МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни

“МЕТОДИ ТА ЗАСОБИ АНАЛІЗУ ЗОБРАЖЕНЬ”

для студентів усіх форм навчання
за спеціальностями 7.05140201, 8.05140201 «Біомедична інженерія»,
7.05140203, 8.05140203 «Інформаційні технології в біомедицині»

Електронне видання

ЗАТВЕРДЖЕНО
кафедрою «Біомедична інженерія».
Протокол № 6 від 18.01.2014 р.

ХАРКІВ 2014

Методичні вказівки до лабораторних робіт з дисципліни “Методи та засоби аналізу зображень” для студентів усіх форм навчання за спеціальностями “Біомедична інженерія” та «Інформаційні технології в біомедицині» [Електронне видання] / Упорядн. О.Г. Аврунін – Харків: ХНУРЕ, 2014. – 56 с.

Упорядник: О.Г. Аврунін

Рецензент: Л.О. Авер’янова, канд.техн. наук, доцент кафедри БМІ

ЗМІСТ

ВСТУП

Методичні вказівки призначені для проведення лабораторних робіт з курсу «Методи та засоби аналізу зображень». Лабораторні роботи введені в навчальний процес з метою вивчення на практиці різних методів аналізу біомедичних зображень та розробці відповідних програмних засобів в середовищі Borland Delphi 6.0 і 7.0.

Методичні вказівки мають описання 6 лабораторних робіт. Кожний розділ курсу відповідає окремій лаборант роботі, яка складається з таких підрозділів: мета роботи, підготовка до виконання роботи, порядок виконання роботи, зміст звіту, контрольні питання.

Для виконання лабораторних робіт студент повинен володіти основними навиками програмування в середовищі Delphi та Object Pascal. Також студент повинен продемонструвати творчий підхід до розробки модулів програмного забезпечення, алгоритмічне мислення.

Виконання кожної лабораторної роботи включає наступні етапи:

- отримання завдання;
- вивення теоретичного матеріалу з теми лабораторної роботи;
- розробка алритму програми;
- написання програмного засобу;
- перевірка правильності виконання завдання;
- оформлення звіту та захист лабораторної роботи.

Обробка біомедичних зображень та даних є найбільш інформативним носієм діагностичної інформації. На сучасному етапі розвитку різних програмних засобів розробка та більш глибокий аналіз зображень є важливою та актуальною задачею в області біомедичної інженерії.

1 МЕТОДИ АВТОМАТИЧНОГО ПІДБОРУ ПОРОГІВ СЕГМЕНТАЦІЇ

1.1 Мета роботи

Ознайомитися з методами автоматичного підбору порогів сегментації для біомедичних зображень. Реалізувати програмне забезпечення, яке дало б змогу в автоматичному режимі обирати пороги сегментації, а також сегментувати біомедичні зображення.

1.2 Підготовка до виконання роботи

Під час підготовки до виконання роботи слід згадати матеріал, що стосується мови програмування Object Pascal – базової мови програмування Delphi.

Для сегментації зображень дуже часто використовується поняття як яскравість пікселів. Найбільш простим є задати порог сегментації, що буде виражати, які пікселі зображення слід вважати фоном, а які досліджуваним об’єктом.

Вибір порогу має свої складнощі, в багатьох випадках він є спеціалізованим для конкретної ситуації в залежності від природи зображення і т.д. Метод шукає поріг сегментації таким чином, щоб мінімізувати дисперсію всередині класу.

Метод збалансованного порогового відсічення гістограми також побудований на припущенні, що існує два класи пікселів – фону та об’єкту. Алгоритм починається з середнього значення яскравості, на кожному кроці вираховується вага пікселів зліва та справа гістограми. Видаляється стовбчик в гістограмі з того боку де вага переважає. Перераховується центральне значення яскравості. Це відбувається до тих пір, поки не лишиться один стовбчик в гістограмі. Це значення й використовується як порогове.

1.3 Порядок виконання роботи

1.3.1 Запустити програму Delphi

1.3.2 На диску у папці зі своїм прізвищем створити нову папку Lab1, у якій зберегти створюваний проект. У цій же папці зберегти медичні зображення видані викладачем.

1.3.3 У вікні Form1 розмістити 4 компоненти типу TImage, 1 компонент типу TButton та 1 компонент типу TOpenPictureDialog. Для перших трьох зображень вставновити властивості Stretch та Proportional в True. Для четвертого компоненту TImage лише властивість Strecth встановити як True.

Вигляд інтерфейсу представлено на рис. 1.1

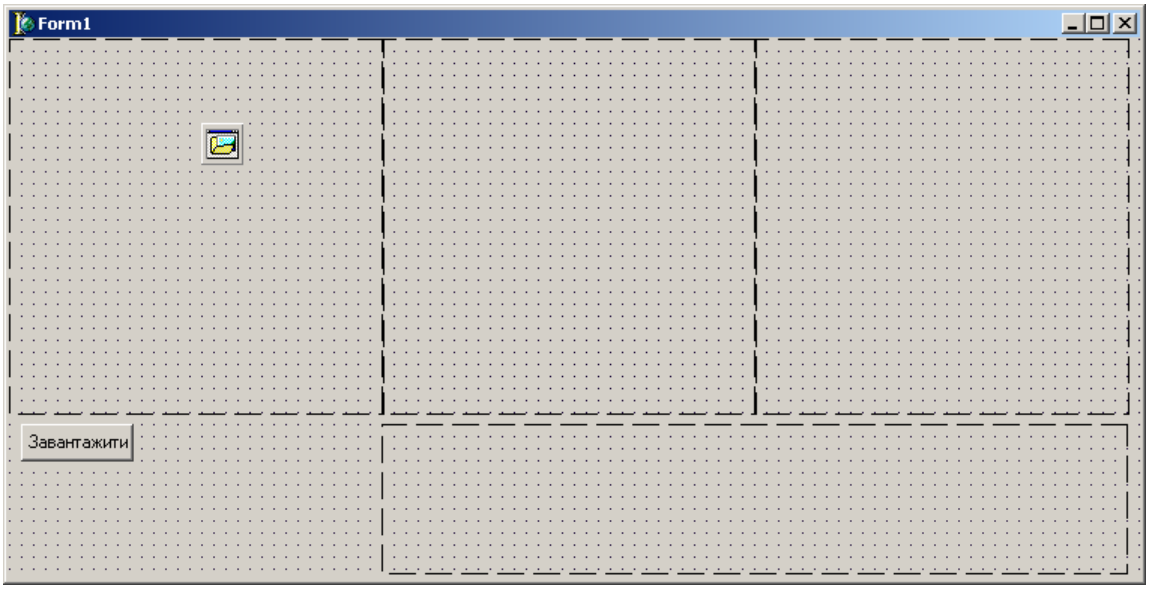

Рисунок 1.1 – Варіант вікна під час проектування інтерфейсу програми

1.3.5 Перейти в вікно редагування коду (F12). В класі TForm1 в приватній секції об’явити поле FHistogram, що буде містити сірошкальну гістограму зображення.

