CENG 477
Introduction to Computer Graphics

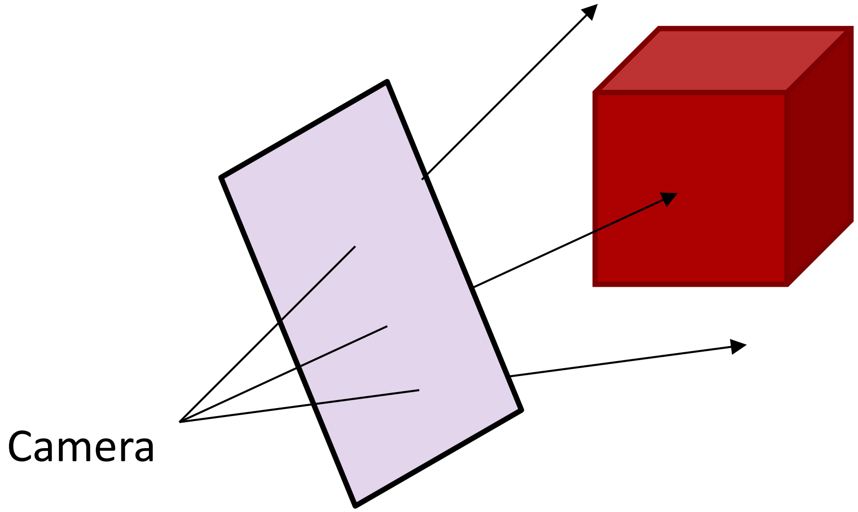
Viewing Transformations

Introduction

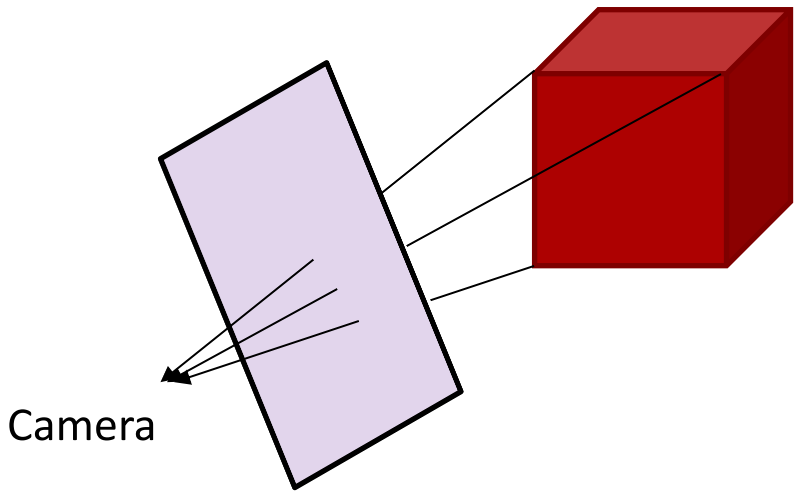
- Until now, we learned how to position the objects in the 3D world space by modeling transformations
- With viewing transformations, we position the objects on a 2D image as seen by a camera with arbitrary position and orientation
- Composed of three parts:
  - Camera (or eye) transformation
  - Projection transformation
  - Viewport transformation

Introduction

- With viewing transformations, we are now transitioning from the backward rendering pipeline (aka. ray tracing) to forward rendering pipeline (aka. object-order, rasterization, z-buffer)



