лекция №4

Тема: Иерархическая система построения графической сцены

цель: Создать составленное векторное изображение с зависимыми подвижными элементами.

ПЛАН

1. Составные объекты

2. Последовательные преобразования координат

3. Порядок преобразования координат

4. Сцена из зависимых объектов: модель Солнце-Земля-Луна

1. Составные объекты

На прошлых занятиях было продемонстрировано реализацию вывода на экран плоского векторного изображения, заданное в файле списком координат точек и индексами полигонов. В виду недвижимости составляющих объекта массив точек и линий удобно сравнивать с отдельной трансформацией координат. Если часть объекта или другой объект должен двигаться относительно другого, то траектории движения программист должен учитывать отдельно. Например, изображение танка с антенной, которая вращается. Нужно для изображения танке учитывать движение во времени, для антенны учитывать движение танка, относительное смещение на место антенны, преобразования вращения для изображения ее движения. Довольно трудно передвигать объект со всеми его составляющими. Однако, объект достаточно удобно представлять частями.

Рассмотрим два объекта. Первый объект пусть движется постепенно вдоль оси OX, а второй - движется вокруг первого по кругу. Запишем преобразования координат для движения вдоль отрезка:

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

Соответственно, для движения по кругу тоже можно использовать параметрический запись:

$$\begin{cases} 0 \le t \le 1. \\ x = x_0 + r_x \cos(2\pi \cdot t),\ Y = y_0 - r_y \sin(2\pi \cdot t), \end{cases}, \text{ где}$$

Но для того чтобы второй объект передвигался вокруг первого, нужно совместить передвижения по прямой и по кругу:

$$\begin{cases} y \sin(2\pi nt), \\ X = x_0 + (x_1 - x_0) \cdot t + r_x \cos(2\pi nt), \; Y = y_0 - (y_1 - y_0) \cdot t - \end{cases}$$

где 0≤t≤1, n - число оборотов на передвижение вдоль отрезка.

По крайней формуле происходит смещение центра вращения, вокруг которого проходит вращения. Для движения в обе стороны, в цикле нужно менять t от 0 до 1 и от 1 до 0 циклически. Это можно обеспечить с помощью **выражения** $t = (1 + \cos(2\pi T))/2$ **, Где T есть время, непрерывно растет. В результате комбинации, можно получить** движения изображены на графике:

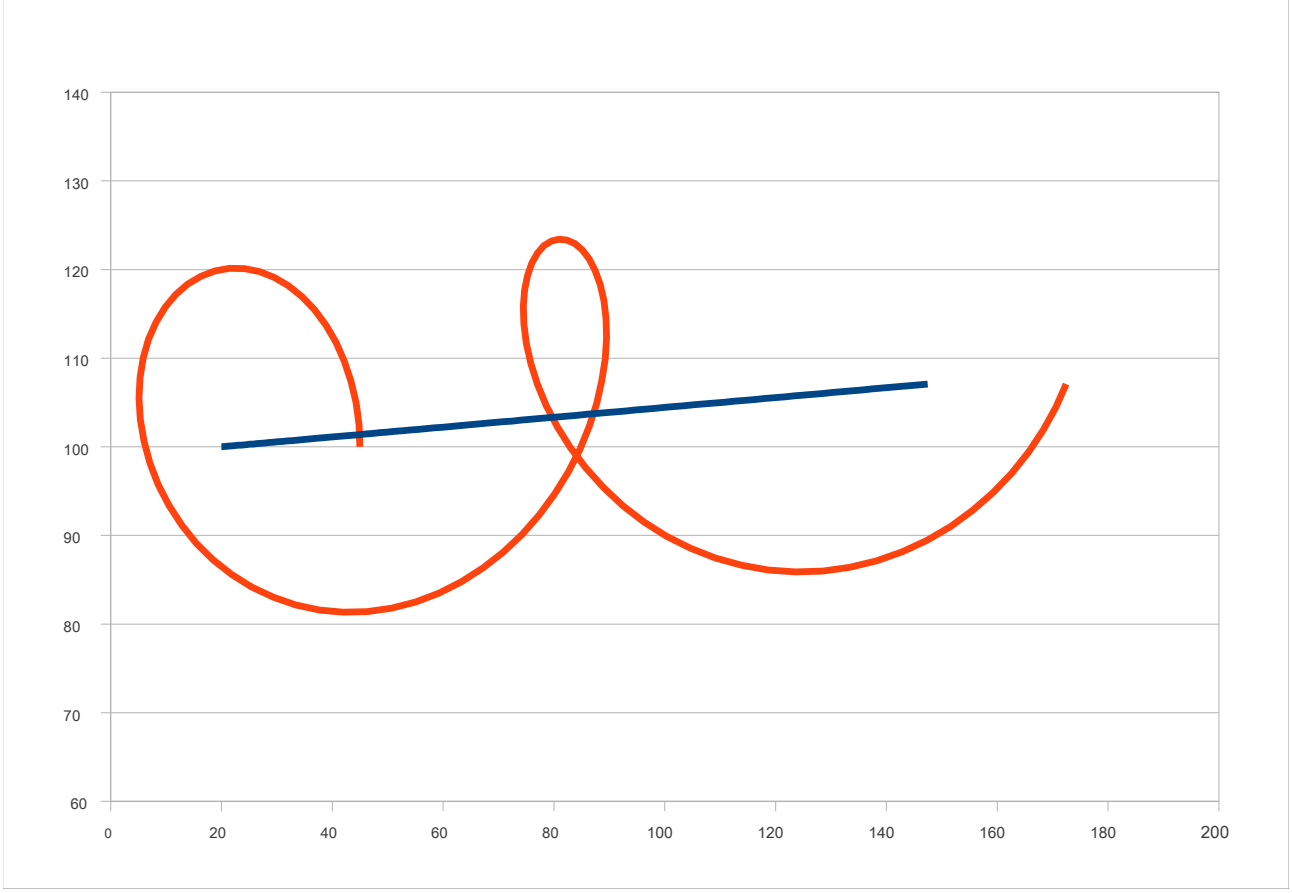


Рис. 1 - Совмещенная траектория движения

Соответственно, при сложенном объекте из многих подвижных частей, анимация превращается в создание параметрических сложных кривых, хоть и не сложной но громоздкой задачей.

2. Последовательные преобразования координат

Упростить ситуацию по превращению координат составленного объекта может последовательное применение преобразования координат. Рассмотрим конкретный составной объект из двух восьмиугольников и двух меньших по размеру квадратов (рис. 2).

