Lab3 graphics

Name : Abdelaziz Mohamed Abdelaziz.

I.D : 19015941.

Source code:

We add some variables like (filled) which Boolean variable which detect whether the sphere is filled or not and variable issphere (in which)detect whether it is sphere or not . and number of turns and num vertices and radius and pitch .

```
13
14  #define _USE_MATH_DEFINES
15
16  #include <cmath>
17  #include <iostream>
18
19  #include <GL/glew.h>
20  #include <GL/freeglut.h>
21
22  // Globals.
23  static float R = 5.0; // Radius of hemisphere.
24  static int p = 6; // Number of longitudinal slices.
25  static int q = 4; // Number of latitudinal slices.
26  static float Xangle = 0.0, Yangle = 0.0, Zangle = 0.0; // Angles to rotate hemisphere.
27  static float offset = -10;
28  static float spinSpeed = 5;
29  static float prev_time = 0;
30  bool filled = false;
31  bool isSphere = false;
32  const int num_turns = 5;
33  int num_vertices = 200;
34  float radius = 2.0f;
35  float pitch = 1.0f;
36
37  // Initialization routine.
38  void setup(void)
39  {
40      glClearColor(1.0, 1.0, 1.0, 0.0);
41  }
42
```

Here we see if it is filled or not if it is filled we use GL_FILL and if not we use GL_LINE

```
43  // Drawing routine.
44  void drawScene(void)
45  {
46
47      int  i, j;
48
49      glClear(GL_COLOR_BUFFER_BIT);
50
51      glLoadIdentity();
52
53      // Command to push the hemisphere, which is drawn centered at the origin,
54      // into the viewing frustum.
55      glTranslatef(0.0, 0.0, offset);
56
57      // Commands to turn the hemisphere.
58      glRotatef(Zangle, 0.0, 0.0, 1.0);
59      glRotatef(Yangle, 0.0, 1.0, 0.0);
60      glRotatef(Xangle, 1.0, 0.0, 0.0);
61      if(isSphere)
62      {
63           if(!filled)
64      {
65          glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
66      }
67      else
68      {
69          glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
70      }
71      // Hemisphere properties.
```

Then we draw sphere as it consist of two parts which is the upper part and lower part each part considered as hemisphere

main.cpp

```
73      glColor3f(0.0, 0.0, 0.0);
74
75      // Array of latitudinal triangle strips, each parallel to the equator, stacked one
76      // above the other from the equator to the north pole.
77      for (j = 0; j < q; j++)
78      {
79          // One latitudinal triangle strip.
80          glBegin(GL_TRIANGLE_STRIP);
81          for (i = 0; i <= p; i++)
82          {
83              glVertex3f(R * cos((float)(j + 1) / q * M_PI / 2.0) * cos(2.0 * (float)i / p * M_PI),
84                  R * sin((float)(j + 1) / q * M_PI / 2.0),
85                  -R * cos((float)(j + 1) / q * M_PI / 2.0) * sin(2.0 * (float)i / p * M_PI));
86              glVertex3f(R * cos((float)j / q * M_PI / 2.0) * cos(2.0 * (float)i / p * M_PI),
87                  R * sin((float)j / q * M_PI / 2.0),
88                  -R * cos((float)j / q * M_PI / 2.0) * sin(2.0 * (float)i / p * M_PI));
89          }
90          glEnd();
91      }
92      for (j = 0; j < q; j++)
93      {
94          // One latitudinal triangle strip.
95          glBegin(GL_TRIANGLE_STRIP);
96          for (i = 0; i <= p; i++)
97          {
98              glVertex3f(R * cos((float)(j + 1) / q * M_PI / 2.0) * cos(2.0 * (float)i / p * M_PI),
99                  -R * sin((float)(j + 1) / q * M_PI / 2.0),
100                 R * cos((float)(j + 1) / q * M_PI / 2.0) * sin(2.0 * (float)i / p * M_PI));
101             glVertex3f(R * cos((float)j / q * M_PI / 2.0) * cos(2.0 * (float)i / p * M_PI),
102                 -R * sin((float)j / q * M_PI / 2.0),
```

Then we use to draw helix if is not sphere in which we draw each vertex with different color.

```
97          {
98              glVertex3f(R * cos((float)(j + 1) / q * M_PI / 2.0) * cos(2.0 * (float)i / p * M_PI),
99                  -R * sin((float)(j + 1) / q * M_PI / 2.0),
100                 R * cos((float)(j + 1) / q * M_PI / 2.0) * sin(2.0 * (float)i / p * M_PI));
101             glVertex3f(R * cos((float)j / q * M_PI / 2.0) * cos(2.0 * (float)i / p * M_PI),
102                 -R * sin((float)j / q * M_PI / 2.0),
103                 R * cos((float)j / q * M_PI / 2.0) * sin(2.0 * (float)i / p * M_PI));
104         }
105         glEnd();
106     }
107     glFlush();
108
109     }
110     else
111     {
112         for (int i = 0; i < num_turns; i++) {
113         // Draw a segment of the helix
114         glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
115         glBegin(GL_LINE_STRIP);
116         for (int j = 0; j <= num_vertices; j++) {
117             // Calculate the position of the current vertex
118             float t = (float)j /num_vertices;
119             float theta = t * 2.0f * M_PI * num_turns;
120             float x = radius * cos(theta);
121             float y = radius * sin(theta);
122             float z = t * pitch + i;
123
124             // Set the color of the current vertex to a random color
125             glColor3f((float)rand() / RAND_MAX, (float)rand() / RAND_MAX, (float)rand() / RAND_MAX);
126
127             // Draw the current vertex
```

```
..
124             // Set the color of the current vertex to a random color
125             glColor3f((float)rand() / RAND_MAX, (float)rand() / RAND_MAX, (float)rand() / RAND_MAX);
126
127             // Draw the current vertex
128             glVertex3f(x, y, z);
129         }
130         glEnd();
131     }
132          glutSwapBuffers();
133     }
134
135 }
136
137 // OpenGL window reshape routine.
138 void resize(int w, int h)
139 {
140     glViewport(0, 0, w, h);
141     glMatrixMode(GL_PROJECTION);
142     glLoadIdentity();
143     glFrustum(-5.0, 5.0, -5.0, 5.0, 5.0, 100.0);
144     glMatrixMode(GL_MODELVIEW);
145 }
146
147 void spinDisplay() {
148     Yangle += spinSpeed * (glutGet(GLUT_ELAPSED_TIME) - prev_time) / 1000;
149     prev_time = glutGet(GLUT_ELAPSED_TIME); // to indicate refresh rate
150     if (Yangle > 360.0) Yangle -= 360.0;
151     glutPostRedisplay();
152 }
```

```
}
void spinDisplayReverse() {

    Yangle -= spinSpeed * (glutGet(GLUT_ELAPSED_TIME) - prev_time) / 1000;
    prev_time = glutGet(GLUT_ELAPSED_TIME); // to indicate refresh rate
    std::cout << Yangle << std::endl;
    if (Yangle > 360.0) Yangle -= 360.0;
    glutPostRedisplay();
}
void mouse(int button, int state, int x, int y)
{
    switch (button)
    {
    case GLUT_LEFT_BUTTON:
        if (state == GLUT_DOWN)
            glutIdleFunc(spinDisplay);
        break;
    case GLUT_RIGHT_BUTTON:
        if (state == GLUT_DOWN)
            glutIdleFunc(spinDisplayReverse);

        break;

    default:
        break;
    }
}
```

```
void keyInput(unsigned char key, int x, int y)
{

       switch (key)
    {
    case 27:
        exit(0);
        break;
    case 'x':
        Xangle += 5.0;
        if (Xangle > 360.0) Xangle -= 360.0;
        glutPostRedisplay();
        break;
    case 'X':
        Xangle -= 5.0;
        if (Xangle < 0.0) Xangle += 360.0;
        glutPostRedisplay();
        break;
    case 'y':
        Yangle += 5.0;
        if (Yangle > 360.0) Yangle -= 360.0;
        glutPostRedisplay();
        break;
    case 'Y':
        Yangle -= 5.0;
        if (Yangle < 0.0) Yangle += 360.0;
        glutPostRedisplay();
        break;
```

If it is sphere we check about the entered key and make it

```
208         break;
209     case 'z':
210         Zangle += 5.0;
211         if (Zangle > 360.0) Zangle -= 360.0;
212         glutPostRedisplay();
213         break;
214     case 'Z':
215         Zangle -= 5.0;
216         if (Zangle < 0.0) Zangle += 360.0;
217         glutPostRedisplay();
218         break;
219     case 'O':
220         offset += 1;
221         glutPostRedisplay();
222         break;
223     case 'o':
224         offset -= 1;
225         glutPostRedisplay();
226         break;
227     case ' ':
228         glutIdleFunc(NULL);
229         break;
230     if(isSphere)
231     {
232         case 'P':
233           p += 1;
234           glutPostRedisplay();
235           break;
236        case 'p':
237           if (p > 3) p -= 1;
```

