AI5026
Computer Graphics

Hochschule Fulda
University of Applied Sciences

Transformations – Model Space

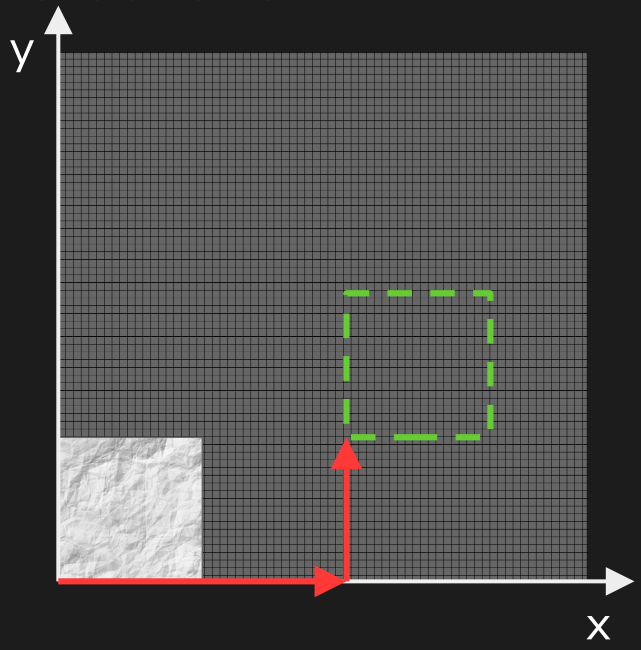
- Models contain local vertices
  - Center of origin is often at model center
- To “move” an object, transform **all** vertices
  - From **object space** to **world space**

Transformations – World Space

- Euclidean coordinate system
- World space is arbitrary
  - Sizes defined
  - Center of origin defined

Transformations – Translation

- Movement in nD space
  - On every axis
- Translate(*x*, *y*)
- Example in 2D:
  - Translate(2, 1)
  - Bottom left as center of origin
- Position + Translation
  - Element-wise addition



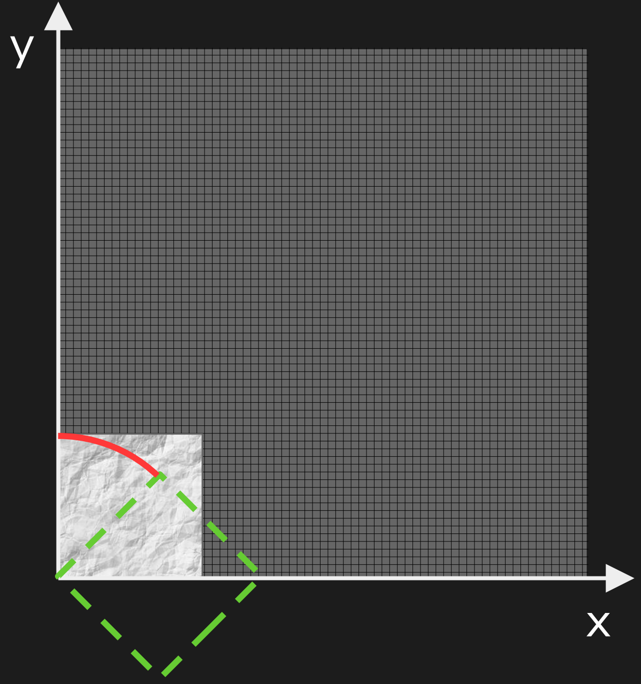
Transformations – Rotation

- Rotation in nD space
  - Around single axis
- Rotate_*axis*(*degrees*)
- Example in 2D:
  - Rotate_Z(45)
  - Bottom left as center of origin

2D around Z:

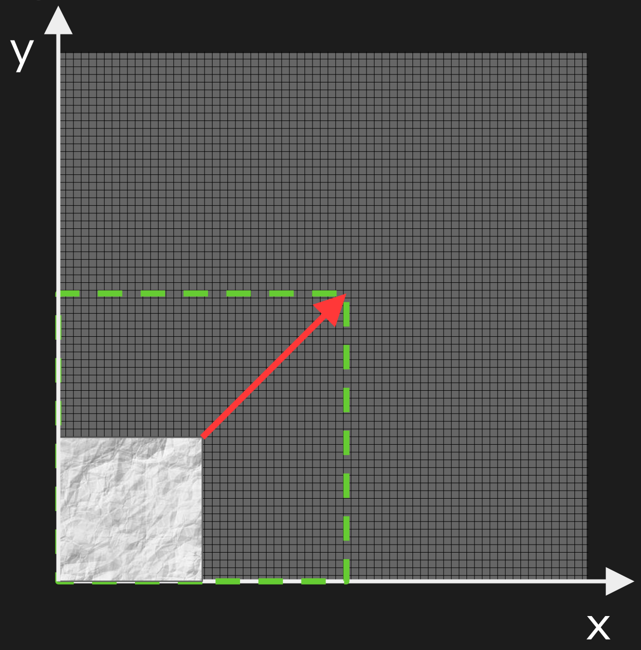
X = X * cos(*degr*) – Y * sin(*degr*)

Y = X * sin(*degr*) + Y * cos(*degr*)



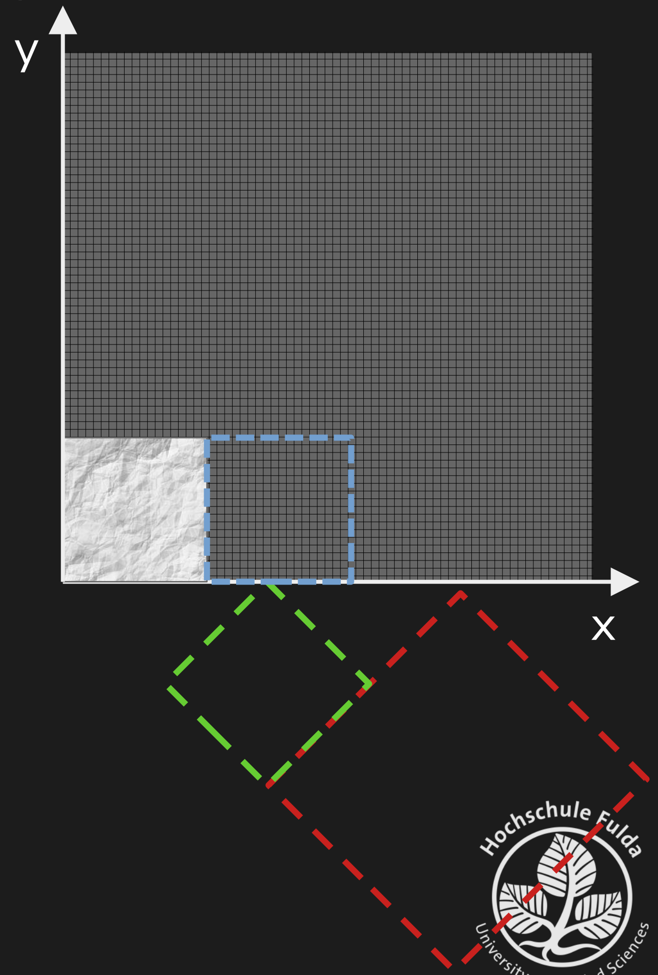
Transformations – Scale

- Scaling in nD space
  - Often uniform
- Scale(*x*, *y*)
- Example in 2D:
  - Scale(2, 2)
  - Bottom left as center of origin
- Position * Scale
  - Element-wise multiplication



