МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЦЕНТРАЛЬНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

Елементи векторної комп’ютерної графіки

Методичні вказівки до виконання лабораторних робіт
для студентів денної та заочної форми навчання за спеціальністю
123 “Комп’ютерна інженерія ”,125 “Кібербезпека”

ЗАТВЕРДЖЕНО
на засіданні кафедри кібербезпеки та
програмного забезпечення,
протокол від 28 квітня 2018 року № 14

КРОПИВНИЦЬКИЙ
2018

Елементи векторної комп’ютерної графіки: метод. вказівки до виконання лабораторних робіт для студентів денної та заочної форми навчання за спеціальністю 123 “Комп’ютерна інженерія”, 125 “Кібербезпека” / уклад. Дрєєва Г.М., Дрєєв О.М., Хох В.Д., Денисенко О.О. — Кропивницький: ЦНТУ, 2018. — 66 с.

Укладачі: Дрєєва Г.М., Дрєєв О.М. , Хох В.Д., Денисенко О.О.

Рецензенти: Якименко М.С. – кандидат фізико-математичних наук, доцент кафедри кібербезпеки та програмного забезпечення.

УДК 378.147:317

Рекомендовано до друку на засіданні кафедри кібербезпеки та програмного забезпечення (протокол № 14 від 28.04.2018 р.). Рецензенти: Якименко М.С. – кандидат фізико-математичних наук, доцент кафедри кібербезпеки та програмного забезпечення.

Методичні вказівки до виконання лабораторних робіт з комп'ютерної графіки: методичні вказівки для студентів денної та заочної форми навчання за спеціальністю 123 "Комп'ютерна інженерія", 125 "Кібербезпека" / уклад. Дрєєва Г.М., Дрєєв О.М., Хох В.Д. — Кропивницький: ЦНТУ, 2018. — 66 с.

Дисципліна є вибірковою у підготовці фахівців з програмування прикладного програмного забезпечення та системного програмування. Дисципліна надає знання в області структури засобів комп'ютерного формування та обробки зображення, його переробки та збереження в різних форматах, а також засоби стиснення графічної інформації.

В дисципліні розглядаються основні засади побудови растрового та векторного зображення, їх визначення, принципи реалізації апаратного відтворення зображення з точки зору програміста, та програмного забезпечення що повинне працювати з великою кількістю даних. Розглянуто основні типи зображень, архітектура програмних засобів, приклади побудови графічних алгоритмів перетворення координат та побудови тривимірних сцен, як програмним так і з апаратним обчисленням.

При вивченні дисципліни розглядаються найбільш відомі підходи, методи, алгоритми та засоби вирішення задач комп'ютерної графіки.

В результаті вивчення дисципліни студенти повинні знати:

– основні методи кодування та отримання комп'ютерного зображення;

– основні методи обробки зображення;

– основні методи креслення графічних примітивів в області апаратнозалежної системи координат;

– технології та програмні засоби розробки програм з використанням тривимірної графіки, зокрема з використанням графічного прискорення OpenGL.

В результаті вивчення дисципліни студенти повинні вміти:

– створювати власний код перетворення векторного простору;

– виводити тривимірне зображення на екран власноруч написаною програмою;

– використовувати графічний апаратний прискорювач для виводу тривимірної графіки;

– використовувати отриманих знань для створення нових форматів збереження векторного зображення.

Лабораторна робота №1

Тема: Комп’ютерні методи кодування зображення. Графічні примітиви.

Мета: Написати програму виведення на екран малюнків які складаються з графічних примітивів; визначити колір графічних примітивів за кольоровою схемою RGB32.

Теоретичні відомості

Поверхні, на яку програма може виводити графіку, мають властивість Canvas. У свою чергу, властивість canvas — це об'єкт типу TCanvas. Методи цього типу забезпечують виведення графічних примітивів (крапок, ліній, кіл, прямокутників і т.д.), а властивості дозволяють задати характеристики графічних примітивів, що виводяться: колір, товщину і стиль ліній; колір і вид заповнення областей; характеристики шрифту при виведенні текстової інформації.

Методи виведення графічних примітивів розглядають властивість Canvas як деяке абстрактне полотно, на якому вони можуть малювати (canvas переводиться як "поверхня", "полотно для малювання"). Полотно складається з окремих крапок — пікселів. Положення пікселів характеризується їх горизонтальними (X) і вертикальними (Y) координатами. Лівий верхній піксель має координати (0, 0). Координати зростають зверху вниз і зліва направо. Значення координат правої нижньої точки полотна залежать від розміру полотна. Розмір полотна можна отримати з властивостей Height і Width області (image).

Художник в своїй роботі використовує олівці і кисті. Методи, що забезпечують викреслювання на поверхні полотна графічних примітивів, теж використовують олівець і кисть. Олівець застосовується для викреслювання

ліній і контурів, а кисть — для зафарбовування (заливки) областей, обмежених контурами.

Олівцю і пензлику, що використовується для виведення графіки на полотні, відповідають властивості Pen (олівець) і Brush (пензлик), які є об'єктами типу TPen і TBrush, відповідно. Значення властивостей цих об'єктів визначають вид графічних елементів, що виводяться.

Pen. Олівець використовується для викреслювання ліній, контурів геометричних фігур: прямокутників, кіл, еліпсів, дуг і ін. Вид лінії, яку залишає олівець на поверхні полотна, визначають властивості об'єкту TPen , які перераховані в таблиці:

Властивості об'єкту TPen (олівець)

Властивість	**Визначає**
Color	Колір лінії
Width	Товщину лінії
Style	Вид лінії
Mode	Режим зображення

Властивість Color задає колір лінії. В таблиці нижче перераховані іменовані константи (тип TColor), які можна використовувати як значення властивості color.

Значення властивості Color визначає колір лінії

Константа	**Колір**	**Константа**	**Колір**
clBlack	Чорний	clSilver	Сріблястий
clMaroon	Каштановий	clRed	Червоний
clGreen	Зелений	clLime	Салатний

clOlive	Оливковий	clBlue	Синій
clNavy	Темно-синій	clFuchsia	Яскраво-рожевий
clPurple	Рожевий	clAqua	Бірюзовий
clTeal	Зелено-голубий	clWhite	Білий
clGray	Сірий		

Властивість Width задає товщину лінії (в пікселях). Наприклад, інструкція Canvas.Pen.width: =2 встановлює товщину лінії в 2 пікселі.

Властивість Style визначає вид (стиль) лінії, яка може бути безперервна або переривиста, складається з штрихів різної довжини. В таблиці нижче перераховані іменовані константи, що дозволяють задати стиль лінії.

Значення властивості Pen.Style визначає вид лінії

Константа	**Вид лінії**
psSolid	Суцільна лінія
psDash	Пунктирна лінія, довгі штрихи
psDot	Пунктирна лінія, короткі штрихи
psDashDot	Пунктирна лінія, чергування довгого і короткого штрихів
psDashDotDot	Пунктирна лінія, чергування одного довгого і двох коротких штрихів

psClear	Лінія не малюється (використовується, якщо не треба зображати межу області, наприклад, прямокутника)

Властивість Mode визначає, як формуватиметься колір точок лінії залежно від кольору точок полотна, через які ця лінія викреслюється. За замовчуванням вся лінія викреслюється кольором, яка визначається значенням властивості Pen.Color. Проте програміст може задати інверсний колір лінії по відношенню до кольору фону. Це гарантує, що незалежно від кольору фону всі ділянки лінії будуть видні, навіть в тому випадку, якщо колір лінії і колір фону збігаються.

Нижче перераховані деякі константи, які можна використовувати як значення властивості Pen.Mode.

Значення властивості Pen.Mode впливає на колір лінії

Константа	**Колір лінії**
pmBlack	Чорний, не залежить від значення властивості Pen.Color
pmWhite	Білий, не залежить від значення властивості Pen.Color
pmCopy	Колір лінії визначається значенням властивості Pen.Color
pmNotCopy	Колір лінії є інверсним по відношенню до значення властивості Pen.Color
pmNot	Колір точки лінії визначається як

	інверсний по відношенню до кольору точки полотна, в яку виводиться точка лінії

Пензель (canvas.Brush) використовується методами, що забезпечують викреслювання замкнених геометричних фігур та їх заливки (зафарбовування). Пензель, як об'єкт, володіє двома властивостями.

Властивості об'єкту TBrush (кисть)

Властивість	**Визначає**
Color	Колір зафарбовування (заливки) замкнутої області
Style	Стиль (тип) заповнення області

Область усередині контура може бути зафарбована або заштрихована. В першому випадку область повністю перекриває фон, а в другому — крізь не заштриховані ділянки області буде видний фон.

Як значення властивості Color можна використовувати будь-яку з констант типу TColor (див. список констант для властивості Pen.color).

Константи, що дозволяють задати стиль заповнення області, приведені в таблиці.

Значення властивості Brush.Style визначають тип зафарбовування

Константа	**Тип заповнення (заливки) області**
bsSolid	Суцільна заливка
bsClear	Область не зафарбовується

bsHorizontal	Горизонтальне штрихування
bsVertical	Вертикальне штрихування
bsFDiagonal	Діагональне штрихування з нахилом ліній вперед
bsBDiagonal	Діагональне штрихування з нахилом ліній назад
bsCross	Горизонтально-вертикальне штрихування, в клітинку
bsDiagCross	Діагональне штрихування, в клітинку

Виведення малюнків. Найбільш просто вивести малюнок, який знаходиться у файлі, можна за допомогою компоненту Image, значок якого знаходиться на вкладці Additional менеджера компонентів.

