# Computer Graphics laboratory

## **"Project MathemGL"** Report

(https://github.com/SriAstitva/project-mathemGL)

## **Aim:**

To implement Standard Math's Interactive Animations using OpenGL, to make understanding of the important concepts of mathematics using Animations.

## **Potential Users:**

Mathematics Enthusiasts.

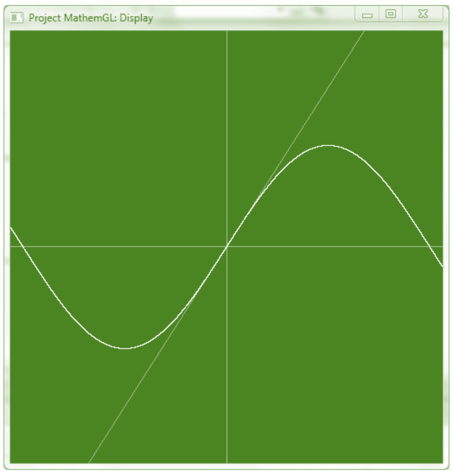
## **Potential Work:**

Animations include as follows:

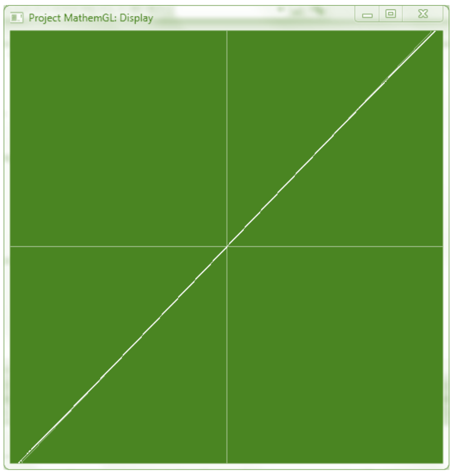
1) Formation of the locus of the **Conic Sections** namely, Circle, Ellipse and Parabola.
2) Cross-section of Cone to show different Conic Sections, including Hyperbola.
3) Animations to show some Standard Limits of Sin, Cos and Tan.

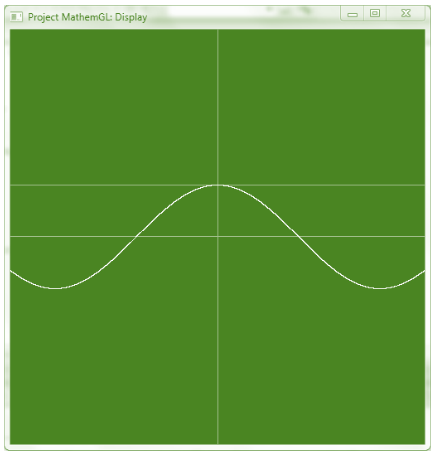
## Description:

