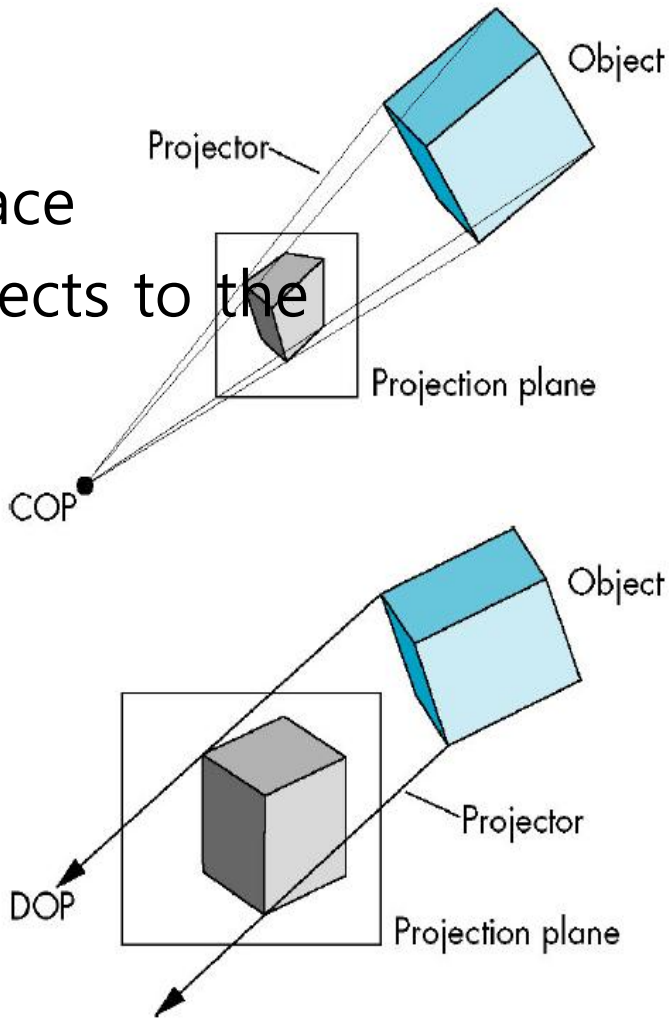
Viewing

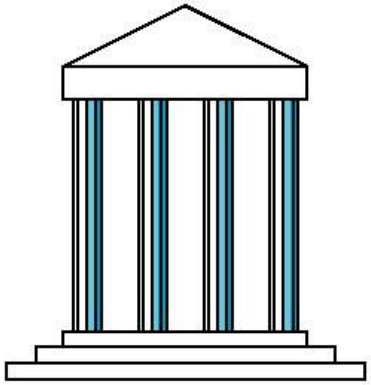
Fall 2026
11/9/2026
Kyoung Shin Park
Computer Engineering
Dankook University

Viewing

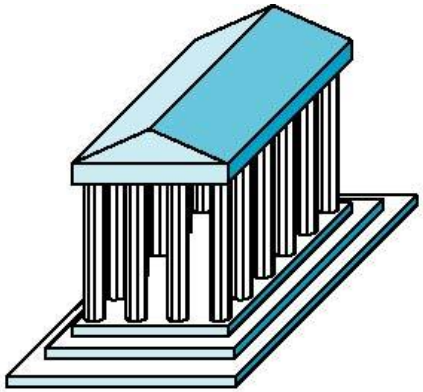
- Viewing requires basic elements
  - One or more objects
  - A viewer with a projection surface
  - Projectors that go from the objects to the projection plane
- COP vs DOP
  - Center Of Projection (COP)
  - – Perspective views
  - Direction Of Projection (DOP)
  - – Parallel views



