**LABORATORY OVERVIEW**

|  |
| --- |
| **Prerequisites:**  1. Basic Geometry, Trigonometry, Vectors and Matrices  2. Basics of Data Structures and Algorithms |
| **Course Objectives :** |
| 1. Understand the foundations of computer graphics: hardware systems, math basis, light and color. 2. Understand the complexities of modeling realistic objects through modeling complex scenes using a high-level scene description language.  3. Become acquainted with some advanced topics in computer graphics. The student should gain an expanded vocabulary for discussing issues relevant to computer graphics (including both the underlying mathematics and the actual programming).  4. The student should gain an appreciation and understanding of the hardware and software utilized in constructing computer graphics applications.  5. The student should gain a comprehension of windows, clipping and view-ports in relation to images displayed on screen.  6. The student should gain an understanding of geometric, mathematical and algorithmic concepts necessary for programming computer graphics. |
| **Course Outcomes :** |
| On completion of the course, learner will be able to –  CO1: Apply mathematical and logical aspects for developing elementary graphics operations like scan conversion of points, lines, circle, and apply it for problem solving.  CO2: Employ techniques of geometrical transforms to produce, position and manipulate Objects in 2 dimensional and 3-dimensional space respectively.  CO3: Describe mapping from a world coordinates to device coordinates, clipping, and projections in order to produce 3D images on 2D output device.  CO4: Apply concepts of rendering, shading, animation, curves and fractals using computer graphics tools in design, development and testing of 2D, 3D modeling applications.  CO5: Perceive the concepts of virtual reality. |

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**List of Assignments**

**Subject: Computer Graphics Lab SE IT**

**SEM: II**

|  |  |
| --- | --- |
| **No.** | **ASSIGNMENT NAME** |
| 1 | Install and explore the OpenGL |
| 2 | Draw the given pattern using Line Drawing algorithm. |
| 3 | Implement Bresenham circle drawing algorithm to draw any object. The object should be displayed in all the quadrants with respect to center and radius |
| 4 | Implement the following polygon filling methods : i) Flood fill / Seed fill ii) Boundary fill ; using mouse click, keyboard interface and menu driven programming |
| 5 | Implement Cohen Sutherland polygon clipping method to clip the polygon with respect the viewport and window. Use mouse click, keyboard interface |
| 6 | Implement following 2D transformations on the object with respect to axis :  i) Scaling ii) Rotation about arbitrary point iii) Reflection |
| 7 | Generate curve patterns using i) Bezier ii) Koch Curve |
| 8 | Animation : Implement any one of the following animation assignments,  i) Clock with pendulum  ii) National Flag hoisting  iii) Vehicle/boat locomotion |

## Assignment No 1: Install and explore the OpenGL

## AIM: Install and explore the OpenGL

## OBJECTIVE:

To understand functions available in openGL.

## THEORY:

## OpenGL Fundamentals

This section explains some of the concepts inherent in OpenGL.

### Primitives and Commands

OpenGL draws *primitives*—points, line segments, or polygons—subject to several selectable modes. You can control modes independently of each other; that is, setting one mode doesn't affect whether other modes are set (although many modes may interact to determine what eventually ends up in the frame buffer). Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls.

Primitives are defined by a group of one or more *vertices*. A vertex defines a point, an endpoint of a line, or a corner of a polygon where two edges meet. Data (consisting of vertex coordinates, colors, normals, texture coordinates, and edge flags) is associated with a vertex, and each vertex and its associated data are processed independently, in order, and in the same way. The only exception to this rule is if the group of vertices must be *clipped* so that a particular primitive fits within a specified region; in this case, vertex data may be modified and new vertices created. The type of clipping depends on which primitive the group of vertices represents.

Commands are always processed in the order in which they are received, although there may be an indeterminate delay before a command takes effect. This means that each primitive is drawn completely before any subsequent command takes effect. It also means that state-querying commands return data that's consistent with complete execution of all previously issued OpenGL commands.

### Procedural versus Descriptive

OpenGL provides you with fairly direct control over the fundamental operations of two- and three-dimensional graphics. This includes specification of such parameters as transformation matrices, lighting equation coefficients, antialiasing methods, and pixel update operators. However, it doesn't provide you with a means for describing or modeling complex geometric objects. Thus, the OpenGL commands you issue specify how a certain result should be produced (what procedure should be followed) rather than what exactly that result should look like. That is, OpenGL is fundamentally procedural rather than descriptive. Because of this procedural nature, it helps to know how OpenGL works—the order in which it carries out its operations, for example—in order to fully understand how to use it.

