МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение
высшего образования
«Курганский государственный университет»

Кафедра программного обеспечения автоматизированных систем

КОМПЬЮТЕРНАЯ ГРАФИКА

Методические указания
к выполнению практических работ
для студентов направления подготовки 09.03.04
«Программная инженерия»

Курган 2018

Кафедра: «Программное обеспечение автоматизированных систем».

Дисциплина: «Компьютерная графика» (направление 09.03.04 «Программная инженерия»).

Составил: канд. техн. наук, доцент А.М. Семахин.

Утверждены на заседании кафедры «28» августа 2017 г.

Рекомендованы методическим советом университета

«12» декабря 2016 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Дисциплина «Компьютерная графика» имеет целью дать студентам теоретические знания и практические навыки в разработке программных приложений при работе с графическими изображениями.

Предмет дисциплины – технологии создания графических изображений с помощью компьютера.

Задачами освоения дисциплины «Компьютерная графика» являются:

- изучение методов и алгоритмов визуализации изображений;
- изучение методов и алгоритмов обработки изображений;
- изучение методов и алгоритмов распознавания изображений;
- развитие алгоритмического мышления и формирование способности аргументированного обоснования выбора методов и алгоритмов визуализации, обработки и распознавания изображений;
- изучение методов формализации на языке программирования Visual C++ алгоритмов визуализации, обработки и распознавания изображений;
- изучение графической библиотеки OpenGL;
- получение практических навыков программирования алгоритмов визуализации, обработки и распознавания изображений в среде программирования Microsoft Visual Studio 2010 Professional и с использованием графической библиотеки OpenGL.

Методические указания содержат теоретическое обоснование и варианты заданий для выполнения практических работ.

Практические работы (34 часа).

Методические указания разработаны в соответствии с требованиями государственного образовательного стандарта по подготовке бакалавров по направлению 09.03.04 «Программная инженерия».

1 Основные понятия и определения компьютерной графики

Компьютерная графика (машинная графика, computer graphics) – дисциплина, изучающая методы генерации, преобразования, обработки и хранения моделей объектов и их изображений средствами вычислительной техники [1].

Направления компьютерной графики:

- изобразительная компьютерная графика;
- обработка изображений;
- распознавание изображений;
- когнитивная компьютерная графика [1].

Изобразительная компьютерная графика (pictorial computer graphics) – создание изображений на основе исходной информации неизобразительной природы. Например, визуализация экспериментальных данных в виде графиков, гистограмм или диаграмм, вывод информации на экран компьютерных игр, синтез сцен на тренажерах [1].

Обработка изображений (image processing) – преобразование изображений. Входными и выходными данными является изображение [2; 3]. Например, передача изображения с устранением шумов и сжатием данных. Целью обработки изображений может быть улучшение качества в зависимости от определенного критерия (реставрация, восстановление) или преобразование, изменяющее изображение [1].

Распознавание изображений (computer vision) – совокупность методов и средств, позволяющих получить формальное описание изображения и отнести его к некоторому классу. Например, оптическое распознавание символов, штрих-кодов, автомобильных номеров, лиц, отпечатков пальцев. Задача распознавания является обратной по отношению к визуализации [1].

Когнитивная компьютерная графика (cognitive computer graphics) – раздел компьютерной графики, визуализирующий научные абстракции с целью создания нового научного знания. Например, фракталы [1].

Графический формат (graphic format) – порядок (структура), согласно которому данные, описывающие изображение, записаны в файле [4].

Графические данные подразделяются на две группы:

- растровые данные;
- векторные данные.

Растровые данные (raster data) – набор числовых значений, определяющих яркость и цвет отдельных пикселов.

Векторные данные (vector data) – данные для представления прямых и кривых линий, многоугольников и т. д. с помощью определённых в числовом виде базовых (опорных) точек [4].

Графический файл (graphic file) – последовательность данных (структур данных), называемых файловыми элементами (элементами данных) [4].

Файловые элементы подразделяются на три категории: поле, тег, поток.

Поле (field) – структура данных в графическом файле, имеющая фиксированный размер.

Тег (tag) – структура данных, размер и позиция которой изменяются от файла к файлу.

Поток (stream) – набор данных, предназначенный для последовательного чтения [4].

2 Графические примитивы

2.1 Создание графических примитивов

Графический примитив – простейший геометрический объект, отображаемый на экране дисплея или на рабочем поле графопостроителя: точка, отрезок прямой, дуга окружности или эллипса, прямоугольник и т. п. Из графических примитивов строятся сложные графические изображения.

Разработаем визуальные приложения, формализующие графические примитивы, используя Windows Forms (WF) и Microsoft Foundation Classes (MFC).

2.1.1 Применение библиотеки Windows Forms

Для создания визуального приложения, отображающего графические примитивы с применением библиотеки WF, выполняются этапы:

1 Запустить среду программирования Microsoft Visual Studio 2010 Professional (рисунок 2.1).

2 В меню **Файл** выбрать команду **Создать ->Проект** (рисунок 2.2).

3 В окне **Создать проект** выбрать тип приложения **CLR** и вид приложения **Windows Forms** (рисунок 2.3).

4 В поле имя проекта ввести **Program_1**. Нажать кнопку **<OK>** (рисунок 2.4).

5 Панель элементов **<Ctrl-Alt-X>**. Выбор элемента **ListBox** (рисунок 2.5).

6 Разместить элемент **ListBox** на форме (рисунок 2.6).

7 Сделать двойной щелчок кнопкой мышки при расположении курсора мышки в пределах формы. Заполнить обработчик события загрузки формы.

private: System:: Void Form1_Load(System::Object^ sender, System::EventArgs^ e) { }

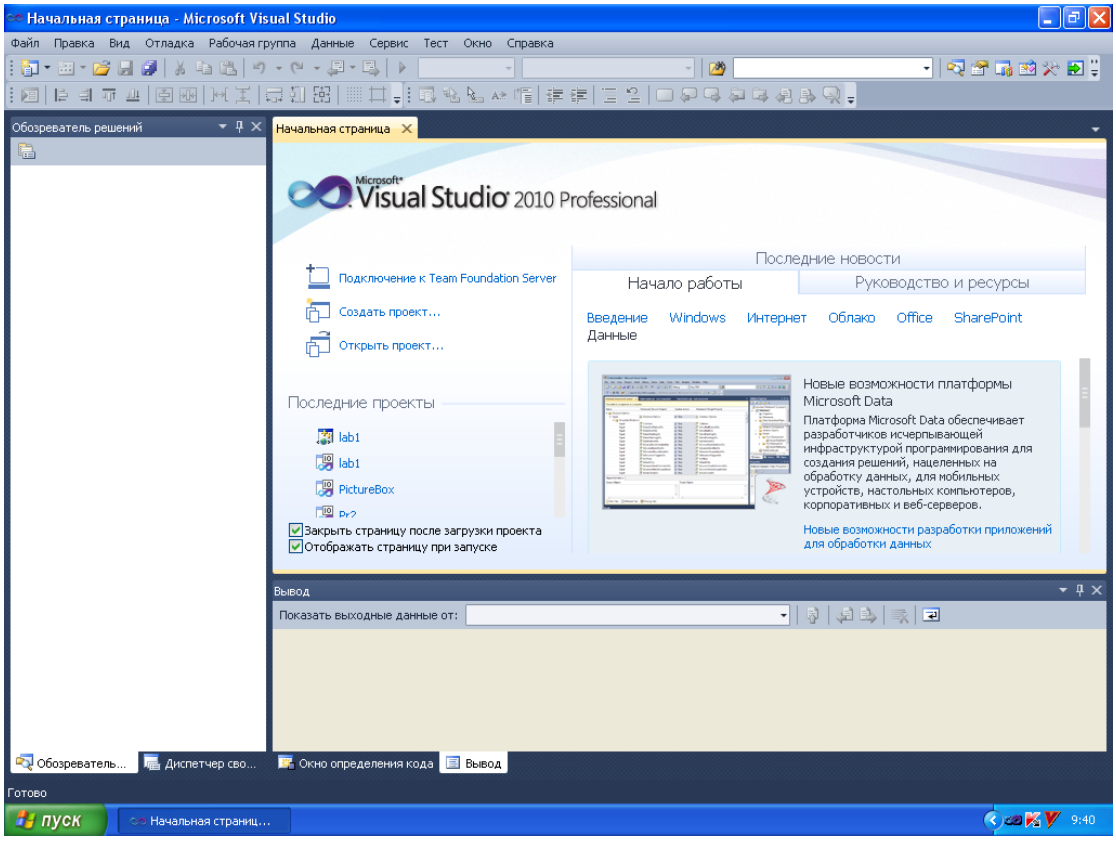

Рисунок 2.1 – Среда программирования Studio 2010 Professional

8 Сделать двойной щелчок кнопкой мышки при расположении курсора мышки в пределах элемента **ListBox**. Заполнить обработчик.

private: System::Void listBox1_SelectedIndexChanged_1 (System::Object^ sender, System::EventArgs^ e) { }.

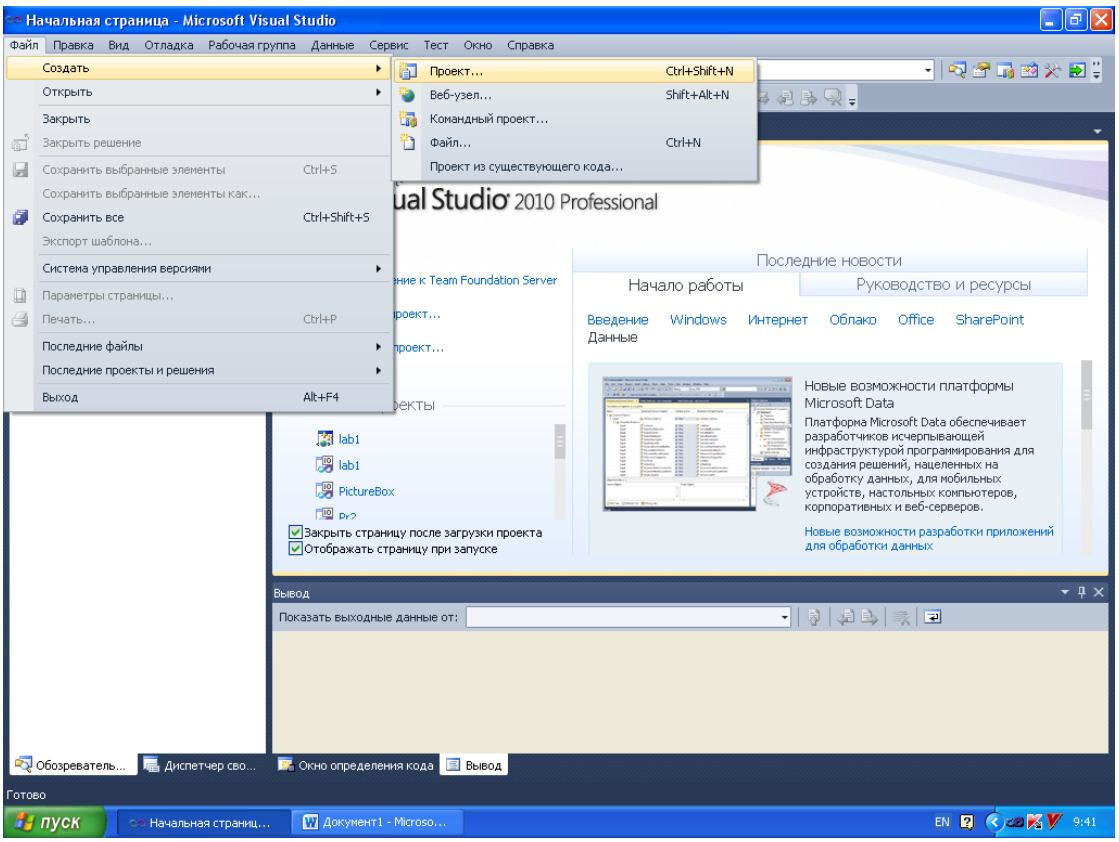

Рисунок 2.2 – Создание проекта

Код программы, позволяющей рисовать в форме графические примитивы: окружность, отрезок, прямоугольник, сектор, текст, эллипс, закрашенный сектор приведён в листинге 2.1.

Листинг 2.1 – Программа, представляющая графические примитивы

```
#pragma endregion
        // Программа позволяет рисовать в форме графические приитивы:
        // окружность, отрезок, прямоугольник, сектор, текст, эллипс
        // закрашенный сектор. Выбор того или иного графического примитива осуществляется с помощью элемента управления ListBox
    private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e) { this->Text="Выберите графический примитив";
    listBox1->Items->AddRange(gcnew array<Object^> {"Окружность", "Отрезок", "Прямоугольник", "Сектор", "Текст", "Эллипс", "Закрашенный сектор"});
        Font=gcnew System::Drawing::Font("Times New Roman", 14.F);  }
        private: System::Void listBox1_SelectedIndexChanged_1 (
                System::Object^ sender, System::EventArgs^ e) {
// Здесь вместо этого события можно было бы обработать событие
// listBox1_Click
            // Создание графического объекта
              Graphics^ Графика=this->CreateGraphics();
            // Создание пера для рисования фигур
```

```
Pen^ Перо=gcnew Pen(Color::Red);
// Создание кисти для закрашивания фигур
Brush^ Кисть=gcnew SolidBrush(Color::Red);
// Очистка области рисования путем ее окрашивания
// в первоначальный цвет формы
Графика->Clear(SystemColors::Control);
// или Графика->Clear(Color::FromName("Control"));
// или Графика->Clear(ColorTranslator::FromHtml("#EFEBDE"));
switch(listBox1->SelectedIndex) // Выбор фигуры
{ case 0: // Выбрана окружность
   Графика->DrawEllipse(Перо, 50, 50, 150, 150); break;
case 1: // Выбран отрезок
   Графика->DrawLine(Перо, 50, 50, 200, 200); break;
case 2: // Выбран прямоугольник
   Графика->DrawRectangle(Перо, 50, 30, 150, 180); break;
case 3: // Выбран сектор
   Графика->DrawPie(Перо, 40, 50, 200, 200, 180, 225); break;
case 4: // Выбран текст
   Графика->DrawString("Работаем с графическими примитивами\n"+"Курганский государственный университет", Font, Кисть, 10, 100);
break;
case 5: // Выбран эллипс
   Графика->DrawEllipse(Перо, 30, 30, 150, 200); break;
case 6: // Выбран закрашенный сектор
   Графика->FillPie(Кисть, 20, 50, 150, 150, 0, 45); break;
} } }; }
```

9 Откомпилировать. Устранить ошибки. Запустить программу на выполнение (рисунок 2.8 – рисунок 2.14).

2.1.2 Применение библиотеки Microsoft Foundation Classes

Для создания приложения, отображающего ломаную линию с применением библиотеки MFC, выполняются этапы:

1 Добавим строки, сохраняющие координаты указателя мышки в момент нажатия левой кнопки мышки (листинг 2.2).

Листинг 2.2 – Сохранение координат указателя мышки.

```
void CPainterView::OnLButtonDown(UINT nFlags, CPoint point)
{  //получили указатель на объект-документ
   CPainterDoc *pDoc=GetDocument();
```

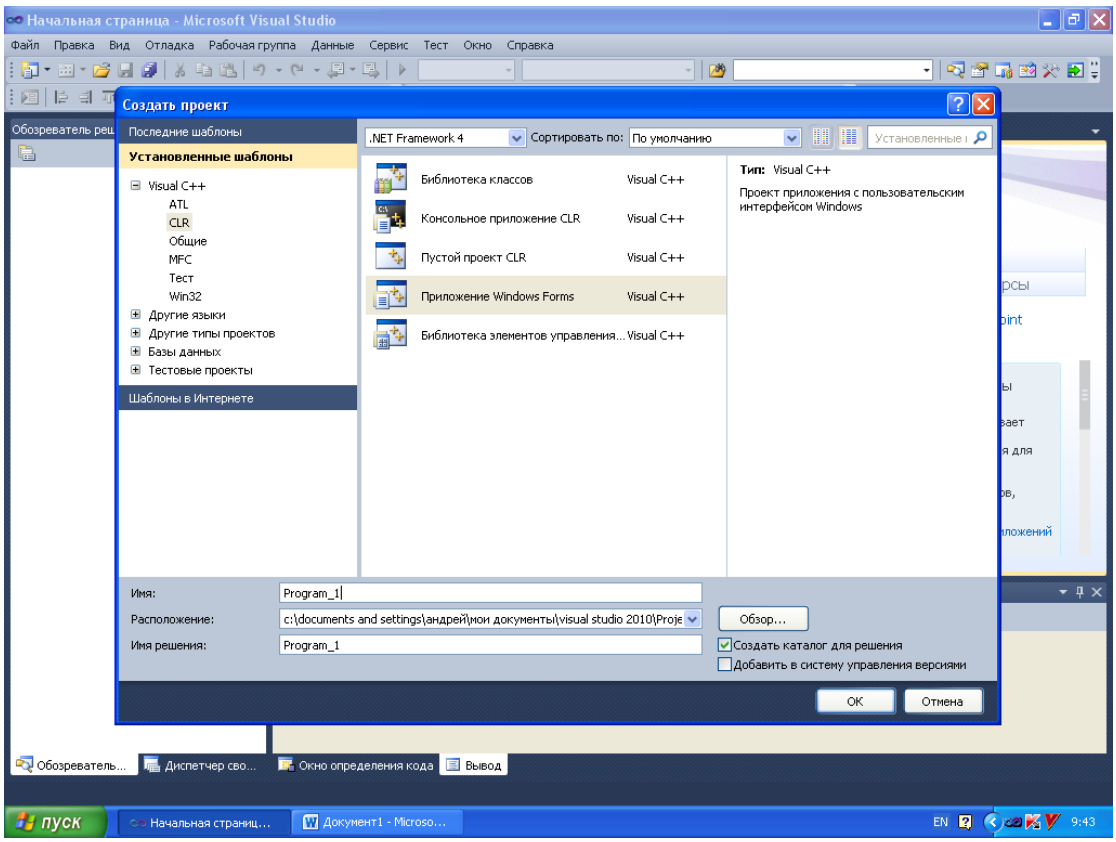

Рисунок 2.3 – Выбор типа и вида приложения

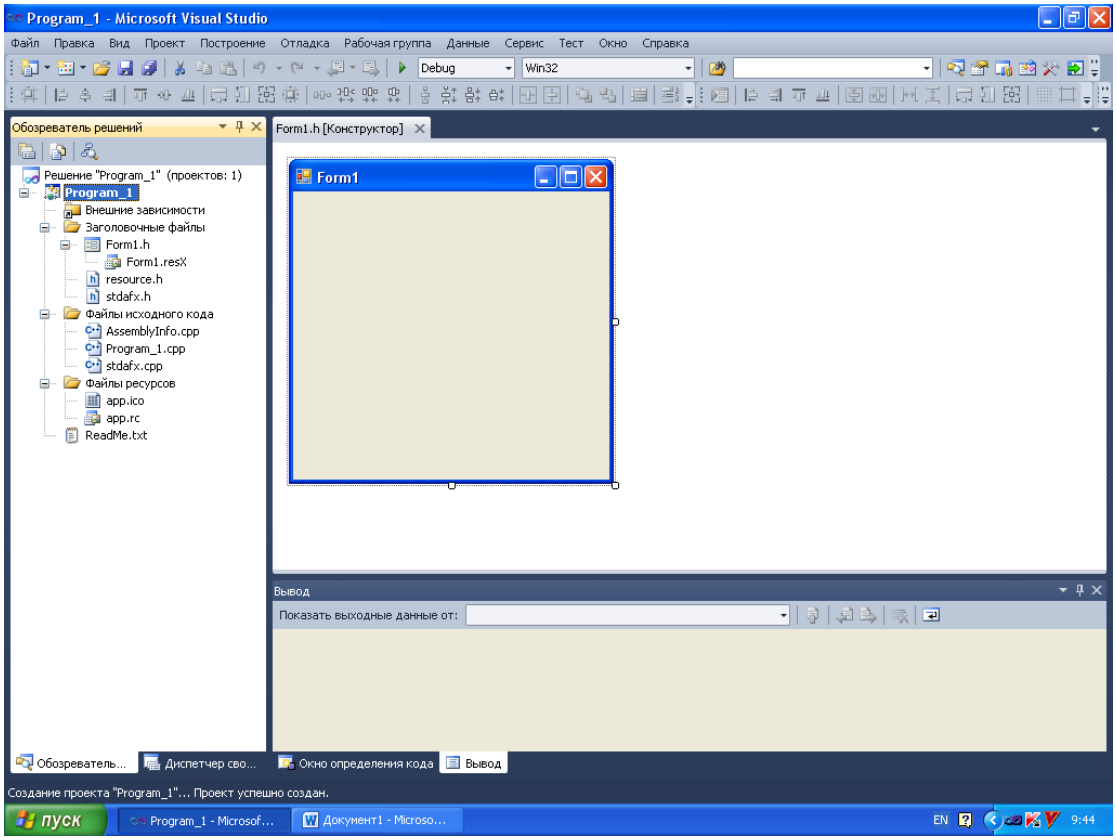

Рисунок 2.4 – Создание каркаса приложения (проекта)

