COMPUTER GRAPHICS AND MULTIMEDIA- (18MCA41C)

UNIT-II

'Three-Dimensional Concepts'

FACULTY:

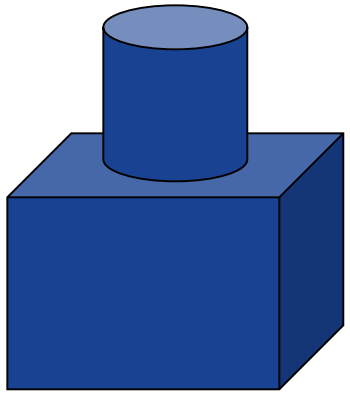
Dr. R. A. Roseline, M.Sc., M.Phil., Ph.D.,
Associate Professor and Head,
Post Graduate and Research Department of Computer Applications,
Government Arts College (Autonomous), Coimbatore – 641 018.

Overview

- 3D Geometric Primitives
  - Point
  - Line
  - Plane
  - Polygon
  - Sphere
- 3D Object Representations
  - Raw data
  - Surfaces
  - Solids
  - High-Level Structure

3D Geometric Primitives

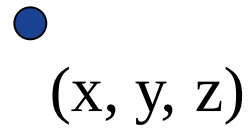
- Point
- Line Segment
- Polygon
- Polyhedron
- Curved Surface
- Solid Object
- Etc.



3D Point

- Specifies a Location
  - Represented by three coordinates
  - Infinitely small

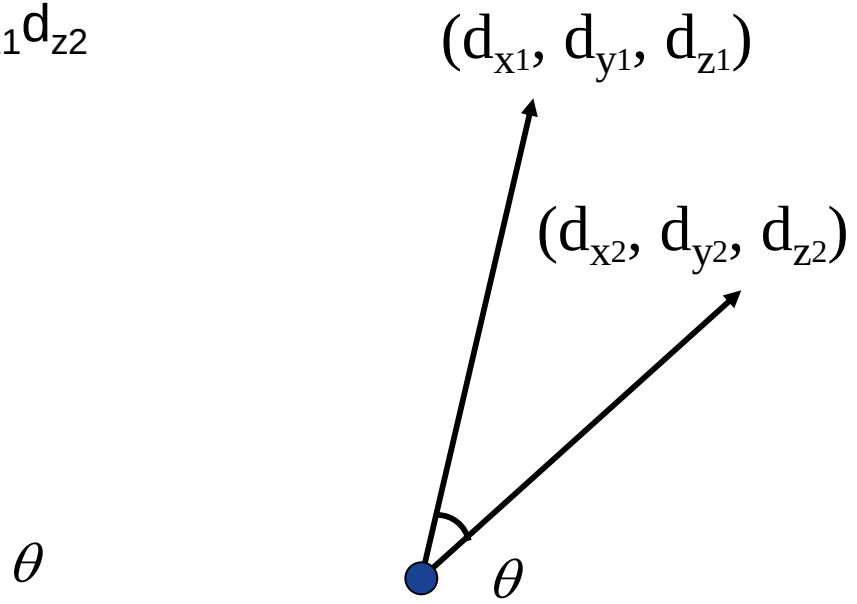
```
typedef struct{
    Coordinate x;
    Coordinate y;
    Coordinate z;
} Point;
```



3D Vector

- Specifies a Direction and a Magnitude
  - Represented by three coordinates
  - Magnitude $\|v\|=\sqrt{d_x d_x + d_y d_y + d_z d_z}$
  - Has no location
- Dot product of two 3D vector
  - $V_1 V_2 = d_{x1}d_{x2} + d_{y1}d_{y2} + d_{z1}d_{z2}$
  - $V_1 V_2 = \|V_1\|\|V_2\| \cos(\ \ )$

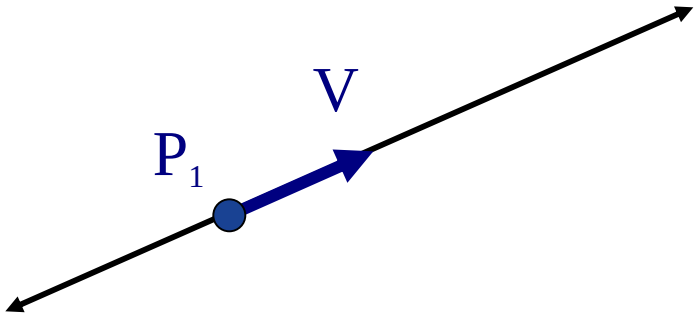
```
typedef struct{
    Coordinate x;
    Coordinate y;
    Coordinate z;
} Vector;
```



3D Line

- Line Segment with Both Endpoints at Infinity
  - Parametric representation
    - $P=P_1+tV$, ($-\infty < t < \infty$)

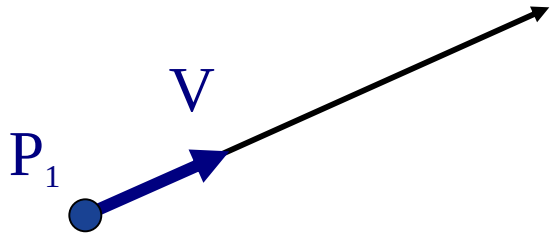
```
typedef struct{
    Point P1;
    Vector V;
} Line;
```



3D Ray

- Line Segment with One Endpoints at Infinity
  - Parametric representation
    - $P=P_1+tV$, ($0 <= t < \infty$)

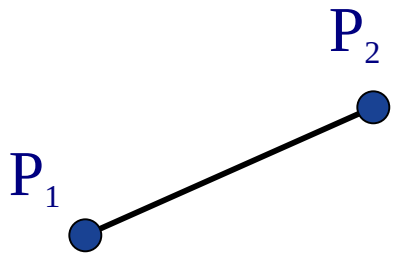
```
typedef struct{
    Point P1;
    Vector V;
} Ray;
```



3D Line Segment

- Specifies a Linear Combination of Two Points
  - Parametric representation
    - $P=P_1 + t(P_2 - P_1)$, $(0 \le t \le 1)$

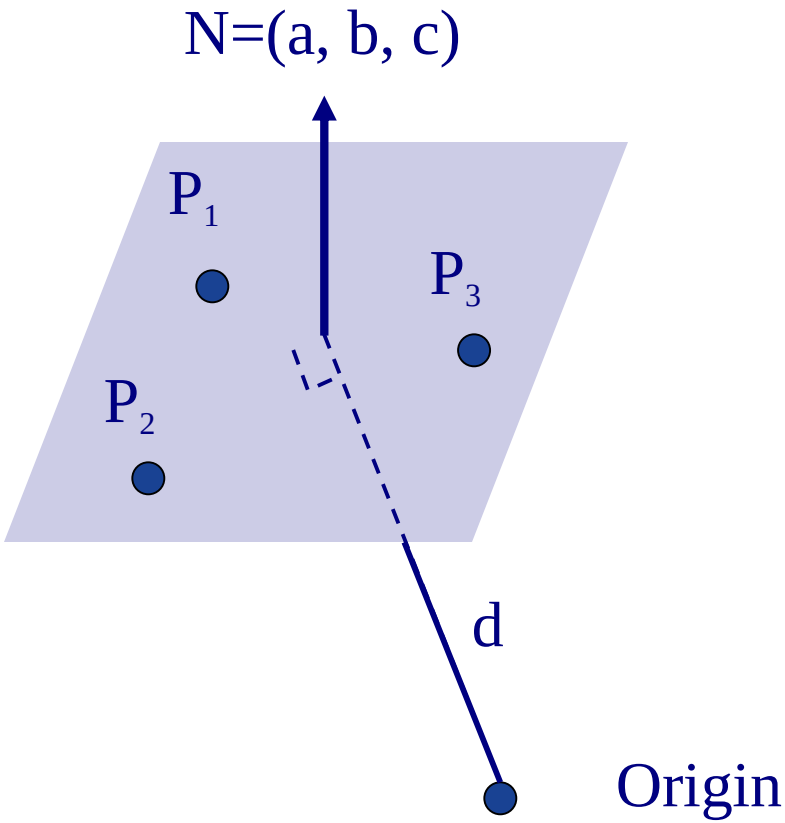
```
typedef struct{
    Point P1;
    Point P2;
} Segment;
```



3D Plane

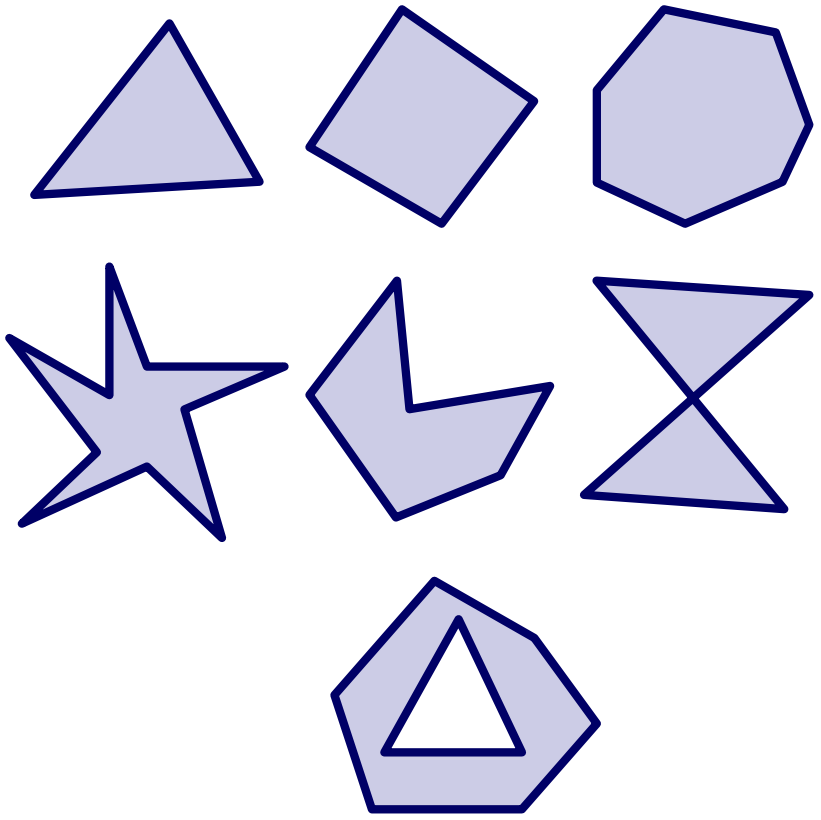
- Specifies a Linear Combination of Three Points
  - Implicit representation
    - P · N + d = 0, or
    - $ax + by + cz + d = 0$

```
typedef struct{
    Vector N;
    Distance d;
} Plane;
```



3D Polygon

- Area “Inside” a Sequence of Coplanar Points
  - Triangle
  - Quadrilateral
  - Convex
  - Star-shaped
  - Concave
  - Self-Intersecting
  - Hole



```
typedef struct{
    Point *Points;
    int npoints;
} Polygon;
```

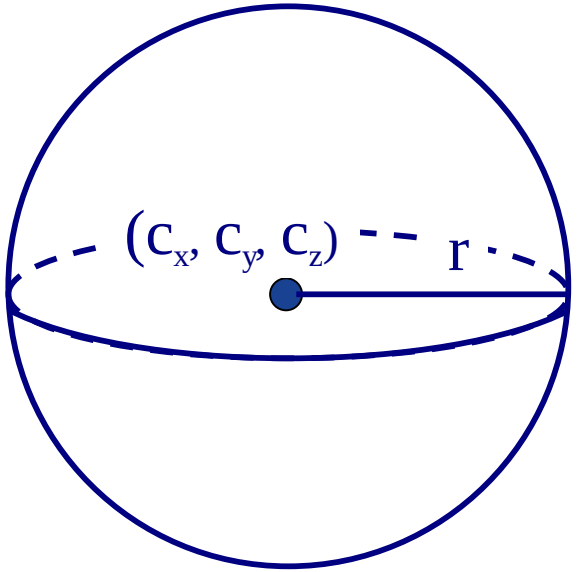
Points are in counter-clockwise order

3D Sphere

- All Points at Distance "r" from Point (c_x, c_y, c_z)
  - Implicit representation
    - $(x-c_x)^2 + (y-c_y)^2 + (z-c_z)^2 = r^2$
  - Parametric representation
    - x= r sin( ) cos( )
    - y= r sin(ϕ) sin(θ)
    - z= r cos(ϕ)

θ

ϕ

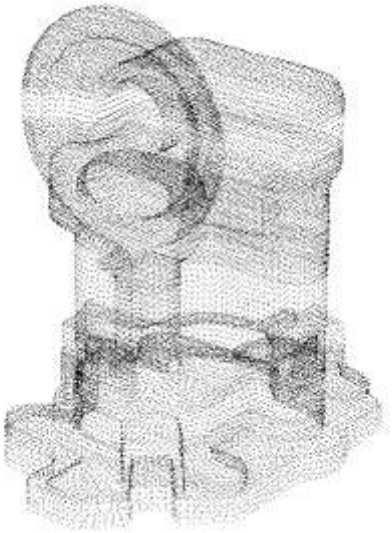


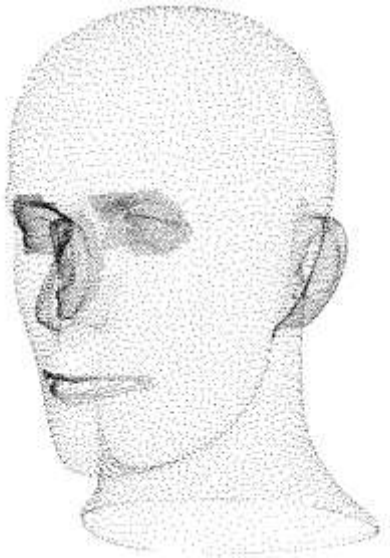
3D Object Representations

- Raw Data
  - Point cloud
  - Range image
  - Polygon soup
- Surfaces
  - Mesh, Subdivision, Parametric, Implicit
- Solids
  - Voxel, BSP tree, CSG, Sweep

Point Cloud

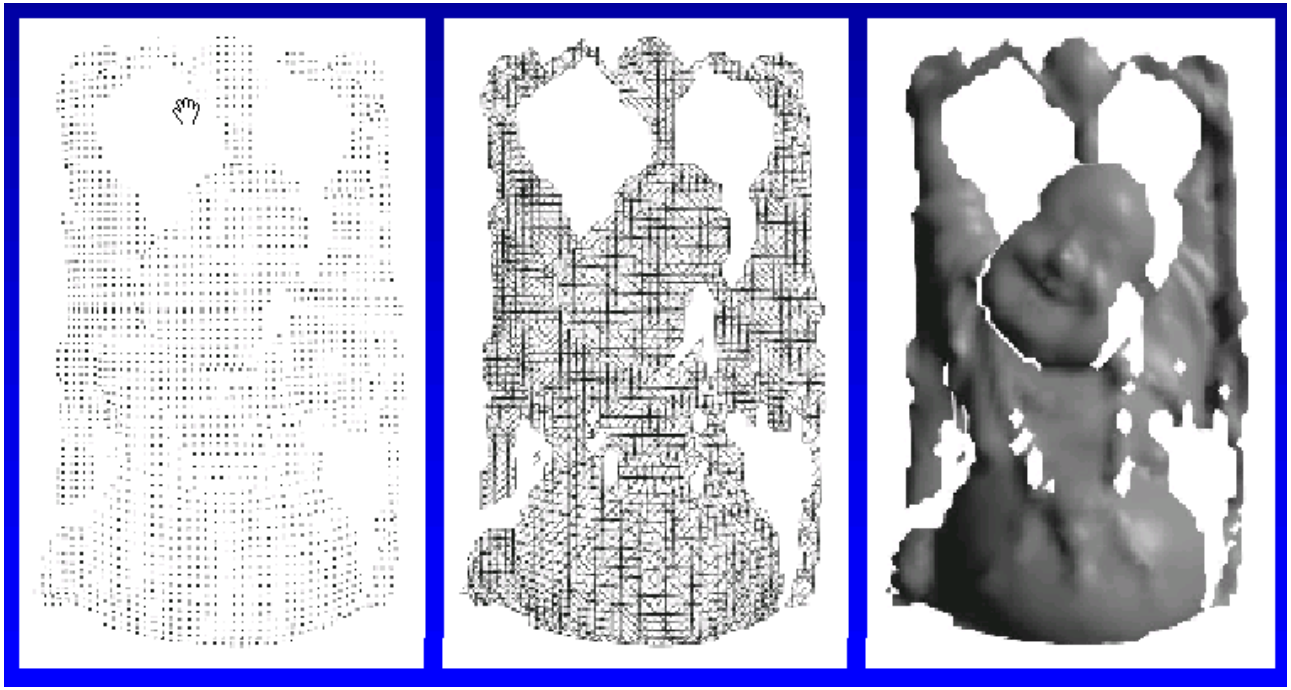
- Unstructured Set of 3D Point Samples
  - Acquired from range finder, computer vision, etc





Range Image

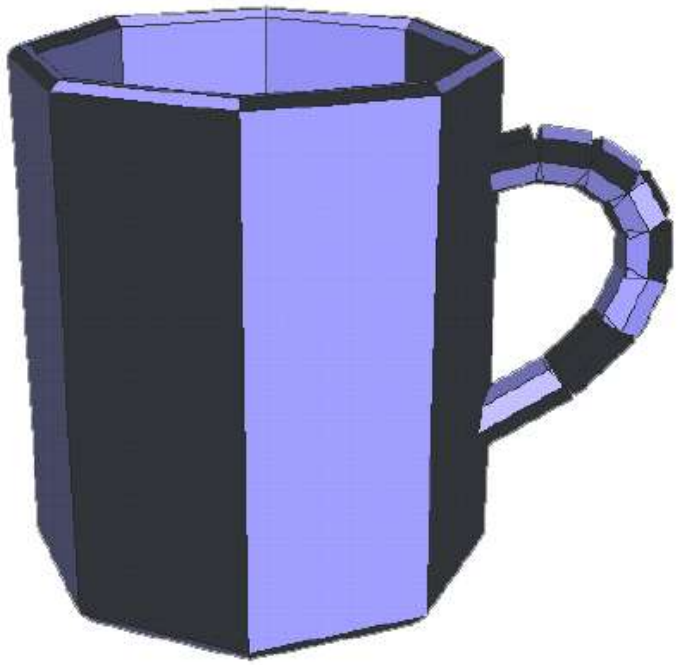
- Set of 3D Points Mapping to Pixels of Depth Image
  - Acquired from range scanner



Range Image Tessellation Range Surface

Polygon Soup

- Unstructured Set of Polygons
  - Created with interactive modeling systems

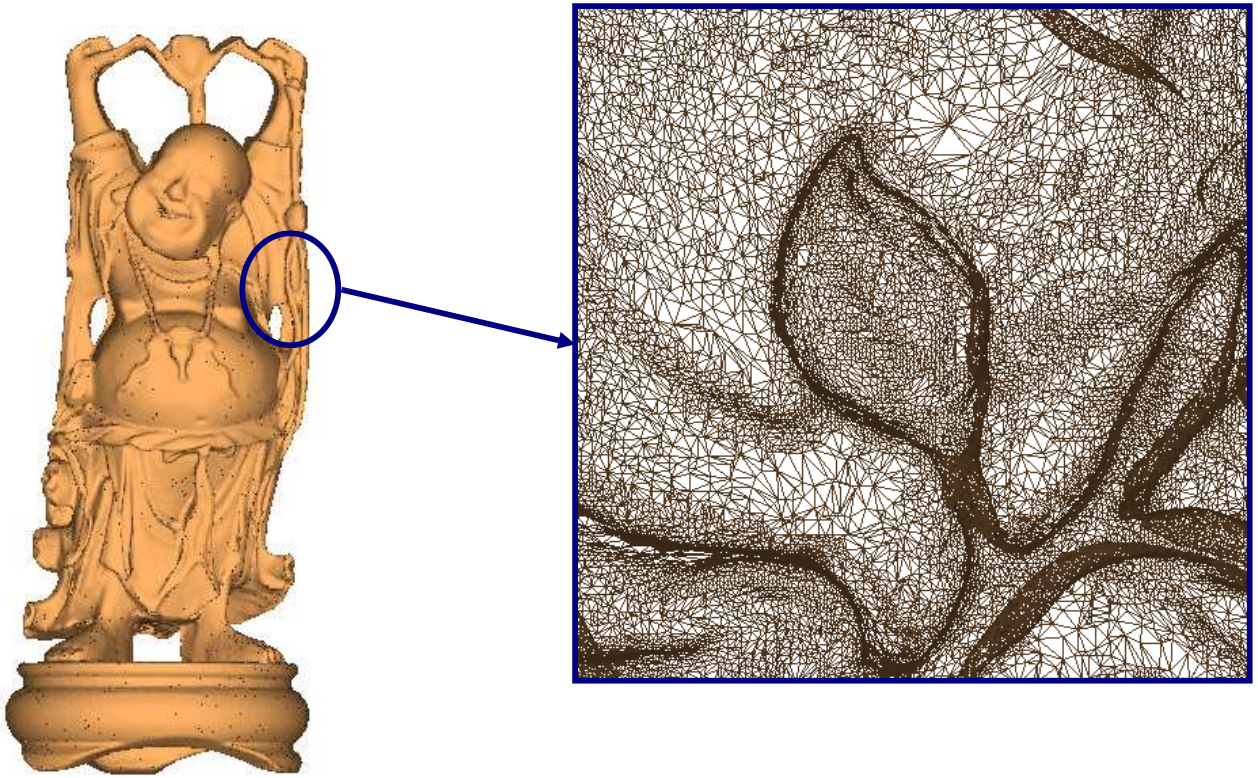


3D Object Representations

- Raw Data
  - Point cloud, Range image, Polygon soup
- Surfaces
  - Mesh
  - Subdivision
  - Parametric
  - Implicit
- Solids
  - Voxel, BSP tree, CSG, Sweep

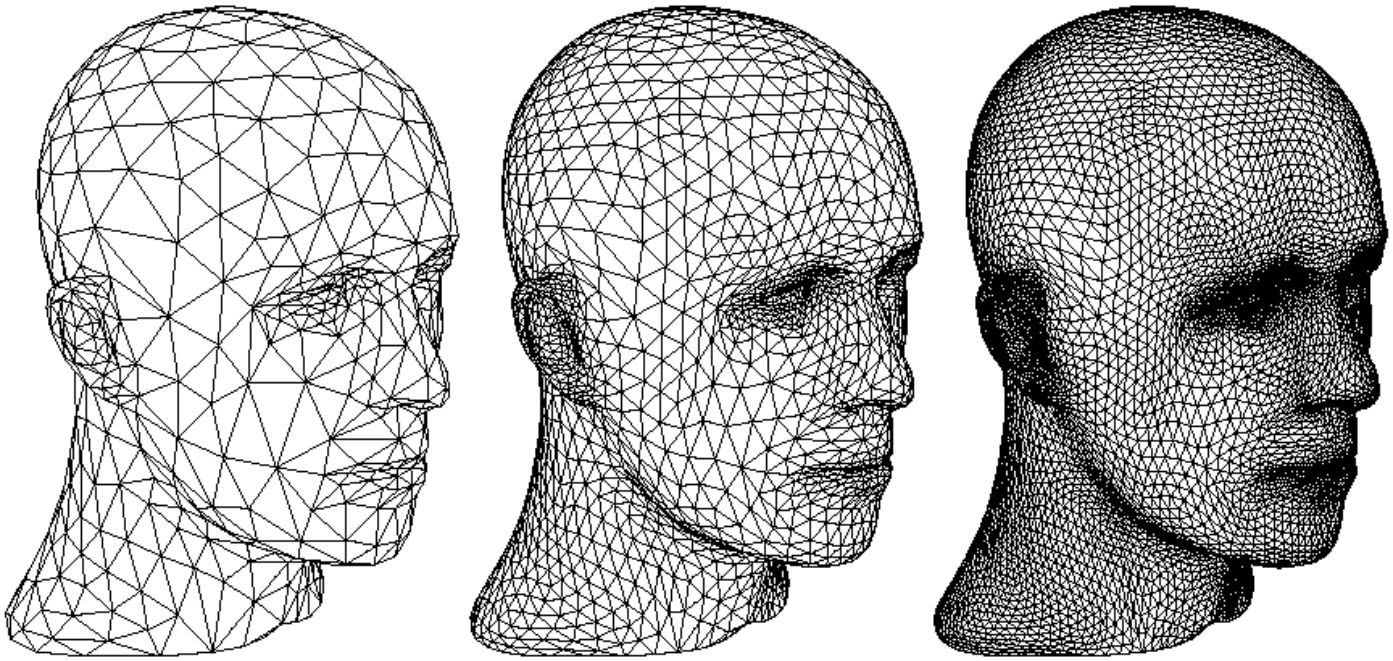
Mesh

- Connected Set of Polygons (Usually Triangles)
  - May not be closed



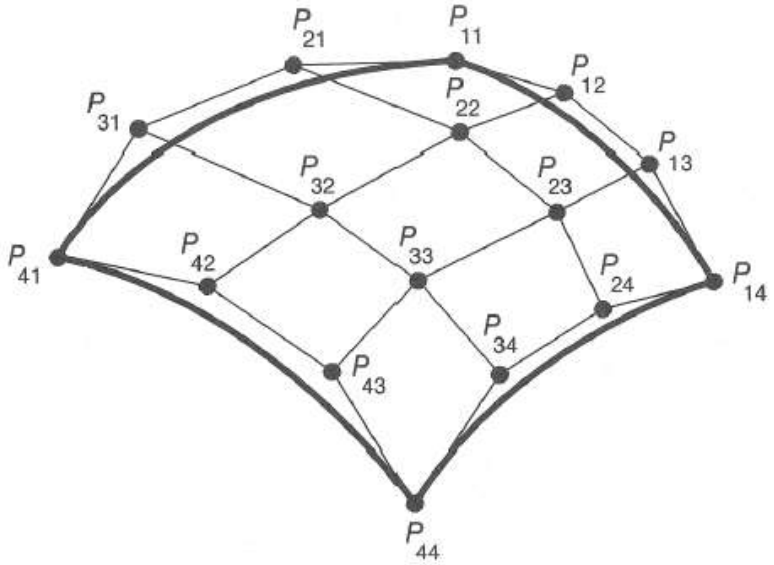
Subdivision Surfaces

- Coarse Mesh & Subdivision Rule
  - Define smooth surface as limit of sequence of refinements



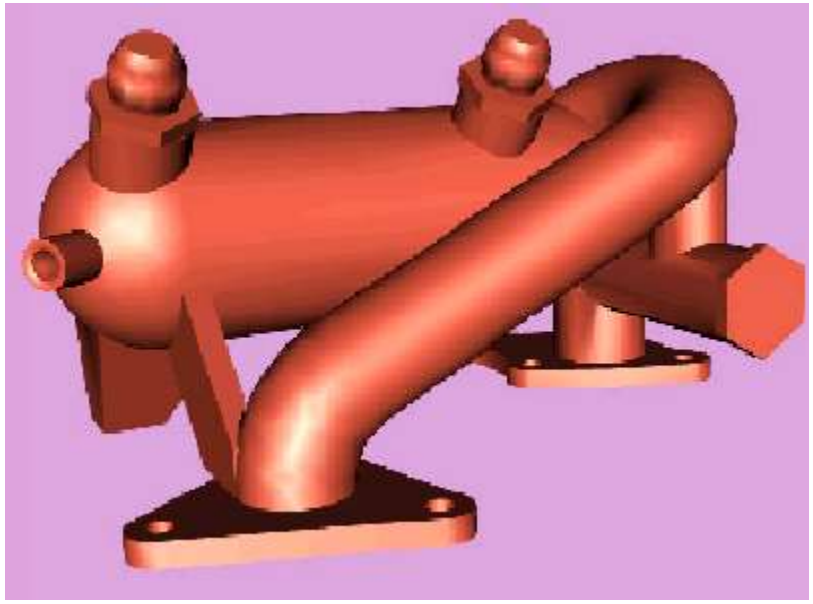
Parametric Surfaces

- Tensor Product Spline Patches
  - Careful constraints to maintain continuity

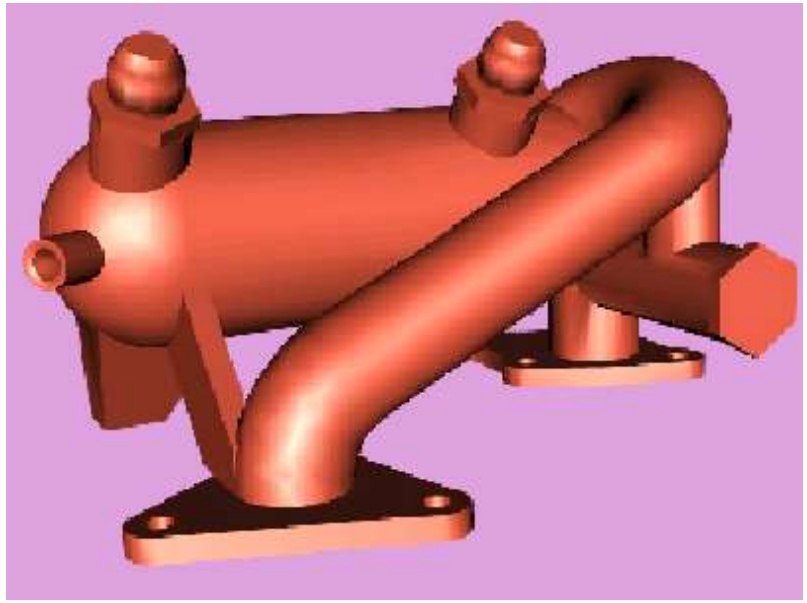


Implicit Surface

- Points satisfying: F(x,y,z) = 0



Polygonal Model



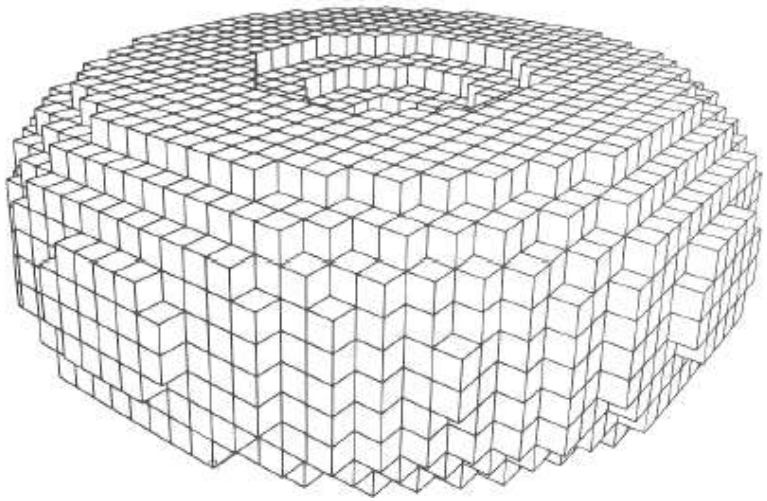
Implicit Model

3D Object Representations

- Raw Data
  - Point cloud, Range image, Polygon soup
- Surfaces
  - Mesh, Subdivision, Parametric, Implicit
- Solids
  - Voxel
  - BSP tree
  - CSG
  - Sweep