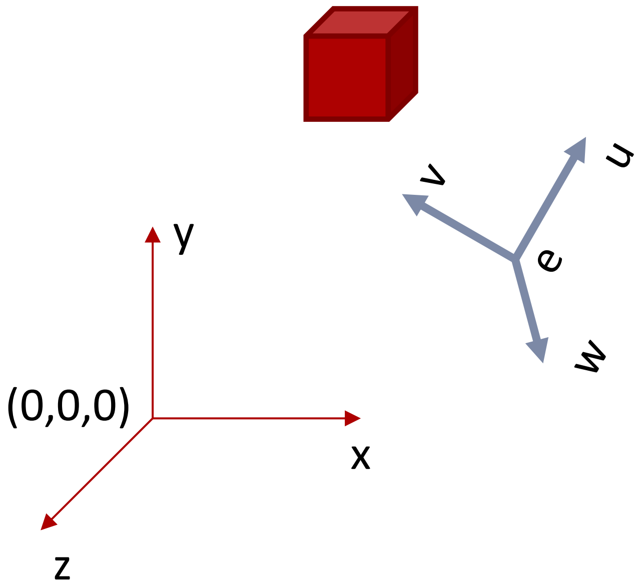
Backward rendering



Forward rendering

Camera Transformation

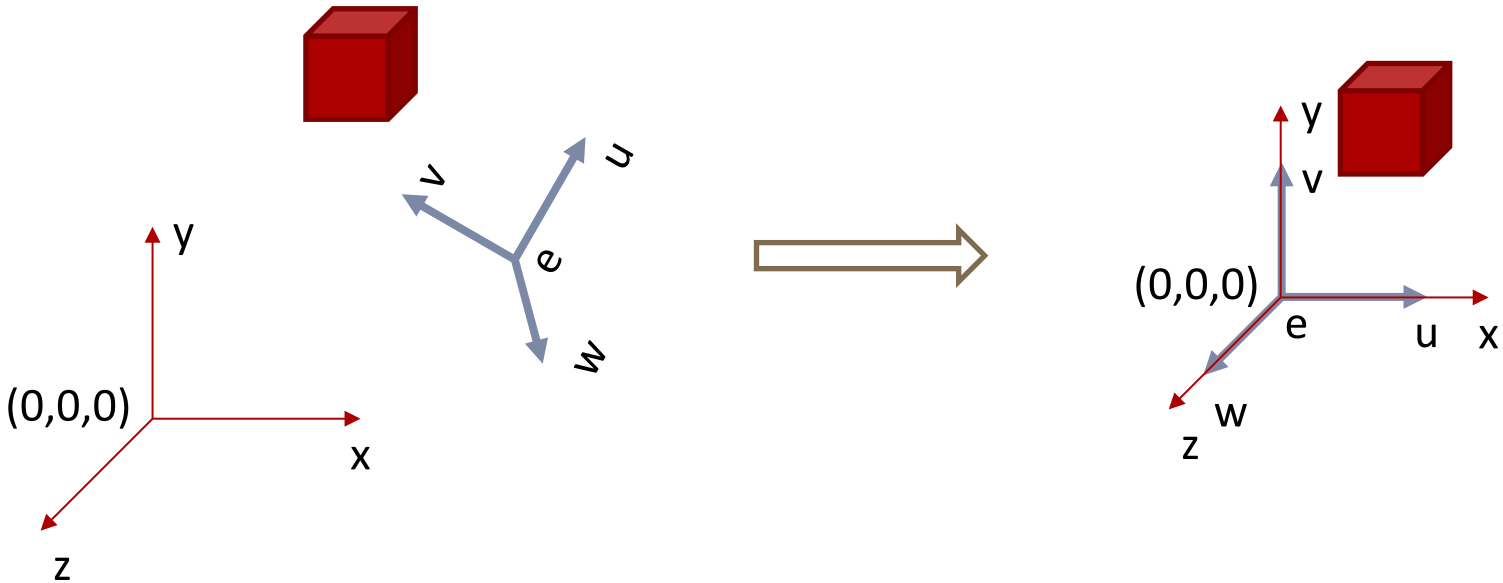
- **Goal:** Given an arbitrary camera position **e** and camera vectors **uvw**, determine the camera coordinates of points given by their world coordinates



What are the coordinates of this cube with respect to the **uvw** CS?

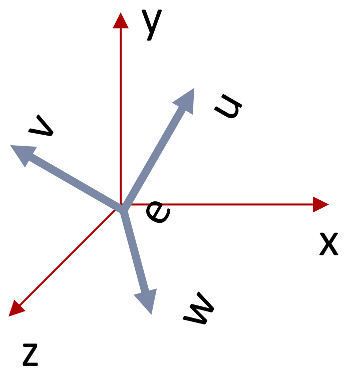
Camera Transformation

- Transform everything such that **uvw** aligns with **xyz**



Camera Transformation

- **Step 1:** Translate **e** to the world origin (0, 0, 0)

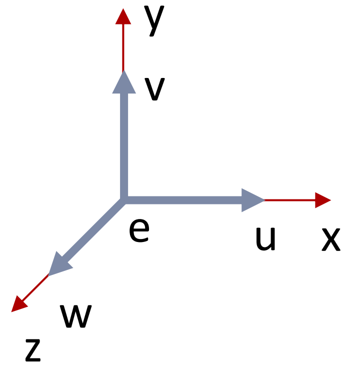


$$T = \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Camera Transformation

- **Step 2:** Rotate **uvw** to align it with **xyz**:

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



We already learned how to do this in modeling transformations!

Camera Transformation

- The composite camera transformation is:

$$M_{cam} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

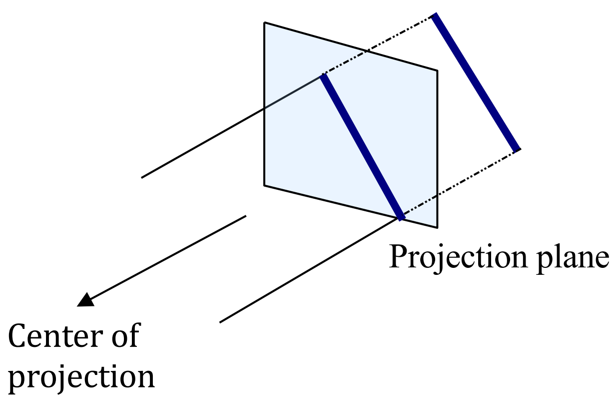
$$M_{cam} = \begin{bmatrix} u_x & u_y & u_z & -(u_x e_x + u_y e_y + u_z e_z) \\ v_x & v_y & v_z & -(v_x e_x + v_y e_y + v_z e_z) \\ w_x & w_y & w_z & -(w_x e_x + w_y e_y + w_z e_z) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Camera Transformation

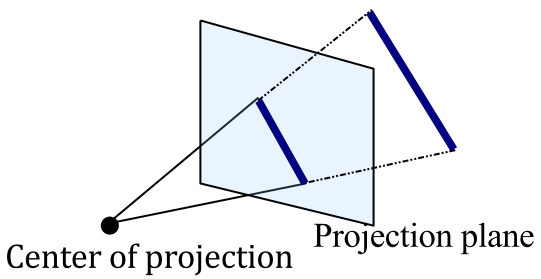
- When points are multiplied with this matrix, their resulting coordinates will be with respect to the **uvw**-**e** coordinate system (i.e. the camera coordinate system)
- Next, we need to apply a projection transformation

Projection Transformation

- Projection transformations depend on the shape of the viewing volume
- Two most commonly used transformations are:
  - Orthographic (parallel) projection
  - Perspective projection



