LECTURE 4: THE GEOMETRY OF VIRTUAL WORLDS

Ehsan Aryafar

earyafar@pdx.edu

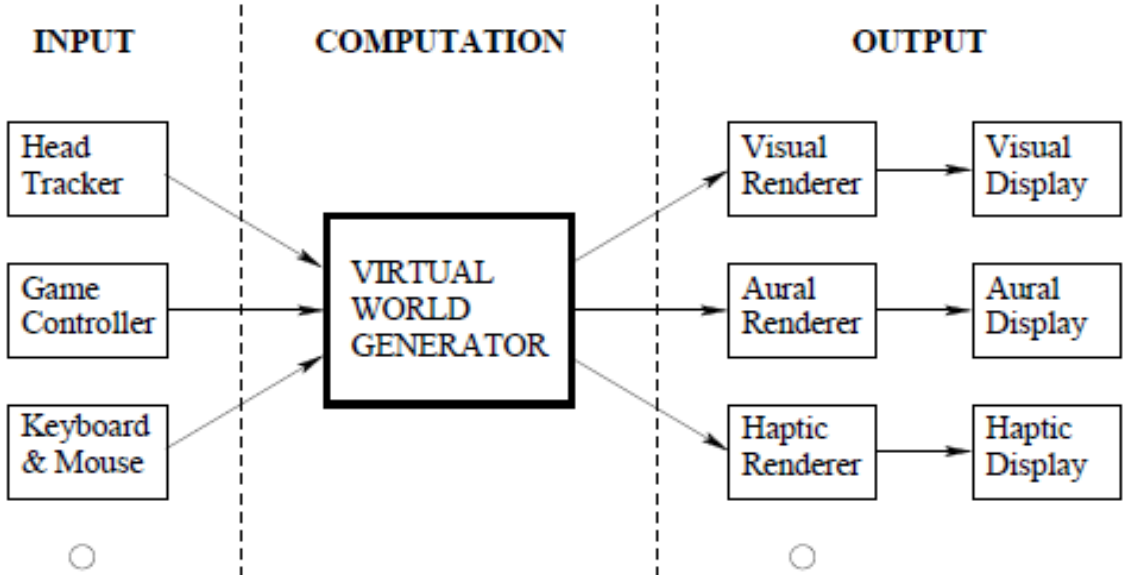
http://web.cecs.pdx.edu/~aryafare/VR.html

Recall: Components of VR Hardware

- Displays (outputs)
  - Devices that each stimulate a sense organ
- Sensors (input)
  - Devices that extract information from the real world
- Computers
  - Devices that process input and output

Recall: Software

- Industry is moving towards full-fledged VR engines
  - Similar to game engines
    - Game engine: a software-development environment designed for people to create video games, which provides the following functionalities: 2D/3D rendering and management of memory, sound, networking, AI, threading, etc.
  - SDKs are being developed for particular headsets
    - Handle low level operations, e.g., device derivers, head tracking, and display

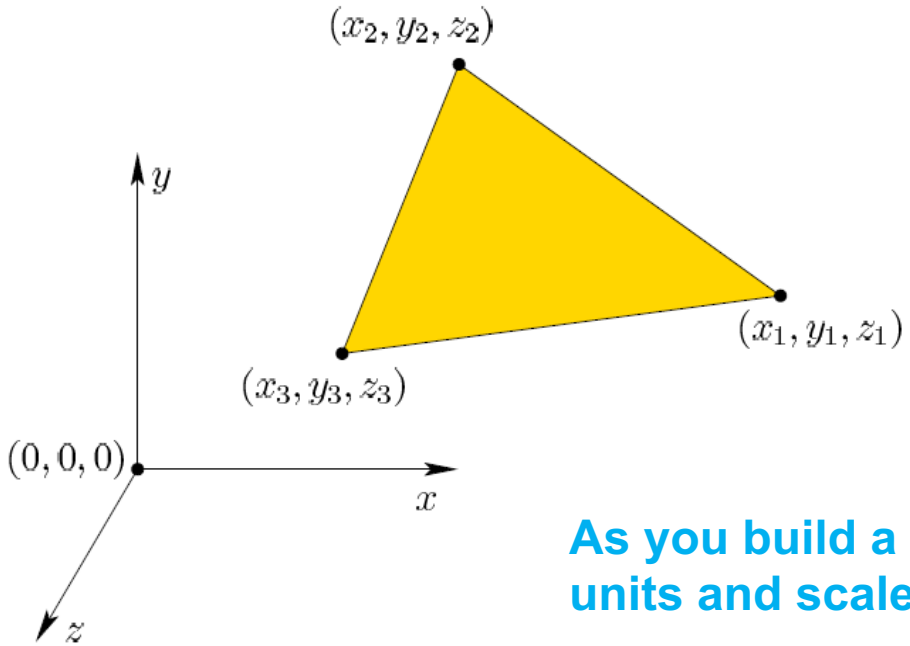


Outline

- The kind of transformation we need to setup VWG
  - **Geometry** and physics has to look reasonable to your brain
    - Assuming you are not targeting a particular experiment
  - **This lecture: Where virtual objects (i.e., rigid bodies) are placed on the display!**