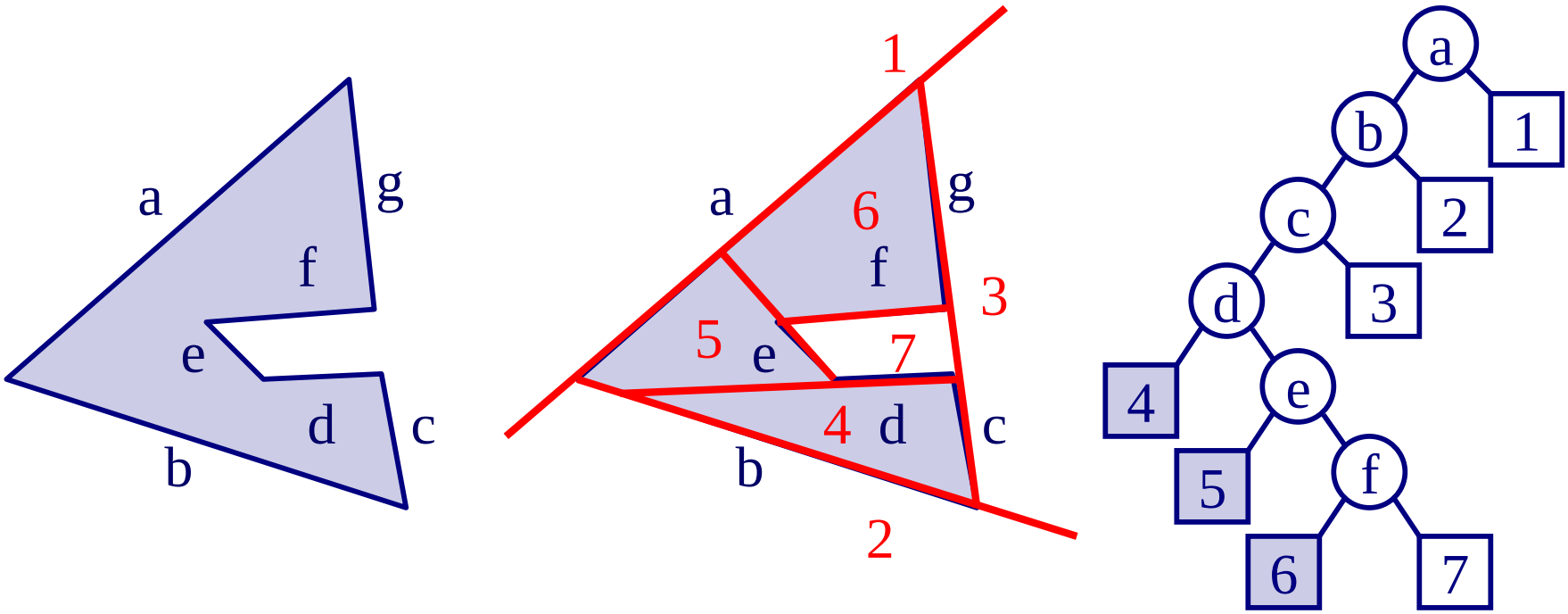
Voxels

- Uniform Grid of Volumetric Samples
  - Acquired from CAT, MRI, etc.



BSP Tree

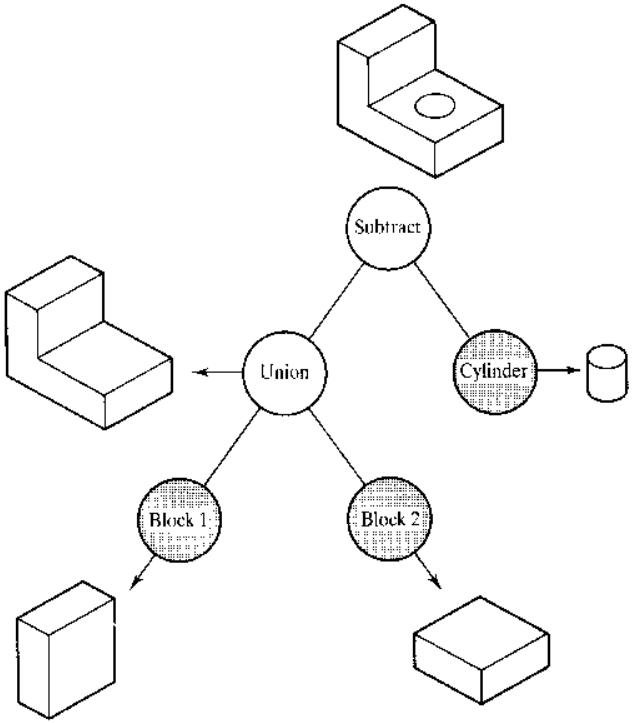
- Binary Space Partition with Solid Cells Labeled
  - Constructed from polygonal representations

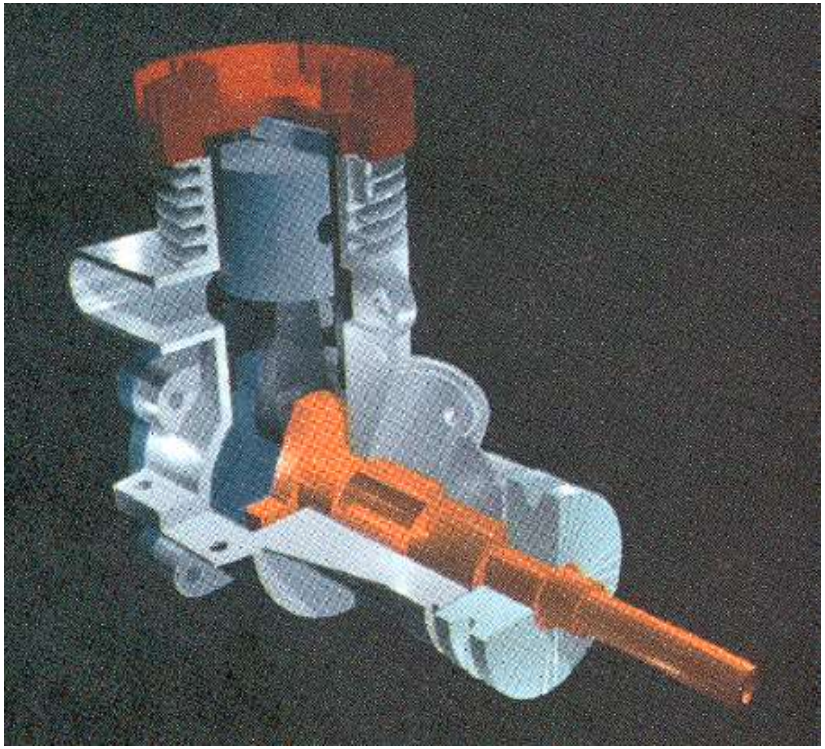


Object Binary Spatial Partition BSP Tree

CSG

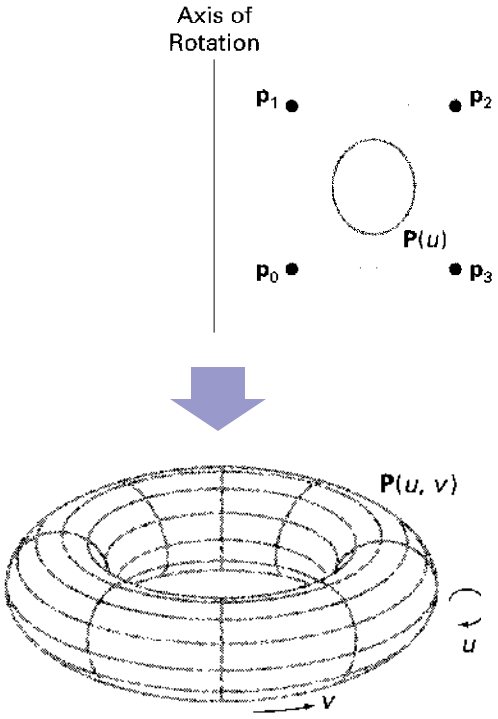
- Hierarchy of Boolean Set Operations (Union, Difference, Intersect) Applied to Simple Shapes



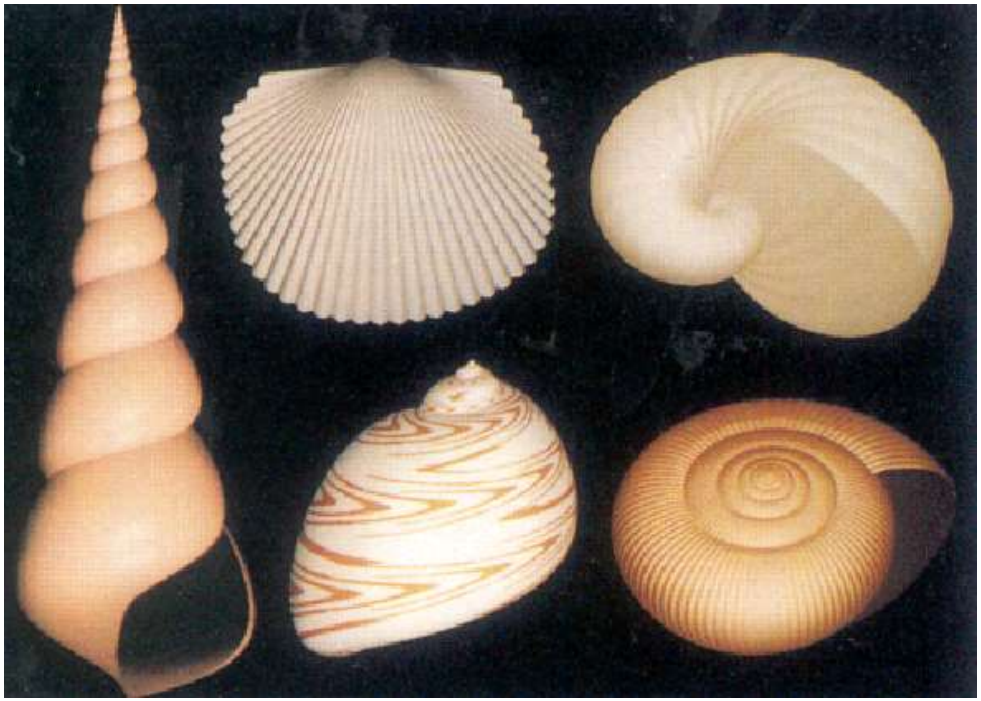


Sweep

- Solid Swept by Curve Along Trajectory

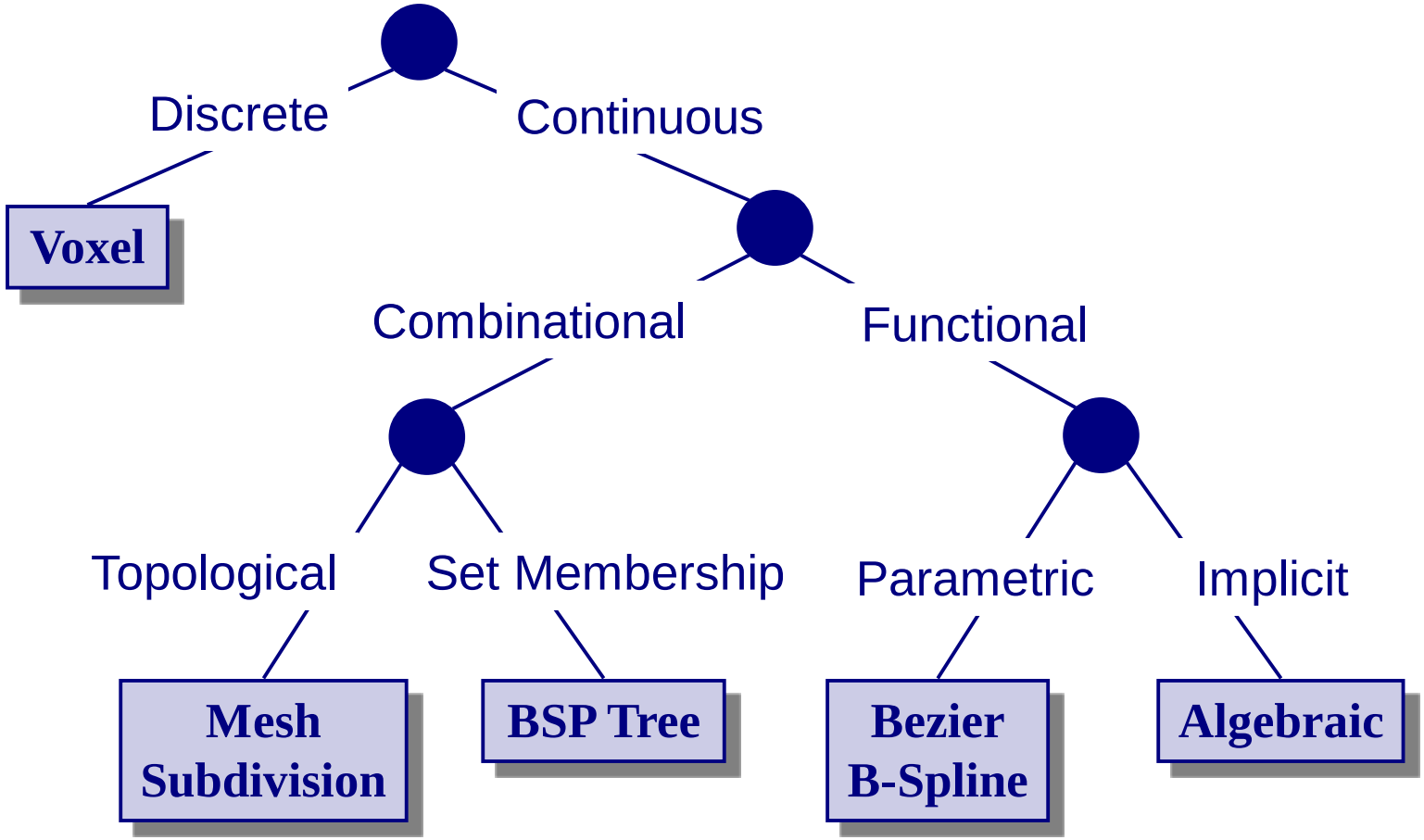


Constructing a Torus using Rotational Sweep



Summary

- Taxonomy of 3D Object Representations

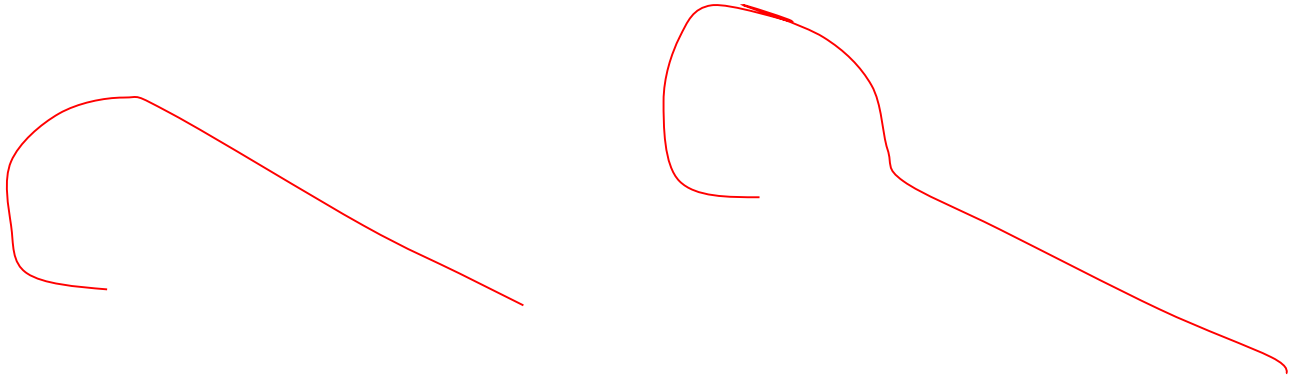


Contents

- Translation
- Scaling
- Rotation
- Rotations with Quaternions
- Other Transformations
- Coordinate Transformations

Curve representation

- Many techniques are available for drawing & Design the curves (Pen , pencil, brush etc)
- No single tool is sufficient for all tasks
- Two dimensional curve generation techniques are as “ A curve may be represented as a collection of points” provided points are properly spaced , connection of points by short straight line segments”



Three dimension representation

- *G*raphics scenes can contain many different kinds of objects: trees , flowers,clouds, rocks, water, bricks, wood paneling, rubber, paper, marble, steel,glass, plastic, and

- cloth, just to mention a few. So it is probably not too surprising that there is no one method that we can use to describe objects that will include all characteristics of these different materials. And to produce realistic displays of scenes, we need to use representations that accurately model object characteristics.
- Polygon and quadric surfaces ,polyhedrons and ellipsoids; spline surfaces are useful for designing air craft wings, gears, and other engineering structures with curved surfaces; procedural methods, such as fractal constructions and particle systems, allow us to give accurate representations for clouds, clumps of grass, and other natural objects;

- Representation schemes for solid objects are often divided into two broad categories, although not all representations fall neatly into one or the other of these two categories. Boundary representations (B-reps) describe a three-dimen-
- sional object as a set of surfaces that separate the object interior from the environment. examples of boundary representations are polygon and spline patches.
- Space-partitioning representations are used to describe interior properties, by partitioning the spatial region containing an object into a set of small, nonoverlapping, contiguous solids (usually **cubes). A** common space partitioning description for a three-dimensional object is an **octree** representation. we consider the features of the various representation schemes and how they are used in applications.

Non parametric Curves

- Mathematically parametric or non parametric form is used to represent the curve. Nonparametric form is $y = f(x)$ & eqn of line $y=m x +C$
- Implicit form is $f(x , y) =0$ and second degree eqn $ax^2 + 2b xy + cy^2 + 2dx + 2ey +f=0$ if $a=b=c=0$ then eqn $dx+ey+f=0$ (line)
- $B=d=e=0$ then ellipse or hyperbola
- If $a=b=e=0$ then parabola
- $A=c=1$ and $b=d=e=0$ then circle

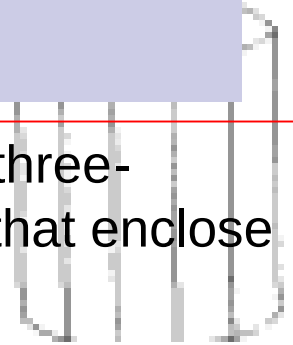
Parametric curves

- Point x=x(t) & y= y(t) then p(t)=[x(t), y(t)] dy / dx is slope
- Line p(t)=p1+(p2-p1)t where o<= t<=1
- X(t)=x1+(x2-x1)t & Y(t)=y1+(y2-y1)t where 0<=t<=1 and dy/dx is slope of line
- Circle $X^2 + y^2 = r^2$ then $Y=+\text{sqrt}(r^2-x^2)$ where 0<=x<=r in first quadrant then parametric form is
- X=r cos(θ) & y=r sin(θ) where 0<=θ<=2 pi
- Then p(θ)=p[x,y]=[r cos(θ), r sin(θ)]
- X_{i+1}=r cos($\theta_l+\delta\theta$) Y_{i+1}=r Sin($\theta_l+\delta\theta$)

- $X_i = r\cos(\theta_{l)}$ $Y_i = r\,\mathrm{Sin}(\theta_l)$ then
- $X_{i+1} = X_i \cos(\delta\theta) - Y_i \mathrm{Sin}(\delta\theta)$
- $Y_{i+1} = X_i \mathrm{Sin}(\delta\theta) + Y_i \mathrm{Cos}(\delta\theta)$
- Prametric eqn of ellipse

- $X^2/a^2 + y^2/b^2 = 1$
- $X = a\cos(\theta)$
- $Y = b\,\mathrm{Sin}(\theta)$
- $X_{i+1} = a\cos(\theta_l+\delta\theta)$ $Y_{i+1} = b\,\mathrm{Sin}(\theta_l+\delta\theta)$
- $X_{i+1} = X_i \cos(\delta\theta) - (a/b)Y_i \mathrm{Sin}(\delta\theta)$
- $Y_{i+1} = (b/a)X_i \mathrm{Sin}(\delta\theta) + Y_i \mathrm{Cos}(\delta\theta)$
- Parabola $y^2 = 4ax$ then $x = \mathrm{Tan}^2(\theta)$ & $y = +\text{-}2\mathrm{sqrt}(a\,\mathrm{Tan}(\theta))$ then $x = a(\theta)^{2}$ & $Y = 2a(\theta)$
- $X_{i+1} = X_i + Y_i(\delta\theta) + a(\delta\theta)^2$
- $Y_{i+1} = Y_i + 2a(\delta\theta)$ And $\theta_{min} = y_{min}/(2a)$ and $\theta_{max} = y_{max}/(2a)$

Polygon Surface

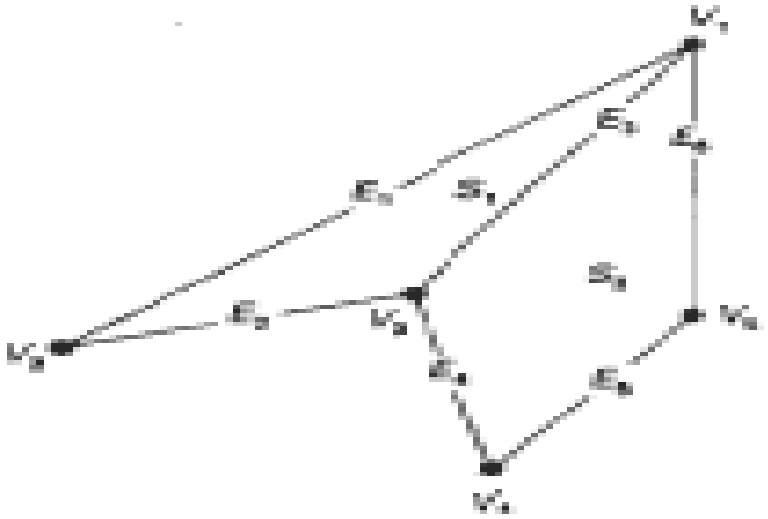


- The most commonly used boundary representation for a three-dimensional graphics object is a set of surface polygons that enclose the object interior.
- Many graphics systems store all object descriptions as sets of surface polygons. since all surfaces are described with linear equations. For this reason, polygon descriptions are often referred to as "standard graphics objects." In some cases, **A** polygon representation for a polyhedron precisely defines the surface features of the object. **But** for other objects, surfaces are tiled) to produce the polygon-mesh approximation. In Fig. the surface of a cylinder is represented as a polygon mesh.

Polygon Table

- We specify a polygon surface with a set of vertex coordinates and associated attribute parameters. **As**

- information for each polygon is input, the data are placed
- into tables that are to be used in the subsequent processing, display, and manipulation of the objects in a scene.
- Polygon data tables can be organized into two groups: geometric tables and attribute tables. Geometric data tables contain vertex coordinates and parameters to identify the spatial orientation of the polygon surfaces. Attribute information for an object includes parameters specifying the degree of transparency of the object and its surface reflectivity and texture characteristics.



VERTEX TABLE	
V_1 :	x_1, y_1, z_1
V_2 :	x_2, y_2, z_2
V_3 :	x_3, y_3, z_3
V_4 :	x_4, y_4, z_4
V_5 :	x_5, y_5, z_5

EDGE TABLE	
E_1 :	V_1, V_2
E_2 :	V_2, V_3
E_3 :	V_3, V_1
E_4 :	V_3, V_4
E_5 :	V_4, V_5
E_6 :	V_5, V_1

POLYGON-SURFACE TABLE	
S_1 :	E_1, E_2, E_3
S_2 :	E_3, E_4, E_5, E_6

E_1 : V_1, V_2, S_1
E_2 : V_2, V_3, S_1
E_3 : V_3, V_1, S_1, S_2
E_4 : V_3, V_4, S_2
E_5 : V_4, V_5, S_2
E_6 : V_5, V_1, S_2

Plane eqn

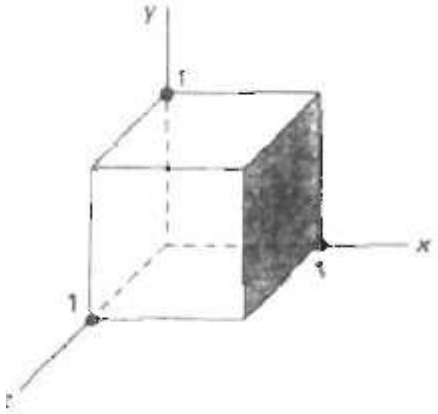
- To produce a display of a three-dimensional object, we must process the input data representation for the object through several
- procedures.
- These processing steps include transformation of the modeling and world-coordinate descriptions to viewing coordinates, then to device coordinates; identification of visible surfaces; we need information about the spatial orientation of the individual surface component object

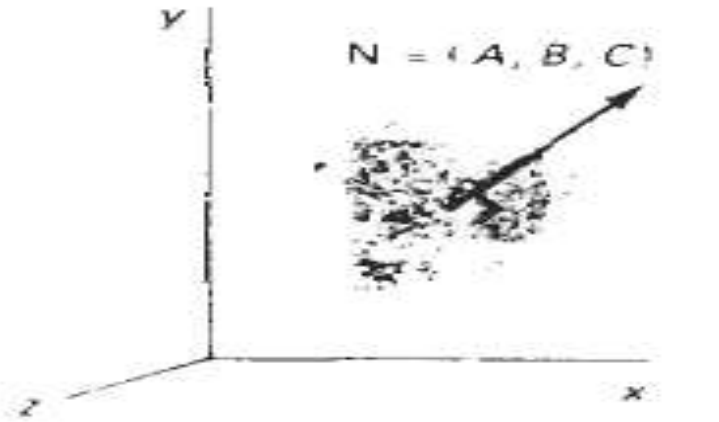
$$Ax + By + Cz + D = 0$$

$$(A/D)x_i + (B/D)y_i + (C/D)z_i = -1, \qquad i = 1, 2, 3$$

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \qquad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix}$$

$$C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \qquad D = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$





$$A = y_1(z_2 - z_3) + y_2(z_3 - z_1) + y_3(z_1 - z_2)$$

$$B = z_1(x_2 - x_3) + z_2(x_3 - x_1) + z_3(x_1 - x_2)$$

$$C = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$$

$$D = -x_1(y_2z_3 - y_3z_2) - x_2(y_3z_1 - y_1z_3) - x_3(y_1z_2 - y_2z_1)$$

Polygon Mess

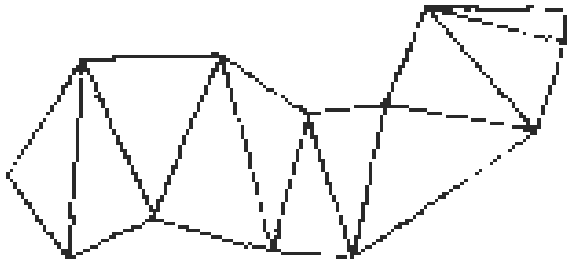


Figure 10-6
A triangle strip formed with 11 triangles connecting 13 vertices.

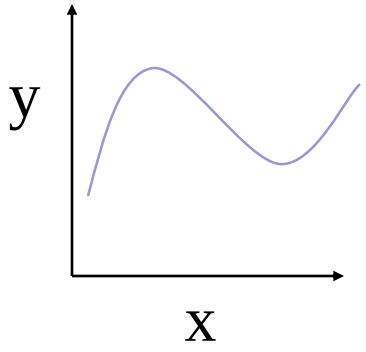
Figure 10-7
A quadrilateral mesh containing 12 quadrilaterals constructed from a 5 by 4 input vertex array.