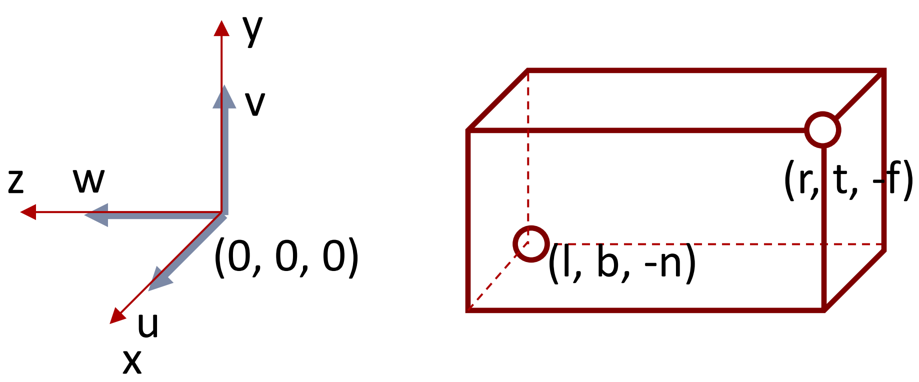
Orthographic



Perspective

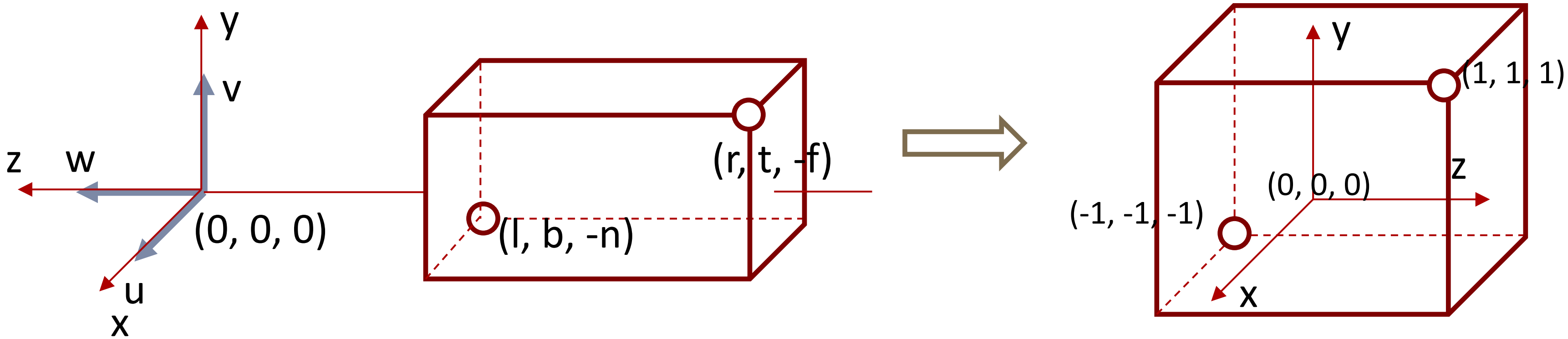
Orthographic Projection

- Defined by a rectangular viewing volume
- Objects inside this volume will be visible (unless they are occluded by other objects)
- This volume will eventually get projected to the screen



Orthographic Projection

- In orthographic (and perspective) projection, our goal is to transform a given viewing volume to the canonical viewing volume (CVV):

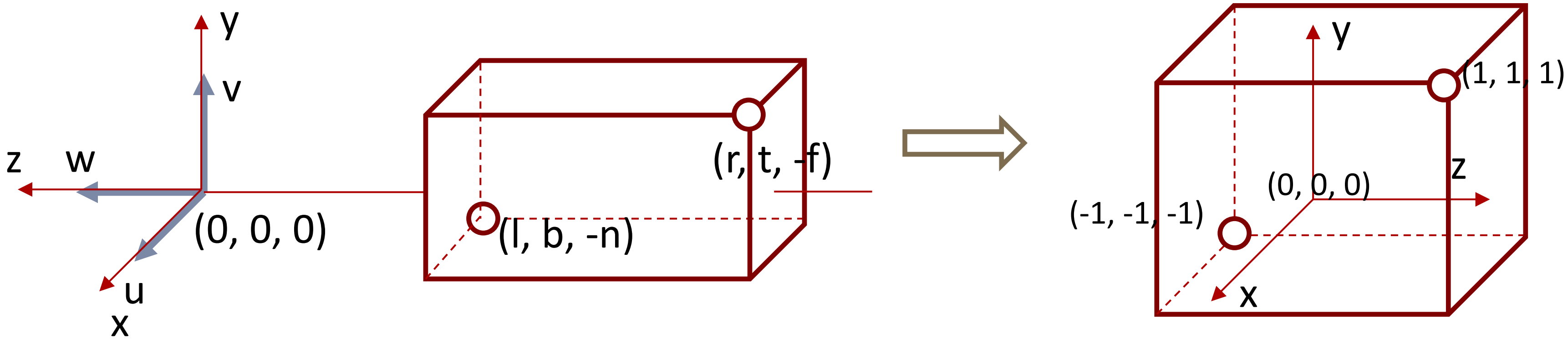


Note that n and f are typically given as *distances* which are always positive and because we are looking towards the –z direction, the actual coordinates become –n and -f

Think of it as compressing a box

Orthographic Projection

- In orthographic (and perspective) projection, our goal is to transform a given viewing volume to the canonical viewing volume (CVV):



Also note the change in the z-direction. This makes objects further away from the camera to have larger z-values. In other words, CVV is a left-handed coordinate system.

