NAME: TAUSIF KHAN

SUBJECT: COMPUTER GRAPHICS

ROLL NO: 12

STD: SE COMPS(B)

ASSIGNMENT NO: 02

1. Derive the matrix for 2D rotation about an arbitary point.

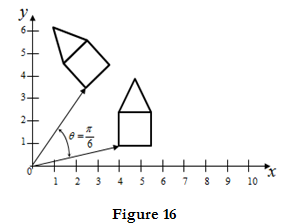
Ans 1. Transformation means change in image.

We can modify the image by performing some basic transformation such as.

* Scaling.
* Rotation.
* Translation.

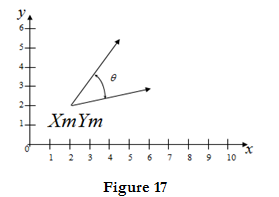
**Rotation:**

For rotation we need trigonometry logic. Suppose we have point P1 = (x1, y1) and we rotate it about the original by an angle θ to get a new position P2 = (x2, y2) as shown in figure 16



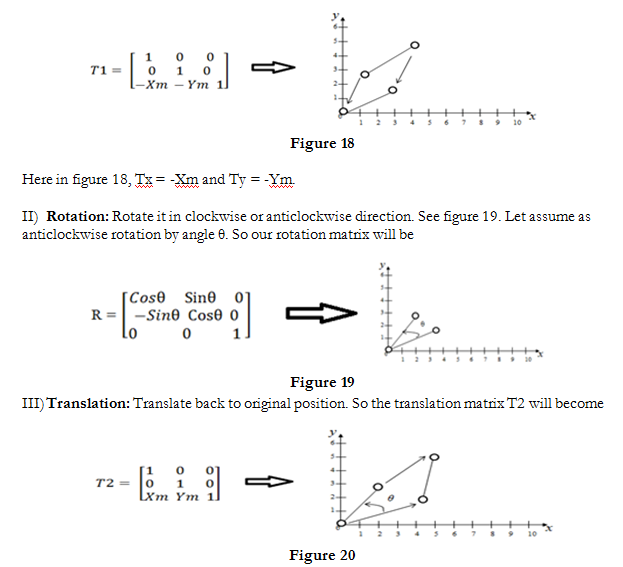
**Rotation about arbitrary point:**

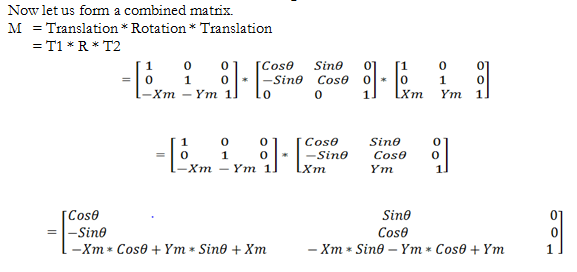
Suppose the reference point of rotation is other than origin, then in that case we have to follow series of transformation.



Consider figure 17, assume that we have to rotate a point P1 with respect to (Xm, Ym) then we have to perform three steps.

I) **Translation:** First we have to translate the (Xm, Ym) to origin as shown in figure 18. Translation matrix (T1) will become



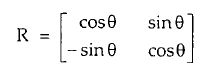


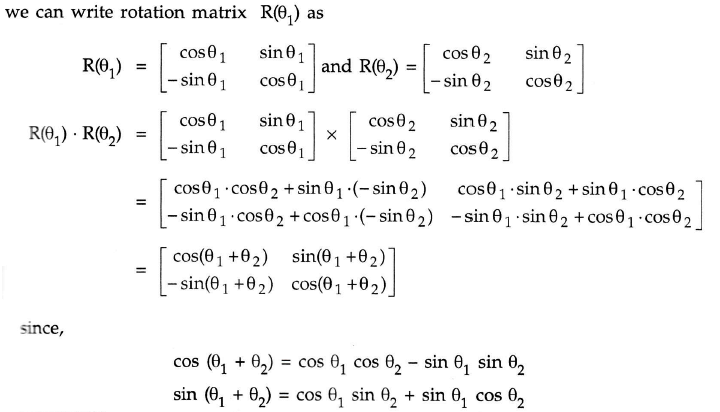
This transformation matrix is the overall transformation matrix for rotation about arbitrary point (Xm, Ym) by an angle θ in anticlockwise direction.

