CSC 412 COMPUTER GRAPHICS AND VISUALIZATION (LECTURE NOTES)

The term *computer graphics* describes any use of computers to create and manipulate images. It involves the introduction of algorithmic and mathematical tools that can be used to create all kinds of images—realistic visual effects, informative technical illustrations, or beautiful computer animations.
Graphics can be two- or three-dimensional; images can be completely synthetic or can be produced by manipulating photographs.

Advantages of Computer Graphics

1. ***Realism***: Computer graphics can create highly realistic and immersive visual experiences, enhancing the quality of simulations, movies, and video games.
2. ***Flexibility***: Digital graphics can be easily modified, allowing quick changes and iterations without the need to recreate physical artifacts.
3. ***Efficiency***: Digital distribution of graphics through the internet or other digital mediums enables widespread access and reduces production and distribution costs.
4. ***Interactivity***: In interactive applications, users can engage with the graphics, making it more engaging and informative.
5. ***Accuracy***: Computer graphics can represent precise data, making it valuable in scientific, engineering, and medical fields.

Applications of Computer Graphics

Almost any endeavour can make some use of computer graphics, but the major consumers of computer graphics technology include the following industries:

1. **Video games** increasingly use sophisticated 3D models and rendering algorithms.
2. **Cartoons** are often rendered directly from 3D models. Many traditional 2D cartoons use backgrounds rendered from 3D models, which allows a continuously moving viewpoint without huge amounts of artist time.
3. **Visual effects** use almost all types of computer graphics technology. Almost every modern film uses digital compositing to superimpose backgrounds with separately filmed foregrounds. Many films also use 3D modeling and animation to create synthetic environments, objects, and even characters that most viewers will never suspect are not real.
4. **Animated films** use many of the same techniques that are used for visual effects, but without necessarily aiming for images that look real.
5. **CAD/CAM** stands for *computer-aided design* and *computer-aided manufacturing*. These fields use computer technology to design parts and products on the computer and then, using these virtual designs, to guide the manufacturing process. For example, many mechanical parts are designed in a 3D computer modeling package and then automatically produced on a computer-controlled milling device.
6. **Simulation:** simulation can be viewed as an accurate video gaming. For example, a flight simulator uses sophisticated 3D graphics to simulate the experience of flying an airplane. Such simulations can be extremely useful for initial training in safety-critical domains such as driving, and for scenario training for experienced users such as specific fire-fighting situations that are too
costly or dangerous to create physically.

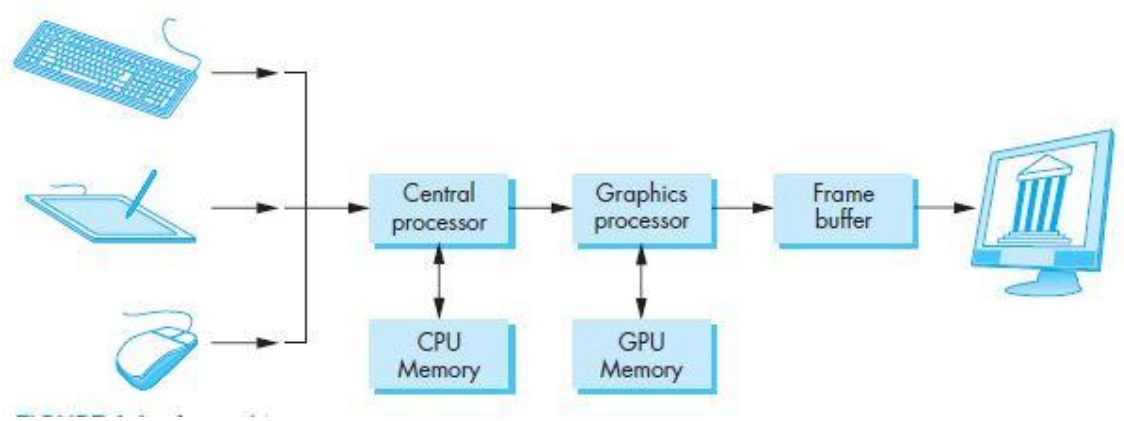
7. **Medical imaging** creates meaningful images of scanned patient data. For example, a computed tomography (CT) dataset is composed of a large 3D rectangular array of density values. Computer graphics is used to create shaded images that help doctors extract the most salient information from such data.
8. **Information visualization** creates images of data that do not necessarily have a "natural" visual depiction. For example, the temporal trend of the price of ten different stocks does not have an obvious visual depiction, but clever graphing techniques can help humans see the patterns in such data.

Graphics System

A computer graphics system is a computer system; as such, it must have all the components of a general-purpose computer system. Let us start with the high-level view of a graphics system. There are six major elements in our system:

1. Input devices
2. Central Processing Unit
3. Graphics Processing Unit
4. Memory
5. Frame buffer
6. Output devices

This model is general enough to include workstations and personal computers, interactive game systems, mobile phones, GPS systems, and sophisticated image generation systems. Although most of the components are present in a standard computer, it is the way each element is specialized for computer graphics that characterizes this diagram as a portrait of a graphics system.



Frame Buffers

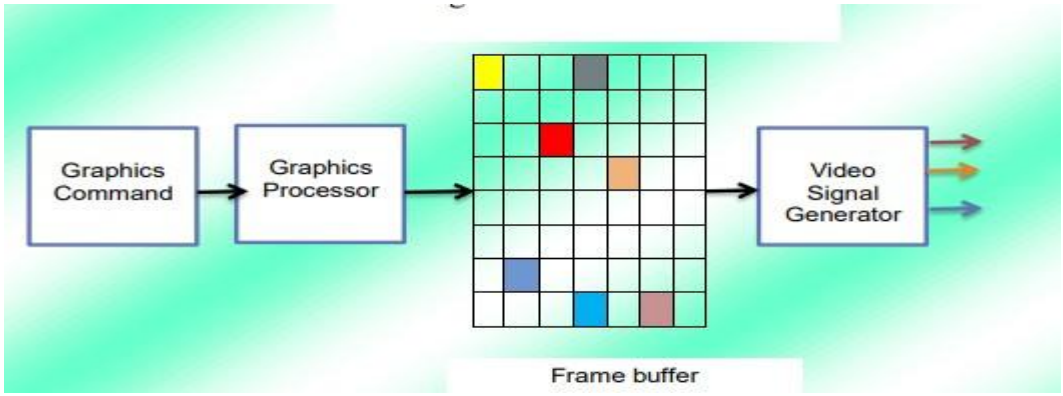
A frame buffer is a portion of memory used to store the image data that is eventually sent to the display. It holds information such as the color, depth, and sometimes transparency (alpha) of every pixel on the screen.

The frame buffer typically includes;

Buffer Type	Purpose
Color Buffer	Stores the final color of each pixel.
Depth Buffer (Z-buffer)	Stores depth info for visibility (which object is in front).
Alpha Buffer	Stores transparency info for blending effects.
Stencil Buffer	Used for masking and complex effects (like shadows).

Why Is It Important?

- It ensures that only fully rendered images are displayed.
- Helps implement effects like anti-aliasing, depth testing, blending, and post-processing.



Greyscale Frame Buffer

Arguably the simplest form of frame buffer is the greyscale frame buffer; often mistakenly called 'black and white' or 'monochrome' frame buffers. Greyscale buffers encodes pixels using various shades of grey. In common implementations, pixels are encoded as an unsigned integer using 8 bits (1 byte) and so can represent $2^8 = 256$ different shades of grey.

Pseudo-colour Frame Buffer

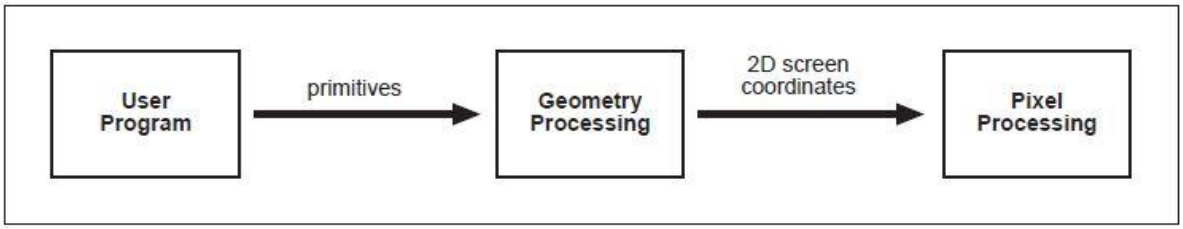
The pseudo-colour frame buffer allows representation of colour images. The storage scheme used possible value (0 − 255) to represent a particular colour; more specifically, an index into a list of 256 different colours maintained by the video hardware. The colours themselves are stored in a "Colour Lookup Table" (CLUT) which is essentially a map < colourindex, colour > i.e. a table indexed with an integer key (0−255) storing a value that represents colour.

True-Colour Frame Buffer

The true-colour frame-buffer also represents colour images, but does not use a CLUT. The RGB colour value for each pixel is stored directly within the frame buffer. So, if we use 8 bits to represent each Red, Green and Blue component, we will require 24 bits (3 bytes) of storage per pixel. With this, we can represent all 16 million colours at once in an image (given a large enough image!)

Graphics Hardware

Graphics hardware describes the hardware components necessary to quickly render 3D objects as pixels on your computer's screen using specialized rasterization based hardware architectures. The use of this term is meant to elicit a sense of the physical components necessary for performing these computations. In other words, we're talking about the chipsets, transistors, buses, and processors found on many current video cards.



The important things to note about the pipeline are as follows:

1. **The user program**: or application, supplies the data to the graphics hardware in the form of *primitives*, such as points, lines, or polygons describing the 3D geometry. Images or bitmaps are also supplied for use in texturing surfaces.
2. **Geometric primitives:** are processed on a per-vertex basis and are transformed from 3D coordinates to 2D screen triangles. On typical graphics hardware, the primitive types are limited to one or more of the following:
   points—single vertices used to represent points or particle systems;
   lines—pairs of vertices used to represent lines, silhouettes, or edge highlighting;
   polygons—triangles, triangle strips, indexed triangles, indexed triangle strips, quadrilaterals, general convex polygons, etc., used for describing triangle meshes, geometric surfaces, and other solid objects, such as spheres, cones, cubes, or cylinders.
3. **Pixel Processing:** Screen objects are passed to the pixel processors, rasterized, and then colored on a per-pixel basis before being output to the frame buffer, and eventually to the monitor.

A **pixel** (short for picture element) is the smallest unit of a digital image or display. It represents a single point in a 2D image and holds color information. Pixels are the building blocks of raster images.

A **raster** is a grid of pixels (picture elements) that collectively form a 2D image. Each pixel in the grid holds color, brightness, or transparency information. Raster graphics are also called bitmap images, and they are the most common image format used in screens, cameras, printers, and digital art.

Facsimile (Fax) and its Problems

Facsimile, commonly known as fax, is a technology that allows the transmission of scanned documents, images, or texts over telephone lines. The sender's fax machine scans the document and converts it into a bitmap image, which is then transmitted to the recipient's fax machine, where it is reproduced as a printed document or displayed on a screen.

Some of the problems associated with fax technology include:

i. ***Image Quality:*** Fax machines typically use low-resolution images, resulting in reduced image quality and legibility, especially for complex graphics or fine text.
ii. ***Transmission Errors:*** Fax transmission can be prone to errors, leading to distorted or incomplete received documents.
iii. ***Slow Transmission Speed:*** Faxing can be time-consuming, especially when dealing with large documents or images.
iv. ***Compatibility:*** Different fax machines may use various protocols, which can lead to compatibility issues between different devices.
v. ***Security Concerns:*** Fax transmissions are not as secure as modern digital communication methods, making them vulnerable to interception or unauthorized access.

Data Reduction for Graphical Input

Data reduction techniques are used to reduce the amount of data required to represent graphical objects accurately, especially when dealing with large and complex scenes. These techniques are crucial for optimizing graphics rendering and transmission processes, as well as for conserving memory and storage resources.

Various data reduction techniques can be applied to graphical input, including:

i. ***Level of Detail (LOD) techniques***: These methods involve creating multiple versions of an object at different levels of complexity (e.g., low, medium, high detail). The appropriate level of detail is chosen based on factors such as distance from the viewer, screen resolution, and available computing resources.
ii. ***Simplification***: This approach involves reducing the number of vertices, edges, or polygons in a 3D model to create a simplified version that approximates the original object's shape.
iii. ***Compression***: Data compression techniques can be used to reduce the size of graphical data files without significant loss of visual quality. Various compression algorithms, such as JPEG for images or video compression standards like H.264, can be applied to graphical data.
iv. ***Culling***: In graphics rendering, culling techniques are used to eliminate objects or parts of objects that are not visible to the camera's viewpoint, thus reducing the amount of data that needs to be processed and displayed.
v. ***LOD-based Texturing***: Textures can also be applied with varying levels of detail based on the object's distance from the viewer, helping to reduce memory and processing requirements.

vi. ***Vectorization***: Converting raster graphics (pixel-based images) into vector graphics (mathematically defined shapes) can reduce file sizes and allow for scalable graphics.

