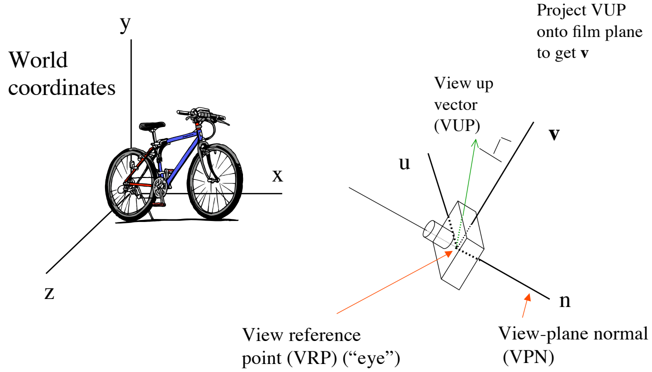
Specifying a camera

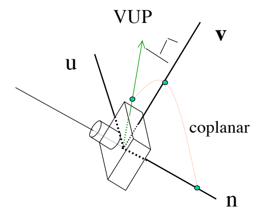


Computing (u,v,n) in world coordinates

v is the projection of VUP into the view plane which is perpendicular to **n**

u is perpendicular to the plane formed by **n** and VUP

So, we can easily compute a vector parallel to **u**.

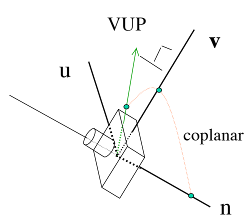


Computing (u,v,n) in world coordinates

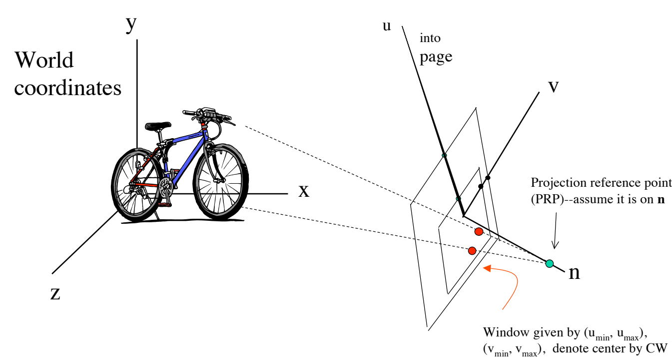
$$\mathbf{u} \parallel \text{VUP} \times \mathbf{n}$$

$$\mathbf{u} = \frac{\text{VUP} \times \mathbf{n}}{\left|\text{VUP} \times \mathbf{n}\right|} = \frac{\text{VUP} \times \mathbf{n}}{\left|\text{VUP}\right|}$$

What about **v**?



Specifying a camera



- VRP, VPN, VUP must be in world coords;
- PRP (focal point) could be in world coords, but more commonly, camera coords (which are the same scale as world coords)
- We will use camera coords, and further assume that PRP = (0,0,f).

Object in world coordinates
(after modeling transforms)

↓

Transform object from world coordinates to standard camera coordinates

↓

Clip against canonical view frustum

↓

Project using standard camera model ✓

➡

Transform object from world coords to camera coords

↓

Further transform so that frustum is canonical frustum.

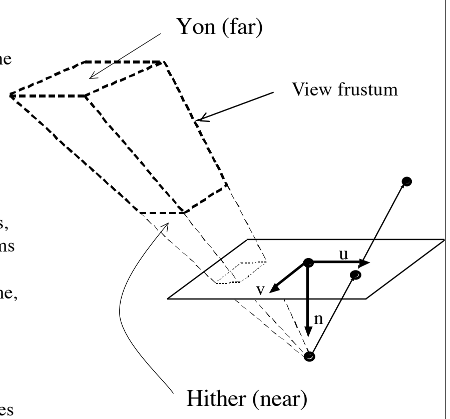
All in H&B chapter 7 “3D viewing”

u and **v** can be used to specify a window in the image plane; only this section of image plane ends up on the screen.