### Execution Model

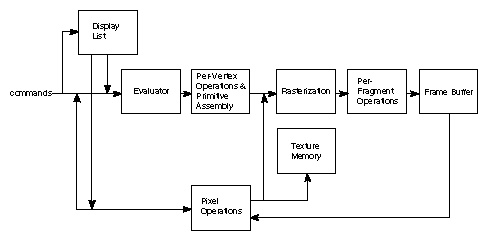
The model for interpretation of OpenGL commands is client-server. An application (the client) issues commands, which are interpreted and processed by OpenGL (the server). The server may or may not operate on the same computer as the client. In this sense, OpenGL is network-transparent. A server can maintain several GL *contexts*, each of which is an encapsulated GL state. A client can connect to any one of these contexts. The required network protocol can be implemented by augmenting an already existing protocol (such as that of the X Window System) or by using an independent protocol. No OpenGL commands are provided for obtaining user input.

The effects of OpenGL commands on the frame buffer are ultimately controlled by the window system that allocates frame buffer resources. The window system determines which portions of the frame buffer OpenGL may access at any given time and communicates to OpenGL how those portions are structured. Therefore, there are no OpenGL commands to configure the frame buffer or initialize OpenGL. Frame buffer configuration is done outside of OpenGL in conjunction with the window system; OpenGL initialization takes place when the window system allocates a window for OpenGL rendering. (GLX, the X extension of the OpenGL interface, provides these capabilities, as described in ["OpenGL Extension to the X Window System."](https://www.glprogramming.com/blue/ch02.html#id86751) )

## Basic OpenGL Operation

The figure shown below gives an abstract, high-level block diagram of how OpenGL processes data. In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages.

**Figure 1-1. OpenGL Block Diagram**



As shown by the first block in the diagram, rather than having all commands proceed immediately through the pipeline, you can choose to accumulate some of them in a *display list* for processing at a later time.

The *evaluator* stage of processing provides an efficient means for approximating curve and surface geometry by evaluating polynomial commands of input values. During the next stage, *per-vertex operations and primitive assembly*, OpenGL processes geometric primitives—points, line segments, and polygons, all of which are described by vertices. Vertices are transformed and lit, and primitives are clipped to the viewport in preparation for the next stage.

*Rasterization* produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each *fragment* so produced is fed into the last stage, *per-fragment operations*, which performs the final operations on the data before it's stored as pixels in the *frame buffer*. These operations include conditional updates to the frame buffer based on incoming and previously stored z-values (for z-buffering) and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values.

Input data can be in the form of pixels rather than vertices. Such data, which might describe an image for use in texture mapping, skips the first stage of processing described above and instead is processed as pixels, in the *pixel operations* stage. The result of this stage is either stored as *texture memory*, for use in the rasterization stage, or rasterized and the resulting fragments merged into the frame buffer just as if they were generated from geometric data.

All elements of OpenGL state, including the contents of the texture memory and even of the frame buffer, can be obtained by an OpenGL application.

OpenGL (Open Graphics Library) is a cross-platform, hardware-accelerated, language-independent, industrial standard API for producing 3D (including 2D) graphics. Modern computers have dedicated GPU (Graphics Processing Unit) with its own memory to speed up graphics rendering. OpenGL is the software interface to graphics hardware. In other words, OpenGL graphic rendering commands issued by your applications could be directed to the graphic hardware and accelerated.

We use 3 sets of libraries in our OpenGL programs:

1. **Core OpenGL (GL)**: consists of hundreds of commands, which begin with a prefix "gl" (e.g., glColor, glVertex, glTranslate, glRotate). The Core OpenGL models an object via a set of geometric primitives such as point, line and polygon.
2. **OpenGL Utility Library (GLU)**: built on-top of the core OpenGL to provide important utilities (such as setting camera view and projection) and more building models (such as qradric surfaces and polygon tessellation). GLU commands start with a prefix "glu" (e.g., gluLookAt, gluPerspective).
3. **OpenGL Utilities Toolkit (GLUT)**: OpenGL is designed to be independent of the windowing system or operating system. GLUT is needed to interact with the Operating System (such as creating a window, handling key and mouse inputs); it also provides more building models (such as sphere and torus). GLUT commands start with a prefix of "glut" (e.g., glutCreatewindow, glutMouseFunc). GLUT is platform independent, which is built on top of platform-specific OpenGL extension such as GLX for X Window System, WGL for Microsoft Window, and AGL, CGL or Cocoa for Mac OS.  
   Quoting from the [opengl.org](http://www.opengl.org/resources/libraries/glut/): "GLUT is designed for constructing small to medium sized OpenGL programs. While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits. *GLUT is simple, easy, and small.*"  
   Alternative of GLUT includes SDL, ....