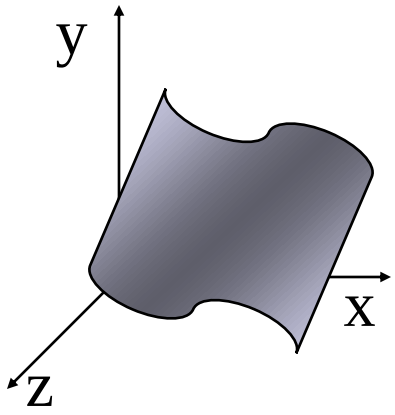
Explicit Representation

- Most familiar form of curve in 2D

  $y=f(x)$

- Cannot represent all curves
  - Vertical lines
  - Circles
- Extension to 3D
  - $y=f(x)$, $z=g(x)$
  - The form $z = f(x,y)$ defines a surface





Implicit Representation

- Two dimensional curve(s)

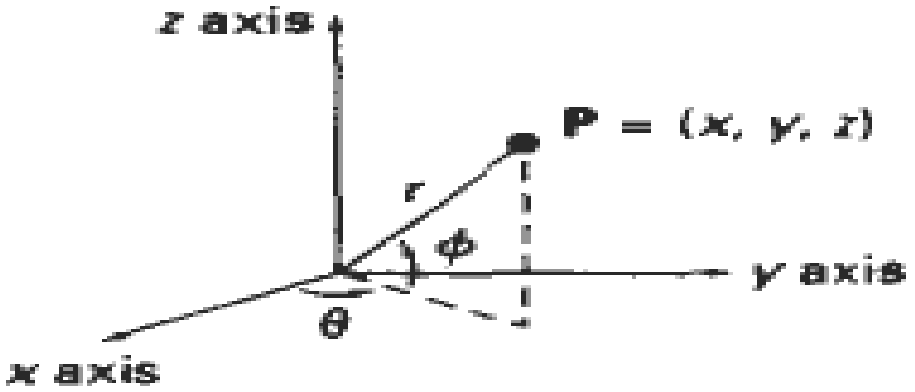
  $g(x,y)=0$

- Much more robust
  - All lines $ax+by+c=0$
  - Circles $x^2+y^2-r^2=0$
- Three dimensions $g(x,y,z)=0$ defines a surface
  - Intersect two surface to get a curve
- In general, we cannot solve for points that satisfy

QUADRIC SURFACES

- A frequently used class of objects are the quadric surfaces, which are described with second-degree equations (quadratics). They include spheres, ellipsoids, torus, paraboloids and hyperboloids.
- More complex objects can be constructed by these objects.

Sphere



- In Cartesian coordinates, a sph
  coordinate origin is defined as
  equation

$$x^2 + y^2 + z^2 = r^2$$

$$x = r\cos\phi\cos\theta, \qquad -\pi/2 \le \phi \le \pi/2$$

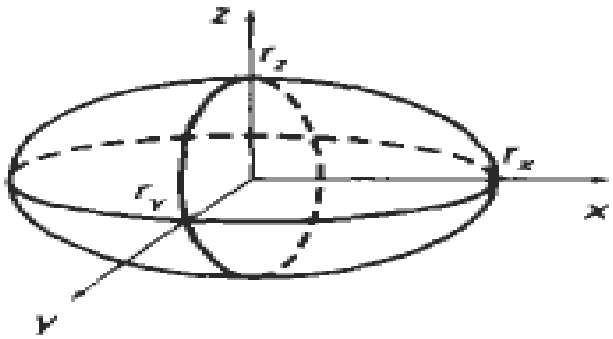
$$y = r\cos\phi\sin\theta, \qquad -\pi \le \theta \le \pi$$

$$z = r\sin\phi$$

Ellipsoid

- An ellipsoidal surface can be described as an extension of a spherical surface,where the radii in three mutually perpendicular directions can have different values. The Cartesian representation for points over the surface of an ellipsoid centered on the origin is

$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1$$



ellipsoid

- And a parametric representation for the ellipsoid in terms of the latitude angle ϕ.and the longitude angle

- θ

$$x = r_x \cos\phi \cos\theta, \qquad -\pi/2 \le \phi \le \pi/2$$

$$y = r_y \cos\phi \sin\theta, \qquad -\pi \le \theta \le \pi$$

$$z = r_z \sin\phi$$

●Torus

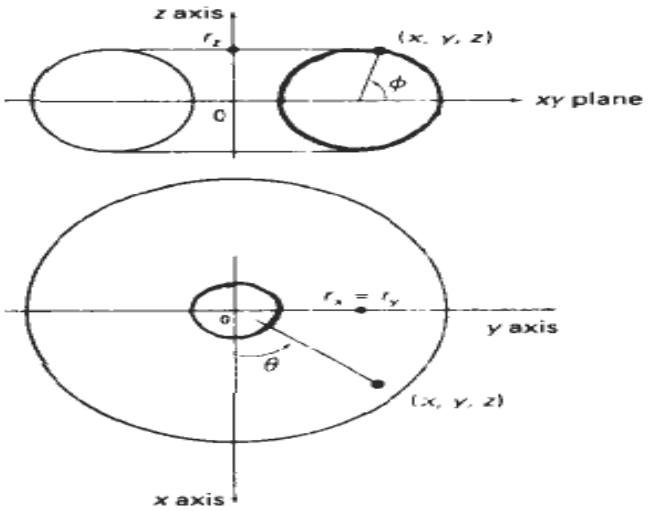


Figure 10-11
A torus with a circular cross section centered on the coordinate origin.

- **A** torus is a doughnut-shaped object, as shown in Fig. It can be generated by rotating a circle or other conic about a specified axis. The Cartesian representation for point over surface of torus can be as

$$\left[r - \sqrt{\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2}\right]^2 + \left(\frac{z}{r_z}\right)^2 = 1$$

Parametric eqn of torus

$$x = r_x(r + \cos\phi)\cos\theta, \qquad -\pi \le \phi \le \pi$$

$$y = r_y(r + \cos\phi)\sin\theta, \qquad -\pi \le \theta \le \pi$$

$$z = r_z \sin\phi$$

Curves and Surfaces

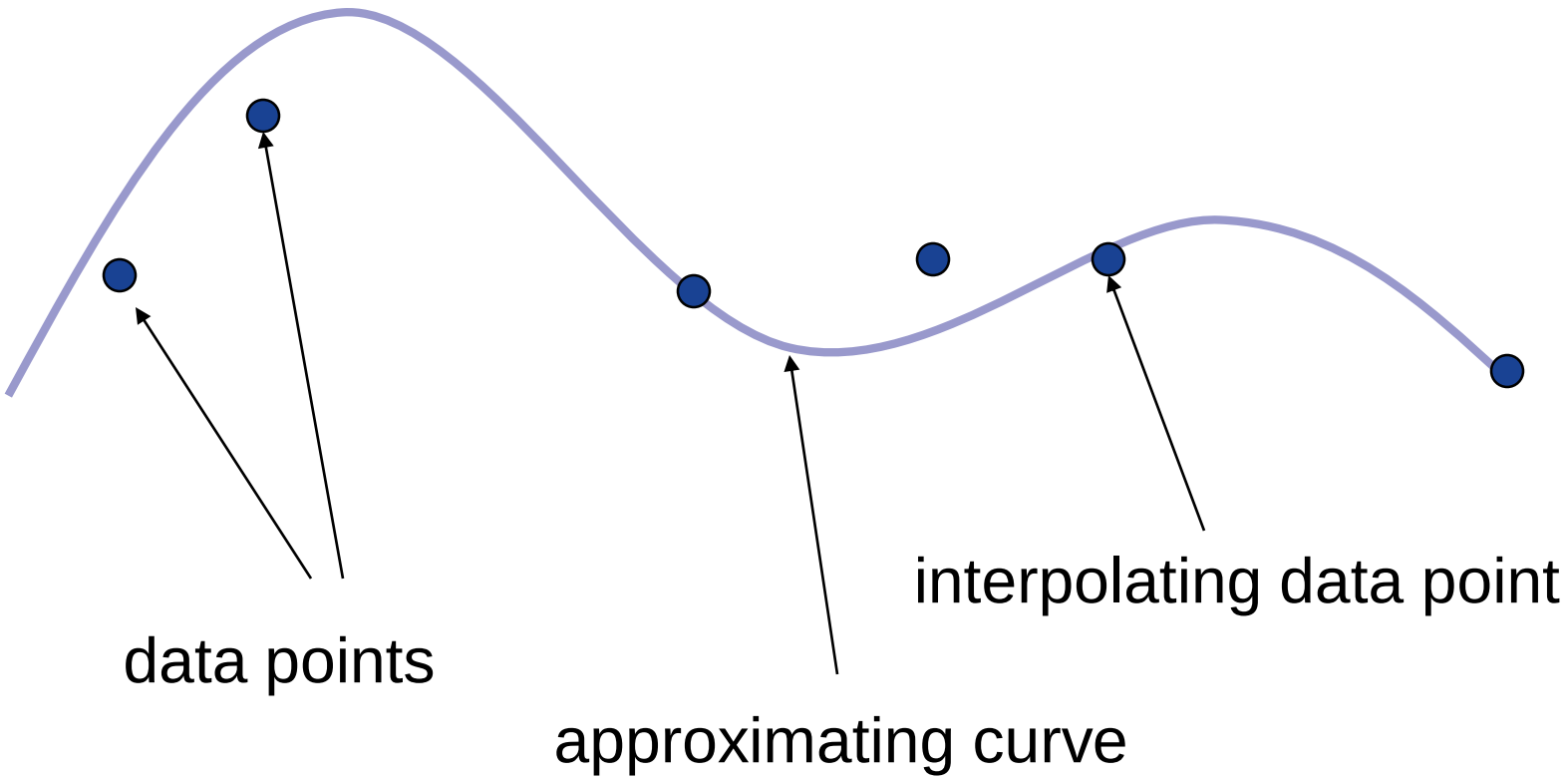
Objectives

- Introduce types of curves and surfaces
  - Explicit
  - Implicit
  - Parametric
  - Strengths and weaknesses
- Discuss Modeling and Approximations
  - Conditions
  - Stability

Escaping Flatland

- Until now we have worked with flat entities such as lines and flat polygons
  - Fit well with graphics hardware
  - Mathematically simple
- But the world is not composed of flat entities
  - Need curves and curved surfaces
  - May only have need at the application level
  - Implementation can render them approximately with flat primitives

Modeling with Curves



Algebraic Surface

$$\sum_i \sum_j \sum_k x^i y^j z^k = 0$$

- Quadric surface $2 \geq i+j+k$
- At most 10 terms
- Can solve intersection with a ray by reducing problem to solving quadratic equation

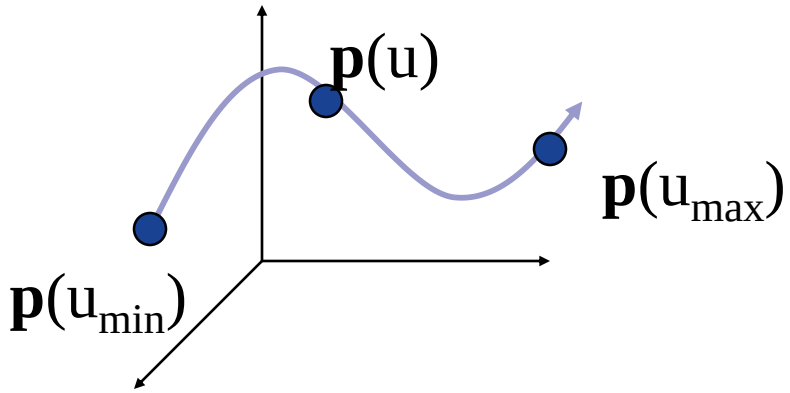
Parametric Curves

- Separate equation for each spatial variable

$$x=x(u)$$
$$y=y(u)$$
$$z=z(u)$$

$$\mathbf{p}(u)=[x(u),\ y(u),\ z(u)]^T$$

- For $u_{max} \geq u \geq u_{min}$ we trace out a curve in two or three dimensions



Selecting Functions

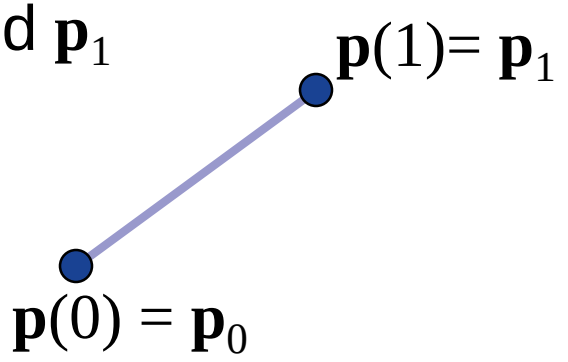
- Usually we can select “good” functions
  - not unique for a given spatial curve
  - Approximate or interpolate known data
  - Want functions which are easy to evaluate
  - Want functions which are easy to differentiate
    - Computation of normals
    - Connecting pieces (segments)
  - Want functions which are smooth

Parametric Lines

We can normalize u to be over the interval (0,1)

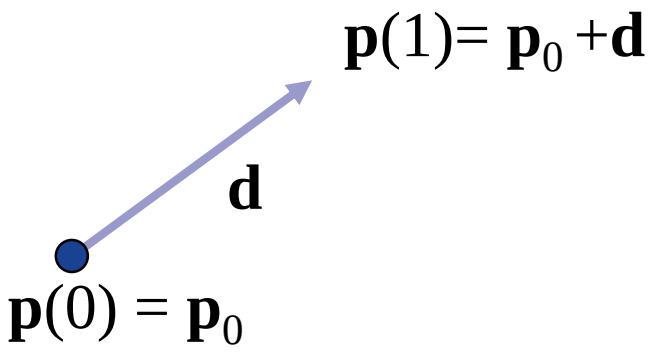
Line connecting two points $\mathbf{p}_0$ and $\mathbf{p}_1$

$$\mathbf{p}(u)=(1-u)\mathbf{p}_0+u\mathbf{p}_1$$



Ray from $\mathbf{p}_0$ in the direction $\mathbf{d}$

$$\mathbf{p}(u)=\mathbf{p}_0+u\mathbf{d}$$

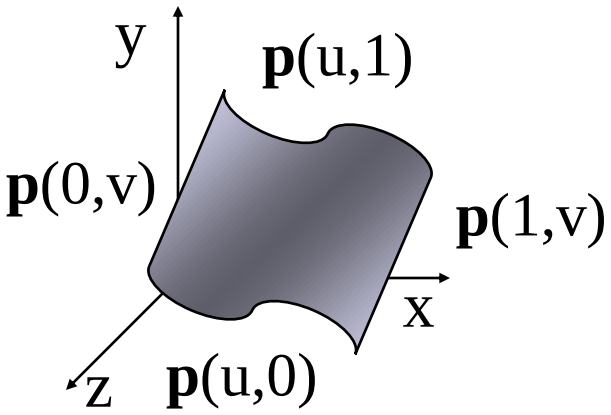


Parametric Surfaces

- Surfaces require 2 parameters

$$x=x(u,v)$$
$$y=y(u,v)$$
$$z=z(u,v)$$
$$\mathbf{p}(u,v) = [x(u,v), y(u,v), z(u,v)]^T$$

- Want same properties as curves:
  - Smoothness
  - Differentiability
  - Ease of evaluation



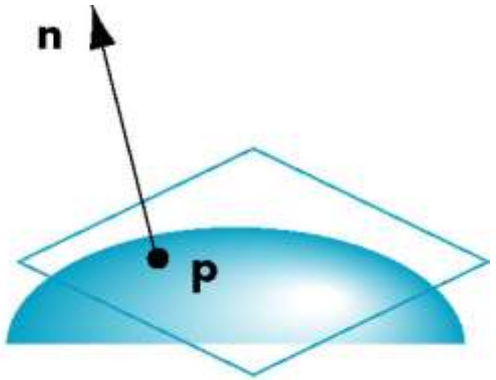
Normals

We can differentiate with respect to u and v to obtain the normal at any point **p**

$$\frac{\partial \mathbf{p}(u,v)}{\partial u} = \begin{bmatrix} \partial \mathrm{x}(u,v)/\partial u \\ \partial \mathrm{y}(u,v)/\partial u \\ \partial \mathrm{z}(u,v)/\partial u \end{bmatrix}$$

$$\frac{\partial \mathbf{p}(u,v)}{\partial v} = \begin{bmatrix} \partial \mathrm{x}(u,v)/\partial v \\ \partial \mathrm{y}(u,v)/\partial v \\ \partial \mathrm{z}(u,v)/\partial v \end{bmatrix}$$

$$\mathbf{n} = \frac{\partial \mathbf{p}(u,v)}{\partial u} \times \frac{\partial \mathbf{p}(u,v)}{\partial v}$$



Parametric Planes

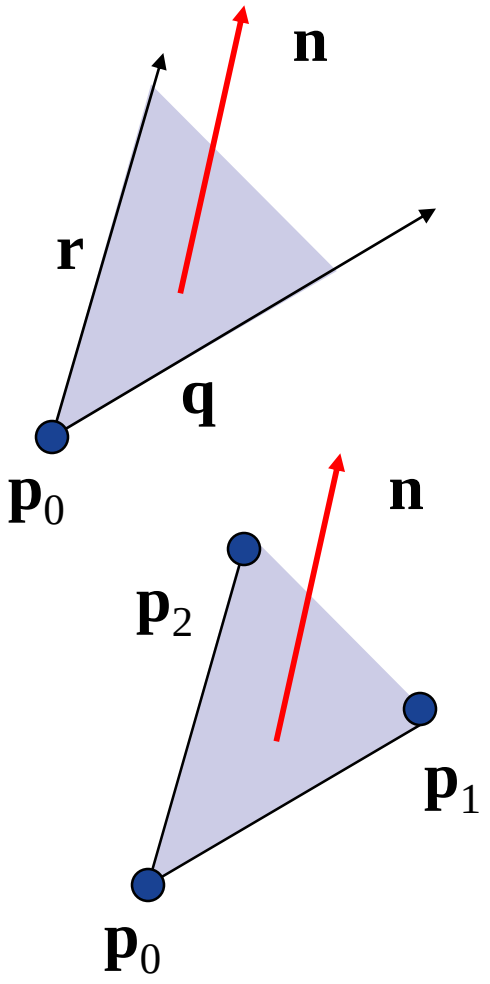
point-vector form

$$\mathbf{p}(u,v)=\mathbf{p}_0+u\mathbf{q}+v\mathbf{r}$$

$$\mathbf{n} = \mathbf{q} \times \mathbf{r}$$

three-point form

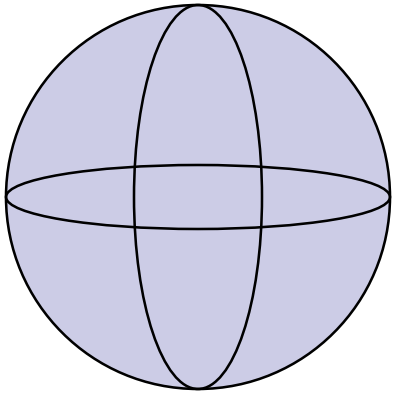
$$\mathbf{q} = \mathbf{p}_1 - \mathbf{p}_0$$
$$\mathbf{r} = \mathbf{p}_2 - \mathbf{p}_0$$



Parametric Sphere

$$x(u,v) = r \cos\theta \sin\phi$$
$$y(u,v) = r \sin\theta \sin\phi$$
$$z(u,v) = r \cos\phi$$

$$360 \geq \theta \geq 0$$
$$180 \geq \phi \geq 0$$



θ constant: circles of constant longitude

ϕ constant: circles of constant latitude

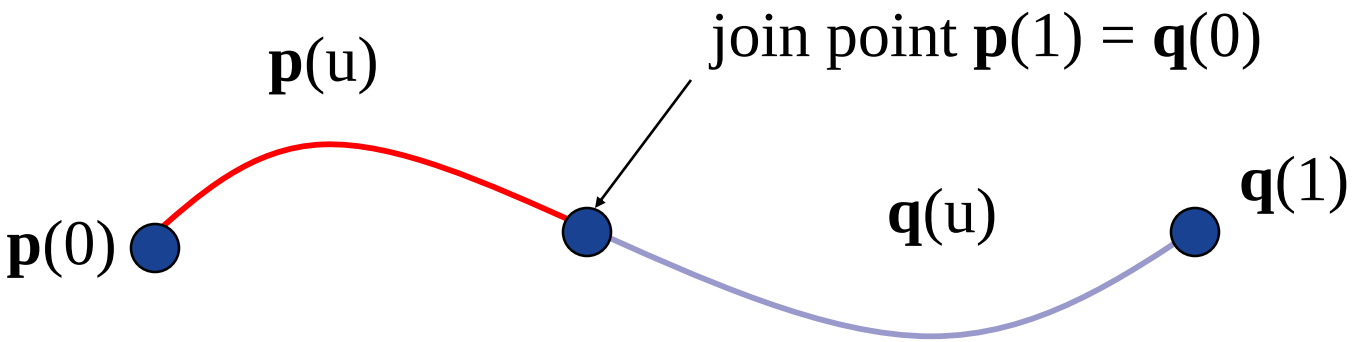
differentiate to show $\mathbf{n} = \mathbf{p}$

Curve Segments

- After normalizing u, each curve is written

  $\mathbf{p}(u)=[x(u), y(u), z(u)]^T, \quad 1 \geq u \geq 0$

- In classical numerical methods, we design a single global curve
- In computer graphics and CAD, it is better to design small connected curve *segments*



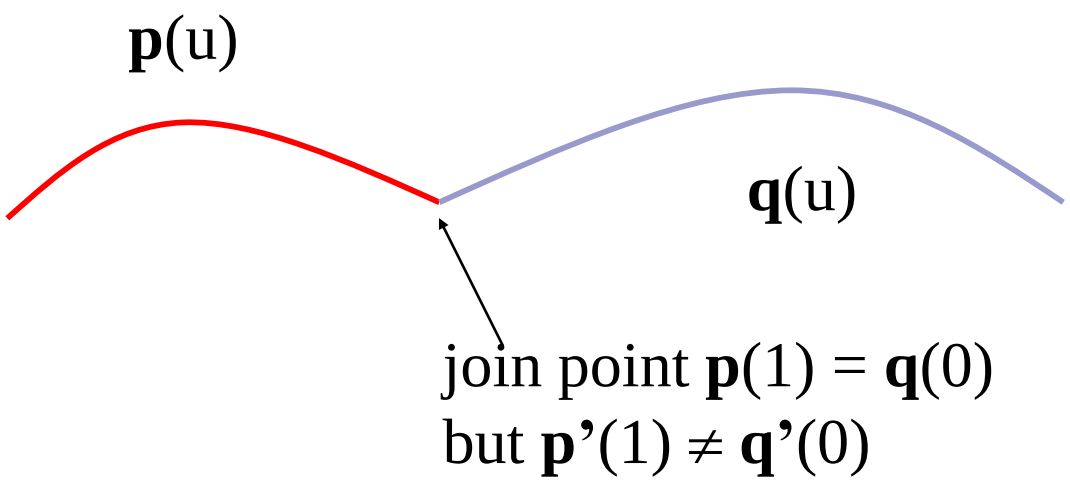
Parametric Polynomial Curves

$$x(u) = \sum_{i=0}^{N} c_{xi}\, u^i \qquad y(u) = \sum_{j=0}^{M} c_{yj}\, u^j \qquad z(u) = \sum_{k=0}^{L} c_{zk}\, u^k$$

- If N=M=K, we need to determine 3(N+1) coefficients
- Equivalently we need 3(N+1) independent conditions
- Noting that the curves for x, y and z are independent, we can define each independently in an identical manner
- We will use the form $p(u) = \sum_{k=0}^{L} c_k\, u^k$ where p can be any of x, y, z

Why Polynomials

- Easy to evaluate
- Continuous and differentiable everywhere
  - Must worry about continuity at join points including continuity of derivatives



Cubic Parametric Polynomials

- N=M=L=3, gives balance between ease of evaluation and flexibility in design

$$\mathrm{p}(u) = \sum_{k=0}^{3} c_k u^k$$

- Four coefficients to determine for each of x, y and z
- Seek four independent conditions for various values of u resulting in 4 equations in 4 unknowns for each of x, y and z
  - Conditions are a mixture of continuity requirements at the join points and conditions for fitting the data

Cubic Polynomial Surfaces

$$\mathbf{p}(u,v)=[x(u,v),\ y(u,v),\ z(u,v)]^T$$

where

$$p(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} c_{ij}\, u^i\, v^j$$

p is any of x, y or z

Need 48 coefficients (3 independent sets of 16) to determine a surface patch

Transformation in 3D

- Transformation Matrix

$$\begin{bmatrix} A & D & G & J \\ B & E & H & K \\ C & F & I & L \\ 0 & 0 & 0 & S \end{bmatrix} \Rightarrow \left[\begin{array}{ccc|c} & 3\times3 & & 3\times1 \\ & & & \\ \hline & 1\times3 & & 1\times1 \end{array}\right]$$

3×3 : Scaling, Reflection, Shearing, Rotation

3×1 : Translation

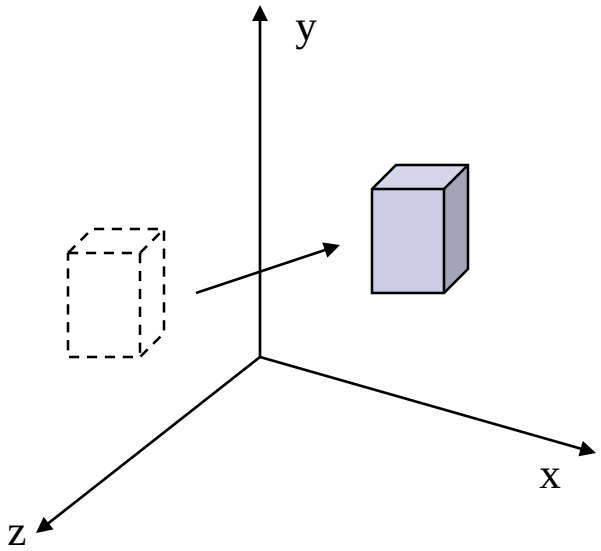
1×1 : Uniform global Scaling

1×3 : Homogeneous representation

3D Translation

- Translation of a Point

$$x' = x + t_x, \quad y' = y + t_y, \quad z' = z + t_z$$

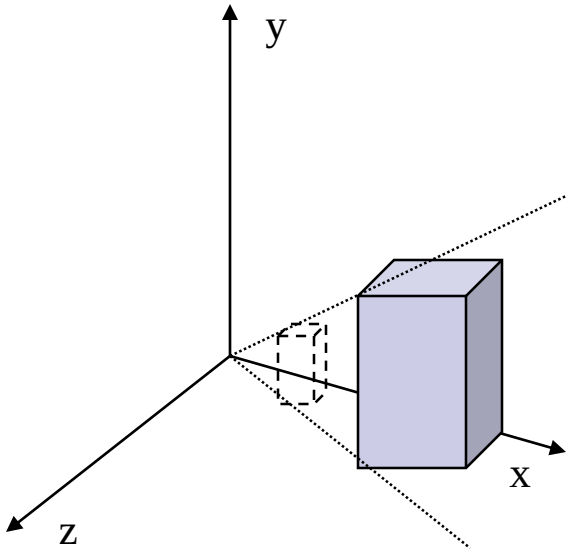


$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3D Scaling

- Uniform Scaling

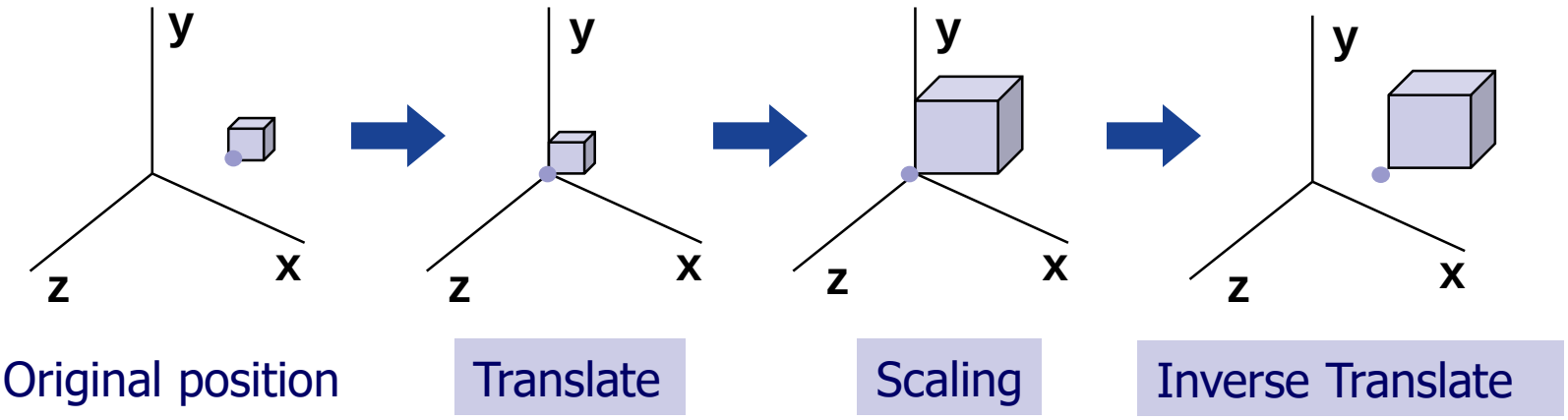
$$x' = x \cdot s_x, \quad y' = y \cdot s_y, \quad z' = z \cdot s_z$$



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Relative Scaling

- Scaling with a Selected Fixed Position



$$T(x_f, y_f, z_f) \cdot S(s_x, s_y, s_z) \cdot T(-x_f, -y_f, -z_f) = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_f \\ 0 & 1 & 0 & y_f \\ 0 & 0 & 1 & z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_f \\ 0 & 1 & 0 & -y_f \\ 0 & 0 & 1 & -z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3D Rotation

- Coordinate-Axes Rotations
  - X-axis rotation
  - Y-axis rotation
  - Z-axis rotation

- General 3D Rotations
  - Rotation about an axis that is parallel to one of the coordinate axes
  - Rotation about an arbitrary axis

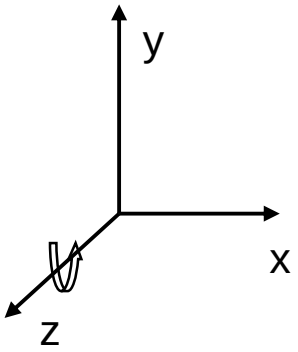
Coordinate-Axes Rotations

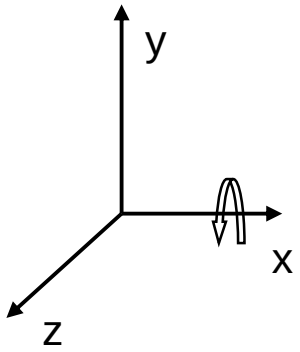
- Z-Axis Rotation
- X-Axis Rotation
- Y-Axis Rotation