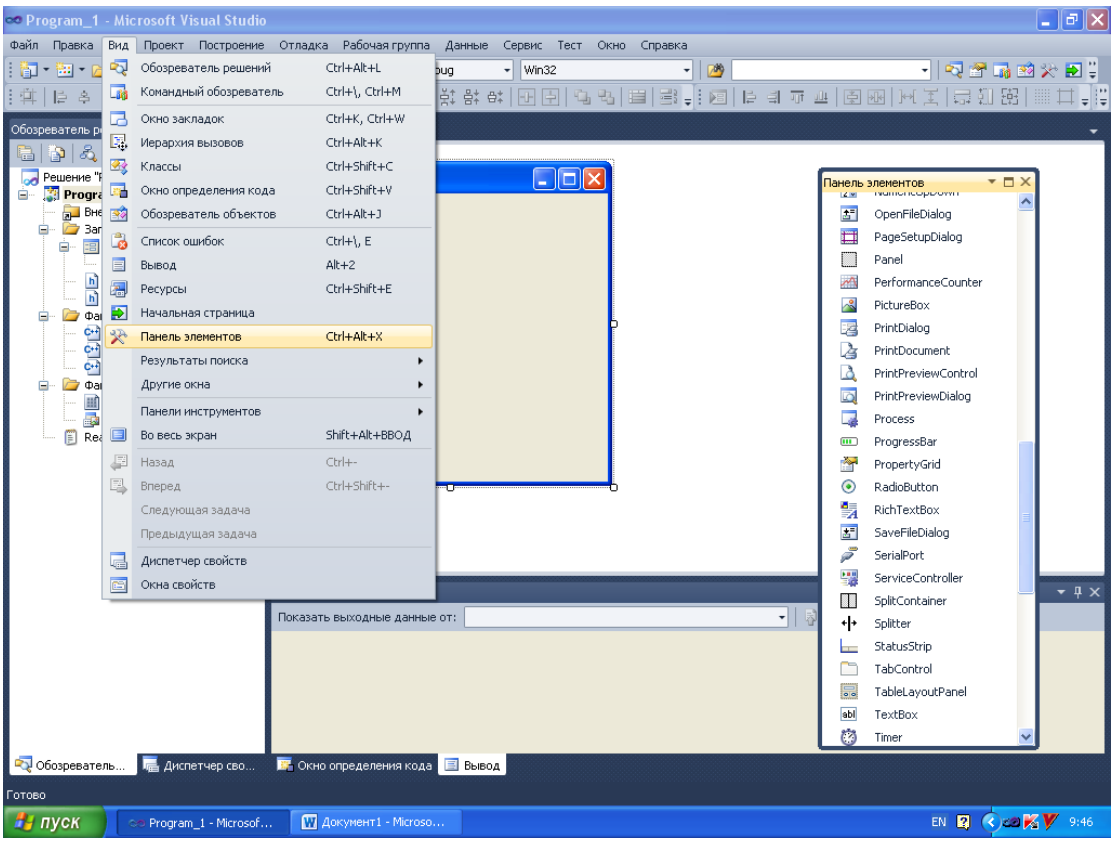

Рисунок 2.5 – Компонент ListBox

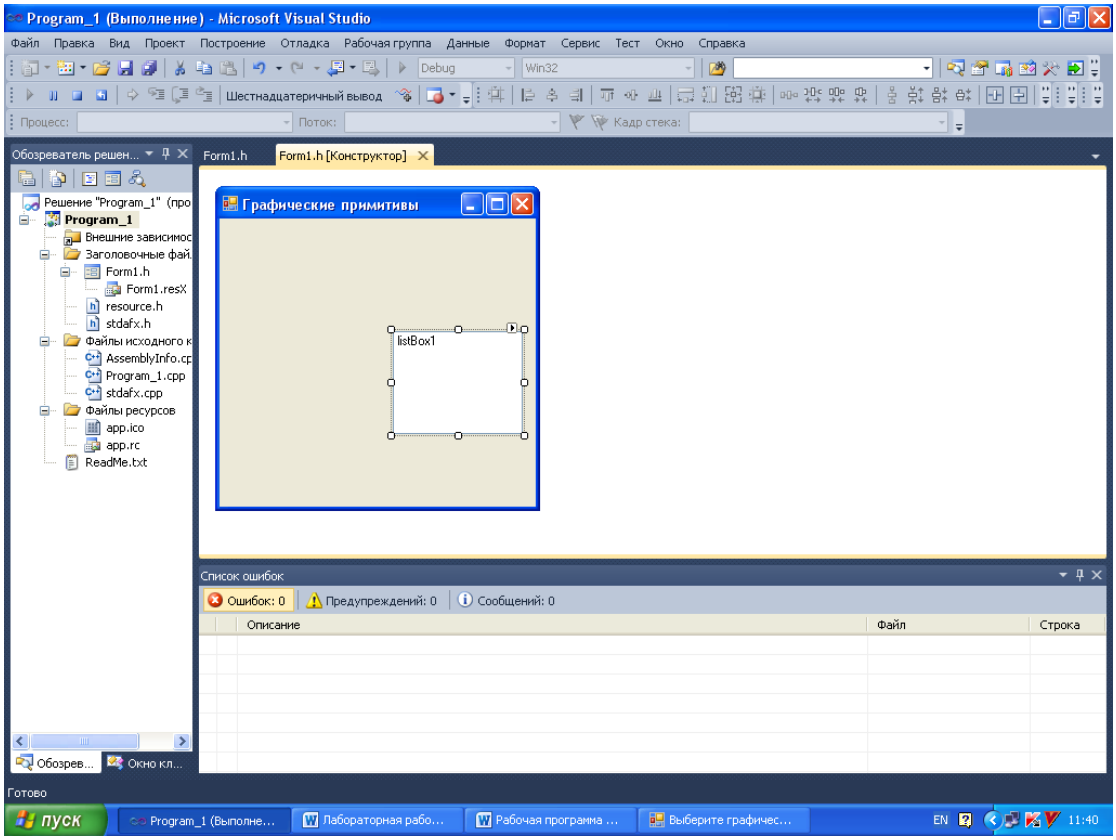

Рисунок 2.6 – Размещение элемента ListBox на форме

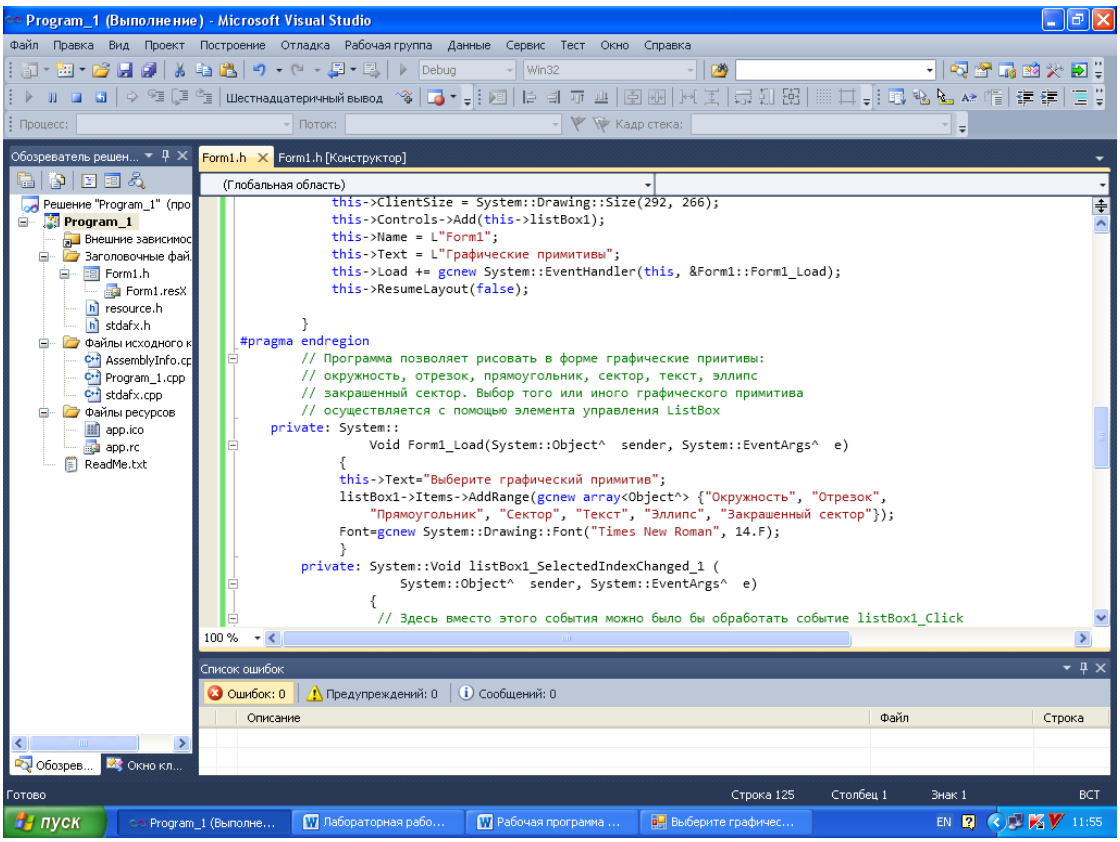

Рисунок 2.7 – Программный код

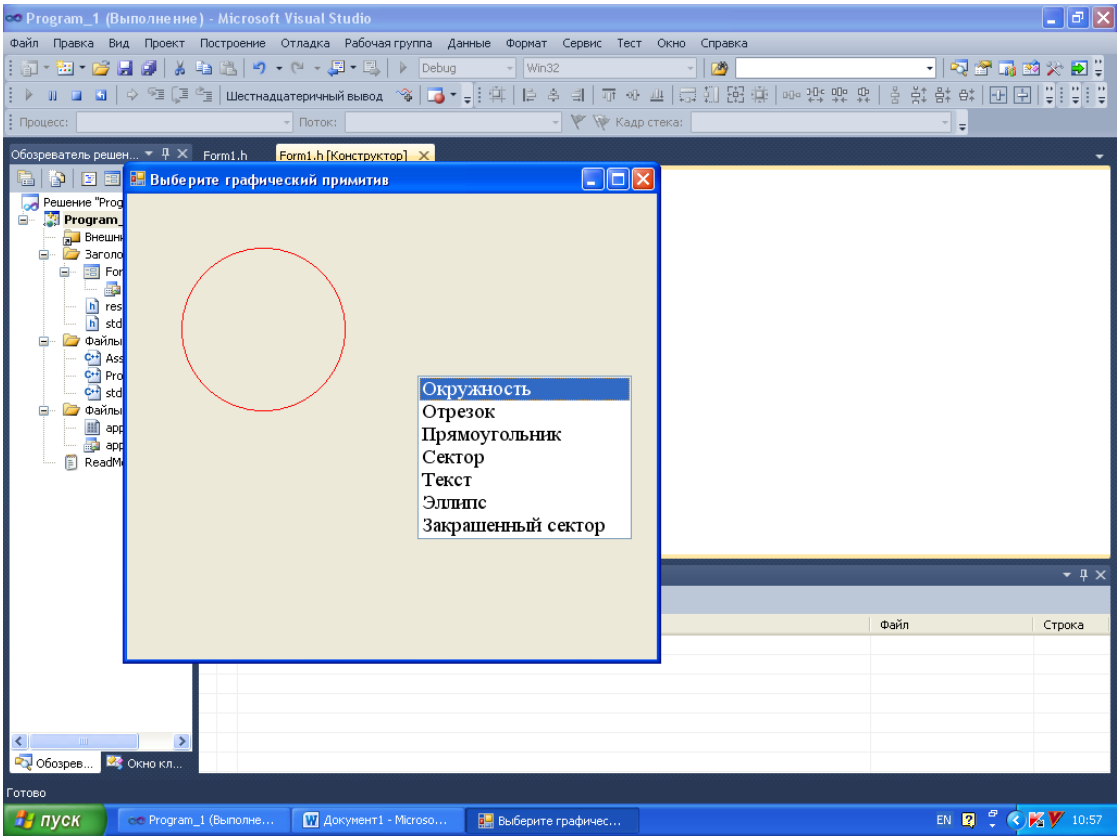

Рисунок 2.8 – Графический примитив окружность

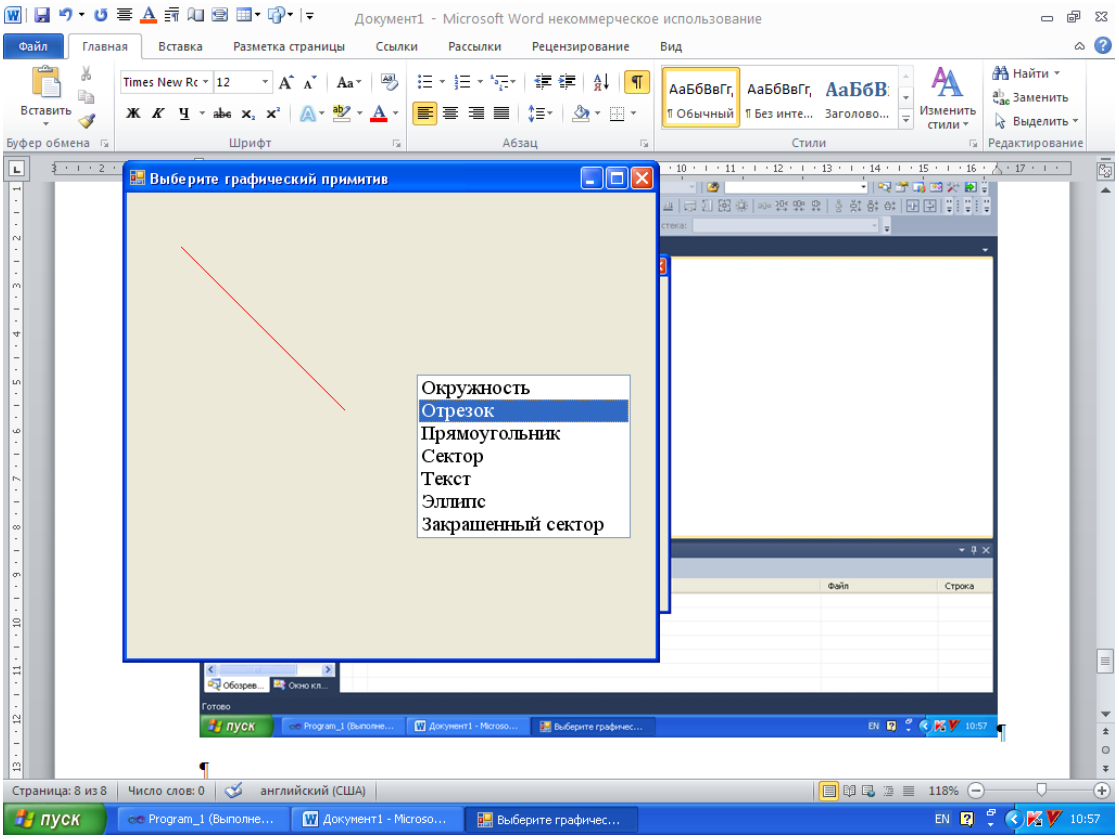

Рисунок 2.9 – Графический примитив отрезок

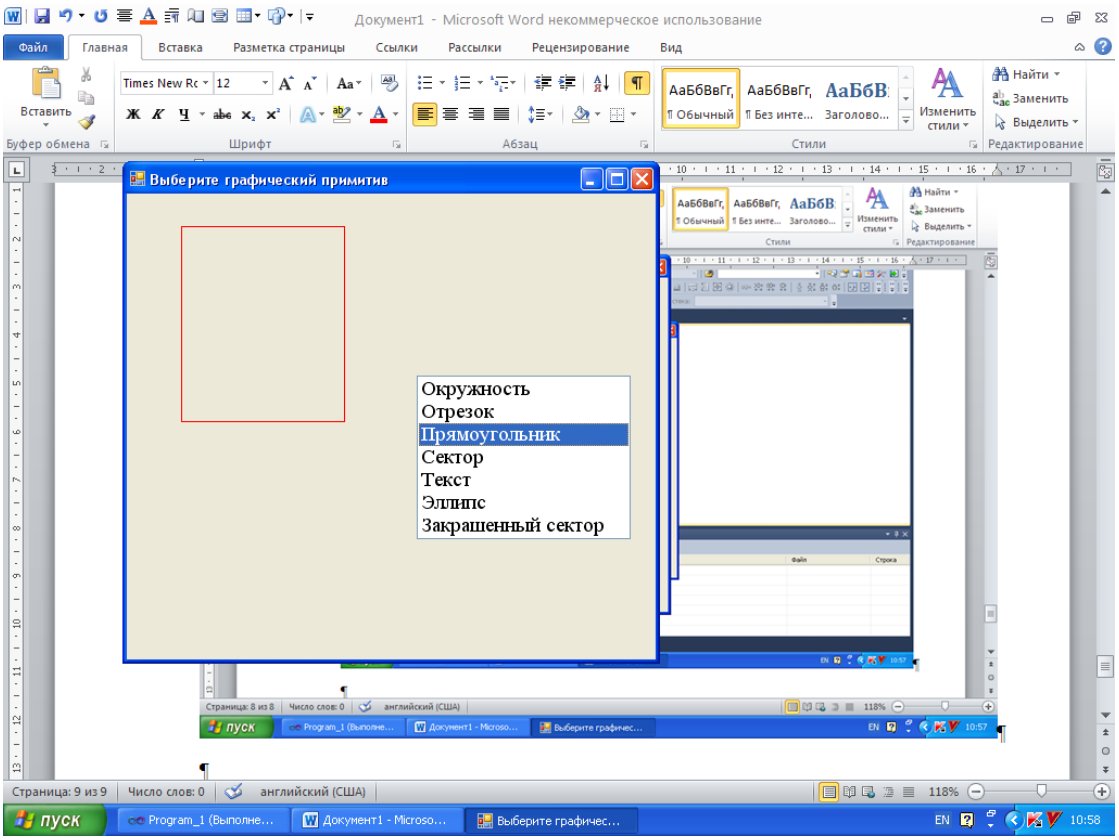

Рисунок 2.10 – Графический примитив прямоугольник

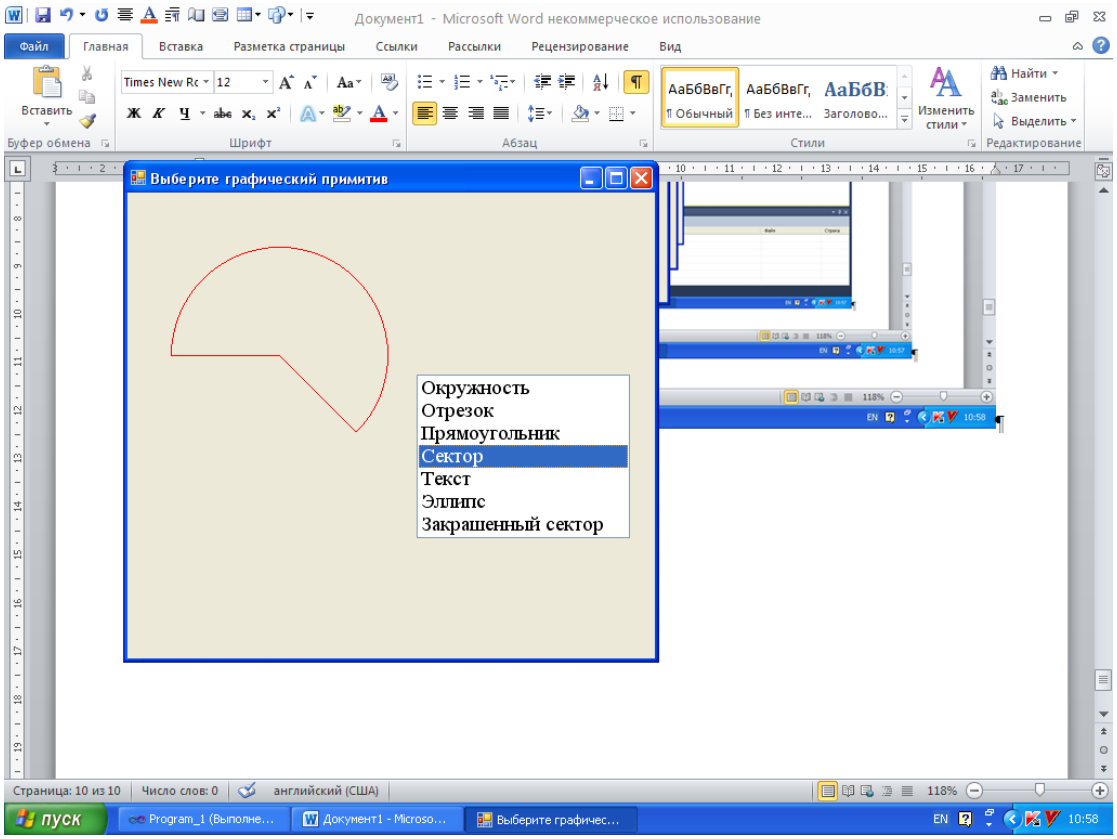

Рисунок 2.11 – Графический примитив сектор

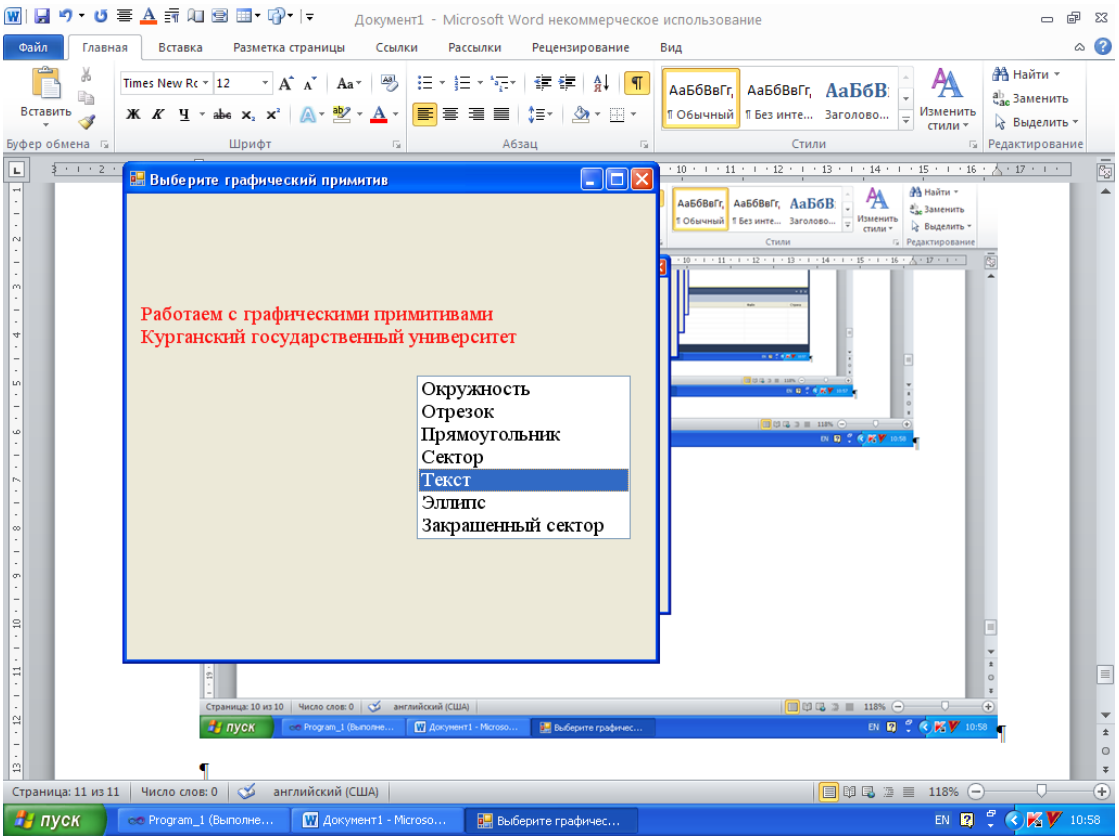

Рисунок 2.12 – Графический примитив текст

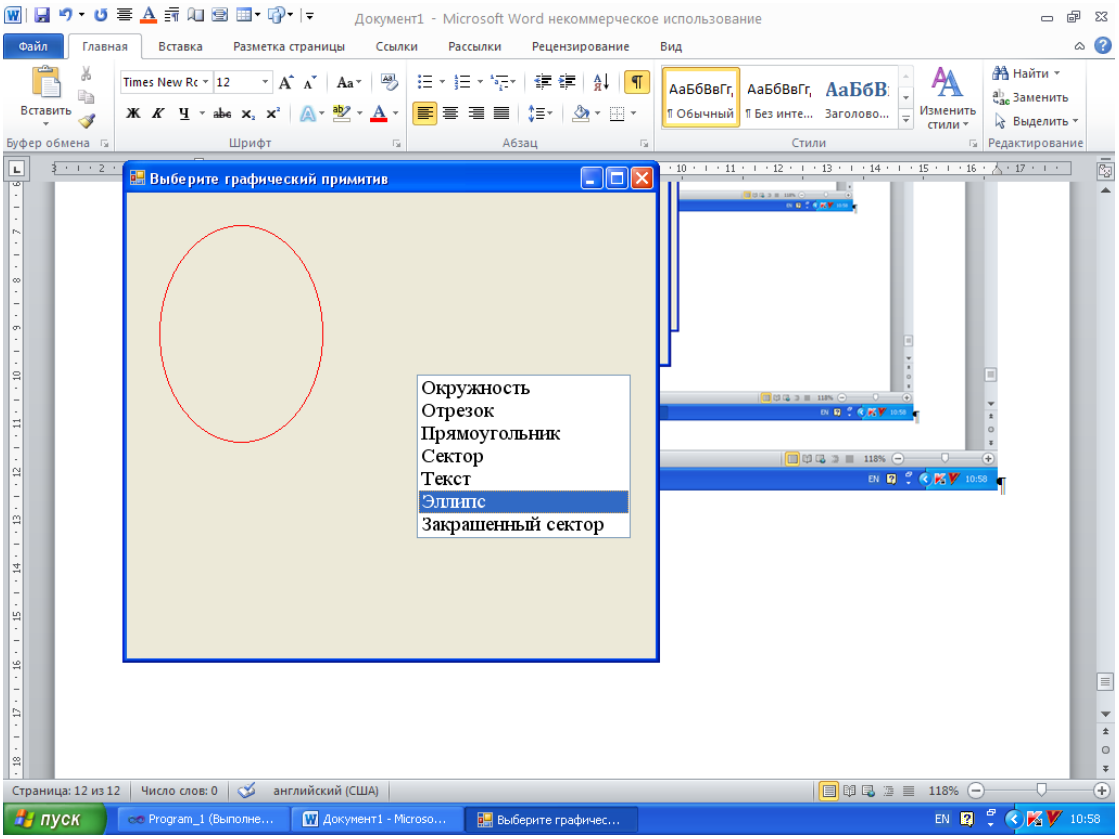

Рисунок 2.13 – Графический примитив эллипс

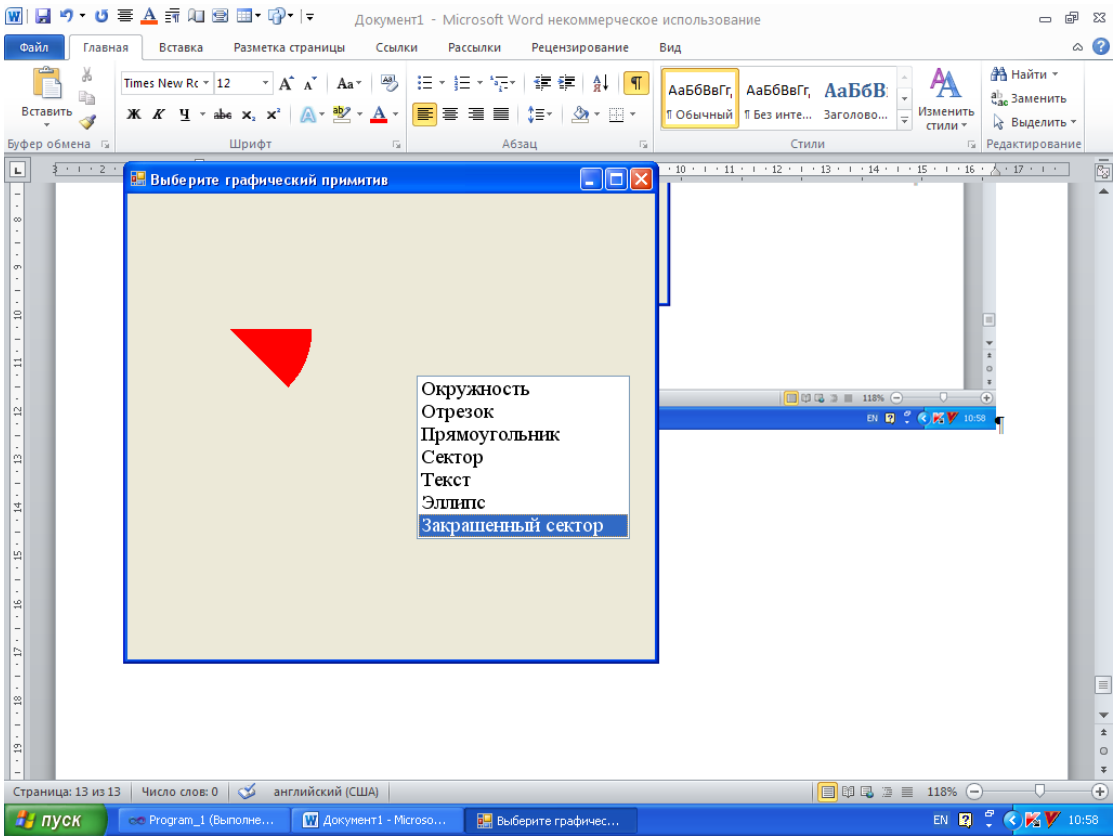

Рисунок 2.14 – Графический примитив закрашенный сектор

```
//проверим не исчерпали ли ресурс
        if(pDoc->m_nIndex==MAXPOINTS)
            AfxMessageBox(_T("Слишком много точек"));
        else   {   //запоминаем точку
        pDoc->m_Points[pDoc->m_nIndex++]=point;
        //указываем, что окно надо перерисовать
        Invalidate();
        // указываем, что документ изменен
        pDoc->SetModifiedFlag(); }
        CView::OnLButtonDown(nFlags, point);
    }
```

2 Добавим в метод OnDraw() класса CPainterView строки, которые будут выполнять вывод на экран линий, соединяющих опорные точки (листинг 2.3).

Листинг 2.3 – Вывод на экран линий, соединяющие опорные точки.

```
void CPR1View::OnDraw(CDC* pDC)
{  CPainterDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   if (!pDoc)
       return;
       //если имеются опорные точки
   CPen aPen;
   aPen.CreatePen(PS_SOLID,2,RGB(255,0,0));;
   CPen* pOldPen=pDC->SelectObject(&aPen);
   if(pDoc->m_nIndex>0)
       //поместим перо в первую из них
       pDC->MoveTo(pDoc->m_Points[0]);
   //пока не кончатся опорные точки будем их соединять
   for(int i=1; i<pDoc->m_nIndex; i++)
       pDC->LineTo(pDoc->m_Points[i]);
   pDC->SelectObject(pOldPen);}
```

3 Создание нового рисунка (листинг 2.4).

Листинг 2.4 – Добавление нового рисунка

```
 BOOL CPainterDoc::OnNewDocument()
{  if (!CDocument::OnNewDocument())
       return FALSE;
   // TODO: добавьте код повторной инициализации
   // (Документы SDI будут повторно использовать этот документ)
   // сбросили счетчик
   m_nIndex=0;
   // перерисовали
   UpdateAllViews(NULL);
   return TRUE;}
```

4 Сохранение рисунков в файл. Модифицируем функцию Serialize() класса CPainterDoc в файле PainterDoc.cpp (листинг 2.5).

Листинг 2.5 – Сохранение рисунков.

```
// сериализация CPainterDoc
void CPainterDoc::Serialize(CArchive& ar)
{  if (ar.IsStoring())
   {       //сохраняем количество точек
      ar << m_nIndex;
      //сохраняем значения координат точек
      for(int i=0; i<m_nIndex; i++) ar << m_Points[i];
   }  else   {       //загружаем количество точек
      ar >> m_nIndex;
      //загружаем значения координат точек
      for(int i=0; i<m_nIndex; i++) ar >> m_Points[i];
   }}
```

Результат работы визуального приложения, отображающего ломаную линия с применением библиотеки Microsoft Foundation Classes, представлен на рисунке 2.15.

2.2 Практическая работа №1. Графические примитивы

Цель работы: Получить теоретические знания и практические навыки в использовании Microsoft Visual C++ 2010 для формализации приложения, представляющего графические примитивы.

Используемые приемы и технологии: Visual C++ 2010 Professional, библиотеки графических интерфейсов Windows Forms и Microsoft Foundation Classes.

Ключевые термины: графический примитив, графическая поверхность, карандаш, кисть, среда программирования, общеязыковая среда выполнения, библиотека Windows Forms, библиотека Microsoft Foundation Classes.

2.2.1 Варианты заданий к практической работе №1

Разработайте визуальное приложение на языке Visual C++, формализующее вывод флага зарубежной страны.

Вариант 1. Австрия.
Вариант 2. Великобритания.
Вариант 3. Бангладеш.
Вариант 4. Бельгия.
Вариант 5. Бенин.
Вариант 6. Ботсвана.
Вариант 7. Венгрия.
Вариант 8. Гамбия.

Вариант 9. Гвинея.
Вариант 10. Германия.
Вариант 11. Гренландия.
Вариант 12. Греция.
Вариант 13. Дания.
Вариант 14. Ирландия.