**Graphical proofs of standard limits:**

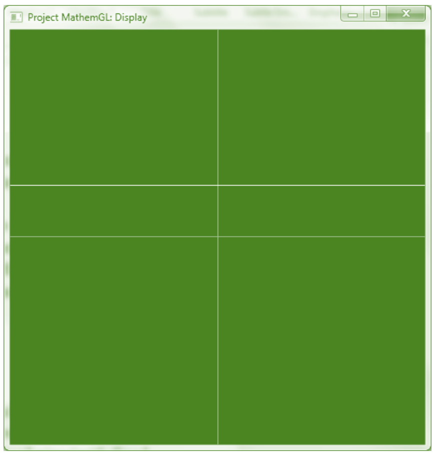


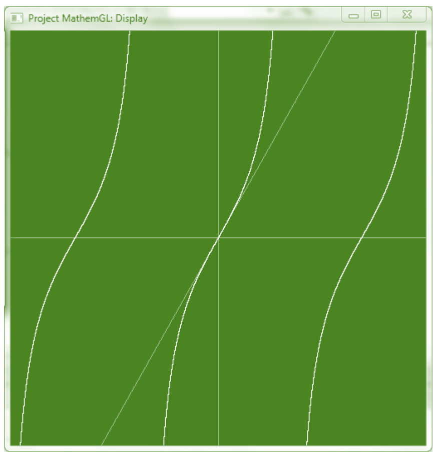
$$\lim_{x\to 0}\frac{sin\ x}{x} = 1$$



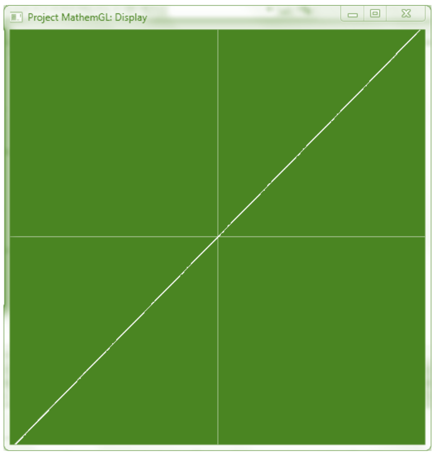


$$\lim_{x\to 0}\cos x = 1$$



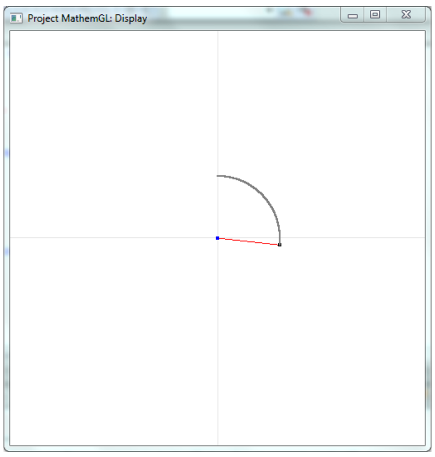


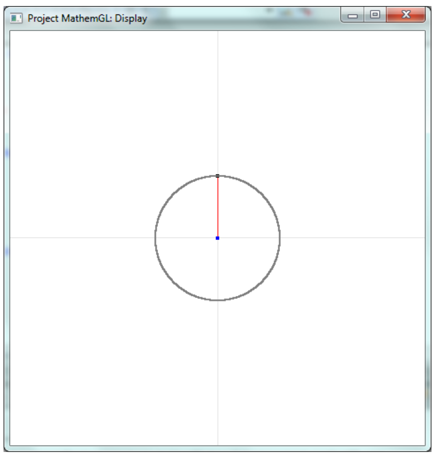
$$\lim_{x\to 0}\frac{\tan x}{x} = 1$$



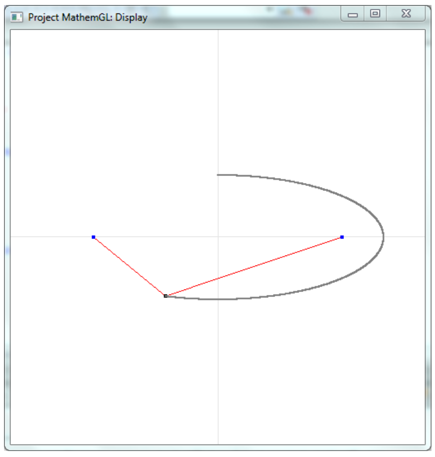
## Locus of Conic Sections:

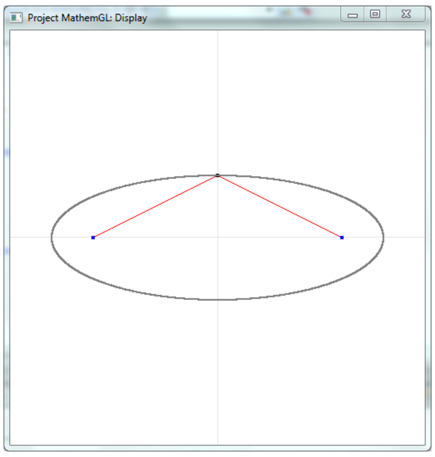
**Circle:** Circle is the locus of all the points at equidistant from a given fixed point, Center.



