Viewing

CSE606: Computer Graphics
Jaya Sreevalsan Nair, IIIT Bangalore
February 10-12, 2025

Viewing: Classical vs. Computer

Different Views

Different views can be achieved of a scene, and represented using appropriate 3-dimensional projections

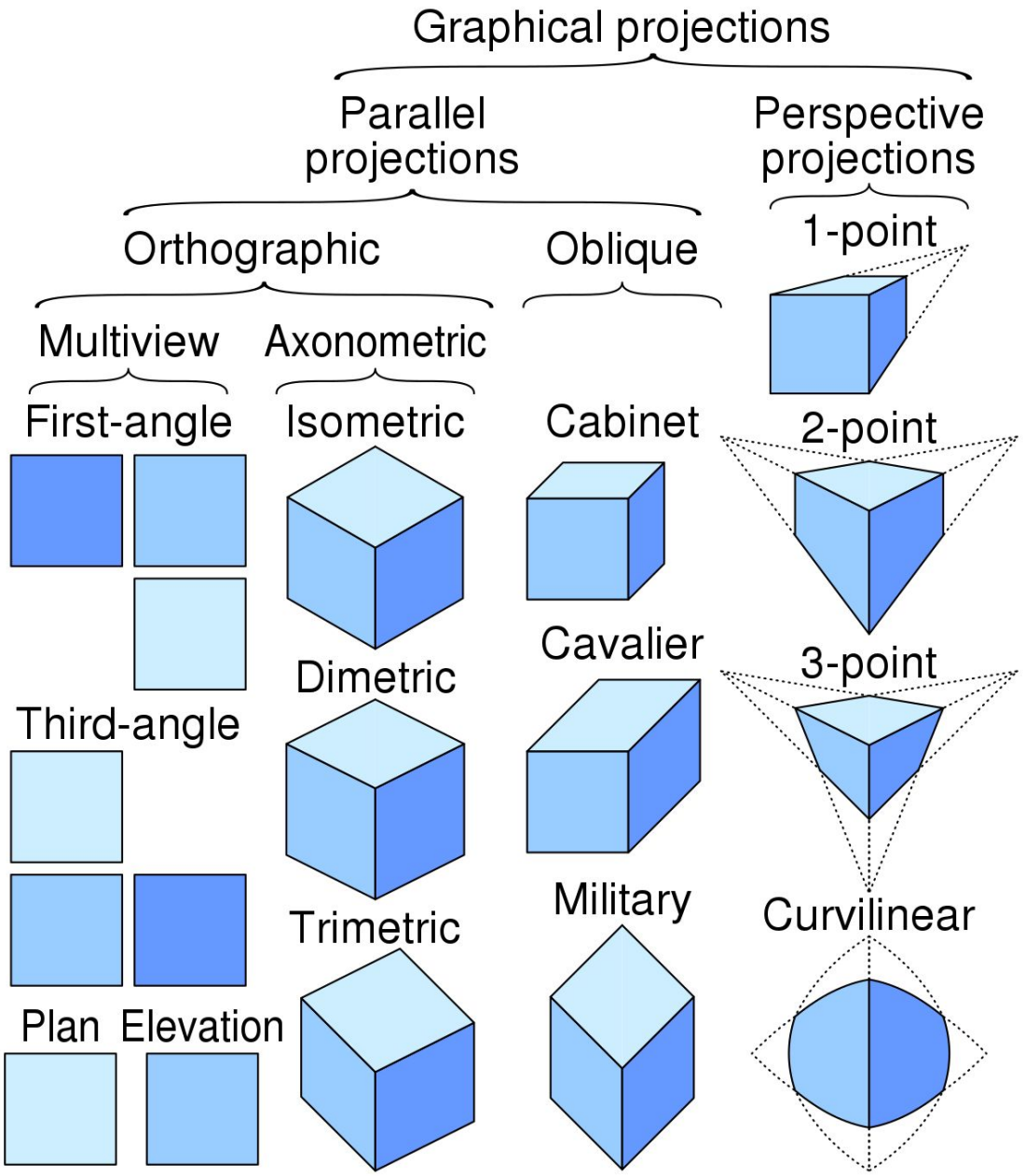


Image courtesy: https://en.wikipedia.org/wiki/3D_projection

Introduction to Viewing

- Why do we need so many different views?
- How does the application program specify a view within OpenGL framework ?

Two parts within the viewing process in OpenGL:

1. Use canonical viewing procedures – by using model-view matrix transformation from object-frame to eye-frame.
2. Specify the type of projection & the extent of the world within the image (clipping volume).

Classical vs. Computer

- Planar Geometric Projections: Class of projections generated by graphics systems.
  - Planar projection surface; and linear projectors.
- Available Projections:
  - Computer: Parallel & Perspective (limited choices)
  - Classical: many more, ranging from multi-view orthographic to one-, two-, and three-point perspectives.

Comparison: Classical vs. Computer

What do comparisons teach us?

- Practitioners of animation, architecture, etc. need to produce classical views – isometrics, elevations, various perspectives, etc.
- Animation in movies, architectural rendering, etc. started out with hand drawing before computer graphics.
- Relationships between classical and computer viewing reflect advantages of, and difficulties with the approach used by most API for graphics systems.

Recall: Synthetic-Camera Model

- Center of Projection (COP) : Intersection of all projectors; typically center of lens in camera/eye ⇒ origin of camera frame.
  - Note: Planar projection surface; and linear projectors.
- Direction of Projection (DOP) : $\lim_{\text{COP} \to \infty} \text{COP} = \text{DOP}$, i.e., the projectors become parallel.
  - With finite COP, we get perspective viewing; and with DOP, parallel views.
  - Parallel views have fixed projection plane and fixed image size.

Parallel and Perspective Viewing

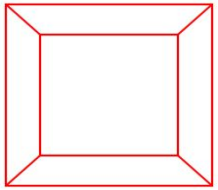
Most modern APIs support both views.

- Though *parallel* is a limiting case of *perspective*, i.e. **parallel=type(perspective)**, they are considered/treated differently.
- More efficient to use specific equations directly to form the corresponding projection matrix, thus using representation of linear transformations in homogeneous coordinate system.
- Using generalization of linear transformations in OpenGL, *perspective* becomes a special case of *parallel*, i.e. **perspective=type(parallel)**.
- Both parallel and perspective viewing preserve lines, but not necessarily angles.

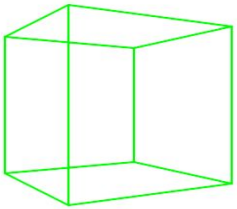
Overview

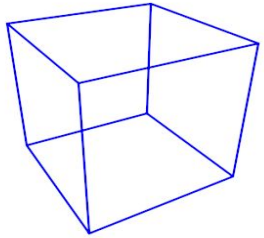
- There exists a specific relationship between object and viewer.
  - To achieve a view, the viewer is deliberately placed.
  - There is a notion of **principal face**, from the point of view of the viewer.
  - In OpenGL, the viewer and the scene are modular and independent, but will have equivalent transformations; hence, the notion of a modelview matrix.
- Examples of views: front elevation, elevation oblique, plan oblique, isometric, one-point perspective, three-point perspective.

