*Федеральное государственное бюджетное образовательное учреждение* *высшего образования*

|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана»***  ***(национальный исследовательский университет)*** |

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА Компьютерные Системы и сети (ИУ6)

**Отчет**

**По домашней работе №1. Часть 1.**

**Название домашней работы:** Программирование с использованием классов. Графика. Обработка события нажатия клавиши мыши. Наследование. Полиморфизм.

**Дисциплина**: Объектно-ориентированное программирование

Вариант №1 (Усложнённый)

Студент гр. ИУ6-22**\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Н. И. Боярских

(Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_**\_\_\_\_\_\_\_\_\_ Черноусова Татьяна Геннадьевна

(Подпись, дата) (И.О. Фамилия)

**Программирование с использованием классов. Графика. Обработка события нажатия клавиши мыши. Наследование. Полиморфизм.**

**Задание**

Разработать иерархию классов Delphi. Разместить определение классов в отдельном модуле. Разработать программу, содержащую описание трех графических объектов:

правильный треугольник, треугольная пирамида, треугольная призма (прозрачная).

Реализуя механизм полиморфизма, привести объекты в синхронное вращение вокруг их вертикальных осей. Параметры задаются с использованием интерфейсных элементов.

В отчете показать иерархии используемых классов Delphi и разработанных классов, граф состояния пользовательского интерфейса и объектную декомпозицию.

**Объектная декомпозиция**

**

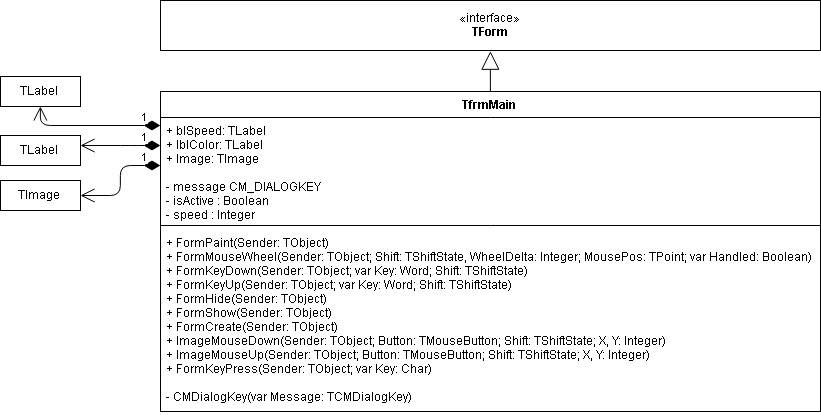
*Рисунок 1 – Объектная декомпозиция задачи*

**Граф состояний пользовательского интерфейса**

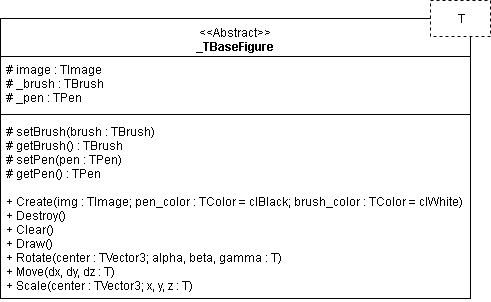
**

*Рисунок 2 – Граф состояний пользовательского интерфейса*

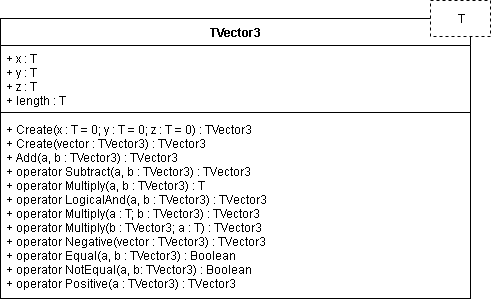
**Иерархия классов, использованных в программе «Вращатель фигур»**

**

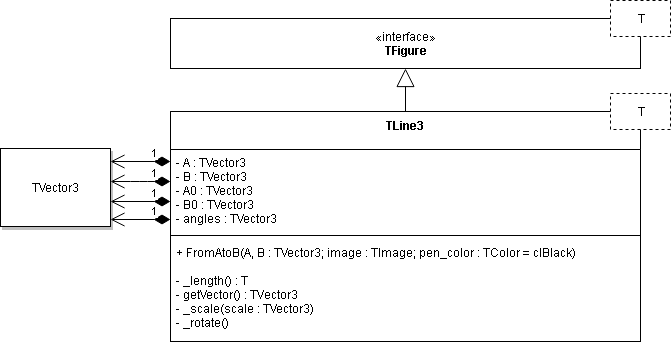
*Рисунок 3 – Диаграмма класса формы главного окна программы*

**

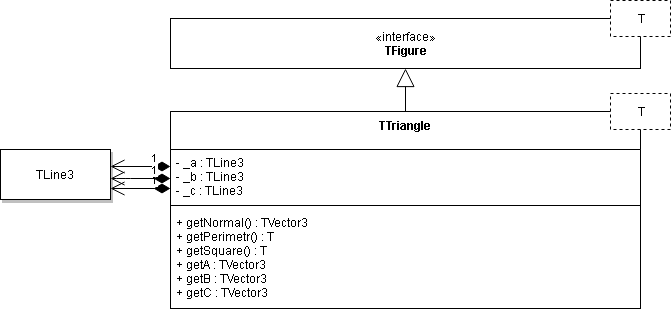
*Рисунок 4 – Диаграмма базового абстрактного класса фигуры*

**

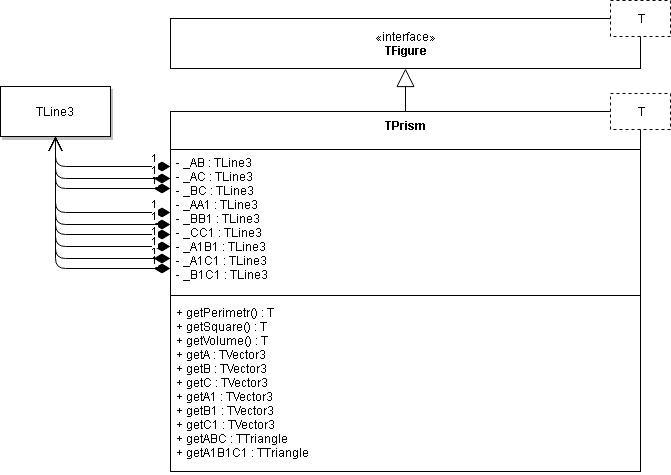
*Рисунок 5 – Диаграмма структуры вектора*

**

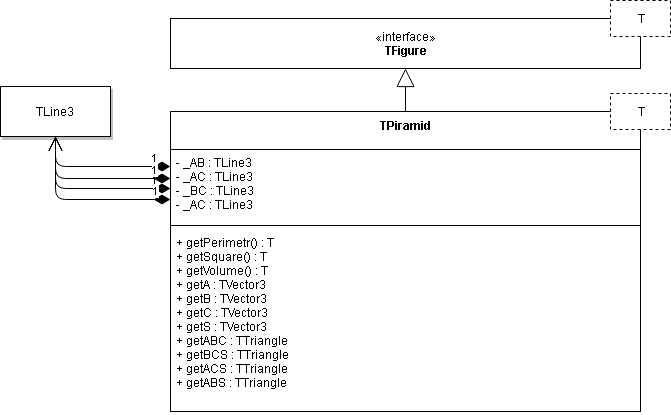
*Рисунок 6 – Диаграмма класса линии*

**

*Рисунок 7 – Диаграмма класса треугольника*

**

*Рисунок 8 – Диаграмма класса призмы*

**

*Рисунок 9 – Диаграмма класса пирамиды*

**Текст программы «Вращатель фигур»**

(Так как программа разрабатывалась в старой версии Delphi (2006 года релиза), которая не поддерживает дженерики, пришлось создать небольшой самописный аналог)

1) Код шаблонизатора:

(\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Шаблонизатор классов и функций \*

\* \*

\* Использование: \*

\* {$DEFINE TYPE\_Template} \*

\* {$INCLUDE Templater.pas} \*

\* \*

\* Теперь T - тип, равный TYPE \*

\* Можно создать много файлов с одинаковым кодом, \*

\* различие будет только в директиве DEFINE \*

\* \*

\* Таким образом, можно создавать шаблонный код \*

\* Для одинаковых действий с разными типами данных \*

\* \*

\* Пример: \*

\* {$DEFINE Integer\_Template} \*

\* {$INCLUDE Templater.pas} \*

\* var number : T; \*

\* number := 12; \*

\* \*

\* Автор: Боярских Никита \*

\* ИУ6-22 МГТУ им. Н. Э. Баумана \*

\* 8 (916) 385-76-22 \*

\* N02@yandex.ru \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

{$IFDEF Shortint\_Template}

{$UNDEF Integer\_Template}

{$UNDEF Longint\_Template}

{$UNDEF Single\_Template}

{$UNDEF Real\_Template}

{$UNDEF Double\_Template}

{$UNDEF Extended\_Template}

Type T = Shortint;

{$ENDIF}

{$IFDEF Integer\_Template}

{$UNDEF Shortint\_Template}

{$UNDEF Longint\_Template}

{$UNDEF Single\_Template}

{$UNDEF Real\_Template}

{$UNDEF Double\_Template}

{$UNDEF Extended\_Template}

Type T = Integer;

{$ENDIF}

{$IFDEF Longint\_Template}

{$UNDEF Shortint\_Template}

{$UNDEF Integer\_Template}

{$UNDEF Single\_Template}

{$UNDEF Real\_Template}

(Продолжение кода на следующей странице)