## Assignment No 2: Draw the given pattern using Line Drawing algorithm

**AIM:**

Draw the given pattern using DDA & Bresanham Line Drawing algorithm.

## OBJECTIVE:

To understand functions available in openGL.

Understand DDA Line drawing algorithm.

Understand Bresenham’s line drawing algorithm

## THEORY:

**DDA Line Drawing:**

It is Digital Differential Analyzer (DDA).

It is also called as **Vector generation line drawing algorithm.**

Here we solve the differential equation for straight line. For drawing a line we need to turn ON the pixels which are on the line segment. For this we consider the slope of line. For simplicity we divide the line segment in two types.

## DDA’s line generation algorithm:

The DDA is a scan conversion line algorithm based on calculating Dy and Dx. We sample the line at unit intervals in one co-ordinate and determine corresponding integer values nearest the line path for the other co-ordinate.

1. We will consider a line with positive slope. If the slope is less than or equal to 1,we sample it at unit x intervals (Dx = 1) and compute each successive y valueas

**yk+1 = yk + m**

Subscript k takes integer values starting from 1 for the first point and increases by 1 until the final end point is reached. The value of m can be any real number between 0 and 1.

1. For lines with positive slope greater than 1, we reverse the roles of x and y. We sample at unit y intervals and calculate each succeeding x valueas

**xk+1 = xk + 1/m**

1. For the above equation we are processing the line equation from the left end point to right end point. If this processing is reversed , so either wehave

**Dx = -1 and yk+1 = yk – Or Dy = -1 and xk+1 = xk –1/m**

## Algorithm:

* 1. Input the two line endpoints and store the left endpoint in(xa,ya).
  2. Plot the first point(xa,ya).
  3. Calculate constants from the two end points such as Dx and Dy for horizontal and verticaldifferences.
  4. The difference with the greater magnitude determines the value of parameter steps.
  5. We determine the offset needed at each step to generate the next pixel position along the linepath.
  6. We loop through this process stepstimes.
  7. If the magnitude of Dxis greater than the magnitude of Dy and xa<xb, the values of increments in the x and y directions are 1 and m respectively and if the magnitude of Dxis greater than Dy and xa>xb , the values of decrements in the x and y directions are -1 and –mrespectively.

## Bresenham's Line Drawing Algorithm:

1. Input the two line endpoints and store the left endpointin(x0,yo)
2. Load (x0,yo) into the frame buffer; that is , plot the firstpoint.
3. Calculate constants Δx, Δy,2 Δy and 2 Ay -2 Δx , and obtain thestartingvalue

forthe decision parameter as:

Po = 2 Δy - Δx

1. At each xk, the next point the line , starting at k=0, perform the followingtest:

If pk< 0, the next point to plot is (xk + 1 ,yk) and pk+1 = pk + 2 Δy

Otherwise,the next point to plot is (xk + 1, yk +1) and pk+1 = pk + 2 Δy-2 Δx

1. Repeat step 4 Δxtimes.
2. Input the two line endpoints and store the left endpointin(x0,yo)
3. Load (x0,yo) into the frame buffer; that is , plot the firstpoint.
4. Calculate constants Δx, Δy,2 Δy and 2 Ay -2 Δx , and obtain thestartingvalue

forthe decision parameter as:Po = 2 Δy - Δx

1. At each xk, the next point the line , starting at k=0, perform the followingtest:

If pk<0 , the next point to plot is (xk+ 1 ,yk) and pk+1 = pk + 2 Δy

Otherwise ,the next point to plot is (xk + 1, yk +1) and pk+1 = pk + 2 Δy-2 Δx

1. Repeat step 4 Δxtimes.

## INPUT:

1. Enter two points.
2. Enter radius and centre ofcircle

## OUTPUT:

## FAQ:

1. What are the different GraphicsStandards?
2. What are the characteristics of displayadapter
3. Differentiate between raster and randomscan.
4. What ispixel?
5. What is the importance of using framebuffer?
6. Is it dynamic storagestructure?

