(STRICTLY FOR INTERNAL CIRCULATION ONLY)

COMPUTER GRAPHICS

Course: Computer Graphics (Semester VI, Computer Engineering, University of Mumbai)

Class: TE – CMPN (Atharva College of Engineering) May 2008

Course Material / Lecture Supplements prepared by Prof. Uday Nayak

Reference Book: Computer Graphics, Hearn and Baker

Note: This course material / lecture supplements are exclusively prepared for the students of TE – CMPN (May 2008) of Atharva College of Engineering. This is for Internal Circulation among TE – CMPN students only. Any distribution, reproduction of this material outside of Atharva College of Engineering is strictly prohibited. Any attempt to do so, will result in severe legal action and/or fine.

Contact:

Prof Uday Nayak

Department of Information technology,

Atharva College of Engineering,

Email: udaychandra@hotmail.com

2-D Geometric Transformations

Refer earlier(2D Transformation) notes.

NOTE: For 2D and 3D transformations, follow either this course material or Tech-Max (Godse); do not follow both, as there is a vast difference between the two. In this course material, the new transformed coordinates is found by multiplying the matrix M with the earlier coordinates i.e., $P' = M.P$ whereas in Tech-Max, it is the reverse i.e., $P' = P.M$ Hence, the matrix M is different in Tech-Max from the one in this course material. Also, the matrix P is also different in Tech-Max.

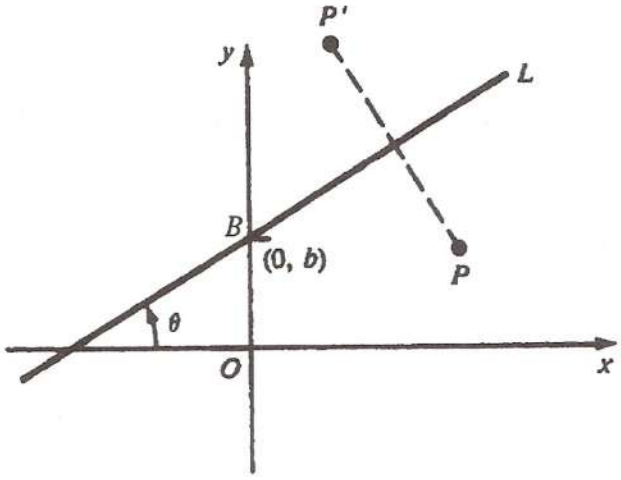
Mumbai University Questions:

May 2004: Derive the matrix for rotation about an arbitrary point.

Solution: Refer earlier notes page 9.

May 2005: Give the sequence of transformation to reflect the object about the line $y = mx + b$. Also, derive the composite matrix for it.

Solution:



Let the line L in above figure have a y-intercept (0, b) and an angle of inclination θ (with respect to the x-axis). We do the following sequence of transformations:

1) Translate the intersection point B to the origin.
2) Rotate by θ so that line L aligns with the x-axis.
3) Mirror-reflect about the x-axis.
4) Rotate back by θ.
5) Translate B back to (0, b).

In transformation notation, we have:

$$M_L = T(0, b).R_\theta.M_x.R_{-\theta}.T(0, -b)$$

Where M_x is the reflection matrix about the x-axis, which is:

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

Now, the angle of inclination of a line is related to its slope m by the equation $\tan\theta = m$, we have:

$$M_L = T(0,b).R_\theta.M_x.R_{-\theta}.T(0,-b)$$

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

Now, if $\tan\theta = m$, then, $\sin\theta = \frac{m}{\sqrt{m^2+1}}$ and $\cos\theta = \frac{1}{\sqrt{m^2+1}}$

Substituting these values for sin θ and cos θ after matrix multiplication, we have:

$$M_L = \begin{pmatrix} \frac{1-m^2}{m^2+1} & \frac{2m}{m^2+1} & \frac{-2bm}{m^2+1} \\ \frac{2m}{m^2+1} & \frac{m^2-1}{m^2+1} & \frac{2b}{m^2+1} \\ 0 & 0 & 1 \end{pmatrix}$$

We are finished now.

Question: Reflect the diamond-shaped polygon whose vertices are A(-1, 0), B(0, -2), C(1, 0), and D(0, 2) about the line y = x + 2.

Solution: Now, the equation of a line is y = mx + b. Here, m = 1 and b = 2.

Substituting these values in the matrix M_L, we get:

$$M_L = \begin{pmatrix} 0 & 1 & -2 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

The required coordinates A', B', C', and D' can now be found:

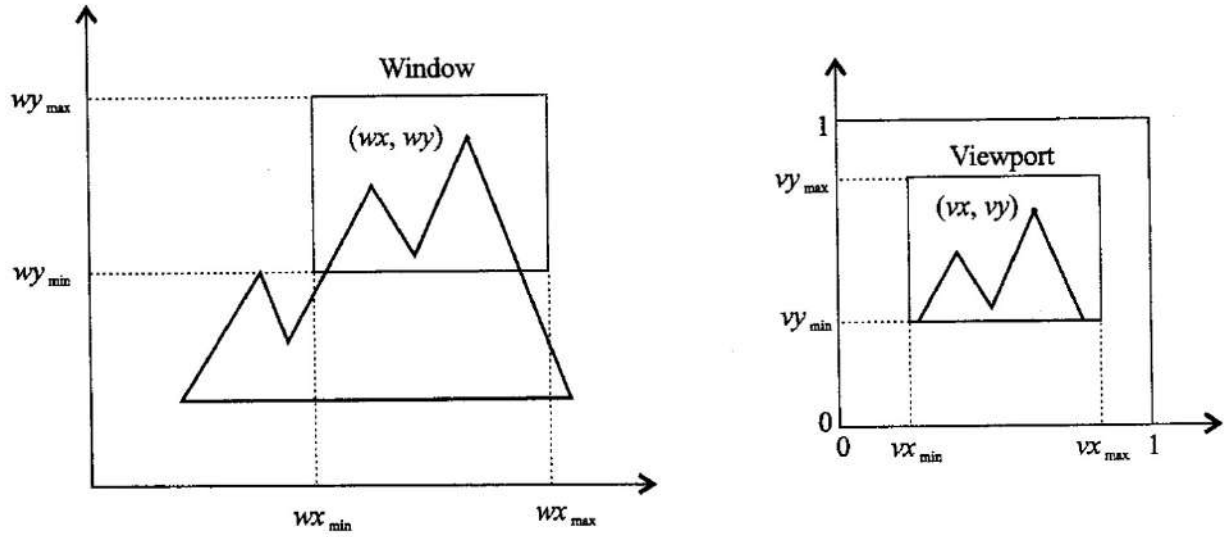
$$M_L.V = \begin{pmatrix} 0 & 1 & -2 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} -1 & 0 & 1 & 0 \\ 0 & -2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} -2 & -4 & -2 & 0 \\ 1 & 2 & 3 & 2 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

So, A' = (-2, 1), B'=(-4, 2), C' = (-2, 3), and D' = (0, 2).

2-D viewing and Clipping

Window-to-Viewport Mapping

A window is specified by four world coordinates: wx_{min}, wx_{max}, wy_{min}, and wy_{max}. (see below figure). Similarly, a viewport is described by four normalized device coordinates: vx_{min}, vx_{max}, vy_{min}, and vy_{max}.



Window-to-viewport mapping.

The objective of window-to-viewport mapping is to convert the world coordinates (wx, wy) of an arbitrary point to its corresponding normalized device coordinates (vx, vy). In order to maintain the same relative placement of the point in the viewport as in the window, we require:

$$\frac{wx-wx_{min}}{wx_{max}-wx_{min}} = \frac{vx-vx_{min}}{vx_{max}-vx_{min}} \quad \text{and} \quad \frac{wy-wy_{min}}{wy_{max}-wy_{min}} = \frac{vy-vy_{min}}{vy_{max}-vy_{min}}$$

Thus,

$$vx = \frac{vx_{max}-vx_{min}}{wx_{max}-wx_{min}}\ (wx - wx_{min}) + \ vx_{min}$$

$$vy = \frac{vy_{max}-vy_{min}}{wy_{max}-wy_{min}}\ (wy - wy_{min}) + \ vy_{min}$$

We can express these two formulas for computing (vx, vy) from (wx, wy) in terms of translate-scale-translate transformation N

$$\begin{pmatrix} vx \\ vy \\ 1 \end{pmatrix} = N \cdot \begin{pmatrix} wx \\ wy \\ 1 \end{pmatrix}$$

where

$$N = \begin{pmatrix} 1 & 0 & vx_{min} \\ 0 & 1 & vy_{min} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -wx_{min} \\ 0 & 1 & -wy_{min} \\ 0 & 0 & 1 \end{pmatrix}$$

where

$s_x = \frac{vx_{max} - vx_{min}}{wx_{max} - wx_{min}}$ and $s_y = \frac{vy_{max} - vy_{min}}{wy_{max} - wy_{min}}$

Example: Find the normalization transformation that maps a window whose lower left corner is at (1, 1) and upper right corner is at (3, 5) onto a viewport that is the entire normalized device screen.

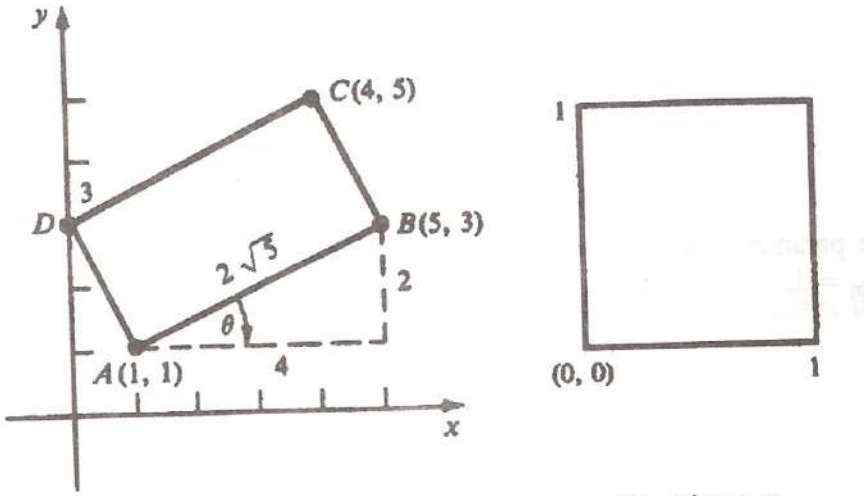
Solution: The window parameters are wx_{min}=1, wx_{max}=3, wy_{min}=1, and wy_{max}=5. The viewport parameters are vx_{min}=0, vx_{max}=1, vy_{min}=0, and vy_{max}=1. Then, s_x=1/2 and s_y=1/4., and

$$N = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 & -\frac{1}{2} \\ 0 & \frac{1}{4} & -\frac{1}{4} \\ 0 & 0 & 1 \end{pmatrix}$$

Mumbai University Questions:

May 2007: Find the normalization transformation which uses the rectangle A (1, 1), B (5, 3), C (4, 5), and D (0, 3) as window and NDC as a viewport.

Solution: Here, NDC means Normalized Device Coordinates. See below figure.



(a) Window. (b) Viewport.

We will first rotate the rectangle A so that it is aligned with the coordinate axes. Next, we calculate s_x, and s_y and finally we compose rotation and the transformation N to find the required normalization transformation N_R. Looking at the figure, we see that –θ will be the direction of the rotation. The angle θ is determined from the slope of a line by the equation $\tan\theta = \frac{1}{2}$. Then,

$$\sin\theta = \frac{1}{\sqrt{5}}, \quad \textit{and so} \quad \sin(-\theta) = -\frac{1}{\sqrt{5}}, \quad \cos\theta = \frac{2}{\sqrt{5}}, \quad \cos(-\theta) = \frac{2}{\sqrt{5}}$$

The rotation matrix about A (1, 1) is then (from page 9 of 2D transformations notes):

$$R_{-\theta,A} = \begin{pmatrix} \cos\theta & -\sin\theta & x_f(1-\cos\theta) + y_f \sin\theta \\ \sin\theta & \cos\theta & y_f(1-\cos\theta) - x_f \sin\theta \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \left(1-\frac{3}{\sqrt{5}}\right) \\ -\frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & \left(1-\frac{1}{\sqrt{5}}\right) \\ 0 & 0 & 1 \end{pmatrix}$$

The x extent of the rotated window is the length of AB. Similarly, the y extent is the length of AD. Now,

$$d(A,B) = \sqrt{2^2+4^2} = 2\sqrt{5} \qquad d(A,D) = \sqrt{1^2+2^2} = \sqrt{5}$$

Also, the x extent of the normalized device screen is 1, as is the y extent. Calculating s_x, and s_y,

$$s_x = \frac{viewport\ x\ extent}{window\ x\ extent} = \frac{1}{2\sqrt{5}} \qquad s_y = \frac{viewport\ y\ extent}{window\ y\ extent} = \frac{1}{\sqrt{5}}$$

So,

$$N = \begin{pmatrix} \frac{1}{2\sqrt{5}} & 0 & -\frac{1}{2\sqrt{5}} \\ 0 & \frac{1}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \\ 0 & 0 & 1 \end{pmatrix}$$

The normalization transformation is then

$$N_R = N.R_{-\theta,A} = \begin{pmatrix} \frac{1}{5} & \frac{1}{10} & -\frac{3}{10} \\ -\frac{1}{5} & \frac{2}{5} & -\frac{1}{5} \\ 0 & 0 & 1 \end{pmatrix}$$

Point Clipping:

Point clipping is essentially the evaluation of the following inequalities:

$x_{min} \leq x \leq x_{max}$ and $y_{min} \leq y \leq y_{max}$

where x_{min}, x_{max}, y_{min} and y_{max} define the clipping window. A point (x, y) is considered inside the window when the inequalities all evaluate to true.