FHist: Array [0..255]of Integer;

В приватній секції також об’явити наступні методи:

```
procedure CalculateHist();
function GetOtsuThreshold(): Integer;
function GetBalansedThresold(): Integer;
```

Скориставшись швидкими клавішами Ctrl+Shift+C, створити пусті реалізації цих класів.

1.3.6 Реалізувати алгоритм побудови сірошкальної гістограми зображення:

```
procedure TForm1.CalculateHist;
var
  i,j,w,h: Integer;
  c: TColor;
begin
```

```
  for i:=0 to 255 do FHist[i] := 0;
  w := Image1.Picture.Bitmap.Width;
  h := Image1.Picture.Bitmap.Height;
  for j:=0 to h-1 do
  begin
    for i:=0 to w-1 do
    begin
      c := Image1.Picture.Bitmap.Canvas.Pixels[i,j];
Inc(FHist[Trunc(0.3*GetRValue(c)+0.6*GetGValue(c)+0.1*GetBValue(c))]);
    end;
  end;
end;
```

1.3.7 Реалізувати алгоритм пошуку оптимального порогу методом сбалансованого порогового відсічення гістограми:

```
function TForm1.GetBalansedThresold: Integer;
var
  size,
  threshold_m,
  threshold_start,
  threshold_finish: Integer;
  weight_left,
  weight_right: Integer;
  i: Integer;
begin
  threshold_start := 0;
  size := 256;
  threshold_finish := size - 1;
  threshold_m := (threshold_finish + threshold_start) div 2;
  weight_left := 0;
  for i:=threshold_start to threshold_m do
    weight_left := weight_left + FHist[i];

  weight_right := 0;
  for i:=threshold_m+1 to threshold_finish do
    weight_right := weight_right + FHist[i];
  while threshold_start<=threshold_finish do
  begin
```

```
    if weight_right>weight_left then
    begin
      weight_right := weight_right - FHist[threshold_finish];
      Dec(threshold_finish);
      if ((threshold_finish + threshold_start) div 2) < threshold_m then
      begin
        weight_right := weight_right + FHist[threshold_m];
        weight_left := weight_left - FHist[threshold_m];
        Dec(threshold_m);
      end;
    end
    else
    begin
      weight_left := weight_left - FHist[threshold_start];
      Inc(threshold_start);
      if ((threshold_finish + threshold_start) div 2) > threshold_m then
      begin
        weight_left := weight_left - FHist[threshold_m+1];
        weight_right := weight_right - FHist[threshold_m+1];
        Inc(threshold_m);
      end;
    end;
  end;
  Result := threshold_m;
end;
```

1.3.8 Реалізувати алгоритм пошуку оптимального порогу методом Оцу:

```
function TForm1.GetOtsuThreshold: Integer;
var
  maxvariance,variance,weight1,weight2,mean1,mean2 : Double;
  resThreshold: Integer;
  currThreshold: Integer;
  i,size: Integer;
begin
  size := 255;
  currThreshold := 1;
  resThreshold := currThreshold;
  weight1 := 0;
```

```
for i:=0 to currThreshold do
  weight1 := weight1 + FHist[i];
weight2 := 0;
for i:=currThreshold to size-1 do
  weight2 := weight2 + FHist[i];
mean1 := 0;
for i:=0 to currThreshold do
begin
  mean1 := mean1 + FHist[i]*i;
end;
if weight1<>0 then
  mean1 := mean1/weight1
else
  mean1 := 0;
mean2 := 0;
for i:=currThreshold to size-1 do
begin
  mean2 := mean2 + FHist[i]*i;
end;
if weight2<>0 then
  mean2 := mean2/weight2
else
  mean2 := 0;
maxvariance := (mean1-mean2)*(mean1-mean2)*weight1*weight2;
for currThreshold:=2 to size-2 do
begin
  weight1 := 0;
  for i:=0 to currThreshold do
    weight1 := weight1 + FHist[i];
  weight2 := 0;
  for i:=currThreshold to size-1 do
    weight2 := weight2 + FHist[i];
  mean1 := 0;
  for i:=0 to currThreshold do
  begin
    mean1 := mean1 + FHist[i]*i;
  end;
```

```
    if weight1<>0 then
      mean1 := mean1/weight1
    else
      mean1 := 0;
    mean2 := 0;
    for i:=currThreshold to size-1 do
    begin
      mean2 := mean2 + FHist[i]*i;
    end;
    if weight2<>0 then
      mean2 := mean2/weight2
    else
      mean2 := 0;
    variance := (mean1-mean2)*(mean1-mean2)*weight1*weight2;
    if variance>maxvariance then
    begin
      resThreshold := currThreshold;
      maxvariance := variance;
    end;
  end;
  Result := resThreshold;
end;
```

