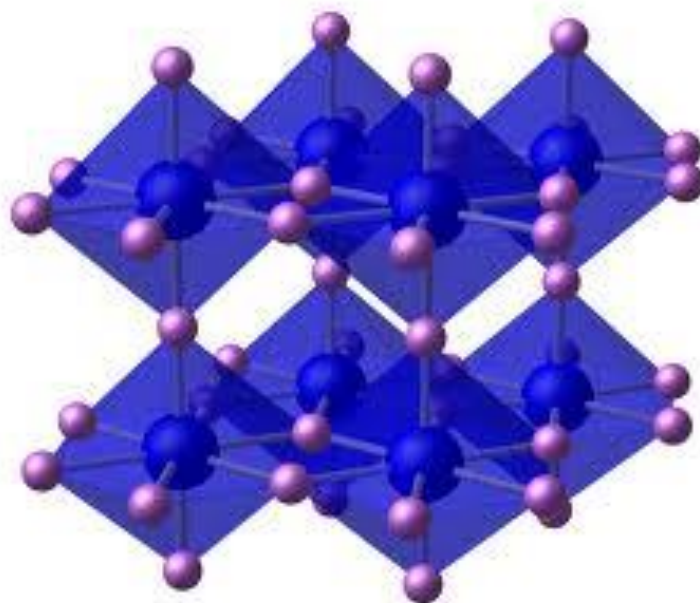


**Н. Ю. Хабибулина, Д.И. Хабибулин**

## **КОМПЬЮТЕРНАЯ ГРАФИКА**

**Учебно-методическое пособие по выполнению  
лабораторных и самостоятельных работ**



Томск – 2020

Министерство науки и высшего образования РФ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования

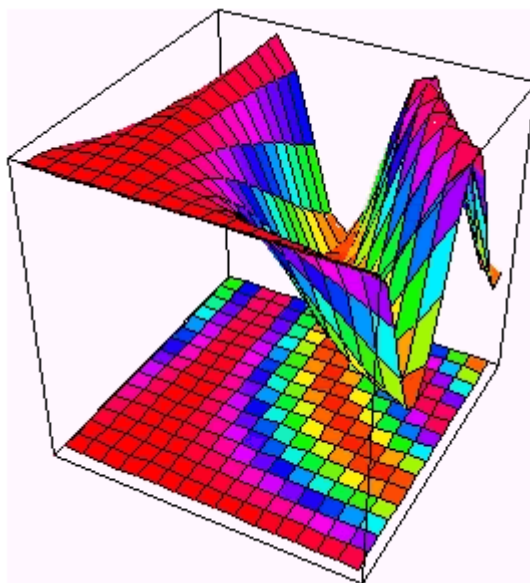
**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)**

Кафедра компьютерных систем в управлении и проектировании (КСУП)

**Н. Ю. Хабибулина, Д.И. Хабибулин**

**КОМПЬЮТЕРНАЯ ГРАФИКА**

Учебно-методическое пособие по выполнению  
лабораторных и самостоятельных работ



Томск – 2020

**Хабибулина Н. Ю., Хабибулин Д.И.**

Компьютерная графика: учеб. методич. пособие по выполнению лабораторных и самостоятельных работ / Н. Ю. Хабибулина, Д.И. Хабибулин – Томск : Томск. гос. ун-т систем упр. и радиоэлектроники, каф. КСУП, 2020. - 89 с.

В пособии изложены основные положения изучения дисциплины «Компьютерная графика»: цели и задачи дисциплины, содержание лекционного материала, методические рекомендации по выполнению, оформлению и защите лабораторных работ, задания к лабораторным работам, рекомендации к самостоятельной работе студентов. Предложен учебный материал для начального изучения среды программирования Visual Studio.

Пособие предназначено для бакалавров высших технических учебных заведений.

© Хабибулина Н. Ю.,  
Хабибулин Д.И.

© Том. гос. ун-т систем упр. и  
радиоэлектроники,  
каф. КСУП, 2020

## Содержание

Введение .....	5
Методические указания по выполнению лабораторных работ. Требования к оформлению отчета ....	9
Лабораторная работа № 1 .....	10
Задания для самостоятельного выполнения к лабораторной работе № 1 .....	11
Контрольные вопросы к лабораторной работе № 1 .....	12
Лабораторная работа № 2.....	13
Задания к лабораторной работе № 2 .....	19
Задания для самостоятельного выполнения к лабораторной работе № 2 .....	23
Контрольные вопросы к лабораторной работе № 2 .....	23
Лабораторная работа № 3 .....	24
Задания к лабораторной работе № 3 .....	31
Задания для самостоятельного выполнения к лабораторной работе № 3 .....	31
Контрольные вопросы к лабораторной работе № 3 .....	32
Лабораторная работа № 4.....	33
Задания к лабораторной работе № 4 .....	42
Задания для самостоятельного выполнения к лабораторной работе № 4 .....	43
Контрольные вопросы к лабораторной работе № 4 .....	45
Лабораторная работа № 5 .....	46
Задания к лабораторной работе № 5 .....	46
Контрольные вопросы к лабораторной работе № 5 .....	48
Задания для самостоятельного выполнения к лабораторной работе № 5 .....	48
Лабораторная работа № 6.....	50
Задания к лабораторной работе № 6 .....	50
Контрольные вопросы к лабораторной работе № 6 .....	53
Лабораторная работа № 7.....	54
Задания к лабораторной работе № 7 .....	67
Контрольные вопросы к лабораторной работе № 7 .....	67
Методические рекомендации по выполнению индивидуальной самостоятельной работы.....	68
Список рекомендуемой литературы и других источников.....	72
Приложение А .....	73
Приложение Б.....	82
Приложение В .....	85
Приложение Г .....	86

## **Введение**

Образовательная программа обучения бакалавров технических направлений предусматривает изучение дисциплины «Компьютерная графика». В соответствии с учебным планом содержание данной дисциплины состоит из лекционных занятий, лабораторных и самостоятельных работ. Описанию последних и посвящено настоящее учебно-методическое пособие.

В начале пособия приведена общая характеристика лекционного материала по рассматриваемой дисциплине.

Раздел 2 посвящен описанию курса лабораторных работ: приведены методические рекомендации по порядку выполнения лабораторных работ, и задания к лабораторным работам, отображены требования к содержанию и оформлению отчета по работе. Кроме того, пособие содержит темы и методические указания по выполнению самостоятельной работы студентов.

## **Цели и задачи дисциплины**

Основная цель изучения дисциплины «Компьютерная графика» – научить будущего специалиста применять методы отображения графической информации в двумерном и трехмерном пространстве, реализовывать алгоритмы компьютерной графики с помощью современных программных сред (в частности, интегрированной среды разработки Visual Studio), использовать программные и аппаратные средства компьютерной графики.

В соответствии с учебным планом данная дисциплина изучается студентами в течение одного семестра. В ходе изучения студенты должны ознакомиться с предоставленным курсом лекций, выполнить лабораторные работы, написать реферат по темам, изучаемым самостоятельно, выполнить контролирующие материалы. В процессе выполнения заданий необходимо проявить свое умение пользоваться дополнительными источниками информации и творческий подход к решению поставленной задачи. Итоговой аттестацией, оценивающей уровень изучения предмета, является зачет.

## **Содержание дисциплины**

### **Лекционный материал**

1) Компьютерная графика; области применения компьютерной графики; тенденции построения современных графических систем: графическое ядро, приложения, инструментарий для написания приложений; основные функциональные возможности современных графических систем; организация диалога в графических системах; классификация и обзор современных графических систем; стандарты в области разработки графических систем;

2) Растровая графика: алгоритмы генерации отрезка - цифровой дифференциальный анализатор (ЦДА обычный и несимметричный), алгоритм Брезенхема; алгоритм построения окружности Брезенхема; стиль линии; способы устранения ступенчатости изображения.

3) Растровая графика: четырех- и восьми связная области, гранично-определенная и внутренне-определенная область, простой и сложный контур, обход простого и сложного контура, закрашка многоугольника, алгоритмы заливки области с затравкой (простой и построчный).

4) Растровая графика: алгоритмы отсечения (простой двумерный и Коэна-Сазерленда).

5) Векторная графика: системы координат; однородные координаты; 2D-преобразование (преобразование на плоскости – масштабирование, сдвиг, поворот вокруг начала координат, поворот вокруг произвольной точки): преобразование точек, преобразование прямых линий, преобразование параллельных линий, преобразование плоских фигур.

6) Векторная графика. Преобразование в трехмерном пространстве (3D-преобразование): виды геометрических моделей, их свойства, параметризация моделей; геометрические операции над моделями; матрицы преобразований (сдвиг, масштабирование, поворот вокруг оси).

7) Векторная графика: процедура перехода от мировых координат к экранным; виды проекций; матрицы перспективных и параллельных проекций.

8) Векторная графика: удаление невидимых линий и поверхностей - метод  $z$  – буфера, алгоритм Варнака, алгоритм, использующий нормаль поверхности, алгоритм Робертса.

9) Способы создания фотореалистических изображений: модели освещения, механизм диффузного и зеркального отражения света, модели закраски, прозрачность, тени, фактура, излучательность.

### **Лабораторные занятия**

1) Лабораторная работа № 1 - Первый графический проект. Матричные операции. *Цель работы* – знакомство с технологией создания Windows-приложения, в частности создание Windows-приложения в интегрированной среде разработки Visual Studio (далее по тексту возможно сокращенное название - VS), изучение основных элементов и принципов функционирования VS, создание первого графического и математического проекта.

2) Лабораторная работа № 2 - Алгоритмы растровой графики. Построение векторов и окружностей. *Цель работы* – изучение и реализация алгоритмов растровой графики: генерация отрезка алгоритмами Брезенхема и цифровым дифференциальным анализатором (обычным и несимметричным), генерация окружности алгоритмом Брезенхема, вывод линий различного стиля.

3) Лабораторная работа № 3 - Алгоритмы растровой графики. Обход и заливка контура. Отсечение отрезков. *Цель работы* – изучение и реализация алгоритмов растровой графики: обход сложного контура, заливка контура с затравкой, закраска многоугольника; отсечения отрезков.

4) Лабораторная работа № 4 - Построение 2D изображений. 2D аффинные преобразования. *Цель работы* – изучение и реализация алгоритмов аффинных преобразований фигур на плоскости, получение навыков моделирования двумерных объектов.

5) Лабораторная работа № 5 - Векторно-полигональная и аналитическая модели объекта. Преобразования в пространстве (3D-преобразования). *Цель работы* – изучение и реализация алгоритмов построения и преобразования

трехмерного объекта с использованием векторно-полигональной и аналитической моделей.

6) Лабораторная работа № 6 - Преобразования в пространстве. Проекция. Удаление невидимых линий. *Цель работы* – изучение и реализация алгоритмов построения проекции фигуры, удаления невидимых линий и граней.

7) Лабораторная работа № 7 - Введение в OpenGL. *Цель работы* – изучение принципов применения библиотеки OpenGL при разработке приложений в C#.

Методические указания по выполнению, оформлению отчета и защите лабораторных работ приведены ниже.

### **Самостоятельная работа**

Методические рекомендации по выполнению самостоятельной работы представлены в одноименных разделах методических указаний к лабораторным работам настоящего пособия.



## **Методические указания по выполнению лабораторных работ.**

### **Требования к оформлению отчета**

По задумке автора, результатом выполнения курса лабораторных работ по дисциплине «Компьютерная графика» должен быть программный продукт, объединяющий в себе выполненные задания всех лабораторных работ. Поэтому перед началом выполнения заданий необходимо создать проект, основная экранная форма которого содержит семь (по числу лабораторных работ) кнопок. Каждая кнопка имеет название «Лабораторная работа № \_\_», при нажатии на кнопку выполняется программный модуль соответствующей лабораторной работы. Заголовок формы должен содержать название дисциплины, фамилию, имя, отчество и группу студента.

Для защиты лабораторной работы необходимо представить созданный программный продукт (исходный код и загрузочный файл) и отчет.

Содержание отчета по лабораторной работе:

- 1) титульный лист;
- 2) цель работы и постановка задачи;
- 3) анализ задачи (краткая теория по теме лабораторной работы: математические модели решения поставленной задачи, описание используемых алгоритмов);
- 4) описание структуры и алгоритма программы;
- 5) описание основных процедур и функций;
- 6) руководство пользователя;
- 7) ответы на контрольные вопросы;
- 8) заключение.

Образец титульного листа и оформления отчета по лабораторной работе представлены в приложении А.

## Лабораторная работа № 1

### Первый графический проект. Матричные операции

*Цель работы* – знакомство с технологией создания Windows-приложения, в частности создание Windows-приложения в интегрированной среде разработки Visual Studio (далее по тексту возможно сокращенное название - VS), изучение основных элементов и принципов функционирования VS, а также создание первого графического и математического проекта.

### Порядок выполнения лабораторной работы № 1

#### 1 Знакомство с интегрированной средой разработки Visual Studio

Изучите учебный материал, посвященный разработке приложений с графическим интерфейсом (Windows Form). Для этого перейдите по ссылке:

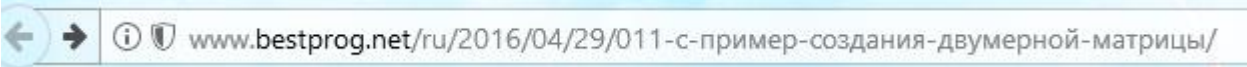
<https://metanit.com/sharp/windowsforms/1.1.php>

**Обязательно выполните примеры главы 1 и главы 2 данного электронного пособия.**

Остальные главы необходимо изучить самостоятельно.

#### 2 Разработка первого математического приложения

Изучите учебный материал, посвященный разработке Windows Form приложения для работы с матрицами. Для этого перейдите по ссылке:

  
[http://www.bestprog.net/ru/2016/04/29/011-с-  
 %D0%BF%D1%80%D0%B8%D0%BC%D0%B5%D1%80-  
 %D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D1%8F-  
 %D0%B4%D0%B2%D1%83%D0%BC%D0%B5%D1%80%D0%BD%D0%BE%D  
 0%B9-%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8B/](http://www.bestprog.net/ru/2016/04/29/011-с-%D0%BF%D1%80%D0%B8%D0%BC%D0%B5%D1%80-%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D1%8F-%D0%B4%D0%B2%D1%83%D0%BC%D0%B5%D1%80%D0%BD%D0%BE%D0%B9-%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8B/)

**и выполните пример создания двумерной матрицы на форме.**

#### 3 Расширение функций математического приложения

Модифицируйте созданный программный модуль, реализовав в нем

операции сложения, вычитания и умножения матриц (размерность матриц вводит пользователь)

#### **4 Защита результатов лабораторной работы**

Представьте разработанные программные модули преподавателю.

##### **Задания для самостоятельного выполнения к лабораторной работе № 1**

###### **– Общие требования к программному продукту и защите проекта**

1. В программе необходимо предусмотреть следующее:

- размер матрицы и вектора вводится с клавиатуры (максимальный размер – 4x4);
- элементами матрицы являются вещественные числа;
- ввод массивов с клавиатуры или заполнение их случайными числами;
- вывод исходных данных и результатов на экран;
- проверка ошибок ввода и возможности выполнения операции (например, сложение матриц выполняется для матриц одинакового размера).

2. При оформлении отчета по индивидуальной части лабораторной работы придумайте по два тестовых вопроса по изученным и реализованным алгоритмам. В каждом отчете должны быть представлены вопросы разного типа (закрытые вопросы с выбором одного правильного ответа, закрытые вопросы с выбором нескольких правильных ответов, открытый вопрос с вводом ответа в виде числа или строки, вопрос на установление соответствия, вопрос-эссе, вопрос на установление правильной последовательности и т.д.).

###### **– Задание для самостоятельной работы**

1. Реализуйте два из нижеперечисленных вариантов заданий (номера вариантов необходимо получить у преподавателя).

2. Подготовьтесь к выполнению лабораторной работы № 2. Для этого ознакомьтесь с теоретическим материалом по теме лабораторной работы № 2.

##### **Варианты индивидуальных заданий на самостоятельную работу к лабораторной работе № 1**

- 1) Умножение матрицы на константу.
- 2) Модуль вектора.
- 3) Скалярное произведение векторов.
- 4) Векторное произведение векторов.
- 5) Транспонирование матрицы.
- 6) Умножение матрицы на вектор.
- 7) Умножение вектора на константу
- 8) Определитель матрицы\*.
- 9) Обратная матрица\*.

### **Контрольные вопросы к лабораторной работе № 1**

1. Что такое VS?
2. Как запустить VS?
3. Что такое «кнопка быстрого доступа»?
4. Что такое «Обозреватель решений (Solution Explorer)»?
5. Что такое «Окно свойств (Properties)»?
6. Что такое «Панель инструментов (Toolbox)»?
7. Что такое «дизайнер форм»? Чем отличается дизайнер форм от формы?
8. Что такое «свойство»?
9. Что такое «событие»?
10. Как получить доступ к свойствам, расположенным на странице «События» (Events)?
11. Для чего нужно окно редактора кода?
12. Что такое «проект»? Перечислите способы создания нового проекта.
13. Как сохранить проект? Перечислите все возможные способы.
14. Как переключиться между формой и модулем?
15. Какие основные файлы входят в проект VS?
16. С какими компонентами VS Вы познакомились в данной лабораторной работе?

## Лабораторная работа № 2

### Алгоритмы растровой графики. Построение векторов и окружностей

*Цель работы* – изучение и реализация алгоритмов растровой графики: генерация отрезка алгоритмами Брезенхема и цифровым дифференциальным анализатором (обычным и несимметричным), генерация окружности алгоритмом Брезенхема, вывод линий различного стиля.

### Краткая теория и порядок выполнения лабораторной работы

Для выполнения лабораторной работы ознакомьтесь с лекционным материалом по рассматриваемой теме.

Продолжим изучать приемы программирования в среде Visual Studio на примере создания проекта вывода отрезка обычным алгоритмом цифрового дифференциального анализатора (ЦДА).

#### **Пример 4.**

#### *Алгоритм генерации отрезка ЦДА*

С помощью ЦДА решается дифференциальное уравнение отрезка, имеющее вид:

$$\frac{dY}{dX} = \frac{Py}{Px}$$

где  $Py = Y_k - Y_n$  приращение координат отрезка по оси Y,

$Px = X_k - X_n$  - приращение координат отрезка по оси X,

$X_n, Y_n$  – координаты начальной точки отрезка,

$X_k, Y_k$  – координаты конечной точки отрезка.

В *обычном ЦДА* тем или иным образом определяется количество узлов N, используемых для аппроксимации отрезка. Затем за N циклов вычисляются координаты очередных узлов:

$$X_0 = X_n; \quad X_{i+1} = X_i + \frac{Px}{N}$$

$$Y_0 = Y_n; \quad Y_{i+1} = Y_i + \frac{Py}{N}$$

Получаемые значения  $X_i$ ,  $Y_i$  преобразуются в целочисленные значения координаты очередного пикселя либо округлением, либо отбрасыванием дробной части. Данный пиксель выводится на экран.

Координаты концов отрезка будем задавать в интерактивном режиме: пользователь нажимает правую кнопку мыши и, не отпуская ее, протягивает отрезок до конечной точки, отпускает правую кнопку – на экране получаем отрезок.

1. Создайте новый проект – LINII.
2. С помощью дизайнера форм создайте следующий интерфейс основной формы программы (рис.2.1):

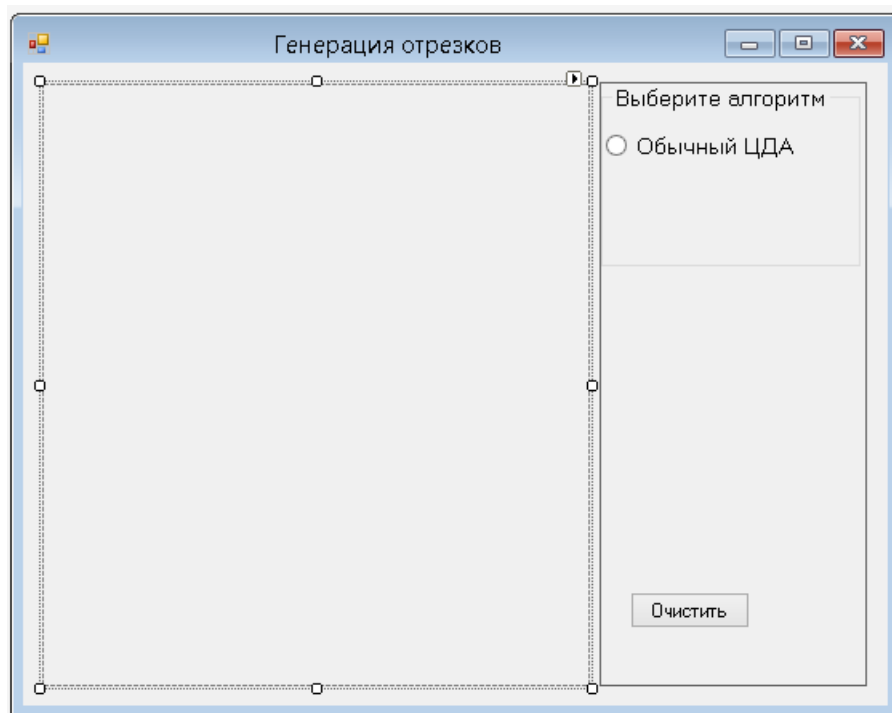


Рисунок 2.1 – Интерфейс программы LINII

- в свойстве Text компонента Form1 наберите «Генерация отрезка»;
- выберите компонент Panel (ветвь панели компонентов Контейнеры) и разместите его на правой стороне формы; установите высоту панели максимально возможную по форме; свойство BorderStyle установите в FixedSingle;
- на панели (Panel) разместите компонент GroupBox (ветвь Контейнеры); его свойство Text замените на «Выберите алгоритм»; в нем

разместите один компонент RadioButton. Его свойство Text замените на «Обычный ЦДА»;

- добавьте на панель кнопку (Button) с названием «Очистить»;
- выберите компонент PictureBox (в ветви Стандартные элементы управления) и разместите его на большую часть формы.

