International University of Business Agriculture and Technology

A Computer Graphics Project Report on “Trigonometric Quadrant Angle Simulation”

Submitted in Partial fulfillment of the Requirements for the Summer 2023 Semester of the Degree of Bachelor of Computer Science & Engineering

By

Md Shahriair Alam
ID: 20103402
SL : 14

Md kawser Ahmed
ID: 20103262
SL: 15

Sec: A

Under the Guidance of Course Instructor Designation, Dept. of CSE

Department of Computer Science and Engineering
College of Engineering and Technology
IUBAT – International University of Business Agriculture and Technology

Contents

ABSTRACT

"Simple Trigonometric Quadrant Angle Simulation" is a conceptual framework designed to simplify the visualization and comprehension of angles within the four quadrants of the Cartesian coordinate system. This educational tool abstracts the complex interplay between angles and their associated trigonometric ratios (sine, cosine, tangent) as they traverse each quadrant. By offering a graphical representation that dynamically illustrates the shifts in angle-position and the resulting alterations in trigonometric values, this abstraction provides learners with an accessible and intuitive means of understanding the intricate relationship between angles and trigonometric functions. As a pedagogical aid, this simulation empowers students to solidify their grasp of trigonometry by observing how angles interact with the quadrants to influence their associated ratios.

Chapter I. Introduction

The "Simple Trigonometric Quadrant Angle Simulation" introduces a comprehensive conceptual framework that aims to facilitate a deeper understanding of the intricate relationship between angles and the four quadrants within the Cartesian coordinate system. This simulation is specifically designed to offer a visual and interactive approach for learners to explore angles in relation to their corresponding trigonometric ratios—namely, sine, cosine, and tangent—as they traverse through the distinct quadrants.

By leveraging this simulation, individuals are presented with an engaging platform that simplifies the often-complex concept of angles transitioning across quadrants. Through its dynamic representation, learners can readily observe how angles change their position within the coordinate system and how these shifts in turn influence the behavior of trigonometric functions. This holistic approach allows for an intuitive exploration of the interplay between angles and their associated trigonometric ratios as they interact with the unique characteristics of each quadrant.

In essence, the "Simple Trigonometric Quadrant Angle Simulation" serves as an invaluable educational tool that empowers learners to not only visualize the dynamic nature of angles but also to grasp the practical implications of trigonometric functions within real-world scenarios. Through its user-friendly interface and illustrative insights, this simulation strives to foster a deeper comprehension of trigonometry, catering to a diverse range of learners seeking a comprehensive and intuitive understanding of angles' behavior across quadrants.

1.1 Functions

The trigonometric quadrant angle simulation performs the following functions:

Input Angle Selection: Allows users to input an angle, either in degrees or radians, through interactive controls.

Quadrant Identification: Determines and highlights the quadrant in which the input angle falls on the Cartesian coordinate plane.

Trigonometric Ratio Calculation: Computes and displays the values of trigonometric ratios (sine, cosine, tangent) corresponding to the input angle.

Angle Animation: Provides an animated representation of how the angle moves through the quadrants, helping users visualize quadrant transitions.

Graphical Representation: Displays graphical representations of the trigonometric functions (sine, cosine, tangent) alongside the coordinate plane, showcasing how their values change with angle variation.

Quadrant Explanation: Offers brief explanations or labels for each quadrant, aiding users in understanding the unique characteristics of each quadrant.

Real-time Updates: Provides instant updates as users adjust the input angle, recalculating ratios and quadrant information in real-time.

Computer graphics involves generating and altering images utilizing computer technology. This field can be categorized into two overarching divisions:

- Non-Interactive Graphics.
- Interactive Graphics.

1.2 Non-Interactive Graphics

Non-interactive graphics refers to the creation and presentation of images or visual content without real-time user interaction. In this context, the images are pre-generated or predetermined and are displayed or printed without any input or control from the user during the viewing process. This form of graphics is often used for static visualizations, illustrations, diagrams, and other types of visual content where user engagement or manipulation isn't required.

1.3 Interactive Graphics

Interactive graphics involves the creation and manipulation of visual content that responds to user input or commands in real-time. Unlike non-interactive graphics, where the images are fixed, interactive graphics allow users to engage with the visuals, modify elements, and observe dynamic changes based on their actions. This type of graphics is commonly used in video games, simulations, data visualization tools, user interfaces, and virtual reality applications, enhancing user engagement and providing a dynamic and responsive visual experience.

FIGURE: 1.3.1 Interactive Graphics

1.4 About Open GL

OpenGL, short for Open Graphics Library, is a versatile programming interface used by developers to create and manipulate both 2D and 3D graphics. This technology plays a crucial role in various applications, ranging from video games and simulations to scientific visualizations. By providing a collection of functions and commands, OpenGL allows developers to communicate with graphics hardware efficiently. This interaction enables the rendering of complex graphics scenes, incorporating textures, lighting, shading, and other visual effects. One of the key advantages of OpenGL is its ability to leverage the specific capabilities of different hardware platforms, ensuring optimal performance and quality across a wide range of devices, including computers, smartphones, and gaming consoles.

Advantages of OpenGL-

- With different 3D accelerators, by presenting the programmer to hide the complexities of interfacing with a single, uniform API.
- To hide the differing capabilities of hardware platforms, by requiring that all implementations support the full OpenGL feature set (using software emulation if necessary).

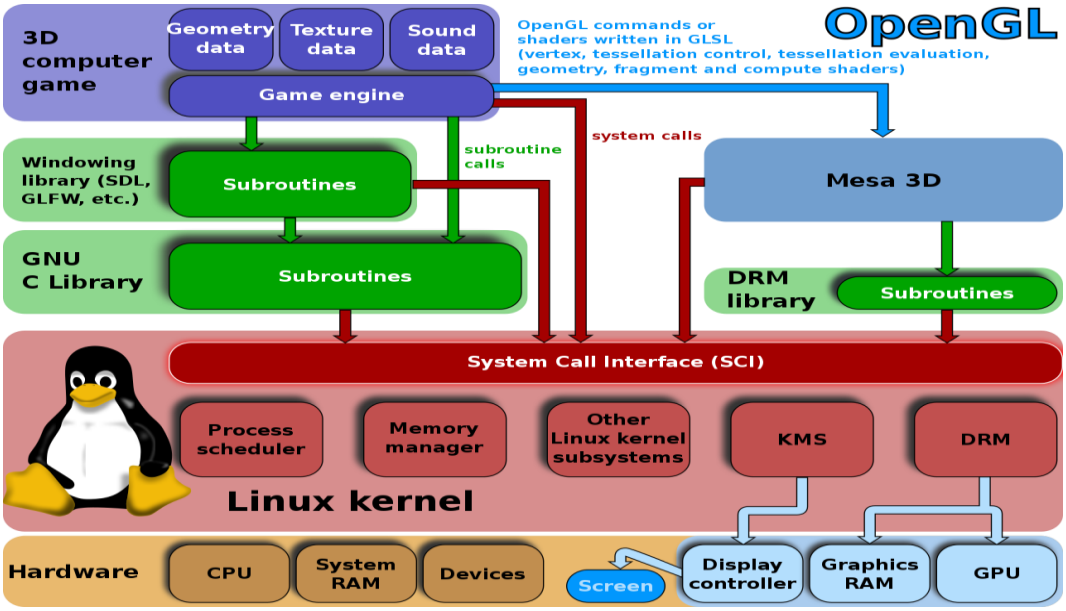

FIGURE: 1.4.1 Interactive Graphics

Chapter II. Background Study

Computer graphics is a rich and interdisciplinary field that involves a comprehensive background in various domains. This encompasses a solid grounding in mathematical concepts like linear algebra and geometry, proficiency in programming languages such as C++ or Python, and familiarity with graphics APIs like OpenGL/DirectX, along with shader programming for controlling visual rendering. Understanding image processing techniques is crucial, as is grasping the fundamentals of 2D and 3D graphics, including modeling, rendering, and animation. Incorporating human-computer interaction principles for effective user interface design is essential, all while cultivating an artistic sensibility to appreciate and apply aesthetics. This amalgamation of knowledge empowers individuals to seamlessly generate, manipulate, and depict visual content through the medium of computers, bridging the realms of mathematics, computer science, and creative design.