Graphics Input Aids

Graphic input aids are tools or devices that help users interact with a graphical system, especially in creating, selecting, or modifying graphical data. They serve as interfaces between the user and the computer graphics system, making graphical operations more efficient and intuitive.These devices include both hardware level and software level aids such as;

1. Keyboard
2. Mouse
3. Trackball
4. Spaceball
5. Joystick
6. Light Pen
7. Digitizer
8. Touch Panels
9. Voice Recognition
10. Image Scanner
11. AR/VR Glasses

Plotters

Plotters are a special type of output device. It is suitable for applications:

1. Architectural plan of the building.
2. CAD applications like the design of mechanical components of aircraft.
3. Many engineering applications.

Plotter

Advantage:

1. It can produce high-quality output on large sheets.
2. It is used to provide the high precision drawing.
3. It can produce graphics of various sizes.
4. The speed of producing output is high.

Drum Plotter:

It consists of a drum. Paper on which design is made is kept on the drum. The drum can rotate in both directions. Plotters comprised of one or more pen and penholders. The holders are mounted perpendicular to drum surface. The pens are kept in the holder, which can move left to the right as well as right to the left. The graph plotting program controls the movement of pen and drum.

Drum Plotter

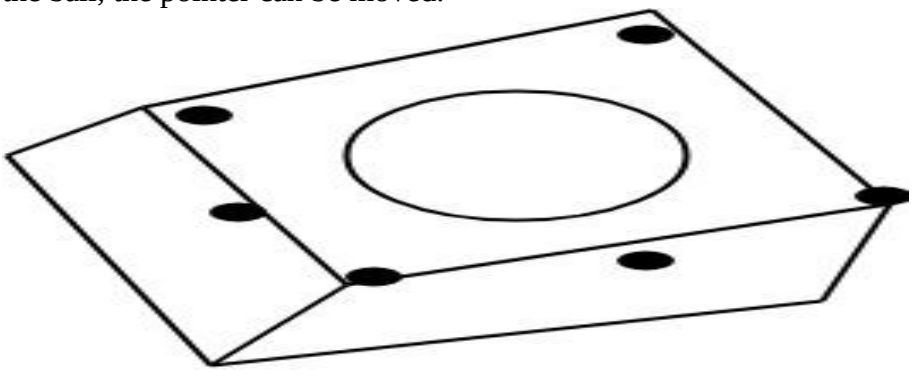
Flatbed Plotter:
It is used to draw complex design and graphs, charts. The Flatbed plotter can be kept over the table. The plotter consists of pen and holder. The pen can draw characters of various sizes. There can be one or more pens and pen holding mechanism. Each pen has ink of different color. Different colors help to produce multicolor design of document. The area of plotting is also variable. It can vary A4 to 21'*52'.

Flatbed Plotter

It is used to draw

1. Cars
2. Ships
3. Airplanes
4. Shoe and dress designing
5. Road and highway design

Trackball
It is a pointing device. It is similar to a mouse. This is mainly used in notebook or laptop computer, instead of a mouse. This is a ball which is half inserted, and by changing fingers on the ball, the pointer can be moved.



TrackBall

Applications:

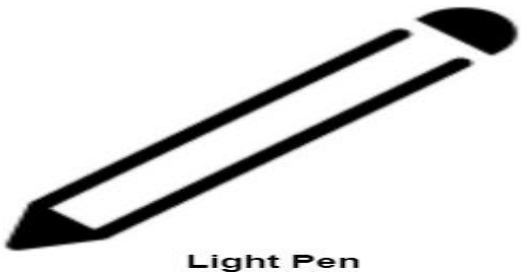
1. It is used for three-dimensional positioning of the object.
2. It is used to select various functions in the field of virtual reality.
3. It is applicable in CAD applications.
4. Animation is also done using spaceball.
5. It is used in the area of simulation and modeling.

Joystick:

A Joystick is also a pointing device which is used to change cursor position on a monitor screen. Joystick is a stick having a spherical ball as its both lower and upper ends as shown in fig. The lower spherical ball moves in a socket. The joystick can be changed in all four directions. The function of a joystick is similar to that of the mouse. It is mainly used in Computer Aided Designing (CAD) and playing computer games.

Light Pen

Light Pen (similar to the pen) is a pointing device which is used to select a displayed menu item or draw pictures on the monitor screen. It consists of a photocell and an optical system placed in a small tube. When its tip is moved over the monitor screen, and pen button is pressed, its photocell sensing element detects the screen location and sends the corresponding signals to the CPU.



Uses:

1. Light Pens can be used as input coordinate positions by providing necessary arrangements.
2. If background color or intensity, a light pen can be used as a locator.
3. It is used as a standard pick device with many graphics system.
4. It can be used as stroke input devices.
5. It can be used as valuators

Digitizers:

The digitizer is an operator input device, which contains a large, smooth board (the appearance is similar to the mechanical drawing board) & an electronic tracking device, which can be changed over the surface to follow existing lines. The electronic tracking device contains a switch for the user to record the desire x & y coordinate positions. The coordinates can be entered into the computer memory or stored or an off-line storage medium such as magnetic tape.

Digitizer

Advantages:

1. Drawing can easily be changed.
2. It provides the capability of interactive graphics.

Disadvantages:

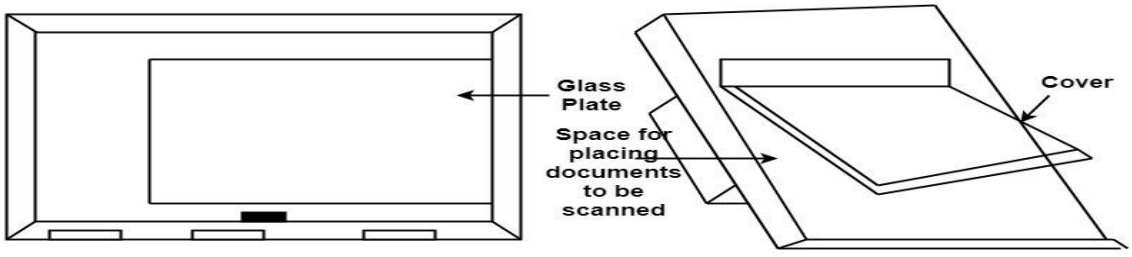
1. Costly
2. Suitable only for applications which required high-resolution graphics.

Scanner

It is an input device. The data or text is written on paper. The paper is feeded to scanner. The paper written information is converted into electronic format; this format is stored in the computer. The input documents can contain text, handwritten material, picture extra. By storing the document in a computer document became safe for longer period of time. The document will be permanently stored for the future.

Types of image Scanner:

- **Flat Bed Scanner:** It resembles a photocopy machine. It has a glass top on its top. Glass top in further covered using a lid. The document to be scanned is kept on glass plate. The light is passed underneath side of glass plate. The light is moved left to right. The scanning is done the line by line. The process is repeated until the complete line is scanned. Within 20-25 seconds a document of 4" * 6" can be scanned.



Flat Bed Scanner

- **Hand Held Scanner:** It has a number of LED's (Light Emitting Diodes) the LED's are arranged in the small case. It is called a Hand held Scanner because it can be kept in hand which performs scanning. For scanning the scanner is moved over document from the top towards the bottom. Its light is on, while we move it on document. It is dragged very slowly over document. If dragging of the scanner over the document is not proper, the conversion will not correct.

AR/VR Glasses

Augmented Reality (AR) Glasses

Overlay digital elements (text, images, 3D models) onto the real world**.** Examples include Microsoft HoloLens, Magic Leap, Google Glass Enterprise Edition

Functionality:

- Transparent lenses or see-through displays.
- Real-time environmental sensing (via cameras or sensors).

- Interaction with virtual objects anchored in real space.

Common Uses:

- Navigation, translation, and instructions.
- Industry maintenance and repair.
- Medical visualization.

Virtual Reality (VR) Glasses

Fully immerse the user in a digital environment. Examples are Meta Quest series, HTC Vive, PlayStation VR, Valve Index.

Functionality:

- Block out the real world.
- Display stereoscopic 3D visuals.
- Track head movements (sometimes hands and body).

Common Uses:

- Gaming (e.g., Meta Quest, PlayStation VR).
- Virtual tours and simulations.
- Education and training (e.g., flight simulators).

Software Level Aids

These are software-level aids used during input operations in graphic applications.

Technique	Description	Example
Rubber Banding	Dynamically shows a line or shape being drawn, stretching like a rubber band.	Drawing lines or rectangles interactively.
Dragging	Moving objects by clicking and holding them.	Moving shapes, repositioning text.
Prompting	System guides the user with messages.	"Click two points to draw a line."
Menus and Toolbars	Offer graphical options for tools and commands.	File menu, drawing tools.
Icons	Small graphic symbols representing commands or objects.	Save icon, zoom icon.
Coordinate Display	Shows real-time cursor coordinates.	Helpful in CAD software.
Grid and Snap Tools	Provide alignment aids.	Snapping a point to a grid line.

Interaction Modes in Computer Graphics

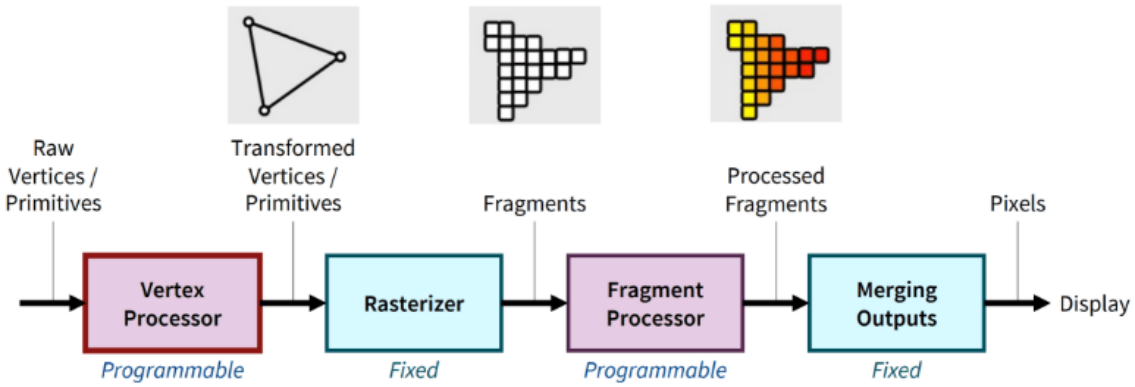
Mode	Description
Positioning	Identifying a location on the screen.
Selection	Choosing a displayed object.
Menu Picking	Selecting commands via GUI.
String Input	Typing text (e.g., labels, names).
Sketching	Freehand drawing using a stylus or touch.

The Concept of Rendering

Rendering is the conversion of a 3D model or scene into an image using computer software. It involves simulating light, color, texture, and geometry to produce realistic or stylized visual outputs. Rendering is a key step in fields like video games, animation, simulations, virtual reality, and visual effects.

The Rendering Pipeline

Rendering pipeline is the step-by-step process that a computer graphics system uses to convert a 3D scene into a 2D image (what you finally see on screen).



1. **Vertex Processing**
   The vertex shader processes and transforms individual vertices to map the 3D vertex locations to 2D screen positions.
   Key steps:
   Vertex Shader: Applies transformations to each vertex (Model → World → View → Clip space).
   Projection: Converts 3D scene to 2D screen using perspective or orthographic projection.
   Clipping: Removes parts of geometry outside the camera view.
   Culling: Discards back-facing polygons to improve performance.

2. **Rasterization**
   The rasterizer converts the primitive (connected vertices), a continuous-geometry representation, into a set of fragments, the discrete geometry of the pixelized display, one by one. Therfore, a fragment is defined for a pixel location covered by the primitive on the screen and thus can be treated as a pixel in 3D spaces which is aligned with the pixel grid. Also, it possesses the attributes such as position, color, normal and texture to determine the rendering output of the pixel.

3. **Fragment Processing**
   The fragment shader processes individual fragments and determines its color through various operations such as texturing and lighting.
4. **Output Merging**
   The output merger combines the fragments of all primitives (in 3D space) into 2D color-pixel for the display. Not programmable.

