2022-23

ज्ञानं सविज्ञानम्

DPES

Dhole Patil Education Society

DHOLE PATIL COLLEGE OF ENGINEERIG

KHARADI PUNE 412207

COMPUTER GRAPHICS LABMANUAL

Prof. Fulsaundar Priyanka

DHOLE PATIL COLLEGE OF ENGINEERING

COMPUTER GRAPHICS LAB
(CGL)

LAB MANUAL

DEPARTMENT OF INFORMATION TECHNOLOGY

(ACADEMIC YEAR - 2022-23)

DPCOE-KHARADI PUNE

COMPUTER GRAPHICS LAB

(CGL) LAB MANUAL

Author:	**Prof. Fulsaundar Priyanka** **(IT Dept., DPCOE Kharadi , Pune – 412207)**
Creation Date:	**29th DEC 2019**
Last Updated:	**14th DEC 2022**
Version:	**1.2**

DEPARTMENT OF INFORMATION TECHNOLOGY

VISION:

To build versatile human resources in Information Technology professionally competent and capable of functioning in global environment for secure and seamless services to society.

MISSION:

M2: To Explore IT Innovations through Collaborative learning and Partnerships with Institutions and Industries.

M3: To evolve startups and strive for holistic development of stakeholders.

M4: To strengthen institutional ethical responsibility for needs of the society.

M5: To augment IT skills through continue education program for lifelong learning.

Course structure

Teaching Scheme	Credits	Examination Scheme
Practical :2 Hours/Week	01	Practical : 25 Marks

Prerequisites

1. Basic Geometry, Trigonometry, Vectors and Matrices
2. Basics of Data Structures and Algorithms

Course Objectives

1. To acquaint the learners with the concepts of OpenGL
2. To acquaint the learners with the basic concepts of Computer Graphics
3. To implement the various algorithms for generating and rendering the objects
4. To get familiar with mathematics behind the transformations
5. To understand and apply various methods and techniques regarding animation

Course Outcomes

CO1: Apply and implement line drawing and circle drawing algorithms to draw the objects.
CO2: Apply and implement polygon filling methods for the object
CO3: Apply and implement polygon clipping algorithms for the object
CO4: Apply and implement the 2D transformations on the object
CO5: Implement the curve generation algorithms
CO6: Demonstrate the animation of any object using animation principles

Guidelines for Faculty's Lab Journal

1. Student should submit term work in the form of handwritten journal based on specified list of assignments.

2. Practical and Oral Examination will be based on all the assignments in the lab manual

3. Candidate is expected to know the theory involved in the experiment.

4. The practical examination should be conducted if and only if the journal of the candidate is complete in all respects.

Guidelines for Lab /TW Assessment

1. Examiners will assess the student based on performance of students considering the parameters such as timely conduction of practical assignment, methodology adopted for implementation of practical assignment, timely submission of assignment in the form of handwritten write-up along with results of implemented assignment, attendance etc.

2. Examiners will judge the understanding of the practical performed in the examination by asking some questions related to theory & implementation of experiments he/she has carried out.

3. Appropriate knowledge of usage of software and hardware related to respective laboratory should be checked by the concerned faculty member.

Reference books

1. S. Harrington, “Computer Graphics”, 2nd Edition, McGraw-Hill Publications, 1987, ISBN 0-07-100472-6

2. D. Rogers, “Procedural Elements For Computer Graphics”, 2nd Edition, McGraw-Hill Publications, 1987, ISBN 0-07-047371-4

3. F.S. Hill JR, “Computer Graphics Using OpenGL”, Pearson Education

ASSIGNMENT 1

AIM : Install and Explore the OPENGL.

We need a C/C++ compiler, either GCC (GNU Compiler Collection) from MinGW or Cygwin (for Windows), or Visual C/C++ Compiler, or others.