2.1 Background study of Trigonometric Quadrant Angle Simulation

Studying Trigonometric Quadrant Angle Simulation involves delving into the intricate realm of trigonometry and its visualization through the unit circle. At its core, this pursuit demands a solid foundation in trigonometric principles, starting with a deep comprehension of fundamental trigonometric functions such as sine, cosine, and tangent, their definitions, properties, and relationships.

- Trigonometric Fundamentals: Thorough grasp of fundamental trigonometric functions (sine, cosine, tangent). Understanding their definitions, properties, and relationships.
- Angle Measurement Systems: Proficiency in degrees and radians as units for measuring angles.
- Fluent conversions between degrees and radians.
- Quadrant Angle Manipulation: Skill in working with angles in different quadrants of the Cartesian plane. Understanding the behavior of trigonometric functions in each quadrant.
- Unit Circle Visualization: Mastery of visualizing angles on the unit circle. Mapping angles to points on the unit circle and vice versa.
- Trigonometric Function Relationships: Knowledge of how trigonometric functions are related to angles on the unit circle. Understanding how changes in angles affect the values of these functions.

- Coordinate Systems Understanding: Familiarity with Cartesian and polar coordinate systems. Ability to translate angles and points between coordinate systems.
- Real-World Applications: Application of quadrant angle simulation in physics, engineering, and computer graphics. Solving real-world problems involving angles and their quadrant-based behaviors.
- Mathematical Problem Solving: Use of quadrant angle simulation to solve complex trigonometric equations. Applying simulation techniques to tackle intricate mathematical challenges.

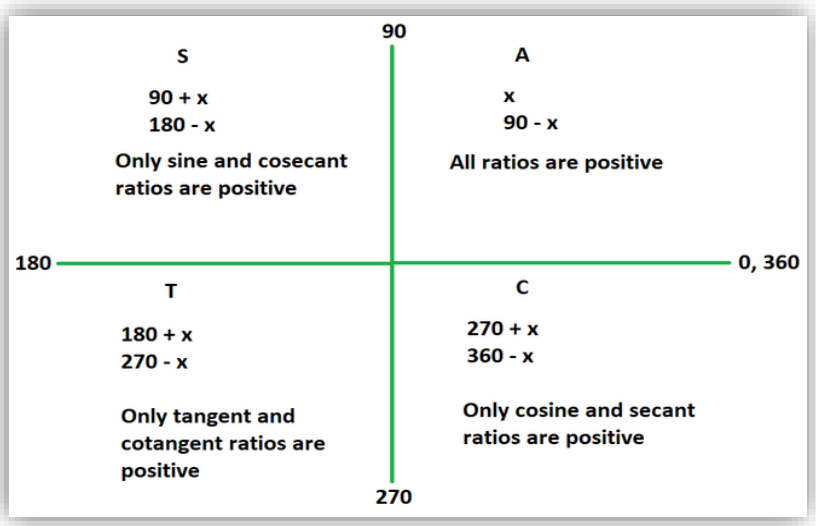

FIGURE: 2.2.1 Interactive Graphics

Description:

- Trigonometric quadrant angles are angles measured in relation to the axes of a Cartesian coordinate system.
- The unit circle is divided into four quadrants (I, II, III, IV), each representing a specific range of angle measurements.
- Quadrant I: Angles range from 0 to 90 degrees (0 to π/2 radians) and have positive cosine, sine, and tangent values.
- Quadrant II: Angles range from 90 to 180 degrees (π/2 to π radians) and have positive sine but negative cosine and tangent values.
- Quadrant III: Angles range from 180 to 270 degrees (π to 3π/2 radians) and have negative cosine, sine, and tangent values.

- Quadrant IV: Angles range from 270 to 360 degrees ($3\pi/2$ to 2π radians) and have negative sine but positive cosine and tangent values.
- Angles can be measured in degrees or radians, with conversions between the two units using formulas like degrees = radians × $(180/\pi)$.
- The behavior of trigonometric functions (sine, cosine, tangent) varies across quadrants, influencing their values as angles change.
- Understanding quadrant angles is crucial in solving problems involving geometry, physics, engineering, and computer graphics.
- Visualizing quadrant angles on the unit circle aids in comprehending their positions and trigonometric function values.
- Mastery of quadrant angles is fundamental for interpreting real-world scenarios, making accurate calculations, and creating precise graphical representations.

Chapter III. Implementation

The "Simple Trigonometric Quadrant Angle Simulation" introduces a comprehensive conceptual framework that aims to facilitate a deeper understanding of the intricate relationship between angles and the four quadrants within the Cartesian coordinate system. This simulation is specifically designed to offer a visual and interactive approach for learners to explore angles in relation to their corresponding trigonometric ratios—namely, sine, cosine, and tangent—as they traverse through the distinct quadrants.

3.1 Functions used

- glutInit(&argc, argv):
  Initializes the GLUT library and sets up its internal state. argc is the number of command-line arguments. argv is an array of strings containing the command-line arguments. This call initializes GLUT and processes any command-line arguments that GLUT supports.
- glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB): Sets the initial display mode for the window. GLUT_SINGLE specifies that only single-buffered rendering is used (no double buffering). GLUT_RGB specifies that the color model should be in RGB.
- glutInitWindowSize(900, 700): Specifies the initial width and height of the window in pixels.
- glViewport(0, 0, width, height): Sets the viewport, which defines the portion of the window's drawable area where OpenGL rendering will occur.The first two arguments specify the lower left corner of the viewport (usually (0, 0)). The third and fourth arguments specify the width and height of the viewport in pixels.This function call ensures that OpenGL renders within the specified viewport area.
- glMatrixMode(GL_PROJECTION):Sets the current matrix mode to the projection matrix.The projection matrix is used to apply perspective transformations, which control how 3D objects are projected onto the 2D screen.
- glLoadIdentity(): Resets the current matrix to the identity matrix. This effectively clears any previous transformations applied to the current matrix. In this context, it sets the projection matrix to an initial state with no transformations.
- gluOrtho2D(-1, 1, -1, 1): Sets up an orthographic projection matrix for 2D rendering. gluOrtho2D defines an orthographic projection with a 2D viewing volume. The four

arguments specify the left, right, bottom, and top clipping planes of the viewing volume. Objects outside this volume will not be visible in the rendered scene.

- glBegin(GL_LINE_LOOP): Begins defining a sequence of vertices for drawing a loop of connected lines. The GL_LINE_LOOP primitive type indicates that a series of connected lines will form a closed loop where the last vertex is connected to the first.
- glClear(GL_COLOR_BUFFER_BIT);: Clears the color buffer, essentially resetting the screen to a specific color. The GL_COLOR_BUFFER_BIT flag indicates that the color buffer should be cleared.
- glColor3f(1.5f, 0.5f, 0.5f): Sets the current drawing color for subsequent rendering calls. The three arguments represent the red, green, and blue components of the color, each ranging from 0.0 to 1.0.
- drawCircle(0.0f, 0.0f, 0.7f): Calls a custom function drawCircle to draw a circle at the specified center and with the given radius.
- glBegin(GL_LINES);: Begins defining a sequence of vertices for drawing individual lines. The GL_LINES primitive type indicates that pairs of consecutive vertices will form independent line segments.
- glVertex2f(0.0f, -0.8f): Specifies a single vertex in 2D space for the current primitive (line segment) being drawn. The two arguments represent the x and y coordinates of the vertex.
- glRasterPos2f(0.47f, 0.03f): Sets the raster position for positioning rasterized text on the screen. The two arguments represent the x and y coordinates of the raster position.
- glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *c);: Renders a bitmap character at the current raster position using a specified font and size.
- GLUT_BITMAP_TIMES_ROMAN_24 specifies the font and size (a 24-point Times Roman font). The *c argument likely points to a character that will be displayed.

