МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ  
БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
Факультет информационных технологий и робототехники (ФИТР)  
  
Кафедра программного обеспечения информационных систем и технологий

**КУРСОВАЯ РАБОТА**по дисциплине: ”Разработка приложений в визуальных средах”  
  
на тему: ***”*** **Разработка приложения формирования и отображения сфер*”***

Выполнил**:** ст. гр. 10701120 Терещенко М.А.

Принял**:** доц. Гурский Н.Н.

Минск 2022

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра программного обеспечения информационных систем и технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовой работе**

**по дисциплине «Разработка приложений в визуальных средах»**

Тема: «Разработка приложения формирования и отображения сфер»

**Исполнитель**: Терещенко М.А. (подпись)

**Студент 2 курса 10701120 группы**

**Руководитель**: Гурский Н.Н. (подпись)

Минск 2022

Содержание

[ВВЕДЕНИЕ](#_Toc66303656) 4

[1.Математическая формулировка задачи 5](#_Toc66303657)

[2.Описание программы](#_Toc66303658) 6

2.1. [Структурная схема программы 6](#_Toc66303657)

2.2. [Описание разработанного класса 6](#_Toc66303657)

2.3. [Основные возможности программы 7](#_Toc66303657)

[3. Руководство пользователя](#_Toc66303659) 8

[4.Методика испытаний](#_Toc66303665) 11

[ЗАКЛЮЧЕНИЕ](#_Toc66303674) 15

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ](#_Toc66303675) 16

[ПРИЛОЖЕНИЕ](#_Toc66303676) 17

**ВВЕДЕНИЕ**

На протяжении многих веков человечество не переставало пополнять свои знания в разнообразных областях наук. Наука о пространственных фигурах – стереометрия – неотъемлемо связана со многими дисциплинами, такими как математика, информатика и программирование, физика, биология, химия. В архитектуре также используются теоремы и следствия стереометрии.

Учёных геометров и обычных людей всегда интересовала такая фигура как шар и его «оболочка» – сфера. Множество реальных объектов в астрономии, физике, биологии и других естественных науках имеют форму шара, и потому изучению его свойств отводилась значительная роль в различные исторические периоды (некоторые были уже известны геометрам Древней Греции), да и отводится теперь в наше время.

Мы каждый день встречаемся с шарами. Например, люди пользуются шариковой ручкой, где в конец стержня вмонтирован металлический шар, носят украшения и вешают ёлочные игрушки такой формы; изготавливаются шаровые опоры в автомобильной промышленности, которые являются важнейшими деталями; всевозможные элементы ракет, самолётов, мотоциклов, плавательных судов и снарядов тоже имеют какие-либо сферические поверхности – обтекатели. Кроме этого в современном мире космические корабли и спускаемые в атмосферу аппараты так же имеют форму шара, как и отсеки, входящие в них, а архитекторы включают в свои проекты различные сферические и шарообразные объекты, будь то необычные колонны или сами здания. Также задачи на сферу и шар встречаются в заданиях на едином государственном экзамене в школе. Во всём этом и заключается актуальность исследования: очень важно изучать различные свойства представленных фигур для их правильного и грамотного применения в реальной жизни.

1. **Математическая формулировка задачи**

В качестве формирования сферы используется параметрическое уравнение сферы:

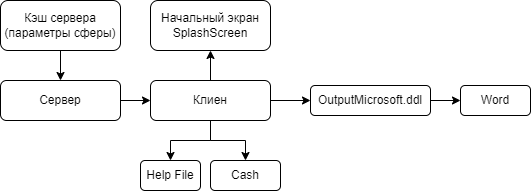
, где θ ϵ [0,π], φ ϵ [0,2π]

Использование параметрического уравнения объясняется тем, что оно имеет все параметры и значения, для построения сферы в объемной среде.

**2.** **Описание программы**

**2.1 Структурная схема программы**

Структурно программа состоит из сервера, клиента, динамических библиотек, Microsoft Office, вспомогательных форм, файлов и обращений к другим программам. Сама структура представлена на рисунке 2.1.

****

**«**Рисунок 2.1 – Структура приложения**»**

**2.2 Описание разработанного класса**

В результате анализа предметной области для формирования и отображения сферы был разработан класс SphereBuild. Класс SphereBuild содержит все требуемые для построения сферы методы и свойства (agent, slicesCounter, stacksCounter, style, color)

private int style; // Стиль отображения сферы (точечный, линейный)

private int stacksCounter; // Кол-во поперечных линий, строящие плотность сферы

private int slicesCounter; // Кол-во продольных линий, строящие плотность сферы

private float[] colorarr; // Цвет сферы

private int agent; // Радиус сферы

void readFromFile() //Чтение параметров сферы, сохраненных после последнего закрытия сферы

private void openGLControl1\_OpenGLDraw(object sender, SharpGL.RenderEventArgs args) // Динамическое слушание изменений сферы

public void draw\_sphere(OpenGL gl, double agent, int style, int stacksCounter, int slicesCounter, float[] arrColor) // Метод для формирования и отображения сферы

Как видно поля классов закрыты, доступ к ним осуществляется через соответствующие свойства и методы.

