ZEAL
INSTITUTES
www.zealeducation.com

Assignment :- 1

Problem Statement:

Install and explore the OpenGL.

Objective:

To Understand basic concepts of OpenGL & its installation.

Outcome:

To implement basic concept of OpenGL.

Theory:

Q.1] What is OpenGL?

→ OpenGL is a library of function calls for doing computer graphics. The current applications that render high-quality color images composed of 3D geometric objects & images. The current version of OpenGL is 4.5 released on August 11 2014, and is eightieth revision since original version 1.0.

Software Organisation in OpenGL.

The openGL API is window & operating system independent. That means that part of application that draws can be platform independent. Generally, that is controlled by windowing system on whatever platform you're working on.

Software Organisation:

application program

OpenGL Motif widget or similar

GLUT

GLX, AGL or WGL

X-Win32 Mac O/S

GLU

GL

software and/or hardware.

The OpenGL Interface consists of functions in three libraries:

i). OpenGL core lib: OpenGL 32 on Windows and GL on most unix / Linux systems (libGL.a)

ii). OpenGL Utility lib. (GLU): Provides func. in OpenGL core but avoids having to rewrite code.

iii). Links and Window System: GLX for X window systems, WGL for Windows, AGL for Macintosh.

Open GL Utility Toolkit (GLUT)

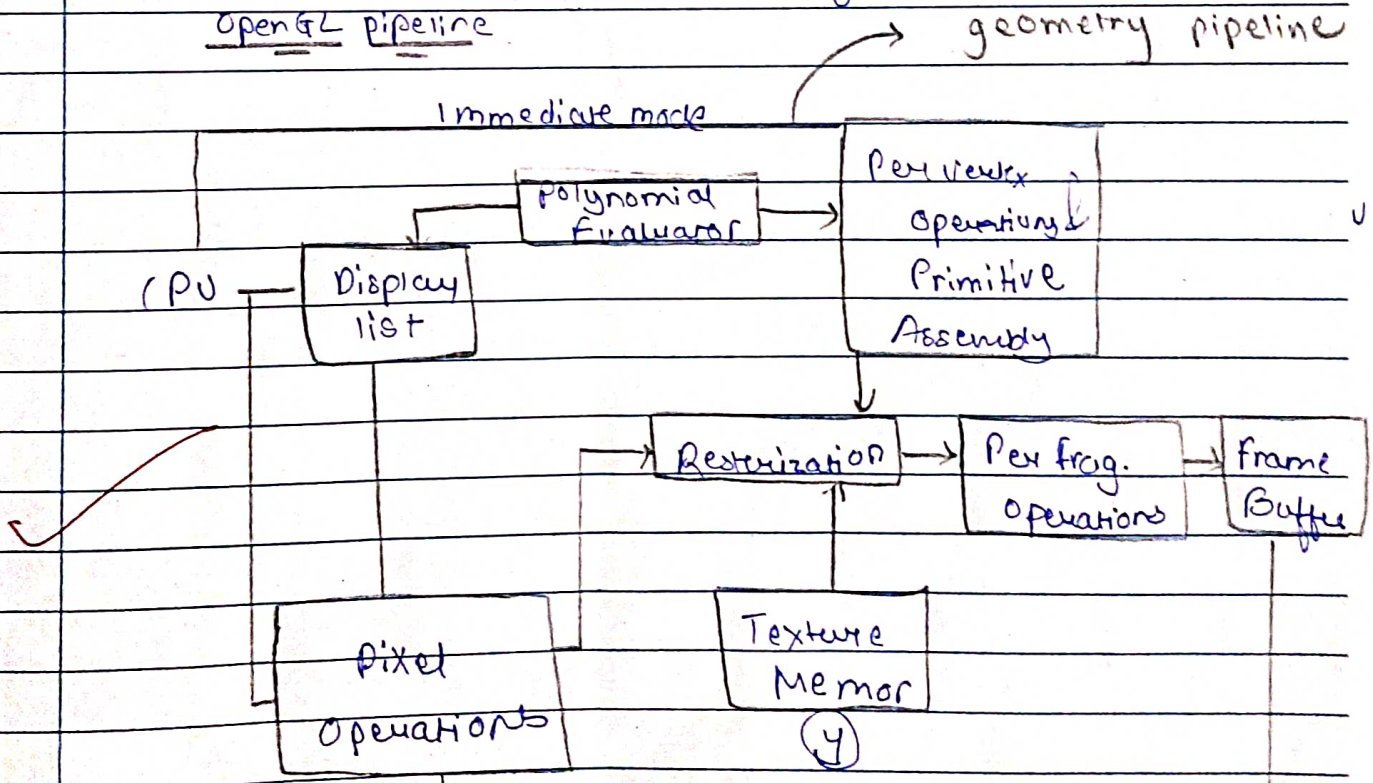
- GLUT provides functionality common to all window system such as opening a window, getting input from mouse & keyboard, menus. GLUT is event-driven & its code is portable.
- GLUT is designed for constructing small to medium sized OpenGL programs.
- So large applications requiring sophisticated user interfaces are better off using native window system toolkits. The current version is 3.7. The toolkit supports multiple windows for OpenGL, rendering, callback driven event processing

Sophisticated input devices an 'idle' routing & binary a simple, cascading pop-up menu facility, utility routines to generate various solid & wire frame objects support for bitmap & stroke Fonts.

++ OpenGL pipeline Architecture:

- Generally data flows from an application through GPU to generate an image in frame buffer. The application will provide vertices, which are collections of data that are composed to form geometric objects to the OpenGL pipeline.
- The other shading stages like tesselation & geometry shadding are also used for vertex processing.

OpenGL pipeline



Primitive & Attributes:

	Open GL Primitive	Descriptive	Total vertices for primitive
1)	GL_POINTS	Render a single pt. per vertex (pts. may be larger than a single pixel)	n
2)	GL_LINES	Connect each pair of vertices with a single line segment	2n
3)	GL_LINE_STRIP	Connect each successive vertex to previous one with line seg.	n+1
4)	GL_LINE_LOOP	Connect all vertices in a loop of line seg.	n
5)	GL_TRIANGLES	Render a triangle for each triple of vertices	3n
6)	GL_TRIANGLE_FAN	Create Δ's by using first vertex in list, pairs of successive vertices	n+2

++ Commonly Required OpenGL functions:

- void glVertex[2/3/4][sifd](TYPE xcoordinate, Type coordinate, ...)
- void glVertex[2/3/4]v(TYPE * coordinate)

Specifies position of vertex in 2, 3 or 4 dimensions. The coordinates can be specified as short s, int i, float f, or double d. The current

normal, texture coordinates, fog coordinate are associated with vertex when glVertex is called.

- Delimits the vertices or a primitive or a group of like primitives.

Mode: Specifies primitive or primitives that will be created from vertices presented betw. glBegin and subsequent glEnd. Ten symbolic constants are accepted.

- GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_QUADS, and GL_POLYGON void glEnd() terminates a list of vertices.

void glRect[sifd](TYPE x1, TYPE y1, TYPE x2, TYPE y2)
void glRect[sifd]v(TYPE * v1, TYPE * v2)

- Specify a 2D axis aligned rectangle by x,y coordinates of diagonally opposite vertices (or pointers to the vertices) using standard data types.

b). Specify color :

- void glColor[34][bifdubusui](TYPE red, TYPE green, TYPE blue, TYPE alpha)
- void glColor[34][bifdubusui]v(TYPE *color)

- void glClearColour (GLclampf r, GLCLampf g, GL Clampf a)

– Specified red, green, blue & alpha values.

- void glIndexi[sifd ub] (TYPE index)

– Set the current colour index.

++ void glutSetColor (int cell, GLfloat red, GL floatgreen,
Cell: Colour. Cell index: GL float blue);
(Starting at zero)

– Sets the color index colormap entry of current window's logical colormap for the layer in use with the colour specified by red, green and blue

++ void glPointSize (GL float size);

Size: Specifies the diameter of rasterized points. The initial value is 1. glPointSize specifies rasterized diameter of both aliased & antialiased points

– To enable and disable point antialiasing, call glEnable and glDisable with argument

– GL-POINT_SMOOTH. Point antialiasing is initially disabled.

++ WORKING WITH WINDOW SYSTEM:

– Different GL implementations buffer commands in several diff. locations, including network buffers and the graphics accelerator itself

Interaction :

1) void glutMouseFunc (void (*func)(int button, int state, int x, int y)).

2) void glutReshapeFunc (void (*func)(int width, int height));

3) void glutKeyboardFunc (void (*func)(unsigned char key, int x, int y));

4) void glutIdleFunc(void (*func)(void));

5) void glutTimerFunc(unsigned int msecs, void (*func)(int value), value);

Transformations :

1) void glMatrixMode (GLenum mode);

2) void glPushMatrix (void);
   void glPopMatrix (void);

3) void glRotate[dF] (GLDouble angle, GLDouble x, GLDouble y, GLDouble z);

4) void glTranslate[dF](GLDouble x, GLDouble y, GLDouble z);

void glLoadMatrixF (const GLfloat m);

Viewing :

1). void glOrtho (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble nearVal, GLdouble farVal);

2). void gluOrtho2D (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);

– define a 2D orthographic projection matrix.

3) void gluLookAt (GLdouble eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ);

4). void gluPerspective (GLDouble fovy, GLDouble aspect, GLDouble zNear, GLDouble zFar);

– The aspect ratio is the ratio of x (width) to y (height) zNear specifies the distance from the viewer to the near clipping plane (away positive) zFar specifies the distance from the viewer to the far clipping plane (always positive).

OpenGL Installation ;

> Open Terminal by using following command
Ctrl + Alt + T

> Go to root directory by using following command
su –

> Then press enter
Enter password

> Run the following commands to install OpenGL.
$sudo apt-get update
$sudo apt-get install lib glul-mesa-dev freeglut-3-dev mesa-common-dev

> Now to test ;
if openGL libraries are working fine on our linux , we will create a C++ program and test it.
- For creating Program type $ gedit on Terminal Then save the program in root directory .cpp extension.

