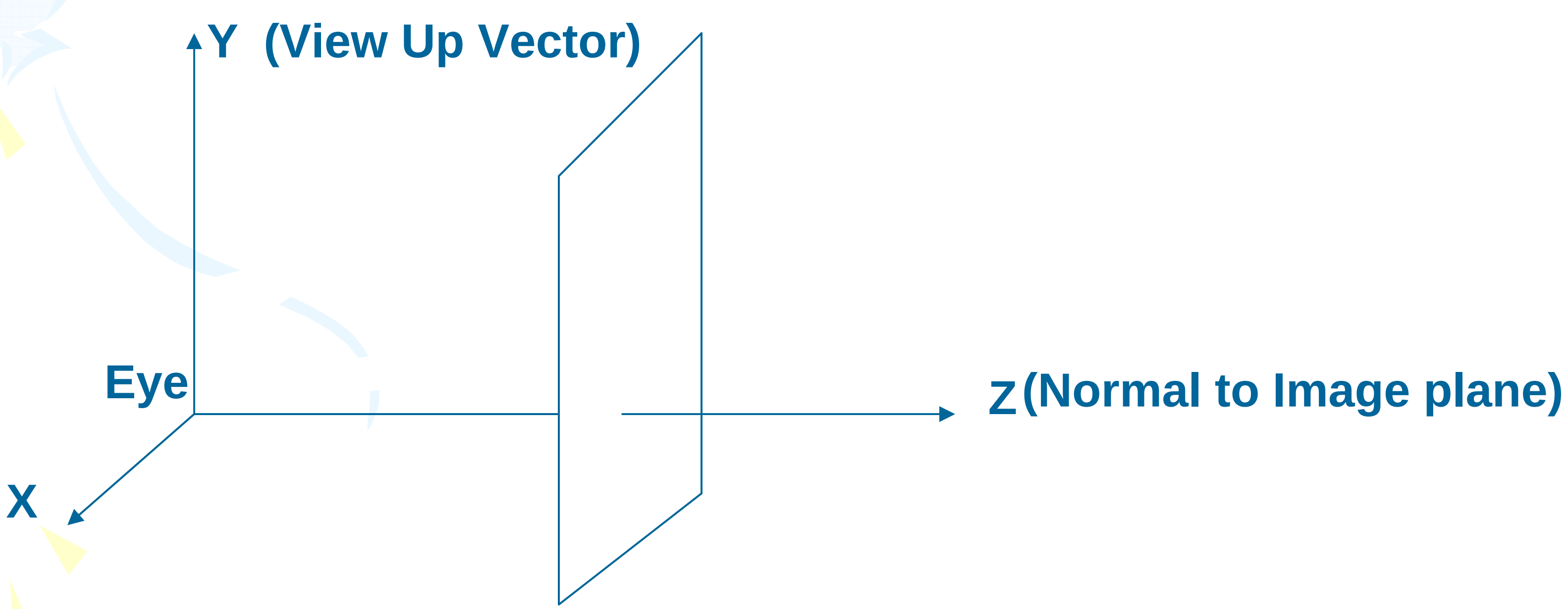
RIGHT!

View Transformation

- View is changing interactively
- Change of view is a complementary change of the model
- OpenGL assumes a default view
- Find transformation to match view to the default
- Apply this to the model
  - This is the complementary transformation

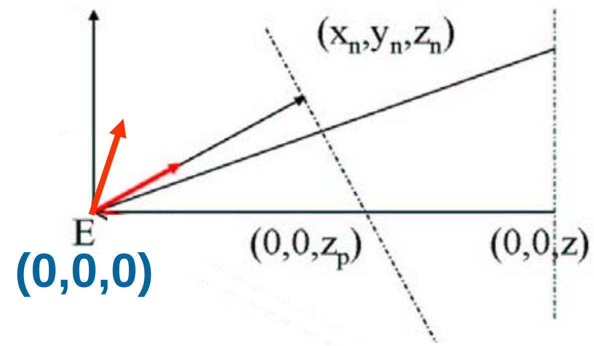
Default OpenGL View

- Eye at Origin
- Image plane perpendicular to negative Z
- View Up Vector coincident with Y



View Transformation

- Eye at $E=(x_0,y_0,z_0)$
- Normal to image plane is not Z, but arbitrary N
  - Normal meets image plane at (x_n,y_n,z_n)
- View Up V is not Y
  - Not perpendicular to N
- Transformation to default OpenGL View