Orthographic Projection

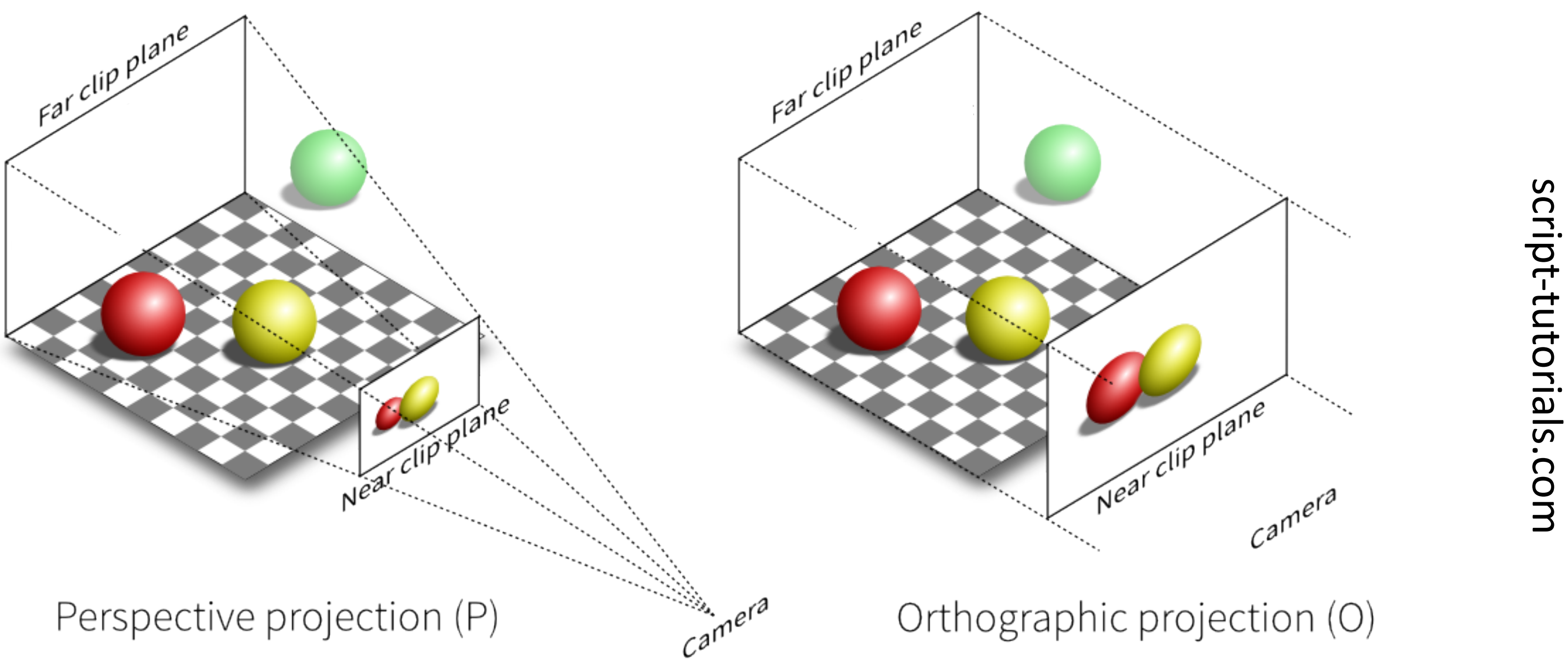
- We need to map the box with corners at (l, b, -n) and (r, t, -f) to the (-1, -1, -1) and (1, 1, 1) of the CVV
- This is accomplished by the following matrix:

$$M_{orth} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Make sure you understand how to derive this!

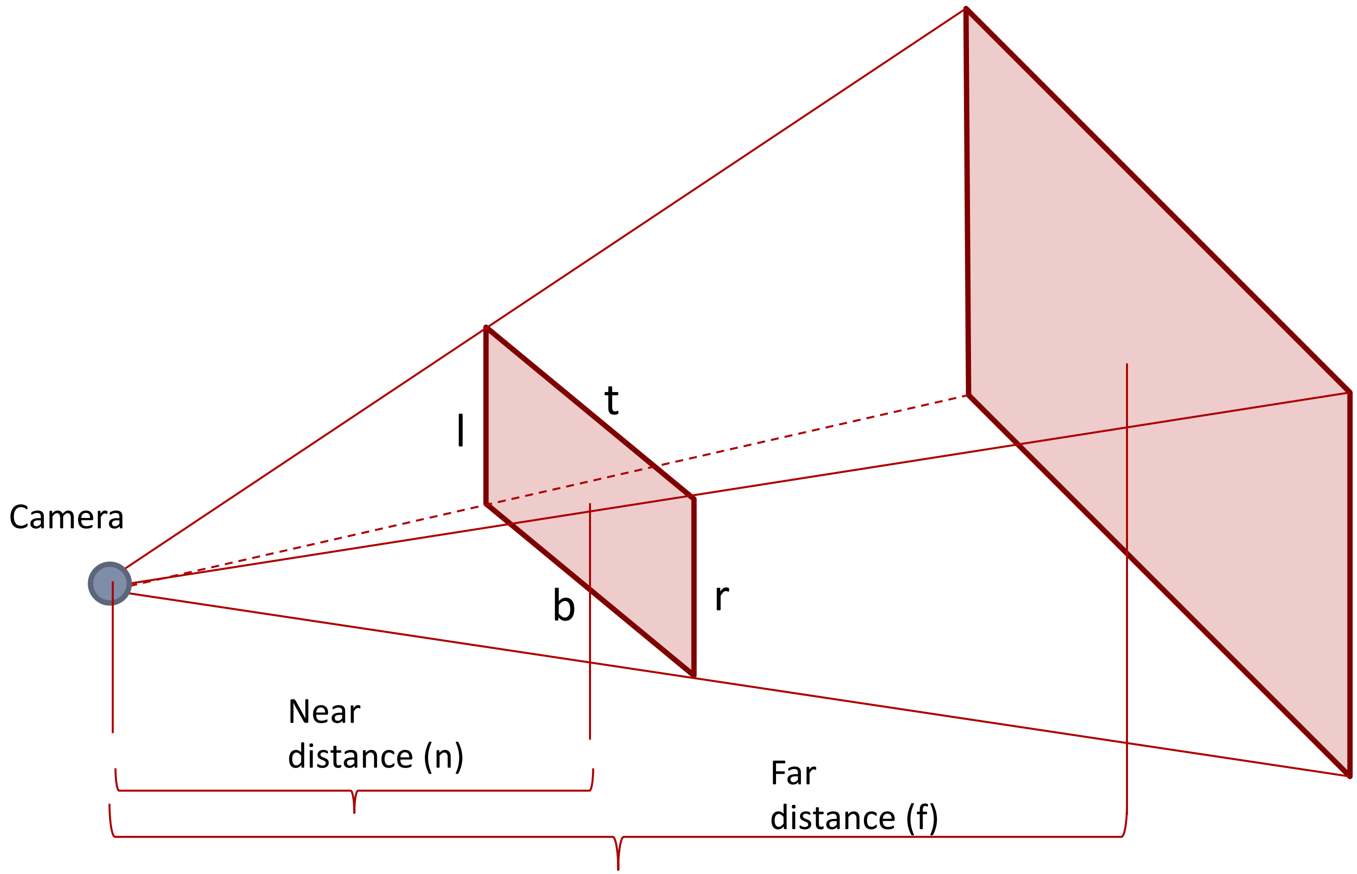
Perspective Projection

- Perspective projection models how we see the real world
  - Objects appear smaller with distance