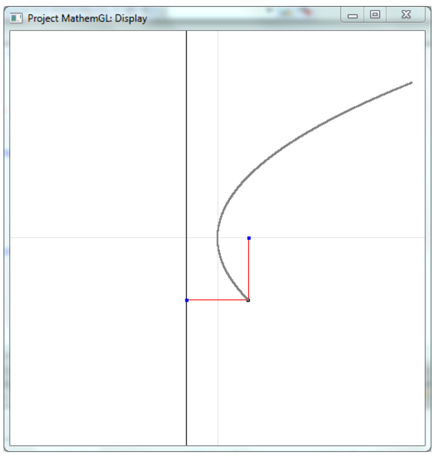


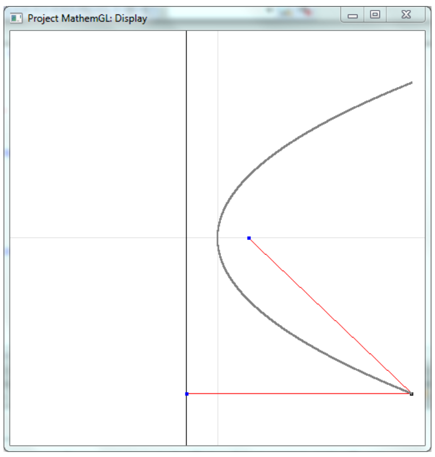
**Ellipse:** Ellipse is the locus of all the points such that sum of distances of the point from two given fixed points, Focii is constant.



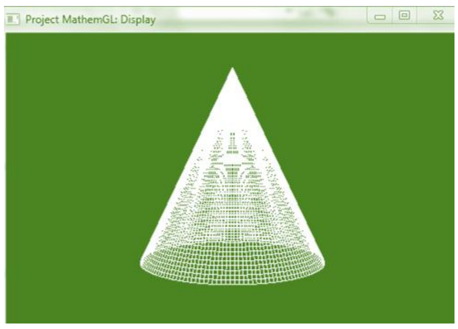


**Parabola:** Parabola is the locus of all the points at equidistant from a given fixed point, Focus and a line, Directrix.

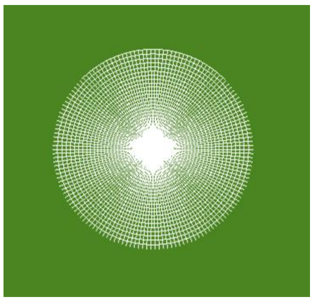




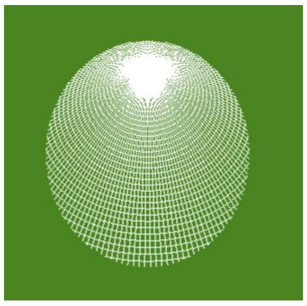
**Formation of Conic Sections from Cone:**



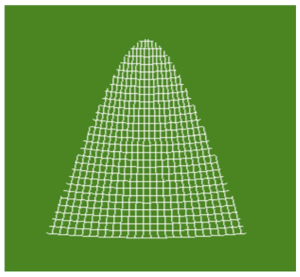
**Circle:** Circle is a conic section obtained by cutting the cone by a plane parallel to the base.



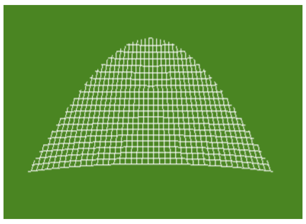
**Ellipse:** Ellipse is a conic section obtained by cutting the cone by a plane at angle < 90 with respect to base.



**Parabola:** Parabola is a conic section obtained by cutting the cone by a plane at angle > 0 (not equal to 0) with respect to its axis.



