Лекція №4

Тема: Ієрархічна система побудови графічної сцени

Мета: Створити складене векторне зображення з залежними рухомими елементами.

ПЛАН

1. Складені об’єкти

2. Послідовні перетворення координат

3. Порядок перетворення координат

4. Сцена з залежних об’єктів: модель Сонце-Земля-Місяць

1. Складені об’єкти

На минулих заняттях було продемонстровано реалізацію виведення на екран плаского векторного зображення, що задане у файлі списком координат точок та індексами полігонів. З причини нерухомості складових об’єкта масив точок та ліній зручно асоціювати з окремою трансформацією координат. Якщо частина об’єкту або інший об’єкт повинен рухатись відносно іншого, то траєкторії руху програміст повинен враховувати окремо. Наприклад, зображення танку з антеною, яка обертається. Потрібно для зображення танку враховувати рух в часі, для антени враховувати рух танку, відносне зміщення на місце антени, перетворення обертання для зображення її руху. Досить важко пересувати об’єкт з усіма його складовими. Однак, об’єкт досить зручно представляти частинами.

Розглянемо два об’єкти. 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 $0 \le t \le 1$.

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 $0 \le t \le 1$.

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.

За останньою формулою відбувається зміщення центру обертання, навколо якого проходить обертання. Для руху в обидві сторони, в циклі потрібно змінювати t від 0 до 1 й від 1 до 0 циклічно. Це можна забезпечити за допомогою виразу $t := (1 + \cos(2\pi T))/2$, де T є час, що безперервно зростає. В результаті комбінації, можна отримати рухи зображені на графіку:

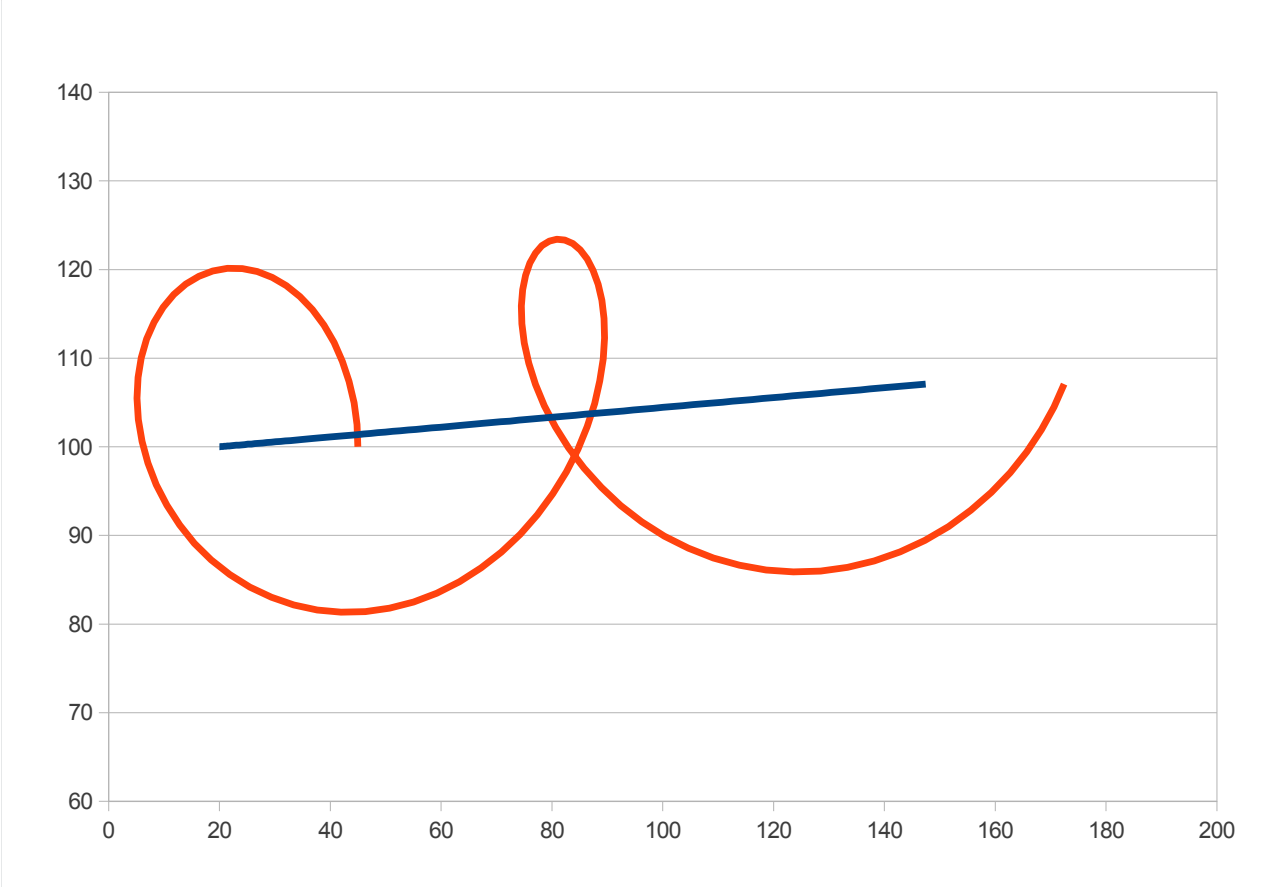


Рис. 1 — Суміщена траєкторія руху

Відповідно, при складеному об’єкті з багатьох рухомих частин, анімація перетворюється на створення параметричних складних кривих, що є хоч і не складною але громіздкою задачею.

