ГОСУДАРСТВЕННЫЙ КОМИТЕТ СВЯЗИ, ИНФОРМАТИЗАЦИИ
И ТЕЛЕКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ РЕСПУБЛИКИ
УЗБЕКИСТАН
ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ
ТЕХНОЛОГИЙ

Кафедра ПОИТ

Курсовая работа по дисциплине

«Объектно-ориентированные языки программирования»

на тему:

«Классы»

Выполнил: Умрияев Шохрух 223-10 ИТр
Проверила: Абдурахманова Н.Н.

Ташкент – 2013

«Утверждаю» Заведующий кафедрой

_____

«__»_____201_г.

Задание

На курсовую работу

По дисциплине: Объектно-ориентированные языки программирования

Студент: Умрияев Шохрух Алишерович, группа 223-10 ИТр

Руководитель: _____

Защита курсовой работы: _____

Тема: Классы (тема №1). Номер варианта 1.1

Создать класс для трехмерной фигуры (в соответствии с вариантом). Реализовать в классе методы вычисления поверхности и объема, а также, рисования и вращения. Применить указанные методы в программе.

Вариант 1.1: Куб

Содержание работы. Объем ___ листов А4.

Подпись преподавателя: _____

Подпись студента: _____

Дата выдачи задания:

Аннотация

Данная курсовая работа по предмету: «Объектно-ориентированные языки программирования» на тему: «Классы (тема №1). Номер варианта 1.1». В данной курсовой работе рассматриваются реализация ООП в языке программирования C#. Программа разработана с помощью Microsoft Visual Studio 2010 на языке C#.

Объем работы листов

Оглавление

1. Цель

Научиться создавать собственные классы, реализовывать их методы. Научиться работать с графической библиотекой GDI+.

2. Постановка задачи

Создать класс для представления куба.

В классе реализовать методы для рисования куба, метод для вращения куба, метод для вычисления площади поверхности куба и метод для вычисления объема куба.

3. Вариант задания

Вариант 1.1: Куб

4. Теоретическая часть.

GDI+ - это набор программных средств, которые используются в .NET.

GDI+ позволяют создателям приложений выводить данные на экран или на принтер без необходимости обеспечивать работу с определенными типами устройств отображения. Для отображения информации программисту достаточно вызывать нужные методы классов GDI+. При этом автоматически учитываются типы определенных устройств и выполняются вызовы соответствующих драйверов.

4.1 Graphics

Класс, который ИНКАПСУЛИРУЕТ поверхность рисования GDI+. Для этого класса не определен ни один конструктор. Видимо, успех в деле ручного конструирования инкапсулятора поверхностей рисования (еще вопрос, сколько разновидностей таких поверхностей) представляется проблематичным.

Конкретный объект – представитель класса Graphics предоставляется в виде ссылки методами-обработчиками событий либо создается в ходе выполнения ряда методов применительно к конкретным объектам, обладающим "поверхностями рисования" (клиентская область формы, кнопки, панели, битовая матрица):

```
Bitmap bmp;
Pen greedPen;
:::::
greedPen = new Pen(Color.FromArgb(50, 0, 0, 255), 1);
// Новая битовая карта под новый размер клиентской области формы.
bmp = new Bitmap(this.ClientSize.Width, this.ClientSize.Height);
Graphics gr = Graphics.FromImage(bmp);
gr.DrawLine(this.greedPen, 0, 0, 100, 100);
gr.Dispose();
```

Ниже представлен список членов класса.

Clip	Получает или задает объект Region, ограничивающий область рисования данного объекта Graphics
ClipBounds	Получает структуру RectangleF, которая заключает в себе вырезанную область данного объекта Graphics
CompositingMode	Получает значение, задающее порядок рисования сложных изображений в данном объекте Graphics

CompositingQuality	Получает или задает качество отображения сложных изображений, которые выводятся в данном объекте Graphics
DpiX	Получает горизонтальное разрешение данного объекта Graphics
DpiY	Получает вертикальное разрешение данного объекта Graphics
InterpolationMode	Получает или задает режим вставки, связанный с данным объектом Graphics
IsClipEmpty	Получает значение, которое указывает, является ли вырезанная область данного объекта Graphics пустой
IsVisibleClipEmpty	Получает значение, которое указывает, является ли видимая вырезанная область данного объекта Graphics пустой
PageScale	Получает или задает масштабирование между универсальными единицами и единицами страницы для данного объекта Graphics
PageUnit	Получает или задает единицу измерения для координат страницы данного объектаGraphics
PixelOffsetMode	Получает или задает значение, которое задает порядок смещения точек во время отображения данного объекта Graphics
RenderingOrigin	Получает или задает исходное заполнение данного объекта Graphics для сглаживания цветовых переходов и для штриховки
SmoothingMode	Получает или задает качество заполнения для данного объекта Graphics
TextContrast	Получает или задает значение коррекции яркости для отображения текста
TextRenderingHint	Получает или задает режим заполнения для текста, связанного с данным объектомGraphics
Transform	Получает или задает универсальное преобразование для данного объекта Graphics
VisibleClipBounds	Получает или задает рабочий прямоугольник видимой вырезанной области данного объекта Graphics

Открытые методы	
AddMetafileComment	Добавляет комментарий к текущему объекту Metafile
BeginContainer	Перегружен. Сохраняет графический контейнер, содержащий текущее состояние данного объекта Graphics, а затем открывает и использует новый графический контейнер
Clear	Очищает всю поверхность рисования и выполняет заливку поверхности указанным цветом фона
CreateObjRef	Создает объект, который содержит всю необходимую информацию для создания прокси-сервера, используемого для коммуникации с удаленными объектами
Dispose	Освобождает все ресурсы, используемые данным объектом Graphics
DrawArc	Перегружен. Рисует дугу, которая является частью эллипса, заданного парой координат, шириной и высотой
DrawBezier	Перегружен. Строит кривую Безье, определяемую четырьмя структурамиPoint
DrawBeziers	Перегружен. Формирует набор кривых Безье из массива структур Point
DrawClosedCurve	Перегружен. Строит замкнутую фундаментальную кривую, определяемую массивом структур Point
DrawCurve	Перегружен. Строит замкнутую фундаментальную кривую через точки указанного массива структур Point
DrawEllipse	Перегружен. Формирует эллипс, который определяется ограничивающим прямоугольником, заданным с помощью пары координат — ширины и высоты
DrawIcon	Перегружен. Формирует изображение, которое представлено указанным объектом Icon, расположенным по указанным координатам
DrawIconUnstretched	Формирует изображение, представленное указанным объектом Icon, не масштабируя его
DrawImage	Перегружен. Рисует заданный объект Image в заданном месте,

