Министерство науки и высшего образования Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

«Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

Кафедра информационных систем

и программной инженерии

**Лабораторная работа № 6**

**по дисциплине**

**«Программирование компьютерной графики»**

Тема работы: Введение в программирование 2D графики

Выполнил:

ст. гр. ПРИ-120

М. А. Бочков

Принял:

преп. каф. ИСПИ

Жигалов И.Е.

Владимир, 2023

**Цель работы:**

Изучение принципов программирования 2D графики в C#.

**Ход работы:**

**Вариант №6**

**Задание**

1. Ознакомиться по методическим указаниям и литературе с теоретическим материалом.

2. Выполнить действия, приведенные в разделах 6.1, 6.2, 6.3, 6.4, 6.5. При разработке программы имя проекта, создаваемого в MS Visual Studio, должно содержать фамилию студента и группу (например, Ivanov\_Ivan\_ISG\_105\_lab\_1).

3. Добавить в программу кисть (возможно большого размера), отрисовывающую на плоскости ваши инициалы и номер группы. Добавить кнопку на панель программы для использования данной кисти. Добавить пункт меню для активации данной кисти. Разработанная программа должна соответствовать разделу 6.5.

**Задание**

Листинг Form1.cs:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Tao.FreeGlut;

using Tao.OpenGl;

namespace Bochkov\_Mikhail\_PRI\_120\_lab\_6

{

public partial class Form1 : Form

{

private anEngine ProgrammDrawingEngine;

public Form1()

{

InitializeComponent();

AnT.InitializeContexts();

}

// текущий активный слой

private int ActiveLayer = 0;

// счетчик слоев

private int LayersCount = 1;

// счетчик всех создаваемых слоев для генерации имен

private int AllLayrsCount = 1;

private void Form1\_Load(object sender, EventArgs e)

{

// добавлние элемента, отвечающего за управления главным слоем в объект LayersControl

LayersControl.Items.Add("Главный слой", true);

// инициализация библиотеки GLUT

Glut.glutInit();

// инициализация режима окна

Glut.glutInitDisplayMode(Glut.GLUT\_RGB | Glut.GLUT\_DOUBLE | Glut.GLUT\_DEPTH);

// устанавливаем цвет очистки окна

Gl.glClearColor(255, 255, 255, 1);

// устанавливаем порт вывода, основываясь на размерах элемента управления AnT

Gl.glViewport(0, 0, AnT.Width, AnT.Height);

// устанавливаем проекционную матрицу

Gl.glMatrixMode(Gl.GL\_PROJECTION);

// очищаем ее

Gl.glLoadIdentity();

Glu.gluOrtho2D(0.0, AnT.Width, 0.0, AnT.Height);

// переходим к объектно-видовой матрице

Gl.glMatrixMode(Gl.GL\_MODELVIEW);

ProgrammDrawingEngine = new anEngine(AnT.Width, AnT.Height, AnT.Width, AnT.Height);

RenderTimer.Start();

}

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

{

}

private void RenderTimer\_Tick(object sender, EventArgs e)

{

// вызываем функцию рисования

Drawing();

}

private void Drawing()

{

// очистка буфера цвета и буфера глубины

Gl.glClear(Gl.GL\_COLOR\_BUFFER\_BIT | Gl.GL\_DEPTH\_BUFFER\_BIT);

// очищение текущей матрицы

Gl.glLoadIdentity();

// установка черного цвета

Gl.glColor3f(0, 0, 0);

// визуализация изображения из движка

ProgrammDrawingEngine.SwapImage();

// дожидаемся завершения визуализации кадра

Gl.glFlush();

// сигнал для обновления элемента, реализующего визуализацию

AnT.Invalidate();

}

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

{

// если нажата левая клавиша мыши

if (e.Button == MouseButtons.Left)

ProgrammDrawingEngine.Drawing(e.X, AnT.Height - e.Y);

}

private void toolStripButton1\_Click(object sender, EventArgs e)

{

// устанавливаем стандартную кисть 4х4

ProgrammDrawingEngine.SetStandartBrush(4);

}

private void toolStripButton2\_Click(object sender, EventArgs e)

{

// устанавливаем специальную кисть

ProgrammDrawingEngine.SetSpecialBrush(0);

}

private void toolStripButton3\_Click(object sender, EventArgs e)

{

// установить кисть из файла

ProgrammDrawingEngine.SetBrushFromFile("brush-1.bmp");

}

private void linkLabel1\_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)

{

// временное хранение цвета элемента color1

Color tmp = color1.BackColor;

// замена:

color1.BackColor = color2.BackColor;

color2.BackColor = tmp;

