

Introduction to Computer Graphics

Lenin Laitonjam

NIT Mizoram

What is Computer Graphics?

Definition

- Producing pictures or images using a computer

➤ Imaging

- Representing 2D images

➤ Modeling

- Representing 3D objects

➤ Rendering

- Constructing 2D images from 3D models

➤ Animation

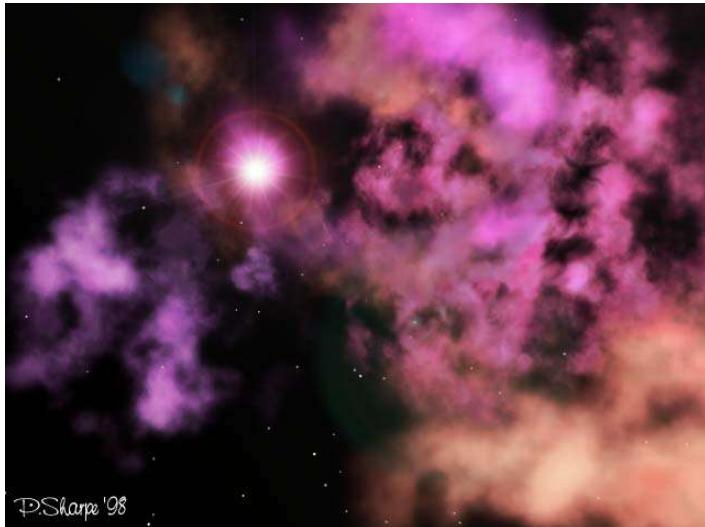
- Simulating changes over time

Applications

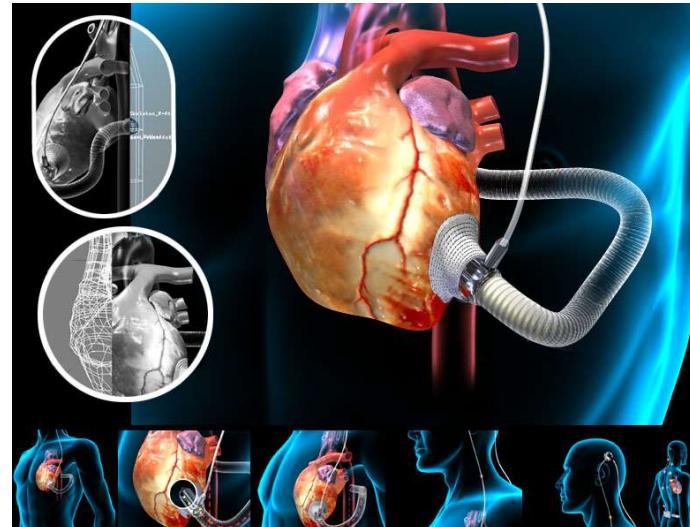
- Display of Information
- Design
- Simulation
- Computer Art
- Entertainment

Display of Information

- Graphics for Scientific, Engineering, and Medical Data



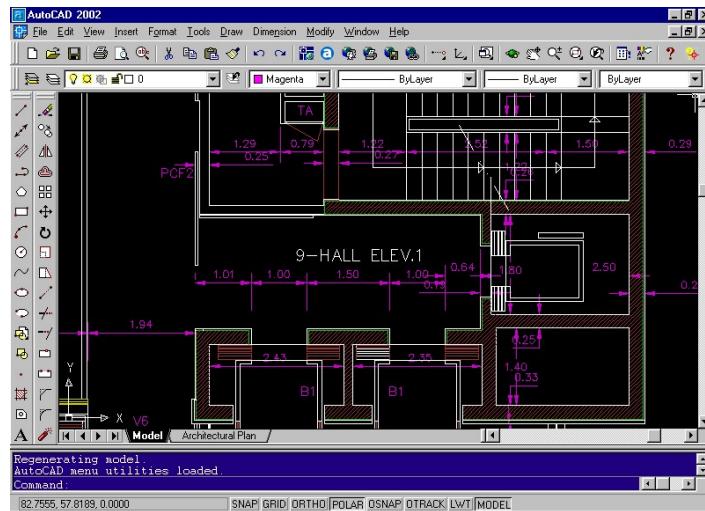
Nebula



Medical Image

Design

- Graphics for Engineering and Architectural System
- Design of Building, Automobile, Aircraft, Machine etc.



AutoCAD 2002



Interior Design

Simulation

- Computer-Generated Models of Physical, Financial and Economic Systems for Educational Aids



Flight Simulator



Mars Rover Simulator

Computer Art

- Graphics for Artist



Metacreation Painter

Entertainment

- Graphics for Movie, Game, VR etc.



Final Fantasy



Online Game

Why computer graphics?

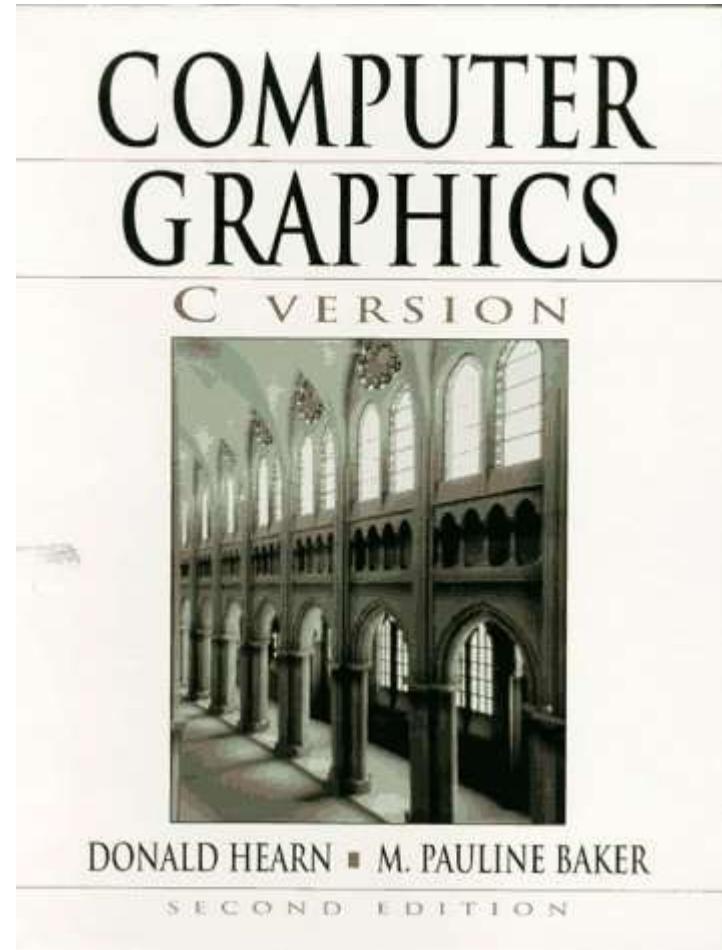
- Fun! Visible!
- Everywhere
- Visual system offers:
 - Parallel input
 - Parallel processing
- Computer graphics: ideal for human-computer communication

What's Our Scope?

- Not a Tutorial on Commercial Software
 - 3DMax, Maya, Photoshop, etc.
- Not about Graphics Business
 - 3D online-game, E-commerce, etc.
- Graphics = Algorithm for Visual Simulation
 - Imaging, Modeling, Rendering, Animation

Textbook

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



Term Projects

- What?
 - Create your own graphics art
 - Not still images but moving pictures
- When?
 - Proposal – after midterm exam
 - Demonstration – after final exam
- How?
 - 1 or 2 person(s) / 1 team
 - Using your assignments

Thank You

Mathematics for Computer Graphics

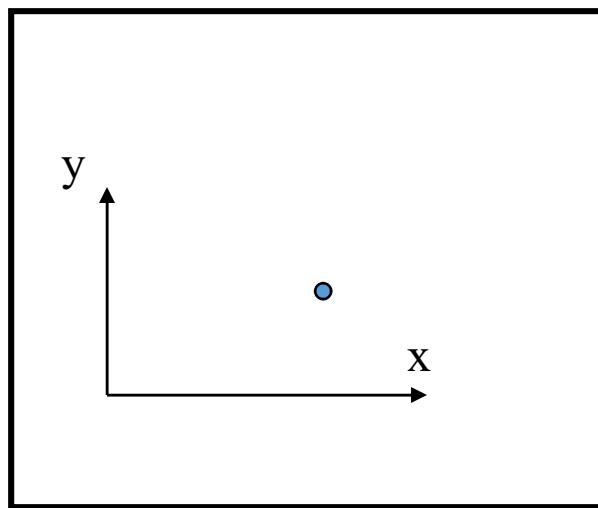
Lenin Laitonjam
NIT Mizoram

Contents

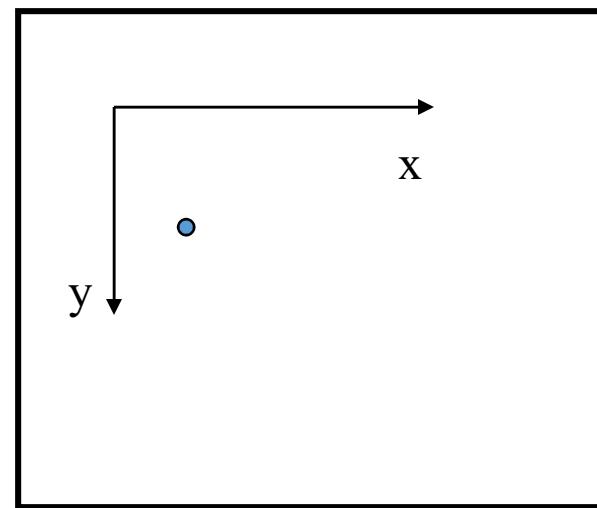
- **Coordinate-Reference Frames**
 - 2D Cartesian Reference Frames / Polar Coordinates
 - 3D Cartesian Reference Frames / Curvilinear Coordinates
- **Points and Vectors**
 - Vector Addition and Scalar Multiplication
 - Scalar Product / Vector Product
- **Basis Vectors and the Metric Tensor**
 - Orthonormal Basis
 - Metric Tensor
- **Matrices**
 - Scalar Multiplication and Matrix Addition
 - Matrix Multiplication / Transpose
 - Determinant of a Matrix / Matrix Inverse

2D Cartesian Reference System

- **2D Cartesian Reference Frames**



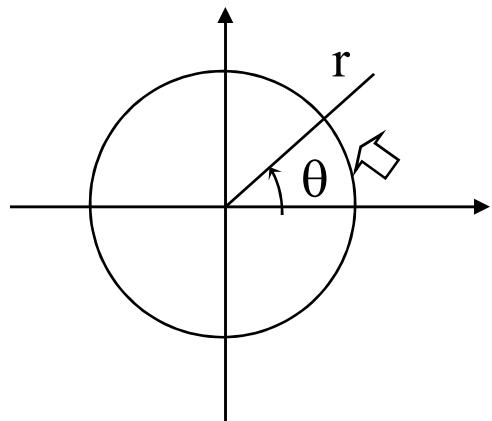
Coordinate origin at the
lower-left screen corner



Coordinate origin in the
upper-left screen corner

Polar Coordinates

- Non-Cartesian System



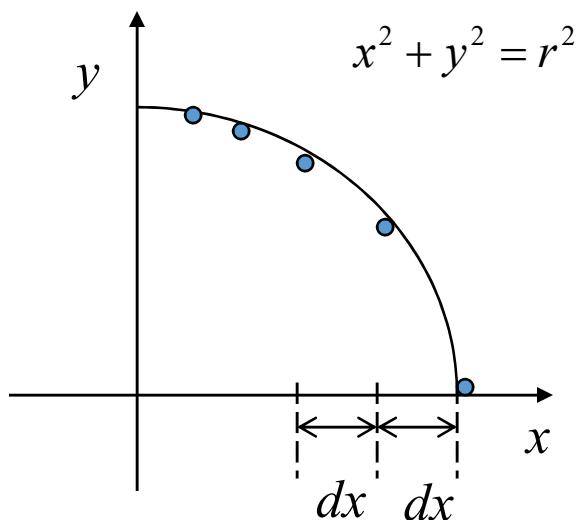
$$x = r \cos \theta, \quad y = r \sin \theta$$
$$r = \sqrt{x^2 + y^2}, \quad \theta = \tan^{-1} \left(\frac{y}{x} \right)$$
$$s = r\theta$$

Why Polar Coordinates?

- **Circle**

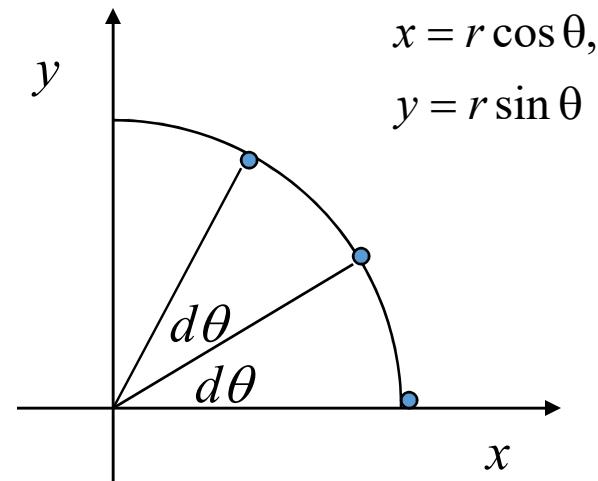
- 2D Cartesian \rightarrow Polar Coordinate

Cartesian Coordinates



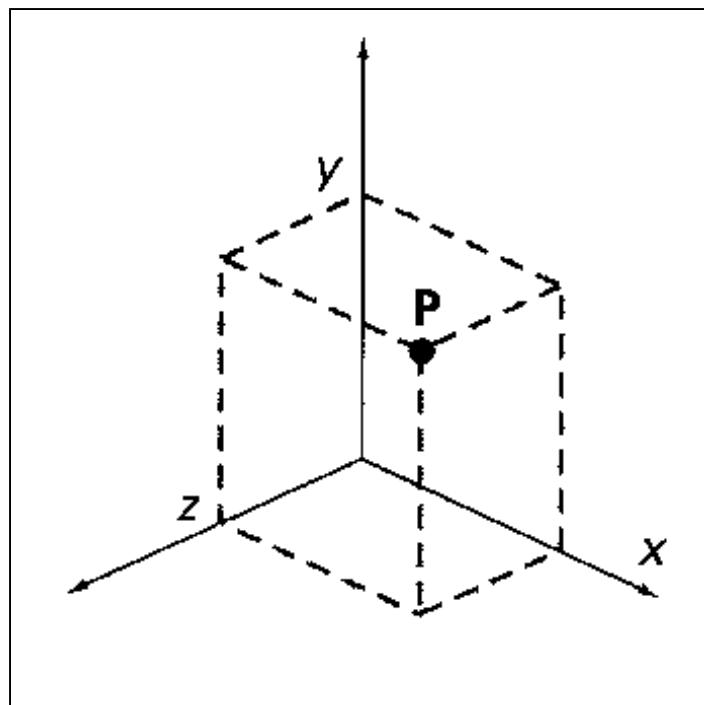
Points not evenly distributed

Polar Coordinates



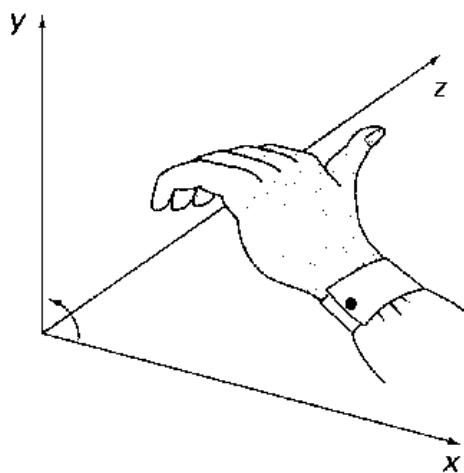
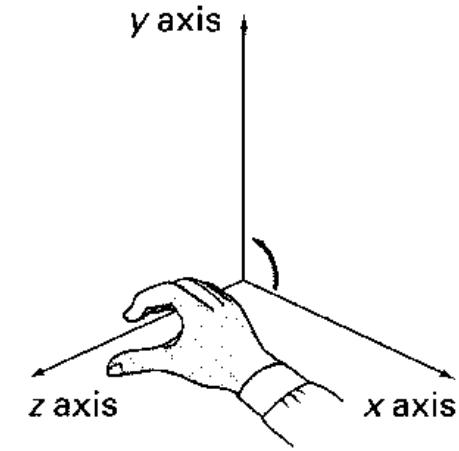
Maintain a certain distance between consecutive points

3D Cartesian Reference Frames



Three Dimensional Point

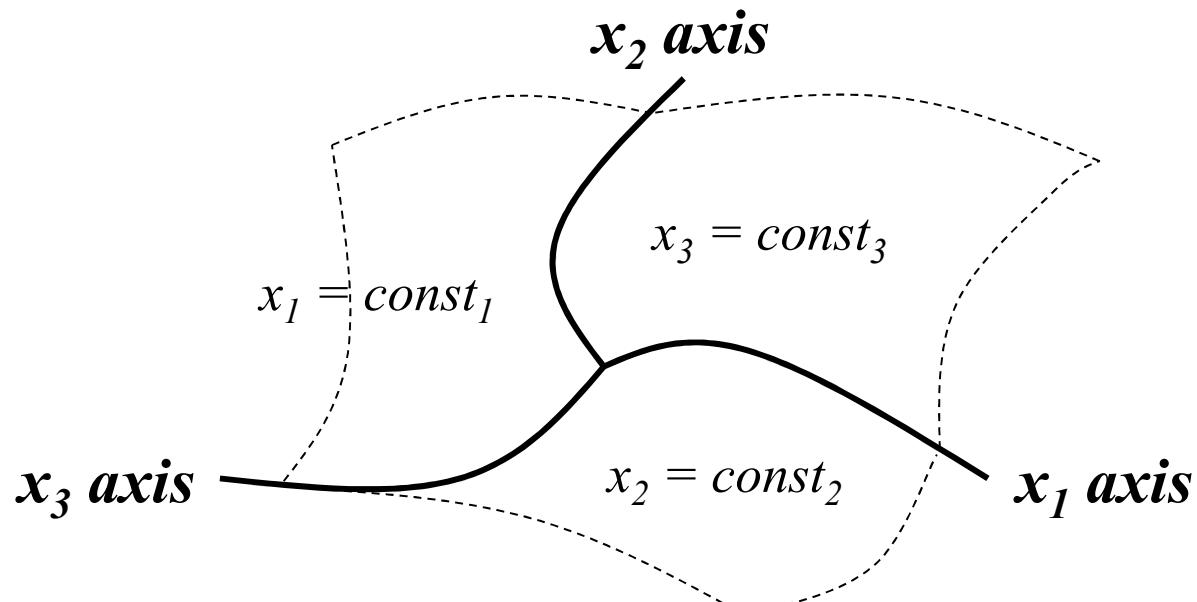
3D Cartesian Reference Frames



- **Right hand coordinate system**
 - Standard in most graphics packages
- **Left hand coordinate system**
 - Video Monitor's coordinate system

3D Curvilinear Coordinate Systems

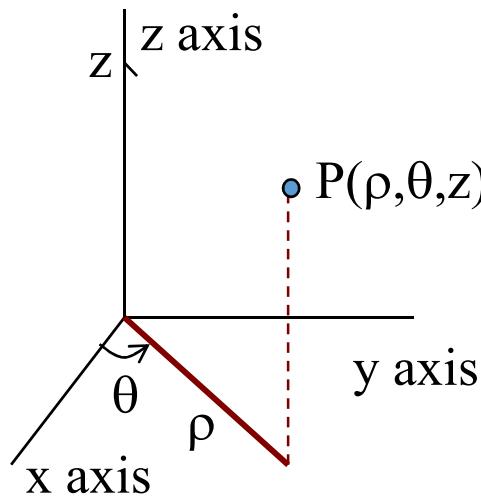
- **General Curvilinear Reference Frame**
 - Orthogonal coordinate system
 - Each coordinate surfaces intersects at right angles



A general Curvilinear coordinate reference frame

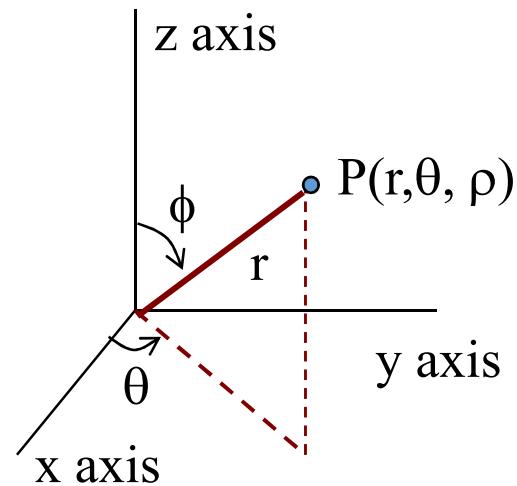
3D Non-Cartesian System

- **Cylindrical Coordinates**



$$\begin{aligned}x &= \rho \cos \theta \\y &= \rho \sin \theta \\z &= z\end{aligned}$$

- **Spherical Coordinates**

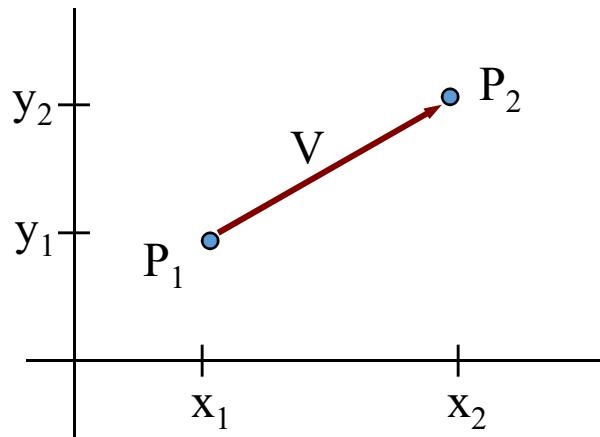


$$\begin{aligned}x &= r \cos \theta \sin \phi \\y &= r \sin \theta \sin \phi \\z &= r \cos \phi\end{aligned}$$

Points and Vectors

- **Point:** Indicates a position.
- **Vector:** Defined the difference between two positions. Has magnitude and direction.

$$V = P_2 - P_1 = (x_2 - x_1, y_2 - y_1) = (V_x, V_y)$$



$$|V| = \sqrt{V_x^2 + V_y^2}$$

$$\alpha = \tan^{-1} \left(\frac{V_y}{V_x} \right)$$

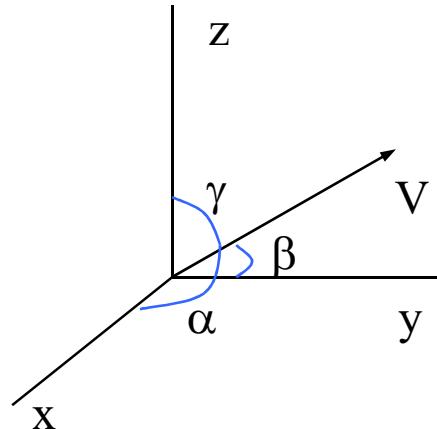
Vectors

- **Vector in 3D**

$$|V| = \sqrt{V_x^2 + V_y^2 + V_z^2}$$

$$\cos\alpha = \frac{V_x}{|V|}, \quad \cos\beta = \frac{V_y}{|V|}, \quad \cos\gamma = \frac{V_z}{|V|}$$

$$\cos^2 \alpha + \cos^2 \beta + \cos^2 \gamma = 1$$



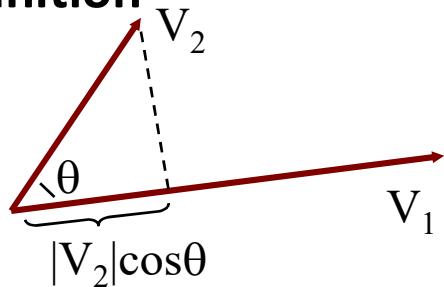
- **Vector Addition and Scalar Multiplication**

$$V_1 + V_2 = (V_{1x} + V_{2x}, V_{1y} + V_{2y}, V_{1z} + V_{2z})$$

$$\alpha V = (\alpha V_x, \alpha V_y, \alpha V_z)$$

Scalar Product/Dot Product/Inner Product

- **Definition**



$$V_1 \cdot V_2 = |V_1| |V_2| \cos \theta, \quad 0 \leq \theta \leq \pi$$

- **For Cartesian Reference Frame**

$$V_1 \cdot V_2 = V_{1x}V_{2x} + V_{1y}V_{2y} + V_{1z}V_{2z}$$

- **Properties**

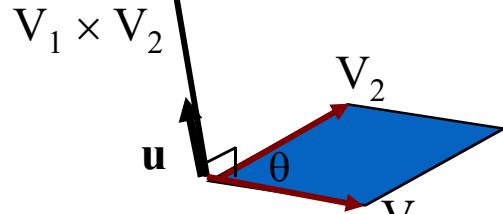
- Commutative
- Distributive

$$V_1 \cdot V_2 = V_2 \cdot V_1$$

$$V_1 \cdot (V_2 + V_3) = V_1 \cdot V_2 + V_1 \cdot V_3$$

Vector Product/Cross Product/ Outer Product

- **Definition**



$$V_1 \times V_2 = \mathbf{u} |V_1| |V_2| \sin \theta, \quad 0 \leq \theta \leq \pi$$

- **For Cartesian Reference Frame**

$$V_1 \times V_2 = (V_{1y}V_{2z} - V_{1z}V_{2y}, V_{1z}V_{2x} - V_{1x}V_{2z}, V_{1x}V_{2y} - V_{1y}V_{2x})$$

- **Properties**

- AntiCommutative
- Not Associative
- Distributive

$$V_1 \times V_2 = -(V_2 \times V_1)$$

$$V_1 \times (V_2 \times V_3) \neq (V_1 \times V_2) \times V_3$$

$$V_1 \times (V_2 + V_3) = (V_1 \times V_2) + (V_1 \times V_3)$$

Matrices

- **Definition**
 - A rectangular array of quantities

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

- **Scalar Multiplication and Matrix Addition**

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \xrightarrow{\hspace{1cm}} A + B = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{pmatrix}$$
$$kA = \begin{pmatrix} ka_{11} & ka_{12} \\ ka_{21} & ka_{22} \end{pmatrix}$$

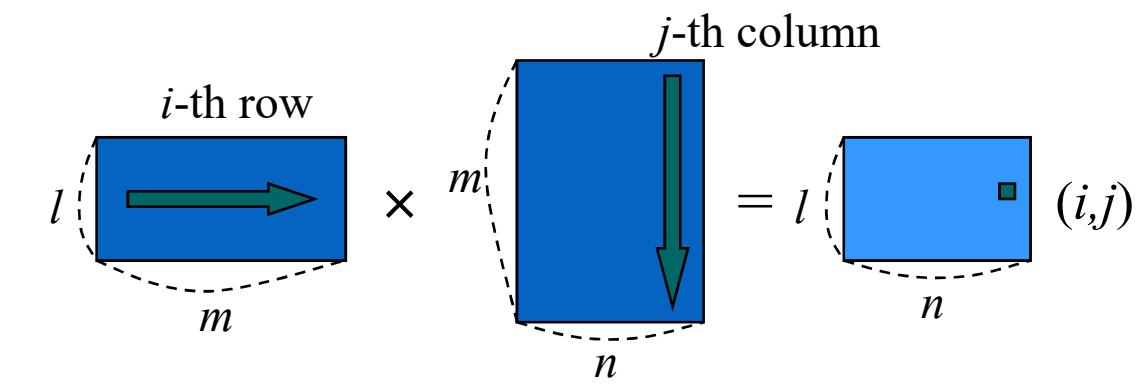
Matrix Multiplication

- **Definition**

$$C = AB$$

\Downarrow

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$



- **Properties**

- Not Commutative
- Associative
- Distributive
- Scalar Multiplication

$$AB \neq BA$$

$$(AB)C = A(BC)$$

$$A(B + C) = AB + BC$$

$$(kA)B = A(kB) = k(AB)$$

Matrix Transpose

- **Definition**

- Interchanging rows and columns

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}, \quad [a \ b \ c]^T = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

- **Transpose of Matrix Product**

$$(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T$$

Determinant of Matrix

- **Definition**
 - For a square matrix, combining the matrix elements to product a single number
- **2×2 matrix**
- **Determinant of $n \times n$ Matrix A ($n \geq 2$)**

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

$$\det \mathbf{A} = \sum_{j=1}^n (-1)^{j+k} a_{jk} \det \mathbf{A}_{jk}$$

Inverse Matrix

- **Definition**

$$AA^{-1} = I \quad A^{-1}A = I$$

- Non-singular matrix

- If and only if the determinant of the matrix is non-zero

- **2×2 matrix**

$$\bullet \text{ Properties} \quad A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \rightarrow \quad A^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

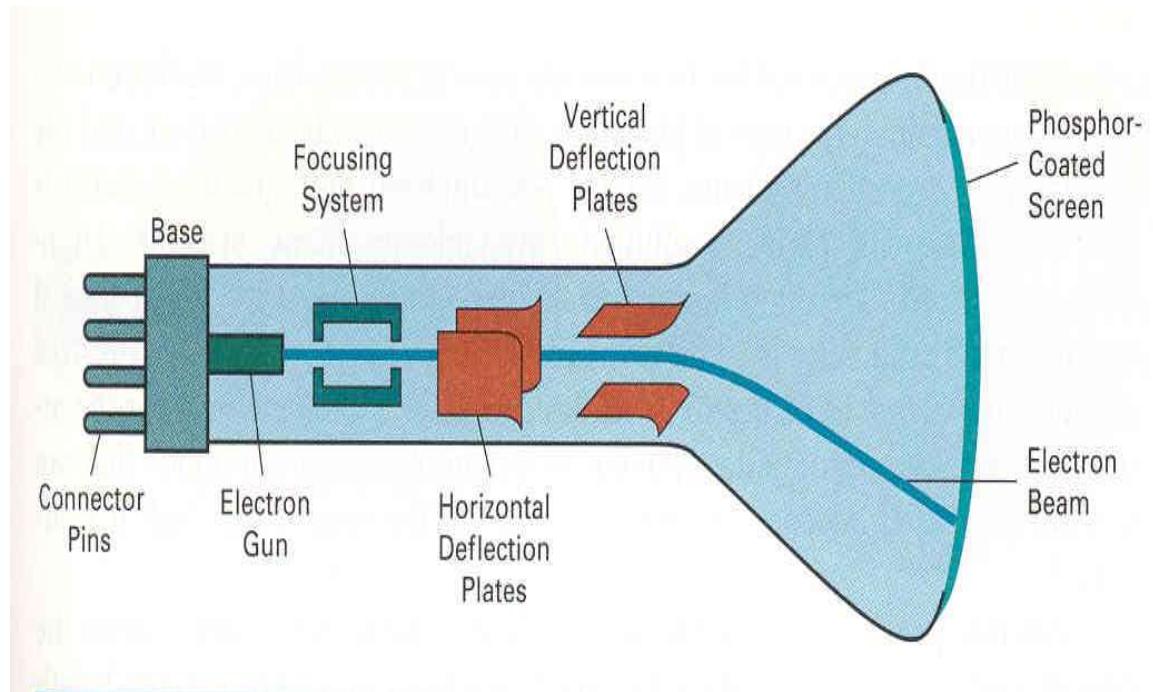
$$(A^{-1})^{-1} = A \quad (AB)^{-1} = B^{-1}A^{-1} \quad (A^T)^{-1} = (A^{-1})^T$$

Thank You

Display Systems

Lenin Laitonjam
NIT Mizoram

Cathode Ray Tube (CRT)



Important Terms

- *Phosphor's Fluorescence* is the light emitted as electrons (unstable) lose their excess energy while the phosphor is being struck by electrons.
- *Phosphorescence* is the light given off by the return of the relatively more stable excited electrons to their unexcited state once the electron beam excitation is removed.
- *Phosphor's persistence* is defined as the time from the removal of excitation to the moment when phosphorescence has decayed to 10% of the initial light output.

Types of CRT Display Devices

1. Direct View Storage Tube (DVST)
2. Calligraphic or Random Scan Display System.
3. Refresh and Raster Scan Display System.

DVST

- CRT with long persistence phosphor.
- Provide flicker-free display.
- Slow moving electron draws a line on the screen.
- Image is stored as a distribution of charges on the inside of the screen.
- Limited interactive support.
- No animation possible with DVST.

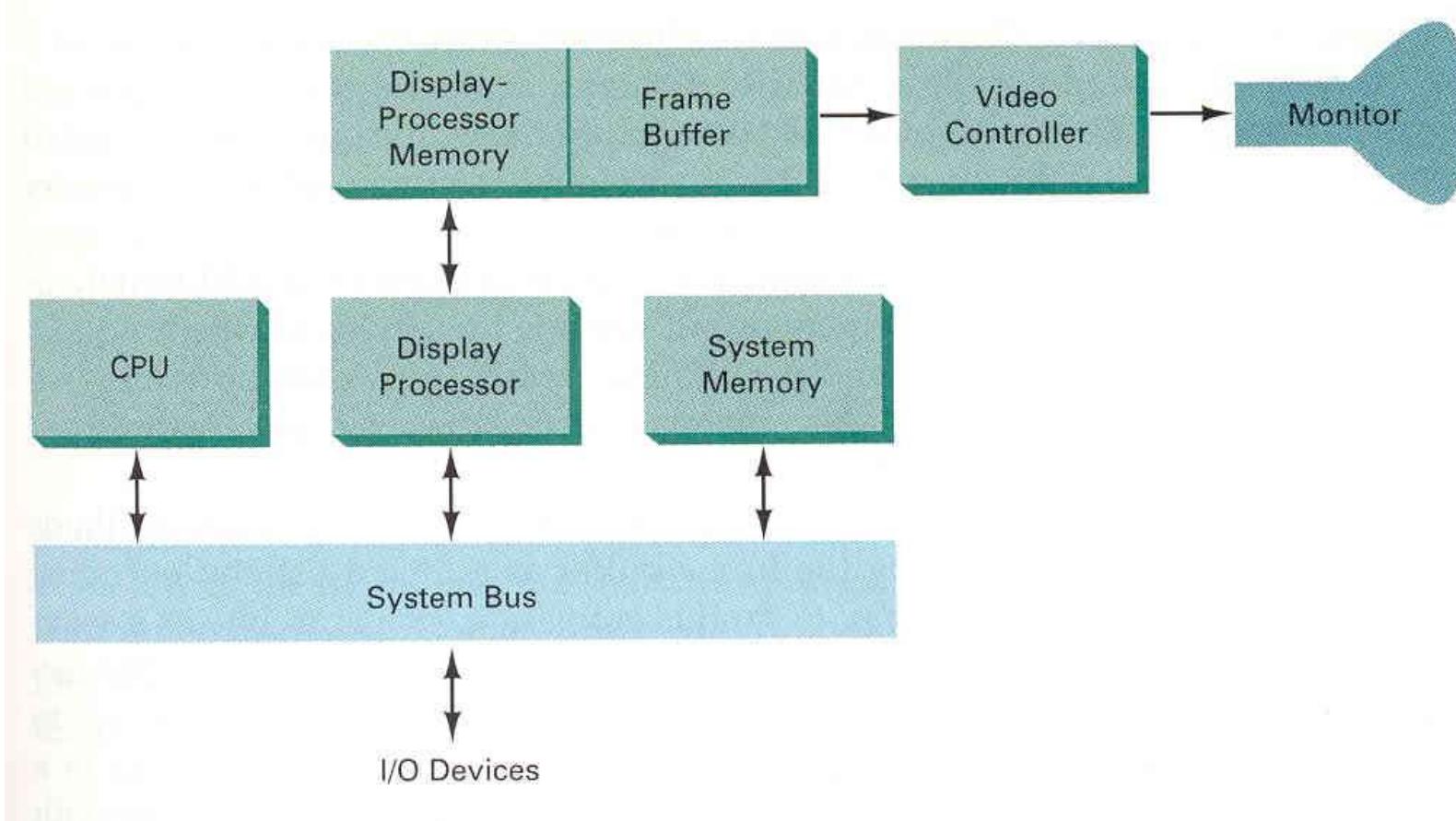
Calligraphic or Random Scan display system

- Also called Vector, Stroke or Line drawing displays
- Characters are also made by sequence of short lines.
- Electron beam is deflected from end-point to end-point.
- Order of deflection is dictated by the arbitrary of the display commands
- Phosphor has short persistence (10 – 100 μ sec)
- Minimum refresh required is 30 Hz (fps) for flicker-free display
- Refresh Buffer is allocated for storing Display List/Display Program
- Little scope for animation