3. Напишем программный код попиксельного вывода отрезка. Координаты начальной точки отрезка фиксируются при нажатии пользователем правой кнопки мыши на компоненте PictureBox. Для этого на странице Events (События) инспектора объектов сделайте двойной щелчок напротив события MouseDown. Перейдите в редактор кода и наберите следующие строки:

```
private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
{
    if (radioButton1.Checked == true)
    {
        xn = e.X;
        yn = e.Y;
    }

    else MessageBox.Show ("Вы не выбрали алгоритм вывода фигуры!");
}
```

Пояснения:

- если в GroupBox не выбран ни один переключатель, то на экран выводится сообщение «Вы не выбрали алгоритм вывода фигуры!».
- Для проверки выбора RadioButton1 используется свойство Checked, принимающее значение True, если выбран данный переключатель, и False – в противном случае.
- Параметры e.X, e.Y – это координаты пикселя, в который попал курсор при нажатии правой кнопки мыши (по умолчанию метод - обработчик события нажатия правой кнопки мыши – возвращает структуру e с двумя полями

X и Y координаты пикселя). В нашем случае это координаты начальной точки отрезка.

- Кроме этого, не забудьте описать глобальные переменные xn, yn. Для этого в программный код вставьте строчку с описанием переменных в начале описания класса Form1

```
public partial class Form1 : Form
{
    Public int xn,yn,xk,yk;

    public Form1()
    {
        InitializeComponent();
    }
}
```

4. При отпускании правой кнопки мыши фиксируем конечную точку отрезка и выводим его на экран красным цветом. Для фиксирования отпускания правой кнопки мыши используется событие MouseUp компонента PictureBox. На странице Events (События) инспектора объектов сделайте двойной щелчок напротив данного события и в редакторе кода напишите:

```
private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
{
    int i, n;
    double xt, yt, dx, dy;

    xk = 0;
    yk = 0;
    xk = e.X;
    yk = e.Y;

    dx = xk - xn;
    dy = yk - yn;
    n = 100;
    xt = xn;
    yt = yn;

    for (i = 1; i <= n; i++)
    {
        //Объявляем объект "myPen", задающий цвет и толщину пера
        Pen myPen = new Pen(Color.Black, 1);

        //Объявляем объект "g" класса Graphics и предоставляем
        //ему возможность рисования на pictureBox1:
    }
}
```



```

Graphics g = Graphics.FromHwnd(pictureBox1.Handle);

//Рисуем прямоугольник:
g.DrawRectangle(myPen, (int)xt, (int)yt, 2, 2);

//Рисуем закрашенный прямоугольник:
//Объявляем объект "redBrush", задающий цвет кисти
// SolidBrush redBrush = new SolidBrush(Color.Red);

//Рисуем закрашенный прямоугольник
// g.FillRectangle(redBrush, (int)xt, (int)yt, 2, 2);

    xt = xt + dx / n;
    yt = yt + dy / n;
}
}

```

Пояснения:

- реализуем обычный алгоритм ЦДА с  $N=100$ ;
  - в данном примере вывод отрезка можно производить двумя способами: первый (не закомментирован) - отрезок рисуется прямоугольниками размером 2X2 пикселя; второй – (закомментировано в примере) – закрашенными прямоугольниками. Попробуйте рисование обоими способами – опишите разницу преподавателю;
  - `(int)xt` - преобразование переменного `xt` типа *double* в переменную типа *int*.
  - **Не забудьте описать глобальные целочисленные переменные `xk`, `yk`, `xn`, `yn`.**
5. Сохраните и запустите программу. Нарисуйте несколько линий. Экранная форма программы может выглядеть так (рис.2.2):

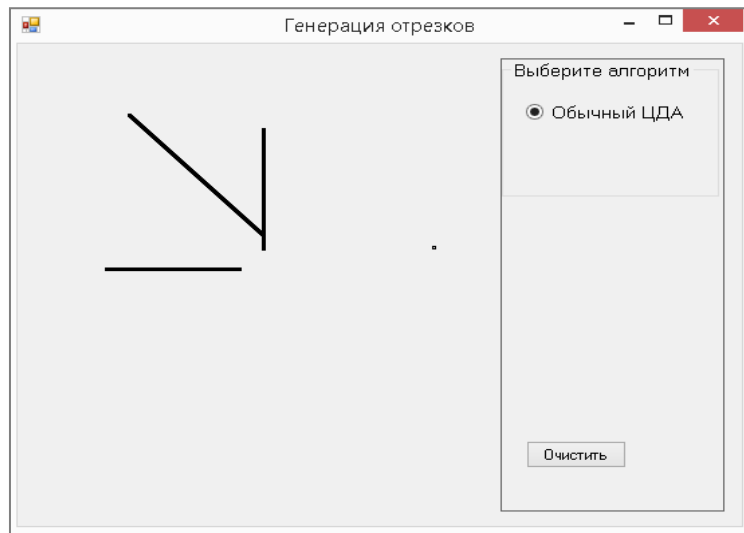


Рисунок 2.2 – Результаты работы программы

6. Пропишем процедуру очистки поля рисования. Дважды щелкните на кнопке «Очистить» и перейдите в редактор кода. Если набрать следующие строки:

```
private void button1_Click(object sender, EventArgs e)
{
    Bitmap myBitmap = new Bitmap(pictureBox1.Height, pictureBox1.Width);

    for (int x = 0; x < myBitmap.Width; x++)
    {
        for (int y = 0; y < myBitmap.Height; y++)
        {
            //Задаем цвет пикселя по схеме RGB (от 0 до 255 для каждого цвета)

            Color newColor = Color.FromArgb(247, 249, 239);
            myBitmap.SetPixel(x, y, newColor);
        }
    }

    pictureBox1.Image = myBitmap;

    // pictureBox1.Image = null;
}
```

то все пиксели компонента PictureBox1 перекрасятся в светлый цвет (рис.2.3, а). Если же прописать:

```
private void button1_Click(object sender, EventArgs e)
{
    pictureBox1.Image = null;
}
```

то использовали метод обновления компонента PictureBox, результат будет аналогичным рис. 2.3,б.

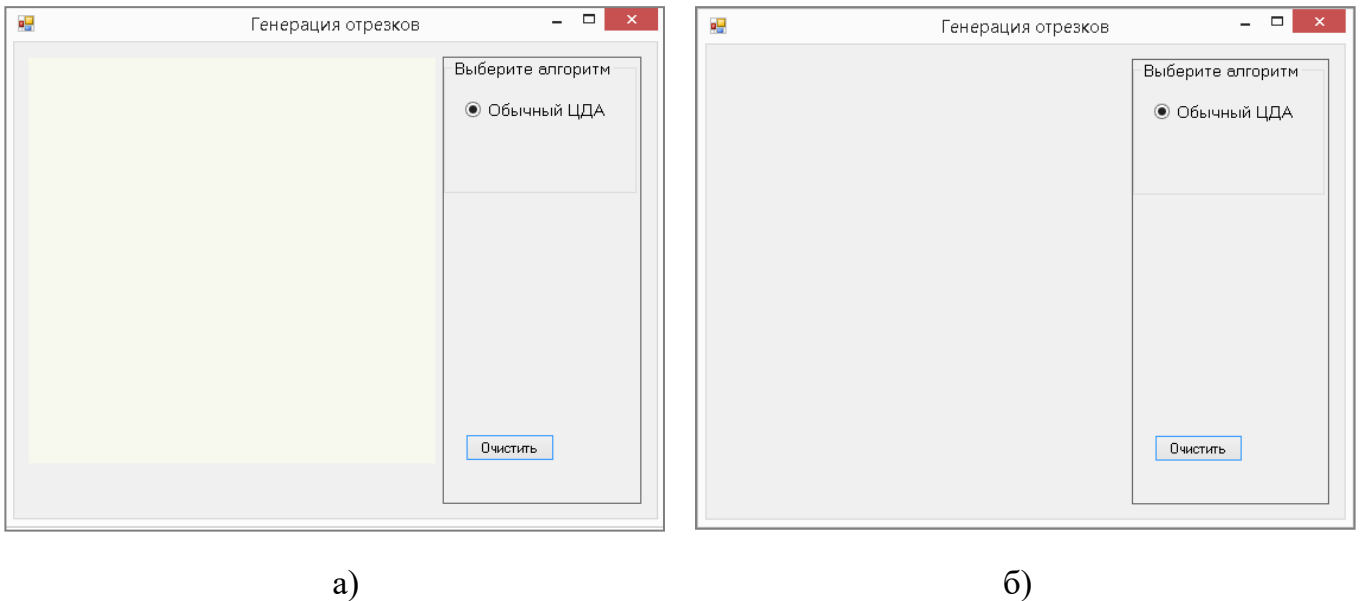


Рисунок 2.3

## Задания к лабораторной работе № 2

1. Реализуйте представленный пример.
2. Добавьте возможность ввода координат отрезка через компоненты TextBox.
3. Создайте обработчик (метод), реализующий следующее:
  - настройку цвета линии (используйте переменную типа Color; например, чтобы преобразовать строку в переменную типа Color, можно воспользоваться следующим преобразованием:

```
string s1 = textBox1.Text;
Color c = (Color)TypeDescriptor.GetConverter(typeof(Color)).ConvertFromString(s1);
);
```

- настройку стиля линии – толстая (предусмотрите настройку толщины линии), тонкая, сплошная, пунктирная линия (предусмотрите настройку шага пунктира).

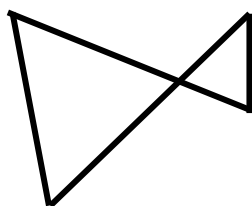
Примечание: для ввода исходных данных (концов отрезка, толщины линии, длины штриха линии) можно использовать диалоговый ввод через компоненты TextBox.

4. Напишите метод, в результате выполнения которого на экране получаем некоторую сцену, содержащую пересекающиеся простые фигуры – отрезки, прямоугольники, квадраты, треугольники.

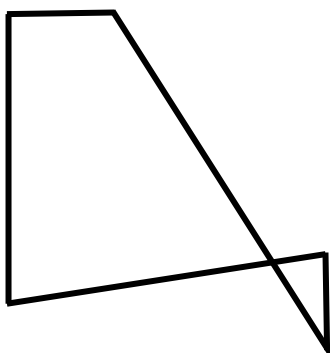
Примечание: для отображения геометрических фигур можно создать соответствующие кнопки и их программные модули (вывод прямоугольника, вывод квадрата, вывод треугольника и т.п.).

5. Напишите метод, реализующий вывод на экран заданной фигуры (вид фигуры выбирайте по вариантам).

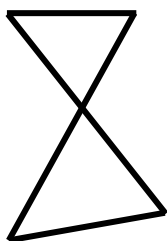
**Вариант 1**



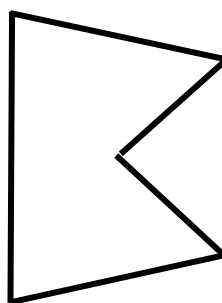
**Вариант 2**



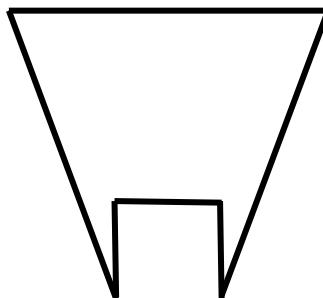
**Вариант 3**



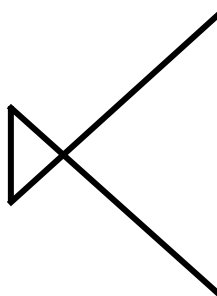
**Вариант 4**

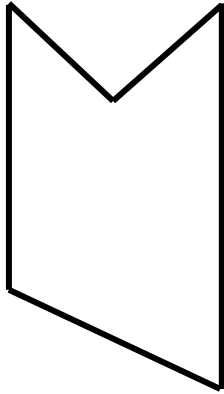
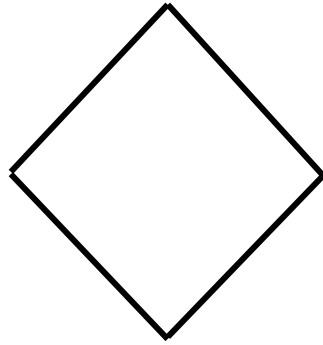
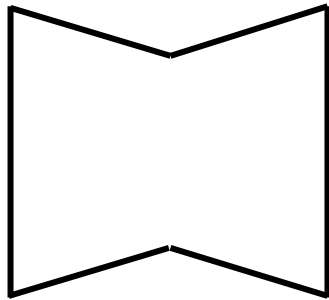
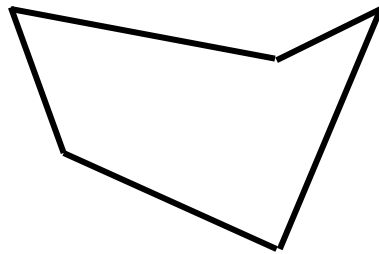
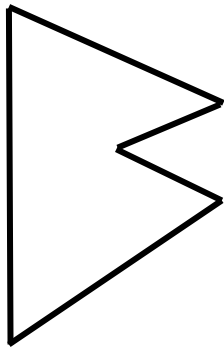
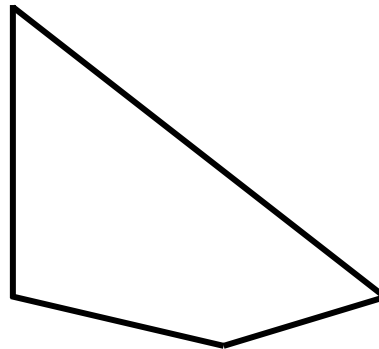
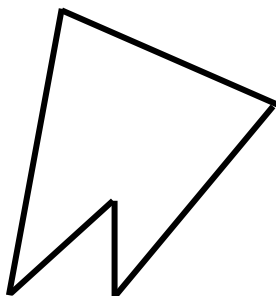
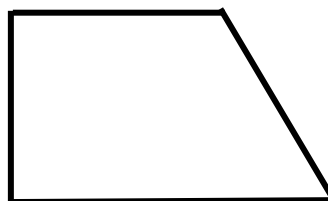


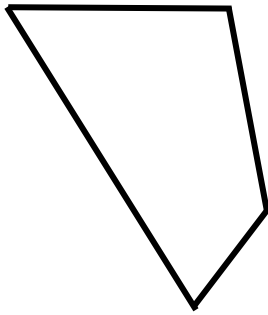
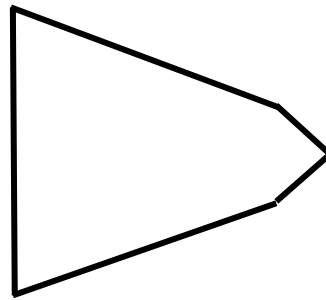
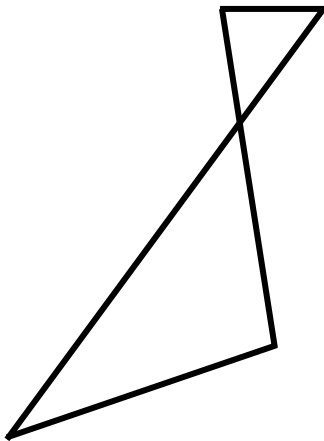
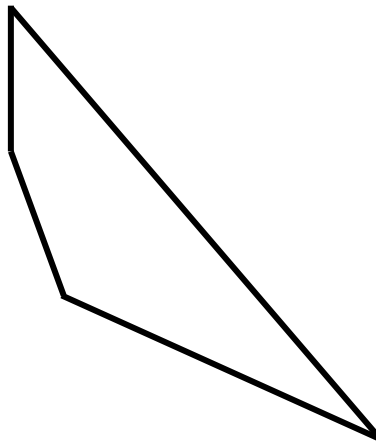
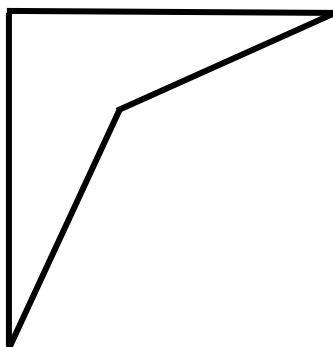
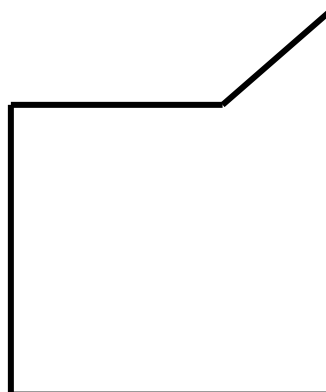
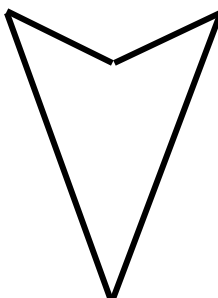
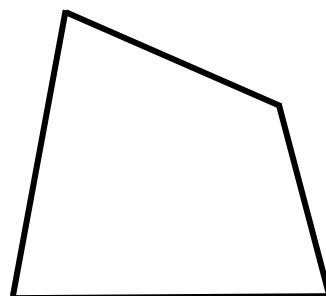
**Вариант 5**

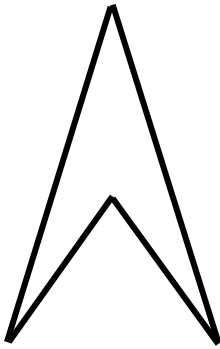
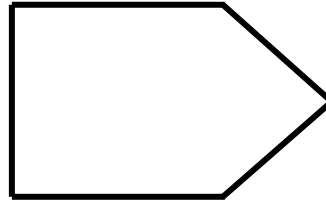


**Вариант 6**



*Вариант 7**Вариант 11**Вариант 8**Вариант 12**Вариант 9**Вариант 13**Вариант 10**Вариант 14*

*Вариант 15**Вариант 19**Вариант 16**Вариант 20**Вариант 17**Вариант 21**Вариант 18**Вариант 22*

**Вариант 23****Вариант 24****Задания для самостоятельного выполнения к лабораторной работе № 2**

1. Реализуйте следующий алгоритм:

- генерацию отрезка по несимметричному алгоритму ЦДА;
- генерацию отрезка по алгоритму Брезенхема;
- генерацию окружности по алгоритму Брезенхема;

2. Подготовьтесь к выполнению лабораторной работы № 3. Для этого ознакомьтесь с теоретическим материалом по теме лабораторной работы № 3.

**Контрольные вопросы к лабораторной работе № 2**

1. Опишите кратко алгоритм генерации отрезка ЦДА.
2. Чем отличаются обычный и несимметричный алгоритмы ЦДА?
3. Кратко опишите основную идею алгоритма Брезенхема генерации отрезка.
4. Опишите алгоритм Брезенхема генерации окружности.
5. Что такое «растр»?
6. Что такое «растровая компьютерная графика»?
7. Что такое «пиксель»?
8. Как получить отображение толстой линии?
9. Как получить отображение пунктирной линии?

### Лабораторная работа № 3

#### Алгоритмы растровой графики. Обход и заливка контура. Отсечение отрезков

*Цель работы* – изучение и реализация алгоритмов растровой графики: обход сложного контура, заливка контура с затравкой, закраска многоугольника; отсечения отрезков.

#### Краткая теория и порядок выполнения лабораторной работы

Для выполнения лабораторной работы ознакомьтесь с лекционным материалом по рассматриваемой теме.

Разработаем программный модуль, реализующий простой рекурсивный алгоритм заливки замкнутой 4-х связной гранично-определенной области с затравкой.

#### **Пример 3.**

*Рекурсивный алгоритм заливки 4-х связной гранично-определенной области:*

- определяется, является ли пиксель граничным или уже закрашенным,
- если нет, то пиксель перекрашивается (выполняется функция ЗАЛИВКА), затем проверяются и если надо перекрашиваются 4 соседних пикселя.

