3D Graphics and Animation

Space, Camera and Projection

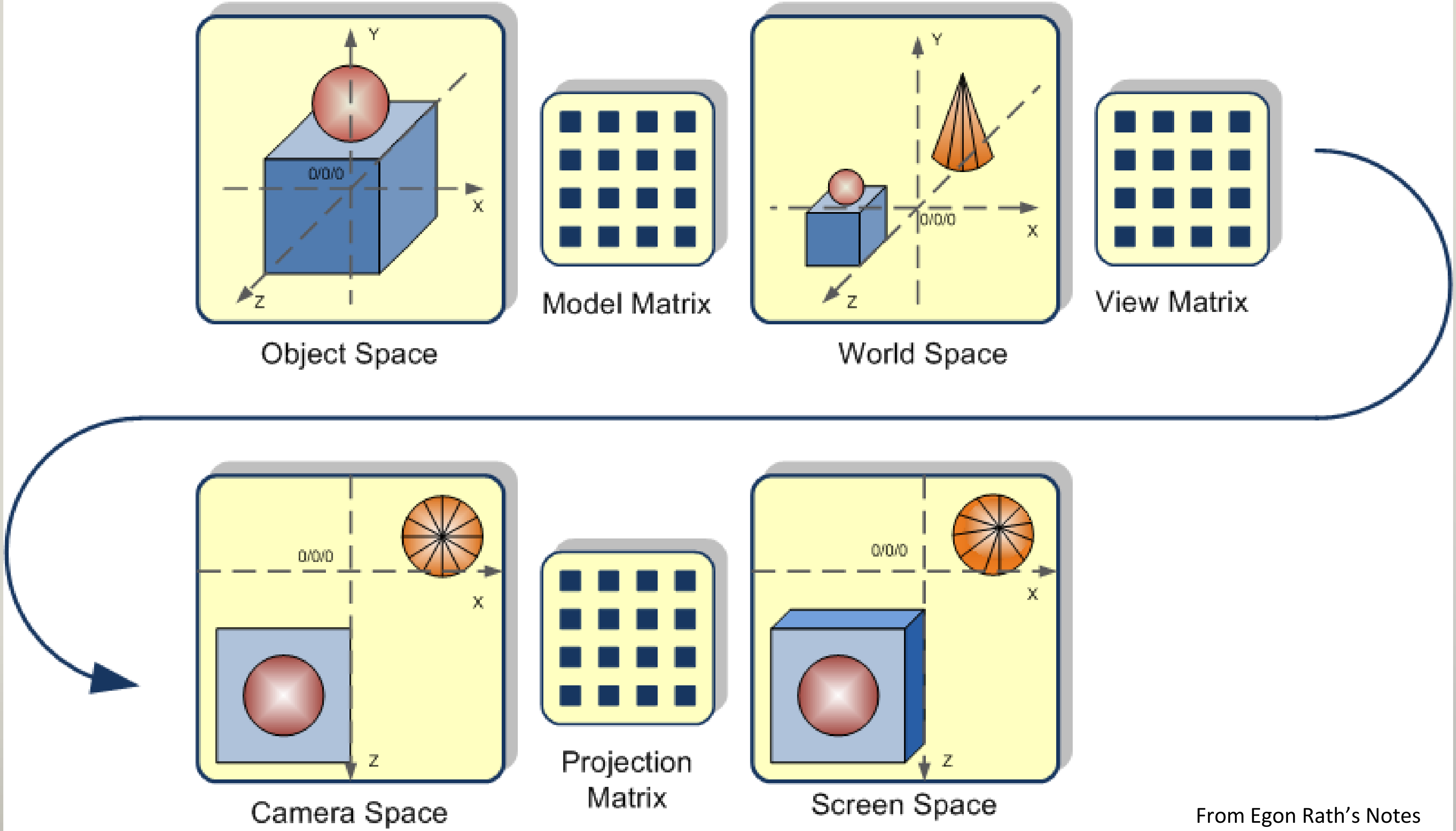
Recap

- In the last lecture, we studied concepts of vector, matrix and transformation for 3D graphics
- These are the basics for understanding objects represented in 3D and basic methods to manipulate them

Different Spaces in Graphics

- Start from 3D object and finally displayed on the 2D screen
- The objects are being transformed to various spaces, they include
  - Object Space
  - World Space
  - Camera Space
  - Screen Space

Different Spaces in Graphics



From Egon Rath's Notes

Different Spaces in Graphics

- To convert between spaces, different matrices are involved
  - **Model Matrix**

    (From object space to world space)
  - **View Matrix**

    (From world space to camera space)
  - **Projection Matrix**

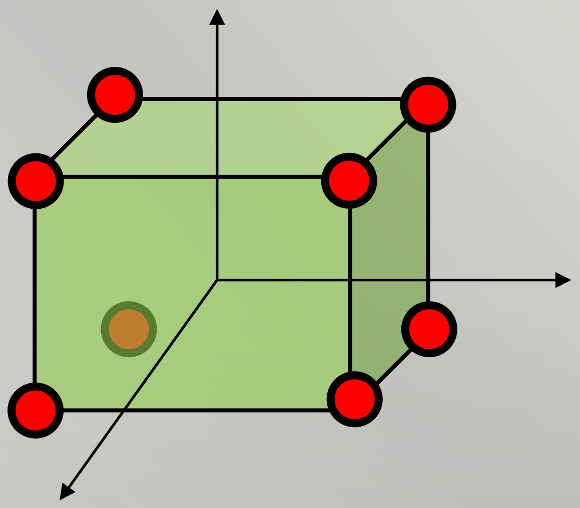
    (From camera space to screen space)

Object Space

- Local coordinate system of the 3D geometrical objects
- In OpenGL, it is the space whenever the 3D geometry is being created
  - E.g. a cube created about origin

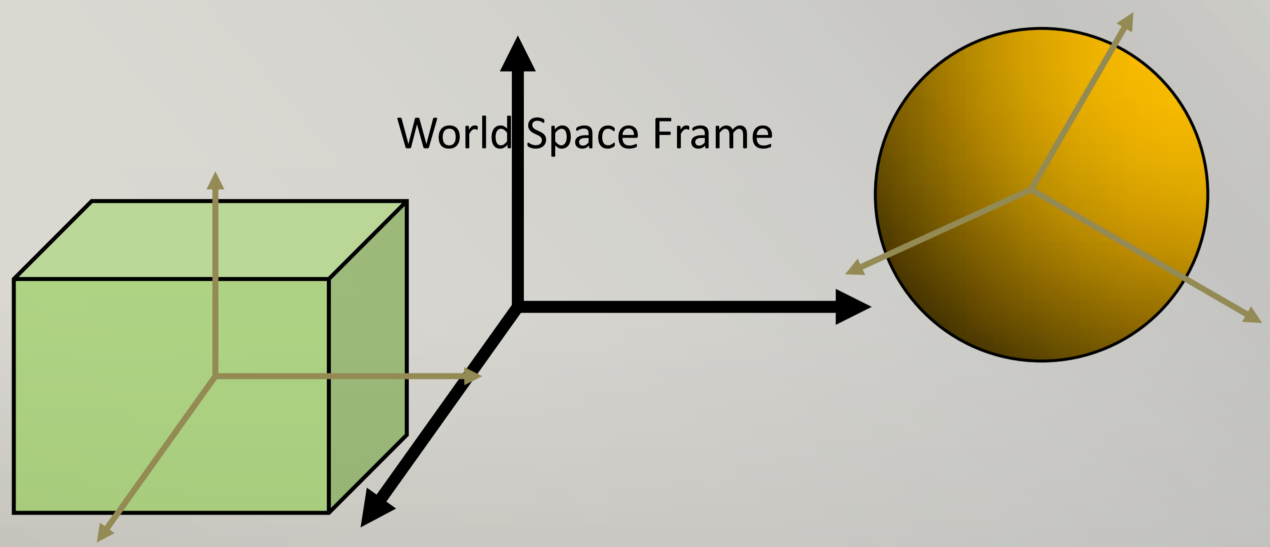
    The vertices' coordinates:

    (-1,1,1), (-1,-1,1), (1,-1,1), (1,1,1), (-1,1,-1), (1,1,-1), (1,-1,-1), (-1,-1,-1)



World Space

- The space where all objects are positioned
  - E.g. our cube are moved to the desired place in the world space's coordinate frame
  - We can do this by multiplying the vertices of the cube with the Model Matrix



