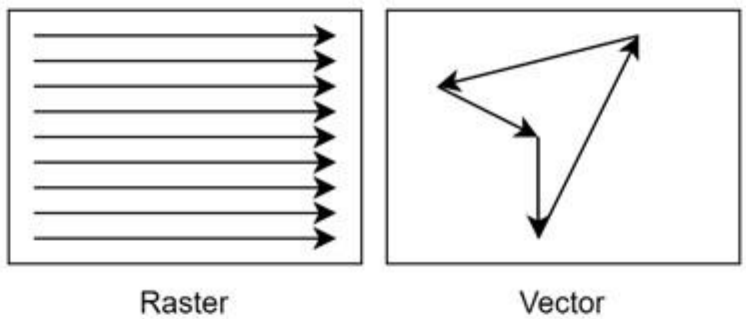
FAQs on Computer Graphics

There are some very **Frequently Asked Questions (FAQs)** on **Computer Graphics**, this section tries to answer them briefly.

1. What is the difference between Raster and Vector Graphics?

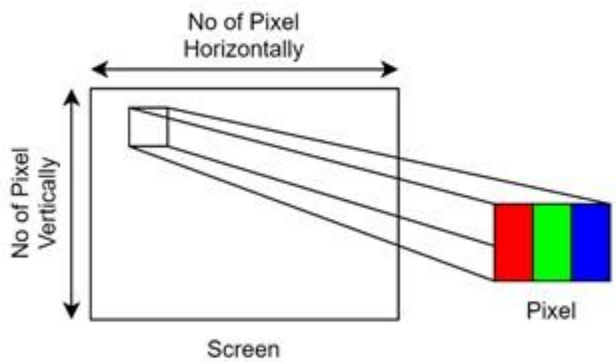
Raster and vector graphics are two common types of digital images.
Raster graphics consist of tiny pixels; they are ideal for detailed and colourful images like photographs. On the other hand, vector graphics use mathematical paths for scaled designs like logos and illustrations.



Raster images use bit maps to store information, requiring a larger bitmap for larger files. Vector graphics, on the other hand, use sequential commands or mathematical statements to place lines or shapes in a 2-D or 3-D environment.

2. What is a Pixel, and how does it relate to image resolution?

Pixels, derived from the term "picture element," are the fundamental unit of digital images, representing visual information. They consist of RGB subpixels, which mix to create diverse colors. Pixels form a raster image, with each pixel representing a specific position and color value.



In an image, how many pixels are there in row and in column manner that signifies the resolution. An image with large number of pixels inside it will be more accurate and detailed image as compared to lower number of picture elements.

3. What is Rendering in Computer Graphics?

The term **rendering** states visualizing the data. In computer graphics showing the generated graphics is rendering. However we generally use 3D rendering with the term "rendering".
3D rendering is a computer-based process that generates a 2D image from a digital 3D scene. It is essential in various industries like architecture, product design, advertising, video games, and visual effects.

4. What are the different types of rendering techniques?

Some of the rendering techniques are listed below −

- **Scanline** is a traditional rendering technique that calculates computer graphics surface.
- **Z-Buffer** is a two-dimensional data system used for depth calculation and storage.
- **Shading** and **lighting** manipulate light and dark levels using software effects.
- **Texture / Bump Mapping** displays color, material, and object details.
- **Ray Tracing** and **Ray Casting** are useful rendering techniques for creating natural lighting effects.

5. What is Graphics Pipeline?

Graphics Pipeline is a framework in computer graphics that outlines the procedures for converting a 3D scene into a 2D representation on a screen. It converts the model into a visually perceivable format on a computer display.
A universally applicable graphics pipeline is not available due to its dependence on specific software, hardware configurations, and display attributes. Instead, graphics application programming interfaces (APIs) like Direct3D, OpenGL, and Vulkan standardize common procedures and oversee the graphics pipeline of a hardware accelerator.

6. How do display devices like CRT and LCD work in the context of graphics?

CRT monitors use electron beams to glow up phosphor dots on the screen. Like this images are formed. On the other hand, LCD monitors use liquid crystals and a backlight to control light and form images.
Both types rely on the graphics card to render and send the image signal. CRTs use an analog signal, while LCDs use a digital signal. The graphics card and monitor must support the same resolution and refresh rate for proper display.

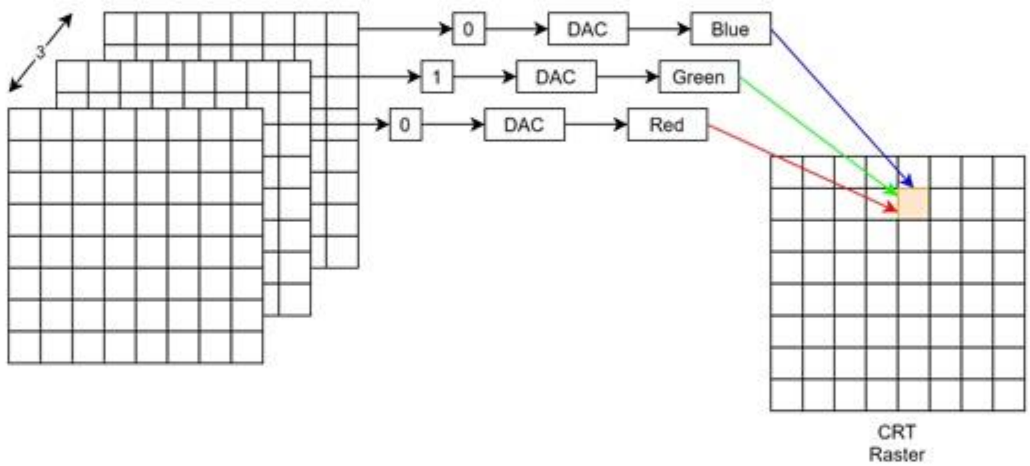
7. What is the role of linear algebra in computer graphics?

Linear algebra is like the math toolkit for computer graphics. It helps in creating and moving objects in 2D or 3D on the screen. For example, it allows us to rotate, scale, and shift objects. Think of it as the math behind making video games and animations look real.

Without linear algebra, it will not be possible to create the graphics. From the linear equation, we form matrices to solve the problems in action.

8. What is a Frame Buffer?

A frame buffer is a memory buffer in RAM that contains a bitmap that drives a video display. It represents all pixels in a complete video frame and converts an in-memory bitmap into a video signal for display on a computer monitor.
The buffer typically contains color values for every pixel, stored in 1-bit binary, 4-bit palettized, 8-bit palettized, 16-bit high color, and 24-bit true color formats. The total memory required depends on the output signal resolution and color depth.



9. How do Spatial Data Structures improve rendering efficiency?

Spatial data structures like grids, octrees, and BSP trees, help organize 3D objects in a scene. They make rendering faster by reducing the number of calculations needed to determine what is visible.
For example, instead of checking every object, the system only checks those in relevant areas. This speeds up processes like ray tracing, making graphics smoother and more efficient. Also for object which comes front or rear of some other objects, these data structures helps us a lot.

10. What is the significance of Color Depth?

Color depth, also known as bit depth, is the number of bits used to indicate the color of a single pixel or each color component of a single pixel. It can be defined as bits per pixel (bpp), bits per component, bits per channel, bits per color (bpc), bits per pixel component, bits per color channel, or bits per sample (bps).
Modern standards typically use bits per component, while historical lower-depth systems used bits per pixel more often. Color precision and gamut are defined through a color encoding specification.

11. What is anti-aliasing, and why is it important?

Antialiasing is a technique in computer graphics that removes the aliasing effect, which results in jagged edges in rasterized images. This issue arises from **undersampling**, a distortion caused by low-frequency scan conversion.
Aliasing occurs when smooth, continuous curves are rasterized using pixels, and the cause of anti-aliasing is undersampling.

After applying anti-aliasing, the images turns to be smoothed and produces more clear lines which makes the picture much more effective in action.

12. What is the difference between 2D and 3D graphics?

2D and 3D graphics differ in their creation and outcome. 2D graphics consist of height and width, while 3D graphics add depth for realism.
2D graphics are commonly used in animation and video games, providing a flat view of movement on the screen. 3D graphics offer realistic depth, allowing viewers to see into spaces, observe light and shadow movement, and gain a fuller understanding of the content. They work with the brain's natural tendency to explore and enrich our understanding of the world.

13. What are Homogeneous Coordinates?

Homogeneous coordinates are fairly used in computer graphics. This serves as the basis for geometry used to display three-dimensional objects on two-dimensional image planes. They provide a standard for performing operations on points in Euclidean space, such as matrix multiplication.
These coordinate systems are used in two ways: by adding an extra value, and by solving problems in representing and implementing geometric object transformations. Most graphics are represented by matrices, which are applied to vectors in cartesian form.

14. What are Transformation Matrices in computer graphics?

A transformation matrix is a square matrix that represents a linear transformation in vector space. It transforms a coordinate system from one to another by maintaining the linearity attribute of the space.
The matrix carries coefficients, making it easier to compute and change geometrical objects.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

For example, in a 2D coordinate system with coordinate vectors i and j, a transformation matrix **T** can transform a vector **v = (x, y)** into a vector **w = (x', y')**, creating a new coordinate system. The matrix's coefficients determine the directions of transformation and are used to find the vector **w**.

15. How do Translation, Scaling, and Rotation work in 2D / 3D transformations?

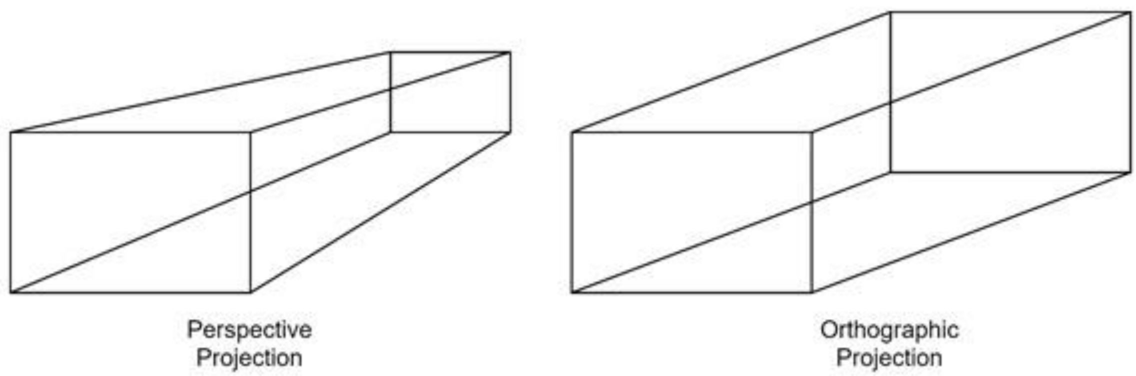
Translation, scaling, and rotation are key techniques used in 2D and 3D transformations to alter an object's position, orientation, and size.
We can form a matrix which adds **x**, **y**, and **z** parameters to translate to another coordinate.
Scaling resizes it in coordinate using scaling factors along the X, Y, and Z directions. So here with scaling matrix, we use the scaling factor for each axis.
For rotation, we use trigonometric functions with sine and cosine. Rotation changes an object's orientation around one or more axes, either around the center of the grid or along a line.

16. What is a View Matrix in graphics rendering?

The view matrix is a matrix used to convert vertices from world-space to view-space, typically concatenated with the object's world and projection matrix. This allows vertices to be directly transformed from object-space to clip-space in the vertex program.
If **M** represents the object's world matrix, **V** represents the view matrix, and **P** is the projection matrix, the concatenated world, view, and projection can be represented by MVP.

17. What is Perspective Projection? How is it different from Orthographic Projection?

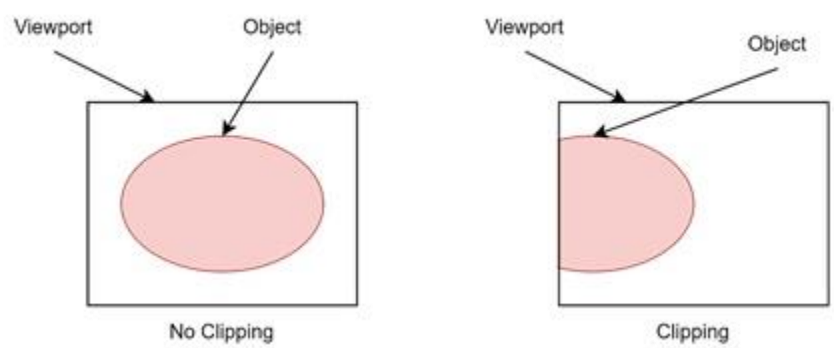
Orthographic projections are parallel projections that can be represented by an affine transformation. Perspective projections are not parallel projections. They are useful for artistic and technical reasons, such as verifying part fit in floor plans or representing problems in different basis for easier coordinates.
In the following example, we can see for perspective projections, we are assuming the lines will meet at a point; but for orthographic, it will meet at infinite distance.



Orthographic projections are commonly used in CAD drawings and technical documentations to ensure dimensions are easy to measure and to represent problems in different bases for easier understanding.

18. What is the concept of Clipping in graphics?

To display a large portion of a picture, scaling and translation are necessary, along with identifying the visible part. This process is challenging as some parts are partially inside, and partially visible lines or elements are omitted.



Clipping is a process used to determine the visible and invisible portions of each element, with visible portions selected and invisible ones discarded. There are various types of clipping, including line and polygon clipping.

19. What are vectors and how are they used in computer graphics?

Vector graphics are a versatile and scalable computer graphics approach that uses mathematical equations and geometric shapes instead of pixel-based raster graphics. They have no resolution limit and retain image quality at any size.
Graphic artists preserve their work as vector declarations. The most prevalent vector graphic format is SVG, and they are based on analytical or coordinate geometry.
Ivan Sutherland invented vector graphics in 1963 and developed the first vector graphics editor, Sketchpad, which allowed users to create and edit vector graphics on a computer screen.

20. What is Ray Tracing, and how does it differ from Rasterization?

Ray tracing is a technique used in real-time computer graphics to display three-dimensional objects on a two-dimensional screen. It involves observing the path of light beams from your eye to the objects that light interacts with.
Rasterization, on the other hand, is a technique used to display three-dimensional objects on a two-dimensional screen. It creates 3D models of objects from a mesh of virtual triangles, where each triangle's vertices intersect with other triangles of different sizes and shapes. This information, including position, color, texture, and "normal," determines the object's surface orientation.

21. What is the role of normal vectors in lighting calculations?

Normal vectors are used in lighting calculations for 3D graphics. They indicate surface orientation at each point. This is needed for light reflection and shading. These vectors are essential for calculating diffuse and specular reflections, creating realistic highlights, and implementing techniques like bump mapping.
In shader programs, normal vectors are key inputs for various lighting models. They allow for efficient creation of complex lighting effects without requiring excessive geometric detail, making them necessary for real-time 3D rendering and smooth lighting across surfaces.

22. What is the significance of Depth Buffers in 3D rendering?

Depth buffers are crucial in 3D rendering for determining visibility of objects and surfaces. Without them, rendering order can cause incorrect layering of faces.
For a simple cube as well, if the system does not know proper ordering it will display each of the faces but they will be in any order and face hidden behind may be visible in front. To applying face culling, it needs depth buffer with it.

23. What is Bresenham's line-drawing algorithm?

Bresenham's line algorithm is a n-dimensional raster drawing algorithm that selects points to form a close approximation to a straight line between two points. It is commonly used for drawing line primitives in bitmap images, as it uses inexpensive operations like integer addition, subtraction, and bit shifting.
As an incremental error algorithm, it is one of the earliest in computer graphics. An extension of this algorithm is called the midpoint circle algorithm can be used for drawing circles.

24. What is Scanline Algorithm, and how is it used in raster graphics?

The Scan-line algorithm is an algorithm that is used in 3D graphics for hidden surface removal. This algorithm processes one line at a time, rather than processing one pixel at a time.

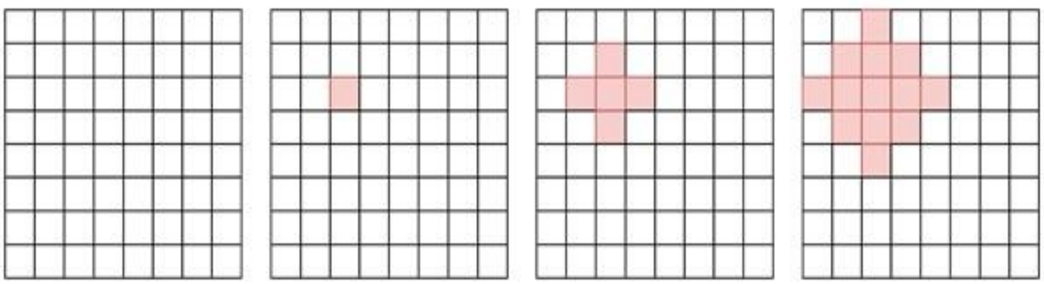
The Scanline algorithm examines points where the scan-line intersects polygon surfaces from left to right, performs depth calculations to identify hidden regions. Then, it updates color-intensity values in the refresh buffer if the flag of the corresponding surface is on.

25. What is Midpoint Circle Algorithm?

The midpoint circle algorithm is a method used in computer graphics to draw circles efficiently. It is an extension of Bresenham's line drawing algorithm. It works by determining which pixels should be colored to form a circle on a digital screen.
Starting from one point, the algorithm moves along the circle's circumference, deciding whether to move straight or diagonally to the next pixel. It makes these decisions by calculating the midpoint between two possible pixels. This approach avoids using complex trigonometric calculations, making it faster and easier to implement.

26. How does the Flood-Fill Algorithm work?

The flood ffill algorithm is a method used to fill multiple color boundaries in a region. It starts with a seed point inside the region and uses four or eight connected approaches to fill it with the desired color.



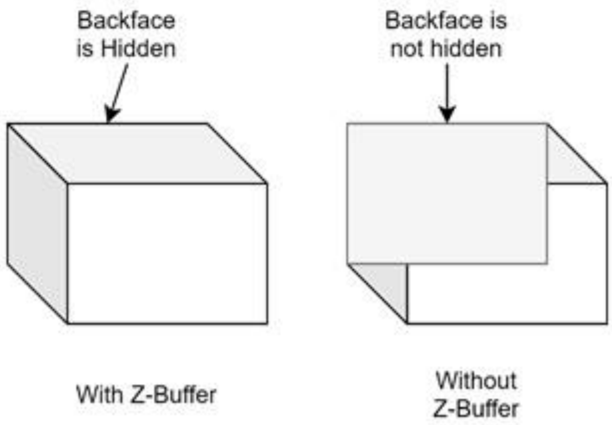
Flood-fill algorithm is more suitable for filling multiple color boundaries and interiors with one color. The fill algorithm starts at a specified interior point (x, y) and reassigns all pixel values to the desired color. It then steps through pixel positions until all interior points are repainted.

27. What is Texture Mapping?

A texture map is an image applied to the surface of a shape or polygon, either bitmap or procedural. They can be stored in image file formats, referenced by 3D models, or assembled into resource bundles. They can have one to three dimensions, with two dimensions being most common for visible surfaces.
Modern hardware may store texture map data in swizzled or tiled orderings for improved cache coherency. Rendering APIs manage texture map resources as buffers or surfaces, allowing 'render to texture' for additional effects.

28. What is the purpose of Z-buffer in 3D rendering?

Z-Buffer is nothing but a type of depth buffer and it stores the depth information of different objects in 3D scene.
Which object is present in front, which one in back these are determined by the Z-buffer. This is essential to represent 3D objects in correct way. Otherwise, some face which are not placed in back will come front and generate invalid images.



29. What is Gouraud Shading, and how does it differ from Phong Shading?

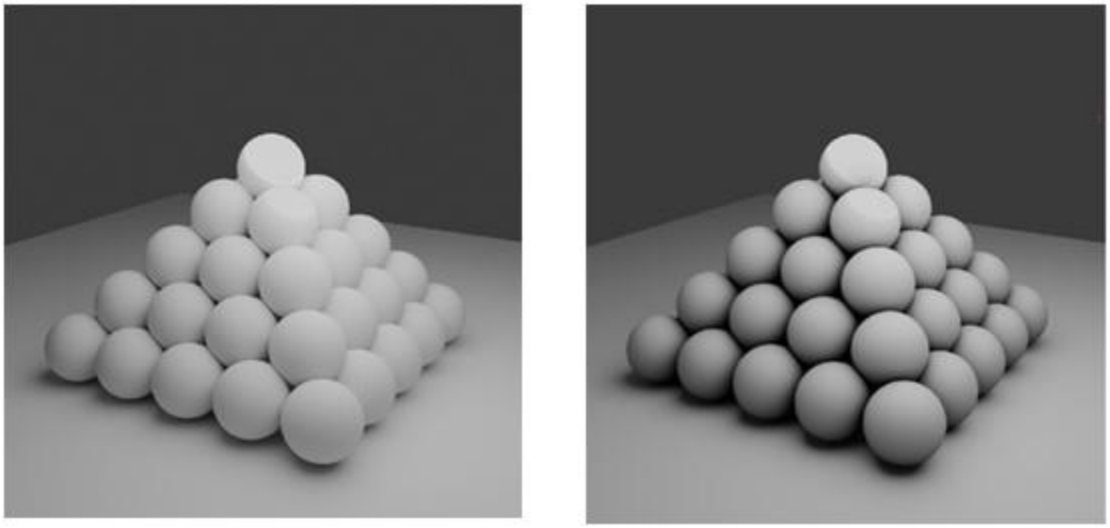
Gouraud shading calculates lighting at vertices and interpolates colors across polygons. This method is computationally efficient but can produce visible artifacts, especially with specular highlights, which may appear angular or jump between vertices as the view changes. The quality heavily depends on model complexity.

Phong shading, on the other hand, interpolates lighting parameters across polygons and calculates lighting per fragment. This approach produces smoother, more realistic results, particularly for specular highlights, which appear round and move fluidly across surfaces.

30. What is Ambient Occlusion, and how is it used in rendering?

Ambient occlusion is a type of lighting technique used in 3D graphics. This adds subtle shadows to make scenes look more realistic.

In the following figures, it will be clearer. It simulates how objects block light in the real world, darkening areas where objects are close together or in corners.



No Ambient Occlusion Added Ambient Occlusion Added

This technique calculates how much ambient light each surface point can "see," creating softer shadows in enclosed spaces. This significantly improves the overall visual quality and realism of 3D rendered scenes.

31. What are the key differences between a Shader and a Texture?

Shaders and textures are two important components in 3D graphics. These are used in different purposes. Shaders are GPU-run programs that control rendering of 3D objects, influencing lighting, color, and special effects. Shaders can create complex visual effects and are highly flexible, allowing dynamic changes based on inputs.

Textures, on the other hand, are images applied to 3D models to add detail and realism, providing surface information like color patterns, roughness, or bump effects.

32. What are the basic principles of animation in computer graphics?

The animation in computer graphics is nothing but a sequence of frames where each frame is generating a graphics output on the scene. Generally, not all frames are generated in a similar way.
The concept of **interpolation** is used for animation creation. These can be done with linear, quadratic or other interpolation types. There are several types of animation including **key-framing, morphing, tween animations**, etc.

33. What is interpolation in computer graphics?

Interpolation techniques are used in computer graphics for animation creation. These are used to make smooth animation by taking two frames and generating intermediate frames for smooth transition.
There are several types of interpolations including linear, quadratic, etc. These can be used for movement rotation, scaling animations along with morphing and some other techniques like character animation (Walking, running) etc.

34. What is a Scene Graph?

In computer graphics, the scene graph is a data structure used in vector-based graphics editing applications and modern computer games to arrange a logical representation of a graphical scene. It consists of nodes in a tree structure, with the effect of a parent applied to all its child nodes.
In many programs, a geometrical transformation matrix is used to efficiently process operations, such as grouping related shapes and objects into a compound object that can be manipulated as a single object.

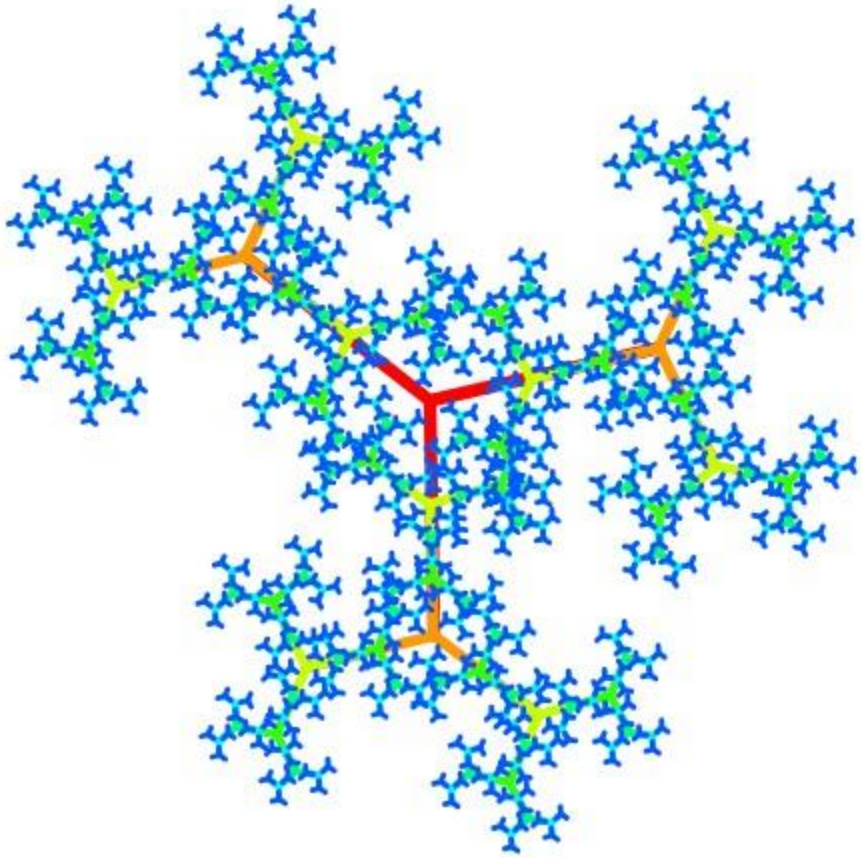
35. What is Global Illumination in computer graphics?

