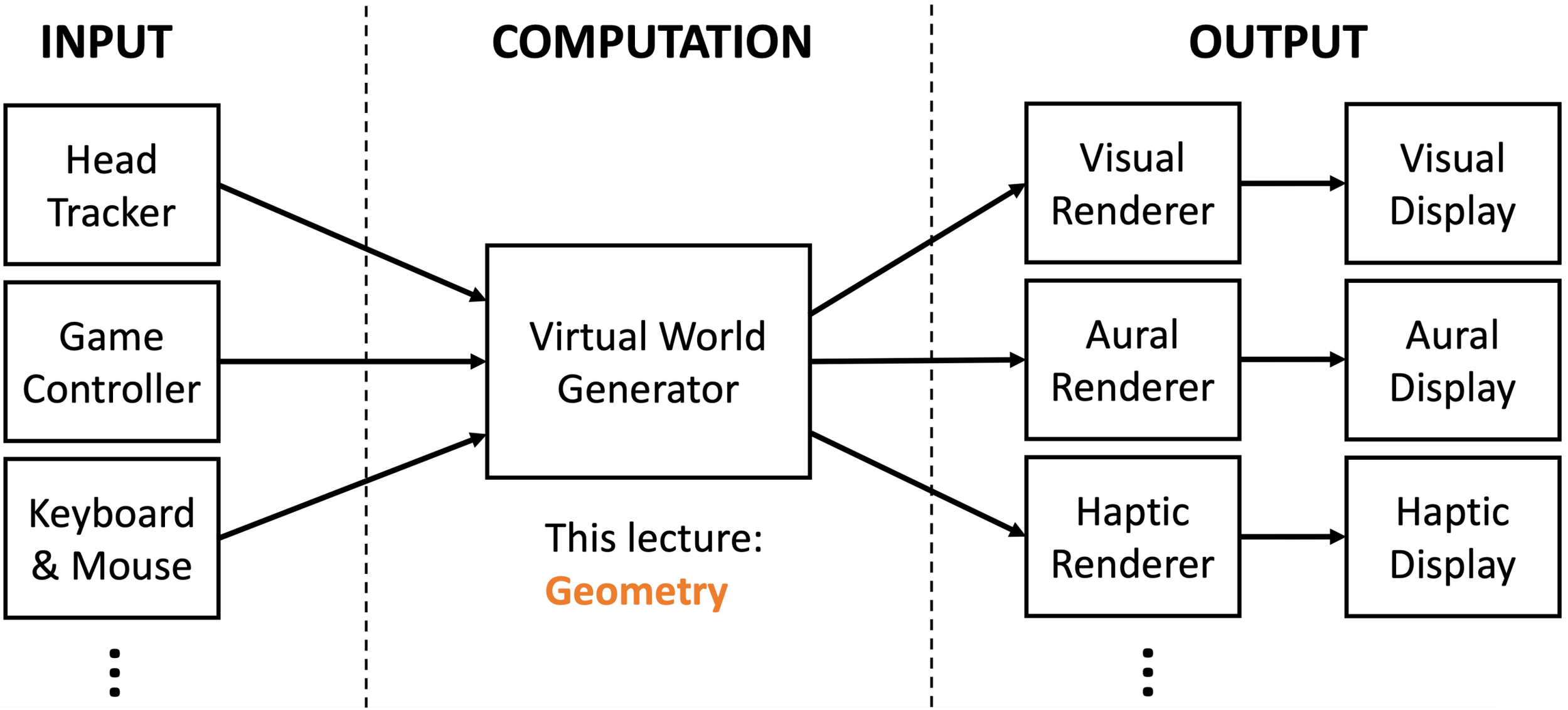
Geometry of Virtual Worlds
Skip 3.3 & 3.5

