Date: 25.4.23

Chapter 1
A survey of computer Graphics

CAD

- CAD (computer Aided design) is the use of computer-based software to aid in design processes.

- CAD software is frequently used by different types of engineers and designers. CAD software can be used to create two-dimensional (2D) drawings or 3D models.

- CAD allows experts to create more accurate design representations. CAD replaced manual design drafting, allowing design development, alternation and optimization.

- CAD enables engineers to craft more precise designs and manipulate them virtually. CAD software calculates how multiple materials relate.

- Some types of CAD :

popular types of CAD software companies use include Solid-works, Inventor, Revit, AutoCAD, Civil 3D etc.

Computer Graphics :

- is the use of computers to display and manipulate information in graphical or pictorial form, either on a visual-display unit or via a printer or plotter.

Application of computer graphics :

- Computer Aided Design
- presentation graphics (chart/graphics)
- computer Art (3D/2D/illustrator/Photoshop)
- Entertainment (motion graphics, gaming)
- Education and training (training in Aircraft, shipping)
- Visualization (data in graphical representation)
- Image processing (image editing)
- GUI (Graphical user interface)

CAM

- computer Aided Manufacturing software prepares a model for machining by working through several actions including: checking if the model has any geometry errors that will impact the manufacturing process.

- Typically it uses software to translate drawings and data into detailed instructions that can drive some sort of automated tool. For example, a 2D digital drawing can be used to guide a laser or physical cutting tool to cut cladding or other components to fit an architect's design.

CAT

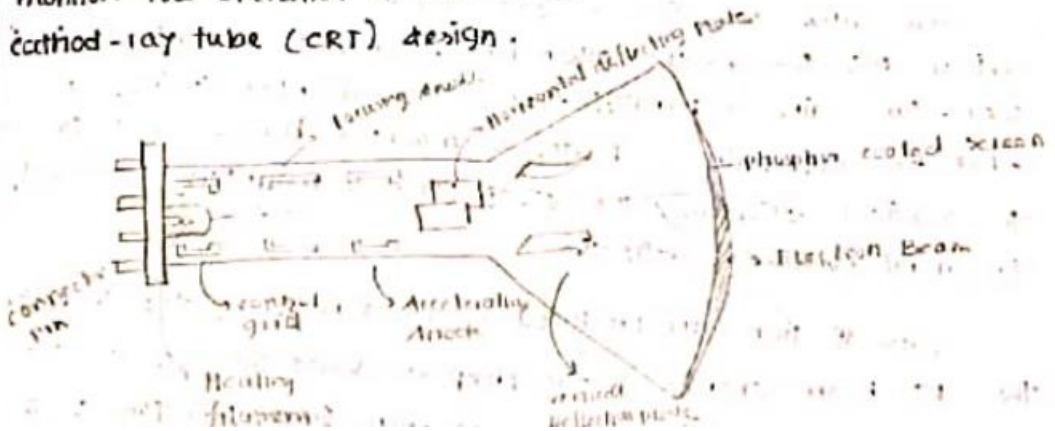
- CAT (Computer Assisted Typesetter) is a highly addressable phototypesetter with full optics control from computer-generated data. Data is normally transmitted to the CAT by paper tape.

- ~~photosetting~~ photolypesetting is most often used with offset printing technology.

Chapter 2

CRT (Cathod Ray tube): → most desktop computer displays make use of CRTs.

- Typically, the primary output device in a graphics system is a video monitor. The operation of most video monitors is based on the standard cathod-ray tube (CRT) design.



- A beam of electrons (cathode rays), emitted by an electron gun, passes through focusing and deflection systems that direct the beam toward specified positions on the phosphor-coated screen. The phosphor then emits a small spot of light at each position contacted by the electron beam.

heated metal cathode
control grid

- The primary components of an electron gun in a CRT are the heated metal cathode and a control grid. Heat is supplied to the cathode by directing a current through a coil of wire, called the filament, inside the cylindrical cathode structure. This causes electrons

to be "boiled off" the hot cathode surface. In the vaccum inside the cylindrical cathode structure CRT envelope, the free, negatively charged electrons are then accelerated toward the phosphor coating by a high positive voltage. Sometimes the electron gun is built to contain the accelerating anode and focusing system within the same unit. The focusing system in a CRT is needed to force the electron beam to converge into a small spot as it strikes the phosphor; Otherwise, the electrons would repel each other, and the beam would spread out as it approaches the screen.

- Horizontal deflection is accomplished with one pair of coils, and verfical deflection by the other pair. When electrostatic deflection is used, two pairs of parallel plates are mounted inside the CRT envelope. One pair of plates is mounted horizontally to control the verfical deflection, the other pair is mounted vertically to control horizontal deflection.

- Spots of light are produced on the screen by the transfer of the CRT beam energy to the phosphor. What we see on the screen is the combined effect of all electron light emissions: a glowing spot that quickly fades after all the excited phosphor electrons have returned to their ground energy level.

persistence: is defined as the time it takes the emitted light from the screen to decay to one-tenth of its original intensity.

low persistence phosphors require higher refresh rate to maintain a picture on the screen without flicker. (animation)

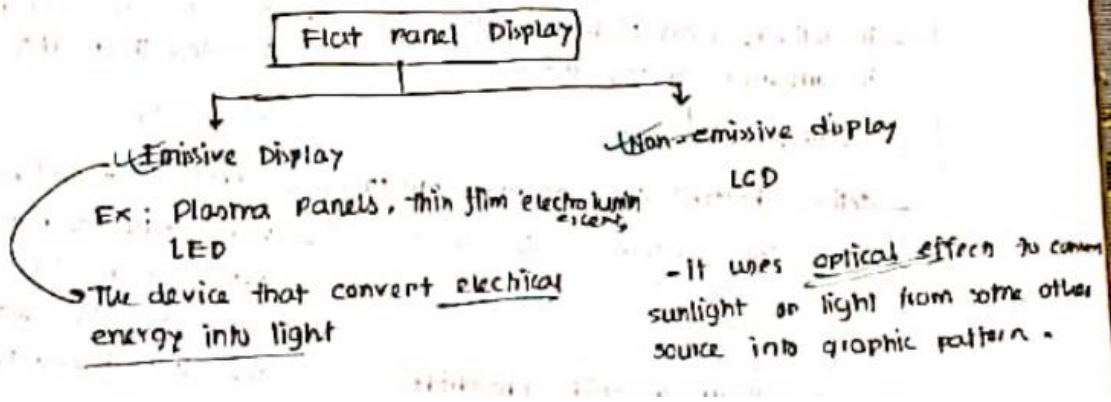
High-persistence phosphor: for displaying highly complex, static pictures.

10 to 60 microsec

Flat Panel Display

- It refers to a class of video device that have reduced volume, weight and power requirement compare to CRT.

Example: Small TV monitor, calculator, pocket video games, laptop computer and advertisement board in elevator.



LCD (Liquid crystal Display)

- is a flat display screen used in electronic device such as laptop, computer, TV, cellphone and portable video games.

~~construction of LCD~~

LED (light-emitting diode) Display

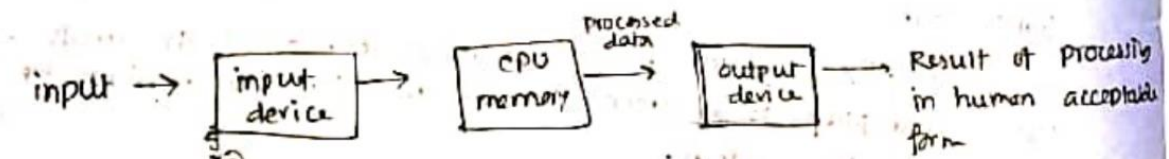
- is a screen display technology that uses a panel of LEDs as the light source.
- currently, a large no. of electronic devices, both small and large, use LED display as a screen and as an interaction medium between the user and the system.

LED	LCD
- LED has a better response time than LCD.	- slower than LED
- LED consumes more power in comparison to LCD.	- consumes less power
- it delivers good picture quality in comparison to LCD display	- also delivers good picture quality but less than LED.
- costlier than LCD	- less costly
- delivers better color accuracy than LCD	- good color accuracy
- has wider viewing angle than LCD.	- wide-angle decreases with 30 degrees from the center in the image then the contrast ratio.
- uses gallium arsenide phosphide	- uses liquid crystals and glass electrons.

Input device:

- are the hardware that is used to transfer input to the computer
- Data can be in the form of text, graphics, sound.

- output device display data from memory of the computer
- output can be text, numeric data, line, polygon and other objects.



Input devices
- keyboard
- Mouse
- Trackball } 2D
- spaceball - 6D
- joystick
- Image scanner
- Touch screen
- light pen

output devices
- printer
- plotter

Data-glove:

- is an input device that is essentially a glove worn on the hand that contains various electronic sensors that monitor the hand's movement and transform them into a form of input for applications such as virtual reality and robotics.

To provide input to virtual reality systems.

Date: 25.9.23

Chapter 3

Output primitives

line → set of points over a path

Line-drawing Algo

Straight line eq ⇒ $y = mx + b$ (m → slope, b → y intercept)

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

$b = y_1 - m x_1$

$\Delta y = m \Delta x$

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

DDA Algorithm (Digital differential Analyzer)

– is a scan-conversion line algorithm based on calculating either Δy or Δx.

Slope, $m = \frac{\Delta y}{\Delta x} = \tan\theta$

$\frac{\Delta y}{\Delta x} > 1$

$\frac{\Delta y}{\Delta x} < 1$

$m < 1$

(x_1, y_1) (x_2, y_2)

$(5, 9)$ $(12, 7)$

$\Delta x = 12 - 5 = 7$

$\Delta y = 7 - 9 = -3$

$m = \frac{\Delta y}{\Delta x} = \frac{-3}{7}$ steps = 7 $(\Delta x > \Delta y)$

$x_{inc} = \frac{7}{7} = 1$ $y_{inc} = \frac{-3}{7} = -0.4$

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

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

x	5	6	7	8	9	10	11	12
y	9	9.4	9.8	5.2	5.6	6	6.4	6.8

⇒ $m > 1$

(x_1, y_1) (x_2, y_2)

$(5, 7)$ $(10, 15)$

$\Delta x = 10 - 5 = 5$

$\Delta y = 15 - 7 = 8$

steps = 8 $(\Delta y > \Delta x)$

⇒ $m = 1$

(x_1, y_1) (x_2, y_2)

$(12, 9)$ $(17, 14)$

$\Delta x = 17 - 12 = 5$

$\Delta y = 14 - 9 = 5$

$m = \frac{5}{5} = 1$ steps = 5

$x_{inc} = \frac{5}{5} = 1$ $y_{inc} = \frac{5}{5} = 1$

$m = \frac{8}{5}$

$x_{inc} = \frac{5}{8} = 0.6$ $\frac{\Delta x}{steps}$

$y_{inc} = \frac{8}{8} = 1$ $\frac{\Delta y}{steps}$

	x	y
5	5	7
6	5.6	8
6	6.2	9
7	6.8	10
7	7.4	11
8	8	12
9	8.6	13
9	9.2	14
10	9.8	15

x	y
12	9
13	10
14	11
15	12
16	13
17	14

if (x_1, y_1) (x_2, y_2)

$(17, 14)$ $(12, 9)$

$\Delta x = \Delta y = -5$

$x_{inc} = -1$

$y_{inc} = -1$

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

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

```
Algorithm DDA (x1, y1, x2, y2)
    dx = x2 - x1
    dy = y2 - y1
if (abs(dx) > abs(dy)) step = abs(dx)
else step = abs(dy).
xinc = dx / step, yinc = dy / step
for i = 1 ; i <= step ; i++
    putpixel (x1, y1);
    x1 = x1 + xinc
    y1 = y1 + yinc
}
```

Advantages

- simple algo
- easy to implement
- it avoids using the multiplication operation which is costly in terms of time complexity

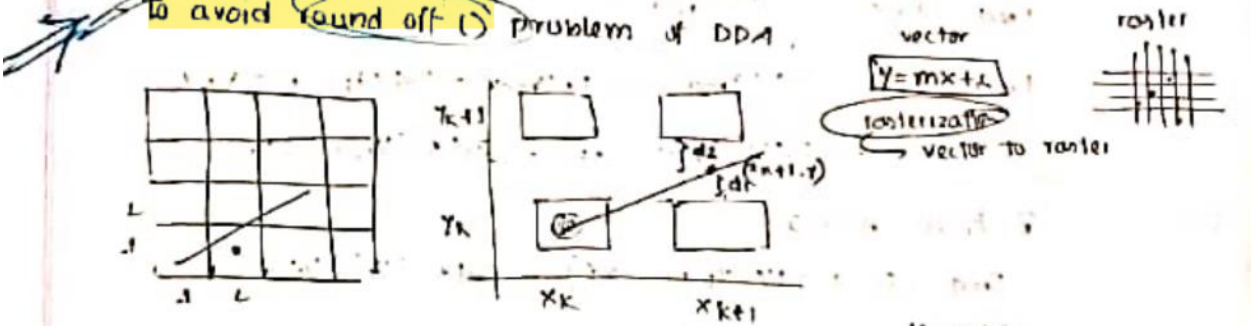
Disadvantages

- time complex increase for round off()
- Resulted lines are not smooth because of round off().
- Points generated by this algo are not accurate

Bresenham's line Algorithm

- An accurate and efficient raster line-generating algorithm, developed by Bresenham, scan converts lines using only incremental integer calculations that can adapted to display circles and other curves.