Global illumination (GI) is a set of 3D computer graphics algorithms that enhance the realistic lighting in 3D scenes. These algorithms consider both direct and indirect light sources including reflections, refractions, and shadows. They simulate these effects, affecting the rendering of other objects.

However, in practice, only diffuse inter-reflection or caustics are considered global illumination, as they simulate the effects of light rays from the same source on other surfaces in the scene.

36. How do Fractals relate to computer graphics?

Fractals are complex images created by a computer using iterations, based on a single formula. Geometric fractals are generated with non-integer shapes found in nature. These are created by starting with an initiator and replacing parts with a generator, resulting in a deterministic, non-random self-similar fractal.



37. How do graphics techniques differ for real-time vs Offline Rendering?

Offline rendering is a traditional method of creating images or animations by computing every pixel in a scene before displaying the final result. It uses powerful software like Autodesk 3ds Max, V-Ray, or Blender Cycles etc.

Real-time rendering, on the other hand, is a fast-paced method used in video games and interactive applications, and is increasingly gaining popularity in architectural visualization.

38. What is a GPU, and how does it accelerate graphic rendering?

A GPU is a specialized circuit that efficiently processes and manipulates computer graphics and image data. This offloads compute-intensive tasks from the CPU, enabling the CPU to handle more general computational workloads.

GPUs are used in gaming and 3D graphics rendering. It helps graphics processing by real-time rendering of complex environments and high-resolution textures, handling transformations, vertex generation, pixel shaders, antialiasing etc.

39. What is OpenGL, and why is it widely used in graphics development?

OpenGL is an API related to graphics. That is used to communicate with graphic hardware (GPU). It provides a graphical representation of an application.

OpenGL allows for 2D and 3D vector graphics to be rendered through hardware acceleration, making it widely used in VR, CAD, and games. OpenGL is cross-platform and language-independent.

40. How does DirectX compare to OpenGL?

OpenGL and DirectX are not similar in terms of their functionality. DirectX is a suite of APIs, including Direct3D and Direct2D, which are only compatible with Microsoft platforms like Windows and Xbox. OpenGL is cross-platform and compatible with Microsoft, Apple, and Linux systems.

Apple has deprecated OpenGL; however on macOS and iOS, it is still supported. In terms of gaming frame rate and system resource use, OpenGL and DirectX 11 are almost on par, with little noticeable difference in most cases.

41. What is the concept of LOD (Level of Detail) in 3D rendering?

Level of detail (LOD) in computer graphics says the complexity of a 3D model representation. It can be decreased as the model moves away from the viewer or based on metrics like object importance or viewpoint-relative speed.

LOD techniques increase rendering efficiency by reducing workload on graphics pipeline stages, usually vertex transformations. LOD is often applied to geometry detail but can be generalized to include shader management and mipmapping for higher rendering quality.

42. What is a Game Engine, and how does it relate to computer graphics?

A game engine is software that helps create video games. It handles many tasks like rendering graphics, processing physics, managing audio, and handling user inputs.

In computer graphics, the game engine’s graphics engine is important. It renders 2D and 3D visuals, ensuring smooth and realistic images. This allows game developers to focus on designing the game rather than building graphics from scratch.

Some of the popular game engines include **Unity**, **Unreal Engine**, and **Godot**.

43. What are some common optimization techniques used in real-time graphics?

Optimization techniques are used to enhance the performance and make the system faster in different applications by removing unnecessary computations. There are many such optimization techniques, including the following −

- **Instancing** − Reuses the same object multiple times with different transformations, reducing the number of draw calls.
- **Culling** − Skips rendering objects not visible to the camera, saving processing power.
- **Level-of-Detail (LOD) Management** − Uses simpler models for distant objects to maintain performance without sacrificing visual quality.

44. How do graphics techniques for VR differ from traditional 3D graphics?

The technology for VR is different. In VR, we use stereoscopic images where two images merge together to get a realistic 3D view. On the other hand, traditional 3D images are just 2D render, where no actual 3D feeling comes.

We can say VR graphics differ in terms of immersion, performance, field of view, and interaction. VR aims for a fully immersive experience. Traditional 3D graphics focus on visual representation. High frame rates and low latency are required for smooth interaction, while a wider field of view mimics human vision.

Computer Graphics:

Computer graphics is an art of drawing pictures on computer screens with the help of programming. It involves computations, creation, and manipulation of data. In other words, we can say that computer graphics is a rendering tool for the generation and manipulation of images.

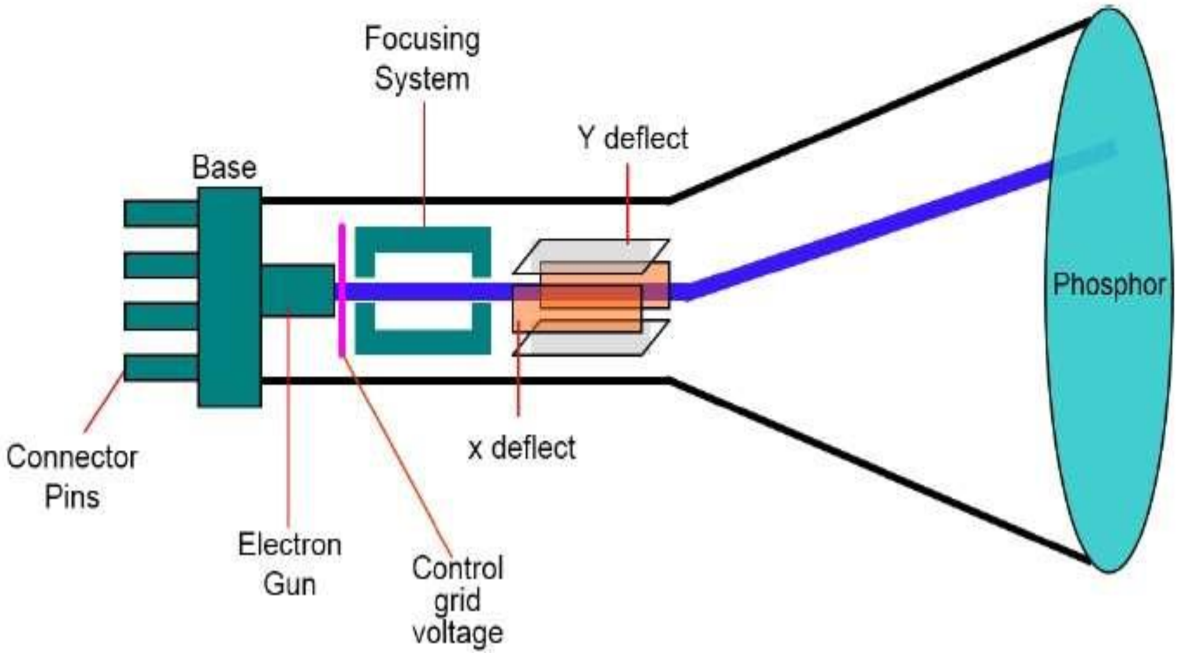
Cathode Ray Tube

The primary output device in a graphical system is the video monitor. The main element of a video monitor is the **Cathode Ray Tube (CRT),** shown in the following illustration.

The operation of CRT is very simple −

- The electron gun emits a beam of electrons (cathode rays).
- The electron beam passes through focusing and deflection systems that direct it towards specified positions on the phosphor-coated screen.

- When the beam hits the screen, the phosphor emits a small spot of light at each position contacted by the electron beam.
- It redraws the picture by directing the electron beam back over the same screen points quickly.



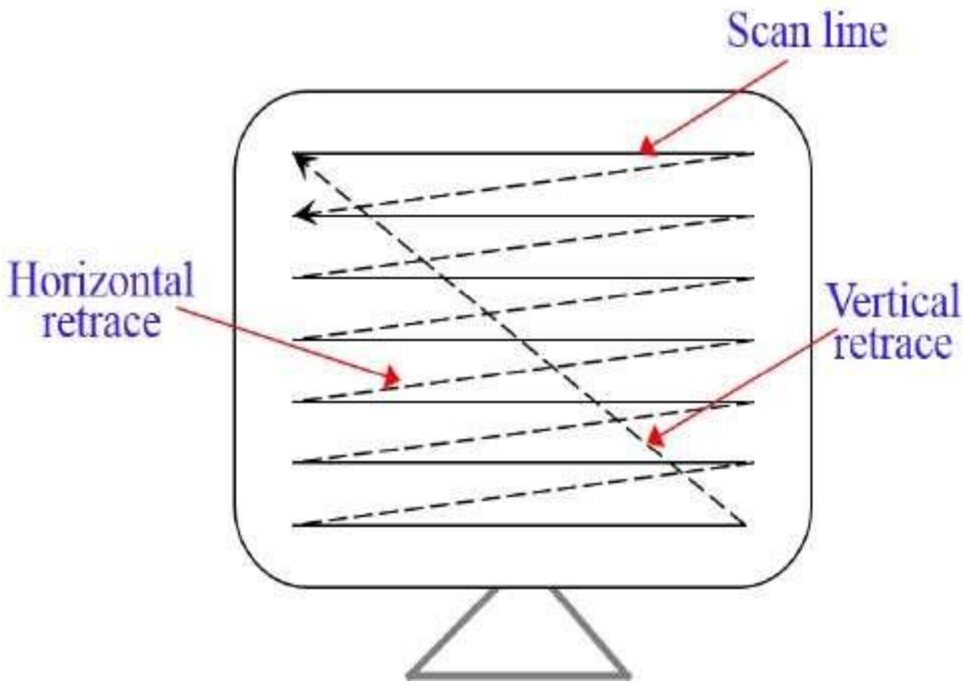
There are two ways (Random scan and Raster scan) by which we can display an object on the screen.

Raster Scan

In a raster scan system, the electron beam is swept across the screen, one row at a time from top to bottom. As the electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots.

Picture definition is stored in memory area called the **Refresh Buffer** or **Frame Buffer**. This memory area holds the set of intensity values for all the screen points. Stored intensity values are then retrieved from the refresh buffer and "painted" on the screen one row (scan line) at a time as shown in the following illustration.

Each screen point is referred to as a **pixel (picture element)** or **pel**. At the end of each scan line, the electron beam returns to the left side of the screen to begin displaying the next scan line.

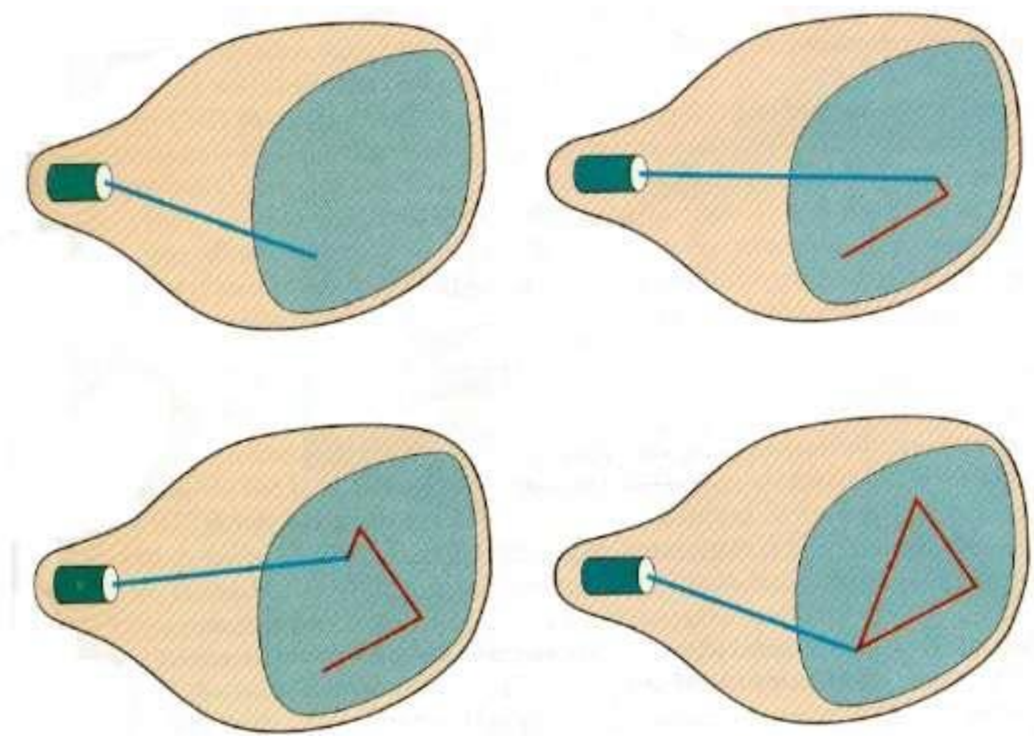


Random Scan (Vector Scan)

In this technique, the electron beam is directed only to the part of the screen where the picture is to be drawn rather than scanning from left to right and top to bottom as in raster scan. It is also called **vector display, stroke-writing display,** or **calligraphic display**.

Picture definition is stored as a set of line-drawing commands in an area of memory referred to as the **refresh display file**. To display a specified picture, the system cycles through the set of commands in the display file, drawing each component line in turn. After all the line-drawing commands are processed, the system cycles back to the first line command in the list.

Random-scan displays are designed to draw all the component lines of a picture 30 to 60 times each second.



Application of Computer Graphics

Computer Graphics has numerous applications, some of which are listed below

- **Computer graphics user interfaces (GUIs)** − A graphic, mouse-oriented paradigm which allows the user to interact with a computer.
- **Business presentation graphics** − "A picture is worth a thousand words".
- **Cartography** − Drawing maps.
- **Weather Maps** − Real-time mapping, symbolic representations.
- **Satellite Imaging** − Geodesic images.
- **Photo Enhancement** − Sharpening blurred photos.
- **Medical imaging** − MRIs, CAT scans, etc. - Non-invasive internal examination.
- **Engineering drawings** − mechanical, electrical, civil, etc. - Replacing the blueprints of the past.
- **Typography** − The use of character images in publishing - replacing the hard type of the past.
- **Architecture** − Construction plans, exterior sketches - replacing the blueprints and hand drawings of the past.
- **Art** − Computers provide a new medium for artists.
- **Training** − Flight simulators, computer aided instruction, etc.
- **Entertainment** − Movies and games.
- **Simulation and modeling** − Replacing physical modeling and enactments

Line Generation Algorithm

A line connects two points. It is a basic element in graphics. To draw a line, you need two points between which you can draw a line. In the following three algorithms, we refer the one point of line as X_0, Y_0 and the second point of line as X_1, Y_1.

DDA Algorithm

Digital Differential Analyzer (DDA) algorithm is the simple line generation algorithm which is explained step by step here.

Step 1 − Get the input of two end points (X_0, Y_0) and (X_1, Y_1).

Step 2 − Calculate the difference between two end points.

dx = $X_1 - X_0$

dy = $Y_1 - Y_0$

Step 3 − Based on the calculated difference in step-2, you need to identify the number of steps to put pixel. If dx > dy, then you need more steps in x coordinate; otherwise in y coordinate.

```
if (absolute(dx) > absolute(dy))
   Steps = absolute(dx);
else
   Steps = absolute(dy);
```

Step 4 − Calculate the increment in x coordinate and y coordinate.

Xincrement = dx / (float) steps;

Yincrement = dy / (float) steps;

Step 5 − Put the pixel by successfully incrementing x and y coordinates accordingly and complete the drawing of the line.

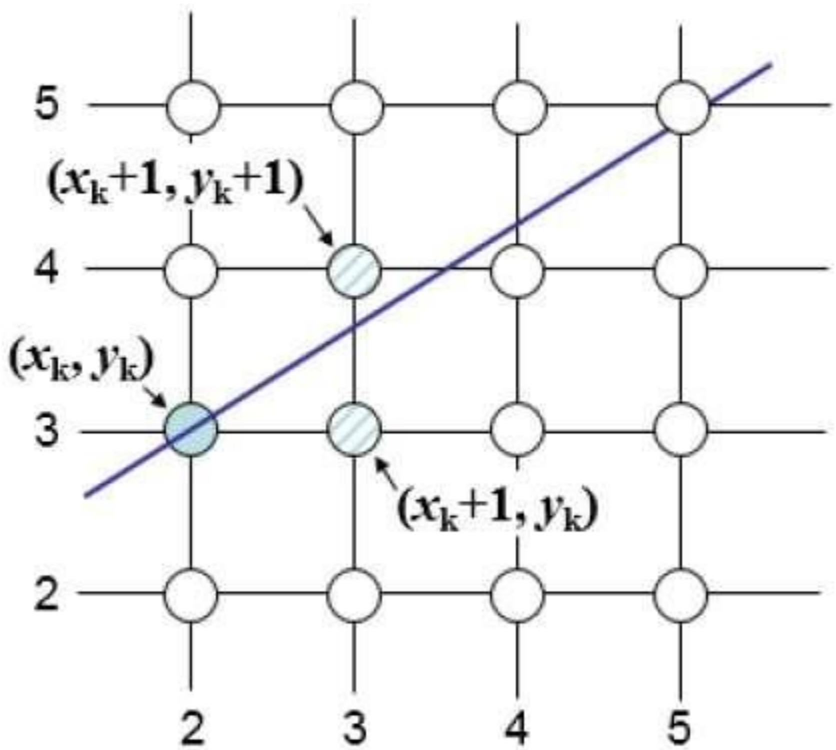
```
for(int v=0; v < Steps; v++)
```

```
{
   x = x + Xincrement;
   y = y + Yincrement;
   putpixel(Round(x), Round(y));
}
```

Bresenham's Line Generation

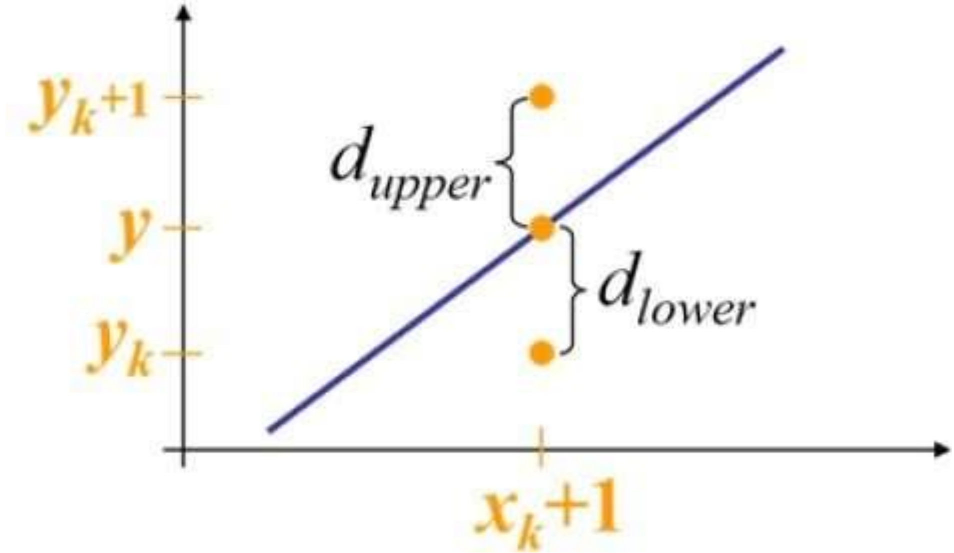
The Bresenham algorithm is another incremental scan conversion algorithm. The big advantage of this algorithm is that, it uses only integer calculations. Moving across the x axis in unit intervals and at each step choose between two different y coordinates.

For example, as shown in the following illustration, from position (2, 3) you need to choose between (3, 3) and (3, 4). You would like the point that is closer to the original line.



At sample position X_k+1,X_{k+1}, the vertical separations from the mathematical

line are labelled as $d_{upper}$$d_{upper}$ and $d_{lower}$$d_{lower}$.



From the above illustration, the y coordinate on the mathematical line at x_k+1xk+1 is –

Y = m(X_kXk+1) + b

So, d_{upper}dupper and d_{lower}dlower are given as follows –

$$d_{lower}=y-y_k$$ dlower=y−yk

$$=m(X_k+1)+b-Y_k$$ =m(Xk+1)+b−Yk

and

$$d_{upper}=(y_k+1)-y$$ dupper=(yk+1)−y

$=Y_k+1-m(X_k+1)-b$ =Yk+1−m(Xk+1)−b

You can use these to make a simple decision about which pixel is closer to the mathematical line. This simple decision is based on the difference between the two pixel positions.

$$d_{lower}-d_{upper}=2m(x_k+1)-2y_k+2b-1$$ dlower−dupper=2m(xk+1)−2yk+2b−1

Let us substitute *m* with *dy/dx* where *dx* and *dy* are the differences between the end-points.

$$dx(d_{lower} - d_{upper}) = dx\left(2\frac{dy}{dx}(x_k + 1) - 2y_k + 2b - 1\right)$$

$$= 2dy.x_k - 2dx.y_k + 2dy + 2dx(2b - 1)$$

$$= 2dy.x_k - 2dx.y_k + C$$

So, a decision parameter P_k for the *k*th step along a line is given by −

$$p_k = dx(d_{lower} - d_{upper})$$

$$= 2dy.x_k - 2dx.y_k + C$$

The sign of the decision parameter P_k is the same as that of $d_{lower} - d_{upper}$.

If p_k is negative, then choose the lower pixel, otherwise choose the upper pixel.

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

$$p_{k+1} = 2dy.x_{k+1} - 2dx.y_{k+1} + C$$

Subtracting p_k from this we get −

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

But, x_{k+1} is the same as $(x_k) + 1$. So −

$$p_{k+1} = p_k + 2dy - 2dx(y_{k+1} - y_k)$$

Where, $Y_{k+1} - Y_k$ is either 0 or 1 depending on the sign of P_k.

The first decision parameter p_0 is evaluated at (x_0, y_0) is given as −

$$p_0 = 2dy - dx$$

Now, keeping in mind all the above points and calculations, here is the Bresenham algorithm for slope m < 1 −

Step 1 − Input the two end-points of line, storing the left end-point in (x_0, y_0).

Step 2 − Plot the point (x_0, y_0).

Step 3 − Calculate the constants dx, dy, 2dy, and (2dy – 2dx) and get the first value for the decision parameter as −

$$p_0 = 2dy - dx$$

Step 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 $(x_k + 1, y_k)$ and

$$p_{k+1} = p_k + 2dy$$

Otherwise,

$$(x_k, y_k + 1)$$

$$p_{k+1} = p_k + 2dy - 2dx$$

Step 5 − Repeat step 4 (dx – 1) times.

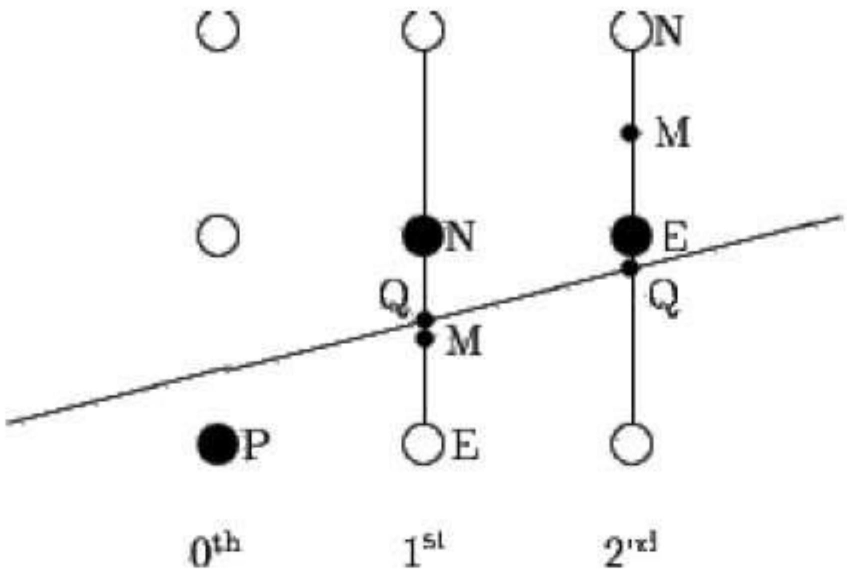
For m > 1, find out whether you need to increment x while incrementing y each time.

After solving, the equation for decision parameter P_k will be very similar, just the x and y in the equation gets interchanged.

Mid-Point Algorithm

Mid-point algorithm is due to Bresenham which was modified by Pitteway and Van Aken. Assume that you have already put the point P at (x, y) coordinate and the slope of the line is 0 ≤ k ≤ 1 as shown in the following illustration.

Now you need to decide whether to put the next point at E or N. This can be chosen by identifying the intersection point Q closest to the point N or E. If the intersection point Q is closest to the point N then N is considered as the next point; otherwise E.