Views



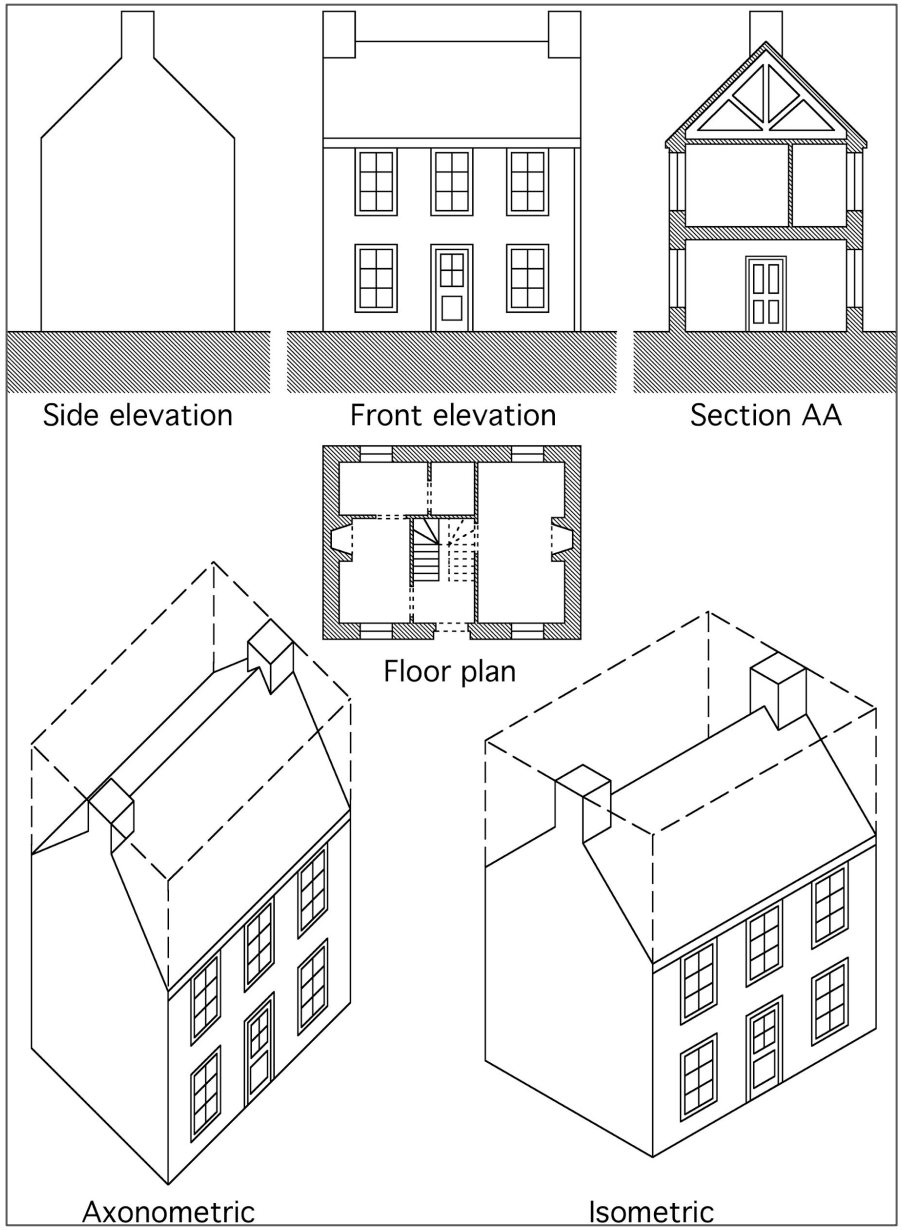
(Top-to-bottom)
1,2,3−point perspective





- Orthographic Projections
- Axonometric Projections
- Oblique Projections

(Image courtesy: Wikimedia Commons.)



Orthographic Projections

Projectors are perpendicular to projection plane.

- Preserves both distances and angles – no distortion.

Multiview orthographic projection: Use multiple orthographic/orthogonal projections, such as, front, top, and right.

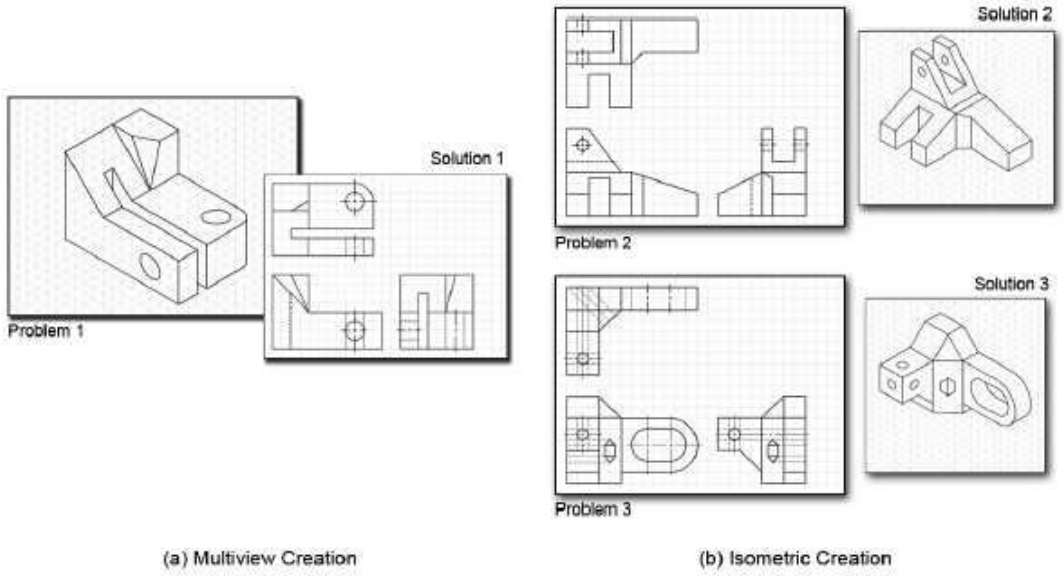


Image Courtesy: https://scholar.lib.vt.edu/ejournals/JITE/v44n3/mohler.html

Comparison of Graphical Projections

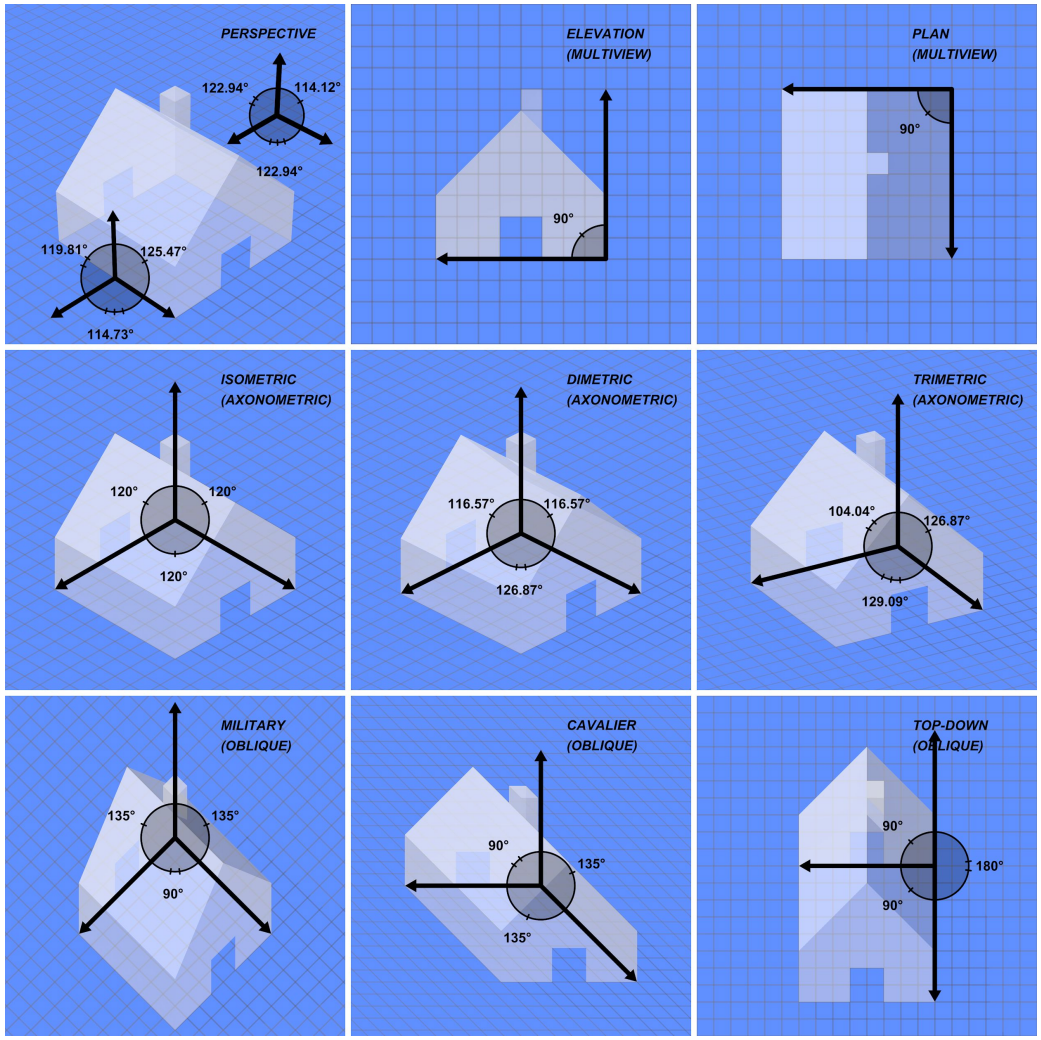


Image courtesy: https://en.wikipedia.org/wiki/3D_projection

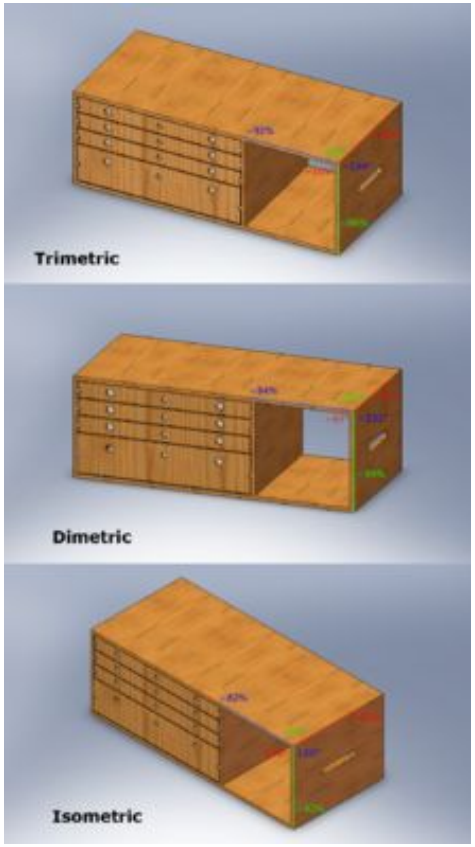
Axonometric Projections

Projectors are orthogonal to projection plane, however, projection plane can have any orientation with respect to object.