2. Послідовні перетворення координат

Спростити ситуацію по перетворенню координат складеного об’єкту може послідовне застосування перетворення координат. Розглянемо конкретний складений об’єкт з двох восьмикутників та двох менших за розміром квадратів (рис. 2).

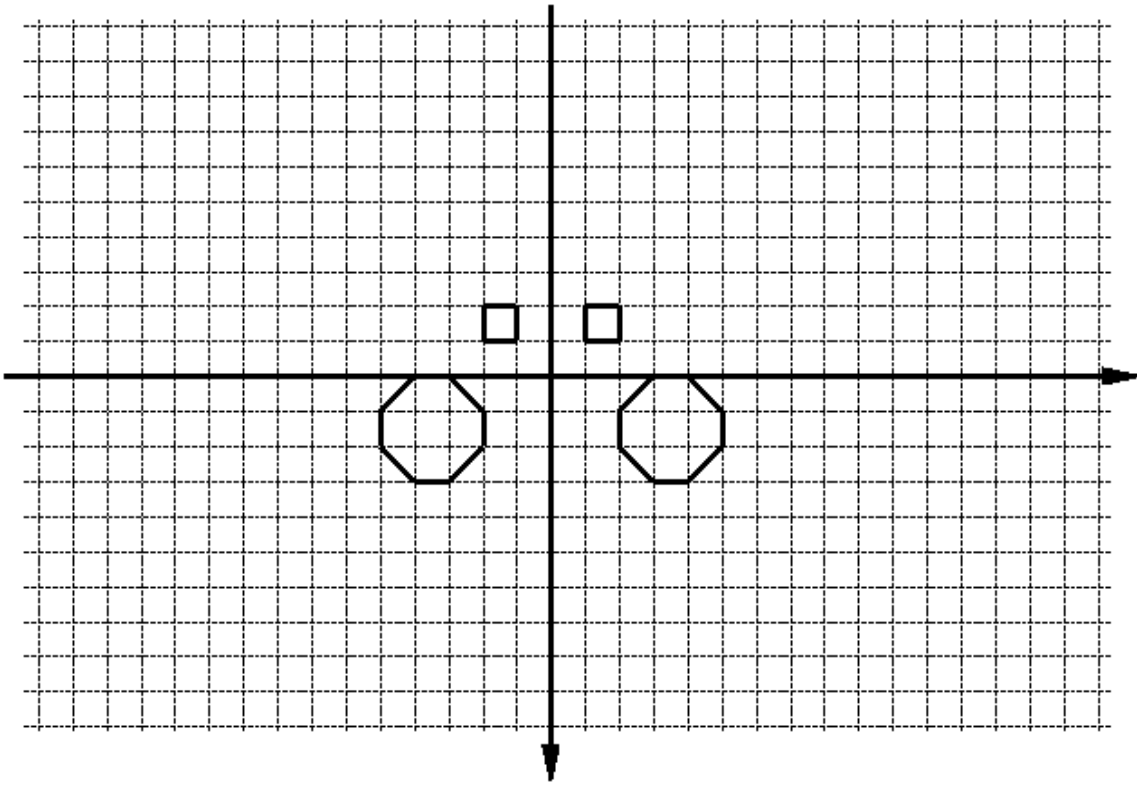


Рис. 2 — Складений об’єкт

Для того, щоб фігура рухалася як єдине ціле, можна для кожного з складових об’єкту змінювати параметри перетворення координат одночасно. Якщо визначено для кожної складової перетворення координат `Tr1, Tr2, Tr3, Tr4: TTransformer;` (дивіться лекцію 2), то потрібно проводити перетворення для кожного класу. Перед малюванням власне компоненту зображення з власними налаштуваннями перетворення, є необхідним активізувати його, зробивши привласнення перетворення до глобального, наприклад `Tra:=Tr1; DrawStraus;`. Звісно, більш зручно було б “прив’язати” залежні елементи об’єкту до основного і при переміщенні першого, отримати аналогічне переміщення інших об’єктів. Це можна досягти перетворенням координат залежних об’єктів, а потім застосувати до отриманих координат перетворення основного об’єкту. Таким чином ми матимемо ієрархічну структуру: елементи мають власний перетворювач координат, який прив’язаний до

основного елементу-початку залежної системи координат. В такому випадку, для пересування малюку в цілому, достатньо перемістити лише основний об’єкт.