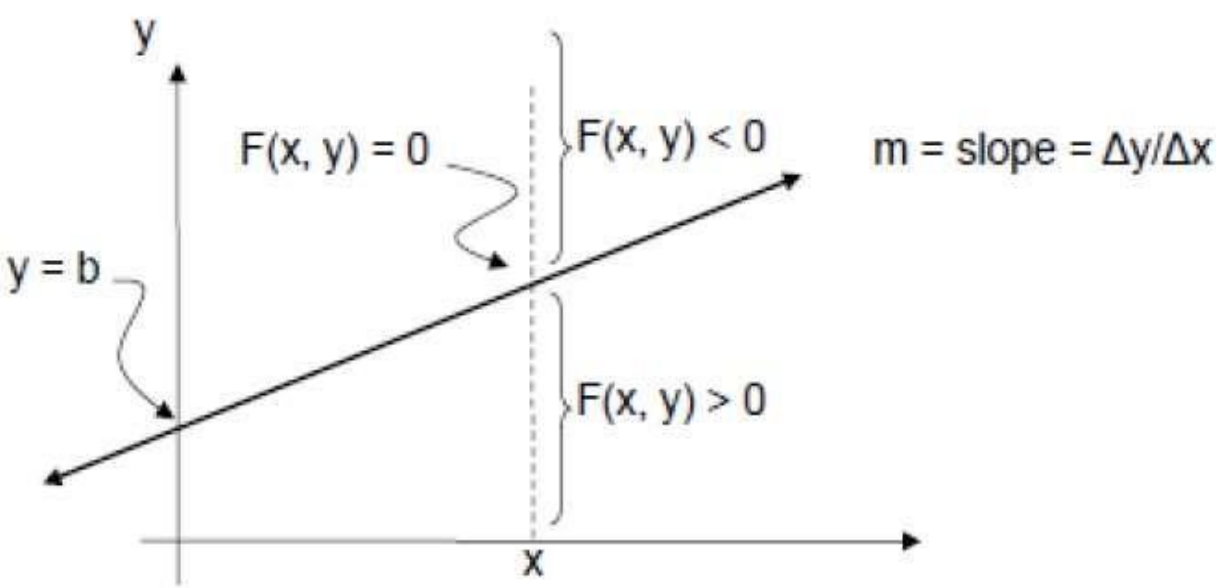
To determine that, first calculate the mid-point M(x+1, y + ½). If the intersection point Q of the line with the vertical line connecting E and N is below M, then take E as the next point; otherwise take N as the next point.

In order to check this, we need to consider the implicit equation −

$$F(x,y) = mx + b - y$$

For positive m at any given X,

- If y is on the line, then F(x, y) = 0
- If y is above the line, then F(x, y) < 0
- If y is below the line, then F(x, y) > 0



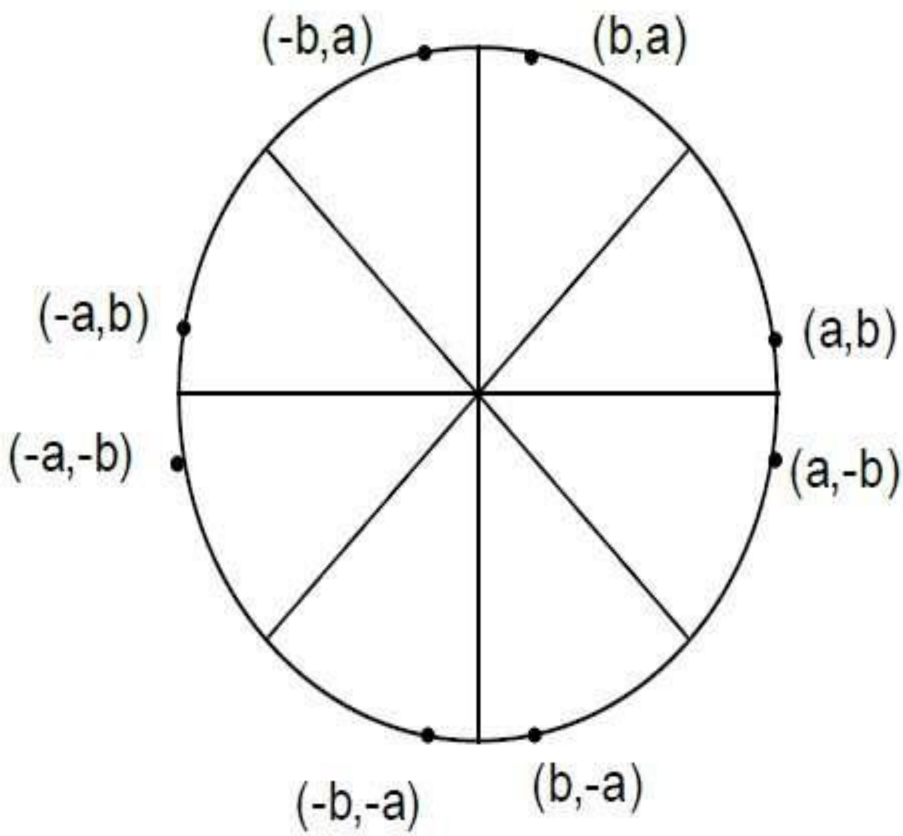
Circle Generation Algorithm

Drawing a circle on the screen is a little complex than drawing a line. There are two popular algorithms for generating a circle − **Bresenham's**

Algorithm and **Midpoint Circle Algorithm**. These algorithms are based on the idea of determining the subsequent points required to draw the circle. Let us discuss

the algorithms in detail −

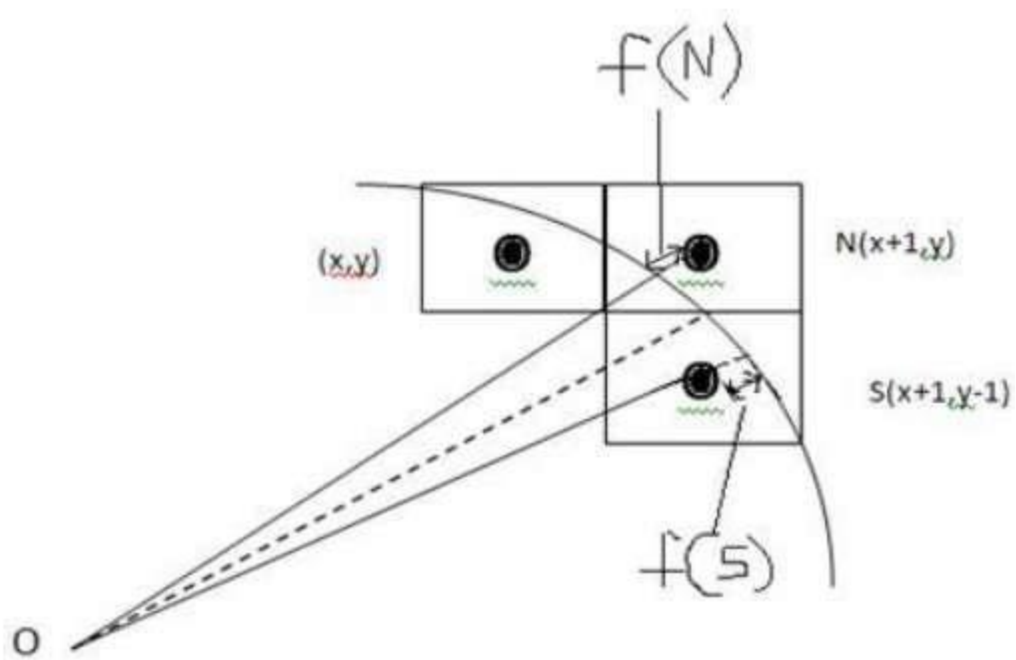
The equation of circle is $X^2+Y^2=r^2, X2+Y2=r2$, where r is radius.



Bresenham's Algorithm

We cannot display a continuous arc on the raster display. Instead, we have to choose the nearest pixel position to complete the arc.

From the following illustration, you can see that we have put the pixel at (X, Y) location and now need to decide where to put the next pixel − at N (X+1, Y) or at S (X+1, Y-1).



This can be decided by the decision parameter **d.**

- If d <= 0, then N(X+1, Y) is to be chosen as next pixel.
- If d > 0, then S(X+1, Y-1) is to be chosen as the next pixel.

Algorithm

Step 1 − Get the coordinates of the center of the circle and radius, and store them in x, y, and R respectively. Set P=0 and Q=R.

Step 2 − Set decision parameter D = 3 – 2R.

Step 3 − Repeat through step-8 while P ≤ Q.

Step 4 − Call Draw Circle (X, Y, P, Q).

Step 5 − Increment the value of P.

Step 6 − If D < 0 then D = D + 4P + 6.

Step 7 − Else Set R = R - 1, D = D + 4(P-Q) + 10.

Step 8 − Call Draw Circle (X, Y, P, Q).

Draw Circle Method(X, Y, P, Q).

Call Putpixel (X + P, Y + Q).
Call Putpixel (X - P, Y + Q).
Call Putpixel (X + P, Y - Q).
Call Putpixel (X - P, Y - Q).
Call Putpixel (X + Q, Y + P).
Call Putpixel (X - Q, Y + P).
Call Putpixel (X + Q, Y - P).
Call Putpixel (X - Q, Y - P).

Mid Point Algorithm

Step 1 − Input radius **r** and circle center (x_c, y_c)(xc,yc) and obtain the first point on the circumference of the circle centered on the origin as

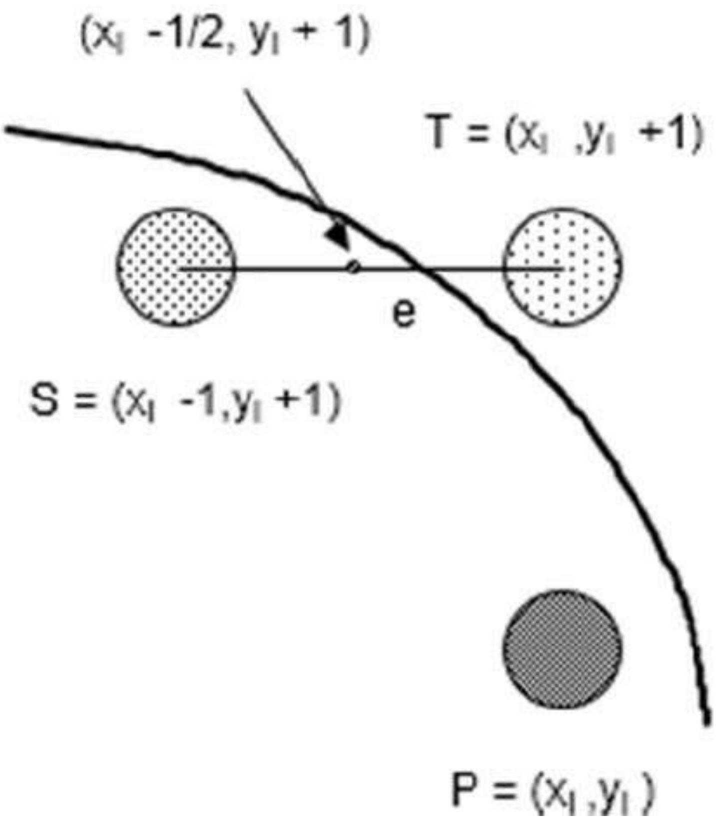
$(x_0, y_0) = (0, r)$

Step 2 − Calculate the initial value of decision parameter as

P_0P0 = 5/4 – r (See the following description for simplification of this equation.)

```
f(x, y) = x² + y² - r² = 0

f(xᵢ - 1/2 + e, yᵢ + 1)
        = (xᵢ - 1/2 + e)² + (yᵢ + 1)² - r²
        = (xᵢ- 1/2)² + (yᵢ + 1)² - r² + 2(xᵢ - 1/2)e + e²
        = f(xᵢ - 1/2, yᵢ + 1) + 2(xᵢ - 1/2)e + e² = 0
```



Let $d_i = f(x_i - 1/2,\ y_i + 1) = -2(x_i - 1/2)e - e^2$
Thus,

If e < 0 then di > 0 so choose point $S = (x_i - 1,\ y_i + 1)$.
$d_{i+1} = f(x_i - 1 - 1/2,\ y_i + 1 + 1) = ((x_i - 1/2) - 1)^2 + ((y_i + 1) + 1)^2 - r^2$
$= d_i - 2(x_i - 1) + 2(y_i + 1) + 1$
$= d_i + 2(y_{i+1} - x_{i+1}) + 1$

If e >= 0 then di <= 0 so choose point $T = (x_i,\ y_i + 1)$
$d_{i+1} = f(x_i - 1/2,\ y_i + 1 + 1)$
$= d_i + 2y_{i+1} + 1$

The initial value of di is
$d_0 = f(r - 1/2,\ 0 + 1) = (r - 1/2)^2 + 1^2 - r^2$
$= 5/4 - r$ {1-r can be used if r is an integer}

When point $S = (x_i - 1,\ y_i + 1)$ is chosen then
$d_{i+1} = d_i + -2x_{i+1} + 2y_{i+1} + 1$

When point $T = (x_i,\ y_i + 1)$ is chosen then
$d_{i+1} = d_i + 2y_{i+1} + 1$

Step 3 − At each XKXK position starting at K=0, perform the following test −

```
If PK < 0 then next point on circle (0,0) is (XK+1,YK) and
   PK+1 = PK + 2XK+1 + 1
Else
   PK+1 = PK + 2XK+1 + 1 – 2YK+1

Where, 2XK+1 = 2XK+2 and 2YK+1 = 2YK-2.
```

Step 4 − Determine the symmetry points in other seven octants.

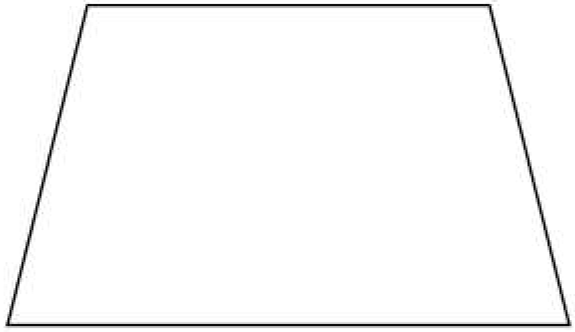
Step 5 − Move each calculate pixel position (X, Y) onto the circular path centered on (X_C, Y_C)(XC,YC) and plot the coordinate values.

$X = X + X_C, \quad Y = Y + Y_C$

Step 6 − Repeat step-3 through 5 until X >= Y.

Polygon Filling Algorithm

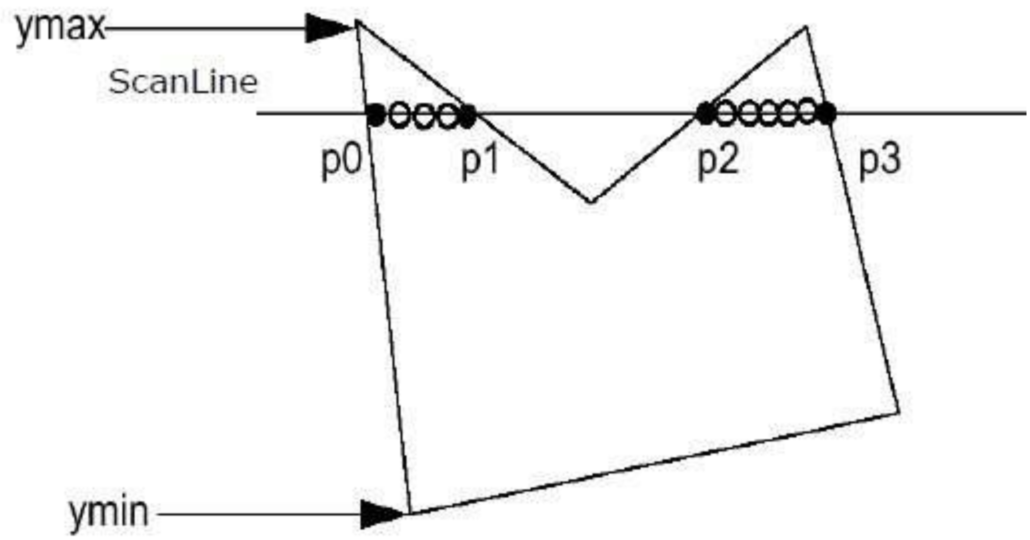
Polygon is an ordered list of vertices as shown in the following figure. For filling polygons with particular colors, you need to determine the pixels falling on the border of the polygon and those which fall inside the polygon. In this chapter, we will see how we can fill polygons using different techniques.



Scan Line Algorithm

This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections. The following steps depict how this algorithm works.

Step 1 − Find out the Ymin and Ymax from the given polygon.



Step 2 − ScanLine intersects with each edge of the polygon from Ymin to Ymax.

Name each intersection point of the polygon. As per the figure shown above, they are named as p0, p1, p2, p3.

Step 3 − Sort the intersection point in the increasing order of X coordinate i.e. (p0, p1), (p1, p2), and (p2, p3).

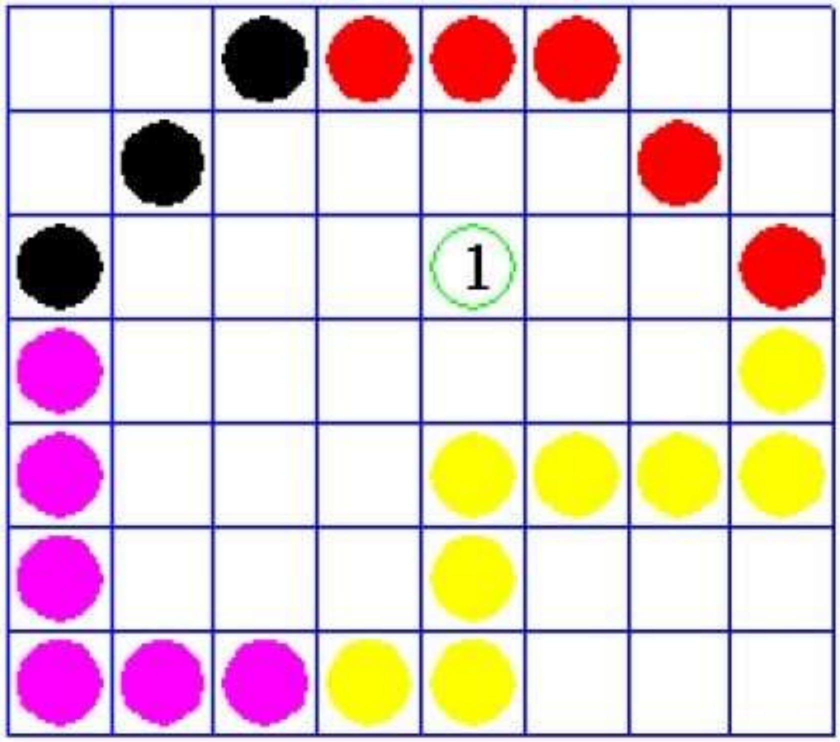
Step 4 − Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

Flood Fill Algorithm

Sometimes we come across an object where we want to fill the area and its boundary with different colors. We can paint such objects with a specified interior color instead of searching for particular boundary color as in boundary filling algorithm.

Instead of relying on the boundary of the object, it relies on the fill color. In other words, it replaces the interior color of the object with the fill color. When no more pixels of the original interior color exist, the algorithm is completed.

Once again, this algorithm relies on the Four-connect or Eight-connect method of filling in the pixels. But instead of looking for the boundary color, it is looking for all adjacent pixels that are a part of the interior.



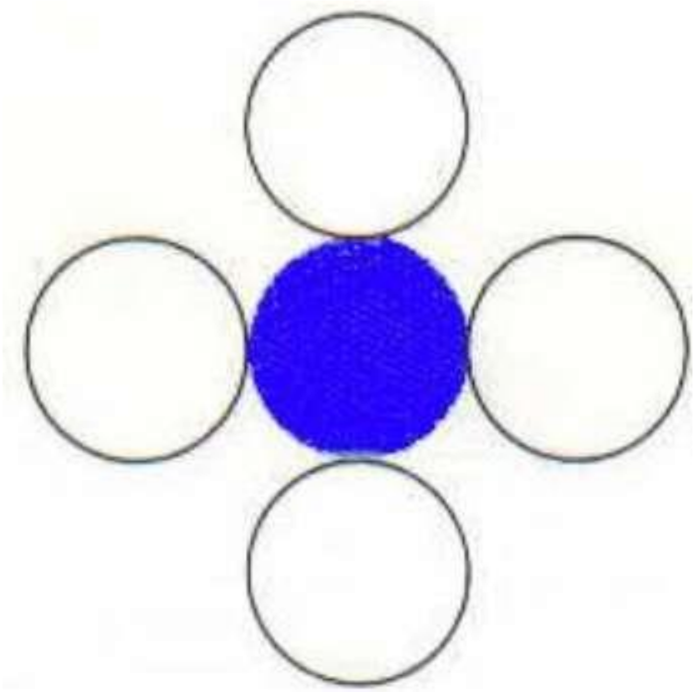
Boundary Fill Algorithm

The boundary fill algorithm works as its name. This algorithm picks a point inside an object and starts to fill until it hits the boundary of the object. The color of the boundary and the color that we fill should be different for this algorithm to work.

In this algorithm, we assume that color of the boundary is same for the entire object. The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

4-Connected Polygon

In this technique 4-connected pixels are used as shown in the figure. We are putting the pixels above, below, to the right, and to the left side of the current pixels and this process will continue until we find a boundary with different color.



Algorithm

Step 1 − Initialize the value of seed point (seedx, seedy), fcolor and dcol.

Step 2 − Define the boundary values of the polygon.

Step 3 − Check if the current seed point is of default color, then repeat the steps 4 and 5 till the boundary pixels reached.

```
If getpixel(x, y) = dcol then repeat step 4 and 5
```

Step 4 − Change the default color with the fill color at the seed point.

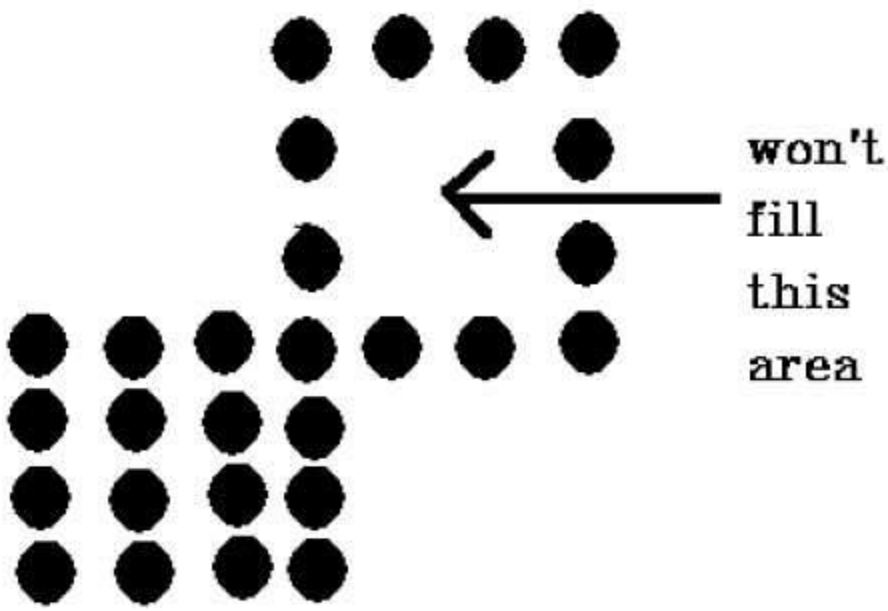
setPixel(seedx, seedy, fcol)

Step 5 − Recursively follow the procedure with four neighborhood points.

FloodFill (seedx – 1, seedy, fcol, dcol)
FloodFill (seedx + 1, seedy, fcol, dcol)
FloodFill (seedx, seedy - 1, fcol, dcol)
FloodFill (seedx – 1, seedy + 1, fcol, dcol)

Step 6 − Exit

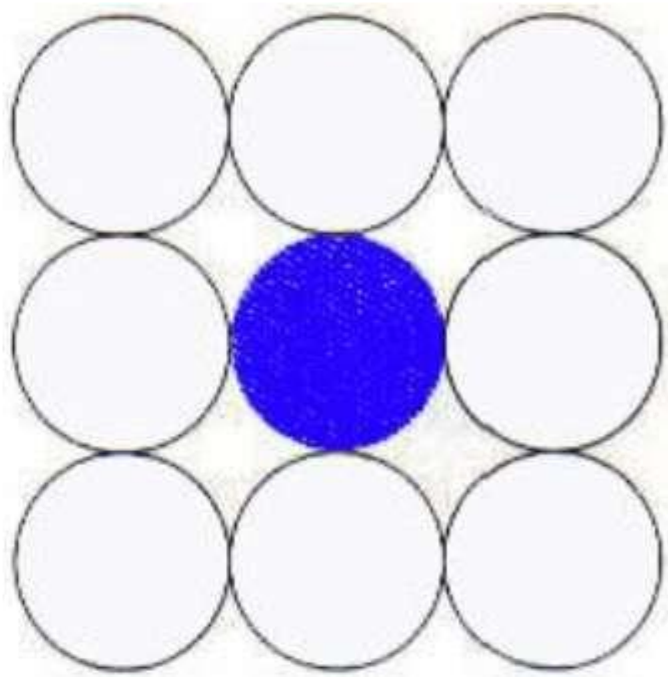
There is a problem with this technique. Consider the case as shown below where we tried to fill the entire region. Here, the image is filled only partially. In such cases, 4-connected pixels technique cannot be used.