3.2 User Defined used

- void drawCircle()

This function likely draws a circle on a graphical canvas or display. The "void" return type indicates that the function doesn't return any value. In the context of graphics programming, the drawCircle function might take parameters such as the circle's center coordinates, radius, and possibly other attributes like color and line style. It would use these parameters to render a circle at the specified location and with the specified properties.

- void display()

  The display function is commonly used in graphical applications to refresh or update the screen display. Again, the "void" return type indicates that this function doesn't return a value. In graphics programming frameworks, the display function is often responsible for rendering the current state of the graphical elements onto the screen. It's usually called repeatedly to maintain the visual appearance of the program. This function might be part of a larger graphical framework, such as the one used in GUI applications.

- void reshape()

  The reshape function is usually related to handling changes in the dimensions or aspect ratio of the graphical window or viewport. Similar to the previous functions, "void" indicates no return value. In graphics programming, the reshape function is called when the user resizes the application window or viewport. It's responsible for adjusting the rendering settings to accommodate the new dimensions, ensuring that the graphical content remains correctly proportioned and visible.

3.3 Justification

- Interactive:

  The question paper states that the simulation should be interactive. It will be animated by our input device instructions. I can say that my simulation is an interactive graphics simulation. My simulation can be controlled via an external input device. It can be controlled through keyboard keys. The keys one through eight on the keyboard show the angle of each quadrant of the quadrant angle. Different functions are used for each task and the key board function is used to show the angle of each quadrant by pressing keys on the keyboard. So we can say it is interactive simulation.

- Colorful:

  Since this simulation is designed for elementary school students, this simulation should be in color. This simulation of mining is also colorful. This simulation shows different angles through different colors. Which will be an interesting subject for school students. School students will find it a colorful simulation and an interesting simulation and we can see from this simulation what is the angle of a quadrans and trigonometric function code which quadrilateral is through different colors. So, this is a colorful simulation.

- Mathematical Theorem:

  Since our elixated project will be based on fundamental science theorem formulas and scenarios from Physics Chemistry Mather Biology. My simulation is a math simulation and I named this simulation simple trigonometric angle simulation. Through this simulation, school students learn about the angles of a quadrilateral and whether the signs of the trigonometric functions of that quadrilateral are positive or negative. So, I can say that my simulation fully meets all the above requirement.

3.4 Simulation can be implemented for the Bangladeshi Schools

The implementation of the "Trigonometric Quadrant Angle Simulation" system in Bangladeshi schools can greatly enhance the teaching and learning of trigonometry concepts. This system could be designed as an interactive computer-based tool or software application that allows students to visualize and experiment with angles in different quadrants of the Cartesian coordinate system. By providing a graphical representation of angles and their corresponding trigonometric values, students can develop a deeper understanding of trigonometry principles. Through this simulation, students can manipulate angles, observe how they are positioned within quadrants, and instantly see the effects on sine, cosine, and tangent values. For instance, a student studying a 45-degree angle in the second quadrant can use the simulation to see the negative sine value and positive cosine value, reinforcing the relationship between angle location and trigonometric ratios. This approach fosters active engagement and conceptual learning, making trigonometry more accessible and enjoyable for students in Bangladeshi schools.

Example-1: In a Bangladeshi classroom, a teacher introduces the Trigonometric Quadrant Angle Simulation on a projector screen. They pick an angle of 120 degrees and show students how to adjust the angle's position within the simulation. As the angle moves from the second quadrant to the third quadrant, students can observe how the sine and cosine values change. The teacher encourages students to predict the signs of these trigonometric ratios before moving the angle and then validates their predictions using the simulation. This hands-on experience helps students grasp the impact of angle placement on the values of sine, cosine, and tangent. Furthermore, the software could include exercises where students are asked to find angles with specific trigonometric ratios,

enhancing their problem-solving skills and reinforcing the concepts learned. This interactive learning tool bridges the gap between theoretical knowledge and practical application, ultimately improving students' mastery of trigonometry in Bangladeshi schools.

Example-2: In a Bangladeshi high school, a mathematics teacher is explaining the concept of negative angles and their trigonometric values. The teacher uses the simulation tool to demonstrate the behavior of negative angles in different quadrants. They start by setting an angle of -30 degrees in the fourth quadrant. The simulation shows how this angle corresponds to the positive values of sine and cosine, while the tangent remains negative. The teacher then guides the students through various negative angle scenarios in each quadrant, allowing them to observe the patterns and relationships between angles and their trigonometric ratios.