Властивості компоненту image

Властивість	**Визначає**
Picture	Малюнок, який зображається в полі компоненту
Width, Height	Розмір компоненту. Якщо розмір компоненту менше розміру малюнка, і значення властивостей AutoSize і Stretch рівно False, то зображається

	частина малюнка
AutoSize	Ознака автоматичної зміни розміру компоненту відповідно до реального розміру малюка
Stretch	Ознака автоматичного масштабування малюнка відповідно до реального розміру компоненту. Щоб було виконано масштабування, значення властивості AutoSize повинне бути False
Visible	Чи зображається компонент, і, відповідно, малюнок, на поверхні форми

Щоб вивести малюнок в полі компоненту image під час роботи програми, потрібно застосувати метод LoadFromFile до властивості Picture, вказавши як параметр ім'я файлу малюнка:

Form1.Image1.Picture.LoadFromFile('e:\Цуценя.bmp')

Хід роботи

1. Створіть та збережіть новий проект в середовищі Lazarus.

2. Додайте на головне вікно компонент TImage.

3. Подвійним кліком на формі (не по малюнку) додайте процедуру Form1.onCreate.

4. Створіть довільний простий малюнок (хатка, кицька, пацюк, тощо). Малюнок повинен бути унікальним для кожного виконавця роботи.

5. Перетворіть створений малюнок, який використовує базове положення, яке занесено в цілі змінні x та y.

6. Оформіть малюнок окремою процедурою, яка приймає значення (x,y:Integer) як параметри. Намалюйте декілька таких малюнків в різних місцях.

7. Прикрасьте свій малюнок декількома елементами.

8. Результат роздрукуйте та додайте до звіту разом з тектом програми.

9. Дайте відповіді на контрольні питання.

10. В разі виконання роботи на поточній парі дозволяється використання електроного звіту з усними відповідями на контрольні питання.

11. Зробіть висновки що до досяжності мети поставленої в лабораторній роботі.

Контрольні питання:

1. Що таке графічний примітив?

2. Колір визначається за яскравістю трьох компонент від 0 до 15. Яку кількість бітів займає значення кольору?

3. Чому при використанні 24 бітного кольору (3 байти, 1 байт на один кольоровий компонент) часто використовують 4 байтні змінні для зберігання пікселів малюнка?

4. Як можна задати графічний примітив “точка”?

5. Як можна задати графічний примітив “відрізок”?

6. Як можна задати графічний примітив “трикутник”?

7. Запишіть засоби Canvas для виведення графічних примітивів, всі що знаєте.

8. Як можна отримати отримані знання в реальних проектах?

Лабораторна робота №2

Тема: Двовимірне векторне зображення

Мета: Написати програму виведення на екран векторних малюнків з використанням перетворень переносу, масштабування та повороту.

Теоретичні відомості

У комп'ютерній графіці перетворення координат застосовують для зміни масштабу зображення, отримання симетричних частин графічних об'єктів, дублювання частин зображення тощо. Перетворення координат об'єкта — це їх переоб-числення за певними формулами. У подальшому розглядатимемо лише лінійні перетворення координат на площині, під час яких зберігаються такі геометричні властивості об'єктів: прямі лінії залишаються прямими, зберігається паралельність прямих, а також відношення площ геометричних фігур. Основними лінійними перетвореннями є паралельне перенесення, розтягування (стискання) і поворот геометричних об'єктів.

Припустимо, що у декартовій системі координат задана плоска геометрична фігура, (x,y) — координати однієї з її точок. Нехай (x1,y1) - координати тієї самої точки фігури після їх перетворення. Наведемо формули паралельного перенесення геометричної фігури на dx одиниць уздовж осі X і на dy одиниць уздовж осі Y (формула а), розтягування уздовж осі X у kx разів і вздовж осі У у ky разів (формула 6) і повороту навколо початку координат на кут α (формула в).

Лінійні перетворення проілюстровано на рис. 5.3. Необхідно зазначити, що будь-яке перетворення координат об'єкта можна розглядати як перетворення координатних осей.

а) $\begin{cases} x_1 = x \pm dx; \\ y_1 = y \pm dy; \end{cases}$ б) $\begin{cases} x_1 = x \cdot k_x; \\ y_1 = y \cdot k_y; \end{cases}$ в) $\begin{cases} x_1 = x \cdot \cos\alpha + y \cdot \sin\alpha; \\ x_1 = -x \cdot \sin\alpha + y \cdot \cos\alpha. \end{cases}$

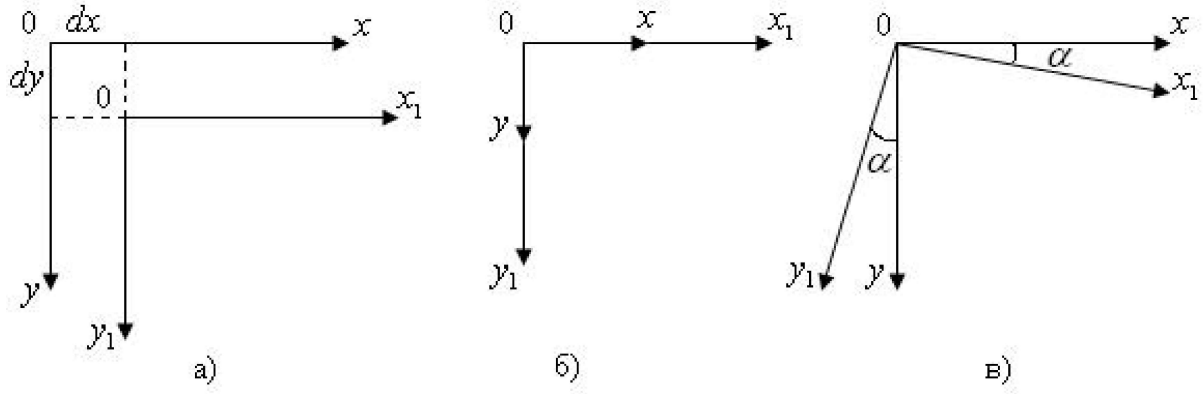


Рис. 1 – Перетворення координат

Хід роботи

1. Створіть та збережіть новий проект в середовищі Lazarus.

2. Додайте на головне вікно компонент TImage та TButton.

3. Подвійним кліком на кнопці додайте процедуру Form1.onButton1Click.

4. Створіть довільний простий малюнок (хатка, кицька, пацюк, тощо). Малюнок повинен бути унікальним для кожного виконавця роботи.

5. Виведіть малюнок в різному масштабуванні та кутах повороту.

6. За допомогою інших елементів керування зробіть налаштування що до поточного масштабу та куту повороту з метою змінювання одного з зображень без перекомпіляції програми.

7. Створіть композицію з намальованих елементів, щоб малюнок виглядав цілісним.

8. Результат роздрукуйте та додайте до звіту разом з текстом програми.

9. Дайте відповіді на контрольні питання.

10. В разі виконання роботи на поточній парі дозволяється використання електронного звіту з усними відповідями на контрольні питання.

11. Зробіть висновки що до досяжності мети поставленої в лабораторній роботі.

Контрольні питання:

1. Поясніть спотворення еліпсу яке зображає око песика з теоретичних відомостей.

2. Що ви запропонуєте для використання почергово декількох перетворювачів координат?

3. Що потрібно для малювання графічного примітиву “прямокутник” з використанням перетворення обертання?

4. Які зміни потрібно зробити, щоб обертання відбувалося в зворотному напрямку?

5. Який максимальний кут обертання зображення?

6. Що ви зробите, щоб намалювати зображення догори ногами без використання обертання?

Додаток А. Програма «песики»

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
Classes, SysUtils, FileUtil, Forms, Controls, Graphics,
Dialogs, ExtCtrls, StdCtrls;

type
{ TForm1 }
TTransformer = class
public
x, y: Integer;
procedure SetXY(dx,dy: Integer);
end;

TForm1 = class(TForm)
Button1: TButton;
Image1: TImage;
procedure Button1Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure FormDestroy(Sender: TObject);
private
{ private declarations }
public
{ public declarations }
Tra: TTransformer;
procedure MLine(x1,y1,x2,y2:Integer);
procedure MEllipse(x1,y1,x2,y2:Integer);
procedure DrawPesik;
```

```
end;

var
Form1: TForm1;

implementation

{$R *.lfm}
{ TForm1 }
procedure TForm1.DrawPesik;
var i: Integer;
begin
Image1.Canvas.Pen.Color:=clBlack;
for i:=0 to 7 do begin Mline(i*2,10,i*2,20); end;
for i:=0 to 7 do begin MLine(14+i*2,0,14+i*2,20); end;
for  i:=0  to  20  do  begin  MLine(24+i*2,20,24+i*2,40);
end;
for i:=0 to 8 do begin MLine(62+i*2,20,62+i*2,25); end;
Image1.Canvas.Brush.Color:=clWhite;
MEllipse(12,3,17,8);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
Image1.Canvas.Brush.Color:=clWhite;
Image1.Canvas.FillRect(0,0,Image1.Width,Image1.Height);
Tra.SetXY(100,100); DrawPesik;
Tra.SetXY(150,150); DrawPesik;
Tra.SetXY(200,200); DrawPesik;
Tra.SetXY(250,100); DrawPesik;
```

