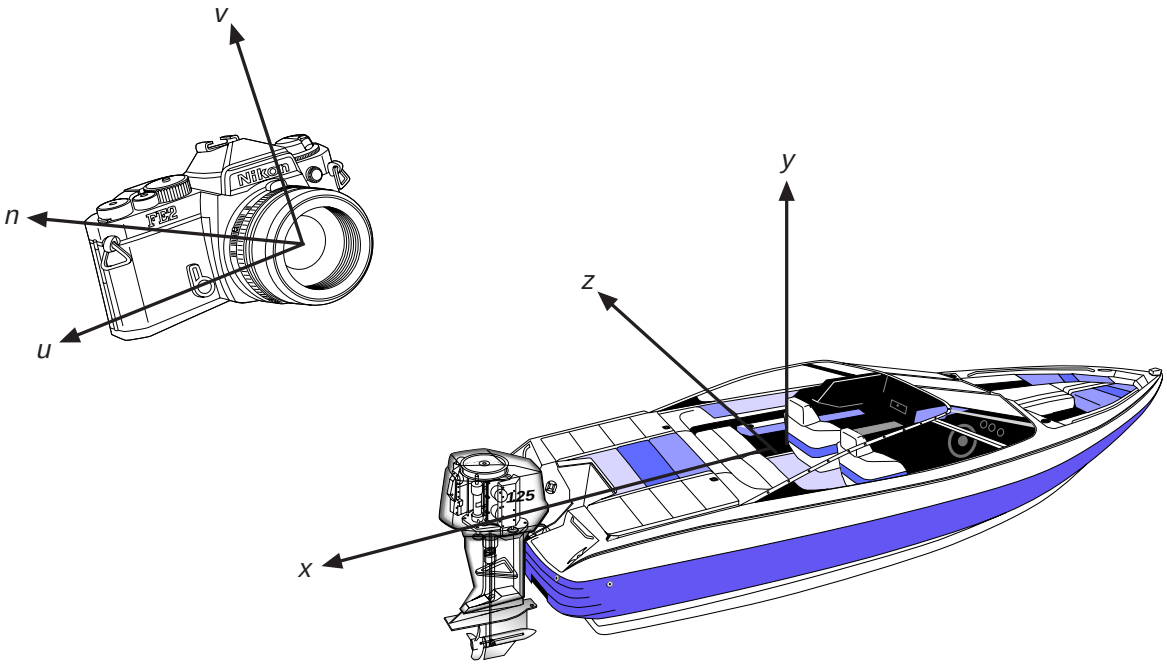
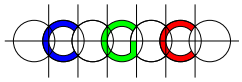
Specifying a View

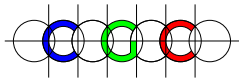
- ☆ How does a programmer specify the view parameters to a computer?
- ☆ Similar to taking a picture.





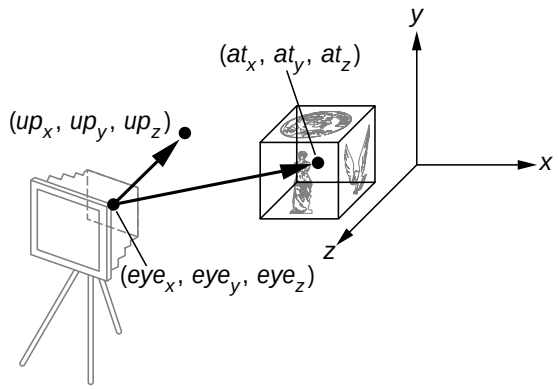
View Parameters

- ☆ Position of the camera.
  $pos = (pos_x, pos_y, pos_z)$.
- ☆ Orientation
  - Point at which camera is focused.
    $look = (look_x, look_y, look_z)$.
  - Orientation of the camera.
    $UP = (UP_x, UP_y, UP_z)$.
- ☆ Field of view.
  - Aspect ratio, angle of view.
- ☆ Depth of field
  - Near and far distances.
- ☆ Perspective/parallel projection.
- ☆ Determine focal distance.



Viewing in OpenGL

`gluLookAt`(*eye*, *at*, *up*)



- ☆ *eye = (eyex, eyey, eyez)*: Viewpoint position
- ☆ *at = (atx, aty, atz)*: Any point along the line of sight; typically the center of the image.

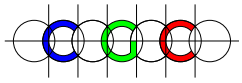
  $look = at \Leftrightarrow eye$
- ☆ *up = (upx, upy, upz)*: Up direction

`gluLookAt`(*eyex*, *eyey*, *eyez*, *atx*, *aty*, *atz*, *upx*, *upy*, *upz*)

Default command

`gluLookAt`(0.0, 0.0, 0.0, 0.0, 0.0, ⇔1.0, 0.0, 1.0, 0.0)

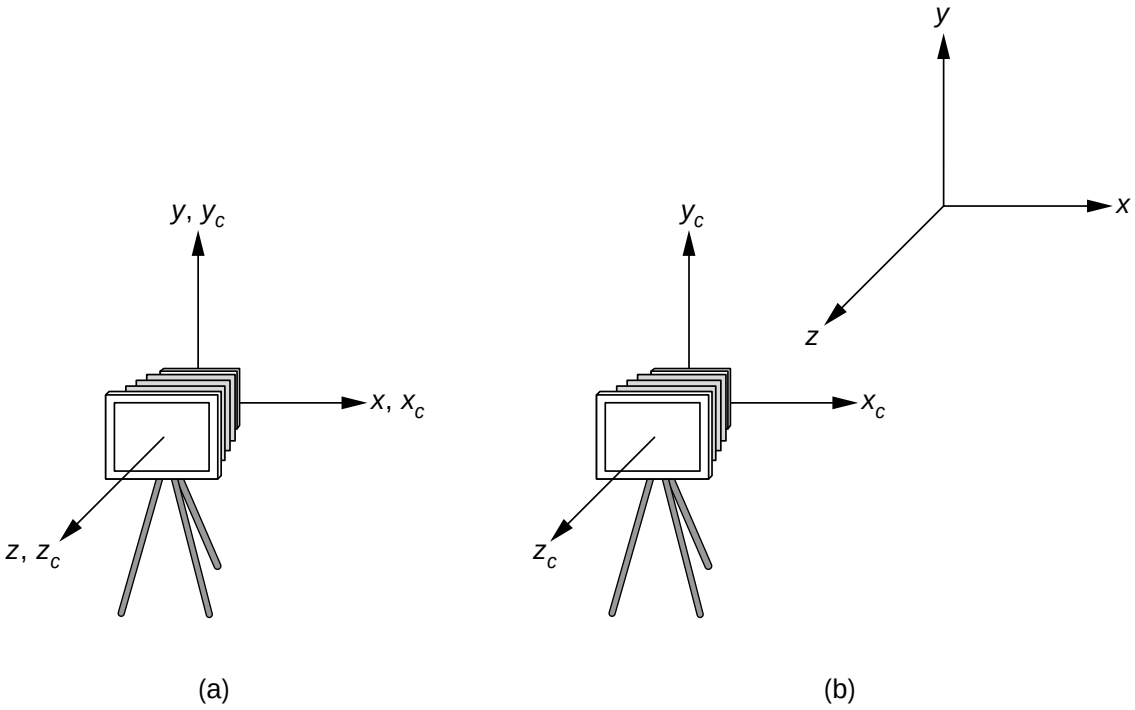
Typically appears at the beginning of the program.



View Parameters

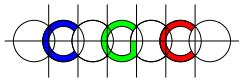
Default OpenGL parameters

- ☆ $pos = (0.0, 0.0, 0.0)$
- ☆ $look = (0.0, 0.0, \Leftrightarrow 1.0)$
- ☆ $UP = (0.0, 1.0, 0.0)$



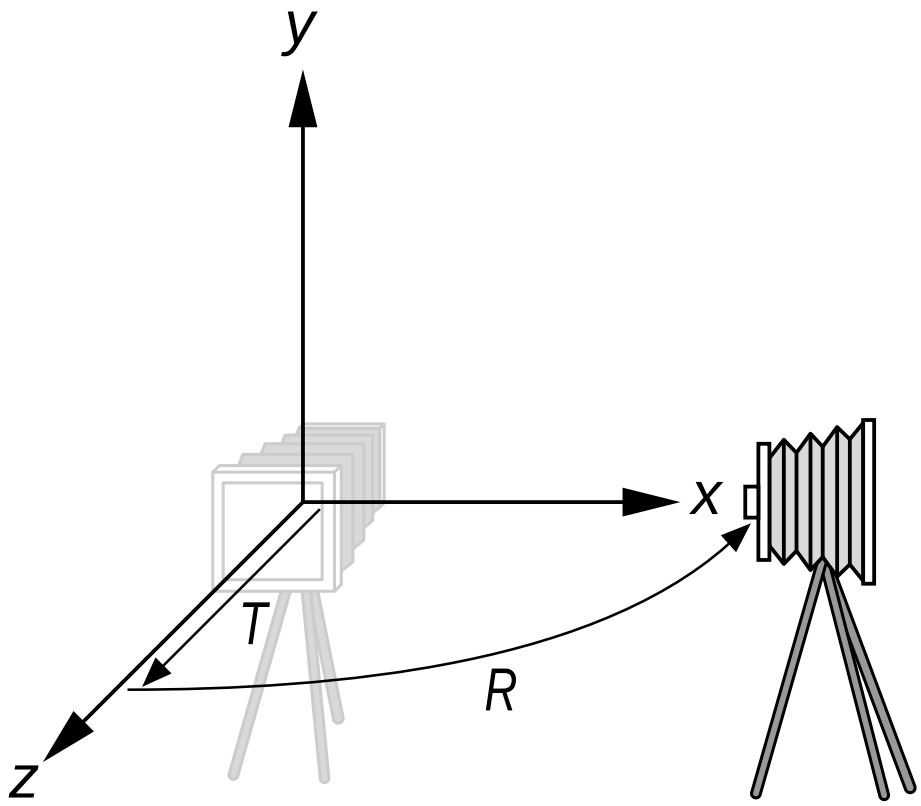
Change the view point to $(0, 0, 10)$

- ☆ Translate every object by $(0, 0, \Leftrightarrow 10)$
  `glTranslated` $(0.0, 0.0, \Leftrightarrow 10.0)$



View Parameters

Change $pos = (5, 0, 0)$ and $look = (\Leftrightarrow 1, 0, 0)$!

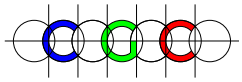


`glMatrixMode` (GL_MODELVIEW)

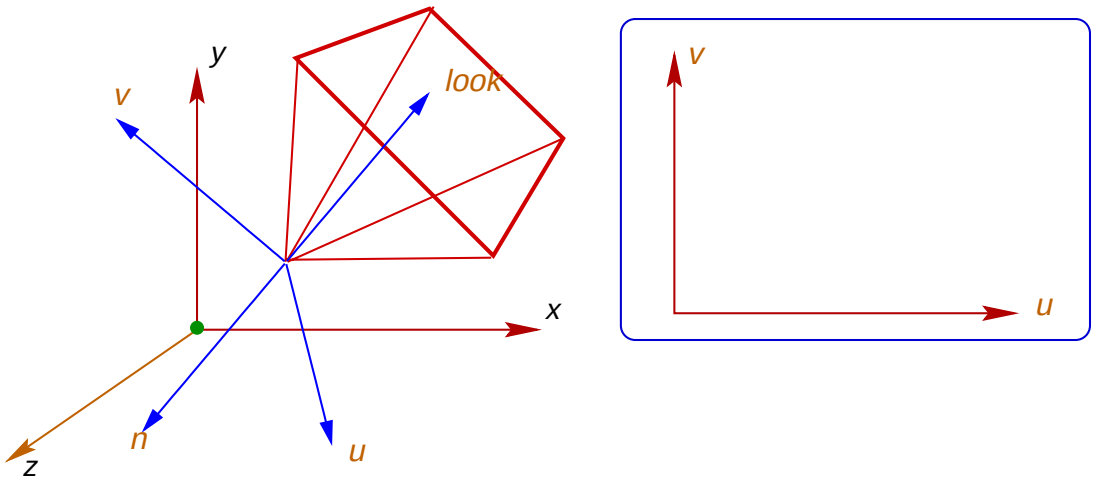
`glLoadIdentity` ()

`glTranslatef` $(0.0, 0.0, \Leftrightarrow 5)$

`glRotatef` $(\Leftrightarrow 90.0, 0.0, 1.0, 0.0)$



Coordinate Systems



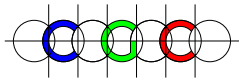
☆ *World coordinate system*

Standard x⇔, y⇔, z-coordinates.