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;
```

Як видно з наведеного коду, в класі векторного об’єкту з’явився список, що містить посилання на залежні об’єкти, а також містить трансформатори координат: власний `Tranform` для відносного переміщення, та загальний `activeTransform` для отримання глобального положення. При цьому `activeTransform` для кожного з об’єктів розраховується як комбінація перетворень з батьківських об’єктів та власного перетворення.

Також, на відміну від попередніх програм, процедура малювання почала входити до самого класу об’єкту. Ця процедура малювання використовує локальний перетворювач координат `activeTransform`, який може бути унікальним для кожного об’єкту.

Нехай маємо основний об’єкт А та залежний об’єкт Б. Для цих об’єктів рух розраховується на початку сцени як спільний. Тоді для об’єкту А, `Tranform.x` поступово змінюватиме своє значення. Для об’єкту Б `Tranform` є сталим. Але для об’єкту А `activeTransform` збігатиметься з `Transform`, коли для об’єкту Б`.activeTransform.x = А.activeTransform.x + Б.Transform.x;`.

Для малювання повної множини об’єктів утворюється алгоритм:

1) Об’єкт перед малюванням, всім залежним об’єктам, змінює на базі власного `activeTransform`, `activeTransform` залежних об’єктів.
2) Виконує власне малювання.
3) Викликає за списком процедуру малювання залежних об’єктів.

За цим алгоритмом батьківські об’єкти відповідають за актуальність значення параметрів перетворювачів координат `activeTransform` в залежних об’єктах:

```
n:=Child.Count;
for i:=0 to n-1 do begin
  t1 := TMyImage(Child.Items[i]).Tranform;
  TmyImage(Child.Items[i]).activeTransform.alpha:=
                                          t1.alpha+activeTransform.alpha;
  TMyImage(Child.Items[i]).activeTransform.mx:=t1.mx*activeTransform.mx;
  TMyImage(Child.Items[i]).activeTransform.my:=t1.my*activeTransform.my;
  TMyImage(Child.Items[i]).activeTransform.x:=t1.x+activeTransform.x;
  TMyImage(Child.Items[i]).activeTransform.y:=t1.y+activeTransform.y;
end;
```

З наведеного лістингу видно, для отримання комбінації перетворень необхідно виконати складання кутів повороту, коефіцієнт пропорційності є добутком базових коефіцієнтів та трансформація переносу є сумами переносів по горизонталі та вертикалі відповідно.

Для зображення об’єктів з рис. 2 нехай створено файли 8.txt та 4.txt. Для відтворення композиції використаємо читання зазначених форм з файлу:

```
A := TMyImage.Create;
B := TMyImage.Create;
C := TMyImage.Create;
D := TMyImage.Create;
A.LoadPicFromFile('8.txt');
B.LoadPicFromFile('8.txt');
C.LoadPicFromFile('4.txt');
D.LoadPicFromFile('4.txt');
A.AddChild(B); A.AddChild(C); A.AddChild(D);
B.Transform.SetXY(40,0);
C.Transform.SetXY(15,-10);
D.Transform.SetXY(30,-10);
A.activeTransform.SetXY(100,100); //Впливатиме на всі залежні об’єкти
```

3. Порядок перетворення координат

Припустимо використання трьох об’єктів, які ланцюгом залежать один від одного A→B→C. Для такої системи досить важливо правильно використовувати послідовність виконання об’єднання перетворень координат. Для ілюстрації проблеми, наведемо

конкретний випадок, нехай об’єкти A, B, C мають положення (50;50), (30;30), (15,15) відповідно, та кути повороту 45º, 0º, 90º. Кожна з функцій перетворення координат має вигляд:

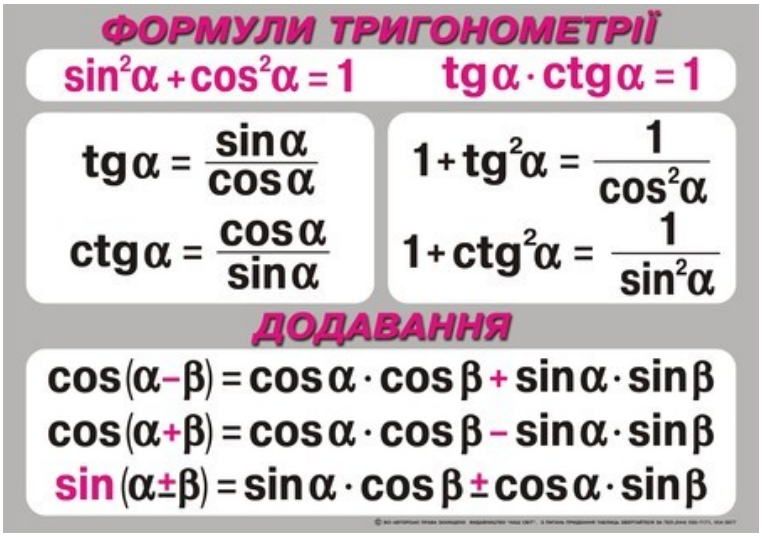
$$\begin{cases} x_1 := dx_0 + x_0\cos(\alpha) + y_0\sin(\alpha) \\ y_1 := dy_0 - x_0\sin(\alpha) + y_0\cos(\alpha) \end{cases}, \quad \begin{cases} x_2 := dx_1 + x_1\cos(\alpha) + y_1\sin(\alpha) \\ y_2 := dy_1 - x_1\sin(\alpha) + y_1\cos(\alpha) \end{cases}.$$

Комбінація цих перетворень є:

$$\begin{cases} x_2 := dx_1 + (dx_0 + x_0\cos(\alpha) + y_0\sin(\alpha))\cos(\beta) + (dy_0 - x_0\sin(\alpha) + y_0\cos(\alpha))\sin(\beta) \\ y_2 := dy_1 - (dx_0 + x_0\cos(\alpha) + y_0\sin(\alpha))\sin(\beta) + (dy_0 - x_0\sin(\alpha) + y_0\cos(\alpha))\cos(\beta) \end{cases} \rightarrow$$

$$\begin{cases} x_2 := dx_1 + dx_0\cos(\beta) + dy_0\sin(\beta) + (x_0\cos(\alpha) + y_0\sin(\alpha))\cos(\beta) + (-x_0\sin(\alpha) + y_0\cos(\alpha))\sin(\beta) \\ y_2 := dy_1 - dx_0\sin(\beta) + dy_0\cos(\beta) - (x_0\cos(\alpha) + y_0\sin(\alpha))\sin(\beta) + (-x_0\sin(\alpha) + y_0\cos(\alpha))\cos(\beta) \end{cases} \rightarrow$$

$$\begin{cases} x_2 := dx_1 + (dx_0\cos(\beta) + dy_0\sin(\beta)) + x_0(\cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)) + y_0(\sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta)) \\ y_2 := dy_1 + (-dx_0\sin(\beta) + dy_0\cos(\beta)) - x_0(\cos(\alpha)\sin(\beta) + \sin(\alpha)\cos(\beta)) - y_0(\sin(\alpha)\sin(\beta) - \cos(\alpha)\sin(\beta)) \end{cases} \rightarrow$$



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

Використання перетворень в іншому порядку, призведе до отримання дещо іншої формули:

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

що призведе до іншого положення предмету. Тому послідовність перетворення координат має важливе значення.

4. Сцена з залежних об’єктів: модель Сонце-Земля-Місяць

