Graphical Projection

CMSC 161: Interactive Computer Graphics

2nd Semester 2014-2015

Institute of Computer Science

University of the Philippines – Los Baños

Lecture by James Carlo Plaras

Graphical Projection

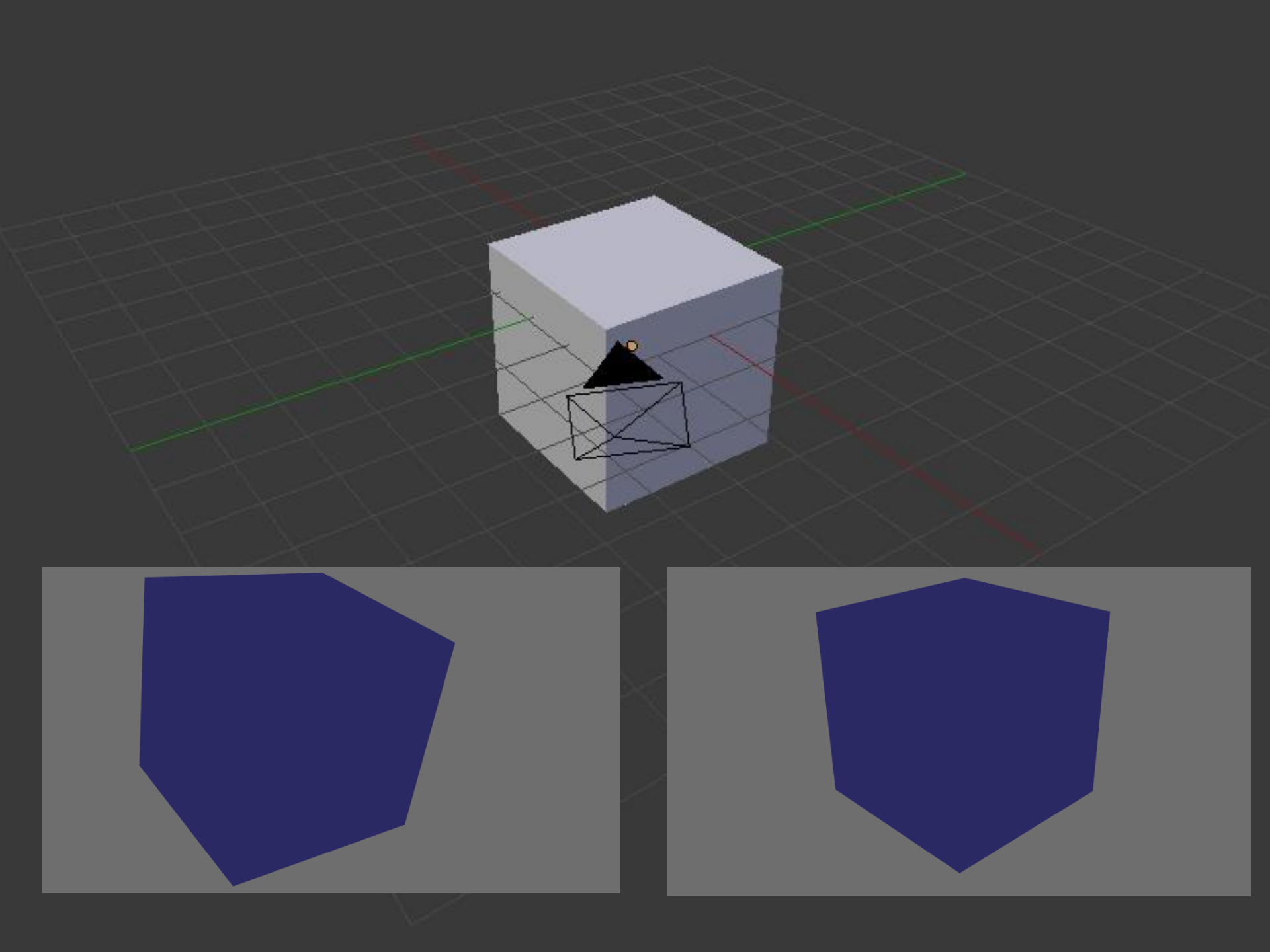
Method of projecting a 3D Object in a 2D Space

Fancy term for **viewing**

Graphical Projection

Utilizes the concept of **principal face**

Objects have front, back, top, bottom, right and left faces

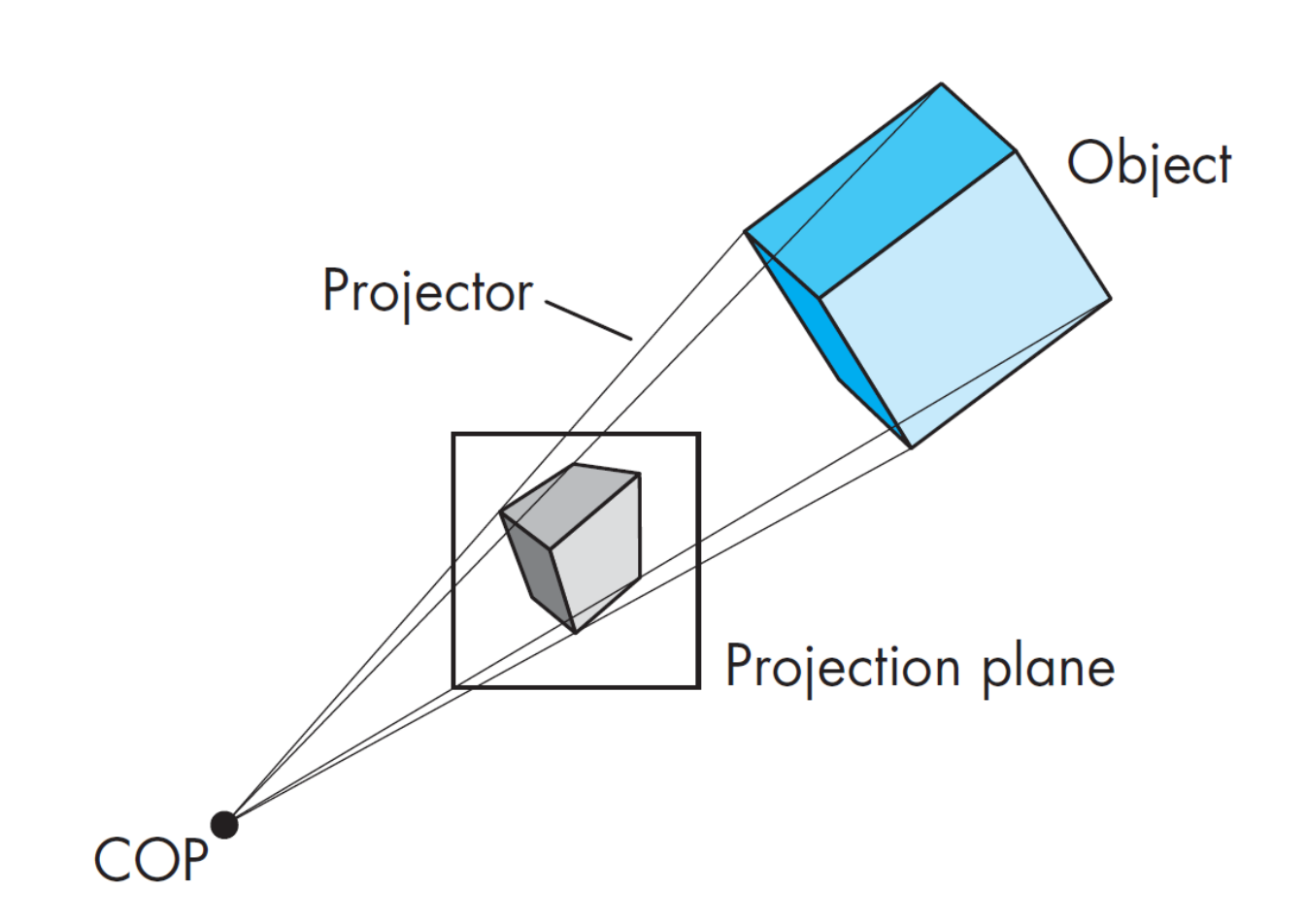


Types of Projection

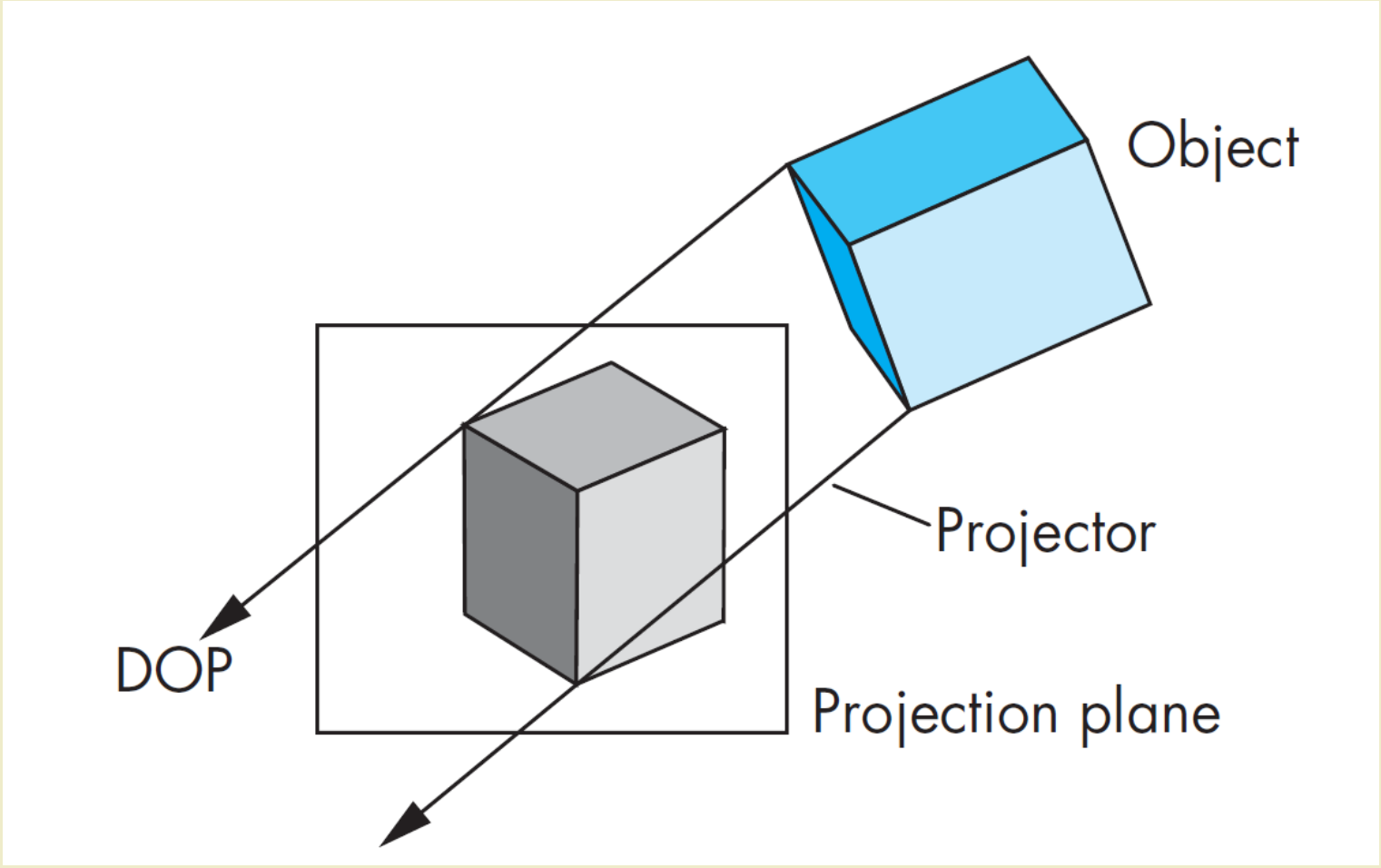
Parallel Projection

Perspective Projection

Perspective Projection



Parallel Projection



PARALLEL PROJECTION

Types of Parallel Projection

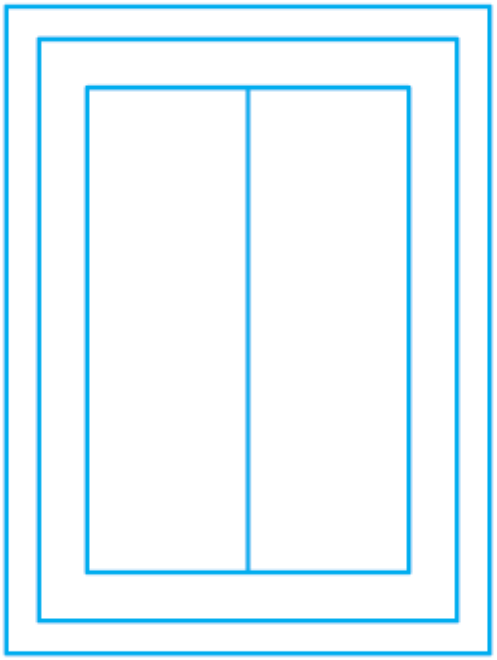
Orthographic Projections

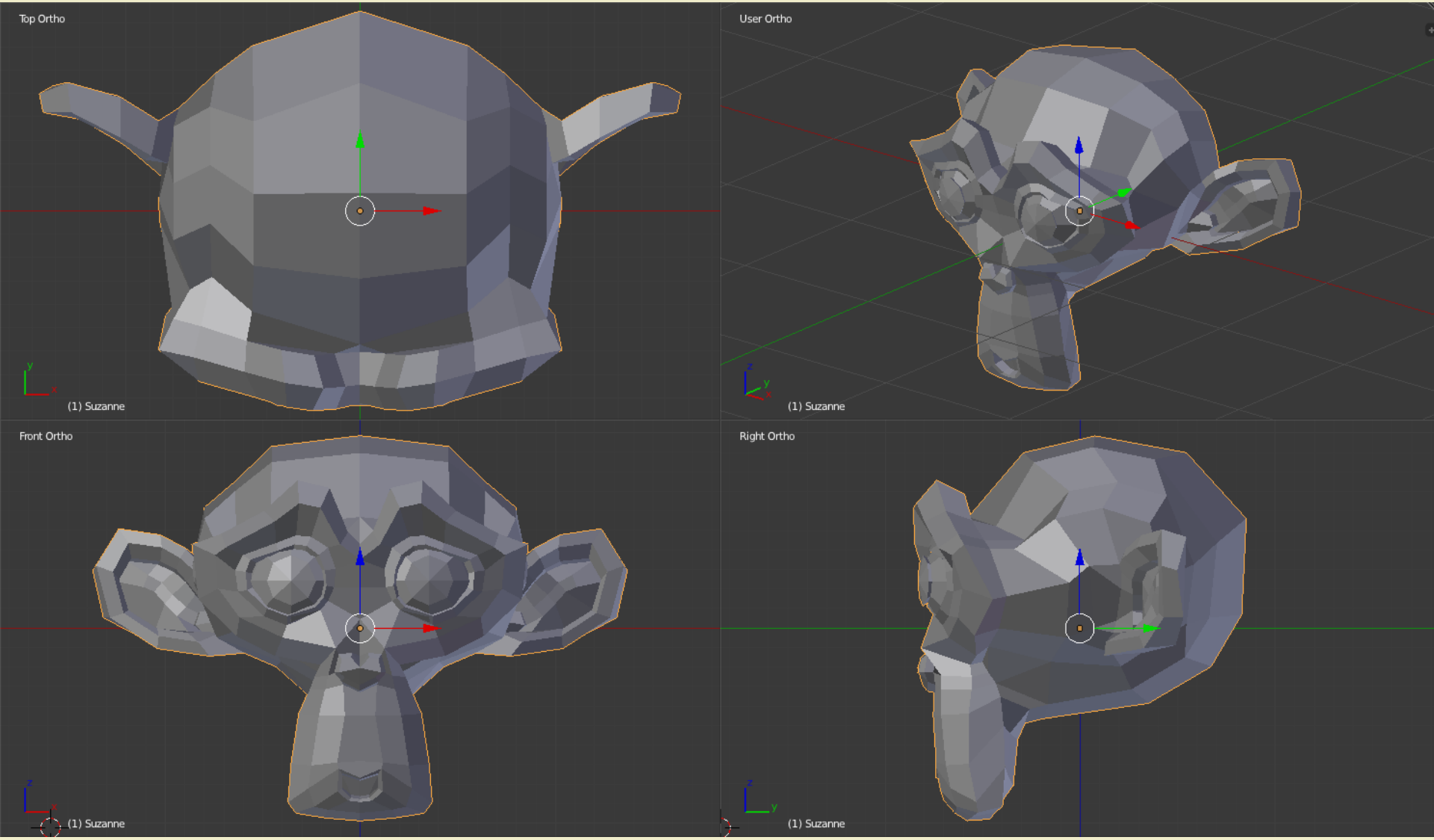
Axonometric Projections

Oblique Projections

Orthographic Projection

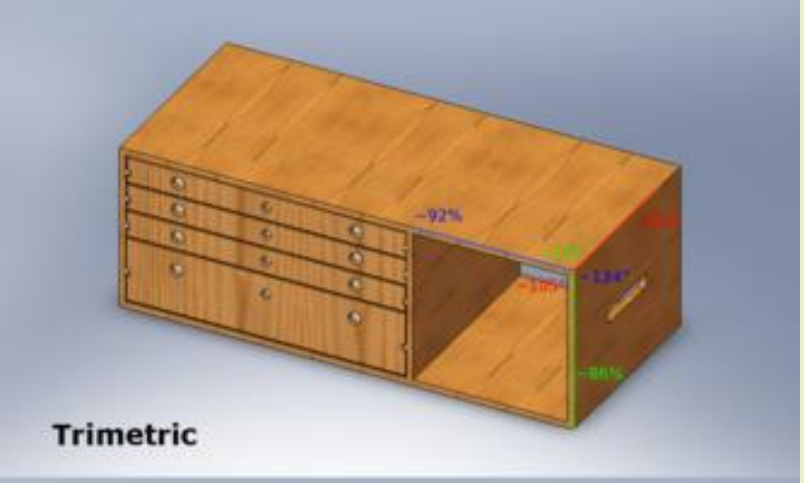
1. Projectors must perpendicular to the projection plane
2. Projection plane is parallel to one of the principal faces of the object



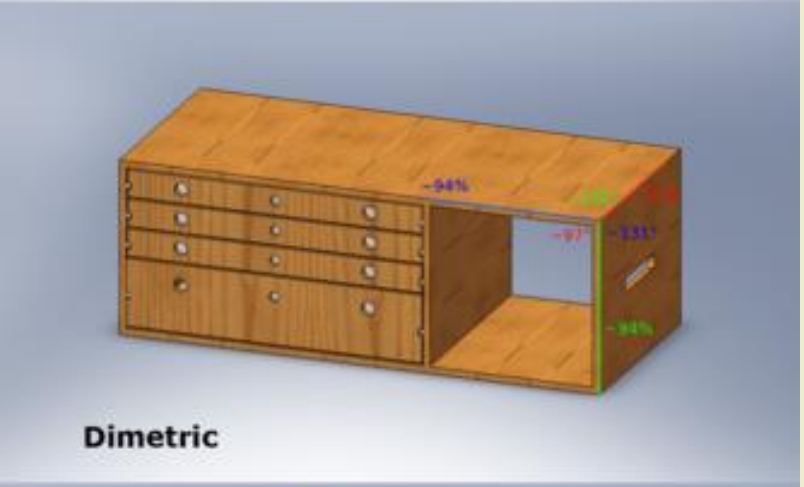

Top Ortho
(1) Suzanne
User Ortho
(1) Suzanne
Front Ortho
(1) Suzanne
Right Ortho
(1) Suzanne

Axonometric Projections

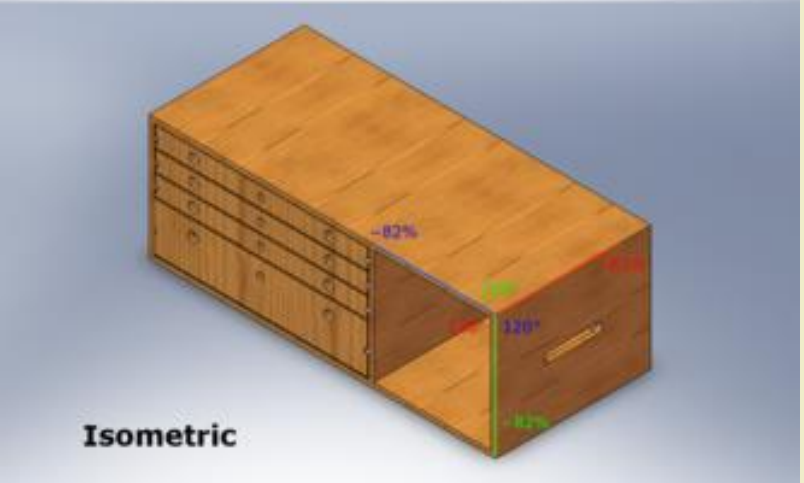
1. Projectors must perpendicular to the projection plane
2. Projection plane **does not need** to be parallel to one of the principal faces of the object



Trimetric



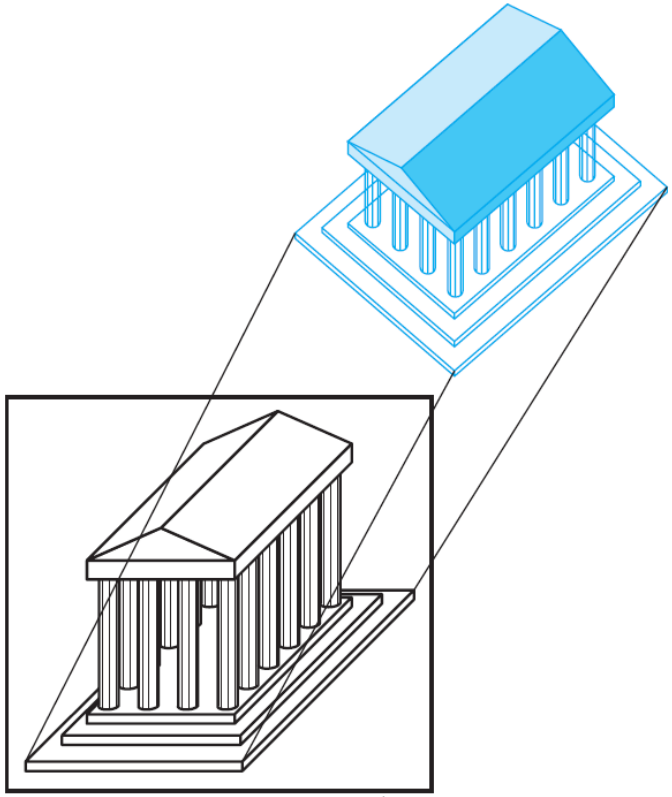
Dimetric

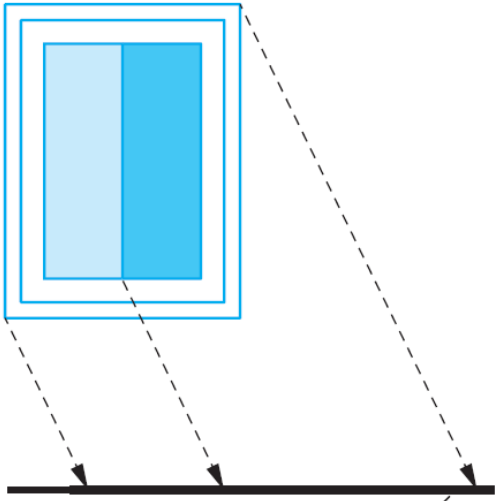


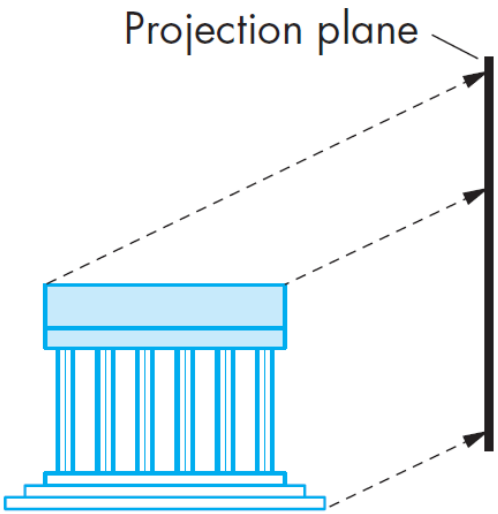
Isometric

Oblique Projections

1. Projectors can form **any** angle to the projection plane
2. Projection plane **does not need** to be parallel to one of the principal faces of the object