Example: Rendering a Cube in a Game Engine

1. You place a cube in a 3D scene with a light and a camera.
2. The engine transforms the cube's vertices from model space to screen space.
3. The GPU rasterizes the cube's triangles.
4. Each pixel on the cube is shaded based on light and material.
5. The image is displayed on your screen.

Types of Rendering

Ray Tracing: Simulates the path of light rays bouncing in a scene, and calculates reflections, refractions, and shadows accurately.The raysof light are traced from the camera through pixels into the scene. It is used in Film, design, modern game engines.

Pros:

- Physically accurate lighting
- Realistic reflections and shadows

Cons:

- Slower than rasterization
- Computationally expensive

Rasterization: Projects 3D geometry into 2D screen space by filling in pixels (fragments) based on texture and shading. It is much faster than ray tracing but less realistic. It is commonly used in games, simulations

Pros:

- Extremely fast
- Well-optimized for GPUs

Cons:

- Limited realism
- Cannot handle complex light interactions well

Radiocity: The processing of one horizontal line (scanline) of the image at a time. It is efficient for scenes with simple lighting and shading. It is used in older games and rendering engines.

Pros:

- Fast and efficient
- Lower computational requirements

Cons:

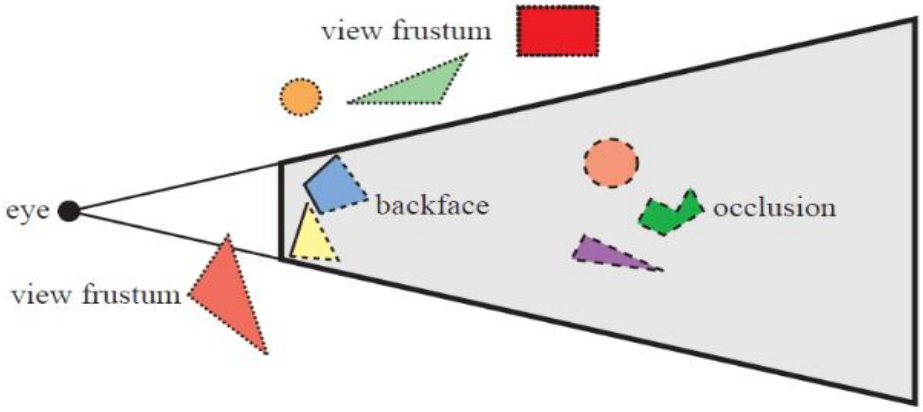
- Less realistic than ray tracing or path tracing

Handling Visibility Problem

To render the scene captured by a camera, it’s crucial to identify which primitives and parts of primitives are not visible from our viewpoint. This consideration is known as visibility problem, which stands as one of the initial significant hurdles in rendering. There are several factors that render the primitives invisible:

- Primitives outside of the viewing frustum ⟹ **clipping & view volume culling**
- Back-facing primitives ⟹ **back-face culling**

- Primitives occluded by other objects closer to the camera ⟹ **occlusion culling (hidden-surface removal)**



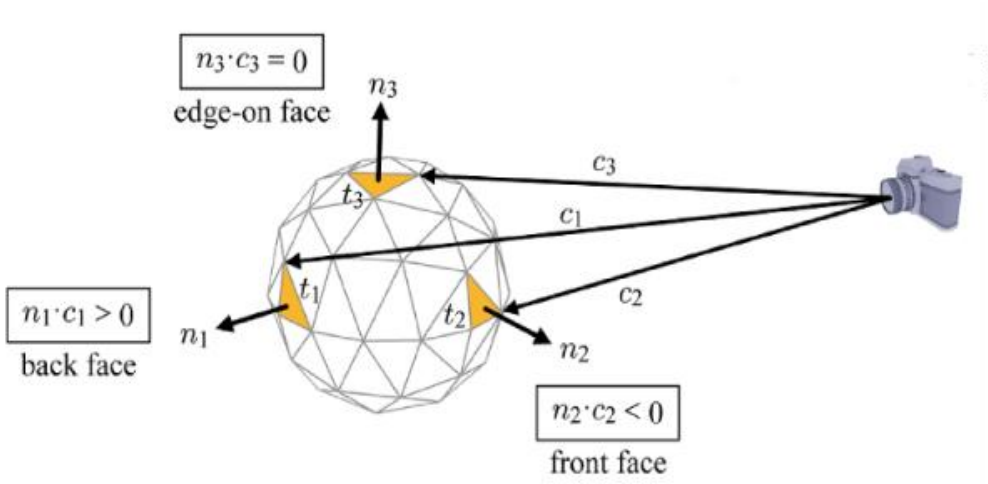
Clipping & View Volume Culling
Clipping is the process of cutting off parts of objects (like lines or polygons) that are outside the camera's view (view volume or frustum). It ensures only the visible portions of geometry are passed on for rasterization.

How it works:
- The camera defines a viewing volume (also called a view frustum in 3D).
- Any part of a primitive (e.g., triangle, line) outside this frustum is discarded or clipped.

View volume culling is the process of eliminating entire objects that are completely outside the camera's view frustum before they reach later stages of the pipeline. The purpose is to improve performance by not processing or rendering invisible objects and reduce the workload on the GPU.

How it works:
- Each object's bounding volume (like a bounding box or sphere) is tested against the view frustum.
- If the object is entirely outside, it's culled (not rendered).
- If it intersects or is fully inside, it's passed on for further processing.

Back-Face Culling
Polygons that face away from the camera (termed the back faces) are usually guaranteed to be overdrawn by polygons facing the camera (termed the front faces). Consequently, these back polygons can be culled, a process known as backface culling. The purpose is to reduce rendering time and eliminate unseen geometry early in the pipeline.



How It Works:
Each triangle (or polygon) has a **front** and **back** face determined by the **winding order** of its vertices:

- Clockwise (CW)
- Counter-clockwise (CCW)
- If a triangle's vertices appear in **counter-clockwise** order from the camera's view, it's a **front face**.
- If the vertices appear **clockwise**, it's a **back face**.

If the **surface normal** (perpendicular vector) of a triangle **points away** from the camera, it's considered a **back face** and is culled (not rendered).

Occlusion Culling
Occlusion culling is a rendering optimization technique used in computer graphics to avoid drawing objects that are not visible to the camera because they are blocked (occluded) by other objects. There are many techniques for occlusion culling;

- Depth Test (Z-buffering)
- Painter's algorithm
- Ray tracing
- Warnock algorithm

How It Works:

- The graphics engine checks each object's position relative to the camera and other objects.
- Determines if an object is completely hidden behind another.
- If fully occluded, the object is skipped (culled) and not sent to the rendering pipeline.

Examples in Practice:

- In a **first-person shooter**, walls may block enemies on the other side. Those enemies aren't drawn until the player moves past the wall.
- In **open-world games** like GTA or Skyrim, buildings may block whole city blocks—occlusion culling prevents unnecessary rendering of hidden geometry.

Painter's Algorithm
The Painter's Algorithm is a visibility determination technique in 3D rendering where surfaces are drawn from back to front, like a painter painting the background before the foreground. Objects that are farthest from the viewer are rendered first, and closer objects are

drawn on top. The core idea just like a painter is: “Draw distant things first, then layer closer objects on top."

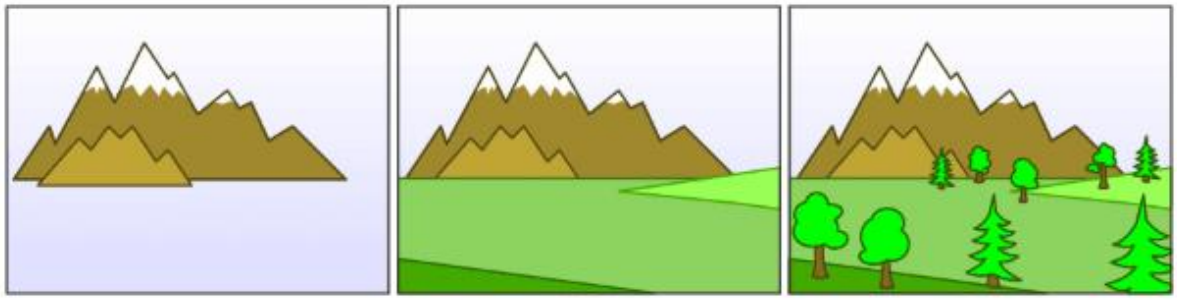
Steps in the Painter’s Algorithm:

1. Sort all polygons/objects in the scene by their depth (z-value)—from farthest to nearest.
2. Draw them one by one in sorted order.
    - Closer objects overwrite (cover) the farther ones.
3. The final image appears correctly—at least most of the time.

Example:

Imagine trees in front of a mountain with a green landscape:

1. Mountain has larger z-depth (farther).
2. Then the green landscape (also farther)
3. Trees has smaller z-depth (closer).
4. Mountain is painted first, then tree on top of it.



Limitations of Painter’s Algorithm

1. Polygons that intersect (e.g., one goes behind and in front of another) can't be drawn in a strict back-to-front order.
2. Sorting large numbers of polygons every frame is computationally costly.
3. May not handle transparency correctly unless specially managed.

Warnock's Algorithm

Warnock's Algorithm is a visibility determination technique used in 3D graphics to decide which surfaces are visible in a scene when projected onto a 2D display. It handles overlapping objects by recursively subdividing the image into smaller regions (usually rectangles) until each region is simple enough to resolve visibility.

This is a divide and conquer algorithm with run-time of ***O(np)***, where *n* is the number of polygons and *p* is the number of pixels in the viewport.

Key Idea

If a region of the image is too complex to resolve visibility (e.g., multiple surfaces overlap), **subdivide the region** into smaller parts and try again. Repeat this until:

- The region is trivial (e.g., contains one polygon),
- Or the region is **one pixel** (where depth comparison can resolve visibility).

Steps of Warnock’s Algorithm

1. **Check for trivial cases**:
    - Region is empty → background color.
    - Region contains only one surface → render that surface.

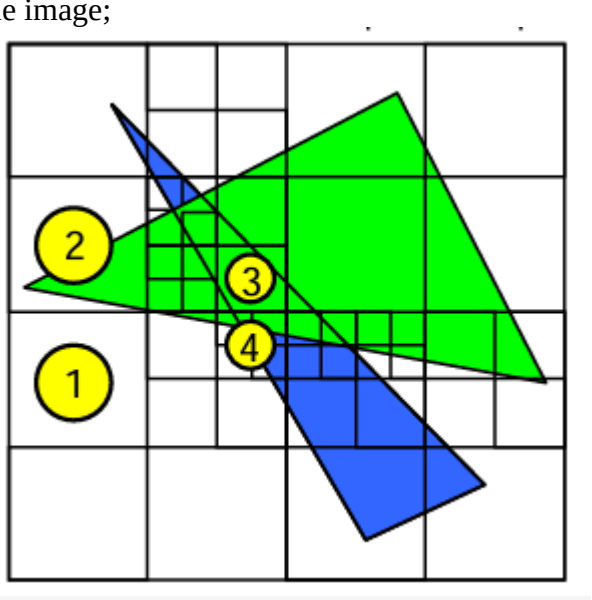
- All surfaces are behind another → render the top surface.
- Region is so small (e.g., 1×1 pixel) → use **depth comparison**.

2. **If not trivial**:

Subdivide the region (usually into 4 quadrants).

Recursively apply Warnock's algorithm to each subregion.

For example, consider the image;



To render the scene below using Warnock's principle. We use Divide and conquer as follows;

1. No object: use background color (white)
2. One object: render (green)
3. More objects, one closest: render closest (blue)

Differences Between Painter's and Warnock's Algorithm

Painter's Algorithm	**Warnock's Algorithm**
Draw surfaces from back to front like a painter	Recursively subdivide the screen until simple visibility is resolved
Depth-sorting (uses depth of objects)	Divide-and-conquer (spatial subdivision)
May fail with complex overlapping or intersecting surfaces	Can handle overlaps by subdividing until simple cases
Simple but inefficient for complex scenes	More efficient for complex scenes with multiple objects
Difficulties with cyclic overlap, intersecting surfaces	Can be slow due to excessive subdivision
Back-to-front painting	Region-based, no specific order of object rendering
Good for simple, non-intersecting objects	Better for detailed scenes with complex visibility issues

Handwriting Recognition

In *handwriting recognition* (HWR) the device interprets the user's handwritten characters or words into a format that the computer understands (e.g., Unicode text). The input device typically comprises a stylus and a touch-sensitive screen. There are many levels of HWR, starting from the recognition of simplified individual characters to the recognition of whole words and sentences of cursive handwriting.

Optical Character Recognition (**OCR**) is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example: from a television broadcast).