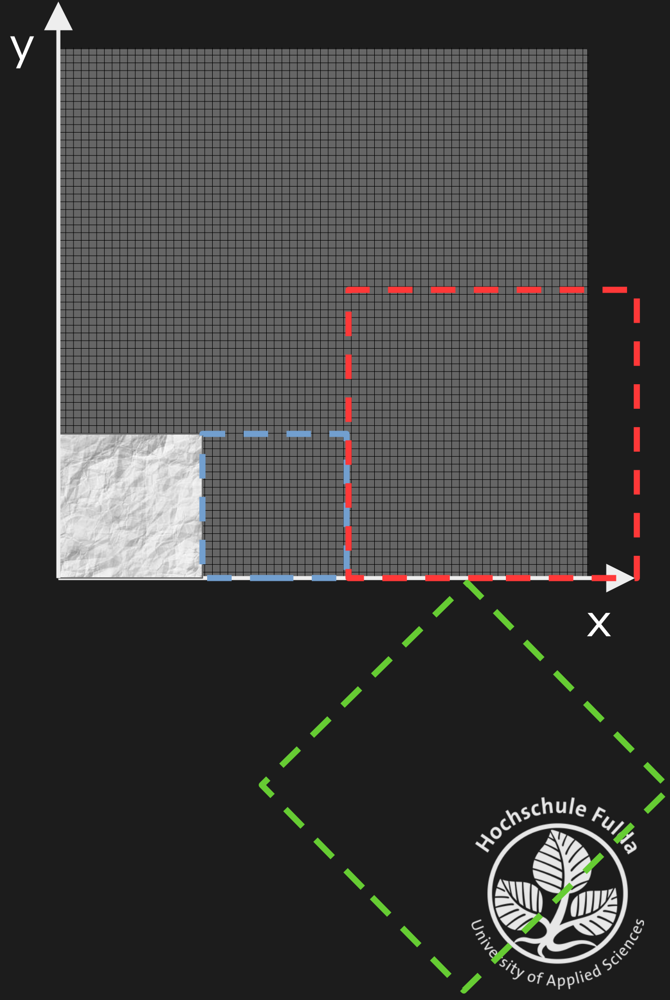
Transformations – Order

- Order of applied transformations important
- Example (approximation):
  - Translation → Rotation → Scale



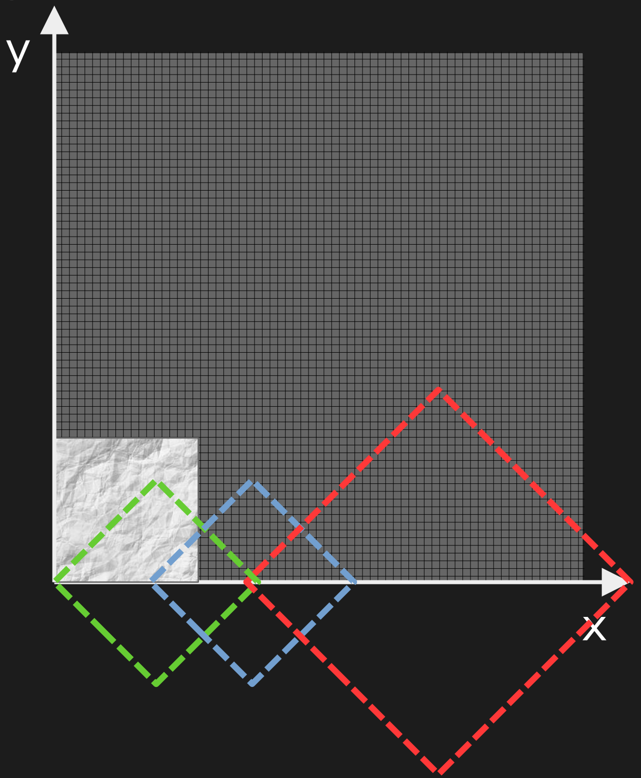
Transformations – Order

- Order of applied transformations important
- Example (approximation):
  - Translation → Scale → Rotation