Example

- A vertex (-1,1,1) in object space is going to transform into world space by the following Matrix M_{model}

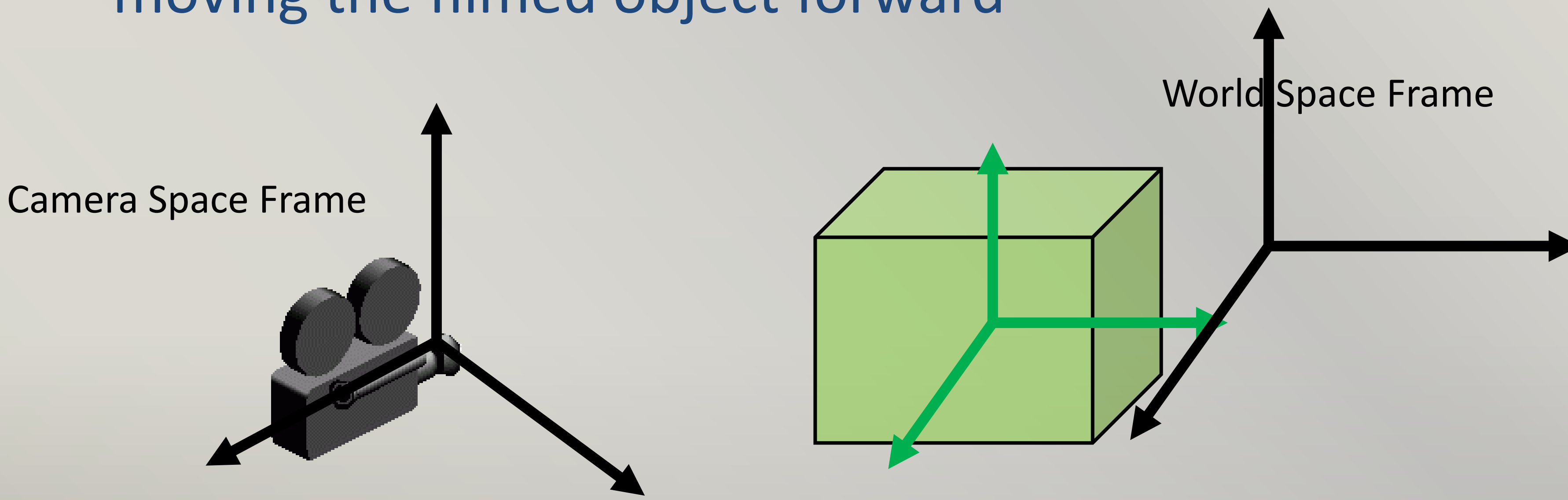
$$M_{model} = \begin{bmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & -3 \\ 0 & 0 & 1 & -6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{model}v = \begin{bmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & -3 \\ 0 & 0 & 1 & -6 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} -1+10 \\ 1-3 \\ 1-6 \\ 1 \end{bmatrix} = \begin{bmatrix} 9 \\ -2 \\ -5 \\ 1 \end{bmatrix}$$

Camera Space / Eye Space

- The space where the camera is being the center (origin)
  - Moving a Video Camera Backward is the same as moving the filmed object forward



Example

- A vertex (9,-2,-5) in world space is going to transform into camera space by the following Matrix M_{view}

$$M_{view} = \begin{bmatrix} 0.707 & -0.707 & 0 & -1 \\ 0.707 & 0.707 & 0 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{view}v = \begin{bmatrix} 0.707 & -0.707 & 0 & -1 \\ 0.707 & 0.707 & 0 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 9 \\ -2 \\ -5 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 9\times 0.707 + 2\times 0.707 - 1 \\ 9\times 0.707 - 2\times 0.707 + 2 \\ -5+2 \\ 1 \end{bmatrix} = \begin{bmatrix} 6.777 \\ 6.949 \\ -3 \\ 1 \end{bmatrix}$$

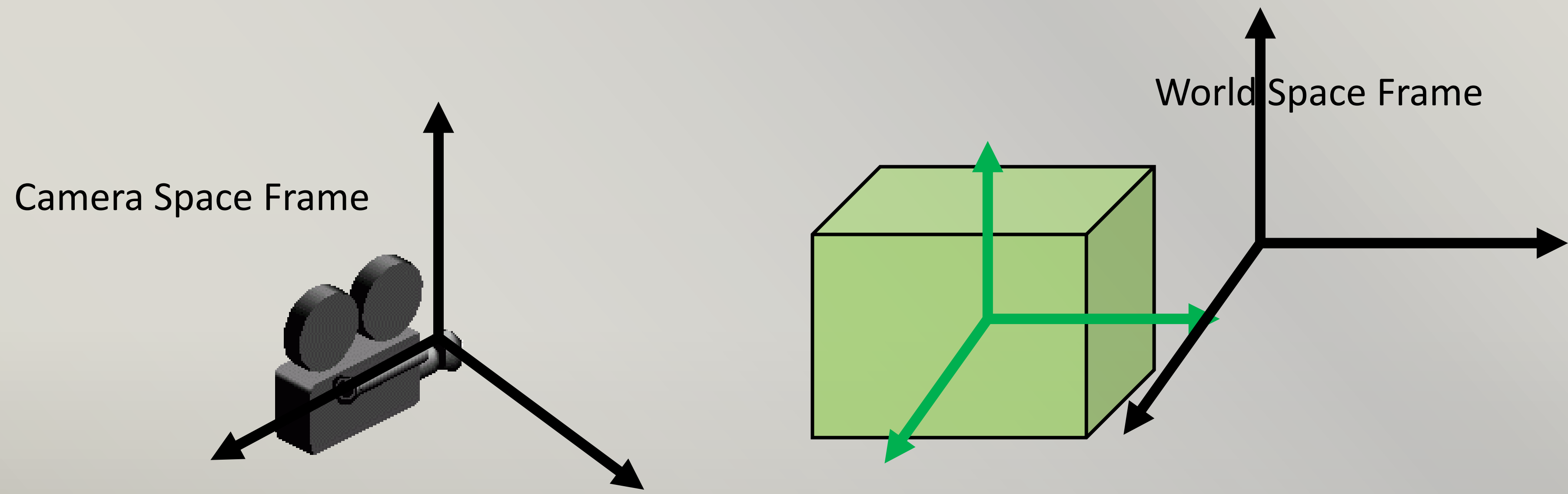
Camera Space / Eye Space

- In some rendering engine, e.g. OpenGL, the Camera is always set at the world's center
  - So, The Model and View Matrix are being combined to form the ModelView Matrix instead
- Using the examples above

$$M_{modelview} = M_{model}M_{view} = \begin{bmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & -3 \\ 0 & 0 & 1 & -6 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.707 & -0.707 & 0 & -1 \\ 0.707 & 0.707 & 0 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Screen Space

- In the camera space, we already have all vertices in a position viewing from the camera
- The last process is the projection of 3D vertices to the 2D coordinates on the screen/film of a camera



Different Spaces in Graphics

- The process of projection is a bit complicated and will be explained in detail in later slides

- Some of the spaces (e.g. world space) are more conceptual than necessary to be explicitly included in the rendering process
  - E.g. in OpenGL world space = camera space

Different Spaces in Graphics

- However, space transformation can simplify the computation in each step
- As things can be computed based on the most convenient coordinate system
- We will see how it benefit the computation of projection if we do that in the camera space
  - i.e. the formation of projection matrix

