Computer Graphics(L01)

EG678EX

Introduction And History of Computer Graphics

Prepared By: KKJ

Course Overview

- Introduction, History
- Application in different fields like CAD, schematic capture, medicine, art etc.
- Hardware concepts
- 2-D and 3-D algorithms
  - Line drawing, viewing transformations, other transforms (scaling, rotation, translation)
- Graphical Languages
  - Brief intro of different languages, Language in project
- Project
  - Project guideline, Project group, Topic Proposal, Project Presentation
- Text Book
  - Computer Graphics C Version – Hearn & Baker

Computer Graphics

- A sub-field of computer science which studies method for digitally synthesizing and manipulating visual content.
- Are texts and sounds graphics ???
  - No (*wikipedia*); but text→??
- Generally refers to (*Ref: Wikipedia*):
  - Representation and manipulation of pictorial data by computer
  - Various technologies used to create and manipulate such pictorial data
  - The image so produced

History of Computer Graphics

- Late 1950's – Whirlwind computer (MIT) and SAGE (Semi Automatic Ground Environment); an automated control system to bombard enemy
  - Used CRT and light pen for user interactive environment
- 1959 – TX-2 (MIT's Lincoln Laboratory); first interactive interface computer system using light pen and bank switches
- 1960 – William Fetter
  - Introduced phrase "Computer Graphics"
  - Boeing man (Fetter called it First Man) for human figure simulation to describe different user environment
- Mid 1960's – MIT activities in computer graphics field promoted early computer graphics industries like TRW, General Electric, IBM
- End 1960's – organization, conferences, graphics standards and publications
  - 1969 – ACM initiated Special Interest Group In Graphics (SIGGRAPH)

History of Computer Graphics

- 1970’s – powerful PCs to draw basic and complex shapes
- 1980’s – artists and graphics designers preferred to use Macintosh and PCs
- Late 1980’s – 3-D computer graphics with SGI (Silicon Graphics) computers
- 1990’s onwards – 3D graphics in gaming, multi media and animation, GUI

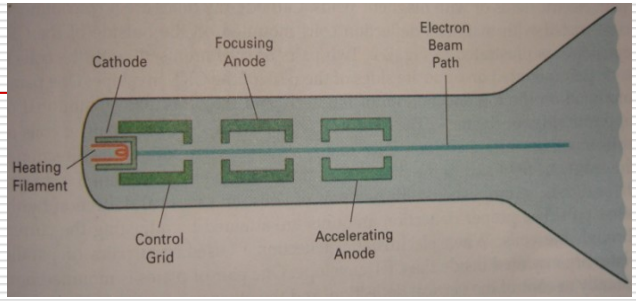
Computer Graphics
EG678EX

Hardware Concepts

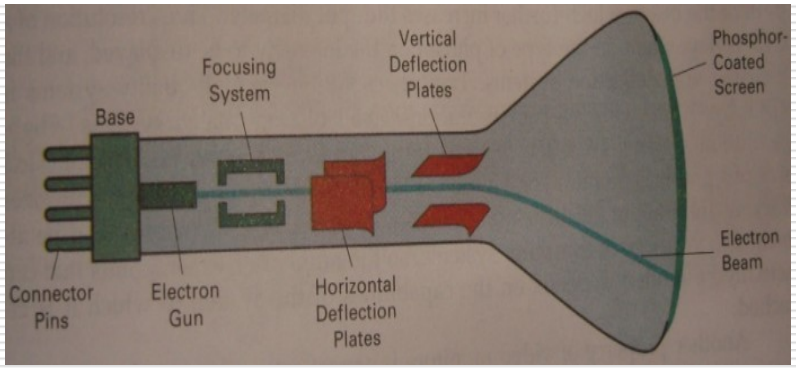
[Note: This presentation is a reference. For detailed study, students must read the text book]

Prepared By: Dipesh Gautam

Video Display Devices



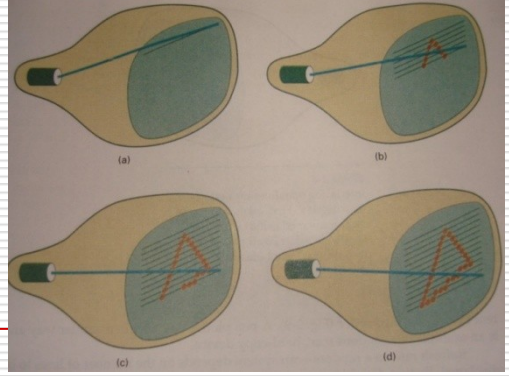
- Cathode Ray Tube (CRT)
  - Components:
    - Electron Gun → composed of heated metal cathode and control grid
    - Accelerating Anode
    - Focusing System
    - Deflection system
    - Phosphor Screen



CRT Operation

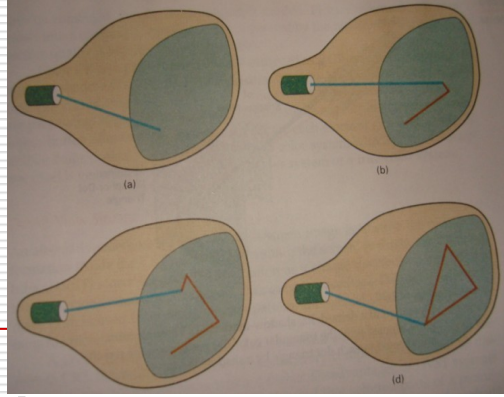
- Heat generated in cathode boils off the electrons
- Electrons are accelerated towards phosphor screen with high positive voltage applied at accelerating anode
- The negative voltage applied at cylindrical control grid controls the intensity of electron beam by repealing electrons
  - High negative voltage stops electron passing from the hole of control grid while small negative voltage decreases electron passage
- Focusing system concentrates electron beam to a small spot
  - In electrostatic focusing, electrons pass through positively charged metal cylinder
  - In magnetic focusing, coils are mounted outside of CRT Envelope which produces smallest spot
- Deflection System deflects electron beam horizontally and vertically
  - Magnetic→ two pairs of coils
  - Electrostatic → two pairs of deflection plates
- **Refresh rate** depends on **persistence** of phosphor

Raster Scan Displays



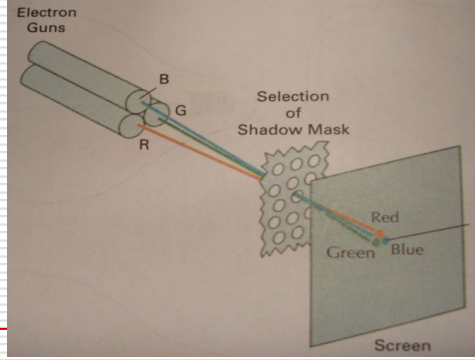
- The electron beam is swept across the screen, one row at a time from top to bottom
- The illuminated spot pattern is created by turning on or off when electron beam moves along each scan line
- Picture definition is stored in a memory area called **Refresh Buffer** *(Frame Buffer)*
  - Refresh Buffer holds the set of intensity values for all the screen points
  - Each screen points is known as **pixel** (short form of *picture element*)
- Example: Home television sets and printers
- For bilevel system (black and white) only 1 bit memory per pixel is sufficient
- For color system more bits per pixel are needed
  - For screen with resolution 1024 by 1024, and 24 bits per pixel (8 bits each for RGB) requires 3 MB of storage is needed
- Refreshing rate for raster scan display is usually 60 to 80 frames per second (i.e 1/80 or 1/60 seconds is taken for electron beam to scan from top left corner to bottom right corner)

Random Scan Display



- Electron beam is directed to the part of screen where picture is to be drawn
- Picture definition is stored as set of line-drawing commands in memory are known as refresh display file or simply refresh buffer
- Also known as Vector display

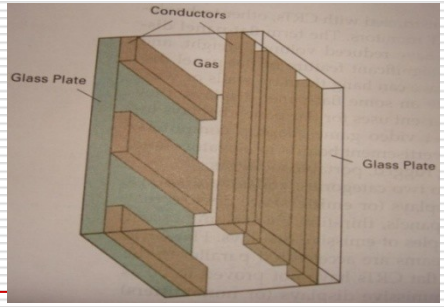
Color CRT Monitors



- Uses combination of phosphors that emit different colored light (usually Red, Green and Blue)
- Two basic methods for color picture display
  - Beam penetration method
    - Two layers of phosphor outer red layer and inner green layer
    - slow electron strikes outer to produce red and faster strikes inner layer to produce green color while intermediate produces orange and yellow
    - Only four colors are possible
  - Shadow mask method
    - Has three phosphor color dots (RGB) for each pixel position
    - Three electron guns one for each color dot
    - A shadow mask grid with holes aligned with the phosphor dot patterns
    - Electron beams passed from a hole of shadow mask activate the phosphor dot pattern to display color picture

Flat Panel Displays

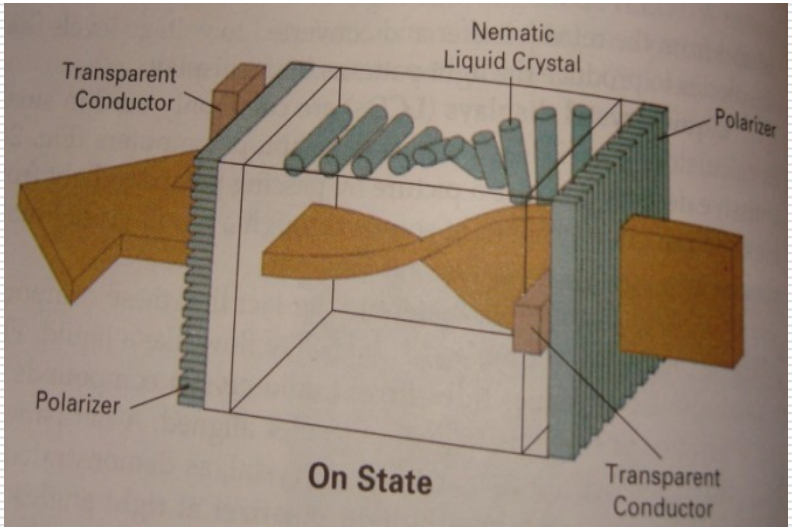
- Emissive → converts electrical energy to light
  - Eg: plasma panels, thin-film electroluminescent display, LEDs
- Non Emissive→ uses optical effects to convert sunlight or light from other source into graphics pattern
  - Eg: Liquid Crystal Display (LCD)

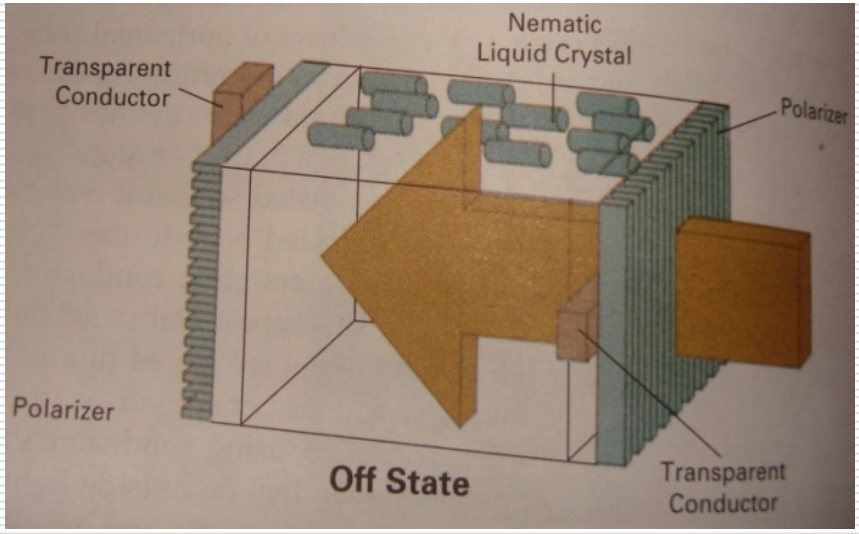


- Plasma Panel (gas discharge display)
  - The region between two glass plates are filled with a mixture of gases (*usually includes neon*)
  - Series of vertical conducting ribbon is placed on one plate and horizontal on the other
  - When firing voltage is applied between one horizontal and vertical conductor, the gas at intersection breaks up into glowing plasma ion and electrons
  - Refresh is needed
  - Monochromatic device; but now are capable of displaying color and grayscale
- Thin Film electroluminescent Display
  - Similar to plasma panel, but the space between glass plates is filled with phosphor, such as zinc sulfide doped with manganese
  - When electricity is passed through two electrodes, the phosphor become the conductor in that area and the electrical energy is absorbed by manganese atom to release the energy as a spot of light
  - Good color and grayscale are hard to achieve
- Light Emitting Diode (LED)
  - Diode matrix is arranged to form pixel position
  - Various voltage level produces various color lights

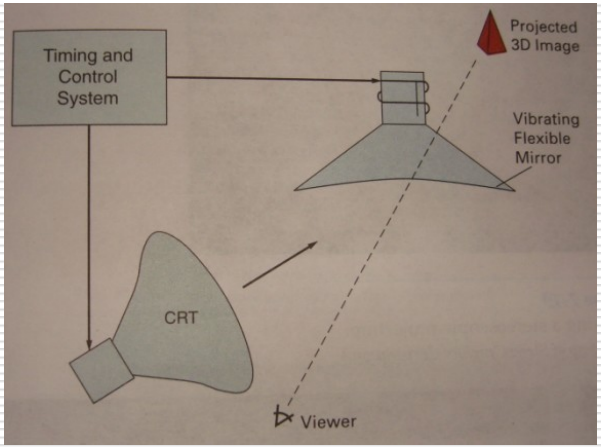
- Liquid Crystal Display (LCD)
  - Uses liquid having crystalline arrangement of molecules
  - The liquid crystal compounds keep the long axes of the rod shaped molecules aligned
  - Two glass plates each containing light polarizer at right angles to other plate are placed to sandwich the liquid crystal
  - Horizontal conductors in one and vertical in other plate are placed thus intersection forms a pixel
  - When no voltage is applied between conductors, the molecules are in **on state** so the polarized light is twisted by the molecule thus passes through the two plates and seen by the viewer
  - When voltage is applied, the molecules are in **off state,** and light passing through one plate is stopped at other thus no light is seen by viewer
  - **passive matrix**: voltage is applied to intersecting conductor to turn off the pixel
  - **Active matrix**: uses transistor to control voltage at pixel locations
  - Color display is possible by using different materials or dyes and by placing a triad of color pixels at each screen location

LCD


Nematic
Liquid Crystal
Transparent
Conductor
Polarizer
Polarizer
On State
Transparent
Conductor


Nematic
Liquid Crystal
Transparent
Conductor
Polarizer
Polarizer
Off State
Transparent
Conductor

- 3D viewing Devices
  - Displays 3-D scene
  - Displayed by reflects CRT image from a vibrating mirror
  - As mirror vibrates it changes focal length
  - The vibration is synchronized with the display of an object on CRT so that each point in object is reflected from the mirror into a spatial position corresponding to distance of that point from a specified viewing position
- Stereoscopic and Virtual Reality Systems
  - READ YOURSELF



Input Devices

- Keyboards
  - ASCII keys are used to input text string
  - Provides with features to facilitate entry of screen coordinates, menu selections or graphics functions
  - Function keys allow user to enter frequently used operations in a single stroke and cursor cont keys are used for cursor position or picture selection
  - Some keyboards consist of trackball or joystick
- Mouse
  - Pointing device to position cursor
  - Wheel or rollers are used to record the amount and direction of movement
  - Optical mouse uses optical sensors to detect mouse motion
  - One two or three buttons are included
  - [Note: For operation detail, please refer to internet sites such as wikipedia]
- Trackball and Spaceball
- Joysticks
- Data Globe
- Digitizers
- Image Scanners
- Touch Panels
- Light Pens

Hard Copy Devices

- Printers
  - Character Impact (Dot matrix) printers
    - Similar to raster scan system
    - Prints one character at a time and scans one line at a time while for next line paper is scrolled
  - Non impact
    - Laser printers, ink jet, bubble jet
    - In laser printers, laser beam creates a charge distribution on a rotating drum coated with a photoelectric materials such as selenium an tonner is applied to the drum and then transferred to paper
    - In ink jet printer, the ink is squirted in horizontal rows across a roll of paper. The electrically charged ink stream is deflected by an electric field to produce to matrix patterns
    - In bubble jet printers, ink is heated in a heating chamber to produce ink bubble. The heating chamber consists of heating filament

Computer Graphics
EG678EX

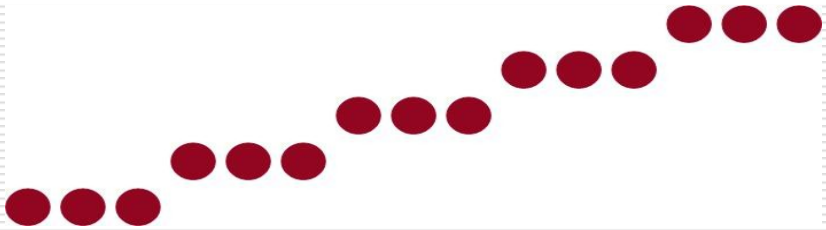
2-D Algorithms

Prepared By: Dipesh Gautam

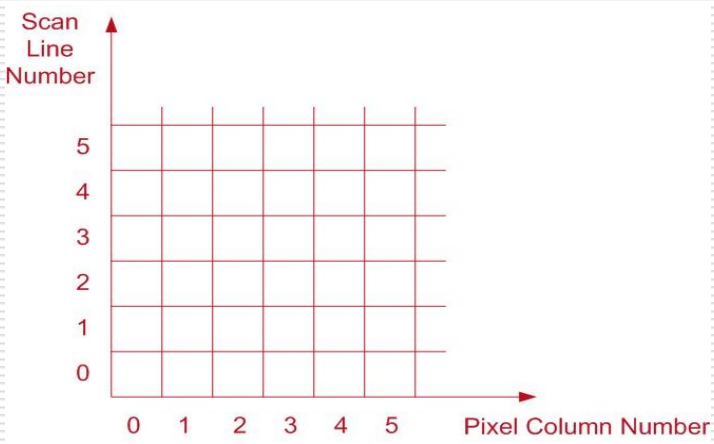
Points and Lines

- Points
  - Plotted by converting co-ordinate position to appropriate operations for the output device (e.g: in CRT monitor, the electron beam is turned on to illuminate the screen phosphor at the selected location.)
  - Line
    - Plotted by calculating intermediate positions along the line path between two specified endpoint positions.
    - Screen locations are referenced with integer values, so plotted positions may only approximate actual line positions between two specified endpoints → "***the jaggies***". E.g: position (10.48,20.51) → (10,21).

- Jaggies



- Pixel position: referenced by scan line number and column number



Line Drawing Algorithms

- Slope-Intercept Equation

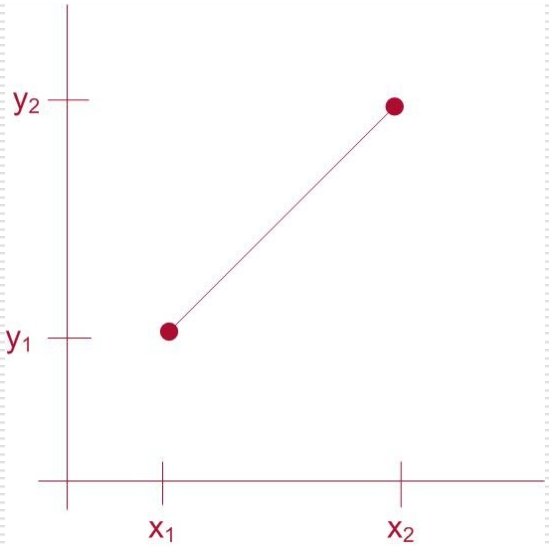
$$y = m.x + b$$

- Slope

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

- Intercept

$$b = y_1 - m.x_1$$



- Interval Calculation

$$\Delta y = m.\Delta x \qquad \Delta x = \frac{\Delta y}{m}$$

- Analog System
  - |m| <1
    - Set Δx proportional to horizontal deflection voltage. Then

$$\Delta y = m.\Delta x$$

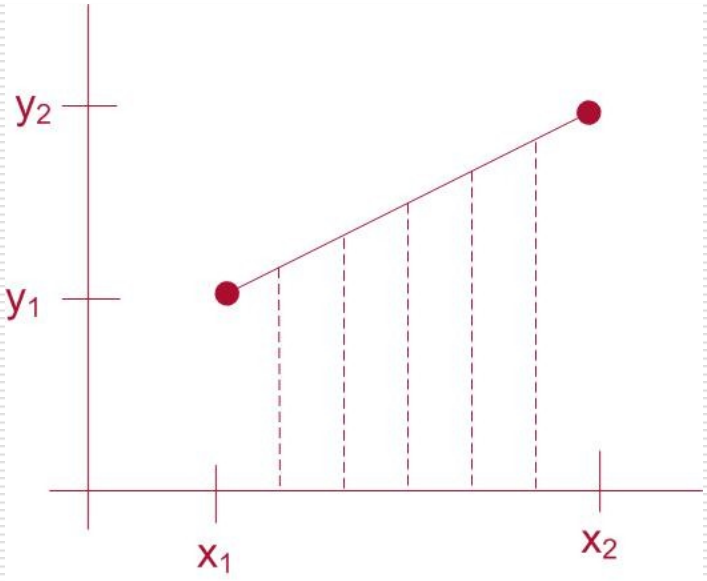
  - |m|>1
    - Set Δy proportional to vertical deflection voltage. Then

$$\Delta x = \frac{\Delta y}{m}$$

  - |m|=1
    - Δx = Δy ➔ horizontal and vertical deflection voltages are equal

Digital System

- Sample a line at discrete positions and determine nearest pixel to the line at each sampled position



DDA Algorithm

- → Digital Differential Analyzer
  - Sample the line at unit intervals in one coordinate
  - Determine the corresponding integer values nearest the line path in another co-ordinate

DDA Algorithm (left to right)

- Slope $m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k} = \frac{\Delta y}{\Delta x}$
- For |m|<1 (|Δy|< |Δx|)
  - Sample line at unit interval in x co-ordinate

$$y_{k+1} = y_k + m \qquad \Delta x = x_{k+1} - x_k = 1$$

- For |m|>1 (|Δy|> |Δx|)
  - Sample line at unit interval in y co-ordinate

$$x_{k+1} = x_k + \frac{1}{m} \qquad \Delta y = y_{k+1} - y_k = 1$$

DDA Algorithm (right to left)

- Slope $m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k} = \frac{\Delta y}{\Delta x}$
- For |m|<1 (|Δy|< |Δx|)
  - Sample line at unit interval in x co-ordinate

$$y_{k+1} = y_k - m \qquad \Delta x = x_{k+1} - x_k = -1$$

- For |m|>1 (|Δy|> |Δx|)
  - Sample line at unit interval in y co-ordinate

$$x_{k+1} = x_k - \frac{1}{m} \qquad \Delta y = y_{k+1} - y_k = -1$$

DDA Algorithm

1. Input the two line endpoints and store the left endpoint in (x_0,y_0)
2. Plot first point (x_0,y_0)
3. Calculate constants Δx, Δy
4. If $|\Delta x| > |\Delta y|$ *steps* = $|\Delta x|$ else *steps* = $|\Delta y|$
5. Calculate XInc = $|\Delta x|$ / *steps* and *Y*Inc = $|\Delta y|$ / *steps*
6. At each x_k along the line, starting at k=0, Plot the next pixel at (xk + XInc, yk + YInc)
7. Repeat step 6 *steps* times

Pseudo Code

```
Void lineDDA(int xa, int ya, int xb, int yb)
{
    int dx = xb – xa, dy = yb – ya, steps, k;
    float xIncrement, yIncrement, x = xa, y = ya;

    if( abs (dx) > abs (dy) ) steps = abs (dx);
    else steps = abs (dy);
    xIncrement = dx / (float) steps;
    yIncrement = dy / (float) steps;
    setPixel (ROUND (x), ROUND (y));
    for (k=0; k<steps; k++){
            x += xIncrement;
            y += yIncrement;
            setPixel (ROUND(x), ROUND(y));
    }
}
```

DDA Algorithm

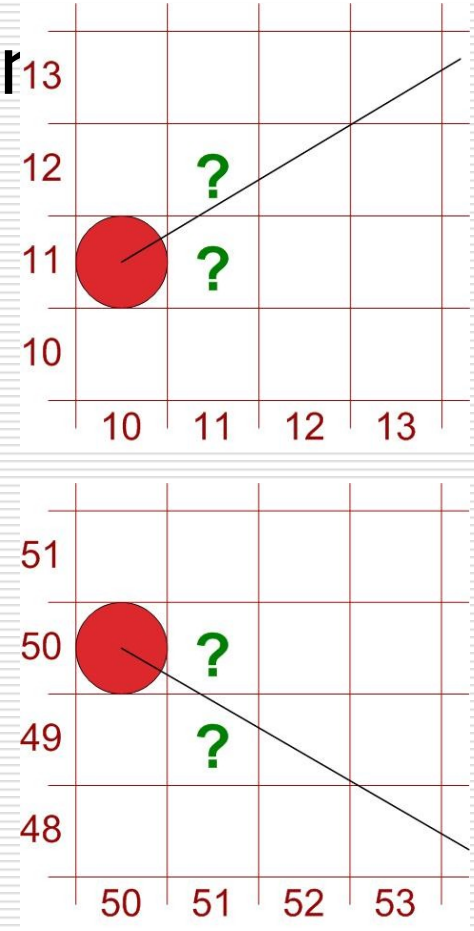
- How about problem and performance ?
  - Assignment:
    - What's the performance problem in above pseudo code ?
    - How DDA performance can be improved ?

Computer Graphics (L04)
EG678EX

2-D Algorithms

Bresenham's Line Algorithm

- Uses only incremental integer calculations
- Which pixel to draw ?
  - (11,11) or (11,12) ?
  - (51,50) or (51,49) ?
  - Answered by Bresenham



- For $|m|<1$
  - Start from left end point (x_0,y_0) step to each successive column (x samples) and plot the pixel whose scan line y value is closest to the line path.
  - After (x_k,y_k) the choice could be (x_k+1,y_k) or (x_k+1,y_k+1)

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

Then

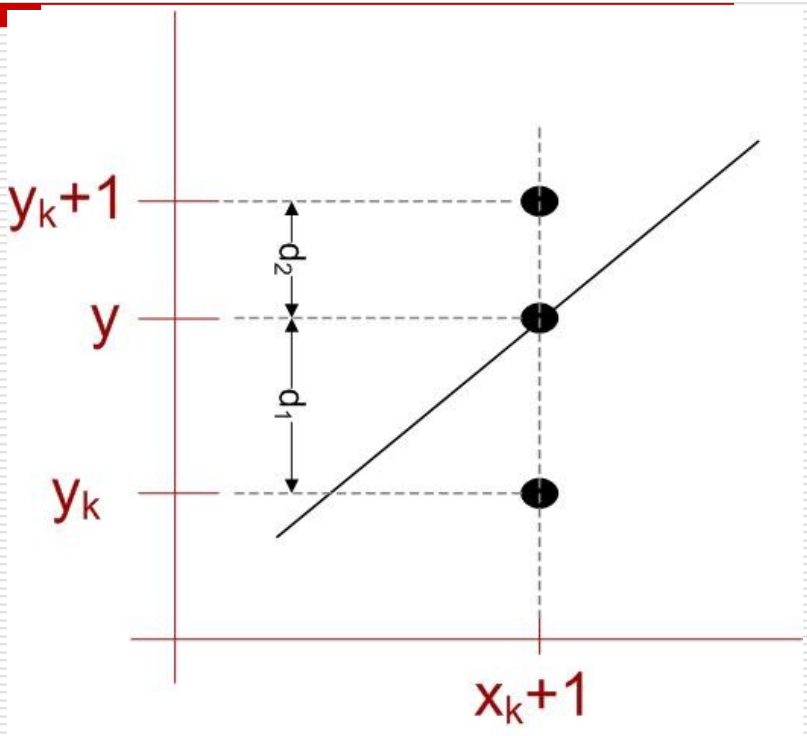
$$d_1 = y - y_k$$
$$= m(x_k + 1) + b - y_k$$

And

$$d_2 = (y_k + 1) - y$$
$$= y_k + 1 - m(x_k + 1) - b$$

Difference between separations

$$d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1$$



Defining decision parameter

Constant=2Δy + Δx(2b-1) Which is independent of pixel position

$$p_k = \Delta x(d_1 - d_2) \quad [1]$$
$$= 2\Delta y.x_k - 2\Delta x.y_k + c$$

Sign of p_k is same as that of d_1-d_2 for Δx>0 (left to right sampling)

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

c eliminated here

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

because $x_{k+1} = x_k + 1$

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

y_{k+1}-y_k = 0 if $p_k < 0$
y_{k+1}-y_k = 1 if $p_k \geq 0$

For Recursive calculation, initially

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

Substitute $b = y_0 - m.x_0$ and $m = \Delta y/\Delta x$ *in [1]*

Algorithm Steps (|m|<1)

1. Input the two line endpoints and store the left endpoint in (x_0,y_0)
2. Plot first point (x_0,y_0)
3. Calculate constants Δx, Δy, $2\Delta y$ and $2\Delta y - 2\Delta x$, and obtain $p_0 = 2\Delta y - \Delta x$
4. At each x_k along the line, starting at k=0, perform the following test:

   If $p_k<0$, the next point plot is (x_k+1,y_k) and

   $P_{k+1} = p_k + 2\Delta y$

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

   $P_{k+1} = p_k + 2\Delta y - 2\Delta x$
5. Repeat step 4 Δx times

What's the advantage?