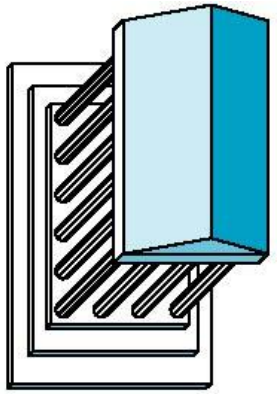
Classical Viewing



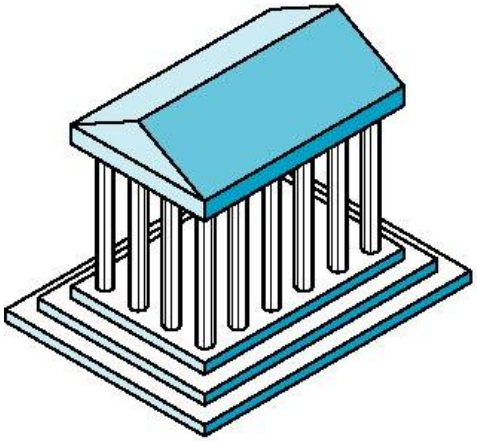
Front elevation



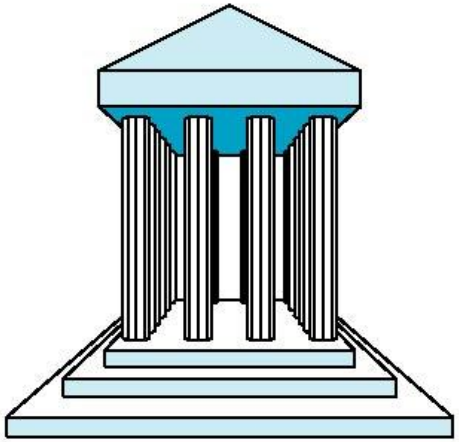
Elevation oblique



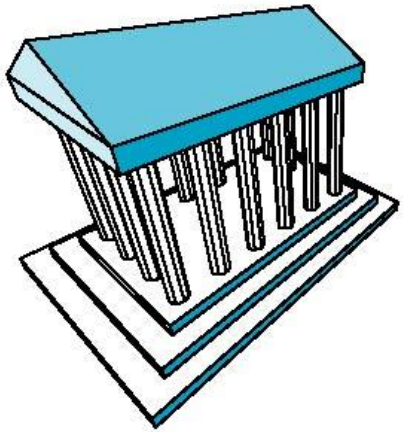
Plan oblique



Isometric

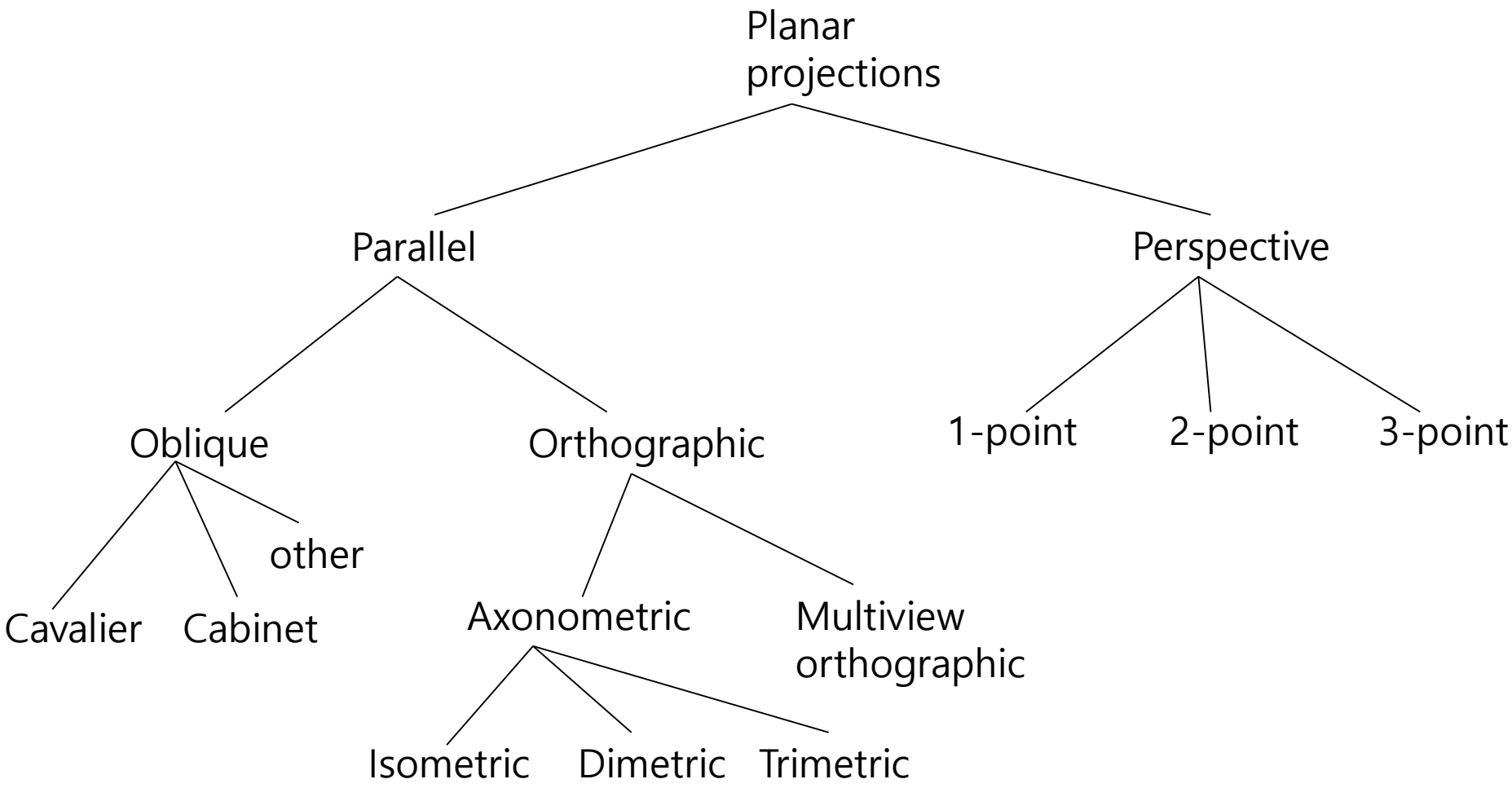


One-point perspective

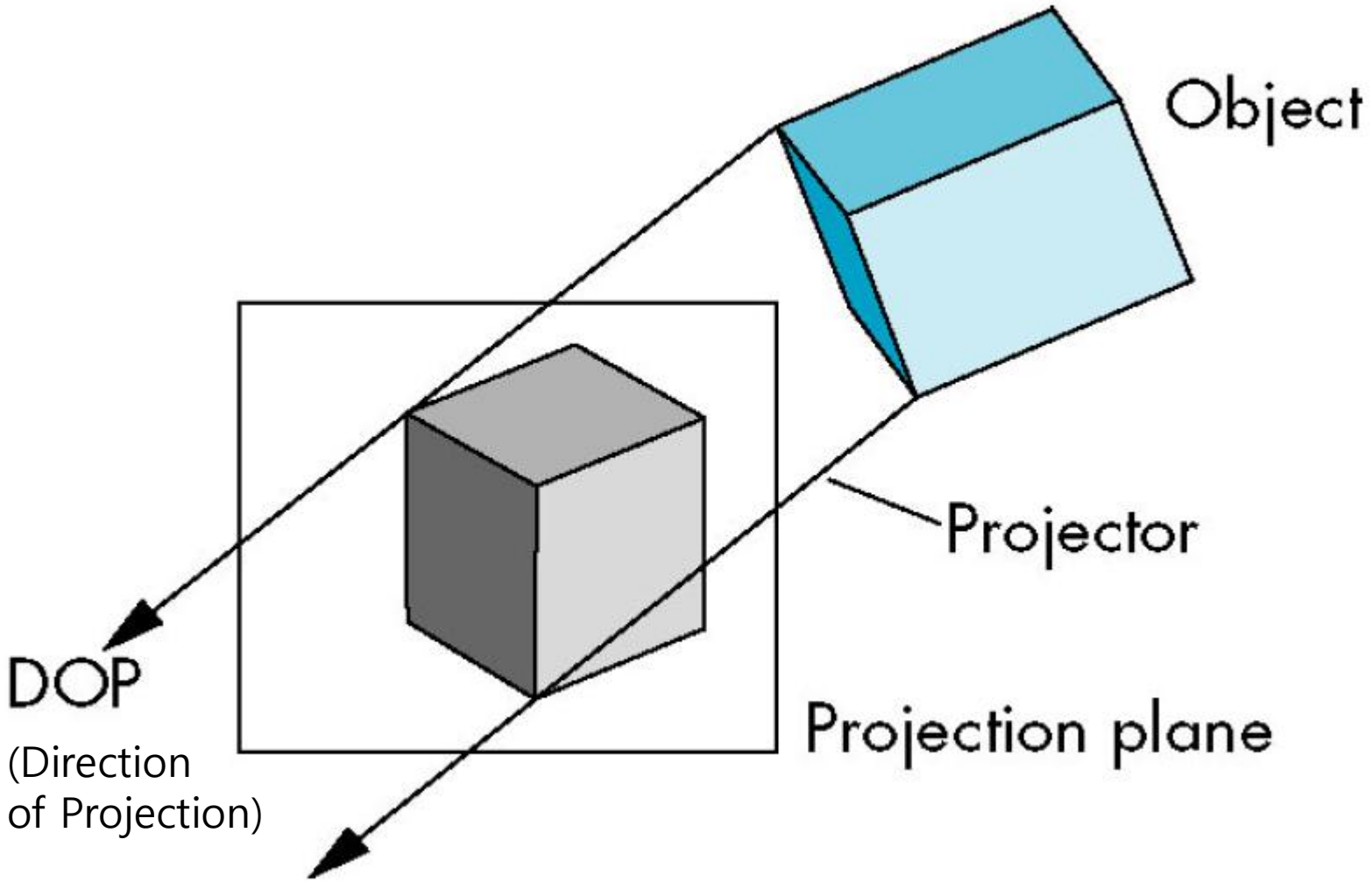


Three-point perspective

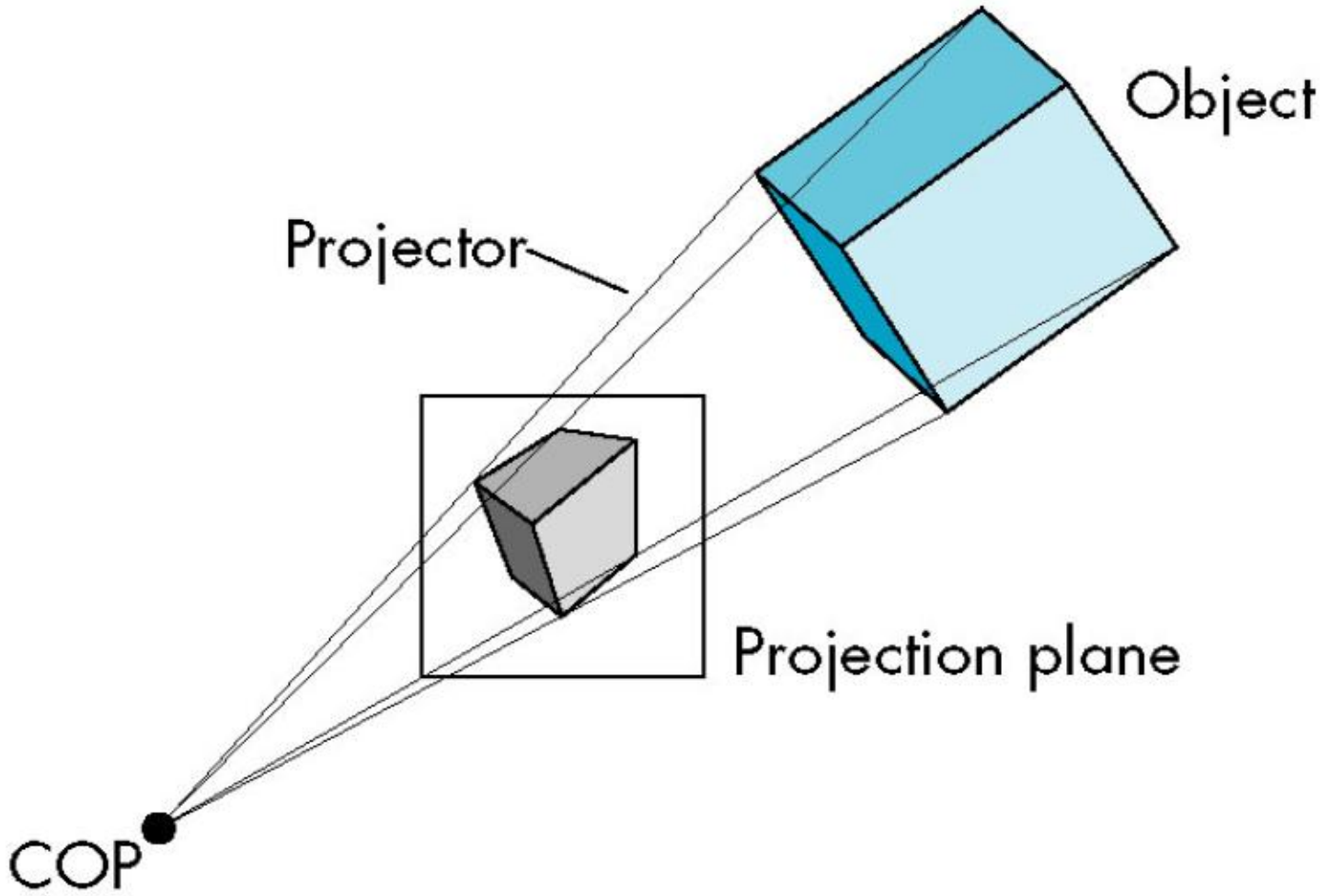
Classical Viewing



Parallel Viewing

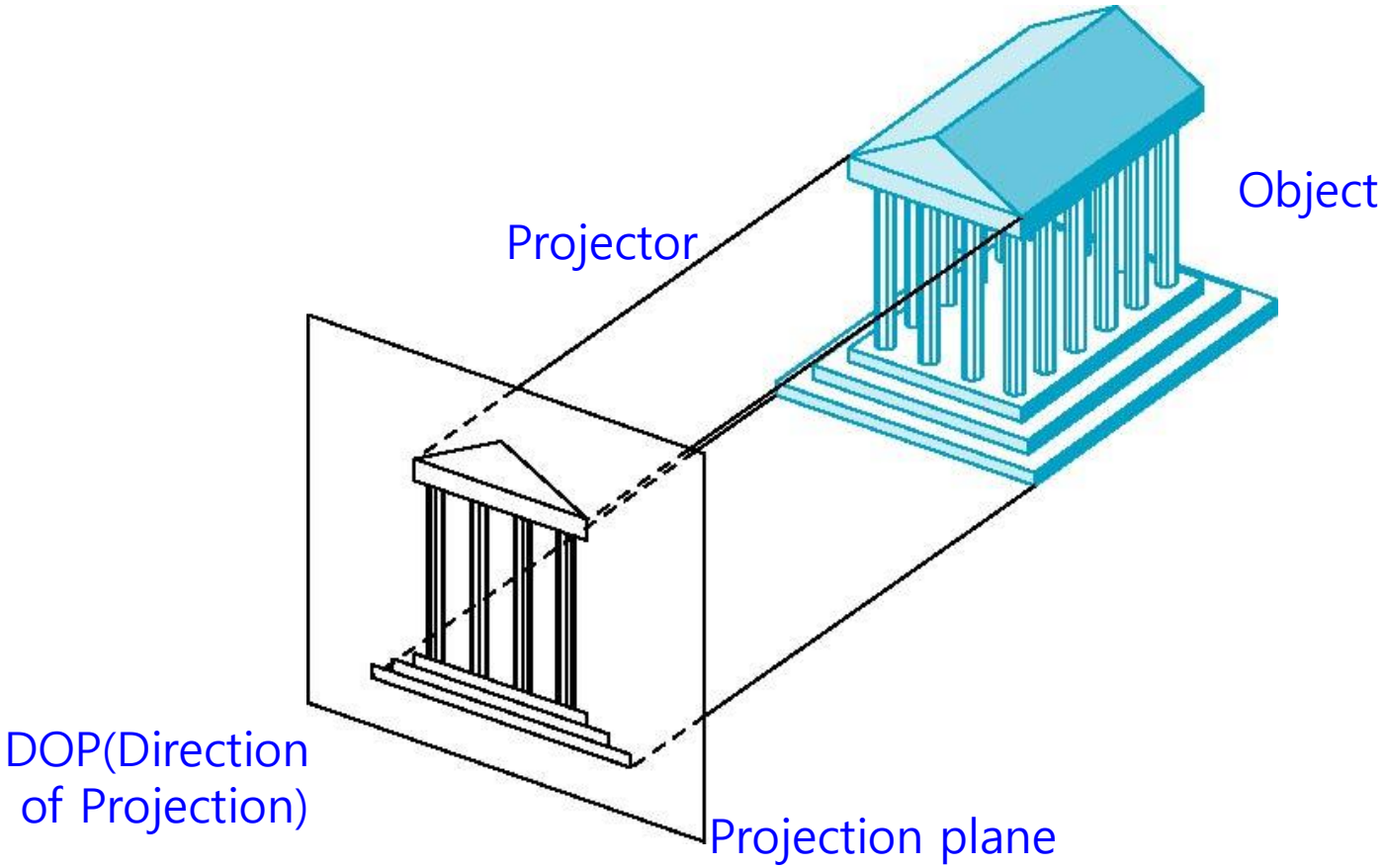


Perspective Viewing



Orthographic Projection

- In the orthographic projection, projectors are orthogonal to projection plane.



Multiview Orthographic Projection

- In the multiview orthographic projection, projection plane parallel to principal face.
- Usually form front, top, side views.

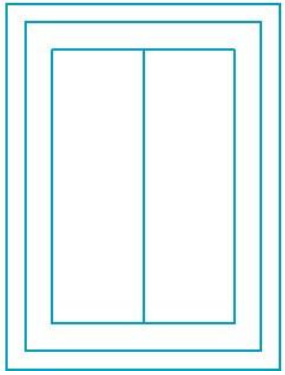
Isometric (not multiview orthographic view)



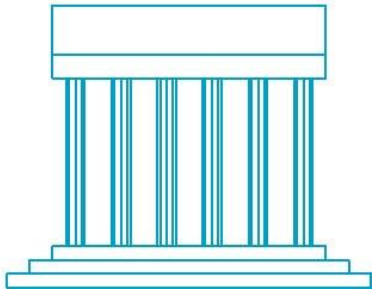


front

top



side



In CAD and architecture, we often display three multiviews plus isometric

Multiview Orthographic Projection Advantages and Disadvantages

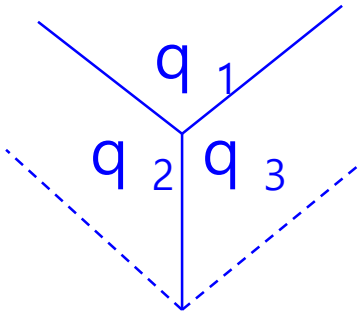
- Preserves both distances and angles
  - Shapes preserved
  - Can be used for measurements
    - Building plans
    - Manuals
- Cannot see what object really looks like because many surfaces hidden from view
  - Often we add the isometric

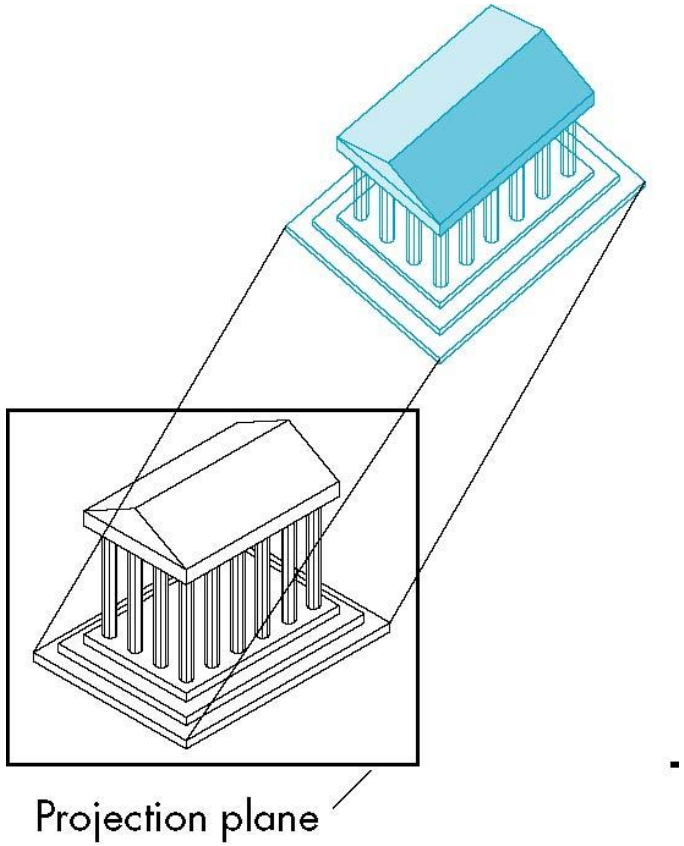
Axonometric Projections

- Axonometric projections allow projection plane to move relative to object.