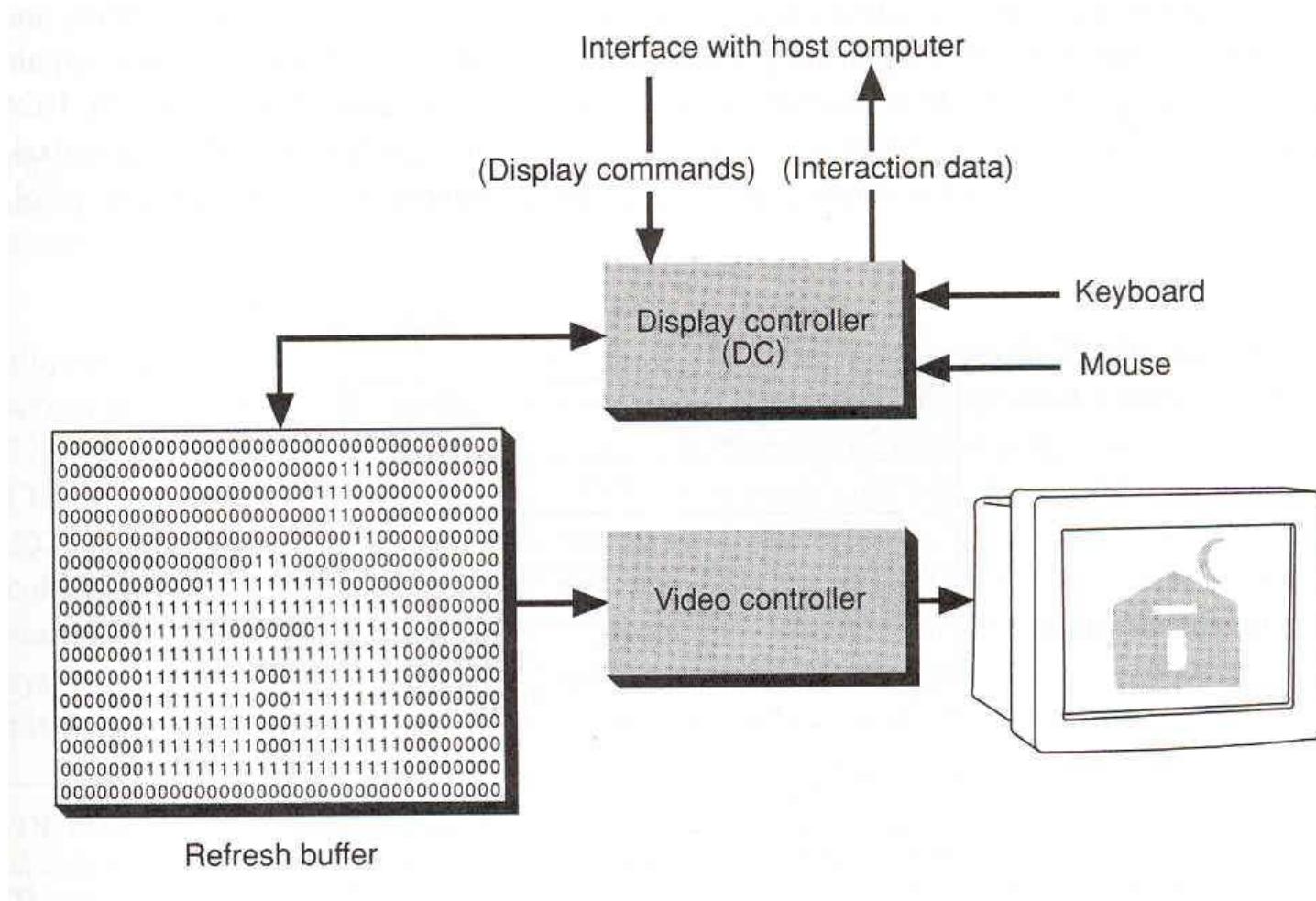
Refresh and Raster Scan Display System

- It is a point-plotting device.
- Raster system store the display primitives (lines, character, shaded and patterned areas) is a display buffer.
- Refresh buffer (frame buffer/bit-plane) stores the drawing primitives in terms of points and pixels components.
- Entire screen is a matrix of pixels
- Each pixel brightness can be controlled
- Frame buffer can be seen as a set of horizontal raster lines or rows of individual pixels
- Each point is addressable

Raster Graphics

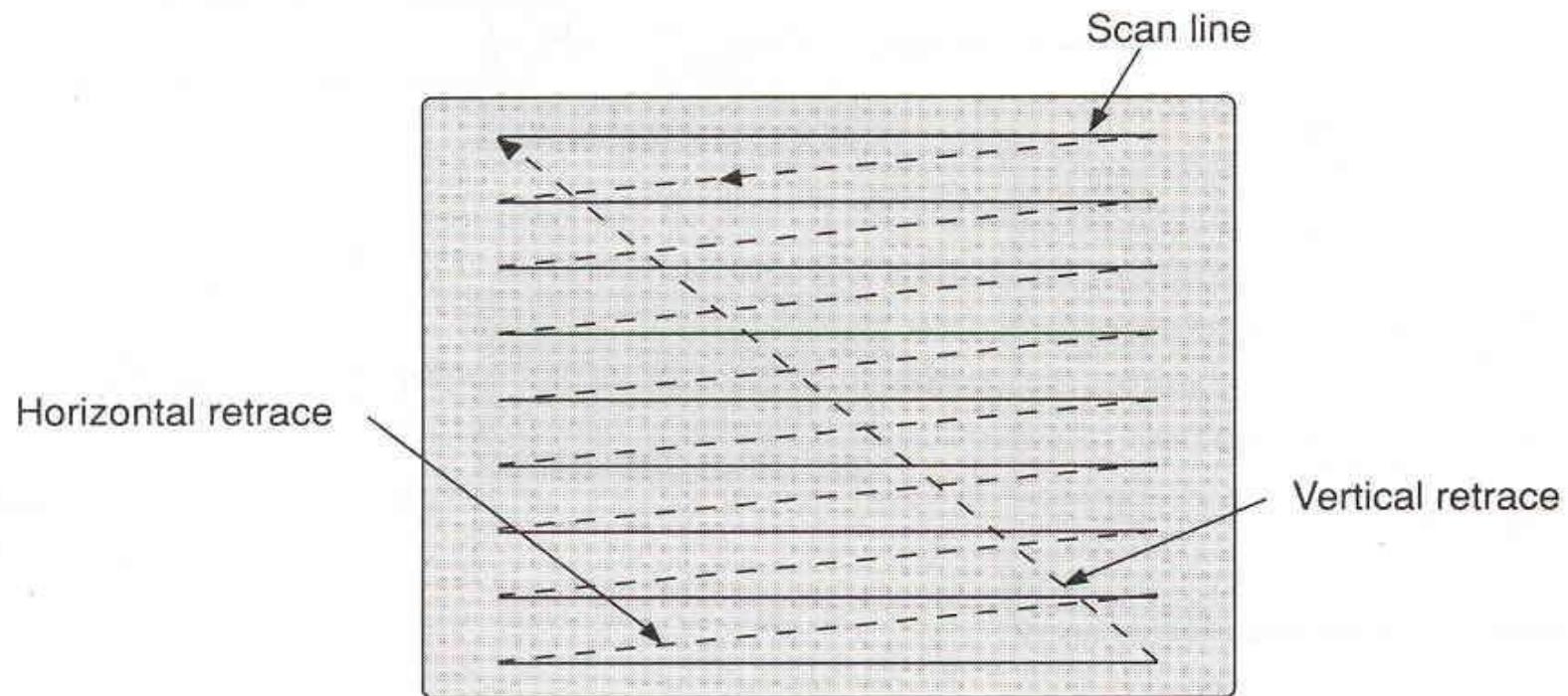


Frame Buffer

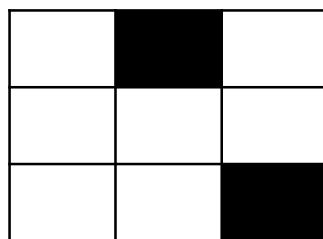


Frame Buffer Refresh

- Refresh Rate
 - Usually 30~75 Hz



Color Frame Buffer (n-bit plane)



255	150	75	0
255	150	75	0
255	150	75	0
255	150	75	0
255	150	75	0
255	150	75	0
255	150	75	0
255	150	75	0

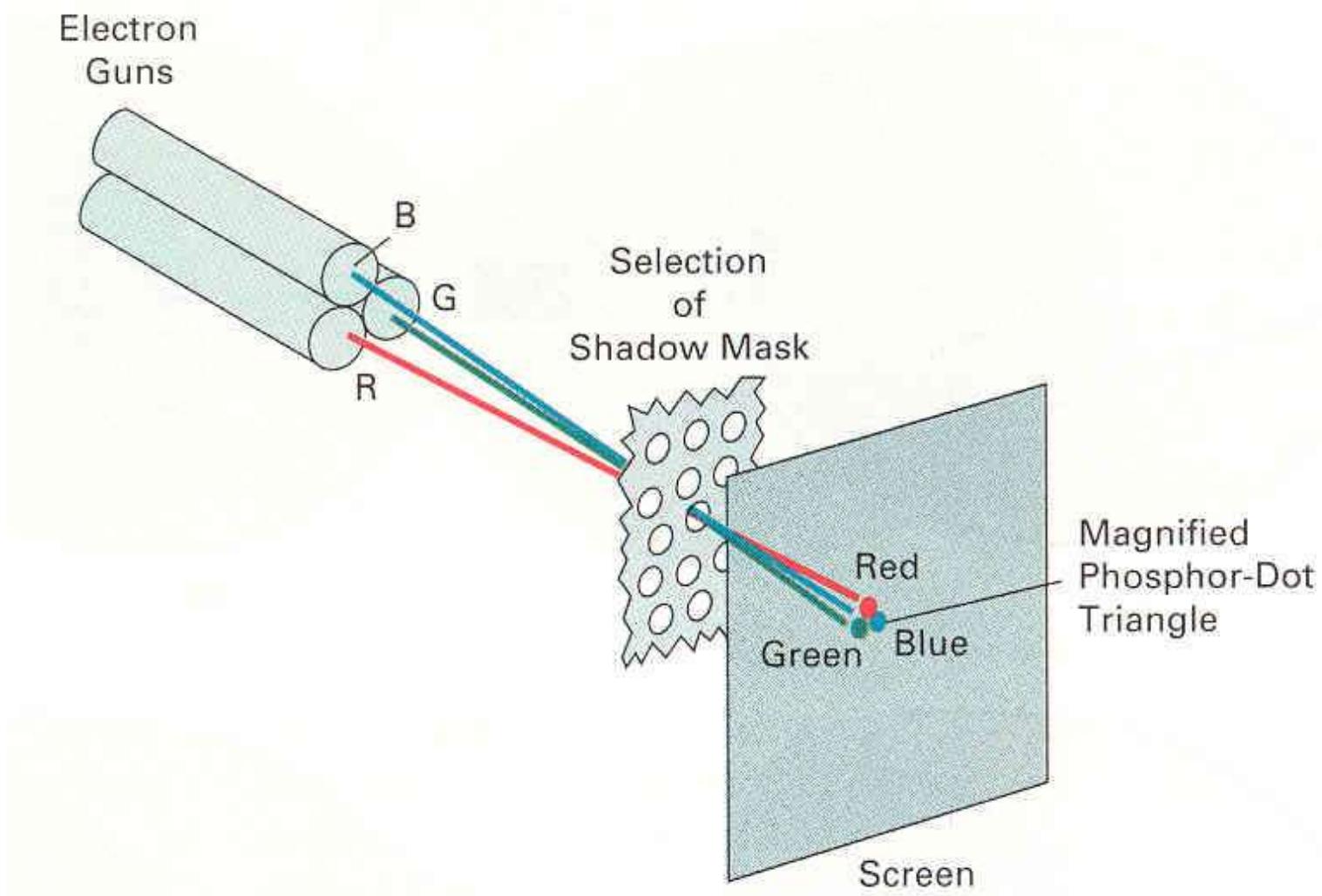
Methods for generating Colors

1. Beam Penetration Method:

- a. Use two phosphor layer (red and green).
- b. Beam with high intensity penetrates through green layer and excites red layer.
- c. Slow moving electrons excites outer red layer.

2. Shadow Mask Method

Shadow Mask Method



Thank You

Rendering – 1

(Scan Conversion of Line and Circle)

Lenin Laitonjam

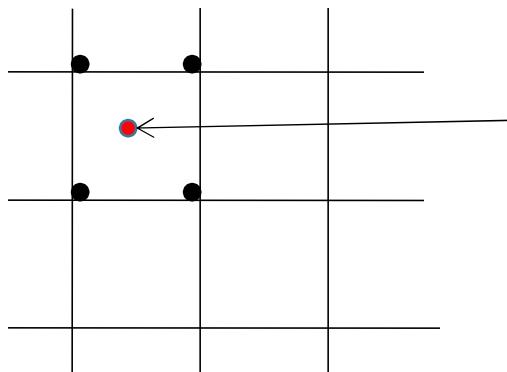
NIT Mizoram

Rendering

- To draw something on the screen, we need to consider pixel grid
 - Discreet space
- We need to map the representations from continuous to discreet space
- The mapping process is called rendering
 - Also called *scan conversion/rasterization*
- We will have a look at scan conversion of
 - Point
 - Line
 - Circle

Point Scan Conversion

- Trivial: simply round off to the nearest pixel position



The point can be mapped to any of the four pixel positions, depending on the coordinate value
e.g. (1.1, 2.6) maps to (1, 3)

Line Scan Conversion

- A line segment is defined by the coordinate positions of the line end-points
- What happens when we try to draw this on a pixel based display?
 - How to choose the correct pixels

Line Equation

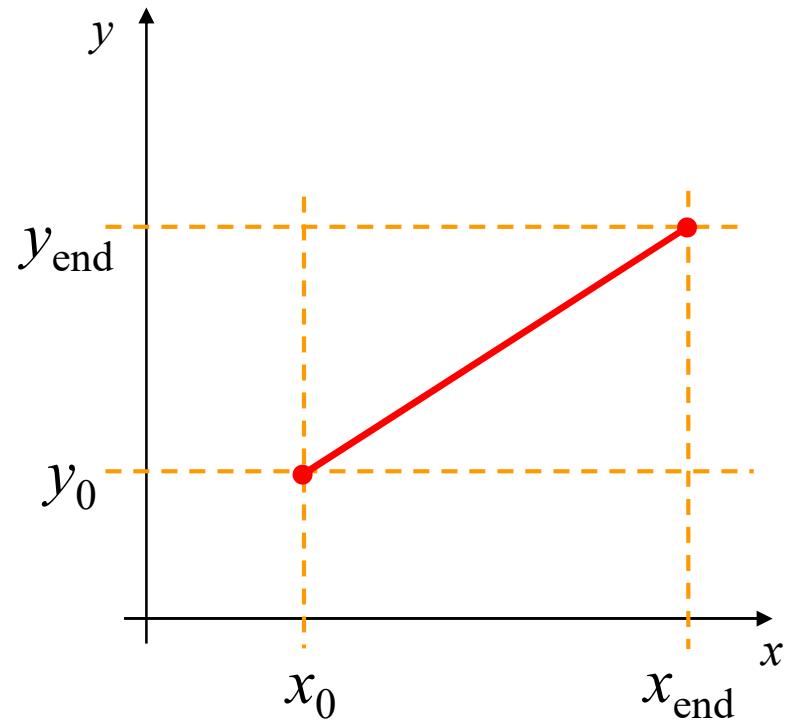
- Slope-intercept line equation

$$y = m \cdot x + b$$

where

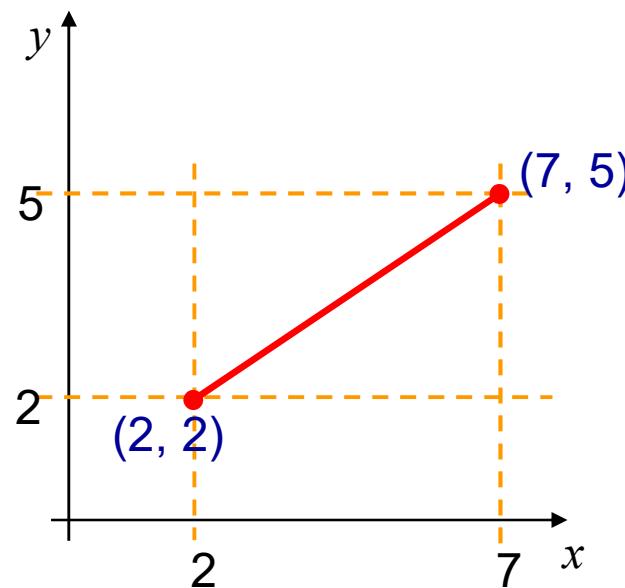
$$m = \frac{y_{end} - y_0}{x_{end} - x_0}$$

$$b = y_0 - m \cdot x_0$$

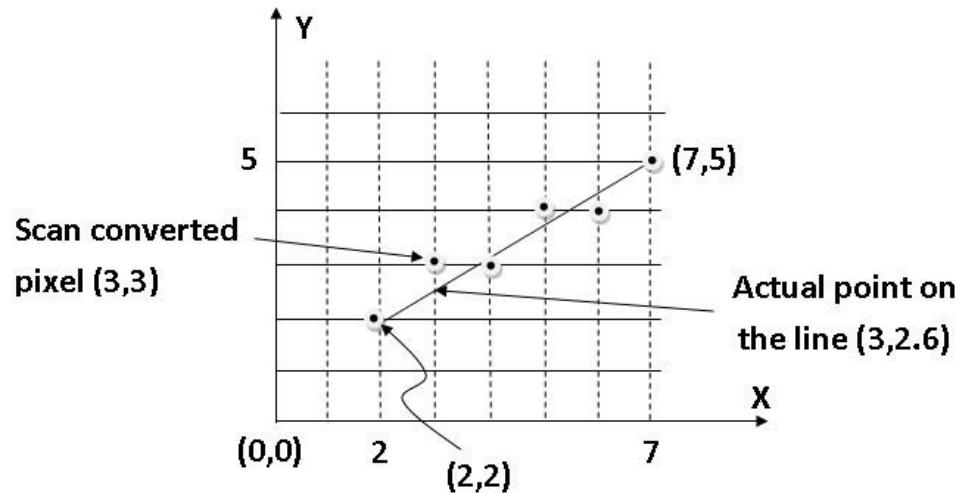


A Very Simple Solution

- Simply work out the corresponding y coordinate for each unit x coordinate
- Let's consider the following example



A Very Simple Solution



- First work out m and b :

$$m = \frac{5 - 2}{7 - 2} = \frac{3}{5}$$

$$b = 2 - \frac{3}{5} * 2 = \frac{4}{5}$$

- Now for each x value work out the y value

$$y(3) = \frac{3}{5} \cdot 3 + \frac{4}{5} = 2\frac{3}{5}$$

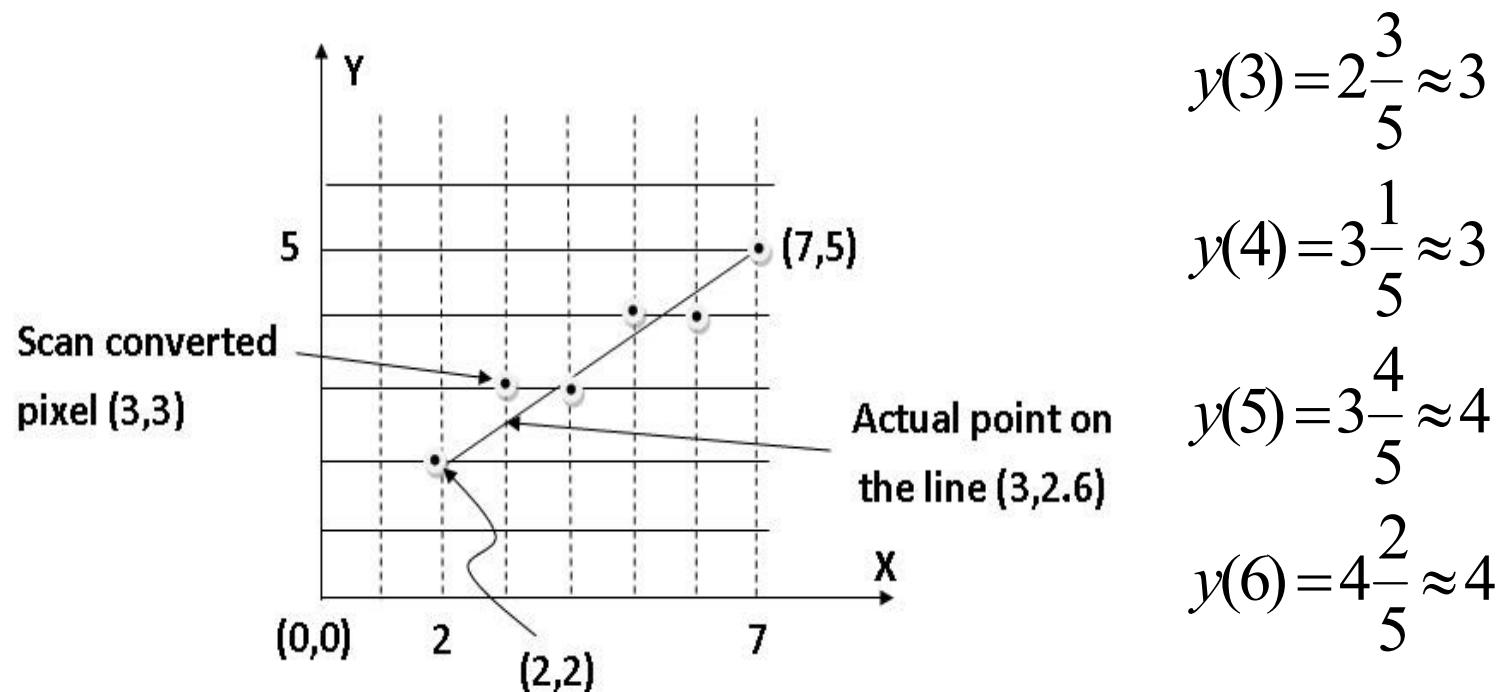
$$y(5) = \frac{3}{5} \cdot 5 + \frac{4}{5} = 3\frac{4}{5}$$

$$y(4) = \frac{3}{5} \cdot 4 + \frac{4}{5} = 3\frac{1}{5}$$

$$y(6) = \frac{3}{5} \cdot 6 + \frac{4}{5} = 4\frac{2}{5}$$

A Very Simple Solution

- Now just round off the results and turn on these pixels to draw our line



A Very Simple Solution

- However, this approach is just way too slow
- In particular,
 - The equation $y = mx + b$ requires the multiplication of m by x
 - Rounding off the resulting y coordinates
- We need a faster solution

A Quick Note About Slopes

- In the previous example, we chose to solve the line equation to get y coordinate for each x coordinate
- What if we had done it the other way around?
- So this gives us: $x = \frac{y - b}{m}$
where: $m = \frac{y_{end} - y_0}{x_{end} - x_0}$ and $b = y_0 - m \cdot x_0$

A Quick Note About Slopes

- Leaving out the details this gives us:

$$x(3) = 3\frac{2}{3} \approx 4 \quad x(4) = 5\frac{1}{3} \approx 5$$

- We can see easily that this line doesn't look very good!
- We choose which way to work out the line pixels based on the slope of the line
 - If the slope of a line is between -1 and 1, work out y coordinates based on x coordinates
 - Otherwise do the opposite – x coordinates are computed based on y coordinates

The DDA Algorithm

- The *digital differential analyzer* (DDA) algorithm takes an incremental approach in order to speed up scan conversion
- Consider the list of points that we determined for the line in our previous example:
 $(2, 2), (3, 2\frac{3}{5}), (4, 3\frac{1}{5}), (5, 3\frac{4}{5}), (6, 4\frac{2}{5}), (7, 5)$
- Notice that as the x coordinates go up by one, the y coordinates simply go up by the slope of the line
 - This is the key insight in the DDA algorithm

The DDA Algorithm

- When m is between -1 and 1, begin at the first point in the line and, by incrementing x by 1, calculate the corresponding y as follows

$$y_{k+1} = y_k + m$$

- When m is outside these limits, increment y by 1 and calculate the corresponding x as follows

$$x_{k+1} = x_k + \frac{1}{m}$$

The DDA Algorithm

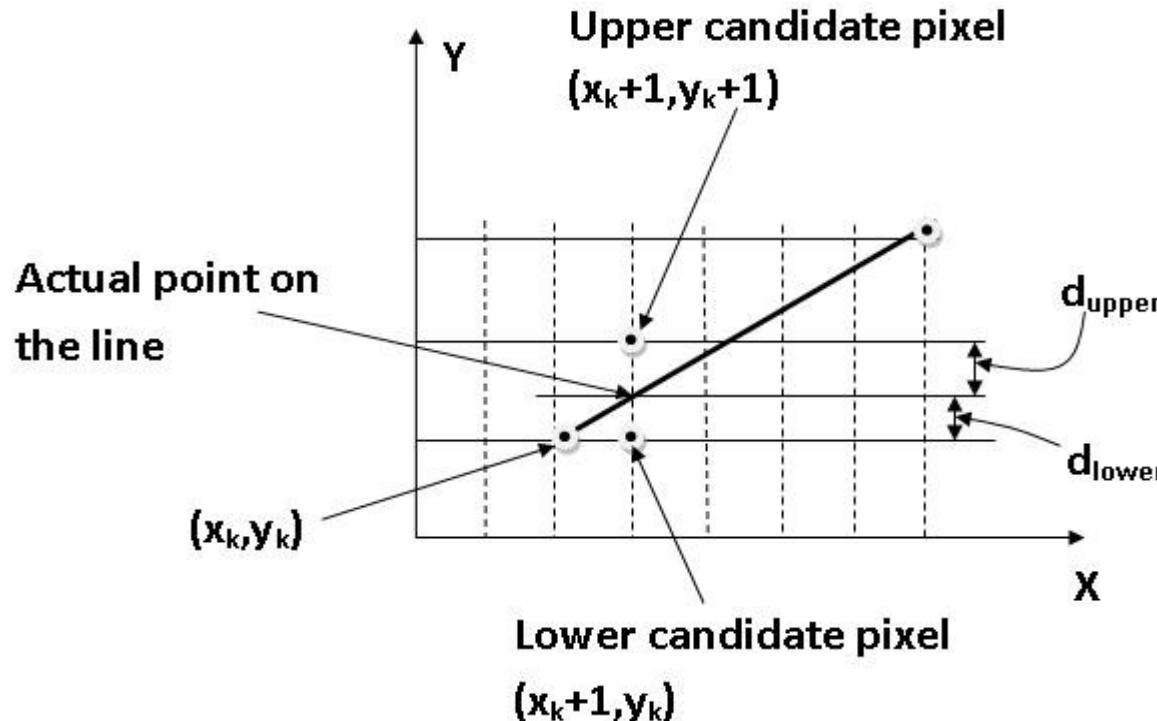
- Again the values calculated by the equations used by the DDA algorithm must be rounded

The DDA Algorithm Summary

- The DDA algorithm is much faster than our previous attempt
 - In particular, there are no longer any multiplications involved
- However, there are still two big issues
 - Accumulation of round-off errors can make the pixelated line drift away from what was intended
 - The rounding operations and floating point arithmetic involved are time-consuming

The Bresenham Line Algorithm

- Move across the x axis in unit intervals and at each step choose between two different y coordinates



Derivation

- At sample position $x_k + 1$ the vertical separations from the mathematical line are labelled d_{upper} and d_{lower}
- The y coordinate on the mathematical line at $x_k + 1$ is:

$$y = m(x_k + 1) + b$$

Derivation

- So, d_{upper} and d_{lower} are given as follows:

$$\begin{aligned}d_{lower} &= y - y_k \\&= m(x_k + 1) + b - y_k\end{aligned}$$

and

$$\begin{aligned}d_{upper} &= (y_k + 1) - y \\&= y_k + 1 - m(x_k + 1) - b\end{aligned}$$

- We can use these to make a simple decision about which pixel is closer to the mathematical line

Derivation

- This simple decision is based on the difference between the two pixel positions:

$$d_{lower} - d_{upper} = 2m(x_k + 1) - 2y_k + 2b - 1$$

- Let's substitute m with $\Delta y / \Delta x$ where Δx and Δy are the differences between the end-points:

$$\begin{aligned}\Delta x(d_{lower} - d_{upper}) &= \Delta x\left(2\frac{\Delta y}{\Delta x}(x_k + 1) - 2y_k + 2b - 1\right) \\ &= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + 2\Delta y + \Delta x(2b - 1) \\ &= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c\end{aligned}$$

Derivation

- So, a decision parameter p_k for the k th step along a line is given by:

$$\begin{aligned} p_k &= \Delta x(d_{lower} - d_{upper}) \\ &= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c \end{aligned}$$

- The sign of the decision parameter p_k is the same as that of $d_{lower} - d_{upper}$
- If p_k is negative, then choose the lower pixel, otherwise choose the upper pixel

Derivation

- Remember coordinate changes occur along the x axis in unit steps, so we can do everything with integer calculations
- At step $k+1$ the decision parameter is given as:

$$p_{k+1} = 2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + c$$

- Subtracting p_k from this we get:

$$p_{k+1} - p_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$$

Derivation

- But, x_{k+1} is the same as $x_k + 1$ so:

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

- where $y_{k+1} - y_k$ is either 0 or 1 depending on the sign of p_k
- The first decision parameter p_0 , evaluated at (x_0, y_0) , is given as:

$$p_0 = 2\Delta y - \Delta x$$

The Bresenham Line Algorithm

1. Input the two line end-points, storing the left end-point in (x_0, y_0)
2. Plot the point (x_0, y_0)
3. Calculate the constants Δx , Δy , $2\Delta y$, and $(2\Delta y - 2\Delta x)$ and get the first value for the decision parameter as:

$$p = 2\Delta y - \Delta x$$

4. At each x_k along the line, starting at $k = 0$, perform the following test. If $p < 0$, the next point to plot is $(x_k + 1, y_k)$ and:

$$p = p + 2\Delta y$$

The Bresenham Line Algorithm

Otherwise, the next point to plot is $(x_k + 1, y_k + 1)$ and:

$$p = p + 2\Delta y - 2\Delta x$$

5. Repeat step 4 until $x < (x_2 - 1)$

- The algorithm and derivation above assumes slopes are less than 1 ($|m| < 1.0$), for other slopes we need to adjust the algorithm slightly

Circle Scan Conversion

- The equation for a circle is:

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

where r is the radius of the circle

- So, we can write a simple circle drawing algorithm by solving the equation for y at unit x intervals using:

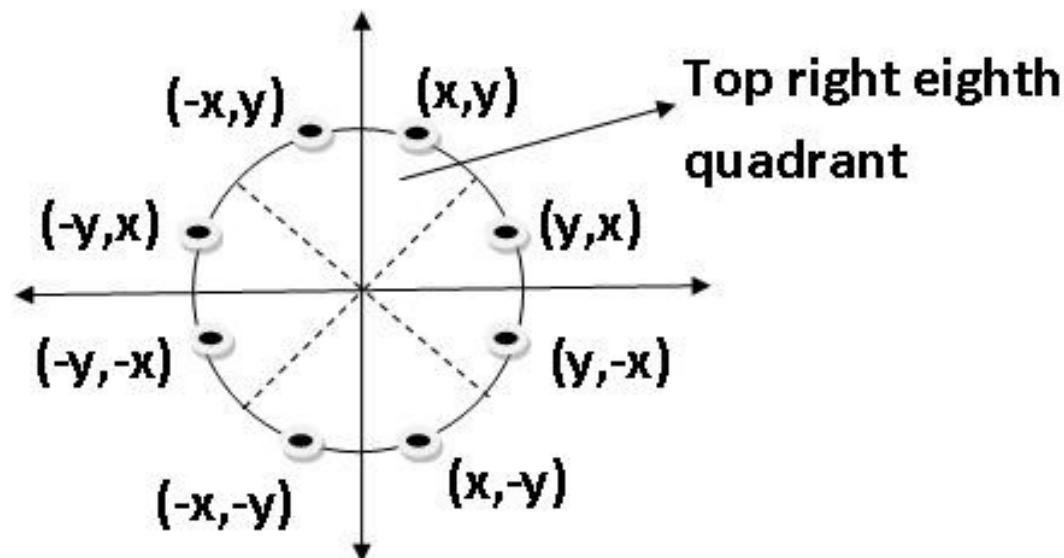
$$y = \pm\sqrt{r^2 - x^2}$$

Circle Scan Conversion

- Unsurprisingly this is not a brilliant solution
- Firstly, the resulting circle has large gaps where the slope approaches the vertical
- Secondly, the calculations are not very efficient
 - The square (multiply) operations
 - The square root operation – try really hard to avoid these!
- We need a more efficient, more accurate solution

Eight-Way Symmetry

- Circles centred at $(0, 0)$ have *eight-way symmetry*
 - this fact can be exploited to design an efficient algorithm

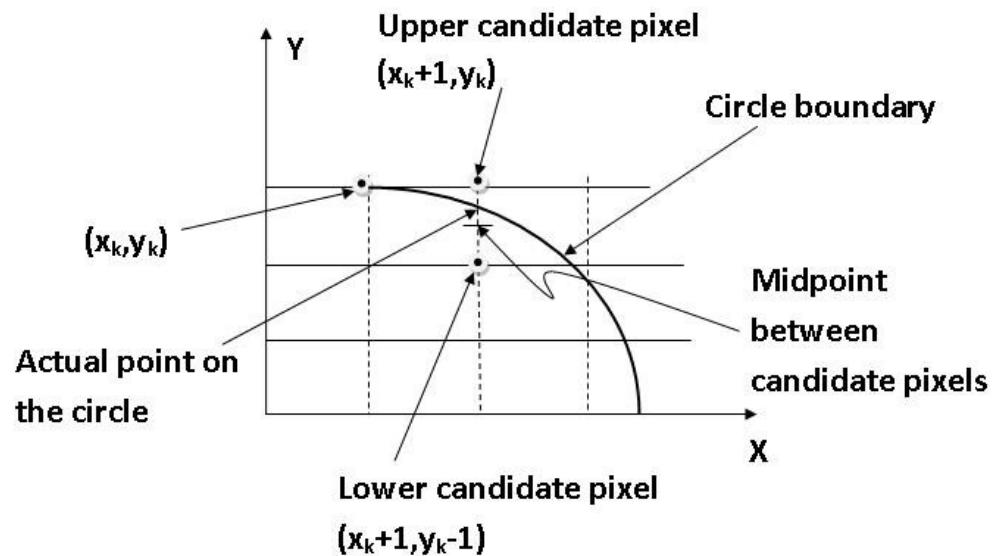


Mid-Point Circle Algorithm

- An incremental algorithm for drawing circles
- Algorithm calculates pixels only for the top right eighth of the circle
- Other points are derived using the eight-way symmetry

Mid-Point Circle Algorithm

- Assume that we have just plotted point (x_k, y_k)
- The next point is a choice between (x_k+1, y_k) and (x_k+1, y_k-1)
- We would like to choose the point that is nearest to the actual circle
 - So how do we make this choice?



Mid-Point Circle Algorithm

- Let's re-jig the equation of the circle slightly to give us:

$$f_{circ}(x, y) = x^2 + y^2 - r^2$$

- The equation evaluates as follows:

$$f_{circ}(x, y) \begin{cases} < 0, & \text{if } (x, y) \text{ is inside the circle boundary} \\ = 0, & \text{if } (x, y) \text{ is on the circle boundary} \\ > 0, & \text{if } (x, y) \text{ is outside the circle boundary} \end{cases}$$

- By evaluating this function at the midpoint between the candidate pixels we can make our decision

Mid-Point Circle Algorithm

- Assuming we have just plotted the pixel at (x_k, y_k) so we need to choose between $(x_k + 1, y_k)$ and $(x_k + 1, y_k - 1)$
- Our decision variable can be defined as:

$$\begin{aligned} p_k &= f_{circ}(x_k + 1, y_k - \frac{1}{2}) \\ &= (x_k + 1)^2 + (y_k - \frac{1}{2})^2 - r^2 \end{aligned}$$

- If $p_k < 0$ the midpoint is inside the circle and the pixel at y_k is closer to the circle
- Otherwise the midpoint is outside and $y_k - 1$ is closer

Mid-Point Circle Algorithm

- To ensure things are as efficient as possible we can do all of our calculations incrementally
- First consider:
$$\begin{aligned} p_{k+1} &= f_{circ} \left(x_{k+1} + 1, y_{k+1} - \frac{1}{2} \right) \\ &= [(x_k + 1) + 1]^2 + \left(y_{k+1} - \frac{1}{2} \right)^2 - r^2 \end{aligned}$$

or

$$p_{k+1} = p_k + 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

where y_{k+1} is either y_k or $y_k - 1$ depending on the sign of p_k

Mid-Point Circle Algorithm

- The first decision variable is given as

$$\begin{aligned} p_0 &= f_{circ}(1, r - \frac{1}{2}) \\ &= 1 + (r - \frac{1}{2})^2 - r^2 \\ &= \frac{5}{4} - r \end{aligned}$$

- Then if $p_k < 0$ then the next decision variable is given as: $p_{k+1} = p_k + 2x_{k+1} + 1$
- If $p_k > 0$ then the decision variable is

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_k + 1$$

Mid-Point Circle Algorithm

- Input radius r and circle centre (x_c, y_c) , then set the coordinates for the first point on the circumference of a circle centred on the origin as:
$$(x, y) = (0, \text{roundoff}(r))$$
- Calculate the initial value of the decision parameter as:
$$p = \frac{5}{4}r^2 - r$$
- If $p < 0$, the next point along the circle centred on $(0, 0)$ is $(x+1, y)$ and:

$$p = p + 2x + 1$$

Mid-Point Circle Algorithm

- Otherwise the next point along the circle is $(x+1, y-1)$ and $p = p + 2x + 1 - 2y$
- Determine symmetry points in the other seven octants
- Move each calculated pixel position (x, y) onto the circular path centred at (x_c, y_c) to plot the coordinate values:

$$x = x + x_c \quad y = y + y_c$$

- Repeat steps 3 to 5 until $x >= y$

Thank You

2D Geometric Transformations

Lenin Laitonjam

Department of Computer Science and
Engineering

NIT Mizoram

Objectives

- Basic 2D Transformations (rigid-body transformations):
 - Translation
 - Rotation
 - Scaling
- Homogeneous Representations and Coordinates
- 2D Composite Transformations

Objectives (cont.)

- Other Transformations:
 - Reflection
 - Shearing

Geometric Transformations

- Sometimes also called modeling transformations
 - Geometric transformations: Changing an object's position (translation), orientation (rotation) or size (scaling)
 - Modeling transformations: Constructing a scene or hierarchical description of a complex object
- Others transformations: reflection and shearing operations

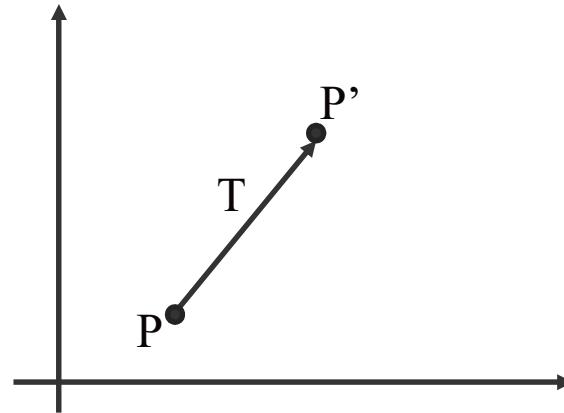
Basic 2D Geometric Transformations

■ 2D Translation

- $x' = x + t_x, y' = y + t_y$

$$P = \begin{bmatrix} x \\ y \end{bmatrix}, P' = \begin{bmatrix} x' \\ y' \end{bmatrix}, T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

- $P' = P + T$
- Translation moves the object without deformation (rigid-body transformation)

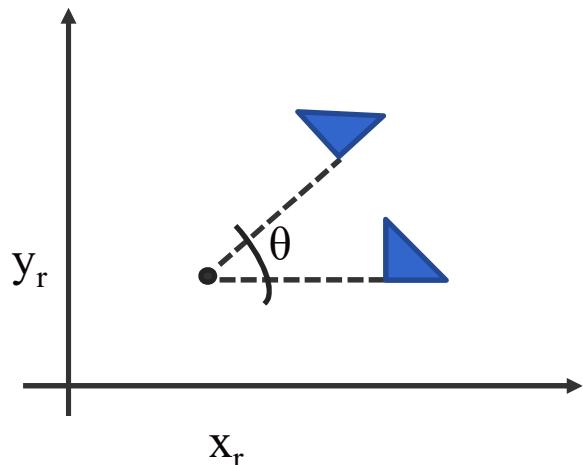


Basic 2D Geometric Transformations (cont.)