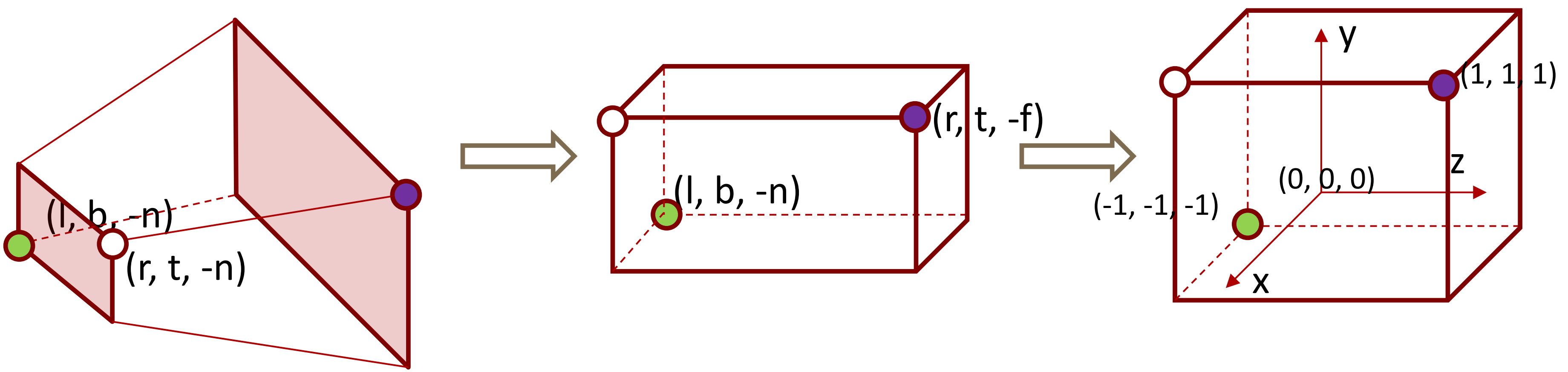
Perspective Projection

- We still have the same 6 parameters



Perspective Projection

- To map to the canonical viewing volume (CVV), we take a two step approach:
  - **Step 1:** Map perspective to orthographic viewing volume
  - **Step 2:** Map orthographic to CVV

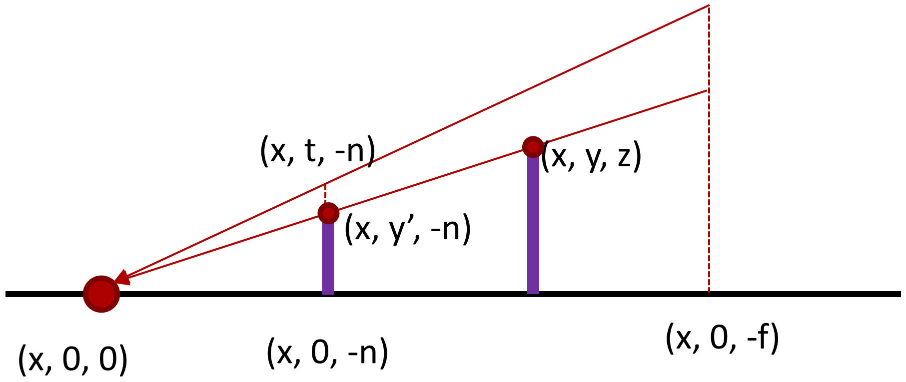


Think of this as compressing a box where you have to apply more pressure towards the back

We already know how to perform the second step!

Perspective Projection

- The key observation is that more distant objects should shrink proportional to their distance to the camera
- Here is a side view (therefore x is constant):



What is y'?

$$\frac{y'}{y} = \frac{-n}{z} \Longrightarrow y' = \frac{-n}{z} y$$

The same geometrical config. applies to the x dimension as well:

$$\frac{x'}{x} = \frac{-n}{z} \Longrightarrow x' = \frac{-n}{z} x$$

Let's ignore the z dimension for the moment

Perspective Projection

- How to represent this as a matrix multiplication?

$$M = \begin{bmatrix} -n/z & 0 & 0 & 0 \\ 0 & -n/z & 0 & 0 \\ 0 & 0 & \dots & \dots \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- The problem is our matrix now contains the coordinates of the transformed points
- This requires a different matrix for each point being transformed
  - Very inefficient and requires too much bookkeeping

Perspective Projection

- Homogeneous coordinates (HC) comes to the rescue here
- Remember that in HC:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} kx \\ ky \\ kz \\ k \end{bmatrix} \Longrightarrow \begin{bmatrix} -nx/z \\ -ny/z \\ \ldots \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ \ldots \\ -z \end{bmatrix}$$

- So if the transformation manages to put –z into the last component, we can achieve the desired result (of course after doing the homogeneous divide, also known as perspective divide)

Perspective Projection

- This can also be represented as a matrix multiplication thanks to homogeneous coordinates:

$$M_{p2o} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- Why does this work?

Perspective Projection

- Let's multiply a point [x, y, z, 1]^T with this matrix:

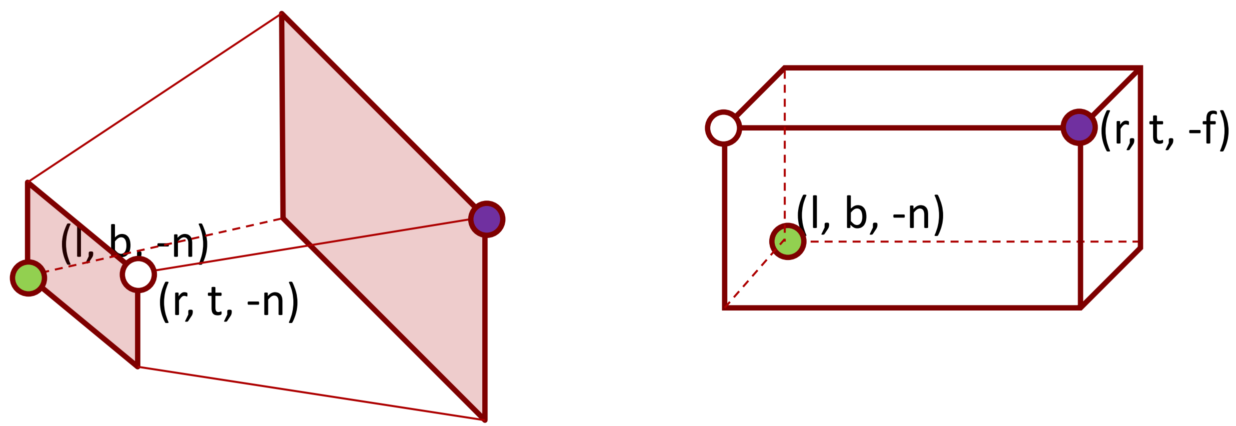
$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ Az + B \\ -z \end{bmatrix}$$

Remember that in homogenous coordinates, scaling all components by the same factor does not change the point. So divide by the last comp.