```
Tra.SetXY(300,150); DrawPesik;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin Tra := TTransformer.Create; end;

procedure TForm1.FormDestroy(Sender: TObject);
begin Tra.Destroy; end;

procedure TForm1.MLine(x1,y1,x2,y2:Integer);
begin
Image1.Canvas.Line(x1+Tra.x,
y1+Tra.y,
x2+Tra.x,
y2+Tra.y);
end;

procedure TForm1.MEllipse(x1,y1,x2,y2:Integer);
begin
Image1.Canvas.Ellipse(x1+Tra.x,
y1+Tra.y,
x2+Tra.x,
y2+Tra.y);
end;

procedure TTransformer.SetXY(dx,dy: Integer);
begin x:=dx; y:=dy; end;

end.
```

Додаток Б. Масштабовані песики.

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
Classes, SysUtils, FileUtil, Forms, Controls, Graphics,
Dialogs, ExtCtrls,
StdCtrls;

type

{ TForm1 }

TTransformer = class
public
x, y: Integer;
mx, my: Double;
procedure SetXY(dx,dy: Integer);
procedure SetMXY(masX, masY: Double);
function GetX(oldX:Integer):Integer;
function GetY(oldY:Integer):Integer;
end;

TForm1 = class(TForm)
Button1: TButton;
Image1: TImage;
```

```
procedure Button1Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure FormDestroy(Sender: TObject);
private
{ private declarations }
public
{ public declarations }
Tra: TTransformer;
procedure MLine(x1,y1,x2,y2:Integer);
procedure MEllipse(x1,y1,x2,y2:Integer);
procedure DrawPesik;
end;

var
Form1: TForm1;

implementation

{$R *.lfm}

{ TForm1 }

procedure TForm1.DrawPesik;
var i: Integer;
begin
Image1.Canvas.Pen.Color:=clBlack;
for i:=0 to 7 do begin
MLine(i*2,10,i*2,20);
end;
for i:=0 to 7 do begin
```

```
MLine(14+i*2,0,14+i*2,20);
end;
for i:=0 to 20 do begin
MLine(24+i*2,20,24+i*2,40);
end;
for i:=0 to 8 do begin
MLine(62+i*2,20,62+i*2,25);
end;
Image1.Canvas.Brush.Color:=clWhite;
MEllipse(12,3,17,8);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
Image1.Canvas.Brush.Color:=clWhite;
Image1.Canvas.FillRect(0,0,Image1.Width,Image1.Height);
Tra.SetXY(50,100); Tra.SetMXY(1.0,1.0);
DrawPesik;
Tra.SetXY(100,150); Tra.SetMXY(1.4,1.4);
DrawPesik;
Tra.SetXY(150,220); Tra.SetMXY(1.8,1.8);
DrawPesik;
Tra.SetXY(200,100); Tra.SetMXY(0.8,0.8);
DrawPesik;
Tra.SetXY(250,150); Tra.SetMXY(1.4,1.4);
DrawPesik;
Tra.SetXY(300,300); Tra.SetMXY(-1.0,1.0);
DrawPesik;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
Tra := TTransformer.Create;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
Tra.Destroy;
end;

procedure TForm1.MLine(x1,y1,x2,y2:Integer);
begin
Image1.Canvas.Line(Tra.GetX(x1),
Tra.GetY(y1),
Tra.GetX(x2),
Tra.GetY(y2));
end;

procedure TForm1.MEllipse(x1,y1,x2,y2:Integer);
begin
Image1.Canvas.Ellipse(Tra.GetX(x1),
Tra.GetY(y1),
Tra.GetX(x2),
Tra.GetY(y2));
end;

{ TTransformer }

procedure TTransformer.SetXY(dx,dy: Integer);
begin
```

```
x:=dx; y:=dy;
end;

procedure TTransformer.SetMXY(masX, masY: Double);
begin
mx:=masX; my:=masY;
end;

function TTransformer.GetX(oldX:Integer):Integer;
begin
GetX := Round( mx*oldX + x);
end;

function TTransformer.GetY(oldY:Integer):Integer;
begin
GetY := Round( my*oldY + y);
end;

end.
```

Додаток В. Собаки з поворотом

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
Classes, SysUtils, FileUtil, Forms, Controls, Graphics,
Dialogs, ExtCtrls,
StdCtrls;

type

{ TForm1 }

TTransformer = class
public
x, y: Integer;
mx, my: Double;
alpha: Double;
procedure SetXY(dx,dy: Integer);
procedure SetMXY(masX, masY: Double);
procedure SetAlpha(a: Double);
function GetX(oldX, oldY:Integer):Integer;
function GetY(oldX, oldY:Integer):Integer;
end;

TForm1 = class(TForm)
```

```
Button1: TButton;
Image1: TImage;
procedure Button1Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure FormDestroy(Sender: TObject);
private
{ private declarations }
public
{ public declarations }
Tra: TTransformer;
procedure MLine(x1,y1,x2,y2:Integer);
procedure MEllipse(x1,y1,x2,y2:Integer);
procedure DrawPesik;
end;

var
Form1: TForm1;

implementation

{$R *.lfm}

{ TForm1 }

procedure TForm1.DrawPesik;
var i: Integer;
begin
Image1.Canvas.Pen.Color:=clBlack;
for i:=0 to 7 do begin
MLine(i*2,10,i*2,20);
```

```
end;
for i:=0 to 7 do begin
MLine(14+i*2,0,14+i*2,20);
end;
for i:=0 to 20 do begin
MLine(24+i*2,20,24+i*2,40);
end;
for i:=0 to 8 do begin
MLine(62+i*2,20,62+i*2,25);
end;
Image1.Canvas.Brush.Color:=clWhite;
MEllipse(12,3,17,8);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
Image1.Canvas.Brush.Color:=clWhite;
Image1.Canvas.FillRect(0,0,Image1.Width,Image1.Height);
Tra.SetAlpha(0.0);
Tra.SetXY(50,100); Tra.SetMXY(1.0,1.0);
DrawPesik;
Tra.SetXY(100,150); Tra.SetMXY(1.4,1.4);
DrawPesik;
Tra.SetXY(150,220); Tra.SetMXY(1.8,1.8);
DrawPesik;
Tra.SetXY(200,100); Tra.SetMXY(0.8,0.8);
DrawPesik;
Tra.SetAlpha(PI/4.0); //Тут змінено кут
Tra.SetXY(250,150); Tra.SetMXY(1.4,1.4);
DrawPesik;
```

```
Tra.SetXY(300,350); Tra.SetMXY(-1.0,1.0);
DrawPesik;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
Tra := TTransformer.Create;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
Tra.Destroy;
end;

procedure TForm1.MLine(x1,y1,x2,y2:Integer);
begin
Image1.Canvas.Line(Tra.GetX(x1,y1),
Tra.GetY(x1,y1),
Tra.GetX(x2,y2),
Tra.GetY(x1,y2));
end;

procedure TForm1.MEllipse(x1,y1,x2,y2:Integer);
begin
Image1.Canvas.Ellipse(Tra.GetX(x1,y1),
Tra.GetY(x1,y1),
Tra.GetX(x2,y2),
Tra.GetY(x2,y2));
end;
```

```
{ TTransformer }
procedure TTransformer.SetXY(dx,dy: Integer);
begin x:=dx; y:=dy; end;

procedure TTransformer.SetMXY(masX, masY: Double);
begin mx:=masX; my:=masY; end;

function TTransformer.GetX(oldX, oldY:Integer):Integer;
begin
GetX := Round( mx*oldX*cos(alpha) - my*oldY*sin(alpha)
+ x);
end;

function TTransformer.GetY(oldX, oldY:Integer):Integer;
begin
GetY := Round( mx*oldX*sin(alpha) + my*oldY*cos(alpha)
+ y);
end;

procedure TTransformer.SetAlpha(a: Double);
begin
alpha:=a;
end;

end.
```

Лабораторна робота №3

Тема: Двовимірне векторне зображення. Списки елементів

Мета: Створити формат збереження векторного зображення. Додати рухомі елементи.

Теоретичні відомості

Доступ до файлу в програмі відбувається за допомогою змінних файлового типу. Змінну файлового типу описують одним з трьох способів:

`file of тип` - типізований файл (вказаний тип компоненти);

`text` - текстовий файл;

`file` - нетипізований файл.

Приклади опису файлових змінних:

```
var
f1: file of char;
f2: file of integer;
f3: file;
t: text;
```

Будь-які дискові файли стають доступними програмі після зв'язування їх з файловою змінною, оголошеної в програмі. Всі операції в програмі здійснюються тільки за допомогою пов'язаної з ним файлової змінної

```
Assign(f, FileName);
```

пов'язує файлову змінну f з фізичним файлом, повне ім'я якого задано в рядку FileName. Встановлений зв'язок буде чинним до кінця роботи програми, або до тих пір, поки не буде зроблено перепризначення.

Після зв'язку файлової змінної з дисковим ім'ям файлу в програмі потрібно вказати напрямок передачі даних (відкрити файл). В залежності від цього напрямку говорять про читання з файлу або запису в файл.

`Reset(f)` відкриває для читання файл, з яким пов'язана файлова змінна f. Після успішного виконання процедури Reset файл готовий до читання з нього першого елемента. Процедура завершується з повідомленням про помилку, якщо зазначений файл не знайдений.

Якщо f - типізований файл, то процедурою reset він відкривається для читання і запису одночасно.

`Rewrite(f)` відкриває для запису файл, з яким пов'язана файлова змінна f. Після успішного виконання цієї процедури файл готовий до запису в нього першого елемента. Якщо вказаний файл вже існував, то всі дані з нього видаляються.

`Close(f)` закриває відкритий до цього файл з файлової змінної f. Виклик процедури Close необхідний при завершенні роботи з файлом. Якщо з якоїсь причини процедура Close не буде виконана, файл усе-таки буде створений на зовнішньому пристрої, але вміст останнього буфера в нього не буде перенесено.

`EOF(f): boolean` повертає значення TRUE, коли при читанні досягнутий кінець файлу. Це означає, що вже прочитаний останній елемент у файлі або файл після відкриття виявився порожній.

Текстовий файл - це сукупність рядків, розділених мітками кінця рядка. Сам файл закінчується міткою кінця файлу. Доступ до кожного рядка можливий лише послідовно, починаючи з першого. Одночасний запис і читання заборонені.

Читання з текстового файлу:

```
Read(f, список змінних);
ReadLn(f, список змінних);
```

Процедури зчитують інформацію з файлу f в змінні. Спосіб читання залежить від типу змінних, що стоять в списку. У змінну char записуються символи з файлу. Запис у числову змінну відбувається за таким принципом: пропускаються символи-роздільники, початкові пробіли і зчитується значення числа до появи наступного роздільника. У змінну типу string

записується кількість символів, яка дорівнює довжині рядка, але тільки в тому випадку, якщо до цього не зустрілися символи кінця рядка або кінця файлу. Відмінність ReadLn від Read полягає в тому, що перша команда після прочитання даних пропустить решту символів рядку, включаючи мітку кінця рядка. Якщо список змінних відсутній, то процедура ReadLn(f) пропускає рядок при читанні текстового файлу.

Запис в текстовий файл:

```
Write(f, список змінних);
WriteLn(f, список змінних);
```

Процедури записують інформацію в текстовий файл. Спосіб запису залежить від типу змінних в списку (як і при виведенні на екран). Враховується формат виводу. WriteLn від Write відрізняється тим, що після запису всіх значень зі змінних записує ще й символ кінця рядка (формується закінчений рядок файлу).

Задання векторних зображень

Полігональна сітка являє собою сукупність ребер, вершин і багатокутників. Вершини з'єднуються ребрами, а багатокутники розглядаються як послідовності ребер або вершин. Сітку можна задавати декількома різними способами. Прикладного програмісту необхідно вибрати спосіб, який найбільш підходить для його завдання. Зрозуміло, в одному завданню може з однаковим успіхом використовуватися відразу кілька підходів: для зовнішньої пам'яті, внутрішнього використання і користувача. Для оцінки цих уявлень використовуються наступні критерії:

- Об'єм пам'яті;
- Простота ідентифікації ребер, інцидентних вершині;
- Простота ідентифікації багатокутників, яким належить дане ребро;
- Простота процедури пошуку вершин, що утворюють ребро;

- Легкість визначення всіх ребер, що утворюють багатокутник;
- Простота отримання зображення полігональної сітки;
- Простота виявлення помилок в поданні (наприклад, відсутність ребра, вершини або багатокутника).

У загальному випадку, чим більше явно виражені залежності між багатокутниками, вершинами і ребрами, тим швидше виконують операції над ними і тим більше пам’яті вимагає відповідне подання. У деяких випадках ребра полігональних сіток є спільними для більше ніж двох багатокутників.

Розглянемо три найбільш поширених способу опису полігональних сіток.

1) Явне завдання багатокутників.

Кожен багатокутник можна представити у вигляді списку координат його вершин:

P = ((X1, Y1, Z1), (X2, Y2, Z2), ..., (XN, YN, ZN)).

Вершини запам’ятовуються в тому порядку, в якому вони зустрічаються при обході навколо багатокутника. При цьому всі послідовні вершини багатокутника, а також перша і остання з’єднуються ребрами. Для кожного окремого багатокутника даний спосіб запису є ефективним, проте для полігональної сітки мають місце втрати пам’яті внаслідок дублювання інформації про координати загальних вершин. Більш того, явного опису загальних ребер і вершин просто не існує. Наприклад, пошук всіх багатокутників, які мають загальну вершину, вимагає порівняння трійок координат одного багатокутника з трійками координат всіх інших багатокутників. Найбільш ефективний спосіб виконати таке порівняння полягає в сортуванні всіх N координатних трійок: для цього буде потрібно в кращому випадку порівнянь. Але і тоді існує небезпека, що одна і та ж вершина внаслідок помилок округлення може в різних багатокутника мати різні значення координат. Полігональна сітка зображується шляхом креслення ребер кожного багатокутника, однак це призводить до того, що

загальні ребра малюються двічі - по одному разу для кожного з багатокутників. Окремий багатокутник зображується тривіально.

2) Завдання багатокутників за допомогою покажчиків.

При використанні цього подання кожен вузол полігональної сітки запам'ятовується лише один раз в списку вершин V = ((X1, Y1, Z1), ..., (XN, YN, ZN)) . Багатокутник визначається списком покажчиків (або індексів) в списку вершин. Багатокутник складений з вершин 3, 5, 7 і 10 цього списку представляється як P = (3,5,7,10). Таке представлення має ряд переваг в порівнянні з явним завданням багатокутників. Оскільки кожна вершина багатокутника запам'ятовується лише один раз, вдається заощадити значний обсяг пам'яті. Крім того, координати вершини можна легко змінювати. Однак все ще не просто відшукувати багатокутники із загальними ребрами; останні при зображенні всієї полігональної фігури як і раніше малюються двічі. Ці проблеми можна вирішити, якщо описувати ребра в явному вигляді.

V = (V1, V2, V3, V4) = ((X1, Y1, Z1), ..., (X4, Y4, Z4)),

P1 = (1,2,4), P2 = (4,2,3).

3) Явне завдання ребер.

У цьому поданні є список вершин V, проте будемо розглядати багатокутник не як список покажчиків на список вершин, а як сукупність покажчиків на елементи списку ребер, в якому ребра зустрічаються лише один раз. Кожне ребро в списку ребер вказує на дві вершини в списку вершин, що визначають це ребро, а також на один або два багатокутника, яким це ребро належить. Таким чином, описуємо багатокутник як P = (E1, .., EN) , а ребро як E = (V1, V2, P1, P2) . Якщо ребро належить тільки одному багатокутнику, то або P1 або P2 - порожньо.

При явному завданні ребер сітка зображується шляхом креслення не всіх багатокутників, а всіх ребер. В результаті вдається уникнути багаторазового малювання загальних ребер. Окремі багатокутники при цьому також зображуються досить просто.

Ні в одному з цих уявлень завдання визначення ребер, інцидентних вершині, не є простою - для її вирішення необхідно перебрати всі ребра.

Хід роботи

1. Створіть та збережіть новий проект в середовищі Lazarus.

2. Додайте на головне вікно компонент TImage та TTimer.

3. Подвійним кліком на TTimer додайте процедуру Form1.onTimer.

4. Створіть довільний простий малюнок (кицька, пацюк, тощо). Малюнок повинен бути унікальним для кожного виконавця роботи. Збережіть малюнок у файл у вигляді списку точок та ламаних ліній.

5. Виведіть малюнок в різному масштабуванні та кутах повороту.

6. За допомогою додаткового лічильника кадрів організуйте рух вашого елементу малюнка.

7. Створіть композицію з намальованих елементів, щоб малюнок виглядав цілісним.

8. Результат роздрукуйте та додайте до звіту разом з текстом програми.

9. Дайте відповіді на контрольні питання.

10. В разі виконання роботи на поточній парі дозволяється використання електронного звіту з усними відповідями на контрольні питання.

11. Зробіть висновки що до досяжності мети поставленої в лабораторній роботі.

Контрольні питання:

1. Що є більш економним для вашого малюнка, список точок та ліній за допомогою номерів точок, чи повний перелік координат для кожної лінії?

2. Як сильно завантажений процесор вашого ПК при перегляді рухомого зображення в створеній програмі?

3. Запропонуйте схему використання декількох перетворювачів координат, по одному для кожного з рухомих елементів?

4. Що Ви запропонуєте для створення зафарбованих полігонів?

5. Запропонуйте схеми додавання кольору до малюнка.

6. Чи можна у використаній схемі кодування малюка в одному файлі тримати декілька окремих об’єктів?

Додаток. Рухомий страус.

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
Classes, SysUtils, FileUtil, Forms, Controls, Graphics,
Dialogs, ExtCtrls,
StdCtrls;

type

TMPoint = record
x, y: Double;
end;
TMyImage = record
Points: array of TMPoint;
PolyLine: array of Integer;
end;

{ TForm1 }

TTransformer = class
public
x, y: integer;
mx, my: double;
alpha: double;
```

```
procedure SetXY(dx, dy: integer);
procedure SetMXY(masX, masY: double);
procedure SetAlpha(a: double);
function GetX(oldX, oldY: integer): integer;
function GetY(oldX, oldY: integer): integer;
end;

TForm1 = class(TForm)
Button1: TButton;
Image1: TImage;
Timer1: TTimer;
procedure Button1Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure FormDestroy(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
private
{ private declarations }
public
{ public declarations }
Tra: TTransformer;
Straus: TMyImage;
frameCount: Integer;
procedure MLine(x1, y1, x2, y2: integer);
procedure MEllipse(x1, y1, x2, y2: integer);
procedure DrawPesik;
procedure DrawStraus;
procedure LoadPicFromFile(var Image:TMyImage; FileName:
String);
procedure DrawMyImage(var Image:TMyImage);
end;
```

```
var
Form1: TForm1;

implementation

{$R *.lfm}

{ TForm1 }

procedure TForm1.DrawMyImage(var Image:TMyImage);
var
t,i,n: Integer;
begin
n:=Length(Image.PolyLine);
t:=-1;
for i:=0 to n-1 do begin
if (t>=0)and(Image.PolyLine[i]>=0) then begin
MLine(Round(Image.Points[t].x),
Round(Image.Points[t].y),
Round(Image.Points[Image.PolyLine[i]].x),
Round(Image.Points[Image.PolyLine[i]].y) );
end;
t:=Image.PolyLine[i];
end;
end;

procedure   TForm1.LoadPicFromFile(var   Image:TMyImage;
FileName: String);
var
```

```
i,n: Integer;
p: TMPoint;
F: Text;
begin
system.assign(F,FileName);
system.reset(F);
ReadLn(F,n);
SetLength(Image.Points,n);
for i:=0 to n-1 do begin
ReadLn(F,p.x,p.y);
Image.Points[i]:=p;
end;
ReadLn(F,n);
SetLength(Image.PolyLine,n);
for i:=0 to n-1 do begin
Read(F,Image.PolyLine[i]);
end;
system.close(F);
end;

procedure TForm1.DrawPesik;
var
i: integer;
begin
Image1.Canvas.Pen.Color := clBlack;
for i := 0 to 7 do
begin
MLine(i * 2, 10, i * 2, 20);
end;
for i := 0 to 7 do
```

```
begin
MLine(14 + i * 2, 0, 14 + i * 2, 20);
end;
for i := 0 to 20 do
begin
MLine(24 + i * 2, 20, 24 + i * 2, 40);
end;
for i := 0 to 8 do
begin
MLine(62 + i * 2, 20, 62 + i * 2, 25);
end;
Image1.Canvas.Brush.Color := clWhite;
MEllipse(12, 3, 17, 8);
end;

procedure TForm1.DrawStraus;
begin
MLine(-30, -20, -20, -25);
MLine(-20, -25, -15, -30);
MLine(-15, -30, -10, -30);
MLine(-10, -30, -5, -5);
MLine(-5, -5, 0, -5);
MLine(0, -5, 15, -10);
MLine(15, -10, 15, 0);
MLine(15, 0, 5, 5);
MLine(15, -5, 20, -5);
MLine(20, -5, 25, -10);
MLine(25, -10, 20, 5);
MLine(20, 5, 15, 10);
MLine(15, 10, 10, 20);
```

```
MLine(10, 20, 5, 10);
MLine(5, 10, -5, 5);
MLine(-5, 5, -15, -20);
MLine(-15, -20, -30, -20);
MLine(-30, -20, -20, -23);
MLine(10, 20, 5, 30);
MLine(-5, 30, 10, 30);
MLine(-5, 25, 5, 30);
MLine(-5, 35, 5, 30);
MLine(10, 0, 15, -5);
MLine(10, -5, 15, -10);
MLine(-15, -25, -15, -28);
MLine(-15, -28, -12, -28);
MLine(-12, -28, -12, -25);
MLine(-12, -25, -15, -25);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
Image1.Canvas.Brush.Color := clWhite;
Image1.Canvas.FillRect(0,        0,        Image1.Width,
Image1.Height);
Tra.SetAlpha(0.0);
Tra.SetXY(50, 100);
Tra.SetMXY(1.0, 1.0);
DrawMyImage(Straus);
Tra.SetXY(100, 150);
Tra.SetMXY(1.4, 1.4);
DrawMyImage(Straus);
Tra.SetXY(150, 220);
```

```
Tra.SetMXY(-1.8, 1.8);
DrawMyImage(Straus);
Tra.SetXY(200, 100);
Tra.SetMXY(0.8, 0.8);
DrawMyImage(Straus);
Tra.SetAlpha(PI / 4.0);
Tra.SetXY(250, 150);
Tra.SetMXY(1.4, 1.4);
DrawMyImage(Straus);
Tra.SetXY(300, 350);
Tra.SetMXY(-1.0, 1.0);
DrawMyImage(Straus);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
frameCount:=0;
Tra := TTransformer.Create;
LoadPicFromFile(Straus,'straus.txt');
Timer1.Enabled:=true;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
Timer1.Enabled:=false;
Tra.Destroy;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
```

```
Image1.Canvas.Brush.Color:=clWhite;
Image1.Canvas.FillRect(0,0,Image1.Width,Image1.Height);
Tra.SetXY(200 + Round(150.0*sin(0.05*frameCount)),200 -
abs(Round(10.0*sin(0.4*frameCount))));
Tra.SetAlpha(0.0);
if cos(0.05*frameCount)>0.0 then begin
Tra.SetMXY(-1.0,1.0);
end else begin
Tra.SetMXY(1.0,1.0);
end;
DrawMyImage(Straus);
frameCount:=frameCount+1;
end;

procedure TForm1.MLine(x1, y1, x2, y2: integer);
begin
Image1.Canvas.Line(Tra.GetX(x1, y1),
Tra.GetY(x1, y1),
Tra.GetX(x2, y2),
Tra.GetY(x2, y2));
end;

procedure TForm1.MEllipse(x1, y1, x2, y2: integer);
begin
Image1.Canvas.Ellipse(Tra.GetX(x1, y1),
Tra.GetY(x1, y1),
Tra.GetX(x2, y2),
Tra.GetY(x2, y2));
end;
```

```
{ TTransformer }

procedure TTransformer.SetXY(dx, dy: integer);
begin
x := dx;
y := dy;
end;

procedure TTransformer.SetMXY(masX, masY: double);
begin
mx := masX;
my := masY;
end;

function TTransformer.GetX(oldX, oldY: integer):
integer;
begin
GetX := Round(mx * oldX * cos(alpha) - my * oldY *
sin(alpha) + x);
end;

function TTransformer.GetY(oldX, oldY: integer):
integer;
begin
GetY := Round(mx * oldX * sin(alpha) + my * oldY *
cos(alpha) + y);
end;

procedure TTransformer.SetAlpha(a: double);
begin
```

```
alpha := a;
end;

end.
```