- 2D Translation
 - To move a line segment, apply the transformation equation to each of the two line endpoints and redraw the line between new endpoints
 - To move a polygon, apply the transformation equation to coordinates of each vertex and regenerate the polygon using the new set of vertex coordinates

Basic 2D Geometric Transformations (cont.)

- 2D Rotation
 - Rotation axis
 - Rotation angle
 - rotation point or pivot point (x_r, y_r)



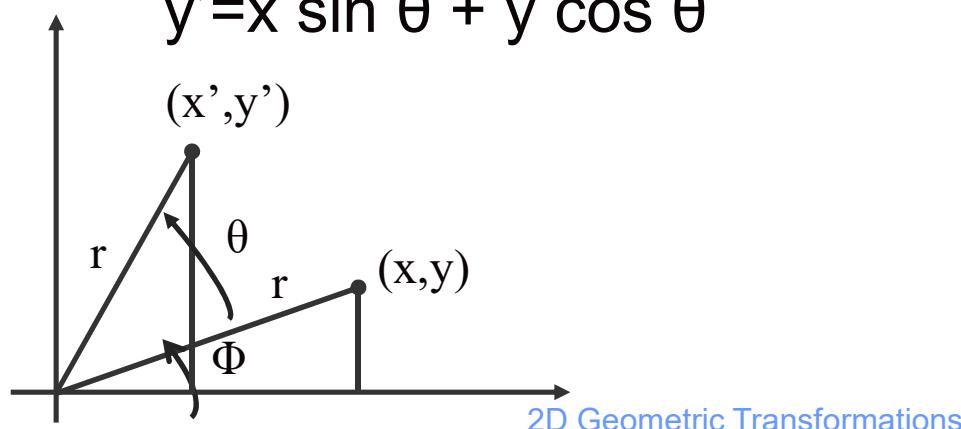
Basic 2D Geometric Transformations (cont.)

- 2D Rotation
 - If θ is positive \rightarrow counterclockwise rotation
 - If θ is negative \rightarrow clockwise rotation
 - Remember:
 - $\cos(a + b) = \cos a \cos b - \sin a \sin b$
 - $\cos(a - b) = \cos a \sin b + \sin a \cos b$

Basic 2D Geometric Transformations (cont.)

■ 2D Rotation

- At first, suppose the pivot point is at the origin
- $x' = r \cos(\theta + \Phi) = r \cos \theta \cos \Phi - r \sin \theta \sin \Phi$
- $y' = r \sin(\theta + \Phi) = r \cos \theta \sin \Phi + r \sin \theta \cos \Phi$
- $x = r \cos \Phi, y = r \sin \Phi$
- $x' = x \cos \theta - y \sin \theta$
- $y' = x \sin \theta + y \cos \theta$

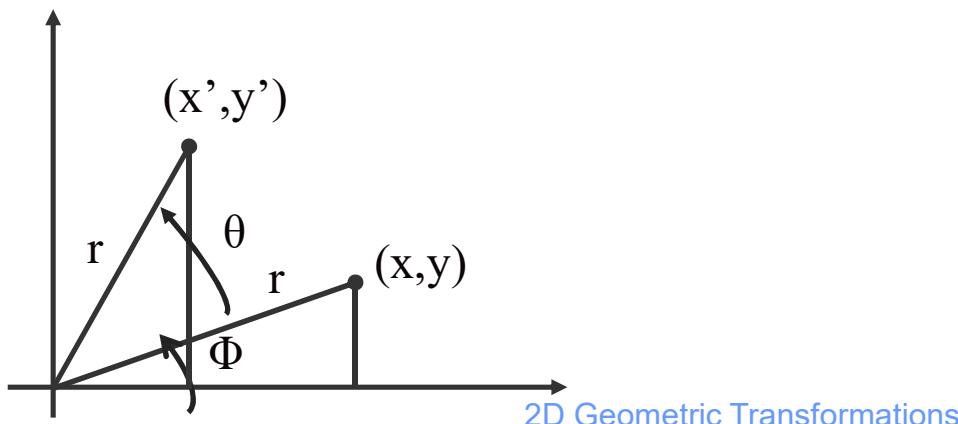


Basic 2D Geometric Transformations

- 2D Rotation

- $P' = R \cdot P$

$$R = \begin{bmatrix} \cos \Theta & -\sin \Theta \\ \sin \Theta & \cos \Theta \end{bmatrix}$$



Basic 2D Geometric Transformations (cont.)

■ 2D Rotation

- Rotation of a point about any specified position (x_r, y_r)
$$x' = x_r + (x - x_r) \cos \theta - (y - y_r) \sin \theta$$
$$y' = y_r + (x - x_r) \sin \theta + (y - y_r) \cos \theta$$
- Rotations also move objects without deformation
- A line is rotated by applying the rotation formula to each of the endpoints and redrawing the line between the new end points
- A polygon is rotated by applying the rotation formula to each of the vertices and redrawing the polygon using new vertex coordinates

Basic 2D Geometric Transformations (cont.)

- 2D Scaling
 - Scaling is used to alter the size of an object
 - Simple 2D scaling is performed by multiplying object positions (x, y) by scaling factors s_x and s_y

$$x' = x \cdot s_x$$

$$y' = y \cdot s_y$$

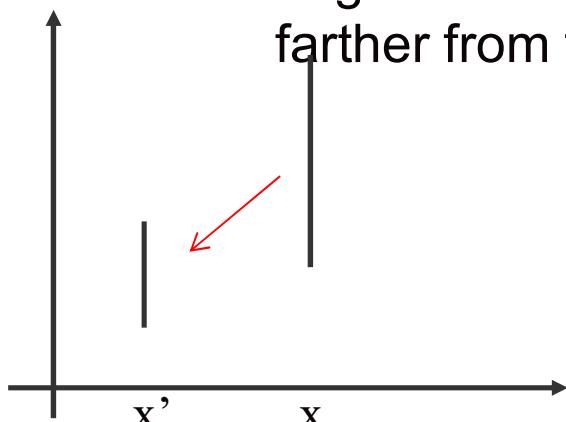
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\text{or } P' = S \cdot P$$

Basic 2D Geometric Transformations (cont.)

■ 2D Scaling

- Any positive value can be used as scaling factor
 - Values less than 1 reduce the size of the object
 - Values greater than 1 enlarge the object
 - If scaling factor is 1 then the object stays unchanged
 - If $s_x = s_y$, we call it uniform scaling
 - If scaling factor <1 , then the object moves closer to the origin and If scaling factor >1 , then the object moves farther from the origin



Basic 2D Geometric Transformations (cont.)

- 2D Scaling
 - Why does scaling also reposition object?
 - Answer: See the matrix (multiplication)
 - Still no clue?
 - $$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x * s_x + y * 0 \\ x * 0 + y * s_y \end{bmatrix}$$

Basic 2D Geometric Transformations (cont.)

■ 2D Scaling

- We can control the location of the scaled object by choosing a position called the **fixed point** (x_f, y_f)

$$x' - x_f = (x - x_f) s_x \quad y' - y_f = (y - y_f) s_y$$

$$x' = x \cdot s_x + x_f(1 - s_x)$$

$$y' = y \cdot s_y + y_f(1 - s_y)$$

- Polygons are scaled by applying the above formula to each vertex, then regenerating the polygon using the transformed vertices

Matrix Representations and Homogeneous Coordinates

- Many graphics applications involve sequences of geometric transformations
 - Animations
 - Design and picture construction applications
- We will now consider matrix representations of these operations
 - Sequences of transformations can be efficiently processed using matrices

Matrix Representations and Homogeneous Coordinates (cont.)

- $P' = M_1 \cdot P + M_2$
 - P and P' are column vectors
 - M_1 is a 2 by 2 array containing multiplicative factors
 - M_2 is a 2 element column matrix containing translational terms
 - For translation M_1 is the identity matrix
 - For rotation or scaling, M_2 contains the translational terms associated with the pivot point or scaling fixed point

Matrix Representations and Homogeneous Coordinates (cont.)

- To produce a sequence of operations, such as scaling followed by rotation then translation, we could calculate the transformed coordinates one step at a time
- A more efficient approach is to combine transformations, without calculating intermediate coordinate values

Matrix Representations and Homogeneous Coordinates (cont.)

- Multiplicative and translational terms for a 2D geometric transformation can be combined into a single matrix if we expand the representations to 3 by 3 matrices
 - We can use the third column for translation terms, and all transformation equations can be expressed as matrix multiplications

Matrix Representations and Homogeneous Coordinates (cont.)

- Expand each 2D coordinate (x, y) to three element representation (x_h, y_h, h) called **homogeneous coordinates**
- h is the **homogeneous parameter** such that
$$x = x_h/h, \quad y = y_h/h,$$
- → infinite homogeneous representations for a point
- A convenient choice is to choose $h = 1$

Matrix Representations and Homogeneous Coordinates (cont.)

- 2D Translation Matrix

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

or, $\mathbf{P}' = \mathbf{T}(t_x, t_y) \cdot \mathbf{P}$

Matrix Representations and Homogeneous Coordinates (cont.)

- 2D Rotation Matrix

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

or, $\mathbf{P}' = \mathbf{R}(\theta) \cdot \mathbf{P}$

Matrix Representations and Homogeneous Coordinates (cont.)

- 2D Scaling Matrix

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

or, $P' = S(s_x, s_y) \cdot P$

Inverse Transformations

- 2D Inverse Translation Matrix

$$T^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

- By the way:

$$T^{-1} * T = I$$

Inverse Transformations (cont.)

- 2D Inverse Rotation Matrix

$$R^{-1} = \begin{bmatrix} \cos \Theta & \sin \Theta & 0 \\ -\sin \Theta & \cos \Theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- And also:

$$R^{-1} * R = I$$

Inverse Transformations (cont.)

- 2D Inverse Rotation Matrix:

- If θ is negative \rightarrow clockwise
 - In

$$R^{-1} * R = I$$

- Only sine function is affected
 - Therefore we can say

$$R^{-1} = R^T$$

- Is that true?
 - Proof: It's up to you ☺

Inverse Transformations (cont.)

- 2D Inverse Scaling Matrix

$$S^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Of course:

$$S^{-1} * S = I$$

2D Composite Transformations

- We can setup a sequence of transformations as a **composite transformation matrix** by calculating the product of the individual transformations
- $P' = M_2 \cdot M_1 \cdot P$
 $= M \cdot P$

2D Composite Transformations (cont.)

- Composite 2D Translations
 - If two successive translation are applied to a point P, then the final transformed location P' is calculated as

$$\mathbf{P}' = \mathbf{T}(t_{x2}, t_{y2}) \cdot \mathbf{T}(t_{x1}, t_{y1}) \cdot \mathbf{P} = \mathbf{T}(t_{x1} + t_{x2}, t_{y1} + t_{y2}) \cdot \mathbf{P}$$

$$\begin{bmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{1x} + t_{2x} \\ 0 & 1 & t_{1y} + t_{2y} \\ 0 & 0 & 1 \end{bmatrix}$$

2D Composite Transformations (cont.)

- Composite 2D Rotations

$$\mathbf{P}' = \mathbf{R}(\theta_1 + \theta_2) \cdot \mathbf{P}$$

$$\begin{bmatrix} \cos\Theta_2 & -\sin\Theta_2 & 0 \\ \sin\Theta_2 & \cos\Theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\Theta_1 & -\sin\Theta_1 & 0 \\ \sin\Theta_1 & \cos\Theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\Theta_1 + \Theta_2) & -\sin(\Theta_1 + \Theta_2) & 0 \\ \sin(\Theta_1 + \Theta_2) & \cos(\Theta_1 + \Theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2D Composite Transformations (cont.)

- Composite 2D Scaling

$$S(s_{x_2}, s_{y_2}) \cdot S(s_{x_1}, s_{y_1}) = S(s_{x_1} \cdot s_{x_2}, s_{y_1} \cdot s_{y_2})$$

$$\begin{bmatrix} s_{2x} & 0 & 0 \\ 0 & s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{1x} & 0 & 0 \\ 0 & s_{1y} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{1x} \cdot s_{2x} & 0 & 0 \\ 0 & s_{1y} \cdot s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

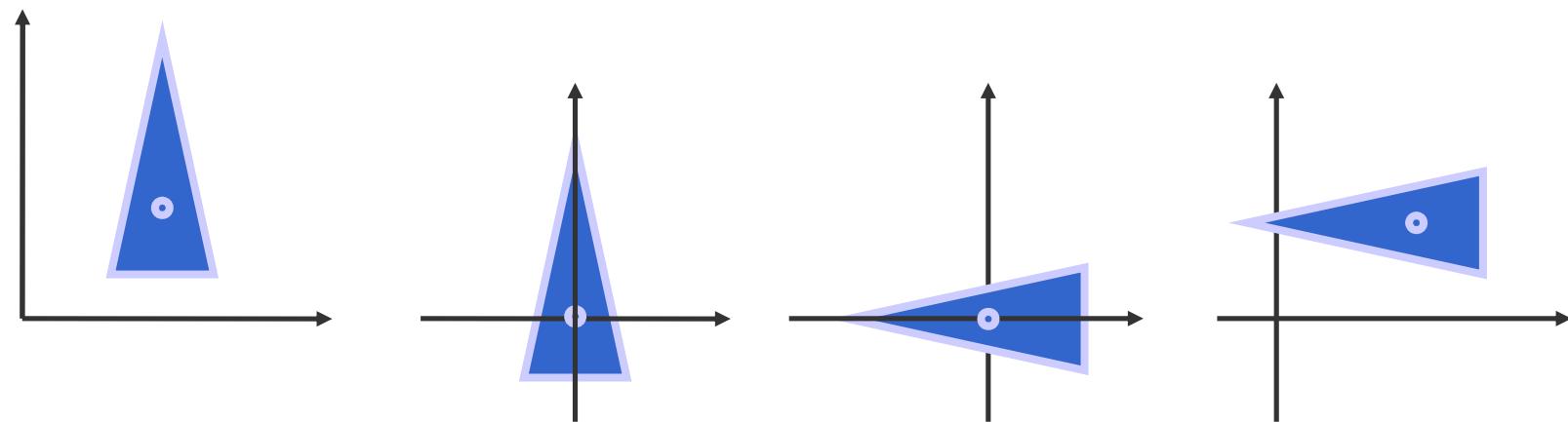
2D Composite Transformations (cont.)

- Don't forget:
- Successive translations are additive
- Successive scalings are multiplicative
 - For example: If we triple the size of an object twice, the final size is nine (9) times the original
 - 9 times?
 - Why?
 - Proof: Again up to you ☺

General Pivot Point Rotation

- Steps:
 1. Translate the object so that the pivot point is moved to the coordinate origin.
 2. Rotate the object about the origin.
 3. Translate the object so that the pivot point is returned to its original position.

General Pivot Point Rotation



2D Composite Transformations (cont.)

- General 2D Pivot-Point Rotation

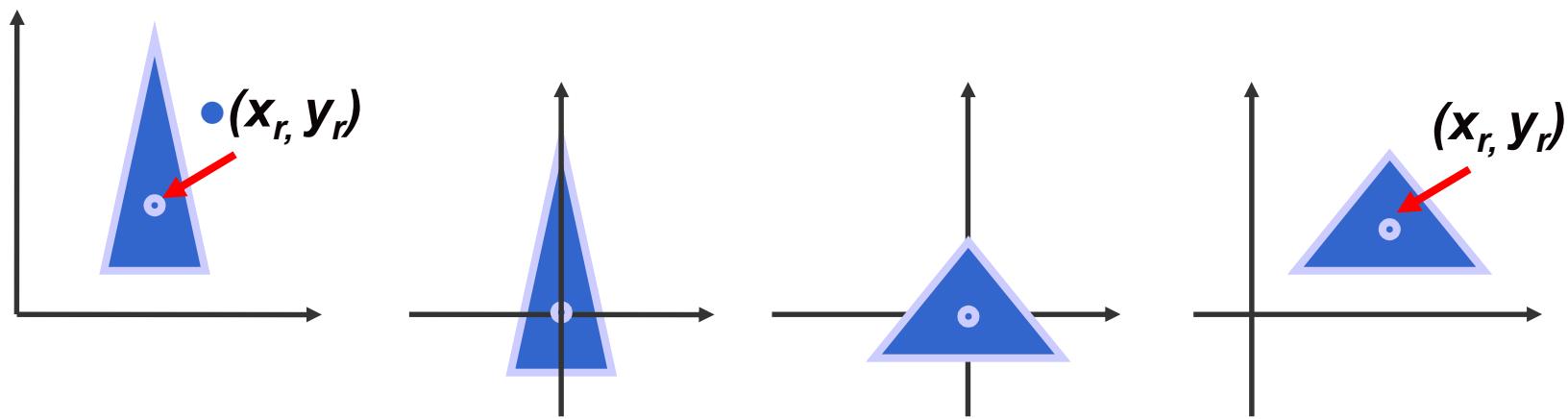
$$\begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \Theta & -\sin \Theta & 0 \\ \sin \Theta & \cos \Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos \Theta & -\sin \Theta & x_r(1-\cos \Theta) + y_r \sin \Theta \\ \sin \Theta & \cos \Theta & y_r(1-\cos \Theta) - x_r \sin \Theta \\ 0 & 0 & 1 \end{bmatrix}$$

General Fixed Point Scaling

- Steps:
 1. Translate the object so that the fixed point coincides with the coordinate origin.
 2. Scale the object about the origin.
 3. Translate the object so that the pivot point is returned to its original position.

General Fixed Point Scaling (cont.)



General Fixed Point Scaling (cont.)

- General 2D Fixed-Point Scaling:

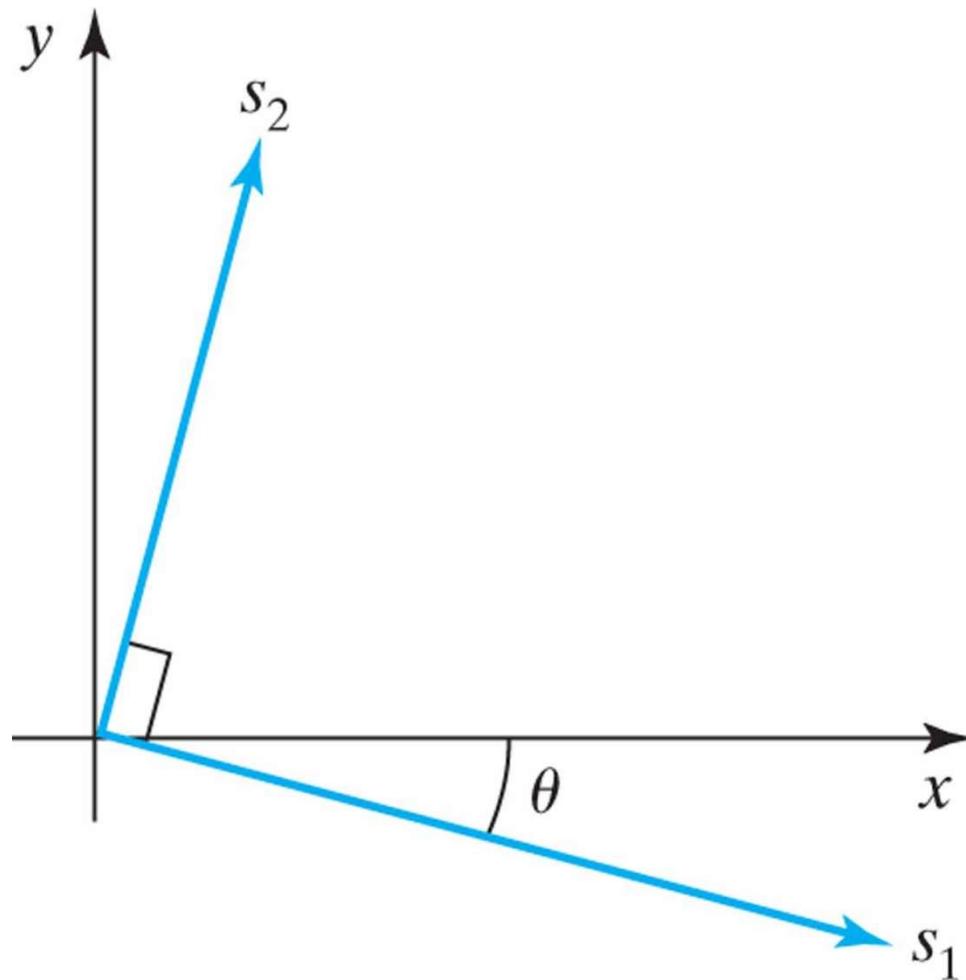
$$\begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & x_f(1 - s_x) \\ 0 & s_y & y_f(1 - s_y) \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}(x_f, y_f) \cdot \mathbf{S}(s_x, s_y) \cdot \mathbf{T}(-x_f, -y_f) = \mathbf{S}(x_f, y_f, s_x, s_y)$$

2D Composite Transformations (cont.)

- General 2D scaling directions:
 - Above: scaling parameters were along x and y directions
 - What about arbitrary directions?
 - Answer: See next slides

General 2D Scaling Directions



Copyright ©2011 Pearson Education, publishing as Prentice Hall

Scaling parameters s_1 and s_2 along orthogonal directions defined by the angular displacement θ .

2D Geometric Transformations

40

General 2D Scaling Directions (cont.)

- General procedure:
 1. Rotate so that directions coincides with x and y axes
 2. Apply scaling transformation $S(s_1, s_2)$
 3. Rotate back
- The composite matrix:

$$R^{-1}(\Theta) * S(s_1, s_2) * R(\Theta) = \begin{bmatrix} s_1 \cos^2 \Theta + s_2 \sin^2 \Theta & (s_2 - s_1) \cos \Theta \sin \Theta & 0 \\ (s_2 - s_1) \cos \Theta \sin \Theta & s_1 \sin^2 \Theta + s_2 \cos^2 \Theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2D Composite Transformations (cont.)

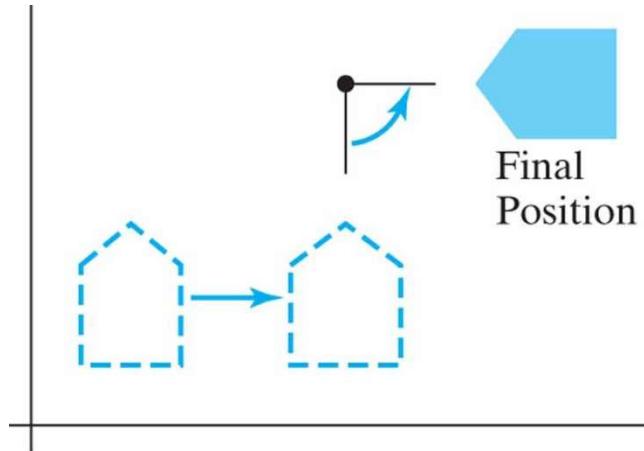
- Matrix Concatenation Properties:
 - Matrix multiplication is associative !
 - $M_3 \cdot M_2 \cdot M_1 = (M_3 \cdot M_2) \cdot M_1 = M_3 \cdot (M_2 \cdot M_1)$
 - A composite matrix can be created by multiplicating left-to-right (premultiplication) or right-to-left (postmultiplication)
 - Matrix multiplication is *not* commutative !
 - $M_2 \cdot M_1 \neq M_1 \cdot M_2$

2D Composite Transformations (cont.)

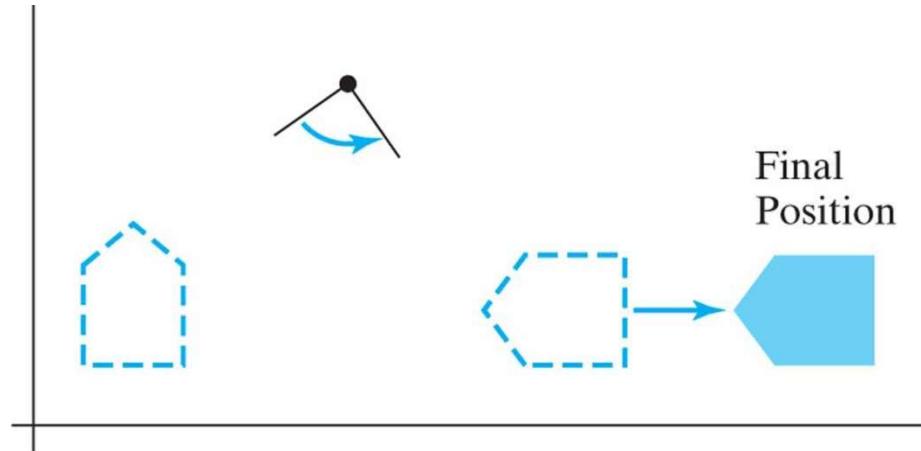
■ Matrix Concatenation Properties:

- But:
 - Two successive rotations
 - Two successive translations
 - Two successive scalings
- **are commutative!**
- Why?
- Proof: You got it: Up to you ☺ ☺

Reversing the order



(a)



(b)

Copyright ©2011 Pearson Education, publishing as Prentice Hall

in which a sequence of transformations is performed may affect the transformed position of an object.

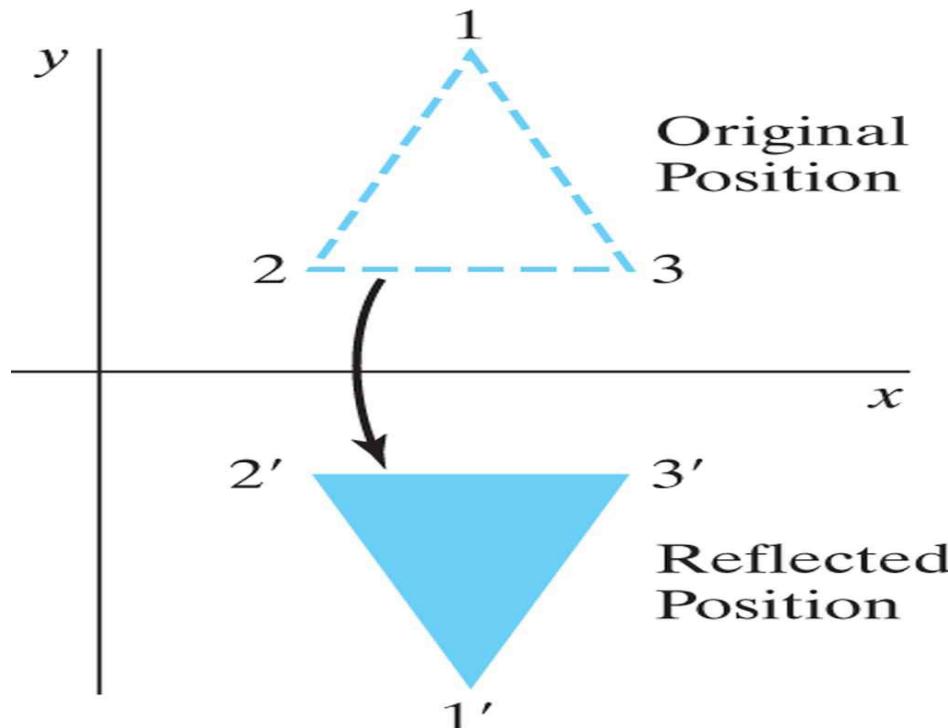
In (a), an object is first translated in the x direction, then rotated counterclockwise through an angle of 45° .

In (b), the object is first rotated 45° counterclockwise, then translated in the x direction

Other 2D Transformations

■ Reflection

- Transformation that produces a mirror image of an object



Other 2D Transformations (cont.)

■ Reflection

- Image is generated relative to an axis of reflection by rotating the object 180° about the reflection axis
- Reflection about the line $y=0$ (the x axis) (previous slide)

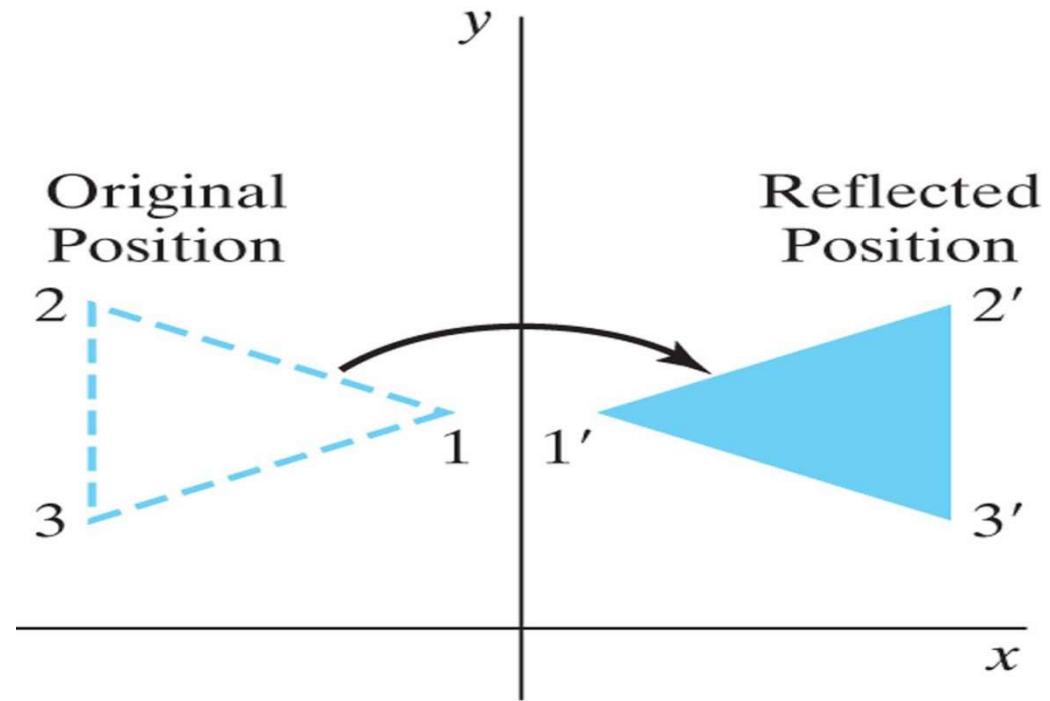
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Other 2D Transformations (cont.)

■ Reflection

- Reflection about the line $x=0$ (the y axis)

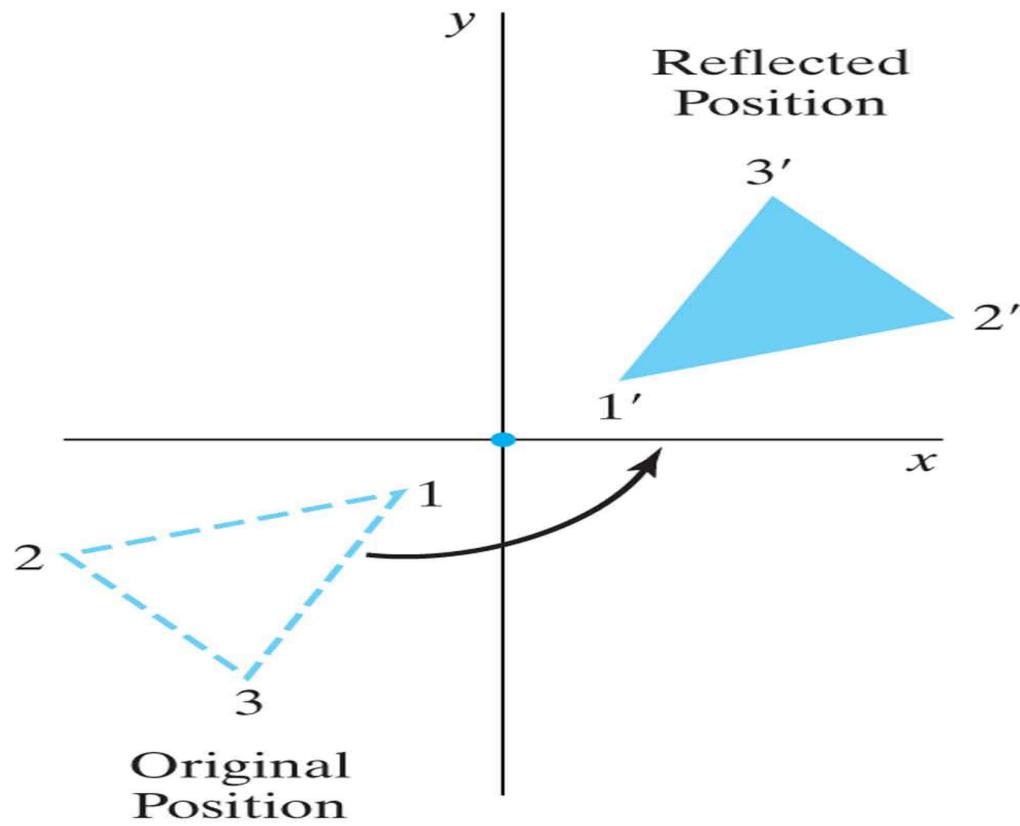
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Other 2D Transformations (cont.)

- Reflection about the origin

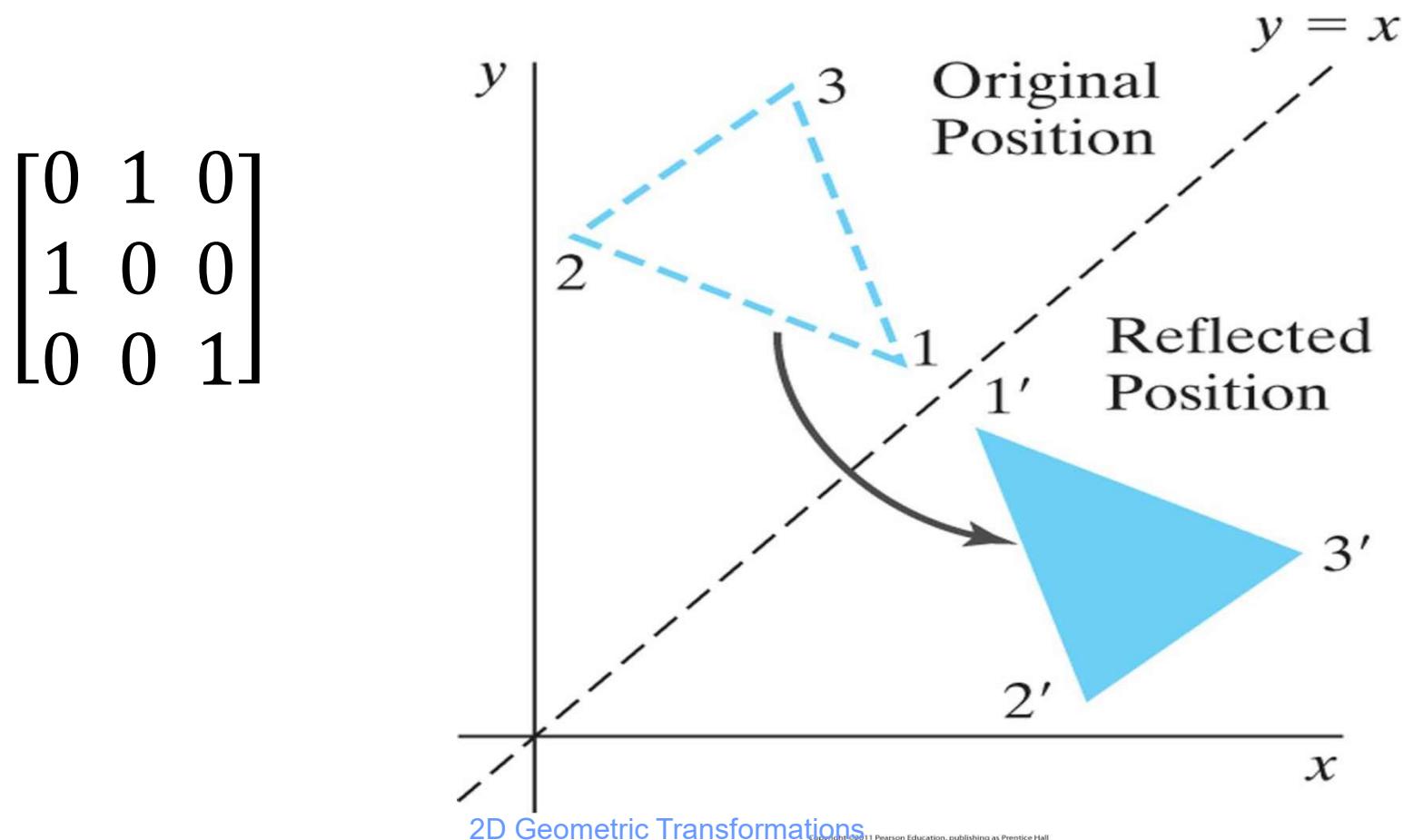
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Copyright ©2011 Pearson Education, publishing as Prentice Hall

Other 2D Transformations (cont.)

