Computer Graphics

Chapter 1: Introduction

What's computer graphics ?

- Creation, storage and manipulation of images.
- Process of creating an object into a computer.
- It deals with graphics created using computers and the representation of images data by a computer specifically with help from specialized graphic hardware and software.

What are the components of computer graphics ?

- Modeling:- process of defining objects in terms of primitives, coordinates and characteristics and creating and representing the geometry of objects.
- Storing:- process of storing scenes and images in memory and on disk.
- Manipulating:- process of changing the shape, position and characteristics of objects in a scene.
- Rendering:- process of applying physically based procedures to generate images from scenes.
- Viewing:- process of displaying images from various viewpoints on various devices.
- Animation:- process of describing how objects move (change) in time.

What the common display devices ?

- Analog display devices
  - Oscilloscope
  - TV CRTs
- Digital display devices
  - LED
  - LCD
  - VF

    - ELDs
- Others
    - Electronic paper, Laser TV

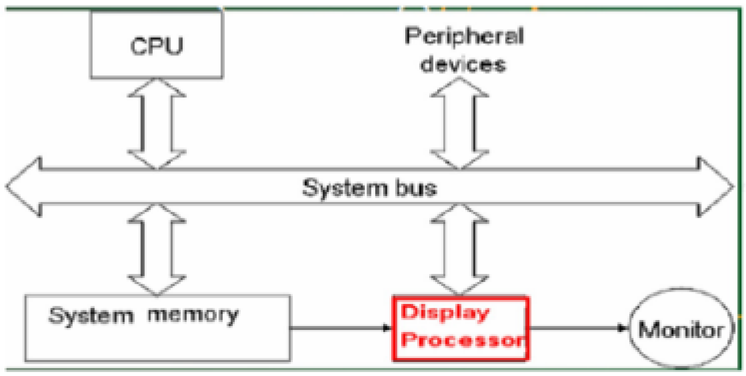
What are the basic components of Cathode Ray Tube (CRT) ?

- Electron gun:- source of narrow high velocity electron beams.
- Focusing system:- acts like a light lens with a focal length such that the center of focus is the screen.
    - Electrostatic focusing:- electrons pass through positively charged metal cylinder.
    - Magnetic focusing:- coils are mounted outside of CRT envelope which produces smallest spot.
- Deflection system:- deflects/ directs electron beam horizontally and / or vertically to any point on the screen.
    - Horizontal and vertical deflectors allow the electron beam to be focused on any spot on the screen.
    - Controlled by electric (plates) or magnetic deflection coils.
- Phosphor screen:- glows when electron beams hits surface of tube coated with a special organic compound.

What’s the difference between Vector Display Technology and Raster Display Technology ?

- **Vector Display Technology**
    - Type of display in which an electron gun is used to draw patterns on the monitor.
    - Unlike standard cathode ray tube (CRT) technology used in televisions, in which horizontal lines are drawn in sequence, a vector display only creates images on the screen where necessary, and skips over blank areas.
    - The electron gun producing the beam of light is controlled by a command (stored in refresh display file or vector file) which signals it when to turn on or off.
    - Lines are smooth and follow the patterns of pure mathematical models.

- Shapes like polygons and bitmaps are not possible to be drawn by vector graphics.
- Displaying artifacts like aliasing and pixelation are absent in vector graphics, but colors are usually limited in CRT vector monitors.
- Architecture



Pros

- High resolution
- Easy animation
- Requires little memory

Cons

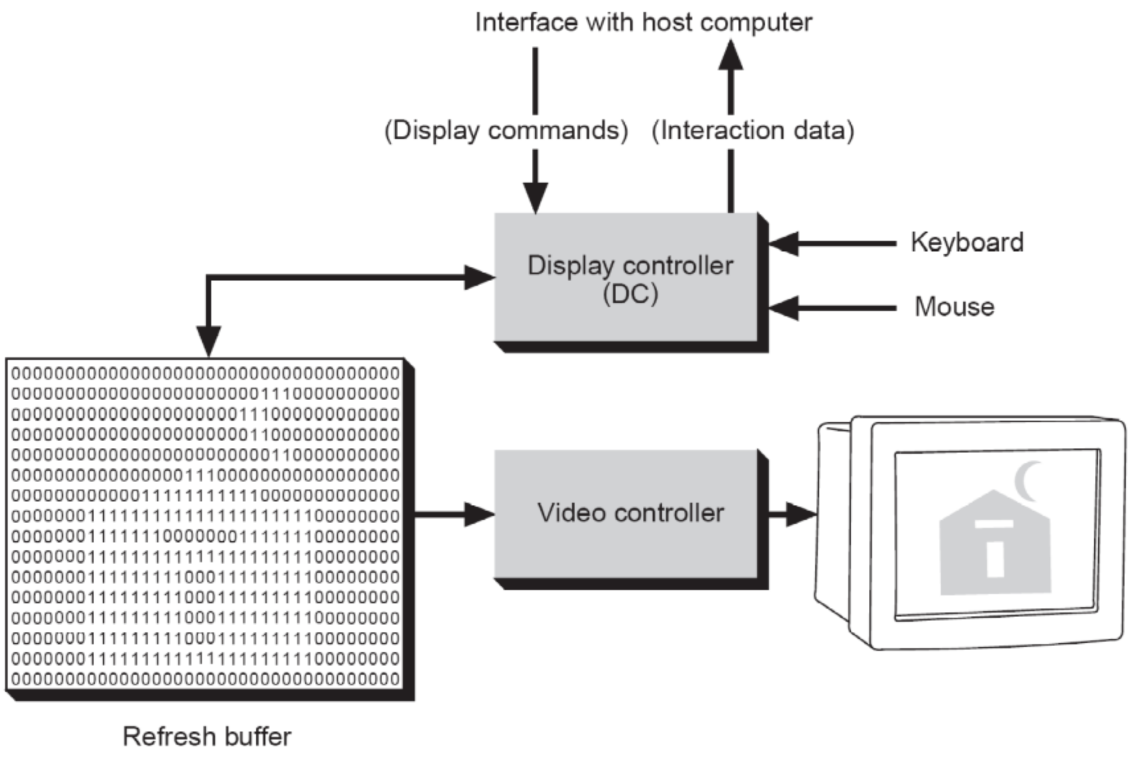
- Requires intelligent electron beam
- Limited screen density
- Limited color capability
- Very expensive

