ПРАКТИЧЕСКАЯ РАБОТА №1

по дисциплине

«Компьютерная графика»

**Вариант 1**

|  |  |
| --- | --- |
| Выполнил  студент гр. 5130904/10101 | Абраамян А. М. |
| Преподаватель | Леонтьева Т.В. |

Оглавление

[Цель работы 3](#_Toc183950112)

[Порядок выполнения 3](#_Toc183950113)

[Результат выполнения 4](#_Toc183950114)

[Код программы 6](#_Toc183950115)

[Вывод 9](#_Toc183950116)

# Цель работы

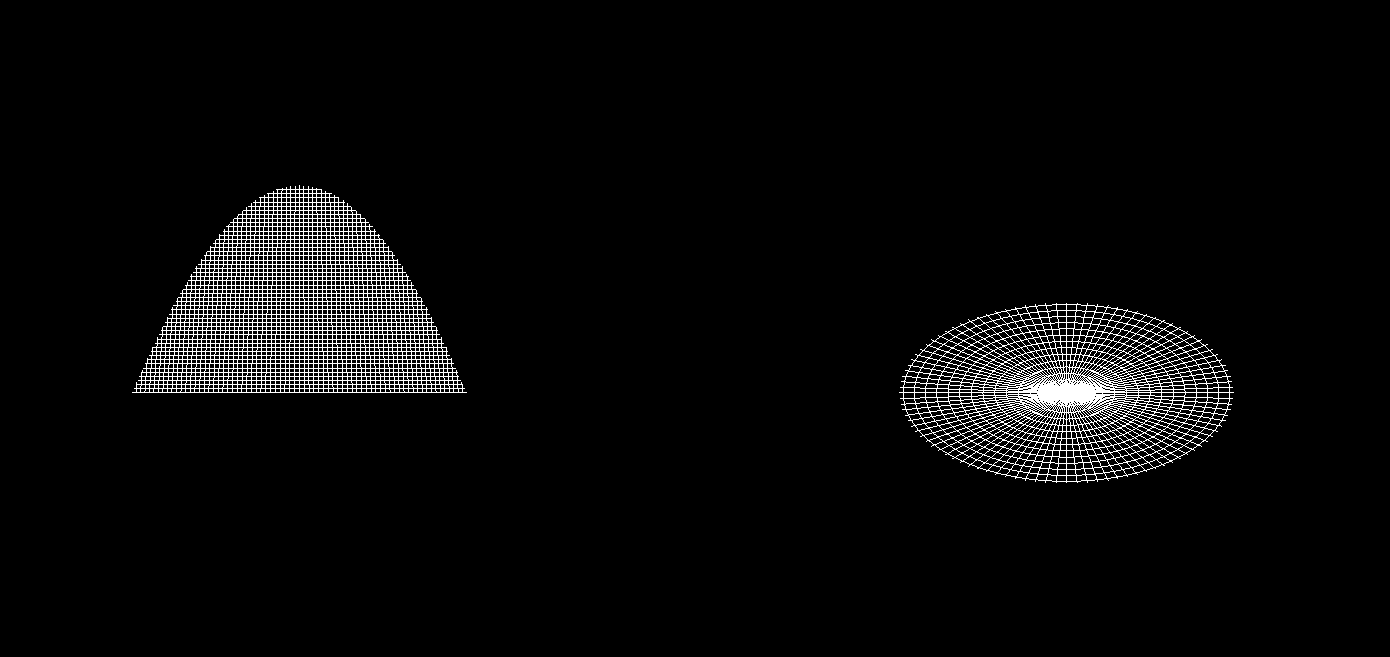
Целью работы является ознакомление с основами векторной графики и получение навыков работы с базовыми функциями графического API и трехмерными графическими примитивами. Требуется при помощи стандартных функций библиотеки (OpenGL/Vulkan или DirectX) изобразить указанные объекты и произвести необходимые преобразования.