Лабораторна робота №4

Тема: Ієрархічна система побудови графічної сцени

Мета: Створити формат збереження векторного зображення. Додати рухомі елементи.

Теоретичні відомості

Перший об’єкт нехай рухається поступово вздовж осі OX, а другий — рухається навколо першого по колу. Запишемо перетворення координат для руху вздовж відрізку:

$$\begin{cases} x := x_0 + (x_1 - x_0) \cdot t, \\ y := y_0 + (y_1 - y_0) \cdot t, \end{cases}$$, де 0≤t≤1.

Відповідно, для руху по колу теж можна використати параметричний запис:

$$\begin{cases} x := x_0 + r_x \cos(2\pi \cdot t), \\ y := y_0 - r_y \sin(2\pi \cdot t), \end{cases}$$, де 0≤t≤1.

Але для того щоб другий об’єкт пересувався навколо першого, потрібно сумістити пересування по прямій та по колу:

$$\begin{cases} x := x_0 + (x_1 - x_0) \cdot t + r_x \cos(2\pi n t), \\ y := y_0 - (y_1 - y_0) \cdot t - r_y \sin(2\pi n t), \end{cases}$$,

де 0≤t≤1, n — кількість обертів на пересування вздовж відрізку.

$$\begin{cases} x_1 := dx_0 + x_0 \cos(\alpha) + y_0 \sin(\alpha) \\ y_1 := dy_0 - x_0 \sin(\alpha) + y_0 \cos(\alpha) \end{cases}, \quad \begin{cases} x_2 := dx_1 + x_1 \cos(\alpha) + y_1 \sin(\alpha) \\ y_2 := dy_1 - x_1 \sin(\alpha) + y_1 \cos(\alpha) \end{cases}.$$

Це можна досягти зміною класу векторних об’єктів додаванням списку залежних від нього об’єктів:

```
TMyImage = class
public
```

```
Tranform: TTransformer; //Відносне перетворення
activeTransform: TTransformer; //Активне перетворення
Child: TList; //Список залежних елементів
Points: array of TMPoint; //Список координат точок
PolyLine: array of Integer; //Список індексів точок
constructor Create;
destructor Destroy; override;
procedure LoadPicFromFile(FileName: String);
procedure DrawMyImage(Canvas: TCanvas); //Малювати на вказаній канві
procedure AddChild(newImage:TMyImage); //Додати залежний об'єкт
end;
```

Та зміною класу перетворення координат:

```
procedure Ttransformer.AddTransform(nextTransform:TTransformer);
```

Хід роботи

1. Створіть та збережіть новий проект в середовищі Lazarus.
2. Додайте на головне вікно компонент TImage та TTimer.
3. Подвійним кліком на TTimer додайте процедуру Form1.onTimer.
4. Створіть довільний малюнок. Малюнок повинен бути унікальним для кожного виконавця роботи. Малюнок повинен складатися з кількох ієрархічно пов'язаних частин з власною формою руху відносно основної частини (колеса, що обертаються та підстрибують, качаються руки в рухомого чоловічка, тощо). Збережіть компоненти малюнка у окремі файли у вигляді списків точок та ламаних ліній.

5. Змініть текст програми з минулих робіт для підтримки послідовного перетворення координат з метою організації відносності положення елементів.

6. За допомогою додаткового лічильника кадрів організуйте рух вашого елементу малюнка.

7. Створіть композицію з намальованих елементів, щоб малюнок виглядав цілісним.

8. Результат роздрукуйте та додайте до звіту разом з текстом програми.

9. Дайте відповіді на контрольні питання.

10. В разі виконання роботи на поточній парі дозволяється використання електронного звіту з усними відповідями на контрольні питання.

11. Зробіть висновки що до досяжності мети поставленої в лабораторній роботі.

Контрольні питання:

1. Яка послідовність перетворення координат потрібна якщо навколо об’єкту, що обертається, обертається інший об’єкт?

2. Чи можна за розробленою схемою використати один файл для збереження багатокомпонентного малюнка?

3. Як змінити програму для малювання фарбованих зображень?

4. Як знайти внутрішню точку для опуклого багатокутника?

5. Чи буде корисним використання зміни послідовності застосувань перетворень координат для малювання?

Лабораторна робота №5

Тема: Матричний запис перетворення координат

Мета: Створити процедури матричної арифметики для матриць розміром 3x3 та використати їх в перетворенні координат.

Теоретичні відомості

В якості перетворювача координат потрібно мати матриці фіксованої розмірності, а саме 3x3. Наведемо опис типу даних для таких матриць:

```
TMatrix = array [0..2,0..2] of Double;
```

Тепер можна визначити процедури множення матриці на іншу матрицю:

```
function MulMatrix(var A: Tmatrix; var B: Tmatrix):TMatrix;
var i,j,k:Integer;
C:TMatrix;
begin
for i:=0 to 3 do begin
for j:=0 to 3 do begin
C[i,j] := 0.0;
for k:=0 to 3 do begin
C[i,j]:=C[i,j] + A[k,i]*B[j,k];
end;
end;
end;
MulMatrix:=C;
end;
```

Для визначення перетворених координат тепер не є доцільним окреме рахування, більш заощадливим рішенням буде сумістити процедуру для знаходження двох координат одним множенням:

```
function Ttransformer.GetX(oldX, oldY:Double):Double;
begin
GetX:=(M[0,0]*oldX + M[1,0]*oldY + M[2,0])/M[2,2];
end;
```

Аналогічна функція є й для розрахування й вертикальної координати.

Також для роботи з матричними перетвореннями буде корисним створити функції для задання одиничної матриці перетворення, додавання до існуючої матриці операції масштабування, повороту та переносу за формулами:

$$\begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = \begin{pmatrix} m_x \cos(\alpha) & m_y \sin(\alpha) & dx \\ -m_x \sin(\alpha) & m_y \cos(\alpha) & dy \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ 1 \end{pmatrix}.$$

Хід роботи

1. Створіть та збережіть новий проект в середовищі Lazarus.