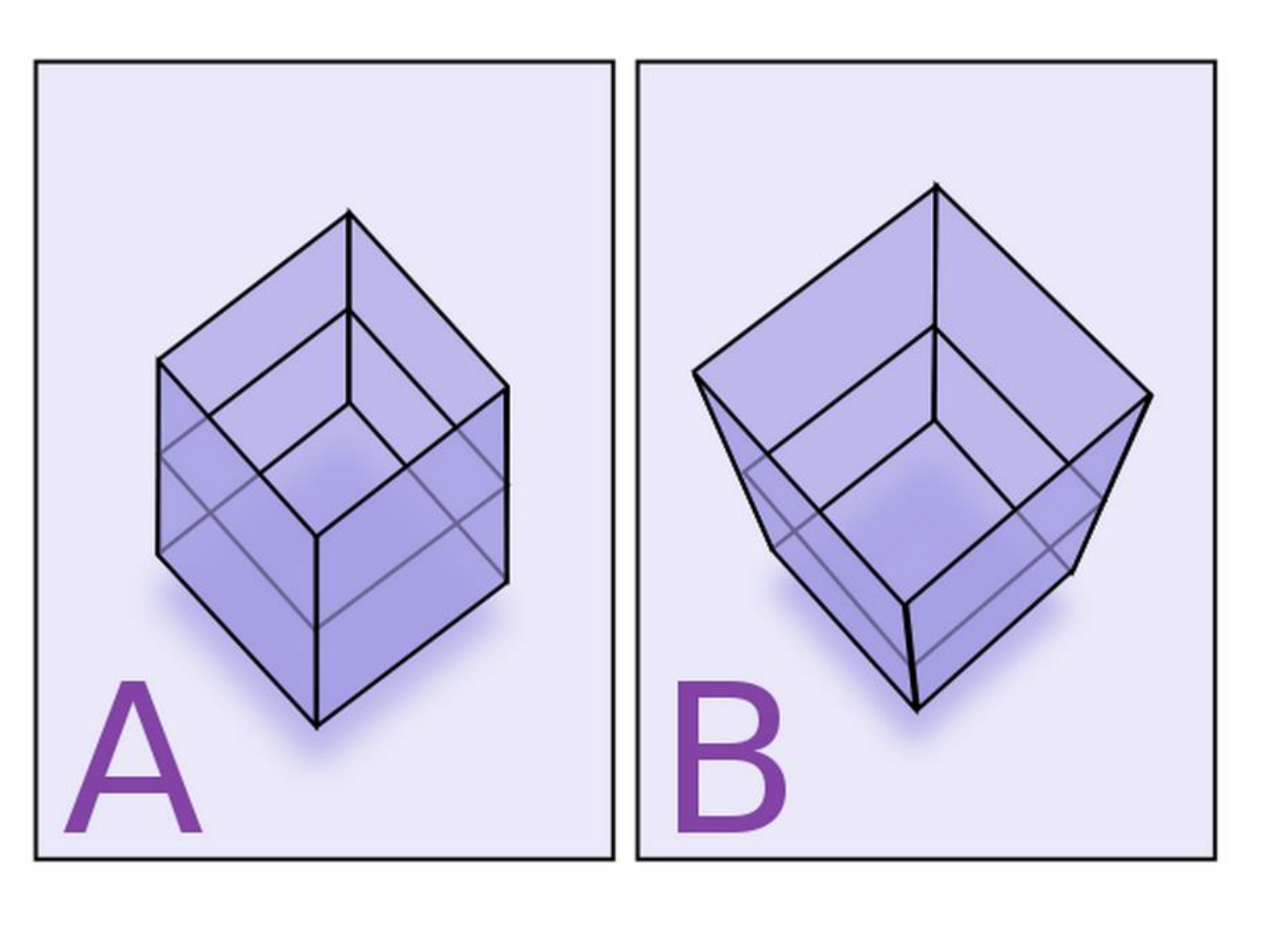

Projection plane


Projection plane


Projection plane

PERSPECTIVE PROJECTION

Foreshortening



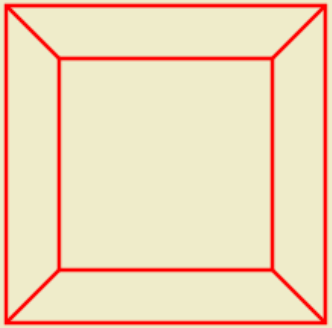
Common Perspective Projection Types

One Point Perspective

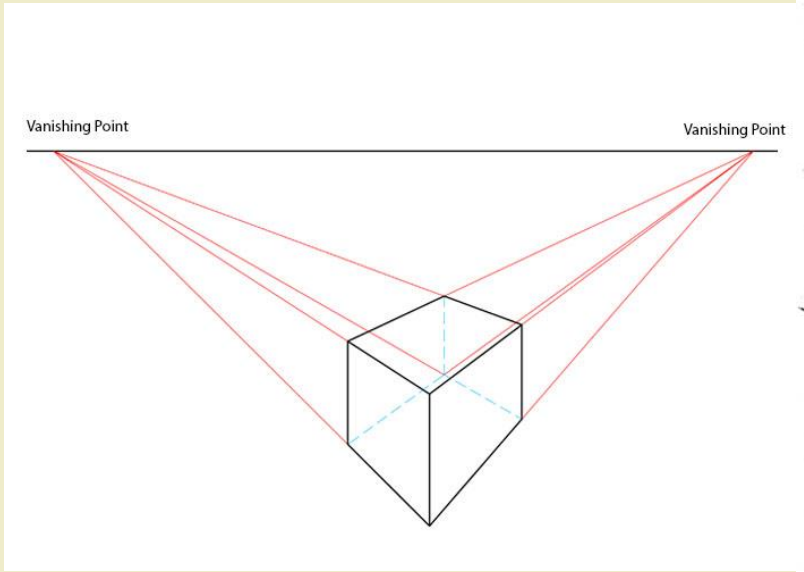
Two Point Perspective

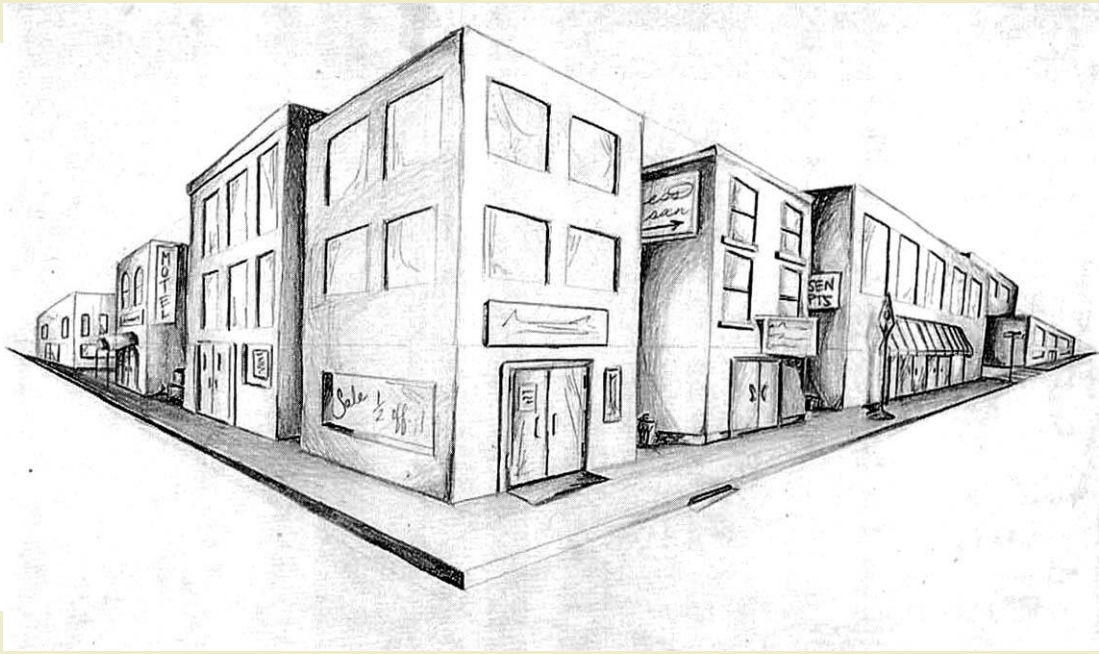
Three Point Perspective

One Point Perspective

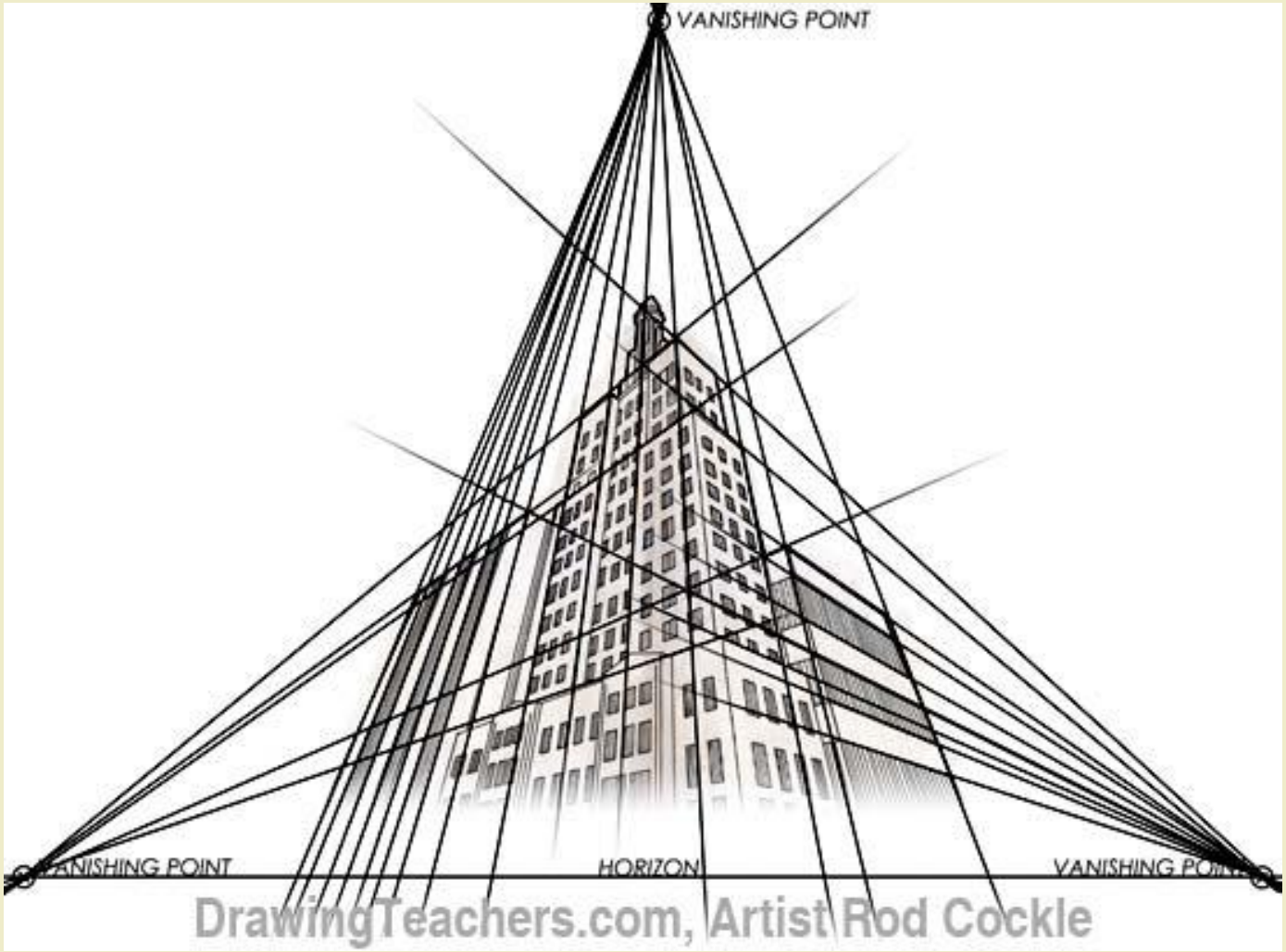


Two Point Perspective





Three Point Perspective



IMPLEMENTING VIEWING

Vertex Transformation

Vertices undergo a series of transformations before it is shown on the screen

Vertex Transformation

These transformations are essentially

Change of Frames

MODEL TRANSFORM

Model Transform

Object vertices are defined on its own object coordinate system (frame)

Model Transform

Placement of an model/object

in the world

Object frame to the World frame

Model Transform

Utilizes the **model matrix**

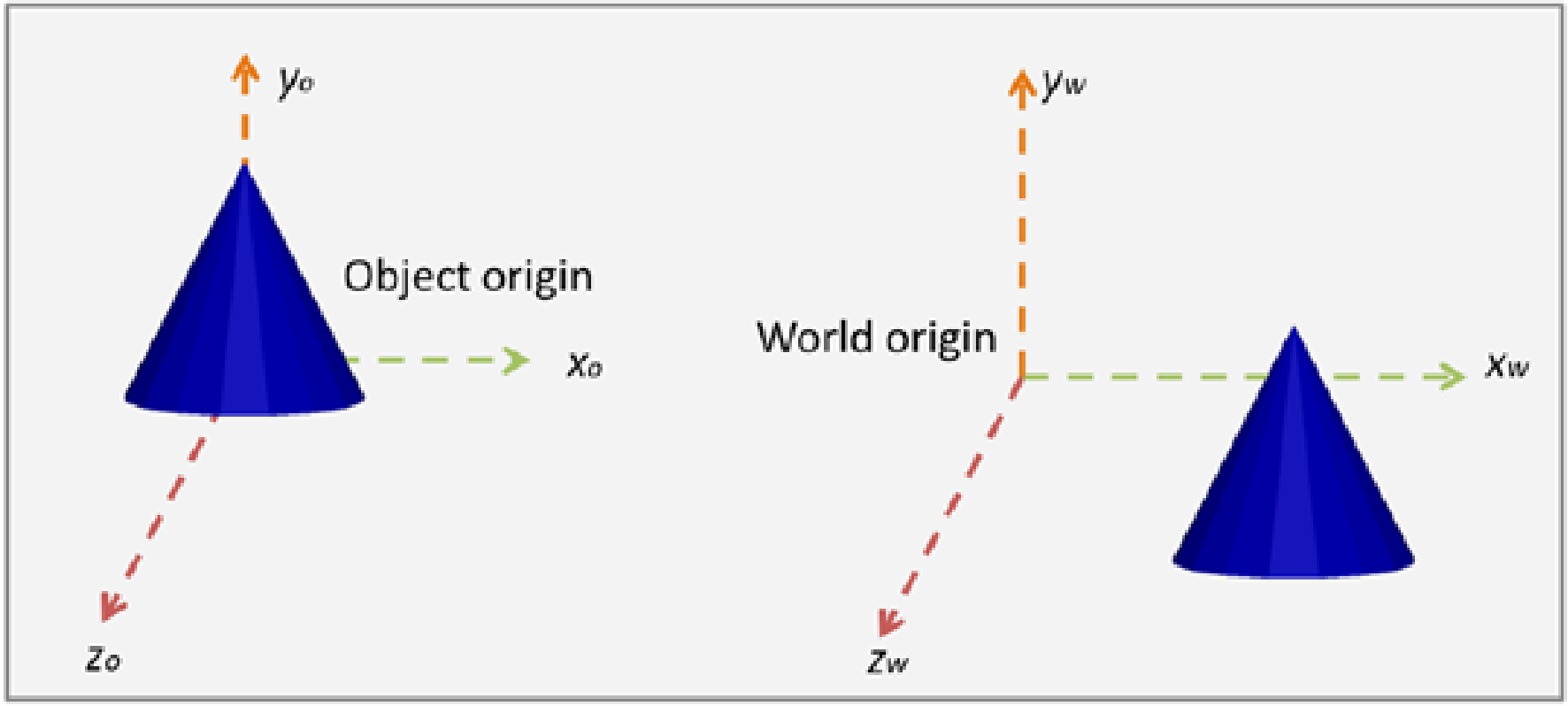
Model Matrix

Matrix that represents the **change of frame**

from object frame to world frame

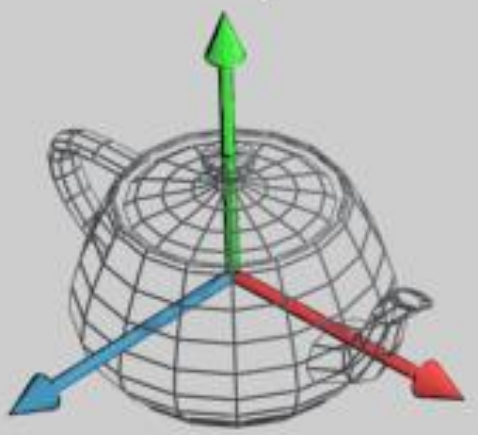
Using different affine transformations in 3D

Model Transform

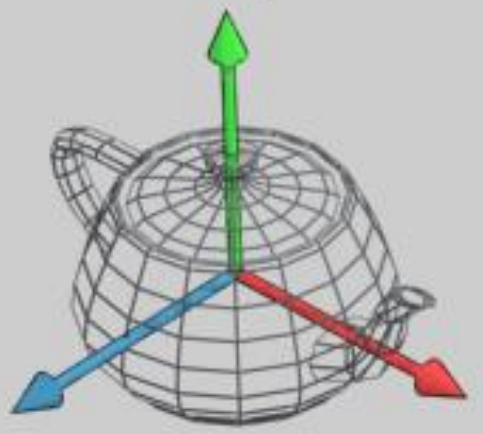


The object has not moved. The model transform assigns coordinates that are shared by all objects: the world coordinates.

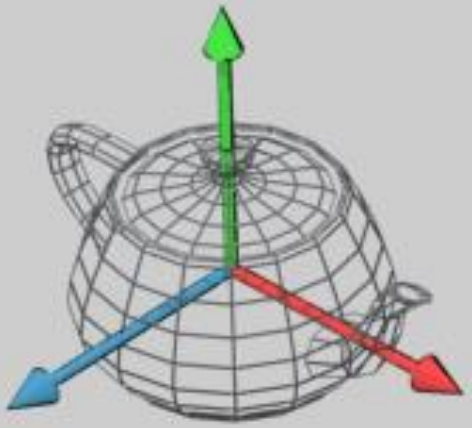
Model Space

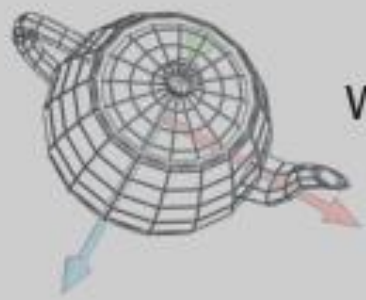


Model Space

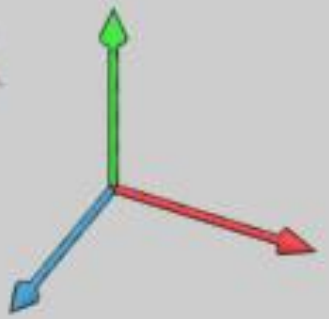


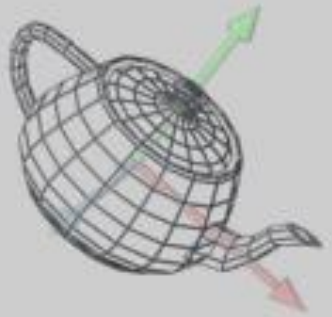
Model Space





World Space





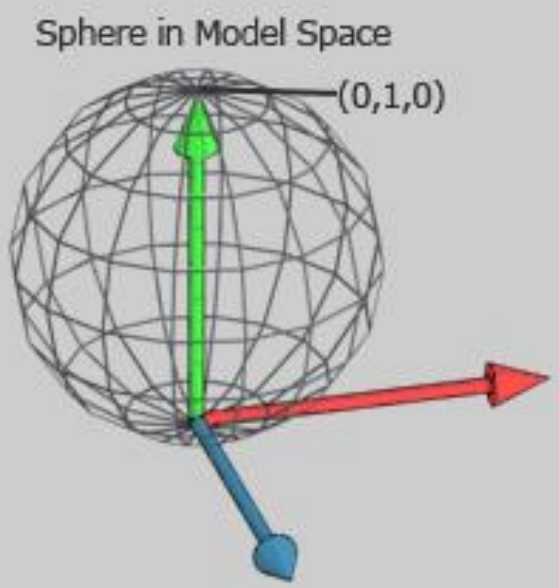
Model Matrix

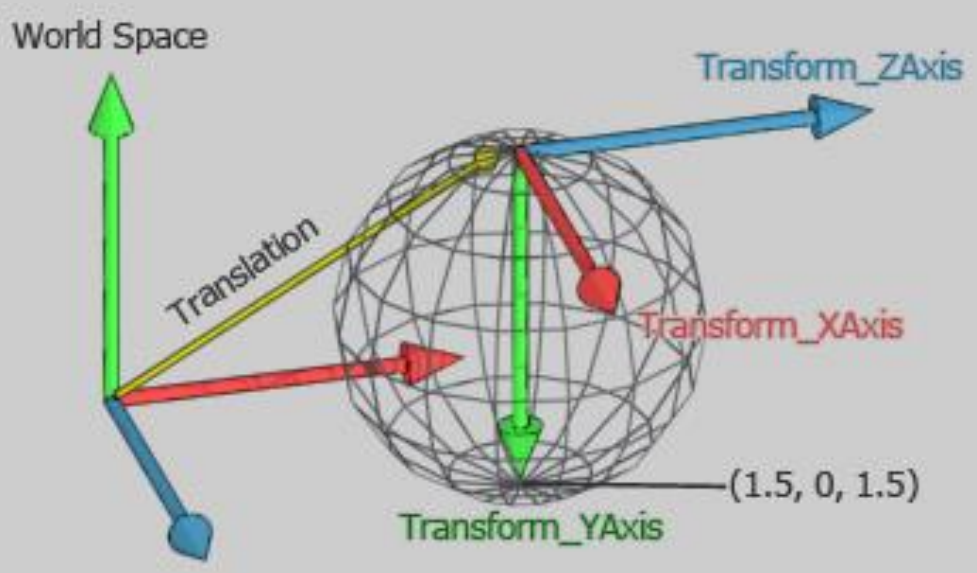
Translation or Rotation or Scaling

Using different affine transformations in 3D

Model Matrix

$$\begin{bmatrix} 0 & 0 & 1 & 1.5 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 1.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$


Sphere in Model Space
(0,1,0)


World Space
Transform_ZAxis
Translation
Transform_XAxis
(1.5, 0, 1.5)
Transform_YAxis

Model Transform

$$\begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \\ w_{world} \end{bmatrix} = ModelMatrix \times \begin{bmatrix} x_{object} \\ y_{object} \\ z_{object} \\ w_{object} \end{bmatrix}$$

$$\begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \\ w_{world} \end{bmatrix} = M \begin{bmatrix} x_{object} \\ y_{object} \\ z_{object} \\ w_{object} \end{bmatrix}$$

