**Computer Graphics Assignment**

**Submitted by – Sumit Kumar(17/1432)**

**Q.10** Show that parallel lines remain parallel after transformation.

**Ans** : **Transformation** : Changes in orientation, size, and shape are accomplished with geometric transformations that alter the coordinate descriptions of objects. Basic geometric transformations are translation, rotation, and scaling.

**Translation** is applied to an object by repositioning it along a straight-line path from one coordinate location to another. We translate a two-dimensional point by adding **translation distances**, **tx** and **ty**, to the original coordinate position **(x, y)** to move the point to a new position **(x’, y’)**.

**x’ = x + tx** , **y’ = y + ty**

the translation distance pair (tx, ty) is called **translation vector** or **shift vector**.

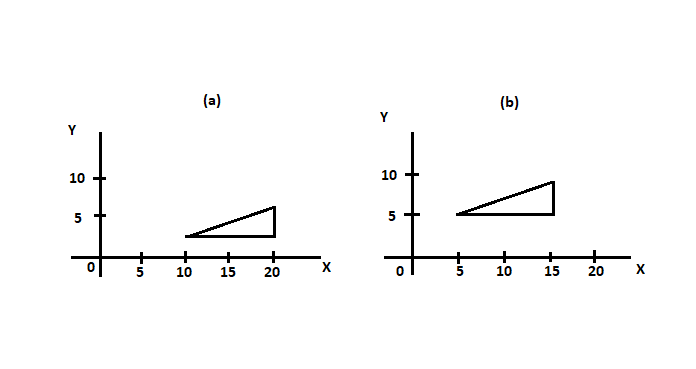
We can express the translation equations as a single matrix equation by using column vectors to represent coordinate positions and the translation vector:

**P = [x1/x2]**, **P’ = [x1’/x2’]**, **T = [tx/ty]**

This allows us to write the two-dimensional translation equations in the matrix form:

**P’ = P + T**

Sometimes matrix-transformation equations are expressed in terms of coordinate row vectors instead of column vectors. In this case, we would write the matrix representation as **P = [x y]** and **T = [tx ty]**.



Translation is a **rigid-body transformation** that moves objects without deformation, i.e., every point on the object is translated by the same amount.

So, its proved that parallel lines remain parallel after transformation.

**Q.12** Explain the working of Cohen -Sutherland line clipping example with example.

**Ans** : This is one of the oldest and most popular line-clipping procedures. In this every line end-point in a picture is assigned a four-digit binary code, is called **region code** that identifies the location of the point relative to boundaries of the clipping rectangle. Regions are set up in reference to the boundaries as shown below ;

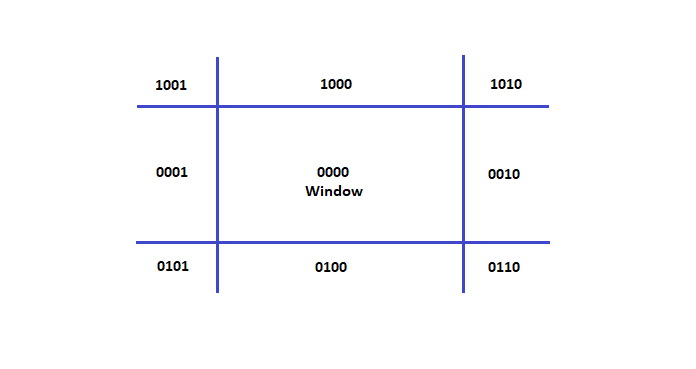
Each bit position in the region code is used to indicate one of the four relative coordinate positions of the point with respect to the clip window: to the left, right, top, or bottom. By numbering the bit positions in the region code as **1** through **4** from right to left, the regions can be correlated with the bit positions as –

**bit 1: left**

**bit 2: right**

**bit 3: below**

**bit 4: above**



A value of **1** in any position indicates that the point is in it that relative position; otherwise, the bit position is set to **0**. If a point is within the clipping rectangle, the region code is **0000**. A point that is below and to the left of the rectangle has region code of **0101**.

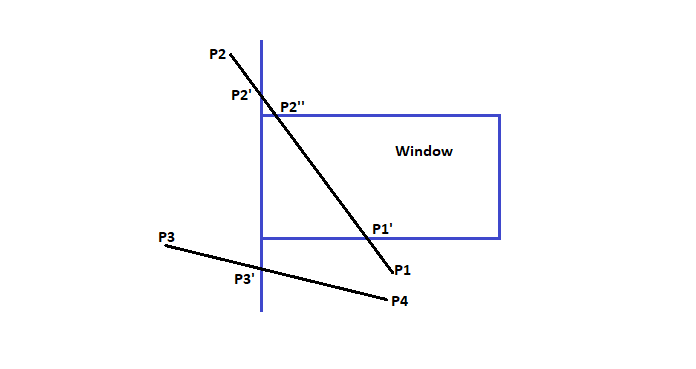
The region-code bit values can be determined by following steps:

1. Calculate differences between endpoint coordinates and clipping boundaries.
2. Use the resultant sign bit of each difference calculation to set the corresponding value in the region code.

* Any lines that are completely contained within the window boundaries have a region code of **0000** for both endpoints, and we trivially accept these lines.
* Any lines that have a **1** in the same bit position in the region codes for each endpoint are completely outside the clipping rectangle, and we trivially reject these lines.
* A method that can be used to test lines for total clipping is to perform the logical and operation with both region codes. If the result is not **0000**, the line is completely outside the clipping region.

To illustrate the specific steps in clipping lines against rectangular boundaries using the Cohen-Sutherland algorithm, we show how the lines in the above diagram could be processed.