- **Isometric** : Projection plane is placed symmetrically with respect to three principal faces, that meet at a corner.
- **Dimetric** : Projection plane is placed symmetrically with respect to two principal faces.
- **Trimetric** : General case - involves no symmetry.

Parallel lines are preserved; not angles.

- Trade-off to see more principal faces in a single view.
- **Foreshortening** – Distances in view are shorter than in object.
    - There are 1,2, and 3, foreshortened distances in iso-, di-, and tri-metric views, respectively.

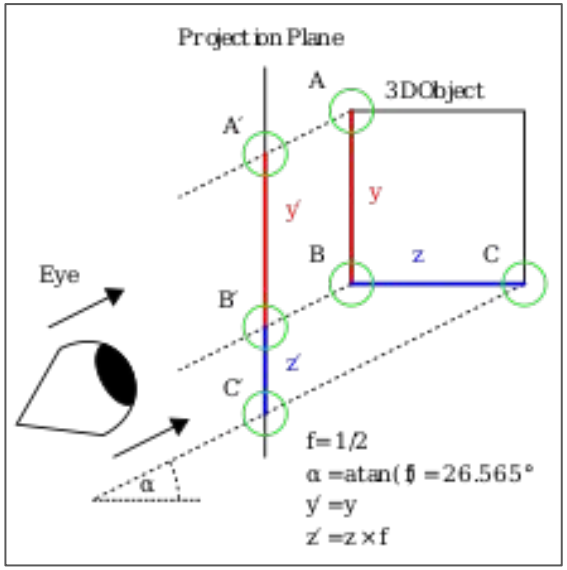


https://en.wikipedia.org/wiki/Axonometric_projection

Oblique Projections

The most general parallel views – allow projectors to make an arbitrary angle with the projection plane.

- Angles in planes parallel to projection planes are preserved.
- Most difficult to construct by hand.
- Allows viewing more than one principal face.



Oblique projection of a cube with foreshortening by half, seen from the side

https://en.wikipedia.org/wiki/Oblique_projection

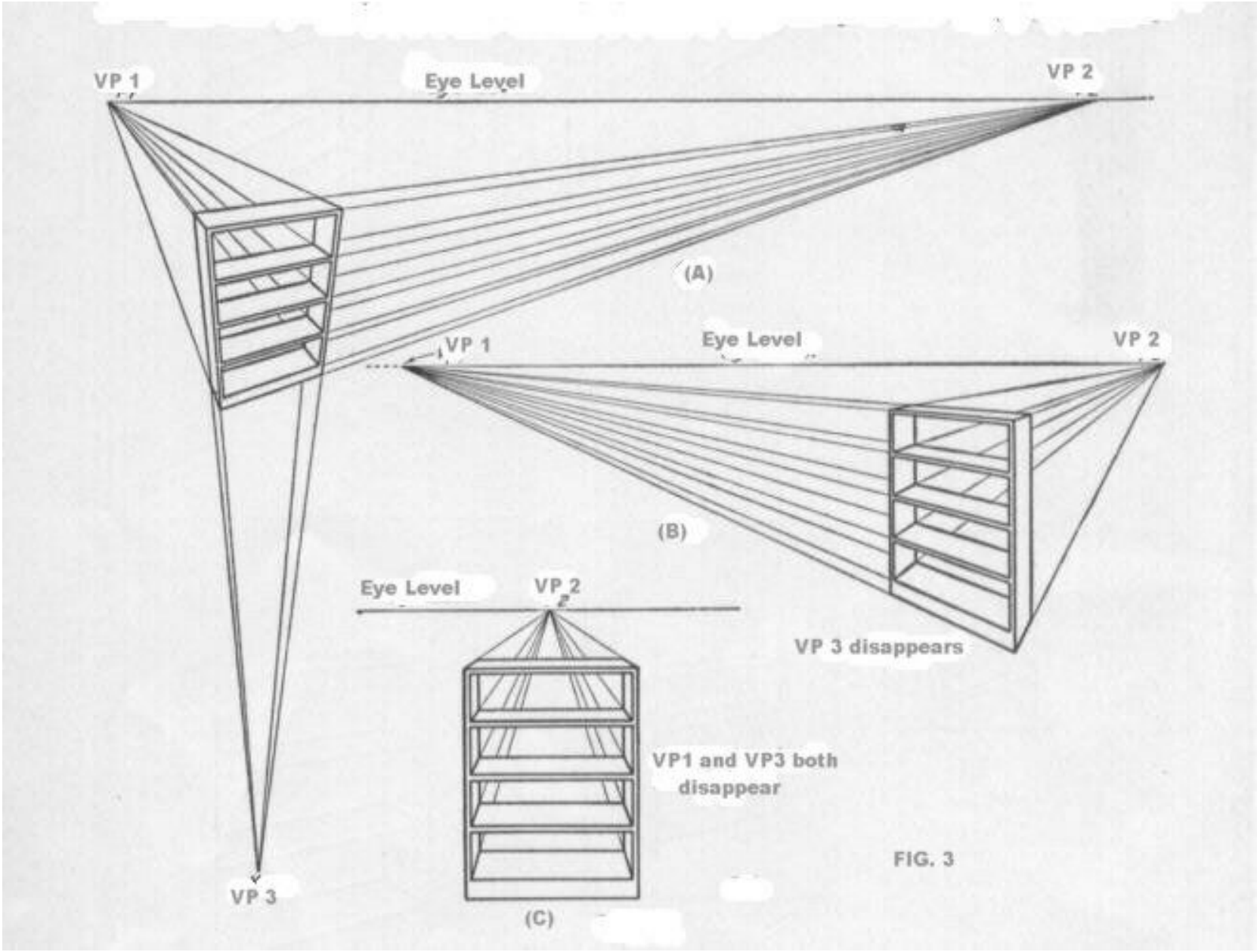
Perspective Viewing

Characterized by **diminution** of size.

- Size of object in view is inversely proportional to the distance of the object from the viewer.

In classical perspective view, viewer and object have a fixed relationship

- Included as a default in computer models.
- Includes one-, two- and three- point perspectives.



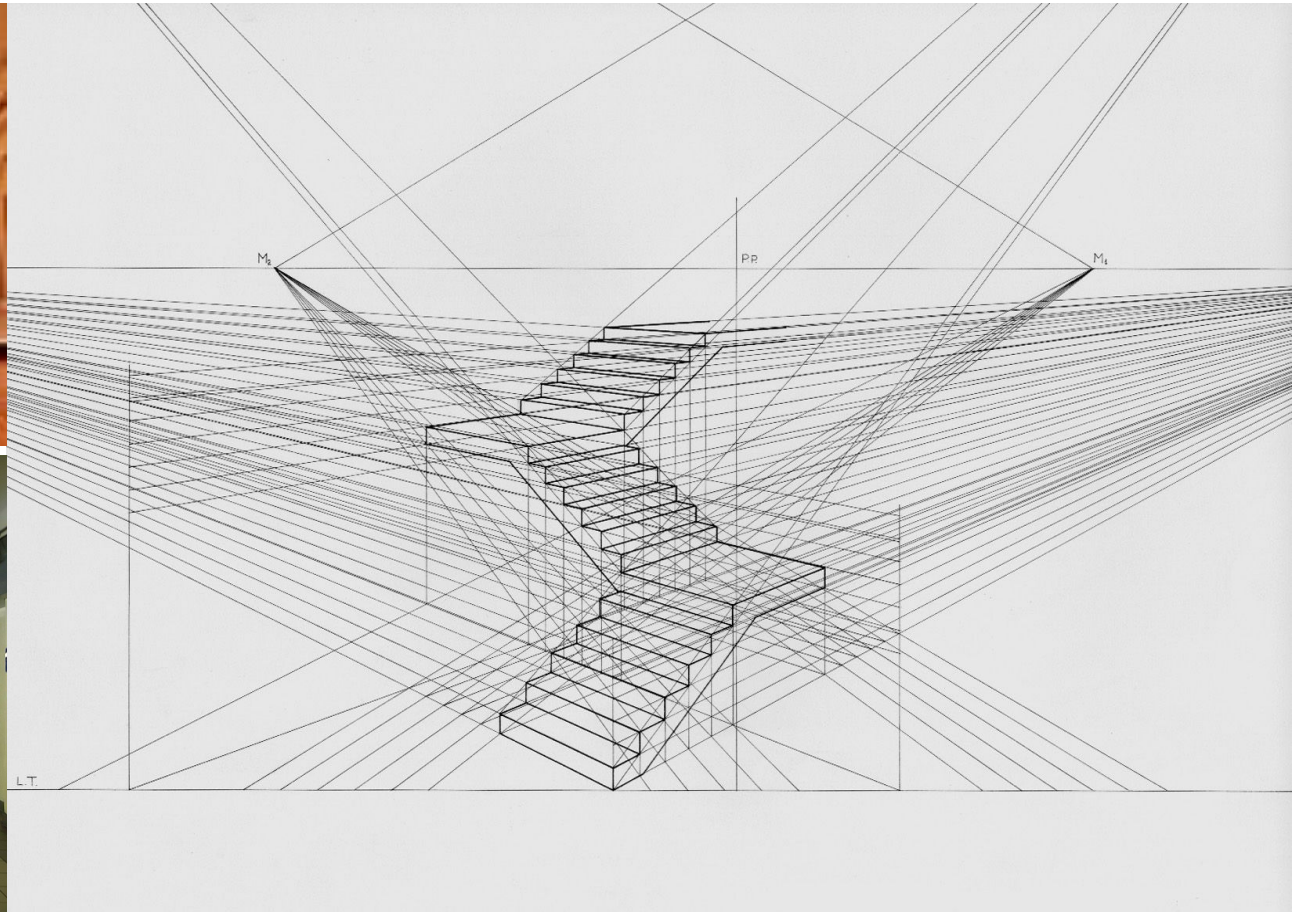
(A) Three-point, (B) two-point, and (C) one-point perspectives, and respective vanishing points.
(Image courtesy: http://www.wetcanvas.com/ArtSchool/Drawing/Perspective/Lesson1/index.html).