☆ *Viewing coordinate system*

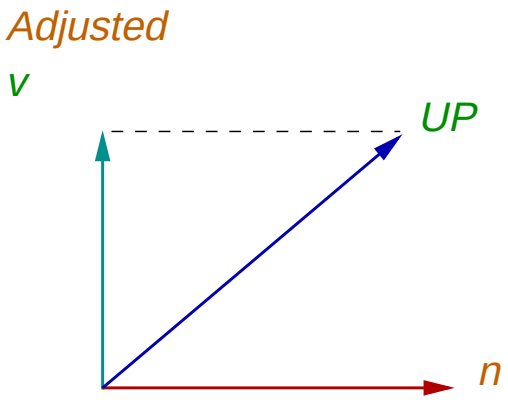
(u, v, n) coordinates.

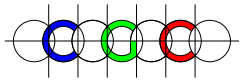
- Origin at *position*.
- *look* is in ⇔n-direction.
- v is determined by *UP* vector. *UP* is not necessarily normal to n, so a correction is required.
- u is normal to *UP* and n.
- Coordinate system satisfies the right-hand rule.



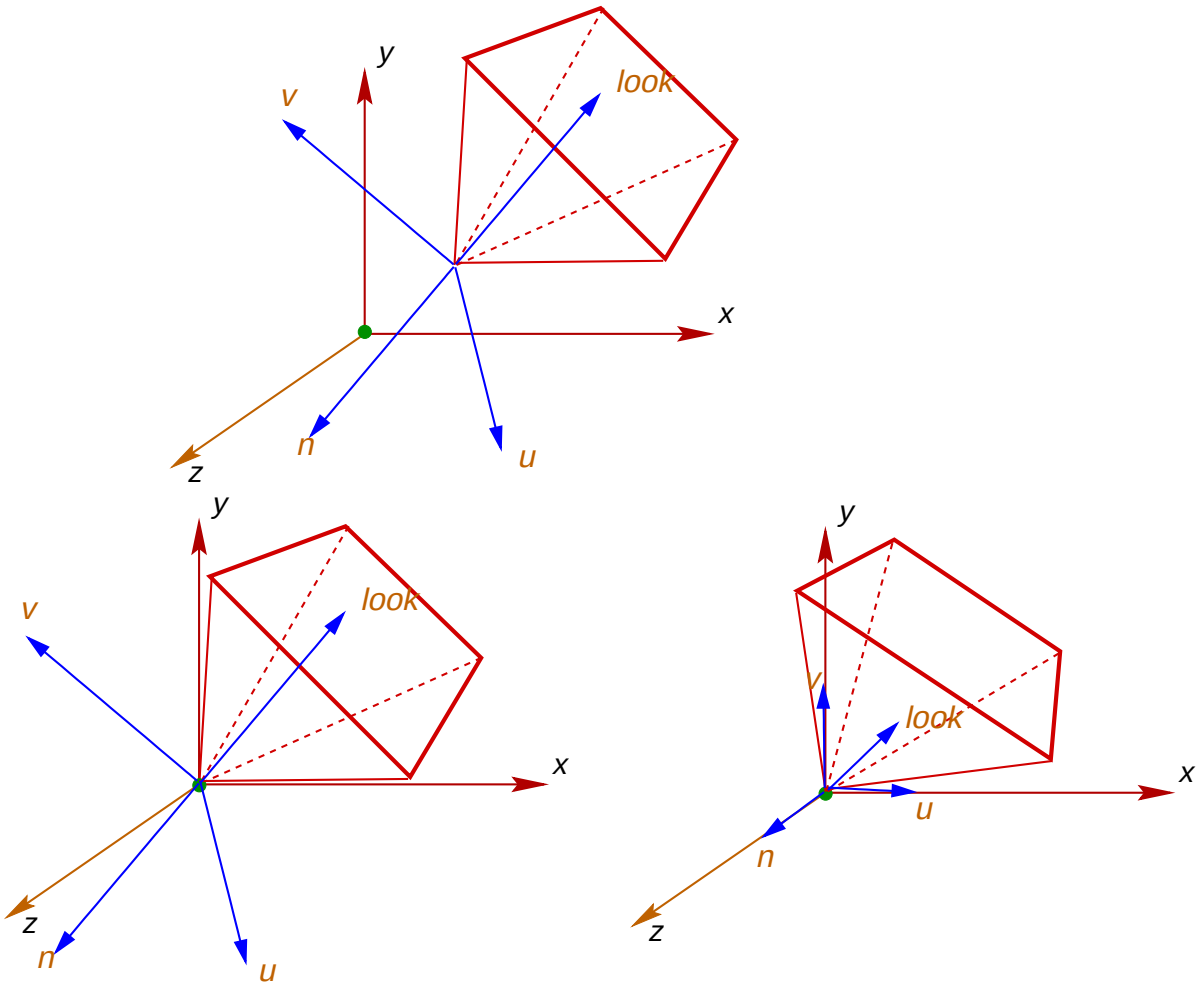
Computation with u, v, n

$$
\begin{aligned}
n &= \frac{\Leftrightarrow look}{\|look\|} \\
u &= \frac{look \times UP}{\|look \times UP\|} \\
v &= \frac{u \times look}{\|u \times look\|}
\end{aligned}
$$





World to View Coordinates



World to View Coordinates

- ☆ $pos = (pos_x, pos_y, pos_z)$,
- ☆ $u = (u_x, u_y, u_z)$,
- ☆ $v = (v_x, v_y, v_z)$,
- ☆ $n = (n_x, n_y, n_z)$

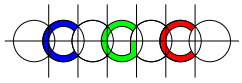
☆ Perform a translation T_C so that pos maps to the origin.

$$T_C = \begin{bmatrix} 0 & 0 & 0 & \Leftrightarrow pos_x \\ 0 & 0 & 0 & \Leftrightarrow pos_y \\ 0 & 0 & 0 & \Leftrightarrow pos_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

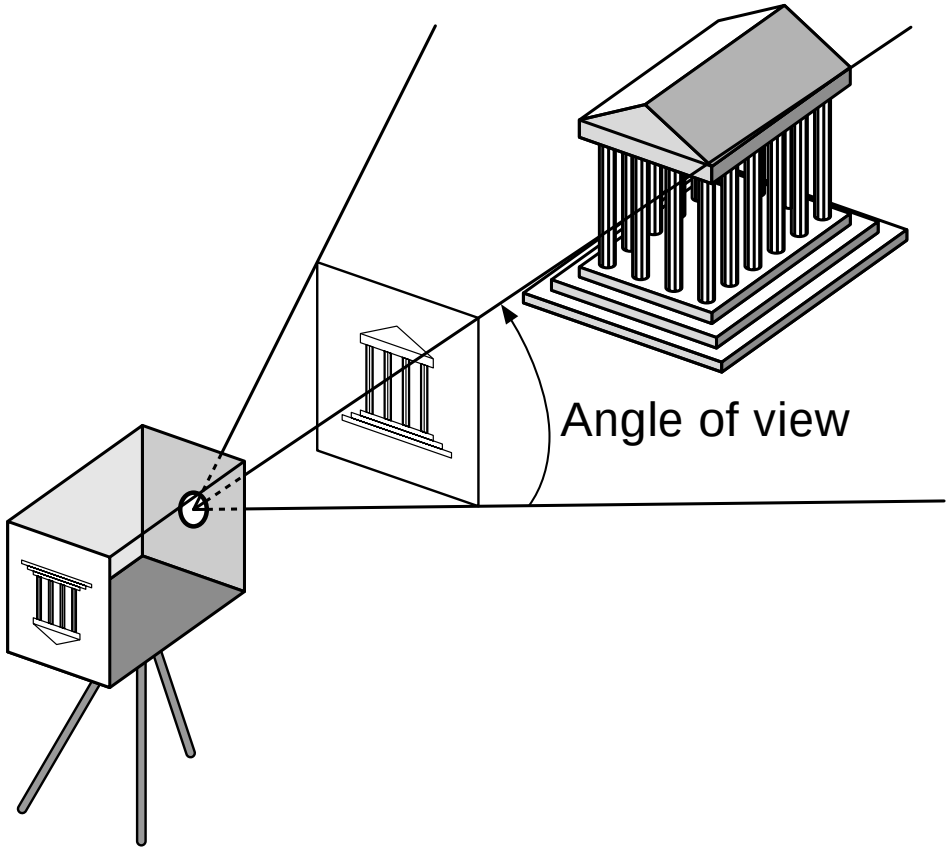
☆ Perform a rotation R_C so that $u \rightarrow x$, $v \rightarrow y$, and $n \rightarrow z$.

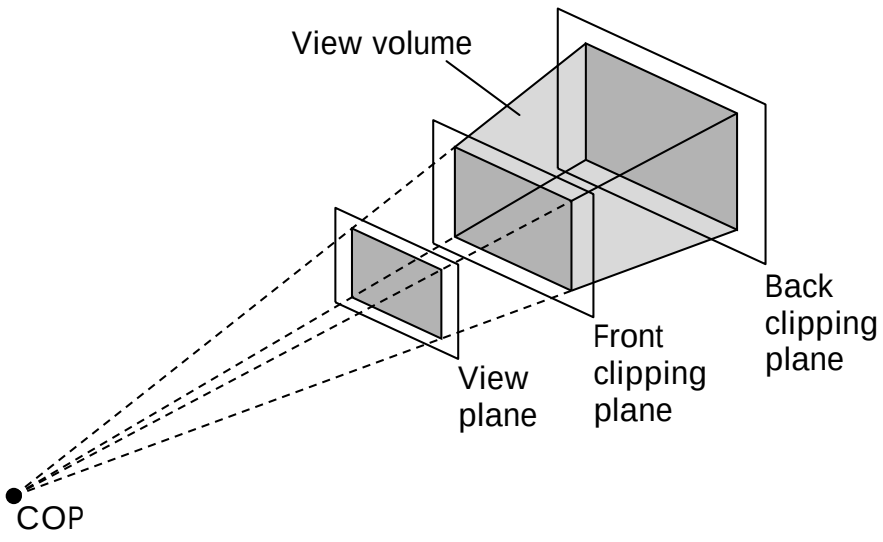
$$R_C = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

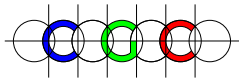
☆ $M_C = R_C \cdot T_C$.



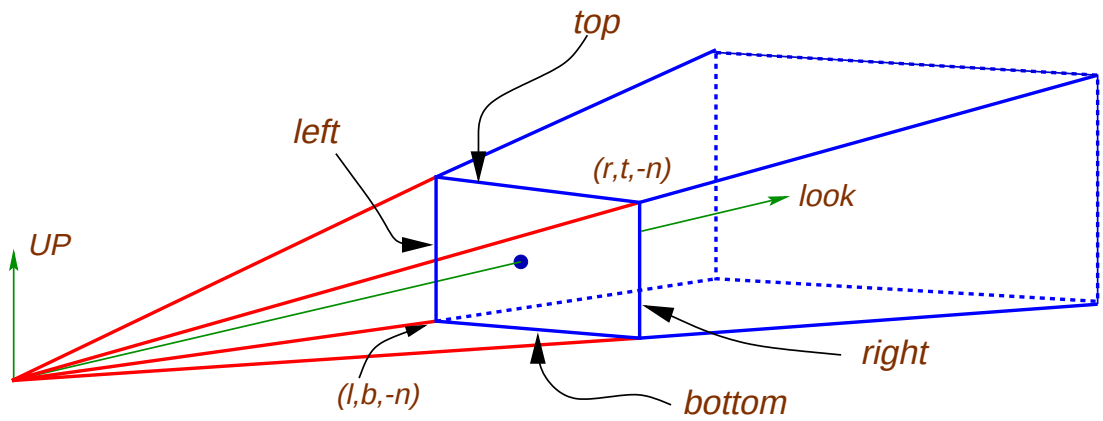
Perspective Projection







Specifying Views in OpenGL

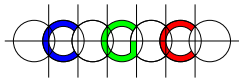


Viewing volume:

- ☆ Defines the region of visible objects.
- ☆ Objects lying outside the viewing volume are discarded.

  `glFrustum`(l, r, b, t, n, f)

- ☆ Frustum need not be symmetric with respect to the line of sight.



Perspective Projection

l: x-coordinate of the left edge of the near plane.

r: x-coordinate of the right edge of the near plane.

b: y-coordinate of the bottom edge of the near plane.

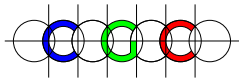
t: y-coordinate of the top edge of the near plane.

n: z-coordinate of the near plane.

f: z-coordinate of the far plane.