classify by how many angles of a corner of a projected cube are the same

none: trimetric
two: dimetric
three: isometric





Construction of an Axonometric Projection

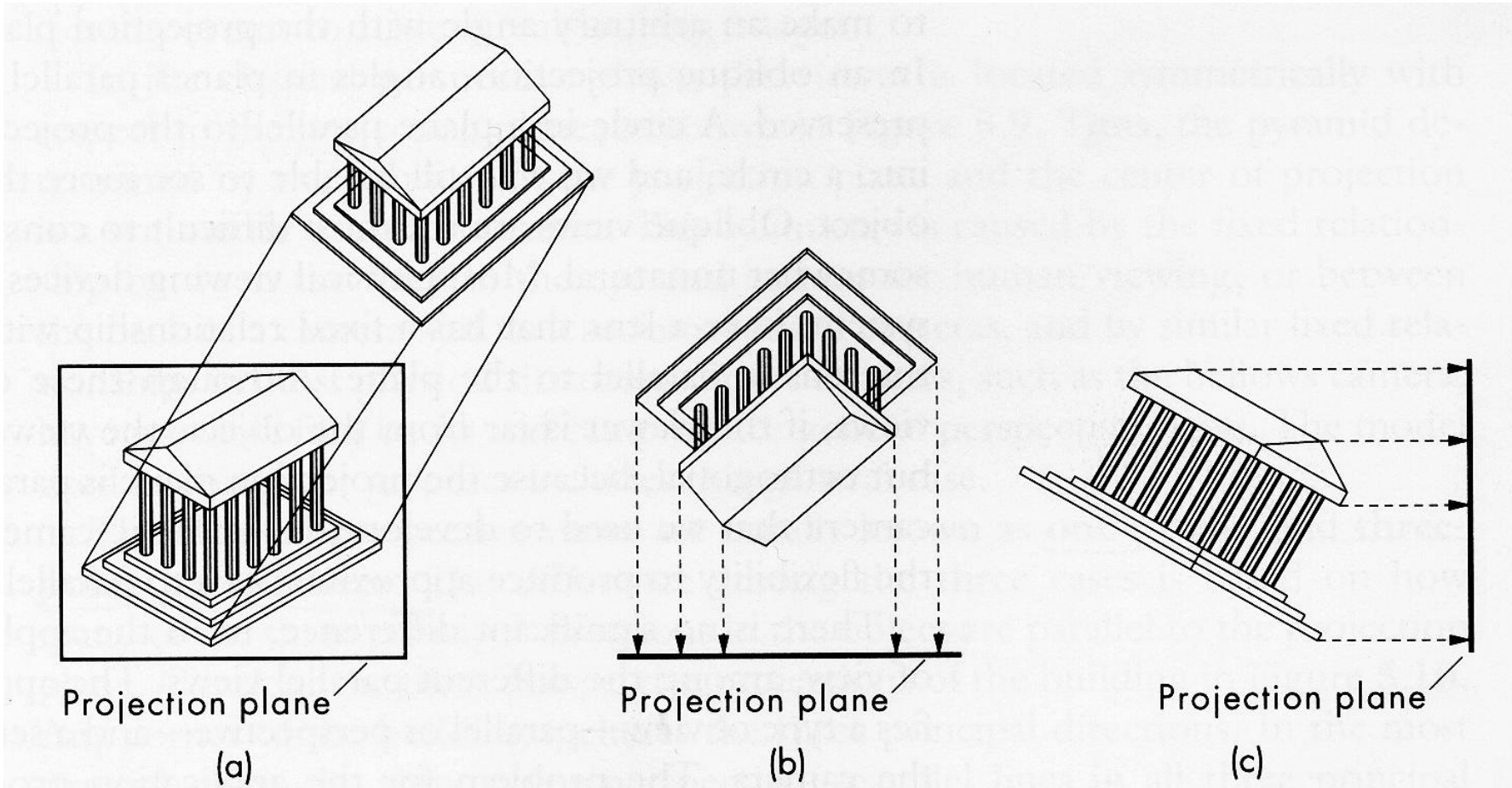
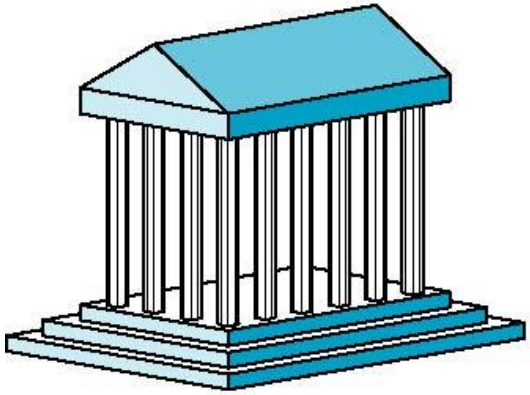
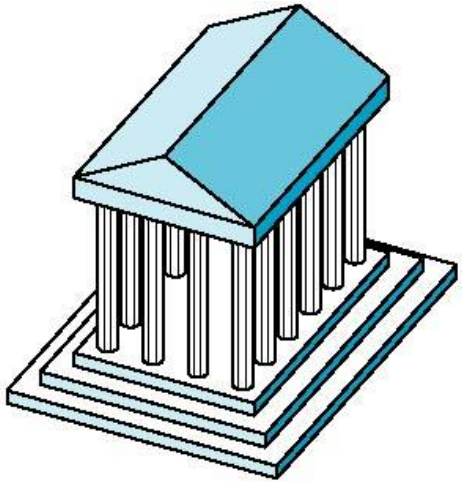


Figure 5.6 (a) Construction of an axonometric projection. (b) Top view. (c) Side view

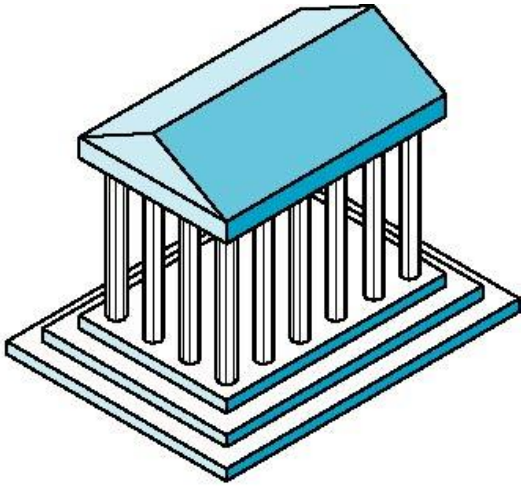
Types of Axonometric Projections



Dimetric



Trimetric



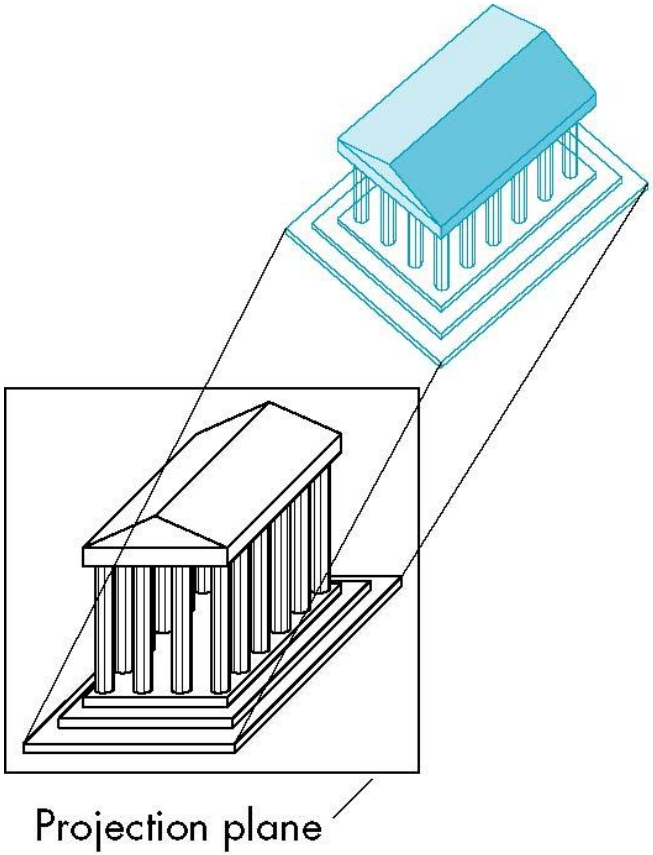
Isometric

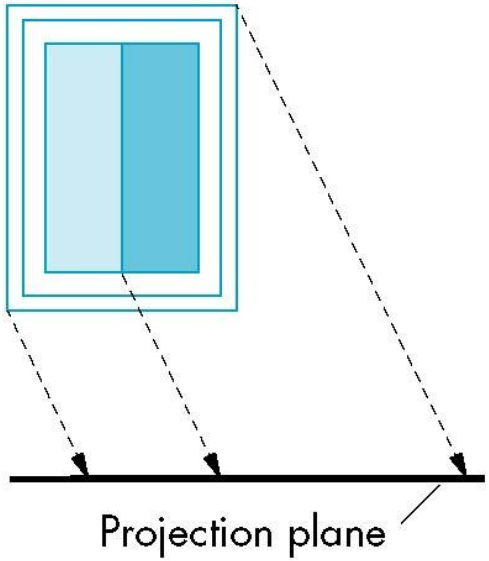
Axonometric Projections Advantages and Disadvantages

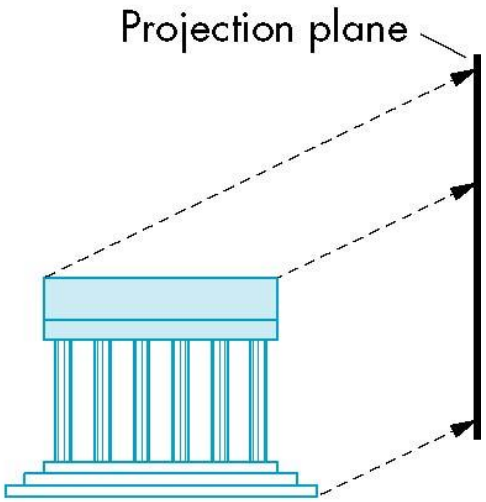
- Lines are scaled (foreshortened) but can find scaling factors
- Lines preserved but angles are not
  - Projection of a circle in a plane not parallel to the projection plane is an ellipse
- Can see three principal faces of a box-like object
- Some optical illusions possible
  - Parallel lines appear to diverge
- Does not look real because far objects are scaled the same as near objects
- Used in CAD applications

Oblique Projection

- Arbitrary relationship between projectors and projection plane







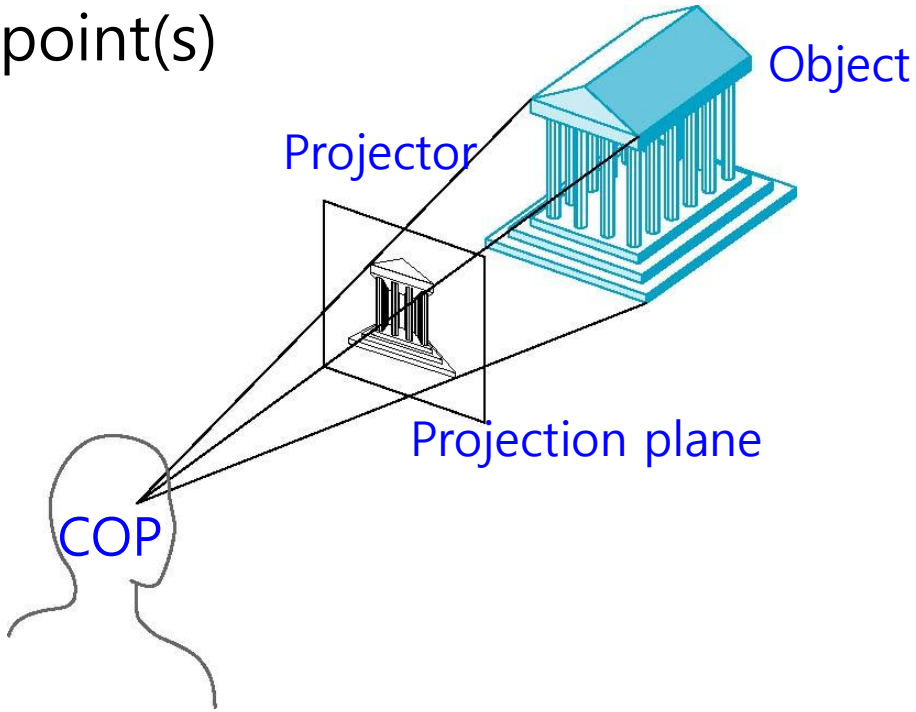
경사 투영의 평면도 & 측면도

Oblique Projection Advantages and Disadvantages

- Can pick the angles to emphasize a particular face
  - Architecture: plan oblique, elevation oblique
- Angles in faces parallel to projection plane are preserved while we can still see “around” side
- In physical world, cannot create with simple camera; possible with bellows camera or special lens (architectural)

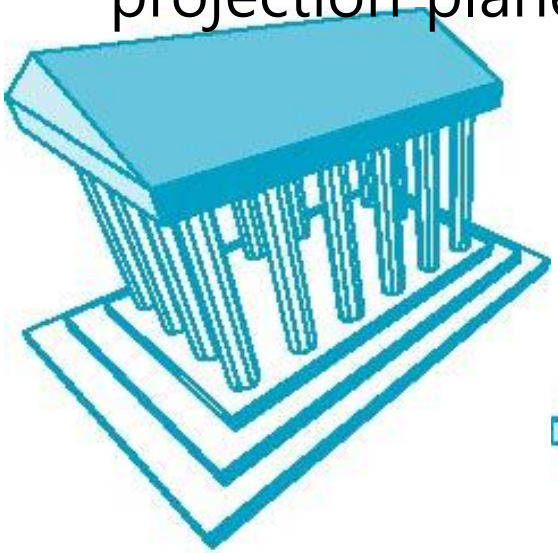
Perspective Projection

- Parallel lines (not parallel to the projection plan) on the object converge at a single point in the projection (the *vanishing point*)
- Drawing simple perspectives by hand uses these vanishing point(s)

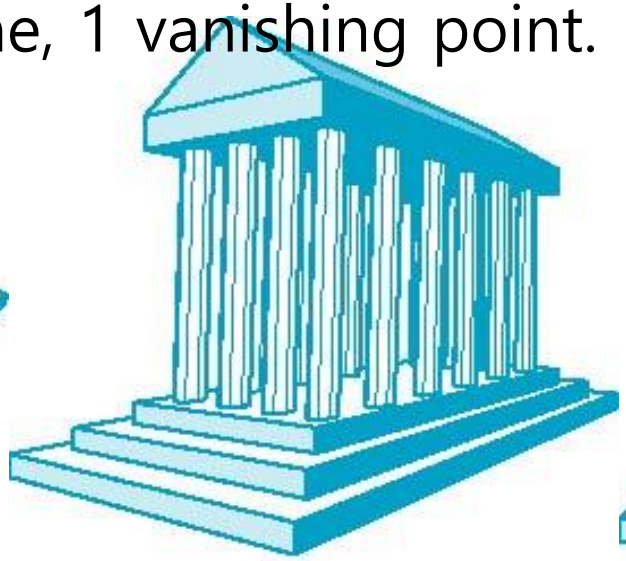


1-,2-,3-Point Perspective

- Three-point perspectives – no principal face parallel to projection plane, 3 vanishing points.
- Two-point perspectives – on principal direction parallel to projection plane, 2 vanishing points.
- One-point perspective – one principal face parallel to projection plane, 1 vanishing point.