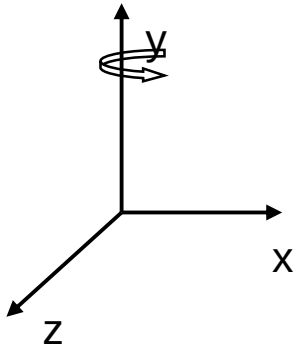
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

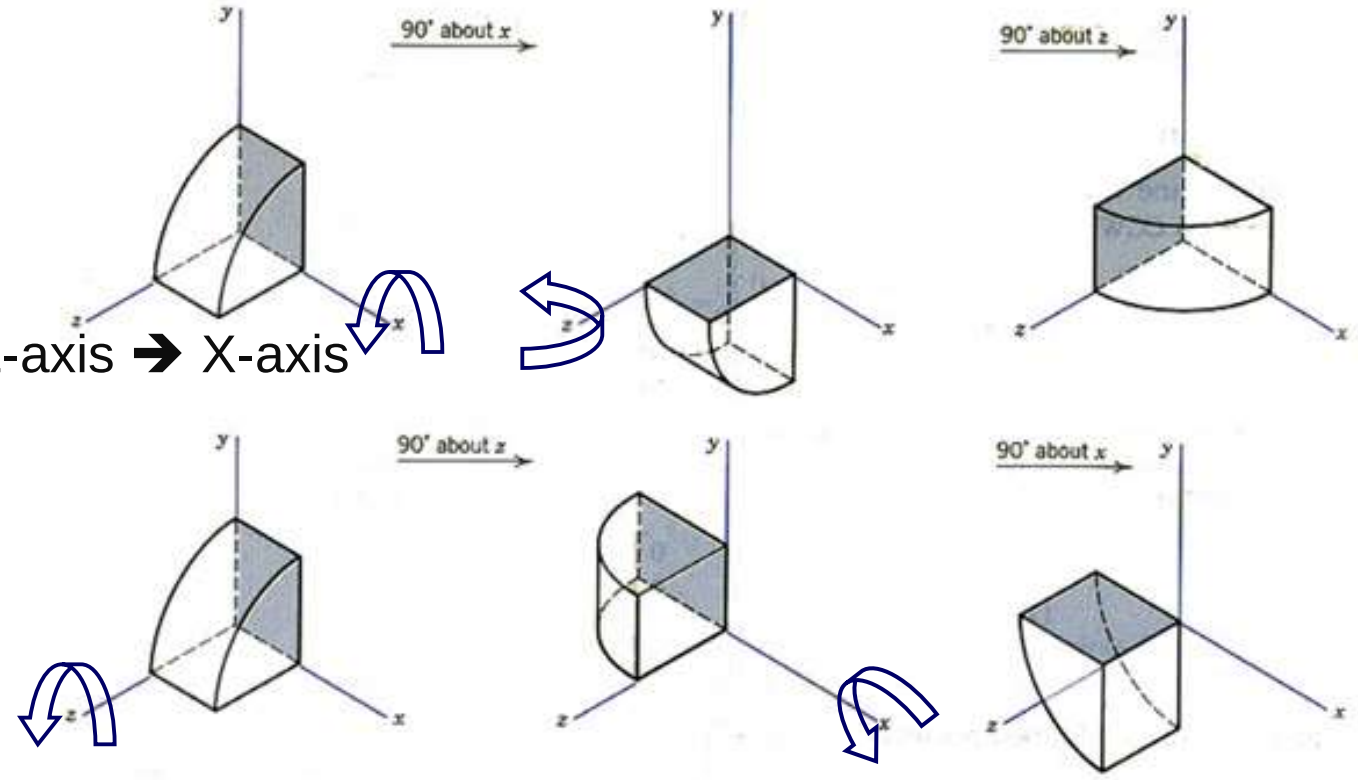






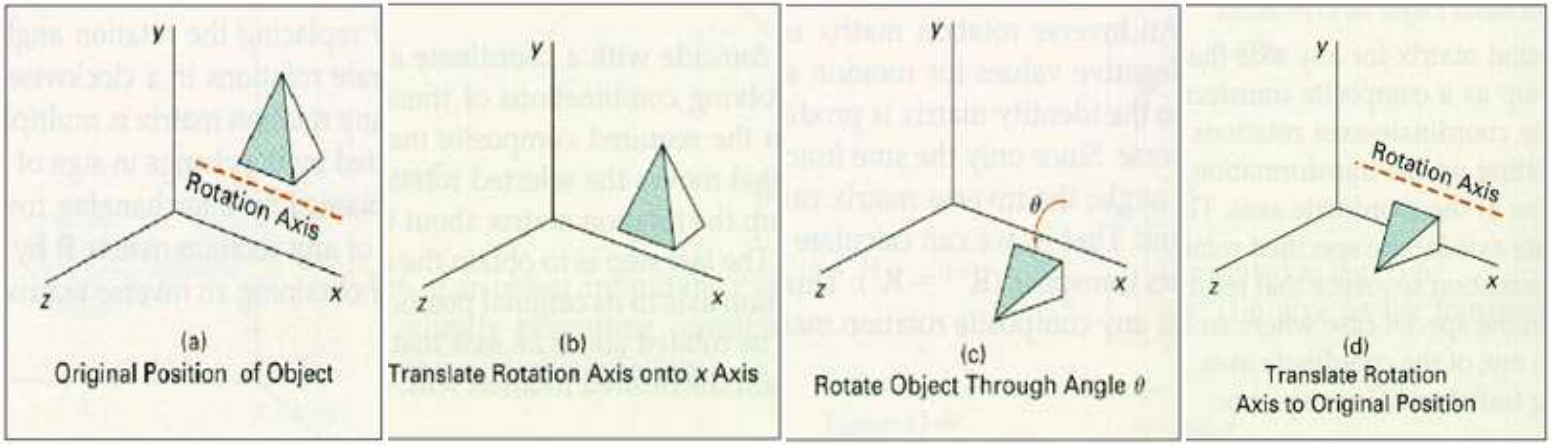
Order of Rotations

- Order of Rotation Affects Final Position
  - X-axis ➔ Z-axis
  - Z-axis ➔ X-axis



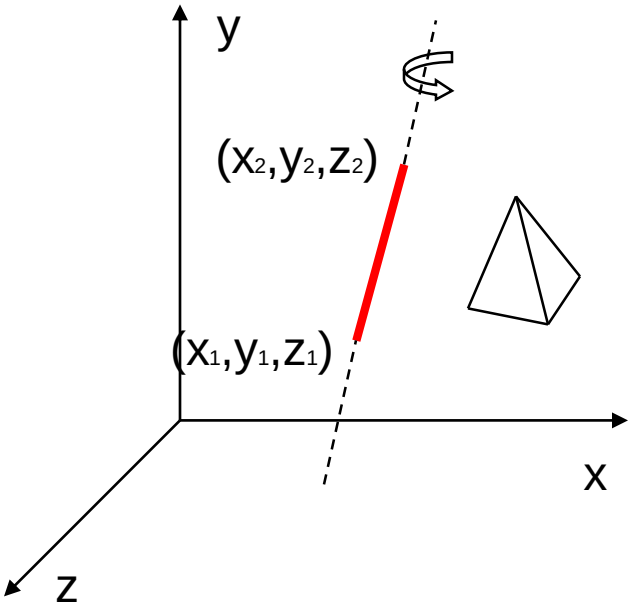
General 3D Rotations

- Rotation about an Axis that is Parallel to One of the Coordinate Axes
  - **Translate** the object so that the rotation axis coincides with the parallel coordinate axis
  - Perform the specified **rotation** about that axis
  - **Translate** the object so that the rotation axis is moved back to its original position



General 3D Rotations

- Rotation about an Arbitrary Axis



T

R

R^{-1}

T^{-1}

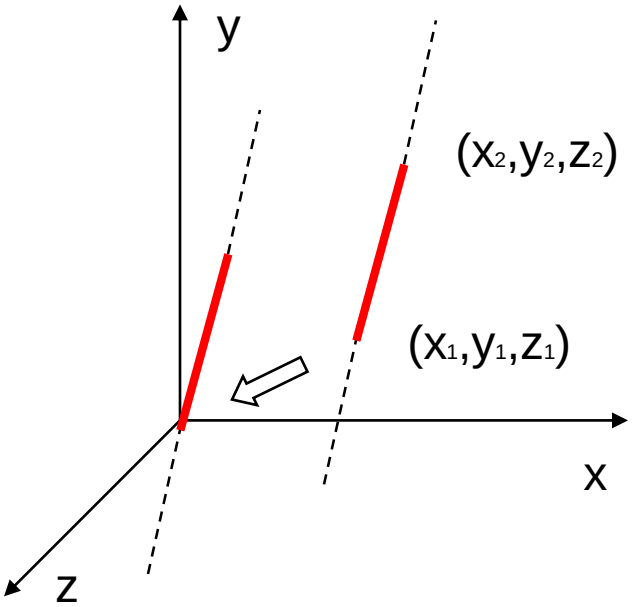
Basic Idea

1. **Translate (x1, y1, z1) to the origin**
2. **Rotate (x'2, y'2, z'2) on to the z axis**
3. **Rotate the object around the z-axis**
4. **Rotate the axis to the original orientation**
5. **Translate the rotation axis to the original position**

$$\mathbf{R}(\theta) = \mathbf{T}^{-1}\mathbf{R}_x^{-1}(\alpha)\mathbf{R}_y^{-1}(\beta)\mathbf{R}_z(\theta)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha)\mathbf{T}$$

General 3D Rotations

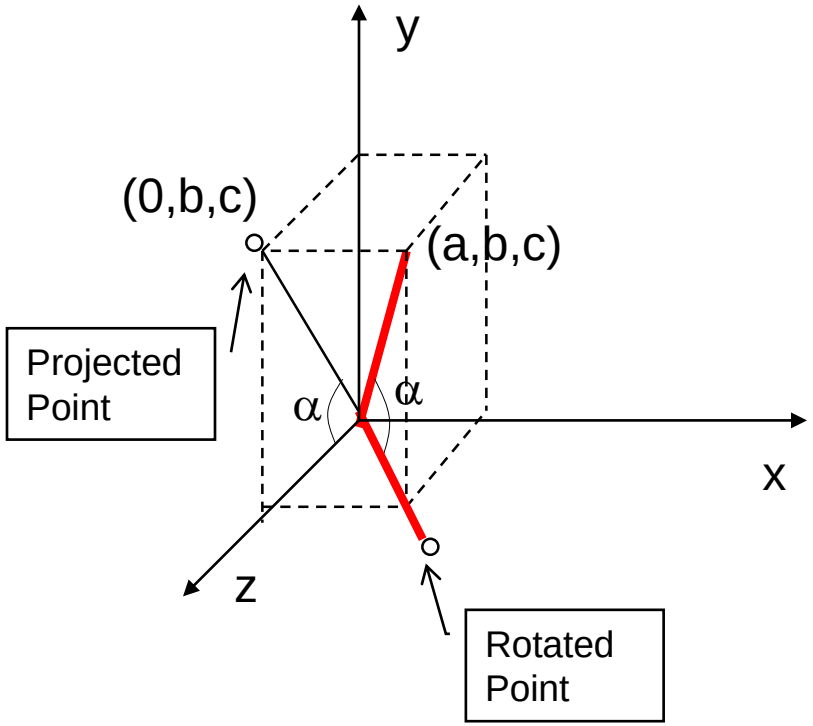
- Step 1. Translation



$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

General 3D Rotations

- Step 2. Establish $[\ T_R\ ]^{\alpha}_{x}$ *x* axis



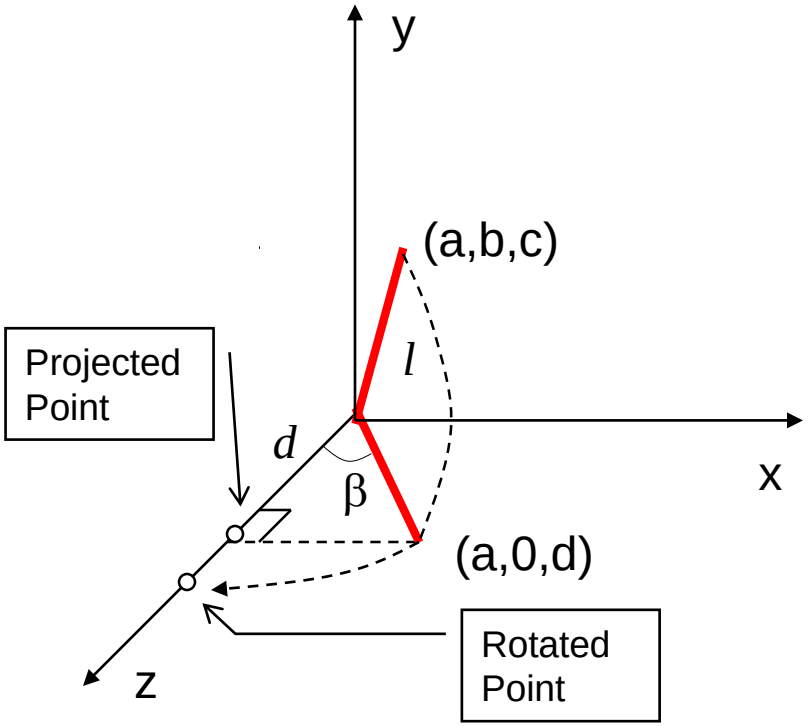
$$\sin\alpha = \frac{b}{\sqrt{b^2 + c^2}} = \frac{b}{d}$$

$$\cos\alpha = \frac{c}{\sqrt{b^2 + c^2}} = \frac{c}{d}$$

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & -b/d & 0 \\ 0 & b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Arbitrary Axis Rotation

- Step 3. Rotate about *y* axis by ϕ



$$\sin\beta = \frac{a}{l}, \qquad \cos\beta = \frac{d}{l}$$

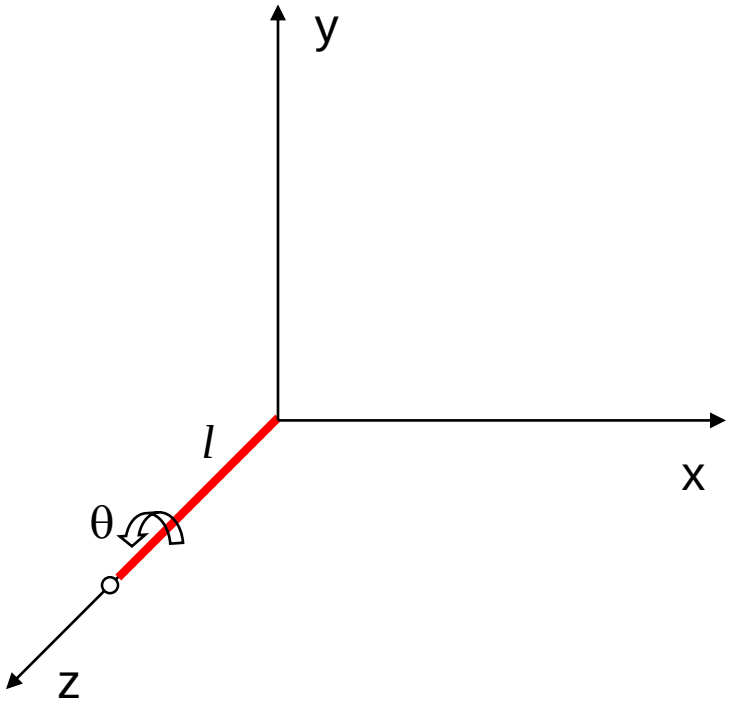
$$l^2 = a^2 + b^2 + c^2 = a^2 + d^2$$

$$d = \sqrt{b^2 + c^2}$$

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} d/l & 0 & -a/l & 0 \\ 0 & 1 & 0 & 0 \\ a/l & 0 & d/l & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Arbitrary Axis Rotation

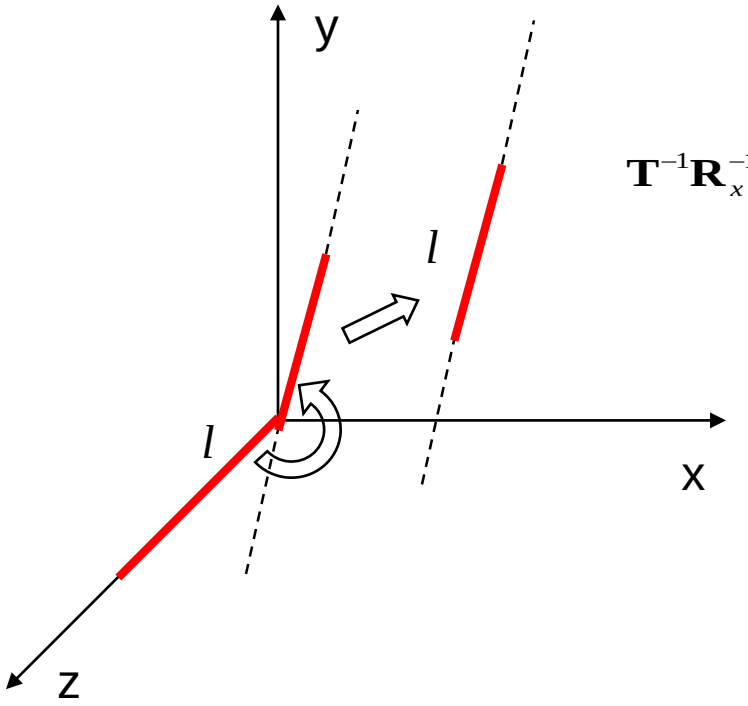
- Step 4. Rotate about *z* axis by the desired angle θ



$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Arbitrary Axis Rotation

- Step 5. Apply the reverse transformation to place the axis back in its initial position

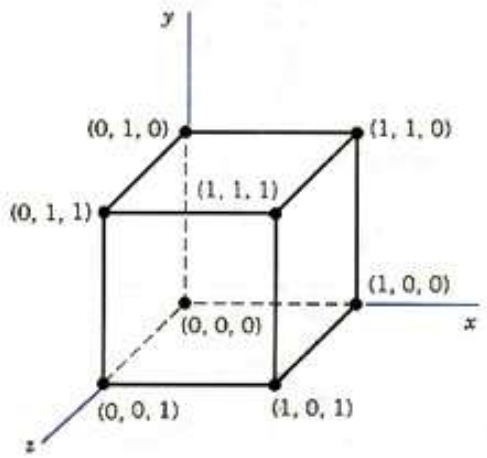


$$\mathbf{T}^{-1}\mathbf{R}_x^{-1}(\alpha)\mathbf{R}_y^{-1}(\beta)=\begin{bmatrix}1&0&0&-x_1\\0&1&0&-y_1\\0&0&1&-z_1\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&\cos\alpha&\sin\alpha&0\\0&-\sin\alpha&\cos\alpha&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}\cos\beta&0&\sin\beta&0\\0&1&0&0\\-\sin\beta&0&\cos\beta&0\\0&0&0&1\end{bmatrix}$$

$$\mathbf{R}(\theta)=\mathbf{T}^{-1}\mathbf{R}_x^{-1}(\alpha)\mathbf{R}_y^{-1}(\beta)\mathbf{R}_z(\theta)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha)\mathbf{T}$$

Example

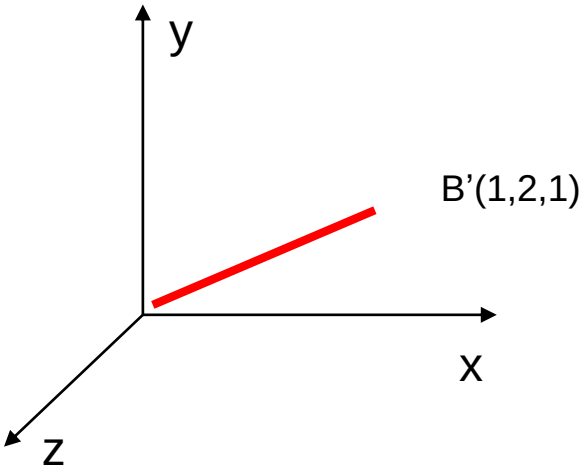
Find the new coordinates of a unit cube 90º-rotated about an axis defined by its endpoints A(2,1,0) and B(3,3,1).



A Unit Cube

Example

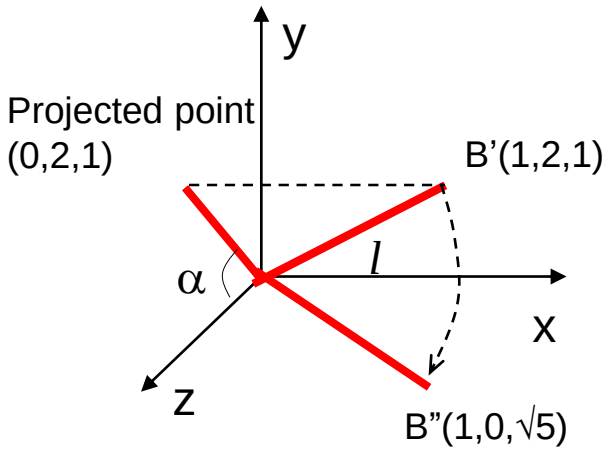
- Step1. Translate point A to the origin



$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -2 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Example

- Step 2. Rotate axis *A'B'* about the x axis by and angle α, until it lies on the *xz* plane.



$$\sin\alpha = \frac{2}{\sqrt{2^2+1^2}} = \frac{2}{\sqrt{5}} = \frac{2\sqrt{5}}{5}$$

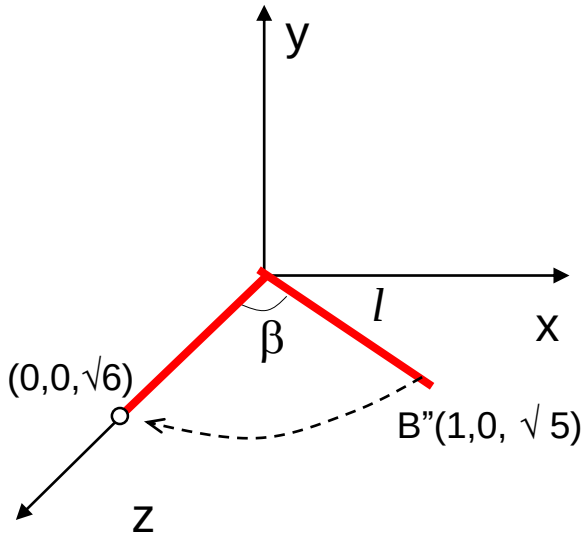
$$\cos\alpha = \frac{1}{\sqrt{5}} = \frac{\sqrt{5}}{5}$$

$$l = \sqrt{1^2+2^2+1^2} = \sqrt{6}$$

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{5}}{5} & -\frac{2\sqrt{5}}{5} & 0 \\ 0 & \frac{2\sqrt{5}}{5} & \frac{\sqrt{5}}{5} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Example

- Step 3. Rotate axis *A'B"* about the y axis by and angle ϕ, until it coincides with the z axis.



$$\sin\beta = \frac{1}{\sqrt{6}} = \frac{\sqrt{6}}{6}$$

$$\cos\beta = \frac{\sqrt{5}}{\sqrt{6}} = \frac{\sqrt{30}}{6}$$

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \frac{\sqrt{30}}{6} & 0 & -\frac{\sqrt{6}}{6} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{\sqrt{6}}{6} & 0 & \frac{\sqrt{30}}{6} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Example

- Step 4. Rotate the cube 90˚ about the z axis

$$\mathbf{R}_z(90^\circ) = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - Finally, the concatenated rotation matrix about the arbitrary axis AB becomes,

$$\mathbf{R}(\theta) = \mathbf{T}^{-1}\mathbf{R}_x^{-1}(\alpha)\mathbf{R}_y^{-1}(\beta)\mathbf{R}_z(90^\circ)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha)\mathbf{T}$$

Example

$$\mathbf{R}(\theta)=\begin{bmatrix}1&0&0&2\\0&1&0&1\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&\frac{\sqrt{5}}{5}&\frac{2\sqrt{5}}{5}&0\\0&-\frac{2\sqrt{5}}{5}&\frac{\sqrt{5}}{5}&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}\frac{\sqrt{30}}{6}&0&\frac{\sqrt{6}}{6}&0\\0&1&0&0\\-\frac{\sqrt{6}}{6}&0&\frac{\sqrt{30}}{6}&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}0&-1&0&0\\1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

$$\begin{bmatrix}\frac{\sqrt{30}}{6}&0&-\frac{\sqrt{6}}{6}&0\\0&1&0&0\\\frac{\sqrt{6}}{6}&0&\frac{\sqrt{30}}{6}&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&\frac{\sqrt{5}}{5}&-\frac{2\sqrt{5}}{5}&0\\0&\frac{2\sqrt{5}}{5}&\frac{\sqrt{5}}{5}&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&-2\\0&1&0&-1\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

$$=\begin{bmatrix}0.166&-0.075&0.983&1.742\\0.742&0.667&0.075&-1.151\\-0.650&0.741&0.167&0.560\\0&0&0&1\end{bmatrix}$$

Example

- Multiplying $\mathbf{R}(\theta)$ by the point matrix of the original cube

$$[\mathbf{P}'] = \mathbf{R}(\theta) \cdot [\mathbf{P}]$$

$$[\mathbf{P}'] = \begin{bmatrix} 0.166 & -0.075 & 0.983 & 1.742 \\ 0.742 & 0.667 & 0.075 & -1.151 \\ -0.650 & 0.741 & 0.167 & 0.560 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2.650 & 1.667 & 1.834 & 2.816 & 2.725 & 1.742 & 1.909 & 2.891 \\ -0.558 & -0.484 & 0.258 & 0.184 & -1.225 & -1.151 & -0.409 & -0.483 \\ 1.467 & 1.301 & 0.650 & 0.817 & 0.726 & 0.560 & -0.091 & 0.076 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Rotations with Quaternions

- Quaternion
  - Scalar part s + vector part $\mathbf{v} = (a, b, c)$
  - Real part + complex part (imaginary numbers i, j, k)

$$q = (s,\ \mathbf{v}) = s + ai + bj + ck$$

- Rotation about any axis
  - Set up a unit quaternion ($\mathbf{u}$: unit vector)

  - Represent any point position $\mathbf{P}$ in quaternion notation ($\mathbf{p} = (x, y, z)$)

$$s = \cos\frac{\theta}{2},\ \mathbf{v} = \mathbf{u}\sin\frac{\theta}{2}$$

$$\mathbf{P} = (0,\ \mathbf{p})$$

Rotations with Quaternions

- Carry out with the quaternion operation ($q^{-1}=(s, -\mathbf{v})$)

$$\mathbf{P}' = q\mathbf{P}q^{-1}$$

- Produce the new quaternion

$$\mathbf{P}' = (0,\ \mathbf{p}')$$

$$\mathbf{p}' = s^2\mathbf{p} + \mathbf{v}(\mathbf{p}\cdot\mathbf{v}) + 2s(\mathbf{v}\times\mathbf{p}) + \mathbf{v}\times(\mathbf{v}\times\mathbf{p})$$

- Obtain the rotation matrix by quaternion multiplication

$$\mathbf{M}_R(\theta) = \mathbf{R}_x^{-1}(\alpha)\mathbf{R}_y^{-1}(\beta)\mathbf{R}_z(\theta)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha)$$

$$= \begin{bmatrix} 1-2b^2-2c^2 & 2ab-2sc & 2ac+2sb \\ 2ab+2sc & 1-2a^2-2c^2 & 2bc-2sa \\ 2ac-2sb & 2bc+2sa & 1-2a^2-2b^2 \end{bmatrix}$$

- Include the translations:

$$\mathbf{R}(\theta) = \mathbf{T}^{-1}\mathbf{M}_R(\theta)\mathbf{T}$$

Example

- Rotation about z axis
  - Set the unit quaternion:
  - Substitute $a=b=0$, $c=\sin(\theta/2)$ into the matrix:

$$s = \cos\frac{\theta}{2},\ \mathbf{v} = (0,\ 0,\ 1)\sin\frac{\theta}{2}$$

$$\mathbf{M}_R(\theta) = \begin{bmatrix} 1-2\sin^2\frac{\theta}{2} & -2\cos\frac{\theta}{2}\sin\frac{\theta}{2} & 0 \\ 2\cos\frac{\theta}{2}\sin\frac{\theta}{2} & 1-2\sin^2\frac{\theta}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