Widely used as a form of data entry from printed paper data records – whether passport documents, invoices, bank statements, computerized receipts, business cards, mail, printouts of static-data, or any suitable documentation

Uses of Character Recognition

- Data entry for business documents, e.g. Cheque, passport, invoice, bank statement and receipt
- Automatic number plate recognition
- In airports, for passport recognition and information extraction
- Traffic sign recognition
- Extracting business card information into a contact list
- More quickly make textual versions of printed documents, e.g. book scanning
- Make electronic images of printed documents searchable, e.g. Google Books
- Converting handwriting in real-time to control a computer (pen computing)
- Defeating CAPTCHA anti-bot systems, these are specifically designed to prevent OCR The purpose can also be to test the robustness of CAPTCHA anti-bot systems.
- Assistive technology for blind and visually impaired users

Techniques of Handwriting/Character Recognition

De-skew – If the document was not aligned properly when scanned, it may need to be tilted a few degrees clockwise or counter clockwise in order to make lines of text perfectly horizontal or vertical.

Despeckle – remove positive and negative spots, smoothing edges

Binarisation – Convert an image from color or greyscale to black-and-white (called a "binary image" because there are two colors). The task of binarisation is performed as a simple way of separating the text (or any other desired image component) from the background.

Line removal – Cleans up non-glyph boxes and lines

Layout analysis or "zoning" – Identifies columns, paragraphs, captions, etc. as distinct blocks. Especially important in multi-column layouts and tables.

Line and word detection – Establishes baseline for word and character shapes, separates words if necessary.

Script recognition – In multilingual documents, the script may change at the level of the words and hence, identification of the script is necessary, before the right OCR can be invoked to handle the specific script.

Character isolation or "segmentation" – For per-character OCR, multiple characters that are connected due to image artifacts must be separated; single characters that are broken into multiple pieces due to artifacts must be connected.

Introduction to 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.

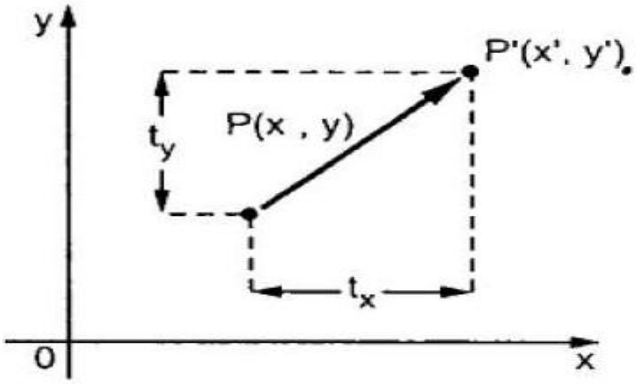
Transformations play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation.

Types of Transformation

1. Translation
2. Scaling
3. Shearing
4. Rotation
5. Reflection

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 −

X = X + t_x

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.

Translation is done using translation vectors. There are three vectors in 3D instead of two. These vectors are in x, y, and z directions. Translation in the x-direction is represented using T_x. The translation is y-direction is represented using T_y. The translation in the z- direction is represented using T_z.

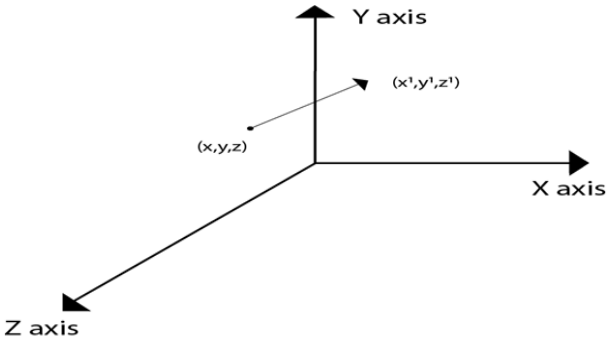
If P is a point having co-ordinates in three directions (x, y, z) is translated, then after translation its coordinates will be (x^1 y^1 z^1) after translation. T_x T_y T_z are translation vectors in x, y, and z directions respectively.

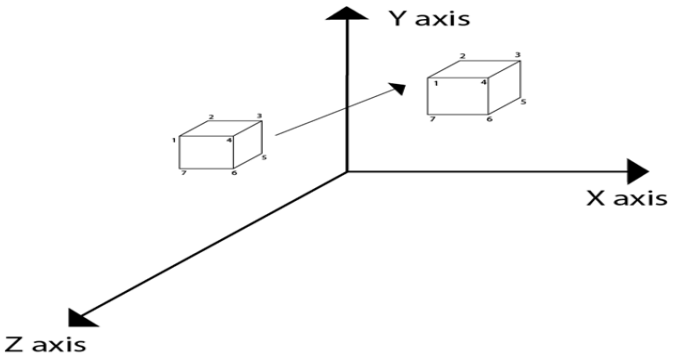
$x^1 = x + T_x$

$y^1 = y + T_y$

$z^1 = z + T_z$

Three-dimensional transformations are performed by transforming each vertex of the object. If an object has five corners, then the translation will be accomplished by translating all five points to new locations. Following figure 1 shows the translation of point figure 2 shows the translation of the cube.





Matrix for translation

$$\begin{Bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{Bmatrix} \text{ or } \begin{Bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

Matrix representation of point translation

Point shown in fig is (x, y, z). It become (x^1, y^1, z^1) after translation. T_x T_y T_z are translation vector.

$$\begin{pmatrix} x^1 \\ y^1 \\ z^1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Example: A point has coordinates in the x, y, z direction i.e., (5, 6, 7). The translation is done in the x-direction by 3 coordinate and y direction. Three coordinates and in the z- direction by two coordinates. Shift the object. Find coordinates of the new position.

Solution: Co-ordinate of the point are (5, 6, 7)

Translation vector in x direction = 3

Translation vector in y direction = 3

Translation vector in z direction = 2

Translation matrix is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{pmatrix}$$

Multiply co-ordinates of point with translation matrix

$$(x^1y^1z^1) = (5,6,7,1) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 3 & 3 & 2 & 1 \end{pmatrix}$$

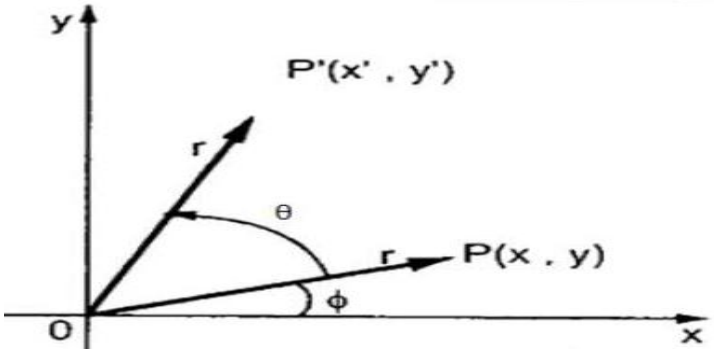
= [5+0+0+3, 0+6+0+3, 0+0+7+2, 0+0+0+1] = [8 9 9 1]

x becomes x^1=8
y becomes y^1=9
z becomes z^1=9

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), we can find the rotation matrix R to be;

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

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.

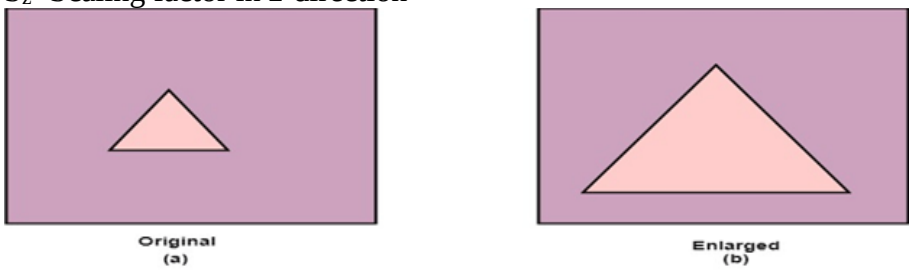
Let us assume that the original coordinates are (X, Y), the scaling factors are (S_X, S_Y), and the produced coordinates are (X, Y). This can be mathematically represented as shown below −

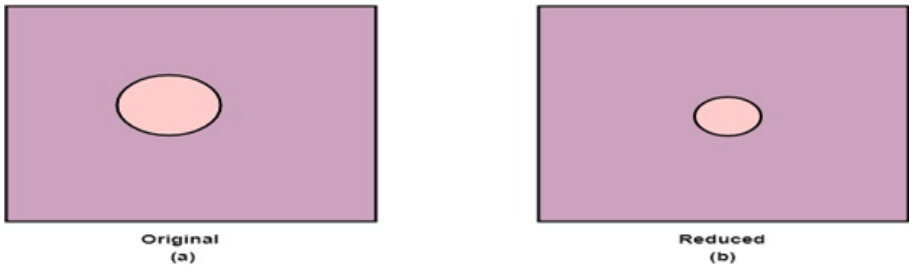
X' = X . S_X and
Y' = Y . S_Y

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 for 2D scaling as follows;

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

For 3D objects the scaling three factors are required S_x S_y and S_z.
S_x=Scaling factor in x- direction
S_y=Scaling factor in y-direction
S_z=Scaling factor in z-direction



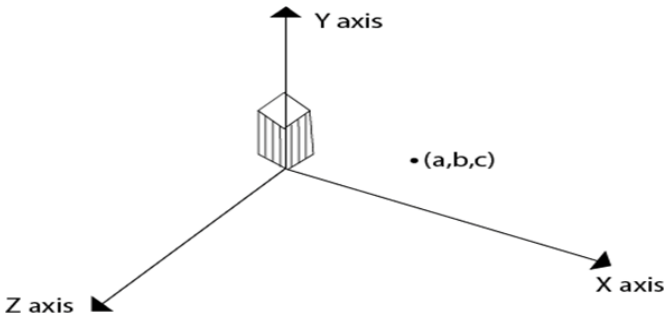


Matrix for 3D Scaling

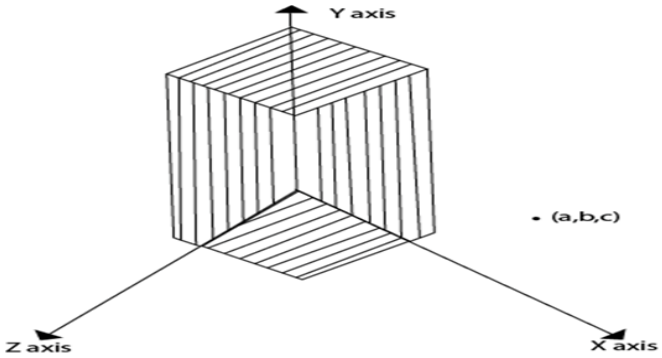
$$\begin{Bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

Scaling of the object relative to a fixed point
Following are steps performed when scaling of objects with fixed point (a, b, c). It can be represented as below:

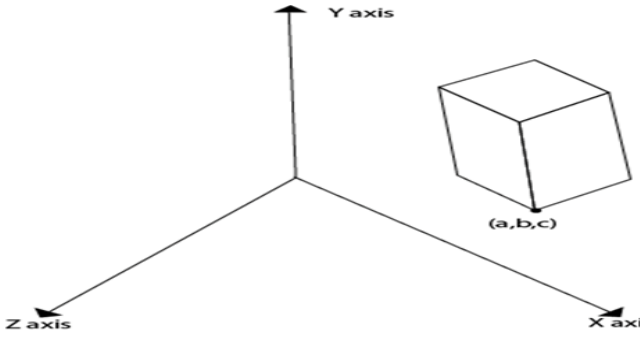
1. Translate fixed point to the origin
2. Scale the object relative to the origin
3. Translate object back to its original position.
4. n figure (a) point (a, b, c) is shown, and object whose scaling is to done also shown in steps in fig (b), fig (c) and fig (d).



Object transfer at origin
(b)



Scaling of object
(c)

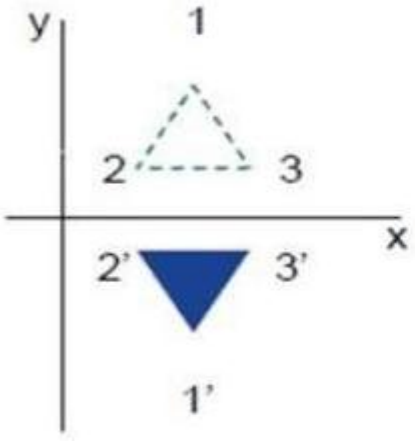


Object shifted to original position after scaling
(d)

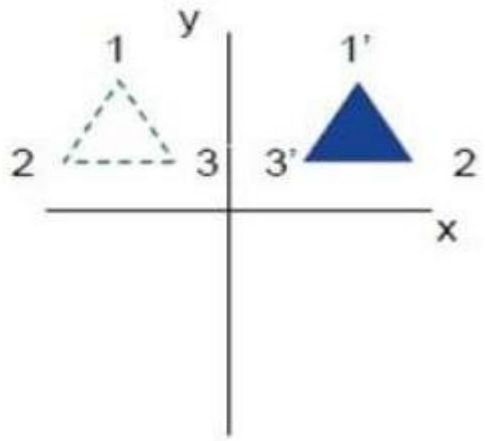
Reflection

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. **Reflection** produces a mirror image of a shape or object across a specific axis or line.