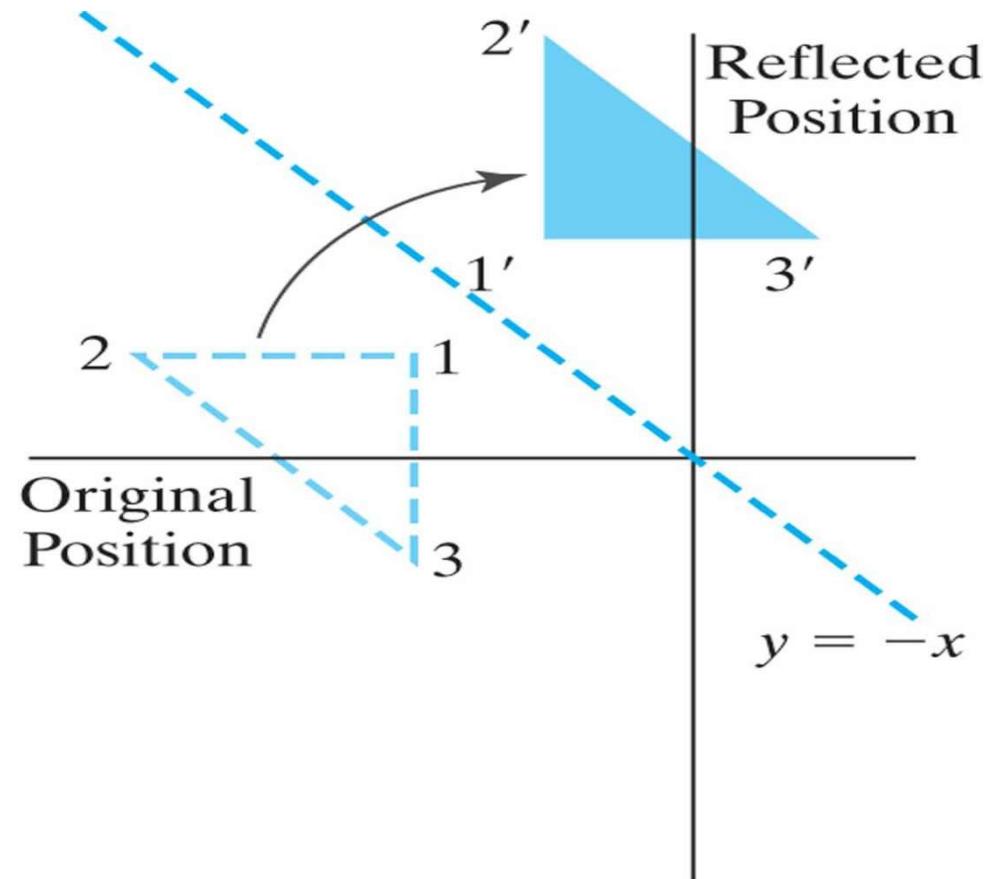
- Reflection about the line $y=x$



Other 2D Transformations (cont.)

- Reflection about the line $y=-x$

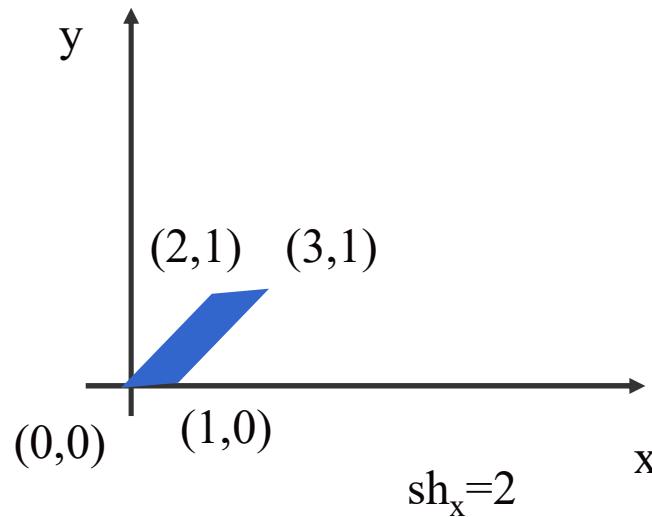
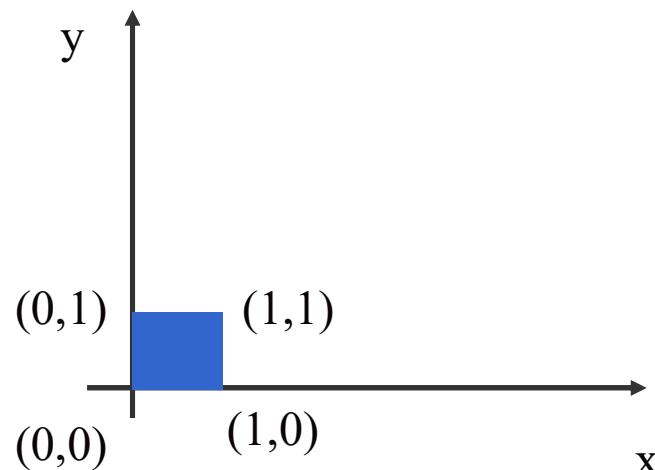
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Other 2D Transformations (cont.)

■ Shear

- Transformation that distorts the shape of an object such that the transformed shape appears as the object was composed of internal layers that had been caused to slide over each other



Other 2D Transformations (cont.)

■ Shear

- An x-direction shear relative to the x axis

$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{aligned} x' &= x + sh_x \cdot y \\ y' &= y \end{aligned}$$

- An y-direction shear relative to the y axis

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Other 2D Transformations (cont.)

■ Shear

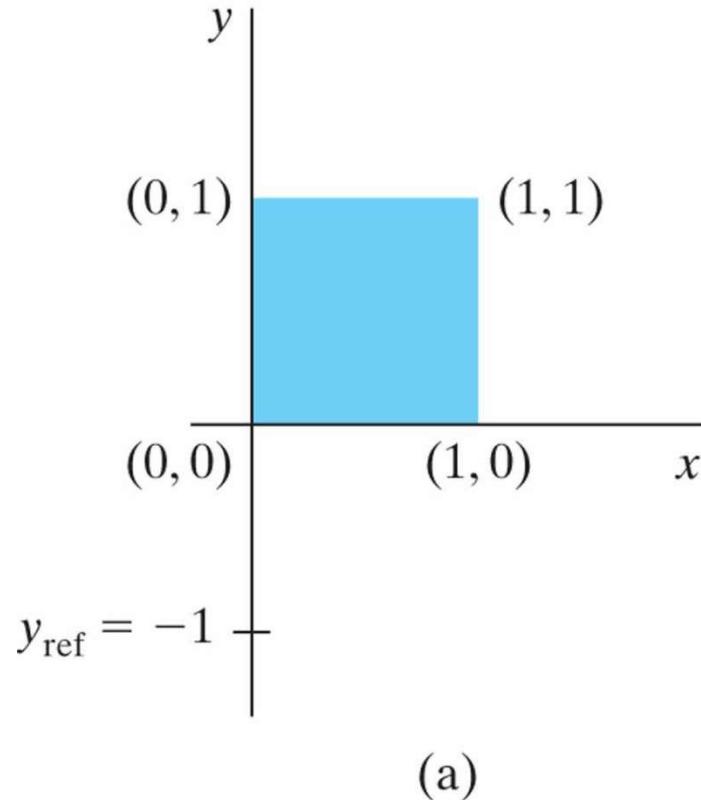
- x-direction shear relative to other reference lines

$$\begin{bmatrix} 1 & sh_x & -sh_x * y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

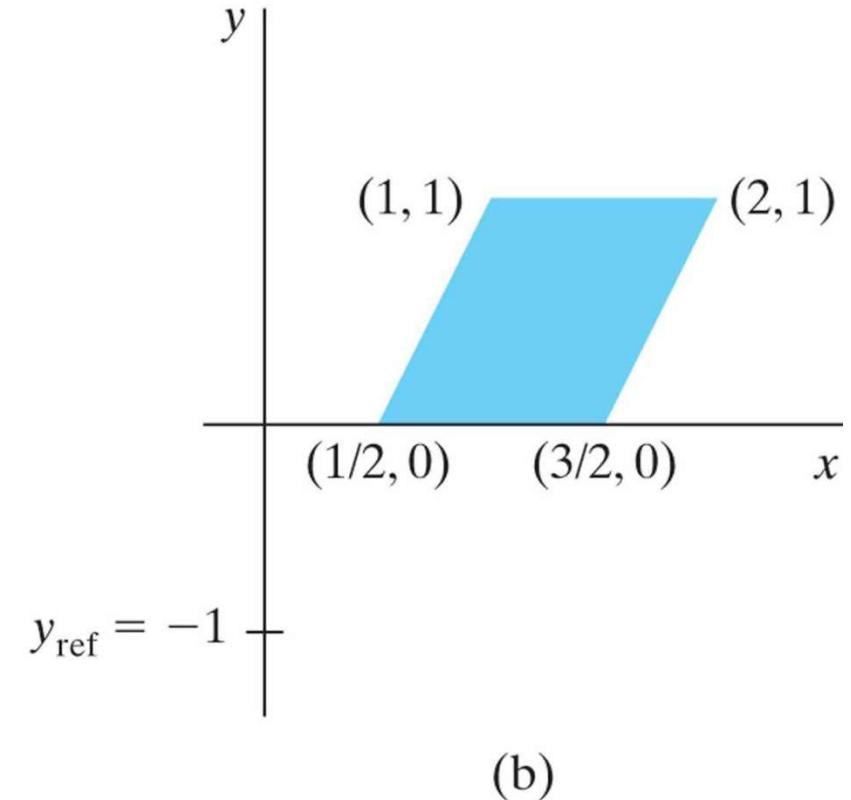
$$x' = x + sh_x * (y - y_{ref})$$

$$y' = y$$

Example



(a)



(b)

Copyright ©2011 Pearson Education, publishing as Prentice Hall

A unit square (a) is transformed to a shifted parallelogram
(b) with $sh_x = 0.5$ and $y_{\text{ref}} = -1$ in the shear matrix from Slide 56

Other 2D Transformations (cont.)

■ Shear

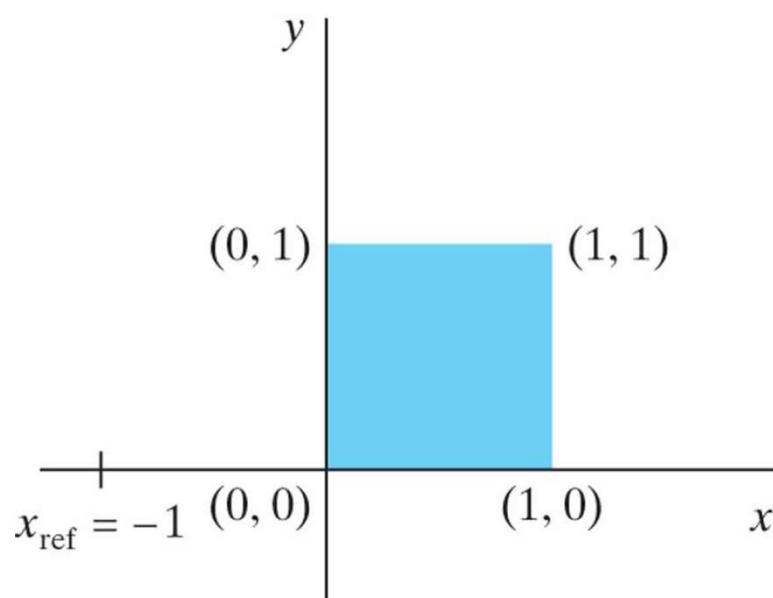
- y-direction shear relative to the line $x = x_{ref}$

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y * x_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

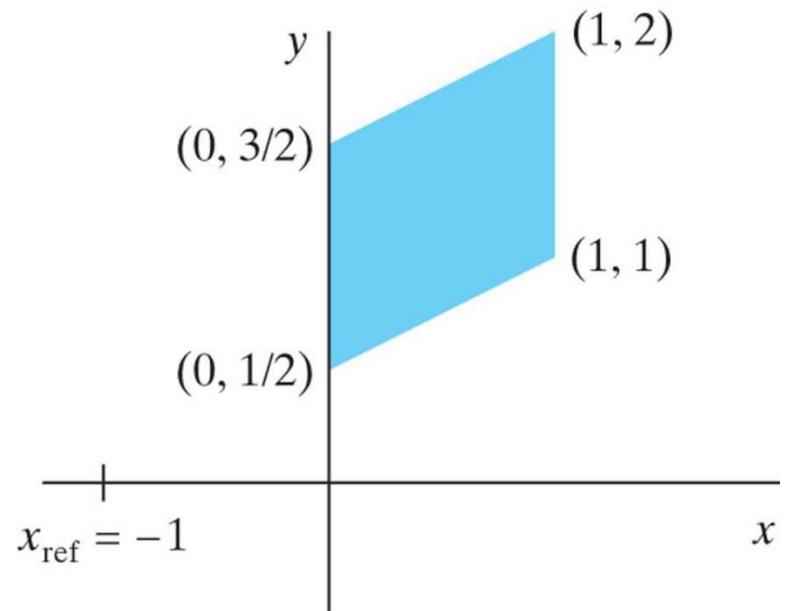
$$x' = x$$

$$y' = x + sh_y * (x - x_{ref})$$

Example



(a)



(b)

A unit square (a) is turned into a shifted parallelogram (b) with parameter values $sh_y = 0.5$ and $x_{\text{ref}} = -1$ in the y -direction shearing transformation from Slide 58

Thank You

3D Geometric Transformation

Lenin Laitonjam

Department of Computer Science and
Engineering
NIT Mizoram

Contents

- Translation
- Scaling
- Rotation
- Other Transformations

Transformation in 3D

■ Transformation Matrix

$$\begin{bmatrix} A & D & G & J \\ B & E & H & K \\ C & F & I & L \\ 0 & 0 & 0 & S \end{bmatrix} \rightarrow \begin{bmatrix} 3 \times 3 & 3 \times 1 \\ 1 \times 3 & 1 \times 1 \end{bmatrix}$$

3x3 : Scaling, Reflection, Shearing, Rotation

3x1 : Translation

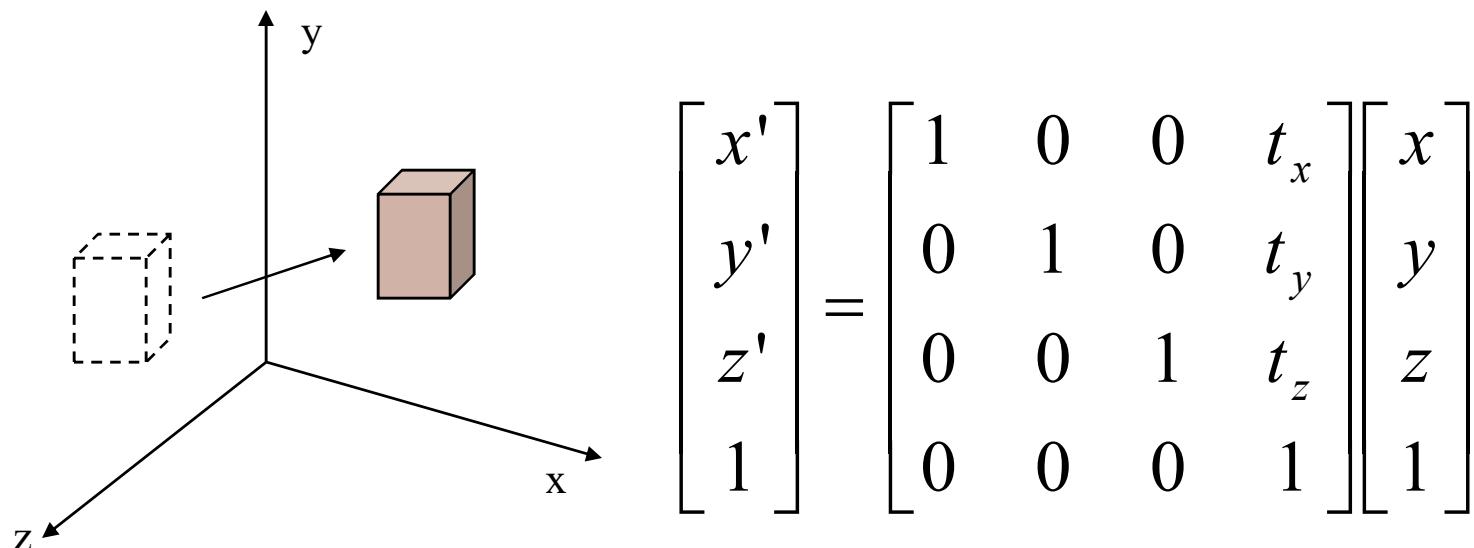
1x1 : Uniform global Scaling

1x3 : Homogeneous representation

3D Translation

■ Translation of a Point

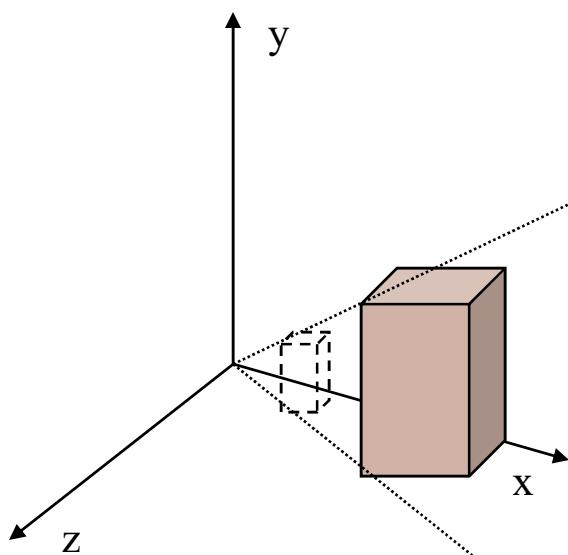
$$x' = x + t_x, \quad y' = y + t_y, \quad z' = z + t_z$$



3D Scaling

■ Uniform Scaling

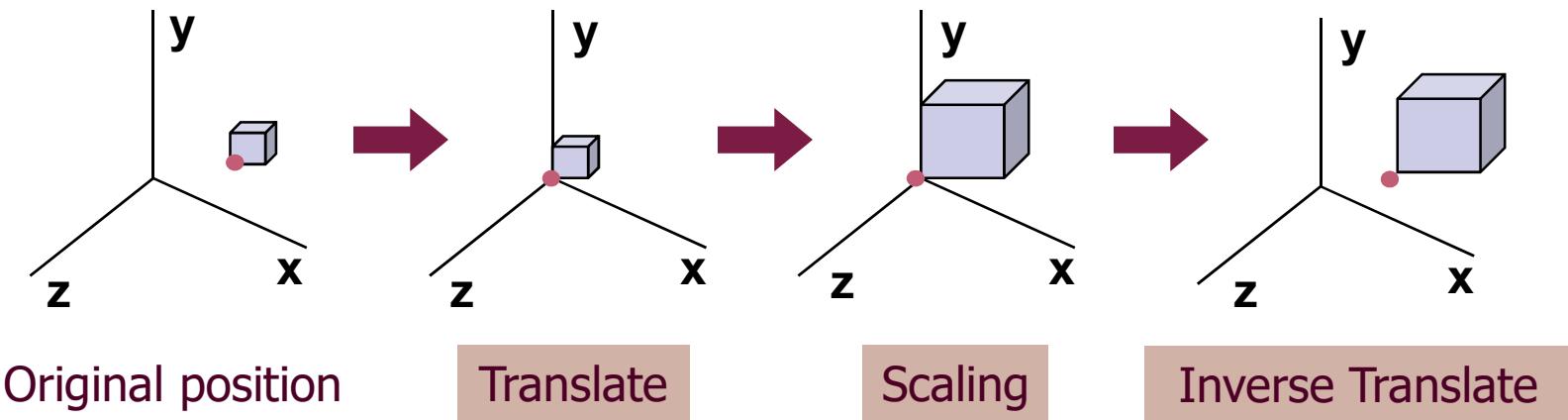
$$x' = x \cdot s_x, \quad y' = y \cdot s_y, \quad z' = z \cdot s_z$$



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

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

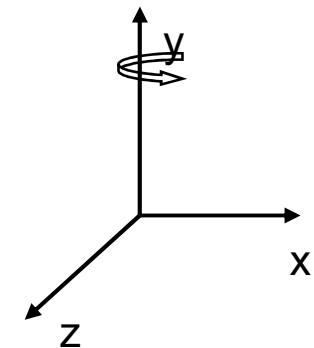
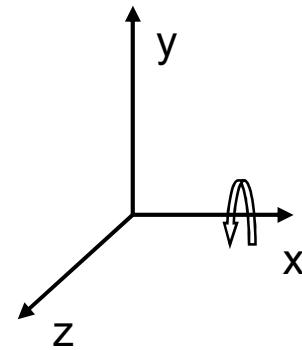
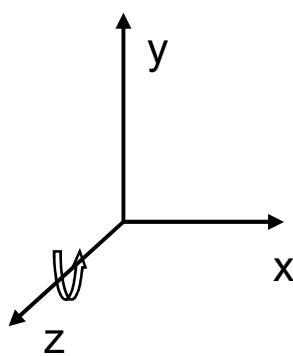
Coordinate-Axes Rotations

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

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

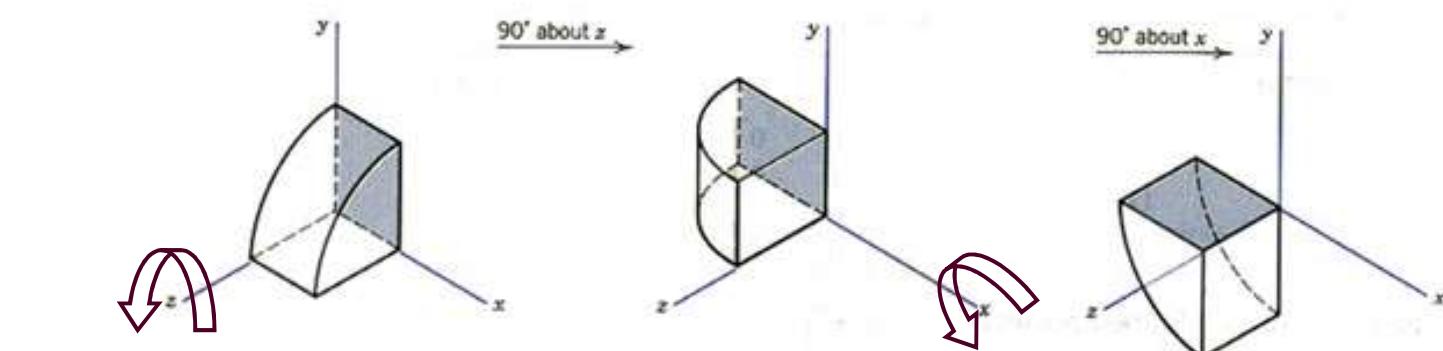
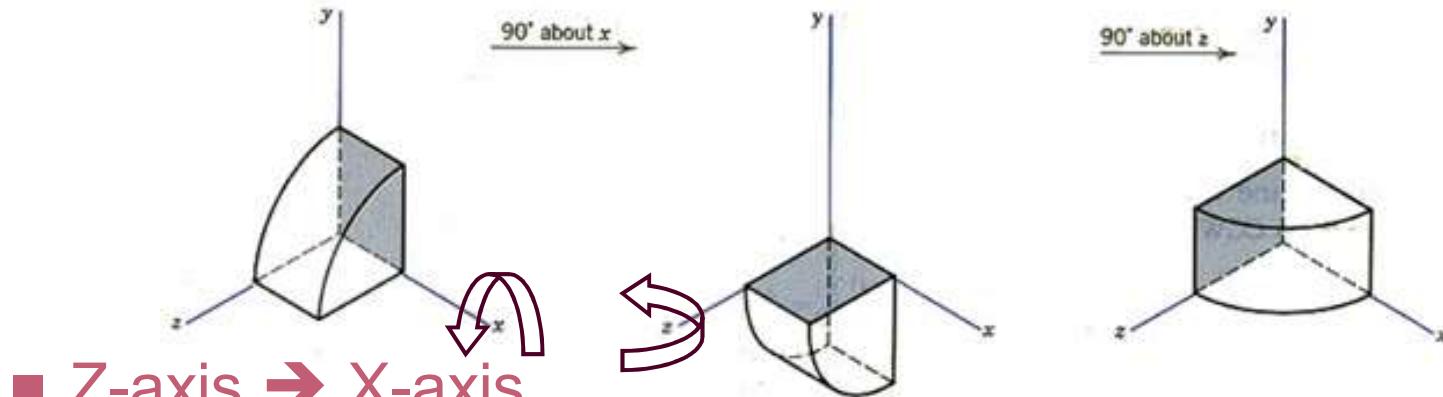
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

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



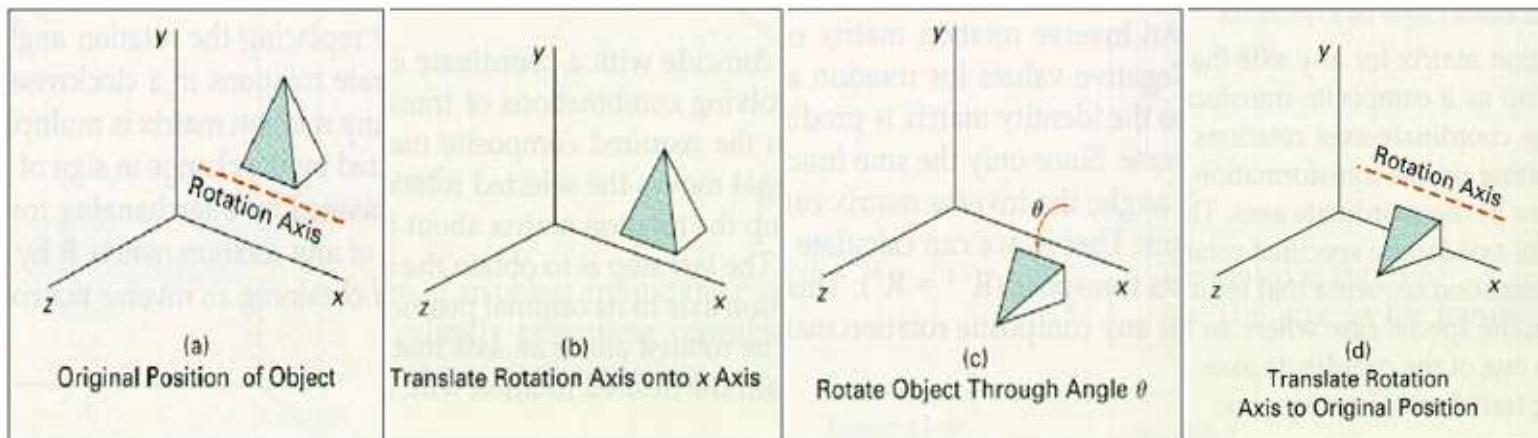
Order of Rotations

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



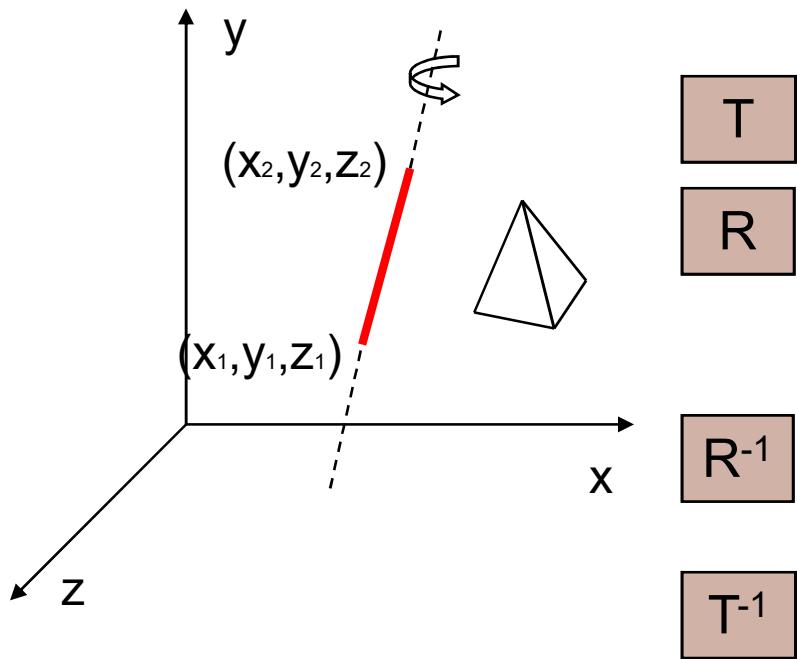
General 3D Rotations

- 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



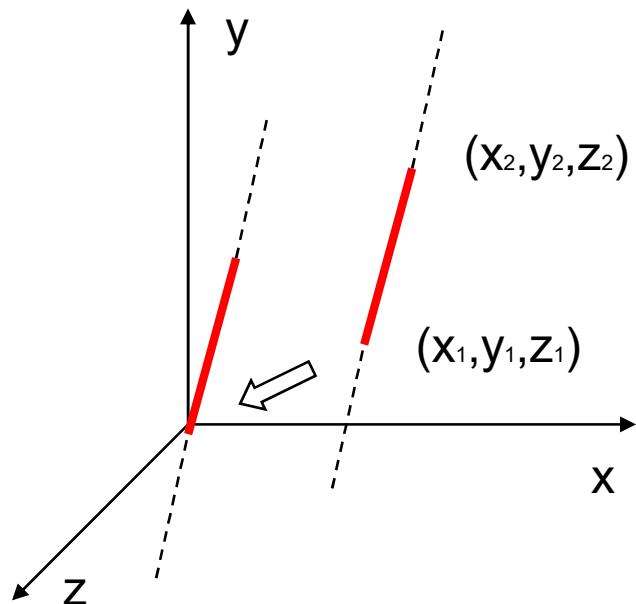
Basic Idea

1. Translate (x_1, y_1, z_1) 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

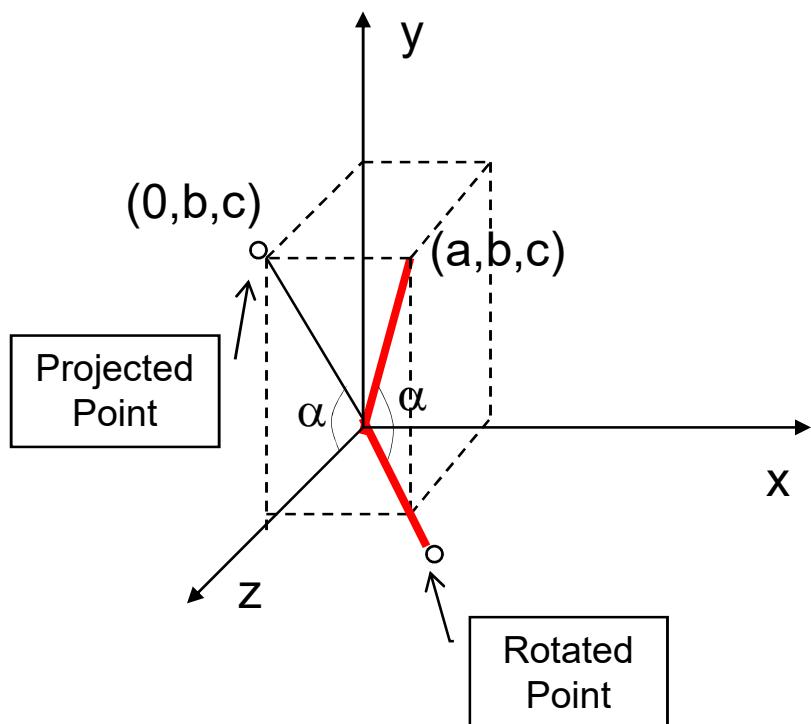
■ Step 1. Translation



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

General 3D Rotations

■ Step 2. Establish $[T_R]_x^\alpha$ x axis



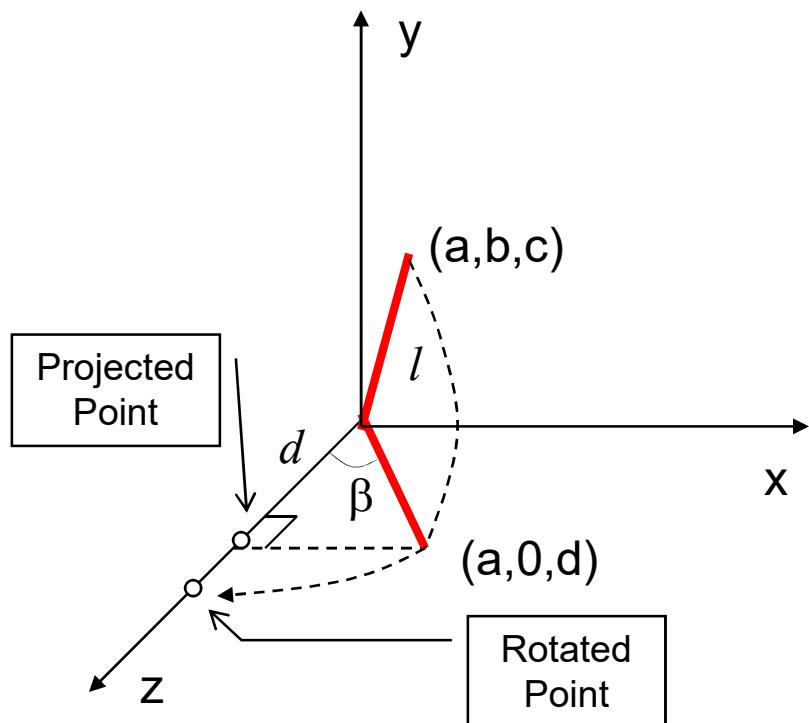
$$\sin \alpha = \frac{b}{\sqrt{b^2 + c^2}} = \frac{b}{d}$$

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

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

Arbitrary Axis Rotation

- Step 3. Rotate about y axis by ϕ



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

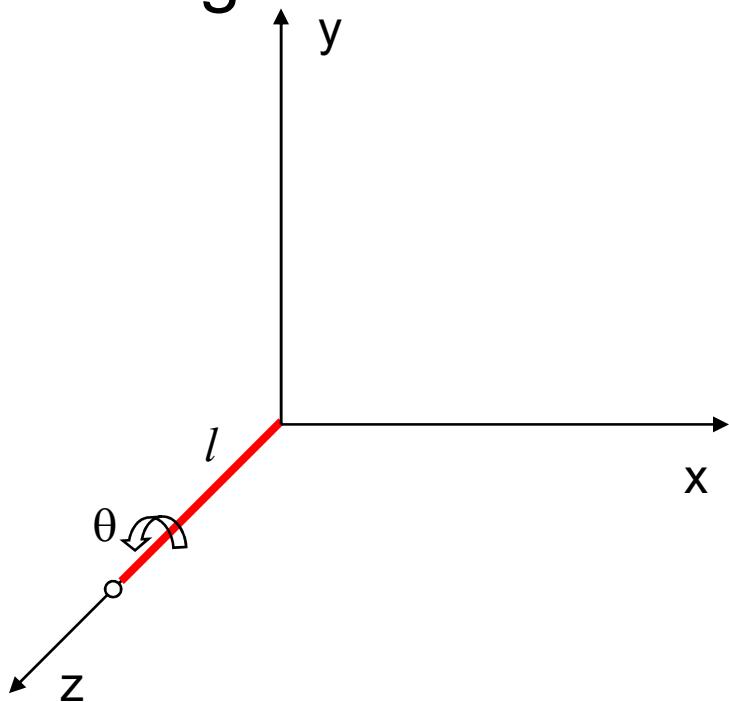
$$l^2 = a^2 + b^2 + c^2 = a^2 + d^2$$

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

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

Arbitrary Axis Rotation

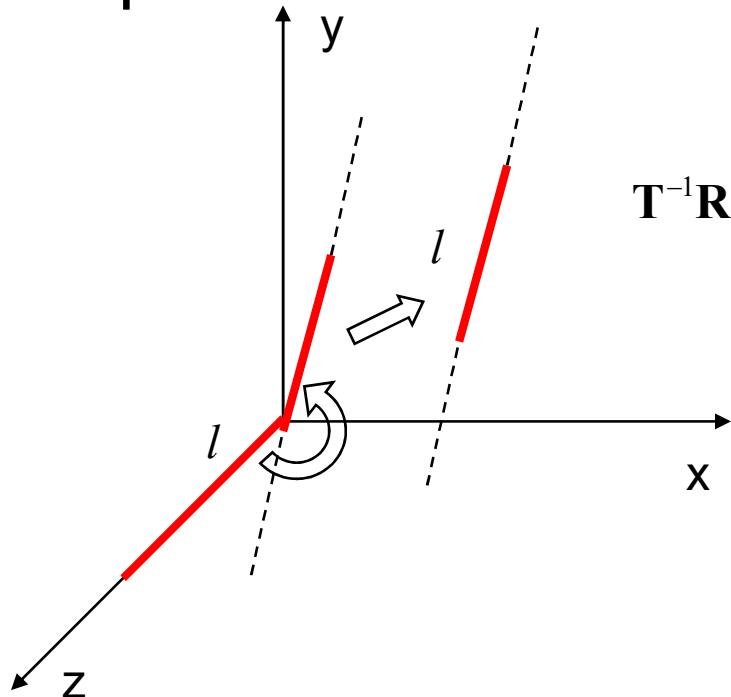
- Step 4. Rotate about z axis by the desired angle θ



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

Arbitrary Axis Rotation

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

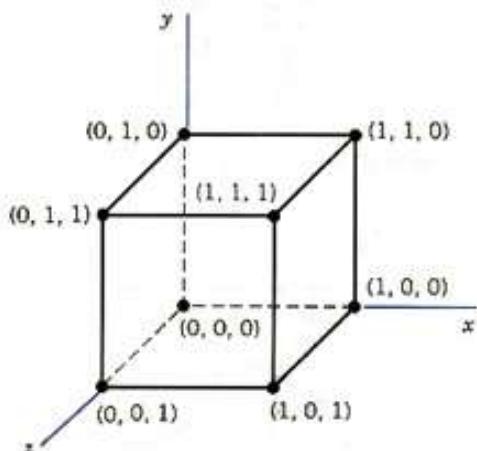


$$\mathbf{T}^{-1} \mathbf{R}_x^{-1}(\alpha) \mathbf{R}_y^{-1}(\beta) = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha & 0 \\ 0 & -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$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

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

Example

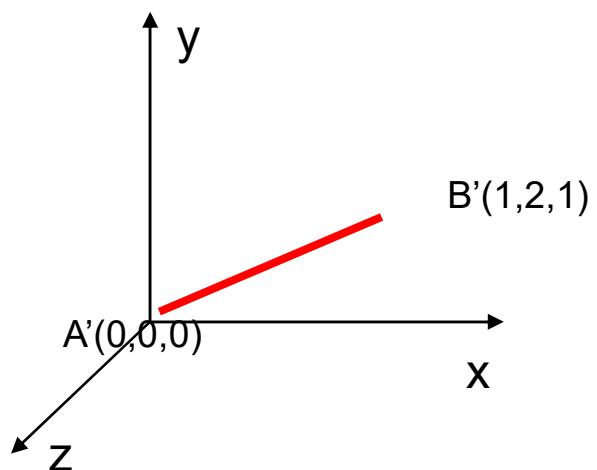
Find the new coordinates of a unit cube 90°-rotated about an axis defined by its endpoints A(2,1,0) and B(3,3,1).