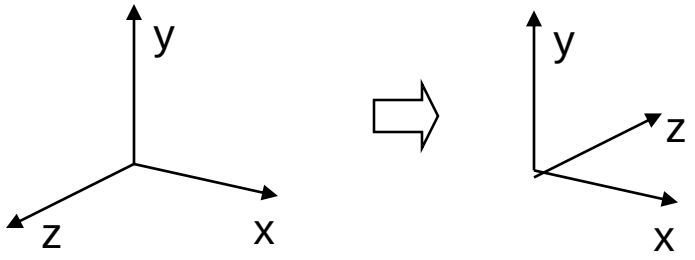
$$= \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$1-2\sin^2\frac{\theta}{2} = \cos\theta$$

$$2\cos\frac{\theta}{2}\sin\frac{\theta}{2} = \sin\theta$$

Other Transformations

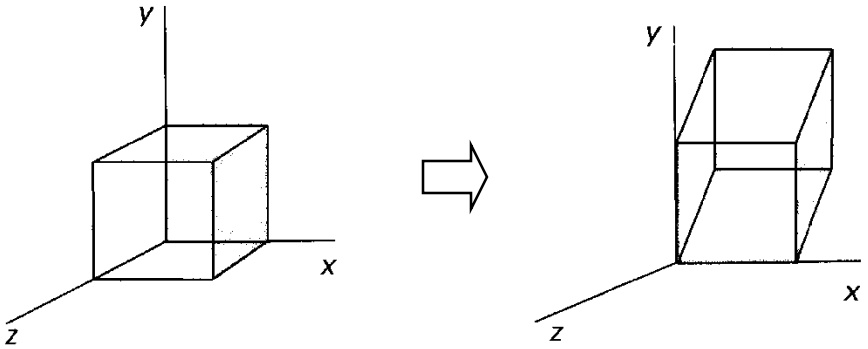
- Reflection Relative to the xy Plane



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

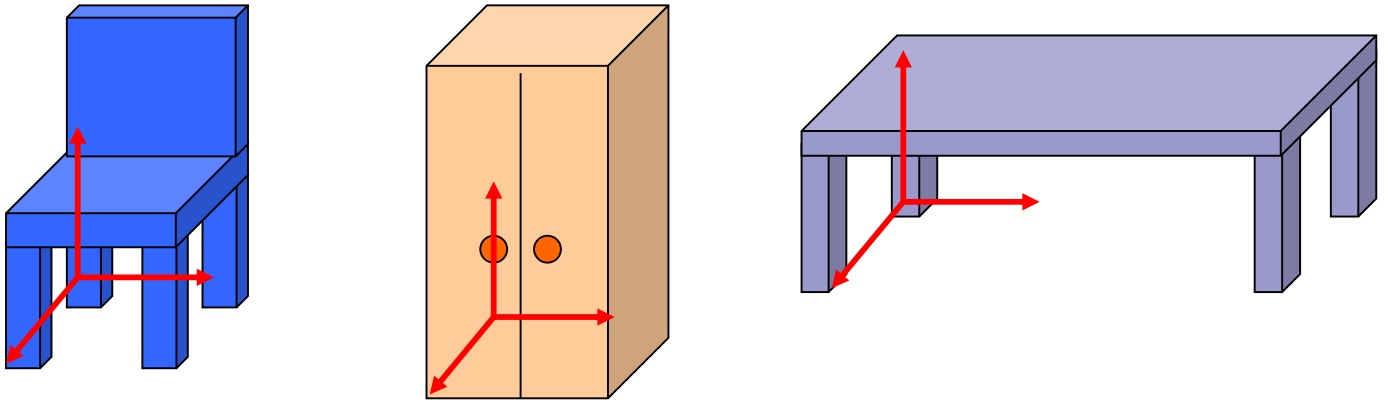
- Z-axis Shear

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Coordinate Transformations

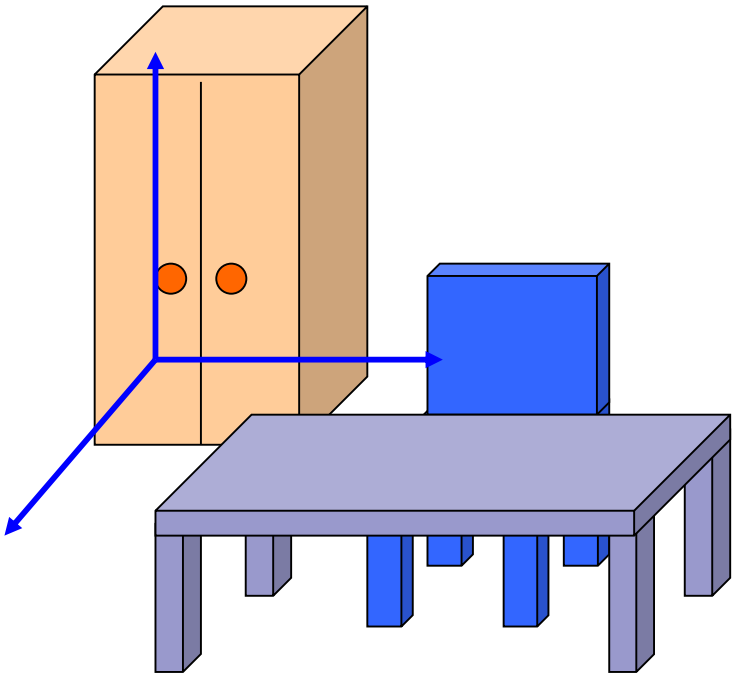
- Multiple Coordinate System
  - Local (modeling) coordinate system
  - World coordinate scene



Local Coordinate System

Coordinate Transformations

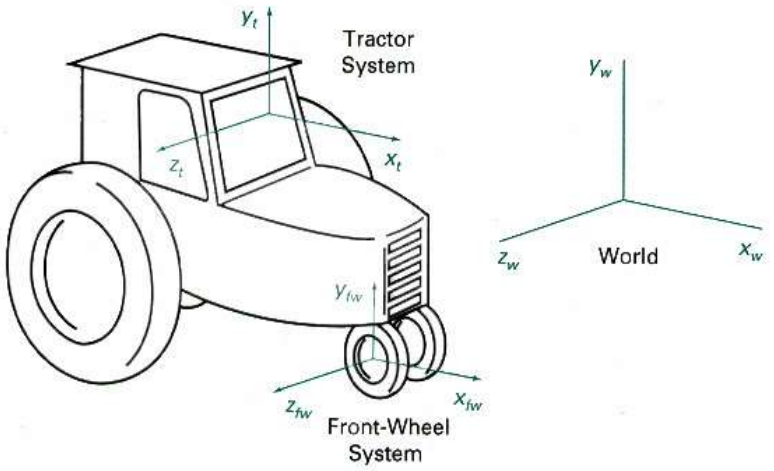
- Multiple Coordinate System
  - Local (modeling) coordinate system
  - World coordinate scene



Word Coordinate System

Coordinate Transformations

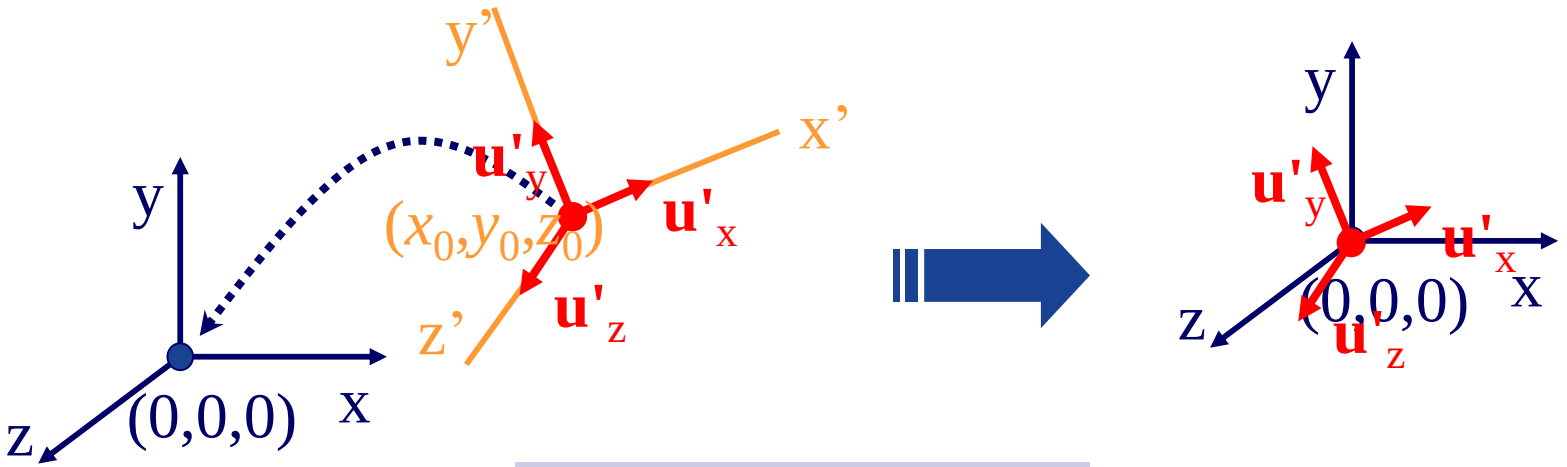
- Example – Simulation of Tractor movement
  - As tractor moves, **tractor coordinate system** and **front-wheel coordinate system** move in world coordinate system
  - **Front wheels** rotate in wheel coordinate system



Coordinate Transformations

- Transformation of an Object Description from One Coordinate System to Another
- Transformation Matrix
  - Bring the two coordinates systems into alignment

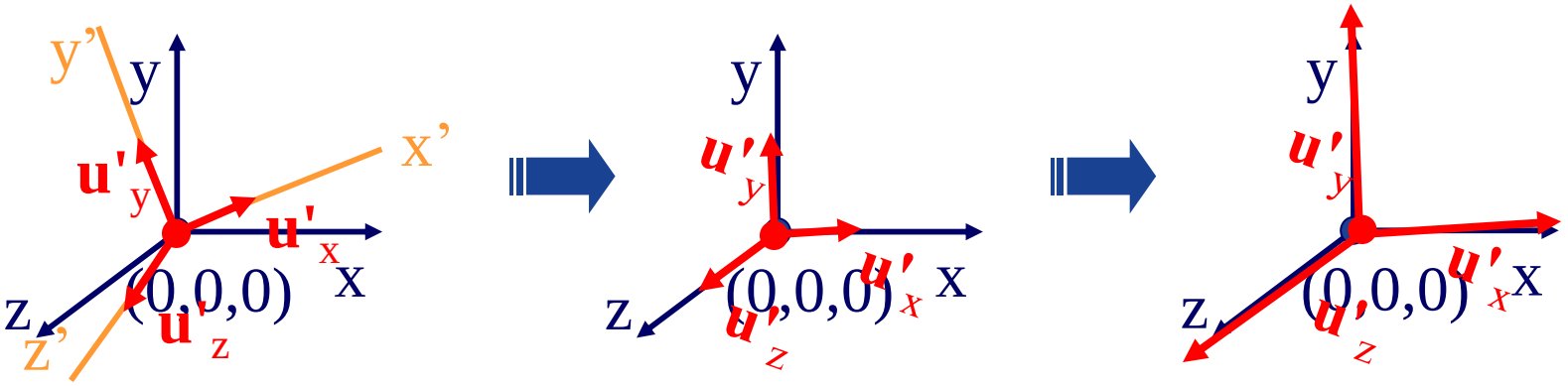
  1. Translation



$$\mathbf{T}(-x_0, -y_0, -z_0)$$

Coordinate Transformations

2. Rotation & Scaling



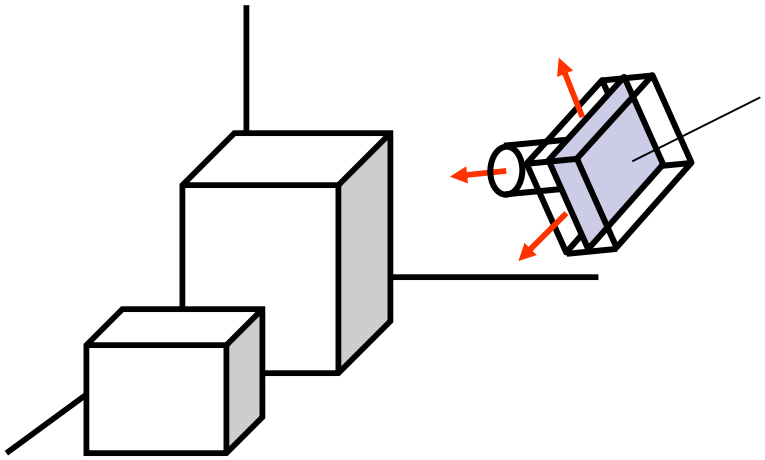
$$\mathbf{R} = \begin{bmatrix} u'_{x_1} & u'_{x_2} & u'_{x_3} & 0 \\ u'_{y_1} & u'_{y_2} & u'_{y_3} & 0 \\ u'_{z_1} & u'_{z_2} & u'_{z_3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D Viewing

- Viewing: virtual camera
- Projection
- Depth
- Visible lines and surfaces
- Surface rendering

3D Viewing pipeline 1

- Similar to making a photo
  - Position and point virtuele camera, press button;



Projection plane aka
Viewing plane

- Pipeline has +/– same structure as in 2D

3D Viewing pipeline 2

MC: Modeling Coordinates

↓ Apply model transformations

WC: World Coordinates

↓ To camera coordinates

VC: Viewing Coordinates

↓ Project

PC: Projection Coordinates

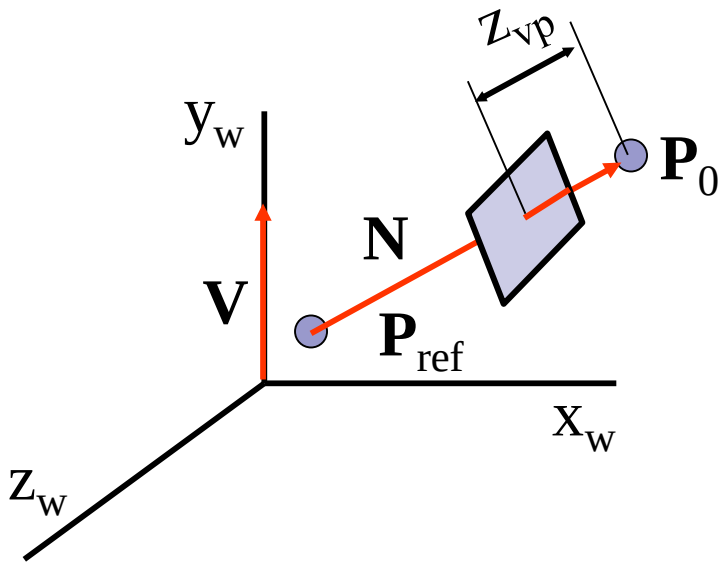
↓ To standard coordinates

NC: Normalized Coordinates

↓ Clip and convert to pixels

DC: Device Coordinates

3D viewing coordinates 1



Specification of projection:

$\mathbf{P}_0$: *View* or *eye point*

$\mathbf{P}_{ref}$: *Center* or *look-at point*

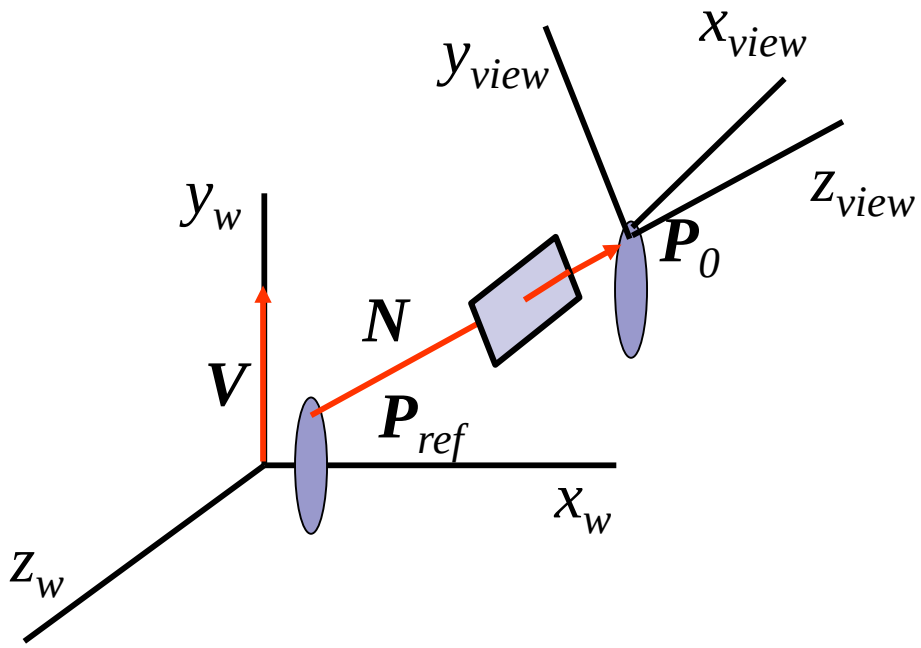
V: *View-up vector* (*projection along vertical axis*)

z_{vp} : *positie view plane*

$\mathbf{P}_0$, $\mathbf{P}_{ref}$, **V**: define viewing coordinate system

Several variants possible

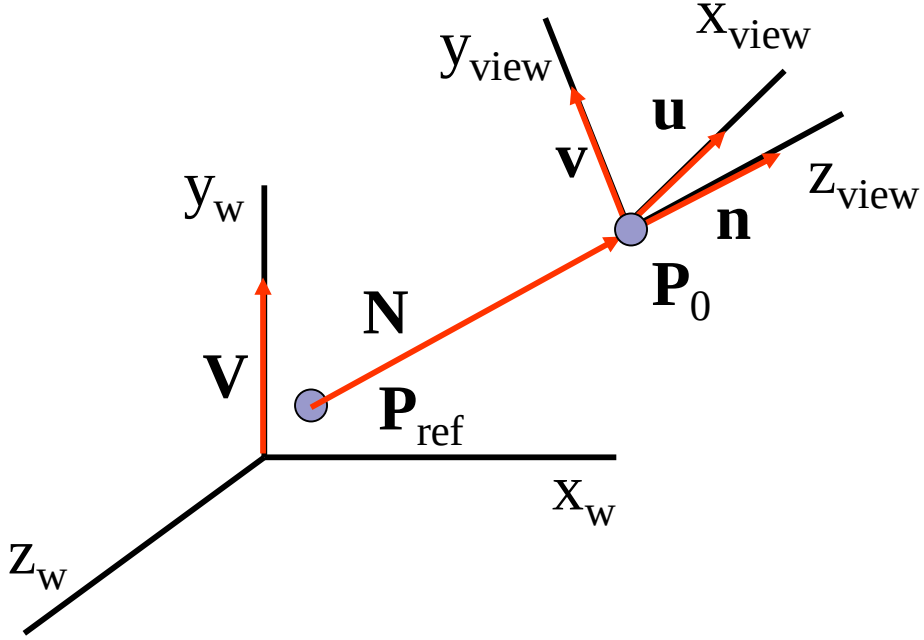
3D viewing coordinates 2



$\boldsymbol{P}_0$, $\boldsymbol{P}_{ref}$, $\boldsymbol{V}$: *define viewing coordinate system*

Several variants possible

3D view coordinates 3



Derivation axis frame:

$$\mathbf{N} = \mathbf{P}_0 - \mathbf{P}_{\text{ref}}$$

$$\mathbf{n} = \frac{\mathbf{N}}{|\mathbf{N}|} = (n_x, n_y, n_z)$$

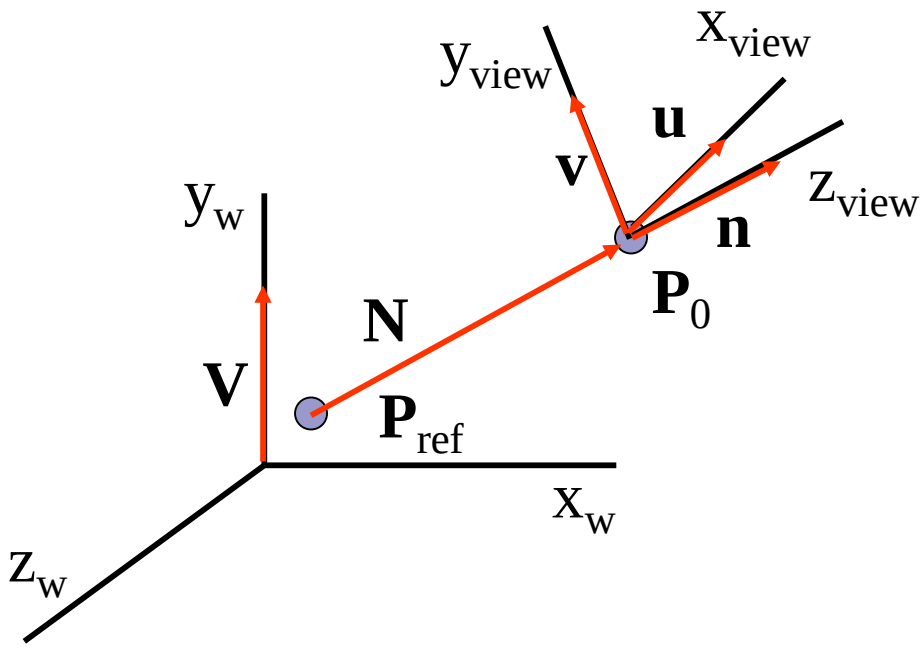
u perpendicular to **V** and **n**:

$$\mathbf{u} = \frac{\mathbf{V} \times \mathbf{n}}{|\mathbf{V}|} = (u_x, u_y, u_z)$$

v perpendicular to **n** and **u**:

$$\mathbf{v} = \mathbf{n} \times \mathbf{u} = (v_x, v_y, v_z)$$

3D viewing coördinaten 4



Transformation world $\rightarrow$ view :

First, translate with $\mathbf{T}(-\mathbf{P_0})$

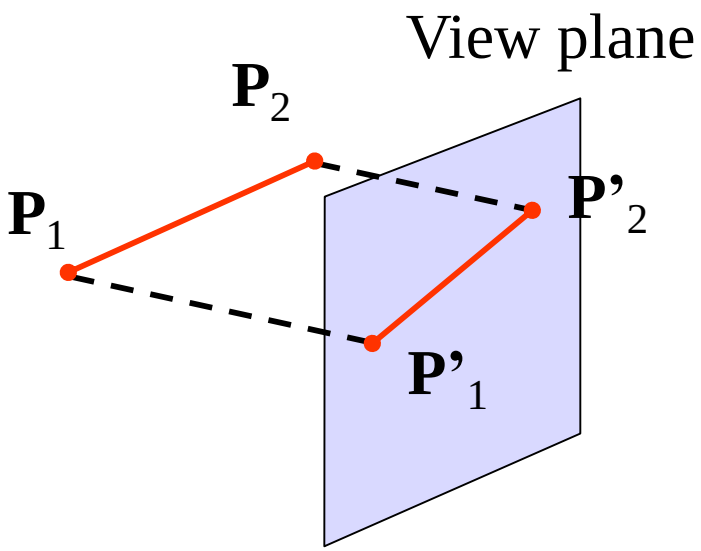
Next, rotate with $\mathbf{R}$:

$$\mathbf{R} = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

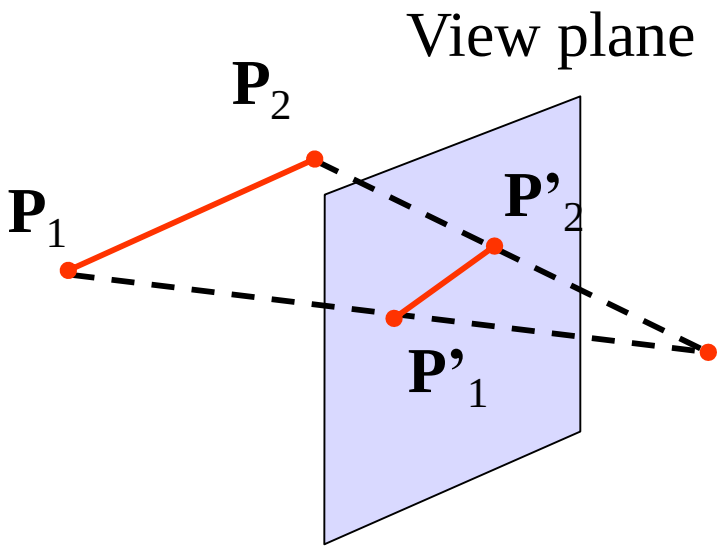
Or :

$$\mathbf{M}_{WC,VC} = \mathbf{RT}$$

Projection transformations

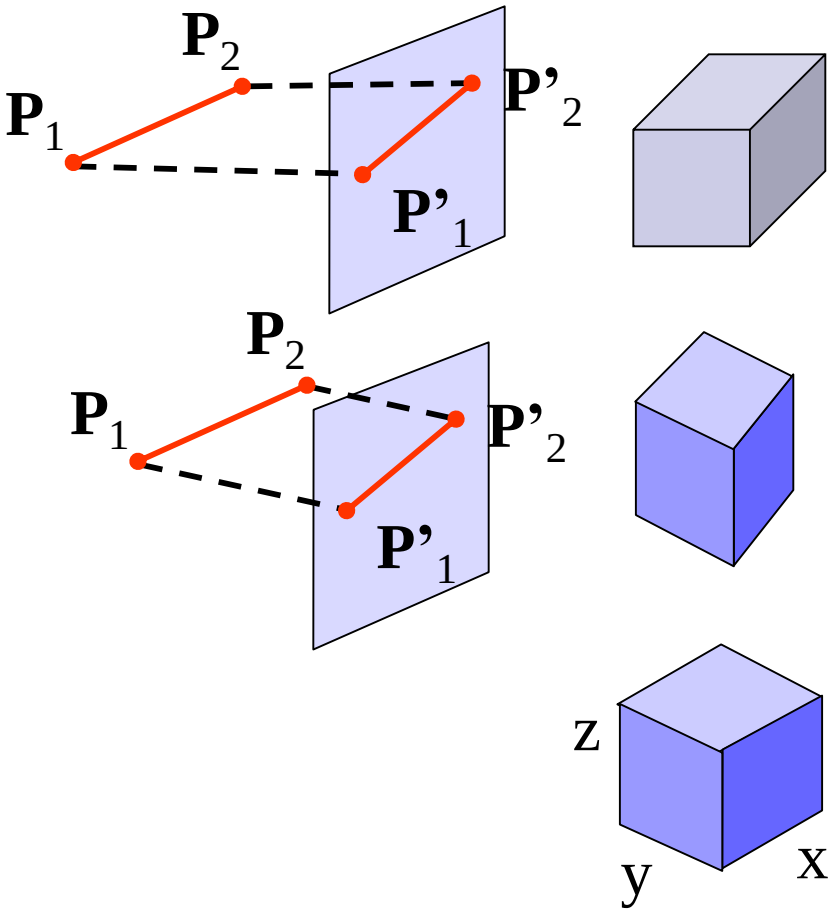


Parallel projection



Perspective projection

Orthogonal projections 1

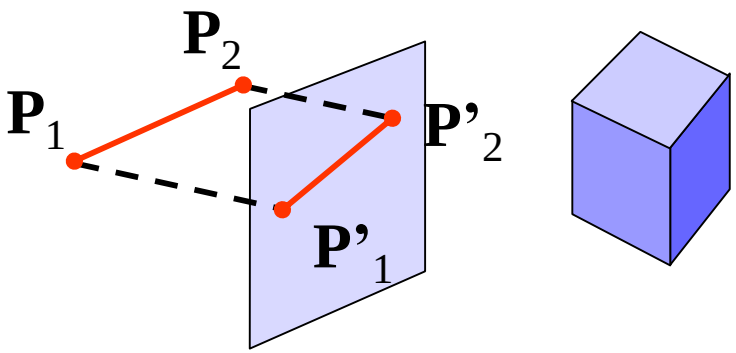


Parallell projection:
Projection lines are parallel

Orthogonal projection:
Projection lines are parallel and perpendicular to projection plane

Isometric projection:
Projection lines are parallel, perpendicular to projection plane, and have the same angle with axes.