Review of VR Systems

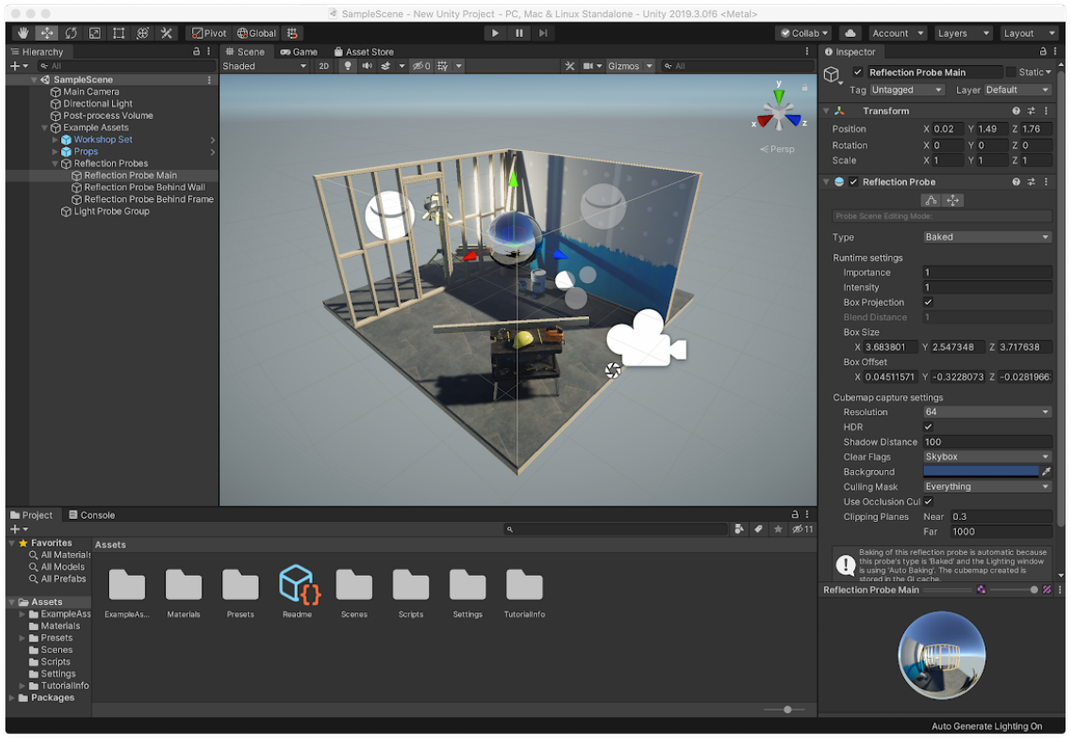


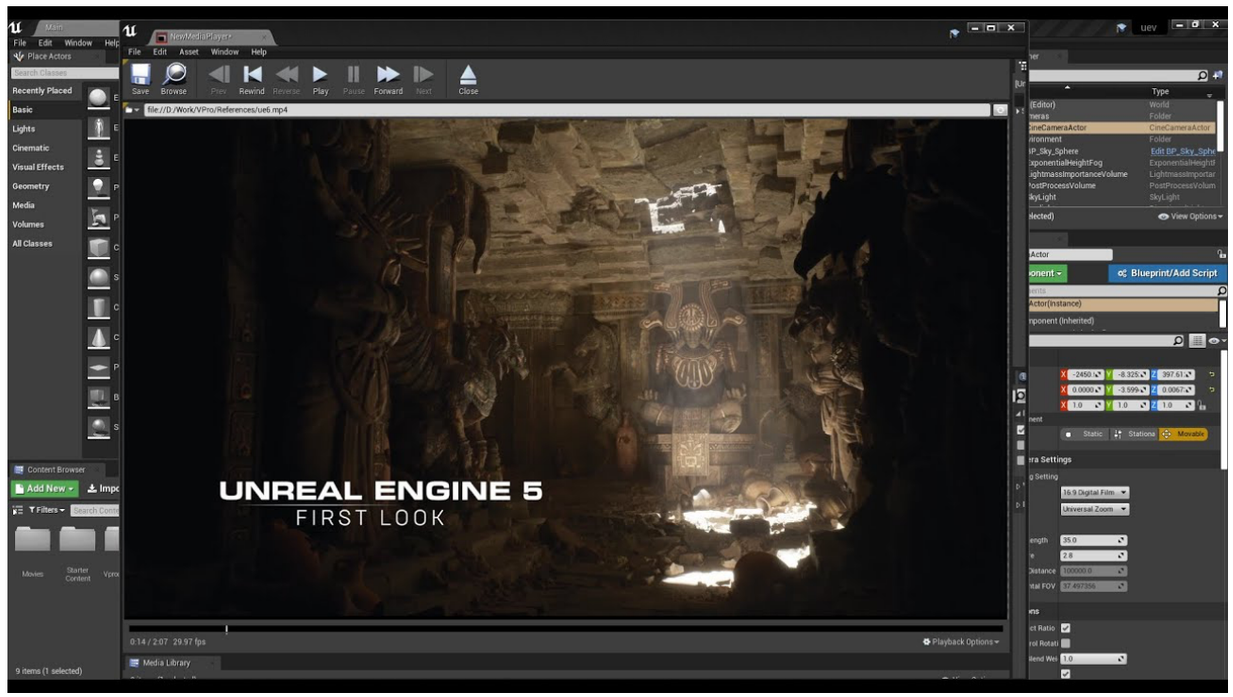
2D Virtual World

3D Virtual World

3D Virtual World, first person

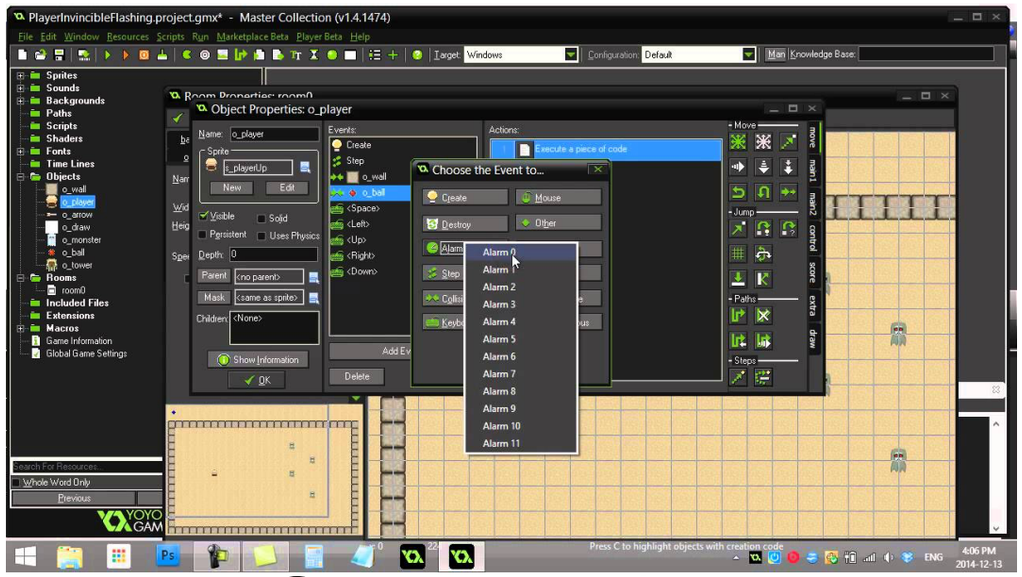
Video Games





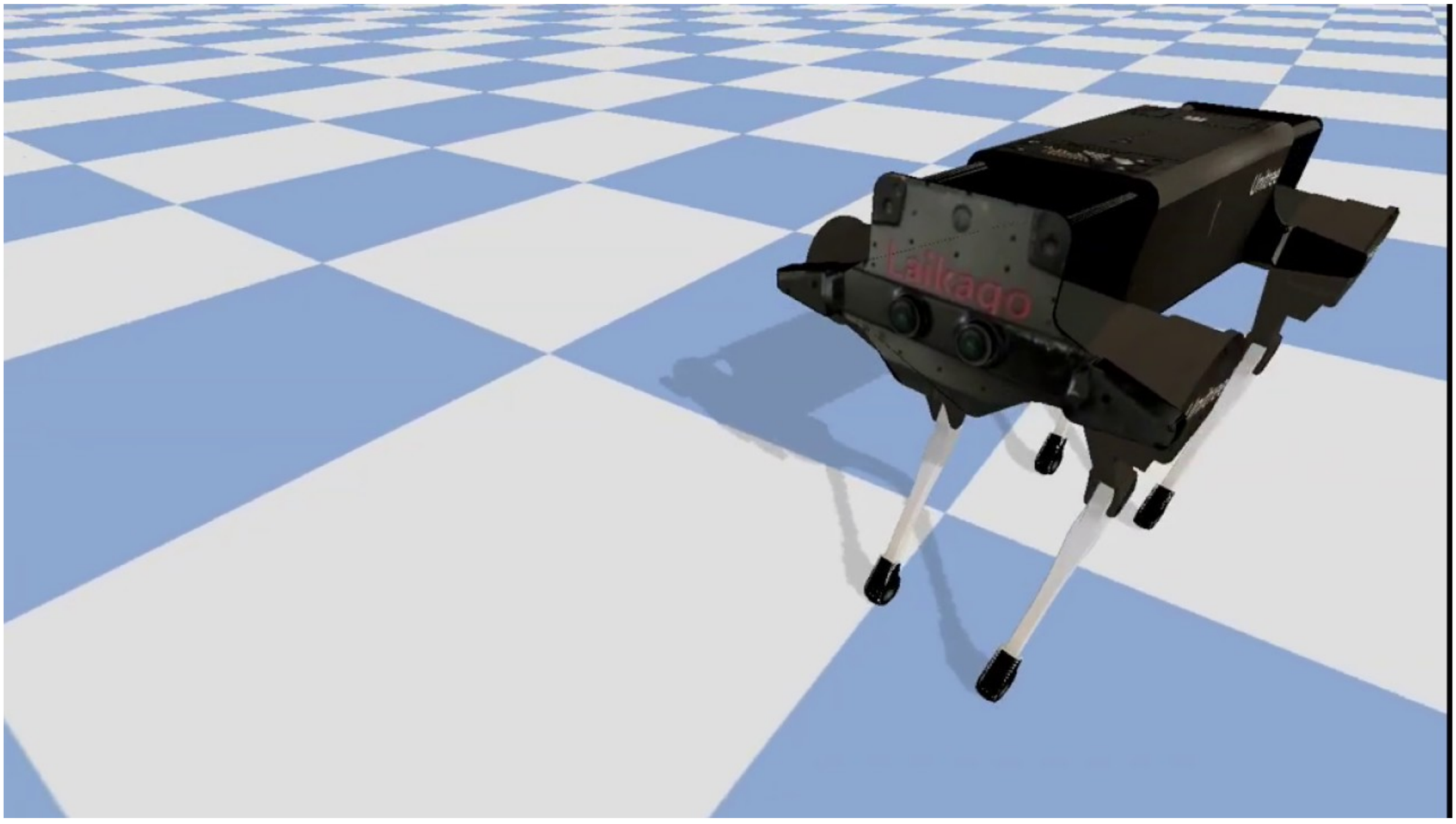
Unity

Unreal



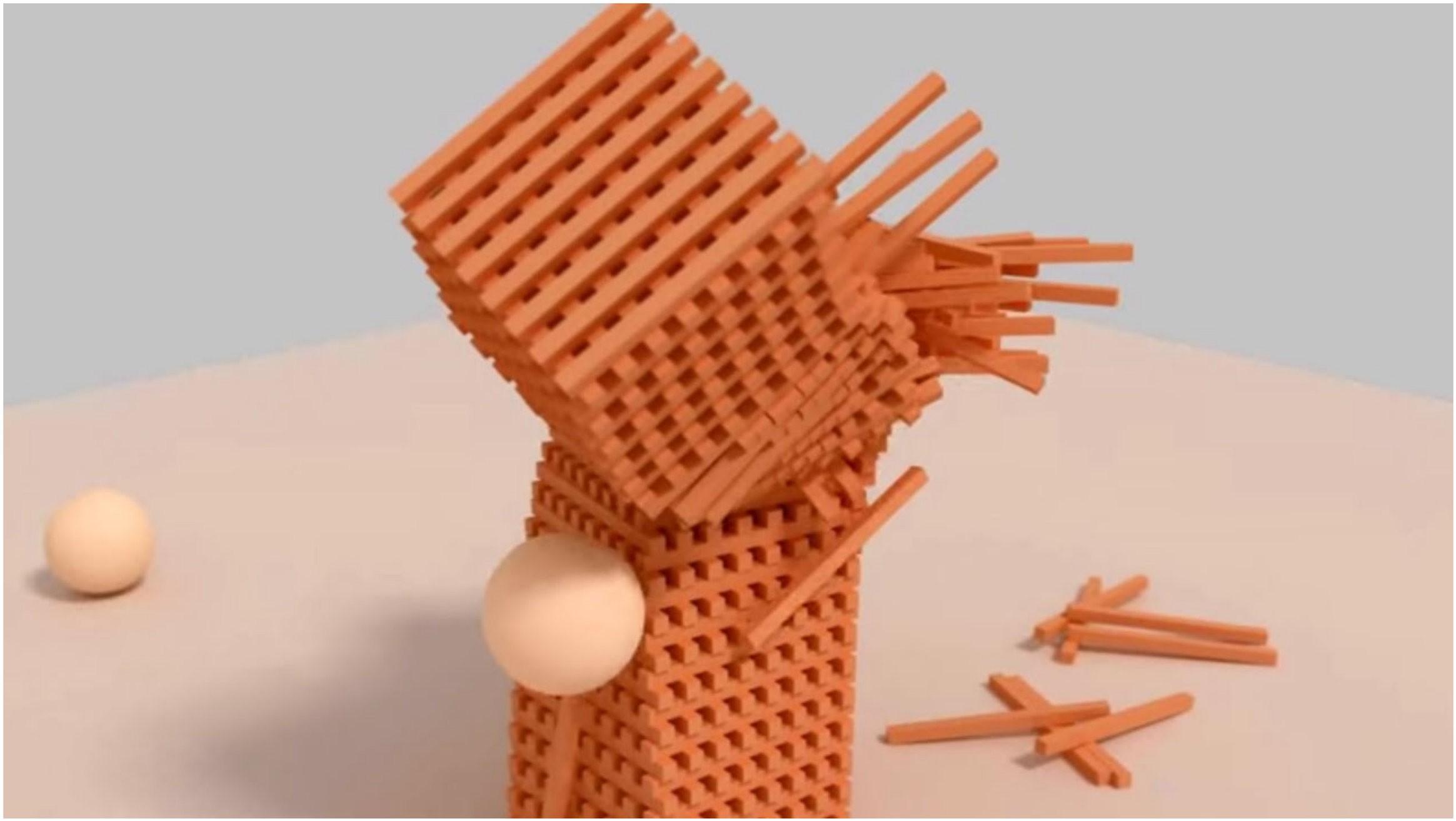
Gamemaker

How to build the VE?



Pybullet

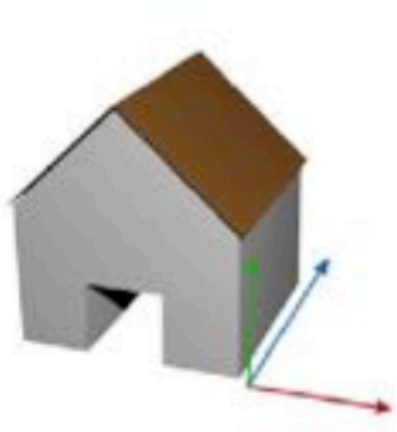
Physics engines



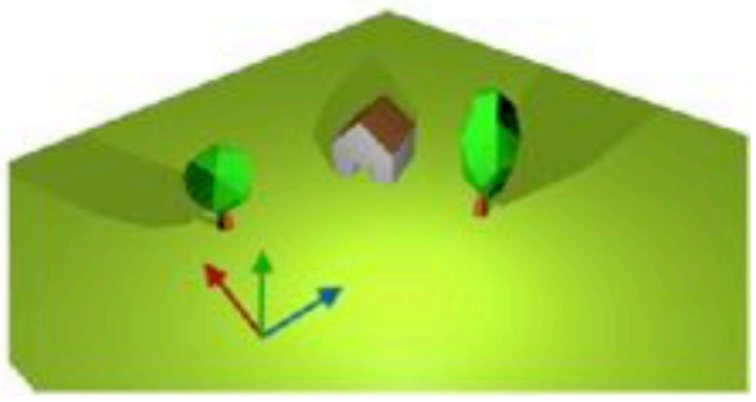
Blender

Physics engines

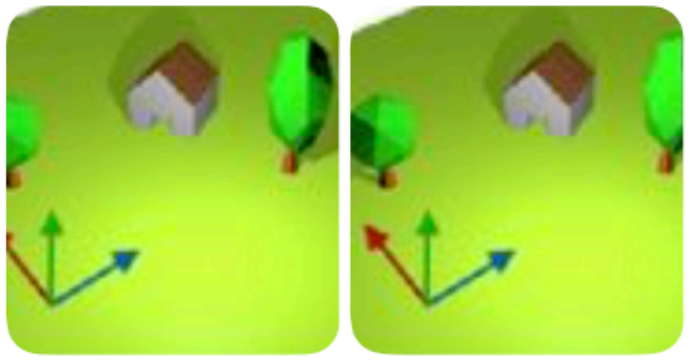
Model-World-View Projection



model space



World space

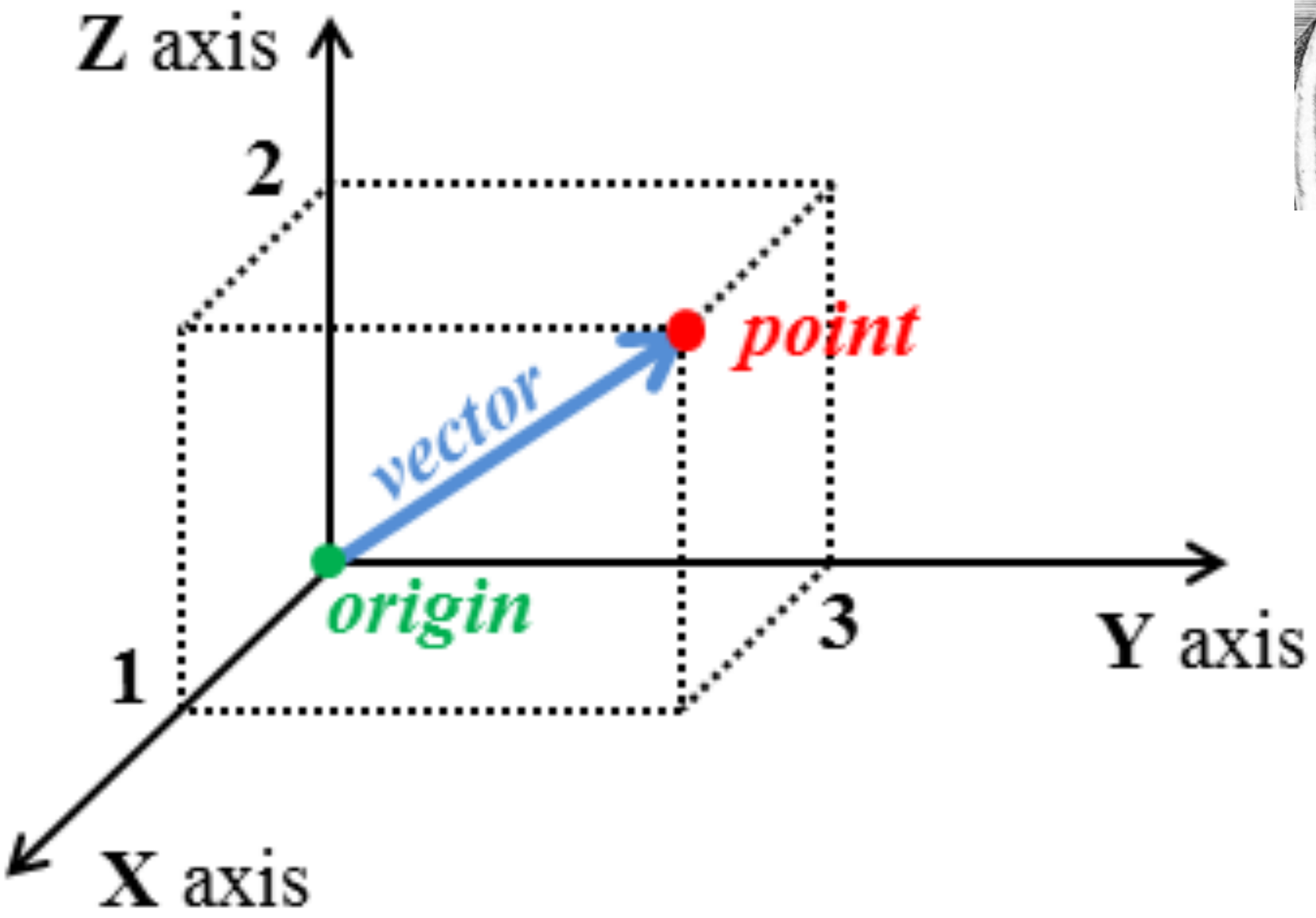


Stereo view space

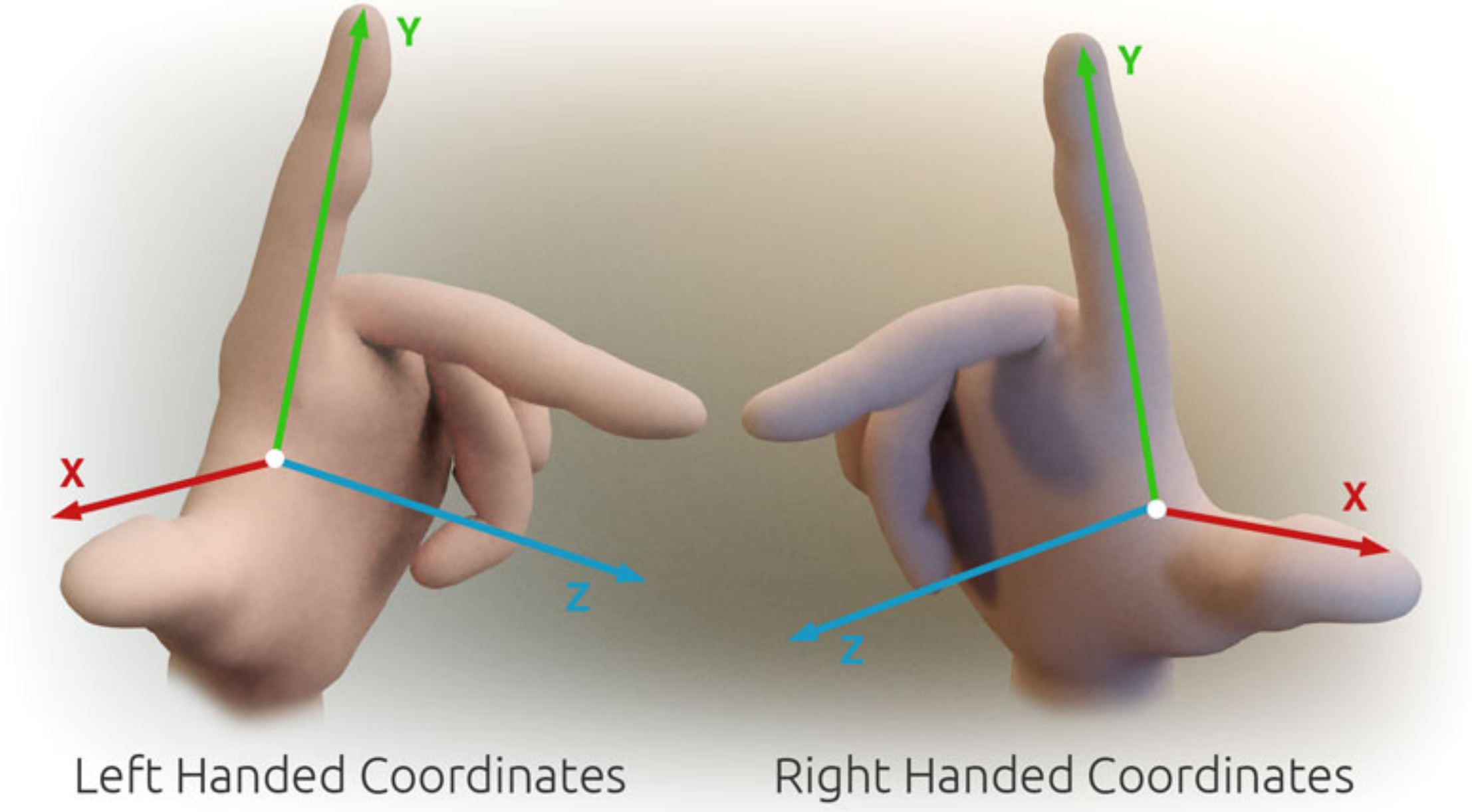
- Vector space = mathematical structure defined by a number of factors (X,Y,Z)
- World space ~ common shared space for your game/VR experience
- Model space ~ vertices/faces relative to a 3D coordinate system of a modeling tool
- View space ~ what the camera sees (e.g. your viewpoint through a VR headset)