Cohen-Sutherland Line Clipping Algorithm:

In this algorithm, we divide the line clipping process into three phases:
Step 1: Identify those lines which intersect the clipping window and so need to be clipped
Step 2: Perform the clipping.
Step 3: Iterative Process.

Step 1: All lines fall into one of the following clipping categories:

Category 1 - Visible – both endpoints of the line lie within the window.
Category 2 - Not Visible – the line definitely lies outside the window.
Category 3 - Clipping candidate – the line is in neither category 1 nor 2.

The algorithm employs an efficient procedure for **finding the category of a line**. It proceeds in two steps:

1. Assign a **4-bit region code** to each endpoint of a line. The code is determined according to which of the following nine regions of the plane, the endpoint lies in:

1001	1000	1010
0001	0000 Window	0010
0101	0100	0110

Starting from the **rightmost bit**, each bit of the code is set to **true (1)** or **false (0)** according to the scheme:

- **Bit 1**: endpoint is to the ***left*** of the window
- **Bit 2**: endpoint is to the ***right*** of the window
- **Bit 3**: endpoint is to the ***below*** of the window
- **Bit 4**: endpoint is to the ***above*** of the window

2. The line is:
   - **Visible**: If both endpoints' region codes are **0000**
   - **Not Visible**: If the ***bitwise logical AND*** of the region codes is not **0000**
   - **Clipping Candidate**: If the ***bitwise logical AND*** of the region codes is **0000**

Step 2: Clipping Process:

For a line in **category 3 (Clipping candidate)**, we proceed to find the **intersection point** of the line with one of the boundaries of the clipping window. We choose an endpoint of the line, say (x_1, y_1), that is outside the window, i.e., whose region code is **not 0000**. Next, we do the following:

- **If bit 1 is 1, intersect with line $x = x_{min}$**
- **If bit 2 is 1, intersect with line $x = x_{max}$**
- **If bit 3 is 1, intersect with line $y = y_{min}$**
- **If bit 4 is 1, intersect with line $y = y_{max}$**

The **coordinates** of the **intersection point** are

$x_i = x_{min}$ or x_{max} AND $y_i = y_1 + m(x_i - x_1)$ if the boundary line is vertical

Or

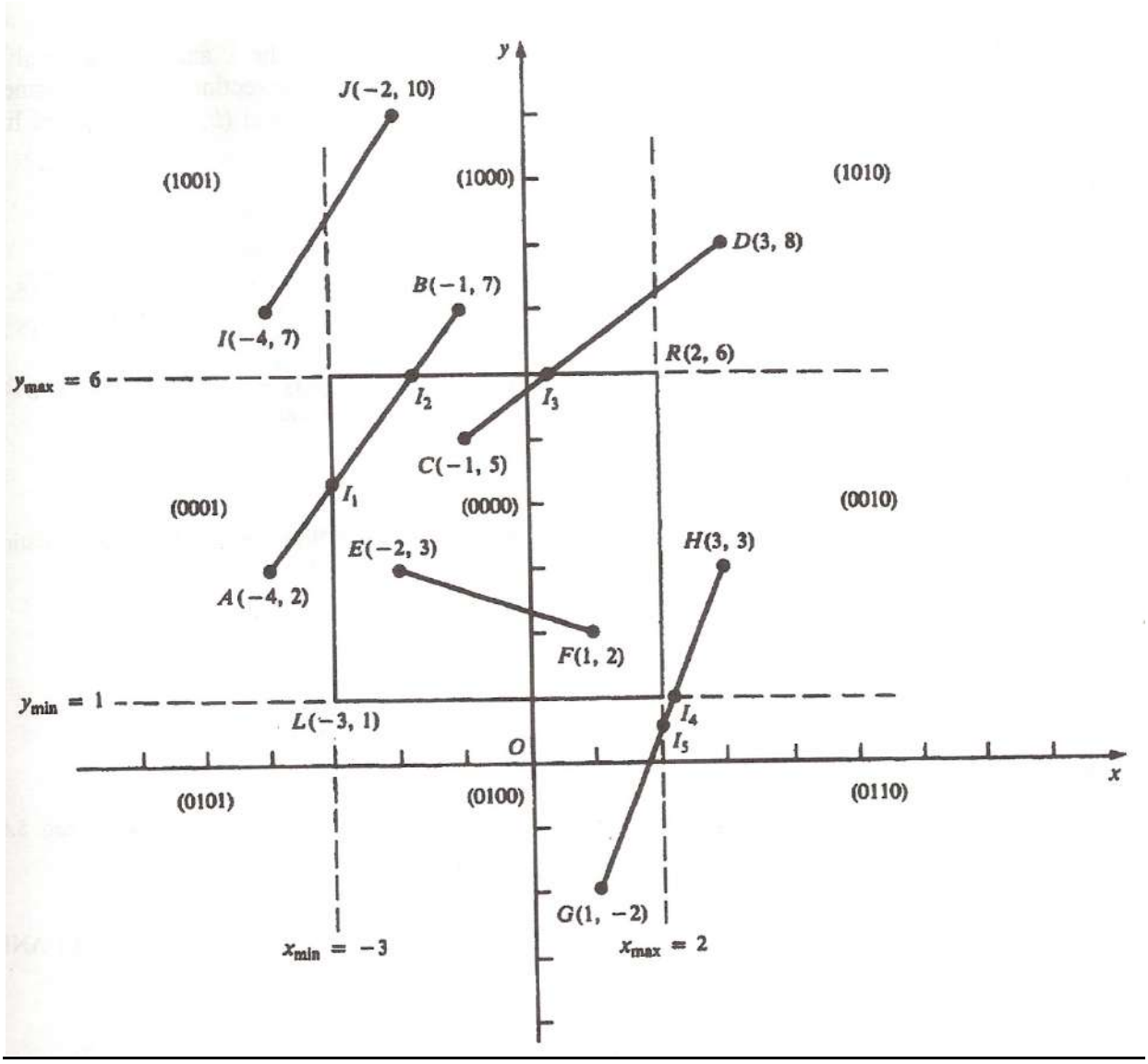
$x_i = x_1 + (y_i - y_1)/m$ AND $y_i = y_{min}$ or y_{max} if the boundary line is horizontal

where $m = (y_2 - y_1)/(x_2 - x_1)$

Step 3: Iterative Process:

- Now, we **replace endpoint** (x_1, y_1) with the **intersection point** (x_i, y_i) , effectively eliminating the portion of the original line that is on the outside of the selected window boundary.
- The new endpoint is then assigned an **updated region code** and the **clipped line re-categorized** and handled in the same way.
- This iterative process terminates when we finally reach a **clipped line** that belongs to either **Category 1 (Visible)** or **Category 2 (Not visible)**.

Cohen-Sutherland Line Clipping Example:



Step 1: Categorize the line segments into one of the three sections.

For line AB, bitwise AND of region codes is 0001 ∩ 1000 = 0000, therefore Clipping candidate.

For line CD, bitwise AND of region codes is 0000 ∩ 1010 = 0000, therefore Clipping candidate.

For line EF, both region codes are 0000, hence, visible.

For line GH, bitwise AND of region codes is 0100 ∩ 0010 = 0000, therefore Clipping candidate.

For line IJ, bitwise AND of region codes is 1001 ∩ 1000 = 1000 ≠0000, therefore not visible.

Step 2: Clipping Process

Consider line AB:

Take A (-4, 2) = (x_1, y_1) = 0001 ; since **bit 1 is 1, intersect with line $x = x_{min}$= -3**

x_i = -3 ; B (-1, 7) = (x_2, y_2) ; $m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{7-2}{-1-(-4)} = \frac{5}{3}$;

$y_i = y_1 + m\,(x_i - x_1) = 2 + \frac{5}{3}\left(-3-(-4)\right) = \frac{11}{3}$; Hence, I_1 = (-3, $\frac{11}{3}$)

Now, consider line I_1B:

For line I_1B, bitwise AND of region codes is 0000 ∩ 1000 = 0000, therefore Clipping candidate.

Since, B is outside the clipping window,

Take B (-1, 7) = (x_1, y_1) = 1000 ; since **bit 4 is 1, intersect with line $y = y_{max}$= 6**

y_i = 6 ; I_1 (-3, $\frac{11}{3}$) = (x_2, y_2) ; $m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{11/3 - 7}{-3-(-1)} = \frac{5}{3}$;

$x_i = x_1 + \frac{y_i - y_1}{m} = -1 + \frac{6-7}{5/3} = -\frac{8}{5}$; Hence, I_2 = (- $\frac{8}{5}$, 6)

Now, consider line I_1I_2:

For line I_1I_2, both region codes are 0000, hence, visible. *Now, we are done with line AB.*

Consider line CD:

Since, D is outside the clipping window,

Take D (3, 8) = (x_1, y_1) = 1010 ; since **bit 4 is 1, intersect with line $y = y_{max}$= 6**

y_i = 6 ; C (-1, 5) = (x_2, y_2) ; $m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{5-8}{-1-3} = \frac{3}{4}$;

$x_i = x_1 + \frac{y_i - y_1}{m} = 3 + \frac{6-8}{3/4} = \frac{1}{3}$; Hence, I_3 = ($\frac{1}{3}$, 6)

Now, consider line CI_3:

For line CI_3, both region codes are 0000, hence, visible. *Now, we are done with line CD.*

Consider line GH:

Take G (1, -2) = (x_1, y_1) = 0100 ; since bit 3 **is 1, intersect with line $y = y_{min}$ = 1**

y_i = 1 ; H (3, 3) = (x_2, y_2) ; $m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{3-(-2)}{3-1} = \frac{5}{2}$;

$x_i = x_1 + \frac{y_i - y_1}{m} = 1 + \frac{1-(-2)}{5/2} = \frac{11}{5}$; Hence, I_4 = ($\frac{11}{5}$, 1)

Now, consider line GI_4:

For line GI_4, bitwise AND of region codes is 0010 ∩ 0010 = 0010, therefore line not visible.

We are now finished with all the lines.

Mumbai University Questions:

May 2004: How a line between (2, 2) and (12, 9) is clipped against window with (xw_{min}, yw_{min}) = (4, 4) and (xw_{max}, yw_{max}) = (9, 8)?

Solution: Let the line be represented by AB where A is (2, 2) and B is (12, 9). Region code for A is 0101 and for B is 1010. Hence, logical AND of 0101, 1010 = 0000, hence, clipping candidate.

Draw the diagram, then, we see that AB intersects with lines x_{min} and y_{min}, we can find both intersection points, but finding the intersection point of AB and x_{min} is a waste, since, anyway the line is going to be clipped at the next intersection point of AB and y_{min}. So, instead of finding both intersection points, let's find out only one intersection point that matters, i.e., intersection point of AB and y_{min}.

Take A (2, 2) = (x_1, y_1) = 0101; since **bit 3 is 1, intersect with line y= y_{min}= 4**

y_i = 4 ; B (12, 9) = (x_2, y_2) ; $m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{9-2}{12-2} = \frac{7}{10}$;

$x_i = x_1 + \frac{y_i - y_1}{m} = 2 + \frac{4-2}{7/10} = \frac{34}{7}$; Hence, I_1 = ($\frac{34}{7}$, 4)

Now, consider line I_1B, bitwise AND of region codes is 0000 ∩ 1010 = 0000, therefore clipping candidate.

Since B is outside the window boundary, let's start with B. Again, there will be two intersection points of line I_1B, one with x_{max}, and one with y_{max}. Let's find the intersection point that really matters, i.e, intersection point of I_1B and x_{max}.

Considering B (12, 9) = (x_1, y_1) = 1010; since **bit 2 is 1, intersect with line x= x_{max}= 9**

x_i = 9 ; I_1 ($\frac{34}{7}$, 4) = (x_2, y_2) ; $m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{9-4}{12-34/7} = \frac{7}{10}$;

$y_i = y_1 + m\,(x_i - x_1) = 9 + \frac{7}{10}\,(9 - 12) = 6.9$; Hence, I_2 = (9, 6.9)

Hence, line I_1B is clipped and we get lineI_1I_2 which is the clipped line between I_1 ($\frac{34}{7}$, 4) and I_2 (9, 6.9). We are finished now.

May 2006 & Dec 2007: Find the clipping coordinates to clip the line segment AB against the window using Cohen-Sutherland line clipping algorithm. A (120, 60), B (160, 92); x_{min}=100; y_{min}=80; x_{max}=150; y_{max}=100.