3-point perspective



2-point perspective



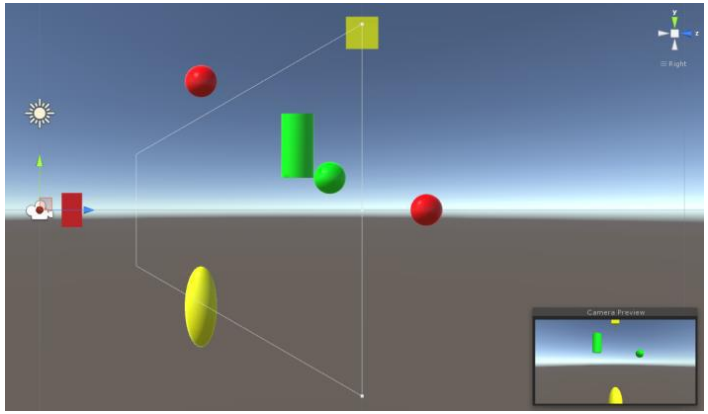
1-point perspective

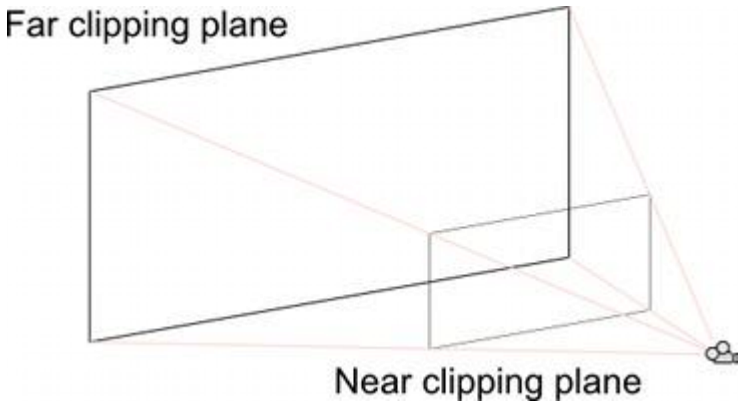
Perspective Projections Advantages and Disadvantages

- Objects further from viewer are projected smaller than the same sized objects closer to the viewer (*diminution*)
  - Looks realistic
- Equal distances along a line are not projected into equal distances (*nonuniform foreshortening*)
- Angles preserved only in planes parallel to the projection plane
- More difficult to construct by hand than parallel projections (but not more difficult by computer)

View Frustum

- View Frustum is the shape of the region that can be seen and rendered by a camera.

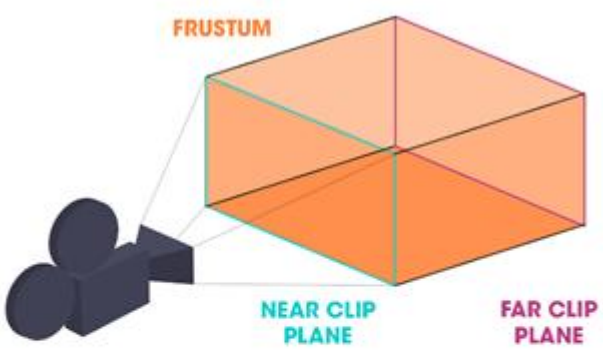




https://docs.unity3d.com/Manual/UnderstandingFrustum.html

View Frustum

- Orthographic projection projects the *rectilinear box viewing volume* onto the screen.
- The size of the object does not change with distance.
- Points are projected onto the z=0 plane towards the z- axis.



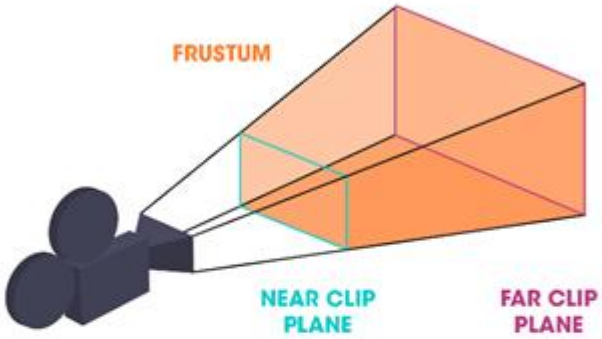
View frustum



http://latedreamer.blogspot.com/2017/08/unity-tutorial-tanks-part-2.html

View Frustum

- Perspective projection projects the *frustum (i.e., truncated pyramid)* viewing space onto the screen.
- Near objects appear larger, and object far away appear smaller.



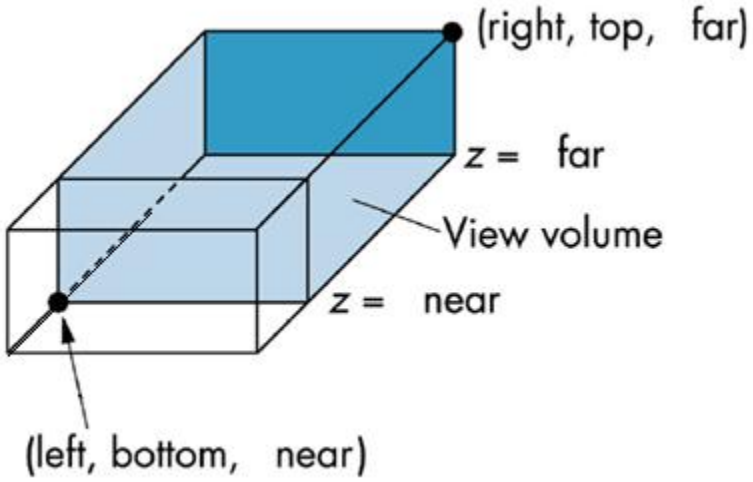
View frustum



http://latedreamer.blogspot.com/2017/08/unity-tutorial-tanks-part-2.html

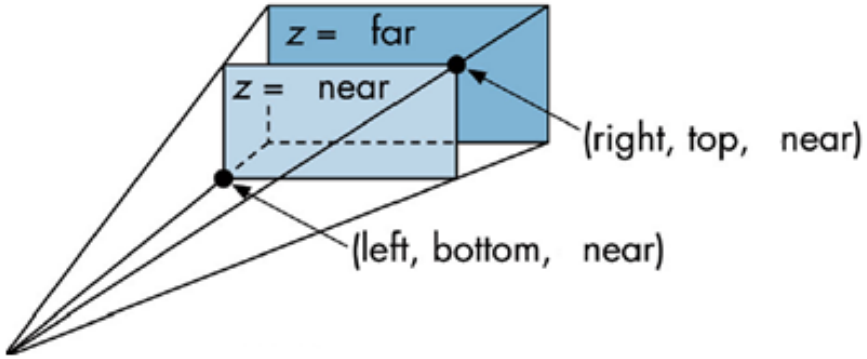
Unity Orthographic Projection

- **Matrix4x4.Ortho(left, right, bottom, top, zNear, zFar)**
  - Projection matrices in Unity follow OpenGL convention, i.e. clip space near plan is at z= -1 and far plan is at z= 1.
  - Camera's projection matrix, creates a projection of the area between left, right, top and bottom, with zNear and zFar as the near and far depth clipping planes into a cube going from **(left, bottom, near) = (-1, -1, -1) to (right, top, far) = (1, 1, 1).**



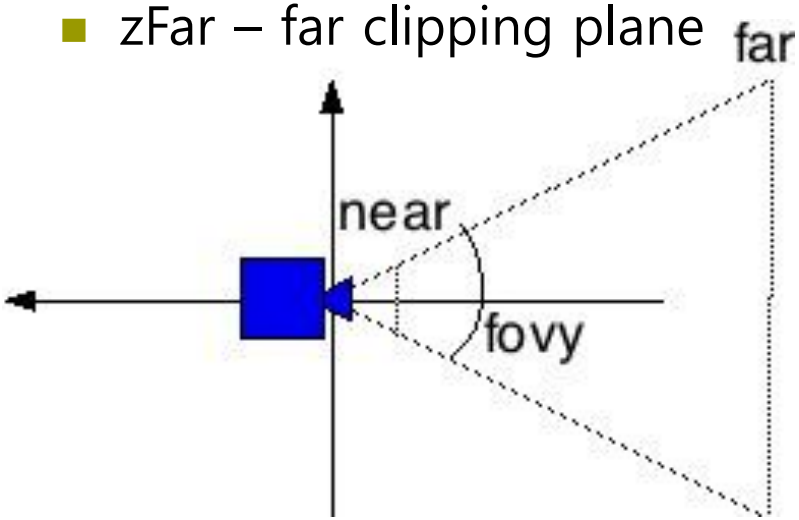
Unity Perspective Projection

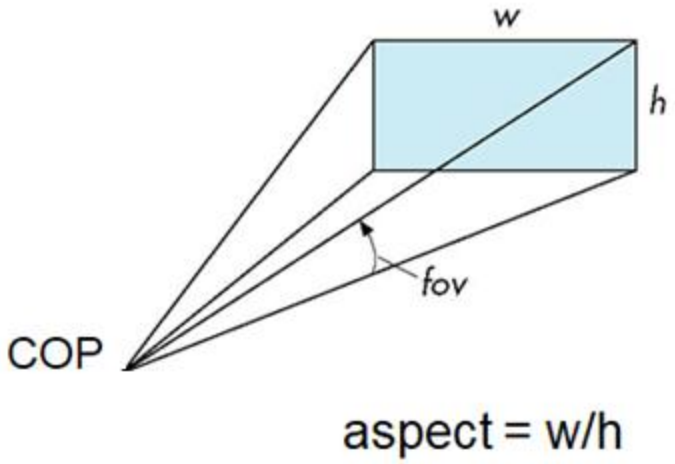
- **Matrix4x4.Frustum(left, right, bottom, top, zNear, zFar)**
  - The distance between near and far must be positive and is measured as the distance from the COP to the near/far plane.
  - The viewing volume is frustum (i.e., truncated pyramid).



Unity Perspective Projection

- **Matrix4x4.Perspective(fovy, aspect, zNear, zFar)** uses the **y-direction viewing angle (FOV)** and **the aspect ratio** (the value of the width of the **nearest** clipping plane divided by the height).
  - fovy – angle of field of view in Y-axis direction
  - aspect – the aspect ratio (width divided by height)
  - zNear – near clipping plane
  - zFar – far clipping plane





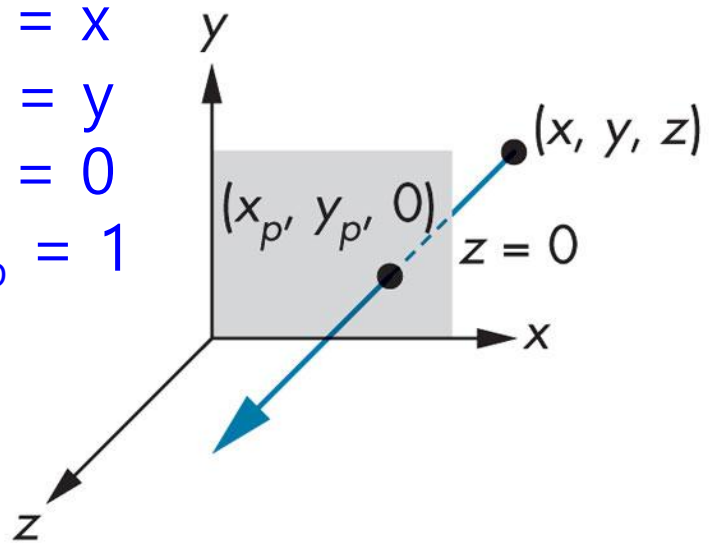
Orthographic Projection

- Orthographic projection
  - Special case of parallel projection in which the projector is orthogonal to the projection plane.
  - The focal length is infinite.