(Продолжение кода)

{$UNDEF Double\_Template}

{$UNDEF Extended\_Template}

Type T = Longint;

{$ENDIF}

{$IFDEF Single\_Template}

{$UNDEF Shortint\_Template}

{$UNDEF Integer\_Template}

{$UNDEF Longint\_Template}

{$UNDEF Real\_Template}

{$UNDEF Double\_Template}

{$UNDEF Extended\_Template}

Type T = Single;

{$ENDIF}

{$IFDEF Real\_Template}

{$UNDEF Shortint\_Template}

{$UNDEF Integer\_Template}

{$UNDEF Longint\_Template}

{$UNDEF Single\_Template}

{$UNDEF Double\_Template}

{$UNDEF Extended\_Template}

Type T = Real;

{$ENDIF}

{$IFDEF Double\_Template}

{$UNDEF Shortint\_Template}

{$UNDEF Integer\_Template}

{$UNDEF Longint\_Template}

{$UNDEF Single\_Template}

{$UNDEF Real\_Template}

{$UNDEF Extended\_Template}

Type T = Double;

{$ENDIF}

{$IFDEF Extended\_Template}

{$UNDEF Shortint\_Template}

{$UNDEF Integer\_Template}

{$UNDEF Longint\_Template}

{$UNDEF Single\_Template}

{$UNDEF Real\_Template}

{$UNDEF Double\_Template}

Type T = Extended;

{$ENDIF}

2) Код главного модуля:

UNIT Main;

INTERFACE

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, ComCtrls, StdCtrls, ExtCtrls,

ExtendedVector, ExtendedLine,

ExtendedTriangle, ExtendedPiramid, ExtendedPrism;

const SPEED\_STEP = 1; // Шаг, на который будет увеличиваться скорость

MAX\_SPEED = 10; // Максимальная скорость вращения

(Продолжение кода на следующей странице)

(Продолжение кода)

type

TfrmMain = class(TForm)

lblSpeed: TLabel;

lblColor: TLabel;

Image: TImage;

udSpeed: TUpDown;

edtSpeed: TEdit;

ColorBox: TColorBox;

{$REGION ' Обработчики событий '}

procedure FormPaint(Sender: TObject);

procedure FormMouseWheel(Sender: TObject; Shift: TShiftState;

WheelDelta: Integer; MousePos: TPoint; var Handled: Boolean);

procedure FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);

procedure FormKeyUp(Sender: TObject; var Key: Word; Shift: TShiftState);

procedure FormHide(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure ImageMouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

procedure ImageMouseUp(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

procedure FormKeyPress(Sender: TObject; var Key: Char);

procedure udSpeedClick(Sender: TObject; Button: TUDBtnType);

procedure ColorBoxChange(Sender: TObject);

{$ENDREGION}

private

{ Private declarations }

isActive : Boolean; // Вращение происходит только когда isActive = True

speed : Integer; // Скорость вращения

procedure CMDialogKey(var Message: TCMDialogKey);

message CM\_DIALOGKEY;

public

{ Public declarations }

end;

var

frmMain: TfrmMain;

Triangle : TTriangle;

Piramid : TPiramid;

Prism : TPrism;

IMPLEMENTATION

{$R \*.dfm}

{$REGION ' Управление состоянием окна '}

// Начальная инициализация значений переменных

procedure TfrmMain.FormCreate(Sender: TObject);

begin

image.Canvas.Brush.Color := clWhite;

speed := 0;

isActive := True;

Triangle := TTriangle.Create(

TVector3.Create(0, 300), TVector3.Create(200, 300),

TVector3.Create(100, 100), Image

);

Piramid := TPiramid.Create(

TVector3.Create(300, 300), TVector3.Create(500, 300),

TVector3.Create(400, 300, 200), TVector3.Create(400, 100, 100), Image

);

(Продолжение кода на следующей странице)

(Продолжение кода)

Prism := TPrism.Create(

TVector3.Create(500, 300), TVector3.Create(700, 300),

TVector3.Create(600, 300, 200), 200, Image

);

FormPaint(frmMain);

frmMain.Refresh;

end;

{ Прекратить вращение при сворачивании формы для экономии

процессорного времени и заряда батареи устройства пользователя }

procedure TfrmMain.FormHide(Sender: TObject);

begin

isActive := False;

end;

// При разворачивании окна продолжить вращение

procedure TfrmMain.FormShow(Sender: TObject);

begin

isActive := True;

FormPaint(frmMain);

end;

{$ENDREGION}

{$REGION ' Управление клавиатурой '}

// Обработка нажатия символьных клавиш (дискретное управление скоростью)

procedure TfrmMain.FormKeyPress(Sender: TObject; var Key: Char);

begin

if isActive then

begin

// клавиши 0..9 включают положительную 0..9 скорость

case Key of

'0' : speed := 0\*round(MAX\_SPEED/10);

'1' : speed := 1\*round(MAX\_SPEED/10);

'2' : speed := 2\*round(MAX\_SPEED/10);

'3' : speed := 3\*round(MAX\_SPEED/10);

'4' : speed := 4\*round(MAX\_SPEED/10);

'5' : speed := 5\*round(MAX\_SPEED/10);

'6' : speed := 6\*round(MAX\_SPEED/10);

'7' : speed := 7\*round(MAX\_SPEED/10);

'8' : speed := 8\*round(MAX\_SPEED/10);

'9' : speed := 9\*round(MAX\_SPEED/10);

end;

if speed > 0 then

begin

if speed > MAX\_SPEED then

speed := MAX\_SPEED;

end

else

if speed < -MAX\_SPEED then

speed := -MAX\_SPEED;

frmMain.Refresh;

end;

end;

// Хук для перехвата сообщений диалоговых сообщений клавиатуры

procedure TfrmMain.CMDialogKey(var Message: TCMDialogKey);

begin

(Продолжение кода на следующей странице)

(Продолжение кода)

// Перехватываем нажатие клавиши TAB

if Message.CharCode <> VK\_TAB then

inherited;

end;

// Обработка нажатия клавиш клавиатуры

procedure TfrmMain.FormKeyDown(Sender: TObject; var Key: Word;

Shift: TShiftState);

begin

// Без управляющих клавиш

if Shift = [] then

begin

case Key of

// Останавливают вращение клавиши Enter, Esc

// А клавиша Space при нажатии замораживает изображение

VK\_RETURN : isActive := not isActive;

VK\_ESCAPE : isActive := not isActive;

VK\_SPACE : isActive := not isActive;

end;

if isActive then

begin

case Key of

{ Управляют скоростью (увеличивают/уменьшают)

клавиши PageUp/PageDown, Home/End, +/-,

стрелки вверх/вниз, вправо/влево }

VK\_PRIOR : speed := speed + SPEED\_STEP;

VK\_NEXT : speed := speed - SPEED\_STEP;

VK\_HOME : speed := speed + SPEED\_STEP;

VK\_END : speed := speed - SPEED\_STEP;

VK\_ADD : speed := speed + SPEED\_STEP;

VK\_SUBTRACT : speed := speed - SPEED\_STEP;

VK\_RIGHT : speed := speed + SPEED\_STEP;

VK\_LEFT : speed := speed - SPEED\_STEP;

VK\_UP : speed := speed + SPEED\_STEP;

VK\_DOWN : speed := speed - SPEED\_STEP;

// Клавиша TAB разворачивает вращение в противоположную сторону

VK\_TAB : speed := -speed;

{ клавиши 0..9 на NUMPAD-панели

включают положительную 0..9 скорость }

VK\_NUMPAD0 : speed := 0\*round(MAX\_SPEED/10);

VK\_NUMPAD1 : speed := 1\*round(MAX\_SPEED/10);

VK\_NUMPAD2 : speed := 2\*round(MAX\_SPEED/10);

VK\_NUMPAD3 : speed := 3\*round(MAX\_SPEED/10);

VK\_NUMPAD4 : speed := 4\*round(MAX\_SPEED/10);

VK\_NUMPAD5 : speed := 5\*round(MAX\_SPEED/10);

VK\_NUMPAD6 : speed := 6\*round(MAX\_SPEED/10);

VK\_NUMPAD7 : speed := 7\*round(MAX\_SPEED/10);

VK\_NUMPAD8 : speed := 8\*round(MAX\_SPEED/10);

VK\_NUMPAD9 : speed := 9\*round(MAX\_SPEED/10);

end;

end;

if speed > 0 then

begin

if speed > MAX\_SPEED then

speed := MAX\_SPEED;

end

else

if speed < -MAX\_SPEED then

speed := -MAX\_SPEED;

end;

(Продолжение кода на следующей странице)

(Продолжение кода)

FormPaint(frmMain);

frmMain.Refresh;

end;

// Возврат в исходное состояние после отпускания клавиш

procedure TfrmMain.FormKeyUp(Sender: TObject; var Key: Word;

Shift: TShiftState);

begin

// При отпускании Space без управляющих клавиш продолжить вращение

if (Shift = []) and (Key = VK\_SPACE) then

isActive := not isActive;

FormPaint(frmMain);

frmMain.Refresh;

end;

{$ENDREGION}

{$REGION ' Управление мышью '}

// Управление скоростью вращения с помощью колёсика мыши

procedure TfrmMain.FormMouseWheel(Sender: TObject; Shift: TShiftState;

WheelDelta: Integer; MousePos: TPoint; var Handled: Boolean);

begin

if isActive then

begin

if WheelDelta > 0 then

if speed <= MAX\_SPEED - SPEED\_STEP then

speed := speed + SPEED\_STEP

else

if speed >= -MAX\_SPEED + SPEED\_STEP then

speed := speed - SPEED\_STEP;