1.3.9 Для кнопки Button1 написати обробник події:

```
procedure TForm1.Button1Click(Sender: TObject);
const
  HIST_HEIGHT = 100;
var
  thresholdOtsu: Integer;
  thresholdBalanced:Integer;
  max,i,j: Integer;
  w,h: Integer;
  intensity: Byte;
  c: TColor;
begin
  if not OpenPictureDialog1.Execute then Exit;
  Image1.Picture.Bitmap.LoadFromFile(OpenPictureDialog1.FileName);
  CalculateHist();
```

```
      thresholdOtsu := GetOtsuThreshold;
      thresholdBalanced := GetBalansedThresold;
      Image4.Picture.Bitmap.Width := 256;
      Image4.Picture.Bitmap.Height := HIST_HEIGHT;
      Image4.Picture.Bitmap.Canvas.Brush.Color := clWhite;
      Image4.Picture.Bitmap.Canvas.FillRect(Rect(0,0,256,HIST_HEIGHT));
      max := FHist[0];
      for i:=1 to 255 do
      begin
        if max<FHist[i]then max := FHist[i];
      end;
      Image4.Picture.Bitmap.Canvas.Pen.Color := clBlack;
      for i:=0 to 255 do
      begin
        Image4.Picture.Bitmap.Canvas.MoveTo(i,HIST_HEIGHT);
        Image4.Picture.Bitmap.Canvas.LineTo(i,Trunc(HIST_HEIGHT-
HIST_HEIGHT*(FHist[i]/max)));
      end;
      Image4.Picture.Bitmap.Canvas.Pen.Color := clGreen;
      Image4.Picture.Bitmap.Canvas.MoveTo(thresholdOtsu,HIST_HEIGHT);
      Image4.Picture.Bitmap.Canvas.LineTo(thresholdOtsu,0);
      Image4.Picture.Bitmap.Canvas.Pen.Color := clBlue;
      Image4.Picture.Bitmap.Canvas.MoveTo(thresholdBalanced,HIST_HEIGHT);
      Image4.Picture.Bitmap.Canvas.LineTo(thresholdBalanced,0);
      w := Image1.Picture.Bitmap.Width;
      h := Image1.Picture.Bitmap.Width;
      Image2.Picture.Bitmap.Width := w;
      Image2.Picture.Bitmap.Height := h;
      Image3.Picture.Bitmap.Width := w;
      Image3.Picture.Bitmap.Height := h;
      for j:=0 to h-1 do
      begin
        for i:=0 to w-1 do
        begin
          c := Image1.Picture.Bitmap.Canvas.Pixels[i,j];
          intensity :=
    Trunc(0.3*GetRValue(c) + 0.6*GetGValue(c) + 0.1*GetBValue(c));
```

```
      if intensity<thresholdBalanced then
        Image2.Picture.Bitmap.Canvas.Pixels[i,j] := clBlack
      else
         Image2.Picture.Bitmap.Canvas.Pixels[i,j] := clWhite;
      if intensity<thresholdOtsu then
        Image3.Picture.Bitmap.Canvas.Pixels[i,j] := clBlack
      else
        Image3.Picture.Bitmap.Canvas.Pixels[i,j] := clWhite;
    end;
  end;
end;
```

1.3.10 Запустити программу на виконання. Для всіх зображень наданих викладачем спостерігати вибір порогу сегментації. Приклад роботи програми зображено на рис. 1.2.

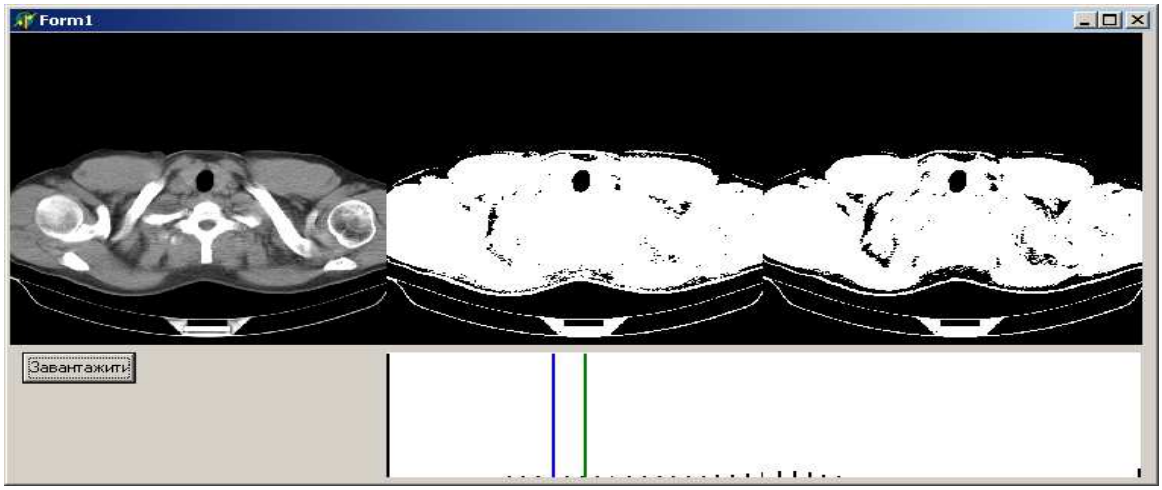

Рисунок 1.2 – Результат роботи програми

1.4 Зміст звіту

У звіті наводять:

– назву роботи;
– мету роботи;
– зображення інтерфейсу програми;
– результати роботи програми для різних зображень
– лістинг програми;
– висновки.

1.5 Контрольні питання

1. Що таке сегментація?
2. На чому базується метод Оцу?
3. Пояснити алгоритм роботи методу Оцу?
4. Пояснити алгоритм роботи методу сбалансованого порогового відсічення гістограми.
5. Назвати переваги методів автоматичного визначення порогів сегментації та їх недоліки.

2 ПЕРЕТВОРЕННЯ ХАФА ДЛЯ ДЕТЕКТУВАННЯ ПРЯМИХ НА ЗОБРАЖЕННІ

2.1 Мета роботи

Ознайомитися з перетворенням Хафа для детектування прямих на зображенні.

2.2 Підготовка до виконання роботи

Під час підготовки до виконання роботи слід згадати матеріал, що стосується мови програмування Object Pascal – базової мови програмування Delphi.

Перетворення Хафа часто використовується в обробці та при аналізі зображень для детектування об’єктів, які можна описати за допомогою певного рівняння з використанням методу голосування.

Класичний алгоритм перетворення Хафа використовувався для детектування прямих, але пізніше його було розширено до кругів та еліпсів. Проте, подальше ускладнення фігур призводить до збільшення часових затрат.

2.3 Порядок виконання роботи

2.3.1 Запустити програму Delphi

2.3.2 На диску у папці зі своїм прізвищем створити нову папку Lab2, у якій зберегти створюваний проект. У цій же папці зберегти медичні зображення видані викладачем.

2.3.3 У вікні Form1 розмістити 2 компоненти типу TImage, 2 компоненти типу TButton та 1 компонент типу TEdit. Для зображень вставновити властивості Stretch в True. Першому компоненту TImage дати ім’я iSrc, а другому iDst. Компоненту TEdit властивість Text встановити в значення 10, а поле Name – eNoise.