Orthographic projection

$$\mathbf{M}_{ortho} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{q} = \mathbf{Mp}$$

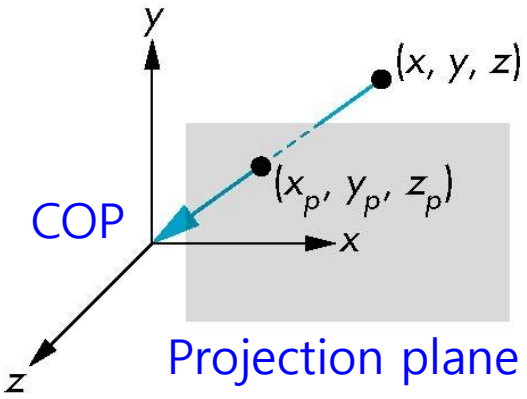
$x_p = x$
$y_p = y$
$z_p = 0$
$w_p = 1$

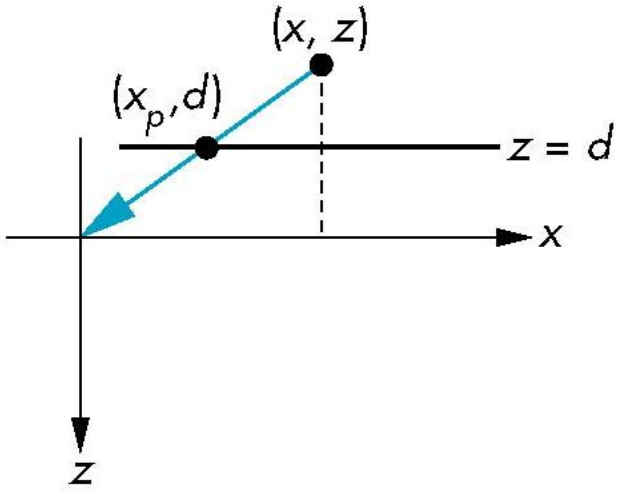


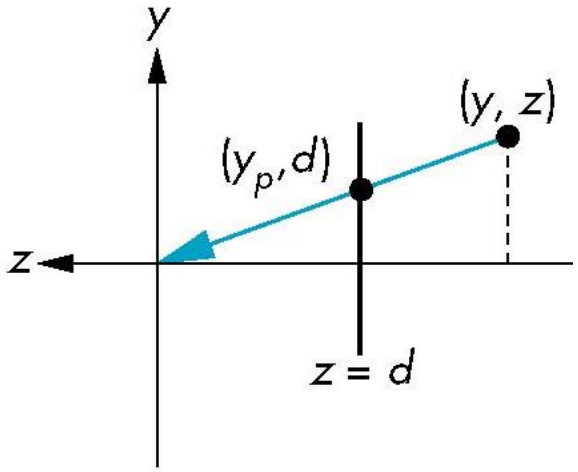
$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad \mathbf{M}_{ortho} \Rightarrow \mathbf{q} = \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}$$

Perspective Projection

- Perspective projection
  - Center of projection is located at the origin
  - Projection plane $z_p = d$







$$x_p = \frac{x}{z/d} \qquad y_p = \frac{y}{z/d} \qquad z_p = d$$

Perspective Projection

Perspective projection

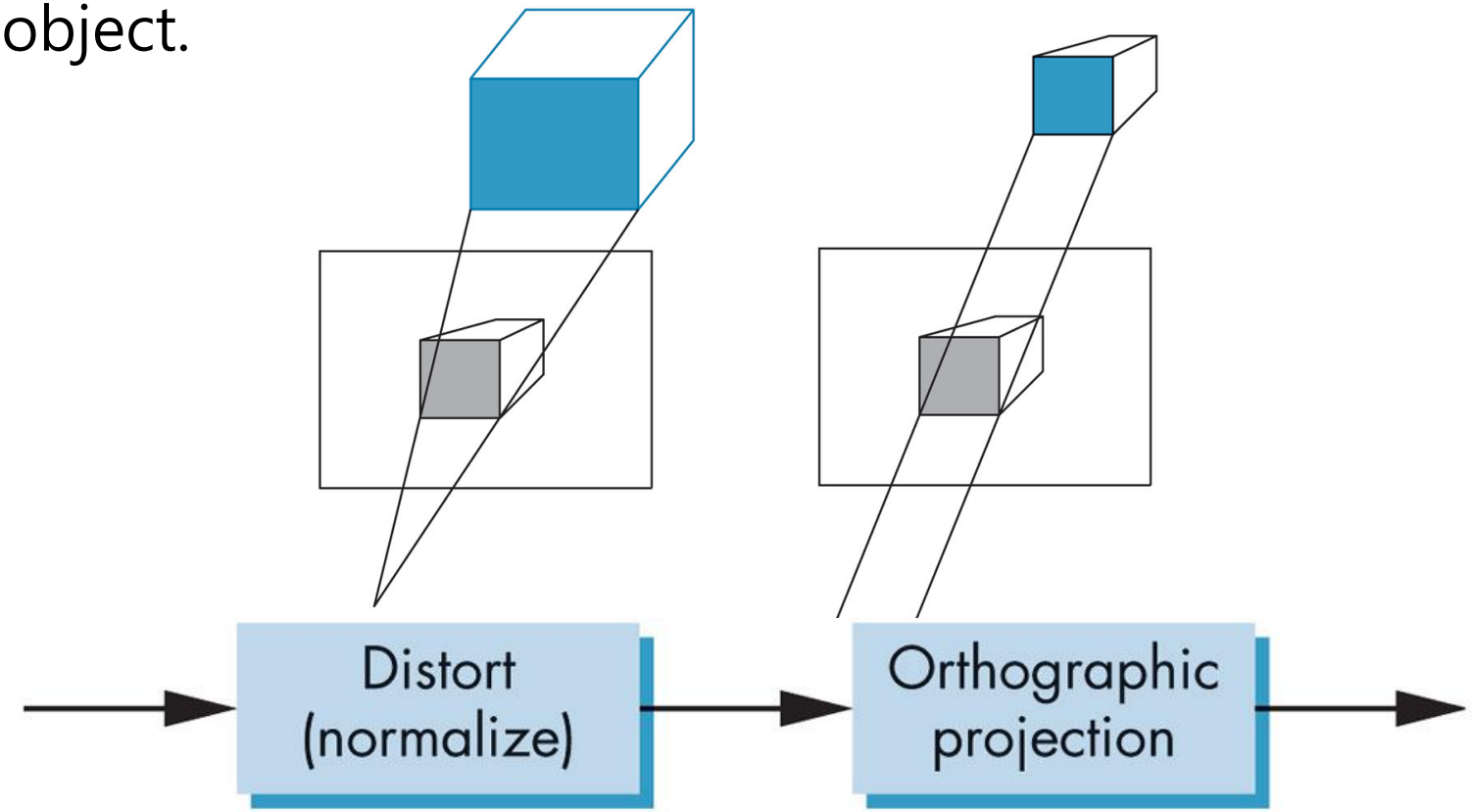
$$x_p = \frac{x}{z/d} \qquad \mathbf{q} = \mathbf{Mp} \qquad \mathbf{M_{pers}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

$$y_p = \frac{y}{z/d}$$

$$z_p = d = \frac{z}{z/d} \qquad \mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad \mathbf{M_{pers}} \Rightarrow \qquad \mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

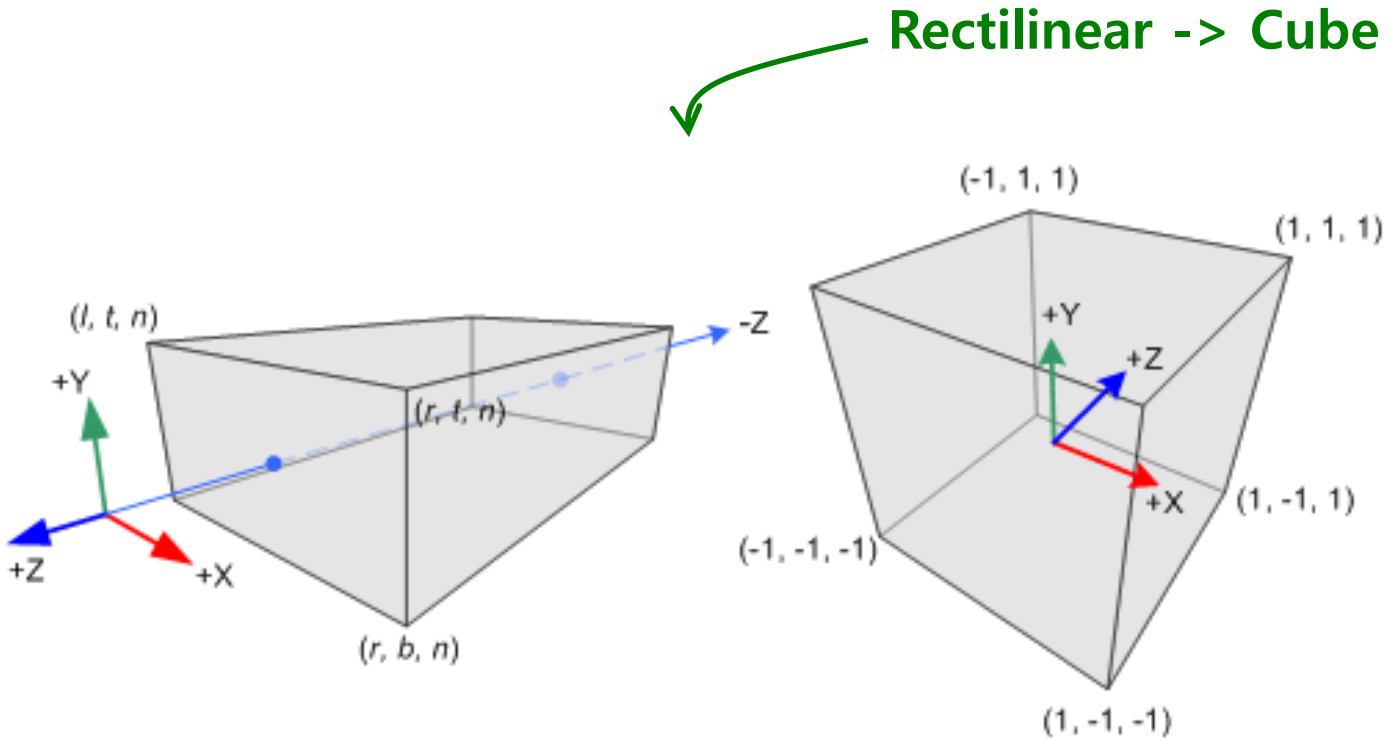
Projection Normalization

- Projection normalization converts all projections into orthogonal projections by distorting the objects such that the orthogonal projection of the distorted object is the same as the desired projection of the original object.



Orthogonal Projection Matrix

- Orthogonal projection maps a rectilinear view volume to Canonical view volume.



Orthogonal Projection Matrix

- Translate the center of viewing volume to the origin

$$T\left(\frac{-(left+right)}{2}, \frac{-(top+bottom)}{2}, \frac{-(-far-near)}{2}\right)$$

- Scale the viewing volume so that its length is 2x2x2

$$S\left(\frac{2}{(right-left)}, \frac{2}{(bottom-top)}, \frac{2}{(-far-(-near))}\right)$$

- **P=ST=**

$$\begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

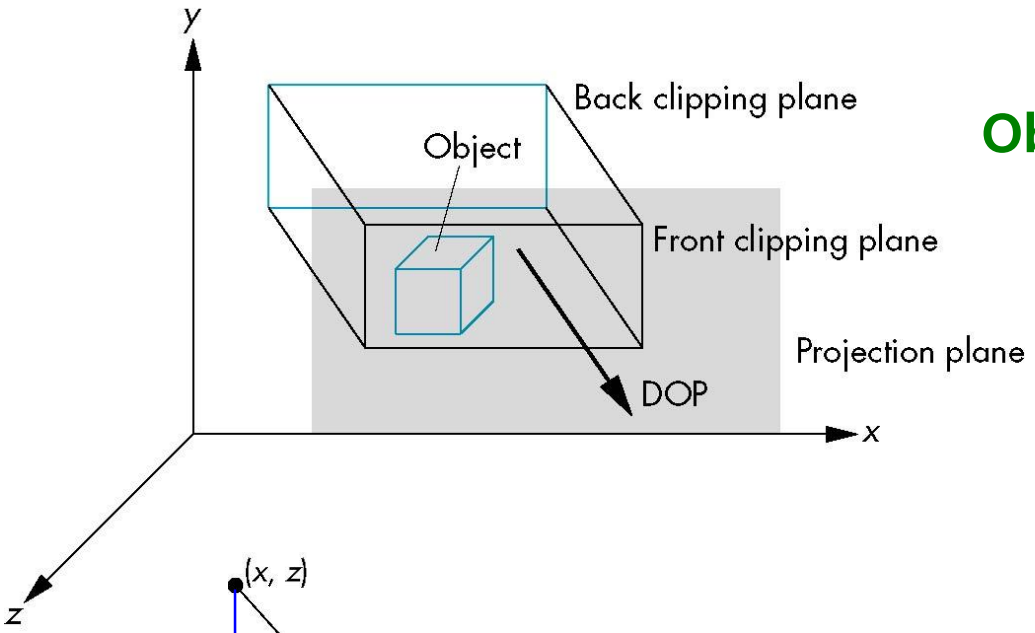
Orthogonal Projection Matrix

ortho=ST

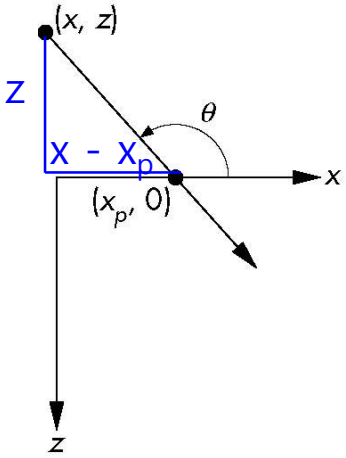
$$= \begin{pmatrix} \frac{2}{(right-left)} & 0 & 0 & 0 \\ 0 & \frac{2}{(top-bottom)} & 0 & 0 \\ 0 & 0 & -\frac{2}{(far-near)} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -\frac{(right+\text{left})}{2} \\ 0 & 1 & 0 & -\frac{(top+bottom)}{2} \\ 0 & 0 & 1 & \frac{(far+near)}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \frac{2}{(right-left)} & 0 & 0 & -\frac{(right+left)}{(right-left)} \\ 0 & \frac{2}{(top-bottom)} & 0 & -\frac{(top+bottom)}{(top-bottom)} \\ 0 & 0 & -\frac{2}{(far-near)} & -\frac{(far+near)}{(far-near)} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Oblique Projection Matrix



