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 $N \log_2 N$ 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.
```