A Unit Cube

Example

- Step1. Translate point A to the origin



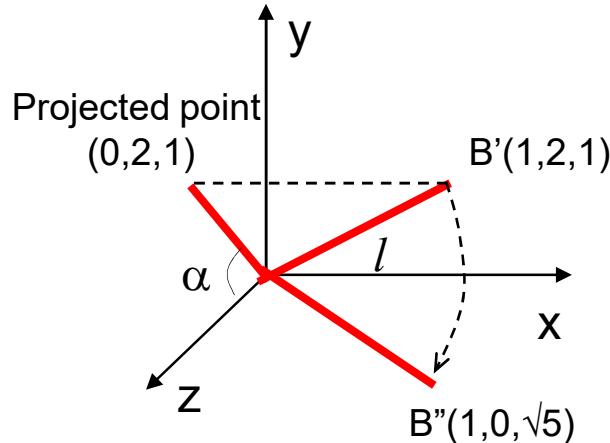
$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -2 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Example

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

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

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

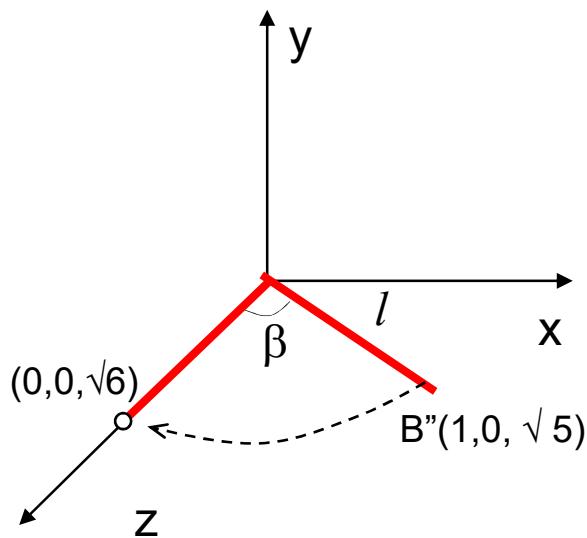


$$l = \sqrt{1^2 + 2^2 + 1^2} = \sqrt{6}$$

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

Example

- Step 3. Rotate axis $A'B''$ about the y axis by an 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,

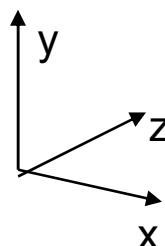
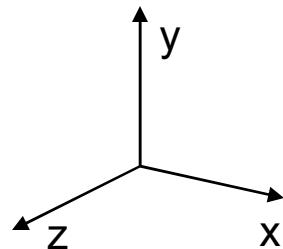
$$\mathbf{R}(\theta) = \mathbf{T}^{-1} \mathbf{R}_x^{-1}(\alpha) \mathbf{R}_y^{-1}(\beta) \mathbf{R}_z(90^\circ) \mathbf{R}_y(\beta) \mathbf{R}_x(\alpha) \mathbf{T}$$

Example

$$\begin{aligned}
 \mathbf{R}(\theta) &= \left[\begin{array}{cccc} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{5}}{5} & \frac{2\sqrt{5}}{5} & 0 \\ 0 & -\frac{2\sqrt{5}}{5} & \frac{\sqrt{5}}{5} & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] \left[\begin{array}{cccc} \frac{\sqrt{30}}{6} & 0 & \frac{\sqrt{6}}{6} & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{\sqrt{6}}{6} & 0 & \frac{\sqrt{30}}{6} & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] \left[\begin{array}{cccc} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] \\
 &\quad \left[\begin{array}{cccc} \frac{\sqrt{30}}{6} & 0 & -\frac{\sqrt{6}}{6} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{\sqrt{6}}{6} & 0 & \frac{\sqrt{30}}{6} & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{5}}{5} & -\frac{2\sqrt{5}}{5} & 0 \\ 0 & \frac{2\sqrt{5}}{5} & \frac{\sqrt{5}}{5} & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] \left[\begin{array}{cccc} 1 & 0 & 0 & -2 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] \\
 &= \left[\begin{array}{cccc} 0.166 & -0.075 & 0.983 & 1.742 \\ 0.742 & 0.667 & 0.075 & -1.151 \\ -0.650 & 0.741 & 0.167 & 0.560 \\ 0 & 0 & 0 & 1 \end{array} \right]
 \end{aligned}$$

Other Transformations

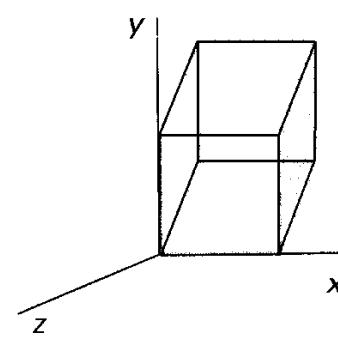
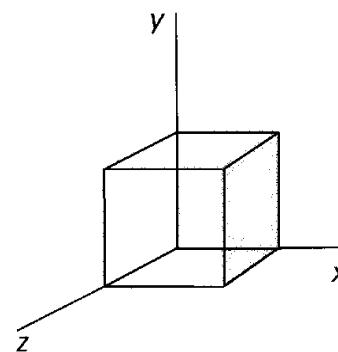
■ Reflection Relative to the xy Plane



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

■ Z-axis Shear

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



Thank You

Computer Graphics

Clipping

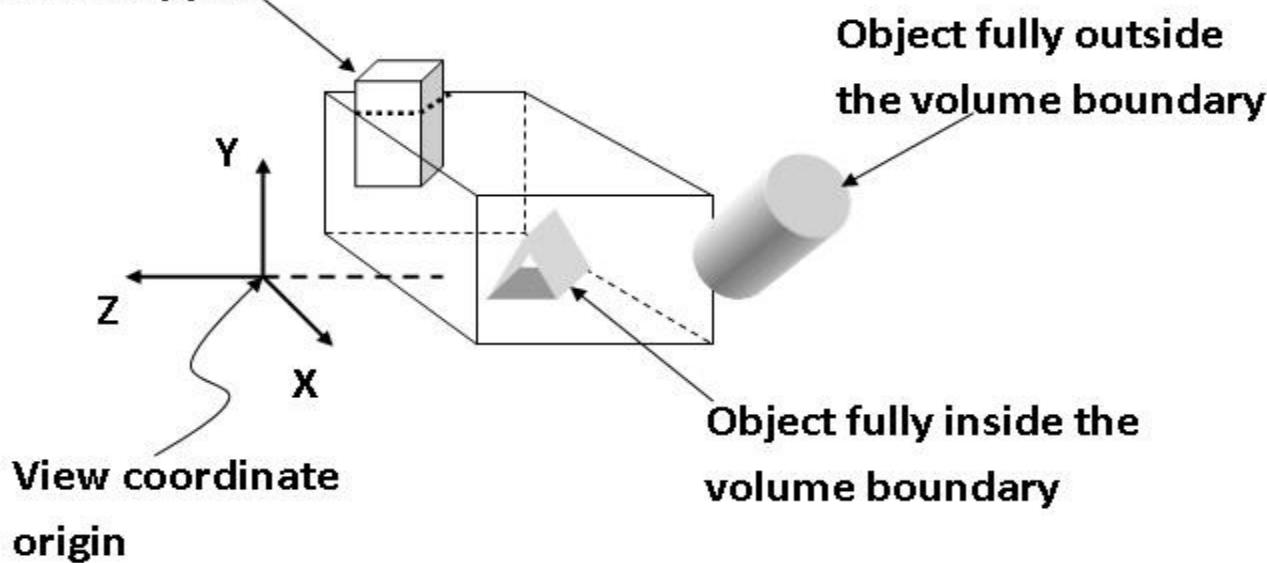
Lenin Laitonjam

NIT Mizoram

Clip Objects

- Objects that are partially within the viewing volume need to be clipped

Object partially inside the volume boundary. The portion that lies outside (above the dotted line) has to be clipped.



3D Clipping

- Many of the algorithms are extension of clipping in 2D
- We will have a look into the 2D clipping algorithms for
 - Point
 - Line
 - Polygon fill area

Clipping

Point Clipping

- Easy - a point (x,y) is not clipped if:

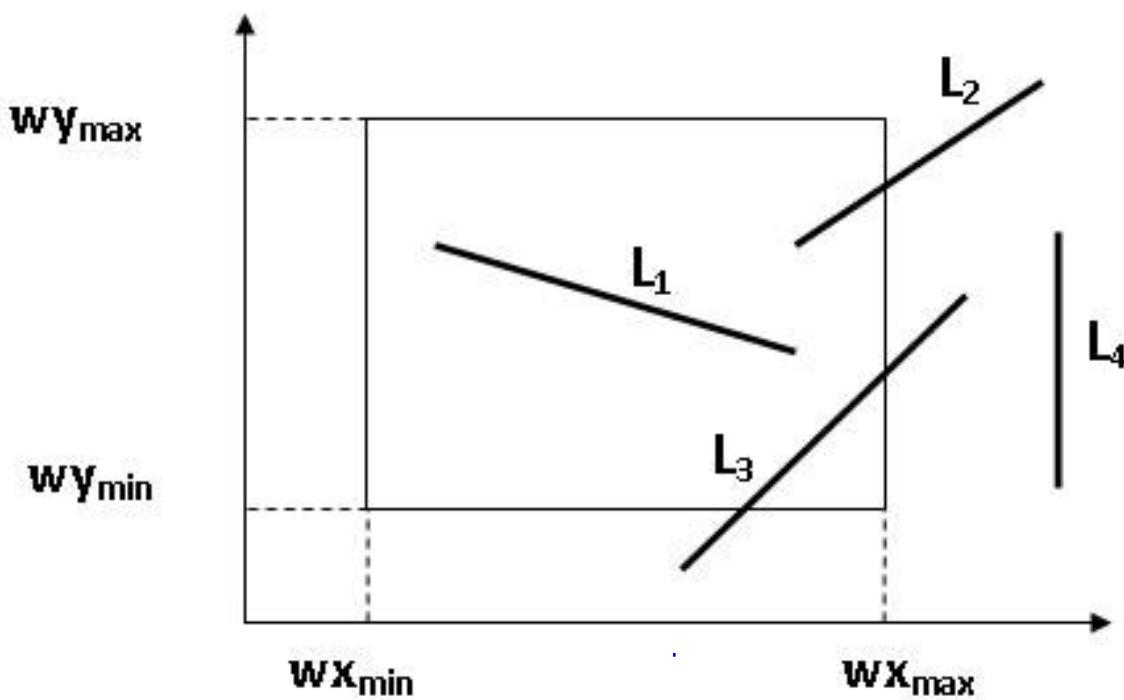
$$wx_{min} \leq x \leq wx_{max} \text{ AND } wy_{min} \leq y \leq wy_{max}$$

- Otherwise it is clipped

Clipping

Line Clipping

- Harder - examine the end-points of each line to see if they are in the window or not



Brute Force Line Clipping

- Brute force line clipping can be performed as follows:
 - Don't clip lines with both end-points within the window
 - For lines with one end-point inside the window and one end-point outside, calculate the intersection point (using the equation of the line) and clip from this point out
 - For lines with both end-points outside the window, test the line for intersection with all of the window boundaries, and clip appropriately

Clipping

Brute Force Line Clipping

- Calculating line intersections is computationally expensive
 - Because a scene can contain so many lines, the brute force approach to clipping is much too slow

Cohen-Sutherland Clipping Algorithm

- An efficient line clipping algorithm
- Key advantage: vastly reduces the number of line intersection calculation
- World space is divided into regions based on the window boundaries
 - Each region has a unique four bit region code
 - Region codes indicate the position of the regions with respect to the window

Above Left	Above	Above Right	1001	1000	1010
Left	Window	Right	0001	0000	0010
Below Left	Below	Below Right	0101	0100	0110

Above	Below	Right	Left
-------	-------	-------	------

The four bit code and significance of each bit

Region Code Assignment

,

- Bit 3 = sign ($y - y_{\max}$)
 - Bit 2 = sign ($y_{\min} - y$)
 - Bit 1 = sign ($x - x_{\max}$)
 - Bit 0 = sign ($x_{\min} - x$)
-
- Sign(a)=1 if a is positive, 0 otherwise

.

Cohen-Sutherland: Steps

- Both endpoint region code 0000, line is completely inside window – retain it
- Logical AND of both endpoints \neq 0000, line is completely outside – discard it entirely

Cohen-Sutherland: Steps

- For all other cases, do the following
 - Calculate line intersection point with window boundaries (follow some order for checking, e.g. Left, Right, Bottom, Top)
 - Line intersects a boundary if the corresponding bit value in the two region codes are not the same
 - Intersection points with the window boundaries are calculated using the line-equation

Cohen-Sutherland: Steps

- For all other cases, do the following
 - Assign region code to the intersection point and discard the line segment “outside” (w.r.t. the particular boundary)
 - Repeat till both endpoints are completely inside or completely outside of the window

Calculating Line Intersections

- Consider a line with the end-points (x_1, y_1) and (x_2, y_2)
 - The y-coordinate of an intersection with a vertical window boundary can be calculated using:

$$y = y_1 + m (x_{\text{boundary}} - x_1)$$

where x_{boundary} can be set to either wx_{\min} or wx_{\max}

- The x-coordinate of an intersection with a horizontal window boundary can be calculated using:

$$x = x_1 + (y_{\text{boundary}} - y_1) / m$$

where y_{boundary} can be set to either wy_{\min} or wy_{\max}

- m is the slope $= (y_2 - y_1) / (x_2 - x_1)$

Cohen-Sutherland Algorithm

- Better than brute force, but not the best
- Works well when the number of lines which can be clipped without further processing is large compared to the size of the input set
 - Still checks for some lines that are completely outside
- Liang-Barsky algorithm
 - Parametric line-clipping algorithm
 - Reduces intersection calculation further than Cohen-Sutherland

Fill-Area Clipping

- To clip a polygon fill area, we cannot directly apply a line-clipping method to the individual polygon edges
 - Line clipping may not produce a closed polyline
- Other efficient algorithms are available
 - Sutherland-Hodgeman
 - Weiler-Atherton

Clipping

Sutherland-Hodgeman Polygon Clipping

- Basic idea
 - Four “clippers” - each corresponding to one of the clipping edges (window boundaries)
 - Left, Right, Bottom, Top
 - Each clipper takes as input a list of ordered pairs of vertices (edges) – produces another list of vertices as output

Sutherland-Hodgeman Polygon Clipping

- Basic idea
 - The original polygon vertices are given as input to the first clipper (usually Left)
 - Follow some clipper order for checking, e.g. Left → Right → Bottom → Top
 - Follow some vertex naming convention (clockwise/anti-clockwise)

Sutherland-Hodgeman Polygon Clipping

- $V = \{v_1, v_2, \dots, v_n\}$ denotes the vertices (assume anti-clockwise naming).
- Then, each successive vertex pair (v_i, v_j) represent an edge.
- For each clipper, the output is generated in the following way
 - Do for each input edge (vertex pair (v_i, v_j))
 - $v_i = \text{inside}, v_j = \text{outside}$; *return* intersection point
 - $v_i = \text{inside}, v_j = \text{inside}$; *return* v_j
 - $v_i = \text{outside}, v_j = \text{inside}$; *return* intersection point and v_j
 - $v_i = \text{outside}, v_j = \text{outside}$; *return* NULL

Sutherland-Hodgeman Polygon Clipping

- When a concave polygon is clipped with the Sutherland-Hodgeman algorithm, extraneous lines may be displayed
- Since there is only one output vertex list, the last vertex in the list is always joined to the first vertex

Weiler-Atherton Polygon Clipping

- Can be used to clip a fill area that is either a convex polygon or a concave polygon
- Basic idea - instead of always proceeding around polygon edges as vertices are processed, sometimes follow window boundaries
 - A boundary is followed whenever a polygon edge crosses to the outside of that boundary

Weiler-Atherton Polygon Clipping

- Two rules
 - For an outside-to-inside vertex pair, follow polygon edges
 - For an inside-to-outside vertex pair, follow window boundary
- The direction of vertex traversal and window boundary traversal should be the same – clockwise/counter-clockwise
 - Linked to vertex naming convention

Weiler-Atherton Polygon Clipping

- Steps (assume anti-clockwise traversal) – Repeat till all the vertices are processed
 - Process vertices in anti-clockwise order *until* an inside-outside pair of vertices is encountered for one of the clipping boundaries
 - Follow window boundaries in anti-clockwise direction from the exit-intersection point to another intersection point *already seen*
 - Form the vertex list for this section of the clipped fill area

Weiler-Atherton Polygon Clipping

- Steps (assume anti-clockwise traversal) – Repeat till all the vertices are processed
 - Return to the exit-intersection point and continue processing the polygon edges in anti-clockwise order

Thank You

Filling and Anti-aliasing

Lenin Laitonjam

Department of Computer Science and Engineering
NIT Mizoram

Fill-Area Scan Conversion

- *Region filling*: “coloring in” a definite image area or region
- Definition at pixel or geometric level
- Pixel level definitions
 - Boundary defined: region defined in terms of boundary pixels
 - Interior defined: region defined in terms of all the pixels within the interior
- Geometric region (usually for polygons): region defined in terms of edges and vertices

Seed Fill Algorithm

- Assume at least one pixel interior to a polygon or region is known – called *seed*
- Regions boundary defined
 - For interior defined regions – flood-fill algorithm
- Two conventions
 - 4-connected: each pixel connected to four adjacent pixels (Top, Bottom, Left, Right)
 - 8-connected: each pixel connected to eight adjacent pixels (Top, Top Left, Top Right, Bottom, Bottom Left, Bottom Right, Left, Right)

A Simple Seed Fill Algorithm

- *Push the seed pixel onto the stack*
- While the stack is not empty
 - Pop a pixel from the stack
 - Set the pixel to the required value
 - For each of the 4 connected pixels adjacent to the current pixel
 - If it is a boundary pixel or if it has already been set to the required value, ignore it
 - Else push it onto the stack
- Easy to modify for 8-connected pixels
 - It also works with holes in the polygons

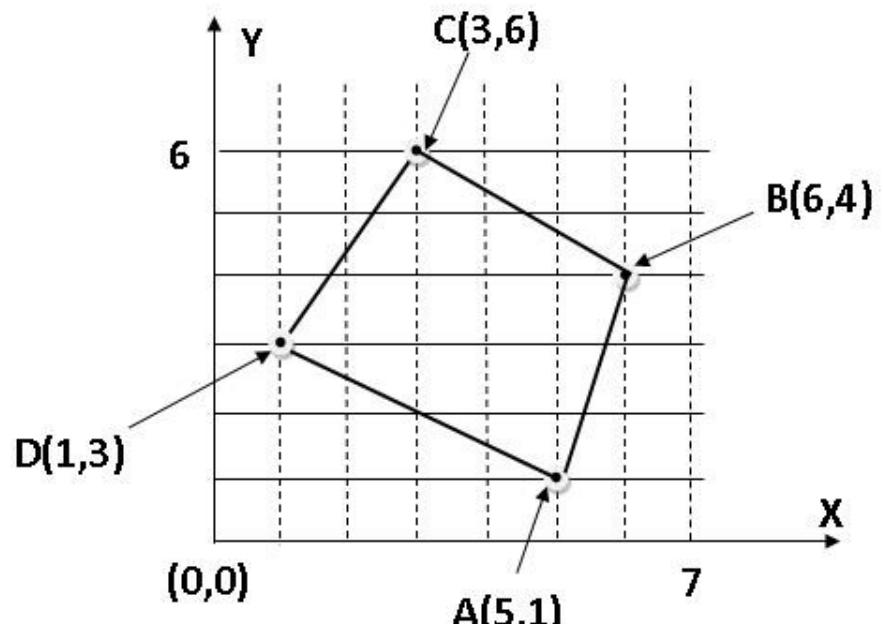
Flood Fill Algorithm

- Idea similar to seed fill
- Difference: while coloring, we take into account the interior color rather than the boundary color
- Other things remain the same

Scan-Line Polygon Fill

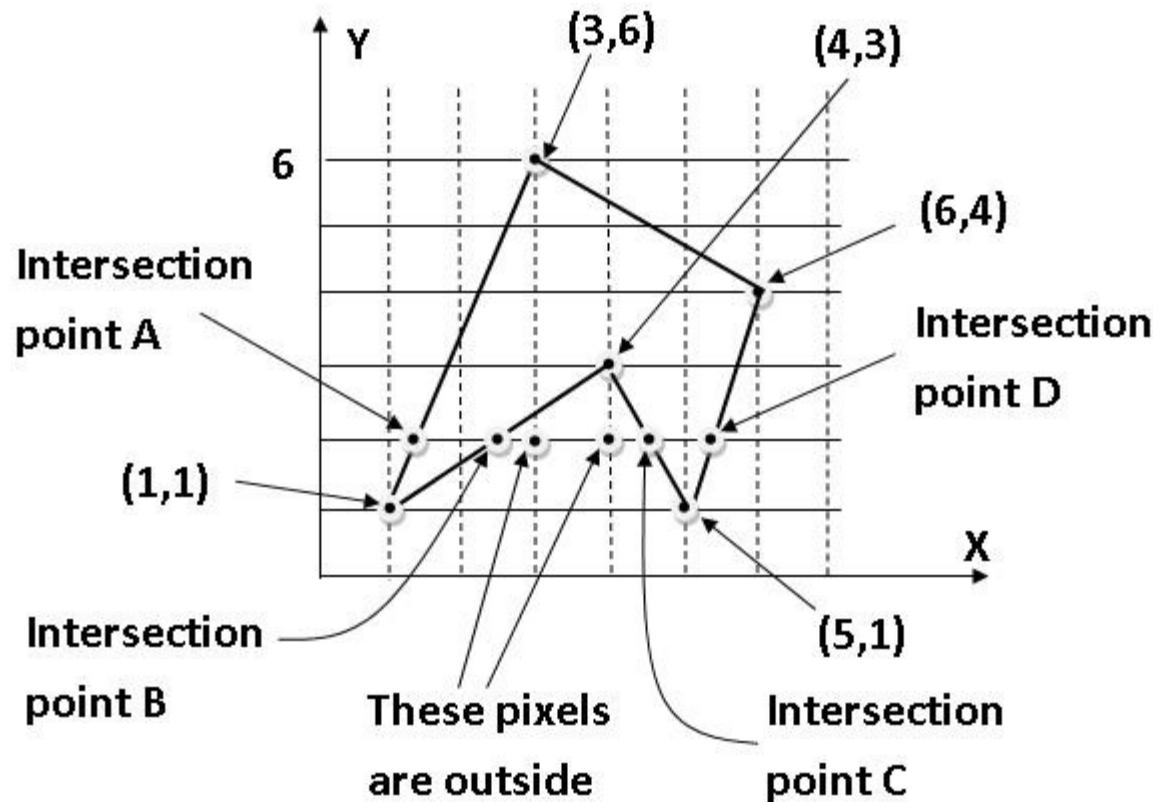
Do the following for every scan line:

1. Compute the intersection of the current scan line with every polygon edge
2. Sort the intersections in increasing order of the x coordinate
3. Draw every pixel that lies between each pair of intersections



Problem

- What will happen in case of concave polygons



A Simple Inside-Outside Test

- Suppose we want to know if a point P is inside
 - Determine the bounding box (max and min x and y extents)
 - Choose a point P' outside the bounding box
 - Join P and P'
 - If the line intersects the polygon edges even number of times, P is outside. Else P is inside

Character Rendering

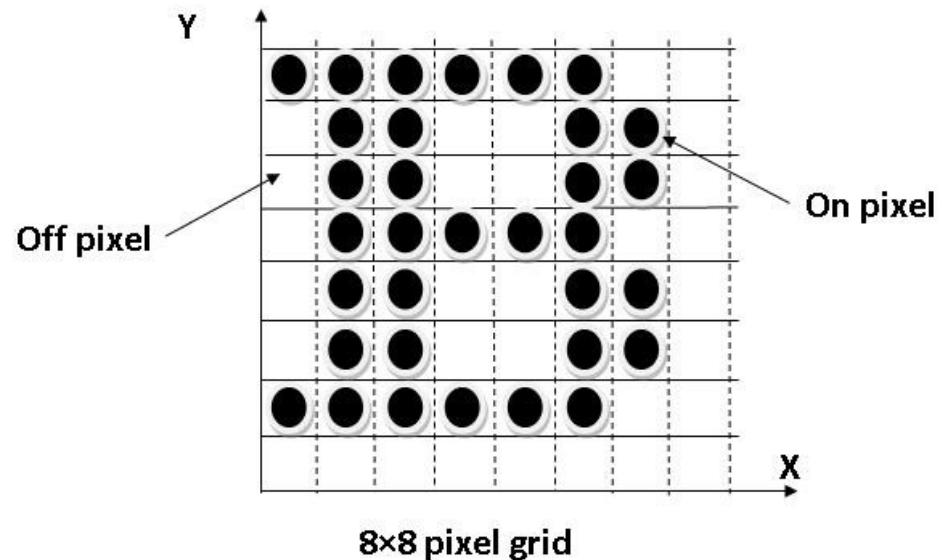
- Letters, digits, non-alphanumeric
- Terms borrowed from typography
 - Typeface: a particular style of characters (Times New Roman, Courier, Arial)
 - Font: cast metal character form to print typeface
 - In CG, the terms are used synonymously
- Fonts
 - Two broad types: Serif, Sans-Serif
 - Can vary in appearance: Normal, bold, italic
- Rendering techniques
 - Bitmap, Outlined

About “Point”

- Font size usually denoted in point (e.g. 10-point, 12-point)
 - Denotes height of the characters in inches
- A term from typography
 - Smallest unit of measure
- We are concerned with desk-top publishing (DTP) point, also called the PostScript point
 - Not the original typographical point
- 1 DTP point = 1/72 of an inch or approx 0.0139 inch

Bitmapped Fonts

- Represents each character as the *on* pixels in a bi-level pixel grid pattern known as *bitmap*
 - Advantages
 - Simple
 - Fast, since the characters are defined in already scan converted form, no further processing is required

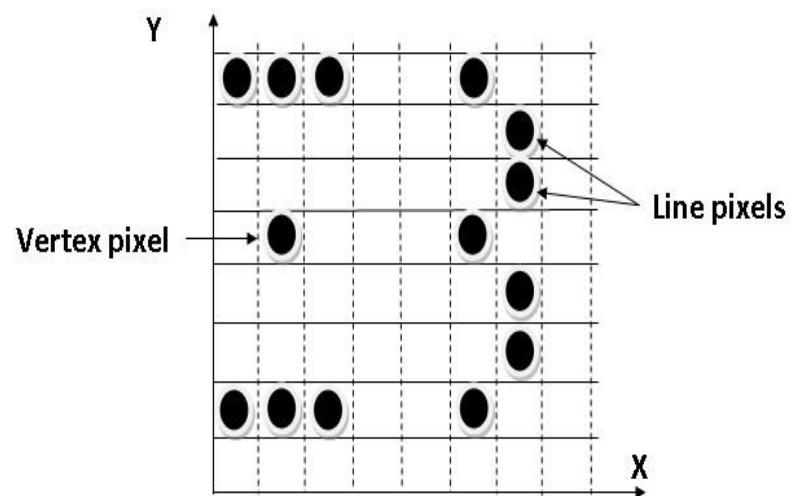


Bitmapped Fonts

- Disadvantages
 - More storage: for each character, we need to store the bitmap
 - Although different style/sizes can be generated from one font, the result is not satisfactory
 - Bitmap font size dependent on resolution (e.g. a 12 pixel high bitmap will produce a 12-point character in a 72 pixels/inch resolution, while the same bitmap will produce 9-point character in a 96 pixels/inch resolution)

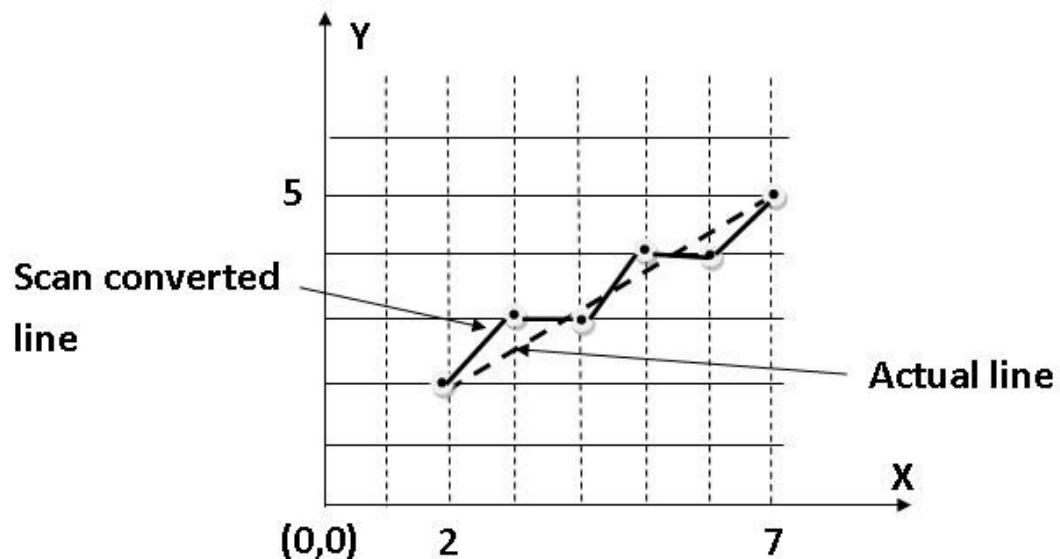
Outlined Fonts

- Character outline is defined using graphical primitives (e.g. line, arcs)
 - PostScript by Adobe
 - Less storage (no need to store bitmaps any more)
 - Good for styles/sizes
 - Scaling transformation to resize
 - Shearing transformation to italicize etc
 - Slower (since scan conversion is involved)



Anti-aliasing

- Aliasing – distortion introduced to the image due to the mapping from continuous to discrete space during scan conversion
 - Anti-aliasing – techniques to overcome the effect of aliasing



Aliasing & Signal Processing

- True intensity treated as continuous signal composed of various frequencies
 - Image as a *continuous signal*
- We need to *sample* it
 - Need some sort of *discrete sampling* technique
- Once we have our sampled signal, we then *reconstruct* it
 - In CG, this reconstruction takes place as a bunch of colored pixels on a monitor
- However, the reconstructed signals are a false representation of the original signals

Aliasing & Signal Processing

- In English, when a person uses a false name, that is known as an *alias*, and so it was adapted in *signal analysis* to apply to falsely represented signals

Anti-aliasing

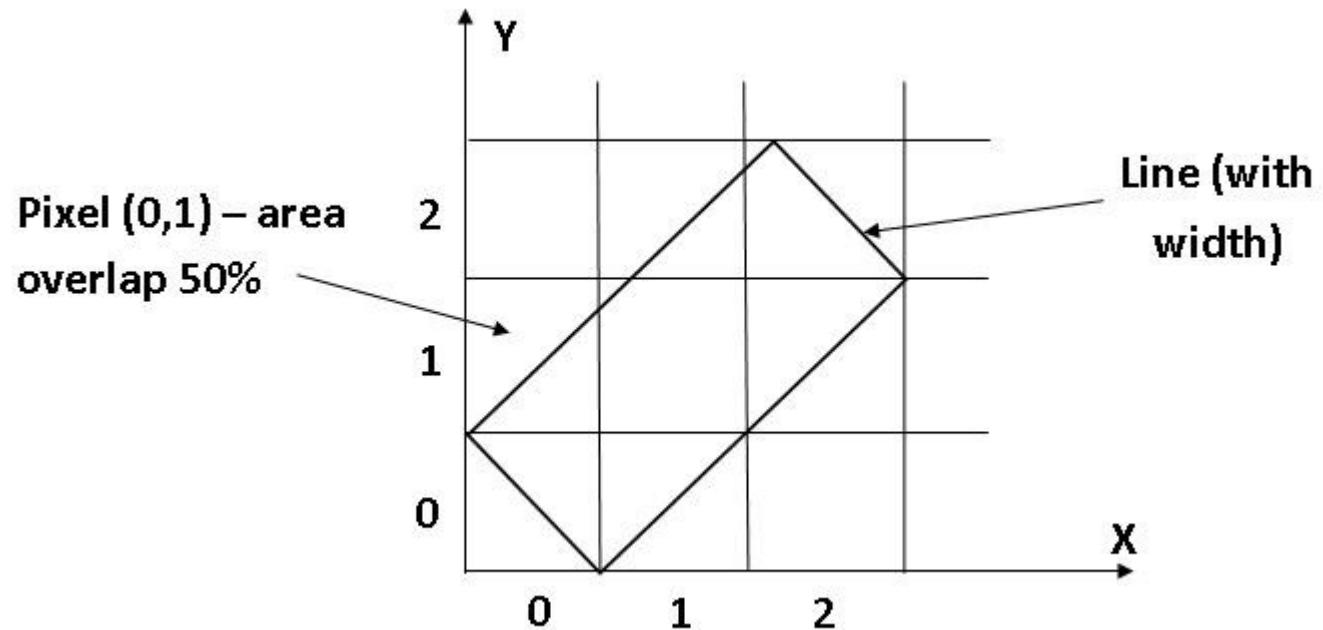
- Continuous intensity signal may be viewed as a composition of various frequency components
 - Uniform regions of constant intensity values correspond to the low frequency components
 - Intensity values that change abruptly and correspond to a sharp edge are at the high end of the frequency spectrum
- To lessen jagged contours, we need to smooth out sudden intensity changes
 - Filter out high frequency components

Anti-aliasing

- General purpose techniques
 - Pre-filtering (area sampling)
 - Post filtering (super-sampling)

Un-weighted Area Sampling

- Set each pixel intensity proportional to the area of overlap of pixel



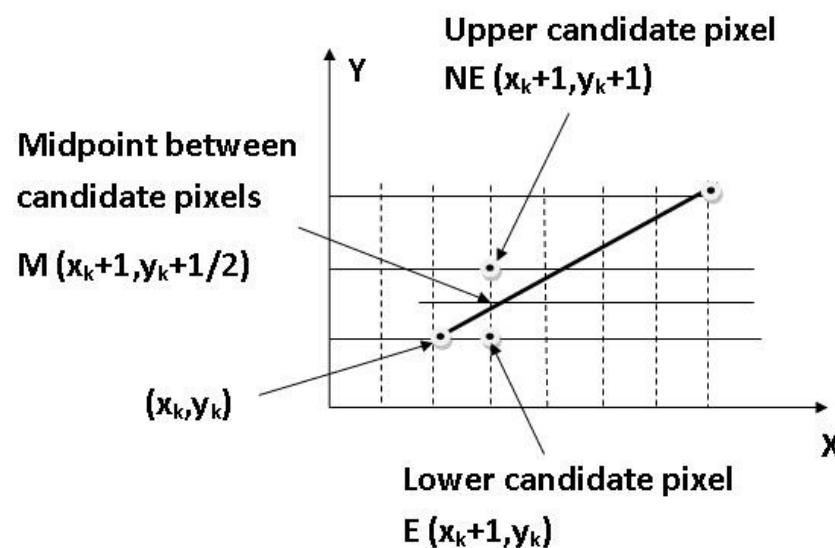
Foreground and Background

- Compute percent of pixel covered by line p
- Line color is c_l
- Background color is c_b
- Pixel color is computed as,

$$color = p \ c_l + (1-p) \ c_b$$

Gupta-Sproull Algorithm

- A pre-filtering technique
- Calculate pixel intensity by computing distance from pixel center to line using the midpoint line algorithm



Midpoint Line Algorithm

- Decision variable

$$\begin{aligned}d &= F(M) \\&= F\left(x_p + 1, y_p + \frac{1}{2}\right) \\&= a(x_p + 1) + b\left(y_p + \frac{1}{2}\right) + c\end{aligned}$$

- $d > 0$: choose NE

$$d_{new} = F\left(x_p + 2, y_p + \frac{3}{2}\right): \quad d_{new} = d_{old} + a + b$$

- $d \leq 0$: choose E

$$d_{new} = F\left(x_p + 2, y_p + \frac{1}{2}\right) \quad d_{new} = d_{old} + a$$

Midpoint Line Algorithm

- Initial Value of d

$$F\left(x_0 + 1, y_0 + \frac{1}{2}\right) = a(x_0 + 1) + b\left(y_0 + \frac{1}{2}\right) + c$$

$$= F(x_0, y_0) + a + \frac{1}{2}b$$

$$\rightarrow F(x, y) = 2(ax + by + c)$$

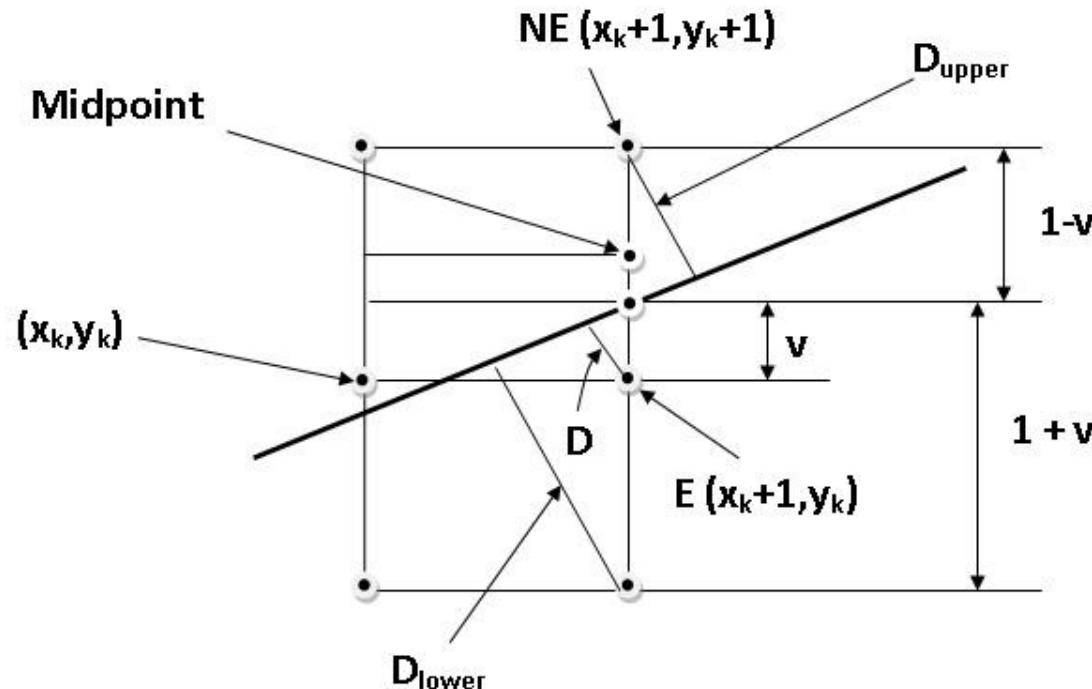
$$\rightarrow d = 2a + b$$

- Update d

$$\text{if } d > 0 \text{ then } \begin{cases} x++ \\ y++ \\ d+ = 2(a+b) \end{cases} \quad \text{if } d \leq 0, \text{ then } \begin{cases} x++ \\ d+ = 2a \end{cases}$$

Gupta-Sproull Algorithm

- Let E is selected by the mid point algorithm
 - D is the perpendicular distance from E to the line
- How do we compute it?



$$\Delta x = x_{end} - x_{start}$$

$$\Delta y = y_{end} - y_{start}$$

$$\cos \theta = \frac{\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}}$$

$$\cos \theta = \frac{D}{v}$$

$$D = \frac{v \Delta x}{\sqrt{\Delta x^2 + \Delta y^2}}$$

Gupta-Sproull Algorithm

Recall from the midpoint algorithm:

$$f(x, y) = 2(ax + by + c) = 0$$

So, $y = -\frac{ax + c}{b}$

and

$$y_v = -\frac{a(x_p + 1) + c}{b}$$

$$\nu = y_v - y_p$$

Therefore

$$\nu = -\frac{a(x_p + 1) + c}{b} - y_p$$

Gupta-Sproull Algorithm

So $-bv = a(x_p + 1) + c + by_p$

From the midpoint computation,

$$b = -\Delta x$$

So:

$$v\Delta x = a(x_p + 1) + by_p + c = \frac{1}{2} f(x_p + 1, y_p)$$

From the midpoint algorithm, we have the decision variable

$$d = f(m) = f(x_{p+1}, y_p + \frac{1}{2})$$

Gupta-Sproull Algorithm

Going back to our previous equation:

$$\begin{aligned}2v\Delta x &= f(x_p + 1, y_p) \\&= 2a(x_p + 1) + 2by_p + 2c \\&= 2a(x_p + 1) + 2b(y_p + \frac{1}{2}) - 2b\frac{1}{2} + 2c \\&= f(x_p + 1, y_p + \frac{1}{2}) - b \\&= f(m) - b \\&= d - b \\&= d + \Delta x\end{aligned}$$

Gupta-Sproull Algorithm

So,

$$D = \frac{v\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}} = \frac{d + \Delta x}{2\sqrt{\Delta x^2 + \Delta y^2}}$$

The denominator is constant.