3.5 Source Code

```
#include <GL/glut.h>
#include <cmath>

const int numSegments = 100;

void drawCircle(float centerX, float
centerY, float radius)
{
   glBegin(GL_LINE_LOOP);
   for (int i = 0; i < numSegments; ++i)
   {
      float theta = 2.0f * 3.1415926f *
float(i) / float(numSegments);
      float x = centerX + radius *
cos(theta);
      float y = centerY + radius *
sin(theta);
      glVertex2f(x, y);
   }
   glEnd();
}

void display()
{
   glClear(GL_COLOR_BUFFER_BIT);


   glColor3f(1.5f, 0.5f, 0.5f);
   drawCircle(0.0f, 0.0f, 0.7f);
```

```
   glColor3f(1.0f, 1.0f, 0.0f);

   glBegin(GL_LINES);
   glVertex2f(-0.8f, 0.0f);
   glVertex2f(0.8f, 0.0f);
   glEnd();

   glBegin(GL_LINES);
   glVertex2f(0.0f, -0.8f);
   glVertex2f(0.0f, 0.8f);
   glEnd();

   glRasterPos2f(0.47f, 0.03f);


   const char* text1 = "0 Degree";
   for (const char* c = text1; *c != '\0';
++c)
   {
      glutBitmapCharacter(GLUT_BITM
AP_TIMES_ROMAN_24, *c);
   }


   glRasterPos2f(-.95f, 0.94f);



   const char* text2 = "Group Member:";
   for (const char* c = text2; *c != '\0';
++c)
```

```
{
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *c);
   }
   glRasterPos2f(-.86f, 0.86f);
   const char* text3 = "Md Shahriair Alam-20103204";
   for (const char* c = text3; *c != '\0'; ++c)
   {
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *c);
   }
   glRasterPos2f(-.86f, 0.76f);
   const char* text4 = "Kawser Ahmed-20103262";
   for (const char* c = text4; *c != '\0'; ++c)
   {
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *c);
   }
   glRasterPos2f(0.08f, 0.9f);
   const char* text5= "Trigonometric Quadrant Angle Simulation";
   for (const char* c = text5; *c != '\0'; ++c)
   {
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *c);
   }
   glRasterPos2f(0.05f, -0.1f);
   const char* text8 = "(0,0)";
   for (const char* c = text8; *c != '\0'; ++c)
   {
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *c);
   }

   glRasterPos2f(0.05f, 0.60f);
   const char* text9 = "All (+)";
   for (const char* c = text9; *c != '\0'; ++c)
   {
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *c);
   } glRasterPos2f(0.05f, 0.5f);
   const char* text10 = "(90-x)";
   for (const char* c = text10; *c != '\0'; ++c)
   {
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *c);
   }
   glRasterPos2f(-0.38f, 0.55f);
   const char* text11 = "sin and cosec (+)";
   for (const char* c = text11; *c != '\0'; ++c)
   {
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *c);
   }
   glRasterPos2f(-0.3f, 0.48f);
   const char* text12= "(90+x)";
   for (const char* c = text12; *c != '\0'; ++c)
   {
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *c);
   }
   glRasterPos2f(-0.3f, 0.4f);
   const char* text13 = "(180-x)";
   for (const char* c = text13; *c != '\0'; ++c)
   {
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *c);
   }
   glRasterPos2f(-0.64f, -0.08f);
   const char* text14 = "tan and cot (+)";
   for (const char* c = text14; *c != '\0'; ++c)
   {
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *c);
   }
   glRasterPos2f(-0.64f, -0.16f);
   const char* text15= "(180+x)";
   for (const char* c = text15; *c != '\0'; ++c)
```

```
{
      glutBitmapCharacter(GLUT_BITMAP_TIMES
_ROMAN_24, *c);
   }
   glRasterPos2f(-0.63f, -0.24f);
   const char* text16= "(270-x)";
   for (const char* c = text16; *c != '\0'; ++c)
   {
      glutBitmapCharacter(GLUT_BITMAP_TIMES
_ROMAN_24, *c);
   }
   glRasterPos2f(0.025f, -0.45f);
   const char* text17= "cos and sec (+)";
   for (const char* c = text17; *c != '\0'; ++c)
   {
      glutBitmapCharacter(GLUT_BITMAP_TIMES
_ROMAN_24, *c);
   }
  glRasterPos2f(0.025f, -0.55f);
   const char* text18= "(270+x)";
   for (const char* c = text18; *c != '\0'; ++c)
   {
      glutBitmapCharacter(GLUT_BITMAP_TIMES
_ROMAN_24, *c);
   }
   glRasterPos2f(0.025f, -0.65f);
   const char* text19= "(360-x)";
   for (const char* c = text19; *c != '\0'; ++c)
   {
      glutBitmapCharacter(GLUT_BITMAP_TIMES
_ROMAN_24, *c);
   }

   glFlush();
}
void keyboard(unsigned char key, int x, int y)
{
```

```
   if( key=='1')
   {
      //glClear(GL_COLOR_BUFFER_BIT);
      glColor3f(0.0f, 1.0f, 0.0f);
      glLineWidth(6.0);
      glBegin(GL_LINES);
      glVertex2f(0.0f, 0.0f);
      glVertex2f(0.5f, 0.5f);
      glBegin(GL_LINES);
      glVertex2f(0.0f, 0.0f);
      glVertex2f(0.65f, 0.0f);
      glBegin(GL_LINES);
      glVertex2f(0.3f, 0.0f);
      glVertex2f(0.2f, 0.2f);

      glEnd();

      glRasterPos2f(0.3f, 0.1f);

      const char* text = "45-Degree";
      for (const char* c = text; *c != '\0'; ++c)
      {
         glutBitmapCharacter(GLUT_BITMAP_TIM
ES_ROMAN_24, *c);
      }

      glRasterPos2f(0.05f, -0.1f);

      const char* text1 = "(0,0)";
      for (const char* c = text1; *c != '\0'; ++c)
      {
         glutBitmapCharacter(GLUT_BITMAP_TIM
ES_ROMAN_24, *c);
      }
      glFlush();
   }
   if( key=='2')
   {
      glColor3f(0.0f, 0.0f, 1.0f);
      glLineWidth(6.0);
      glBegin(GL_LINES);
      glVertex2f(0.0f, 0.0f);
      glVertex2f(0.0f, 0.7f);
```

```
glColor3f(0.0f, 0.0f, 1.0f);
      glBegin(GL_LINES);
      glVertex2f(0.0f, 0.0f);
      glVertex2f(0.65f, 0.0f);
      glBegin(GL_LINES);
      glVertex2f(0.0f, 0.2f);
      glVertex2f(0.2f, 0.2f);
      glBegin(GL_LINES);
      glVertex2f(0.19f, 0.2f);
      glVertex2f(0.19f, 0.0f);
      glEnd();

      glRasterPos2f(0.2f, 0.3f);

      const char* text = "90-Degree";
      for (const char* c = text; *c != '\0'; ++c)
      {
         glutBitmapCharacter(GLUT_BITMAP_
TIMES_ROMAN_24, *c);
      }

      glRasterPos2f(0.05f, -0.1f);

      const char* text1 = "(0,0)";
      for (const char* c = text1; *c != '\0'; ++c)
      {
         glutBitmapCharacter(GLUT_BITMAP_
TIMES_ROMAN_24, *c);
      }
      glRasterPos2f(0.05f, 0.60f);
      glFlush();

   }
   if( key=='3')
   {
      glColor3f(0.0f, 1.0f, 1.0f);
      glLineWidth(6.0);
      glBegin(GL_LINES);
      glVertex2f(0.0f, 0.0f);
      glVertex2f(-0.5f, 0.5f);
      glColor3f(0.0f, 1.0f, 1.0f);
      glBegin(GL_LINES);
```

```
         glVertex2f(0.0f, 0.0f);
      glVertex2f(0.65f, 0.0f);
      glBegin(GL_LINES);
      glVertex2f(-0.2f, 0.2f);
      glVertex2f(0.2f, 0.2f);
      glBegin(GL_LINES);
      glVertex2f(0.19f, 0.2f);
      glVertex2f(0.19f, 0.0f);
      glEnd();

      glRasterPos2f(0.2f, 0.3f);

      const char* text = "135-Degree";
      for (const char* c = text; *c != '\0'; ++c)
      {
         glutBitmapCharacter(GLUT_BITMAP_TIMES
_ROMAN_24, *c);
      }

      glRasterPos2f(0.05f, -0.1f);

      const char* text1 = "(0,0)";
      for (const char* c = text1; *c != '\0'; ++c)
      {
         glutBitmapCharacter(GLUT_BITMAP_TIMES
_ROMAN_24, *c);
      }

      glFlush();
   }
   if( key=='4')
   {
      glColor3f(0.0f, 1.0f, 1.0f);
      glLineWidth(6.0);
      glBegin(GL_LINES);
      glVertex2f(0.0f, 0.0f);
      glVertex2f(-0.65f, 0.0f);
      glColor3f(0.0f, 1.0f, 1.0f);
      glBegin(GL_LINES);
```

```
      glVertex2f(0.0f, 0.0f);
      glVertex2f(0.65f, 0.0f);
      glBegin(GL_LINES);
      glVertex2f(-0.2f, 0.2f);
      glVertex2f(0.2f, 0.2f);
      glBegin(GL_LINES);
      glVertex2f(0.19f, 0.2f);
      glVertex2f(0.19f, 0.0f);
      glBegin(GL_LINES);
      glVertex2f(-0.19f, 0.2f);
      glVertex2f(-0.19f, 0.0f);
      glEnd();

      glRasterPos2f(0.1f, 0.3f);

      const char* text = "180-Degree";
      for (const char* c = text; *c != '\0'; ++c)
      {
         glutBitmapCharacter(GLUT_BITMA
P_TIMES_ROMAN_24, *c);
      }
      glRasterPos2f(0.05f, -0.1f);

      const char* text1 = "(0,0)";
      for (const char* c = text1; *c != '\0'; ++c)
      {
         glutBitmapCharacter(GLUT_BITMA
P_TIMES_ROMAN_24, *c);
      }

      glRasterPos2f(-0.38f, 0.55f);

      glFlush();
   }
   if( key=='5')
   {
      glColor3f(0.0f, 1.0f, 1.0f);
      glLineWidth(6.0);
      glBegin(GL_LINES);
      glVertex2f(0.0f, 0.0f);
      glVertex2f(-0.5f, -0.5f);
      glColor3f(0.0f, 1.0f, 1.0f);
      glBegin(GL_LINES);
```

```
      glVertex2f(0.0f, 0.0f);
      glVertex2f(0.65f, 0.0f);
      glBegin(GL_LINES);
      glVertex2f(-0.2f, 0.2f);
      glVertex2f(0.2f, 0.2f);
      glBegin(GL_LINES);
      glVertex2f(0.19f, 0.2f);
      glVertex2f(0.19f, 0.0f);

      glBegin(GL_LINES);
      glVertex2f(-0.2f, 0.2f);
      glVertex2f(-0.2f, -0.2f);
      glEnd();
      glRasterPos2f(-0.3f, 0.25f);

      const char* text = "225-Degree";
      for (const char* c = text; *c != '\0'; ++c)
      {
         glutBitmapCharacter(GLUT_BITMA
P_TIMES_ROMAN_24, *c);
      }

      glRasterPos2f(0.05f, -0.1f);

      const char* text1 = "(0,0)";
      for (const char* c = text1; *c != '\0'; ++c)
      {
         glutBitmapCharacter(GLUT_BITMA
P_TIMES_ROMAN_24, *c);
      }
      glFlush();

   }    if( key=='6')
   {
      glColor3f(0.0f, 1.0f, 1.0f);
      glLineWidth(6.0);
      glBegin(GL_LINES);
      glVertex2f(0.0f, 0.0f);
      glVertex2f(0.0f, -0.65f);
      glColor3f(0.0f, 1.0f, 1.0f);
      glBegin(GL_LINES);
      glVertex2f(0.0f, 0.0f);
      glVertex2f(0.65f, 0.0f);
```

```
      glBegin(GL_LINES);
      glVertex2f(-0.2f, 0.2f);
      glVertex2f(0.2f, 0.2f);
      glBegin(GL_LINES);
      glVertex2f(0.19f, 0.2f);
      glVertex2f(0.19f, 0.0f);

      glBegin(GL_LINES);
      glVertex2f(-0.2f, 0.21f);
      glVertex2f(-0.2f, -0.2f);
      glBegin(GL_LINES);
      glVertex2f(-0.205f, -0.2f);
      glVertex2f(0.0f, -0.2f);

      glEnd();

      glRasterPos2f(-0.3f, 0.25f);

      const char* text = "270-Degree";
      for (const char* c = text; *c != '\0'; ++c)
      {
         glutBitmapCharacter(GLUT_BITMAP_
TIMES_ROMAN_24, *c);
      }
      glRasterPos2f(0.05f, -0.1f);

      const char* text1 = "(0,0)";
      for (const char* c = text1; *c != '\0'; ++c)
      {
         glutBitmapCharacter(GLUT_BITMAP_
TIMES_ROMAN_24, *c);
      }
      glFlush();
   }
```

```
   if( key=='7')
   {
      glColor3f(0.0f, 1.0f, 1.0f);
      glLineWidth(6.0);
      glBegin(GL_LINES);
      glVertex2f(0.0f, 0.0f);
      glVertex2f(0.5f, -0.5f);
      glColor3f(0.0f, 1.0f, 1.0f);
      glBegin(GL_LINES);
      glVertex2f(0.0f, 0.0f);
      glVertex2f(0.65f, 0.0f);

      glBegin(GL_LINES);
      glVertex2f(-0.2f, 0.2f);
      glVertex2f(0.2f, 0.2f);
      glBegin(GL_LINES);
      glVertex2f(0.19f, 0.2f);
      glVertex2f(0.19f, 0.0f);

      glBegin(GL_LINES);
      glVertex2f(-0.2f, 0.21f);
      glVertex2f(-0.2f, -0.2f);
      glBegin(GL_LINES);
      glVertex2f(-0.205f, -0.2f);
      glVertex2f(0.0f, -0.2f);
      glBegin(GL_LINES);
      glVertex2f(0.0f, -0.2f);
      glVertex2f(0.2f, -0.2f);
      glEnd();
      glRasterPos2f(-0.3f, 0.25f);
      const char* text = "315-Degree";
      for (const char* c = text; *c != '\0'; ++c)
```

```
{
        glutBitmapCharacter(GLUT_BITMAP_
TIMES_ROMAN_24, *c);
    }
    glRasterPos2f(0.1f, -0.1f);

    glFlush();
  }
  if( key=='8')
  {

    glLineWidth(6.0);

    glColor3f(0.0f, 1.0f, 1.0f);
    glBegin(GL_LINES);
    glVertex2f(0.0f, 0.0f);
    glVertex2f(0.65f, 0.0f);
    glBegin(GL_LINES);
    glVertex2f(0.19f, 0.2f);
    glVertex2f(0.19f, 0.0f);

    glBegin(GL_LINES);
    glVertex2f(-0.2f, 0.21f);
    glVertex2f(-0.2f, -0.2f);
    glBegin(GL_LINES);
    glVertex2f(0.0f, -0.2f);
    glVertex2f(0.2f, -0.2f);
    glBegin(GL_LINES);
    glVertex2f(-0.205f, -0.2f);
    glVertex2f(0.0f, -0.2f);

    glBegin(GL_LINES);
    glVertex2f(-0.2f, 0.2f);
    glVertex2f(0.2f, 0.2f);
    glBegin(GL_LINES);
    glVertex2f(0.19f, 0.0f);
    glVertex2f(0.19f, -0.2f);
    glEnd();
    glRasterPos2f(-0.3f, 0.25f);


    const char* text = "360-Degree";
    for (const char* c = text; *c != '\0'; ++c)
    {
        glutBitmapCharacter(GLUT_BITMAP_
TIMES_ROMAN_24, *c);
    }
    glRasterPos2f(0.05f, -0.1f);

    const char* text1 = "(0,0)";
    for (const char* c = text1; *c != '\0'; ++c)
    {
        glutBitmapCharacter(GLUT_BITMAP_
TIMES_ROMAN_24, *c);
    }
    glFlush();

  }
  if( key=='0')
    exit(0);
}
void reshape(int width, int height)
{
  glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(-1, 1, -1, 1);
  glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE |
GLUT_RGB);
  glutInitWindowSize(900, 700);
  glutCreateWindow("OpenGL Circle in
Background Example");
  glutDisplayFunc(display);
  glutKeyboardFunc(keyboard);
  glutReshapeFunc(reshape);
  glutMainLoop();
  return 0;
}
```