OBJECTIVE:

We need the following sets of libraries in programming OpenGL:

1. **Core OpenGL (GL)**: consists of hundreds of functions, 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 and more building models (such as qradric surfaces). GLU functions start with a prefix "glu" (e.g., gluLookAt, gluPerspective)

3. **OpenGL Utilities Toolkit (GLUT)**: provides support to interact with the Operating System (such as creating a window, handling key and mouse inputs); and more building models (such as sphere and torus). GLUT functions start with a prefix of "glut" (e.g., glutCreatewindow, glutMouseFunc).

   Quoting from the opengl.org: "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, ..

4. **OpenGL Extension Wrangler Library (GLEW):**

"GLEW is a cross-platform open-source C/C++ extension loading library. GLEW provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform." Source and pre-build binary available at http://glew.sourceforge.net/.

Each of the software package consists of:

1. A *header* file: "gl.h" for core OpenGL, "glu.h" for GLU, and "glut.h" (or "freeglut.h") for GLUT, typically kept under "include\GL" directory.

2. A *static library*: for example, in Win32, "libopengl32.a" for core OpenGL, "libglu32.a" for GLU, "libglut32.a" (or "libfreeglut.a" or "glut32.lib") for GLUT, typically kept under "lib" directory.

3. An optional *shared library*: for example, "glut32.dll" (for "freeglut.dll") for GLUT under Win32, typically kept under "bin" or "c:\windows\system32".

It is important to locate the *directory path* and the *actual filename* of these header files and libraries in your operating platform in order to properly setup the OpenGL programming environment.

Eclipse CDT with Cygwin or MinGW:

1.1 Installing Eclipse CDT / Cygwin or MinGW, OpenGL, GLU and GLUT

Step 1: Setup the Eclipse CDT (C Development Toolkit)
Read "How to install Eclipse CDT".

Step 2: Setup a GCC Compiler

We could use either MinGW or Cygwin.

- **MinGW:** For MinGW, we need to install GLUT separately. Download freeglut (@ http://freeglut.sourceforge.net/index.php). I recommend using the pre-package version for MinGW (freeglut 2.8.0 MinGW Package) available at http://www.transmissionzero.co.uk/software/freeglut-devel/.

  Download, unzip and copy header files from "include\GL" to "<MINGW_HOME>\include\GL"; the libraries from "lib" to "<MINGW_HOME>\lib", and shared library from "bin" to "<MINGW_HOME>\bin" (which should be included in the PATH environment variable), where <MINGW HOME> is the MinGW installed directory.

  Take note of the headers and libraries:

  1. Headers: the OpenGL header "gl.h", GLU header "glu.h" and GLUT header "glut.h" (or "freeglut.h") are kept in "<MINGW_HOME>\include\GL" directory. Since "<MINGW_HOME>\include" is in the implicit *include-path*. We can include the headers as <GL/glut.h>, <GL/glt.h>, and <GL/gl.h>.
  2. Libraries: the OpenGL library "libopengl32.a", GLU library "libglu32.a" and GLUT library "libfreeglut.a" are kept in "<MINGW_HOME>\lib" directory. This directory is in the implicit *library-path*.
  3. Nonetheless, we need to include these libraries in linking. They shall be referred to as "opengl32", "glu32", "freeglut" without the prefix "lib" and suffix ".a".

  (Alternatively, you could download Nate Robin's original Win32 port of GLUT from @ http://www.xmission.com/~nate/glut.html, which has not been updated since 2001. Download, unzip and copy "glut.h" to "<MINGW_HOME>\include\GL", "glut32.lib" to "<MINGW_HOME>\lib", and "glut32.dll" to "<MINGW_HOME>\bin" (which should be included in the PATH))

- **Cygwin:** We need to install "gcc", "g++", "gdb", "make" (under the "Devel" category) and "opengl", "freeglut" (under the "Graphics" category).

  1. Headers: the OpenGL header "gl.h", GLU header "glu.h", and GLUT header "glut.h" are provided in the "<CYGWIN_HOME>\usr\include\w32api\GL" directory. As "<CYGWIN_HOME>\usr\include\w32api" is in the implicit *include-path*. We can include the headers as <GL/glut.h>, <GL/glt.h>, and <GL/gl.h>.