> Now give command (to compile);
→ $ g++ programme.cpp -o pathnfilename -lglut -lGLU -lGL.

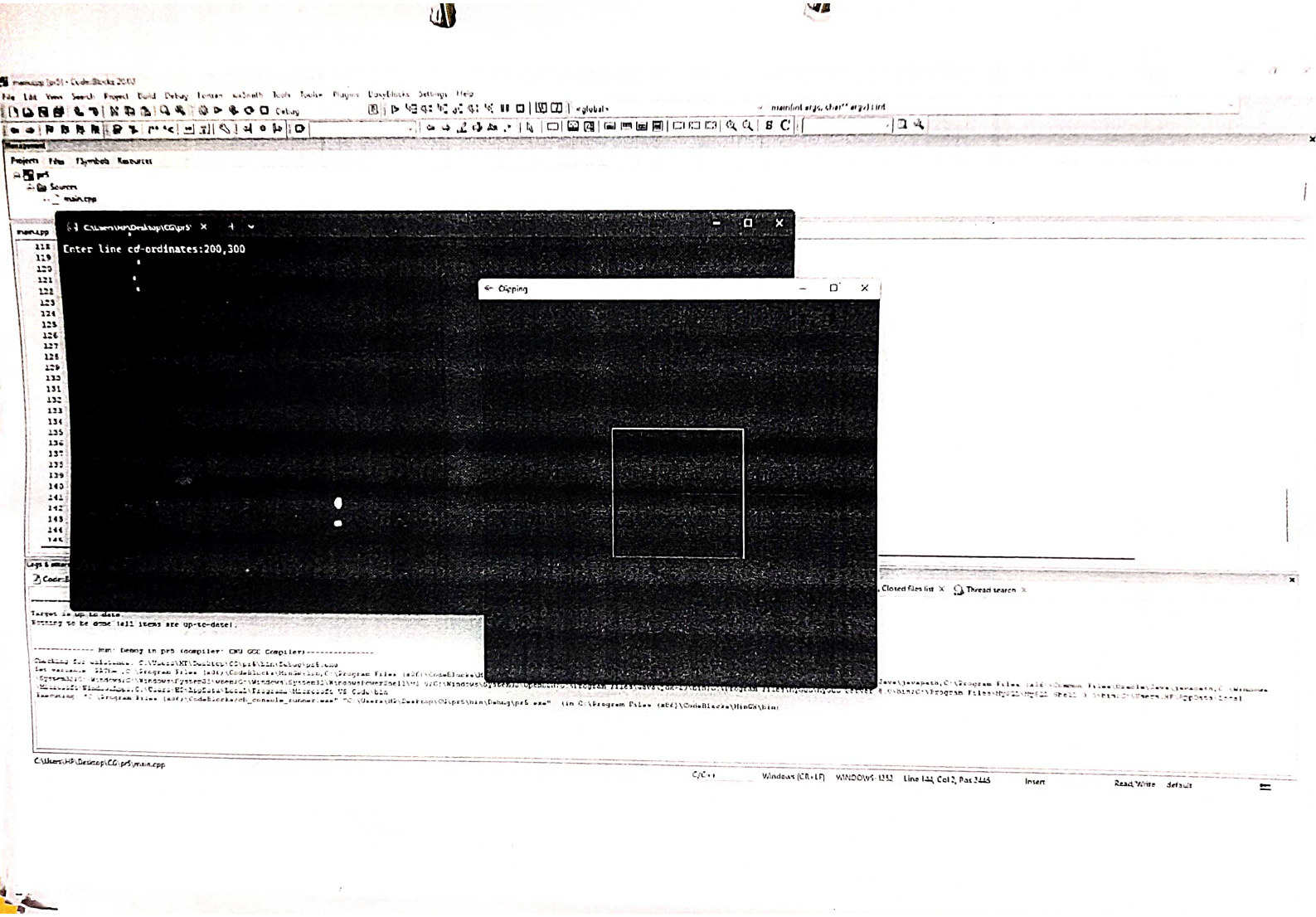
> How run your OpenGL program with command

→ $ ./testfilename.

[Output] :
(Execute program & attach the printout)

H [Conclusion] ;

In this way we have Studied that how to Install & explore OpenGL.


Enter line cd-ordinates:200,300
Clipping

Assignment :- 2

Problem Statement;

- Implement DDA and Bresenhams line drawing algorithm to draw ;
  i). Simple line
  ii). Dotted line
  iii) Dashed line
  iv) Solid line
  using mouse interface Divide the screen in 4 quadrants with center as (0,0). The line should work for all the slopes +ve as well as -ve.

Objective;

1). To lear DDA line drawing algorithm.
2). To learn Bresenham line drawing algorithm.

CO Relevance;

CO_2

PO's PSO's Relevance;

PO1, PO2, PO5, PO6

Raster Scan Display;

- When the beam reaches the bottom of the screen, it is made off and rapidly retraced back to top left to start again.

Components;

a) Graphic Commands
b) Frame Buffer.
c) Visual Display Unit.

Characteristics;

Host CPU Graphics Commands | Display file Translator | Display file Program | Display file Processor

VDU

Line;

- It is the path between two end points. Any point (x,y) on the line must follow the line equation.

$y = m * x + b$, where
- m is the slope of the line
- b is a constant represent the intercept on y-axis.

The slope of the line (m) between these points can be calculated as:

$$m = \Delta y / \Delta x = (y_1 - y_0) / (x_1 - x_0)$$

To draw a line, we can use two algorithms:

I. DDA line Drawing Algorithm;

1). Start
2) Declare variables & gDriver = DETECT & gMode.
3). Initialize graphics mode using initgraph( )
4). Read line end points (x_1, y_1) & (x_2, y_2)
5). Calculate difference betw. 2 end points.

$dx = x_1 - x_2$

$dy = y_1 - y_2$

6). If $dx >= dy$ & $x_1 <= x_2$ then
Increment x by 1
" y by m
Else if $dx >= dy$ & $x_1 > x_2$ then
Increment x by -1
" y by -m
Else if $dx < dy$ & $y_1 <= y_2$ then
Increment x by 1/m
Increment y by 1
Else if $dx < dy$ & $y_1 > y_2$ then
Increment x by -1/m
" y by -1

7) Plot x increment & y increment.
8) repeat step 6
9). Close graph
10). Stop

Bresenham's Line Drawing Algorithm :

1) Start
2) Declare the variables and gdriver = DETECT & gMode.
3) Initialize the graphics mode using initgraph().
4) Read the line end points (x_1, y_1) & (x_2, y_2)
5) Calculate $dx = x_2 - x_1$ & $dy = y_2 - y_1$
6) $x = x1$ & $y = y1$
7) $e = 2 * dy - dx$
8) $i = 1$
9) Plot (x, y)
10) while $(e >= 0)$
    {
    $y = y + 1$
    $e = e - 2 * dx$
    }
    $x = x + 1$
    $e = e + 2 * dy$
11) $i = i + 1$
12) if $(i <= dx)$ then go to step 9.

13) Close graph

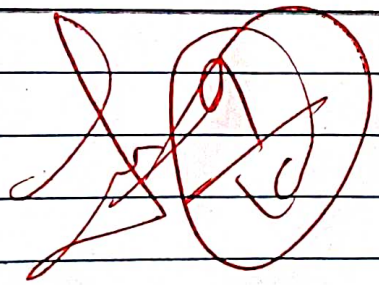
14) Stop.

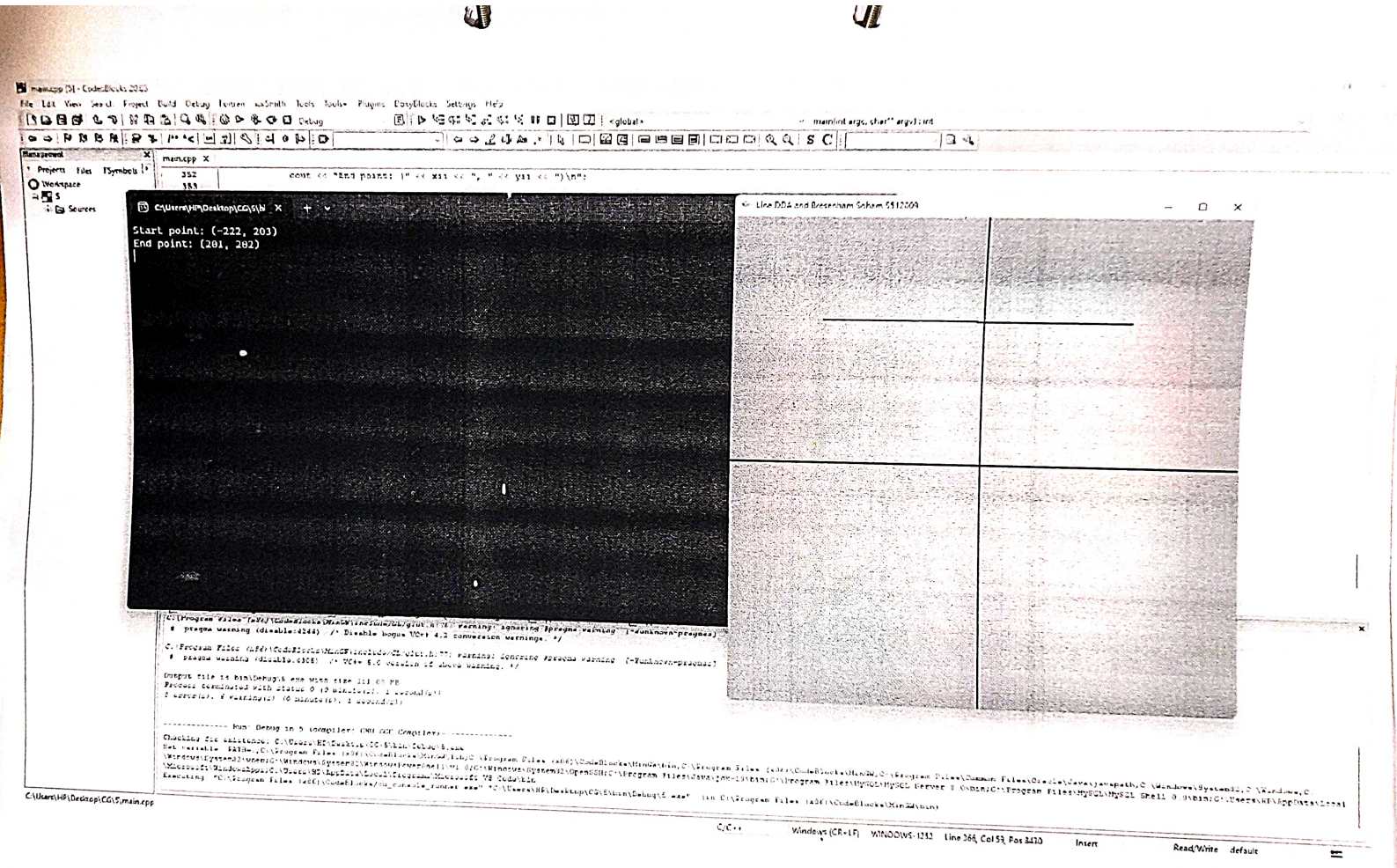
Output ;