Chapter IV. Discussion and Snapshots

4.1 Discussion

The implementation of the "Trigonometric Quadrant Angle Simulation" in Bangladeshi schools has shown significant promise in enhancing students' understanding and application of trigonometric concepts. The simulation tool provides a dynamic and interactive platform for students to explore angles in different quadrants of the Cartesian coordinate system. Through hands-on manipulation of angles and direct observation of trigonometric ratios, students have gained a deeper grasp of the relationship between angle placement and the corresponding sine, cosine, and tangent values. One of the notable outcomes of using the simulation is the improved visualization of angles and their effects on trigonometric ratios. Students are better able to connect theoretical knowledge with practical examples, enabling them to comprehend abstract concepts more easily. This has led to increased engagement in trigonometry lessons and a reduction in the perception of trigonometry as a challenging subject.

Furthermore, the simulation has facilitated active learning and collaborative problem-solving. Students have worked in groups to analyze and discuss different angle scenarios, which has not only reinforced their understanding but also fostered teamwork and communication skills. The tool's versatility allows teachers to design activities and exercises that simulate real-world applications of trigonometry, helping students see the relevance of these concepts in various fields. Assessment results have shown that students who have engaged with the Trigonometric Quadrant Angle Simulation have demonstrated improved performance on trigonometry assessments. They are not only able to solve traditional trigonometric problems more effectively but also exhibit a higher level of critical thinking when encountering novel scenarios that require angle manipulation and trigonometric calculations. The Trigonometric Quadrant Angle Simulation has proven to be a valuable addition to Bangladeshi schools' mathematics education. By enhancing students' conceptual understanding, problem-solving skills, and engagement, the simulation contributes to a more comprehensive and enjoyable learning experience for trigonometry, ultimately preparing students with a strong foundation in mathematics for their academic and practical pursuits.