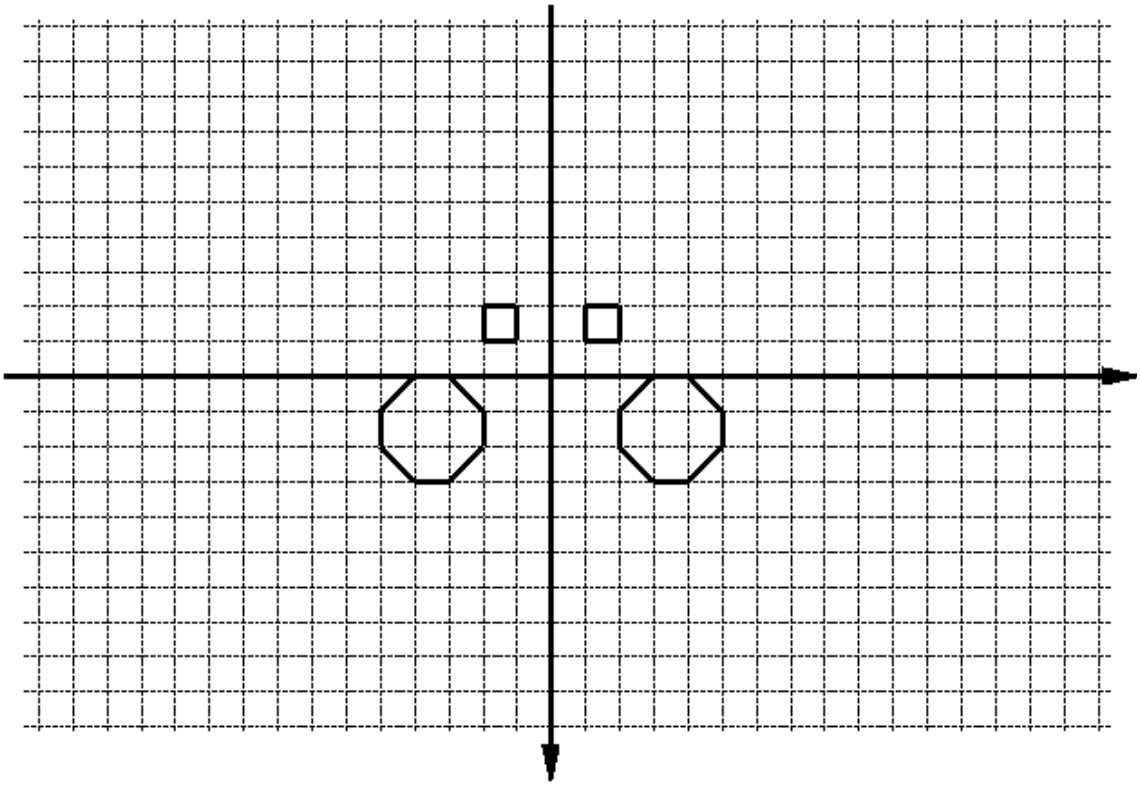


Рис. 2 - Составной объект

Для того, чтобы фигура двигалась как единое целое, можно для каждого из составляющих объекта изменять параметры преобразования координат одновременно. Если определено для каждой составляющей преобразования координат Tr1, Tr2, Tr3, Tr4: TTransformer; (смотрите лекцию 2), то нужно проводить преобразования для каждого класса. Перед рисованием собственно компонента изображения с собственными настройками преобразования, необходимо активизировать его, сделав присвоение преобразования к глобальному, например

Tra = Tr1; DrawStraus ;. Конечно, более удобно было бы "привязать" зависимые элементы объекта к основному и при перемещении первого, получить аналогичное перемещение других объектов. Это можно достичь преобразованием координат зависимых объектов, а затем применить к полученным координат преобразования основного объекта. Таким образом мы получим иерархическую структуру: элементы имеют собственный преобразователь координат, который привязан к

основного элемента начале зависимой системы координат. В таком случае, для передвижения малышу в целом, достаточно переместить только основной объект.

Это можно достичь изменением класса векторных объектов добавлением списка зависимых от него объектов:

```
TMyImage = class public

    Tranform: TTransformer; // относительное преобразования
    activeTransform: TTransformer; // активное преобразование
    Child: TList; // Список зависимых элементов
    Points: array of TMPoint; // Список координат точек
    PolyLine: array of Integer; // Список индексов точек
    constructor Create;
    destructor Destroy; override; procedure LoadPicFromFile (FileName: String) procedure DrawMyImage (Canvas:
    TCanvas) // Рисовать на указанной канве

    procedure AddChild (newImage: TMyImage)  // Добавить зависимый объект
end;
```

Как видно из приведенного кода, в классе векторного объекта появился список, содержащий ссылки на зависимые объекты, а также содержит трансформаторы координат: собственный Tranform для относительного перемещения, и общий activeTransform для получения глобального положения. при этом activeTransform для каждого из объектов рассчитывается как комбинация преобразований из родительских объектов и собственного преобразования.

Также, в отличие от предыдущих программ, процедура рисования начала входить в самого класса объекта. Эта процедура рисования использует локальный преобразователь координат activeTransform, который может быть уникальным для каждого объекта.

Пусть имеем основной объект А и зависимый объект Б. Для этих объектов движение рассчитывается в начале сцены как общий. Тогда для объекта А, Tranform.x постепенно менять свое значение. Для объекта Б Tranform является постоянным. Но для объекта А activeTransform совпадать с Transform, когда для объекта Б. activeTransform.x = A. activeTransform.x + Б. Transform.x ;.