Geometric Model of a Virtual World

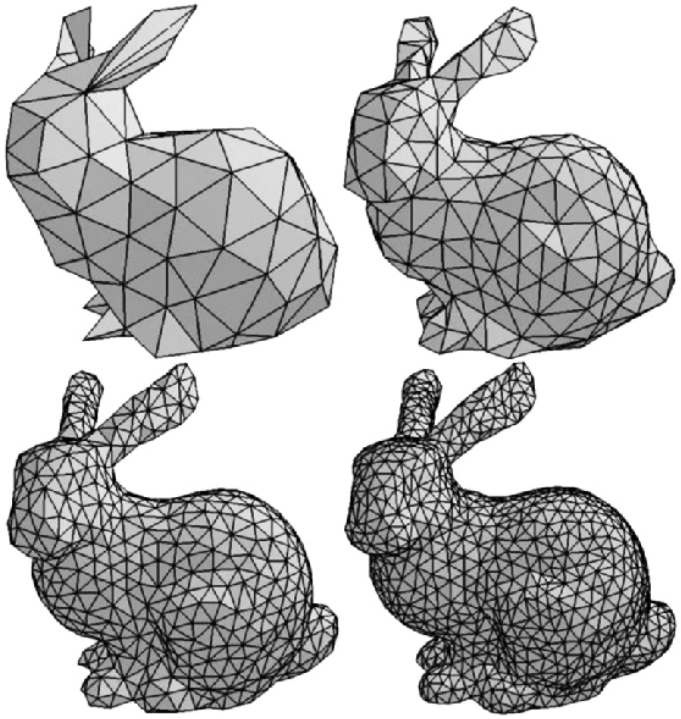
- We assume a 3D virtual world in Euclidean space
  - Each point is represented as a triple of real-valued coordinates (x, y, z)
  - We consider a right-hand coordinate system
  - Models are objects placed in the virtual world
  - Models can be fixed (walls) or movable (bullet)



As you build a virtual world, try to keep the units and scale like the real world!

How To Represent Geometric Models

- Geometric models are solid regions in 3D space
  - Geometric models are represented in terms of primitives, the simplest form is a 3D triangle
    - Triangles are heavily used, e.g., for ease of manipulation on GPUs



3D mesh-triangles with different resolutions!

Viewing the Models

- One of the most important aspects of VR is how the models are going to look when viewed on a display, which has two parts:
  - **Where** the points in the virtual world should appear on the display
    - Topic of this lecture
  - How should each part of the model appear after lighting and other impacts
    - **This is called rendering and will be covered later**

Changing Position

- Consider the following triangle (three vertices defined)

$$((x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3))$$

- What does the following operation do:

$$\begin{aligned}(x_1, y_1, z_1) &\mapsto (x_1 + x_t, y_1 + y_t, z_1 + z_t)\\ (x_2, y_2, z_2) &\mapsto (x_2 + x_t, y_2 + y_t, z_2 + z_t)\\ (x_3, y_3, z_3) &\mapsto (x_3 + x_t, y_3 + y_t, z_3 + z_t)\end{aligned}$$

Changing Position

- Consider a triangle model (VR object) with the following three coordinates for its vertices

$$((x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3))$$

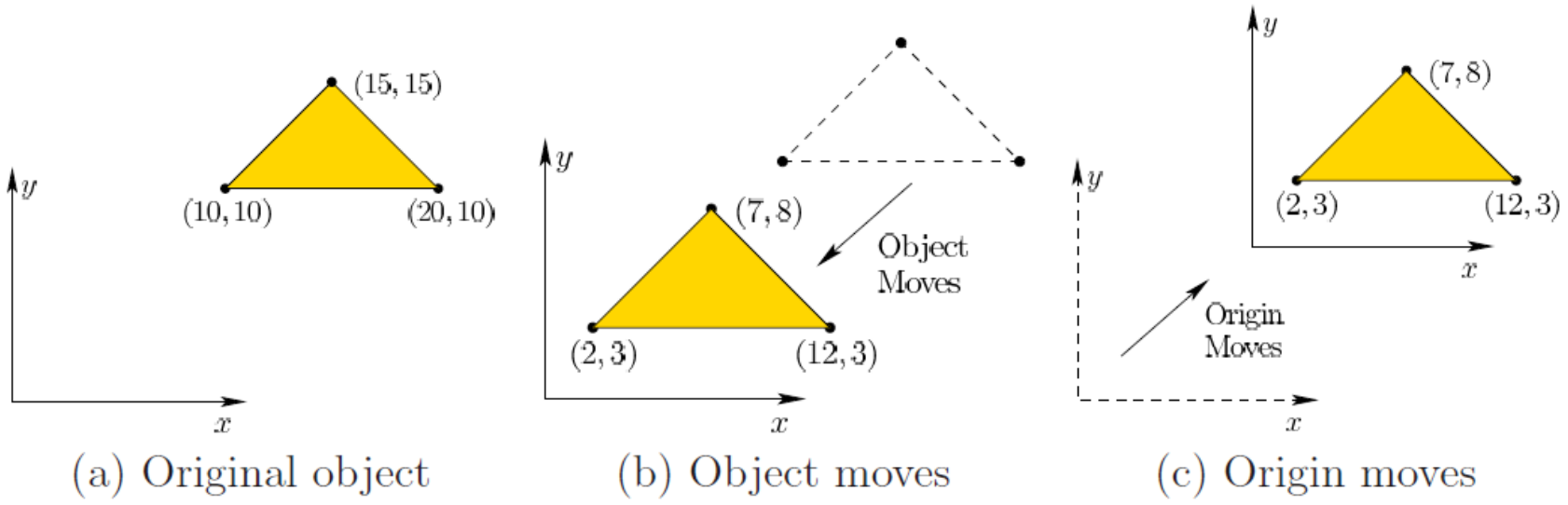
- What does the following operation do:

$$\begin{aligned}(x_1, y_1, z_1) &\mapsto (x_1 + x_t, y_1 + y_t, z_1 + z_t)\\ (x_2, y_2, z_2) &\mapsto (x_2 + x_t, y_2 + y_t, z_2 + z_t)\\ (x_3, y_3, z_3) &\mapsto (x_3 + x_t, y_3 + y_t, z_3 + z_t)\end{aligned}$$

This operation of changing position is called **translation**!