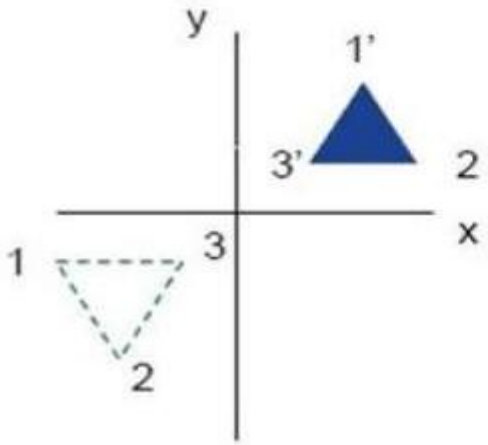
The following figures show reflections with respect to X and Y axes, and about the origin respectively.



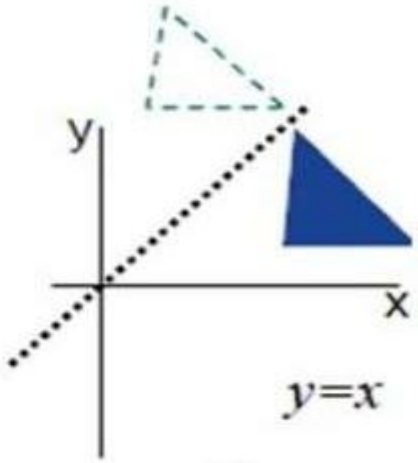
(a)



(b)



(c)



(d)

Common Reflection Types in 2D

1. Reflection about X-axis

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ -y \end{bmatrix}$$

2. Reflection about Y-axis

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -x \\ y \end{bmatrix}$$

3. Reflection about origin (0, 0)

$$x' = -x, \quad y' = -y$$

4. Reflection about the line y = x

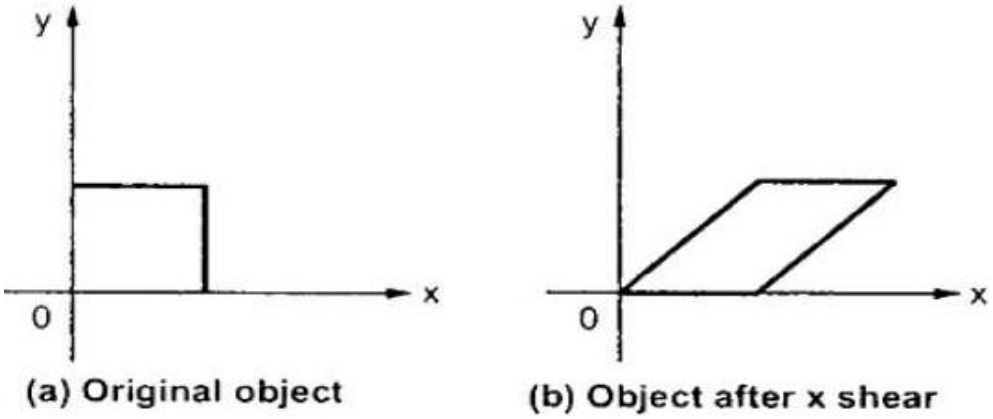
$$x' = y, \quad y' = x$$

Shearing

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.



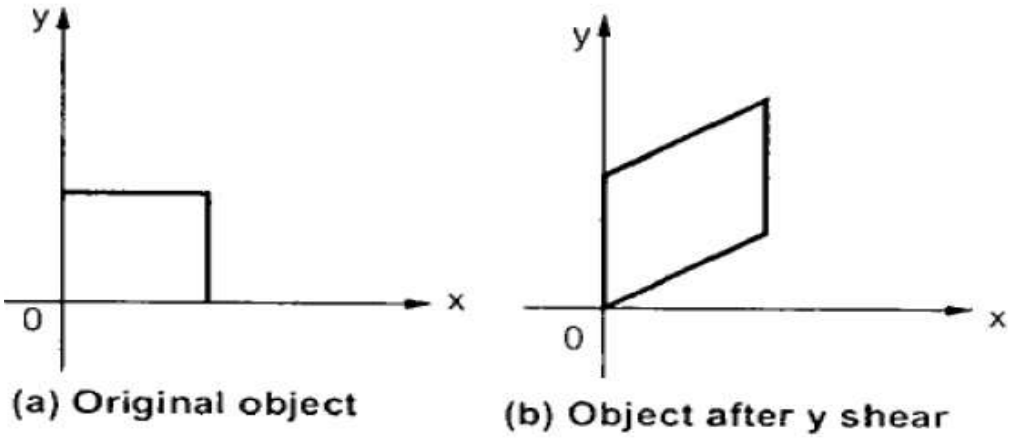
For X-direction shear;

The new position after shear will be defined as;

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x + sh_x \cdot y \\ y \end{bmatrix}$$

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.



For Y-direction shear;

The new position after shear will be defined as;

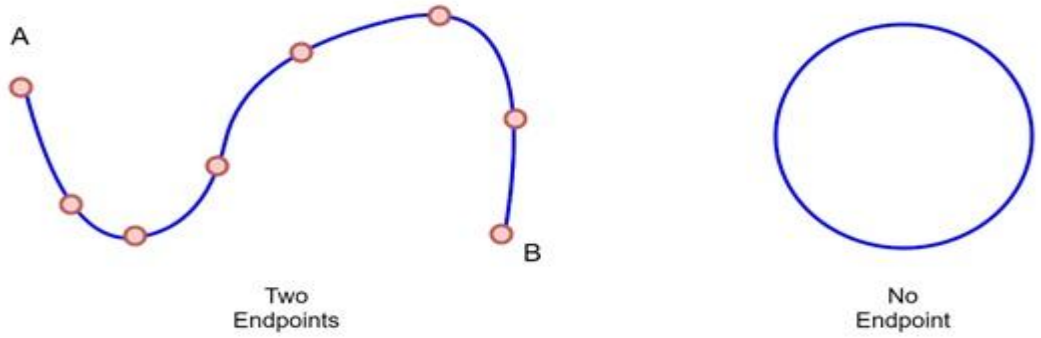
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y + sh_y \cdot x \end{bmatrix}$$

Intoduction to Curve Synthesis

A curve consists of an infinite set of points that are connected in a continuous manner. Each point on the curve typically has two neighbours, except for the endpoints. These endpoints, if they exist, have only one neighbour. They are widely used in animation, modeling, and

simulations. Curves are fundamental tools in graphics and visualization. They combine mathematical power with artistic flexibility, enabling developers and designers to build realistic, efficient, and visually appealing digital content.

For example, in a circle, which is a closed curve, there are no endpoints because the curve forms a continuous loop. On the other hand, in an open curve like a line, the endpoints are the two points at the beginning and end of the curve.



Types of Curves

Curves can be classified into three categories;

1. Explicit curves
2. Implicit curves
3. Parametric curves.
    - Bézier Curves
    - B-spline 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;

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

A common example is the circle, whose implicit representation is

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

Explicit Curves

A mathematical function y = fxx 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 −

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

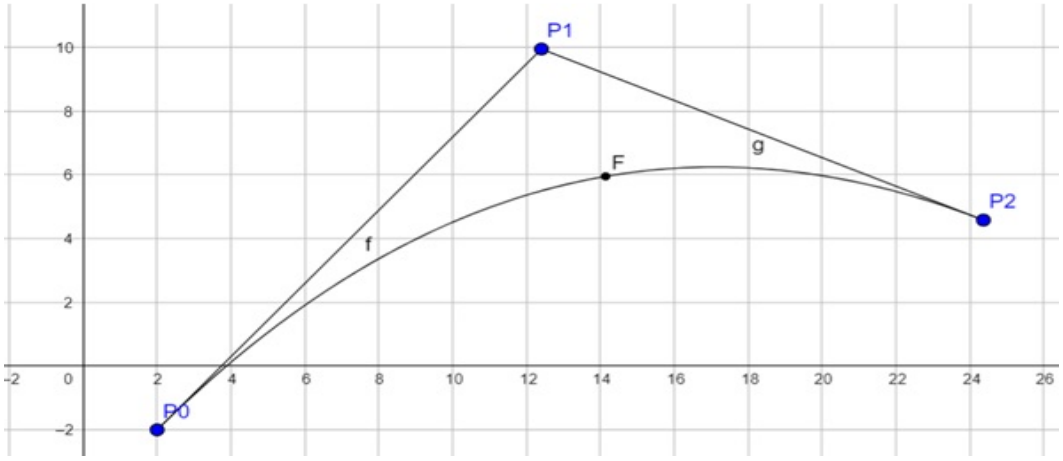
The functions f and g become the x,y coordinates of any point on the curve, and the points are obtained when the parameter t is varied over a certain interval [a, b], normally [0, 1].

Bézier Curves

Bezier curves are used extensively in computer graphics, especially in vector graphics and animations. These curves are defined using control points, which determine the shape of the curve. A quadratic Bezier curve is defined as:

$$B(t) = (1 - t)^2 P_0 + 2(1 - t)tP_1 + t^2 P_2$$

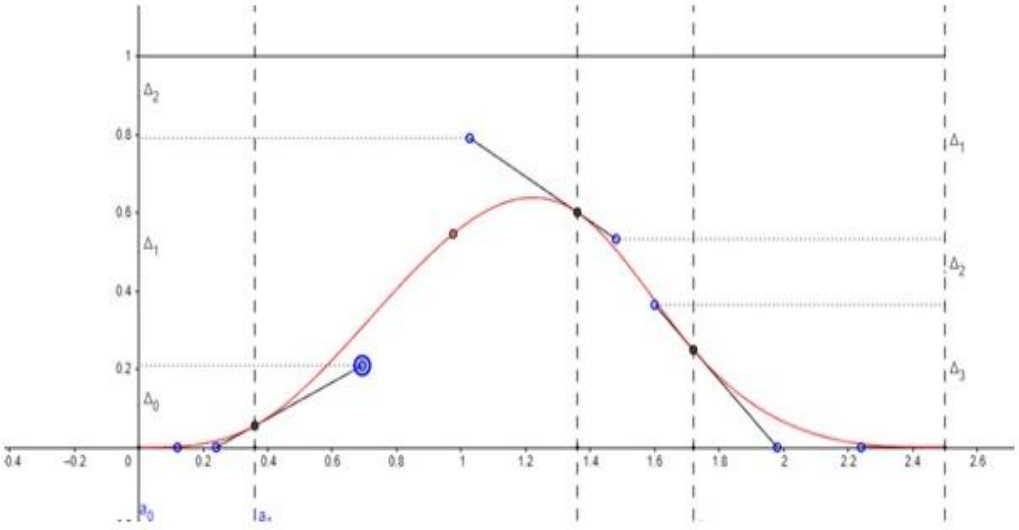
where P_0, P_1, and P_2 are control points, and **t** is a parameter that varies between 0 and 1. Bezier curves allow designers to create smooth, flowing shapes.



B-spline Curves

B-splines offer better control over the shape of the curve by allowing local adjustments to the curve without affecting the entire shape. This makes them particularly useful in 3D modelling and computer-aided design (CAD) applications.

Unlike Bézier curves, where changing one control point affects the entire curve, B-spline curves allow for more localized changes. If a control point is moved in a B-spline curve, only a specific segment of the curve changes, leaving the rest of the curve intact.



Importance of Curves in Graphics and Visualization

1. Genaration of smooth and natural shapes.
2. Precision in modelling.
3. Scalability and resolution independence.
4. Representation of animation and motion paths.
5. User interface design.
6. Data visualization.