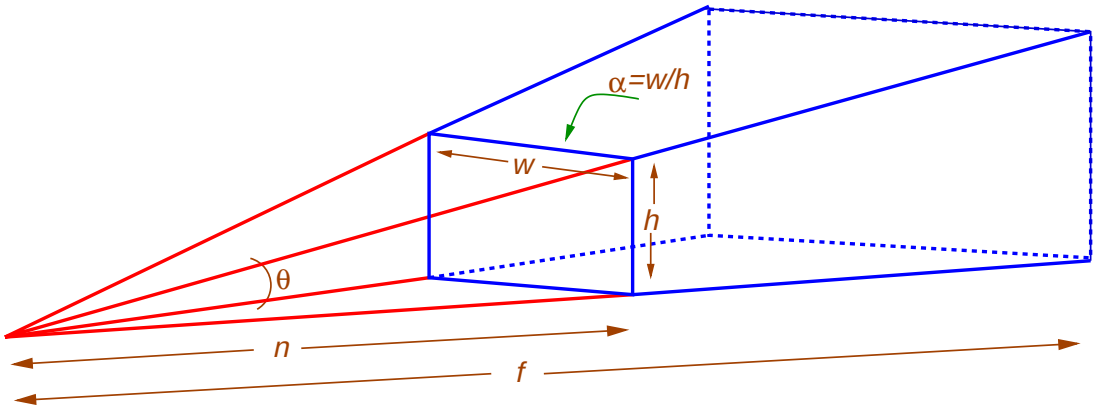
$(l, b, \Leftrightarrow n)$: left-bottom corner of the near plane.

$(r, t, \Leftrightarrow n)$: top-right corner of the near plane.



Perspective Projection

Estimating l, r, b, t is difficult!



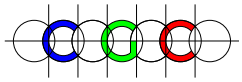
`gluPerspective`(θ, α, n, f)

θ: Angle of the field view; $\theta \in [0.0, 180.0]$.

α: Aspect ratio (x-length/y-length).

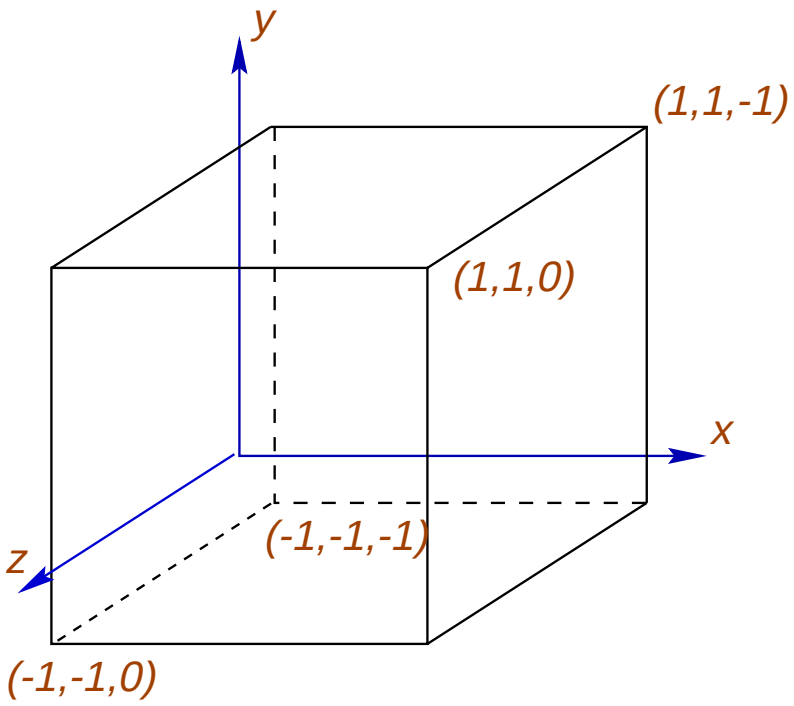
Line of sight is the ($\Leftrightarrow z$)-axis.

Frustum symmetric with respect to the line of sight.



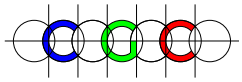
Canonical View

- $eye = (0, 0, 0)$, $look = (0, 0, \Leftrightarrow 1)$, $UP = (0, 1, 0)$.
- Viewing volume is always the parallel box $[\Leftrightarrow 1, +1] \times [\Leftrightarrow 1, +1] \times [\Leftrightarrow 1, 0]$



Simplifies clipping, projection, hidden surface removal

- Projection: ignore the z-coordinate
- Hidden surface removal: compare z-coordinates

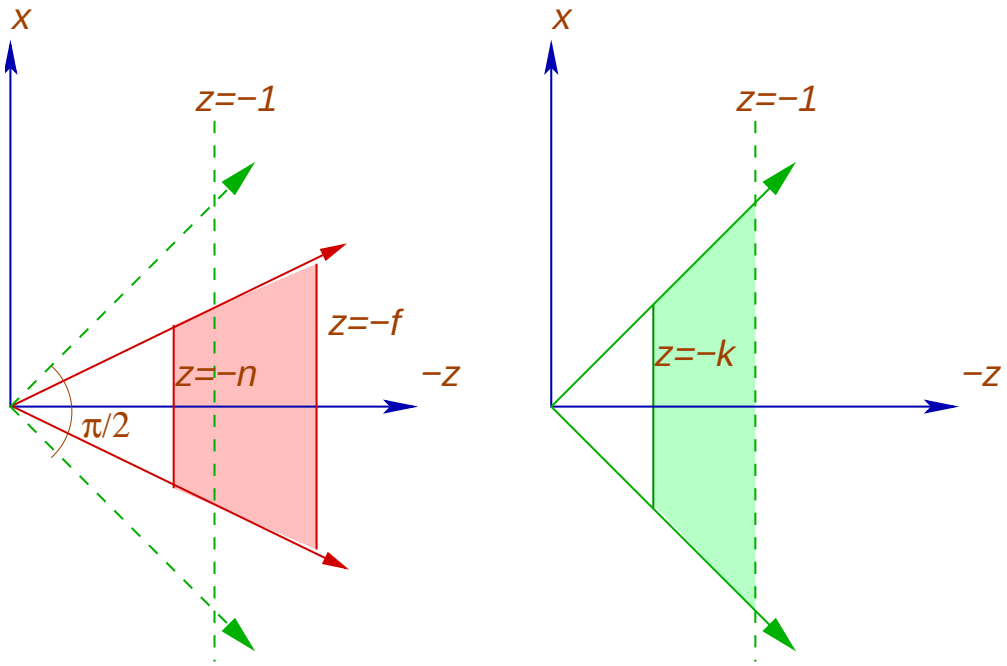


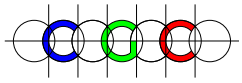
Canonical View

Perspective transform

☆ Perform a transformation so that

- The far clipping plane is $z = \Leftrightarrow 1$
- Corners of the clipping rectangle on the far plane are $(\pm 1, \pm 1, \Leftrightarrow 1)$

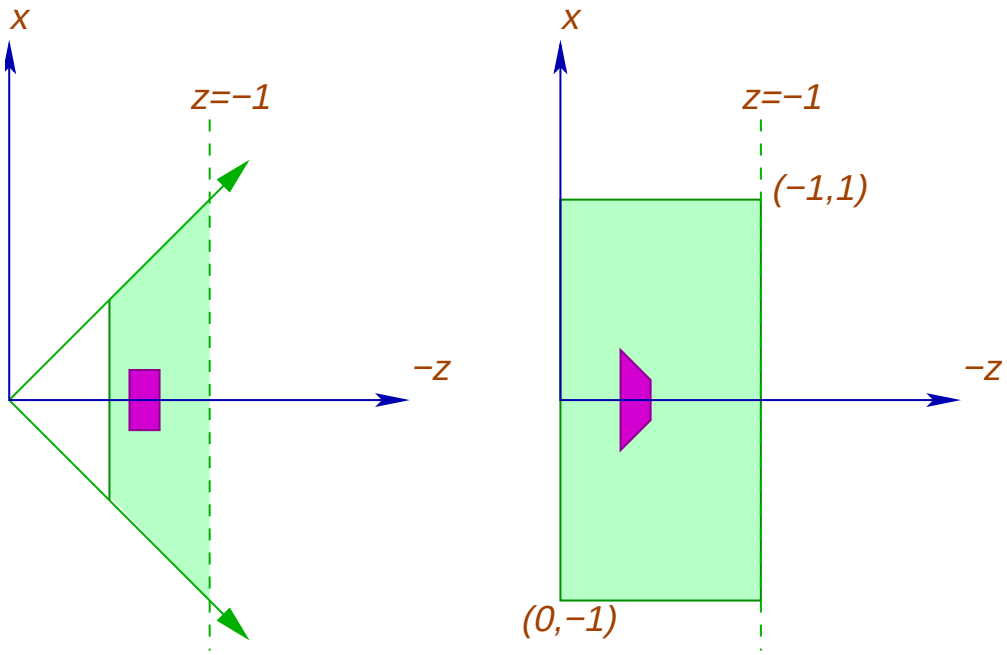


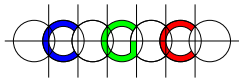


Canonical View

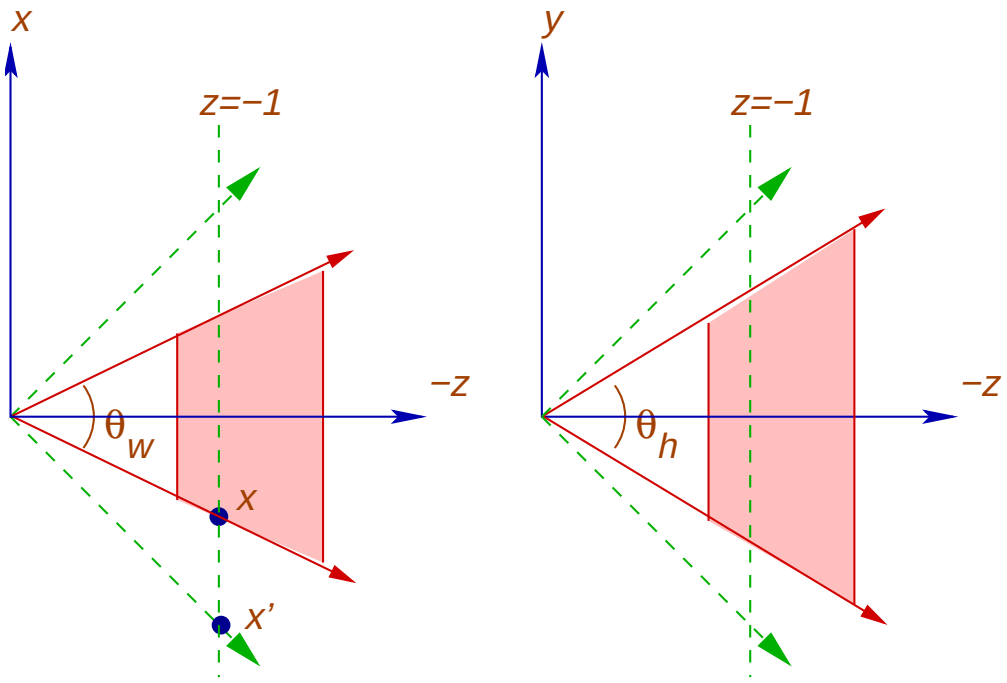
☆ Perform a perspective transformation so that

- Viewing volume is a parallel box
- near clipping plane is $z = 0$,
  far plane is at $z = \Leftrightarrow 1$
- Distorts the objects





Canonical View: Step I



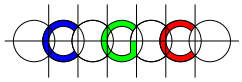
$x = (\tan(\theta_w/2), 0, \Leftrightarrow 1),\ x' = (1, 0, \Leftrightarrow 1)$

$y = (0, \tan(\theta_h/2), \Leftrightarrow 1),\ x' = (0, 1, \Leftrightarrow 1)$

★ Need to scale x by $1/\tan(\theta_w/2) = \cot(\theta_w/2)$

★ Need to scale y by $1/\tan(\theta_h/2) = \cot(\theta_h/2)$

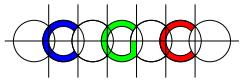
$$S_{xy} = \begin{bmatrix} \cot(\theta_w/2) & 0 & 0 & 0 \\ 0 & \cot(\theta_h/2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