- **Raster Display Technology**
  - Terminologies
    - Raster:- series of adjacent parallel lines which together form an image on display screen.
    - Raster scan:- representation of images as collection of pixels/ dots.
    - Pixel:- one dot or picture element of the raster.
    - Scan line:- a row of pixels.
  - Raster is shared as matrix of pixels representing entire screen area.
  - Brightness of each pixel controlled by the intensity of the electron beam.
  - Electron beams take 1/80 - 1/60 seconds to scan raster lines and draw a specific shape.

- Refresh rate is minimum 24Hz to avoid flicker. Current raster-scan displays have a refresh rate of at least 60 frames (60 Hz) per second, up to 120 (120 Hz).
- Refresh procedure:-
  - Horizontal retrace – beam returns to left of screen.
  - Vertical retrace – bean returns to top left corner of screen.
- Interlaced refresh – display first even-numbered lines, then odd-numbered

lines permits to see the image in half the time

- Architecture



What are the components of Computer Graphics Architecture ?

- **Application model**:- contains data and objects that are to be displayed on a monitor, such as a scene description or image file.
- **Application program**:- contains programs that contain convert graphic model graphical content by generating output commands.
- **Graphics system**:- contains GPU (graphics chip and memory) that process scene content for display rendering based on application commands.

Chapter 2: Simple drawing algorithms

What are output primitives ?

- Basic geometric structures used to describe scenes.

How is output primitive is represented in Vector Display Technology and Raster Display Technology ?

<u>Raster</u>

- Completely specified by set of intensities for the pixel positions in the display.
- Shapes and colors described with pixel array.
- Picture displayed by loading pixel array to frame buffer.

<u>Vector</u>

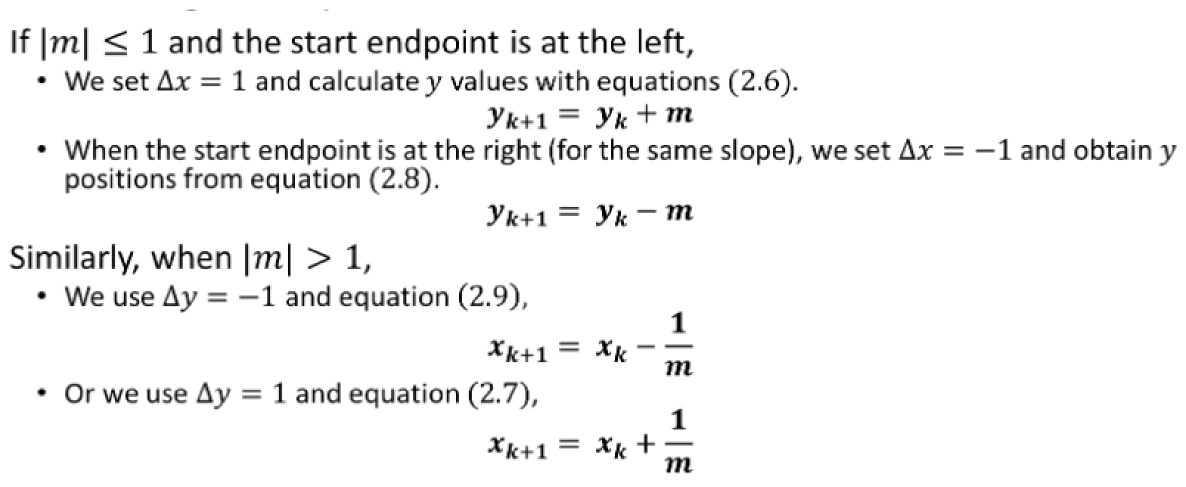
- Picture described as set of complex objects positioned at specific coordinate.
- Shapes and colors are described with set of geometric structure.
- The picture or scene is displayed by scan converting the geometric structure.

What's the basic concept behind point drawing ?

- Accomplished by converting a single coordinate by application program into appropriate operation for the output device in use.

$$(x, y) = (Round(x'),\quad Round(y'))$$

What's the basic concept behind Digital Differential Analyzer (DDA) algorithm ?



If $|m| \leq 1$ and the start endpoint is at the left,

- We set $\Delta x = 1$ and calculate y values with equations (2.6).

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

- When the start endpoint is at the right (for the same slope), we set $\Delta x = -1$ and obtain y positions from equation (2.8).