Introduction to Contouring Ring Structures

Contouring is a fundamental technique used in computer graphics, geographic information systems (GIS), and other fields to represent continuous data as discrete lines or curves of constant values. Contour lines are useful for visualizing 3D surfaces on 2D planes, as well as for various analysis tasks. Several techniques are commonly used for contouring:

i. ***Marching Squares/Rectangles***: Marching squares (2D) or marching cubes (3D) is a widely used technique for contouring raster data. It divides the input data into a grid of cells and analyzes the values at the corners of each cell. Based on the relative values, it constructs contour lines by interpolating and connecting the edges of the cells.
ii. ***Contour Tree***: The contour tree is a hierarchical representation of the contour lines in a data set. It is constructed by identifying critical points (peaks, saddles, and valleys) in the data and connecting them to form a tree-like structure. Contour trees are useful for analyzing topological features in the data.
iii. ***Triangulated Irregular Network (TIN)***: A TIN is a data structure that represents a set of irregularly spaced data points as a triangulated surface. Contouring with a TIN involves interpolating the data values at the vertices of the triangles and connecting the contour lines along the triangle edges.
iv. ***Delaunay Triangulation***: Delaunay triangulation is a technique to triangulate a set of points such that no point is inside the circumcircle of any triangle. It is commonly used for contouring scattered data points. Similar to TIN, contour lines are constructed by interpolating along the edges of the triangles.
v. ***Level Set Methods***: Level set methods represent a shape or contour as the zero level set of a higher-dimensional function. The contour lines are then extracted as the points where this higher-dimensional function evaluates to zero. Level set methods are powerful for handling topological changes and evolving contours over time.
vi. ***Isocontouring***: Isocontouring is a general term used for contouring when data has isolines of constant value. It involves identifying points or regions in the data where the value matches a specified threshold and connecting these points to create contour lines.
vii. ***Interpolation Techniques***: Interpolation plays a crucial role in contouring, especially when dealing with scattered data points. Various interpolation methods, such as linear interpolation, bilinear interpolation, or cubic spline interpolation, can be used to estimate values at arbitrary positions between data points.
viii. ***Simplification and Smoothing***: After contour lines are generated, simplification and smoothing techniques can be applied to reduce the complexity of the contour representation and create visually pleasing outputs.

Ring Structures

Ring structure is a data structure used to represent and store contours efficiently. It is particularly useful when you want to represent closed contours, such as the outline of a shape or the border of an object.

A ring structure consists of a sequence of vertices (points) that form a closed loop. Each vertex is connected to its adjacent vertices to form the ring. This data structure allows for easy traversal of the contour, making it convenient for various algorithms like rendering or processing.

Ring structures are particularly useful when dealing with shapes or objects that have a well-defined boundary, such as the outline of a polygon or the border of an object in an image. Some of the techniques used in the design of ring structures:

i. ***Linked Lists***: One of the simplest and most common ways to design a ring structure is to use a linked list. Each node in the linked list represents a vertex or a point on the contour, and each node points to the next node in the sequence. The last node in the list points back to the first node, forming a closed loop. This design allows for easy traversal of the contour, making it convenient for various algorithms.
ii. ***Half-Edge Data Structure***: The half-edge data structure is a more sophisticated approach to representing ring structures. It is commonly used in computer graphics and mesh processing. In this data structure, each half-edge represents an edge of the contour and stores information about its two endpoints (vertices), the face it belongs to (useful for handling non-manifold contours), and pointers to the previous and next half-edges. The half-edges form a closed loop around the contour, and this representation is efficient for various contour processing operations.
iii. ***Circular Array***: A circular array is a memory-efficient approach for storing ring structures. Instead of using pointers to connect the nodes (vertices) of the contour, a fixed-size array is used, and each node contains an index to the next node in the array. The last node points back to the first node, forming a circular connection. This approach avoids the overhead of allocating memory for pointers but requires careful management of the array size.
iv. ***Dynamic Arrays or Vectors***: Dynamic arrays or vectors can also be used to represent ring structures. Each node in the ring contains information about the vertex and its index in the array. The array itself is dynamically resized as needed to accommodate additional vertices. This approach is flexible and allows for efficient dynamic modifications of the contour.
v. ***Quad-Edge Data Structure***: The quad-edge data structure is a versatile representation for handling planar subdivisions, including closed contours. It is based on the concept of subdividing each edge into four sub-edges, forming a quad-edge. Quad-edges can be traversed to represent both the interior and the exterior boundaries of a shape, making it useful for certain algorithms in computational geometry.

Concept of Shading

Shading in Computer graphics refers to the process of altering the color of a surface based on lighting conditions to create the illusion of depth, material, and lighting. It enhances realism by simulating how light interacts with surfaces. Shading is used traditionally in drawing for depicting a range of darkness by applying media more densely or with a darker shade for darker areas, and less densely or with a lighter shade for lighter areas

Shading model is used to compute the intensities and colors to display the surface. The shading model has two primary ingredients: properties of the surface and properties of the illumination falling on it. The principal surface property is its reflectance, which determines how much of the incident light is reflected. If a surface has different reflectance for the light of different wavelengths, it will appear to be colored.

Importance of Shading

1. Creates a sense of depth and 3D structure.
2. Differentiates materials (metal, glass, wood, etc.)
3. Simulates lighting conditions (sunlight, ambient, spotlight)
4. Helps viewers perceive the shape and position of objects.

Shading techniques

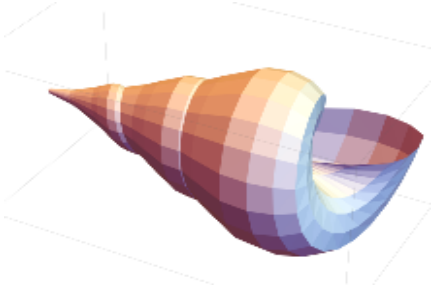
During shading a surface normal is often needed for lighting computation. The normals can be precomputed and stored for each vertex of the model.

Flat shading

Uses a single color for an entire polygon. Each polygon is shaded using one normal (usually the face normal). It is Fast, but unrealistic and create poor visual quality on curved surfaces. For instance, Flat shading a textured cuboid. Here, the lighting is evaluated only once for each polygon (usually for the first vertex in the polygon, but sometimes for the centroid for triangle meshes), based on the polygon's surface normal and on the assumption that all polygons are flat. The computed color is used for the whole polygon, making the corners look sharp.

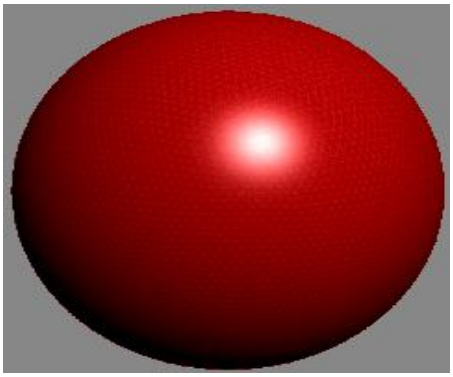
Smooth shading

In contrast to flat shading where the colors change discontinuously at polygon borders, with smooth shading the color changes from pixel to pixel, resulting in a smooth color transition between two adjacent polygons. Usually, values are first calculated in the vertices and bilinear interpolation is used to calculate the values of pixels between the vertices of the polygons.



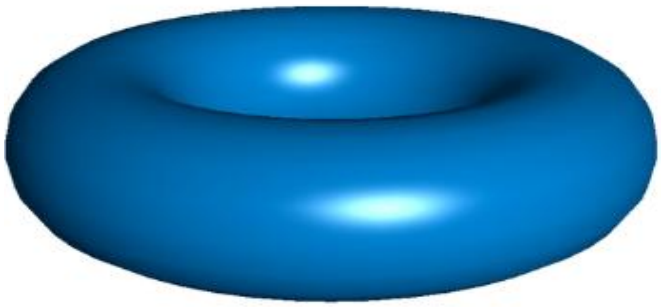
Gouraud Shading

Here, vertex normals are calculated and computes color at vertices and interpolates across surfaces. It may give shapes smooth look, but may miss specular highlights unless they fall on a vertex



Phong Shading

Here, lighting is calculated per-pixel using the interpolated normals across the surface and computes color per pixel. It ensures high realism, but more computationally intensive.



The Concept of Scan-Conversion

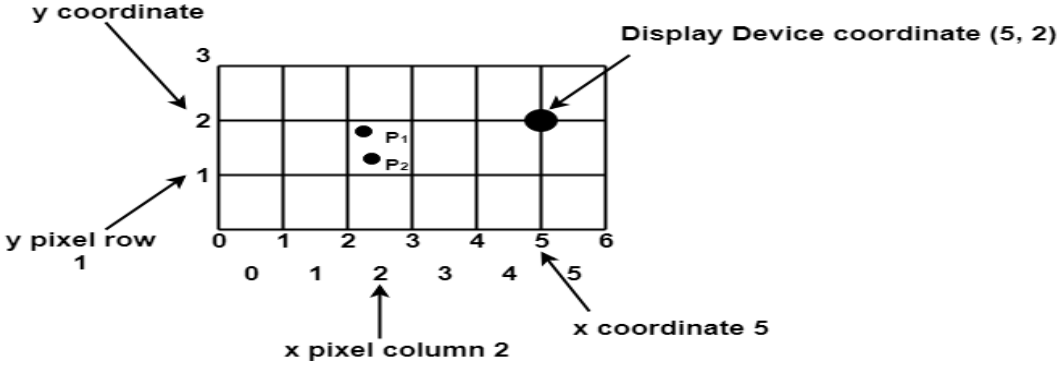
The process of representing continuous graphics objects as a collection of discrete pixels is called scan conversion.

Scan conversion is a video processing technique for changing the vertical / horizontal scan frequency of video signal for different purposes and applications.
The application of scan conversion is wide and covers video projectors, cinema equipment, TV and video capture cards, standard and HDTV televisions, LCD monitors, radar displays and many different aspects of picture processing.

Converting a Point

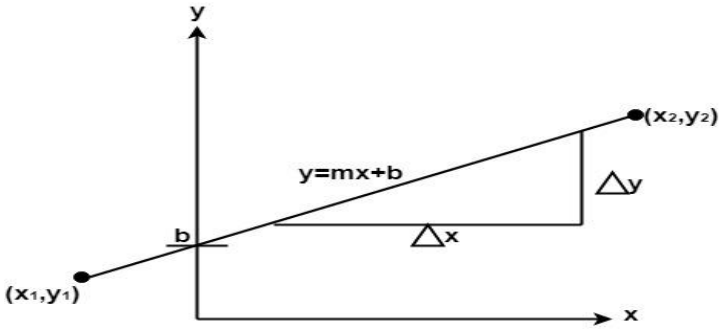
Each pixel on the graphics display does not represent a mathematical point. Instead, it means a region which theoretically can contain an infinite number of points. Scan-Converting a point involves illuminating the pixel that contains the point.

Example: Display coordinates points $P_1\ (2\frac{1}{4}, 1\frac{3}{4})$& $P_2(2\frac{2}{3}, 1\frac{1}{4})$as shown in fig would both be represented by pixel (2, 1). In general, a point p (x, y) is represented by the integer part of x & the integer part of y that is pixels [(INT (x), INT (y).



Converting a Straight Line

A straight line may be defined by two endpoints & an equation. In fig the two endpoints are described by (x_1,y_1) and (x_2,y_2). The equation of the line is used to determine the x, y coordinates of all the points that lie between these two endpoints.



Using the equation of a straight line, $y = mx + b$ where $m = \frac{\Delta y}{\Delta x}$ & b = the y interrupt, we can find values of y by incrementing x from $x = x_1$, to $x = x_2$. By scan-converting these calculated x, y values, we represent the line as a sequence of pixels.

Line Drawing Algorithms

a **line drawing algorithm** is an algorithm for approximating a line segment on discrete graphical media, such as pixel-based displays and printers. On such media, line drawing requires an approximation (in nontrivial cases).

Properties of Good Line Drawing Algorithm:

1. Line should appear Straight: We must appropriate the line by choosing addressable points close to it. If we choose well, the line will appear straight, if not, we shall produce crossed lines.

2. Lines should terminate accurately: Unless lines are plotted accurately, they may terminate at the wrong place.

3. Lines should have constant density: Line density is proportional to the no. of dots displayed divided by the length of the line.

4. Line density should be independent of line length and angle: This can be done by computing an approximating line-length estimate and to use a line-generation algorithm that keeps line density constant to within the accuracy of this estimate.

5. Line should be drawn rapidly: This computation should be performed by special-purpose hardware.

Algorithms for line Drawing:

1. Direct use of line equation
2. DDA (Digital Differential Analyzer)
3. Bresenham's Algorithm

Direct use of line equation:

It is the simplest form of conversion. First of all scan P_1 and P_2 points. P_1 has co-ordinates (x_1',y_1') and $(x_2'\ y_2')$.