Since we are blurring the line, we also need to compute the color at the pixels above and below the E pixel

$$D_{up} = \frac{(1-v)\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}}$$

$$D_{lower} = \frac{(1+v)\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}}$$

Gupta-Sproull Algorithm

If the NE pixel had been chosen:

$$\begin{aligned}2v\Delta x &= f(x_p + 1, y_p + 1) \\&= 2a(x_p + 1) + 2b(y_p + 1) + 2c \\&= 2a(x_p + 1) + 2b(y_p + \frac{1}{2}) + 2b\frac{1}{2} + 2c \\&= f(x_p + 1, y_p + 1/2) + b \\&= f(m) + b \\&= d + b \\&= d - \Delta x\end{aligned}$$

$$D = \frac{v\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}} = \frac{d - \Delta x}{2\sqrt{\Delta x^2 + \Delta y^2}} \quad D_{up} = \frac{(1-v)\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}} \quad D_{lower} = \frac{(1+v)\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}}$$

Gupta-Sproull Algorithm Summary

- Compute midpoint line algorithm, with the following alterations at each iteration:
- At each iteration of the algorithm:

- If the E pixel is chosen

$$D = \frac{d + \Delta x}{2\sqrt{\Delta x^2 + \Delta y^2}}$$

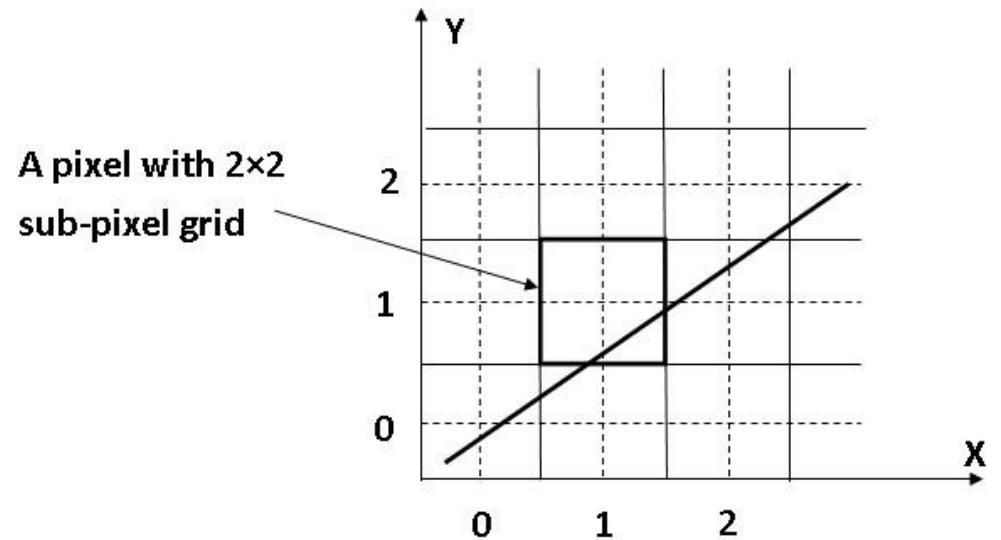
- If the NE pixel is chosen

$$D = \frac{d - \Delta x}{2\sqrt{\Delta x^2 + \Delta y^2}}$$

- Update d as in the regular algorithm
 - Color the current pixel according to D
 - Compute $D_{up} = \frac{(1-v)\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}}$ $D_{lower} = \frac{(1+v)\Delta x}{\sqrt{\Delta x^2 + \Delta y^2}}$
 - Color upper and lower pixels accordingly

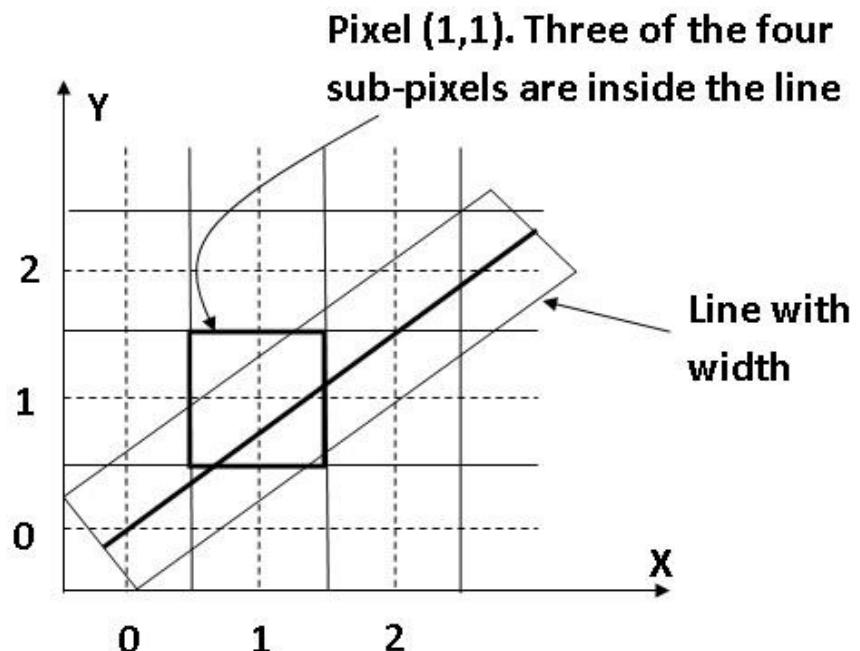
Super-sampling

- Divide pixel up into “sub-pixels”: 2×2 , 3×3 etc.
 - Increase resolution
- Count no of sub-pixels through which the line passes
 - Set pixel intensity proportional to this count



Super-sampling

- We can also consider lines having finite width
 - (usually 1-pixel wide)
- Sub-pixel is colored if inside line
 - Lower left corner of sub-pixel inside
- Pixel color = average of its sub-pixel colors



Pixel-Weighting Masks

- Give more weight to sub-pixels near the center of a pixel area
 - Distribute weights

1	2	1
2	4	2
1	2	1

Example:

- Central sub-pixel has 4 times more weight than corner ones.
- While calculating pixel intensity, the central sub-pixel intensity contribution $1/4^{\text{th}}$, $1/8^{\text{th}}$ each for the top, bottom, left and right sub-pixel intensities and $1/16^{\text{th}}$ each for the rest (corner sub-pixels)
- It's possible to include contribution from neighboring sub-pixels (extend the grid)

Thank You

Computer Graphics

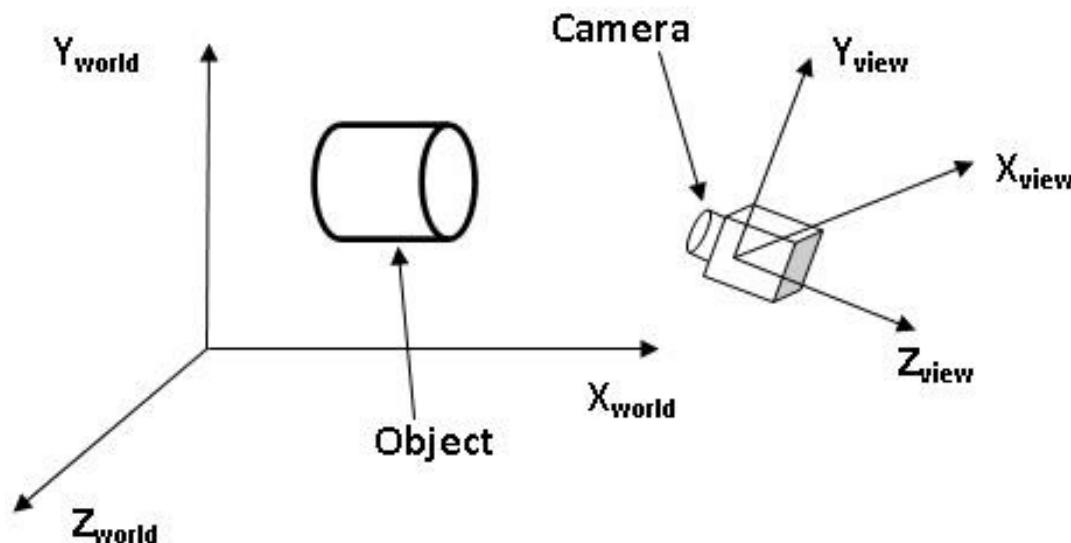
3-D Viewing

Lenin Laitonjam

NIT Mizoram

3D Viewing

- Just like taking a photograph!
- World coordinates to Viewing coordinates:
Viewing transformations



Camera Parameters

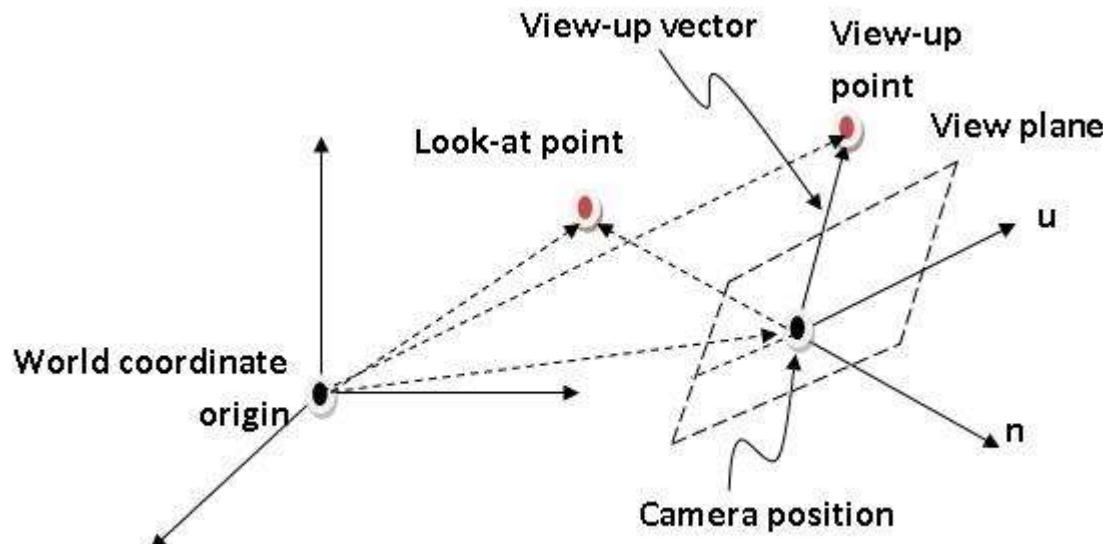
- Important camera parameters to specify
 - Camera (eye) position in world coordinate system
 - Also called *viewpoint/viewing position*
 - Center of interest
 - Also called *look-at point*
 - Orientation (which way is up?) View-up vector

View Coordinate Frame

- Known: eye position, center of interest, view-up vector
- To find out: new origin and three basis vectors
 - Assumption: the direction of view is orthogonal to the view plane (the plane that objects will be projected onto)

View Coordinate Frame

- Origin: camera position
- Three basis vectors: one is the normal vector (\mathbf{n}) of the viewing plane, the other two are the ones (\mathbf{u} and \mathbf{v}) that span the viewing plane



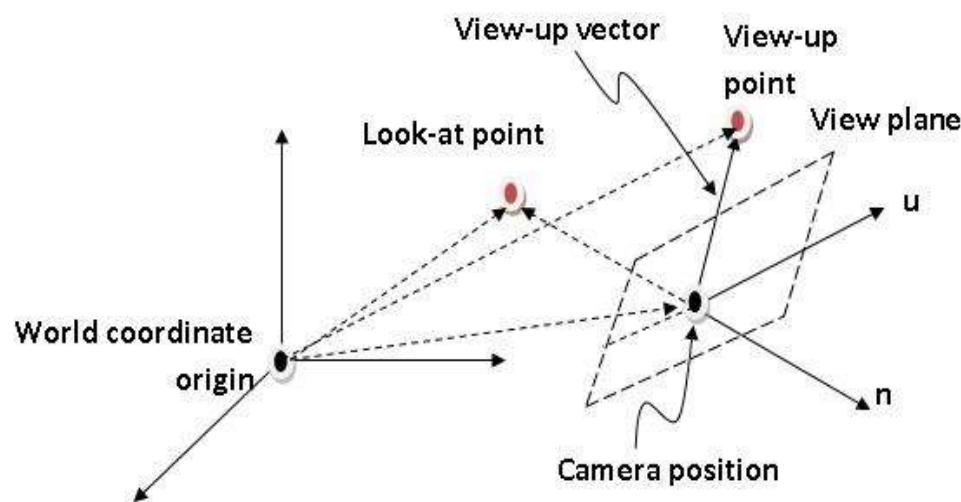
\mathbf{n} is pointing away from the world because we use right hand coordinate system
 $\mathbf{N} =$

$$\mathbf{n} = \mathbf{N} / |\mathbf{N}|$$

Remember $\mathbf{u}, \mathbf{v}, \mathbf{n}$ should all be unit vectors

View Coordinate Frame

- What about \mathbf{u} and \mathbf{v} ?



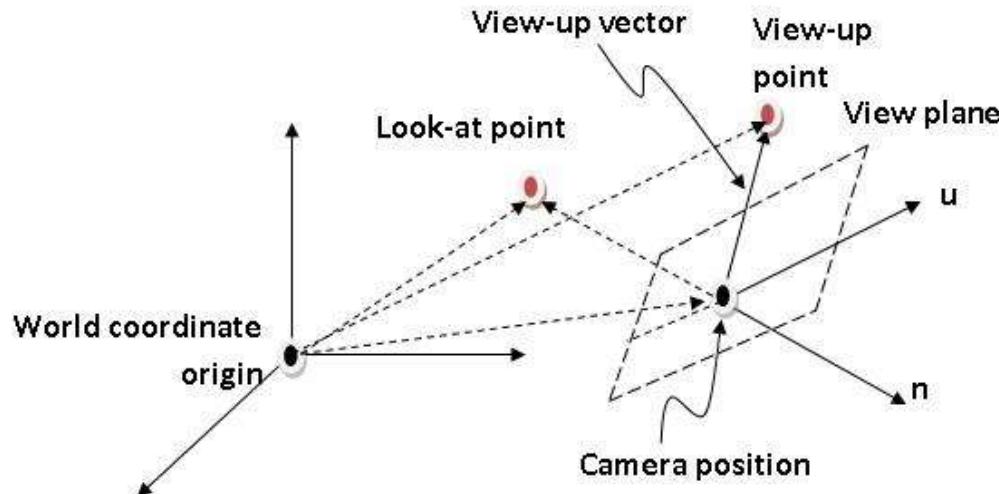
We can get \mathbf{u} first -

\mathbf{u} is a vector that is perpendicular to the plane spanned by \mathbf{n} and view up vector (V_{up})

View Coordinate Frame

- How about \mathbf{v} ?

Knowing \mathbf{n} and \mathbf{u} , getting \mathbf{v} is easy

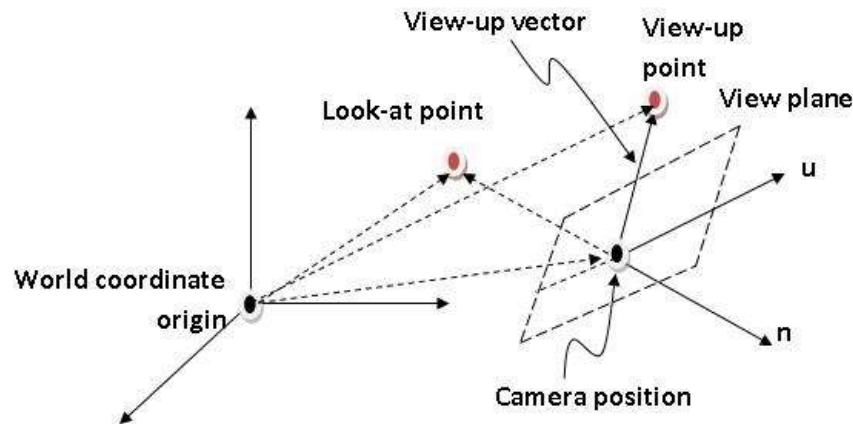


$$\mathbf{v} = \mathbf{n} \times \mathbf{u}$$

\mathbf{v} is already normalized

View Coordinate Frame

- Put it all together



Eye space origin: (ex, ey, ez)

Basis vectors:

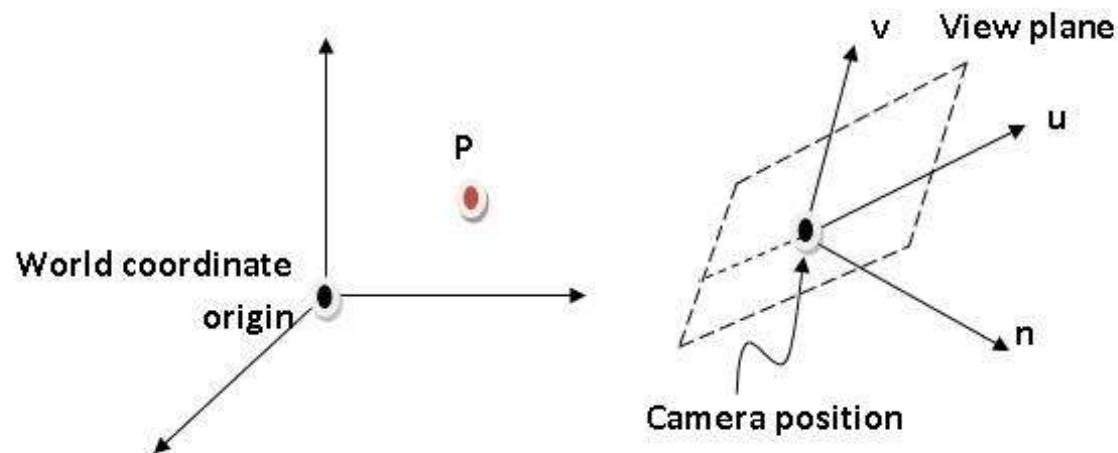
$$\mathbf{n} =$$

$$\mathbf{u} =$$

$$\mathbf{v} =$$

World to VC Transformation

- Transform object description from WC to VC
 - Equivalent of transformation between coordinate systems
- Transformation matrix (M_{w2e}); $P' = M_{w2v} \cdot P$
 - Come up with the transformation sequence to move view coordinate frame to the world
 - Apply this sequence to the point P in a reverse order



WC to VC Transformation

- Sequence: translate, rotate

$$M_{W2V} = \begin{bmatrix} ux & uy & uz & 0 \\ vx & vy & vz & 0 \\ nx & ny & nz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -ovx \\ 1 & 0 & 0 & -ovy \\ 1 & 0 & 0 & -ovz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Generating 3D Viewing Effects

- By varying the viewing parameters
 - Composite display consisting of multiple views from a fixed camera position
 - Fixed eye position, change \mathbf{n}
 - Simulate animation panning effect
 - \mathbf{n} fixed, change eye position
 - To view an object from different positions
 - Move the eye position around the object

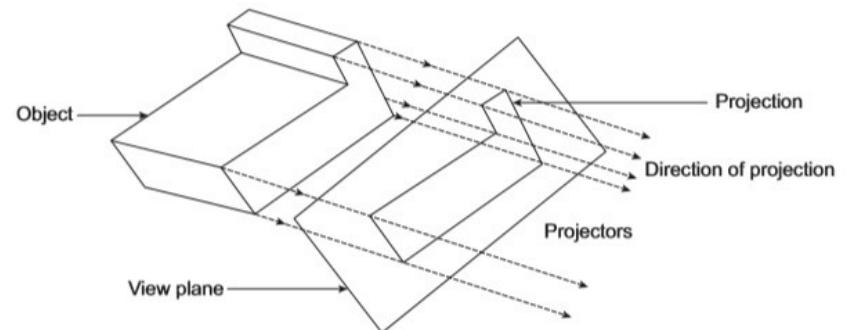
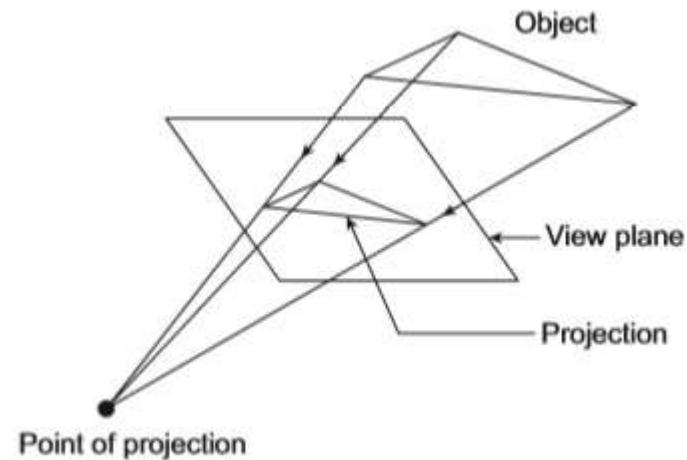
Projection

Projection

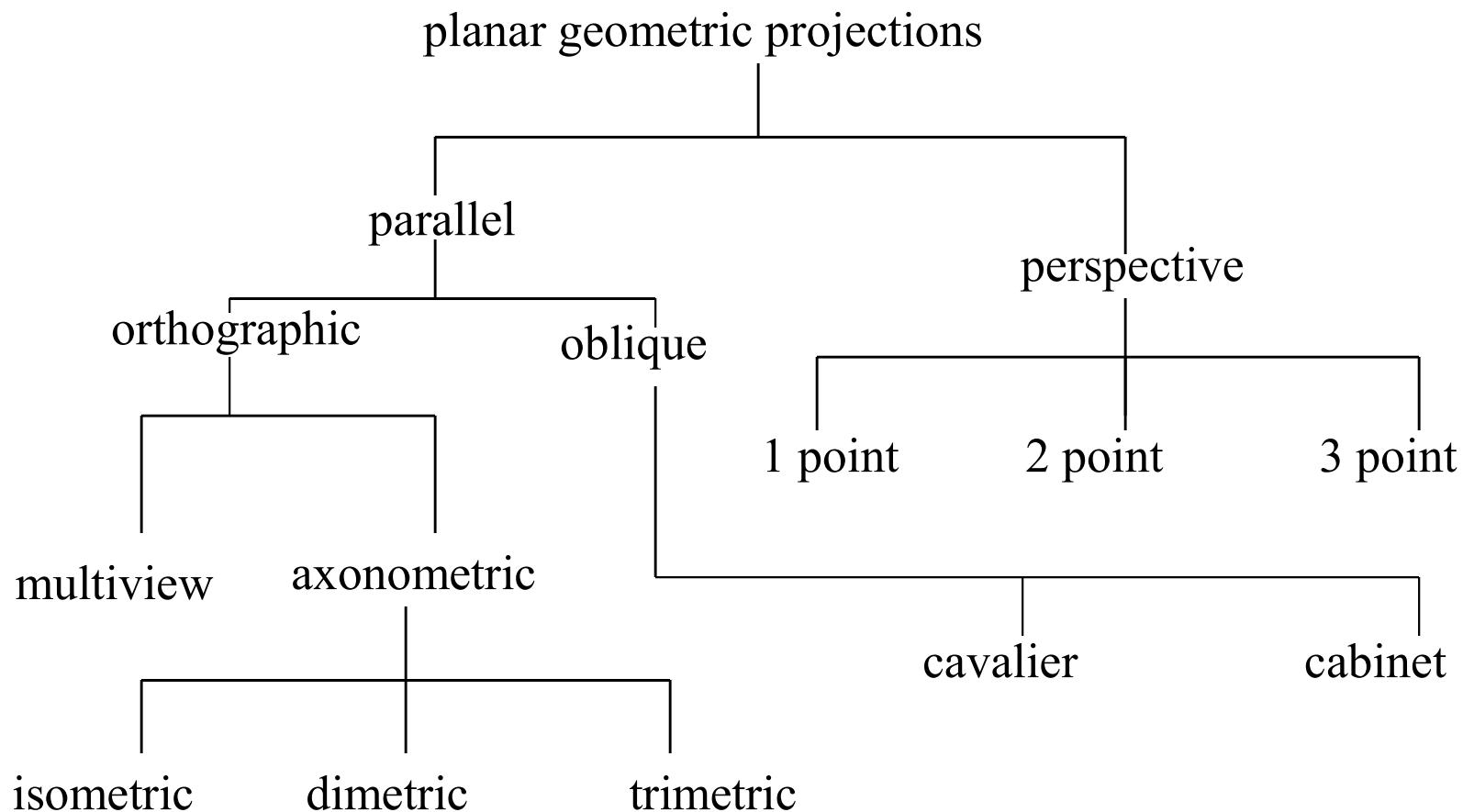
- Map objects from 3D space to 2D screen
 - Defined by straight lines called *projectors*
- Graphics deals with
 - Planar projections (projection surface is a plane)
 - Geometric projections – the projectors are straight lines

Planar Geometric Projections

- Projectors are lines that either
 - Converge at a center of projection (perspective projection)
 - Are parallel (parallel projection) – center of projection at infinity

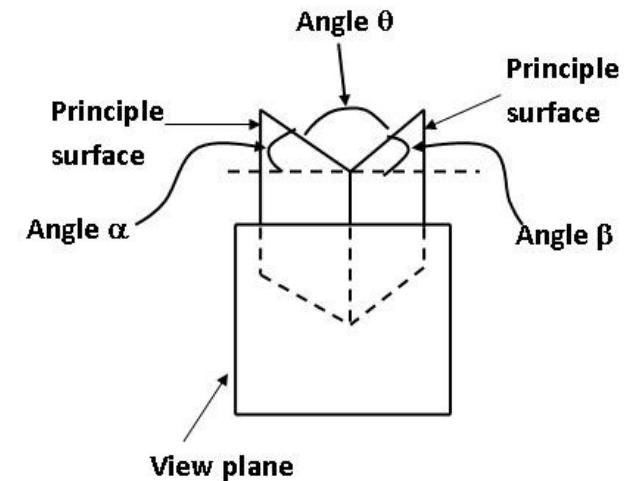
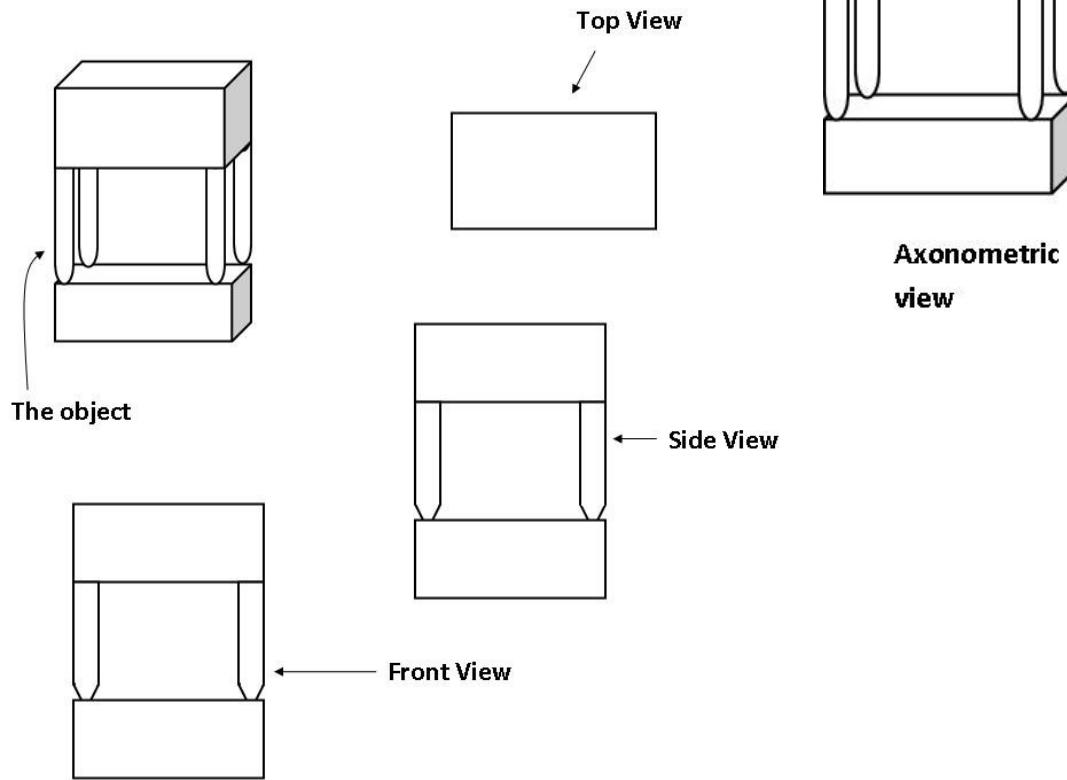


Projection Taxonomy



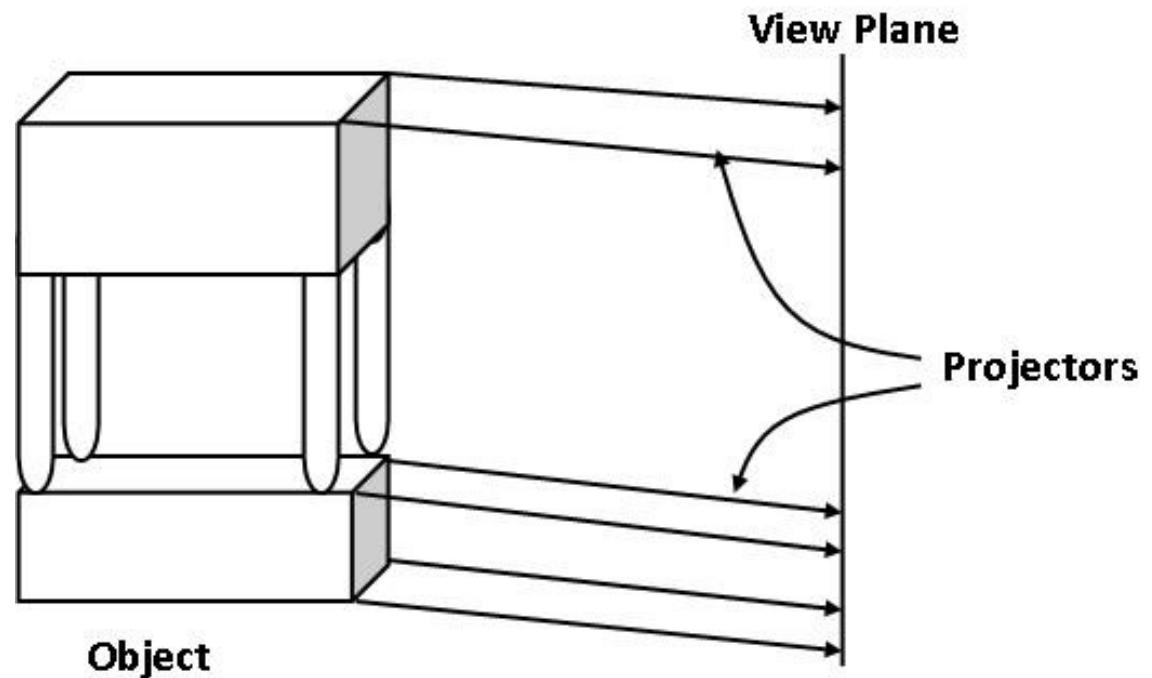
Parallel Projection

- Orthographic



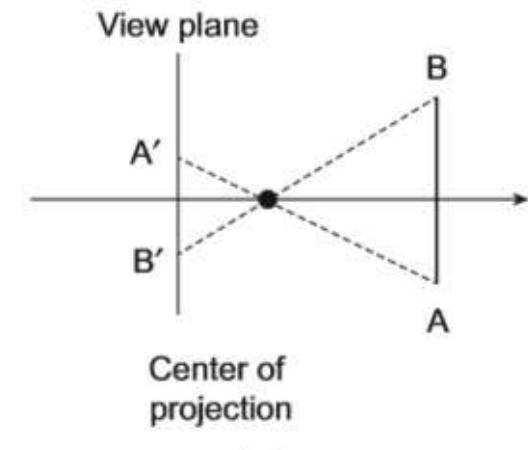
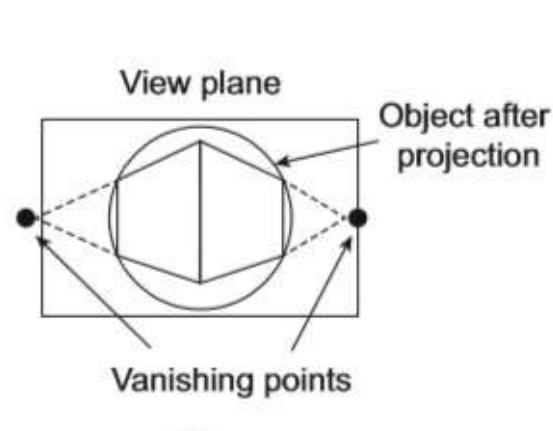
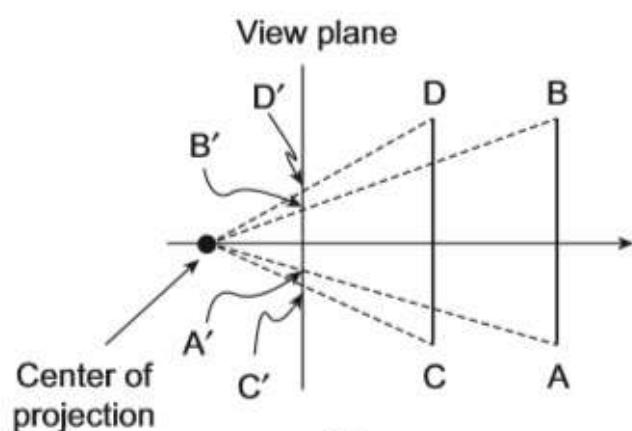
Parallel Projection

- Oblique



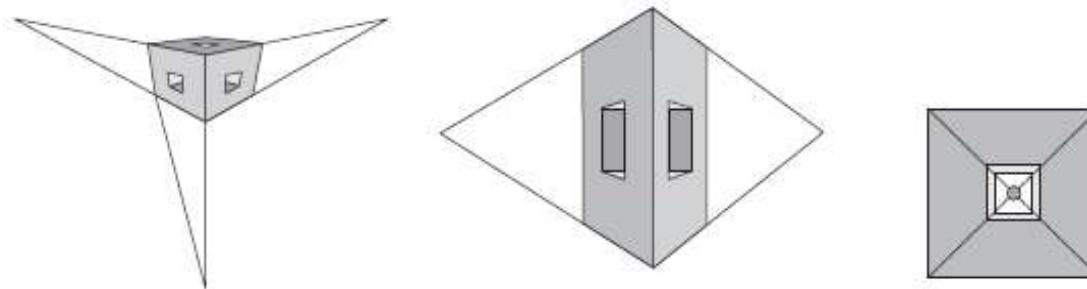
Perspective Projection

- Projectors converge at center of projection
- gives rise to anomalies
 - perspective foreshortening
 - vanishing points
 - view confusion