// предача нового цвета в ядро растрового редактора

ProgrammDrawingEngine.SetColor(color1.BackColor);

}

private void color1\_MouseClick(object sender, MouseEventArgs e)

{

// если цвет успешно выбран

if (changeColor.ShowDialog() == DialogResult.OK)

{

// установить данный цвет

color1.BackColor = changeColor.Color;

// и передать его в класс anEngine для установки активным цветом текущего слоя

ProgrammDrawingEngine.SetColor(color1.BackColor);

}

}

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

{

// счетчик созданных слоев

LayersCount++;

// вызываем функцию добавления слоя в движке графического редактора

ProgrammDrawingEngine.AddLayer();

// добавляем слой, генерируем имя "Слой №" в объекте LayersControl

// обязательно после функции ProgrammDrawingEngine.AddLayer();,

// иначе произойдет попытка установки активного цвета для еще не существующего цвета

int AddingLayerNom = LayersControl.Items.Add("Слой" + LayersCount.ToString(), false);

// выделяем его

LayersControl.SelectedIndex = AddingLayerNom;

// устанавливаем его как активный

ActiveLayer = AddingLayerNom;

}

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

{

// запрашиваем подтверждение действия с помощью messageBox

DialogResult res = MessageBox.Show("Будет удален текущий активный слой, действительно продолжить?", "Внимание!", MessageBoxButtons.YesNo, MessageBoxIcon.Warning); // если пользователь нажал кнопку "ДА" в окне подтверждения

if (res == DialogResult.Yes)

{

// если удаляемый слой - начальный

if (ActiveLayer == 0)

{

// сообщаем о невозможности удаления

MessageBox.Show("Вы не можете удалить нулевой слой.", "Внимание!", MessageBoxButtons.OK, MessageBoxIcon.Stop);

}

else // иначе

{

// уменьшаем значение счетчика слоев

LayersCount--;

// сохраняем номер удаляемого слоя, т.к. SelectedIndex измениться после операций в LayersControl

int LayerNomForDel = LayersControl.SelectedIndex;

// удаляем запись в элементе LayersControl (с индексом LayersControl.SelectedIndex - текущим выделенным слоем)

LayersControl.Items.RemoveAt(LayerNomForDel);

// устанавливаем выделенный слоем - нулевой (главный слой)

LayersControl.SelectedIndex = 0;

// помечаем активный слой - нулевой

ActiveLayer = 0;

// помечаем галочкой нулевой слой

LayersControl.SetItemCheckState(0, CheckState.Checked);

// вызываем функцию удаления слоя в движке программы

ProgrammDrawingEngine.RemoveLayer(LayerNomForDel);

}

}

}

private void LayersControl\_SelectedValueChanged(object sender, EventArgs e)

{

// если отметили новый слой, необходимо снять галочку выделения со старого

if (LayersControl.SelectedIndex != ActiveLayer)

{

// если выделенный индекс является корректным (больше либо равен нулю и входит в диапазон элементов)

if (LayersControl.SelectedIndex != -1 && ActiveLayer < LayersControl.Items.Count)

{

// снимаем галочку с предыдущего активного слоя

LayersControl.SetItemCheckState(ActiveLayer, CheckState.Unchecked);

// сохраняем новый индекс выделенного элемента

ActiveLayer = LayersControl.SelectedIndex;

// помечаем галочкой новый активный слой

LayersControl.SetItemCheckState(LayersControl.SelectedIndex, CheckState.Checked);

// посылаем движку программы сигнал об изменении активного слоя

ProgrammDrawingEngine.SetActiveLayerNom(ActiveLayer);

}

}

}

private void toolStripButton4\_Click(object sender, EventArgs e)

{

добавитьСлойToolStripMenuItem\_Click(sender, e);

}

private void toolStripButton5\_Click(object sender, EventArgs e)

{

удалитьСлойToolStripMenuItem\_Click(sender, e);

}

private void toolStripButton6\_Click(object sender, EventArgs e)

{

// установка кисти-ластика

ProgrammDrawingEngine.SetSpecialBrush(1);

}

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

{

// вызываем уже существующую функцию

toolStripButton1\_Click(sender, e);

}

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

{

// вызываем уже существующую функцию

toolStripButton2\_Click(sender, e);

}

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

{

// вызываем уже существующую функцию

toolStripButton6\_Click(sender, e);

}

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

{

// вызываем диалог подтверждения

DialogResult reslt = MessageBox.Show("В данный момент проект уже начат, сохранить изменения перед закрытием проекта?", "Внимание!", MessageBoxButtons.YesNoCancel);

// в зависимости от результата нажатия кнопки пользователем в окне MessageBox

switch (reslt)

{

// если отказ пользователя

case DialogResult.No:

{

// просто создаем чистый проект

ProgrammDrawingEngine = new anEngine(AnT.Width, AnT.Height, AnT.Width, AnT.Height);

// очищаем информацию о добавленных ранее слоях

LayersControl.Items.Clear();

// вновь инициализируем нулевой слой:

// текущий активный слой

ActiveLayer = 0;

// счетчик слоев

LayersCount = 1;

// счетчик всех создаваемых слоев для генерации имен

AllLayrsCount = 1;

// добавление элемента, отвечающего за управление главным слоем, в объект LayersControl

LayersControl.Items.Add("Главный слой", true);

break;

}

case DialogResult.Cancel:

{

// возвращаемся

return;

}

case DialogResult.Yes:

{

// открываем окно сохранения файла и, если имя файла указано и DialogResult вернуло сигнал об успешном нажатии кнопки ОК

if (saveFileDialog1.ShowDialog() == DialogResult.OK)

{

// получаем результирующее изображение слоя

Bitmap ToSave = ProgrammDrawingEngine.GetFinalImage();

// сохраняем используя имя файла указанное в диалоговом окне сохранения файла

ToSave.Save(saveFileDialog1.FileName, System.Drawing.Imaging.ImageFormat.Jpeg);

// сохранили - начинаем новый проект:

// создаем новый объект "движка" программы

ProgrammDrawingEngine = new anEngine(AnT.Width, AnT.Height, AnT.Width, AnT.Height);

// очищаем информацию о добавляемых ранее слоях

LayersControl.Items.Clear();

// вновь инициализируем нулевой слой:

// текущий активный слой

ActiveLayer = 0;

// счетчик слоев

LayersCount = 1;

// счетчик всех создаваемых слоев для генерации имен

AllLayrsCount = 1;

// добавление элемента, отвечающего за управления главным слоем, в объект LayersControl

LayersControl.Items.Add("Главный слой", true);

}

else

{

// если сохранение не заврешилось нормально (скорее всего пользователь закрыл окно сохранения файла)

// возвращаемся в проект

return;

}

break;

}

}

}

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

{

}

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

{

// открываем окно сохранения файла и, если имя файла указано и DialogResult вернуло сигнал об успешном нажатии кнопки ОК

if (saveFileDialog1.ShowDialog() == DialogResult.OK)

{

// получаем результирующее изображение слоя

Bitmap ToSave = ProgrammDrawingEngine.GetFinalImage();

// сохраняем, используя имя файла, указанное в диалоговом окне сохранения файла

ToSave.Save(saveFileDialog1.FileName, System.Drawing.Imaging.ImageFormat.Jpeg);

}

}

private void выходToolStripMenuItem1\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void toolStripButton7\_Click(object sender, EventArgs e)

{

ProgrammDrawingEngine.SetStandartBrush(8);

}

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

{

toolStripButton7\_Click(sender, e);

}

}

}

Листинг anEngine.cs:

using System.Collections;

using System.Drawing;

namespace Bochkov\_Mikhail\_PRI\_120\_lab\_6

{

internal class anEngine

{

private Color LastColorInUse;

private int picture\_size\_x, picture\_size\_y;

// положение полос прокрутки будет использовано в будующем

private int scroll\_x, scroll\_y;

// размер оконной части (объекта AnT)

private int screen\_width, screen\_height;

// номер активного слоя

private int ActiveLayerNom;

// массив слоев

private ArrayList Layers = new ArrayList();

// стандартная кисть

private anBrush standartBrush;

// конструктор класса

public anEngine(int size\_x, int size\_y, int screen\_w, int screen\_h)

{

standartBrush = new anBrush(3, false);

// при инициализации экземпляра класса сохраним настройки

// размеров элементов и изображения в локальных переменных

picture\_size\_x = size\_x;

picture\_size\_y = size\_y;

screen\_width = screen\_w;

screen\_height = screen\_h;

// полосы прокрутки у нас пока отсутствуют, поэтому просто обнулим значение переменных

scroll\_x = 0;

scroll\_y = 0;

// добавим новый слой для работы, пока что он будет единственным

Layers.Add(new anLayer(picture\_size\_x, picture\_size\_y));

// номер активного слоя - 0

ActiveLayerNom = 0;

// и создадим стандартную кисть

standartBrush = new anBrush(1, false);

}

// функция для установки номера активного слоя

public void SetActiveLayerNom(int nom)

{

// текущий слой больше не будет активным, следовательно надо создать новый дисплейный список для его быстрой визуализации

((anLayer)Layers[ActiveLayerNom]).CreateNewList(); // новый активный слой получает установленный активный цвет для предыдущего активного слоя

