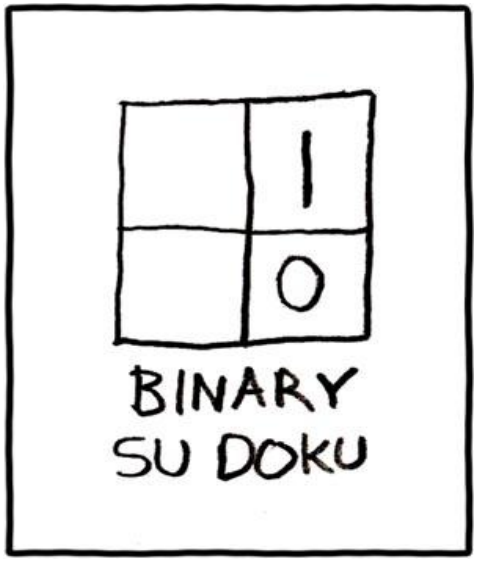


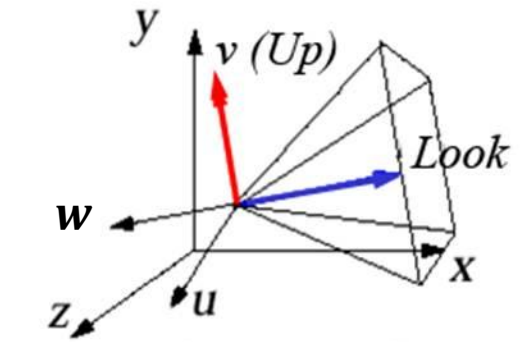
It looks like a matrix…
Sort of…

Viewing III

Projection in Practice

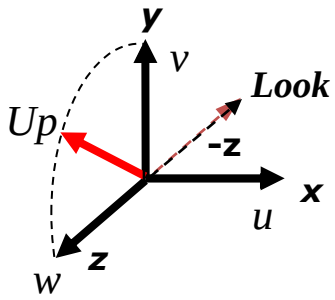
Arbitrary 3D views

- Now that we have familiarity with terms we can say that these view volumes can be specified by **placement** and **shape**
- **Placement:**
  - ***Position*** (a point)
  - ***Look*** and ***Up*** vectors
- **Shape:**
  - horizontal and vertical ***view angles*** (for a perspective view volume)
  - front and back clipping planes
  - Note camera coordinate system ***(u, v, w)*** is defined in the world ***(x, y, z)*** coordinate system



Arbitrary Perspective Frustum

Finding $\boldsymbol{u}$, $\boldsymbol{v}$, and $\boldsymbol{w}$ from Position, Look, and Up (1/5)

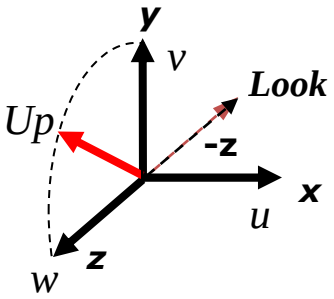


- We know that we want the $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$ axes to have the following properties:
  - our arbitrary *Look Vector* will lie along negative *w-axis*
  - a projection of *Up Vector* into plane defined by the w-axis as its normal will lie along the $\boldsymbol{v}$-axis
  - The $\boldsymbol{u}$-axis will be mutually perpendicular to the $\boldsymbol{v}$ and $\boldsymbol{w}$-axes, and will form a right-handed coordinate system
- Plan of attack: first find $\boldsymbol{w}$ from *Look*, then find $\boldsymbol{v}$ from *Up* and $\boldsymbol{w}$ vector, then find $\boldsymbol{u}$ as a normal to plane defined by $\boldsymbol{w}$ and $\boldsymbol{v}$

Finding $\boldsymbol{u}$, $\boldsymbol{v}$, and $\boldsymbol{w}$ (2/5)

- Finding $\boldsymbol{w}$
- Finding $\boldsymbol{w}$ is easy. *Look vector* in canonical volume lies on $-z$. Since z maps to $\boldsymbol{w}$, $\boldsymbol{w}$ is a normalized vector pointing in opposite direction from our arbitrary *Look vector.*

$$w = \frac{-Look}{\|Look\|}$$



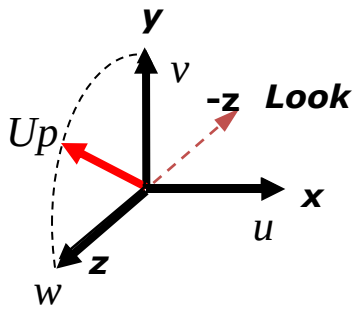
- Note that
  - *Up* and $\boldsymbol{w}$ define a plane
  - $\boldsymbol{u}$ is a normal to that plane
  - $\boldsymbol{v}$ is a normal to plane defined by $\boldsymbol{w}$ and $\boldsymbol{u}$

Finding u, v, and w (3/5)

- Finding v
- Problem: find a vector, v, perpendicular to w
- Solution: project out the w component of the *Up* vector and normalize

$$v = Up - (Up \bullet w)w$$

$$v = \frac{v}{\|v\|}$$



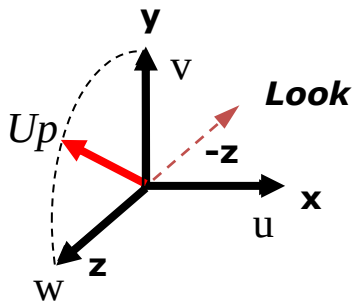
- w is unit length, but *Up* vector might not be unit length or perpendicular to w, so we have to remove the w component and then normalize

- By removing the w component from the *Up* vector, the resulting vector is the component of *Up* in a direction perpendicular to w

Finding $\boldsymbol{u}$, $\boldsymbol{v}$, and $\boldsymbol{w}$ (4/5)

- Finding $\boldsymbol{u}$
- We can use cross-product, but which one should we use?
  - $\boldsymbol{w} \times \boldsymbol{v}$ and $\boldsymbol{v} \times \boldsymbol{w}$ are both perpendicular to the plane, but in different directions . . .
- Answer: cross-products are right-handed, so use $\boldsymbol{v} \times \boldsymbol{w}$ to create a right-handed coordinate frame

$$u = v \times w$$



- As a reminder, the cross product of two vectors a and b is:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix}$$

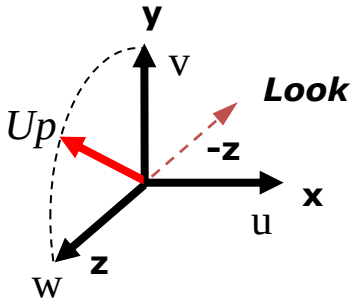
Finding u, v, and w (5/5)

- To Summarize:

$$w = \frac{-Look}{\|Look\|}$$

$$v = \frac{Up - (Up \bullet w)w}{\|Up - (Up \bullet w)w\|}$$

$$u = v \times w$$



- Given camera coordinate system, how to calculate projection?

The canonical view volume

- How exactly do we take contents of an arbitrary view volume and project them to a 2D surface?
- Arbitrary view volume is too complex…
- Reduce it to a simpler problem! The **canonical view volume!**
- Can also be called the *standard* or *unit* view volume
  - Specific orientation, position, height and width that make operations like projecting and clipping much easier, as we will see
  - Transform complex view volume and all objects in volume to the canonical volume **(normalizing transformation)** and then project contents onto normalized film plane
    - Not to be confused with animation where camera may move relative to objects! Normalization applies to an arbitrary camera view at a given instant
- Let's start with easiest case: parallel view volume