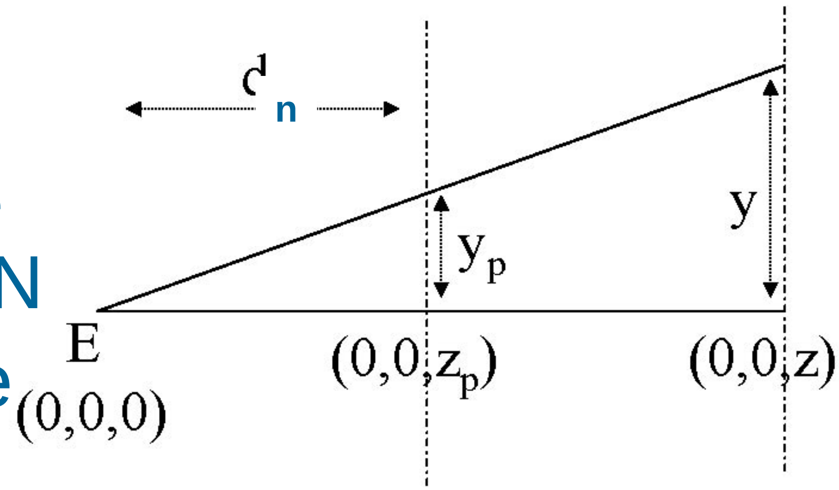


T(-E)

$u'_z = N/|N|$
$u'_x = (V/|V|) \times u'_z$
$u'_y = u'_z \times u'_x$

View Transformation

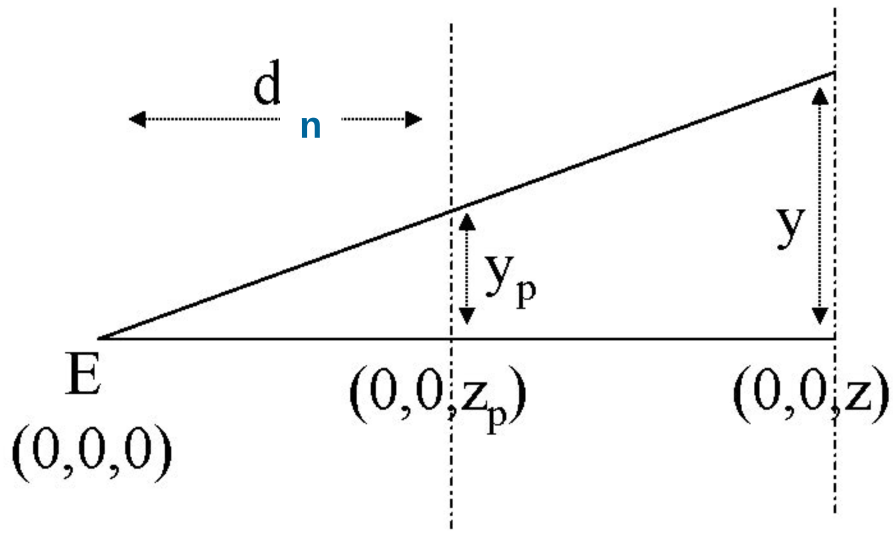
- Eye at E=(x0,y0,z0)
- Normal to image plane is not Z, but arbitrary N
  - Normal meets image plane at (xn,yn,zn)
- View Up V is not Y
  - Not perpendicular to N
- Transformation to default OpenGL View



R(N,V).T(-E)

gluLookAt

- gluLookAt
  - Eye coordinate (E)
  - Look At vector – where normal meets the plane
    - Find N and n
  - View Up Vector (V)
- Generates this matrix and postmultiplies it modelview matrix