Vanishing Points in Perspective Projection

n-point perspective has n **vanishing points**, i.e., points at which a set of parallel lines in a particular direction seem to converge.

- Three-point perspective: Projection plane not parallel to any of the principal directions.
- Two-point perspective: One principal direction being parallel to projection plane.
- One-point perspective: Two principal directions parallel to projection plane.

Examples of Perspective Viewing



https://en.wikipedia.org/wiki/Perspective_(graphical)

False Perspective

Satire on False Perspective is the title of an engraving produced by William Hogarth in 1754 for his friend Joshua Kirby's pamphlet on linear perspective.

The intent of the work is clearly given by the subtitle:

> *Whoever makes a* DESIGN *without the Knowledge of* PERSPECTIVE *will be liable to such Absurdities as are shewn in this* Frontiſpiece

The work shows a scene that provides many deliberate examples of confused and misplaced perspective effects. Although the individual components of the scene seem self-consistent, the scene itself can be classed as an example of an impossible object.

https://en.wikipedia.org/wiki/Satire_on_False_Perspective

Summary of Classical and Computer Viewing

Classical: many more, ranging from multi-view orthographic to one-, two-, and three-point perspectives.

- Dependence of object specifications and camera parameters.

Computer: Parallel & Perspective viewing are possible.

- Independence of object specifications and camera parameters, decided by the programmer.
- Trade-off: Cannot achieve different forms a view such as, 1-,2-, and 3-point perspective views.

Viewing and Camera in OpenGL

OpenGL Viewing

Viewing using pipeline architecture consists of two fundamental operations:

1. Position & Orient the camera – part of view transformation.
2. Application of projection – projection transformation.

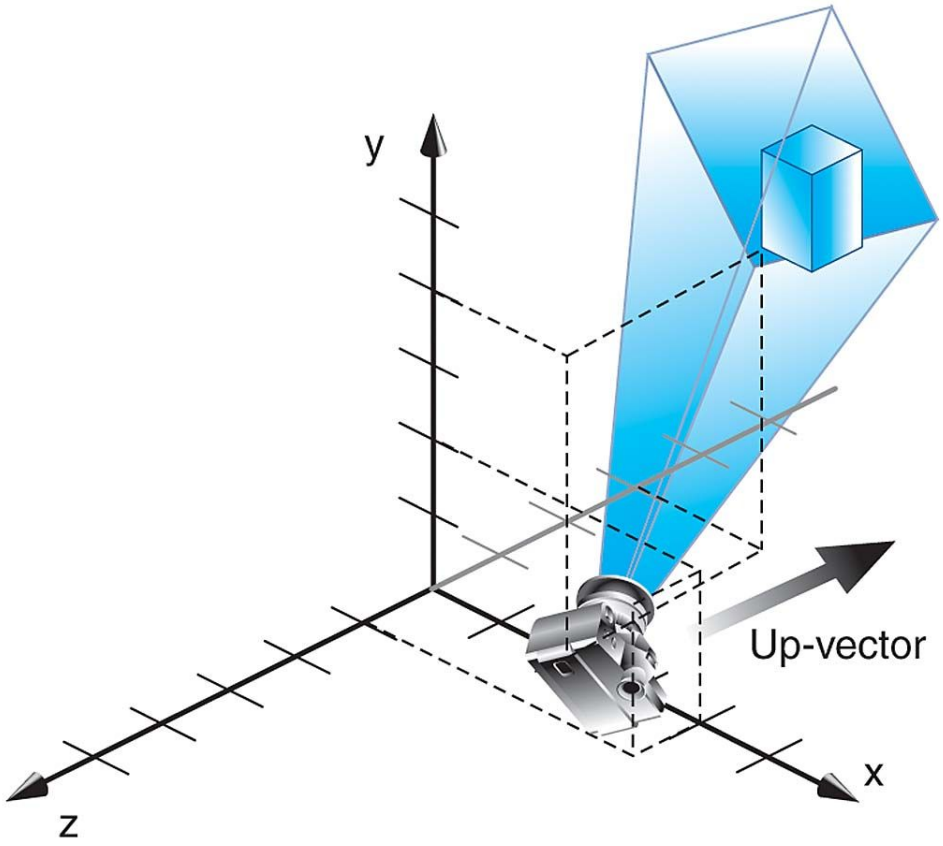


Image courtesy: OpenGL Red Book

Default OpenGL Settings

- Camera @ origin of object frame.
- Orthogonal View: Viewing volume is a solid cube of length 2, centered at origin.
- Projection Plane: Plane z = 0.
- Direction of Projection: Along z-axis.
- Recap on matrices:
  - Model-view and projection matrices: initialized to identity matrix.
  - Parameters in glOrtho set the projection matrix.

Positioning of the Camera

Uses of model-view matrix:

1. Modeling: Positioning objects in space.
2. Viewing: Transforming object frame to camera frame, i.e. world coordinates to eye coordinates.

Default position of camera (0,0,0); and objects ≈ (0,0,0).

- It is impossible to view objects through camera, unless
  - the camera is moved towards the object; or
  - the objects are moved in front of the camera.

In OpenGL, camera faces -z axis, by default.

Positioning Camera Frame

Always, (model-)view matrix affects the camera's view of objects that are defined afterwards.

- Aligns to the *state-machine* design, i.e., camera setting is a state parameter.

Due to the equivalence in scenario, **modeling and viewing are combined in modelview matrix**. Let object and camera be at x_0, initially; and model-view matrix changed to C , then,

- Primitives at x_0 in object frame, will be $C.x_0$ in camera frame -- occurs in the viewing pipeline of OpenGL.
- Equivalently, camera stays at x_0 in the object frame, and model-view matrix applied to all other primitives.

Camera-Positioning Approaches

METHOD-1: Specify the position indirectly by applying sequence of transformations. It is non-intuitive, as,

- one would want to define camera before placing objects in scene;
- transformations on camera are backward from expected sequence.

Direct application of **instance transformation**.

- In lines with classical viewing – viewer moves; object in fixed frame.
- Classical viewing is achieved using left-handed camera frame and right-handed object frame.
- OpenGL follows right-handed camera frame and distances measured from the viewer.