VIEW TRANSFORM

View Transform

Emulates where the **camera/eye** is located

Shifts the world origin to the view origin

View Transform

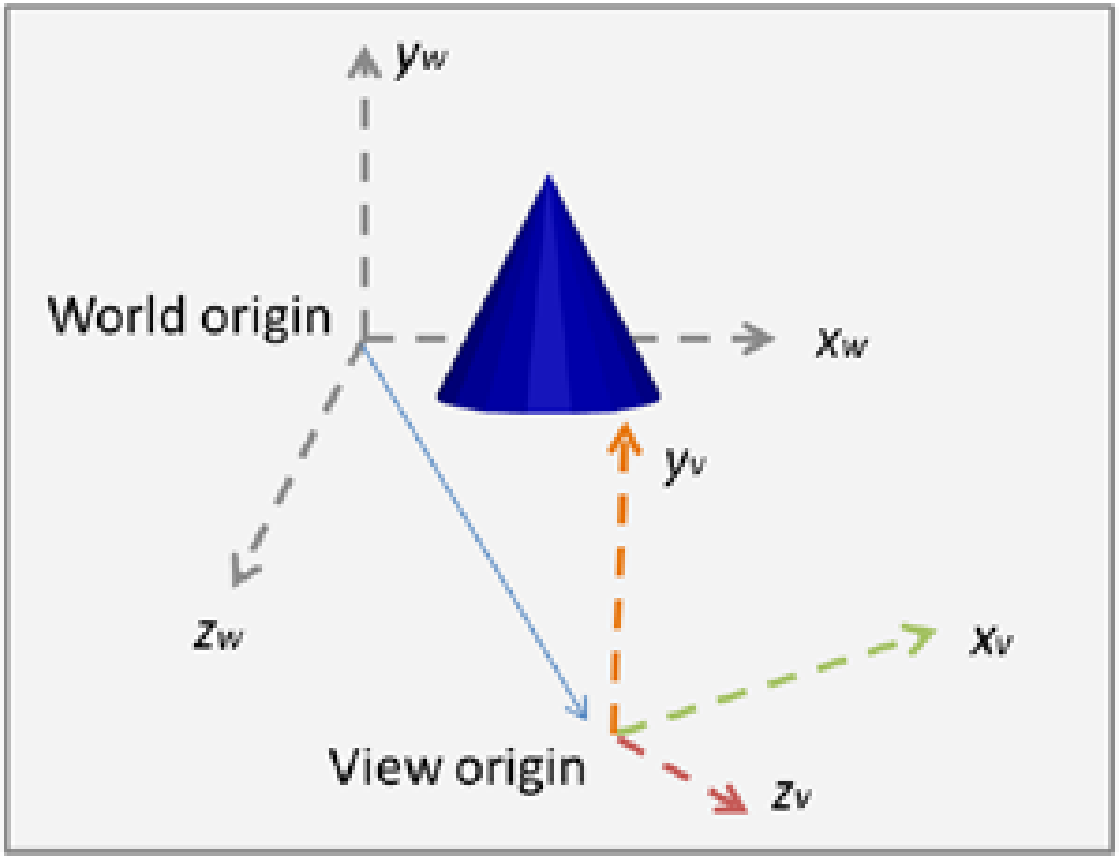
View origin is where the eye or the camera is located with respect to the world

From world frame to view frame

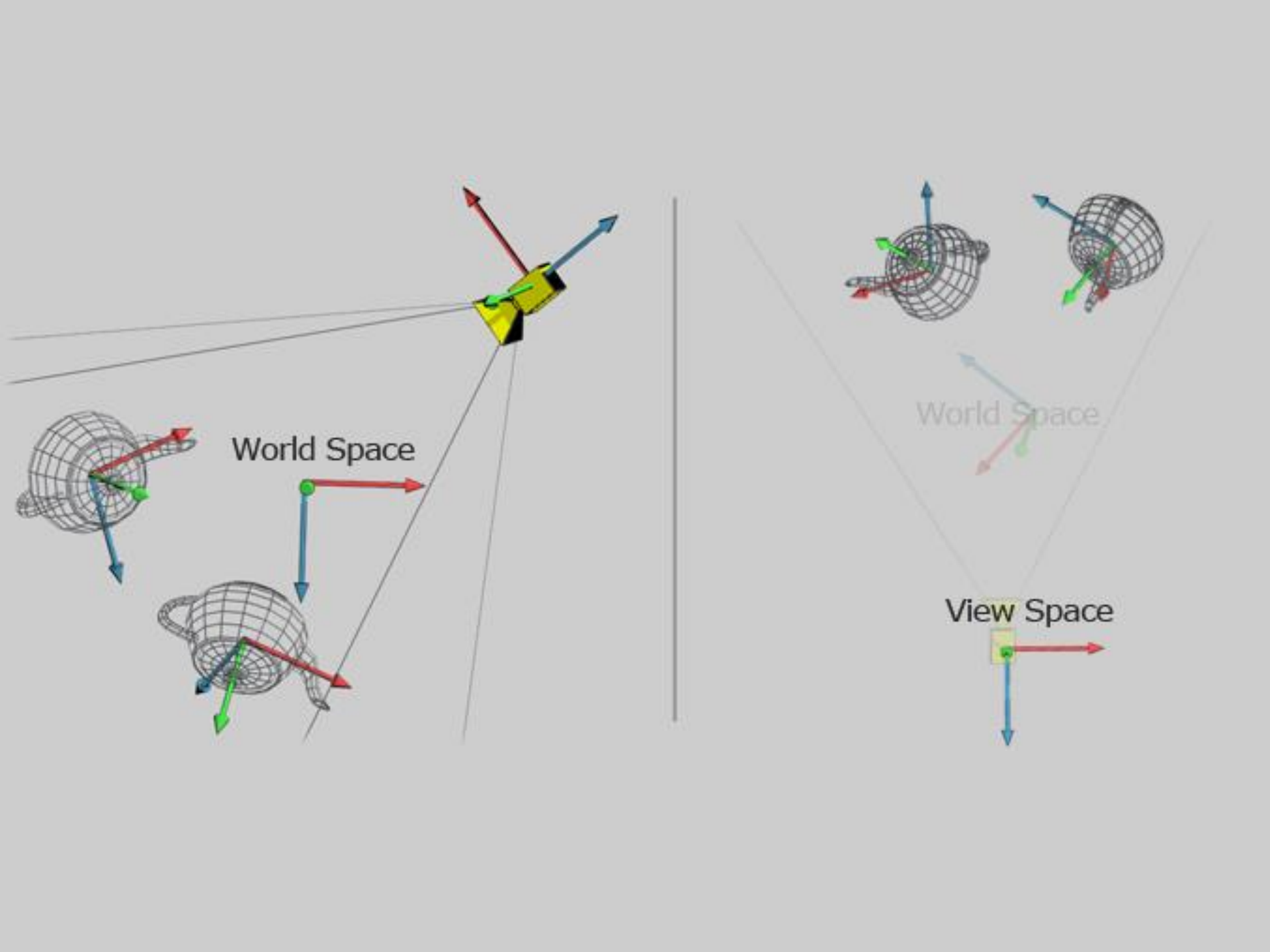
View Transform

Utilizes the **view matrix**

View Transform



The view transform moves the origin of the world to the coordinates of the view. This is where our *camera* is located.


World Space
World Space
View Space

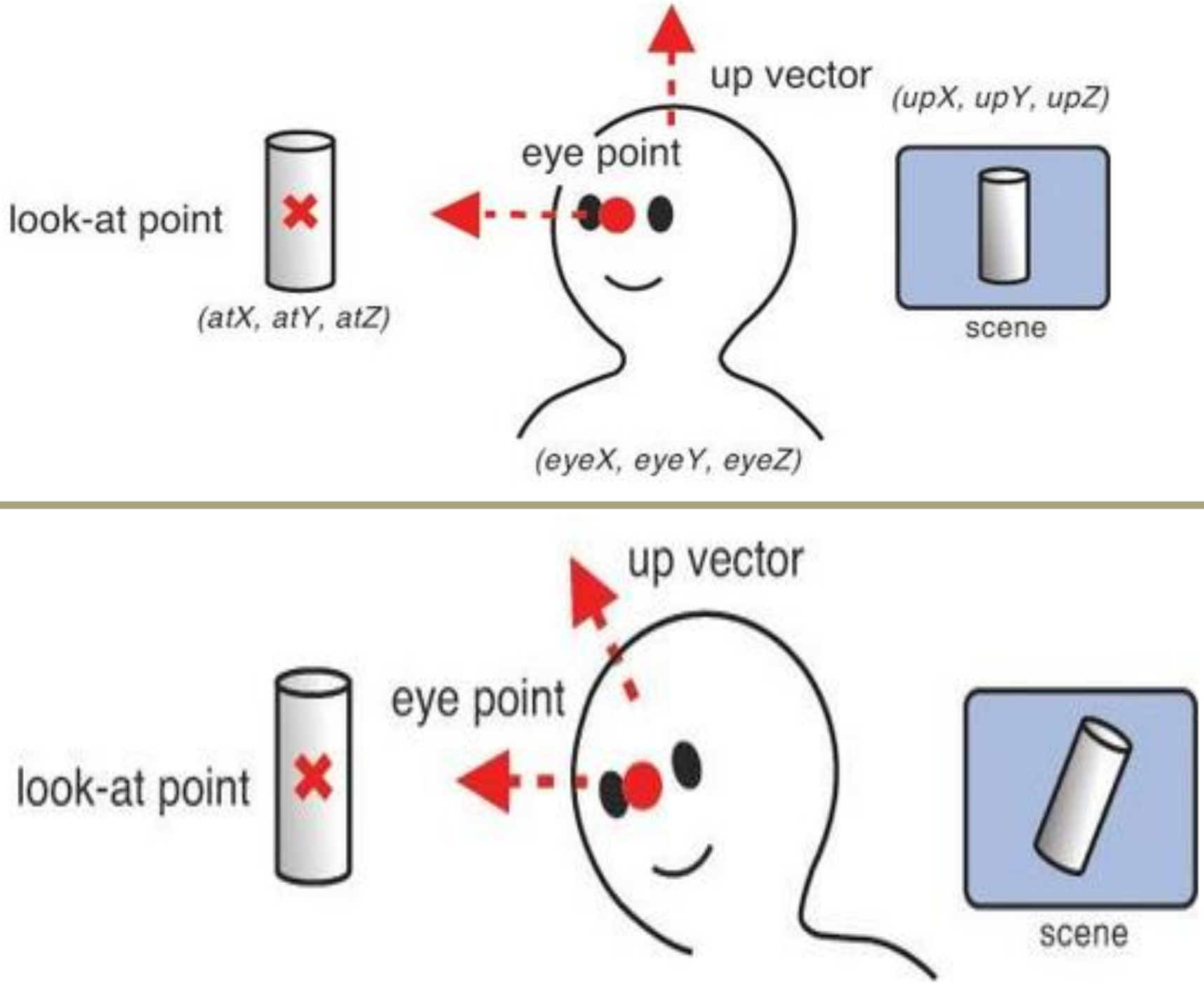
View Matrix Creation

3 parameters needed:

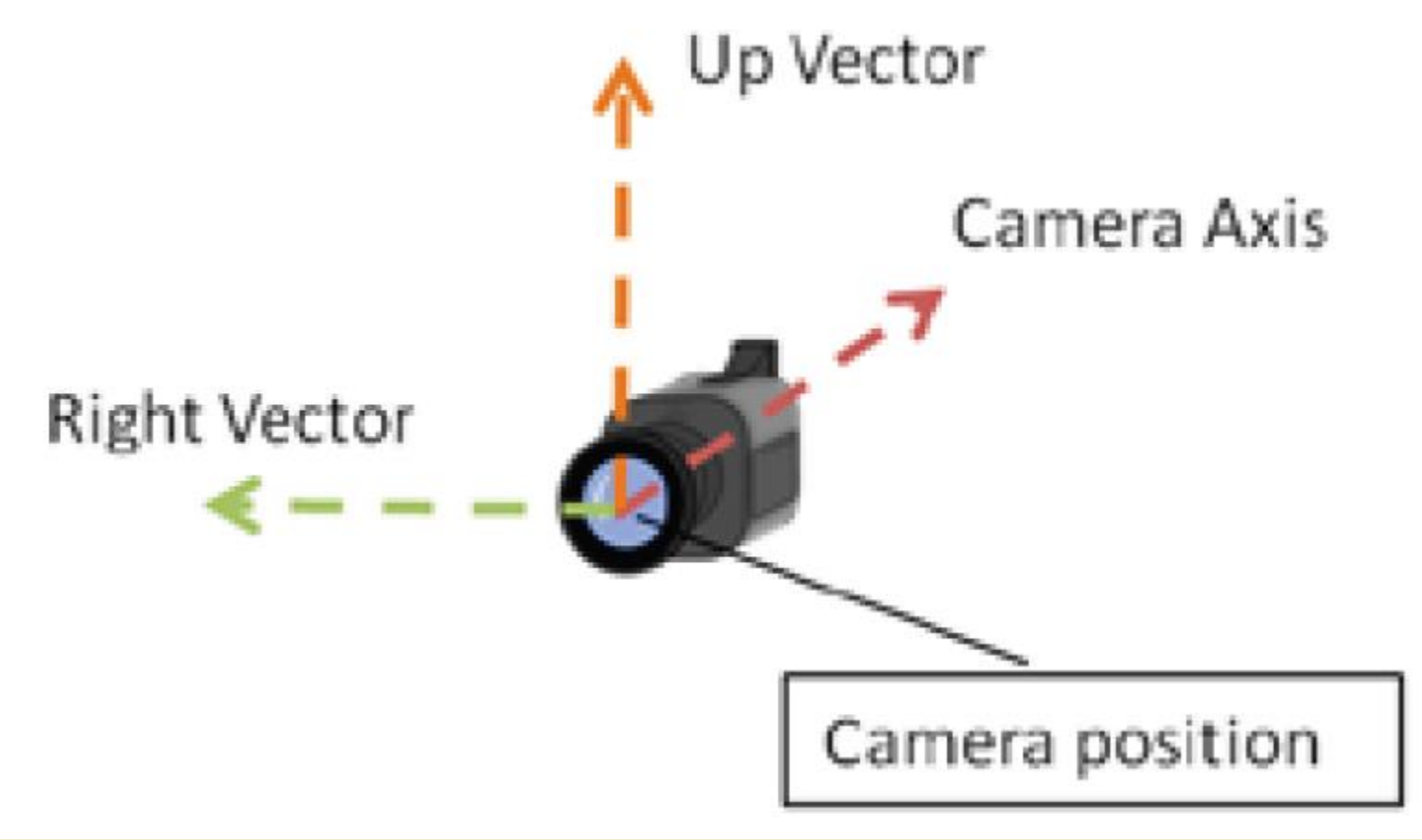
Look at point (A)

Camera/Eye point (E)

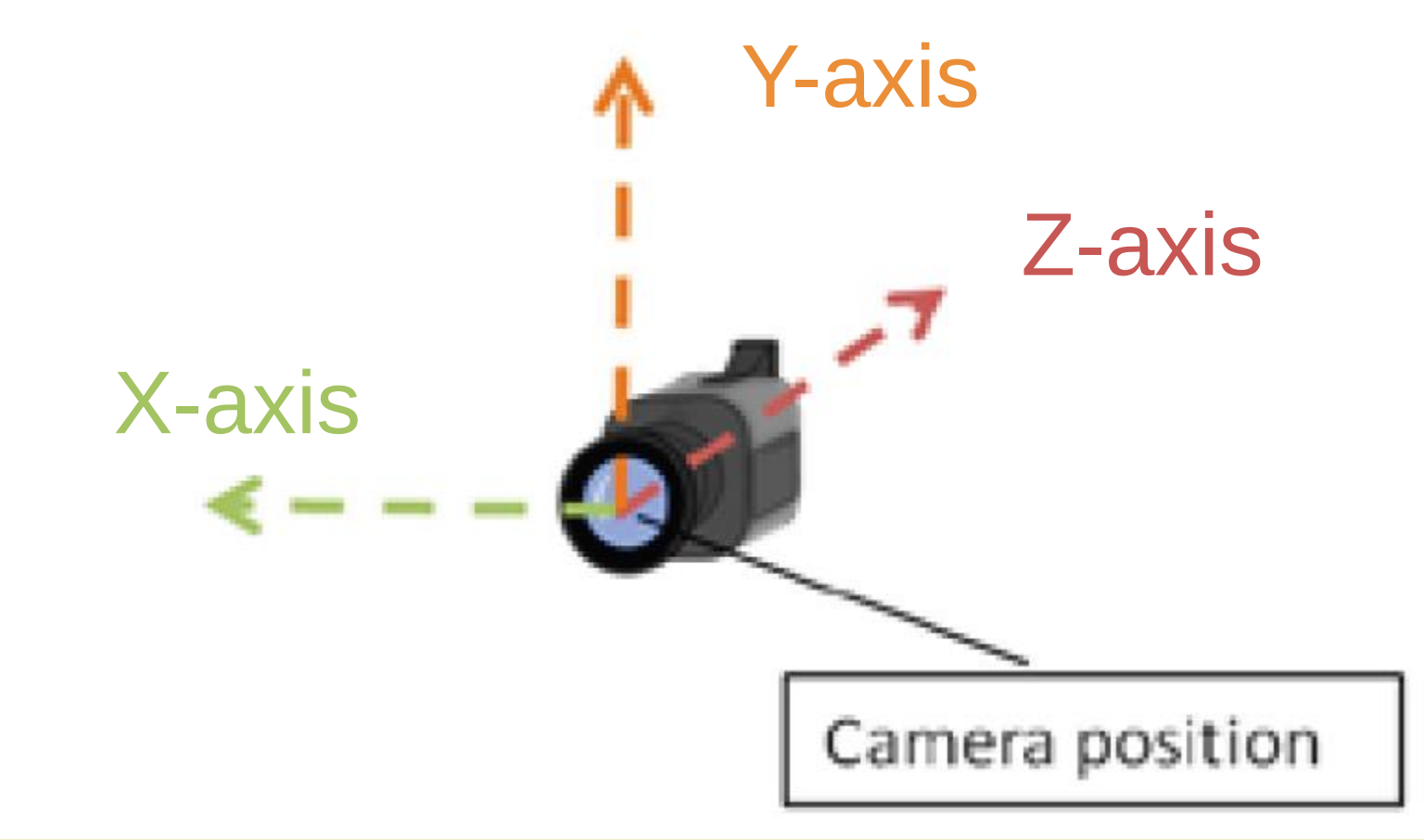
Up Vector (u)


up vector
(upX, upY, upZ)
eye point
look-at point
(atX, atY, atZ)
scene
(eyeX, eyeY, eyeZ)
up vector
eye point
look-at point
scene

Representing View Matrix (Frames)



Representing View Matrix (Frames)



View Matrix Creation

Compute for the normalized/unit z-axis

$$\hat{z} = normal(E - A)$$

View Matrix Creation

Compute for the normalized/unit x-axis

$$\hat{x} = normal(u \times \hat{z})$$

View Matrix Creation

Compute for the normalized/unit x-axis

$$\hat{x} = normal(u \times \hat{z})$$

View Matrix Creation

Compute for the normalized/unit y-axis

$$\hat{y} = normal(\hat{z} \times \hat{x})$$

View Matrix Creation

Create the initial view matrix with the new basis vectors x, y, z

$$\begin{bmatrix} x_x & x_y & x_z & 0 \\ y_x & y_y & y_z & 0 \\ z_x & z_y & z_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

View Matrix Creation

Multiply the movement (translation) from world origin to view origin

$$\begin{bmatrix} x_x & x_y & x_z & 0 \\ y_x & y_y & y_z & 0 \\ z_x & z_y & z_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -E_x \\ 0 & 1 & 0 & -E_y \\ 0 & 0 & 1 & -E_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

View Matrix Creation

Multiply the movement (translation) from world origin to view origin

$$\begin{bmatrix} x_x & x_y & x_z & -(x \cdot E) \\ y_x & y_y & y_z & -(y \cdot E) \\ z_x & z_y & z_z & -(z \cdot E) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Final View Matrix

$$\begin{bmatrix} x_x & x_y & x_z & -(x_x E_x + x_y E_y + x_z E_z) \\ y_x & y_y & y_z & -(y_x E_x + y_y E_y + y_z E_z) \\ z_x & z_y & z_z & -(z_x E_x + z_y E_y + z_z E_z) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

View Transform

$$\begin{bmatrix} x_{view} \\ y_{view} \\ z_{view} \\ w_{view} \end{bmatrix} = ViewMatrix \times \begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \\ w_{world} \end{bmatrix}$$

$$\begin{bmatrix} x_{view} \\ y_{view} \\ z_{view} \\ w_{view} \end{bmatrix} = VM \begin{bmatrix} x_{object} \\ y_{object} \\ z_{object} \\ w_{object} \end{bmatrix}$$

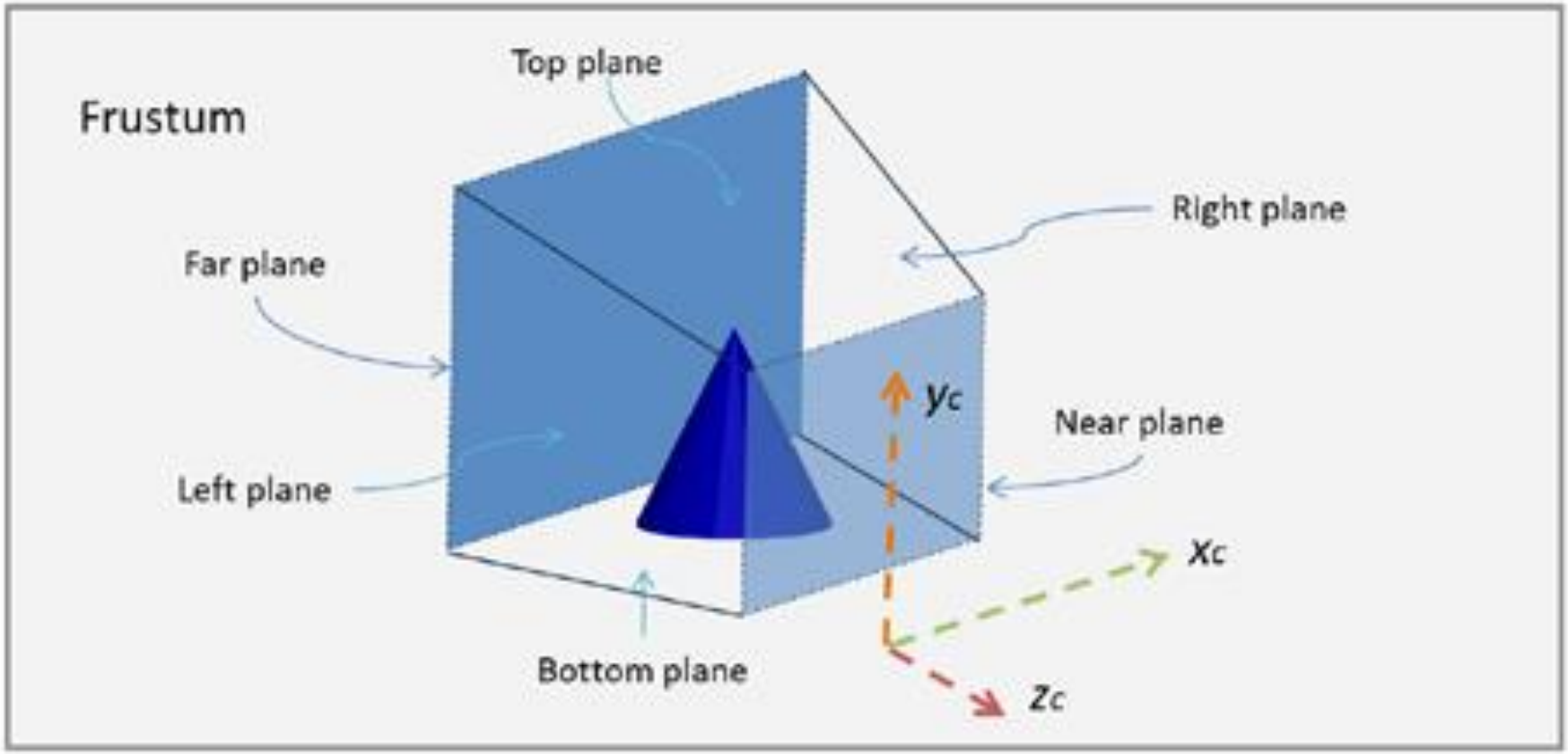
PROJECTION TRANSFORM

Projection Transform

Determines how much of the view space will be rendered

The region is represented by a square **frustum**

Projection Transform



The frustum determines which objects or portion of objects will be *clipped out* and discarded.