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ Az + B \\ -z \end{bmatrix} = \begin{bmatrix} -nx/z \\ -ny/z \\ -A - B/z \\ 1 \end{bmatrix}$$

Perspective Projection

- For the z-axis, we have the following constrains:
  - (–n) maps to (–n)
  - (–f) maps to (–f)



- We can solve for A and B using these constrains

Perspective Projection

- Remember that we had:

$$z' = -A - B/z$$

- Now plug (-n) and (-f) and solve for the unknowns:

$$\left.\begin{aligned} -n &= -A + B/n \\ -f &= -A + B/f \end{aligned}\right\} \quad \begin{aligned} \mathrm{A} &= f + n \\ B &= fn \end{aligned}$$

Perspective Projection

- The final perspective to orthographic matrix becomes:

$$M_{p2o} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & f+n & fn \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- Note that this was Step 1
- In Step 2, we multiply this matrix with the orthographic to canonical viewing volume transformation matrix

Perspective Projection

- The final perspective transformation matrix is:

$$M_{per} = M_{orth} M_{p2o}$$

$$M_{per} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

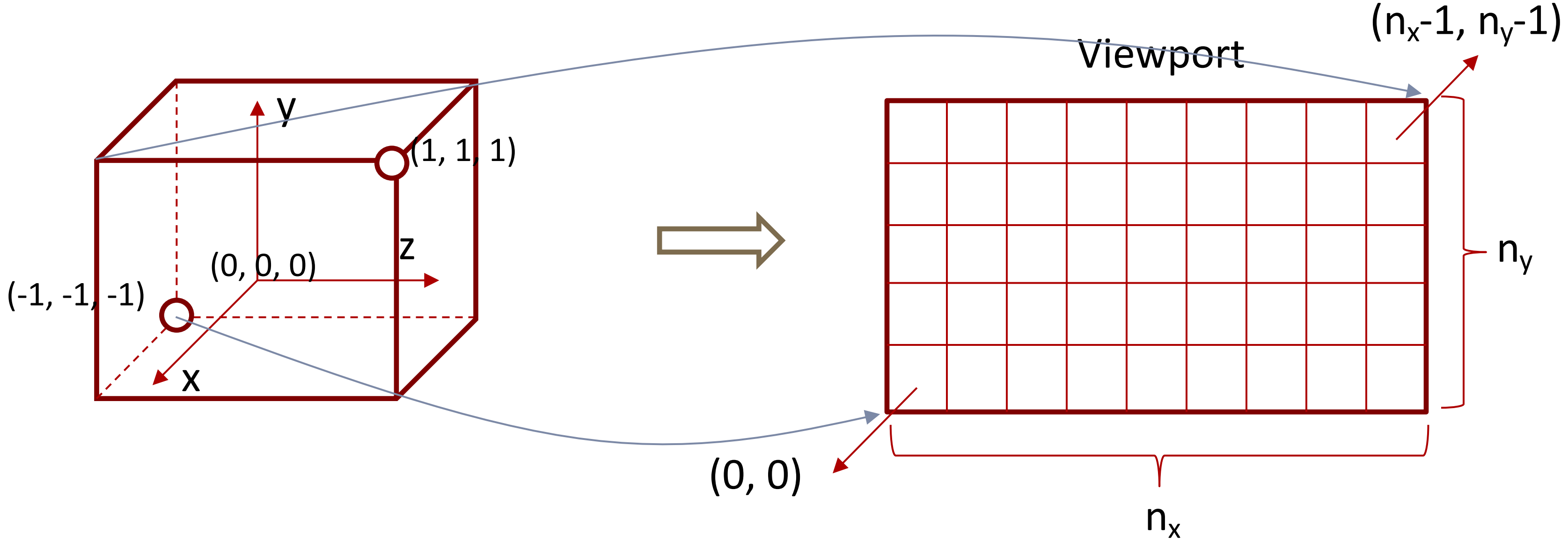
Perspective Divide

- Note that after the perspective projection, *w* coordinates of transformed points may not be *1*
- For the perspective projection to take effect, each point is divided by its *w* coordinate before the next stage
- This is called the perspective divide

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \xrightarrow{\text{Perspective divide}} \begin{bmatrix} x/w \\ y/w \\ z/w \\ 1 \end{bmatrix}$$

Viewport Transformation

- After perspective transformation (and perspective divide), all objects inside the viewing volume are transformed into CVV
- Viewport transformation maps them to the screen (window) coordinates