This window defines four planes; points outside these planes are not rendered.

Hither and yon clipping planes, which are always given in terms of camera coordinates, and always parallel to the film plane, give a volume - known as the view frustum.

Orthographic case: - view frustum is cuboid (i.e. all angles right angles, but edges not necessarily of equal length).



We will first map world coordinates to the camera coordinates (top figure)

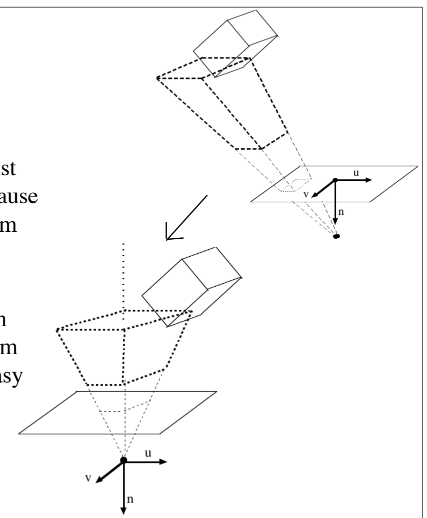
However, if clipping against the frustum is difficult because planes bounding the frustum have a complex form

Solution: further transform frustum to a canonical form where clip planes are an easy form to deal with:

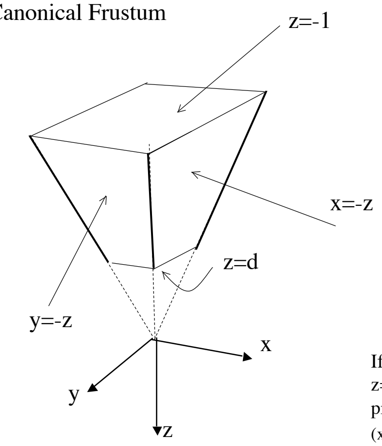
$z=x,\ z=-x$

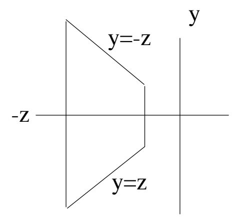
$z=y,\ z=-y$

$z=-1,\ z=d$



Canonical Frustum





If image plane transforms to $z=-f'$ then in new frame, projection is easy:
$(x, y, z) \rightarrow (-f'x/z, -f'y/z)$

Transform object from world coords to camera coords

Step 1. Translate the camera at VRP to the world origin. Call this T_1.

Translation vector is simply negative VRP.

(We are changing the coordinate system of the world, which is the same thing mathematically as moving the camera. We want object world coordinates to **change** so that the camera location (VRP) **becomes** the origin).

Transform object from world coords to camera coords

Step 2. Rotate camera coordinate frame (in w.c.) so that so that **u** is **x**, **v** is **y**, and **n** is **z**. The matrix is ?

(We are changing the coordinate system of the world, which is the same thing mathematically as moving the camera. We want object world coordinates to **change** so that the camera axis **becomes** the standard axis—e.g, **u** becomes (1,0,0), **v** becomes (0,1,0) and **n** becomes (0,0,1)).

Transform object from world coords to camera coords

Step 2. Rotate camera coordinate frame (in w.c.) so that so that **u** is **x**, **v** is **y**, and **n** is **z**. The matrix is:

$$\begin{vmatrix} \mathbf{u}^T & 0 \\ \mathbf{v}^T & 0 \\ \mathbf{n}^T & 0 \\ 0\ 0\ 0 & 1 \end{vmatrix}$$

(why?)