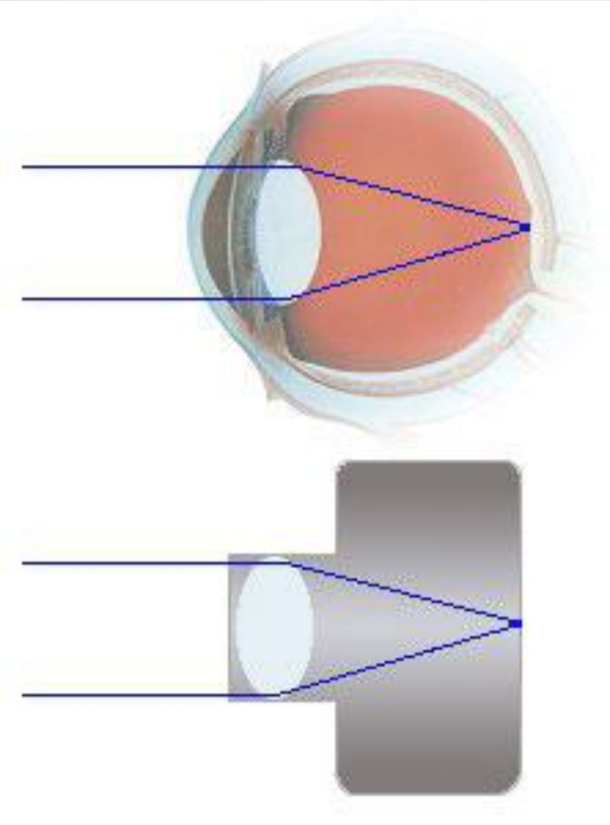
Camera and Projection

Camera in Real World

- Both camera and eyes contains
  - LENS
  - Projection plane

    Camera: Film

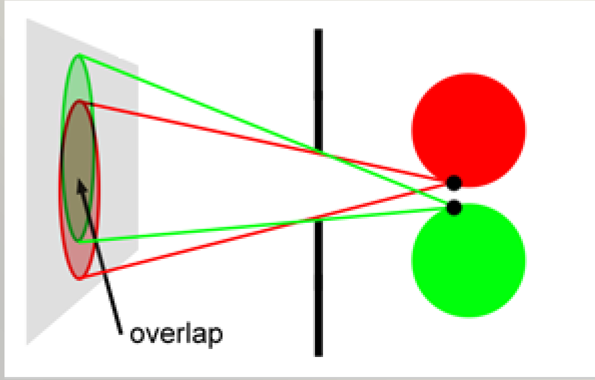
    Eye: Retina
- Things will look blurry without lens



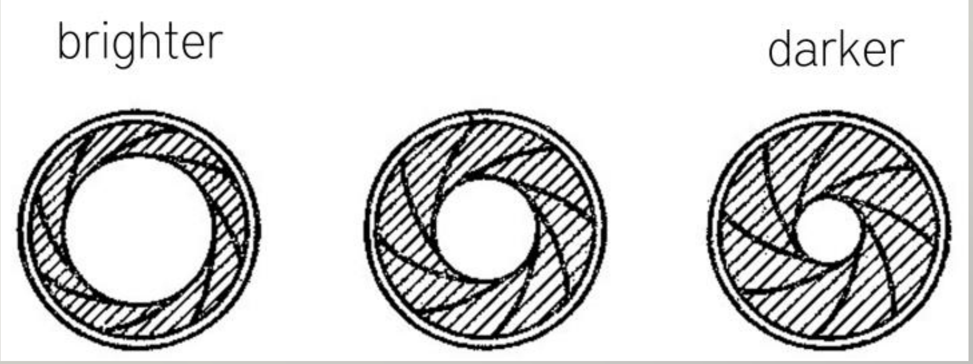
Nidek Co. Ltd.

Camera in Real World

- Since we have a large aperture, it causes light from different objects overlap
  - This overlap causes the blurriness

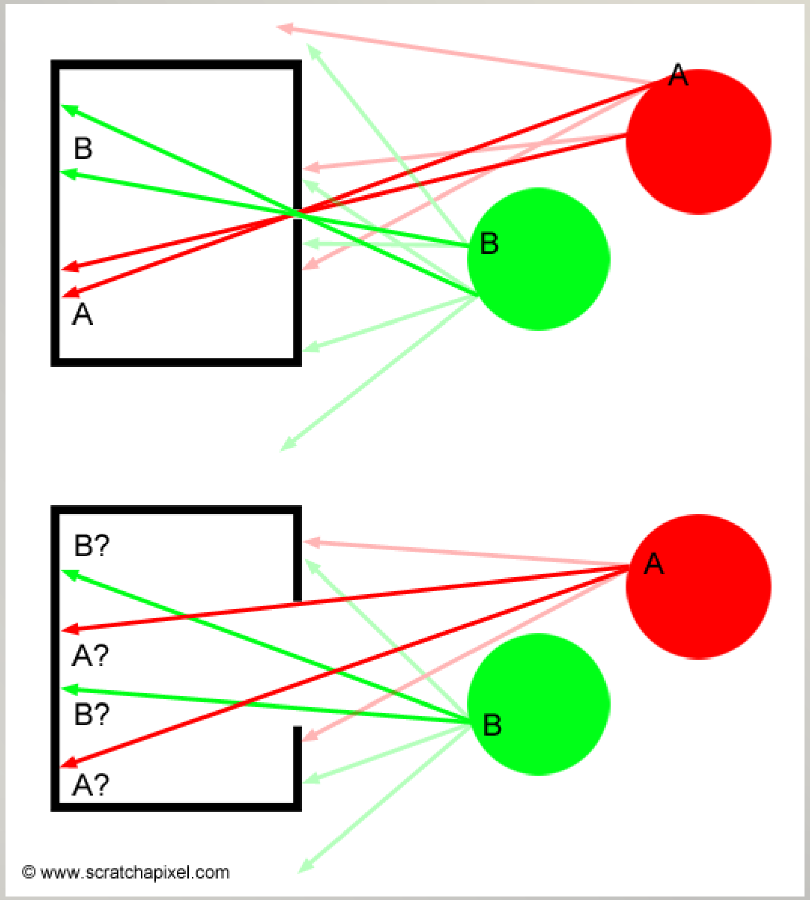


scratchapixel.com



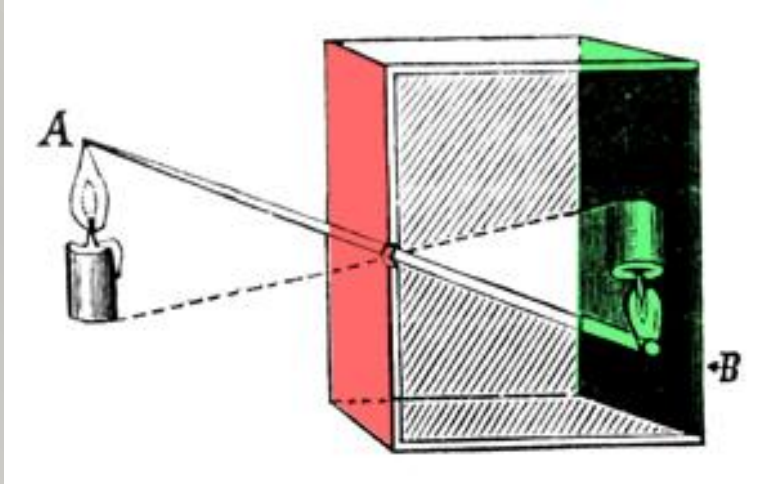
Camera in Real World

- The projection will look less blurry when the aperture size decreases
- No matter far or near, all objects will look sharp and clear

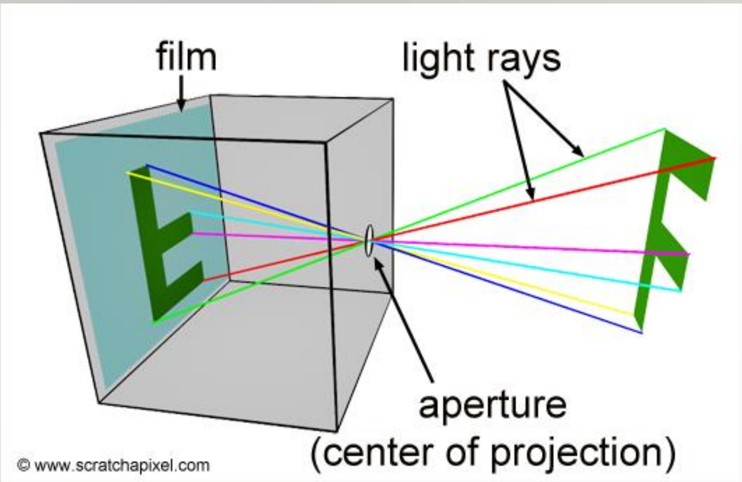


Camera in 3D Graphics

- The light transfer in lens are complicated
- Usually we will use a simple camera model without lens
- Pinhole camera model
  - Assume an infinitely small sized aperture
  - No lens (so, no depth of field)