((anLayer)Layers[nom]).SetColor(((anLayer)Layers[ActiveLayerNom]).GetColor());

// установка номера активного слоя

ActiveLayerNom = nom;

}

// установка видимости / невидимости слоя

public void SetWisibilityLayerNom(int nom, bool visible)

{

// вернемся к этой функции позже

}

// рисование текущей кистью

public void Drawing(int x, int y)

{

// транслируем координаты, в которых проходит рисование, стандартной кистью

((anLayer)Layers[ActiveLayerNom]).Draw(standartBrush, x, y);

}

// визуализация

public void SwapImage()

{

// вызываем функцию визуализации в нашем слое

for (int ax = 0; ax < Layers.Count; ax++)

{

// если этот слой является активным в данный момент

if (ax == ActiveLayerNom)

{

// вызываем визуализацию данного слоя напрямую

((anLayer)Layers[ax]).RenderImage(false);

}

else

{

// вызываем визуализацию слоя из дисплейного списка

((anLayer)Layers[ax]).RenderImage(true);

}

}

}

// функция установки стандартной кисти, предается только размер

public void SetStandartBrush(int SizeB)

{

standartBrush = new anBrush(SizeB, false);

}

// функция установки специальной кисти

public void SetSpecialBrush(int Nom)

{

standartBrush = new anBrush(Nom, true);

}

// установка кисти из файла

public void SetBrushFromFile(string FileName)

{

standartBrush = new anBrush(FileName);

}

public void SetColor(Color NewColor)

{

((anLayer)Layers[ActiveLayerNom]).SetColor(NewColor);

LastColorInUse = NewColor;

}

// функция добавления слоя

public void AddLayer()

{

// добавляем слой в массив слоев ArrayList

int AddingLayer = Layers.Add(new anLayer(picture\_size\_x, picture\_size\_y));

// устанавливаем его активным

SetActiveLayerNom(AddingLayer);

}

// функция удаления слоев

public void RemoveLayer(int nom)

{

// если номер корректен (в диапазоне добавленных в ArrayList)

if (nom < Layers.Count && nom >= 0)

{

// удаляем запись о слое

Layers.RemoveAt(nom);

// делаем активным слой 0

SetActiveLayerNom(0);

}

}

// получение финального изображения

public Bitmap GetFinalImage()

{

// заготовка результирующего изображения

Bitmap resaultBitmap = new Bitmap(picture\_size\_x, picture\_size\_y); // данное решение также не является оптимальным по быстродействию, но при этом является самым простым способом решения задачи

for (int ax = 0; ax < Layers.Count; ax++)

{

// получаем массив пикселей данного слоя

int[,,] tmp\_layer\_data = ((anLayer)Layers[ax]).GetDrawingPlace();

// пройдем двумя циклами по информации о пикселях данного слоя

for (int a = 0; a < picture\_size\_x; a++)

{

for (int b = 0; b < picture\_size\_y; b++)

{

// если пиксель не помечен как "прозрачный"

if (tmp\_layer\_data[a, b, 3] != 1)

{

// устанавливаем данный пиксель на результирующее изображение

resaultBitmap.SetPixel(a, b, Color.FromArgb(tmp\_layer\_data[a, b, 0], tmp\_layer\_data[a, b, 1], tmp\_layer\_data[a, b, 2]));

}

else

{

if (ax == 0) // нулевой слой - необходимо закрасить белым остутствующие пиксели

{

// закрашиваем белым цветом

resaultBitmap.SetPixel(a, b, Color.FromArgb(255, 255, 255));

}

}

}

}

}

// поворачиваем изображение для корректного отображения

resaultBitmap.RotateFlip(RotateFlipType.Rotate180FlipX);

// возвращаем результат

return resaultBitmap;

}

// получение изображения для главного слоя

public void SetImageToMainLayer(Bitmap layer)

{

// поворачиваем изображение (чтобы оно корректно отображалось в области редактирования)

layer.RotateFlip(RotateFlipType.Rotate180FlipX);

// проходим двумя циклами по всем пикселям изображения, подгруженного в класс Bitmap

// получая цвет пикселя, устанавливаем его в текущий слой с помощью функции Drawing

// данный алгоритм является медленным, но простым

// оптимальным решением здесь было бы написание собственного загрузчика файлов изображений,

// что дало бы возможность без "посредников" получать массив значений пикселей изображений,

// но данная задача является намного более сложной и выходит за рамки обучающей программы

for (int ax = 0; ax < layer.Width; ax++)

{

for (int bx = 0; bx < layer.Height; bx++)

{

// получения цвета пикселя изображения

SetColor(layer.GetPixel(ax, bx));

// отрисовка данного пикселя в слое

Drawing(ax, bx);