Для рисования полного множества объектов образуется алгоритм:

1) Объект перед рисованием, всем зависимым объектам, меняет на базе собственного activeTransform, activeTransform зависимых объектов.

2) выполняет собственно рисования.

3) Вызывает по списку процедуру рисования зависимых объектов.

По этому алгоритму родительские объекты отвечают за актуальность значения параметров преобразователей координат activeTransform в зависимых объектах:

```
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;
```

Из приведенного листинга видно, для получения комбинации преобразований необходимо выполнить сложение углов поворота, коэффициент пропорциональности является произведением базовых коэффициентов и трансформация переноса является суммами переносов по горизонтали и вертикали соответственно.

Для изображения объектов с рис. 2 пусть создан файлы 8.txt и 4.txt. Для воспроизведения композиции используем чтения указанных форм из файла:

```
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) // Влиять на все зависящие объекты
```

3. Порядок преобразования координат

Допустимо использование трех объектов, которые цепью зависят друг от друга A → B → C. Для такой системы весьма важно правильно использовать последовательность выполнения объединения преобразований координат. Для иллюстрации проблемы, приведем

конкретном случае, пусть объекты A, B, C имеют положения (50, 50), (30; 30), (15,15) соответственно, и углы

поворота 45º, 0º, 90º. Каждая из функций преобразования координат имеет вид: $\{ x_1 = dx_0 + x_0 \cos(\alpha) + y_0 \sin(\alpha)$

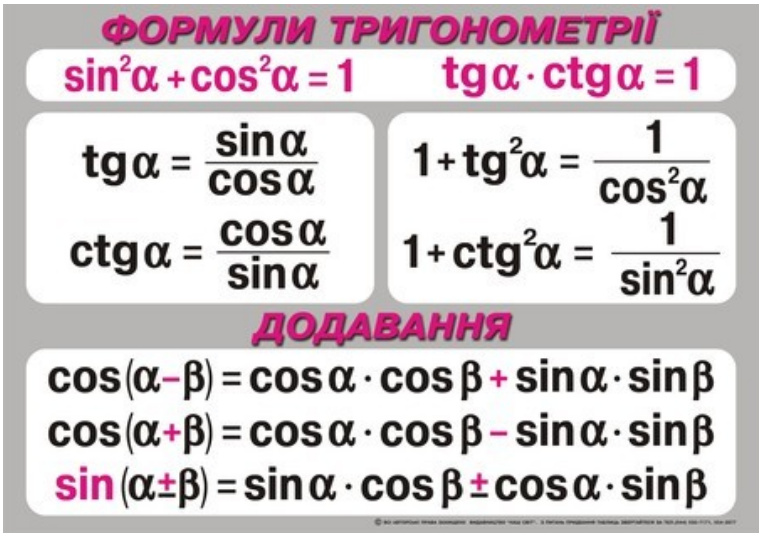
$y_1 = dy_0 - x_0 \sin(\alpha) + y_0 \cos(\alpha)$, $\begin{cases} x_2 = dx_1 + x_1 \cos(\alpha) + y_1 \sin(\alpha) \\ y_2 = dy_1 - x_1 \sin(\alpha) + y_1 \cos(\alpha) \end{cases}$.

Комбинация этих преобразований являются:

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

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

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



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

Использование преобразований в ином порядке, приведет к получению несколько иной формуле:

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

что приведет к другому положению предмета. Поэтому последовательность преобразования координат имеет важное значение.

4. Сцена из зависимых объектов: модель Солнце-Земля-Луна

используем полученные знания для создание трехуровневого движения "Солнце" - "Земля" - "Луна", когда движение Солнца влияет на движение Земли, которая в свою очередь влияет на движение Луны. При этом заставим "Землю" и "Луна" вращаться вокруг оси с различными скоростями. Для обеспечения движения каждого тела разберем их движение отдельно. Введем глобальную переменную frameCount: Integer. Эта переменная будет увеличиваться на единицу с каждым срабатыванием таймера с рисованием кадра.

