Rendering

Rasterization and Perspective Projection

CS 415: Game Development

Professor Eric Shaffer

ILLINOIS

Perspective Projection

Perspective projection is one of the ways 3D computer graphics looks 3D

This projection maps 3D scene data to a 2D image

It mimics the way human eyes see

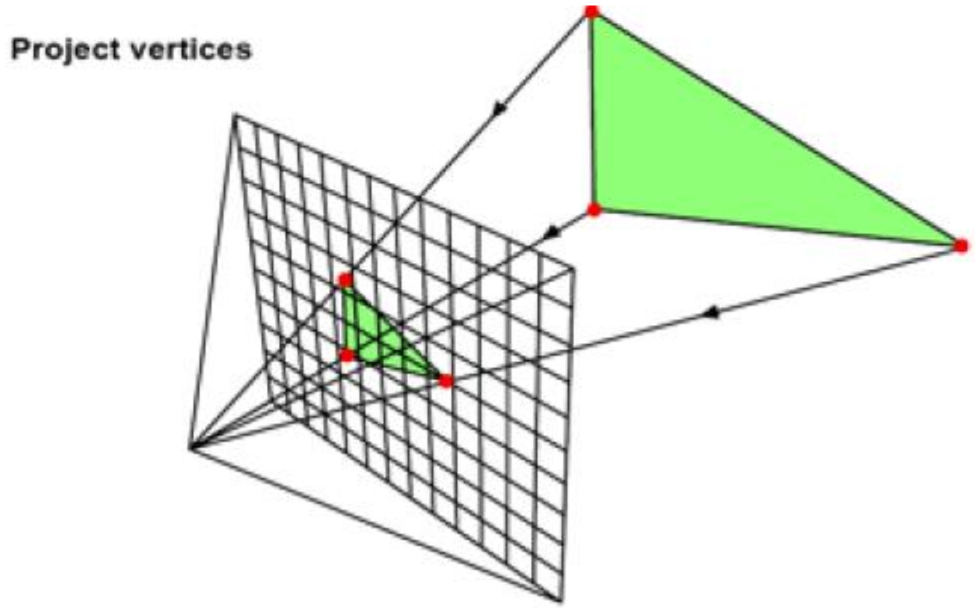
An object will project to smaller size the more distant it is from the eye

Perspective Projection

What is another effect in this picture that makes people see it as 3D?

Rasterization: 3D Computer Graphics

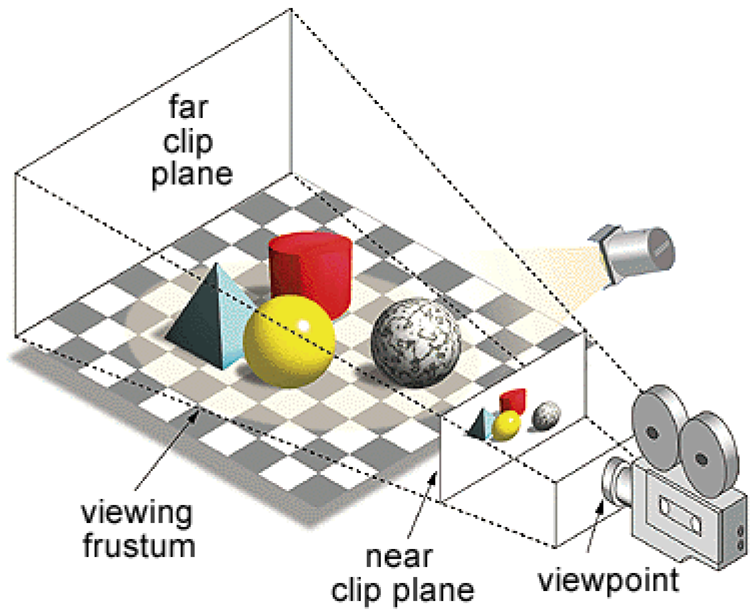
Uses a simple mathematical model of a camera.