3. Show that the composition of two successive rotation are additive.

i.e R(ϴ1). R(ϴ2)=R(ϴ1+ϴ2)

Ans 3.  The Rotation matrix R is given as,





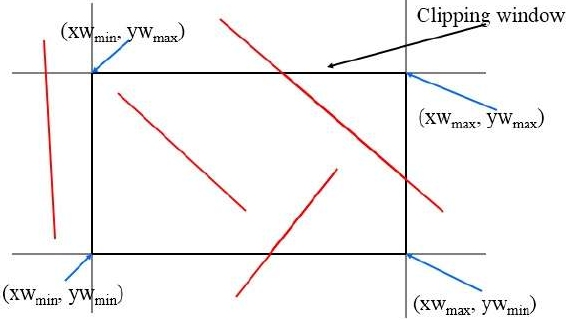
4. Explain cohen sutherland clipping algorithm for line with suitable example.

Ans 4. **Line Clipping:**

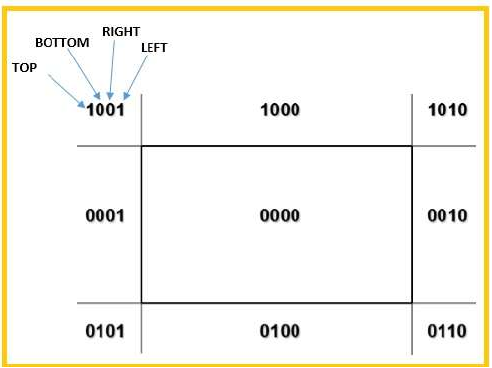
The concept of line clipping is same as point clipping. In line clipping, we will cut the portion of line which is outside of window and keep only the portion that is inside the window.

**Cohen-Sutherland Line Clippings:**

* This algorithm uses the clipping window as shown in the following figure. The minimum coordinate for the clipping region is( XWmmiinn,YWmmiinn)(XWmmiinn,YWmmiinn) and the maximum coordinate for the clipping region is (XWmmaaxx,YWmmaaxx)(XWmmaaxx,YWmmaaxx).



* 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 ANDoperation 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.

**Conclusion:**

In summary, the C-S algorithm is efficient when outcode testing can be done cheaply (for example, by doing bitwise operations in assembly language) and trivial acceptance or rejection is applicable to the majority of line segments .(For example, large windows - everything is inside , or small windows - everything is outside).

5. Explain Liang Barsky line clipping algorithm with suitable example.

# Ans 5. Liang-Barsky Algorithm

The **Liang-Barsky algorithm** is a line clipping algorithm. This algorithm is more efficient than Cohen–Sutherland line clipping algorithm and can be extended to 3-Dimensional clipping. This algorithm is considered to be the faster parametric line-clipping algorithm. The following concepts are used in this clipping:

1. The parametric equation of the line.
2. The inequalities describing the range of the clipping window which is used to determine the intersections between the line and the clip window.

The parametric equation of a line can be given by,

X = x1 + t(x2-x1)

Y = y1 + t(y2-y1)

Where, t is between 0 and 1.

Then, writing the point-clipping conditions in the parametric form:

xwmin <= x1 + t(x2-x1) <= xwmax

ywmin <= y1 + t(y2-y1) <= ywmax

The above 4 inequalities can be expressed as,

tpk <= qk

Where k = 1, 2, 3, 4 (correspond to the left, right, bottom, and top boundaries, respectively).

The p and q are defined as,

p1 = -(x2-x1), q1 = x1 - xwmin (Left Boundary)

p2 = (x2-x1), q2 = xwmax - x1 (Right Boundary)

p3 = -(y2-y1), q3 = y1 - ywmin (Bottom Boundary)

p4 = (y2-y1), q4 = ywmax - y1 (Top Boundary)

When the line is parallel to a view window boundary, the p value for that boundary is zero.  
When pk < 0, as t increase line goes from the outside to inside (entering).  
When pk > 0, line goes from inside to outside (exiting).  
When pk = 0 and qk < 0 then line is trivially invisible because it is outside view window.  
When pk = 0 and qk > 0 then the line is inside the corresponding window boundary.

Using the following conditions, the position of line can be determined:

| **CONDITION** | **POSITION OF LINE** |
| --- | --- |
| pk = 0 | parallel to the clipping boundaries |
| pk = 0 and qk < 0 | completely outside the boundary |
| pk = 0 and qk >= 0 | inside the parallel clipping boundary |
| pk < 0 | line proceeds from outside to inside |
| pk > 0 | line proceeds from inside to outside |

Parameters t1 and t2 can be calculated that define the part of line that lies within the clip rectangle.  
When,

1. pk < 0, maximum(0, qk/pk) is taken.
2. pk > 0, minimum(1, qk/pk) is taken.

If t1 > t2, the line is completely outside the clip window and it can be rejected. Otherwise, the endpoints of the clipped line are calculated from the two values of parameter t.