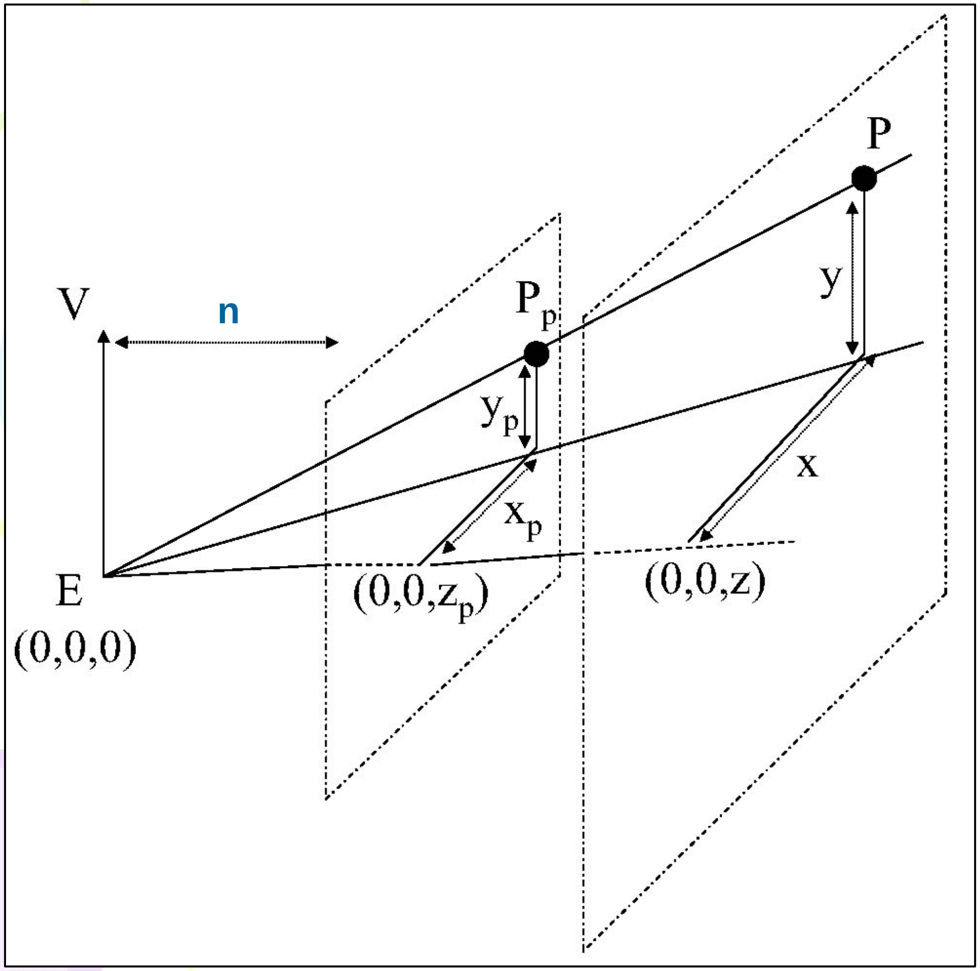


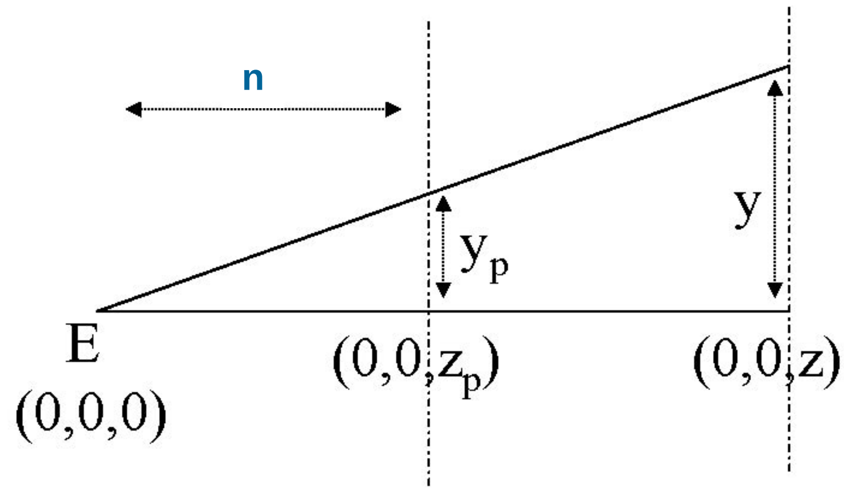
$$\underbrace{R(N,V).T(-E)}$$

Have you seen this before?