FormPaint(frmMain);

frmMain.Refresh;

end;

end;

// При зажатии клавиши мыши на изображении заморозить вращение

procedure TfrmMain.ImageMouseDown(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

begin

isActive := False;

end;

// При отпускании клавиши мыши на изображении разморозить вращение

procedure TfrmMain.ImageMouseUp(Sender: TObject; Button: TMouseButton;

Shift: TShiftState; X, Y: Integer);

begin

isActive := True;

FormPaint(frmMain);

frmMain.Refresh;

end;

{$ENDREGION}

{$REGION ' Обработчики событий '}

procedure TfrmMain.udSpeedClick(Sender: TObject; Button: TUDBtnType);

begin

if Button = btNext then

begin

speed := speed + SPEED\_STEP;

if speed > MAX\_SPEED then

speed := MAX\_SPEED;

end

(Продолжение кода на следующей странице)

(Продолжение кода)

else if Button = btPrev then

begin

speed := speed - SPEED\_STEP;

if speed < -MAX\_SPEED then

speed := -MAX\_SPEED;

end;

FormPaint(frmMain);

frmMain.Refresh;

end;

procedure TfrmMain.ColorBoxChange(Sender: TObject);

var pen : TPen;

begin

pen := TPen.Create;

pen.Color := ColorBox.Selected;

Triangle.pen := pen;

Piramid.pen := pen;

Prism.pen := pen;

end;

{$ENDREGION}

// Отрисовка окна

procedure TfrmMain.FormPaint(Sender: TObject);

var Line2 : TLine3;

begin

if isActive then

begin

Triangle.Clear;

Triangle.Rotate(TVector3.Create(100, 200), 0, 0.001\*speed);

Triangle.Draw;

Piramid.Clear;

Piramid.Rotate(TVector3.Create(400, 200, 100), 0, 0.001\*speed);

Piramid.Draw;

Prism.Clear;

Prism.Rotate(TVector3.Create(600, 200, 100), 0, 0.001\*speed);

Prism.Draw;

edtSpeed.Text := IntToStr(speed);

end;

end;

END.

3) Код модуля вектора:

UNIT ExtendedVector;

INTERFACE

{$DEFINE Extended\_Template}

{$INCLUDE Graphics/Templater}

{$REGION ' TVector3 '}

type TVector3 = record

x, y, z : T;

length : T;

class function Create(x : T = 0; y : T = 0; z : T = 0) : TVector3; overload; static;

class function Create(vector : TVector3) : TVector3; overload; static;

class operator Add(a, b : TVector3) : TVector3; overload;

class operator Subtract(a, b : TVector3) : TVector3; overload;

(Продолжение кода на следующей странице)

(Продолжение кода)

class operator Multiply(a, b : TVector3) : T; overload;

class operator LogicalAnd(a, b : TVector3) : TVector3; overload;

class operator Multiply(a : T; b : TVector3) : TVector3; overload;

class operator Multiply(b : TVector3; a : T) : TVector3; overload;

class operator Negative(vector : TVector3) : TVector3; overload;

class operator Equal(a, b : TVector3) : Boolean; overload;

class operator NotEqual(a, b: TVector3) : Boolean; overload;

class operator Positive(a : TVector3) : TVector3;

end;

function normalize(a : TVector3) : TVector3;

{$ENDREGION}

IMPLEMENTATION

{$REGION ' TVector3 '}

// Нормализация длины вектора

function normalize(a : TVector3) : TVector3;

begin

Result := a \* (1/a.length);

end;

{$REGION ' Конструкторы '}

class function TVector3.Create(x : T = 0; y : T = 0; z : T = 0) : TVector3;

var rec : TVector3;

begin

rec.x := x;

rec.y := y;

rec.z := z;

rec.length := sqrt(x\*x + y\*y + z\*z);

Result := rec;

end;

class function TVector3.Create(vector : TVector3) : TVector3;

var rec : TVector3;

begin

rec.x := vector.x;

rec.y := vector.y;

rec.z := vector.z;

rec.length := vector.length;

Result := rec;

end;

{$ENDREGION}

{$REGION ' Арифметические операторы '}

// C := A + B;

class operator TVector3.Add(a, b : TVector3): TVector3;

begin

Result := TVector3.Create(a.x + b.x, a.y + b.y, a.z + b.z);

end;

// C := A - B;

class operator TVector3.Subtract(a, b : TVector3): TVector3;

begin

Result := TVector3.Create(a.x - b.x, a.y - b.y, a.z - b.z);

end;

(Продолжение кода на следующей странице)

(Продолжение кода)

// number := A \* B (скалярное произведение векторов);

class operator TVector3.Multiply(a, b : TVector3): T;

begin

Result := a.x\*b.x + a.y\*b.y + a.z\*b.z;

end;

// C := A and B (векторное произведение векторов);

class operator TVector3.LogicalAnd(a, b : TVector3): TVector3;

begin

Result := TVector3.Create(a.y\*b.z - a.z\*b.y, a.z\*b.x - a.x\*b.z, a.x\*b.y - a.y\*b.x);

end;

// C := 3 \* A;

class operator TVector3.Multiply(a : T; b : TVector3): TVector3;

begin

Result := TVector3.Create(b.x \* a, b.y \* a, b.z \* a);

end;

// C := A \* 3;

class operator TVector3.Multiply(b : TVector3; a : T): TVector3;

begin

Result := TVector3.Create(b.x \* a, b.y \* a, b.z \* a);

end;

// C := -B;

class operator TVector3.Negative(vector : TVector3): TVector3;

begin

Result := TVector3.Create(-vector.x, -vector.y, -vector.z);

end;

// C := +A; (нормализация вектора A)

class operator TVector3.Positive(a : TVector3) : TVector3;

begin

Result := a \* (1/a.length);

end;

{$ENDREGION}

{$REGION ' Операторы сравнения '}

// C = B;

class operator TVector3.Equal(a, b : TVector3): Boolean;

begin

Result := (a.x = b.x) and (a.y = b.y) and (a.z = b.z);

end;

// C <> B;

class operator TVector3.NotEqual(a, b: TVector3): Boolean;

begin

Result := (a.x <> b.x) or (a.y <> b.y) or (a.z <> b.z);

end;

{$ENDREGION}

{$ENDREGION}

END.

4) Код базового графического модуля – абстрактной фигуры:

UNIT BaseFigure;

INTERFACE

uses ExtCtrls, Graphics;

type \_TBaseFigure = class

protected

image : TImage;

\_pen : TPen;

procedure setPen(pen : TPen);

function getPen : TPen;

public

constructor Create(img : TImage; pen\_color : TColor = clBlack);

destructor Destroy; reintroduce;

property pen : TPen read getPen write setPen;

procedure Clear;

procedure Draw; virtual; abstract;

end;

IMPLEMENTATION

constructor \_TBaseFigure.Create(img : TImage;

pen\_color : TColor = clBlack);

begin

image := TImage.Create(img.GetParentComponent);

image := img;

\_pen := TPen.Create;

\_pen.Color := pen\_color;

end;

{$REGION ' Сеттеры и геттеры '}

procedure \_TBaseFigure.setPen(pen : TPen);

begin

\_pen := pen;

end;

function \_TBaseFigure.getPen : TPen;

begin

Result := \_pen;

end;

{$ENDREGION}

// Очистка фигуры цветом фона

procedure \_TBaseFigure.Clear;

var

temp\_pen : TPen;

begin

temp\_pen := TPen.Create;

temp\_pen.Color := \_pen.Color;

\_pen.Color := image.Canvas.Brush.Color;

Draw; {нарисовать фигуру цветом фона - стереть}

\_pen.Color := temp\_pen.Color;

end;

destructor \_TBaseFigure.Destroy;

begin

Clear;

inherited Destroy;

end;

END.

6) Код конкретизации базового графического модуля на тип данных:

UNIT ExtendedFigure;

INTERFACE

uses BaseFigure, ExtendedVector;

{$DEFINE Extended\_Template}

{$INCLUDE Graphics/Templater}

type TFigure = class (\_TBaseFigure)

public

procedure Rotate(center : TVector3; alpha, beta, gamma : T); virtual; abstract;

procedure Move(dx, dy, dz : T); virtual; abstract;

procedure Scale(center : TVector3; x, y, z : T); virtual; abstract;

end;

IMPLEMENTATION

END.