**Algorithm –**

1. Set tmin=0, tmax=1.
2. Calculate the values of t (t(left), t(right), t(top), t(bottom)),  
   (i) If t < tmin ignore that and move to the next edge.  
   (ii) else separate the t values as entering or exiting values using the inner product.  
   (iii) If t is entering value, set tmin = t; if t is existing value, set tmax = t.
3. If tmin < tmax, draw a line from (x1 + tmin(x2-x1), y1 + tmin(y2-y1)) to (x1 + tmax(x2-x1), y1 + tmax(y2-y1))
4. If the line crosses over the window, (x1 + tmin(x2-x1), y1 + tmin(y2-y1)) and (x1 + tmax(x2-x1), y1 + tmax(y2-y1)) are the intersection point of line and edge.

This algorithm is presented in the following code. Line intersection parameters are initialised to the values t1 = 0 and t2 = 1.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| #include"graphics.h"  #define ROUND(a) ((int)(a+0.5))  int clipTest (float p,float q, float \* tl, float \* t2)  {  float r ;  int retVal = TRUE;    //line entry point  if (p < 0.0) {        r = q /p ;        // line portion is outside the clipping edge      if ( r > \*t2 )      retVal = FALSE;        else      if (r > \*t1 )      \*tl = r;  }    else    //line leaving point  if (p>0.0) {      r = q/p ;        // line portion is outside      if ( r<\*t1 )          retVal = FALSE;        else i f (r<\*t2)          \*t2 = r;  }    // p = 0, so line is parallel to this clipping edge  else    // Line is outside clipping edge  if (q<0.0)  retVal = FALSE;    return ( retVal ) ;  }    void clipLine (dcPt winMin, dcPt winMax, wcPt2 pl , wcPt2 p2)    {  float t1 = 0.0, t2 = 1.0, dx = p2.x-p1.x, dy;     // inside test wrt left edge  if(clipTest (-dx, p1.x - winMin.x, &t1, &t2))     // inside test wrt right edge  if(clipTest (dx, winMax.x - p1.x, &t1, &t2))    {      dy = p2.y - p1.y;        // inside test wrt bottom edge      if(clipTest (-dy, p1.y - winMin.y, &t1, &t2))            // inside test wrt top edge          if(clipTest (dy, winMax.y - p1.y, &t1, &t2)) {            if(t2 < 1.0) {              p2.x = p1.x + t2\*dx;              p2.y = p1.y + t2\*dy;          }            if(t1 > 0.0) {              p1.x += t1\*dx;              p1.y += t1\*dy;          }            lineDDA ( ROUND(p1.x), ROUND(p1.y), ROUND(p2.x), ROUND(p2.y) );            }  }    } |

6. Explain weiler-Atherton algorithm for polygon clipping. What are the advantages over the other polygon clipping algorithm. Explain its working with an example.

Ans 5.