If it is sphere we check about the entered key and make it

```
238           glutPostRedisplay();
239        break;
240     case 'Q':
241        q += 1;
242        glutPostRedisplay();
243        break;
244     case 'q':
245       if (q > 3) q -= 1;
246       glutPostRedisplay();
247       break;
248    case 'W':
249       glClearColor(1.0, 1.0, 1.0, 0.0);
250       filled = false;
251       glutDisplayFunc(drawScene);
252       glutPostRedisplay();
253       break;
254    case 'w':
255       glClearColor(1.0, 1.0, 1.0, 0.0);
256       filled = true;
257       glutDisplayFunc(drawScene);
258       glutPostRedisplay();
259       break;
260    }
261    else
262    {
263        case 'R':
264            radius += 0.1f;
265            glutPostRedisplay();
266            break;
267        case 'r':
```

```
            radius -= 0.1f;
            glutPostRedisplay();
            break;
        case 'H':
            pitch += 0.1f;
            glutPostRedisplay();
            break;
        case 'h':
            pitch -= 0.1f;
            glutPostRedisplay();
            break;
        case 'N':
            num_vertices += 10;
            glutPostRedisplay();
            break;
        case 'n':
            num_vertices -= 10;
            glutPostRedisplay();
            break;
    }
    default:
        break;
    }


}
```

```
// Routine to output interaction instructions to the C++ window.
void printInteraction(void)
{
    if(isSphere)
    {
        std::cout << "Interaction:" << std::endl;
        std::cout << "Press P/p to increase/decrease the number of longitudinal slices." << std::endl
        << "Press Q/q to increase/decrease the number of latitudinal slices." << std::endl
        << "Press W/w to draw  sphere in wireframe/filled sphere." << std::endl
        << "Press x, X, y, Y, z, Z to turn the sphere." << std::endl;
    }
    else
    {
          std::cout << "Interaction:" << std::endl;
        std::cout << "Press R/r to increase/decrease radius of the helix." << std::endl
        << "Press H/h to increase/decrease pitch of helix." << std::endl
        << "Press N/n to  increase/decrease number of vertices used to draw the helix" << std::endl
        << "Press x, X, y, Y, z, Z to turn the helix." << std::endl;
    }
}

// Main routine
```

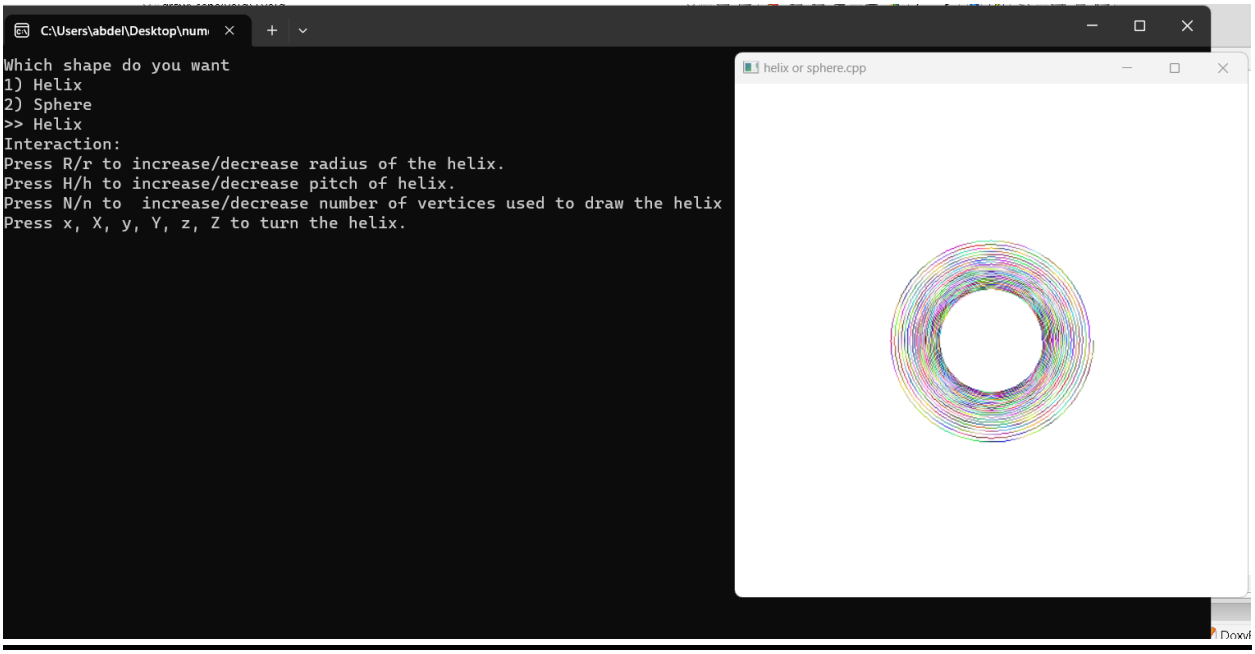
```cpp
// Main routine.
int main(int argc, char** argv)
{

    glutInit(&argc, argv);
     std::cout << "Which shape do you want\n1) Helix\n2) Sphere\n>> ";
     std::string choice;
     std::getline(std::cin, choice);
     if (choice == "Helix")
    {
        isSphere= false;
    }
    else if (choice == "Sphere")
    {
        isSphere= true;
    }
    printInteraction();
    glutInitContextVersion(4, 3);
    glutInitContextProfile(GLUT_COMPATIBILITY_PROFILE);

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("helix or sphere.cpp");
    glutDisplayFunc(drawScene);
    glutReshapeFunc(resize);
    glutKeyboardFunc(keyInput);
    glutMouseFunc(mouse);
    glewExperimental = GL_TRUE;
    glewInit();
```

```cpp
     std::getline(std::cin, choice);
     if (choice == "Helix")
    {
        isSphere= false;
    }
    else if (choice == "Sphere")
    {
        isSphere= true;
    }
    printInteraction();
    glutInitContextVersion(4, 3);
    glutInitContextProfile(GLUT_COMPATIBILITY_PROFILE);

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("helix or sphere.cpp");
    glutDisplayFunc(drawScene);
    glutReshapeFunc(resize);
    glutKeyboardFunc(keyInput);
    glutMouseFunc(mouse);
    glewExperimental = GL_TRUE;
    glewInit();

    setup();

    glutMainLoop();
}
```

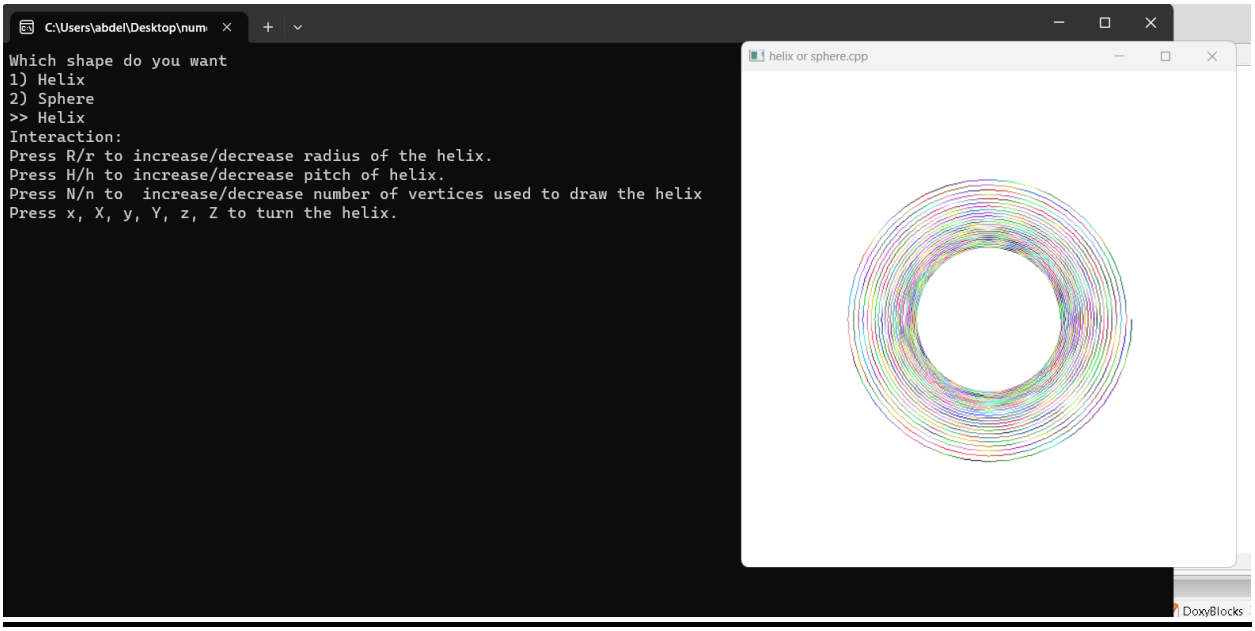
Test cases :

Helix:

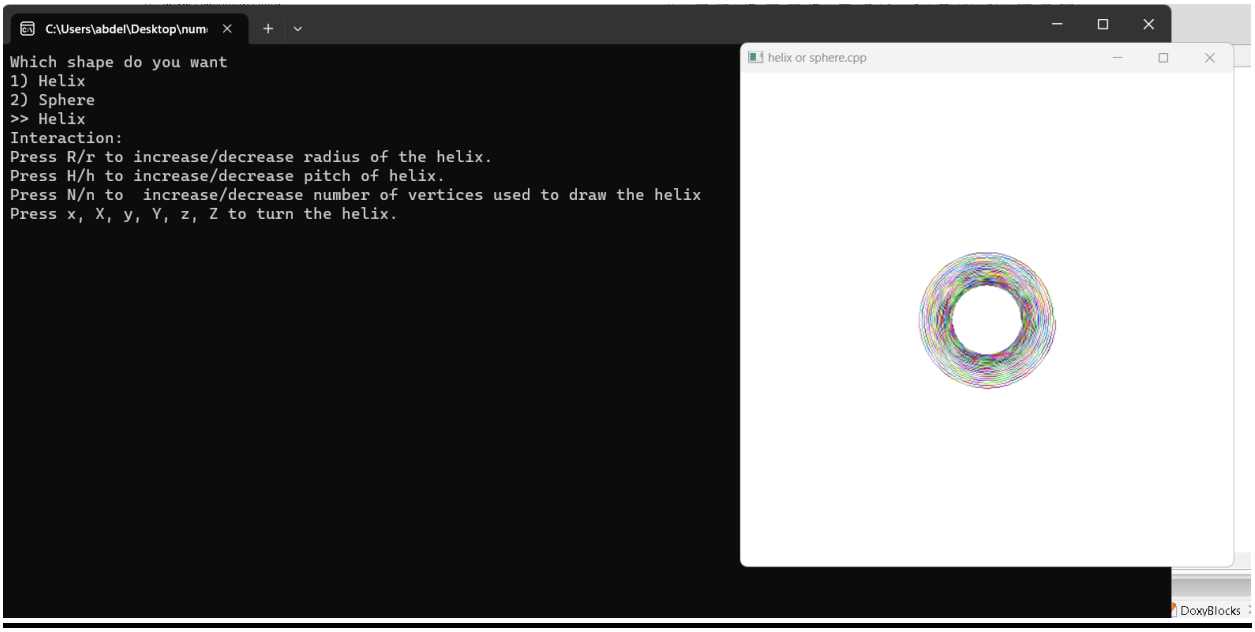
Normal helix



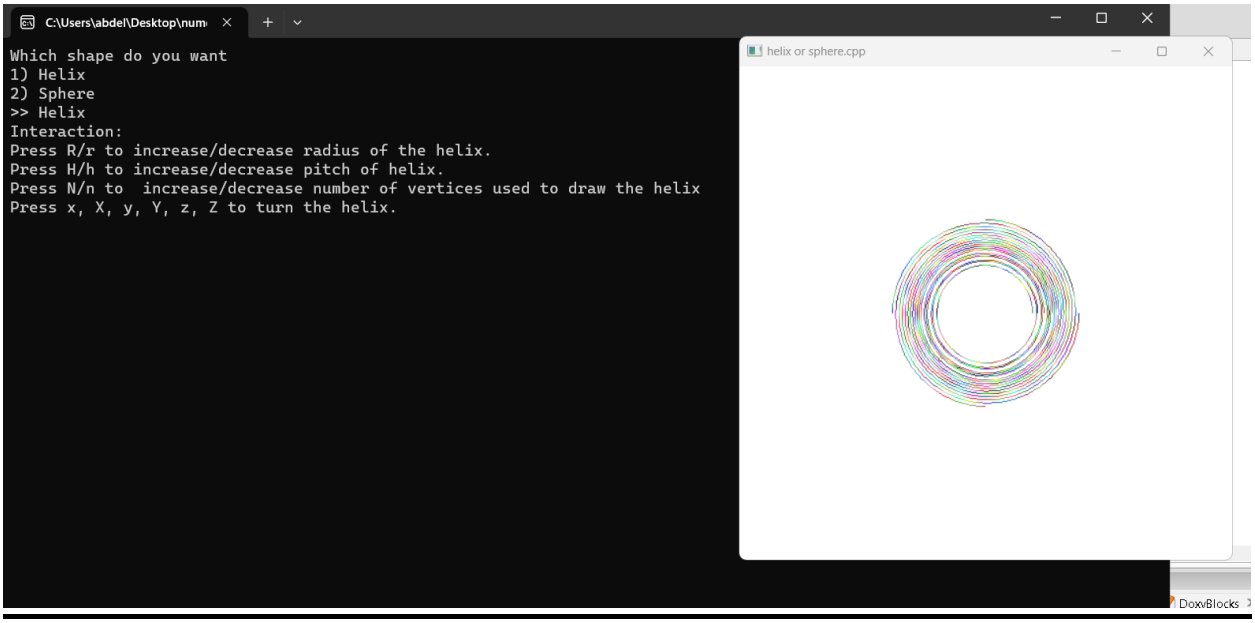
When Press R to increase radius



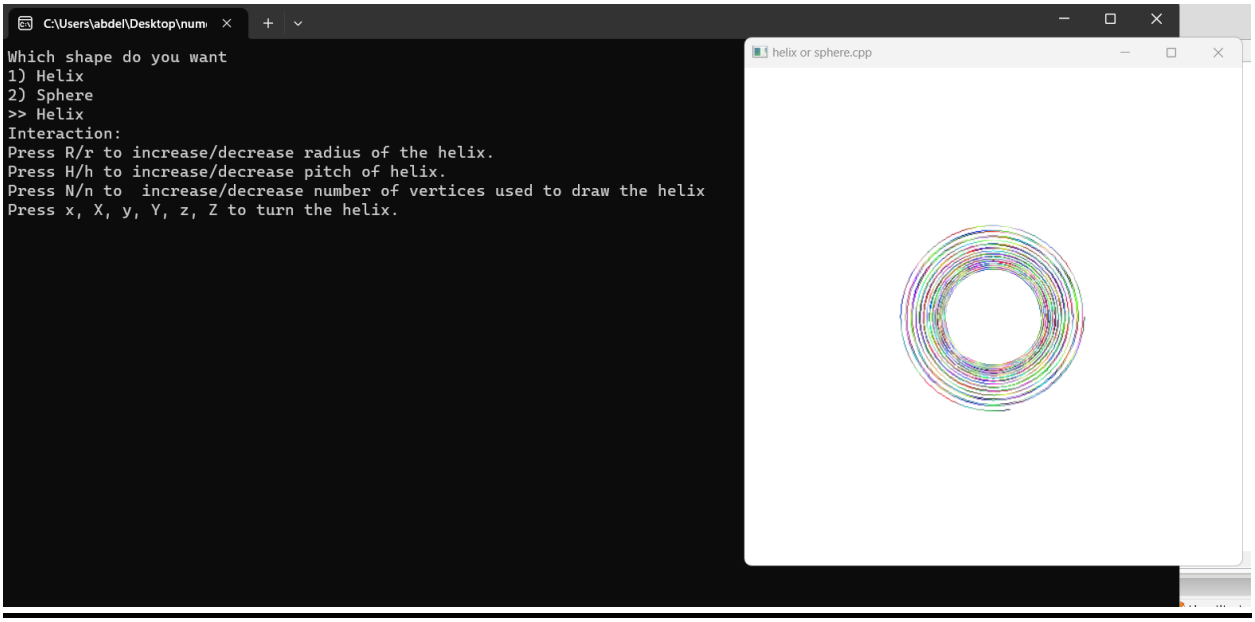
When Press r to decrease radius



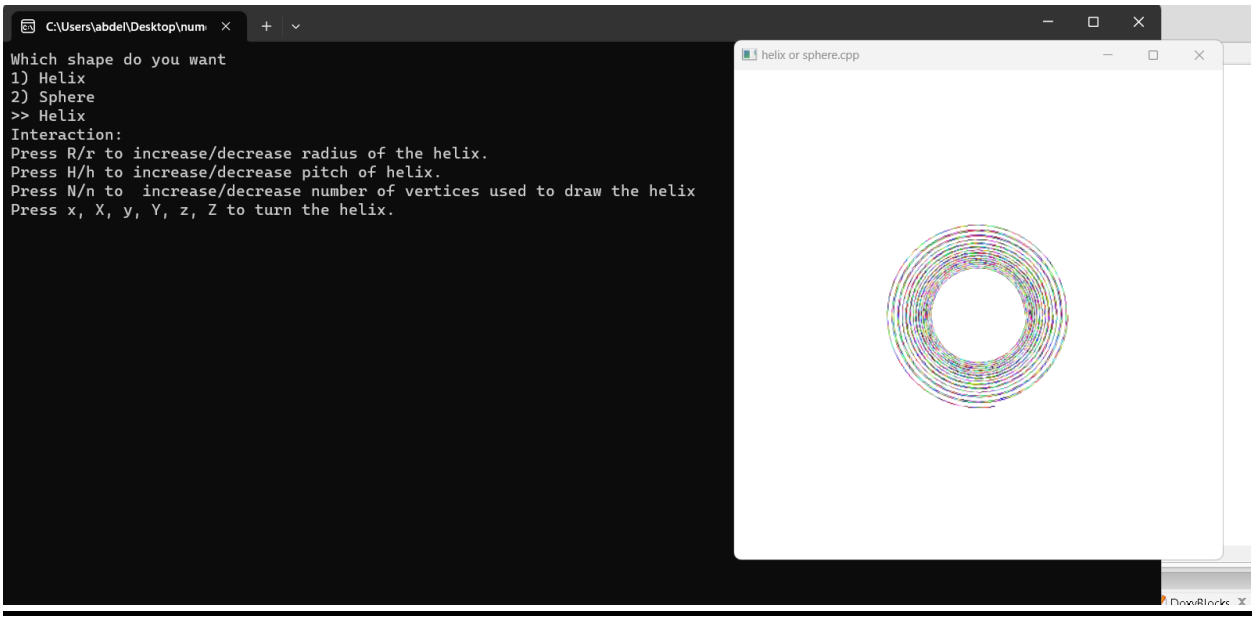
When press H increase pitch of helix.