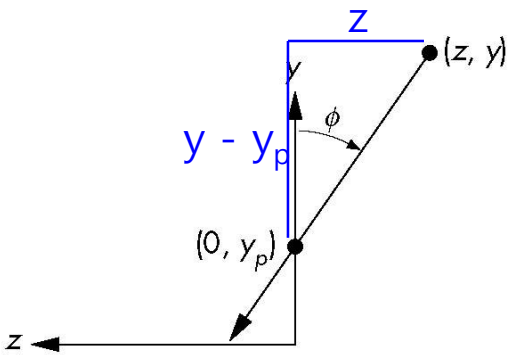
Oblique -> Orthogonal



top view

$$\tan\theta = \frac{z}{x - x_p}$$

$$x_p = x - z\cot\theta$$



side view

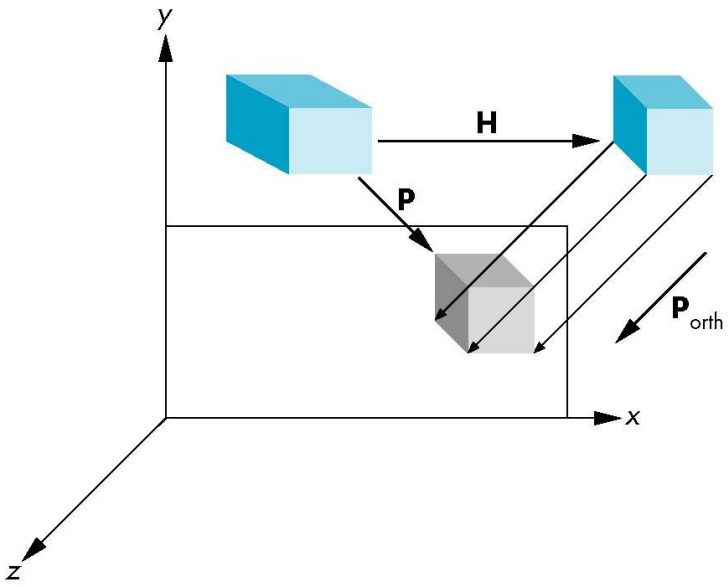
$$\tan\phi = \frac{z}{y - y_p},$$

$$y_p = y - z\cot\phi$$

Oblique Projection Matrix

- *xy* shear (*z* values unchanged)

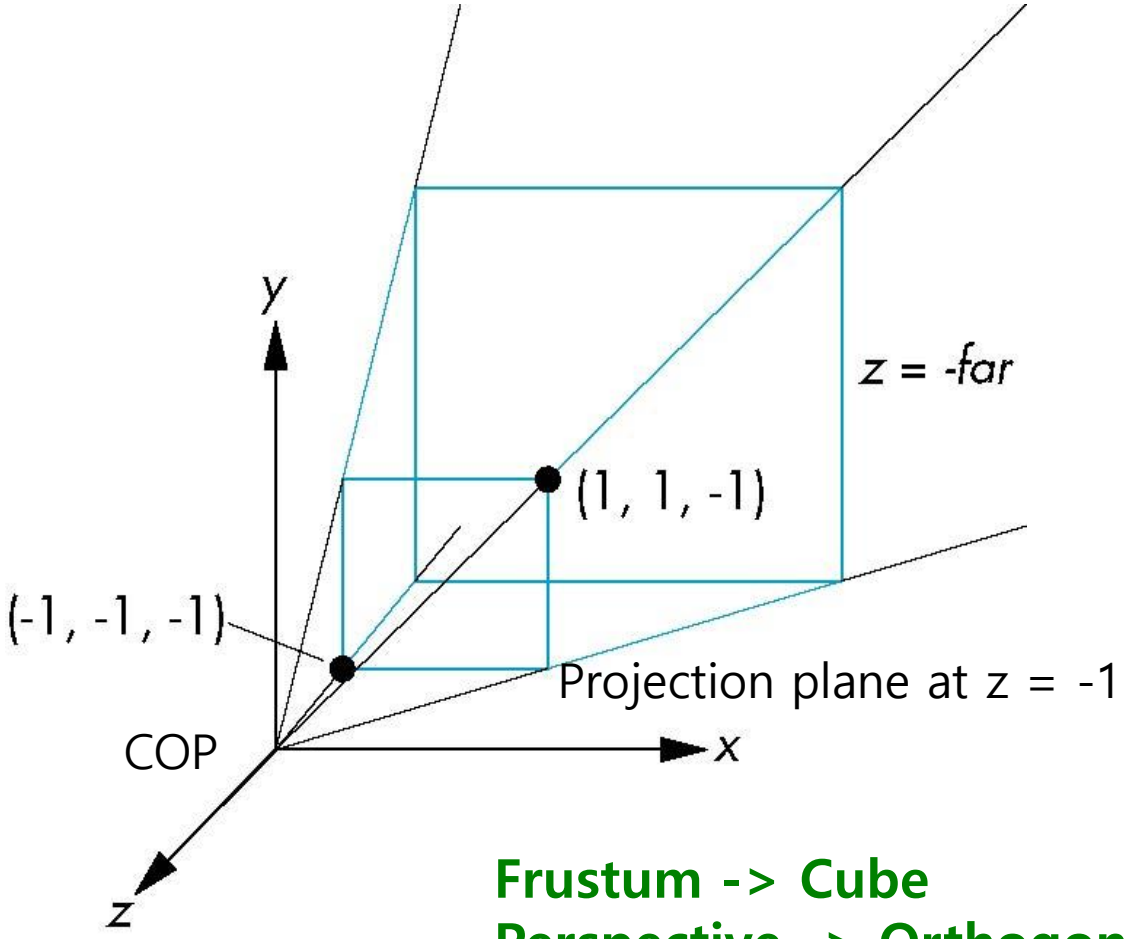
$$\mathbf{H}(\theta,\phi) = \begin{bmatrix} 1 & 0 & -\cot\theta & 0 \\ 0 & 1 & -\cot\varphi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



- $\mathbf{P} = \mathbf{M}_{ortho}\ \mathbf{H}(\theta,\phi)$

- General case: $\mathbf{P} = \mathbf{M}_{ortho}\ \mathbf{ST}\ \mathbf{H}(\theta,\phi)$

Perspective Projection Matrix



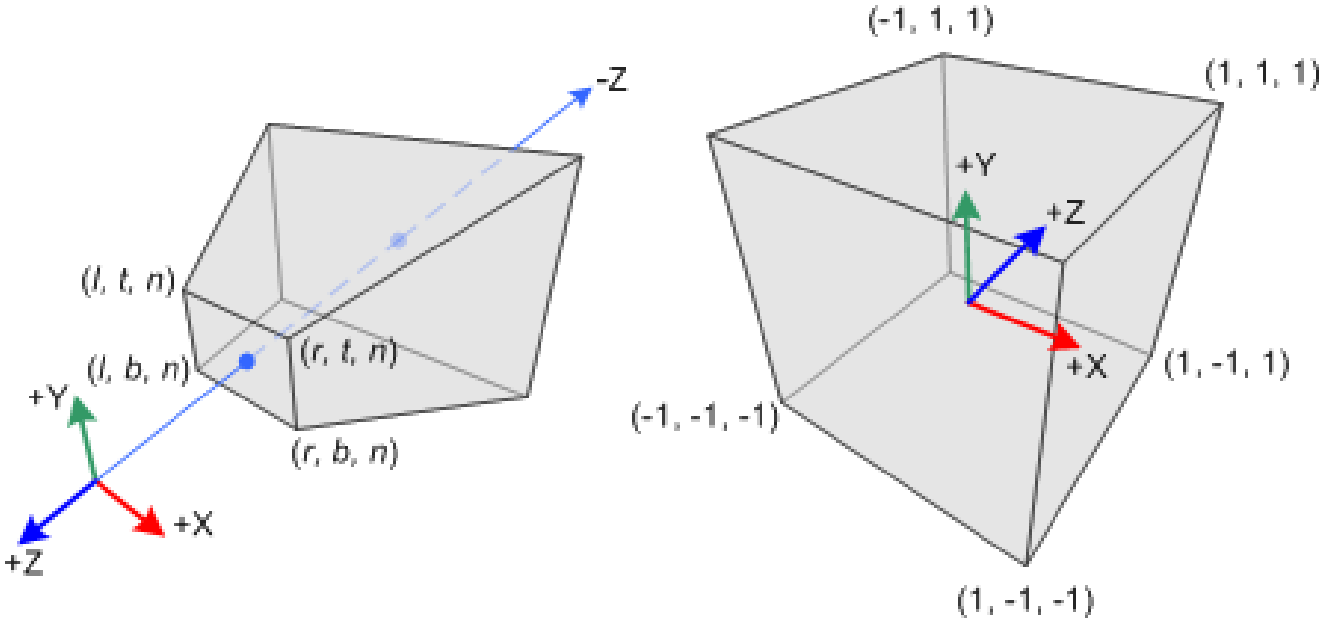
Frustum -> Cube
Perspective -> Orthogonal

Perspective Projection Matrix

- Perspective projection maps a frustum view volume to Canonical view volume.

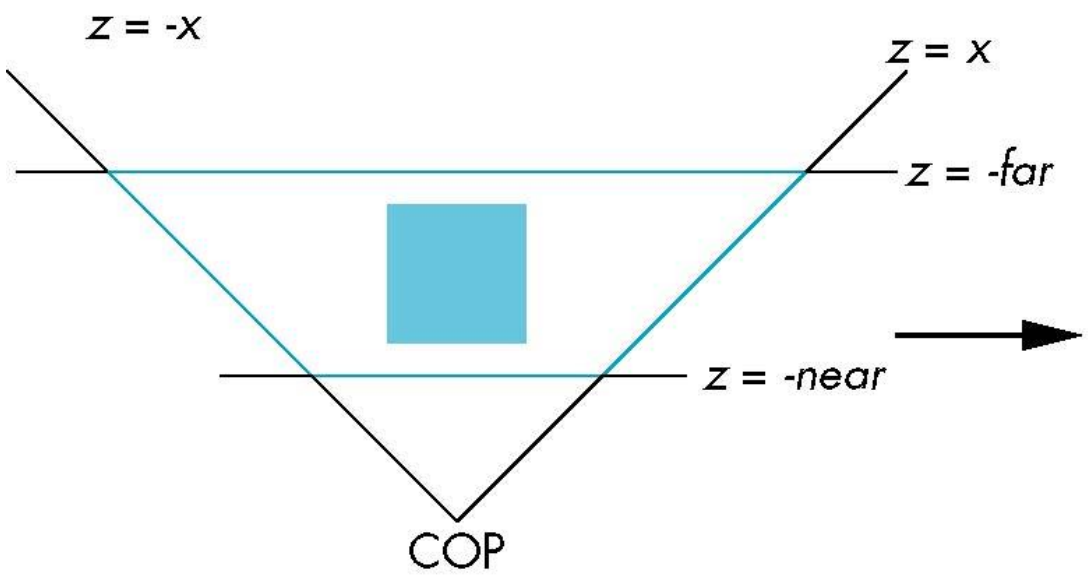
  [l, r] => [-1, 1], [b, t] => [-1, 1], [-n, -f] => [-1, 1]