### Algorithm:

**1. First make a list of all intersection points namely i1, i2, i3, ...**

**2. Classify those intersection points as entering or exiting.**

**3. Now, make two lists, one for the clipping polygon, and the other**

**for the clipped polygon.**

**4. Fill both the lists up in such a way that the intersection points**

**lie between the correct vertices of each of the polygon. That is**

**the clipping polygon list is filled up with all the vertices of**

**the clipping polygon along with the intersecting points lying**

**between the corresponding vertices.**

**5. Now, start at the 'to be clipped' polygon's list.**

**6. Choose the first intersection point which has been labelled as**

**an entering point. Follow the points in the list (looping back to**

**the top of the list, in case the list ends) and keep on pushing**

**them into a vector or something similar of the sorts. Keep on following**

**the list until an exiting intersection point is found.**

**7. Now switch the list to the 'polygon that is clipping' list, and find**

**the exiting the intersection that was previously encountered. Now keep**

**on following the points in this list (similar to how we followed the**

**previous list) until the entering intersection point is found (the**

**one that was found in the previous 'to be clipped' polygon's list).**

**8. This vector now formed by pushing all the encountered points in the**

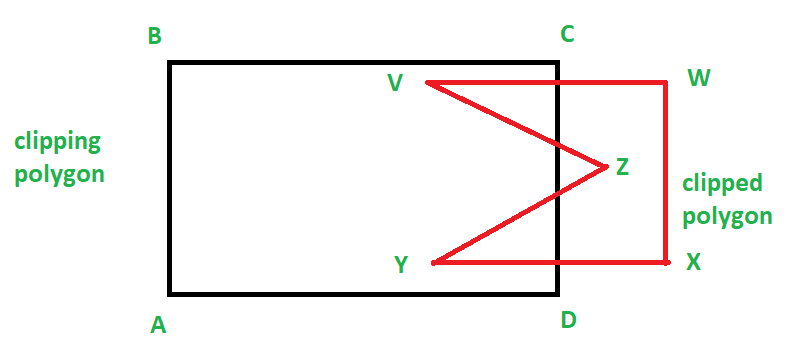
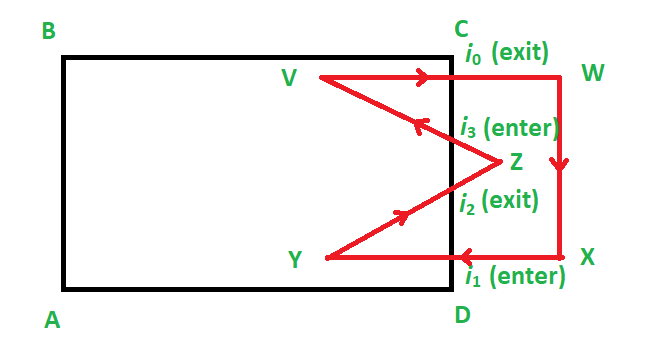
**two lists, is now the clipped polygon (one of the many clipped**

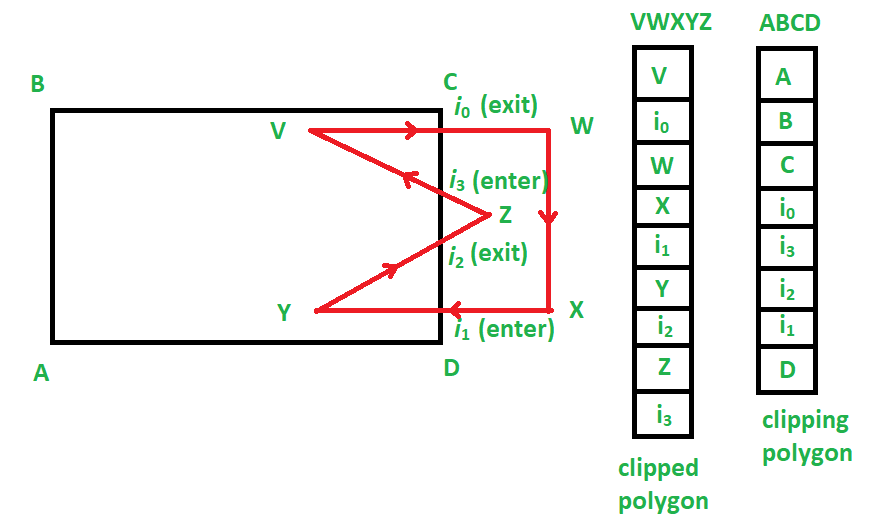
**polygons if any of the clipping polygons is concave).**