Orthogonal projections 2



Orthogonal projection:

Projection of (x, y, z) (from view coordinates to projection coordinates):

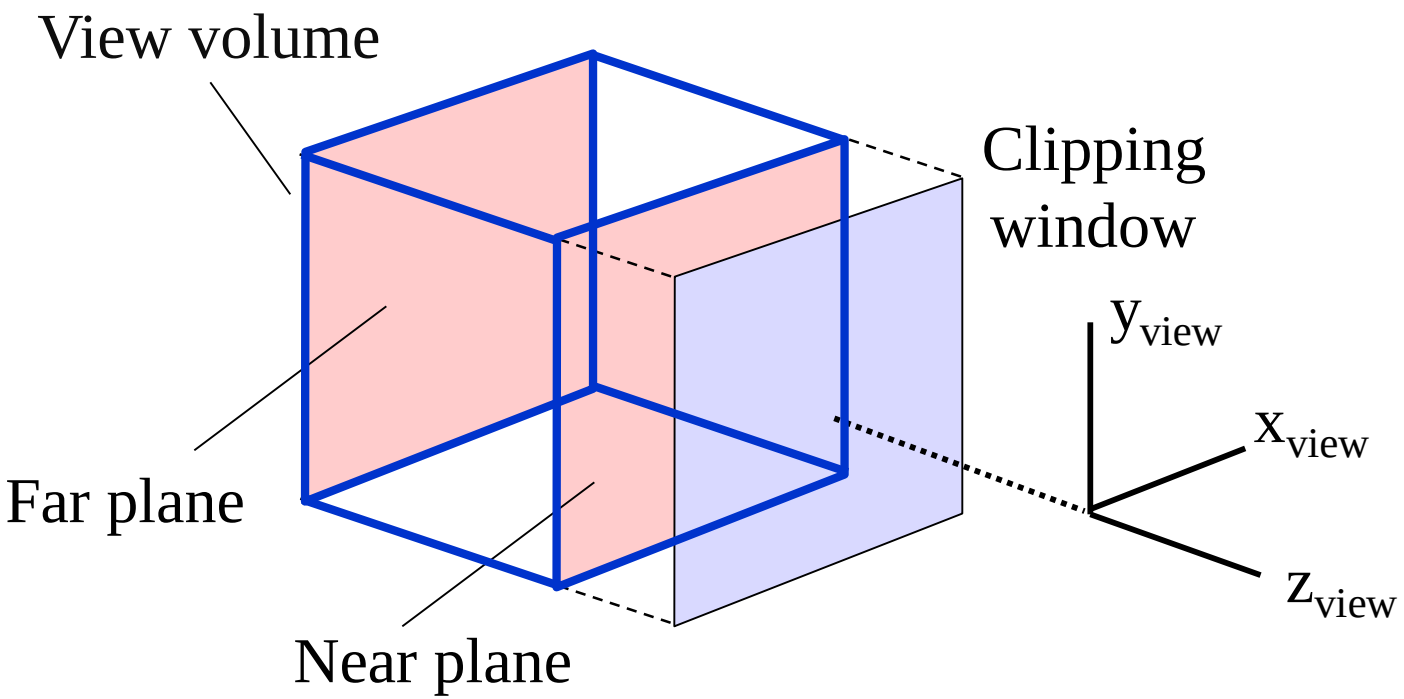
$$x_p = x$$

$$y_p = y$$

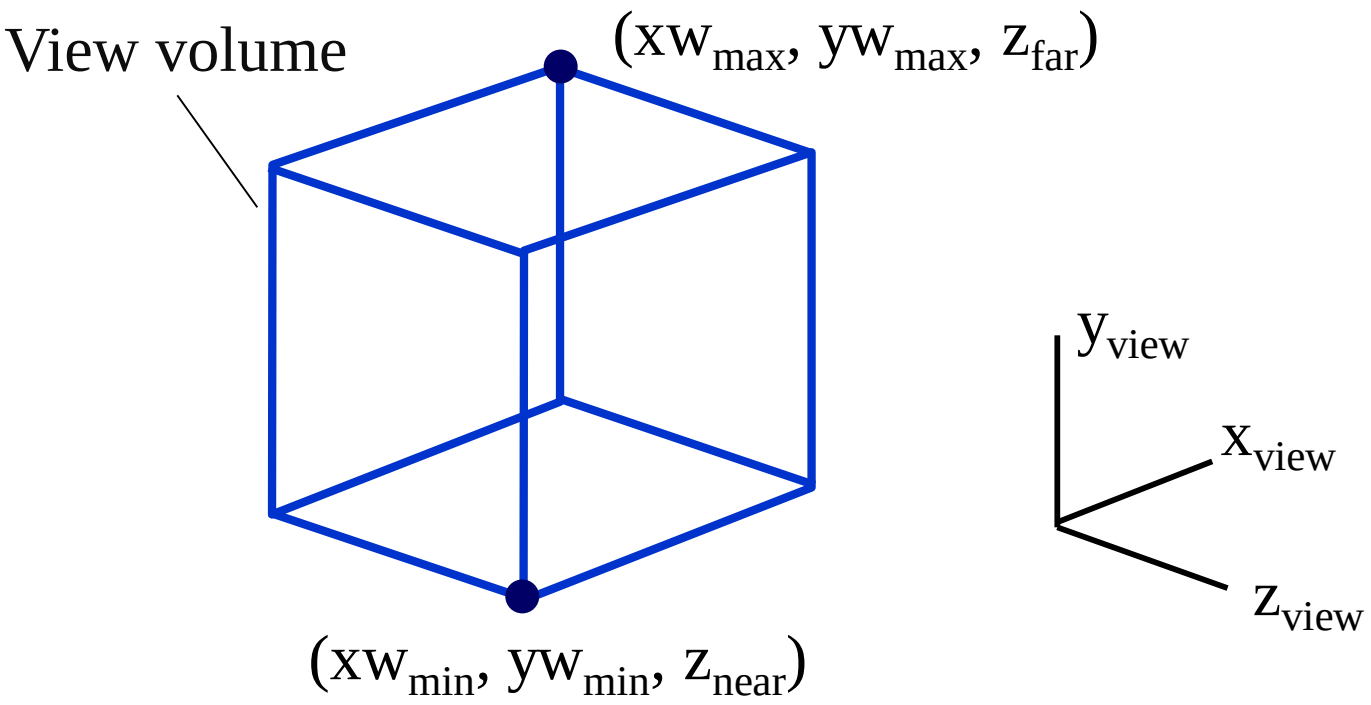
$$z_p = z$$

Trivial!

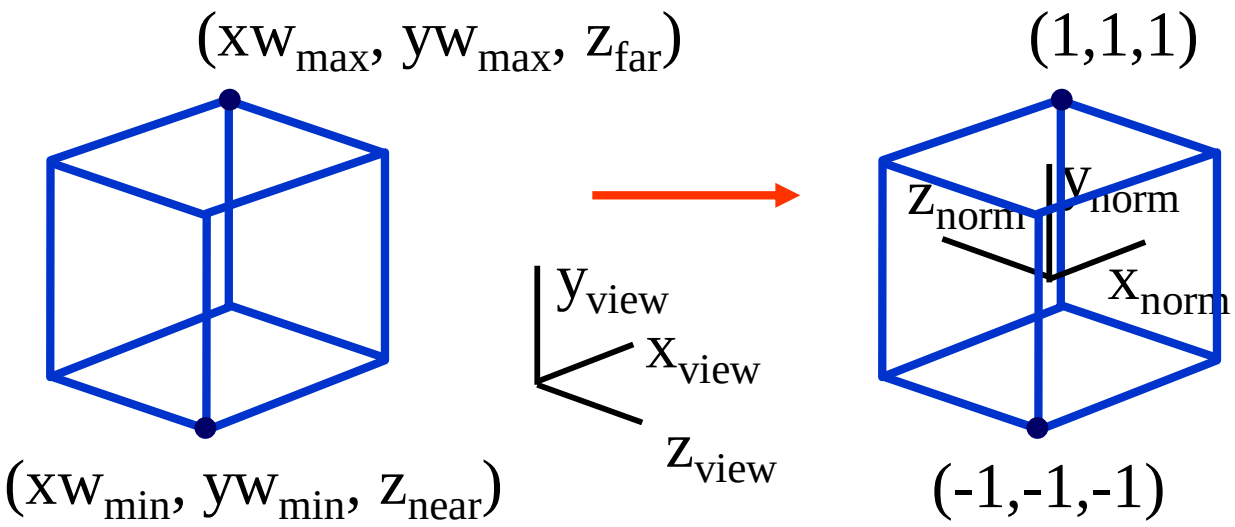
Orthogonal projections 3



Orthogonal projections 4



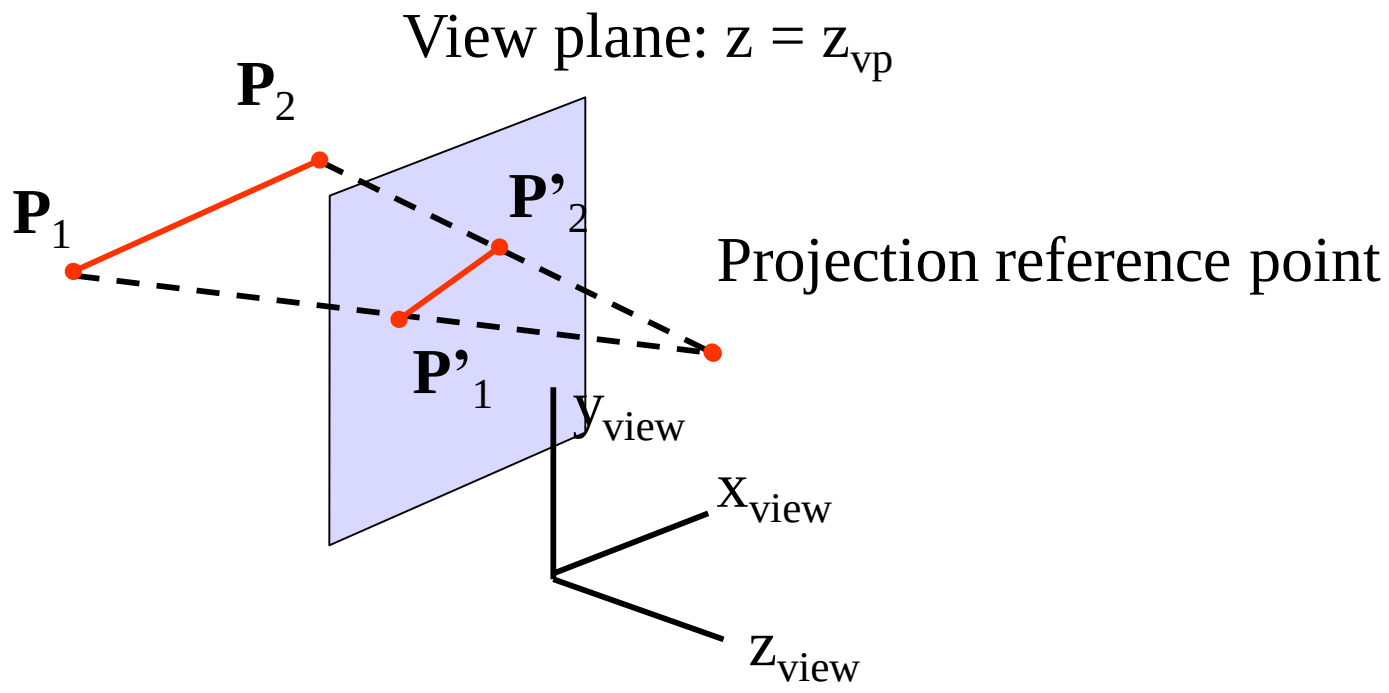
Orthogonal projections 4



View volume → Normalized View volume

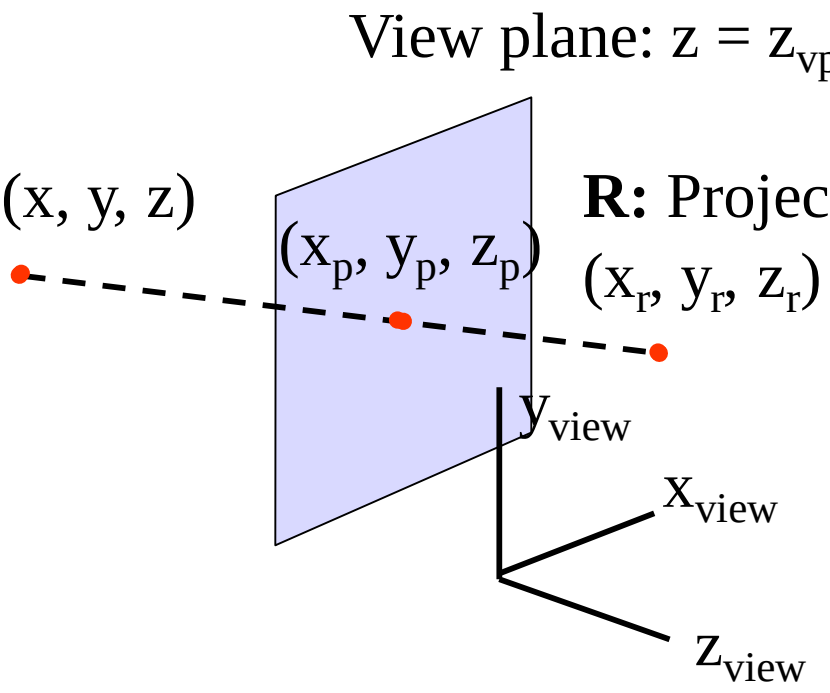
Translation
Scaling
From right- to left handed

Perspective projection 1



View plane: orthogonal to z_{view} axis.

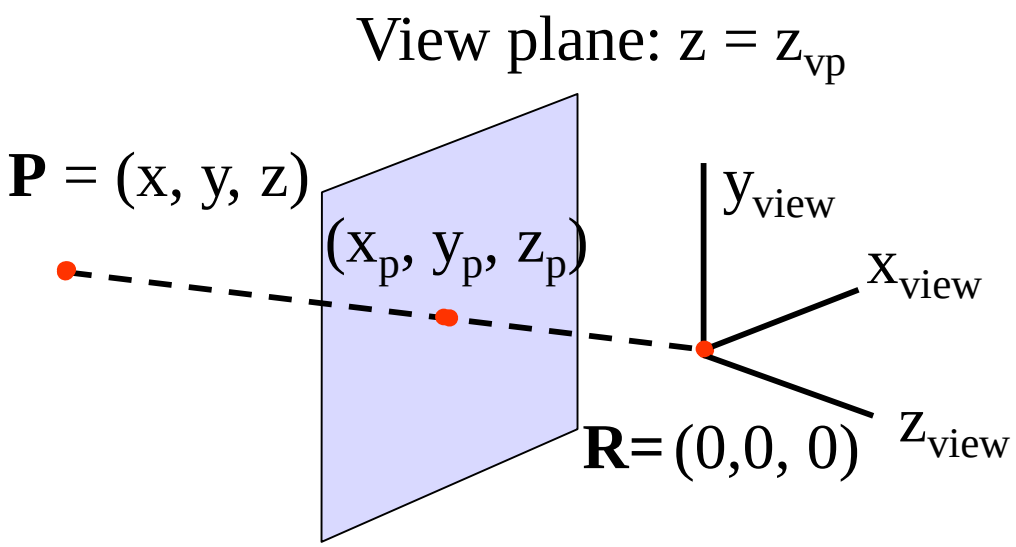
Perspective projection 2



To simplify,
Assume **R** in origin

Question: What is the projection of **P** on the view plane?

Perspective projection 3



Line from **R** (origin) to **P**:

$\mathbf{X}' = u\mathbf{P}$, with $0 \le u \le 1$,

or $x' = ux$; $y' = uy$; and $z' = uz$.

At crossing with plane:

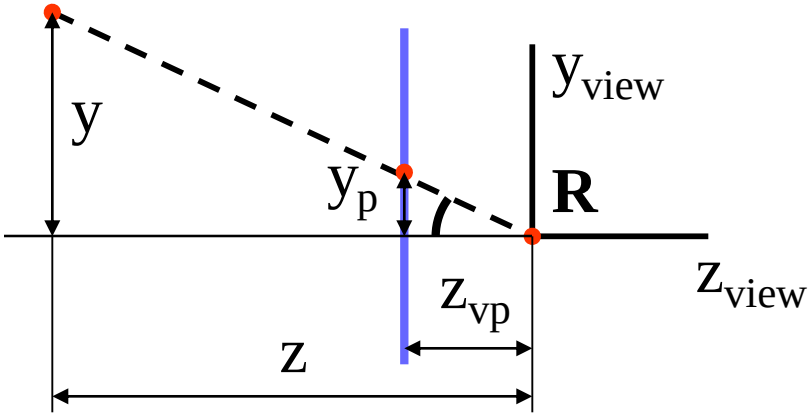
$z' = z_{vp}$ hence $u = \frac{z_{vp}}{z}$.

Substitution gives

$$x_p = \frac{z_{vp}}{z} x \text{ and } y_p = \frac{z_{vp}}{z} y$$

Perspective projection 4

P = (x, y, z) View plane



Viewed from the side

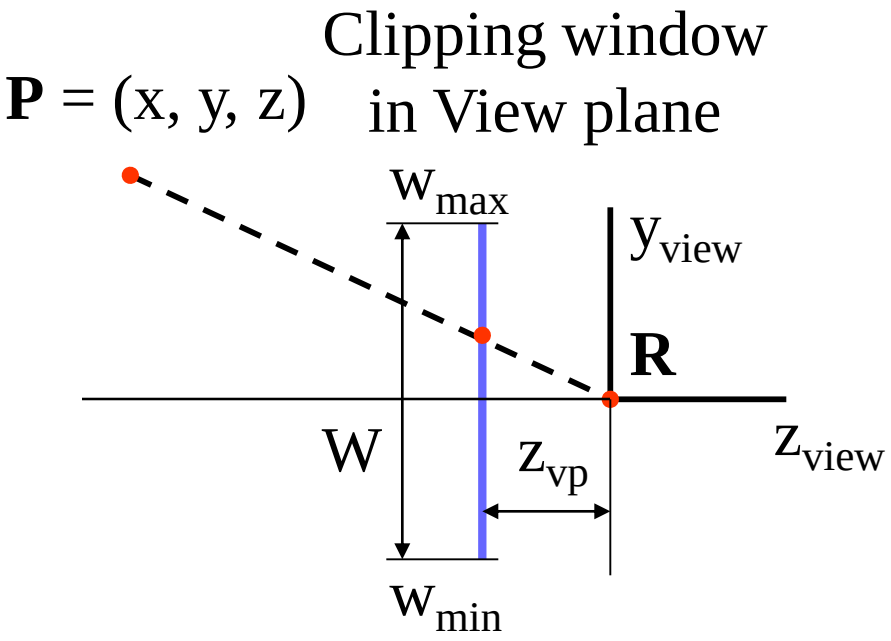
We can see that :

$$\frac{y}{z} = \frac{y_p}{z_{vp}}$$

hence

$$y_p = \frac{z_{vp}}{z} y$$

Perspective projection 5



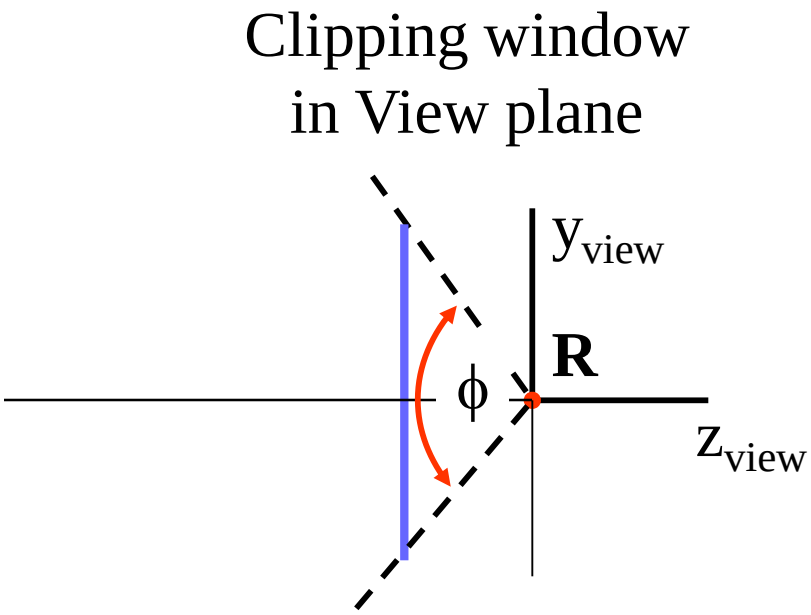
Viewed from the side

Ratio between

$W=w_{max}-w_{min}$ and z_{vp}

determines strenght perspective

Perspective projection 6



Viewed from the side

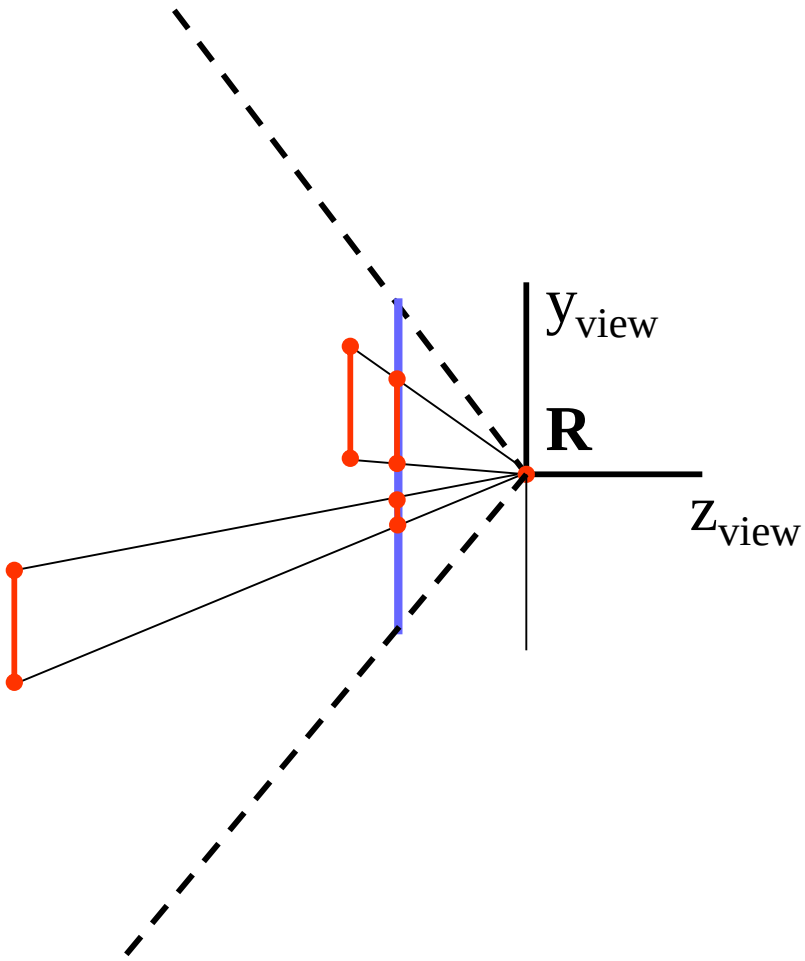
Ratio between

$W=w_{max}-w_{min}$ and z_{vp}

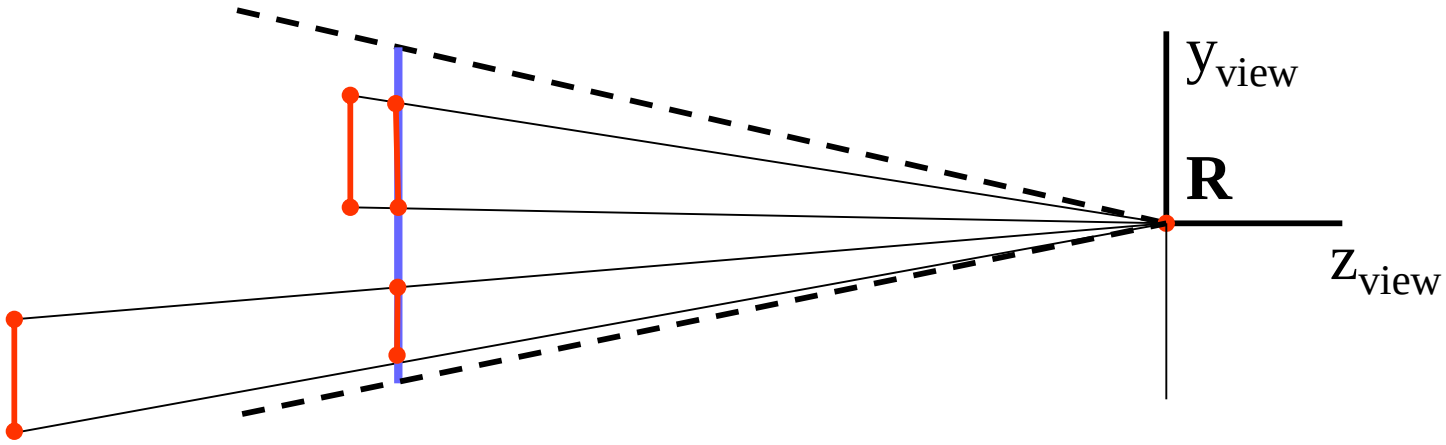
determines strenght perspective.

This ratio is $2\tan(\phi/2)$,
with ϕ the view angle.

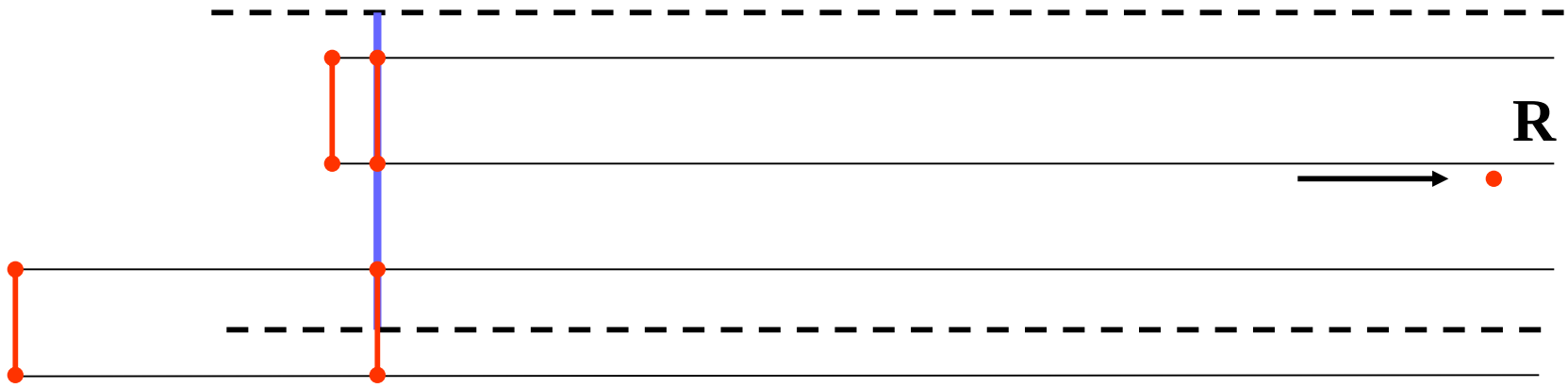
Perspective projection 7



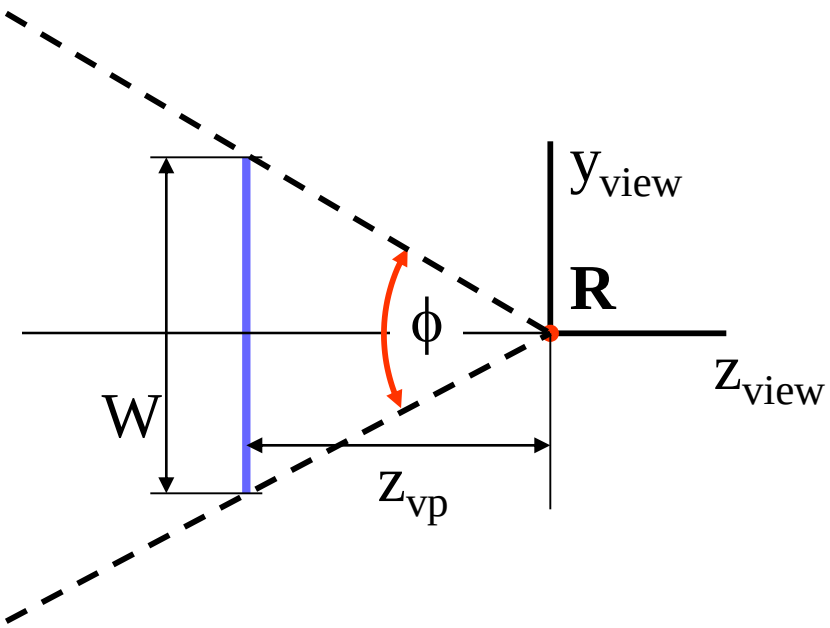
Perspective projection 7



Perspective projection 7



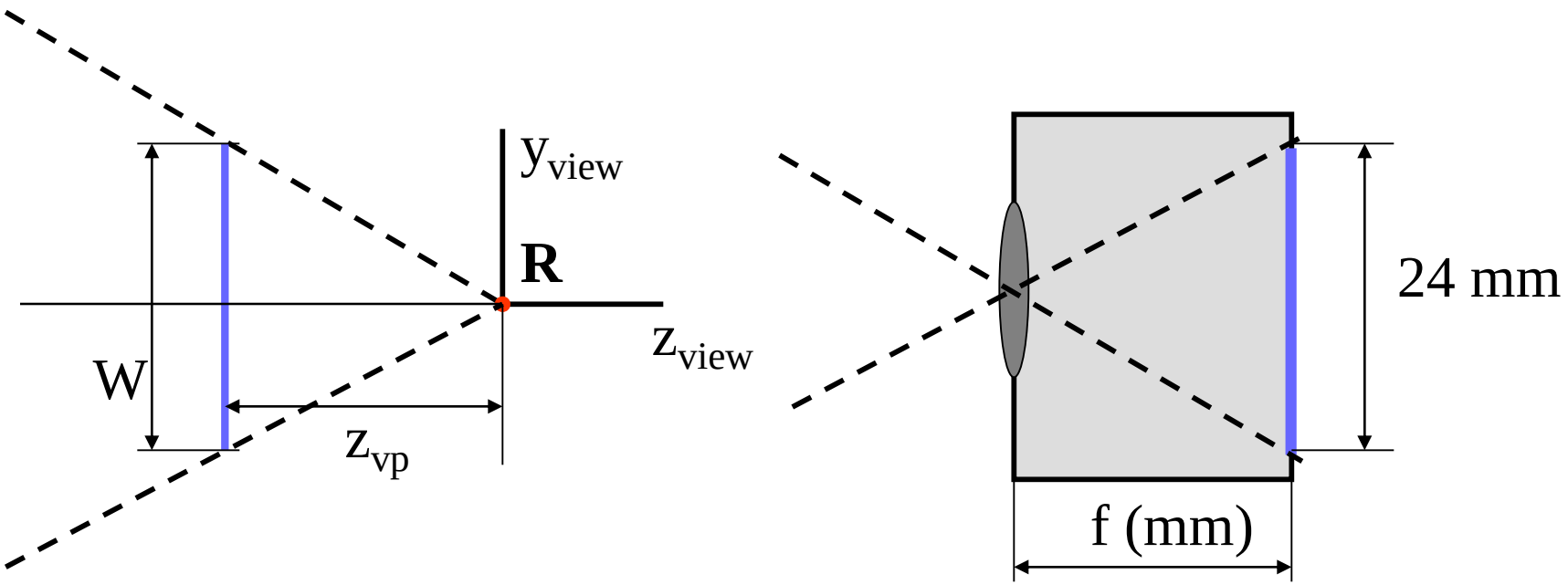
Perspective projection 8



Option 1: Specify view angle ϕ.