Вариант 15. Испания.
Вариант 16. Италия.
Вариант 17. Колумбия.
Вариант 18. Коста-Рика.
Вариант 19. Багамские острова.
Вариант 20. Исландия.
Вариант 21. Маврикий.
Вариант 22. Мадагаскар.
Вариант 23. Норвегия.
Вариант 24. Объединённые Арабские Эмираты.
Вариант 25. Таиланд.

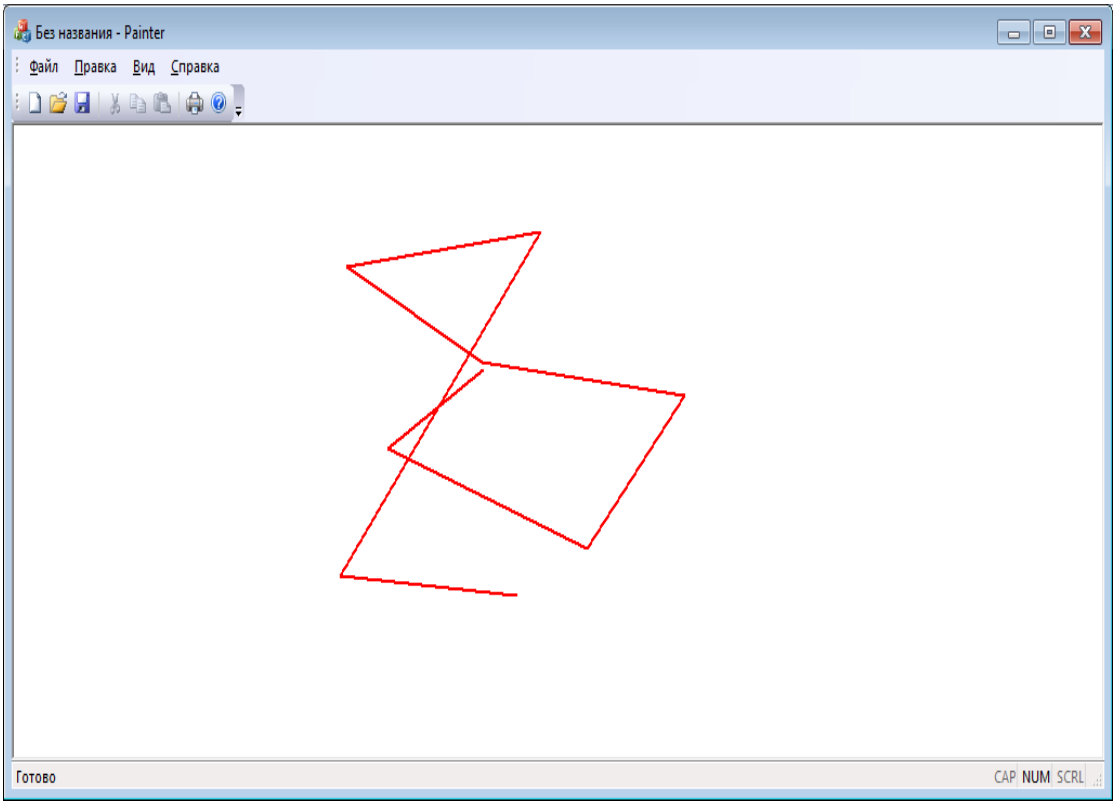

Рисунок 2.15 – Результат работы программы рисования ломаной линии

Методические указания

1 Запустите среду программирования Visual Studio 2010 Professional.
2 Создайте проект «Windows Forms Application Visual C++».

3 Формализуйте алгоритм решения задачи на ПЭВМ.

4 Выведите на экран видеомонитора результаты работы программы.

5 Оформите отчет по практической работе.

Контрольные вопросы

1 Что называется компьютерной графикой?

2 Какие направления применения имеет компьютерная графика?

3 Что называется графической поверхностью?

4 Что такое карандаш и кисть?

5 Что называется графическим примитивом? Приведите примеры графических примитивов.

6 Что такое графический формат? Приведите примеры графических форматов.

7 Что называется графическим файлом?

8 Сколько и какие категории файловых элементов существуют?

9 Сколько и какие наборы карандашей существуют?

10 Сколько и какие типы кистей существуют?

11 Какой объект представляет графическая поверхность компонента PictureBox?

12 Какие существуют этапы создания визуального приложения на Visual C++, формализующего вывод графического изображения?

3 Работа с прозрачными и анимированными файлами формата GIF

3.1 Битовый образ

Битовый образ – графическое изображение, находящееся в оперативной памяти компьютера. Битовые образы применяются для формирования сложных изображений.

Создание битового образа (объект Bitmap) производится способами:

- загрузка из графического файла (bmp, jpg, gif) ресурса;
- копирование из графического объекта (Image).

Загрузка битового образа из файла осуществляется конструктором, которому в качестве параметра передаётся имя файла. Метод *DrawImage()* выводит на графическую поверхность (PictureBox) битовый образ. Прозрачный цвет задаёт метод *MakeTransparent()*.

3.2 Создание динамического изображения

Анимация (animation) – последовательность кадров, которые воспринимаются как кино. В анимации используются спрайты.

Спрайт(sprite) – растровое изображение объекта рисунка, которое сохраняется в битовом массиве и быстро копируется в нужное место.

Альфа-канал – характеристика прозрачности.

Создание динамического изображения (анимации) производится способами:

- вывод на графическую поверхность последовательности подготовленных кадров анимации;
- формирование кадров из подготовленных фрагментов во время выполнения программы.

3.3 Вывод изображений с использованием библиотеки GDI+

Библиотека GDI+ – библиотека, основанная на классах интерфейса прикладных программ. GDI+ можно использовать для вывода изображений, которые существуют в приложении в качестве файлов. Это осуществляется путем создания объекта класса Image (объект Bitmap), создания объекта Graphics, который ссылается на поверхность рисования, и вызова метода DrawImage( ) объекта Graphics. Изображение будет выведено на поверхность рисования, предоставленную графическим классом. С помощью редактора изображений можно создавать и редактировать файлы изображений и отображать их с использованием GDI+ в режиме выполнения.

Вывод изображения с помощью GDI+:

1 Создайте объект, представляющий изображение для вывода. Объект должен быть членом класса, наследуемого от Image, например Bitmap или Metafile.

```
// Uses the System.Environment.GetFolderPath to get the path to the
// current user's MyPictures folder.
Bitmap myBitmap = new Bitmap
(System.Environment.GetFolderPath
(System.Environment.SpecialFolder.MyPictures));
```

2 Создайте объект Graphics, представляющий поверхность рисования для использования.

```
// Creates a Graphics object that represents the drawing surface of // Button1.
Graphics g = Button1.CreateGraphics();
```

3 Вызовите метод DrawImage ( ) графического объекта, чтобы вывести изображение. Следует указать изображение и координаты для отображения.

```
g.DrawImage(myBitmap, 1, 1);
```

Программный код создания обработчика рисования и вывода изображений приведён в листинге 3.1.