Projection Transform

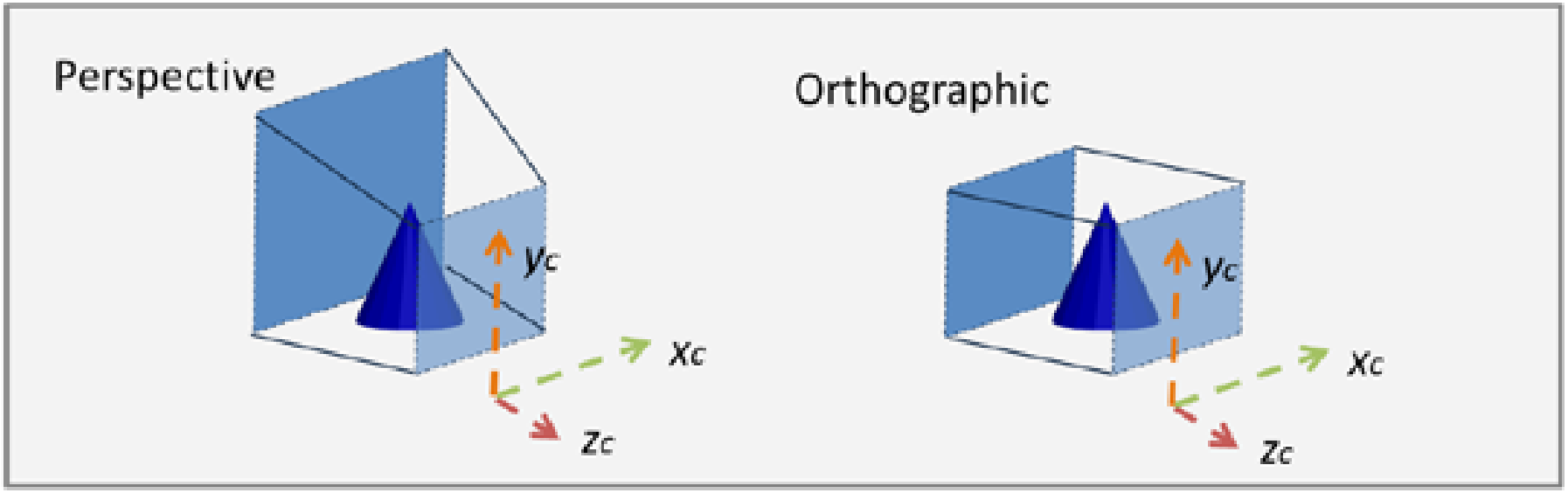
Vertices outside the frustum are *clipped out*

Projection Transform

Shape of frustum determines the type of projection

- Orthographic (Parallel): $farplane = nearplane$
- Perspective: $farplane \neq nearplane$

Frustum shape



The extent and shape of the frustum determines how much of the 3D view space Is mapped to the screen and the type of 3D to 2D projection that takes place.

Projection Transform

Utilizes the projection matrix

Projection Matrix Creation

Perspective Viewing (6 bounds specification)

near, far, top, bottom, left, right plane

Perspective Viewing: 6 bounds specification

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

Projection Matrix Creation

Perspective Viewing

(4 parameter specification)

Field of view, Aspect Ratio,

Near and Far Plane

Perspective Viewing: 4 parameter specification

$$\begin{bmatrix} \frac{1}{aspect \times \tan\left(\frac{fovy}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{fovy}{2}\right)} & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Projection Matrix Creation

Orthographic Viewing

(6 bounds specification)

near, far, top, bottom, left, right plane

Orthogonal Viewing: 6 parameter specification

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

Projection Transform

$$\begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{bmatrix} = ProjectionMatrix \times \begin{bmatrix} x_{view} \\ y_{view} \\ z_{view} \\ w_{view} \end{bmatrix}$$

$$\begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{bmatrix} = PVM \begin{bmatrix} x_{object} \\ y_{object} \\ z_{object} \\ w_{object} \end{bmatrix}$$

PERSPECTIVE DIVISION

Perspective Division

The viewing volume (**frustum**) is mapped into the near plane to produce 2D image

Perspective Division

Vertices are mapped to the **normalized device coordinates (NDC)**

Coordinate system that is independent from hardware

Perspective Division

Vertices are mapped to the normalized device coordinates (NDC)

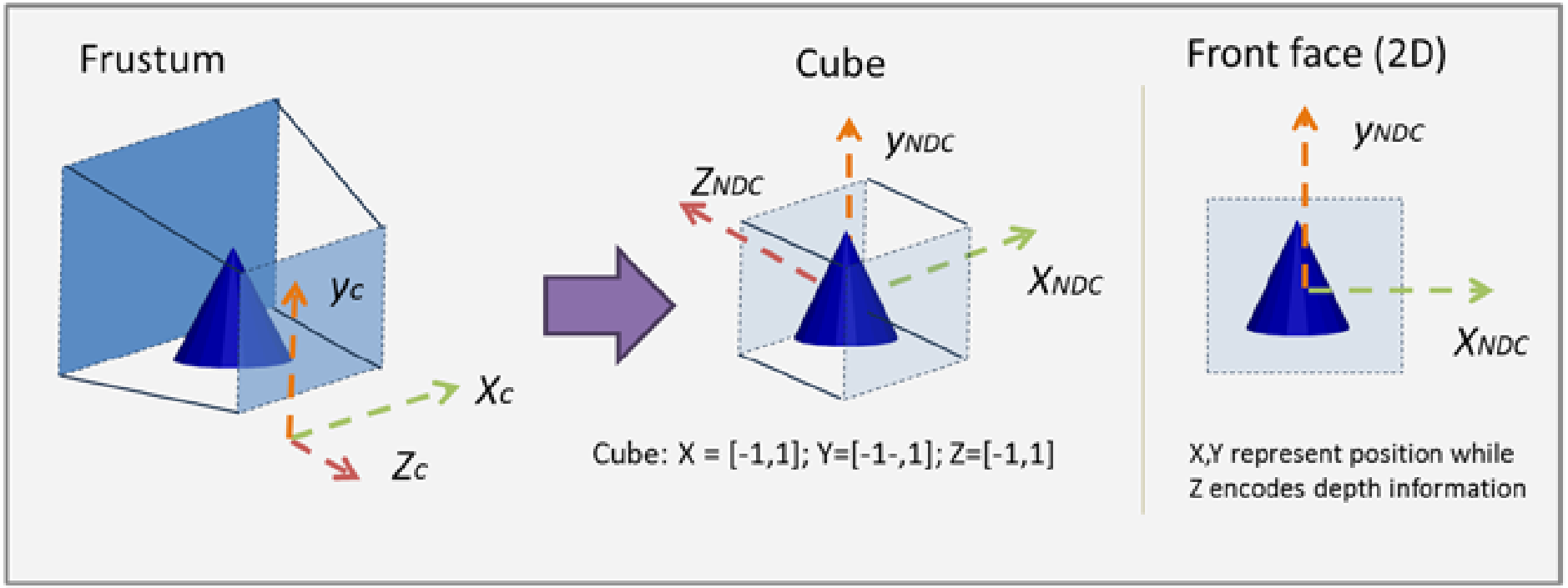
Range of values are from -1 to 1

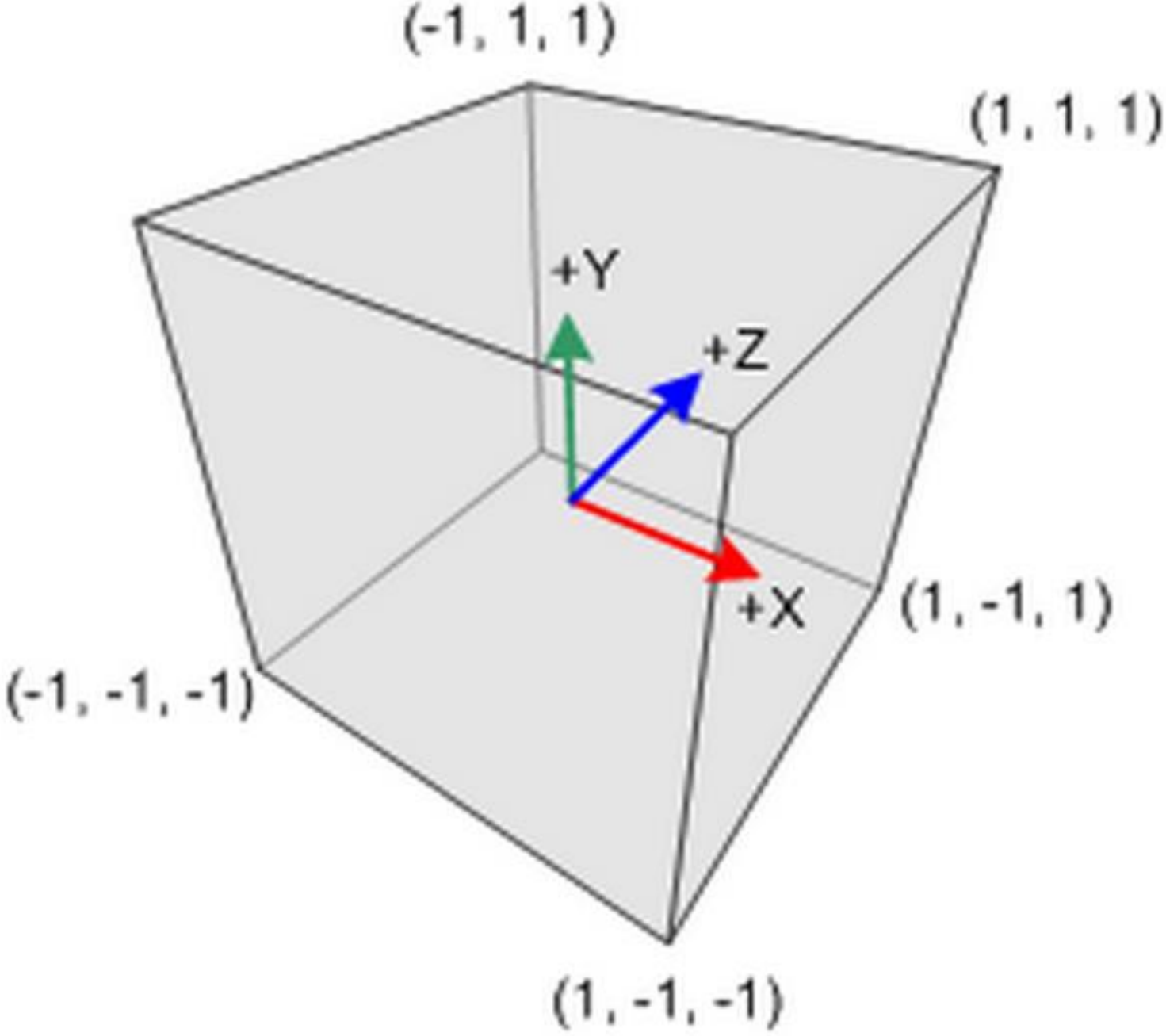
Perspective Division

Division of the clipping coordinates using the

w-component

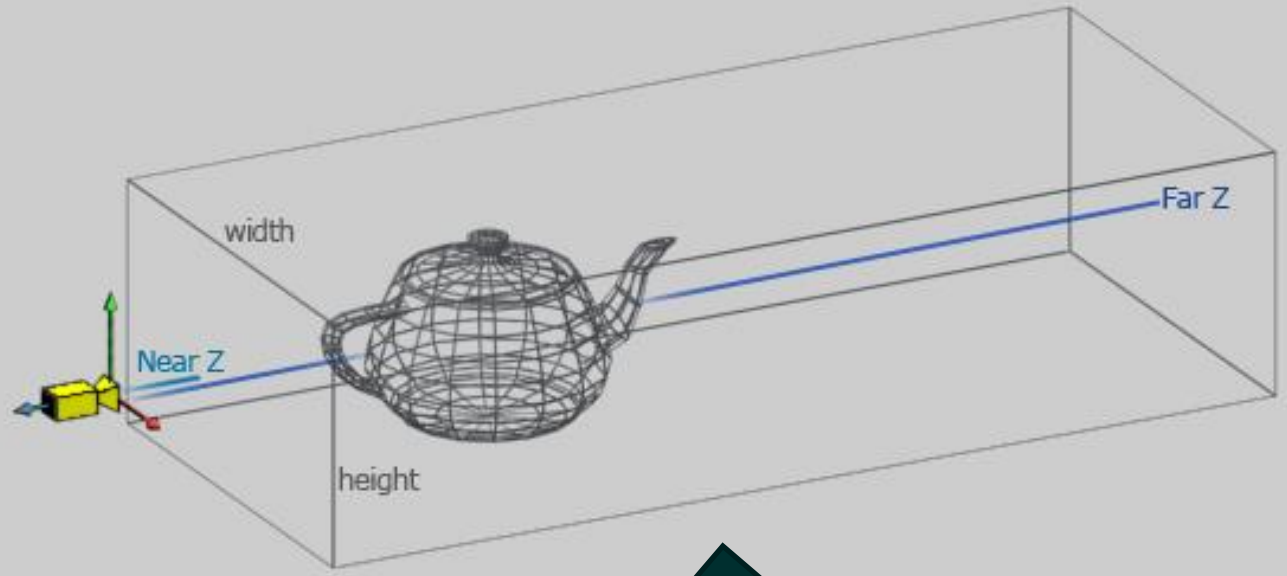
Normalized Device Coordinates

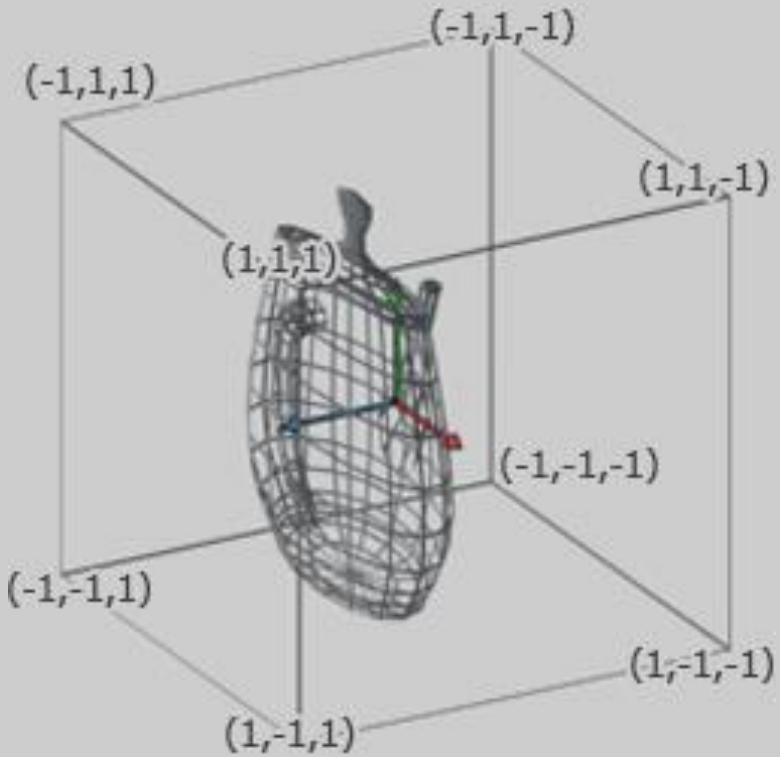


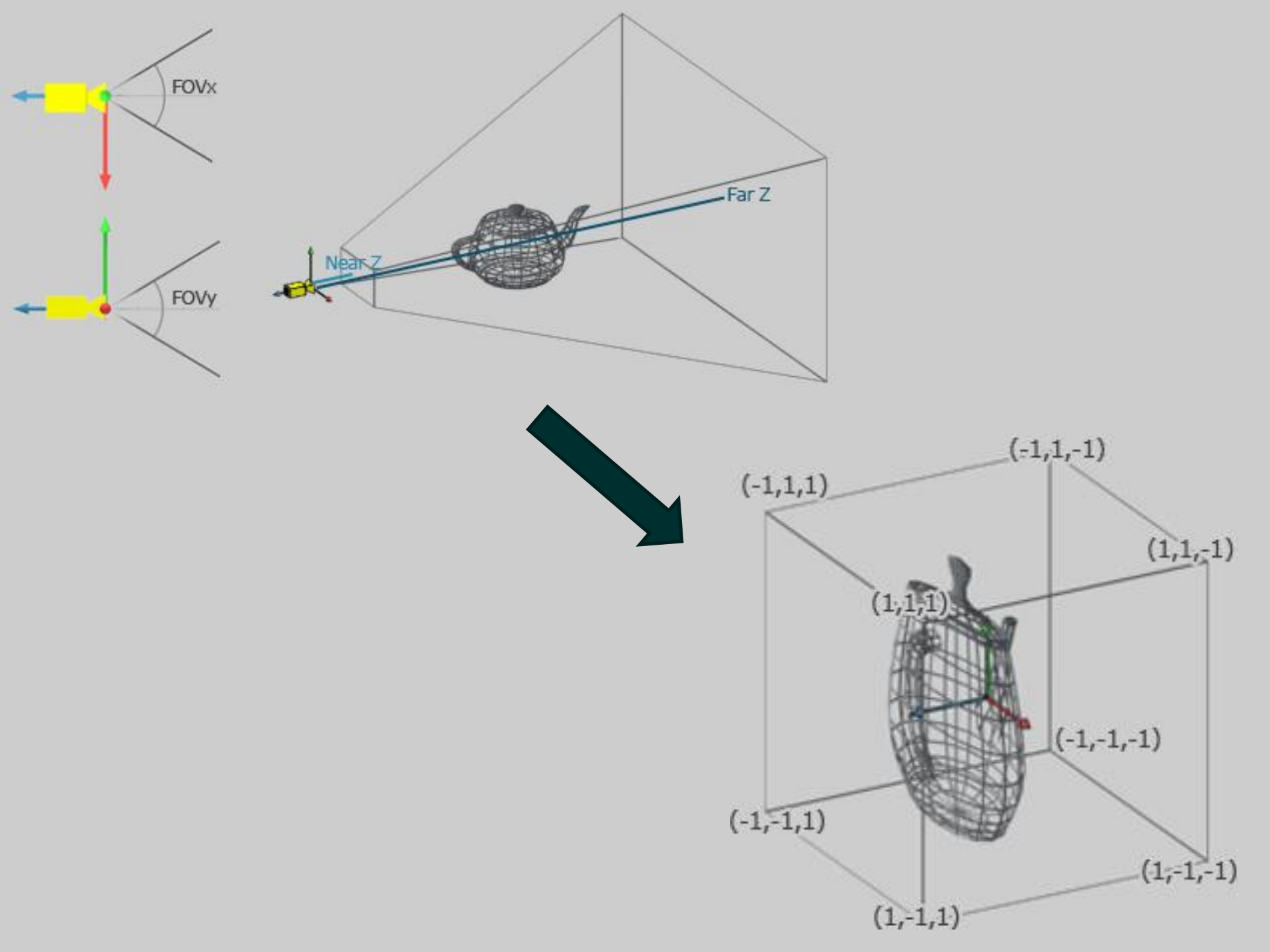

(-1, 1, 1)
(1, 1, 1)
+Y
+Z
+X
(1, -1, 1)
(-1, -1, -1)
(1, -1, -1)

Perspective Division

$$\begin{bmatrix} x_{ndc} \\ y_{ndc} \\ z_{ndc} \end{bmatrix} = \begin{bmatrix} x_{clip}/w_{clip} \\ y_{clip}/w_{clip} \\ z_{clip}/w_{clip} \end{bmatrix}$$


width
Far Z
Near Z
height


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


FOVx
FOVy
Near Z
Far Z
(-1,1,-1)
(-1,1,1)
(1,1,-1)
(1,1,1)
(-1,-1,-1)
(-1,-1,1)
(1,-1,-1)
(1,-1,1)

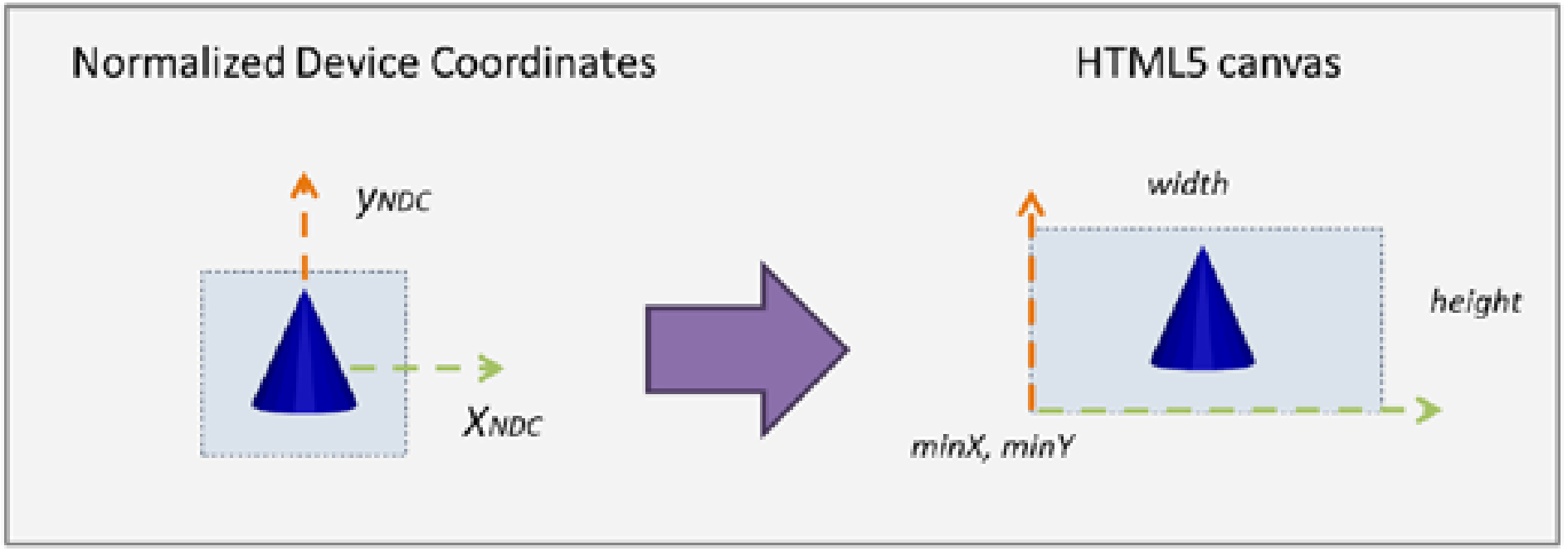
VIEWPORT TRANSFORM

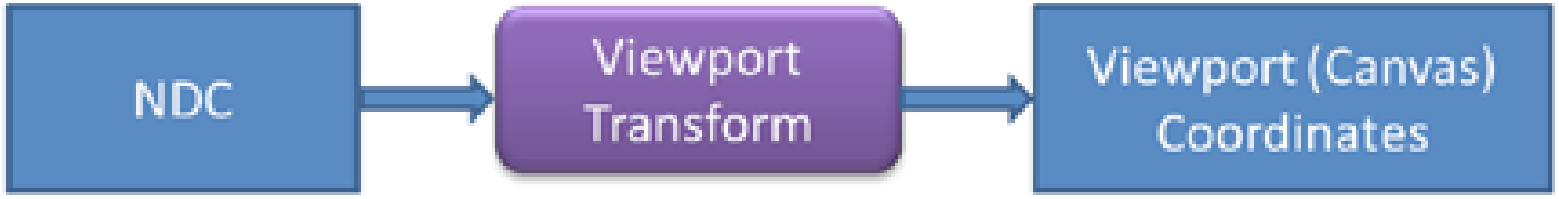
Viewport Transform

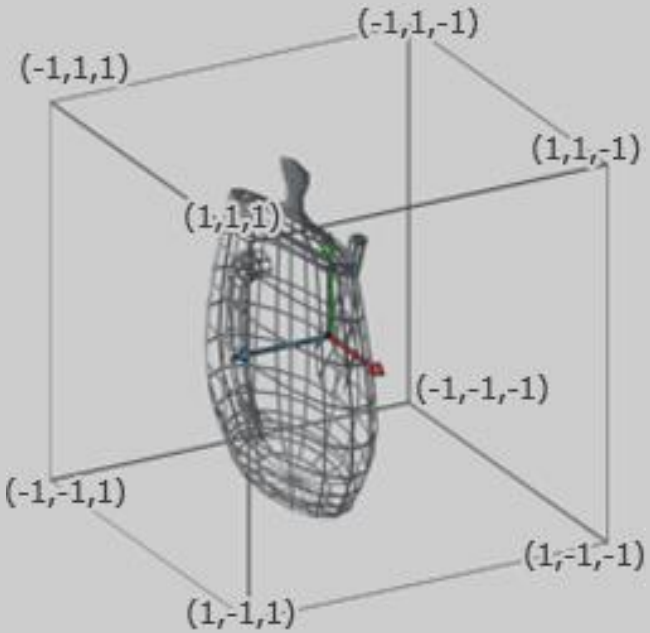
Normalized Device Coordinates are mapped