Euclidean space

300BC

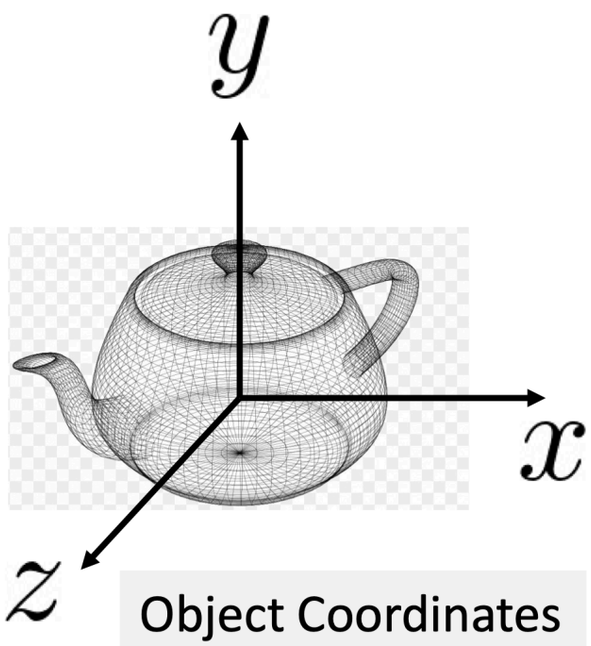


Left vs right hand coordinate systems

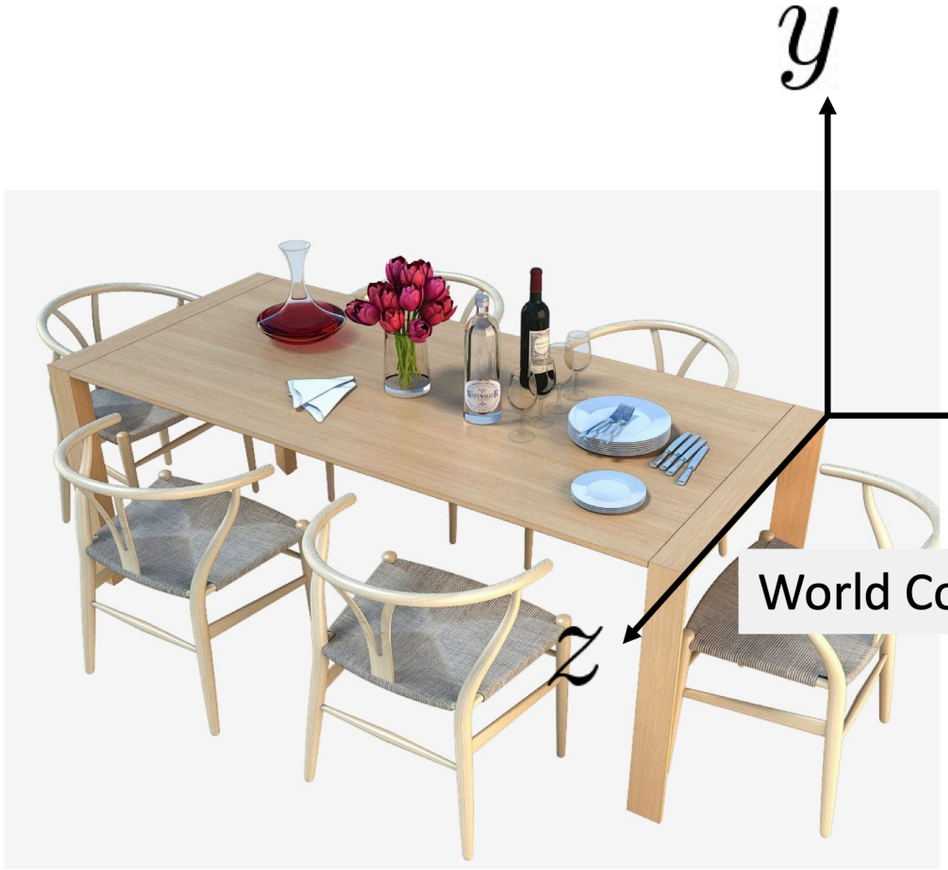


Like driving on the left side of the road ;-)