	используя исходный размер
DrawImageUnscaled	Перегружен. Рисует заданное изображение, используя его исходный фактический размер, в расположении, заданном парой координат
DrawLine	Перегружен. Проводит линию, соединяющую две точки, определенные парами координат
DrawLines	Перегружен. Формирует набор сегментов линии, которые соединяют массив структур Point
DrawPath	Рисует объект GraphicsPath
DrawPie	Перегружен. Рисует сектор, определенный эллипсом, который задан парой координат, шириной, высотой и двумя радиальными линиями
DrawPolygon	Перегружен. Рисует многоугольник, определяемый массивом структурPoint
DrawRectangle	Перегружен. Рисует прямоугольник, который определен парой координат, шириной и высотой
DrawRectangles	Перегружен. Рисует набор прямоугольников, определяемых структуройRectangle
DrawString	Перегружен. Создает текстовую строку в заданном месте с указанными объектами Brush и Font
EndContainer	Закрывает текущий графический контейнер и восстанавливает состояние данного объекта Graphics, которое было сохранено при вызове методаBeginContainer
EnumerateMetafile	Перегружен. Отправляет записи указанного объекта Metafile по отдельности методу обратного вызова, который отображает их в заданной точке
Equals	Перегружен. Определяет, равны ли два экземпляра Object
ExcludeClip	Перегружен. Обновляет вырезанную область данного объекта Graphics, чтобы исключить из нее часть, определенную структурой Rectangle
FillClosedCurve	Перегружен. Заполняет замкнутую фундаментальную кривую,

	определяемую массивом структур Point
FillEllipse	Перегружен. Заполняет внутреннюю часть эллипса, который определяется ограничивающим прямоугольником, заданным с помощью пары координат — ширины и высоты
FillPath	Заполняет внутреннюю часть объекта GraphicsPath
FillPie	Перегружен. Заполняет внутреннюю часть сектора, определенного эллипсом, который задан парой координат, шириной, высотой и двумя радиальными линиями
FillPolygon	Перегружен. Заполняет внутреннюю часть многоугольника, определенного массивом точек, заданных структурами Point
FillRectangle	Перегружен. Заполняет внутреннюю часть прямоугольника, который определен парой координат, шириной и высотой
FillRectangles	Перегружен. Заполняет внутреннюю часть набора прямоугольников, определяемого структурами Rectangle
FillRegion	Заполняет внутреннюю часть объекта Region
Flush	Перегружен. Вызывает принудительное выполнение всех отложенных графических операций и немедленно возвращается, не дожидаясь их окончания
FromHdc	Статический. Перегружен. Создает новый объект Graphics из указанного дескриптора для контекста устройства
FromHdcInternal	Статический. Внутренний метод. Не используется
FromHwnd	Статический. Создает новый объект Graphics из указанного дескриптора для окна
FromHwndInternal	Статический. Внутренний метод. Не используется
FromImage	Статический. Создает новый объект Graphics из заданного объекта Image
GetHalftonePalette	Статический. Получает дескриптор текущей полутоновой палитры Windows
GetHashCode	Служит хэш-функцией для конкретного типа, пригоден для использования в алгоритмах хэширования и структурах данных, например в хэш-таблице

GetHdc	Получает дескриптор контекста устройства, связанный с данным объектомGraphics
GetLifetimeService	Извлекает служебный объект текущего срока действия, который управляет средствами срока действия данного экземпляра
GetNearestColor	Получает цвет, ближайший к указанной структуре Color
GetType	Возвращает Type текущего экземпляра
InitializeLifetimeService	Получает служебный объект срока действия, для управления средствами срока действия данного экземпляра
IntersectClip	Перегружен. Обновляет вырезанную область данного объекта Graphics, включая в нее пересечение текущей вырезанной области и указанной структуры Rectangle
IsVisible	Перегружен. Указывает, содержится ли точка, заданная с помощью пары координат, в видимой вырезанной области данного объекта Graphics
MeasureCharacterRanges	Получает массив объектов Region, каждый из которых связывает диапазон позиций символов в рамках указанной строки
MeasureString	Перегружен. Измеряет указанную строку в процессе ее создания с помощью заданного объекта Font
MultiplyTransform	Перегружен. Умножает универсальное преобразование данного объектаGraphics на преобразование указанного объекта Matrix
ReleaseHdc	Освобождает дескриптор контекста устройства, полученный в результате предыдущего вызова метода GetHdc данного объекта Graphics
ReleaseHdcInternal	Внутренний метод. Не используется
ResetClip	Сбрасывает вырезанную область данного объекта Graphics и делает ее бесконечной
ResetTransform	Сбрасывает матрицу универсального преобразования данного объектаGraphics и делает ее единичной матрицей
Restore	Восстанавливает состояние данного объекта Graphics, возвращая его к состоянию объекта GraphicsState
RotateTransform	Перегружен. Применяет заданное вращение к матрице

	преобразования данного объекта Graphics
Save	Сохраняет текущее состояние данного объекта Graphics и связывает сохраненное состояние с объектом GraphicsState
ScaleTransform	Перегружен. Применяет указанную операцию масштабирования к матрице преобразования данного объекта Graphics путем ее добавления к матрице преобразования объекта
SetClip	Перегружен. Задает в качестве вырезанной области данного объектаGraphics свойство Clip указанного объекта Graphics
ToString	Возвращает String, который представляет текущий Object
TransformPoints	Перегружен. Преобразует массив точек из одного координатного пространства в другое, используя текущее универсальное преобразование и преобразование страницы данного объекта Graphics
TranslateClip	Перегружен. Переводит вырезанную область данного объекта Graphics в указанном объеме в горизонтальном и вертикальном направлениях.
TranslateTransform	Перегружен. Добавляет заданный перевод к матрице преобразования данного объекта Graphics

4.2 Битовая карта как поверхность для рисования

Приводимое ниже приложение демонстрирует технику рисования на невидимых виртуальных поверхностях – в буквальном смысле в оперативной памяти.