8-Connected Polygon

In this technique 8-connected pixels are used as shown in the figure. We are putting pixels above, below, right and left side of the current pixels as we were doing in 4-connected technique.

In addition to this, we are also putting pixels in diagonals so that entire area of the current pixel is covered. This process will continue until we find a boundary with different color.



Algorithm

Step 1 − Initialize the value of seed point (seedx, seedy), fcolor and dcol.

Step 2 − Define the boundary values of the polygon.

Step 3 − Check if the current seed point is of default color then repeat the steps 4 and 5 till the boundary pixels reached

If getpixel(x,y) = dcol then repeat step 4 and 5

Step 4 − Change the default color with the fill color at the seed point.

setPixel(seedx, seedy, fcol)

Step 5 − Recursively follow the procedure with four neighbourhood points

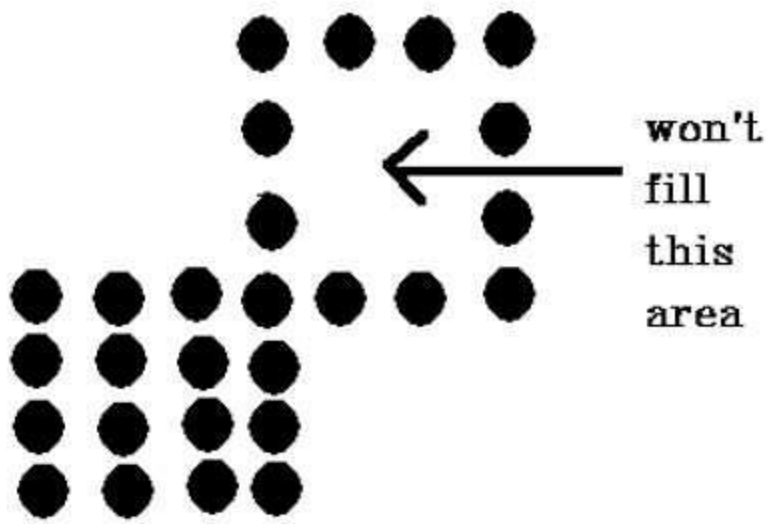
FloodFill (seedx – 1, seedy, fcol, dcol)
FloodFill (seedx + 1, seedy, fcol, dcol)
FloodFill (seedx, seedy - 1, fcol, dcol)
FloodFill (seedx, seedy + 1, fcol, dcol)
FloodFill (seedx – 1, seedy + 1, fcol, dcol)
FloodFill (seedx + 1, seedy + 1, fcol, dcol)

FloodFill (seedx + 1, seedy - 1, fcol, dcol)

FloodFill (seedx – 1, seedy - 1, fcol, dcol)

Step 6 − Exit

The 4-connected pixel technique failed to fill the area as marked in the following figure which won't happen with the 8-connected technique.



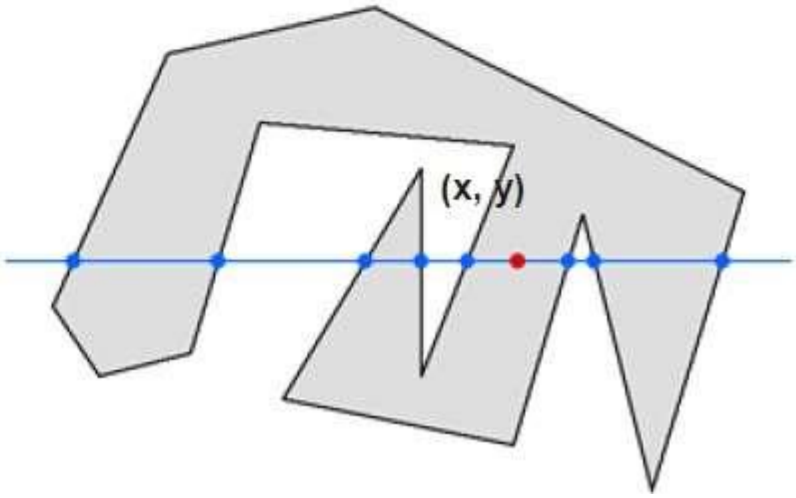
Inside-outside Test

This method is also known as **counting number method.** While filling an object, we often need to identify whether particular point is inside the object or outside it. There are two methods by which we can identify whether particular point is inside an object or outside.

- Odd-Even Rule
- Nonzero winding number rule

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.

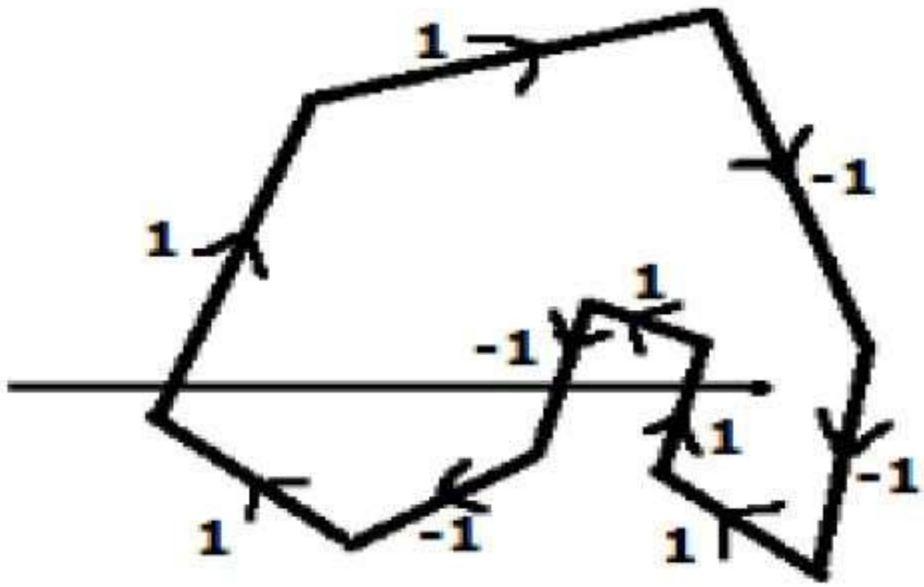


From the above figure, we can see that from the point (x,y), the number of interactions point on the left side is 5 and on the right side is 3. From both ends, the number of interaction points is odd, so the point is considered within the object.

Nonzero Winding Number Rule

This method is also used with the simple polygons to test the given point is interior or not. It can be simply understood with the help of a pin and a rubber band. Fix up the pin on one of the edge of the polygon and tie-up the rubber band in it and then stretch the rubber band along the edges of the polygon.

When all the edges of the polygon are covered by the rubber band, check out the pin which has been fixed up at the point to be test. If we find at least one wind at the point consider it within the polygon, else we can say that the point is not inside the polygon.



In another alternative method, give directions to all the edges of the polygon. Draw a scan line from the point to be test towards the left most of X direction.

- Give the value 1 to all the edges which are going to upward direction and all other -1 as direction values.
- Check the edge direction values from which the scan line is passing and sum up them.
- If the total sum of this direction value is non-zero, then this point to be tested is an **interior point,** otherwise it is an **exterior point.**
- In the above figure, we sum up the direction values from which the scan line is passing then the total is 1 – 1 + 1 = 1; which is non-zero. So the point is said to be an interior point.

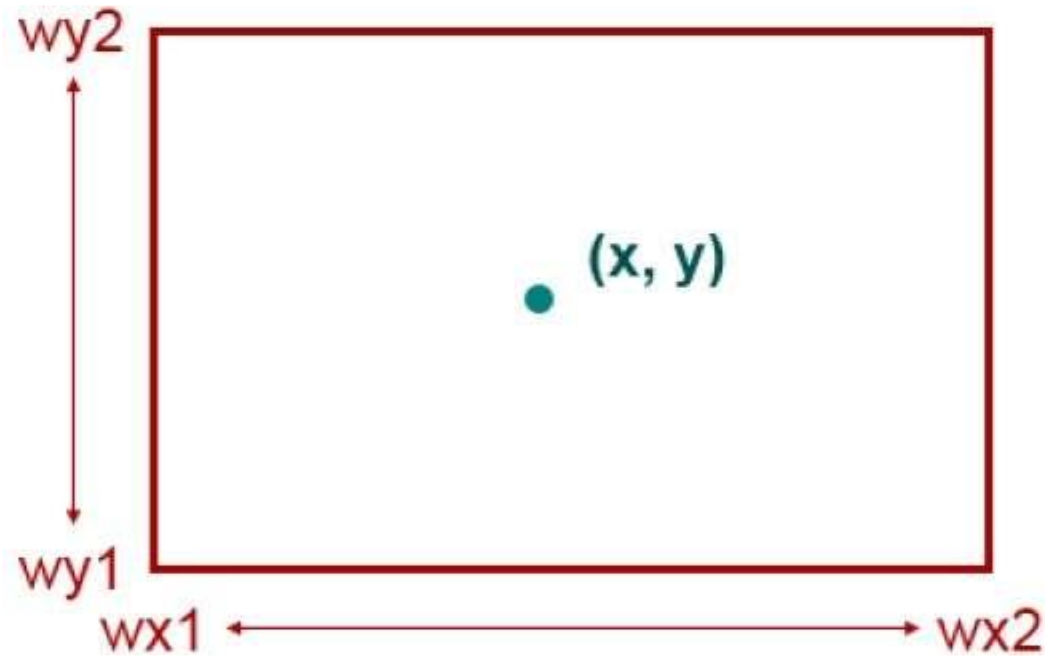
Viewing & Clipping

The primary use of clipping in computer graphics is to remove objects, lines, or line segments that are outside the viewing pane. The viewing transformation is insensitive to the position of points relative to the viewing volume − especially those points behind the viewer − and it is necessary to remove these points before generating the view.

Point Clipping

Clipping a point from a given window is very easy. Consider the following figure, where the rectangle indicates the window. Point clipping tells us whether the given point (X, Y) is within the given window or not; and decides whether we will use the minimum and maximum coordinates of the window.

The X-coordinate of the given point is inside the window, if X lies in between Wx1 ≤ X ≤ Wx2. Same way, Y coordinate of the given point is inside the window, if Y lies in between Wy1 ≤ Y ≤ Wy2.

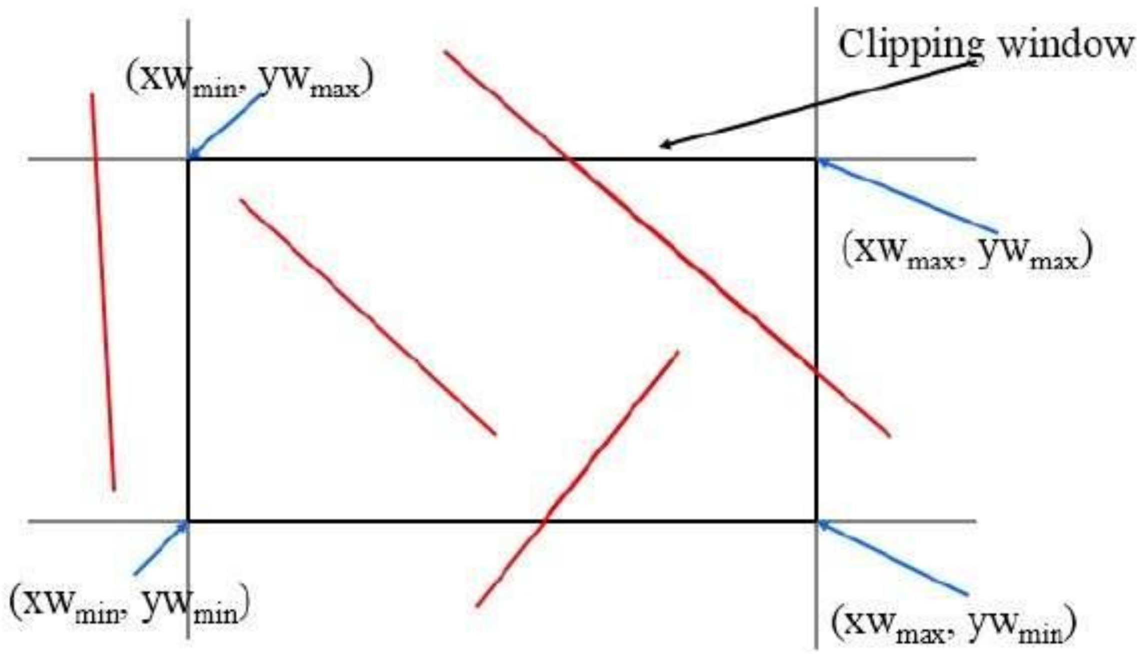


Line Clipping

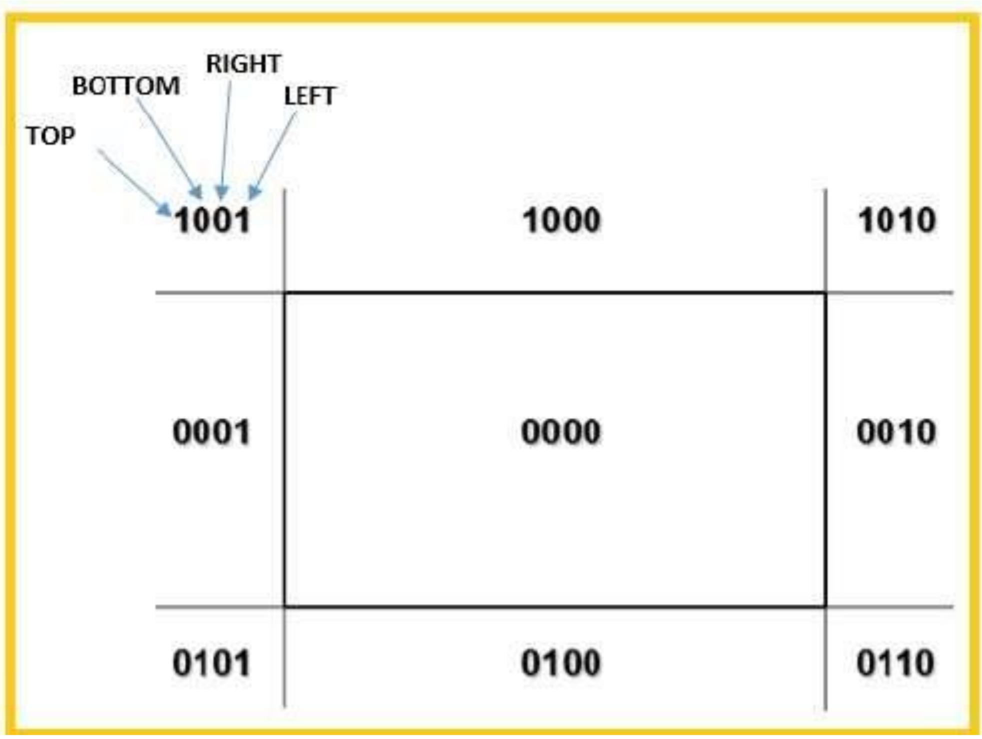
The concept of line clipping is same as point clipping. In line clipping, we will cut the portion of line which is outside of window and keep only the portion that is inside the window.

Cohen-Sutherland Line Clippings

This algorithm uses the clipping window as shown in the following figure. The minimum coordinate for the clipping region is (XW_{min}, YW_{min})(XWmin,YWmin) and the maximum coordinate for the clipping region is (XW_{max}, YW_{max})(XWmax,YWmax).



We will use 4-bits to divide the entire region. These 4 bits represent the Top, Bottom, Right, and Left of the region as shown in the following figure. Here, the **TOP** and **LEFT** bit is set to 1 because it is the **TOP-LEFT** corner.



There are 3 possibilities for the line −

- Line can be completely inside the window (This line should be accepted).
- Line can be completely outside of the window (This line will be completely removed from the region).
- Line can be partially inside the window (We will find intersection point and draw only that portion of line that is inside region).

Algorithm

Step 1 − Assign a region code for each endpoints.

Step 2 − If both endpoints have a region code **0000** then accept this line.

Step 3 − Else, perform the logical **AND** operation for both region codes.

Step 3.1 − If the result is not **0000,** then reject the line.

Step 3.2 − Else you need clipping.

Step 3.2.1 − Choose an endpoint of the line that is outside the window.

Step 3.2.2 − Find the intersection point at the window boundary (base on region code).

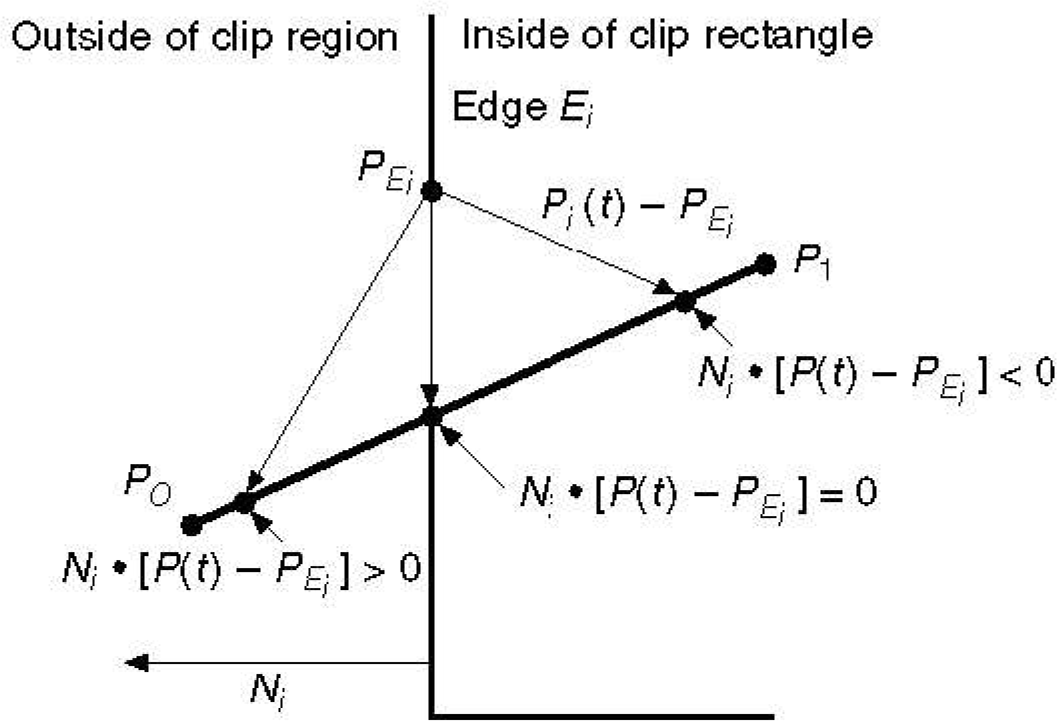
Step 3.2.3 − Replace endpoint with the intersection point and update the region code.

Step 3.2.4 − Repeat step 2 until we find a clipped line either trivially accepted or trivially rejected.

Step 4 − Repeat step 1 for other lines.

Cyrus-Beck Line Clipping Algorithm

This algorithm is more efficient than Cohen-Sutherland algorithm. It employs parametric line representation and simple dot products.



Parametric equation of line is −

$P_0P_1:P(t) = P_0 + t(P_1 - P_0)$

Let N_i be the outward normal edge E_i. Now pick any arbitrary point P_{Ei} on edge

E_i then the dot product $N_i.[P(t) – P_{Ei}]$ determines whether the point P(t) is "inside the clip edge" or "outside" the clip edge or "on" the clip edge.

The point P(t) is inside if $N_i.[P(t) – P_{Ei}] < 0$

The point P(t) is outside if $N_i.[P(t) – P_{Ei}] > 0$

The point P(t) is on the edge if $N_i.[P(t) – P_{Ei}] = 0$ (Intersection point)

$N_i.[P(t) – P_{Ei}] = 0$

$N_i.[P_0 + t(P_1 - P_0) – P_{Ei}] = 0$ (Replacing P(t) with $P_0 + t(P_1 - P_0)$)

$N_i.[P_0 – P_{Ei}] + N_i.t[P_1 - P_0] = 0$

$N_i.[P_0 – P_{Ei}] + N_i \cdot tD = 0$ (substituting D for $[P_1 - P_0]$)

$N_i.[P_0 – P_{Ei}] = - N_i \cdot tD$

The equation for t becomes,

$$t=N_i.[P_o-P_{Ei}]-N_i.Dt=Ni.[Po-PEi]-Ni.D$$

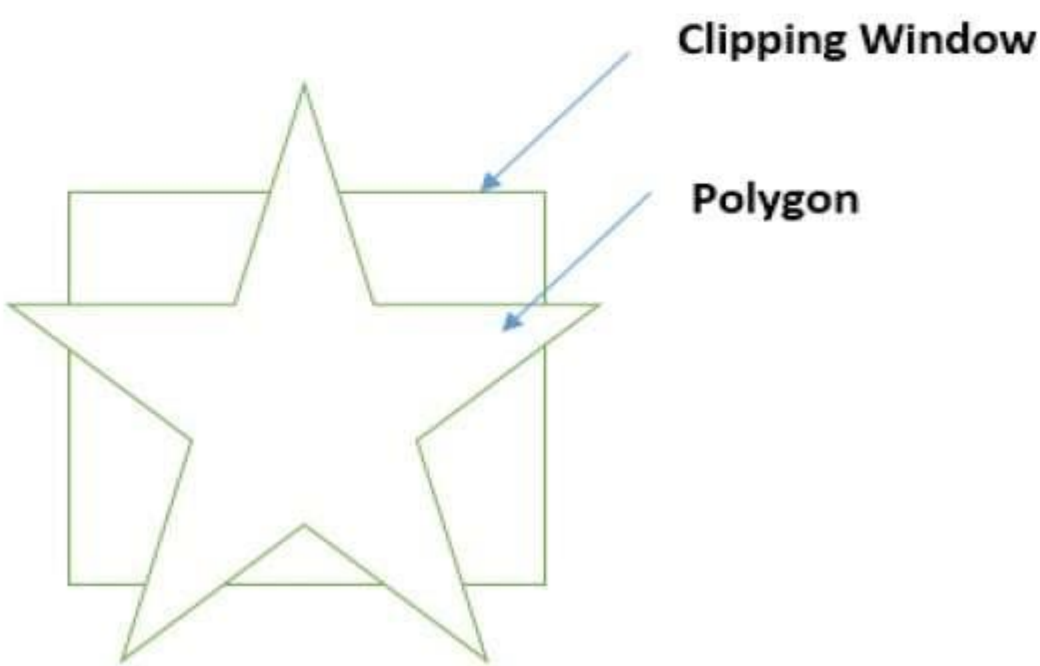
It is valid for the following conditions −

- $N_i \neq 0$ (error cannot happen)
- $D \neq 0$ ($P_1 \neq P_0$)
- $N_i \cdot D \neq 0$ (P_0P_1 not parallel to E_i)

Polygon Clipping (Sutherland Hodgman Algorithm)

A polygon can also be clipped by specifying the clipping window. Sutherland Hodgeman polygon clipping algorithm is used for polygon clipping. In this algorithm, all the vertices of the polygon are clipped against each edge of the clipping window.

First the polygon is clipped against the left edge of the polygon window to get new vertices of the polygon. These new vertices are used to clip the polygon against right edge, top edge, bottom edge, of the clipping window as shown in the following figure.