- (Execute program and attach printout here)

Conclusion ;

- In this way we have studied that how to draw a line using DDA and Bresenham is.




C:\Users\HP\Desktop\CG\5\N
Start point: (-222, 203)
End point: (201, 202)
Line DDA and Bresenham Scham 5512009

Assignment : 3

Problem Statement :

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

Objective :

- To Understand the basic concepts of circle drawi

Outcome :

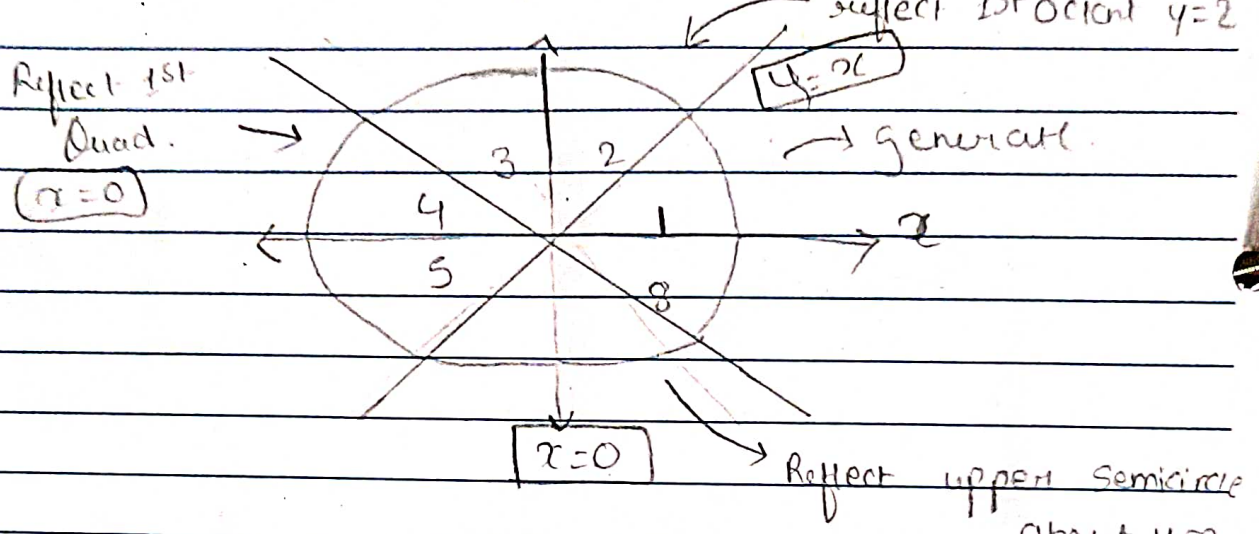
- To implement basic concept of circle drawing using Bresenhams algorithm.

Theory Concepts :

- One of the most efficient and esiest to draw circle is by using Bresenhams circle drawing algorithm.
- The other parts can be obtained by successive reflections. The line $y = x$ to yield the 1st quadrants.
- The result in the 1st Quadrant are reflected through the line $x = 0$ to obtain these in 2nd Quadrant.

Diagram:

Generation of complete Circle from the fist octant



- This can be decided by parameter p
  - If $p <= 0$, then $N(x+1, y)$
  - If $p > 0$, then $S(x+1, y-1)$

\# **Algorithm** :

1) Input radius r and circle center (x_c, y_c) obtained the first point $(x_0, y_0) = (0, r)$

2) Calculate the initial value of decision parameter as $\boxed{P_0 = 3 - 2r}$

3) At each x_k position, Starting at $\boxed{k=0}$, perform the following test:

IF $P_K < 0$ → the next pt. is (x_{k+1}, y_k)
$P_{k+1} = P_k + 4x_k + 6$

Otherwise next point is $(x_{k+1}, y_k - 1)$
$P_{k+1} = P_k + 4(x_k - y_k) + 10$

4) Determine symmetry points in other seven octant.

5) Move each pixel position (x, y) into circular path $x = x + xc$ & $y = y + yc$

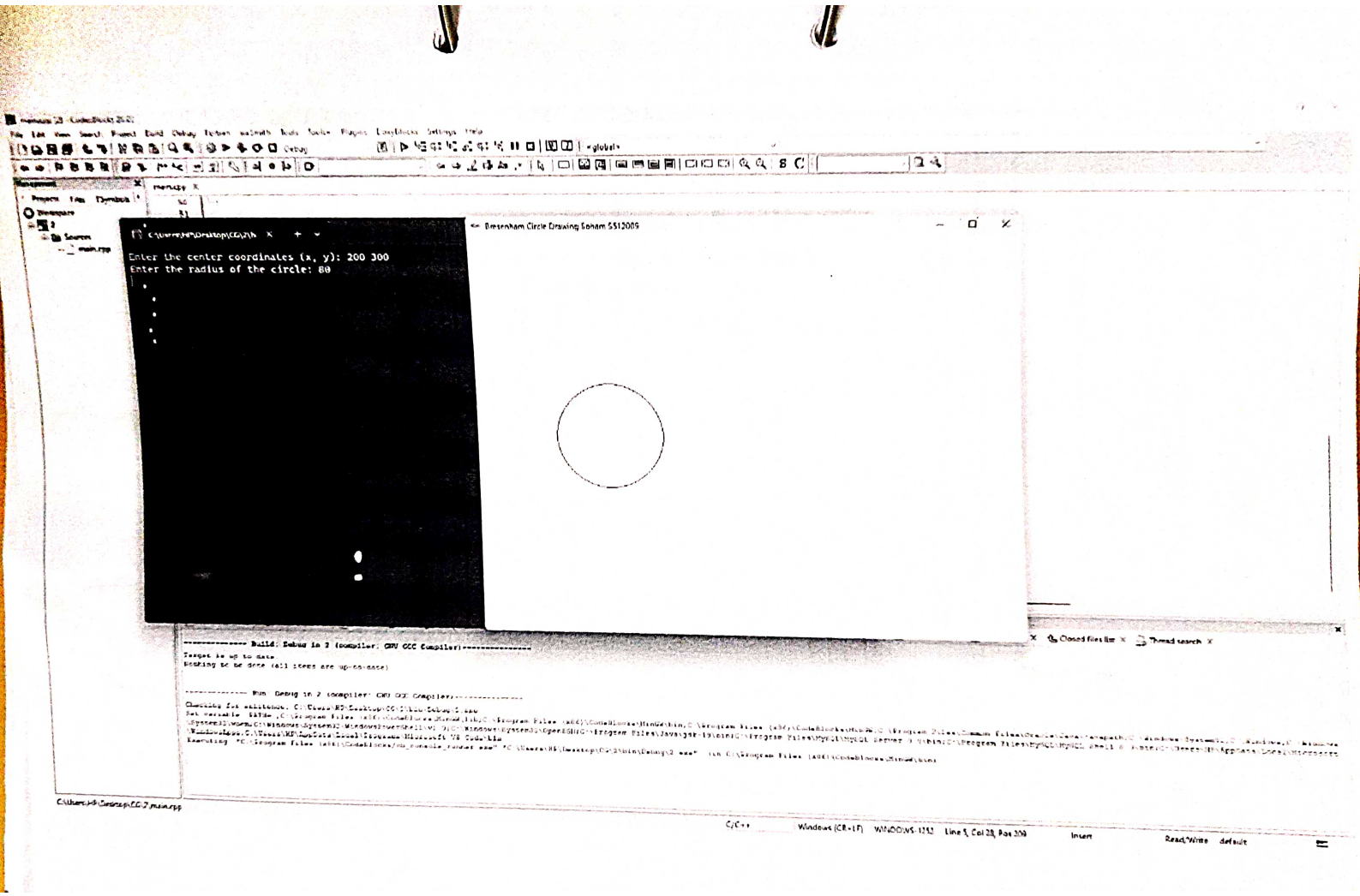
6) Repeat step 3 & 5.

Output :

Execute program & attach.

Conclusion :

In this way we have studied that how to draw a circle using Bresenham's circle drawing algorithm.


Enter the center coordinates (x, y): 200 300
Enter the radius of the circle: 80
Bresenham Circle Drawing Soham 5512009

Assignment :- 4

Problem Statement:

- Implement following polygon filling methods; i) Flood fill / Seed fill ; ii) Boundary fill ; using mouse click, keyboard interface & menu driven programming.

Objective :

- To Understand the basic concepts of polygon filling.

Outcome:

- To implement a polygon filling methods using flood fill / Seed fill & Boundary fill.