Camera-Positioning Approach

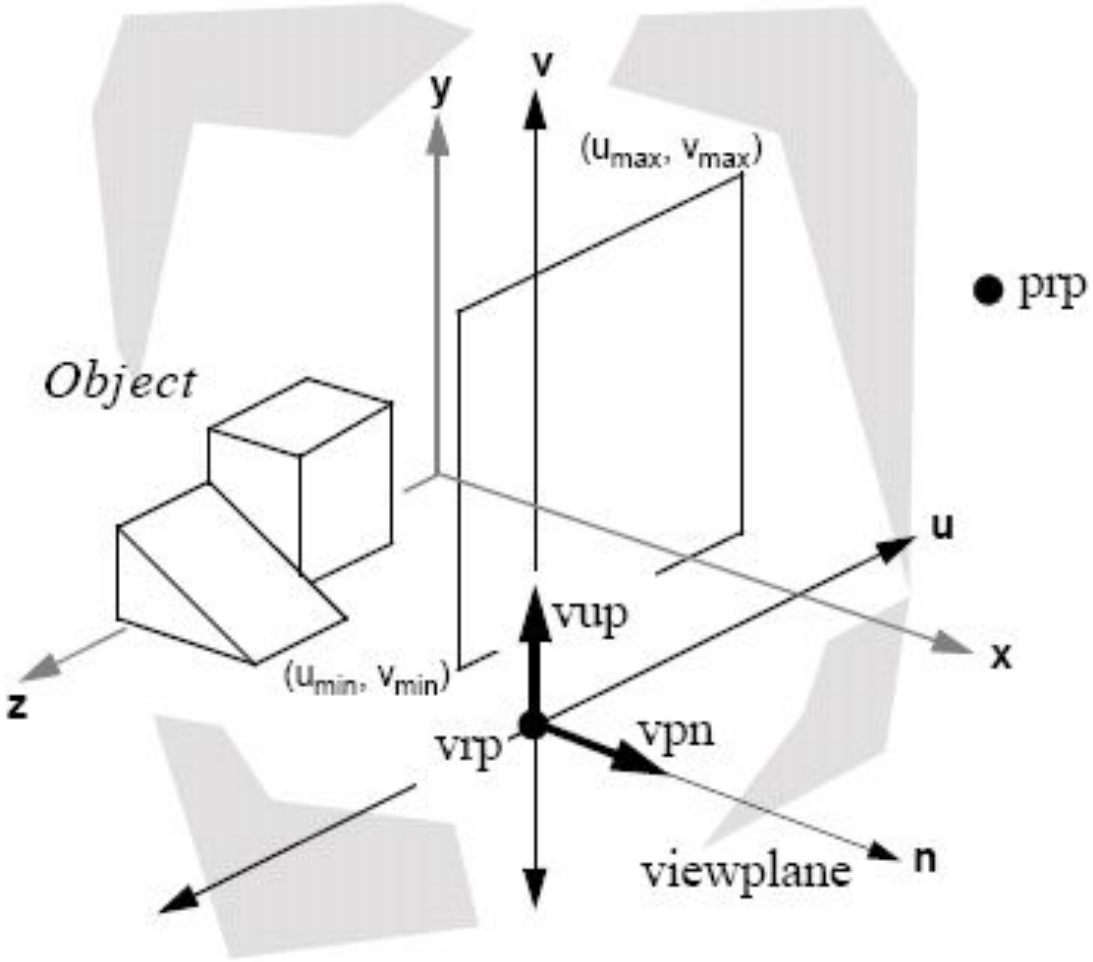
METHOD-2:Use of VRP, VPN, and VUP (i.e., View-Reference Point (VRP), View-Plane Normal (VPN), View-Up vector (VUP)).

1. set-view-reference-point(x,y,z) ; // move camera at VRP in object-frame.
2. set-view-plane-normal(n_x, n_y, n_z) ; // set orientation of projection plane using its normal (VPN).
3. set-view-up(vup_x, vup_y, vup_z) ; // set orientation of camera using its up vector (VUP).

Describe position and orientation of camera in object frame – using the **view-orientation matrix**.

Camera-Positioning Approach

VRP, VUP, and VPN form a set of basis vectors for the coordinate system.



https://www.pling.org.uk/cs/cgv.html

Camera Positioning Approach

METHOD-3: Use eye-point (i.e., camera position) and at-point (i.e., where eye pointed towards).

- **vpn** = **a** − **e**; **vrp** = **e**.
- **a**, **e** in object-frame.

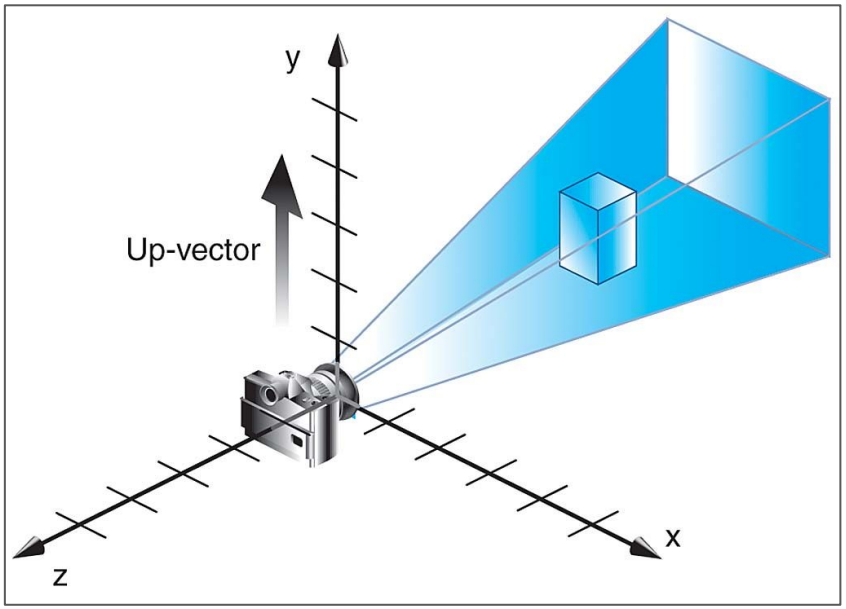
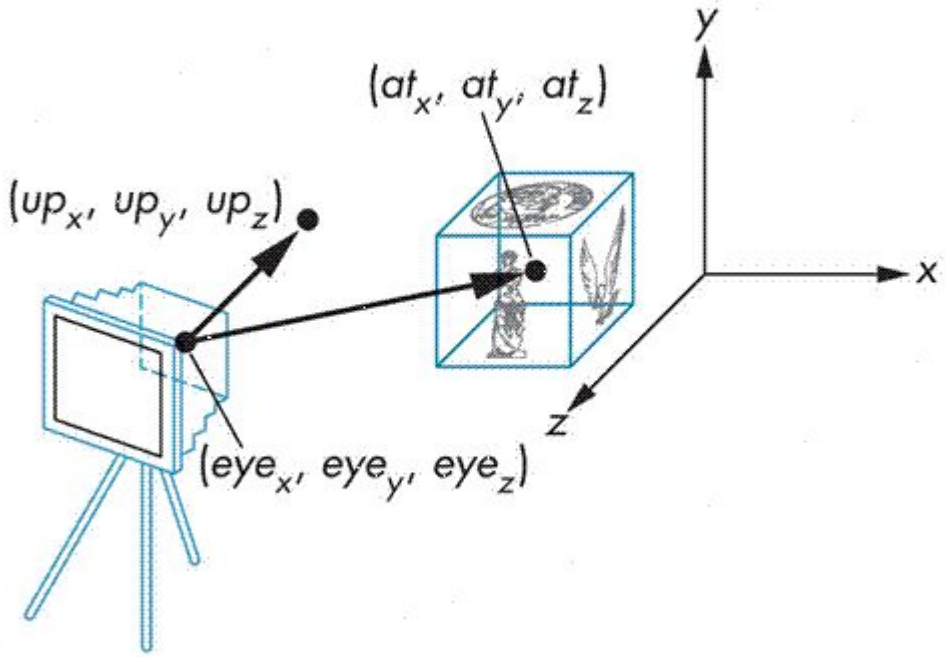


Image courtesy: OpenGL Red Book



(Image Courtesy: Edward Angel's Interactive Computer Graphics).

Camera Positioning Approaches

```
mat4 LookAtRH( vec3 eye, vec3 target, vec3 up )
{
    vec3 zaxis = normal(eye - target);  // The "forward" vector.
    vec3 xaxis = normal(cross(up, zaxis));// The "right" vector.
    vec3 yaxis = cross(zaxis, xaxis);   // The "up" vector.

    // Create a 4x4 view matrix from the right, up, forward and eye position vectors
    mat4 viewMatrix = {
        vec4(       xaxis.x,            yaxis.x,            zaxis.x,       0 ),
        vec4(       xaxis.y,            yaxis.y,            zaxis.y,       0 ),
        vec4(       xaxis.z,            yaxis.z,            zaxis.z,       0 ),
        vec4(-dot( xaxis, eye ), -dot( yaxis, eye ), -dot( zaxis, eye ),  1 )
    };

    return viewMatrix;
}
```

https://www.3dgep.com/understanding-the-view-matrix/

Camera Positioning Approaches

METHOD-4: Using roll, pitch, and yaw – e.g. flight simulator.

METHOD-5: Using polar coordinates – e.g. satellite orbiting a planet.