- Answer: involves only the calculation of constants Δx, Δy, $2\Delta y$ and $2\Delta y - 2\Delta x$ once and integer addition and subtraction in each steps

Example

Endpoints (20,10) and (30,18)

Slope m = 0.8

$\Delta x = 10, \Delta y = 8$

$P_0 = 2\Delta y - \Delta x = 6$

$2\Delta y = 16, \quad 2\Delta y - 2\Delta x = -4$

Plot ?

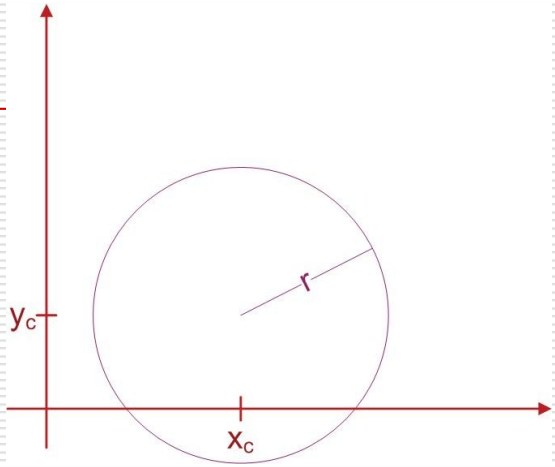
Plot $(x_0, y_0) = (20,10)$

k	p_k	(x_{k+1}, y_{k+1})	k	p_k	(x_{k+1}, y_{k+1})
0	6	(21,11)	5	6	(26,15)
1	2	(22,12)	6	2	(27,16)
2	-2	(23,12)	7	-2	(28,16)
3	14	(24,13)	8	14	(29,17)
4	10	(25,14)	9	10	(30,18)

Computer Graphics (L05)
EG678EX

2-D Algorithms

Circle-Generating Algorithms (Basic Foundations)



- Circle Equation:

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

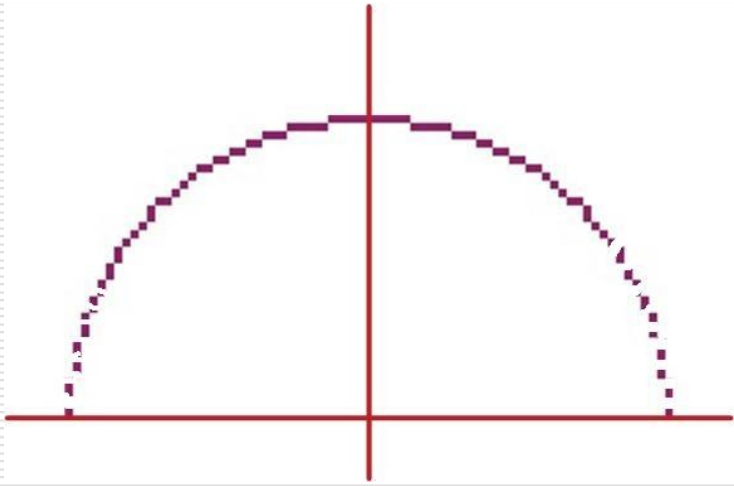
- Points along circumference could be calculated by stepping along x-axis:

$$y = y_c \pm \sqrt{r^2 - (x_c - x)^2}$$

- Problem ????

Problem (in above method)

- Computational complexity
- Spacing:
  - Non-uniform spacing of plotted pixels



Adjustments (To fix problems)

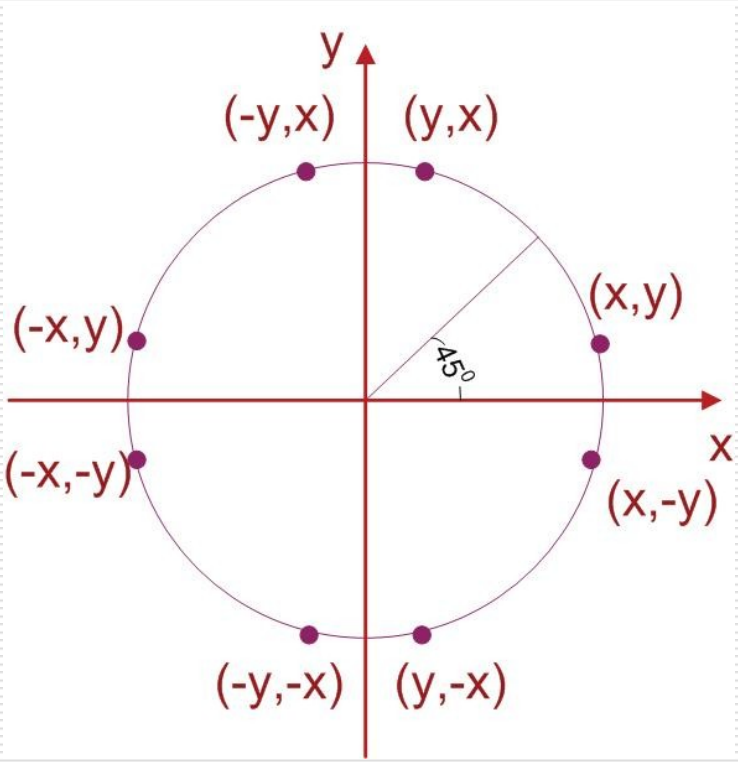
- Spacing problem (2 ways):
  - Interchange the role of x and y whenever the absolute value of the slope of the circle tangent > 1
  - Use polar co-ordinate:

$$x = x_c + r\cos\theta$$

$$y = y_c + r\sin\theta$$

    - Equally spaced points are plotted along the circumference with fixed angular step size.
    - step size chosen for θ depends on the application and display device.
      - For more continuous boundary on a raster display, set step size at 1/r; i.e pixel positions are approximately 1 unit apart.
- Computation Problem:
  - Use symmetry of circle; i.e calculate for one octant and use symmetry for others.

Circle Symmetry



Bresenham's Algorithm Could Be Adapted ??

- Yes
- How ?
  - Setting decision parameter for finding the closest pixel to the circumference
- And what to do For Non-linear equation of circle ?
  - Comparison of squares of the pixel separation distance avoids square root calculations

Computer Graphics (L06)
EG678EX

2-D Algorithms

Midpoint Circle Algorithm

- Circle function defined as:

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

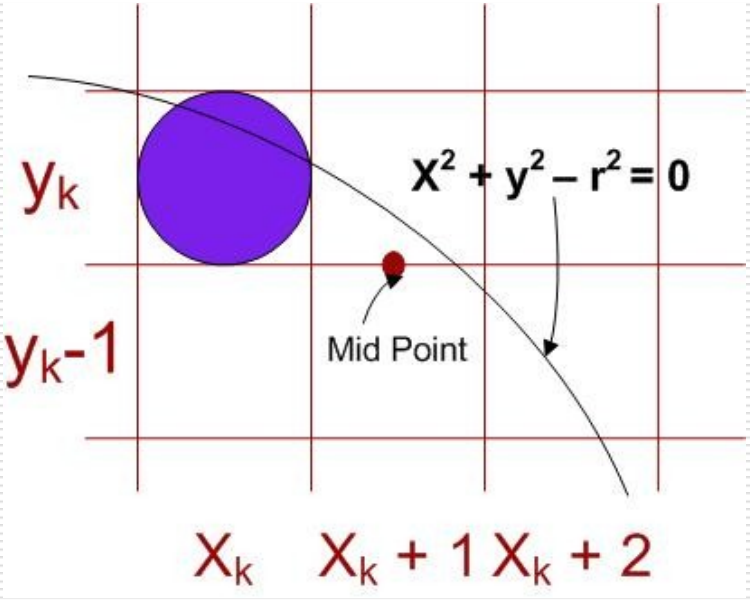
- Any point (x,y) satisfies following conditions

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

- Decision parameter is the circle function; evaluated as:

$$p_k = f_{circle}(x_k + 1, y_k - \frac{1}{2})$$

$$= (x_k + 1)^2 + (y_k - \frac{1}{2})^2 - r^2$$



$$p_{k+1} = f_{circle}(x_{k+1} + 1, y_{k+1} - \frac{1}{2})$$

$$= [(x_k + 1) + 1]^2 + (y_{k+1} - \frac{1}{2})^2 - r^2$$

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

$y_{k+1} = y_k$ *if* $p_k<0$

$y_{k+1} = y_k-1$ *otherwise*

- Thus

  $P_{k+1} = P_k + 2x_{k+1}+1$ *if* $pk<0$

  $P_{k+1} = P_k + 2x_{k+1}+1-2y_{k+1}$ *otherwise*

- Also incremental evaluation of $2x_{k+1}$ and $2y_{k+1}$

  $2x_{k+1} = 2x_k + 2$

  $2y_{k+1} = 2y_k - 2$ *if* $p_k >0$

- At start position $(x_0,y_0) = (0,r)$

  $2x_0 = 0$ *and* $2y_0 = 2r$

- Initial decision parameter

$$p_0 = f_{circle}(1, r - \frac{1}{2})$$
$$= 1 + (r - \frac{1}{2})^2 - r^2$$
$$= \frac{5}{4} - r$$

- For r specified as an integer, round p_0 to

$$P_0 = 1 - r$$

(because all increments are integers)

Algorithm

1. Input radius r and circle center (x_c, y_c) and obtain the first point on the circumference of a circle centered on the origin as

   $(x_0, y_0) = (0, r)$
2. Calculate the initial value of the decision parameter as

   $P_0 = 5/4 - r$
3. At each x_k position, starting at k = 0, perform the following test:

   If $p_k < 0$, the next point along the circle centered on $(0,0)$ is (x_{k+1}, y_k) and

   $P_{k+1} = p_k + 2x_{k+1} + 1$

   Otherwise, the next point along the circle is (x_k+1, y_k-1) and

   $P_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$

   Where $2x_{k+1} = 2x_k + 2$ and $2y_{k+1} = 2y_k - 2$
4. Determine the symmetry points in the other seven octants.
5. Move each calculated pixel position (x,y) onto the circular path centered on (x_c, y_c) and plot the co-ordinate values:

   $x = x + x_c, \quad y = y + y_c$
6. Repeat steps 3 through 5 until $x \geq y$

Computer Graphics (L07)
EG678EX

2-D Algorithms

Ellipse Generating Algorithms

- Equation of ellipse:

$$d_1 + d_2 = constant$$

- F1→(x_1,y_1), F2→(x_2,y_2)

$$\sqrt{(x - x_1)^2 + (y - y_1)^2} + \sqrt{(x - x_2)^2 + (y - y_2)^2} = constant$$

- General Equation

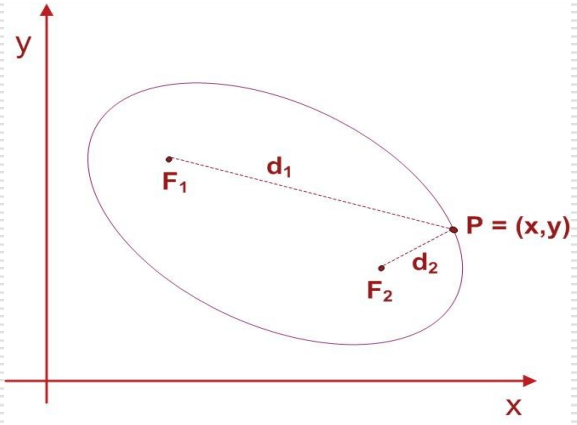
$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$$

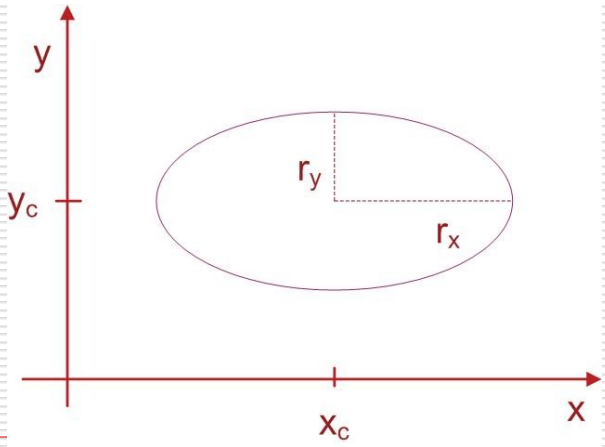
- Simplified Form

$$\left(\frac{x - x_c}{r_x}\right)^2 + \left(\frac{y - y_c}{r_y}\right)^2 = 1$$

- In polar co-ordinate

$$x = x_c + r_x \cos\theta$$
$$y = y_c + r_y \sin\theta$$



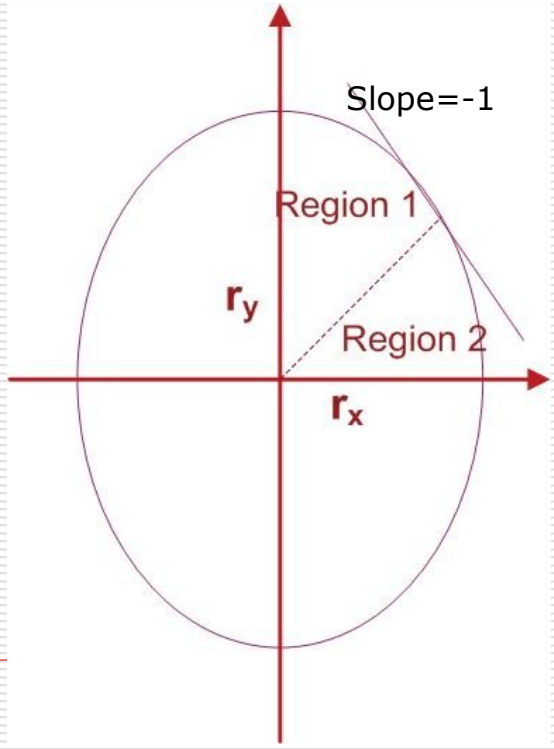


Ellipse function

$$f_{ellipse}(x,y) = r_y^{\,2}x^2 + r_x^{\,2}y^2 - r_x^{\,2}r_y^{\,2}$$

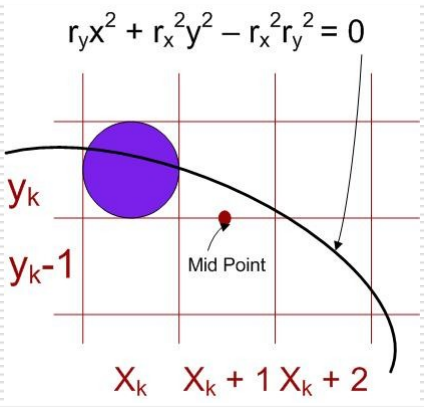
$$f_{ellipse}(x,y)\begin{cases} <0, & if\,(x,y) \quad is \;\; inside \;\; the \;\; ellipse \;\; boundary \\ =0, & if\,(x,y) \quad is \;\; on \;\; the \;\; ellipse \;\; boundary \\ >0, & if\,(x,y) \quad is \;\; outside \;\; the \;\; ellipse \;\; boundary \end{cases}$$

- From ellipse tangent slope: $\frac{dy}{dx} = -\frac{2r_y^2 x}{2r_x^2 y}$
- At boundary region (slope = -1): $2r_y^2 x = 2r_x^2 y$
- Start from $(0, r_y)$, take x samples to boundary between 1 and 2
- Switch to sample y from boundary between 1 and 2
  (i.e whenever $2r_y^2 x \geq 2r_x^2 y$)



- In the region 1

$$p1_k = f_{ellipse}(x_k + 1, y_k - \frac{1}{2})$$
$$= r_y^2 (x_k + 1)^2 + r_x^2 (y_k - \frac{1}{2})^2 - r_x^2 r_y^2$$



- For next sample

$$p1_{k+1} = f_{ellipse}(x_{k+1} + 1, y_{k+1} - \frac{1}{2})$$
$$= r_y^2 [(x_k + 1) + 1]^2 + r_x^2 (y_{k+1} - \frac{1}{2})^2 - r_x^2 r_y^2$$
$$p1_{k+1} = p1_k + 2r_y^2 (x_k + 1) + r_y^2 + r_x^2 \left[\left(y_{k+1} - \frac{1}{2} \right)^2 - \left(y_k - \frac{1}{2} \right)^2 \right]$$

- Thus increment

$$increment = \begin{cases} 2r_y^2 x_{k+1} + r_y^2, & if \quad p1_k < 0 \\ 2r_y^2 x_{k+1} + r_y^2 - 2r_x^2 y_{k+1}, & if \quad p1_k \geq 0 \end{cases}$$

- For increment calculation; Initially:

$$2r_y^2 x = 0$$
$$2r_x^2 y = 2r_x^2 r_y$$

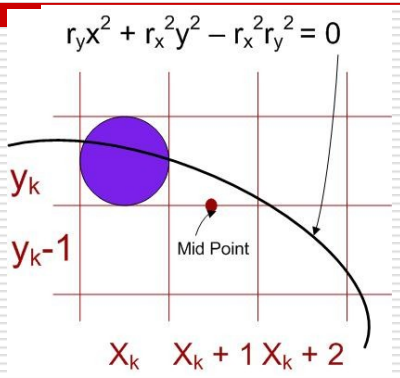
- Incrementally:
  Update x by adding $2r_y^2$ to first equation and update y by subtracting $2r_x^2$ to second equation

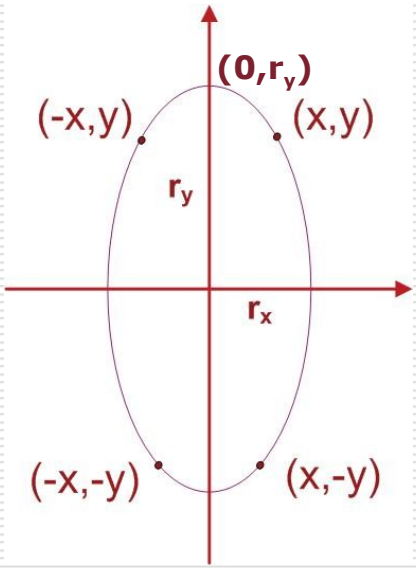
- Initial value

$$p1_0 = f_{ellipse}\left(1, r_y - \frac{1}{2}\right)$$

$$= r_y^2 + r_x^2\left(r_y - \frac{1}{2}\right)^2 - r_x^2 r_y^2$$

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$





- In the region 2

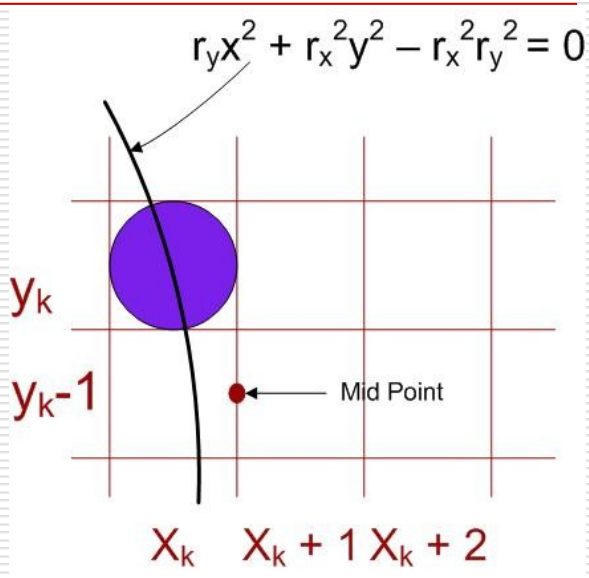
$$p2_k = f_{ellipse}(x_k + \frac{1}{2}, y_k - 1)$$
$$= r_y^2 (x_k + \frac{1}{2})^2 + r_x^2 (y_k - 1)^2 - r_x^2 r_y^2$$

- For next sample

$$p2_{k+1} = f_{ellipse}(x_{k+1} + \frac{1}{2}, y_{k+1} - 1)$$
$$= r_y^2 \left(x_{k+1} + \frac{1}{2} \right)^2 + r_x^2 [(y_k - 1) - 1]^2 - r_x^2 r_y^2$$
$$p2_{k+1} = p2_k + 2r_x^2 (y_k - 1) + r_x^2 + r_y^2 \left[\left(x_{k+1} + \frac{1}{2} \right)^2 - \left(x_k - \frac{1}{2} \right)^2 \right]$$

- Initially

$$p2_0 = f_{ellipse} \left(x_0 + \frac{1}{2}, y_0 - 1 \right)$$
$$p2_0 = r_y^2 \left(x_0 + \frac{1}{2} \right)^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$



For simplification calculation of $p2_0$ can be done by selecting pixel positions in counter clockwise order starting at $(r_x,0)$ and unit samples to positive y direction until the boundary between two regions

Algorithm

1. Input r_x,r_y, and the ellipse center(x_c,yc) and obtain the first point on an ellipse centered on the origin as

   $(x_0,y_0) = (0,r_y)$
2. Calculate the initial value of the decision parameter in region 1 as

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

3. At each x_k position in region 1, starting at k = 0, perform the following test: If $p1_k < 0$, the next point along the ellipse centered on (0,0) is (x_{k+1},y_k) and

$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} + r_y^2$$

Otherwise, the next point along the ellipse is (x_k+1,y_k-1) and

$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$$

With

$$2r_y^2 x_{k+1} = 2r_y^2 x_k + 2r_y^2, \qquad 2r_x^2 y_{k+1} = 2r_x^2 y_k - 2r_x^2$$

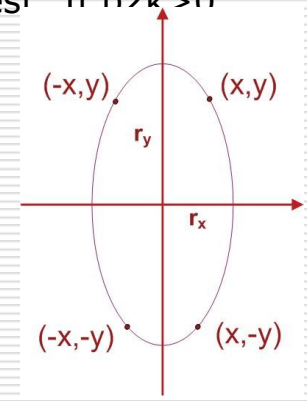
and continue until $2r_y^2 x \geq 2r_x^2 y$

4. Calculate the initial value of decision parameter in region 2 using the last point (x_0,y_0) calculated in region 1 as

$$p2_0 = r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2$$

5. At each y_k position in region 2, starting at k = 0, perform the following test. If p2k>0 the next point along the ellipse centered on (0,0) is (x_k,y_k-1) and

$$p2_{k+1} = p2_k - 2r_x^2 y_{k+1} + r_x^2$$



Otherwise the next point along the ellipse is (x_k+1,y_k-1) and

$$p2_{k+1} = p2_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$$

Using the same incremental calculations for x and y as in region 1.

6. Determine the symmetry points in the other three quadrants.
7. Move each calculated pixel position (x,y) onto the elliptical path centered on (x_c,y_c) and plot the co-ordinate values:

$$X = x + x_c,\ y = y + y_c$$

8. Repeat the steps for region 1 until

$$2r_y^2 x \geq 2r_x^2 y$$

Computer Graphics (L08)
EG678EX

2-D Algorithms

Translation

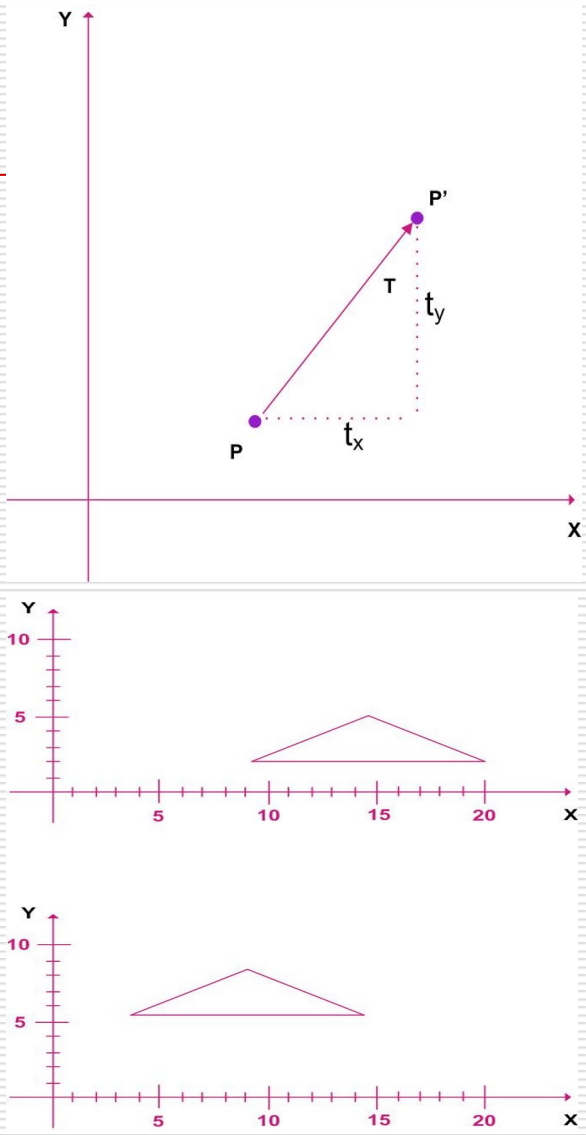
$$x' = x + t_x, \qquad y' = y + t_y$$

$$P = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \qquad p' = \begin{bmatrix} x_1' \\ x_2' \end{bmatrix}, \qquad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$P' = P + T$$

In homogeneous representation if position P = (x,y) is translated to new position p'= (x',y') then:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow P' = T(t_x, t_y).P$$



Rotation

$$x' = r\cos(\phi+\theta) = r\cos\phi.\cos\theta - r\sin\phi.\sin\theta$$
$$y' = r\sin(\phi+\theta) = r\cos\phi.\sin\theta + r\sin\phi.\cos\theta$$

$$x = r\cos\phi, \qquad y = r\sin\phi$$

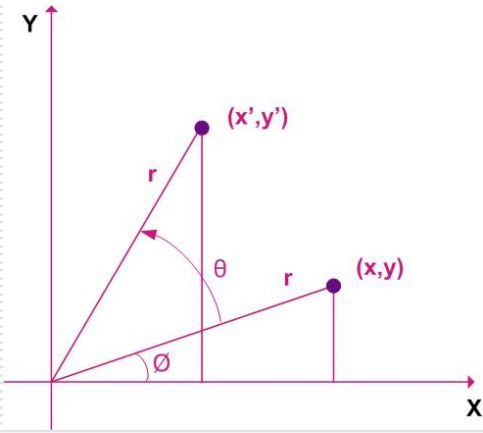
$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

$$P' = R.P$$

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

If Co-ordinates represented as row vector, Then:

$$P'^{T} = (R.P)^{T}$$
$$= P^{T}.R^{T}$$



In homogeneous co-ordinate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow P' = R(\theta).P$$

General Pivot Rotation

$$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$$
$$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$$

Steps
1. Translate object so as to coincide pivot to origin
2. Rotate object about the origin
3. Translate object back so as to return pivot to original position

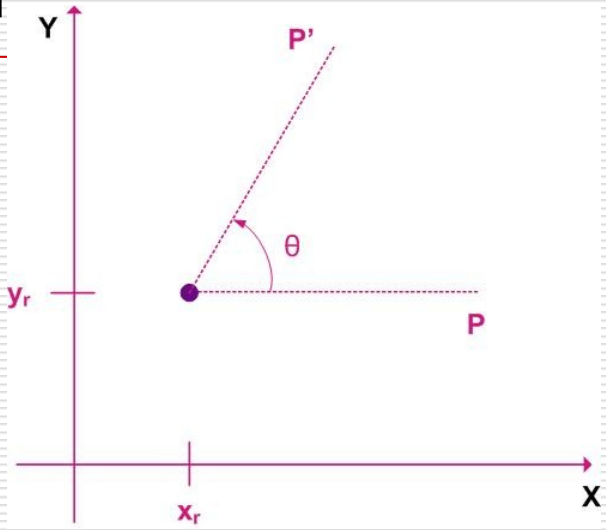
Composite Transformations

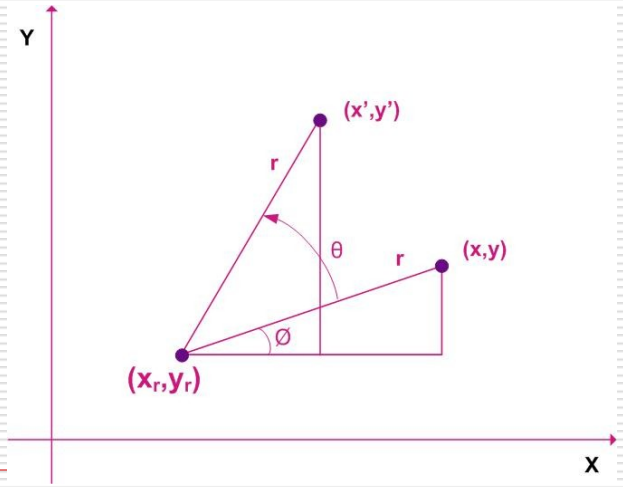
$$\begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix}$$