  2. Libraries: the OpenGL library "libopengl32.a", GLU library "libglu32.a" and GLUT library "libglut32.a" are provided in the "<CYGWIN_HOME>\lib\w32api" directory. This directory is in the implicit *library-path*.

  3. Nonetheless, we need to include these libraries in linking. They shall be referred to as "opengl32", "glu32", "glut32" without the prefix "lib" and suffix ".a".

Step 3: Configuring the Include-Path, Lib-Path and Library: We can configure on per-project basis by right-click on the project ⇒ Properties ⇒ C/C++ general ⇒ Paths and Symbols ⇒ Use "Includes" panel to configure the Include-Path; "Library Paths" panel for the Lib-Path; and "Libraries" panel for individual libraries. We will do this later.
On command-line (for GCC), we could use option -I<dir> for include-path, -L<dir> for lib-path, and -l<lib> for library.

1.2 Writing Your First OpenGL Program

1. Launch Eclipse.
2. Create a new C++ project: Select "File" menu ⇒ New ⇒ Project... ⇒ C/C++ ⇒ C++ Project ⇒ Next.
   In "Project name", enter "Hello" ⇒ In "Project type", select "Executable", "Empty Project" ⇒ In "Toolchain", select "Cygwin GCC" or "MinGW GCC" (depending on your setup) ⇒ Next ⇒ Finish.
3. Create a new Source file: Right-click on the project node ⇒ New ⇒ Other... ⇒ C/C++ ⇒ Source file ⇒ Next.
   In "Source file", enter "GL01Hello.cpp" ⇒ Finish.
4. In the editor panel for "GL01Hello.cpp", type the following source codes:

   NOTE: For Windows, you should include "windows.h" header before the OpenGL headers.

```
/*
 * GL01Hello.cpp: Test OpenGL C/C++ Setup
 */
#include <windows.h> // For MS Windows
#include <GL/glut.h> // GLUT, includes glu.h and gl.h

/* Handler for window-repaint event. Call back when the window first appears and
   whenever the window needs to be re-painted. */
void display() {
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black and opaque
   glClear(GL_COLOR_BUFFER_BIT);         // Clear the color buffer

   // Draw a Red 1x1 Square centered at origin
```

Configuring the "include-paths", "library-paths" and "libraries":

Right-click on the project ⇒ Property ⇒ C/C++ general ⇒ Paths and Symbols.
Open the "Libraries" tab ⇒ Add ⇒ Enter "glut32" (Cygwin) or "freeglut" (MinGW with freeglut) ⇒ Add ⇒ Enter "glu32" ⇒ Add ⇒ Enter "opengl32".

There is no need to configure the "include-paths" and "library-paths", as they are implicitly defined.

Build (right-click on the project node ⇒ "Build Project") and Run (right-click on the project node ⇒ Run As ⇒ Local C/C++ Application).

Conclusion: Hence we are able to implement and Write a program in OPENGL platform.

ASSIGNMENT : 2

AIM : Implement DDA and Bresenham line drawing algorithm to draw
I) Simple line
ii) Dotted line
iii) Dashed line
iv) Solid line
using mouse interface. Divide the screen in four quadrants with center as (0, 0). The line should work for all the slopes +ve, -ve, >1,<1

OBJECTIVE:

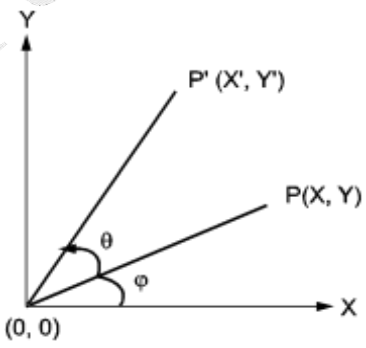
1. Understand bresenham line drawing algorithm.
2. Understand 2 D transformations
3. Understand seed fill algorithm
4. Further using these algorithms to draw real time pictures
5. To Learn openGL functions

OUTCOME:

1. Apply and implement line drawing and circle drawing algorithms to draw specific shape given in theproblem.
2. Apply and implement polygon filling algorithm for a given polygon

Problem Statement:
Implement DDA and Bresenham line drawing algorithm to draw:
i) Simple Line ii) Dotted Line iii) Dashed Line iv) Solid line
using mouse interface Divide the screen in four quadrants with center as (0, 0). The line should work for all the slopes positive as well as negative.