Приложение моделирует случайное блуждание множества однородных частиц. Несмотря на интенсивный вывод графической информации, удается избежать эффекта мигания, который возникает при непосредственной модификации внешнего вида элементов управления.

Демонстрация предварительно нарисованной в памяти картинки происходит при помощи элемента управления типа PictureBox, который благодаря свойству объекта Image обеспечивает быстрое отображение выводимой графической информации.

Итак, данное Windows-приложение включает объявление трех классов:

- класса xPoint, объекты которого периодически изменяют собственное положение в пределах клиентской области окна;
- класса cForm, который обладает свойствами сохранения постоянного соотношения собственных размеров и размеров клиентской области, и обеспечивает отображение множества блуждающих объектов – представителей класса xPoint;
- класса Program, который в соответствии с предопределенными алгоритмами обеспечивает создание и выполнение приложения.

```
using System;
 using System.Collections.Generic;
 using System.Drawing;
 using System.Text;
 namespace cForm
 {
 class xPoint
 {
 // Собственный статический генератор случайных чисел.
 // Используется в конструкторе.
 public static Random rnd = new Random();

 // Цвет объекта.
 public Color xColor;

 // Текущая позиция. Относительные значения координат X и Y.
 public Point p;

 // Обеспечение перемещения. Текущие значения координат могут
 // изменяться как по X, так и по Y.
 public Point direction;

 // Счетчик циклов сохранения выбранного направления.
 // Объект на протяжении фиксированного интервала времени может
 // сохранять ранее избранное направление движения.
 // В течение n.X "тиков" для X в течение n.Y "тиков" для Y.
```

```
public Point n;

public xPoint(int X, int Y, int Xn, int Yn)
{
n = new Point(rnd.Next(0, Xn), rnd.Next(0, Yn));
direction = new Point(rnd.Next(-1, 2), rnd.Next(-1, 2));
p = new Point(X, Y);
xColor =
  Color.FromArgb(150, rnd.Next(0, 256), rnd.Next(0, 256), rnd.Next(0, 256));
}

// Статический метод. Еще вопрос, где следовало размещать его
// объявление... Определение нового положения амебы.
// Просматривается ВЕСЬ список.
// Ее перемещение ограничивается физическими размерами клиентской области
// окна приложения и определяется по двум координатам (X и Y).
// В каждый момент она может оставаться на месте, либо изменить
//  свое положение на один "шаг" по каждой из осей координат
// ("вверх" или "вниз" по оси Y и "вперед" или "назад" по оси X).
public static void setNextPosition(cForm cf, xPoint[] pt)
{
int i;
float xy;

// Итак, определение текущей позиции объекта.
// Просмотр массива...
for (i = 0; i < pt.Length; i++)
{
// Сначала разбираемся со значением координаты X.
// Вычисляем возможную позицию после перемещения объекта.
xy = (float)((pt[i].p.X + pt[i].direction.X) * cf.rPointGet);
// Вполне возможно, что это вполне подходящая позиция.
// И надо всего лишь проверить две вещи:
// 1. не выскочит ли объект за пределы клиентской области окна
// приложения,
// 2. не настало ли время поменять направление движения.
if (
xy < 0 || xy > cf.ClientSize.Width                  // 1.
||
pt[i].n.X > cf.NX                          // 2.
)
{
pt[i].n.X = 0; // Обнулили счетчик циклов сохранения выбранного направления.
// Процедура изменения направления перемещения по оси X.
// На ближайшую перспективу объект может переместиться
// вперед:           pt[i].direction.X == 1,
// назад:            pt[i].direction.X == –1,
// остаться на месте: pt[i].direction.X == 0.
// Главное — это не выйти за пределы клиентской области окна приложения.
   pt[i].direction.X = xPoint.rnd.Next(-1, 2);
   xy = (float)((pt[i].p.X + pt[i].direction.X) * cf.rPointGet);
   if (xy >= 0 && xy <= cf.ClientSize.Width)
   {
      // Направление выбрано, перемещение произведено.
      pt[i].p.X += pt[i].direction.X;
   }
   else
   {
      // Выбранное направление движения приводит к выходу объекта
      // за пределы клиентской области окна приложения.
      // На ближайшие cf.NX тактов объект остается неподвижен по оси X.
      pt[i].direction.X = 0;
   }
```

```
}
else
{
// Осуществили очередное перемещение по оси X.
pt[i].p.X += pt[i].direction.X;
pt[i].n.X++;
}

xy = (float)((pt[i].p.Y + pt[i].direction.Y) * cf.rPointGet);
// Вполне возможно, что это вполне подходящая позиция.
// И надо всего лишь проверить две вещи:
// 1. не выскочит ли объект за пределы клиентской области
// окна приложения,
// 2. не настало ли время поменять направление движения.
if (
xy < 0 || xy > cf.ClientSize.Height // 1.
||
pt[i].n.Y > cf.NY                   // 2.
)
{
pt[i].n.Y = 0; // Обнулили счетчик циклов сохранения выбранного направления.
// Процедура изменения направления перемещения по оси Y.
// На ближайшую перспективу объект может переместиться
// вверх:            pt[i].direction.Y == 1,
// вниз:             pt[i].direction.Y == -1,
// остаться на месте: pt[i].direction.Y == 0.
// Главное — это не выйти за пределы клиентской области окна приложения.
pt[i].direction.Y = xPoint.rnd.Next(-1, 2);
xy = (float)((pt[i].p.Y + pt[i].direction.Y) * cf.rPointGet);
if (xy >= 0 && xy <= cf.ClientSize.Height)
{
   // Направление выбрано, перемещение произведено.
   pt[i].p.Y += pt[i].direction.Y;
}
else
{
   // Выбранное направление движения приводит к выходу объекта
   // за пределы клиентской области окна приложения.
   // На ближайшие cf.NY тактов объект остается неподвижен по оси Y.
   pt[i].direction.Y = 0;
}
}
else
{
// Осуществили очередное перемещение по оси Y.
pt[i].p.Y += pt[i].direction.Y;
pt[i].n.Y++;
} } } } }
```

4.3 Собственные элементы управления

Известно по крайней мере три возможных подхода к разработке новых элементов управления:

- объединение стандартных элементов управления в группы (составные элементы управления);
- объявление новых классов, наследующих от существующих элементов управления;
- написание новых элементов "с нуля".