Листинг 3.1 Программный код создания обработчика рисования и вывода изображений.

```
// Обработчик рисования
void OnPaint(HDC hdc, RECT& rc)
{  Graphics g(hdc);
   Rect paintRect(0, 0, rc.right, rc.bottom);
   Bitmap backBuffer(rc.right, rc.bottom, &g);
```

```
    Graphics temp(&backBuffer);
    PaintBackground(temp, paintRect);
    PaintSocket(temp, paintRect);
    PaintPterodactyl(temp, paintRect);
    g.DrawImage(&backBuffer, 0, 0, 0, 0, rc.right, rc.bottom, UnitPixel);
//  i++;
}
//Рисование фона
void PaintBackground(Graphics& g, Rect& rc)
{  WCHAR text[]=L"Демонстрация работы \n"
            L"с прозрачными \n"
            L"и анимированными \n"
            L"файлами формата GIF.";
#ifdef DRAW_COMPLEX_BACKGROUND
  g.DrawImage(GIF2, 0,0, GIF2->GetWidth()+90, GIF2->GetHeight());
#else
    g.Clear(Color(255, 200, 200, 200));
#endif
//  g.DrawImage(GIF2, 0,0, GIF2->GetWidth()+90, GIF2->GetHeight());
#ifdef DRAW_TEXT_BACKGROUND
    g.DrawString(text, -1, font,
       PointF(float(rc.Width/2), float(rc.Height/2)), stringFormat, textBrush);
#endif }
//Рисование футболиста
void PaintPterodactyl(Graphics& g, Rect& rc)
{ else  { if(SocketPos.X-PterodactylPos.X>0) PterodactylPos.X+=5;
       else PterodactylPos.X-=5;
  if(SocketPos.Y-PterodactylPos.Y>0) PterodactylPos.Y+=5;
       else PterodactylPos.Y-=5;
  PterodactylImage->SelectActiveFrame(&FrameDimensionTime, activeFrame);
       g.SetInterpolationMode(InterpolationModeHighQualityBicubic);
       g.DrawImage(PterodactylImage, PterodactylPos.X-15, PterodactylPos.Y-40, PterodactylImage->GetWidth(), PterodactylImage->GetHeight());
       activeFrame=(activeFrame+1)%frameCount; }}
//Рисование мяча
void PaintSocket(Graphics& g, Rect& rc)
{ if(!SocketImage)
   {g.DrawString(L"Socket image load error", -1, font,
       PointF(float(rc.Width/2), float(rc.Height-20)),
       stringFormat, textBrush);}
else { g.DrawImage(SocketImage, SocketPos.X, SocketPos.Y,
SocketImage->GetWidth(), SocketImage->GetHeight());}}
```

Рисунок 3.1 – Результат работы программы вывода графических изображений и наложения текстового сообщения

3.4 Практическая работа №2. Создание динамических изображений и наложение текстового сообщения на gif изображение

Цель работы: Получить теоретические знания и практические навыки в применении Microsoft Visual C++ 2010 для разработки приложения, создающего динамические изображения с возможностью наложения текстового сообщения на изображение.

Используемые приемы и технологии: Visual C++ 2010 Professional, библиотеки графических интерфейсов Windows Forms и Microsoft Foundation Classes.

Ключевые термины: графический формат, битовый образ, анимация, алфа-канал.

3.5 Варианты заданий к выполнению практической работы №2

Разработайте визуальное приложение на Visual C++, отображающее перемещение двух объектов на фоне графического изображения, с наложенным текстовым сообщением. Перемещающиеся объекты стремятся соединиться друг с другом.

Вариант 1. Футбольное поле, мяч, футболист.

Вариант 2. Хоккейный корт, шайба, хоккеист.

Вариант 3. Новогодний праздник, ёлка, дед Мороз.

Вариант 4. Двор, косточка, собака.

Вариант 5. Космос, планета, космический корабль.

Вариант 6. Поляна, цветок, бабочка.
Вариант 7. Море, шлюпка, пловец.
Вариант 8. Небо, вражеский самолёт, зенитная ракета.
Вариант 9. Небо, самолёт-заправщик, бомбардировщик.
Вариант 10. Море, остров сокровищ, пиратский корабль.
Вариант 11. Комната, мышка, кошка.
Вариант 12. Пасека, медведь, рой пчёл.
Вариант 13. Море, вражеский авианосец, противокарабельная ракета.
Вариант 14. Арена цирка, горящий обруч, лев.
Вариант 15. Сад, корзина, яблоко.
Вариант 16. Огород, капуста, коза.
Вариант 17. Танковый биатлон, финиш, танк.
Вариант 18. Море, авианосец, самолёт.
Вариант 19. Соревнования по стрельбе из лука, мишень, стрела.
Вариант 20. Дорога, автомобиль, полицейский автомобиль.
Вариант 21. Аэродром, ограниченная по размеру площадка, планер.
Вариант 22. Город, автобус, пассажир.
Вариант 23. Университет, книга, студент.
Вариант 24. Концертный зал, барабан, барабанные палочки.
Вариант 25. Ипподром, конь, наездник.

Методические указания

1 Запустите среду программирования Visual Studio 2010 Professional.
2 Создайте проект «Windows Forms Application Visual C++».
3 Формализуйте алгоритм решения задачи на ПЭВМ.
4 Выведите на экран видеомонитора результаты работы программы.
5 Оформите отчет по практической работе.

Контрольные вопросы

1 Что называется битовым образом?
2 Какие существуют способы создания битового образа?
3 Какие методы используются при работе с битовым образом?
4 Что называется анимацией?
5 Какие существуют способы создания анимации?
6 Что называется альфа-каналом?
7 Какой метод задаёт прозрачный цвет?
8 Что называется спрайтом?
9 Каким образом достигается эффект равномерного движения объекта?
10 Что выполняет программная инструкция
Graphics->DrawImage(bitmap, r1, r2, GraphicsUnit::Pixel);?

4 Сплайновые кривые и сплайновые поверхности

4.1 Основные понятия и определения

Сплайн (англ. spline – лекало) – функция, область определения которой разбита на конечное число отрезков, на каждом из которых она совпадает с некоторым алгебраическим многочленом (полиномом).

Степень сплайна – максимальная из степеней использованных полиномов.

Дефект сплайна – разность между степенью сплайна и получившейся гладкостью.

Базовые (опорные точки) – набор точек, на основе которых выполняется построение кривой линии.

Интерполяция (лат. interpolis – разглаженный) – построение кривой линии, точно проходящей через набор базовых точек.

Аппроксимация (лат. proxima – ближайшая) – сглаживание, построение гладкой кривой линии, проходящей вблизи базовых точек [4].

Экстраполяция (лат. extra – вне, снаружи и лат. polire – изменяю) – построение линии за пределами интервала, заданного набором базовых точек.

Пусть задана сетка w на отрезке $[a,b]$:

$$a = x_0 < x_1 < \ldots < x_{m-1} < x_m = b. \tag{4.1}$$

Пусть точки x_0 и x_m – граничные узлы сетки w, а точки x_1 и x_{m-1} – внутренние узлы сетки w. Сетка называется равномерной, если расстояние между соседними узлами одинаковое.

Функция $S(x)$, заданная на отрезке $[a,b]$ называется сплайном $p+1$ (степени p), если выполняются условия:

1) на каждом из отрезков $\Delta_i = [x_i, x_{i+1}]$, $i = \overline{0,m-1}$ является многочленом заданной степени $p \geq 2$ и записывается в виде:

$$S(x) = S_i(x) = \sum_{k=0}^{p} a_k^i (x - x_i)^k, \; i = \overline{0,m-1}; \tag{4.2}$$

2) дифференцируема $p-1$ раз на отрезке $[a,b]$:

$$S(x) \in C^{p-1}[a,b]. \tag{4.3}$$

Условие (4.2) означает непрерывность функции $S(x)$ и производных $S'(x), S''(x), \ldots, S^{(p-1)}(x)$ во всех внутренних узлах сетки w. На каждом из от-

резков Δ_i, $i=\overline{0,m-1}$ сплайн $S(x)$ является многочленом степени p и определяется на этом отрезке $p+1$ коэффициентом. Количество частичных отрезков m. Для определения сплайна необходимо найти $(p+1)*m$ чисел [5]:

$$a_k^i, k=\overline{0,p}, i=\overline{0,m-1}. \quad (4.4)$$

4.2 Сплайновые кривые

4.2.1 Параметрическое задание кривых

Параметрическое задание кривой записывается уравнением вида:

$$x=x(t),\ y=y(t),\ z=z(t),\ \alpha \le t \le \beta, \quad (4.5)$$

где t – параметр.

Пусть вектор **P** – массив опорных точек

$$P=\{P_i(x_i,y_i,z_i), i=0,1,2,..n\}. \quad (4.6)$$

Вектор **R** включает функции $x(t),y(t),z(t)$:

$$R(t)=(x(t),y(t),z(t)). \quad (4.7)$$

Для построения сплайновой кривой применяется метод составления линии из отдельных сегментов, описываемых элементарными уравнениями третьей степени:

1 Выбирают четыре точки с номерами $i-1, i, i+1, i+2$.

2 Задают диапазон изменения параметра $\alpha \le t \le \beta$.

3 Разбивают диапазон изменения параметра на m частей.

4 Рассчитываются m промежуточных точек сплайновой кривой между базовыми точками i и $i+1$.

5 Рассчитанные точки соединяются прямыми линиями.

Преимущества метода составления линии из отдельных сегментов:

1 Упрощение расчётов.

2 Применение уравнений невысоких степеней [4].

4.2.2 Интерполяционная кривая Catmull-Rom

Интерполяционная сплайновая кривая Catmul-Rom определяется по заданному массиву точек P_0, P_1, P_2, P_3 уравнением вида:

$$R(t) = \frac{1}{2}(-t(1-t)^2 P_0 + (2 - 5t^2 + 3t^3)P_1 + t(1 + 4t - 3t^2)P_2 - t^2(1-t)P_3). \quad (4.8)$$

Свойства составной сплайновой кривой Catmul-Rom:

- проходит через опорные точки;
- непрерывная;
- отсутствие регулирования формы [4].

4.2.3 Элементарная бета-сплайновая кривая

Элементарная бета-сплайновая кривая по заданному массиву точек P_0, P_1, P_2, P_3 описывается уравнением:

$$R(t) = b_0(t)P_0 + b_1(t)P_1 + b_2(t)P_2 + b_3(t)P_3. \quad (4.9)$$

Функциональные коэффициенты b_0, b_1, b_2, b_3 рассчитываются по формулам:

$$b_0(t) = \frac{2\beta_1^3}{\delta}(1-t)^3; \quad (4.10)$$

$$b_1 = \frac{1}{\delta}(2\beta_1^3 t(t^2 - 3t + 3) + 2\beta_1^2(t^3 - 3t + 2) + 2\beta_1(t^3 - 3t + 2) + \beta_2(2t^3 - 3t + 1)); \quad (4.11)$$

$$b_2(t) = \frac{1}{\delta}(2\beta_1^2 t^2(-t + 3) + 2\beta_1 t(-t^2 + 3) + 2\beta_2 t^2(-2t + 3) + 2(-t^3 + 1)); \quad (4.12)$$

$$b_3(t) = \frac{2t^3}{\delta}, \quad (4.13)$$

где $\beta_1 > 0$, $\beta_2 \geq 0$, $\delta = 2\beta_1^3 + 4\beta_1^2 + 4\beta_1 + \beta_2 + 2$.

Параметр β_1 – параметр скоса (смещения). Параметр β_2 – параметр натяжения.

Свойства бета-сплайновой кривой:

- проходит внутри выпуклой оболочки, заданной опорными точками;
- дважды геометрически непрерывная кривая;
- параметры β_1 и β_2 позволяют регулировать форму [4].

4.2.4 Сплайновая кривая Безье

Сплайновая кубическая элементарная кривая Безье по заданному массиву точек P_0, P_1, P_2, P_3 описывается уравнением:

$$R(t) = (((1-t)P_0 + 3tP_1)(1-t) + 3t^2P_2)(1-t) + t^3P_3. \quad (4.14)$$

Параметр t удовлетворяет условию $0 \le t \le 1$.

Кривая Безье начинается в точке P_0 и заканчивается в точке P_3, касаясь отрезков P_0P_1 и P_2P_3.

Свойства составной кривой Безье:

- проходит внутри выпуклой оболочки, заданной опорными точками;
- невозможно регулировать форму.

4.3 Сплайновые поверхности

Сплайновые поверхности строятся по заданным опорным векторам $\overrightarrow{V_{ij}}$, $i = \overline{0, N}$, $j = \overline{0, M}$.

Параметрическая форма записи сплайновой поверхности имеет вид:

$$\bar{r}(u,v) = \sum_{i=0}^{N} \sum_{j=0}^{M} a_i(u) b_j(v) \vec{V}_{ij}, \quad (4.15)$$

где $a(u)$, $b(v)$ – сплайновые функции, $u_{\min} \le u \le u_{\max}$, $v_{\min} \le v \le v_{\max}$.

Преобразуем уравнение 4.15 к виду:

$$\bar{r}(u,v) = \sum_{i=0}^{N} a_i(u) \vec{r}_i(v); \quad (4.16)$$

$$\bar{r}_i(v) = \sum_{j=0}^{M} b_i(v) \vec{V}_{ij}, \quad (4.17)$$

где $\bar{r}_i(v)$ – сплайновые кривые в параметрическом виде;

$\bar{r}(u,v)$ – сплайновые поверхности [5].

4.3.1 Бикубическая поверхность Безье

Элементарная бикубическая поверхность Безье задаётся 16 опорными векторами:

$$p = \begin{bmatrix} \vec{V}00 & \vec{V}01 & \vec{V}02 & \vec{V}03 \\ \vec{V}10 & \vec{V}11 & \vec{V}12 & \vec{V}13 \\ \vec{V}20 & \vec{V}21 & \vec{V}22 & \vec{V}23 \\ \vec{V}30 & \vec{V}31 & \vec{V}32 & \vec{V}33 \end{bmatrix}. \qquad (4.18)$$

В качестве функций $a_{i(u)}$, $bj(\nu)$ выбираем кубические сплайновые функции Безье. Сплайновая бикубическая элементарная поверхность Безье имеет вид:

$$\bar{r}(u,\nu) = \sum_{i=0}^{3} \sum_{j=0}^{3} b_{3,i}(u) b_{3,j}(\nu) \vec{V}ij, \qquad (4.19)$$
$$0 \le u \le 1,\ 0 \le \nu \le 1.$$

В матричной форме записи сплайновая бикубическая поверхность Безье запишется в виде:

$$\vec{r}(u,\nu) = u^T M_b p M_b \nu, \qquad (4.20)$$

где M_b – матрица Безье;

p – матрица 16 опорных векторов;

u – вектор-столбец $u = \begin{bmatrix} 1 & u^1 & u^2 & u^3 \end{bmatrix}^T$;

ν – вектор-столбец $\nu = \begin{bmatrix} 1 & \nu^1 & \nu^2 & \nu^3 \end{bmatrix}^T$ [5, 6, 7, 8, 9].

4.4 Создание сплайновой кривой Безье

Разработать визуальное приложение, формализующее сплайновую кривую Безье, используя библиотеку Windows Forms.

Последовательность выполнения задания.

1 Запустить среду программирования Microsoft Visual Studio 2010 Professional (рисунок 4.1).

2 В меню **Файл** выбрать команду **Создать ->Проект** (рисунок 4.2).

3 В окне **Создать** проект выбрать тип приложения **CLR** и вид **Приложение Windows Forms** (рисунок 4.3).

4 В поле имя проекта ввести **Program_3**. Навести указатель мышки на кнопку **<OK>** и щёлкнуть левой кнопкой мышки (рисунок 4.3). В результате работы мастера **Приложения Windows Forms** создаётся проект (рисунок 4.4).

5 Реализовать программный код.

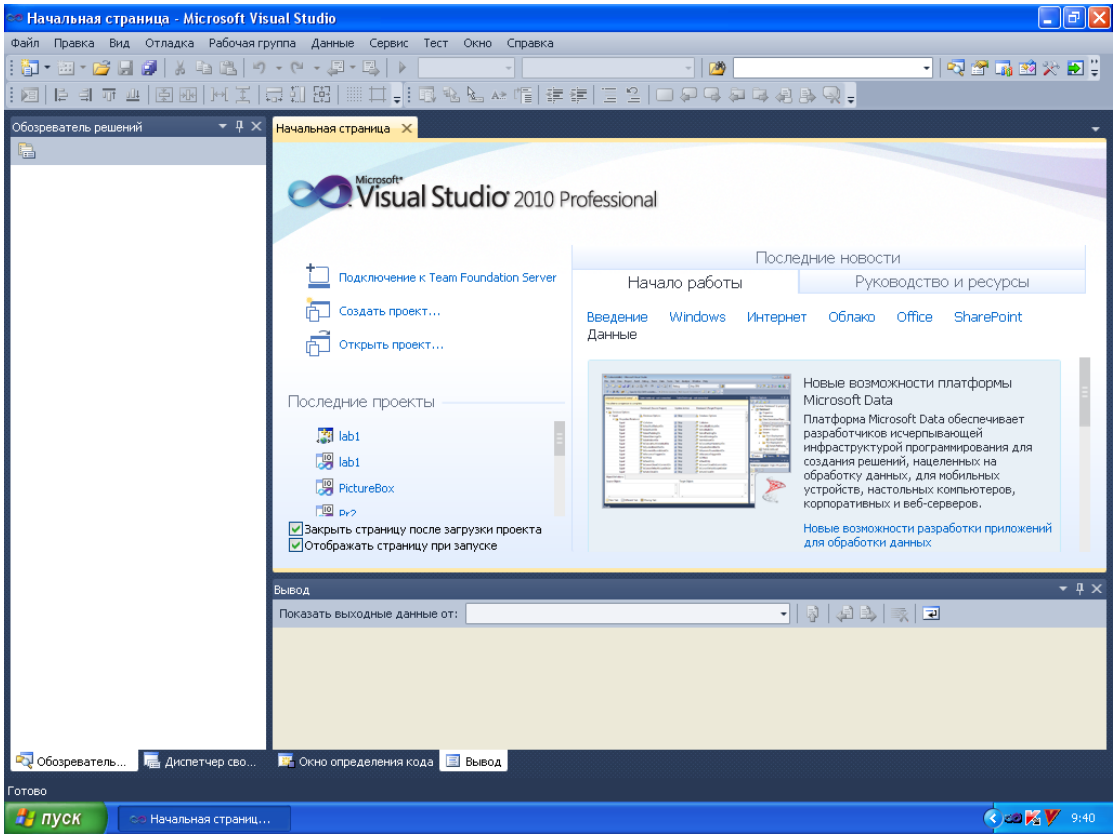

Рисунок 4.1 – Среда программирования Visual C++ 2010 Professional

Код программы построения сплайна Безье по двум узловым точкам приведён в листинге 4.1.

Листинг 4.1 – Программа построения сплайна Безье

```
#pragma endregion
        //Программа строит сплайн Безье по двум узловым точкам, а две
        //контрольные (управляющие) точки совмещены в одну. Эта одна
        //управляющая точка отображается в форме в виде красного
        //прямоугольника. Перемещая указателем мышки управляющую // точку, мы регулируем форму сплайна (кривой)
        array<PointF> ^МассивТочек;
        //Запрещение управлять формой кривой
        Boolean Управлять;
    private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e) { Управлять=false;
            this->Text="Управление сплайном Безье";
            МассивТочек=gcnew array<PointF>(4);
        //Начальная узловая точка:
            МассивТочек[0]=PointF(50.0f, 50.0f);
        //Две контрольные (управляющие) точки, мы их совместим в одну
            МассивТочек[1]=PointF(125.0f, 125.0f);
```

```
            МассивТочек[2]=PointF(125.0f, 125.0f);
        //Конечная узловая точка
            МассивТочек[3]=PointF(200.0f, 200.0f); }
    private: System::Void Form1_Paint(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ e) {//Задаем поверхность для рисования из аргумента события e
            Graphics ^Графика=e->Graphics;
            Pen ^Перо=gcnew Pen(Color::Blue, 3);
    //Рисуем начальную и конечную узловые точки диаметром 4 пикселя
            Графика->DrawEllipse(Перо, МассивТочек[0].X-2, МассивТочек[0].Y-2, 4.0f, 4.0f);
            Графика->DrawEllipse(Перо, МассивТочек[3].X-2, МассивТочек[3].Y-2, 4.0f, 4.0f );
    //Одна управляющая точка в виде прямоугольника красного цвета
            Перо->Color=Color::Red;
            Графика->DrawRectangle(Перо, МассивТочек[1].X-2, МассивТочек[1].Y-2, 4.0f, 4.0f);
            Перо->Color=Color::Blue;
    //Рисуем сплайн Безье
            Графика->DrawBeziers(Перо, МассивТочек);
            delete Графика; }
    private: System::Void Form1_MouseMove(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {
    //Событие перемещения указателя мышки в области экранной формы
    //Если указатель мышки расположен над управляющей точкой
            if(Math::Abs(e->X-МассивТочек[1].X)<4.0f &&
               Math::Abs(e->Y-МассивТочек[1].Y)<4.0f &&
    //и при этом нажата кнопка мышки
               Управлять==true)
            { //то меняем координаты управляющей точки
            МассивТочек[1].X=(float)e->X;
            МассивТочек[1].Y=(float)e->Y;
            МассивТочек[2].X=(float)e->X;
            МассивТочек[2].Y=(float)e->Y;
    //обновляем (перерисовываем) форму
            this->Invalidate(); } }
    private: System::Void Form1_MouseUp(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {
    //Если кнопку мышки отпустили, то запрещаем
    //управлять формой кривой
            Управлять=false; }
```

