

---

# CSE 472

## Projections

## Projection

---

*Projection* –  $PP=P$

3D to 2D, where the 3D space represents a world coordinate system and the 2D space is a window which is mapped to a screen viewport.

# Specifying Projection

---

Two things must be specified

- *Projection plane* and a *center of projection*.

*Projection plane*

- A 2D coordinate system onto which the 3D image is projected. Called VRP for view reference plane.

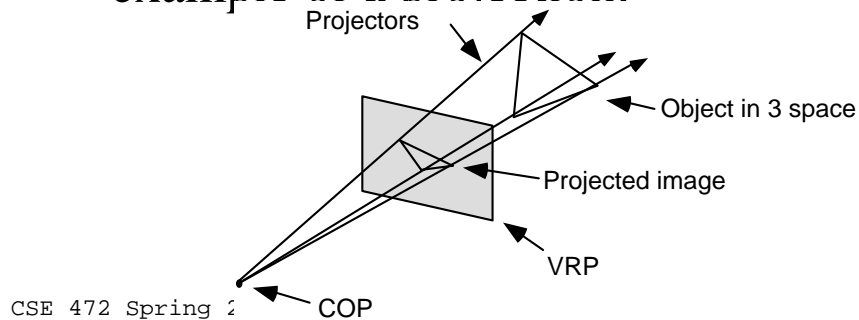
*Center of projection (COP)*

- A point in space which serves as an end point for projectors. a.k.a. PRP for a *projection reference point*.

# Projection Ray

---

a ray originating at the center of projection and passing through a point to be projected. Here is an example of a projection:

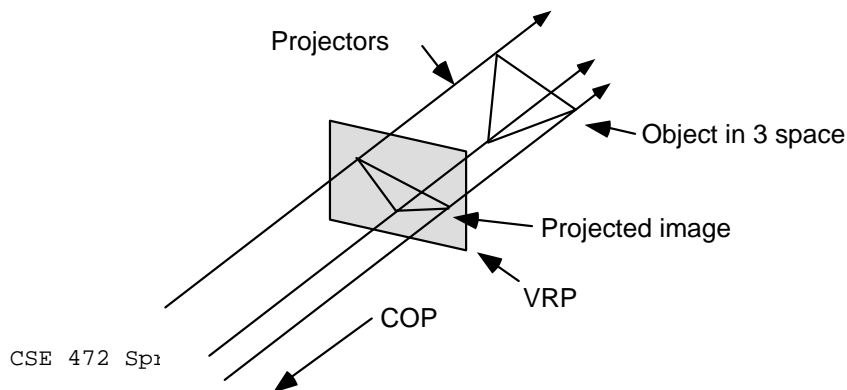


# Parallel Projection

---

A simple case of a projection is if the rays are all in parallel.

- What does this imply about the COP?



# Direction

---

We can't specify the COP for parallel projection

- We'll use Direction of Project (DOP) instead

# Some Trivia

---

## *Planar geometric projection*

- A projection onto a planar surface (plane) using straight lines (geometric). Why would we do otherwise???

## *Foreshortening*

- Varying lengths of lines due to angle of presentation and/or distance from center of projection. Applies to both parallel and perspective projections.

# Orthographic Proj.

---

## *Orthographic projection*

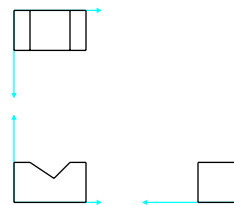
- parallel projection with the direction of projection and the projection plane normal parallel.

## *Elevation*

- an orthographic projection in which the view plane normal is parallel to an axis. This is the simplest of the projections. How do you do it?

## The three elevations

- *front-elevation*
- *top-elevation (plan-elevation)*
- *side-elevation.*



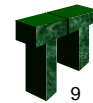
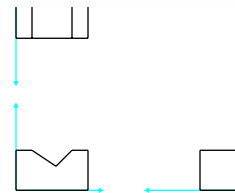
# Elevations

---



## The three elevations

- *front-elevation*
- *top-elevation (plan-elevation)*
- *side-elevation.*



# *Axonometric Ortho. Proj.*

---

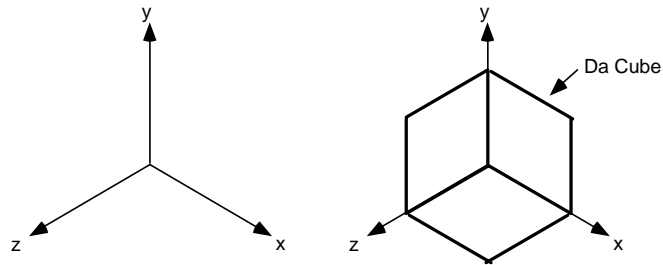
## *Axonometric orthographic projections*

- Use projection planes which are not normal to an axis. They show more than one face of an object at a time. They induce uniform foreshortening unrelated to depth.
- AOP preserves parallelism of lines. It does not preserve angles.