To avoid round off () problem of DDA



$x_{next} = x_k + 1$

$y_{next} = y_k$ or y_{k+1}

$y = mx + c$

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

$d_1 = y - y_k$

$d_2 = y_{k+1} - y$

$d_1 = [m(x_{k+1}) + c] - y_k$

$d_2 = y_{k+1} - [m(x_k + 1) + c]$

if $d_1 - d_2 < 0$ closer to lower (y_k)

if $d_1 - d_2 > 0$ closer to upper (y_{k+1})

$d_1 - d_2 = [m(x_{k+1}) + c - y_k] - [y_{k+1} - m(x_{k+1}) - c]$

$= m(x_{k+1}) + c - y_k - y_{k+1} + m(x_{k+1}) + c$

$= 2m(x_{k+1}) - 2y_k + 2c - 1$

$\Delta x (d_1 - d_2) = \Delta x \left[2\frac{\Delta y}{\Delta x}(x_{k+1}) - 2y_k + 2c - 1\right]$

$\Delta x (d_1 - d_2) = 2\Delta y (x_{k+1}) - 2\Delta x y_k + 2\Delta x c - \Delta x$

$P_k = 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + 2\Delta x c - \Delta x$ (constant)

$P_k = 2\Delta y x_k - 2\Delta x y_k$

$P_{next} = P_{k+1} = 2\Delta y x_{next} - 2\Delta x y_{next}$

$P_{next} - P_k = 2\Delta y x_{next} - 2\Delta x y_{next} - 2\Delta y x_k + 2\Delta x y_k$

$= 2\Delta y (x_{next} - x_k) - 2\Delta x (y_{next} - y_k)$

Decision variable

if $P_{next} - P_k < 0 \Rightarrow$

$P_{next} = P_k + 2\Delta y (x_{k+1} - x_k) - 2\Delta x (y_k - y_k)$

if $P_k < 0$

$P_{k+1} = P_k + 2\Delta y$ $\quad y_{next} = y_k$

if $P_{next} - P_k \geq 0 \Rightarrow$

if $P_k \geq 0$

$P_{k+1} = P_{next} = P_k + 2\Delta y - 2\Delta x$ $\quad y_{next} = y_k + 1$

Initially,

(x_1, y_1) $\quad y_1 = mx_1 + c \Rightarrow c = y_1 - mx_1$

$P_1 = 2\Delta y x_1 - 2\Delta x y_1 + 2\Delta y + 2\Delta x c - \Delta x$

$= 2\Delta y x_1 - 2\Delta x y_1 + 2\Delta y + 2\Delta x \left[y_1 - \frac{\Delta y}{\Delta x} x_1\right] - \Delta x$

$= 2\Delta y x_1 - 2\Delta x y_1 + 2\Delta y + 2\Delta x y_1 - 2\Delta y x_1 - \Delta x$

$P_1 = 2\Delta y - \Delta x$

For $|m| < 1$ ⇒ ...

1. Input the two line endpoints and store the left endpoint in (x_0, y_0).
2. load (x_0, y_0) into the frame buffer, that is, plot the first point.
3. Calculate constants Δx, Δy, $2\Delta y$, $2\Delta y - 2\Delta x$, and obtain the starting value of decision parameter
$$P_0 = 2\Delta y - \Delta x$$
4. At each x_k along the line, starting at $k = 0$, perform the following test:
If $P_k < 0$, the next point to plot is (x_{k+1}, y_k)
$$P_{k+1} = P_k + 2\Delta y$$
otherwise, the next point to plot is (x_{k+1}, y_{k+1})
$$P_{k+1} = P_k + 2\Delta y - 2\Delta x$$
5. Repeat step 4 Δx times

```
Algo Bresenham (x1, y1, x2, y2) {
    x = x1 ;
    y = y1 ;
    dx = x2 - x1   dy = y2 - y1
    p = 2dy - dx
    while (x ≤ x2) {
        putpixel (x, y) ;
        x++ ;
        if (P < 0)
            P = P + 2dy
        else {
            P = P + 2dy - 2dx
            y++ ;
        }
    }
```

2020. 2021

To illustrate the algorithm, we digitized the line with endpoints (20, 10) and (30, 18). This line has a slope of 0.8 with $\Delta x = 10$, $\Delta y = 8$

→ The initial decision parameter has the value

$$P_0 = 2\Delta y - \Delta x = 16 - 10 = 6$$

and the increments for calculating successive decision parameters are
$$2\Delta y - 2\Delta x = -4 \quad ; \quad 2\Delta y = 16$$

We plot the initial point $(x_0, y_0) = (20, 10)$ and determine successive pixel positions along the line path from the decision parameter as -

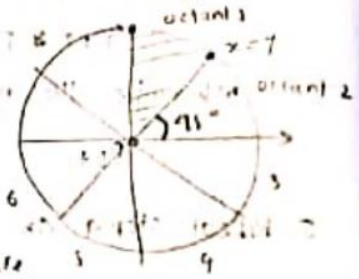
Date: 26.9.

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)

Midpoint Circle drawing:

- To apply the midpoint method we define a circle function:

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



- In the diagram we divide a circle into 8 different octants.

- For the symmetry, if we draw only one octant we can easily draw point on other seven octants using reflection procedure.

- Here we have to find points for the first octant; when $x \geq y$ then it indicates the end of octant 1.

- The first coordinate point is $(0, r)$ to draw the circle, the x increasing as unit interval & the y decreasing along the values of x.

- The next point be either $(x_{K+1}, y_K - 1)$ or $(x_K + 1, y_K)$ so the midpoint of these two points is (x_m, y_m) where

$$x_m = (x_K + 1 + x_{K+1})/2$$

$$y_m = (y_K - 1 + y_K)/2 = y_K - 1/2$$

$$\therefore (x_m, y_m) = (x_K + 1, y_K - 1/2)$$

We know the equation of circle is $x^2 + y^2 = r^2$.

putting (x_m, y_m) in the circle equation, we get:

Decision parameter $P_k = (x_k+1)^2 + (y_k - 1/2)^2 - r^2$

and $P_{k+1} = (x_{k+1}+1)^2 + (y_{k+1} - 1/2)^2 - r^2$

Therefore, $P_{k+1} - P_k = (x_{k+1}+1)^2 + (y_{k+1} - \frac{1}{2})^2 - r^2 - (x_k+1)^2 - (y_k - \frac{1}{2})^2 + r^2$

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

$= (x_k+1)^2 + 2(x_k+1)\cdot 1 + 1 + (y_{k+1})^2 - 2(y_{k+1}\cdot\frac{1}{2}) + (\frac{1}{2})^2 - (y_k)^2 + 2 y_k \cdot \frac{1}{2} - 1 - (x_k+1)^2$

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

$\therefore P_{k+1} = P_k + 2(x_k+1) + \{(y_{k+1})^2 - (y_k)^2\} - (y_{k+1} - y_k) + 1$

Now,

The starting point is $(0, r)$, so $x_k = 0$

$y_k = r$ for the initial decision parameter P_0

$P_0 = (0+1)^2 + (r - \frac{1}{2})^2 - r^2$

$= 1 + r^2 + \frac{1}{4} - r - r^2$

$= \frac{5}{4} - r$

$P_0 = 1 - r$ [Approximate; to avoid the fractal part]

now, if $P_k < 0$ then

$y_{k+1} = y_k$

next pt (x_k+1, y_k)

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

else if $P_k \geq 0$ then

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

next point is $(x_k+1, y_k - 1)$

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

Midpoint circle Algo.

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:

$$P_0 = \frac{5}{4} - r$$

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 $2y_{k+1} = 2y_k - 2$

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

4. Determine 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 coordinate values:

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

6. Repeat steps 3 through 5 until $x \geq y$.

Recursion circle is optimized form of midpoint

Advantages

- used to generate curves on raster displays.
- powerful and efficient algo
- based on only $x^2 + y^2 = r^2$

Given a circle radius $r = 10$, we demonstrate the midpoint circle algo by drawing, determining positions along the circle octant in the first quadrant from $x = 0$ to $x = y$. The initial value of the decision parameter is

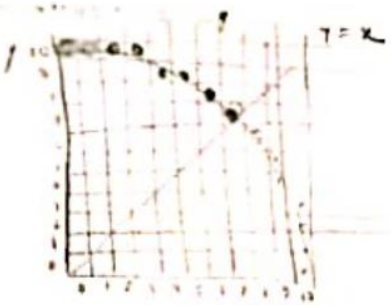
$$P_0 = 1 - r = -9$$

For the circle centered on the coordinate origin, the initial point is $(x_0, y_0) = (0, 10)$, and initial increment terms for calculating the decision parameters are

$$2x_0 = 0 \qquad 2y_0 = 20$$

Successive decision parameter values and positions along the circle path are calculated using the midpoint method as

k	p_k	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
0	-9	(1, 10)	2	20
1	-6	(2, 10)	4	20
2	-1	(3, 10)	6	20
3	6	(4, 9)	8	18
4	-3	(5, 9)	10	18
5	8	(6, 8)	12	16
6	5	(7, 7)	14	14

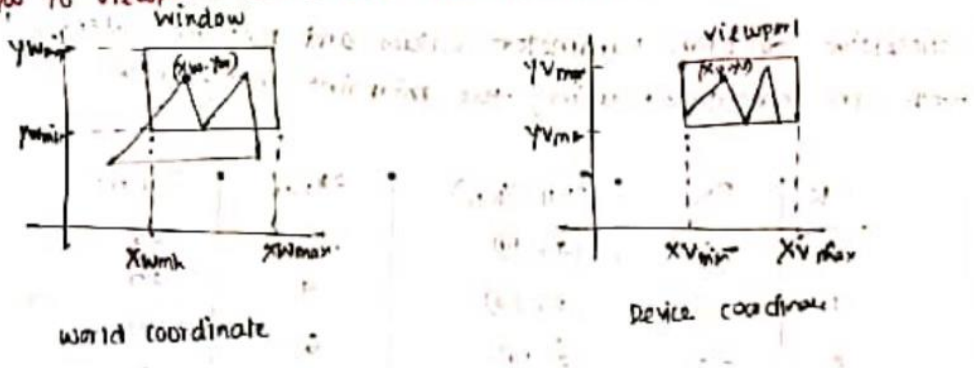


Chapter 6
Dimensional viewing

Window: A world-coordinate area selected for display is called a window.
- what is to be viewed
- Area of a picture that is selected for viewing

Viewport: An area on a display device to which a window is mapped is called a viewport.
- where it is to be displayed

Window to viewport coordinate Transformation



(x_w, y_w) (x_v, y_v)

Normalized window/coordinate point $\left(\frac{x_w - x_{wmin}}{x_{wmax} - x_{wmin}}, \frac{y_w - y_{wmin}}{y_{wmax} - y_{wmin}}\right)$

Normalized viewport point $\left(\frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}}, \frac{y_v - y_{vmin}}{y_{vmax} - y_{vmin}}\right)$

For x-coordinate, $\frac{x_w - x_{wmin}}{x_{wmax} - x_{wmin}} = \frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}}$

$$\Rightarrow x_v = x_{vmin} + (x_w - x_{wmin})\left(\frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}\right)$$

$$= x_{vmin} + (x_w - x_{wmin})\, s_x$$

where scaling factors are

$$S_x = \frac{X_{Vmax} - X_{Vmin}}{X_{Wmax} - X_{Wmin}}$$

For y cordinate.

$$y_v = y_{Vmin} + (y_w - y_{Wmin}) s_y$$

- viewing transformation:

The mapping of a part of a world-coordinate scene to device coordinate

- 2D viewing transformation ⇒ window-to-viewport transformation / window transformation

- clipping:

Generally, any procedure that identifies those portions of a picture that are either inside or outside of a specified region of space is referred to as clipping algorithm or simply clipping.