While processing an edge of a polygon with clipping window, an intersection point is found if edge is not completely inside clipping window and the a partial edge from the intersection point to the outside edge is clipped. The following figures show left, right, top and bottom edge clippings −

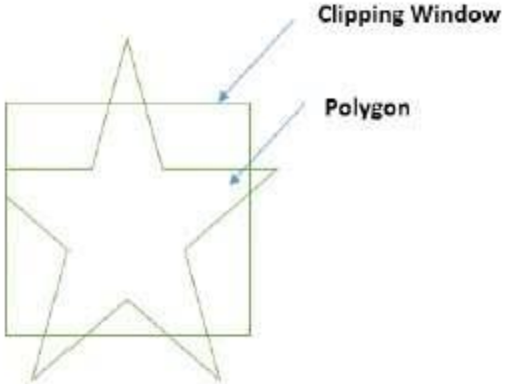


Figure: Clipping Left Edge

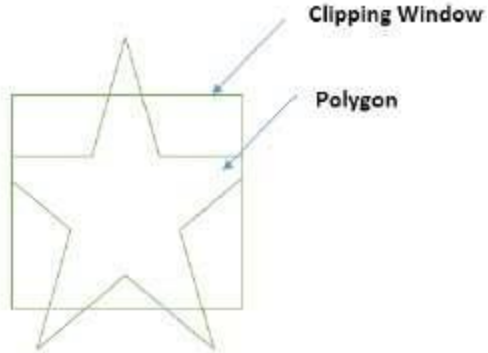


Figure: Clipping Right Edge

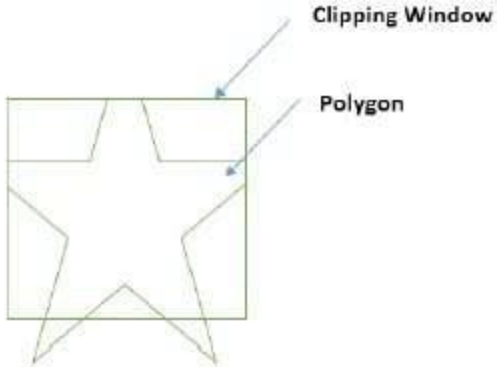


Figure: Clipping Top Edge

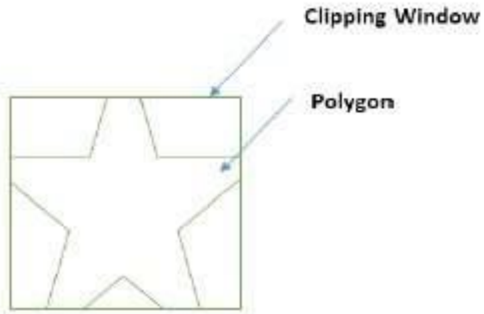


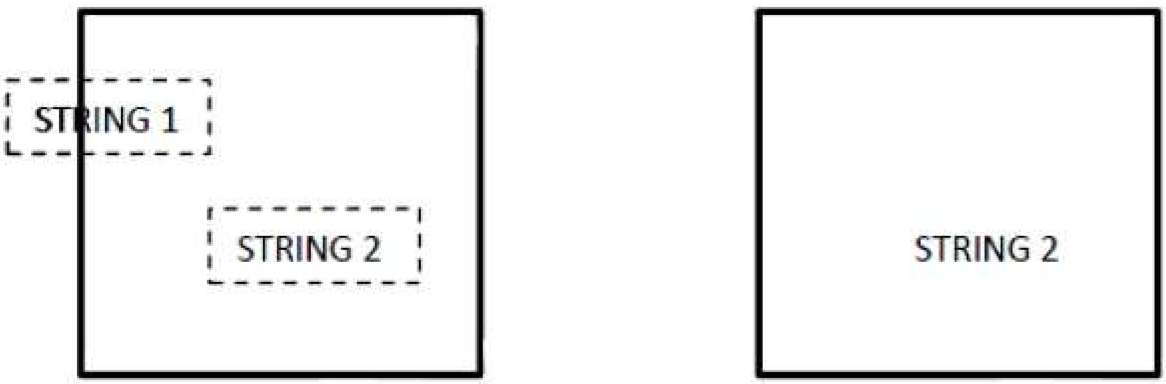
Figure: Clipping Bottom Edge

Text Clipping

Various techniques are used to provide text clipping in a computer graphics. It depends on the methods used to generate characters and the requirements of a particular application. There are three methods for text clipping which are listed below −

- All or none string clipping
- All or none character clipping
- Text clipping

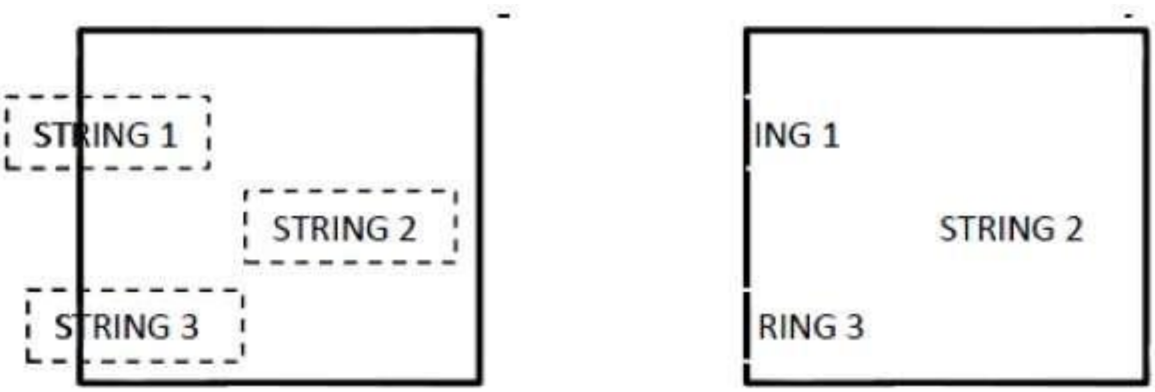
The following figure shows all or none string clipping −



Before Clipping After Clipping

In all or none string clipping method, either we keep the entire string or we reject entire string based on the clipping window. As shown in the above figure, STRING2 is entirely inside the clipping window so we keep it and STRING1 being only partially inside the window, we reject.

The following figure shows all or none character clipping −

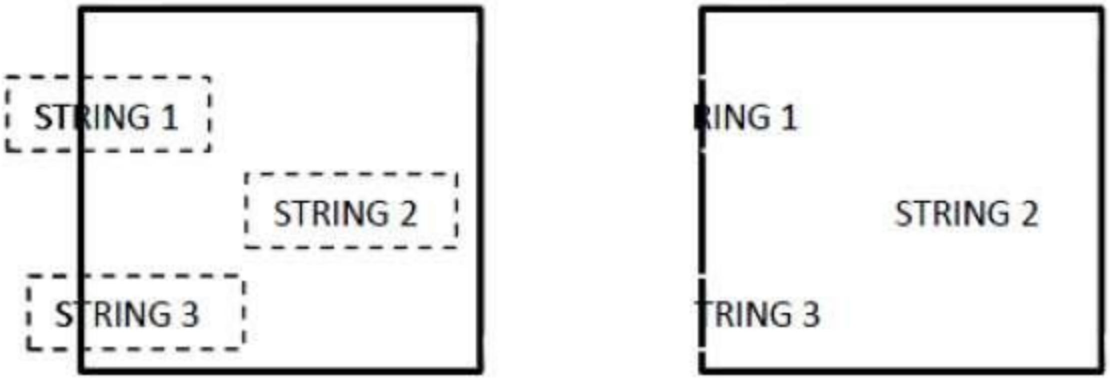


Before Clipping After Clipping

This clipping method is based on characters rather than entire string. In this method if the string is entirely inside the clipping window, then we keep it. If it is partially outside the window, then −

- You reject only the portion of the string being outside
- If the character is on the boundary of the clipping window, then we discard that entire character and keep the rest string.

The following figure shows text clipping −



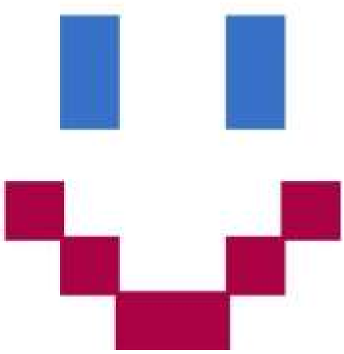
Before Clipping After Clipping

This clipping method is based on characters rather than the entire string. In this method if the string is entirely inside the clipping window, then we keep it. If it is partially outside the window, then

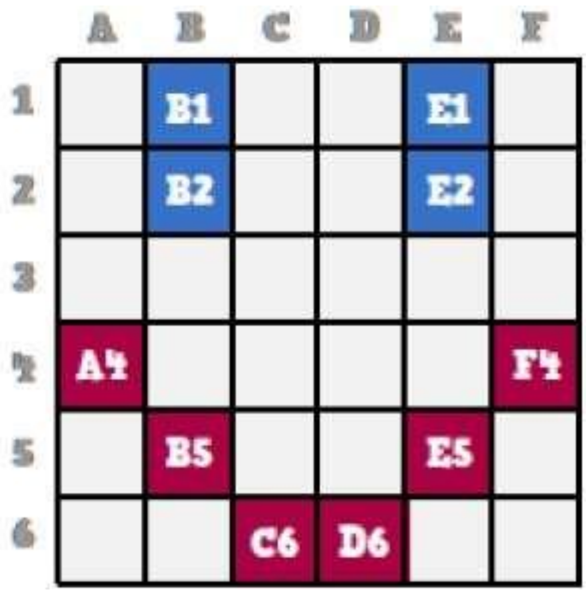
- You reject only the portion of string being outside.
- If the character is on the boundary of the clipping window, then we discard only that portion of character that is outside of the clipping window.

Bitmap Graphics

A bitmap is a collection of pixels that describes an image. It is a type of computer graphics that the computer uses to store and display pictures. In this type of graphics, images are stored bit by bit and hence it is named Bit-map graphics. For better understanding let us consider the following example where we draw a smiley face using bit-map graphics.



Now we will see how this smiley face is stored bit by bit in computer graphics.



By observing the original smiley face closely, we can see that there are two blue lines which are represented as B1, B2 and E1, E2 in the above figure.

In the same way, the smiley is represented using the combination bits of A4, B5, C6, D6, E5, and F4 respectively.

The main disadvantages of bitmap graphics are −

- We cannot resize the bitmap image. If you try to resize, the pixels get blurred.
- Colored bitmaps can be very large.

2D Transformation

Transformation means changing some graphics into something else by applying rules. We can have various types of transformations such as translation, scaling up or down, rotation, shearing, etc. When a transformation takes place on a 2D plane, it is called 2D transformation.

Transformations play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation.

Homogenous Coordinates

To perform a sequence of transformation such as translation followed by rotation and scaling, we need to follow a sequential process −

- Translate the coordinates,
- Rotate the translated coordinates, and then

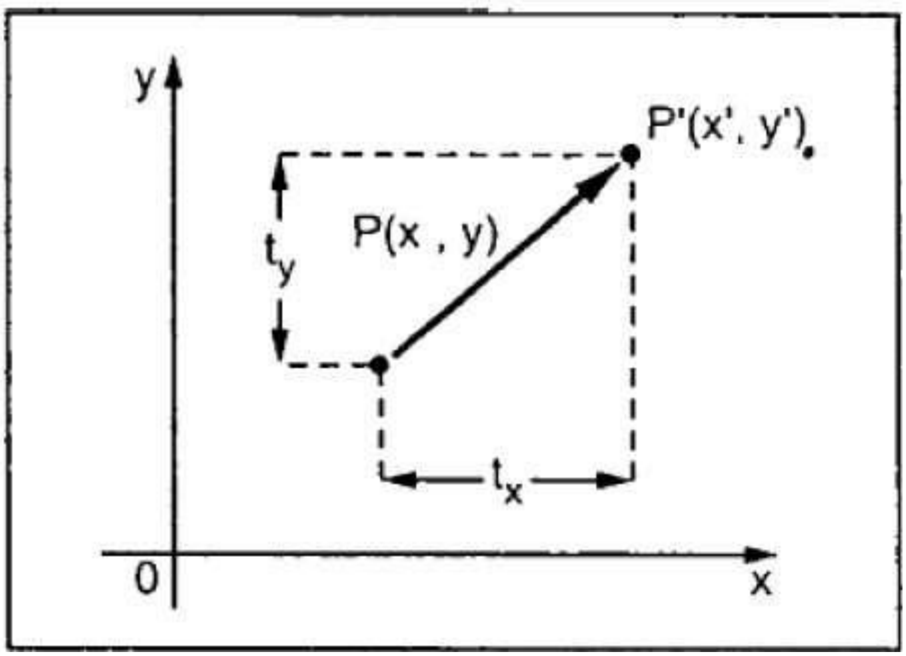
- Scale the rotated coordinates to complete the composite transformation.

To shorten this process, we have to use 3×3 transformation matrix instead of 2×2 transformation matrix. To convert a 2×2 matrix to 3×3 matrix, we have to add an extra dummy coordinate W.

In this way, we can represent the point by 3 numbers instead of 2 numbers, which is called **Homogenous Coordinate** system. In this system, we can represent all the transformation equations in matrix multiplication. Any Cartesian point P(X, Y) can be converted to homogenous coordinates by P' (X_h, Y_h, h).

Translation

A translation moves an object to a different position on the screen. You can translate a point in 2D by adding translation coordinate (t_x, t_y) to the original coordinate (X, Y) to get the new coordinate (X', Y').



From the above figure, you can write that −

$$\mathbf{X' = X + t_x}$$

$$\mathbf{Y' = Y + t_y}$$

The pair (t_x, t_y) is called the translation vector or shift vector. The above equations can also be represented using the column vectors.

$$P=[X][Y]P=[X][Y] \quad p' = [X'][Y'][X'][Y'] \quad T = [t_x][t_y][tx][ty]$$

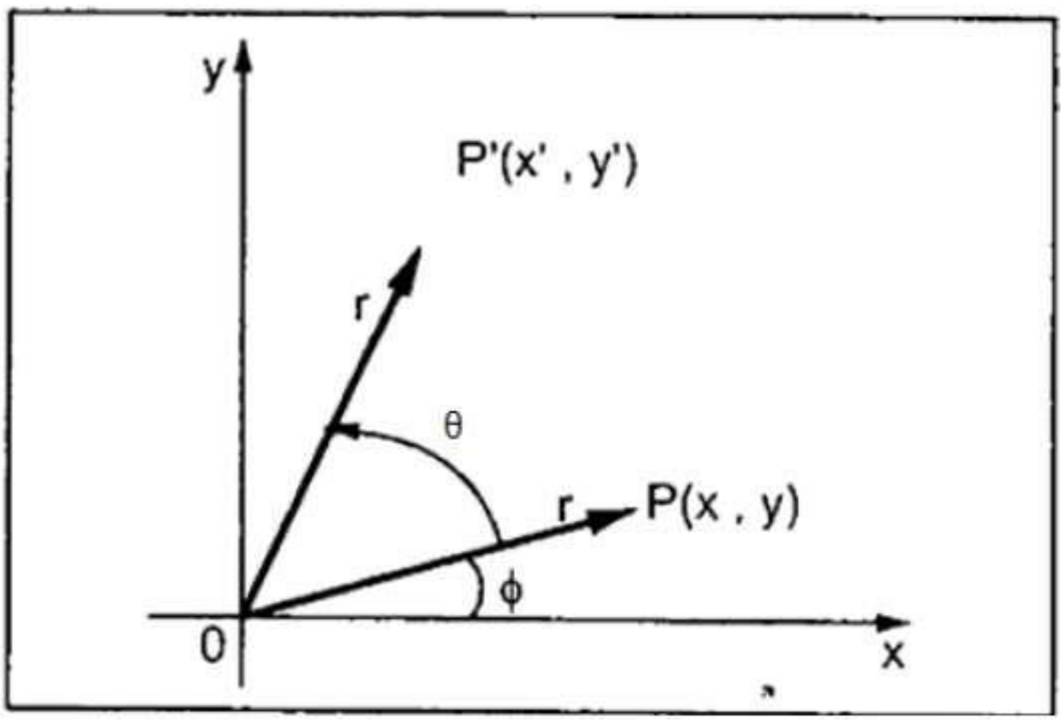
We can write it as −

$$\mathbf{P' = P + T}$$

Rotation

In rotation, we rotate the object at particular angle θ (theta) from its origin. From the following figure, we can see that the point P(X, Y) is located at angle φ from the horizontal X coordinate with distance r from the origin.

Let us suppose you want to rotate it at the angle θ. After rotating it to a new location, you will get a new point P’ (X’, Y’).



Using standard trigonometric the original coordinate of point P(X, Y) can be represented as −

$$X=r\cos\phi......(1)X=r\cos\phi......(1)$$

$$Y=r\sin\phi......(2)Y=r\sin\phi......(2)$$

Same way we can represent the point P’ (X’, Y’) as −

$$x' = r\cos(\phi+\theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta \quad ....(3)$$

$$y' = r\sin(\phi+\theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta \quad ....(4)$$

Substituting equation (1) & (2) in (3) & (4) respectively, we will get

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

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

Representing the above equation in matrix form,

$$\begin{bmatrix} X' & Y' \end{bmatrix} = \begin{bmatrix} X & Y \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} OR$$

P′ = P . R

Where R is the rotation matrix

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

The rotation angle can be positive and negative.

For positive rotation angle, we can use the above rotation matrix. However, for negative angle rotation, the matrix will change as shown below −

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

$$= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \quad (\because \cos(-\theta) = \cos\theta \text{ and } \sin(-\theta) = -\sin\theta)$$

Scaling

To change the size of an object, scaling transformation is used. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result.

Let us assume that the original coordinates are (X, Y), the scaling factors are (S_X, S_Y), and the produced coordinates are (X′, Y′). This can be mathematically represented as shown below −

$$\mathbf{X' = X \cdot S_X \text{ and } Y' = Y \cdot S_Y}$$

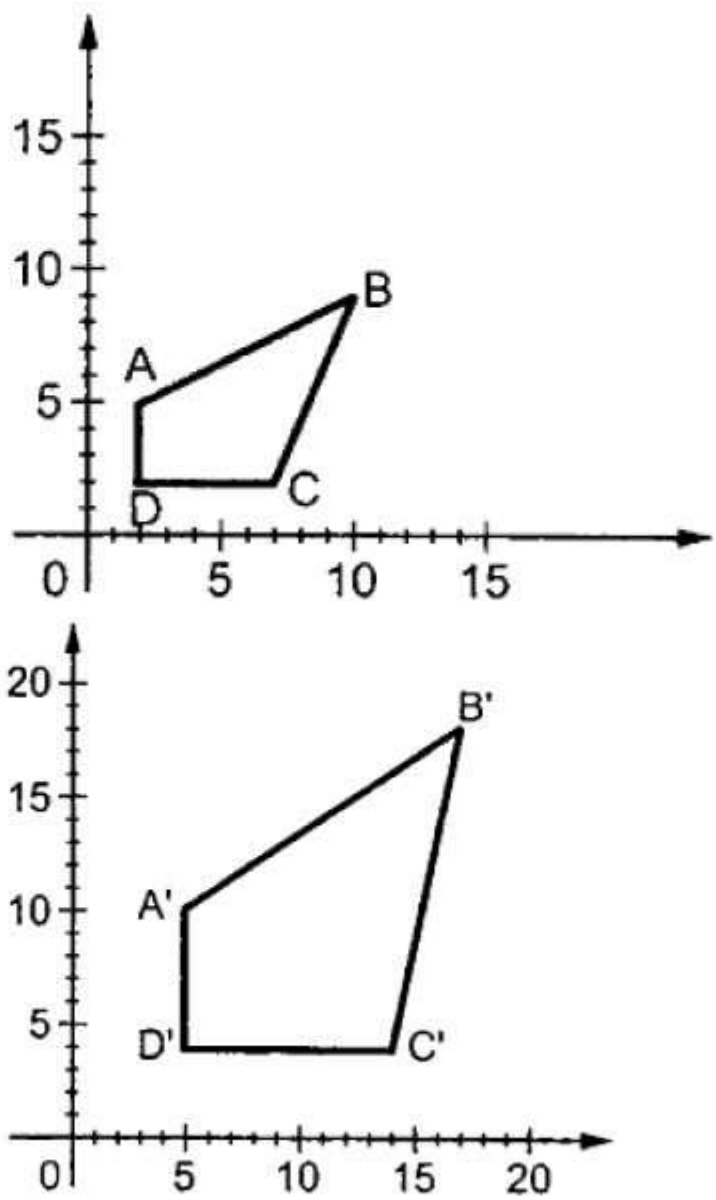
The scaling factor S_X, S_Y scales the object in X and Y direction respectively. The above equations can also be represented in matrix form as below −

$$\begin{pmatrix} X' \\ Y' \end{pmatrix} = \begin{pmatrix} X \\ Y \end{pmatrix} \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

OR

$$\mathbf{P' = P \cdot S}$$

Where S is the scaling matrix. The scaling process is shown in the following figure.

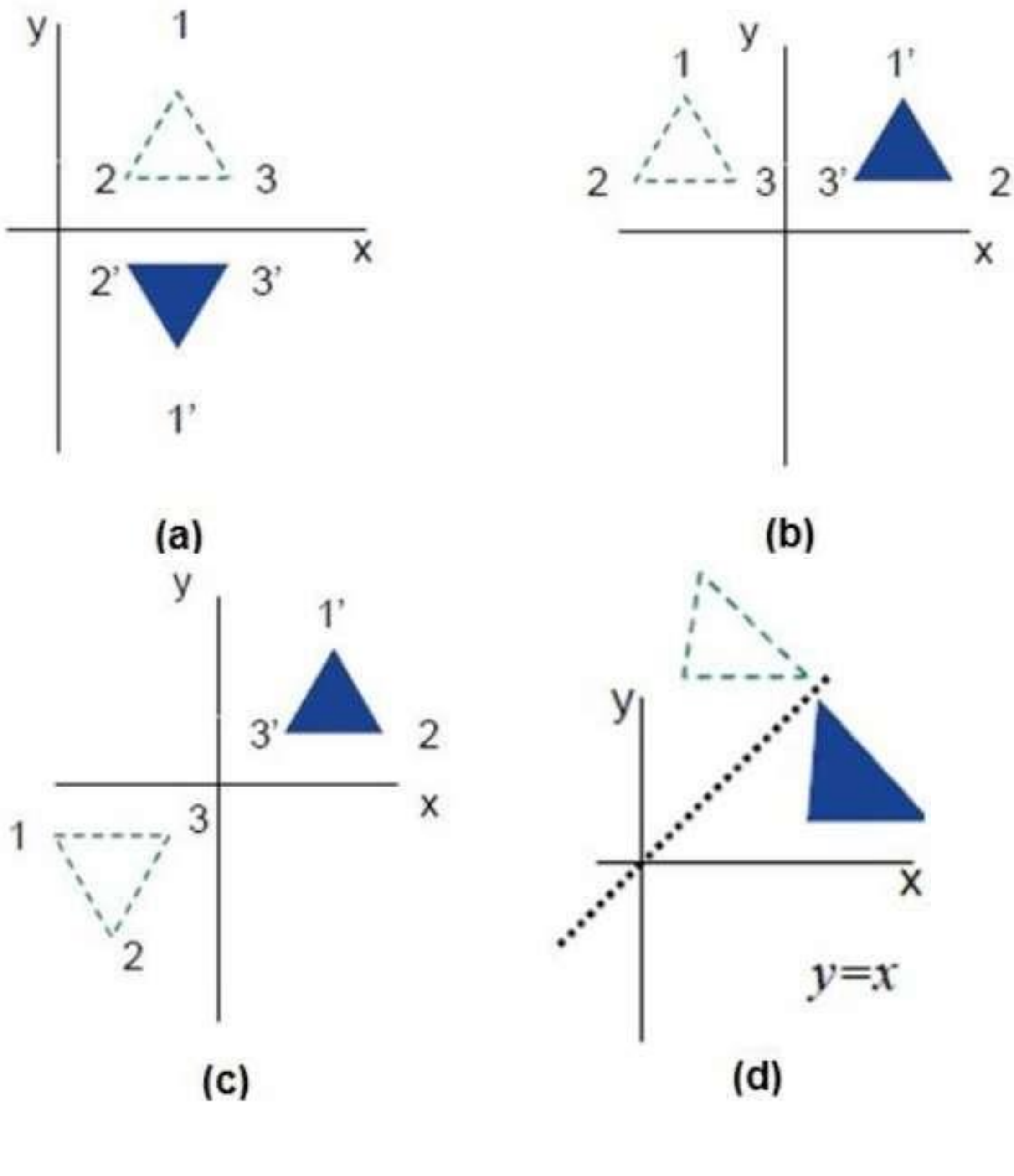


If we provide values less than 1 to the scaling factor S, then we can reduce the size of the object. If we provide values greater than 1, then we can increase the size of the object.

Reflection

Reflection is the mirror image of original object. In other words, we can say that it is a rotation operation with 180°. In reflection transformation, the size of the object does not change.

The following figures show reflections with respect to X and Y axes, and about the origin respectively.

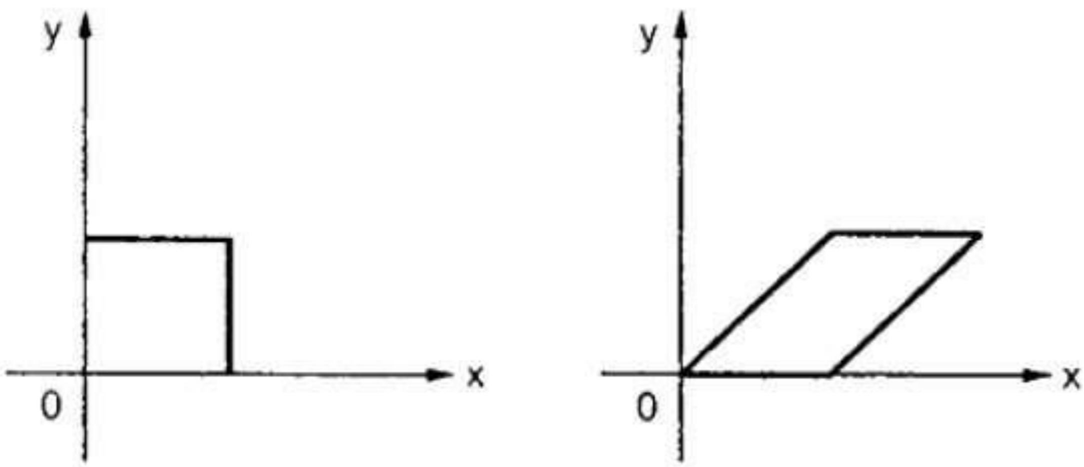


Shear

A transformation that slants the shape of an object is called the shear transformation. There are two shear transformations **X-Shear** and **Y-Shear**. One shifts X coordinates values and other shifts Y coordinate values. However; in both the cases only one coordinate changes its coordinates and other preserves its values. Shearing is also termed as **Skewing**.

X-Shear

The X-Shear preserves the Y coordinate and changes are made to X coordinates, which causes the vertical lines to tilt right or left as shown in below figure.