Camera Frame in Object Frame

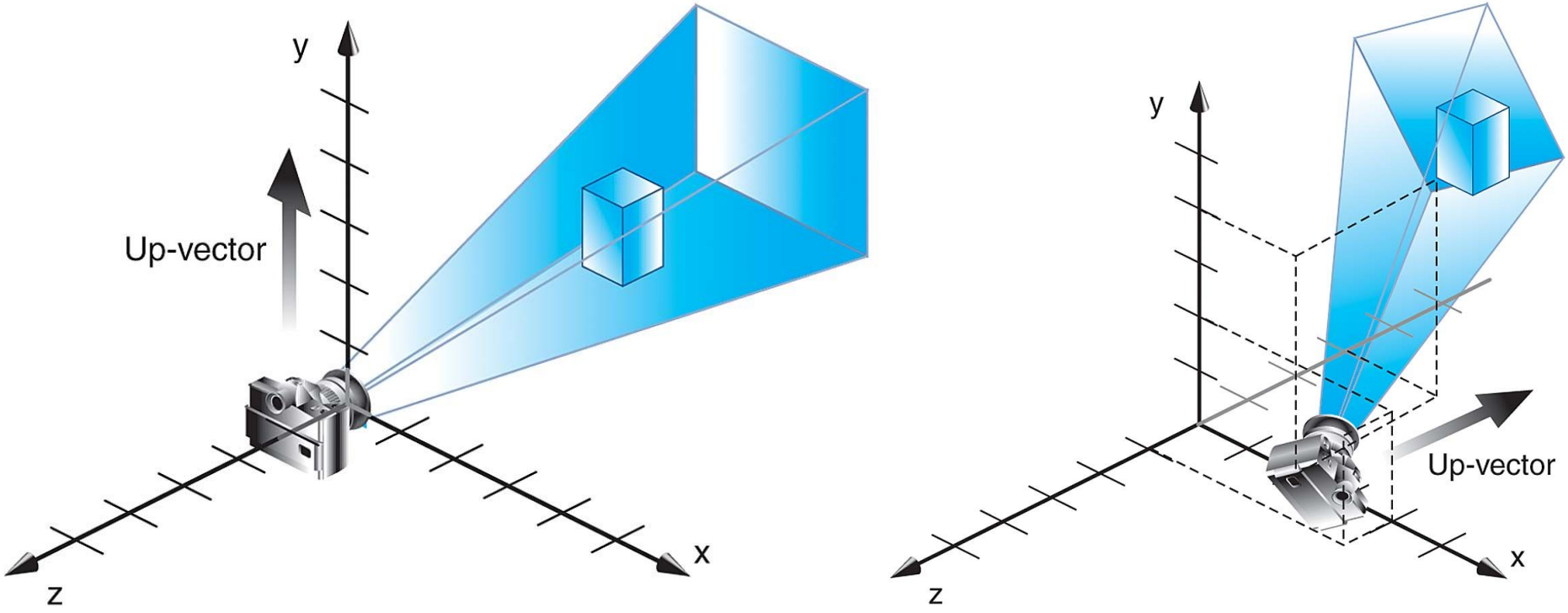


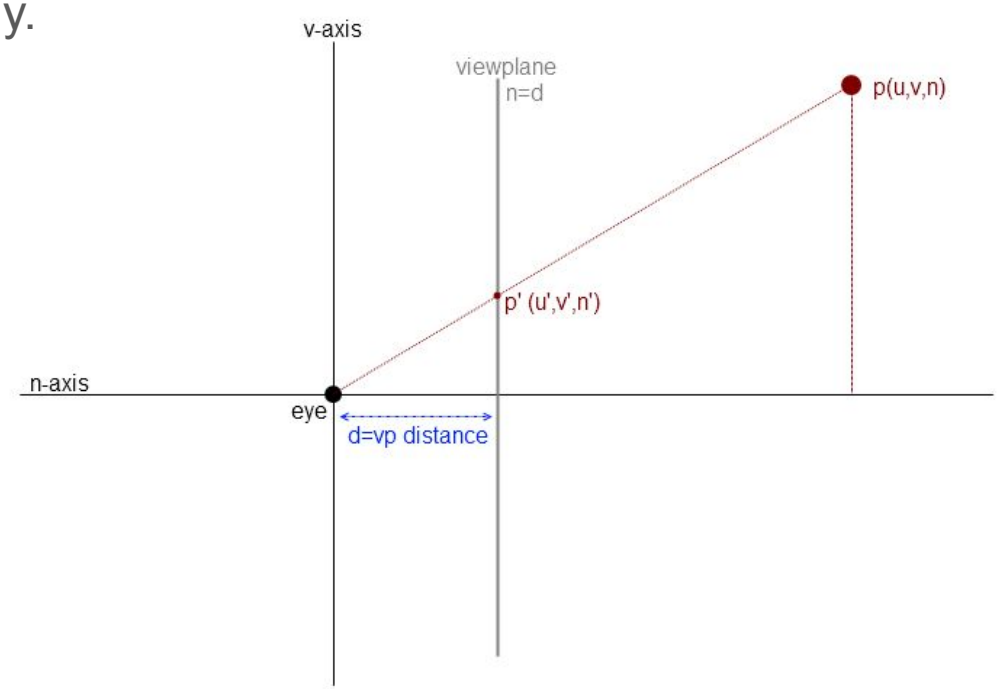
Image courtesy: OpenGL Red Book

Perspective Projection

Perspective Projection

Nonlinear transformations for x and y.

Nonuniform foreshortening:
Division by z – causes **diminution**.



For $p(x, y, z)$ and $p'(x', y', z')$,

$$x' = x.\frac{d}{z} = \frac{x}{\frac{z}{d}}$$

$$y' = y.\frac{d}{z} = \frac{y}{\frac{z}{d}}$$

http://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes/projection/perspective_projection.html

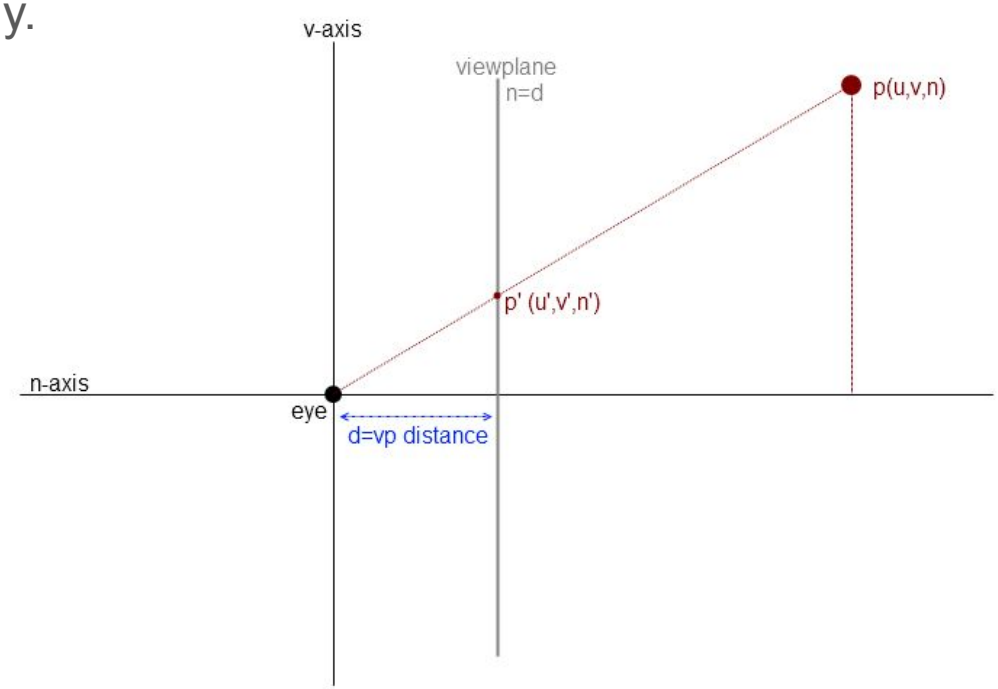
Perspective Projection

Nonlinear transformations for x and y.

Nonuniform foreshortening:
Division by z – causes **diminution**.

Nonuniform foreshortening:
Division by z – causes **diminution**.

- Non-affine transformation; but preserves lines.
- Irreversible transformation – all points on line joining a point to origin are projected on the same point.



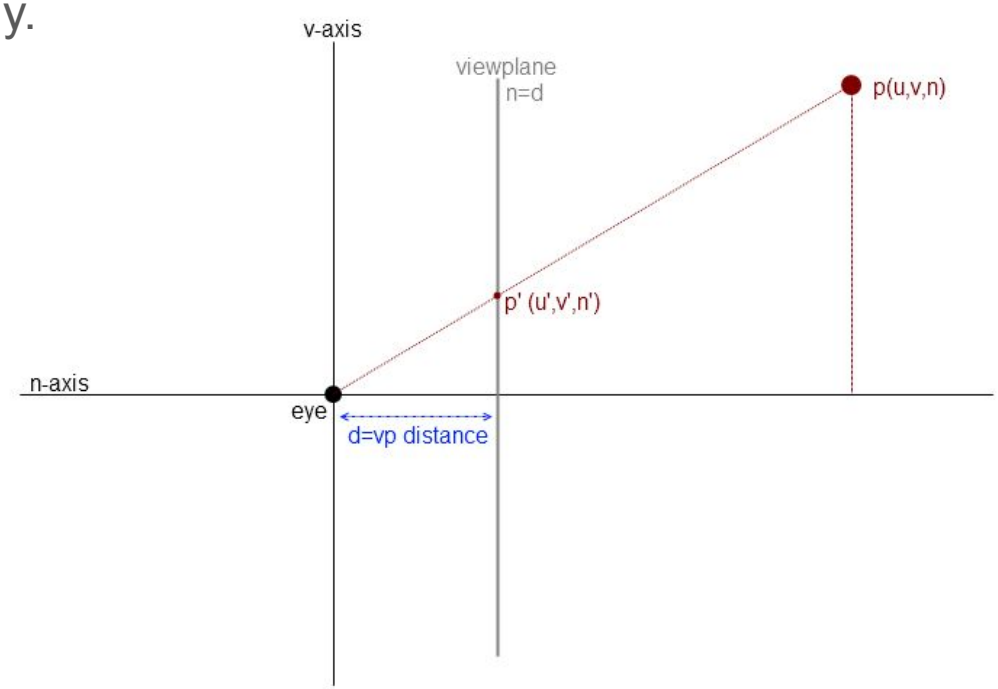
http://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes/projection/perspective_projection.html

Perspective Projection

Nonlinear transformations for x and y.