Translation

- Two interpretations: object moves, or the origin is changed



In the example above, x_t, y_t, and z_t are -8, -7, and 0, respectively!
As if you moved the origin to the point (8, 7, 0)

Remember the "changing the origin" interpretation!

Rotation

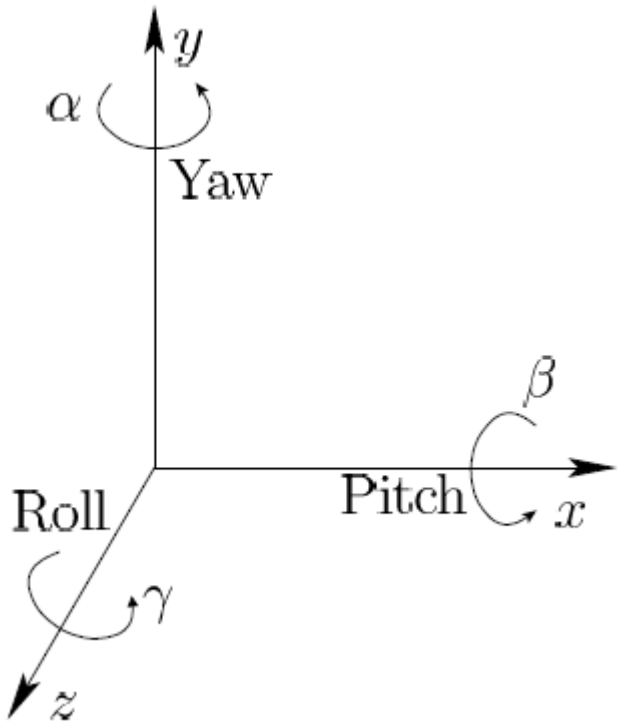
- Any 3D rotation can be described as a sequence of yaw, pitch, and roll rotations!

To rotate a point (x, y, z), you just multiply it by the appropriate rotation matrix!

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_x(\beta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta \\ 0 & \sin\beta & \cos\beta \end{bmatrix}$$

$$R_y(\alpha) = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix}$$



Matrix Multiplication

- What does the following sequence of matrix multiplication do to a 3D point?

$$R(\alpha, \beta, \gamma) = R_y(\alpha) R_x(\beta) R_z(\gamma)$$

- What does the following operation do to the point (x, y, z)?

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix}$$

- What if you do the above operation to every point (x, y, z) (e.g., every object) in the world?

4x4 Matrix Representation

- In computer graphics, we routinely use 4x4 matrices
  - For a variety of industrial and technical reasons, e.g., ease of operation
- What does the operation below do:

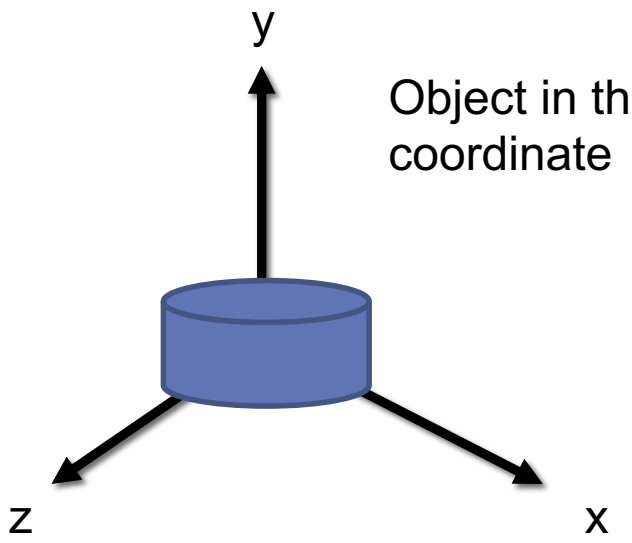
$$\left[\begin{array}{ccc|c} & R & & \begin{matrix}x_t\\y_t\\z_t\end{matrix} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

Rigid body translation T_{rb}

Side note: When R is a rotation matrix, then the **inverse matrix and transpose matrix would be equal.** Let I be the identity matrix, then we have:

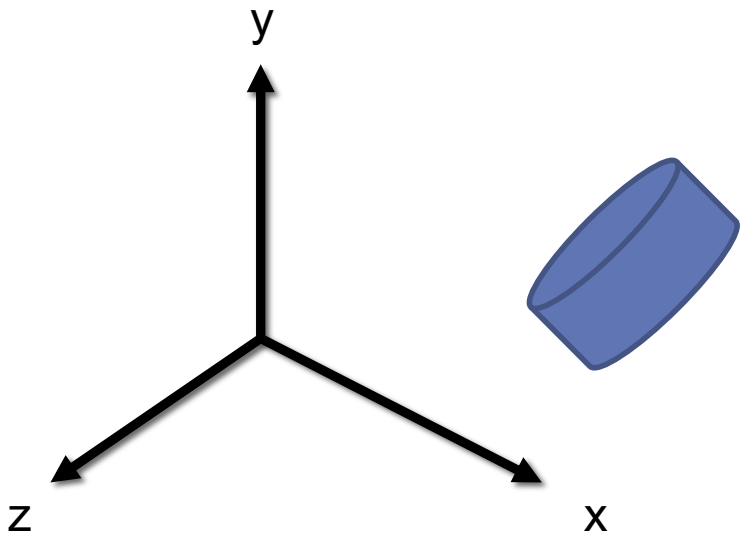
$$R^{-1} = R^T$$
$$R^T R = I$$

Moving an Object from Local Coordinate to Virtual World Coordinate



Object in the local coordinate

Placing object in the VR world



Undoing Operations

- Suppose we want to undo the rigid body translation from the previous slide. Which of the below two does that?