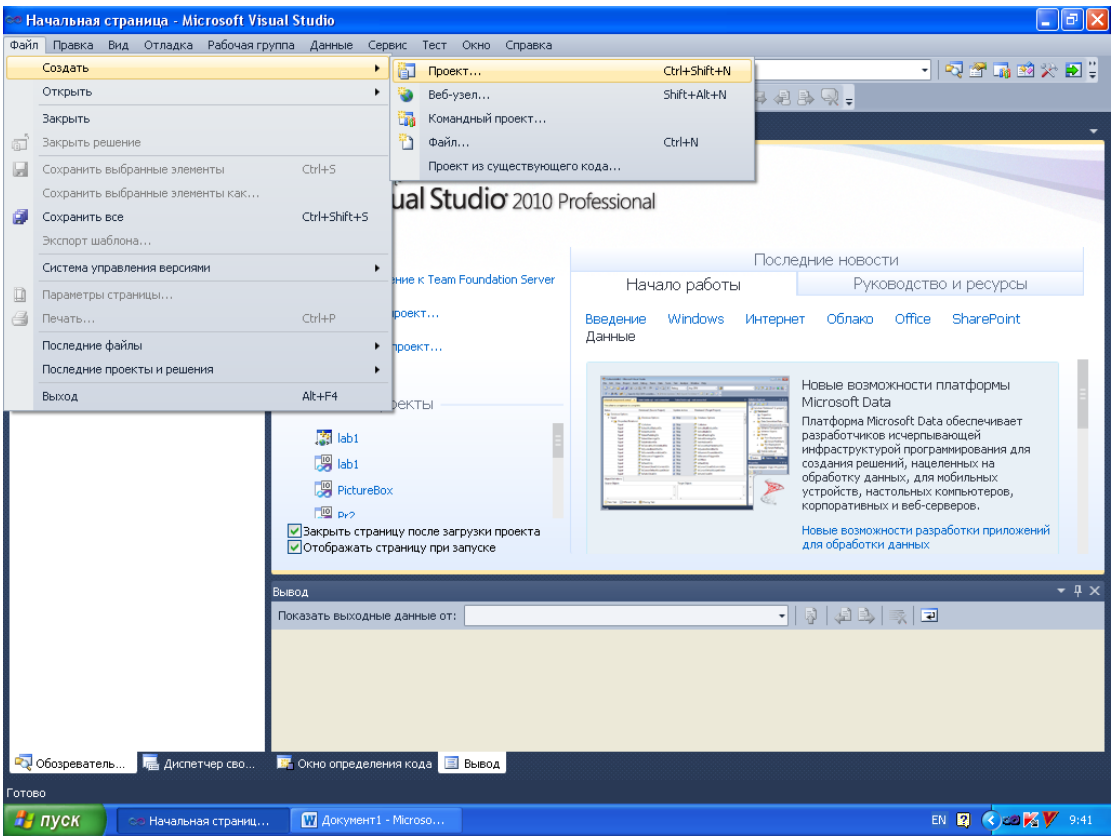

Рисунок 4.2 – Команда Файл -> Создать -> Проект

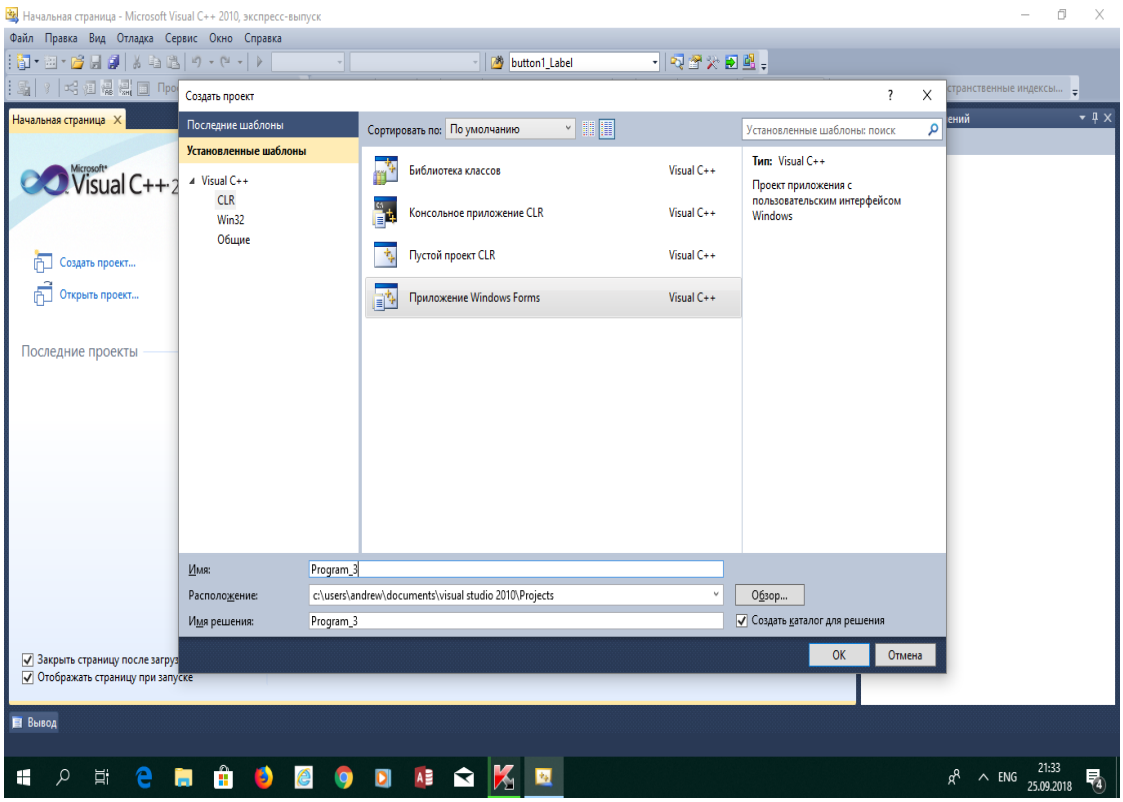

Рисунок 4.3 – Выбор типа и вида приложения

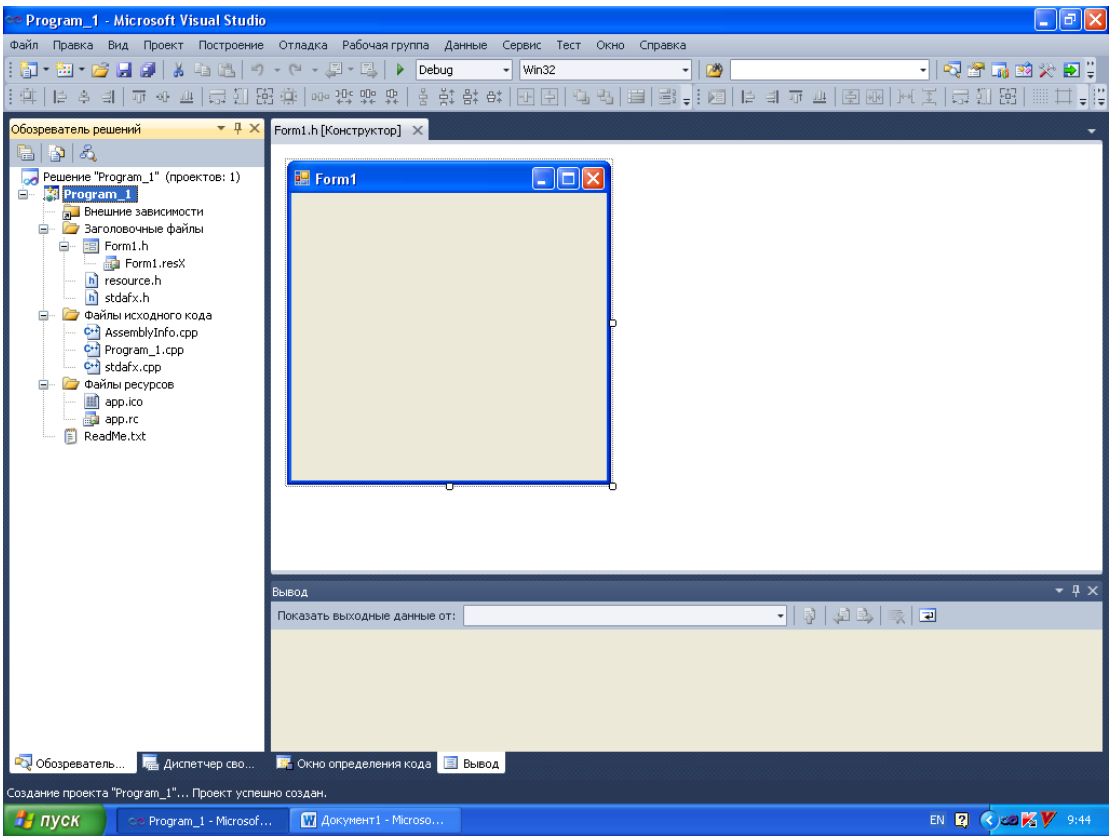

Рисунок 4.4 – Создание каркаса приложения (проекта)

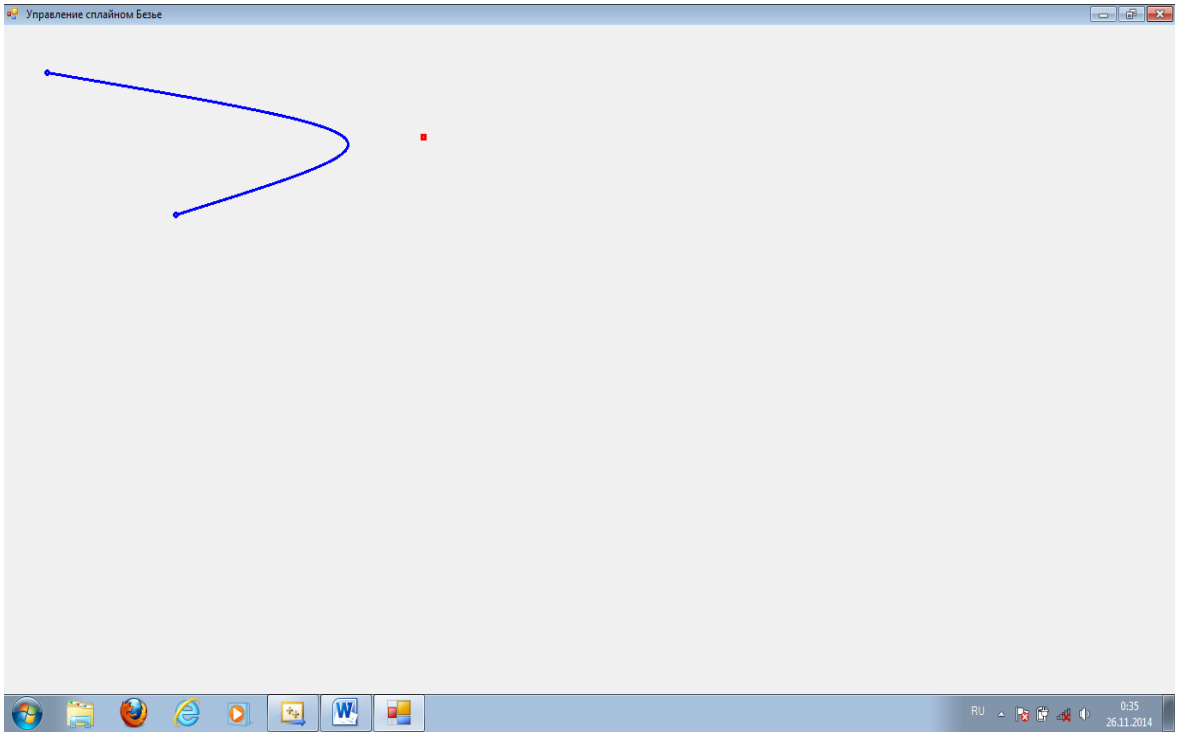

Рисунок 4.5 – Кривая Безье

private: System::Void Form1_MouseDown(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {//Если нажата кнопка мышки, то разрешаем управлять формой кривой

Управлять=true; }};}

6 Откомпилировать. Устранить ошибки. Запустить программу на выполнение (рисунок 4.5).

4.5 Практическая работа №3. Сплайновые кривые и сплайновые поверхности

Цель: Получить теоретические знания и практические навыки в применении Visual C++ для формализации приложения сплайновые кривые.

Используемые приемы и технологии: Visual C++ 2010 Professional, библиотеки графических интерфейсов Windows Forms и Microsoft Foundation Classes.

Ключевые термины: сплайн, базовые (опорные) точки, интерполяция, аппроксимация, экстраполяция, полигональная сетка, сплайновые кривые, сплайновые поверхности.

4.5.1 Варианты заданий к практической работе №3

Разработайте визуальное приложение на языке Visual C++, формализующее построение сплайновых кривых и сплайновых поверхностей. Варианты заданий приведены в таблице 4.1.

При реализации в программе B-сплайновых кривых и кривых Безье необходимо выполнить действия:

1 Формализовать алгоритм построения элементарных сплайновых кривых.

2 Формализовать алгоритм построения составных сплайновых кривых.

3 Формализовать алгоритм построения замкнутых сплайновых кривых.

4 Реализовать возможность в интерактивном режиме изменять положение опорных точек (с помощью мышки или клавиатуры).

При реализации в программе составных сплайновых поверхностей Безье и составных B-сплайновых поверхностей, аппроксимирующих заданные в вариантах поверхности второго порядка, выполните действия:

1 Реализуйте в программе создание сплайновых поверхностей из элементарных бикубических сплайновых поверхностей. Каждая элементарная бикубическая сплайновая поверхность строится по 16 опорным векторам, концы которых находятся в заданной поверхности второго порядка.

2 Число элементарных сплайновых поверхностей студент выбирает самостоятельно, но не менее 6. Программа должна быть оформлена так, чтобы можно было легко менять число элементарных поверхностей.

3 Концы опорных векторов должны быть соединены отрезками, чтобы можно было видеть каркасную модель изображаемого объекта.

4 На полученные сплайновые поверхности необходимо нанести узор – одну или несколько пространственных кривых, лежащих на этих поверхностях.

5 В программе должна быть предусмотрена возможность в интерактивном режиме вращать построенное изображение вместе с координатными осями (с помощью мышки или клавиатуры).

Таблица 4.1 – Варианты заданий

Номер варианта	**Кривые**	**Поверхности**
1	B-сплайн	Эллипсоид – Безье $\frac{x^2}{2^2}+\frac{y^2}{3^2}+\frac{z^2}{4^2}=1,\quad 0\le z\le 3.5$
2	Безье	Эллипсоид – B-сплайн $\frac{x^2}{2^2}+\frac{y^2}{3^2}+\frac{z^2}{4^2}=1,\quad 0\le z\le 3.5$
3	B-сплайн	Однополосный гиперболоид – Безье $\frac{x^2}{2^2}+\frac{y^2}{2^2}-\frac{z^2}{3^2}=1,\quad -3\le z\le 3$
4	Безье	Однополосный гиперболоид – B-сплайн $\frac{x^2}{2^2}+\frac{y^2}{2^2}-\frac{z^2}{3^2}=1,\quad -3\le z\le 3$
5	B-сплайн	Двухполосный гиперболоид – Безье $\frac{x^2}{3^2}+\frac{y^2}{3^2}-\frac{z^2}{1^2}=-1,\ -3\le z\le -1.5,\ 1.5\le z\le 3$
6	Безье	Двухполосный гиперболоид – B-сплайн $\frac{x^2}{3^2}+\frac{y^2}{3^2}-\frac{z^2}{1^2}=-1,\ -3\le z\le -1.5,\ 1.5\le z\le 3$
7	B-сплайн	Эллиптический параболоид – Безье $x^2+2y^2=z,\quad 1\le z\le 4$
8	Безье	Эллиптический параболоид – B-сплайн $x^2+2y^2=z,\quad 1\le z\le 4$
9	B-сплайн	Гиперболический параболоид – Безье $x^2-y^2=z,\ -2\le z\le 2,\ -2\le y\le 2$
10	Безье	Гиперболический параболоид – B-сплайн $x^2-y^2=z,\ -2\le z\le 2,\ -2\le y\le 2$

Продолжение таблицы 4.1

11	B-сплайн	Эллиптический цилиндр – Безье $\frac{x^2}{2^2}+\frac{y^2}{3^2}=1,\ 0\le z\le 3$
12	Безье	Эллиптический цилиндр – B-сплайн $\frac{x^2}{2^2}+\frac{y^2}{3^2}=1,\ 0\le z\le 3$
13	B-сплайн	Гиперболический цилиндр – Безье $\frac{x^2}{1^2}-\frac{y^2}{2^2}=1,\ -2\le z\le 2,\ -2\le x\le 1,\ 1\le x\le 2$
14	Безье	Гиперболический цилиндр – B-сплайн $\frac{x^2}{1^2}-\frac{y^2}{2^2}=1,\ -2\le z\le 2,\ -2\le x\le 1,\ 1\le x\le 2$
15	B-сплайн	Эллипсоид – Безье $\frac{x^2}{2^2}+\frac{y^2}{3^2}+\frac{z^2}{4^2}=1,\quad -2\le y\le 2$
16	Безье	Эллипсоид – B-сплайн $\frac{x^2}{2^2}+\frac{y^2}{3^2}+\frac{z^2}{4^2}=1,\quad -2\le y\le 2$
17	B-сплайн	Однополосный гиперболоид – Безье $\frac{x^2}{2^2}+\frac{y^2}{2^2}-\frac{z^2}{3^2}=1,\ -1\le z\le 4,\ x\ge 0$
18	Безье	Однополосный гиперболоид – B-сплайн $\frac{x^2}{2^2}+\frac{y^2}{2^2}-\frac{z^2}{3^2}=1,\ -1\le z\le 4,\ x\ge 0$
19	B-сплайн	Двухполосный гиперболоид – Безье $\frac{x^2}{3^2}+\frac{y^2}{3^2}-\frac{z^2}{1^2}=-1,\ 1.5\le z\le 3,\ x\ge 0$
20	Безье	Двухполосный гиперболоид – B-сплайн $\frac{x^2}{3^2}+\frac{y^2}{3^2}-\frac{z^2}{1^2}=-1,\ 1.5\le z\le 3,\ x\ge 0$
21	B-сплайн	Эллиптический параболоид – Безье $2x^2+y^2=z,\ 1\le z\le 5,\ x\ge 0,\ y\ge 0$

Продолжение таблицы 4.1

22	Безье	Эллиптический параболоид – B-сплайн $2x^2 + y^2 = z,\ 1 \le z \le 5,\ x \ge 0,\ y \ge 0$
23	B-сплайн	Эллипсоид – Безье $\frac{x^2}{2^2} + \frac{y^2}{3^2} + \frac{z^2}{4^2} = 1,\ -3.5 \le z \le 0,\ x \ge 0,\ y \ge 0$
24	Безье	Эллипсоид – B-сплайн $\frac{x^2}{2^2} + \frac{y^2}{3^2} + \frac{z^2}{4^2} = 1,\ -3.5 \le z \le 0,\ x \ge 0,\ y \ge 0$
25	B-сплайн	Однополосный гиперболоид – Безье $\frac{x^2}{2^2} + \frac{y^2}{2^2} - \frac{z^2}{3^2} = 1,\ -4 \le z \le 1,\ x \ge 0,\ y \ge 0$
26	Безье	Однополосный гиперболоид – B-сплайн $\frac{x^2}{2^2} + \frac{y^2}{2^2} - \frac{z^2}{3^2} = 1,\ -4 \le z \le 1,\ x \ge 0,\ y \ge 0$

Методические указания

1 Запустите среду программирования Visual Studio 2010 Professional.
2 Создайте проект «Windows Forms Application Visual C++».
3 Формализуйте алгоритм решения задачи на ПЭВМ.
4 Выведите на экран видеомонитора результаты работы программы.
5 Оформите отчет по практической работе.

Контрольные вопросы

1 Что называется сплайном?
2 Какие существуют свойства сплайновых кривых?
3 Какой существует алгоритм метода составления линии из отдельных сегментов?
4 Какое уравнение описывает интерполяционную кривую Catmull-Rom?
5 Какое уравнение описывает элементарную бета-сплайновую кривую?
6 Какое уравнение описывает сплайновую кривую Безье?
7 Что называется сплайновой поверхностью?
8 Какой вид имеет параметрическая форма записи сплайновой поверхности?
9 Какой вид имеет аналитическая форма записи бикубической сплайновой поверхности Безье?
10 Какой вид имеет матричная форма записи бикубической сплайновой поверхности Безье?

5 Растровые алгоритмы

Растровые алгоритмы – алгоритмы растеризации и алгоритмы обработки растровых изображений.

Алгоритмы растеризации включают:

- алгоритмы перевода графических примитивов в растровую форму;
- алгоритмы заполнения фигур.

Алгоритмы обработки растровых изображений:

- регулировка яркости и контрастности;
- масштабирование изображений;
- геометрические преобразования;
- алгоритмы фильтрации.

Растеризация – создание растрового изображения на основе векторного (или другого) описания элементов изображения.

Связность – соединение двух пикселей растровой линией (последовательным набором пикселей).

При переводе объектов в растровое представление разработаны алгоритмы, использующие четырёхсвязность и восьмисвязность [10].

5.1 Алгоритмы вывода прямой линии

Растеризация отрезка – определение точек двумерного растра, которые необходимо закрасить, чтобы получить близкое приближение прямой линии между двумя заданными точками.