Projects 3D geometry to a plane

Maps a rectangle on that plane to a grid of pixels

Scene Data



The camera is part of the scene. It has:

1. A view point (position)
2. An orientation (direction and rotation)
3. A field of view that defines the viewing frustum

Geometry outside the frustum is clipped off and not rendered

The screen can be thought of as part of the scene as well

Each pixel corresponds to some amount of space in the 3D world

Camera Space

The perspective projection in 3D graphics is done in a camera space coordinate system

World space geometry is transformed to camera space

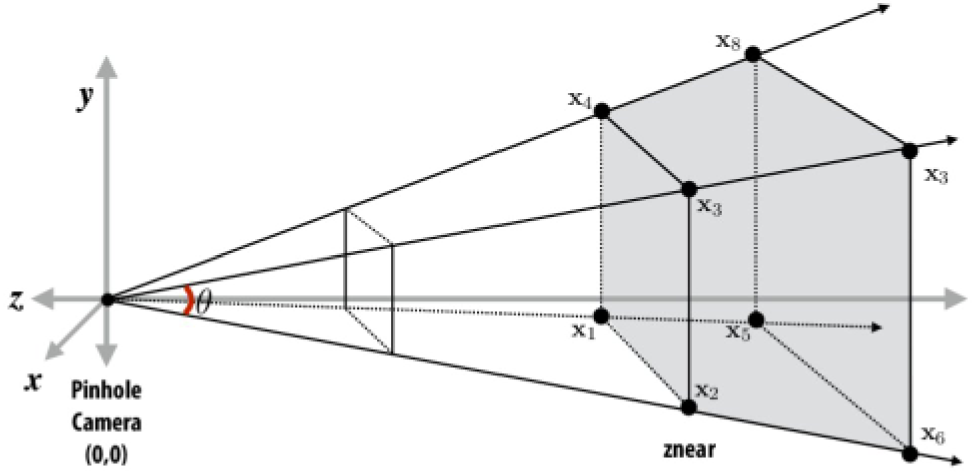
- The camera is at the origin
- The camera is looking down the z axis (usually –z)

The screen is defined as being

- *d* units away from the camera
- Orthogonal to the z axis

So for the screen…

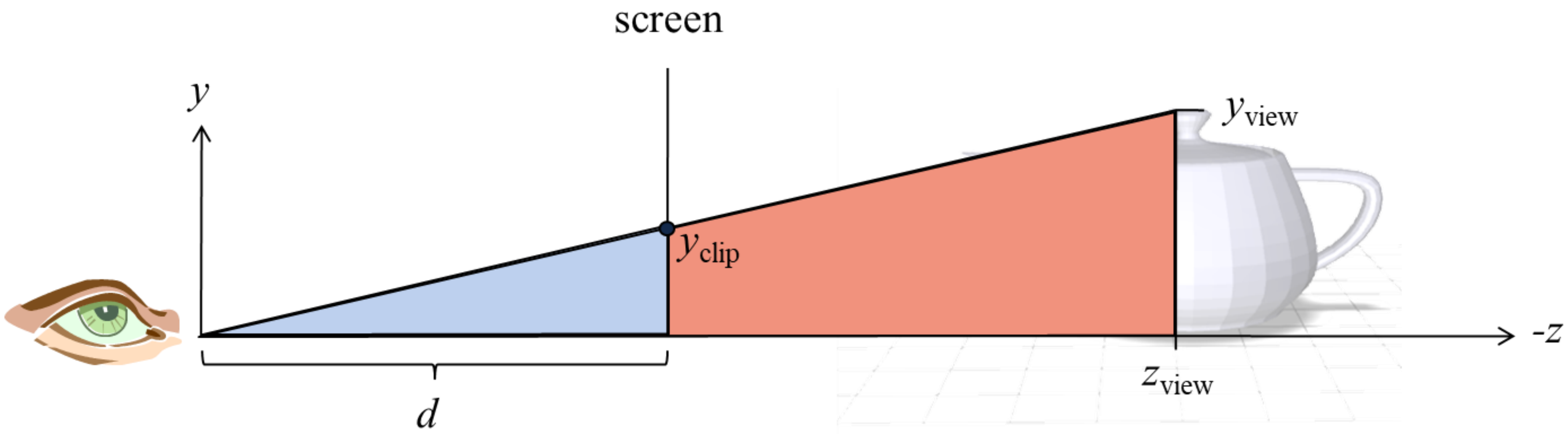
- The y axis is up-down
- The x axis is left-right



Camera-space positions: 3D

Anyone want to guess why we do this?