$$\left[\begin{array}{ccc|c} & R^T & & \begin{matrix}-x_t\\-y_t\\-z_t\end{matrix} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

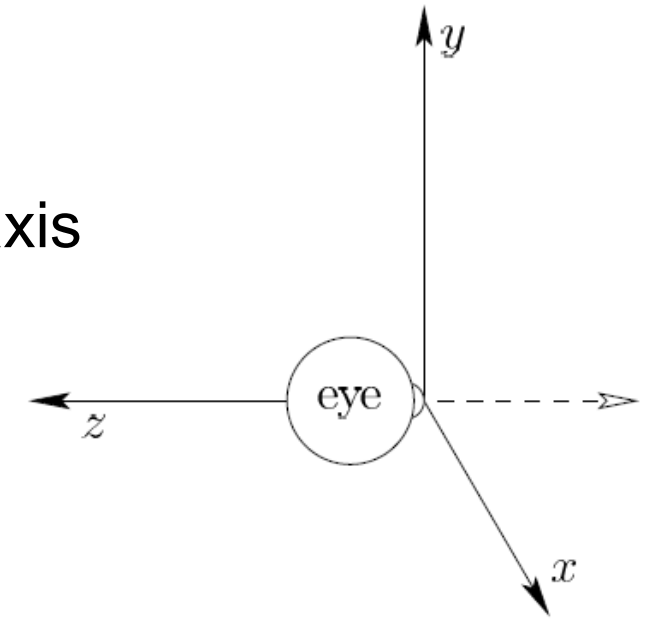
$$\left[\begin{array}{ccc|c} & R^T & & \begin{matrix}0\\0\\0\end{matrix} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} 1 & 0 & 0 & -x_t \\ 0 & 1 & 0 & -y_t \\ 0 & 0 & 1 & -z_t \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Viewing Transformations

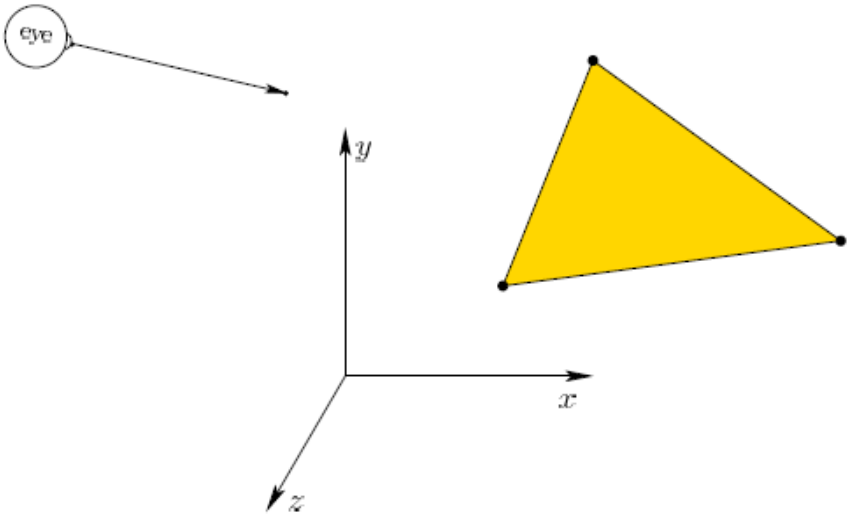
- How to transform the models in the virtual world so that they appear on a virtual screen
  - **Think of it as a transformation pipeline**
  - Rendering later adds effects due to lighting and quantization
    - **Determining the exact RGB value for each pixel**
  - The result ultimately appears on the physical screen
- Assume a virtual camera in the virtual world
  - **What should the virtual picture taken by camera look like?**
  - **Camera would be one of the virtual human eyes**

Virtual Camera Viewing

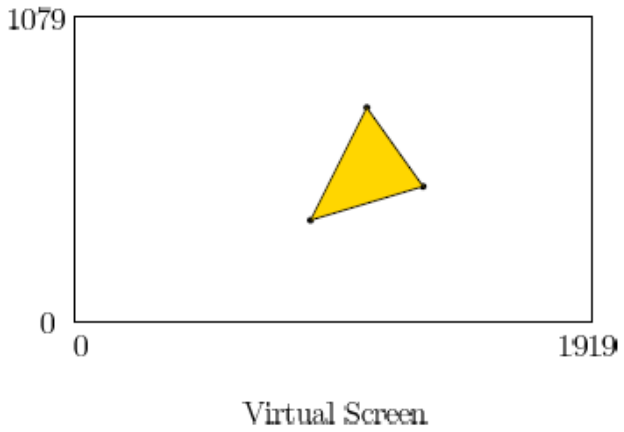
Consider an eye looking down the z axis



Eye looking at the virtual world!