2. Додайте на головне вікно компонент TImage та TTimer.

3. Подвійним кліком на TTimer додайте процедуру Form1.onTimer.

4. Створіть довільний малюнок. Малюнок повинен бути унікальним для кожного виконавця роботи. Малюнок повинен складатися з кількох ієрархічно пов’язаних частин з власною формою руху відносно основної частини (колеса, що обертаються та підстрибують, качаються руки в рухомого чоловічка, тощо). Збережіть компоненти малюнка у окремі файли у вигляді списків точок та ламаних ліній.

5. Змініть текст програми з минулих робіт для підтримки послідовного перетворення координат з метою організації відносності положення елементів.

6. За допомогою додаткового лічильника кадрів організуйте рух вашого елементу малюнка.

7. Створіть композицію з намальованих елементів, щоб малюнок виглядав цілісним.

8. Результат роздрукуйте та додайте до звіту разом з текстом програми.

9. Дайте відповіді на контрольні питання.

10. В разі виконання роботи на поточній парі дозволяється використання електронного звіту з усними відповідями на контрольні питання.

11. Зробіть висновки що до досяжності мети поставленої в лабораторній роботі.

Контрольні питання:

1. Яка послідовність перетворення координат потрібна якщо навколо об’єкту, що обертається, обертається інший об’єкт?

2. Чи можна за розробленою схемою використати один файл для збереження багатокомпонентного малюнка?

3. Як змінити програму для малювання фарбованих зображень?

4. Як знайти внутрішню точку для опуклого багатокутника?

5. Чи буде корисним використання зміни послідовності застосувань перетворень координат для малювання?

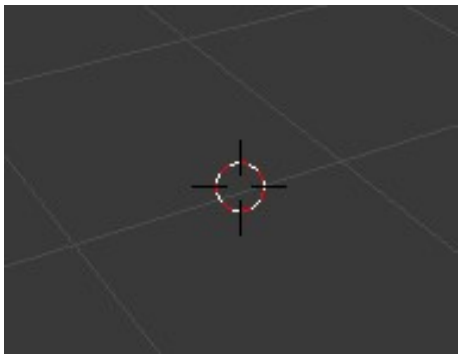
Лабораторна робота №6

Тема: Тривимірні координати. Тривимірне векторне зображення

Мета: Реалізувати каркасне зображення тривимірних об’єктів.

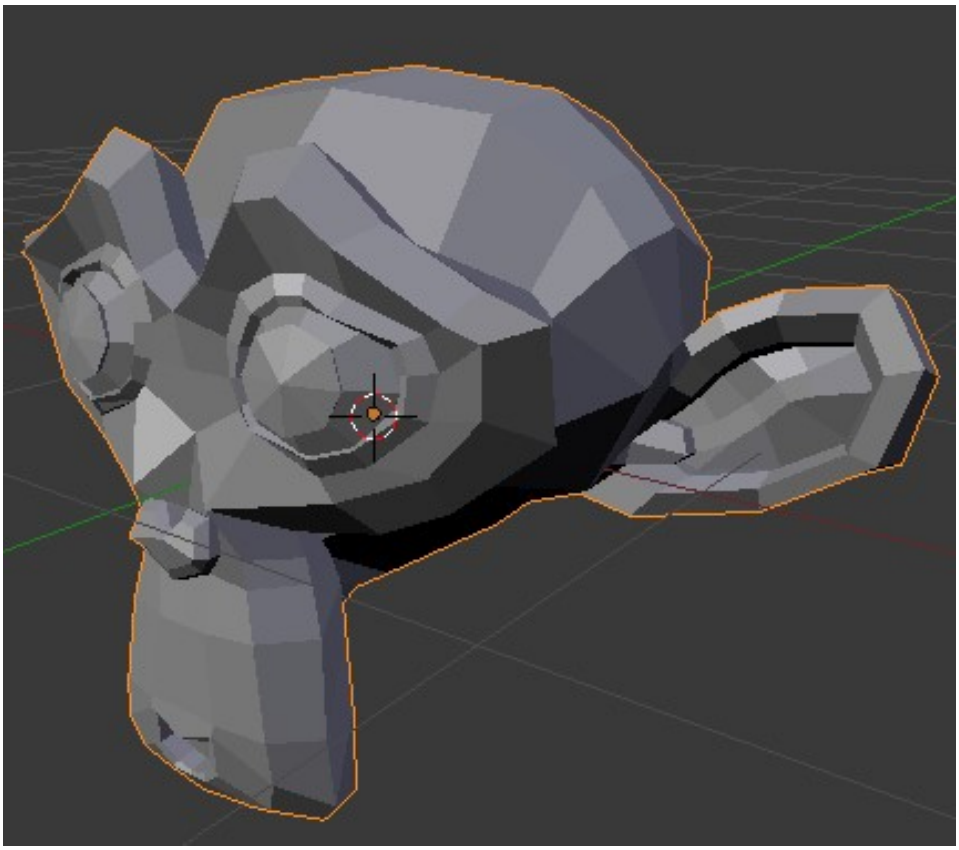
Теоретичні відомості

Заздалегідь визначені об'єкти в 3D програмах називаються примітивами. Деякі з них представляють собою прості геометричні об'єкти. А деякі можуть бути подарунком для програмістів готових експериментувати з програмою, але не займаючись моделюванням. Перемкніть вікно 3D-вигляду на вигляд зверху (Numpad 7) і клацніть лівою кнопкою миші в центрі вікна. Це перемістить 3D-курсор в місце розташування курсора миші.



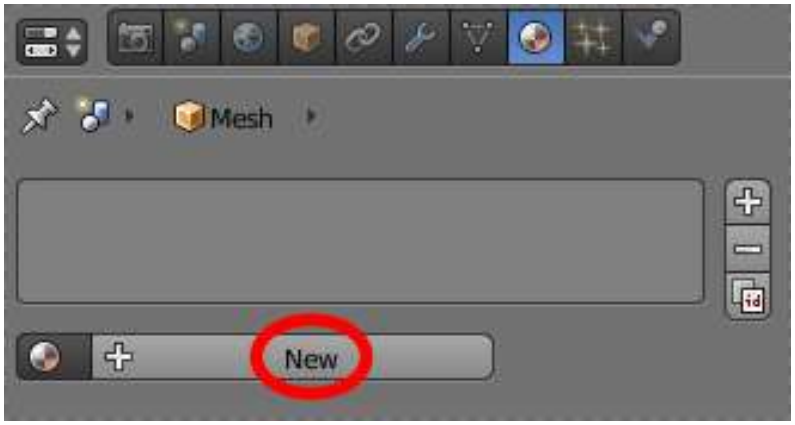
Це важливо, тому що при створенні нового об'єкта Blender розмістить його саме в тому місці сцени, де розташований 3D-курсор. Саме час перейти до меню Add / Mesh / Monkey.

Спробуйте покрутити і переміщувати навколо сцени. Отримуйте задоволення від процесу:)



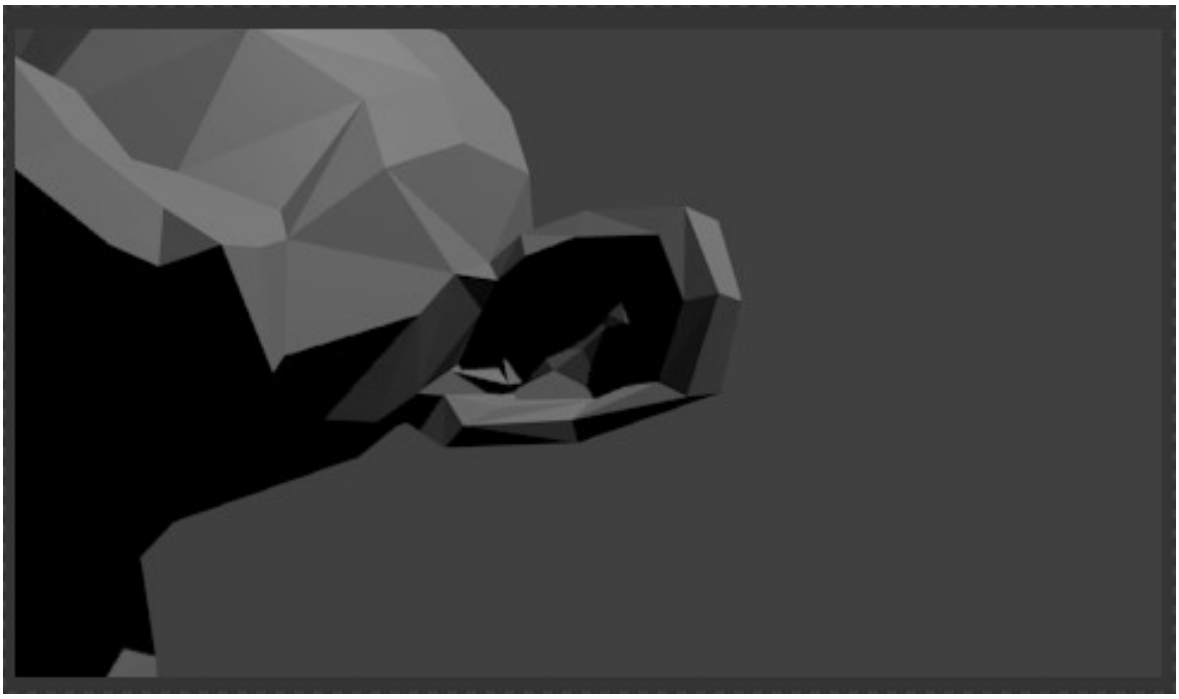
Цей об'єкт є талісман програми і називається Suzanne. Стривайте, час бігти і дзвонити своїм друзям з розповіддю про те, які чудові речі ви вмієте робити, ще не настав (не турбуйтеся, ви зробите це трохи пізніше).