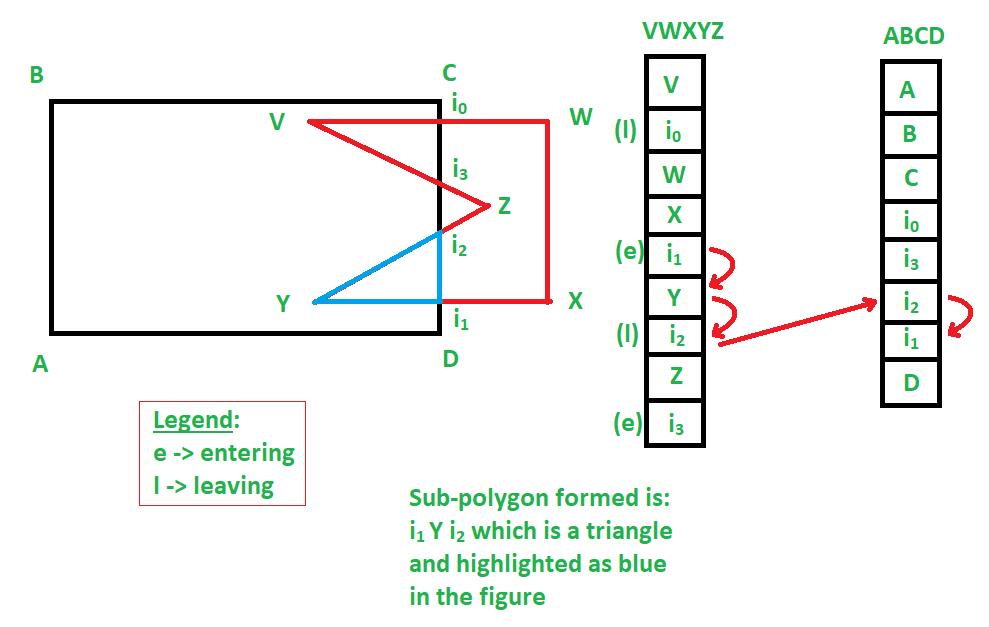
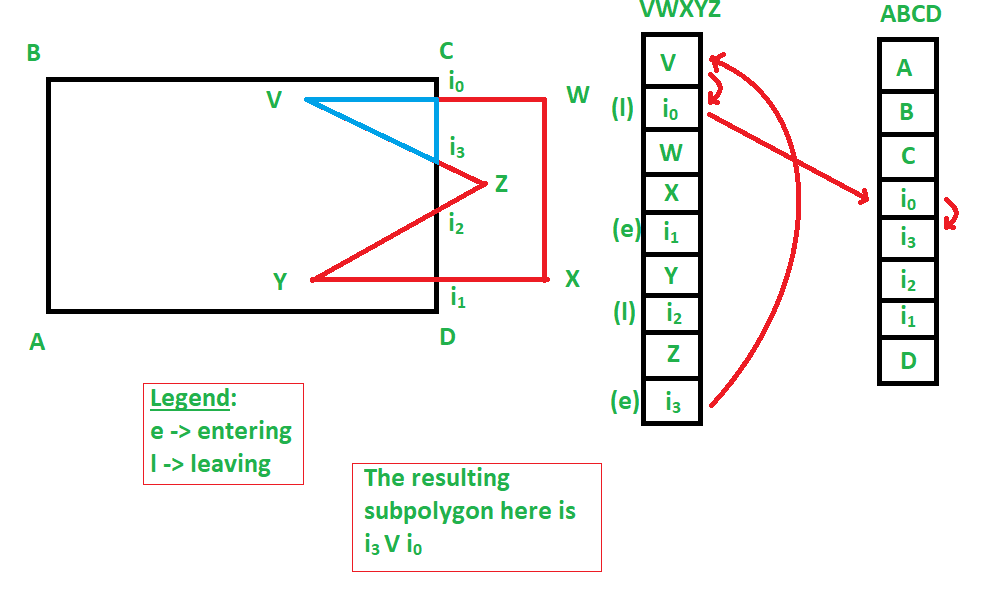
**9. Repeat this clipping procedure (i.e. from step 5) until all the**

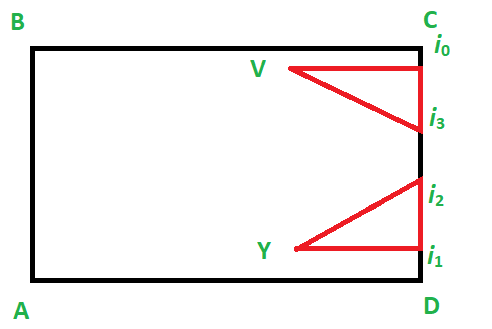
**entering intersection points have been visited once.**

### Explanation:

**1. Finding all the intersection points and grouping them**  
Here, let there be a polygon ABCD and another polygon VWXYZ. Let ABCD be the clipping polygon and let VWXYZ be the clipped polygon.  
  
So, we can find the intersection points using any method. For example, we can find the intersecting points separately and then find for each intersecting point find if it is entering or leaving, or, we can use Cyrus Beck and find all the intersecting points and also get if a point is entering or exiting. Refer [Cyrus Beck](https://www.geeksforgeeks.org/cyrus-beck/) for more information on this algorithm.  


**2. Making and filling of two lists**  
Now, we make two lists. One for the clipping polygon and one for the clipped polygon.  
Now this is how we fill it:  


**3. Running of the algorithm**  
We start at the clipped polygon’s list, i.e. VWXYZ.  
Now, we find the first intersecting point that is entering. Hence we choose i1.  
From here we begin the making of the list of vertices (or vector) to make a clipped sub-polygon.  
  
According to the given example, **i1 Y i2** is a clipped sub-polygon.  
Similarly, we get:  
  
**i0 V i3** as another sub-polygon also.

Hence, we were able to get two sub-polygons as a result of this polygon clipping, which involved a concave polygon, which resulted in:  
  
Similarly, this clipping works for convex polygons.

NAME: TAUSIF KHAN

SUBJECT: COMPUTER GRAPHICS

STD: SE COMPS(B)

ROLL NO: 12

ASSIGNMENT NO: 03

1. What are parallel and perspective projection and derive the matrix for perspective projection.

Ans 1.