Вигляд інтерфейсу представлено на рис. 2.1

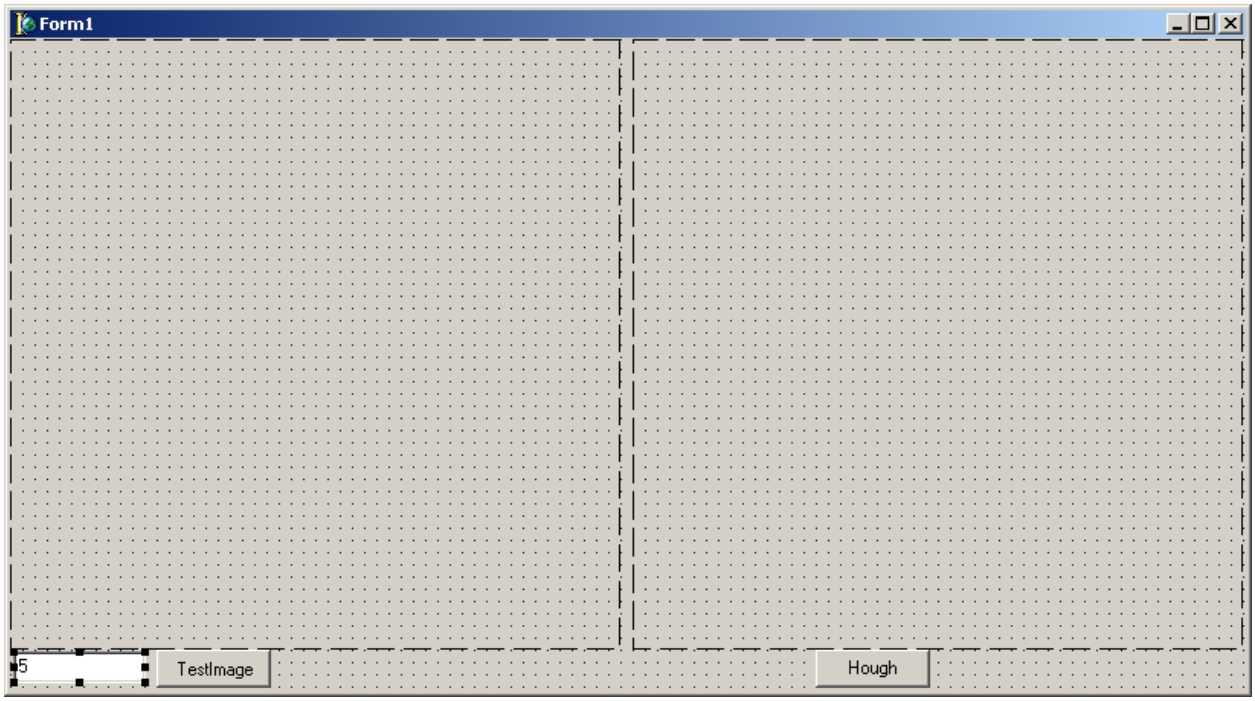

Рисунок 2.1 – Варіант вікна під час проектування інтерфейсу програми

2.3.5 Перейти в вікно редагування коду (F12). В класі TForm1 в приватній секції об’явити два методи:

```
TForm1 = class(TForm)
…
private
  procedure CreateTestImage(const LineCount: Integer;
                            const Noise: Integer);
  procedure Hough();
public
end;
```

2.3.6 Реалізувати методо CreateTestImage, для цього перейти в секцію коду implementation і описати дію по створенню тестового зображення:

```
procedure TForm1.CreateTestImage(const LineCount, Noise: Integer);
```

```
const
  IMAGE_WIDTH = 100;
  IMAGE_HEIGHT = 100;
  function RandomPoint(): TPoint;
  begin
    Result.X := Random(IMAGE_WIDTH);
    Result.Y := Random(IMAGE_HEIGHT);
  end;
const
  BACKGROUND_COLOR = clBlack;
  PEN_COLOR = clWhite;
var
  i: Integer;
  start,finish,p: TPoint;
begin
  iSrc.Picture.Bitmap.Width := IMAGE_WIDTH;
  iSrc.Picture.Bitmap.Height := IMAGE_HEIGHT;
  with iSrc.Picture.Bitmap.Canvas do
  begin
    Pen.Color := BACKGROUND_COLOR;
    Brush.Color := BACKGROUND_COLOR;
    FillRect(Rect(0,0,IMAGE_WIDTH,IMAGE_HEIGHT));
    Pen.Color := PEN_COLOR;
    for i:=0 to LineCount-1 do
    begin
      start  := RandomPoint();
      finish := RandomPoint();
      MoveTo(start.X,start.Y);
      LineTo(finish.X,finish.Y);
    end;
    for i:=0 to Noise-1 do
    begin
      p := RandomPoint();
      Pixels[p.x,p.y] := PEN_COLOR;
    end;
  end;
end;
```

2.3.7 Реалізувати метод по пошуку прямих з використанням перетворення Хафа:

```
procedure TForm1.Hough();
const
  PI_180 = 180;
var
  RMax, W, H, MaxPhaseValue: integer;
  Teta,R: Double;
  PhaseSpace: array of array of integer;
  i, j, m, n: integer;
begin
  W  := iSrc.Picture.Bitmap.Width;
  H := iSrc.Picture.Bitmap.Height;
  RMax := round(sqrt(W * W + H * H));
  SetLength(PhaseSpace, PI_180);
  for i := Low(PhaseSpace) to High(PhaseSpace) do SetLength(PhaseSpace[i], RMax);
  for i := 0 to PI_180-1 do
   for j := 0 to RMax-1 do
   begin
    PhaseSpace[i,j] := 0;
   end;
  for m := 0 to W-1 do
  begin
   for n := 0 to H-1 do
   begin
    if (iSrc.Picture.Bitmap.Canvas.Pixels[m,n] >0 ) then
    begin
     for i := 0 to PI_180-1 do
     begin
      for j := 0 to RMax-1 do
      begin
       Teta:=i/PI_180*Pi;
       if ( abs((m*sin(Teta) + n*cos(Teta)) - j)<0.1 ) then inc(PhaseSpace[i,j]);
      end;
     end;
    end;
```

```
    end;
  end;
  Teta := 0;
  R := 0;
  iDst.Picture.Bitmap.Width := PI_180;
  iDst.Picture.Bitmap.Height := RMax;
  MaxPhaseValue := 0;
  for i := 0 to 179 do
  begin
    for j := 0 to RMax-1 do
    begin

iDst.Picture.Bitmap.Canvas.Pixels[i,j]:=RGB(PhaseSpace[i,j],PhaseSpace[i,j],PhaseS
pace[i,j]);
      if (PhaseSpace[i,j] > MaxPhaseValue) then
      begin
        MaxPhaseValue := PhaseSpace[i,j];
        Teta:=i;
        R:=j;
      end;
    end;
  end;
  Teta:=Teta/PI_180*Pi;
  for m := 0 to W-1 do
    for n := 0 to H-1 do
      if ( round(((m) * sin(Teta) + (n) * cos(Teta))) = R) then
iSrc.Picture.Bitmap.Canvas.Pixels[m,n]:=clBlue;
end;
```