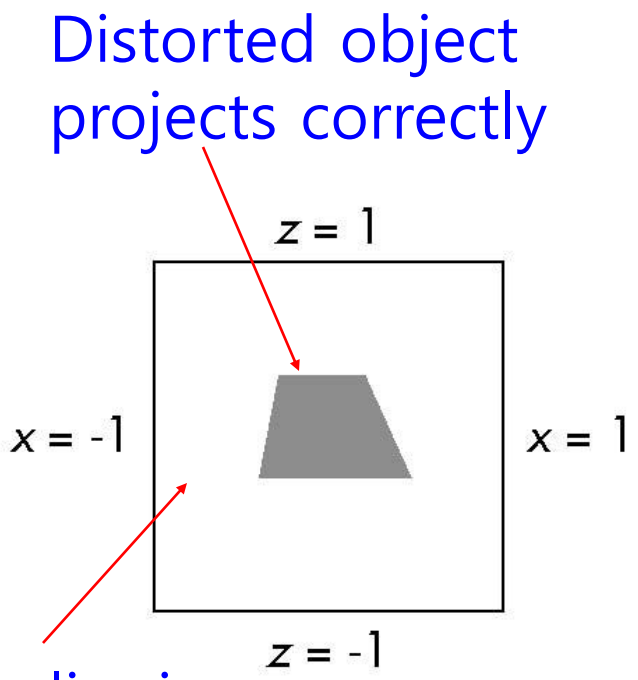
  [n, f] => [1, -1]



Perspective Projection Matrix

- Perspective normalization





$x = \pm z \rightarrow \pm 1$

$y = \pm z \rightarrow \pm 1$

$z = near/far \rightarrow \pm 1$

Perspective Projection Matrix

- Perspective normalization converts perspective projection to orthogonal projection.
  - Perspective projection matrix with the projection plane as z = -1, and the center of projection as the origin, M

$$\mathbf{M}_{pers} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

  - The field of view is fixed at 90 degrees by making the side of the viewing volume as 45 degree.

$$x = \pm z$$

$$y = \pm z$$

Perspective Projection Matrix

- N matrix:

$$\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- p′=Np:

$$x' = x,\ \ y' = y,\ \ z' = \alpha z + \beta,\ \ w' = -z$$

- Perspective division, p′->p″:

$$\Rightarrow x'' = -\frac{x}{z},\ \ y'' = -\frac{y}{z},\ \ z'' = \frac{\alpha z + \beta}{-z}$$

Perspective Projection Matrix

- If x = ± z, x″ = ± 1
- If y = ± z, y″ = ± 1
- If far plane z = -far, $z'' = \frac{\alpha(-far) + \beta}{far} = 1$

  If near plane z = -near, $z'' = \frac{\alpha(-near) + \beta}{near} = -1$

- To become z″ -> ± 1, select α and β: (-near, -1) & (-far, 1)

$$\alpha = -\frac{far + near}{far - near}$$

$$\beta = -\frac{2\,far\ near}{far - near}$$

Perspective Projection Matrix

$$\alpha(-far) + \beta = far \;\&\; \alpha(-near) + \beta = -near$$

$$\beta = -near + \alpha near$$

$$\alpha(-far) + (-near + \alpha near) = far$$

$$\alpha(near - far) = near + far$$

$$\alpha = \frac{near + far}{near - far} = -\frac{far + near}{far - near}$$

$$\beta = -near + \frac{near + far}{near - far} near$$

$$\beta = \frac{-near(near-far)}{near-far} + \frac{near(near+far)}{near-far}$$

$$\beta = \frac{-near(near-far)+near(near+far)}{near-far}$$

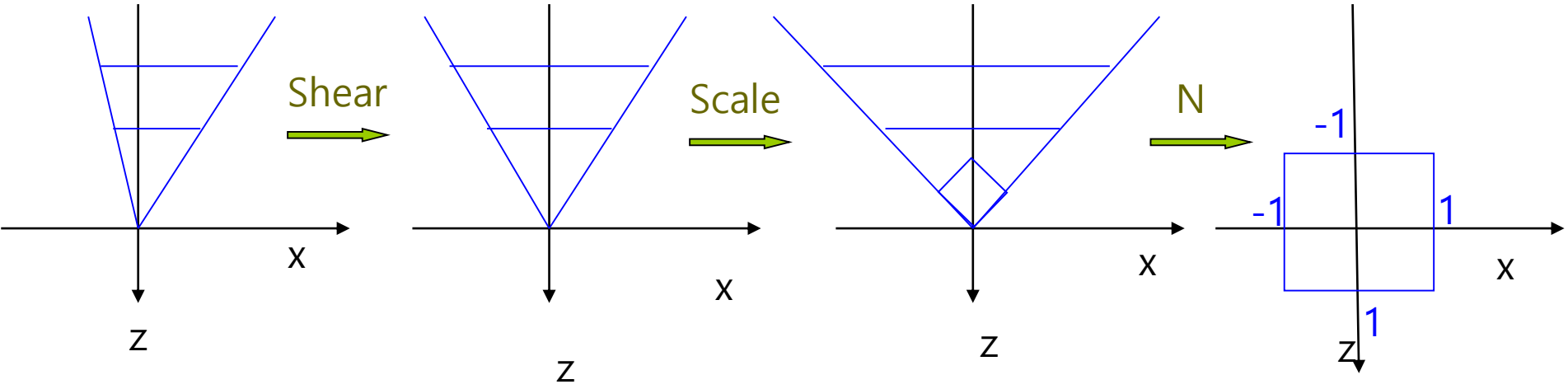
$$\beta = \frac{2near\ far}{near-far} = -\frac{2far\ near}{far-near}$$

$$\alpha = -\frac{far + near}{far - near}$$

$$\beta = -\frac{2\,far\ \ near}{far - near}$$

Perspective Projection Matrix

- Frustum(left, right, bottom, top, near, far)



Perspective Projection

- Shear $H(cot\theta, cot\emptyset) = H\left(\frac{right + left}{-2near}, \frac{top + bottom}{-2near}\right)$
- Then, $x = \pm\frac{right-left}{-2near}, y = \pm\frac{top-bottom}{-2near}, z = -near, z = -far$
- Scale $S = S\left(\frac{2near}{right - left}, \frac{2near}{top - bottom}, 1\right)$
- Then, $x = \pm z, y = \pm z$
- Normalize

$$\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$\alpha = -\frac{far + near}{far - near}$$

$$\beta = -\frac{2far\ near}{far - near}$$

Perspective Projection Matrix

Frustum=NSH

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{(far+near)}{(far-near)} & -\frac{2farnear}{(far-near)} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} \frac{2near}{(right-left)} & 0 & 0 & 0 \\ 0 & \frac{2near}{(top-bottom)} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & \frac{(right+left)}{2near} & 0 \\ 0 & 1 & \frac{(top+bottom)}{2near} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \frac{2near}{(right-left)} & 0 & \frac{(right+left)}{(right-left)} & 0 \\ 0 & \frac{2near}{(top-bottom)} & \frac{(top+bottom)}{(top-bottom)} & 0 \\ 0 & 0 & -\frac{(far+near)}{(far-near)} & -\frac{2farnear}{(far-near)} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

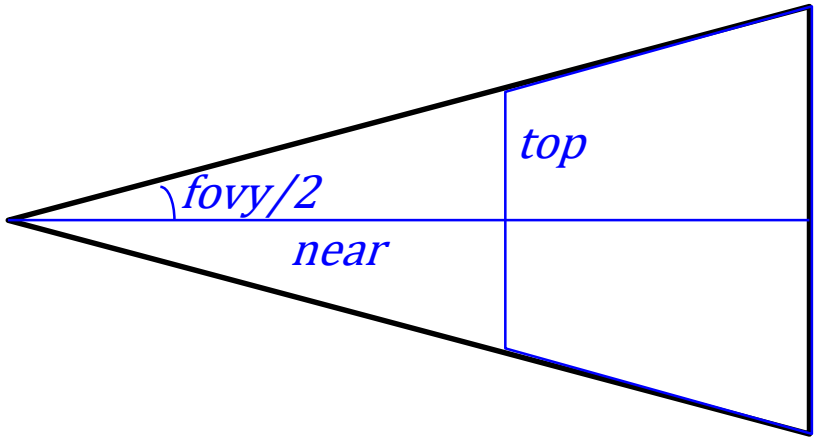
Perspective Projection Matrix

$$top = near * \tan\left(\frac{fovy}{2}\right)$$

$$bottom = -top$$

$$right = top * aspect$$

$$left = -right$$



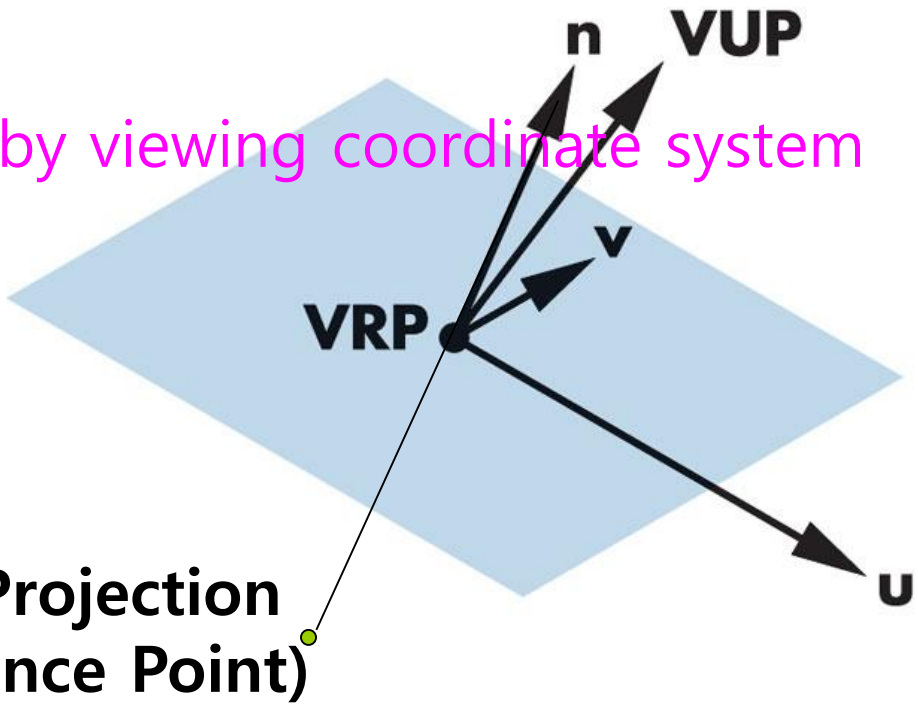
$$Perspective = \begin{pmatrix} \frac{near}{right} & 0 & 0 & 0 \\ 0 & \frac{near}{top} & 0 & 0 \\ 0 & 0 & -\frac{(far + near)}{(far - near)} & -\frac{2\ far\ near}{(far - near)} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

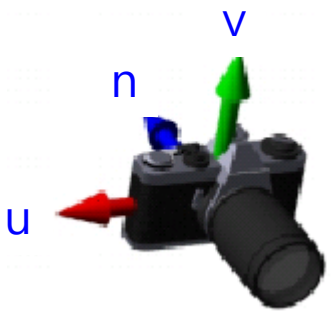
Computer Viewing

- Viewing
  - Set the position and direction of the camera.
    - Model-view transformation matrix
  - Apply the projection transformation matrix.
    - Projection transformation matrix
  - Clipping
    - View volume
- Default camera in Unity
  - Is placed at the **(0, 0, -10)** world coordinate system
  - Faces to the **positive z-axis** direction
  - By default, **perspective projection viewing frustum** is used

Camera Frame

- View reference point (VRP)
- View plane normal (VPN) n = VRP - PRP
- View-up vector (VUP)
- Side vector u = VUP x n
- Up vector v = n x u
- u, v, n normalize
- Camera frame is defined by viewing coordinate system (u'-v'-n') and VRP.