Then $m = (y_2',y_1')/(x_2',x_1')$ and $b = y_1^1 + mx_1^1$

If value of $|m| \leq 1$ for each integer value of x. But do not consider x_1^1 and x_2^2

If value of |m|>1 for each integer value of y. But do not consider y_1^1 and y_2^2

Example: A line with starting point as (0, 0) and ending point (6, 18) is given. Calculate value of intermediate points and slope of line.

Solution: P_1 (0,0) P_7 (6,18)

$x_1=0$

$y_1=0$

$x_2=6$

$y_2=18$

$$M = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{18-0}{6-0} = \frac{18}{6} = 3$$

We know equation of line is

$y = m x + b$

$y = 3x + b$.....equation (1)

put value of x from initial point in equation (1), i.e., (0, 0) x =0, y=0

$0 = 3 \times 0 + b$

$0 = b \Longrightarrow b=0$

put b = 0 in equation (1)

$y = 3x + 0$

$y = 3x$

Now calculate intermediate points

Let $x = 1 \Longrightarrow y = 3 \times 1 \Longrightarrow y = 3$

Let $x = 2 \Longrightarrow y = 3 \times 2 \Longrightarrow y = 6$

Let $x = 3 \Longrightarrow y = 3 \times 3 \Longrightarrow y = 9$

Let $x = 4 \Longrightarrow y = 3 \times 4 \Longrightarrow y = 12$

Let $x = 5 \Longrightarrow y = 3 \times 5 \Longrightarrow y = 15$

Let $x = 6 \Longrightarrow y = 3 \times 6 \Longrightarrow y = 18$

So points are P_1 (0,0)

P_2 (1,3)

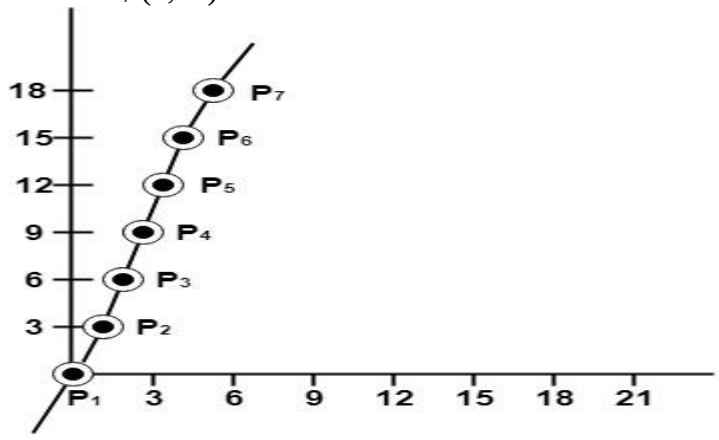
P_3 (2,6)

P_4 (3,9)

P_5 (4,12)

P_6 (5,15)

P_7 (6,18)



DDA Algorithm

DDA stands for Digital Differential Analyzer. It is an incremental method of scan conversion of line. In this method, calculation is performed at each step but by using results of previous steps. The difference in pixel points are to be analyzed before drawing a line segment.

STEP 1: Accept start and end points coordinates (X_1, Y_1) and (X_2, Y_2)
STEP 2: Calculate Dx = X_2 - X_1
Dy = Y_2 - Y_1
STEP 3: If abs (Dx) > abs(Dy) // M<1
then K = abs (Dx)
else K = abs(Dy)
STEP 4: Calculate $Dx = \frac{Dx}{k} and\ Dy = \frac{Dy}{k}$
Case1: When |M|<1 then $Xn = X1 + 1$, and $Yn = Y1 + M$
Case2: When |M| >1 then $Xn = X1 + \frac{1}{M}$, and $Yn = Y1 + 1$
Case3: When |M| =1 then $Xn = X1 + 1$, and $Yn = Y1 + 1$

Advantage:

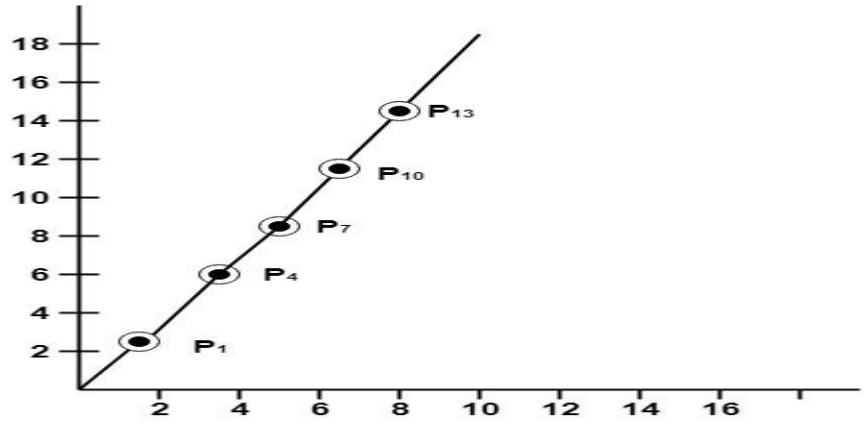
1. It is a faster method than method of using direct use of line equation.
2. This method does not use multiplication theorem.
3. It allows us to detect the change in the value of x and y, so plotting of same point twice is not possible.
4. This method gives overflow indication when a point is repositioned.
5. It is an easy method because each step involves just two additions.

Disadvantage:

1. It involves floating point additions rounding off is done. Accumulations of round off error cause accumulation of error.
2. Rounding off operations and floating point operations consumes a lot of time.
3. It is more suitable for generating line using the software. But it is less suited for hardware implementation.

Example: If a line is drawn from (2, 3) to (6, 15) with use of DDA. How many points will needed to generate such line?
Solution: P_1 (2,3) P_{11} (6,15)
x_1=2
y_1=3
x_2= 6
y_2=15
dx = 6 - 2 = 4
dy = 15 - 3 = 12
m = $\frac{dy}{dx} = \frac{12}{4}$
For calculating next value of x takes x = x + $\frac{1}{m}$

$P_1(2, 3)$	point plotted
$P_2(2\frac{1}{3}, 4)$	point plotted
$P_3(2\frac{2}{3}, 5)$	point not plotted
$P_4(3, 6)$	point plotted
$P_5(3\frac{1}{3}, 7)$	point not plotted
$P_6(3\frac{2}{3}, 8)$	point not plotted
$P_7(4, 9)$	point plotted
$P_8(4\frac{1}{3}, 10)$	point not plotted
$P_9(4\frac{2}{3}, 11)$	point not plotted
$P_{10}(5, 12)$	point plotted
$P_{11}(5\frac{1}{3}, 13)$	point not plotted
$P_{12}(5\frac{2}{3}, 14)$	point not plotted
$P_{13}(6, 15)$	point plotted



Bresenham's Line Algorithm

This algorithm is used for scan converting a line. It was developed by Bresenham. It is an efficient method because it involves only integer addition, subtractions, and multiplication operations. These operations can be performed very rapidly so lines can be generated quickly. The main purpose of the Bresenham's algorithm is to determine the points of an n-dimensional raster that should be selected in order to form a close approximation to a straight line between two points.

In this method, next pixel selected is that one who has the least distance from true line.
The method works as follows:
Assume a pixel $P_1'(x_1',y_1')$,then select subsequent pixels as we work our way to the night, one pixel position at a time in the horizontal direction toward $P_2'(x_2',y_2')$.
Once a pixel in choose at any step
The next pixel is

1. Either the one to its right (lower-bound for the line)
2. One top its right and up (upper-bound for the line)

The line is best approximated by those pixels that fall the least distance from the path between P_1',P_2'.

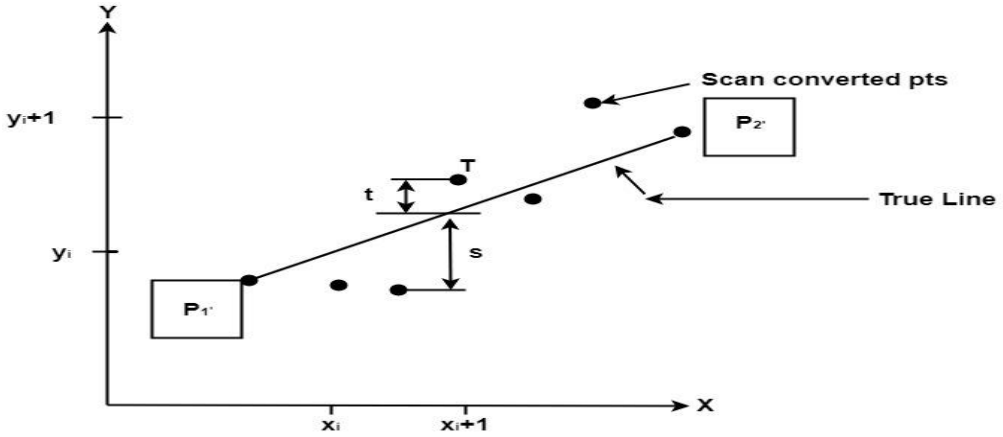


Fig: Scan Converting a line.

Advantage:

1. It involves only integer arithmetic, so it is simple.
2. It avoids the generation of duplicate points.
3. It can be implemented using hardware because it does not use multiplication and division.
4. It is faster as compared to DDA (Digital Differential Analyzer) because it does not involve floating point calculations like DDA Algorithm.

Disadvantage:

1. This algorithm is meant for basic line drawing only. Initializing is not a part of Bresenham's line algorithm. So to draw smooth lines, you should want to look into a different algorithm.

Bresenham's Line Algorithm:

Step1: Select the start and end point (x_1,y_1) ,(x_2,y_2)

Step2: Find the decision parameter value using;

$$di = 2dy - dx$$

(Decision parameter is used to find the exact point to draw the line)

where; $dy= y_2 - y_1$ and $dx = x_2 - x_1$

Step3: If $d_i > 0$ (Above true line)

$d_{i+1} = d_i + 2dy - 2dx$

$x_n = x_1 + 1$

$y_n = y_1 + 1$

Step4: If $d_i < 0$ (Below true line)

$d_{i+1} = d_i + 2dy$

$x_n = x_1 + 1$

$y_n = y_1$

Step5: Repeat step3 until the end point is reached.

Example: Starting and Ending position of the line are (1, 1) and (8, 5). Find intermediate points using Bresenham's algorithm.

Solution: $x_1=1$

$y_1=1$

$x_2=8$

$y_2=5$

$dx= x_2-x_1=8-1=7$

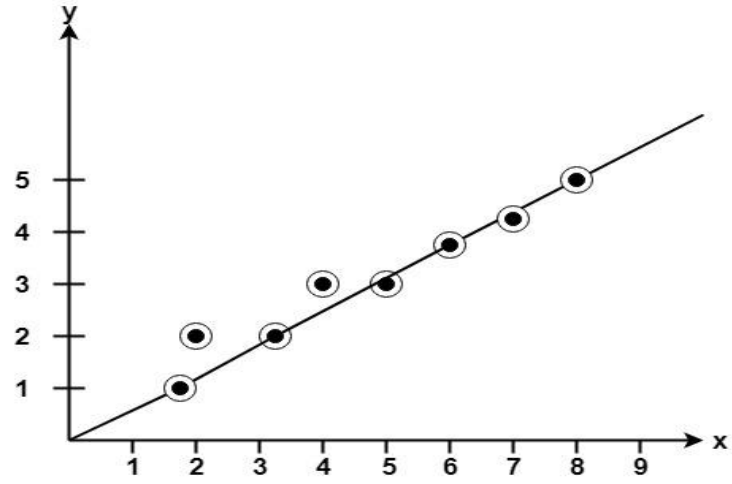
$dy=y_2-y_1=5-1=4$

$I_1=2* \Delta y=2*4=8$

$I_2=2*(\Delta y-\Delta x)=2*(4-7)=-6$

$d = I_1-\Delta x=8-7=1$

X	y	d=d+I_1 or I_2
1	1	d+I_2=1+(-6)=-5
2	2	d+I_1=-5+8=3
3	2	d+I_2=3+(-6)=-3
4	3	d+I_1=-3+8=5
5	3	d+I_2=5+(-6)=-1
6	4	d+I_1=-1+8=7
7	4	d+I_2=7+(-6)=1
8	5	



Example: The starting and ending points of a line is given by (9, 18) and (14, 22). Find the intermediate point that should be selected in order to form a close approximation of the line.