Функцию ЗАЛИВКА определим следующим образом:

```

Функция  ЗАЛИВКА  (x,  y)
{
    Если цвет пикселя (x,  y) не равен цвету границы
и цвету заполнения, то
    {
        Установить для пикселя (x,  y) цвет
заполнения;
        ЗАЛИВКА  (x+ 1,  y);
        ЗАЛИВКА  (x-1,  y);
        ЗАЛИВКА  (x,  y+1);
        ЗАЛИВКА  (x,  y-1);
    }
}
  
```



В качестве закрашиваемой области возьмем фигурку, полученную пересечением отрезков (координаты концов отрезка вводятся в программе). Отрезки будут выводиться красным цветом, а область будет заливаться зеленым цветом.

1. Создайте новый проект – ZAPOLNENIE.
2. С помощью дизайнера форм создайте следующий интерфейс основной формы программы (рис.3.1):

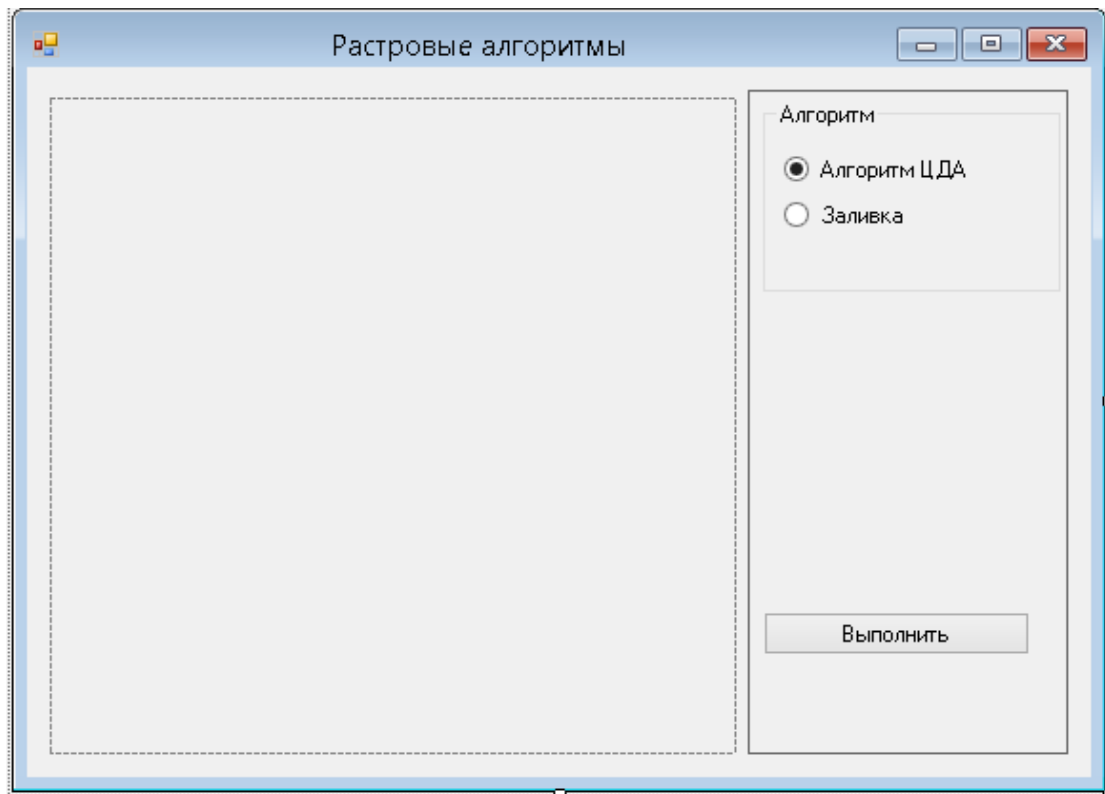


Рисунок 3.1

- в свойстве Text компонента Form1 наберите «Растровые алгоритмы»;
- разместите компонент GroupBox; его свойство Text замените на «Алгоритм»; в нем разместите два компонента RadioButton, назначив соответственно свойствам Text «Алгоритм ЦДА» и «Заливка»;
- добавьте кнопку (Button) с названием «Выполнить»;
- выберите компонент PictureBox и разместите его на большую часть формы.

**Рекомендация:** для быстрого поиска элемента на панели элементов можно воспользоваться строкой поиска данной панели - достаточно начать набирать название искомого элемента в строке.

3. Для вывода отрезков используем программный код метода вывода отрезка обычным алгоритмом ЦДА, созданный в лабораторной работе 2. Оформим его в виде отдельной функции *CDA* с входными параметрами:  $x1$ ,  $y1$ ,  $x2$ ,  $y2$  —соответственно координаты начального и конечного концов отрезка.

```
// вывод отрезка
private void CDA(int x1, int y1, int x2, int y2)
{
    int i, n;
    double xt, yt, dx, dy;

    xn = x1;
    yn = y1;
    xk = x2;
    yk = y2;

    dx = xk - xn;
    dy = yk - yn;
    n = 100;
    xt = xn;
    yt = yn;

    for (i = 1; i <= n; i++)
    {
        Pen myPen = new Pen(current_color, 1);
        mybitmap.SetPixel((int)xt, (int)yt, current_color);
        xt = xt + dx / n;
        yt = yt + dy / n;
    }
}
```

**Пояснения:**

1) не забудьте объявить глобальные переменные

```
public partial class Form1 : Form
{
    public int xn, yn, xk, yk; // концы отрезка
    Bitmap mybitmap; // объект Bitmap для вывода отрезка
    Color current_color; // текущий цвет отрезка
}
```

2) переменная `current_color` будет определять цвет линии. Ее значение будем получать из диалогового окна выбора цвета.

4. Создадим метод-обработчик выбора цвета линии. Добавьте еще одну кнопку на форму, назовите ее «Цвет линии». Добавьте элемент `ColorDialog` (рис.3.2)

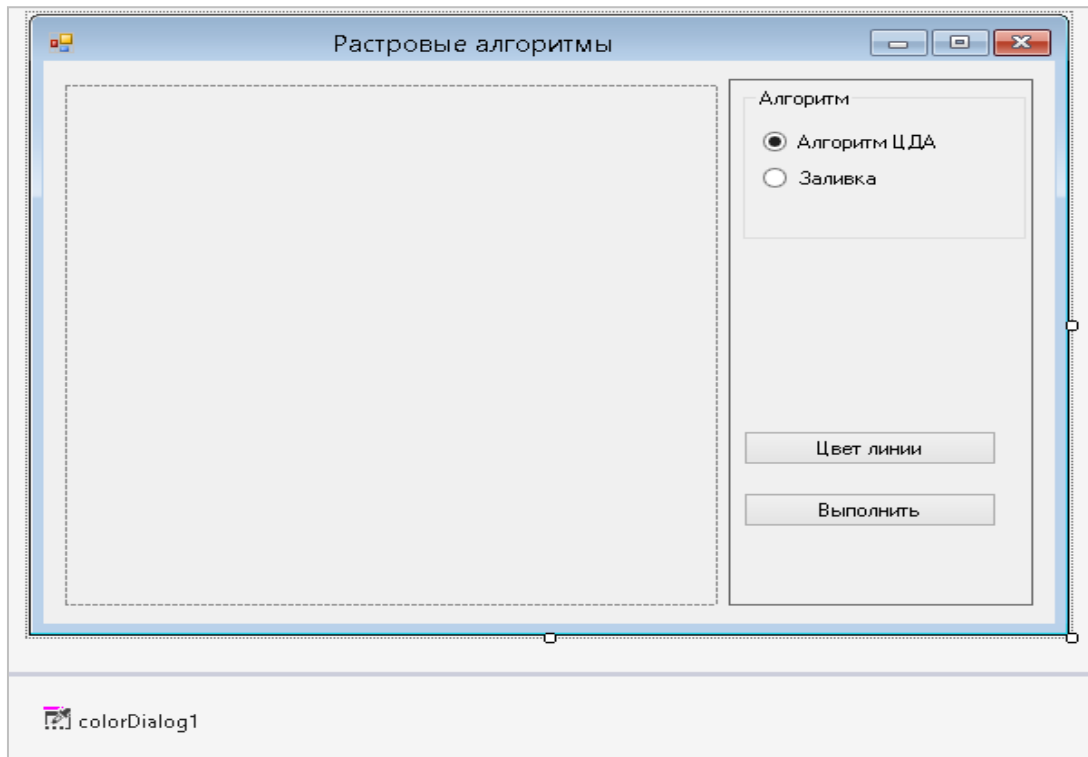


Рисунок 3.2

Перейдите в код обработчика нажатия кнопки «Цвет линии» (например, дважды кликните на кнопке «Цвет линии»). Введите программный код в метод-обработчик нажатия на кнопку. Должен получиться следующий код:

```
private void button2_Click(object sender, EventArgs e)
{
    DialogResult D = colorDialog1.ShowDialog();
    if (D == System.Windows.Forms.DialogResult.OK)
    {
        current_color = colorDialog1.Color;
    }
}
```

5. Создадим обработчик нажатия на кнопку «Выполнить». При нажатии на данную кнопку в случае выбора «Алгоритм ЦДА» на экране появляется прямоугольник и треугольник, нарисованные выбранным цветом. Для их вывода на экран вызовем несколько раз функцию CDA с разными координатами концов отрезка. Если же на форме выбрано «Заливка» - вызываем метод заливки:

```
private void button1_Click(object sender, EventArgs e)
{
    //отключаем кнопки
    button1.Enabled = false;
    button2.Enabled = false;

    //создаем новый экземпляр Bitmap размером с элемент
    //pictureBox1 (mybitmap)
    //на нем выводим попиксельно отрезок

    mybitmap = new Bitmap(pictureBox1.Width, pictureBox1.Height);
    using (Graphics g = Graphics.FromHwnd(pictureBox1.Handle))
    {
        if (radioButton1.Checked == true)
        {
            //рисует прямоугольник
            CDA(10, 10, 10, 110);
            CDA(10, 10, 110, 10);
            CDA(10, 110, 110, 110);
            CDA(110, 10, 110, 110);

            //рисует треугольник
            CDA(150, 10, 150, 200);
            CDA(150, 50, 150, 150);
            CDA(250, 50, 150, 150);
            CDA(150, 10, 250, 150);
        }
        else
        {
            if (radioButton2.Checked == true)
            {
                //получаем растр созданного рисунка в mybitmap
                mybitmap = pictureBox1.Image as Bitmap;

                // задаем координаты затравки
                xn = 160;
                yn = 40;

                // вызываем рекурсивную процедуру заливки с затравкой
                Zaliv(xn, yn);
            }
        }
    }
}
```

```

    }
}
//передаем полученный растр mybitmap в элемент pictureBox
pictureBox1.Image = mybitmap;

// обновляем pictureBox и активируем кнопки
pictureBox1.Refresh();
button1.Enabled = true;
button2.Enabled = true;
}
}

```

**Пояснения:** для вывода рисунка на элемент PictureBox создаем растр (Bitmap), на нем можно работать отдельно с каждым пикселем.

6. Создадим рекурсивную процедуру заливки:

```

// Заливка с затравкой (рекурсивная)
private void Zaliv(int x1, int y1)
{
    // получаем цвет текущего пикселя с координатами x1, y1
    Color old_color = mybitmap.GetPixel(x1, y1);

    // сравнение цветов происходит в формате RGB
    // для этого используем метод ToArgb объекта Color

    if ((old_color.ToArgb() != current_color.ToArgb()) &&
        (old_color.ToArgb() != Color.Green.ToArgb()))
    {
        //перекрашиваем пиксель
        mybitmap.SetPixel(x1, y1, Color.Green);

        //вызываем метод для 4-х соседних пикселей
        Zaliv(x1 + 1, y1);
        Zaliv(x1 - 1, y1);
        Zaliv(x1, y1 + 1);
        Zaliv(x1, y1 - 1);

    }
    else
    {
        //выходим из метода
        return;
    }
}
}

```

7. Сохраните и запустите программу (F5). Выберите «Алгоритм ЦДА», нажмите кнопку «Цвет линии» и выберите цвет линии. Нажмите кнопку «Выполнить». Экранная форма программы может выглядеть так (рис.3.3).

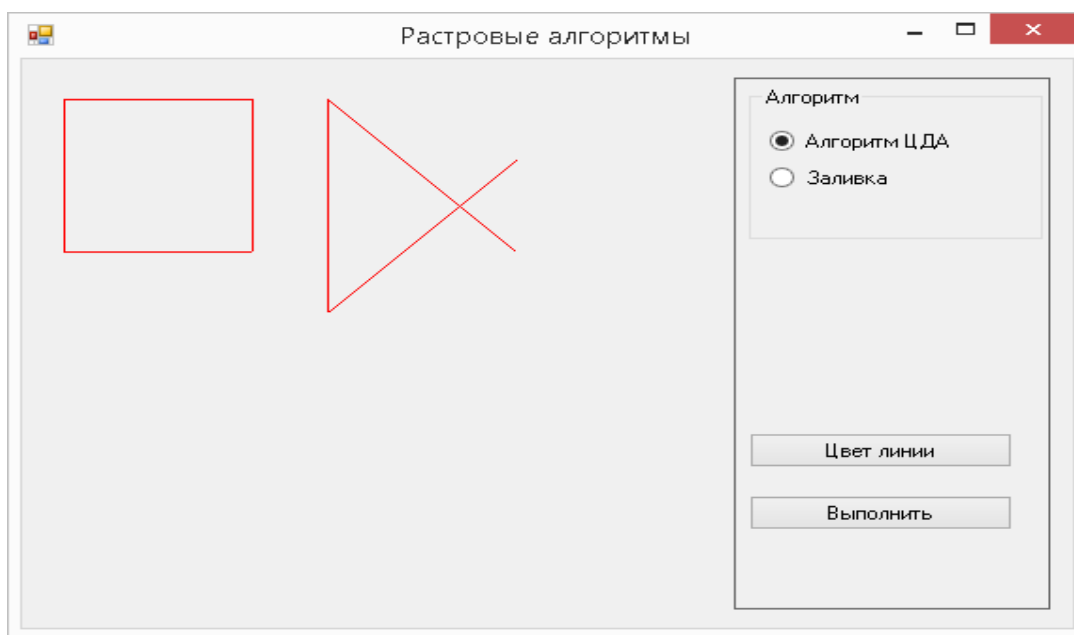


Рисунок 3.3 – Результат работы алгоритма вывода отрезка

8. Выберите «Заливка» и нажмите кнопку «Выполнить». Экранная форма программы может выглядеть так (рис.3.4).

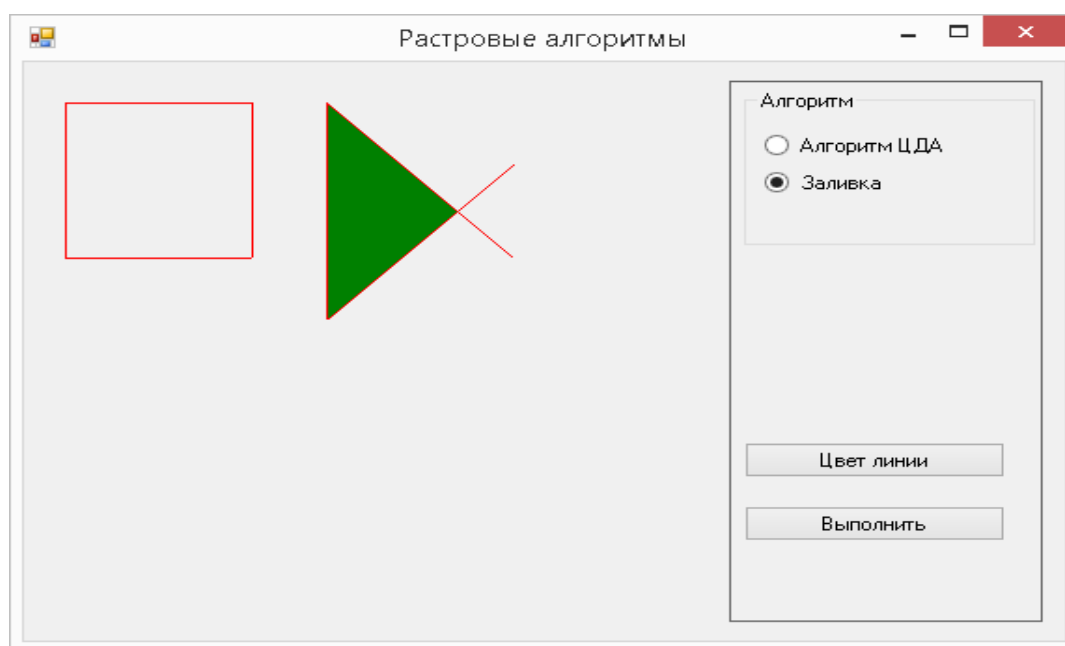


Рисунок 3.4 – Результат работы алгоритма заливки с затравкой

### Задания к лабораторной работе № 3

1. Реализуйте рекурсивный алгоритм заливки 4-х связной гранично-определенной области.
2. Добавьте на форму кнопку для очистки поля рисования.
3. Добавьте в проект возможность выбора цвета заливки.
4. Выведите на экран фигуру из лабораторной работы 2.
5. Модифицируйте данный программный модель так, чтобы можно было выбирать фигуру для заливки интерактивно (задавать точку затравки с помощью клика мышки).
6. Модифицируйте данный программный модуль так, чтобы можно было рисовать отрезки интерактивно на форме. Можно использовать, например, обработчики `PaintBox1MouseDown` (для задания начального конца отрезка и координат затравки) и `PaintBox1MouseUp` – для задания конечного конца отрезка.

### Задания для самостоятельного выполнения к лабораторной работе № 3

1. Добавьте в созданный проект следующие возможности: создайте обработчик (метод), реализующий алгоритм обхода сложного контура и алгоритм заливки. Реализуемый алгоритм заливки выбирайте по вариантам:  
*1 вариант* – простой итеративный алгоритм заливки с затравкой.  
*2 вариант* – построчный алгоритм заливки с затравкой.  
*3 вариант* – алгоритм закраски многоугольника
2. Программный модуль должен выводить исходный контур (сцена, состоящая из произвольного набора отрезков и фигур, полученная в лабораторной работе 2) на экран, затем контур должен медленно (по мере его обхода) закрашиваться другим цветом.
3. После обхода контура должно происходить закрашивание замкнутых областей.
4. Предусмотрите настройку цвета обхода контура и заливки.
5. Напишите метод, реализующий алгоритм отсекающего для сцены,

состоящей из набора отрезков. Реализуемый алгоритм выбирайте по вариантам:

**1 вариант** – двумерный алгоритм отсечения Козна-Сазерленда

**2 вариант** – простой алгоритм двумерного отсечения

6. На входе программный модуль получает координаты отсекающего окна и концов отрезков (предусмотрите интерактивное добавление пользователем отрезков). Затем выводятся отсекающее окно и отрезки на экран. После нажатия соответствующей кнопки, отсеченные части отрезков, расположенные вне окна, должны быть закрашены другим цветом.

7. Подготовьтесь к выполнению лабораторной работы № 4. Для этого ознакомьтесь с теоретическим материалом по теме лабораторной работы № 4

### **Контрольные вопросы к лабораторной работе № 3**

1. Что означают термины «гранично-определенная область» и «внутренне-определенная область»?
2. Объясните понятия «4-х связная» и «8-ми связная» область.
3. Чем отличаются простой и сложный контуры?
4. Опишите алгоритмы обхода простого и сложного контура.
5. Что такое «затравка»?
6. Опишите простой алгоритм заливки с затравкой.
7. Опишите кратко построчный алгоритм заливки с затравкой.
8. Опишите алгоритм закрашки многоугольника.
9. Что означают термины «полностью видимые», «полностью невидимые», «подозрительные» отрезки?
10. Кратко опишите основную идею отсечения отрезков алгоритмом Козна-Сазерленда.
11. Опишите простой алгоритм двумерного отсечения.



## Лабораторная работа № 4

### Построение 2D изображений. 2D аффинные преобразования

*Цель работы* – изучение и реализация алгоритмов аффинных преобразований фигур на плоскости, получение навыков моделирования двумерных объектов.

#### Краткая теория и порядок выполнения лабораторной работы

Для выполнения лабораторной работы ознакомьтесь с лекционным материалом по рассматриваемой теме.

Для задания модели двумерного объекта используем массив координат вершин фигур (назовем его «матрица тела»). Преобразования производятся умножением матрицы тела на матрицу преобразований, в результате чего получаем новые значения координат вершин фигуры.

#### Пример 4.1

Разработаем программный модуль для дискретного сдвига фигурки (квадрата) вверх-вниз, вправо-влево. Сдвиг фигурки будет происходить по нажатию кнопки.