Object in world coordinates (after modeling transforms)

↓

Transform object from world coordinates to standard camera coordinates

↓

Clip against canonical view frustum

↓

Project using standard camera model ✓

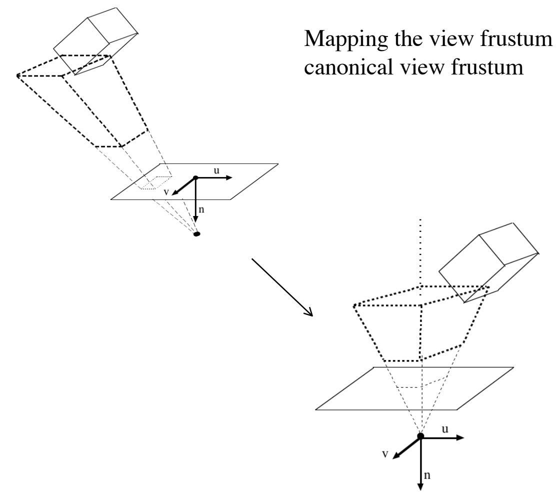
→

Transform object from world coords to camera coords ✓

↓

Further transform so that frustum is canonical frustum.

Mapping the view frustum to the canonical view frustum

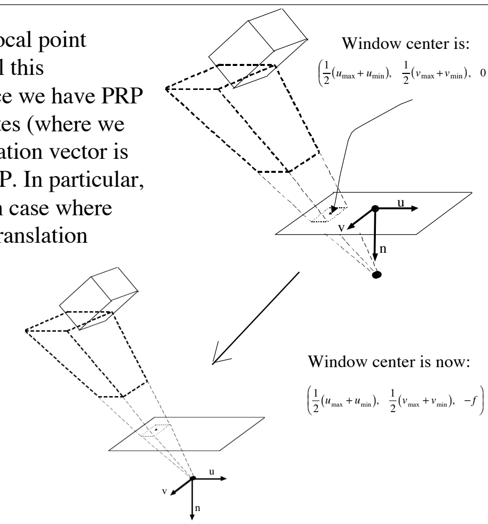


Further transform so that frustum is canonical frustum.

Since we are now in camera coordinates, we will often refer to them as (x,y,z) not (u,v,n).

1. Translate focal point to origin
2. Shear so that central axis of frustum lies along the z axis
3. Scale x, y so that faces of frustum lie on conical planes
4. Isotropic scale so that back clipping plane lies at z=-1

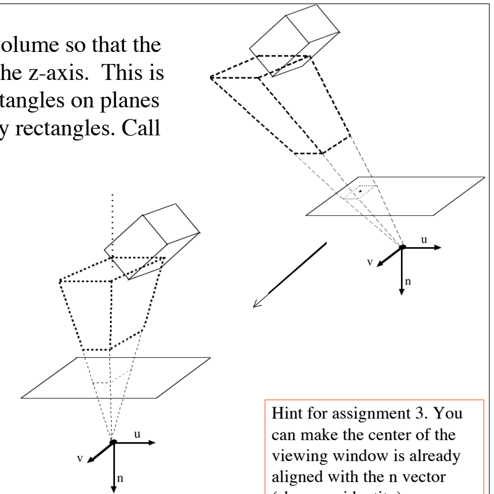
Step 1: Translate focal point (PRP) to origin; call this translation T_2. Since we have PRP in camera coordinates (where we now are), the translation vector is simply negative PRP. In particular, in the very common case where PRP is (0,0,f), the translation vector is (0,0,-f)



Step 1 is relatively straightforward, but notice that the location of the clipping planes also gets shifted.

So, before we had the back clipping plane at B (which is negative). Now it is at: B-f.

Step 2: Shear this volume so that the central axis lies on the z-axis. This is a shear, because rectangles on planes z=constant must stay rectangles. Call this shear S_1

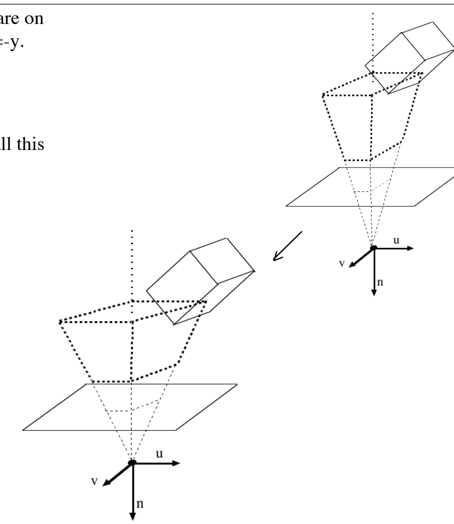


Hint for assignment 3. You can make the center of the viewing window is already aligned with the n vector (shear == identity).