**Parallel Projection**

1. Parallel projection discards z-coordinate and parallel lines from each vertex on the object are extended until they intersect the view plane.
2. In parallel projection, we specify a direction of projection instead of center of projection.
3. In parallel projection, the distance from the center of projection to project plane is infinite.
4. In this type of projection, we connect the projected vertices by line segments which correspond to connections on the original object.
5. Parallel projections are less realistic, but they are good for exact measurements.
6. In this type of projections, parallel lines remain parallel and angles are not preserved.
7. Various types of parallel projections are shown in the following hierarchy.

**a) 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

**b) Oblique Projection**

* In orthographic 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 –

**Perspective Projection**

1. 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.
2. The distance and angles are not preserved and parallel lines do not remain parallel.
3. Instead, they all converge at a single point called center of projection or projection reference point.
4. 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.
5. The following figure shows all the three types of perspective projection

**Matrix for Perspective Projection**

* Homogenous coordinates allow to work with the translation as the rotation and scale
  + In order to get a square matrix, a new row is added and a new coordinate w’ appears
  + If the last row is [0 0 1] then w’ = 1
  + If w’<>1 it is projected over the plane w=1, that’s called the homogenous division
  + In the case of 3D, we work with 4 components and the 3D point we get is [x/w, y/w, z/w, 1]
* Simple projection matrices
  + A LEFT - HANDED system is used
  + CP at the origin of coordinates and PP perpendicular to Z at a distance of d
* Simple perspective projection

2. Explain the properties of Bezier Curves

Ans 2.

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.

3. Define Fractals? Give Classification of Fractals? What is Fractal Dimensiions?

Ans 3.

Fractals:

A curve or geometrical figure, each part of which has the same statistical character as the whole. They are useful in modelling structures (such as snowflakes) in which similar patterns recur at progressively smaller scales, and in describing partly random or chaotic phenomena such as crystal growth and galaxy formation.

Fractals can also be classified according to their self-similarity. There are three types of self-similarity found in fractals:

* Exact self-similarity — This is the strongest type of self-similarity; the fractal appears identical at different scales. Fractals defined by iterated function systems often display exact self-similarity.
* Quasi-self-similarity — This is a loose form of self-similarity; the fractal appears approximately (but not exactly) identical at different scales. Quasi-self-similar fractals contain small copies of the entire fractal in distorted and degenerate forms. Fractals defined by recurrence relations are usually quasi-self-similar but not exactly self-similar.
* Statistical self-similarity — This is the weakest type of self-similarity; the fractal has numerical or statistical measures which are preserved across scales. Most reasonable definitions of "fractal" trivially imply some form of statistical self-similarity. (Fractal dimension itself is a numerical measure which is preserved across scales.) Random fractals are examples of fractals which are statistically self-similar, but neither exactly nor quasi-self-similar.

Fractal dimension:

All fractals have dimensions that are fractions, not whole numbers. We can make some sense out of the dimension, by comparing it to the simple, whole number dimensions. If a line is 1-Dimensional, and a plane is 2-Dimensional, then a fractional dimension of 1.26 falls somewhere in between a line and a plane.

4. Explain sweep Representation.

Ans 4.

1. Solid Modeling Presented by Pramod Poudel 1
2. Solid Modeling Computer representation of a physical solid object Creation or visualization of digital model 2
3. Solid Modeling • Solid modeling is based on complete, valid and unambiguous geometric representation of physical object. – Complete ◊ points in space can be classified.(inside/outside) – Valid ◊vertices, edges, faces are connected properly. – Unambiguous ◊ there can only be one interpretation of object • Solid model consist of geometric and topological data – Geometry ◊ graphical information like shape, size, angle – Topology ◊invisible information like connectivity 3
4. Representing Solid 4
5. 5 A solid model is the more complete representation than it’s surface (wireframe model)
6. Boolean Operation • allows objects by defining new objects 6 a) Objects A and B b)The AU B c) A ∩ B d) A - B e) B - A
7. Sweep representation • Is a method to generate solid with the help of 2D structures • Sweeping an object along a trajectory through space defines a new object called a sweep.The displacement of an object according to a trajectory defines another object 7
8. 8 Rotational Sweep
9. Translational Sweep 9
10. B - Representation • extension to the wireframe model. • describes the solid in terms of its surface boundaries:Vertices, edges and faces • There are 2 types of information in a B – rep topological and geometric. 10
11. Polyhedron • A polyhedron is a solid in three dimensions with flat polygonal faces, straight edges and sharp corners 11
12. • A simple polyhedron without holes, obeys Euler's formula: 12 V - E + F = 2 V -Vertex E - Edges (edges) F - Faces
13. Generalization of Euler’s Formula for polyhedron with holes 13 V - E + F - H = 2 (C - G) V -Vertex E - Edges (edges) F - Faces H - number of holes in the sides G - Number of holes crossing the object C - number of parts of the object

5. Explain z-buffer algorithm for removing hidden surface.

Ans 5.