}

}

}

}

}

Листинг anLayer.cs:

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Tao.OpenGl;

using static System.Windows.Forms.VisualStyles.VisualStyleElement.StartPanel;

namespace Bochkov\_Mikhail\_PRI\_120\_lab\_6

{

internal class anLayer

{

// размеры экранной области

public int Width, Heigth;

// массив , представляющий область рисунка (координаты пикселя и его цвет)

private int[,,] DrawPlace;

// флаг видимости слоя: true - видимый, false - невидимый

private bool isVisible;

// текущий установленный цвет

private Color ActiveColor;

private int ListNom;

// функция получения массива пикселей

public int[,,] GetDrawingPlace()

{

return DrawPlace;

}

// конструктор класса, в качестве входных параметров

// мы получаем размеры изображения, чтобы создать в памяти массив,

// который будет хранить растровые данные для этого слоя

public anLayer(int s\_W, int s\_H)

{

// запоминаем значения размеров рисунка

Width = s\_W;

Heigth = s\_H;

// создаем в памяти массив, соотвествующий размерам рисунка

// каждая точка на полскости массива будет иметь 3 составляющие цвета

// + 4 ячейка - индикатор того, что данный пиксель пуст (или полность прозрачен)

DrawPlace = new int[Width, Heigth, 4];

// проходим по всей плоскости и устанавливаем всем точкам

// индикатор прозрачности

for (int ax = 0; ax < Width; ax++)

{

for (int bx = 0; bx < Heigth; bx++)

{

// флаг прозачности точки в координатах ax,bx.

DrawPlace[ax, bx, 3] = 1;

}

}

// устанавливаем флаг видимости слоя (по умолчанию создаваемый слой всегда видимый)

isVisible = true;

// текущим активным цветом устанавливаем черный

// в следующей главе мы реализуем функции установки цветов из оболочки

ActiveColor = Color.Black;

}

// функция установки режима видимости слоя

public void SetVisibility(bool visiblityState)

{

isVisible = visiblityState;

}

// функция получения текущего состояния видимости слоя

public bool GetVisibility()

{

return isVisible;

}

// функция рисования

// получает в качестве параметров кисть для рисования и координаты,

// где сейчас необходимо перерисовать пиксели заданной кистью

public void Draw(anBrush BR, int x, int y)

{

// определяем начальную позицию рисования

int real\_pos\_draw\_start\_x = x - BR.myBrush.Width / 2;

int real\_pos\_draw\_start\_y = y - BR.myBrush.Height / 2;

// корректируем ее для невыхода за границы массива

// проверка на отрицательные значения (граница "слова")

if (real\_pos\_draw\_start\_x < 0)

real\_pos\_draw\_start\_x = 0;

if (real\_pos\_draw\_start\_y < 0)

real\_pos\_draw\_start\_y = 0;

// проверки на выход за границу "справа"

int boundary\_x = real\_pos\_draw\_start\_x + BR.myBrush.Width;

int boundary\_y = real\_pos\_draw\_start\_y + BR.myBrush.Height;

if (boundary\_x > Width)

boundary\_x = Width;

if (boundary\_y > Heigth)

boundary\_y = Heigth;

// счетчик пройденных строк и столбцов массива, прдстваляющего собой маску кисти

int count\_x = 0, count\_y = 0;

// цикл по области с учетом смещения кисти и коорекции для невыхода за границы массива

for (int ax = real\_pos\_draw\_start\_x; ax < boundary\_x; ax++, count\_x++)

{

count\_y = 0;

for (int bx = real\_pos\_draw\_start\_y; bx < boundary\_y; bx++, count\_y++)

{

// проверяем, не является ли данная кисть ластиком

if (BR.IsBrushErase())

{

// данная кисть - ластик

// помечаем данный пиксель как незакрашенный

// получаем текущий цвет пикселя маски

Color ret = BR.myBrush.GetPixel(count\_x, count\_y);

// цвет не красный

if (!(ret.R == 255 && ret.G == 0 && ret.B == 0))

{

// заполняем данный пиксель активным цветом из маски

DrawPlace[ax, bx, 3] = 1;

}

}

else

{

// получаем текущий цвет пикселя маски

Color ret = BR.myBrush.GetPixel(count\_x, count\_y);

// цвет не красный

if (!(ret.R == 255 && ret.G == 0 && ret.B == 0))

{

// заполняем данный пиксель активным цветом из маски

DrawPlace[ax, bx, 0] = ActiveColor.R;

DrawPlace[ax, bx, 1] = ActiveColor.G;

DrawPlace[ax, bx, 2] = ActiveColor.B;

DrawPlace[ax, bx, 3] = 0;