$$\boldsymbol{y_{k+1} = y_k - m}$$

Similarly, when $|m| > 1$,

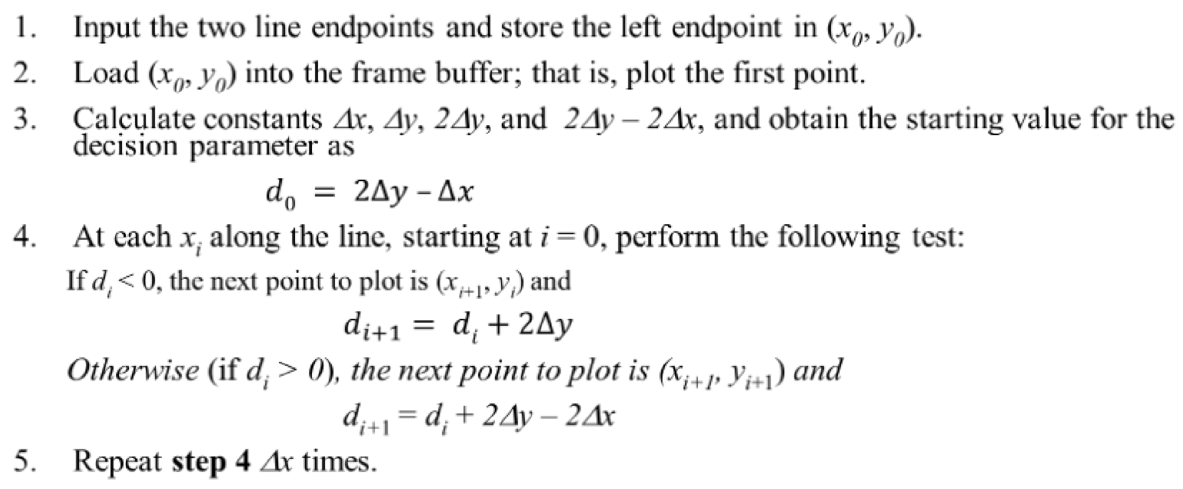
- We use $\Delta y = -1$ and equation (2.9),

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

- Or we use $\Delta y = 1$ and equation (2.7),

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

What’s the basic concept behind Bresenham’s Algorithm ?



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, $2\Delta y$, and $2\Delta y - 2\Delta x$, and obtain the starting value for the decision parameter as

$$d_0 = 2\Delta y - \Delta x$$

4. At each x_i along the line, starting at $i = 0$, perform the following test:

If $d_i < 0$, the next point to plot is (x_{i+1}, y_i) and

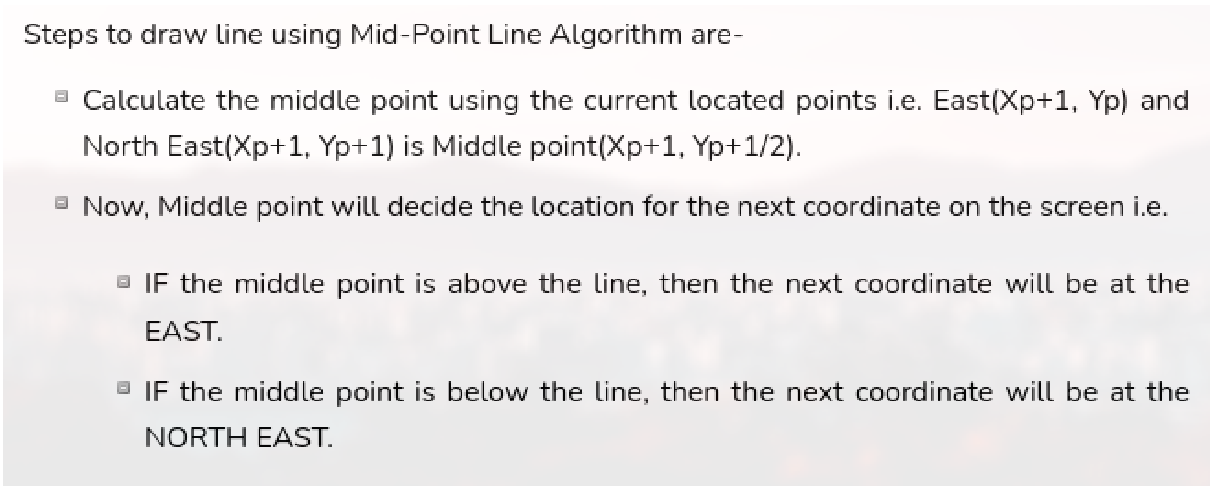
$$d_{i+1} = d_i + 2\Delta y$$

Otherwise (if $d_i > 0$), *the next point to plot is* (x_{i+1}, y_{i+1}) *and*

$$d_{i+1} = d_i + 2\Delta y - 2\Delta x$$

5. Repeat **step 4** Δx times.

What’s the basic concept behind the mid-point algorithm ?



Steps to draw line using Mid-Point Line Algorithm are-

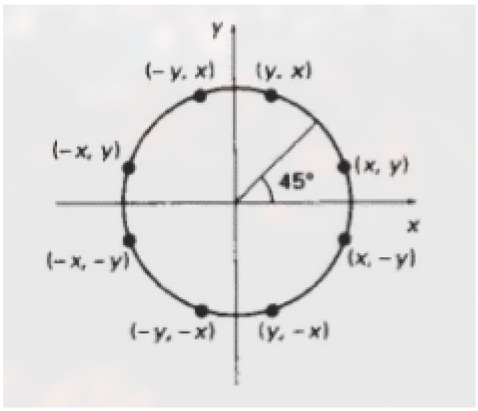
- Calculate the middle point using the current located points i.e. East(Xp+1, Yp) and North East(Xp+1, Yp+1) is Middle point(Xp+1, Yp+1/2).
- Now, Middle point will decide the location for the next coordinate on the screen i.e.
  - IF the middle point is above the line, then the next coordinate will be at the EAST.
  - IF the middle point is below the line, then the next coordinate will be at the NORTH EAST.