Solution: Solve using the same method as above.

May 2007: Find the clipping coordinates of the line joining A (-1, 5) and B (3, 8). Given x_{min}=-3; y_{min}=1; x_{max}=2; y_{max}=6.

Solution: Solve using the same method as above.

Liang-Barsky Line Clipping Algorithm:

The following parametric equations represent a line from (x_1, y_1) to (x_2, y_2).

$$x = x_1 + \Delta x \,.\, u = x_1 + (x_2 - x_1).u$$

$$y = y_1 + \Delta y \,.\, u = y_1 + (y_2 - y_1).u$$

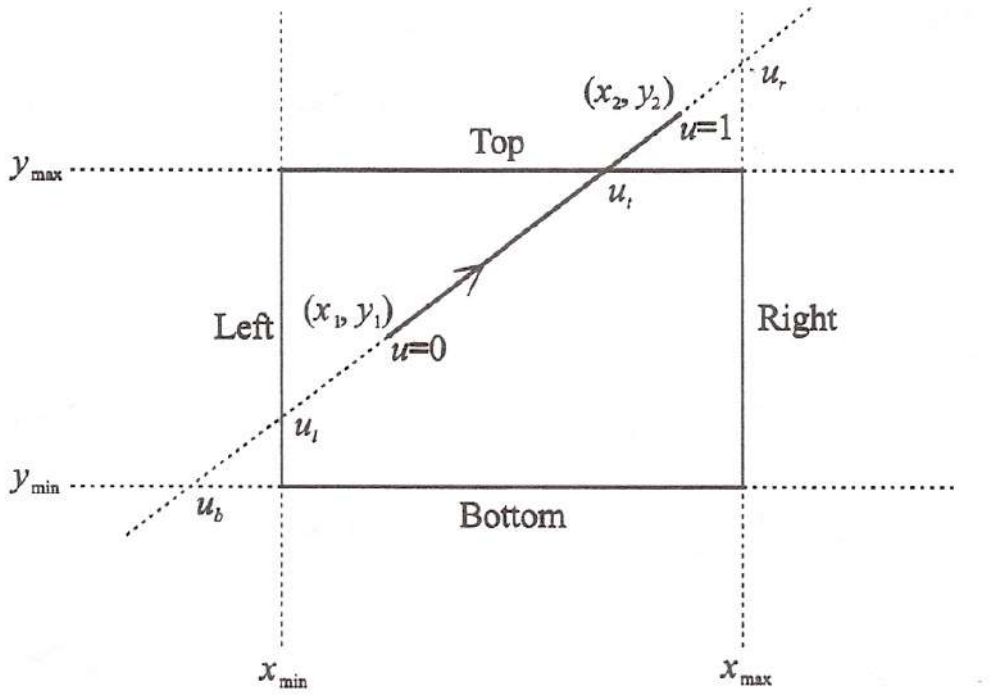
With $0 \leq u \leq 1$

When u = 0, we have x = x_1, y = y_1 and when u = 1, we have x = x_2, y = y_2

If we use u_1 and u_2 , where $u_1 \leq u_2$, to represent the beginning and end of the visible portion of the line, we have

u_1 = maximum (0, u_l, u_b) and u_2 = minimum (1, u_t, u_r),

where u_l, u_b, u_t, and u_r correspond to the intersection point of the extended line with the window's left, bottom, top, and right boundary, respectively.



For a point (x, y) inside the clipping window, we have

$$x_{min} \leq x_1 + \Delta x \,.\, u \leq x_{max}$$

$$y_{min} \leq y_1 + \Delta y \,.\, u \leq y_{max}$$

Rewrite the four inequalities as

$$p_k.u \leq q_k \,, \quad k = 1,2,3,4$$

Where

$p_1 = -\Delta x$; $q_1 = x_1 - x_{min}$; (left)

$p_2 = \Delta x$; $q_2 = x_{max} - x_1$; (right)

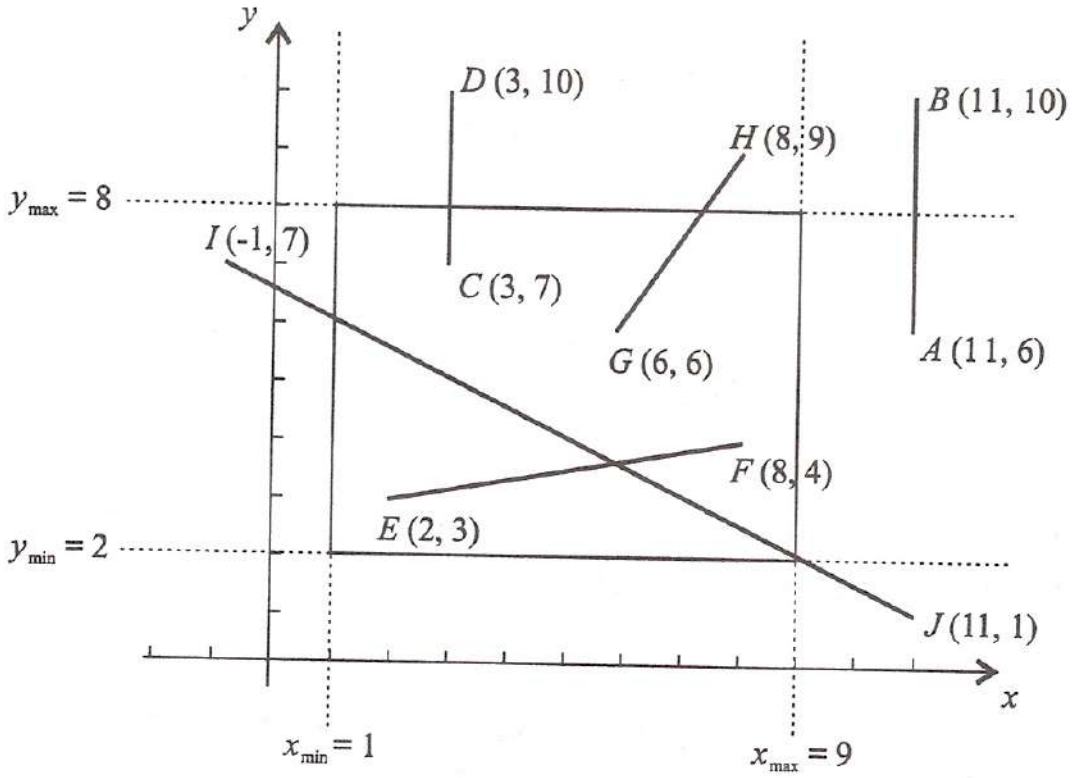
$p_3 = -\Delta y$; $q_3 = y_1 - y_{min}$; (bottom)

$p_4 = \Delta y$; $q_4 = y_{max} - y_1$; (top)

Observe the following facts:

- If $p_k = 0$, the line is parallel to the corresponding boundary and
  - if $q_k < 0$, the line is completely outside the boundary and can be eliminated.
  - If $q_k \geq 0$, the line is inside the boundary and needs further consideration.
- If $p_k < 0$, the extended line proceeds from the outside to the inside of the corresponding boundary line,
- If $p_k > 0$, the extended line proceeds from the inside to the outside of the corresponding boundary line,
- When $p_k \neq 0$, the value of u that corresponds to the intersection point is $^{q_k}/_{p_k}$.

Liang-Barsky Line Clipping Example:



For line AB, we have

$p_1 = 0$; $q_1 = 10$

$p_2 = 0$; $q_2 = -2$

$p_3 = -4$; $q_3 = 4$

$p_4 = 4$; $q_4 = 2$

Since, $p_2 = 0$ and $q_2 < 0$, AB is completely outside the right boundary.

For line CD, we have

$p_1 = 0$; $q_1 = 2$

$p_2 = 0$; $q_2 = 6$

$p_3 = -3$; $q_3 = 5$; $r_3 = -\frac{5}{3}$; (bottom)

$p_4 = 3$; $q_4 = 1$; $r_4 = \frac{1}{3}$; (top)

Since, $p_k = 0$ and $q_k > 0$, hence, clipping candidate.

Now, $u_1 = \max\left(0, -\frac{5}{3}\right) = 0$ *and* $u_2 = \min\left(1, \frac{1}{3}\right) = \frac{1}{3}$. Since, $u_1 < u_2$ we proceed.

Therefore, intersection point for u_2:

$x = x_1 + \Delta x \,.\, u = 3 + (0).\frac{1}{3} = 3$; $y = y_1 + \Delta y \,.\, u = 7 + (3).\frac{1}{3} = 8$; $I_1 = (3, 8)$

For line EF, we have

$p_1 = -6$; $q_1 = 1$; $r_1 = -\frac{1}{6}$; (left)

$p_2 = 6$; $q_2 = 7$; $r_2 = \frac{7}{6}$; (right)

$p_3 = -1$; $q_3 = 1$; $r_3 = -1$; (bottom)

$p_4 = 1$; $q_4 = 5$; $r_4 = 5$; (top)

Now, $u_1 = \max\left(0, -\frac{1}{6}, -1\right) = 0 \quad and \quad u_2 = \min\left(1, \frac{7}{6}, 5\right) = 1$.

Since $u_1 = 0$ and $u_2 = 1$, line EF is completely inside the clipping window.

For line GH, we have

$p_1 = -2$; $q_1 = 5$; $r_1 = -\frac{5}{2}$; (left)

$p_2 = 2$; $q_2 = 3$; $r_2 = \frac{3}{2}$; (right)

$p_3 = -3$; $q_3 = 4$; $r_3 = -\frac{4}{3}$; (bottom)

$p_4 = 3$; $q_4 = 2$; $r_4 = \frac{2}{3}$; (top)

Now, $u_1 = \max\left(0, -\frac{5}{2}, -\frac{4}{3}\right) = 0 \quad and \quad u_2 = \min\left(1, \frac{3}{2}, \frac{2}{3}\right) = \frac{2}{3}$. Since, $u_1 < u_2$ we proceed.

Therefore, intersection point for u_2:

$x = x_1 + \Delta x \,.\, u = 6 + (2).\frac{2}{3} = \frac{22}{3}$; $y = y_1 + \Delta y \,.\, u = 6 + (3).\frac{2}{3} = 8$; $I_2 = (\frac{22}{3}, 8)$.

For line IJ, we have

$p_1 = -12$; $q_1 = -2$; $r_1 = \frac{1}{6}$; (left)

$p_2 = 12$; $q_2 = 10$; $r_2 = \frac{5}{6}$; (right)

$p_3 = 6$; $q_3 = 5$; $r_3 = \frac{5}{6}$; (bottom)

$p_4 = -6$; $q_4 = 1$; $r_4 = -\frac{1}{6}$; (top)

This is a special case since line IJ is going from left, top to right, bottom.

Hence, $u_1 = \max(0, u_l, u_t)$; and $u_2 = \min(1, u_r, u_b)$;

i.e., $u_1 = \max\left(0, \frac{1}{6}, -\frac{1}{6}\right) = \frac{1}{6}$ *and* $u_2 = \min\left(1, \frac{5}{6}, \frac{5}{6}\right) = \frac{5}{6}$. Since, $u_1 < u_2$ we proceed.

Therefore, intersection point for u_1:

$x = x_1 + \Delta x \,.\, u = -1 + (12).\frac{1}{6} = 1$; $y = y_1 + \Delta y \,.\, u = 7 + (-6).\frac{1}{6} = 6$; $I_3 = (1, 6)$.

Therefore, intersection point for u_2:

$x = x_1 + \Delta x \,.\, u = -1 + (12).\frac{5}{6} = 9$; $y = y_1 + \Delta y \,.\, u = 7 + (-6).\frac{5}{6} = 2$; $I_4 = (9, 2)$.

We are finished with all the lines now.

Mumbai University Questions:

May 2005: Find the clipping coordinates to clip the line segment AB against the window using Liang-Barsky line clipping algorithm. A (20, 20), B (80, 110) and the window coordinates are: lower left corner of the window (40, 40) and upper right corner (100, 90).

Solution: We have: A (x_1, y_1), B (x_2, y_2) and x_{min}=40; y_{min}=40; x_{max}= 100; y_{max}= 90;

$p_1 = -\Delta x = -(80 - 20) = -60;$ $q_1 = x_1 - x_{min} = 20 - 40 = -20$; $r_1 = \frac{-20}{-60} = \frac{1}{3}$ (left)

$p_2 = \Delta x = 80 - 20 = 60$; $q_2 = x_{max} - x_1 = 100 - 20 = 80;$ $r_2 = \frac{80}{60} = \frac{4}{3}$ (right)

$p_3 = -\Delta y = -(110 - 20) = -90$; $q_3 = y_1 - y_{min} = 20 - 40 = -20$; $r_3 = \frac{-20}{-90} = \frac{2}{9}$ (bottom)

$p_4 = \Delta y = 110 - 20 = 90$; $q_4 = y_{max} - y_1 = 90 - 20 = 70$; $r_4 = \frac{70}{90} = \frac{7}{9}$ (top)

Now, $u_1 = \max\left(0, \frac{1}{3}, \frac{2}{9}\right) = \frac{1}{3}$ *and* $u_2 = \min\left(1, \frac{4}{3}, \frac{7}{9}\right) = \frac{7}{9}$.