Viewport Transformation

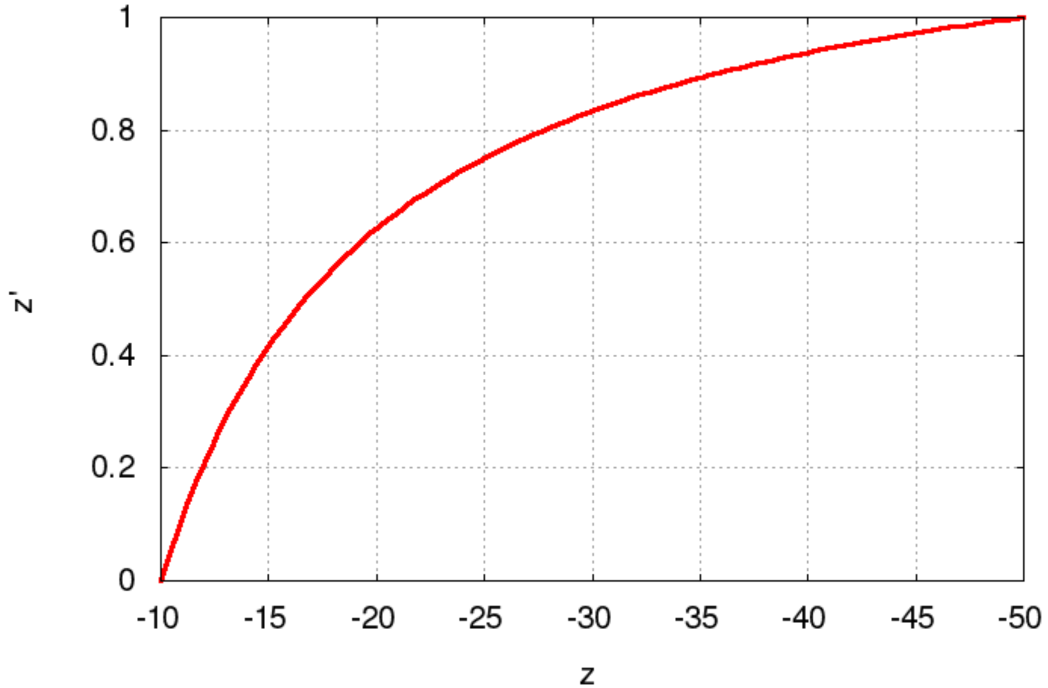
- x values in range [-1,1] are transformed to [-0.5, n_x-0.5]
- y values in range [-1,1] are transformed to [-0.5, n_y-0.5]
- z values in range [-1,1] are transformed to [0,1] for later use in depth testing

$$M_{vp} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x - 1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y - 1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

Note that we don't need to preserve the w component anymore

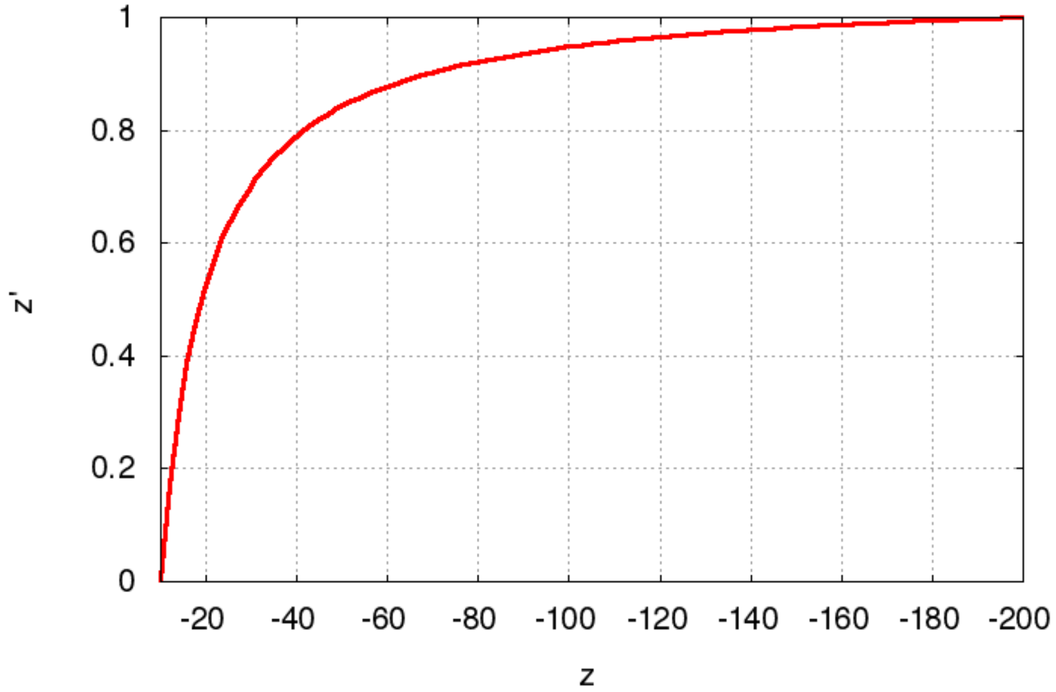
Z-Fighting

- Note that the z-values get compressed to [0, 1] range from the [-n:-f] range
- Observe how it looks for n = 10 and f = 50



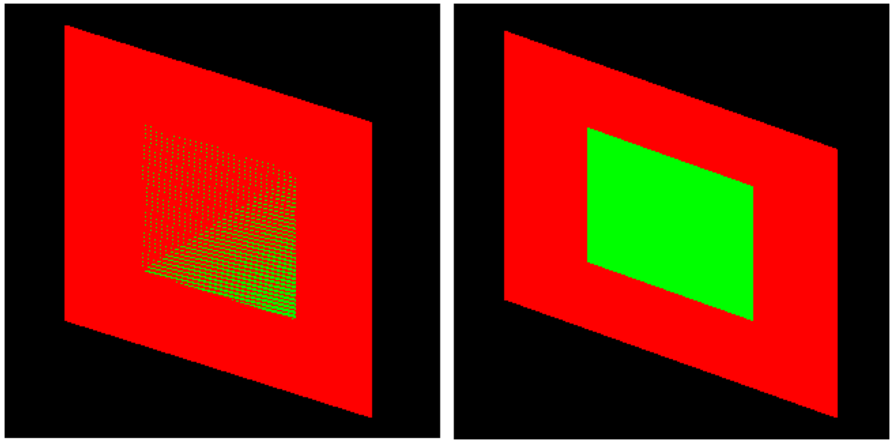
Z-Fighting

- Note that the z-values get compressed to [0, 1] range from the [-n:-f] range
- Observe the same for n = 10 and f = 200