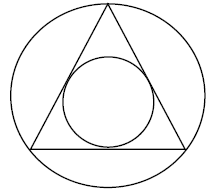
## PRACTICE ASSIGNMENTS:

1. Draw a Christmas tree using the DDAalgorithm
2. Draw a Christmas tree using the Bresenham’salgorithm

## Assignment No 3: Draw circle using circle generation algorithm.

**AIM:**

Implement Bresenham circle drawing algorithm to draw any object. The object should be displayed in all the quadrants with respect to center and radius



## OBJECTIVE:

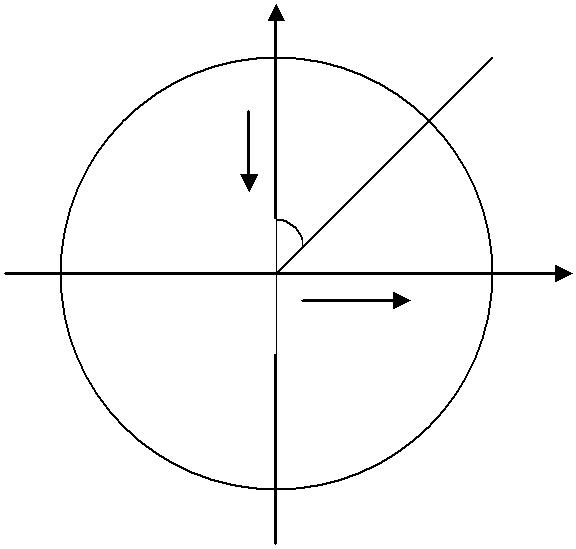
Understand Bresenham’s Circle algorithm. Further using these algorithms to draw real time pictures

## THEORY:

**Bresenham’s Circle Drawing:**

A circle is a symmetrical figure. It has eight - way symmetry. If we know any singlepoint of circle we can plot all remaining seven pixels using 8- waysymmetry.This algorithm considers 8 –way symmetry of circle and generates the whole circle.1/8thpartofcirclei.e.from90°to45°isdrawn,duringthisxincrementsinpositive

direction and y increments in negative direction. In this algorithm we have to select proper pixel which is either on the circle or closed to the circle port.



y

-y

90°

0

45°

+x

x

Decision Variable is given as d = 3 -2r

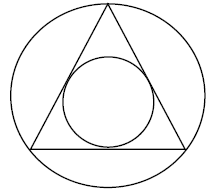
## Algorithm:

1. Set initial values of (xc, yc) and (x, y)
2. Set decision parameter d to d = 3 – (2 \* r).
3. calldrawCircle(int xc, intyc, int x, int y) function.
4. Repeat steps 5 to 8 until x < = y
5. Increment value of x.
6. If d < 0, set d = d + (4\*x) + 6
7. Else, set d = d + 4 \* (x – y) + 10 and decrement y by 1.
8. call drawCircle(int xc, intyc, int x, int y) function

## INPUT:

Enter two points.

## OUTPUT:

****

**FAQ:**

What is antaliasing?

What is Bresanham circle generation algorithm?

What is rastarization?

Any other circle generation algorithm?

## Assignment No 4: Implement program to draw 4 \* 4 Chessboard rotated by 45 degree and filled with black square.

**AIM:**

Implement the following polygon filling methods : i) Flood fill / Seed fill ii) Boundary fill ; using mouse click, keyboard interface and menu driven programming

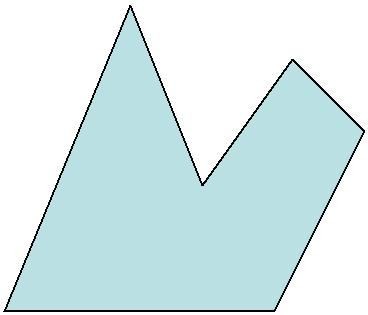
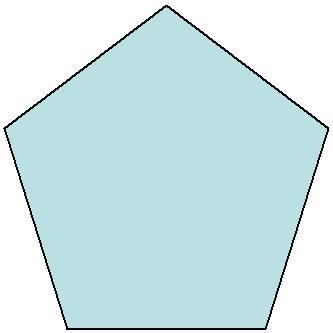
## OBJECTIVE:

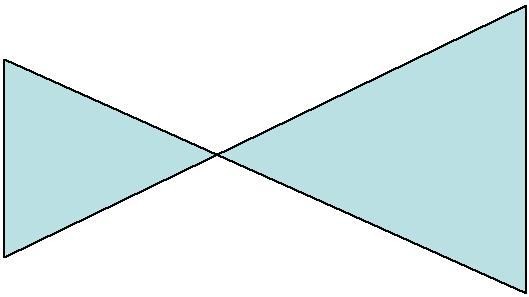
Understand polygon fill algorithms and transformation.