Program:



ASSIGNMENT : 3

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

PROBLEM STATEMENT : Draw inscribed and Circumscribed circles in the triangle as shown as an example below (Use any Circle drawing and Line drawing algorithms)

OBJECTIVE:

1. Understand bresenham circle drawing algorithm.
2. Further using these algorithms to draw real time pictures
3. To Learn openGL functions

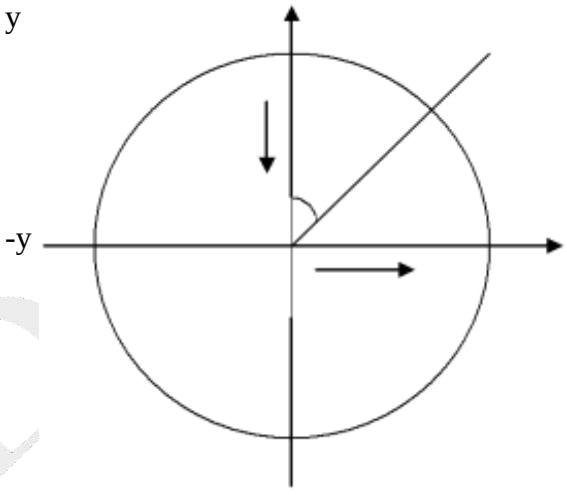
OUTCOME:

Apply and implement line drawing and circle drawing algorithms to draw specific shape given in the problem.

THEORY:

Bresenham's Circle Drawing:

A circle is a symmetrical figure. It has eight - way symmetry. If we know any single point of circle we can plot all remaining seven pixels using 8- way symmetry. This algorithm considers 8 –way symmetry of circle and generates the whole circle. $1/8^{th}$ part of circle i.e. from 90° to 45° is drawn, during this x increments in positive 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.



Decision Variable is given as d = 3 -2r

Both of these algorithms uses the key feature of circle that it is highly symmetric. So, for whole 360 degree of circle we will divide it in 8-parts each octant of 45 degree. In order to do that we will use Bresenham's Circle Algorithm for calculation of the locations of the pixels in the first octant of 45 degrees. It assumes that the circle is centered on the origin. So for every pixel (x, y) it calculates, we draw a pixel in each of the 8 octants of the circle as shown below :

Now, we will see how to calculate the next pixel location from a previously known pixel location (x, y). In Bresenham's algorithm at any point (x, y) we have two option either to choose the next pixel in the east i.e. (x+1, y) or in the south east i.e. (x+1, y-1).

And this can be decided by using the decision parameter d as:

- If d > 0, then (x+1, y-1) is to be chosen as the next pixel as it will be closer to the arc.
- else (x+1, y) is to be chosen as next pixel.

Now to draw the circle for a given radius 'r' and centre (xc, yc) We will start from (0, r) and move in first quadrant till x=y (i.e. 45 degree). We should start from listed initial condition:

d = 3 - (2 * r)

x = 0

y = r

Now for each pixel, we will do the following operations:

1. Set initial values of (xc, yc) and (x, y)
2. Set decision parameter d to d = 3 – (2 * r).
3. call drawCircle(int xc, int yc, 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, int yc, int x, int y) function

CONCLUSION: Hence we are able to implement algorithm for circle drawing.

ASSIGNMENT: 4

AIM : IMPLEMENT THE FOLLOWING POLYGON FILLING METHODS (1 WEEK, 2 HRS)
i) Flood fill / Seed fill
ii) Boundary fill

OBJECTIVE :

1) The code contains algorithm to draw polygon using co-ordinate system and in-built line drawing functio .