Ви напевно захочете змінити колір матеріалу, як ми це робили з кубом. Але не лякайтеся, якщо при переході до розділу матеріалів ви побачите таке



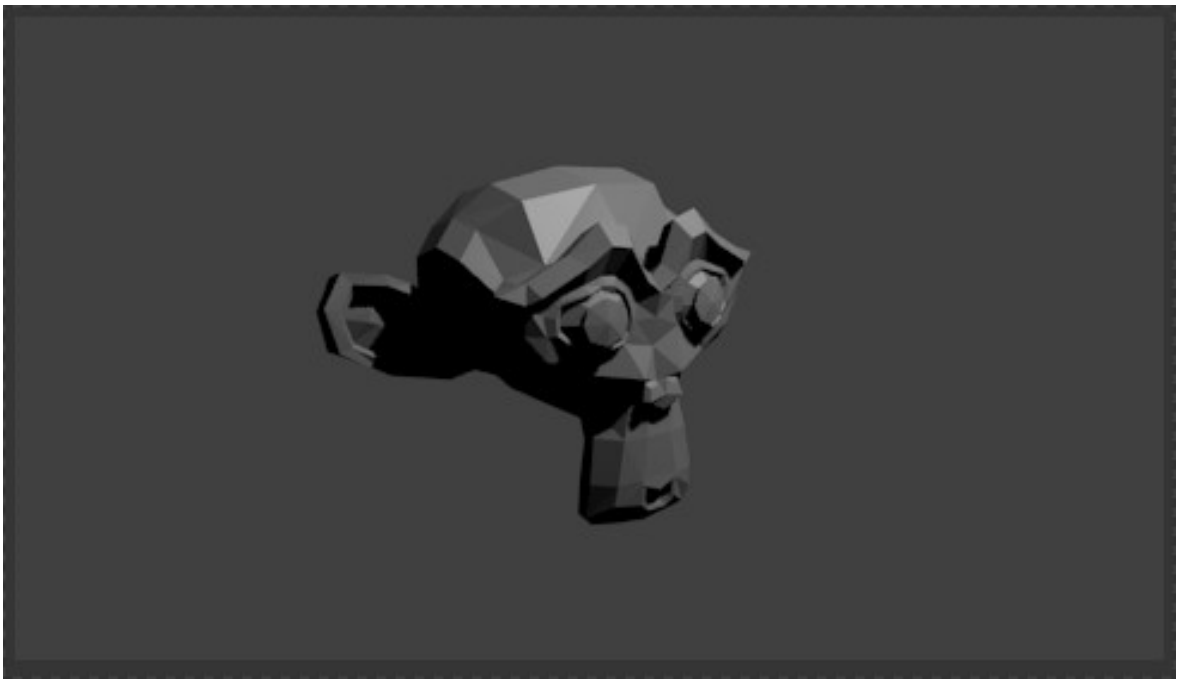
просто натисніть на кнопку New і перед вами відкриється весь набір кнопок, які були при роботі з кубом.

Не пропускайте цей етап, це задоволення за яке не потрібно платити. Не турбуйтеся якщо у вас виходить щось таке:



Погана постановка світла та камери привели до не зовсім зрозумілого результату рендерингу. Це трапилося через видалення попереднього об'єкта і додавання Suzanne без перевірки положення камери і світла відносно нього.

Для виправлення ситуації нам потрібно перемістити і повернути Suzanne за допомогою 3D-віджетів Модифікації (наприклад, з вигляду з камери «Numpad 0»). Постарайтеся домогтися гарного вигляду в камері і правильного падіння світла на мавпочку. Має вийти щось на зразок цього:



Правильніше буде повернути камеру. Можна повертати камеру вручну, але є 2 чудових способи. Спочатку просто поверніть вигляд за допомогою коліщатка миші так, як вам подобається, потім просто натисніть ctrl + alt + num 0 - активна камера переміститься так, щоб відобразити саме той ракурс який ви обрали. Але можливо, деякі деталі опиняться поза прямокутником рендера. Врятувати ситуацію допоможе Shift + f. Якщо натиснути його з камери, ви перейдете в режим польоту: керуючи мишкою та клавіатурою, можна плавно переміщати камеру до тих пір, поки ви не натиснете ЛКМ, щоб застосувати нове положення камери і ПКМ, щоб скинути переміщення.

Хід роботи

1. Створіть копію проекту попередньої роботи та збережіть новий проект в середовищі Lazarus.

2. Створіть довільний тривимірний малюнок. Малюнок повинен бути унікальним для кожного виконавця роботи. Малюнок повинен складатися з кількох ієрархічно пов’язаних частин з власною формою руху відносно основної частини (колеса, що обертаються та підстрибують, качаються руки в рухомого чоловічка, тощо). Збережіть компоненти малюнка у окремі файли у вигляді списків точок та ламаних ліній.

3. Змініть текст програми з минулої роботи для підтримки послідовного матричного перетворення координат з розмірністю матриці 4x4.

4. Додайте перетворення матриці для обертання по осям OX, OY.

5. Додайте глобальну матрицю перетворення-камеру з паралельною проекцією. До функції малювання відрізку додайте необхідні перетворення.

6. За допомогою додаткового лічильника кадрів організуйте рух камери навколо малюнка.

7. Створіть рух окремих елементів композиції сцени (мінімум 3 об’єкти).

8. Результат роздрукуйте та додайте до звіту разом з текстом програми.

9. Дайте відповіді на контрольні питання.

10. В разі виконання роботи на поточній парі дозволяється використання електронного звіту з усними відповідями на контрольні питання.

11. Зробіть висновки що до досяжності мети поставленої в лабораторній роботі.

Контрольні питання:

1. Яка послідовність перетворення координат потрібна якщо навколо об’єкту, що обертається, обертається інший об’єкт?

2. Чи можна за розробленою схемою використати один файл для збереження багатокомпонентного малюнка?

3. Як змінити програму для малювання фарбованих зображень?

4. Як знайти внутрішню точку для опуклого багатокутника?

5. Чи буде корисним використання зміни послідовності застосувань перетворень координат для малювання?

Лабораторна робота №7

Тема: Апаратне прискорення растеризації тривимірного зображення OpenGL

Мета: Реалізувати каркасне тривимірних об’єктів за допомогою апаратного прискорювача OpenGL.

Теоретичні відомості

Якщо ви ще не встановили GLScene на свій Delphi або Lazarus, то спочатку встановіть його. Якщо GLScene коректно встановлений, то виконаємо дії:

1. Створюємо новий проект.

2. Подвійним клацанням по іконці компонента TGLScene  (на вкладці GLScene), додаємо GLScene1.

3. Подвійним клацанням по іконці компонента TGLSceneViewer  (на вкладці GLScene), додаємо GLSceneViewer1.

4. У об'єкта GLSceneViewer1 у властивості Align вибираємо параметр alClient, для того щоб відображається сцена розгорнулася в усі вікно.

5. Подвійним клацанням по іконці компонента TGLCadencer  (на вкладці компонентів GLScene), додаємо GLCadencer1. Він потрібен для синхронізації і обробки всіх об'єктів GLScene.

6. У об'єкта GLCadencer1 у властивості Scene вибираємо GLScene1 для того щоб GLCadencer1 функціонував в нашій сцені.

7. Подвійним клацанням по GLScene1 на нашій формі відкриваємо редактор сцени

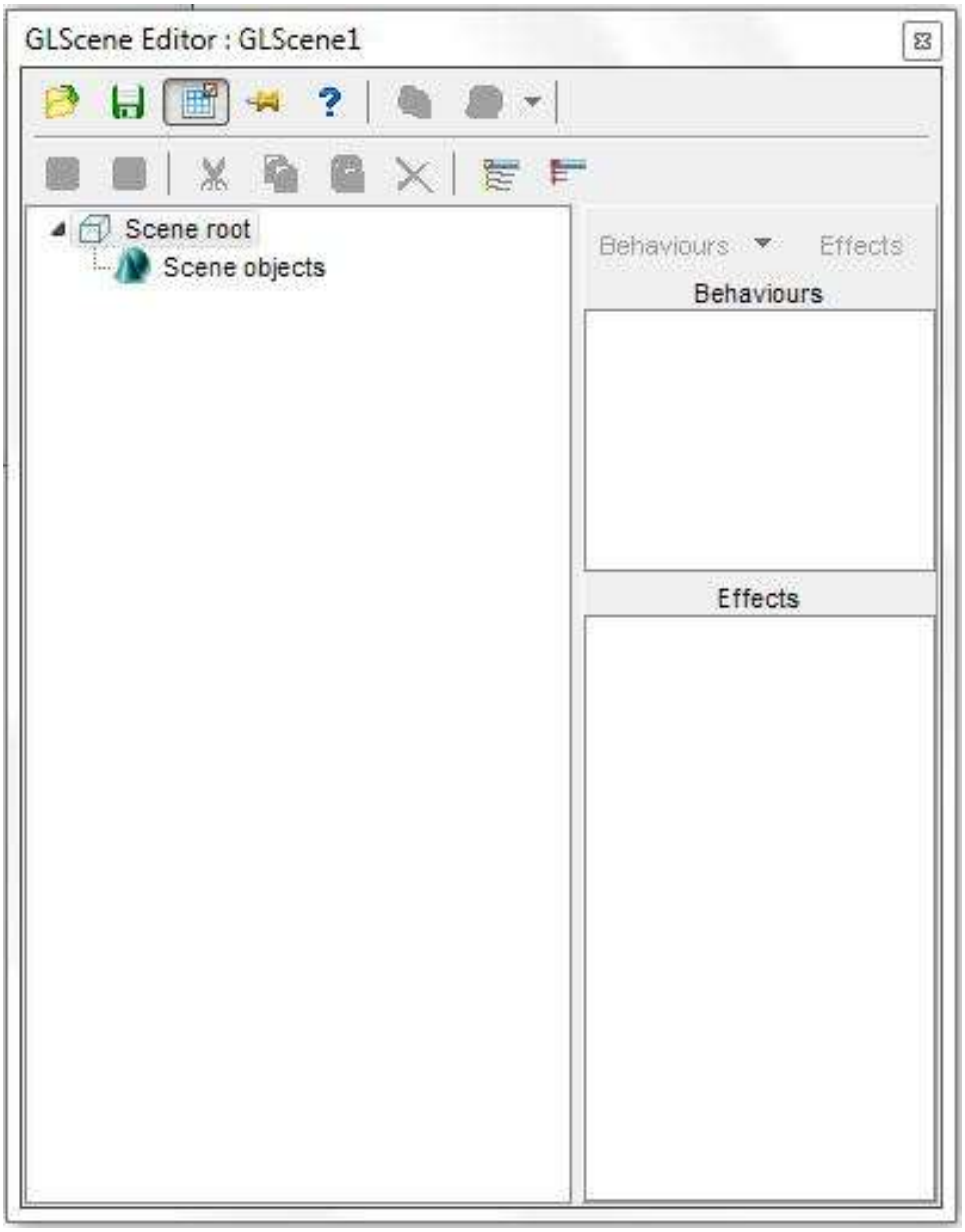


Рис. 1 — Редактор сцени

8. Тепер додамо камеру - правий клік на Scene objects в редакторі сцени, потім у випадаючому меню вибираємо Add Camera.