Разработка составных элементов управления предполагает объявление класса, производного от класса UserControl и использование Мастера UserControl, для добавления вложенных элементов управления с последующей настройкой образующих элементов.

Новый элемент управления может быть построен на основе класса – наследника какого-либо из существующих элементов управления. В этом случае в новом классе удается частично использовать функциональности ранее объявленного класса, возможно, сохраняя при этом внешний вид элемента. Например, можно объявить собственный вариант класса кнопки, который будет наследовать классу Button.

Написание нового элемента "с нуля" отличается от предыдущего варианта разработки выбором базового класса. В этом случае основываются на классе Control, который не предоставляет потомкам даже элементарного графического интерфейса. Процесс визуализации в этом случае обеспечивается переопределяемым обработчиком события Paint. При этом переопределяется виртуальный метод базового класса OnPaint с единственным аргументом типа PaintEventArgs, который содержит информацию о клиентской области элемента управления. Член этого класса объект типа Graphics – обеспечивает формирование представления элемента управления. Второй член класса – объект типа ClipRectangle – описывает доступную клентскую область элемента управления.

Следует отметить, что между двумя последними способами определения элементов управления не существует четких границ. В обоих случаях основанием для классификации оказывается объем работы по доопределению и переопределению методов и свойств вновь создаваемого класса элементов управления.

5. Описание классов.

В программе имеются 3 класса: Vector3D, Cube и класс формы FrmRender

Vector3D – представляет собой трехмерный вектор в пространстве. У него есть 3 поля для описания координат в пространстве. Тип этих полей был выбран float.

В нем реализованы следующие методы:

- Конструктор с параметрами int
- Конструктор с параметрами double
- Конструктор с параметрами float
- Конструктор без параметров

А также был перегружен метод ToString().

Cube – представляет собой трехмерный куб в пространстве. В кубе для представления сторон куба был реализован класс Face, внутри которого реализовано перечисление Side, для представления сторон куба.

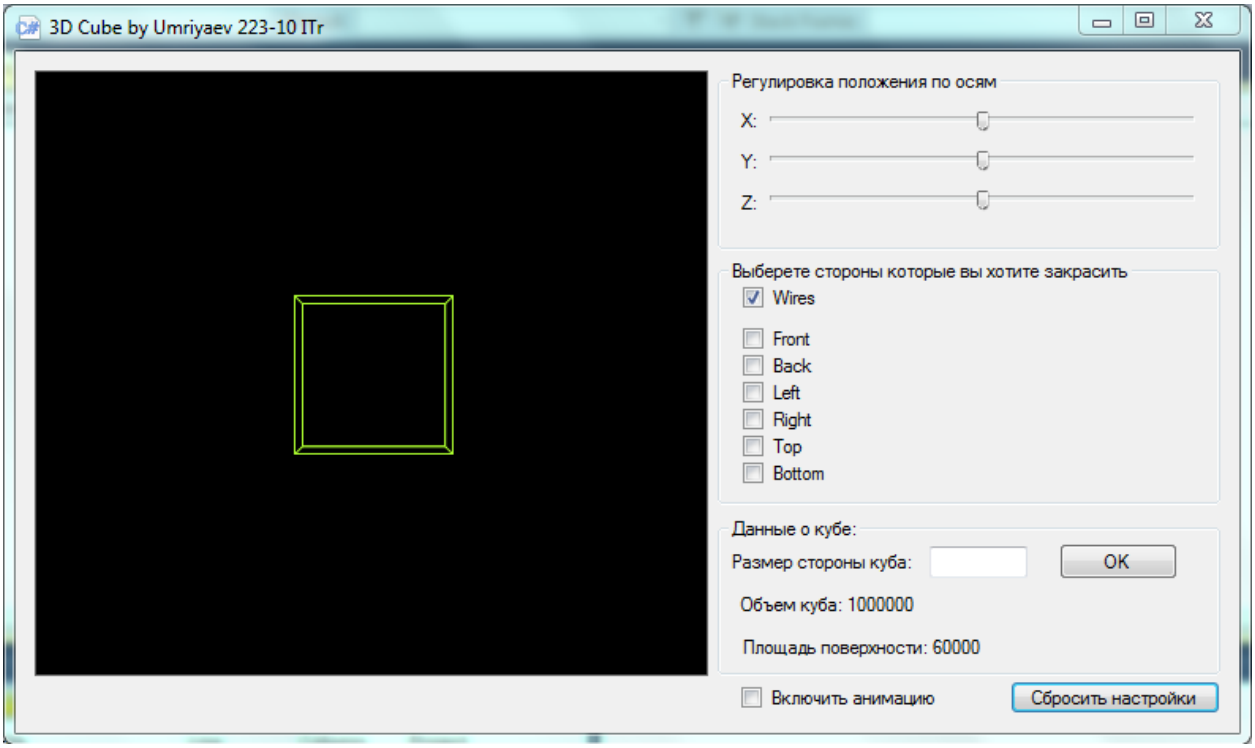
В классе реализованы следующие методы:

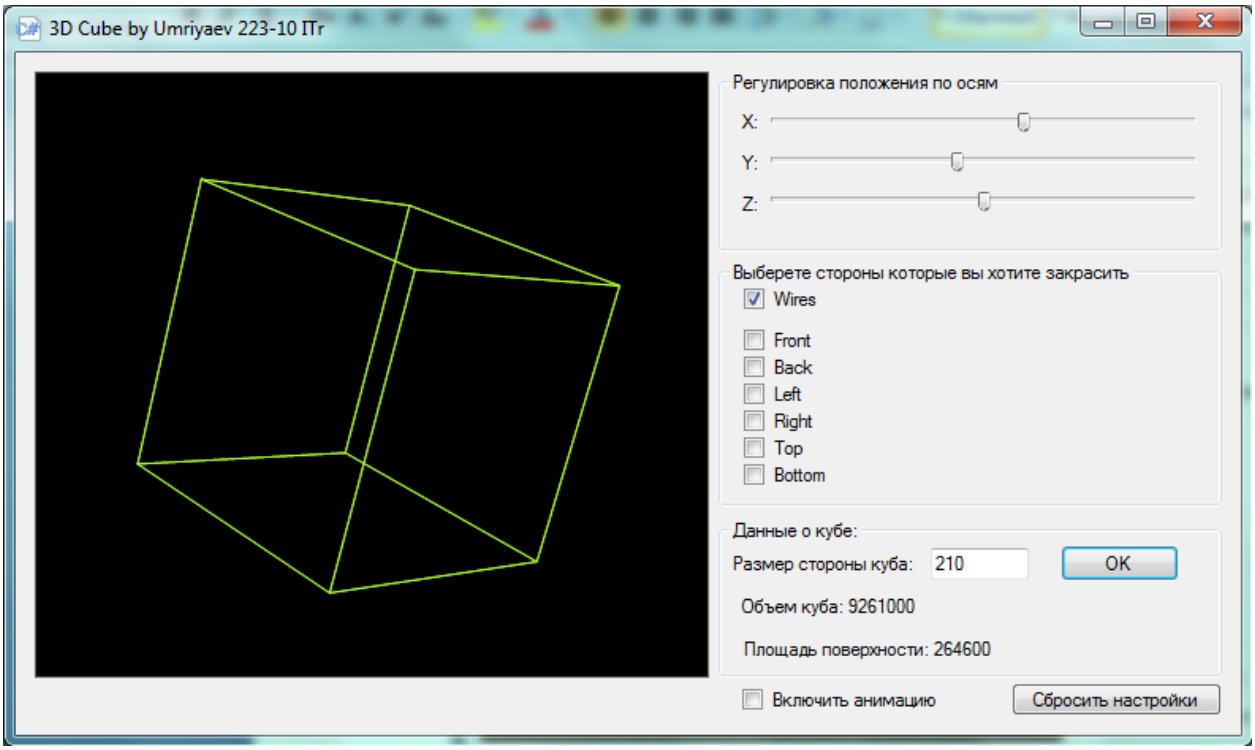
- RotateX – поворот куба по оси X
- RotateY – поворот куба по оси Y
- RotateZ – поворот куба по оси Z
- Update2DPoints – обновление двухмерных точек куба при любом изменении куба (повороты)
- RotateCubeX, RotateCubeY, RotateCubeZ – поворот куба по соответствующим осям.
- DrawCube – собственно рисование куба
- Get2D – перевод двухмерных координат на трехмерные
- Translate – изменение координат куба при повороте

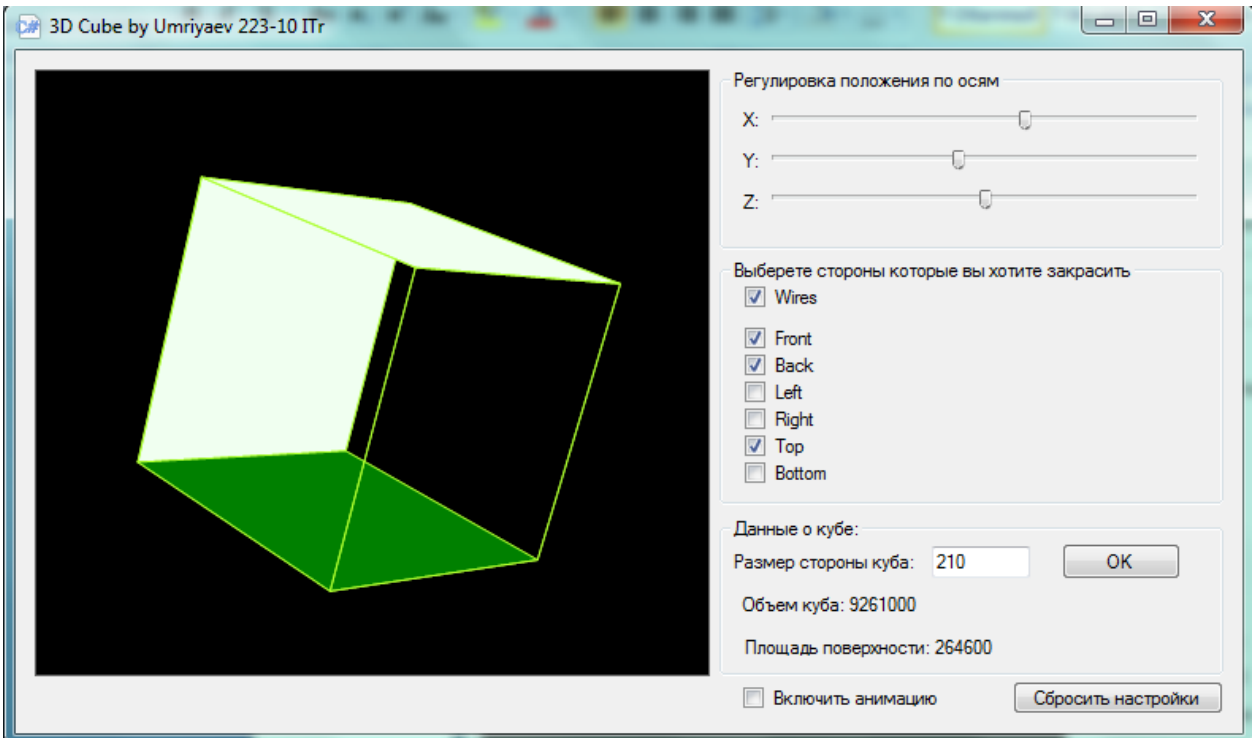
Основное отображение данных на окне приложения производится с помощью класса FrmRender.

В этом классе реализованы обработчики событий слайдеров поворота, кнопок.

6. Результат работы программы






3D Cube by Umriyaev 223-10 ITr
Регулировка положения по осям
X:
Y:
Z:
Выберете стороны которые вы хотите закрасить
Wires
Front
Back
Left
Right
Top
Bottom
Данные о кубе:
Размер стороны куба:
210
OK
Объем куба: 9261000
Площадь поверхности: 264600
Включить анимацию
Сбросить настройки

7. Заключение

Данная курсовая работа по предмету: «Объектно-ориентированные языки программирования» на тему: «Классы». В данной курсовой работе были рассмотрены методы для рисования GDI+, а также создавались свои классы. Данная программа была разработана на языке C# с помощью Microsoft Visual Studio 2010 .