Shear S_1 takes previous window midpoint $\left(\frac{1}{2}(u_{max}+u_{min}),\ \frac{1}{2}(v_{max}+v_{min}),\ -f\right)$ to (0, 0, -f) - this means that matrix is

?

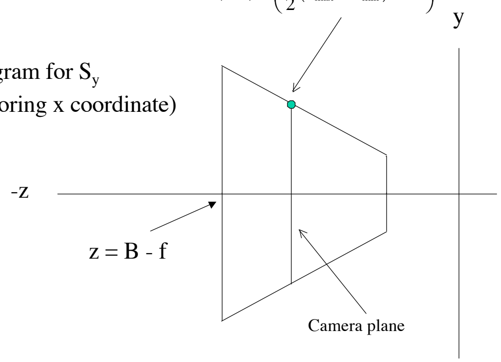
3. Scale x, y so that planes are on z=x, z=-x and z=y and z=-y. Call this scale Sc_1
4. Isotropic scale so that far clipping plane is z=-1; call this scale Sc_2



3. Scale x, y so that planes are on z=x, z=-x and z=y and z=-y. Call this scale Sc_1

$$(y,z)=\left(\frac{1}{2}(v_{max}-v_{min}),-f\right)$$

Diagram for S_y
(ignoring x coordinate)



4. Scale x, y so that planes are on z=x, z=-x and z=y and z=-y. Call this scale Sc_1

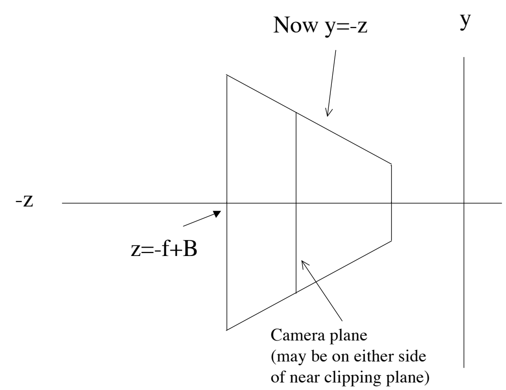
$$\left(\frac{1}{2}(v_{max}-v_{min}),\ -f\right) \Rightarrow (f, -f) \quad \text{(because } y=-z\text{)}$$

$$k_y\frac{1}{2}(v_{max}-v_{min})=f$$

$$k_y=\frac{2f}{(v_{max}-v_{min})} \qquad (k_y \text{ is y scale factor})$$

$$\mathbf{Sc}_1=\begin{vmatrix} \frac{2f}{(u_{max}-u_{min})} & 0 & 0 & 0 \\ 0 & \frac{2f}{(v_{max}-v_{min})} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

5. Now isotropic scale so that far clipping plane is z=-1; call this scale Sc_2



5. Now isotropic scale so that far clipping plane is z=-1; call this scale Sc_2

Currently, at far clipping plane, z=-f+B

Want a factor k so that k(-f+B)=-1

So, k = -1 / (-f + B) = 1 / (f - B)

(Note that B is negative, and k is positive)

$$\mathbf{Sc}_2 = \begin{vmatrix} \frac{1}{f-B} & 0 & 0 & 0 \\ 0 & \frac{1}{f-B} & 0 & 0 \\ 0 & 0 & \frac{1}{f-B} & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Note that the focal length, f, also gets transformed (needed for the perspective transformation coming up).