Так же организован класс ServerApi, позволяющий работать с сервером, отправляя запрос и получая необходимую информацию.

Так же разработан класс WordHelper, позволяющий структурно сохранять данные о сфере документ Word.

**2.3 Основные возможности программы**

Программа начинается с отображения заставки (содержится в SplashScreen). Далее в соответствии с сохранённым в txt файле на экране появляется главное окно программы (SphereBuild) с уже отображенной сферой, которая была сохранена после последнего закрытия программы. Основное приложение управляет работой всех других модулей в соответствии с запросами пользователя.

Он содержит:

· Изменение радиуса сферы;

· Изменение цвета сферы;

· Изменение плотности сферы;

· Изменение стиля сферы;

· Запрос параметров с сервера;

· Вызов help файла «О программе»;

· Сохранение данных в Word-файле;

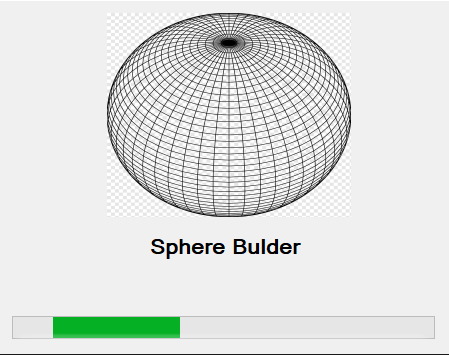
· Автоматическое кэширование данных и создание скриншота последней модели сферы, после закрытия программы.

Все возможные сценарии действий пользователя предсказываются, в соответствии с этим экстренно программа закрыта быть не может.

В случае худших сценариев работы пользователь будет получать разного рода ошибки, в зависимости от сценария.

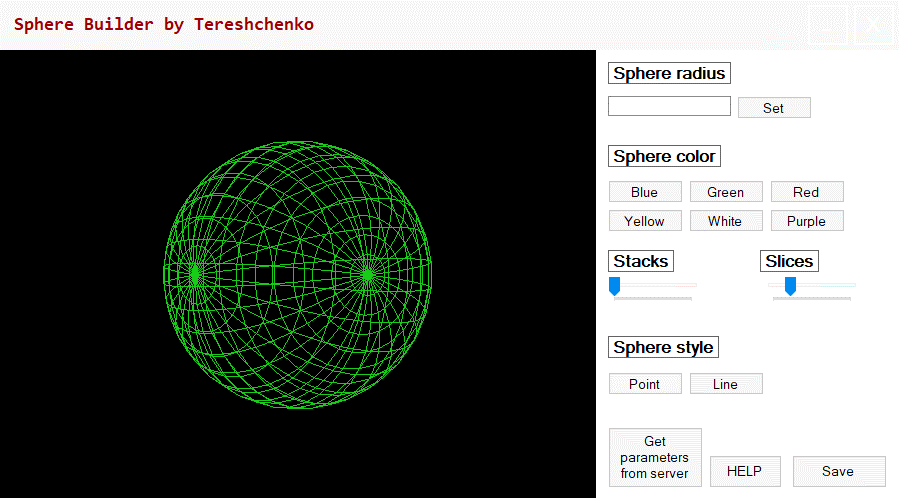
**3. Руководство пользователя**

Для открытия программы нужно вызвать файл Sphere Builder.exe. При этом, на экране отобразится заставка, представленная на рисунке 3.1.