- Multiplication of rotation and translation matrix
- [R | RT] = RT
- This is the same as the extrinsic parameter matrix we saw in the pin-hole camera model

Perspective Projection

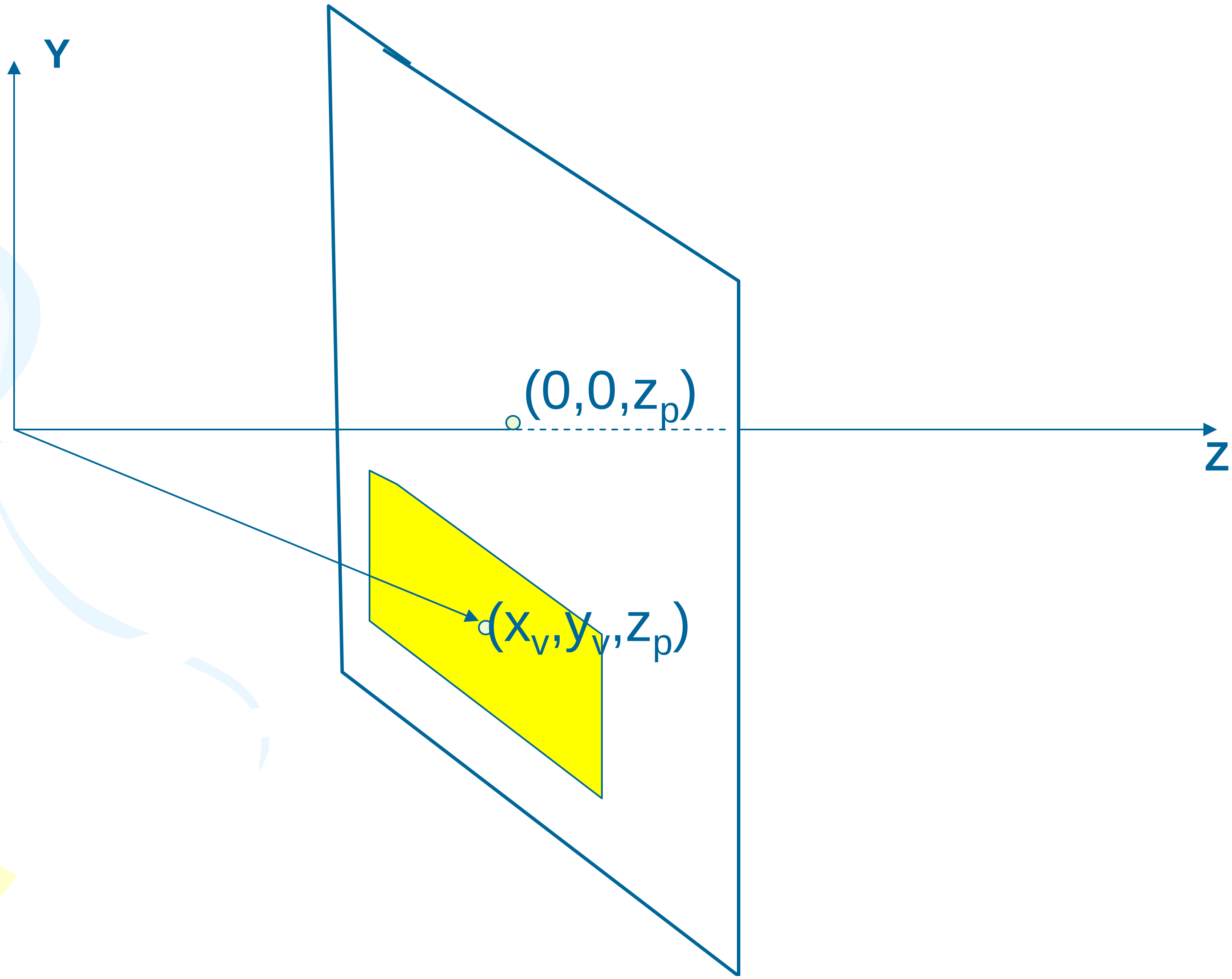




$$x_p/x = y_p/y = z_p/z$$

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

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

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

Naming confusions

- View direction
- Gaze direction
- OpenGL calls
  - Normal (Head direction)
  - View direction

Following Shear

- Center of image plane is (0,0)
- l = -r
- t = -b

Now normalize X and Y

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

Together

- Proj. Scale. Shear

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

Together

- Remove 3rd row and 4th col
- Looks like K
  - n is focal length
  - r+l is change of center
  - r-l is inversely proportional to number of pixels

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

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

What do we do?