}

}

}

}

}

// функция визуализации слоя

public void RenderImage(bool FromList)

{

if (FromList) // указана визуализация из дисплейного списка, следовательно данный слой не активен

{

// вызываем дисплейный список

Gl.glCallList(ListNom);

}

else // данный слой активен и визуализацию необходимо делать на ходу

{

// счетчик номеров элементов, которые должны участвовать в визуализации

int count = 0;

// проходим по всем точкам рисунка

for (int ax = 0; ax < Width; ax++)

{

for (int bx = 0; bx < Heigth; bx++)

{

// если точка в координатах ax,bx не помечена флагом "прозрачная"

if (DrawPlace[ax, bx, 3] != 1)

{

// не лучшие способ, но так мы подсчитаем количество действительно значимых точек слоя,

// которые должны быть визуализированны

count++;

}

}

}

// данный массив будет заполнен, а затем передан для быстрой отрисовки геометрии (в нашем случае - точек)

// колич. точек \* 2 (для хранения координат x и y каждой точки, которая будет отрисована)

int[] arr\_date\_vertex = new int[count \* 2];

// данный массив будет содержать значения цветов для всех отрисовываемых точек

// колич. точек \* 3 (для хранения координат R G B значений цветов каждой точки, которая будет отрисована)

float[] arr\_date\_colors = new float[count \* 3];

// счетчик элементов для создания массивов, которые будут переданы в реализацию OpenGL c

// помощью функции glDrawArrays

int now\_element = 0;

// теперь, когда мы выделили массив необходимого размера,

// заполним его необходимыми значениями

for (int ax = 0; ax < Width; ax++)

{

for (int bx = 0; bx < Heigth; bx++)

{

// если точка с координатами ax,bx не помечена флагом "прозрачная"

// если данная точка НЕ помечена флагом, сигнализирующим о том, что она не должна быть визуализированна

if (DrawPlace[ax, bx, 3] != 1)

{

// заносим координаты точки (ax , bx ) в массив, который будет передан для визуализации

arr\_date\_vertex[now\_element \* 2] = ax;

arr\_date\_vertex[now\_element \* 2 + 1] = bx;

// заносим значения составляющих цвета, сразу перенося их в формат float

arr\_date\_colors[now\_element \* 3] = (float)DrawPlace[ax, bx, 0] / 255.0f;

arr\_date\_colors[now\_element \* 3 + 1] = (float)DrawPlace[ax, bx, 1] / 255.0f;

arr\_date\_colors[now\_element \* 3 + 2] = (float)DrawPlace[ax, bx, 2] / 255.0f;

// подсчет добавленных элементов в массивы

now\_element++;

}

}

}

// теперь, когда массивы с геометрическими данными и данными о цветах подготовлены,

// включаем функцию использования массивов вершин и цветов

Gl.glEnableClientState(Gl.GL\_VERTEX\_ARRAY);

Gl.glEnableClientState(Gl.GL\_COLOR\_ARRAY);

// передаем массивы вершин и цветов, указывая количество элементов массива, приходящихся

// на один визуализируемый элемент (в случае точек - 2 координаты: х и у, в случае цветов - 3 составляющие цвета)

Gl.glColorPointer(3, Gl.GL\_FLOAT, 0, arr\_date\_colors);

Gl.glVertexPointer(2, Gl.GL\_INT, 0, arr\_date\_vertex);

// вызываем функцию glDrawArrays, которая позволит нам визуализировать наши массивы, передав их целиком,

// а не передавая в цикле каждую точку

Gl.glDrawArrays(Gl.GL\_POINTS, 0, count);

// деактивируем режим использования массивов геометрии и цветов

Gl.glDisableClientState(Gl.GL\_VERTEX\_ARRAY);

Gl.glDisableClientState(Gl.GL\_COLOR\_ARRAY);

}

}

// установка текущего цвета для рисования в слое

public void SetColor(Color NewColor)

{

ActiveColor = NewColor;

}

// получение текущего активного цвета

public Color GetColor()

{

// возвращаем цвет

return ActiveColor;

}

public void ClearList()

{

// проверяем факт существования дисплейного списка с номером, хранимым в ListNom

if (Gl.glIsList(ListNom) == Gl.GL\_TRUE)

{

// удаляем его в случае существования

Gl.glDeleteLists(ListNom, 1);

}

}

public void CreateNewList()

{

// проверяем факт существования дисплейного списка с номером, хранимым в ListNom

if (Gl.glIsList(ListNom) == Gl.GL\_TRUE)

{

// удаляем его в случае существования

Gl.glDeleteLists(ListNom, 1);

// и генерируем новый номер

ListNom = Gl.glGenLists(1);