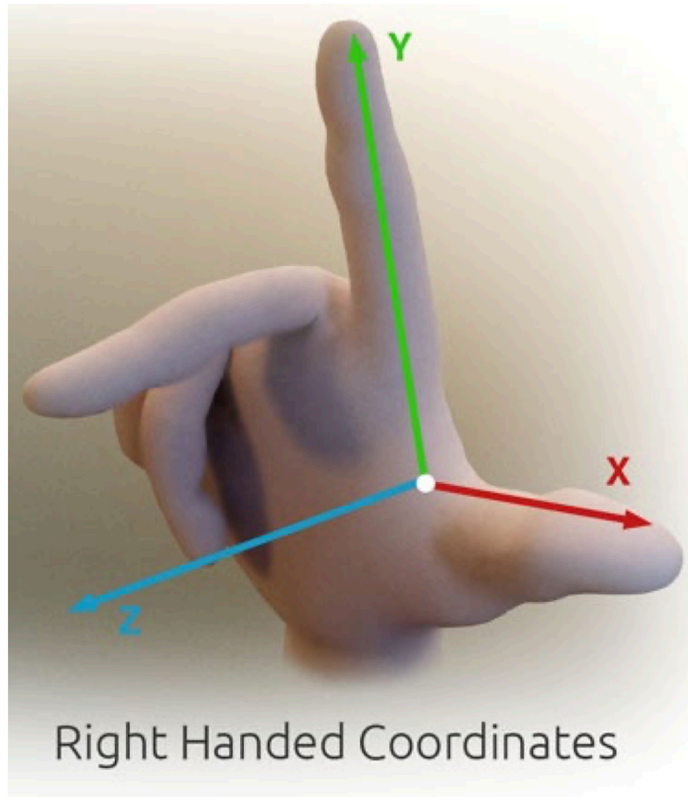
Coordinate Systems



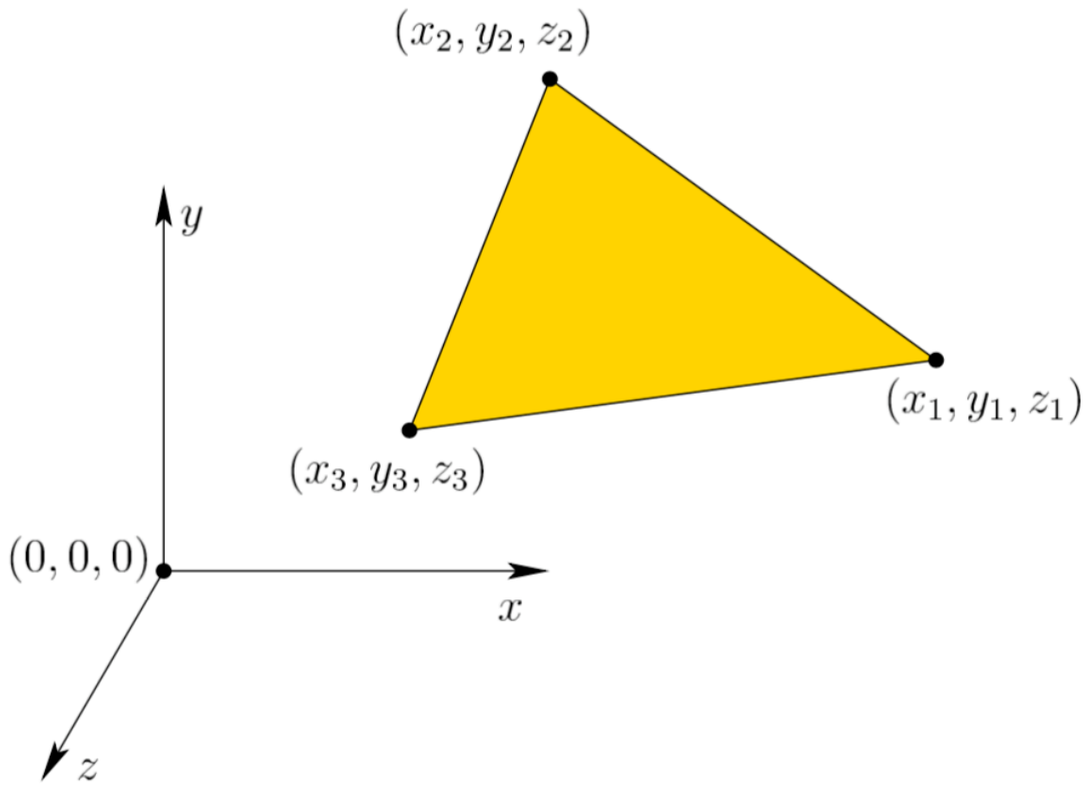
Object Coordinates



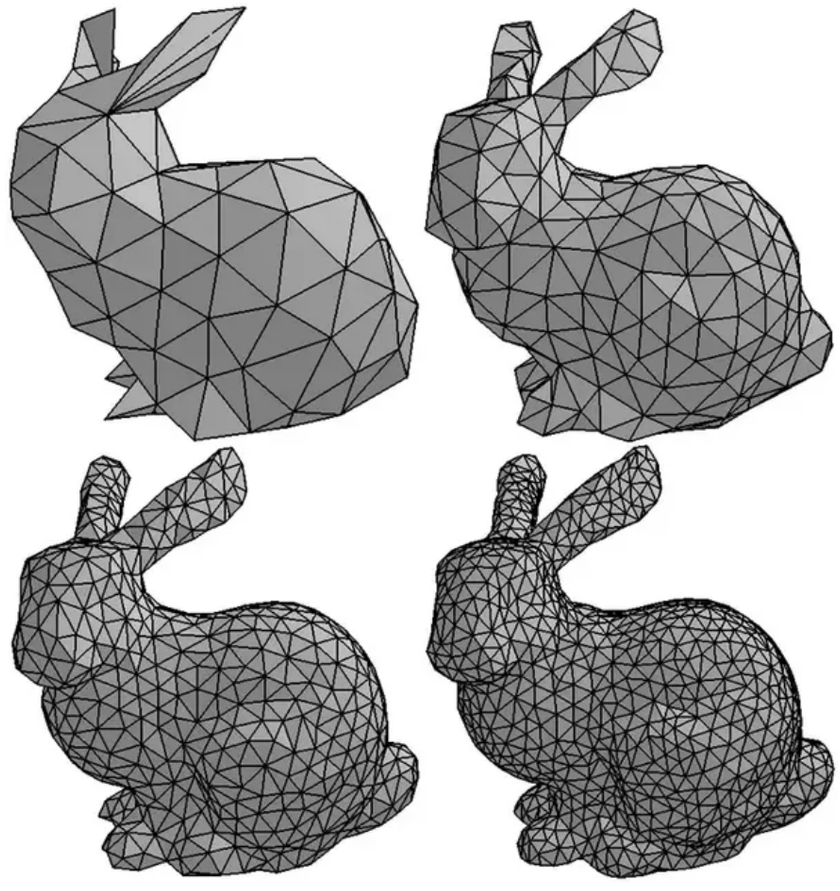
World Coordinates



Define primitives



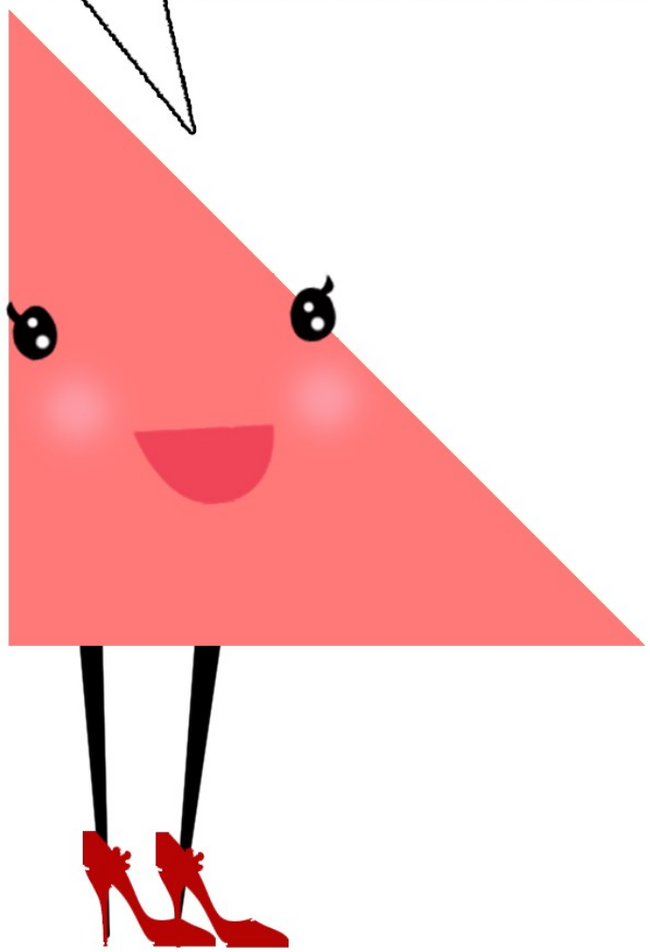
Triangle

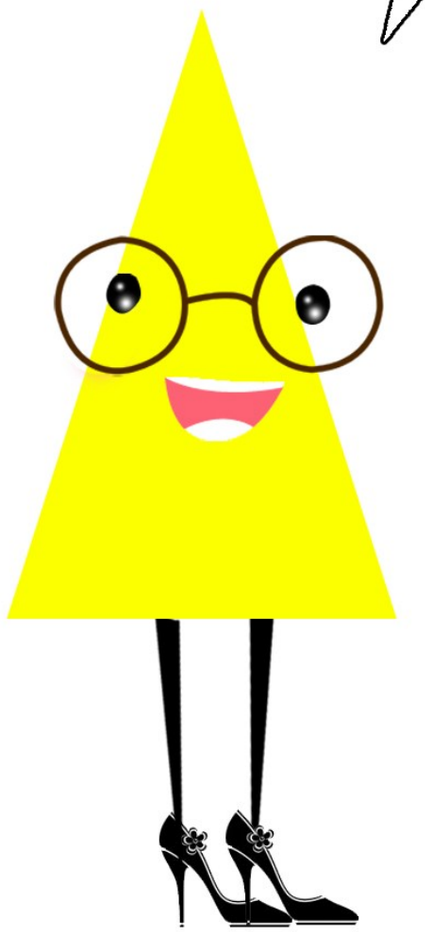


Mesh

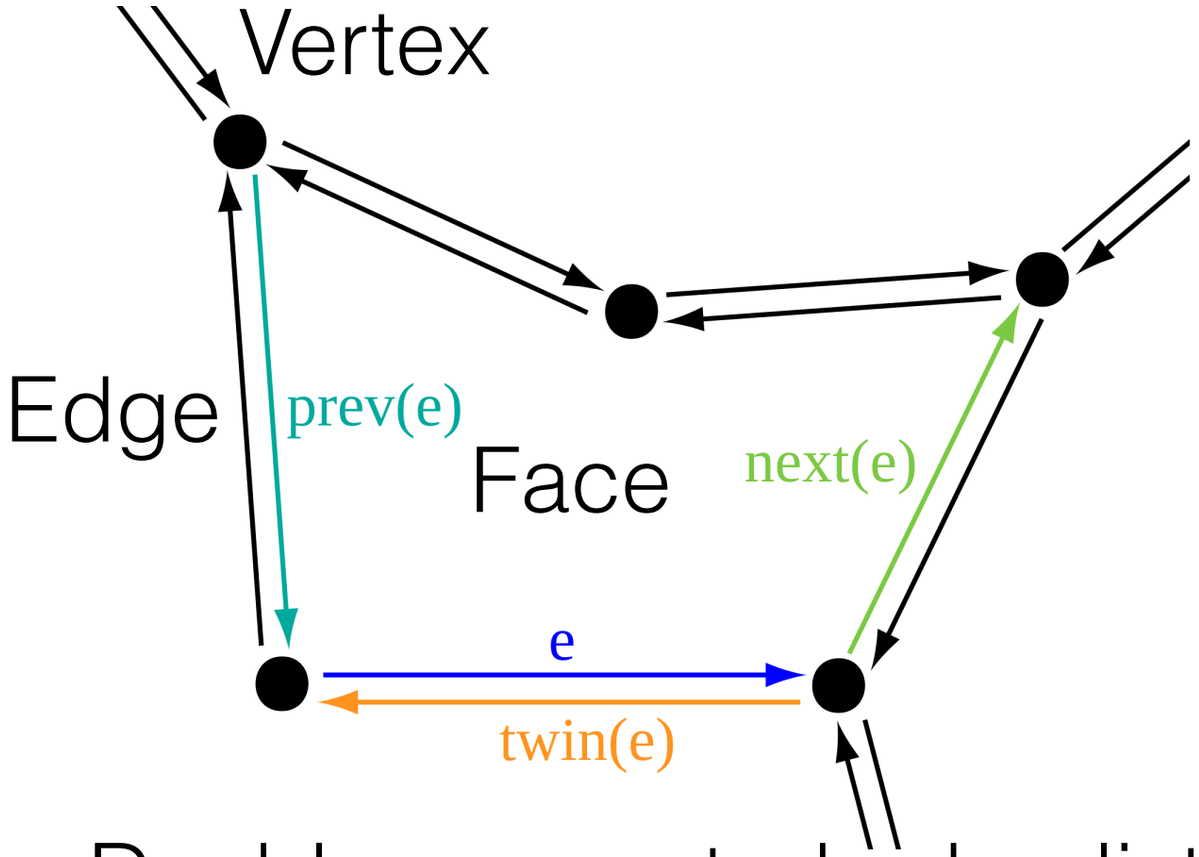
Why is the obtuse triangle always upset?

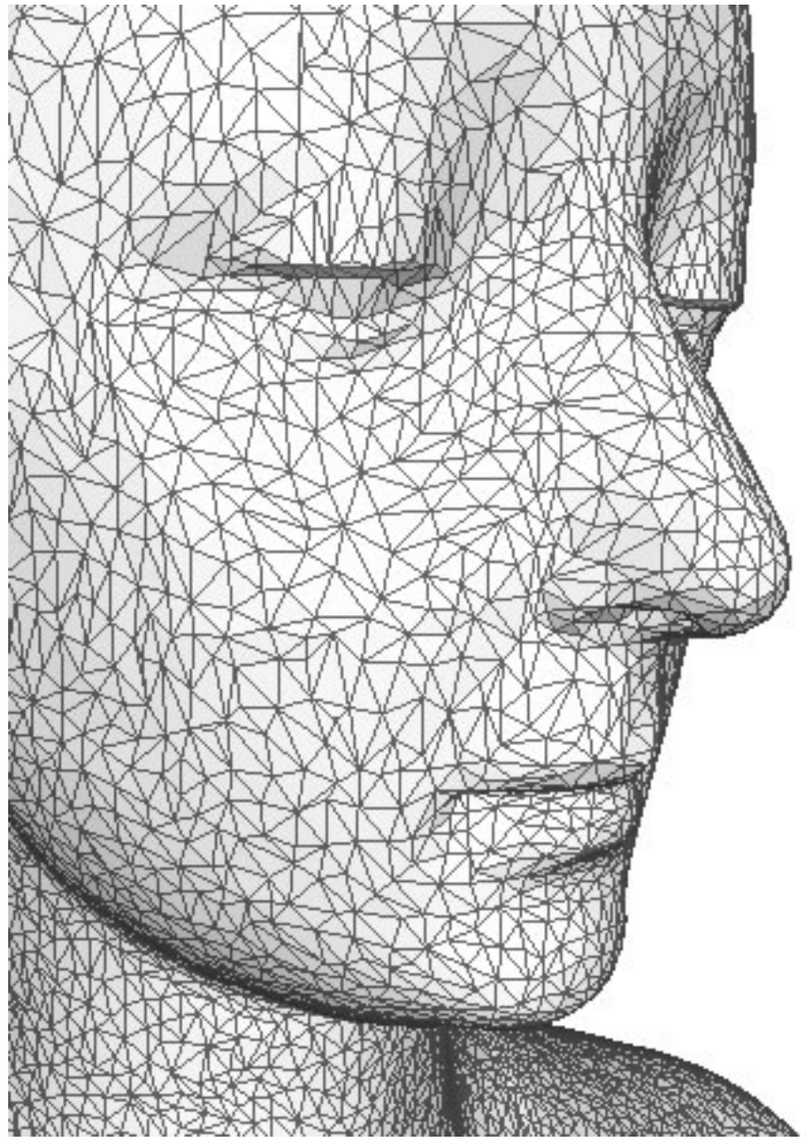
Because it is never right.





How do we store triangles?





Doubly connected edge list
Quickly traverse in a way to how
They are connected

The Virtual World as 3D Triangle Meshes

Face-Vertex Meshes

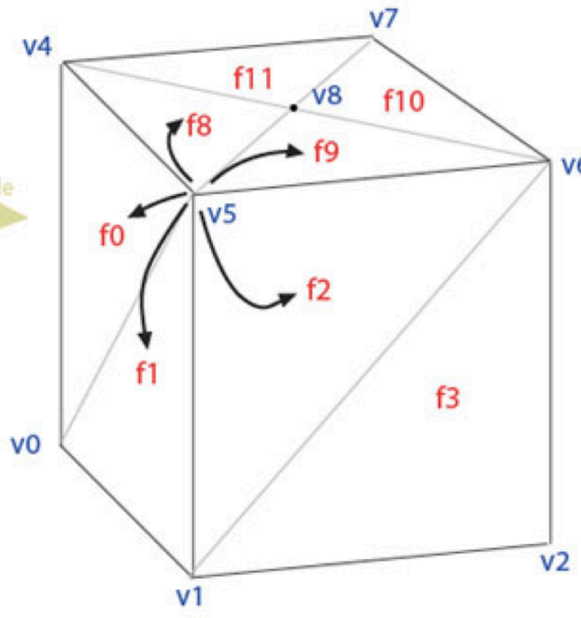
Face List

f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

Vertex List

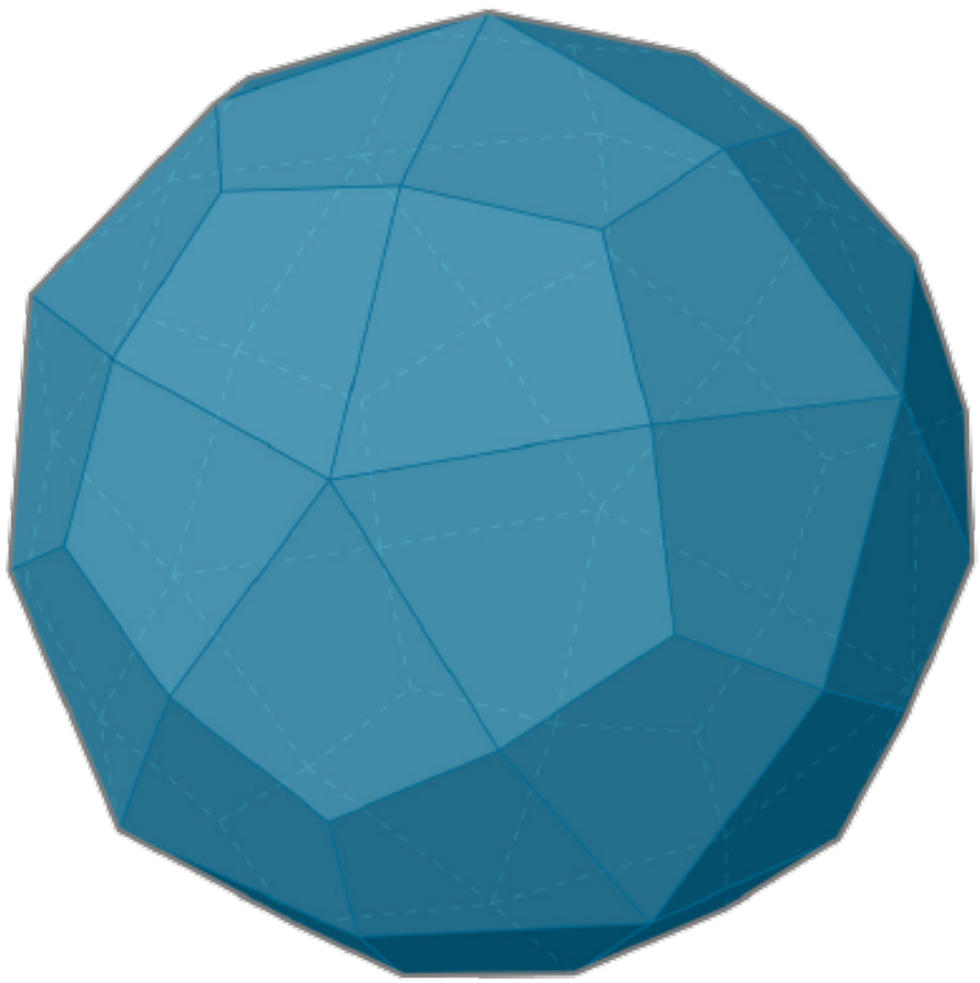
v0	0,0,0	f0 f1 f12 f15 f7
v1	1,0,0	f2 f3 f13 f12 f1
v2	1,1,0	f4 f5 f14 f13 f3
v3	0,1,0	f6 f7 f15 f14 f5
v4	0,0,1	f6 f7 f0 f8 f11
v5	1,0,1	f0 f1 f2 f9 f8
v6	1,1,1	f2 f3 f4 f10 f9
v7	0,1,1	f4 f5 f6 f11 f10
v8	.5,.5,0	f8 f9 f10 f11
v9	.5,.5,1	f12 13 14 15