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

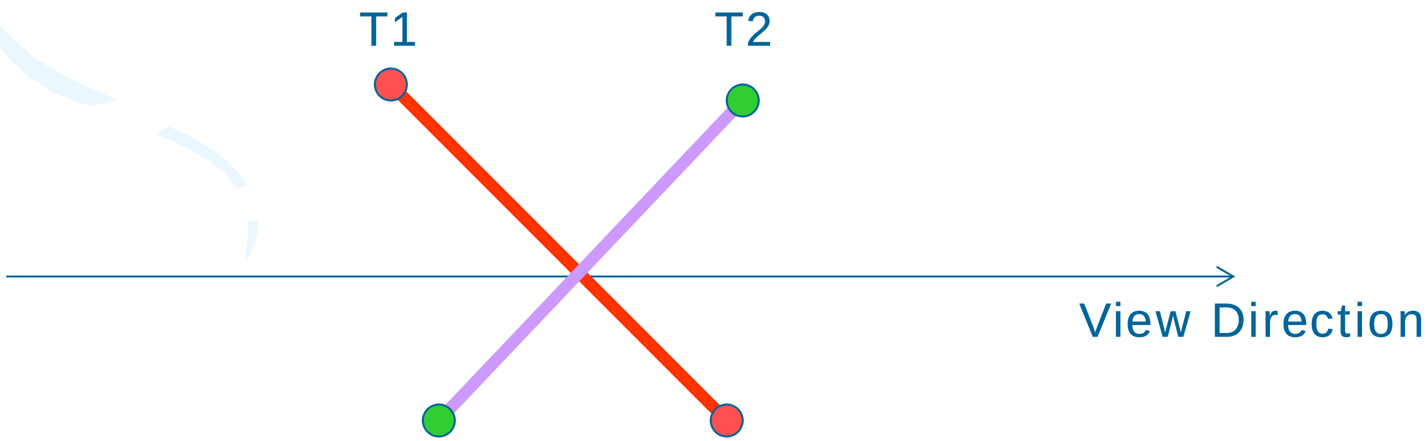
This is the correct perspective transform

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

We would like to retain the value of z. We are only changing the value of z, which is anyway not useful for 2D image generation using perspective projection.

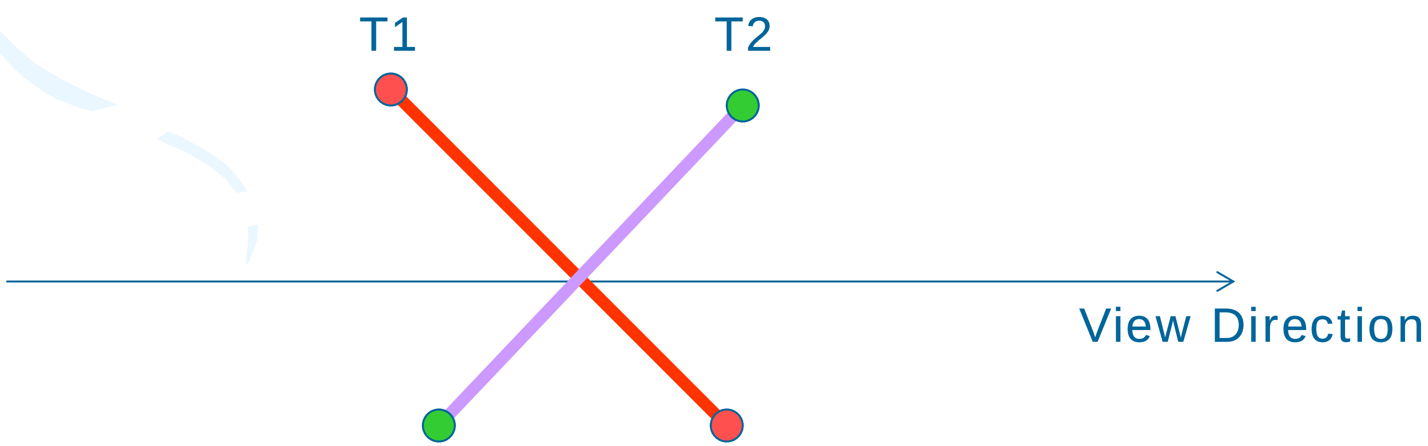
How do we use the z?