Алгоритмы растеризации отрезков прямой линии:

- простейший алгоритм растрирования отрезка;
- реккурентный алгоритм растрирования отрезка;
- алгоритм Брезенхема растеризации отрезка;
- алгоритм цифрового дифференциального анализатора.

Алгоритм Брезенхема включает этапы:

1 Определение координат начальной и конечной точек отрезка прямой линии.

2 Определение основной и вспомогательной осей Декартовой системы координат. Основная ось – ось с максимальной проекцией отрезка прямой линии. Вспомогательная ось – ось с минимальной проекцией отрезка прямой линии.

3 Перемещение по основной оси на один пиксель с проверкой ошибки ε по вспомогательной оси. Если $\varepsilon \le 0.5$, то изменение координаты точки по вспомогательной оси не происходит, иначе координата точки по вспомогательной оси увеличивается на 1.

4 Процесс итерационный пока координаты точек основной оси меньше-равно координаты конечной точки отрезка прямой линии.

5.2 Алгоритмы вывода кривых второго порядка

Кривые второго порядка – алгебраические кривые, определяемые уравнениями второго порядка:

$$a_{11}X^2 + a_{12}XY + a_{22}Y^2 + a_{13}X + a_{23}Y + a_{33} = 0. \quad (5.1)$$

Кривые второго порядка:

- окружность (равносторонний эллипс);
- эллипс;
- гипербола;
- парабола.

Окружность – кривая второго порядка, точки которой равноудалены от центра окружности на величину, равную радиусу окружности.

Уравнение окружности с центром в точке $C(x_0, y_0)$ и радиусом R имеет вид:

$$(x - x_0)^2 + (y - y_0)^2 = R^2, \quad (5.2)$$

где x_0 – абсцисса центра окружности;

y_0 – ордината центра окружности;

R – радиус окружности.

Эллипс – кривая второго порядка, сумма расстояний от точек до фокусов постоянна.

Уравнение эллипса с центром в точке O(0, 0) имеет вид:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1, \quad (5.3)$$

где a – большая полуось эллипса;

b – малая полуось эллипса.

Алгоритмы растрирования окружностей:

- первый простой алгоритм растровой развёртки окружности;
- второй простой алгоритм растровой развёртки окружности;
- алгоритм Харденбурга;
- алгоритм Брезенхема.

Первый простой алгоритм растровой развёртки окружности:

1 Значение координаты x изменяется с единичным шагом от 0 до R.

2 Значение координаты y рассчитывается по формуле:

$$y = \pm\sqrt{R^2 - x^2}. \quad (5.4)$$

Второй простой алгоритм растровой развёртки окружности:

1 Значение координаты x рассчитывается по формуле:

$$x = R * \cos\alpha . \quad (5.5)$$

2 Значение координаты y рассчитывается по формуле:

$$y = R * \sin\alpha . \quad (5.6)$$

3 Угол α изменяется от 0 до 90 градусов.

5.3 Алгоритмы вывода и закрашивания фигур

Фигура – плоский геометрический объект, состоящий из контура и точек внутренней области.

Графический вывод фигур включает:

- вывод контура;
- вывод точек заполнения.

Алгоритмы закрашивания:

- простейший алгоритм закрашивания;
- волновой алгоритм закрашивания;
- алгоритм закрашивания линиями.

5.4 Практическая работа №4. Алгоритмы растеризации

Цель: Получить теоретические знания и практические навыки в применении Visual C++ для создания приложения, формализующего алгоритмы растеризации отрезков прямых линий, кривых второго порядка и закрашивания фигур.

Используемые приемы и технологии: Visual C++ 2010 Professional, библиотеки графических интерфейсов Windows Forms и Microsoft Foundation Classes.

Ключевые термины: растровые алгоритмы, растеризация, связность, кривые второго порядка, фигура, закрашивание фигур.

5.5 Варианты заданий к выполнению практической работы №4

Разработайте визуальное приложение на языке Visual C++, формализующее, отображение отрезка наклонной прямой линии, кривую второго порядка и закрашенную фигуру.

Вариант 1. Простейший алгоритм растеризации отрезка, алгоритм генерации окружности Брезенхема, алгоритм закрашивания линиями правильного шестиугольника.

Вариант 2. Реккурентный алгоритм растеризации отрезка, алгоритм генерации окружности Харденбурга, простейший алгоритм закрашивания правильного пятиугольника.

Вариант 3. Алгоритм симметричного цифрового дифференциального анализатора растеризации отрезка, алгоритм генерации эллипса, алгоритм закрашивания линиями квадрата.

Вариант 4. Алгоритм несимметричного цифрового дифференциального анализатора растеризации отрезка, первый алгоритм растрирования окружности, простейший алгоритм закрашивания эллипса.

Вариант 5. Алгоритм Брезенхема растеризации отрезка, второй алгоритм растрирования окружности, алгоритм закрашивания линиями треугольника.

Вариант 6. Оптимизированный алгоритм Брезенхема растеризации отрезка, алгоритм генерации эллипса, алгоритм закрашивания линями окружности.

Вариант 7. Простейший алгоритм растеризации отрезка, алгоритм генерации окружности Харденбурга, простейший алгоритм закрашивания эллипса.

Вариант 8. Реккурентный алгоритм растеризации отрезка, алгоритм генерации эллипса, алгоритм закрашивания линиями треугольника.

Вариант 9. Алгоритм симметричного цифрового дифференциального анализатора растеризации отрезка, первый алгоритм растрирования окружности, алгоритм закрашивания линиями квадрата.

Вариант 10. Алгоритм несимметричного цифрового дифференциального анализатора растеризации отрезка, второй алгоритм растрирования окружности, алгоритм закрашивания линиями правильного шестиугольника.

Вариант 11. Алгоритм Брезенхема растеризации отрезка, алгоритм генерации эллипса, простейший алгоритм закрашивания правильного пятиугольника.

Вариант 12. Оптимизированный алгоритм Брезенхема растеризации отрезка, алгоритм генерации окружности Брезенхема, алгоритм закрашивания линиями прямоугольника.

Вариант 13. Простейший алгоритм растеризации отрезка, первый алгоритм растрирования окружности, простейший алгоритм закрашивания шестиугольника.

Вариант 14. Реккурентный алгоритм растеризации отрезка, второй алгоритм растрирования окружности, алгоритм закрашивания линиями окружности.

Вариант 15. Алгоритм симметричного цифрового дифференциального анализатора растеризации отрезка, алгоритм генерации эллипса, простейший алгоритм закрашивания ромба.

Вариант 16. Алгоритм несимметричного цифрового дифференциального анализатора растеризации отрезка, алгоритм генерации окружности Брезенхема, алгоритм закрашивания линиями параллелограмма.

Вариант 17. Алгоритм Брезенхема растеризации отрезка, алгоритм генерации окружности Харденбурга, простейший алгоритм закрашивания треугольника.

Вариант 18. Оптимизированный алгоритм Брезенхема растеризации отрезка, алгоритм генерации окружности Брезенхема, алгоритм закрашивания линиями эллипса.

Вариант 19. Простейший алгоритм растеризации отрезка, алгоритм генерации окружности Брезенхема, простейший алгоритм закрашивания пятиугольника.

Вариант 20. Реккурентный алгоритм растеризации отрезка, первый алгоритм растрирования окружности, алгоритм закрашивания линиями окружности.

Вариант 21. Алгоритм симметричного цифрового дифференциального анализатора растеризации отрезка, алгоритм генерации эллипса, простейший алгоритм закрашивания шестиугольника.

Вариант 22. Алгоритм несимметричного цифрового дифференциального анализатора растеризации отрезка, алгоритм генерации окружности Брезенхема, алгоритм закрашивания линиями квадрата.

Вариант 23. Алгоритм Брезенхема растеризации отрезка, алгоритм генерации окружности Харденбурга, простейший алгоритм закрашивания эллипса.

Вариант 24. Оптимизированный алгоритм Брезенхема растеризации отрезка, алгоритм генерации эллипса, алгоритм закрашивания линиями ромба.

Вариант 25. Простейший алгоритм растеризации отрезка, первый алгоритм растрирования окружности, простейший алгоритм закрашивания прямоугольника.

Методические указания

1 Запустите среду программирования Visual Studio 2010 Professional.

2 Создайте проект «Windows Forms Application Visual C++».

3 Формализуйте алгоритм решения задачи на ПЭВМ.

4 Выведите на экран видеомонитора результаты работы программы.

5 Оформите отчет по практической работе.

Контрольные вопросы

1 Что называется растеризацией?

2 Что понимается под связностью?

3 Какой используется простейший алгоритм растеризации отрезка?

4 Какой используется реккурентный алгоритм растеризации отрезка?

5 Какой используется алгоритм цифрового дифференциального анализатора?

6 Какой применяется алгоритм Брезенхема растеризации отрезка?

7 В чём заключается преимущество алгоритма Брезенхема?

8 Какой применяется алгоритм Брезенхема генерации окружности?

9 Какой применяется первый алгоритм растрирования окружности?

10 Какой применяется второй алгоритм растрирования окружности?

11 Какой применяется алгоритм генерации окружности Харденбурга?

12 Что называется окружностью?

13 Что называется эллипсом?

14 Какое математическое выражение в общем виде описывает кривые второго порядка?

15 Какое применяется свойство окружности и эллипса?

16 Какую часть окружности необходимо рассчитать?

17 Какую часть эллипса необходимо рассчитать?

18 Какой применяется алгоритм вывода фигур?

19 Какой применяется простой алгоритм закрашивания фигур?

20 Какой применяется алгоритм закрашивания фигур линиями?

6 Отсечение отрезков и поверхностей

Алгоритмы удаления невидимых линий и поверхностей служат для определения линии рёбер, поверхностей или объёмов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке.

Сущность алгоритмов удаления невидимых линий и поверхностей заключается в следующем: чем дальше расположен объект от точки наблюдения, тем больше вероятность, что он будет полностью или частично заслонен другим, более близким к наблюдателю, объектом. После определения расстояний или приоритетов по глубине проводится сортировка по горизонтали и по вертикали, для выяснения, будет ли рассматриваемый объект действительно заслонен объектом, расположенным ближе к точке наблюдения.

Выделяют три группы алгоритмов:

1 Алгоритмы, работающие в пространстве объекта:

- алгоритм Робертса.

Для определения видимости данной поверхности сравнивается ее взаимное расположение с остальными поверхностями объекта в трехмерной сцене.

2 Алгоритмы, работающие в пространстве изображения (экрана):

- алгоритм плавающего горизонта;
- алгоритм *Коэна-Сазерленда*;
- модифицированный вариант алгоритма *Коэна-Сазарленда*;
- алгоритм с использованием *z-буфера*;
- метод трассировки лучей;
- алгоритм Варнока;
- алгоритм *Вейлера-Азертона*.

Они основаны на нахождении точки ближайшей поверхности, которую пересекает луч зрения, проходящий через заданную точку на растре.

3 Алгоритмы, формирующие список приоритетов:

- алгоритм *Ньюэла-Санча*.

Алгоритмы работают попеременно в обеих системах координат (объекта и изображения).

6.1 Алгоритмы удаления невидимых линий

Алгоритмы удаления невидимых линий:

1 Алгоритм плавающего горизонта.

2 Алгоритм Робертса.

Алгоритм плавающего горизонта применяется для удаления невидимых линий. Сущность метода заключается в сведении трехмерной задачи к двумерной посредством пересечения исходной поверхности последовательностью параллельных секущих плоскостей, имеющих постоянные значения координаты *z*.

Алгоритм сначала упорядочивает плоскости $z = const$ по возрастанию расстояния до них от точки наблюдения. Затем для каждой плоскости, начиная с ближайшей к точке наблюдения, строится кривая, лежащая на ней, т.е. для каждого значения координаты *x* в пространстве изображения определяется соответствующее значение *y*.

Если на текущей плоскости при заданном значении *x* соответствующее значение *y* на кривой больше значения *y* для всех предыдущих кривых при этом значении *x*, то текущая кривая видима в этой точке; иначе она невидима.

6.2 Алгоритмы удаления невидимых поверхностей

Алгоритмы удаления невидимых поверхностей:

1 Алгоритм Вейлера-Азертона.

2 Алгоритм Варнока.

3 Алгоритм Z-буфера.

Алгоритм Вейлера-Азертона включает этапы:

1 Предварительная сортировка по глубине.

2 Отсечение по границе ближайшего к наблюдателю многогранника (сортировка многоугольников на плоскости).

3 Удаление многоугольников, экранированных многоугольником, ближайшим к точке наблюдения.

4 Если требуется, то рекурсивное разбиение и окончательная сортировка для устранения всех неопределённостей.

6.3 Практическая работа №5. Методы удаления невидимых линий и поверхностей

Цель: Получить теоретические знания и практические навыки в применении Visual C++ для создания приложения, формализующего алгоритмы удаления невидимых линий и поверхностей.

Используемые приемы и технологии: Visual C++ 2010 Professional, библиотеки графических интерфейсов Windows Forms и Microsoft Foundation Classes.

Ключевые термины: невидимая линия, невидимая поверхность, алгоритм Вейлера-Азертона, алгоритм Варнока, алгоритм Z-буфера, алгоритм плавающего горизонта, алгоритм Робертса.

6.4 Варианты заданий к выполнению практической работы №5

Разработайте визуальное приложение на языке Visual C++, формализующее, отображение фигуры в режиме аксонометрического (или другого) проецирования с удалёнными невидимыми рёбрами и гранями.

Используя средства управления, расположенные на панели, реализуйте в приложении:

1 Изменение масштаба фигуры.

2 Изменение положения (перенос) фигуры с применением кнопок клавиатуры.

3 Вращение фигуры в двух плоскостях при нажатии кнопок <Влево>, <Вправо>, <Вверх>, <Вниз>, расположенных на панели управления окна проекта, а также при движении мышки в горизонтальном или вертикальном направлениях (при нажатой левой кнопки мышки).

4 Изменение скорости вращения фигуры при нажатии на заданную кнопку клавиатуры.

5 Включение/выключение закрашивания поверхностей фигуры разными цветами.

Варианты заданий приведены в таблице 6.1.

Таблица 6.1 – Варианты заданий

Номер варианта	**Тип фигуры**	**Алгоритм удаления**
1	Прямоугольный параллелепипед	Алгоритм плавающего горизонта
2	Косоугольный параллелепипед	Алгоритм Робертса
3	Треугольная пирамида (тетраэдр)	Алгоритм Варнока
4	Четырёхугольная пирамида	Алгоритм Вейлера-Азертона
5	Усечённая четырёхугольная пирамида	Z-буфер
6	Треугольная призма	Алгоритм плавающего горизонта
7	Шестиугольная призма	Алгоритм Робертса
8	Гексаэдр	Алгоритм Варнока
9	Октаэдр	Алгоритм Вейлера-Азертона
10	Шестиугольная пирамида	Z-буфер
11	Додекаэдр	Алгоритм плавающего горизонта

Продолжение таблицы 6.1

12	Икосаэдр	Алгоритм Робертса
13	Усечённая шестиугольная пирамида	Алгоритм Варнока
14	Пятиугольная призма	Алгоритм Вейлера-Азертона
15	Пятиугольная пирамида	Z-буфер
16	Усечённая треугольная пирамида	Алгоритм плавающего горизонта
17	Усечённая шестиугольная пирамида	Алгоритм Робертса
18	Усечённая пятиугольная пирамида	Алгоритм Варнока
19	Наклонный параллелепипед	Алгоритм Вейлера-Азертона
20	Наклонная треугольная призма	Z-буфер
21	Наклонная четырёхугольная пирамида	Алгоритм плавающего горизонта
22	Восьмиугольная призма	Алгоритм Робертса
23	Восьмиугольная пирамида	Алгоритм Варнока
24	Восьмиугольная усечённая пирамида	Алгоритм Вейлера-Азертона
25	Наклонная шестиугольная пирамида	Z-буфер

Методические указания

1 Запустите среду программирования Visual Studio 2010 Professional.
2 Создайте проект «Windows Forms Application Visual C++».
3 Формализуйте алгоритм решения задачи на ПЭВМ.
4 Выведите на экран видеомонитора результаты работы программы.
5 Оформите отчет по практической работе.

Контрольные вопросы

1 С какой целью производится удаление невидимых линий и поверхностей?
2 В чём заключается сущность алгоритмов удаления невидимых линий и поверхностей?
3 Сколько и какие применяются группы алгоритмов удаления невидимых линий и поверхностей?
4 Какие применяются алгоритмы удаления невидимых линий?
5 Какие применяются алгоритмы удаления невидимых поверхностей?
6 Какие применяются этапы алгоритма Вейлера-Азертона?
7 Какие применяются этапы алгоритма плавающего горизонта?
8 Какие применяются этапы алгоритма Робертса?

9 Какие применяются этапы алгоритма Варнока?
10 Какие применяются этапы алгоритма Z-буфера?

7 Фракталы

Фрактал – геометрическая фигура, обладающая свойством самоподобия.

Латинское слово *fractus* означает «составленный из фрагментов». Самоподобие – свойство фрактала, при котором отдельные части фрактала похожи по форме на весь фрактал в целом.

7.1 Алгебраические фракталы

Алгебраический фрактал – фрактал, созданный на основе математического выражения. Итерационная формула фрактала Мандельброта имеет вид:

$$z_{k+1} = z_k^2 + z_0, \tag{7.1}$$

где z_k – комплексные числа, $k = \overline{0,n}$.

Программный код фрактала Мандельброта приведён в листинге 7.1. Изображение фрактала Мандельброта приведено на рисунке 7.1

Формула итераций для фрактала Джулия имеет вид:

$$z_{k+1} = z_k^2 + c, \tag{7.2}$$

где z_k – комплексные числа, $k = \overline{0,n}$.

c – комплексная константа.

Изображение фрактала Джулия приведено на рисунке 7.2

Итерационная формула фрактала Ньютона имеет вид:

$$z_{k+1} = \frac{3z_k^4 + 1}{4z_k^3}, \tag{7.3}$$

где z_k – комплексные числа, $k = \overline{0,n}$.

Изображение фрактала Ньютона приведено на рисунке 7.3.

Программный код фрактала Мандельброта приведён в листинге 7.1.