# Порядок выполнения

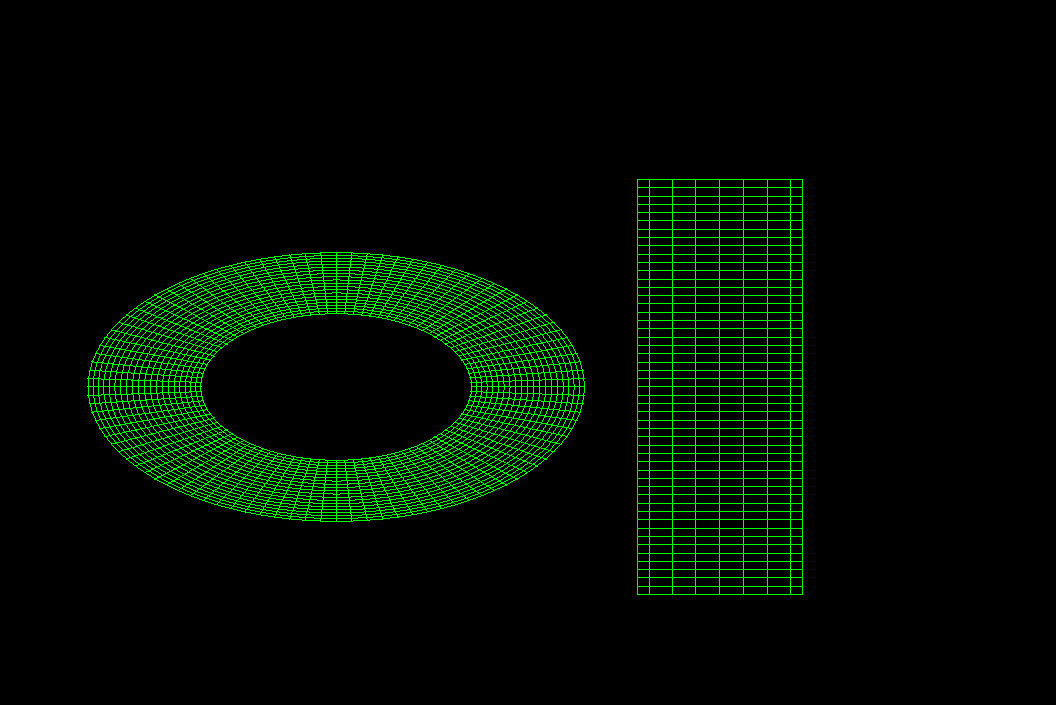
1. Изобразить каркасный конус и каркасную сферу, расположенные на некотором расстоянии друг от друга.
2. Совместить центр основания конуса и центр сферы.
3. Изобразить тор и цилиндр. Размеры и местоположение примитивов задать самостоятельно.
4. Выполнить последовательно сначала поворот цилиндра вокруг оси Х, а затем растяжение тора в 2 раза.

# Результат выполнения

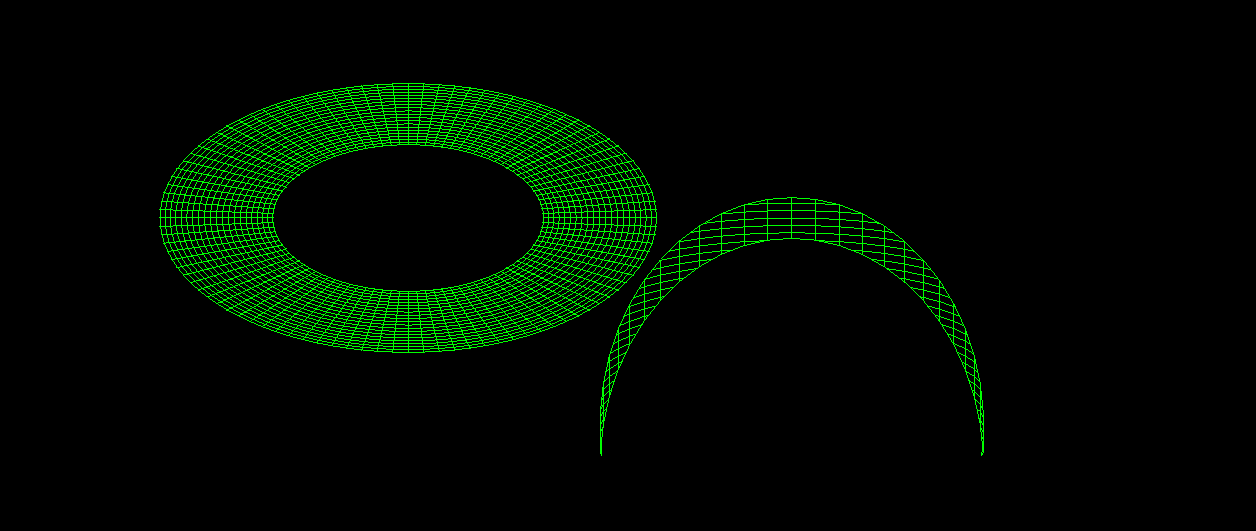
1. Изобразить каркасный конус и каркасную сферу, расположенные на некотором расстоянии друг от друга.