- 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



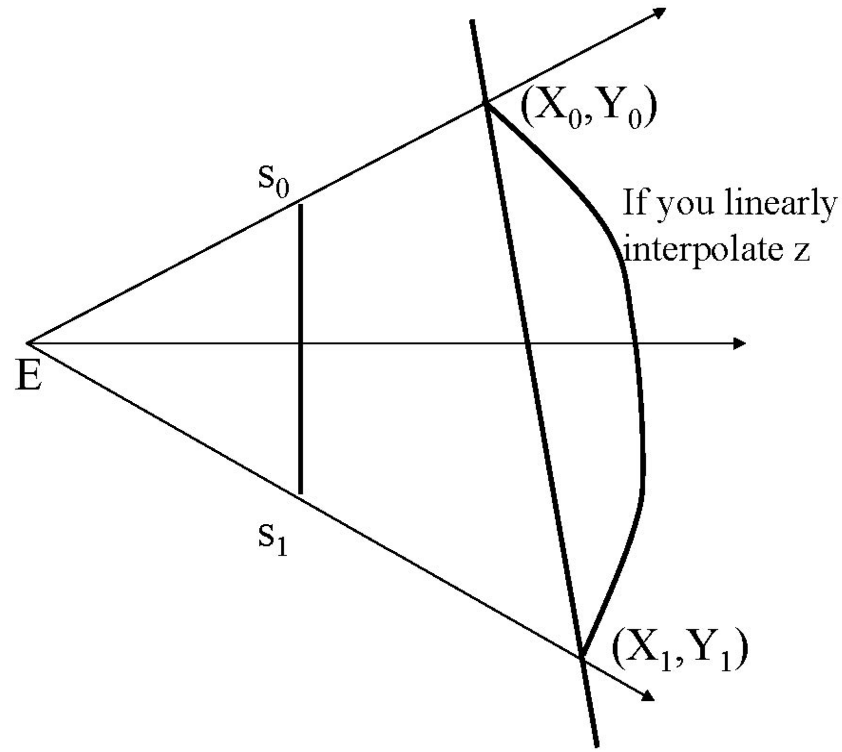
How do we use the z?

- 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)



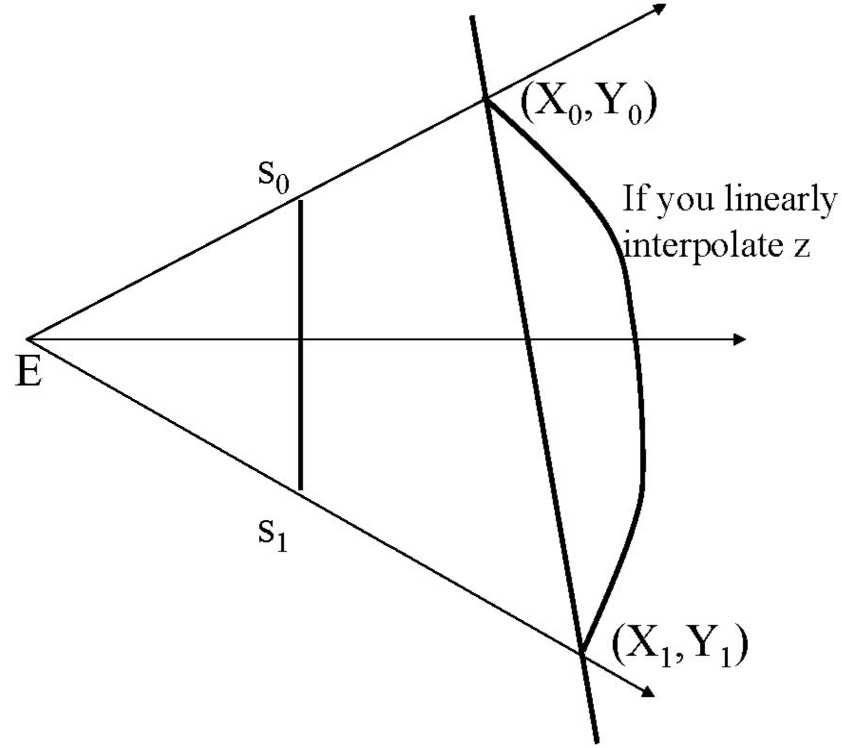
Screen Space Interpolation

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



What is correct?