# Isometric projection

## Isometric projection

- Axonometric orthographic projection where the projection plane normal (and the direction of projection) makes identical angles with each principle axis. How many of these are there?



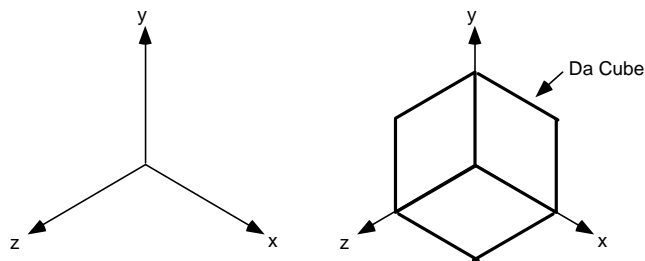
CSE 472 Spring 2009

11

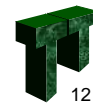
# Isometric Proj.

## Isometric projection

- Identical angles with each principle axis.



CSE 472 Spring 2009



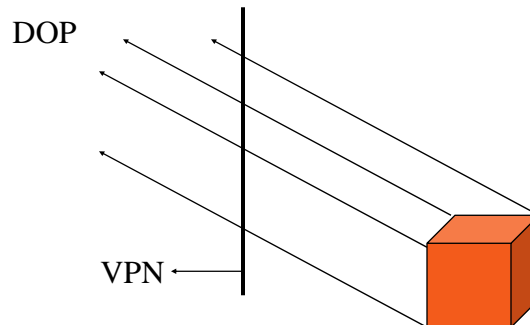
12

# Oblique Projection

---

## *Oblique projection*

- the projection plane normal and the direction of projection at angles to each other.

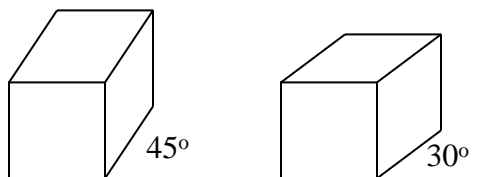


# Cavalier Projection

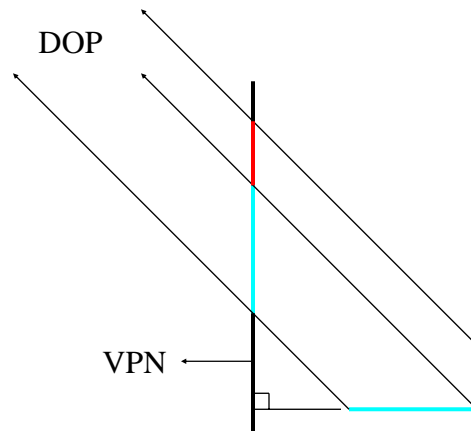
---

## An Oblique projection

- DOP is at 45 degree angle to VPN
- Lines parallel to any axis are foreshortened equally. Lines parallel to the z axis appear at an angle  $\alpha$ , which is dependent upon the direction of projection.
- Two common projections have  $\alpha$  as  $45^\circ$  and  $30^\circ$ .



# Cavalier Proj. Angles



CSE 472 Spring 2009

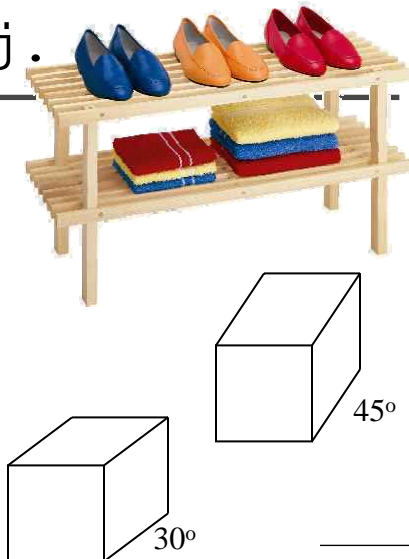
15

# Cavalier Proj.

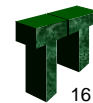
An Oblique projection  
DOP is at 45 degree angle to  
VPN

Lines parallel to any axis are  
foreshortened equally.  
Lines parallel to the z axis  
appear at an angle  $\alpha$ , which  
is dependent upon the  
direction of projection.

Two common projections  
have  $\alpha$  as  $45^\circ$  and  $30^\circ$ .