Начальными настройками для "Солнечной системы" будет:

```
procedure TForm1.FormCreate (Sender: TObject);
begin
    Framecount: = 0;  // начальное количество показанных кадров
    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) //
    Вдвое меньше чем "Солнце"

    Eath.Tranform.SetAlpha (PI * 0.25) Sun.AddChild (Eath) // В "Солнца" присоединим
    "Землю"
    Moon = TMyImage.Create;
    Moon.LoadPicFromFile ( 'moon.txt'); Moon.Tranform.SetXY (50,50) Moon.Tranform.SetMXY
    (0.5,0.5) // Вдвое меньше чем "Земля"

    Moon.Tranform.SetAlpha (0); Eath.AddChild (Moon) // В "Земле" присоединяем "Луна"

    Timer1.Enabled = true;
end;
```

Процедура рисования содержит комбинацию собственного преобразования координат с присоединенными объектами, рисование собственных линий и вызова процедур рисования присоединенных объектов:

```
procedure TMyImage.DrawMyImage (Canvas: TCanvas)
var
    t, i, n: Integer;
    p1, p2: TMPoint; t1:
    TTransformer;
begin
    n = Child.Count; // Определение количества присоединенных объектов
```

```
    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; // Преобразуем координаты присоединенных объектов
    n = Length (PolyLine) // Количество индексов в линиях объектов
    t = - 1; // Значения предыдущего индекса точки
    for i = 0 to n-1 do begin
        if ( t> = 0) and ( PolyLine [i]> = 0) then begin // Если один из индексы 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]; // Получаем следующий индекс
    end;
    n = Child.Count; // Определение количества присоединенных объектов
    for i = 0 to n-1 do begin // Рисуем присоединены объекты
        TmyImage (Child.Items [i]). DrawMyImage (Canvas)
    end;
end;
```

Остается использовать обработчик события таймера для перетахунку координат каждого из объектов:

```
procedure TForm1.Timer1Timer (Sender: TObject);
begin
    Image1.Canvas.Brush.Color = clWhite;
    Image1.Canvas.FillRect (0,0, Image1.Width, Image1.Height)
// очистили фон белым
    Sun.activeTransform.x = 200 + Round ( 100 * sin (0.015 * frameCount)) Sun.activeTransform.y = 200; // "Солнце"
    перемещается только по X
    Eath.Tranform.SetXY (100.0 * cos (0.1154687 * frameCount),
        100.0 * sin (0.1154687 * frameCount)) // "Земля" вращается по кругу
    Eath.Tranform.alpha = 0.3215354 * frameCount; // Обращаем "Землю"
    Moon.Tranform.SetXY (50.0 * cos (0.4154687 * frameCount),
      50.0 * sin (0.4154687 * frameCount)) // "Луна" вращается по
```

```
                                                    // меньшему радиусу
    Moon.Tranform.alpha = 0.4215354 * frameCount; // "Луна" тоже вращается
    Sun.DrawMyImage (Image1.Canvas) // Рисуем "Солнце" все присоединенные объекты
                                                    // рисуются автоматически
    frameCount = frameCount + 1; // Нарисованный следующий кадр
end;
```

Также для рисования "космических объектов" сформируем изображения, показанном на рис. 3, 4, 5. В этих изображениях использовано внутренние узоры для иллюстрации вращательного движения. В результате изменений в программном коде и сформированному изображению, мы имеем возможность наблюдать взаимное движение нескольких объектов. Один из фрагментов взаимного движения показано на рис. 6.