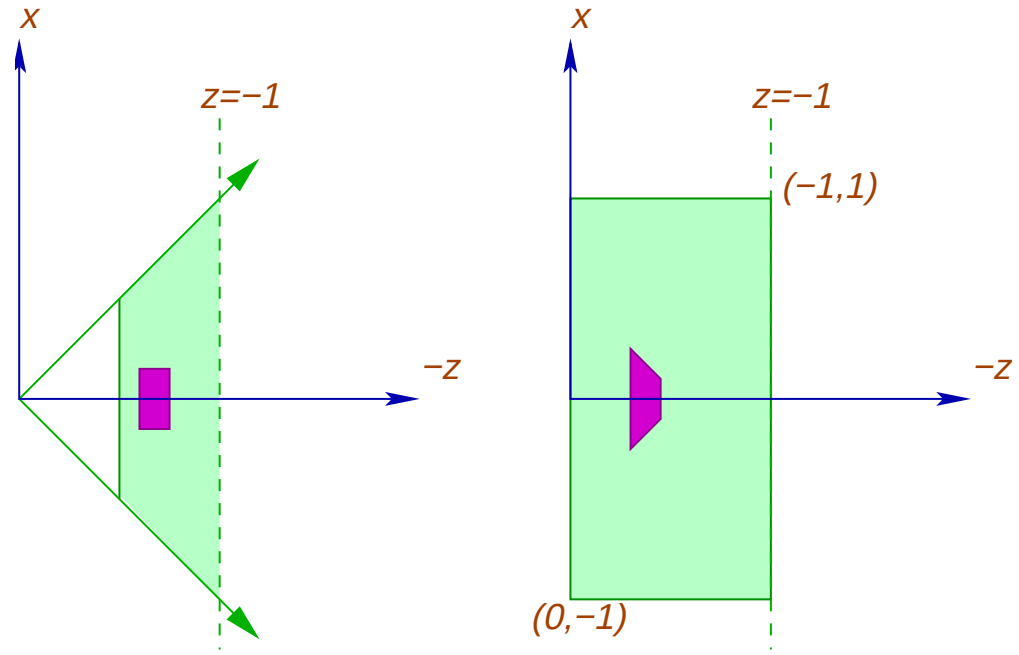
Canonical View: Step I

- ☆ Far clipping plane may lie at $z \neq 1$
- ☆ Scale so that the far clipping plane is $z = \Leftrightarrow 1$
- ☆ S_{xy} does not scale the z-coordinates, so far plane is at $z = \Leftrightarrow f$.
- ☆ Perform a uniform scaling by $1/f$

$$S_2 = \begin{bmatrix} 1/f & 0 & 0 & 0 \\ 0 & 1/f & 0 & 0 \\ 0 & 0 & 1/f & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



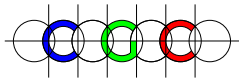
Canonical View: Step II



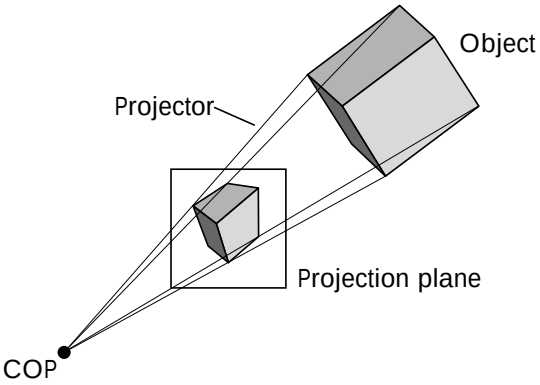
- ☆ Suppose that the near clipping plane ($z = \Leftrightarrow n$) after applying $S_2 \cdot S_{xy}$ is $z = \Leftrightarrow k$.
- ☆ Show that $k = n/f$!

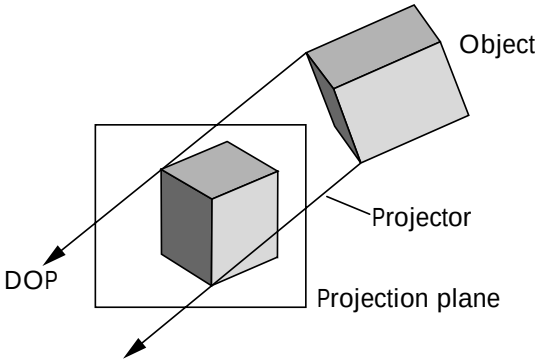
$$D = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/(1 \Leftrightarrow k) & k/(1 \Leftrightarrow k) \\ 0 & 0 & \Leftrightarrow 1 & 0 \end{bmatrix}$$

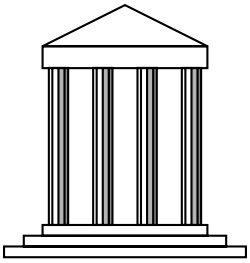
$$\textsc{Proj} = D \cdot S_2 \cdot S_{xy}$$



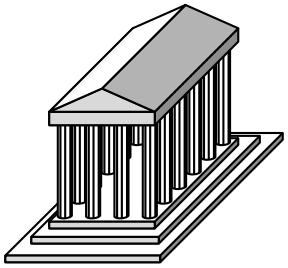
Geometric Projections



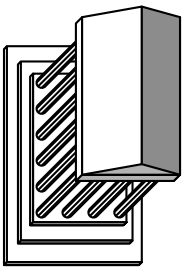




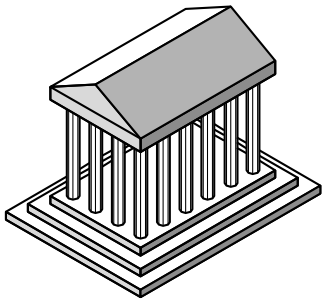
Front elevation



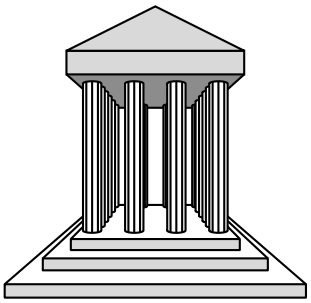
Elevation oblique



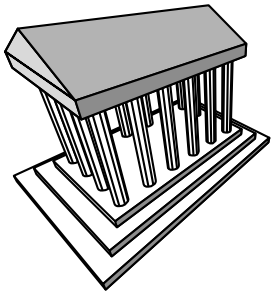
Plan oblique



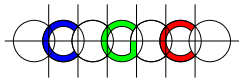
Isometric



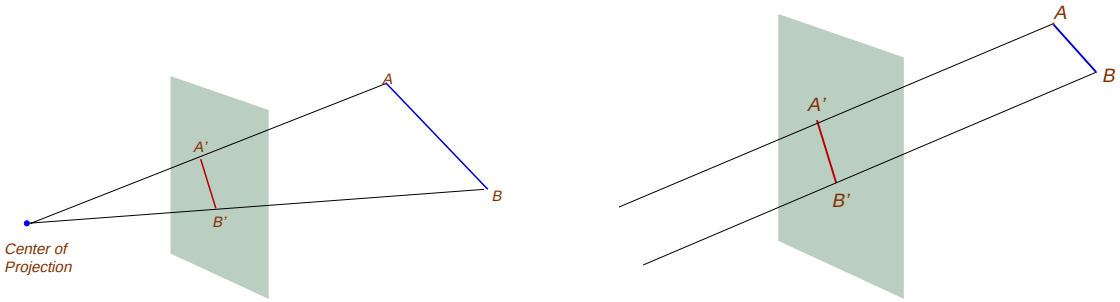
One-point perspective



Three-point perspective



Geometric Projections



☆ *Projection plane:* Plane Π

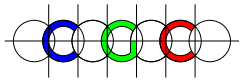
☆ *Projectors*

Rays emanating from the center of projection and passing thru points of the object.

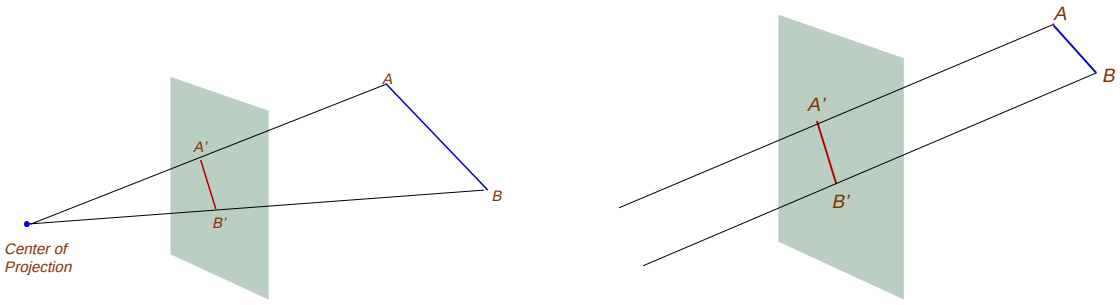
☆ *Projection*

Intersection of projectors with plane Π.

Non-geometric projections used in cartography.



Different Projections

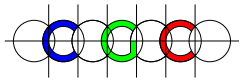


☆ *Perspective:*

- Center of projection is at a finite distance from Π.
- *Perspective foreshortening*.

☆ *Parallel:*

- Center of Projection is at ∞.
- Defined by the direction $(x, y, z, 0)$.
- Directions $\Leftrightarrow$ Points at infinity.

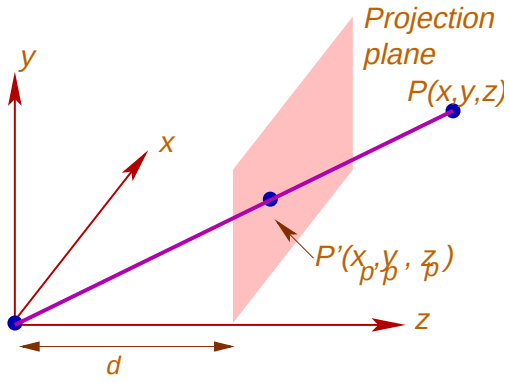


Mathematics of Projections

Center of projection at origin

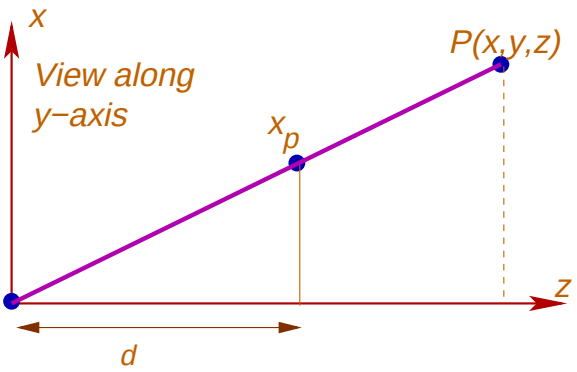
Using similar triangles

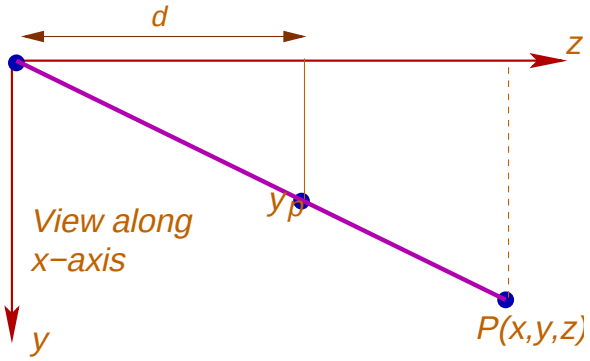
$$\frac{x_p}{d} = \frac{x}{z}; \frac{y_p}{d} = \frac{y}{z}$$

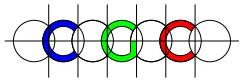


$$x_p = \frac{x}{z/d}; y_p = \frac{y}{z/d}$$

$$z_p = d$$





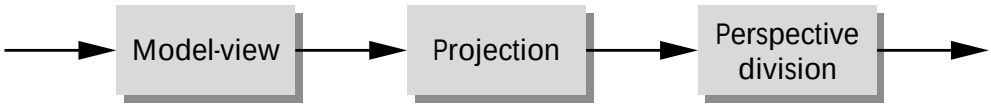


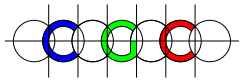
Mathematics of Projections

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

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad \mathbf{q} = M_{per}\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

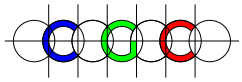
$$\mathbf{q}' = \begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$