**«**Рисунок 3.1 – Заставка приложения**»**

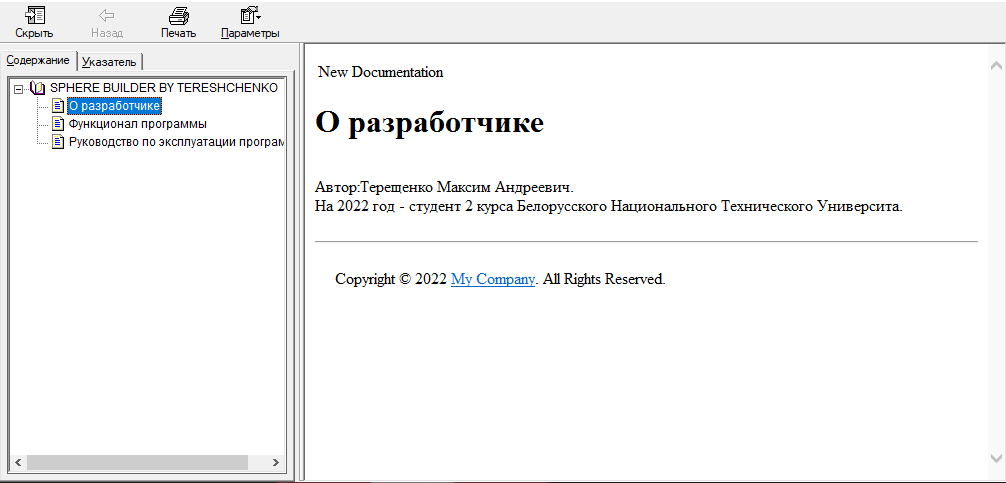
После ожидания загрузки приложения на экране появится главное окно приложения. Пример на рисунке 3.2.



**«**Рисунок 3.2 – Основное окно приложения**»**

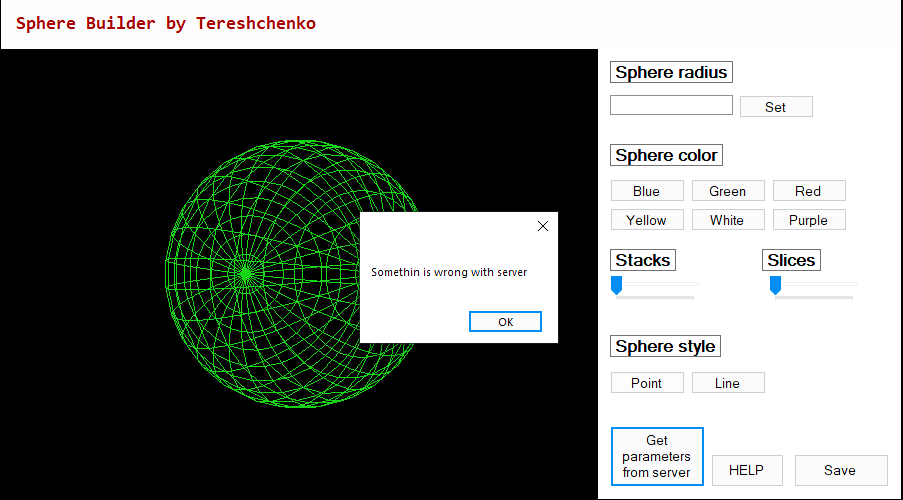
В главном окне можно изменять радиус сферы (кнопка Set), изменять цвет сферы, изменять стиль отображения сферы. Ползунки Stacks и Slices позволяют изменить плотность сферы. Так же можно получить параметры сферы с сервера (кнопка Get parameters from server), и сохранить данные в файл Word при нажатии на кнопку Save.

При нажатии на кнопку HELP появляется Helper, который выступает в качестве мануала по программе. (рисунок 3.3)



**«**Рисунок 3.3 – окно Help**»**

В случае, если пользователь допустил ошибку при вводе необходимы входных данных – ему будет предоставлено сообщение об ошибке. (рисунок 3.4)



**«**Рисунок 3.3 – пример сообщения об ошибке**»**

Так же приложение можно свернуть или закрыть, при нажатии на соответствующие кнопки.

**4. Методика испытаний**

Целью проведения испытаний является проверка работоспособности программы при различных условиях ее функционирования.

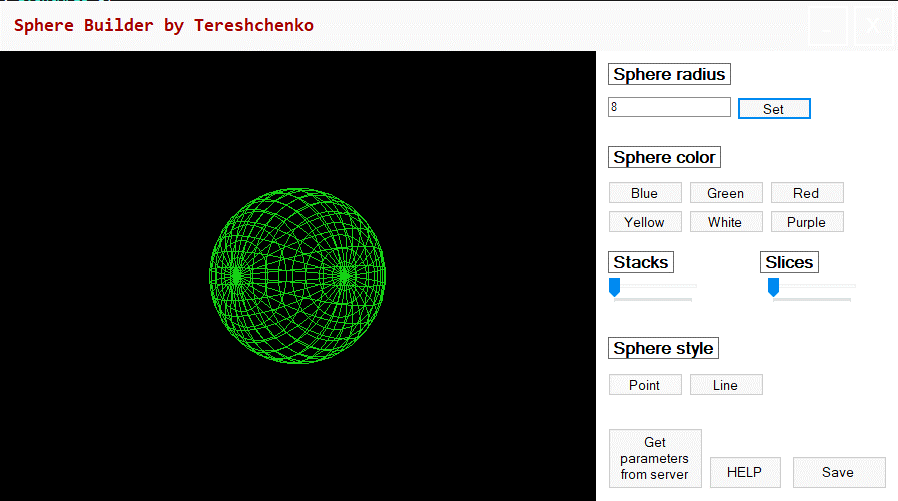
Программа должна обеспечивать корректность ввода исходных данных, а также получение непротиворечивого результата.