to **viewport coordinates**

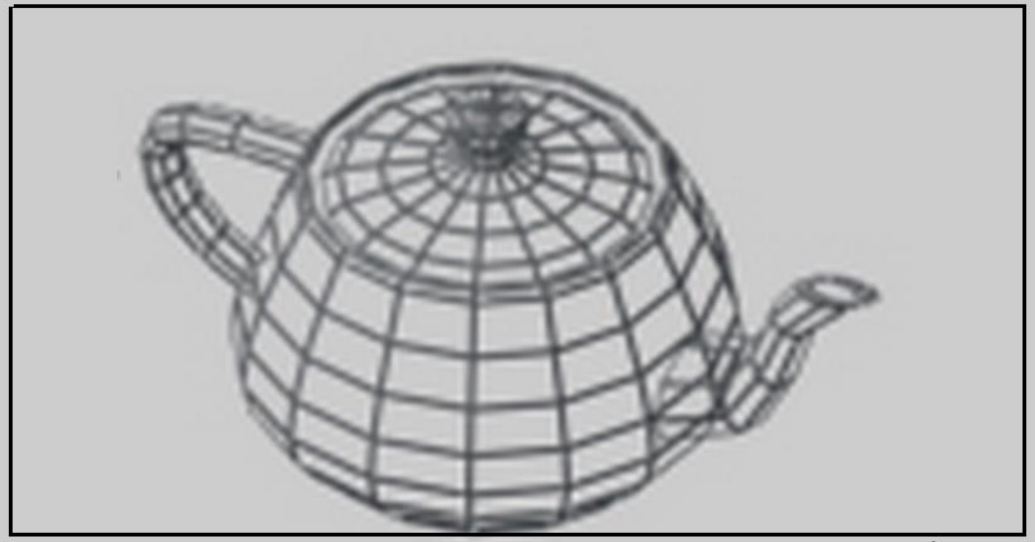
Viewport Transform





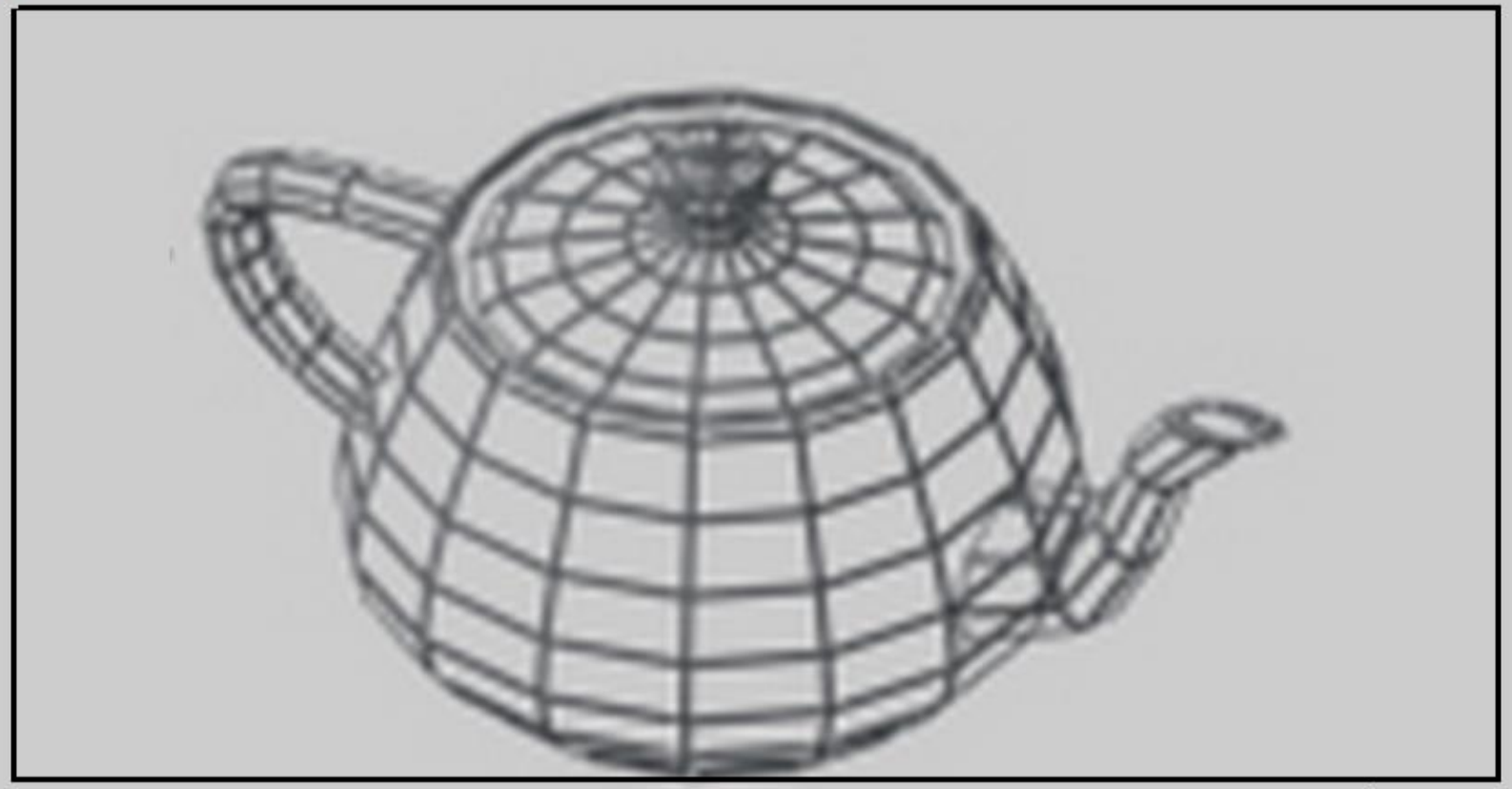


(x,y)



(width,height)

(x,y)



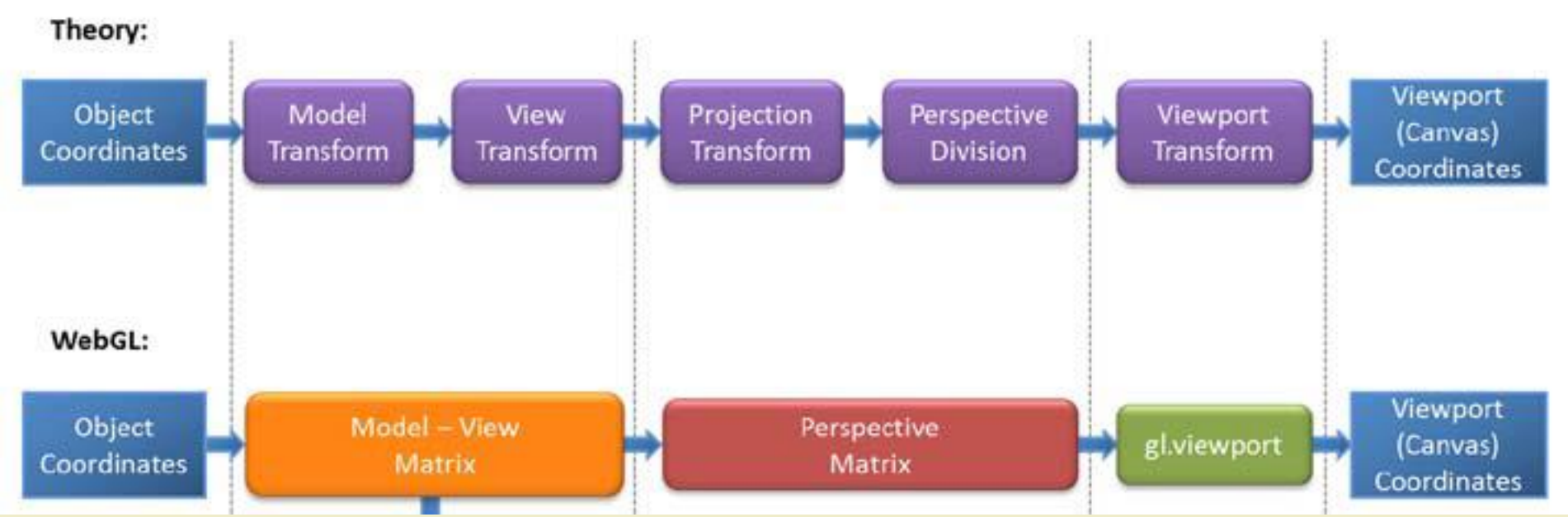
(width,height)

Viewport Transform

$$\begin{bmatrix} x_{window} \\ y_{window} \\ z_{window} \end{bmatrix} = \begin{bmatrix} \frac{width}{2} x_{ndc} + (x + \frac{width}{2}) \\ \frac{height}{2} x_{ndc} + (y + \frac{height}{2}) \\ \frac{f-n}{2} z_{ndc} + \frac{f+n}{2} \end{bmatrix}$$

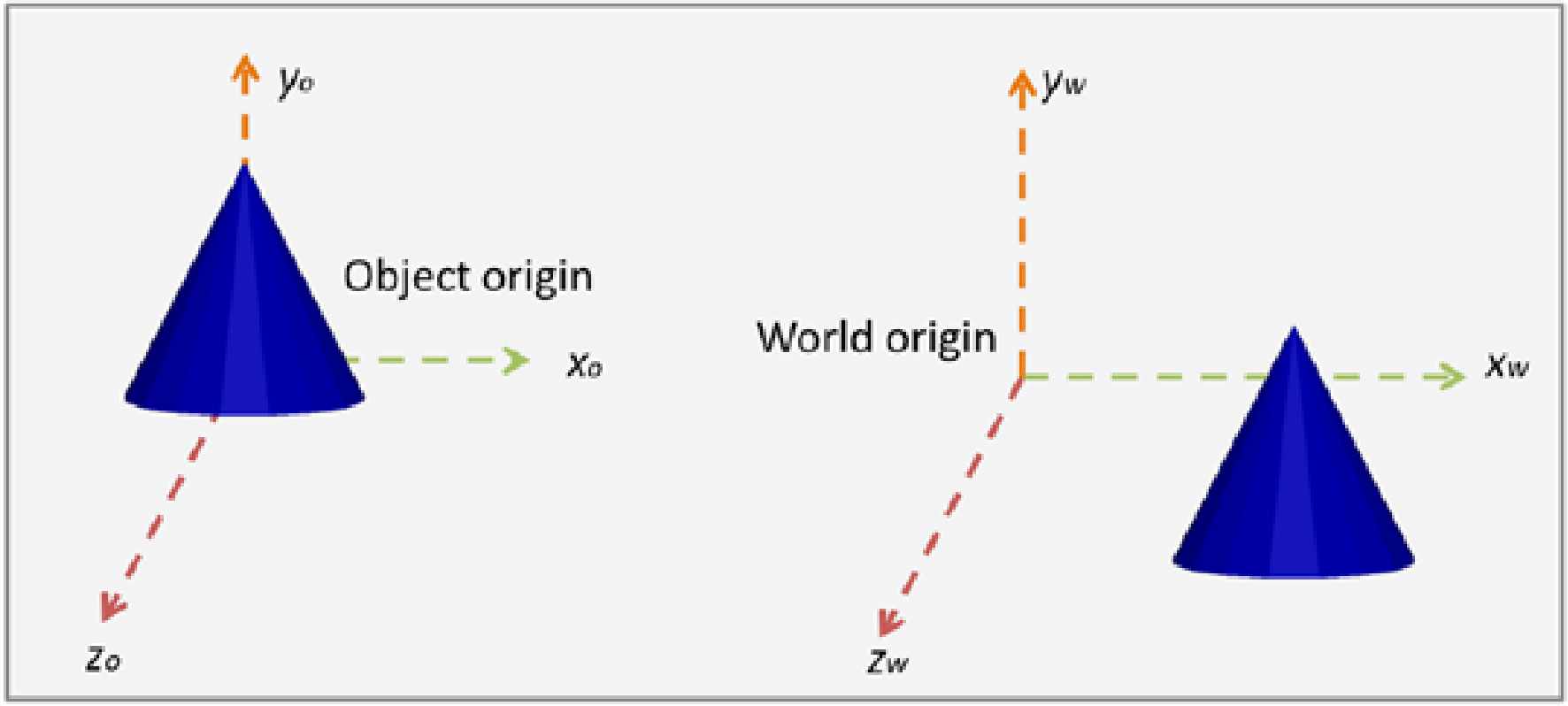
PROJECTION: THEORY TO APPLICATION

Projection: Theory to Application



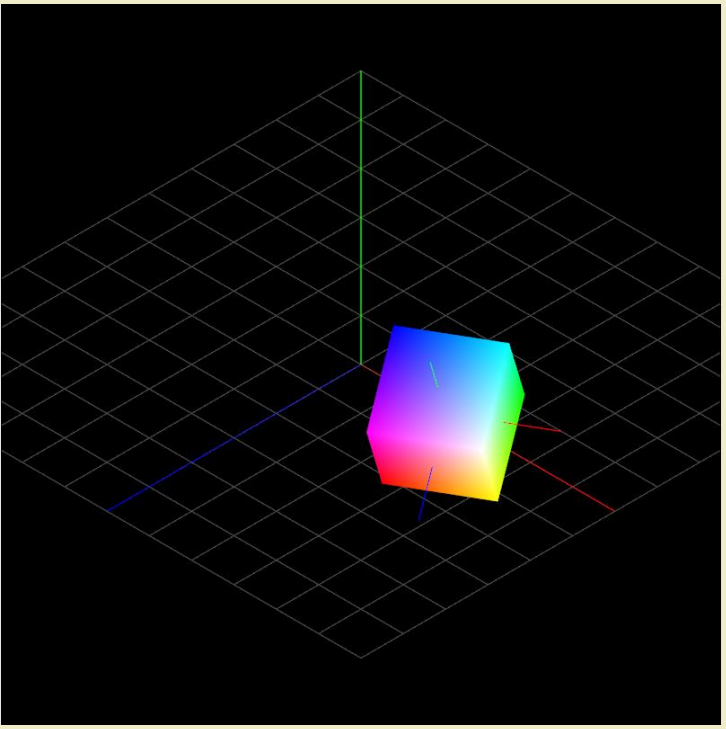
MODEL TRANSFORM

Model Transform



The object has not moved. The model transform assigns coordinates that are shared by all objects: the world coordinates.

Model Matrix



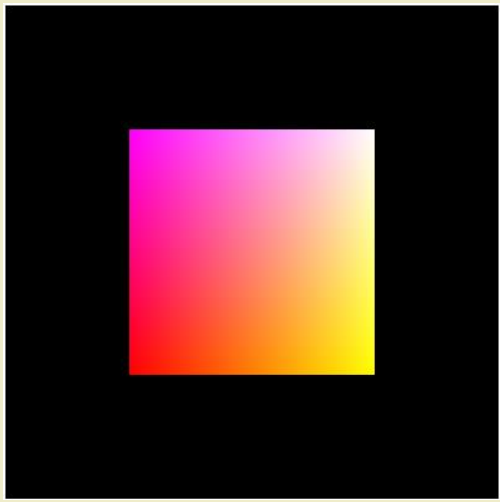
Cube Vertices (World Coordinates)

$$= \begin{bmatrix} 0.87 & 0.25 & 0.43 & 2 \\ 0 & 0.87 & -0.5 & 1 \\ -0.5 & 0.43 & 0.75 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ x}$$

Model Matrix

(Using Affine Transformations)

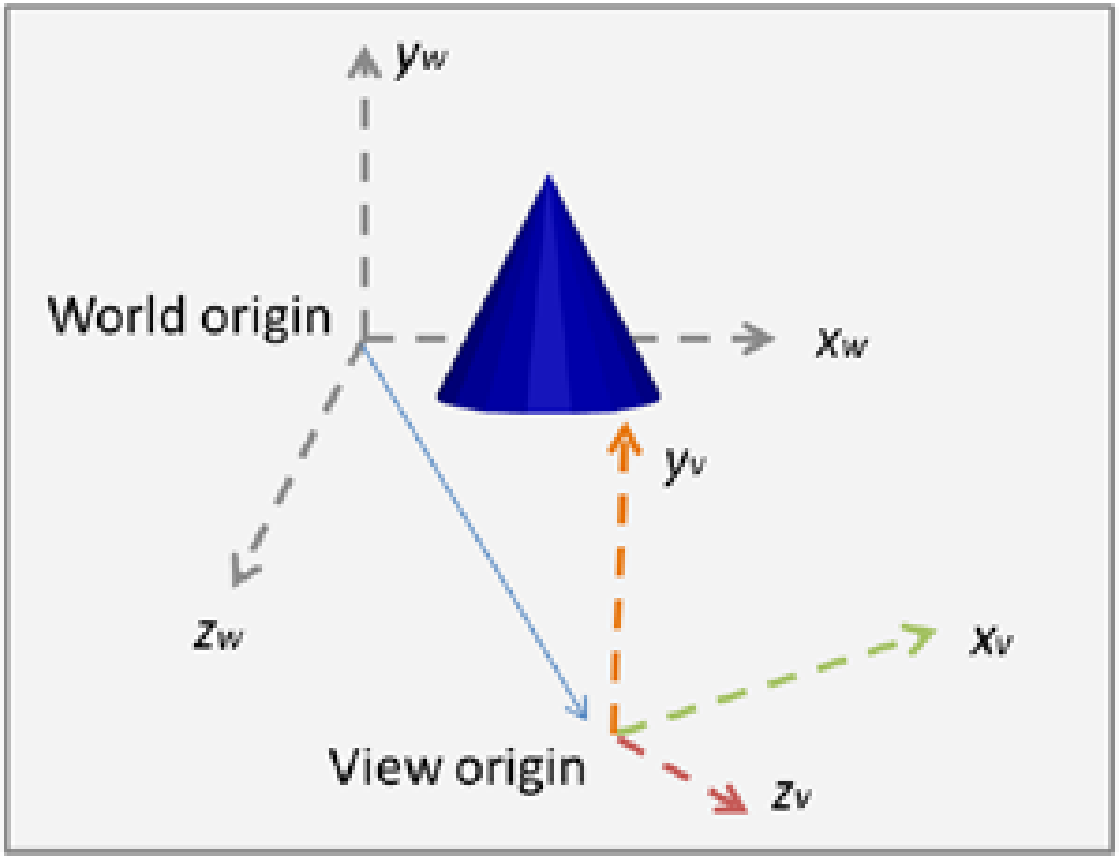
Translate by (2,1,1)
RotateX by 30
RotateY by 30



Cube Vertices (Object Coordinates)

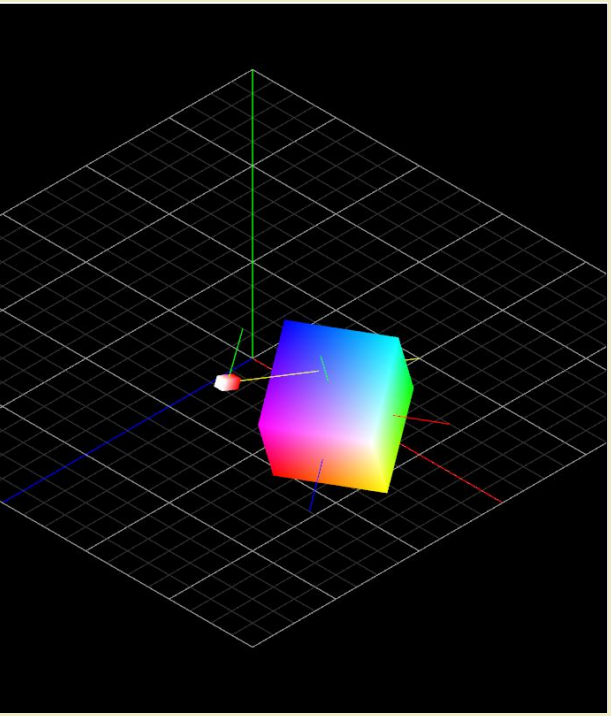
VIEW TRANSFORM

View Transform

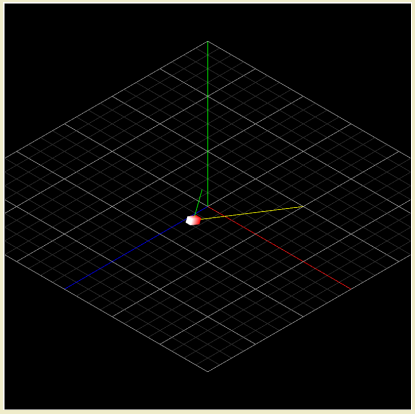


The view transform moves the origin of the world to the coordinates of the view. This is where our *camera* is located.

View Matrix

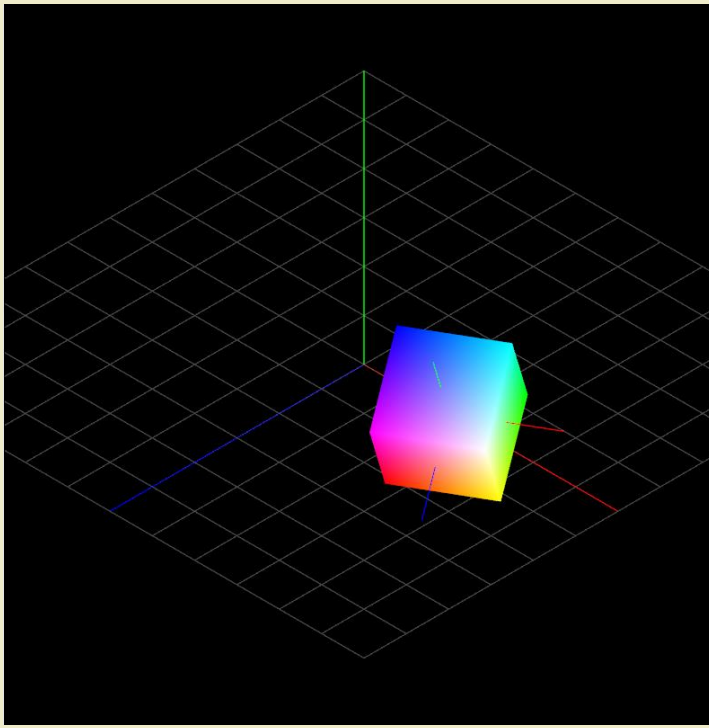


=



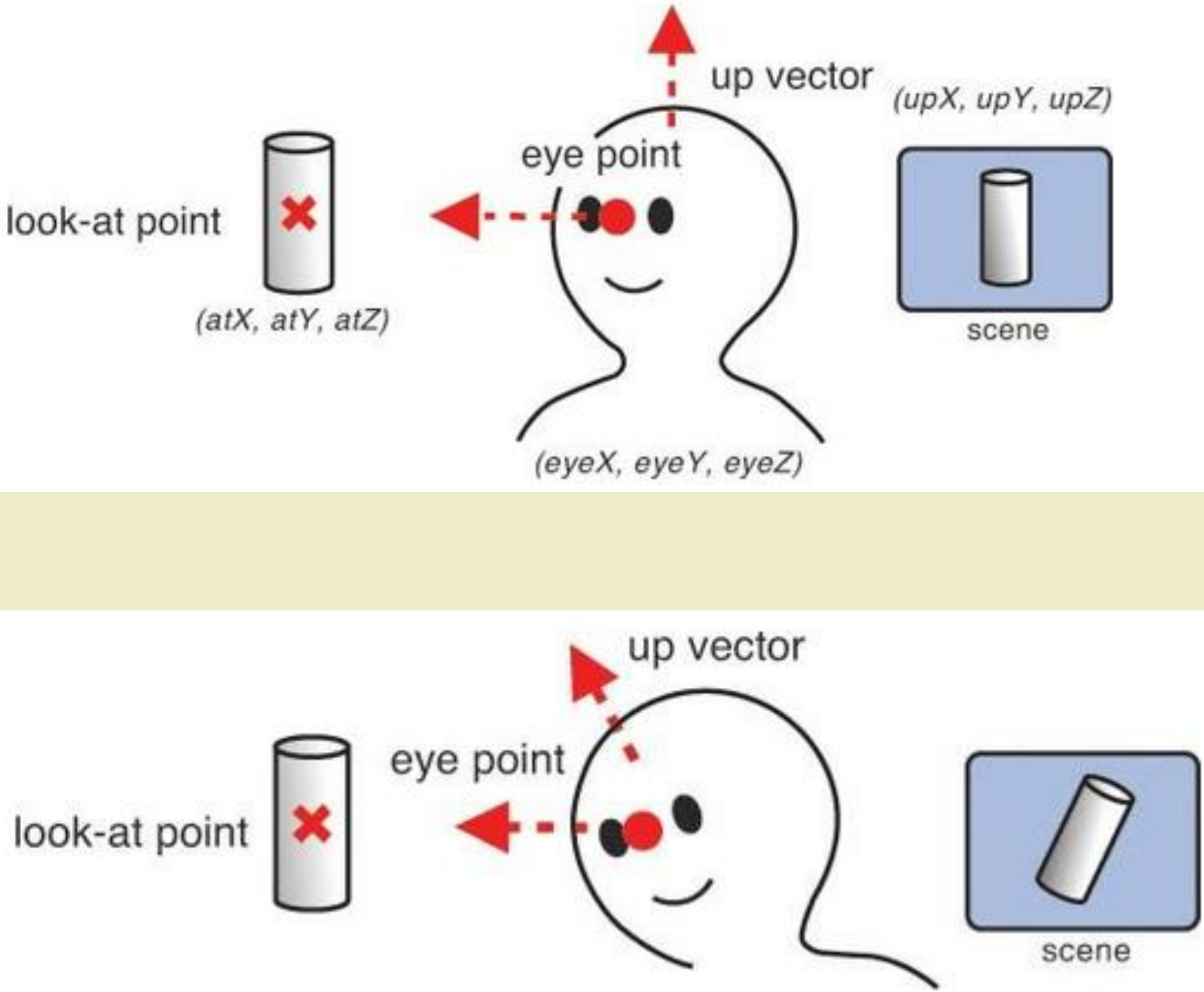
lookAt(eye,center,up)