Nonuniform foreshortening:
Division by z – causes **diminution**.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \frac{z}{d} \end{bmatrix} = \begin{bmatrix} \frac{x}{\frac{z}{d}} \\ \frac{y}{\frac{z}{d}} \\ d \\ 1 \end{bmatrix}$$



Perspective division: Division by weight $w = \frac{z}{d}$

http://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes/projection/perspective_projection.html

Reference Video

https://youtu.be/EqNcqBdrNyI?si=i7JIP_2-CPKYiasR

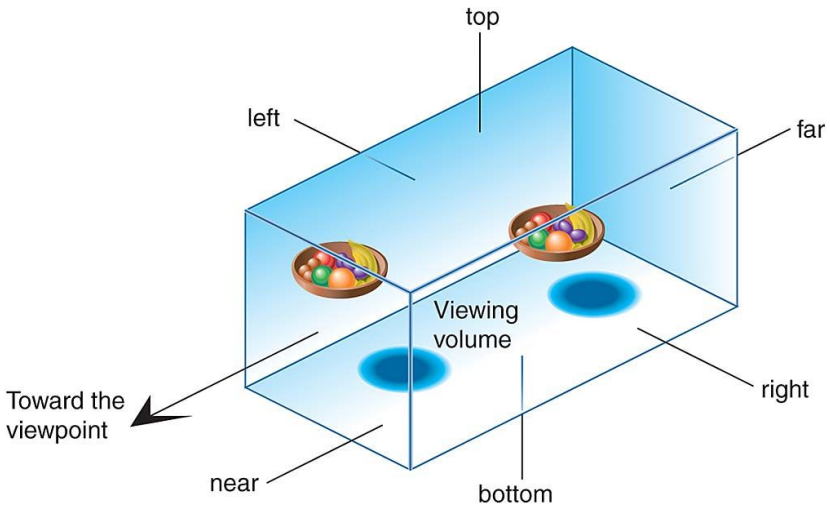
Projection Transformations

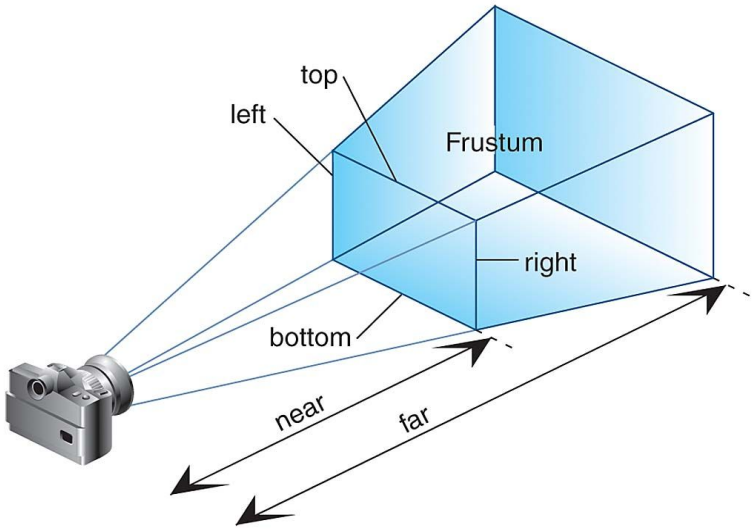
Basic Orthographic Projection

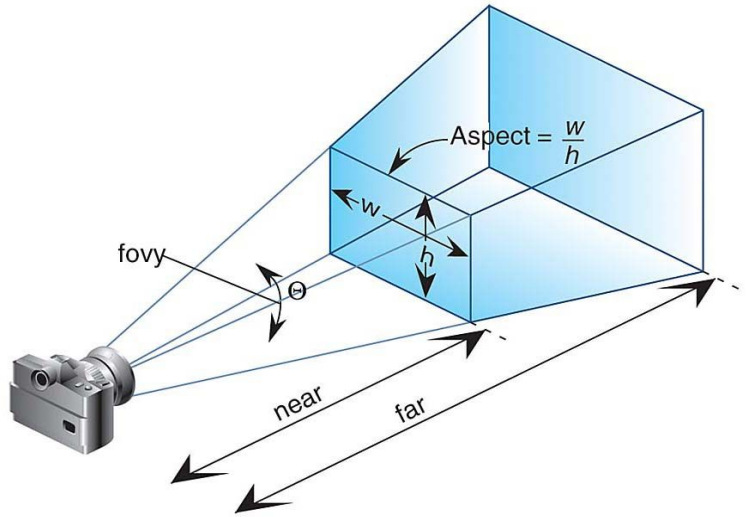
For projection plane z=0, (x,y,z) is projected to (x,y,0).

Using the transformation matrix: $M_{ortho} \cdot p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} xp \\ yp \\ zp \\ 1 \end{bmatrix}$

Specifications in OpenGL







(Top-left, clockwise) Defining 6 planes in viewing volume for orthographic projection, for perspective projection, and defining field of view for perspective projection.

Image courtesy: OpenGL Red Book

Projection Normalization

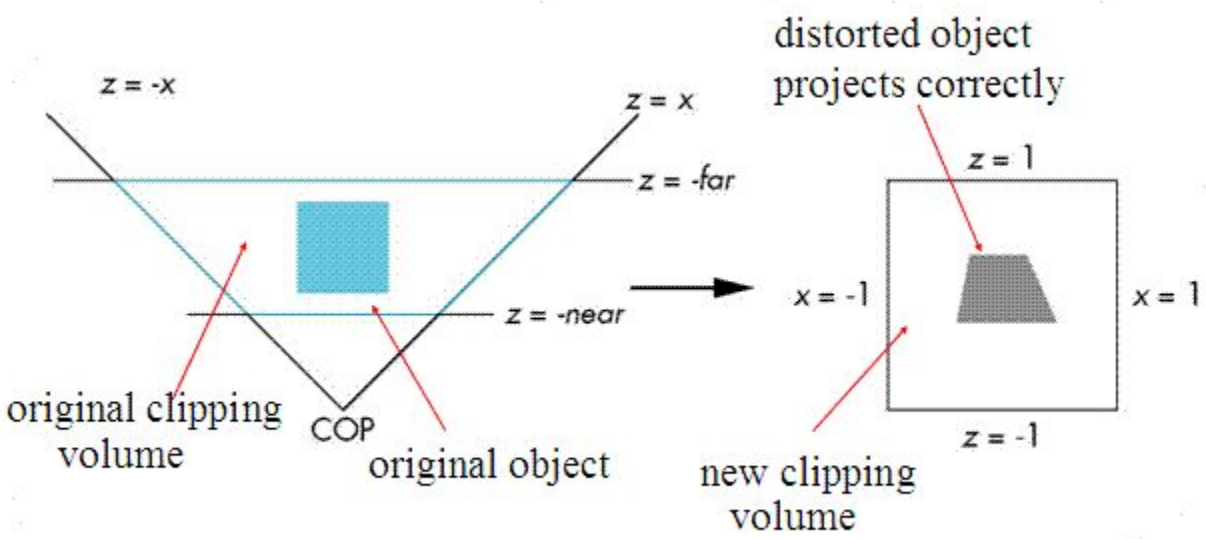
Computer viewing, different from classical viewing, is characterized by:

1. Use of homogeneous coordinate system (for generalization),
2. Retaining depth information (for z-buffer),
3. Use of projection normalization for generalizing projections as linear transformations.

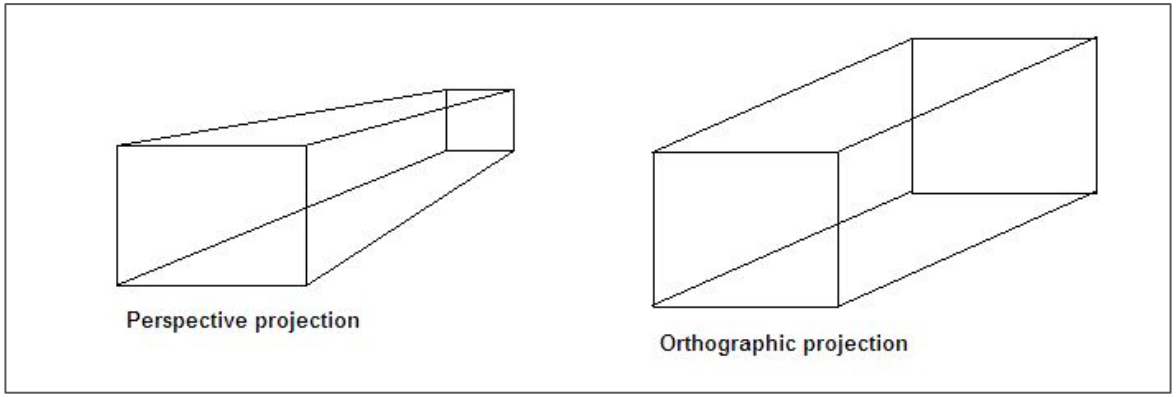
Projection Normalization: Convert all projections to orthogonal projection by first distorting objects such that:

orthogonal projection of distorted object == desired projection of original object.