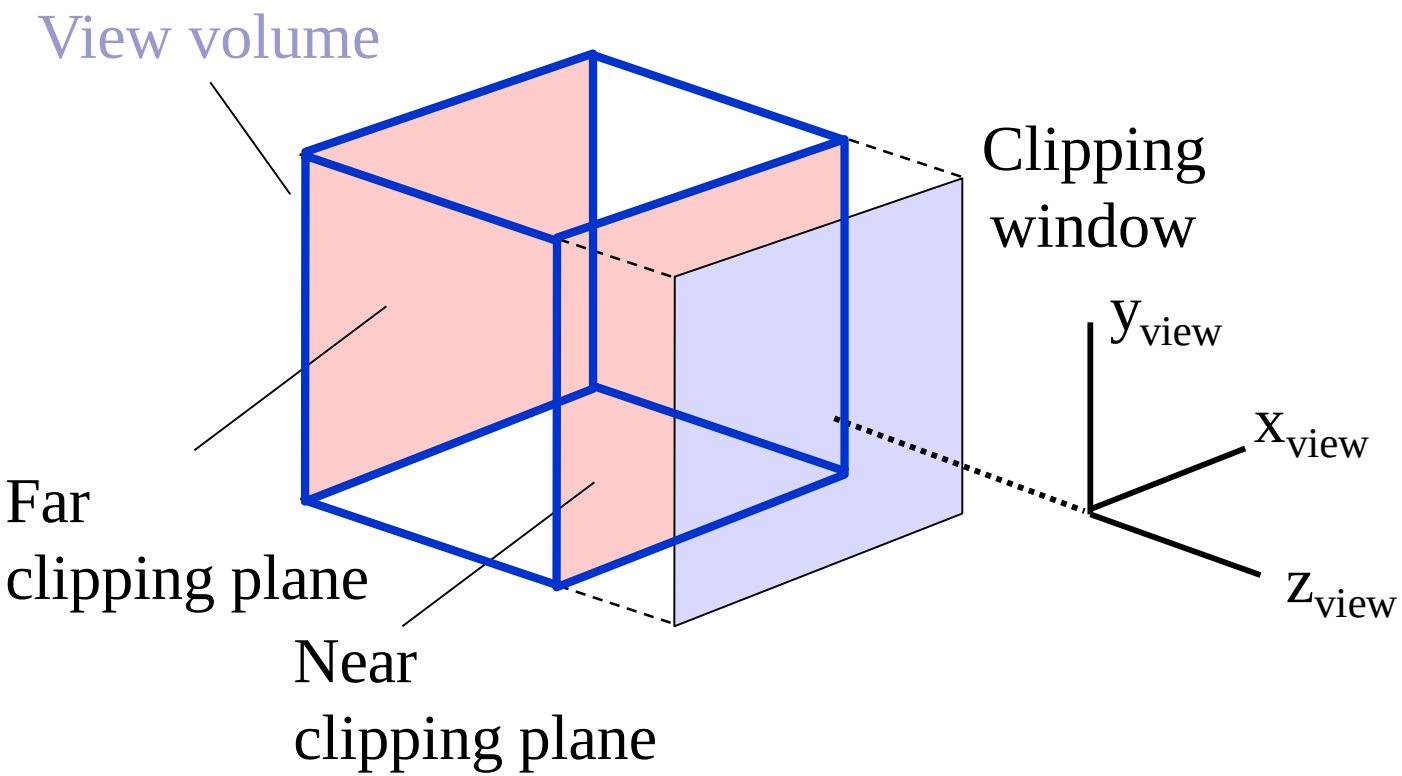
How to specify the ratio between W en z_{vp}?
How to specify ‘how much’ perspective I want to see?

Perspective projection 9

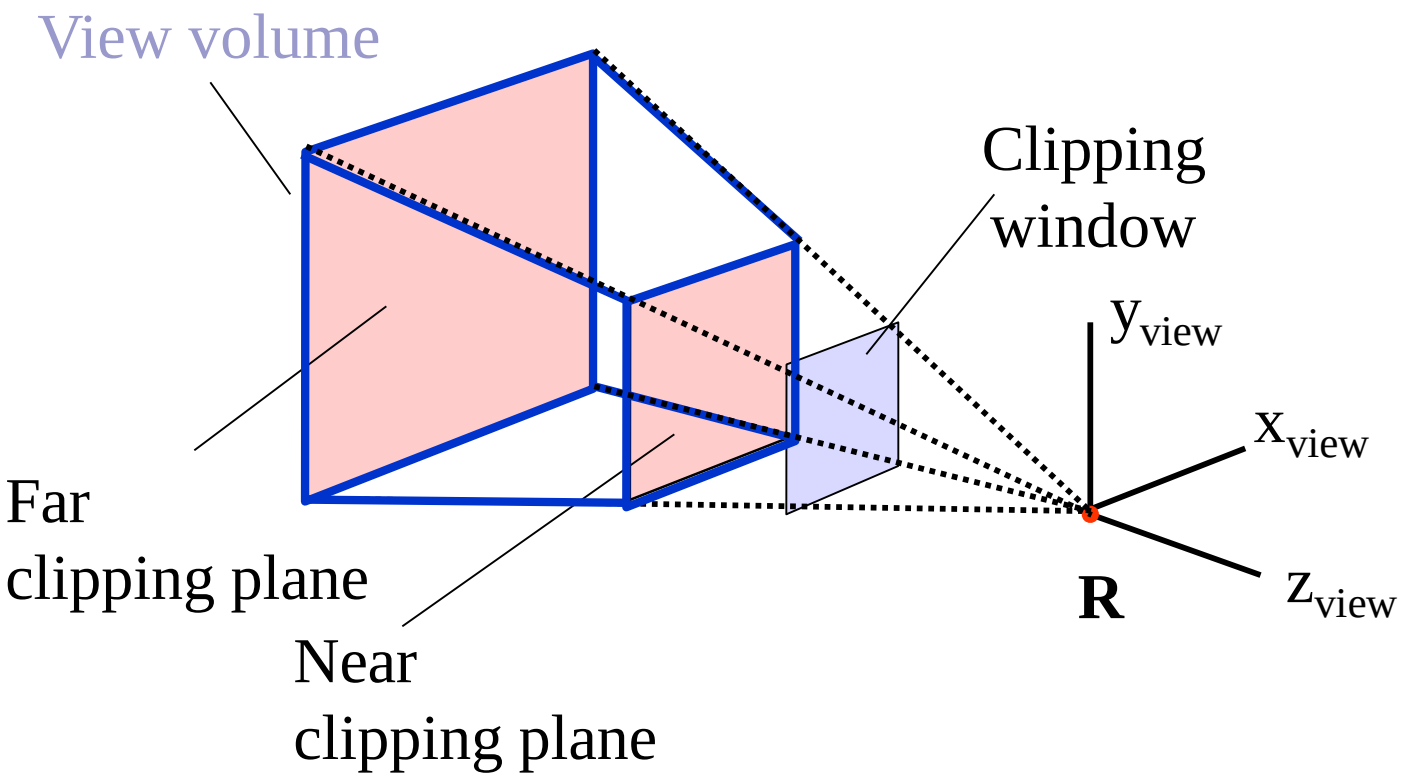


Use camera as metaphor. User specifies focal length f .
(20 mm – wide angle, 50 mm – normal, 100 mm – tele).
Application adjusts W and/or z_{vp}, such that $W/z_{vp} = 24/f$.
For x: $W/z_{vp} = 36/f$.

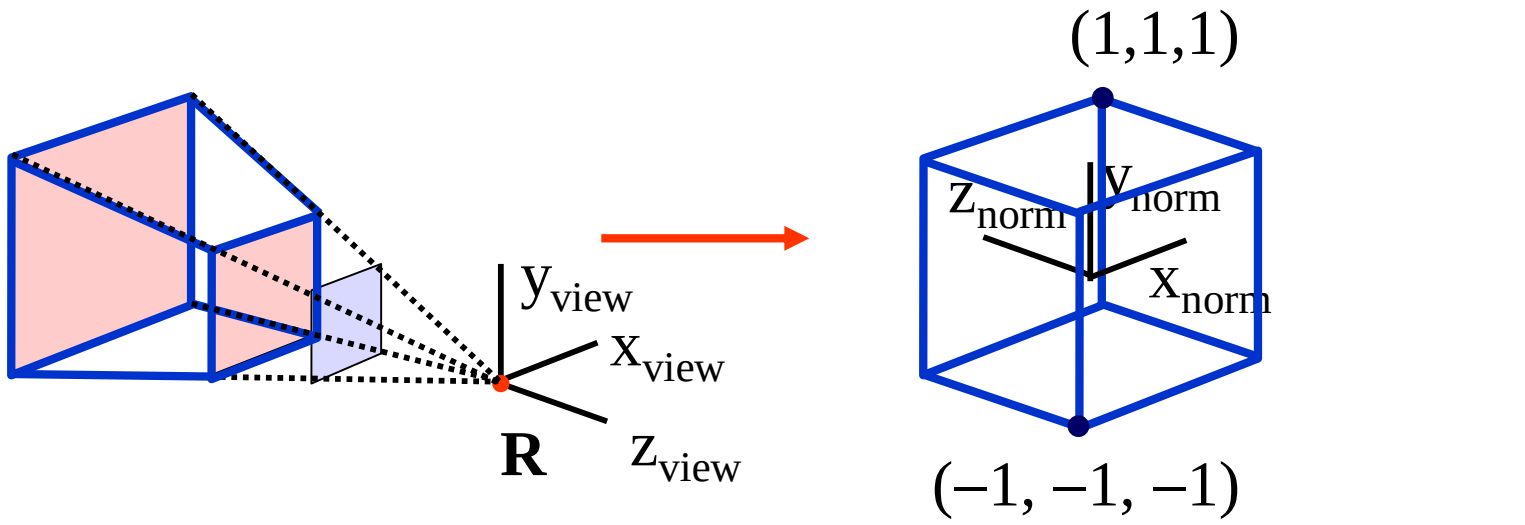
View volume orthogonal...



View volume perspective



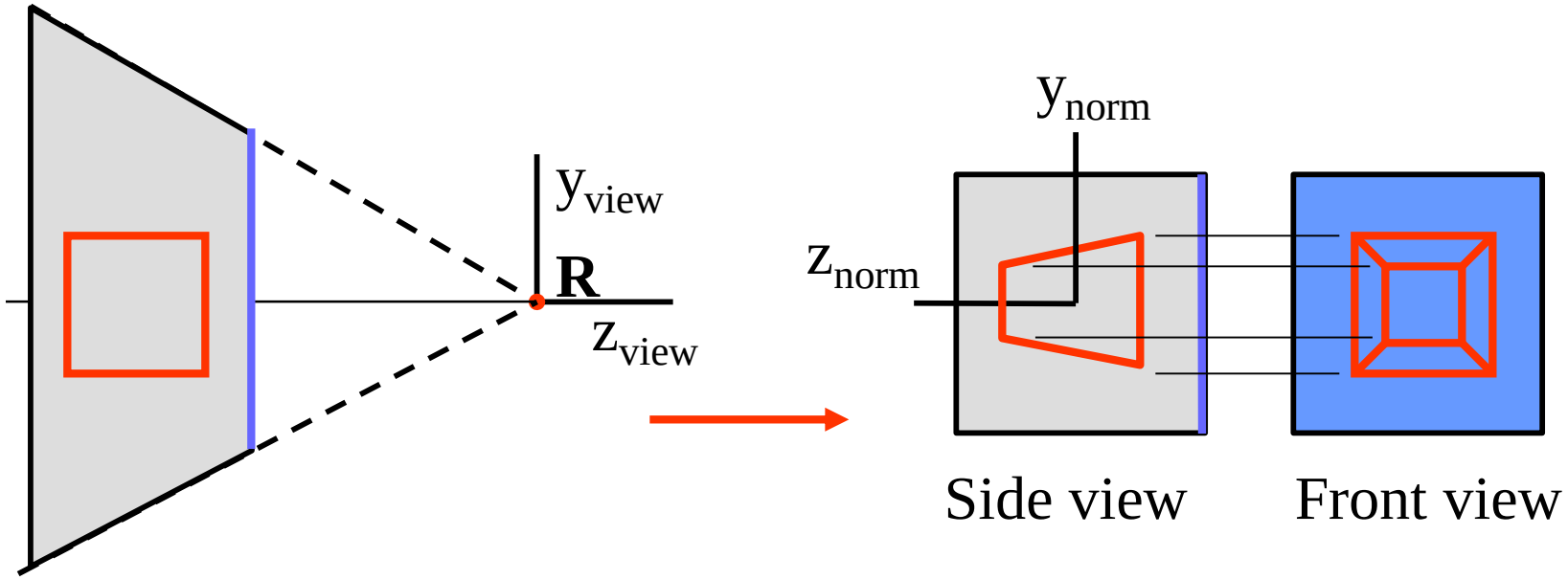
To Normalized Coordinates…



Rectangular frustum
View Volume

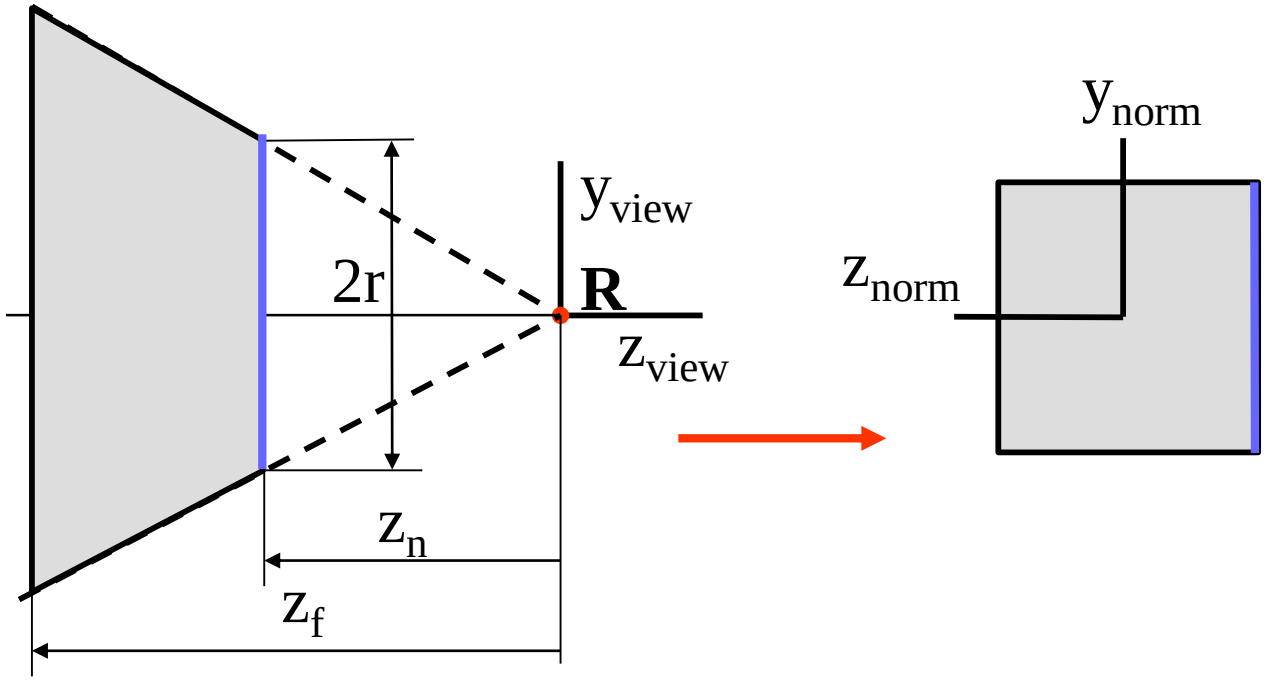
Normalized View volume

Perspective projection 10



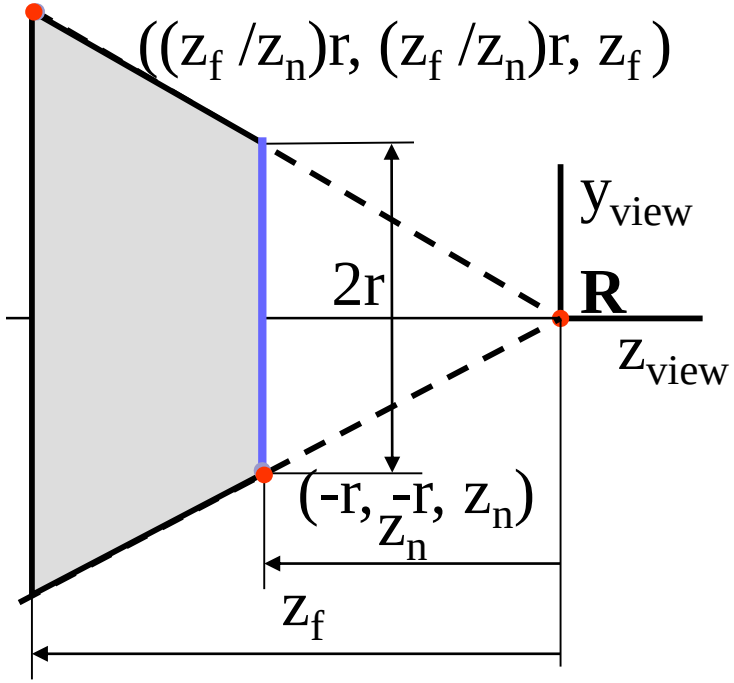
Perspective transformation:
Distort space, such that perpendicular projection gives an image in perspective.

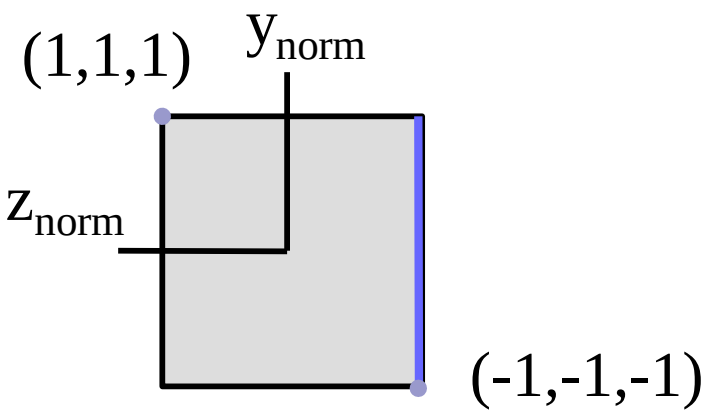
Perspective projection 10



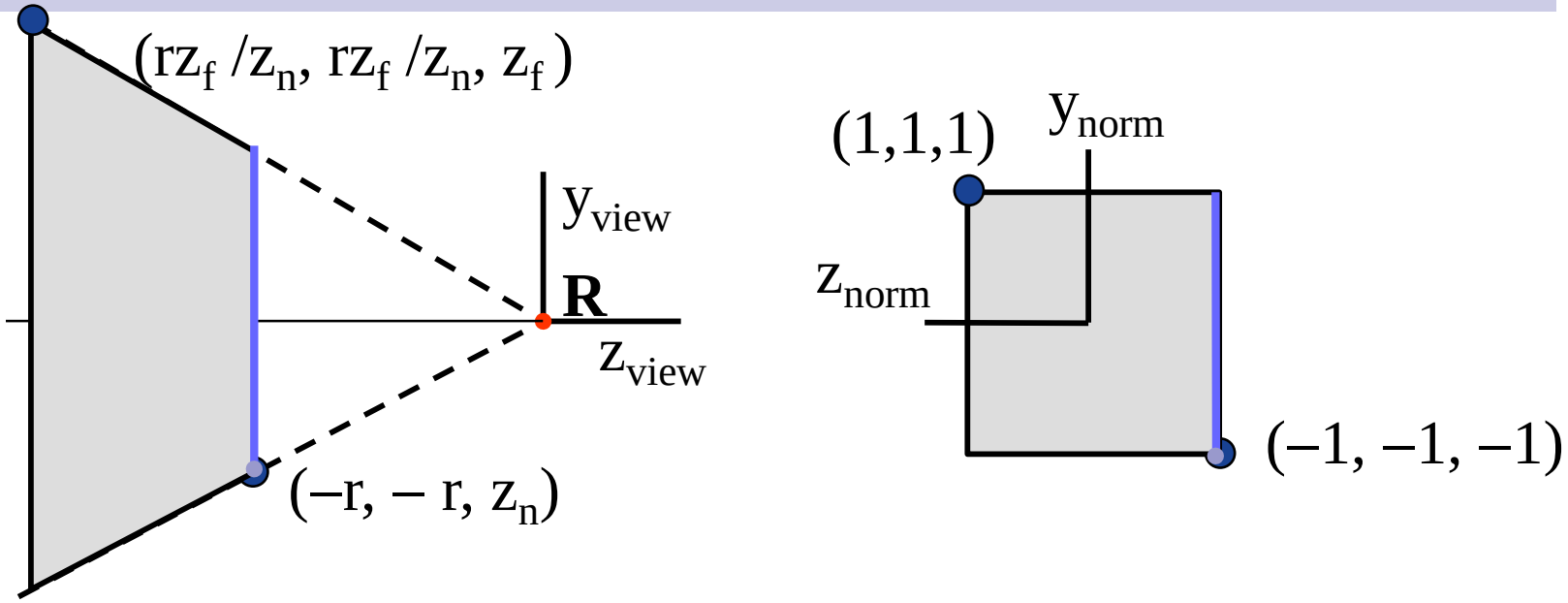
Simplest case:
Square window,
clipping plane coincides with view plane: $z_n = z_{vp}$

Perspective projection 11





Perspective projection 11



Earlier: $x_p = \frac{z_{vp}}{z} x, \quad y_p = \frac{z_{vp}}{z} y$

How to put this transformation in the pipeline?
How to process division by z?

Homogeneous coordinates (reprise)

- Add extra coordinate:

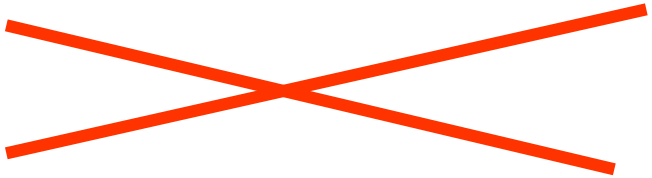
  $\mathbf{P} = (p_x , p_y , p_z , p_h)$ or

  $\mathbf{x} = (x, y, z, h)$

- Cartesian coordinates: divide by h

  $\mathbf{x} = (x/h, y/h, z/h)$

- Points: $h = 1$ (temporary…)



perspective: h = –z !

Homogeneous coordinates (reprise)

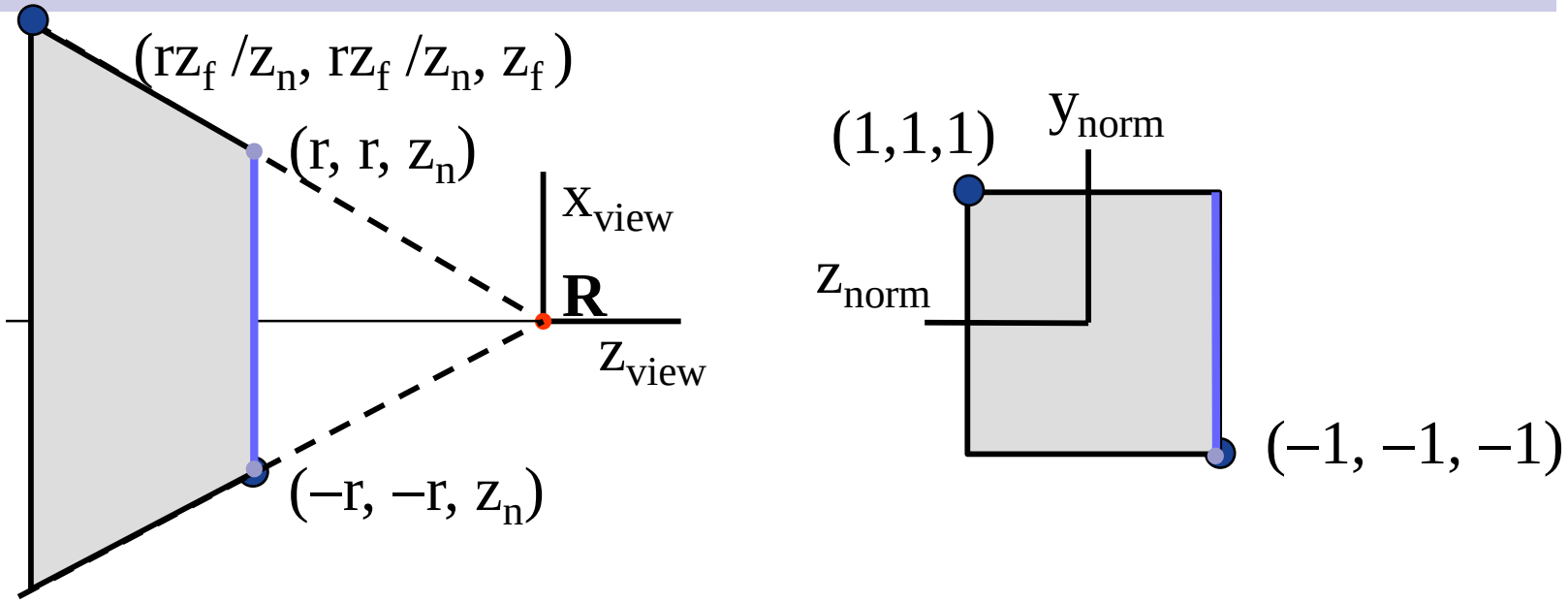
Perspective transformation can be described by :

$$\begin{pmatrix} x_h \\ y_h \\ z_h \\ h \end{pmatrix} = \begin{pmatrix} s_{xx} & s_{xy} & s_{xz} & t_x \\ s_{yx} & s_{yy} & s_{yz} & t_y \\ s_{zx} & s_{zy} & s_{zz} & t_z \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_v \\ y_v \\ z_v \\ 1 \end{pmatrix}$$

such that projected coordinates are given by :

$$\begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix} = \begin{pmatrix} x_h / h \\ y_h / h \\ z_h / h \end{pmatrix}.$$

Perspective projection 12



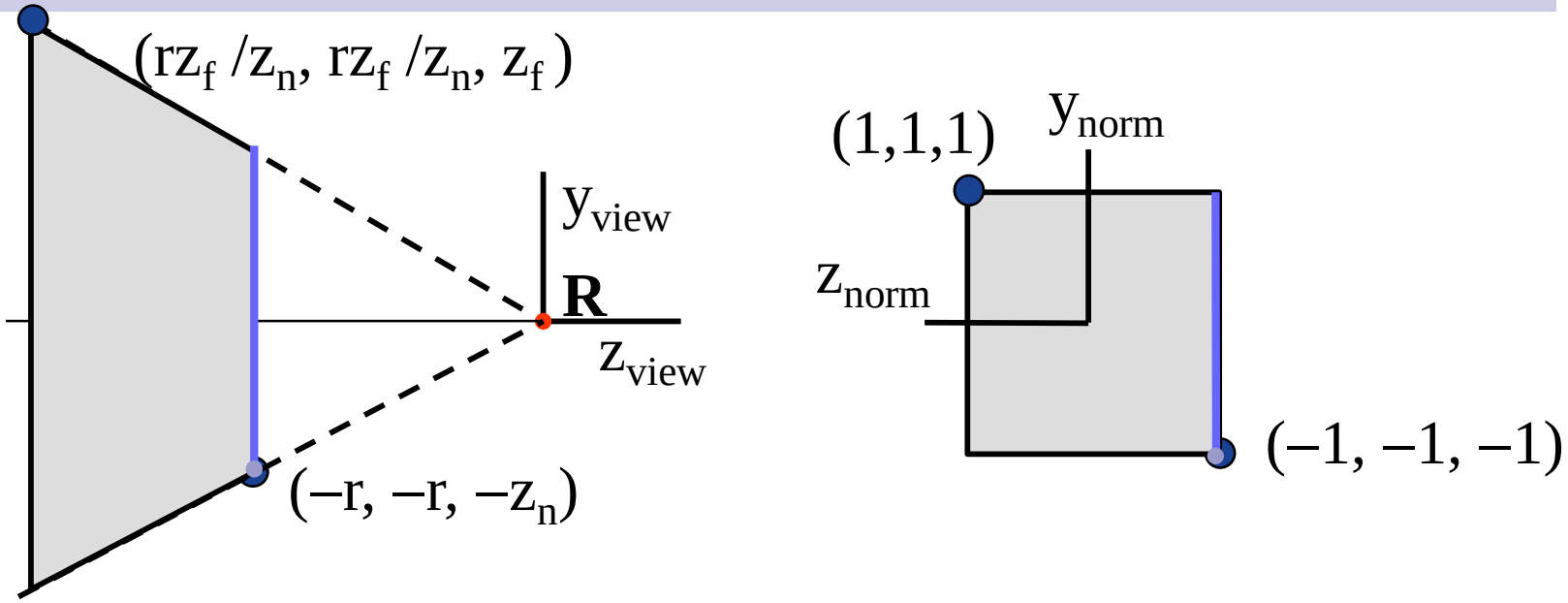
First x. Generic form is $x_p = (s_{xx}x + s_{xy}y + s_{xz}z + t_x)/-z.$