Perspective Projection

- One-point, two-point, three-point

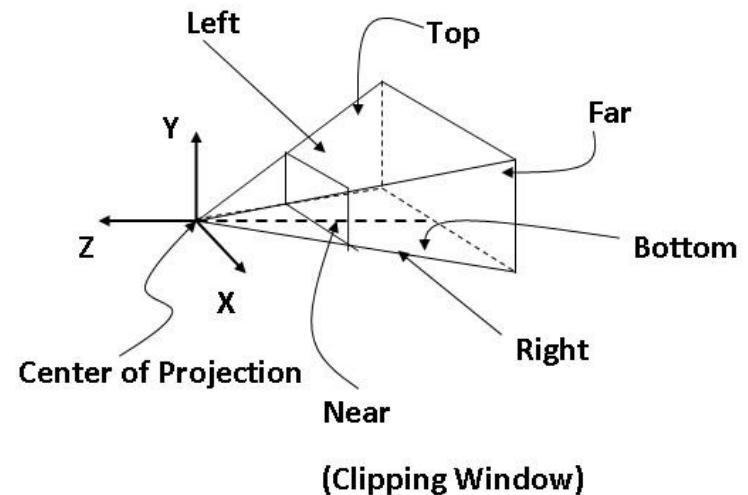
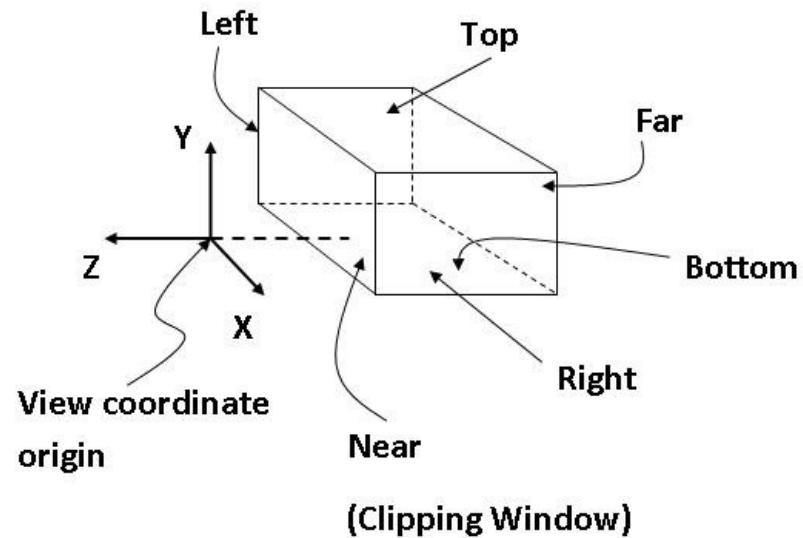


Projection Transformation

- Once WC \rightarrow VC transformation is done, the 3D objects are projected on the 2D view plane
- Important Considerations
 - Clipping window – an area on the view/projection plane, which will contain the projected points
 - View volume – a region in the 3D space that contains the objects to be projected
 - Note that we don't project the entire 3D scene

Projection Transformation

- Important things to control (to define view volume)
 - Parallel projection – view volume is rectangular parallelepiped
 - Perspective projection – view volume is a frustum



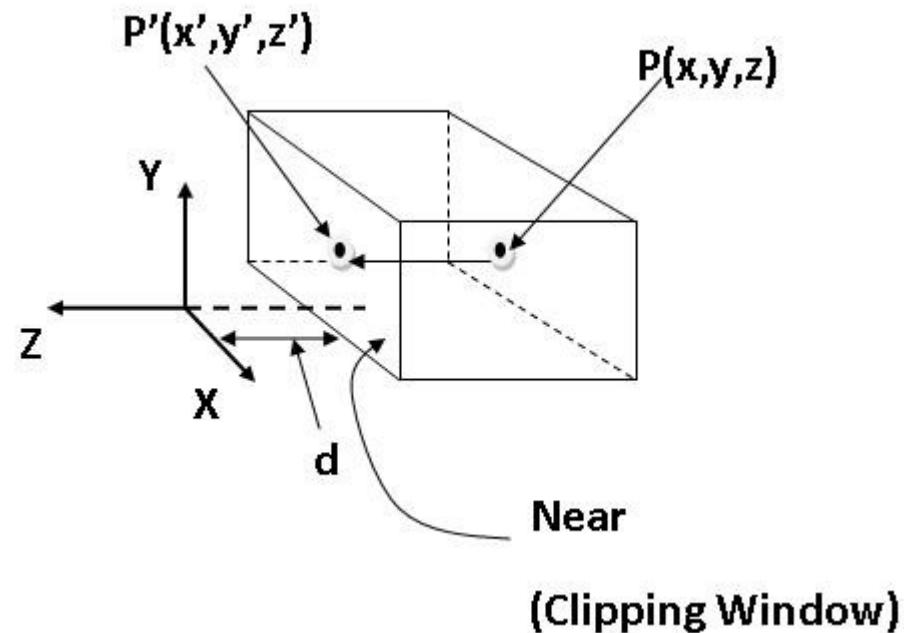
Parallel Projection

- After transforming the object to the VC, parallel projection is relatively easy – just drop the Z

$$x' = x$$

$$y' = y$$

$$z' = -d$$



Parallel Projection

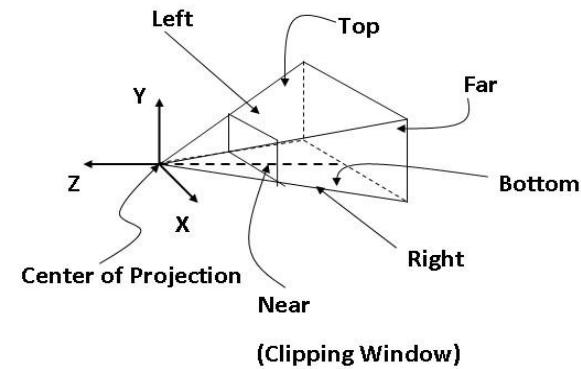
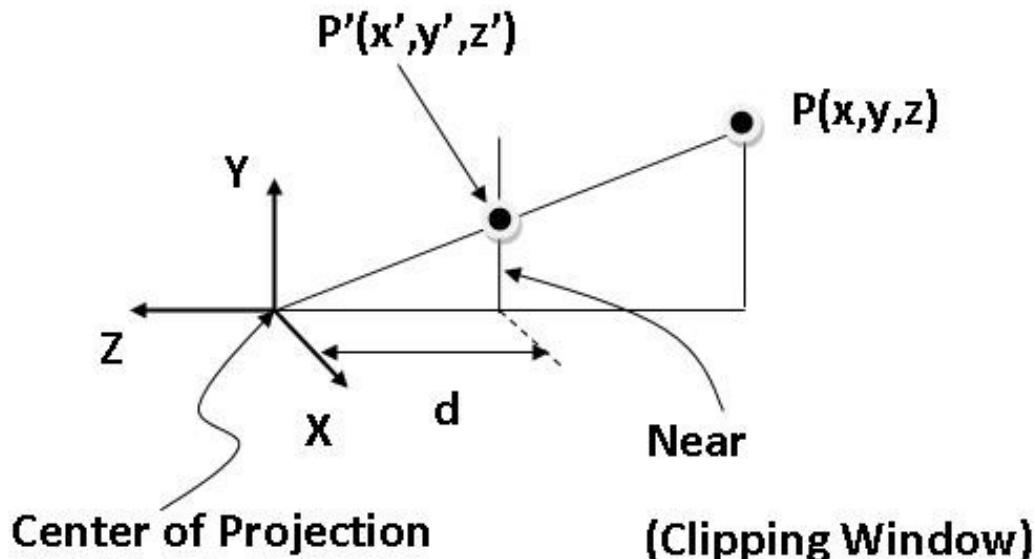
- Put in a matrix form (assuming the view plane at a distance d from origin, along the $-z$ direction)

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

- Note that this is in homogeneous coordinate

Perspective Projection

- Side view



Based on similar triangle:

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

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

Perspective Projection

- Same for x, so we have

$$x' = x \times d/z$$

$$y' = y \times d/z$$

$$z' = -d$$

- Put in a matrix form (in homogeneous coordinate system)

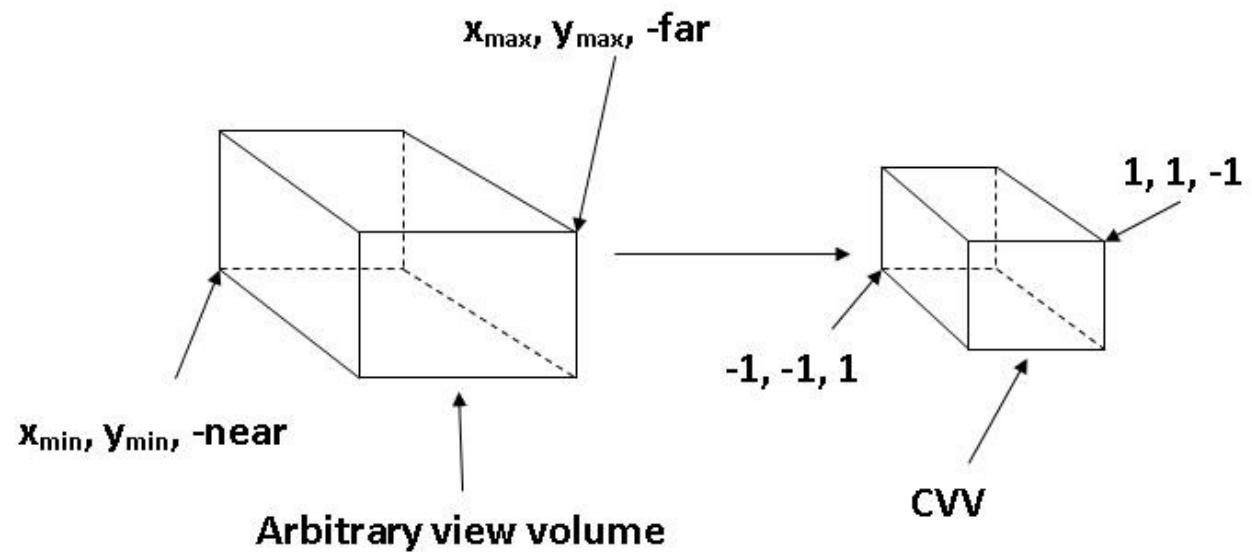
$$\begin{bmatrix} x'' \\ y'' \\ z'' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Canonical View Volume

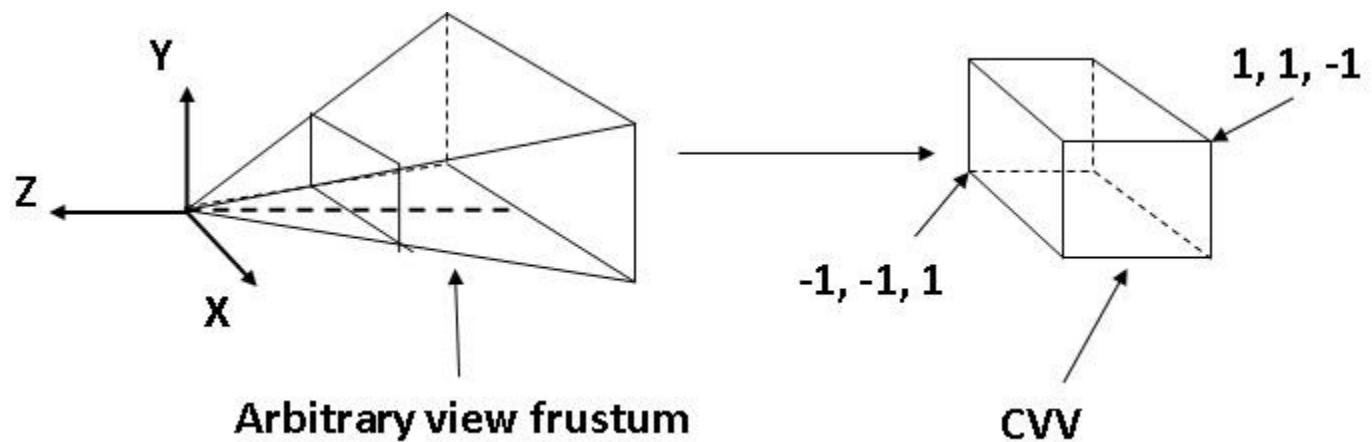
- Objects outside view volume are clipped
- Clipping can be done in two ways
 - Direct clipping: clip against whatever view volume is given
 - May involve calculation of intersection points of view volume boundary planes and lines. For arbitrary bounding planes, such computations may take significant time
 - Much easier (and sometimes faster) to clip against *canonical* view volumes

CVV

Parallel CVV



Perspective CVV



CVV

- For perspective projection, the canonical view is still a frustum - perspective canonical view volume (PeC)
 - With unit slope planes instead of arbitrary slopes
- Usually PeC is transformed to parallel canonical view volume (PrC)
- PeC → PrC can be achieved using a combination of transformations on PeC
 - Shear
 - Followed by scaling (scaling factor a function of distance)

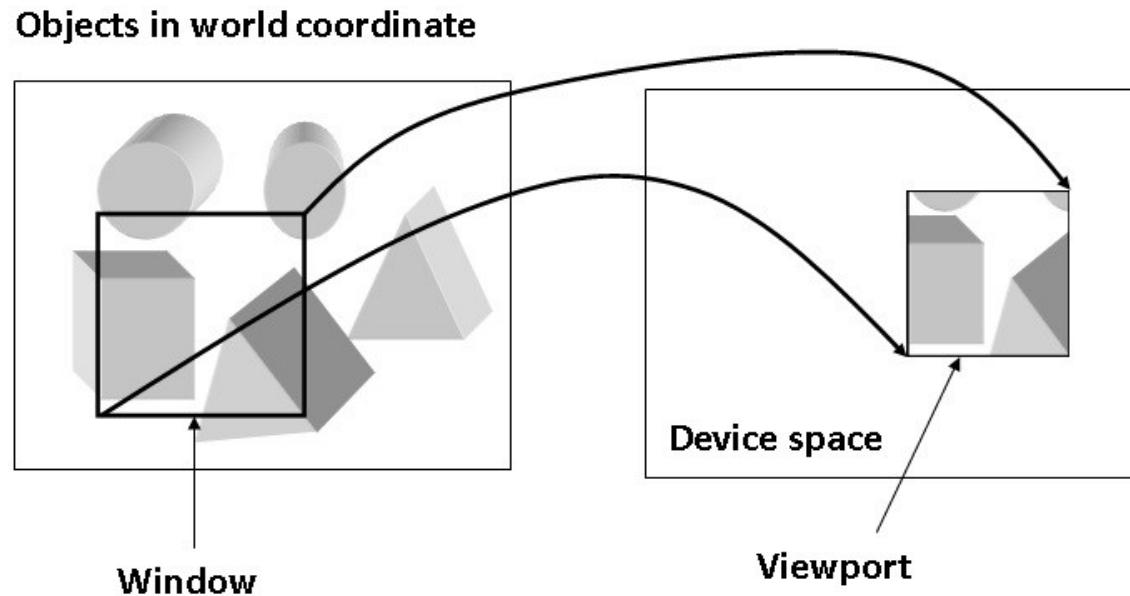
Note

- The projection matrices shown before are not final
- They undergo some changes
 - In order to preserve depth (z) information – required for hidden surface removal
 - Due to normalization (CVV) transformations
- The near plane of the CVV (symmetric cube) considered as the view/projection plane
 - Normalized clipping window

Viewport Transformation

Viewport Transformation

- Transform from projection coordinates (normalized clipping window) to device coordinates

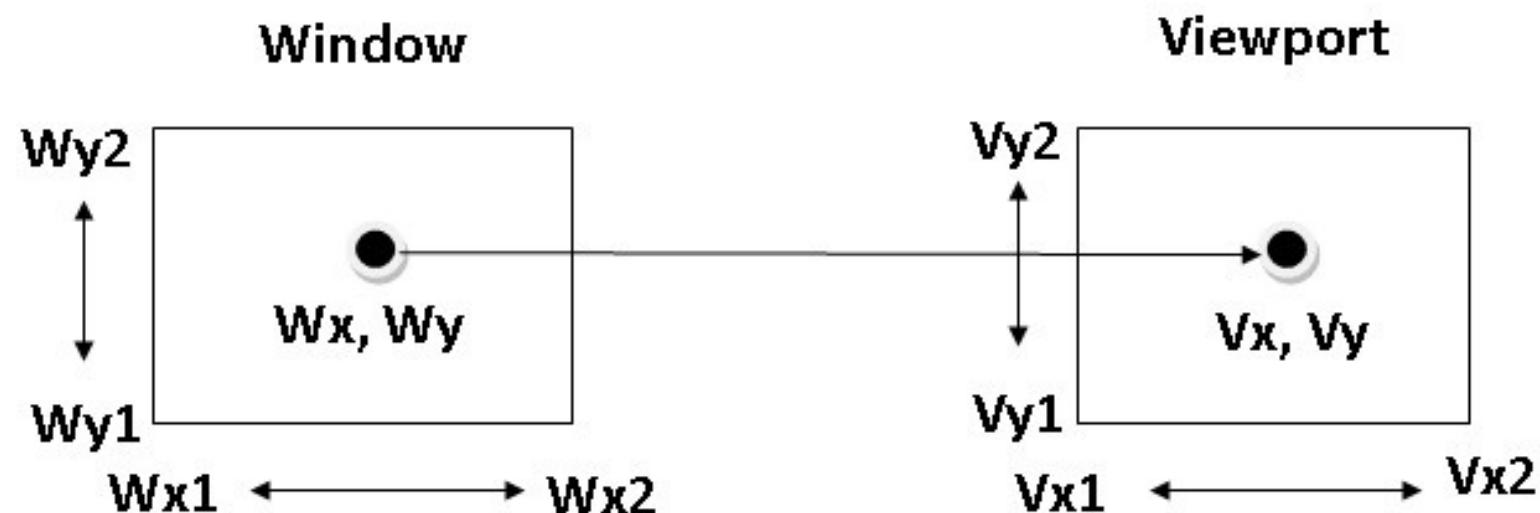


Window vs Viewport

- Window
 - World-coordinate area selected for display
 - What is to be viewed
- Viewport
 - Area on the display device to which a window is mapped
 - Where it is to be displayed

Viewport Transformation

- Window-to-Viewport Mapping



Viewport Transformations

Viewport Transformations

Note

- The normalized clipping window is *not* really 2D
 - It preserves depth information
- Hence viewport transformation is *not* between 2D window to 2D viewport
 - Actually, between window to viewport in 3D (since we have depth info)
 - Viewport is the mapping surface of 3D device space

Thank You

Lighting and Shading

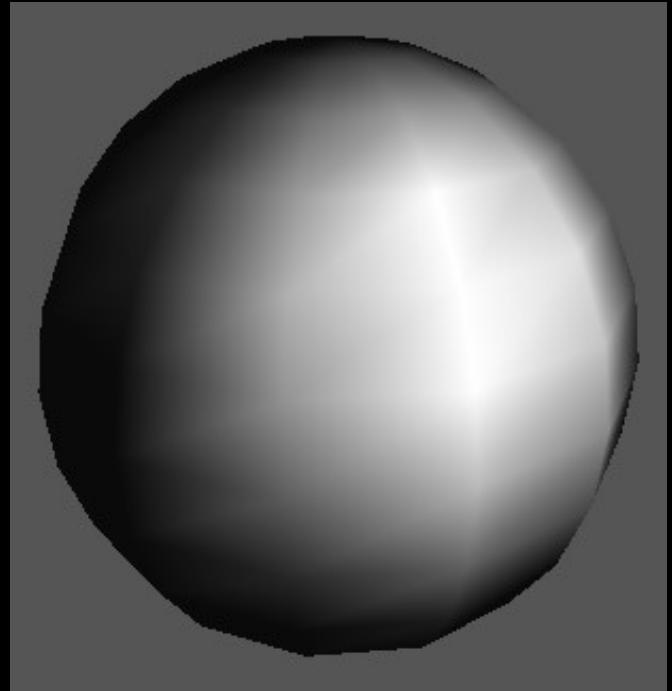
Lenin Laitonjam

Department of Computer Science and Engineering

NIT Mizoram

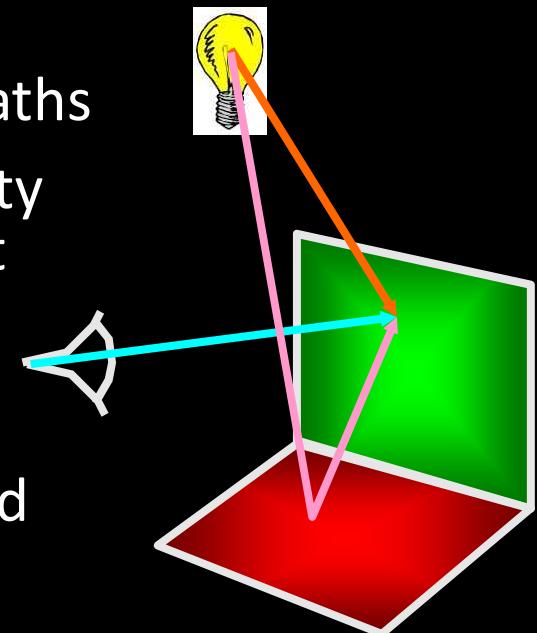
“Lighting”

- Two components:
 - **Lighting Model or Shading Model** - how we calculate the intensity at a point on the surface
 - **Surface Rendering Method** - How we calculate the intensity at each pixel



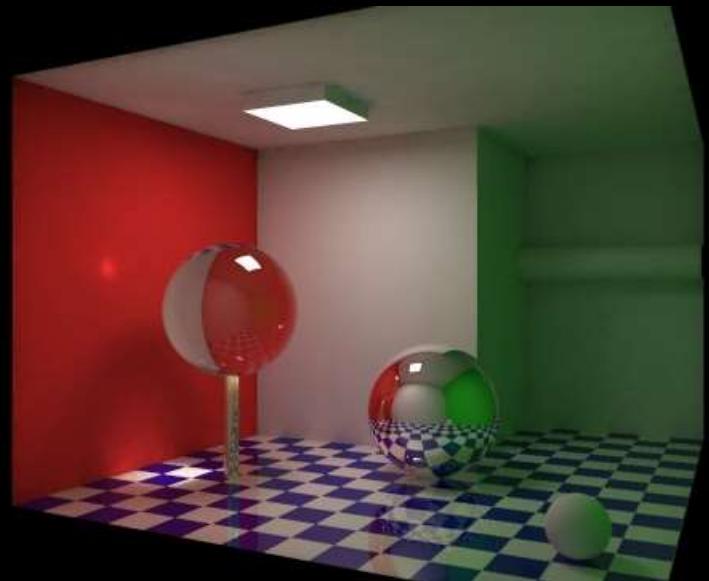
Background

- Illumination - the transport of light from a source to a point via direct and indirect paths
- Lighting - computing the luminous intensity for a specified 3D point, given a viewpoint
- Shading - assigning colors to pixels
- Illumination Models:
 - Empirical - approximations to observed light properties
 - Physically based - applying physics properties of light and its interactions with matter



The lighting problem...

- What are we trying to solve?
- **Global illumination** – the transport of light within a scene.
- What factors play a part in how an object is “lit”?



Two components

- Light Source Properties
 - Color (Wavelength(s) of light)
 - Shape
 - Direction
- Object Properties
 - Material
 - Geometry
 - Absorption



Light Source

- Point light source
- Directional light source/spotlight
- Ambient light

Light Position

- We can specify the position of a light one of two ways, with an x , y , and z coordinate.
 - What are some examples?
 - These lights are called ***positional lights***
- Q: Are there types of lights that we can simplify?

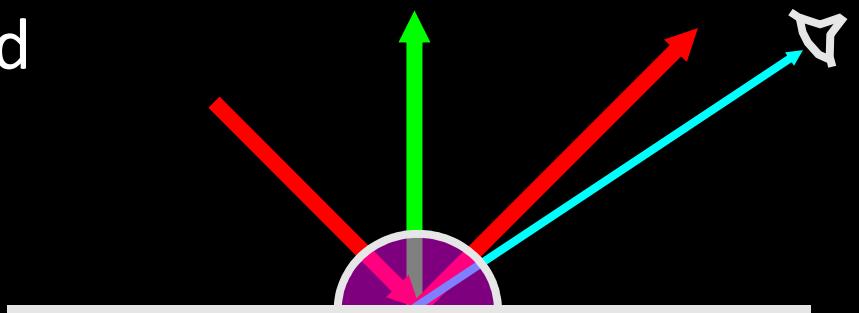
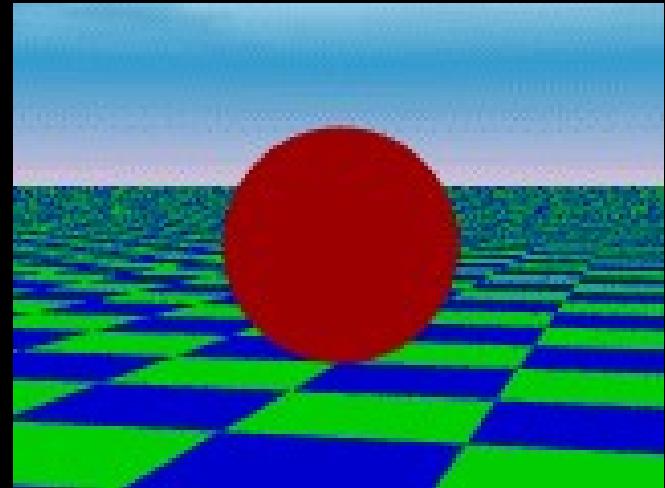
A: Yep! Think about the sun. If a light is significantly far away, we can represent the light with *only* a direction vector. These are called ***directional lights***. How does this help?

Contributions from lights

- We will breakdown what a light does to an object into three different components. This APPROXIMATES what a light does. To actually compute the rays is too expensive to do in real-time.
 - Light at a pixel from a light = Ambient + Diffuse + Specular contributions.
 - $I_{\text{light}} = I_{\text{ambient}} + I_{\text{diffuse}} + I_{\text{specular}}$
 - The reflected light intensity is a fraction of the incident light intensity. This fraction is determined by a surface property known as reflected coefficient/reflectivity (k 's)

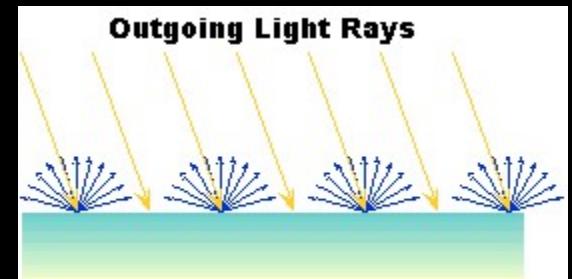
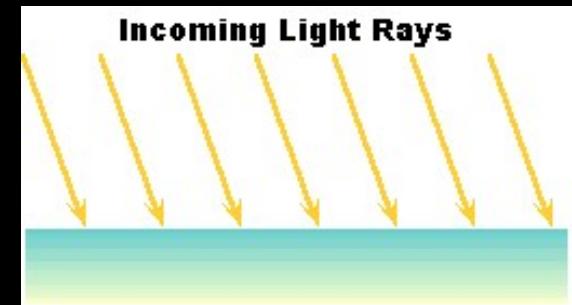
Ambient Term - Background Light

- The ambient term is a HACK!
- It represents the approximate contribution of the light to the general scene, regardless of location of light and object
- Indirect reflections that are too complex to completely and accurately compute
- $I_{\text{ambient}} = k_{\text{ambient}} I_{\text{ambient}}$



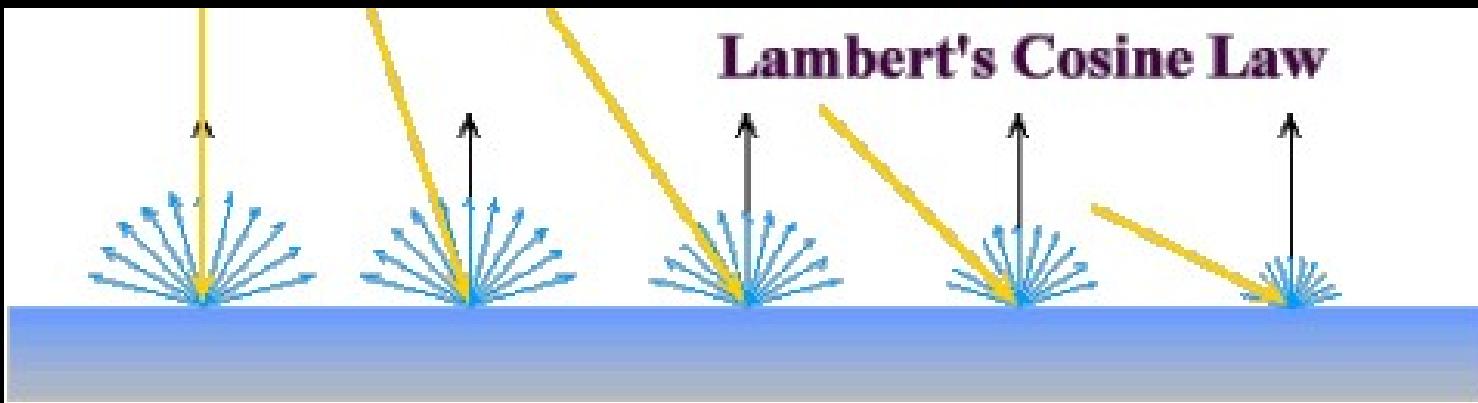
Diffuse Term

- Contribution that a light has on the surface, *regardless of viewing direction.*
- Diffuse surfaces, on a microscopic level, are very rough. This means that a ray of light coming in has an equal chance of being reflected in *any* direction.



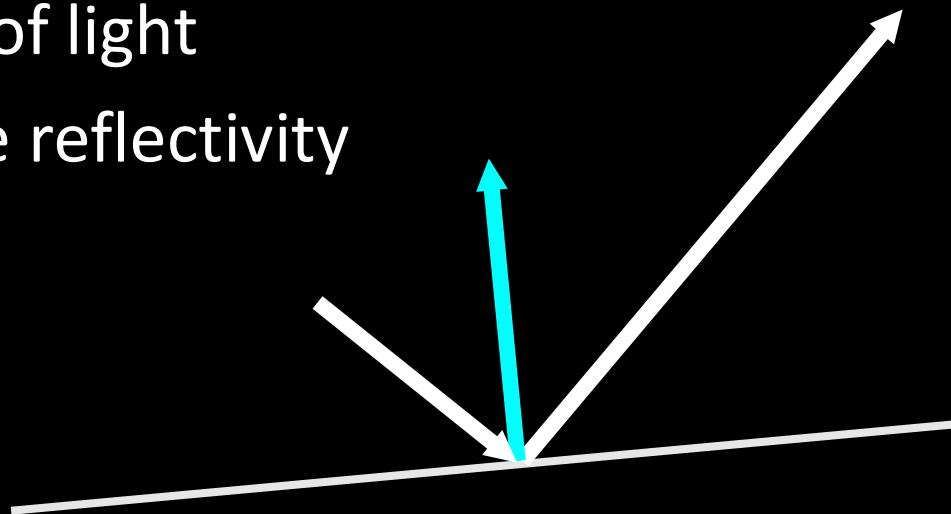
Lambert's Cosine Law

- Diffuse surfaces follow Lambert's Cosine Law
- Lambert's Cosine Law - reflected energy from a small surface area in a particular direction is proportional to the cosine of the angle between that direction and the surface normal.
- Think about surface area and # of rays



Diffuse Term

- To determine how much of a diffuse contribution a light supplies to the surface, we need the surface normal and the direction on the incoming ray
- What is the angle between these two vectors?
- $I_{\text{diffuse}} = k_d I_{\text{light}} \cos\theta = k_d I_{\text{light}} (N \cdot L)$
- I_{light} = diffuse (intensity) of light
- $k_d [0..1]$ = surface diffuse reflectivity
- What CS are L and N in?
- How expensive is it?



Specular Reflection

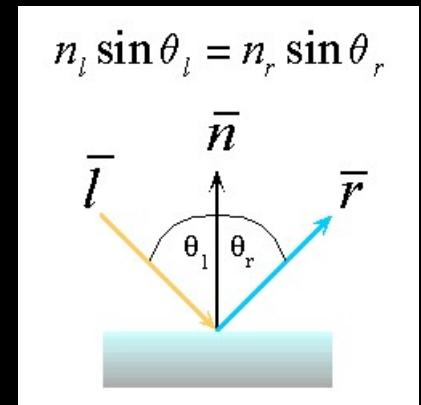
- Specular contribution can be thought of as the “shiny highlight” of a plastic object.
- On a microscopic level, the surface is very smooth. Almost all light is reflected.
- What is an ideal purely specular reflector?
- What does this term depend on?

Viewing Direction

Normal of the Surface

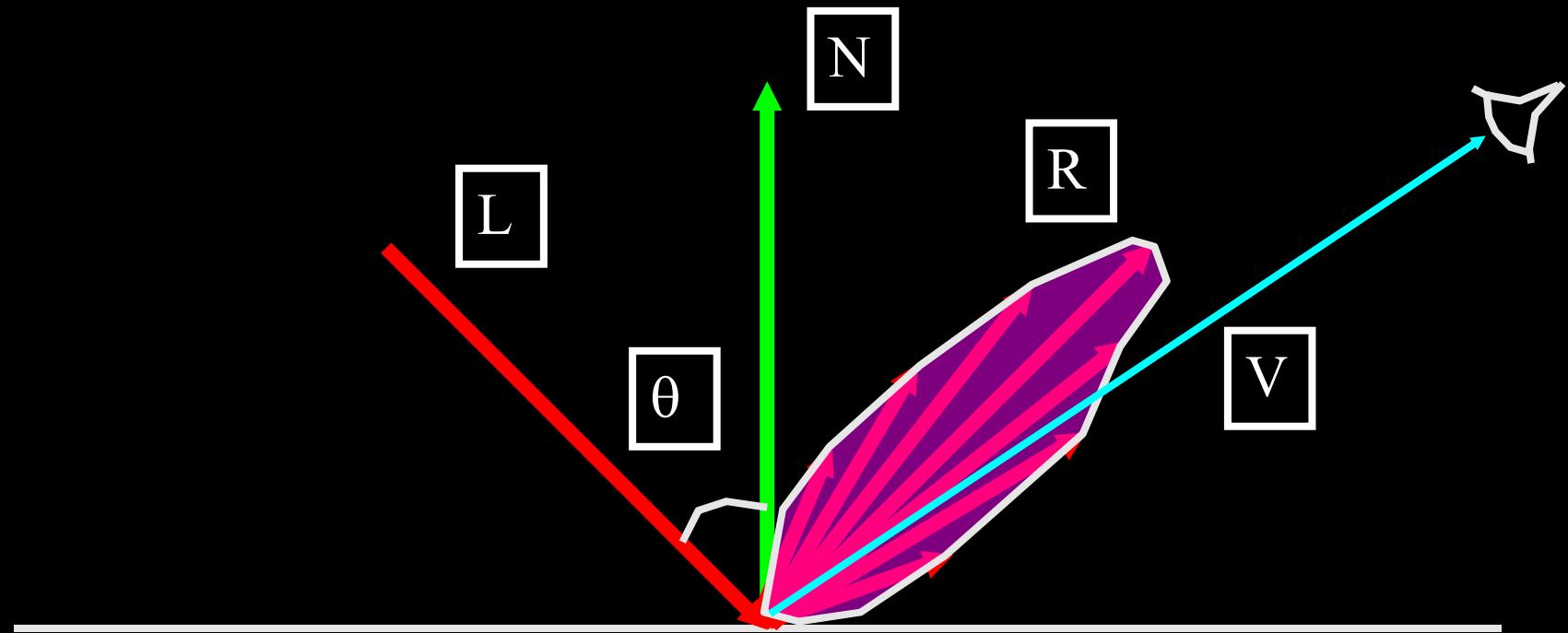
Snell's Law

- Specular reflection applies Snell's Law.
 - The incoming ray, the surface normal, and the reflected ray all lie in a common plane.
 - The angle that the reflected ray forms with the surface normal is determined by the angle that the incoming ray forms with the surface normal, and the relative speeds of light of the mediums in which the incident and reflected rays propagate according to:
 - We assume $\theta_i = \theta_r$

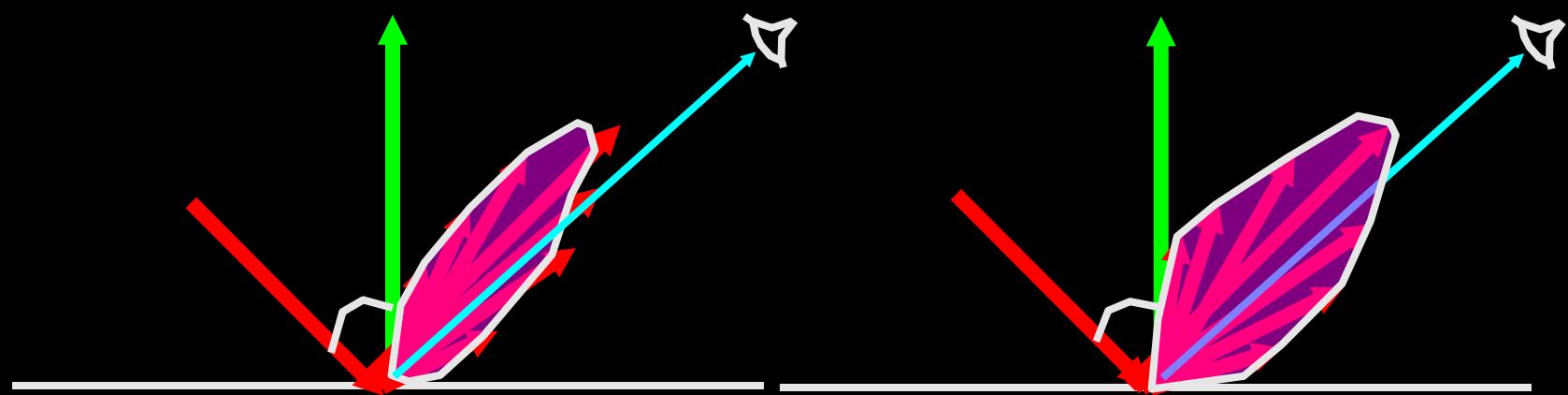
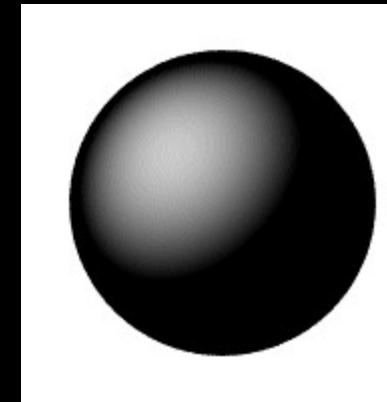
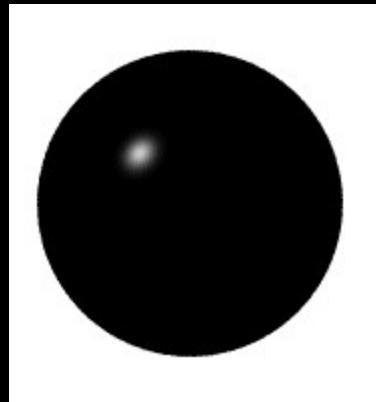


Snell's Law is for IDEAL surfaces

- Think about the amount of light reflected at different angles.