Використаємо отримані знання для створення трирівневого руху “Сонце”-“Земля”-“Місяць”, коли рух Сонця впливає на рух Землі, яка в свою чергу впливає на рух Місяця. При цьому змусимо “Землю” та “Місяць” обертатися навколо осі з різними швидкостями. Для забезпечення руху кожного тіла розберемо їх рух окремо. Введемо глобальну змінну `frameCount: Integer`. Ця змінна буде збільшуватися на одиницю з кожним спрацюванням таймеру з малюванням кадру.

Початковими налаштуваннями для “Сонячної системи” буде:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Framecount:=0;  // початкова кількість показаних кадрів
  Sun := TMyImage.Create;
  Sun.LoadPicFromFile('sun.txt');
  Eath := TMyImage.Create;
  Eath.LoadPicFromFile('eath.txt');
  Eath.Tranform.SetXY(50,50);
  Eath.Tranform.SetMXY(0.5,0.5); //Вдвічі менше ніж “Сонце”
  Eath.Tranform.SetAlpha(PI*0.25);
  Sun.AddChild(Eath); //До “Сонця” приєднаємо “Землю”
  Moon := TMyImage.Create;
  Moon.LoadPicFromFile('moon.txt');
  Moon.Tranform.SetXY(50,50);
  Moon.Tranform.SetMXY(0.5,0.5); //Вдвічі менше ніж “Земля”
  Moon.Tranform.SetAlpha(0);
  Eath.AddChild(Moon); //До “Землі” приєднуємо “Місяць”
  Timer1.Enabled:=true;
end;
```

Процедура малювання містить комбінацію власного перетворення координат з приєднаними об’єктами, малювання власних ліній та виклику процедур малювання приєднаних об’єктів:

```
procedure TMyImage.DrawMyImage(Canvas: TCanvas);
var
  t,i,n: Integer;
  p1, p2: TMPoint;
  t1: TTransformer;
begin
  n:=Child.Count; //Визначення кількості приєднаних об’єктів
```

```
  for i:=0 to n-1 do begin
    t1 := TMyImage(Child.Items[i]).Tranform;
    TmyImage(Child.Items[i]).activeTransform.alpha:=t1.alpha+
                                        activeTransform.alpha;
    TMyImage(Child.Items[i]).activeTransform.mx:=t1.mx*activeTransform.mx;
    TMyImage(Child.Items[i]).activeTransform.my:=t1.my*activeTransform.my;
    TMyImage(Child.Items[i]).activeTransform.x:=t1.x+activeTransform.x;
    TMyImage(Child.Items[i]).activeTransform.y:=t1.y+activeTransform.y;
  end; //Перетворимо координати приєднаних об'єктів
  n:=Length(PolyLine); //Кількість індексів в лініях об'єктів
  t:=-1; //Значення попереднього індексу точки
  for i:=0 to n-1 do begin
    if (t>=0)and(PolyLine[i]>=0) then begin //Якщо один з індексів не -1
      p1:=Points[t];
      p2:=Points[PolyLine[i]];
      Canvas.Line(activeTransform.GetX(p1.x,p1.y),
                  activeTransform.GetY(p1.x,p1.y),
                  activeTransform.GetX(p2.x,p2.y),
                  activeTransform.GetY(p2.x,p2.y)  );
    end;
    t:=PolyLine[i]; //Отримуємо наступний індекс
  end;
  n:=Child.Count; //Визначення кількості приєднаних об'єктів
  for i:=0 to n-1 do begin //Малюємо приєднані об'єкти
    TmyImage(Child.Items[i]).DrawMyImage(Canvas);
  end;
end;
```