2.3.8 Для кнопки по генерації тестового зображення реалізувати подію OnClick:

```
procedure TForm1.bb1Click(Sender: TObject);
begin
  CreateTestImage(1,StrToInt(eNoise.Text));
end;
```

2.3.9 Для кнопки по запуску алгоритма Хафа реалізувати подію OnClick:

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

```
  Hough();
end;
```

2.3.10 Описати в кінці програми секцію initialization:

```
initialization
  Randomize();
```

2.3.11 Запустити программу на виконання. Провести дослідження роботи методу Хафа по детектуванні прямих з різними значеннями шуму. Приклад роботи програми зображено на рис. 2.2.

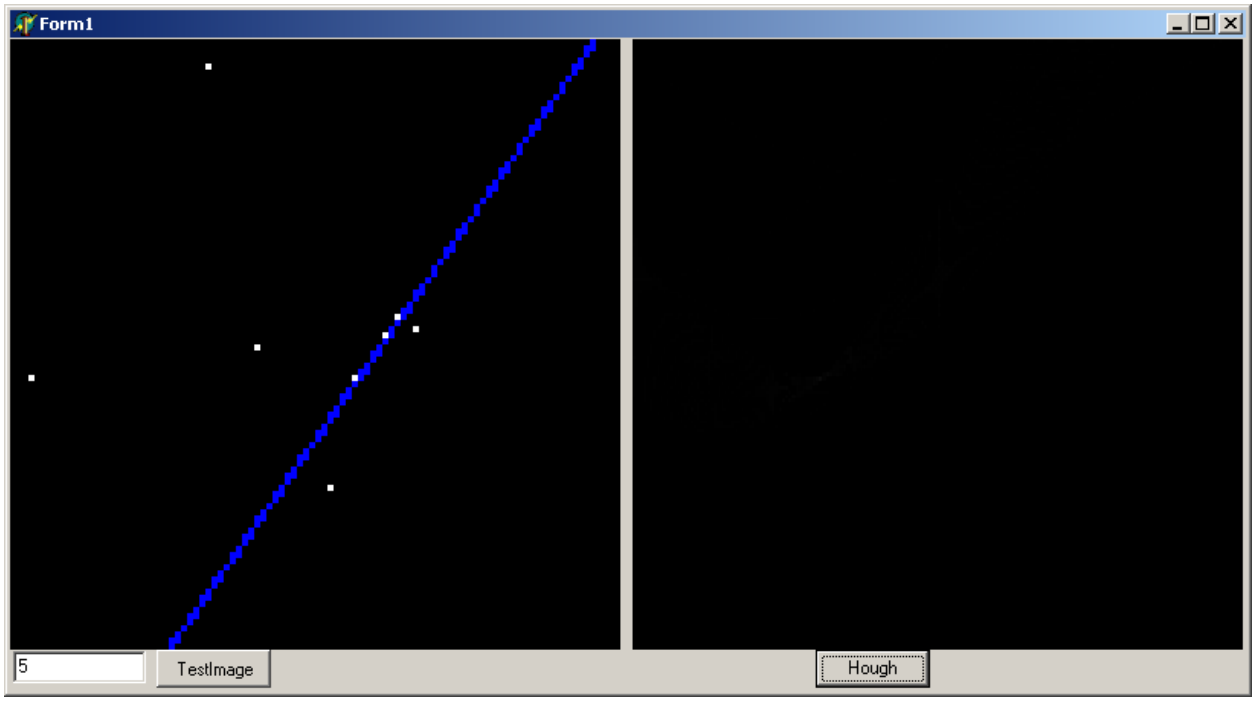

Рисунок 2.2 – Результат роботи програми

2.4 Зміст звіту

У звіті наводять:

– назву роботи;
– мету роботи;
– зображення інтерфейсу програми;
– результати роботи програми для різних значень шуму
– лістинг програми;
– висновки.

2.5 Контрольні питання

1. Що таке перетворення Хафа?
2. Де застосовується перетворення Хафа?
3. Описати видозміну програми для детектування декількох прямих?
4. Як впливає шум на коректність роботи програми?
5. Назвати переваги та недоліки перетворення Хафа?

3 СЕГМЕНТАЦІЯ БІОМЕДИЧНИХ ЗОБРАЖЕНЬ МЕТОДОМ *K*-СЕРЕДНІХ

3.1 Мета роботи

Ознайомитися із застосуванням метода к-середніх для сегментації медичних зображень. Реалізувати програмне забезпечення, що дозволить в автоматичному режимі сегментувати медичне зображення методом *к*-середніх.

3.2 Підготовка до виконання роботи

Під час підготовки до виконання роботи слід згадати матеріал, що стосується мови програмування Object Pascal – базової мови програмування Delphi.

Метод *к*-середніх є одним із найбільш простіших методів кластеризації, якщо відома кількість класів на яку слід розбити дані. Цей алгоритм можна застосовувати для сегментації медичних зображень.

Алгоритм кластеризації методом к-середніх полягає в наступному:

а) створюється N класів (N – кількість класів, на яку слід розбити дані). Центри мас класів ініціалізуються певними значенням. Можна ініціалізувати випадковими значеннями.

б) дані, які необхідно кластеризувати, послідовно аналізуються і відносяться до того класу, до якого вони найближчі (по Евклідовій відстані і центру мас).

в) для кожного класу перераховується центр маси.

г) якщо у якогось класу центр маси змінений то відбувається перехід на крок “б”.

д) виходячи з найменшої відстані центру мас кластеру і відповідних даних, дані відносять до певного класу.

3.3 Порядок виконання роботи

3.3.1 Запустити програму Delphi

3.3.2 На диску у папці зі своїм прізвищем створити нову папку Lab3, у якій зберегти створюваний проект. У цій же папці зберегти медичні зображення, видані викладачем.