Since, $u_1 < u_2$ we proceed.

Therefore, intersection point for u_1:

$x = x_1 + \Delta x \,.\, u = 20 + (60).\frac{1}{3} = 40$; $y = y_1 + \Delta y \,.\, u = 20 + (90).\frac{1}{3} = 50$; $I_1 = (40, 50)$.

Therefore, intersection point for u_2:

$x = x_1 + \Delta x \,.\, u = 20 + (60).\frac{7}{9} = 66.7$; $y = y_1 + \Delta y \,.\, u = 20 + (90).\frac{7}{9} = 90$; $I_2 = (66.7, 90)$.

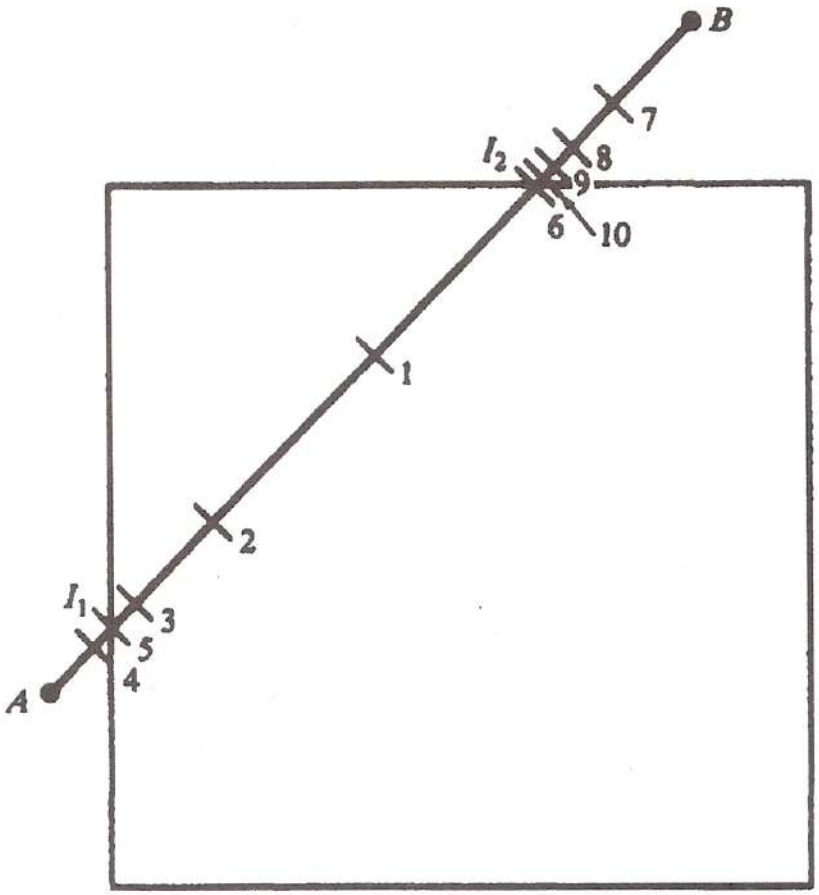
Therefore, the clipped line is I_1I_2 from (40, 50) to (66.7, 90).

Midpoint Subdivision Method:

An alternative way to process a line in category 3 is based on binary search. The line is divided at its midpoint into two shorter line segments. The clipping categories of the two new line segments are then determined by their region codes. Each segment in category 3 is divided again into shorter segments and categorized. This bisection and categorization process continues until each line segment that spans across a window boundary (hence encompasses an intersection point) reaches a threshold for line size and all other segments are either in category 1 (visible) or in category 2 (invisible). The midpoint coordinates (x_m, y_m) of a line joining (x_1, y_1) and (x_2, y_2) are given by

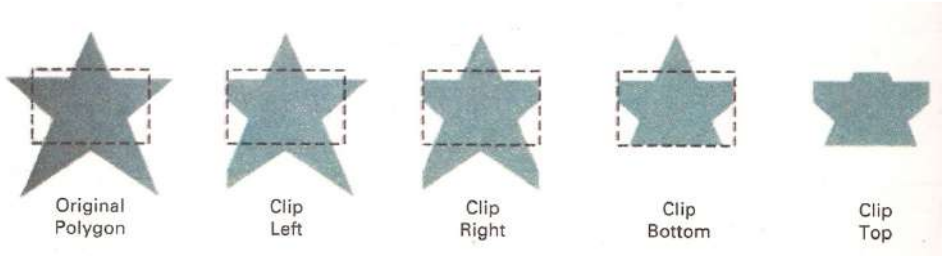
$$x_m = \frac{x_1 + x_2}{2}; \qquad y_m = \frac{y_1 + y_2}{2};$$

The example in below figure illustrates how midpoint subdivision is used to zoom in onto the intersection points I and I with 10 bisections. The process continues until we reach two line segments that are, say, pixel-sized, i.e., mapped to one single pixel each in the image space. If the maximum number of pixels in a line is *M,* this method will yield a pixel-sized line segment in *N* subdivisions, where $2^N = M \ \ or \ \ N = log_2 M$. For instance, when *M* = 1024 we need at most *N* = $\log_2 1024 = 10$ subdivisions.



Sutherland-Hodgeman Polygon Clipping:

We can correctly clip a polygon by processing the polygon boundary as a whole against each window edge. This could be accomplished by processing all polygon vertices against each clip rectangle boundary in turn. Beginning with the initial set of polygon vertices, we could first clip the polygon against the left rectangle boundary to produce a new sequence of vertices. The new set could then be successively passed to a right boundary clipper, a bottom boundary clipper, and a top boundary clipper, as in below figure. At each step, a new sequence of output vertices is generated and passed to the next window boundary clipper.



There are four possible cases when processing vertices in sequence around the perimeter of a polygon. As each pair of adjacent polygon vertices is passed to a window boundary clipper, we make the following tests:

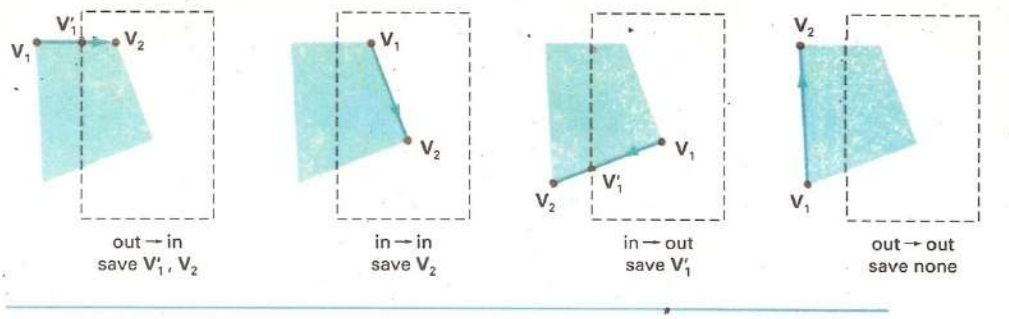
(1) If the first vertex is outside the window boundary and the second vertex is inside, both the intersection point of the polygon edge with the window boundary and the second vertex are added to the output vertex list.

(2) If both input vertices are inside the window boundary, only the second vertex is added to the output vertex list.

(3) If the first vertex is inside the window boundary and the second vertex is outside, only the edge intersection with the window boundary is added to the output list.

(4) If both input vertices are outside the window boundary, nothing is added to the output list.

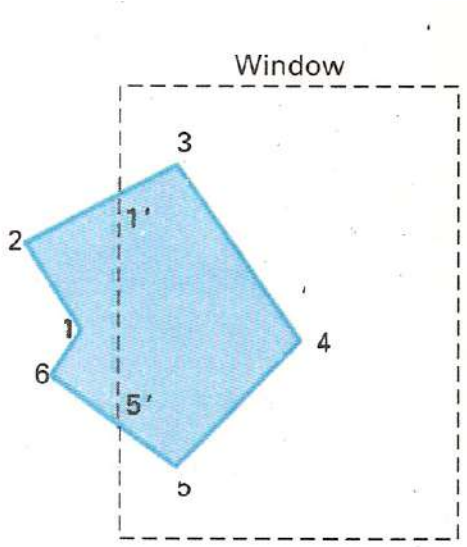
These four cases are illustrated in below figure for successive pairs of polygon vertices. Once all vertices have been processed for one clip window boundary, the output list of vertices is clipped against the next window boundary.



Successive processing of pairs of polygon vertices against the left window boundary.

Example 1 of Polygon Clipping:

Clipping the polygon against the left boundary of the window.



Edge 12 is out-to-out (case 4), hence save none.

Edge 23 is out-to-in (case 1), hence save intersection point 1' and 3.

Edge 34 is in-to-in (case 2), hence save 4.

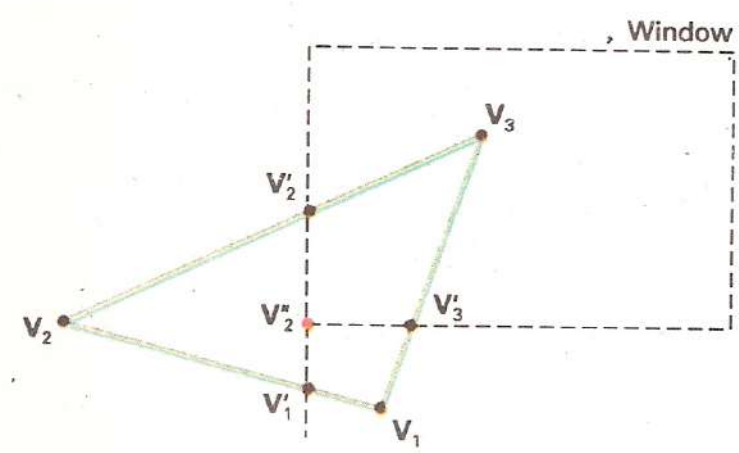
Edge 45 is in-to-in (case 2), hence save 5.

Edge 56 is in-to-out (case 3), hence save 5'.

Edge 61 is out-to-out (case 4), hence save none.

Now, the saved vertices are 1', 3, 4, 5, and 5', which is the clipped polygon.

Example 2 of Polygon Clipping:



First, start with left clipper,

Edge V_1 V_2 is in-to-out (case 3), hence, save intersection point V_1'.

Edge V_2 V_3 is out-to-in (case 1), hence, save intersection point V_2' and V_3.

Edge V_3 V_1 is in-to-in (case 2), hence, save V_1.

We now have polygon V_2' V_3 V_1 V_1' .

Next, start with bottom clipper,

Edge V_2' V_3 is in-to-in (case 2), hence, save V_3.

Edge V_3 V_1 is in-to-out (case 3), hence, save intersection point V_3'.

Edge V_1 V_1' is out-to-out (case 4), hence save none.

Edge V_1' V_2' is out-to-in (case 1), hence, save intersection point V_2'' and V_2'.

Now, we have clipped polygon, V_2'' V_2' V_3 V_3'.

Now, we are done.

3-D Transformations:

In a three-dimensional homogeneous coordinate representation, a point is translated (figure) from position P = *(x, y,* z) to position P' = *(x', y', z')* with the matrix operation

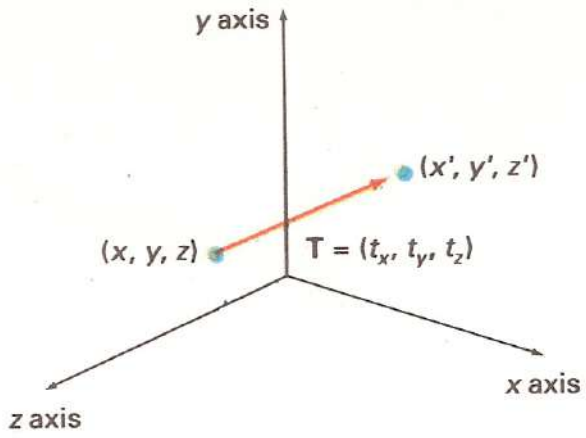
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

or

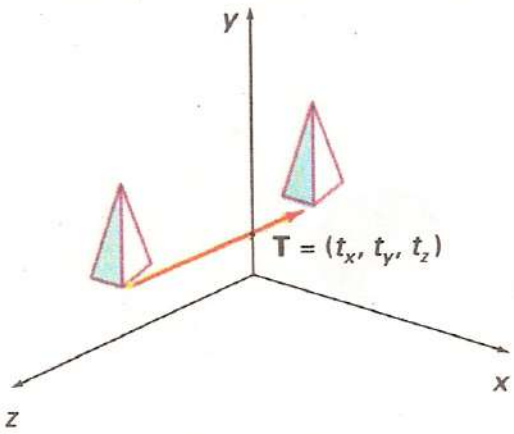
$$P' = T \cdot P$$

Parameters $t_x, t_y, and\ t_z$ specifying translation distances for the coordinate directions *x, y,* and z, are assigned any real values. The matrix representation is equivalent to the three equations:

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

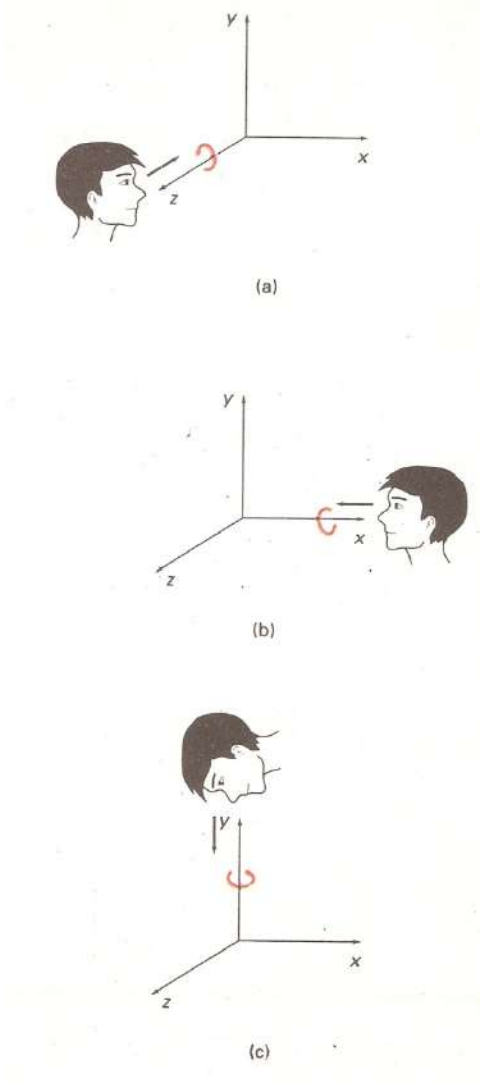


Translating a point with translation vector $\mathbf{T} = (t_x, t_y, t_z)$.