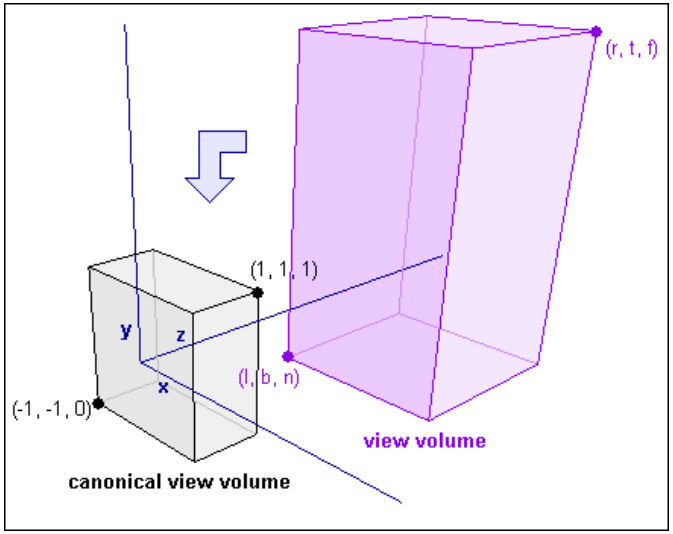
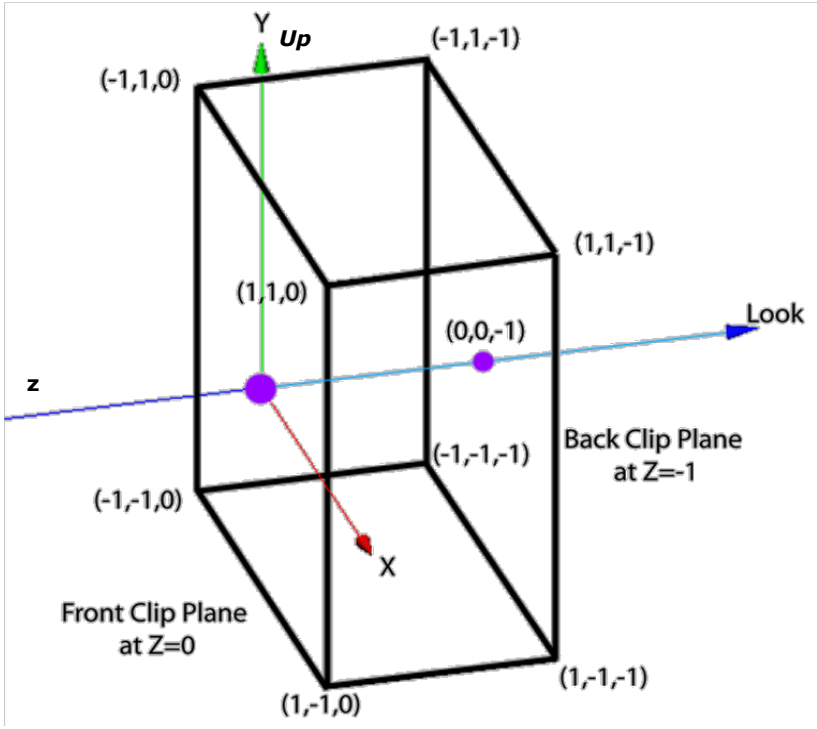


Image credit: http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__2/

The canonical parallel view volume

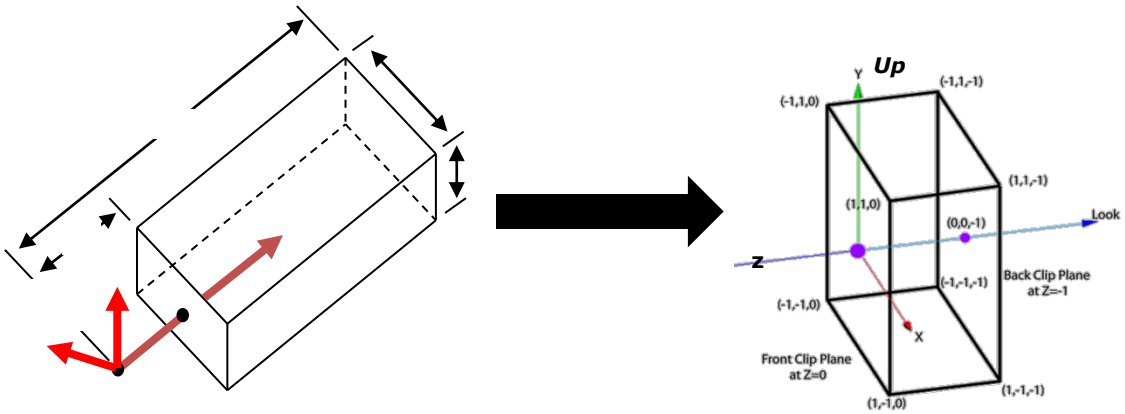
- Sits at origin:
  - Center of near clipping plane = (0,0,0)
- Looks along negative z-axis:
  - Look Vector = (0,0,-1)
- Oriented upright:
  - Up Vector = (0,1,0)
- Viewing window bounds normalized:
  - -1 to 1 in x and y directions
- Near and far clipping planes:
  - Near at z = 0 plane
  - Far at z= 1 plane



- Note: *Look vector* along negative z-axis seems like an odd choice, but it makes the math easier. Same with choosing -1 to 1 as our film plane bounds

The normalizing transformation

- Goal: transform arbitrary view and scene to canonical view volume, maintaining relationship between view volume and scene, then render
- For parallel view volumes need only rotations, scales, and translations
- The composite transformation composed of these scales, rotations and translations is a 4x4 homogenous matrix called the **normalizing transformation** (the inverse is called the **viewing transformation** and turns a canonical view volume into an arbitrary one)



Remember that our camera Is just a model, there is no actual camera in our scene. The normalizing matrix needs to be applied to every vertex in our scene to simulate this transformation

- Note: the scene resulting from normalization will not appear any different from the original - every vertex is transformed in the same way. The goal is to simplify our view volume, not change what we see.

View Volume Translation

- Our goal is to send the $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$ axes of camera's coordinate system to $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ axes of world coordinate system
- Start by moving camera from its position to origin
  - Given camera position P, $\boldsymbol{w}$ axis, and the distances to the $near$ and far clipping planes, the center of the near clipping plane is located at $P_n = P + near * w$
  - The following matrix will translate all world points and camera so that P_n is now at the origin

$$\begin{bmatrix} 1 & 0 & 0 & -P_{n_x} \\ 0 & 1 & 0 & -P_{n_y} \\ 0 & 0 & 1 & -P_{n_z} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

View Volume Rotation (1/3)

- Rotating the camera/scene can't be done by inspection
- Our Camera is now at the origin, we need to align the $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$ axes with the $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ axes
- Let's leave out the homogenous coordinate *for now*
- $\boldsymbol{e_1} = \begin{bmatrix}1\\0\\0\end{bmatrix}, \boldsymbol{e_2} = \begin{bmatrix}0\\1\\0\end{bmatrix}, \boldsymbol{e_3} = \begin{bmatrix}0\\0\\1\end{bmatrix}$
- Need to rotate $\boldsymbol{u}$ into $\boldsymbol{e_1}$, $\boldsymbol{v}$ into $\boldsymbol{e_2}$, and $\boldsymbol{w}$ into $\boldsymbol{e_3}$
- Need to find some matrix $\boldsymbol{R_{rot}}$, such that:
  - $\boldsymbol{R_{rot}u = e_1}$
  - $\boldsymbol{R_{rot}v = e_2}$
  - $\boldsymbol{R_{rot}w = e_3}$

View Volume Rotation (2/3)

- So how do we find $\boldsymbol{R_{rot}}$?
- The brute force way is to find angles between pairs of vectors $(u, e_1), (v, e_2), (w, e_3)$ and compose all 3 of the rotation matrices together to form a single rotation
- Too much math, not efficient -- there's a better way, using the linear algebra concept or orthogonal vectors (Transformations Lecture)

View Volume Rotation (2/3)

- Consider this:
  - $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$ are all orthogonal unit vectors
  - Want a matrix that converts each of these vectors to standard basis vectors
- For vector $\boldsymbol{u}$, $\boldsymbol{R_{rot}u}$ must equal $\boldsymbol{e_1} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$
- Think about each entry of this vector and compose a matrix that, when applied to $\boldsymbol{u}$, can obtain it

View Volume Rotation (3/3)

- Recall from linear algebra
  - A unit vector dotted with itself equals 1
  - A unit vector dotted with a vector orthogonal (perpendicular) to it equals $\mathbf{0}$
- To obtain the 1 we need the first row of our matrix to be $\boldsymbol{u}$ itself
- To get the other two to be $\mathbf{0}$ we need vectors perpendicular to u
- Why not use $\boldsymbol{v}$ and $\boldsymbol{w}$?
- Our matrix $\mathbf{R}_{\boldsymbol{rot}}$ now looks like this, $\begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \\ \boldsymbol{w} \end{bmatrix}$, where $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$ are row vectors
- Using the same reasoning as we did for u for the first row, we need v as the second row to get $(\mathbf{R}_{\boldsymbol{rot}})\boldsymbol{v}$ to equal $\boldsymbol{e_2}$
- We also need $\boldsymbol{w}$ as the third row to get $(\mathbf{R}_{\boldsymbol{rot}})\boldsymbol{w}$ to equal $\boldsymbol{e_3}$
- Feel free to confirm this by doing these matrix-vector multiplications

Final Rotation Matrix

- Our Rotation Matrix with homogenous coordinates:

$$\begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling the view volume

- So now we have a view volume sitting at the origin, oriented upright with the look vector pointing down the $-z$ axis
- But the size of our volume has not met our specifications yet
- We want the (x, y) bounds to be -1 and 1 and we want the far clipping plane to be at z = -1
- Given $width$, $height$, and far clipping plane distance, far, of a parallel view volume our scaling matrix S_{xyz} is as follows:
- $$\begin{bmatrix} \frac{2}{width} & 0 & 0 & 0 \\ 0 & \frac{2}{height} & 0 & 0 \\ 0 & 0 & \frac{1}{far} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ (note that $\frac{1}{\frac{width}{2}} = \frac{2}{width}$, etc.)
- Now all vertices are bounded in between planes x = (-1, 1), y = (-1, 1), z = (0, -1)