(a) Original object (b) Object after x shear

The transformation matrix for X-Shear can be represented as −

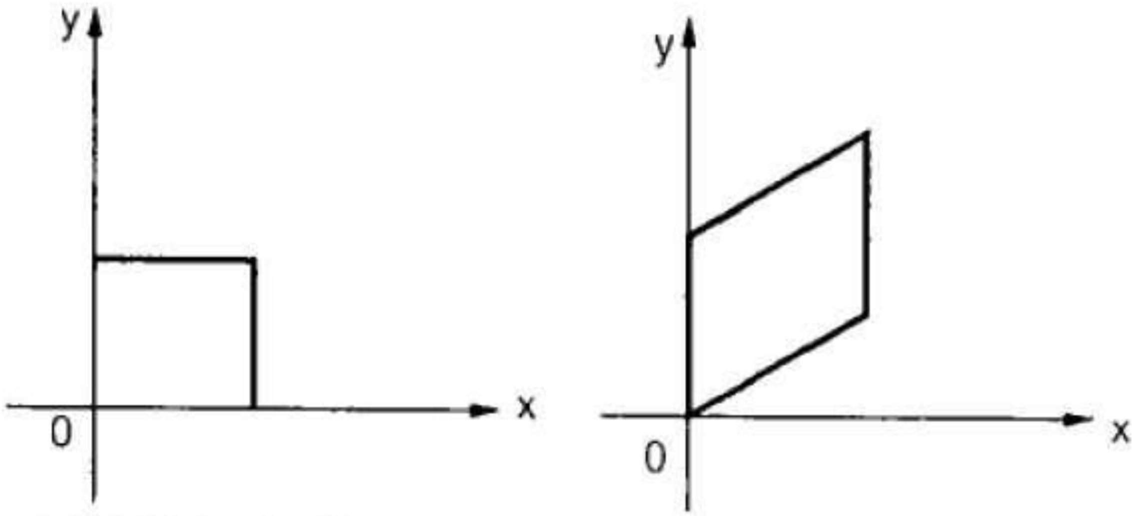
$$X_{sh} = \begin{bmatrix} 1 & 0 & 0 \\ sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad X_{sh} = [1 \; sh_x \; 0 \; 0 \; 1 \; 0 \; 0 \; 0 \; 1]$$

$$Y' = Y + Sh_y \,.\, X$$

$$X' = X$$

Y-Shear

The Y-Shear preserves the X coordinates and changes the Y coordinates which causes the horizontal lines to transform into lines which slopes up or down as shown in the following figure.



(a) Original object (b) Object after y shear

The Y-Shear can be represented in matrix from as −

Y_{sh}⎡⎣⎢⎢1shy0010001⎤⎦⎥⎥Ysh[100shy10001]

$$X' = X + Sh_x . Y$$

$$Y' = Y$$

Composite Transformation

If a transformation of the plane T1 is followed by a second plane transformation T2, then the result itself may be represented by a single transformation T which is the composition of T1 and T2 taken in that order. This is written as T = T1·T2.

Composite transformation can be achieved by concatenation of transformation matrices to obtain a combined transformation matrix.

A combined matrix −

[T][X] = [X] [T1] [T2] [T3] [T4] …. [Tn]

Where [Ti] is any combination of

- Translation
- Scaling
- Shearing
- Rotation
- Reflection

The change in the order of transformation would lead to different results, as in general matrix multiplication is not cumulative, that is [A] . [B] ≠ [B] . [A] and the order of multiplication. The basic purpose of composing transformations is to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformation, one after another.

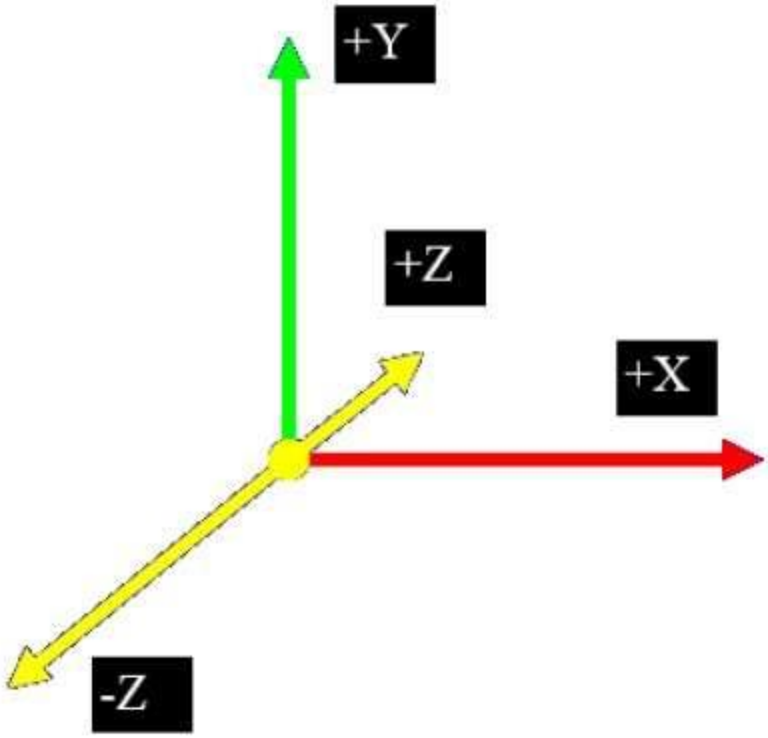
For example, to rotate an object about an arbitrary point (X_p, Y_p), we have to carry out three steps −

- Translate point (X_p, Y_p) to the origin.
- Rotate it about the origin.
- Finally, translate the center of rotation back where it belonged.

3D Computer Graphics

In the 2D system, we use only two coordinates X and Y but in 3D, an extra coordinate Z is added. 3D graphics techniques and their application are fundamental to the entertainment, games, and computer-aided design industries. It is a continuing area of research in scientific visualization.

Furthermore, 3D graphics components are now a part of almost every personal computer and, although traditionally intended for graphics-intensive software such as games, they are increasingly being used by other applications.

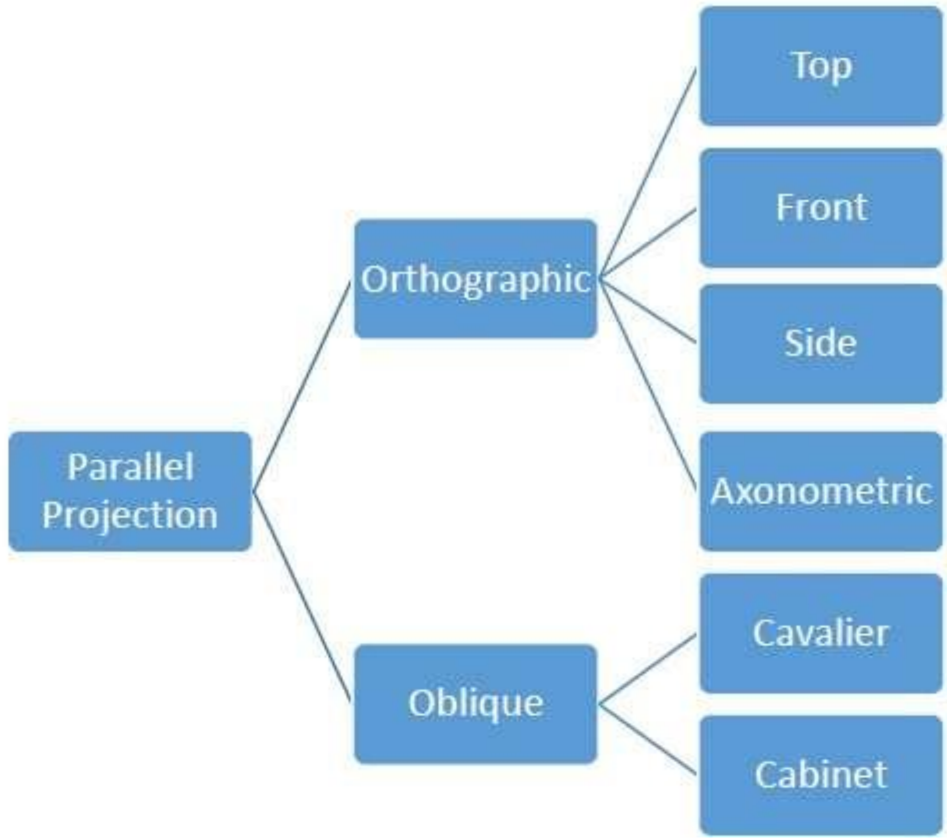


Parallel Projection

Parallel projection discards z-coordinate and parallel lines from each vertex on the object are extended until they intersect the view plane. In parallel projection, we specify a direction of projection instead of center of projection.

In parallel projection, the distance from the center of projection to project plane is infinite. In this type of projection, we connect the projected vertices by line segments which correspond to connections on the original object.

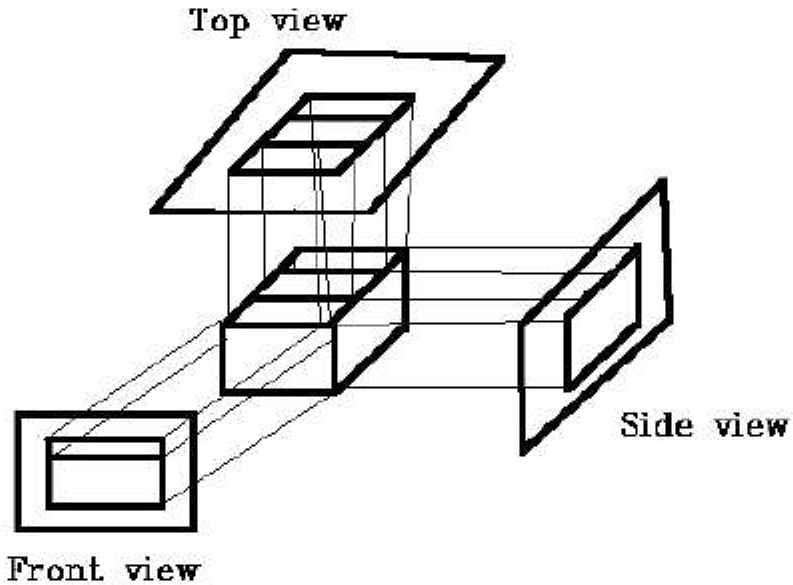
Parallel projections are less realistic, but they are good for exact measurements. In this type of projections, parallel lines remain parallel and angles are not preserved. Various types of parallel projections are shown in the following hierarchy.



Orthographic Projection

In orthographic projection the direction of projection is normal to the projection of the plane. There are three types of orthographic projections −

- Front Projection
- Top Projection
- Side Projection

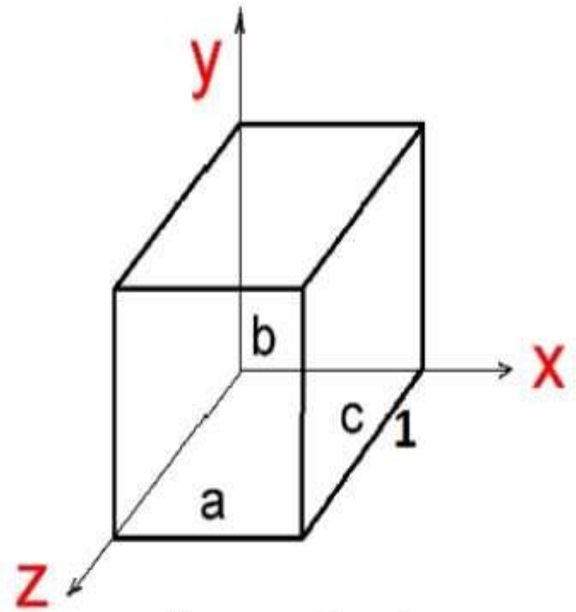


Oblique Projection

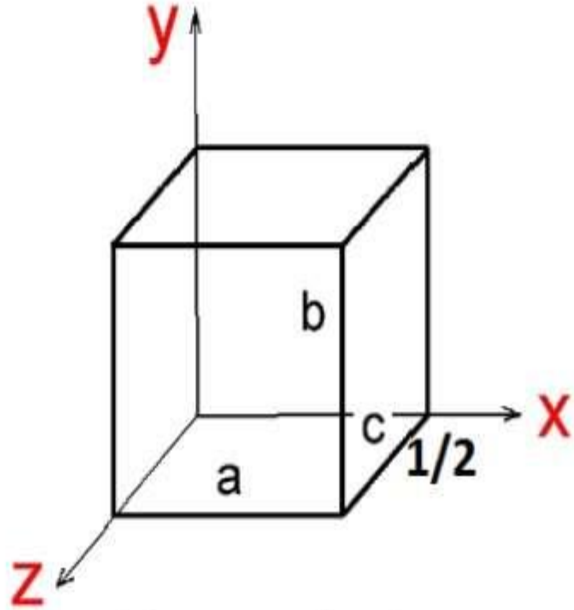
In oblique projection, the direction of projection is not normal to the projection of plane. In oblique projection, we can view the object better than orthographic projection.

There are two types of oblique projections − **Cavalier** and **Cabinet.** The Cavalier projection makes 45° angle with the projection plane. The projection of a line perpendicular to the view plane has the same length as the line itself in Cavalier projection. In a cavalier projection, the foreshortening factors for all three principal directions are equal.

The Cabinet projection makes 63.4° angle with the projection plane. In Cabinet projection, lines perpendicular to the viewing surface are projected at ½ their actual length. Both the projections are shown in the following figure −



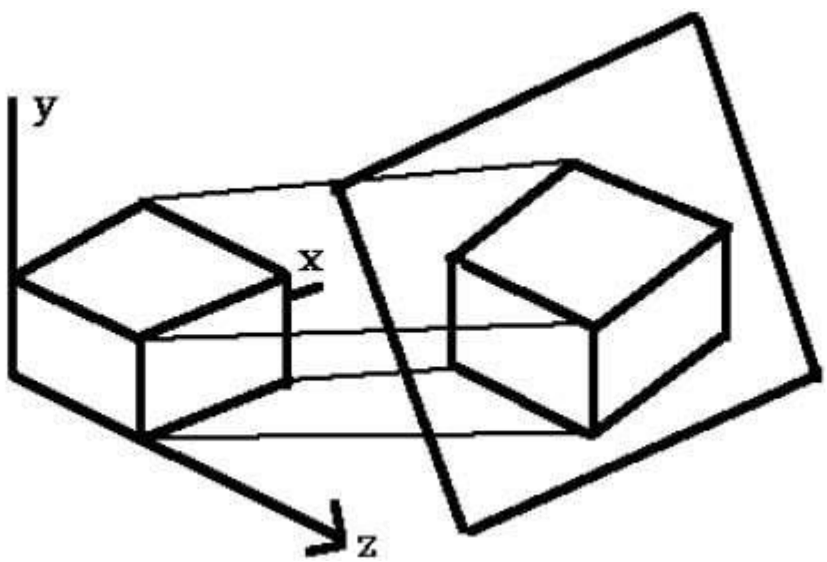
Cavalier Projection



Cabinet Projection

Isometric Projections

Orthographic projections that show more than one side of an object are called **axonometric orthographic projections**. The most common axonometric projection is an **isometric projection** where the projection plane intersects each coordinate axis in the model coordinate system at an equal distance. In this projection parallelism of lines are preserved but angles are not preserved. The following figure shows isometric projection −

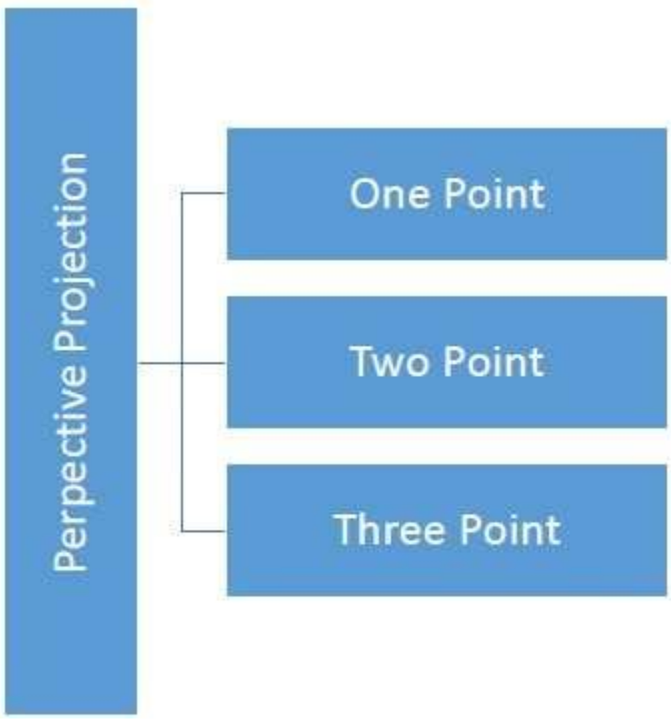


Perspective Projection

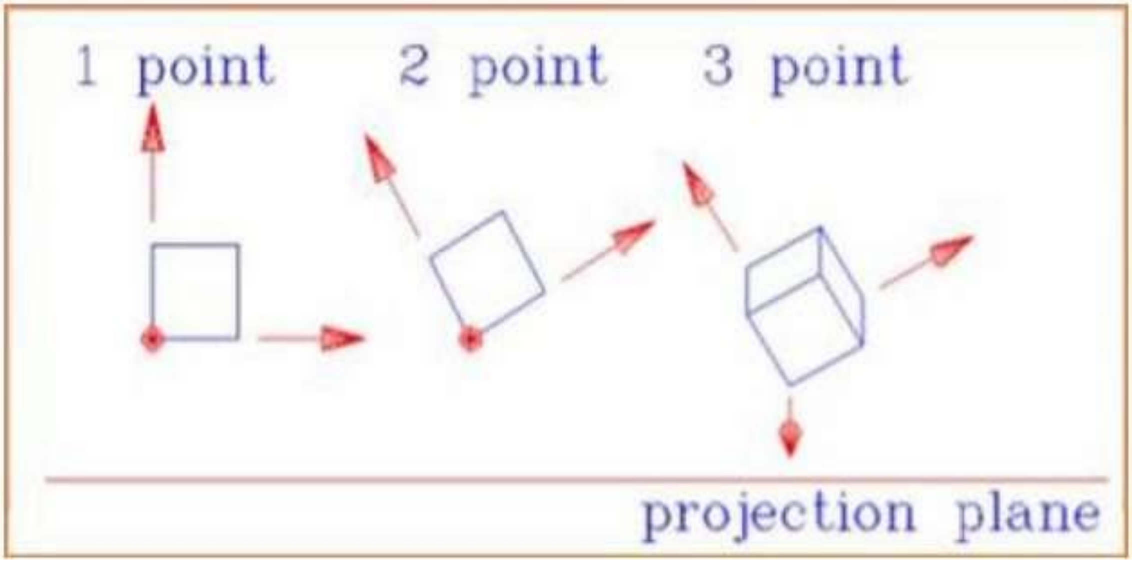
In perspective projection, the distance from the center of projection to project plane is finite and the size of the object varies inversely with distance which looks more realistic.

The distance and angles are not preserved and parallel lines do not remain parallel. Instead, they all converge at a single point called **center of projection** or **projection reference point**. There are 3 types of perspective projections which are shown in the following chart.

- **One point** perspective projection is simple to draw.
- **Two point** perspective projection gives better impression of depth.
- **Three point** perspective projection is most difficult to draw.



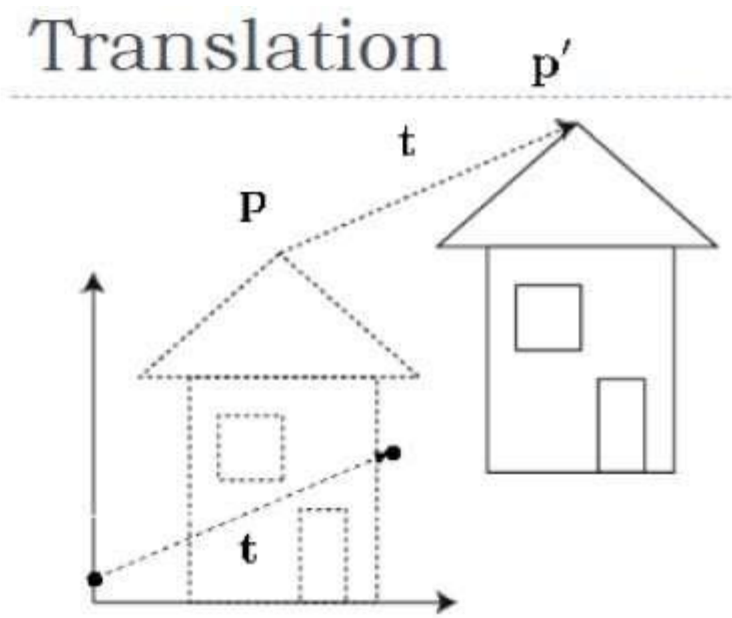
The following figure shows all the three types of perspective projection −



Translation

In 3D translation, we transfer the Z coordinate along with the X and Y coordinates. The process for translation in 3D is similar to 2D translation. A translation moves an object into a different position on the screen.

The following figure shows the effect of translation −



A point can be translated in 3D by adding translation coordinate (t_x, t_y, t_z) to the original coordinate (X, Y, Z) to get the new coordinate (X’, Y’, Z’).

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

P’ = P·T

$$[X' \; Y' \; Z' \; 1] = [X \; Y \; Z \; 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

$$= [X + t_x \quad Y + t_y \quad Z + t_z \quad 1]$$

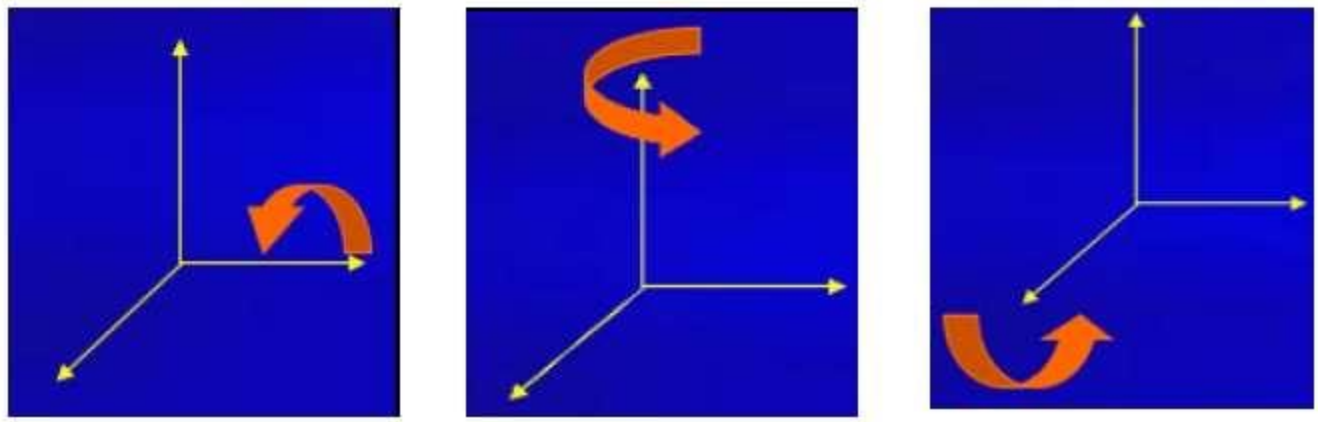
3D Transformation

Rotation

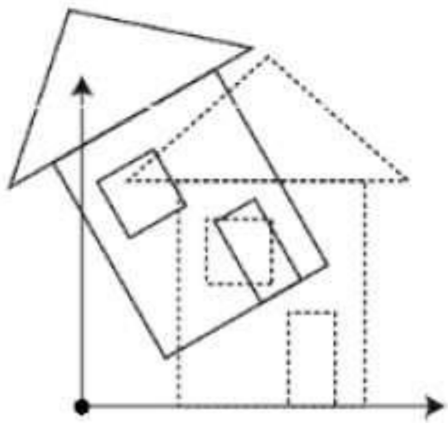
3D rotation is not same as 2D rotation. In 3D rotation, we have to specify the angle of rotation along with the axis of rotation. We can perform 3D rotation about X, Y, and Z axes. They are represented in the matrix form as below −

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

The following figure explains the rotation about various axes −

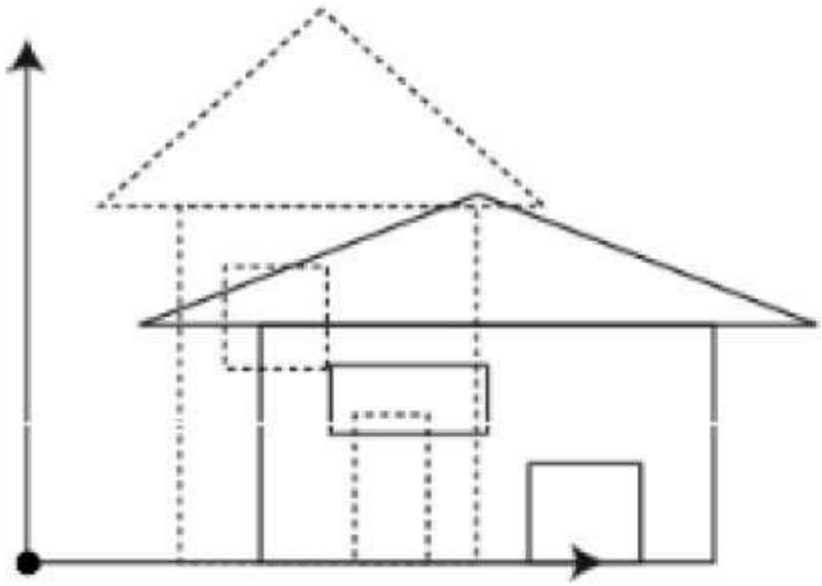


Rotation about x-axis   Rotation about y-axis   Rotation about z-axis



Scaling

You can change the size of an object using scaling transformation. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result. The following figure shows the effect of 3D scaling −



In 3D scaling operation, three coordinates are used. Let us assume that the original coordinates are (X, Y, Z), scaling factors are (S_X, S_Y, S_z) respectively, and the produced coordinates are (X′, Y′, Z′). This can be mathematically represented as shown below −

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

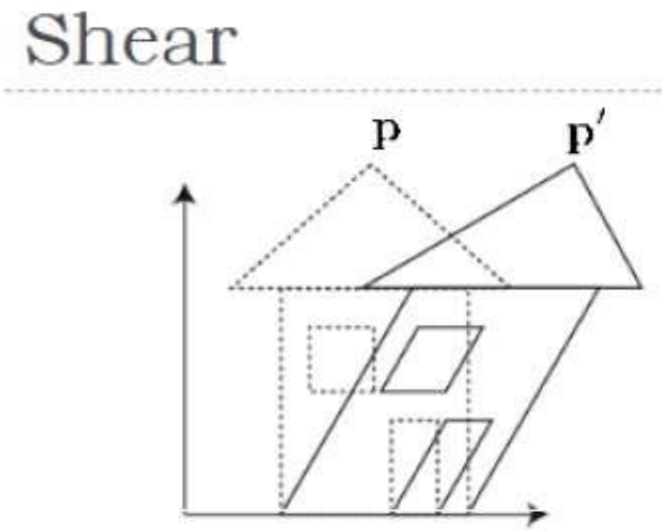
P′ = P·S

$$[X' \; Y' \; Z' \; 1] = [X \; Y \; Z \; 1] \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= [X.S_x \; Y.S_y \; Z.S_z \; 1]$$

Shear

A transformation that slants the shape of an object is called the **shear transformation.** Like in 2D shear, we can shear an object along the X-axis, Y-axis, or Z-axis in 3D.