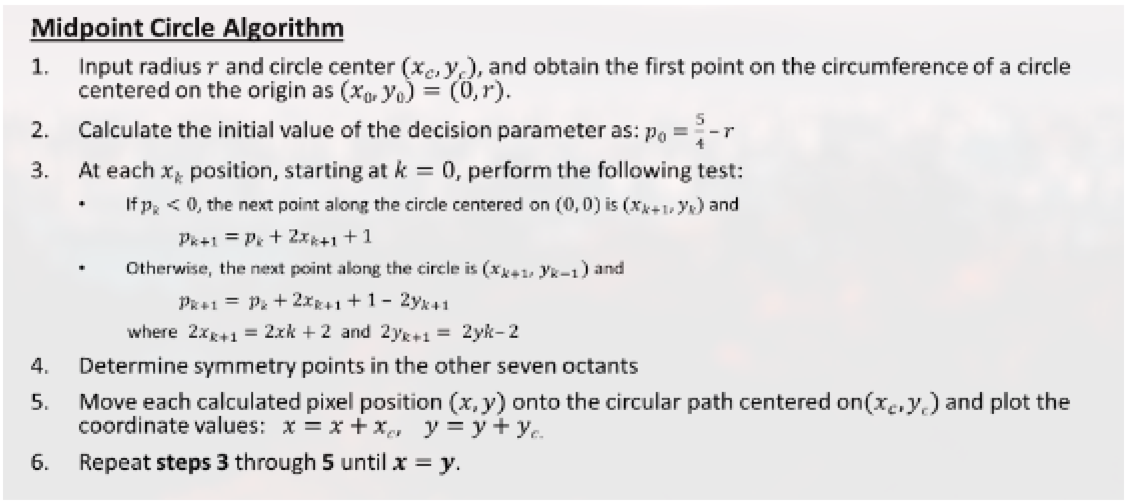
```
#include<bits/stdc++.h>
using namespace std;

void Mid_Point(int x_1, int y_1, int x_2, int y_2){
   int dx = x_2 - x_1;
   int dy = y_2 - y_1;

   if(dy <= dx){
      int d = dy - (dx / 2);
      int first_pt = x_1;
      int second_pt = y_1;

      cout<< first_pt << "," << second_pt << "\n";
      while(first_pt < x_2){
         first_pt++;
         if(d < 0){
            d = d + dy;
         }
         else{
            d = d + (dy - dx);
            second_pt++;
         }
            cout << first_pt << "," << second_pt << "\n";
      }
   }
   else if(dx < dy){
      int d = dx - (dy/2);
      int first_pt = x_1;
      int second_pt = y_1;
      cout << first_pt << "," << second_pt << "\n";
      while(second_pt < y_2){
         second_pt++;
         if(d < 0){
            d = d + dx;
         }
         else{
            d += (dx - dy);
            first_pt++;
         }
         cout << first_pt << "," << second_pt << "\n";
      }
   }
}

int main(){
   int x_1 = 3;
```

```
    int y_1 = 3;
    int x_2 = 10;
    int y_2 = 8;
    cout<<"Mid-Points through Line Generation Algorithm are: "
    Mid_Point(x_1, y_1, x_2, y_2);
    return 0;
 }
```

What’s the basic idea behind circle drawing algorithm ?

- The main concept behind Mid-Point Circle Drawing Algorithm is to determine the points on a circle using only the first octant of the circle. This can be done by finding the coordinates of the points on the circle using a midpoint calculation, based on the previous points found on the circle. For each pixel in the first octant, the algorithm finds the corresponding pixel in the other seven octants by using symmetry.





Midpoint Circle Algorithm

1. Input radius r and circle center (x_c, y_c), and obtain the first point on the circumference of a circle centered on the origin as $(x_0, y_0) = (0, r)$.
2. Calculate the initial value of the decision parameter as: $p_0 = \frac{5}{4} - r$
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 $2x_{k+1} = 2xk + 2$ and $2y_{k+1} = 2yk - 2$
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: $x = x + x_c,\quad y = y + y_c.$
6. Repeat **steps 3** through **5** until $\boldsymbol{x = y}$.

Note: Initial point p can be approximated to $1 - r$ since all increments are integers

```
void midPoint(int xc, int yc, int radius){
    // xc and yc are x and  y coordinates of center of t
  int x = 0, y = radius;
    int p = 1 - r;

  while(x < y){
     x++;
     if(p < 0) {
        p = p + (2 * x + 1);
     }else {
        y--;
        p = p + (2 * (x - y)) + 1;
     }
     plotAllPoints(x, y, xc, yc);
  }
}

void plotAllPoints(int x, int y, int xc, int yc) {
    draw(xc + x, yc + y);
    draw(xc + y, yc + x);
    draw(xc - y, yc + x);
    draw(xc - x, yc + y);
         draw(xc - x, yc - y);
    draw(xc - y, yc - x);
    draw(xc + x, yc - y);
    draw(xc + y, yc - x);
}
```

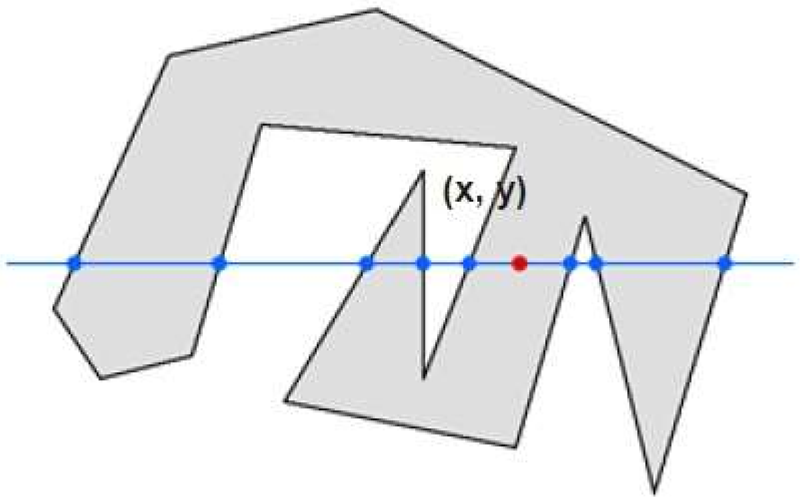
What is polygon and which algorithms used to draw polygon ?