2) Both seedfill and floodfill have similar process of color filling, only the difference is in function call and conditions provided

THEORY :

1) How recursive functions work

2) getpixel() and putpixel() in your own framework

3) Difference in boundary fill and seed fill.

1) For drawing polygon:

To fill a polygon, we should first draw a polygon.

Here we are taking (x,y) inputs in an array and drawing line for each consecutive pair of points

Finally, we complete the polygon using joining the first and last point of the polygon

Note: If you know the equivalent polygon drawing version using mouse events to draw polygon using mouse, you can use that instead.

2) For seed fill:

Some people also call this as seed fill algorithm (pixels scatter like seeds of grass)

The ideology is simple.

Condition: check if the color is similar to old color

condition true: put new color, go for surrounding pixels

condition false: return

This is a recursive function

This is the function to fill "bucket" in "paint" app

The surrounding pixels are 4 pixels in direction north, south, east, west to the current pixel

3) For boundary fill:

Some people also call this as block fill algorithm (pixels scatter like seeds of grass)

The ideology is different than seed fill. Regardless of old pixel value, one fills the whole polygon based until boundary not reached Condition: check if the color not new color and check if color is not boundary color condition true: put new color, go for surrounding pixels condition false: return This is a recursive function The surrounding pixels are 4 pixels in direction north, south, east, west to the current pixel.

PROGRAM :

CONCLUSION: Implement the following polygon filling methods (1 week, 2 hrs)
i) Flood fill / Seed fill ii) Boundary fill

ASSIGNMENT: 5

AIM: Implement Cohen Sutherland polygon clipping method to clip the polygon with respect the view-port and window.

THEORY: 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): trivial reject.
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 reject occurs.
The numbers in the figure below are called outcodes. The outcode is 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

Sutherland Hodgman Polygon Clipping: -
It is used for clipping polygons. It works by extending each line of the convex *clip polygon* in turn and selecting only vertices from the *subject polygon* those are on the visible side.
An algThe algorithm begins with an input list of all vertices in the subject polygon. Next, one side of the clip polygon is extended infinitely in both directions, and the path of the subject polygon is traversed. Vertices from the input list are inserted into an output list if they lie on the visible side of the extended clip polygon line, and new vertices are added to the output list where the subject polygon path crosses the extended clip polygon line.
This process is repeated iteratively for each clip polygon side, using the output list from one stage as the input list for the next. Once all sides of the clip polygon have been processed, the final generated list of vertices defines a new single polygon that is entirely visible. Note that if the subject polygon was concave at vertices outside the clipping polygon, the new polygon may have coincident (i.e. overlapping) edges – this is acceptable for rendering, but not for other applications such as computing shadows.
Clipping ping polygons would seem to be quite complex. A single polygon can actually be split into multiple polygons .The Sutherland-Hodgman algorithm clips a polygon against all edges of the clipping region in turn. The algorithm steps from vertex to vertex, adding 0, 1, or 2 vertices to the output list at each step.
The Sutherland-Hodgeman Polygon-Clipping Algorithms clips a given polygon successively against the edges of the given clip-rectangle. These clip edges are denoted with e1, e2, e3, and e4, here. The closed polygon is represented by a list of its verteces (v1 to vn; Since we got 15 verteces in the example shown above, vn = v15.).

Clipping is computed successively for each edge. The output list of the previous clipping run is used as the inputlist for the next clipping run. 1st run: Clip edge: e1; inputlist = {v1, v2, ..., v14, v15}, the given polygon

CONCLUSION : In this way we studied the Cohen Sutherland polygon clipping and Line Clipping algorithm.

FAQ :

1.What is clipping?
2.What do you mean by interior and exterior clipping?
3.Explain how exterior clipping is useful in multiple window environments
4 The Sutherland-Hodgeman algorithm can be used to clip lines against a non rectangular boundary.
5.What uses might this have?
6.What modifications to the algorithm would be necessary? What restrictions would apply to the shape of clipping region?
7. Explain why Sutherland-Hodgeman algorithm works only for convex clipping regions?