Листинг 7.1 Программный код фрактала Мандельброта

```
//Определение констант
#define CImMAX 1.25 //с увеличением значения рисунок смещается по
//вертикали вниз
#define CImMIN -1.25 //с уменьшением значения рисунок смещается по
```

```
//вертикали вверх
#define CReMAX 1.25 //с увеличением значения рисунок смещается влево
#define CReMIN -2.0 //с уменьшением значения рисунок смещается вправо
#define MAX_ITERATION 128 //количество итераций
//Таблица цветов
DWORD ColorTable [6] =
{0x0000ff,  //красный – RGB(255,0,0)
0x00ff00,  //зеленый – RGB(0,255,0)
0xff0000,  //синий – RGB(0,0,255)
0x00ffff,  //желтый – RGB(255,255,0)
0xffff00,  //бирюзовый – RGB(0,255,255)
0xff00ff,   //сиреневый – RGB(255,0,255)
    }; //Функция рисования фрактала
void CMandelView::DrawCol(void)
{ CClientDC ClientDC (this);
int Iteration; //номер текущей итерации
float Im;      //значение мнимой части комплексной переменной
float ImSqr;   //квадрат мнимой части комплексной переменной
float ReSqr;   //квадрат действительной части комплексной переменной
float Re;      //значение действительной части комплексной переменной
int Row;  //текущая строка рисунка - номер позиции в столбце пикселов
//если заполнен последний столбец или окно минимизировано, то выход
if (m_Col >= m_ColMax || GetParentFrame ()->IsIconic ())
return;
m_CIm = CImMAX;  //текущее значение мнимой части комплексной
//переменной
for (Row = 0; Row < m_RowMax; ++Row) //проход по всему столбцу
{ Re =0.0; Im = 0.0; ReSqr = 0.0; ImSqr = 0.0; Iteration = 0;
//пока не закончились все итерации и точка не вышла из круга радиуса 2
while (Iteration < MAX_ITERATION && ReSqr + ImSqr < 4)
{ ++Iteration;      //перейти к следующей итерации
ReSqr = Re * Re;  //вычислить составляющие произведения
//двух комплексных переменных
ImSqr = Im * Im; Im = 2 * Im * Re + m_CIm;
Re = ReSqr - ImSqr + m_CRe;} //отобразить пиксел заданным цветом
ClientDC.SetPixelV (m_Col, Row, ColorTable [Iteration % 6]);
m_CIm -= m_DCIm; //прирастить мнимую часть комплексной переменной
} m_Col++; //перейти к следующему столбцу
m_CRe += m_DCRe; //прирастить действительную часть комплексной
//переменной
}
```

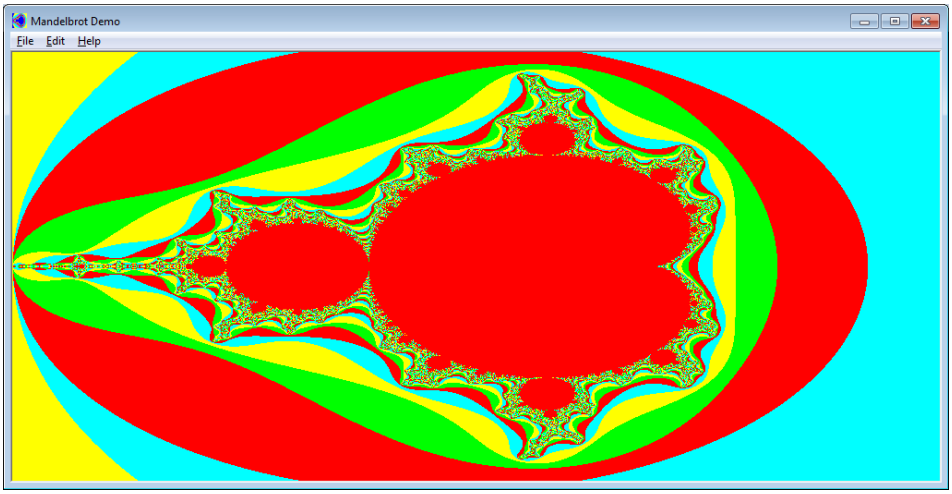

Рисунок 7.1 – Фрактал Мандельброта

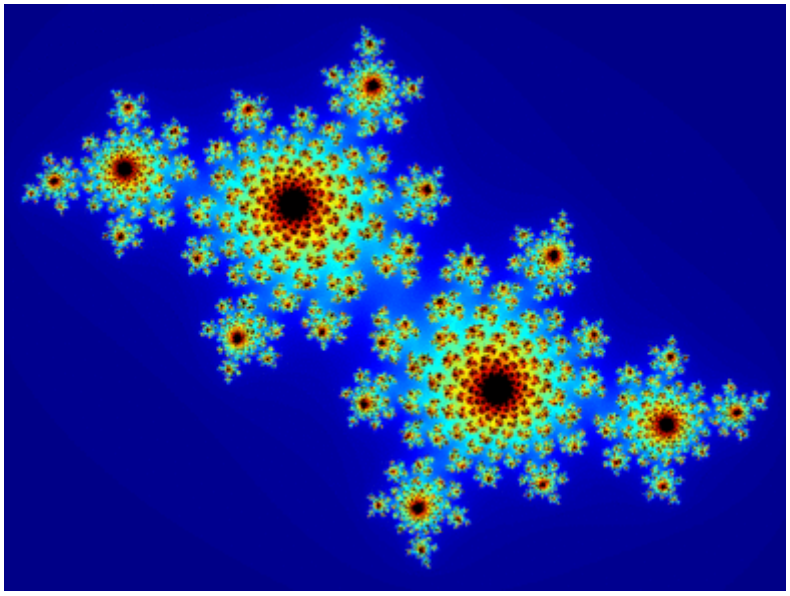

Рисунок 7.2 – Фрактал Джулия

7.2 Геометрические фракталы

Геометрический фрактал – фрактал, форма которого описывается последовательностью простых геометрических операций. Кривая Коха становится фракталом в результате бесконечного количества итераций, в ходе которых выполняется деление отрезка прямой на три части [3].

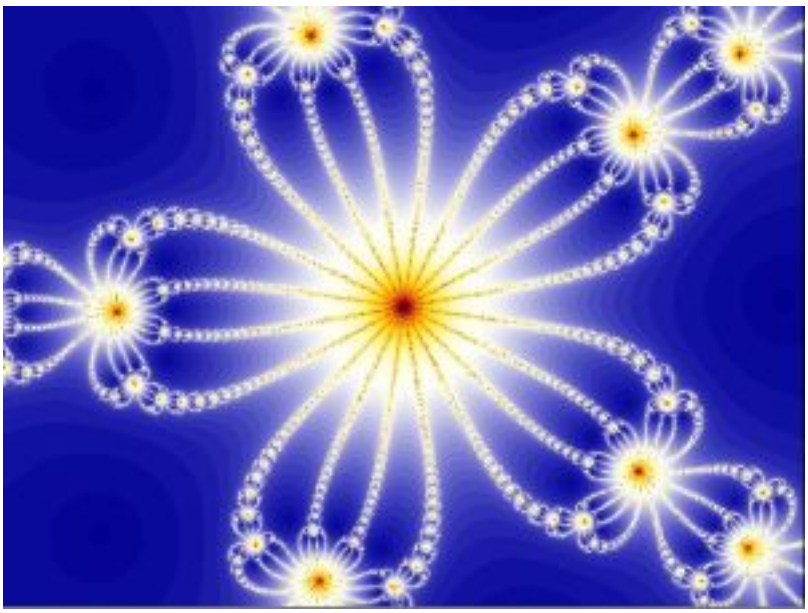

Рисунок 7.3 – Фрактал Ньютона

7.3 Фракталы IFS

Фракталы IFS – фракталы, которые генерируются согласно методу систем итеративных функций (Iterated Fuctions Systems, IFS). Метод систем итеративных функций описывается как последовательный итеративный расчёт координат новых точек в пространстве по математическим выражениям.

$$x_{k+1} = F_x(x_k, y_k),\ y_{k+1} = F_y(x_k, y_k), \tag{7.4}$$

где F_x, F_y – функции преобразования координат [3].

Фракталы IFS получили широкое распространение благодаря работам Майкла Барнсли из технологического института штата Джорджия. Он кодировал изображения с помощью фракталов. Запатентовав несколько идей по кодированию изображений с помощью фракталов, он основал фирму «Iterated Systems», которая выпустила программный продукт «Images Incorporated», позволяющий переводить изображения из растровой формы в фрактальную.

7.4 Стохастические фракталы

Стохастические фракталы – фракталы, получающиеся в результате выполнения итерационного процесса со случайным образом изменяющимися параметрами. Полученные объекты похоже на природные – несимметричные деревья, изрезанные береговые линии и т. д. Двумерные стохастические фракталы используются при моделировании рельефа местности и поверхности моря.

7.5 L-системы

L-система основывается на двух принципах:

- «черепашья графика» (оператор draw) GWBASIC, Turbo Basic, QBasic при прорисовке движения пошагово в приращениях относительно текущей точки, или моделируется поведение, задавая движение в приращениях координат;
- единичное движение заменяется на весь рисунок.

L-системы кодируются в общепринятых обозначениях:

- движение вперед обозначается буквой F (forward (англ.) – вперед);
- поворот по часовой стрелке обозначается «+»;
- поворот против часовой стрелки обозначается «-»;
- значение поворота задается в программе и постоянно для всех движений;
- возврат без прорисовки обозначается буквой B (back (англ.) – назад) [3, 11].

7.6 Практическая работа №6. Фракталы

Цель работы: Получить теоретические знания и практические навыки в применении Microsoft Visual C++ 2010 для формализации приложения, создающего фракталы.

Используемые приемы и технологии: Visual C++ 2010 Professional, библиотеки графических интерфейсов Windows Forms и Microsoft Foundation Classes.

Ключевые термины: фрактал, свойство самоподобия, комплексное число, система итеративных функций, L-система.

7.6.1 Варианты заданий к практической работе №6

Разработайте визуальное приложение на языке Visual C++, формализующее отображение фракталов.

Вариант 1. Фрактал «Кривая Коха». Аксиома: F. Правило: F → F-F++F-F. Угол: $\frac{\pi}{3}$. Кривая Коха (рисунок 7.4).

Рисунок 7.4 – Фрактал Кривая Коха

Вариант 2. Фрактал «Снежинка Коха». Аксиома: F++F++F. Правило: F → F-F++F-F. Угол: $\frac{\pi}{3}$ Снежинка Коха (рисунок 7.5).

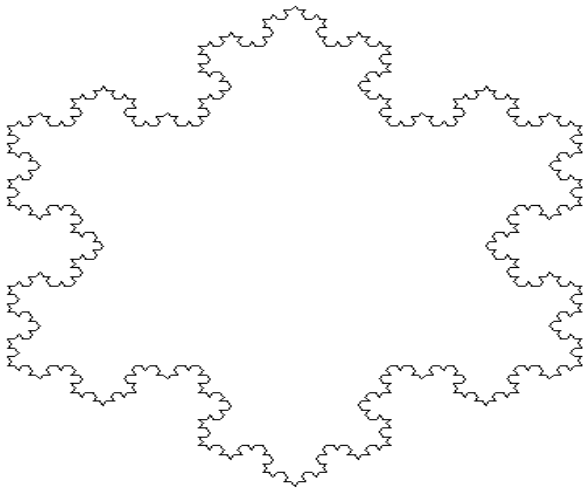

Рисунок 7.5 – Фракталы Снежинка Коха

Вариант 3. Фрактал «Ледяные фракталы». Аксиома: F+F+F+F. Правило: F → FF+F++F+F. Угол: $\frac{\pi}{2}$. Ледяные фракталы (рисунок 7.6)

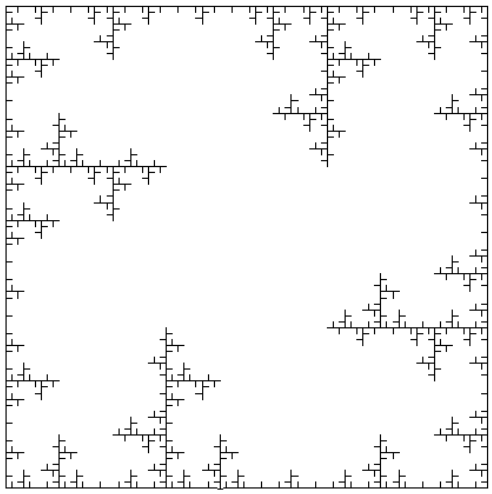

Рисунок 7.6 – Фрактал Ледяные узоры

Вариант 4. Фрактал «Кривая дракона». Аксиома: FX. Правила: X → X+YF+. Y → -FX-Y Угол: $\frac{\pi}{2}$. Кривая дракона (рисунок 7.7).

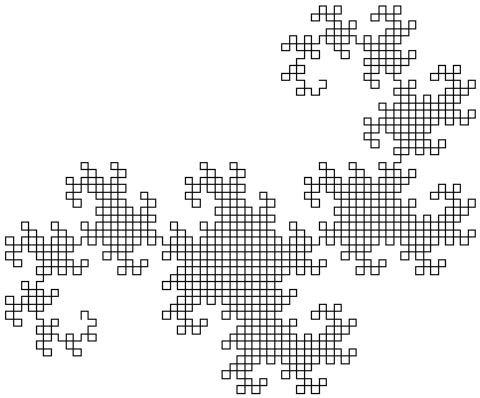

Рисунок 7.7 – Фрактал Кривая дракона

Вариант 5. Фрактал «Кривая Госпера».
Аксиома: XF. Правила: X → X+YF++YF-FX--FXFX-YF+,
Y → -FX+YFYF++YF+FX--FX-Y. Угол: $\frac{\pi}{3}$. Кривая Госпера (рисунок 7.8).

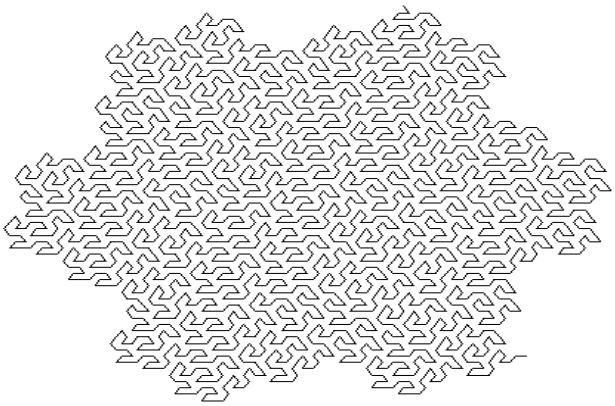

Рисунок 7.8 – Фрактал Кривая Госпера

Вариант 6. Фрактал «Кривая Серпинского». Аксиома: F+XF+F+XF.
Правило: X → XF-F+F-XF+F+XF-F+F-X. Угол: $\frac{\pi}{2}$ (рисунок 7.9).

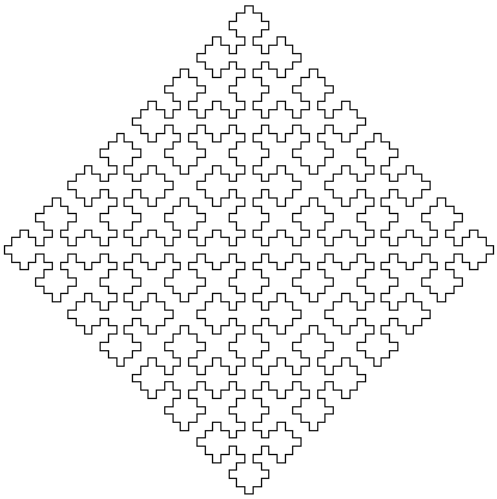

Рисунок 7.9 – Фрактал Кривая Серпинского

Вариант 7. Фрактал «Треугольник Серпинского». Аксиома: FXF--FF—FF. Правила: F → FF, X → --FXF++FXF++FXF--. Угол: $\frac{\pi}{3}$. Треугольник Серпинского (рисунок 7.10).

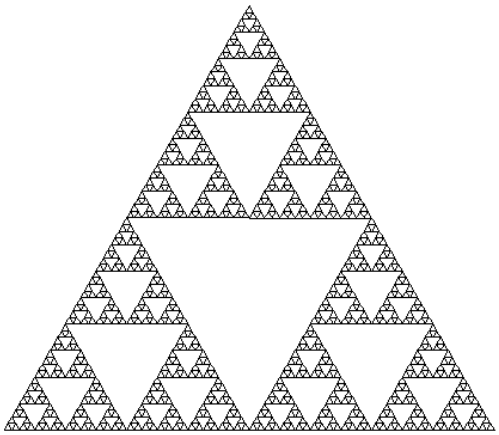

Рисунок 7.10 – Фрактал Треугольник Серпинского

Вариант 8. Фрактал «Ковёр Серпинского». Аксиома: F.

Правило: F → FFF[+FFF+FFF+FFF]. Угол: $\frac{\pi}{2}$. Ковёр Серпинского (рисунок 7.11).

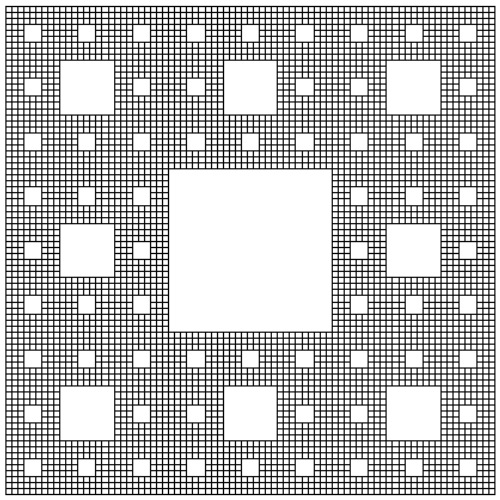

Рисунок 7.11 – Фрактал Ковёр Серпинского

Вариант 9. Фрактал «Кривая Леви». Аксиома: F++F++F++F. Правило: F → -F++F-. Угол: $\frac{\pi}{4}$. Кривая Леви (рисунок 7.12).

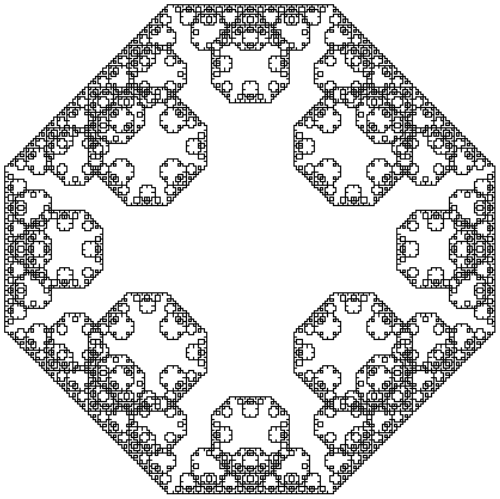

Рисунок 7.12 – Фрактал Кривая Леви

Вариант 10. Фрактал «Pentigree». Аксиома: F-F-F-F-F
Правило: F → F-F++F+F-F-F. Угол: $\frac{2\pi}{5}$ (рисунок 7.13).

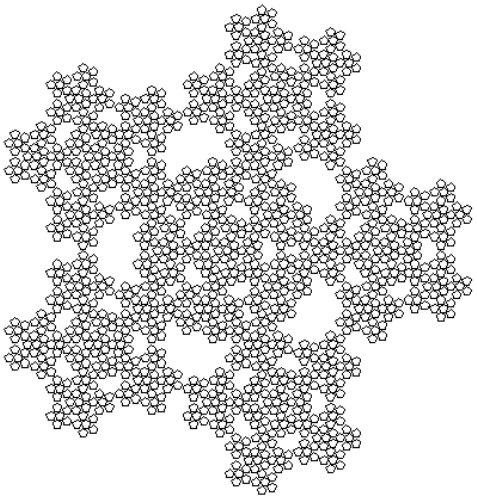

Рисунок 7.13 – Фрактал Pentigree

Вариант 11. Фрактал «Обобщения кривой Коха». Аксиома: F+F+F+F. Правило: F → F+F-F-FF+F+F-F.

Угол: $\frac{\pi}{2}$. Обобщения кривой Коха (рисунок 7.14).

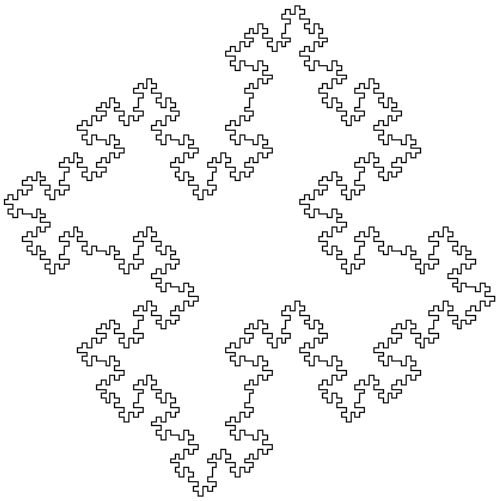

Рисунок 7.14 – Фрактал Обобщения кривой Коха

Вариант 12. Фрактал «Квадратичный остров Коха». Аксиома: F+F+F+F. Правило: F → F-FF+FF+F+F-F-FF+F+F-F-FF-FF+F. Угол: $\frac{\pi}{2}$.
Обобщения кривой Коха (рисунок 7.15).

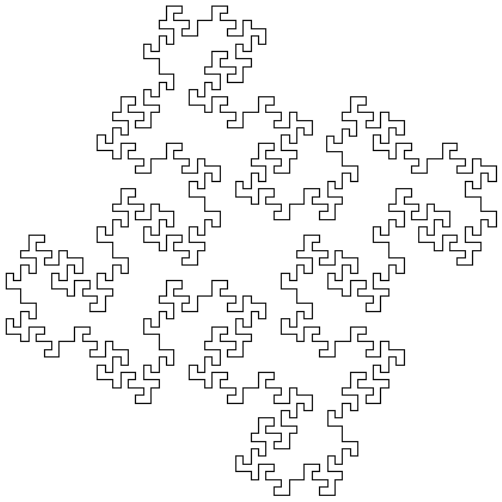

Рисунок 7.15 – Фрактал Квадратичный остров Коха

Вариант 13. Фрактал «Quadratic Snowflake». Аксиома: F.
Правило: F → F-F+F+F-F. Угол: $\frac{\pi}{2}$ (рисунок 7.16).

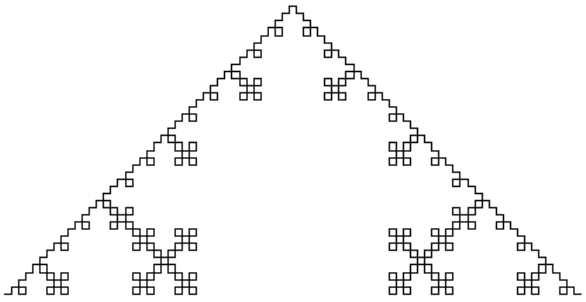

Рисунок 7.16 – Фрактал Quadratic Snowflake

Вариант 14. «Фрактал Sierpinski Arrowhead». Аксиома: YF.
Правила: X → YF+XF+Y. Y → XF-YF-X. Угол: $\frac{\pi}{3}$ (рисунок 7.17).

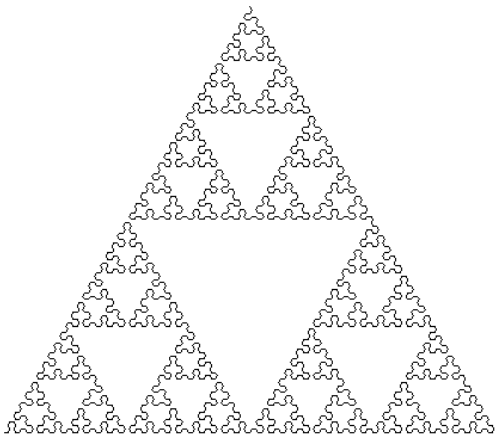

Рисунок 7.17 – Фрактал Sierpinski Arrowhead

Вариант 15. Фрактал «Board». Аксиома: F+F+F+F.
Правило: F → FF+F+F+F+FF. Угол: $\frac{\pi}{2}$ (рисунок 7.18).