1080p display

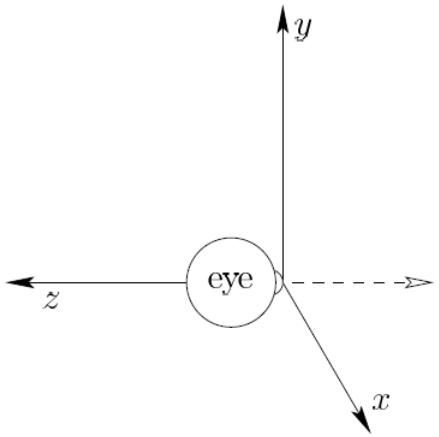


In VR the position and look out of eye is given by tracking systems or controller inputs

Eye Translation

- Suppose that eye is an object model that we want to place at the virtual world position (e_1, e_2, e_3) and orientation given by matrix

$$R_{eye} = \begin{bmatrix} \hat{x}_1 & \hat{y}_1 & \hat{z}_1 \\ \hat{x}_2 & \hat{y}_2 & \hat{z}_2 \\ \hat{x}_3 & \hat{y}_3 & \hat{z}_3 \end{bmatrix}$$



If we apply the following transformation to all the objects in the virtual world, what does that mean?

$$T_{eye} = \begin{bmatrix} \hat{x}_1 & \hat{x}_2 & \hat{x}_3 & 0 \\ \hat{y}_1 & \hat{y}_2 & \hat{y}_3 & 0 \\ \hat{z}_1 & \hat{z}_2 & \hat{z}_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -e_1 \\ 0 & 1 & 0 & -e_2 \\ 0 & 0 & 1 & -e_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Eye Position and Gaze

- Eye position (**e**), gaze, and up is given by the VR trackers
  - Position of the eye: e
  - Looking direction of the eye: c (with unit size, i.e., vector size of 1)
  - Up direction: u (with unit size, i.e., vector size of 1)

- The orientation matrix is then derived based on the above information

$$\hat{z} = -\hat{c}$$

$$\hat{x} = \hat{u} \times \hat{z}$$

$$\hat{y} = \hat{z} \times \hat{x}$$

Not in Exam!

Cross product!

Chaining Transformation

- We now discuss the set of transformation to display geometric models on to a display (for each eye)
- Some of the matrices may appear unnecessarily complicated
  - **Design motivated by algorithm and hardware issues**
  - **Simplify/complicate things to make algebraic calculations efficient later**
- The chain generally appears as follows

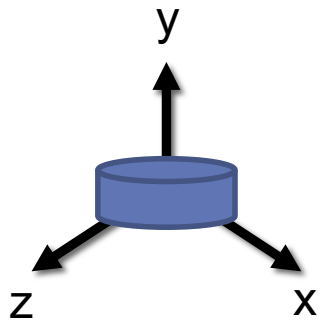
$$T = T_{vp} T_{can} T_{eye} T_{rb}$$

Applied to every point (x,y,z,1) or essentially model in the world!

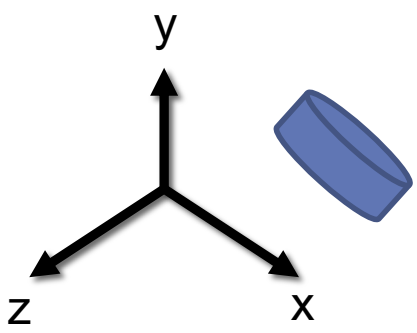
- **Remember that we added an auxiliary fourth dimension!**

Chaining Transformations

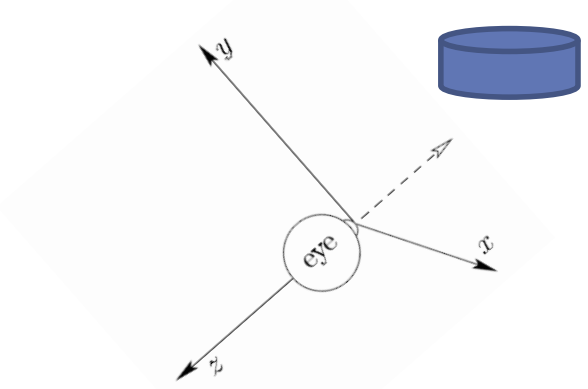
Objects in their local coordinates



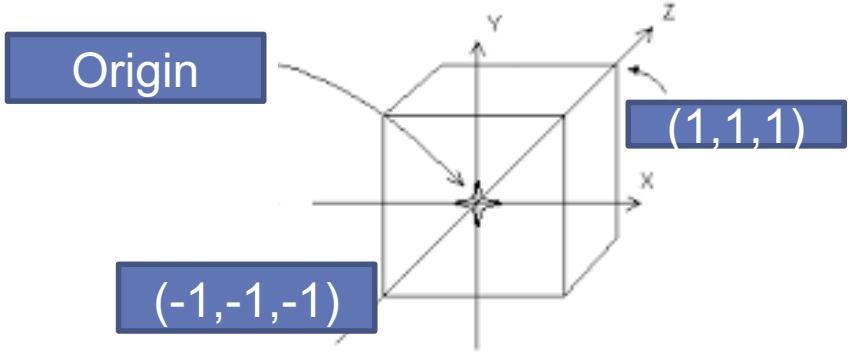
Place Objects in the virtual world



Bring the world into eye's perspective

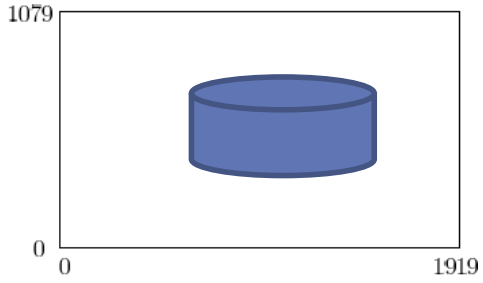


Form a viewing frustum (what goes on screen), scale it to a perfect cube, and center it at the origin. At this point, all distortions due to perspective projection are figured out