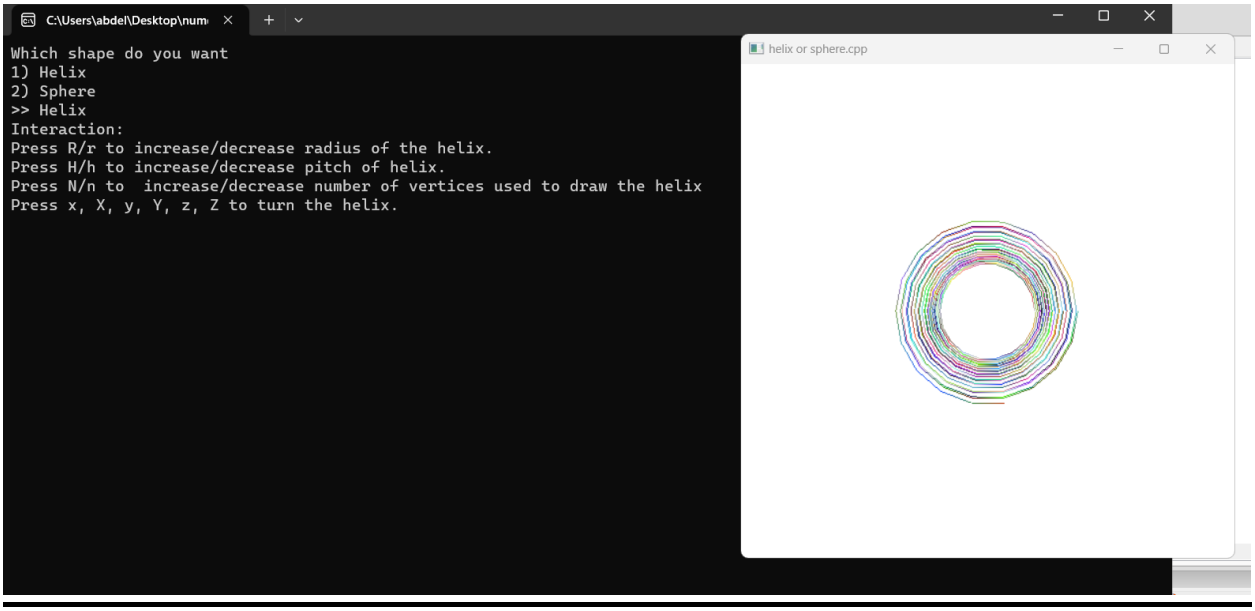
When press h decrease pitch of helix.



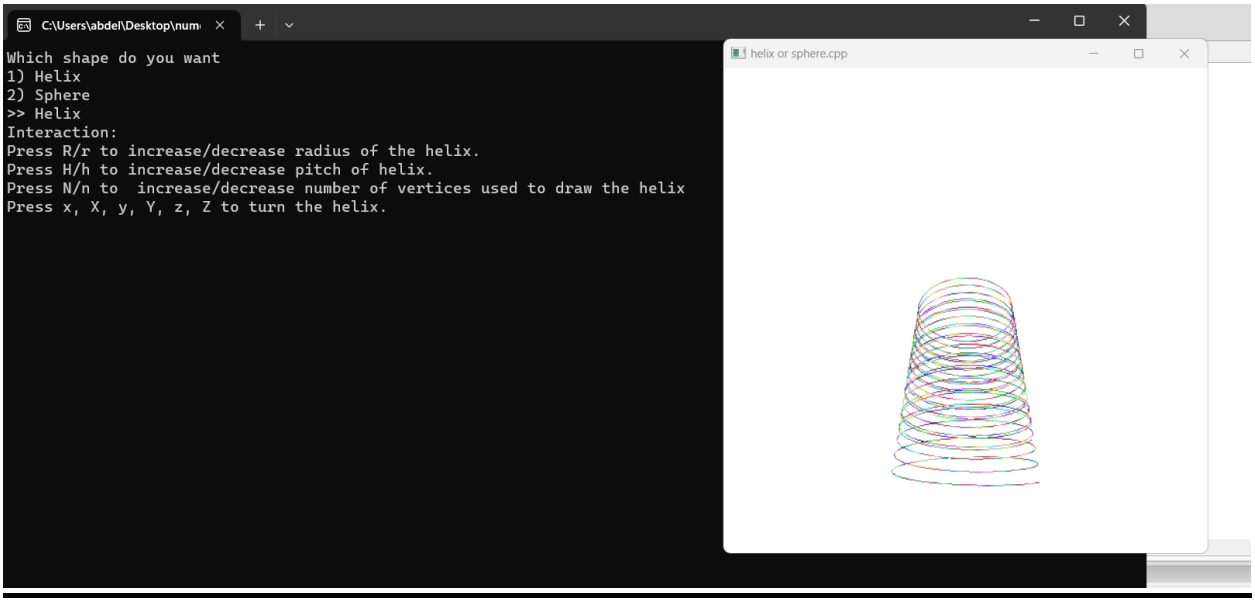
When press N to increase number of vertices of helix.



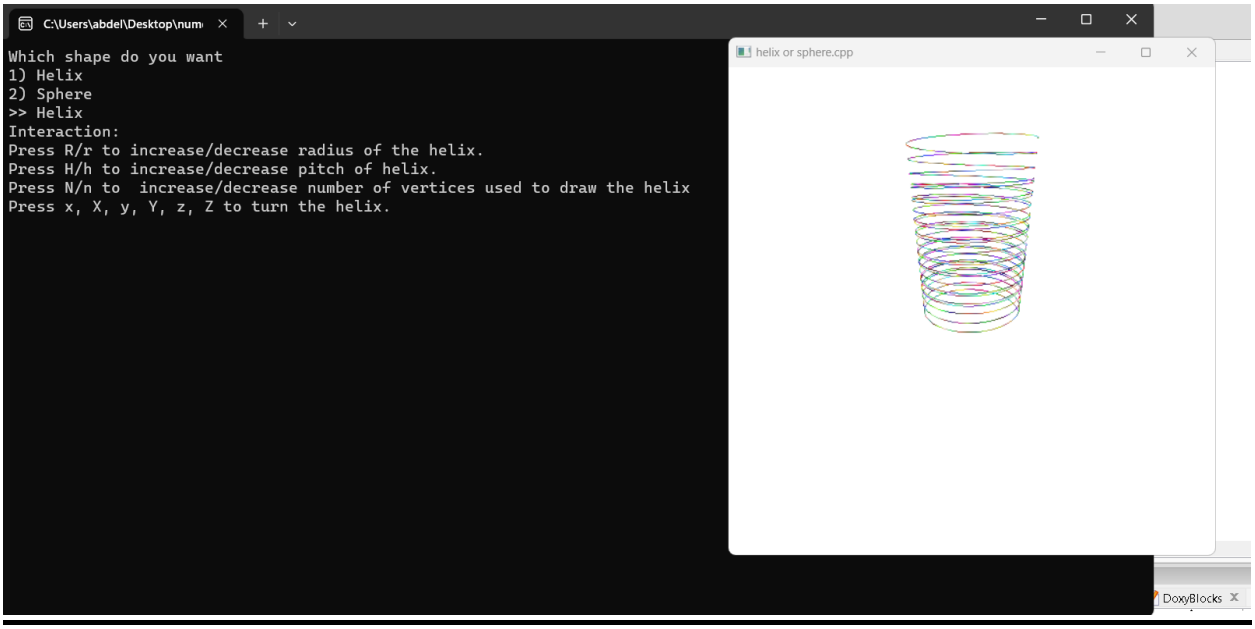
When press n to decrease number of vertices of helix.