8. Список использованной литературы

1. C# 3.0: полное руководство. Полный справочник по C# 2008, Герберт Шилдт; 992 стр., с ил.; 2009, 3 кв.; Вильямс

2. C# 2008 и платформа .NET 3.5 Framework: вводный курс, 2-е издание, Кристиан Гросс; 480 стр., с ил.; 2009, 3 кв.; Вильямс

3. Visual C# 2008: базовый курс. Visual Studio® 2008, Карли Уотсон, Кристиан Нейгел, Якоб Хаммер Педерсен, и др.; 1216 стр., с ил.; 2009, 2 кв.; Диалектика

4. Эффективная работа с унаследованным кодом, Майкл Физерс; 400 стр., с ил.; 2009, 1 кв.; Вильямс

5. Объектно-ориентированный анализ и проектирование с примерами приложений (UML 2). Третье издание, Гради Буч, Роберт А. Максимчук, Майкл У. Энгл, Бобби Дж. Янг, Джим Коналлен, Келли А. Хьюстон ; 720 стр., с ил.; 2010, 1 кв.; Вильямс

6. Применение DDD и шаблонов проектирования: проблемно-ориентированное проектирование приложений с примерами на C# и .NET, Джимми Нильссон; 560 стр., с ил.; 2007, 3 кв.; Вильямс

Приложение

Исходный код программы:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace _DCubeNoGimbalLock
{
    class Math3D
    {
        const double PIOVER180 = Math.PI / 180.0;

        public class Vector3D
        {
            public float x;
            public float y;
            public float z;

            public Vector3D(int _x, int _y, int _z)
            {
                x = _x;
                y = _y;
                z = _z;
            }

            public Vector3D(double _x, double _y, double _z)
            {
                x = (float)_x;
                y = (float)_y;
                z = (float)_z;
            }

            public Vector3D(float _x, float _y, float _z)
            {
                x = _x;
                y = _y;
                z = _z;
            }

            public Vector3D()
            {
            }

            public override string ToString()
            {
                return "(" + x.ToString() + ", " + y.ToString() + ", " + z.ToString() +
")";
            }
        }

        internal class Camera
        {
            public Vector3D position = new Vector3D();
        }

        public class Cube
        {
```

```
internal class Face : IComparable<Face>
{
    public enum Side
    {
        Front,
        Back,
        Left,
        Right,
        Top,
        Bottom
    }

    public PointF[] Corners2D;
    public Vector3D[] Corners3D;
    public Vector3D Center;
    public Side CubeSide;

    public Face()
    {
    }

    public int CompareTo(Face otherFace)
    {
        return (int)(this.Center.z - otherFace.Center.z);
    }
}

public int width = 0;
public int height = 0;
public int depth = 0;

float xRotation = 0.0f;
float yRotation = 0.0f;
float zRotation = 0.0f;

bool drawWires = true;
bool fillFront;
bool fillBack;
bool fillLeft;
bool fillRight;
bool fillTop;
bool fillBottom;

Vector3D cubeOrigin;

Face[] faces;

public float RotateX
{
    get { return xRotation; }
    set
    {
        RotateCubeX(value - xRotation);
        xRotation = value;
    }
}

public float RotateY
{
    get { return yRotation; }
    set
    {
        RotateCubeY(value - yRotation);
        yRotation = value;
```

```
        }
    }

    public float RotateZ
    {
        get { return zRotation; }
        set
        {
            RotateCubeZ(value - zRotation);
            zRotation = value;
        }
    }

    public bool DrawWires
    {
        get { return drawWires; }
        set { drawWires = value; }
    }
    public bool FillFront
    {
        get { return fillFront; }
        set { fillFront = value; }
    }
    public bool FillBack
    {
        get { return fillBack; }
        set { fillBack = value; }
    }
    public bool FillLeft
    {
        get { return fillLeft; }
        set { fillLeft = value; }
    }
    public bool FillRight
    {
        get { return fillRight; }
        set { fillRight = value; }
    }
    public bool FillTop
    {
        get { return fillTop; }
        set { fillTop = value; }
    }
    public bool FillBottom
    {
        get { return fillBottom; }
        set { fillBottom = value; }
    }



    private void Update2DPoints(Point drawOrigin)
    {
        for (int i = 0; i < faces.Length; i++)
        {
            Update2DPoints(drawOrigin, i);
        }
    }

    private void Update2DPoints(Point drawOrigin, int faceIndex)
    {
        PointF[] point2D = new PointF[4];
        float zoom = (float)Screen.PrimaryScreen.Bounds.Width / 1.5f;
        Point tmpOrigin = new Point(0, 0);
```

```
                Math3D.Vector3D vec;
                for (int i = 0; i < point2D.Length; i++)
                {
                    vec = faces[faceIndex].Corners3D[i];
                    point2D[i] = Get2D(vec, drawOrigin);
                }

                faces[faceIndex].Corners2D = point2D;
            }

            private void RotateCubeX(float deltaX)
            {
                for (int i = 0; i < faces.Length; i++)
                {
                    Vector3D point0 = new Vector3D(0, 0, 0);
                    faces[i].Corners3D = Math3D.Translate(faces[i].Corners3D, cubeOrigin,
point0);
                    faces[i].Corners3D = Math3D.RotateX(faces[i].Corners3D, deltaX);
                    faces[i].Corners3D = Math3D.Translate(faces[i].Corners3D, point0,
cubeOrigin);

                    faces[i].Center = Math3D.Translate(faces[i].Center, cubeOrigin,
point0);
                    faces[i].Center = Math3D.RotateX(faces[i].Center, deltaX);
                    faces[i].Center = Math3D.Translate(faces[i].Center, point0,
cubeOrigin);
                }
            }


private void RotateCubeY(float deltaY)
            {
                for (int i = 0; i < faces.Length; i++)
                {
                    Vector3D point0 = new Vector3D(0, 0, 0);
                    faces[i].Corners3D = Math3D.Translate(faces[i].Corners3D, cubeOrigin,
point0);
                    faces[i].Corners3D = Math3D.RotateY(faces[i].Corners3D, deltaY);
                    faces[i].Corners3D = Math3D.Translate(faces[i].Corners3D, point0,
cubeOrigin);
                    faces[i].Center = Math3D.Translate(faces[i].Center, cubeOrigin,
point0);
                    faces[i].Center = Math3D.RotateY(faces[i].Center, deltaY);
                    faces[i].Center = Math3D.Translate(faces[i].Center, point0,
cubeOrigin);
                }
            }

            private void RotateCubeZ(float deltaZ)
            {
                for (int i = 0; i < faces.Length; i++)
                {
                    //Apply rotation
                    //------Rotate points
                    Vector3D point0 = new Vector3D(0, 0, 0);
                    faces[i].Corners3D = Math3D.Translate(faces[i].Corners3D, cubeOrigin,
point0); //Move corner to origin
                    faces[i].Corners3D = Math3D.RotateZ(faces[i].Corners3D, deltaZ);
                    faces[i].Corners3D = Math3D.Translate(faces[i].Corners3D, point0,
cubeOrigin); //Move back