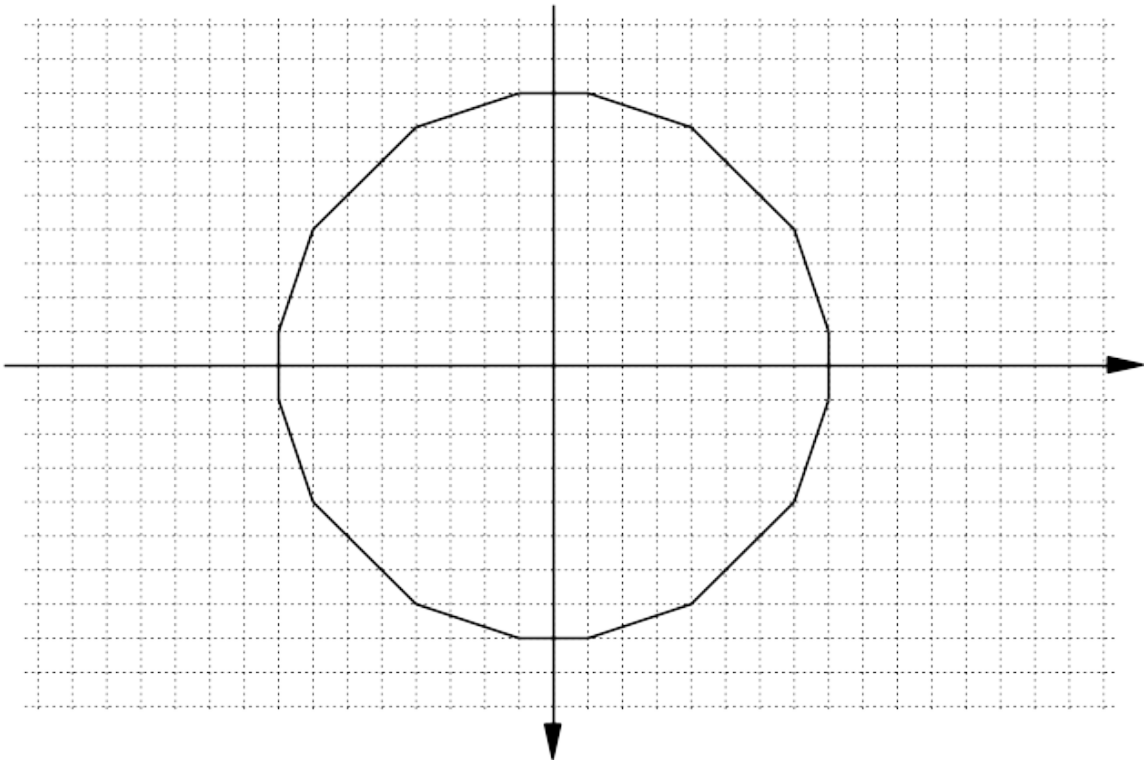


Рис. 3 - изображение "Солнца"