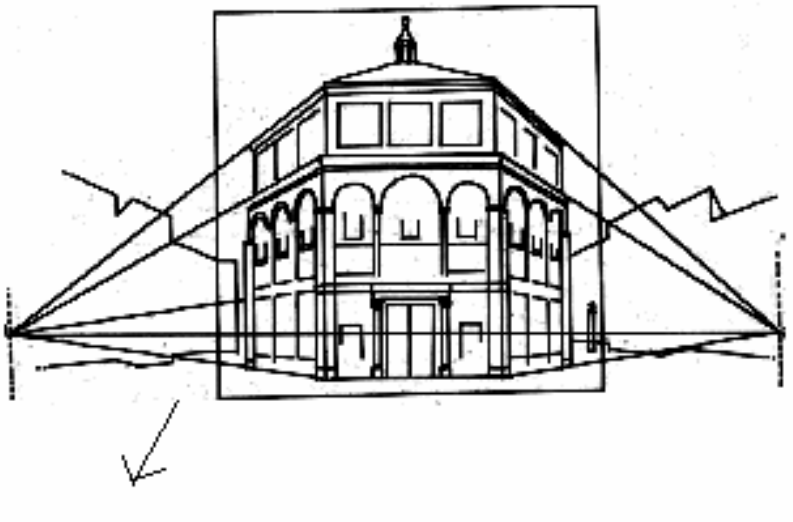
History of Projections

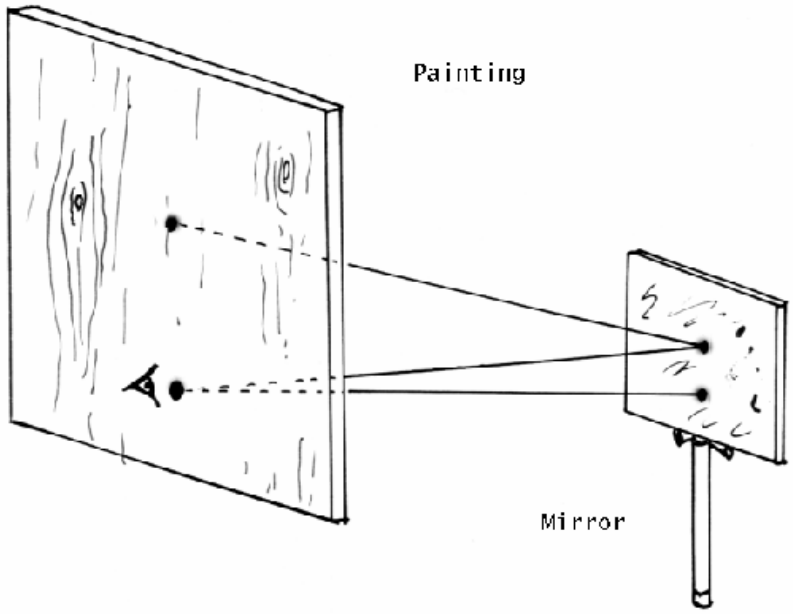
- ☆ Plan from Mesopotamia, ≈ 2000BC.
- ☆ Early Greeks: *Agatharchus* (≈ 500 BC) *Apollonius* studied projections of conics.
- ☆ Romans: *Vitruvius* wrote *De Architectura* Published specifications of plan and elevation drawings, and perspective.
- ☆ Early Renaissance period: Emphasis on point of view, interpretation of world.
  - Giotto
  - Duccio
  - Mossacio
  - Dontallo
  - Dürer
  - Vinci
  - Raphael

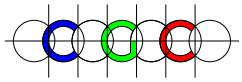


History of Perspective

Filippo Brunelleschi invented systematic method of determining perspective projections in early 1400s.



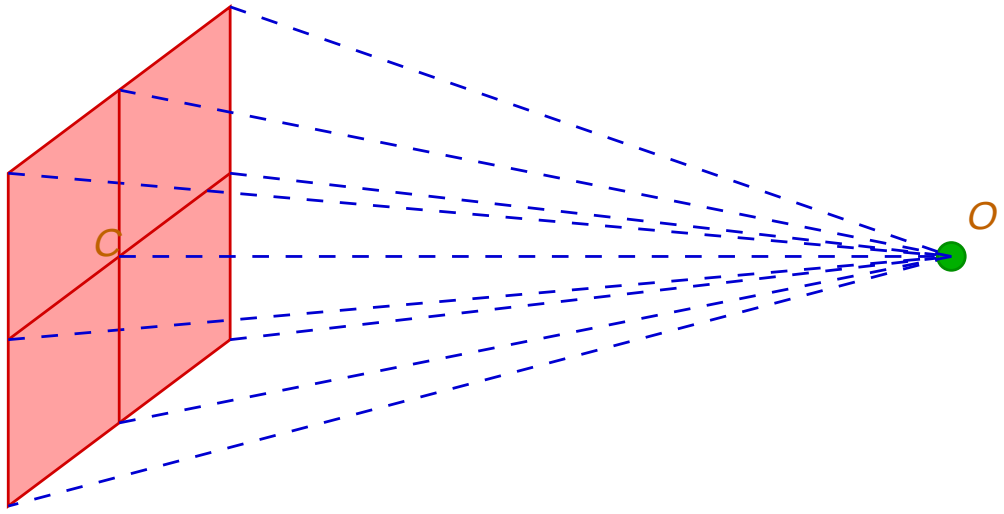


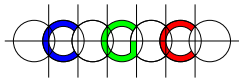


History of Perspective

★ *Leone Battista Alberti* wrote the first treatise on perspective, *Della Pittura*, in 1435.

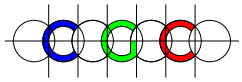
> *A painting is the intersection of a visual pyramid at a given distance, with a fixed center and a definite position of light, represented by art with lines and colors on a given surface.*
>
> – *On Painting*



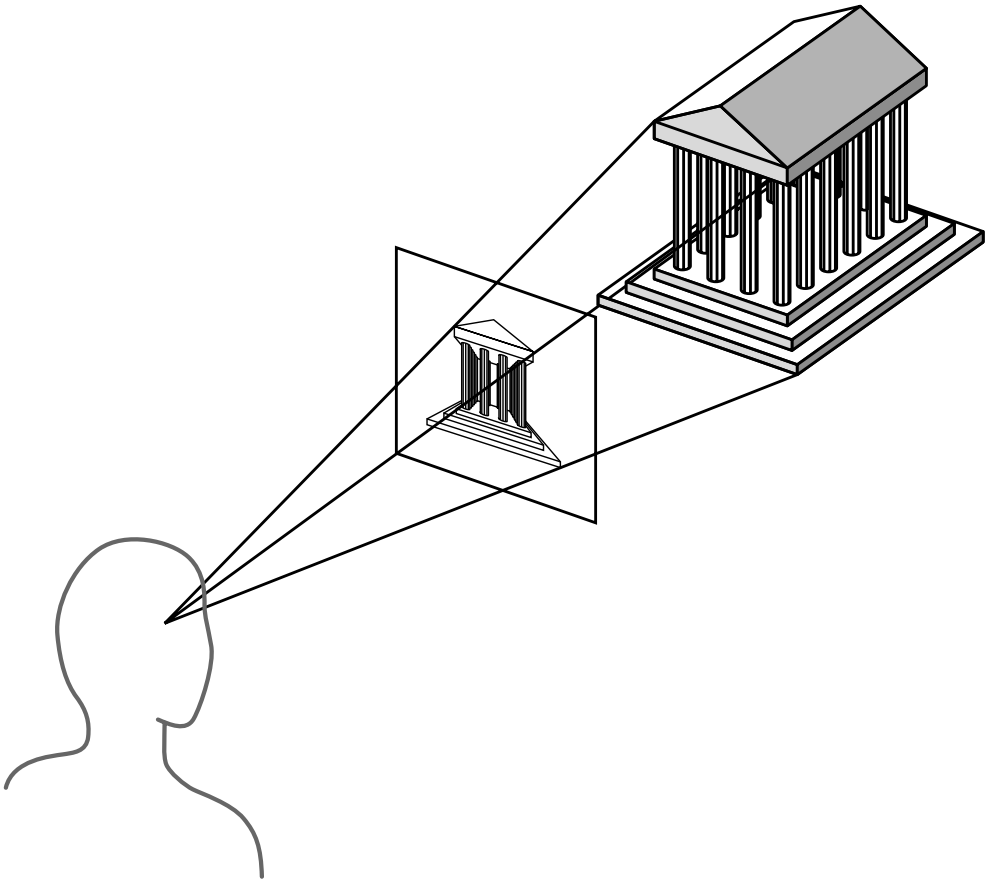


History of Perspective

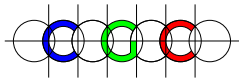
- ☆ Piero della Francesca: *De Prospettiva Pengendi*
- ☆ Domencio Veneziano: *St. Lucy Altarpiece.*
- ☆ Leonardo da Vinci: *The Last Supper, Annunciation*
- ☆ Gerard Desargues: French architect, engineer,
- ☆ Gaspard Monge: multiple orthographic projections



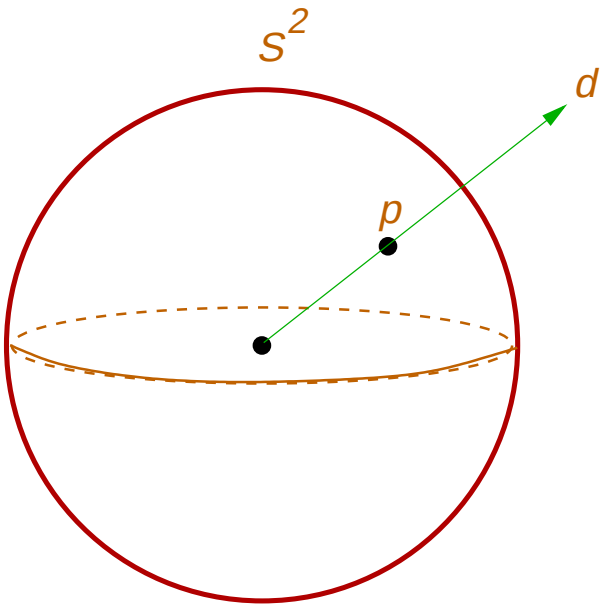
Perspective Projection



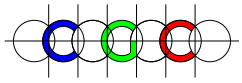
- ☆ Foreshortening gives a realistic view for 3-dimensional objects.
- ☆ Foreshortening is not uniform.
- ☆ Parallel edges don't remain parallel; scales and other geometric properties are not preserved.
- ☆ Used for advertising, fine art, architecture.



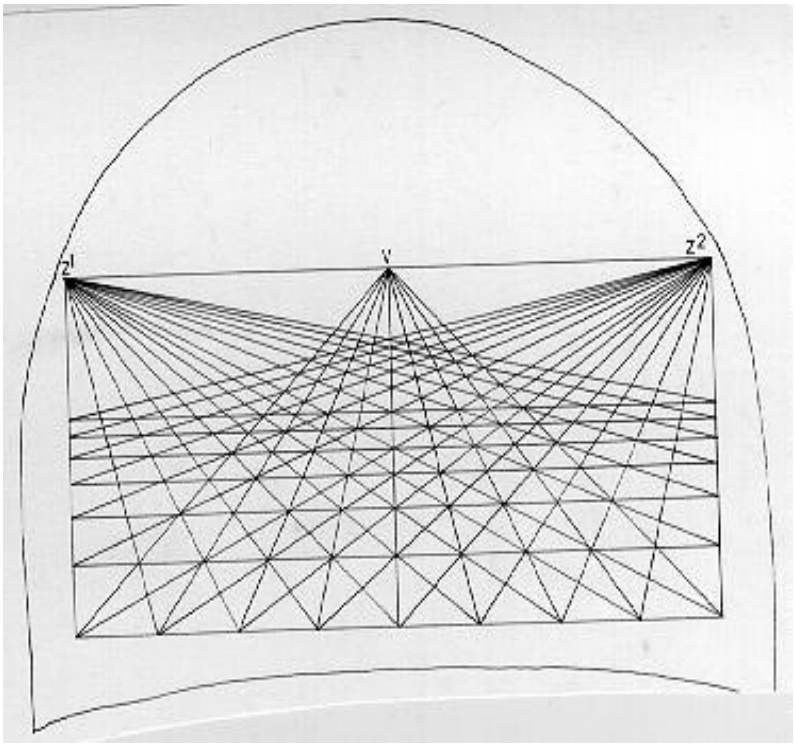
Perspective Projection



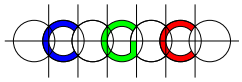
- ☆ Parallel lines meet at infinity.
- ☆ Fix a direction d.
- ☆ $p(d)$: Point at infinity in direction d.
- ☆ $L(d)$: Set of all lines parallel to direction d.
- ☆ Π: A plane not parallel to d.



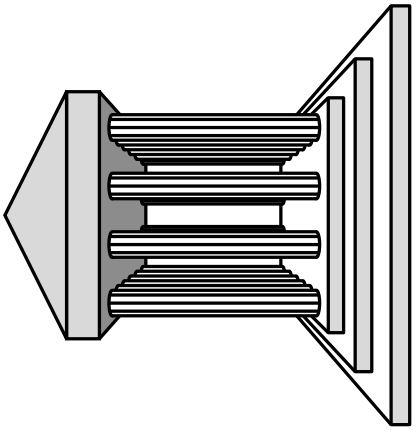
Vanishing Points



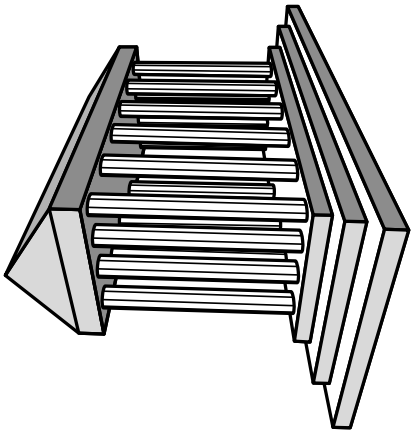
- ☆ Projection of all lines in $L(d)$ on Π meet at a common point $p'(d)$, which is the projection of $p(d)$ on Π.
- ☆ $p'(d)$ is called the *vanishing point* of $L(d)$. If d is one of the axes, $p'(d)$ is called axis vanishing point.
- ☆ There are at most 3 axis vanishing points.



