ПРАКТИЧЕСКАЯ РАБОТА №2

по дисциплине

«Компьютерная графика»

**Вариант 1**

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

Оглавление

[Цель работы 3](#__RefHeading___Toc4389_1830874876)

[Порядок выполнения 3](#__RefHeading___Toc4391_1830874876)

[Результат работы 4](#__RefHeading___Toc4393_1830874876)

[Код программы 5](#__RefHeading___Toc4395_1830874876)

[Вывод 25](#__RefHeading___Toc4397_1830874876)

# Цель работы

Целью работы является ознакомление с основными функциями API, описывающими свойства материалов объектов и позволяющими задавать параметры источника освещения.

# Порядок выполнения

Требуется разработать программу, изображающую заданный набор из трех предметов с указанными свойствами материалов и параметры источника освещения. При этом в качестве базового набора объектов выступают 3D примитивы, указанные в вашем варианте задания №1. (**Конус, сфера, циллиндр**).

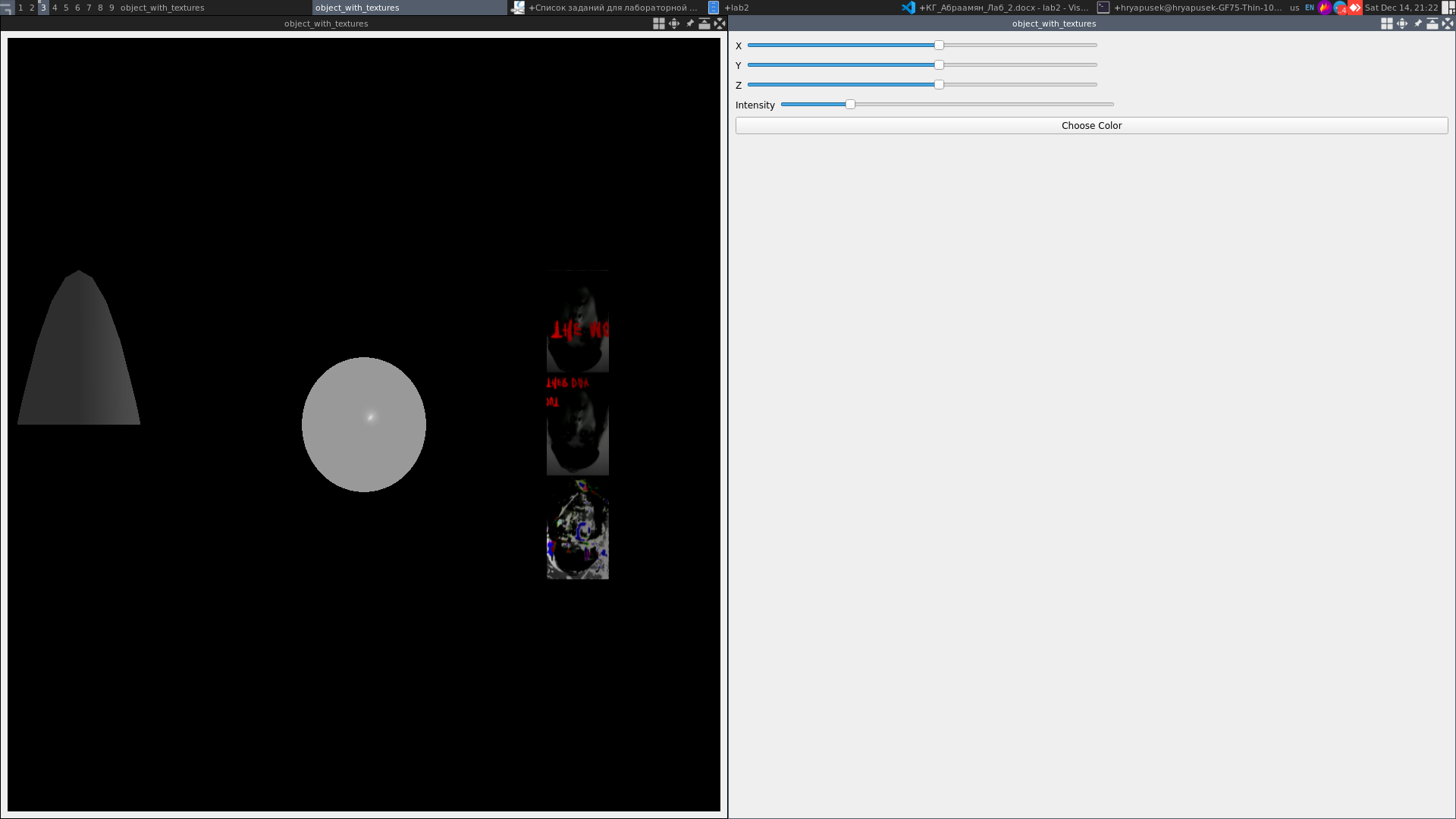
Следует наделить один из объектов свойствами прозрачности (значение параметра должно быть выше 0,5). Другой выбранный объект должен имитировать отполированную поверхность (shininess, значение указывается максимальным). В качестве такого объекта следует выбирать примитивы с выпуклыми поверхностями, например - цилиндр, тор, конус, сферу, чайник. Третий объект должен быть диффузно-рассеивающим, матовым.