                    //-------Rotate center
                    faces[i].Center = Math3D.Translate(faces[i].Center, cubeOrigin,
point0);
                    faces[i].Center = Math3D.RotateZ(faces[i].Center, deltaZ);
```

```
                    faces[i].Center = Math3D.Translate(faces[i].Center, point0,
cubeOrigin);
                }
            }

            public Bitmap DrawCube(Point drawOrigin)
            {
                //Get the corresponding 2D
                Update2DPoints(drawOrigin);

                //Get the bounds of the final bitmap
                Rectangle bounds = getDrawingBounds();
                bounds.Width += drawOrigin.X;
                bounds.Height += drawOrigin.Y;

                Bitmap finalBmp = new Bitmap(bounds.Width, bounds.Height);
                Graphics g = Graphics.FromImage(finalBmp);

                g.SmoothingMode = SmoothingMode.AntiAlias;

                Array.Sort(faces); //sort faces from closets to farthest
                //message();
                for (int i = faces.Length - 1; i >= 0; i--) //draw faces from back to
front
                {
                    switch (faces[i].CubeSide)
                    {
                        case Face.Side.Front:
                            if (fillFront)
                                g.FillPolygon(Brushes.Honeydew, GetFrontFace());
                            break;
                        case Face.Side.Back:
                            if (fillBack)
                                g.FillPolygon(Brushes.Green, GetBackFace());
                            break;
                        case Face.Side.Left:
                            if (fillLeft)
                                g.FillPolygon(Brushes.Honeydew, GetLeftFace());
                            break;
                        case Face.Side.Right:
                            if (fillRight)
                                g.FillPolygon(Brushes.Green, GetRightFace());
                            break;
                        case Face.Side.Top:
                            if (fillTop)
                                g.FillPolygon(Brushes.Honeydew, GetTopFace());
                            break;
                        case Face.Side.Bottom:
                            if (fillBottom)
                                g.FillPolygon(Brushes.Green, GetBottomFace());
                            break;
                        default:
                            break;
                    }

                    if (drawWires)
                    {
                        g.DrawLine(Pens.GreenYellow, faces[i].Corners2D[0],
faces[i].Corners2D[1]);
                        g.DrawLine(Pens.GreenYellow, faces[i].Corners2D[1],
faces[i].Corners2D[2]);
                        g.DrawLine(Pens.GreenYellow, faces[i].Corners2D[2],
faces[i].Corners2D[3]);
                        g.DrawLine(Pens.GreenYellow, faces[i].Corners2D[3],
faces[i].Corners2D[0]);
```

```
            }
        }

        g.Dispose();

        return finalBmp;
    }

    //Converts 3D points to 2D points
    private PointF Get2D(Vector3D vec, Point drawOrigin)
    {
        PointF point2D = Get2D(vec);
        return new PointF(point2D.X + drawOrigin.X, point2D.Y + drawOrigin.Y);
    }

    private PointF Get2D(Vector3D vec)
    {
        PointF returnPoint = new PointF();

        float zoom = (float)Screen.PrimaryScreen.Bounds.Width / 1.5f;
        Camera tempCam = new Camera();

        tempCam.position.x = cubeOrigin.x;
        tempCam.position.y = cubeOrigin.y;
        tempCam.position.z = (cubeOrigin.x * zoom) / cubeOrigin.x;

        float zValue = -vec.z - tempCam.position.z;

        returnPoint.X = (tempCam.position.x - vec.x) / zValue * zoom;
        returnPoint.Y = (tempCam.position.y - vec.y) / zValue * zoom;

        return returnPoint;
    }

    public PointF[] GetFrontFace()
    {
        //Returns the four points corresponding to the front face
        //Get the corresponding 2D
        return getFace(Face.Side.Front).Corners2D;
    }

    public PointF[] GetBackFace()
    {
        return getFace(Face.Side.Back).Corners2D;
    }

    public PointF[] GetRightFace()
    {
        return getFace(Face.Side.Right).Corners2D;
    }

    public PointF[] GetLeftFace()
    {
        return getFace(Face.Side.Left).Corners2D;
    }

    public PointF[] GetTopFace()
    {
        return getFace(Face.Side.Top).Corners2D;
    }

    public PointF[] GetBottomFace()
    {
        return getFace(Face.Side.Bottom).Corners2D;
    }
```

```
    private Face getFace(Face.Side side)
    {
        //Find the correct side
        //Since faces are sorted in order of closest to farthest
        //They won't always be in the same index
        for (int i = 0; i < faces.Length; i++)
        {
            if (faces[i].CubeSide == side)
                return faces[i];
        }

        return null; //not found
    }

public static Vector3D RotateX(Vector3D point3D, float degrees)
{
    //[ a  b  c ] [ x ]   [ x*a + y*b + z*c ]
    //[ d  e  f ] [ y ] = [ x*d + y*e + z*f ]
    //[ g  h  i ] [ z ]   [ x*g + y*h + z*i ]

    //[ 1    0        0   ]
    //[ 0   cos(x)  sin(x)]
    //[ 0   -sin(x) cos(x)]

    double cDegrees = degrees * PIOVER180;
    double cosDegrees = Math.Cos(cDegrees);
    double sinDegrees = Math.Sin(cDegrees);

    double y = (point3D.y * cosDegrees) + (point3D.z * sinDegrees);
    double z = (point3D.y * -sinDegrees) + (point3D.z * cosDegrees);

    return new Vector3D(point3D.x, y, z);
}

public static Vector3D RotateY(Vector3D point3D, float degrees)
{
    //[ cos(x)   0    sin(x)]
    //[   0      1      0   ]
    //[-sin(x)   0    cos(x)]

    double cDegrees = degrees * PIOVER180;
    double cosDegrees = Math.Cos(cDegrees);
    double sinDegrees = Math.Sin(cDegrees);

    double x = (point3D.x * cosDegrees) + (point3D.z * sinDegrees);
    double z = (point3D.x * -sinDegrees) + (point3D.z * cosDegrees);