It is: $$f' = \frac{f}{f-B}$$

3D Viewing Pipeline

$$\left(\text{Point in canonical camera coordinates} \right) \quad Sc_2 Sc_1 S_1 T_2 R_1 T_1 \quad \left(\text{Point in world coordinates} \right)$$

Then clip (more later), and then project

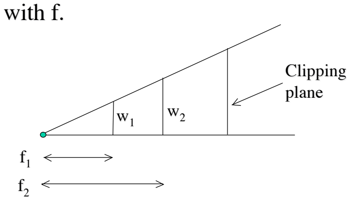
Further comments on the canonical frustum

u_{min}, u_{max}, v_{min}, v_{max}, are thought of as being in the camera coordinate system ==> units are that of world coordinate system

For assignment three, you need to choose u_{min}, u_{max}, v_{min}, v_{max}, and f.

I suggest simply setting u_{min}, u_{max}, v_{min}, v_{max}, to reflect your understanding of your screen window in world coordinates, and set f accordingly. (Best to keep the aspect ratio the same).

Note the approximate reciprocal relation of u_{min}, u_{max} and v_{min}, v_{max}, with f.



f_1 and w_1 give the same image as f_2 and w_2, but to see this in the math note that the camera center has shifted.

Because of this shift, the clipping plane values change, and $B_1 - f_1 == B_2 - f_2$

Determining the screen coordinates

Once you have (x,y) you need to map them back to the screen coordinates.

Use primes (‘) for mapped quantities (including projection) and carets (^) for screen quantities.

The canonical frustum gives the screen as a square that is 2f’ by 2f’. Note that f’ is between 0 and 1 (why?)

Our window on the screen has corners: $(\hat{u}_{min}, \hat{u}_{max})$ and $(\hat{v}_{min}, \hat{v}_{max})$
(Unless we want to distort things we assume the same aspect ratio as the camera window.)

Determining the screen coordinates

Our screen coordinates are then:

$$\hat{x} = \left(\frac{\hat{u}_{max} + \hat{u}_{min}}{2}\right) + \left(\frac{x'}{2f'}\right) \bullet \left(\hat{u}_{max} - \hat{u}_{min}\right)$$
$$\hat{y} = \left(\frac{\hat{v}_{max} + \hat{v}_{min}}{2}\right) + \left(\frac{y'}{2f'}\right) \bullet \left(\hat{v}_{max} - \hat{v}_{min}\right)$$

Or, equivalently:

$$\hat{x} = \hat{u}_{min} + \left(\frac{x' + f'}{2f'}\right) \bullet \left(\hat{u}_{max} - \hat{u}_{min}\right)$$
$$\hat{y} = \hat{v}_{min} + \left(\frac{y' + f'}{2f'}\right) \bullet \left(\hat{v}_{max} - \hat{v}_{min}\right)$$

Plan A: Clipping against the canonical frustum

2D algorithms are easily extended. For line clipping with Cohen Sutherland we use the following 6 out codes:

$y>-z \quad y<z \quad x>-z \quad x<z \quad z<-1 \quad z>z_{min}$

($z_{min} = (f-F)/(B-f)$)

Recall C.S for segments

```
Compute out codes for endpoints
While not trivial accept and not trivial reject:
        Clip against a problem edge (one point in, one out)
        Compute out codes again
Return appropriate data structure
```

Clipping against the canonical frustum

Clipping polygons in 3D against canonical frustum planes is simpler and more efficient than the general case.

Recall the S.H. gives four cases:

- Polygon edge crosses clip **plane** going from out to in
  - emit crossing, next vertex
- Polygon edge crosses clip **plane** going from in to out
  - emit crossing
- Polygon edge goes from out to out
  - emit nothing
- Polygon edge goes from in to in
  - emit next vertex

(The above is from before, just change "edge" to "plane")

Object in world coordinates (after modeling transforms)