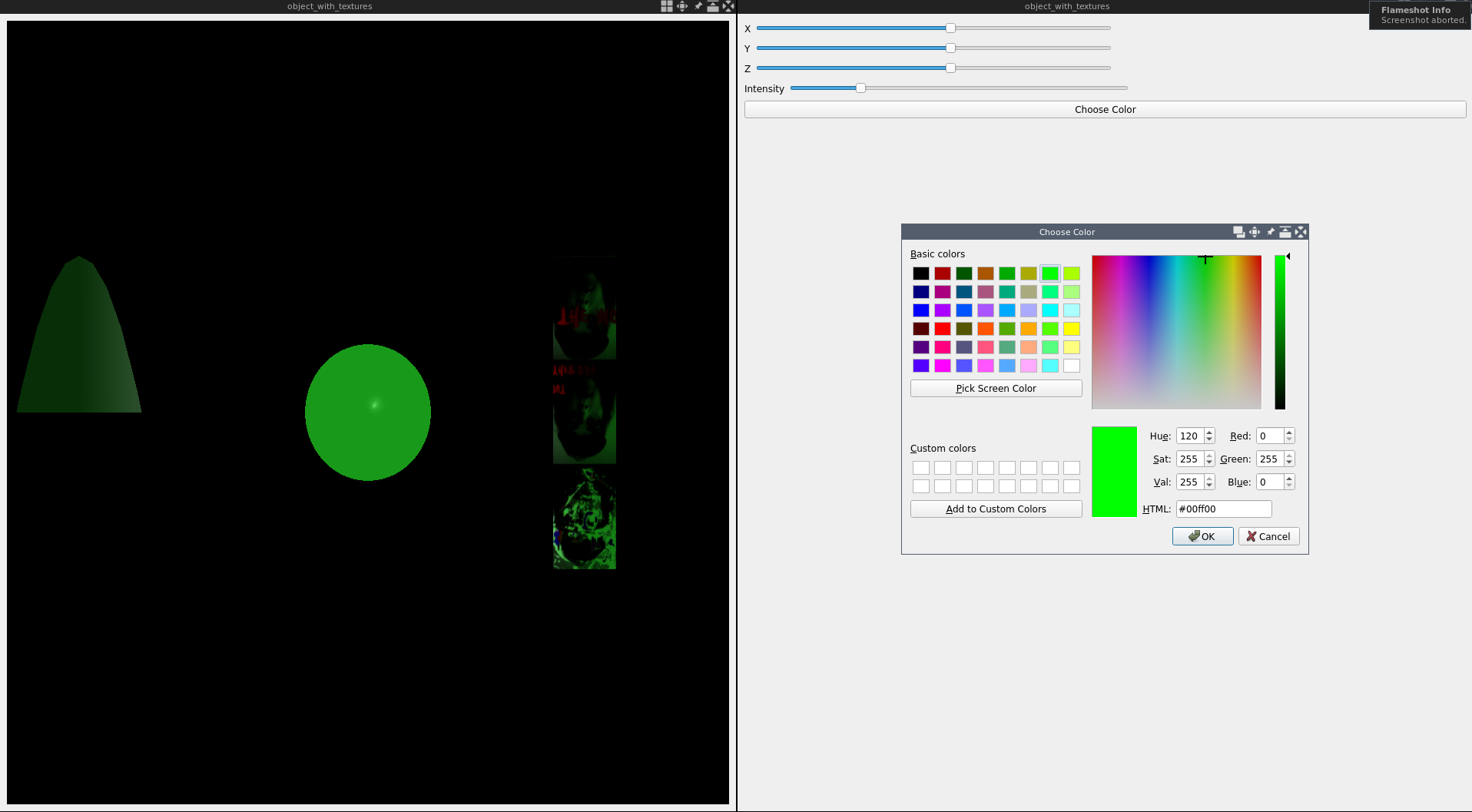
# Результат работы

Были построены фигуры:

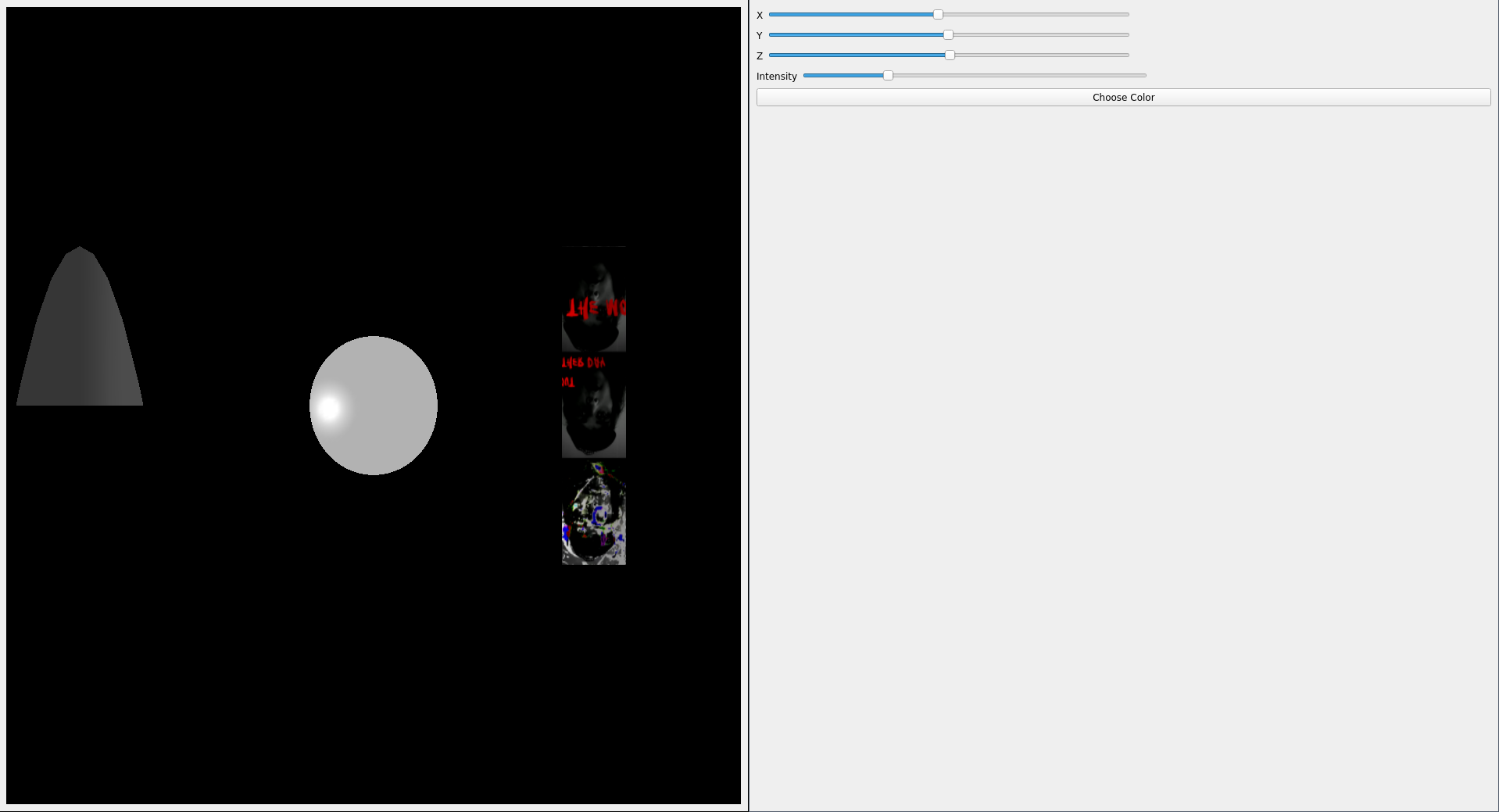
1. Прозрачный конус
2. Отполированная сфера
3. Текстурированный циллиндр