    return new Vector3D(x, point3D.y, z);
}

public static Vector3D RotateZ(Vector3D point3D, float degrees)
{
    //[ cos(x)  sin(x) 0]
    //[ -sin(x) cos(x) 0]
    //[    0     0     1]

    double cDegrees = degrees * PIOVER180;
    double cosDegrees = Math.Cos(cDegrees);
    double sinDegrees = Math.Sin(cDegrees);

    double x = (point3D.x * cosDegrees) + (point3D.y * sinDegrees);
    double y = (point3D.x * -sinDegrees) + (point3D.y * cosDegrees);

    return new Vector3D(x, y, point3D.z);
```

```
        }

        public static Vector3D Translate(Vector3D points3D, Vector3D oldOrigin, Vector3D
newOrigin)
        {
            Vector3D difference = new Vector3D(newOrigin.x - oldOrigin.x, newOrigin.y -
oldOrigin.y, newOrigin.z - oldOrigin.z);
            points3D.x += difference.x;
            points3D.y += difference.y;
            points3D.z += difference.z;
            return points3D;
        }



namespace _DCubeNoGimbalLock
{
    public partial class FrmRender : Form
    {
        [System.Runtime.InteropServices.DllImportAttribute("gdi32.dll")]
        private static extern bool BitBlt(IntPtr hdcDest, int nXDest, int nYDest, int
nWidth, int nHeight, IntPtr hdcSrc, int nXSrc, int nYSrc, System.Int32 dwRop);

        [System.Runtime.InteropServices.DllImportAttribute("user32.dll")]
        public static extern IntPtr GetDC(IntPtr hwnd);

        [System.Runtime.InteropServices.DllImportAttribute("user32.dll")]
        public static extern IntPtr ReleaseDC(IntPtr hwnd, IntPtr hdc);

        public FrmRender()
        {
            InitializeComponent();
        }

        Math3D.Cube mainCube;
        Point drawOrigin;
        int sideLength = 100;

        private void FrmRender_Load(object sender, EventArgs e)
        {
            mainCube = new Math3D.Cube(sideLength, sideLength, sideLength);
            drawOrigin = new Point(pictureBox1.Width / 2, pictureBox1.Height / 2);
            lblVolume.Text = "Объем куба: " + (sideLength * sideLength *
sideLength).ToString();
            lblArea.Text = "Площадь поверхности: " + (6 * sideLength *
sideLength).ToString();
        }

        private void Render()
        {
            mainCube.RotateX = (float)tX.Value;
            mainCube.RotateY = (float)tY.Value;
            mainCube.RotateZ = (float)tZ.Value;

            pictureBox1.Image = mainCube.DrawCube(drawOrigin);
        }

        private void tX_Scroll(object sender, EventArgs e)
        {
            this.Refresh();
        }

        private void tY_Scroll(object sender, EventArgs e)
        {
            this.Refresh();
```

```
}

private void tZ_Scroll(object sender, EventArgs e)
{
    this.Refresh();
}

private void btnReset_Click(object sender, EventArgs e)
{
    tX.Value = 0;
    tY.Value = 0;
    tZ.Value = 0;

    chWires.Checked = true;
    chFront.Checked = false;
    chBack.Checked = false;
    chLeft.Checked = false;
    chRight.Checked = false;
    chTop.Checked = false;
    chBottom.Checked = false;

    mainCube = new Math3D.Cube(100, 100, 100); //Start over
    this.Refresh();
}

private void Form1_Paint(object sender, PaintEventArgs e)
{
    Render();
}

private void chWires_CheckedChanged(object sender, EventArgs e)
{
    mainCube.DrawWires = chWires.Checked;
    this.Refresh();
}

private void chFront_CheckedChanged(object sender, EventArgs e)
{
    mainCube.FillFront = chFront.Checked;
    this.Refresh();
}

private void chBack_CheckedChanged(object sender, EventArgs e)
{
    mainCube.FillBack = chBack.Checked;
    this.Refresh();
}

private void chLeft_CheckedChanged(object sender, EventArgs e)
{
    mainCube.FillLeft = chLeft.Checked;
    this.Refresh();
}

private void chRight_CheckedChanged(object sender, EventArgs e)
{
    mainCube.FillRight = chRight.Checked;
    this.Refresh();
}

private void chTop_CheckedChanged(object sender, EventArgs e)
{
    mainCube.FillTop = chTop.Checked;
    this.Refresh();
}
```

```
private void chBottom_CheckedChanged(object sender, EventArgs e)
{
    mainCube.FillBottom = chBottom.Checked;
    this.Refresh();
}

private void timer1_Tick(object sender, EventArgs e)
{
    if (checkBox1.Checked == true)
    {
        timer2.Enabled = true;
    }
    else
    {
        timer2.Enabled = false;
    }
}
public int Zvr = 0;
bool plus= true;
private void timer2_Tick(object sender, EventArgs e)
{
    tX.Value = tX.Value == 360 ? 0 : tX.Value+1;
    this.Refresh();
    label4.Text = tX.Value.ToString();
}

private void KeyPressed(object sender, KeyPressEventArgs e)
{
    TextBox txtBox = (TextBox)sender;
    if (e.KeyChar >= '0' && e.KeyChar <= '9')
        return;

    if (e.KeyChar == '.') e.KeyChar = ',';

    if (e.KeyChar == ',')
    {
        if ((txtBox.Text.IndexOf(',') != -1 || txtBox.Text.Length == 0))
            e.Handled = true;

        return;
    }

    if (char.IsControl(e.KeyChar))
    {
        if (e.KeyChar == (char)Keys.Back)
            return;
    }

    e.Handled = true;
}
```

```
 private void button1_Click(object sender, EventArgs e)
        {
            float side;
            float.TryParse(textBox1.Text, out side);
            if (side > 300 || side < 10)
                MessageBox.Show("Размер куба не может быть больше чем 300 и меньше чем
1");
            else
            {
                sideLength = (int)side;
                mainCube = new Math3D.Cube(sideLength, sideLength, sideLength);
                drawOrigin = new Point(pictureBox1.Width / 2, pictureBox1.Height / 2);
                Render();
                this.Refresh();
                lblVolume.Text = "Объем куба: " + (side * side * side).ToString();
                lblArea.Text = "Площадь поверхности: " + (6 * side * side).ToString();
            }
        }
    }
}
```