Canonical View Transform

Bring the x, y projected points to coordinates used to index pixels on a physical display



Viewport Transform
Pixel Coordinates

Accounting for Stereoscopic Viewing

- Left and right eyes have overlap but also distinct views
  - Our brain combines the images that each eye sees
  - Let t be the average distance between the left and right eyes
    - The average across humans is 0.064 meters
- We need to construct two virtual cameras for each eye
- The appropriate transform for left eye is

$$T = T_{vp}T_{can}T_{left}T_{eye}T_{rb}$$

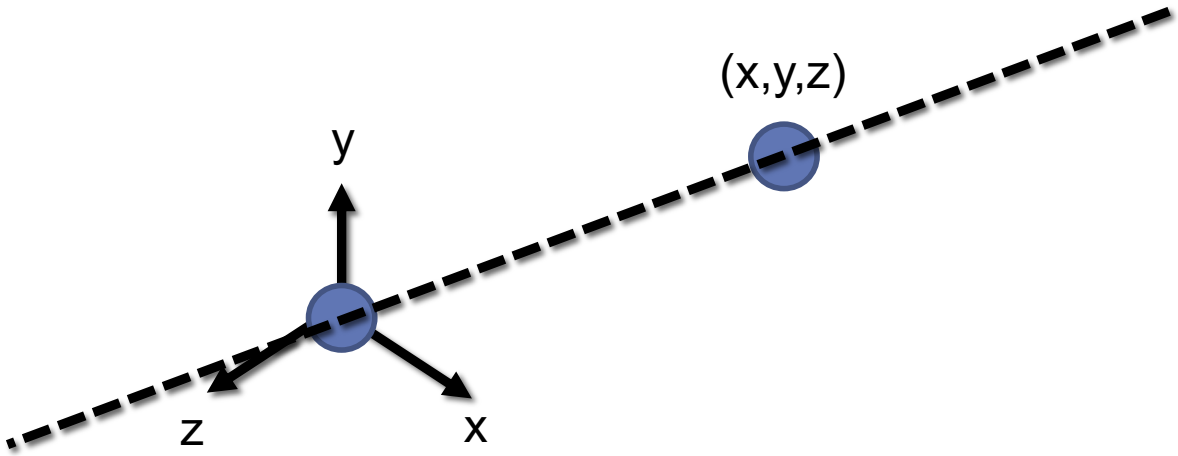
$$T_{left} = \begin{bmatrix} 1 & 0 & 0 & \frac{t}{2} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{right} = \begin{bmatrix} 1 & 0 & 0 & -\frac{t}{2} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective Projection

- Consider a point (x,y,z) in the virtual world
- What does the set (αx,αy,αz) show for all possible values of α?

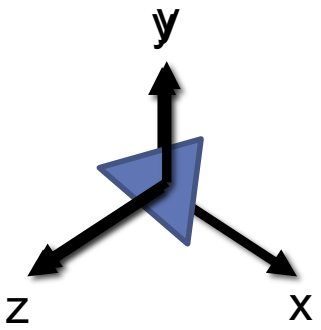
Perspective Projection

- Consider a point (x,y,z) in the virtual world
- What does the set (αx,αy,αz) show for all possible values of α?
  - **The line passing through origin and (x,y,z)**

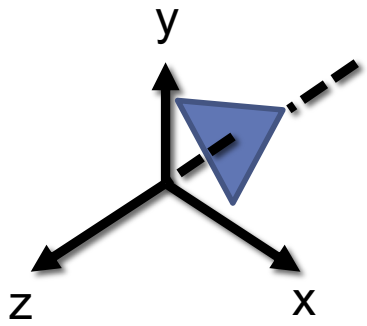


Perspective Projection

- Consider a triangle in the origin
- Let every (x,y,z) of the triangle is changed to (αx,αy,αz)
  - How does the new triangle look like (let α = 2)?



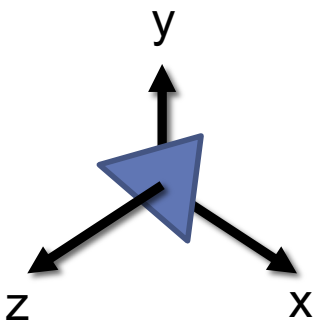
Triangle at z = 0



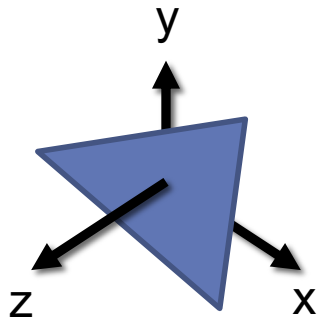
Triangle at z = -1

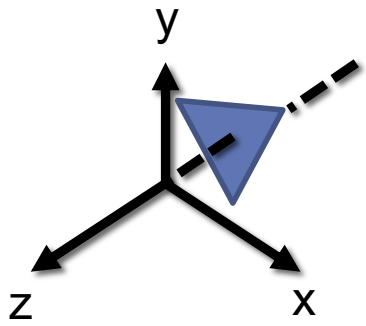
Perspective Projection

- Consider a triangle in the origin
- Let every of the triangle is changed to (αx,αy,αz)
  - How does the new triangle look like (let α = 2)?