Залишається використати обробник події таймеру для перетахунку координат кожного з об'єктів:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Image1.Canvas.Brush.Color:=clWhite;
  Image1.Canvas.FillRect(0,0,Image1.Width,Image1.Height);
//Очистили фон білим
  Sun.activeTransform.x:=200+Round(100*sin(0.015*frameCount));
  Sun.activeTransform.y:=200; //"Сонце" переміщується лише по X
  Eath.Tranform.SetXY(100.0*cos(0.1154687*frameCount),
    100.0*sin(0.1154687*frameCount)); //"Земля" обертається по колу
  Eath.Tranform.alpha:=0.3215354*frameCount; //Обертаємо "Землю"
  Moon.Tranform.SetXY(50.0*cos(0.4154687*frameCount),
   50.0*sin(0.4154687*frameCount)); //"Місяць" обертається по
```

```
                                    //меншому радіусу
    Moon.Tranform.alpha:=0.4215354*frameCount; //"Місяць" теж обертається
    Sun.DrawMyImage(Image1.Canvas); //Малюємо "Сонце" всі приєднані об'єкти
                                    // малюються автоматично
    frameCount:=frameCount+1; //Намальований наступний кадр
end;
```

Також для малювання "космічних об'єктів" сформуємо зображення, які показано на рис. 3, 4, 5. В цих зображеннях використано внутрішні візерунки для унаочнення обертального руху. В результаті змін в програмному коді та сформованому зображенню, ми маємо можливість спостерігати взаємний рух декількох об'єктів. Один з фрагментів взаємного руху показано на рис. 6.

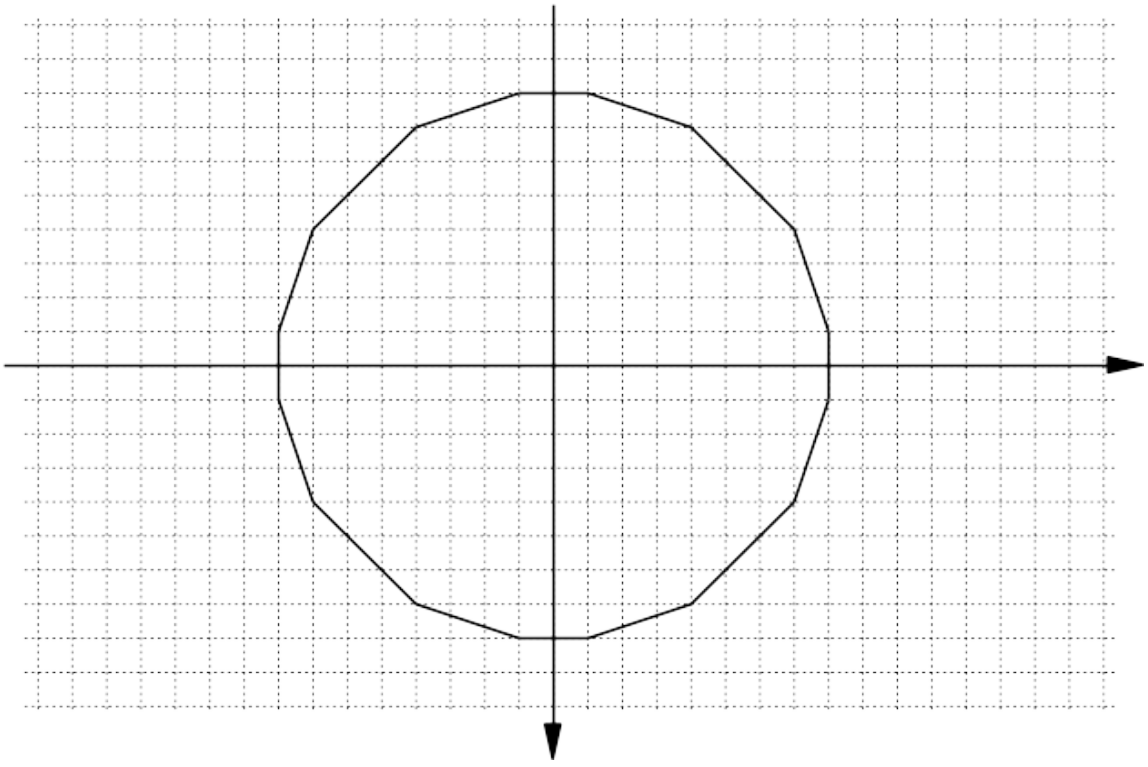


Рис. 3 — Зображення "Сонця"

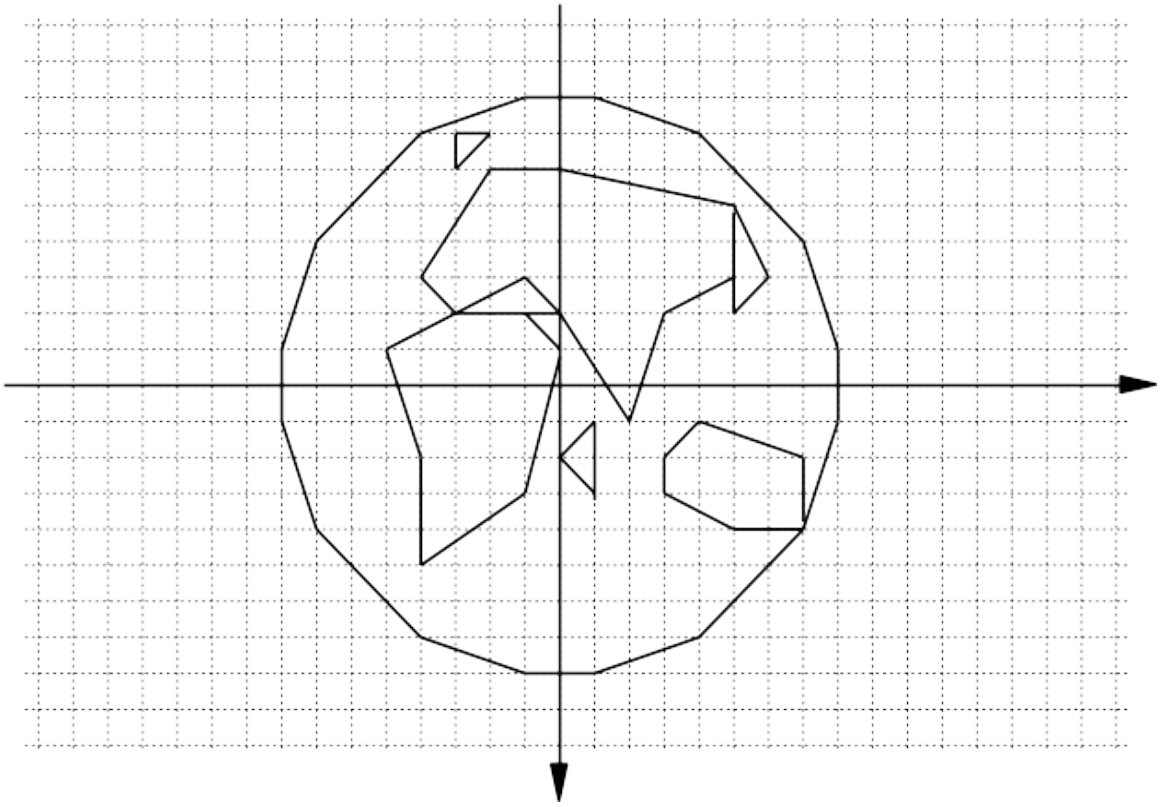


Рис. 4 — Зображення “Земля”

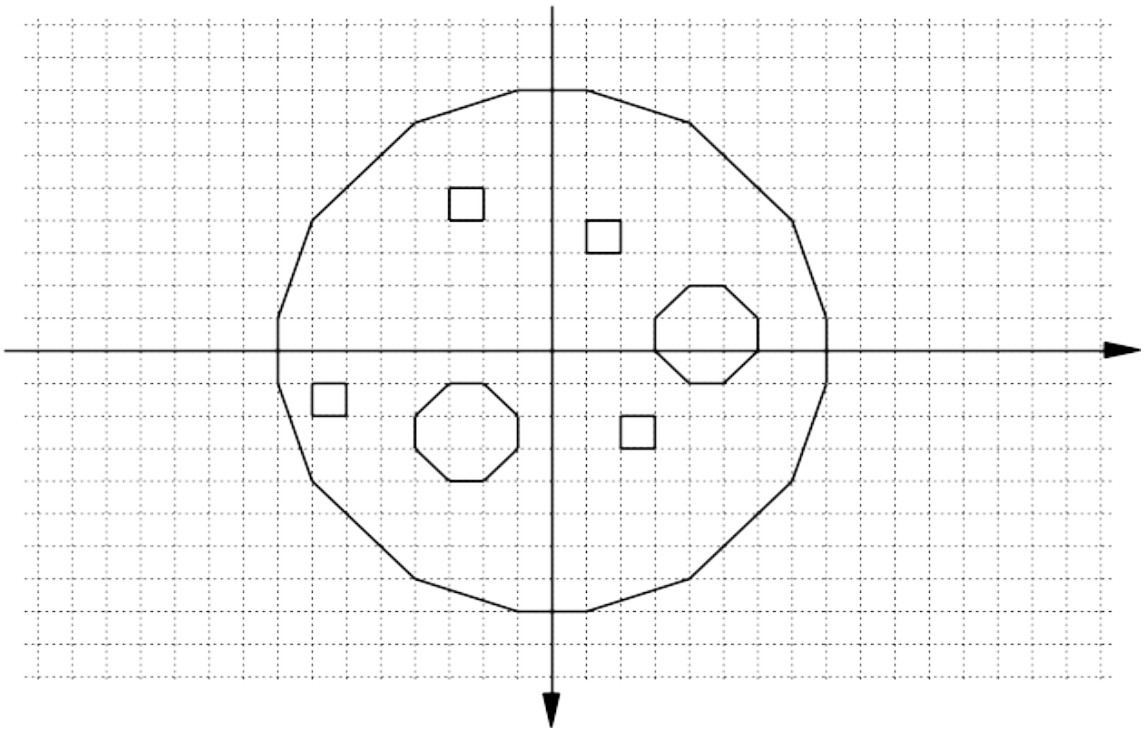


Рис. 5 — Зображення “Місяць”

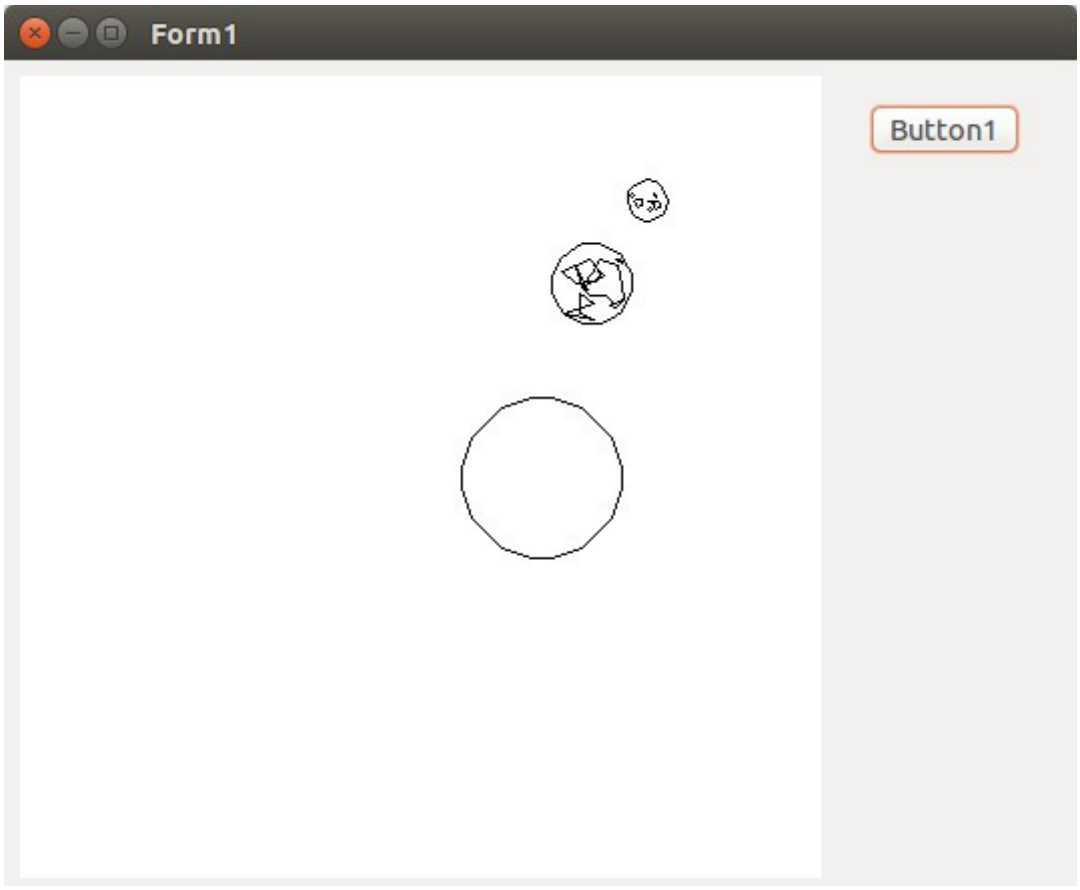


Рис. 6 — Взаємний рух трьох об’єктів

Додаток А. Текст файлу з зображенням “Сонця”, sun.txt