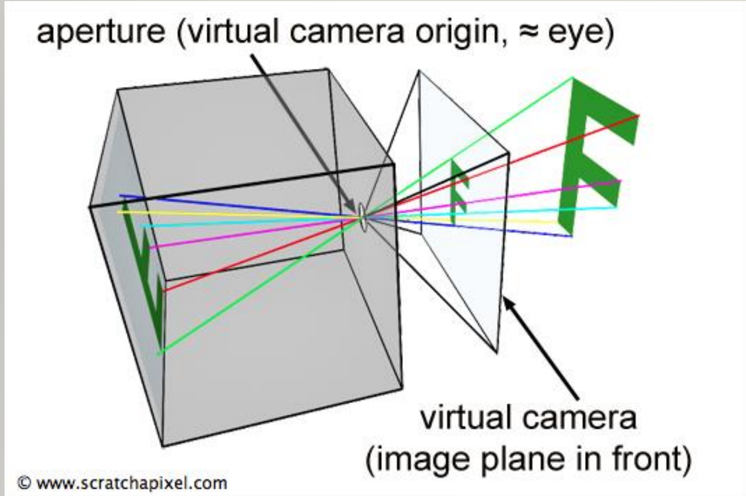
The Boy Scientist. 1925



scratchapixel.com

View Projections

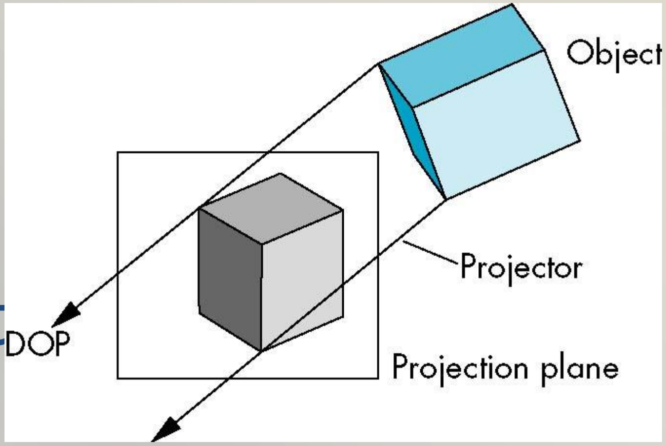
- It is common that we move the projection plane to the front, so that the projection is not inverted in our virtual camera

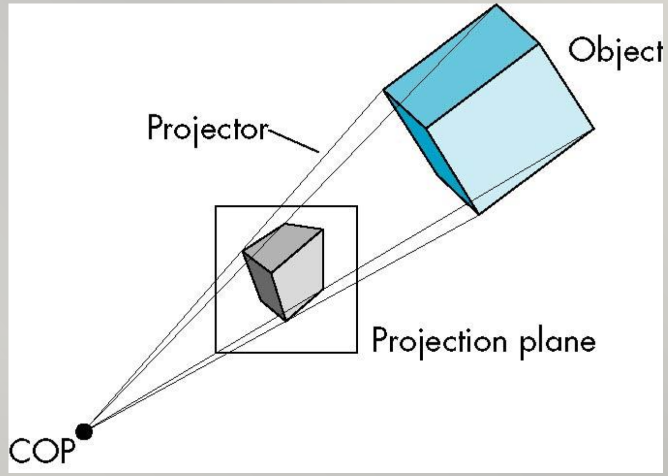


- Also, this simplified the computation of projection in the virtual camera

Perspective vs Parallel

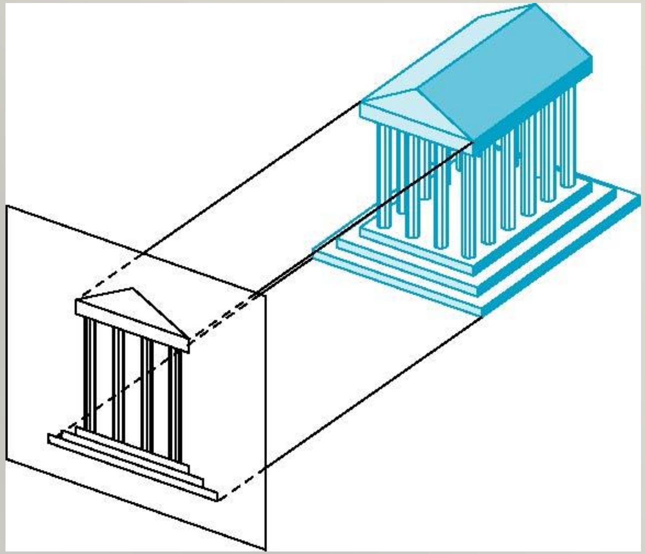
- In CG, 2 common view projections are
  - Parallel/Orthographic projection
  - Perspective projection
- But we can still treat all projections the same way
  - again by matrix multiplication