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

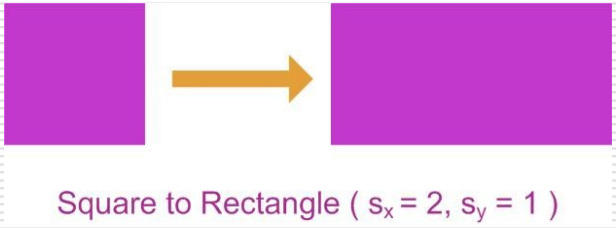
$$T(x_r, y_r).R(\theta).T(-x_r, -y_r) = R(x_r, y_r, \theta)$$

$$T(-x_r, -y_r) = T^{-1}(x_r, y_r)$$





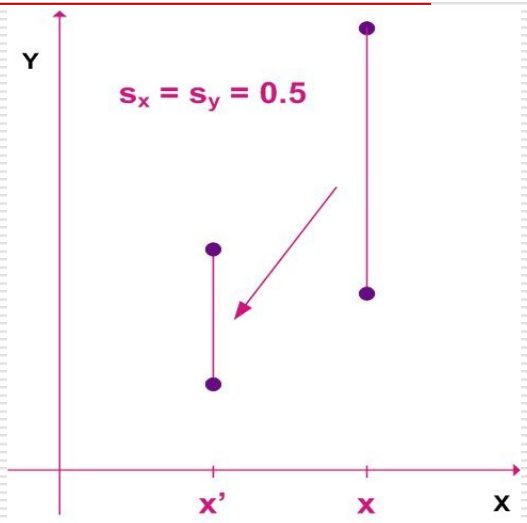
Scaling



$$x' = x.s_x, \qquad y' = y.s_y$$

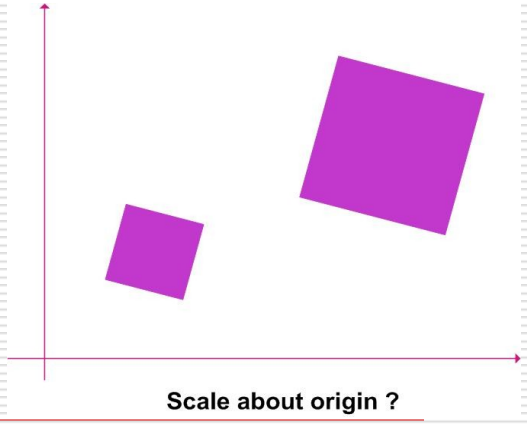
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} . \begin{bmatrix} x \\ y \end{bmatrix}$$

$$P' = S.P$$

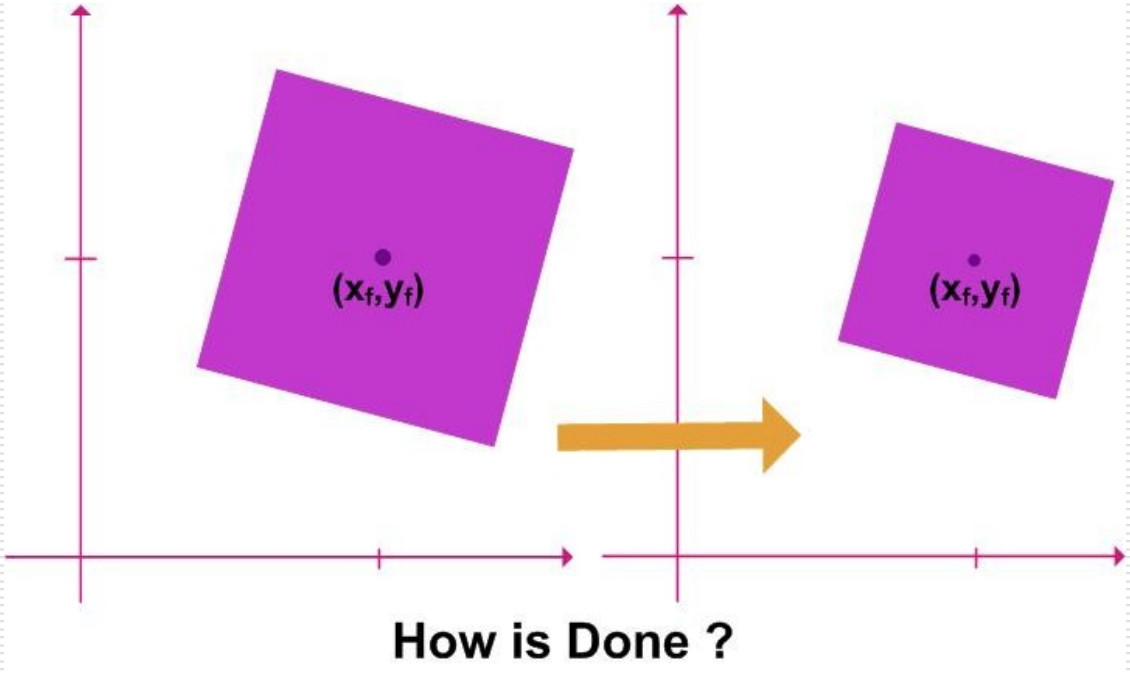


In homogeneous co-ordinate

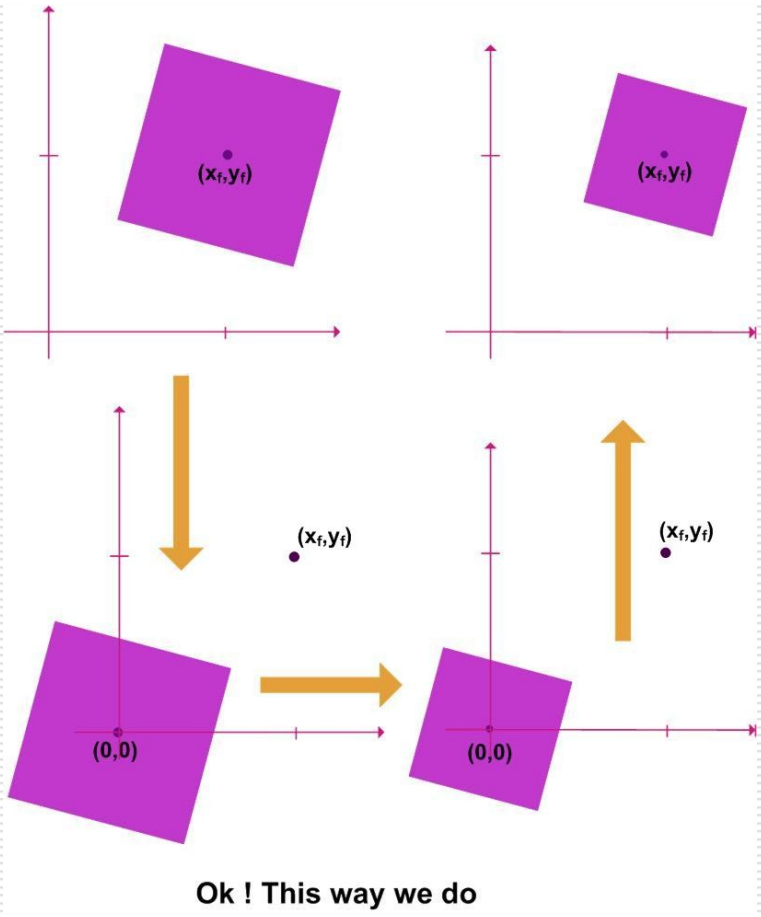
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow P' = S(s_x, s_y).P$$

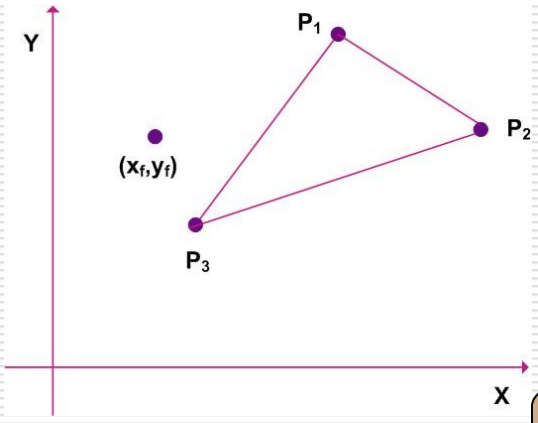


Fixed Point Scaling



Fixed Point Scaling





$$x' = x_f + (x - x_f).s_x,$$
$$y' = y_f + (y - y_f).s_y$$

$$x' = x.s_x + x_f(1 - s_x)$$
$$y' = y.s_y + y_f(1 - s_y)$$

Additive terms $x_f(1-s_x)$ and $y_f(1-s_y)$ are constant for all points in the object

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} . \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_f(1 - s_x) \\ y_f(1 - s_y) \end{bmatrix}$$

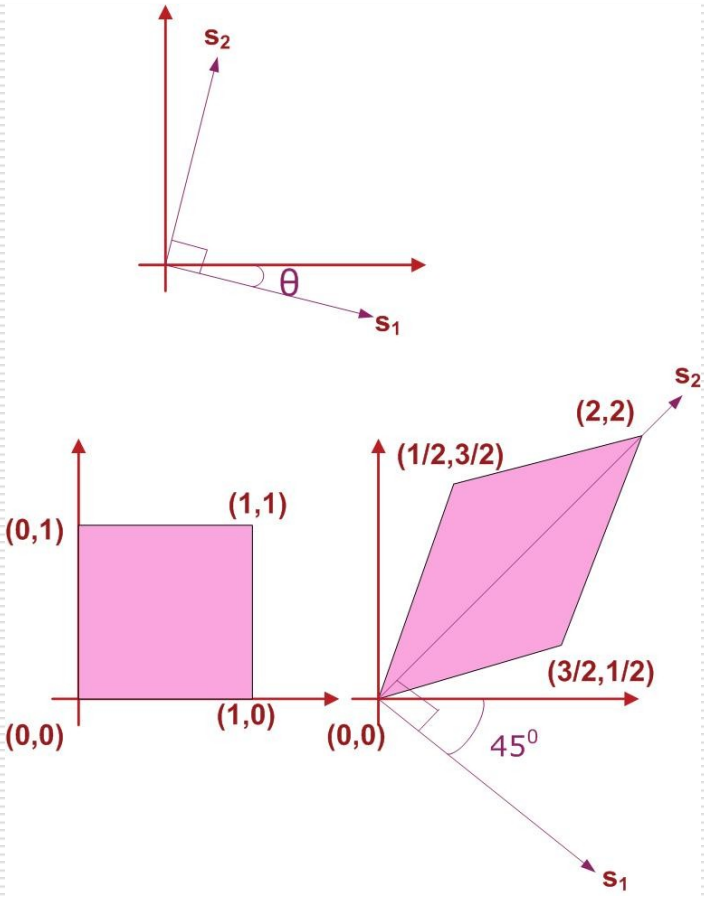
$$P' = S.P + C$$

General Scaling Directions

- Rotate the object so that x and y axes coincide with s_1 and s_2
- Apply scaling transformation in s_1 and s_2 direction
- Rotate the object in opposite direction to return points to the original orientations
- Example: square converted to parallelogram with s_1=1 and s_2 = 2 and θ = 45^0 as shown in figure

$$R^{-1}(\theta).S(s_1,s_2).R(\theta)$$

$$= \begin{bmatrix} s_1\cos^2\theta + s_2\sin^2\theta & (s_2 - s_1)\cos\theta.\sin\theta & 0 \\ (s_2 - s_1)\cos\theta.\sin\theta & s_1\sin^2\theta + s_2\cos^2\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Composite Transformations

For Successive Translation vectors (tx1,ty1) and (tx2,ty2)

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

For Successive Rotations θ_1 and θ_2

$$P' = R(\theta_2).\{R(\theta_1).P\}$$
$$= \{R(\theta_2).R(\theta_1)\}.P$$

Successive Translations are additive

$$\begin{bmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{x1} + t_{x2} \\ 0 & 1 & t_{y1} + t_{y2} \\ 0 & 0 & 1 \end{bmatrix}$$

$$T(t_{x2}, t_{y2}).T(t_{x1}, t_{y1}) = T(t_{x1} + t_{x2}, t_{y1} + t_{y2})$$

Successive Rotations are additive.

$$R(\theta_2).R(\theta_1) = R(\theta_1 + \theta_2)$$

!!!!!!! PROVE YOURSELF !!!!!!!!!

Composite Transformations

Successive Scaling are multiplicative

$$S(s_{x2}, s_{y2}).S(s_{x1}, s_{y1}) = S(s_{x1}.s_{x2}, s_{y1}.s_{y2})$$

$$\begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix}.\begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{x1}.s_{x2} & 0 & 0 \\ 0 & s_{y1}.s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Fixed Point Scaling

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

$$T(x_f, y_f).S(s_x, s_y).T(-x_f, -y_f) = S(x_f, y_f, s_x, s_y)$$

Concatenation Properties

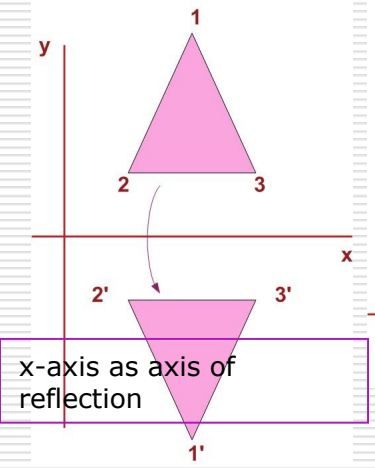
Matrix multiplication is associative, so

A.B.C = (A.B).C = A.(B.C)
Transformation product is not commutative, so

A.B ≠ B.A

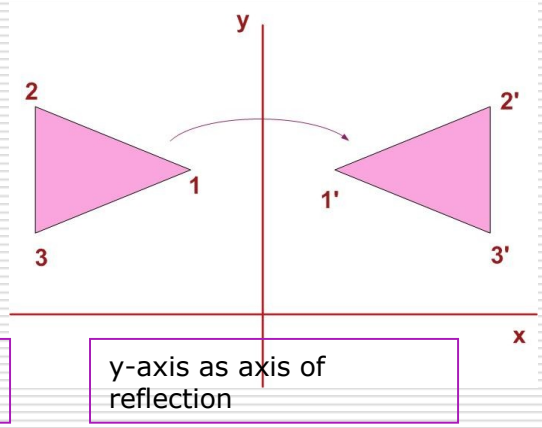
Reflection

- Transformation that produces mirror image of an object
- Mirror image is produced when rotated 180 degree about axis of reflection



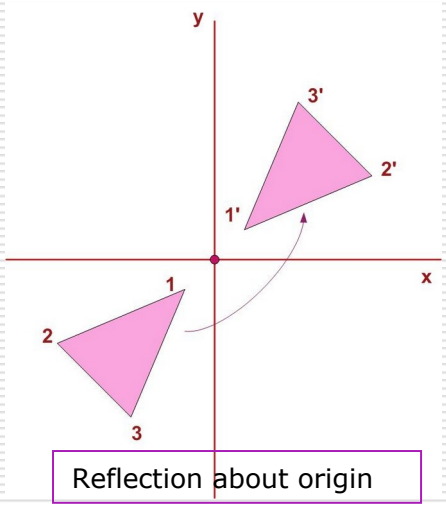
x-axis as axis of reflection

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



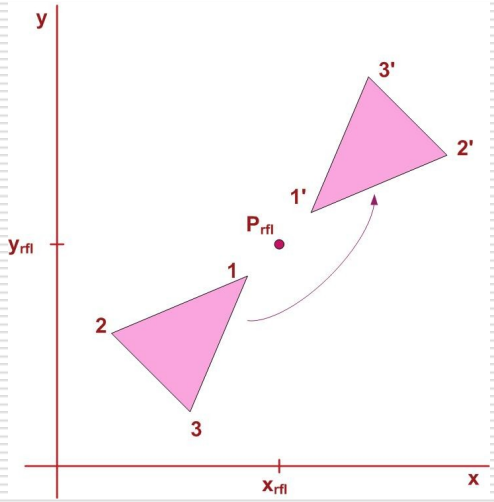
y-axis as axis of reflection

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



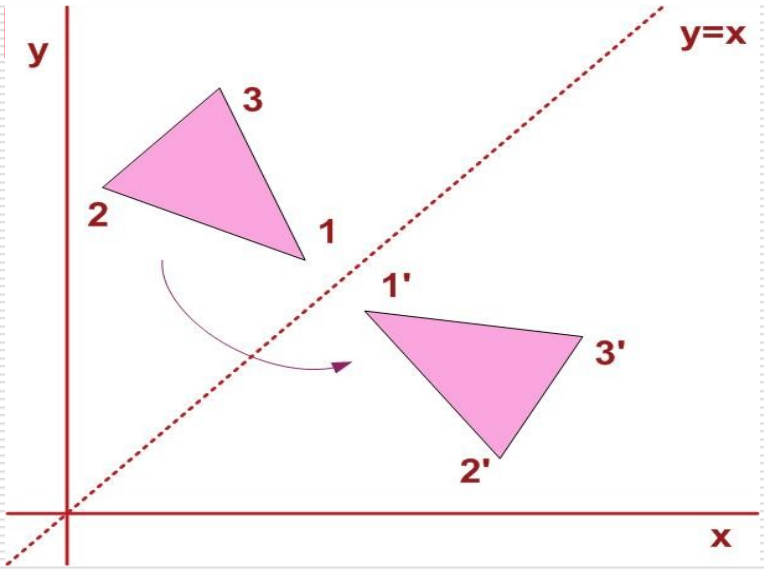
Reflection about origin

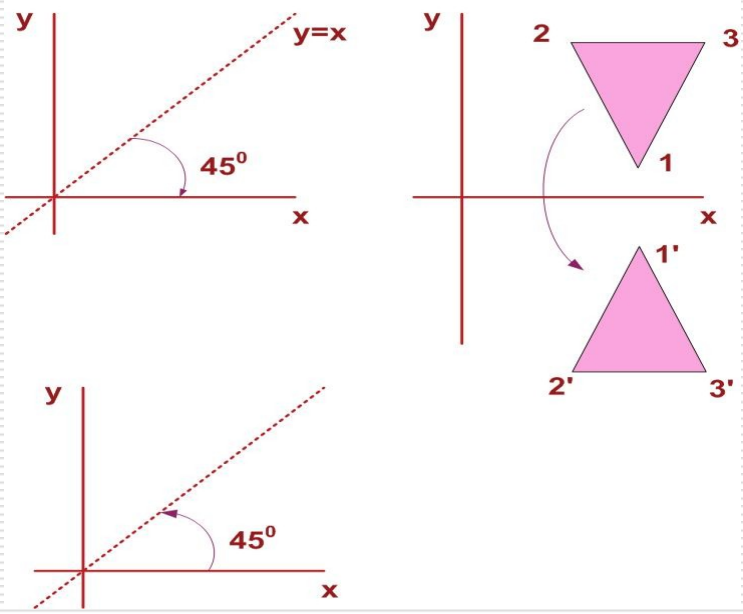
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Reflection about fixed point

Reflection about a line





- Reflection about a line y=x can be accomplished in the sequence of left→right→down in the second figure. The reflection matrix is as follows:

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

- Similar sequence could be applied for line y=mx + b

Shear

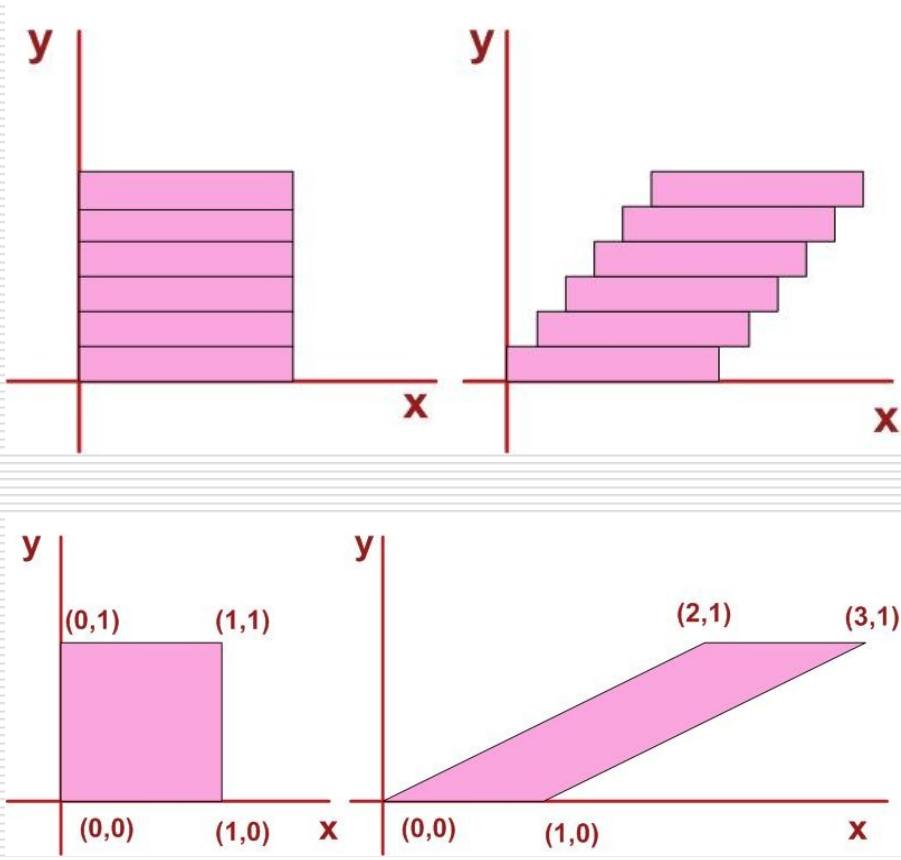
- Transformation that distorts the shape of an object such that Internal layers are shifted w.r.t each other.
- Mathematically: For fixed y, all points are shifted by fixed amount in the x-direction

$$x' = x + sh_x.y$$
$$y' = y$$

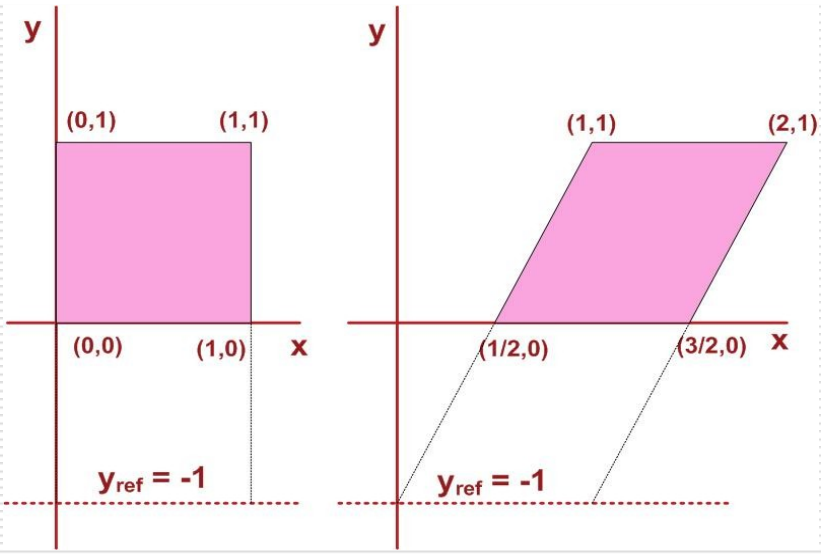
- All points on reference line (y=0 in figure) stays fixed under transformation

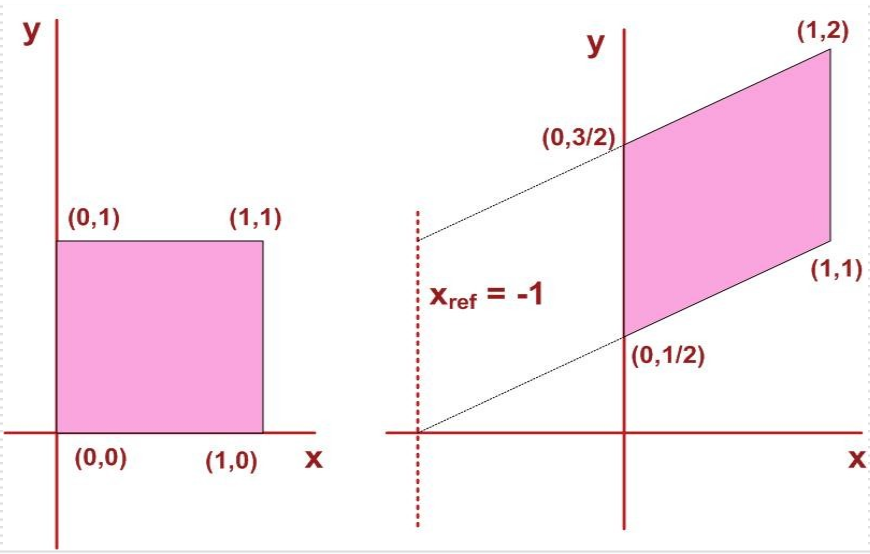
- Shear matrix in homogeneous co-ordinate:

$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Shear





- Shear w.r.t $y_{ref} = -1$

$$x' = x + sh_x(y - y_{ref}), \quad y' = y$$

$$\begin{bmatrix} 1 & sh_x & -sh_x.y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1/2 & -(1/2).(-1) \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Shear w.r.t $x_{ref} = -1$

$$x' = x, \quad y' = y + sh_y(x - x_{ref})$$

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y.x_{ref} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & -(1/2).(-1) \\ 0 & 0 & 1 \end{bmatrix}$$

Computer Graphics (L09)
EG678EX

3-D Transformations

Translation

$x' = x + t_x$

$y' = y + t_y$

$z' = z + t_z$

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

$$P' = T.P$$

Rotation (about co-ordinate axes)

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$
$$z' = z$$

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

$$P' = R_z(\theta).P$$

$$y' = y\cos\theta - z\sin\theta$$
$$z' = y\sin\theta + z\cos\theta$$
$$x' = x$$

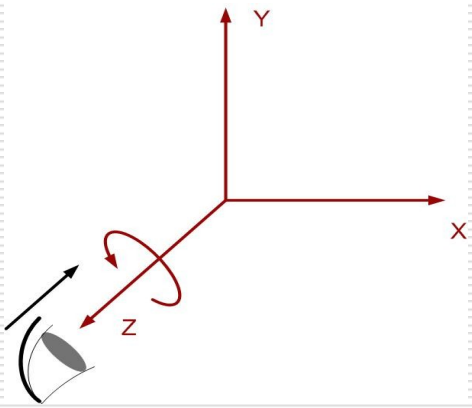
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

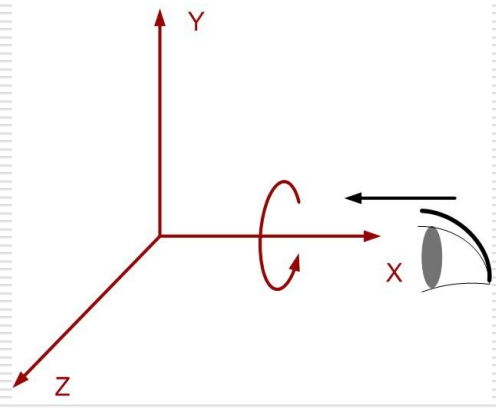
$$P' = R_x(\theta).P$$

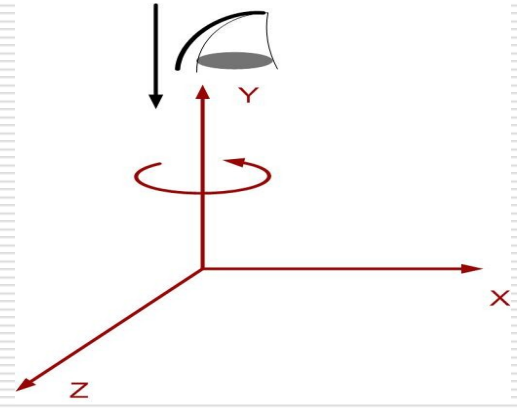
$$z' = z\cos\theta - x\sin\theta$$
$$x' = z\sin\theta + x\cos\theta$$
$$y' = y$$