1. Для вывода результатов создадим форму:

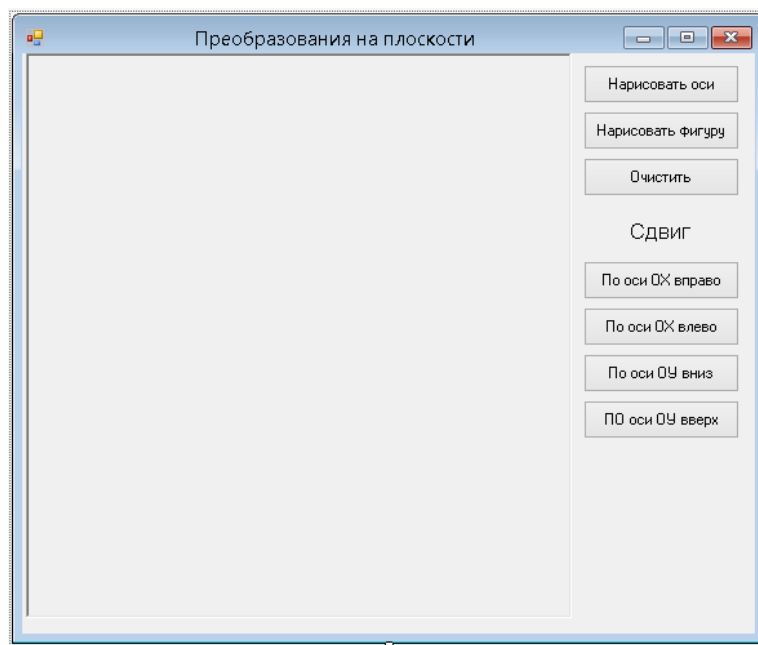


Рисунок 4.1

2. Зададим мировые координаты вершин квадрата:

$(-50, 0); (0, 50); (50, 0), (0, -50)$ .

Представим их в однородных координатах:

//инициализация матрицы тела

private void Init\_kvadrat()

```
{
    kv[0, 0] = -50;    kv[0, 1] = 0;    kv[0, 2] = 1;
    kv[1, 0] = 0;     kv[1, 1] = 50;   kv[1, 2] = 1;
    kv[2, 0] = 50;    kv[2, 1] = 0;    kv[2, 2] = 1;
    kv[3, 0] = 0;     kv[3, 1] = -50;  kv[3, 2] = 1;
}
```

Однородные  
координаты

Для перевода мировых координат в экранную систему координат, умножаем их на матрицу сдвига в центр *pictureBox*.

3. Зададим матрицу сдвига:

//инициализация матрицы сдвига

private void Init\_matr\_preob (int k1, int l1)

```
{
    matr_sdv[0,0]=1;    matr_sdv[0,1]=0;    matr_sdv[0,2]=0;
    matr_sdv[1,0]=0;    matr_sdv[1,1]=1;    matr_sdv[1,2]=0;
    matr_sdv[2,0]=k1;   matr_sdv[2,1]=l1;    matr_sdv[2,2]=1;
}
```

**Пояснения:** т.к. данная матрица будет использоваться дальше для получения двумерного преобразования «сдвиг по оси», определим компоненты матрицы, отвечающие за сдвиг, переменными *k1* и *l1*. Задавая разные значения этих переменных, можно получать разные варианты сдвига.

4. Для вывода координатных осей создадим массив мировых координат точек, принадлежащих осям:

//инициализация матрицы осей

private void Init\_osi()

```
{
    osi[0, 0] = -200;   osi[0, 1] = 0;    osi[0, 2] = 1;
    osi[1, 0] = 200;    osi[1, 1] = 0;    osi[1, 2] = 1;
    osi[2, 0] = 0;      osi[2, 1] = 200;   osi[2, 2] = 1;
    osi[3, 0] = 0;      osi[3, 1] = -200;  osi[3, 2] = 1;
}
```

Аналогично, для вывода осей на экран, необходимо переводить их в экранную систему координат, умножая на матрицу сдвига.

**Примечание:** используемые массивы задайте глобальными

```
int[,] kv = new int[4, 3]; // матрица тела
int[,] osi = new int[4, 3]; // матрица координат осей
int[,] matr_sdv = new int[3, 3]; // матрица преобразования
```

5. Создадим метод умножения матриц:

```
//умножение матриц
private int[,] Multiply_matr(int[,] a, int[,] b)
{
    int n = a.GetLength(0);
    int m = a.GetLength(1);
    int[,] r = new int[n, m];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            r[i, j] = 0;
            for (int ii = 0; ii < m; ii++)
            {
                r[i, j] += a[i, ii] * b[ii, j];
            }
        }
    }
    return r;
}
```

Входные параметры

Тип результата

**Пояснения:** входные параметры метода – две матрицы *a* и *b*, результат метода – матрица *r*, поэтому в заголовке метода задан тип возвращаемого значения – двумерный массив (int[,] *Multiply\_matr*); для передачи результата работы метода (умножения матриц) в другие методы программы в конце тела метода обязательно необходимо задать оператор *return r*

6. Создадим метод вывода фигуры на экран (*Draw\_Kv*). В данном методе:

6.1) инициализируем массив координат фигуры (*Init\_kvdrat*)

6.2) инициализируем матрицу сдвига (*Init\_matr\_preob(k, l)*)

- 6.3) умножаем матрицу тела на матрицу сдвига (*Multiply\_matr(kv, matr\_sdv)*)
- 6.4) задаем цвет и ширину карандаша для рисования
- 6.5) выводим стороны квадрата, используя стандартные методы C#
- 6.6) освобождаем все используемые ресурсы

```
//вывод квадрата на экран
private void Draw_Kv()
{
    Init_kvadrat(); //инициализация матрицы тела
    Init_matr_preob(k, l); //инициализация матрицы преобразования
    int [,] kv1 = Multiply_matr(kv, matr_sdv); //перемножение матриц

    Pen myPen = new Pen(Color.Blue, 2); // цвет линии и ширина

    //создаем новый объект Graphics (поверхность рисования) из pictureBox
    Graphics g = Graphics.FromHwnd(pictureBox1.Handle);
    // рисуем 1 сторону квадрата

    g.DrawLine(myPen, kv1[0, 0], kv1[0, 1], kv1[1, 0], kv1[1, 1]);
    // рисуем 2 сторону квадрата
    g.DrawLine(myPen, kv1[1, 0], kv1[1, 1], kv1[2, 0], kv1[2, 1]);
    // рисуем 3 сторону квадрата
    g.DrawLine(myPen, kv1[2, 0], kv1[2, 1], kv1[3, 0], kv1[3, 1]);
    // рисуем 4 сторону квадрата
    g.DrawLine(myPen, kv1[3, 0], kv1[3, 1], kv1[0, 0], kv1[0, 1]);

    g.Dispose(); // освобождаем все ресурсы, связанные с отрисовкой
    myPen.Dispose(); //освобождем ресурсы, связанные с Pen
}
```

**Примечание:** не забудьте задать глобальные переменные *k, l*

```
int k, l; // элементы матрицы сдвига
```

7. Вызовем процедуру *Draw\_Kv*. в обработчике события нажатия кнопки «Нарисовать фигуру».

```
//вывод квадратика в центре pictureBox
private void button2_Click(object sender, EventArgs e)
{
    //середина pictureBox
    k = pictureBox1.Width / 2;
    l = pictureBox1.Height / 2;
```

```

        //вывод квадрата в середине
        Draw_Kv();
    }

```

8. Выполним аналогичные действия для вывода осей координат на экран - создадим метод вывода осей на экран и обработчик события нажатия кнопки «Нарисовать оси»:

```

//вывод осей на экран
private void Draw_osi()
{
    Init_osi();
    Init_matr_preob(k, l);
    int[,] osi1 = Multiply_matr(osi, matr_sdv);

    Pen myPen = new Pen(Color.Red, 1); // цвет линии и ширина
    Graphics g = Graphics.FromHwnd(pictureBox1.Handle);

    // рисуем ось OX
    g.DrawLine(myPen, osi1[0, 0], osi1[0, 1], osi1[1, 0], osi1[1, 1]);

    // рисуем ось OY
    g.DrawLine(myPen, osi1[2, 0], osi1[2, 1], osi1[3, 0], osi1[3, 1]);

    g.Dispose();
    myPen.Dispose();
}

//вывод осей в центре pictureBox
private void button1_Click(object sender, EventArgs e)
{
    k = pictureBox1.Width / 2;
    l = pictureBox1.Height / 2;
    Draw_osi();
}

```

9. Выводим изображение (рис.4.2).

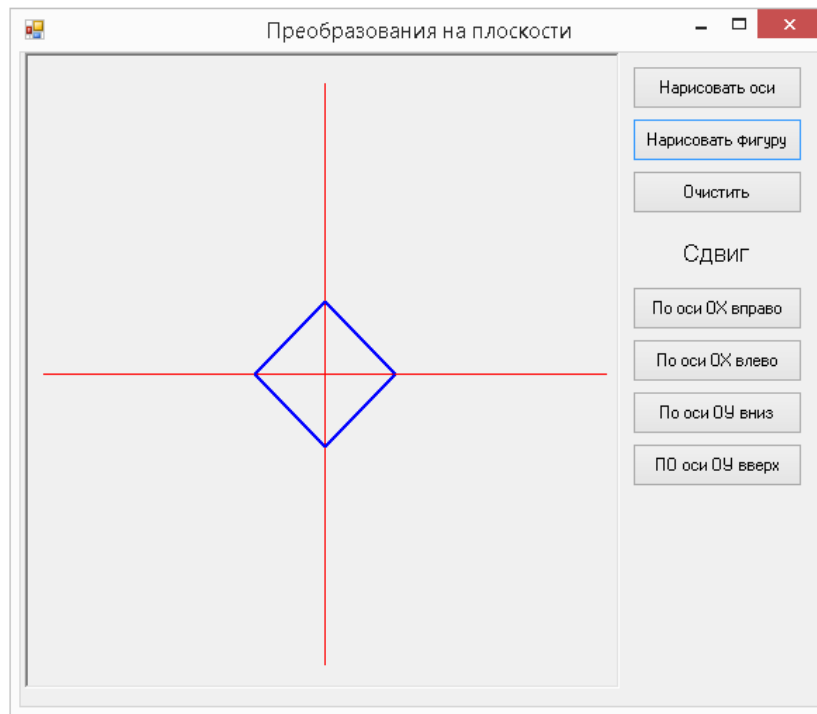


Рисунок 4.2

10. Далее выполняем аффинные преобразования (согласно заданию). Например, для сдвига квадрата право на 5 единиц достаточно увеличить параметр  $k$  матрицы сдвига, умножить матрицу тела на измененную матрицу сдвига и вывести фигуру на экран (кнопка «По оси OX вправо»):

```
//сдвиг вправо
private void button4_Click(object sender, EventArgs e)
{
    k += 5; //изменение соответствующего элемента матрицы сдвига
    Draw_Kv(); //вывод квадрата
}
```

Сделав несколько нажатий на кнопку «По оси OX вправо», получаем (рисунок 4.3):

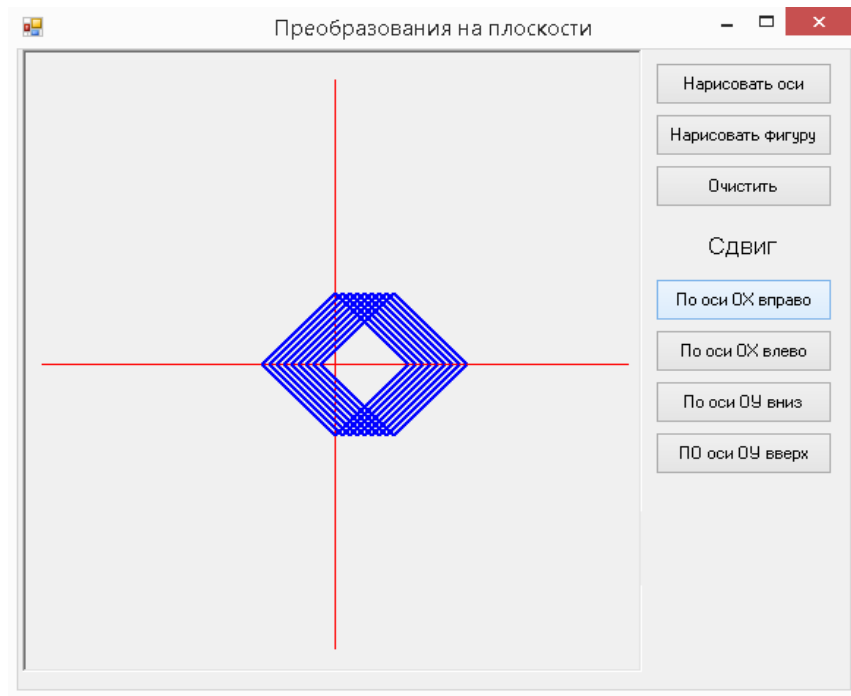


Рисунок 4.3

### Пример 4.2

Модифицируем пример 4.1 таким образом, чтобы смещение квадратика вправо происходило не дискретно (т.е. по нажатию кнопки), а непрерывно - автоматически после нажатия кнопки «Старт» (при этом название кнопки «Старт» сменится на «Стоп»). Для остановки перемещения необходимо нажать кнопку «Стоп».

Один из алгоритмов реализации данного задания можно представить так. Введем глобальную булевскую переменную ( $f$ ). Значение  $f$  при первоначальном выводе квадратика равняется *true*. Цикл перемещения будет происходить по таймеру, пока не нажмем кнопку «Стоп». При нажатии кнопки «Стоп» значению  $f$  присваивается противоположное значение и движение квадратика останавливается.

Для реализации смещения используем преобразование координат фигуры, описанное в примере 4.1: изменяем компоненты матрицы преобразования, отвечающие за сдвиг, умножаем матрицу тела на матрицу преобразования (в

данном случае, матрицу сдвига), получаем новые координаты фигуры и выводим фигуру на экран. Эти действия повторяем.

1. Добавим на форму кнопку *Button8* и *таймер* (рис.4.4).

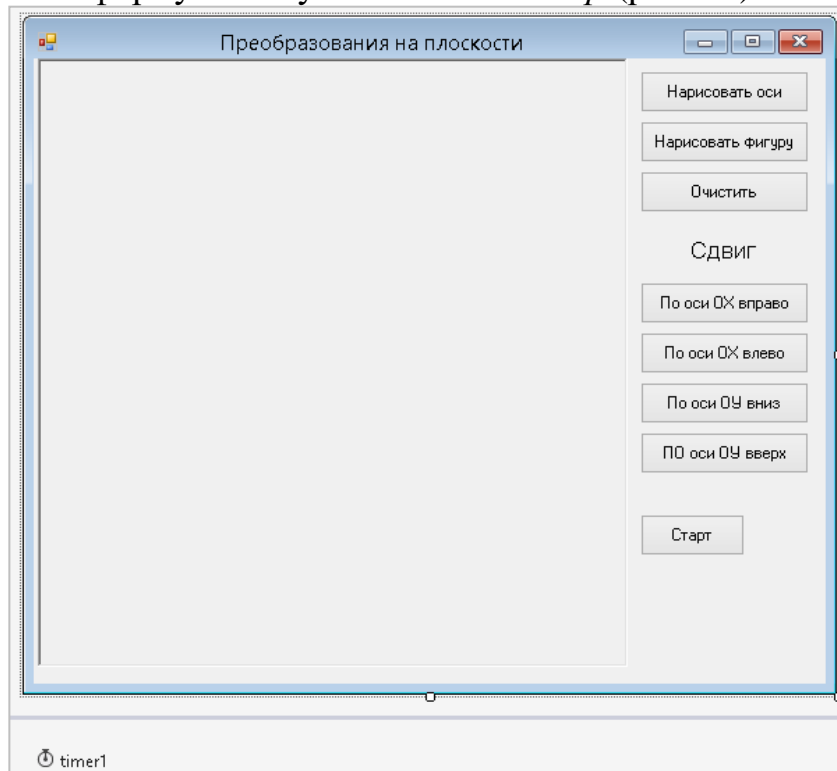


Рисунок 4.4

2. Напишем обработчик нажатия кнопки «Старт»:

```
//непрерывное перемещение
private void button8_Click(object sender, EventArgs e)
{
    timer1.Interval = 100;

    button8.Text = "Стоп";
    if (f == true)
        timer1.Start();
    else
    {
        timer1.Stop();
        button8.Text = "Старт";
    }
    f = !f;
}
```



**Пояснения:** меняем название кнопки на «Стоп». Если значение  $f$  равно «истина», то запускаем таймер. Иначе останавливаем таймер и меняем название кнопки. Значение переменной  $f$  заменяем на противоположное.

3. Напишем метод, который для каждого тика таймера производит сдвиг и перерисовку фигуры:

```
private void timer1_Tick_1(object sender, EventArgs e)
{
    k++;
    Draw_Kv();
    Thread.Sleep(100);
}
```

**Пояснения:**

- данный метод должен быть создан как обработчик единственного события (*Tick*) объекта *timer1*;

- в *C#* можно устанавливать задержку выполнения текущего потока на заданный интервал времени. Для этого необходимо сначала подключить *Threading* (*using System.Threading;*). Затем написать следующий код:

```
Thread.Sleep(100);
```

Время измеряется в миллисекундах.

4. Сохраните проект и запустите на исполнение. Нажмите на кнопку «Старт» - на экране должно появиться смещение квадрата. Нажмите «Стоп» - движение остановится. На экране должна появиться примерно такая форма (рисунок 4.5):

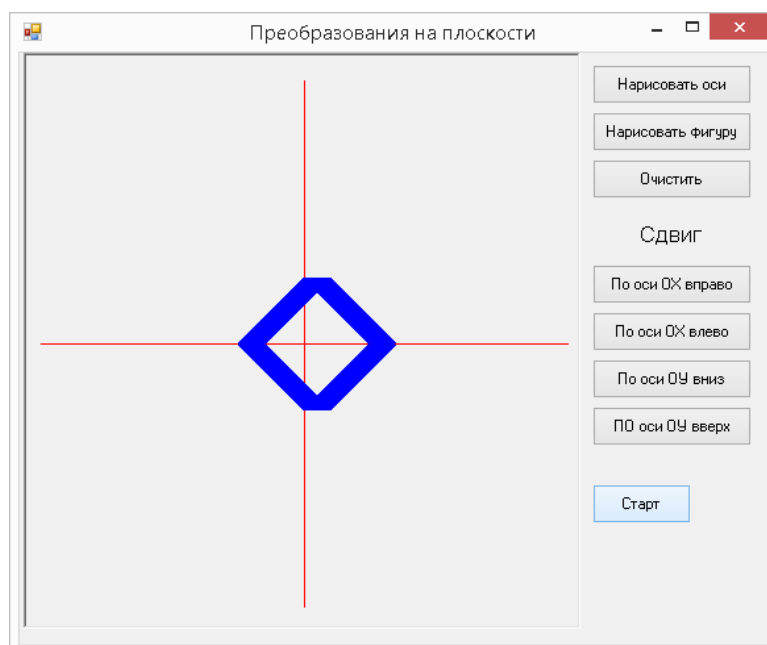


Рисунок 4.5

При повторном нажатии кнопки «Старт» движение продолжится с текущей позиции фигуры.

#### Задания к лабораторной работе № 4

1. Добавьте методы для остальных вариантов сдвига, как дискретно, так и непрерывно.
2. Добавьте обработчик нажатия кнопки для очистки поля рисования.
3. Модифицируйте все методы смещения фигуры так, чтобы на экране находилось только одно изображение фигуры (старых изображений фигуры не должно быть видно).
4. Постройте двумерное изображение фигуры, соответствующее заданию индивидуального варианта лабораторной работы 2. Начало координат экранной системы координат должно располагаться приблизительно в центре фигуры.
5. Реализуйте над фигурой все преобразования: сдвиг, отражение, масштабирование, поворот.
6. Добавьте в проект возможность выбора цвета и стиля линий.

### **Задания для самостоятельного выполнения к лабораторной работе № 4**

1. Добавьте в проект модуль, реализующий решение задачи в соответствии с вариантом.
2. В честь 55-летия ТУСУР отобразите в проекте фразу «55 лет ТУСУР».
3. Подготовьтесь к выполнению лабораторной работы № 5. Для этого ознакомьтесь с теоретическим материалом по теме лабораторной работы № 5.

### **Варианты индивидуальных заданий на самостоятельную работу к лабораторной работе № 4**

1. Написать программу, которая имитирует движение парусной лодки и полет чайки на фоне заката солнца (используя, операции масштабирования и смещения).
2. Написать программу, которая имитирует приближение бумажного самолетика, выпущенного из окна замка (используя, операции масштабирования и смещения).
3. Изобразить на экране правильный треугольник и пятиугольник, каждый из которых вращается вокруг своего центра. Фигуры расположены на расстоянии друг от друга. Начальное вращение происходит в противоположных направлениях. Предоставить возможность управления с клавиатуры скоростью и направлением вращения отдельно для треугольника и пятиугольника.
4. Изобразить на экране два вращающихся правильных треугольника разных размеров, один из которых лежит внутри другого. Начальное вращение происходит в противоположных направлениях. В процессе вращения меньший треугольник начинает расти (до заданного размера), а больший уменьшаться. Предоставить возможность управления с клавиатуры скоростью и направлением вращения отдельно для каждого треугольника.
5. Написать программу, которая имитирует движение велосипеда (вращение колес и педалей) и перемещение по экрану. Предусмотрите возможность превращать пользователю велосипед в тандем.
6. Изобразить шестиугольник, растущий из центра экрана. При этом изображение меньших размеров не стирать и каждое новое изображение

выводить развернутым относительно предыдущего на угол В (угол поворота задает пользователь). Каждый шестиугольник изобразите разным цветом.