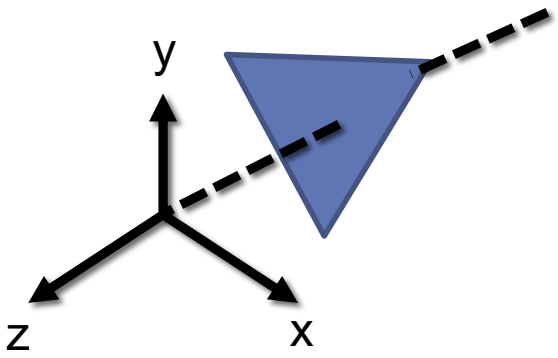


Triangle at z = 0



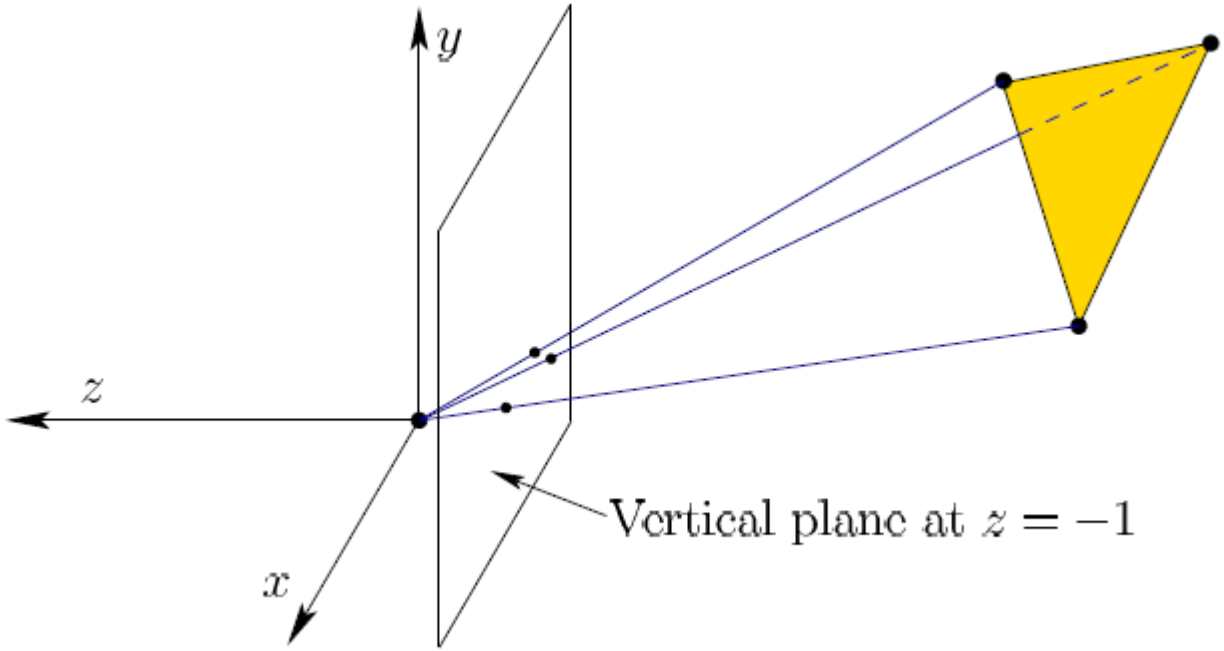


Triangle at z = -1

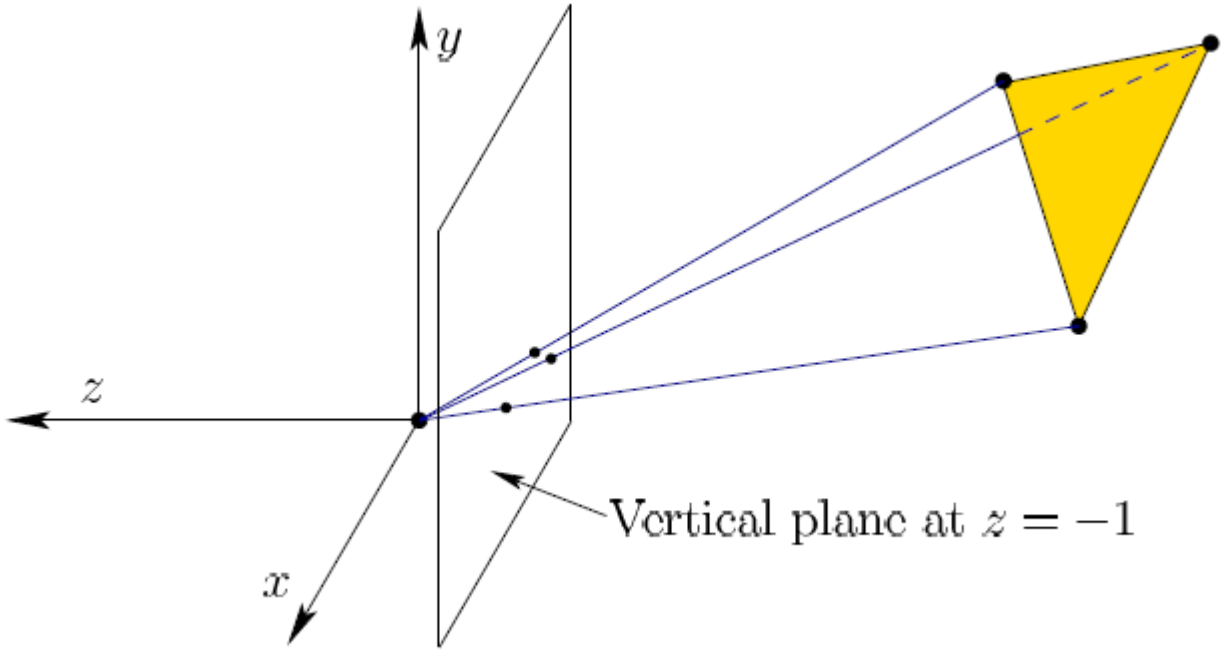


Triangle at z = -1

How to Project a Shape onto Vertical Plane at z = -1?