6) Код графического модуля – «Треугольник»:

UNIT ExtendedTriangle;

INTERFACE

uses SysUtils, ExtCtrls, Graphics, ExtendedFigure, ExtendedVector, ExtendedLine;

{$DEFINE Extended\_Template}

{$INCLUDE Graphics/Templater}

const EPS = 3.4e-4932; // Допуск погрешности

type TTriangle = class (TFigure)

private

\_a, \_b, \_c : TLine3;

function getNormal : TVector3;

function getPerimetr : T;

function getSquare : T;

function getA : TVector3;

function getB : TVector3;

function getC : TVector3;

public

property normal : TVector3 read getNormal;

property Square : T read getSquare;

property Perimetr : T read getPerimetr;

property A : TVector3 read getA;

property B : TVector3 read getB;

property C : TVector3 read getC;

property AB : TLine3 read \_a;

property BC : TLine3 read \_b;

property AC : TLine3 read \_c;

constructor Create(A, B, C : TVector3; img : TImage;

pen\_color : TColor = clBlack); overload;

constructor Create(AB : TLine3; C : TVector3; img : TImage;

pen\_color : TColor = clBlack); overload;

constructor Create(ABC : TTriangle); overload;

{ TODO: Constructors for true and ortogonal triangle }

procedure Draw; override;

(Продолжение кода на следующей странице)

(Продолжение кода)

procedure Rotate(center : TVector3; alpha : T = 0;

beta : T = 0; gamma : T = 0); reintroduce; overload;

procedure Rotate(alpha : T = 0; beta : T = 0; gamma : T = 0); reintroduce; overload;

procedure Move(dx : T = 0; dy : T = 0; dz : T = 0); reintroduce; overload;

procedure Move(delta : TVector3); reintroduce; overload;

procedure Scale(center : TVector3; x : T = 1; y : T = 1; z : T = 1); reintroduce; overload;

procedure Scale(center : TVector3; scale : TVector3); reintroduce; overload;

procedure Scale(x : T = 1; y : T = 1; z : T = 1); reintroduce; overload;

end;

IMPLEMENTATION

{$REGION ' Конструкторы '}

constructor TTriangle.Create(A, B, C : TVector3; img : TImage;

pen\_color : TColor = clBlack);

begin

inherited Create(img, pen\_color);

\_a := TLine3.FromAtoB(A, B, img, pen\_color);

\_b := TLine3.FromAtoB(B, C, img, pen\_color);

\_c := TLine3.FromAtoB(C, A, img, pen\_color);

end;

constructor TTriangle.Create(AB : TLine3; C : TVector3; img : TImage;

pen\_color : TColor = clBlack);

begin

inherited Create(img, pen\_color);

\_a := AB;

\_b := TLine3.FromAtoB(AB.getB, C, img, pen\_color);

\_c := TLine3.FromAtoB(C, AB.getA, img, pen\_color);

end;

constructor TTriangle.Create(ABC : TTriangle);

begin

inherited Create(ABC.image, ABC.getPen.Color);

\_a := ABC.AB;

\_b := ABC.BC;

\_c := ABC.AC;

end;

{$ENDREGION}

{$REGION ' Функции доступа к свойствам '}

function TTriangle.getA() : TVector3;

begin

Result := \_a.getA;

end;

function TTriangle.getB() : TVector3;

begin

Result := \_a.getB;

end;

function TTriangle.getC() : TVector3;

begin

Result := \_b.getB;

end;

function TTriangle.getNormal() : TVector3;

var ort : TVector3;

begin

ort := (\_a.vector and \_b.vector);

(Продолжение кода на следующей странице)

(Продолжение кода)

Result := +ort;

end;

function TTriangle.getPerimetr() : T;

begin

Result := \_a.length + \_b.length + \_c.length;

end;

function TTriangle.getSquare() : T;

var P : T;

begin

p := getPerimetr/2;

Result := sqrt((p - \_a.length)\*(p - \_b.length)\*(p - \_c.length)\*p);

end;

{$ENDREGION}

{$REGION ' Функции преобразования '}

procedure TTriangle.Move(dx : T = 0; dy : T = 0; dz : T = 0);

begin

\_a.Move(dx, dy, dz);

\_b.Move(dx, dy, dz);

\_c.Move(dx, dy, dz);

end;

procedure TTriangle.Move(delta : TVector3);

begin

\_c.Move(delta);

\_b.Move(delta);

\_c.Move(delta);

end;

procedure TTriangle.Rotate(center : TVector3; alpha : T = 0; beta : T = 0; gamma : T = 0);

begin

\_a.Rotate(center, alpha, beta, gamma);

\_b.Rotate(center, alpha, beta, gamma);

\_c.Rotate(center, alpha, beta, gamma);

end;

procedure TTriangle.Rotate(alpha : T = 0; beta : T = 0; gamma : T = 0);

begin

\_a.Rotate(alpha, beta, gamma);

\_b.Rotate(alpha, beta, gamma);

\_c.Rotate(alpha, beta, gamma);

end;

procedure TTriangle.Scale(center : TVector3; x : T = 1; y : T = 1; z : T = 1);

begin

\_a.Scale(center, x, y, z);

\_b.Scale(center, x, y, z);

\_c.Scale(center, x, y, z);

end;

procedure TTriangle.Scale(center : TVector3; scale : TVector3);

begin

\_a.Scale(center, scale);

\_b.Scale(center, scale);

\_c.Scale(center, scale);

end;

(Продолжение кода на следующей странице)

(Продолжение кода)

procedure TTriangle.Scale(x : T = 1; y : T = 1; z : T = 1);

begin

\_a.Scale(x, y, z);

\_b.Scale(x, y, z);

\_c.Scale(x, y, z);

end;

{$ENDREGION}

procedure TTriangle.Draw;

var

center : TVector3;

temp\_pen : TPen;

begin

temp\_pen := TPen.Create;

temp\_pen.Color := image.Canvas.Pen.Color;

\_a.pen.Color := \_pen.Color;

\_b.pen.Color := \_pen.Color;

\_c.pen.Color := \_pen.Color;

\_a.Draw;

\_b.Draw;

\_c.Draw;

image.Canvas.Pen.Color := \_pen.Color;

end;

END.

7) Код графического модуля – «Призма»:

UNIT ExtendedPrism;

INTERFACE

uses SysUtils, ExtCtrls, Graphics,

ExtendedFigure, ExtendedVector, ExtendedLine, ExtendedTriangle;

{$DEFINE Extended\_Template}

{$INCLUDE Graphics/Templater}

type TPrism = class (TFigure)

private

\_AB, \_BC, \_AC,

\_AA1, \_BB1, \_CC1,

\_A1B1, \_B1C1, \_A1C1 : TLine3;

function getPerimetr() : T;

function getSquare() : T;

function getVolume() : T;

function getA : TVector3;

function getB : TVector3;

function getC : TVector3;

function getA1 : TVector3;

function getB1 : TVector3;

function getC1 : TVector3;

function getABC : TTriangle;

function getA1B1C1 : TTriangle;

public

property Volume : T read getVolume;

property Square : T read getSquare;

property Perimetr : T read getPerimetr;

property A : TVector3 read getA;

(Продолжение кода на следующей странице)

(Продолжение кода)

property B : TVector3 read getB;

property C : TVector3 read getC;

property A1 : TVector3 read getA1;

property B1 : TVector3 read getB1;

property C1 : TVector3 read getC1;

property A\_B : TLine3 read \_AB;

property B\_C : TLine3 read \_BC;

property A\_C : TLine3 read \_AC;

property A\_A1 : TLine3 read \_AA1;

property B\_B1 : TLine3 read \_BB1;

property C\_C1 : TLine3 read \_CC1;

property A1\_B1 : TLine3 read \_A1B1;

property B1\_C1 : TLine3 read \_B1C1;

property A1\_C1 : TLine3 read \_A1C1;

property ABC : TTriangle read getABC;

property A1B1C1 : TTriangle read getA1B1C1;

{ TODO: get side areas (AA1BB1)}

constructor Create(ABC : TTriangle; A1 : TVector3; img : TImage;

pen\_color : TColor = clBlack); overload;

constructor Create(ABC : TTriangle; height : T; img : TImage;

pen\_color : TColor = clBlack); overload;

constructor Create(A, B, C : TVector3; height : T; img : TImage;

pen\_color : TColor = clBlack); overload;

constructor Create(A, B, C, A1 : TVector3; img : TImage;

pen\_color : TColor = clBlack); overload;

constructor Create(ABCA1B1C1 : TPrism); overload;

procedure Draw; override;

procedure Rotate(center : TVector3; alpha : T = 0;

beta : T = 0; gamma : T = 0); reintroduce; overload;

procedure Rotate(alpha : T = 0; beta : T = 0; gamma : T = 0); reintroduce; overload;

procedure Move(dx : T = 0; dy : T = 0; dz : T = 0); reintroduce; overload;

procedure Move(delta : TVector3); reintroduce; overload;

procedure Scale(center : TVector3; x : T = 1; y : T = 1; z : T = 1); reintroduce; overload;

procedure Scale(center : TVector3; scale : TVector3); reintroduce; overload;

procedure Scale(x : T = 1; y : T = 1; z : T = 1); reintroduce; overload;

end;

IMPLEMENTATION

{$REGION ' Конструкторы '}

constructor TPrism.Create(ABC : TTriangle; A1 : TVector3; img : TImage;

pen\_color : TColor = clBlack);

begin

inherited Create(img, pen\_color);

\_AB := TLine3.Create(ABC.AB);

\_AC := TLine3.Create(ABC.AC);

\_BC := TLine3.Create(ABC.BC);

\_AA1 := TLine3.FromAtoB(ABC.A, A1, img, pen\_color);

\_BB1 := TLine3.FromAtoB(ABC.B, TVector3.Create(ABC.B.x, A1.y, ABC.B.z),

img, pen\_color);

\_CC1 := TLine3.FromAtoB(ABC.C, TVector3.Create(ABC.C.x, A1.y, ABC.C.z),

img, pen\_color);

\_A1B1 := TLine3.FromAtoB(A1, TVector3.Create(ABC.B.x, A1.y, ABC.B.z), img, pen\_color);

\_A1C1 := TLine3.FromAtoB(A1, TVector3.Create(ABC.C.x, A1.y, ABC.C.z), img, pen\_color);

(Продолжение кода на следующей странице)

(Продолжение кода)

\_B1C1 := TLine3.FromAtoB(TVector3.Create(ABC.B.x, A1.y, ABC.B.z),

TVector3.Create(ABC.C.x, A1.y, ABC.C.z), img, pen\_color);

end;

constructor TPrism.Create(ABC : TTriangle; height : T; img : TImage;

pen\_color : TColor = clBlack);

begin

inherited Create(img, pen\_color);