7. Изобразить восьмиугольник, уменьшающийся с краев экрана. При этом изображение больших размеров не стирать и каждое новое изображение выводить развернутым относительно предыдущего на угол В (угол поворота задает пользователь). Каждый восьмиугольник изобразите разным цветом.

8. Написать программу, демонстрирующую различные виды отображения, произвольной фигуры (фигура задается интерактивно пользователем): относительно оси X, относительно оси Y, относительно прямой  $X=Y$ . Предусмотрите два варианта отображения: 1) каждое отображение происходит, исходя из начальных координат фигуры; 2) каждое отображение происходит, исходя из текущих координат фигуры. Каждое отображение изображайте разным цветом.

9. Написать программу, которая выполняет следующие преобразования над словом, написанным прописными буквами с помощью отдельных точек: масштабирование, перемещение, вращение, растягивание по диагонали (наклон букв). Слово вводится пользователем в указанном месте экрана.

10. Написать программу для отображения полета 2-х космических кораблей вокруг Земли. Каждый корабль совершает полет по своей орбите. Орбита представляет собой окружность. При перемещении корабля на нижний край экрана, он увеличивается в размере (имитация приближения корабля). При переходе на верхний край – уменьшается (имитация удаления корабля).

11. Написать программу для отображения полета 2-х космических кораблей вокруг Земли. Корабли совершают полет на одной орбите, но с разными скоростями и в разных направлениях. Орбита представляет собой окружность. При перемещении корабля на верхний край экрана, он увеличивается в размере (имитация приближения корабля). При переходе на нижний край – уменьшается (имитация удаления корабля).

12. Написать программу для отображения полета космического корабля вокруг центра. Центр указывается в интерактивном режиме. Маркер центра – треугольник. Орбита представляет собой окружность. При перемещении корабля на нижний край экрана, он увеличивается в размере (имитация

приближения корабля). При переходе на верхний край – уменьшается (имитация удаления корабля).

13. Написать программу, которая имитирует движение поезда (с пассажирским вагоном) с движущимися шатунами и шпалами методом перемещения фона. Задний фон имеет разные виды деревьев. В окнах вагонов происходит отображение меняющихся деревьев, имитирующее движение поезда.

14. Написать программу, которая демонстрирует вращение слова относительно точки, расположенной в центре средней буквы. Слово вводится с клавиатуры в любом месте экрана. Предусмотрите возможность задания размера шрифта и смену цвета букв слова случайным образом.

15. Написать следующую программу: в центре экрана расположена антенна (в качестве прообраза можно взять антенну – символ ТУСУР). Вокруг антенны происходит пульсация окружности с помощью операции масштабирования. Окружность должна увеличиваться в диаметре до тех пор, пока не достигнет границ экрана, затем она начинает сжиматься. Процесс должен циклически повториться, при этом необходимо обеспечить чередование цветов при увеличении и уменьшении диаметра окружности.

16. Написать программу, изображающую падающую в лужицу дождевую каплю и появляющиеся в результате этого расходящиеся круги (овалы) на поверхности воды. Предусмотреть возможность изменения количества и скорости падения капель.

#### **Контрольные вопросы к лабораторной работе № 4**

1. Что такое «однородные координаты»? Почему появилась необходимость использовать однородные координаты?
2. Опишите матрицы, используемые для 2D-преобразования.
3. Опишите процесс получения 2D-преобразований.

## Лабораторная работа № 5

### Векторно-полигональная и аналитическая модели объекта.

#### Преобразования в пространстве (3D-преобразования)

*Цель работы* – изучение и реализация алгоритмов построения и преобразования трехмерного объекта с использованием векторно-полигональной и аналитической моделей.

Для выполнения лабораторной работы ознакомьтесь с лекционным материалом по рассматриваемой теме.

#### Задания к лабораторной работе № 5

1. Создайте программный модуль, реализующий процедуры преобразований многогранника в пространстве.

2. Первым действием отобразите (поверните) многогранник на экране так, чтобы было видно его форму. Для отображения многогранника используйте ортогональную проекцию.

3. В программном модуле необходимо предусмотреть следующие возможности:

- перемещение многогранника (по каждой координатной оси, по двум или трем осям одновременно);
- отображение относительно координатных плоскостей;
- изменение размера многогранника (по каждой оси, по двум или трем координатным осям одновременно);
- вращение относительно заданной оси вращения;
- настройки цвета и стиля линий;
- изменение направления и скорости вращения / перемещения.

4. Нарисуйте систему координат и ось вращения.

5. Координаты точек – мировые. Максимальное и минимальное значение мировых координат изобразите на экране.

6. Отображаемую фигуру и ось вращения выберите в соответствии с вариантом (таблица 2). Описание фигур представлено в приложении Б:

Таблица 2 – Варианты заданий лабораторной работы 5

№ вар.	Фигура	Ось вращения
1.	Тетраэдр	Горизонтальная прямая, параллельная оси ОХ. Смещение от оси ОХ задается пользователем интерактивно.
2.	Тетраэдр	Прямая, под углом 45 градусов ко всем осям и проходящая через начало координат
3.	Тетраэдр	Вертикальная прямая, параллельная оси ОУ. Смещение от оси ОУ задается пользователем интерактивно.
4.	Тетраэдр	Прямая, определяемая произвольными направляющими косинусами и проходящая через произвольную точку
5.	Гексаэдр (куб)	Прямая под углом 45 градусов ко всем осям и проходящая через начало координат
6.	Гексаэдр (куб)	Горизонтальная прямая, параллельная оси ОХ. Смещение от оси ОХ задается пользователем интерактивно.
7.	Гексаэдр (куб)	Вертикальная прямая, параллельная оси ОУ. Смещение от оси ОУ задается пользователем интерактивно.
8.	Гексаэдр (куб)	Прямая, определяемая произвольными направляющими косинусами и проходящая через произвольную точку
9.	Тригональная бипирамида	Вертикальная прямая, параллельная оси ОУ. Смещение от оси ОУ задается пользователем интерактивно.
10.	Тригональная бипирамида	Прямая под углом 45 градусов ко всем осям и проходящая через начало координат
11.	Тригональная бипирамида	Горизонтальная прямая, параллельная оси ОХ. Смещение от оси ОХ задается пользователем интерактивно.
12.	Тригональная бипирамида	Прямая, определяемая произвольными направляющими косинусами и проходящая через произвольную точку
13.	Октаэдр	Горизонтальная прямая, параллельная оси ОХ. Смещение от оси ОХ задается пользователем интерактивно.
14.	Октаэдр	Вертикальная прямая, параллельная оси ОУ. Смещение от оси ОУ задается пользователем интерактивно.

## Окончание таблицы 2

№ вар.	Фигура	Ось вращения
15.	Октаэдр	Прямая под углом 45 градусов ко всем осям и проходящая через начало координат
16.	Октаэдр	Высота фигуры
17.	Пятиугольная призма (карандаш)	Вертикальная прямая, параллельная оси ОУ. Смещение от оси ОУ задается пользователем интерактивно.
18.	Пятиугольная призма (карандаш)	Прямая под углом 45 градусов ко всем осям и проходящая через начало координат
19.	Пятиугольная призма (карандаш)	Высота фигуры (проходит через центр пятиугольника)
	Призматокд	Высота фигуры (проходит через центр пятиугольника)
20.	Призматокд	Прямая под углом 45 градусов ко всем осям и проходящая через начало координат
21.	Призматокд	Горизонтальная прямая, параллельная оси ОХ. Смещение от оси ОХ задается пользователем интерактивно.
22.	Призматокд	Прямая, определяемая произвольными направляющими косинусами и проходящая через произвольную точку

**Контрольные вопросы к лабораторной работе № 5**

1. Что такое векторно-полигональная модель объекта?
2. Опишите способы представления векторной полигональной модели
3. Опишите матрицу преобразований в пространстве общего вида. Раскройте назначение каждого элемента.
4. Опишите матрицы, используемые для 3D-преобразования.
5. Опишите процесс получения 3D-преобразований.
6. Что такое «аналитическая модель»?
7. Опишите процедуру отображения графика трехмерной функции на экране.
8. Опишите процедуры получения вращения, масштабирования и перемещения полученной поверхности.

**Задания для самостоятельного выполнения к лабораторной работе № 5**

1. Создайте программный модуль, реализующий следующее:



- построение на экране графика поверхности - трехмерной функции.
- Вид функции выбирается по вариантам (табл. 3):

Таблица 3

№ вар.	Вид функции	Диапазон изменения x	Диапазон изменения y
1	$R = x^2 + y^2 \quad Z = \frac{\sin(R)}{R}$	[-3; 3]	[-3; 3]
2	$Z = \sin^2(x) + \sin^2(y)$	[-3; 3]	[-3; 3]
3	$Z = (\sin(x) + \cos(y))^2$	[-3; 3]	[-3; 3]
4	$Z = \sin^2(x) + \cos(y)$	[-2; 2]	[-2; 2]
5	$Z = \sin(x) + \cos^2(y)$	[-2; 2]	[-2; 2]
6	$Z = x^2 - y^2 - 100$	[-3; 3]	[-3; 3]
7	$Z = e^{\sin(x) - y^2}$	[-3; 3]	[-3; 3]
8	$Z = e^{\sin(y) - x^2}$	[-3; 3]	[-3; 3]
9	$Z = e^{\cos(y) - x^2}$	[-3; 3]	[-3; 3]
10	$Z = e^{\sin(x) + y^2}$	[-3; 3]	[-3; 3]
11	$Z = e^{\sin(x) - \cos(y)}$	[-3; 3]	[-3; 3]
12	$Z = e^{\sin(x) + \cos(y)}$	[-3; 3]	[-3; 3]
13	$Z = e^{\sin(x) * \cos(y)}$	[-3; 3]	[-3; 3]
14	$Z = \cos(x^2 - y^2)$	[-2; 2]	[-2; 2]
15	$Z = \sin(x^2 - y^2)$	[-2; 2]	[-2; 2]

- преобразование данного графика – вращение, перемещение и масштабирование

*Пояснения:* на форму рекомендуется добавить управляющие компоненты - например, кнопки, полосы прокрутки и т.п. - для изменения направления или скорости преобразования;

- настройку цвета и стиля линий графика.

## Лабораторная работа № 6

### Преобразования в пространстве. Проекции. Удаление невидимых линий

*Цель работы* – изучение и реализация алгоритмов построения проекции фигуры, удаления невидимых линий и граней.

Для выполнения лабораторной работы ознакомьтесь с лекционным материалом по рассматриваемой теме.

### Задания к лабораторной работе № 6

1. Напишите программный модуль для отображения результатов преобразований многогранника в пространстве с удалением невидимых частей. Для изображения многоугольника используйте каркасную модель. Невидимые ребра должны быть изображены пунктирными линиями.

2. Первым действием отобразите (поверните) многогранник на экране так, чтобы было видно его форму.

3. Реализуйте:

- построение оси вращения;
- поворот многогранника вокруг заданной оси вращения.

4. Реализуйте следующие функции:

- перемещение многогранника (по каждой координатной оси, по двум или трем осям одновременно);
- изменение размера многогранника (по каждой оси, по двум или трем координатным осям одновременно);
- настройку цвета и толщины линий;
- изменение направления и скорости вращения / перемещения.

5. Нарисуйте систему координат и ось вращения.

6. Координаты точек – мировые.

7. Отображаемую фигуру, ось вращения, вид проекции и алгоритм удаления невидимых линий выберите в соответствии с вариантом (таблица 6.1):

Таблица 6.1

№	Фигура	Проекция	Ось вращения	Алгоритм удаления невидимых линий
1.	Октаэдр	Диметрия	Прямая под углом 45 градусов ко всем координатным осям и проходящая через начало координат	алгоритм Робертса
2.	Тетраэдр	Диметрия	Прямая под углом 45 градусов ко всем координатным осям и проходящая через начало координат	алгоритм, использующий нормаль грани
3.	Тригональная бипирамида	Перспектива (2 точки схода)	Вертикальная прямая, параллельная оси ОУ. Смещение от оси ОУ задается пользователем интерактивно	алгоритм Робертса
4.	Тетраэдр	Перспектива (1 точка схода)	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм, использующий нормаль грани
5.	Октаэдр	Перспектива (1 точка схода)	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм Робертса
6.	Гексаэдр (куб)	Диметрия	Прямая под углом 45 градусов ко всем координатным осям и проходящая через начало координат	алгоритм, использующий нормаль грани
7.	Пятиугольная призма (карандаш)	Перспектива (1 точка схода)	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм Робертса
8.	Тригональная бипирамида	Изометрия	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм, использующий нормаль грани
9.	Октаэдр	Перспектива (2 точки схода)	Горизонтальная прямая, параллельная оси ОХ. Смещение от оси ОХ задается пользователем интерактивно	алгоритм Робертса
10.	Тригональная бипирамида	Диметрия	Прямая под углом 45 градусов ко всем координатным осям и проходящая через начало координат	алгоритм, использующий нормаль грани

Продолжение таблицы 6.1

№	Фигура	Проекция	Ось вращения	Алгоритм удаления невидимых линий
11.	Пятиугольная призма (карандаш)	Диметрия	Прямая под углом 45 градусов ко всем координатным осям и проходящая через начало координат	алгоритм Робертса
12.	Тригональная бипирамида	Перспектива (1 точка схода)	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм, использующий нормаль грани
13.	Гексаэдр (куб)	Изометрия	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм Робертса
14.	Октаэдр	Изометрия	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм, использующий нормаль грани
15.	Гексаэдр (куб)	Перспектива (2 точки схода)	Вертикальная прямая, параллельная оси ОУ. Смещение от оси ОУ задается пользователем интерактивно	алгоритм Робертса
16.	Октаэдр	Перспектива (2 точки схода)	Горизонтальная прямая, параллельная оси ОХ. Смещение от оси ОХ задается пользователем интерактивно	алгоритм, использующий нормаль грани
17.	Пятиугольная призма (карандаш)	Изометрия	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм Робертса
18.	Гексаэдр (куб)	Изометрия	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм, использующий нормаль грани
19.	Пятиугольная призма (карандаш)	Перспектива (2 точки схода)	Вертикальная прямая, параллельная оси ОУ. Смещение от оси ОУ задается пользователем интерактивно	алгоритм Робертса
20.	Гексаэдр (куб)	Перспектива (1 точка схода)	Прямая, заданная произвольными направляющими векторами и проходящая через начало координат	алгоритм, использующий нормаль грани

Окончание таблицы 6.1

№	Фигура	Проекция	Ось вращения	Алгоритм удаления невидимых линий
21.	Икосаэдр	Изометрия	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм Робертса
22.	Икосаэдр	Диметрия	Прямая под углом 45 градусов ко всем координатным осям и проходящая через начало координат	алгоритм, использующий нормаль грани
23.	Икосаэдр	Перспектива (2 точки схода)	Горизонтальная прямая, параллельная оси ОХ. Смещение от оси ОХ задается пользователем интерактивно	алгоритм Робертса
24.	Икосаэдр	Перспектива (1 точка схода)	Прямая, заданная произвольными направляющими косинусами и проходящая через начало координат	алгоритм, использующий нормаль грани

### Контрольные вопросы к лабораторной работе № 6

1. Что такое проецирование? Для чего оно используется.
2. Опишите существующую классификацию видов проекций.
3. В чем основное различие параллельной и перспективной проекции.
4. Опишите матрицы проецирования.
5. Опишите процедуру получения вращения трехмерной фигуры на экране.

## Лабораторная работа № 7

### Введение в OpenGL

*Цель работы* – изучение принципов применения библиотеки OpenGL при разработке приложений в C#.

### OpenGL

*OpenGL* означает *Open Graphics Library*, что переводится как «открытая графическая библиотека».

Другими словами, *OpenGL* – это некая спецификация, включающая в себя несколько сотен функций. Она определяет независимый от языка программирования кросс-платформенный программный интерфейс, с помощью которого программист может создавать приложения, использующие двухмерную и трехмерную компьютерную графику. Первая базовая версия *OpenGL* появилась в 1992 году, она была разработана компанией *Silicon Graphics Inc*, занимающейся разработками в области трехмерной компьютерной графики.

В библиотеку заложен механизм расширений, благодаря которому производители аппаратного обеспечения (например, производители видеокарт) могли выпускать расширения *OpenGL* для поддержки новых специфических возможностей, не включенных в текущую версию библиотеки. Благодаря этому программисты могли сразу использовать эти новые возможности в отличие от библиотеки *Microsoft Direct3D* (в этом случае им бы пришлось ждать выхода новой версии *DirectX*).

Библиотеки *OpenGL* и *DirectX* являются конкурентами на платформе *MS Windows*. *Microsoft* продвигает свою библиотеку *DirectX* и стремится замедлить развитие библиотеки *OpenGL*, что ослабило бы графическую систему конкурирующих ОС, где используется исключительно библиотека *OpenGL* для реализации вывода всей графики.

Мы будем учиться визуализации компьютерной графики именно с применением этой библиотеки. Однако прямой поддержки данной библиотеки в *.NET Framework* нет, поэтому мы будем использовать библиотеку *Tao Framework*.

### ***TAO Framework***

*Tao Framework* – это свободно распространяемая библиотека с открытым исходным кодом, предназначенная для быстрой и удобной разработки кросс-платформенного мультимедийного программного обеспечения в среде *.NET Framework* и *Mono*.

На сегодняшний день *Tao Framework* – оптимальный путь для использования библиотеки *OpenGL* при разработке приложений в среде *.NET* на языке *C#*.

В состав библиотеки на данный момент входят все современные средства, которые могут понадобиться в ходе разработки мультимедиа программного обеспечения: реализация библиотеки *OpenGL*, реализация библиотеки *FreeGlut*, содержащей все самые новые функции этой библиотеки, библиотека *DevIL* (легшая в основу стандарта *OpenIL – Open Image Library*) и многие другие.

Самые интересные библиотеки, включенные в *Tao Framework*:

- *OpenGL 2.1.0.12* – свободно распространяемый аппаратнопрограммный интерфейс для визуализации 2D- и 3D-графики.
- *FreeGLUT 2.4.0.2* – библиотека с открытым исходным кодом, являющаяся альтернативой библиотеке *GLUT (OpenGL Utility Toolkit)*.
- *DevIL 1.6.8.3* (она же *OpenIL*) – кросс-платформенная библиотека, реализующая программный интерфейс для работы с изображениями. На данный момент библиотека поддерживает работу с изображениями 43 форматов для чтения и 17 форматов для записи.
- *Cg 2.0.0.0* – язык высокого уровня, созданный для программирования текстурных и вершинных шейдеров.
- *OpenAL 1.1.0.1* – свободно распространяемый аппаратно-программный интерфейс для обработки аудиоданных. (В том числе 3D-звука и EAX эффектов).
- *PhysFS 1.0.1.2* – библиотека для работы с вводом/выводом файловой системы, а также различного вида архивами на основе собственного *API*.
- *SDL 1.2.13.0* – кросс-платформенная мультимедийная библиотека, активно используемая для написания мультимедийных приложений в операционной системе *GNU/Linux*.

- *ODE 0.9.0.0* – свободно распространяемый физический программный интерфейс, главной особенностью которого является реализация системы динамики абсолютно твёрдого тела и системы обнаружения столкновений.

- *FreeType 2.3.5.0* – библиотека, реализующая растеризацию шрифтов. Данная библиотека используется в *X11* – оконной системе, которая обеспечивает все стандартные инструменты и протоколы для построения *GUI* (графического интерфейса пользователя) в *UNIX* подобных операционных системах.

- *FFmpeg 0.4.9.0* – набор свободно распространяемых библиотек с открытым исходным кодом. Данные мультимедийные библиотеки позволяют работать с аудио- и видеоданными в различных форматах.

Таким образом, библиотека *Tao Framework* является мощным и удобным свободно распространяемым инструментом для решения любых мультимедийных задач, преимущественно кросс-платформенного характера. Работая с данной библиотекой, разработчики могут использовать базу алгоритмов и реализованных за многие годы методов, что сокращает время разработки программных продуктов.

### **Создание проекта и подключение библиотеки *Tao OpenGL* в *C#***

Сначала создайте новый проект, в качестве шаблона установив приложение Windows Forms. Назовите его, например, OpenGLTest.

Дождитесь, пока MS Visual Studio закончит генерацию кода шаблона.

Назовите главное окно «Примитивы OpenGL».