Differentiate between DDA Algorithm and Bresenham's Line Algorithm:

DDA Algorithm	Bresenham's Line Algorithm
1. DDA Algorithm use floating point, i.e., Real Arithmetic.	1. Bresenham's Line Algorithm use fixed point, i.e., Integer Arithmetic
2. DDA Algorithms uses multiplication & division its operation	2.Bresenham's Line Algorithm uses only subtraction and addition its operation

3. DDA Algorithm is slowly than Bresenham's Line Algorithm in line drawing because it uses real arithmetic (Floating Point operation)	3. Bresenham's Algorithm is faster than DDA Algorithm in line because it involves only addition & subtraction in its calculation and uses only integer arithmetic.
4. DDA Algorithm is not accurate and efficient as Bresenham's Line Algorithm.	4. Bresenham's Line Algorithm is more accurate and efficient at DDA Algorithm.
5.DDA Algorithm can draw circle and curves but are not accurate as Bresenham's Line Algorithm	5. Bresenham's Line Algorithm can draw circle and curves with more accurate than DDA Algorithm.

Circle Generation Algorithm

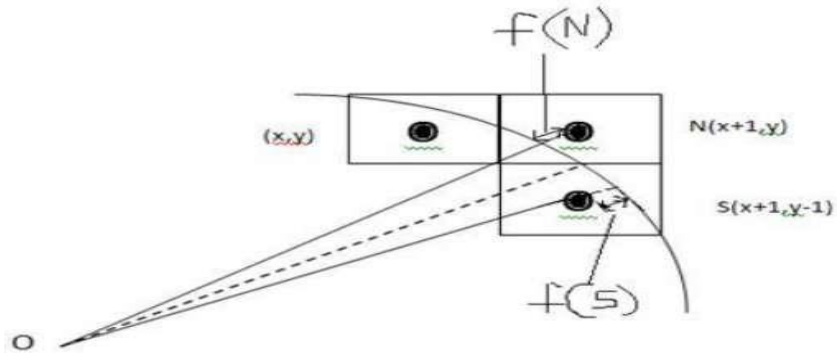
Drawing a circle on the screen is a little complex than drawing a line.
There are two popular algorithms for generating a circle

- Bresenham's Algorithm
- 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 $X^2 + Y^2 = r^2$, 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.

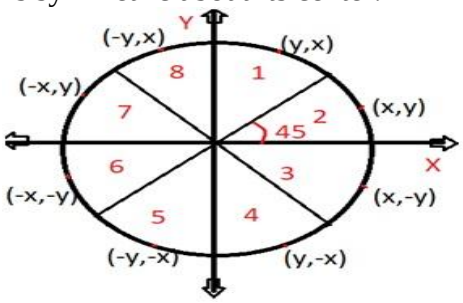
Algorithm

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*.

Midpoint circle Algorithm

This is an algorithm which is used to calculate the entire perimeter points of a circle in a first octant so that the points of the other octant can be taken easily as they are mirror points; this is due to circle property as it is symmetric about its center.



In this algorithm decision parameter is based on a circle equation. As we know that the equation of a circle is $x^2 + y^2 = r^2$ when the centre is (0, 0).

Now let us define the function of a circle i.e.: **fcircle(x,y)= $x^2 + y^2 - r^2$**

1. If **fcircle < 0** then **x**, **y** is inside the circle boundary.
2. If **fcircle > 0** then **x**, **y** is outside the circle boundary.
3. If **fcircle = 0** then **x**, **y** is on the circle boundary.

Decision parameter

p_k =fcircle(x_{k+1},$y_{k-1/2}$) where **p_k** is a decision parameter and in this ½ is taken because it is a midpoint value through which it is easy to calculate value of **y_k** and **y_{k-1}**.

I.e. **$p_k = (x_{k+1})^2 + (y_{k-1/2})^2 - r^2$**

If **p_k <0** then midpoint is inside the circle in this condition we select **y** is **y_k** otherwise we will select next **y** as **y_{k-1}** for the condition of **p_k > 0**.

Conclusion

1. If **p_k < 0** then **$y_{k+1}=y_k$**, by this the plotting points will be **(x_{k+1} ,y_k)**. By this the value for the next point will be given as:
   $P_{k+1}=p_k +2(x_{k+1}) +1$
2. If **p_k > 0** then **$y_{k+1}=y_{k-1}$**, by this the plotting points will be **(x_{k+1}, y_{k-1})**. By this the value of the next point will be given as:
   $P_{k+1}=p_k+2(x_{k+1}) +1-2(y_{k+1})$

Initial decision parameter

P_0 = fcircle (1, r-1/2)

This is taken because of **(x_0, y_0) = (0, r)**

i.e. **p_0 =5/4-r or 1-r**, (**1-r** will be taken if **r** is integer)

ALGORITHM

1. In this the input radius **r** is there with a centre **(x_c , y_c)**. To obtain the first point **m** the circumference of a circle is centered on the origin as **(x_0,y_0) = (0,r)**.
2. Calculate the initial decision parameters which are:
   p_0 =5/4-r or 1-r
3. Now at each **x_k** position starting **k=0**, perform the following task.
   if **p_k < 0** then plotting point will be **(x_{k+1} ,y_k)** and
   $P_{k+1}=p_k +2(x_{k+1}) +1$

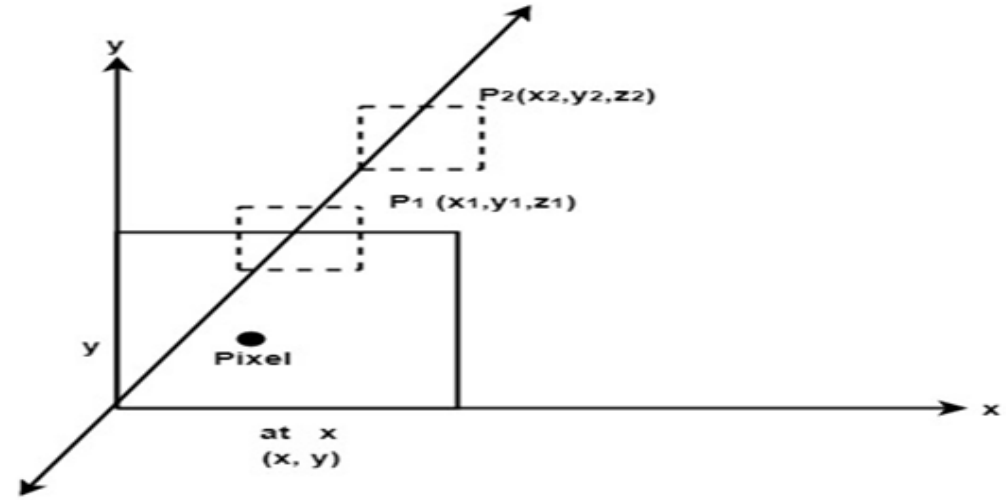
else the next point along the circle is (x_{k+1}, y_{k-1}) and
$P_{k+1}=p_k+2(x_{k+1})+1-2(y_{k+1})$

4. Determine the symmetry points in the other quadrants.
5. Now move at each point by the given centre that is:
   $x=x+x_c$
   $y=y+y_c$
6. At last repeat steps from 3 to 5 until the condition **x>=y**.

Hidden-surface problem.
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.

Z-Buffer Algorithm
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 visiblevisible 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 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.

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

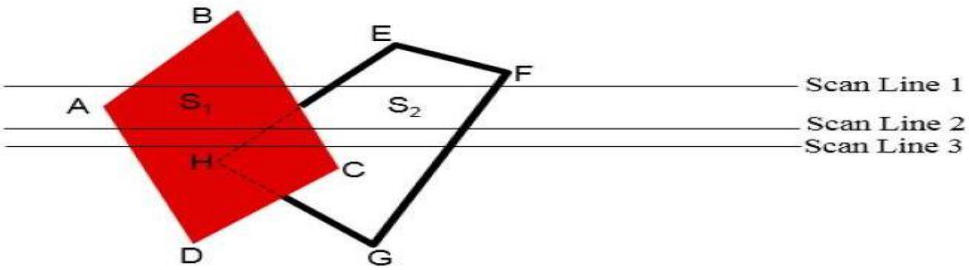
- It requires large memory.
- It is time consuming process.

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.

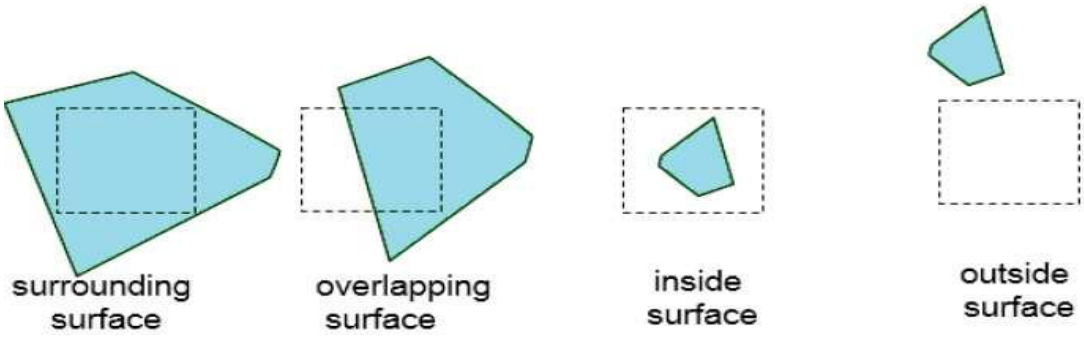
Area-Subdivision Method

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.

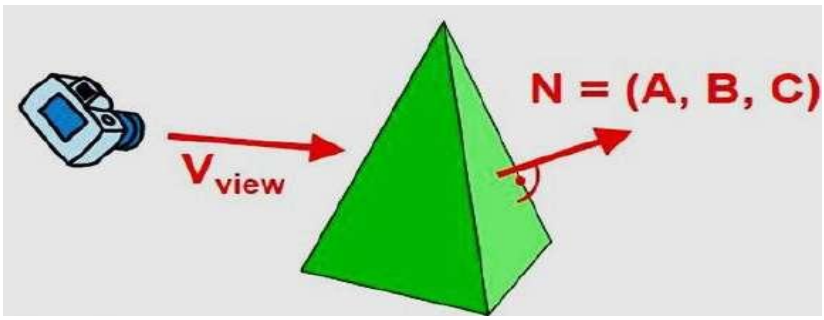


The tests for determining surface visibility within an area can be stated in terms of these four classifications. No further subdivisions of a specified area are needed if one of the following conditions is true −

- 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

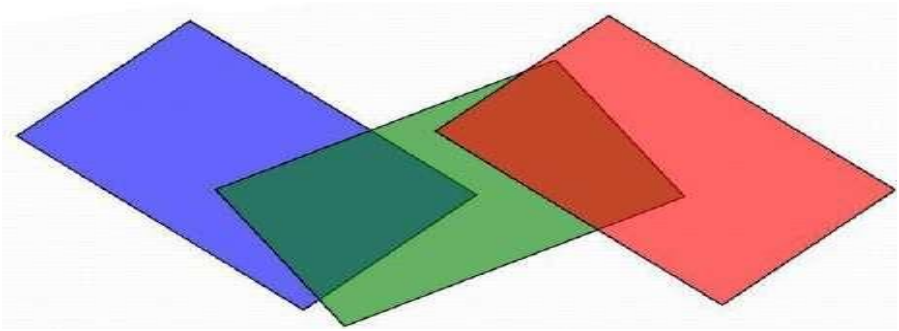
A fast and simple object-space method for identifying the back faces of a polyhedron is based on the "inside-outside" tests. A point x,y,z is "inside" a polygon surface with plane parameters A, B, C, and D if When an inside point is along the line of sight to the surface, the polygon must be a back face



A-Buffer Method

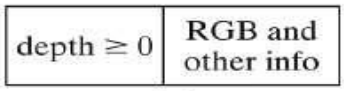
The A-buffer method is an extension of the depth-buffer method. The A-buffer method is a visibility detection method developed at Lucas film Studios for the rendering system Renders Everything You Ever Saw REYESREYES.

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

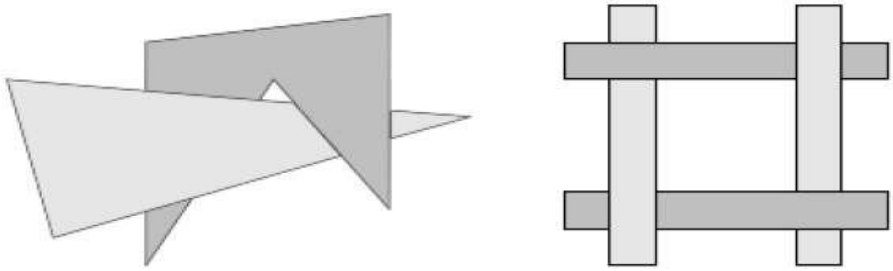
The algorithm proceeds just like the depth buffer algorithm. The depth and opacity values are used to determine the final color of a pixel.

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.