**Hyperbola:** Hyperbola is a conic section obtained by cutting the cone by a plane parallel to its axis.



## Code:

```
#include <gl/freeglut.h>
#include <iostream>
#include <cmath>

#define PRECISION 0.0001
#define ZOOM 0.01
#define PI 3.14159265359
#define MULT_FACTOR 0.1

#define PARABOLA_A 1.5          // Parabola semi-latus recum (a)
#define PARABOLA_T 2.5          // Extremes of parabola parameter
#define PARABOLA_DEL 0.0004     // Difference b/w parameter value b/w frames

double scaleFactorX = 1 /(4 * PI), scaleFactorY = 0.25;
int clickPosX, clickPosY;
float t = 0;
float tp = PARABOLA_T; // Parameter for Parabola
float transZ = -6, rotAngle = 0; // Constants for Conic Generation

// maxAngle: Ellipse: -150, Circle: -180, Parabola: -110, Hyperbola: -90
// maxTranslation: Ellipse, Circle: -2.2, Parabola: -2.2, Hyperbola: -2.4
// yTranslation: Ellipse: -0.2, Circle: 0, Parabola: -0.2, Hyperbola: -0.4
float maxAngle = -150, maxTranslationZ = -2.2, yTranslation = -0.2;

using namespace std;

void plotSine() {
    double x = -4*PI, y;
        glClearColor(0.0f,0.0f,0.0f,1.0f);
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_LINES);
        glColor3f(0.5, 0.5, 0.5);
        glVertex2d(-1, 0);
        glVertex2d(1, 0);
        glVertex2d(0, 1);
        glVertex2d(0, -1);
        for(int i=0 ; i<100 ; i++){
            for(int j=0 ; j<100 ; j++){
                if (i == j ){
                   glVertex2d(i * scaleFactorX, j * scaleFactorY);
                   glVertex2d(-i * scaleFactorX, -j * scaleFactorY);
                }
            }
        }
        glEnd();

        glBegin(GL_POINTS);
        glColor3f(1, 1, 1);
        do {
            y = sin(x);
            x = x + PRECISION;

            glVertex2d(x * scaleFactorX, y * scaleFactorY);
                } while(x<=4*PI);
        glEnd();
        glFlush();

        scaleFactorX += ZOOM;
        scaleFactorY += ZOOM;

        glutPostRedisplay();
}

void plotCos() {
    double x = -4*PI, y;
        glClearColor(0.0f,0.0f,0.0f,1.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        glBegin(GL_LINES);
        glColor3f(0.5, 0.5, 0.5);
        glVertex2d(-1, 0);
        glVertex2d(1, 0);

        glVertex2d(0, 1);
        glVertex2d(0, -1);

        glVertex2d(-1, 1 * scaleFactorY);
        glVertex2d(1, 1 * scaleFactorY);
        glEnd();

        glBegin(GL_POINTS);
        glColor3f(1, 1, 1);
        do {
            y = cos(x);
            x = x + PRECISION;
            glVertex2d(x * scaleFactorX, y * scaleFactorY);
                } while(x<=4*PI);
        glEnd();
        glFlush();

        scaleFactorX += ZOOM;

        glutPostRedisplay();
}

void plotTan() {
    double x = -4*PI, y;
        glClearColor(0.0f,0.0f,0.0f,1.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        glBegin(GL_LINES);
        glColor3f(0.5, 0.5, 0.5);
        glVertex2d(-1, 0);
        glVertex2d(1, 0);

        glVertex2d(0, 1);
        glVertex2d(0, -1);

        for(int i=0 ; i<100 ; i++){
            for(int j=0 ; j<100 ; j++){
                if (i == j ){
                   glVertex2d(i * scaleFactorX, j * scaleFactorY);
                   glVertex2d(-i * scaleFactorX, -j * scaleFactorY);
                }
            }
        }
        glEnd();

        glBegin(GL_POINTS);
        glColor3f(1, 1, 1);
        do {