- Polygons are areas enclosed by single closed loops of line segments.

Algorithms

1. Inside-Outside test

- **Odd-Even Rule**
    - In this technique, we will count the edge crossing along the line from any point *x*, *y* to infinity. If the number of interactions is odd, then the point *x*, *y* is an interior point; and if the number of interactions is even, then the point *x*, *y* is an exterior point. The following example depicts this concept.

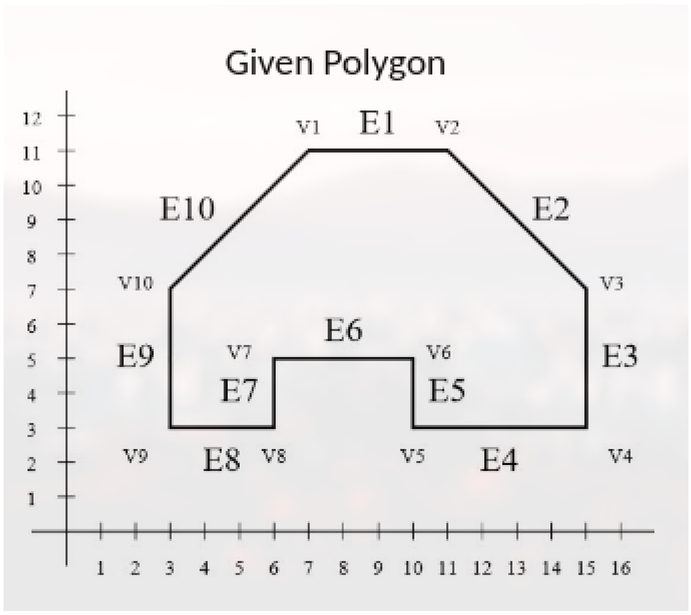


Point (x, y) is inside in the polygon because the number of intersecting is 5 in left and 3 on the right, therefore it satisfies Odd-Even rule.

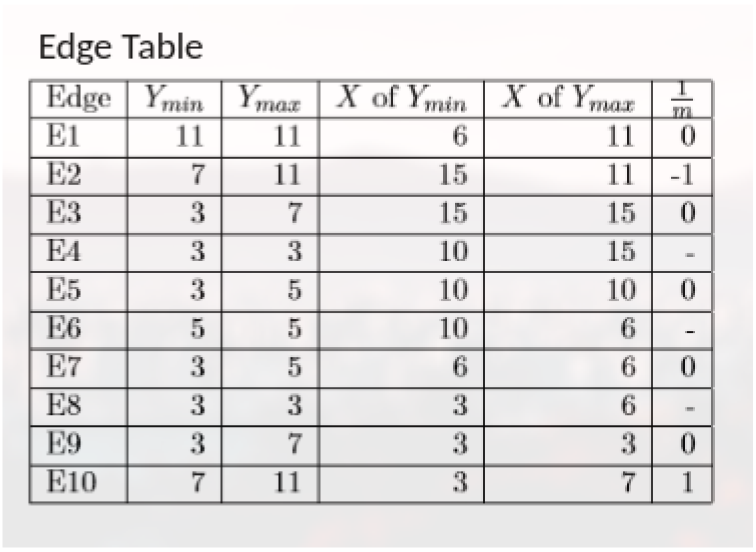
2. Scan line filling algorithm

Scan Line Filling Algorithm

- This algorithm is used to fill the color in the polygon. The polygon is considered to be formed by a set of line segments.
- **Steps**
    - The polygon is scanned from the top to bottom of y-coordinate in the polygon.



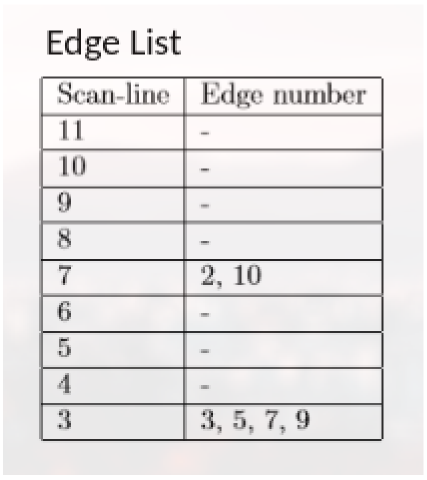
- First, the y-coordinate of the first line segment is noted.
- The edge table is used to store the information of the line segments of the polygon.



Edge Table

Edge	Y_{min}	Y_{max}	X of Y_{min}	X of Y_{max}	$\frac{1}{m}$
E1	11	11	6	11	0
E2	7	11	15	11	-1
E3	3	7	15	15	0
E4	3	3	10	15	-
E5	3	5	10	10	0
E6	5	5	10	6	-
E7	3	5	6	6	0
E8	3	3	3	6	-
E9	3	7	3	3	0
E10	7	11	3	7	1

- The active edge table is used to store the information of the edges that are intersecting the current scan line.


Edge List

Scan-line	Edge number
11	-
10	-
9	-
8	-
7	2, 10
6	-
5	-
4	-
3	3, 5, 7, 9

- The edges are sorted according to their x-coordinates of the intersection with the current scan line.
- The color of the polygon is filled between the intersections of the consecutive edges in the active edge table.
- The edges that are crossed completely are removed from the active edge table.
- The intersection of the remaining edges with the current scan line is computed and added to the active edge table.

3. Boundary filling algorithm

- Boundary filling algorithm is a filling algorithm that's used to fill the inside and border of the polygon with different colors.
- The algorithm works by starting at a boundary point and then filling all the interior points until a boundary point is reached again. The algorithm can use either the 4-connected or 8-connected approach to determine which pixels to fill.
- The 4-connected approach checks the four pixels in the north, south, east, and west directions, while the 8-connected approach checks all eight pixels that surround the current pixel. The algorithm can be implemented recursively or iteratively.