\_AB := TLine3.Create(ABC.AB);

\_AC := TLine3.Create(ABC.AC);

\_BC := TLine3.Create(ABC.BC);

\_AA1 := TLine3.FromAtoB(ABC.A, TVector3.Create(ABC.A.x, ABC.A.y - height, ABC.A.z),

img, pen\_color);

\_BB1 := TLine3.FromAtoB(ABC.B, TVector3.Create(ABC.B.x, ABC.B.y - height, ABC.B.z),

img, pen\_color);

\_CC1 := TLine3.FromAtoB(ABC.C, TVector3.Create(ABC.C.x, ABC.C.y - height, ABC.C.z),

img, pen\_color);

\_A1B1 := TLine3.FromAtoB(TVector3.Create(ABC.A.x, ABC.A.y - height, ABC.A.z),

TVector3.Create(ABC.B.x, ABC.B.y - height, ABC.B.z), img, pen\_color);

\_A1C1 := TLine3.FromAtoB(TVector3.Create(ABC.A.x, ABC.A.y - height, ABC.A.z),

TVector3.Create(ABC.C.x, ABC.C.y - height, ABC.C.z), img, pen\_color);

\_B1C1 := TLine3.FromAtoB(TVector3.Create(ABC.B.x, ABC.B.y - height, ABC.B.z),

TVector3.Create(ABC.C.x, ABC.C.y - height, ABC.C.z), img, pen\_color);

end;

constructor TPrism.Create(A, B, C : TVector3; height : T; img : TImage;

pen\_color : TColor = clBlack);

begin

inherited Create(img, pen\_color);

\_AB := TLine3.FromAtoB(A, B, img, pen\_color);

\_AC := TLine3.FromAtoB(A, C, img, pen\_color);

\_BC := TLine3.FromAtoB(B, C, img, pen\_color);

\_AA1 := TLine3.FromAtoB(A, TVector3.Create(A.x, A.y - height, A.z), img, pen\_color);

\_BB1 := TLine3.FromAtoB(B, TVector3.Create(B.x, B.y - height, B.z), img, pen\_color);

\_CC1 := TLine3.FromAtoB(C, TVector3.Create(C.x, C.y - height, C.z), img, pen\_color);

\_A1B1 := TLine3.FromAtoB(TVector3.Create(A.x, A.y - height, A.z),

TVector3.Create(B.x, B.y - height, B.z), img, pen\_color);

\_A1C1 := TLine3.FromAtoB(TVector3.Create(A.x, A.y - height, A.z),

TVector3.Create(C.x, C.y - height, C.z), img, pen\_color);

\_B1C1 := TLine3.FromAtoB(TVector3.Create(B.x, B.y - height, B.z),

TVector3.Create(C.x, C.y - height, C.z), img, pen\_color);

end;

constructor TPrism.Create(A, B, C, A1 : TVector3; img : TImage;

pen\_color : TColor = clBlack);

begin

inherited Create(img, pen\_color);

\_AB := TLine3.FromAtoB(A, B, img, pen\_color);

\_AC := TLine3.FromAtoB(A, C, img, pen\_color);

\_BC := TLine3.FromAtoB(B, C, img, pen\_color);

\_AA1 := TLine3.FromAtoB(A, A1, img, pen\_color);

\_BB1 := TLine3.FromAtoB(B, TVector3.Create(B.x, A1.y, B.z), img, pen\_color);

\_CC1 := TLine3.FromAtoB(C, TVector3.Create(C.x, A1.y, C.z), img, pen\_color);

\_A1B1 := TLine3.FromAtoB(A1, TVector3.Create(B.x, A1.y, B.z), img, pen\_color);

\_A1C1 := TLine3.FromAtoB(A, TVector3.Create(C.x, A1.y, C.z), img, pen\_color);

\_B1C1 := TLine3.FromAtoB(TVector3.Create(B.x, A1.y, B.z),

TVector3.Create(C.x, A1.y, C.z), img, pen\_color);

end;

(Продолжение кода на следующей странице)

(Продолжение кода)

constructor TPrism.Create(ABCA1B1C1 : TPrism);

begin

inherited Create(image, \_pen.Color);

\_AB := TLine3.Create(ABCA1B1C1.A\_B);

\_BC := TLine3.Create(ABCA1B1C1.B\_C);

\_AC := TLine3.Create(ABCA1B1C1.A\_C);

\_AA1 := TLine3.Create(ABCA1B1C1.A\_A1);

\_BB1 := TLine3.Create(ABCA1B1C1.B\_B1);

\_CC1 := TLine3.Create(ABCA1B1C1.C\_C1);

\_A1B1 := TLine3.Create(ABCA1B1C1.A1\_B1);

\_B1C1 := TLine3.Create(ABCA1B1C1.B1\_C1);

\_A1C1 := TLine3.Create(ABCA1B1C1.A1\_C1);

end;

{$ENDREGION}

{$REGION ' Функции доступа к свойствам '}

function TPrism.getPerimetr() : T;

begin

Result := \_AB.length + \_BC.length + \_AC.length +

\_A1B1.length + \_B1C1.length + \_A1C1.length +

\_AA1.length + \_BB1.length + \_CC1.length;

end;

function TPrism.getSquare() : T;

var ABC, A1B1C1 : TTriangle;

begin

ABC := TTriangle.Create(\_AB.getA, \_AB.getB, \_BC.getB, image, \_pen.Color);

A1B1C1 := TTriangle.Create(\_A1B1.getA, \_A1B1.getB, \_B1C1.getB, image, \_pen.Color);

Result := ABC.Square + A1B1C1.Square + \_AA1.length \* ABC.Perimetr;

end;

function TPrism.getVolume() : T;

var ABC : TTriangle;

begin

{ TODO: Only for ortogonal prism }

ABC := TTriangle.Create(\_AB.getA, \_AB.getB, \_AC.getB, image, \_pen.Color);

Result := ABC.Square \* \_AA1.length;

end;

function TPrism.getA : TVector3;

begin

Result := \_AB.getA;

end;

function TPrism.getB : TVector3;

begin

Result := \_AB.getB;

end;

function TPrism.getC : TVector3;

begin

Result := \_AC.getB;

end;

function TPrism.getA1 : TVector3;

begin

Result := \_A1C1.getA;

end;

(Продолжение кода на следующей странице)

(Продолжение кода)

function TPrism.getB1 : TVector3;

begin

Result := \_A1B1.getB;

end;

function TPrism.getC1 : TVector3;

begin

Result := \_A1C1.getB;

end;

function TPrism.getABC : TTriangle;

begin

Result := TTriangle.Create(\_AB.getA, \_AB.getB, \_BC.getB, image, \_pen.Color);

end;

function TPrism.getA1B1C1 : TTriangle;

begin

Result := TTriangle.Create(\_A1B1.getA, \_A1B1.getB, \_B1C1.getB, image, \_pen.Color);

end;

{$ENDREGION}

{$REGION ' Функции преобразования '}

procedure TPrism.Rotate(center : TVector3; alpha : T = 0;

beta : T = 0; gamma : T = 0);

begin

\_AB.Rotate(center, alpha, beta, gamma);

\_AC.Rotate(center, alpha, beta, gamma);

\_BC.Rotate(center, alpha, beta, gamma);

\_AA1.Rotate(center, alpha, beta, gamma);

\_BB1.Rotate(center, alpha, beta, gamma);

\_CC1.Rotate(center, alpha, beta, gamma);

\_A1B1.Rotate(center, alpha, beta, gamma);

\_A1C1.Rotate(center, alpha, beta, gamma);

\_B1C1.Rotate(center, alpha, beta, gamma);

end;

procedure TPrism.Rotate(alpha : T = 0; beta : T = 0; gamma : T = 0);

begin

\_AB.Rotate(alpha, beta, gamma);

\_AC.Rotate(alpha, beta, gamma);

\_BC.Rotate(alpha, beta, gamma);

\_AA1.Rotate(alpha, beta, gamma);

\_BB1.Rotate(alpha, beta, gamma);

\_CC1.Rotate(alpha, beta, gamma);

\_A1B1.Rotate(alpha, beta, gamma);

\_A1C1.Rotate(alpha, beta, gamma);

\_B1C1.Rotate(alpha, beta, gamma);

end;

procedure TPrism.Move(dx : T = 0; dy : T = 0; dz : T = 0);

begin

\_AB.Move(dx, dy, dz);

\_AC.Move(dx, dy, dz);

\_BC.Move(dx, dy, dz);

\_AA1.Move(dx, dy, dz);

\_BB1.Move(dx, dy, dz);

\_CC1.Move(dx, dy, dz);

\_A1B1.Move(dx, dy, dz);

\_A1C1.Move(dx, dy, dz);

\_B1C1.Move(dx, dy, dz);

end;

(Продолжение кода на следующей странице)

(Продолжение кода)

procedure TPrism.Move(delta : TVector3);

begin

\_AB.Move(delta);

\_AC.Move(delta);

\_BC.Move(delta);

\_AA1.Move(delta);

\_BB1.Move(delta);

\_CC1.Move(delta);

\_A1B1.Move(delta);

\_A1C1.Move(delta);

\_B1C1.Move(delta);

end;

procedure TPrism.Scale(center : TVector3; x : T = 1; y : T = 1; z : T = 1);

begin

\_AB.Scale(center, x, y, z);

\_AC.Scale(center, x, y, z);