            y = tan(x);
            x = x + PRECISION;
            glVertex2d(x * scaleFactorX, y * scaleFactorY);
                } while(x<=4*PI);
        glEnd();
        glFlush();

        scaleFactorX += ZOOM;
        scaleFactorY += ZOOM;

        glutPostRedisplay();
}

void plotCircle(){

    // Circle is being made using this eqn. x^2 + y^2 = 1
    glClearColor(1,1,1,1.0f);
        glClear(GL_COLOR_BUFFER_BIT);
        float x,y,px,py,pt;
        glBegin(GL_LINES);


            // X Axis
            glColor3f(0.9, 0.9, 0.9);
            glVertex2d(-1, 0);
            glVertex2d(1, 0);

            // Y Axis
            glVertex2d(0, 1);
            glVertex2d(0, -1);

            //Parametric Equations
            x = 3*sin(t);
            y = 3*cos(t);
            glColor3f(1.0, 0, 0);
            glVertex2f(0 *MULT_FACTOR,0*MULT_FACTOR);
            glVertex2f(x*MULT_FACTOR,y*MULT_FACTOR);
        glEnd();

        glPointSize(4);
        glBegin(GL_POINTS);
            glColor3f(0.0, 0.0, 1.0);
            glVertex2f(0 *MULT_FACTOR,0*MULT_FACTOR);
            glColor3f(0, 0, 0);
            glVertex2f(x*MULT_FACTOR,y*MULT_FACTOR);
        glEnd();

       glPointSize(2);
       glBegin(GL_POINTS);
            pt = 0;

            // Loop to make locus till the point where drawing point has reached
            while(pt<=t){
              px = 3*sin(pt);
              py = 3*cos(pt);
              glColor3f(0.5, 0.5, 0.5);
              glVertex2f(px*MULT_FACTOR,py*MULT_FACTOR);
              pt = pt + PI / 5000;
            }
       glEnd();

       // Animation: Updating the circle parameter if the drawing hasn't finished.
        if(t < 2*PI) {
            t = t + PI / 5000;
            glutPostRedisplay();
        }

        glFlush();
}
```

```
void plotEllipse(){
    // Ellipse is being made using this eqn. x^2 / 64 + y^2 / 9 = 1
    glClearColor(1,1,1,1.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    float x,y,px,py,pt;
        glBegin(GL_LINES);

            // X Axis
            glColor3f(0.9, 0.9, 0.9);
            glVertex2d(-1, 0);
            glVertex2d(1, 0);

            // Y Axis
            glVertex2d(0, 1);
            glVertex2d(0, -1);

            //parametric Equations
            x = 8*sin(t);
            y = 3*cos(t);

            // Lines joining Focii to Ellipse
            glColor3f(1.0, 0, 0);
            glVertex2f(-6 *MULT_FACTOR,0*MULT_FACTOR);
            glVertex2f(x*MULT_FACTOR,y*MULT_FACTOR);
            glColor3f(1.0, 0.0, 0.0);
            glVertex2f(x*MULT_FACTOR,y*MULT_FACTOR);
            glVertex2f(6*MULT_FACTOR,0*MULT_FACTOR);
        glEnd();


        glPointSize(4);
        glBegin(GL_POINTS);
            glColor3f(0.0, 0.0, 1.0);
            glVertex2f(6 *MULT_FACTOR,0*MULT_FACTOR);
            glVertex2f(-6 *MULT_FACTOR,0*MULT_FACTOR);
            glColor3f(0, 0, 0);
            glVertex2f(x*MULT_FACTOR,y*MULT_FACTOR);
        glEnd();

       glPointSize(2);
       glBegin(GL_POINTS);
            pt = 0;

            // Loop to make locus till the point where drawing point has reached
            while(pt<=t){

              //Parametric Equations
              px = 8*sin(pt);
              py = 3*cos(pt);
              glColor3f(0.5, 0.5, 0.5);
              glVertex2f(px*MULT_FACTOR,py*MULT_FACTOR);