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

$$P' = R_y(\theta).P$$



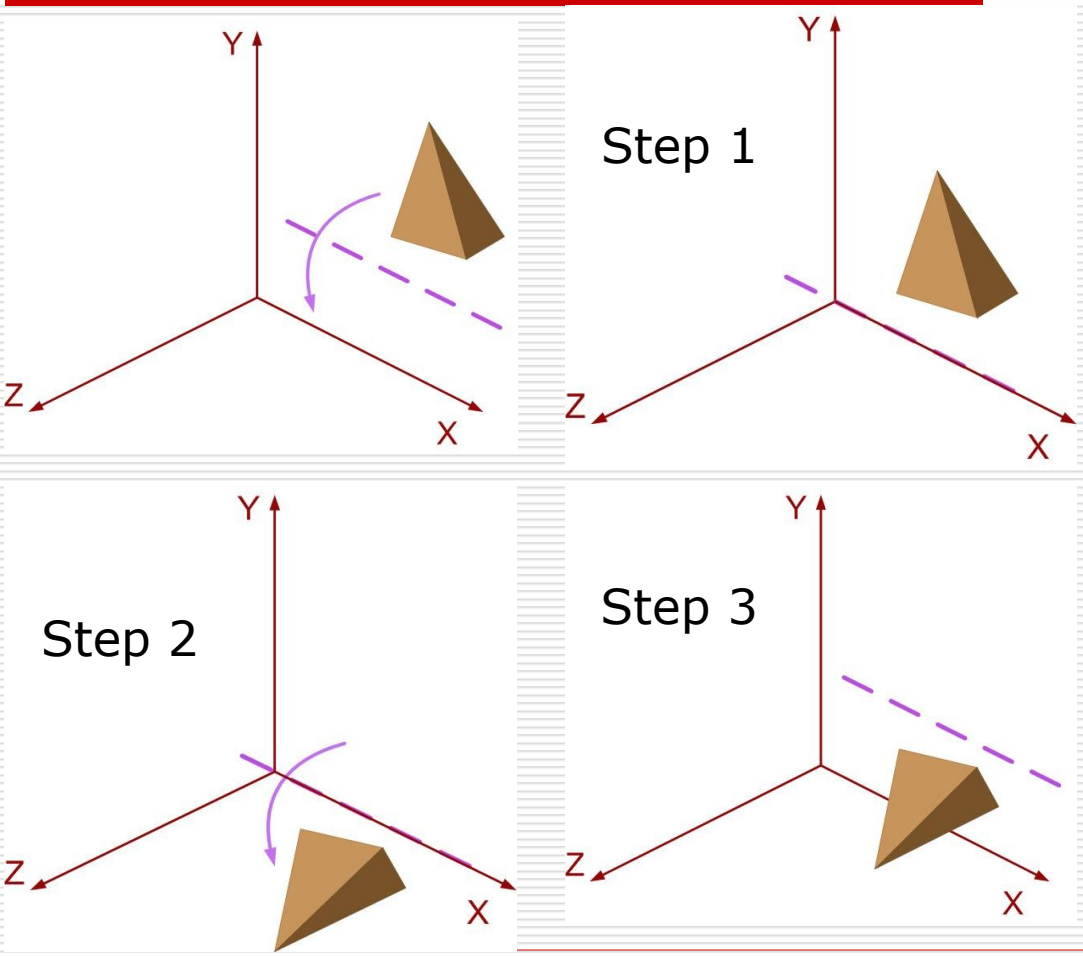




Remember The cyclic order:

$$x \rightarrow y \rightarrow z \rightarrow x$$

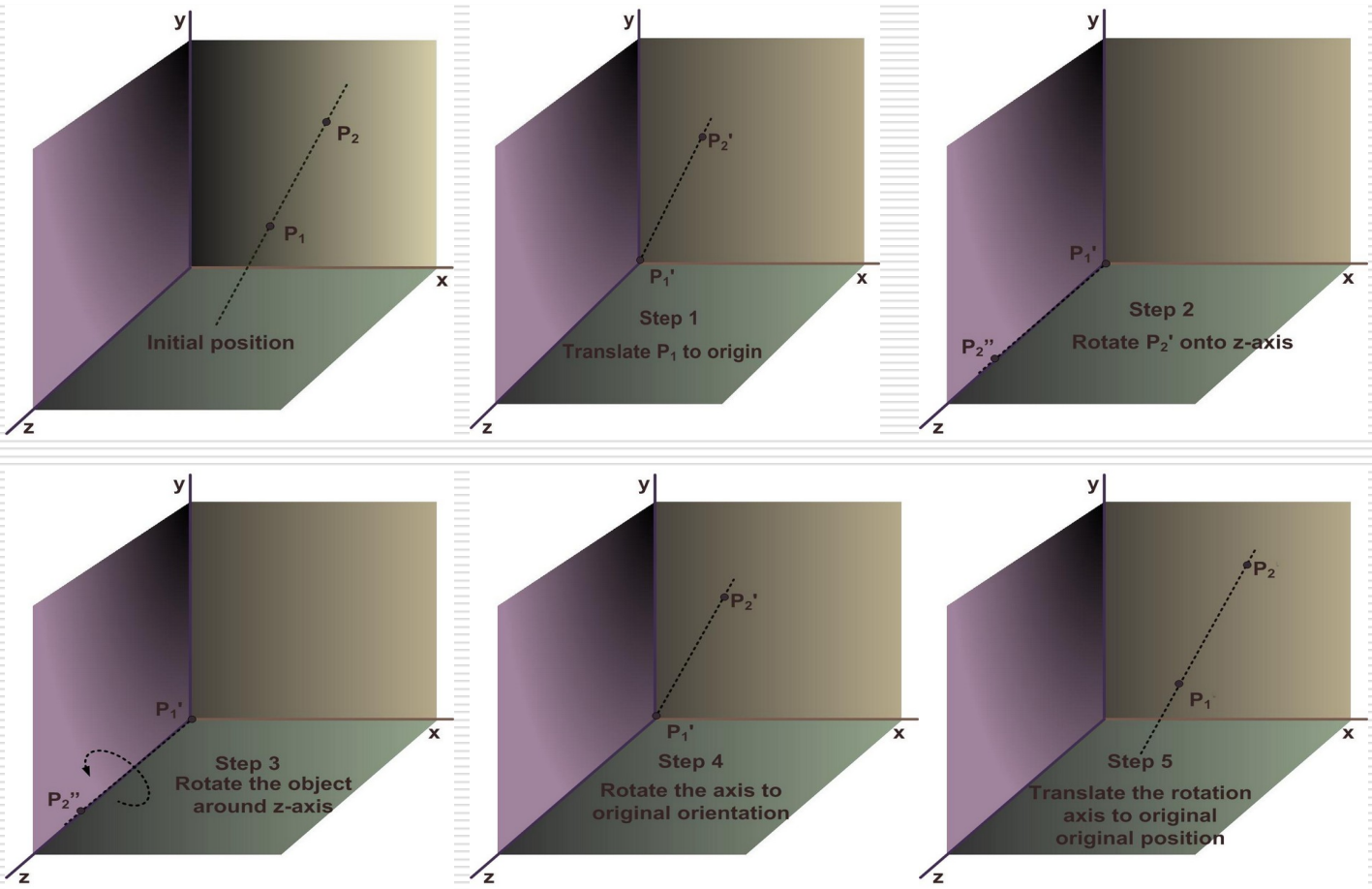
Rotation about axis parallel to co-ordinate axis



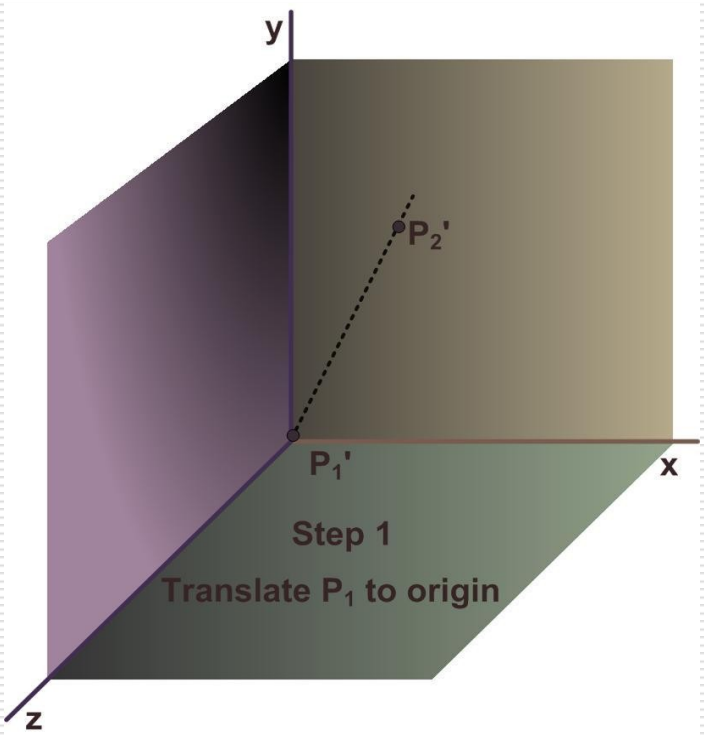
Steps

- Translate the object so as to coincide rotation axis to parallel co-ordinate axis
- Perform the rotation about the axis
- Translate back the object so as to move rotation axis to original position

General 3-D Rotation



General 3-D rotation Mathematics

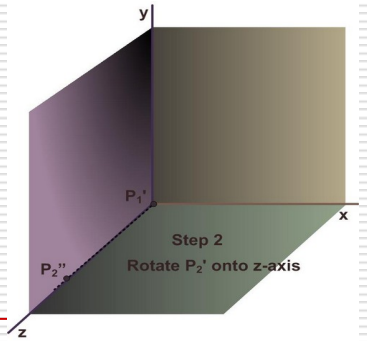


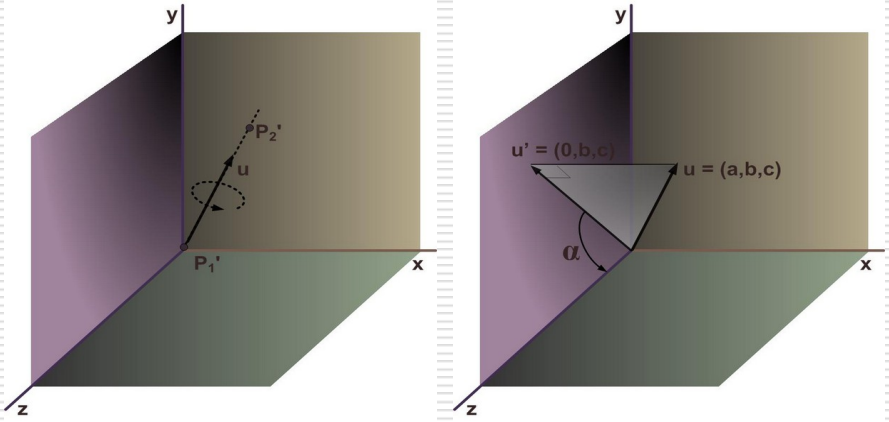
Step 1

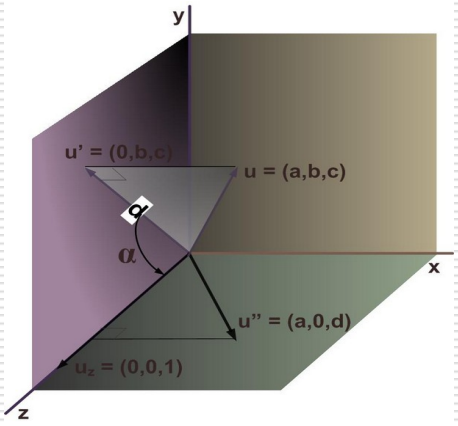
Translate object so as to coincide P_1 to origin

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

General 3-D Rotation Mathematics







Step 2

- ❑ Accomplished in two steps

i. Perform x axis rotation with angle α so as to bring rotation axis to zx plane

ii. Perform y axis rotation with angle ß so as to coincide the rotation axis with z-axis

X- Axix rotation (α)

$$V = p_2 - p_1 = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

$$\vec{u} = \frac{V}{|V|} = (a,b,c)$$

$$a = \frac{x_2 - x_1}{|V|}, \quad b = \frac{y_2 - y_1}{|V|}, \quad c = \frac{z_2 - z_1}{|V|}$$

$$\cos\alpha = \frac{u' \cdot u_z}{|u'||u_z|} = \frac{c}{d} \qquad d = \sqrt{b^2 + c^2}$$

$$u' \times u_z = u_x |u'||u_z| \sin\alpha$$

$$\vec{u'} \times \vec{u_z} = \begin{vmatrix} u_x & u_y & u_z \\ 0 & b & c \\ 0 & 0 & 1 \end{vmatrix} = \vec{u_x}.b$$

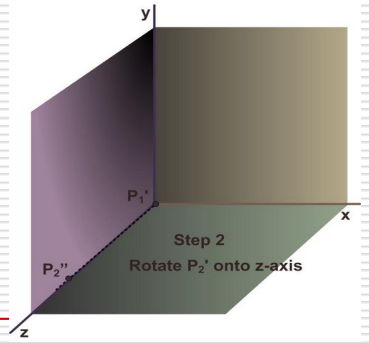
$$|u'| = d \text{ and } |u_z| = 1$$

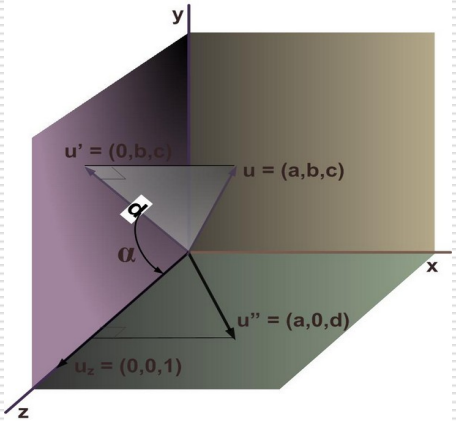
$$d\sin\alpha = b$$

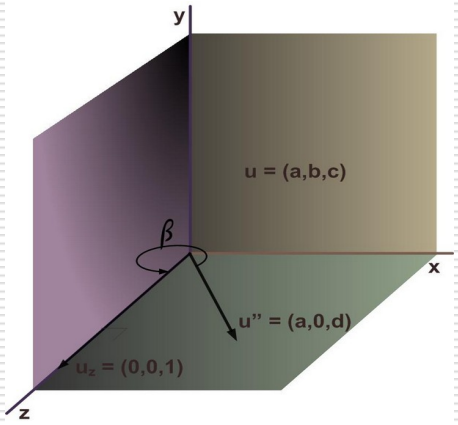
$$\sin\alpha = \frac{b}{d}$$

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

General 3-D Rotation Mathematics







Y-Axis Rotation (β)

$$\cos\beta = \frac{\vec{u}''.\vec{u}_z}{|u''||u_z|} = d$$

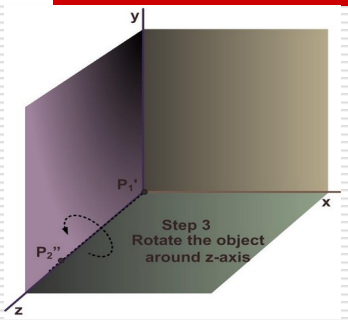
$$u'' \times u_z = u_y |u''||u_z| \sin\beta$$

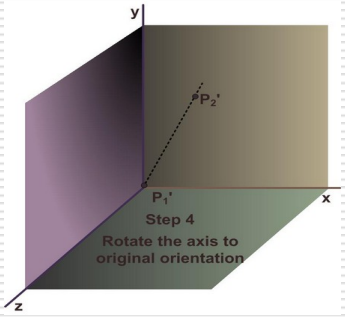
$$\vec{u}'' \times \vec{u}_z = \begin{vmatrix} u_x & u_y & u_z \\ a & 0 & d \\ 0 & 0 & 1 \end{vmatrix} = \vec{u}_y.(-a)$$

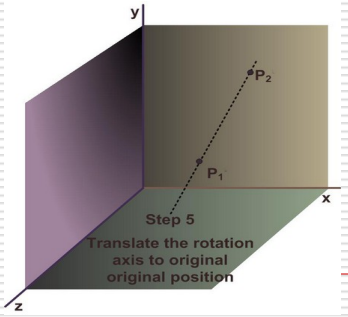
$$\sin\beta = -a$$

$$R_y(\beta) = \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

General 3-D Rotation Mathematics







Step 3

❑Perform z-axis Rotation with angle θ (the angle which the object is to be rotated about the given axis)

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

Step 4

❑Perform inverse Rotation and i.e $\mathbf{R_x^{-1}(\alpha)}$ and $\mathbf{R_y^{-1}(ß)}$

Step 5

❑Perform inverse Translation i.e $\mathbf{T^{-1}}$

Final composite Matrix is obtained as:

$$R(\theta) = T^{-1}.R_x^{-1}(\alpha).R_y^{-1}(\beta).R_z(\theta).R_y(\beta).R_x(\alpha).T$$

3-D Scaling

Scaling about origin

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

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fixed Point Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & (1- s_x)x_f \\ 0 & s_y & 0 & (1- s_y)y_f \\ 0 & 0 & s_z & (1- s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$S(x_f, y_f, z_f, s_x, s_y, s_z) = T(x_f, y_f, z_f).S(s_x, s_y, s_z).T(- x_f, - y_f, - z_f)$$

3-D Reflection

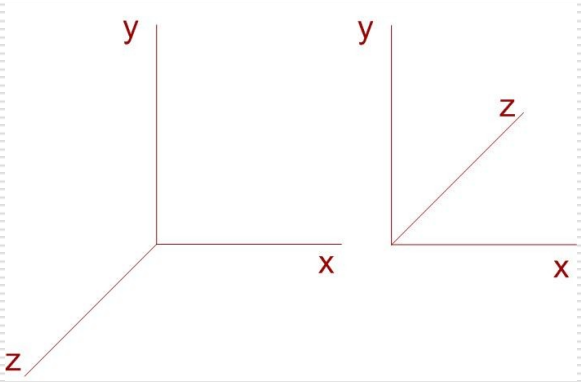
- Performed relative to a reflection axis or reflection plane
- Axis reflection→ equivalent to 180 degree rotation about the axis in 3-D space
- Plane reflection → equivalent to 180 degree rotation in 4-D space
  - 4-D space ?? → not visualized in euclidian space
- Reflection about a plane converts right handed co-ordinate system to left handed co-ordinate system and vice versa

- Reflection in xy plane

  $x' = x$
  $y' = y$
  $z' = -z$

  - Matrix is as

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



3-D Shear

- Z-axis shear

  x′ = x + a.z

  y′ = y + b.z

  z′ = z

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

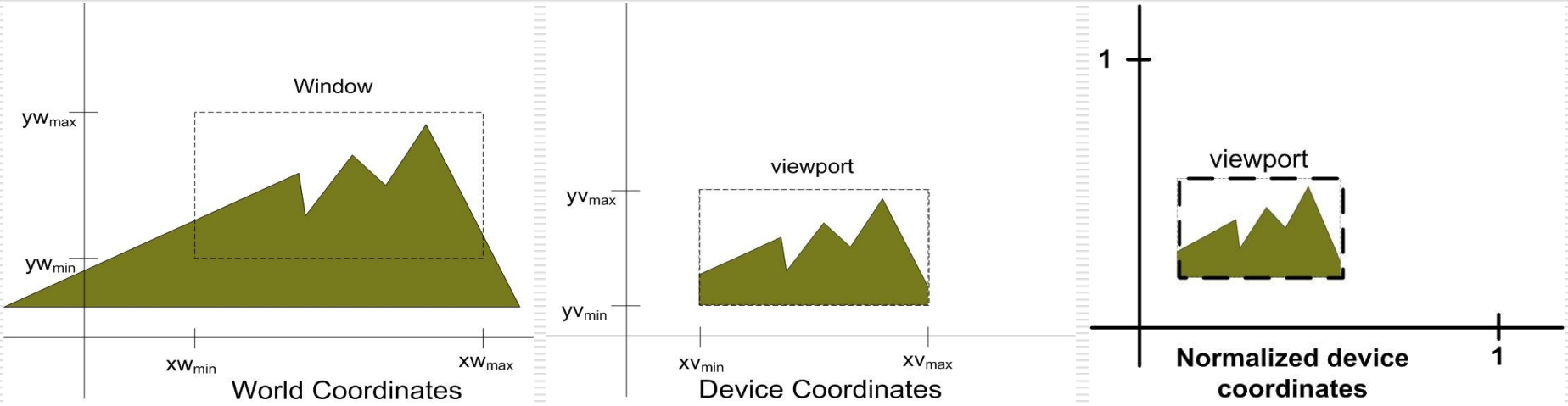
$$SH_z = \begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Computer Graphics (L10)
EG678EX

2-D Viewing

Viewing Transformation

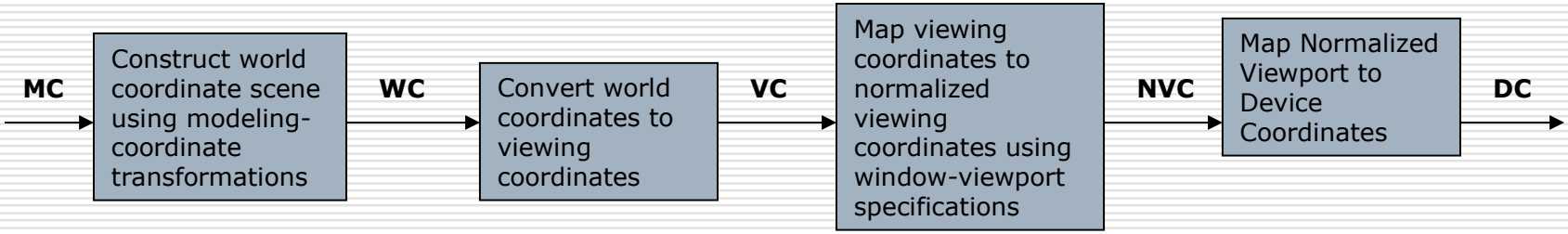
- Transformation from world to viewport



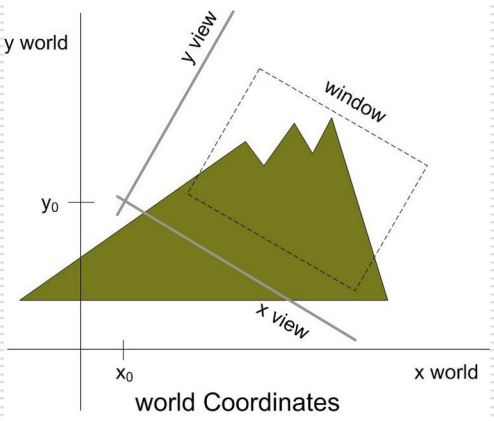
- For Fixed sized viewport, **zooming in** effect is attained if window size is decreased and **zooming out** effect if window size is increased.
- Panning effect is attained by successively changing the position of viewing window
- The normalized device coordinates maps the world co-ordinate to the viewport of maximum size = unit square as shown in figure. The advantage is the mapping is display device independent

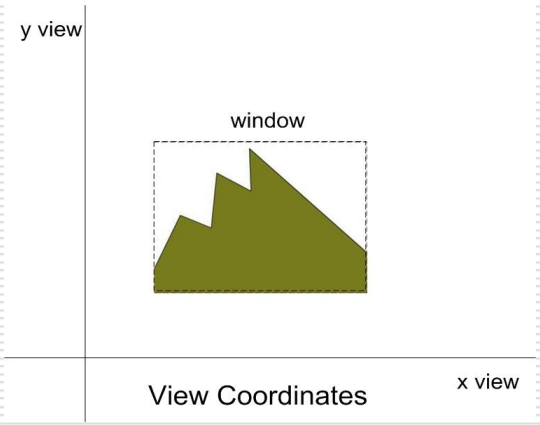
2-D Viewing pipeline

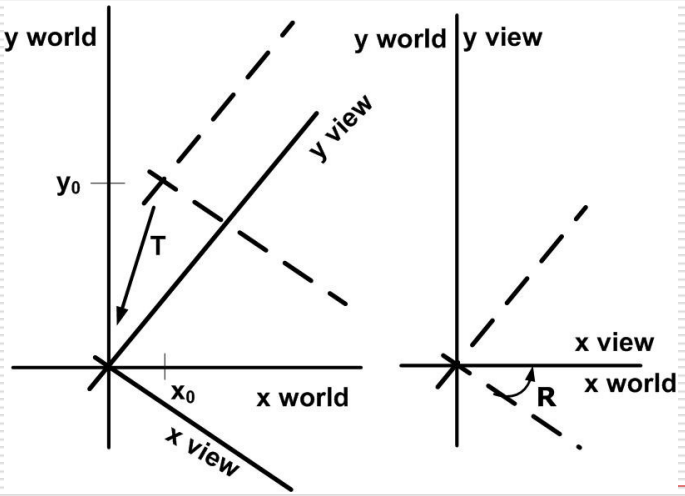
- Procedures for displaying views of a two-dimensional picture on an output device:
  - Specify which parts of the object to display (clipping window, or world window, or viewing window)
  - Where on the screen to display these parts (viewport).
- Clipping window is the selected section of a scene that is displayed on a display window.
- Viewport is the window where the object is viewed on the output device.