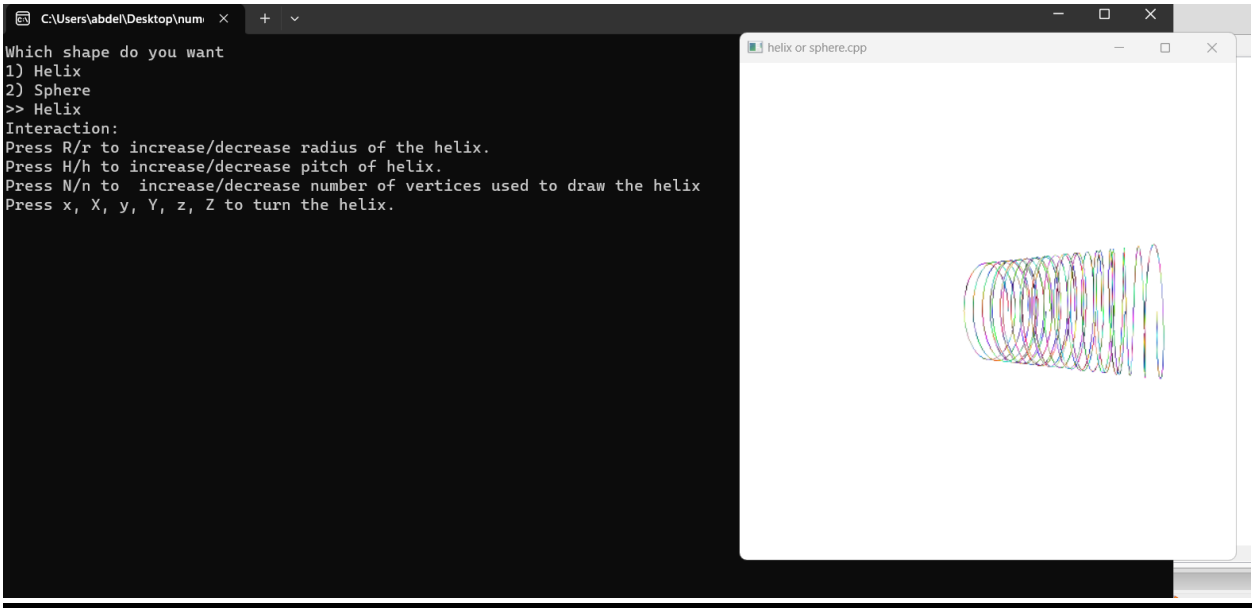
when press x to turn the helix



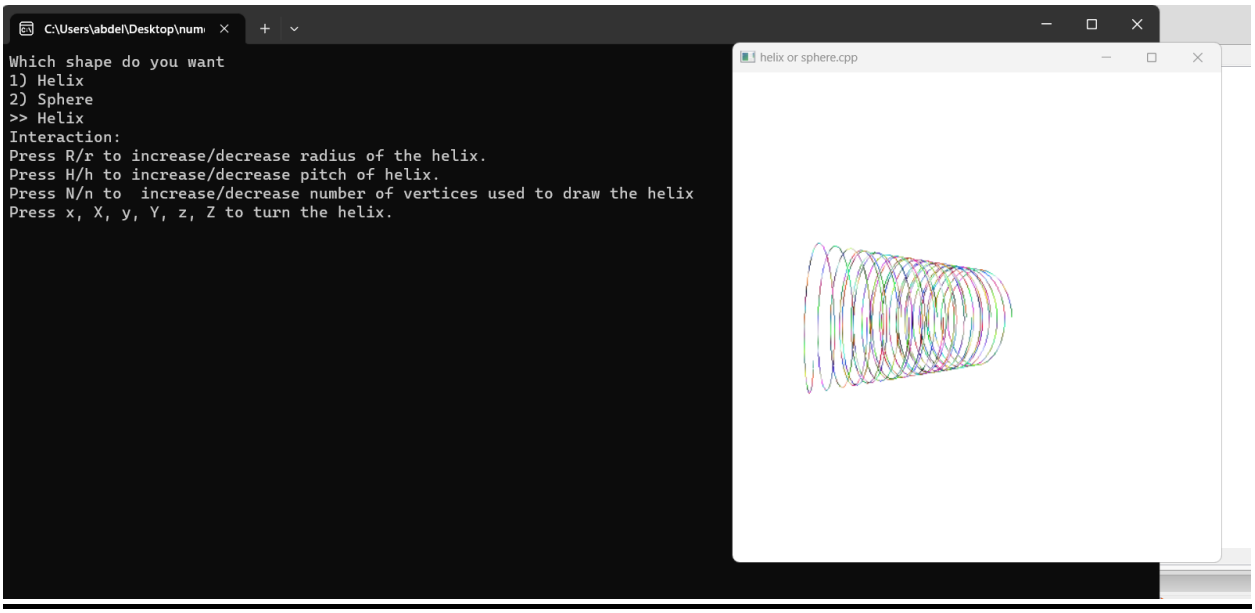
when press X to turn the helix



when press y to turn the helix

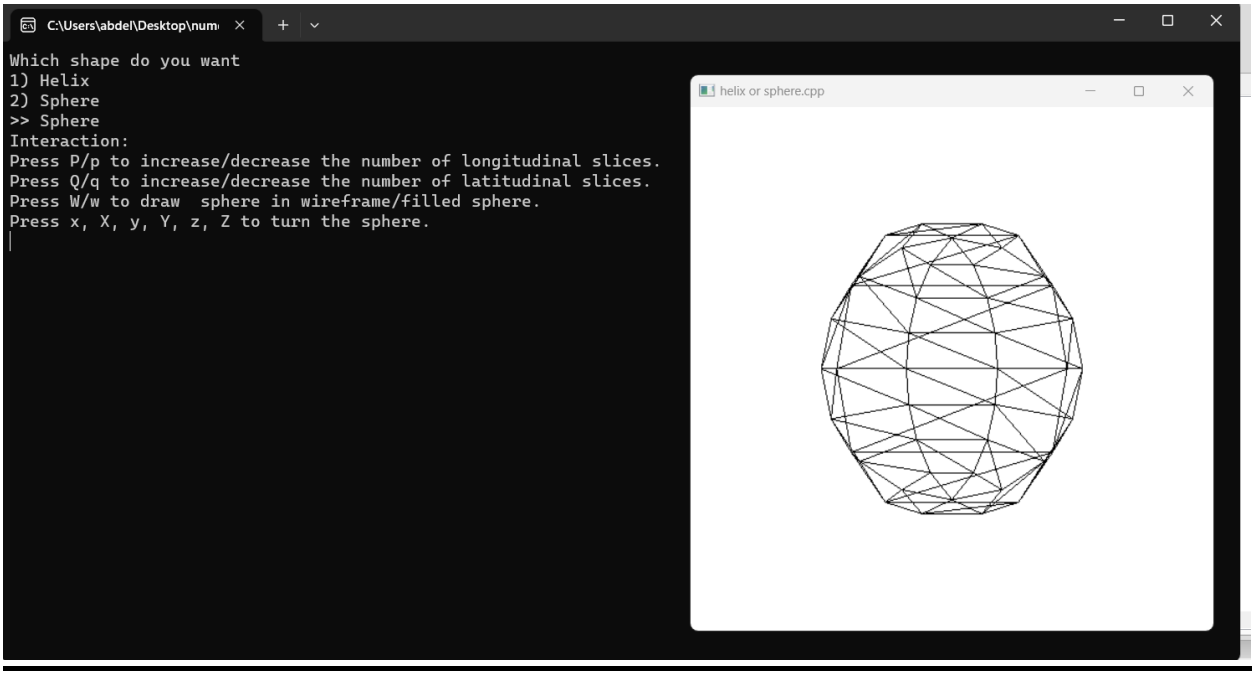


when press Y to turn the helix

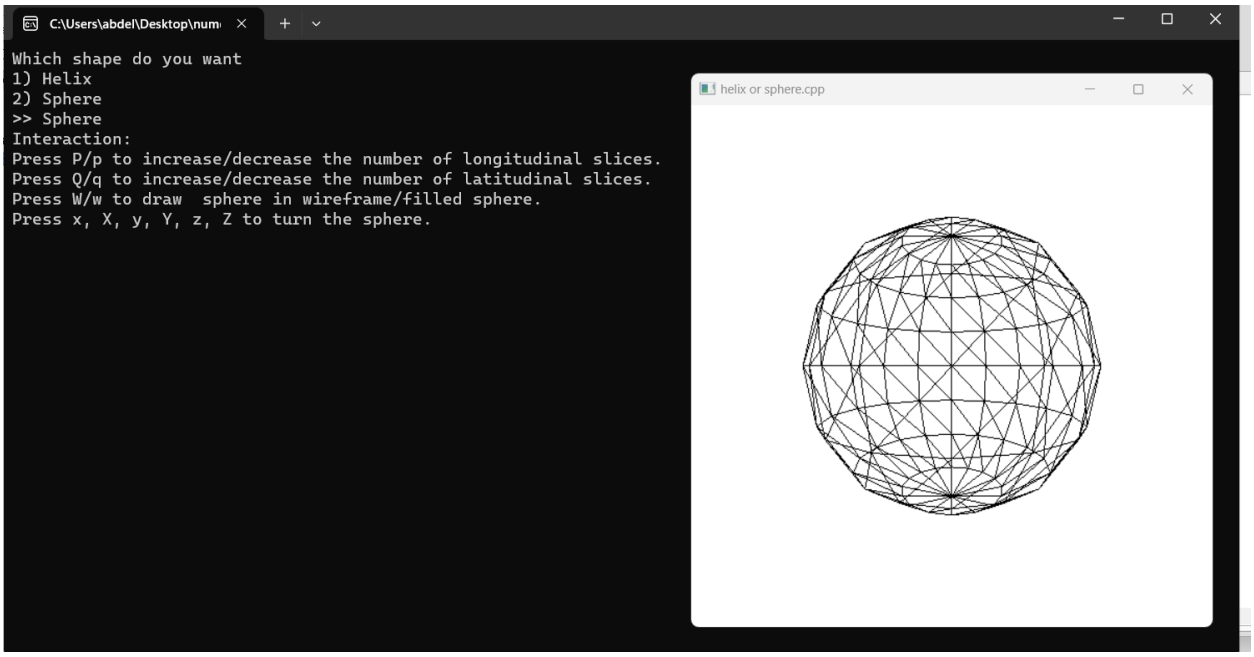


Sphere :