The normalizing transformation (parallel) and re-homogenization

- Now have a complete transformation from an arbitrary parallel view volume to canonical parallel view volume
- First translate to origin using translation matrix, $\boldsymbol{T_{trans}}$
- Then align $\boldsymbol{u, v, w}$ axes with $\boldsymbol{x, y, z}$ axes using rotation matrix $\boldsymbol{R_{rot}}$
- Finally scale view volume using scaling matrix $\boldsymbol{S_{xyz}}$
- Composite normalizing transformation is simply, $\boldsymbol{S_{xyz}\, R_{rot}\, T_{trans}}$

$$\begin{bmatrix} \frac{2}{width} & 0 & 0 & 0 \\ 0 & \frac{2}{height} & 0 & 0 \\ 0 & 0 & \frac{1}{far} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -P_{n_x} \\ 0 & 1 & 0 & -P_{n_y} \\ 0 & 0 & 1 & -P_{n_z} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Since each individual transformation results in $w = 1$, no division by w is necessary at this stage

Notation

- The book groups all of these three transformations together into one transformation matrix
- For the parallel case we will call it $\boldsymbol{M_{orthogonal}}$
- For the perspective case, which we will get to, it is called, $\boldsymbol{M_{perspective}}$
- For ease of understanding we split all three up, but they can be represented more compactly by the following:

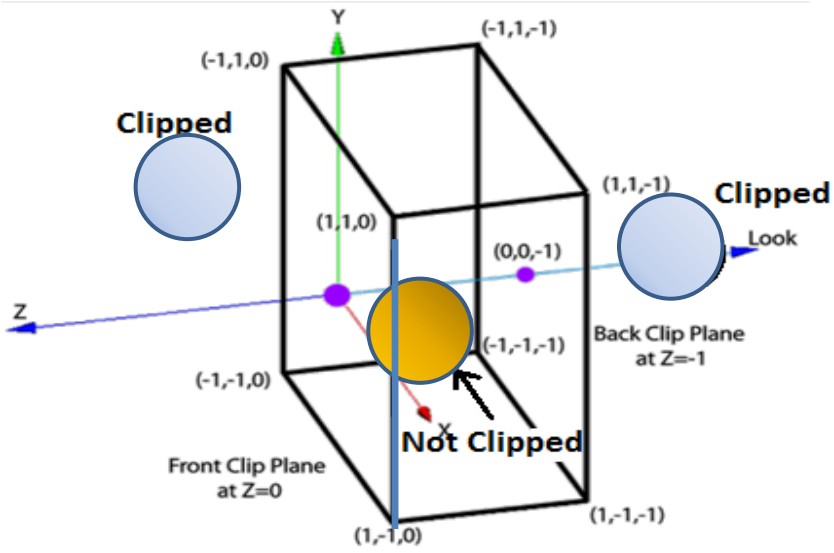
N is the 3x3 matrix representing rotations and scaling

$$\boldsymbol{N} = \begin{bmatrix} \frac{2}{width} & 0 & 0 \\ 0 & \frac{2}{height} & 0 \\ 0 & 0 & \frac{1}{far} \end{bmatrix} \begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{bmatrix}$$

$$\boldsymbol{M_{orthogonal}} = \begin{bmatrix} & N & & -P_{n_x} \\ & & & -P_{n_y} \\ & & & -P_{n_z} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Clipping against the parallel view volume

- Before returning to original goal of projecting scene onto film plane, how to clip?
- With arbitrary view volume, testing needed to decide whether a vertex is in or out and clipping is done by solving simultaneous equations
- With canonical view volume, clipping is much easier
- After we have applied the normalizing transformation to all vertices in the scene, anything that falls outside the bounds of the planes x = (-1,1), y = (-1, 1) and z = (0, -1), is clipped. Primitives that intersect the view volume must be partially clipped
- Most graphics packages such as OpenGL will do this step for you



Note: Clipping edges that intersect the boundaries of view volume is another step explored in next lecture

Projecting in the normalized view volume

- So how do we project the scene in this normalized view volume onto a the (x, y) plane, where the film plane is now located?
- If there is a point (x, y, z) that we want to project to the (x, y) plane, just get rid of the z coordinate!

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- We could take this a step farther and use the following matrix to convert 3D homogenous vectors into 2D homogenous

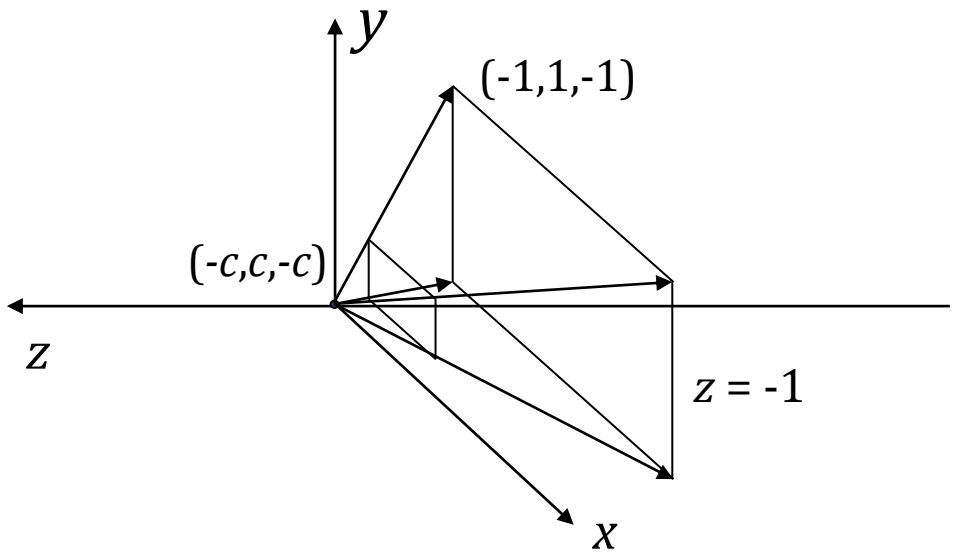
$$\mathbf{M}_{proj} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Like clipping, in most graphics packages, this step is also handled for you

The Perspective View Volume

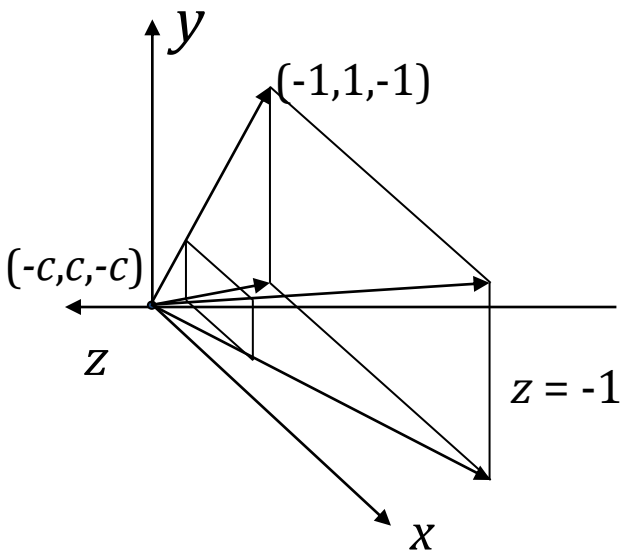
- Need to find a transformation to turn an arbitrary view volume into a canonical (unit) view volume

Canonical view volume:



Properties of the canonical view volume – as shown next

- Sits at origin:
  - Position = (0,0,0)
  - This time it's the actual given camera position that is going to move to the origin
- Looks along negative *z*-axis:
  - Look Vector = (0,0,-1)
- Oriented upright:
  - Up Vector = (0,1,0)
- Near and far clipping planes:
  - Near at $z = -\frac{near}{far}$ plane
  - Far at *z* = -1 plane
- Far clipping plane bounds:
  - (*x*, *y*) from -1 to 1
- Note: *The perspective canonical view volume is just like the parallel one except that the "film"/viewing window is more ambiguous here, so we bound just the far clipping plane for now*



Translation and Rotation

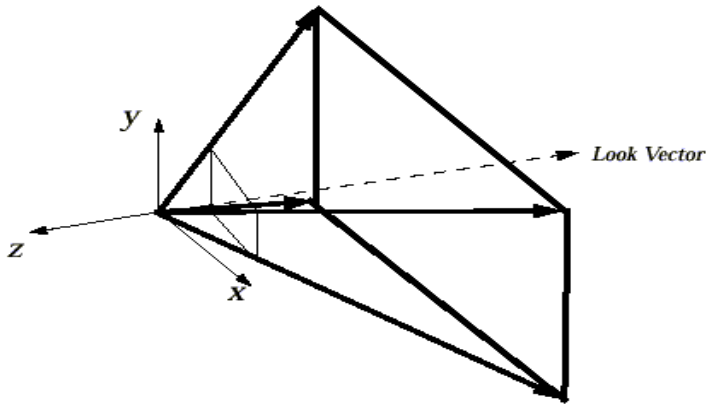
- For our normalizing transformation, the first two steps are the same
  - The translation matrix $\mathbf{T}_{trans}$ is even easier to calculate this time, since we are given the point to translate to the origin:

$$\begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

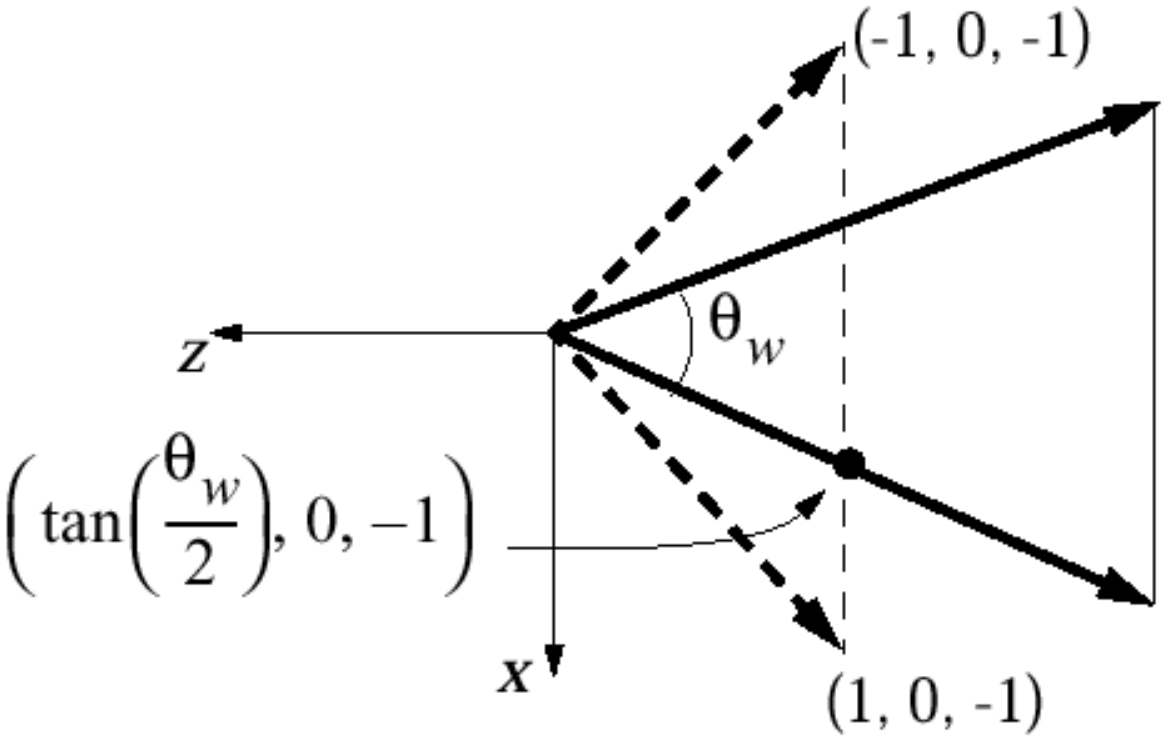
  - And we use the same matrix $\mathbf{R}_{rot}$ to align the camera axes:

$$\begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Our current situation:



Scaling the perspective view volume



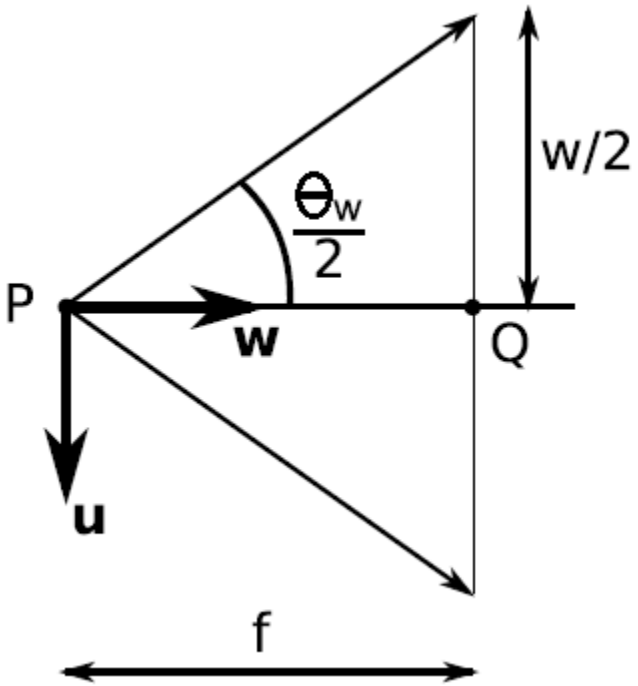
Problems with Scaling

- For perspective volumes, scaling is more complicated and requires some trigonometry
- It was easy to scale the parallel view volume if we know the width and height of our view volume
- Based on our definition of our perspective view volume however, we are not given these two values
- We need a scaling transformation $\boldsymbol{S_{xyz}}$, that:
  - Finds the width and height of the far clipping plane based on width angle θ_w and height angle θ_h and the distance to the clipping plane, far
  - Scales our view volume based on these dimensions

- Scaling the position of the far clipping plane to z = -1 remains the same as the parallel case since we are still given far, however, unlike the parallel case, the *near* plane is not immediately mapped to z = 0.

Scaling the perspective view volume (1/3)

- It's too hard to think about scaling in both the x and y directions at the same time, so consider the two separately
- Start with just the X
- We want to scale by $\frac{width}{2}$ of the far clipping plane to bring it to z = -1
  - Divide viewing frustum down the middle with line PQ
  - The length of PQ is just far and we know $\frac{\theta_w}{2}$



Note: θ_w refers to the width angle of the viewing frustum here. *w* shouldn't be confused with the *w-axis* of the camera coordinate system!

Scaling the perspective view volume (2/3)

- To find $\frac{w}{2}$ we need to use a little bit of trig

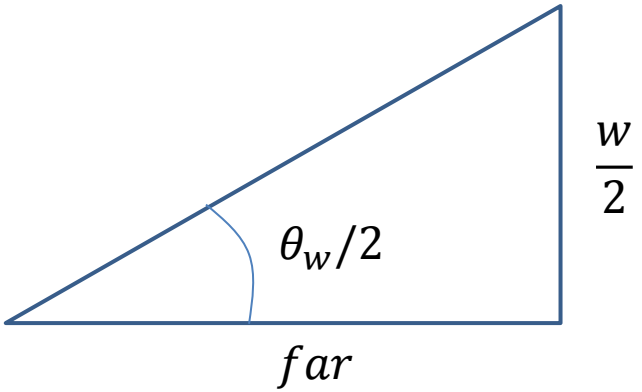
$$\frac{w}{2} = \tan\left(\frac{\theta_w}{2}\right) * far$$

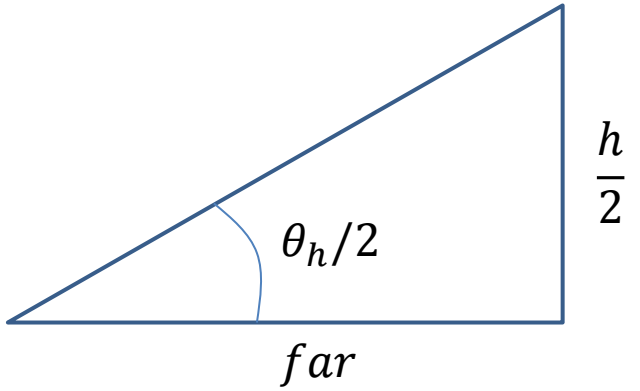
- Now, scale to send the X-coordinates of the far plane to [-1,1]

$$\frac{1}{\tan\left(\frac{\theta_w}{2}\right) * far}$$

- Repeat for the Y-coordinates,
  - just replace θ_w with θ_h
  - scale in the Y by

$$\frac{1}{\tan\left(\frac{\theta_h}{2}\right) * far}$$



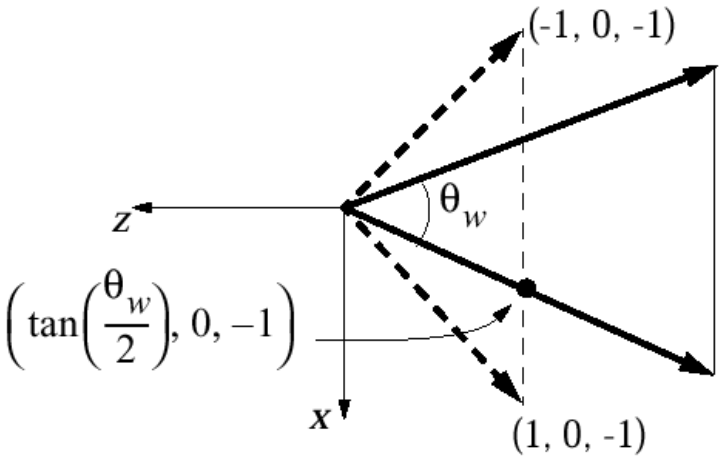


Scaling the perspective view volume (3/3)

- We know what to scale by to bound X and Y between -1 and 1
- To bound the far clipping plane to be between 0 and -1, we scale by $\frac{1}{far}$ as in the parallel case

$$\begin{bmatrix} \frac{1}{\tan\left(\frac{\theta_w}{2}\right) far} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\theta_h}{2}\right) far} & 0 & 0 \\ 0 & 0 & 1/far & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Here's what our transformation looks like in the xz plane:



The normalizing transformation (perspective)