\_BC.Scale(center, x, y, z);

\_AA1.Scale(center, x, y, z);

\_BB1.Scale(center, x, y, z);

\_CC1.Scale(center, x, y, z);

\_A1B1.Scale(center, x, y, z);

\_A1C1.Scale(center, x, y, z);

\_B1C1.Scale(center, x, y, z);

end;

procedure TPrism.Scale(center : TVector3; scale : TVector3);

begin

\_AB.Scale(center, scale);

\_AC.Scale(center, scale);

\_BC.Scale(center, scale);

\_AA1.Scale(center, scale);

\_BB1.Scale(center, scale);

\_CC1.Scale(center, scale);

\_A1B1.Scale(center, scale);

\_A1C1.Scale(center, scale);

\_B1C1.Scale(center, scale);

end;

procedure TPrism.Scale(x : T = 1; y : T = 1; z : T = 1);

begin

\_AB.Scale(x, y, z);

\_AC.Scale(x, y, z);

\_BC.Scale(x, y, z);

\_AA1.Scale(x, y, z);

\_BB1.Scale(x, y, z);

\_CC1.Scale(x, y, z);

\_A1B1.Scale(x, y, z);

\_A1C1.Scale(x, y, z);

\_B1C1.Scale(x, y, z);

end;

{$ENDREGION}

procedure TPrism.Draw;

var

center : TVector3;

temp\_pen : TPen;

begin

temp\_pen := TPen.Create;

temp\_pen.Color := image.Canvas.Pen.Color;

(Продолжение кода на следующей странице)

(Продолжение кода)

image.Canvas.Pen.Color := \_pen.Color;

\_AB.pen.Color := \_pen.Color;

\_AC.pen.Color := \_pen.Color;

\_BC.pen.Color := \_pen.Color;

\_AA1.pen.Color := \_pen.Color;

\_BB1.pen.Color := \_pen.Color;

\_CC1.pen.Color := \_pen.Color;

\_A1B1.pen.Color := \_pen.Color;

\_A1C1.pen.Color := \_pen.Color;

\_B1C1.pen.Color := \_pen.Color;

\_AB.Draw;

\_AC.Draw;

\_BC.Draw;

\_AA1.Draw;

\_BB1.Draw;

\_CC1.Draw;

\_A1B1.Draw;

\_A1C1.Draw;

\_B1C1.Draw;

image.Canvas.Pen.Color := temp\_pen.Color;

end;

END.

8) Код графического модуля – «Пирамида»:

UNIT ExtendedPiramid;

INTERFACE

uses SysUtils, ExtCtrls, Graphics,

ExtendedFigure, ExtendedVector, ExtendedLine, ExtendedTriangle;

{$DEFINE Extended\_Template}

{$INCLUDE Graphics/Templater}

type TPiramid = class (TFigure)

private

\_AB, \_BC, \_AC, \_AS, \_BS, \_CS : TLine3;

function getPerimetr() : T;

function getSquare() : T;

function getVolume() : T;

function getA : TVector3;

function getB : TVector3;

function getC : TVector3;

function getS : TVector3;

function getABC : TTriangle;

function getBCS : TTriangle;

function getACS : TTriangle;

function getABS : TTriangle;

public

property Volume : T read getVolume;

property Square : T read getSquare;

property Perimetr : T read getPerimetr;

property A : TVector3 read getA;

property B : TVector3 read getB;

property C : TVector3 read getC;

property S : TVector3 read getS;

property A\_B : TLine3 read \_AB;

(Продолжение кода на следующей странице)

(Продолжение кода)

property B\_C : TLine3 read \_BC;

property A\_C : TLine3 read \_AC;

property A\_S : TLine3 read \_AS;

property B\_S : TLine3 read \_BC;

property C\_S : TLine3 read \_CS;

property ABC : TTriangle read getABC;

property ABS : TTriangle read getABS;

property BCS : TTriangle read getBCS;

property ACS : TTriangle read getACS;

constructor Create(ABC : TTriangle; S : TVector3; img : TImage;

pen\_color : TColor = clBlack); overload;

constructor Create(A, B, C, S : TVector3; img : TImage;

pen\_color : TColor = clBlack); overload;

constructor Create(ABCS : TPiramid); overload;

{ TODO: Constructors for true and simmetrical piramid }

procedure Draw; override;

procedure Rotate(center : TVector3; alpha : T = 0;

beta : T = 0; gamma : T = 0); reintroduce; overload;

procedure Rotate(alpha : T = 0; beta : T = 0; gamma : T = 0); reintroduce; overload;

procedure Move(dx : T = 0; dy : T = 0; dz : T = 0); reintroduce; overload;

procedure Move(delta : TVector3); reintroduce; overload;

procedure Scale(center : TVector3; x : T = 1; y : T = 1; z : T = 1); reintroduce; overload;

procedure Scale(center : TVector3; scale : TVector3); reintroduce; overload;

procedure Scale(x : T = 1; y : T = 1; z : T = 1); reintroduce; overload;

end;

IMPLEMENTATION

{$REGION ' Конструкторы '}

constructor TPiramid.Create(ABC : TTriangle; S : TVector3; img : TImage;

pen\_color : TColor = clBlack);

begin

inherited Create(img, pen\_color);

\_AB := TLine3.Create(ABC.AB);

\_BC := TLine3.Create(ABC.BC);

\_AC := TLine3.Create(ABC.AC);

\_AS := TLine3.FromAtoB(ABC.A, S, image, pen\_color);

\_BS := TLine3.FromAtoB(ABC.B, S, image, pen\_color);

\_CS := TLine3.FromAtoB(ABC.C, S, image, pen\_color);

end;

constructor TPiramid.Create(A, B, C, S : TVector3; img : TImage;

pen\_color : TColor = clBlack);

begin

inherited Create(img, pen\_color);

\_AB := TLine3.FromAtoB(A, B, image, pen\_color);

\_BC := TLine3.FromAtoB(B, C, image, pen\_color);

\_AC := TLine3.FromAtoB(A, C, image, pen\_color);

\_AS := TLine3.FromAtoB(A, S, image, pen\_color);

\_BS := TLine3.FromAtoB(B, S, image, pen\_color);

\_CS := TLine3.FromAtoB(C, S, image, pen\_color);

end;

constructor TPiramid.Create(ABCS : TPiramid);

begin

inherited Create(ABCS.image, ABCS.getPen.Color);

\_AB := TLine3.Create(ABCS.A\_B);

(Продолжение кода на следующей странице)

(Продолжение кода)

\_BC := TLine3.Create(ABCS.B\_C);

\_AC := TLine3.Create(ABCS.A\_C);

\_AS := TLine3.Create(ABCS.A\_S);

\_BS := TLine3.Create(ABCS.B\_S);

\_CS := TLine3.Create(ABCS.C\_S);

end;

{$ENDREGION}

{$REGION ' Функции доступа к свойствам '}

function TPiramid.getPerimetr() : T;

begin

Result := \_AB.length + \_BC.length + \_AC.length +

\_AS.length + \_BS.length + \_CS.length;

end;

function TPiramid.getSquare() : T;

var ABC, ABS, BCS, ACS : TTriangle;

begin

ABC := TTriangle.Create(\_AB.getA, \_AB.getB, \_BC.getB, image, \_pen.Color);

ABS := TTriangle.Create(\_AB.getA, \_AB.getB, \_BS.getB, image, \_pen.Color);

BCS := TTriangle.Create(\_BC.getA, \_BC.getB, \_CS.getB, image, \_pen.Color);

ACS := TTriangle.Create(\_AC.getA, \_AC.getB, \_CS.getB, image, \_pen.Color);

Result := ABC.Square + ABS.Square + ACS.Square + BCS.Square;

end;

function TPiramid.getVolume() : T;

begin

{ TODO: Not Implemented Yet }

end;

function TPiramid.getA : TVector3;

begin

Result := \_AB.getA;

end;

function TPiramid.getB : TVector3;

begin

Result := \_AB.getB;

end;

function TPiramid.getC : TVector3;

begin

Result := \_AC.getB;

end;

function TPiramid.getS : TVector3;

begin

Result := \_AS.getB;

end;

function TPiramid.getABC : TTriangle;

begin

Result := TTriangle.Create(A\_B.getA, A\_B.getB, B\_C.getB, image, \_pen.Color);

end;

function TPiramid.getBCS : TTriangle;

begin

Result := TTriangle.Create(\_BC.getA, \_BC.getB, \_CS.getB, image, \_pen.Color);

end;

(Продолжение кода на следующей странице)

(Продолжение кода)

function TPiramid.getACS : TTriangle;

begin

Result := TTriangle.Create(\_AC.getA, \_AC.getB, \_CS.getB, image, \_pen.Color);

end;

function TPiramid.getABS : TTriangle;

begin

Result := TTriangle.Create(\_AB.getA, \_AB.getB, \_BS.getB, image, \_pen.Color);

end;

{$ENDREGION}

{$REGION ' Функции преобразования '}

procedure TPiramid.Rotate(center : TVector3; alpha : T = 0;

beta : T = 0; gamma : T = 0);

begin

\_AB.Rotate(center, alpha, beta, gamma);

\_AC.Rotate(center, alpha, beta, gamma);

\_AS.Rotate(center, alpha, beta, gamma);

\_BC.Rotate(center, alpha, beta, gamma);

\_BS.Rotate(center, alpha, beta, gamma);

\_CS.Rotate(center, alpha, beta, gamma);

end;

procedure TPiramid.Rotate(alpha : T = 0; beta : T = 0; gamma : T = 0);

begin

\_AB.Rotate(alpha, beta, gamma);