Theory Concepts:

1). Polygon Filling Approaches:

Types:

a). Boundary fill Algorithm.
b). Flood fill Algorithm.

I]. Boundary fill Algorithm:

Algorithm ;

1). The Boundary fill procedure accepts input as coordinates of an interior point (x, y) Fill color and boundary color.

2). Starting from (x, y) which is seed pixel, the procedure tests the neighbouring positions to determine whether they are boundary color.

3). If not, they are painted with fill color, and neighbours are tested.

4). This process continues untill all pixels up to the boundary color for the are have been tested.

- These are two methods for filling the pixel and find the neighbour pixel ;

i). 4-connected

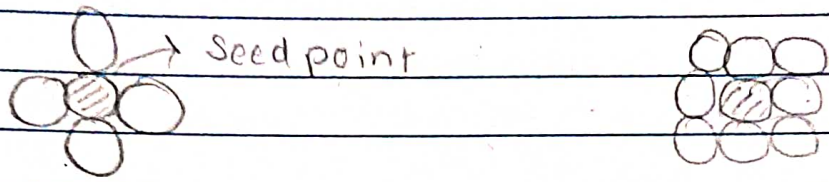
ii). 8-connected.

II). Flood fill ;

- The purpose of flood fill is to color an entire

We start with seed pixel, seed pixel is examined for specified interior color instead of boundary color.

Diagram:



i) 4-connected ii). 8-connected

Pseudo Code for boundary fill-Algorithm;

```
void boundaryfill4 (int x, int y, int fill_color, int
                    boundary_color)
{
  putpixel (x, y, fill_color);
  boundaryfill4 (x+1, y, fill_color, boundary
                 _color);
  .
  .
  .
  boundaryfill4 (x, y-1, fill_color, boundary
                 _color);
}
}
```

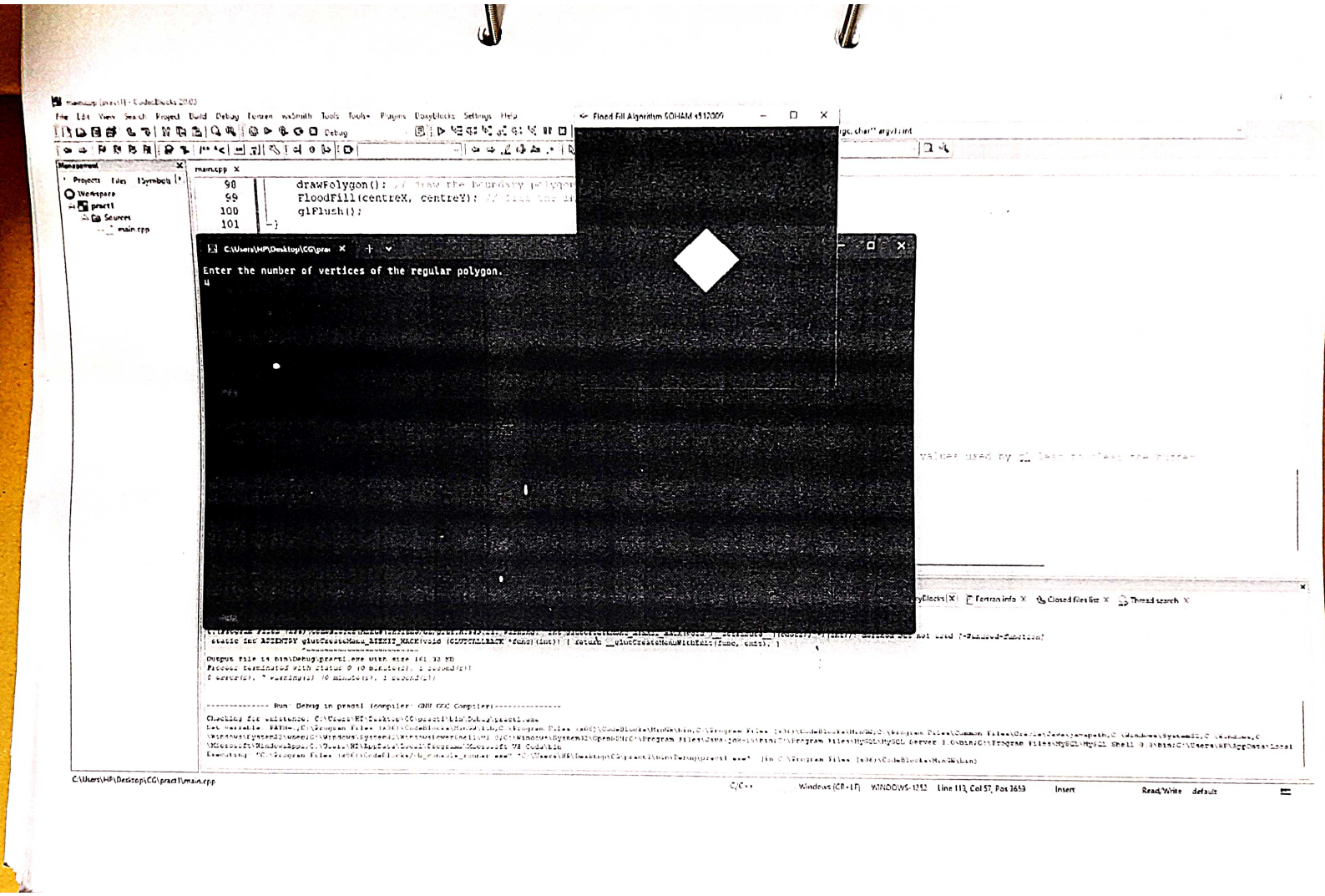
Pseudo code for Flood fill Algorithm;

```
IF (getpixel (x, y) = old_color)
{
    setpixel(x, y, fill_color);
    fill (x+1, y, fill_color, old_color)
    fill (x-1, y, fill_color, old_color);
    (x, y-1, Fill_color, old_color);
}
```

Output;

Conclusion;

In This way we have studied that how to fill polygon using flood fill / seed fill & boundary fill method.


Filled Fill Algorithm SOHAM s512009
C:\Users\HP\Desktop\CG\pra
Enter the number of vertices of the regular polygon.
4
drawPolygon(); // Draw the boundary polygon
FloodFill(centreX, centreY); // Fill the in
glFlush();
}
main.cpp
Workspace
pract1
Sources
main.cpp
Debug
Run: Debug in pract1 (compiler: GNU GCC Compiler)
C:\Users\HP\Desktop\CG\pract1\main.cpp
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 113, Col 57, Pos 3653
Insert
Read/Write
default

Assignment No:-5

Problem Statement:

- Implement Cohen Southerland polygon clipping method to clip the polygon with respect the view-port and window.
- Use mouse click, Keyboard interface.

Objective;

- To Understand the basic concepts of polygon clipping.

Outcome;

- To implent basic concept of polygon clipping using Cohen Southerland clipping dgorithm.

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.
- The algorithm was developed in 1967 during flight simulator work by Dany Cohen and Ivan Sutherland. The design state includes excludes or partially

includes line based on where;

- Both endpoints are in viewport region (bitwise OR of endpoints == 0);
- Both endpoints share at least on non-visible region. (bitwise AND of endpoints != 0)

The numbers in the figure below are called Outcodes.

- The Outcode is computed for each of the two pts. in. The outcode will have four bits for two-dimensional clipping, or six bits on three dimensional case.

- The first bit is set if is the point is above the viewport. The bits in the 2D outcode represent.
- Top Bottom, Right, Left.
- for example, the Outcode 1010 represent a point i.e. 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 :

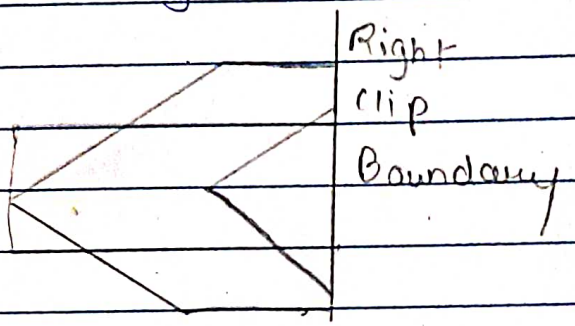
- Used for clipping polygons. It works by extending each line of the convex clip polygon in turn and selecting only vertices from the subject polygon that are on the visible state.

- This process is repeated for 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 i.e entirely visible.

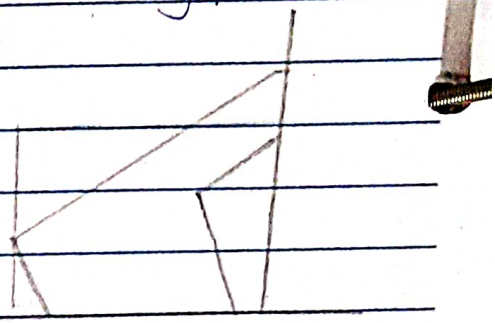
- Note i.e. the subject polygon was concave at vertices outside the clipping polygon, the new polygon may have

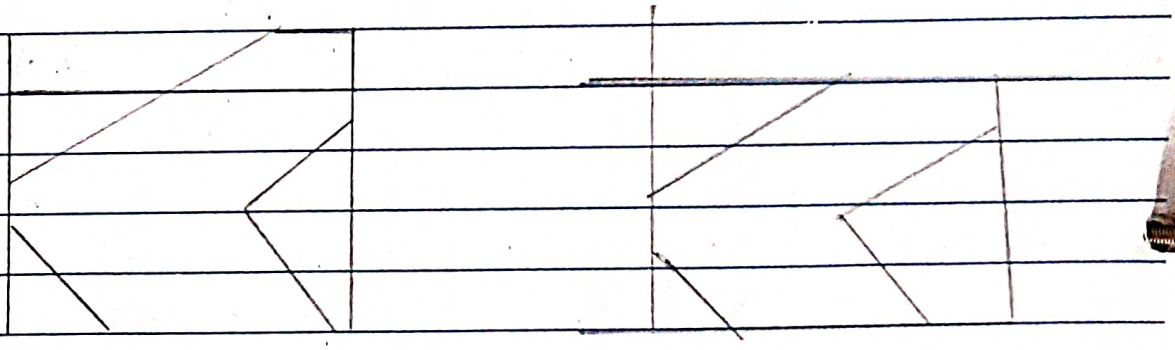
coincident (i.e. overlapping) edges - this is acceptable for rendering, but not for other applications such as computing shadows.

a].



b].