Viewing Reference Frame







Setting up viewing Reference Frame

The matrix is obtained in two steps:

1. Translate Viewing reference frame origin to world origin
2. Rotate Viewing reference frame to coincide with world coordinate axes

Thus we get the matrix as

$$\mathbf{M_{wc,vc} = R.T}$$

Windows To Viewport Co-ordinate Transformation

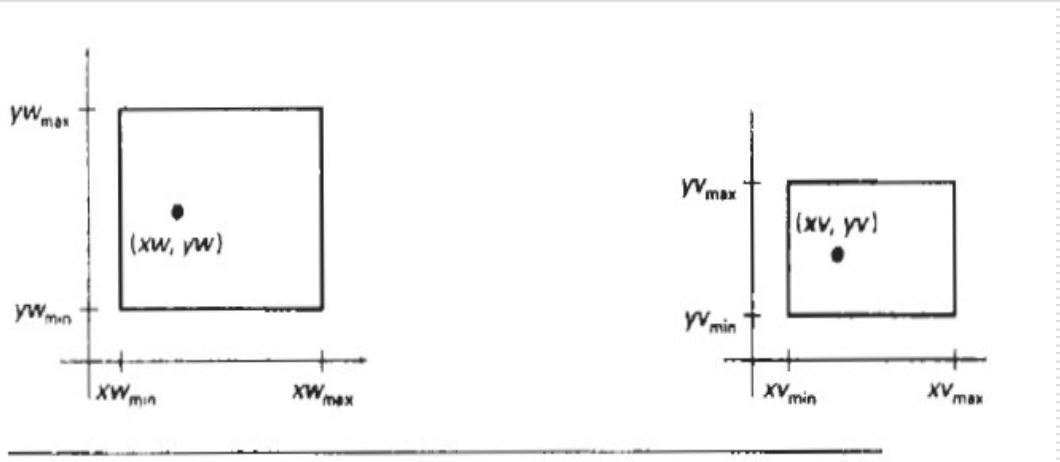
$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}}$$

Solving these we get

$$xv = xv_{min} + (xw - xw_{min})sx$$

$$yv = yv_{min} + (yw - yw_{min})sy$$



Where scaling factors are

$$sx = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$$

$$sy = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$

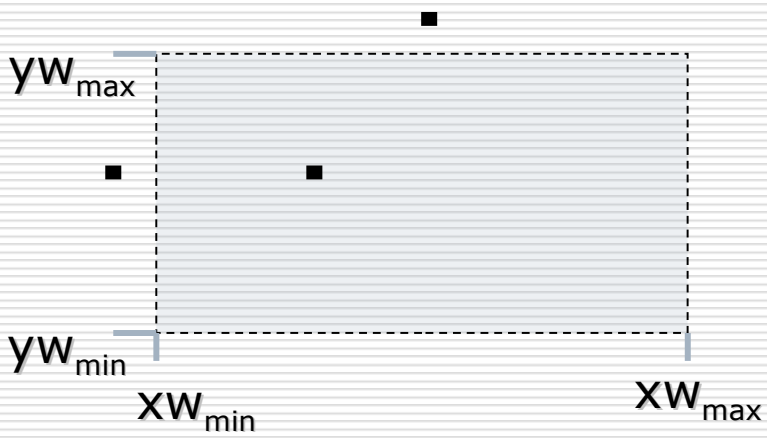
If sx = sy, the proportion is maintained otherwise the scene is stretched

Point Clipping

$$xw_{min} \leq x \leq xw_{max}$$

$$yw_{min} \leq y \leq yw_{max}$$

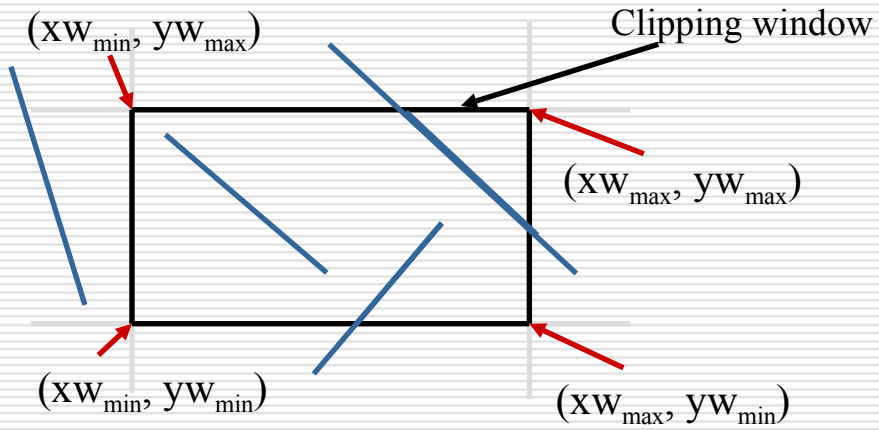
If all the four inequalities are satisfied for a point with co-ordinate (x,y), the point is accepted; i.e not clipped



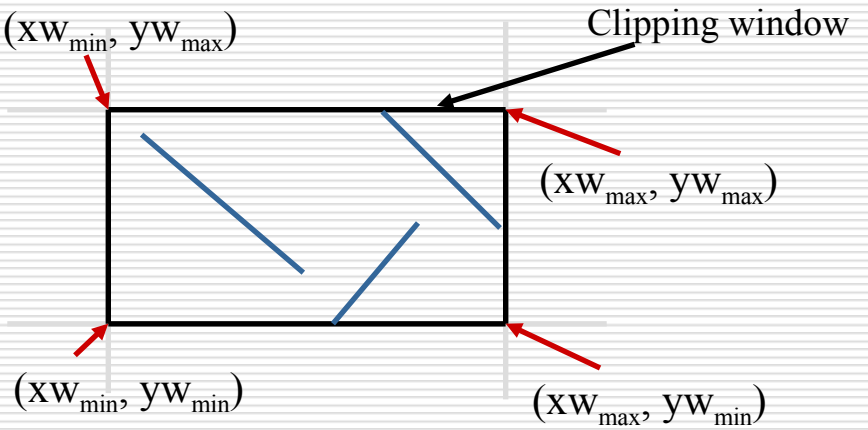
Line Clipping

[Note: most of the slides about line clipping are from some internet resource]

Line Clipping



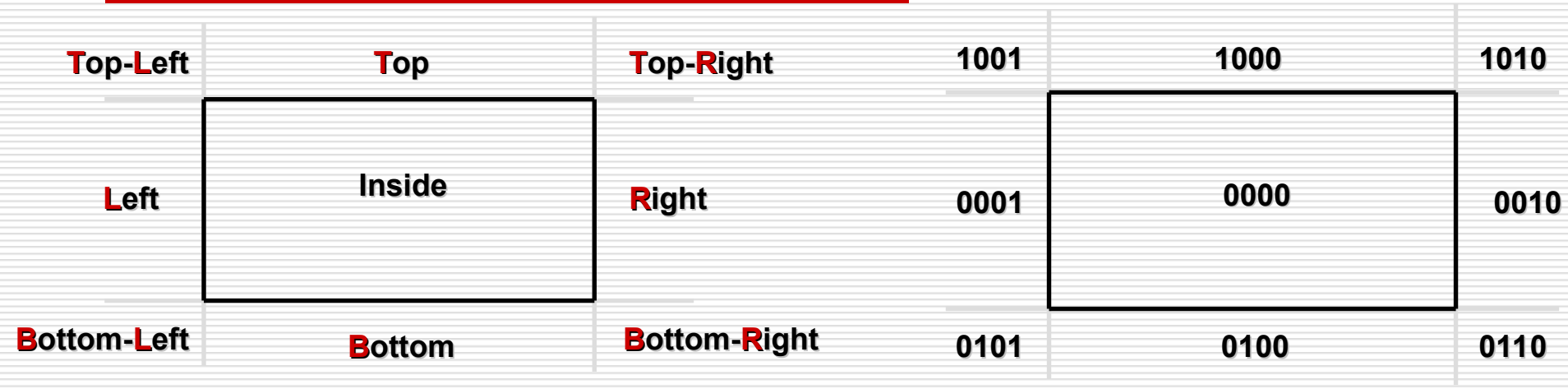
Before clipping



After clipping

What are the methods (algorithms) to perform clipping operations ?

1. Cohen-Sutherland line clipping



LRBT

Region codes	T	B	R	L
Bit position	1	2	3	4

Basic Idea

- label the Left, Right, Bottom and Top of clipping rectangle with region codes
- Test for Trivial Acceptance and Rejection (How?)
- If not trivially accepted or rejected successively clip out the portion of line outside the clip boundary and test whether it is trivially accepted or rejected

Cohen-Sutherland line clipping

Region coding

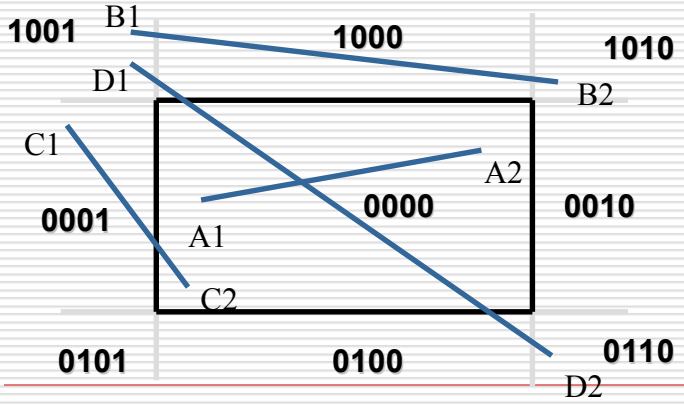
How would you decide which region an endpoint is in?

e.g

if ($x<xw_{min}$) && ($y>y_{max}$) → the point is at the **Top-Left**

Are there cases we can **trivially accept or reject**?

How would you test for those?



- **A1=0000,A2=0000**
  both (0000) → accept trivially

- **B1=1001,B2=1010**
  both NOT (0000)
  AND Operation
  B1 → 1001
  B2 → 1010
  Result 1000

  Result = NOT (0000) → reject trivially

Cohen-Sutherland line clipping

Algorithm Steps:

1. Assign a region code for each endpoints.
2. If both endpoints have a region code 0000 ---→ trivially accept these line.
3. Else, perform the logical AND operation for both region codes.
   - 3.1 **if** the result is **NOT** 0000 → trivially reject the line.
   - 3.2 **else** (i.e. result = 0000, need clipping)
     - 3.2.1. Choose an endpoint of the line that is outside the window.
     - 3.2.2. Find the intersection point at the window boundary (base on region code).
     - 3.2.3. Replace endpoint with the intersection point and update the region code.
     - 3.2.4. Repeat step 2 until we find a clipped line either trivially accepted or trivially rejected.
4. Repeat step 1 for other lines.

Cohen-Sutherland line clipping

Intersection calculations:

Intersection with vertical boundary

$y = y_1 + m(x-x_1)$

Where

$x = xw_{min}$ or xw_{max}

Intersection with horizontal boundary

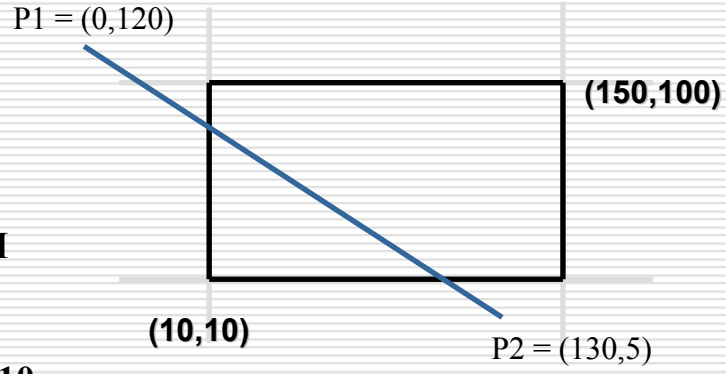
$x = x_1 + (y-y_1)/m$

Where

$y = yw_{min}$ or yw_{max}

Cohen-Sutherland line clipping

- Example:



1. P1=1001, P2=0100
2. (both 0000) – No
3. AND Operation
P1 → 1001
P2 → 0100
Result 0000
3.1 (not 0000) – no

3.2 (0000) yes
3.2.1 choose P2
3.2.2 intersection with BOTTOM boundary
m = (5-120)/(130-0) = -0.8846
x = x1 + (y –y1)/m where y = 10;
x = 130 + (10-5)/ -0.8846 = 124.35 = 124
P2' = (124, 10)
3.2.3 update region code P2' = 0000
3.2. 4 repeat step 2

2. Liang-Barsky Line Clipping

- Based on parametric equation of a line:

  $x = x_1 + u.\triangle x$

  $y = y_1 + u.\triangle y \qquad 0 \le u \le 1$

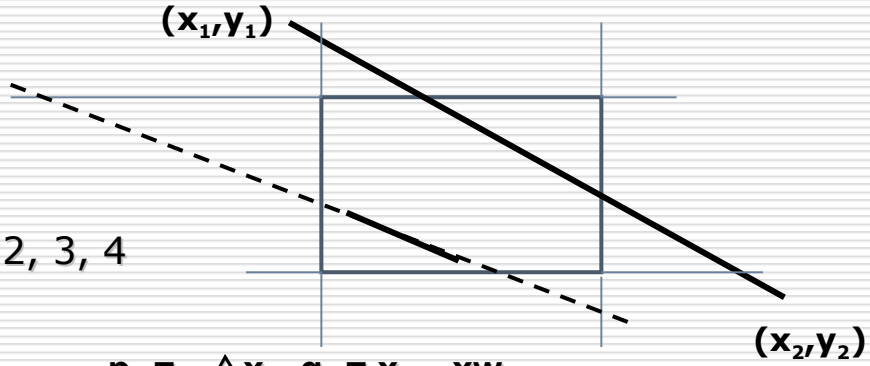
- Similarly, by adopting expressions for point clipping, the clipping window is represented by:

  $xw_{min} \le x_1 + u.\triangle x \le xw_{max}$

  $yw_{min} \le y_1 + u.\triangle y \le yw_{max}$

  … or,

  $u.\ p_k \le q_k \qquad k = 1, 2, 3, 4$



- where:

k = 1 (is the line inside left boundary ?) $p_1 = -\triangle x$, $q_1 = x_1 - xw_{min}$

k = 2 (is the line inside right boundary ?) $p_2 = \triangle x$, $q_2 = xw_{max} - x_1$

k = 3 (is the line inside bottom boundary ?) $p_3 = -\triangle y$, $q_3 = y_1 - yw_{min}$

k = 4 (is the line inside top boundary ?) $p_4 = \triangle y$, $q_4 = yw_{max} - y_1$

$P_k<0$ infinite extension of the line proceeds from outside to inside the infinitely extended boundary

$P_k>0$ infinite extension of the line proceeds from inside to outside of the infinitely extended boundary

Liang-Barsky Line Clipping

- Trivial rejection
  - Reject line with $p_k = 0$ for some k and one $q_k < 0$ for these k.
- For line with $p_k = 0$ for some k and all $q_k \geq 0$ for these k
  - Line is parallel to one of clip boundary
  - Some portion of line is inside
- For intersection with boundaries the parameters are supposed to be r_k given by

$$r_k = q_k / p_k$$

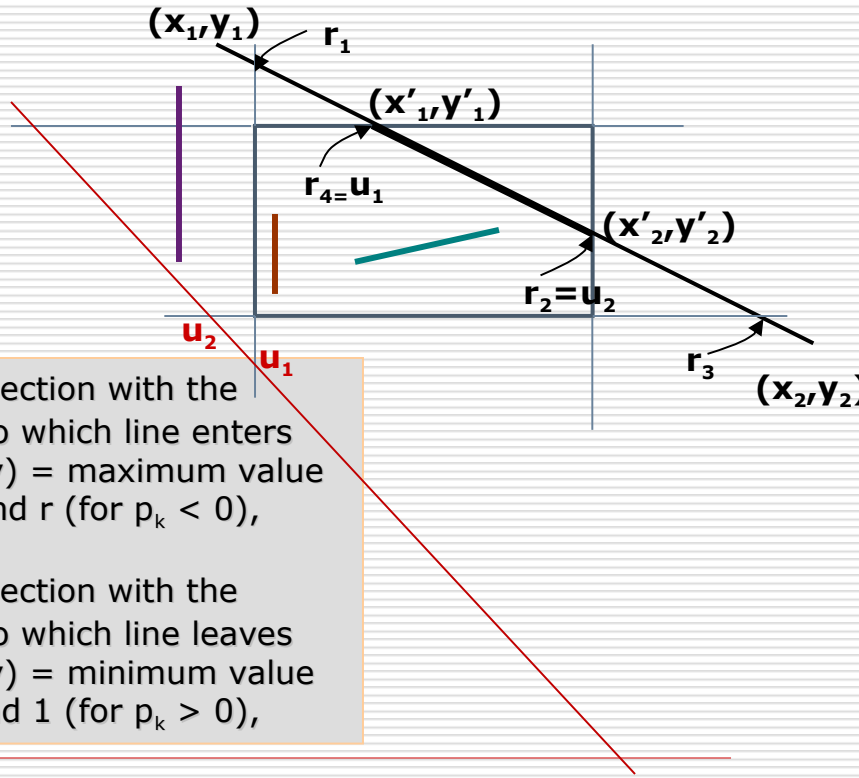
- Clipped line will be:

$x_1' = x_1 + u_1 . \triangle x; \quad u_1 \geq 0$

$y_1' = y_1 + u_1 . \triangle y;$

$x_2' = x_1 + u_2 . \triangle x; \quad u_2 \leq 1$

$y_2' = y_1 + u_2 . \triangle y;$



u_1 (For intersection with the boundaries to which line enters the boundary) = maximum value between 0 and r (for $p_k < 0$),

u_2 (For intersection with the boundaries to which line leaves the boundary) = minimum value between r and 1 (for $p_k > 0$),

Liang-Barsky Algorithm Steps

1. If $p_k = 0$ for some k then the line is parallel to a clipping boundary. Now test q_k :

   if one $q_k < 0$ for these k then line is outside

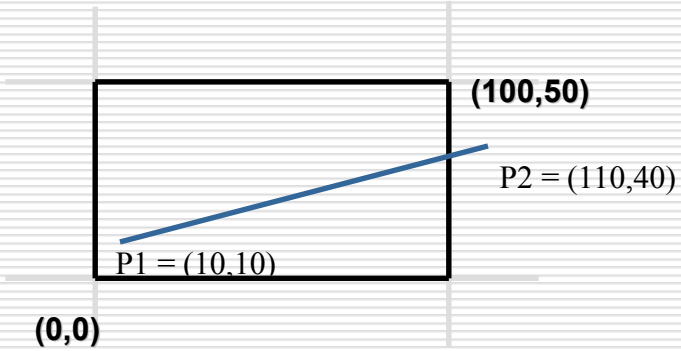
   if all $q_k \geq 0$ for these k then some portion of line is inside

2. For all $p_k < 0$ (i.e. line proceeds from outside to inside the boundary) calculate $u = \max(0, \{r_k : r_k = q_k / p_k\})$ to determine intersection point with the possibly extended clipping boundary k and obtain a new starting point for the line at u_1.
3. For all $p_k > 0$ (i.e. line proceeds from inside to outside the boundary) calculate $u_2 = \min(1, \{r_k : r_k = q_k/p_k\})$ to determine intersection point with extended clipping boundary k and obtain a new end point at u_2.
4. If $u_1 > u_2$ then discard the line
5. The line is now between $[u_1, u_2]$

Liang-Barsky Algorithm Steps

- Example:

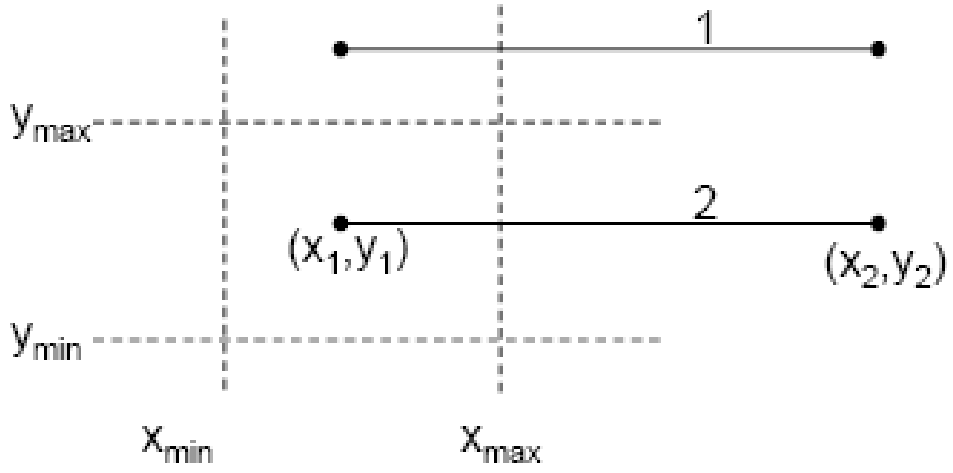
k	p_k	q_k	r_k
1	$-\triangle x$ $= -(110-10)$ $= -100$ i.e $p_k<0$	$x_1 - xw_{min}$ $= 10-0 = 10$	$r_1=10/(-100)$ $=-1/10$ **u_1**
2	$\triangle x$ $=110-10=100$ i.e $p_k>0$	$xw_{max} - x_1$ $= 100 - 10 = 90$	$r_2=90/100$ $=9/10$ **u_2**
3	$-\triangle y$ $= -(40-10)$ $=-30$ i.e $p_k<0$	$y_1 - yw_{min}$ $= 10-0 = 10$	$r_3=10/(-30)$ $=-1/3$ **u_1**
4	$\triangle y$ $= 40-10=30$ i.e $p_k>0$	$yw_{max} - y_1$ $= 50 - 10 = 40$	$r_4=40/30$ $=4/3$ **u_2**



We take
$u_1 = 0$
And
$u_2 = 0.9$

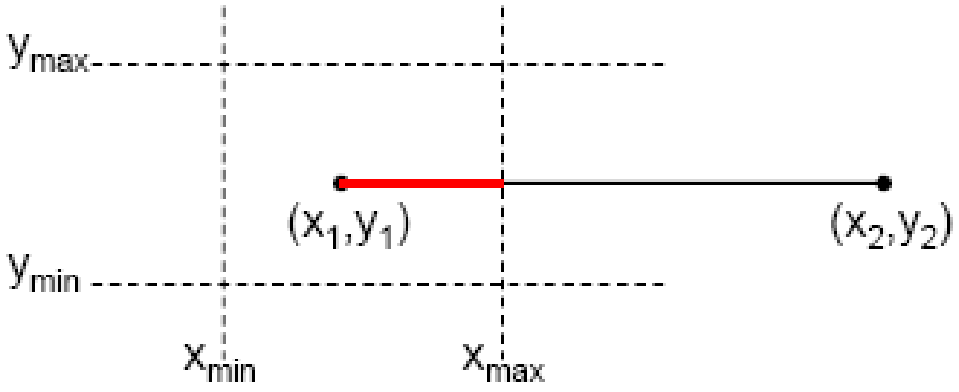
Line Clipping: Liang-Barsky

- Example 2: Consider horizontal lines with $\Delta y = 0$, $p_3 = p_4 = 0$.
- For line 1, $q_4 < 0$ and will be discarded.
- For line 2, $p_1 < 0$, $p_2 > 0$, $q_3 > 0$ and $q_4 > 0$. We proceed to calculate u_1 and u_2.



Line Clipping: Liang-Barsky

- $p_1 < 0$, note that $q_1 > 0$ hence $q_1/p_1 < 0$, and $u_1 = \max\{0, q_1/p_1\} = 0$
- $p_2 > 0$, calculate q_2/p_2 and $u_2 = \min(q_2/p_2, 1) = q_2/p_2$.
- $u_1 < u_2$ and the line is between $[u_1, u_2]$



Computer Graphics (L11)
EG678EX

3-D Viewing

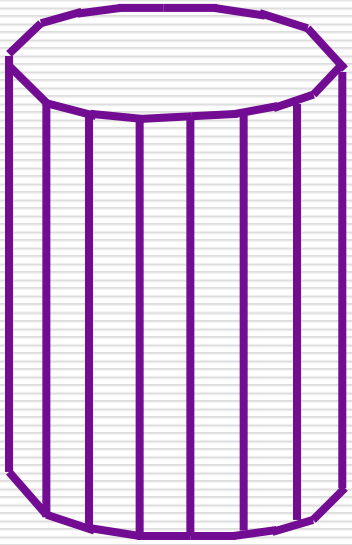
Some Background Concepts

3-D Object Representation

- Many kinds of objects in graphics scene. E.g. flowers, trees, clouds, rocks, bricks, marble, steel etc
- All characteristics can't be described with single way of representation of objects
- Two broad categories to represent **Euclidean** (geometric) solid objects:
  - Boundary Representation (B-reps)→ object is assumed as set of surfaces that separate object from surroundings. E.g. surface polygons
  - Space-Partitioning Representations→ represented by set of non overlapping contiguous solids. E.g. Octree representation (with cubes)
- **Natural objects** (Non-Euclidean i.e. non-geometrical shape) → Represented by **Fractal Geometry**.

3-D Object Representation

- Polygon Surfaces
  - Approximate representation 3-D objects by set of surface polygons
    - Advantage:- speeds up rendering; since all surfaces are represented with linear equations
  - More accurate representation for polyhedron
  - Curved surface approximation can be improved with increase in number of polygons
- Polygon Tables:- To specify polygon surfaces
  - Geometric Tables
    - Consist of parameters that specify polygon vertices and spatial orientation of polygons
  - Attribute Tables
    - Consist of parameters that specify transparency, surface texture, color etc.



Polygon Surface Approximation of Cylinder

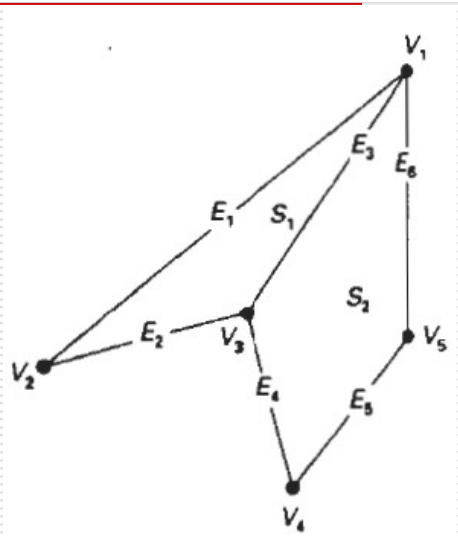
3-D Object Representation

- Geometric Tables
  - Specified by three lists (each successive tables consists of pointers back to previous table
    - Vertex Table
    - Edge Table
    - Polygon Surface Table
  - Only Edge Tables or (Vertex table + Polygon Table) is sufficient ?
    - Yes but redundancy (repeated) in drawing
  - Alternate representation:- we add extra information for quick identification of redundancy. E.g. as in figure (Alternate Representation), pointers to polygon table are added to edge table so as to notice common edges quickly.

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

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

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



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

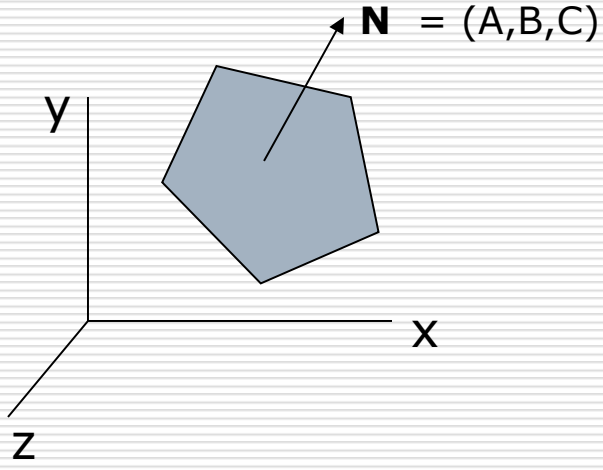
Alternate Representation

3-D Object Representation

- Spatial Orientation of Surface
  - Orientation of surface normal
  - How to find surface normal if surface vertices (x_1,y_1,z_1), (x_2,y_2,z_2) and (x_3,y_3,z_3) are given ?
    - Plane equation: $Ax + By + Cz + D = 0$
    - Put the vertex coordinates to get

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

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



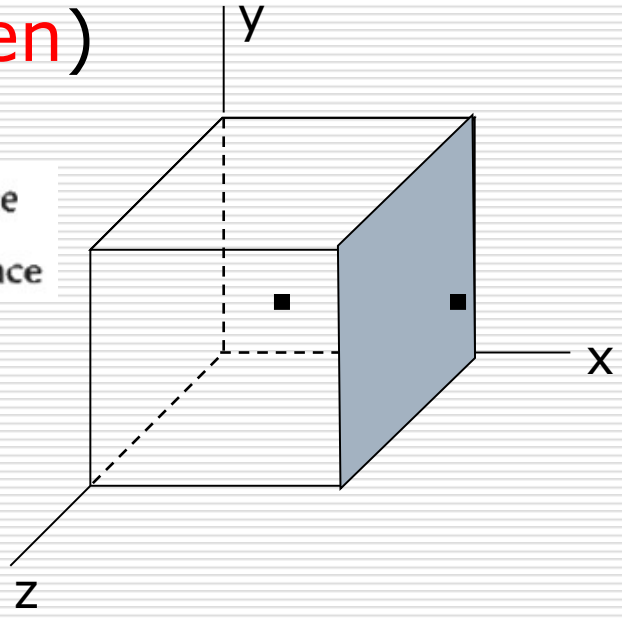
    - One of the surface normals is: **N** = (A,B,C)
      (Note: Remember plane equation of form lx + my +nz = p
      i.e. **N** = (l,m,n))

3-D Object Representation

- Two sides of a surface
  - For right handed cartesian system (i.e if vertices are taken in C-Clockwise order to evaluate A,B,C and D then)

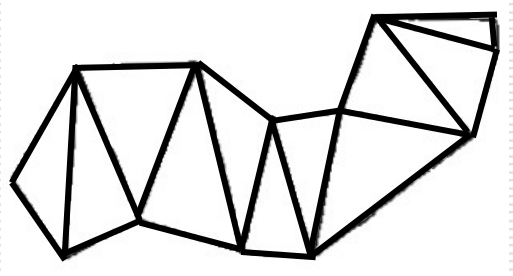
if $Ax + By + Cz + D < 0$, the point (x, y, z) is inside the surface

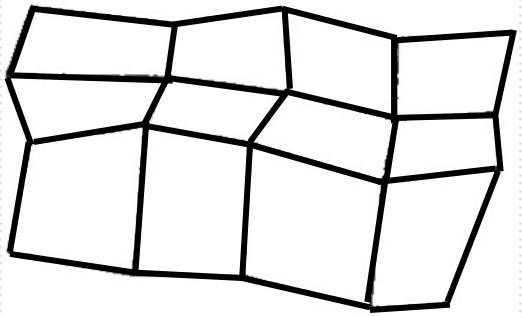
if $Ax + By + Cz + D > 0$, the point (x, y, z) is outside the surface



3-D Object Representation

- Polygon Meshes:- object represented as:
  - Triangle Strip
    - Given n vertices co-ordinates, n-2 trangles are tiled to produce strip
  - Quadrilateral Mesh
    - Given n by m array of vertices co-ordinates, (n-1) by (m-1) quadrilaterals produce mesh

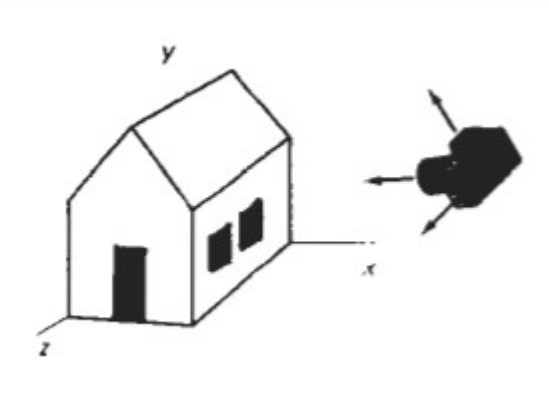


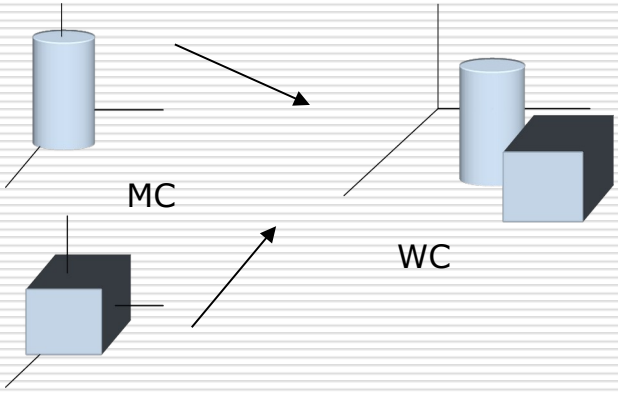


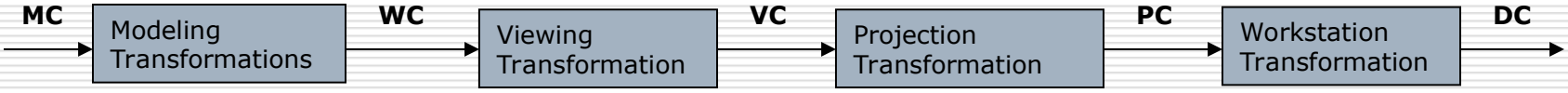
3-D Viewing

3-D Viewing Pipeline

- Analogous to 2-D scene generation
- 3-D scene generation process analogous to taking photographs with camera
  - More flexible method than camera analogy in graphics to generate 3-D scene