As shown in the above figure, there is a coordinate P. You can shear it to get a new coordinate P', which can be represented in 3D matrix form as below −

$$Sh = \begin{bmatrix} 1 & sh_{x}^{y} & sh_{x}^{z} & 0 \\ sh_{y}^{x} & 1 & sh_{y}^{z} & 0 \\ sh_{z}^{x} & sh_{z}^{y} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P' = P \cdot Sh$$

$$X' = X + Sh_{x}^{y} Y + Sh_{x}^{z} Z$$

$$Y' = Sh_{y}^{x} X + Y + sh_{y}^{z} Z$$

$$Z' = Sh_{z}^{x} X + Sh_{z}^{y} Y + Z$$

Transformation Matrices

Transformation matrix is a basic tool for transformation. A matrix with n x m dimensions is multiplied with the coordinate of objects. Usually 3 x 3 or 4 x 4 matrices are used for transformation. For example, consider the following matrix for various operation.

Computer Graphics Curves

In computer graphics, we often need to draw different types of objects onto the screen. Objects are not flat all the time and we need to draw curves many times to draw an object.

Types of Curves

A curve is an infinitely large set of points. Each point has two neighbors except endpoints. Curves can be broadly classified into three categories − **explicit, implicit,** and **parametric curves**.

Implicit Curves

Implicit curve representations define the set of points on a curve by employing a procedure that can test to see if a point in on the curve. Usually, an implicit curve is defined by an implicit function of the form −

$$f(x, y) = 0$$

It can represent multivalued curves (multiple y values for an x value). A common example is the circle, whose implicit representation is

$$\mathbf{x^2 + y^2 - R^2 = 0}$$

Explicit Curves

A mathematical function y = f(x) can be plotted as a curve. Such a function is the explicit representation of the curve. The explicit representation is not general, since it cannot represent vertical lines and is also single-valued. For each value of x, only a single value of y is normally computed by the function.

Parametric Curves

Curves having parametric form are called parametric curves. The explicit and implicit curve representations can be used only when the function is known. In practice the parametric curves are used. A two-dimensional parametric curve has the following form −

$$P(t) = f(t), g(t) \text{ or } P(t) = x(t), y(t)$$

The functions f and g become the (x, y) coordinates of any point on the curve, and the points are obtained when the parameter t is varied over a certain interval [a, b], normally [0, 1].

Bezier Curves

Bezier curve is discovered by the French engineer **Pierre Bézier**. These curves can be generated under the control of other points. Approximate tangents by using control points are used to generate curve. The Bezier curve can be represented mathematically as −

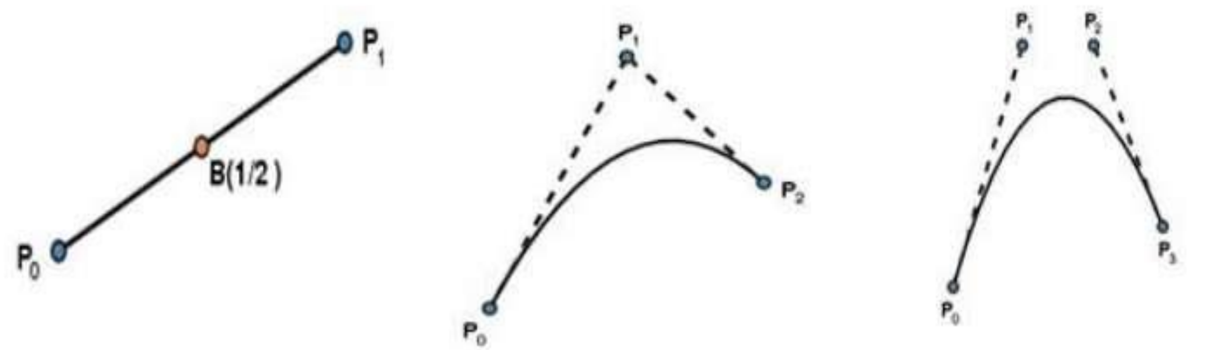
$$\sum_{k=0}^{n} P_i B_i^n(t)$$

Where p_i is the set of points and $B_i^n(t)$ represents the Bernstein polynomials which are given by −

$$B_i^n(t) = \binom{n}{i} (1-t)^{n-i} t^i$$

Where **n** is the polynomial degree, **i** is the index, and **t** is the variable.

The simplest Bézier curve is the straight line from the point P_0 to P_1. A quadratic Bezier curve is determined by three control points. A cubic Bezier curve is determined by four control points.



Simple Bezier Curve Quadratic Bazier Curve Cubic Bazier Curve

Properties of Bezier Curves

Bezier curves have the following properties −

- They generally follow the shape of the control polygon, which consists of the segments joining the control points.
- They always pass through the first and last control points.
- They are contained in the convex hull of their defining control points.

- The degree of the polynomial defining the curve segment is one less that the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial.
- A Bezier curve generally follows the shape of the defining polygon.
- The direction of the tangent vector at the end points is same as that of the vector determined by first and last segments.
- The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
- No straight line intersects a Bezier curve more times than it intersects its control polygon.
- They are invariant under an affine transformation.
- Bezier curves exhibit global control means moving a control point alters the shape of the whole curve.
- A given Bezier curve can be subdivided at a point t=t0 into two Bezier segments which join together at the point corresponding to the parameter value t=t0.

B-Spline Curves

The Bezier-curve produced by the Bernstein basis function has limited flexibility.

- First, the number of specified polygon vertices fixes the order of the resulting polynomial which defines the curve.
- The second limiting characteristic is that the value of the blending function is nonzero for all parameter values over the entire curve.

The B-spline basis contains the Bernstein basis as the special case. The B-spline basis is non-global.

A B-spline curve is defined as a linear combination of control points Pi and B-spline basis function $N_{i,}$ k (t) given by

$$C(t)=\sum_{i=0}^{n} P_i N_{i,k}(t),\quad n \geq k-1,\quad t \epsilon [t_{k-1}, t_{n+1}]$$

Where,

- $\{p_i$: i=0, 1, 2….n$\}$ are the control points
- k is the order of the polynomial segments of the B-spline curve. Order k means that the curve is made up of piecewise polynomial segments of degree k - 1,
- the $N_{i,k}(t)$ are the "normalized B-spline blending functions". They are described by the order k and by a non-decreasing sequence of real numbers normally called the "knot sequence".

$$t_i : i=0, \ldots n+K$$

The N_i, k functions are described as follows −

$$N_{i,1}(t) = \begin{cases} 1, & \text{if } u \in [t_i, t_{i+1}) \\ 0, & \text{Otherwise} \end{cases}$$

and if k > 1,

$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k-1}} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t)$$

and

$$t \in [t_{k-1}, t_{n+1})$$

Properties of B-spline Curve

B-spline curves have the following properties −

- The sum of the B-spline basis functions for any parameter value is 1.
- Each basis function is positive or zero for all parameter values.
- Each basis function has precisely one maximum value, except for k=1.
- The maximum order of the curve is equal to the number of vertices of defining polygon.
- The degree of B-spline polynomial is independent on the number of vertices of defining polygon.
- B-spline allows the local control over the curve surface because each vertex affects the shape of a curve only over a range of parameter values where its associated basis function is nonzero.
- The curve exhibits the variation diminishing property.
- The curve generally follows the shape of defining polygon.
- Any affine transformation can be applied to the curve by applying it to the vertices of defining polygon.
- The curve line within the convex hull of its defining polygon.

Computer Graphics Surfaces

Polygon Surfaces

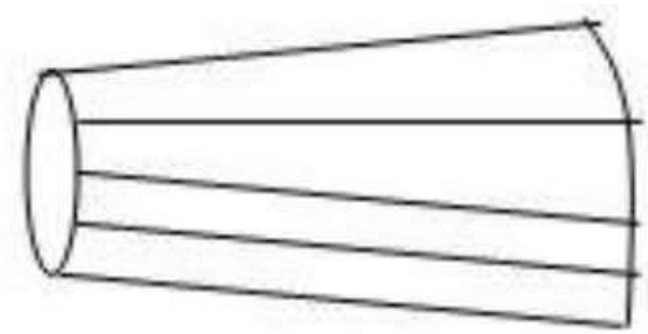
Objects are represented as a collection of surfaces. 3D object representation is divided into two categories.

- **Boundary Representations (B-reps)** − It describes a 3D object as a set of surfaces that separates the object interior from the environment.

- **Space–partitioning representations** − It is used to describe interior properties, by partitioning the spatial region containing an object into a set of small, non-overlapping, contiguous solids (usually cubes).

The most commonly used boundary representation for a 3D graphics object is a set of surface polygons that enclose the object interior. Many graphics system use this method. Set of polygons are stored for object description. This simplifies and speeds up the surface rendering and display of object since all surfaces can be described with linear equations.

The polygon surfaces are common in design and solid-modeling applications, since their **wireframe display** can be done quickly to give general indication of surface structure. Then realistic scenes are produced by interpolating shading patterns across polygon surface to illuminate.

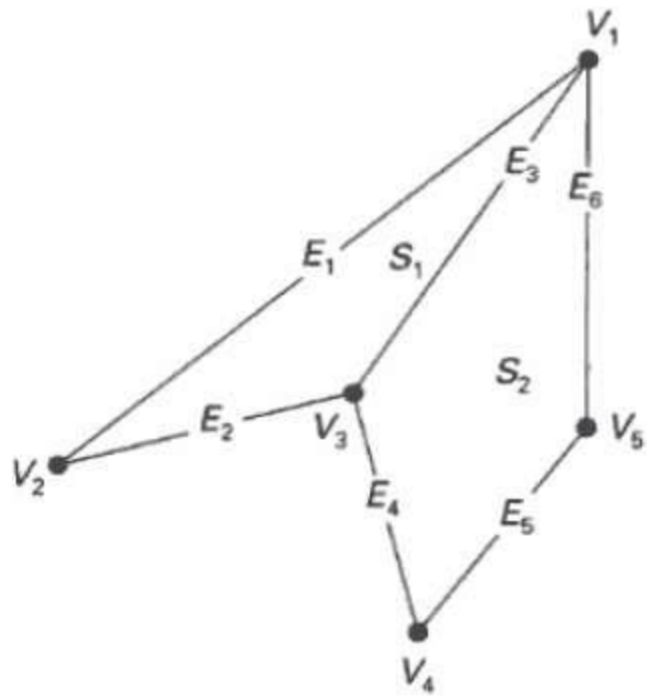


A 3D object represented by polygons

Polygon Tables

In this method, the surface is specified by the set of vertex coordinates and associated attributes. As shown in the following figure, there are five vertices, from v_1 to v_5.

- Each vertex stores x, y, and z coordinate information which is represented in the table as v_1: x_1, y_1, z_1.
- The Edge table is used to store the edge information of polygon. In the following figure, edge E_1 lies between vertex v_1 and v_2 which is represented in the table as E_1: v_1, v_2.
- Polygon surface table stores the number of surfaces present in the polygon. From the following figure, surface S_1 is covered by edges E_1, E_2 and E_3 which can be represented in the polygon surface table as S_1: E_1, E_2, and E_3.



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

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

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

Plane Equations

The equation for plane surface can be expressed as −

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

Where (x, y, z) is any point on the plane, and the coefficients A, B, C, and D are constants describing the spatial properties of the plane. We can obtain the values of A, B, C, and D by solving a set of three plane equations using the coordinate values for three non collinear points in the plane. Let us assume that three vertices of the plane are (x_1, y_1, z_1), (x_2, y_2, z_2) and (x_3, y_3, z_3).

Let us solve the following simultaneous equations for ratios A/D, B/D, and C/D. You get the values of A, B, C, and D.

$$(A/D)\, x_1 + (B/D)\, y_1 + (C/D)\, z_1 = -1$$

$$(A/D)\, x_2 + (B/D)\, y_2 + (C/D)\, z_2 = -1$$

$$(A/D)\, x_3 + (B/D)\, y_3 + (C/D)\, z_3 = -1$$

To obtain the above equations in determinant form, apply Cramer's rule to the above equations.

$$A = \begin{bmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{bmatrix} \quad B = \begin{bmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{bmatrix} \quad C = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \quad D = -\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix}$$

For any point (x, y, z) with parameters A, B, C, and D, we can say that −

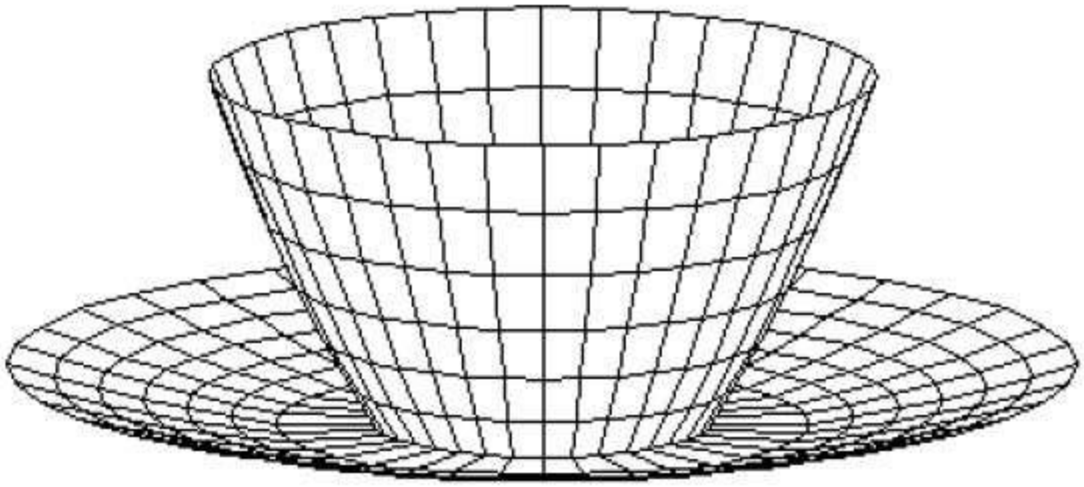
- $Ax + By + Cz + D \neq 0$ means the point is not on the plane.
- $Ax + By + Cz + D < 0$ means the point is inside the surface.
- $Ax + By + Cz + D > 0$ means the point is outside the surface.

Polygon Meshes

3D surfaces and solids can be approximated by a set of polygonal and line elements. Such surfaces are called **polygonal meshes**. In polygon mesh, each edge is shared by at most two polygons. The set of polygons or faces, together form the "skin" of the object.

This method can be used to represent a broad class of solids/surfaces in graphics. A polygonal mesh can be rendered using hidden surface removal algorithms. The polygon mesh can be represented by three ways −

- Explicit representation
- Pointers to a vertex list
- Pointers to an edge list



Advantages

- It can be used to model almost any object.
- They are easy to represent as a collection of vertices.
- They are easy to transform.
- They are easy to draw on computer screen.

Disadvantages

- Curved surfaces can only be approximately described.
- It is difficult to simulate some type of objects like hair or liquid.

Visible Surface Detection

When we view a picture containing non-transparent objects and surfaces, then we cannot see those objects from view which are behind from objects closer to eye. We must remove these hidden surfaces to get a realistic screen image. The identification and removal of these surfaces is called **Hidden-surface problem**.

There are two approaches for removing hidden surface problems − **Object-Space method** and **Image-space method**. The Object-space method is implemented in physical coordinate system and image-space method is implemented in screen coordinate system.

When we want to display a 3D object on a 2D screen, we need to identify those parts of a screen that are visible from a chosen viewing position.

Depth Buffer (Z-Buffer) Method

This method is developed by Cutmull. It is an image-space approach. The basic idea is to test the Z-depth of each surface to determine the closest (visible) surface.

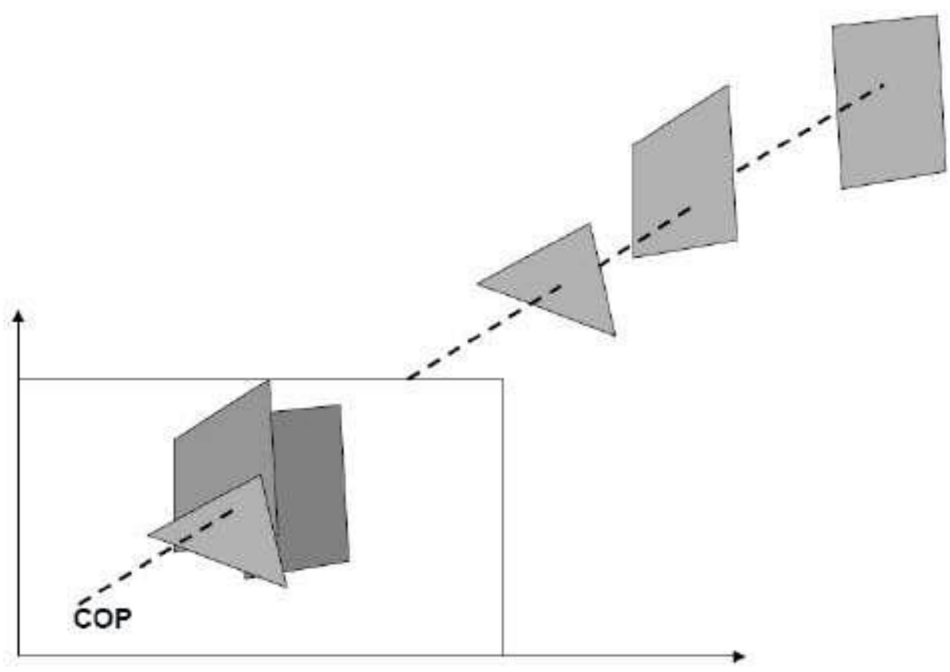
In this method each surface is processed separately one pixel position at a time across the surface. The depth values for a pixel are compared and the closest (smallest z) surface determines the color to be displayed in the frame buffer.

It is applied very efficiently on surfaces of polygon. Surfaces can be processed in any order. To override the closer polygons from the far ones, two buffers named **frame buffer** and **depth buffer,** are used.

Depth buffer is used to store depth values for (x, y) position, as surfaces are processed (0 ≤ depth ≤ 1).

The **frame buffer** is used to store the intensity value of color value at each position (x, y).

The z-coordinates are usually normalized to the range [0, 1]. The 0 value for z-coordinate indicates back clipping pane and 1 value for z-coordinates indicates front clipping pane.



Algorithm

Step-1 − Set the buffer values −

Depthbuffer (x, y) = 0

Framebuffer (x, y) = background color

Step-2 − Process each polygon (One at a time)

For each projected (x, y) pixel position of a polygon, calculate depth z.

If Z > depthbuffer (x, y)

Compute surface color,

set depthbuffer (x, y) = z,

framebuffer (x, y) = surfacecolor (x, y)

Advantages

- It is easy to implement.
- It reduces the speed problem if implemented in hardware.
- It processes one object at a time.

Disadvantages

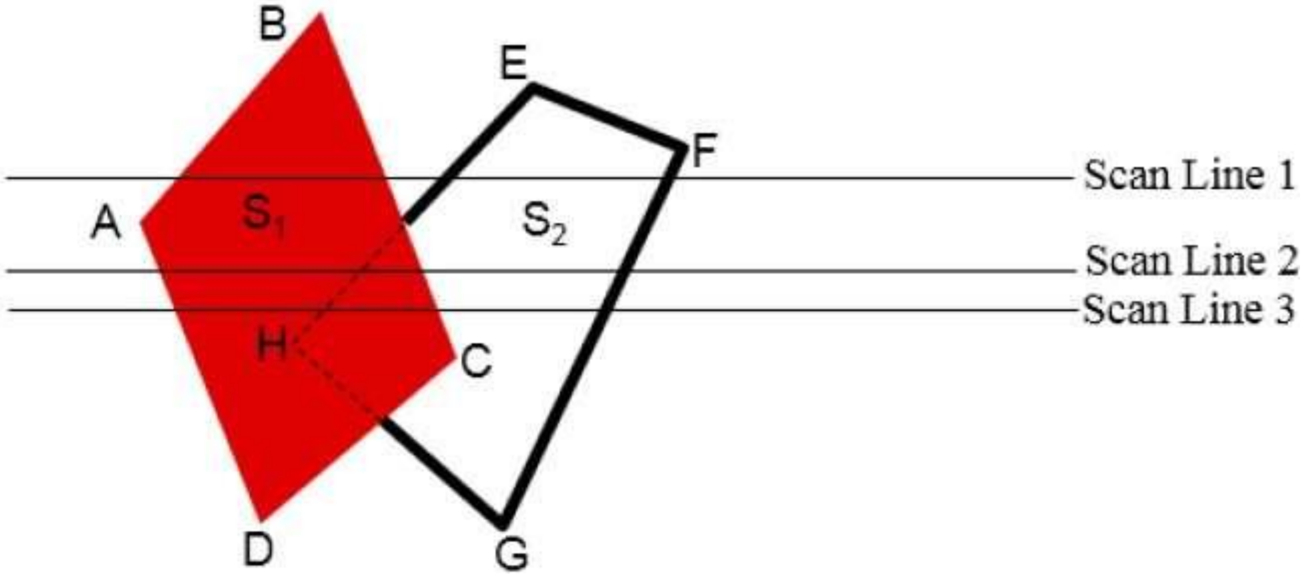
- It requires large memory.
- It is time consuming process.

Scan-Line Method

It is an image-space method to identify visible surface. This method has a depth information for only single scan-line. In order to require one scan-line of depth values, we must group and process all polygons intersecting a given scan-line at the same time before processing the next scan-line. Two important tables, **edge table** and **polygon table,** are maintained for this.

The Edge Table − It contains coordinate endpoints of each line in the scene, the inverse slope of each line, and pointers into the polygon table to connect edges to surfaces.

The Polygon Table − It contains the plane coefficients, surface material properties, other surface data, and may be pointers to the edge table.



To facilitate the search for surfaces crossing a given scan-line, an active list of edges is formed. The active list stores only those edges that cross the scan-line in order of increasing x. Also a flag is set for each surface to indicate whether a position along a scan-line is either inside or outside the surface.