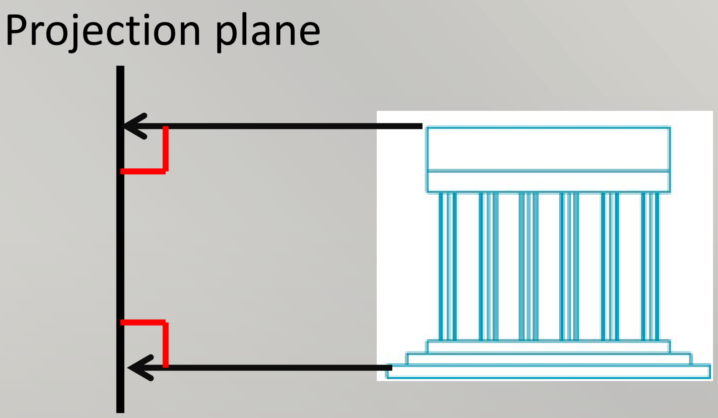


Orthographic Projection

- All projection lines are orthogonal (perpendicular) to the projection plane

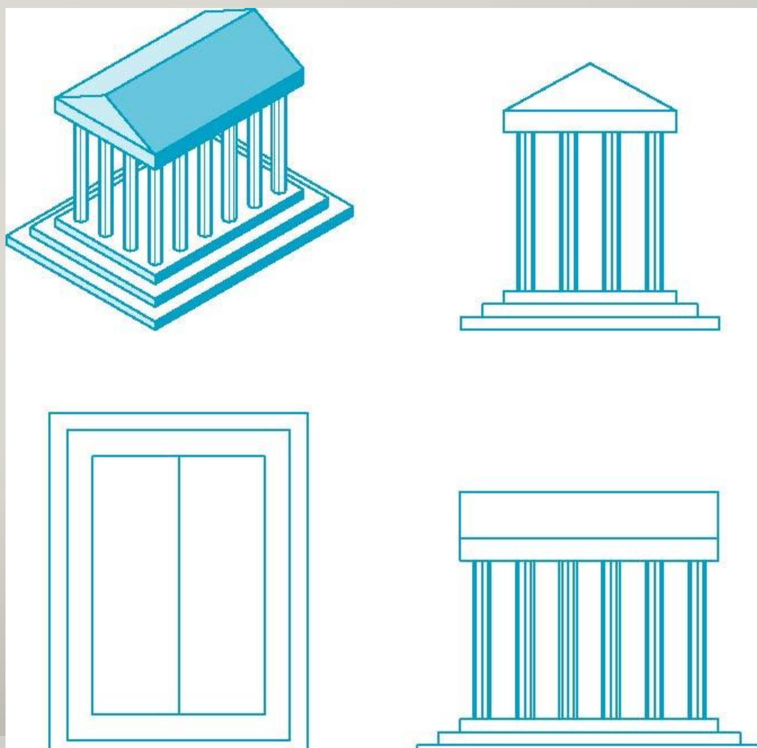


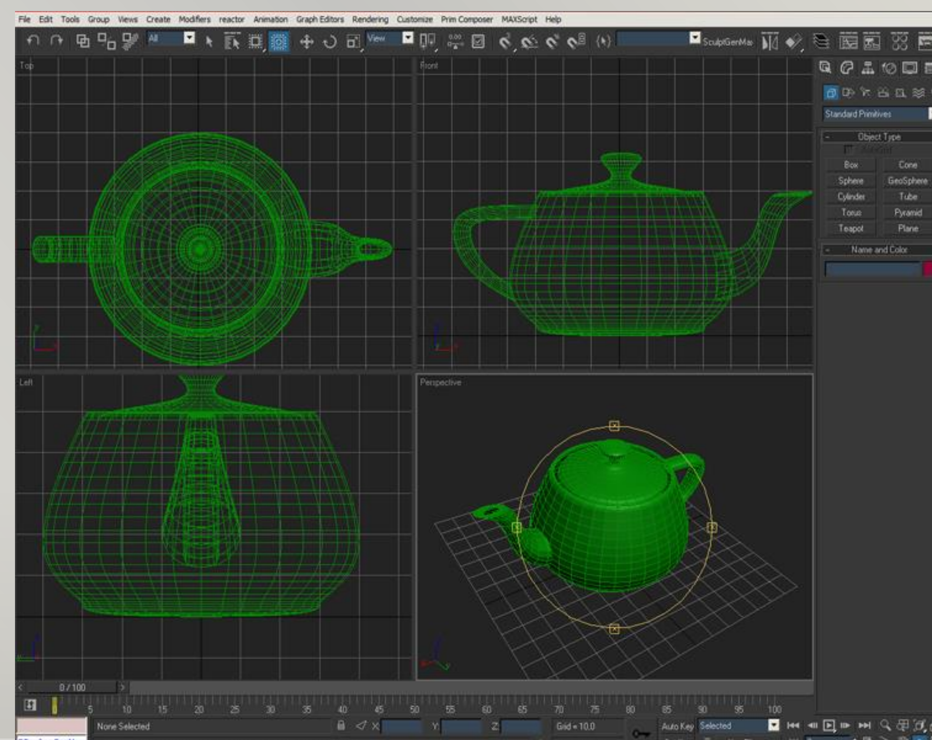
- Preserves both distances and angles
  - Shapes preserved
  - Suitable for measurements



Orthographic Projection

- Commonly used in graphics design and Computer Aided Design (CAD)
  - Frontal, Rear, Top views





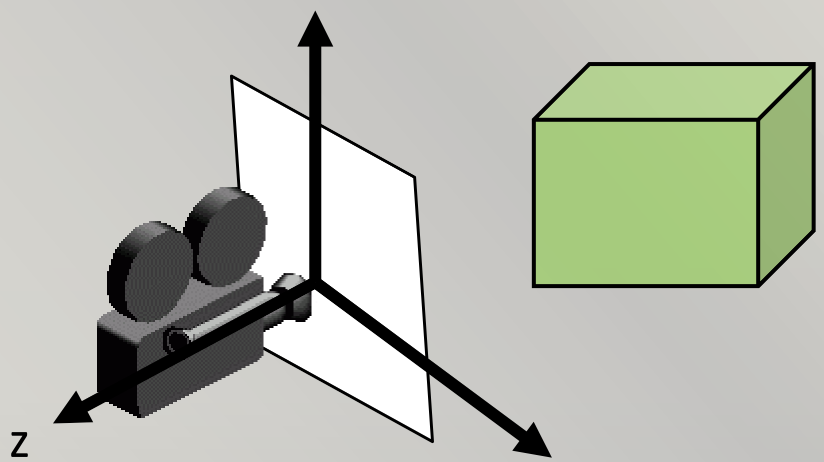
The Orthographic Projection Matrix

- Assume an orthographic projection is onto a projection plane at z = 0, its projection matrix is

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

- So, for any vertex v

$$Pv = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 1 \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ 0 \\ 1 \end{bmatrix}$$



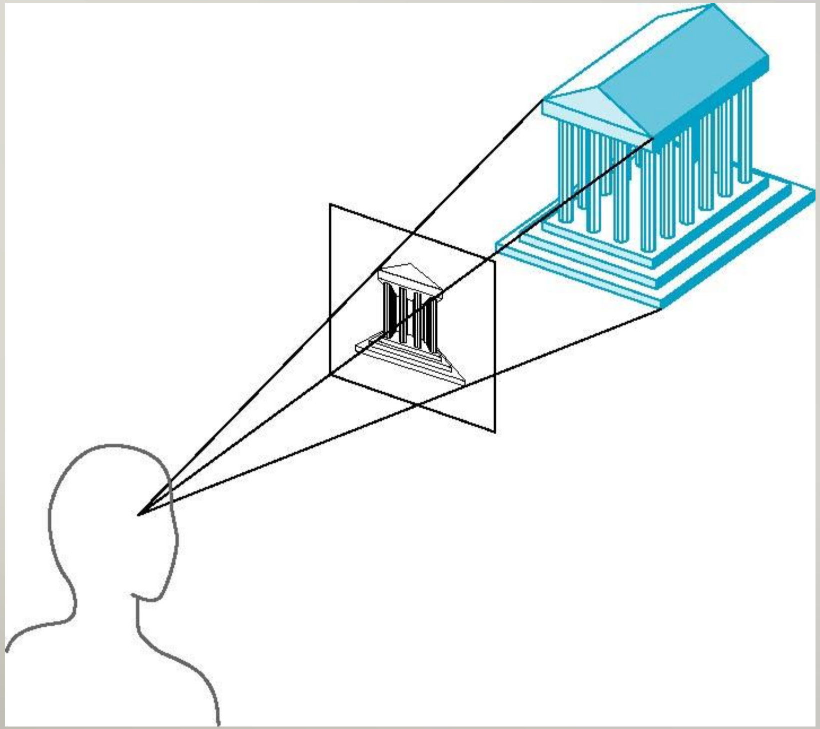
The Orthographic Projection Matrix

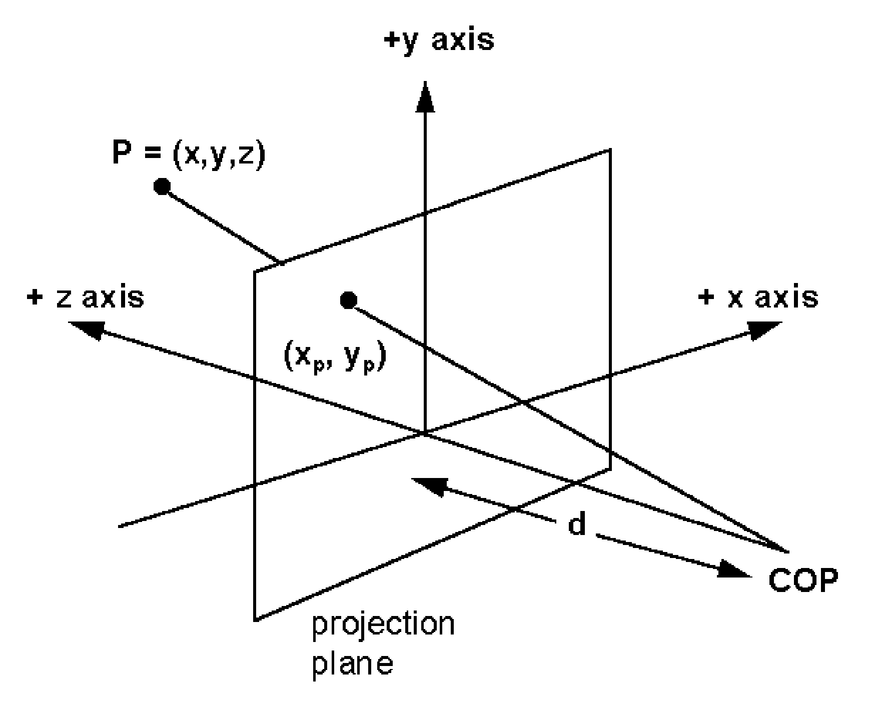
- The effect is simply ignoring the z coordinate
- No change of the x and y coordinates
- Example, v = (3,5,10), the projected vertex will be (3, 5) on the projection plane

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 5 \\ 10 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 0 \\ 1 \end{bmatrix}$$