When viewing a picture containing non transparent objects and surfaces, it is not possible to see those objects from view which are behind from the objects closer to eye. To get the realistic screen image, removal of these hidden surfaces is must. The identification and removal of these surfaces is called as the **Hidden-surface problem.**

Z-buffer, which is also known as the Depth-buffer method is one of the commonly used method for hidden surface detection. It is an **Image space method**. Image space methods are based on the pixel to be drawn on 2D. For these methods, the running time complexity is the number of pixels times number of objects. And the space complexity is two times the number of pixels because two arrays of pixels are required, one for frame buffer and the other for the depth buffer.

The Z-buffer method compares surface depths at each pixel position on the projection plane. Normally z-axis is represented as the depth. The algorithm for the Z-buffer method is given below :

**Algorithm :**

First of all, initialize the depth of each pixel.

i.e, d(i, j) = infinite (max length)

Initialize the color value for each pixel

as c(i, j) = background color

for each polygon, do the following steps :

for (each pixel in polygon's projection)

{

find depth i.e, z of polygon

at (x, y) corresponding to pixel (i, j)

if (z < d(i, j))

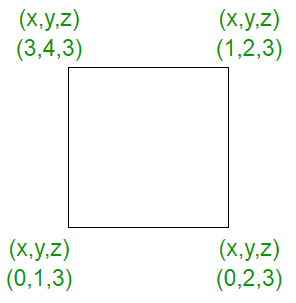
{

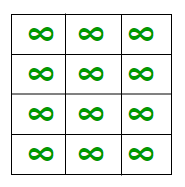
d(i, j) = z;

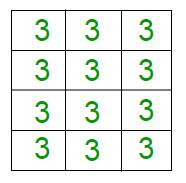
c(i, j) = color;

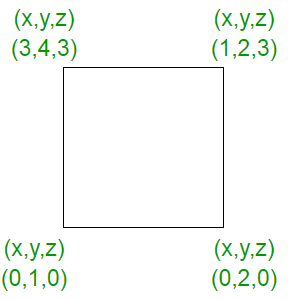
}

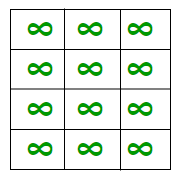
}

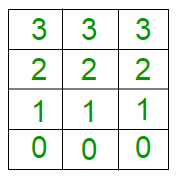
Let’s consider an example to understand the algorithm in a better way. Assume the polygon given is as below :  


In starting, assume that the depth of each pixel is infinite.  


As the z value i.e, the depth value at every place in the given polygon is 3, on applying the algorithm, the result is:  


Now, let’s change the z values. In the figure given below, the z values goes from 0 to 3.  


In starting, the depth of each pixel will be infinite as :  


Now, the z values generated on the pixel will be different which are as shown below :  


Therefore, in the Z buffer method, each surface is processed separately one position at a time across the surface. After that the depth values i.e, the z values for a pixel are compared and the closest i.e, (smallest z) surface determines the color to be displayed in frame buffer. The z values, i.e, the depth values are usually normalized to the range [0, 1]. When the z = 0, it is known as ***Back Clipping Pane*** and when z = 1, it is called as the ***Front Clipping Pane***.

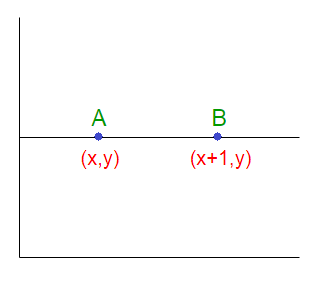
In this method, 2 buffers are used :

1. Frame buffer
2. Depth buffer

**Calculation of depth :**  
As we know that the equation of the plane is :

ax + by + cz + d = 0, this implies

z = -(ax + by + d)/c, c!=0

Calculation of each depth could be very expensive, but the computation can be reduced to a single add per pixel by using an increment method as shown in figure below :  


Let’s denote the depth at point A as Z and at point B as Z’. Therefore :

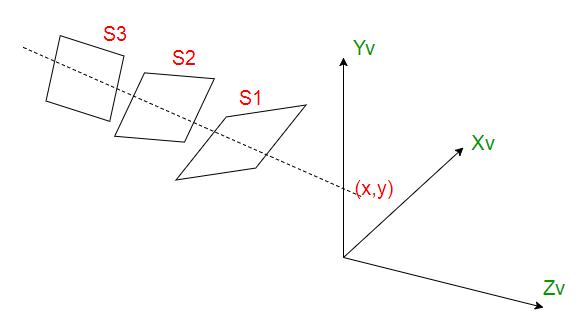
AX + BY + CZ + D = 0 implies

Z = (-AX - BY - D)/C ------------(1)