Translating an object with translation vector T.

3-Dimensional Coordinate Axis Rotation:



[In above figure, positive rotation directions about the coordinate axes are counterclockwise, when looking toward the origin from a positive coordinate position on each axis.]

Rotation about Z-axis:

In homogenous coordinate form, the three-dimensional z-axis rotation equations are expressed as:

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

which we can write more compactly as

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

Where

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

$y' = x \sin\theta + y \cos\theta$; ----------Equation (1)

$z' = z$;

Transformation equations for rotations about the other two coordinate axes can be obtained with a cyclic permutation of the coordinate parameters x, y, and z in equation (1).

That is, we use the replacements

$x \rightarrow y \rightarrow z \rightarrow x$ ---------- Equation (2)

Substituting permutations (Equation 2) in Equation (1), we get the equations for an **x-axis rotation**:

$y' = y \cos\theta - z \sin\theta$;

$z' = y \sin\theta + z \cos\theta$;

$x' = x$;

This can be written in homogenous form

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

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

Similarly, the matrix for **y-axis rotation** is:

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

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

General 3-Dimensional Rotations:

To rotate an object that is parallel to one of the coordinate axis:

To do so, we do the following:

1) Translate the object so that the rotation axis coincides with the parallel coordinate axis
2) Perform the specified rotation about that axis
3) Translate the object so that the rotation axis is moved back to its original position

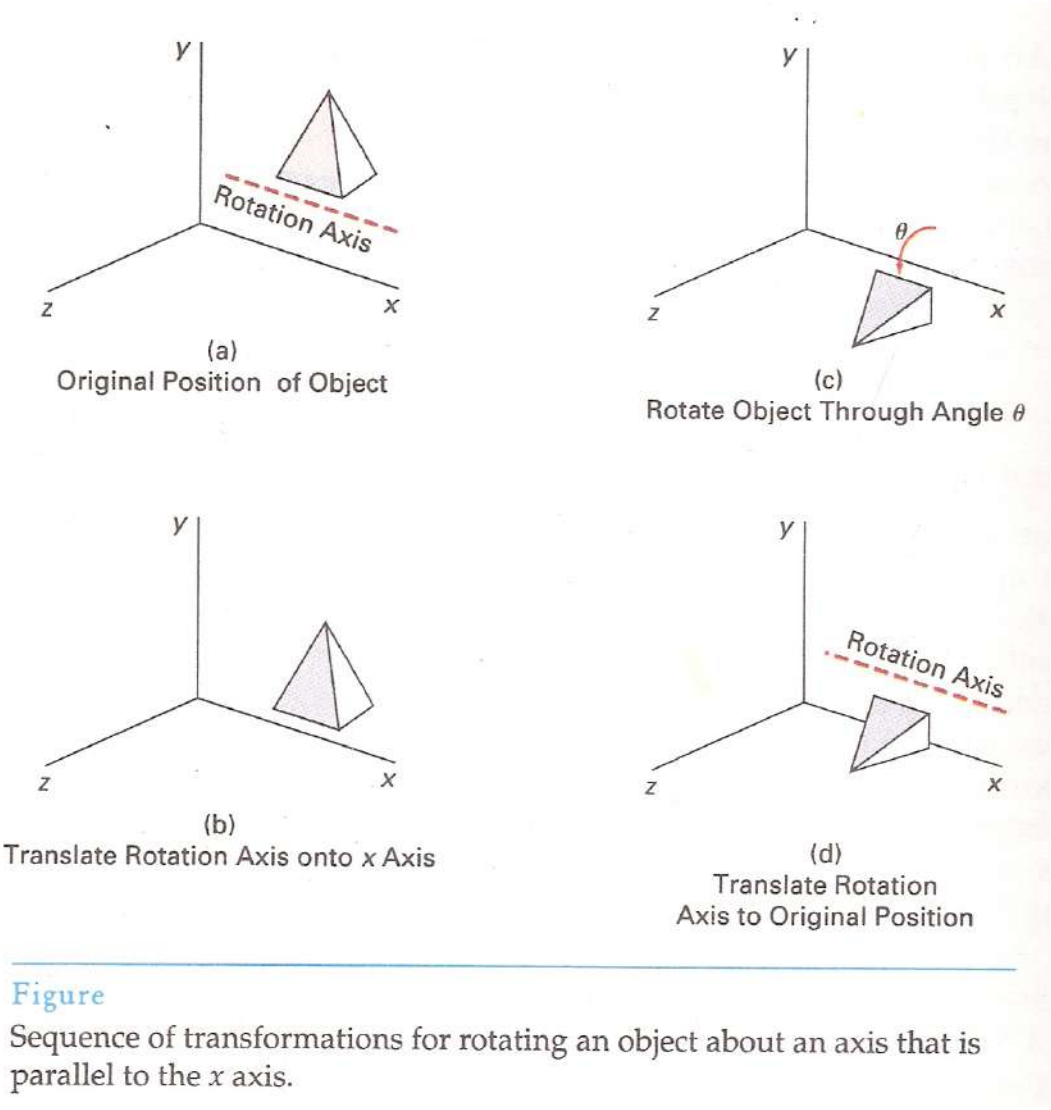


Figure
Sequence of transformations for rotating an object about an axis that is parallel to the x axis.

Any coordinate position P on the object in the above figure is transformed with the sequence shown as:

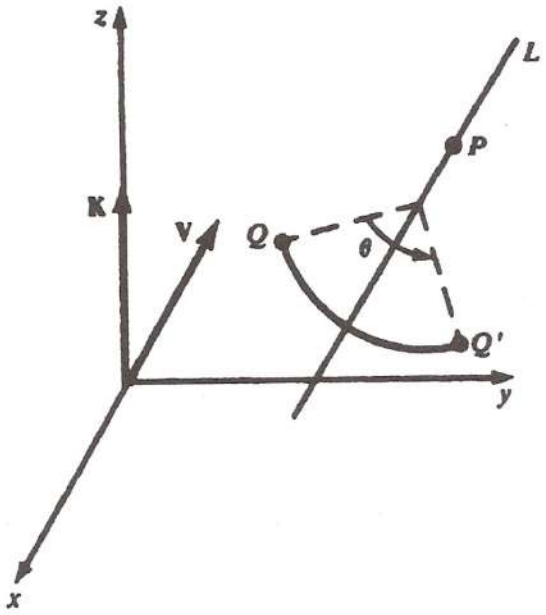
$$P' = T^{-1} . R_x(\theta) . T . P$$

Where the composite matrix for the transformation is

$$R(\theta) = T^{-1} . R_x(\theta) . T$$

To rotate an object about an axis that is not parallel to one of the coordinate axis:

Let an axis of rotation L be specified by a direction vector V and a location point P. To find the transformation for a rotation of θ about L.



We can find the required transformation by the following steps:

1) Translate P to the origin
2) Align V with the vector K (z-axis) using matrix A_V
3) Rotate by θ about K
4) Reverse steps 2 and 1.

This sequence of transformations can be represented in matrix form as:

$$R_{\theta,L} = T_{-P}^{-1}.A_V^{-1}.R_{\theta,K}.A_V.T_{-P}$$

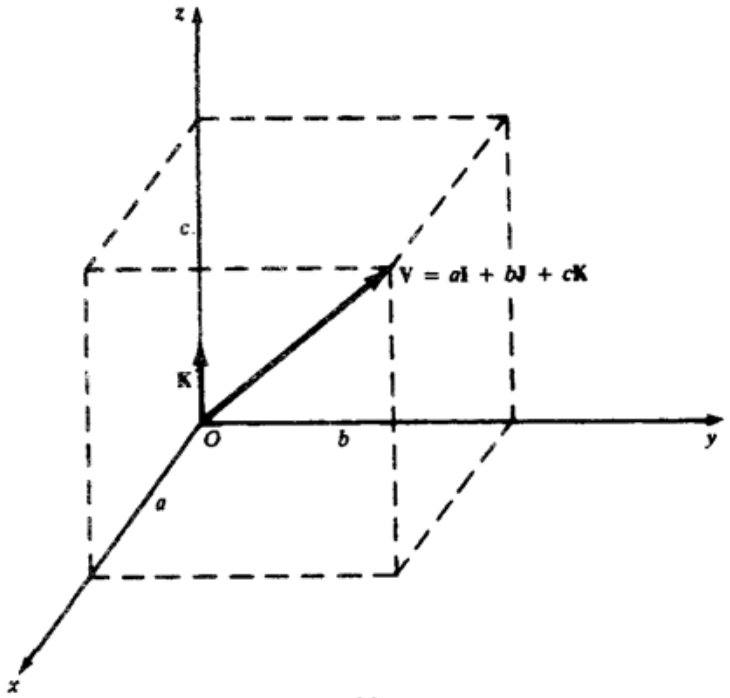
Now, to find a transformation A_V which aligns a given vector V along with the vector K along the positive z axis.

See figure (a) on next page. Let $V = aI + bJ + cK$. We perform the alignment through the following sequence of transformations [figures (b) and (c)].

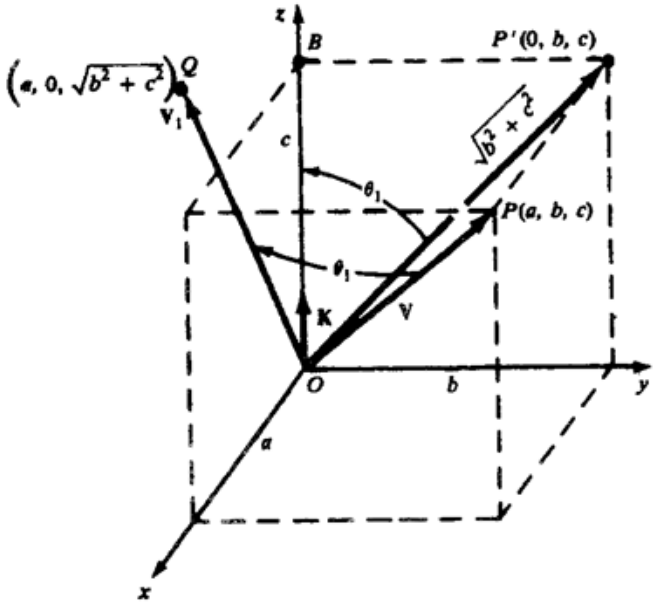
1) Rotate about the x-axis about an angle θ_1 so that V rotates into the upper half of the xz plane (as the vector V_1).
2) Rotate the vector V_1 about the y-axis by an angle $-\theta_2$ so that V_1 rotates to the positive z-axis (as the vector V_2).