9. Вибираємо GLCamera1 і встановлюємо її параметри положення щодо центру сцени: Position.X: = 3, Position.Y: = 3, Position.Z: = 3.

10. Правим кліком по Scene objects в редакторі сцени - Add object - LightSource - додаємо на сцену джерело світла.

11. Вибираємо GLLightSource1 (створений раніше в пункті 10 джерело світла) і встановлюємо його параметр Position.Z: = 10 у властивостях для того щоб джерело світла був трохи віддалений від об'єкту.

12. Правим кліком по Scene objects в редакторі сцени - Add object - Basic Geometry - Cube - додаємо в середину сцени (0;0; 0) куб.

13. Вибираємо GLCamera1 (камера) і встановлюємо в її властивості TargetObject:= GLCube1, щоб наша камера була спрямована на куб.

14. Вибираємо GLSceneViewer1 і встановлюємо їй властивість Camera:= GLCamera1, щоб у вікні відображалося те, що «бачить» GLCamera1.

15. Створюємо подія Progress для GLCadencer1.

16. Додаємо в подія рядок коду:

GLCube1.TurnAngle: = GLCube1.TurnAngle + deltaTime * 100;

Цей рядок відповідає за поворот кубика навколо осі Z.

17. Тепер запустіть додаток, щоб побачити ваш перший тривимірний обертається куб.

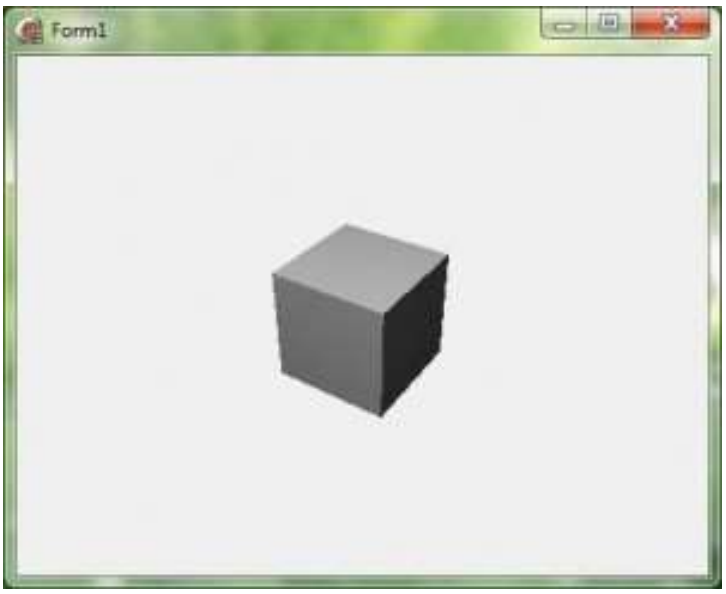


Рис. 2. - Отримане зображення куба

І все-таки обертається куб - досить тривіальний приклад. І це дійсно так з GLScene, але якщо розглянути те, що потрібно, щоб малювати як куб

обертається без GLScene, то відразу стане ясно, яку роботу зробили за нас розробники цієї бібліотеки. За прикладом також варто звернути увагу, що об’єкт (куб) затінюється щодо джерела світла.

Тепер давайте поставимо перед собою завдання: нехай поруч з кубиком треба розмістити 25 сфер, по 5 × 5 і зафарбувати їх випадковим кольором. Створимо глобальний двовимірний масив

MySphere: array [1..5,1..5] of TGLSphere;

Тепер додамо обробник події procedure TForm1.FormCreate (Sender: Tobject); і наведемо його до наступного вигляду:

```
procedure TForm1.FormCreate ( Sender TObject );
var
i, j: integer;
begin
randomize;
for i: = 1 to 5 do
begin
for j: = 1 to 5 do
begin
MySphere[i,j]   :=   TGLSphere   (   GLCube1.AddNewChild(
TGLSphere ) );
// створюємо нову сферу, як дочірній об’єкт
// елементу для GLCube1
```

```
with MySphere[i,j] do
begin
Position.X := -i; // позиція по осі x
Position.Z := -j; // позиція по осі y
material.FrontProperties.Diffuse.RandomColor;
// задаємо випадкову заливку
material.FrontProperties.Ambient.RandomColor;
// задаємо випадковий відтінок затінення на самому об'єкті
// щодо джерела світла
end;
end;
end;
end;
```

Ще один приклад демонструє завантаження об'єкта з файлу 3ds і його текстурізацію. Для роботи з текстурами, потрібно додати компонент TGLMaterialLibrary. Він знаходиться на вкладці GLScene. Створіть його. У свою властивість 'Materials' цей компонент зберігає масив матеріалів. У свою чергу кожен матеріал є сукупність текстур і різних параметрів матеріалу. Тому цей компонент і називається 'Бібліотека матеріалів'. Змінимо обробник події OnForm1Create на наведений нижче:

```
procedure TForm1.FormCreate(Sender: Tobject);
var
i, j: integer;
begin
// читаємо текстури з файлів
```

```
// tex1,tex2,tex3 — імена матеріалів в GLMaterialLibrary1
GLMaterialLibrary1.AddTextureMaterial('tex1','tex1.jpg',true);
GLMaterialLibrary1.AddTextureMaterial('tex2','tex2.jpg',true);
GLMaterialLibrary1.AddTextureMaterial('tex3','tex3.jpg',true);
for i := 1 to 5 do
begin
for j := 1 to 15 do
begin
// створюємо новий об’єкт TGLFreeForm
MyFreeForm[i,j]:=TGLFreeForm(GLCube1.AddNewChild(TGLFreeForm));
with MyFreeForm[i, j] do
begin
// задаємо нові координати об’єкту
Position.X := i-3;
Position.Z := -j+7;
Position.y := sin(j);
// завантажуємо меш об’єкту з файлу
LoadFromFile('obj.3ds');
// обираємо бібліотеку матеріалів для об’єкту
material.MaterialLibrary:=GLMaterialLibrary1;
// вказуємо для об’єкту одну з текстур
if (i=1)or(j=1)or(i=5)or(j mod 5=0) then
Material.LibMaterialName:='tex2'
else if (i=3)then Material.LibMaterialName:='tex3'
else Material.LibMaterialName:='tex1';
```

```
// повертаємо об’єкт
pitch(90);
// змінемо розміри об’єкту
Scale.X:=0.4;
Scale.y:=0.4;
Scale.z:=0.7;
end;
end;
end;
end;
```

Хід роботи

1. В графічному редакторі (рекомендується вільний до використання Blender) створіть декілька елементів графічної сцени.

2. За вказівками створіть програму для зображення створеної сцени. Елементи сцени зробіть завантаженими з файлів типу *.3ds.

3. Створіть ієрархічний об’єкт та організуйте його лінійне переміщення по сцені.

4. Для залежного елементу рухомого об’єкту створіть окреме незначне але добре помітне періодичне переміщення.

5. Прив’яжіть положення камери до рухомого об’єкту так, щоб він завжди був в центрі зображення сцени.

6. Організуйте коловий рух камери навколо рухомого об’єкту.

Контрольні питання:

1. Дайте означення поняття мешу.
2. Яке призначення компоненту TGLScene?

3. Яке призначення компоненту TGLCadencer?
4. Яке призначення компоненту TGLSceneViewer?
5. Яке призначення компоненту TGLCamera?
6. Яке призначення компоненту TGLLightSource?
7. Яке призначення компоненту TGLFreeForm?
8. Яке призначення компоненту TGLMaterial?

Зміст

Список літератури

1. А. С. Василюк, Н. І. Мельникова. Комп'ютерна графіка: навч. посіб. для студентів напряму підгот. 6.040303 «Систем. аналіз». — Львів: Вид-во Львів. політехніки, 2016. — 308 с. : іл. — Бібліогр.: с. 305—306 (23 назви). — ISBN 978-617-607-882-1

2. Веселовська Г. В. Комп'ютерна графіка: Навчальний посібник для вузів. — Херсон: ОЛДІ-плюс, 2004. — 582 с.

3. Компьютерная графика. / С. В. Глушаков, А. В. Капитанчук, Е. В. Вещев, Г. А. Кнабе . — 3-е издание, дополненное и перераб.. — Х.: Фолио, 2006. — 511 с.

4. Основи комп'ютерної графіки: курс лекцій / О. Я. Різник ; М-во освіти і науки, молоді та спорту України, Нац. ун-т «Львів. політехніка». — Л. : Вид-во Львів. політехніки, 2012. — 220 с. : іл. — Бібліогр.: с. 213—214 (24 назви). — ISBN 978-617-607-351-2

5. Петров М. Н. Компьютерная графика: Учебник для вузов. — СПб.; М.; Х.; Минск: Питер, 2003. — 736 с.