Perspective Projection

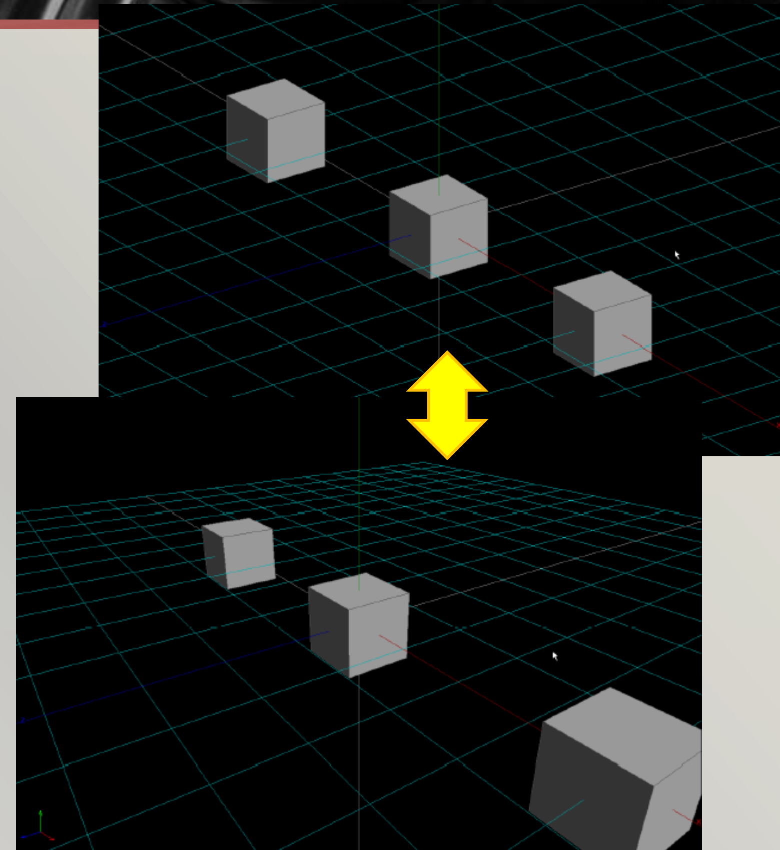
- All projection lines converge to a point : the center of projection (COP)
  - COP is the aperture of camera



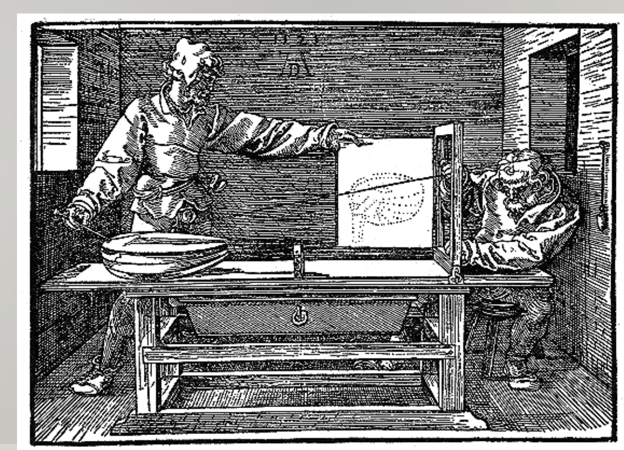


Perspective Projection

- Objects further from viewer are projected smaller than the same sized objects closer to the viewer
  - Looks realistic
  - Feeling of depth
- More difficult to construct by hand than parallel projections (but not more difficult by computer)



rbwhitaker.wikidot.com



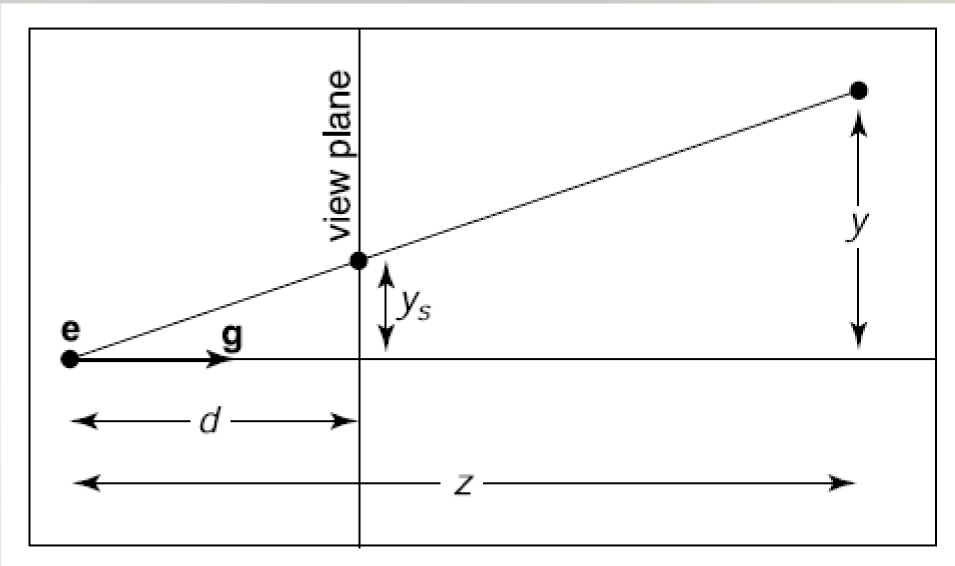
Perspective Projection Matrix

- Let's look at the projection of a vertex (x,y,z) from one side
- By rule of similar triangles, we have ratio:

$$y_s = \frac{d}{z} y$$

This also applies to x direction, i.e.

$$x_s = \frac{d}{z} x$$



Perspective Projection Matrix

- Finally, projected z_s will always be d
- According to the above

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

$$Pv = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

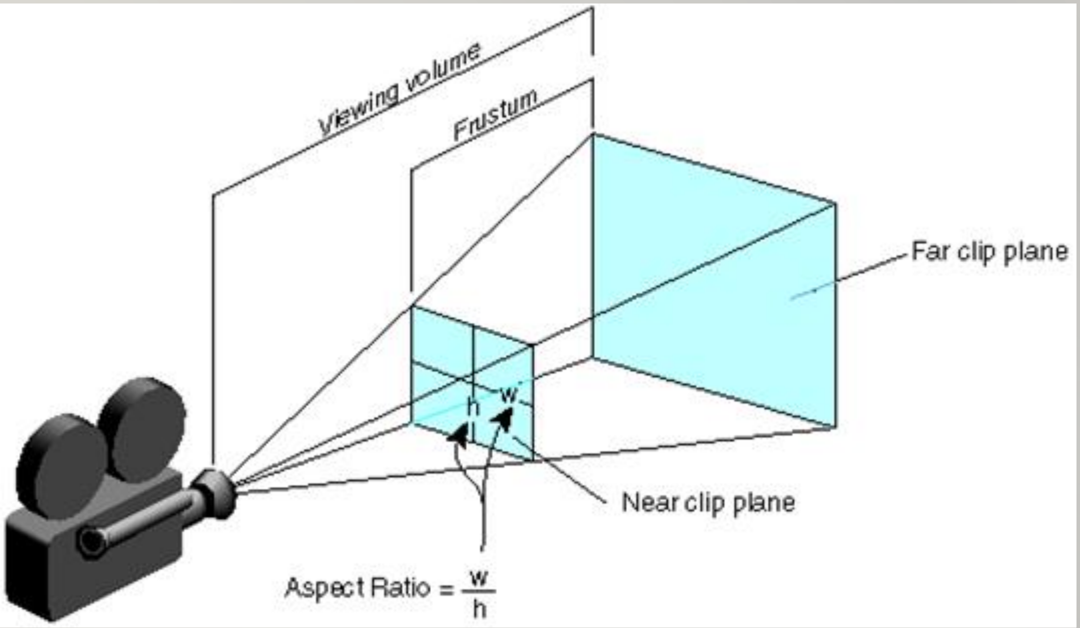
$$= \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

Convert from homogenous coordinates

$$\Rightarrow \quad (\frac{d}{z}x, \frac{d}{z}y, d)$$

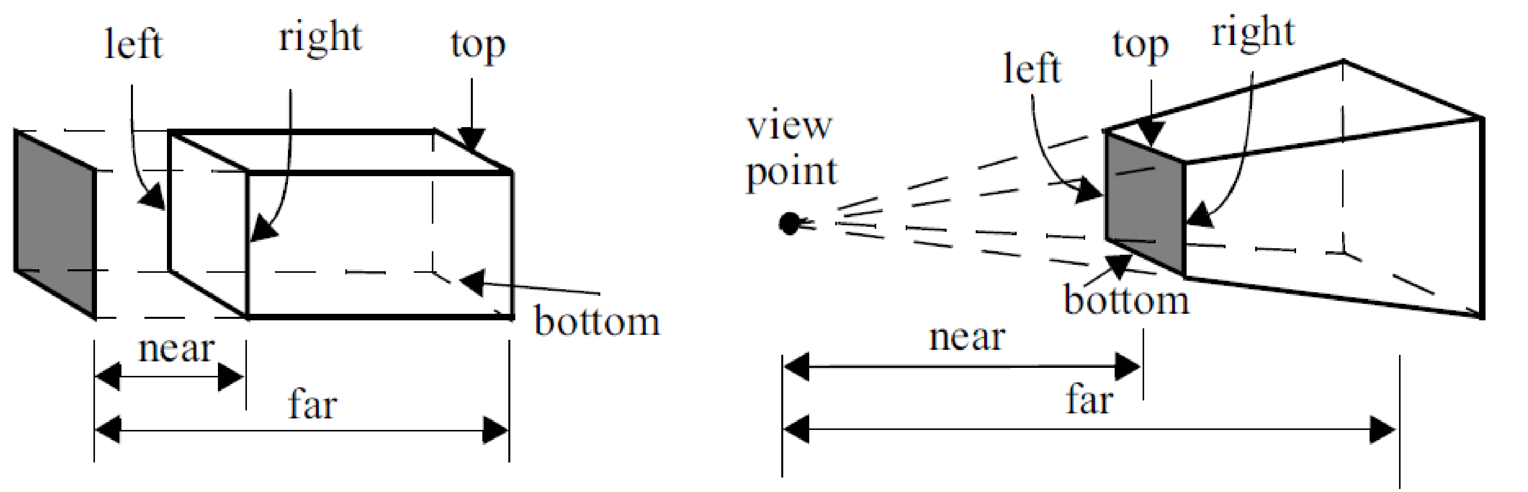
Clipping planes

- To save computational power, virtual camera commonly included clipping planes
  - Only objects within the clip volume (frustum) are included
  - Things far away are not included