x



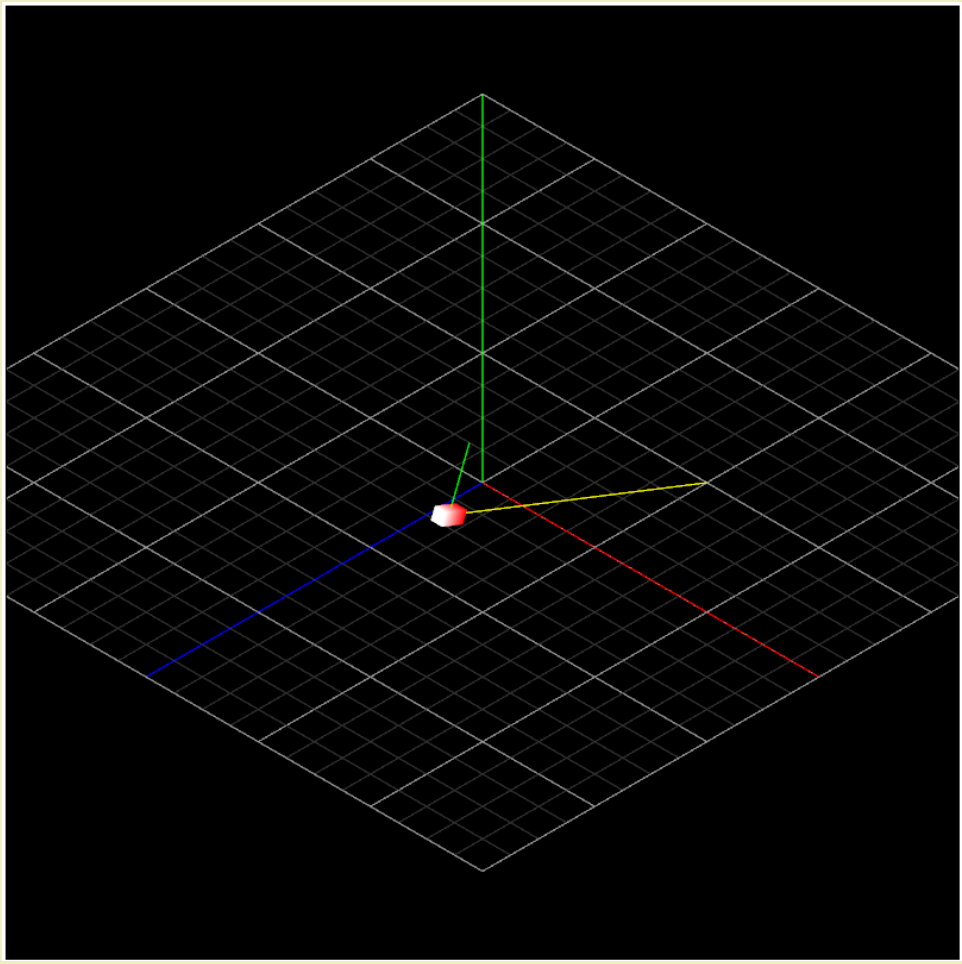
Cube Vertices (View Coordinates)

Cube Vertices (World Coordinates)

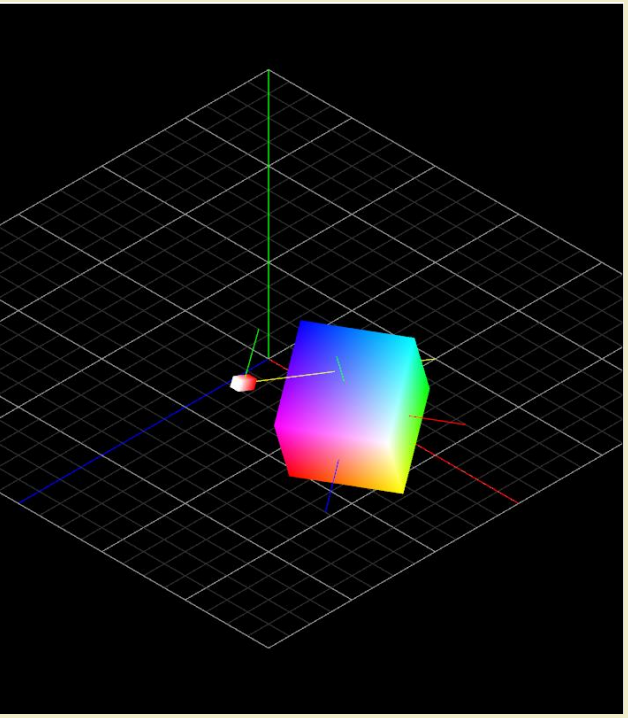

up vector
(upX, upY, upZ)
eye point
look-at point
(atX, atY, atZ)
scene
(eyeX, eyeY, eyeZ)
up vector
eye point
look-at point
scene

lookAt([2,1.9,2.3],[2,1,0],[0,1,0])

$$\begin{bmatrix} 1 & 0 & 0 & -2 \\ 0 & 0.93 & -0.36 & -0.93 \\ 0 & 0.36 & 0.93 & -2.83 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

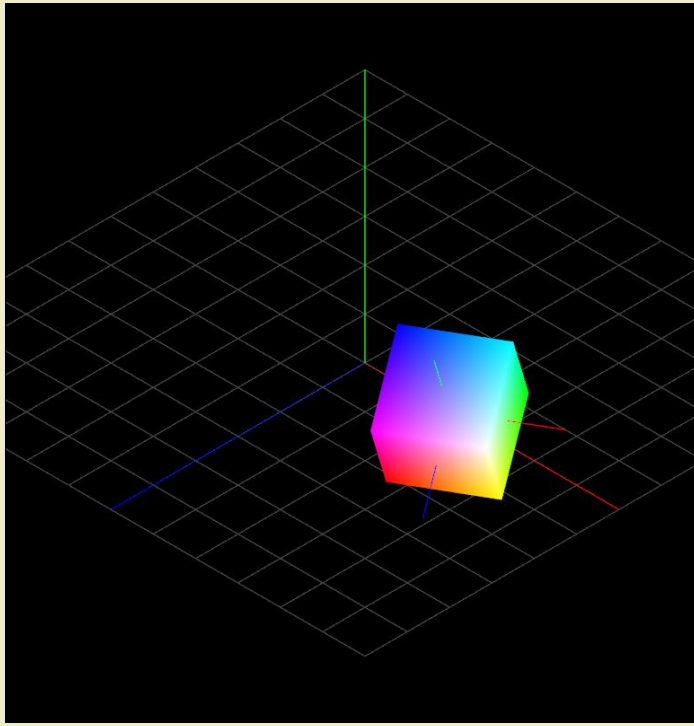


View Matrix



$$= \begin{bmatrix} 1 & 0 & 0 & -2 \\ 0 & 0.93 & -0.36 & -0.93 \\ 0 & 0.36 & 0.93 & -2.83 \\ 0 & 0 & 0 & 1 \end{bmatrix} X$$

lookAt([2,1.9,2.3],[2,1,0],[0,1,0])

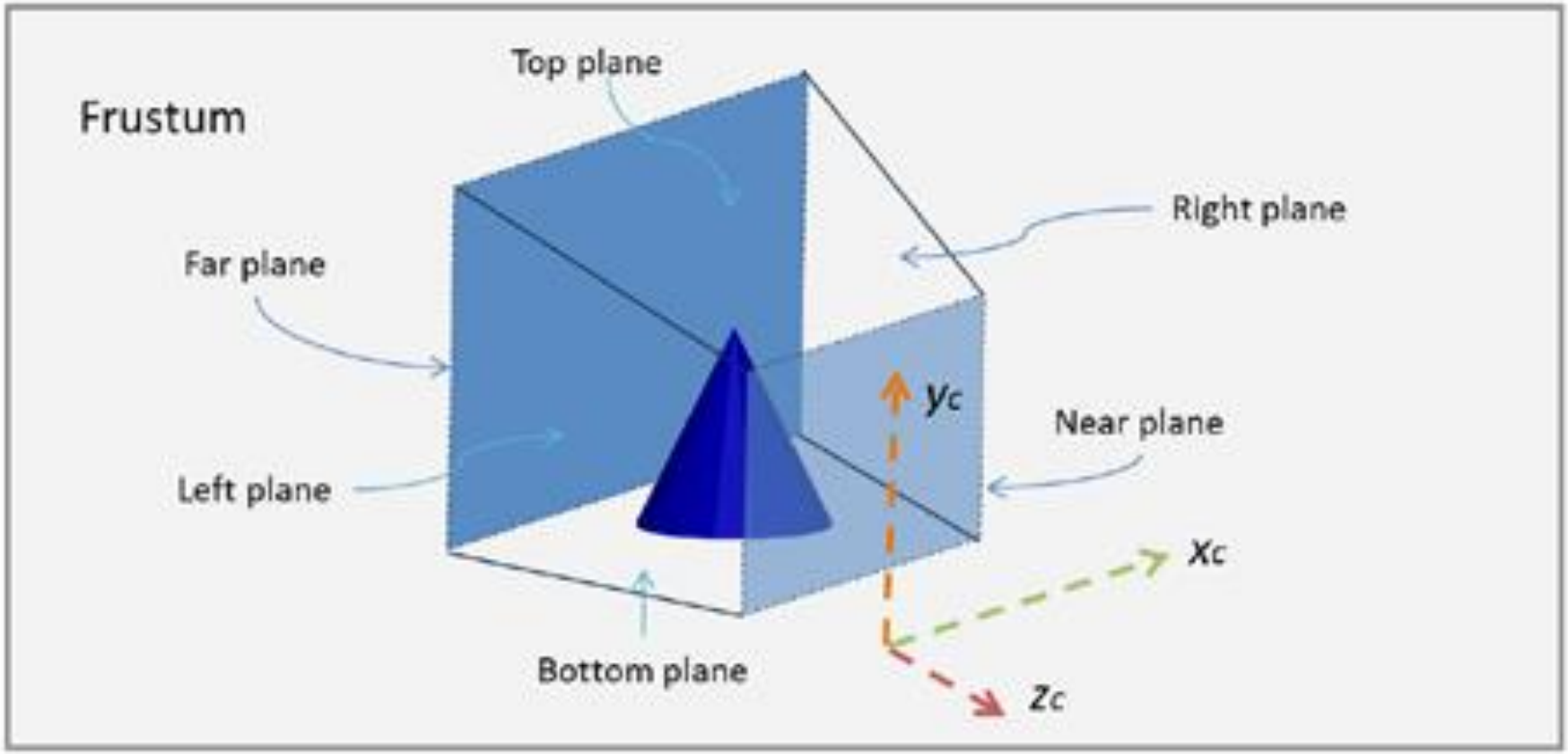


Cube Vertices (View Coordinates)

Cube Vertices (World Coordinates)

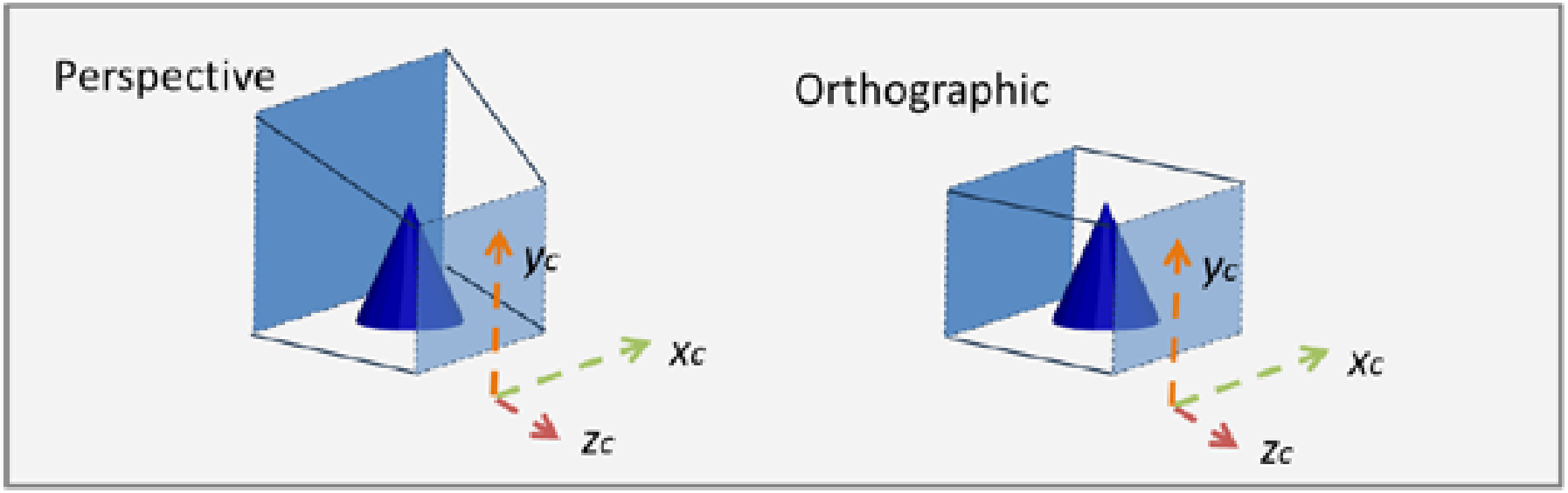
PROJECTION TRANSFORM

Projection Transform



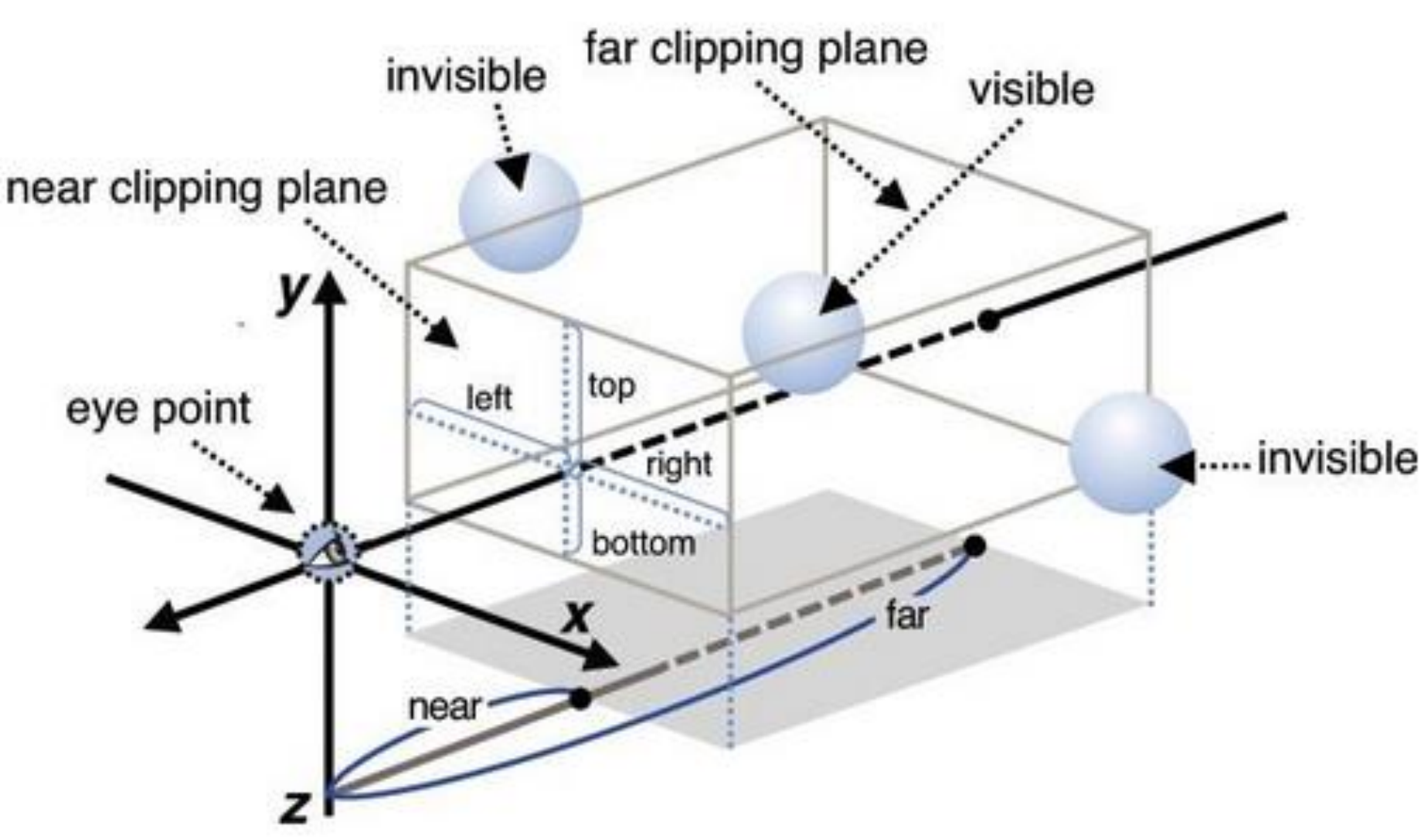
The frustum determines which objects or portion of objects will be *clipped out* and discarded.

Frustum shape

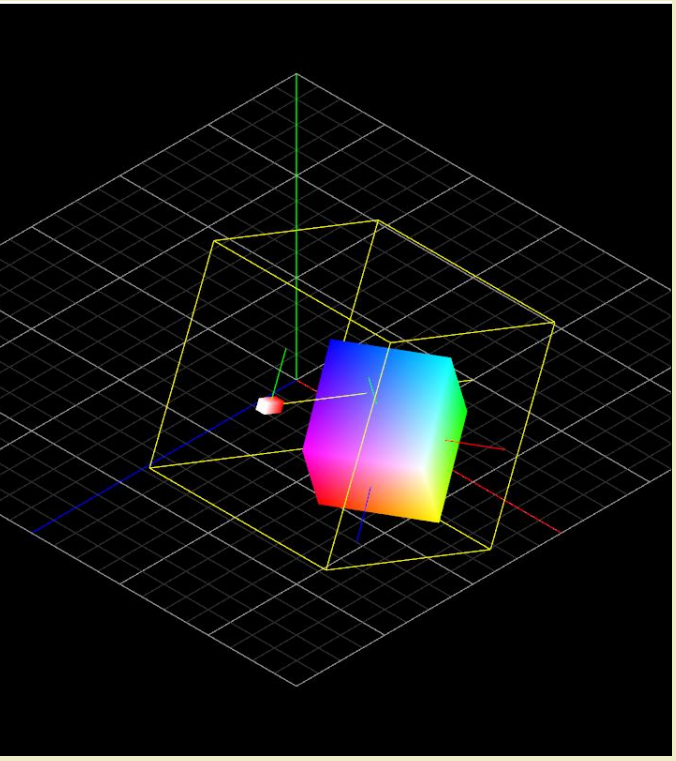


The extent and shape of the frustum determines how much of the 3D view space Is mapped to the screen and the type of 3D to 2D projection that takes place.

Orthographic Viewing Volume



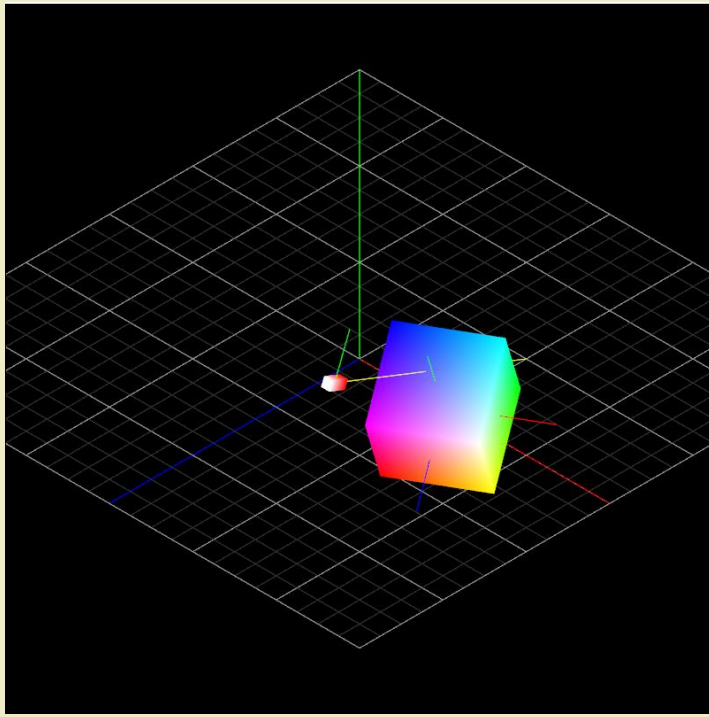
Projection Matrix (Orthographic)



=

ortho(
left,right,
bottom,top,
near,far)

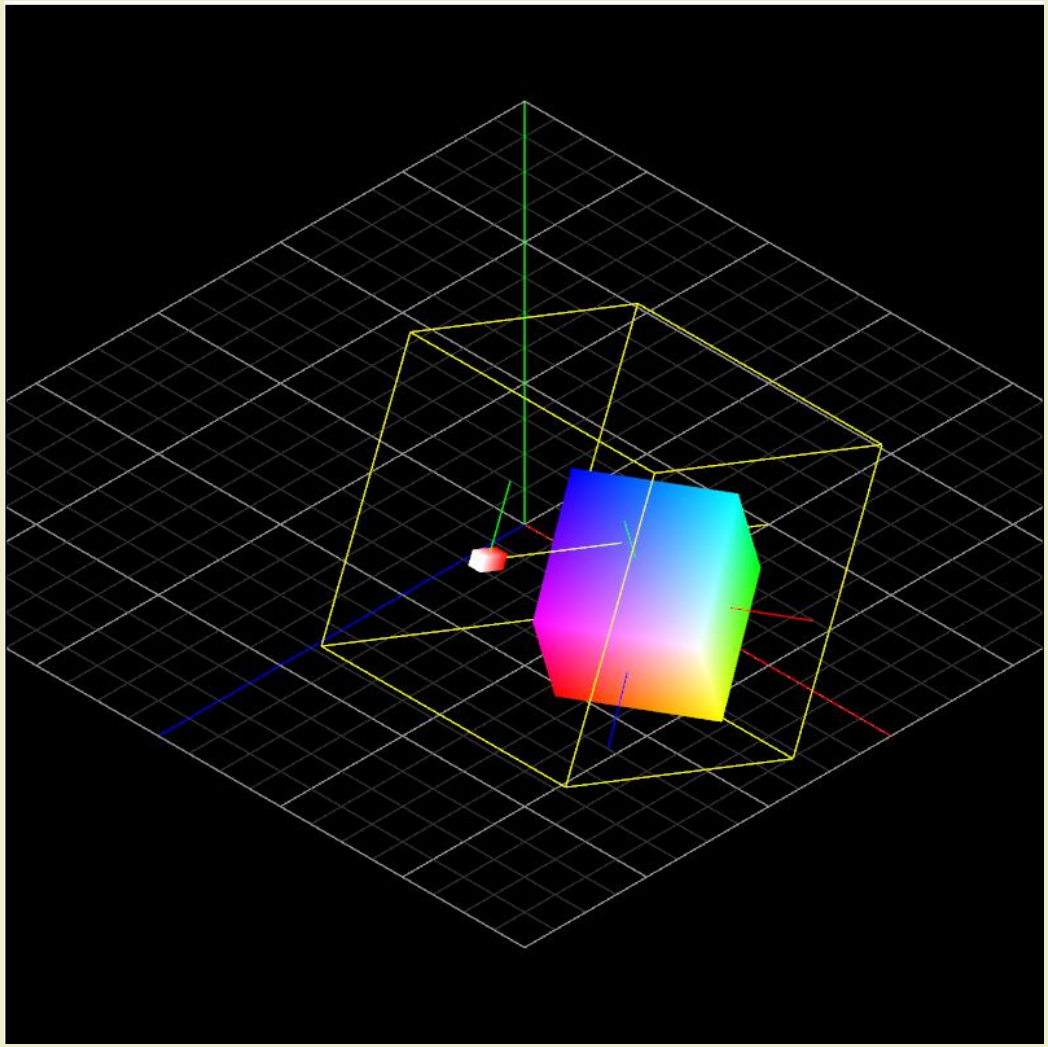
x



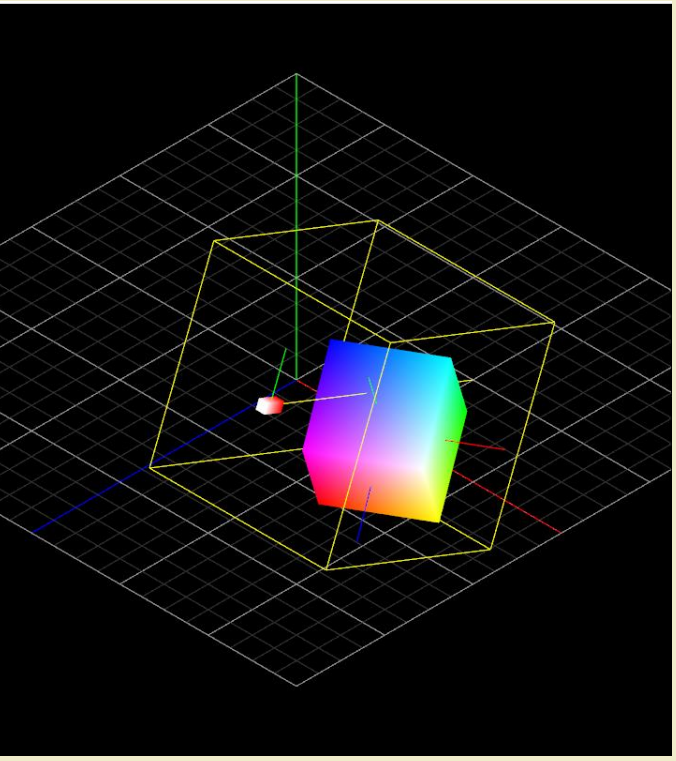
Cube Vertices (View Coordinates)