↓

Transform object from world coordinates to standard camera coordinates ✓

↓

Clip against canonical view frustum

↓

Project using standard camera model ✓

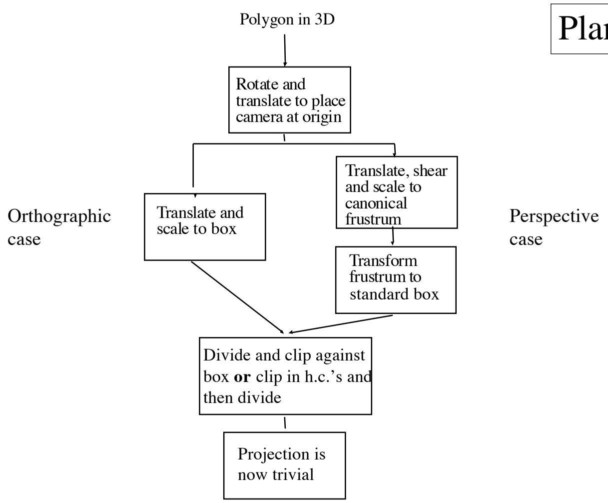
Plan A: Clip against canonical frustum ✓ (relatively easy—we chose the canonical frustum so that it would be easy!)

Plan B: Be even more clever. Further transform to cube and clip in homogenous coordinates.

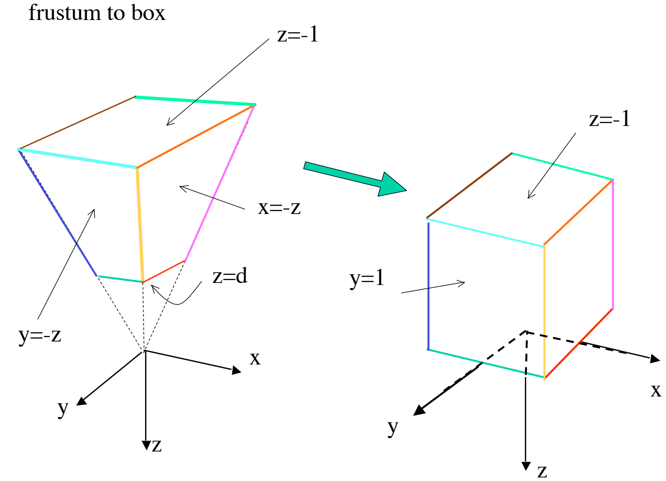
Plan B: Clipping in homogenous coords

- For any camera, can turn the view frustrum into a regular parallelepiped (box). We will use the box bounded by $x = \pm 1$, $y = \pm 1$, $z = -1$, and $z = 0$.
- Advantages
  - Simplified clipping in homogenous coordinates
  - Extends to cases where we use homogenous coordinates to represent additional information (and w could be negative).
  - Can simplify visibility algorithms.
- Approach: clever use of homogenous coordinates

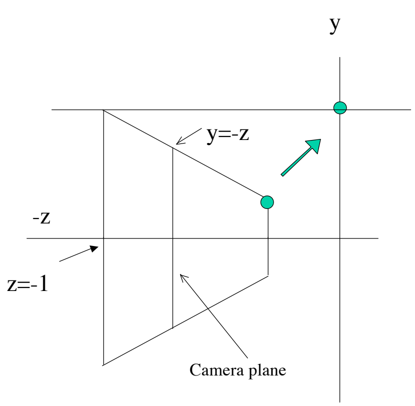
Plan B



Transforming canonical frustum to box

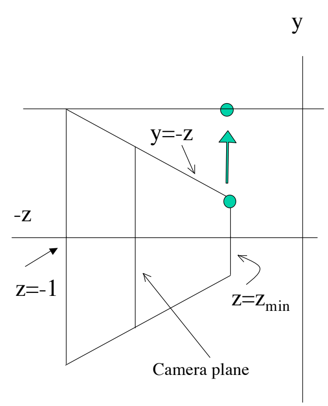


Transforming canonical frustum to box



Do this in two steps. One stretch in y (and x), and on stretch in z.

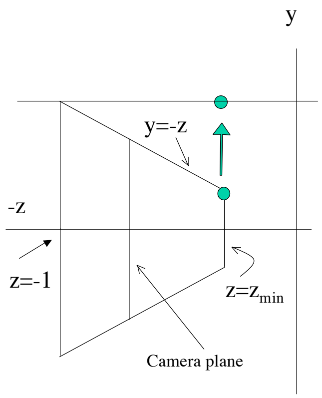
Transforming canonical frustum to box



The picture should suggest an appropriate scaling for y.