How to Project a Shape onto Vertical Plane at z = -1?



For each vertex of the triangle (x_1,y_1,z_1), find the alpha such that for $(\alpha x_1, \alpha y_1, \alpha z_1)$, $\alpha z_1 = -1$!

Projecting to z = n Plane!

- We can achieve this through the following transformation

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ nz \\ z \end{bmatrix}$$

- **From here on, the resulting 4D vector is interpreted as a 3D vector that is scaled by dividing out to its fourth component**
- The result of above transformation is interpreted as

$$(nx/z, ny/z, n)$$

Tracking Depth

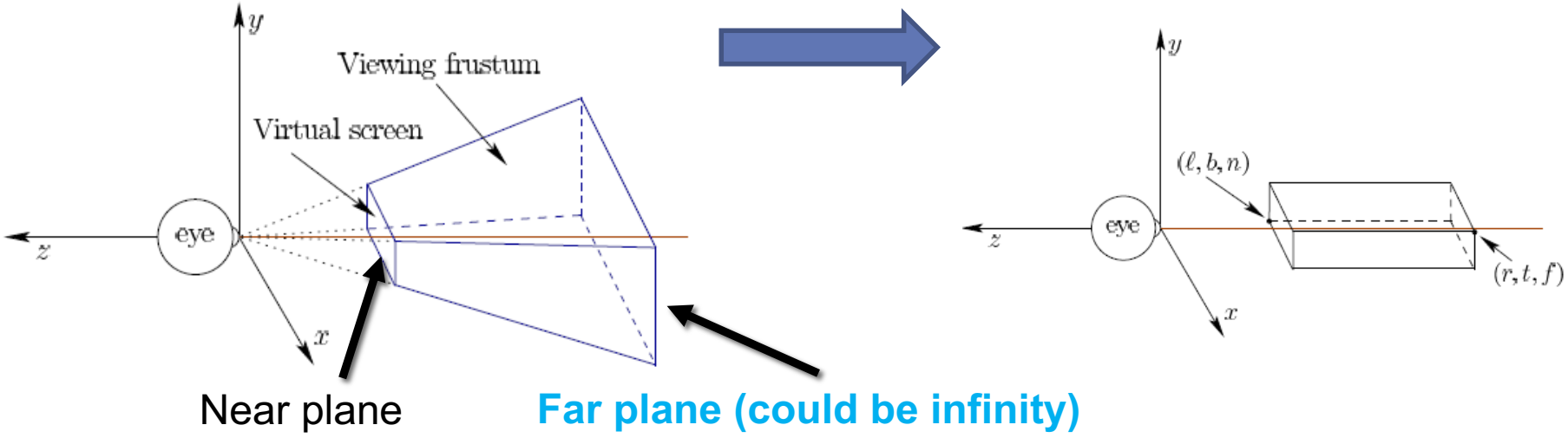
- The following transform is used in computer graphics

$$T_p = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad 0 < n < f$$

- It is unnecessary for the purpose of placing points on the virtual screen
  - The z coordinate is used for the purpose of tracking depth
    - Which object is in front of the other
    - If point p if further from the eye than q, then it remains so after T_p

Viewing Frustum

- Viewing frustum
  - **Region of space that may appear on the screen**
  - A volume in which objects are visible
    - **Items outside frustum are removed or clipped to remove unnecessary calculations**
- When T_p is applied to the view frustum, it turns it to



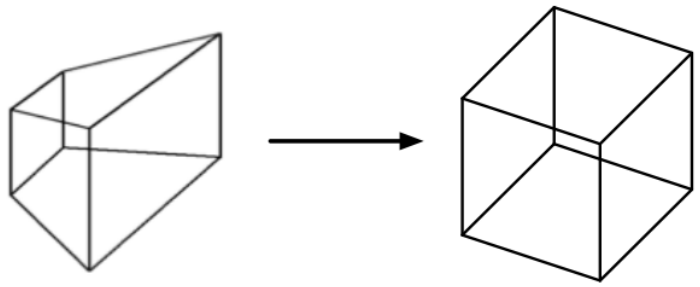
Canonical View Transform

- Is defined as

$$T_{can} = T_{st} T_p$$

- T_{st} is for translation and scaling
  - Places the box such that its center is origin and turns it into a cube with a width of 2
    - Coordinates of the corners are $(\pm 1, \pm 1, \pm 1)$

$$T_{st} = \begin{bmatrix} \frac{2}{r-\ell} & 0 & 0 & -\frac{r+\ell}{r-\ell} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Makes clipping much easier

Viewport Transform

- The last step is to bring the projected points to the coordinates used to index pixels on a physical display
- Note that before viewport, the coordinates range from-1 to 1
- Let m be # of horizontal pixels and n # of vertical pixels
  - N = 1080 and m = 1920 for a 1080p display
  - (0,0) is defined to be the lower left corner

$$T_{vp} = \begin{bmatrix} \frac{m}{2} & 0 & 0 & \frac{m-1}{2} \\ 0 & \frac{n}{2} & 0 & \frac{n-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$