example



From Wikipedia

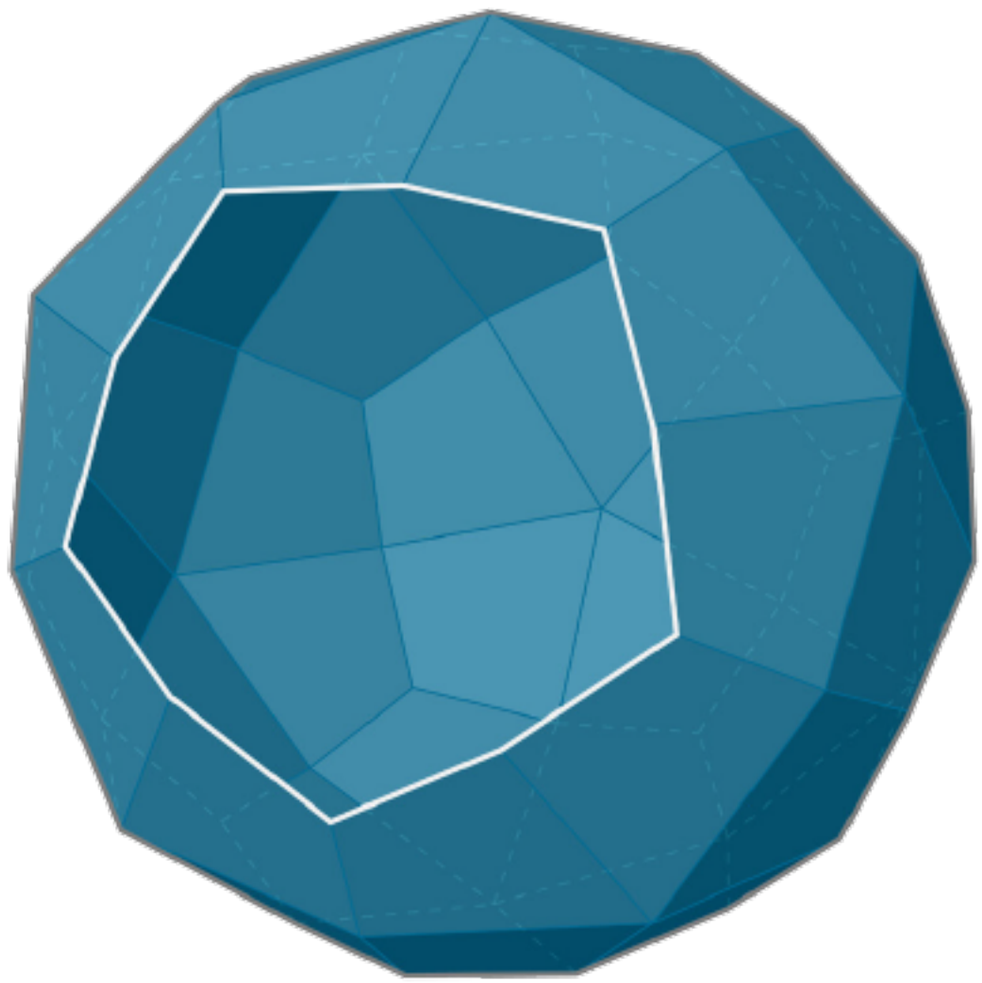
Topology



outside/inside

Coherent

1

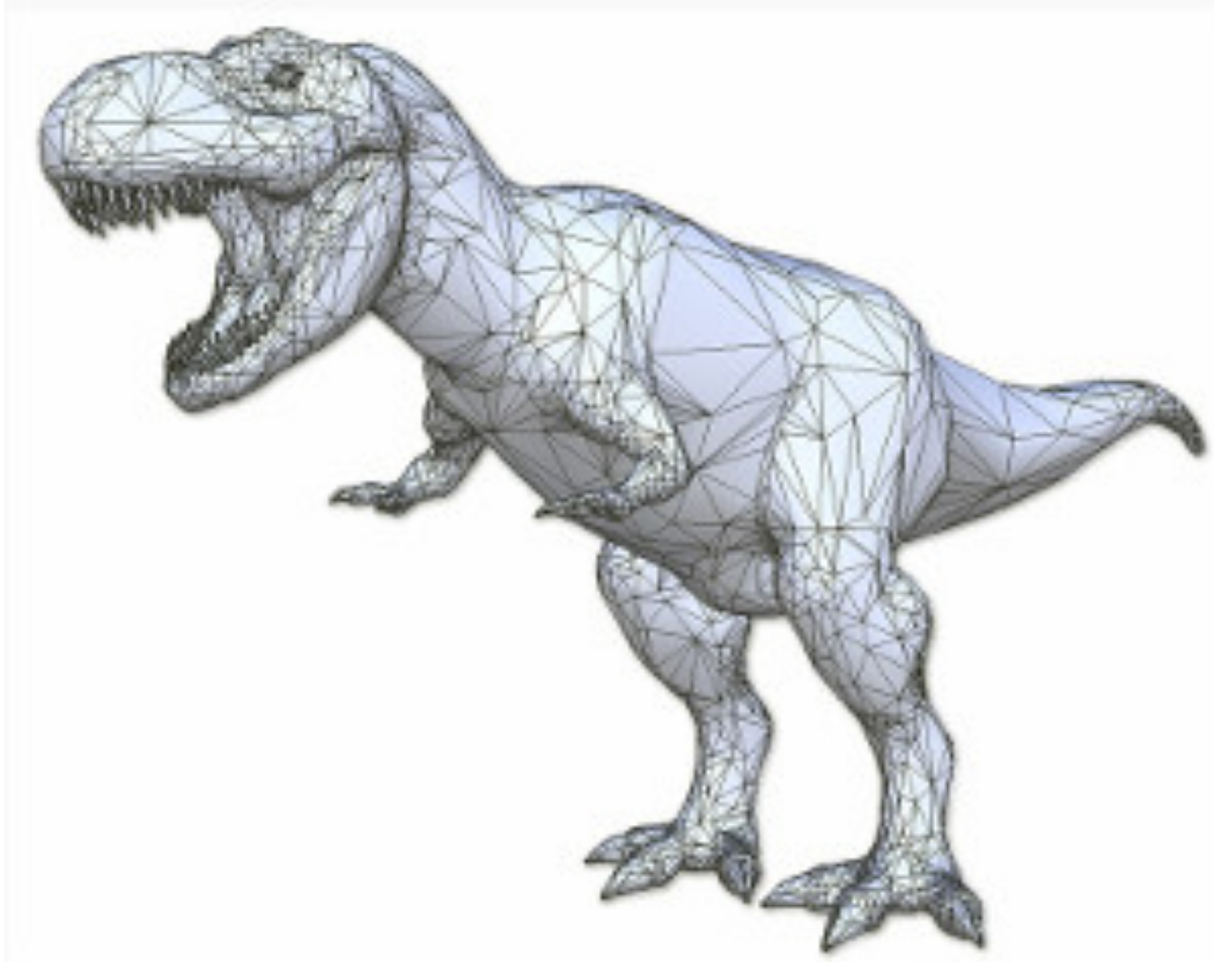


outside/inside

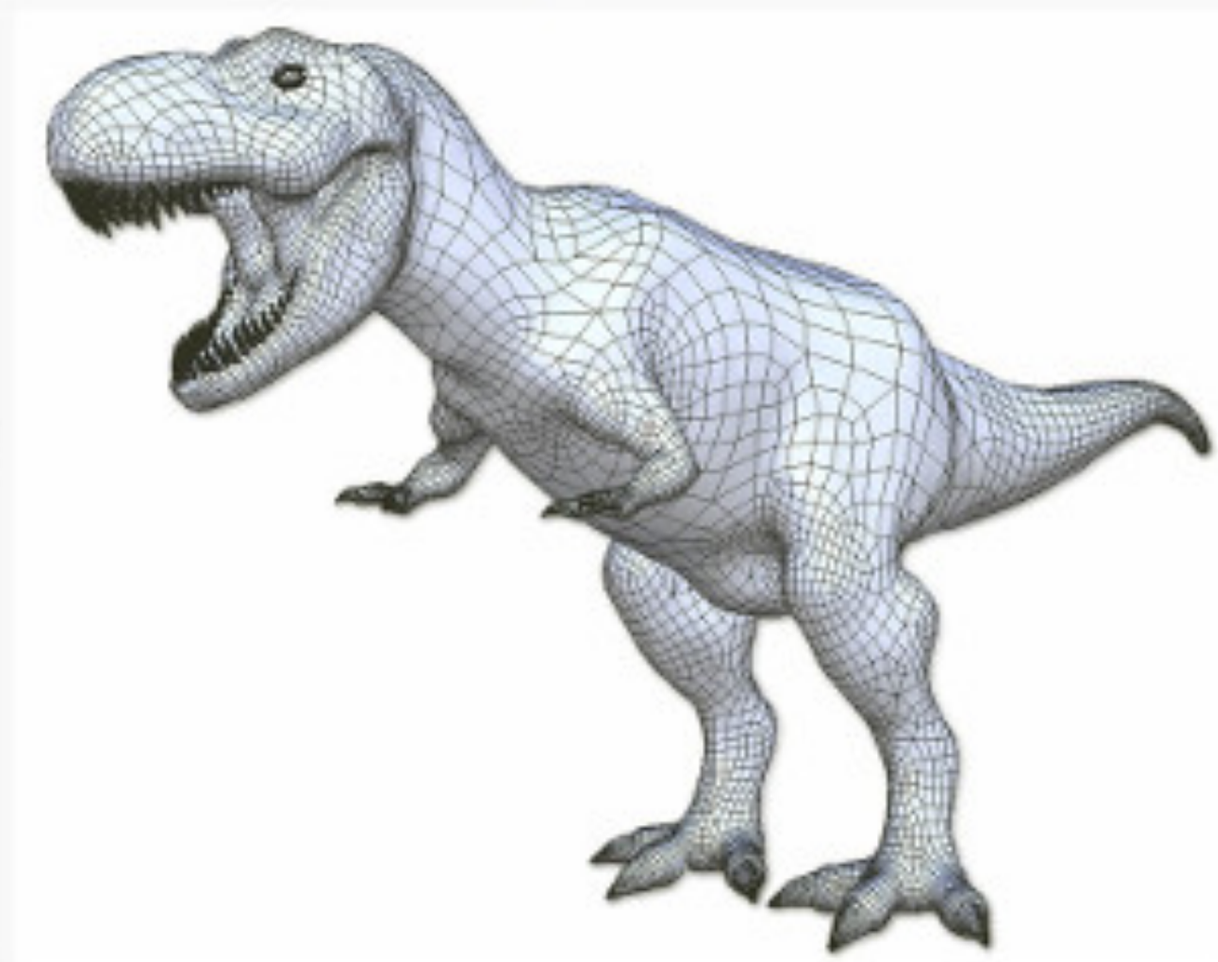
inCoherent

2

Why triangles?



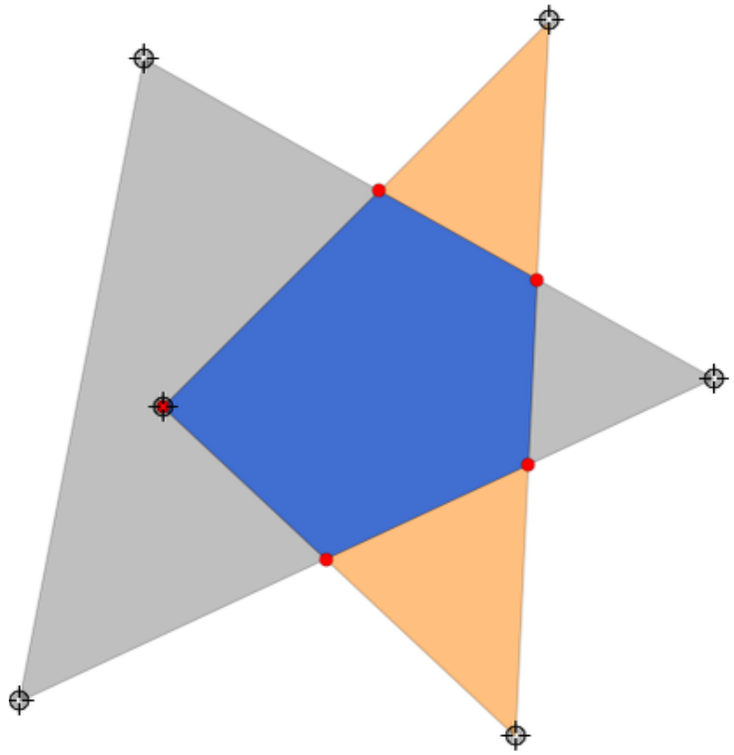
Simple to render
More primitives



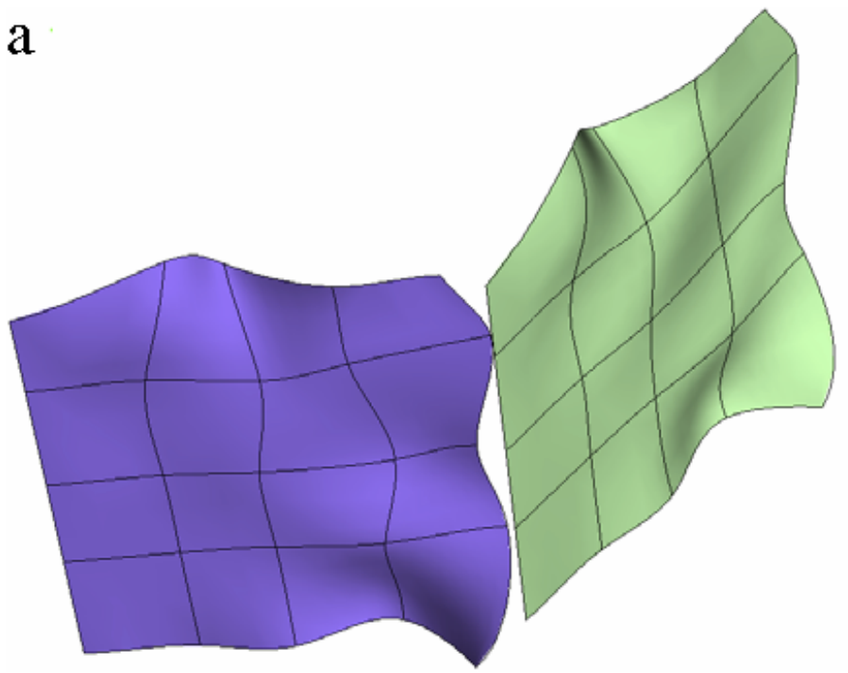
Fewer primitives
computational intensive