              pt = pt + PI / 5000;

            }
       glEnd();

       // Animation: Updating the ellipse parameter if the drawing hasn't finished.
        if(t < 2*PI) {
            t = t + PI / 5000;
            glutPostRedisplay();
        }

    glFlush();
}

void plotParabola() {
    // Parabola is being made using this eqn. y^2 = 4 * a * x
    float x, y, px, py, pt;
```

```
glClearColor(1, 1, 1, 1);
    glClear(GL_COLOR_BUFFER_BIT);

glBegin(GL_LINES);
    x = PARABOLA_A * tp * tp;
    y = 2 * PARABOLA_A * tp;

    // X Axis
    glColor3f(0.9, 0.9, 0.9);
    glVertex2d(-1, 0);
    glVertex2d(1, 0);

    // Y Axis
    glVertex2d(0, 1);
    glVertex2d(0, -1);

    // Directrix
    glColor3f(0, 0, 0);
    glVertex2f(-PARABOLA_A * MULT_FACTOR, 1);
    glVertex2f(-PARABOLA_A * MULT_FACTOR, -1);

    // Line joining Directrix to Parabola
    glColor3f(1, 0, 0);
    glVertex2f(-PARABOLA_A * MULT_FACTOR, y * MULT_FACTOR);
    glVertex2f(x * MULT_FACTOR, y * MULT_FACTOR);

    // Line joining Focus to Parabola
    glVertex2f(x * MULT_FACTOR, y * MULT_FACTOR);
    glVertex2f(PARABOLA_A * MULT_FACTOR, 0 * MULT_FACTOR);
glEnd();

glPointSize(4);
glBegin(GL_POINTS);

    // Focus
    glColor3f(0, 0, 1);
    glVertex2f(PARABOLA_A * MULT_FACTOR, 0 * MULT_FACTOR);

    // Point (wrt. y-coordinate) on Directrix
    glVertex2f(-PARABOLA_A * MULT_FACTOR, y * MULT_FACTOR);

    // Point on the Parabola (where it is being drawn)
    glColor3f(0, 0, 0);
    glVertex2f(x * MULT_FACTOR, y * MULT_FACTOR);
glEnd();

glPointSize(2);
glBegin(GL_POINTS);
    pt = PARABOLA_T;
    // Loop to make locus till the point where drawing point has reached
    while(pt >= tp) {
      px = PARABOLA_A * pt * pt;
      py = 2 * PARABOLA_A * pt;

      glColor3f(0.5, 0.5, 0.5);

      glVertex2f(px * MULT_FACTOR, py * MULT_FACTOR);

      pt = pt - PARABOLA_DEL;
    }
glEnd();

// Animation: Updating the parabola parameter if the drawing hasn't finished.
if(tp <= PARABOLA_T && tp >= -PARABOLA_T) {
    tp = tp - PARABOLA_DEL;
    glutPostRedisplay();
}

    glFlush();
```

```
}

void generateConic() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);

    glLoadIdentity();
    glTranslatef(0.0f, yTranslation, transZ);
    glRotatef(rotAngle, 1, 0, 0);
    glColor3f(1, 1, 1);
    glutWireCone(1,2,150,50);

    if( rotAngle > maxAngle ) {
        rotAngle = rotAngle - 0.03;
        glutPostRedisplay();
    }

    if(rotAngle <= maxAngle && transZ <= maxTranslationZ) {
       transZ = transZ + 0.0005;
       glutPostRedisplay();
    }
    glFlush();
}

void reshapeConic(GLsizei width, GLsizei height) {
   if (height == 0) height = 1;

   GLfloat aspect = (GLfloat)width / (GLfloat)height;

   glViewport(0, 0, width, height);

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();

   gluPerspective(45.0f, aspect, 3.0f, 100.0f);
}

int main(int argc, char** argv) {
    int type;
        glutInit(&argc, argv);

    cout<< "Select option : \n 1) Sine \n 2) Cosine \n 3) Tangent \n 4) Circle \n 5) Ellipse \n 6)
Parabola \n "
        << "7) Generate Ellipse \n 8) Generate Circle \n 9) Generate Parabola \n 10) Generate
Hyperbola \n ";
    cin >> type;

        glutInitWindowSize(500, 500);
        glutInitWindowPosition(800, 100);
        glutCreateWindow("Project MathemGL: Display");

        switch(type) {
        case 1 :