- Our current perspective transformation takes on the same form as the parallel case: $\boldsymbol{S_{xyz}\, R_{rot}\, T_{trans}}$
  - $\boldsymbol{T_{trans}}$ takes the camera's *Position* and moves the camera to the world origin
  - $\boldsymbol{R_{rot}}$ takes the *Look* and *Up* vectors and orients the camera to look down the *–z* axis
  - $\boldsymbol{S_{xyz}}$ takes and scales the view volume so that the corners are at (±1, ±1) and takes the far clipping plane and scales it to lie on the *z=-1* plane

$$\begin{bmatrix} \frac{1}{\tan\left(\frac{\theta_w}{2}\right) far} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\theta_h}{2}\right) far} & 0 & 0 \\ 0 & 0 & 1/far & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- So given point P, if we multiply $\boldsymbol{S_{xyz}\, R_{rot}\, T_{trans}}$ * P = P′, the position of resulting point P′ will be translated, rotated and scaled to match our normalization but the projected scene will still look the same as if we had projected our scene using the arbitrary frustum

Notation

- We can represent this composite matrix as $\boldsymbol{M_{perspective}}$ by the following:

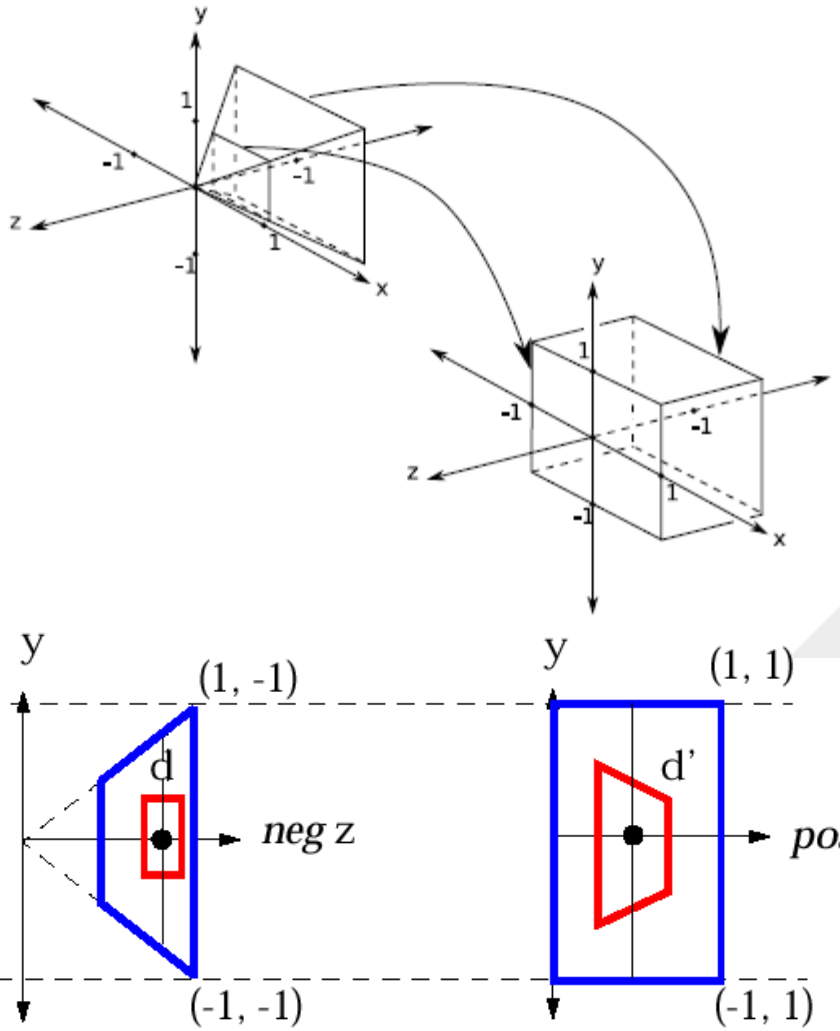
N is the 3x3 matrix representing rotations and scaling

$$\boldsymbol{N} = \begin{bmatrix} \frac{1}{\tan\left(\frac{\theta_w}{2}\right) far} & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\theta_h}{2}\right) far} & 0 \\ 0 & 0 & \frac{1}{far} \end{bmatrix} \begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{bmatrix}$$

$$M_{perspective} = \begin{bmatrix} & N & & \begin{matrix} -P_x \\ -P_y \\ -P_z \end{matrix} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective and Projection

- Now we have our canonical perspective view volume
- However, projecting a perspective view volume on to a 2D plane is much more difficult than it was in the parallel case
- The solution? Reduce it to a simpler problem!
- The final step of our normalizing transformation, transforming the perspective view volume into a parallel one
- Think of this transformation as the ***unhinging transformation***, represented by matrix $\boldsymbol{M_{pp}}$



The perspective transformation(1/2)

- We've put the perspective view volume into canonical position, orientation and size
- Let's look at a particular point on the original near clipping plane lying on the *Look vector*:

$$p = Position + near * Look$$

It gets moved to a new location

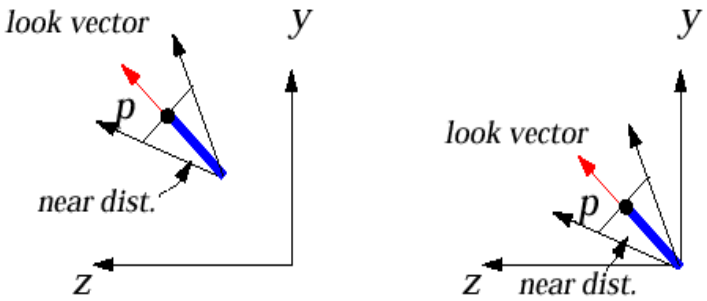
$$p' = S_{xyz} M_{rot} T_{trans} p$$

on the negative *z*-axis, say

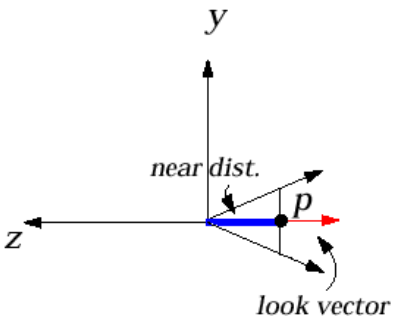
$$p' = \begin{pmatrix} 0 & 0 & c \end{pmatrix}$$

The perspective transformation(2/2)

- What is the value of *c*? Trace through the steps.
- *p* first gets moved to just $near * Look$



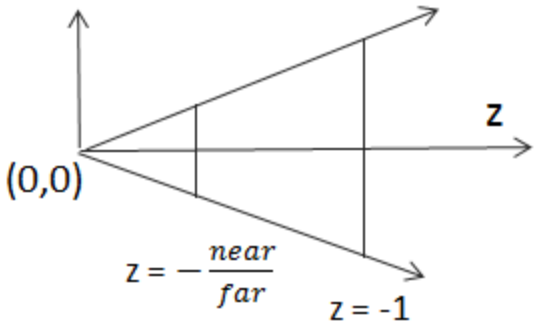
- This point is then rotated to $-near * \boldsymbol{e_3}$



- The xy scaling has no effect, and the *far* scaling changes this to $\left(-\frac{near}{far}\right)\boldsymbol{e_3}$, so it must be that $c = (-\frac{near}{far})$

Unhinging the View Volume (1/4)

- Note from figure that we don't have to do anything to far clipping plane, already in right position
- Near clipping plane needs to lie on Z=0 plane and should be bounded by -1 and 1 in X and Y
- Need to know where near clipping plane is in canonical frustum
- In arbitrary frustum distance to near clipping plane, $near$, can be represented as $\frac{near}{far} * far$
- $\frac{near}{far}$ remains constant before and after the normalizing transformation
- Our normalized far clipping plane is at z = -1
- So, the normalized near clipping plane is at $\boldsymbol{c = -\frac{near}{far}}$



Unhinging the View Volume (2/4)

- A restatement of our problem: We want to transform the portion standard frustum between c and -1 to a standard parallel view volume
- The derivation of this matrix is very complex, our approach will be instead to give you the matrix and show that it works by example
- Our unhinging transformation matrix, $\boldsymbol{M_{pp}}$