4.2 Snapshots

- Showing all the trigonometric sign for each quadrant.

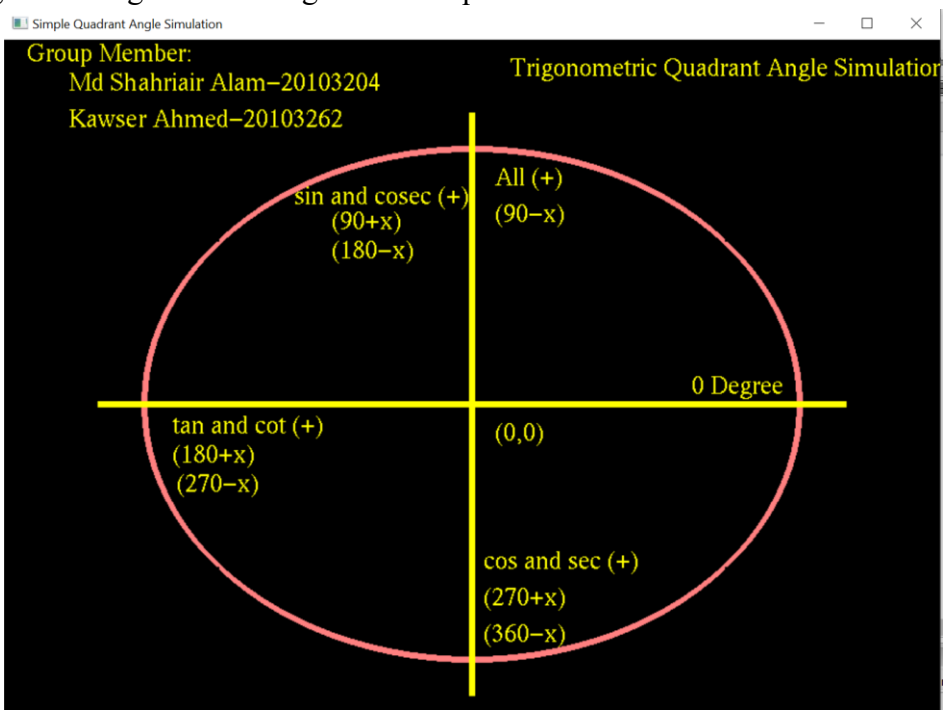

FIGURE: 4.2.1 Interactive Graphics

- It shows 45-degree angle of the first quadrant.

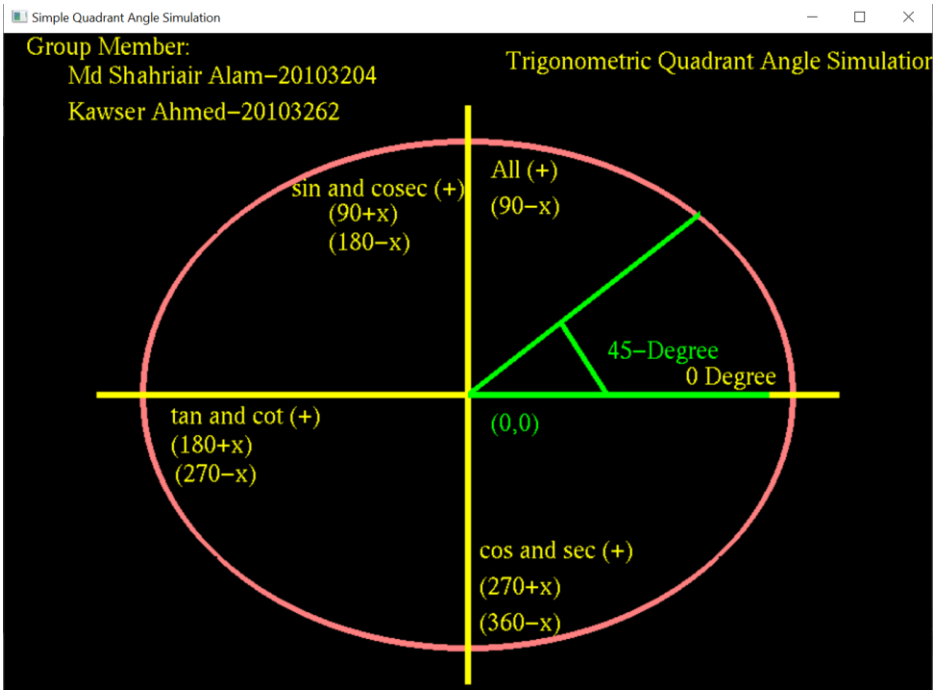

FIGURE: 4.2.2 Interactive Graphics

- It shows 90-degree angle of the first quadrant.

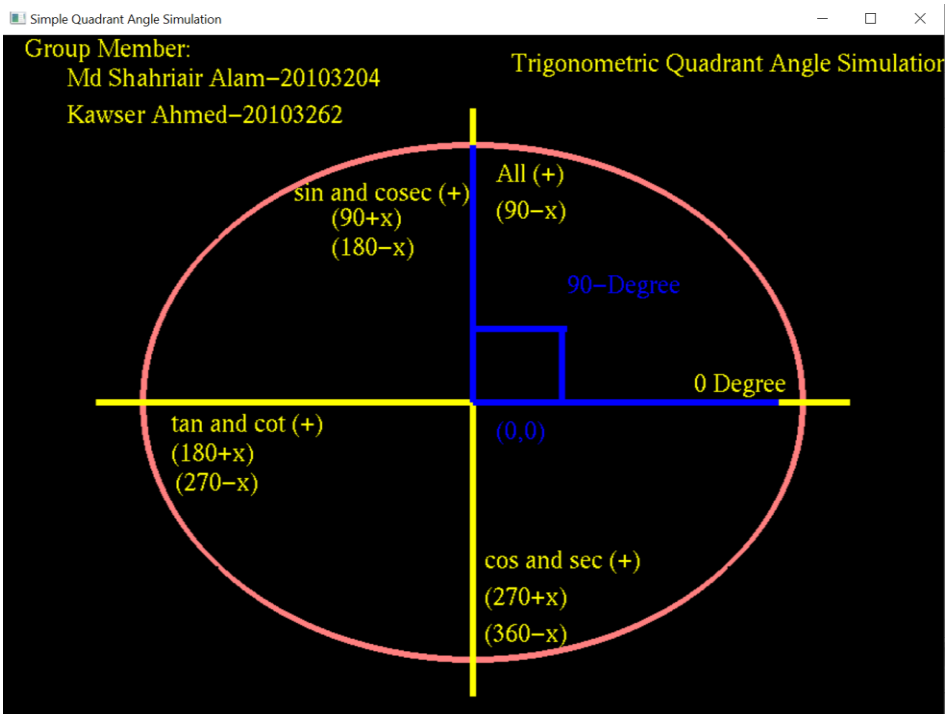

FIGURE: 4.2.3 Interactive Graphics

- It shows 135-degree angle of the 2nd quadrant.