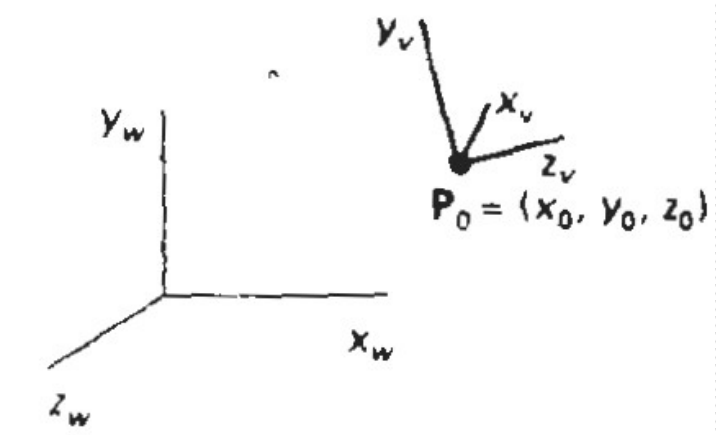




Viewing Coordinates

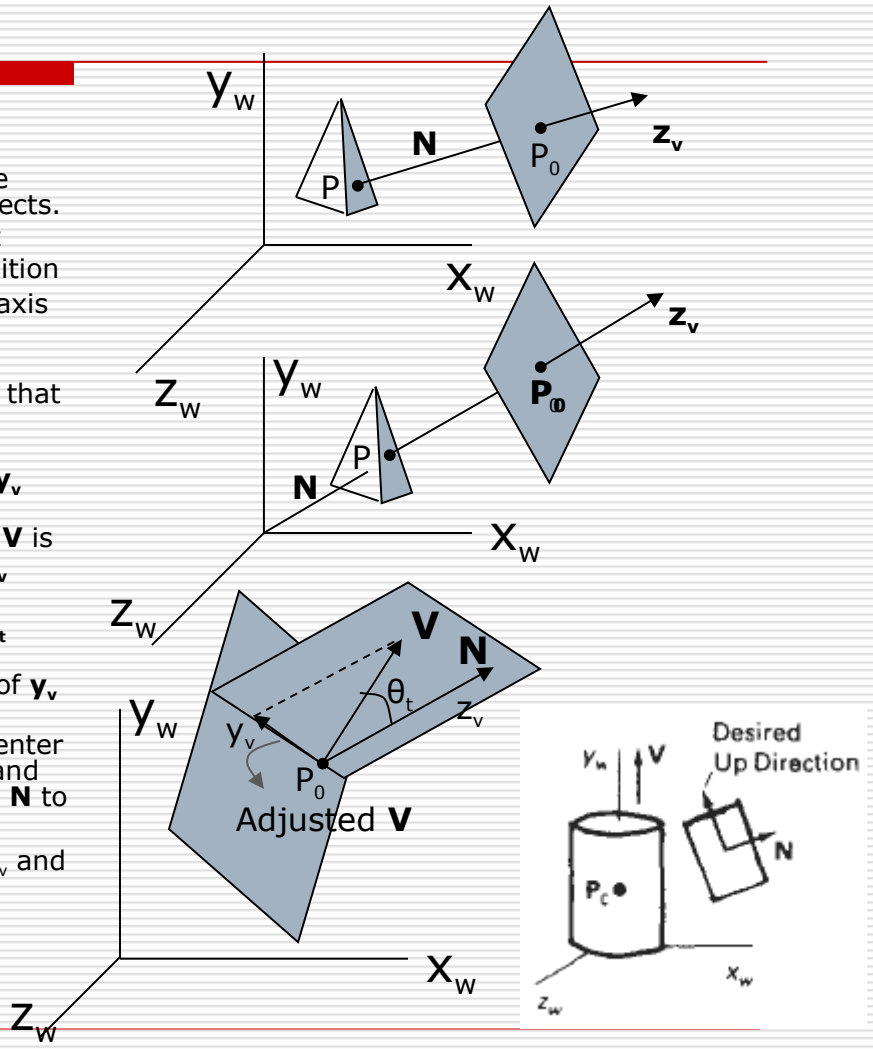
- Specifying Viewing Plane similar to camera orientation
- Steps:
  - Setup viewing Coordinate Reference → how?
  - Setup a view plane (projection plane) perpendicular to a viewing z_v axis
    - Note: plane could be assumed to be similar to camera film
  - Transform world co-ordinate scene to viewing coordinate scene

Viewing co-ordinates are projected unto the view plane



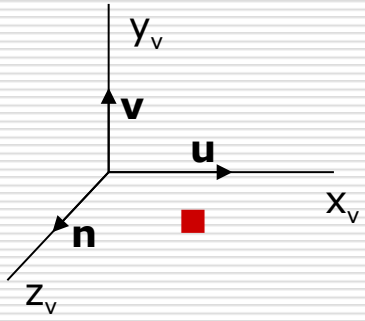
Setting Viewing Coordinates Reference

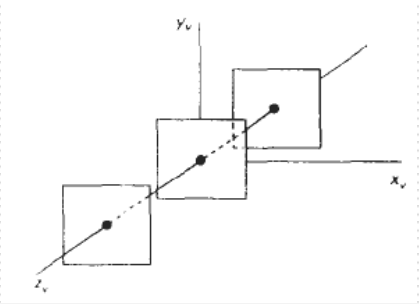
- Pick a point in world coordinate as **view reference point** (origin of viewing coordinate system)
  - view reference point often chosen closed to or on the surface of some object or at center of a group of objects.
  - If it is near to object→ we aim camera to that object
  - If it is some distance away from scene→ camera position
- Specify view plane orientation with Normal vector **N** i.e. z_v axis direction. **How?**
  - Chose a point in world coordinate
  - Vector from world origin to that point or vector from that point to view reference point is direction of Normal Vector **N**
- Specify *view-up vector* **V** to establish positive direction for $\mathbf{y_v}$ axis
  - Specifying **V** that is perpendicular to **N** is difficult so **V** is specified in any convenient direction and establish $\mathbf{y_v}$ direction by projecting **N** to View plane
  - In some packages **V** is specified as the twist angle $\boldsymbol{\theta_t}$ about the $\mathbf{z_v}$ axis and adjusted by projecting to view plane. The projection specifies the desired direction of $\mathbf{y_v}$ axis.
  - In some packages view reference point P_0 is at the center of the object, **V** is specified by world vector (0,1,0) and this vector is projected to the plane perpendicular to **N** to establish y_v axis as shown in figure
- Establish third vector **U** perpendicular to both the vectors y_v and z_v to specify vector x_v

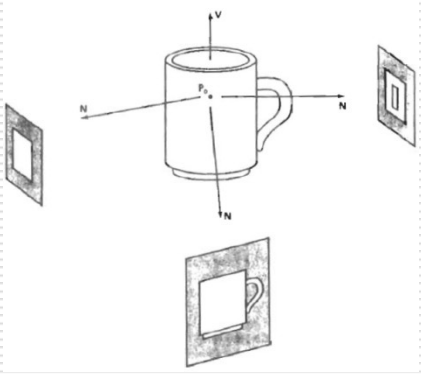


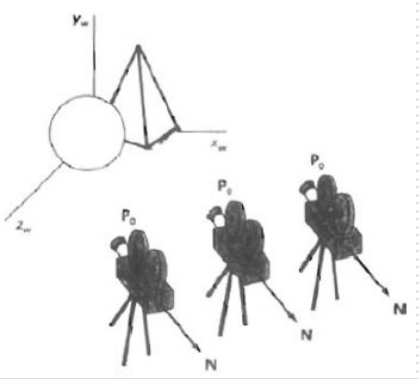
Setting Viewing Coordinates Reference (contd…)

- Normalized viewing reference axis
  - Specified by unit vectors u, v and n (uvn system) along viewing axes
- View plane position
  - Specified by view-plane-distance from viewing origin
  - View plane is always parallel to $x_v y_v$ plane
  - Series of view is obtained by changing direction of **N** and keeping view reference point fixed. (How to obtain view along the line of **v**?)
  - To simulate camera motion through a scene keep the direction of N fixed and change the view reference point



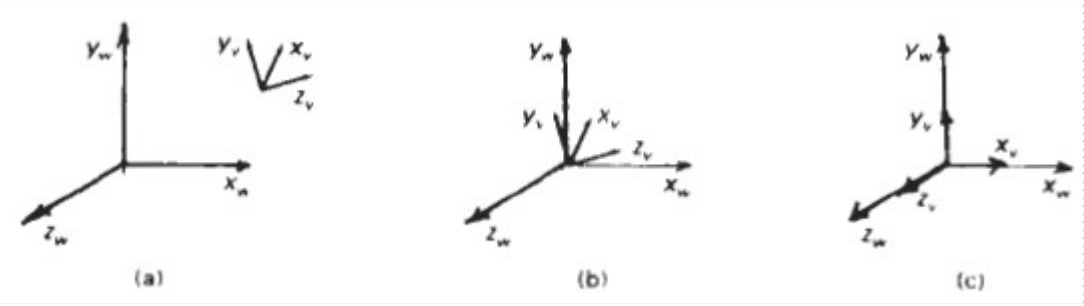






Transformation From World To Viewing Coordinates

1. Translate the view reference point to world origin
2. Rotate the view reference axis x_v, y_v, z_v to align with world axes x_w, y_w, z_w respectively



$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{n} = \frac{\mathbf{N}}{|\mathbf{N}|} = (n_1, n_2, n_3)$$

$$\mathbf{u} = \frac{\mathbf{V} \times \mathbf{N}}{|\mathbf{V} \times \mathbf{N}|} = (u_1, u_2, u_3)$$

$$\mathbf{v} = \mathbf{n} \times \mathbf{u} = (v_1, v_2, v_3)$$

$$\mathbf{R} = \begin{bmatrix} u_1 & u_2 & u_3 & 0 \\ v_1 & v_2 & v_3 & 0 \\ n_1 & n_2 & n_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M}_{WC,VC} = \mathbf{R} \cdot \mathbf{T}$$

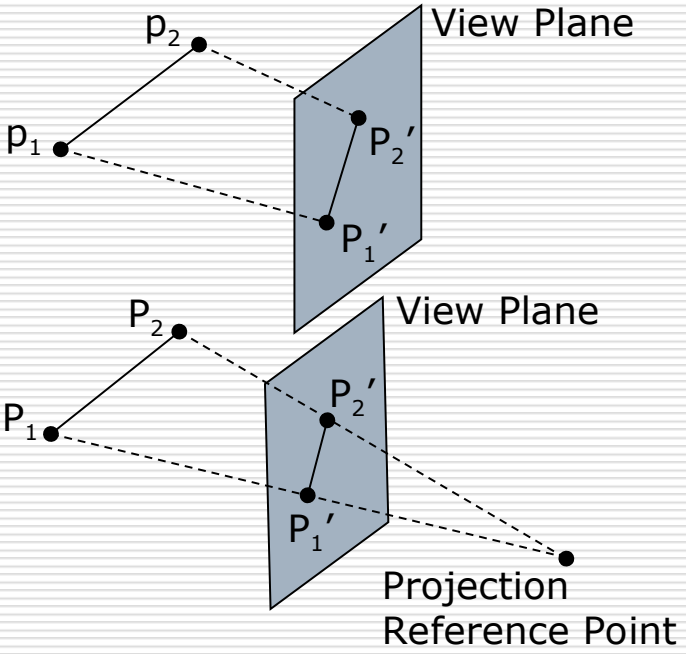
$$\mathbf{R}. \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

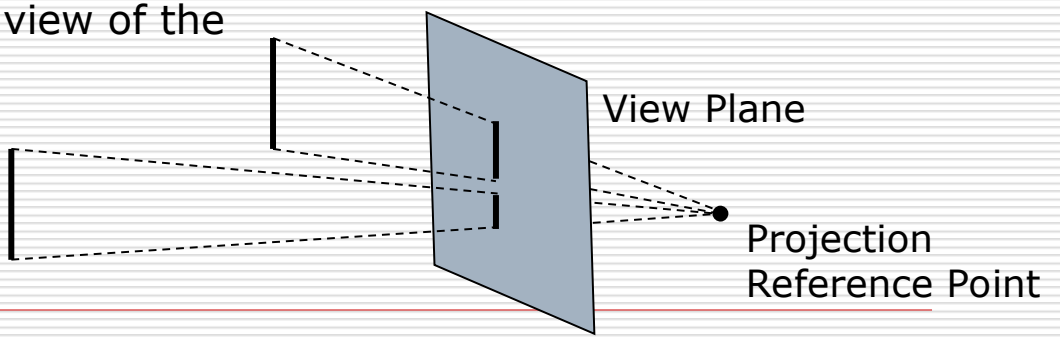
Thus the rotation matrix aligns vector **u** to x-axis
Similarly it alligns vectors **v** and **n** to y and z axis of world

Thus the rotation matrix is the desired matrix to align view reference frame to world coordinate frame

Projections

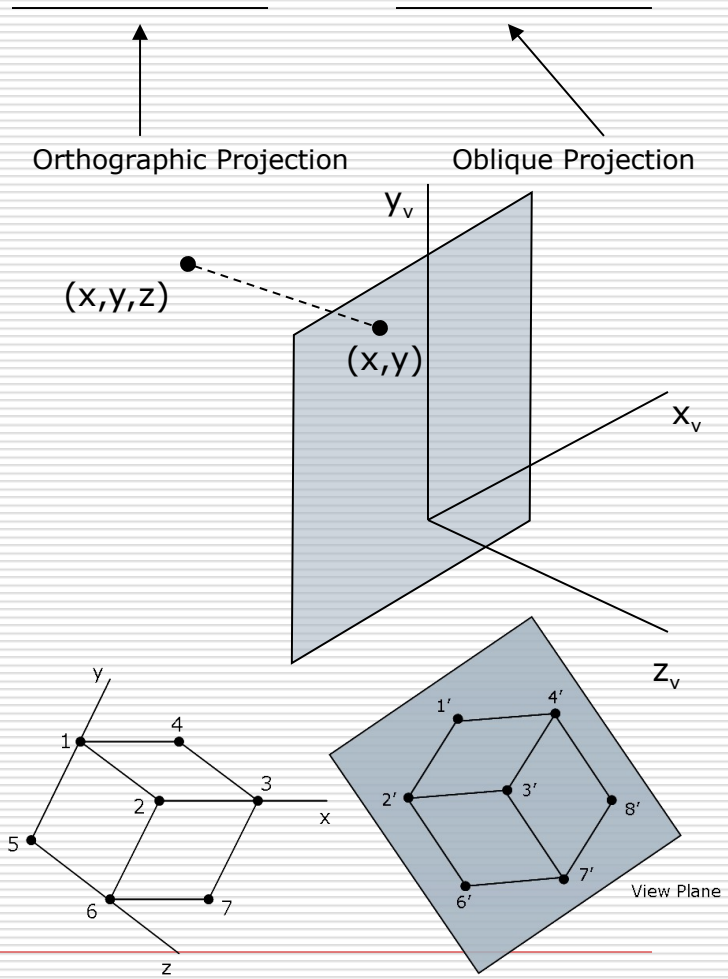
- Converts 3-D viewing co-ordinates to 2-D projection co-ordinates
  - Two types of projection
    1. Parallel Projection
       - Coordinate positions are transformed to view plane along parallel lines (projection lines)
       - **Orthographic** and **Oblique** Projection
    2. Perspective Projection
       - Coordinate positions are transformed to view plane along lines (projection lines) that converges to a point called projection reference point (center of projection)
       - Equal sized object appears in different size according as distance from view plane
  - Intersection of projection lines and view plane gives the projected view of the object





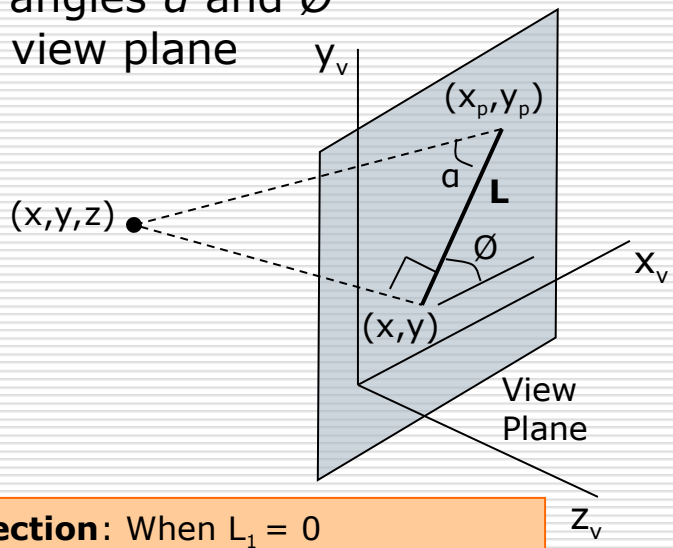
Parallel Projection

- **Orthographic projection**:
  - Projection lines are perpendicular to view plane
  - Used to produce Front, Side and Top view of an object
  - Axonometric Orthographic Projection:
    - To display more than one face
    - **Isometric Projection** is commonly used orthographic projection
      - Generated by aligning the projection plane so that it intersect each coordinate axis in which object is defined at same distance from origin
- **Oblique projection**: Projection lines are not perpendicular to view plane



Oblique Projection

- Oblique projection vectors are specified by two angles *a* and *Ø*
- (x,y) are also the orthographic co-ordinates on view plane



Projection Matrix Calculation

$x_p = x + L.cosØ$
$y_p = y + L.sinØ$

Coordinates (x,y,z) are viewing coordinates

$tana = z/L$
$L = z/tana = z.L_1$ where, $L_1 = 1/tana$

$x_p = x + z(L_1 cosØ)$
$y_p = y + z(L_1 sinØ)$

- Final projection matrix will be

$$M_{parallel} = \begin{bmatrix} 1 & 0 & L_1\cos\phi & 0 \\ 0 & 1 & L_1\sin\phi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Orthographic projection: When $L_1 = 0$
$x_p = x$
$y_p = y$

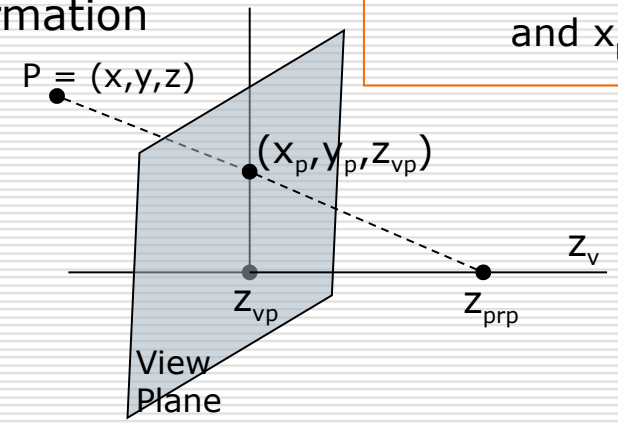
Cavalier projection: when tan*a = 1 (i.e a = 45⁰)*
Ø = 30⁰ or 45⁰
Cabinet Projection: when tan*a = 2 (i.e a = 63.4⁰)*
Ø = 30⁰ or 45⁰

Perspective Projection

- Projection vectors meet at projection reference point z_{prp} along the z_v axis
  - $x' = x - xu$
  - $y' = y - yu$
  - $z' = z - (z - z_{prp})u$
- On view plane
  - $z' = z_{vp}$; therefore
  - $u = (z_{vp} - z)/(z_{prp} - z)$
- thus projection transformation equations are

$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) = x\left(\frac{d_p}{z_{prp} - z}\right)$$

$$y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) = y\left(\frac{d_p}{z_{prp} - z}\right)$$



Projection Equation in homogeneous coordinates

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -z_{vp}/d_p & z_{vp}(z_{prp}/d_p) \\ 0 & 0 & -1/d_p & z_{prp}/d_p \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Where, $h = (z_{prp} - z)/d_p$

and $x_p = x_h/h$, $y_p = y_h/h$

Special cases

uv plane as view plane (i.e $z_{vp} = 0$)

Viewing coordinate origin as projection reference point (i.e $z_{prp} = 0$)

Perspective projection

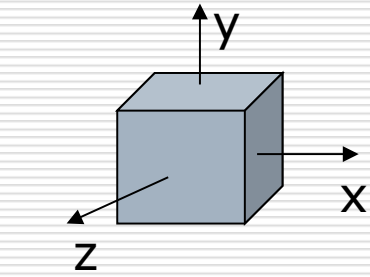
Vanishing Point: a set of parallel lines that are not parallel to view plane are projected as converging lines that appear to converge at a point called vanishing point

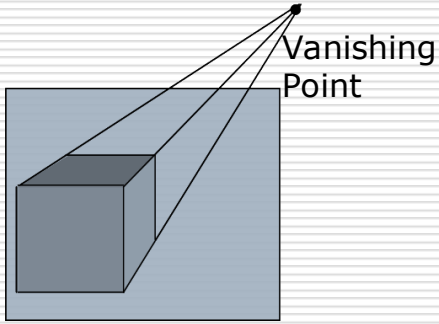
•a set of parallel lines that are parallel to view plane are projected as parallel lines

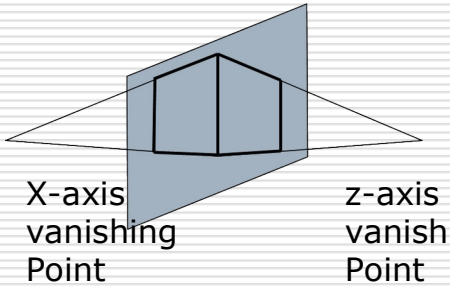
•More than one set of parallel lines form more than one vanishing points in the scene

Principal Vanishing point: Vanishing point for a set of parallel lines parallel to one of the principal axis of object

•We can control the number of principal vanishing point to one, two or three with the orientation of projection plane and classify as **one, two or three point perspective projection**





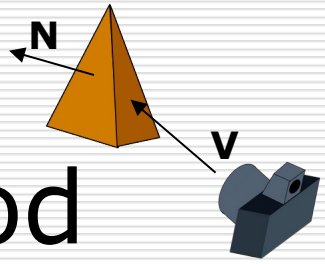


Computer Graphics (L12)
EG678EX

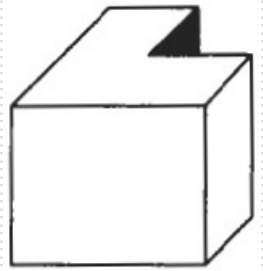
Visible Surface Detection

Visible Surface Detection

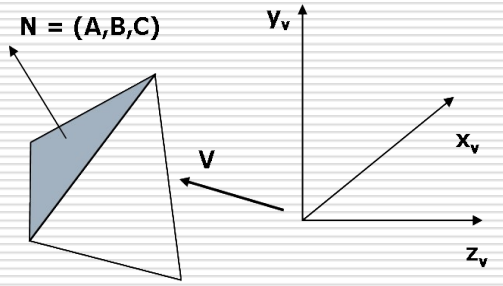
- Two approaches
  - Depends upon object definition or projected image
    - Object-space
      - Compares objects and parts of objects to each other within the scene
    - Image-space
      - Visibility is decided point by point at each pixel position on projected plane



Back Face Detection Method



- Fast and simple object-space method
- Based on "inside" or "outside" test
- Test whether view point is inside or outside the polygon surface
  - Tested by the inequality: Ax + By + Cz + D = 0
    - If view point is outside the surface → visible
    - If inside → back face
  - The test could be simplified by considering normal vector **N** = (A,B,C) to polygon surface and Vector **V** in viewing direction as:
    the polygon is back face if: **V.N**>0
- If object description is converted to viewing co-ordinates then, **V** is along z_v axis i.e **V** = $(0,0,V_z)$, then:
  V.N = V_zC
  i.e V.N > 0 → V_zC
  So we need to consider only sign of C
- For right handed viewing system with viewing direction along negative z_v axis, c<0 → back face
- Convex polyhedron?→ no problem because either completely visible or completely hidden
- Concave polyhedron?→ some problem; needs additional faces
- More than one objects? in scene → problem
- Eliminates about half of the polygon surface in a scene from fu visibility tests



What if C = 0 ??
• Grazing Viewing i.e viewing direction is perpendicular to surface normal→ also back face

So, back face if C≤0

Depth Buffer (z-Buffer) method

- Image-space method
- Surface depth is compared at each pixel position on the projection plane
- Usually applied to scene containing only polygon surfaces; but possible to apply for non-planar surfaces
- For each pixel position (x,y) on projection plane, object depths can be compared by comparing z-values as each (x,y,z) position on a polygon surface corresponds to the orthographic projection point (x,y) on projection plane
- Two buffers required
  - Depth buffer: to store depth value for each position
  - Refresh buffer: to store intensity values of each position
- Drawback: Deals only with opaque surface but not with transparent surface

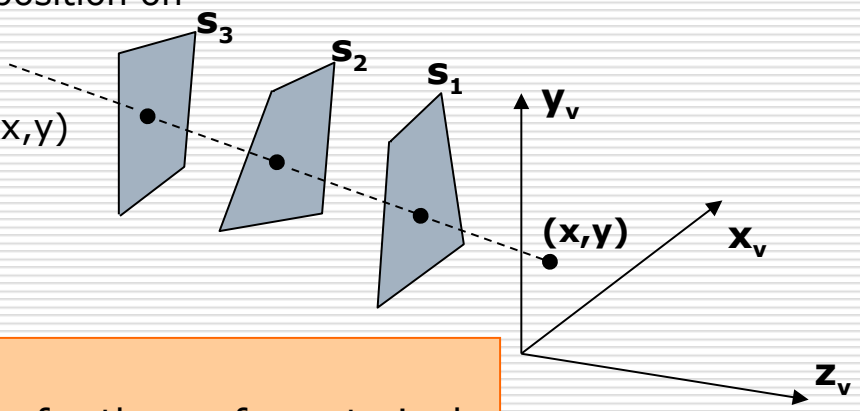
Depth Buffer (contd…)

1. Initialize the depth buffer and refresh buffer so that for all buffer positions (x,y),

   depth(x,y) = 0, refresh(x,y) = $I_{backgrnd}$

2. For each position on each polygon surface, compare depth values to previously stored values in the depth buffer to determine visibility
   - Calculate the depth z for each (x,y) position on the polygon
   - If z>depth(x,y), then set

     depth(x,y)=z, refresh(x,y)=I_{surf}(x,y)



$I_{backgrnd}$ = background intensity
I_{surf}(x,y) = projected intensity value for the surface at pixel position (x,y)

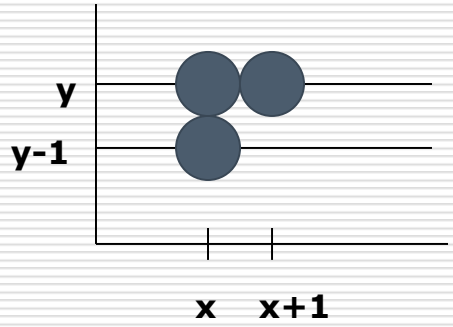
Depth Buffer (depth calculation)

- From plane equation, depth is

$$z = \frac{-Ax - By - D}{C}$$

- For the next adjacent pixel in a scan line, depth is

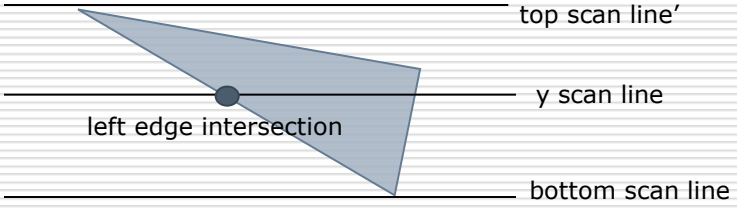
$$z' = \frac{-A(x+1) - By - D}{C} \quad \text{or} \quad z' = z - \frac{A}{C}$$