It is ?

Transforming canonical frustum to box



On top, y—> 1, so scaling is (1/y) Recall that y=-z there.

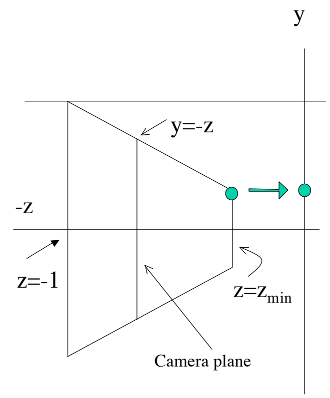
On bottom, y—> -1 so scaling is (–1/y). Recall that y=z there.

So scaling is y'= y/(-z)

Similarly, x' = x/(-z)

Transformation is **non-linear**, but in h.c., we can make w = (-z).

Transforming canonical frustum to box



For z, we translate near plane to origin. But now box is too small. Specifically it has z dimension $(1 + z_{min})$ (recall z_{min} is negative)

So we have an extra scale factor $1 / (1 + z_{min})$ and thus $z' = (z - z_{min}) / (1 + z_{min})$

But we want x and y to work nicely in h.c., with w=-z, so we use

$z' = ((z - z_{min}) / (1 + z_{min})) / (-z)$

(Thus in our box, depth transforms **non-linearly**)

In h.c.,

x=>x

y=>y

$z \Rightarrow (z - z_{min}) / (1 + z_{min})$

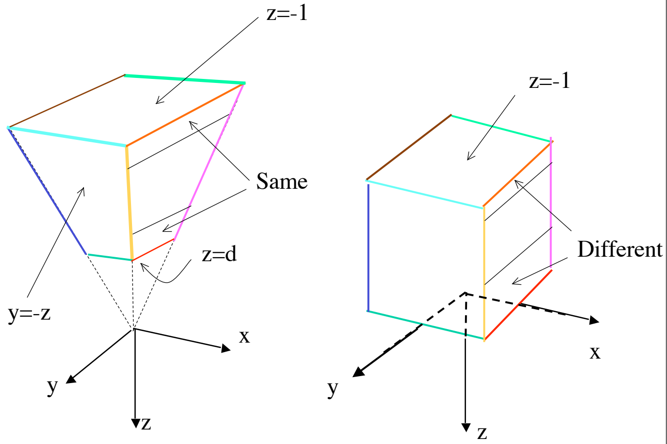
1=>-z

So, the matrix is ?

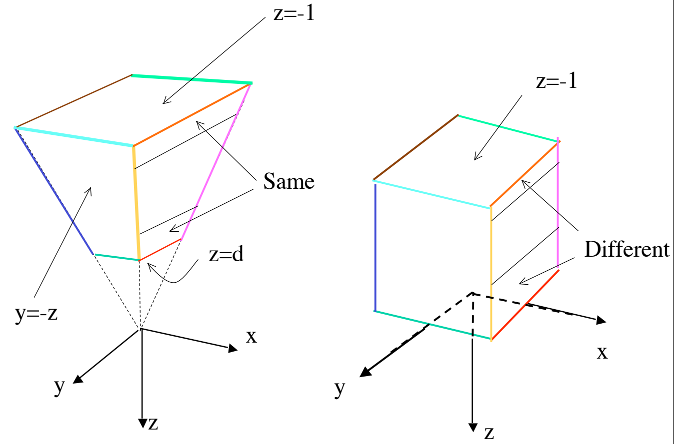
Mapping to standard view volume (additional comments)

- The mapping from $[z_{min}, -1]$ to [0,-1] is non-linear. (Of course, there exists a linear mapping, but not if we want everything else to work out nicely in h.c.).
- So a change in depth of $\triangle$ D at the near plane maps to a larger depth difference in screen coordinates than the same $\triangle$ D at the far plane.
- But order is preserved (important!); the function is monotonic (proof?).
- And lines are still lines (proof?) and planes are still planes (important!).