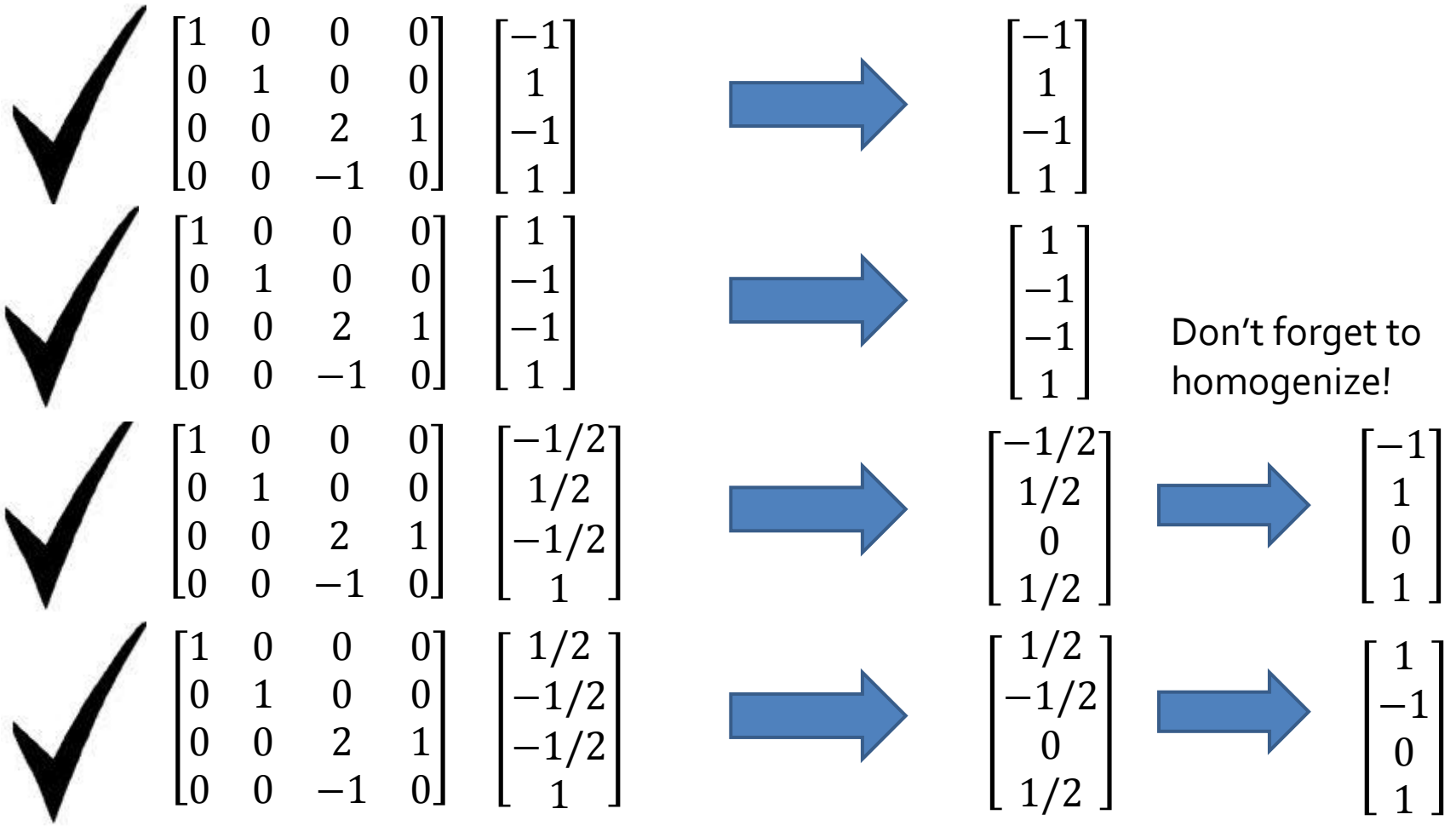
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{c+1} & \frac{-c}{c+1} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Unhinging the View Volume (3/4)

- Our perspective transformation do the following:
  - Send all points on the z = -1 far clipping plane to itself
    - We'll check (-1, 1, -1, 1) and (1, -1, -1, 1)
  - Sends all points on the z = c near clipping plane on to the z = 0 plane
    - Note that the corners of the canonical clipping plane are actually $(-c, c), (c, -c), (c, c)\ and\ (-c, -c)$ (Similar triangle geometry)
    - We'll check to see that $(-c, c, c, 1)$ gets sent to (-1, 1, 0, 1)
    - And that $(c, -c, c, 1)$ gets sent to (1, -1, 0, 1)
  - Let's say $c = -\frac{1}{2}$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{c+1} & \frac{-c}{c+1} \\ 0 & 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

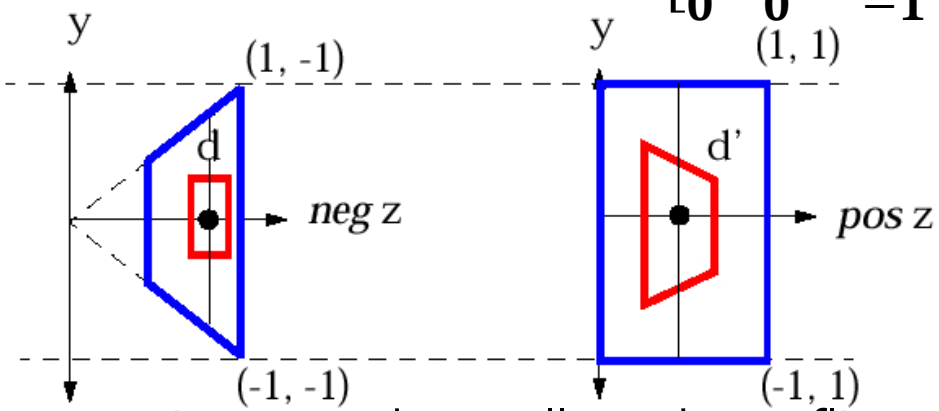
Unhinging the View Volume (4/4)



Practical Considerations: The z-buffer

- Typically depth testing is done using a z-buffer that determines the order in which objects are rendered based on the normalized z-values of the vertices
- The expected range for these values are from 0.0 to 1.0 where 0.0 is the closest an object can be before getting clipped away, and 1.0 is the farthest
- Thus we present an alternate form of M_{pp} that does the same thing as the original but negates the z-term:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{c+1} & \frac{-c}{c+1} \\ 0 & 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{\frac{-1}{c+1}} & \mathbf{\frac{c}{c+1}} \\ \mathbf{0} & \mathbf{0} & \mathbf{-1} & \mathbf{0} \end{bmatrix}$$



- Use this one in your assignments, but we'll use the un-flipped version for the remainder of the lecture

The normalizing transformation (perspective)

- We now have our final normalizing transformation, call it $\boldsymbol{N_{perspective}}$, to convert and arbitrary perspective view volume into a canonical parallel view volume
- $\boldsymbol{N_{perspective} = M_{pp} S_{xyz} R_{rot} T_{trans}}$ or $\boldsymbol{M_{pp} M_{perspective}}$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-1}{c+1} & \frac{c}{c+1} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\tan\left(\frac{\theta_w}{2}\right) far} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\theta_h}{2}\right) far} & 0 & 0 \\ 0 & 0 & 1/far & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Remember to homogenize your points after you apply this transformation
- We can now project our points to the viewing window easily since we're using a parallel view volume: Just get rid of the z-coordinate!
- After that we can map our viewing window to the viewport using the windowing transformation

The windowing transformation (1/2)

- The last step in our rendering process after projecting is to take our "film"/viewing window, and resize it to match the dimensions of the viewport so that we can easily map the contents of our film to our viewport
- To do this we want to have a film/viewing window with the lower left corner at (0,0) and the $width$ and $height$ of the viewport
- This can be done using the windowing transformation:

$$M_{wind} = \begin{bmatrix} width & 0 \\ 0 & height \end{bmatrix} \begin{bmatrix} 1/2 & 0 & 1/2 \\ 0 & 1/2 & 1/2 \end{bmatrix}$$

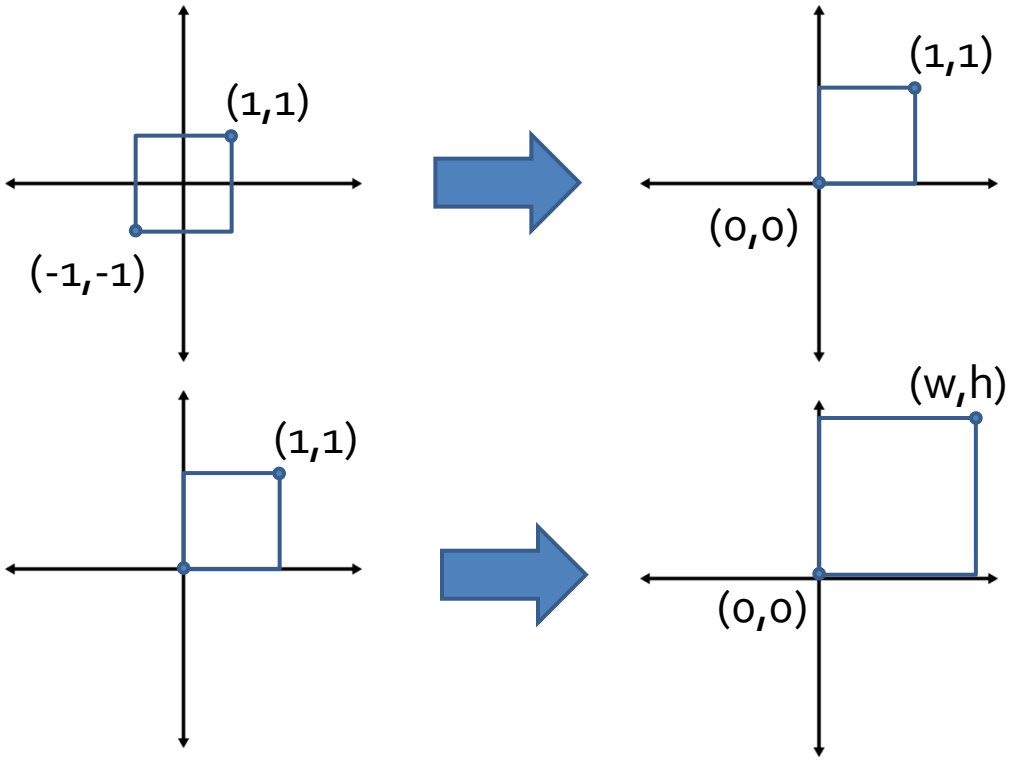
The windowing transformation (2/2)