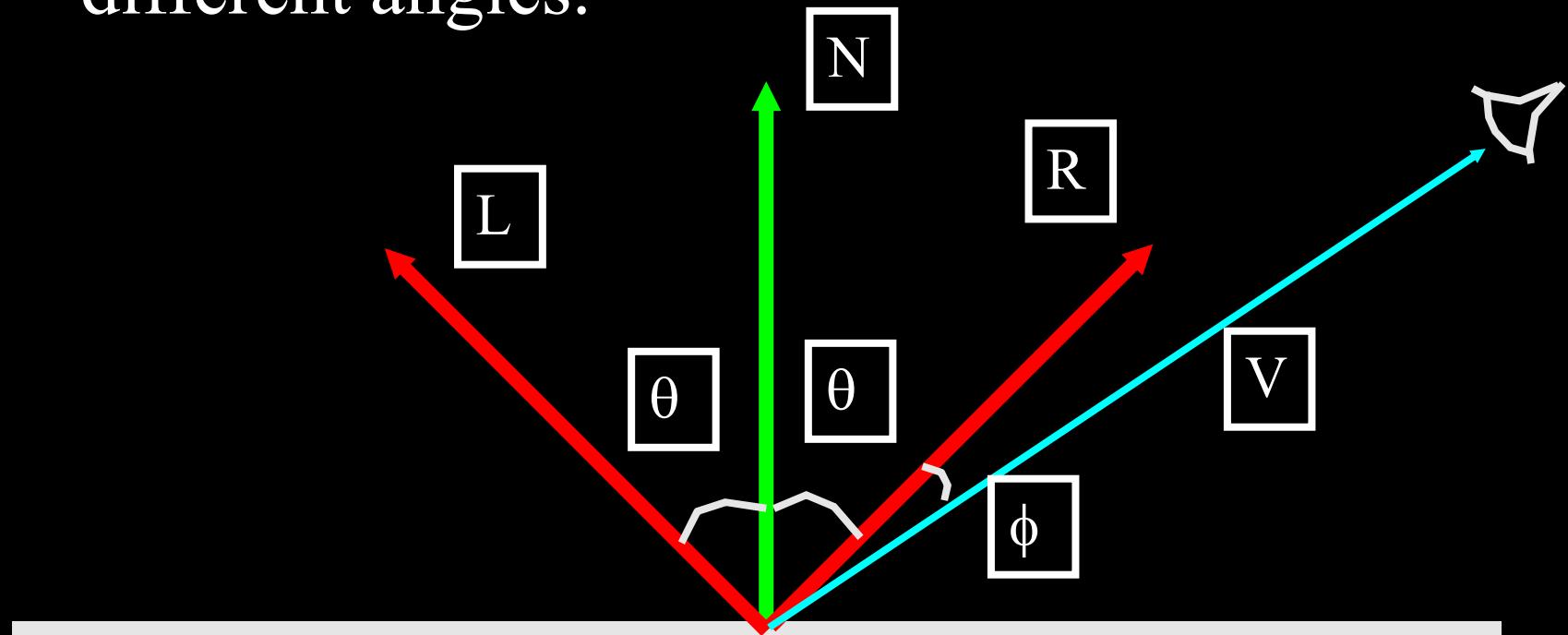


Different for shiny vs. dull objects



Snell's Law is for IDEAL surfaces

- Think about the amount of light reflected at different angles.



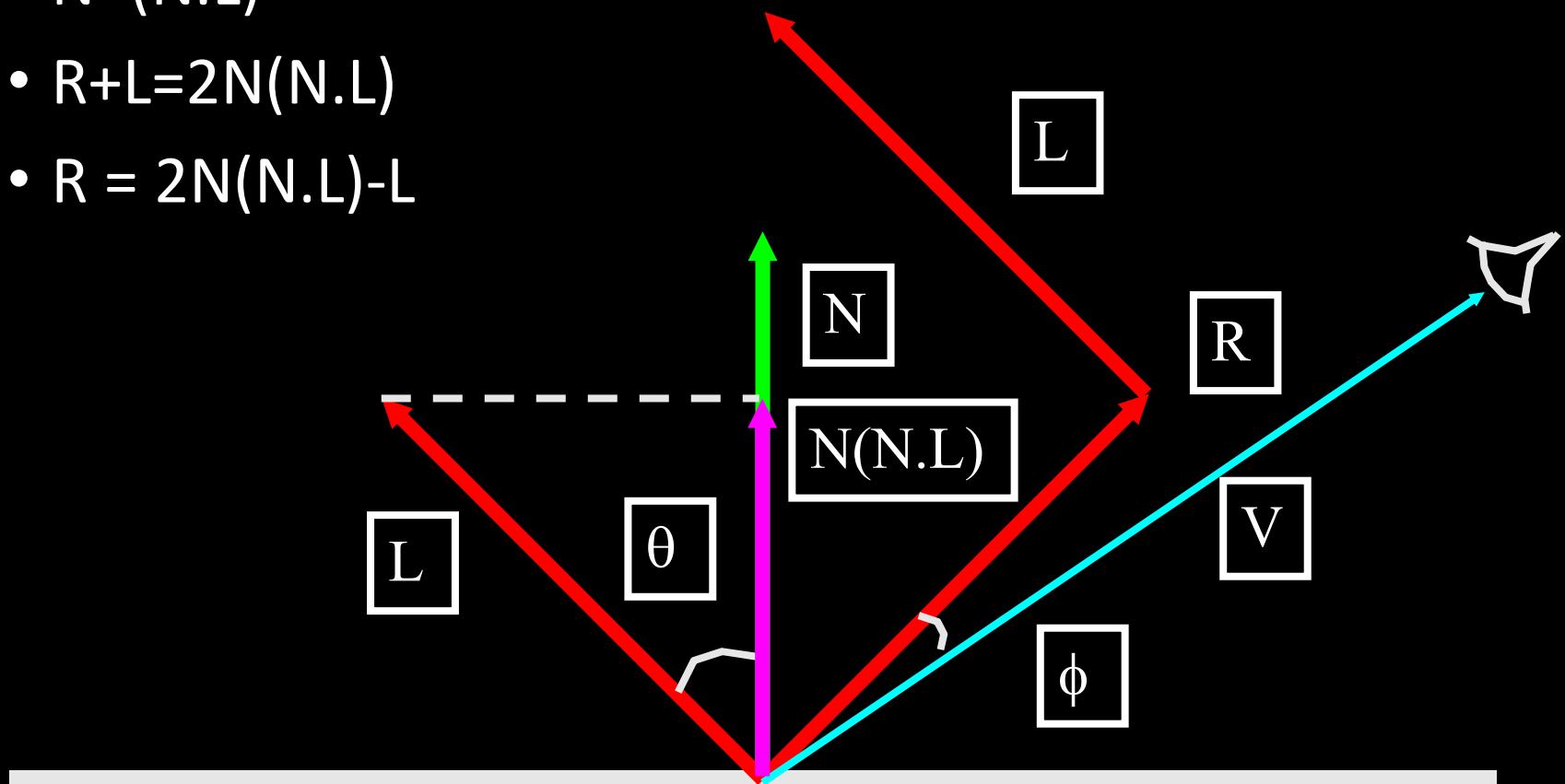
Phong Model

Phong Reflection Model

- An approximation is sets the intensity of specular reflection proportional to $(\cos \phi)^{\text{shininess}}$
- What are the possible values of $\cos \phi$?
- What does the value of *shininess* mean?
- How do we represent shiny or dull surfaces using the Phong model?
- What is the real thing we probably SHOULD do?
- $I_{\text{specular}} = k_s I_{\text{light}} (\cos \phi)^{\text{shininess}} = k_s I_{\text{light}} (\text{V.R})^{\text{shininess}}$

How do we compute R?

- $N^*(N.L)$
- $R+L=2N(N.L)$
- $R = 2N(N.L)-L$

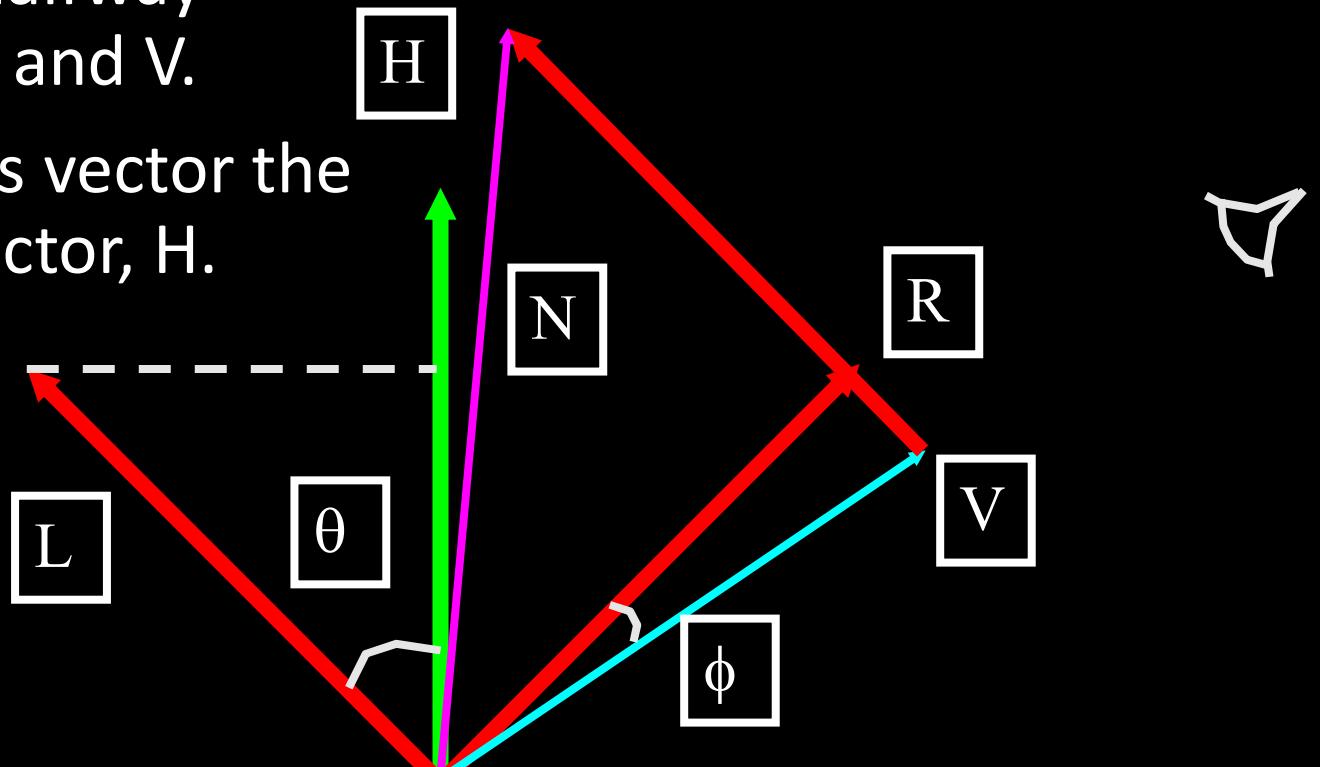


Simplify this

$$H = \frac{L + V}{|L + V|}$$

$$I_{specular} = k_s I_{light_specularity} (N \cdot H)^{shininess}$$

- Instead of R, we compute halfway between L and V.
- We call this vector the halfway vector, H.



Let's compare the two

$$R = 2N(N \cdot L) - L$$

$$I_{specular} = k_s I_{light_specularity} (V \cdot R)^{shininess}$$

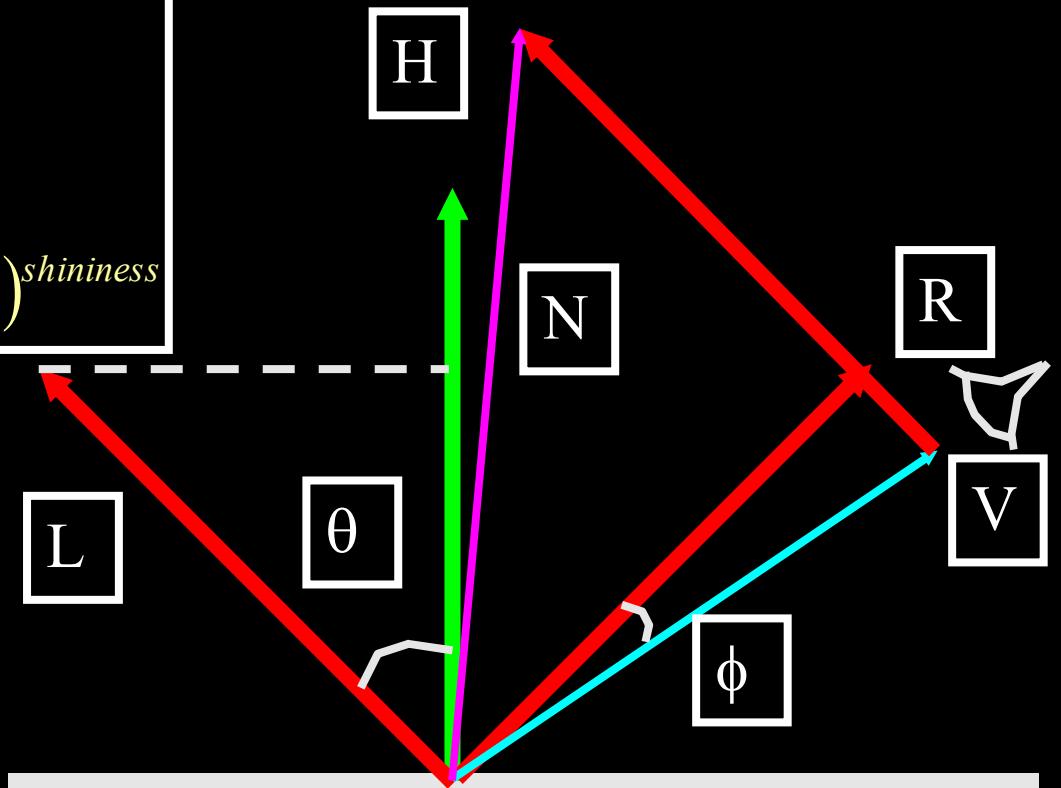
$$H = \frac{L + V}{|L + V|}$$

$$I_{specular} = k_s I_{light_specularity} (N \cdot H)^{shininess}$$

Q: Which vectors stay constant when viewpoint is far away?

A: V and L vectors \rightarrow H

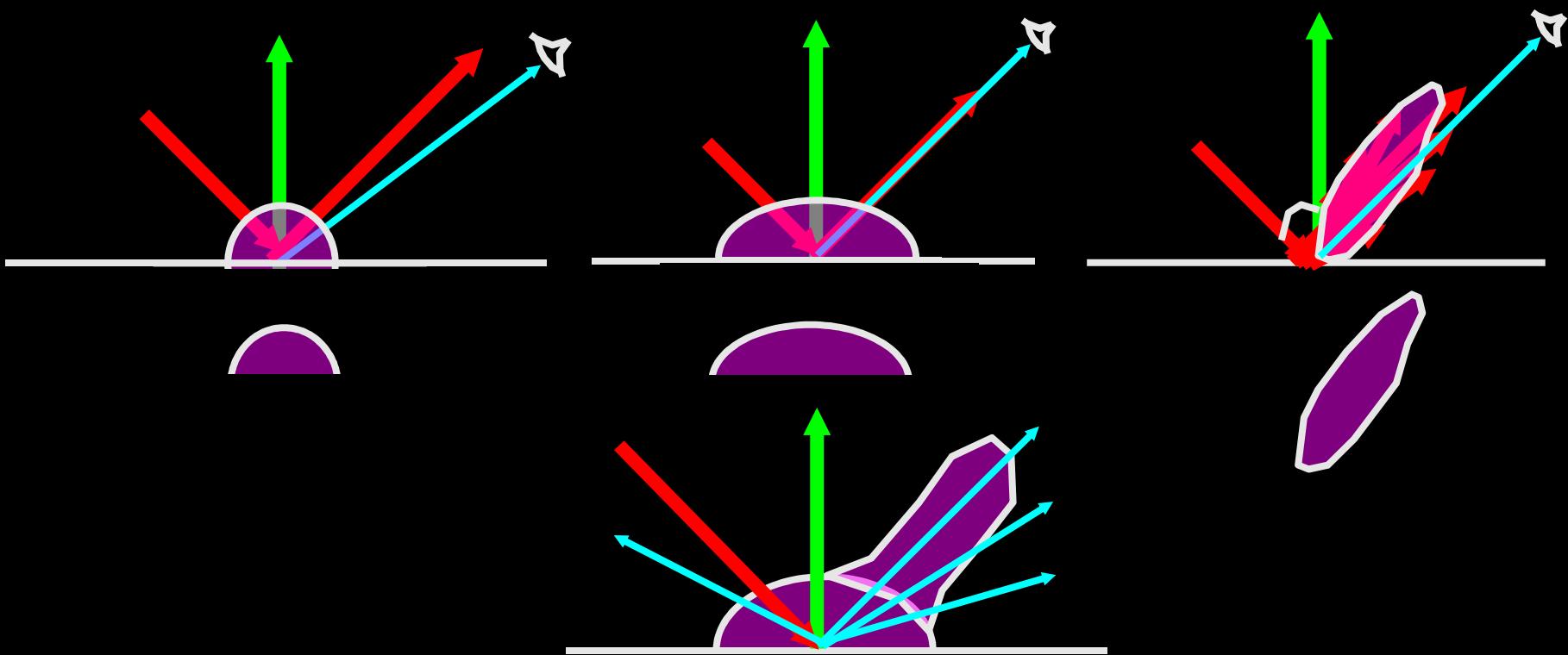
Q: What does this buy us?



Combining the terms

- Ambient - the combination of light reflections from various surfaces to produce a uniform illumination. Background light.
- Diffuse - uniform light scattering of light rays on a surface. Proportional to the “amount of light” that hits the surface. Depends on the surface normal and light vector.
- Specular - light that gets reflected. Depends on the light ray, the viewing angle, and the surface normal.

Ambient + Diffuse + Specular



Lighting Equation

$$I_{final} = I_{ambient}k_{ambient} + I_{diffuse}k_{diffuse}(N \cdot L) + I_{specular}k_{specular}(N \cdot H)^{shininess}$$

$$I_{final} = \sum_{l=0}^{lights-1} I_{l_{ambient}}k_{ambient} + I_{l_{diffuse}}k_{diffuse}(N \cdot L) + I_{l_{specular}}k_{specular}(N \cdot H)^{shininess}$$

$I_{l_{ambient}}$ = light source l 's ambient component

$I_{l_{diffuse}}$ = light source l 's diffuse component

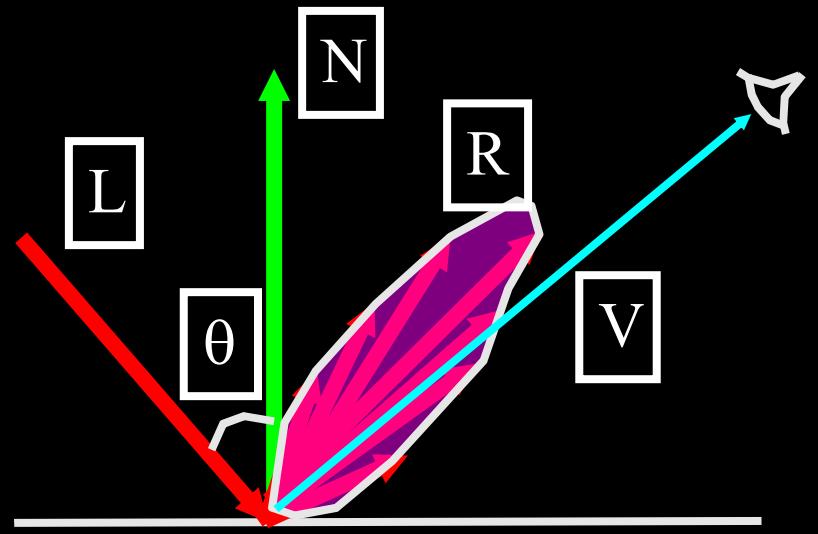
$I_{l_{specular}}$ = light source l 's specular component

$k_{ambient}$ = surface material ambient reflectivity

$k_{diffuse}$ = surface material diffuse reflectivity

$k_{specular}$ = surface material specular reflectivity

$shininess$ = specular reflection parameter (1 -> dull, 100+ -> very shiny)



Attenuation

- One factor we have yet to take into account is that a light source contributes a higher incident intensity to closer surfaces.
- The energy from a point light source falls off proportional to $1/d^2$.
- What happens if we *don't* do this?

What would attenuation do for:

- Actually, using *only* $1/d^2$, makes it difficult to correctly light things.
Think if $d=1$ and $d=2$. Why?
- Remember, we are approximating things. Lighting model is too simple AND most lights are not point sources.
- We use:

$$f(d) = \frac{1}{a_0 + a_1 d + a_2 d^2}$$

Subtleties

- What's wrong with:

$$f(d) = \frac{1}{a_0 + a_1 d + a_2 d^2}$$

What's a good fix?

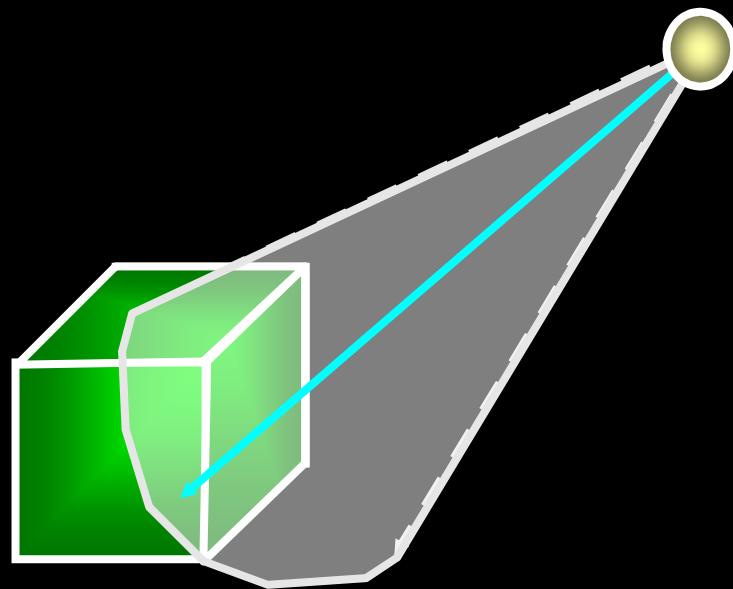
$$f(d) = \min\left(1, \frac{1}{a_0 + a_1 d + a_2 d^2}\right)$$

Radial Attenuation Factor

$$I_{final} = I_{l_{ambient}} k_{ambient} + \sum_{l=0}^{lights-1} f(d_l) \left[I_{l_{diffuse}} k_{diffuse} (N \cdot L) + I_{l_{specular}} k_{specular} (N \cdot H)^{shininess} \right]$$

$$f(d) = \min\left(1, \frac{1}{a_0 + a_1 d + a_2 d^2}\right)$$

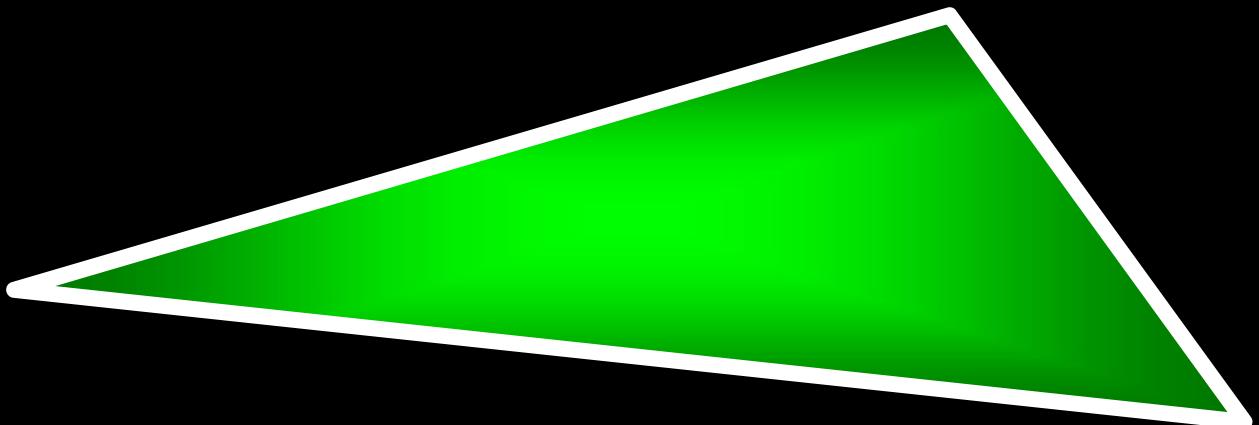
Angular Attenuation Factor



Shading

$$I_{final} = I_{l_{ambient}} k_{ambient} + \sum_{l=0}^{lights-1} f(d_l) [I_{l_{diffuse}} k_{diffuse} (N \cdot L) + I_{l_{specular}} k_{specular} (N \cdot H)^{shininess}]$$

- When do we do the lighting equation?
- What is the cost to compute the lighting for a 3D point?

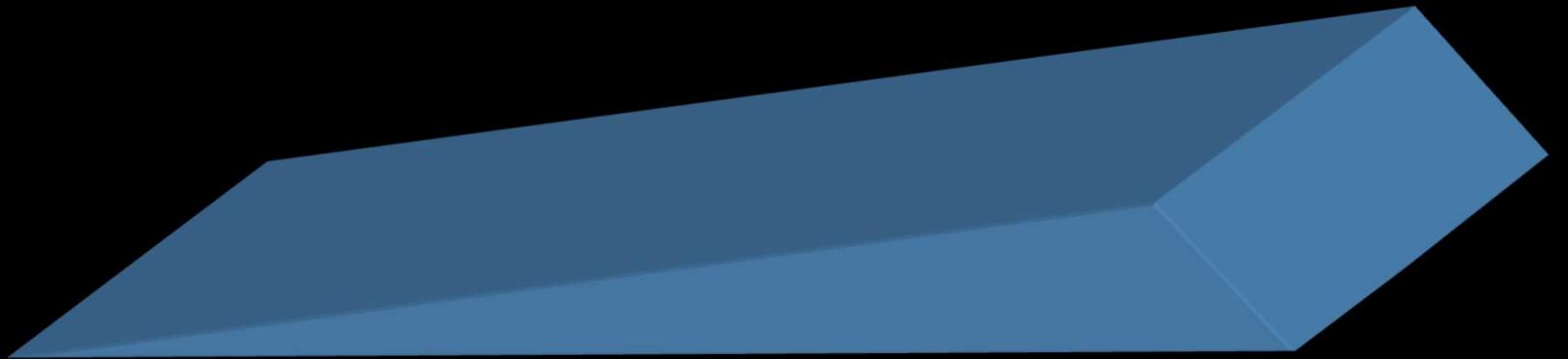


Shading

- Shading is how we “color” a triangle.
- Constant Shading
- Gouraud Shading
- Phong Shading

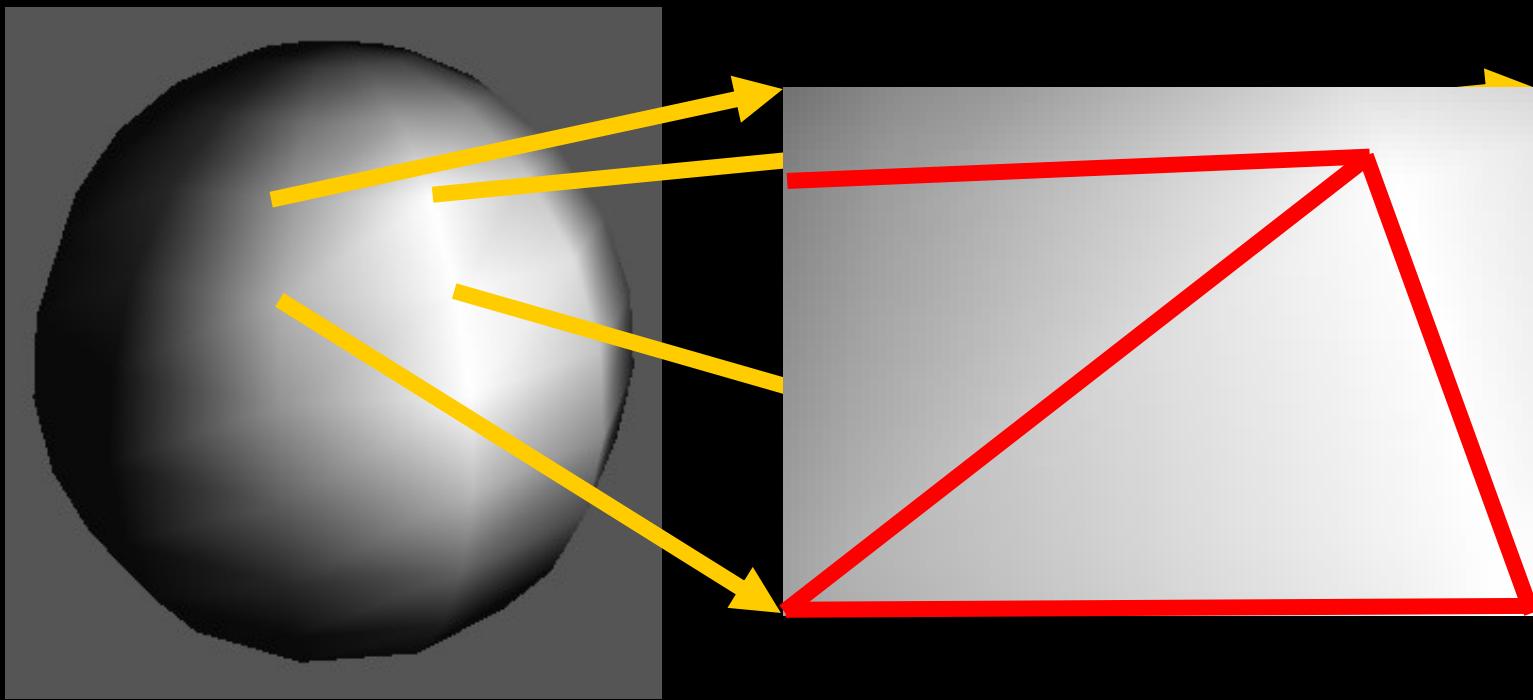
Constant Shading

- Constant Intensity or Flat Shading
- One color for the entire triangle
- Fast
- Good for some objects
- What happens if triangles are small?
- Sudden intensity changes at borders



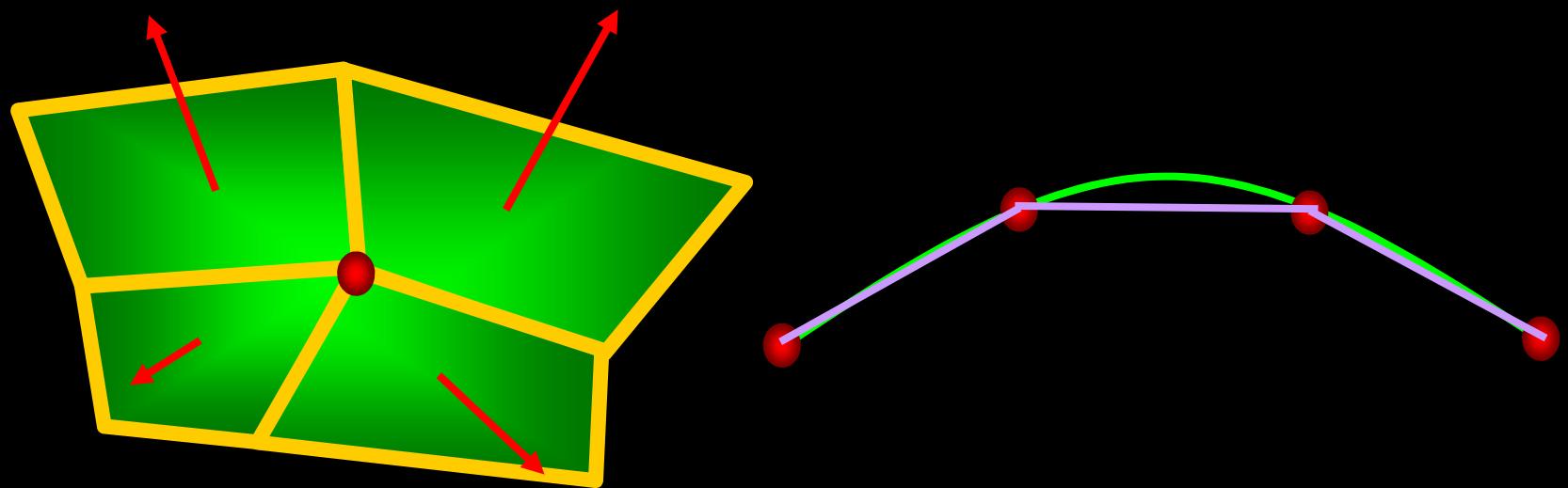
Gouraud Shading

- Intensity Interpolation Shading
- Calculate lighting at the vertices. Then interpolate the colors as you scan convert



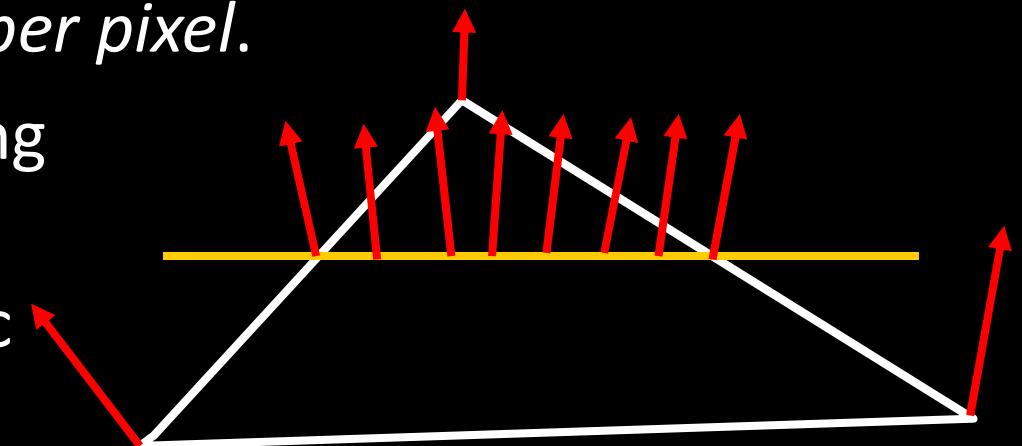
Gouraud Shading

- Relatively fast, only do three calculations
- No sudden intensity changes
- What can it not do?
- What are some approaches to fix this?
- Question, what is the normal at a vertex?



Phong Shading

- Interpolate the normal, since that is the information that represents the “curvature”
- Linearly interpolate the vertex normals.
For ***each*** pixel, as you scan convert,
calculate the lighting per pixel.
- True “per pixel” lighting
- Not done by most hardware/libraries/etc



Shading Techniques

- Constant Shading
 - Calculate one lighting calculation (pick a vertex) per triangle
 - Color the *entire* triangle the same color
- Gouraud Shading
 - Calculate three lighting calculations (the vertices) per triangle
 - Linearly interpolate the colors as you scan convert
- Phong Shading
 - While you scan convert, linearly interpolate the normals.
 - With the interpolated normal at each pixel, calculate the lighting at each pixel

Thank You

Computer Graphics

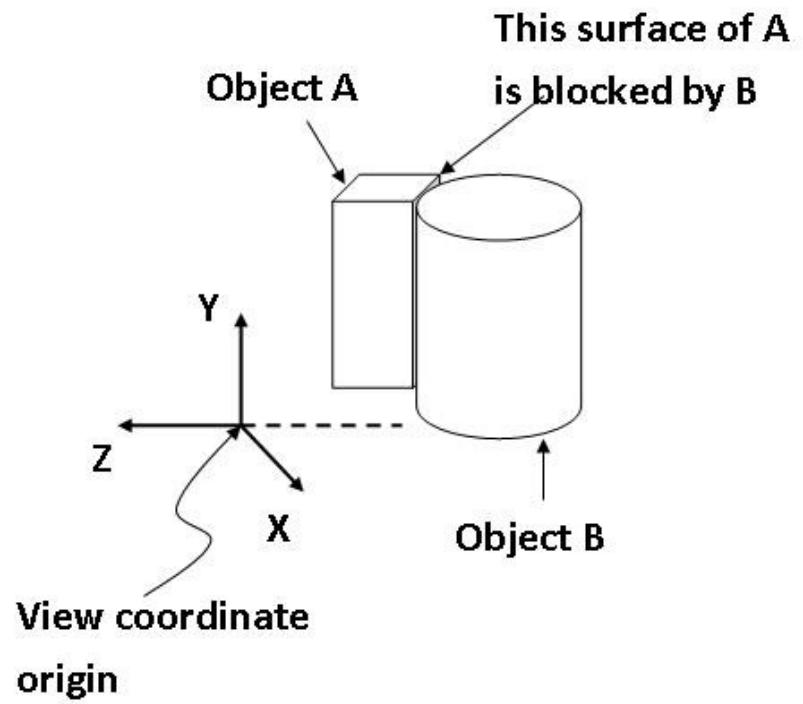
Hidden Surface Removal

Lenin Laitonjam

NIT Mizoram

Why?

- We must determine what is visible within a scene from a chosen viewing position
- For 3D worlds this is known as **visible surface detection** or **hidden surface removal**



Two Main Approaches

- **Object Space Methods:** Compares objects and parts of objects to each other within the scene definition to determine which surfaces are visible
- **Image Space Methods:** Visibility is decided point-by-point at each pixel position on the projection plane
- Image space methods are more common

Coherence Properties

- Helps to reduce computations
- types of coherence
 - object coherence
 - face coherence
 - edge coherence
 - scan line coherence
 - area and span coherence
 - depth coherence
 - frames coherence

Back-Face Elimination

- The simplest thing
 - Find the faces on the backs of polyhedral and discard them
- We know a point (x, y, z) is behind a polygon surface if:

$$Ax + By + Cz + D < 0$$

where A, B, C & D are the plane parameters for the surface

Thank You