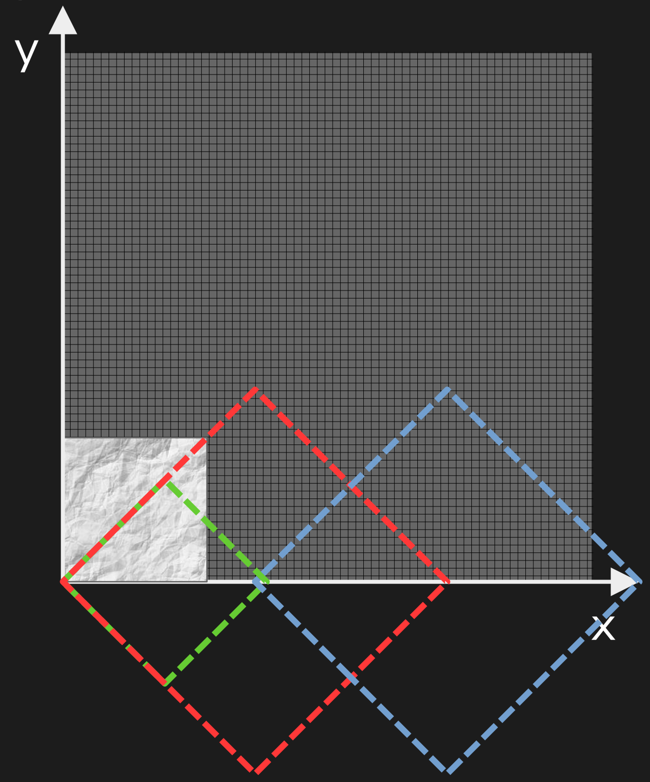
Transformations – Order

- Order of applied transformations important
- Example (approximation):
  - Rotation → Translation → Scale



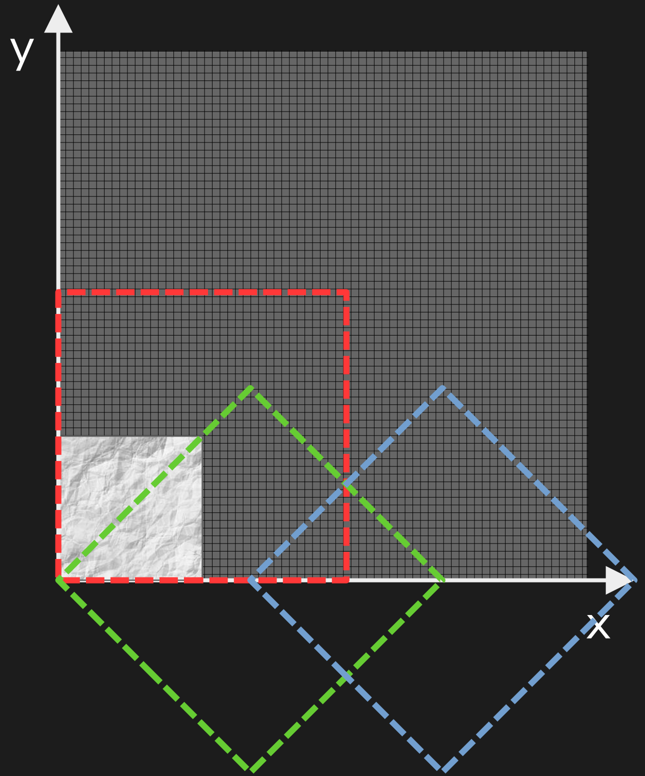
Transformations – Order

- Order of applied transformations important
- Example (approximation):
  - Rotation → Scale → Translation



Transformations – Order

- Order of applied transformations important
- Example (approximation):
  - Scale → Rotation → Translation



Transformations – Matrices

- Transforms are one of the major operations in rendering
  - Needs to be performant
- Solution: Matrices
  - Every transform can be represented by matrix
  - Matrix can be multiplied to combine operations
    - Single matrix for [translation, rotation, scale]
    - Retains order of operations

$$M_{SRT} = M * S * R * T$$

$$=$$

$$M_S = M * S$$
$$M_{SR} = M_S * R$$
$$M_{SRT} = M_{SR} * T$$

Transformations – Matrices

As a refresher on matrices:

row major

$$[x' \; y' \; z'] = [x \; y \; z] \times \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

$$x' = x \times a + y \times d + z \times g$$
$$x' = x \times b + y \times e + z \times h$$
$$x' = x \times c + y \times f + z \times i$$

column major

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$x' = a \times x + b \times y + c \times z$$
$$x' = d \times x + e \times y + f \times z$$
$$x' = g \times x + h \times y + i \times z$$

In our case

- **No** mismatched matrix sizes (always NxN * NxN or Nx1)
- Free to use either major layout, can always transpose()

Transformations – Scale

- Row and column major are the same
- Scale matrix S
  - Parameters a, b, c
  - Dimensions x, y, z respectively