\_AC.Rotate(alpha, beta, gamma);

\_AS.Rotate(alpha, beta, gamma);

\_BC.Rotate(alpha, beta, gamma);

\_BS.Rotate(alpha, beta, gamma);

\_CS.Rotate(alpha, beta, gamma);

end;

procedure TPiramid.Move(dx : T = 0; dy : T = 0; dz : T = 0);

begin

\_AB.Move(dx, dy, dz);

\_AC.Move(dx, dy, dz);

\_AS.Move(dx, dy, dz);

\_BC.Move(dx, dy, dz);

\_BS.Move(dx, dy, dz);

\_CS.Move(dx, dy, dz);

end;

procedure TPiramid.Move(delta : TVector3);

begin

\_AB.Move(delta);

\_AC.Move(delta);

\_AS.Move(delta);

\_BC.Move(delta);

\_BS.Move(delta);

\_CS.Move(delta);

end;

procedure TPiramid.Scale(center : TVector3; x : T = 1; y : T = 1; z : T = 1);

begin

\_AB.Scale(center, x, y, z);

\_AC.Scale(center, x, y, z);

\_AS.Scale(center, x, y, z);

\_BC.Scale(center, x, y, z);

\_BS.Scale(center, x, y, z);

(Продолжение кода на следующей странице)

(Продолжение кода)

\_CS.Scale(center, x, y, z);

end;

procedure TPiramid.Scale(center : TVector3; scale : TVector3);

begin

\_AB.Scale(center, scale);

\_AC.Scale(center, scale);

\_AS.Scale(center, scale);

\_BC.Scale(center, scale);

\_BS.Scale(center, scale);

\_CS.Scale(center, scale);

end;

procedure TPiramid.Scale(x : T = 1; y : T = 1; z : T = 1);

begin

\_AB.Scale(x, y, z);

\_AC.Scale(x, y, z);

\_AS.Scale(x, y, z);

\_BC.Scale(x, y, z);

\_BS.Scale(x, y, z);

\_CS.Scale(x, y, z);

end;

{$ENDREGION}

procedure TPiramid.Draw;

var

center : TVector3;

temp\_pen : TPen;

begin

temp\_pen := TPen.Create;

temp\_pen.Color := image.Canvas.Pen.Color;

image.Canvas.Pen.Color := \_pen.Color;

\_AB.pen.Color := \_pen.Color;

\_AC.pen.Color := \_pen.Color;

\_AS.pen.Color := \_pen.Color;

\_BC.pen.Color := \_pen.Color;

\_BS.pen.Color := \_pen.Color;

\_CS.pen.Color := \_pen.Color;

\_AB.Draw;

\_AC.Draw;

\_AS.Draw;

\_BC.Draw;

\_BS.Draw;

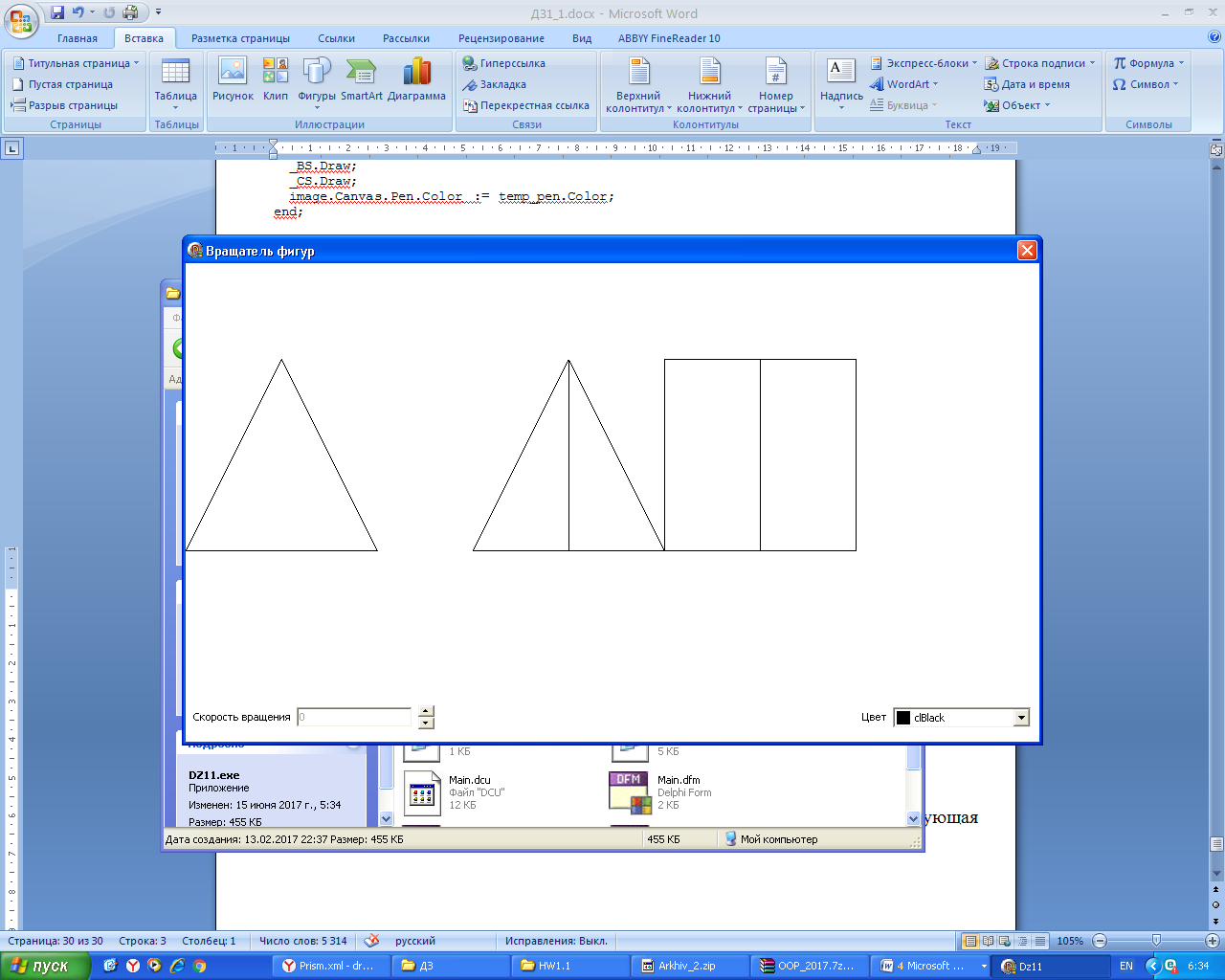
\_CS.Draw;

image.Canvas.Pen.Color := temp\_pen.Color;

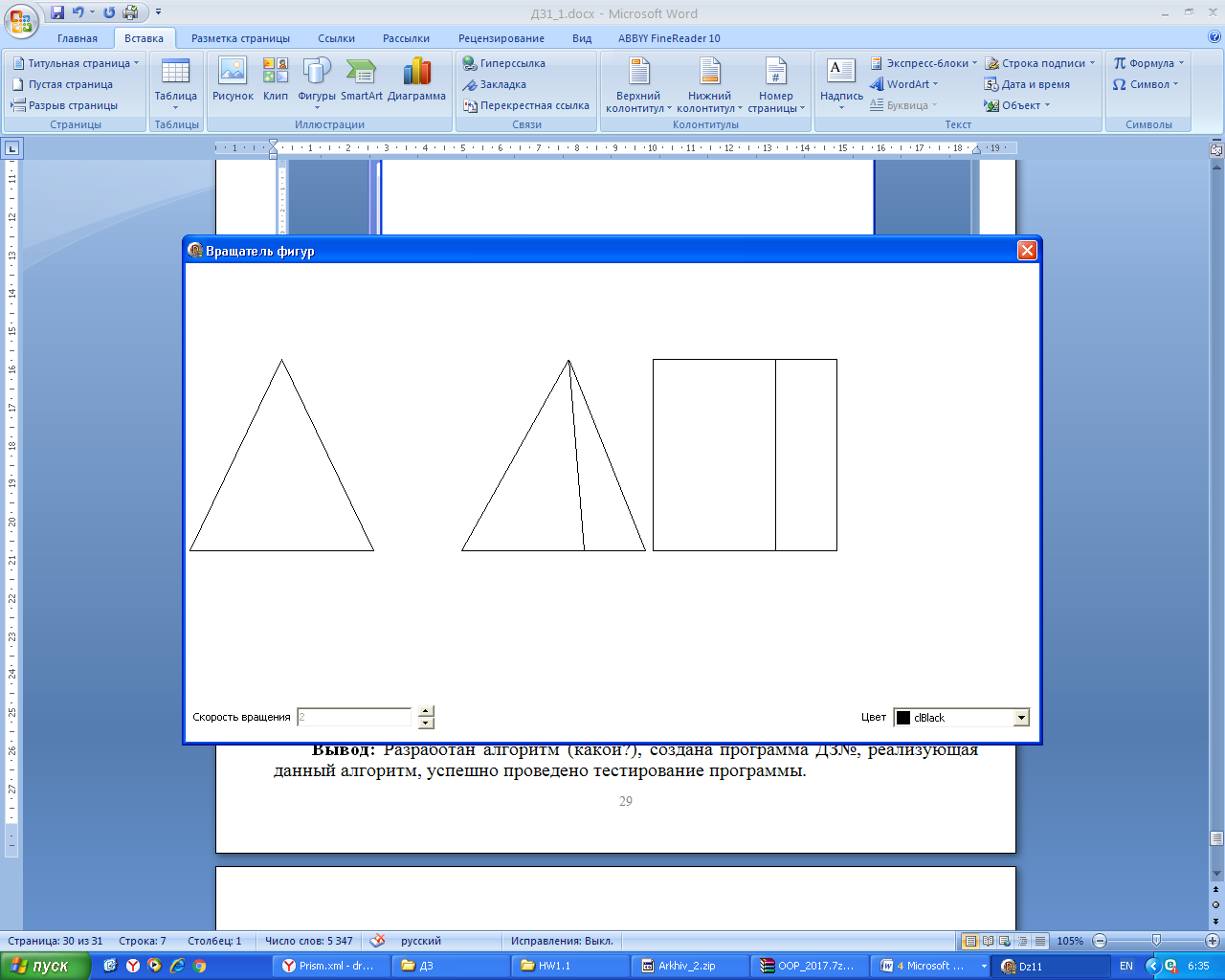
end;

END.