- We first scale our viewing window to be between – ½ and ½ in the X and Y and then translate by the window by (½, ½) to put the lower left corner at the origin

$$\begin{bmatrix} 1/2 & 0 & 1/2 \\ 0 & 1/2 & 1/2 \end{bmatrix}$$

- Then we scale by the $width$ and $height$ of the viewing window to get our desired result

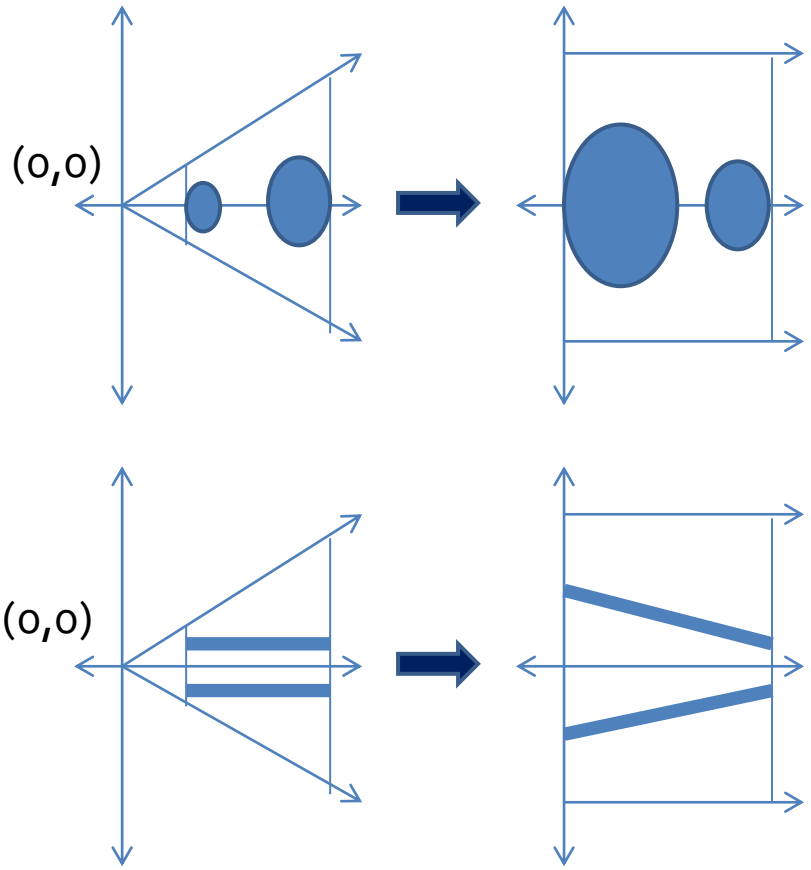
$$\begin{bmatrix} width & 0 \\ 0 & height \end{bmatrix}$$



- Note: You can confirm this matches the more general windowing transformation we presented to you in the transformations lecture, with the exception that this transformation gets rid of the homogenous coordinate
- This step is also usually handled by most graphics packages

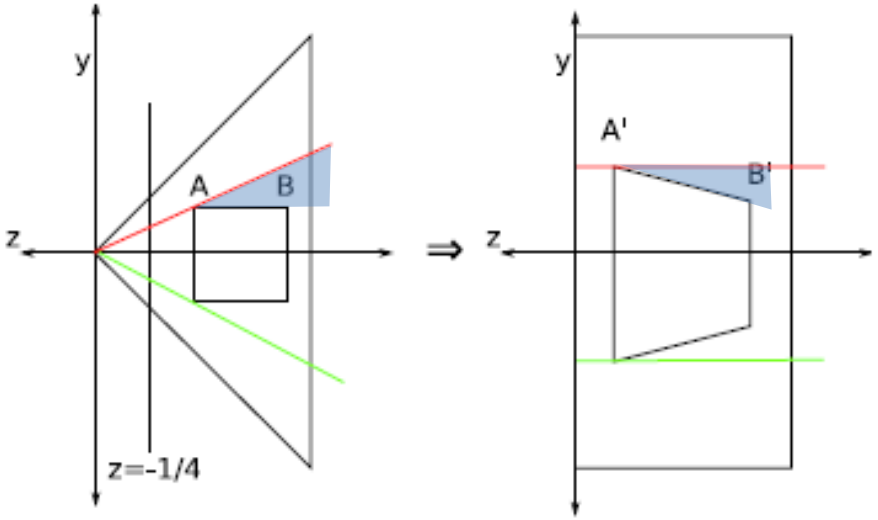
Why it works (1/2)

- You may question, how exactly does this transformation result in a perspective scene?
- The key is in the **unhinging step**
- We can take an intuitive approach to see this
  - The closer the object is to the near clipping plane, the more it is enlarged during the unhinging step
  - Thus, closer objects are larger and farther away objects are smaller as is to be expected
- Another way to see it is to use the parallel lines
  - Draw parallel lines in a perspective volume
  - When we unhinge the volume, the lines fan out at the near clipping
  - The result is converging lines
  - Think of a pair of railroad tracks that appear to converge in the distance



Why it works (2/2)

- Yet another way to demonstrate how this works is to use occlusion (when elements in the scene are blocked by other elements)
- Looking at the top view of the frustum, we see a square
- Draw a line from your eye point to the left corner of the square, we can see that points behind this corner are obscured
- Now unhinge the perspective and draw a line again to the left corner, we can see that all points obscured before are still obscured and all points that were visible before are still visible



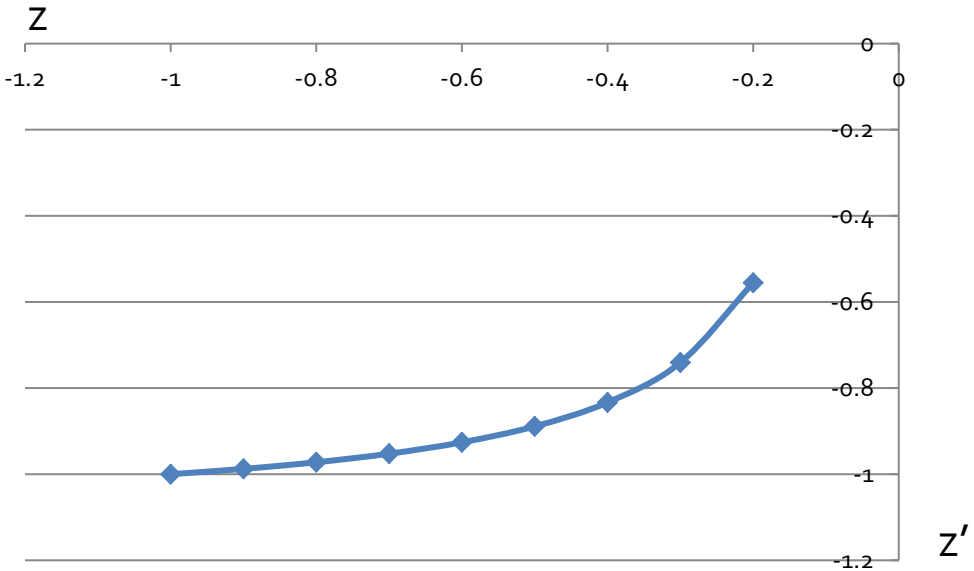
Unhinging considerations (1/3)

- One important effect of the unhinging transformation is that points are compressed towards the far clipping plane
- Let's look at the general case of multiplying $M_{pp} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{c+1} & \frac{-c}{c+1} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ \frac{z-c}{c+1} \\ -z \end{bmatrix} \Rightarrow \begin{bmatrix} -x/z \\ -y/z \\ \frac{\frac{c}{z}-1}{c+1} \\ 1 \end{bmatrix}$$

- Let's focus on the new z-term, call it z'. This represents the new depth of the point along the z-axis after normalization and homogenization
- $z' = \frac{\frac{c}{z}-1}{c+1}$, now let's hold c constant and plug in some values for z
- Let's have $near = -.1,\ far = -1,\ so\ c = .1$
- The following is a graph of z' dependent on z:

Unhinging considerations (2/3)