Implementing step 1 from figure (b), we observe that the required angle of rotation θ_1 can be found by looking at the projection of V onto the yz plane. From triangle OP'B:

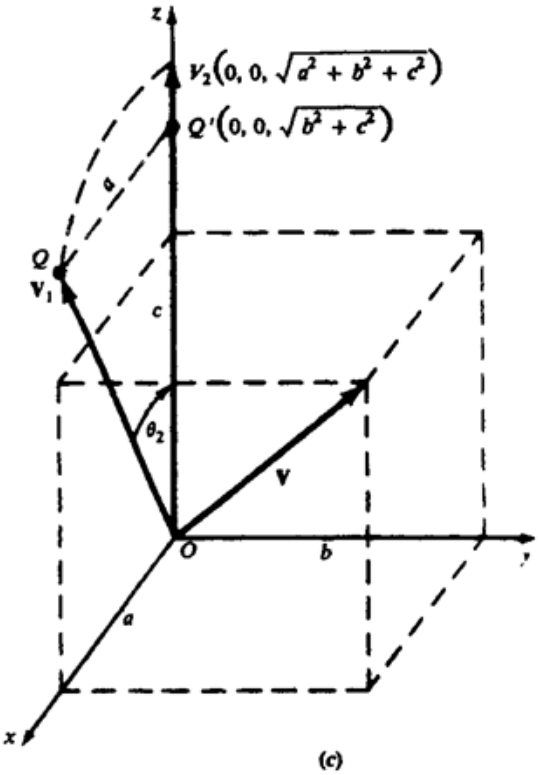
$$\sin\theta_1 = \frac{b}{\sqrt{b^2+c^2}} \qquad and \qquad \cos\theta_1 = \frac{c}{\sqrt{b^2+c^2}}$$


z
c
V = aI + bJ + cK
K
O
b
y
a
x

(a)


z
B
P'(0, b, c)
$(a, 0, \sqrt{b^2 + c^2})$
Q
V_1
c
$\sqrt{b^2 + c^2}$
θ_1
P(a, b, c)
θ_1
K
V
O
b
y
a
x

(b)


z
$V_2(0, 0, \sqrt{a^2 + b^2 + c^2})$
$Q'(0, 0, \sqrt{b^2 + c^2})$
Q
V_1
c
θ_2
V
O
b
a
x

(c)

The required rotation is:

$$R_{\theta_1,I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{\sqrt{b^2+c^2}} & -\frac{b}{\sqrt{b^2+c^2}} & 0 \\ 0 & \frac{b}{\sqrt{b^2+c^2}} & \frac{c}{\sqrt{b^2+c^2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Applying this rotation to the vector V produces the vector V_1 with the components $(a, 0, \sqrt{b^2+c^2})$.

Implementing step 2 from figure (c), we see that a rotation of $-\theta_2$ degrees is required, and so from triangle OQQ':

$$\sin(-\theta_2) = -\sin\theta_2 = -\frac{a}{\sqrt{a^2+b^2+c^2}}$$

and

$$\cos(-\theta_2) = \cos\theta_2 = \frac{\sqrt{b^2+c^2}}{\sqrt{a^2+b^2+c^2}}$$

Then,

$$R_{-\theta_2,J} = \begin{bmatrix} \frac{\sqrt{b^2+c^2}}{\sqrt{a^2+b^2+c^2}} & 0 & -\frac{a}{\sqrt{a^2+b^2+c^2}} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{a}{\sqrt{a^2+b^2+c^2}} & 0 & \frac{\sqrt{b^2+c^2}}{\sqrt{a^2+b^2+c^2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Since $|V| = \sqrt{a^2+b^2+c^2}$, and introducing the notation $\lambda = \sqrt{b^2+c^2}$, we find

$$A_V = R_{-\theta_2.J} \,.\, R_{\theta_1.I}$$

$$= \begin{bmatrix} \frac{\lambda}{|V|} & \frac{-ab}{\lambda|V|} & \frac{-ac}{\lambda|V|} & 0 \\ 0 & \frac{c}{\lambda} & \frac{-b}{\lambda} & 0 \\ \frac{a}{|V|} & \frac{b}{|V|} & \frac{c}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In the same manner we calculate the inverse transformation that aligns the vector K with the vector V:

$$A_V^{-1} = \left(R_{-\theta_2.J} \,.\, R_{\theta_1.I}\right)^{-1} = R_{\theta_1.I}^{-1} \,.\, R_{-\theta_2.J}^{-1} = R_{-\theta_1.I} \,.\, R_{\theta_2.J}$$

$$= \begin{bmatrix} \frac{\lambda}{|V|} & 0 & \frac{a}{|V|} & 0 \\ \frac{-ab}{\lambda|V|} & \frac{c}{\lambda} & \frac{b}{|V|} & 0 \\ \frac{-ac}{\lambda|V|} & \frac{-b}{\lambda} & \frac{c}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

So,

$$R_{\theta,L} = T_{-P}^{-1}.A_V^{-1}.R_{\theta,K}.A_V.T_{-P}$$

Example:

The pyramid defined by the coordinates A (0, 0, 0), B (1, 0, 0), C (0, 1, 0), and D (0, 0, 1) is rotated by 45 degrees about the line L that has the direction V = J + K and passing through point C (0, 1, 0). Find the coordinates of the rotated figure.

Solution:

The rotation matrix $R_{\theta,L}$ can be found by concatenating the matrices

$$R_{\theta,L} = T_{-P}^{-1}.A_V^{-1}.R_{\theta,K}.A_V.T_{-P}$$

With $(0,1,0)$, then

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

Now, V = J + K, so, we have a = 0, b = 1, c = 1, hence, $\lambda = \sqrt{b^2 + c^2} = \sqrt{2}$,

and $|V| = \sqrt{a^2 + b^2 + c^2} = \sqrt{2}$, and

$$A_V = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad A_V = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Also,

$$R_{45,K} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad T_{-P}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Then,

$R_{\theta,L} = T_{-P}^{-1}.A_V^{-1}.R_{\theta,K}.A_V.T_{-P}$

That is,

$$R_{\theta,L} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{2+\sqrt{2}}{4} & \frac{2-\sqrt{2}}{4} & \frac{2-\sqrt{2}}{4} \\ -\frac{1}{2} & \frac{2-\sqrt{2}}{4} & \frac{2+\sqrt{2}}{4} & \frac{\sqrt{2}-2}{4} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To find the coordinates of the rotated figure, we apply the rotation matrix $R_{\theta,L}$ to the matrix of homogenous coordinates of the vertices A, B, C, and D:

$$C = (ABCD) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

So,

$$R_{\theta,L}.C = \begin{bmatrix} \frac{1}{2} & \frac{1+\sqrt{2}}{2} & 0 & 1 \\ \frac{2-\sqrt{2}}{4} & \frac{4-\sqrt{2}}{4} & 1 & \frac{2-\sqrt{2}}{2} \\ \frac{2-\sqrt{2}}{4} & \frac{\sqrt{2}-4}{4} & 0 & \frac{\sqrt{2}}{2} \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

The rotated coordinates are:

A' = $\left(\frac{1}{2}, \frac{2-\sqrt{2}}{4}, \frac{2-\sqrt{2}}{4}\right)$ B' = $\left(\frac{1+\sqrt{2}}{2}, \frac{4-\sqrt{2}}{4}, \frac{\sqrt{2}-4}{4}\right)$ C' = (0, 1, 0) D' = $\left(1, \frac{2-\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right)$

We are done now.

Mumbai University Questions:

Dec 2007: With reference to 3D transformations, describe the steps to be carried out when an object is to be rotated about an axis that is not parallel to any of the coordinate axis. Specify all the required matrices. State your assumptions clearly.

Solution: Refer to the topic **To rotate an object about an axis that is not parallel to one of the coordinate axis** on page 25.

Light Shading:

Introduction:

An illumination model, also called a lighting model or shading model, is used to calculate the intensity of light that we should see at a given point on the surface of an object.

A surface rendering algorithm uses the intensity calculations from an illumination model to determine the light intensity for all projected pixel positions for the various surfaces in a scene.

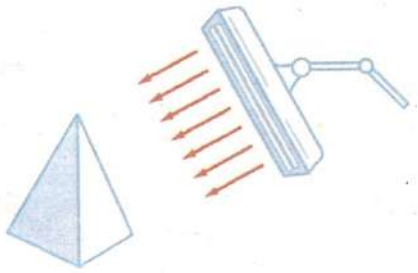
Light sources:

Two types:

- **Point Source**: Rays from the source follow radially diverging paths from the source position. For example, a bulb or the Sun, which are sufficiently far from the scene.

Figure
Diverging ray paths from a point light source.

- **Distributed Light Source**: A nearby source, such as the long fluorescent light.

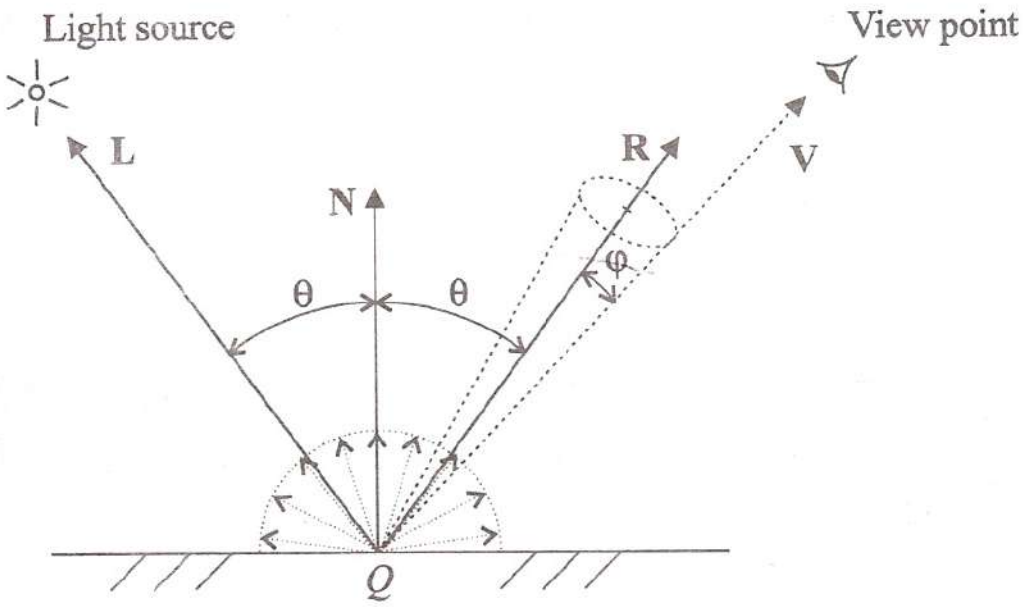


When light is incident on an opaque surface, part of it is reflected and part is absorbed. Shiny materials reflect more light, and dull surfaces absorb more of the incident light.

Surfaces that are rough, or grainy, tend to scatter the reflected light in all directions. This scattered light is called **diffuse reflection**. What we call the color of an object is the color of the diffuse reflection of the object. A blue object illuminated by a white light source reflects the blue component of the white light and totally absorbs all other components. If the blue object is viewed under a red light, it appears black since all of the incident light is absorbed.

Ambient Light: is a combination of light reflections from various surfaces, to produce a uniform illumination. The amount of ambient light incident on each object is a constant for all surfaces and over all surfaces. But the intensity of reflected light for each surface depends on the optical properties of the surface; that is, how much of the incident energy is to be reflected and how much absorbed.

The Phong Illumination Model:



This is a widely used and highly effective way to mimic the reflection of light from object to the viewer's eye. Now consider a point light source (see above figure), which is an idealized light with all its energy coming out from a single point in space. Our eye is at the viewpoint looking at point Q on the surface of the object. What should be the color of the light reflected into our eye from Q (in the direction of vector V)?

There are two extreme cases of light reflection:-

1) **Diffuse reflection**: In this case light energy from the light source (in the direction of $-L$) gets reflected/bounced off equally in all directions (see the small arrows forming a half-circle/hemisphere). We also want the energy level of the reflection to be a function of the incident angle θ (between L and surface normal N). The smaller the angle, the higher the reflection. The mathematical tool we use to achieve these is to have the reflection proportional to $\cos\theta$. Diffuse reflections are constant over each surface in a scene, independent of the viewing direction. The fractional amount of the incident light that is diffusely reflected can be set for each surface with parameter k_d, the diffuse-reflection coefficient. We have $0 \leq k_d \leq 1$. If k_d = 1 then, highly reflective; if k_d = 0, then, full absorption.
2) **Specular reflection**: When we look at an illuminated shiny surface, such as polished metal, an apple, or a person's forehead, we see a highlight, or bright spot, at certain viewing directions. This phenomenon, called specular reflection, is the result of total, or near total, reflection of the incident light in a concentrated region around the specular-reflection angle. In the figure, the reflected energy is distributed across a small cone-shaped space centered around R, with the reflection being the strongest along the direction of R (i.e., φ = 0) and decreasing quickly as φ increases. The mathematical means for modeling this effect is $cos^k(\varphi)$, where the parameter k is a measure of the degree of shininess (k = 1 for a dull surface and k = 100 for a mirror-like surface). For a given scene we can find out the amount of specular reflection in the direction of V using the actual angle φ between R and V.