…….Computers are very good at doing simple things billions of times

Collision detection



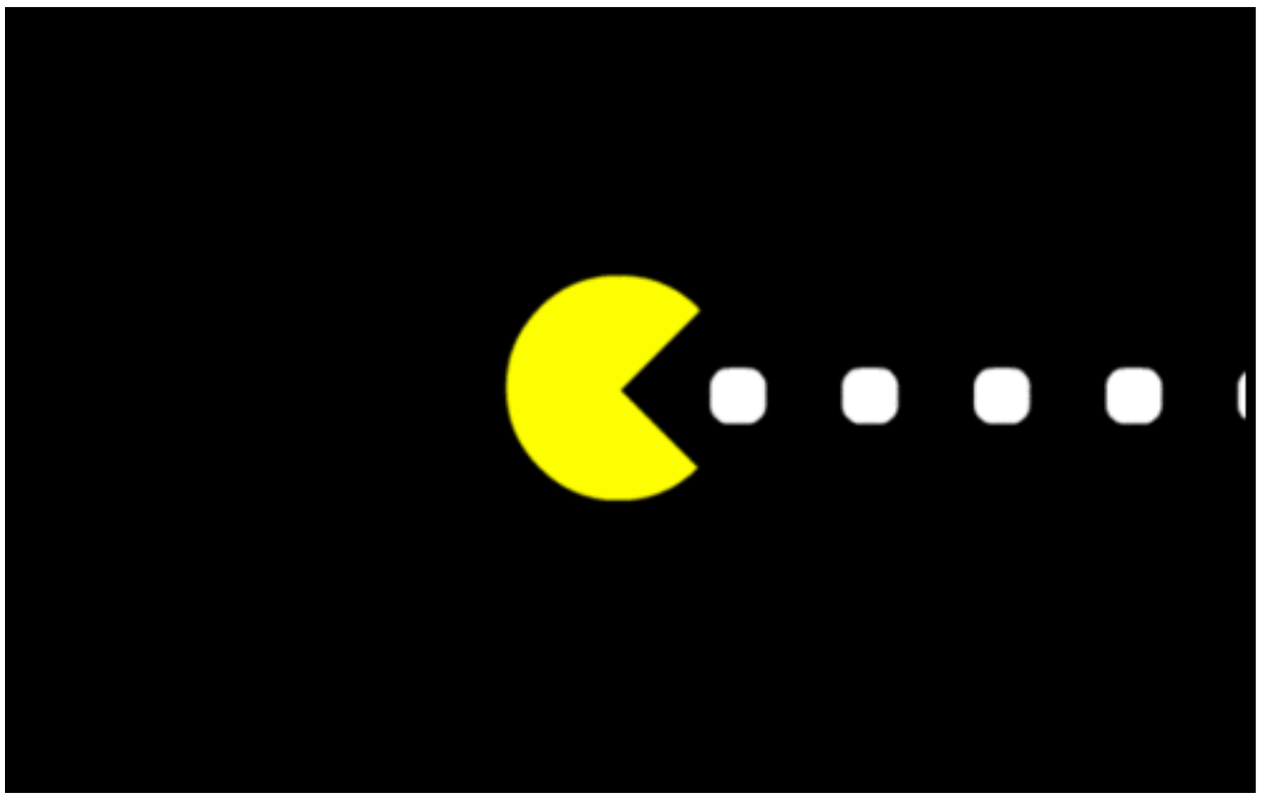
Easier



Harder

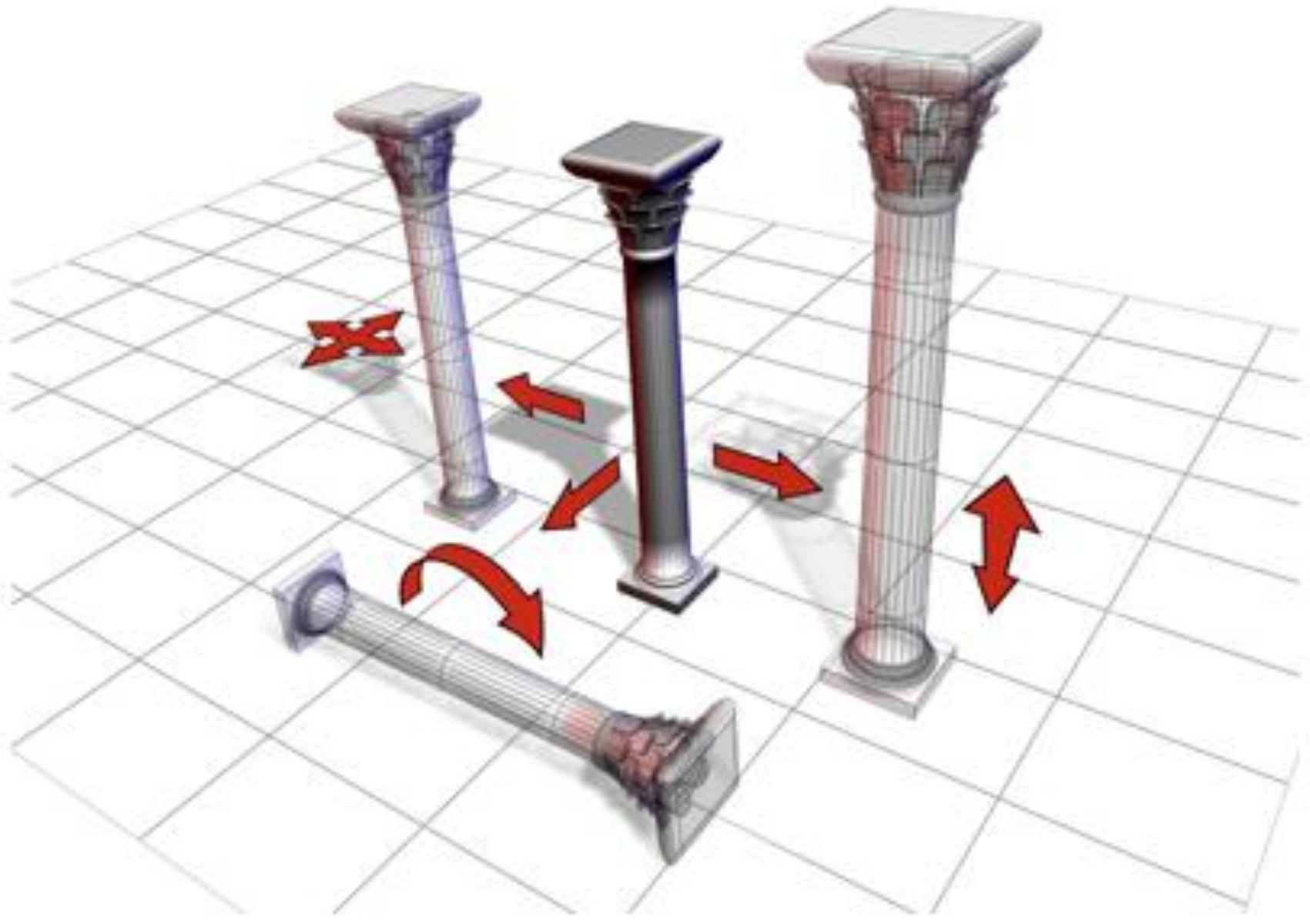
Stationary vs movable

Animation

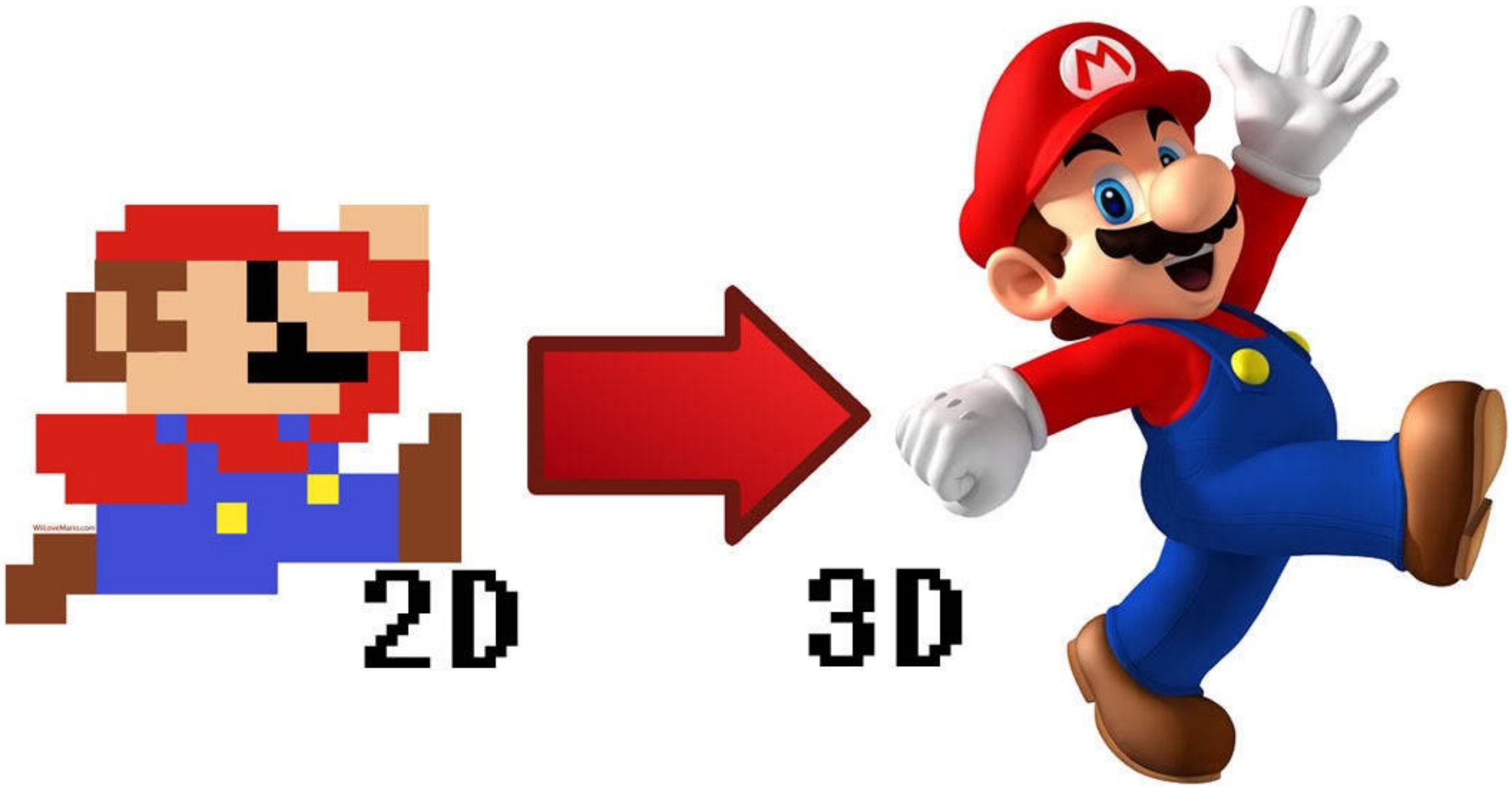


Every type of behavior needs to be defined using geometric Transformations

Types of transformations

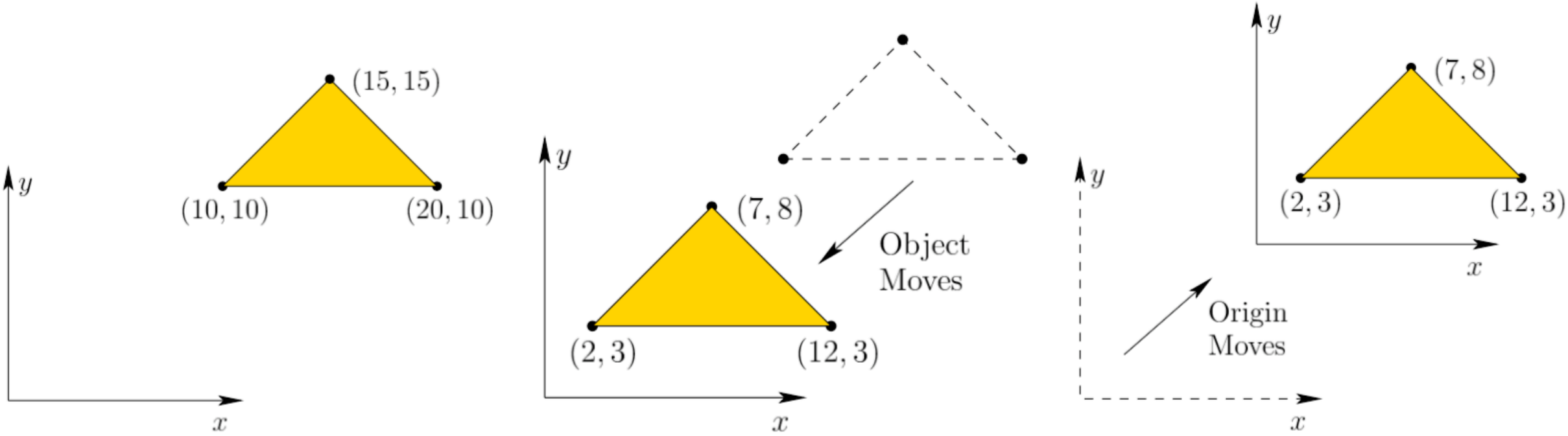


First consider 2D



Translation 2D first

Xt = -8 and Yt = -7



(a) Original object (b) Object moves (c) Origin moves

Two ways to interpret

Transformations 2D

$$M = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}$$

matrix

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

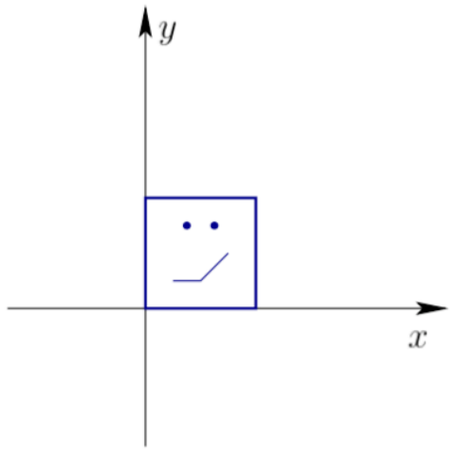
Multiplication

$$x' = m_{11}x + m_{12}y$$
$$y' = m_{21}x + m_{22}y.$$

Result

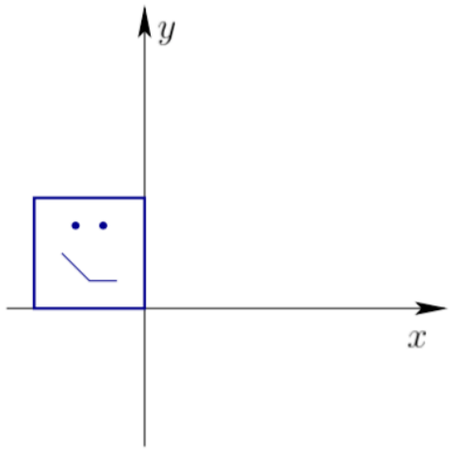
Examples 2D

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



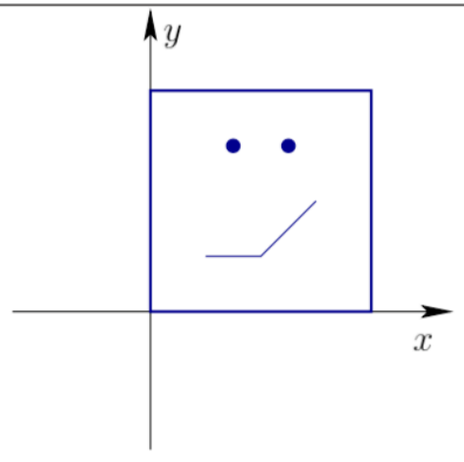
Identity

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$



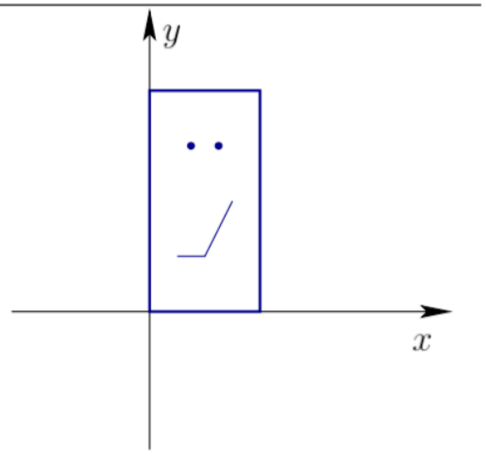
Mirror

$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$



Scale

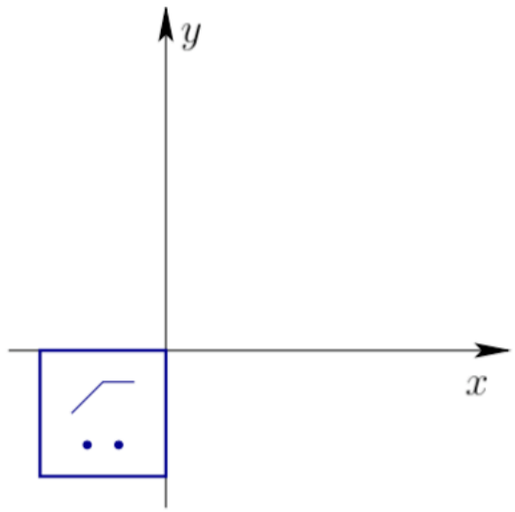
$$\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$