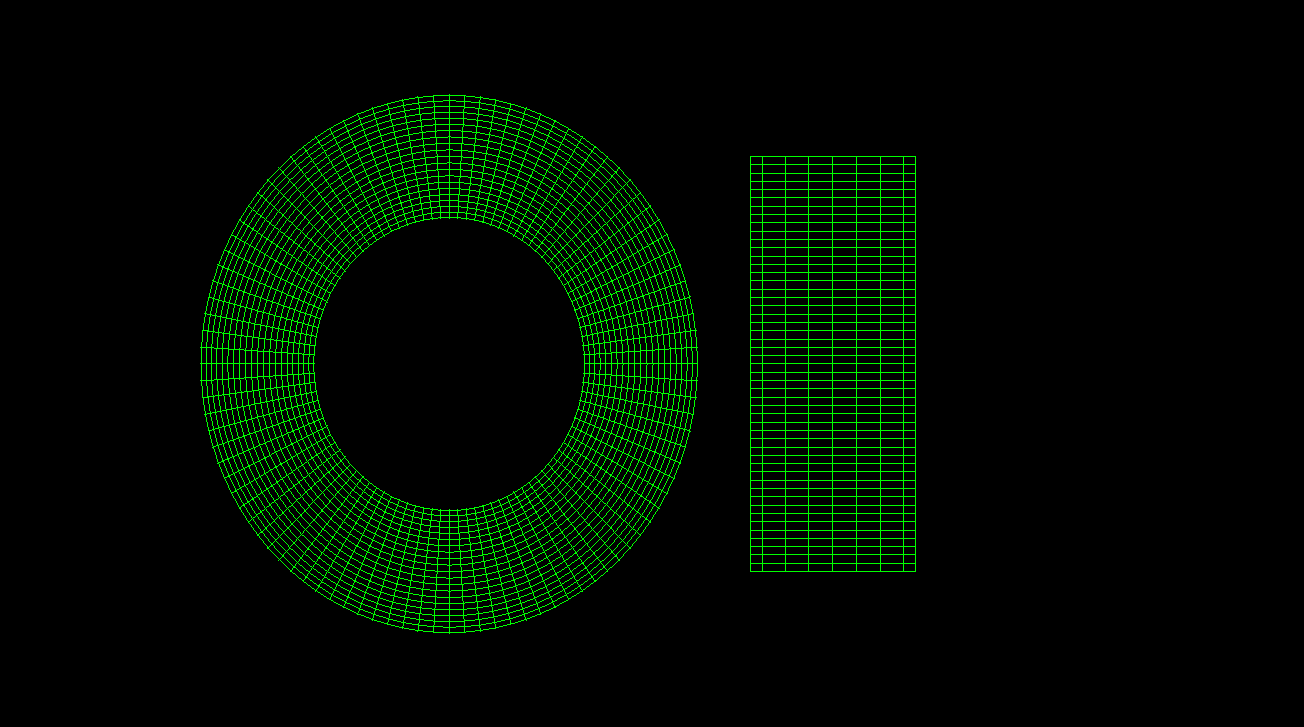


1. Совместить центр основания конуса и центр сферы.
2. Изобразить тор и цилиндр. Размеры и местоположение примитивов задать самостоятельно.



4. Выполнить последовательно сначала поворот цилиндра вокруг оси Х, а затем растяжение тора в 2 раза