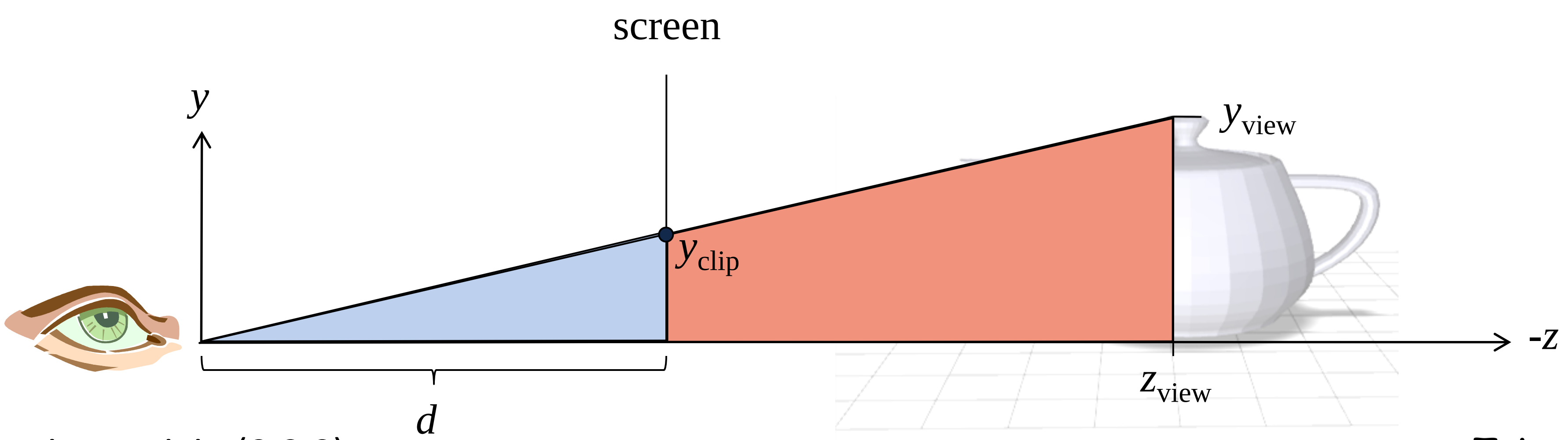
Let’s Do Perspective Projection



We are looking at a slice of 3D space where x=0

- So…z is left & right and y is down & up
- A vertex on a model has y coordinate of yview
- What does that yvuew coordinate project to on the screen?
  - In other words, what is yclip?

Perspective



Eye is at origin (0,0,0)
Screen is distance d from the eye.
Looking down negative z-axis.

The two triangles are *similar*
(two angles are obviously congruent)

This means corresponding sides
are in the same proportions

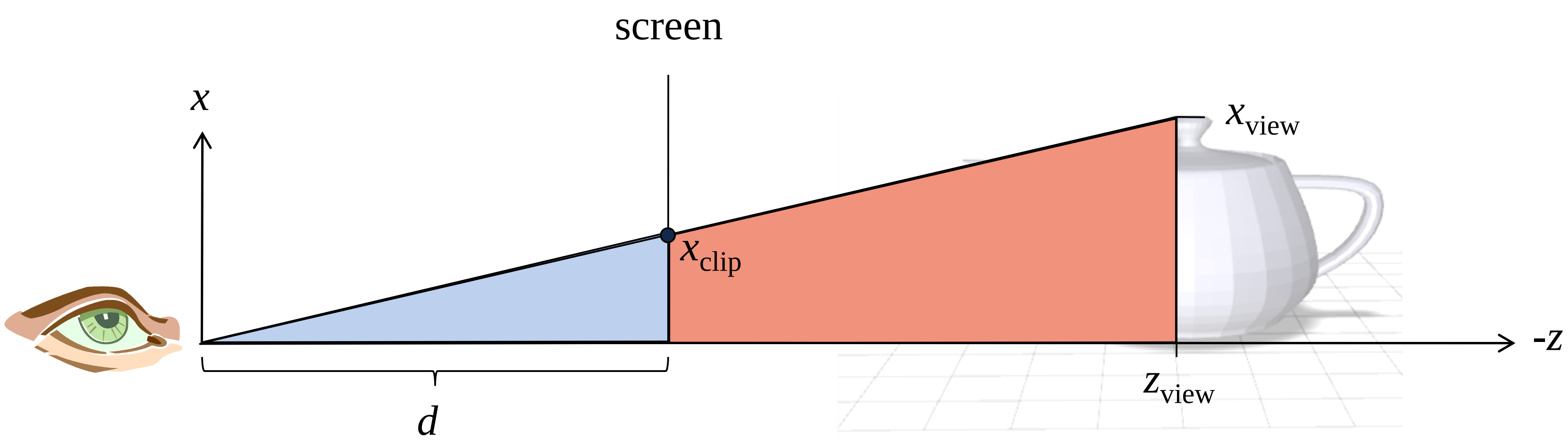
$$\frac{y_{\text{clip}}}{d} = \frac{y_{\text{view}}}{-z_{\text{view}}}$$