Starting with the bottom endpoint of the line from **P1** to **P2**, we check **P1**, against the left, right, and bottom boundaries in turn and find that this point is below the clipping rectangle. We then find the intersection point **P1’** with the bottom boundary and discard the line section from **P1** to **P1’**. The line now has been reduced to the section from **P1’** to **P2**. Since **P2** is outside the clip window, we check this endpoint against the boundaries and find that it is to the left of the window. Intersection point **P2’** is calculated, but this point is above the window. So the final intersection calculation yields **P2’’**, and the line from **P1’** to **P2’’** is saved. The completeness processing for this line, so we save this part and go on to the next line.



Intersection points with a clipping boundary can be calculated using the slope-intersect form of the line equation. For a line with endpoint coordinates **(x1, y1)** and **(x2, y2)**, the **y** coordinate of the intersection point with a vertical boundary can be obtained with the calculation

**y = y1 + m ( x – x1 )**

where the **x** value is set either to **xwmin** or to **xwmax** and the slope of the line is calculated as

**m = ( y2 – y1 ) / ( x1 - x2 ).**

Similarly, if we are looking for the intersection with a horizontal boundary, the **x** coordinate can be calculated as

**x = x1 + ( y – y1 ) / m**

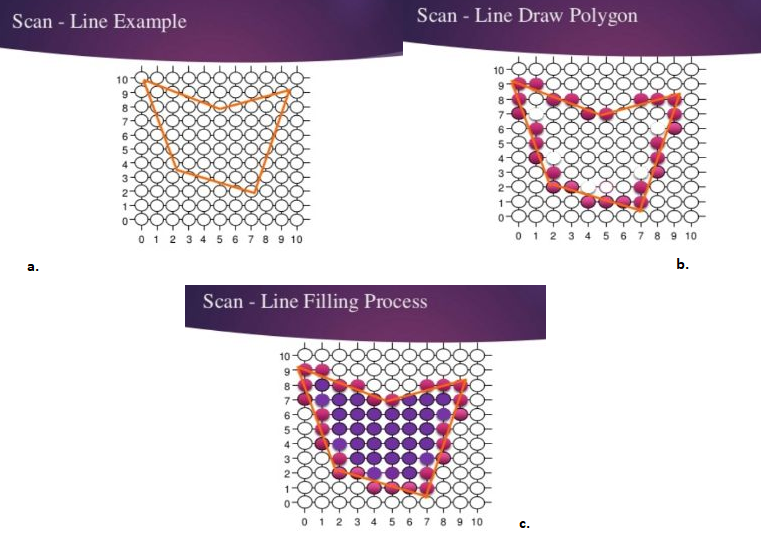
with **y** set either to **ywmin** or to **ywmax**.

**Q.14** Give the structure of Global Edge Table and Active edge table used in scan line fill algorithm.

**Ans** : **Scan-Line Fill algorithm** : Used in Raster Scan Devices. It works as follows –

1. Find intersections of the scan-line with all edges
2. Sort intersections in increasing x
3. Fill all the pixels between pairs of intersections

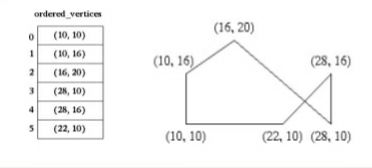
This algorithm include some parameters as GET(Global Edge Table) and AET(Active Edge Table);



Using an example to describe :

Initially each vertices of the polygon is given in the form of (x, y) and is in an ordered array as such:

Unfilled the polygon would look like this to human eye:

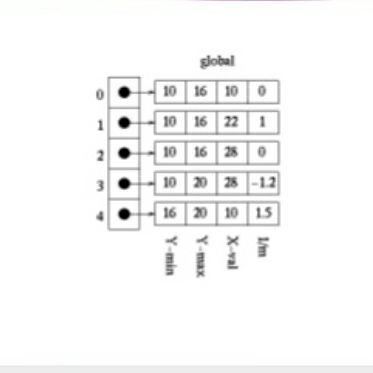


**Global Edge Table : GET** – The global edge table will be used to keep track of the edges that are still needed to complete the polygon.

Edges with the same minimum y values are sorted on minimum x values as follows;

1. Placed the first edge with a slope that is not equal to zero in the global edge table.
2. If the slope of the edge is zero, do not add that edge to the global edge table.
3. Parity is initially set to even. Yet now no edges have crossed.
4. We can safely choose lowest y value in the global edge table as our initial scan-line.

* In our example it is 10.

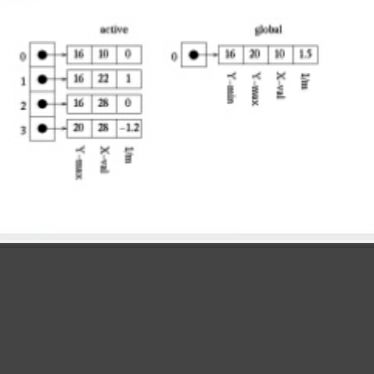


1. **Initializing the Active Edge Table : AET** – The active edge table will be used to keep track of the edges that are intersected by the current scan-line. This should also contain ordered edges.

This is initially set up as follows :

1. The global edge table is ordered on minimum y and x values, search through the global edge table and, for each edge found having a minimum y value equal to current scan-line.
2. Append the edge information in AET for the
3. Maximum y value
4. x value
5. 1/m
6. Do this until an edge is found with a minimum y value greater than the scan line value.

* This active edge table will now contain ordered edges of those edges that are being filled as such :



**Q.18** A clipping window has two opposite vertices lying at (0,0) and (80,40). Use the line end point codes

to determine whether the lines PQ, P(40,20)and Q(70,50) AND RS, R(100,20) and S(120,60) would be

visible, partially visible or totally invisible.

**Ans** : Drawing a rough end-line code chart on paper…..