Stretch

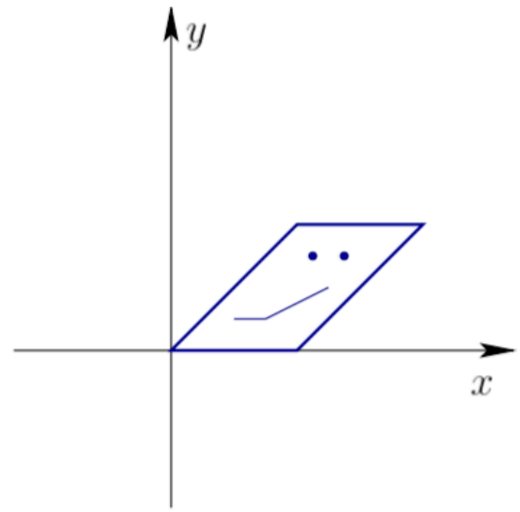
Examples 2D

$$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$



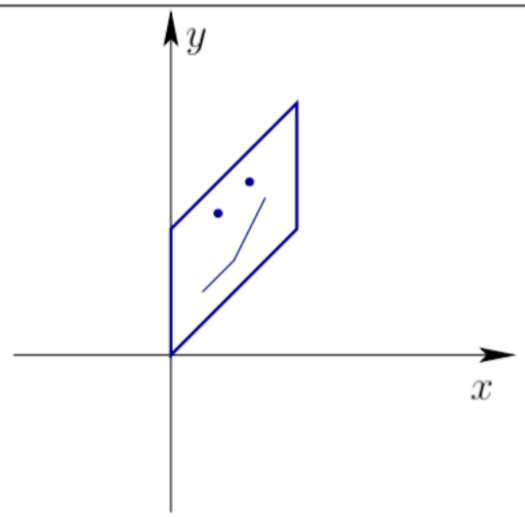
Rotate 180

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



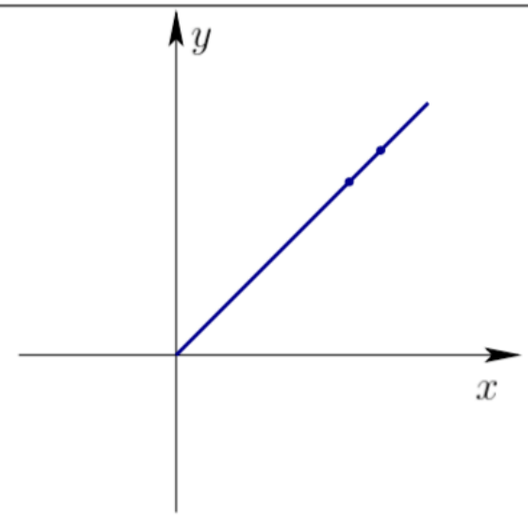
x-shear

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



y-shear

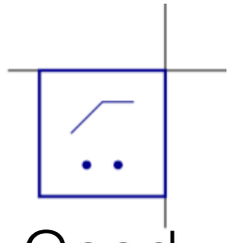
$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$



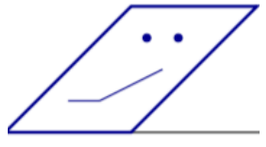
Singular

Rotations 2D

- Problem: how to do a rotation without destroying the object?



Good



Bad

$$M = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}$$

- Constrain M
  - No stretching of axis (columns must have unit length) $m_{11}^2 + m_{21}^2 = 1 \text{ and } m_{12}^2 + m_{22}^2 = 1.$

$$m_{11}m_{12} + m_{21}m_{22} = 0.$$

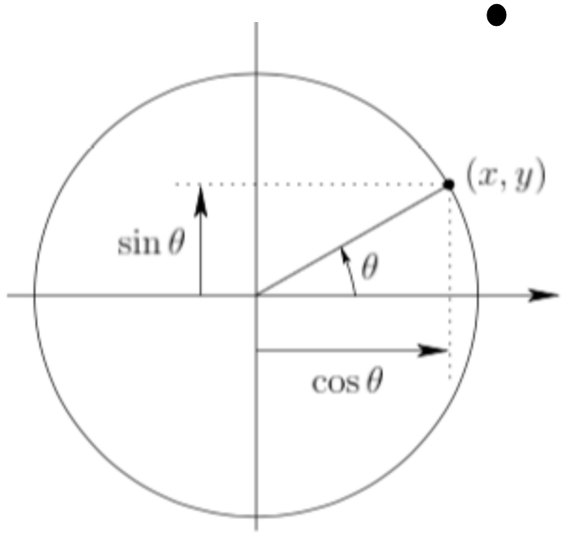
  - No shearing (inner dot product is zero)
  - No mirror images (determinant must >0)

$$\det \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} = m_{11}m_{22} - m_{12}m_{21} = 1.$$

Rotations as angle in 2D

$$m_{11}^2 + m_{21}^2 = 1 \text{ and } m_{12}^2 + m_{22}^2 = 1.$$

- First constraint mandates that components lie on a circle
  - $x^2+y^2=1$



$x = \cos\theta \text{ and } y = \sin\theta.$

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix},$$

Vary **ϕ from 0 to 2π radians**

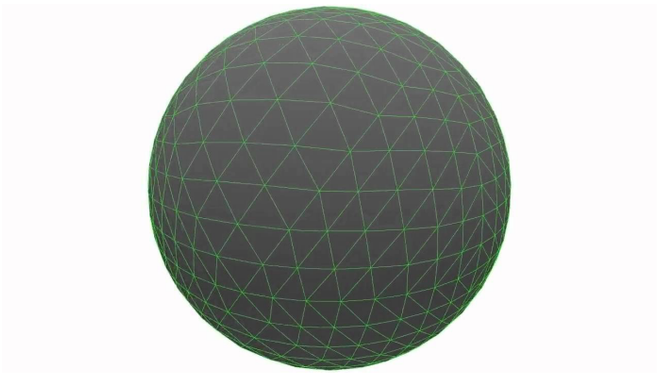
Create any rotation between 0 and 360

Doing it in 3D

- Constraints on M $$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}.$$
  - No stretching of axis (columns must have unit length) $$m_{11}^2 + m_{21}^2 + m_{31}^2 = 1.$$
  - No shearing (inner dot product is zero) $$m_{11}m_{12} + m_{21}m_{22} + m_{31}m_{32} = 0.$$
  - No mirror images (determinant must >0)

    det[M] = 1

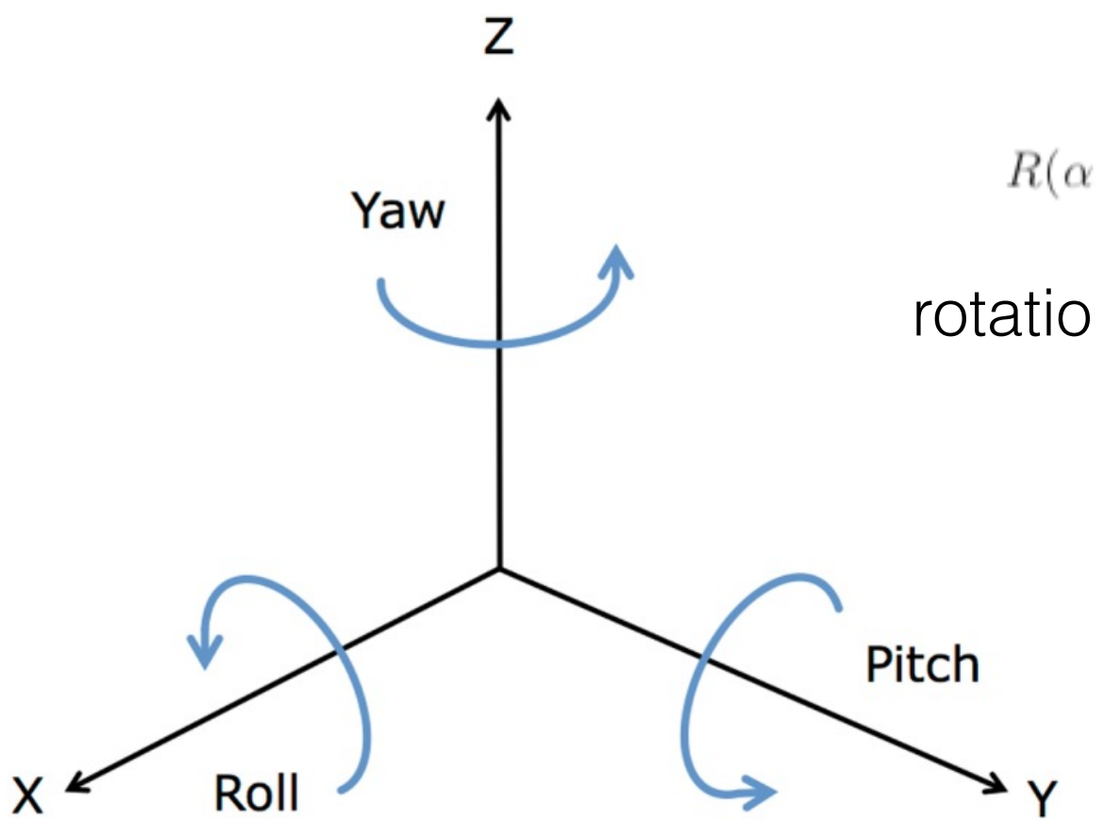
Rotations as angle in 3D



- $x^2+y^2+z^2=1$

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Similar as 2D (gamma)



combine:

$$R(\alpha, \beta, \gamma) = R_y(\alpha)R_x(\beta)R_z(\gamma).$$

rotations are applied backwards

$$R_x(\beta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta \\ 0 & \sin\beta & \cos\beta \end{bmatrix}.$$

$$R_y(\alpha) = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix}.$$

Rotation and translation in one matrix

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} .$$

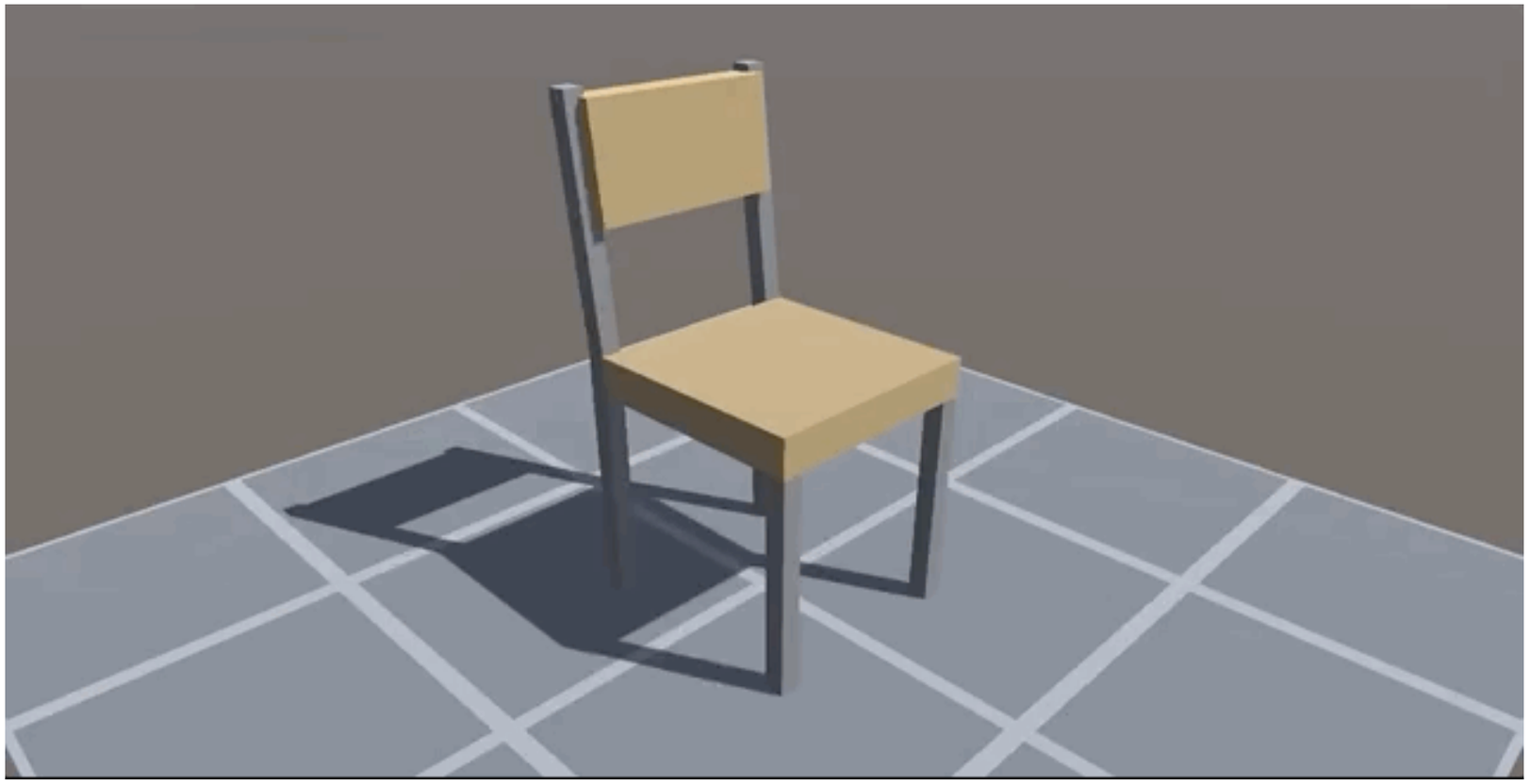
Homogenous transformation matrix

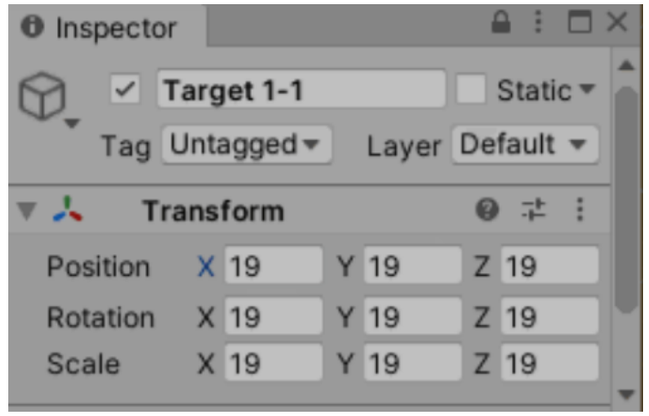
$$\begin{bmatrix} \boxed{R} & \begin{matrix} x_t \\ y_t \\ z_t \end{matrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} .$$