3.3.3 У вікні Form1 розмістити 2 компоненти типу TImage, 1 компонент типу TButton та 1 компонент типу TMemo. Для усіх зображень встановити властивості Stretch та Proportional в True. Для першого зображення встановити властивість Picture на зображенні. Для компоненту Memo1 встановити властивість ScrollBar в ssVertical. Приклад вигляду інтерфейсу представлено на рис. 3.1.

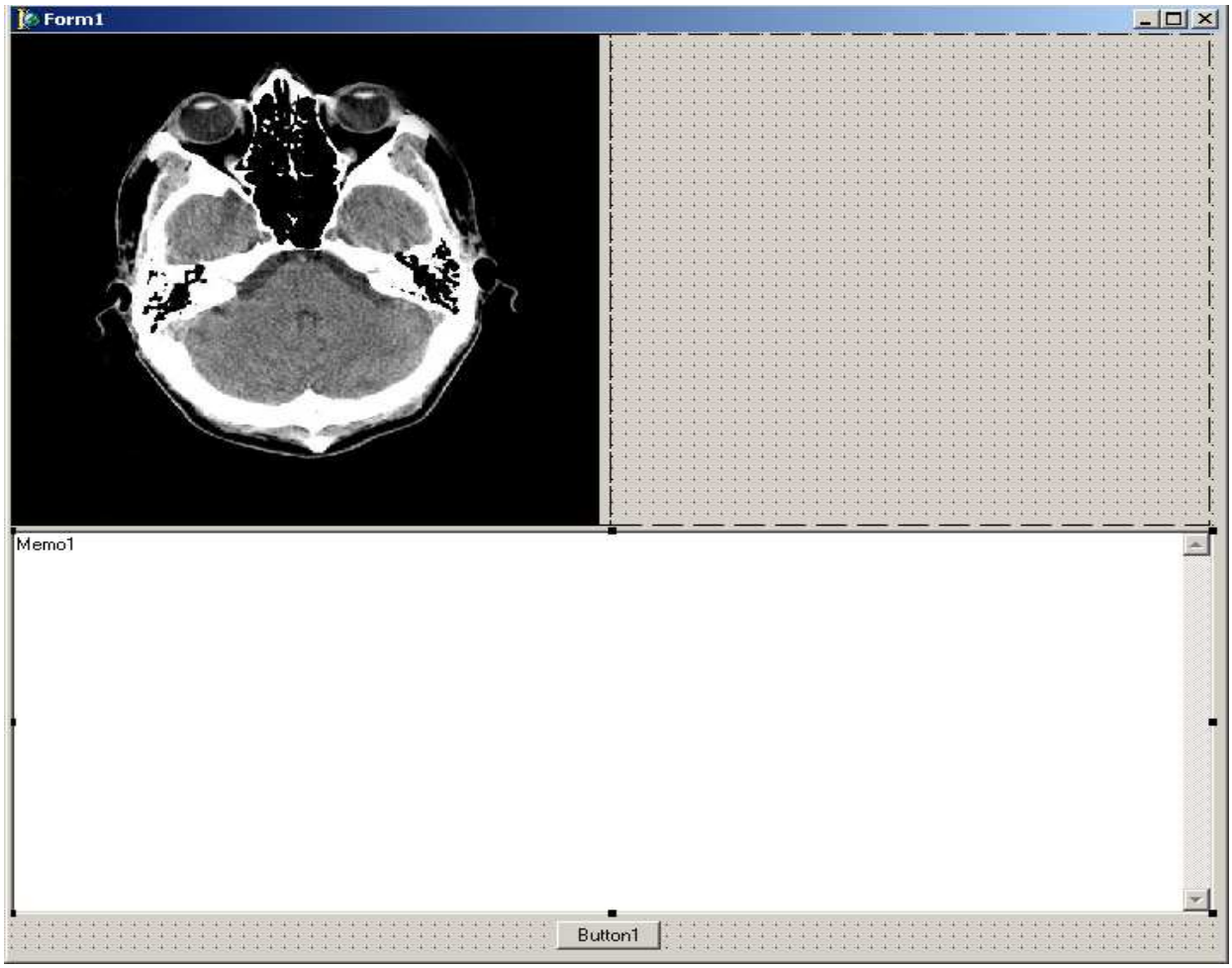

Рисунок 3.1 – Варіант вікна під час проектування інтерфейсу програми

3.3.4 Створити обробник форми OnCreate. Для цього двічі натиснути на формі, при цьому з’явиться вікно редагування коду. Також обробник OnCreate

форми можна створити шляхом двійного натискання миші на події OnCreate вкладки Events інспектору об’єктів. В коді прописати властивість форми DoubleBuffered рівну True. Це необхідно задля уникнення миготіння при пере рисовці форми. Також проініціалізувати генератор випадкових чисел новим значенням.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  DoubleBuffered := True;
  Randomize();
end;
```

3.3.5 Реалізувати алгоритм сегментації методом к-середніх. Для цього створити обробник OnClick для кнопки Button1 (або подвійним кліком на компоненті, або знову ж за допомогою вкладки Events).

```
procedure TForm1.Button1Click(Sender: TObject);
const
  MAXIMUM_ITERATION_COUNT = 50;
  CLASSES_COUNT = 2;
var
  x,y,g: Integer;
  v: Byte;
  gmin,gcurr: Single;
  gnum: Integer;
  group: Array [0..CLASSES_COUNT-1]of record
    v,c,nv: Integer;
  end;
  changed: Boolean;
  iteration: Integer;
begin
  Image2.Picture.Bitmap.Width := Image1.Picture.Bitmap.Width;
  Image2.Picture.Bitmap.Height := Image1.Picture.Bitmap.Height;
  Memo1.Lines.Clear();
  for g:=0 to High(group)do
  begin
    group[g].v := Random(255);
    group[g].nv := 0;
    group[g].c := 0;
  end;
```