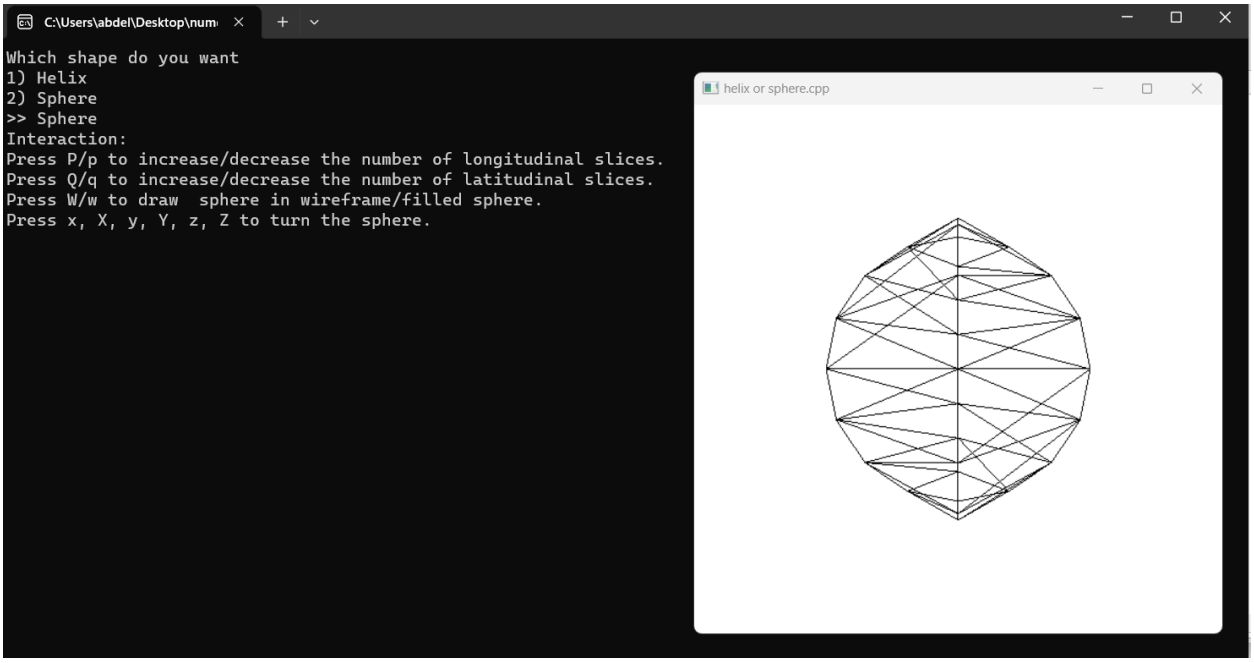
Normal sphere:



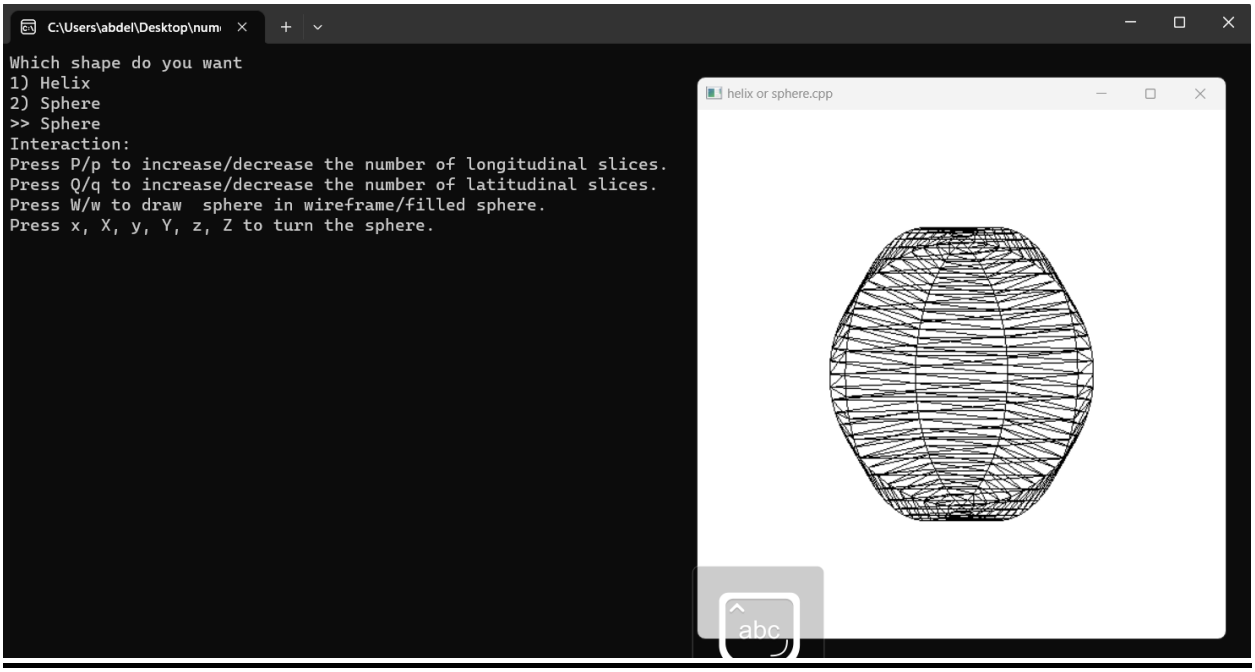
When press P to increase longitudinal slices.



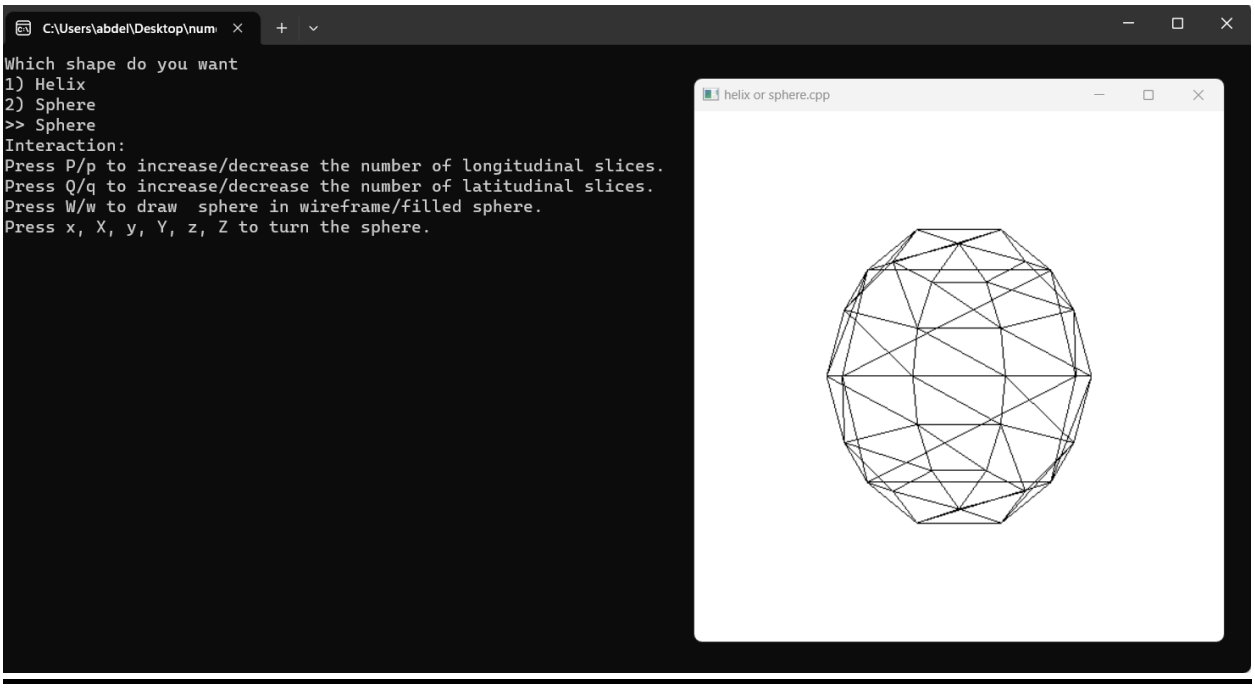
When press p to decrease longitudinal slices.