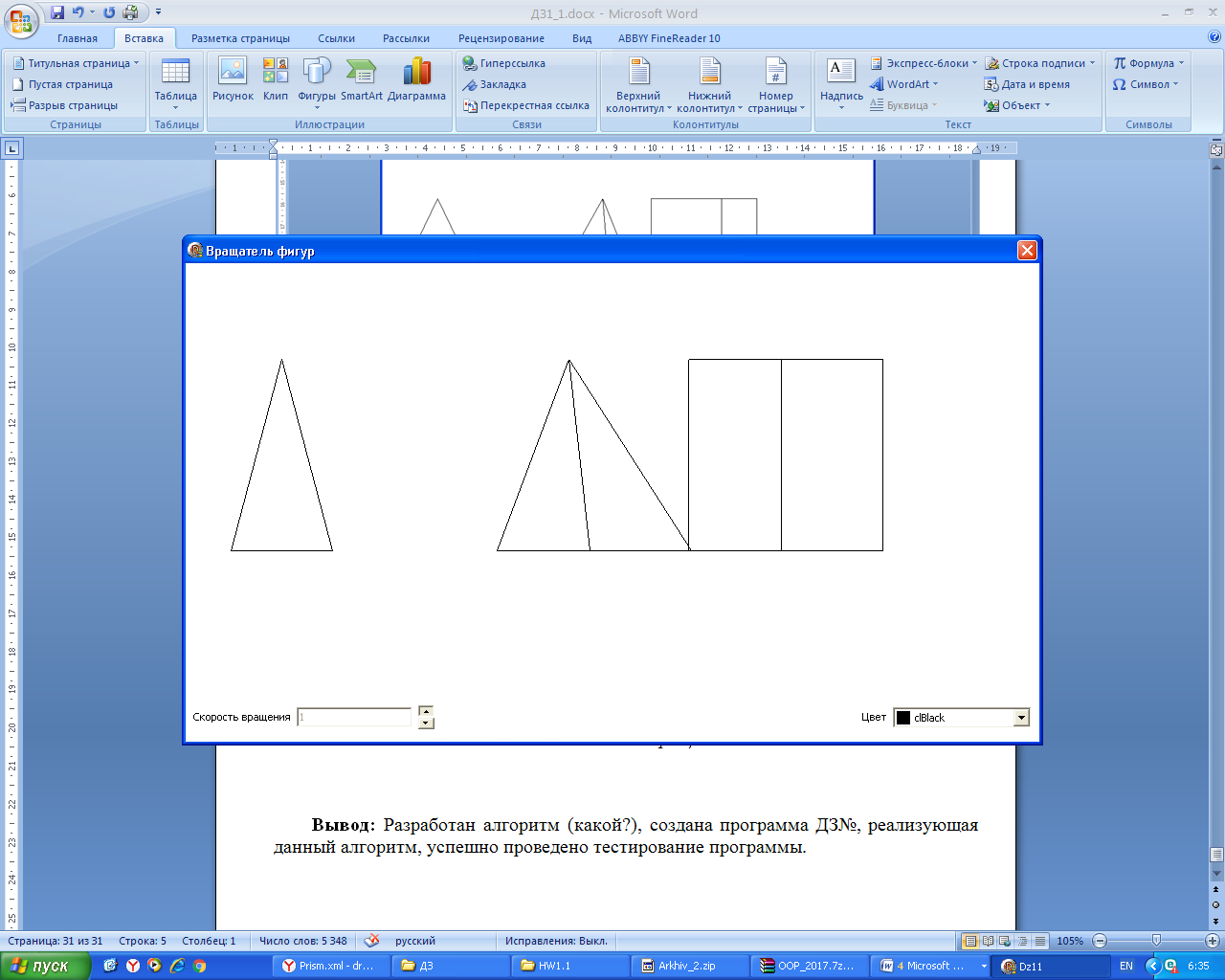
**Скриншоты, демонстрирующие работу программы «Вращатель фигур»**

****

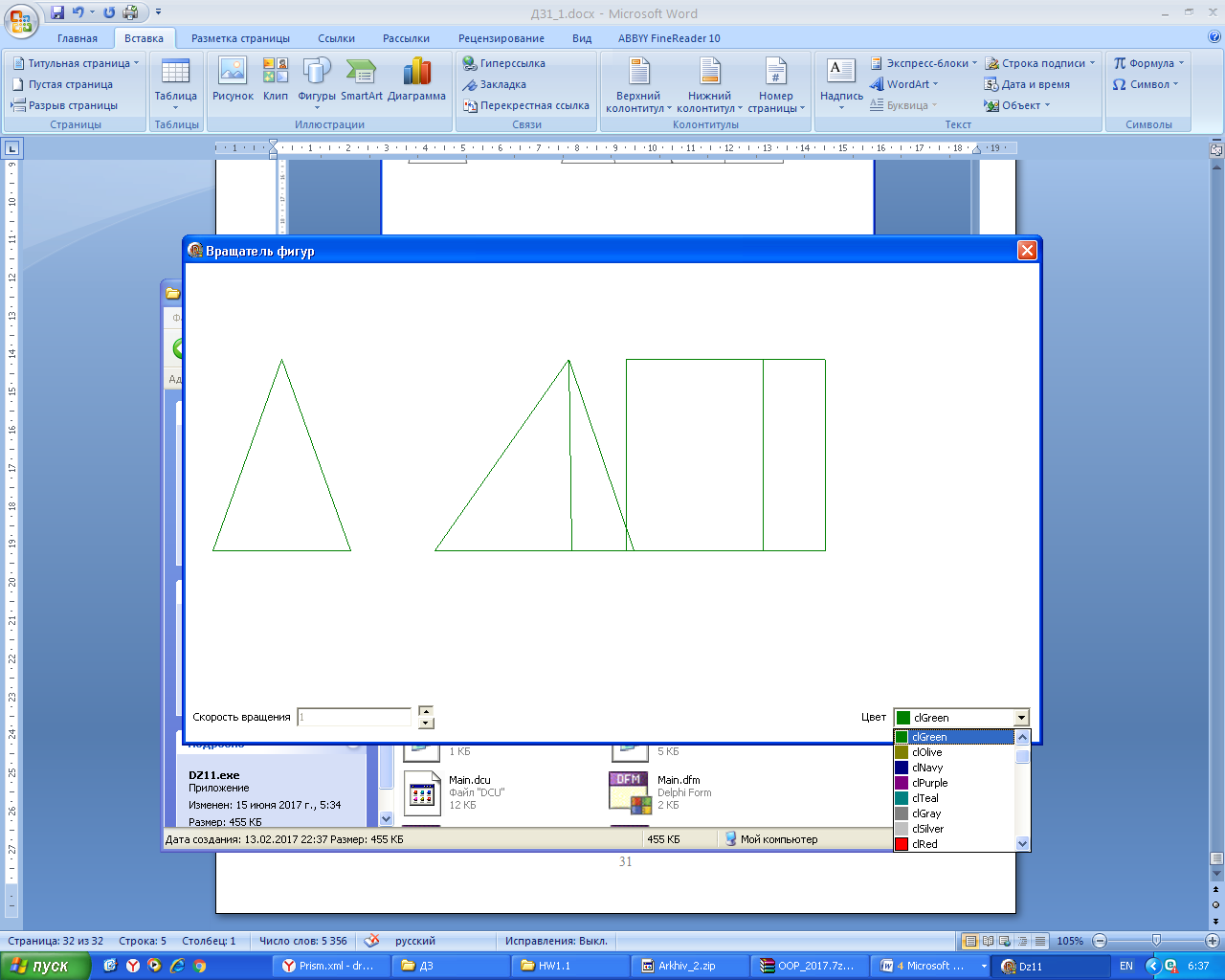
*Рисунок 10 – скриншот главной формы программы «Вращатель фигур» при остановленном вращении*

****

*Рисунок 11 – скриншот главной формы программы «Вращатель фигур». Фигуры находятся в движении*

****

*Рисунок 12 – скриншот главной формы программы «Вращатель фигур». Демонстрация работы интерфейсного управления скоростью вращения.*

****

*Рисунок 13 – скриншот главной формы программы «Вращатель фигур». Демонстрация возможность поменять цвет фигур во время вращения*

**Вывод:** Разработана иерархия классов, граф состояний пользовательского интерфейса. Выполнена объектная декомпозиция. На основе этих данных созданы модули для работы с 3D графикой и разработана программа, содержащая синхронно-вращающиеся три графических объекта – треугольник, пирамида и призма.

Проведено тестирование программа. По его результатам можно заключить, что программа работает корректно.