CSE 472 Spring 2009

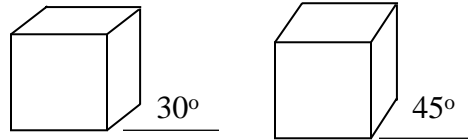


16

# Cabinet projection

---

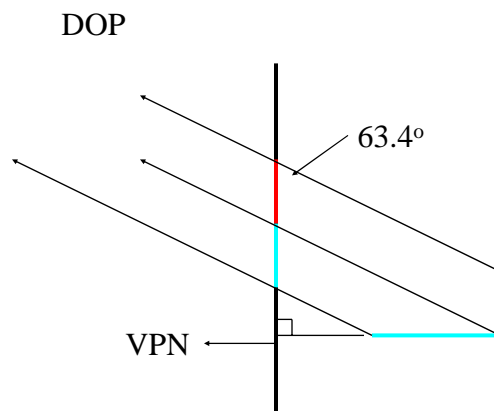
Oblique projection



- projection plane normal is at an  $\arctan(2) = 63.4^\circ$  degree angle to the projection plane. (typically projecting onto the x,y plane)
- Lines parallel of the axis defining the projection plane are foreshortened equally. Lines parallel to the projection plane normal are halved!

# Cabinet Projection

---



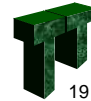
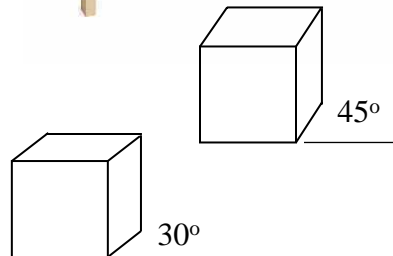
## Cabinet Proj.

---

Oblique projection

Projection plane normal is at an  $\arctan(2) = 63.4^\circ$  degree angle to the projection plane.

Lines parallel of the axis defining the projection plane are foreshortened equally. Lines parallel to the projection plane normal are halved!



## Perspective Proj.

---

Perspective projections have projectors at angles to each other radiating from a center of projection.

- Parallel lines not parallel to the projection plane will not appear parallel in the projection.

# Vanishing Points

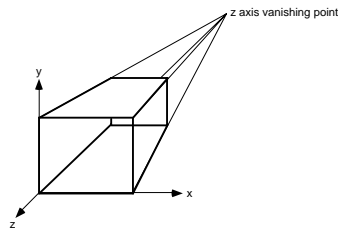
---

If not parallel?

- lines not parallel meet somewhere. In 3D space that point will be at infinity and is referred to as a *vanishing point*. infinite number of vanishing points.

*Axis vanishing points*

- Lines parallel to one of the major axis come to a vanishing point, these are called axis vanishing points. Only three axis vanishing points in 3D space.



# COP in OpenGL

---

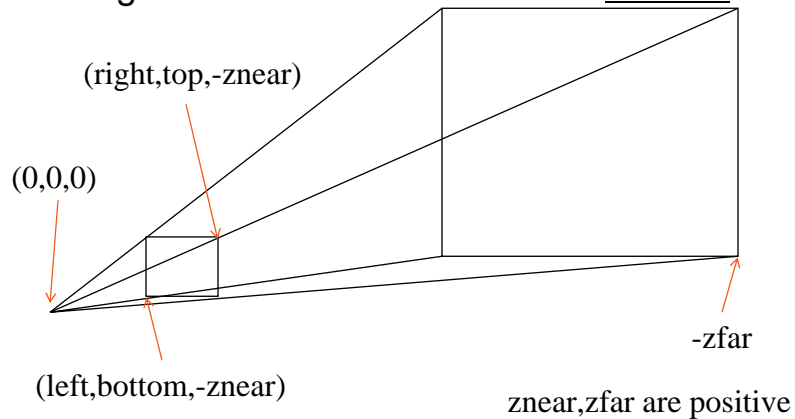
OpenGL always puts the center of projection at 0,0,0

- The projection plane is at  $z = -d$
- This is sometimes called the "focal length" or "f"