- The region against which an object is to clipped is called a clip window.

Algorithms for clipping -

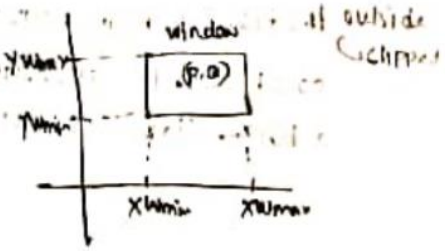
- point clipping
- line clipping (straight-line seg)
- Area clipping (polygon)
- Curve clipping
- Text

point clipping:

- is a process which is used to define the point position. The point is either inside the ~~viewpoint~~ view plane (window) or outside the view plane.

$$X_{Wmin} \le P \le X_{Wmax}$$

$$y_{Wmin} \le Q \le y_{Wmax}$$



$x_{wmax} \leq P \leq x_{wmin}$
$y_{wmax} \leq Q \leq y_{wmin}$
} outside
↳ must be clipped

If any one of these four inequalities is not satisfied, the point is clipped (not saved for display) ⟶ outside

Line clipping

2020

→ Cohen - Sutherland line clipping Algo

- This is one of the ~~best~~ oldest and most popular line-clipping
- Generally, the method speeds up the processing of line segments by performing initial tests that reduce the number of intersecting that must be calculated.
- Every line end-point in a picture is assigned a four-digit binary code, called a region code that identifies the location of the point relative to the boundaries of the clipping rectangle.

⟹ Note

Polygon clipping

→ Sutherland Hodgeman P.C.A

- We can correctly clip a polygon by processing the polygon boundary as a whole against each window edge.
- 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 of vertices could then be successively passed to a right boundary clipper, a bottom / top 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 second vertex are added to the output vertex list.

outside to inside (A to B)
- intersecting pt A'
- Destination pt B

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

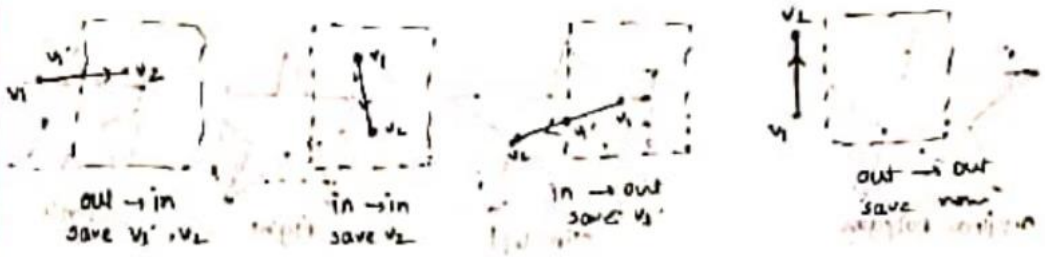
inside to inside (P to Q)
- Destination Q

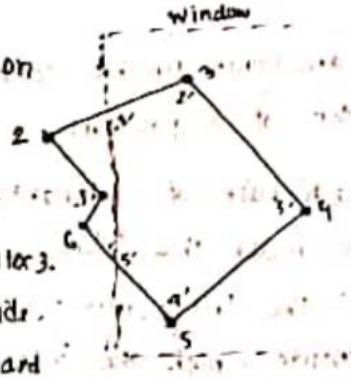
(3) If the first vertex is inside the window boundary and the second vertex is outside, only the edge intersection with the window boundary is added to the output vertex list.

inside to outside (B to C)
- intersecting point B'

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

outside to outside
- null





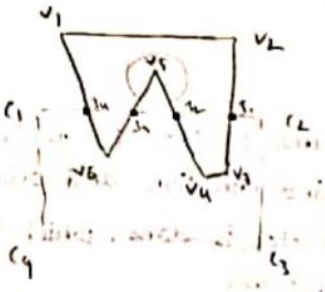
- Vertex 1 and 2 are found to be on the outside of the boundary. Moving along to vertex 3, which is inside, we calculate the intersection and save both the intersection point and vertex 3. Vertex 4 and 5 are determined to be inside and they also are saved. The sixth and final vertex is outside, so we find and save the intersection point.

- We can eliminate the intermediate output vertex list by simply clipping individual vertices at each step and passing the clipped vertices on to the next boundary clipper.

- A point (either an input vertex or a calculated intersection point) is added to the output vertex list only after it has been determined to be inside or on a window boundary by all four boundary clippers. Otherwise the point does not continue in the pipeline.

→ weiler Atherton polygon clipping Algorithm

↳ for concave polygon

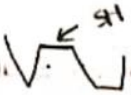


solution

limitations of SH

only if inner angle of a polygon is $<180°$

$>180° \rightarrow$

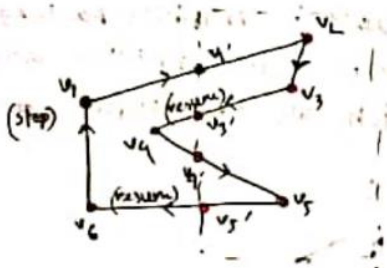


- The basic idea in this algorithm is that instead of always proceeding around the polygon edges as vertices are processed, we sometimes want to follow the window boundaries. Which path we follow depends on the polygon-processing direction (clockwise or counterclockwise) and whether the pair of polygon vertices currently being processed represents an outside-to-inside pair or an inside to outside pair. //

- For clockwise processing of polygon vertices, we use the following rules:

  - for an outside-to-inside pair of vertices, follow the polygon boundary
  - for an inside-to-outside pair of vertices, follow the window boundary in a clockwise direction.

- An improvement on the Weiler-Atherton Algo is the weiler algo, which applies constructive solid geometry ideas to clip an arbitrary polygon against any polygon-clipping region.



generates the two separate polygon areas.

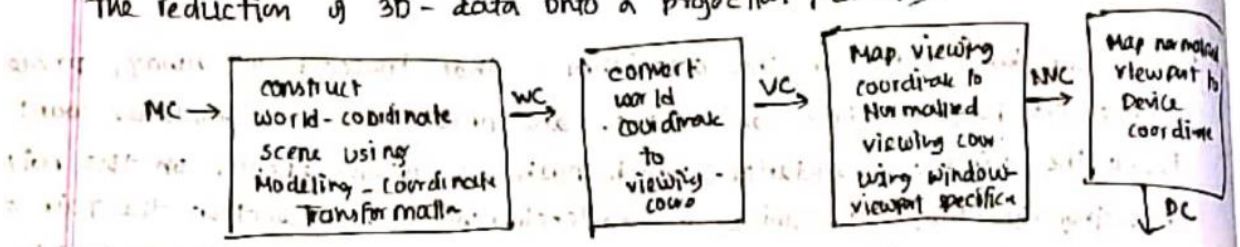
CH-12

Chapter 12

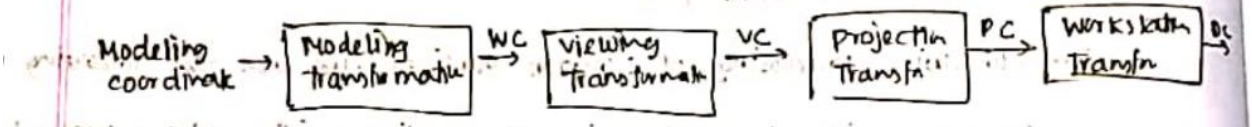
Three-dimensional viewing

-Viewing pipeline:

• The viewing pipeline in 3D is almost the same as the 2D-viewing pipeline. only after the definition of the viewing direction and orientation (i.e. of the camera) an additional projection step is done, which is the reduction of 3D-data onto a projection plane



2D viewing transformation pipeline



3D ~~viewing~~ transformation from modeling coordinate to final device coordinate

-The general processing steps for modeling and converting a world-coordinate description of a scene to device coordinate. once the scene has been modeled, world-coordinate positions are converted to viewing co-ordinates.

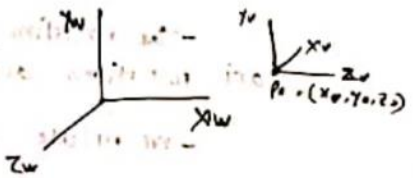
- The viewing coordinate system is used in graphics packages as a reference for specifying the observer viewing position and the position of the projection plane, which we can think of an analogy with the camera film plane

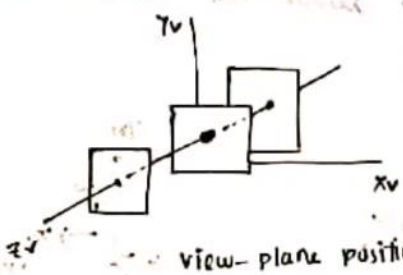
- Next, projection operations are performed to convert the viewing-coordinate description of the scene to coordinate positions on the projection plane, which will then be mapped to the output device.

View reference point:

- we choose a particular view for a scene by first establishing the viewing-coordinate system, also called the view reference coordinate system.

- To establish the viewing-coordinate reference frame, we first pick a world-coordinate position called view reference point:





view-plane positioning along the Zv axis.

Transformation from world to viewing coordinates:

- Before object descriptions can be projected to the view plane, they must be transferred to viewing coordinates.

- The transformation sequence is

(1) Translate the view reference point to the origin of the world-coordinate system.

(2) Apply rotations to align the Xv, Yv and Zv axes with the world Xw, Yw and Zw axes respectively.

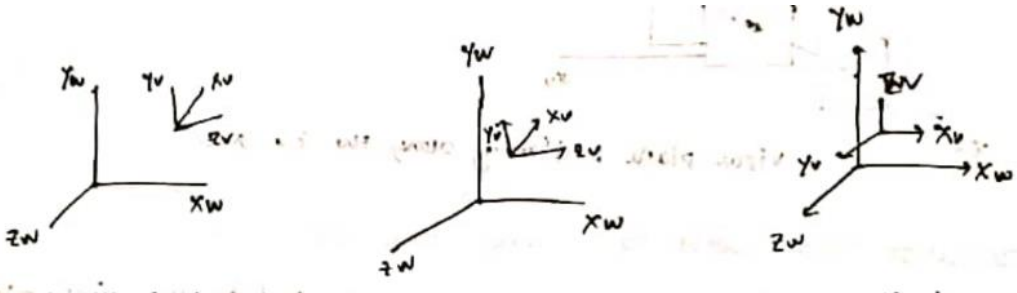
-If the view reference point is specified at world position (x_0, y_0, z_0) this point is translated to the world origin with the matrix transformation

bring to origin

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

-The rotation sequence can require up to three coordinate-axis rotations, depending on the direction we choose for N.

-We rotate around the world y_w axis to align the z_w and z_v axes. The final rotation is about the z_w axis to align the y_w and y_v axes.



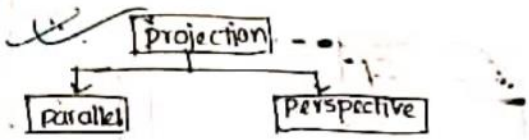
Aligning a viewing system with the world-coordinate axes using a sequence of translate-rotate transformation.