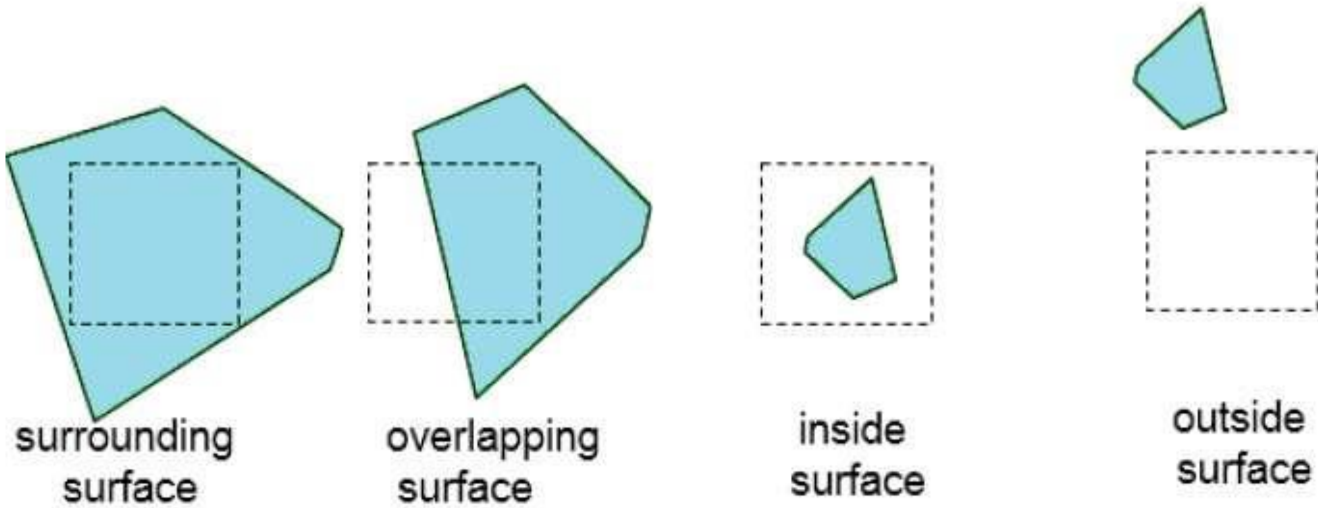
Pixel positions across each scan-line are processed from left to right. At the left intersection with a surface, the surface flag is turned on and at the right, the flag is turned off. You only need to perform depth calculations when multiple surfaces have their flags turned on at a certain scan-line position.

Area-Subdivision Method

The area-subdivision method takes advantage by locating those view areas that represent part of a single surface. 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.

Continue this process until the subdivisions are easily analyzed as belonging to a single surface or until they are reduced to the size of a single pixel. An easy way to do this is to successively divide the area into four equal parts at each step. There are four possible relationships that a surface can have with a specified area boundary.

- **Surrounding surface** − One that completely encloses the area.
- **Overlapping surface** − One that is partly inside and partly outside the area.
- **Inside surface** − One that is completely inside the area.
- **Outside surface** − One that is completely outside the area.



The tests for determining surface visibility within an area can be stated in terms of these four classifications. No further subdivisions of a specified area are needed if one of the following conditions is true −

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

Back-Face Detection

A fast and simple object-space method for identifying the back faces of a polyhedron is based on the "inside-outside" tests. A point (x, y, z) is "inside" a polygon surface with plane parameters A, B, C, and D if When an inside point is along the line of sight to the surface, the polygon must be a back face (we are inside that face and cannot see the front of it from our viewing position).

We can simplify this test by considering the normal vector **N** to a polygon surface, which has Cartesian components (A, B, C).

In general, if V is a vector in the viewing direction from the eye (or "camera") position, then this polygon is a back face if

$$\mathbf{V.N} > \mathbf{0}$$

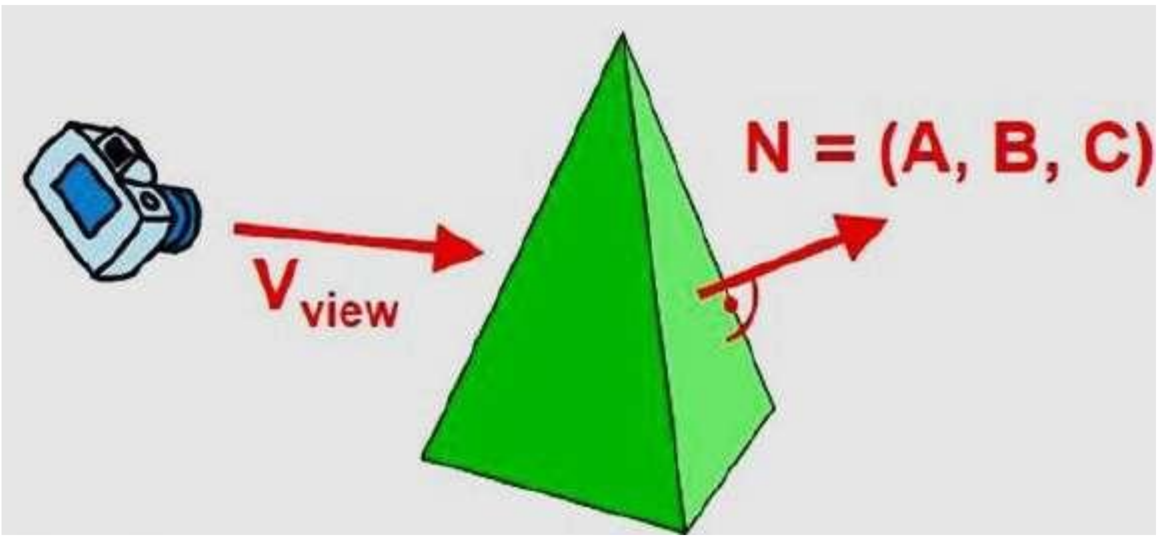
Furthermore, if object descriptions are converted to projection coordinates and your viewing direction is parallel to the viewing z-axis, then −

$$V = (0, 0, V_z) \text{ and } V.N = V_Z C$$

So that we only need to consider the sign of C the component of the normal vector **N**.

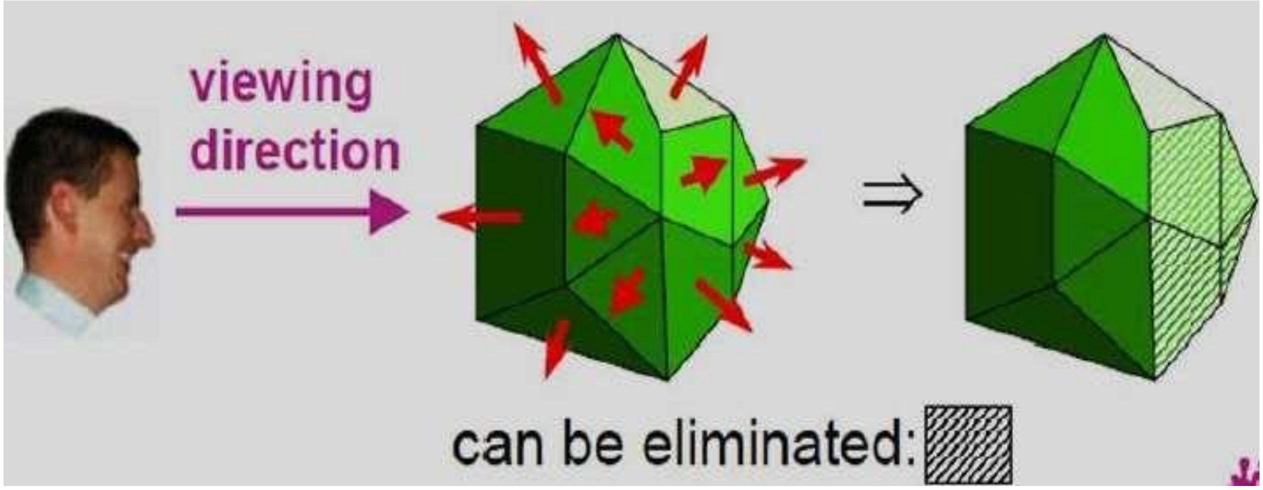
In a right-handed viewing system with viewing direction along the negative ZVZV axis, the polygon is a back face if C < 0. Also, we cannot see any face whose normal has z component C = 0, since your viewing direction is towards that polygon. Thus, in general, we can label any polygon as a back face if its normal vector has a z component value −

$$C <= 0$$



Similar methods can be used in packages that employ a left-handed viewing system. In these packages, plane parameters A, B, C and D can be calculated from polygon vertex coordinates specified in a clockwise direction (unlike the counterclockwise direction used in a right-handed system).

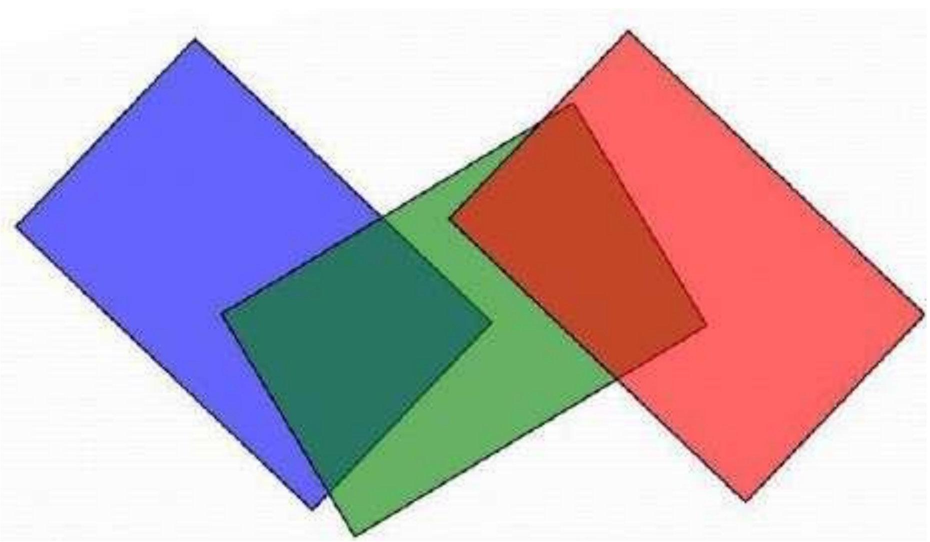
Also, back faces have normal vectors that point away from the viewing position and are identified by C >= 0 when the viewing direction is along the positive ZvZv axis. By examining parameter C for the different planes defining an object, we can immediately identify all the back faces.



A-Buffer Method

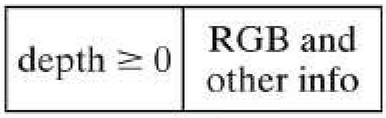
The A-buffer method is an extension of the depth-buffer method. The A-buffer method is a visibility detection method developed at Lucas film Studios for the rendering system Renders Everything You Ever Saw (REYES).

The A-buffer expands on the depth buffer method to allow transparencies. The key data structure in the A-buffer is the accumulation buffer.

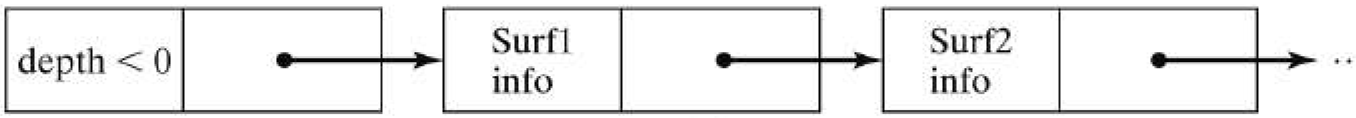


Each position in the A-buffer has two fields −

- **Depth field** − It stores a positive or negative real number
- **Intensity field** − It stores surface-intensity information or a pointer value



(a)



(b)

If depth >= 0, the number stored at that position is the depth of a single surface overlapping the corresponding pixel area. The intensity field then stores the RGB components of the surface color at that point and the percent of pixel coverage.

If depth < 0, it indicates multiple-surface contributions to the pixel intensity. The intensity field then stores a pointer to a linked list of surface data. The surface buffer in the A-buffer includes −

- RGB intensity components
- Opacity Parameter
- Depth
- Percent of area coverage
- Surface identifier

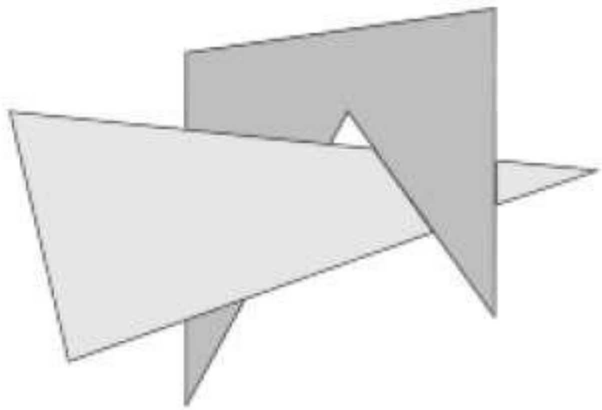
The algorithm proceeds just like the depth buffer algorithm. The depth and opacity values are used to determine the final color of a pixel.

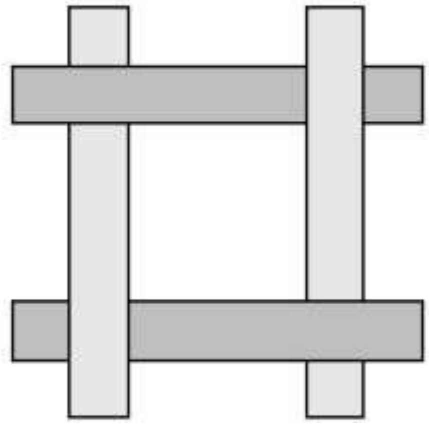
Depth Sorting Method

Depth sorting method uses both image space and object-space operations. The depth-sorting method performs two basic functions −

- First, the surfaces are sorted in order of decreasing depth.
- Second, the surfaces are scan-converted in order, starting with the surface of greatest depth.

The scan conversion of the polygon surfaces is performed in image space. This method for solving the hidden-surface problem is often referred to as the **painter's algorithm**. The following figure shows the effect of depth sorting −





The algorithm begins by sorting by depth. For example, the initial “depth” estimate of a polygon may be taken to be the closest z value of any vertex of the polygon.

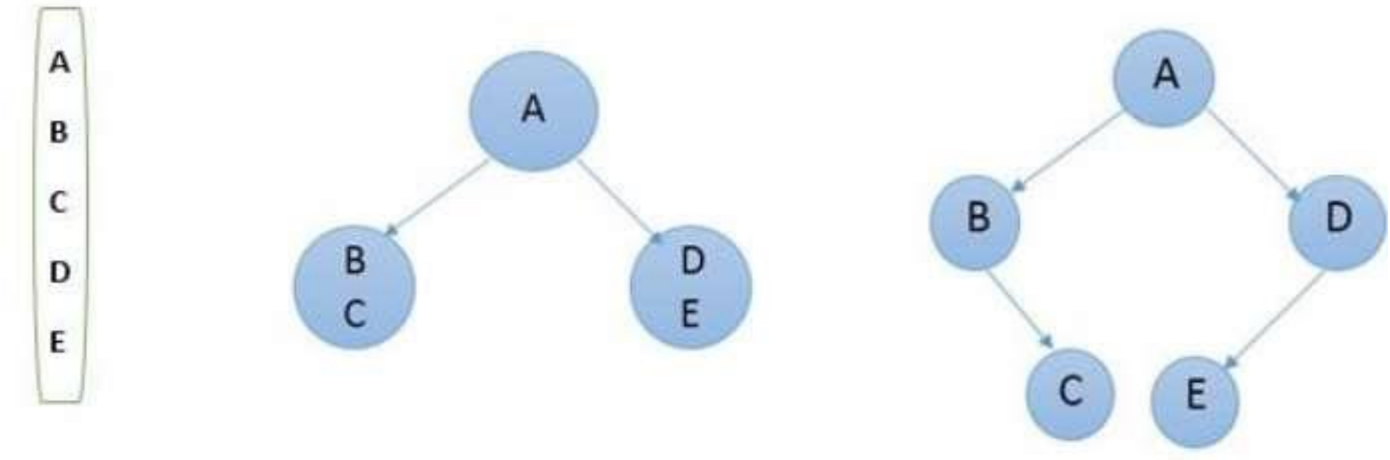
Let us take the polygon P at the end of the list. Consider all polygons Q whose z-extents overlap P’s. Before drawing P, we make the following tests. If any of the following tests is positive, then we can assume P can be drawn before Q.

- Do the x-extents not overlap?
- Do the y-extents not overlap?
- Is P entirely on the opposite side of Q’s plane from the viewpoint?
- Is Q entirely on the same side of P’s plane as the viewpoint?
- Do the projections of the polygons not overlap?

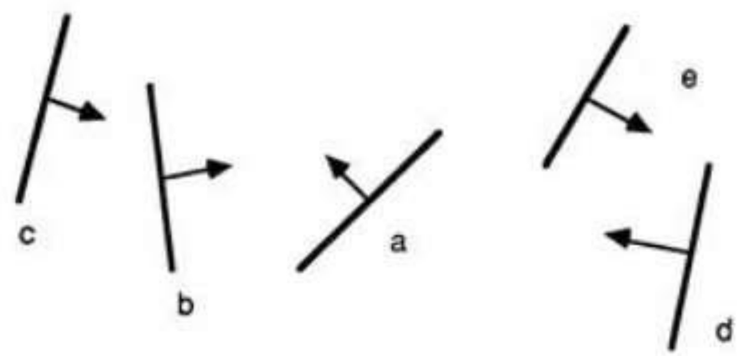
If all the tests fail, then we split either P or Q using the plane of the other. The new cut polygons are inserting into the depth order and the process continues. Theoretically, this partitioning could generate $O(n^2)$ individual polygons, but in practice, the number of polygons is much smaller.

Binary Space Partition (BSP) Trees

Binary space partitioning is used to calculate visibility. To build the BSP trees, one should start with polygons and label all the edges. Dealing with only one edge at a time, extend each edge so that it splits the plane in two. Place the first edge in the tree as root. Add subsequent edges based on whether they are inside or outside. Edges that span the extension of an edge that is already in the tree are split into two and both are added to the tree.



(a) (b) (c)



(d)

- From the above figure, first take **A** as a root.
- Make a list of all nodes in figure (a).
- Put all the nodes that are in front of root **A** to the left side of node **A** and put all those nodes that are behind the root **A** to the right side as shown in figure (b).
- Process all the front nodes first and then the nodes at the back.
- As shown in figure (c), we will first process the node **B.** As there is nothing in front of the node **B**, we have put NIL. However, we have node **C** at back of node **B**, so node **C** will go to the right side of node **B.**
- Repeat the same process for the node **D.**

Computer Graphics Fractals

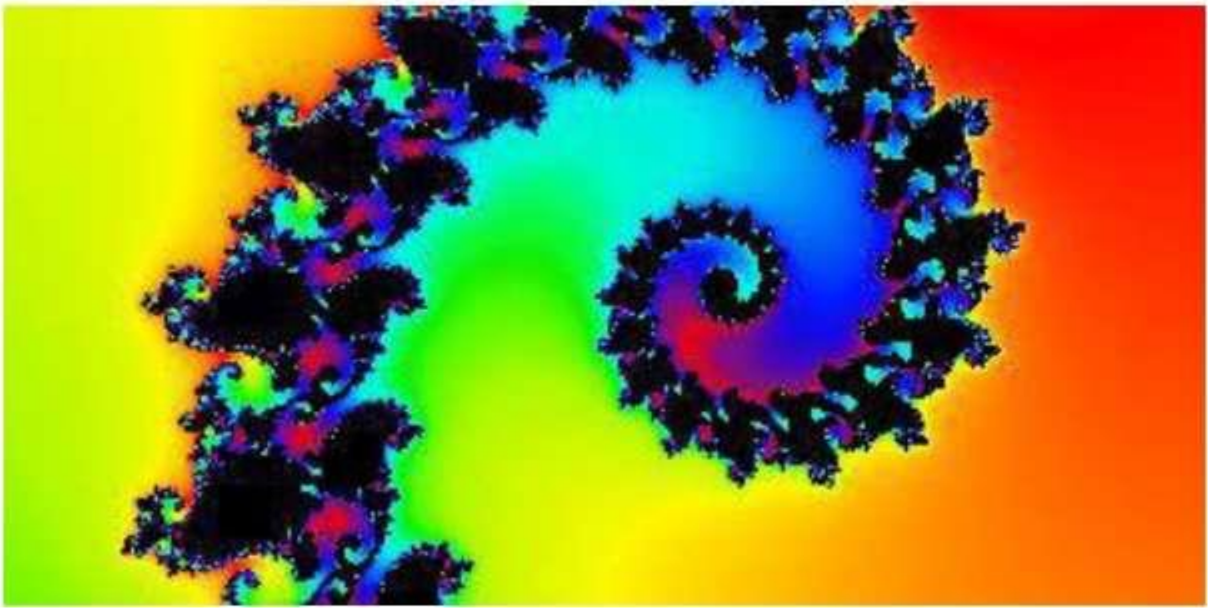
A French/American mathematician Dr Benoit Mandelbrot discovered Fractals. The word fractal was derived from a Latin word *fractus* which means broken.

What are Fractals in Computer Graphics?

Fractals are very complex pictures generated by a computer from a single formula. They are created using iterations. This means one formula is repeated with slightly different values over and over again, taking into account the results from the previous iteration.

Fractals are used in many areas such as −

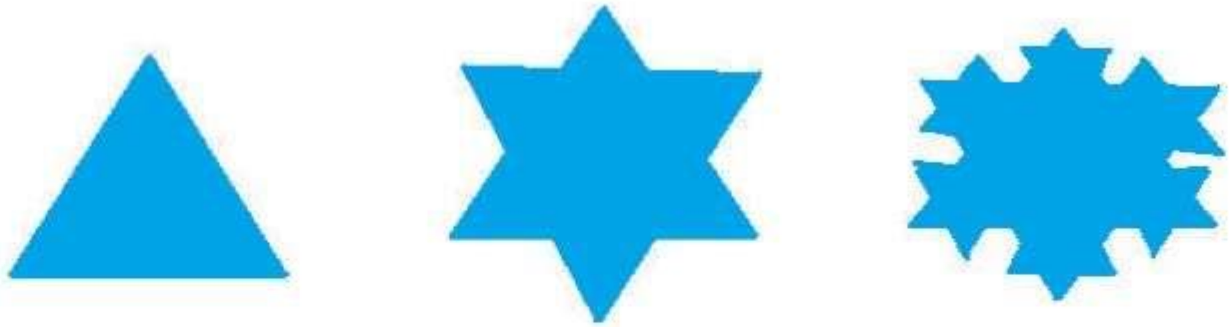
- **Astronomy** − For analyzing galaxies, rings of Saturn, etc.
- **Biology/Chemistry** − For depicting bacteria cultures, Chemical reactions, human anatomy, molecules, plants,
- **Others** − For depicting clouds, coastline and borderlines, data compression, diffusion, economy, fractal art, fractal music, landscapes, special effect, etc.



Generation of Computer Graphics Fractals

Fractals can be generated by repeating the same shape over and over again as shown in the following figure. In figure (a) shows an equilateral triangle. In figure (b), we can see that the triangle is repeated to create a star-like shape. In figure (c), we can see that the star shape in figure (b) is repeated again and again to create a new shape.

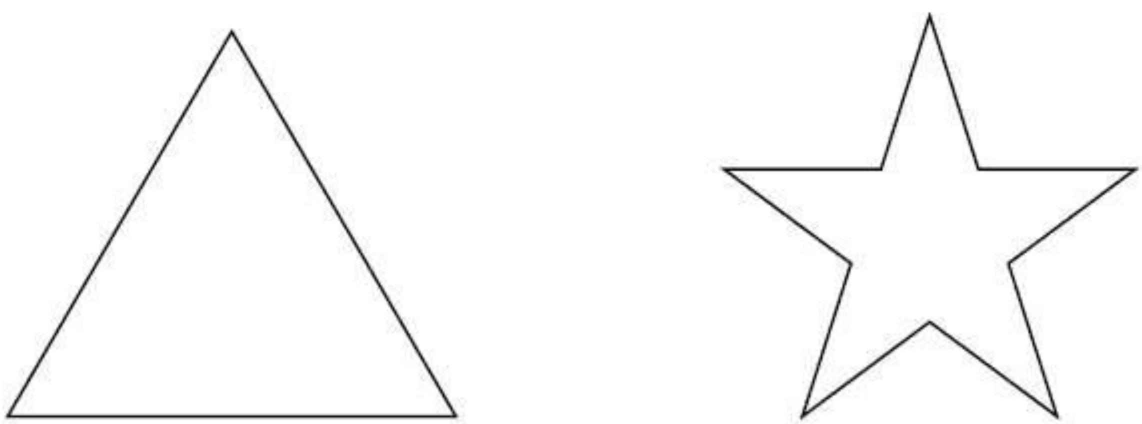
We can do unlimited number of iteration to create a desired shape. In programming terms, recursion is used to create such shapes.



(a) Zeroth Generation (b) First Generation (c) Second Generation

Geometric Fractals

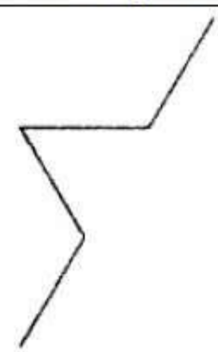
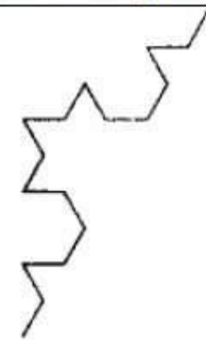
Geometric fractals deal with shapes found in nature that have non-integer or fractal dimensions. To geometrically construct a deterministic (nonrandom) self-similar fractal, we start with a given geometric shape, called the **initiator**. Subparts of the initiator are then replaced with a pattern, called the **generator**.



Initiator Generator

As an example, if we use the initiator and generator shown in the above figure, we can construct good pattern by repeating it. Each straight-line segment in the initiator is replaced with four equal-length line segments at each step. The scaling factor is 1/3, so the fractal dimension is $D = \ln 4/\ln 3 \approx 1.2619$.

Also, the length of each line segment in the initiator increases by a factor of 4/3 at each step, so that the length of the fractal curve tends to infinity as more detail is added to the curve as shown in the following figure –

Segment Length = 1	**Segment Length = 1/3**	**Segment Length = 1/9**
		
Length = 1	Length = 4/3	Length = 16/9