- Determine minimum and maximum y-coordinates for each polygon
- Start from top scan line to bottom scan line
- Starting from top vertex, calculate x position down the left edge of the polygon recursively as

$$x' = x - 1/m$$

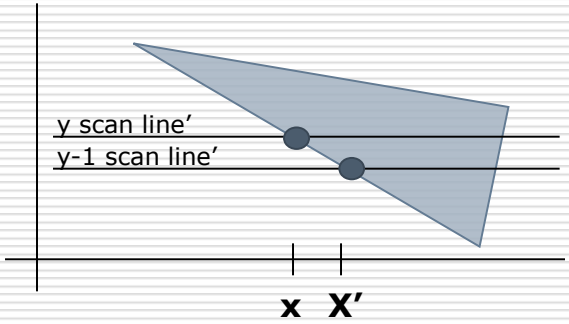
$m(x'-x)=y'-y=-1$



- Thus depth value for the next pixel in left edge is obtained as

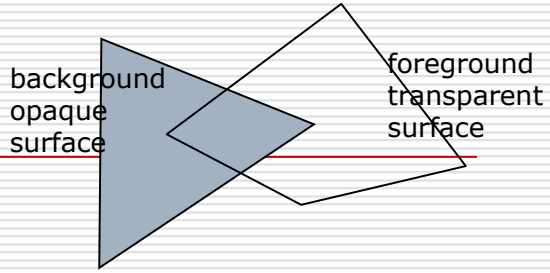
$$z' = z + \frac{\frac{A}{m} + B}{C}$$

put x' = x-1/m and y' = y-1

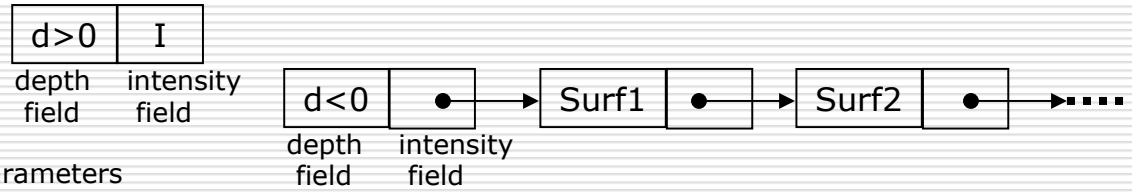


- For vertical edge m = infinity, so: $z' = z + \frac{B}{C}$

A-Buffer Method



- Extension of depth buffer method
- Antialiased, area-average, accumulation-buffer
- The depth buffer is expanded so that each position in the buffer can reference a linked list of surfaces thus enabling more than one surface intensity consideration
- The object boundary could be antialiased
- Each pixel position in the A-Buffer has two fields
  - Depth Field → stores a positive or negative real number
    - Positive→ single surface contributes to pixel intensity
    - Negative → multiple surfaces contribute to pixel intensity
  - Intensity Field → stores surface-intensity information or a pointer value
    - Surface intensity if single surface → stores the RGB components of the surface color at that point and percent of pixel coverage
    - Pointer value if multiple surfaces
  - in case of surface linked list, the data for each surface in the linked list includes
    - RGB intensity
    - Opacity parameter
    - Depth
    - Percent of area coverage
    - Surface identifier
    - Other surface-rendering parameters
    - Pointer to next surface



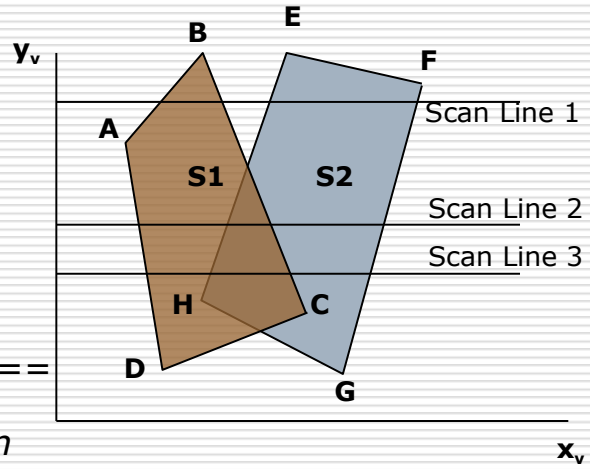
  - Scanlines are processed to determine surface overlaps of pixels across the individual scanlines.
  - Surfaces are subdivided into a polygon mesh and clipped against the pixel boundaries
  - The opacity factors and percent of surface overlaps are used to determine the pixel intensity as an average of the contribution from the overlapping surfaces

Scan-Line Method

- Extension of scanline algorithm for filling polygon interiors
- Instead of filling just one surface, we deal with multiple surfaces
- Construct edge tables and polygon tables
  - Edge table contains → endpoints of each edge in the scene, inverse slope of each line and pointers to the polygon table to identify its corresponding surface
  - Polygon table contains → plane equation coefficients, surface intensity, pointers to the edge table (optional)
- Setup an active list (why active ??) of edges for those edges which cross the current scan line
  - The edges are sorted in order of increasing x
- Define flags for each surface to indicate whether a position is inside or outside the surface
  - At leftmost boundary of surface flag == *on* and at rightmost boundary flag == *off*

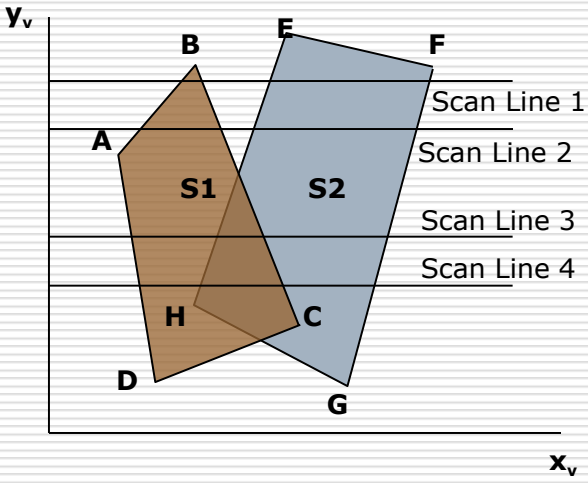
Scan-Line Method (contd…)

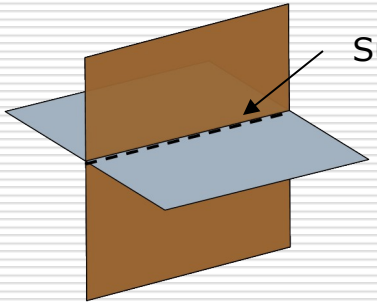
- For scan line 1
  - The active edge list contains edges AB,BC,EH, FG
  - Between edges AB and BC, only *flags for s1 == on* and between edges EH and FG, only *flags for s2==on*
    - no depth calculation needed and corresponding surface intensities are entered in refresh buffer
- For scan line 2
  - The active edge list contains edges AD,EH,BC and FG
  - Between edges AD and EH, only the *flag for surface s1 == on*
  - Between edges EH and BC *flags for both surfaces == on*
    - Depth calculation (using plane coefficients) is needed.
    - In this example depth for s1 is less than for s2, so intensities for surface s1 are loaded into the refresh buffer until boundary BC is encountered
  - Between edges BC and FG *flag for s1 == off* and *flag for s2 == on*
    - Intensities for s2 are loaded on refresh buffer
- For scan line 3
  - Same **coherent** property as scan line 2 as noticed from active list, so no depth calculation needed between edges BC and EH

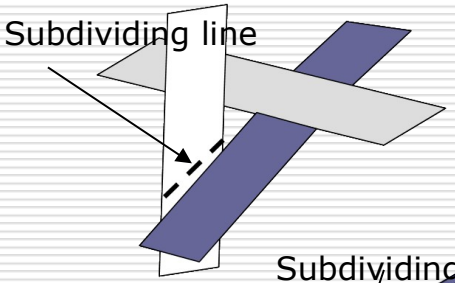


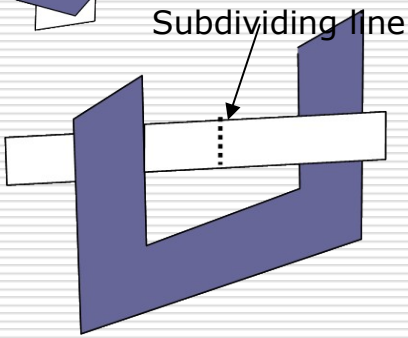
Scan-Line Method (contd…)

- Problem: Dealing with **cut through** surfaces and **cyclic overlap** is problematic when used coherent properties
  - Solution: Divide the surface to eliminate the overlap or cut through









Computer Graphics (L13)
EG678EX

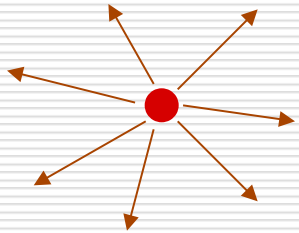
Illumination Models & Surface Rendering Methods

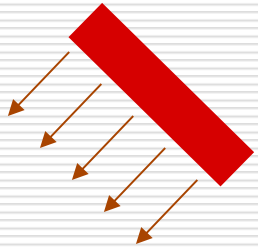
Background

- Perspective Projection + Natural Lighting Effects to Visible Surface → Realistic Effect in Scene
- Illumination model (Shading Model) is used to Calculate intensity of light that we see at a given point on the surface of an object
- Surface Rendering algorithm uses the intensity calculation from an illumination model to determine the light intensity of all pixel positions for various surfaces in the scene
- Photorealism in computer graphics involves two elements:
  - Accurate graphical representations of object
  - Good physical descriptions of lighting effect in scene
- Illumination model: derived from physical laws that describe surface light intensities

Light Sources

- Luminous objects
  - Light Emitting Sources
  - Reflecting Sources





- Point Source: Simplest model of light emitter
- Distributed Light Source: modeled as accumulated illumination effects of the points over the surface of the source

Basic Illumination Model

- Ambient Light → Background Light
  - Object not exposed directly to a light source is visible with ambient light
  - Has no spatial or directional characteristics
  - Amount of ambient light incident on each object is a constant for all surfaces and over all directions
  - Level for the ambient light in scene by parameter I_a
  - Each surface in the scene is illuminated with the constant intensity level I_a
  - The intensity of reflected light (intensity of illumination) depends upon optical properties of the surface
  - Ambient light produces flat shading → not desirable in general, so scenes are illuminated with other light source together with ambient light

Basic Illumination Model

- Diffuse Reflection
  - Reflected light intensity are constant over each surface in a scene independent of viewing direction → *ideal diffuse reflectors*
  - the diffuse-reflection coefficient (diffuse reflectivity) → k_d
    - Sets the fraction of light intensity that is reflected from each surface
    - k_d is a the function of surface color, but for our purpose we assume k_d to be constant
  - Diffuse reflection intensity when scene illuminated only with ambient light:

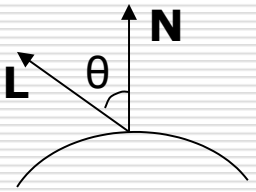
    $I_{ambdiff} = k_d I_a$

  - Lambertian reflectors
    - Follow lambart's cosine law → radiant energy from a small surface area *dA* is proportional to the cosine of angle θ between surface normal and incident light direction
      - For a point source with intensity I_l, the diffuse reflection intensity is

        $I_{l,diff} = k_d I_l \cos\theta$

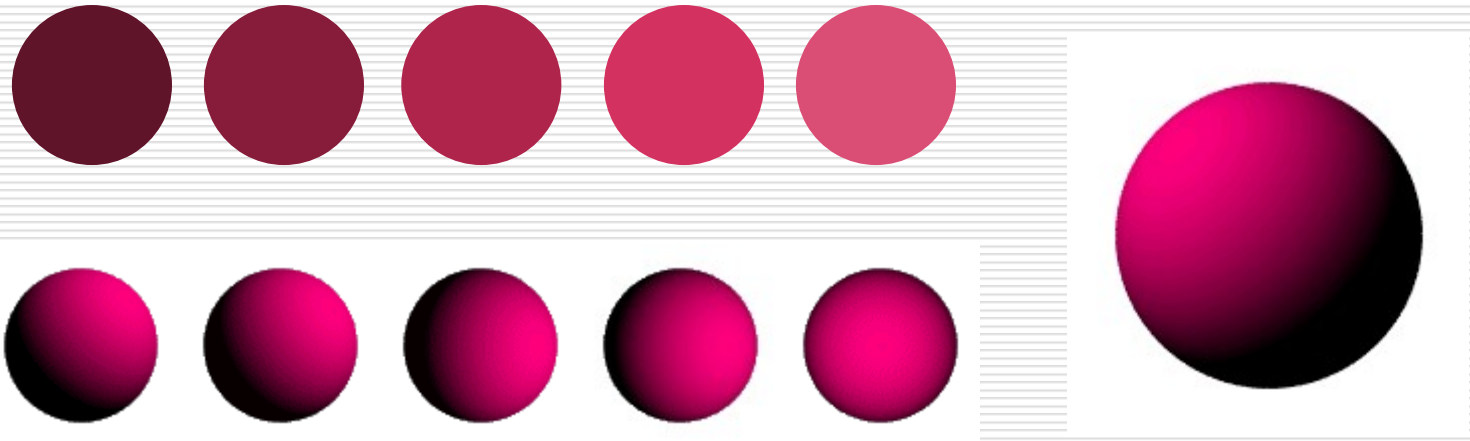
        $I_{l,diff} = k_d I_l (\mathbf{N}.\mathbf{L})$

      Where **N** and **L** are unit vectors



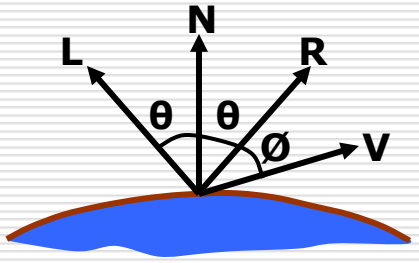
Basic Illumination Model

- Diffuse reflection for ambient light source + a point light source
  - $I_{diff} = k_a I_a + k_d I_l(\mathbf{N.L})$
- Fig:
  - sphere illuminated with different intensity ambient light
  - Illuminated with varying direction light source



Specular Reflection

- Bright spot seen at an illuminated shiny surface when viewed at certain direction
  - Polished metal surface, person's forehead, apple etc. exhibit specular reflection
- In fact an image of light source
- Result of total or near total reflection of incident light in a concentrated region around the specular reflection angle θ
- Fig:
  - **L** → unit vector pointing to light source
  - **N** → unit surface normal vector
  - **R** → unit vector in direction of specular reflection
  - **V** → unit vector pointing viewer
- Ideal reflector exhibit specelur reflection in the direction of R only (i.e Ø=0) but for non-ideal case specular reflection is seen over finite range of viewing positions



Phong Model

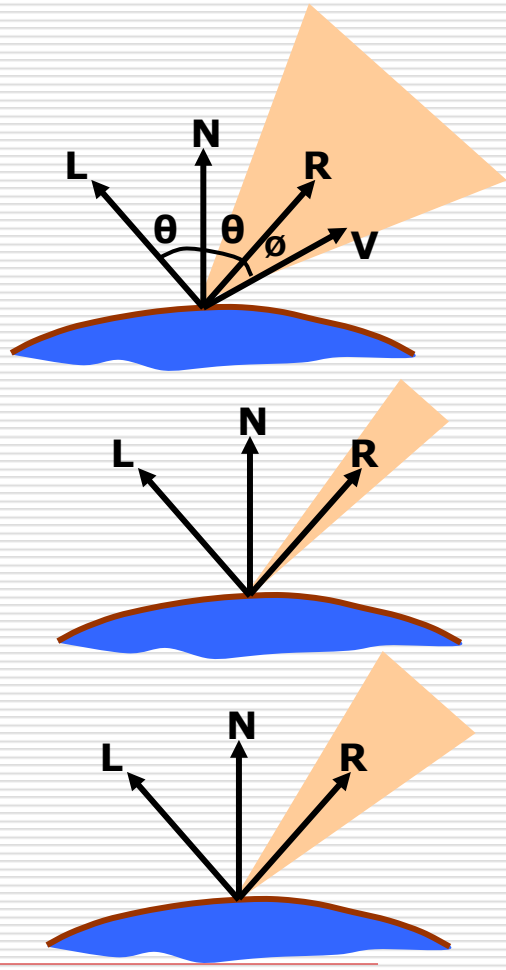
- Intensity of specular reflection: proportional to

$$\cos^{n_s}\phi$$

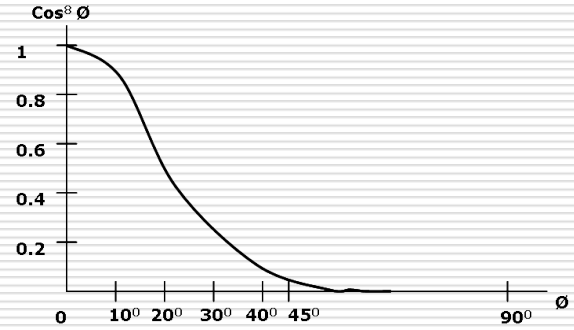
  - n_s → **specular reflection parameter** (depends on surface)
  - Ø ranges from 0 to 90⁰ (i.e *cos* Ø varies from 0 to 1)
- Intensity of specular reflection depends on:
  - Material properties of surface
  - Angle of incidence *θ*
  - Other factors such as polarization and color of the incident light
- Monochromatic specular intensity variations can be approximated using **specular-reflection coefficient**, *w(θ)* for each surface

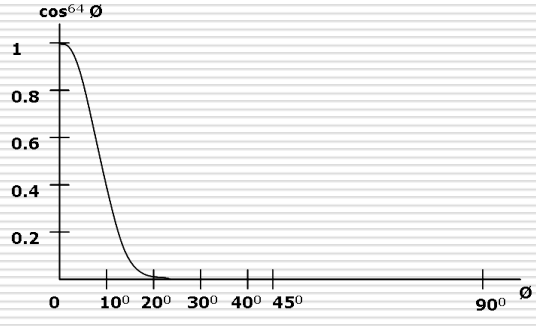
$$I_{spec} = w(\theta) I_l \cos^{n_s}\phi$$

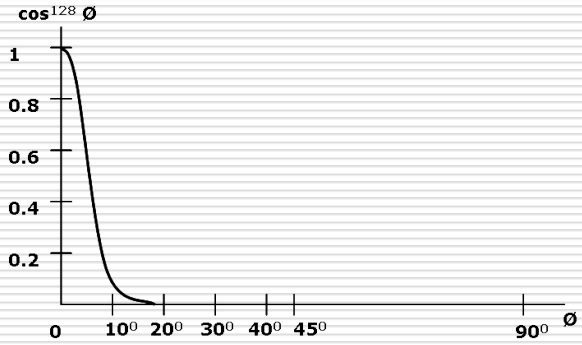
- At *θ = 90⁰, w(θ)* = 1 → all incident light is reflected

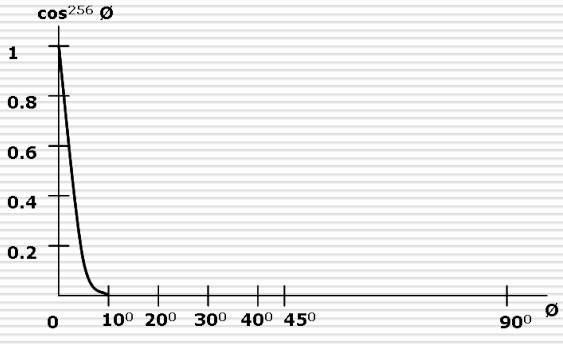


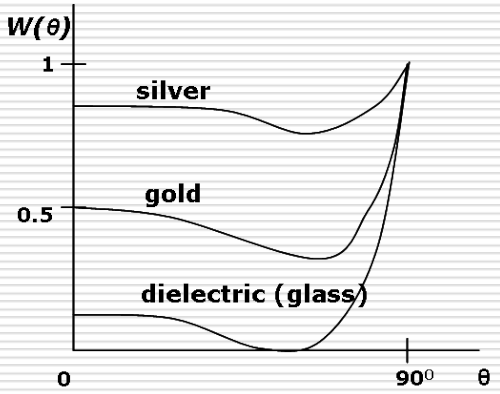
Plot For $\cos^{n_s}\phi$ and Specular Reflection Coefficient For Various Materials











Phong Model (contd…)

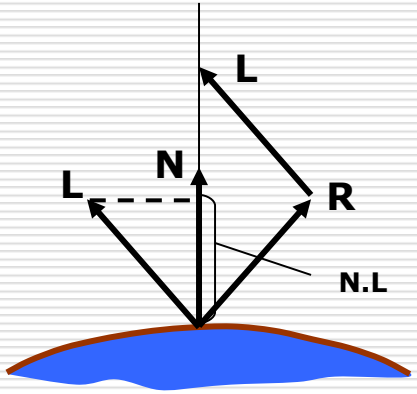
- Simplified form: assume $w(\theta) = k_s = constant$

$$I_{spec} = k_s I_l (V.R)^{n_s}$$

- Vector **R** can be evaluated from vectors **L** an **N** as:

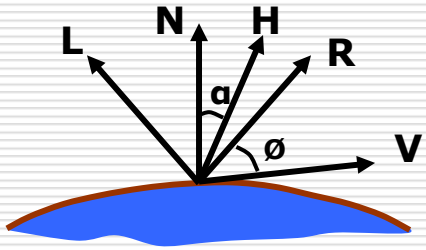
  R + **L** = (2**N**.**L**)**N**

  i.e **R** = (2**N**.**L**)**N** – **L**



- Further simplified by replacing **V**.**R** with **N**.**H** where **H** is halfway vector between **L** and **V** (i.e **H** is unit bisector vector of angle between **L** and **V**)

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



- Thus $I_{spec} = k_s I_l (N.H)^{n_s}$
- If we add ambient light and diffuse reflection component then total intensity is given as:

$$I = I_{diff} + I_{spec}$$
$$= k_a I_a + k_d I_l (N.L) + k_s I_l (N.H)^{n_s}$$

- For multiple light sources (n light sources)

$$I = k_a I_a + \sum_{i=1}^{n} I_{li} \left[k_d (N.L_i) + k_s (N.H_i)^{n_s} \right]$$

When v is coplanar with L and R
α = Ø/2 otherwise α > Ø/2

Warn Model

- Provides method for simulating studio lighting effects
- Light intensities are controlled in different direction
- Light sources are modeled as points on a reflecting surface using the Phong model for the surface points (i.e calculate intensities considering specular and diffuse reflection parameters). Then the intensity in different directions is controlled by selecting values for Phong exponent.
- Light controls used by studio photographers can be simulated

Intensity Attenuation

- The intensity of radiant energy at a point *d* distance far from source is attenuated by $1/d^2$
- Considering intensity attenuation produces realistic lighting effect
  - E.g: if two parallel surfaces with same optical parameters overlap, they would be displayed as one surface
- Using merely $1/d^2$ as attenuation factor for our simple single point light source model, too much intensity variation is produced when *d* is small and a little variation when *d* is large
- Graphical packages have compensated the problem by using inverse linear quadratic function of d for intensity attenuation as:

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

  - a_0 can be adjusted to prevent f(d) from becoming too large when d is very small
  - Magnitude of attenuation function is limited to 1 as

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

- The Phong illumination model considering attenuation is:

$$I = k_a I_a + \sum_{i=1}^{n} f(d_i) I_{li} \left[k_d (N.L_i) + k_s (N.H_i)^{n_s} \right]$$

Color Consideration in Phong Illumination model

- For RGB description, each color in a scene is expressed in terms of R,G and B components
- Various methods:
  - Described by considering the RGB components for e.g. (k_{dR}, k_{dG}, k_{dB}) of diffuse reflection coefficient vector
    - E.g: For blue light ($k_{dR}=k_{dG}=0$)

$$I_B = k_{aB} I_{aB} + \sum_{i=1}^{n} f(d_i) I_{lBi} \left[k_{dB} (N.L_i) + k_{sB} (N.H_i)^{n_s} \right]$$

  - Described by specifying components of diffuse and specular color vectors for each surface and retaining the reflectivity (k) as a single valued constants

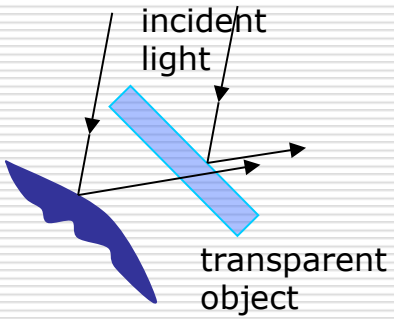
$$I_B = k_a S_{dB} I_{aB} + \sum_{i=1}^{n} f(d_i) I_{lBi} \left[k_d S_{dB} (N.L_i) + k_s S_{sB} (N.H_i)^{n_s} \right]$$

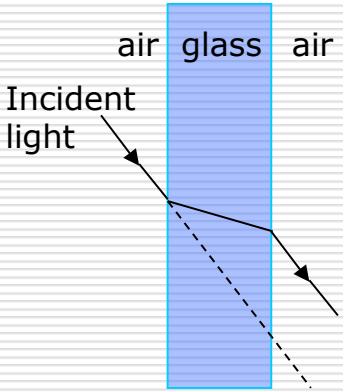
  - Described by specifying wavelength for a color specification. This specification is useful to specify color as more than three components

$$I_B = k_a S_{d\lambda} I_{a\lambda} + \sum_{i=1}^{n} f(d_i) I_{l\lambda i} \left[k_d S_{d\lambda} (N.L_i) + k_s S_{s\lambda} (N.H_i)^{n_s} \right]$$

Transparency

- Transparent surface produces both reflected and transmitted light
- Light intensity depends on relative transparency and position of light source or illuminated object behind or in front of the transparent surface
- To model transparent surface, intensity contribution of light from various sources (illuminated objects) that are transmitted from the surface must be considered in the intensity equation
- Both diffuse and specular reflection take place on transparent surface
- Diffuse effects are important for partially transparent surfaces such as frosted glass
- Diffuse refraction can be generated by decreasing the intensity of the refracted light and spreading intensity contributions at each point on the refracting surface onto a finite area to obtain blurred image of background surface





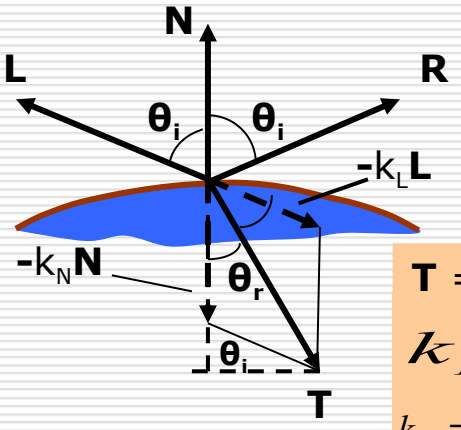
Transparency (contd…)

- The Snell's law is used to calculate the refracted ray direction

$$\sin\theta_r = \frac{\eta_i}{\eta_r}\sin\theta_i$$

- The unit vector along the transmitted light path is given by

$$T = \left(\frac{\eta_i}{\eta_r}\sin\theta_i - \cos\theta_r\right)N - \frac{\eta_i}{\eta_r}L$$



$\mathbf{T} = -k_L\mathbf{L} - k_N\mathbf{N}$

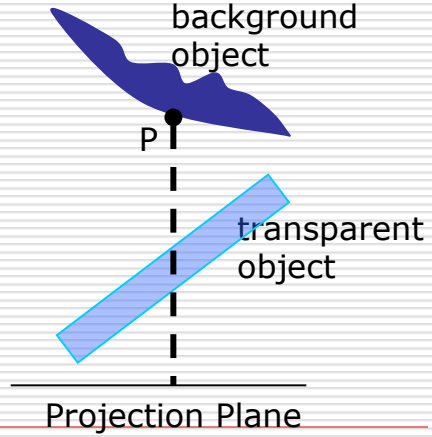
$k_L = \frac{\eta_i}{\eta_r}$

$k_N = \cos\theta_r - k_L\cos\theta_i$

- Transmitted intensity I_{trans} through a transparent surface from a background object and Reflected intensity I_{refl} from the transparent surface with ***transparency coefficient*** k_t is given by

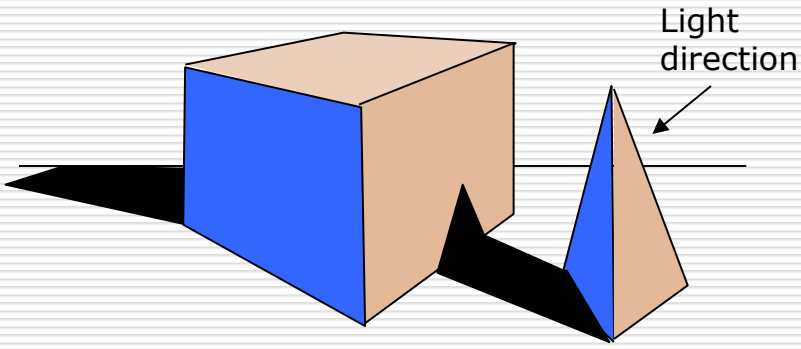
$$I = (1-k_t)I_{refl} + k_tI_{trans}$$

where $(1-k_t)$ is ***opacity fac***



Shadow

- Hidden surface method with light source at the view position can be used
- The shadow area for all light sources are determined and these shadows could be treated as a surface pattern arrays