## THEORY:

*Different types of Polygons*

* Simple Convex
* Simple Concave
* Non-simple :self-intersecting
* Withholes





## Polygon Filling Approaches:

* 1. Scan Line fill approaches

Determine overlap intervals for scan lines that cross the area. This approach is mostly used in general graphics packages

e.g. Scanline fill algorithm

* 1. Seed fillapproaches

Start from interior point & paint outward until boundary conditions are encountered. This approach is used in applications having complex boundaries and interactive paintingsystems

e.g. Boundary Fill & Flood Fill algorithm

**Flood Fill Algorithm**

voidFloodFill4(int x, int y, color newcolor, color oldColor)

{

if(ReadPixel(x, y) == oldColor)

{

FloodFill4(x+1, y, newcolor,oldColor);

FloodFill4(x-1, y, newcolor,oldColor);

FloodFill4(x, y+1, newcolor,oldColor);

FloodFill4(x, y-1, newcolor,oldColor);

}

}

**Boundary Fill Algorithm**

void boundaryFill4(int x, int y, intfill\_color,intboundary\_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);

}

}

## INPUT:

Coordinates of vertices of polygon

## OUTPUT:

Properly filled polygon

## FAQ:

1. Explain concave and convexpolygons?
2. Explain different tests used for checking whether pixel is inside or outside the polygon.
3. Explain different ways of filling thepolygon.
4. What is meant by Active Edge Table and Global EdgeTable.

## PRACTICE ASSIGNMENTS:

Draw different objects such as house and apply following algorithms for filling the objects.

* 1. Scan line polygon fillingalgorithm
  2. Flood fill polygon filling algorithm(Recursive and nonrecursive)
  3. Boundary fill polygon filling algorithm(Recursive and non recursive)

## Assignment No 5: 2D Polygon Clipping

**AIM:**

Implement Cohen Sutherland line clipping method to clip the line with respect the viewport

and window. Use mouse click, keyboard interface

## OBJECTIVE:

Understand and Implement basic 2D clipping in Laboratory.

## THEORY:

1. **Line Clipping – Cohen Sutherland**

In computer graphics, '***line clipping'*** is the process of removing lines or portions of lines outside of an area of interest. Typically, any line or part thereof which is outside of the viewing area is removed.

The Cohen–Sutherland algorithm is a computer graphics algorithm used for line clipping. The algorithm divides a two-dimensional space into 9 regions (or a three-dimensional space into 27 regions), and then efficiently determines the lines and portions of lines that are visible in the center region of interest (the viewport).

The algorithm was developed in 1967 during flight simulator work by Danny Cohen and Ivan Sutherland

The design stage includes, excludes or partially includes the line based on where:

* + Both endpoints are in the viewport region (bitwise OR of endpoints == 0): trivial accept.
  + Both endpoints share at least one non-visible region which implies that the line does not cross the visible region. (bitwise AND of endpoints != 0): trivialreject.
  + Both endpoints are in different regions: In case of this nontrivial situation the algorithm finds one of the two points that is outside the viewport region (there will be at least one point outside). The intersection of the outpoint and extended viewport border is then calculated (i.e. with the parametric equation for the line) and this new point replaces the outpoint. The algorithm repeats until a trivial accept or rejectoccurs.

The numbers in the figure below are called outcodes. The outcodeis computed for each of the two points in the line. The outcode will have four bits for two-dimensional clipping, or six bits in the three-dimensional case. The first bit is set to 1 if the point is above the viewport. The bits in the 2D outcode represent: Top, Bottom, Right, Left. For example the outcode 1010 represents a point that is top-right of the viewport. Note that the outcodes for endpoints **must** be recalculated on each iteration after the clipping occurs.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1001 1000 1010 | | |
| 0001 | 0000 | 0010 |
| 0101 | 0100 | 0110 |