## Frustums `glFrustum(left,right,bottom,top,znear,zfar)`

---

The region we can see is called the frustum



## `gluPerspective`

---

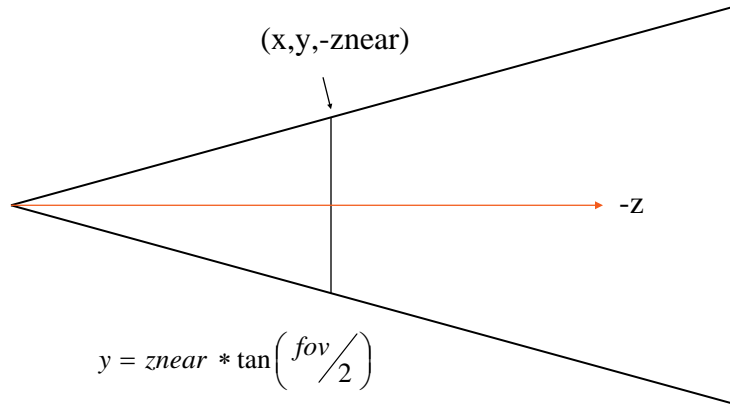
How do we get from:

- `gluPerspective(fovy, aspect, znear, zfar)`

To

- `glFrustum(left,right,bottom,top,znear,zfar)`

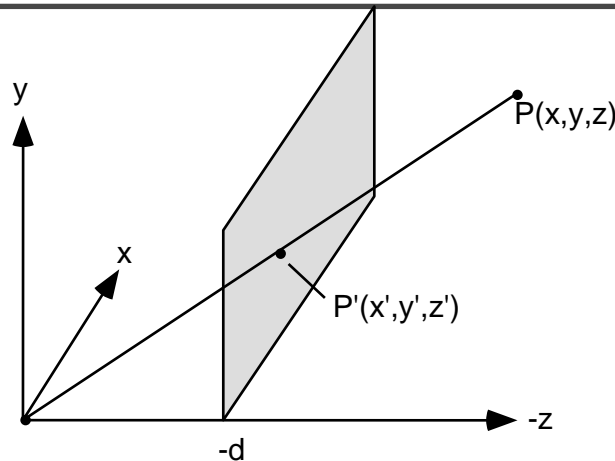
# fov to near frustum



$$y = z_{near} * \tan\left(\frac{fov}{2}\right)$$

$$x = ?$$

# Projection Structure



Pinhole  
Camera  
Model  
of  
Projection

Proportional!

$$\frac{x'}{-d} = \frac{x}{z}, \frac{y'}{-d} = \frac{y}{z}$$

$$x' = \frac{-dx}{z} = \frac{x}{-z/d}, y' = \frac{-dy}{z} = \frac{y}{-z/d}$$

## Matrix?

---

We need division to do projection!

But, matrix multiplication only does multiplication and addition

What about:

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

$$M_{per}P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ -z/d \end{bmatrix}$$

## Homogenous Coords

---

A 3D homogeneous coordinate:

- $(x, y, z, w)$
- We had been saying that  $w$  is 1
- But –
  - $(x, y, z, w)$  corresponds to  $(x/w, y/w, z/w)$
  - Dividing by  $w$  is called **homogenizing**
  - If  $w=1$ ,  $x, y, z$  are unchanged.
  - But, if  $w=-z/d$ ?
    - $(x/(-z/d), y/(-z/d), z/(-z/d)) = (-dx/z, -dy/z, -d)$

$$x' = \frac{-dx}{z} = \frac{x}{-z/d}, y' = \frac{-dy}{z} = \frac{y}{-z/d}$$

## Pseudodepth

---

Map (-near,-far) to (-1,1)

$$z' = \frac{far + near}{far - near} + \frac{2far \cdot near}{far - near} \frac{1}{z}$$

Lines are preserved through the transformation

See [Newman & Sproull '81] for the full derivation

## The Entire Viewing Process

---

Rotate world so that the COP is at 0,0,0  
and DOP is parallel to the Z axis

- gluLookAt()

Apply perspective projection

Homogenize

Window transformation

Viewport transformation

# Window Transformation

---

## Frustum

- Has a left/right/bottom/top

## Window

- Area of the projection plane
- Typically some normalized area with 0,0 in the center
- OpenGL uses (-1,-1) to (1,1)

# Window Transformation

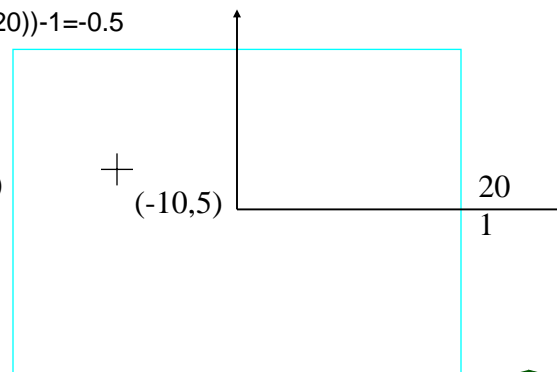
---

$$x_{\text{normalized}} = 2(x' - L) / (R - L) - 1$$

Suppose  $L = -20$ ,  $R = 20$ ,  $x' = -10$

$$x_{\text{normalized}} = 2(-10 - (-20)) / (20 - (-20)) - 1 = -0.5$$

projected: -20  
normalized: -1



# Viewport Trans.

---

## Window

- Area of the projection plane
- Typically some normalized area with 0,0 in the center
- OpenGL uses (-1,-1) to (1,1)