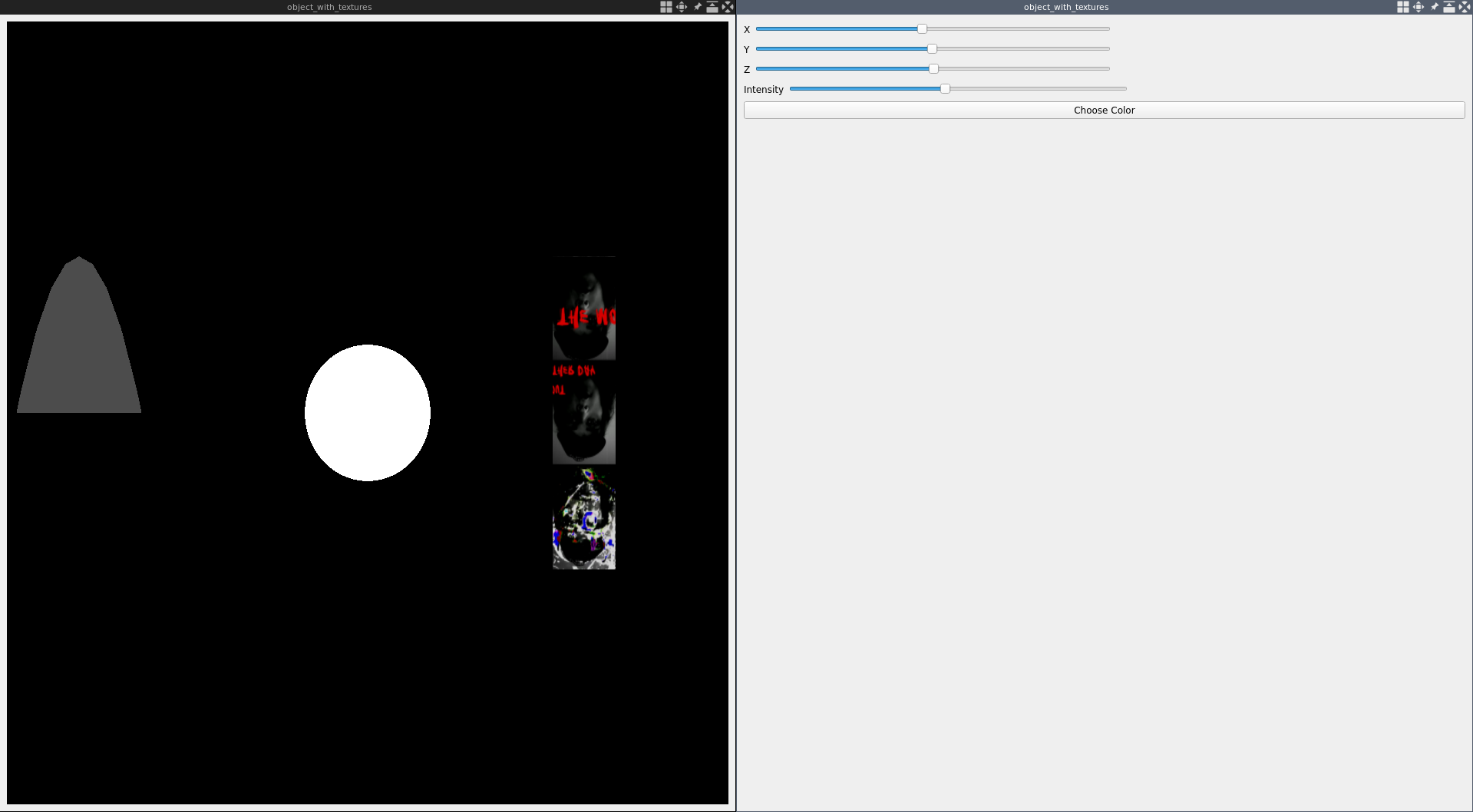
Установлен источник света, который при перемещении ползунка перемещается во всех плоскостях, меняет цвет и яркость. Powered by Qt.