$$y_{\text{clip}} = d\,\frac{y_{\text{view}}}{-z_{\text{view}}} = \frac{y_{\text{view}}}{-z_{\text{view}}/d}$$

Zview is coordinate
on the –z axis…

To use that number
as a distance in the
formula we need to
make it positive.

Perspective



Same process derives the projection for the x coordinate.

$$x_{clip} = \frac{x_{view}}{-z_{view}/d}$$

What is z_{clip}?

Perspective Projection

So that's it:

$$x_{clip} = \frac{x_{view}}{-z_{view}/d}$$

$$y_{clip} = \frac{y_{view}}{-z_{view}/d}$$

$$z_{clip} = -d$$

There are More Transformations

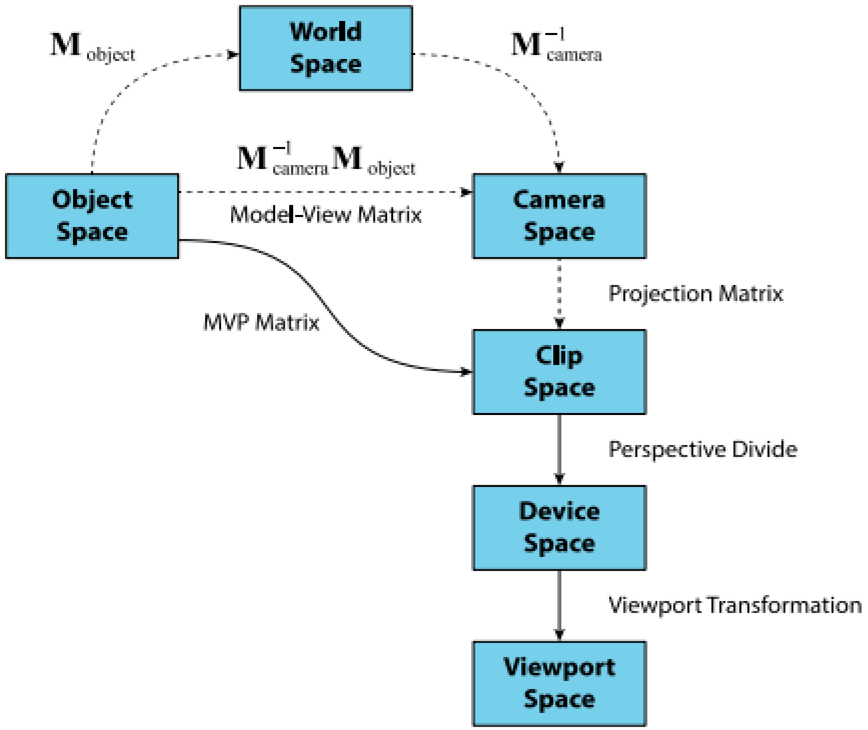
After the projection coordinates are transformed to Normalized Device Coordinates

- Simply by dividing each coordinate by a value
- X and Y are in [-1,1] and z in [0,1]
  - This can vary depending on the system…WebGL vs. D3D12

Then the window-to-viewport transforms to Viewport Space

For a w by h pixel screen

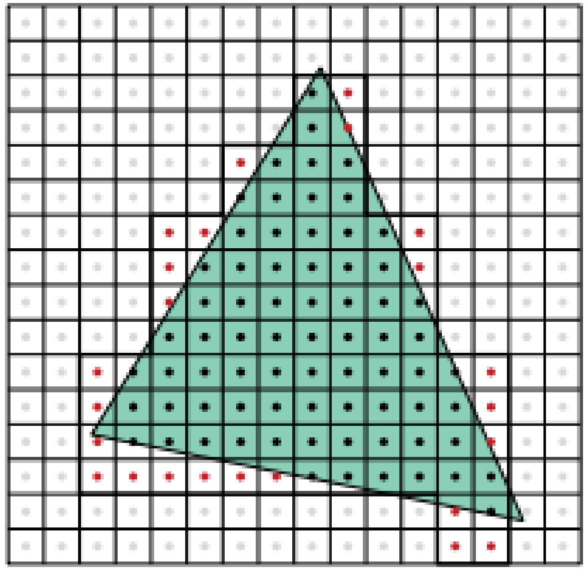
- Coordinates have x in [0,w]
- Coordinates have y in [0,h]