Date: 27.4.23

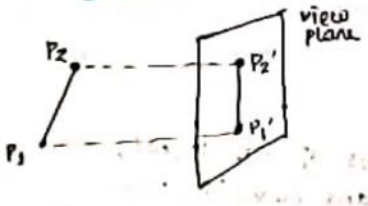
Projection:

- once world-coordinate descriptions of the objects in a scene are converted to viewing coordinates.

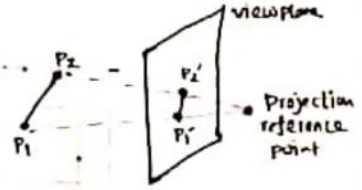
- We can project the 3D objects onto the 2D view plane. There are two basic projection methods



- coordinate positions are transformed to the view plane along parallel lines



- object positions are transformed to the view plane along lines that converge to a point called perspective reference point / center of projection

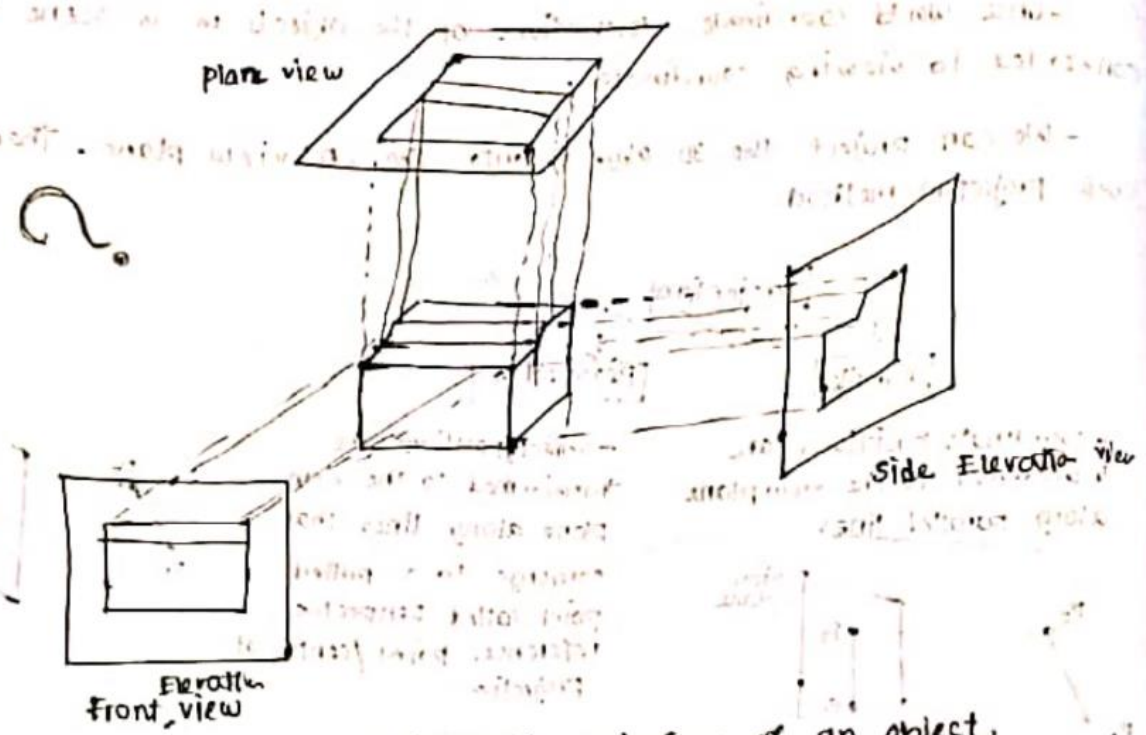


Orthographic parallel projection:

- We can specify a parallel projection with a projection vector that defines the direction for the projection lines. When the projection is perpendicular to the view plane, we have an orthographic parallel projection.

Axonometric orthographic projection:

- We can also form orthographic projections that display more than one face of an object. Such views are called axonometric orthographic projections. The most commonly used axonometric projection is the isometric projection.



orthographic projections of an object, displaying plane plan and elevation views

View volume:

- is the part of the window that is visible in the image.
- is determined by a combination of the viewing transformation and the projection transformation.
- The size of the view volume depends on the size of the window, while the shape of the view volume depends on the size of the window type of projection to be used to generate the display.

page : 999

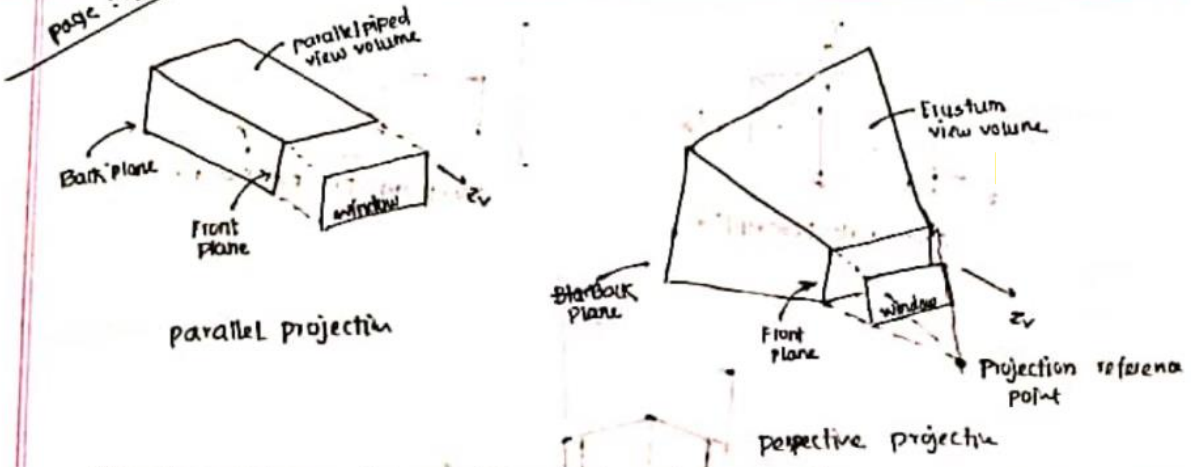


Fig: view volumes bounded by front and back planes, and by top, bottom, and side planes. Front and back planes are parallel to the view plane at positions z_{front} and z_{back} along the z_v axis.

- A finite view volume is obtained by limiting the extent of the volume in the z_v direction. This is done by specifying positions for one, or two additional boundary planes. These z_v - boundary planes are referred to as the Front plane and back plane, or near plane and the far plane of the view volume.

- The front and back planes are parallel to the view plane at specified positions z_{front} and z_{back}

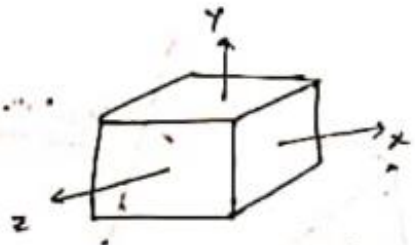
⇒ Frustum:

- With a perspective projection, the front and back clipping planes truncate the infinite pyramidal view volume to form a frustum.

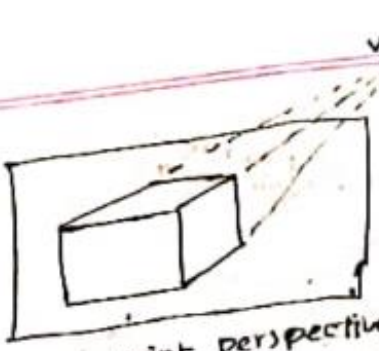
Vanishing point:

- The point at which a set of projected parallel lines appears to converge is called a vanishing point.

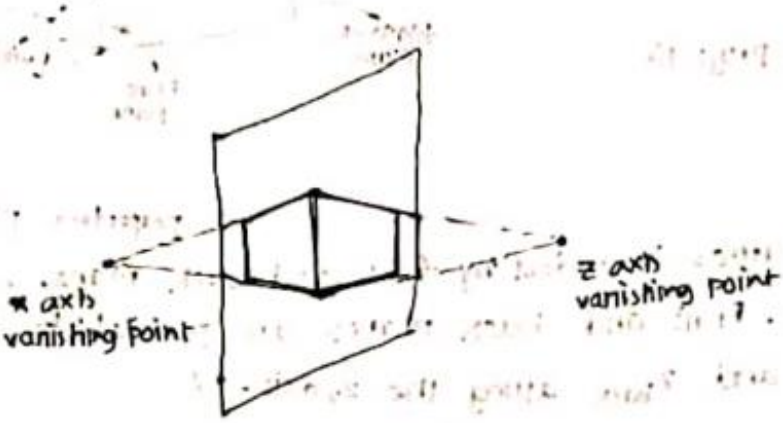
- The vanishing point for any set of lines that are parallel to one of the ~~principle~~ principal axes of an object is referred to as a principal vanishing point.



coordinate description



one-point perspective projection



Two point perspective projection

Fig: Perspective views and principal vanishing points of a cube for various orientations of the view plane relative to the principal axes of the object. //

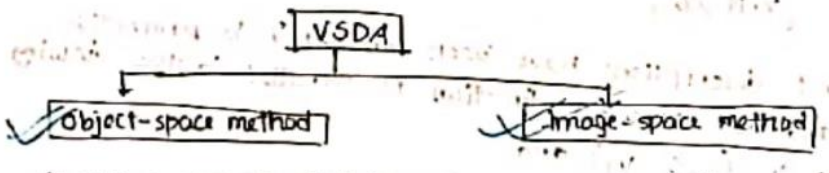
Chapter 13

Visible-Surface Detection Methods

↳ Hidden-surface elimination method

Classification of VSDA

- Visible-surface detection algorithms are broadly classified according to whether they deal with object definitions directly or with their projected images.



- An object-space method compares objects and parts of object to each other within the scene definition to determine which surfaces, as a whole, we should label as visible.

- In an image-space algo, visibility is decided point by point at each pixel position on the projection plane.

- Most visible-surface algo use image-space methods, although object-space methods can be used effectively to locate visible surfaces in some cases.

Object space

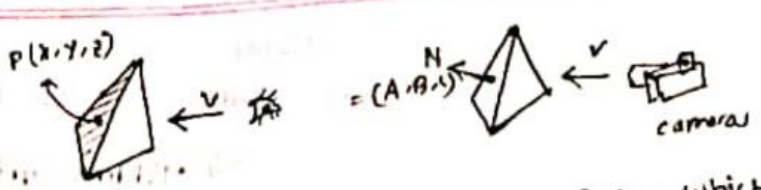
Back-Face Detection:

- A fast and simple object-space method for identifying the back face of the object.

- A point (x, y, z) is "inside" a polygon surface with plane parameters A, B, C, D if

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

When an inside point is along the line of sight to the surface, the polygon must be a back-face.



- consider the normal vector N to a polygon surface, which has cartesian components (A, B, C).

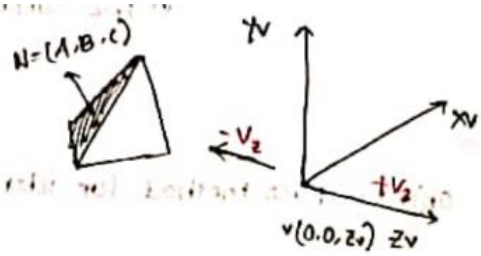
- If V is a vector in the viewing direction from the eye (or camera) position, then the polygon is a back face if

$$V \cdot N > 0$$

- If object descriptions have been converted to projection coordinates and our viewing direction is parallel to the viewing z_v axis, then $V = (0, 0, V_z)$ and

$$V \cdot N = V_z C$$

so that we only need to consider the sign of C, the z component of the normal vector N.



for nonoverlapping object

- In a right handed viewing system with viewing direction along the negative z_v axis, the polygon is a back face if $C < 0$. Also, we cannot see any face whose normal has z component $C = 0$.