```
iteration := 0;
repeat
  Memo1.Lines.Add('Iteration #'+intToStr(iteration+1)+': ');
  for y:=0 to Image1.Picture.Bitmap.Height-1 do
  begin
    for x:=0 to Image1.Picture.Bitmap.Width-1 do
    begin
      v:= GetRValue(Image1.Picture.Bitmap.Canvas.Pixels[x,y]);
      gcurr := (group[0].v-v)*(group[0].v-v);
      gmin := gcurr;
      gnum := 0;
      for g:=1 to High(group)do
      begin
        gcurr := (group[g].v-v)*(group[g].v-v);
        if (gcurr<gmin) then
        begin
          gmin := gcurr;
          gnum := g;
        end;
      end;
      group[gnum].nv := group[gnum].nv + v;
      group[gnum].c  := group[gnum].c + 1;
    end;
    Application.ProcessMessages();
  end;
  for g:=0 to High(group)do
  begin
    if group[g].c = 0 then Continue;
    group[g].nv := group[g].nv div group[g].c;
  end;
  Inc(iteration);
  changed := False;
  for g:=0 to High(group)do
  begin
    if (group[g].v<>group[g].nv)and(group[g].c<>0) then
    begin
      changed := True;
```

```
          Break;
        end;
      end;
      for g:=0 to High(group)do
      begin
        Memo1.Lines.Add('        group['+IntToStr(g)+'].v = '+
                                  IntToStr(group[g].v)+
                        ', group['+IntToStr(g)+'].nv = '+
                                  IntToStr(group[g].nv)+';');
        group[g].v := group[g].nv;
        group[g].c := 0;
        group[g].nv := 0;
      end;
    Application.ProcessMessages();
    until not( changed and (iteration<MAXIMUM_ITERATION_COUNT));
    for y:=0 to Image1.Picture.Bitmap.Height-1 do
    begin
      for x:=0 to Image1.Picture.Bitmap.Width-1 do
      begin
        v:= GetRValue(Image1.Picture.Bitmap.Canvas.Pixels[x,y]);
        gcurr := (group[0].v-v)*(group[0].v-v);
        gmin := gcurr;
        gnum := 0;
        for g:=1 to High(group)do
        begin
          gcurr := (group[g].v-v)*(group[g].v-v);
          if (gcurr<gmin) then
          begin
            gmin := gcurr;
            gnum := g;
          end;
        end;
        Image2.Picture.Bitmap.Canvas.Pixels[x,y]                    :=
RGB(group[gnum].v,group[gnum].v,group[gnum].v);
      end;
      Application.ProcessMessages();
    end;
```

end;

3.3.6 Запустити программу на виконання та зафіксувати результат. Змінюючи значення CLASSES_COUNT в програмі від 2 до 6 спостерігати за отриманими результатами (При цьому слід перекомпілювати програму – запустити з Делфі знову на виконання). Фіксувати кількість ітерацій необхідну до повного сходження алгоритму (виконувати сегментацію для одного й того ж зображення декілька разів).

Приклад роботи програми зображено на рис. 3.2.

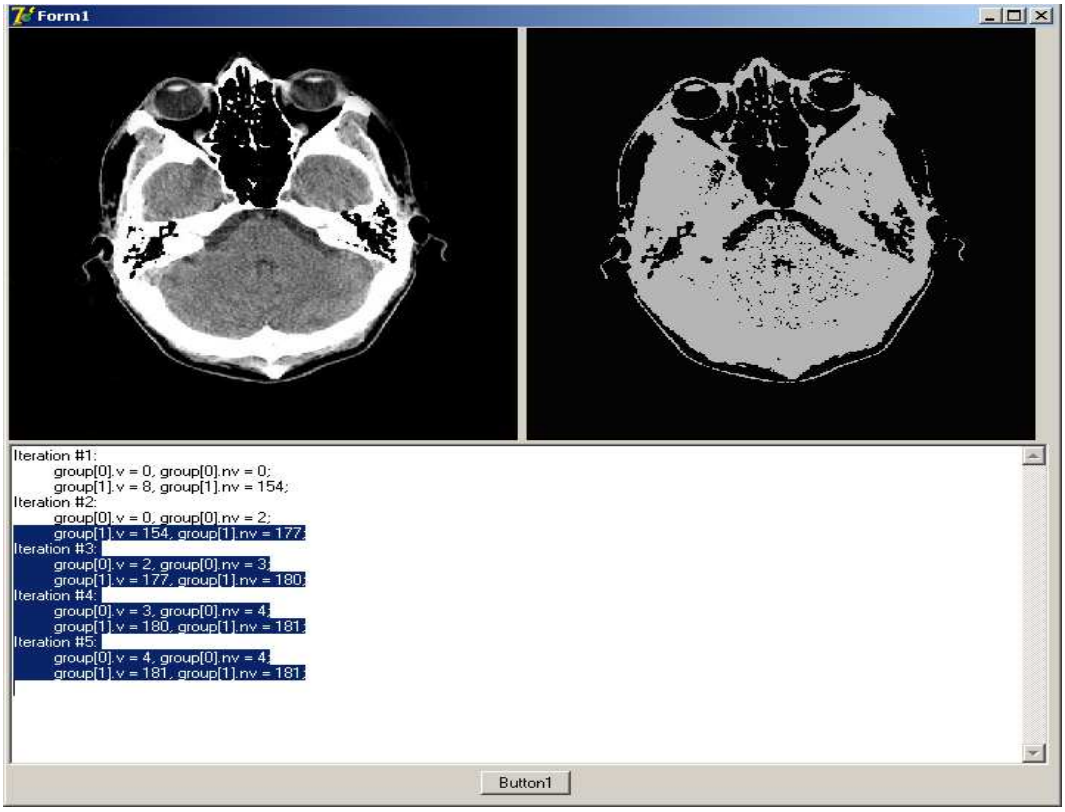

Рисунок 3.2 – Результат роботи програми

3.4 Зміст звіту

У звіті наводять:

– назву роботи;
– мету роботи;
– зображення інтерфейсу програми;
– результати роботи програми для різних зображень;
– лістинг програми;
– висновки.

3.5 Контрольні питання

1. Що таке сегментація?
2. На чому базується метод *к*-середніх?
3. Які методи кластеризації ще відомі?
4. Пояснити алгоритм роботи методу *к*-середніх?
5. Чи завжди кількість ітерацій для одного й того ж зображення однакова?
6. Яке значення використовувалося для класифікації пікселя зображення до того або іншого класу?

4 ДЕТЕКТУВАННЯ ОСОБЛИВИХ ТОЧОК НА ЗОБРАЖЕННІ

4.1 Мета роботи

Ознайомитися із методами отримання особливих точок на зображенні. Реалізувати програмне забезпечення, що дозволить в автоматичному режимі отримувати розташування цих точок. Дослідити можливі області застосування даного методу.

4.2 Підготовка до виконання роботи

Під час підготовки до виконання роботи слід згадати матеріал, що стосується мови програмування Object Pascal – базової мови програмування Delphi.

Детектування кутів (corner detection) або особливих точок представляє собою широкий клас методів, котрі знаходять застосування для отримання певної інформації із зображення.

Таким чином особлива точка, або характерна точка, повинна мати високу локальну інформативність. Також така точка повинна бути виділена на зображенні з врахуванням різних масштабів, поворотів, умов зйомки і т.п.

Наразі існує значна кількість алгоритмів детектування цих особливих точок, а саме: оператор Моравека, оператор Харіса, детектор Хедлі та Трайковича, FAST та інші.

В даній лабораторній роботі буде використовуватися один з найперших створених алгоритмів по детектуванню кутів – алгоритм Моравека. Суть алгоритму полягає в наступному: для кожного пікселя зображення, не враховуючі крайні пікселі (в залежності від розміру оператора), розраховується

мінімальне значення (сума квадратів різниць ділянок) серед восьми можливих зміщень відносно вхідної області. Якщо це число не значне, то це свідчить про схожість зміщеної ділянки та початкової. У випадку, коли піксель є особливою точкою, то жодна ділянка не буде схожа, а значить сума квадратів різниць ділянок буде значною величиною.

4.3 Порядок виконання роботи

4.3.1 Запустити програму Delphi

4.3.2 На диску у папці зі своїм прізвищем створити нову папку Lab4, у якій зберегти створюваний проект. У цій же папці зберегти зображення отримані у викладача.

4.3.3 У вікні Form1 розмістити 1 компонент типу TImage, 1 компонент типу TButton та 1 компонент типу TOpenPictureDialog. Для зображення встановити властивості Stretch та Proportional в True. Приклад вигляду інтерфейсу представлено на рис. 4.1.

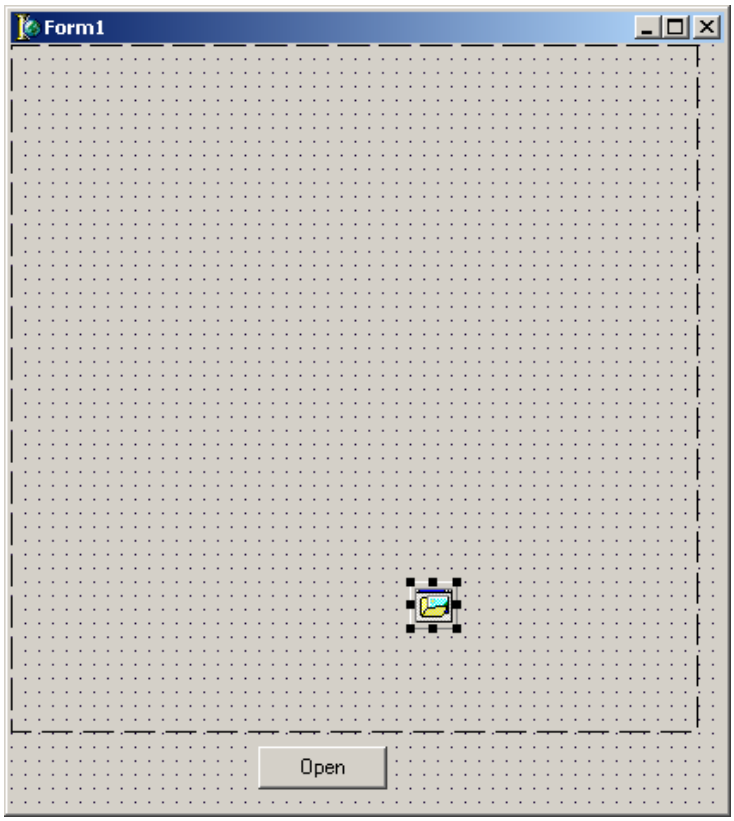

Рисунок 4.1 – Варіант вікна під час проектування інтерфейсу програми

4.3.4 Описати типи, які будемо використовувати для спрощеної роботи з зображенням. Для цього в розділі interface, після декларації форми, записати наступне.