**Изменение цвета источника света**



**Перемещение источника света**

**Изменение яркости источника света**

****

# Код программы

|  |
| --- |
| #pragma once  #include <QWidget>  #include <QMainWindow>  #include <QVBoxLayout>  #include <QHBoxLayout>  #include <QSlider>  #include <QPushButton>  #include <glwidget/glwidget.h>  #include <custom\_widgets/fancy\_slider.h>  namespace params\_window\_ns {  class ParamsWindow : public QMainWindow  {  Q\_OBJECT  public:  ParamsWindow(QWidget \*parent = nullptr);  signals:  void sig\_light\_position\_updated(glm::fvec3 position);  void sig\_light\_position\_updated\_x(GLfloat new\_x);  void sig\_light\_position\_updated\_y(GLfloat new\_y);  void sig\_light\_position\_updated\_z(GLfloat new\_z);  void sig\_color\_changed(QColor color);  void sig\_light\_intensity\_updated(GLfloat new\_intensity);  private:  QWidget \*main\_widget = nullptr;  QVBoxLayout \*main\_layout = nullptr;  custom\_widgets::FancySlider \*x\_slider = nullptr;  custom\_widgets::FancySlider \*y\_slider = nullptr;  custom\_widgets::FancySlider \*z\_slider = nullptr;  QPushButton \*choose\_color\_button = nullptr;  custom\_widgets::FancySlider \*light\_intensity\_slider = nullptr;  };  } // namespace main\_window\_ns  #include <QLabel>  #include <params\_window/params\_window.h>  #include <glwidget/glwidget.h>  #include <QColorDialog>  namespace params\_window\_ns {  ParamsWindow::ParamsWindow(QWidget\* parent)  : QMainWindow(parent)  {  main\_widget = new QWidget(this);  main\_layout = new QVBoxLayout(main\_widget);  setCentralWidget(main\_widget);  x\_slider = new custom\_widgets::FancySlider(Qt::Horizontal);  y\_slider = new custom\_widgets::FancySlider(Qt::Horizontal);  z\_slider = new custom\_widgets::FancySlider(Qt::Horizontal);  light\_intensity\_slider = new custom\_widgets::FancySlider(Qt::Horizontal);  for (auto& slider : {x\_slider, y\_slider, z\_slider}) {  slider->setMinimum(-1000);  slider->setMaximum(1000);  slider->setValue(100);  slider->setTickPosition(QSlider::TickPosition::NoTicks);  }  {  QHBoxLayout\* pair\_layout = new QHBoxLayout;  QLabel\* label = new QLabel("X");  pair\_layout->addWidget(label);  pair\_layout->addWidget(x\_slider);  main\_layout->addLayout(pair\_layout);  connect(x\_slider, &QSlider::valueChanged,  [this](int value) { sig\_light\_position\_updated\_x(value / 10.0f); });  pair\_layout->addSpacerItem(new QSpacerItem(40, 20, QSizePolicy::Expanding, QSizePolicy::Minimum));  }  {  QHBoxLayout\* pair\_layout = new QHBoxLayout;  QLabel\* label = new QLabel("Y");  pair\_layout->addWidget(label);  pair\_layout->addWidget(y\_slider);  main\_layout->addLayout(pair\_layout);  connect(y\_slider, &QSlider::valueChanged,  [this](int value) { sig\_light\_position\_updated\_y(value / 10.0f); });  pair\_layout->addSpacerItem(new QSpacerItem(40, 20, QSizePolicy::Expanding, QSizePolicy::Minimum));  }  {  QHBoxLayout\* pair\_layout = new QHBoxLayout;  QLabel\* label = new QLabel("Z");  pair\_layout->addWidget(label);  pair\_layout->addWidget(z\_slider);  main\_layout->addLayout(pair\_layout);  connect(z\_slider, &QSlider::valueChanged,  [this](int value) { sig\_light\_position\_updated\_z(value / 10.0f); });  pair\_layout->addSpacerItem(new QSpacerItem(40, 20, QSizePolicy::Expanding, QSizePolicy::Minimum));  }  {  QHBoxLayout\* pair\_layout = new QHBoxLayout;  QLabel\* label = new QLabel("Intensity");  pair\_layout->addWidget(label);  pair\_layout->addWidget(light\_intensity\_slider);  main\_layout->addLayout(pair\_layout);  connect(light\_intensity\_slider, &QSlider::valueChanged,  [this](int value) { sig\_light\_intensity\_updated(value / 10.0f); });  pair\_layout->addSpacerItem(new QSpacerItem(40, 20, QSizePolicy::Expanding, QSizePolicy::Minimum));  }  light\_intensity\_slider->setMinimum(0);  light\_intensity\_slider->setMaximum(50);  light\_intensity\_slider->setValue(10);  light\_intensity\_slider->setTickPosition(QSlider::TickPosition::NoTicks);  choose\_color\_button = new QPushButton();  choose\_color\_button->setText("Choose Color");  connect(choose\_color\_button, &QPushButton::clicked, [this]() {  auto new\_color = QColorDialog::getColor(Qt::black, this, "Choose Color");  if (new\_color.isValid()) {  sig\_color\_changed(new\_color);  }  });  main\_layout->addWidget(choose\_color\_button);  main\_layout->addSpacerItem(new QSpacerItem(20, 20, QSizePolicy::Expanding, QSizePolicy::Expanding));  }  } // namespace params\_window\_ns  #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\_stay\_state/constants.h>  #include <states/tor\_cylinder\_state/constants.h>  #include <utils/utils.h>  using namespace states::conus\_sphere\_stay\_state::constants;  using namespace states::tor\_cylinder\_state::constants;  namespace states {  GLState\* ConusSphereStayState::display()  {  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // Clear both color and depth buffers  // --- Draw Opaque Objects First ---  glPushMatrix();  glLoadIdentity();  // Draw opaque sphere  glTranslated(sphere\_start\_pos.x, sphere\_start\_pos.y, sphere\_start\_pos.z);  glColor4f(1.0f, 1.0f, 1.0f, 1.0f); // Fully opaque  glMaterialfv(GL\_FRONT, GL\_SPECULAR, states::conus\_sphere\_stay\_state::constants::mat\_specular);  glMaterialfv(GL\_FRONT, GL\_SHININESS, states::conus\_sphere\_stay\_state::constants::mat\_shininess);  glutSolidSphere(sphere\_radius, sphere\_slices, sphere\_stacks); // Draw filled sphere  glPopMatrix();  // --- Draw Textured Cylinder ---  glPushMatrix(); // Save the current matrix state  static GLuint textureID = utils::loadTexture(cone\_texture\_path);  // Create a quadric object for the cylinder  GLUquadric\* quadric = gluNewQuadric();  gluQuadricTexture(quadric, GL\_TRUE);  gluQuadricNormals(quadric, GLU\_SMOOTH);  glRotated(90, 1, 0, 0);  glTranslated(cylinder\_start\_pos.y + 20, -cylinder\_start\_pos.x, cylinder\_start\_pos.z);  glMaterialfv(GL\_FRONT, GL\_AMBIENT, mat\_ambient);  glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse);  glMaterialfv(GL\_FRONT, GL\_SPECULAR, states::tor\_cylinder\_state::constants::mat\_specular);  glMaterialfv(GL\_FRONT, GL\_SHININESS, states::tor\_cylinder\_state::constants::mat\_shininess);  glBindTexture(GL\_TEXTURE\_2D, textureID);  // Draw the textured cylinder  gluCylinder(quadric, cylinder\_base, cylinder\_top, cylinder\_height, cylinder\_slices,  cylinder\_stacks);  glBindTexture(GL\_TEXTURE\_2D, 0); // Unbind texture  // Clean up the quadric object  gluDeleteQuadric(quadric);  glPopMatrix(); // Restore the previous matrix state  // --- Draw Transparent Cone Last ---  // Enable blending for transparency  glEnable(GL\_BLEND);  glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);  // Disable depth writing for the transparent object  glDepthMask(GL\_FALSE);  glPushMatrix();  glLoadIdentity();  glTranslated(cone\_start\_pos.x, cone\_start\_pos.y, cone\_start\_pos.z);  glRotated(90, -1, 0, 0);  // Set color with transparency (for lighting)  glColor4f(1.0f, 1.0f, 1.0f, constants::LOWER\_ALPHA);  // Set the material with transparency  GLfloat mat\_diffuse\_with\_alpha[] = {1.0f, 1.0f, 1.0f, constants::LOWER\_ALPHA};  glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_diffuse\_with\_alpha);  // Draw the transparent cone  glutSolidCone(cone\_base, cone\_height, cone\_slices, cone\_stacks);  // Restore depth writing and disable blending  glDepthMask(GL\_TRUE);  glDisable(GL\_BLEND);  glPopMatrix();  return this;  }  void ConusSphereStayState::timeout()  {  std::cout << "ConusSphereStayState timeout " << frames\_count << std::endl;  frames\_count++;  // 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 = 60;  static constexpr GLint cone\_stacks = 300;  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(-40, 0, -19);  static constexpr glm::dvec3 sphere\_start\_pos = glm::dvec3(0, 0, -19.0);  static constexpr GLfloat mat\_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };  static constexpr GLfloat mat\_shininess[] = { 120.0f }; // Shininess factor (higher values mean more polished)  }  #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);  static constexpr GLfloat mat\_ambient[] = { 0.5f, 0.5f, 0.5f, 1.0f }; // Low intensity ambient (grey)  static constexpr GLfloat mat\_diffuse[] = { 1.0f, 1.0f, 1.0f, 0.5f }; // Bright green diffuse (matte green)  static constexpr GLfloat mat\_specular[] = { 0.0f, 0.0f, 0.0f, 1.0f }; // No specular highlights for matte  static constexpr GLfloat mat\_shininess[] = { 0.0f }; // Shininess (set to 0 for matte finish)  static constexpr const char \*cone\_texture\_path = "textures/cone\_texture.png";  }  #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 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::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();  }  #include "glwidget.h"  #include <iostream>  #include <GL/glut.h> // Include GLUT for cool drawing utils  #include <QWindow>  #include <QOpenGLContext>  #include <states/conus\_sphere\_stay\_state/conus\_sphere\_stay\_state.h>  #include <constants.h>  namespace glwidget {  GLWidget::GLWidget(QWidget\* parent)  : QGLWidget(parent)  {  }  void GLWidget::initializeGL()  {  glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);  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  glDepthFunc(GL\_LEQUAL); // Set the type of depth-test  glDisable(GL\_CULL\_FACE); // Disable face culling  glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);  glEnable(GL\_BLEND);  glEnable(GL\_LIGHTING);  glEnable(GL\_LIGHT0);  glEnable(GL\_TEXTURE\_2D);  startTimer(1000 / constants::FRAMES\_PER\_SECONDS);  }  void GLWidget::paintGL()  {  paint\_light();  state\_machine\_obj->display();  }  void GLWidget::resizeGL(int width, int height) { state\_machine\_obj->reshape(width, height); }  void GLWidget::timerEvent(QTimerEvent\* event)  {  state\_machine\_obj->timeout();  update();  QGLWidget::timerEvent(event);  }  void GLWidget::set\_light\_position(glm::fvec3 position)  {  light\_position[0] = position.x;  light\_position[1] = position.y;  light\_position[2] = position.z;  }  void GLWidget::paint\_light()  {  std::array<GLfloat, 4> intensity\_light\_ambient = {};  std::array<GLfloat, 4> intensity\_light\_diffuse = {};  std::array<GLfloat, 4> intensity\_light\_specular = {};  std::array<GLfloat, 4> intensity\_light\_position = {};  std::transform(light\_ambient.begin(), light\_ambient.end(), intensity\_light\_ambient.begin(),  [this](GLfloat val) { return val \* light\_intensity; });  std::transform(light\_diffuse.begin(), light\_diffuse.end(), intensity\_light\_diffuse.begin(),  [this](GLfloat val) { return val \* light\_intensity; });  std::transform(light\_specular.begin(), light\_specular.end(), intensity\_light\_specular.begin(),  [this](GLfloat val) { return val \* light\_intensity; });  std::transform(light\_position.begin(), light\_position.end(), intensity\_light\_position.begin(),  [this](GLfloat val) { return val \* light\_intensity; });  glLightfv(GL\_LIGHT0, GL\_AMBIENT, intensity\_light\_ambient.data()); // Ambient light  glLightfv(GL\_LIGHT0, GL\_DIFFUSE, intensity\_light\_diffuse.data()); // Diffuse light  glLightfv(GL\_LIGHT0, GL\_SPECULAR, intensity\_light\_specular.data()); // Specular light  glLightfv(GL\_LIGHT0, GL\_POSITION, intensity\_light\_position.data());  }  } // namespace glwidget  #pragma once  #include <QGLWidget>  #include <QWidget>  #include <glm/glm.hpp>  #include <state\_machine/state\_machine.h>  namespace glwidget {  class GLWidget : public QGLWidget  {  public:  GLWidget(QWidget\* parent);  virtual ~GLWidget() = default;  void initializeGL() override;  void paintGL() override;  void resizeGL(int width, int height) override;  void timerEvent(QTimerEvent\* event) override;  public slots:  void set\_light\_position(glm::fvec3 position);  void set\_light\_position\_x(GLfloat x) {  light\_position[0] = x;  }  void set\_light\_position\_y(GLfloat y) {  light\_position[1] = y;  }  void set\_light\_position\_z(GLfloat z) {  light\_position[2] = z;  }  void set\_light\_color(QColor color) {  light\_ambient[0] = color.redF();  light\_ambient[1] = color.greenF();  light\_ambient[2] = color.blueF();  }  void set\_light\_intensity(GLfloat new\_intensity) {  light\_intensity = new\_intensity;  }  protected:  state\_machine::StateMachine\* state\_machine\_obj = state\_machine::StateMachine::instance();  std::array<GLfloat, 4> light\_position = {1.0f, 1.0f, 1.0f, 1.0f};  std::array<GLfloat, 4> light\_ambient = { 1.f, 1.f, 1.f, 1.0f }; // Low intensity ambient light  std::array<GLfloat, 4> light\_diffuse = { 1.0f, 1.0f, 1.0f, 1.0f }; // Full intensity white diffuse light  std::array<GLfloat, 4> light\_specular = { 1.0f, 1.0f, 1.0f, 1.0f }; // Full intensity white specular light  GLfloat light\_intensity = 1.0f;  void paint\_light();  };  } // namespace glwidget  #include <QLabel>  #include <main\_window/main\_window.h>  #include <glwidget/glwidget.h>  namespace main\_window\_ns {  MainWindow::MainWindow()  {  gl\_widget = new glwidget::GLWidget(this);  main\_widget = new QWidget(this);  main\_layout = new QVBoxLayout(main\_widget);  setCentralWidget(main\_widget);  main\_layout->addWidget(gl\_widget);  params\_window = new params\_window\_ns::ParamsWindow(this);  connect(params\_window, &params\_window\_ns::ParamsWindow::sig\_light\_position\_updated,  [this](glm::fvec3 position) { gl\_widget->set\_light\_position(position); });  connect(params\_window, &params\_window\_ns::ParamsWindow::sig\_light\_position\_updated\_x,  [this](GLfloat new\_x) { gl\_widget->set\_light\_position\_x(new\_x); });  connect(params\_window, &params\_window\_ns::ParamsWindow::sig\_light\_position\_updated\_y,  [this](GLfloat new\_y) { gl\_widget->set\_light\_position\_y(new\_y); });  connect(params\_window, &params\_window\_ns::ParamsWindow::sig\_light\_position\_updated\_z,  [this](GLfloat new\_z) { gl\_widget->set\_light\_position\_z(new\_z); });  connect(params\_window, &params\_window\_ns::ParamsWindow::sig\_color\_changed,  [this](QColor color) { gl\_widget->set\_light\_color(color); });    connect(params\_window, &params\_window\_ns::ParamsWindow::sig\_light\_intensity\_updated,  [this](GLfloat new\_intensity) { gl\_widget->set\_light\_intensity(new\_intensity); });  params\_window->show();  }  } // namespace main\_window\_ns  #pragma once  #include <QWidget>  #include <QMainWindow>  #include <QVBoxLayout>  #include <QHBoxLayout>  #include <QSlider>  #include <glwidget/glwidget.h>  #include <params\_window/params\_window.h>  namespace main\_window\_ns {  class MainWindow : public QMainWindow  {  public:  MainWindow();  private:  QWidget \*main\_widget = nullptr;  QVBoxLayout \*main\_layout = nullptr;  glwidget::GLWidget \*gl\_widget = nullptr;  params\_window\_ns::ParamsWindow \*params\_window = nullptr;  };  } // namespace main\_window\_ns  #pragma once  #include <stdio.h> // For size\_t  #include <GL/gl.h>  namespace constants  {  static constexpr size\_t FRAMES\_PER\_SECONDS = 60;  static constexpr GLfloat LOWER\_ALPHA = 0.3f;  // 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::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);  }  }  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 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  #include <utils/utils.h>  #include <iostream>  #include <map>  #include <memory>  #include <mutex>  #include <GL/glut.h>  #include <image\_loader/stb\_image.h>  namespace utils {  struct CachedTexture  {  GLuint id;  int width;  int height;  int nrChannels;  GLenum format;  std::unique\_ptr<unsigned char[]> data;  };  GLuint loadTexture(const std::string& filepath)  {  static std::map<std::string, CachedTexture> textureMap;  static std::once\_flag flag;  auto found\_texture = textureMap.find(filepath);  if (found\_texture == textureMap.end()) {  CachedTexture texture = {};  glGenTextures(1, &texture.id);  // Flip the image vertically during loading  stbi\_set\_flip\_vertically\_on\_load(true);  unsigned char\* data =  stbi\_load(filepath.c\_str(), &texture.width, &texture.height, &texture.nrChannels, 0);  if (!data) {  std::cerr << "Failed to load texture at path: " << filepath << std::endl;  return 0;  }  texture.data = std::unique\_ptr<unsigned char[]>(data);  if (texture.nrChannels == 1)  texture.format = GL\_RED;  else if (texture.nrChannels == 3)  texture.format = GL\_RGB;  else if (texture.nrChannels == 4)  texture.format = GL\_RGBA;  else {  throw std::runtime\_error("Unsupported number of channels: " +  std::to\_string(texture.nrChannels));  }  textureMap[filepath] = std::move(texture);  }  auto& texture = textureMap[filepath];  glBindTexture(GL\_TEXTURE\_2D, texture.id);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR\_MIPMAP\_LINEAR);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  // Upload the texture data  gluBuild2DMipmaps(GL\_TEXTURE\_2D, texture.format, texture.width, texture.height, texture.format,  GL\_UNSIGNED\_BYTE, texture.data.get());  return texture.id;  }  GLuint getTextureId(const std::string& filepath)  {  static std::map<std::string, GLuint> textureMap;  if (textureMap.find(filepath) == textureMap.end()) {  GLuint textureID = loadTexture(filepath);  textureMap[filepath] = textureID;  return textureID;  } else {  return textureMap[filepath];  }  }  } // namespace utils  #pragma once  #ifdef \_MSC\_VER  #define CURRENT\_FUNCTION \_\_FUNCTION\_\_  #elif defined(\_\_GNUC\_\_)  #define CURRENT\_FUNCTION \_\_PRETTY\_FUNCTION\_\_  #else  #define CURRENT\_FUNCTION \_\_func\_\_  #endif  #include <GL/gl.h>  #include <string>  namespace utils {  GLuint loadTexture(const std::string& filepath);  GLuint getTextureId(const std::string& filepath);  }  #include <iostream>  #include <GL/glut.h>  #include <QApplication>  #include <main\_window/main\_window.h>  #include <state\_machine/state\_machine.h>  #include <states/conus\_sphere\_stay\_state/conus\_sphere\_stay\_state.h>  #include <constants.h>  state\_machine::StateMachine \*state\_machine\_obj = state\_machine::StateMachine::instance();  //Запросить режим двойной буферизации  //Зарегистрировать функции обработки мышиного ввода  int main(int argc, char\*\* argv)  {  glutInit(&argc, argv);  QApplication app(argc, argv);  main\_window\_ns::MainWindow main\_window;  main\_window.resize(constants::WINDOW\_WIDTH, constants::WINDOW\_HEIGHT);  state\_machine\_obj->set\_state(new states::ConusSphereStayState());  main\_window.show();  return app.exec();  }  #include <custom\_widgets/fancy\_slider.h>  #include <QStyleOptionSlider>  #include <QToolTip>  namespace custom\_widgets {  FancySlider::FancySlider(QWidget\* parent)  : QSlider(parent)  {  }  FancySlider::FancySlider(Qt::Orientation orientation, QWidget\* parent)  : QSlider(orientation, parent)  {  }  void FancySlider::sliderChange(QAbstractSlider::SliderChange change)  {  QSlider::sliderChange(change);  if (change == QAbstractSlider::SliderValueChange) {  QStyleOptionSlider opt;  initStyleOption(&opt);  QRect sr = style()->subControlRect(QStyle::CC\_Slider, &opt, QStyle::SC\_SliderHandle, this);  QPoint bottomRightCorner = sr.bottomLeft();  QToolTip::showText(mapToGlobal(QPoint(bottomRightCorner.x(), bottomRightCorner.y())),  QString::number(value()), this);  }  }  } // namespace custom\_widgets  #pragma once  #include <QSlider>  namespace custom\_widgets {  class FancySlider : public QSlider  {  Q\_OBJECT  public:  explicit FancySlider(QWidget\* parent = 0);  explicit FancySlider(Qt::Orientation orientation, QWidget\* parent = 0);  protected:  virtual void sliderChange(SliderChange change);  };  } // namespace custom\_widgets |

# Вывод

В процессе работы были изучены ключевые функции API, которые описывают характеристики материалов объектов и позволяют настраивать параметры источников света. Также ознакомились с библиотекой OpenGL и процессом создания установки, а также настройки материалов для 3D-объектов и освещения.