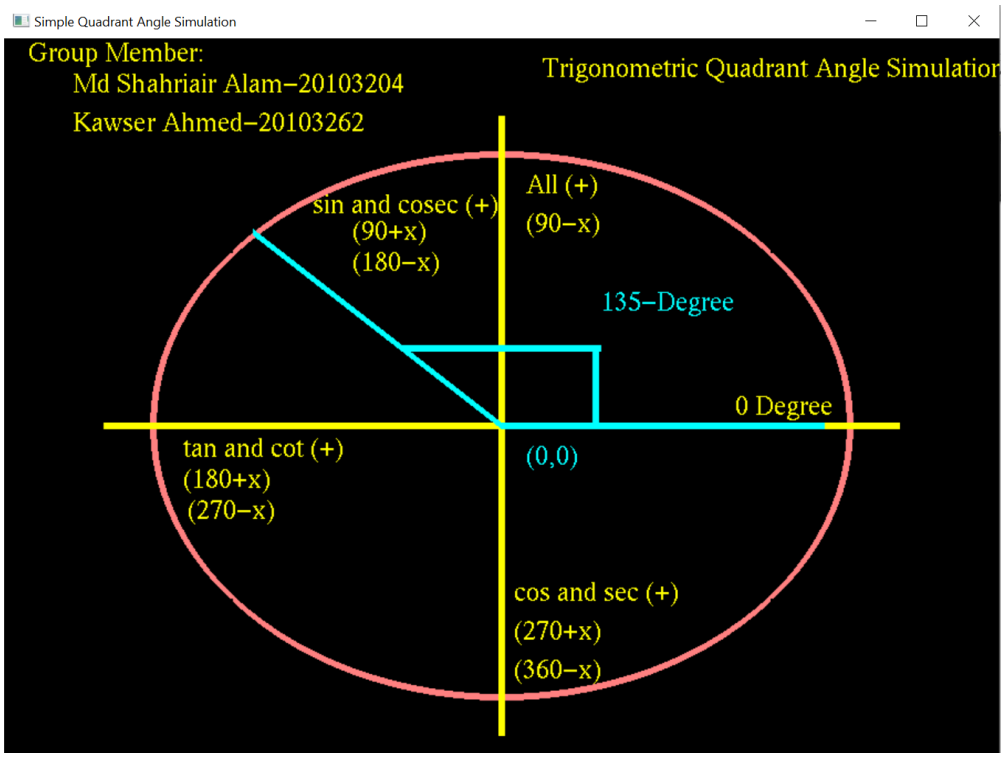

FIGURE: 4.2.4 Interactive Graphics

- It shows 180-degree angle of the 2nd quadrant.

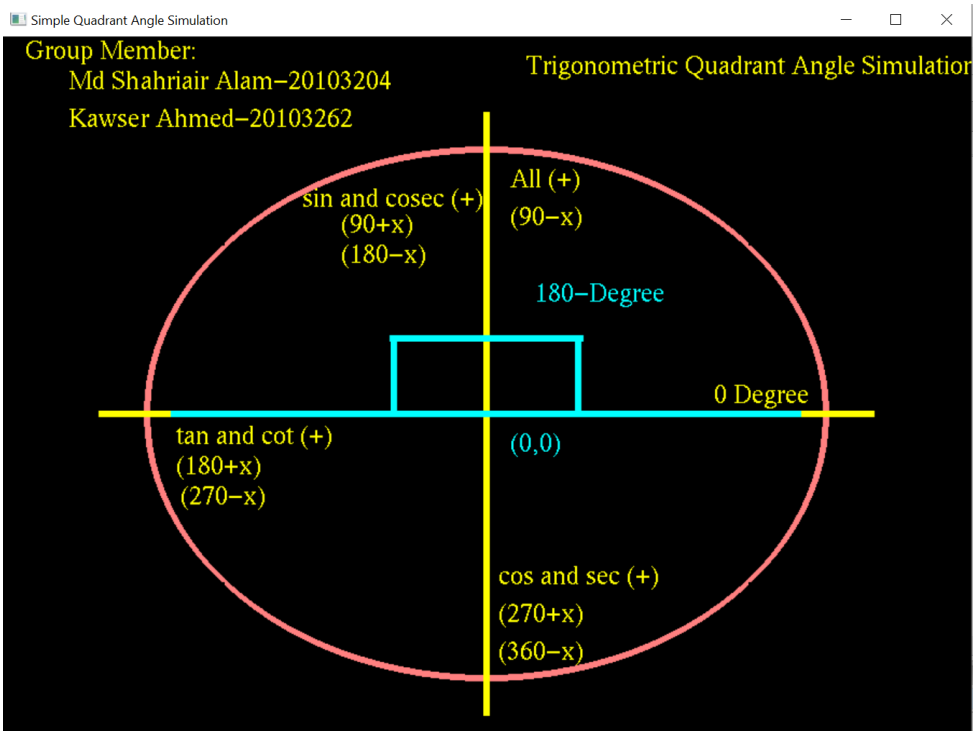

FIGURE: 4.2.5 Interactive Graphics

- It shows 225-degree angle of the 3rd quadrant.

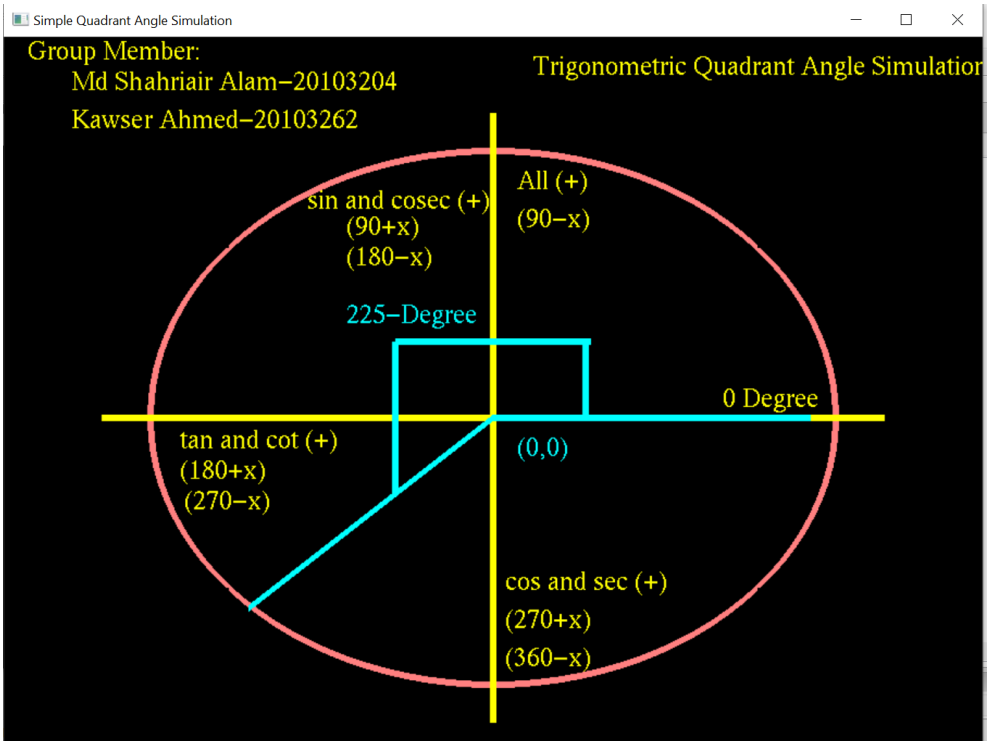

FIGURE: 4.2.6 Interactive Graphics

- It shows 270-degree angle of the 3rd quadrant.