Thus, in the Phong Illumination model, object surfaces are thought to produce a combination of ambient-light reflection and light-source-dependent diffuse/specular reflection. Mathematically, we can state the total reflected energy intensity I as:

$$I = I_a k_a + I_p(k_d \cos(\theta) + k_s \cos^k(\varphi))$$

where

I_a = Intensity of ambient light,

I_p = Intensity of point light,

k_a = Ambient light reflection coefficient,

k_d = Diffuse reflection coefficient,

k_s = Specular reflection coefficient, and,

$0 \leq k_a, k_d, k_s \leq 1$

If L, N, R, and V are unit vectors, then dot product of L and N = $L.N = \cos\theta$ and $.V = \cos\varphi$. Hence, we have:

$$I = I_a k_a + I_p(k_d L.N + k_s(R.V)^k)$$

For two or more light sources, effects are cumulative:

$$I = I_a k_a + \sum_{i=1}^{n} I_{p_i}(k_d L_i.N + k_s(R_i.V)^k)$$

Polygon-Rendering Methods:

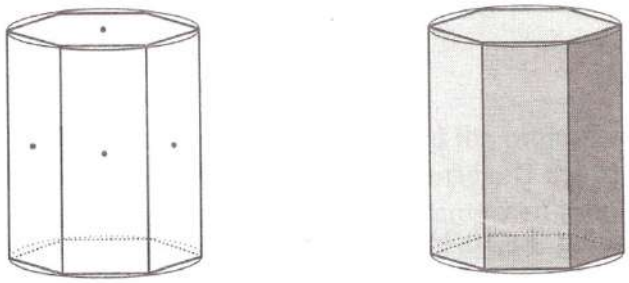
Constant Intensity Shading:

A fast and simple method for rendering an object with polygon surfaces is constant-intensity shading, also called flat shading. In this method, a single intensity is calculated for each polygon. All points over the surface of the polygon are then displayed with the same intensity value. In general, flat shading of polygon facets provides an accurate rendering for an object if all of the following assumptions are valid:

- The object is a polyhedron and is not an approximation of an object with a curved surface.
- All light sources illuminating the object are sufficiently far from the surface so that N.L is constant over the surface.
- The viewing position is sufficiently far from the surface so that V.R is constant over the surface.

Example of an approximation of a curved surface:

A cylindrical object may be modeled by a polygon mesh as shown in figure (a). We can evaluate the Phong formula using the geometric center of each polygon and set the resultant color to every pixel that belongs to the corresponding polygon (see figure (b)).



(a) (b)

Constant shading can produce good results for dull polyhedrons lit by light sources that are relatively far away. The primary weakness of this method is that false contours appear between adjacent polygons that approximate a curved surface (see figure (b)). In addition, the specular reflection of the light source tends to get lost.

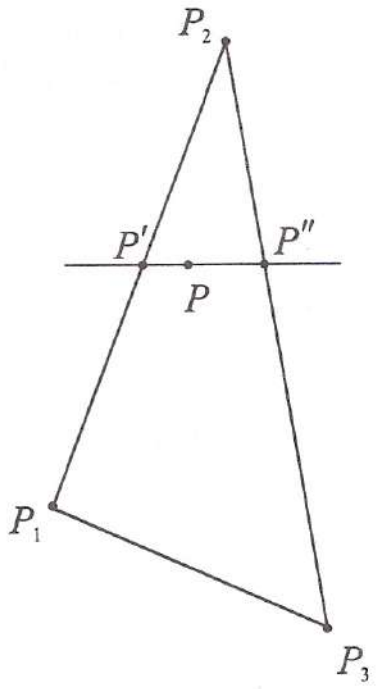
Gouraud Shading:

This intensity-interpolation scheme, developed by Gouraud, renders a polygon surface by linearly interpolating intensity values across the surface. Intensity values for each polygon are matched with the values of adjacent polygons along the common edges, thus eliminating the intensity discontinuities that can occur in flat shading.

Each polygon surface is rendered with Gouraud shading by performing the following calculations:

- Determine the average unit normal vector at each polygon vertex.
- Apply an illumination model to each vertex to calculate the vertex intensity.
- Linearly interpolate the vertex intensities over the surface of the polygon.

For example, we use bilinear interpolation to find the color of point P inside a triangle whose vertices are P_1, P_2, and P_3 (see below figure).



The scan line containing P intersects the two edges at points P' and P''. We interpolate the color of P_1 and the color of P_2 to get the color of P'. We then interpolate the color of P_2 and the color of P_3 to get the color of P''. Finally, we interpolate the color of P' and the color of P'' to get the color of P.

Let the coordinates and intensity be:

P(x, y) with intensity I,

$P_1(x_1, y_1)$ with intensity I_1,

$P_2(x_2, y_2)$ with intensity I_2,

$P_3(x_3, y_3)$ with intensity I_3,

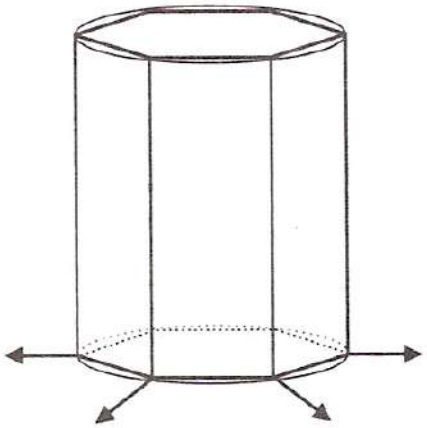
P'(x', y') with intensity I',

P''(x'', y'') with intensity I''.

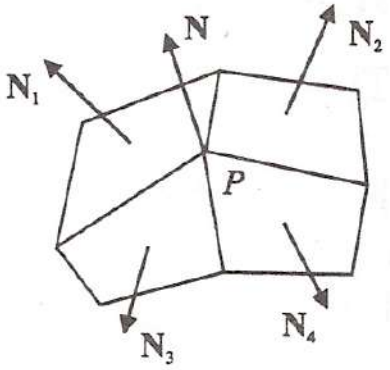
We have:

$$I' = I_1 \frac{y_2 - y'}{y_2 - y_1} + I_2 \frac{y' - y_1}{y_2 - y_1} \qquad and \qquad I'' = I_2 \frac{y'' - y_3}{y_2 - y_3} + I_3 \frac{y_2 - y''}{y_2 - y_3}$$

$$I = I' \frac{x'' - x}{x'' - x'} + I'' \frac{x - x'}{x'' - x'}$$



(a)



(b)

To decide the normal vectors at the vertices is to average the normal vectors of adjacent polygons. For example, to determine the normal vector N at vertex P in figure (b), we use the average of the normal vectors of the polygon that meet at P:

$N = N_1 + N_2 + N_3 + N_4$

where N is then normalized by dividing it by |N|.

Gouraud shading removes the intensity discontinuities associated with the constant-shading model, but it has some other deficiencies. Highlights on the surface are sometimes displayed with anomalous shapes, and the linear intensity interpolation can cause bright or dark intensity streaks, called Mach bands, to appear on the surface. These effects can be reduced by dividing the surface into a greater number of polygon faces or by using other methods, such as Phong shading, that require more calculations.

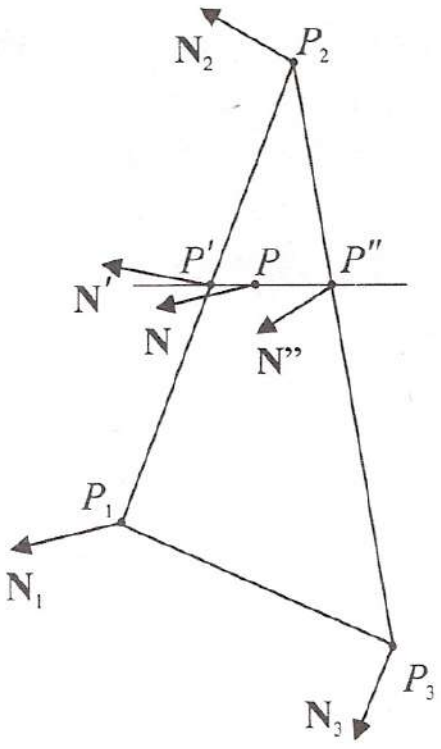
Phong Shading:

A more accurate method for rendering a polygon surface is to interpolate normal vectors, and then apply the illumination model to each surface point. This method is called Phong shading or normal-vector interpolation shading. It displays more realistic highlights on a surface and greatly reduces the Mach-band effect.

A polygon surface is rendered using Phong shading by carrying out the following steps:

- Determine the average unit normal vector 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.

To find the normal vector N at point P in a triangle (see figure), we first interpolate N_1 and N_2 to get N', and then interpolate N_2 and N_3 to get N'', and finally interpolate N' and N'' to get N.



This technique is relatively time-consuming since the illumination model is evaluated at every point of interest using the interpolated normal vectors. However, it is very effective in dealing with specular highlights.

Splines, Bezier Curves, B-Splines:

Splines:

In drafting terminology, a spline is a flexible strip used to produce a smooth curve through a designated set of points. Mathematically, a spline is a piecewise cubic polynomial function whose first and second derivatives are continuous across the various curve sections.

In computer graphics, the term spline curve refers to any composite curve formed with polynomial sections satisfying continuity conditions at the boundary of the pieces. A spline surface can be described with two sets of orthogonal spline curves.

Interpolation and Approximation Splines:

We specify a spline curve by giving a set of coordinate positions, called control points, which indicate the general shape of the curve. These control points are then fitted with piecewise continuous parametric polynomial functions in one of two ways. When polynomial sections are fitted so that the curve passes through each control point as in the first figure, the resulting curve is said to interpolate the set of control points.

Figure
A set of six control points interpolated with piecewise continuous polynomial sections.

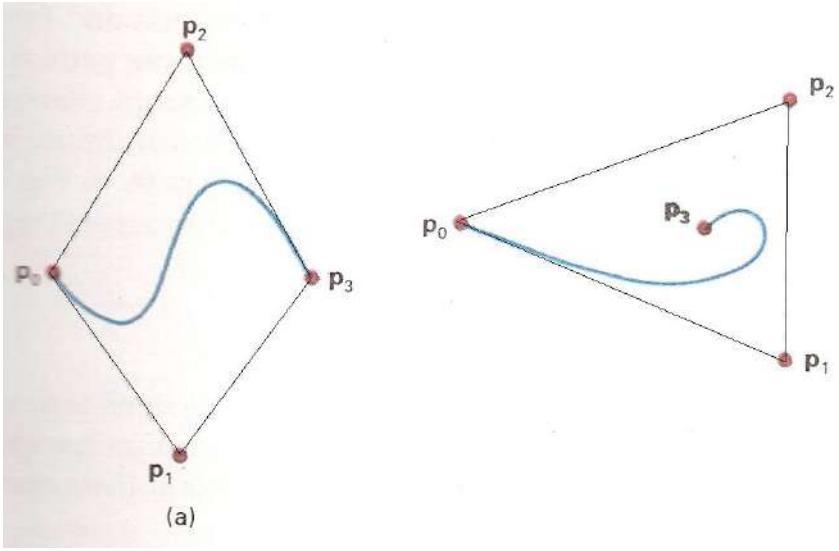
Figure
A set of six control points approximated with piecewise continuous polynomial sections.

On the other hand, when the polynomials are fitted to the general control-point path without necessarily passing through any control point, the resulting curve is said to approximate the set of control points (see second figure).

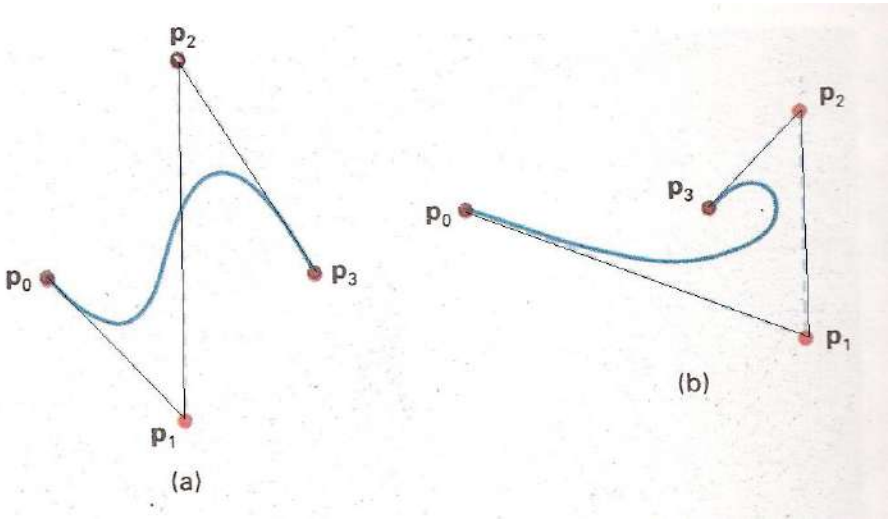
We will now deal with only approximation splines, mainly the Bezier curves and the B-splines.

A spline curve is defined, modified, and manipulated with operations on the control points. The curve can be translated, rotated, or scaled with transformations applied to the control points.

The convex polygon boundary that encloses a set of control points is called the **convex hull** (see below figure).



A polyline connecting the sequence of control points for an approximation spline is usually referred to as the **control graph** of the curve. Other names for the control graph are **control polygon** and **characteristic polygon** (see below figure).



Parametric Continuity Conditions:

To ensure a smooth transition from one section of a piecewise parametric curve to the next, we can impose various continuity conditions at the connection points. If each section of a spline is described with a set of parametric coordinate functions of the form

$$x = x(u), \quad y = y(u), \quad z = z(u), \qquad u_1 \leq u \leq u_2$$

we set parametric continuity by matching the parametric derivatives of adjoining curve sections at their common boundary.

Zero-order parametric continuity, described as C^0 continuity, means simply that the curves meet. That is the values of x, y, and z evaluated at u_2 for the first curve section are equal, respectively, to the values of x, y, and z evaluated at u_1 for the next curve section.