# Algorithm:

1. Create a class sulc with functions drawwindow, drawline, setcode, visibility and reset endpoint.
2. Using the function line set the parameters to drawwindow.
3. Using the function defined in class sulc, setcodeis used to save the line inside the window and to the line outside thewindow.
4. Using the functionvisibility
   1. Check the code to know the points inside or outside thewindow.
   2. If the code value is zero the point is inside thewindow.
5. Using the function reset endpoint
   1. If the code value for the line is outside thewindow.
   2. Reset the endpoint to the boundary of thewindow.
6. Initialize the graphicsfunctions
7. Declare the variables x1, x2, y1, y2 of arraytype.
8. Get the value of two endpoints x1, y1 and x2, y2 to draw theline.
9. Using the object c, display the window beforeclipping.
10. Using the function setcode, visibility display the clipped window only with lines inside the window class was displayed afterclipping.

## INPUT:

Enter an object of as line and polygon

## OUTPUT:

Output object should undergo above mention line as well as polygon clipping.

## FAQ:

1. What isclipping?
2. What do you mean by interior and exteriorclipping?
3. Explain how exterior clipping is useful in multiple windowenvironments

## PRACTICE ASSIGNMENTS:

1. Draw a polygon for any vertex performclipping.

## Assignment NO 6: Write aprogram toimplement 2-D transformations on Equilateral Triangle/Rhombus

**AIM:**

Write a program to implement following 2D transformation Translation, Scaling, Rotation and Shear.

## OBJECTIVE:

Understand and Implement basic 2D transformations in Laboratory.

## THEORY:

**2D transformation:**

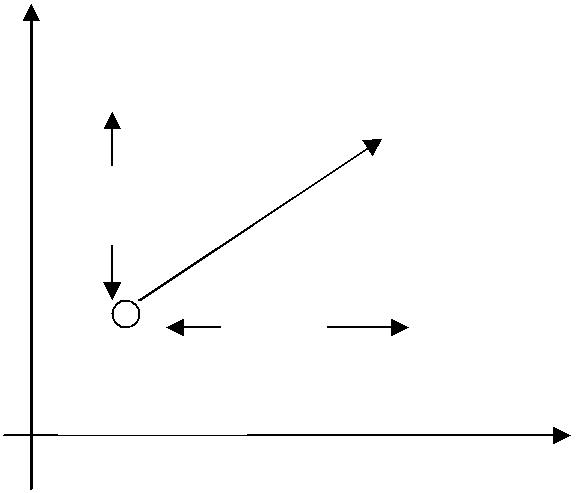
Transformation allows us to uniformly alter the entire picture instead of drawing a new picture.

Transformation means translation, scaling, rotation, shearing or combination of all.

## Translation:

Translation is applied to an object by repositioning it along a straight line path from one coordinate location to another. This can be easily done by adding to each point the amount by which we want to shift thepoint.

x’ = x + tx



P(x’, y’)

ty

tx

P(x, y)

y’ = y + ty

TranslationMatrix: 1 0 0

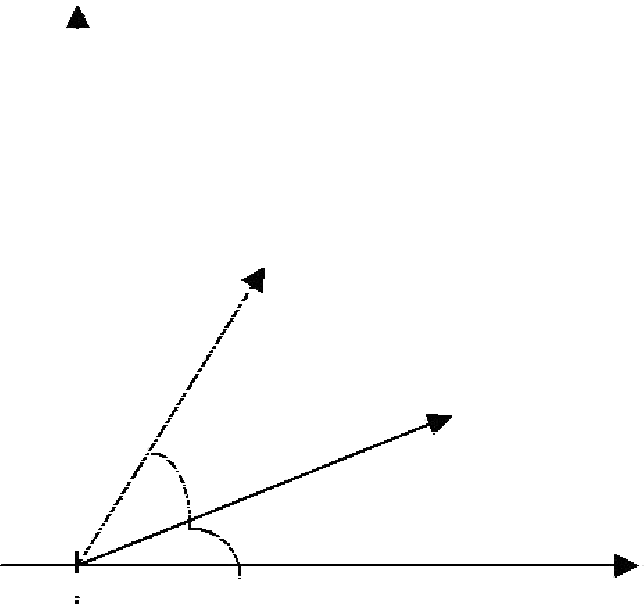
T= 0 1 0

tx ty 1

## Rotation:

The 2D rotation is applied to object by repositioning it along circular path in xy. To generate rotation we specify a rotation angle and position (xr ,yr) of rotation point.

x ‘ = x cos θ - y sin θ y’ = x sin θ + y cos θ



P(x’, y’)

r

P(x, y)