Для демонстрации работоспособности программы необходимо провести ряд испытаний с различными начальными условиями. Для обеспечения нормальной работы программы требуется возможность подключения к серверу, наличие соответствующих DLL, а также приложений Word.

Тестовые примеры выполнялись в среде операционной системы Windows 10 при использовании процессора Intel(R) Core(TM) i5 и разрешении экрана 1920\*1080. Для работы с программой необходимо также наличие клавиатуры и манипулятора типа «мышь».

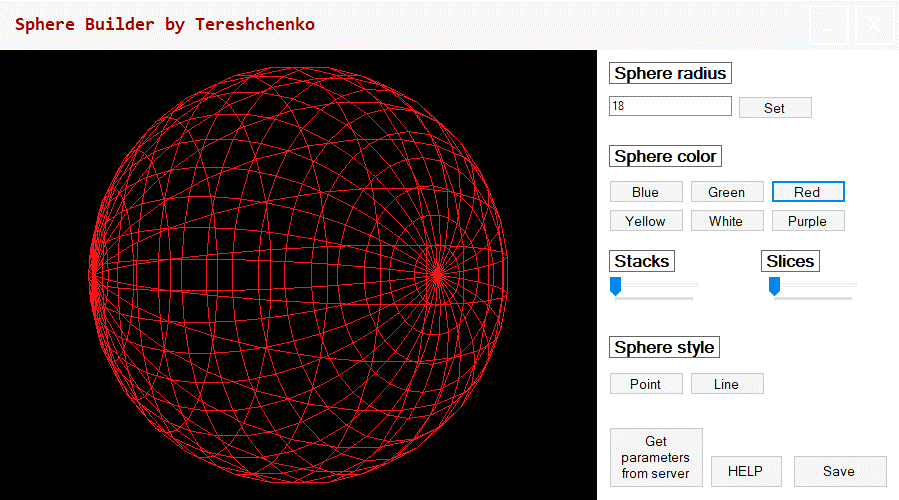
Первое испытание проведем со значениями радиуса сферы 8.

Результаты представлены на рисунке 4.1.



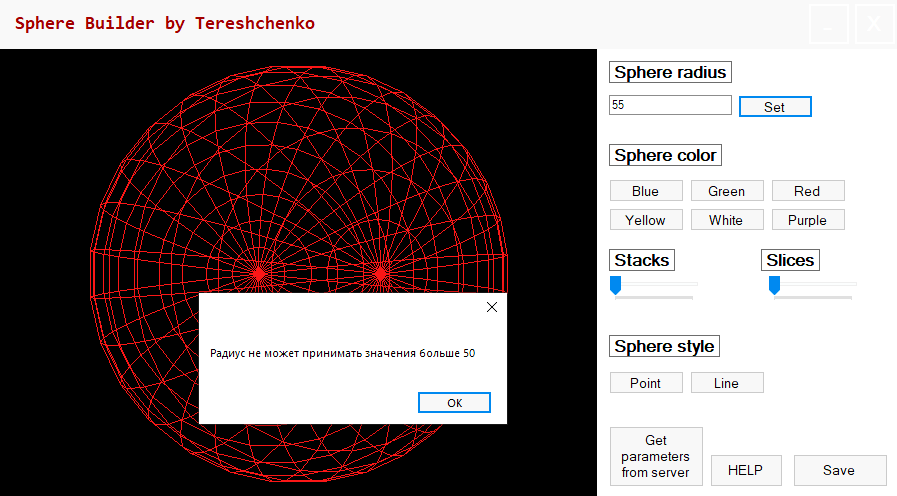
**«**Рисунок 4.1 – Начальные условия»

Мы убеждаемся что все проработало корректно. Проведем испытание с другими данными (например при радиусе равном 18). Результат выполнения программы дал результаты, представленные на рисунке 4.2.