Теперь перейдите к окну Solution Explorer (Обозреватель решений). Здесь нас интересует узел References (Ссылки), который отображает связи с библиотеками, необходимыми для работы нашего приложения (рисунок 7.1).



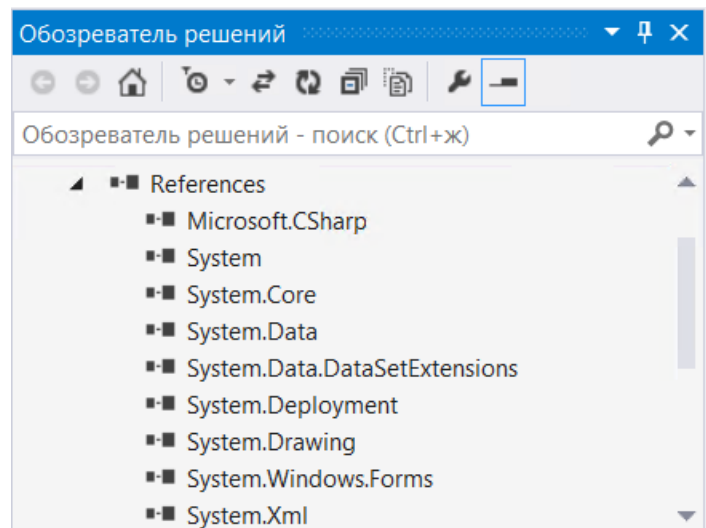
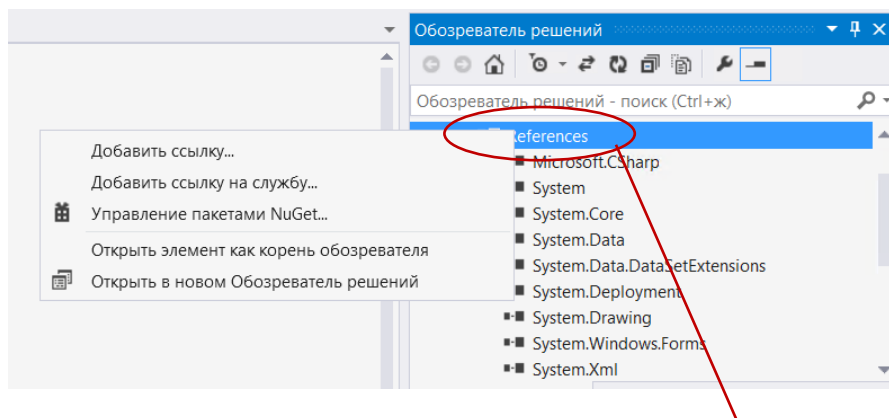


Рисунок 7.1

Щелкните по этому узлу правой клавишей мыши, после чего в открывшемся контекстном меню выберите «Добавить ссылку» (“Add Link”), как показано на рисунок 7.2.



Щелчок правой кнопкой мыши

Рисунок 7.2

В открывшемся окне «Добавить ссылку» перейдите к закладке «Обзор» и нажмите кнопку «Обзор». После этого перейдите к директории, в которую была установлена библиотека *Tao Framework*. (В нашем случае это папка “C:\Program Files (x86)\Microsoft Visual Studio 12.0\OpenGL”).

Нам потребуется папка *bin*, в ней хранятся необходимые нам библиотеки.

Перейдите в папку *bin* и выберите три библиотеки: ***Tao.OpenGL.dll***, ***Tao.FreeGlut.dll***, ***Tao.Platform.Windows.dll***

*Tao.OpenGL.dll* отвечает за реализацию библиотеки *OpenGL*.

*Tao.FreeGlut.dll* отвечает за реализацию функций библиотеки *Glut*. Мы будем ее использовать для инициализации рендера, а также для других целей.

*Tao.Platform.Windows.dll* отвечает за поддержку элементов для визуализации на платформе *Windows*.

На рисунке 7.3 мы видим все добавившиеся библиотеки в узле «References» (Ссылки).

Теперь перейдите к исходному коду окна. Для работы с нашими библиотеками необходимо подключить соответствующие пространства имен:

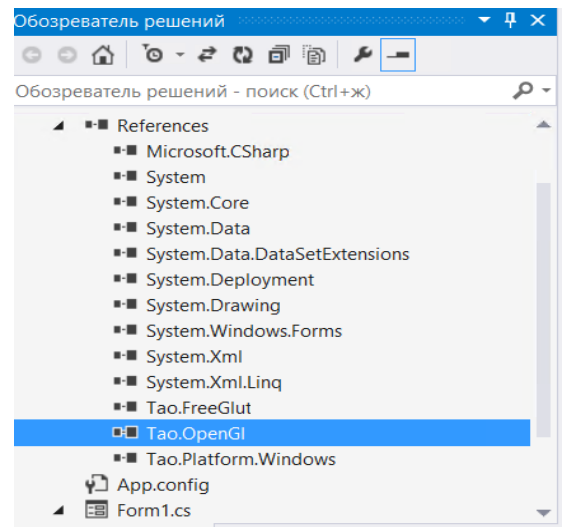


Рисунок 7.3

```
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 System.Threading;
// для работы с библиотекой OpenGL
using Tao.OpenGl;
// для работы с библиотекой FreeGLUT
using Tao.FreeGlut;
// для работы с элементом управления SimpleOpenGLControl
using Tao.Platform.Windows;
```

Теперь вернитесь к конструктору диалогового окна и перейдите к Панели элементов. Щелкните правой кнопкой на вкладке «Общие» и в раскрывшемся контекстном меню выберите пункт «Выбрать элементы» (Choose Items), как показано на рисунке 7.4.

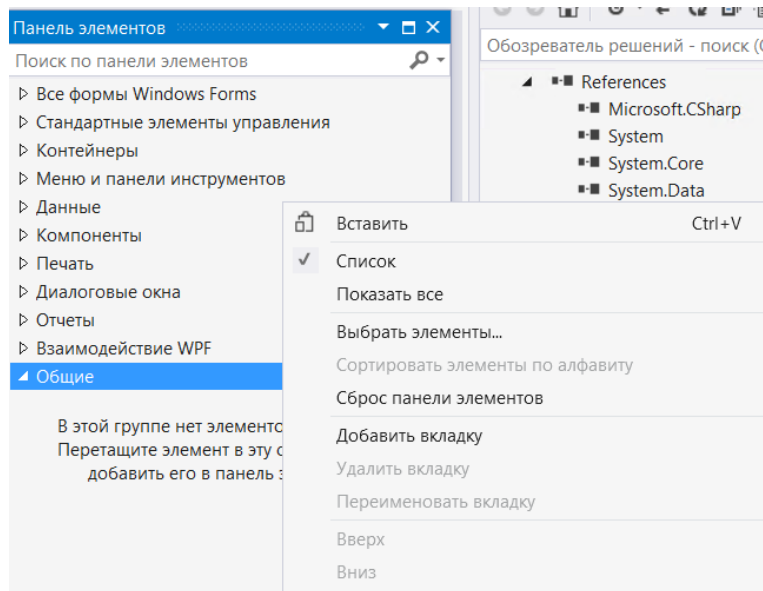


Рисунок 7.4

В открывшемся окне найдите элемент *SimpleOpenGLControl* и установите возле него галочку, как показано на рисунке 7.5. Если в представленном списке элементов нет элемента *SimpleOpenGLControl*, нажмите кнопку «Обзор» и в появившемся окне выберите и откройте библиотеку *Tao.Platform.Windows.dll*. Затем выберите галочкой *SimpleOpenGLControl* и нажмите ОК.

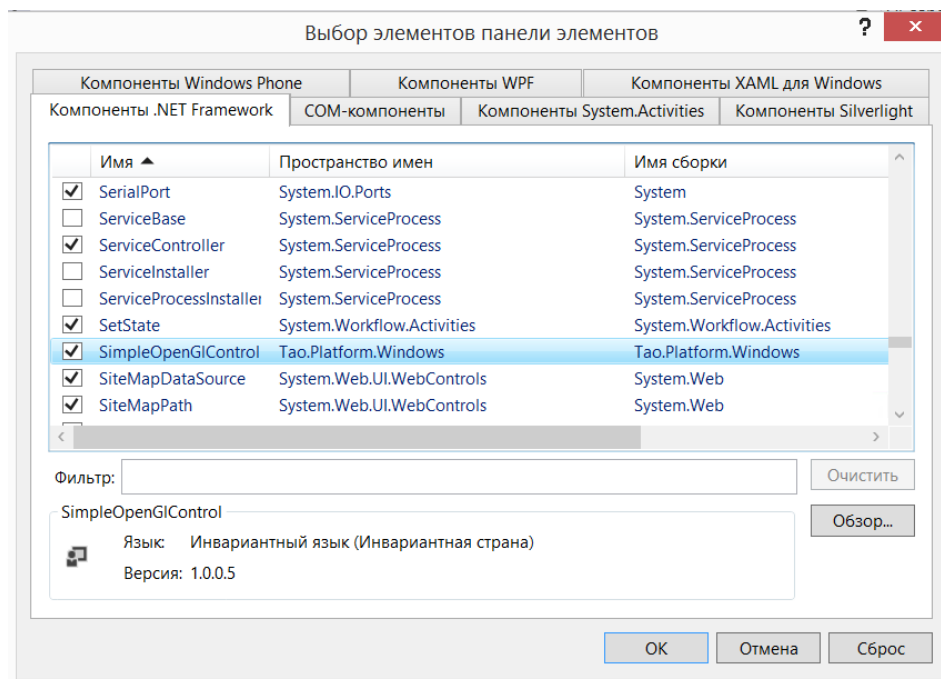


Рисунок 7.5

Теперь данный элемент станет доступным для размещения на форме приложения. Перетащите элемент на форму и разместите так, как показано на рисунке 7.6. Справа от размещенного элемента установите необходимые элементы.

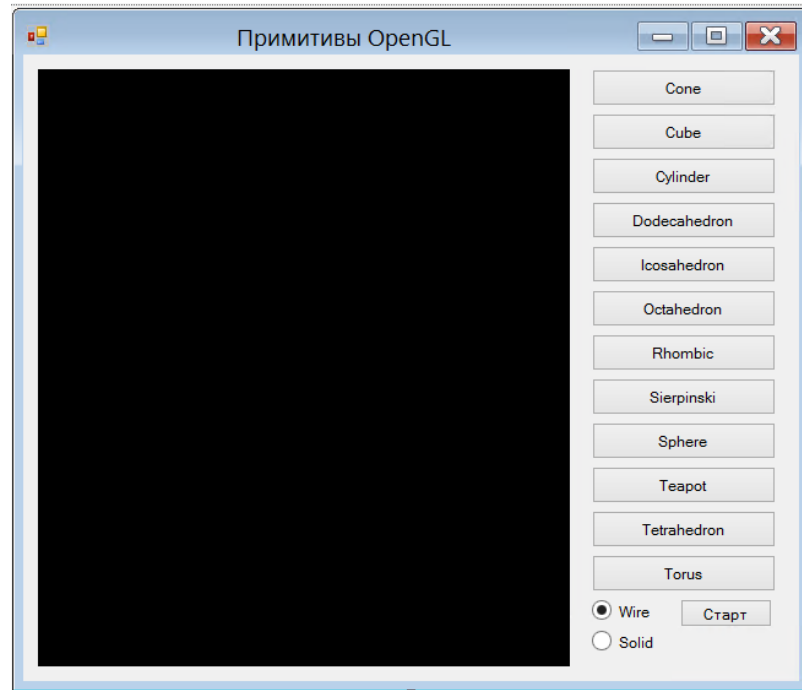


Рисунок 7.6

Выделите элемент *simpleOpenGLControl1*, расположенный на форме и перейдите к его свойствам. Измените параметр *name* на значение «Pr». Далее элементы *simpleOpenGLControl* будем называть *Pr*.

### Инициализация *OpenGL* и *Glut* в *C#*

Теперь необходимо инициализировать работу *OpenGL*. Сначала в конструкторе класса необходимо инициализировать работу элемента *Pr*:

```
public Form1()
{
    InitializeComponent();
    // Инициализация контекста окна графического вывода
    Pr.InitializeContexts();
}
```

Снова перейдите к конструктору и сделайте двойной щелчок левой клавишей мыши на форме – создастся функция обработчик события загрузки формы.

В ней поместим код инициализации *OpenGL* и *Glut*. Подробное описание кода визуализации объекта рассмотрено в приложении Г.

```
private void Form1_Load(object sender, EventArgs e)
{
    // Черный цвет фона
    Gl.glClearColor(0, 0, 0, 1);
    // Инициализация Glut
    Glut.glutInit();
    // Используем в Glut систему цветов RGB, двойную буферизацию
    // (экранный и внеэкранный буферы) и буфер глубины
    Glut.glutInitDisplayMode(Glut.GLUT_RGB | Glut.GLUT_DOUBLE |
    Glut.GLUT_DEPTH);
    // Активируем тест глубины
    Gl.glEnable(Gl.GL_DEPTH_TEST);
}
```

Откомпилируйте и запустите приложение.

Если при компиляции приложения *Microsoft Visual Studio* выдает ошибку с сообщением о том, что не найдена какая-либо библиотека *OpenGL* (например, *freeglut.dll*), необходимо в настройках операционной системы в переменную среды *PATH* добавить каталог данной библиотеки. При невозможности это сделать, допустимо из каталога *C:\Program Files (x86)\Microsoft Visual Studio 12.0\OpenGL\lib* скопировать все файлы необходимых библиотек (в нашем случае – *freeglut.dll*) в каталог *Debug* нашего проекта.

### Реализация программы вывода примитивов библиотеки Glut

Реализуем программный код вывода стандартных примитивов:

- конус;
- куб;
- цилиндр;
- додекаэдр;
- икосаэдр;
- октаэдр;
- ромбический додекаэдр;
- губка Серпиского;
- сфера;

- чайник;
- тетраэдр;
- тор.

Объекты представлены в библиотеке *FreeGlut* как полигональные модели. Их можно отобразить в виде каркаса (*glutWire*) либо залитыми цветом (*glutSolid*) или текстурой (*glutSolid* + текстура, в данном случае необходимо еще подключить библиотеку *DevIL.dll*).

Программа обеспечивает вывод любого из перечисленных выше представлений.

При проецировании используется изометрия, получаемая в результате умножения единичной матрицы на матрицы поворота по часовой стрелки на  $30^\circ$  и против нее на  $45^\circ$  соответственно вокруг осей X и Y:

```
// Матрица проецирования
Gl.glMatrixMode(Gl.GL_PROJECTION);
Gl.glLoadIdentity();
Gl.glRotated(-30, 1, 0, 0);
Gl.glRotated(45, 0, 1, 0);
```

Видовая матрица преимущественно является единичной:

```
// Видовая матрица
Gl.glMatrixMode(Gl.GL_MODELVIEW);
Gl.glLoadIdentity();
```

При выводе додекаэдра единичная видовая матрица умножается на матрицу масштабирования:

```
Gl.glScaled(0.5, 0.5, 0.5);
```

При выводе тонированных тел (*glutSolid*) используется модель освещенности с одним источником света. Нормали к *glut*-объектам генерируются автоматически.

Заметим, что Платоновы тела выводятся без сглаживания, даже если указан

```
Gl.glShadeModel(Gl.GL_SMOOTH); // Вывод с интерполяцией цветов
```

Примитив выводится после нажатия на одну из кнопок.

В зависимости от положения переключателя (*Wire* или *Solid*) отображается либо каркасная, либо тоновая, либо текстурированная модель (рисунок 7.7).

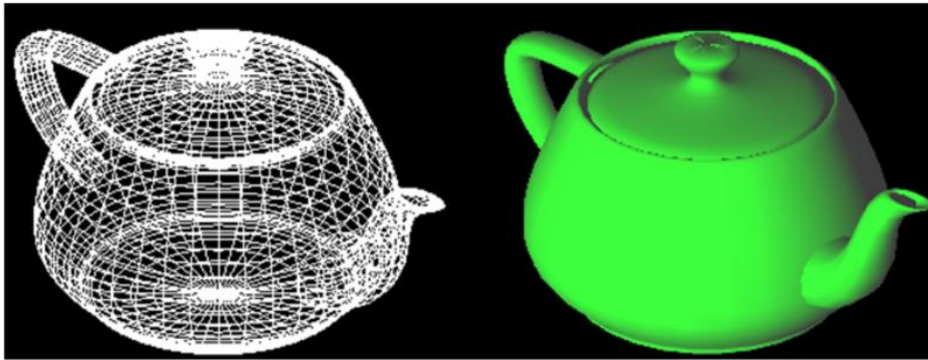


Рисунок 7.7 Два способа вывода примитива

Полный код приложения:

```
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 System.Threading;

using Tao.OpenGl;
using Tao.FreeGlut;
using Tao.Platform.Windows;

namespace OGL_Primet
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            // Инициализация контекста окна графического вывода
            Pr.InitializeContexts();
        }

        private void showSolid(int obj)
        {
            switch (obj)
            {
                case 1:
                    Glut.glutSolidCone(0.2, 0.75, 16, 8); break; // Конус
                case 2:
                    Glut.glutSolidCube(0.75); break; // Куб
            }
        }
    }
}
```

```

        case 3:
            Glut.glutSolidCylinder(0.2, 0.75, 16, 16); break; //Цилиндр
        case 4:
            Gl.glScaled(0.5, 0.5, 0.5);
            Glut.glutSolidDodecahedron(); break; // Додекаэдр
        case 5:
            Glut.glutSolidIcosahedron(); break; // Икосаэдр
        case 6:
            Glut.glutSolidOctahedron(); break; // Октаэдр
        case 7:
            Glut.glutSolidRhombicDodecahedron(); break; // Ромбический
додекаэдр
        case 8:
            double[] offset = { 0.0 };
            Glut.glutSolidSierpinskiSponge(7, offset, 1); break; //
Фрактал Губка Серпиского
        case 9:
            Glut.glutSolidSphere(0.75, 16, 16); break; // Сфера
        case 10:
            Glut.glutSolidTeapot(0.5); break; // Чайник
        case 11:
            Gl.glRotated(180, 0, 1, 0);
            Glut.glutSolidTetrahedron(); break; // Тетраэдр
        case 12:
            Glut.glutSolidTorus(0.15, 0.65, 16, 16); break; // Тор
    }
}

private void draw(int obj)
{
    // Очистка буфера цвета и буфера глубины
    Gl.glClear(Gl.GL_COLOR_BUFFER_BIT | Gl.GL_DEPTH_BUFFER_BIT |
Gl.GL_ACCUM_BUFFER_BIT);
    // Матрица проецирования
    Gl.glMatrixMode(Gl.GL_PROJECTION);
    Gl.glLoadIdentity();
    Gl.glRotated(-30, 1, 0, 0);
    Gl.glRotated(45, 0, 1, 0);
    // Видовая матрица
    Gl.glMatrixMode(Gl.GL_MODELVIEW);
    Gl.glLoadIdentity();
    if (radioButton1.Checked)
    {
        // Белый цвет
        Gl.glColor3f(1, 1, 1);
        // Выводим glut-примитив в виде каркаса
        switch (obj)
        {
            case 1:
                Glut.glutWireCone(0.2, 0.75, 16, 8); break; // Конус
            case 2:
                Glut.glutWireCube(0.75); break; // Куб
            case 3:
                Glut.glutWireCylinder(0.2, 0.75, 16, 16); break; //
Цилиндр
            case 4:
                Gl.glScaled(0.5, 0.5, 0.5);
                Glut.glutWireDodecahedron(); break; // Додекаэдр

```



```

        case 5:
            Glut.glutWireIcosahedron(); break; // Икосаэдр
        case 6:
            Glut.glutWireOctahedron(); break; // Октаэдр
        case 7:
            Glut.glutWireRhombicDodecahedron(); break; //
Ромбический додекаэдр
        case 8:
            double[] offset = { 0, 0, 0 };
            Glut.glutWireSierpinskiSponge(7, offset, 1); break; //
Фрактал Губка Серпиского
        case 9:
            Glut.glutWireSphere(0.75, 16, 16); break; // Сфера
        case 10:
            Glut.glutWireTeapot(0.5); break; // Чайник
        case 11:
            Gl.glRotated(180, 0, 1, 0);
            Glut.glutWireTetrahedron(); break; // Тетраэдр
        case 12:
            Glut.glutWireTorus(0.15, 0.65, 16, 16); break; // Тор
    }
}
else if (radioButton2.Checked)
{
    // Модель освещенности с одним источником цвета
    float[] light_position = { 10, 10, -30, 0 }; // Координаты
источника света
    float[] lghtClr = { 1, 1, 1, 0 }; // Источник излучает белый
цвет
    float[] mtClr = { 0, 1, 0, 0 }; // Материал зеленого цвета
    Gl.glPolygonMode(Gl.GL_FRONT, Gl.GL_FILL); // Заливка полигонов
    Gl.glShadeModel(Gl.GL_SMOOTH); // Вывод с интерполяцией цветов
    Gl.glEnable(Gl.GL_LIGHTING); // Будем рассчитывать освещенность
    Gl.glLightfv(Gl.GL_LIGHT0, Gl.GL_POSITION, light_position);
    Gl.glLightfv(Gl.GL_LIGHT0, Gl.GL_AMBIENT, lghtClr); //
Рассеивание
    Gl.glEnable(Gl.GL_LIGHT0); // Включаем в уравнение освещенности
источник GL_LIGHT0
    // Диффузионная компонента цвета материала
    Gl.glMaterialfv(Gl.GL_FRONT, Gl.GL_DIFFUSE, mtClr);
    // Выводим тонированный glut-примитив
    showSolid(obj);
    Gl.glDisable(Gl.GL_LIGHTING); // Будем рассчитывать
освещенность
}
Pr.Invalidate();
}

private void button1_Click(object sender, EventArgs e)
{
    draw(1); // Конус
}

private void button2_Click(object sender, EventArgs e)
{
    draw(2); // Куб
}

```

```

private void button3_Click(object sender, EventArgs e)
{
    draw(3); // Цилиндр
}

private void button4_Click(object sender, EventArgs e)
{
    draw(4); // Додекаэдр
}

private void button5_Click(object sender, EventArgs e)
{
    draw(5); // Икосаэдр
}

private void button6_Click(object sender, EventArgs e)
{
    draw(6); // Октаэдр
}

private void button7_Click(object sender, EventArgs e)
{
    draw(7); // Ромбический додекаэдр
}

private void button8_Click(object sender, EventArgs e)
{
    draw(8); // Фрактал Губка Серпиского
}

private void button9_Click(object sender, EventArgs e)
{
    draw(9); // Сфера
}

private void button10_Click(object sender, EventArgs e)
{
    draw(10); // Чайник
}

private void button11_Click(object sender, EventArgs e)
{
    draw(11); // Тетраэдр
}

private void button12_Click(object sender, EventArgs e)
{
    draw(12); // Top
}

private void Form1_Load(object sender, EventArgs e)
{
    // Черный цвет фона
    Gl.glClearColor(0, 0, 0, 1);
    // Инициализация Glut
    Glut.glutInit();
    // Используем в Glut систему цветов RGB, двойную буферизацию
    (экранный и внеэкранный буферы) и буфер глубины

```

```

        Glut.glutInitDisplayMode(Glut.GLUT_RGB | Glut.GLUT_DOUBLE |
Glut.GLUT_DEPTH);
        // Активируем тест глубины
        Gl.glEnable(Gl.GL_DEPTH_TEST);
    }
}
}

```

### Задания к лабораторной работе № 7

1. Реализуйте пример
2. Модифицируйте пример так, чтобы происходило непрерывный поворот выбранного объекта.
3. Реализуйте возможность изменения цвета объекта и перемещения точки освещения.