- So, we can label any polygon as a back face if its normal vector has a z component value

$$C \leq 0$$

$C \geq 0$ → when viewing direction is along (+) axis (left)

BOOK 472 page

Advan

- For polygon surface it is easy to implement
- requires no sorting of surfaces in a scene

32

Disadvantages

- it requires the availability of a second buffer in addition to the the refresh buffer.
- it can find only one visible surface at each pixel position.

Depth Buffer Method (z-buffer)

A commonly used image-space approach to detecting visible surface is the depth-buffer method, which compares surface depths at each pixel position on the projection plane. This procedure is also refferred as the z-buffer method, since object depth is usually measured from the view plane along the z-axis of a viewing system.

Steps:

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

$$depth(x,y) = 0 \quad refresh(x,y) = I_{backgnd}$$

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

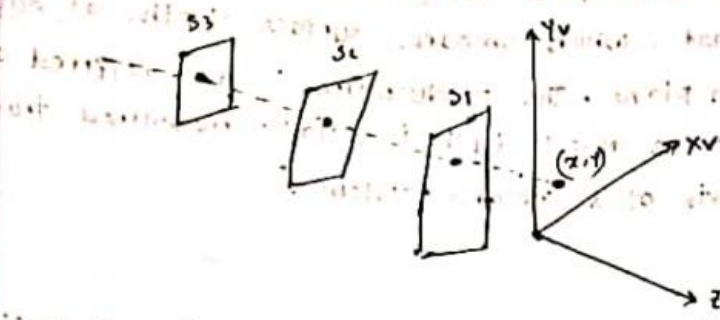
$$depth(x,y) = z \quad refresh(x,y) = I_{surf}(x,y)$$

Where $I_{backgnd}$ is the value for the background intensity, and

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

After all surfaces have been processed, the depth buffer contains depth values for the visible surfaces and the refresh buffer contains the corresponding intensity values for those surfaces.

- For each pixel position (x,y) on the view plane, object depth can be compared by comparing z value.
  - range from (0) at the back clipping plane to z_{max} at the front-clipping plane.
  - The value z_{max} can be set either to 1 or to the largest value that can be stored on the system.



S_1 has the smallest depth from the view plane and so is visible at that position.

- Three surfaces at varying distances along the orthographic projection line from position (x,y) in a view plane taken as the $x_v y_v$ plane. Surface S_1 is the closest at this position, so its surface intensity value at (x,y) is saved.

- 2 buffer areas are required.

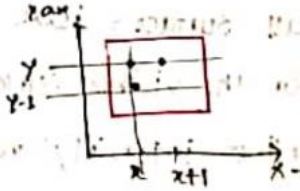
(1) Depth buffer
- is used to store depth values for each (x,y) pos as surfaces are processed.

(2) Refresh buffer
- stores the intensity values for each pos

Depth values for a surface position (x,y) -

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

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



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

$$z' = z - \frac{A}{C}$$

The ratio $-A/C$ is constant for each surface, so succeeding depth values across a scan line obtained from preceding values with a single addition.

Depth-Sorting Method (Painter's algo):

-Using both image-space and object-space operations, the depth-sorting method performs the following basic functions:

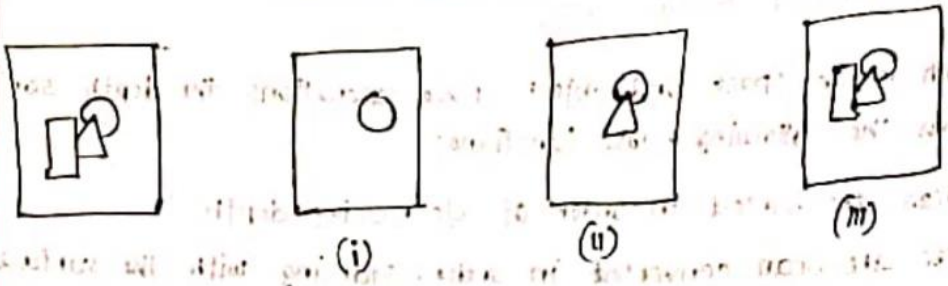
1. Surfaces are sorted in order of decreasing depth.
2. Surfaces are scan converted in order, starting with the surface of greatest depth.

-This method for solving the hidden-surface problem is often referred to as the painter's algo. In creating an oil painting, an artist first paints the background colors. Next, the most distant objects are added, then the nearest objects, and so forth. At the final step, the foreground objects are painted on the canvas over the bg and other objects that have been painted on the canvas.

-Each layer of paint covers up the previous layer. Similarly, we first sort surfaces according to their distance from the view plane. The intensity values for the farthest surface are then entered into the refresh buffer. Taking each succeeding surface in turn (in decreasing depth order), we paint the surface intensities onto the frame buffer over the intensities of the previously processed surfaces.

-If any one of these tests is true, no reordering is necessary for that surface. The tests are listed in order of increasingly difficulty.

1. The boundary rectangles in the xy plane for the two surfaces do not overlap.
2. Surface S is completely behind the overlapping surface relative to the viewing position.
3. The overlapping surface is completely in front of S relative to the viewing pos
4. The projections of the two surfaces onto the view plane don't overlap.

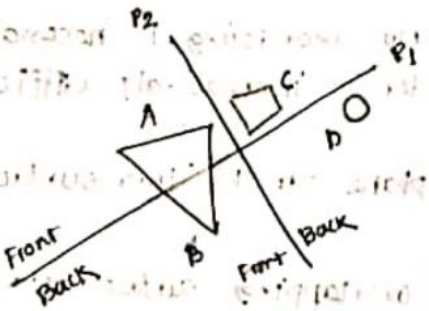


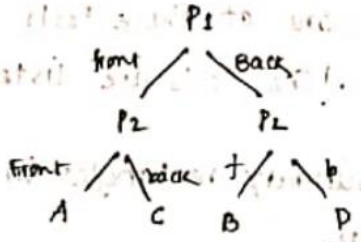
Binary space-partitioning-Tree Method (BSP tree)

- A Binary space-partitioning (BSP) tree is an efficient method for determining object visibility by painting surfaces onto the screen from back to front, as in the painter's algo.
- It is a two step procedure.

Step 1: construction of BSP tree

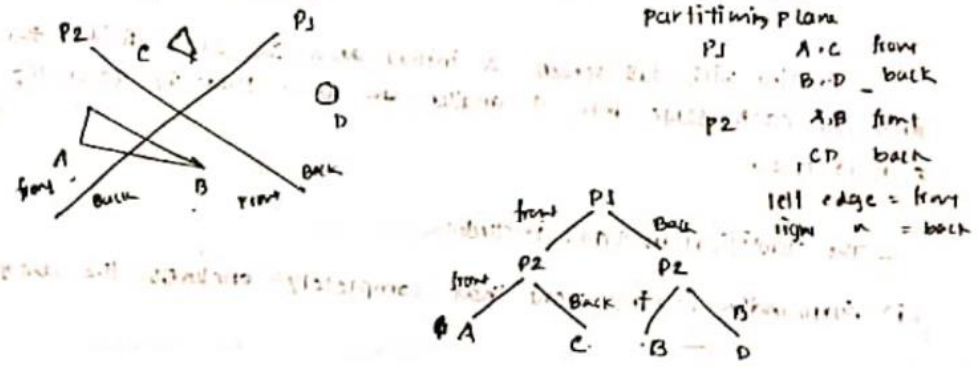
- BSP algo recursively divides the space into two half spaces keeping in account that the dividing plane is one of polygon of picture. Then this dividing polygon is taken as root of tree.





BSP tree method

↳ view reference point (v.r.p) is changing



Inorder

- In this tree, the objects are represented as terminal nodes, with front objects as left branches and back objects as right branches.

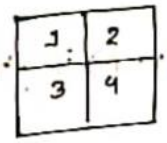
Area Subdivision Method

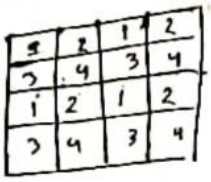
→ comparing pixel of projected images

- is an image-space method that divides the total viewing area into smaller and smaller rectangles until a single visible surface or no surface at all.

Divide and conquer

- It follows (divide and conquer) approach.
- uses area coherence property (to decrease its calculation)
- This method applied to successively divide the total viewing area into smaller and smaller rectangles until each small area is the projection of part of a single visible surface or no surface at all.
- The subdivision is based on (2) steps:
  (1) It determines which of the projected polygons onto a given area is give visible ✓
  (2) In each area these polygons are further tested to find which one will be visible with this area and it should be displayed





- starting with full screen as initial area the algo divides the area at each stage into 4 smaller area there by generating a quad tree.

- The classification types include:

(i) Surrounding surface : one that completely encloses the area

(II) Overlapping surface : one that is partly inside and partly outside the area

(III) Inside surface : one that is completely inside the area

(IV) outside surface : one that is completely outside the are

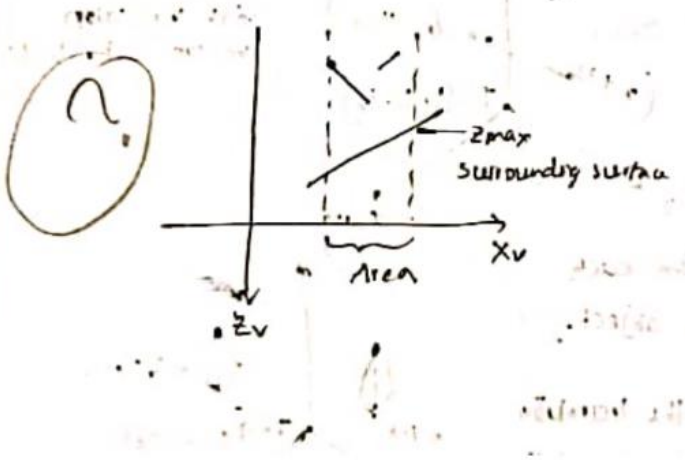
- No further subdivision of a specified area are needed if one of the following conditions is true :

1. All surfaces are outside surfaces with respect to the area.
2. only one inside, overlapping or surrounding surfaces is in the area.
3. A surrounding surface obscures all other surfaces within the area boundaries.

- Test 1 can be carried out by checking the bounding rectangles of all surfaces against the area boundaries.

- Test 2 can also use the bounding rectangles in the xy plane to identify an inside surface.

- one method for implementing test 3 is to order surfaces a/c to their minimum depth from the view plane. For each surrounding surface, we then compute the maximum depth within the area under consideration. If the maximum depth of one of these surrounding surface is closer to the view plane than the min depth of all other surfaces within the area, test 3 is satisfied.



Chapter 11.

Three dimensional Geometric and modeling transformation

Translation:

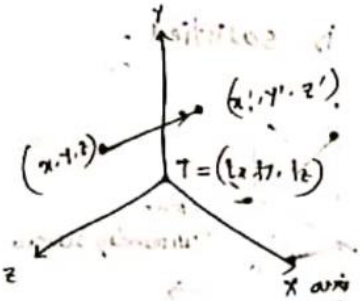
- In a three-dimensional homogeneous coordinate representation, a point is translated from position $P = (x, y, z)$ to position $P' = (x', y', z)$ with the matrix operation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{——— (1)}$$

$$P' = T \cdot P \quad \text{——— (II)}$$

- 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 in eq (1) is equivalent to the three eqn.

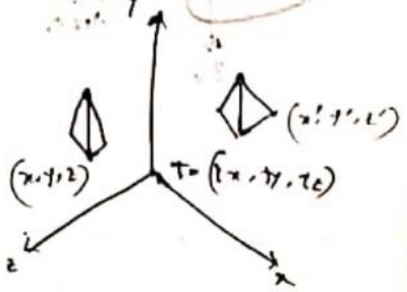
$$\begin{cases} x' = x + t_x \\ y' = y + t_y \\ z' = z + t_z \end{cases}$$



Translating a point with translation vector $T = (t_x, t_y, t_z)$

- An object is translated in three dimensions by transforming each of the defining points of the object.