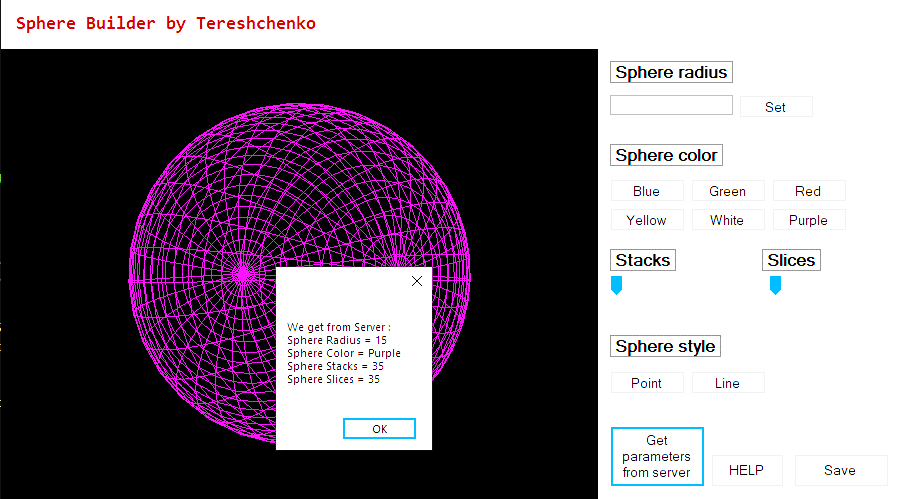
**«**Рисунок 4.2 – Результат выполнения программы»

Снова все проработало корректно, далее возьмем значение 55. Результат представлен на рисунке 4.3.В случае если сервер не был запущен, появляется соответствующее сообщение, а графики отражают информацию о той вакансии, которая была введена последняя. Проведём повторные испытания с не запущенным сервером, результат работы программы представлен на рисунке 4.3.

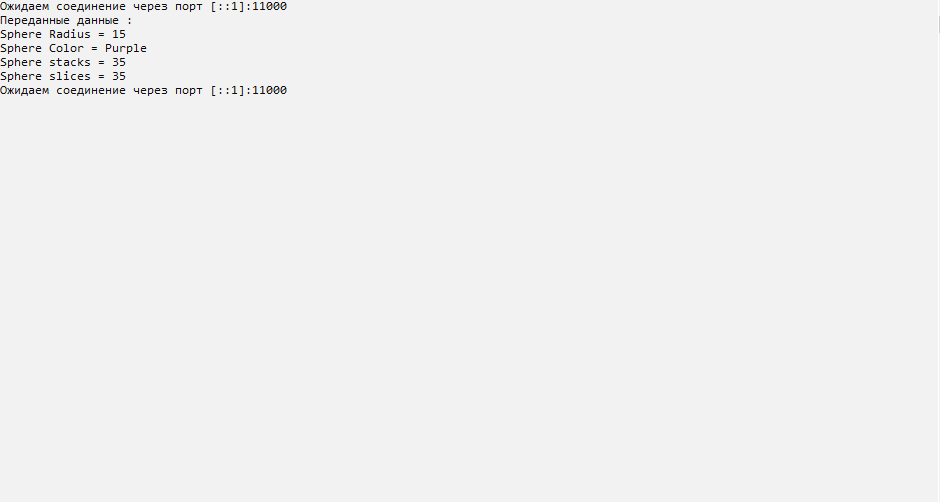


**«**Рисунок 4.3 – Результат выполнения при значении в 50»

Было выведено сообщение об ошибке «Радиус не может принимать значения больше 50». Далее протестируем взаимодействие с сервером, при получении данных с сервера. Результат будут представлены на рисунках 4.4 – 4.5.



**«**Рисунок 4.4 – Состояние клиента после запроса на сервер»



**«**Рисунок 4.5 – Состояние сервера после запроса клиента»

Все отработало корректно , и даже после ошибки неверно введенного радиуса интерфейс программы продолжил быть доступным для действий пользователя. Таким образом, проведенное тестирование программы не выявило сбойных ситуаций и некорректностей в ее работе. Следует считать, что программа протестирована и отвечает поставленным требованиям и вполне работоспособна.

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы:

1. Были закреплены знания по курсу «Разработка приложений в визуальных средах»;
2. Приобретен опыт при разработке объектно-ориентированных программ;
3. Изучены принципы создания динамических библиотек;
4. Освоены методы передачи данных между приложениями;
5. Изучены возможности создания справочной системой высокой степени сложности и различных форматов.

При разработке программы использовался модульный принцип программирования. Такая структура программы позволяет в дальнейшем модифицировать отдельные её части без нарушения работоспособности и потери функциональности приложения. Программа имеет дружелюбный пользовательский интерфейс, обеспечивающий удобство в работе и обеспечивающий получение необходимой пользователю информации.

Таким образом, разработанное в ходе выполнения курсовой работы приложение является законченным программным продуктом. Программа может быть использована для формирования и представления сфер.

**СПИСОК ИСПОЛЬЗОВАНННОЙ ЛИТЕРАТУРЫ**

1. Биллиг В.A. Основы объектного программирования на С# (C# 3.0, Visual Studio 2008) [Электронный ресурс] : учебное пособие / В.A. Биллиг. — Электрон. текстовые данные. — Москва, Саратов: Интернет-Университет Информационных Технологий (ИНТУИТ), Вузовское образование, 2017. — 583 c.