ASSIGNMENT : 6

AIM : IMPLEMENT FOLLOWING 2D TRANSFORMATIONS ON THE OBJECT WITH RESPECT TO AXIS

a. Scaling b. Rotation about arbitrary point c. Reflection

OBJECTIVE :

It is a process of changing the angle of the object. Rotation can be clockwise or anticlockwise. For rotation, we have to specify the angle of rotation and rotation point. Rotation point is also called a pivot point. It is print about which object is rotated.

TYPES OF ROTATION:

1. Anticlockwise
2. Counterclockwise

The positive value of the pivot point (rotation angle) rotates an object in a counter-clockwise (anti-clockwise) direction.

The negative value of the pivot point (rotation angle) rotates an object in a clockwise direction.

When the object is rotated, then every point of the object is rotated by the same angle.

Straight Line: Straight Line is rotated by the endpoints with the same angle and redrawing the line between new endpoints.

Polygon: Polygon is rotated by shifting every vertex using the same rotational angle.

Curved Lines: Curved Lines are rotated by repositioning of all points and drawing of the curve at new positions.

Circle: It can be obtained by center position by the specified angle.

Ellipse: Its rotation can be obtained by rotating major and minor axis of an ellipse by the desired angle.

Matrix for rotation is a clockwise direction.

Matrix for rotation is an anticlockwise direction.

Matrix for homogeneous co-ordinate rotation (clockwise)

Matrix for homogeneous co-ordinate rotation (anticlockwise)

Rotation about an arbitrary point: If we want to rotate an object or point about an arbitrary point, first of all, we translate the point about which we want to rotate to the origin. Then rotate point or object about the origin, and at the end, we again translate it to the original place. We get rotation about an arbitrary point.

Example: The point (x, y) is to be rotated

The (x_c y_c) is a point about which counterclockwise rotation is done

Step1: Translate point (x_c y_c) to origin

Step2: Rotation of (x, y) about the origin

Step3: Translation of center of rotation back to its original position

Example1: Prove that 2D rotations about the origin are commutative i.e. R_1 R_2=R_2 R_1.

Solution: R_1 and $R_2$are rotation matrices

Example2: Rotate a line CD whose endpoints are (3, 4) and (12, 15) about origin through a 45° anticlockwise direction.

Solution: The point C (3, 4)

Example3: Rotate line AB whose endpoints are A (2, 5) and B (6, 12) about origin through a 30° clockwise direction.

Solution: For rotation in the clockwise direction. The matrix is

Step1: Rotation of point A (2, 5). Take angle 30°

Step2: Rotation of point B (6, 12)

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 $T_V.ST_V^{-1}$

CONCLUSION : I studied 2d Composite transformation having 2D Translation Scaling, Rotation.

ASSIGNMENT : 07

AIM: GENERATE FRACTAL PATTERNS USING I) BEZIER II) KOCH CURVE

OBJECTIVE:

Bezier Curves: 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 Beziercurve 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.

Koch Curve :

Fractals are geometric objects. Many real-world objects like ferns are shaped like fractals. Fractals are formed by iterations. Fractals are self-similar.

In computer graphics, we use fractal functions to create complex object

The object representations uses Euclidean-geometry methods; that is, object shapes were described with equations. These methods are adequate for describing manufactured objects: those that have smooth surfaces and regular shapes. But natural objects, such as mountains and clouds, have irregular or fragmented features, and Euclidean methods do not realistically model these objects. Natural objects can be realistically described with fractal-geometry methods, where procedures rather than equations are used to model objects.

In computer graphics, fractal methods are used to generate displays of natural objects and visualizations

The self-similarity properties of an object can take different forms, depending on the choice of fractal representation.

In computer graphics, we use fractal functions to create complex objects

A mountain outlined against the sky continues to have the same jagged shape as we view it from a

closer and closer. We can describe the amount of variation in the object detail with a number called the fractal dimension.