4. Flood fill algorithm

- The algorithm starts by selecting a point that is known to be inside the boundary of the polygon.
- The algorithm then fills the current pixel with a specified color.

- The algorithm then checks the neighboring pixels of the current pixel.
- If the neighboring pixel is not already filled with the specified color, the algorithm fills the neighboring pixel with the specified color and adds it to a list of pixels to be checked.
- The algorithm then moves to the next pixel in the list and repeats the process.
- The algorithm continues this process until all pixels inside the boundary have been filled with the specified color.

4-connected

- In 4-connected filling algorithm, only the four pixels in the north, south, east, and west are considered as neighboring pixels.

8-connected

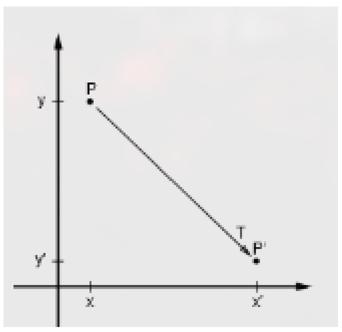
- In 8-connected filling algorithm, the eight pixels that surround the current pixel are considered as neighboring pixels.

5. Pattern fill
   - Uses $setpixel(x, y, pixmap[xmodm, ymodn])$ rather than $plotpixel(x, y)$ function to fill pixels with patterns.

Chapter 3: 2D Transformations

What are the basic 2D transformations ?

- Translation



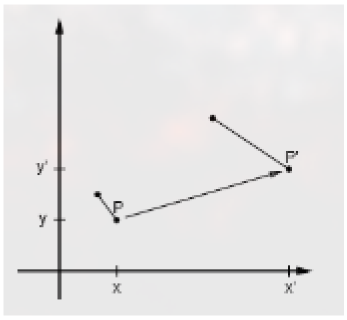
$P' = P + T$,

where $P = (x, y)$ is the original point,

$T = (t_x, t_y)$ is the translation coordinate and

$P' = (x + t_x, y + t_y)$ is the new coordinate

- Scaling



$P' = P.S$,

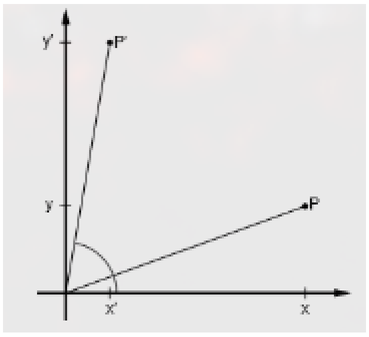
where $P = \begin{bmatrix} x \\ y \end{bmatrix}$ is the original point,

$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$ is the scaling matrix and

$P' = (x.s_x, y.s_y)$ is the new coordinate

  - If $sx = sy$ it's called uniform scaling
  - if $sx \neq sy$ it's called differential scaling

- Rotation



$P' = P.R$,

where $P = \begin{bmatrix} x \\ y \end{bmatrix}$ is the original point,

$R = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$ is the rotation matrix and

$P' = \begin{bmatrix} xcos\theta - ysin\theta \\ xsin\theta + ycos\theta \end{bmatrix}$ is the new coordinate

Note: Translation, Scaling and Rotaiona are called rigid body transformation.

What are homogenous coordinates, and how can we convert Cartesian coordinate to homogenous and vice versa ?

- Homogeneous coordinates are a mathematical tool used in computer graphics to represent geometric objects and transformations in a way that is independent of translation. They are a way of extending the Cartesian coordinate system to include points at infinity.
- To convert Cartesian coordinates to homogeneous coordinates, we add an extra dimension to the vector and set it to 1. For example, the point (x, y) would become (x, y, 1). To convert homogeneous coordinates back to Cartesian coordinates, we divide the first two elements of the vector by the third.

  For example, the homogeneous coordinate (2, 3, 1) would correspond to the Cartesian coordinate (2/1, 3/1), or (2, 3).

How can we apply basic transformations in homogenous coordinates ?

- Translation

  $P' = P + T$,

  where $P = \begin{bmatrix} x \\ y \\ W \end{bmatrix}$ is the original point,

  $T = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$ is the translation coordinate and

  $P' = \begin{bmatrix} x + t_x \\ y + t_y \\ W \end{bmatrix}$ is the new coordinate

- Scaling

$P' = P.S$,

where $P = \begin{bmatrix} x \\ y \\ W \end{bmatrix}$ is the original point,

$T = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$ is the translation coordinate and

$P' = \begin{bmatrix} x.s_x \\ y.s_y \\ W \end{bmatrix}$ is the new coordinate

- Rotation

$P' = P.R$,

where $P = \begin{bmatrix} x \\ y \\ W \end{bmatrix}$ is the original point,

$R = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$ is the rotation matrix and

$P' = \begin{bmatrix} xcos\theta - ysin\theta \\ xsin\theta + ycos\theta \\ W \end{bmatrix}$ is the new coordinate

How can we simplify composition of 2D transformations ?

- $T(t_{x1}, t_{y1}).T(t_{x2}, t_{y2}) = T(t_{x1} + t_{x2}, t_{y1} + t_{y2})$
- $S(s_{x1}, s_{y1}).S(s_{x2}, s_{y2}) = S(s_{x1}.s_{x2}, s_{y1}.s_{y2})$
- $R(\theta_1).R(\theta_2) = R(\theta_1 + \theta_2)$

How can we apply general fixed point scaling and rotation ?