```
16
-40 -5
-35 -20
-20 -35
-5 -40
5 -40
20 -35
35 -20
40 -5
40 5
35 20
20 35
5 40
-5 40
-20 35
-35 20
-40 5
18
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 -1
```

Додаток Б. Зображення “Землі”, eath.txt

```
46
-40 -5
-35 -20
-20 -35
-5 -40
5 -40
20 -35
35 -20
40 -5
40 5
35 20
20 35
5 40
-5 40
-20 35
-35 20
-40 5
-15 -30
-15 -35
-10 -35
-10 -30
0 -30
25 -25
30 -15
25 -10
25 -15
15 -10
10 5
0 10
-5 -15
-15 -10
-20 -15
```

-25 -5

-15 -10

-5 -10

0 -5

-5 15

-20 25

-20 10

0 10

5 5

5 15

20 5

35 10

25 20

10 15

10 10

65

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 -1

16 17 18 16 -1

19 20 21 22 23 24 25 26 27 28 29 30 19 -1

31 29 32 33 34 35 36 31 -1

21 24 -1

27 32 -1

37 38 39 37 -1

40 41 10 42 43 44 40 -1

Додаток В. Зображення “Місяця”, moon.txt.

```
47
-40 -5
-35 -20
-20 -35
-5 -40
5 -40
20 -35
35 -20
40 -5
40 5
35 20
20 35
5 40
-5 40
-20 35
-35 20
-40 5
-35 10
-35 5
-30 5
-30 10
-15 -20
-15 -25
-10 -25
-10 -20
-20 10
-15 5
-10 5
-5 10
-5 15
-10 20
-15 20
```

-20 15

5 -15

5 -20

10 -20

10 -15

10 10

15 10

15 15

10 15

15 -5

20 -10

25 -10

30 0

25 5

20 5

15 0

56

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 -1

16 17 18 19 16 -1

24 25 26 27 28 29 30 31 24 -1

32 33 34 35 32 -1

36 37 38 39 36 -1

40 41 42 43 44 45 46 47 40 -1

Додаток Г. Текст програмного коду для виводу залежних об’єктів

```
unit Unit1;

{$mode objfpc}{$H+}

interface

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

type

  TMPoint = record
    x, y: Double;
  end;

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

  TMyImage = class
  public
    Tranform: TTransformer;
    activeTransform: TTransformer;
    Child: TList;
    Points: array of TMPoint;
    PolyLine: array of Integer;
    constructor Create;
    destructor Destroy; override;
    procedure LoadPicFromFile(FileName: String);
```

```
    procedure DrawMyImage(Canvas: TCanvas);
    procedure AddChild(newImage:TMyImage);
  end;

  { TForm1 }

  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 }
    Sun, Eath, Moon: TMyImage;
    frameCount: Integer;
  end;

var
  Form1: TForm1;

implementation

{$R *.lfm}

{ TForm1 }

procedure TForm1.Button1Click(Sender: TObject);
begin
  Timer1.Enabled:= not Timer1.Enabled;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  frameCount:=0;
  Sun := TMyImage.Create;
  Sun.LoadPicFromFile('sun.txt');
```

```
  Eath := TMyImage.Create;
  Eath.LoadPicFromFile('eath.txt');
  Eath.Tranform.SetXY(50,50);
  Eath.Tranform.SetMXY(0.5,0.5);
  Eath.Tranform.SetAlpha(PI*0.25);
  Sun.AddChild(Eath);

  Moon := TMyImage.Create;
  Moon.LoadPicFromFile('moon.txt');
  Moon.Tranform.SetXY(50,50);
  Moon.Tranform.SetMXY(0.5,0.5);
  Moon.Tranform.SetAlpha(0);
  Eath.AddChild(Moon);

  Timer1.Enabled:=true;
end;

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

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Image1.Canvas.Brush.Color:=clWhite;
  Image1.Canvas.FillRect(0,0,Image1.Width,Image1.Height);

  Sun.activeTransform.x:=200+Round(100*sin(0.015*frameCount));
  Sun.activeTransform.y:=200;
  Sun.DrawMyImage(Image1.Canvas);

  Eath.Tranform.SetXY(100.0*cos(0.1154687*frameCount),
                          100.0*sin(0.1154687*frameCount));
  Eath.Tranform.alpha:=0.3215354*frameCount;

  Moon.Tranform.SetXY(50.0*cos(0.4154687*frameCount),
                             50.0*sin(0.4154687*frameCount));
  Moon.Tranform.alpha:=0.4215354*frameCount;
  frameCount:=frameCount+1;
end;
```

```
{ TTransformer }

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

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

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

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

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

constructor TMyImage.Create;
begin
  Child := TList.Create;
  Tranform:=TTransformer.Create;
  activeTransform:=TTransformer.Create;
  Tranform.SetMXY(1.0,1.0);
  Tranform.SetAlpha(0.0);
  Tranform.SetXY(200,200);
  activeTransform.SetMXY(1.0,1.0);
  activeTransform.SetAlpha(0.0);
  activeTransform.SetXY(200,200);
```

```
end;

destructor TMyImage.Destroy;
var i, n: Integer;
begin
  Tranform.Destroy;
  activeTransform.Destroy;
  n:=Child.Count;
  for i:=0 to n-1 do begin
      TMyImage(Child.Items[i]).Destroy;
  end;
  Child.Clear;
end;

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

procedure TMyImage.DrawMyImage(Canvas: TCanvas);
var
  t,i,n: Integer;
  p1, p2: TMPoint;
  t1: TTransformer;
```

```
begin
  n:=Child.Count;
  for i:=0 to n-1 do begin
    t1 := TMyImage(Child.Items[i]).Tranform;
    //t2 := TMyImage(Child.Items[i]).activeTransform;
     TMyImage(Child.Items[i]).activeTransform.alpha:=t1.alpha+activeTransfor
m.alpha;
    TMyImage(Child.Items[i]).activeTransform.mx:=t1.mx*activeTransform.mx;
    TMyImage(Child.Items[i]).activeTransform.my:=t1.my*activeTransform.my;
    TMyImage(Child.Items[i]).activeTransform.x:=t1.x+activeTransform.x;
    TMyImage(Child.Items[i]).activeTransform.y:=t1.y+activeTransform.y;
  end;

  n:=Length(PolyLine);
  t:=-1;
  for i:=0 to n-1 do begin
    if (t>=0)and(PolyLine[i]>=0) then begin
      p1:=Points[t];
      p2:=Points[PolyLine[i]];
      Canvas.Line(activeTransform.GetX(p1.x,p1.y),
                  activeTransform.GetY(p1.x,p1.y),
                  activeTransform.GetX(p2.x,p2.y),
                  activeTransform.GetY(p2.x,p2.y)  );
    end;
    t:=PolyLine[i];
  end;

  n:=Child.Count;
  for i:=0 to n-1 do begin
    TMyImage(Child.Items[i]).DrawMyImage(Canvas);
  end;
end;

procedure TMyImage.AddChild(newImage:TMyImage);
begin
  Child.Add(newImage);
end;

end.
```