Increase dimension by 1

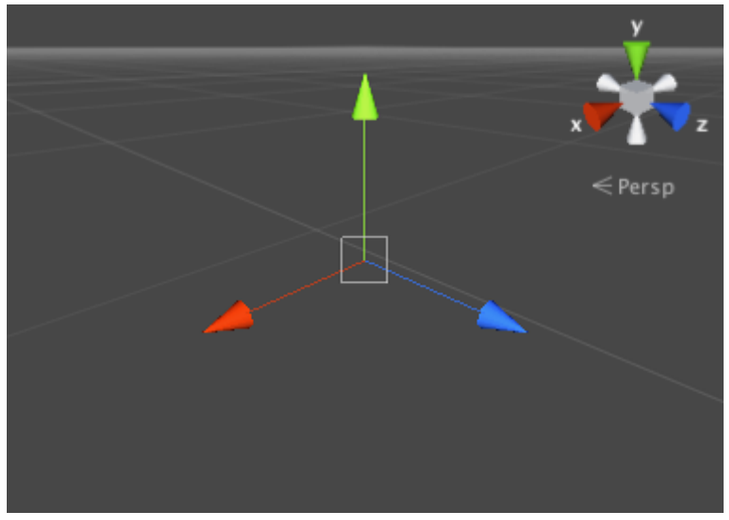
Not commutative

Easy in Unity

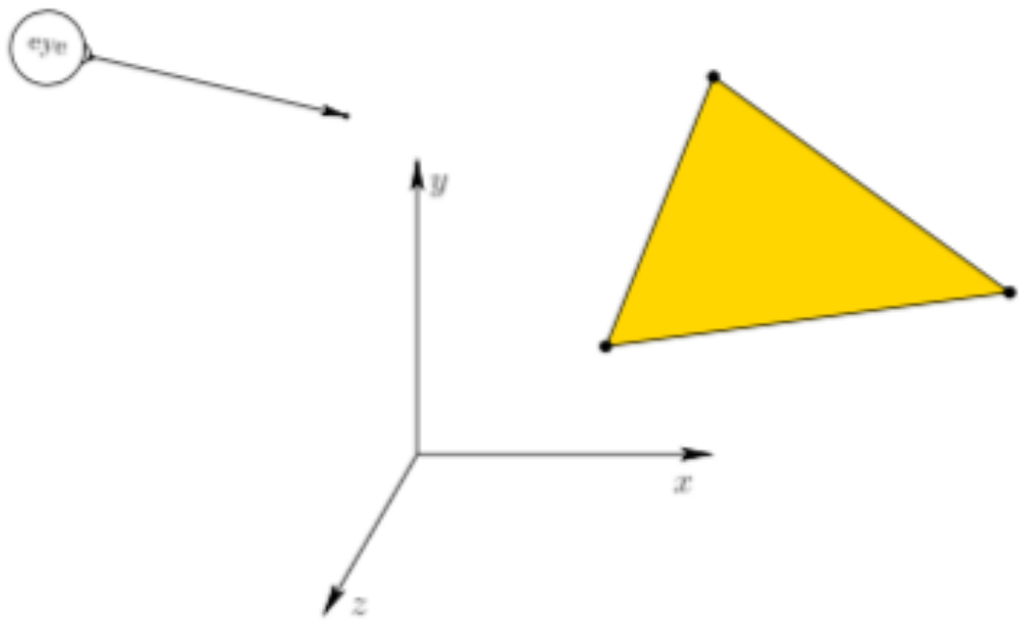




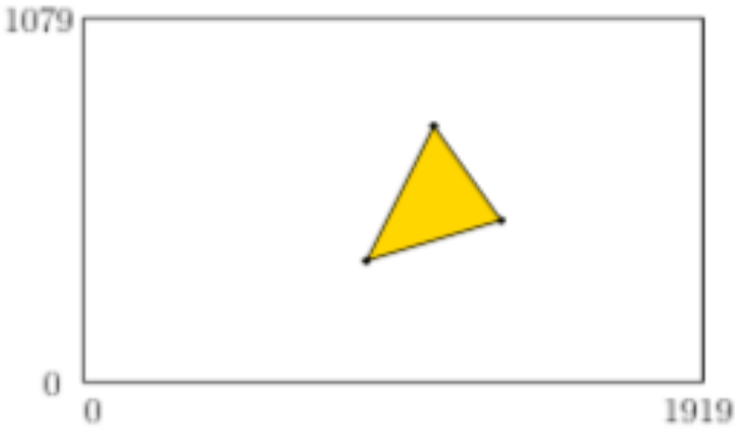

```
void RotateByDegrees()
    {
        Vector3 rotationToAdd = new Vector3(0, 90, 0);
        transform.Rotate(rotationToAdd);
    }
```



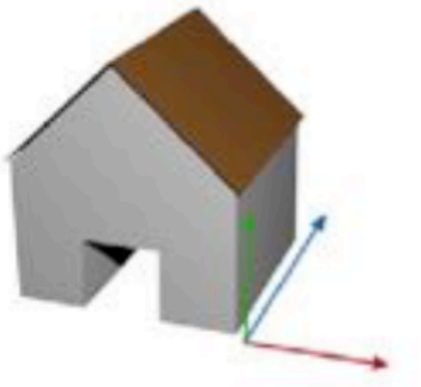
Viewing transformation

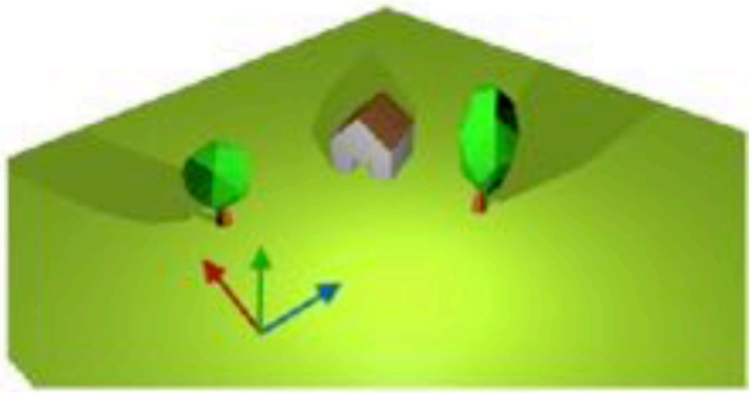


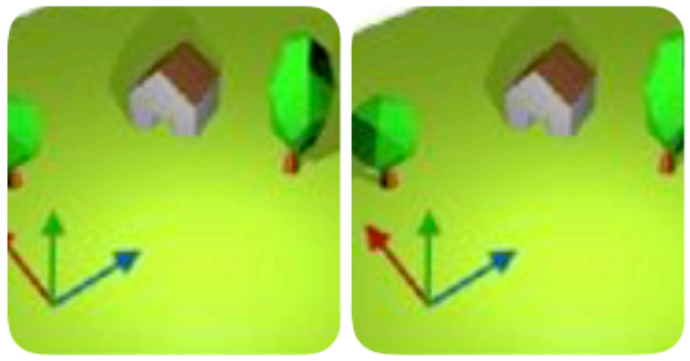
Virtual World



Virtual Screen

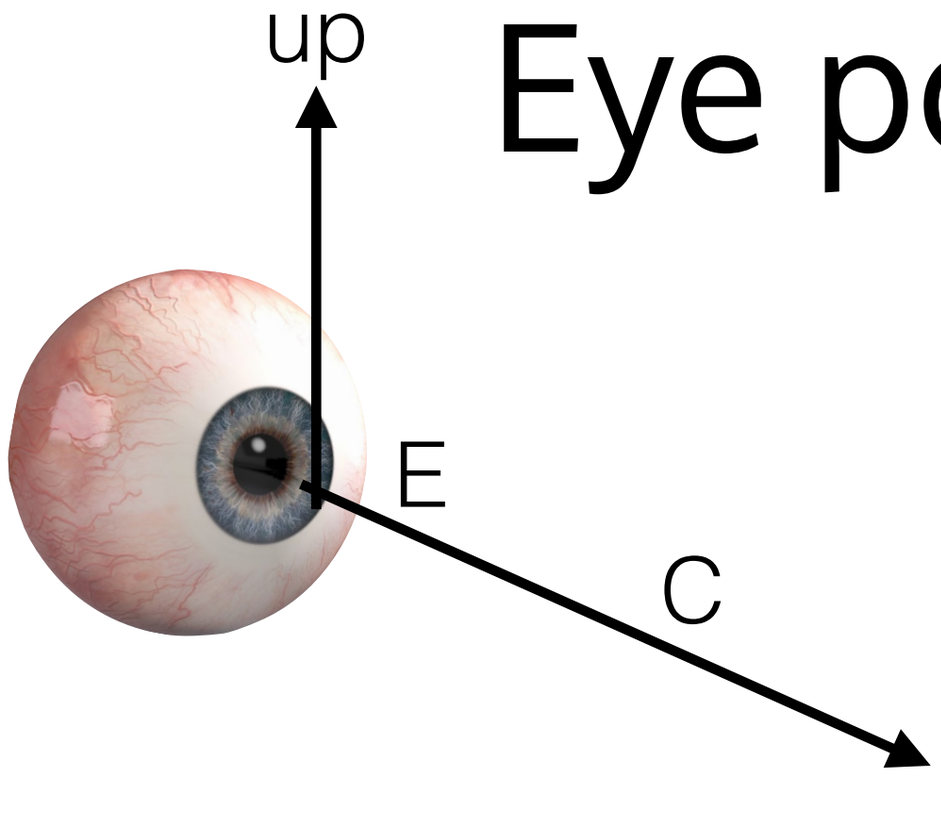






Two ‘eyes’

Eye position



e=(e_1,e_2,e_3)

$$R_{eye} = \begin{bmatrix} \hat{x}_1 & \hat{y}_1 & \hat{z}_1 \\ \hat{x}_2 & \hat{y}_2 & \hat{z}_2 \\ \hat{x}_3 & \hat{y}_3 & \hat{z}_3 \end{bmatrix} \cdot$$ Rotation

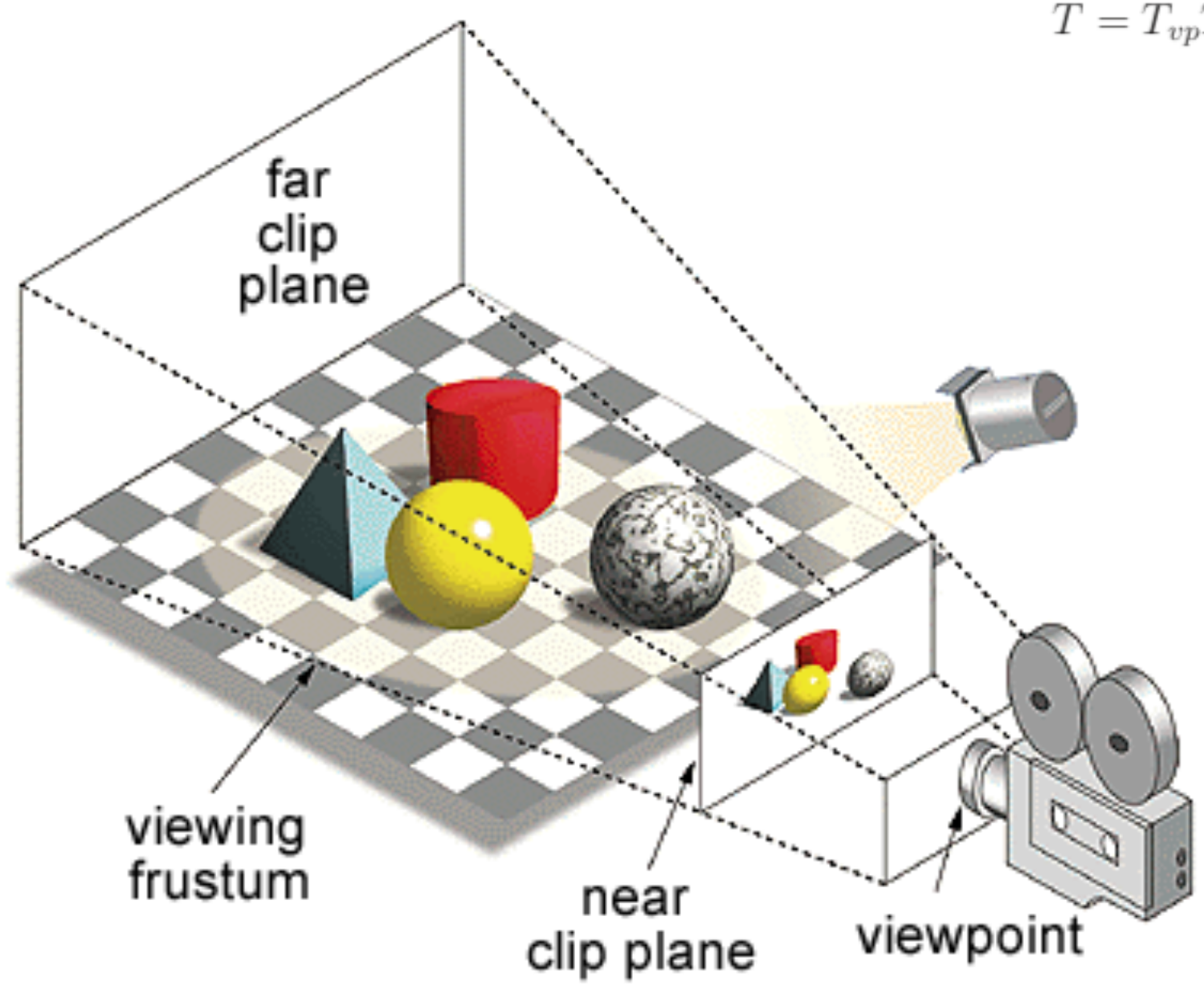
Inverse

$$T_{eye} = \begin{bmatrix} \hat{x}_1 & \hat{x}_2 & \hat{x}_3 & 0 \\ \hat{y}_1 & \hat{y}_2 & \hat{y}_3 & 0 \\ \hat{z}_1 & \hat{z}_2 & \hat{z}_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -e_1 \\ 0 & 1 & 0 & -e_2 \\ 0 & 0 & 1 & -e_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Move all models in VE to the eye's frame of reference

Canonical view transform

$$T = T_{vp}T_{can}T_{eye}T_{rb}.$$



More in chapter 7