## Viewport

- Area of the computer display window
- Example:
  - (0, 0) to (640, 480)

# Window2Viewport Ex.

---

Assume Window (-1,-1) to (1,1)

- OpenGL calls these normalized device coordinates

Viewport (0, 0) to (640, 480)

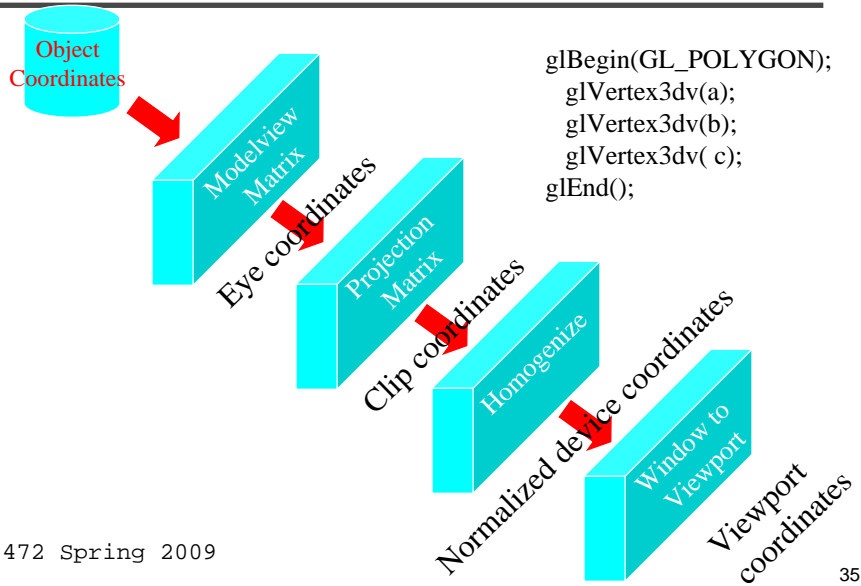
- OpenGL calls these window coordinates

$$x_w = (x_{nd} + 1) \left( \frac{width}{2} \right) + x_{lowerleft}$$

$$y_w = (y_{nd} + 1) \left( \frac{height}{2} \right) + y_{lowerleft}$$

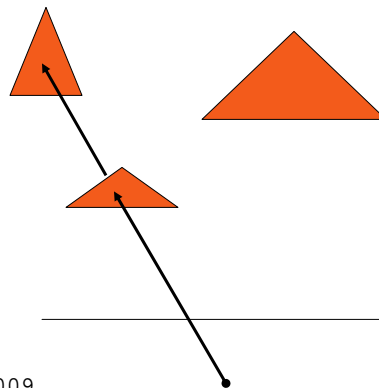


# Within OpenGL



# Raycasting (Visability #1)

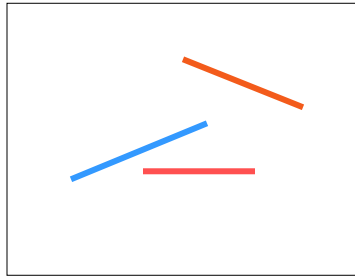
Perform a ray-intersection with every object  
Keep the closest intersection



## Painter's Algorithm(#2)

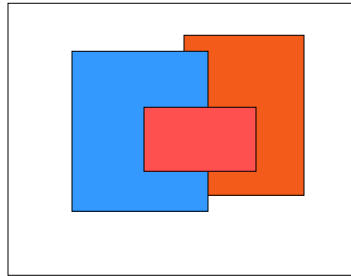
---

Sort objects by farthest z extent  
Draw objects in back to front order  
Closer objects draw over further objects



top

CSE 472 Spring 2009



front

37

## Z Buffer Algorithm(#3)

---

For every pixel, maintain a depth value along  
with the color

Initialize the z buffer to infinity

Interpolate z during scan conversion

setPixel(int x, int y, float z, color c)

if( $z < zBuffer[x][y]$ )

colorBuffer[x][y] = c;

zBuffer[x][y] = z;

**Invariant:** zBuffer[x][y] contains the depth to the closest point  
seen through pixel (x,y) of all objects that have been drawn.

CSE 472 Spring 2009

38

# Graphics System (GPU)