        glutDisplayFunc(plotSine);
                        cout<<endl<<"lim (x->0) (sin x / x ) = 1"<<endl;
            break ;
        case 2 :
        glutDisplayFunc(plotCos);
        cout<<endl<<"lim (x->0) (cos x) = 1"<<endl;
                break ;
        case 3 :
        glutDisplayFunc(plotTan);
        cout<<endl<<"lim (x->0) (tan x / x ) = 1"<<endl;
                break ;
        case 4 :
        glutDisplayFunc(plotCircle);
        cout<<endl<<"Circle is the locus of all the points\n at equidistant from a given \nfixed
point, Center.";
                break ;
        case 5 :
        glutDisplayFunc(plotEllipse);
        cout<<endl<<"Ellipse is the locus of all the points\n such that sum of distances of the point
from \ntwo given fixed points, Focii is constant.";
                break ;
        case 6 :
        glutDisplayFunc(plotParabola);
        cout<<endl<<"Parabola is the locus of all the points\n at equidistant from a given \nfixed
point, Focus and a line, Directrix.";
                break ;
        case 7 :
            // Ellipse
            maxAngle = -150, maxTranslationZ = -2.2, yTranslation = -0.2;
            glutDisplayFunc(generateConic);
            glutReshapeFunc(reshapeConic);
            cout<<endl<<"Ellipse is a conic section \nobtained by cutting the cone by a plane at angle
< 90 wrt base.";
            break;
        case 8 :
            // Circle
            maxAngle = -180, maxTranslationZ = -2.2, yTranslation = 0;
            glutDisplayFunc(generateConic);
            glutReshapeFunc(reshapeConic);
            cout<<endl<<"Circle is a conic section \nobtained by cutting the cone by a plane parallel
to the base.";
            break;
        case 9 :
            // Parabola
            maxAngle = -110, maxTranslationZ = -2.2, yTranslation = -0.2;
            glutDisplayFunc(generateConic);
            glutReshapeFunc(reshapeConic);
            cout<<endl<<"Parabola is a conic section \nobtained by cutting the cone by a plane at
angle > 0 (!= 0) wrt its axis.";
            break;
        case 10 :
            // Hyperbola
            maxAngle = -90, maxTranslationZ = -2.4, yTranslation = -0.4;
            glutDisplayFunc(generateConic);
            glutReshapeFunc(reshapeConic);
            cout<<endl<<"Hyperbola is a conic section \nobtained by cutting the cone by a plane
parallel to its axis.";
            break;
        default:
            cout<<"Enter a value b/w 1 and 10"<<endl<<endl;
            break;
        }
        glutMainLoop();
        return 0;
}
```

## Limitations:

1) After selecting one animation, program needs to be run again for selecting other animation to be displayed.

2) For showing the cross-section of cone to generate different conic sections, clipping viewport has been used. But, the idea was to use a plane with some colour so that it is clearly visible that the cone is being cut by a plane to get the required conic section.

## Conclusion:

We have shown graphical proofs of standard limits using plot of trigonometric functions. We have also shown how different conic sections can be drawn using pencil, thread and fixed pins (with fixed lines in case of parabola). We have visualized how conic sections are formed from cross-section of a cone.

Group Members:

Astitva Srivastava (114CS0384)

Sunil Gudivada (114CS0385)