Ortho(-1,1, -1,1, 0,2)

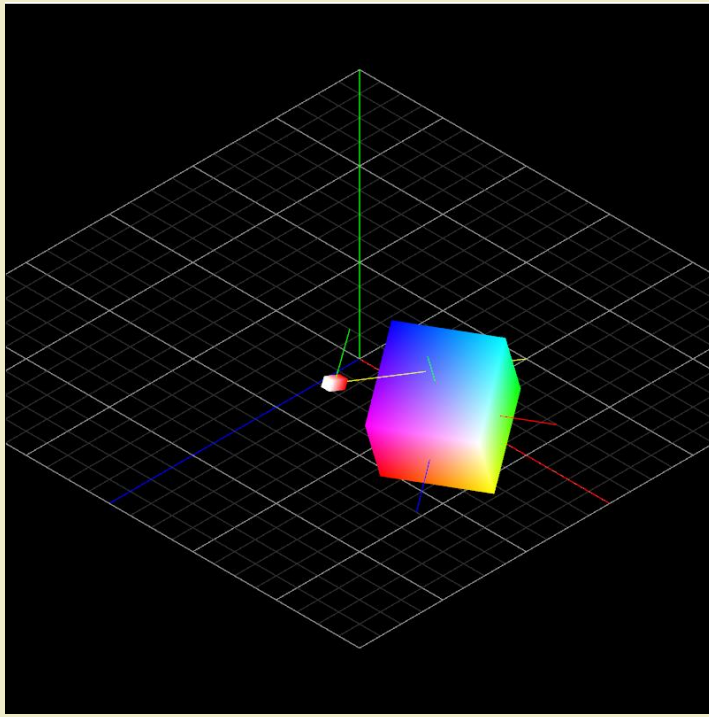
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Projection Matrix (Orthographic)



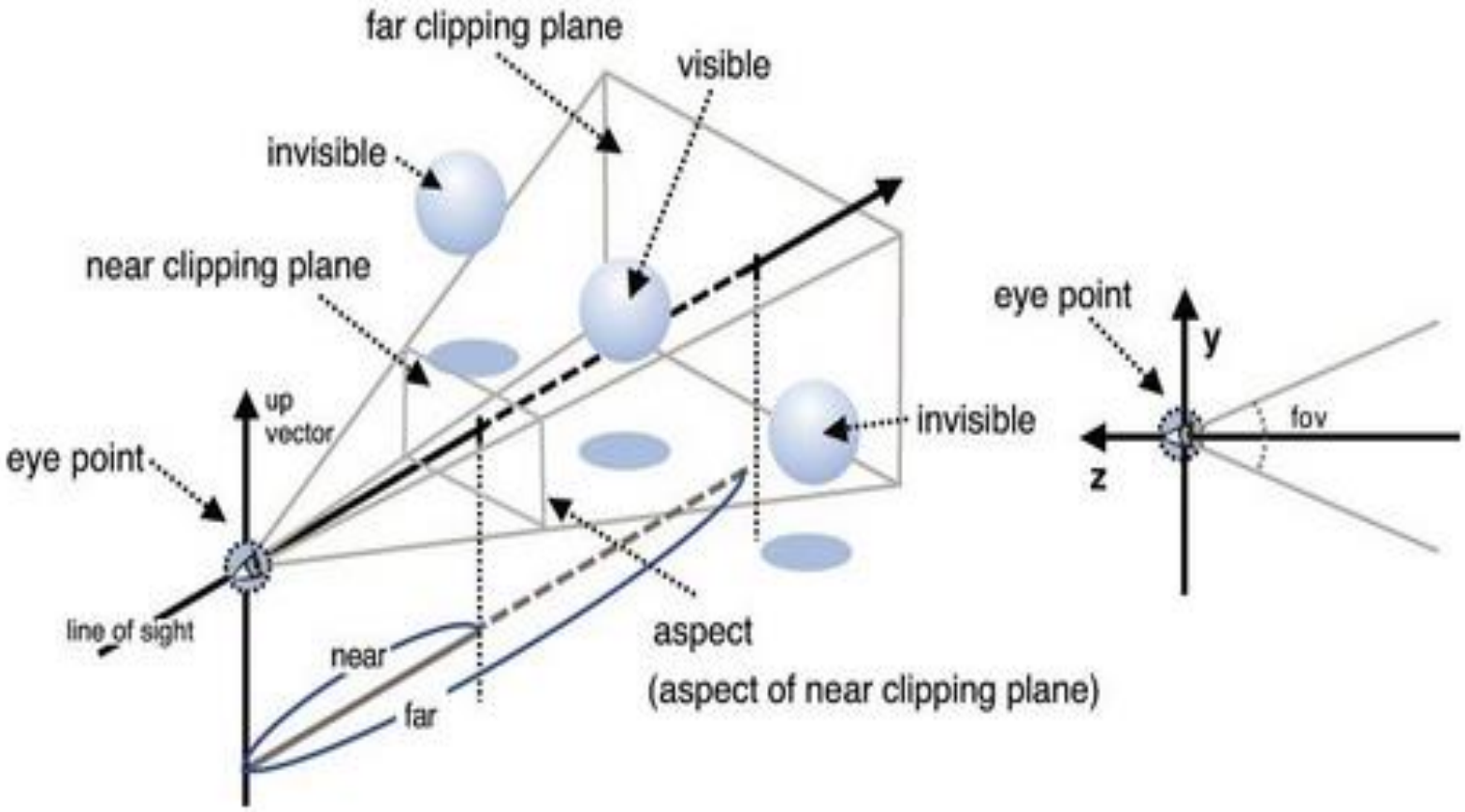
$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} x$$



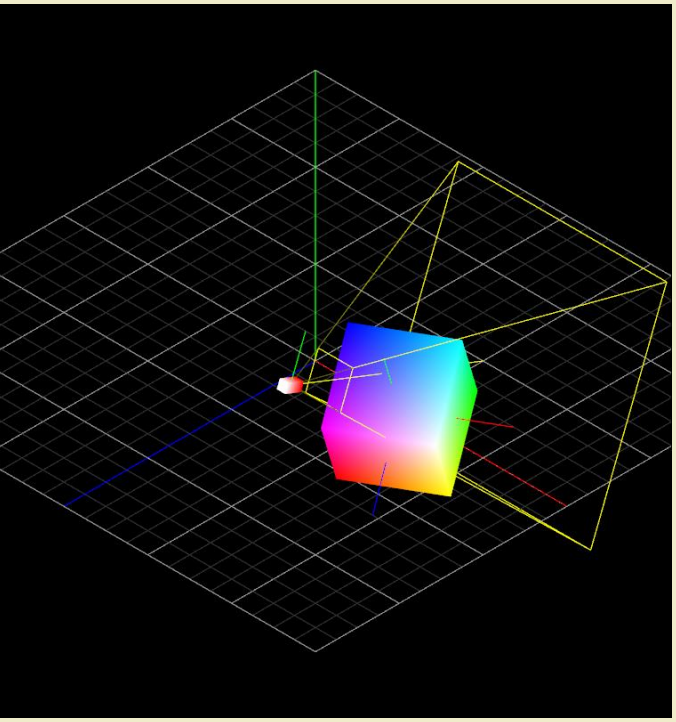
Cube Vertices (Clip Coordinates)

Cube Vertices (View Coordinates)

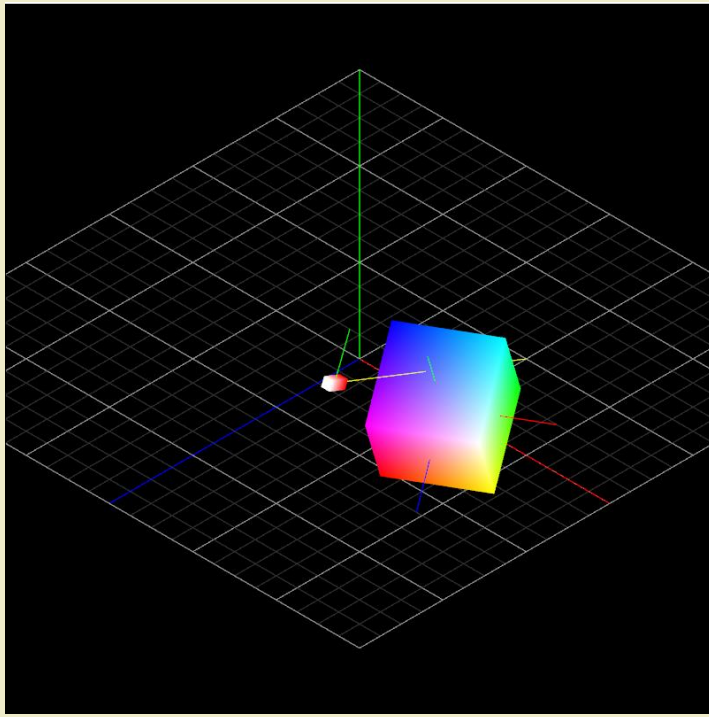
Perspective Viewing Volume



Projection Matrix (Perspective)



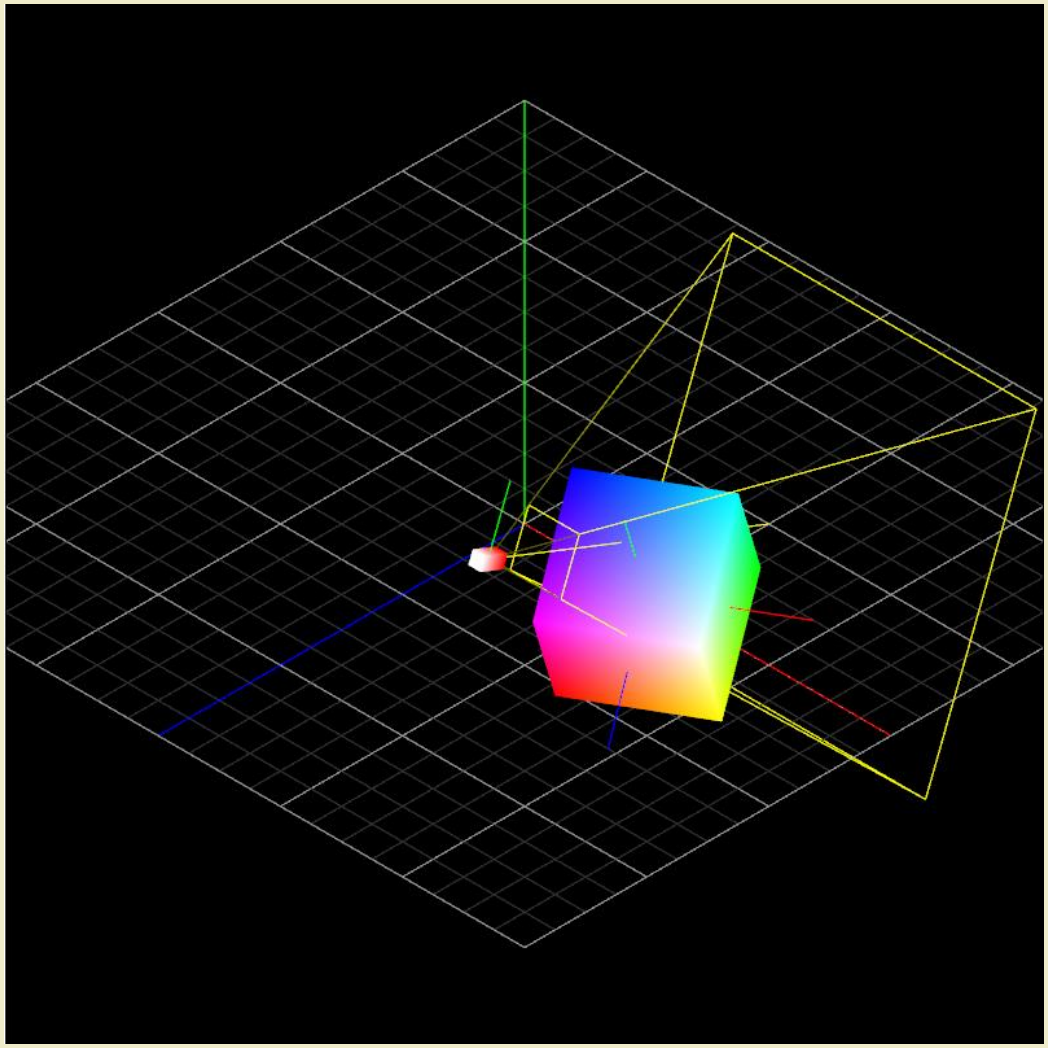
= **perspective**(fovy,aspect, near,far) X



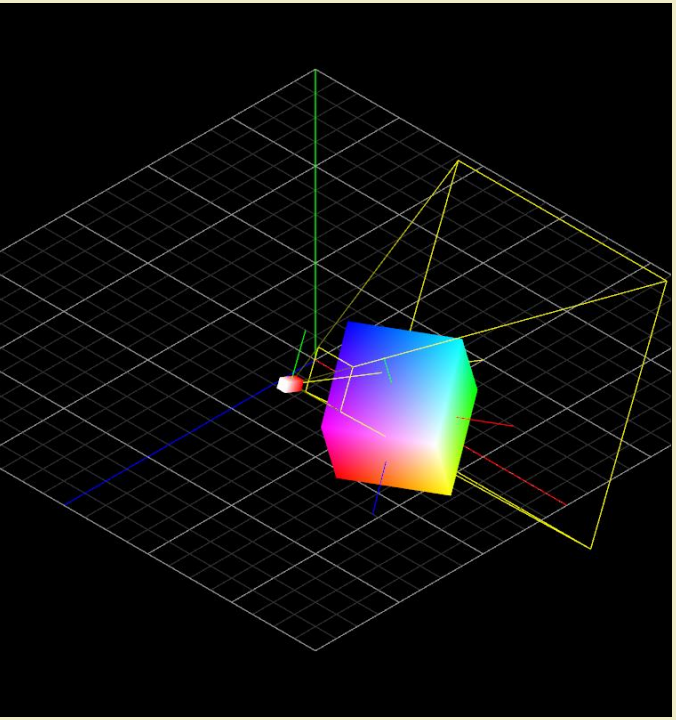
Cube Vertices (View Coordinates)

Perspective(45°,1,0.5,3)

$$\begin{bmatrix} 2.41 & 0 & 0 & 0 \\ 0 & 2.41 & 0 & 0 \\ 0 & 0 & -1.40 & -1.20 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

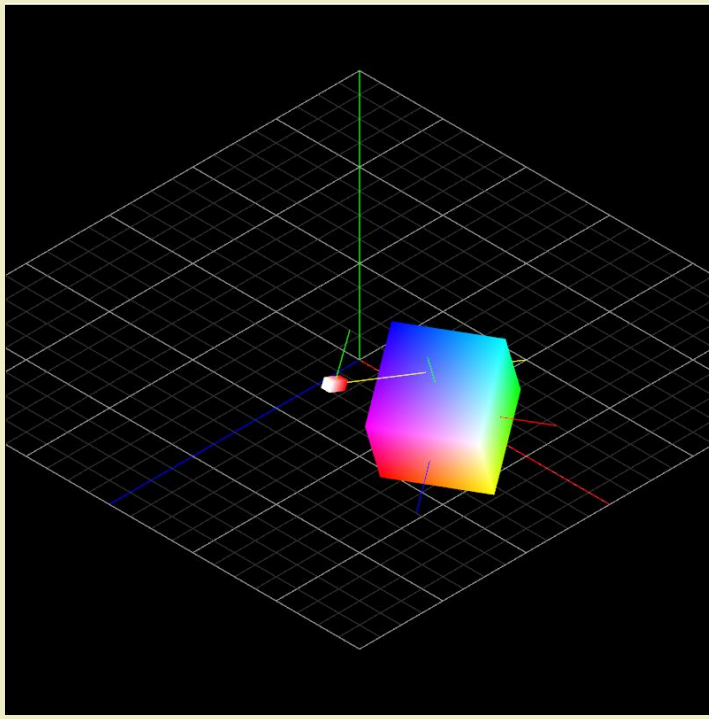


Projection Matrix (Perspective)



Cube Vertices (Clip Coordinates)

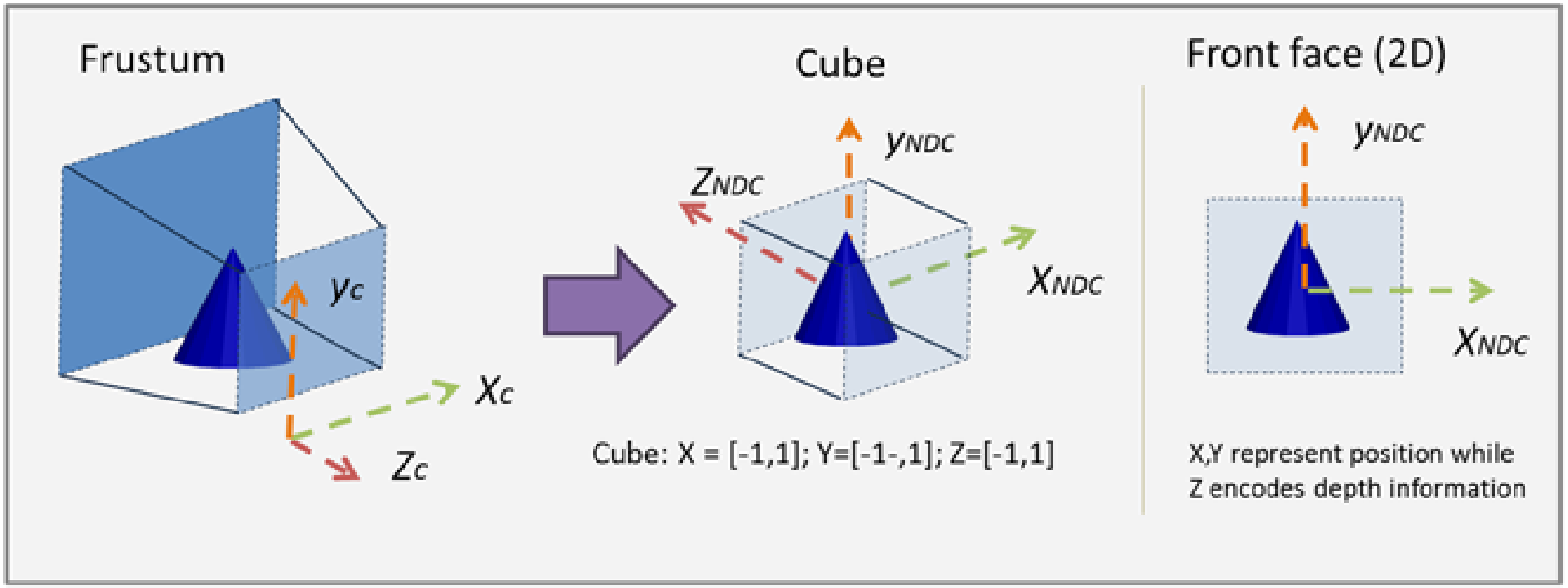
$$= \begin{bmatrix} 2.41 & 0 & 0 & 0 \\ 0 & 2.41 & 0 & 0 \\ 0 & 0 & -1.40 & -1.20 \\ 0 & 0 & -1 & 0 \end{bmatrix} \times$$



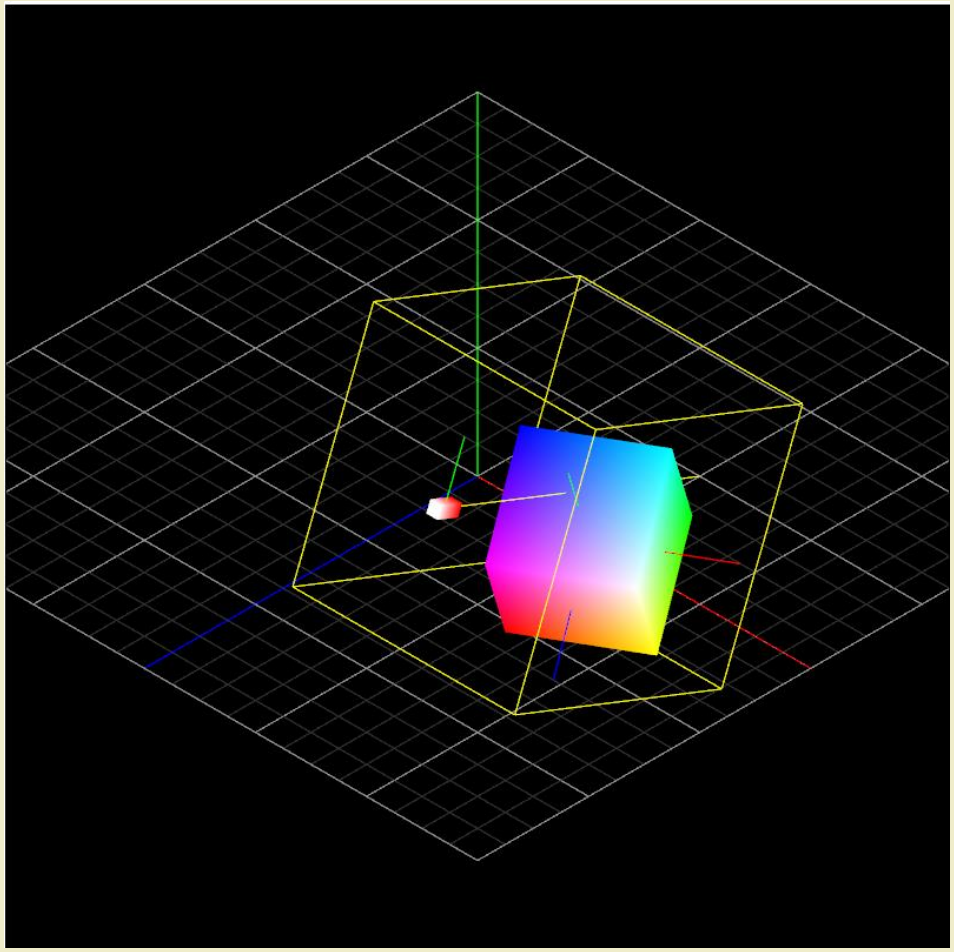
Cube Vertices (View Coordinates)