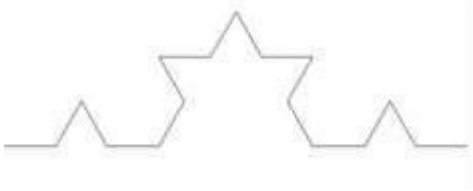
ALGORITHM:

1. The Koch snowflake can be constructed by starting with an equilateral triangle, then recursivelyaltering each line segment as follows:

2. Divide the line segment into three segments of equal length.

I. Draw an equilateral triangle that has the middle segment from step 1 as its base and pointsoutward.

II. Remove the line segment that is the base of the triangle from step 2.



III. After one iteration of this process, the resulting shape is the outline of a hexagram.

3. The Koch snowflake is the limit approached as the above steps are followed over and over again.

4. The Koch curve originally described by Koch is constructed with only one of the three sides of theoriginal triangle.

5. In other words, three Koch curves make a Koch snowflake.

Examples: In graphics applications, fractal representations are used to model terrain, clouds, water, trees and other plants, feathers, fur, and various surface textures, and just to make pretty patterns. In other disciplines, fractal patterns have been found in the distribution of stars, river islands, and moon craters; in rain fields; in stock market variations; in music; in traffic flow; in urban property utilization; and in the boundaries of convergence regions for numerical- analysis techniques

- Add Some Randomness:
- The fractals we've produced so far seem to be very regular and "artificial".
- To create some realism and variability, simply change the angles slightly sometimes based on a random number generator.
- For example, you can curve some of the ferns to one side.
- For example, you can also vary the lengths of the branches and the branching factor.

Terrain (Random Mid-point Displacement):

- Given the heights of two end-points, generate a height at the mid-point.
- Suppose that the two end-points are a and b. Suppose the height is in the y direction, such that the height at a is y(a), and the height at b is y(b).
- Then, the height at the mid-point will be:

  $y_{mid} = (y(a)+y(b))/2 + r$, where

  r is the random offset
- This is how to generate the random offset r:

  $r = sr_g|b-a|$, where

  s is a user-selected "roughness" factor, and

  r_g is a Gaussian random variable with mean 0 and variance 1

CODE FOR BEZIER CURVES :

CODE FOR KOCH CURVE :

```
#include <iostream>
#include <math.h>
#include <time.h>
#include <GL/glut.h>

using namespace std;

double x,y,len,angle;
int it;
```

```
void init(){ glClearColor(1.0,1.0,1.0,0.0);
glMatrixMode(GL_PROJECTION);
gluOrtho2D(0,640,0,480);
    glClear(GL_COLOR_BUFFER_BIT);
  }

  void line1(int x1, int y11, int x2,int y2){

    glColor3f(0,1,0);
    glBegin(GL_LINES);
      glVertex2i(x1,y11);
      glVertex2i(x2,y2);
    glEnd();
    glFlush();

  }

  void k_curve(double x, double y, double len, double angle, int it){

    if(it>0){

      len                              /=3;
      k_curve(x,y,len,angle,(it-1));
      x += (len * cosl(angle * (M_PI)/180));
      y += (len * sinl(angle * (M_PI)/180));
      k_curve(x,y, len, angle+60,(it-1));
      x += (len * cosl((angle + 60) * (M_PI)/180));
      y += (len * sinl((angle + 60) * (M_PI)/180));
      k_curve(x,y, len, angle-60,(it-1));
      x += (len * cosl((angle - 60) * (M_PI)/180));
      y += (len * sinl((angle - 60) * (M_PI)/180));
      k_curve(x,y,len,angle,(it-1));
    }
    else
    {
      line1(x,y,(int)(x + len * cosl(angle * (M_PI)/180) + 0.5),(int)(y + len * sinl(angle * (M_PI)/180) + 0.5));
    }
  }

  void Algorithm()
  {
    k_curve(x,y,len,angle,it);     }

  int main(int argc, char** argv){

    cout<<"\n Enter Starting Point x space y ";
    cin>>x>>y;

    cout <<"\n Lenght of line and space angle of line";
    cin>>len>>angle;

    cout<<"\n No. of ittration ";
    cin>>it;

    glutInit(&argc, argv);
```

```
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(640,480);
    glutInitWindowPosition(200,200);
    glutCreateWindow("Koch");
    init();
    glutDisplayFunc(Algorithm);

    glutMainLoop();
    return 0;
}
```