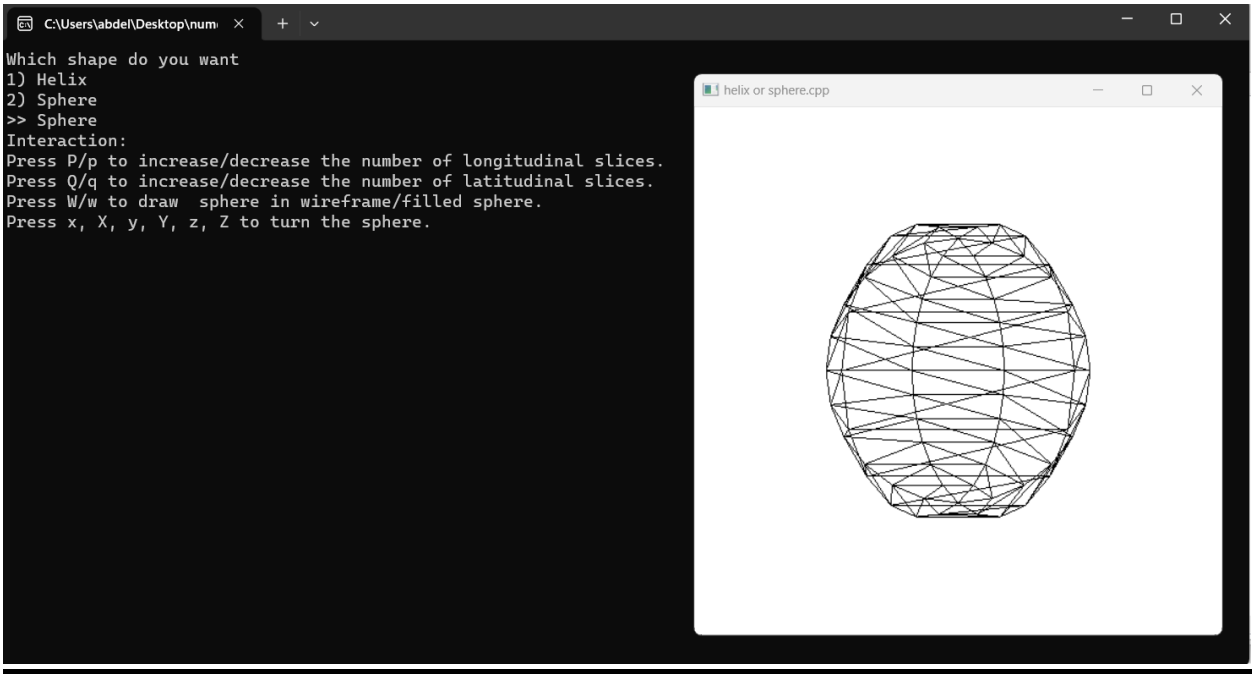
When press Q to increase latitudinal slices.



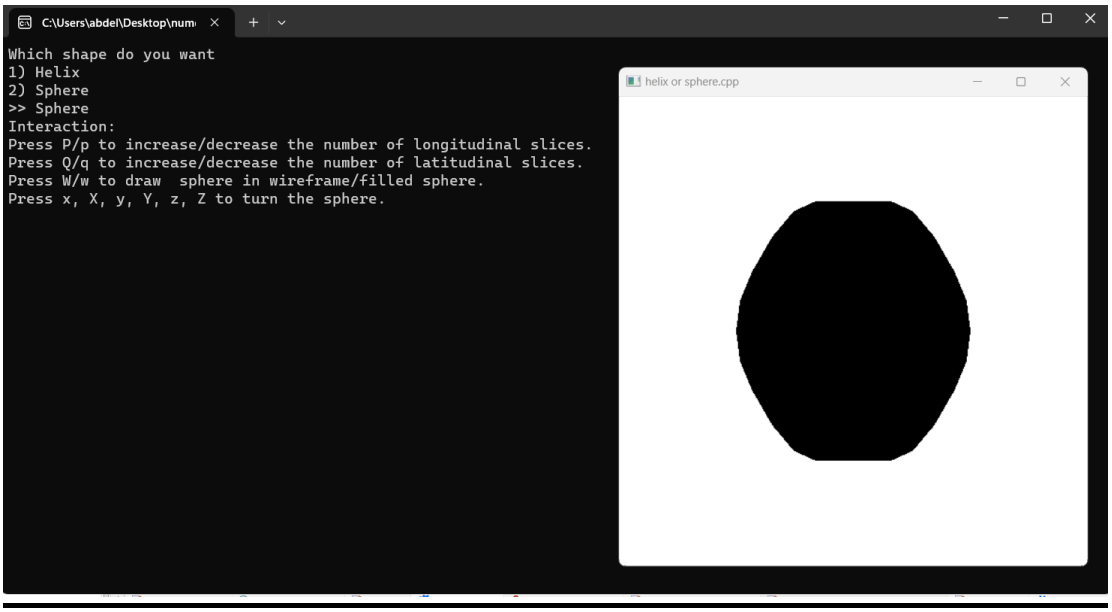
When press q to decrease latitudinal slices.



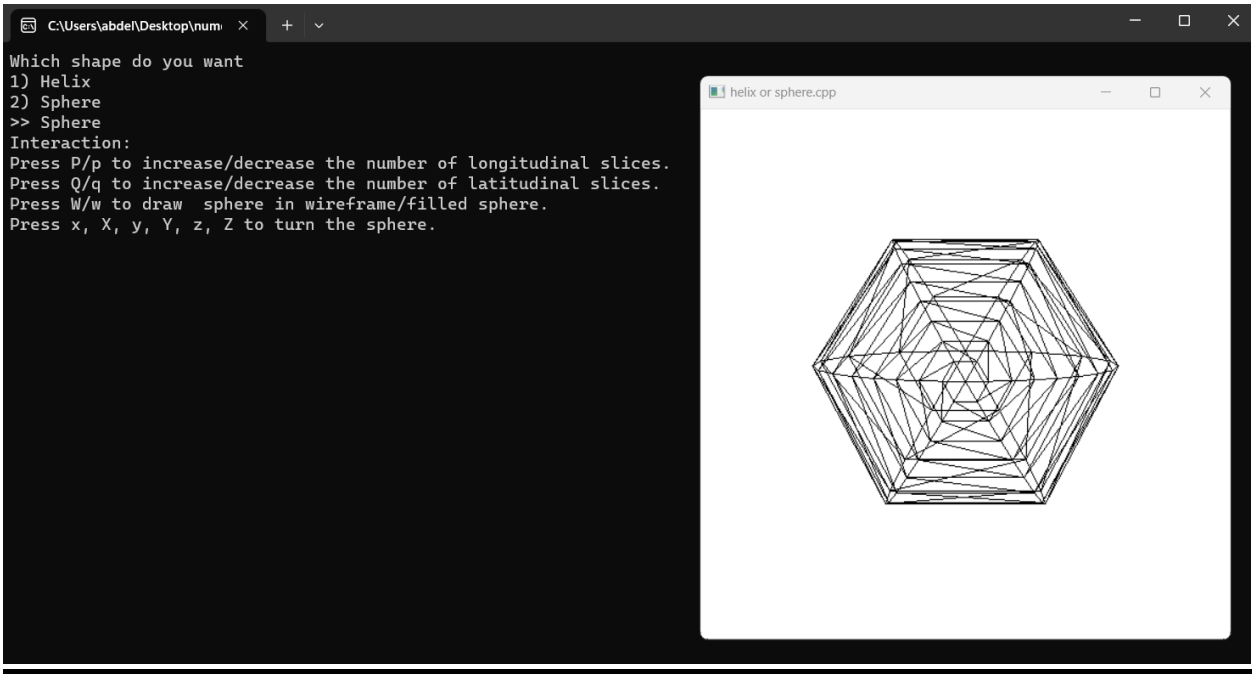
When press W to draw wireframe sphere.



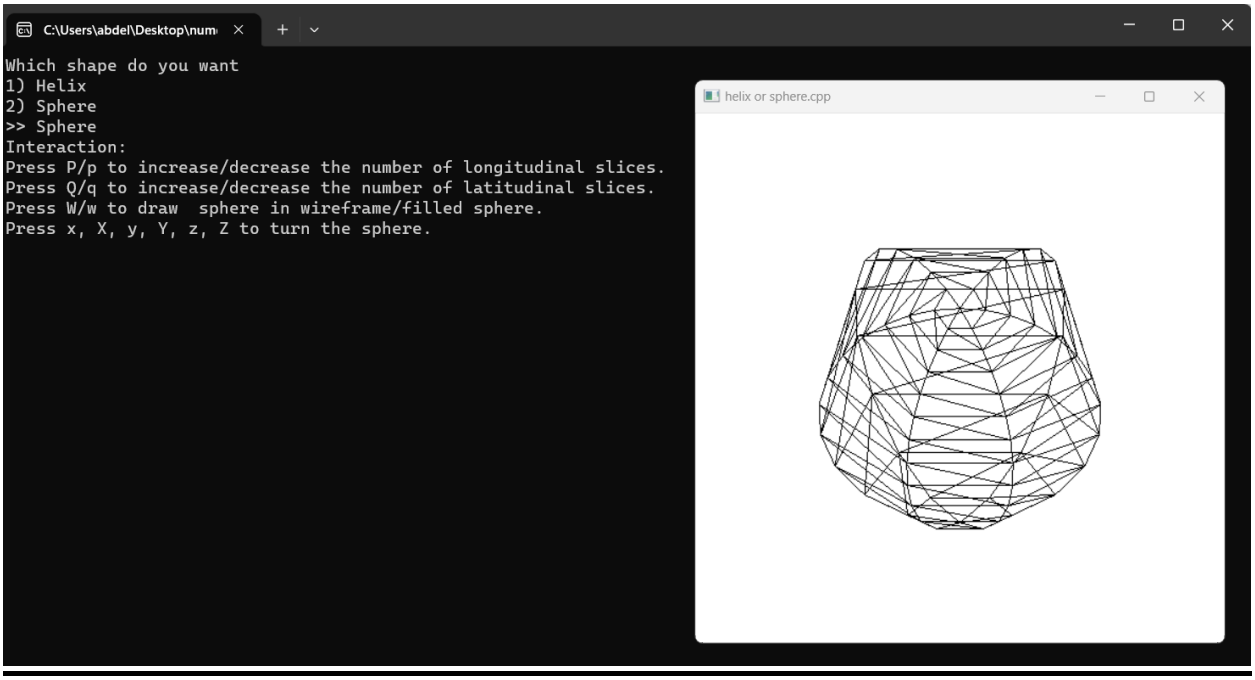
When press w to draw filled sphere.