### Контрольные вопросы к лабораторной работе № 7

1. Сущность и назначение *OpenGL*.
2. Назначение *Tao Framework*.
3. Порядок установки и подключения библиотек *TAO*.
4. Поддержка *OpenGL* в *VisualC#*.
5. Инициализация *OpenGL* в *C#*.
6. Опишите основные методы преобразования объектов, реализованные в *OpenGL*.

## **Методические рекомендации по выполнению индивидуальной самостоятельной работы**

Для более глубокого и самостоятельного изучения студентам предлагаются следующие темы:

- 1) История компьютерной графики, основные даты и события (например, 50-е годы: от текстовых изображений к графической консоли; 60-е годы: от "Альбома" к анимации; 70-е годы: эпоха алгоритмов; 80-е годы: компьютерная графика в кино; 90-е годы: время стандартов, Интернета и компьютерных игр; 21 век, перспективы компьютерной графики; или предложить свои этапы развития компьютерной графики).
- 2) Выдающиеся личности в компьютерной графике (П. Безье, А. Сазерленд, Стив Рассел, Джон Уорнок, Джим Кларк, Генри Гуро, Мартин Ньюелл, Ву Тонг Фонг, Бенуа Мандельброт, Джеймс Блинн, Эд Катмалл, Лорен Карпентер, Алвай Рей Смит, и др.).
- 3) Области применения компьютерной графики (научная графика, деловая графика, конструкторская графика, иллюстративная графика, художественная и рекламная графика, компьютерная анимация, мультимедиа).
- 4) Стандарты и языки компьютерной графики (CGI, IGES, Direct3D, DirectX, VRML, OpenGL, ActionScrip)
- 5) Классификация компьютерной графики (двухмерная, трехмерная, растровая, векторная, фрактальная, воксельная и т.д.)
- 6) Форматы хранения графической информации.
- 7) Представление цветов в компьютере.
- 8) Технические средства компьютерной графики: мониторы.
- 9) Технические средства компьютерной графики: графические адаптеры.
- 10) Технические средства компьютерной графики: плоттеры.
- 11) Технические средства компьютерной графики: принтеры.
- 12) Технические средства компьютерной графики: сканеры.

- 13) Технические средства компьютерной графики: световое перо, джойстик, трекбол, тачпад, трекпойнт, дигитайзеры.
- 14) Технические средства компьютерной графики: графические планшеты.
- 15) Графические процессоры.
- 16) Программные средства компьютерной графики: CorelDraw.
- 17) Программные средства компьютерной графики: PhotoShop.
- 18) Программные средства компьютерной графики: 3DMax.
- 19) Виртуальная и дополненная реальность.
- 20) Графика и игры. Технологии, применяющиеся в «игрострое». История и современность.
- 21) Графические технологии будущего.
- 22) Цифровое искусство.
- 23) ГИС как разновидность интерактивной компьютерной графики

В результате самостоятельного изучения каждый студент должен написать реферат на одну из предложенных тем. Студент может предложить собственную тему реферата, соответствующую дисциплине «Компьютерная графика».

*Требования к структуре, оформлению и защите реферата*

Структура реферата

- титульный лист (образец – приложение В);
- введение
- основная часть
- заключение
- список использованных источников
- приложения.

Введение должно содержать краткое описание рассматриваемой темы, цель работы и рассматриваемые вопросы. Объем введения – 1-2 стр. Основная часть может быть разбита на несколько пунктов. Заключение содержит основные выводы по проделанной работе (объем – 1 стр.). Список

использованных источников содержит перечень учебников, журналов, электронных источников и источников Интернет, используемых при написании реферата. Необходимо использовать информацию не старше 5 лет. На каждый источник по тексту реферата обязательно должны быть ссылки.

Стиль написания реферата - строгий технический. Поэтому по тексту нельзя использовать рекламные слоганы, слэнги и личные местоимения. Повествование должно вестись от третьего неопределенного лица (например, «рассмотрены...», «предложены...», «предложена классификация...» и т.п.).

### Оформление реферата

Оформление реферата должно быть выполнено в соответствии с «Образовательный стандарт вуза ОС ТУСУР 01-2013. Работы студенческие по направлениям подготовки и специальностям технического профиля. Общие требования и правила оформления» (<https://regulations.tusur.ru/documents/70>).

Наиболее важные требования:

- формат листа – А4;
- поля: верхнее, нижнее – 2 см; правое – 1 см; левое – 3 см;
- межстрочный интервал – полуторный;
- красная строка – 1,25 см;
- шрифт – Times New Roman 12-14 пт;
- все рисунки и таблицы должны иметь номер и название;
- в конце заголовков точка не ставиться;
- нумерация страниц – по центру вверху страницы (первая страница – титульный лист, на нем номер не проставляется).

Объем реферата – 20-25 стр.

*Реферат не может быть засчитан при наличии хотя бы одного из ниже перечисленных недостатков:*

- если полностью или в значительной части работа выполнена несамостоятельно, т.е. путем механического переписывания учебников, специальной или другой литературы, копирования источников Интернет;
- если выявлены существенные ошибки, свидетельствующие о том, что содержание тем не раскрыто;
- если работа отличается узконаправленным замкнутым подходом к решаемым проблемам без применения комплексного анализа, позволяющего студенту проявить широкий объем знаний;
- если работа написана небрежно, неразборчиво, с несоблюдением правил оформления.

### Защита реферата

До 10 декабря необходимо сдать реферат на проверку преподавателю.

В период с 15 по 29 декабря проводится защита реферата в форме лекции-презентации (точное время проведения назначает преподаватель заранее - чаще всего защита проводится за неделю до окончания семестра). На данном занятии студент выступает с докладом продолжительностью 5-7 минут. Доклад должен содержать краткое описание рассмотренной в реферате теме. Рекомендуется в процессе доклада использовать демонстрационный материал в виде мультимедийной презентации (возможно включение роликов). После доклада предоставляется возможность присутствующим задать докладчику вопросы. Результатом защиты являются рейтинговые баллы.

### Список рекомендуемой литературы и других источников

1. Роджерс, Д. Алгоритмические основы машинной графики / Д. Роджерс // Пер. с англ.- М.: Мир, 1989. - 512с.
2. Роджерс, Д. Математические основы машинной графики / Д. Роджерс, Дж. Адамс // Пер. с англ. - М.: Машиностроение, 1996. - 240с.
3. Блинова, Т. А. Компьютерная графика / Т. А. Блинова, В. Н. Порев. - Киев: Издательство "Юниор", 2005.
4. Порев, В. Н. Компьютерная графика / В. Н. Порев. - Киев: Издательство "Юниор", 2000.
5. Поляков, А. Ю. Машинная графика и геометрическое моделирование / А. Ю. Поляков. - Томск : ТМЦДО, 2001. - 142 с.
6. Шелестов, А. А. Компьютерная графика / А. А. Шелестов. - Томск: ТМЦДО, 2002.
7. Шикин, Е. В. Компьютерная графика. Динамика, реалистические изображения / Е. В. Шикин, А. В. Боресков. - М.: «ДИАЛОГ–МИФИ», 1995. – 288с.
8. Фоли, Дж. Основы интерактивной машинной графики. Кн.1 и 2. / Дж. Фоли, Вэн Дем. – М.: Мир, 1990.
9. Павлидис, Т. Алгоритмы машинной графики и обработки изображений / Т. Павлидис. – М. : Радио и связь, 1991. – 400 с.
10. Гардан, И. Машинная графика и автоматизация конструирования / И. Гардан, М. Люка. – М.:Мир, 1994.
11. Тихомиров, Ю. Программирование трехмерной графики / Ю. Тихомиров. – СПб.: ВHV – Санкт-Петербург, 1998. - 250с.
12. Корриган Дж. Компьютерная графика: Секреты и решения. Пер. с англ. – М.:Энтроп, 1995.



## Приложение А

### Образец титульного листа и отчета по лабораторной работе

Министерство науки и высшего образования РФ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Томский государственный университет систем управления и радиоэлектроники  
(ТУСУР)»

Кафедра компьютерных систем в управлении и проектировании (КСУП)

### ПРЕОБРАЗОВАНИЯ В ПРОСТРАНСТВЕ. ПРОЕКЦИИ. УДАЛЕНИЕ НЕВИДИМЫХ ЛИНИЙ

Отчет по лабораторной работе № 6  
по дисциплине «Компьютерная графика»

Вариант –

Студенты гр.581-1

\_\_\_\_\_ И. И. Иванов

\_\_\_\_\_ И. И. Петров

«\_\_» \_\_\_\_\_ 20\_\_ г.

Проверил

канд. техн. наук,

доцент каф.КСУП

\_\_\_\_\_ Н. Ю. Хабибулина

«\_\_» \_\_\_\_\_ 20\_\_ г.

20\_\_ г.

### **Цель работы и постановка задачи**

*Цель работы* – изучение и реализация алгоритмов построения проекции фигуры, удаления невидимых линий и граней.

### **Задания к лабораторной работе № 6**

1. Откройте проект, созданный в предыдущих лабораторных работах. Для кнопки «Лабораторная работа №6» создайте соответствующий обработчик (модуль), удовлетворяющий нижеследующим требованиям.

2. Напишите программный модуль для отображения преобразований многогранника в пространстве.

Первым действием отобразите многогранник на экране – высота многогранника параллельна вертикальной оси.

Далее реализуйте:

- построение оси вращения (при этом координаты точки, через которую проходит ось вращения, задаются пользователем интерактивно);
- поворот многогранника так, чтобы его высота совпала с заданной осью вращения;
- вращение относительно высоты многогранника, совпадающей с заданной осью вращения;
- перемещение многогранника (по каждой координатной оси, по двум или трем осям одновременно);
- изменение размера многогранника (по каждой оси, по двум или трем координатным осям одновременно);
- настройку цвета и толщины линий;
- изменение направления и скорости вращения / перемещения.

3. Нарисуйте систему координат и ось вращения.

4. Координаты точек – мировые.

Максимальное и минимальное значение мировых координат изобразите на экране. Осуществите автоматический подбор размеров изображения.

5. При выполнении этого задания при удалении невидимых линий программа должна выдавать контурное изображение (невидимые линии рисовать штрих-пунктиром).

6. Отображаемую фигуру, ось вращения, вид проекции и алгоритм удаления невидимых линий выберите в соответствии с вариантом (таблица 3):

№	Фигура	Проекция	Ось вращения	Алгоритм удаления невидимых линий
	Тетраэдр	Изометрия	Прямая, заданная произвольными направляющими косинусами	алгоритм, использующий нормаль грани

### Анализ задачи

Для выполнения поставленной задачи необходимо реализовать следующие алгоритмы:

1. Переход от мировых координат к экранным координатам.
2. Трёхмерные сдвиг и масштабирование.
3. Трёхмерное вращение вокруг произвольной оси.
4. Изометрическое проецирование трёхмерного изображения на плоскость
5. Удаление невидимых линий (прорисовка пунктиром).

1. Алгоритм перехода от мировых координат к экранным координатам.

Для получения 3D-преобразований воспользуемся векторно-полигональной моделью фигуры. В соответствии с ней любая фигура в трехмерном пространстве может быть представлена матрицей координат своих вершин. Затем, используя данную матрицу, проводят вектора, которые соединяют соответствующие вершины. В результате получают каркасное изображение фигуры. Далее, используя алгоритмы удаления невидимых линий, получают изображение фигуры с отсечением невидимых ребер.

Изначально любая трехмерная фигура задается мировыми однородными координатами. Для ее отображения на экране необходимо получить экранные координаты. Для получения экранных координат используют следующую цепочку преобразований:

*мировые координаты  $(x, y, z)$   $\rightarrow$  видовые координаты  $(x_v, y_v, z_v)$   $\rightarrow$  координаты проекций  $(x_{np}, y_{np}, z_{np})$   $\rightarrow$  экранные координаты  $(x_э, y_э, z_э)$ .*

2. Преобразования сдвига и масштабирования в однородных координатах относительно центра координат имеют одинаковую форму произведения вектора исходных координат вершины фигуры на матрицу преобразования.

*Преобразование сдвига.*

Преобразование сдвига выглядит следующим образом:

$$v \cdot T = (x \ y \ z \ h) \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{pmatrix},$$

где  $v$  – вектор трехмерных однородных координат вершины фигуры,  $h$  – произвольный множитель не равный нулю (в нашем случае 1),  $T$  – матрица сдвига,  $l, m, n$  – величины смещения по осям  $OX, OY, OZ$  соответственно.

*Преобразование масштабирования.*

Преобразование сдвига выглядит следующим образом:

$$v \cdot T = (x \ y \ z \ h) \cdot \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & j & 0 \\ 0 & 0 & 0 & s \end{pmatrix},$$

где  $v$  – вектор трехмерных однородных координат вершины фигуры,  $T$  – матрица масштабирования:  $a, e, j$  – величины масштабирования по осям  $OX, OY, OZ$  соответственно;  $s$  – полное изменение масштаба (в данном случае  $a, e, j$  должны быть равны 1).

### 3. Трёхмерное вращение вокруг произвольной оси.

Преобразование поворота вокруг произвольной оси, заданной направляющими косинусами и проходящей через заданную точку, выглядит следующим образом:

$$v \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -l & -m & -n & 1 \end{pmatrix} \cdot R \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{pmatrix},$$

где  $v$  – вектор трехмерных однородных координат вершины многогранника,  $l, m, n$  – заданные координаты точки, через которую проходит ось вращения,  $R$  – матрица поворота вокруг оси, заданной направляющими косинусами:

$$R := \begin{bmatrix} (n_1)^2 + (1 - n_1)^2 \cos(\theta) & n_1 \cdot n_2 (1 - \cos(\theta)) + n_3 \sin(\theta) & n_1 \cdot n_3 (1 - \cos(\theta)) & 0 \\ n_1 \cdot n_2 (1 - \cos(\theta)) - n_3 \sin(\theta) & (n_2)^2 + (1 - n_2)^2 \cos(\theta) & n_2 \cdot n_3 (1 - \cos(\theta)) + n_1 \sin(\theta) & 0 \\ n_1 \cdot n_3 (1 - \cos(\theta)) + n_2 \sin(\theta) & n_2 \cdot n_3 (1 - \cos(\theta)) - n_1 \sin(\theta) & (n_3)^2 + (1 - n_3)^2 \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

где  $n_1, n_2, n_3$  – направляющие косинусы относительно осей  $OX, OY$  и  $OZ$  соответственно,  $\Theta$  – угол поворота.

Положительным направлением считается направление против часовой стрелки, если смотреть вдоль оси вращения к началу координат.

#### 4. Проецирование изображения.

В данной лабораторной работе применяется изометрическая проекция. Матрица проецирования имеет вид:

$$pro := \begin{pmatrix} -\cos\left(\frac{\pi}{6}\right) & \sin\left(\frac{\pi}{6}\right) & 0 & 0 \\ \cos\left(\frac{\pi}{6}\right) & \sin\left(\frac{\pi}{6}\right) & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Проецирование осуществляется умножением вектора трехмерных однородных координат на матрицу проецирования. Оси системы координат расположены под углом 120 градусов относительно друг друга.

5. Для построения более реалистичного изображения трёхмерных сцен необходимо удалять невидимые части объектов (рёбра и грани).

Существует 2 основных подхода к решению данной задачи.

Первый подход заключается в непосредственном сравнении объектов друг с другом для выяснения того, какие части объектов являются видимыми. В данном случае работа ведётся в пространстве объектов.

Второй подход заключается в определении для каждого пикселя экрана ближайшего к нему объекта (вдоль направления проецирования). При этом работа ведётся в пространстве экранных координат.

Рассмотрим один из алгоритмов, реализующий первый подход.

Отсечение нелицевых граней.

Пусть для каждой грани некоторой фигуры задан единичный вектор внешней нормали. Если вектор нормали грани составляет с направлением проецирования (направлением взгляда на объект) тупой угол, то такая грань не может быть видна и называется нелицевой. В случае, когда данный угол является острым, грань видна и называется лицевой.

В случае, когда трёхмерная сцена представляет собой один выпуклый многогранник, удаление нелицевых граней полностью решает задачу удаления невидимых граней.

В общем случае описанная проверка не решает задачу полностью, но позволяет значительно сократить количество рассматриваемых граней.

Алгоритм удаления нелицевых плоскостей, используемый в данной работе.

1. Сформировать многоугольники граней, исходя из списка вершин тела.
2. Вычислить нормальный вектор для каждой полигональной грани тела.
3. Определить нелицевые плоскости:

Определить косинус угла между нормалью к грани и вектором направления наблюдателя.

Если этот косинус угла меньше 0, то плоскость невидима.

Вывести весь многоугольник, лежащий в этой плоскости пунктиром.

4. Вывести остальные грани.

### Описание структуры и алгоритма программы

Разработанная программа имеет структуру, изображенную на рисунке 1.