- We obtain the inverse of the translation matrix in eq (1) by negating the translation distances t_x, t_y, t_z. This produces a translation in the opposite direction and the product of a t. matrix and its inverse produces the identity matrix



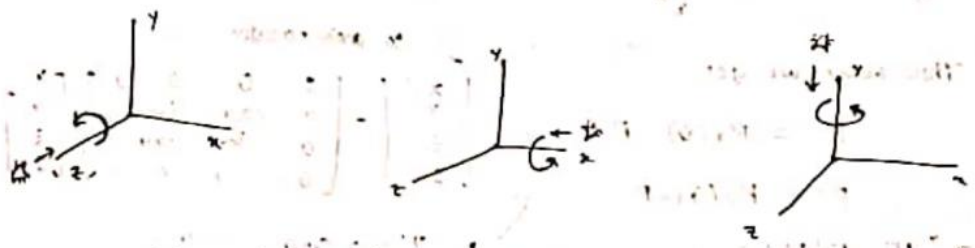
an object

Rotation

- To generate a rotation transformation for an object, we must designate an axis of rotation (about which the object is to be rotated) and the amount of angular rotation.

- The easiest rotation axes to handle are those that are parallel to the coordinate axes. Also, we can use combinations of coordinate-axis rotations (along with appropriate translations) to specify any general rotation.

- positive rotations in the xy plane are counter-clockwise about axes parallel to the z axis.



positive rotation directions about the coordinate axes are counterclockwise when looking toward the origin from a positive coordinate position on each axis

Coordinate-Axes Rotation

- The 2D z-axis rotation eqs are easily extended to 3D:

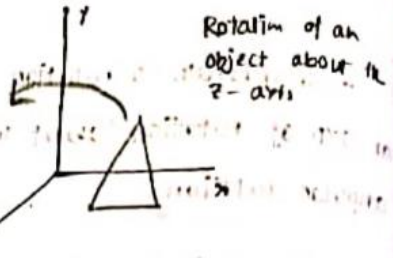
$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta \quad \text{——— (1)}$$
$$z' = z$$

Parameter θ specifies the rotation angle. In homogeneous coordinate form, the 3D z-axis rotation equations are expressed as

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{——— (II)}$$

which we can write more compactly as

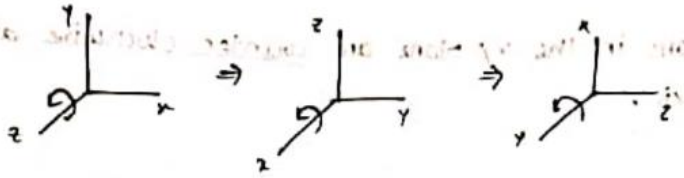
$$P' = R_z(\theta) \cdot P \quad \text{——— (3)}$$



Rotation of an object about the z-axis

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

$$x \to y \to z \to x$$



Other Figs?

Thus we get,

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

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

x-axis rotation

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

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

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

$$x' = x$$

y axis rotation -

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

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

$$y' = y$$

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

- An inverse rotation matrix is formed by replacing the rotation angle θ by $-\theta$.

- since only the sine function is affected by the change in sign of the rotation angle, the inverse matrix can be also be obtained by interchanging rows and columns. That is, we can calculate the inverse of any rotation matrix R by evaluating its transpose ($R^{-1} = R^T$).

General 3D Rotations

- A rotation matrix for any axis that does not coincide with a coordinate axis can be set up as a composite transformation involving combinations of translations and the coordinate axes rotation.

- In the special case where an object is to be rotated about an axis that is parallel to one of the coordinate axes, we can attain the desired rotation with the following transformation sequence.

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.

- Any coordinate position P on the object is transformed with the seq

$$P' = T^{-1} R_x(\theta) \cdot T \cdot P \qquad ; R(\theta) = T^{-1} \cdot R_x(\theta) \cdot T$$

- When an object is to be rotated about an axis that is not parallel to one of the coordinate axes, we need to perform some additional transform In this case, we also need rotations to align the axis with a selected coordinate axis and to bring the axis back to its original orientation.

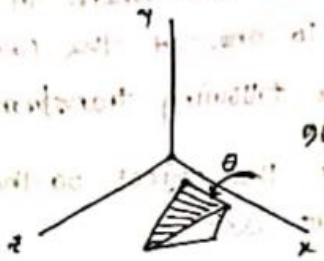
5 steps are -

1. Translate the object so that the rotation axis passes through the coordinate origin.
2. Rotate the object so that the axis of rotation coincides with one of the coordinate axes.
3. perform the specified rotation about that coordinate axis.

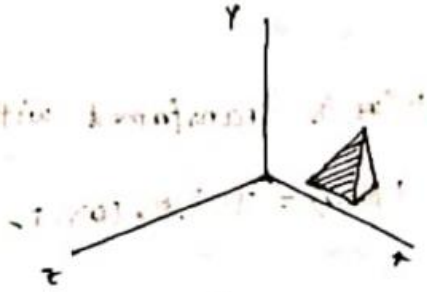
4. Apply inverse rotations to bring the rotation axis back to its original orientation.

5. Apply the inverse translation to bring the rotation axis back to its original position.

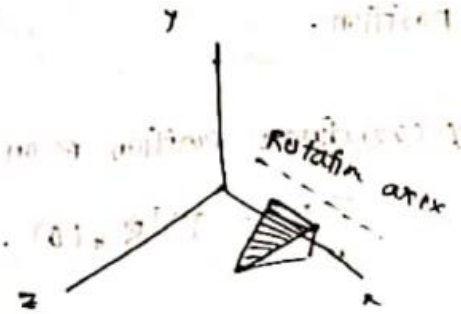
(a) original pos of object



(c) Rotate object through angle θ



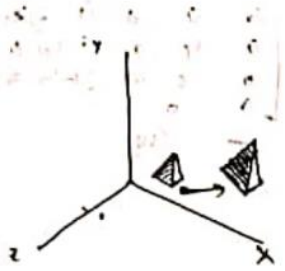
(b) Translate rotation Axis onto x Axis



(d) Translate Rotation Axis to original position

Scaling

- The matrix expression for the scaling transformation of a position $P = (x, y, z)$ relative to the coordinate origin can be written as



Doubling the size of an object with transformation eq(1) also moves the object farther from the origin.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{---(I)} \Rightarrow P' = S \cdot P \quad \text{---(II)}$$

where scaling parameters s_x, s_y, s_z are assigned any positive values. Explicit expressions for the coordinate transformations for scaling relative to the origin are

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

Scaling an object with transformation eq.(1) changes the size of the object and repositions the object relative to the coordinate origin.

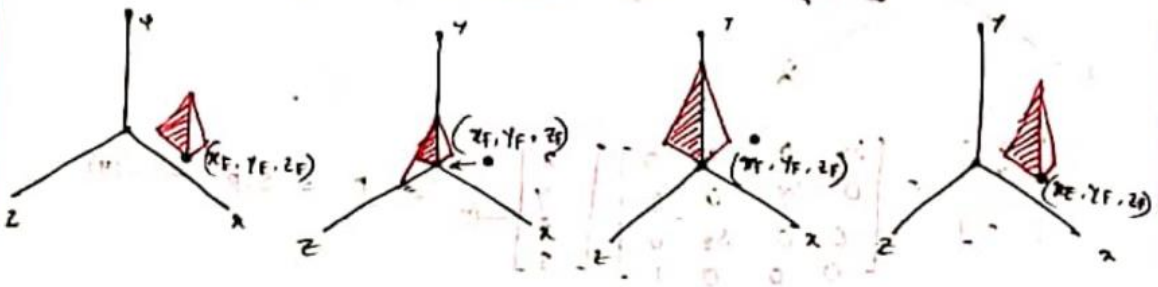
- We preserve the original shape of an object with a uniform scaling $(s_x = s_y = s_z)$.

- Scaling with respect to a selected fixed position (x_f, y_f, z_f) can be represented with the following transformation seq:

1. Translate the fixed point to the origin
2. Scale the object relative to the coordinate origin using eq.(1)
3. Translate the fixed point back to its original position

- The matrix representation for an arbitrary fixed-point scaling can then be expressed as the concatenation of these translate-scale translate transformation as

$$T(x_f, y_f, z_f) \cdot S(s_x, s_y, s_z) \cdot T(-x_f, -y_f, -z_f) = \begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(III)}$$



- We form the inverse scaling matrix for either (I) or (III) by replacing the scaling parameters s_x, s_y, and s_z with their reciprocals.

- The inverse matrix generates an opposite scaling transformation, so the concatenation of any scaling matrix and its inverse produces the identity matrix.

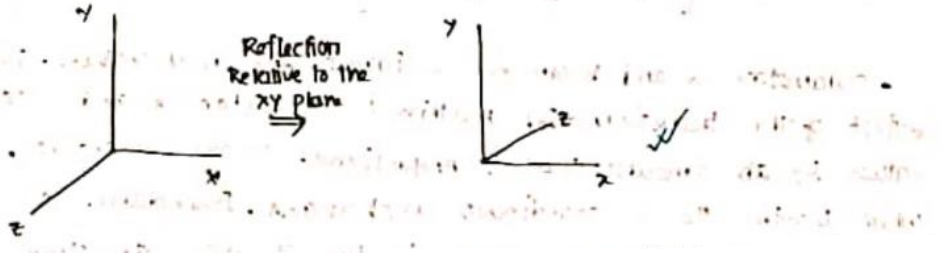
Other transformations

Reflections :

- A three dimensional reflection can be performed relative to a selected reflection axis or with respect to a selected reflection plane.

- In general, 3D ~~reflection~~ reflection matrices are set up similarly to those for two dimensions.

- Reflections relative to a given axis are equivalent to 180° rotations in ~~3D space~~ about that axis.



conversion of coordinate specification from a right-handed to a left-handed system can be carried out with the reflection transformation.

- The matrix representation for this reflection of points relative to the xy plane is

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

- Transformation matrices for inverting x and y values are defined similarly, as reflections relative to the yz-plane and xz-plane, respectively. Reflections about other planes can be obtained as a combination of rotations and coordinate-plane reflections.

Chapter 10

Three-dimensional Object Representation

Plane Equations:

- To produce a display of a 3D object, we must process the input data representation for the object through several procedures. These processing steps include transformation of the modeling and world-coordinate descriptions to viewing coordinates, then to device coordinates; identification of visible surfaces; and the application of surface-rendering procedures.

- The equation for a plane surface can be expressed in the form

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

where (x, y, z) is any point on the plane

Coefficients A, B, C, D = constants describing the spatial properties of the plane.

Polygon meshes:

- Some graphics packages (for example, PHIGS) provide polygon functions for modeling objects. A single plane surface can be specified with a function such as fillArea. But when object surfaces are to be tiled, it is more convenient to specify the surface facets with a mesh function.

- One type of polygon mesh is the triangle strip. This function produces $(n-2)$ connected triangles for (n) vertices.

- When polygons are specified with more than three vertices, it is possible that the vertices may not lie in one plane.

Quadratic surfaces

- A frequently used class of objects are the quadric surfaces, which are described with second-degree equations (quadratics).

- They include spheres, ellipsoid, tori, paraboloids, hyperboloids.

- Quadric surfaces, particularly spheres and ellipsoids are common elements of graphics scenes, and they are often available in graphics packages as primitives from which more complex objects can be constructed.

Sphere:

- In cartesian coordinates, a spherical surface with radius r centered on the coordinate origin is defined as the set of points (x, y, z) that satisfy the equation

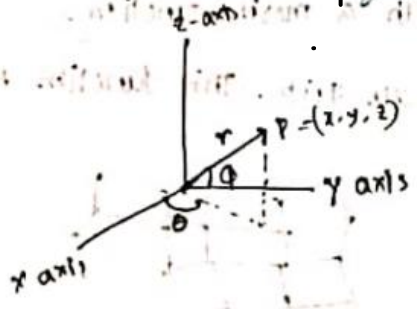
$$x^2 + y^2 + z^2 = r^2$$

We can also describe the spherical surface in parametric form, using latitude and longitude angles

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

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

$$z = r\sin\phi$$



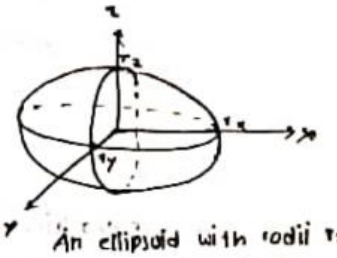
Parametric coordinate position (r, θ, ϕ) on the surface of a sphere with radius r

Ellipsoid:

- An ellipsoid surface can be described as an extension of a spherical surface, where the radii in three mutually perpendicular directions can have different values.