CONCLUSION: Thus we have to studied Different kind of curve and its implementation

ASSIGNMENT : 08

AIM: IMPLEMENT ANIMATION PRINCIPLES FOR ANY OBJECT

OBJECTIVES:

Animation is defined as a series of images rapidly changing to create an illusion of movement. We replace the previous image with a new image which is a little bit shifted. Animation Industry is having a huge market nowadays. To make an efficacious animation there are some principles to be followed.
Principle of Animation:

THEORY:

There are 12 major principles for an effective and easy to communicate animation.
Squash and Strech:
This principle works over the physical properties that are expected to change in any process. Ensuring proper squash and stretch makes our animation more convincing.
For Example: When we drop a ball from height, there is a change in its physical property. When the ball touches the surface, it bends slightly which should be depicted in animation properly.
Anticipation:
Anticipation works on action.Animation has broadly divided into 3 phases:

1. Preparation phase
2. Movement phase
3. Finish

In Anticipation, we make our audience prepare for action. It helps to make our animation look more realistic.
For Example: Before hitting the ball through the bat, the actions of batsman comes under anticipation. This are those actions in which the batsman prepares for hitting the ball.
Arcs:
In Reality, humans and animals move in arcs. Introducing the concept of arcs will increase the realism. This principle of animation helps us to implement the realism through projectile motion also.
For Example, The movement of the hand of bowler comes under projectile motion while doing bowling.
Slow in-Slow out:
While performing animation, one should always keep in mind that in reality object takes time to accelerate and slow down. To make our animation look realistic, we should always focus on its slow in and slow out proportion.
For Example, It takes time for a vehicle to accelerate when it is started and similarly when it stops it takes time.
Appeal:
Animation should be appealing to the audience and must be easy to understand. The syntax or font style used should be easily understood and appealing to the audience. Lack of symmetry and complicated design of character should be avoided.
Timing:
Velocity with which object is moving effects animation a lot. The speed should be handled with care in case of animation.
For Example, An fast-moving object can show an energetic person while a slow-moving object can symbolize a lethargic person. The number of frames used in a slowly moving object is less as compared to the fast-moving object.
3D Effect:
By giving 3D effects we can make our animation more convincing and effective. In 3D Effect, we convert our object in a 3-dimensional plane i.e., X-Y-Z plane which improves the realism of the object.
For Example, a square can give a 2D effect but cube can give a 3D effect which appears more realistic.
8. Exaggeration:
Exaggeration deals with the physical features and emotions. In Animation, we represent emotions and feeling in exaggerated form to make it more realistic. If there is more than one element in a scene then it is necessary to make a balance between various exaggerated elements to avoid conflicts.

Staging:

Staging is defined as the presentation of the primary idea, mood or action. It should always be in presentable and easy to manner. The purpose of defining principle is to avoid unnecessary details and focus on important features only. The primary idea should always be clear and unambiguous.
Secondary Action:
Secondary actions are more important than primary action as they represent the animation as a whole. Secondary actions support the primary or main idea.
For Example, A person drinking a hot tea, then his facial expressions, movement of hands, etc comes under the secondary actions.

Follow Through:

It refers to the action which continues to move even after the completion of action. This type of action helps in the generation of more idealistic animations.
For Example: Even after throwing a ball, the movement of hands continues.

Overlap:

It deals with the nature in which before ending the first action, the second action starts.
For Example: Consider a situation when we are drinking Tea from the right hand and holding a sandwich in the left hand. While drinking a tea, our left-hand start showing movement towards the mouth which shows the interference of the second action before the end of the first action.
Reference of Theory: https://www.geeksforgeeks.org/principles-of-animation/

CODE:

CONCLUSION : Thus we have to studied the implement animation principles for any object