}

// создаем дисплейный список

Gl.glNewList(ListNom, Gl.GL\_COMPILE);

// вызывая обычную визуализацию (не из списка)

RenderImage(false);

// заверщаем создание дисплейного списка

Gl.glEndList();

}

}

}

Листинг anBrush.cs:

using System;

using System.Collections.Generic;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Bochkov\_Mikhail\_PRI\_120\_lab\_6

{

internal class anBrush

{

public Bitmap myBrush;

private bool IsErase = false;

// стандартная (квадратная) кисть, с указанием масштаба

// и флагом закраски узлов

public anBrush(int Value, bool Special)

{

if (!Special)

{

myBrush = new Bitmap(Value, Value);

for (int ax = 0; ax < Value; ax++)

for (int bx = 0; bx < Value; bx++)

myBrush.SetPixel(0, 0, Color.Black);

// не является ластиком

IsErase = false;

}

else

{

// здесь мы будем размещать предустановленные кисти

// созданная нами ранее кисть в виде перекрестия двух линий будет китсью по умолчанию

// на тот случай, если задан не описанный номер кисти

switch (Value)

{

// специальная кисть по умолчанию

default:

{

myBrush = new Bitmap(5, 5);

for (int ax = 0; ax < 5; ax++)

for (int bx = 0; bx < 5; bx++)

myBrush.SetPixel(ax, bx, Color.Red);

myBrush.SetPixel(0, 2, Color.Black);

myBrush.SetPixel(1, 2, Color.Black);

myBrush.SetPixel(2, 0, Color.Black);

myBrush.SetPixel(2, 1, Color.Black);

myBrush.SetPixel(2, 2, Color.Black);

myBrush.SetPixel(2, 3, Color.Black);

myBrush.SetPixel(2, 4, Color.Black);

myBrush.SetPixel(3, 2, Color.Black);

myBrush.SetPixel(4, 2, Color.Black);

// не является ластиком

IsErase = false;

break;

}

case 1: // стерка

{

// создается так же, как и обычная кисть,

// но имеет флаг IsErase равный true

myBrush = new Bitmap(5, 5);

for (int ax = 0; ax < Value; ax++)

for (int bx = 0; bx < Value; bx++)

myBrush.SetPixel(0, 0, Color.Black);

// является ластиком

IsErase = true;

break;

}

}

}

}

// второй конструктор будет позволять загружать кисть из стороннего файла

public anBrush(string FromFile)

{

string path = Directory.GetCurrentDirectory();

path += "\\" + FromFile;

myBrush = new Bitmap(path);

}

public bool IsBrushErase()

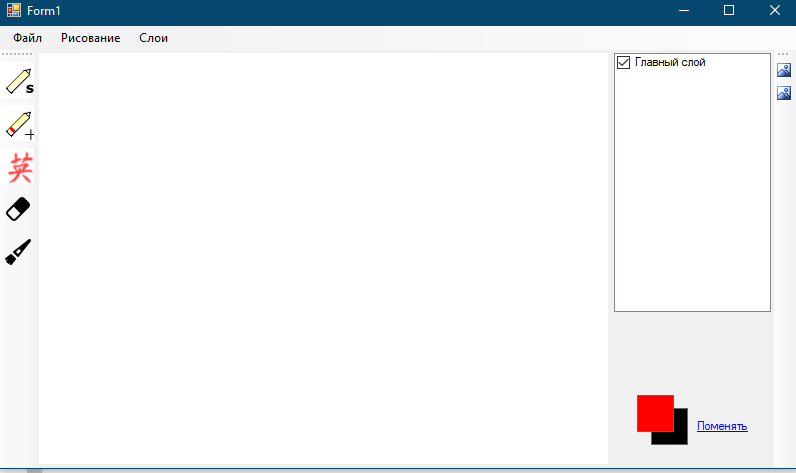
{

return IsErase;