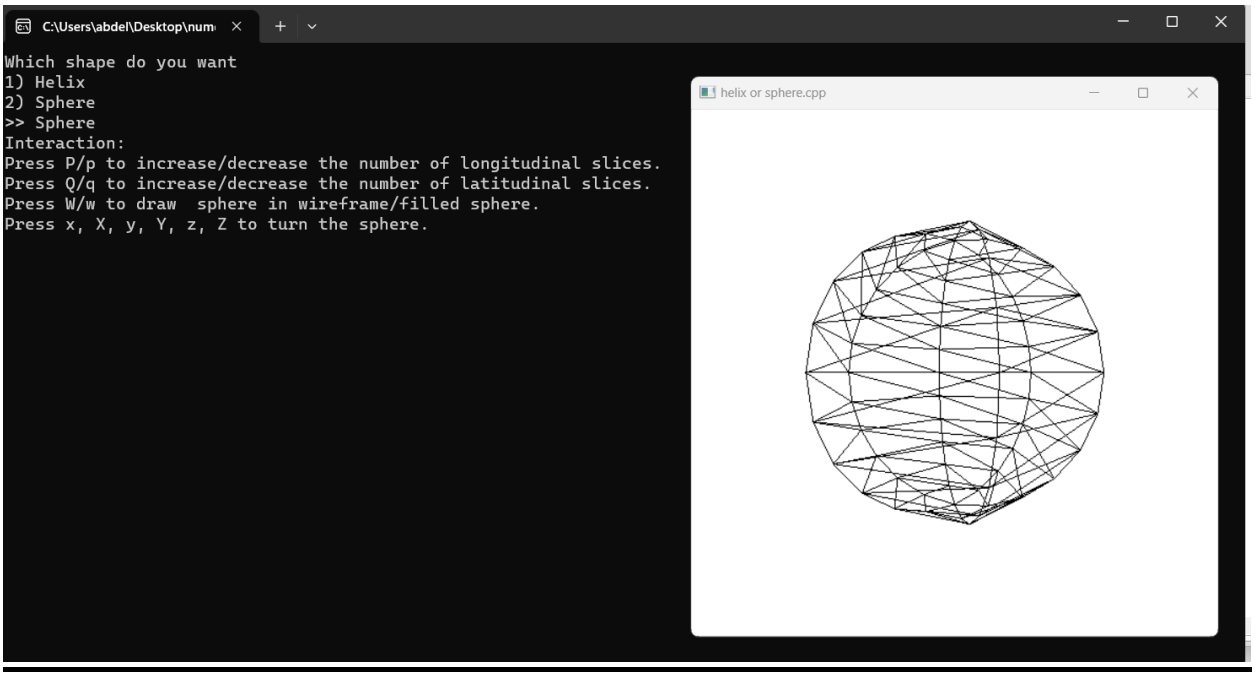
When press x to turn sphere.



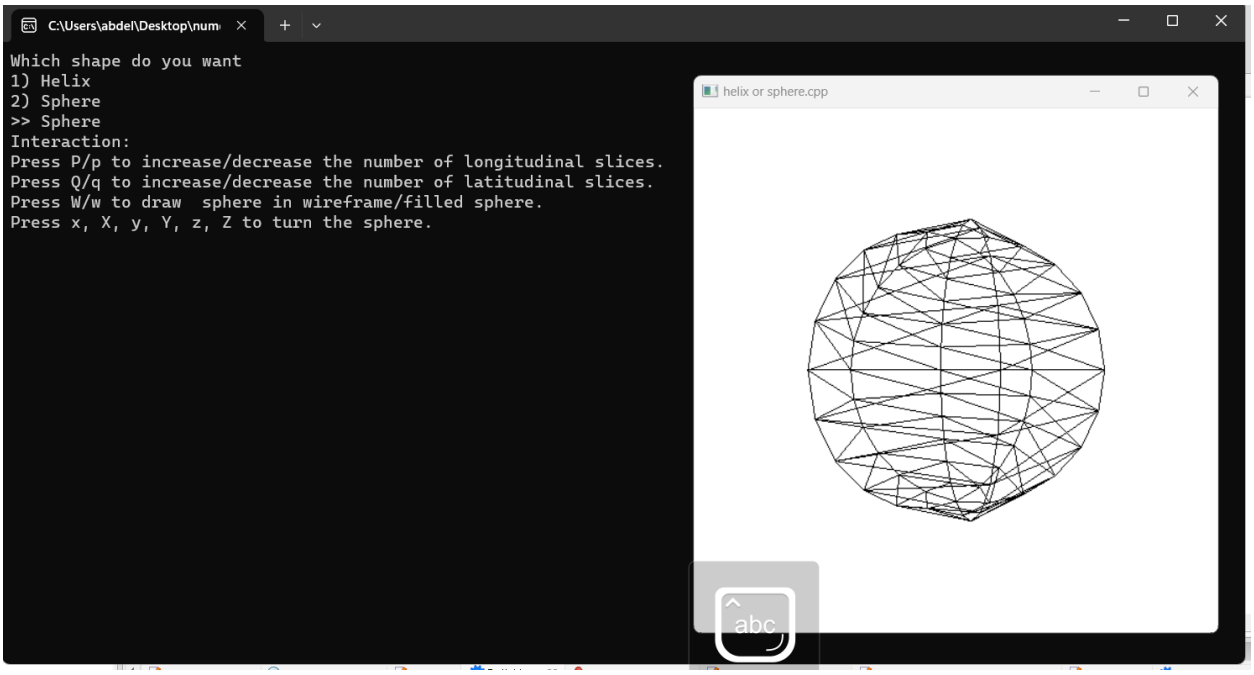
When press X to turn sphere.



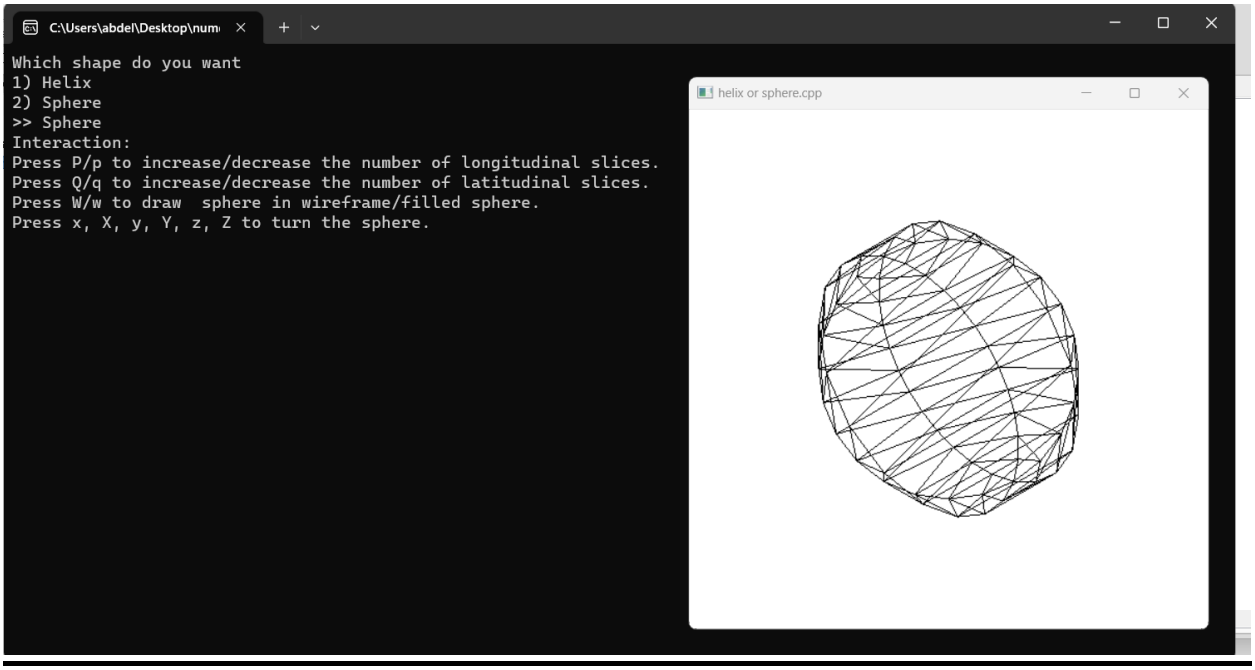
When press y to turn sphere.



When press Y to turn sphere.



When press z to turn sphere.



When press Z to turn sphere.