2. Есенин, Сергей DirectX и Delphi. Разработка графических и мультимедийных приложений / Сергей Есенин. - М.: БХВ-Петербург , 2015. - 512 c.

3. Windows Forms. Программирование на C# [Электронный ресурс]. — Режим доступа: http://csharpcoding.org/category/windows-forms/ (дата обращения: 15.03.2017).

**ПРИЛОЖЕНИЕ**

**Листинг исходных кодов программ**

**Файл SphereBuild.cs**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Drawing;

using System.IO;

using System.Threading;

using System.Windows.Forms;

using SharpGL;

using SharpGL.SceneGraph;

using SharpGL.SceneGraph.Effects;

using SharpGL.SceneGraph.Quadrics;

namespace SphereBuilder

{

public partial class Form1 : Form

{

public Form1()

{

Thread t = new Thread(new ThreadStart(StartForm));

t.Start();

Thread.Sleep(3000);

InitializeComponent();

readFromFile();

t.Abort();

}

public void StartForm()

{

Application.Run(new splashScreen());

}

private void button1\_Click(object sender, EventArgs e)

{

StreamWriter f = new StreamWriter("C:\\Users\\Max\\Desktop\\SphereBuilder\\result.txt");

f.WriteLine(agent);

f.WriteLine(style);

f.WriteLine(slicesCounter);

f.WriteLine(stacksCounter);

f.Close();

Bitmap printscreen = new Bitmap(Screen.PrimaryScreen.Bounds.Width, Screen.PrimaryScreen.Bounds.Height);

Graphics graphics = Graphics.FromImage(printscreen as Image);

graphics.CopyFromScreen(520, 380, 630, 350, new Size(570,430));

printscreen.Save(@"C:\Users\Max\Desktop\SphereBuilder\printscreen.jpg", System.Drawing.Imaging.ImageFormat.Jpeg);

this.Close();

}

private void button2\_Click(object sender, EventArgs e)

{

this.WindowState = FormWindowState.Minimized;

}

Point StatePoint;

private void label1\_MouseMove(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left)

{

this.Left = this.Left + e.X - StatePoint.X;

this.Top = this.Top + e.Y - StatePoint.Y;

}

}

private void label1\_MouseDown(object sender, MouseEventArgs e)

{

StatePoint = new Point(e.X, e.Y);

}

private void panel1\_MouseMove(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left)

{

this.Left = this.Left + e.X - StatePoint.X;

this.Top = this.Top + e.Y - StatePoint.Y;

}

}

private void panel1\_MouseDown(object sender, MouseEventArgs e)

{

StatePoint = new Point(e.X, e.Y);

}

private void button2\_MouseEnter(object sender, EventArgs e)

{

button2.ForeColor = Color.Black;

}

private void button2\_MouseLeave(object sender, EventArgs e)

{

button2.ForeColor = Color.White;

}

private float rot = 10.0f;

private int style = 0; // Стиль отображения сферы (точечный, линейный)

private int stacksCounter = 20; // Кол-во поперечных линий, строящие плотность сферы

private int slicesCounter = 20; // Кол-во продольных линий, строящие плтность сферы

private float[] colorarr = { 0.1f, 0.7f, 0.1f }; // Цвет сферы

private int agent = 13; // Радиус сферы

void readFromFile() //Чтение параметров сферы, сохраненных после последнего закрытия сферы

{

StreamReader r = new StreamReader("C:\\Users\\Max\\Desktop\\SphereBuilder\\result.txt");

string line = r.ReadLine();

bool success = int.TryParse(line, out agent);

line = r.ReadLine();

bool successs = int.TryParse(line, out style);

line = r.ReadLine();

bool successss = int.TryParse(line, out slicesCounter);

line = r.ReadLine();

bool successsss = int.TryParse(line, out stacksCounter);

r.Close();

}

private void openGLControl1\_Load(object sender, EventArgs e)

{

}

private void openGLControl1\_OpenGLDraw(object sender, SharpGL.RenderEventArgs args) // Динамическое слушание изменений сферы

{

OpenGL gl = openGLControl1.OpenGL;

gl.Clear(OpenGL.GL\_COLOR\_BUFFER\_BIT | OpenGL.GL\_DEPTH\_BUFFER\_BIT);

gl.LoadIdentity();

gl.Translate(0.0f, 0.0f, -50.0f);

gl.Rotate(rot, 0.0f, 0.1f, 0.0f);

draw\_sphere(gl, agent, style, stacksCounter, slicesCounter, colorarr);

gl.Flush();

rot++;