First-order parametric continuity, referred to as C^1 continuity, means that the first parametric derivatives (tangent lines) of the coordinate functions for two successive curve sections are equal at their joining point.

Second-order parametric continuity, or C^2 continuity, means that both the first and second parametric derivatives of the two curve sections are same at the intersection. With second-order continuity, the rates of change of the tangent vectors for connecting sections are equal at their intersection. Thus, the tangent line transitions smoothly from one section of the curve to the next.

A camera travelling along the curve path in figure (b) with equal steps in parameter u would experience an abrupt change in acceleration at the boundary of the two sections, producing a discontinuity in the motion sequence. But if the camera were traveling along the path in figure (c), the frame sequence for the motion would smoothly transition across the boundary.

Spline Specifications:

There are three equivalent methods for specifying a particular spline representation:

1) We can state the set of boundary conditions that are imposed on the spline; or
2) We can state the matrix that characterizes the spline; or
3) We can state the set of blending functions that determine how specified geometric constraints on the curve are combined to calculate positions along the curve path.

For example, suppose we have the following parametric cubic polynomial representation for the x-coordinate along the path of a spline section:

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x \,, \qquad 0 \le u \le 1$$

Method 1: Boundary conditions for this curve might be set on the endpoint coordinates x(0) and x(1) and on the parametric first derivatives at the endpoints x'(0) and x'(1). These four boundary conditions are sufficient to determine the values of the four coefficients $a_x, b_x, c_x, and\ d_x$.

Method 2: From the boundary conditions we obtain the matrix that characterizes this spline curve by rewriting the above equation as the matrix product:

$$x(u) = [u^3 \quad u^2 \quad u \quad 1] \cdot \begin{bmatrix} a_x \\ b_x \\ c_x \\ d_x \end{bmatrix}$$

$$= U \cdot C$$

$$= U \cdot M_{spline} \cdot M_{geom}$$

where $C = M_{spline} \cdot M_{geom}$

where M_{geom} is a four element column matrix containing the geometric constraint values (boundary conditions) on the spline, and

M_{spline} is the 4-by-4 matrix that transforms the geometric constraint values to the polynomial coefficients and provides a characterization for the spline curve.

Method 3:

Finally, we can write the above equation as a polynomial representation for coordinate x in terms of the geometric constraint parameters

$$x(u) = \sum_{k=0}^{3} g_k . BF_k(u)$$

Where g_k are the constraint parameters, such as the control-point coordinates and slope of the curve at the given points, and $BF_k(u)$ are the polynomial blending functions.

If the blending functions are Bernstein polynomials, then, we get a Bezier curve, and if the blending functions are basis functions, then, we get B-splines.

Bezier Curves:

This spline approximation method was developed by the French engineer **Pierre Bezier** for use in the design of Renault automobile bodies. Bezier splines have a number of properties that make them highly useful and convenient for curve and surface design. They are also easy to implement. For these reasons, Bezier splines are widely available in various CAD systems and in general graphics packages (such as GL on Silicon Graphics systems).

A Bezier curve can be fitted to any number of control points. Suppose we are given n+1 control-point positions: $p_k = (x_k, y_k, z_k)$ with k varying from 0 to n. These coordinate points can be blended to produce the following position vector P(u), which describes the path of an approximating Bezier polynomial function between p_0 and p_n .

$$P(u) = \sum_{k=0}^{n} p_k . BEZ_{k,n}(u) \qquad 0 \le u \le 1$$

The Bezier blending functions $BEZ_{k,n}(u)$ are the **Bernstein polynomials**:

$$BEZ_{k,n}(u) = C(n,k).u^k.(1-u)^{n-k}$$

where the C(n, k) are the binomial coefficients:

$$C(n,k) = \frac{n!}{k!\,(n-k)!}$$

Equivalently, we can define Bezier blending functions with the recursive calculation:

$$BEZ_{k,n}(u) = (1-u)BEZ_{k,n-1}(u) + u\,BEZ_{k-1,n-1}(u) \qquad n > k \ge 1$$

Above vector equation represents a set of three parametric equations for the individual curve coordinates:

$$x(u) = \sum_{k=0}^{n} x_k . BEZ_{k,n}(u)$$

$$y(u) = \sum_{k=0}^{n} y_k . BEZ_{k,n}(u)$$

$$z(u) = \sum_{k=0}^{n} z_k . BEZ_{k,n}(u)$$

As a rule, a Bezier curve is a polynomial of degree one less than the number of control points used: Three points generate a parabola, four points generate a cubic curve, and so forth. Below figure demonstrates the appearance of some Bezier curves for various selections of control points in the xy plane.

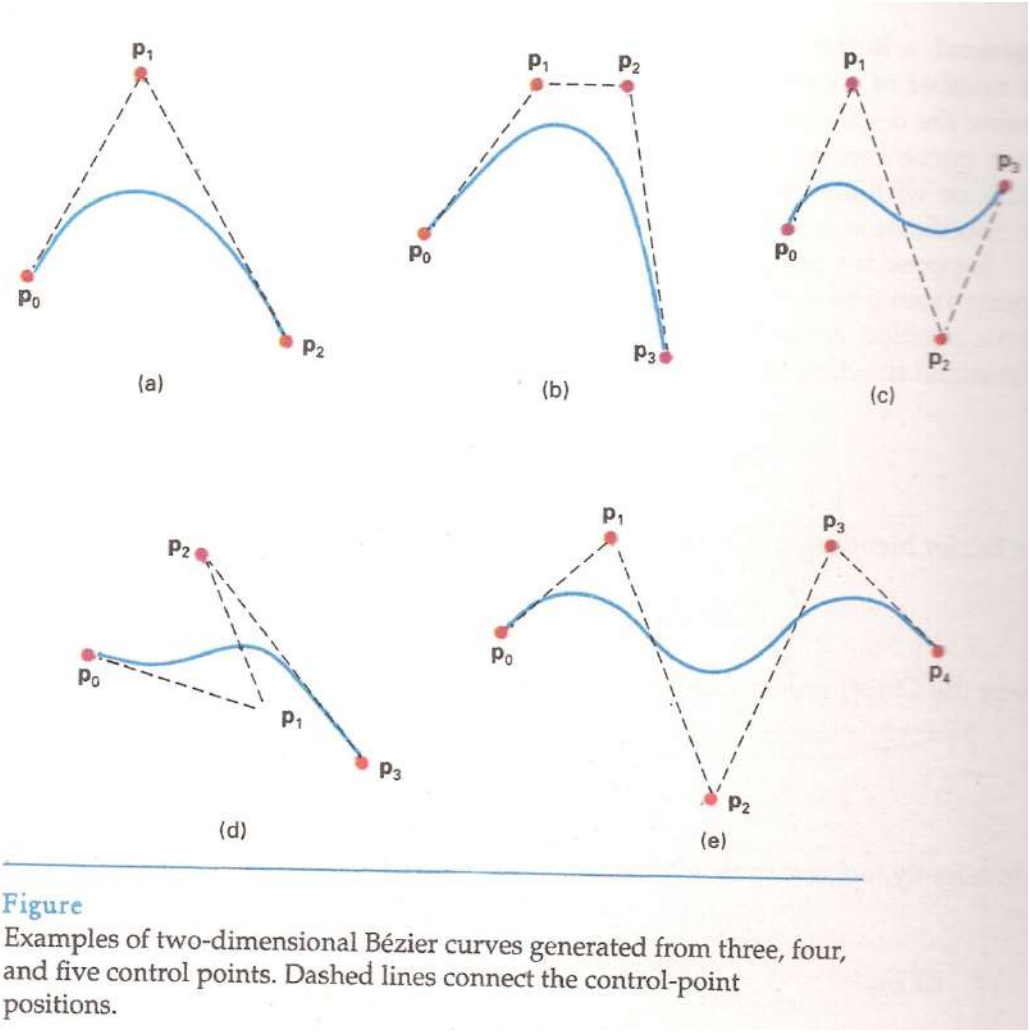


Figure
Examples of two-dimensional Bézier curves generated from three, four, and five control points. Dashed lines connect the control-point positions.

Properties of Bezier curves:

1) The Bezier curve always passes through the first and last control points. That is, the boundary conditions at the two ends of the curve are P(0) = p_0 and P(1) = p_n.
2) The slope at the beginning of the curve is along the first two control points, and the slope at the end of the curve is along the line joining the last two endpoints.
3) All Bezier curves lie within the convex hull of the control points.
4) The blending functions are all positive and their sum is always 1, i.e.,$\sum_{k=0}^{n} BEZ_{k,n}(u) = 1$
5) The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points without erratic oscillations.
6) The degree of the Bezier polynomial is one less than the number of control points used.

Bezier Surfaces:

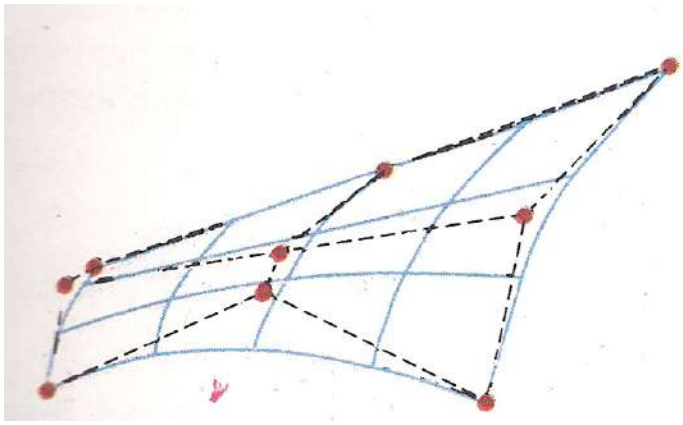
Two sets of orthogonal Bezier curves can be used to design an object surface by specifying an input mesh of control points.

The parametric vector function for the Bezier surface is formed as the Cartesian product of Bezier blending functions:

$$P(u,v) = \sum_{j=0}^{m}\sum_{k-0}^{n} p_{j,k} BEZ_{j,m}(v) BEZ_{k,n}(u)$$

with $p_{j,k}$ specifying the location of the (m + 1) by (n + 1) control points.

Below figure illustrates a Bezier surface plot. The control points are connected by dashed lines, and the solid lines show curves of constant u and constant v. Each curve of constant u is plotted by varying v over the interval from 0 to 1, with u fixed at one of the values in this unit interval.



Bezier surfaces have the same properties as Bezier curves, and they provide a convenient method for interactive design applications.

B-splines:

B-splines are the most widely used class of approximating splines. B-splines have two advantages over Bezier splines:

1) The degree of a B-spline polynomial can be set independently of the number of control points (with certain limitations),
2) B-splines allow local control over the shape of a spline curve or surface.

The only disadvantage is that B-splines are more complex than Bezier splines.

We can write a general expression for the calculation of coordinate positions along a B-spline curve in a blending function formulation as

$$P(u) = \sum_{k=0}^{n} p_k B_{k,d}(u) \qquad u_{min} \le u \le u_{max}, \qquad 2 \le d \le n+1$$

where the p_k are an input set of n + 1 control points.

Blending functions for B-spline curves are defined by the **Cox-deBoor** recursion formulas:

$$B_{k,1} = \begin{cases} 1, & if\ u_k \le u \le u_{k+1} \\ 0, & otherwise, \end{cases}$$

$$B_{k,d}(u) = \frac{u - u_k}{u_{k+d-1} - u_k} B_{k,d-1}(u) + \frac{u_{k+d} - u}{u_{k+d} - u_{k+1}} B_{k+1,d-1}(u)$$

Where each blending function is defined over d subintervals of the total range of u. The selected set of subinterval endpoints u_j is referred to as the knot vector. We can choose any values for the subinterval endpoints satisfying the relation $u_j \le u_{j+1}$. Values for u_{min} and u_{max} then depend on the number of control points we select, the value we choose for parameter d, and how we set up the subintervals (knot vector).

Properties of B-spline curves:

1) The polynomial curve has degree $d - 1$ and C^{d-2} continuity over the range of u.
2) For (n + 1) control points, the curve is described with n + 1 blending functions.
3) Each blending function $B_{k,d}$ is defined over d subintervals of the total range of u, starting at knot value u_k.
4) The range of parameter u is divided into (n + d) subintervals by the (n + d + 1) values specified in the knot vector.
5) Each section of the spline curve (between two successive knot values) is influenced by d control points.
6) Any one control point can affect the shape of at most d curve sections.

B-spline Surfaces:

Formulation of a B-spline surface is similar to that for Bezier splines. We can obtain a vector point function over a B-spline surface using the Cartesian product of B-spline blending functions in the form:

$$P(u,v) = \sum_{k_1=0}^{n_1} \sum_{k_2=0}^{n_2} p_{k_1,k_2} B_{k_1,d_1}(u) B_{k_2,d_2}(v)$$

Where the vector values for p_{k_1,k_2} specify positions of the (n_1 + 1) by (n_2 +1) control points.

B-spline surfaces exhibit the same properties as those of their component B-spline curves. A surface can be constructed from selected values for parameters d_1 and d_2 (which determine the polynomial degrees to be used) and from the specified knot vector.