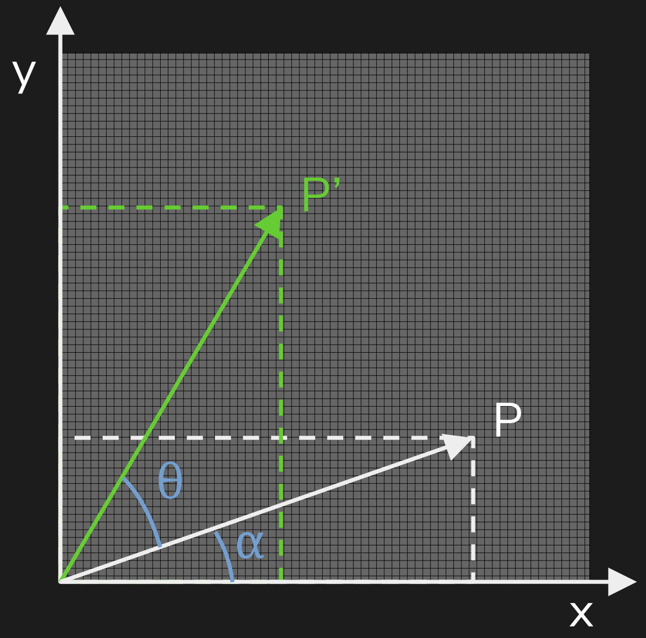
$$S = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix}$$

Transformations – Rotation

- Simple derivation for 2D as example
  - Rotating point P by angle θ

$$r = |P|$$
$$P_x = r \times \cos(\alpha)$$
$$P_y = r \times \sin(\alpha)$$
$$P'_x = r \times \cos(\alpha + \theta)$$
$$P'_y = r \times \sin(\alpha + \theta)$$

$$P'_x = P_x \times \cos(\theta) - P_y \times \sin(\theta)$$
$$P'_y = P_x \times \cos(\theta) + P_y \times \sin(\theta)$$



Transformations – Rotation

- Row Major
- Need to rotate around every axis individually
  - Combine as matrices
- Rotate by angle θ around given axis

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Transformations – Translation

- Row Major
- 3D Translation not representable as 3x3 matrix
  - Extend dimensions by 1
  - Homogeneous coordinates
    - Every vertex position should be [x, y, z, 1]
- Translation matrix T to translate point by [x, y, z]

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x & y & z & 1 \end{bmatrix}$$

Transformations - Overview

Rotation

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scale

$$S = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x & y & z & 1 \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

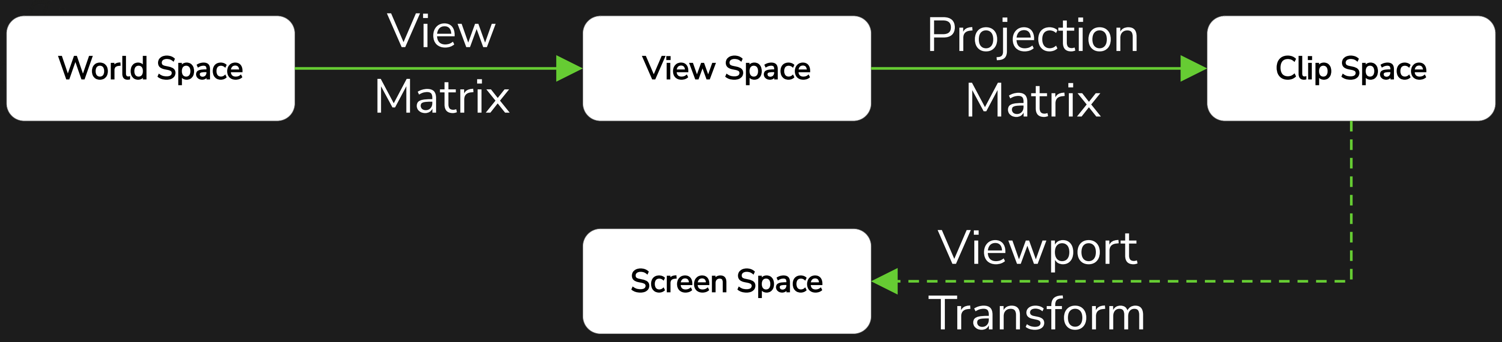
Camera Space

World space: World origin as center

View space: Camera as the world center

Clip space: 3D → 2D projection

Screen space: Screen coordinates (e.g. 1920 x 1080)