# Код программы

|  |
| --- |
| #include <states/conus\_sphere\_stay\_state/conus\_sphere\_stay\_state.h>  #include <iostream>  #include <GL/glut.h> // Include GLUT for cool drawing utils  #include <state\_machine/state\_machine.h>  #include <states/conus\_sphere\_moving\_state/conus\_sphere\_moving\_state.h>  #include <states/conus\_sphere\_stay\_state/constants.h>  using namespace states::conus\_sphere\_stay\_state::constants;  namespace states {  GLState\* ConusSphereStayState::display()  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glPushMatrix();  glLoadIdentity();  glTranslated(cone\_start\_pos.x, cone\_start\_pos.y, cone\_start\_pos.z);  glRotated(90, -1, 0, 0);  glutWireCone(cone\_base, cone\_height, cone\_slices, cone\_stacks);  glPopMatrix();  glLoadIdentity();  glPushMatrix();  glLoadIdentity();  glTranslated(sphere\_start\_pos.x, sphere\_start\_pos.y, sphere\_start\_pos.z);  glutWireSphere(sphere\_radius, sphere\_slices, sphere\_stacks);  glutSwapBuffers();  glPopMatrix();  return this;  }  void ConusSphereStayState::timeout()  {  std::cout << "ConusSphereStayState timeout " << frames\_count << std::endl;  frames\_count++;  glutPostRedisplay();  if (frames\_count == MAX\_FRAMES\_COUNT) {  std::cout << "Switching the state\n";  state\_machine::StateMachine::instance()->set\_state(new states::ConusSphereMovingState());  }  }  } // namespace states  #pragma once  #include <constants.h>  #include <states/states.h>  namespace states {  class ConusSphereStayState : public GLState  {  public:  GLState\* display() override final;  void timeout() override final;  private:  size\_t frames\_count = 0;  };  } // namespace states  #pragma once  #include <GL/glut.h> // Include GLUT for cool drawing utils  #include <glm/vec3.hpp>  #include <constants.h>  namespace states::conus\_sphere\_stay\_state::constants {  static constexpr size\_t STAY\_SECONDS = 1;  static constexpr size\_t MAX\_FRAMES\_COUNT = ::constants::FRAMES\_PER\_SECONDS \* STAY\_SECONDS;  static constexpr GLdouble cone\_base = 20.0;  static constexpr GLdouble cone\_height = 200.0;  static constexpr GLint cone\_slices = 500;  static constexpr GLint cone\_stacks = 500;  static constexpr GLint sphere\_radius = 20;  static constexpr GLint sphere\_slices = 100;  static constexpr GLint sphere\_stacks = 100;  static constexpr glm::dvec3 cone\_start\_pos = glm::dvec3(-20, 0, -19);  static constexpr glm::dvec3 sphere\_start\_pos = glm::dvec3(20, 0, -19.0);  }  #pragma once  #include <constants.h>  #include <states/states.h>  namespace states {  class TorStretchCylinderState : public GLState  {  public:  GLState\* display() override final;  void timeout() override final;  private:  size\_t frames\_count = 0;  };  } // namespace states  #include <states/tor\_stretch\_cylinder\_state/tor\_stretch\_cylinder\_state.h>  #include <iostream>  #include <cmath>  #include <GL/glu.h>  #include <GL/glut.h> // Include GLUT for cool drawing utils  #include <state\_machine/state\_machine.h>  #include <states/tor\_cylinder\_state/constants.h>  #include <states/tor\_stretch\_cylinder\_state/constants.h>  using namespace states::tor\_cylinder\_state::constants;  using namespace states::tor\_stretch\_cylinder\_state::constants;  namespace states {  GLState\* TorStretchCylinderState::display()  {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // Clear both color and depth buffers  // --- Draw the solid cylinder ---  glPushMatrix(); // Save the current matrix state  glLoadIdentity();  // Create a quadric object for the cylinder  GLUquadric\* quadric = gluNewQuadric();  frames\_count = frames\_count > tor\_stretch\_cylinder\_state::constants::MAX\_FRAMES\_COUNT ? tor\_stretch\_cylinder\_state::constants::MAX\_FRAMES\_COUNT : frames\_count;  constexpr auto max\_frames = tor\_stretch\_cylinder\_state::constants::MAX\_FRAMES\_COUNT;  glScaled(torus\_result\_scale.x, torus\_result\_scale.y - (1.0 \* max\_frames - frames\_count) / max\_frames, 1);  glTranslated(torus\_start\_pos.x, torus\_start\_pos.y, torus\_start\_pos.z);  glutWireTorus(torus\_inner\_radius, torus\_outer\_radius, torus\_sides, torus\_rings);  // --- Draw the wireframe cylinder on top ---  glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE); // Switch to wireframe mode  // Set the color for the wireframe  glColor3f(0.0, 1.0, 0.0); // Green color for the wireframe lines  glPopMatrix(); // Restore the previous matrix state  glPushMatrix(); // Save the current matrix state  