Similarly, Z' = (-A(X + 1) - BY -D)/C ----------(2)

Hence from (1) and (2), we conclude :

Z' = Z - A/C ------------(3)

Hence, calculation of depth can be done by recording the plane equation of each polygon in the (normalized) viewing coordinate system and then using the incremental method to find the depth Z.  
So, to summarize, it can be said that this approach compares surface depths at each pixel position on the projection plane. Object depth is usually measured from the view plane along the z-axis of a viewing system.  
Example :  


Let S1, S2, S3 are the surfaces. The surface closest to projection plane is called visible surface. The computer would start (arbitrarily) with surface 1 and put it’s value into the buffer. It would do the same for the next surface. It would then check each overlapping pixel and check to see which one is closer to the viewer and then display the appropriate color. As at view-plane position (x, y), surface S1 has the smallest depth from the view plane, so it is visible at that position.

6. Explain the back face removal algorithm.

Ans 6.

# Back Face Removal Algorithm

It is used to plot only surfaces which will face the camera. The objects on the back side are not visible. This method will remove 50% of polygons from the scene if the parallel projection is used. If the perspective projection is used then more than 50% of the invisible area will be removed. The object is nearer to the center of projection, number of polygons from the back will be removed.

It applies to individual objects. It does not consider the interaction between various objects. Many polygons are obscured by front faces, although they are closer to the viewer, so for removing such faces back face removal algorithm is used.

When the projection is taken, any projector ray from the center of projection through viewing screen to object pieces object at two points, one is visible front surfaces, and another is not visible back surface.

This algorithm acts a preprocessing step for another algorithm. The back face algorithm can be represented geometrically. Each polygon has several vertices. All vertices are numbered in clockwise. The normal M1 is generated a cross product of any two successive edge vectors. M1represent vector perpendicular to face and point outward from polyhedron surface

                          N1=(v2-v1 )(v3-v2)  
                          If         N1.P≥0 visible  
                          N1.P<0 invisible

## Advantage

1. It is a simple and straight forward method.
2. It reduces the size of databases, because no need of store all surfaces in the database, only the visible surface is stored.

## Back Face Removed Algorithm

Repeat for all polygons in the scene.

1. Do numbering of all polygons in clockwise direction i.e.  
                 v1 v2 v3.....vz
2. Calculate normal vector i.e. N1  
                 N1=(v2-v1 )\*(v3-v2)
3. Consider projector P, it is projection from any vertex  
                 Calculate dot product  
                 Dot=N.P
4. Test and plot whether the surface is visible or not.  
                 If Dot ≥ 0 then  
               surface is visible  
               else  
                       Not visible

7. Explain and compare goraud shading and phong shading.

Ans 7.

|  |  |  |
| --- | --- | --- |
| Sno | Gouraud Shading | Phong Shading |
| 1. | Gouraud shading, named after Henri Gouraud. | Phong Shading model named after Bui Tuong Phong. |
| 2. | Gouraud first published the technique in 1971. | Phong Shading published the  technique in 1973. |
| 3. | Computes illumination at border vertics and interpolates. | Illumination at every point of polygon surface. |
| 4. | Interpolates colors along edges and scanline. | Interpolates normals instead of colors |
| 5. | Not So expensive. | More Expensive than Gouraud Shading |
| 6. | Requires Moderate processing and time. | Required Complex processing and it is Slower. Its Produce Good Quality. |
| 7. | Lighting Equation used at each Vertex. | Lighting Equation used at each pixel. |
| 8. | Gourard shading is in between the two: like Phong shading, each polygon has one normal vector per vertex, but instead of interpolating the vectors, the color of each vertex is computed and then interpolated across the surface of the polygon. | Each rendered polygon has one normal vector per vertex; shading is performed by interpolating the vectors across the surface and computing the color for each point of interest. |
| 9. | Smooth surfaces. | Smooth and shining surfaces. |

8. What do your understand by diffuse illumination and point source illumintion

Ans 8.

Diffuse illumination: :

An objects illumination is as important as its surface properties in computing its intensity. ... When such illumination is uniform from all directions, the illumination is called diffuse illumination. Basically the diffuse illumination is a background light which is reflected from walls, floor, and ceiling.

Examples:

 glossy paints as used in home painting, which give also a fraction of specular reflection, while matte paints give almost exclusively diffuse reflection.

Point source illumination:

For oblique surfaces, the illumination decreases by a factor of cos I, where I is the angle between the direction of the light and the direction normal to the surface plane. The angle I is known as angle of incidence.