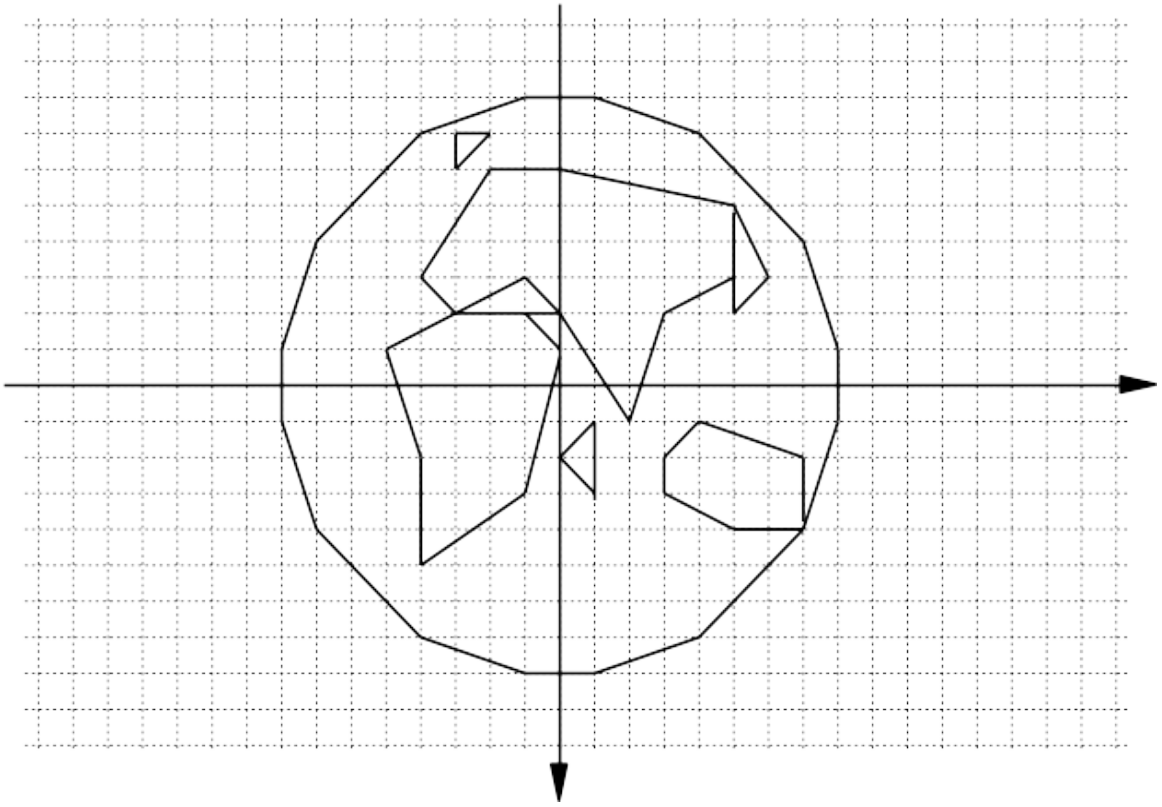


Рис. 4 - изображение "Земля"

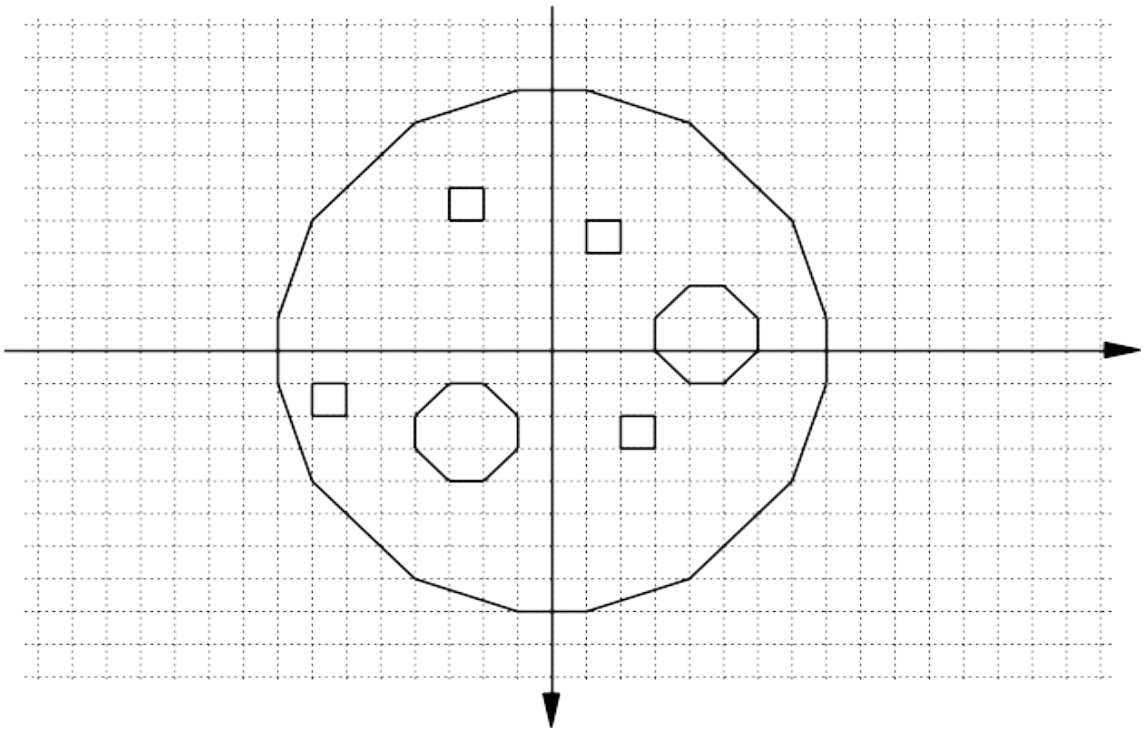


Рис. 5 - изображение "Луна"