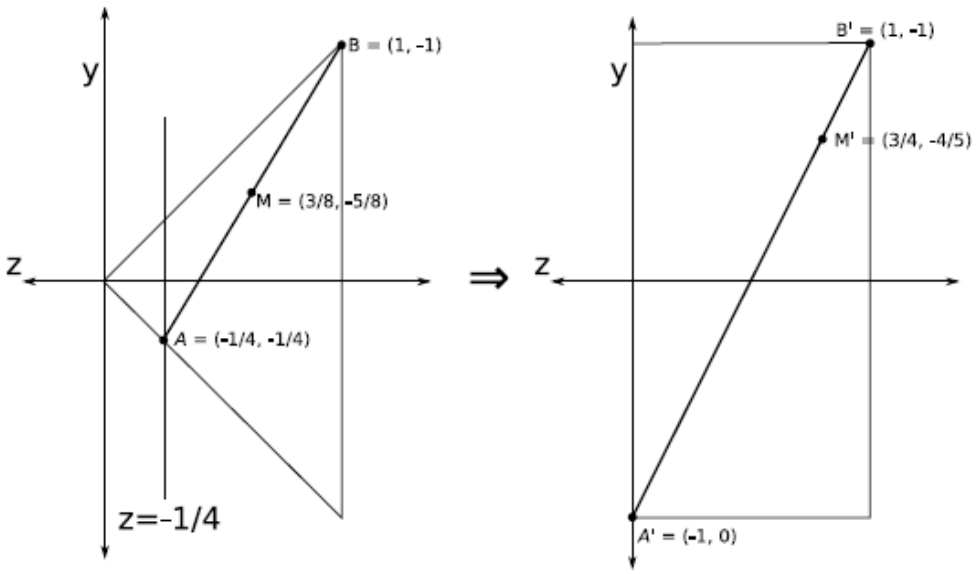
- We can see that if the z-values of points are being compressed towards z = -1 in our canonical view volume, the compression is more noticeable for points originally closer to the near clipping plane
- If you try playing around with the near and far clipping planes, another important observation is that as you bring the near clipping plane closer to z = 0, or extend the far clipping plane out more, the compression becomes more severe
- Caution when choosing near and far clipping planes, if compression is too severe, depth testing become more inaccurate near the back of the view volume and errors in rounding can cause objects to be rendered out of order

Unhinging considerations (3/3)

- One may be tempted to place the near clipping plane at z = 0, or the far clipping plane very far away (z = ∞)
- First note that the value of $c = \frac{near}{far}$ as either $near$ approaches 0 or far approaches ∞, approaches 0
- Applying this to our value for $z' = \frac{\frac{c}{z}-1}{c+1}$, we sub in 0 for c to get $z' = -\frac{1}{1} = -1$
- From this we can see that if our far clipping plane approaches infinity, or if our near clipping plane approaches 0, points will cluster at $z = -1$, the far clipping plane of our canonical view volume

Aside: Projection and Interpolation(1/3)

- This converging of points at the far clipping also poses problems when trying to interpolate values, such as color, between points
- Say for example we color the midpoint between two vertices, **call them A and B,** in a scene as the average of the two colors of A and B
- If we were just using a parallel view volume it would be safe to just set the midpoint to the average and be done
- We can't do that for perspective transformations since the point that was originally the midpoint gets compressed towards the far clipping plane and isn't the actual midpoint anymore
- Another way to say this is that the color, **call it G,** does not interpolate between points linearly anymore, so we can't just assign the new midpoint the average color



Aside: Projection and Interpolation(2/3)

- However, while G does not interpolate linearly, G/w does, where w is the homogenous coordinate after being multiplied by our normalizing transformation, but before being homogenized
  - In our case w will always be $-z$
- Knowing this, how can we find the color at this new midpoint?
- When we transform A and B, we get two w values, w_a and w_b
- We also know the values of G_a and G_b
- If we interpolate linearly between $\frac{G_a}{w_a}$ and $\frac{G_b}{w_b}$ (which in this case is just taking the average), we will know the $\frac{G}{w}$ value for the new midpoint $\frac{G_m}{w_m}$
- We can also find the average of $1/w_a$ and $1/w_b$ to get $1/w_m$ by itself
- Dividing $\frac{G_m}{w_m}$ by $\frac{1}{w_m}$, we can get our new value of $\boldsymbol{G_m}$

Aside: Projection and Interpolation(3/3)

- Let's make this slightly more general
- Say we have a function f that represents a property of a point (we used color in the last example)
- The point we want to apply the function to between points A and B is: $(\mathbf{1}-\boldsymbol{t})\boldsymbol{A}+\boldsymbol{tB}$, (let's call it $\boldsymbol{P}$)
- t ranges from 0 to 1, and represents the fraction of the way from point A' to point B' your point of interest is (in our last example, t = .5)
- Goal: Compute $\boldsymbol{f}(\boldsymbol{P})$

$$\frac{\mathbf{1}}{\boldsymbol{W_t}} = (\mathbf{1}-\boldsymbol{t})\left(\frac{\mathbf{1}}{\boldsymbol{w_a}}\right)+\boldsymbol{t}\left(\frac{\mathbf{1}}{\boldsymbol{w_b}}\right)$$

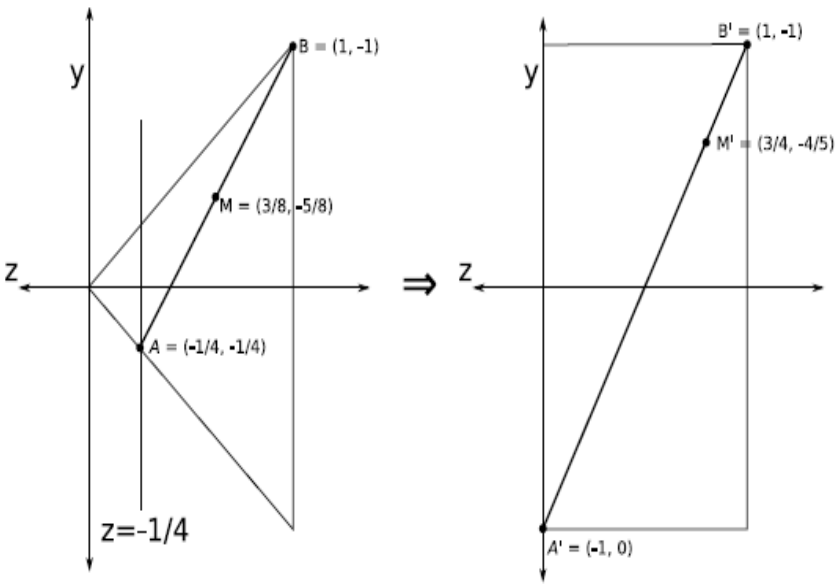
$$\frac{\boldsymbol{f}(\boldsymbol{P})}{\boldsymbol{w_t}} = (\mathbf{1}-\boldsymbol{t})*\frac{\boldsymbol{f}(\boldsymbol{A})}{\boldsymbol{w_a}}+\boldsymbol{t}*\frac{\boldsymbol{f}(\boldsymbol{B})}{\boldsymbol{w_b}}$$

- So to find the value of our function at the point specified by t we compute

$$\frac{\boldsymbol{f}(\boldsymbol{P})}{\boldsymbol{w_t}}\Big/\frac{\mathbf{1}}{\boldsymbol{W_t}} = \boldsymbol{f}(\boldsymbol{P})$$

Proof by example

- Let's revisit the setup from this image:
- Say we want the $f(A) = 0, f(B) = 1$, and thus $f(M) = .5$
- <u>After unhinging transformation:</u>
- The new midpoint, M', is 4/5 of the way from A' to B' , which can be found by dividing: $(A'M')/(A'B')$
- Like $f(M)$, $f(M')$ should be .5
- $w_a = ¼$ and $w_b = 1$
- $\frac{1}{W_t} = (1-.8)\left(\frac{1}{.25}\right) + .8\left(\frac{1}{1}\right) = 1.6$
- $\frac{f(P)}{w_t} = (1-.8) * \frac{0}{.25} + .8 * \frac{1}{1} = .8$
- $f(M') = \frac{f(P)}{w_t} / \frac{1}{W_t} = .5$



Final Words (1/2)

- We know about camera and object modeling transformations now, let's put them together:
- 1) $N_{perspective} = M_{pp} M_{perspective}$
- 2) $CMTM = SRT$
  - The CMTM (Composite Modeling Transformation Matrix) is a composite matrix of all of our object modeling transformations (Scaling, Rotating, Translations, etc)
- 3) $CTM = N_{perspective} * CMTM$
  - The CTM (Composite Transformation Matrix) is the combination of all our camera and modeling transformations
    - In OpenGL it is referred to as the ModelViewProjection Matrix
    - Model: Modeling Transformations
    - View: Camera translate/rotate
    - Projection: Frustum scaling/unhinging

Final Words (2/2)

- With our CTM we now have a start to finish view of our rendering process:
  - **1) Apply the CTM to all points in the scene**
  - **2) Project normalized scene on to film plane (into viewing window)**
  - **3) Resize viewing window to match viewport size**
  - **4) Map colors at (x,y) coordinates of viewing window to (u,v) pixels of viewport and our rendering is complete!**

- **Applications in CS123:**
- *Camtrans:*
  - You will be computing the normalizing transformation for adjustable camera settings
- *Sceneview:*
  - You will extend your camtrans code and object transformations to build the CMTM from a scene-graph and then the CTM for each primitive in the scene