**BCA 6th Semester**

**Sub: Computer Graphics**

## Antialiasing

Jaggies appear when an output device does not have a high enough resolution to represent a smooth line correctly. This is also an inherent problem on a computer monitor. The pixels that make up the screen of the monitor are all shaped in rectangles or squares. Because lighting up only half of one of these square pixels is not possible, the result is a jagged line.

The jagged line effect can be minimized by increasing the resolution of the monitor, making the pixels small enough that the human eye cannot distinguish them individually. This is not a good solution, however, because images are displayed based on their resolution. A single image pixel may take up many monitor pixels, making it impossible for a higher resolution monitor to mask the jagged edges. This is where antialiasing is required.

Antialiasing removes jagged edges by adding subtle color changes around the lines, tricking the human eye into thinking that the lines are not jagged. The slight changes in color around the edges of an image help the line blend around curves, giving the impression that the line is true. These color changes are made on a very small scale that the human eye cannot detect under normal circumstances. In order to be able to see that an image has been antialiased, it would have to be magnified.

Antialiasing is often implemented by graphics cards and computer games. Depending on the application, different methods of antialiasing may be applied. This technique is also used in digital photography and digital audio.

## Homogeneous Coordinates

The rotation of a point, straight line or an entire image on the screen, about a point other than origin, is achieved by first moving the image until the point of rotation occupies the origin, then performing rotation, then finally moving the image to its original position.

The moving of an image from one place to another in a straight line is called a translation. A translation may be done by adding or subtracting to each point, the amount, by which picture is required to be shifted.

Translation of point by the change of coordinate cannot be combined with other transformation by using simple matrix application. Such a combination is essential if we wish to rotate an image about a point other than origin by translation, rotation again translation.

To combine these three transformations into a single transformation, homogeneous coordinates are used. In homogeneous coordinate system, two-dimensional coordinate positions (x, y) are represented by triple-coordinates.

Homogeneous coordinates are generally used in design and construction applications. Here we perform translations, rotations, scaling to fit the picture into proper position.

**Example of representing coordinates into a homogeneous coordinate system:** For two-dimensional geometric transformation, we can choose homogeneous parameter h to any non-zero value. For our convenience take it as one. Each two-dimensional position is then represented with homogeneous coordinates (x, y, 1).

**Following are matrix for two-dimensional transformation in homogeneous coordinate:**

**1.** Translation $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$ or $\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$

**2.** Scaling $\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**3.** Rotation (clockwise) $\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**4.** Rotation (anti-clock) $\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**5.** Reflection against X axis $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**6.** Reflection against Y axis $\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**7.** Reflection against origin $\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**8.** Reflection against line Y=X $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**9.** Reflection against Y= –X $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**10.** Shearing in X direction $\begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**11.** Shearing in Y direction $\begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**12.** Shearing in both x and y direction $\begin{bmatrix} 1 & Sh_y & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

## **Composite Transformation:**

A number of transformations or sequence of transformations can be combined into single one called as composition. The resulting matrix is called as composite matrix. The process of combining is called as concatenation.

Suppose we want to perform rotation about an arbitrary point, then we can perform it by the sequence of three transformations

1. Translation
2. Rotation
3. Reverse Translation

The ordering sequence of these numbers of transformations must not be changed. If a matrix is represented in column form, then the composite transformation is performed by multiplying matrix in order from right to left side. The output obtained from the previous matrix is multiplied with the new coming matrix.

Example showing composite transformations:

The enlargement is with respect to center. For this following sequence of transformations will be performed and all will be combined to a single one

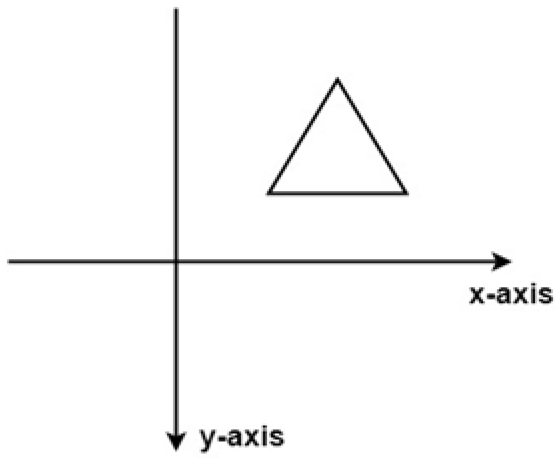
**Step1:** The object is kept at its position as in fig (a)