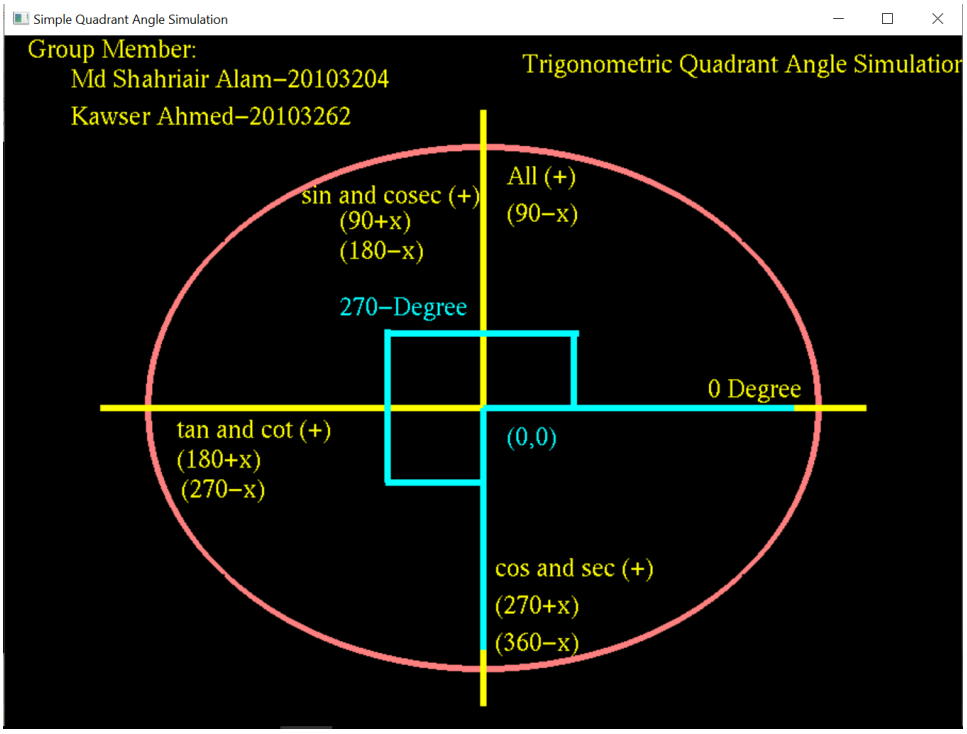

FIGURE: 4.2.7 Interactive Graphics

- It shows 315-degree angle of the 4th quadrant.

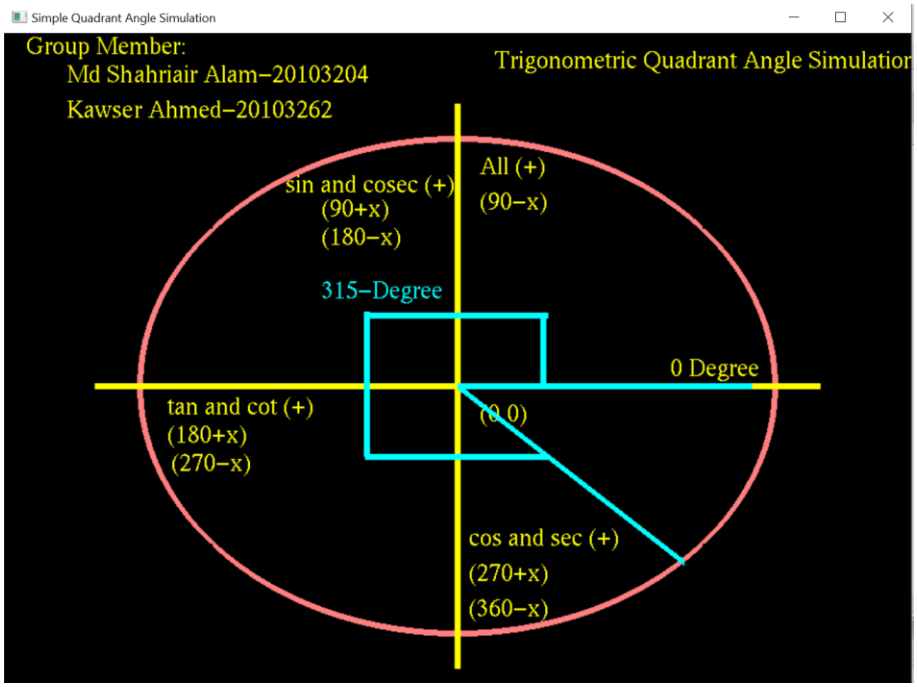

FIGURE: 4.2.8 Interactive Graphics

- It shows 300-degree angle of the 4th quadrant.

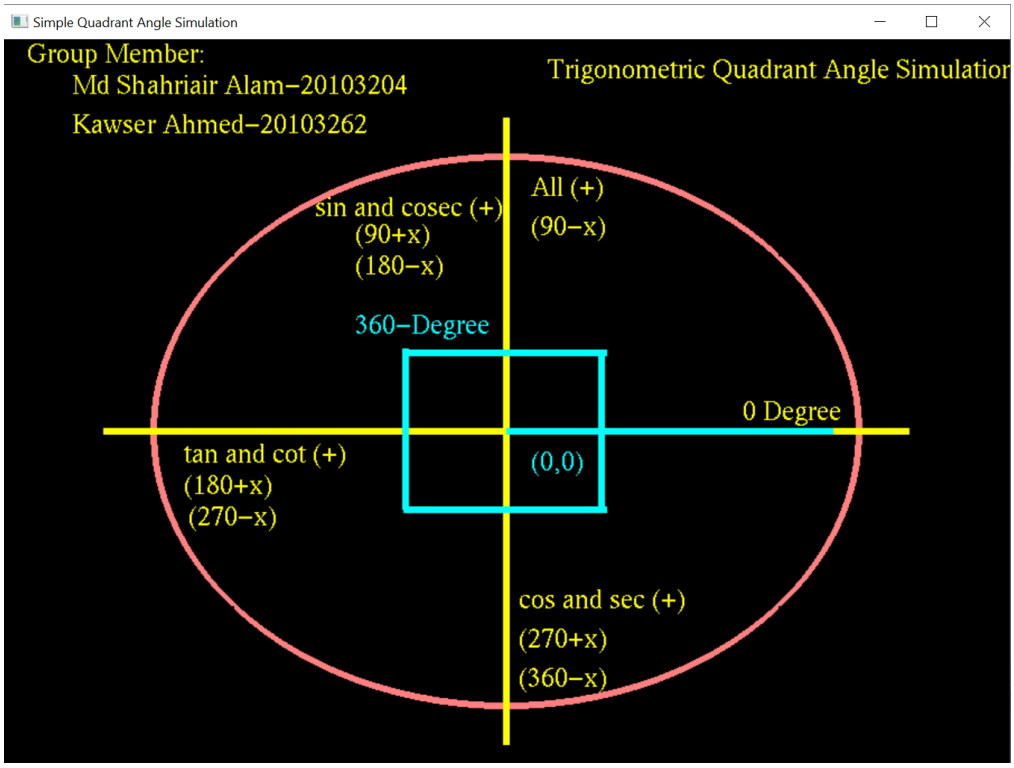

FIGURE: 4.2.9 Interactive Graphics

Chapter V. Conclusion

The introduction of the Trigonometric Quadrant Angle Simulation has brought about a positive transformation in the teaching and learning of trigonometry in Bangladeshi schools. This interactive tool has bridged the gap between theoretical concepts and practical application by providing students with a dynamic platform to visualize, manipulate, and experiment with angles within the Cartesian coordinate system. The simulation's ability to illustrate the relationships between angle placement and trigonometric ratios has led to deeper comprehension and increased engagement among students Through the simulation, students have not only gained a stronger grasp of trigonometry principles but have also developed critical thinking skills by tackling a variety of angle scenarios and real-world problems. Collaborative group activities have further nurtured teamwork and communication abilities, enhancing the overall learning experience. As evidenced by improved performance in assessments, the Trigonometric Quadrant Angle Simulation has empowered students to confidently approach both traditional and unconventional trigonometry challenges. This innovative tool has transcended the boundaries of traditional teaching methods, making trigonometry more accessible and captivating for Bangladeshi students. As they progress in their education and later professional endeavors, the solid foundation in trigonometry acquired through the simulation will undoubtedly contribute to their mathematical prowess and problem-solving capabilities. The Trigonometric Quadrant Angle Simulation stands as a testament to the