glRotated(90, 1, 0, 0);  glTranslated(cylinder\_start\_pos.y, -cylinder\_start\_pos.x, cylinder\_start\_pos.z);  // Draw the wireframe cylinder (same dimensions for the overlay)  gluCylinder(quadric, cylinder\_base, cylinder\_top, cylinder\_height, cylinder\_slices, cylinder\_stacks); // Same cylinder dimensions as the solid, increased slices  // Reset the polygon mode to solid rendering  glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);  // Clean up the quadric object  gluDeleteQuadric(quadric);  glPopMatrix(); // Restore the previous matrix state  // Swap the buffers to display the rendered frame  glutSwapBuffers();  return this;  }  void TorStretchCylinderState::timeout()  {  std::cout << \_\_PRETTY\_FUNCTION\_\_ << " timeout " << frames\_count << std::endl;  frames\_count++;  glutPostRedisplay();  // if (frames\_count == MAX\_FRAMES\_COUNT)  // state\_machine::StateMachine::instance()->set\_state(new states::ConusSphereMoveState());  }  } // namespace states  #pragma once  #include <GL/glut.h> // Include GLUT for cool drawing utils  #include <glm/vec3.hpp>  #include <constants.h>  namespace states::tor\_stretch\_cylinder\_state::constants {  static constexpr size\_t STAY\_SECONDS = 1;  static constexpr size\_t MAX\_FRAMES\_COUNT = ::constants::FRAMES\_PER\_SECONDS \* STAY\_SECONDS;  static constexpr glm::dvec3 torus\_result\_scale = glm::dvec3(1, 2, 1);  }  #include <states/tor\_cylinder\_rotate\_state/tor\_cylinder\_rotate\_state.h>  #include <iostream>  #include <GL/glu.h>  #include <GL/glut.h> // Include GLUT for cool drawing utils  #include <state\_machine/state\_machine.h>  #include <states/tor\_cylinder\_rotate\_state/constants.h>  #include <states/tor\_cylinder\_state/constants.h>  #include <states/tor\_stretch\_cylinder\_state/tor\_stretch\_cylinder\_state.h>  using namespace states::tor\_cylinder\_rotate\_state::constants;  using namespace states::tor\_cylinder\_state::constants;  namespace states {  GLState\* TorCylinderRotateState::display()  {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // Clear both color and depth buffers  // --- Draw the solid cylinder ---  glPushMatrix(); // Save the current matrix state  // Create a quadric object for the cylinder  GLUquadric\* quadric = gluNewQuadric();  glTranslated(torus\_start\_pos.x, torus\_start\_pos.y, torus\_start\_pos.z);  glutWireTorus(torus\_inner\_radius, torus\_outer\_radius, torus\_sides, torus\_rings);  // --- Draw the wireframe cylinder on top ---  glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE); // Switch to wireframe mode  // Set the color for the wireframe  glColor3f(0.0, 1.0, 0.0); // Green color for the wireframe lines  glPopMatrix(); // Restore the previous matrix state  glPushMatrix(); // Save the current matrix state  constexpr auto max\_frames\_count = tor\_cylinder\_rotate\_state::constants::MAX\_FRAMES\_COUNT;  frames\_count = frames\_count > max\_frames\_count ? max\_frames\_count : frames\_count;  glRotated(90 + frames\_count \* 360 / max\_frames\_count, 1, 0, 0);  glTranslated(cylinder\_start\_pos.y, -cylinder\_start\_pos.x, cylinder\_start\_pos.z);  // Draw the wireframe cylinder (same dimensions for the overlay)  gluCylinder(quadric, cylinder\_base, cylinder\_top, cylinder\_height, cylinder\_slices, cylinder\_stacks); // Same cylinder dimensions as the solid, increased slices  // Reset the polygon mode to solid rendering  glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);  // Clean up the quadric object  gluDeleteQuadric(quadric);  glPopMatrix(); // Restore the previous matrix state  // Swap the buffers to display the rendered frame  glutSwapBuffers();  return this;  }  void TorCylinderRotateState::timeout()  {  std::cout << \_\_func\_\_ << " timeout " << frames\_count << std::endl;  frames\_count++;  glutPostRedisplay();  if (frames\_count == ::states::tor\_cylinder\_rotate\_state::constants::MAX\_FRAMES\_COUNT)  state\_machine::StateMachine::instance()->set\_state(new states::TorStretchCylinderState());  }  } // namespace states  #pragma once  #include <constants.h>  #include <states/states.h>  namespace states {  class TorCylinderRotateState : public GLState  {  public:  GLState\* display() override final;  void timeout() override final;  private:  size\_t frames\_count = 0;  };  } // namespace states  #pragma once  #include <GL/glut.h> // Include GLUT for cool drawing utils  #include <glm/vec3.hpp>  #include <constants.h>  namespace states::tor\_cylinder\_rotate\_state::constants {  static constexpr size\_t STAY\_SECONDS = 4;  static constexpr size\_t MAX\_FRAMES\_COUNT = ::constants::FRAMES\_PER\_SECONDS \* STAY\_SECONDS;  }  #include <states/conus\_sphere\_moving\_state/conus\_sphere\_moving\_state.h>  #include <iostream>  #include <GL/glut.h> // Include GLUT for cool drawing utils  #include <state\_machine/state\_machine.h>  #include <states/conus\_sphere\_stay\_state/constants.h>  #include <states/conus\_sphere\_moving\_state/constants.h>  #include <states/tor\_cylinder\_state/tor\_cylinder\_state.h>  using namespace states::conus\_sphere\_moving\_state::constants;  using namespace states::conus\_sphere\_stay\_state::constants;  namespace states {  GLState\* ConusSphereMovingState::display()  {  glClear(GL\_COLOR\_BUFFER\_BIT);  glPushMatrix();  glLoadIdentity();  frames\_count = frames\_count <= MAX\_ANIMATION\_FRAMES ? frames\_count : MAX\_ANIMATION\_FRAMES;  glm::dvec3 new\_cone\_pos {  cone\_start\_pos.x + (cone\_end\_pos.x - cone\_start\_pos.x) / MAX\_ANIMATION\_FRAMES \* frames\_count,  cone\_start\_pos.y + (cone\_end\_pos.y - cone\_start\_pos.y) / MAX\_ANIMATION\_FRAMES \* frames\_count,  cone\_start\_pos.z + (cone\_end\_pos.z - cone\_start\_pos.z) / MAX\_ANIMATION\_FRAMES \* frames\_count,  };  glTranslated(new\_cone\_pos.x, new\_cone\_pos.y, new\_cone\_pos.z);  glRotated(90, -1, 0, 0);  glutWireCone(cone\_base, cone\_height, cone\_slices, cone\_stacks);  glPopMatrix();  glLoadIdentity();  glPushMatrix();  glLoadIdentity();  glm::dvec3 new\_sphere\_pos {  sphere\_start\_pos.x + (sphere\_end\_pos.x - sphere\_start\_pos.x) / MAX\_ANIMATION\_FRAMES \* frames\_count,  sphere\_start\_pos.y + (sphere\_end\_pos.y - sphere\_start\_pos.y) / MAX\_ANIMATION\_FRAMES \* frames\_count,  sphere\_start\_pos.z + (sphere\_end\_pos.z - sphere\_start\_pos.z) / MAX\_ANIMATION\_FRAMES \* frames\_count,  };  glTranslated(new\_sphere\_pos.x, new\_sphere\_pos.y, new\_sphere\_pos.z);  glutWireSphere(sphere\_radius, sphere\_slices, sphere\_stacks);  glutSwapBuffers();  glPopMatrix();  return this;  }  void ConusSphereMovingState::timeout()  {  std::cout << "ConusSphereMovingState timeout " << frames\_count << std::endl;  frames\_count++;  glutPostRedisplay();  if (frames\_count == MAX\_FRAMES\_COUNT)  {  state\_machine::StateMachine::instance()->set\_state(new states::TorCylinderState());  }  }  } // namespace states  #pragma once  #include <constants.h>  #include <states/states.h>  namespace states {  class ConusSphereMovingState : public GLState  {  public:  GLState\* display() override final;  void timeout() override final;  private:  size\_t frames\_count = 0;  };  } // namespace states  #pragma once  #include <GL/glut.h> // Include GLUT for cool drawing utils  #include <glm/vec3.hpp>  #include <constants.h>  namespace states::conus\_sphere\_moving\_state::constants {  static constexpr size\_t ANIMATION\_SECONDS = 1;  static constexpr size\_t MAX\_ANIMATION\_FRAMES = ANIMATION\_SECONDS \* ::constants::FRAMES\_PER\_SECONDS;  static constexpr glm::dvec3 cone\_end\_pos = glm::dvec3(0, 0, -19);  static constexpr glm::dvec3 sphere\_end\_pos = glm::dvec3(0, 0, -19.0);  }  #include <states/tor\_cylinder\_state/tor\_cylinder\_state.h>  #include <iostream>  #include <GL/glu.h>  #include <GL/glut.h> // Include GLUT for cool drawing utils  #include <state\_machine/state\_machine.h>  #include <states/tor\_cylinder\_state/constants.h>  #include <states/tor\_cylinder\_rotate\_state/tor\_cylinder\_rotate\_state.h>  using namespace states::tor\_cylinder\_state::constants;  namespace states {  GLState\* TorCylinderState::display()  {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // Clear both color and depth buffers  // --- Draw the solid cylinder ---  glPushMatrix(); // Save the current matrix state  // Create a quadric object for the cylinder  GLUquadric\* quadric = gluNewQuadric();  glTranslated(torus\_start\_pos.x, torus\_start\_pos.y, torus\_start\_pos.z);  glutWireTorus(torus\_inner\_radius, torus\_outer\_radius, torus\_sides, torus\_rings);  // --- Draw the wireframe cylinder on top ---  glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE); // Switch to wireframe mode  // Set the color for the wireframe  glColor3f(0.0, 1.0, 0.0); // Green color for the wireframe lines  glPopMatrix(); // Restore the previous matrix state  glPushMatrix(); // Save the current matrix state  glRotated(90, 1, 0, 0);  glTranslated(cylinder\_start\_pos.y, -cylinder\_start\_pos.x, cylinder\_start\_pos.z);  // Draw the wireframe cylinder (same dimensions for the overlay)  gluCylinder(quadric, cylinder\_base, cylinder\_top, cylinder\_height, cylinder\_slices, cylinder\_stacks); // Same cylinder dimensions as the solid, increased slices  // Reset the polygon mode to solid rendering  glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);  // Clean up the quadric object  gluDeleteQuadric(quadric);  glPopMatrix(); // Restore the previous matrix state  // Swap the buffers to display the rendered frame  glutSwapBuffers();  return this;  }  void TorCylinderState::timeout()  {  std::cout << \_\_func\_\_ << " timeout " << frames\_count << std::endl;  frames\_count++;  glutPostRedisplay();  if (frames\_count == MAX\_FRAMES\_COUNT)  state\_machine::StateMachine::instance()->set\_state(new states::TorCylinderRotateState());  }  } // namespace states  #pragma once  #include <GL/glut.h> // Include GLUT for cool drawing utils  #include <glm/vec3.hpp>  #include <constants.h>  namespace states::tor\_cylinder\_state::constants {  static constexpr size\_t STAY\_SECONDS = 1;  static constexpr size\_t MAX\_FRAMES\_COUNT = ::constants::FRAMES\_PER\_SECONDS \* STAY\_SECONDS;  static constexpr GLdouble torus\_inner\_radius = 5;  static constexpr GLdouble torus\_outer\_radius = 10;  static constexpr GLint torus\_sides = 100;  static constexpr GLint torus\_rings = 100;  static constexpr GLint cylinder\_base = 10;  static constexpr GLint cylinder\_top = 10;  static constexpr GLint cylinder\_height = 40;  static constexpr GLint cylinder\_slices = 50;  static constexpr GLint cylinder\_stacks = 50;  static constexpr glm::dvec3 torus\_start\_pos = glm::dvec3(-10, 0, 5);  static constexpr glm::dvec3 cylinder\_start\_pos = glm::dvec3(-10, 10, -20);  }  #pragma once  #include <constants.h>  #include <states/states.h>  namespace states {  class TorCylinderState : public GLState  {  public:  GLState\* display() override final;  void timeout() override final;  private:  size\_t frames\_count = 0;  };  } // namespace states  #pragma once  #include <memory>  namespace states {  enum State  {  CONUS\_SPHERE\_STAY,  CONUS\_SPHERE\_MOVE,  TOR\_CYLLINDRE\_STAY,  CYLLINDRE\_SPIN,  TOR\_STRETCH  };  class GLState  {  public:  virtual GLState\* display() = 0;  virtual void timeout() = 0;  virtual void keyboard\_callback(unsigned char key, int param1, int param2);  virtual void mouse(int button, int state, int x, int y);  virtual void reshape(int w, int h);  };  } // namespace states  #include <states/states.h>  #include <iostream>  #include <GL/glut.h>  #include <GL/gl.h>  #include "states.h"  void states::GLState::keyboard\_callback(unsigned char key, int param1, int param2)  {  if (key == 'q') {  std::cout << "Quitting application";  glutDestroyWindow(param1);  exit(0);  }  }  void states::GLState::mouse(int button, int state, int x, int y) {}  void states::GLState::reshape(int w, int h)  {  glViewport(0, 0, (GLsizei)w, (GLsizei)h);  glMatrixMode(GL\_PROJECTION);  glLoadIdentity();  glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);  glMatrixMode(GL\_MODELVIEW);  glLoadIdentity();  }  #pragma once  #include <stdio.h> // For size\_t  #include <GL/gl.h>  namespace constants  {  static constexpr size\_t FRAMES\_PER\_SECONDS = 60;  // Perspective Projection Settings  static constexpr GLdouble FOV\_Y = 45.0; // Field of view in the Y direction  static constexpr GLdouble NEAR\_PLANE = 0.1; // Near clipping plane  static constexpr GLdouble FAR\_PLANE = 100.0; // Far clipping plane  // Camera Settings (move the camera further back and slightly up)  static constexpr GLdouble CAMERA\_POS\_X = 0.0; // Camera position X  static constexpr GLdouble CAMERA\_POS\_Y = 5.0; // Move camera slightly up along the Y-axis  static constexpr GLdouble CAMERA\_POS\_Z = 15.0; // Move camera further back along the Z-axis  static constexpr GLdouble LOOK\_AT\_X = 0.0; // Look-at point X  static constexpr GLdouble LOOK\_AT\_Y = 0.0; // Look-at point Y  static constexpr GLdouble LOOK\_AT\_Z = 0.0; // Look-at point Z  static constexpr GLdouble UP\_X = 0.0; // Up vector X  static constexpr GLdouble UP\_Y = 1.0; // Up vector Y  static constexpr GLdouble UP\_Z = 0.0; // Up vector Z  // Window Settings  static constexpr size\_t WINDOW\_WIDTH = 800;  static constexpr size\_t WINDOW\_HEIGHT = 600;  static constexpr size\_t WINDOW\_POSITION\_X = 0;  static constexpr size\_t WINDOW\_POSITION\_Y = 0;  }  #include <state\_machine/state\_machine.h>  #include <iostream>  #include <GL/glut.h>  void state\_machine::StateMachine::keyboard\_callback(unsigned char key, int param1, int param2) {  if (current\_state)  current\_state->keyboard\_callback(key, param1, param2);  else  std::cout << "No state set" << std::endl;  }  void state\_machine::StateMachine::mouse(int button, int state, int x, int y) {  if (current\_state)  current\_state->mouse(button, state, x, y);  else  std::cout << "No state set" << std::endl;  }  void state\_machine::StateMachine::reshape(int w, int h) {  if (current\_state)  current\_state->reshape(w, h);  else  std::cout << "No state set" << std::endl;  }  void state\_machine::StateMachine::display() {  if (current\_state)  current\_state->display();  else  {  std::cout << "No state set" << std::endl;  glClearColor(0.0, 0.0, 0.0, 1);  glClear(GL\_COLOR\_BUFFER\_BIT);  glutSwapBuffers();  }  }  bool state\_machine::StateMachine::timeout() {  if (current\_state)  {  current\_state->timeout();  return true;  }  else  return false;  }  #pragma once  #include <states/states.h>  namespace state\_machine {  class StateMachine  {  public:  void keyboard\_callback(unsigned char key, int param1, int param2);  void mouse(int button, int state, int x, int y);  void reshape(int w, int h);  void display();  bool timeout();  void set\_state(states::GLState\* state) {  if (current\_state != nullptr) {  delete current\_state;  current\_state = nullptr;  }  current\_state = state;  }  static StateMachine \*instance()  {  static StateMachine instance;  return &instance;  }  private:  states::GLState\* current\_state = nullptr;  StateMachine() = default;  };  } // namespace state\_machine  #pragma once  #include <iostream>  #include <GL/glut.h> // Include GLUT for cool drawing utils  #include <states/conus\_sphere\_stay\_state/conus\_sphere\_stay\_state.h>  #include <state\_machine/state\_machine.h>  #include <constants.h>  state\_machine::StateMachine \*state\_machine\_obj = state\_machine::StateMachine::instance();  void init()  {  glMatrixMode(GL\_PROJECTION); // Switch to projection matrix  glLoadIdentity(); // Reset the projection matrix  // Set up a perspective projection using the constants  gluPerspective(constants::FOV\_Y,  (double)constants::WINDOW\_WIDTH / (double)constants::WINDOW\_HEIGHT,  constants::NEAR\_PLANE,  constants::FAR\_PLANE);  // Set the camera position and look-at point using the constants  gluLookAt(constants::CAMERA\_POS\_X, constants::CAMERA\_POS\_Y, constants::CAMERA\_POS\_Z, // Camera position  constants::LOOK\_AT\_X, constants::LOOK\_AT\_Y, constants::LOOK\_AT\_Z, // Look-at point  constants::UP\_X, constants::UP\_Y, constants::UP\_Z); // Up vector  glMatrixMode(GL\_MODELVIEW); // Switch back to model-view matrix  glLoadIdentity(); // Reset the model-view matrix  glEnable(GL\_DEPTH\_TEST); // Enable depth testing  glDisable(GL\_CULL\_FACE); // Disable face culling  }  void display()  {  state\_machine\_obj->display();  }  void reshape(int w, int h)  {  state\_machine\_obj->reshape(w, h);  }  void keyboard\_callback(unsigned char key, int param1, int param2)  {  state\_machine\_obj->keyboard\_callback(key, param1, param2);  }  void mouse(int button, int state, int x, int y)  {  state\_machine\_obj->mouse(button, state, x, y);  }  void timeout(int /\*unused\*/)  {  if (state\_machine\_obj->timeout())  glutTimerFunc(1000 / constants::FRAMES\_PER\_SECONDS, timeout, 0);  }  //Запросить режим двойной буферизации  //Зарегистрировать функции обработки мышиного ввода  int main(int argc, char\*\* argv)  {  glutInit(&argc, argv);  glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);  init();  glutInitWindowSize(constants::WINDOW\_WIDTH, constants::WINDOW\_HEIGHT);  glutInitWindowPosition(constants::WINDOW\_POSITION\_X, constants::WINDOW\_POSITION\_Y);    glutCreateWindow("Simple GLUT App with animations");  state\_machine\_obj->set\_state(new states::ConusSphereStayState());  glutTimerFunc(1000 / constants::FRAMES\_PER\_SECONDS, timeout, 0);  glutDisplayFunc(display);  glutReshapeFunc(reshape);  glutMouseFunc(mouse);  glutKeyboardFunc(keyboard\_callback);  glutMainLoop();  return 0;  } |

# Вывод

В процессе выполнения работы мы изучили основы векторной графики и освоили базовые функции графического API, необходимые для создания и обработки трехмерных графических примитивов. Мы также ознакомились с библиотекой OpenGL, исследуя методы создания, позиционирования, масштабирования и перемещения трехмерных объектов.