- 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

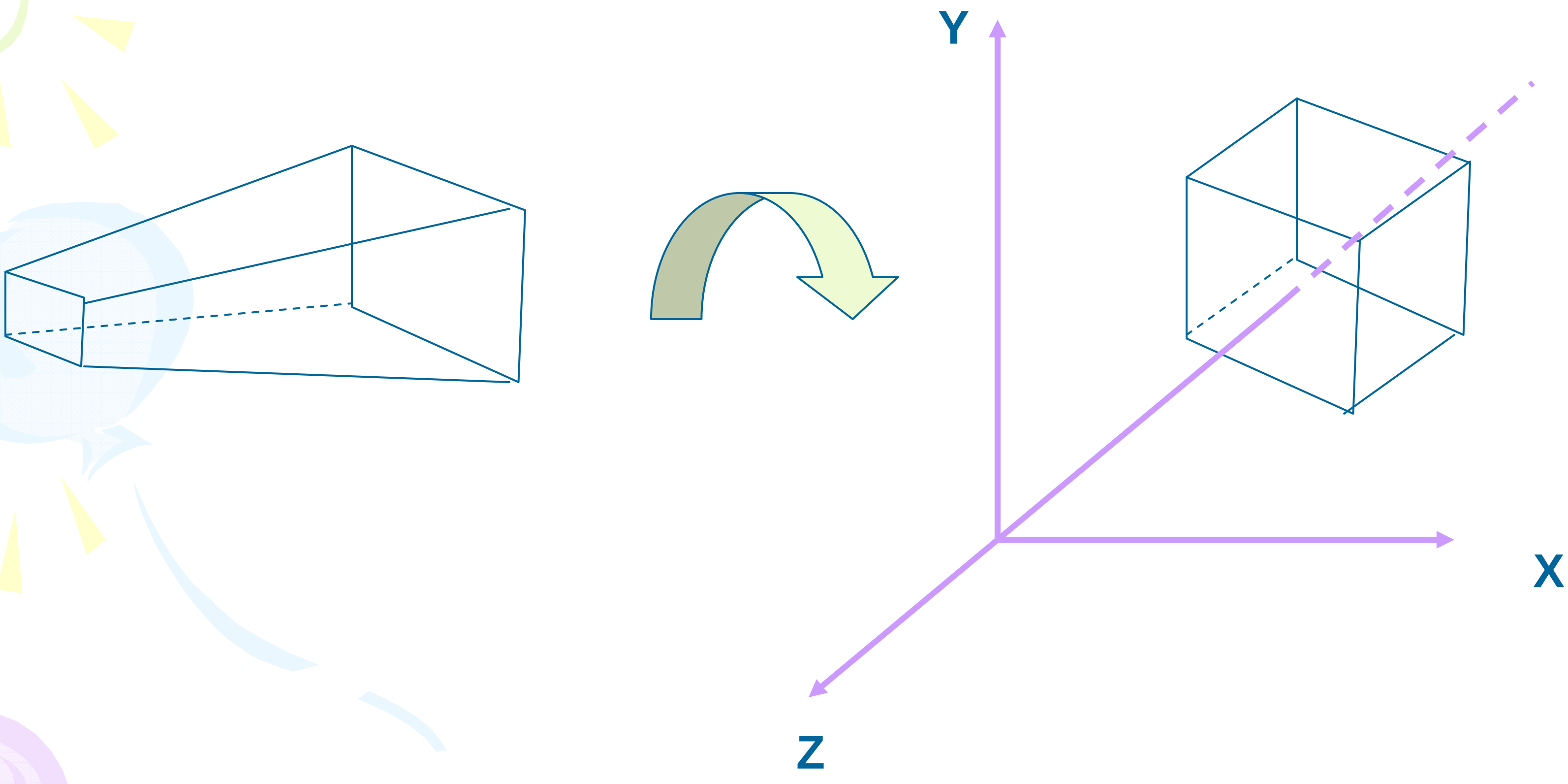
Bounding Z

- 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

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

OpenGL stack of matrices

- Two matrices
  - MODELVIEW (M) and PROJECTION (P)
  - $PM(x,y,z,1)^T$
- Two stacks
- All operations carried out on the top of the stack by postmultiplying
  - Push and Pop Matrices
- Why?
  - Huge advantage in animation applications

Red Book is the answer

- http://www.opengl.org/documentation/red_book/
- Now online
- Provide you with template
- You have to just insert code to make it work