- the cartesian representation for points over the surface of an ellipsoid centered on the origin is

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



An ellipsoid with radii r_x, r_y, r_z centered on the coordinate origin.

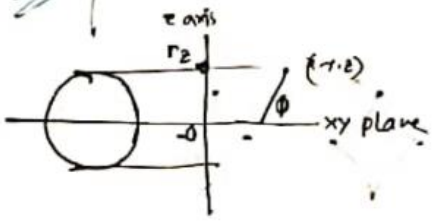
And a parametric representation for the ellipsoid in terms of the latitude angle ϕ and the longitude angle θ.

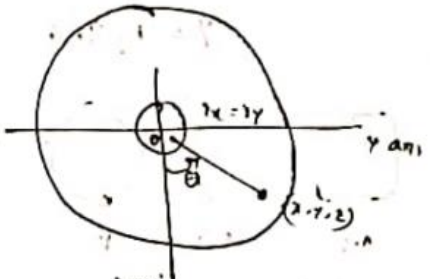
$$x = r_x \cos\phi \cos\theta \qquad -\pi/2 \le \phi \le \pi/2$$
$$y = r_y \cos\phi \sin\theta \qquad -\pi \le \theta \le \pi$$
$$z = r_z \sin\phi$$

Torus

- A torus is a doughnut-shaped object.
- It can be generated by rotating a circle or other conic about a specified axis.
- The cartesian representation for points over the surface of a torus can be written in the form

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





A torus with a circular cross section centered on the coordinate origin

where r is any given offset value.

$$x = r_x (r + \cos\phi)\cos\theta \qquad -\pi \le \phi \le \pi$$
$$y = r_y (r + \cos\phi)\sin\theta \qquad -\pi \le \theta \le \pi$$
$$z = r_z \sin\phi$$

Superquadrics

- superquadrics are formed by incorporating additional parameters into the quadric equations to provide increased feasibility for adjusting object shapes.

- The no. of additional parameter used is equal to the dimension of the object : 1 parameter for curves
2 " for surfaces

Superellipse

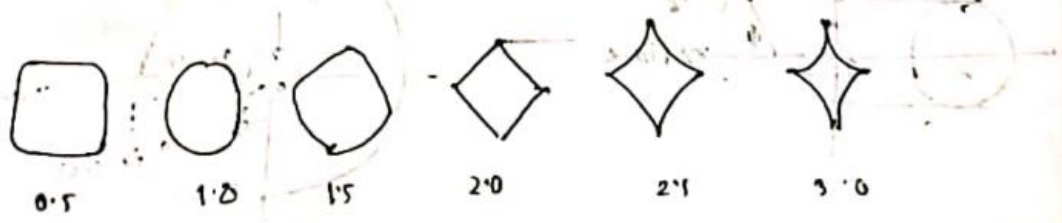
- We obtain a cartesian representation for a superellipse from one corresponding eq for an ellipse by allowing the exponent on the x and y terms to be variable.

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

where parameter s can be assigned any real value. When
$s = 1$ we get an ordinary ellipse

corresponding parametric equations

$$x = r_x \cos^s\theta \qquad -\pi \le \theta \le \pi$$
$$y = r_y \sin^s\theta$$



superellipses plotted with different values for parameter s and with $r_x = r_y$.

Date: 29.4.23

Superellipsoid

- A cartesian representation for a superellipsoid is obtained from the eq for an ellipsoid by incorporating two exponent parameters:

$$\left[\left(\frac{x}{r_x}\right)^{2/s_2} + \left(\frac{y}{r_y}\right)^{2/s_2}\right]^{s_2/s_1} + \left(\frac{z}{r_z}\right)^{2/s_1} = 1$$

For $s_1 = s_2 = 1$, we have an ordinary ellipsoid.

$$x = r_x \cos^{s_1}\phi \cos^{s_2}\theta \qquad -\pi/2 \le \phi \le \pi/2$$
$$y = r_y \cos^{s_1}\phi \sin^{s_2}\theta \qquad -\pi \le \theta \le \pi$$
$$z = r_z \sin^{s_1}\phi$$

Blobby objects:

- some objects do not maintain a fixed shape, but change their surface characteristics in certain motion or when in proximity to other objects. i.e. molecular structures, water droplets and other liquid effects, melting objects, muscle shapes in human body etc.

- These objects can be described as exhibiting "blobbiness" and are often simply referred to as blobby objects, since their shapes show a certain degree of fluidity.

- several models have been developed for representing blobby objects as distribution function over a region of space.

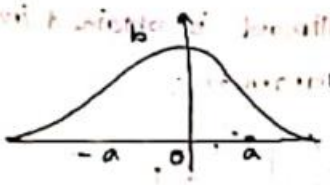
By Gaussian density function, a surface function can be defined as

$$f(x,y,z) = \sum_K b_K e^{-a_K r_K^2} - T = 0 \qquad \text{where,}$$

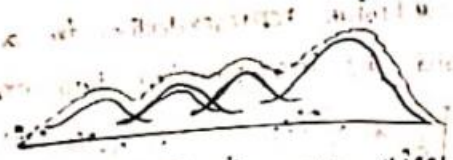
$r_K^2 = \sqrt{x_K^2 + y_K^2 + z_K^2}$

T = some specified threshold

Blobby muscle shapes in a human arm



A 3D gaussian bump centered at position 0, with height b and standard deviation a.



A composite blobby object formed with four gaussian bumps.

Spline representation

- In drafting ~~tech~~ terminology, a spline is a flexible strip used to produce a smooth curve through a designated set of points.

- several small weights are distributed along the length of the strip to hold it in position on the drafting table as the curve is drawn.

- the term spline curve originally referred to a curve drawn in this manner.

- A spline surface can be described with two sets of orthogonal spline curves.

- splines are used in graphics applications to design curve and surface shapes, to digitize drawings for computer storage, and to specify animation paths for the objects or the camera in a scene.

- Typical CAD applications for splines include the design of automobile bodies, aircraft, ship hulls etc.

Interpolation and approximation splines

control pt:

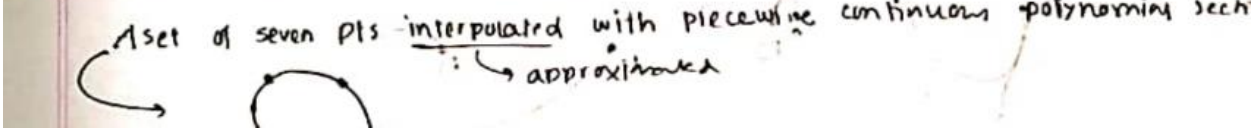
- we specify a spline curve by giving a set of coordinate positions, called control points, which indicates the general shape of the curve.

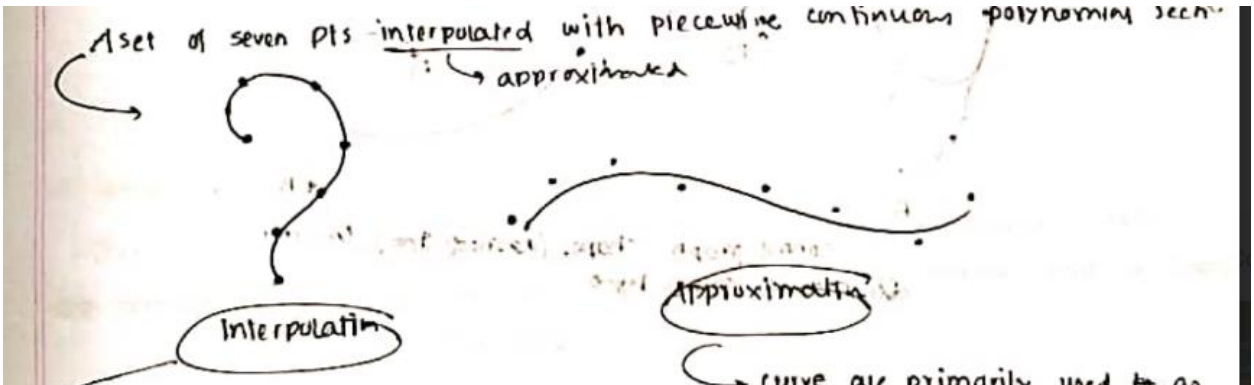
Interpolation:

- when polynomial sections are fitted so that the curve passes through each control point then it is said to interpolate the set of control points.

Approximation:

- 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.





→ curves are commonly used to digitize drawings or to specify animation paths.

→ curve are primarily used as design tools to structure object surfac

convex hull:

- The convex polygon boundary that encloses a set of control points is called convex hull.

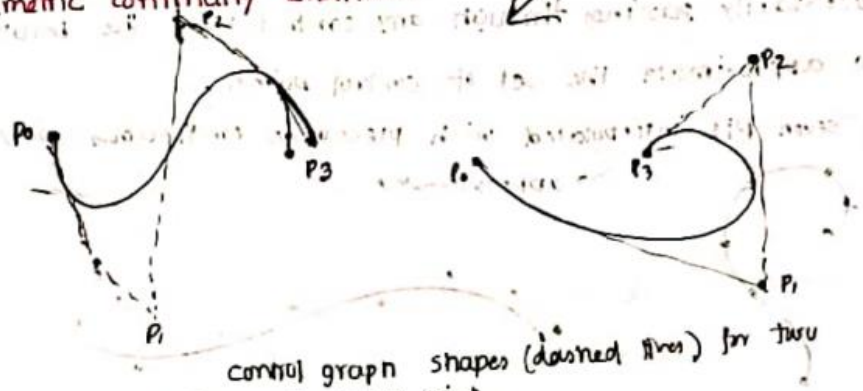
control graph :

- A ~~polygon~~ polyline connecting the sequence of control points for an approximation spline is usually displayed to remind a designer of the control-point ordering. This set of connected line segment is often referred to as the control graph of the curve.

control polygon :

- Other names for the series of straight-line sections connecting the control points in the order specified are control polygon or characteristics polygon.

parametric continuity conditions :



control graph shapes (dashed lines) for two diff. sets of control points.

- 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.

1) zero-order parametric ~~contin~~ continuity (C^0 continuity):

- means simply that the curves meet.

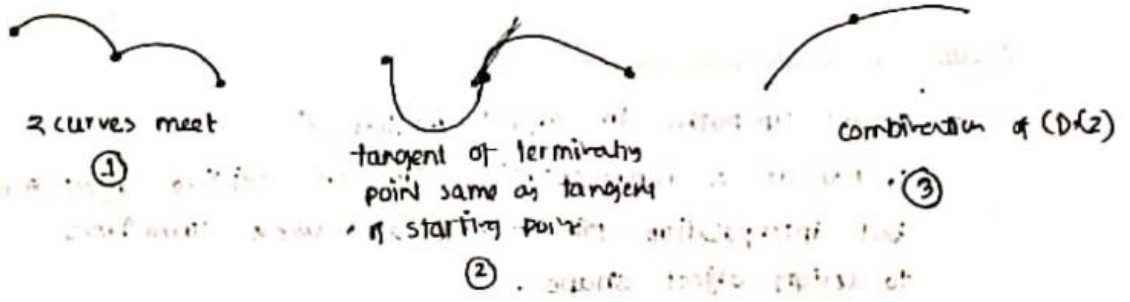
- That is, the values of x, y, z evaluated at u_2 for the first curve section are equal, respectively to the values of x, y, z evaluated at u_1 ~~continuity~~ for the next curve section.

(2) First - order (C^1) :

- means that first parametric derivatives (tangent lines) of the coordinate functions for two successive curve sections are equal at their joining point.

(3) Second - order (C^2) : → useful for setting up animation paths for camera motion and for many precision CAD requirements

- both the first and second parametric derivatives of two curve sections are the same at the intersection.