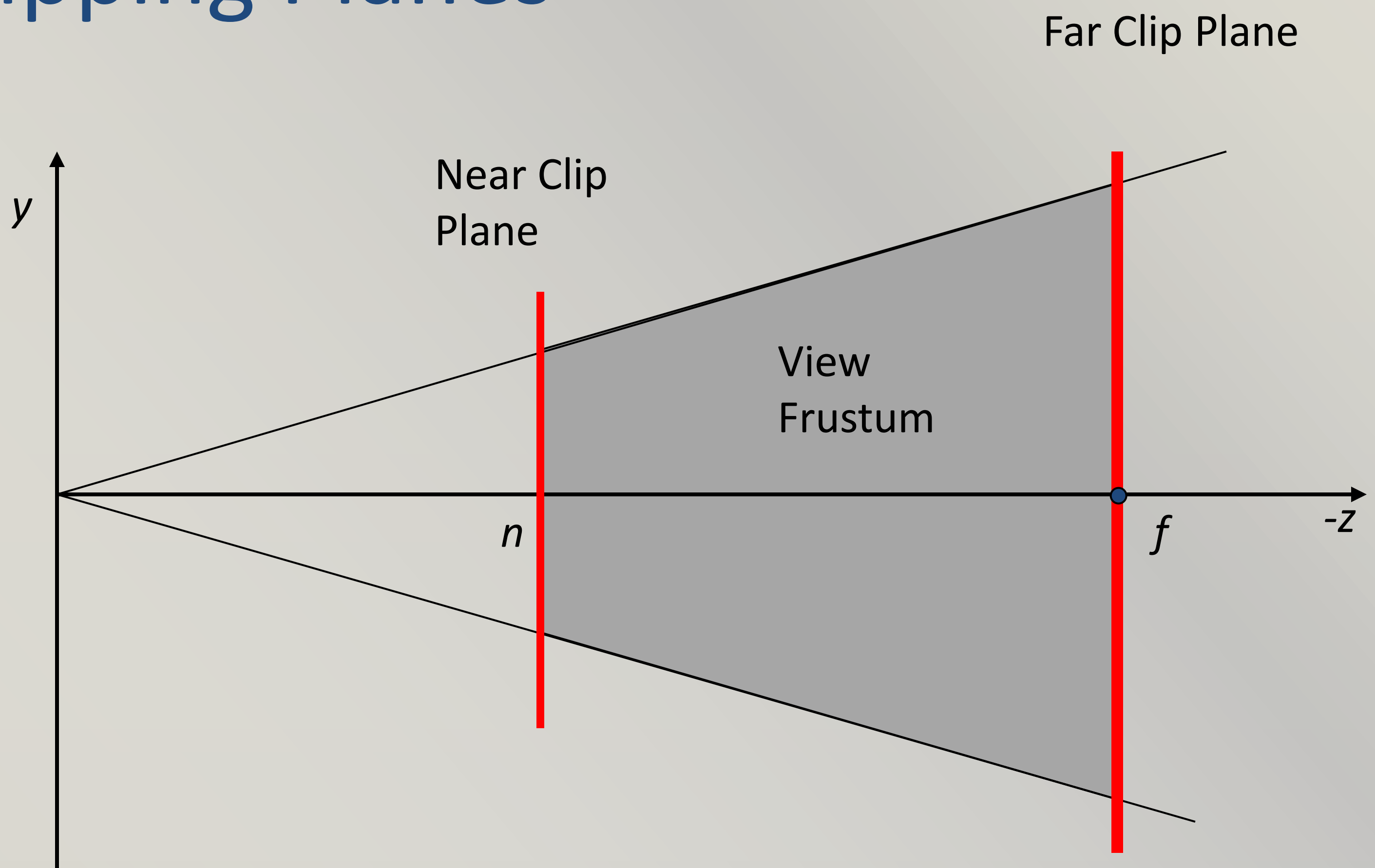
Clipping planes in Orthographic and perspective projections

- In both orthographic and perspective projection
  - Near plane (commonly used as the projection plane)
  - Far plane



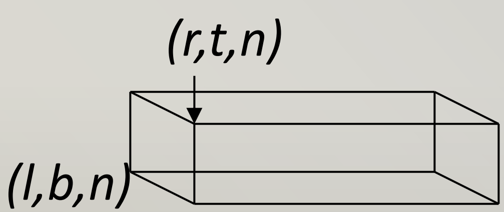
(a) Parallel projection *(b) Perspective projection*

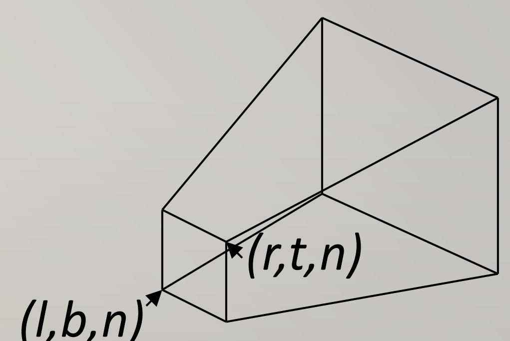
Clipping Planes



Perspective Projection Matrices

- Apart from the near plane position n and far plane position f
- The viewing frustum is also defined by the top, bottom, left and right positions
  - They are corresponding to t,b,l,r in the following figures





Perspective Matrix

$$\mathbf{M}_P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & (n+f)/n & -f \\ 0 & 0 & 1/n & 0 \end{bmatrix} \equiv \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

- This matrix the simple projection matrix, but it does some extra things to z to map the depth properly
  - Other projection matrix configuration will map z non-linearly
- We can multiply a homogenous matrix by any number without changing the final point, so the two matrices above have the same effect

Perspective Matrix

- This perspective matrix will map the z-coordinate nicely and suitable to use in hidden surface removal (more details in later lectures)
  - It will preserve the relative order of z after projection

    i.e. if z1 > z2 then projected z1 > projected z2

$$\mathbf{P}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z\frac{n+f}{n} - f \\ \frac{z}{n} \end{bmatrix} \sim \begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ n + f - \frac{fn}{z} \\ 1 \end{bmatrix}$$

Example

- A vertex (1,1,-1) in camera space are going to project on the screen space which its near plane n at z = -0.5, far plane f at z=-10.0

$$Pv = \begin{bmatrix} -0.5 & 0 & 0 & 0 \\ 0 & -0.5 & 0 & 0 \\ 0 & 0 & -0.5-10 & -0.5\times 10 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.5 \\ -0.5 \\ 10.5-5 \\ -1 \end{bmatrix}$$

The projected point = (0.5,0.5)

OpenGL Perspective Projection

- For OpenGL you give the distance to the near and far clipping planes
- The total perspective projection matrix resulting from a `glFrustum` call is:

$$\mathbf{M}_{OpenGL} = \begin{bmatrix} \frac{2|n|}{(r-l)} & 0 & \frac{(r+l)}{(r-l)} & 0 \\ 0 & \frac{2|n|}{(t-b)} & \frac{(t+b)}{(t-b)} & 0 \\ 0 & 0 & \frac{(|n|+|f|)}{(|n|-|f|)} & \frac{2|f||n|}{(|n|-|f|)} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Field of View (FOV)