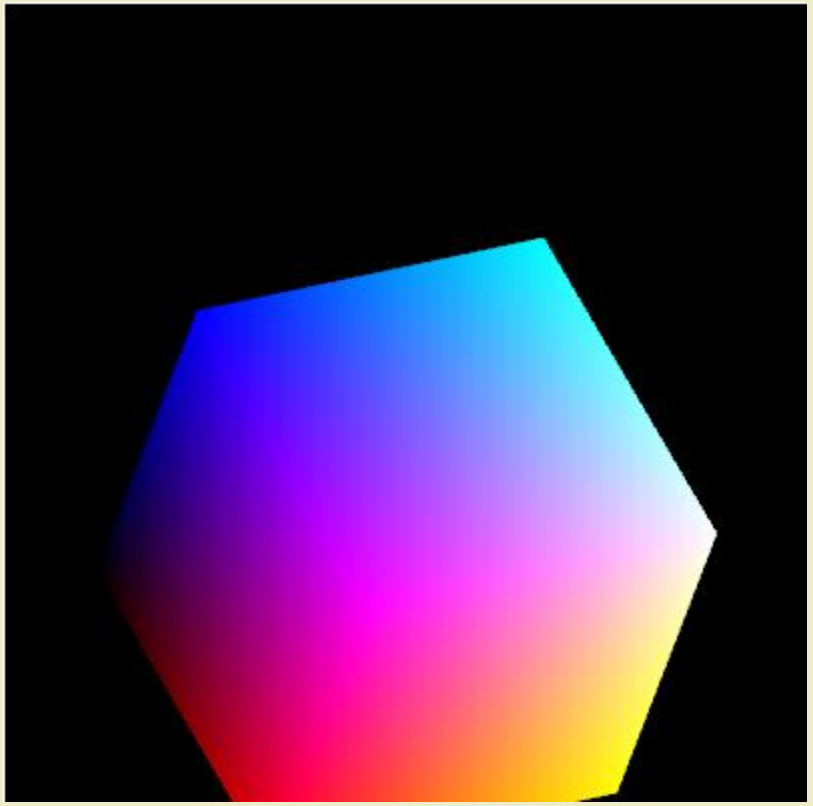
PERSPECTIVE DIVISION

Normalized Device Coordinates

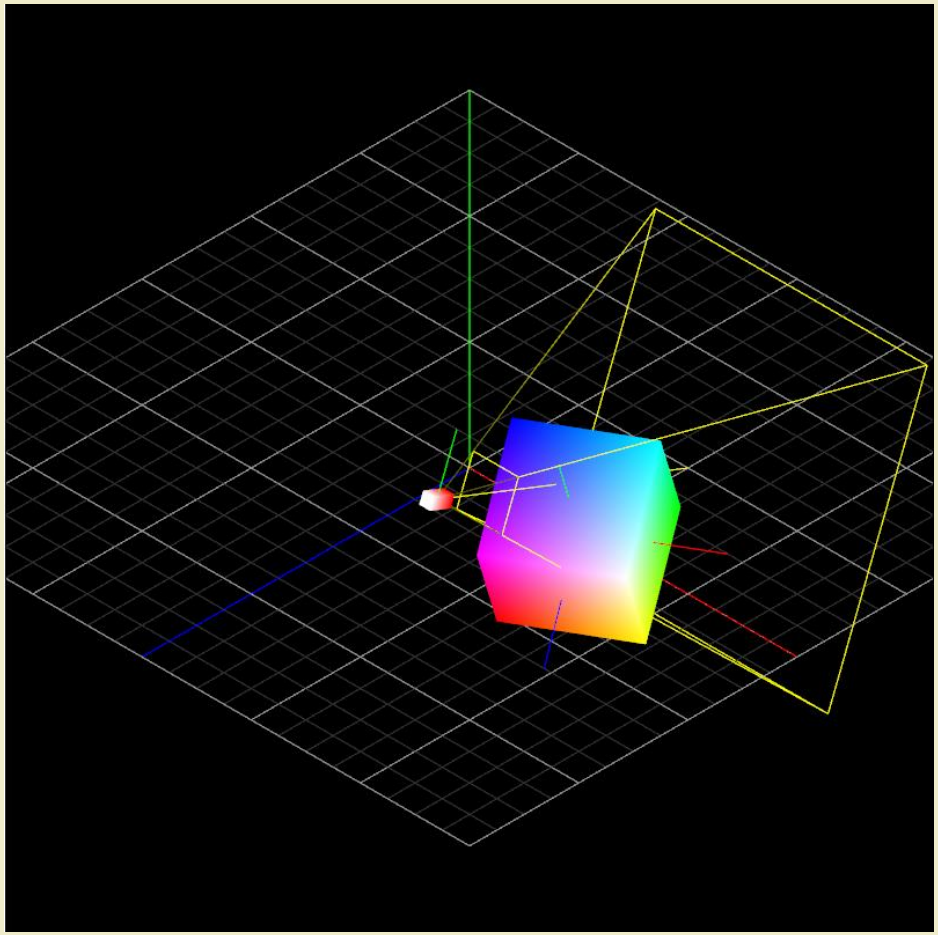


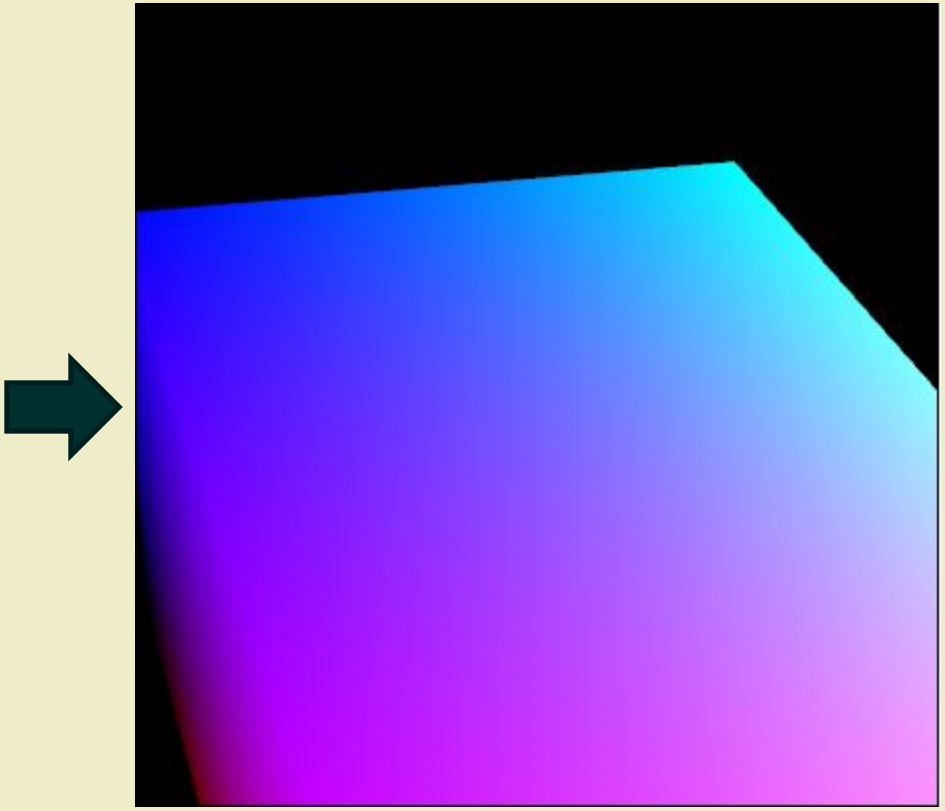
Perspective Division (Orthographic)





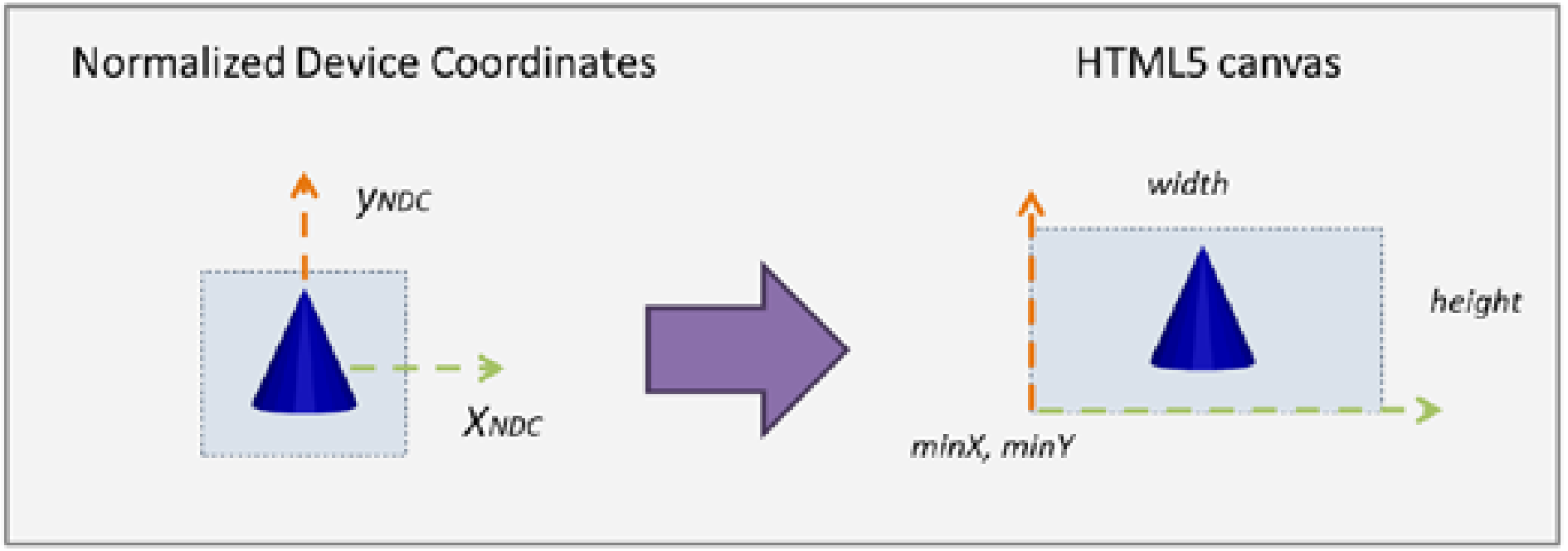
Perspective Division (Perspective)

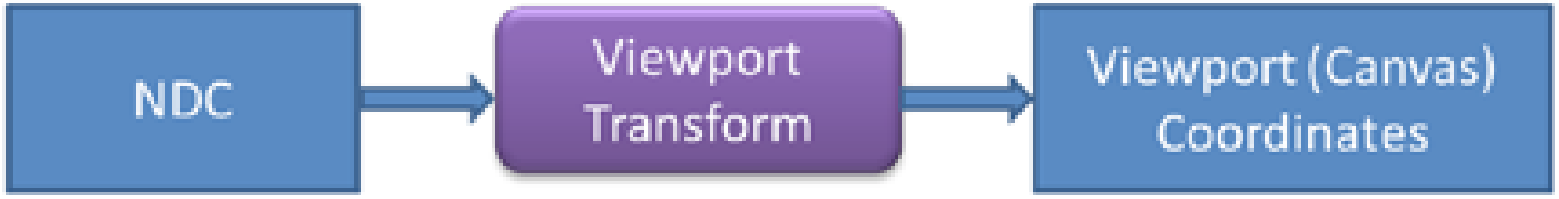




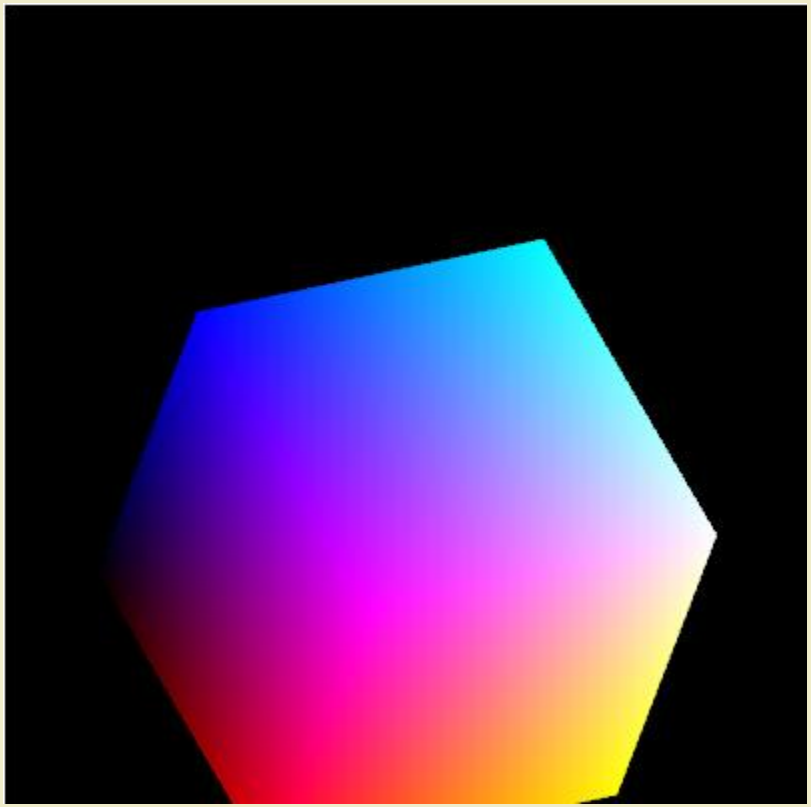
VIEWPORT TRANSFORM

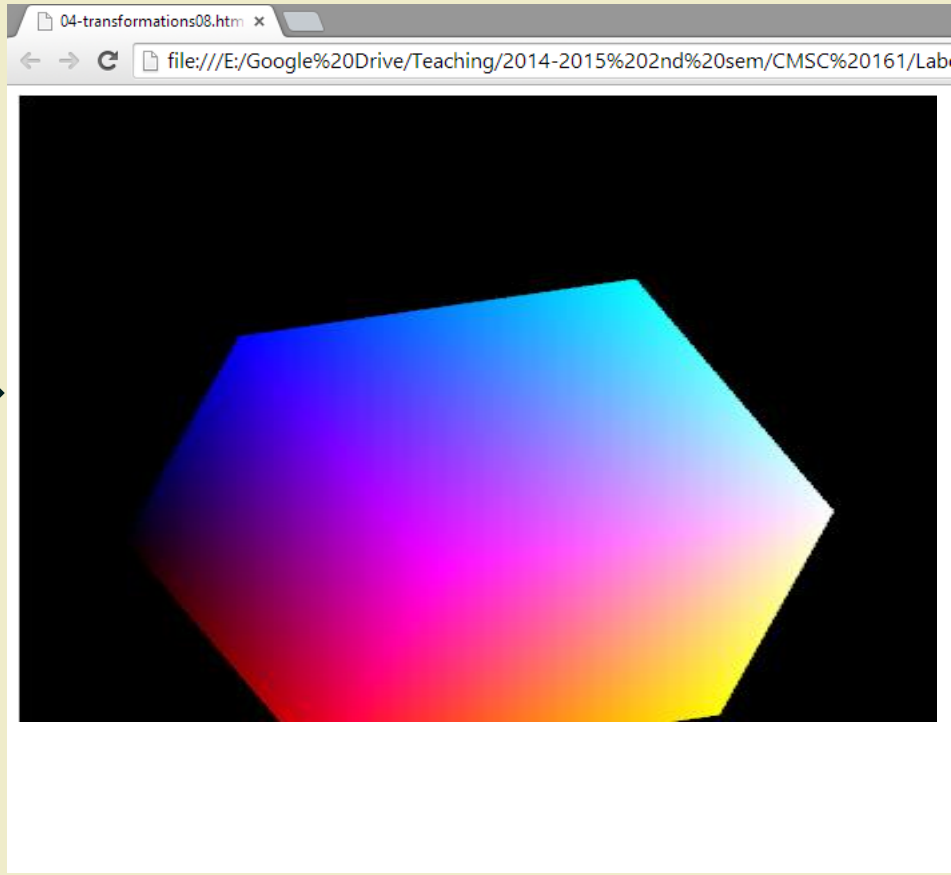
Viewport Transform



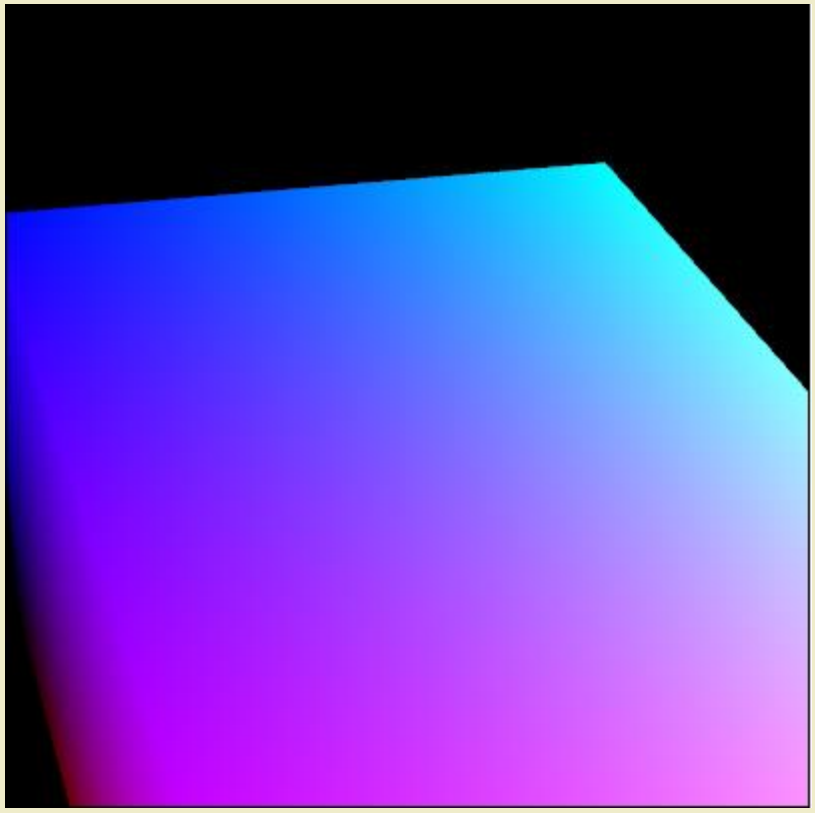


Perspective Division (Orthographic)

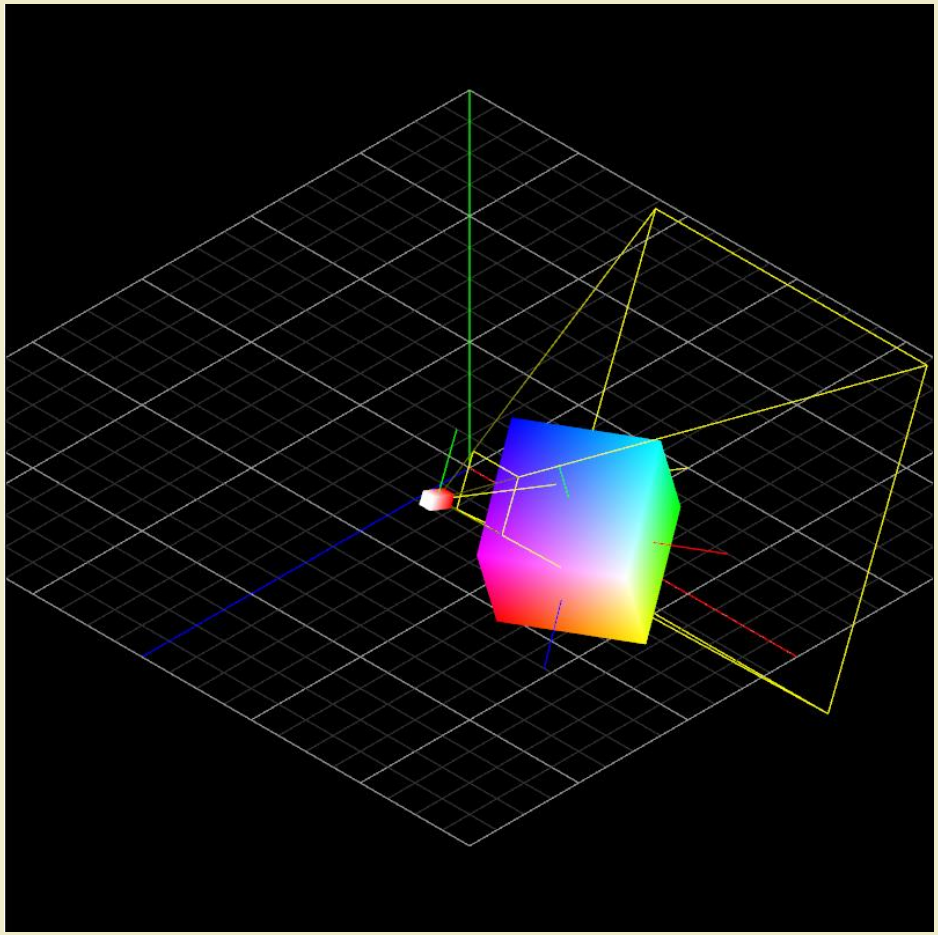


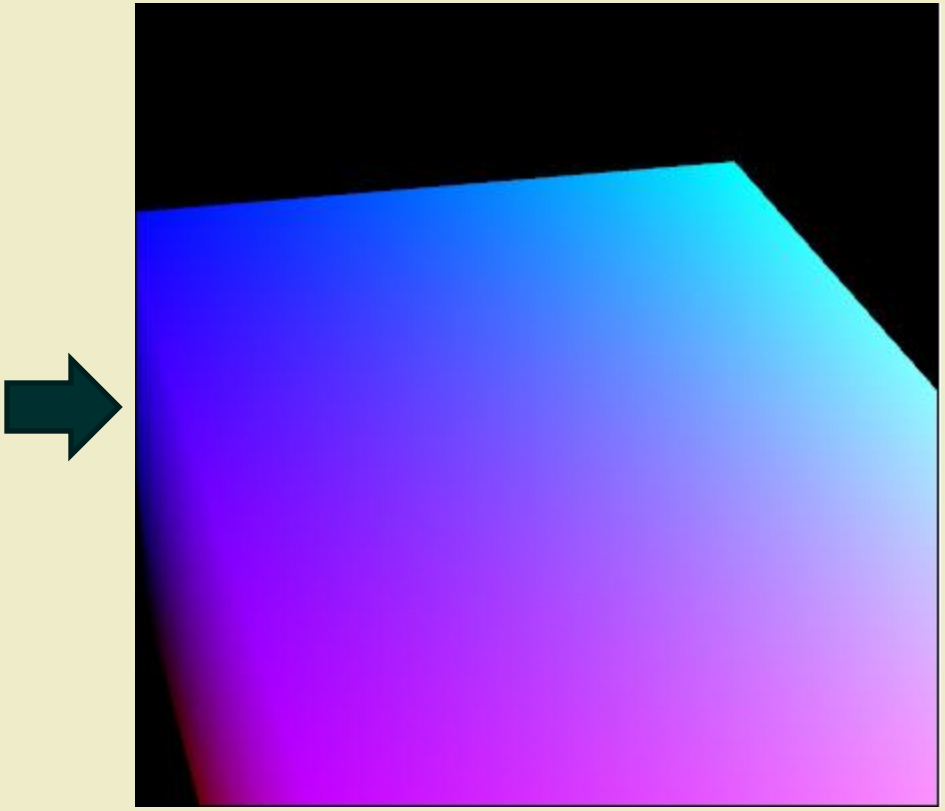


Perspective Division (Perspective)



Perspective Division (Perspective)





References

Books

- ANGEL, E. AND SHREINER, D. 2012. Interactive computer graphics : a top-down approach with shader-based OpenGL. Addison-Wesley. 6.ed. Boston, MA.
- CANTOR, D. AND JONES, B. 2012. WebGL Beginner's Guide. Packt Publishing. Birmingham, UK.
- MATSUDA, K. AND LEA, R. 2013. WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL.. Addison-Wesley. Upper Saddle River, NJ

Lecture Slides:

- CLARIÑO, M. CMSC 161 2nd Semester 2011-12 Topic 1 Lecture Slides.
- ALAMBRA, A. CMSC 161 1st Semester 2013-14 Lecture Slides

Images

- http://www.codinglabs.net/article_world_view_projection_matrix.aspx
- http://www.songho.ca/opengl/gl_transform.html