**Step2:** The object is translated so that its center coincides with the origin as in fig (b)

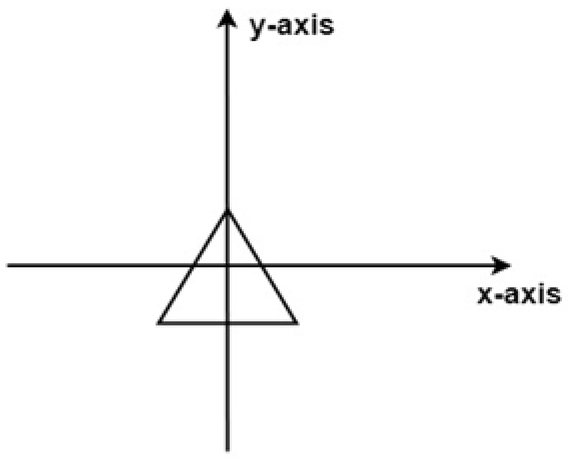
**Step3:** Scaling of an object by keeping the object at origin is done in fig (c)

**Step4:** Again translation is done. This second translation is called a reverse translation. It will position the object at the origin location.

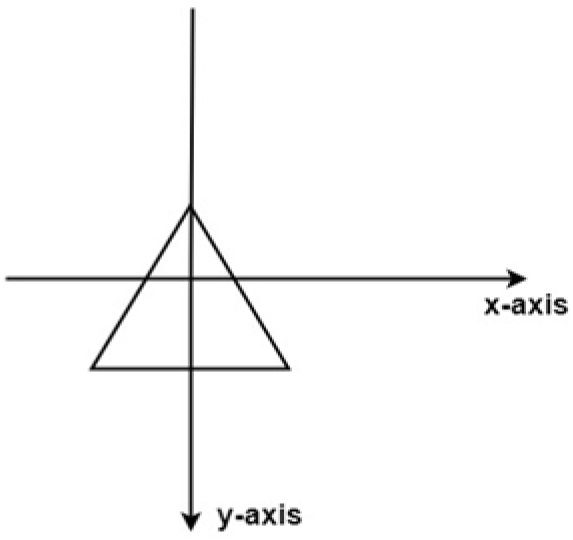
Above transformation can be represented as $T_V.ST_V^{-1}$



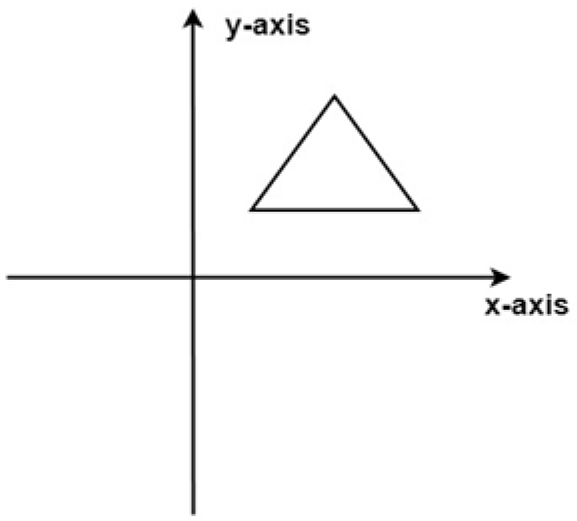
(original position of object)
Fig:(a)



(object is translated to origin)
Fig:(b)



(object is enlarged)
Fig:(c)



(object is retranslated to original position)
Fig:(d)