Spline specification:

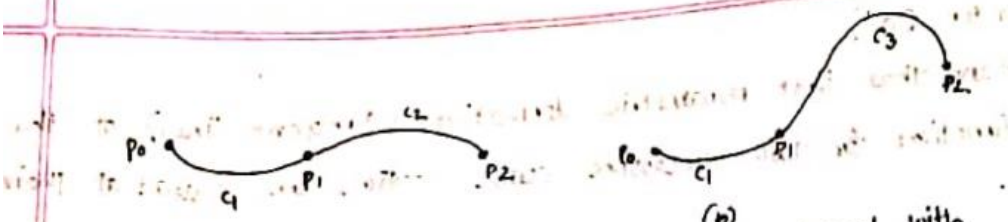
- There are three equivalent methods for specifying a particular spline representation. (1) we can state the set of boundary conditions that are imposed on the spline.

(2) we can state the matrix that characterizes the spline

(3) we can state the set of blending functions (or basis func.) that determines how specified geometric constraints on the curve are combined to calculate positions along the curve path.

Parametric cubic ~~functions~~ polynomial representation to illustrate the methods.

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



(a) (b)

Three control points fitted with two curve sections joined with
(a) parametric continuity
(b) geometric continuity

where the tangent of curve C_3 at point P_1 has a greater magnitude than the tangent vector of curve C_1 at P_1.

Cubic spline interpolation method :

- To set up paths for object motions
- to provide a representation for an existing object or drawing
but interpolation splines are also used sometimes to design object shapes.

- we can describe the parametric cubic polynomial that is to be fitted between each pair of control points with the following set of equations :

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$$

$$y(u) = a_y u^3 + b_y u^2 + c_y u + d_y \qquad 0 \le u \le 1$$

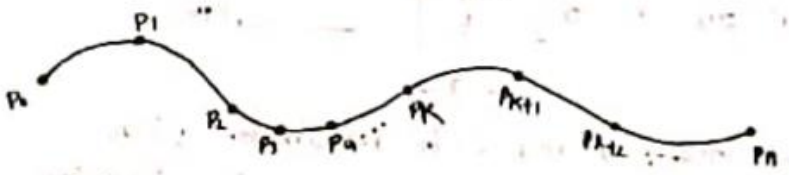
$$z(u) = a_z u^3 + b_z u^2 + c_z u + d_z$$

Natural cubic splines

- one of the first spline curves to be developed for graphical applications is the natural cubic spline.

- This interpolation curve is a mathematical representation of the original drafting spline.

- If we have n+1 control points to fit, then we have n curve sections with a total of 4n polynomial coefficients to be determined.



A piecewise continuous cubic-spline interpolation of n+1 control points

Hermite Interpolation:

- A Hermite spline (named after the French mathematician charles Hermite) is an interpolating piecewise cubic polynomial with a specified tangent at each control point

- Unlike the natural cubic splines, Hermite splines can be adjusted locally because each curve section is only dependent on its endpoint constraints.

- If P(u) represents a parametric cubic point function for the curve section, then the boundary conditions that define this Hermite curve section are

$$P(0) = P_K$$
$$P(1) = P_{K+1}$$
$$P'(0) = DP_K \quad \text{——(1)}$$
$$P'(1) = DP_{K+1}$$

with DP_K and DP_{K+1} specifying the values for the parametric derivatives (slope of the curve) at control points P_K and P_{K+1} respectively.

- we can write the vector equivalent for ↑ for hermite curve section as

$$P(u) = au^3 + bu^2 + cu + d, \quad 0 \le u \le 1 \qquad \text{(ii)}$$

where the x component of P is $x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$ and similarly for y and z components. The matrix equivalent of eq (ii) is

$$P(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \qquad \text{(iii)}$$

and the derivative of the point function can be expressed as

$$P'(u) = \begin{bmatrix} 3u^2 & 2u & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \qquad \text{(iv)}$$

Substituting endpoint values 0 and 1 for parameter u into the previous two equations, we can express the Hermite boundary conditions ~~in the~~ eq (i) in the matrix form:

$$\begin{bmatrix} P_K \\ P_{K+1} \\ DP_K \\ DP_{K+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \qquad \text{(v)}$$

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

parametric point function $P(u)$ for a Hermite curve section between control points P_K and P_{K+1}.

- solving this equation for the polynomial coefficients, we have

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} P_K \\ P_{K+1} \\ DP_K \\ DP_{K+1} \end{bmatrix}$$

$$= \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_K \\ P_{K+1} \\ DP_K \\ DP_{K+1} \end{bmatrix} \quad \text{---------- (VI)}$$

$$= M_H \cdot \begin{bmatrix} P_K \\ P_{K+1} \\ DP_K \\ DP_{K+1} \end{bmatrix}$$

where M_H, the Hermite Matrix, is the inverse of the boundary constraint matrix.

$$P(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \cdot M_H \cdot \begin{bmatrix} P_K \\ P_{K+1} \\ DP_K \\ DP_{K+1} \end{bmatrix} \quad \text{---------- (VII)}$$

Finally, we can determine expression for the Hermite blending functions by carrying out the matrix multiplication in eq (VII) and collecting coefficients for the boundary constraints to obtain the polynomial form:

$$P(u) = P_K (2u^3 - 3u^2 + 1) + P_{K+1} (-2u^3 + 3u^2) + DP_K (u^3 - 2u^2 + u) + DP_{K+1} (u^3 - u^2)$$

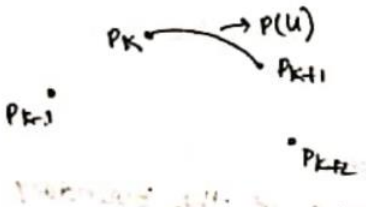
$$= P_K H_0(u) + P_{K+1} \{H_1(u) + DP_K H_2(u) + DP_{K+1} H_3(u)$$

Cardinal Splines

- As with Hermite splines, cardinal splines are interpolating piecewise cubics with specific endpoint tangents at the boundary of each curve section.

- The diff is that we don't have to give the values for the endpoint tangents.

- For a cardinal spline, the value for the slope at a control point is calculated from the coordinates of the two adjacent control points.



- A cardinal spline section is completely specified with four consecutive control points. The middle two control points are the section endpoints, and the other two points are used in the calculation of the endpoint slopes. So, the boundary conditions for cardinal spline section as

$$P(0) = P_K$$

$$P(1) = P_{K+1}$$

$$P'(0) = \frac{1}{2}(1-t)(P_{K+1} - P_{K-1}) \quad \text{——— (1)}$$

$$P'(1) = \frac{1}{2}(1-t)(P_{K+2} - P_K)$$

check figure

$$P(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \cdot M_c \cdot \begin{bmatrix} P_{K-1} \\ P_K \\ P_{K+1} \\ P_{K+2} \end{bmatrix} \quad \text{——— (II)}$$

where the cardinal matrix b.

$$M_c = \begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad \text{——— (III)}$$

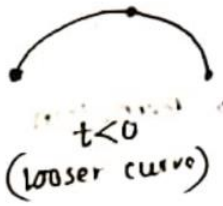
with $s = (1-t)/2$.

Expanding matrix eq (II) into polynomial form we have

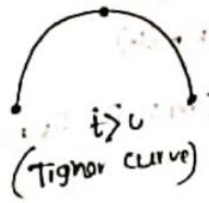
$$P(u) = P_{K-1}(-su^3 + 2su^2 - su) + P_K[(2-s)u^3 + (s-3)u^2 + 1] + P_{K+1}[(s-2)u^3 + (3-2s)u^2 + su] + P_{K+2}(su^3 - su^2)$$

$$= P_{K-1} CAR_0(u) + P_K CAR_1(u) + P_{K+1} CAR_2(u) + P_{K+2} CAR_3(u)$$

where the polynomials $CAR_K(u)$ for $K = 0, 1, 2, 3$ are the cardinal blending functions.



$t<0$
(looser curve)



$t>0$
(Tighter curve)

Effect of tension parameter on the shape of a cardinal spline sect

Bezier curves

- A Bezier curve section can be fitted to any number of control points.
- The number of control points to be approximated and their relative position determine the degree of the Bezier polynomial.
- As with the interpolation splines, a Bezier curve can be specified with boundary conditions, with a characterizing matrix, or with blending functions.
- Bezier splines have a no. of properties that make them highly useful and convenient for curve and surface design.
- Suppose we are given n+1 control-point positions:

$P_K = (x_K, y_K, z_K)$, with 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.

$$P(u) = \sum_{K=0}^{n} P_K \, BEZ_{K,n}(u), \quad 0 \le u \le 1$$

The Bezier blending functions $BEZ_{K,n}(u)$ are the Bernstein polynomial:

$$BEZ_{K,n}(u) = C(n,K)\, u^K (1-u)^{n-K}$$

where the $C(n,K)$ are the binomial coefficients:

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

Properties of Bezier curve:

- A very useful property of a Bezier curve is that it always passes through the first and last control point.

That is, the boundary conditions at the two ends of the curve are

$$P(0) = P_0$$
$$P(1) = P_n$$

Values of the parametric first derivatives of a Bezier curve at the endpoints can be calculated from control-point coordinates as

$$P'(0) = -nP_0 + nP_1$$
$$P'(1) = -nP_{n-1} + nP_n$$

Thus the slope at the beginning of the curve is along the line joining the first two control points, and the slope at the end of the curve is along the line joining the last two endpoints.

$$P''(0) = n(n-1)[(P_2 - P_1) - (P_1 - P_0)]$$
$$P''(1) = n(n-1)[(P_{n-2} - P_{n-1}) - (P_{n-1} - P_n)]$$

- Another important property of any Bezier curve is that it lies within the convex hull (convex polygon boundary) of the control points:

$$\sum_{k=0}^{n} BEZ_{k,n}(u) = 1$$

Fractal

- Natural object can be realistically described with fractal geometry methods, where procedures rather than equations are used to model object.

- Fractal geometry representation for object are commonly applied in many fields to describe and explain the features of natural phenomena.

- In computer graphics, we use fractal methods to generate displays of natural objects and visualizations of various mathematical and physical systems.

- A fractal object has two basic characteristics:
  - infinite detail at every point
  - a certain self-similarity between the object parts and overall features of the object.

- The self-similarity properties of an object can take different forms, depending on the choice of fractal representation.

- By repeated operation for producing the detail in the object structure.

Fractal Dimension:

- We can describe the amount of variation in the object detail with a number called the fractal dimension.

⇒ Used to model terrain, clouds, water, trees and other plants, feathers, fur and varian surface.

Date : 30.9.23

Fractal

- Self-similar
  - have parts that are scaled-down versions of the entire object.
  - similar parts ; statistically self-similar
- Self-affine
  - formed with diff scaling parameters s_x, s_y, s_z in diff coordi
- Invariant fractal
  - are formed with nonlinear transformation

Fractal Dimension

- The detail variation in a fractal object can be described with a number D, called the fractal dimension, which is a measures of the roughness, or fragmentation, of the object.
- More jagged looking objects have larger fractal dimensions.
- we can get up some iterative procedures to generate fractal objects using a given value for the fractal Dimension D.

$D_E = 1$ $\quad s = 1/n, n = 2$

$ns' = 1$

$ns' = 1$

$D_E = 2, \quad s = 1/n\ \frac{1}{2}, n = 4$

$ns^2 = 1$

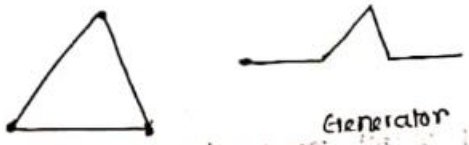
$D_E = 3, \quad s = \frac{1}{n}\ \frac{1}{2}, n = 8$

$ns^3 = 1$

Subdividing objects with Euclidean dimensions $D_E = 1$, $D_E = 2$, $D_E = 3$ using scaling factor $s = \frac{1}{2}$

Geometric construction of deterministic self-similar fractals

- To geometrically construct a deterministic (nonrandom) self-similar fractal, we start with a given geometric shape, called the initiator.
- Subparts of the initiator are then replaced with a pattern called the generator.



Initiator

Generator

First three iterations in the generation of the Koch curve.