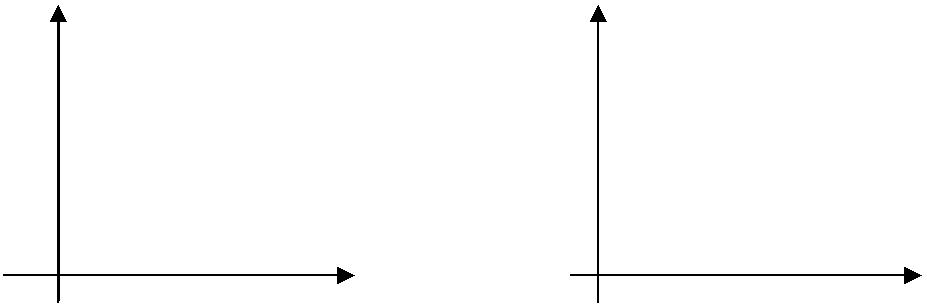
θ

r

|  |  |  |  |
| --- | --- | --- | --- |
| Φ |  | | |
| Rotation Matrix : | R = | cos θ  -sin θ  0 | sinθ 0  cos θ 0  0 1 |
| **Scaling:** |  |  |  |

Scaling transformation alters the size of an object. This operation can be carried out for polygon by multiplying coordinates valued (x, y) of each vertex by scaling factors Sx and Sy to produce the transformed coordinates

BeforeScaling After Scaling



x’ = x . Sx

y’ = y . Sy

Scaling Matrix: Sx 0 0

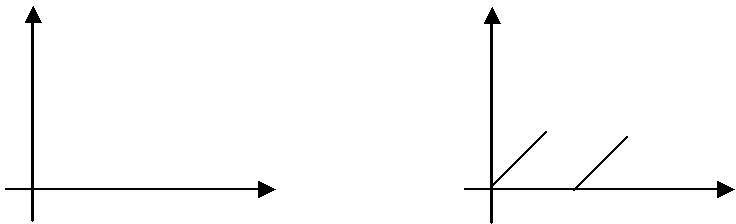
S= 0 Sy 0

0 0 1

## Shearing:

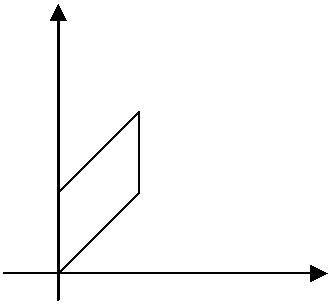
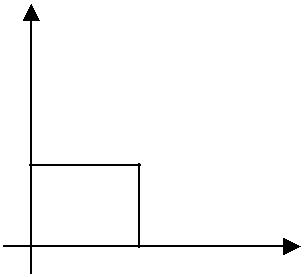
A transformation that slants the shape of an object is called the shear transformation.

X Shear:



OriginalObject Object after xshear

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 0 | 0 |  |
| X\_sh = | Shx | 1 |  | 0 |
|  | 0 | 0 | 1 |  |
| Y Shear: |  |  |  |  |



OriginalObject Object after yshear

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 | Shy | 0 |
| Y\_sh = | 0 | 1 | 0 |
|  | 0 | 0 | 1 |

## INPUT:

Enter an object of any shape.

## OUTPUT:

Output object should undergo above mention transformations.

## FAQ:

1. What is homogeneous co-ordinatesystem

## PRACTICE ASSIGNMENTS:

1. Draw a house and perform all thetransformations.
2. Draw a doll and perform rotation andtranslation.

**Assignment NO 7: Generate Fractal Patterns.**

**AIM:**

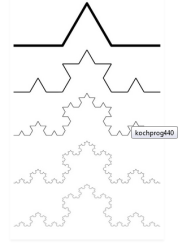
Generate fractal patterns using i) Bezier ii) Koch Curve.

## OBJECTIVE:

Understand and Implement different curve generation techniques in Laboratory.

## THEORY:

## The Koch snowflake (also known as the Koch curve, Koch star, or Koch island) is a mathematical curve.



**Step1:**

Draw an equilateral triangle. You can draw it with a compass or protractor, or just eyeball it if you don’t want to spend too much time drawing the snowflake.

**Step2:**

Divide each side in three equal parts. This is why it is handy to have the sides divisible by three.

**Step3:**

Draw an equilateral triangle on each middle part. Measure the length of the middle third to know the length of the sides of these new triangles.