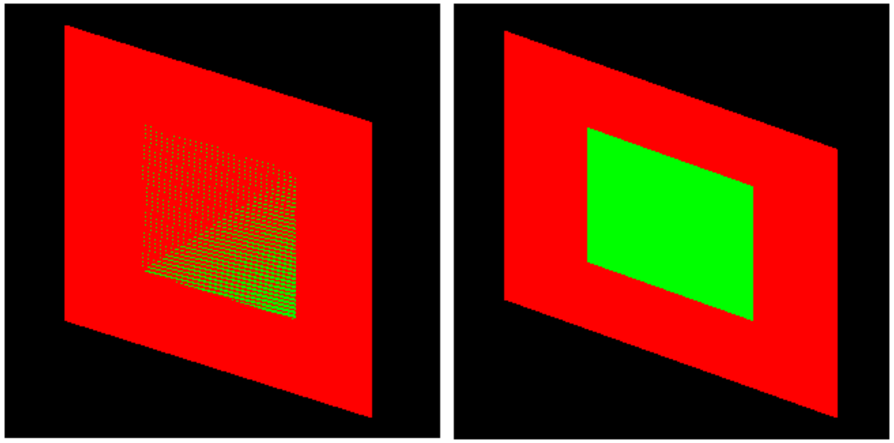
Z-Fighting

- The compression is more severe for with larger depth range
- This may cause a problem known as z-fighting:
  - Objects with originally different z-values get mapped to the same final z-value (due to limited precision) making it impossible to distinguish which one is in front and which one is behind

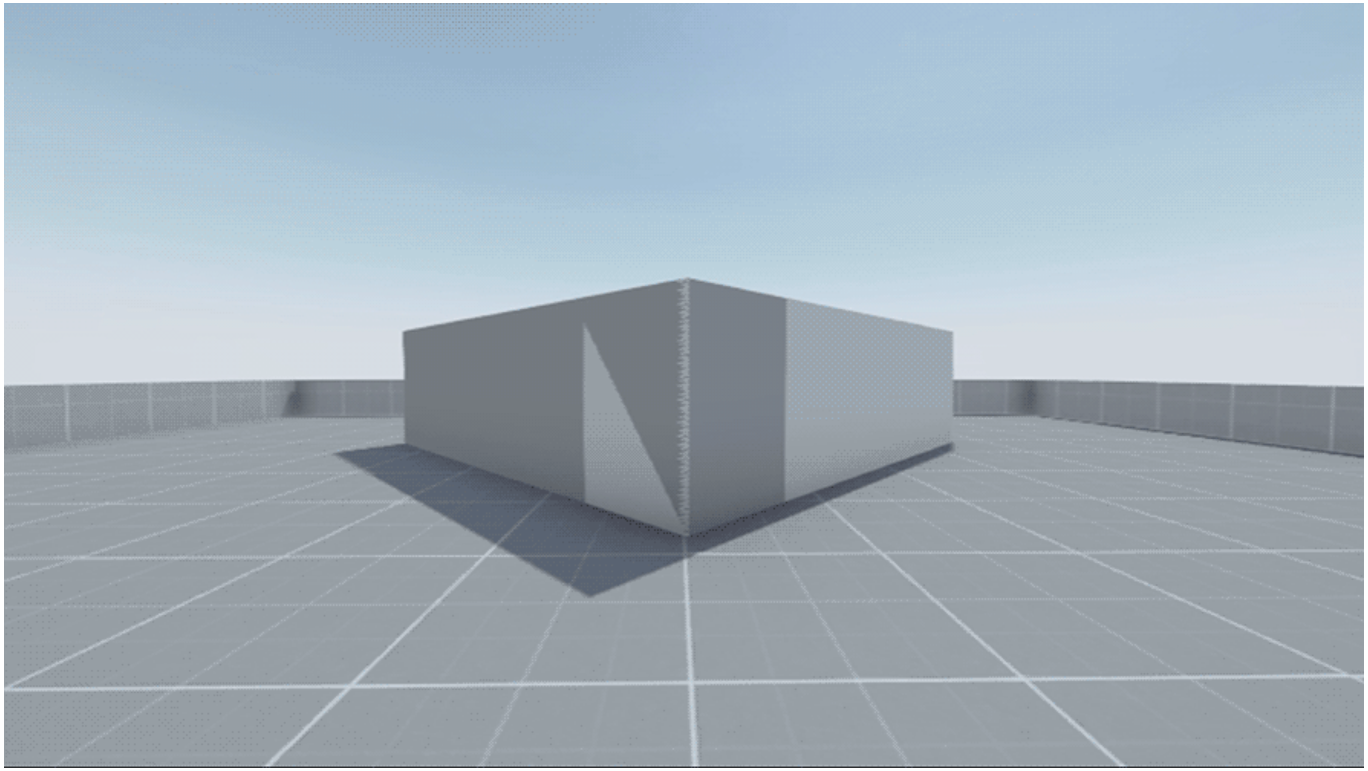


Z-Fighting

- The compression is more severe for with larger depth range
- This may cause a problem known as z-fighting:
  - The problem is even worse if the input z-values are very close to begin with



Z-Fighting



http://wiki.reflexfiles.com/

Z-Fighting

To avoid z-fighting, the depth range should be kept as small as possible for keeping the compression less severe

Summary

- A point $[x_w, y_w, z_w]^T$ in the world coordinate system can be transformed to its viewport coordinates by:

$$\begin{bmatrix} x_{vp} \\ y_{vp} \\ z_{vp} \end{bmatrix} = M_{vp} M_{per} M_{cam} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

perspective divide

Summary

- If the point is defined in its local coordinate system and we are given modeling transformations we use:

$$\begin{bmatrix} x_{vp} \\ y_{vp} \\ z_{vp} \end{bmatrix} = M_{vp} M_{per} M_{cam} M_{model} \begin{bmatrix} x_l \\ y_l \\ z_l \\ 1 \end{bmatrix}$$

perspective divide

Summary

- Remember that we transform only the vertices
- We must reconstruct the triangles (or other primitives) from their projected coordinates
- We must decide:
  - Which pixels belong to a primitive
  - Which fragment of which primitive is closest to the viewer
  - How to compute the color for each pixel, etc.
- Questions of this type are what we will focus on in the following weeks