Displaying Light Intensities

- The intensity calculated by illumination model must be converted to the allowable intensity of a particular graphics system
- Graphics systems are bilevel (i.e two levels; on and off) and others are capable of displaying several levels
  - For the case of bilevel system, intensities are converted into halftone patterns
- The difference between intensities 0.20 and 0.22 should be perceived same as that of 0.40 and 0.88
- The intensity level in a monitor should be spaced so that the ratio of successive intensities is constant

  for n+1 successive intensity levels

  $I_1/I_0=I_2/I_1=\ldots\ldots..I_n/I_{n-1} = r$

  $I_k = r^k I_0$

  also $I_n = 1 \rightarrow r = (1/I_0)^{1/n}$

  so, $I_k = I_0^{(n-k)/n}$

- The lowest intensity value I_0 depends on the characteristics of the monitor (I_0 ranges from 0.005 to around 0.025)
  - the black level displayed on monitor will have some intensity
- The highest intensity value is 1
- For color system (blue color for example)

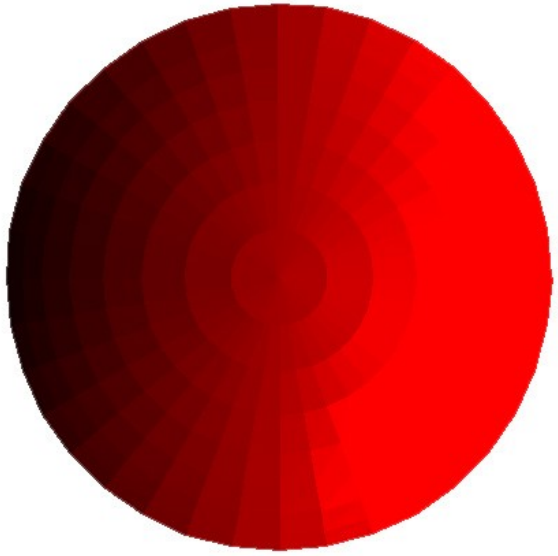
  $I_{Bk} = r_B^k I_{B0}$

Polygon Rendering Methods

- Illumination model is applied to fill the interior of polygons
- Curved surfaces are approximated with polygon meshes
  - But polyhedra that are not curved surfaces are also modeled with polygon meshes
- Two ways of polygon surface rendering
  - Single intensity for all points in a polygon
  - Interpolation of intensities for each point in a polygon
- Methods:
  - Constant Intensity Shading
  - Gouraud Shading
  - Phong Shading

Constant Intensity Shading

- Flat shading
  - Each polygon shaded with single intensity calculated for the polygon
- Useful for displaying general appearance of a curved surface
- Accurate rendering conditions:
  - Object is a polyhedron and not an curved surface approximation
  - All light sources should be sufficiently far from the surface (i.e **N**.**L** and attenuation function are constant over the polygon surfaces) :constant diffuse reflection??
  - Viewing position is sufficiently far (i.e **V**.**R** is constant over the surface) : Specular Reflection ??
  - ***Note****: Approximate rendering is possible even the conditions are not satisfied*
- Drawback: intensity discontinuity at the edges of polygons



Credit goes to the students

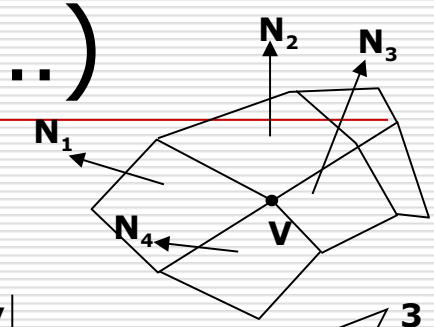
Gouraud Shading

- Calculation Steps:
  - Determine the average unit normal vector at each polygon vertex
  - Calculate each of the vertex intensities by applying an illumination model
  - Linearly interpolate the vertex intensities over the polygon surface
- Intensity discontinuity at the edges of polygons is eliminated
- Drawback:
  - **Mach bands**: bright and dark intensity streaks caused by linear interpolation of intensities
    - Could be reduced by dividing the surface into large number of polygons or by using other methods, such as *Phong shading*

Gouraud Shading (contd…)

- Average Unit Normal: Obtained by averaging the surface normals of all polygons sharing the vertex

$$N_v = \frac{\sum_{k=1}^{n} N_k}{\left|\sum_{k=1}^{n} N_k\right|}$$

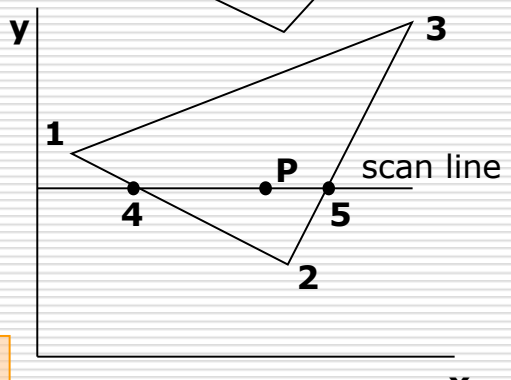


- Intensity interpolation:
  - Along the polygon edges are obtained by interpolating intensities at the edge ends

$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_4}{y_1 - y_2} I_2$$

Recursive calculation along the edge

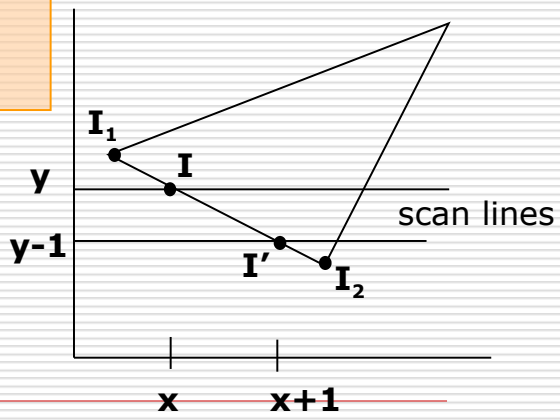
$$I' = I + \frac{I_2 - I_1}{y_1 - y_2}$$



  - Along the scan line between the polygon edges are obtained by interpolating intensities at the intersection of scan line and polygon edges

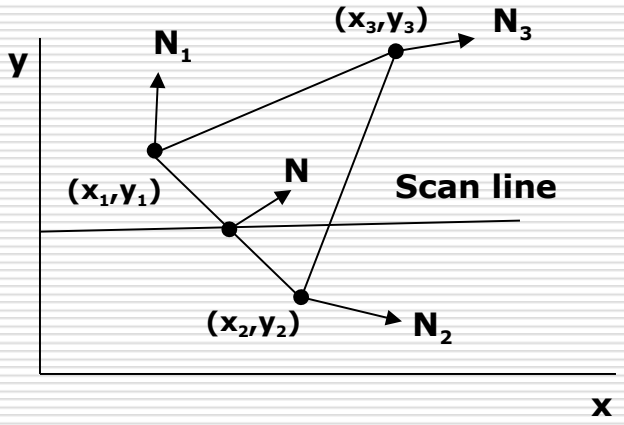
$$I_p = \frac{x_5 - x_p}{x_5 - x_4} I_4 + \frac{x_p - x_4}{x_5 - y_4} I_5$$

Recursive Calculation along the scan line ??



Phong Shading

- More accurate method for rendering
- Fundamental: *Interpolate normal vectors and apply illumination model to each surface point*
- Calculation steps:
  - Determine average unit normal vectors at each polygon vertex
  - Linearly interpolate the vertex normals over the surface of the polygon
  - Apply an illumination model along each scan line to calculate projected pixel intensities for the surface points
- **Trade-off**: requires considerably more calculations



$$N = \frac{y - y_2}{y_1 - y_2} N_1 + \frac{y_1 - y}{y_1 - y_2} N_2$$

Note: Students are encouraged to read Fast Phong Shading which could be useful for project works

Computer Graphics
EG678EX

Blunder

2. Liang-Barsky Line Clipping

- Based on parametric equation of a line:

  $x = x_1 + u.\triangle x$

  $y = y_1 + u.\triangle y \qquad 0 \le u \le 1$

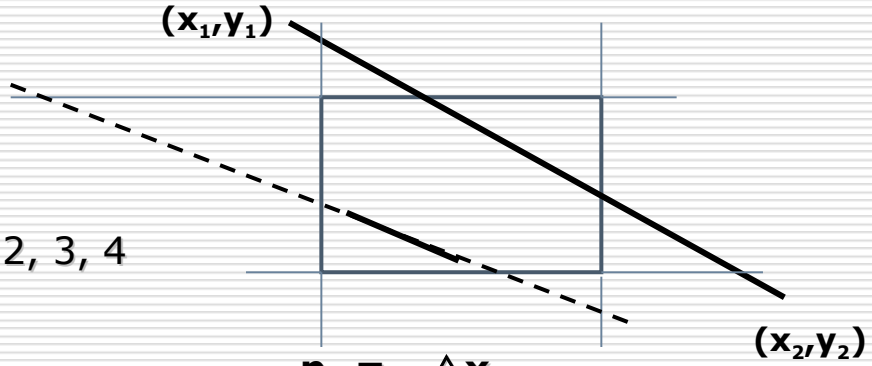
- Similarly, by adopting expressions for point clipping, the clipping window is represented by:

  $xw_{min} \le x_1 + u.\triangle x \le xw_{max}$

  $yw_{min} \le y_1 + u.\triangle y \le yw_{max}$

  … or,

  $u.\ p_k \le q_k \qquad k = 1, 2, 3, 4$



- where:

k = 1 (is the line inside left boundary ?) **$p_1 = -\triangle x$,**

$q_1 = x_1 - xw_{min}$

k = 2 (is the line inside right boundary ?) **$p_2 = \triangle x$, $q_2 =$**

$xw_{max} - x_1$

$P_k<0$ infinite extension of the line proceeds from outside to inside the infinitely extended boundary

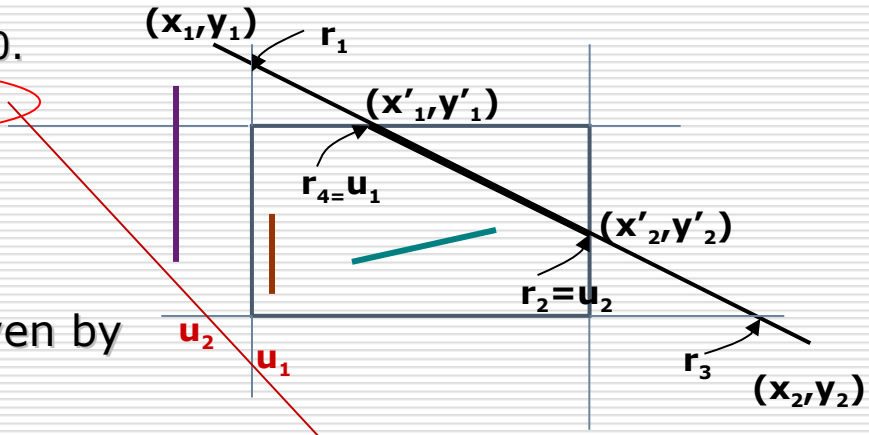
$P_k>0$ infinite extension of the line proceeds from inside to outside of the infinitely extended boundary

k = 4 (is the line inside top boundary ?) **$p_4 = \triangle y$, $q_4 = yw_{max}$**

$- y_1$

Liang-Barsky Line Clipping

- Trivial rejection and acceptance
  - Reject line with $p_k = 0$ and one $q_k < 0$.
  - Accept line with $p_k = 0$ and all $q_k \geq 0$



- For intersection with boundaries the parameters are supposed to be r_k given by

$$r_k = q_k/p_k$$

- Clipped line will be:

$x_1' = x_1 + u_1 . \triangle x; \quad u_1 \geq 0$

$y_1' = y_1 + u_1 . \triangle y;$

$x_2' = x_1 + u_2 . \triangle x; \quad u_2 \leq 1$

$y_2' = y_1 + u_2 . \triangle y;$

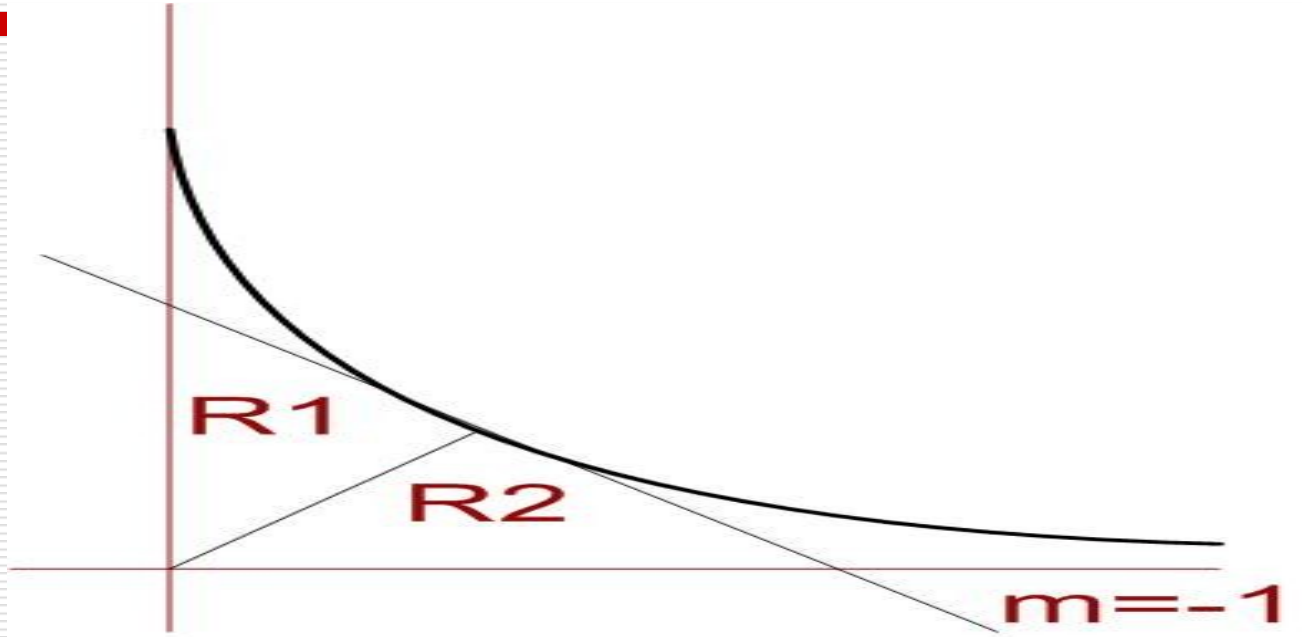
Merely satisfying this condition doesn't imply trivial acceptance,
however this condition implies that the line shouldn't be rejected trivially so needs further test

Liang-Barsky Algorithm Steps

1. If $p_k = 0$ for some k then the line is parallel to a clipping boundary.
   Now test q_k :
   if one $q_k < 0$ for these k then line is outside
   if all $q_k \geq 0$ then line is inside
2. For all $p_k < 0$ (i.e. line proceeds from outside to inside the boundary) calculate $u = \max(0, \{r_k : r_k = q_k / p_k\})$ to determine intersection point with the possibly extended clipping boundary k and obtain a new starting point for the line at u_1.
3. For all $p_k > 0$ (i.e. line proceeds from inside to outside the boundary) calculate $u_2 = \min(1, \{r_k : r_k = q_k/p_k\})$ to determine intersection point with extended clipping boundary k and obtain a new end point at u_2.
4. If $u_1 > u_2$ then discard the line
5. The line is now between $[u_1, u_2]$

Computer Graphics

EG678EX



Mid Point Algorithm for the curve

$$y = Ae^{-\alpha x}$$

$$\frac{dy}{dx} = -A\alpha.e^{-\alpha.x}$$

- At boundary

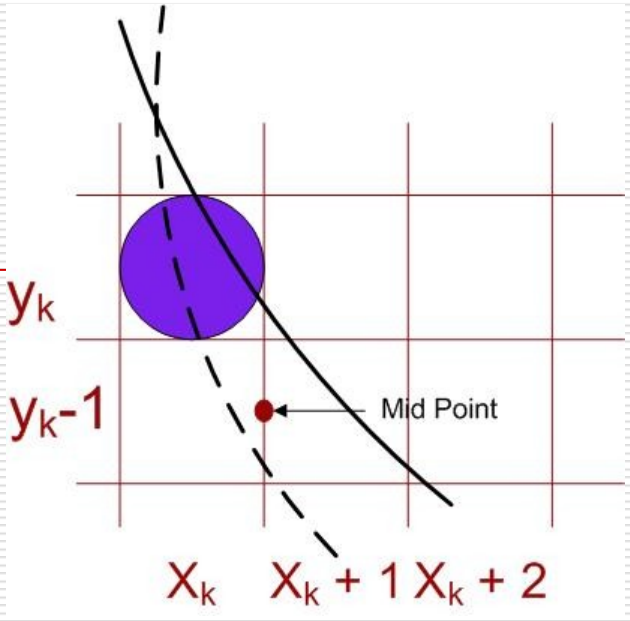
$$\frac{dy}{dx} = -1$$

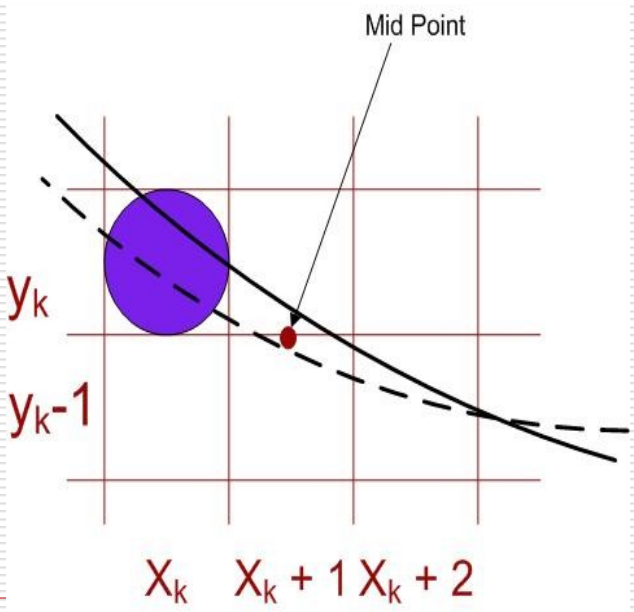
$$\Rightarrow x = \frac{\ln(A.\alpha)}{\alpha}$$

- Two regions
- Region 1→ |m|>1
- Region 2 → |m|<1

$$f(x,y) = Ae^{-\alpha x} - y$$

$$f(x,y)\begin{cases} <0, & if\ (x,y)\ is\ above\ the\ curve \\ =0, & if\ (x,y)\ is\ on\ the\ curve \\ >0, & if\ (x,y)\ is\ below\ the\ curve \end{cases}$$

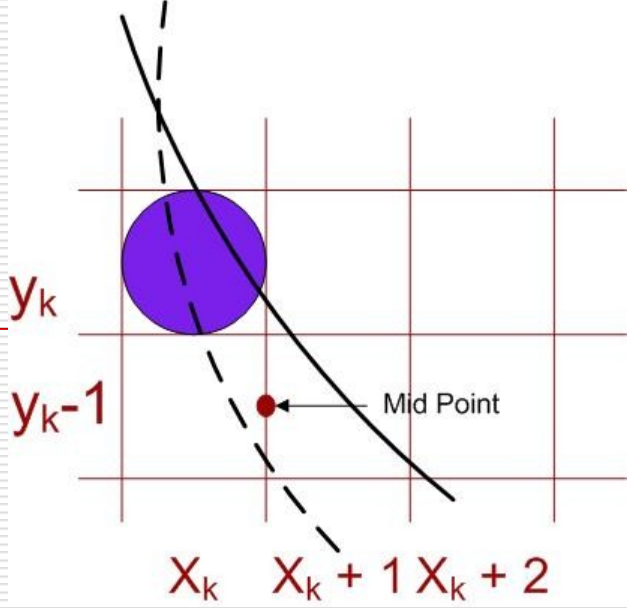




Region 1 $x < \frac{\ln(A.\alpha)}{\alpha}$

$$p1_k = f(x_k + \frac{1}{2}, y_k - 1)$$
$$= Ae^{-\alpha(x_k + \frac{1}{2})} - (y_k - 1)$$
$$= Ae^{-\alpha/2}.e^{-\alpha.x_k} - y_k + 1$$

$$p1_{k+1} = f(x_{k+1} + \frac{1}{2}, y_{k+1} - 1)$$
$$= Ae^{-\alpha(x_{k+1} + \frac{1}{2})} - [(y_k - 1) - 1]$$
$$= Ae^{-\alpha/2}.e^{-\alpha.x_{k+1}} - y_k + 2$$

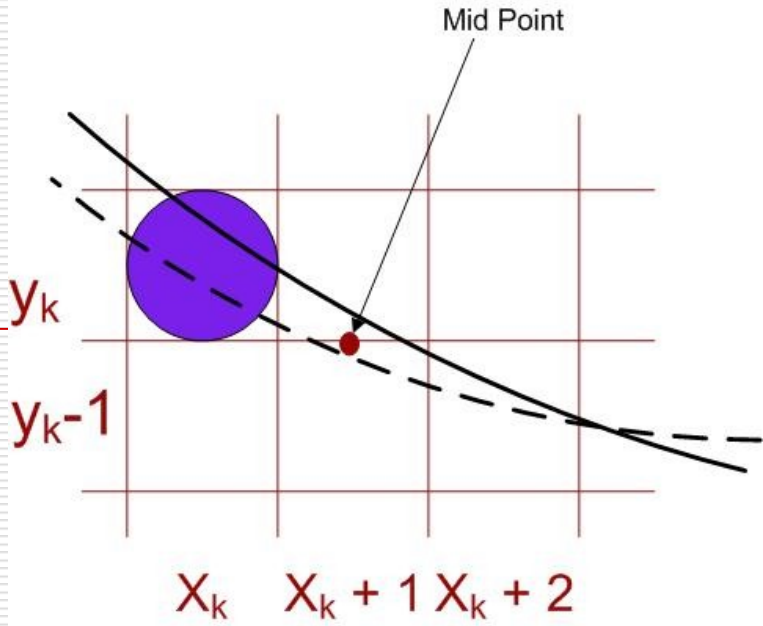


$$p1_0 = f(x_0 + \frac{1}{2}, y_0 - 1)$$
$$= Ae^{-\alpha(0 + \frac{1}{2})} - (A - 1)$$
$$= A(e^{-\alpha/2} - 1) + 1$$

$$p1_{k+1} = p1_k + Ae^{-\alpha/2}(e^{-\alpha.x_{k+1}} - e^{-\alpha.x_k}) + 1$$
$$= p1_k + 1 \quad if \quad p1_k < 0$$
$$= p1_k + Ae^{-\alpha/2}(e^{-\alpha.(x_k+1)} - e^{-\alpha.x_k}) + 1 \quad otherwise$$

Region 2

$$x > \frac{\ln(A.\alpha)}{\alpha}$$



$$p2_k = f(x_k + 1, y_k - \frac{1}{2})$$
$$= Ae^{-\alpha(x_k+1)} - (y_k - \frac{1}{2})$$
$$= Ae^{-\alpha}.e^{-\alpha.x_k} - y_k + \frac{1}{2}$$

$$p2_{k+1} = f(x_{k+1} + 1, y_{k+1} - \frac{1}{2})$$
$$= Ae^{-\alpha(x_k+2)} - (y_{k+1} - \frac{1}{2})$$
$$= Ae^{-2\alpha}.e^{-\alpha.x_k} - y_{k+1} + \frac{1}{2}$$

$$p2_{k+1} = p2_k + A(e^{-\alpha} - 1).e^{-\alpha.(x_k+1)} - (y_{k+1} - y_k)$$
$$= p2_k + A(e^{-\alpha} - 1).e^{-\alpha.(x_k+1)} \quad if\ p2_k > 0$$
$$= p2_k + A(e^{-\alpha} - 1).e^{-\alpha.(x_k+1)} + 1 \quad otherwise$$

$$p2_0 = f(x_0 + 1, y_0 - \frac{1}{2})$$
$$= Ae^{-\alpha}.e^{-\alpha.x_0} - y_0 + \frac{1}{2}$$

Computer Graphics
EG678EX

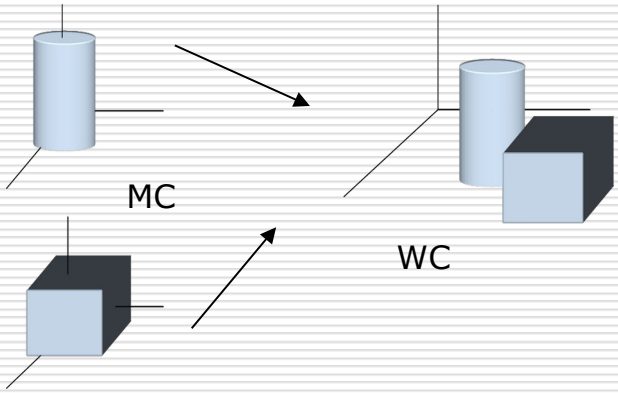
Graphics Software

Graphics Software

- Two categories
  - General Purpose programming package
    - Provides extensive sets of graphics functions that can be used in high level programming languages such as C or FORTRAN
    - Designed for programmers
    - Example: GL (Graphics Library)
    - Basic functions
      - to generate picture components as: straight lines, polygons, circles etc
      - Setting intensities, view selections and transformations
  - Special purpose application package
    - Designed for non-programmers
    - Users need not to know how graphics operations work
    - Interface to graphics routine allows users to communicate with programs in their own terms
    - Example: artist's painting programs and various business, medical and CAD systems

General Purpose programming package

- Coordinate representations: most graphics packages use cartesian coordinate specifications
  - Modeling coordinates: individual objects in a scene are described at individual coordinate reference frame called **modeling coordinates**
  - world coordinates
  - Device coordinates and normalized device coordinates
- Graphics Functions: a general purpose packages provides various functions
  - Categorized as output, input, attributes, transformations, viewing or general controls (clearing a display screen or initializing parameters)



Software Standards

- Portability of graphics software is primary goal
- Standards help to fulfill this goal
- Graphical Kernel System (GKS): first standard by International Standards Organization (ISO) and various organizations including American National Standards Institute (ANSI)
  - GKS: for 2D and 3D graphics
- Programmer's Hierarchical Interactive Graphics Standard (PHIGS)
  - Extension of GKS
  - Increased capability of object modeling, color specifications, surface rendering and picture manipulaions
  - PHIGS+ : extension of PHIGS for 3D surface shading that PHIG is not capable
- Computer Graphics Interface (CGI)
  - For graphics interface to output device
- Computer Graphics Metafile (CGM)
  - For archiving (archive = collection of pictures or documents) and transporting pictures

Project Management and Program Debugging

Project Management

- Essential part of software engineering
- Software management is distinct from other engineering management in as:
  - Intangibility of product:
    - ship engineering
      - product progress is visible
    - Software Engineering
      - Product progress is not visible rather than to rely on documentation
  - Lack of standard software processes
    - In other engineering there are standards and impact of particular process could be predicted but in case of software engineering, process varies from organization to organization and difficult to predict the impact of particular process
  - Uniqueness of the project
    - Large software projects are different from previous projects
    - Lessons from previous project may not be transferrable

Project Management Activities

- Most General activities
  - Proposal writing
  - Project planning and scheduling
    - Iterative process and continues throughout the project
    - More information becomes available → plans and schedules must be revised
  - Project cost estimation
    - Budgeting
    - Optimizing the project cost by time management and human resource management
  - Project monitoring and reviews
  - Personnel selection and evaluation
    - Skilled staff → deliverable outcome
  - Report writing and presentations
    - Managers responsibility to present the progress reports both to the clients and contractors
  - Risk Management
    - Project Risk → affected by project schedule or resources
    - Product Risk → quality and performance
    - Business risks → affects the organization developing or procuring the software

Program Debugging

- Process of finding and reducing the number of bugs
- General procedures
  - Reproduce the problem → reproduce the problem in while monitoring the output
    - Simplify the input after bug is produce to debug
  - Use debugger to track the origin of the problem after test case is simplified
  - Tracing is other alternative
    - Simple print statement is also a tracing
  - Remote debugging
    - Program runs on different machine than debugger
    - Debugger logs on to remote machine and controls the execution of program to identify the origin of problem