**Step4:**

Divide each outer side into thirds. You can see the 2nd generation of triangles covers a bit of the first. These three line segments shouldn’t be parted in three.

**Step5:**

Draw an equilateral triangle on each middle part.

## OUTPUT:

Objects such as flower, waves

## FAQ:

1. Explain different types ofcurves.
2. Explain different curve generation techniques.

## Assignment No 8: Implement program for Animation

**AIM:**

Implement anyone of the following Animation

1. Clock with pendulum
2. National Flag Hoisting
3. Vehicle or Boat Locomotion

## OBJECTIVE:

Understand animation in Laboratory.

## THEORY:

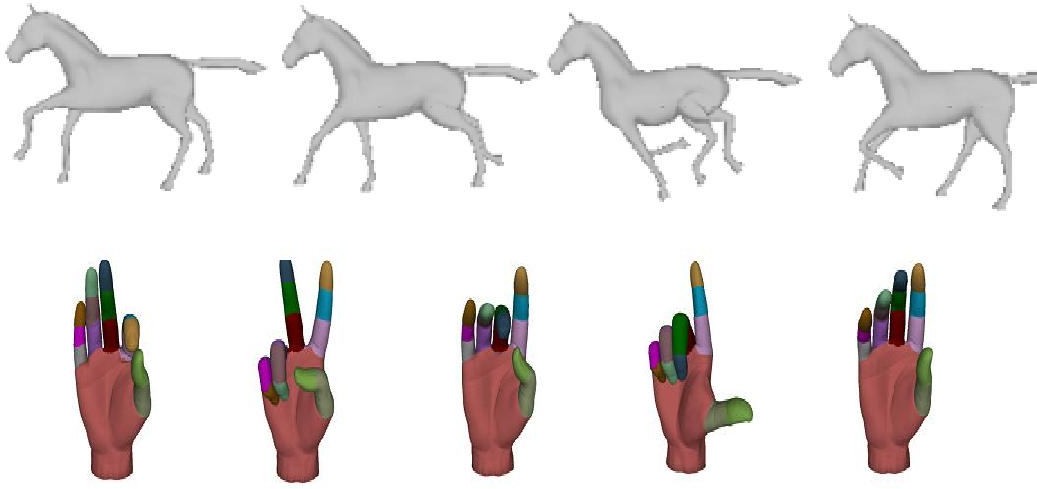
**Animation**: Animation is the rapid display of a sequence of static images and/or objects to create an illusion of movement. The most common method of presenting animation is as a motion picture or video program, although there are other methods. This type of presentation is usually accomplished with a camera and a projector or a computer viewing screen which can rapidly cycle through images in a sequence. Animation can be made with either hand rendered art, computer generated imagery, or three-dimensional objects, e.g., puppets or clay figures, or a combination of techniques. The position of each object in any particular image relates to the position of that object in the previous and following images so that the objects each appear to fluidly move independently of one another. The viewing device displays these images in rapid succession, usually 24, 25, or 30 frames per second.

**Segmentation**: Segmentation Models of dynamic visual processes can be used to partition the spatio-temporal domain of a video sequence into regions, each of which is considered to be homogeneous with respect to some spatio-temporal property. Addressing this very important problem is useful not only for content-based video retrieval, but also for other tasks like robot navigation, video surveillance, restoration, editing, andcompression.

Due to the speed and visual complexity of animation, learners may confused and overwhelmed throughout animation viewing as they were unable to properly process all incoming information before it disappears . Therefore, designs that do not provide appropriate time for learners to focus their attention on the information being presented may be among the reason for failure of animation in assisting learning Segmented- animation with features that allow learners to control the segment viewing rather than passively view the whole animation continuously can be a design solution

**Segmentation in Animation:** In segmented-animation design, the whole animation will be chunked into meaningful segments, including pause or time break after each segment and learner-control features to move from segment to segment. Thus, pause or time break between segments and learner-control features will allow learners to rehearse in order to extract necessary information from one segment before moving to the following segment. In addition, during the pauses learners can analyze the visual spatial structure of the content on the screen, something that can be difficult to do when a display continuously changes. Research findings indicates that segmented-animation with learner-control features will allow appropriate exposure duration on animation that helps learners in interpreting and understanding the animation better . Comparison studieson

segmented-animation and continuous-animation also revealed that segmented-animation appears to be more beneficial in enhancing students learning performance.



## Input:

Different Images

## Output:

Animated Sequence using the openGL.