}

string gg = "Линейный";

public void draw\_sphere(OpenGL gl, double agent, int style, int stacksCounter, int slicesCounter, float[] arrColor) // Метод для формирования и отображения сферы

{

GLColor glClr = new GLColor(0.0f, 0.0f, 1.0f, 0.1f);

Sphere sphere = new SharpGL.SceneGraph.Quadrics.Sphere();

sphere.Radius = agent;

OpenGLAttributesEffect glEffect = new OpenGLAttributesEffect();

glEffect.ColorBufferAttributes.ColorModeClearColor = glClr;

glEffect.ColorBufferAttributes.ColorModeWriteMask = glClr;

gl.Color(arrColor[0], arrColor[1], arrColor[2]);

sphere.NormalGeneration = SharpGL.SceneGraph.Quadrics.Normals.Smooth;

sphere.NormalOrientation = SharpGL.SceneGraph.Quadrics.Orientation.Outside;

sphere.QuadricDrawStyle = SharpGL.SceneGraph.Quadrics.DrawStyle.Line;

if (style == 1)

{

sphere.QuadricDrawStyle = SharpGL.SceneGraph.Quadrics.DrawStyle.Point;

gg = "Точечный";

}

if (style == 2)

{

sphere.QuadricDrawStyle = SharpGL.SceneGraph.Quadrics.DrawStyle.Line;

gg = "Линейный";

}

sphere.AddEffect(glEffect);

sphere.CreateInContext(gl);

sphere.Slices = slicesCounter;

sphere.Stacks = stacksCounter;

sphere.TextureCoords = false;

sphere.PushObjectSpace(gl);

sphere.Render(gl, SharpGL.SceneGraph.Core.RenderMode.Render);

sphere.PopObjectSpace(gl);

gl.End();

}

private void buttonPoint\_Click(object sender, EventArgs e)

{

style = 1;

}

private void button4\_Click(object sender, EventArgs e)

{

style = 2;

}

private void trackBar1\_Scroll(object sender, EventArgs e)

{

stacksCounter = trackBar1.Value;

}

private void trackBar2\_Scroll(object sender, EventArgs e)

{

slicesCounter = trackBar2.Value;

}

private void button3\_Click\_1(object sender, EventArgs e)

{

colorarr[0] = 0.1f;

colorarr[1] = 0.1f;

colorarr[2] = 0.9f;

}

private void button5\_Click(object sender, EventArgs e)

{

colorarr[0] = 0.1f;

colorarr[1] = 0.9f;

colorarr[2] = 0.1f;

}

private void button6\_Click(object sender, EventArgs e)

{

colorarr[0] = 0.9f;

colorarr[1] = 0.1f;

colorarr[2] = 0.1f;

}

private void button7\_Click(object sender, EventArgs e)

{

colorarr[0] = 0.9f;

colorarr[1] = 0.9f;

colorarr[2] = 0.1f;

}

private void button8\_Click(object sender, EventArgs e)

{

colorarr[0] = 0.9f;

colorarr[1] = 0.9f;

colorarr[2] = 0.9f;

}

private void button9\_Click(object sender, EventArgs e)

{

colorarr[0] = 0.9f;

colorarr[1] = 0.1f;

colorarr[2] = 0.9f;

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

string input = textBox1.Text;

try

{

if (input != "")

{

int chislo = Convert.ToInt32(input);

}

}

catch

{

MessageBox.Show("Введено не число!", "Ошибка!");

textBox1.Text = "";

}

}

private void button10\_Click(object sender, EventArgs e)

{

int radius = 1;

if (textBox1 != null)

{

bool success = int.TryParse(textBox1.Text, out radius);

if (success && radius != 0 && radius < 51) agent = radius;

}

if(textBox1 == null || textBox1.Text == "")

{

MessageBox.Show("Field is empty!!!");

}

if(radius > 50)

{

MessageBox.Show("Радиус не может принимать значения больше 50");

textBox1.Text = "";

}

}

private void button11\_Click(object sender, EventArgs e)

{

var helper = new WordHelper("resultSphere.docx");

var items = new Dictionary<string, string>

{

{"ORG", agent.ToString() },

{"OOO", gg },

};

helper.Process(items);

MessageBox.Show("Сохранено успешно");

}

private void button12\_Click(object sender, EventArgs e)

{

ServerAPI serv = new ServerAPI();

int[] r = ServerAPI.GetMessageFromSocket(11000);

if (r[0] != 228)

{

agent = r[0];

}

else

{

MessageBox.Show("Somethin is wrong with server");

return;

}

if (r[1] == 1)

{

colorarr[0] = 0.9f;

colorarr[1] = 0.1f;

colorarr[2] = 0.9f;