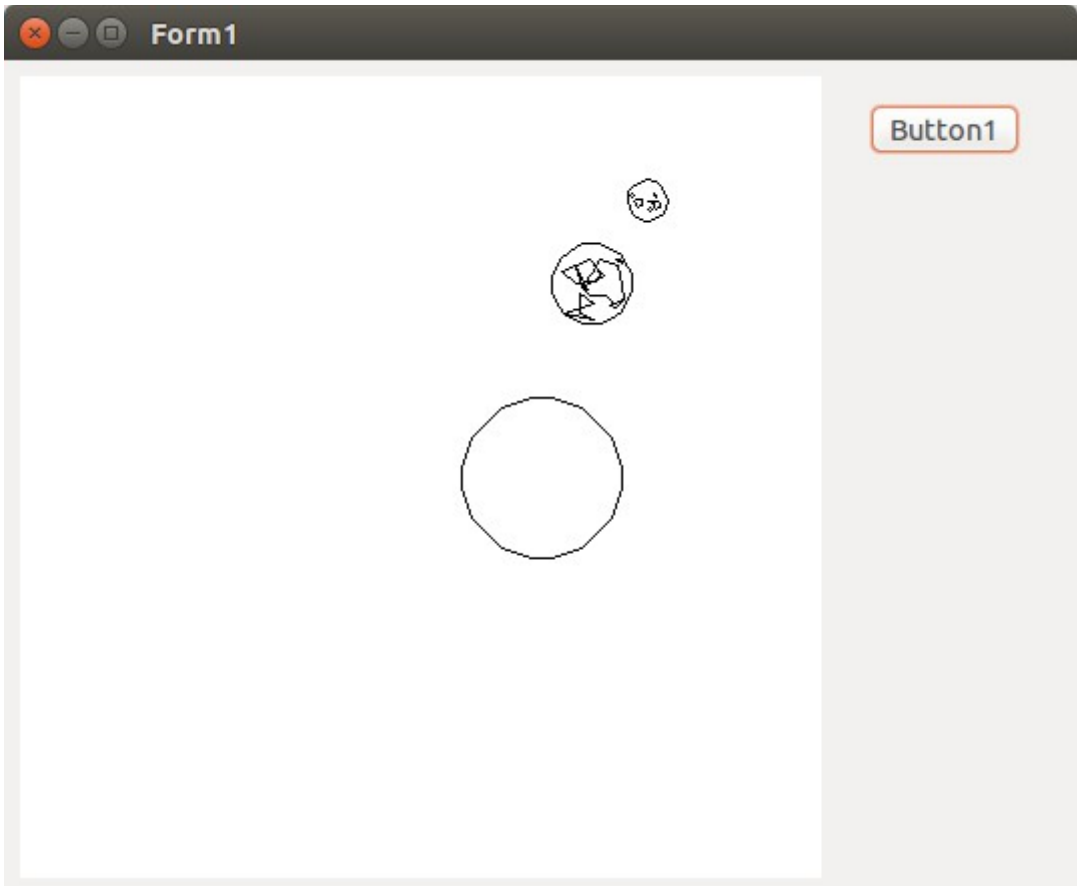


Рис. 6 - Взаимный движение трех объектов

Приложение А. Текст файла с изображением "Солнца", 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 18

мая

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

Приложение Б. Изображения "Земли", 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

-15 мая

-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

Приложение В. Изображения "Луны", 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 мая

-30 октября

-15 -20

-15 -25

-10 -25

-10 -20

-20 октября

-15 мая

-10 Мая

-10 Мая

-15 мая

-20 октября

-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

Приложение Г. Текст программного кода для вывода зависимых объектов

```
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, px, py) 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.
```