Fixed point scaling

- $S_{x,y}(s_x, s_y) = T(x, y).S(s_x, s_y).T(-x, -y)$

$$= \begin{bmatrix} s_x & 0 & x(1-s_x) \\ 0 & s_y & y(1-s_y) \\ 0 & 0 & 1 \end{bmatrix}$$

Fixed point rotation

- $R_{x,y}(\theta) = T(x,y).R(\theta).T(-x,-y)$

$$= \begin{bmatrix} cos\theta & -sin\theta & x(1-cos\theta)+ysin\theta \\ sin\theta & cos\theta & y(1-scos\theta)+xsin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

What’s inverse of transformation ?

- If transformation a given 2D object is given by $P' = M.P$ then the inverse of transformation (original point) can be calculated

$$P = M^{-1}.P'$$

Translation

- $T(t_x,t_y)^{-1} = T(-t_x,-t_y)$

Scaling

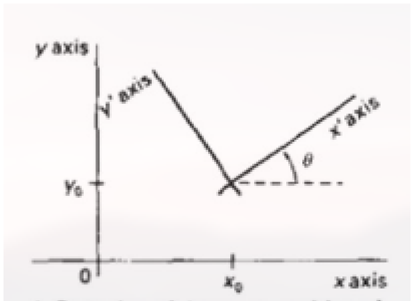
- $S(s_x,s_y)^{-1} = S(\frac{1}{s_x},\frac{1}{s_y})$

Rotation

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

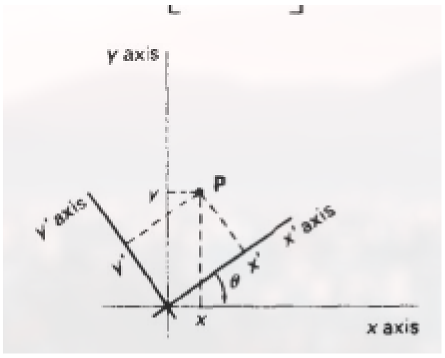
How can we convert one coordinate system to others ?

- Given 2 carthersian coordinates positioned and oriented differently, we can use the following steps to convert one to another.



A Cartesian $x'y'$ system positioned at (x_0, y_0) with orientation θ in an xy Cartesian system.

1. Translate so that the origin $(x_\circ, y_\circ)$ of the $x'y'$ system is moved to the origin of xy system.



2. Rotate the x' axis onto the x axis.

$$M_{xy,x'y'} = R(-\theta).T(-x,-y)$$

What are the basic concepts of 2D viewing ?

- Viewing:- involves transforming an object specified in a world coordinates frame of reference into a device coordinates frame of reference.
- Viewing pipeline:- steps required for transforming world coordinates to device coordinates.
- World coordinates:- carthesian coordinates in images and objects.
- Window:- region in the world coordinate system which is selected for display.
- View port:- region of a display device into which a window is mapped.

Steps in viewing pipeline

1. Transform world coordinates to into viewing coordinates.
2. Normalize viewing coordinates.
3. Transform the normalized viewing coordinates into device coordinates.

What are the basic concepts of clipping ?

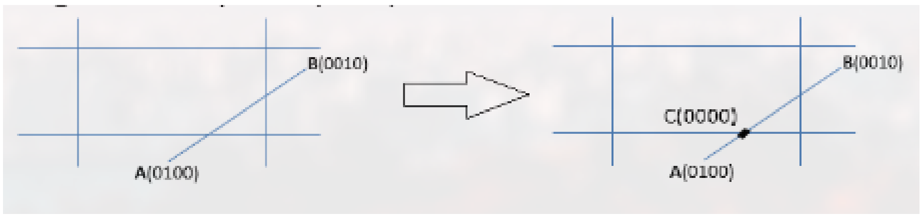
- Clipping window:- specific region of a picture needs to be displayed.
- Clipping algorithim:- an algorithim that displays only those primitives which lie inside a clip window.

- Clipping points:- For a point P (x, y) to be in clip window it must satisfy the following conditons.

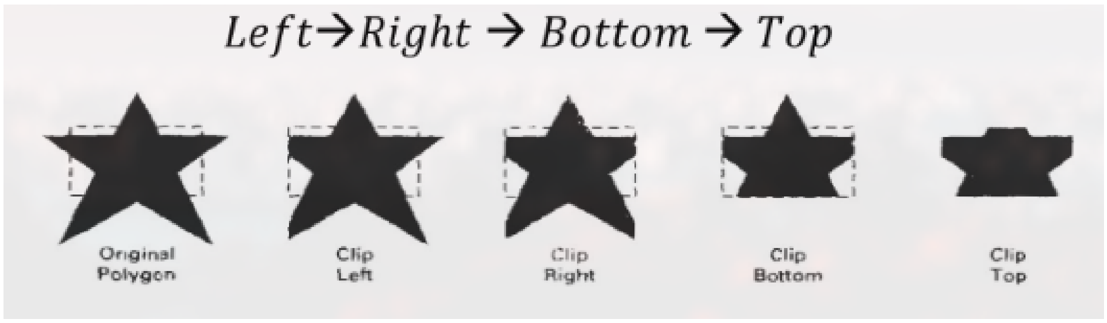
$$x_{min} \leq x \leq x_{max}$$
$$y_{min} \leq y \leq y_{max}$$

What are the basic algorithms used in 2D clipping ?

- Cohen-Sutherland line clipping algorithim
    - The algorithm works by assigning a 4-bit binary code to each endpoint of the line segment,based on its relative position to the clipping window. The bits in the
code represent whether the endpoint is to the left, right, above, or below the clipping window.If both endpoints have a code of 0000 (i.e., they are completely inside the clipping window), the line segment is accepted and no further
processing is necessary.
    - If one or both endpoints have a code containing a nonzero bit, the algorithm checks for intersection of the line segment with the edges of the clipping window, one edge at a time. If an intersection is found, the endpoint is replaced by the intersection point and its code is updated. The algorithm then repeats until both endpoints have codes of 0000 or the line segment is completely outside the clipping window.