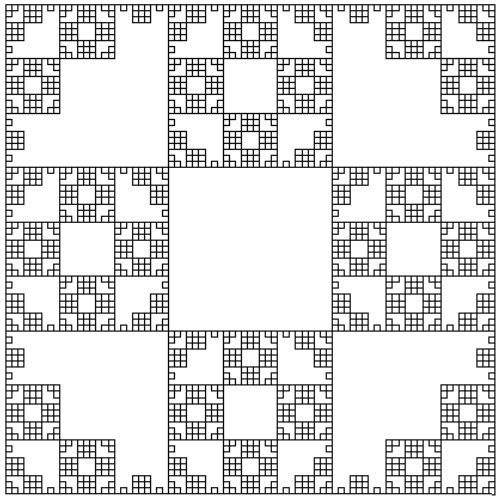

Рисунок 7.18 – Фрактал Board

Вариант 16. Фрактал «Rings». Аксиома: F+F+F+F.
Правило: F → FF+F+F+F+F+F-F. Угол: $\frac{\pi}{2}$ (рисунок 7.19).

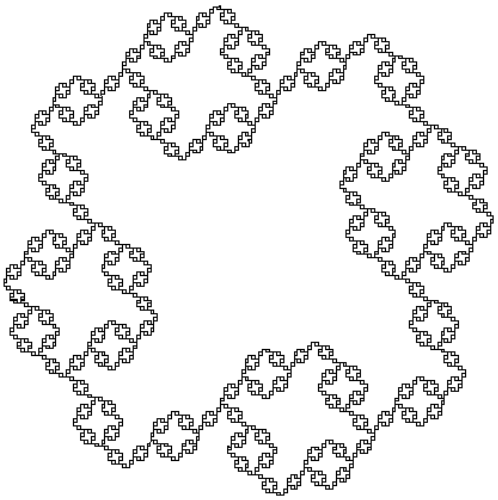

Рисунок 7.19 – Фрактал Rings

Вариант 17. Фрактал «Cross». Аксиома: F+F+F+F.
Правило: F → F+F-F+F+F. Угол: $\frac{\pi}{2}$ (рисунок 7.20).

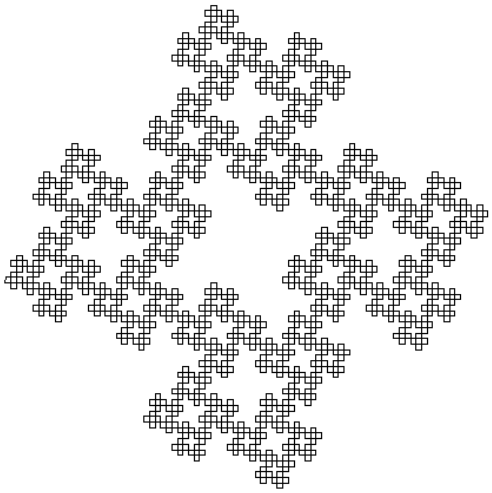

Рисунок 7.20 – Фрактал Cross

Вариант 18. Фрактал «Box Fractal». Аксиома: F-F-F-F.
Правило: F → F-F+F+F-F. Угол: $\frac{\pi}{2}$ (рисунок 7.21).

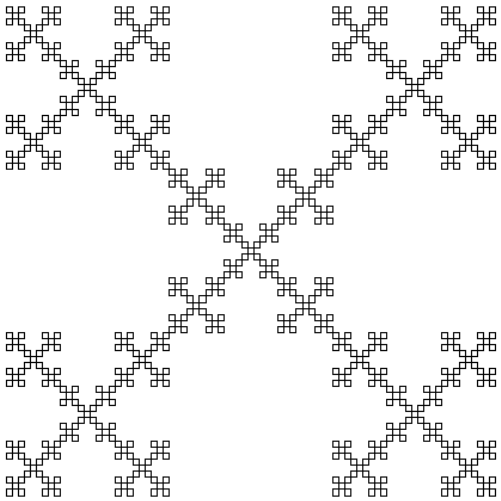

Рисунок 7.21 – Фрактал Box Fractal

Вариант 19. Фрактал «Pentaplexity». Аксиома: F++F++F++F++F. Правило: F → F++F++F+++++F-F++F. Угол: $\frac{\pi}{5}$ (рисунок 7.22).

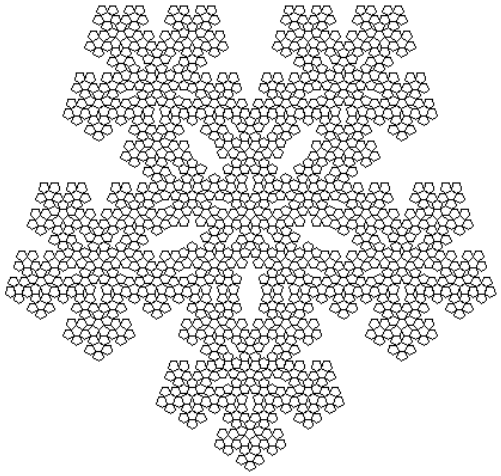

Рисунок 7.22 – Фрактал Pentaplexity

Вариант 20. Фрактал «Terdragon». Аксиома: F. Правило: F → F+F-F. Угол: $\frac{2\pi}{3}$ (рисунок 7.23).

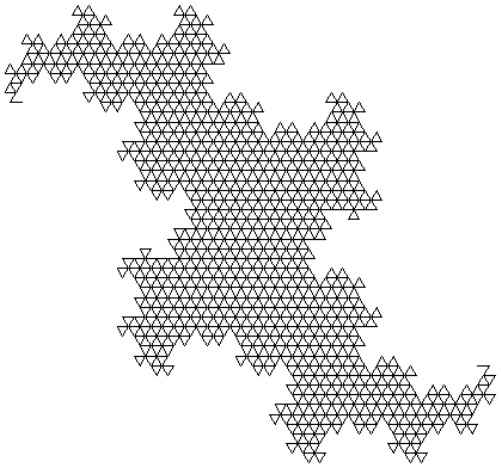

Рисунок 7.23 – Фрактал Terdragon

Вариант 21. Фрактал «Hex-7-b». Аксиома: X. Правила:
F → X → -F++F-X-F--F+Y---F--F+Y+F++F-X+++F++F-X-F++F-X+++F--F+Y--Y → +F++F-X-F--F+Y+F--F+Y---F--F+Y---F++F-X+++F++F-X+++F--F+Y
Угол: $\frac{\pi}{6}$ (рисунок 7.24).

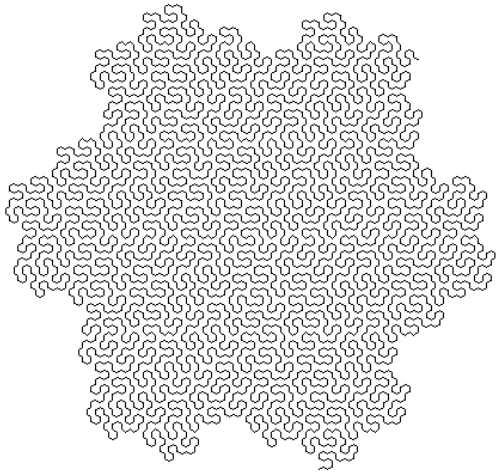

Рисунок 7.24 – Фрактал Hex-7-b

Вариант 22. Фрактал «Peano-c». Аксиома: FX. Правила: F →
X → FX-FY-FX+FY+FX+FY+FX+FY+FX-FY-FX-FY-FX-FY-FX+FY+FX
Y → FY. Угол: $\frac{\pi}{4}$ (рисунок 7.25).

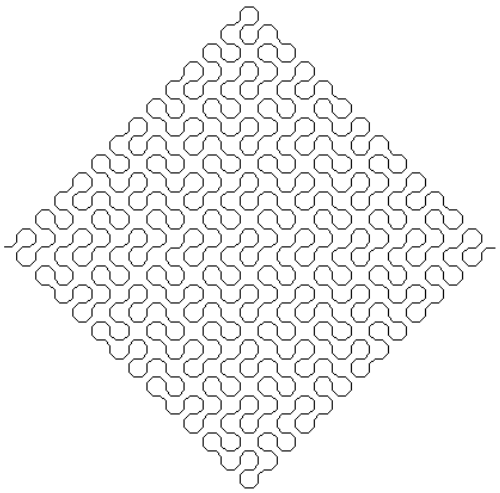

Рисунок 7.25 – Фрактал Peano-c

Вариант 23. Фрактал «Maze01». Аксиома: F+F+F. Правило: F → F+FF-F. Угол: $\frac{2\pi}{3}$ (рисунок 7.26).

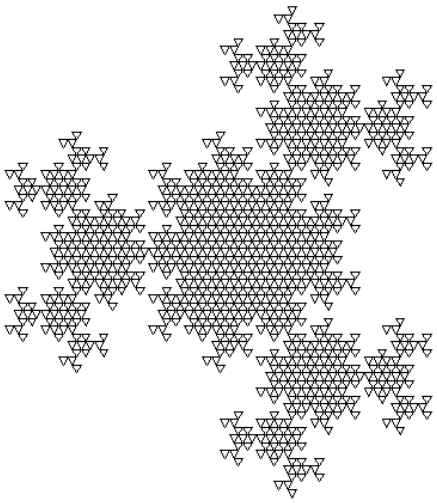

Рисунок 7.26 – Фрактал Maze01

Вариант 24. Фрактал «Tiling1». Аксиома: X.
Правила: X → F-F-F+F+FX++F-F-F+F+FX--F-F-F+F+FX
F → . Угол: $\frac{\pi}{3}$ (рисунок 7.27).

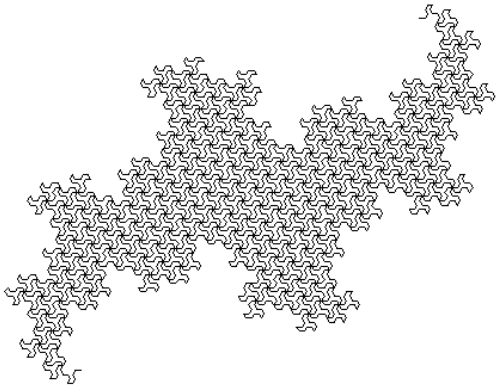

Рисунок 7.27 – Фрактал Tiling1

Вариант 25. Фрактал «Кривая Гилберта». Аксиома: X. Правила:
X → -YF+XFX+FY-
Y → +XF-YFY-FX+
Угол: $\frac{\pi}{2}$ Кривая Гильберта (рисунок 7.28).

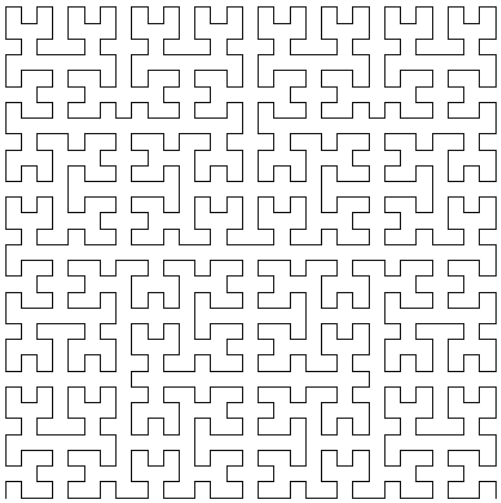

Рисунок 7.28 – Фрактал Кривая Гилберта

Методические указания

1 Запустите среду программирования Visual Studio 2010 Professional.
2 Создайте проект «Windows Forms Application Visual C++».
3 Формализуйте алгоритм решения задачи на ПЭВМ.
4 Выведите на экран видеомонитора результаты работы программы.
5 Оформите отчет по практической работе.

Контрольные вопросы

1 Что называется фракталом?

2 Какие существуют свойства фракталов?

3 Что называется алгебраическим фракталом?

4 Какое математическое выражение используется для создания фрактала «Множество Мандельброта»?

5 Какое используется условие прекращения выполнения цикла итераций для каждой точки изображения фрактала «Множество Мандельброта»?

6 Какое математическое выражение используется для создания фрактала «Джулия»?

7 Какое математическое выражение используется для создания фрактала «Ньютона»?

8 Какое используется условие прекращения выполнения цикла итераций для каждой точки изображения фрактала «Ньютона»?

9 Что называется геометрическим фракталом?

10 Что называется стохастическим фракталом?

11 Что называется фракталом IFS?

12 Что понимается под L-системой?

8 Визуализация трёхмерных объектов. Методы закрашивания поверхностей

8.1 Визуализация объёмных поверхностей

Визуализация трёхмерных изображений – создание объёмных изображений. Способы визуализации трёхмерных изображений подразделяются на виды:

- каркасная модель;
- изображение поверхностей в виде многогранников с плоскими гранями или сплайнов с удалением невидимых точек;
- изображение поверхностей в виде многогранников с плоскими гранями или сплайнов с удалением невидимых точек со сложным закрашиванием объектов для имитации отражения света, затенения, прозрачности и использование текстур [3].

8.2 Закрашивание поверхностей

Закрашивание поверхностей – процесс заливки цветом поверхностей.

Закрашивание поверхностей производится методами:

- метод Гуро;
- метод Фонга.

Метод Гуро – способ закрашивания граней трёхмерных объектов, использующий интерполяцию интенсивностей отражения света в вершинах граней. Метод Гуро предназначен для создания изображения гладкой криволиней-

ной поверхности, описанной полигональной сеткой с плоскими гранями или многогранниками.

Алгоритм метода Гуро включает этапы:

1 Определяются нормали к граням.

2 Определяются нормали в вершинах граней.

3 Определяются значения интенсивностей в вершинах.

4 Закрашивание граней цветом, соответствующей линейной интерполяции значений интенсивности в вершинах.

Метод Фонга – способ закрашивания граней трёхмерных объектов, основанный на интерполяции векторов нормалей в вершинах.

Алгоритм метода Фонга включает этапы:

1 Определяются нормали к граням.

2 Определяются нормали в вершинах граней.

3 Цвет точек граней определяется по направлению векторов нормали в соответствии с моделью отражения света [3].

8.3 Практическая работа №7. Визуализация трехмерных объектов. Закрашивание поверхностей методами Гуро и Фонга

Цель работы: Получить теоретические знания и практические навыки в применении Microsoft Visual C++ 2010, графической библиотеки Open GL для разработки приложения, создающего трёхмерные объекты с закрашенными гранями и осуществляющего поворот и масштабирование.

Используемые приемы и технологии: Visual C++ 2010 Professional, библиотеки графических интерфейсов Windows Forms и Microsoft Foundation Classes, графическая библиотека Open GL.

Ключевые термины: визуализация трёхмерных изображений, каркасная модель, закрашивание поверхностей, модели отражения света, метод Гуро, метод Фонга, поворот, масштабирование.

8.4 Варианты заданий к выполнению практической работы №7

Разработайте приложение на языке Visual C++ или используя графическую библиотеку OpenGL, формализующее визуализацию трёхмерного объекта с закрашиванием граней методами Фонга и Гуро, с реализацией поворота и масштабирования.

Вариант 1. Гексаэдр.

Вариант 2. Треугольная призма.

Вариант 3. Конус.

Вариант 4. Треугольная пирамида.

Вариант 5. Прямой круговой цилиндр.

Вариант 6. Прямоугольный параллелепипед.

Вариант 7. Усечённый конус.

Вариант 8. Шестиугольная пирамида.
Вариант 9. Шар.
Вариант 10. Шестиугольная призма.
Вариант 11. Наклонный круговой цилиндр.
Вариант 12. Усечённая четырёхугольная пирамида.
Вариант 13. Четырёхугольная призма.
Вариант 14. Четырёхугольная пирамида.
Вариант 15. Пятиугольная призма.
Вариант 16. Пятиугольная пирамида.
Вариант 17. Октаэдр.
Вариант 18. Икосаэдр.
Вариант 19. Наклонный параллелепипед.
Вариант 20. Прямой параллелепипед.
Вариант 21. Додекаэдр.
Вариант 22. Пятиугольная усечённая пирамида.
Вариант 23. Наклонная четырёхугольная пирамида.
Вариант 24. Тор.
Вариант 25. Эллипсоид.

Методические указания

1 Запустите среду программирования Visual Studio 2010 Professional.
2 Создайте проект «Windows Forms Application Visual C++».
3 Формализуйте алгоритм решения задачи на ПЭВМ.
4 Выведите на экран видеомонитора результаты работы программы.
5 Оформите отчет по практической работе.

Контрольные вопросы

1 Что понимается под визуализацией объёмных изображений?
2 Какие существуют способы визуализации объёмных изображений?
3 Какие существуют модели отражения света?
4 Что понимается под закрашиванием поверхностей?
5 Какие существуют методы закрашивания поверхностей?
6 Какое предназначение метода Гуро?
7 Какие существуют этапы алгоритма метода Гуро?
8 Какие существуют этапы алгоритма метода Фонга?

ЗАКЛЮЧЕНИЕ

Компьютерная графика – наука, разрабатывающая и исследующая методы формирования и обработки изображений посредством компьютера.

Задачи, решаемые средствами компьютерной графики – создание, преобразование и распознавание изображений.

Основные направления применения компьютерной графики – визуализация научных данных, геометрическое проектирование и моделирование, распознавание образов, изобразительное искусство, виртуальная реальность и цифровое видео.

В методических указаниях представлены варианты заданий для выполнения практических работ и теоретическое обоснование по темам графические примитивы, сплайновые кривые, растровые алгоритмы, фракталы, удаление невидимых линий и поверхностей, наложение текстовых сообщений на изображения, визуализация трёхмерных объектов и закрашивание поверхностей методами Гуро и Фонга.

Методические указания к выполнению практических работ позволяют студентам закрепить теоретический знания по дисциплине «Компьютерная графика» и приобрести практические навыки в разработке программных приложений визуализации, обработки и распознавания изображений на языке Visual C++ в среде программирования Microsoft Visual Studio 2010 Professional.

СПИСОК ЛИТЕРАТУРЫ

1 Кудрина М. А. Компьютерная графика. – Самара : Изд-во Самар. Гос. аэрокосм. ун-та, 2013. – 138 с.

2 Петров М. Н. Компьютерная графика : учебник для вузов. – Санкт-Петербург : Питер, 2011. – 544 с.

3 Порев В. Н. Компьютерная графика. – Санкт-Петербург : БХВ – Петербург, 2004. – 432 с.

4 Поляков А. Ю., Брусенцев В. А. Методы и алгоритмы компьютерной графики в примерах на Visual C++. – Санкт-Петербург : БХВ-Петербург, 2003. – 560 с.

5 Корнеев В. И., Гагарина Л. Г., Корнеева М. В. Программирование графики на C++. Теория и примеры : учебное пособие. – Москва : BL «ФОРУМ» : ИНФРА-М, 2017. – 517 с.

6 Вельтмандер П. В. Машинная графика : учебное пособие. в 3-х книгах. Новосибирск : Изд-во НГУ, 1997.

7 Клементьев К. Е. Методы и средства компьютерной графики. – Самара : СНЦ-РАН, 2005. – 168 с.

8 Шикин Е. В., Боресков А. В. Компьютерная графика. Динамика, реалистические изображения. – Москва : ДИАЛОГ – МИФИ, 1995. – 288 с.

9 Шикин Е. В., Плис А. И. Кривые и поверхности на экране компьютера. Руководство по сплайнам для пользователей. – Москва : Диалог-МИФИ, 1996. – 240 с.

10 Фень Юань. Программирование графики для Windows. – Санкт-Петербург : Питер, 2002. – 1072 с.

11 Уэлстид С. Фракталы и вейвлеты для сжатия изображений в действии. – Москва : Триумф, 2003. – 319 с.

12 Попов А. DirectX 10 – это просто. Программируем графику на C++. – Санкт-Петербург : BHV-СПб, 2008. – 464 с.

13 Флёнов М. DirectX и C++. Искусство программирования. – Санкт-Петербург : BHV-СПб, 2006. – 384 с.

14 Торн А. Direct X9. Осваиваем 3D-пространство. – Москва : НТ Пресс, 2007. – 288 с.

15 Верма Р. Д. Введение в Open GL. – Москва : Горячая линия – Телеком, 2013. – 294 с.

Семахин Андрей Михайлович

КОМПЬЮТЕРНАЯ ГРАФИКА

Методические указания
к выполнению практических работ
для студентов направления подготовки 09.03.04
«Программная инженерия»

Редактор Н.Н. Погребняк

Подписано в печать 27.11.18	Формат 60x84 1/16	Бумага 65 г/м^2
Печать цифровая	Усл. печ. л. 4,25	Уч.-изд. л. 4,25
Заказ №211	Тираж 15	Не для продажи

БИЦ Курганского государственного университета.
640020, г. Курган, ул. Советская, 63/4.
Курганский государственный университет.