View

World Space $\xrightarrow{\text{View Matrix}}$ View Space

- Camera as center of world
  - Camera has its own transform
    - Rotation, translation
  - Apply inverse camera translation and rotation to every vertex
    - Invert [x, y, z] or θ
- **View matrix** V = inverse(T) * inverse(R)

Example:

Translation

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x & y & z & 1 \end{bmatrix}$$

Camera Translation

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x & -y & -z & 1 \end{bmatrix}$$

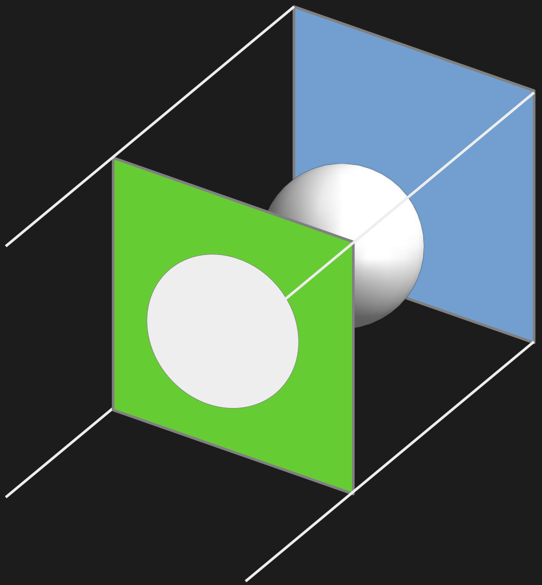
Projection

- Transform to clip space through projection
  - Normalized coordinates { -1.0f, 1.0f }
- Projection of objects onto **near plane**
  - Near plane can be viewed as the **screen**

https://learnopengl.com/Getting-started/Coordinate-Systems

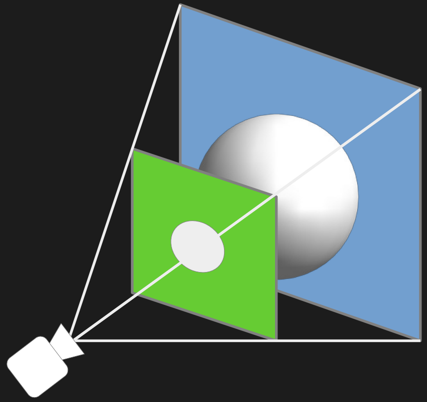
Projection – Orthographic

- Render objects in view frustrum
  - Between near and far plane
  - Near plane as screen
  - No perspective
- Useful for modelling software
  - We won’t use this one



Projection – Perspective

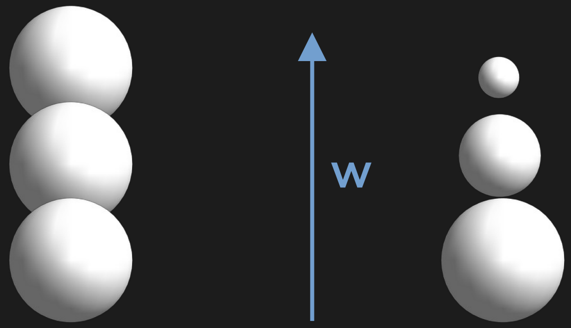
- Most common projection
  - 3D perspective modelling reality
- Near and far planes
  - Near plane smaller
  - Near plane as screen
  - View frustrum volume



https://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/opengl-perspective-projection-matrix.html

Projection

- Objects are smaller at a distance
- Projection to clip space with **projection matrix**
  - Clip space is normalized frustrum
  - Clip space coordinate w contains depth



Orthographic Perspective

https://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/opengl-perspective-projection-matrix.html

Projection

View Space $\xrightarrow{\text{Projection Matrix}}$ Clip Space

Perspective projection matrices can differ

- One example with horizontal FOV

Field of view as *fov* and near/far plane distances as *n* and *f* respectively

$$S = \frac{1}{\tan\left(\frac{fov}{2} * \frac{\pi}{180}\right)}$$

$$T = \begin{bmatrix} S & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & -\left(\frac{f}{f-n}\right) & -1 \\ 0 & 0 & -\left(\frac{f*n}{f-n}\right) & 0 \end{bmatrix}$$

https://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/opengl-perspective-projection-matrix.html