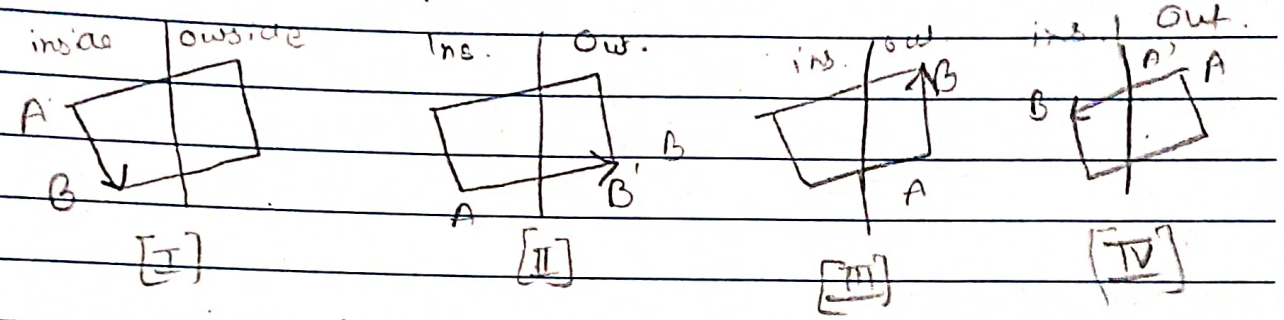




c] left Clip Boundary

d] Top Clip Boundary

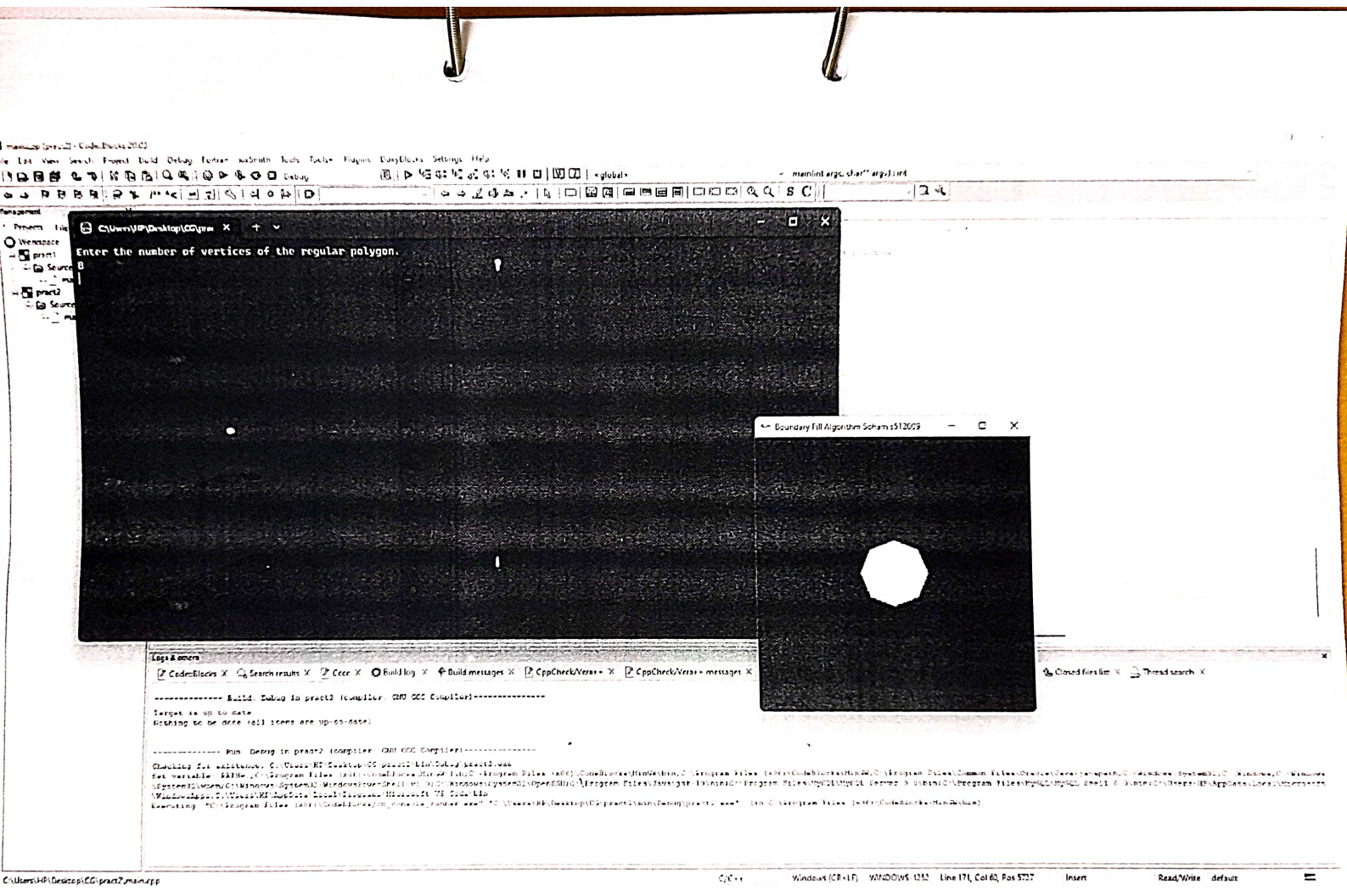
- The Sutherland - Hodgman algorithm clips a polygon to be quite complex.
- The algorithm steps from vertex to vertex adding 0, 1 or 2 vertices to the output list at each step



- The Sutherland - Hodgeman Polygon - Clipping Algorithms clips a given polygon successively against the edges of the given clip-rectangle.

Conclusion :

- In this way we have studied that how to clip polygon using Sutherland Clipping algorithm.


C:\Users\HP\Desktop\CG\pra
Enter the number of vertices of the regular polygon.
8
Boundary Fill Algorithm Scharms12009

Assignment :- 6

Problem Statement;

- Implement following 2D transformations on the object with respect to axis;
  i). Scaling ; ii). Rotation about arbitor
  iii). Reflection.

Objective ;

1). To understand 2D Homogenous coordi. System

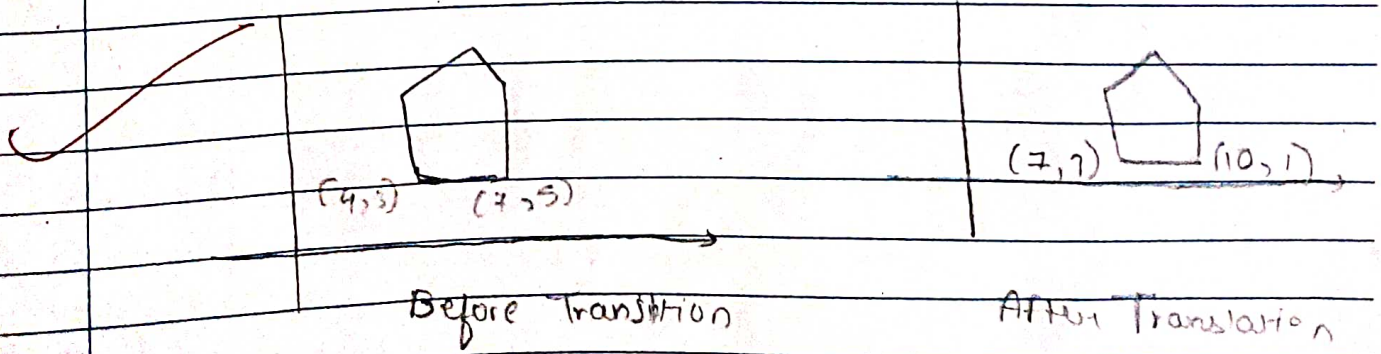
2). Understand and implement 2D transformations in Laboratory.

Outcome ;

-- To implement transformation operations.

Theory ;

1). **Translation ;**



- We can move the objects based on translation distances along x & y axis

- Consider (x, y) are old coordinates of a point. Then the new coordinates of that same point (x', y') can be obtained as follows;

$$x' = x + tx$$
$$y' = y + ty$$

2). **Scaling** ;

- Scaling refers to changing size of the object either by increasing or decreasing.

- If (x, y) are old coordinates of object, then new coordinates of object after applying scaling transformation are obtained as;

$$x' = x * Sx$$
$$y' = y * Sy$$

$$\therefore \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix}$$

Rotation :

- A rotation repositions all points in an object along a circular path in the plane centered at the pivot point. We rotate an object by an angle theta.

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

$$P' = R * P$$

R - Rotation Matrix.

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

formula :
$$X = x\cos A - y\sin A$$
$$y = x\sin A + y\cos A$$

The above formula will rotate the point around the origin.
To rotate around a diff. point, the formula;

$$X = cx + (x - cx) * \cos A - (y - cy) * \sin A$$
$$y = cx + (x - cx) * \sin A + (y - cy) * \cos A$$

Cx, Cy is centre coordinates,
A is the angle of rotation,

Reflection ;

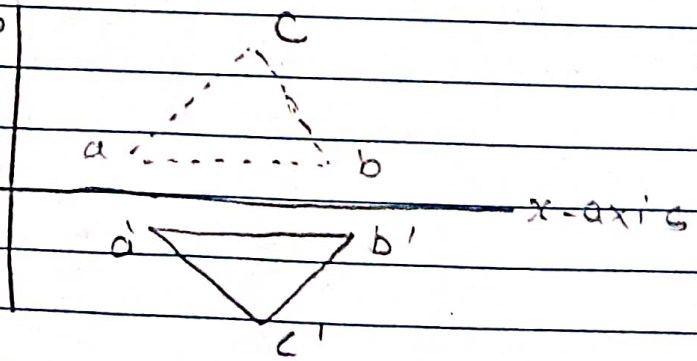
- It is transformation which produces a mirror image of an object. The mirror image can be either about x-axis or y-axis or object rotated by 180°.