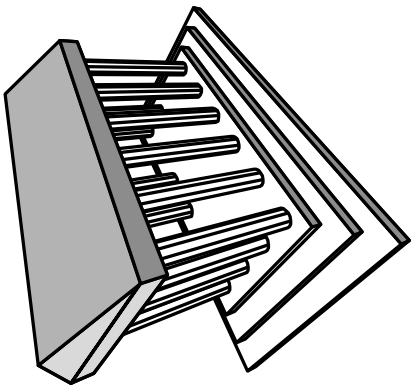
Different Perspective Views



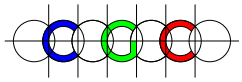
(c)



(b)

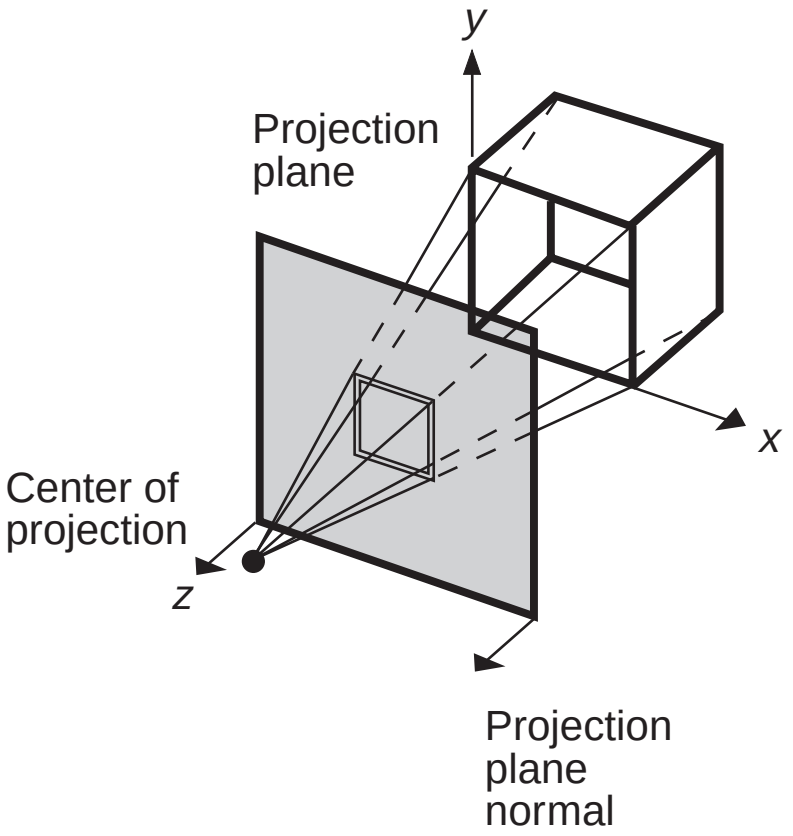


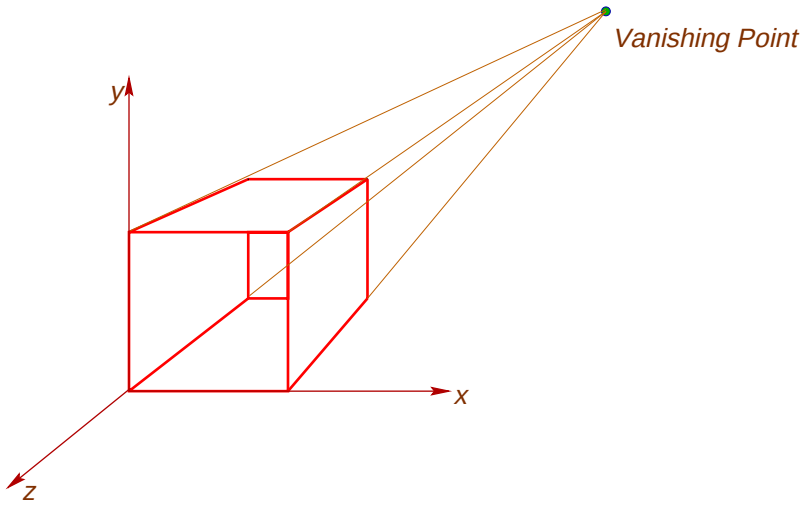
(a)

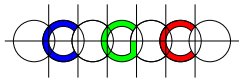


One Vanishing Point

Π parallel to the xy-plane $\Leftrightarrow$ One axis vanishing point.

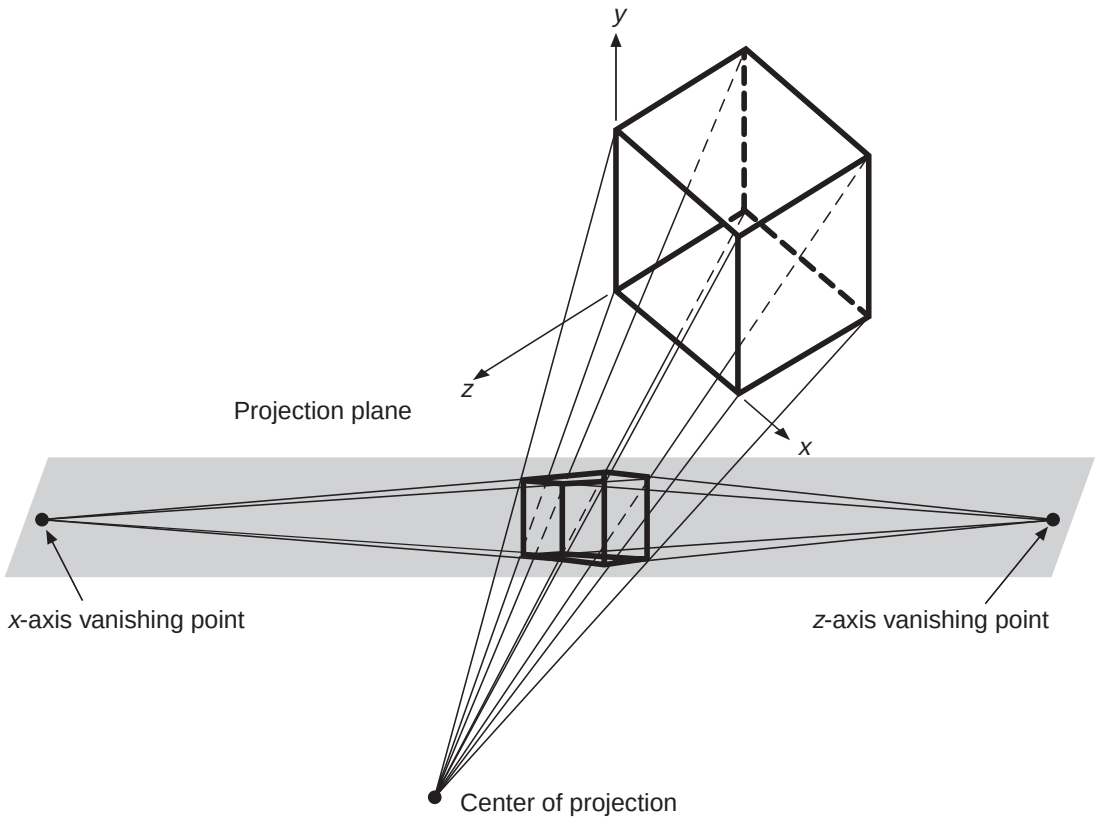




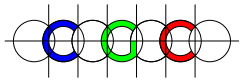


Two Vanishing Points

Viewing plane is parallel to only one axis.

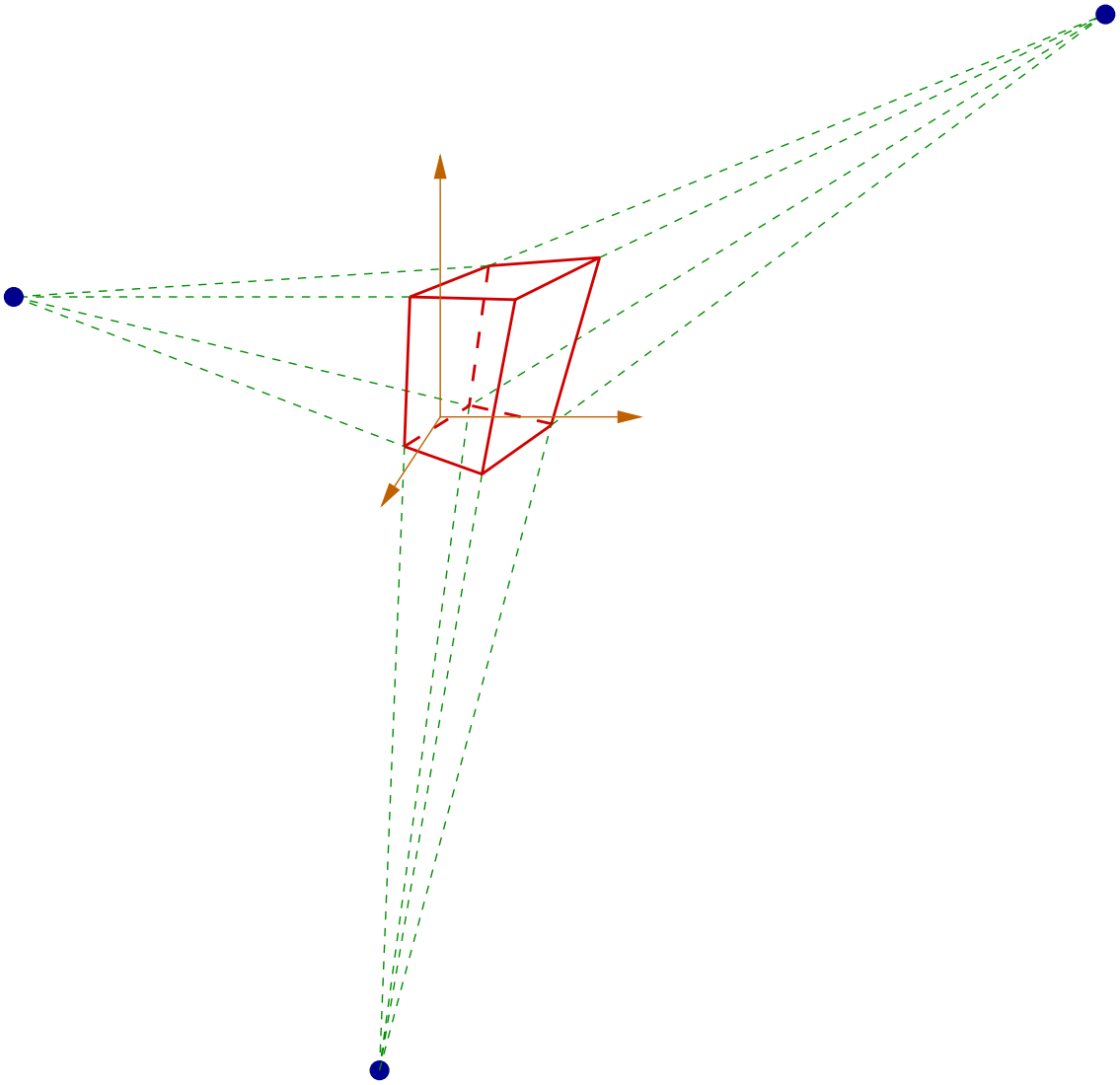


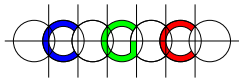
Projections with two vanishing points are used in architecture drawings.