Transforming canonical frustum to box



Transforming canonical frustum to box



Clipping in homogeneous coordinates

- We have a cube in (x,y,z), but it is **not** a cube in homogeneous coordinates, so we must divide if we want to take advantage of this particularly nice clipping situation.
- However, dividing before clipping might be inefficient if many points are excluded, so we often clip in homogeneous coordinates.

Clipping in homogeneous coord.'s

- Write h.c.'s in caps, ordinary coords in lowercase.
- Consider case of clipping stuff where x>1, x<-1
- Rearrange clipping inequalities:

$$\left(\frac{X}{W}\right) > 1 \qquad \left(\frac{X}{W}\right) < -1$$

becomes

$$X > W, \quad X < -W, \quad W > 0$$

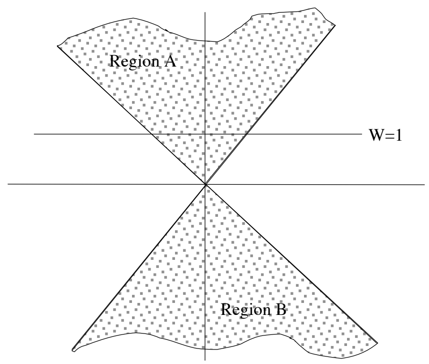
AND

$$X < W, \quad X > -W, \quad W < 0$$

(So far W has been positive, but negatives occur if we further overload the use of h.c.'s)

Clipping in homogeneous coord.'s

The clipping volume in cross section



Clipping in homogeneous coord.'s

- If we know that W is positive (the case so far!), simply clip against region A
- If we are using the h.c. for additional deferred division, then W can be negative.
- If W is negative, then we use region B. The clipping can be done by negating the point, and clipping against A, due to the nature of A and B.
- Case where object has both positive and negative W is a little more complex.
- Notice that the actual clipping computations are not that different from the case in Plan A---no free lunch!

Reminder of the last steps

In both plans we need to project into 2D.

If we are working in the canonical view space, then we project using the standard camera model (easy) and divide

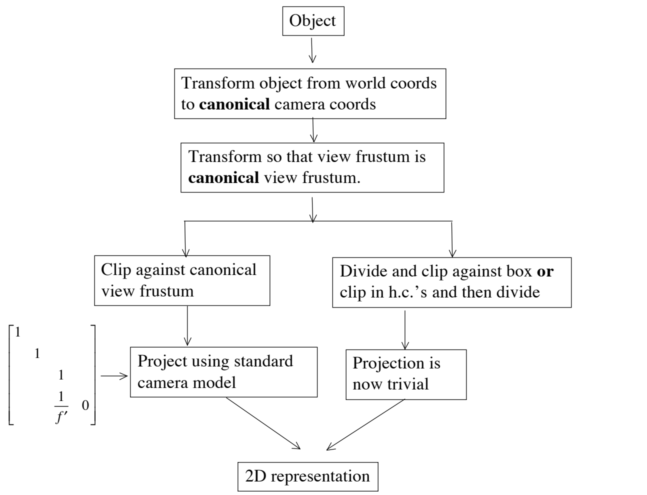
Recall that the matrix for the standard camera model using homogeneous coordinates is:

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & \frac{1}{f'} & 0 \end{bmatrix}$$

Reminder of the last steps

If we are working in homogenous coordinates, then we first divide and then projection is even easier (ignore z coordinate).

The mapping to the box—which was complete once the division was done—implicitly did the perspective projection—essentially we transformed the world so that orthographic projections holds.



Reminder of the last steps

Finally, we may need to do additional 2D transformations.

In the canonical frustum case, our (x,y) coordinates are relative to (-f',f'). They need to be mapped to the viewport (possibly implicitly by the graphics package).

In the canonical box case, our (x,y) coordinates are relative to (–1,1). They need to be mapped to the viewport (possibly implicitly by the graphics package).