Types of Reflection;

I] Reflection about x-axis;

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

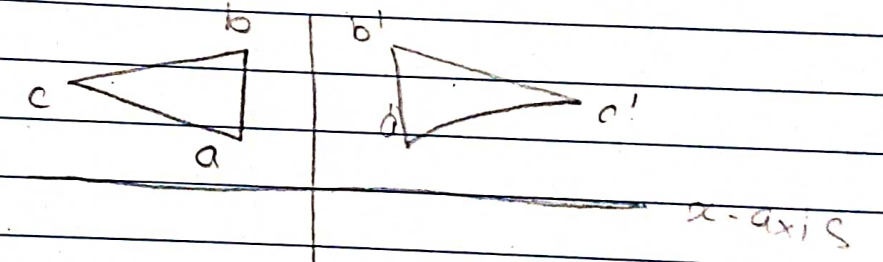
- In this transformation value of x will remain same whereas value of y will become -ve. Following figure shows the reflection of the object axis. The obj. will lies on another side of the x-



II]. Reflection about y-axis;

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

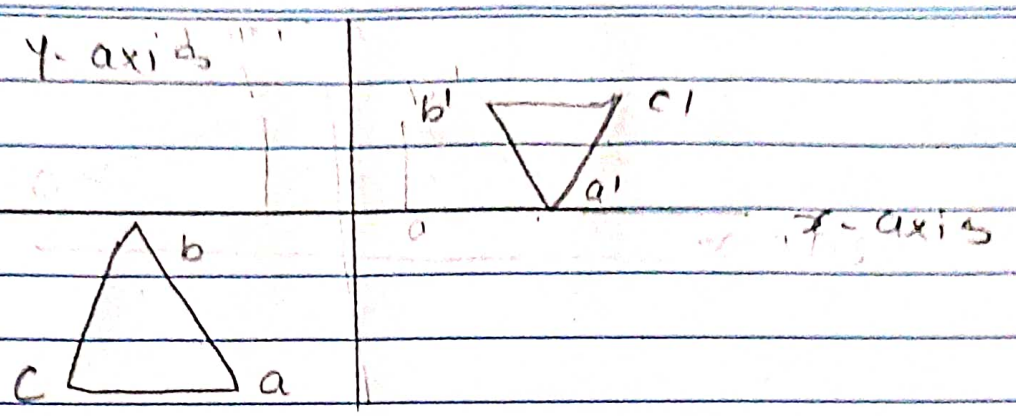
Here the values of x will be reversed, whereas the value of y will remain the same. The object will lie another side of the y-axis.



III]. Reflection about an axis perpendicular to xy plane & passing through origin;

In the matrix of this transformation is given below

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

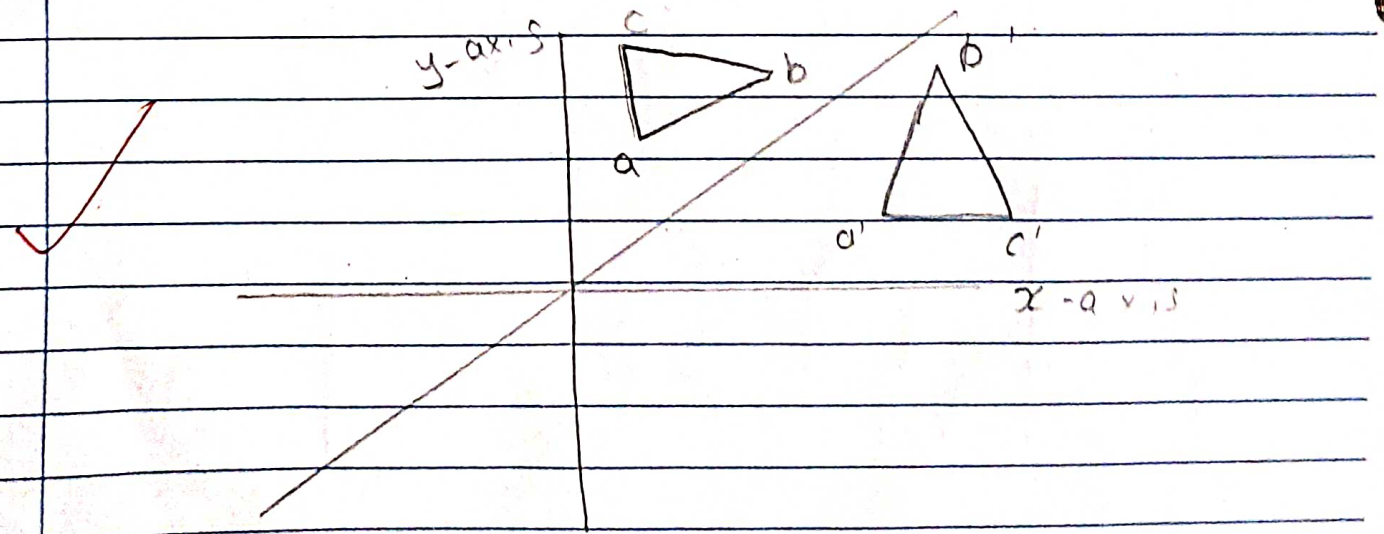


In this value of x and y both will be reversed. This is also called as 1/2 revolution about the origin.

IV] Reflection about line $y=x$:

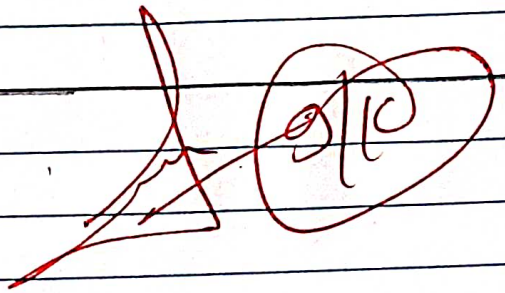
- The object may be reflected about line $y=x$ with the help of following transformation matrix.

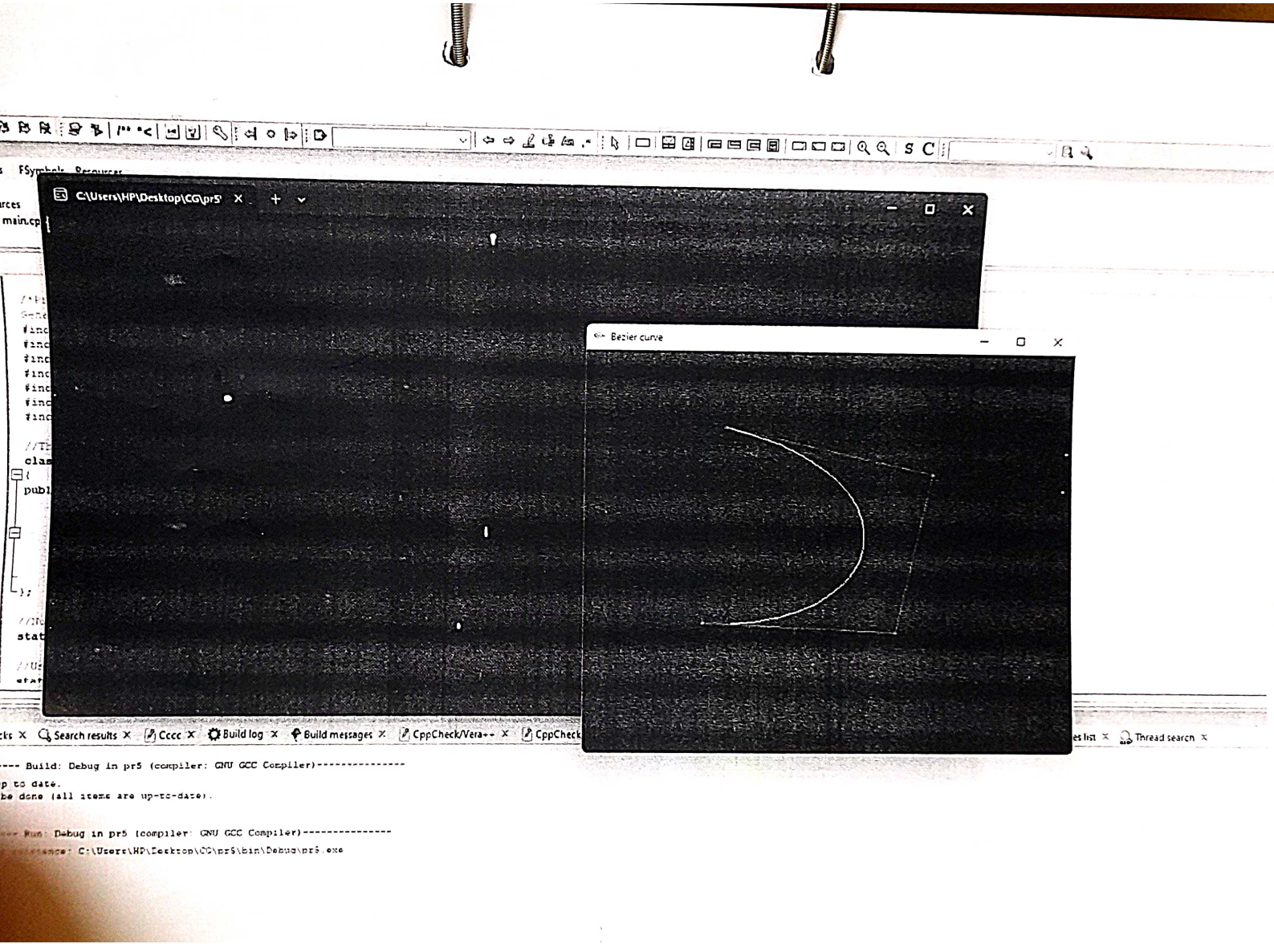
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



11 Conclusion:

In this way we have Studied that how to perform 2 dimension operations on object.