Three Vanishing Points

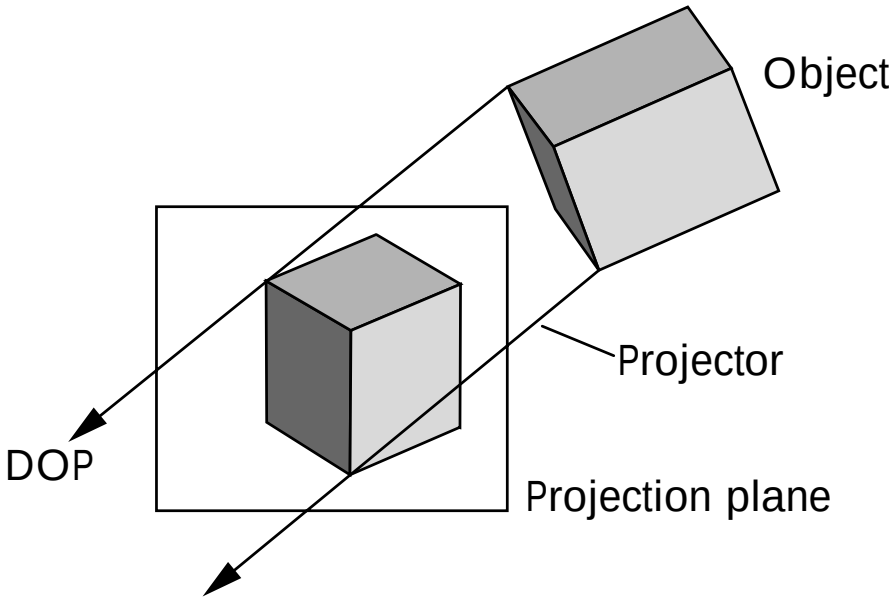
Viewing plane is not parallel to any axis.

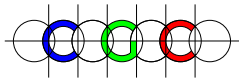




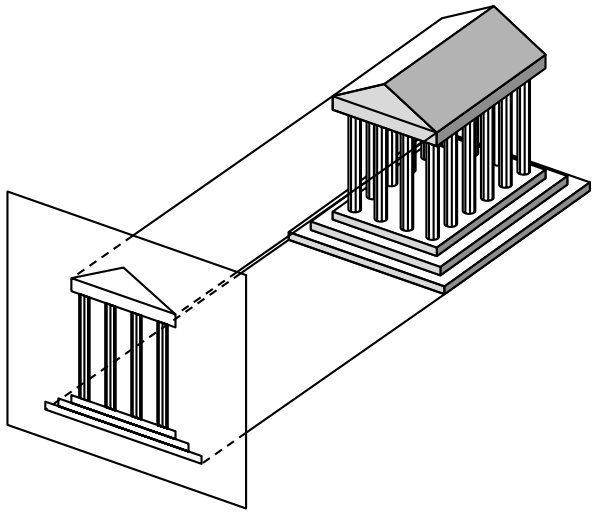
Parallel Projections

- ☆ *Orthographic:*
  Projectors are perpendicular to the projection plane.
- ☆ *Oblique:*
  Projectors not perpendicular to the projection plane.

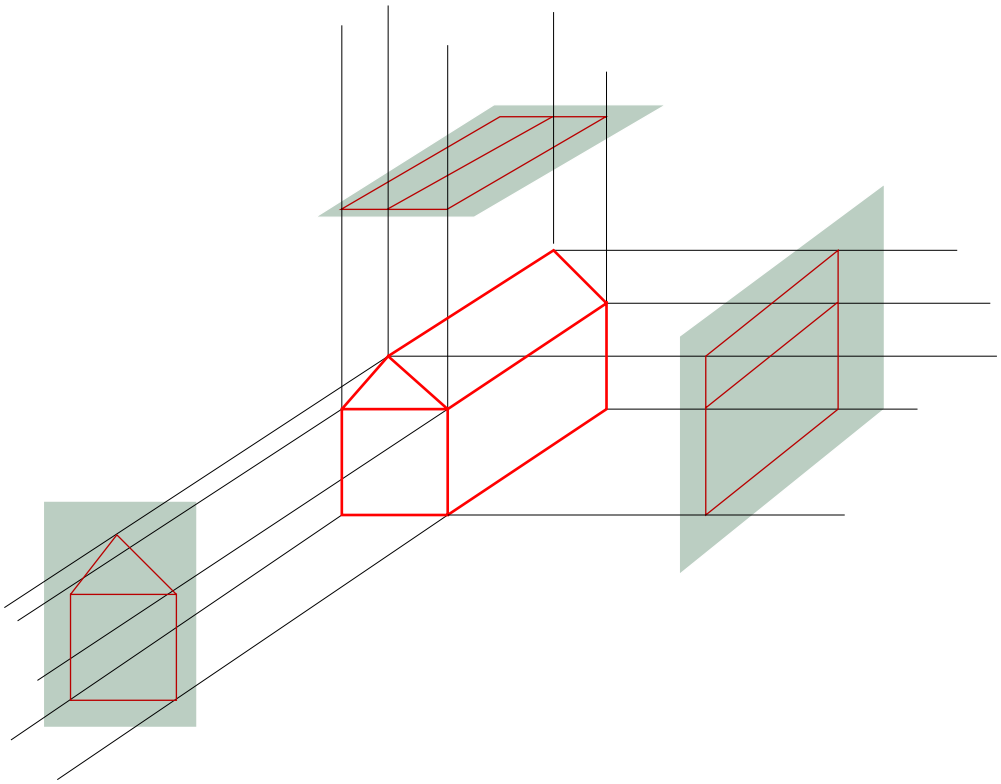


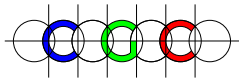


Orthographic Projections

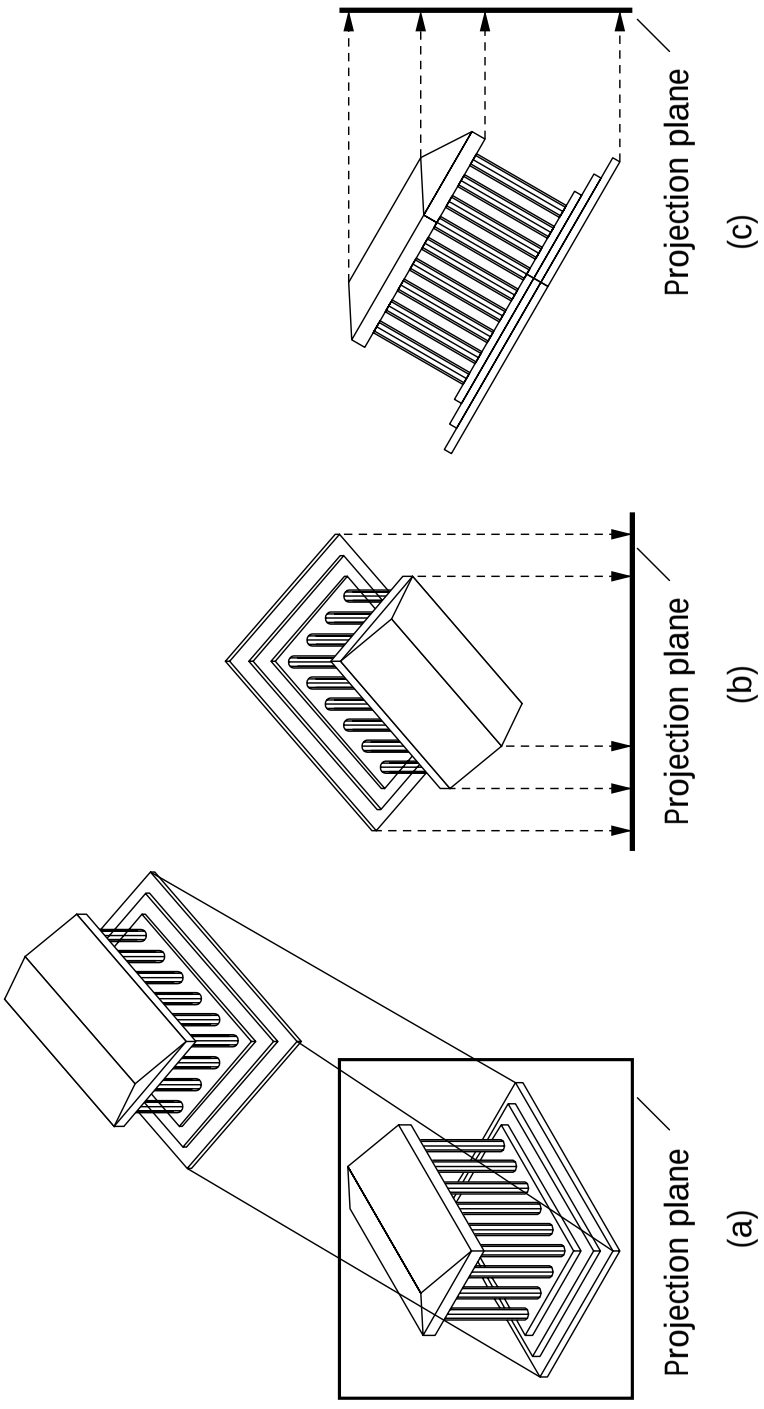


- ☆ *Front, top, side views:* Projectors parallel to one of the principal axes
- ☆ *Front, top, side views:*

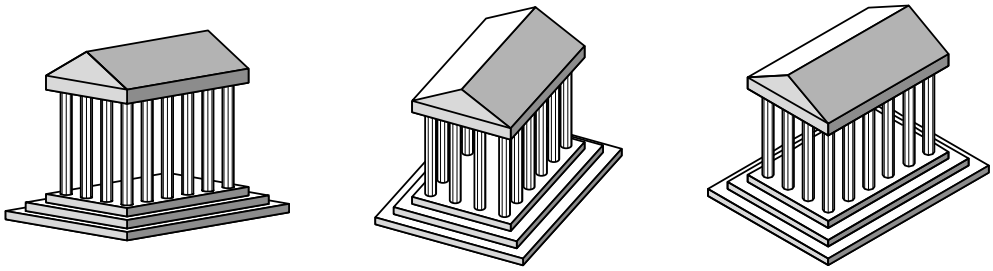




Axonometric Projections

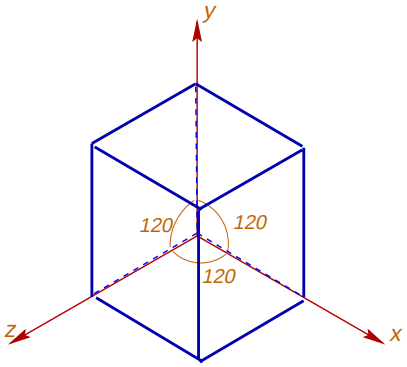


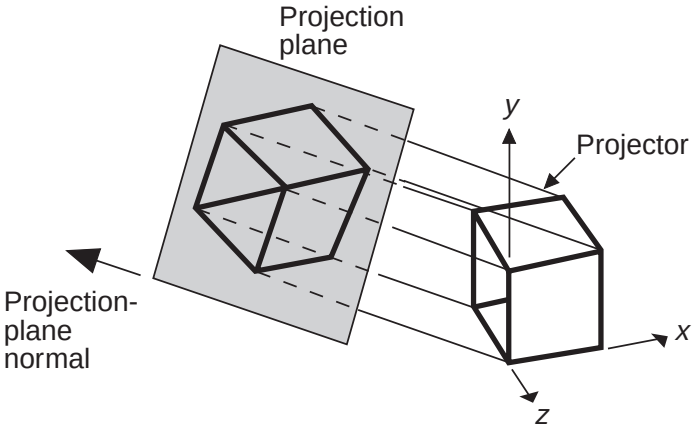
Orthographic Projections



★ *Isometric:*

Projectors make equal angle with each axis

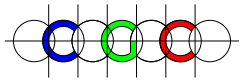




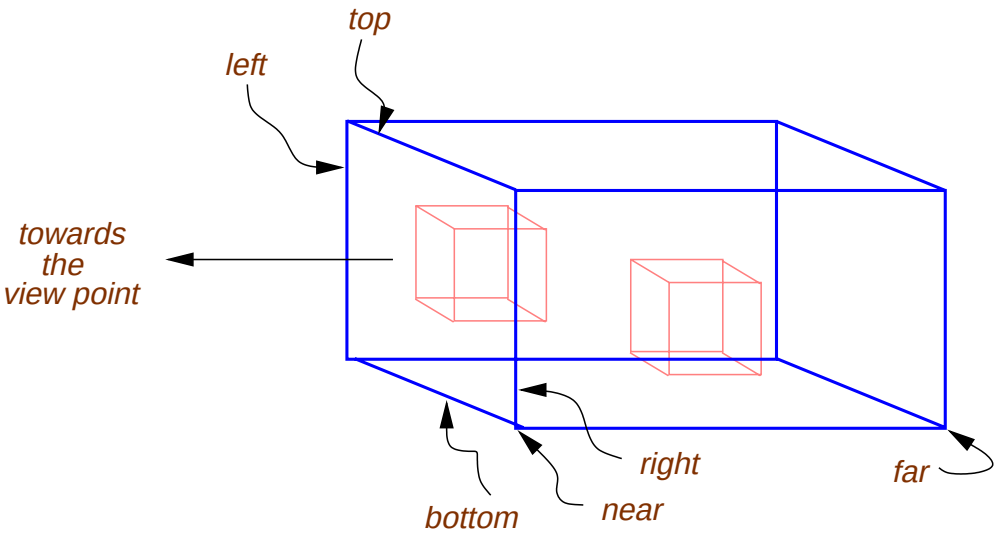
Eight possible directions $(\pm 1, \pm 1, \pm 1)$.