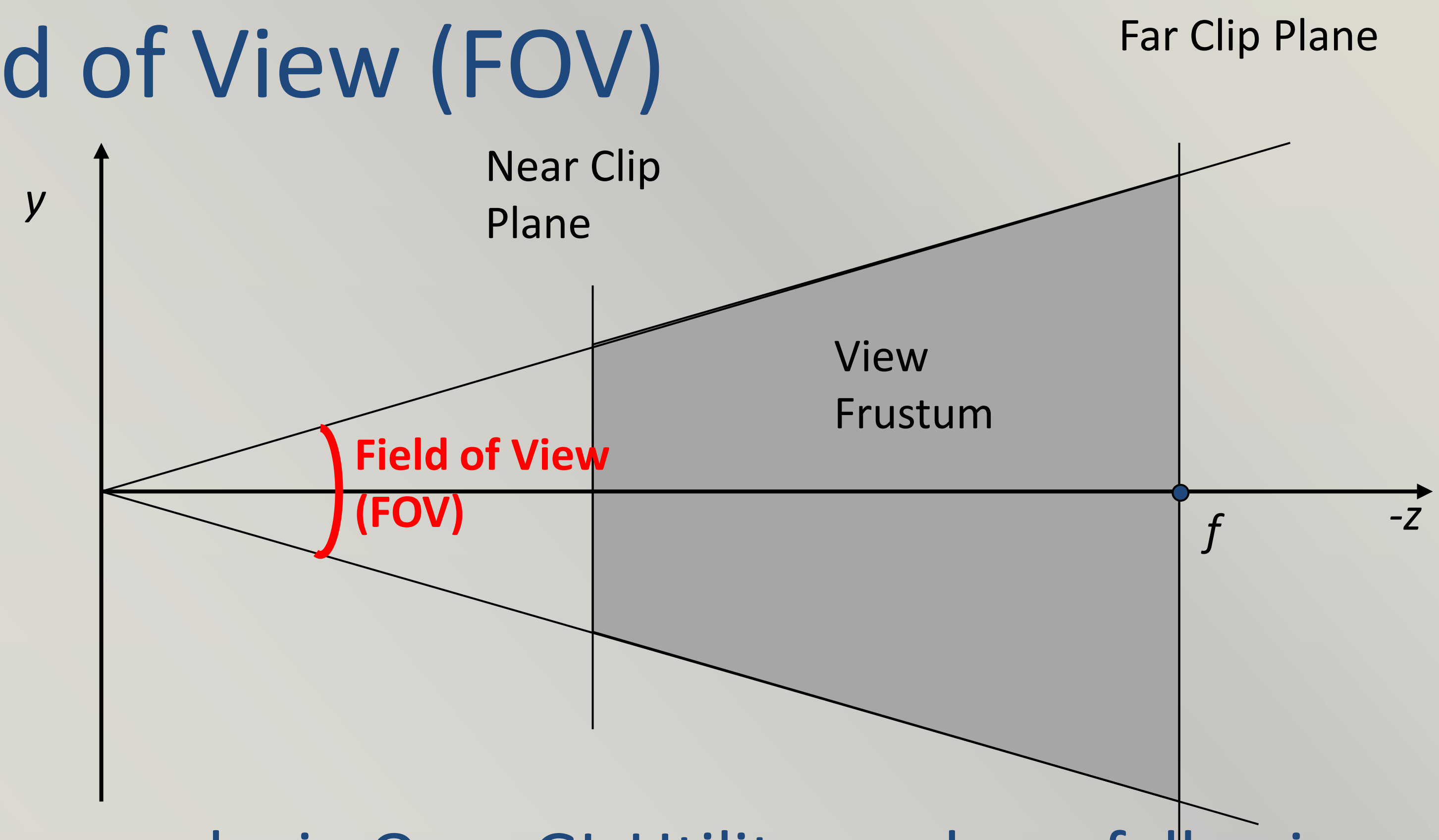
- Instead of representing the projection plane by its 4 corner's offset to center, we can further simplify by assuming it to be symmetric about the center

$$l = -r \,,\ \ b = -t$$

- Assume pixels are square, the field of view (FOV) will be the last parameter left for us to define the camera configuration

- It is a specific feature in perspective projection

Field of View (FOV)

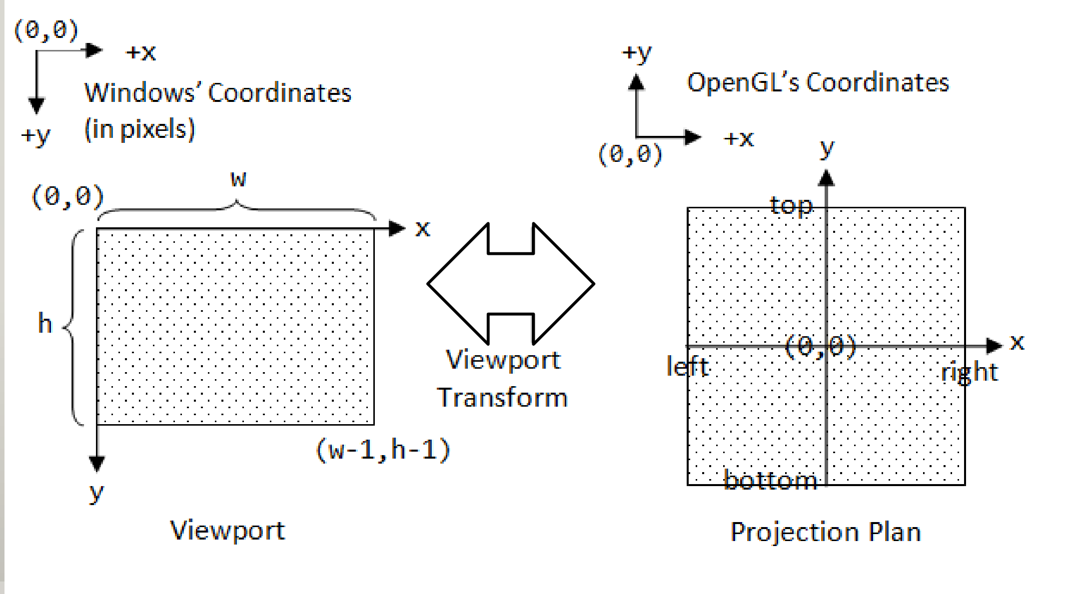


- For example, in OpenGL Utility, we have following function to define the projection:

```
void gluPerspective(double fovy, double aspect,
double zNear, double zFar);
```

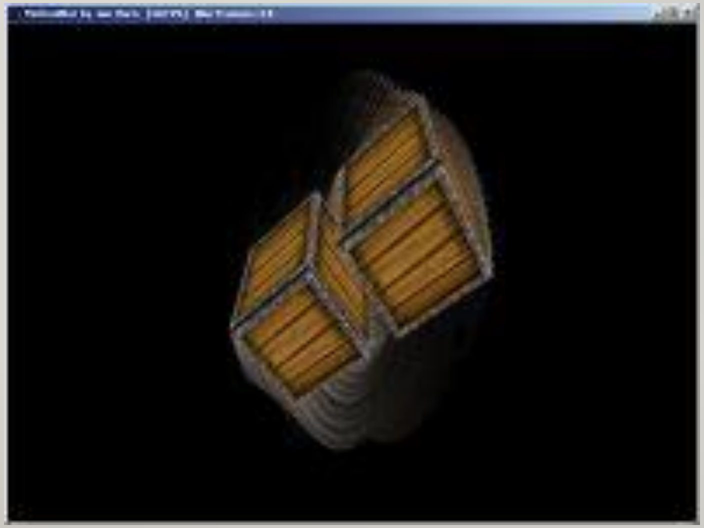
ViewPort

- In CG, it refers to the region of screen where shows the rendered results
  - So it is 2D and no longer in 3D
  - A fundamental difference between viewport and projection plane



Advanced Camera Effects

- Lens flare
  - Simulate the effect with the use of the flare texture
- Motion blur
  - Accumulate the rendering result of several frames



Motion Blur - OpenGL projects in Delphi

Fast OpenGL-rendering of Lens Flares

Summary

- Transformation of coordinate systems
  - Object Space, World Space, Camera Space, Screen Space
- Pin-hole camera is the most commonly used camera model in CG, due to its simplified computation of 3D to 2D projection
  - Orthographic and Perspective
  - Projection Matrix
- Clipping plane are usually included to define a finite viewing frustum for rendering