Rasterization

Once a set of triangle vertices have been transformed to viewport space

The triangle can be rasterized



The GPU figures out which pixels have centers inside the triangle edges

Those pixels will get filled with color based the ***shading*** of the triangle

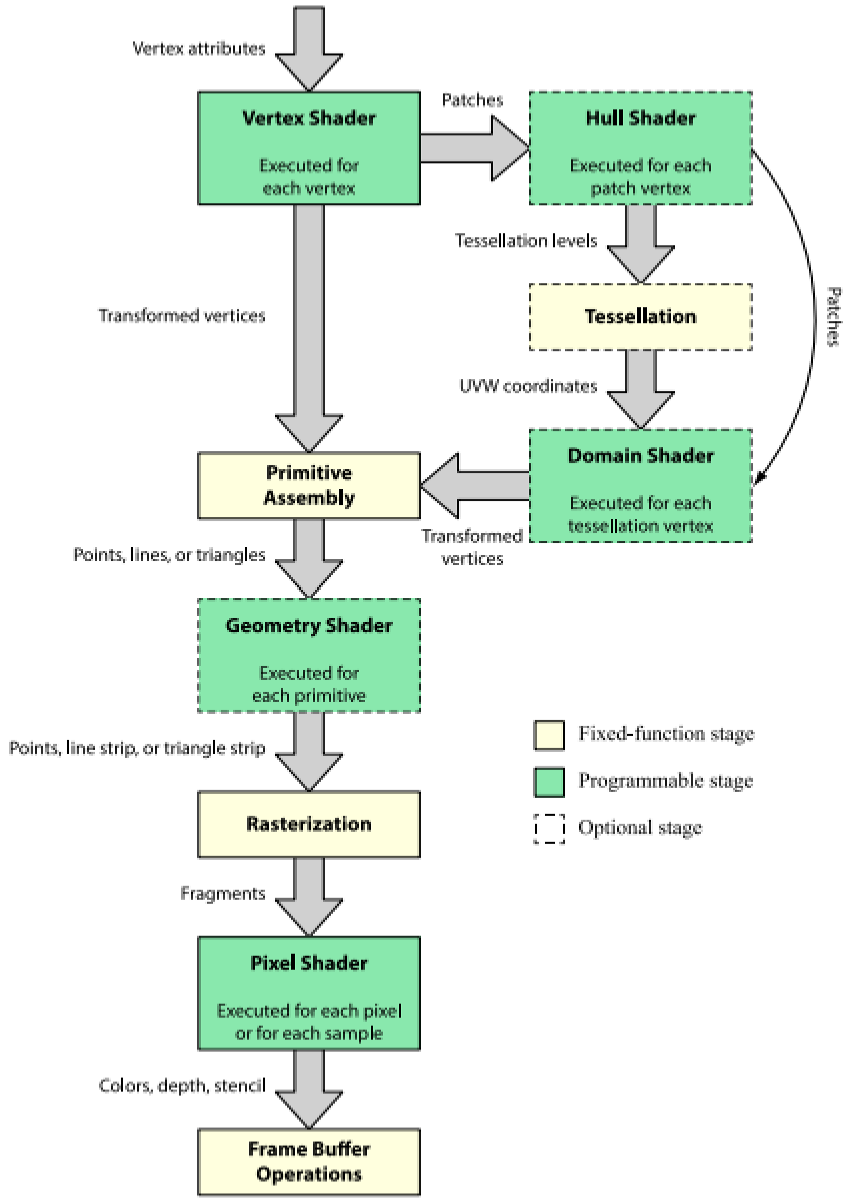
In Practice…It Get More Complicated

Each vertex projection is actually done using a matrix multiplication

- And then scaling the projected coordinates (*the perspective divide*)

This happens on the GPU…

- Using a vertex shader program
- Vertex shader code applies transformations to vertex coordinates
- Lots of cores each running that code process the vertices in parallel



ILLINOIS