**Difference between Raster Scan System and Random Scan System**

| **Base of Difference** | **Raster Scan System** | **Random Scan System** |
|---|---|---|
| **Electron Beam** | The electron beam is swept across the screen, one row at a time, from top to bottom. | The electron beam is directed only to the parts of screen where a picture is to be drawn. |
| **Resolution** | Its resolution is poor because raster system in contrast produces zigzag lines that are plotted as discrete point sets. | Its resolution is good because this system produces smooth lines drawings because CRT beam directly follows the line path. |
| **Picture Definition** | Picture definition is stored as a set of intensity values for all screen points, called pixels in a refresh buffer area. | Picture definition is stored as a set of line drawing instructions in a display file. |
| **Realistic Display** | The capability of this system to store intensity values for pixel makes it well suited for the realistic display of scenes contain shadow and color pattern. | These systems are designed for line-drawing and can't display realistic shaded scenes. |
| **Draw an Image** | Screen points/pixels are used to draw an image. | Mathematical functions are used to draw an image. |

## Boundary Fill Algorithm

**Introduction:** Boundary Fill Algorithm starts at a pixel inside the polygon to be filled and paints the interior proceeding outwards towards the boundary. This algorithm works **only if** the color with which the region has to be filled and the color of the boundary of the region are different. If the boundary is of one single color, this approach proceeds outwards pixel by pixel until it hits the boundary of the region.
***Boundary Fill Algorithm is recursive in nature.*** It takes an interior point(x, y), a fill color, and a boundary color as the input. The algorithm starts by checking the color of (x, y). If its color is not equal to the fill color and the boundary color, then

it is painted with the fill color and the function is called for all the neighbors of (x, y). If a point is found to be of fill color or of boundary color, the function does not call its neighbors and returns. This process continues until all points up to the boundary color for the region have been tested.
The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

**4-connected pixels:** After painting a pixel, the function is called for four neighboring points. These are the pixel positions that are right, left, above and below the current pixel. Areas filled by this method are called 4-connected. Below given is the algorithm:
**Algorithm :**

```
void boundaryFill4(int x, int y, int fill_color,int boundary_color)

{

   if(getpixel(x, y) != boundary_color &&

      getpixel(x, y) != fill_color)

   {

       putpixel(x, y, fill_color);

       boundaryFill4(x + 1, y, fill_color, boundary_color);

       boundaryFill4(x, y + 1, fill_color, boundary_color);

       boundaryFill4(x - 1, y, fill_color, boundary_color);

       boundaryFill4(x, y - 1, fill_color, boundary_color);

   }

}
```

## 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.

Suppose at step i, the pixels is $(x_i,y_i)$

The line of equation for step i
$y_i=mx_{i+b}$......................equation 1

Next value will be

$y_{i+1}=m_{xi+1}+b$.................equation 2

$m = \frac{\Delta y}{\Delta x}$

$y_{i+1}-y_i=\Delta y$.......................equation 3

$y_{i+1}-x_i=\Delta x$......................equation 4

$y_{i+1}=y_i+\Delta y$

$\Delta y=m\Delta x$

$y_{i+1}=y_i+m\Delta x$

$\Delta x=\Delta y/m$

$x_{i+1}=x_i+\Delta x$

$x_{i+1}=x_i+\Delta y/m$

## **Case1:** When |M|<1 then (assume that $x_1<x_2$)

x= x1,y=y1 set Δx=1

yi+1=y1+m,     x=x+1

Until x = x2</x

**Case2:** When |M|<1 then (assume that y1<y2)

x= x1,y=y1 set Δy=1

xi+1=$\frac{1}{m}$,     y=y+1

Until y → y2</y

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.

DDA Algorithm:

```
Step1: Start Algorithm

Step2: Declare x1,y1,x2,y2,dx,dy,x,y as integer variables.

Step3: Enter value of x1,y1,x2,y2.

Step4: Calculate dx = x2-x1

Step5: Calculate dy = y2-y1

Step6: If ABS (dx) > ABS (dy)
       Then step = abs (dx)
       Else

Step7: xinc=dx/step
       yinc=dy/step
       assign x = x1
       assign y = y1
```

```
Step8: Set pixel ($x$, $y$)

Step9: $x = x + xinc$
       $y = y + yinc$
       Set pixels (Round ($x$), Round ($y$))

Step10: Repeat step 9 until $x = x2$

Step11: End Algorithm
```