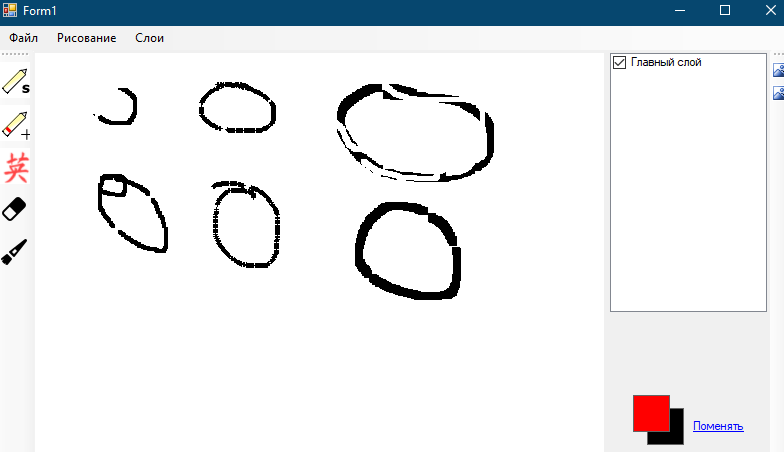
}

}

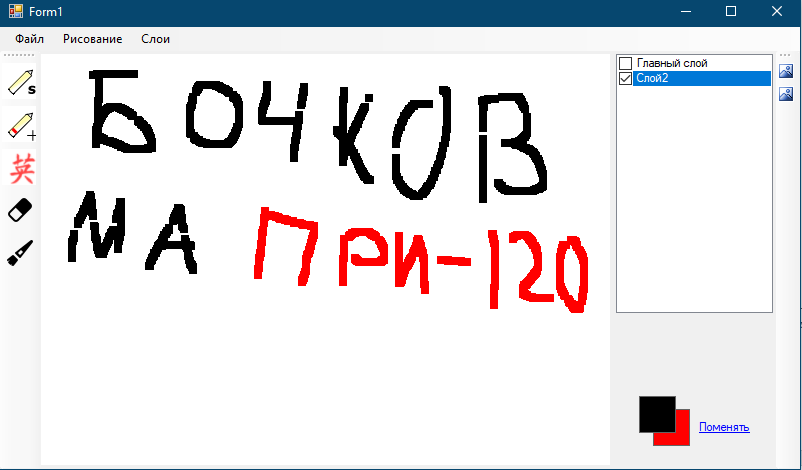
}



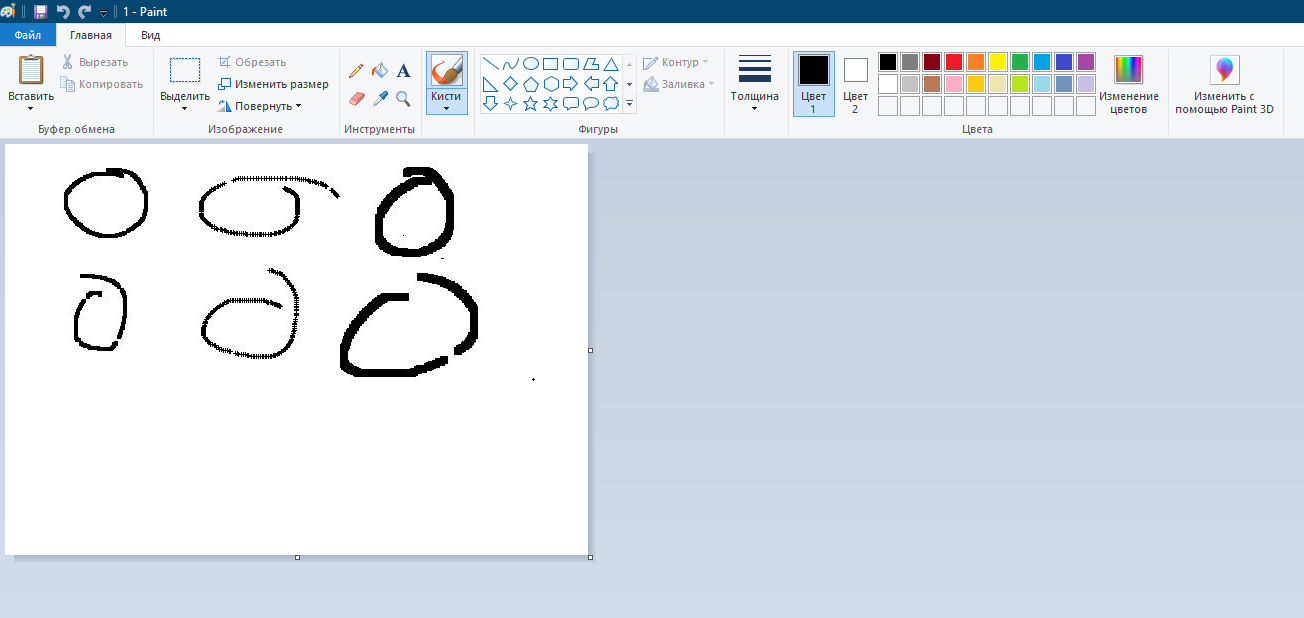
Скриншот 1- Стартовое окно



Скриншот 2- Использование разных инструментов



Скриншот 3 - Требуемая надпись на новом слое



Скриншот 4- сохраненный файл

**Вывод**

В процессе выполнения работы были изучены принципы программирования 2D-графики в C#