C:\Users\HP\Desktop\CG\pr5
Bezier curve
Search results
Cccc
Build log
Build messages
CppCheck/Vera++
Thread search
---- Build: Debug in pr5 (compiler: GNU GCC Compiler)----------
p to date.
be done (all items are up-to-date).
--- Run: Debug in pr5 (compiler: GNU GCC Compiler)----------
: C:\Users\HP\Desktop\CG\pr5\bin\Debug\pr5.exe

Assignment :- 7

Problem Statement;

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

Objective;

- Understand different types of curve.
- Implement diff. Objects using Koch curve & bazier curve fractal pattern.

Outcome;

- To implement fractals patterns using bazier & koch curve.

Theory Concepts;

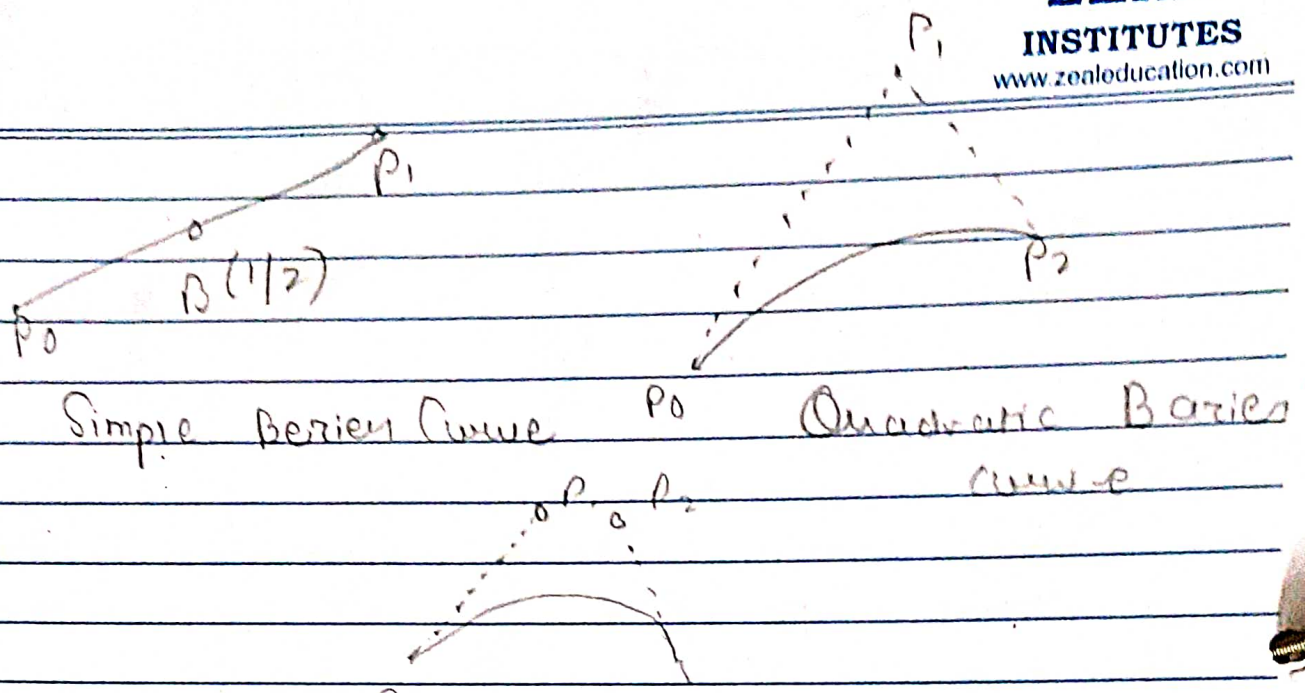
1) Bezier Curves:

- Bezier curve is discovered by French engineer Pierre Bézier. These curves can be generated under control of other points. Approximate tangents by using control points 're used to generate curve. The Bezier can be represented as;

$$\Sigma K = On\ Pi\ Bni(t)$$

(where P is set of points & Bin (t) represents Bernstein polynomials given by;

$$Bni(t) = (ni)(1-t)n-ti$$



Simple Bezier Curve Quadratic Bezier curve

Cubic Bezier Curve

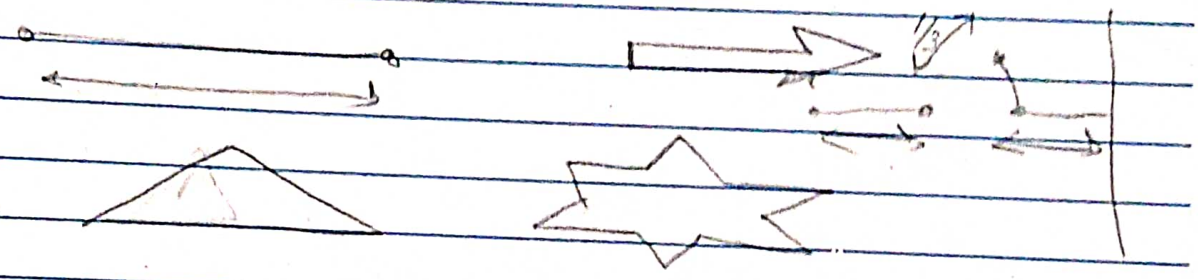
Properties of Bezier Curves:

- They generally follow shape of control polygon
- They always pass through first & last.
- They are contained in convex hull of their defining control pts.
- The degree of polynomial defining curve segment is one less that no. of defining polygon point.
- A Bezier curve generaly follows shape of defining polygon.
- The convex hull property for Bezier curve ensures that polynomial smoothly follows control pts.
- No straight line intersects Bezier curve more times it intersects its control polygon.
- They are invariant under an affine transformation.

Surface textures & just to make pretty patterns.

- In other disciplines, fractal patterns've been found in distrib. of stars, river islands and moon craters; in rain field; in stock market variations; in music; in traffic flow.

Koch Fractals (Snow-flakes)



Add Some Randomness:

> Fractals we've produced so far seem to be very regular & "artificial".

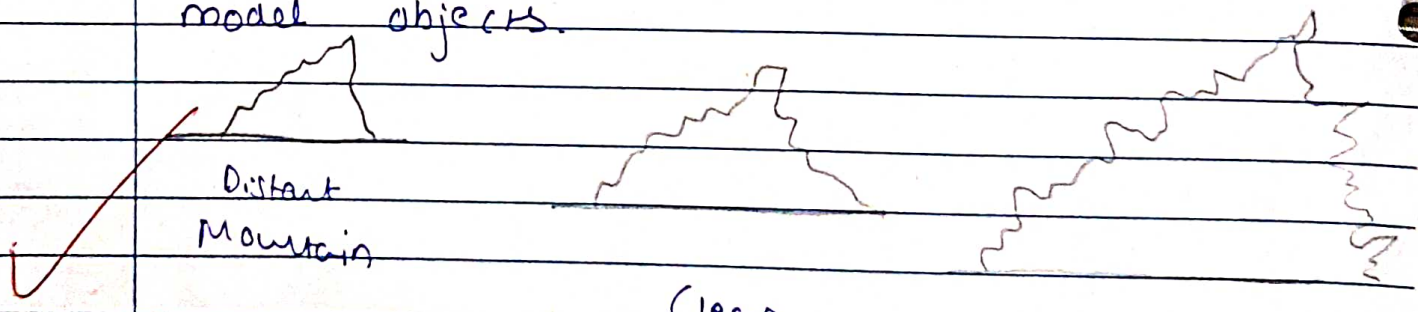
> To create some realism & variability simply change angles slightly sometimes based on "random" no. generator.

? For eg:- you can curve some of trees to one side

- Bezier curves exhibit global control means moving a control point alters shape of whole curve.

Koch Curve;

- Fractals are geometric objects. Many real-world objects like ferns are shaped like fractals. Fractals are self-similar. In CG, we use fractal functions to create complex object. The obj. representations uses Euclidean - geo. methods; obj. shapes were described the eqns. Those that're smooth surfaces & regular shapes.
- Natural objects can be realistically described fractal-geo. methods, where procedures rather than eqn're used to model objects.



Ex: In graphics application, fractal representations are used to model terrain, clouds, water, trees & other plans, feathers, fur & various

Terrain (Random Mid-Point Displacement):

> Given hts. of two end-pts. → generate a ht. at mid-pt.

> Then → the height at mid-pt. will be:

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

r is random offset.