- Sutherland-Hodgman polygon clipping algorithim
    - The algorithm works by iteratively clipping the polygon against each edge of the clipping region. At each iteration, the polygon is clipped against one edge of the clipping region, resulting in a new polygon with fewer vertices. This process is repeated for each edge of the clipping region until the entire polygon has been clipped.



Chapter 4: 3D Graphics

What are the basic 3D transformations ?

- Translation

$P' = P + T$,

where $P = \begin{bmatrix} x \\ y \\ z \\ W \end{bmatrix}$ is the original point,

$T = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$ is the translation coordinate and

$P' = \begin{bmatrix} x + t_x \\ y + t_y \\ z + t_z \\ W \end{bmatrix}$ is the new coordinate

- Scaling

$P' = P.S$,

where $P = \begin{bmatrix} x \\ y \\ z \\ W \end{bmatrix}$ is the original point,

$T = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ is the translation coordinate and

$$P' = \begin{bmatrix} x.s_x \\ y.s_y \\ z.s_z \\ W \end{bmatrix}$$ is the new coordinate

- Rotation

$P' = P.R,$

where $P = \begin{bmatrix} x \\ y \\ z \\ W \end{bmatrix}$ is the original point,

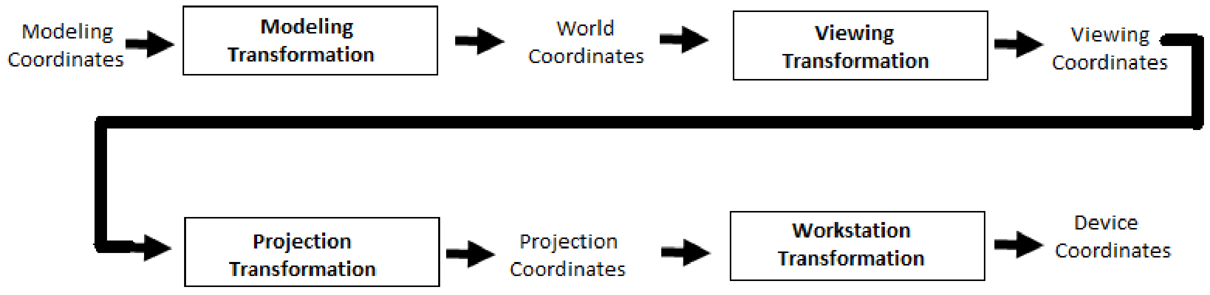
$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ is the rotation matrix about x-axis

$$R_y = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ is the rotation matrix about y-axis

$$R_z = \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ is the rotation matrix about z-axis

What are methods used to display 3D objects ?

1. Transforming world coordinates into viewing coordinates.
2. Transforming viewing coordinates into projection coordinates
3. Transforming projection coordinates to device coordinates.

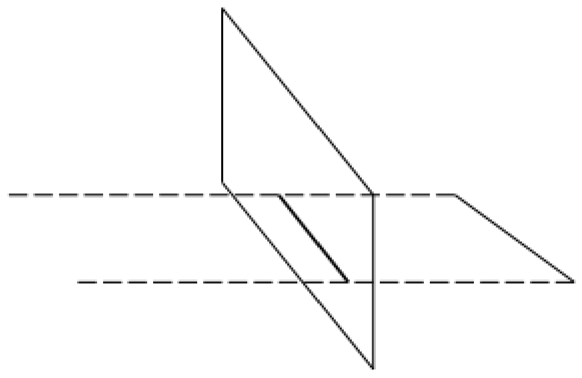


What’s projection ?

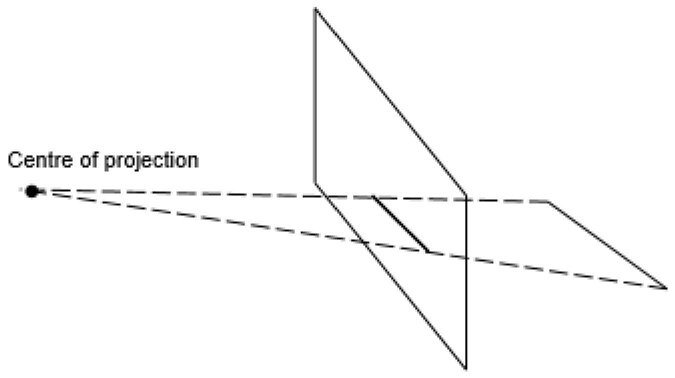
- Transformation of n dimensional space to m dimensional space where $m < n$.

What’s the difference between parallel and perspective projection ?

- Parallel projection and perspective projection are two different methods of projecting 3D objects onto a 2D plane in computer graphics.
- In parallel projection, the lines of sight from the viewer to the object are parallel. This means that objects at different depths in the 3D scene are projected onto the 2D plane with the same scale. Parallel projection is often used in engineering and architectural drawings, where it is important to maintain accurate measurements.
- In perspective projection, the lines of sight from the viewer to the object converge at a single point, called the vanishing point. This means that objects at different depths in the 3D scene are projected onto the 2D plane with different scales, depending on their distance from the viewer. Perspective projection is often used in art and animation, where the goal is to create a realistic sense of depth and distance.
- In general, perspective projection is considered more visually appealing and natural-looking, while parallel projection is considered more accurate and precise.



Parallel projection



Perspective projection

What are the subtypes of parallel projection ?

- Orthographic projection: This is the most common type of parallel projection, also known as a 'parallel projection in which all lines remain parallel'. In orthographic projection, the object is projected onto a plane parallel to the viewing plane. It is often used in technical drawings and architectural plans.
- Oblique projection: In this type of projection, the object is projected onto the viewing plane at an angle, rather than being parallel to the plane. This creates a distorted view of the object, but can be useful for certain types of drawings.