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:

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

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

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:

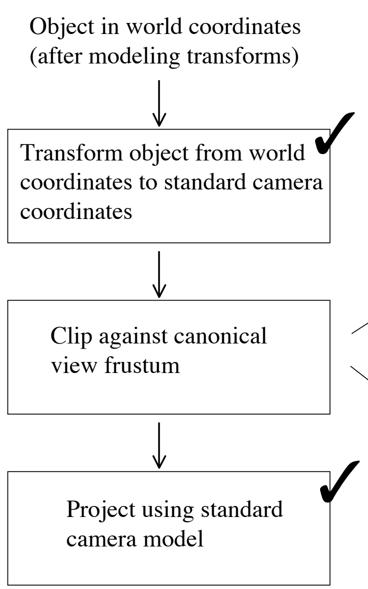
Polygon edge crosses clip **plane** going from out to in
- emit crossing, next vertex

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”)



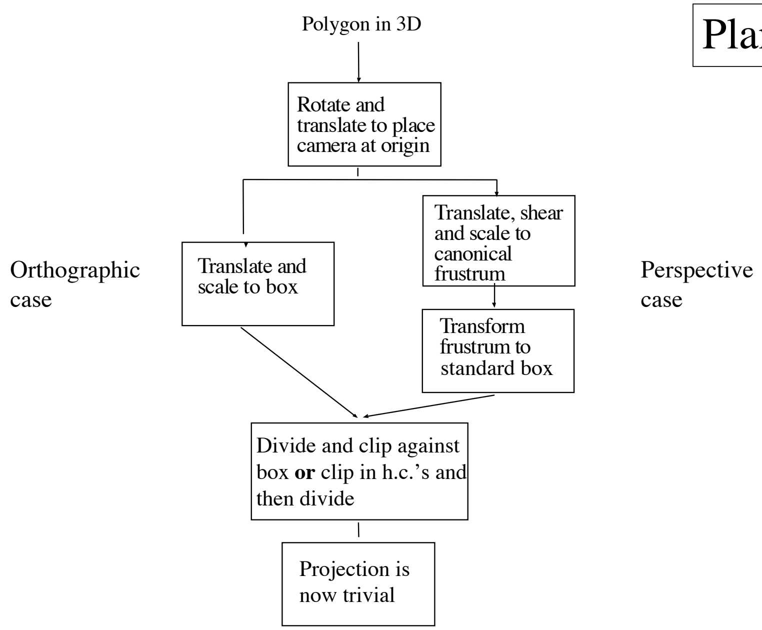
Plan A: Clip against canonical frustum (relatively easy—we chose the canonical frustum so that it would be easy!)

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

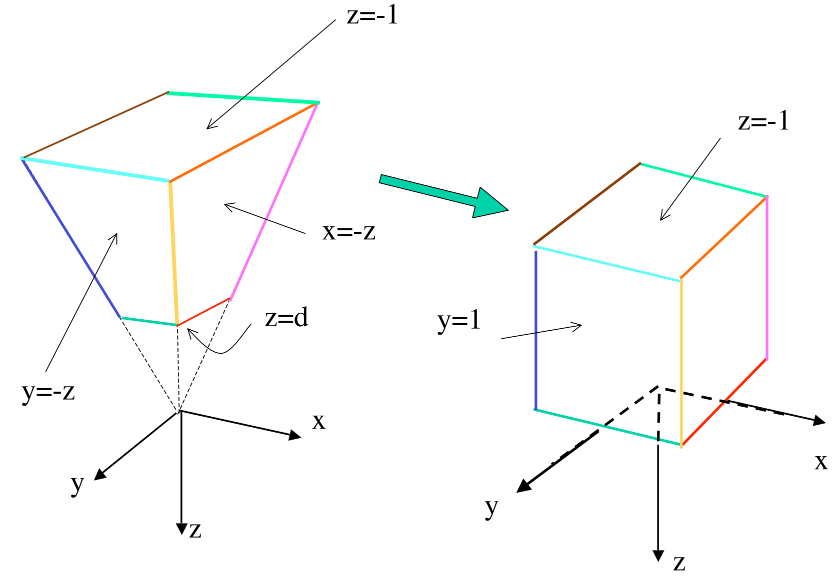
Plan B: Clipping in homogenous coords

- 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

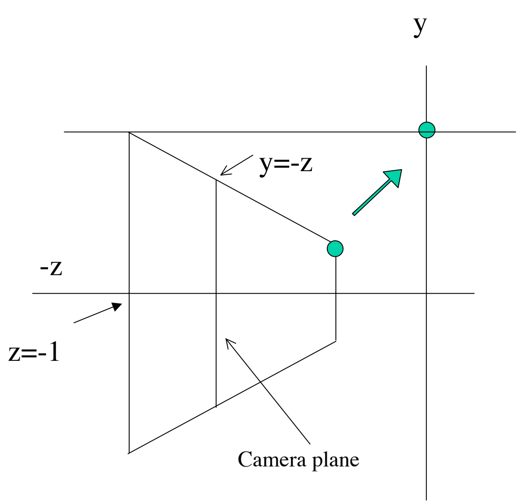
Plan B



Transforming canonical frustum to box

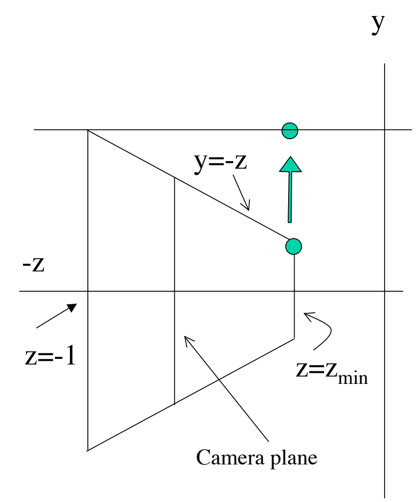


Transforming canonical frustum to box



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

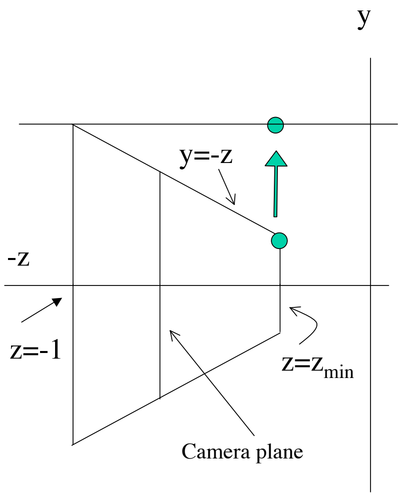
Transforming canonical frustum to box



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.

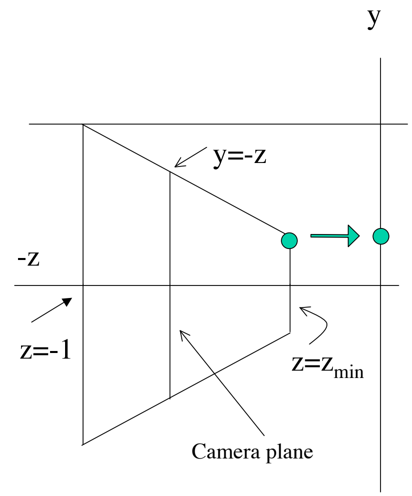
On bottom, y—> -1 so scaling is (–1/y). Recall that y=z there.

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 ?

In h.c.,

x=>x

y=>y

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

1=>-z

So, the matrix is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1+z_{\min}} & \frac{-z_{\min}}{1+z_{\min}} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

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