> This is how to generate random offset r

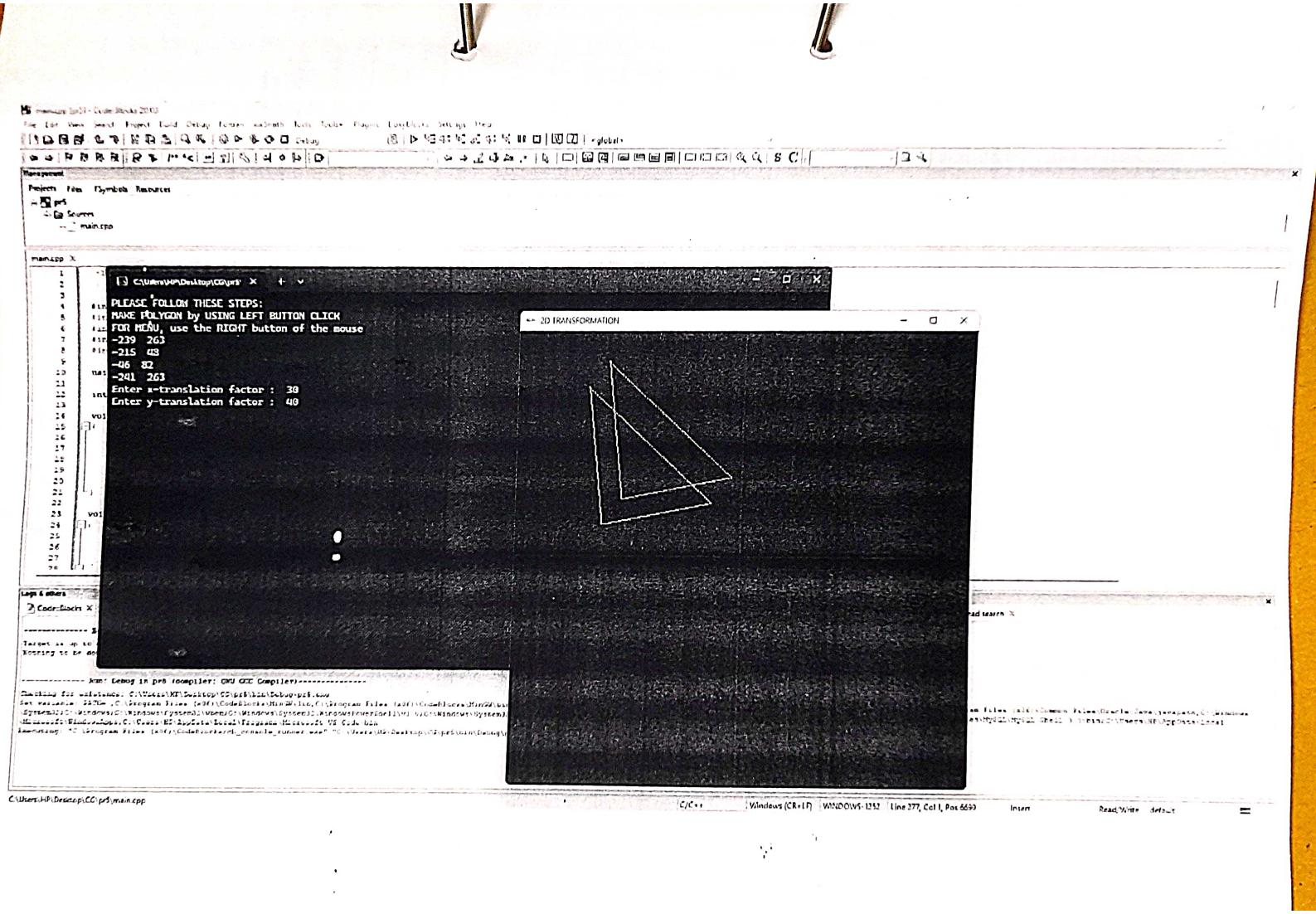
$$r = s.rg(b-a),$$

rg is Gaussian random variable with mean 0 & variance 1.

Conclusion:

In this way we've studied that how to generate fractal patterns using curve generation methods.

9/10


C:\Users\HP\Desktop\CG\pr5
PLEASE FOLLOW THESE STEPS:
MAKE POLYGON by USING LEFT BUTTON CLICK
FOR MENU, use the RIGHT button of the mouse
-239 263
-215 403
-46 82
-241 263
Enter x-translation factor : 30
Enter y-translation factor : 40
2D TRANSFORMATION

Assignment :- 8

Problem Statement:

— Implement animation principles for any object

Objective;

1). To learn different types of animation
2). To learn OpenGL function which support for animation.

Outcome;

— To implement animation for an object.

Theory Concepts;

— Animation is defined as series of images rapidly changing to create an illusion of movement. We replace previous image with new image is little bit shifted.

— Animation Industry is having a huge market nowadays. To make an efficacious animation there are some principle to be followed.

Principle;

— There are 12 major principles for an effective and easy to communicate animation.

I]. Squash & Stretch;

— 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.

II] Anticipation;

– Anticipation works on action. Animation has broadly divided into 3 phases:

1) Preparation phase
2) Movement phase
3) Finish.

– In Anticipation; we make audience prepare for action.

III] Arcs;

– In Reality, humans & animals move in arcs. Introducing concept of arcs will increase realism. This principle of animation helps us to implement the realism through projectile motion also.

IV] 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 slow out

V]. Appeal;

- Animation should be appealing to the audience and must be easy to understand. The Syntax or font style user should be easily understood and appealing to the audience.

VI]. Timing;

- Velocity with which object is moving effects animation a lot. The speed is handled with care in case of animation.

VII]. 3D- Effect;

- By giving 3D-effects we can make our animation more convincing and effective.
- In 3D Effect, we convert our object in 3D plane i.e. X-Y-Z plane which improves realism of the object.

VIII]. 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 balance between various exaggerated element to avoid conflicts.

IX] Staging;

- Staging is defined as presentation of primary idea, mood or action.
- It should always be in presentable and easy to manner.
- The purpose of defining principle is to avoid unecessary details and focus on important features only.
- The primary idea should always be clear & unambigious.

X] Follow Through;

- It refers to the action which continues to move even after the completion of action. This type of action helps in generation of more idealistic animations.

XI. Overlap;

- It deals with the nature in which before ending first action, the second action starts.

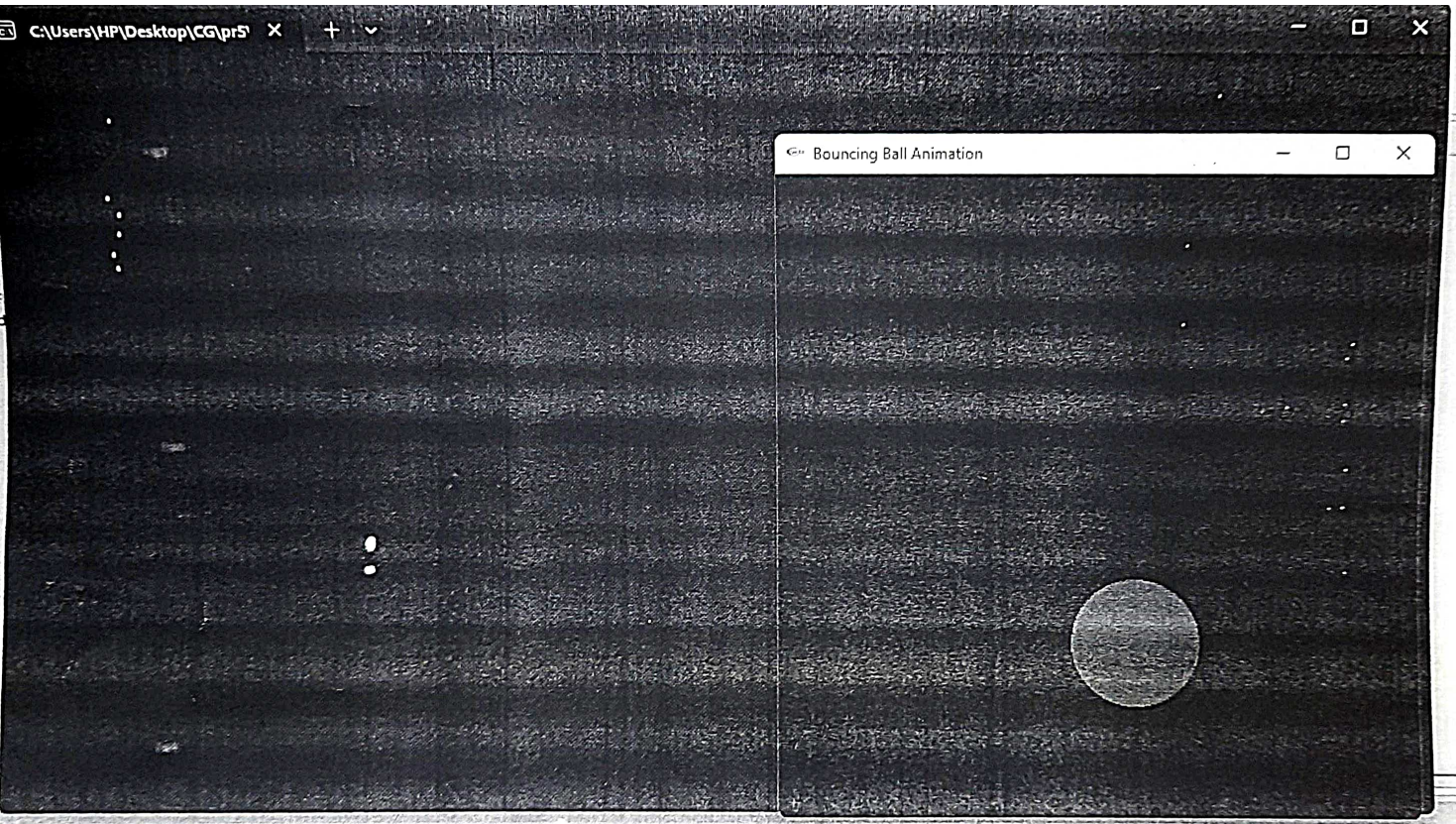
For eg:- Consider situation when we are drinking Tea from right hand and holding sandwich in the left hand.
- While drinking a tea our left hand start showing movement towards mouth which shows interference of second action before end of 1st action.

Output;
(Execute)

Conclusion;

- In this way we have studied that how to use animation techniques.

9/10



Search results | Cccc | Build log | Build messages | CppCheck/Vera++ | CppCheck/Vera++ messages | Cscope | Debugger | DoxyBlocks | Fortran info | Clo

```
(disable:4305)  /* VC++ 5.0 version of above warning. */

from C:\Users\HP\Desktop\CG\pr5\main.cpp:3:
x86)\CodeBlocks\MinGW\include/GL/glut.h:549:21: warning: 'int glutCreateMenu ATEXIT HACK(void (  attribute  ((cdecl)) *)(int))' defined but not used [-Wunused-function]
```