If $x = r$ and $z = z_n$, then $x_p = 1.$

If $x = rz_f / z_n$ and $z = z_f$, then also $x_p = 1.$

Elaboration gives : $s_{xx} = -z_n / r, s_{xy} = s_{xz} = t_x = 0.$

Perspective projection 13

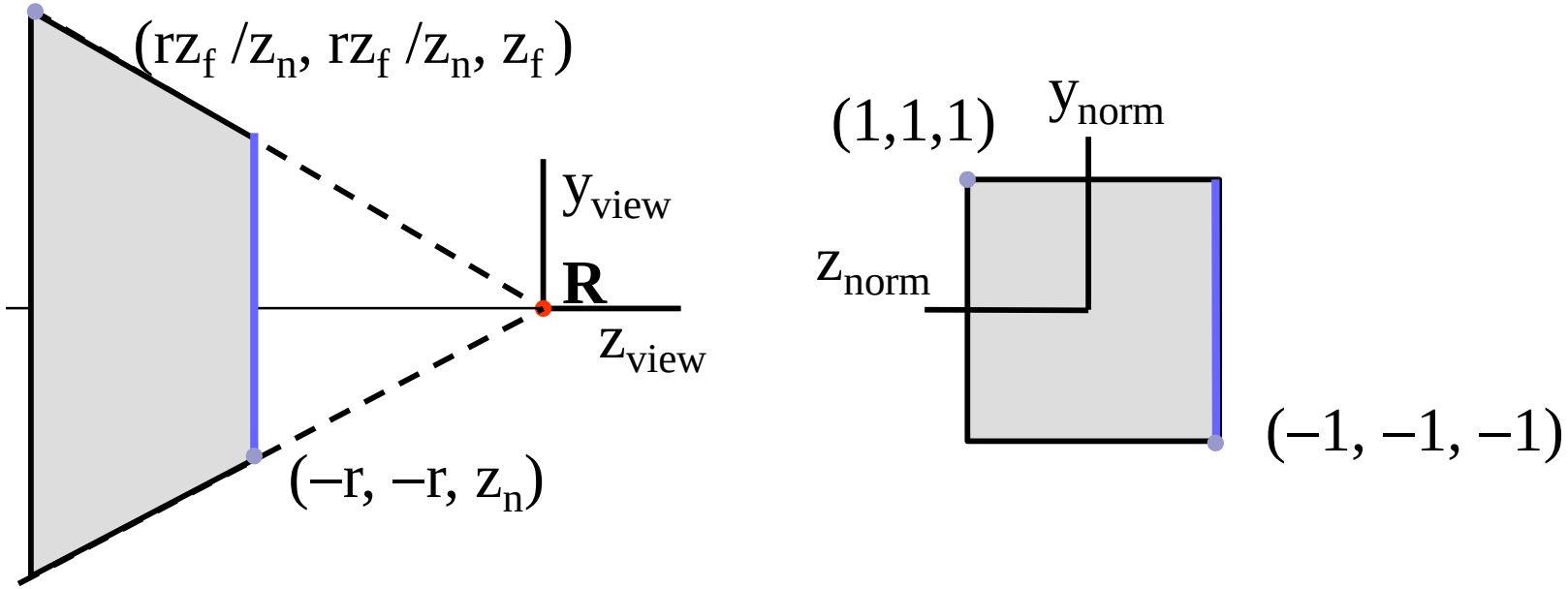


Next the *y*. Same as *x*, gives :

$$y_p = (-z_n / r) y / - z.$$

$$\text{Or}: s_{yy} = (-z_n / r), s_{yx} = s_{yz} = t_y = 0.$$

Perspective projection 14



Finally : z. Generic form is : $z_p = (s_{zx}x + s_{zy}y + s_{zz}z + t_z)/-z.$

If $z = z_n$, then $z_p = -1.$

If $z = z_f$, then $z_f = 1.$ Elaboration gives

$$s_{zz} = \frac{z_n + z_f}{z_n - z_f}, t_z = \frac{-2z_n z_f}{z_n - z_f}, s_{zx} = s_{zy} = 0.$$

Perspective projection 15

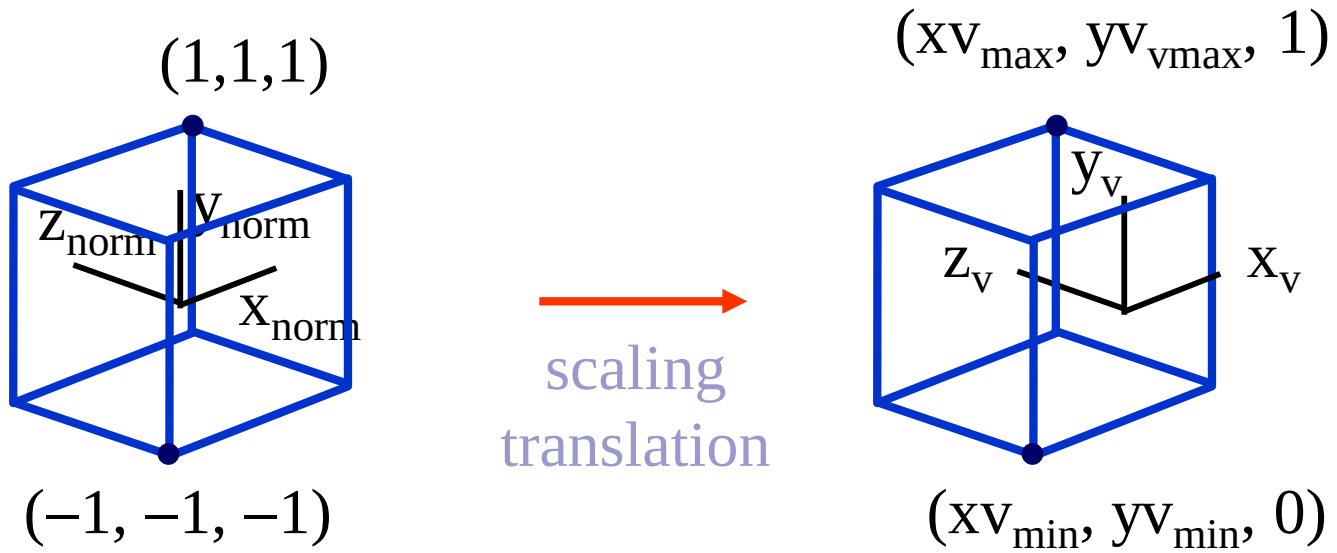
Perspective transformation can hence be described by :

$$\begin{pmatrix} x_h \\ y_h \\ z_h \\ h \end{pmatrix} = \begin{pmatrix} -z_n / r & 0 & 0 & 0 \\ 0 & -z_n / r & 0 & 0 \\ 0 & 0 & \dfrac{z_n + z_f}{z_n - z_f} & \dfrac{-2 z_n z_f}{z_n - z_f} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_v \\ y_v \\ z_v \\ 1 \end{pmatrix}$$

where the projected coordinates follow from :

$$\begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix} = \begin{pmatrix} x_v / h \\ y_v / h \\ z_v / h \end{pmatrix}.$$

3D Viewport coordinates 1



Normalized View volume

3D screen

3D Viewport coordinaten 2

Transformation from normalized view coordinates to 3D screencoordinates:

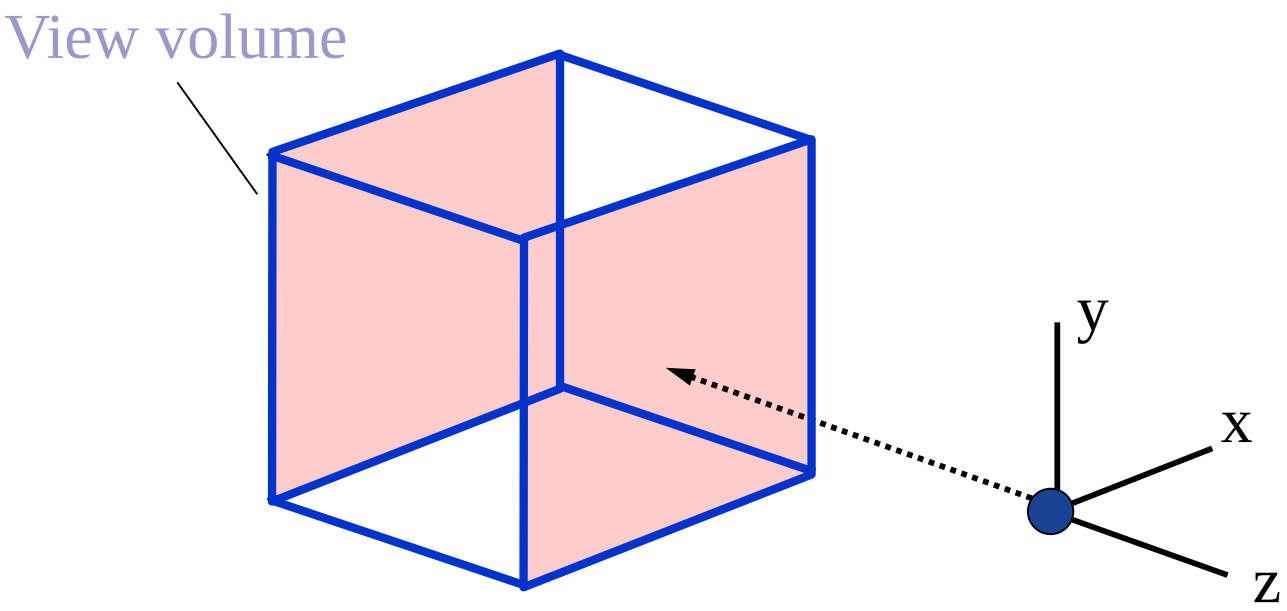
$$\begin{pmatrix} x_s \\ y_s \\ z_s \\ h \end{pmatrix} = \begin{pmatrix} \frac{xv_{\max} - xv_{\min}}{2} & 0 & 0 & 0 \\ 0 & \frac{yv_{\max} - yv_{\min}}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix}$$

OpenGL 3D Viewing 1

3D Viewing in OpenGL:

- Position camera;
- Specify projection.

OpenGL 3D Viewing 2



Camera always in origin, in direction of negative z-axis.

Convenient for 2D, but not for 3D.

OpenGL 3D Viewing 3

Solution for view transform: Transform your model such that you look at it in a convenient way.

Approach 1: Do it yourself. Apply rotations, translations, scaling, etc., before rendering the model. Surprisingly difficult and error-prone.

Approach 2: Use **`gluLookAt();`**

OpenGL 3D Viewing 4

```
MatrixMode(GL_MODELVIEW);
gluLookAt(x0,y0,z0, xref,yref,zref, Vx,Vy,Vz);
```

x0,y0,z0: $\mathbf{P}_0$, viewpoint, location of camera;

xref,yref,zref: $\mathbf{P}_{ref}$, centerpoint;

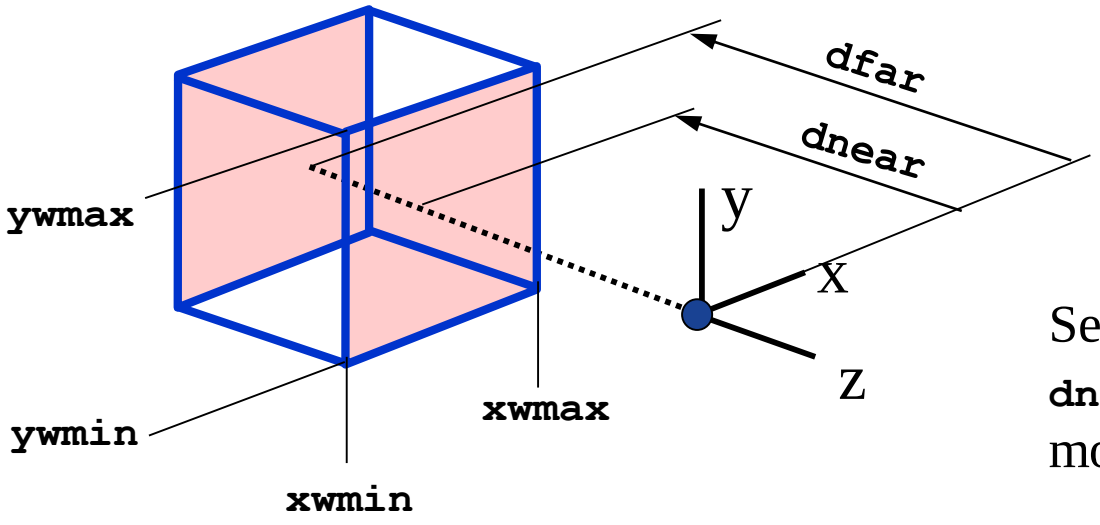
Vx,Vy,Vz: **V**, view-up vector.

Default: $\mathbf{P}_0 = (0, 0, 0)$; $\mathbf{P}_{ref} = (0, 0, -1)$; $\mathbf{V}=(0, 1, 0)$.

OpenGL 3D Viewing 5

Orthogonal projection:

```
MatrixMode(GL_PROJECTION);
glOrtho(xwmin, xwmax, ywmin, ywmax, dnear, dfar);
```



xwmin, xwmax, ywmin,ywmax: specification window

dnear: distance to near clipping plane

dfar : distance to far clipping plane

Select **dnear** and **dfar** right:

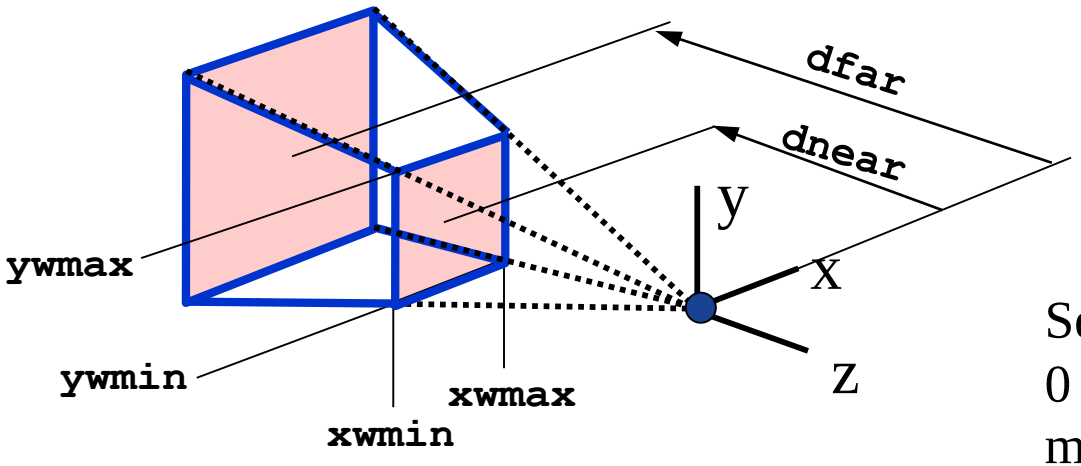
dnear < **dfar**,

model fits between clipping planes.

OpenGL 3D Viewing 6

Perspective projection:

```
MatrixMode(GL_PROJECTION);
glFrustrum(xwmin, xwmax, ywmin, ywmax, dnear, dfar);
```



xwmin, xwmax, ywmin,ywmax: specification window
dnear: distance to near clipping plane
dfar : distance to far clipping plane

Select **dnear** and **dfar** right:
0 < **dnear** < **dfar**,
model fits between clipping planes.

Standard projection: **xwmin = -xwmax,**
ywmin = -ywmax

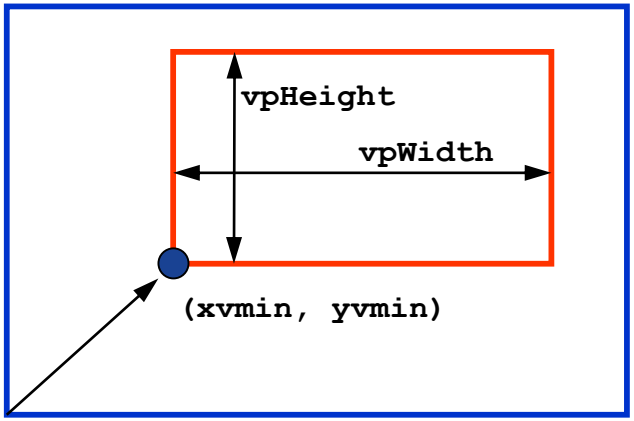
OpenGL 3D Viewing 7

Finally, specify the viewport (just like in 2D):

```
glViewport(xvmin, yvmin, vpWidth, vpHeight);
```

xvmin, yvmin: coordinates lower left corner (in pixel coordinates);

vpWidth, vpHeight: width and height (in pixel coordinates);



OpenGL 2D Viewing 8

In short:

```
glMatrixMode(GL_PROJECTION);
glFrustrum(xwmin, xwmax, ywmin, ywmax, dnear, dfar);
glViewport(xvmin, yvmin, vpWidth, vpHeight);
glMatrixMode(GL_MODELVIEW);
gluLookAt(x0,y0,z0, xref,yref,zref, Vx,Vy,Vz);
```

To prevent distortion, make sure that:

```
(ywmax – ywmin)/(xwmax – xwmin) = vpWidth/vpHeight
```

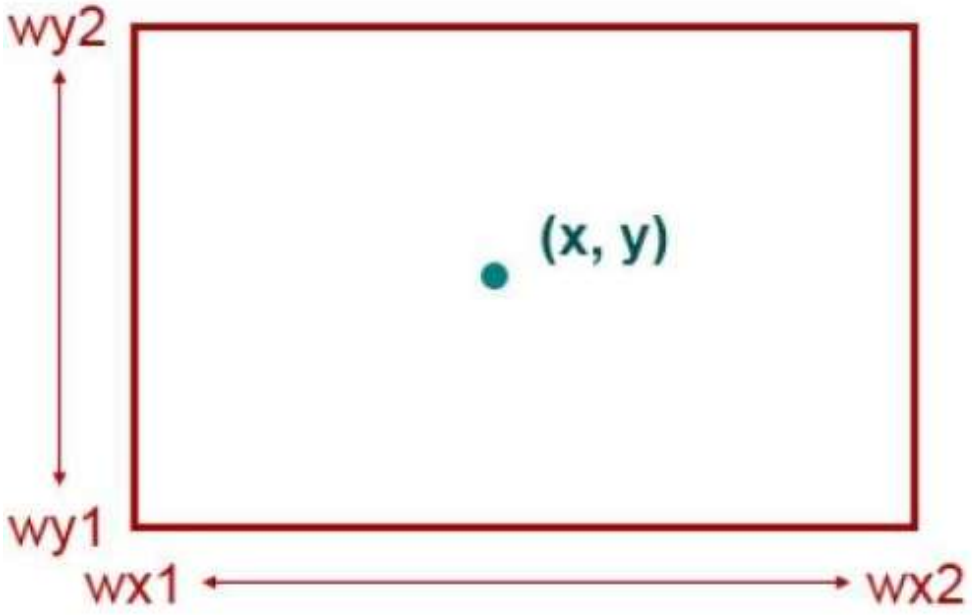
Make sure that you can deal with resize/reshape of the (OS) window.

Clipping

- The primary use of clipping in computer graphics is to remove objects, lines, or line segments that are outside the viewing pane.
- The viewing transformation is insensitive to the position of points relative to the viewing volume − especially those points behind the viewer − and it is necessary to remove these points before generating the view.

Point Clipping

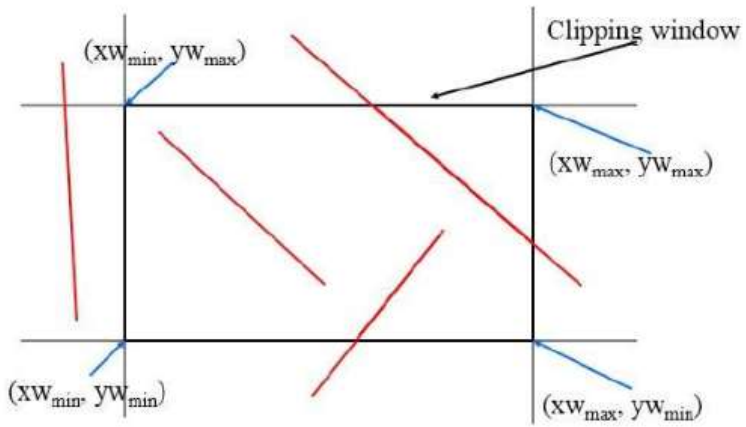
- The X-coordinate of the given point is inside the window, if X lies in between Wx1 ≤ X ≤ Wx2. Same way, Y coordinate of the given point is inside the window, if Y lies in between Wy1 ≤ Y ≤ Wy2.

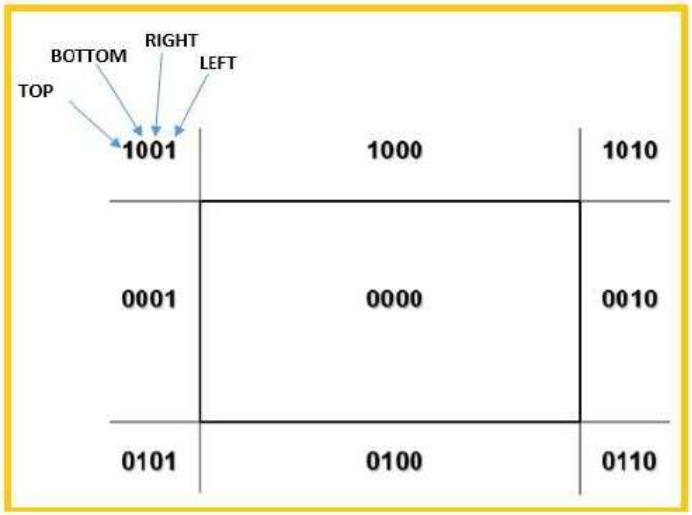


Line Clipping

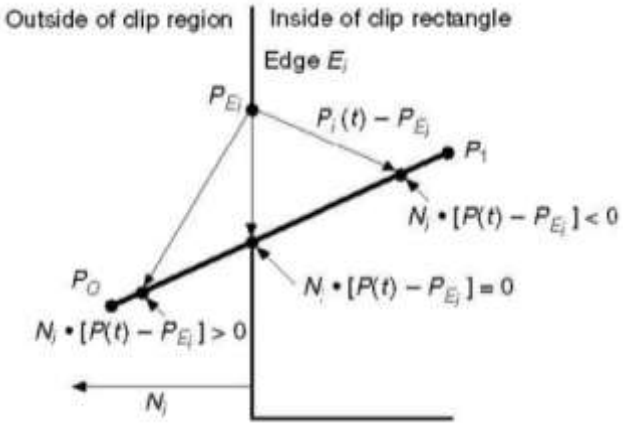
- The concept of line clipping is same as point clipping. In line clipping, we will cut the portion of line which is outside of window and keep only the portion that is inside the window.

Cohen-Sutherland Line Clippings

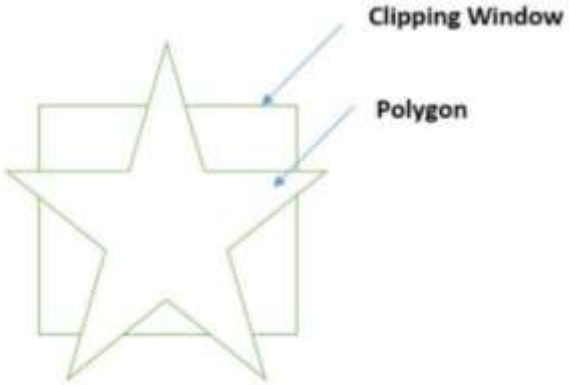




Cyrus-Beck Line Clipping Algorithm



Polygon Clipping SutherlandHodgman Algorithm



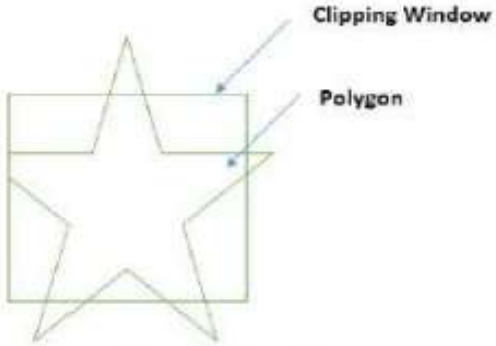


Figure: Clipping Left Edge

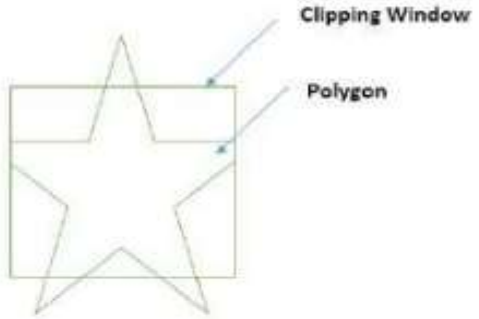


Figure: Clipping Right Edge

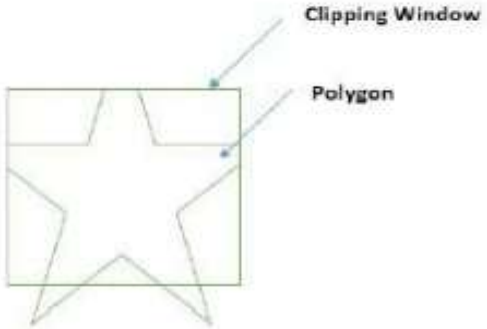


Figure: Clipping Top Edge

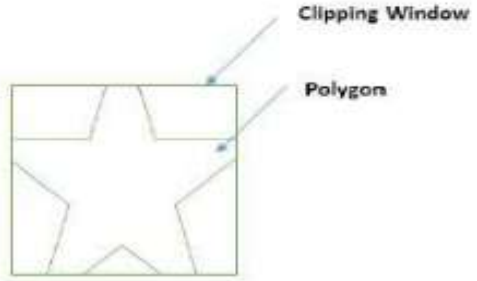
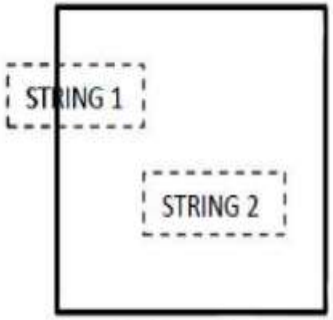


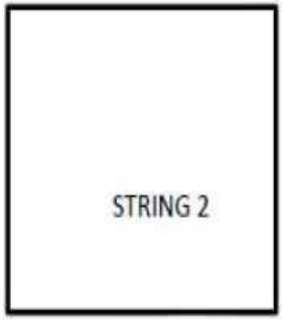
Figure: Clipping Bottom Edge

Text Clipping

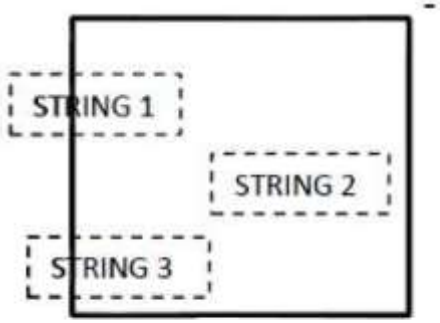
- Various techniques are used to provide text clipping in a computer graphics. It depends on the methods used to generate characters and the requirements of a particular application. There are three methods for text clipping which are listed below −
- All or none string clipping
- All or none character clipping
- Text clipping



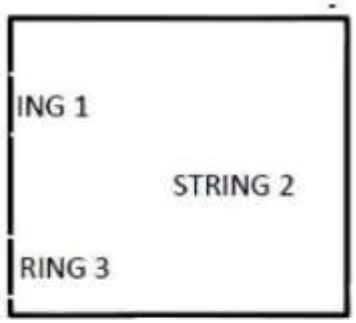
Before Clipping



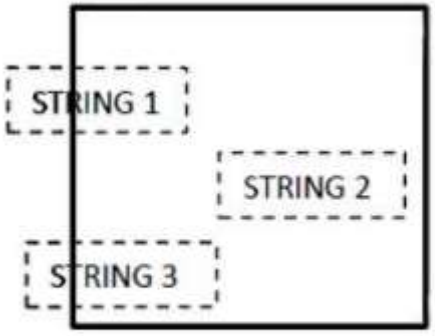
After Clipping



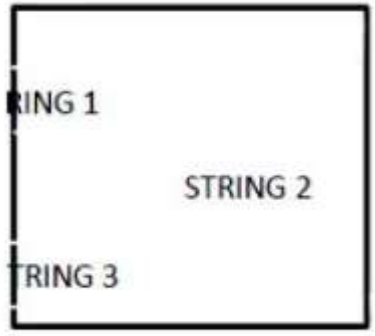
Before Clipping



After Clipping



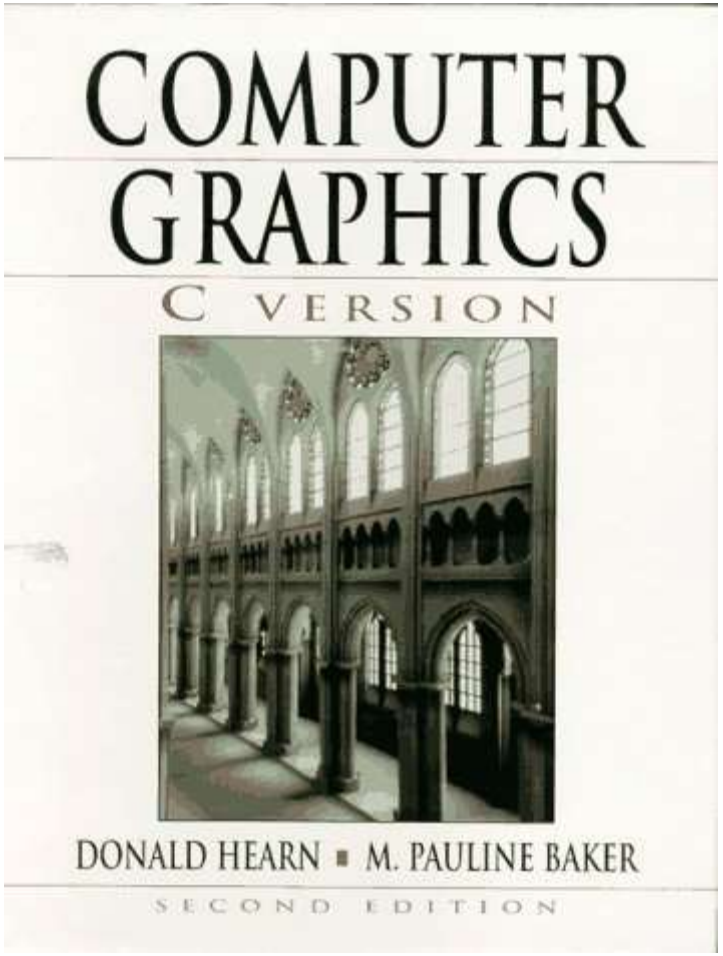
Before Clipping



After Clipping

Textbook

- Computer Graphics
  C Version
  - D. Hearn and M. P. Baker
  - 2nd Edition
  - PRENTICE HALL



Thank you

The Content in this Material are from the Textbooks and Reference books given in the Syllabus