}

stacksCounter = r[2];

slicesCounter = r[3];

MessageBox.Show("We get from Server : \nSphere Radius = " + r[0] + "\nSphere Color = Purple" + "\nSphere Stacks = " + r[2] + "\nSphere Slices = " + r[3]);

}

private void button13\_Click(object sender, EventArgs e)

{

Process process = Process.Start(@"C:\Users\Max\Desktop\SphereBuilder\help.chm");

int id = process.Id;

Process tempProc = Process.GetProcessById(id);

this.Visible = false;

tempProc.WaitForExit();

this.Visible = true;

}

}

}

**Class WordHelper.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Word = Microsoft.Office.Interop.Word;

namespace SphereBuilder

{

class WordHelper

{

private FileInfo \_fileInfo;

public WordHelper(string fileName)

{

if (File.Exists(fileName))

{

\_fileInfo = new FileInfo(fileName);

}

else

{

throw new ArgumentException("File not found");

}

}

internal bool Process(Dictionary<string, string> items)

{

Word.Application app = null;

try

{

app = new Word.Application();

Object file = \_fileInfo.FullName;

Object missing = Type.Missing;

app.Documents.Open(file);

foreach(var item in items)

{

Word.Find find = app.Selection.Find;

find.Text = item.Key;

find.Replacement.Text = item.Value;

Object wrap = Word.WdFindWrap.wdFindContinue;

Object replace = Word.WdReplace.wdReplaceAll;

find.Execute(FindText: Type.Missing,

MatchCase: false,

MatchWholeWord: false,

MatchWildcards: false,

MatchSoundsLike: missing,

MatchAllWordForms: false,

Forward: true,

Wrap: wrap,

Format: false,

ReplaceWith: missing, Replace: replace);

}

Object newFileName = Path.Combine(\_fileInfo.DirectoryName, DateTime.Now.ToString("yyyyMMdd HHmmss ") + \_fileInfo.Name);

app.ActiveDocument.SaveAs2(newFileName);

app.ActiveDocument.Close();

return true;

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

if(app != null)

{

app.Quit();

}

}

return false;

}

}

}

**Class ServerAPI.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading.Tasks;

namespace SphereBuilder

{

class ServerAPI

{

internal static int[] GetMessageFromSocket(int port)

{

int[] arrError = { 228 };

try

{

// Буфер для входящих данных

byte[] bytes = new byte[1024];

// Соединяемся с удаленным устройством

// Устанавливаем удаленную точку для сокета

IPHostEntry ipHost = Dns.GetHostEntry("localhost");

IPAddress ipAddr = ipHost.AddressList[0];

IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, port);

Socket sender = new Socket(ipAddr.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

// Соединяем сокет с удаленной точкой

sender.Connect(ipEndPoint);

// Получаем ответ от сервера

int bytesRec = sender.Receive(bytes);

string answer = Encoding.UTF8.GetString(bytes, 0, bytesRec);

int[] arrayResult = answer.Split(' ').Select(int.Parse).ToArray();

// Освобождаем сокет

sender.Shutdown(SocketShutdown.Both);

sender.Close();

return arrayResult;

}

catch(Exception e)

{

return arrError;

}

}

}

}

**Class Server.cs**

// SocketServer.cs

using System;

using System.Text;

using System.Net;

using System.Net.Sockets;

namespace ServerForKurs

{

class Program

{

static void Main(string[] args)

{

// Устанавливаем для сокета локальную конечную точку

IPHostEntry ipHost = Dns.GetHostEntry("localhost");

IPAddress ipAddr = ipHost.AddressList[0];

IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, 11000);

// Создаем сокет Tcp/Ip

Socket sListener = new Socket(ipAddr.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

// Назначаем сокет локальной конечной точке и слушаем входящие сокеты

try

{

sListener.Bind(ipEndPoint);

sListener.Listen(10);

// Начинаем слушать соединения

while (true)

{

Console.WriteLine("Ожидаем соединение через порт {0}", ipEndPoint);

// Программа приостанавливается, ожидая входящее соединение

Socket handler = sListener.Accept();

string data = null;

int radius = 15;

int stacks = 35;

int slices = 35;

Console.Write("Переданные данные : \n" + "Sphere Radius = " + radius + "\n" +

"Sphere Color = Purple\n" + "Sphere stacks = " + stacks + "\n" + "Sphere slices = " + slices + "\n");

// Отправляем ответ клиенту\

string reply = radius + " " + 1 + " " + stacks + " " + slices;

byte[] msg = Encoding.UTF8.GetBytes(reply);

handler.Send(msg);

handler.Shutdown(SocketShutdown.Both);

handler.Close();

}

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

finally

{

Console.ReadLine();

}

}

}

}