Normalization Transformation



Schreiner, Angel: Interactive Computer Graphics 6E, 2012



https://gamedev.stackexchange.com/questions/76111/open-gl-perspective-projection-vs-orthographic-projection
https://github.com/ifilot/opengl-cpp-course/blob/master/lesson02/README.md

Projection Normalization

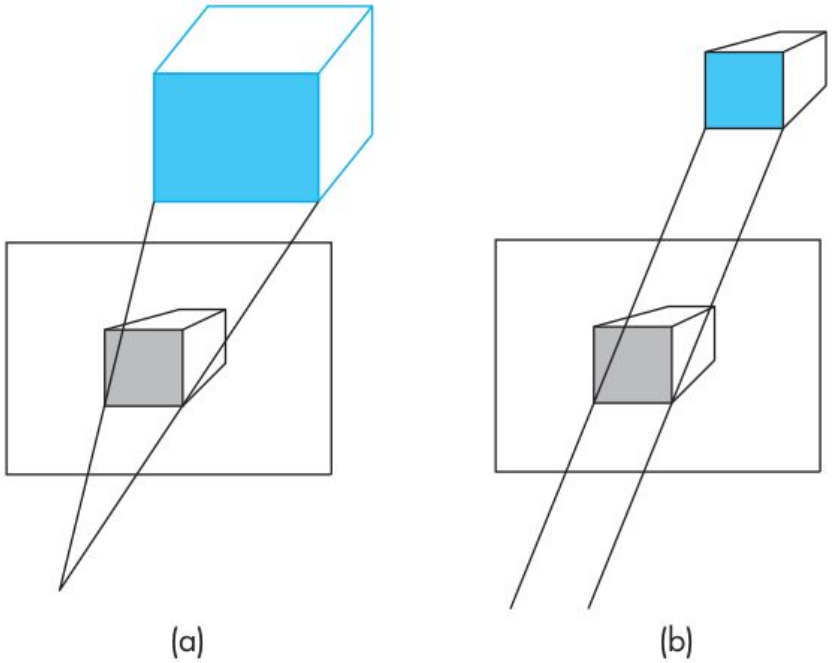


FIGURE 4.23 Predistortion of objects. (a) Perspective view. (b) Orthographic projection of distorted object.

FIGURE 4.24 Normalization transformation.

Projection Normalization

Thus the Projection (Transformation) Matrix is a concatenation of normalization matrix (distortion) and orthographic projection.

- Thus making **perspective=type(parallel).**

Advantages of normalization:

- View volume is distorted to canonical view volume –a cube defined by planes x = ±1, y = ±1, and z = ±1, centered at the origin (0,0,0).
- Both perspective and parallel viewing are supported in the same pipeline – with appropriate loading of normalization matrix.

Orthogonal Projection Matrix in OpenGL

Orthogonal, or orthographic, projection includes the following operations:

- Translating center of viewing volume to center of canonical view volume.
- Scaling viewing volume to canonical view volume.

Using basic orthogonal matrix M_{ortho}, and computing normalization matrix N_{ortho}: we get the final orthographic projection matrix P.

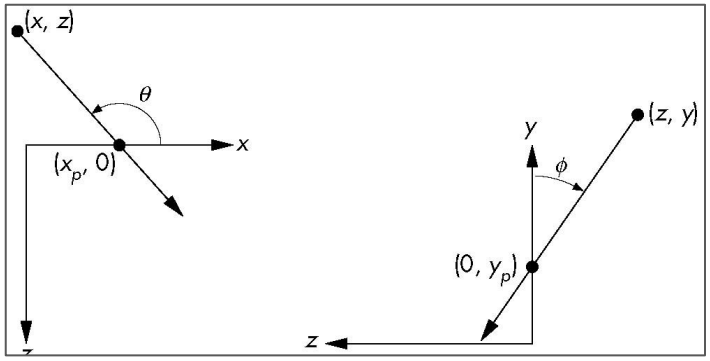
$$N_{ortho} = ST = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P = M_{orth} N_{ortho} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot N_{ortho}$$

Oblique Projection Matrix in OpenGL

$$Pp = \begin{bmatrix} 1 & 0 & \cot\theta & 0 \\ 0 & 1 & \cot\phi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + z\cot\theta \\ y + z\cot\phi \\ 0 \\ 1 \end{bmatrix}$$

$$\Rightarrow P = M_{orth} H(\theta, \phi)$$



Schreiner, Angel: Interactive Computer Graphics 6E, 2012

Oblique Projection Matrix in OpenGL

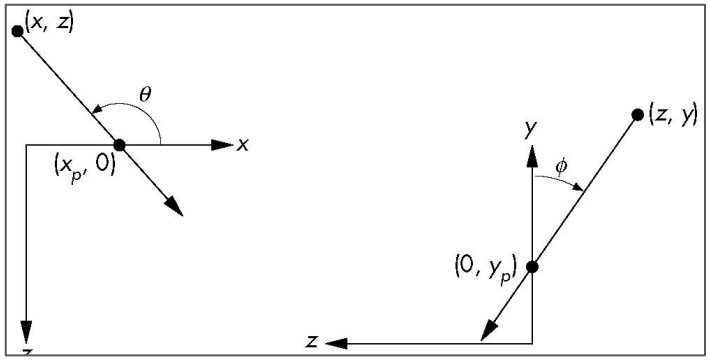
Add specifications of canonical view volume:

$$P = M_{orth} STH(\theta, \phi)$$

$$= M_{orth} \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{far-near} & -\frac{far+near}{near-far} \\ 0 & 0 & 0 & 1 \end{bmatrix} . H(\theta, \phi)$$

$$= M_{orth} N_{ortho} H(\theta, \phi)$$

Thus, $N_{oblique} = N_{ortho} . H(\theta, \phi)$



Schreiner, Angel: Interactive Computer Graphics 6E, 2012

Perspective Projection Matrix in OpenGL

Perspective projections are:

- Represented as a canonical orthogonal projection of distorted objects.
- Characterized by angles that projectors make with the projection plane (z=0).

Hence, 2-step transformation:

1. Oblique projections using shear
2. Perspective division

Perspective Projection Matrix in OpenGL

Consider,

$$N.p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ \alpha z + \beta \\ -z \end{bmatrix} = \begin{bmatrix} -x/z \\ -y/z \\ -(\alpha + \beta/z) \\ 1 \end{bmatrix}$$

After an orthographic projection along z-axis, we get:

$$(M_{ortho}N)p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ -z \end{bmatrix} = \begin{bmatrix} -\frac{x}{z} \\ -\frac{y}{z} \\ 0 \\ 1 \end{bmatrix}$$

N is a non-singular matrix that transforms original volume to a new *distorted* volume.

Perspective Projection Matrix in OpenGL

Consider,

$$N.p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ \alpha z + \beta \\ -z \end{bmatrix} = \begin{bmatrix} -x/z \\ -y/z \\ -(\alpha + \beta/z) \\ 1 \end{bmatrix}$$

After an orthographic projection along z-axis, we get:

$$(M_{ortho}N)p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ -z \end{bmatrix} = \begin{bmatrix} -\frac{x}{z} \\ -\frac{y}{z} \\ 0 \\ 1 \end{bmatrix}$$

N is a non-singular matrix that transforms original volume to a new *distorted* volume.

Consider $0 < near < far$ which are near/front and far/back clipping planes. Let us now consider as the near and front clipping planes, respectively.

With $z = -near$ and $z = -far$

Mapping to (-1,1), respectively, we get: $-\left(\alpha + \frac{\beta}{-near}\right), -\left(\alpha + \frac{\beta}{-far}\right)$

Thus, these values in N make it the **perspective- normalization matrix.**

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

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

A Note on Perspective Division

The following nonlinear mapping preserves depth order:

$$z'' = -(\alpha + \frac{\beta}{z})$$

For $z_1 > z_2$, after transformation we get: $-\left(\alpha + \frac{\beta}{z_1}\right) > -\left(\alpha + \frac{\beta}{z_2}\right)$

Thus, this representation preserves depth sorting order and facilitates **hidden surface removal** in the normalized volume.

Perspective Projection Matrix in OpenGL

For $0 < near < far$,

$N(\alpha, \beta), S(k_x, k_y, 1), H(\theta, \phi)$ where

$k_x = -\frac{2*near}{right-left}, k_y = -\frac{2*near}{top-bottom}$

$\theta = \cot^{-1}\left(\frac{left+right}{-2*near}\right),$

$\phi = \cot^{-1}\left(\frac{top+bottom}{-2*near}\right)$

we get: $P = N(\alpha, \beta).\, S(k_x, k_y, 1).\, H(\theta, \phi) =$

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

Summary

- Classical vs Computer viewing
- Camera/viewing matrix in OpenGL
- Perspective projection
- Projection transformations
  - Basic orthographic projection
  - Orthographic projection matrix in OpenGL
  - Perspective projection matrix in OpenGL