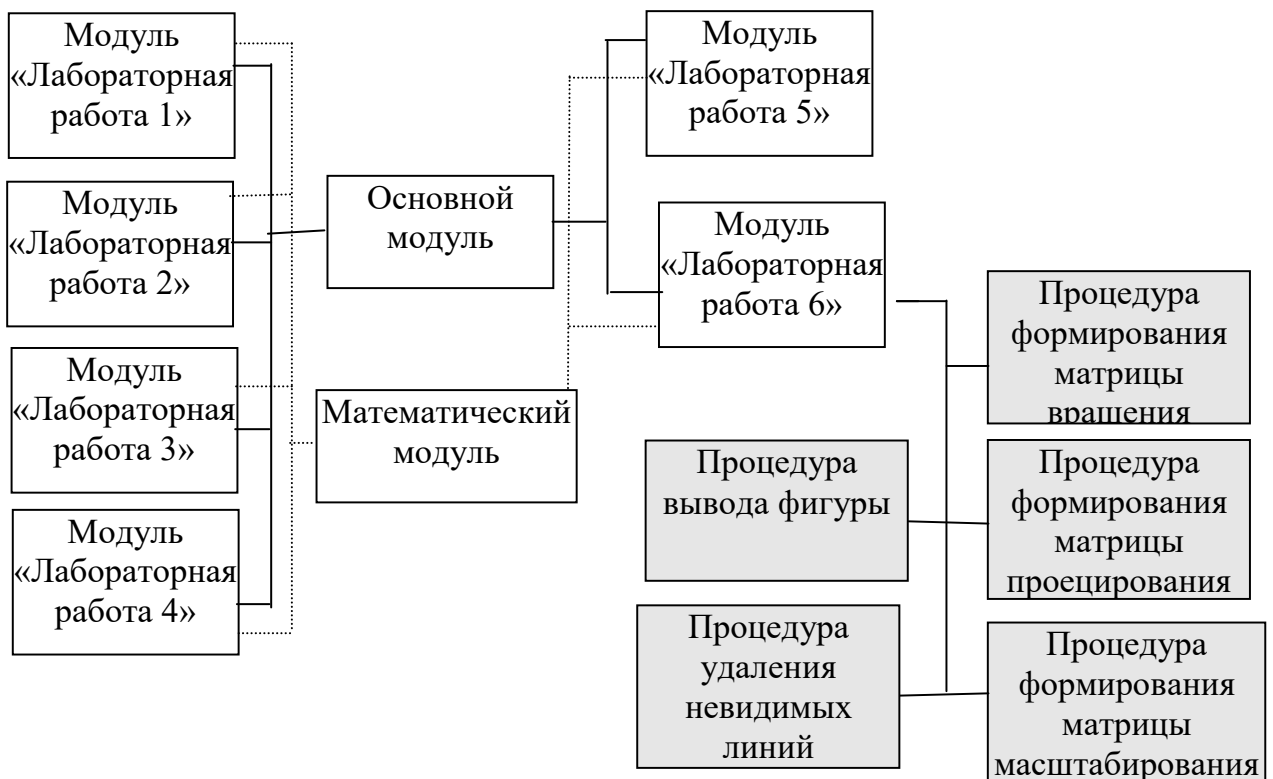


Рисунок 1

Разработанный в данной лабораторной работе модуль имеет имя lab6.pas. Его форма lab6.dfm.

Основной алгоритм программного модуля:

1. Формируем матрицу мировых координат фигуры
2. Формируем матрицы преобразования.

### 3. Организуем цикл поворота фигуры:

- 3.1 очистка экрана;
- 3.2 умножаем матрицу фигуры на матрицу смещения;
- 3.3 умножаем матрицу фигуры на матрицу поворота;
- 3.4 умножаем матрицу фигуры на матрицу обратного смещения;
- 3.5 умножаем матрицу фигуры на матрицу проецирования;
- 3.6 определяем видимость ребер и выводим фигуру красным цветом с новыми координатами;
- 3.7 увеличиваем угол поворота;
- 3.8 переходим на шаг 3.

### Описание основных процедур и функций

Модуль состоит из нескольких процедур:

...

(описание процедур – название, входные, выходные параметры, ее назначение и подробное пошаговое описание основных процедур)

procedure FormCreate(Sender: TObject); - процедура для создания формы.

procedure Draw(A:TMatrix; f:boolean); – в зависимости от значения f выводит грань пунктирной или сплошной линией.

procedure ImageClear; – очистка экрана.

function VectorMul(A:TMatrix):Tmatrix; – функция возвращает нормаль к грани.

function ScalarMul(Vector:TMatrix):real; – функция определяет косинус угла между нормалью и направлением проецирования.

procedure DrawXY(A:TMatrix); – выводит оси на экран.

#### *Пошаговое описание основной процедуры Draw*

Шаг 1. Выделяем память под массивы граней фигуры, нормалей к этим граням, матрицу переноса, матрицу поворота фигуры вокруг оси, матрицу изометрической проекции;

Шаг 2. Задаем начальные значения для всех этих матриц;

Шаг 3. Начальный угол поворота равен 0;

Шаг 4. Цикл пока не нажата кнопка «Стоп»;

Шаг 5. Повернуть 1-ю грань фигуры на угол Phi

Шаг 6. Определить нормаль к этой грани

Шаг 7. Определить угол между нормалью и вектором проецирования

Шаг 8. Если угол тупой, грань не лицевая f1=false, иначе лицевая f1= true

Шаг 9. Повернуть 2-ю грань фигуры на угол  $\Phi$

Шаг 10. Определить нормаль к этой грани

Шаг 11. Определить угол между нормалью и вектором проецирования

Шаг 12. Если угол тупой, грань не лицевая  $f2=false$ , иначе лицевая,  $f2=true$

Шаг 13. Повернуть 3-ю грань фигуры на угол  $\Phi$

Шаг 14. Определить нормаль к этой грани

Шаг 15. Определить угол между нормалью и вектором проецирования

Шаг 16. Если угол тупой, грань не лицевая  $f3=false$ , иначе лицевая  $f3=true$

Шаг 17. Повернуть 4-ю грань фигуры на угол  $\Phi$

Шаг 18. Определить нормаль к этой грани

Шаг 19. Определить угол между нормалью и вектором проецирования

Шаг 20. Если угол тупой, грань не лицевая  $f4=false$ , иначе лицевая  $f4=true$

Шаг 21. Если 1-ая грань не лицевая – нарисовать пунктиром, иначе ничего не выводить

Шаг 22. Если 2-ая грань не лицевая – нарисовать пунктиром, иначе ничего не выводить

Шаг 23. Если 3-ая грань не лицевая – нарисовать пунктиром, иначе ничего не выводить

Шаг 24. Если 4-ая грань не лицевая – нарисовать пунктиром, иначе ничего не выводить

Шаг 25. Если 1-ая грань лицевая – нарисовать сплошной линией, иначе ничего не выводить

Шаг 26. Если 2-ая грань лицевая – нарисовать сплошной линией, иначе ничего не выводить

Шаг 27. Если 3-ая грань лицевая – нарисовать сплошной линией, иначе ничего не выводить

Шаг 28. Если 4-ая грань лицевая – нарисовать сплошной линией, иначе ничего не выводить

Шаг 29. Увеличить угол вращения

Шаг 30. Обновить матрицу вращения

Шаг 31. Кнопка «Стоп» нажата? Да выход, Иначе переход на Шаг 5.

### **Руководство пользователя**

Исполняемый файл программы – labb.exe. После запуска программы появляется основная форма (рисунок 2).

...

Для запуска лабораторной работы 6 нажмите кнопку «Лабораторная работа7».

Появится форма (рисунок 3).

...



Для поворота фигуры нажмите кнопку «Поворот». Для изменения скорости вращения передвиньте ползунок на полосе прокрутки (рисунок 4).

...

Для выхода из программы закройте форму.

### **Ответы на контрольные вопросы**

.....

### **Тестовые вопросы**

.....

### **Заключение**

В ходе проделанной работы изучили алгоритмы получения 3D-преобразований фигуры, а именно: алгоритм формирования векторно-полигональной модели фигуры, алгоритм перехода от мировых координат к экранным, алгоритмы вращения, смещения, масштабирования в пространстве, алгоритм проецирования, алгоритм удаления невидимых линий.

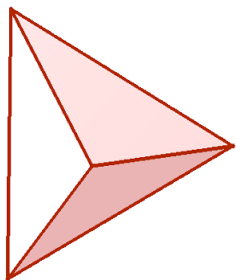
В результате создан программный продукт, реализующий все изученные алгоритмы и предоставляющий следующие возможности:

- отображение трехмерной фигуры на экране;
- задание оси вращения;
- вращение фигуры вокруг заданной оси;
- перемещение и масштабирование фигуры по каждой координатной оси, по двум или трем осям одновременно;
- удаление невидимых ребер фигуры;
- настройку скорости и направления вращения / перемещения;
- настройку цвета и стиля линий.

## Приложение Б

### Основные геометрические фигуры

#### ***Тетраэдр***

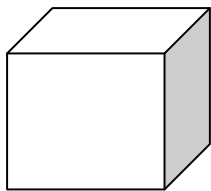


Тетраэдр составлен из четырех равносторонних треугольников. Каждая его вершина является вершиной трех треугольников. Сумма плоских углов при каждой вершине равна 180 градусов. Таким образом, тетраэдр имеет 4 грани, 4 вершины и 6 ребер.

*Элементы симметрии:*

Тетраэдр не имеет центра симметрии, но имеет 3 оси симметрии и 6 плоскостей симметрии.

#### ***Гексаэдр (Куб)***

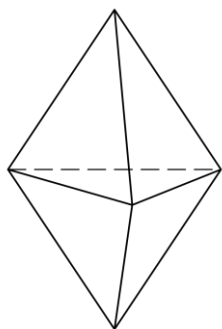


Куб составлен из шести квадратов. Каждая его вершина является вершиной трех квадратов. Сумма плоских углов при каждой вершине равна 270 градусов. Таким образом, куб имеет 6 граней, 8 вершин и 12 ребер.

*Элементы симметрии:*

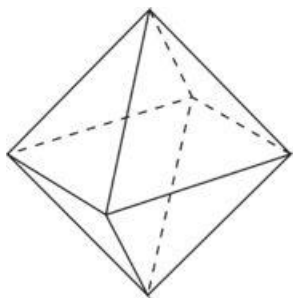
Куб имеет центр симметрии - центр куба, 9 осей симметрии и 9 плоскостей симметрии.

#### ***Тригональная бипирамида (шестигранник, образованный равносторонними треугольниками)***



Составлен из шести равносторонних треугольников. Две противоположные осевые вершины являются вершинами трех треугольников.

*Элементы симметрии:* имеет 4 оси симметрии.

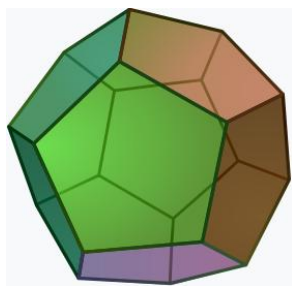


### ***Октаэдр***

Октаэдр составлен из восьми равносторонних треугольников. Каждая его вершина является вершиной четырех треугольников. Сумма плоских углов при каждой вершине равна 240 градусов. Таким образом, октаэдр имеет 8 граней, 6 вершин и 12 ребер.

*Элементы симметрии:*

Октаэдр имеет центр симметрии - центр октаэдра, 9 осей симметрии и 9 плоскостей симметрии.

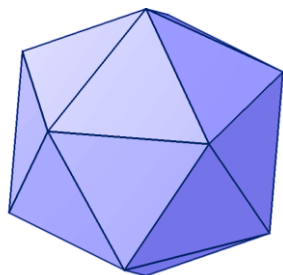


### ***Додекаэдр***

Додекаэдр составлен из двенадцати равносторонних пятиугольников. Каждая его вершина является вершиной трех пятиугольников. Сумма плоских углов при каждой вершине равна 324 градусов. Таким образом, додекаэдр имеет 12 граней, 20 вершин и 30 ребер.

*Элементы симметрии:*

Додекаэдр имеет центр симметрии - центр додекаэдра, 15 осей симметрии и 15 плоскостей симметрии.

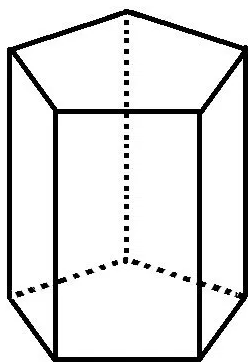


### ***Икосаэдр***

Икосаэдр составлен из двадцати равносторонних треугольников. Каждая его вершина является вершиной пяти треугольников. Сумма плоских углов при каждой вершине равна 300 градусов. Таким образом, икосаэдр имеет 20 граней, 12 вершин и 30 ребер.

*Элементы симметрии:*

Икосаэдр имеет центр симметрии - центр икосаэдра, 15 осей симметрии и 15 плоскостей симметрии.



### ***Пятиугольная призма (карандаш)***

Пятиугольная призма составлена из двух равносторонних пятиугольников в основании, соединенных прямыми линиями.

*Элементы симметрии:*

Пятиугольная призма имеет две оси симметрии.



### ***Призматойд***

Призматойд — многогранник, две грани которого (основания призматойда) лежат в параллельных плоскостях, а остальные являются треугольниками (или трапециями), причём у треугольников одна сторона, а у трапеций оба основания являются сторонами оснований призматойда.

## Приложение В

### Пример оформления титульного листа реферата

Министерство науки и высшего образования РФ  
 Федеральное государственное бюджетное образовательное учреждение  
 высшего образования  
 «ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
 СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)»  
 Кафедра компьютерных систем в управлении и проектировании (КСУП)

### ГРАФИЧЕСКИЕ ПРОЦЕССОРЫ

Тематический реферат  
 по дисциплине «Компьютерная графика»

Студент гр. 581-1

\_\_\_\_\_ И. И. Иванов  
 «\_\_» \_\_\_\_\_ 20\_\_ г.

Руководитель:

канд.техн. наук,  
 доцент каф. КСУП,

\_\_\_\_\_ Н. Ю. Хабибулина  
 «\_\_» \_\_\_\_\_ 20\_\_ г.

\_\_\_\_\_  
 (оценка)

20\_\_ г.

## Приложение Г

### Некоторые сведения по визуализации объекта с помощью библиотеки OpenGL

#### Инициализация *OpenGL*

После того как объект *SimpleOpenGLControl* прошел инициализацию, стартует загрузка формы. Необходимо всегда отслеживать это событие, так как именно здесь выполняется начальная настройка программы.

Для визуализации трехмерной сцены используем следующий код.

```
private void Form1_Load(object sender, EventArgs e)
{
    // инициализация Glut
    Glut.glutInit();
    Glut.glutInitDisplayMode(Glut.GLUT_RGB | Glut.GLUT_DOUBLE |
    Glut.GLUT_DEPTH);
    // очистка окна
    Gl.glClearColor(255, 255, 255, 1);
    // установка порта вывода в соответствии с размерами элемента Pr
    Gl.glViewport(0, 0, Pr.Width, Pr.Height);
    // настройка проекции
    Gl.glMatrixMode(Gl.GL_PROJECTION);
    Gl.glLoadIdentity();
    Glu.gluPerspective(45, (float)Pr.Width / (float)Pr.Height, 0.1, 200);
    Gl.glMatrixMode(Gl.GL_MODELVIEW);
    Gl.glLoadIdentity();
    // настройка параметров OpenGL для визуализации
    Gl.glEnable(Gl.GL_DEPTH_TEST);
    Gl.glEnable(Gl.GL_LIGHTING);
    Gl.glEnable(Gl.GL_LIGHT0);
}
```

Здесь в первую очередь выполняется инициализация библиотеки *Glut*.

Как видно из кода, для работы с функциями библиотеки *OpenGL* используется класс *Gl*, находящийся в пространстве имен *Tao.OpenGL*.

Для работы с функциями библиотеки *Glut* используется класс *Glut*.

Обязательно необходимо вызвать функцию *glutInit()* перед тем, как начать использовать любые другие функции данной библиотеки, так как эта функция производит инициализацию библиотеки *Glut*. Далее вызываем функцию

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

Эта функция устанавливает режим отображения. В данном случае устанавливается режим *RGB* для визуализации (*GLUT\_RGB* – это псевдоним *GLUT\_RGBA*, он устанавливает режим *RGBA* битовой маски окна). Далее устанавливаем двойную буферизацию окна, которая, как правило, используется для устранения мерцания, возникающего в процессе

быстрой перерисовки кадров несколько раз подряд. *GLUT\_DEPTH* указывает при инициализации окна, будет ли в приложении использоваться буфер глубины.

После инициализации окна устанавливаем цвет очистки окна с помощью функции *Gl.glClearColor(255, 255, 255, 1);*

Перед дальнейшим изучением кода перечислим этапы создания сцены в *OpenGL*:

1. Позиционирование объема видимости в пространстве (установка координат камеры).
2. Установка в данном пространстве модели (объекта), которая будет попадать в объем видимости нашей камеры.
3. Проецирование, которое определяет форму объема видимости.
4. В рамках порта просмотра получаем изображение объекта.

Теперь необходимо определить значение порта вывода, устанавливая значения в функции

*Gl.glViewport(0, 0, Pr.Width, Pr.Height);*

Здесь указываем библиотеке *OpenGL* на то, что вывод будет осуществляться во всей области элемента *Pr* (элемент, расположенный на форме для визуализации в него сцены). Определяем тот самый порт просмотра, в который последним шагом будет визуализироваться модель.

После этого происходит настройка проекции. Для этого сначала вызываем функцию *Gl.glMatrixMode(Gl.GL\_PROJECTION);*

Функция *glMatrixMode* предназначена для того, чтобы задавать матричный режим: будет определена матрица, над которой в дальнейшем будут производиться операции. В нашем случае это *GL\_PROJECTION* – матрица проекций.

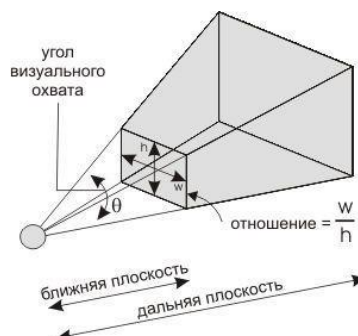
Следующей командой очищаем матрицу с помощью функции *glLoadIdentity* (функция заменяет текущую матрицу на единичную). Далее устанавливаем тип текущей проекции с помощью функции *gluPerspective*.

*Gl.glLoadIdentity();*

*Glu.gluPerspective(45, (float)Pr.Width / (float)Pr.Height, 0.1, 200);*

Функция *gluPerspective* определена в библиотеке *GLU – OpenGL Utility Library (GLU)*. Эта библиотека является надстройкой над библиотекой *OpenGL*, реализующей ряд более продвинутых функций. Она также является свободно распространяемой и поставляется вместе с библиотекой *OpenGL*.

Данная функция строит пирамиду охвата видимости, основываясь на угле визуального охвата, отношении сторон порта просмотра и установке ближней и дальней плоскости просмотра (рис. Г.1).



Теперь, когда проекция определена, устанавливаем в качестве текущей матрицы объектно-видовую матрицу и очищаем ее.

```
Gl.glMatrixMode(Gl.GL_MODELVIEW);
```

```
Gl.glLoadIdentity();
```

Теперь остается только включить некоторые опции, необходимые для корректной визуализации сцены. Это тест глубины, а также отображение цветов материалов.

```
Gl.glEnable(Gl.GL_DEPTH_TEST);
Gl.glEnable(Gl.GL_COLOR_MATERIAL);
```

### Визуализация объектов

Теперь рассмотрим функцию, визуализирующую трехмерную сферу. Код этой функции:

```
// обработчик кнопки "визуализировать" private void
button1_Click(object sender, EventArgs e)
{
    Gl.glClear(Gl.GL_COLOR_BUFFER_BIT |
Gl.GL_DEPTH_BUFFER_BIT);
    Gl.glLoadIdentity();
    Gl.glPushMatrix();
    Gl.glTranslated(0,0,-6);
    Gl.glRotated(45, 1, 1, 0);
    // рисуем сферу с помощью библиотеки FreeGLUT
    Glut.glutWireSphere(2, 32, 32);
    Gl.glPopMatrix();
    Gl.glFlush();
    Pr.Invalidate();
}
```

Когда пользователь нажимает на кнопку и вызывается данная функция, первым делом производится очистка окна (так как до этого уже мог быть реализован какой-либо вывод; очистка экрана перед визуализацией кадра – это стандартный шаг).

Для этого используется функция *glClear*. В качестве параметра функция получает наименования буферов, которые необходимо очистить; в нашем случае это буфер цвета и буфер глубины.

Далее очищаем объектно-видовую матрицу – таким образом камера устанавливается в начало координат. Теперь можем совершить ее перемещение в пространстве.

Но перед этим вызываем функцию

```
Gl.glColor3f(255, 0, 0);
```

устанавливающую красный цвет для отрисовки (основывается на *RGB* составляющих).



Займемся перемещением.

Функция *glPushMatrix* помещает текущую матрицу в стек матриц, откуда в дальнейшем можно ее вернуть с помощью функции *glPopMatrix*.

Таким способом осуществим перемещение отрисовываемого объекта в пространстве, не изменив саму матрицу, отвечающую за положение камеры (наблюдателя).

Если не использовать такой подход, то с каждым визуализированным кадром камера будет перемещаться и очень скоро вообще не сможем ее найти.

Сохранив матрицу в стеке, производим перемещение объекта на 6 единиц по оси Z, после чего выполняем поворот сцены на 45 градусов сразу по двум осям

```
Gl.glTranslated(0,0,-6);
```

```
Gl.glRotated(45, 1, 1, 0);
```

После этого выполняем рисование объекта в той области, куда переместились (сфера радиусом 2 в виде сетки). Сфера будет разбита на 32 меридиана и 32 параллели.

Для визуализации используется библиотека *FreeGlut*.

```
//рисует сферу с помощью библиотеки FreeGLUT
```

```
Glut.glutWireSphere(2, 32, 32);
```

Возвращаем сохраненную в стеке матрицу

```
Gl.glPopMatrix();
```

Дожидаемся, пока библиотека *OpenGL* завершит визуализацию этого кадра

```
Gl.glFlush();
```

и посылаем нашему элементу *Pr*, в котором происходит визуализация сцены, сигнал о том, что необходимо обновить отображаемый кадр, т.е. вызываем его перерисовку. *Pr.Invalidate()*;

На этом визуализация объекта заканчивается.