Camera Frame

- View-orientation matrix, M

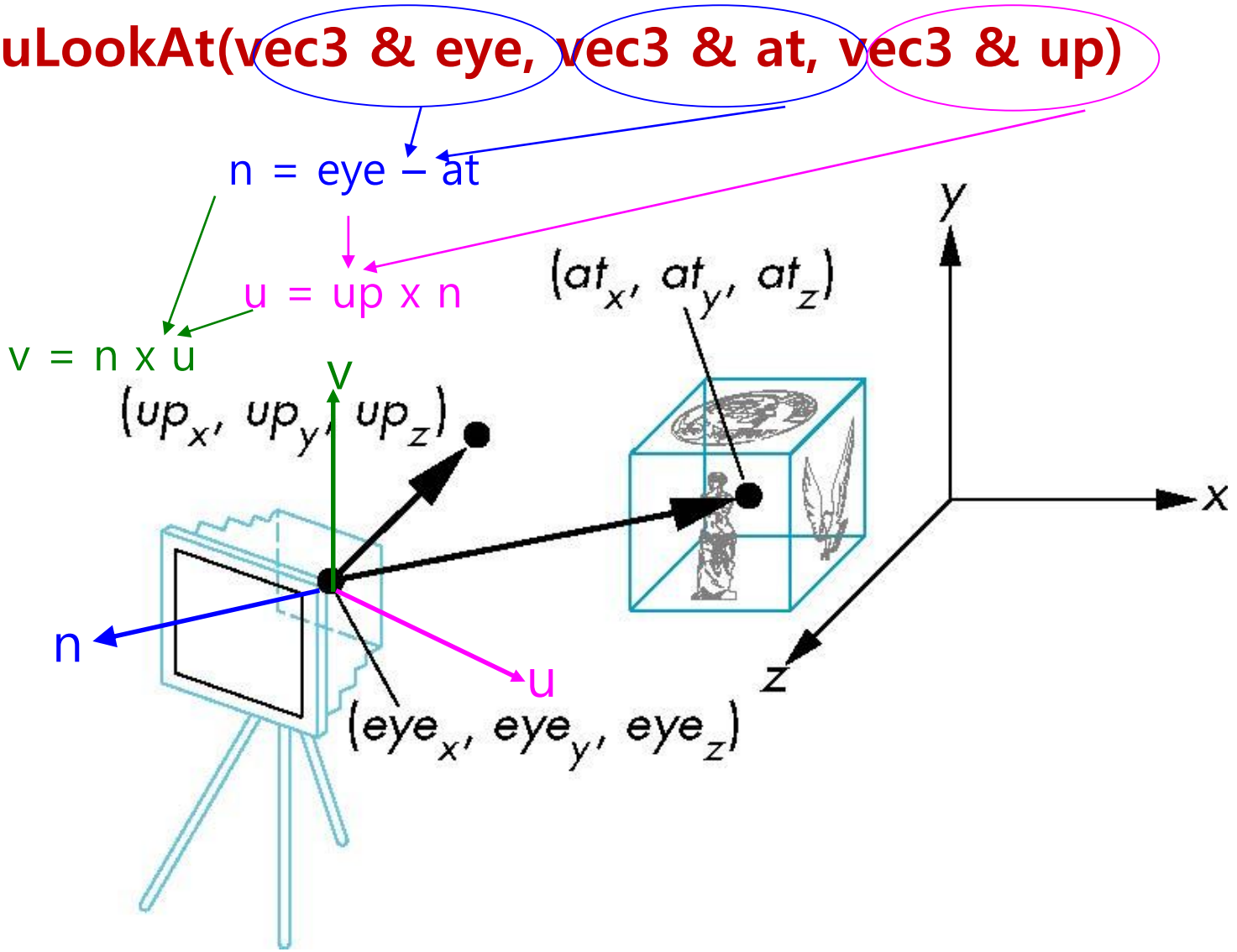
$$\mathbf{M} = \begin{bmatrix} u'_x & v'_x & n'_x & 0 \\ u'_y & v'_y & n'_y & 0 \\ u'_z & v'_z & n'_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotation matrix, $M^{-1} = M^T = R$
- Camera position in World frame: V = RT

$$\begin{bmatrix} u'_x & u'_y & u'_z & 0 \\ v'_x & v'_y & v'_z & 0 \\ n'_x & n'_y & n'_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} u'_x & u'_y & u'_z & -e \bullet u' \\ v'_x & v'_y & v'_z & -e \bullet v' \\ n'_x & n'_y & n'_z & -e \bullet n' \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

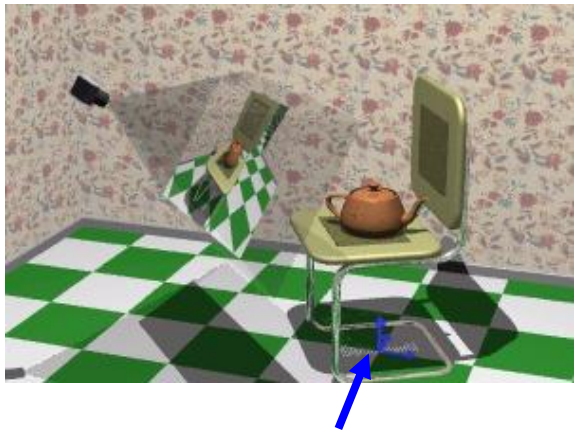
lookAt

- gluLookAt(vec3 & eye, vec3 & at, vec3 & up)



lookAt

- *Eye Point* : camera origin (in World Coordinate System)
- *Look-At* : the position where the camera is looking at (the center of the camera image)
- *Up-Vector* : the camera up vector (in World Coordinate System)



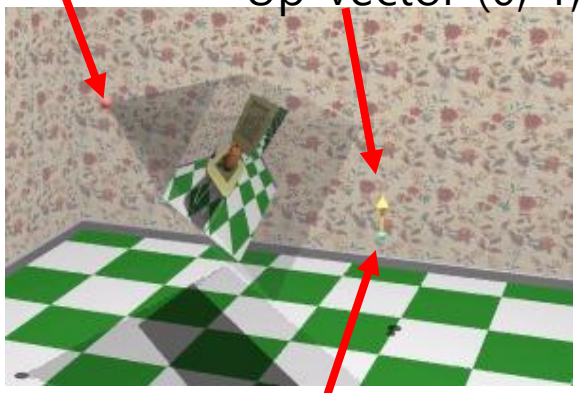
World space origin



Camera space origin

Eye point (c_x, c_y, c_z)

Up-vector (0, 1, 0)



Look-at point (p_x, p_y, p_z)

gluLookAt

```
void gluLookAt(GLdouble ex, GLdouble ey, GLdouble ez, GLdouble ax, GLdouble ay, GLdouble az,
    GLdouble ux, GLdouble uy, GLdouble uz) {
    GLdouble M[16]; GLdouble u[3], v[3], n[3]; GLdouble mag;

    n[0] = ex – ax; n[1] = ey – ay; n[2] = ez – az;          // n (camera frame Z)
    mag = sqrt(n[0]*n[0] + n[1]*n[1] + n[2]*n[2]);
    if (mag) { n[0] /= mag; n[1] /= mag; n[2] /= mag; }

    v[0] = ux; v[1] = uy; v[2] = uz;                          // u (camera frame X)
    u[0] = v[1]*n[2] – v[2]*n[1]; u[1] = -v[0]*n[2] + v[2]*n[0]; u[2] = v[0]*n[1] - v[1]*n[0];
    mag = sqrt(u[0]*u[0] + u[1]*u[1] + u[2]*u[2]);
    if (mag) { u[0] /= mag; u[1] /= mag; u[2] /= mag; }

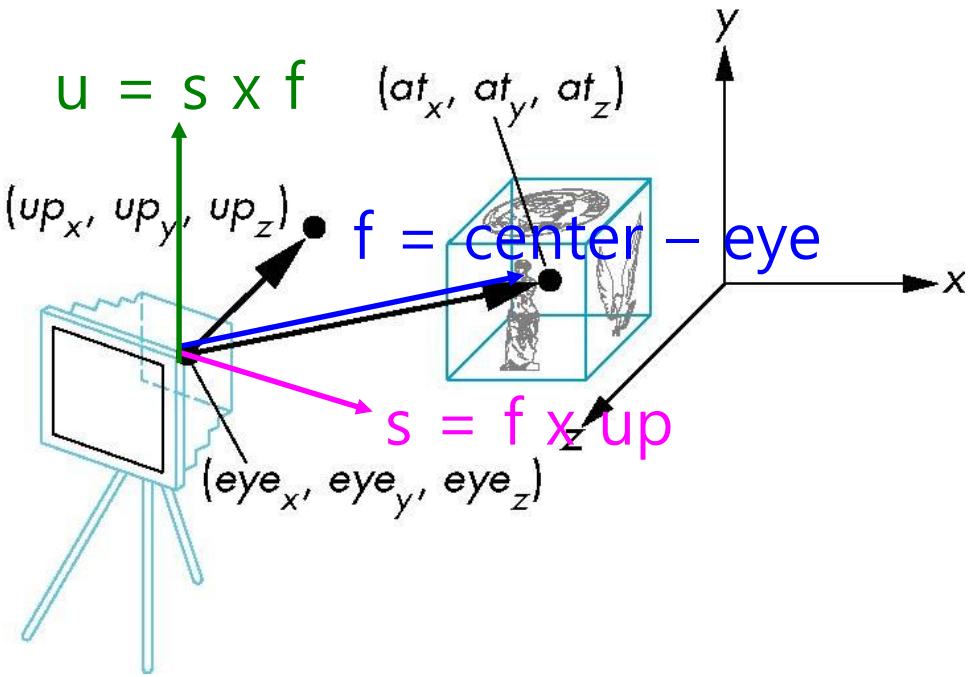
    v[0] = n[1]*u[2] – n[2]*u[1]; v[1] = -n[0]*u[2] + n[2]*u[0]; v[2] = n[0]*u[1] - n[1]*u[0];  // v (camera
    frame Y)
    mag = sqrt(v[0]*v[0] + v[1]*v[1] + v[2]*v[2]);
    if (mag) { v[0] /= mag; v[1] /= mag; v[2] /= mag; }

    M[0] = u[0];  M[4] = u[1];  M[8]  = u[2];    M[12] = 0.0;          // R
    M[1] = v[0];  M[5] = v[1];  M[9]  = v[2];    M[13] = 0.0;
    M[2] = n[0];  M[6] = n[1];  M[10] = n[2];    M[14] = 0.0;
    M[3] = 0.0;    M[7] = 0.0;    M[11] = 0.0;   M[15] = 1.0;
    glMultMatrix(M);

    glTranslated(-ex, -ey, -ez);                              // RT
}
```

glm::lookAt Matrix

```
template <typename T, precision P>
 GLM_FUNC_QUALIFIER tmat4x4<T, P> lookAtRH
 (tvec3<T, P> const & eye, tvec3<T, P> const & center, tvec3<T, P> const & up) {
    tvec3<T, P> const f(normalize(center - eye));
    tvec3<T, P> const s(normalize(cross(f, up)));
    tvec3<T, P> const u(cross(s, f));
    tmat4x4<T, P> Result(1);
    Result[0][0] = s.x;
    Result[1][0] = s.y;
    Result[2][0] = s.z;
    Result[0][1] = u.x;
    Result[1][1] = u.y;
    Result[2][1] = u.z;
    Result[0][2] = -f.x;
    Result[1][2] = -f.y;
    Result[2][2] = -f.z;
    Result[3][0] = -dot(s, eye);
    Result[3][1] = -dot(u, eye);
    Result[3][2] = dot(f, eye);
    return Result;
}
```



Unity Matrix4x4.LookAt

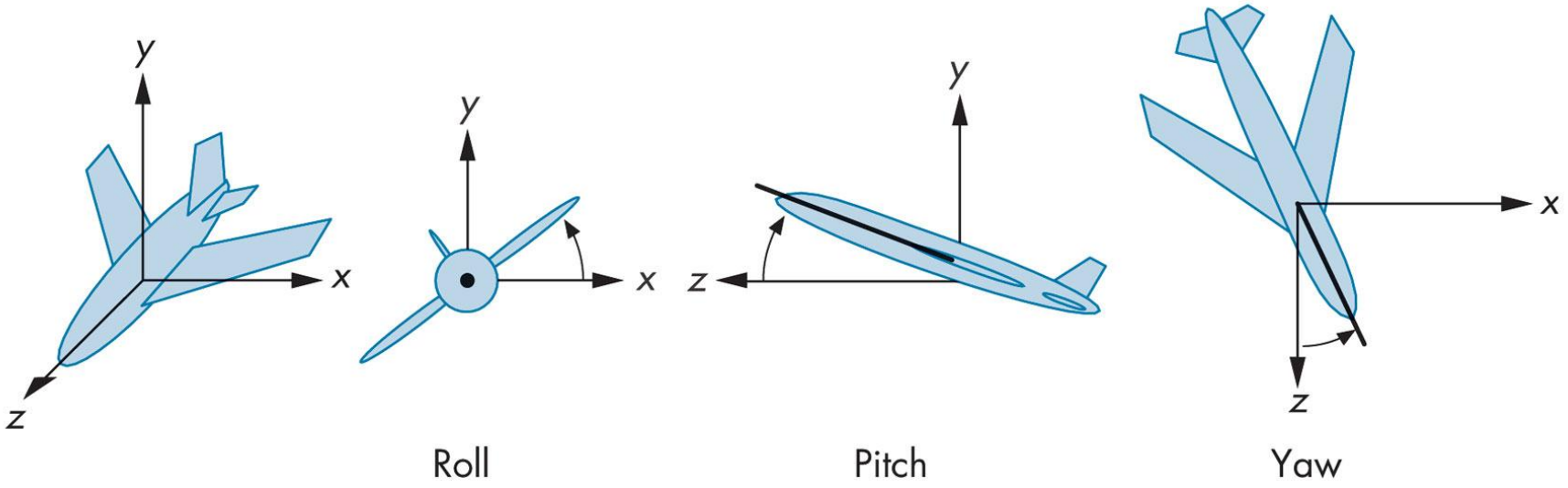
- **Matrix4x4.LookAt(Vector3 from, Vector4 to, Vector3 up)** creates a "look at" matrix.
  - Given a source point (from), a target point (to), and an up vector (up), computes a transformation matrix that corresponds to a camera viewing the target from the source, such that the right-hand vector is perpendicular to the up vector.
  - The resulting matrix corresponds to

Matrix4x4.TRS(from, Quaternion.LookRotation((to-from).normalized, up.normalized), Vector3.one)

  - Note that: **glm::lookat != Matrix4x4.LookAt**

Yaw, Pitch, Roll

- Yaw – Y-axis rotation
- Pitch – X-axis rotation
- Roll – Z-axis rotation



Elevation and Azimuth

- Azimuth – X-axis rotation (-180 ~ 180)
- Elevation – Y-axis rotation (-90 ~ 90)
- Twist angle – Z-axis rotation (-180 ~ 180)

Spherical Polar Coordinates System