★ *Dimetric:* Equal angle with two axes.

★ *Trimetric:* Distinct angle with all three axes.



Otrhographic Projection



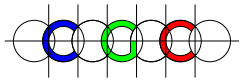
☆ `glOrtho` (l, r, b, t, n, f)

Arguments are the same as in `glFrustum`.

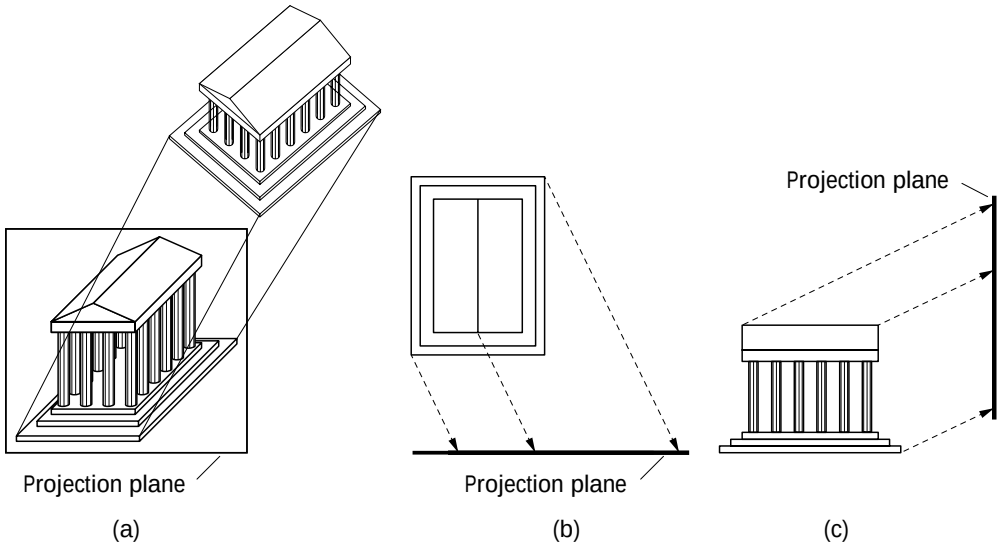
☆ `glOrtho2d` (l,r,b,t)

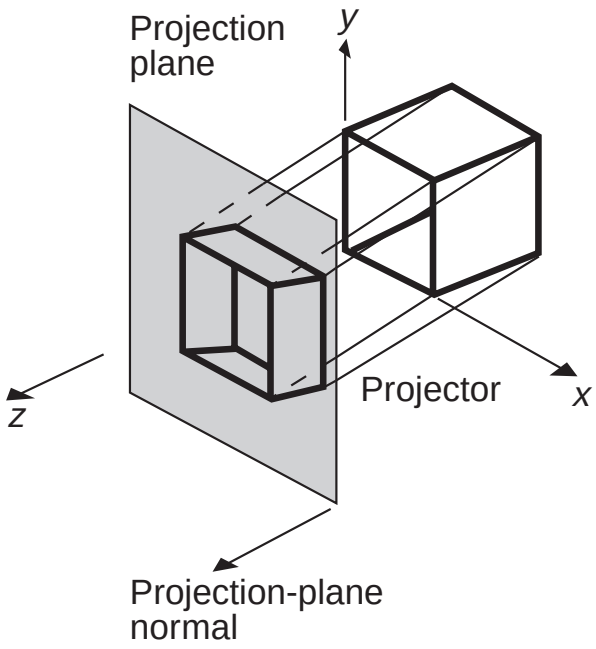
Same as `glOrtho` (l, r, b, t, 0, 0).

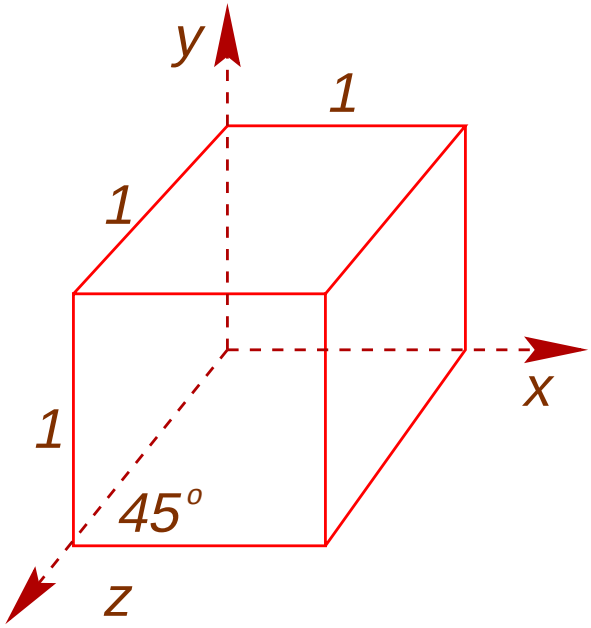
Useful for 2D viewing.



Oblique Projections

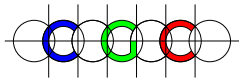






Cavalier projection:

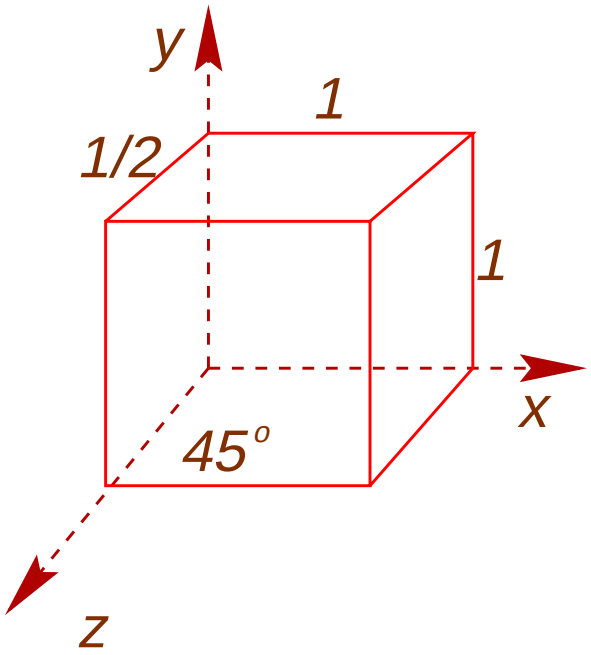
- ☆ Angle between projectors and projection plane is $\pi/4$
- ☆ Length of a segment $\perp$ the projection plane = Length of the projection of the segment.

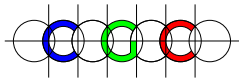


Oblique Projections

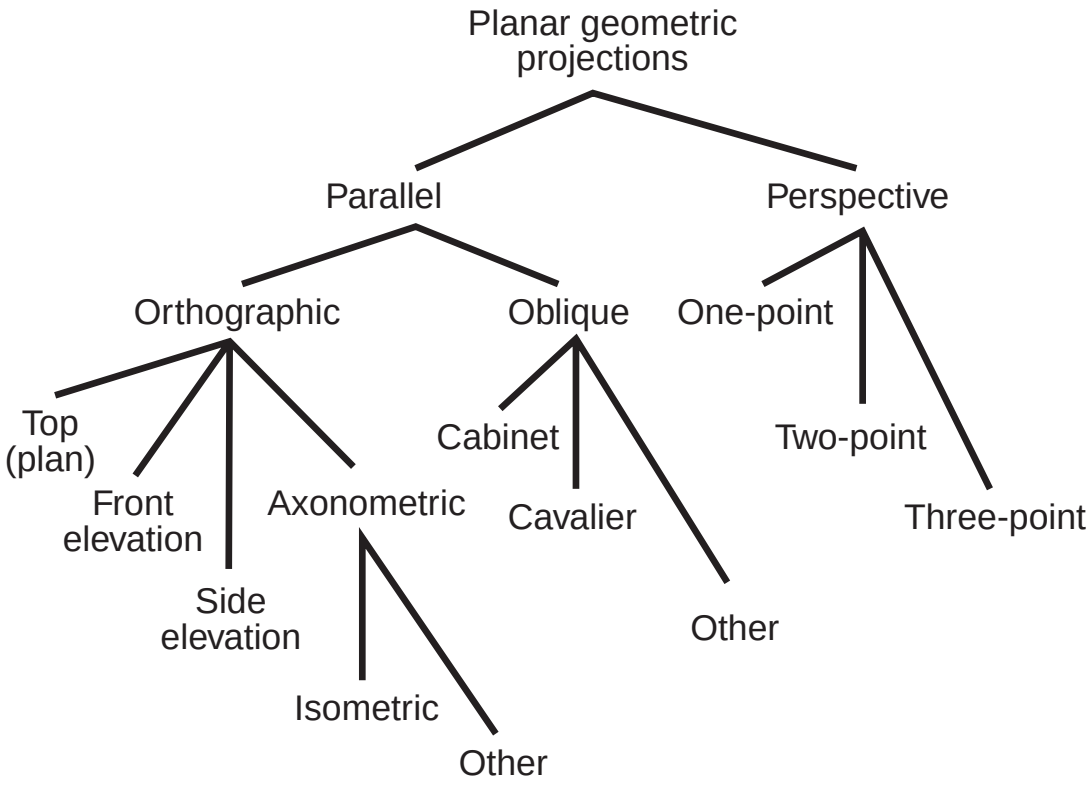
Cabinet projection:

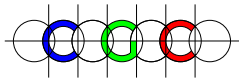
- ☆ Angle between projectors and projection plane is $\tan^{-1} 2 \approx 63.4$.
- ☆ Length of a line normal to the projection plane = Twice the length of the projection of the line.





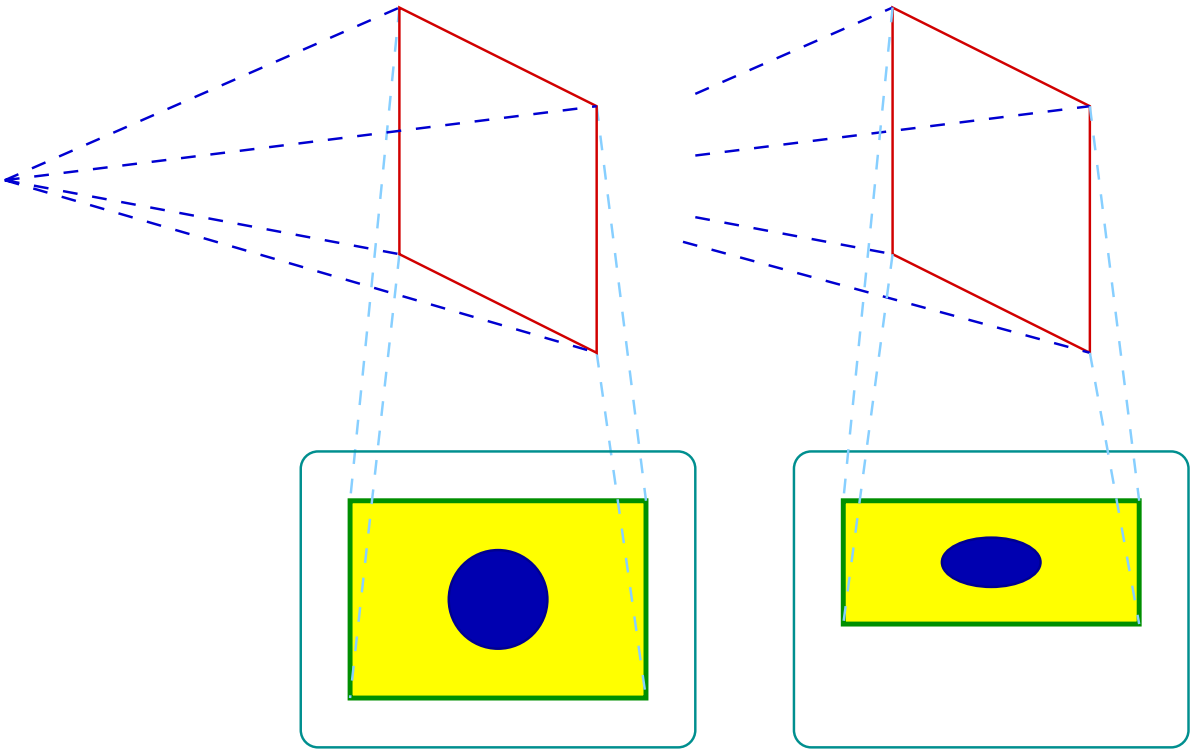
Summary of Projections





ViewPort Transform in OpenGL

Projection plane to screen transformation.



`glViewport`(x, y, w, h)

$x, y,$: Lower left corner of the viewport.

w: Width of the viewport.

h: Height of the viewport.