```
PImageArray = ^TImageArray;
TImageArray = Array of Array of Byte;
PCorners = ^TCorners;
TCorners = Array of TPoint;
```

Як видно, тип TImageArray це дворозмірний масив, що може зберігати значення розміром в байт. У той же час, тип TCorners представляє собою масив точок.

4.3.5 Реалізувати функцію по завантаженню зображення з класу TBitmap в клас TImageArray. Для цього в розділі interface описати її заголовок:

```
procedure BitmapToImageArray(const Bitmap: TBitmap; const imageArray: PImageArray);
```

А також в розділі реалізації (implementation) описати сааме тіло функції:

```
procedure BitmapToImageArray(const Bitmap: TBitmap; const imageArray: PImageArray);
var
  i,j,w,h: Integer;
begin
  if (imageArray=nil) then raise Exception.Create('Null Pointer Exception!');
  w := Bitmap.Width;
  h := Bitmap.Height;
  SetLength(imageArray^,h,w);
  for j:=0 to h-1 do
  begin
    for i:=0 to w-1 do
    begin
      imageArray^[j][i] := GetRValue(Bitmap.Canvas.Pixels[i,j]);
    end;
  end;
end;
```

Як видно з представленного лістингу, функція отримує два параметра – TBitmap та вказівник на TImageArray. По даним параметра Bitmap відбувається заповнення поля imageArray.

4.3.6 Описати процедуру, котра відображала б отримані особливі точки (corners) на зображенні (bitmap). Заголовок:

```
procedure DrawCorners(const Bitmap: TBitmap; const corners: PCorners);
```

А також тіло процедури:

```
procedure DrawCorners(const Bitmap: TBitmap; const corners: PCorners);
const
  CIRCLE_RADIUS = 5;
var
  i: Integer;
begin
  if (corners=nil) then raise Exception.Create('Null Pointer Exception!');
  Bitmap.Canvas.Pen.Width := 2;
  Bitmap.Canvas.Pen.Color := clGreen;
  Bitmap.Canvas.Brush.Style := bsClear;
  for i:=0 to High(corners^)do
  begin
    Bitmap.Canvas.Ellipse(corners^[i].X - CIRCLE_RADIUS,
                          corners^[i].Y - CIRCLE_RADIUS,
                          corners^[i].X + CIRCLE_RADIUS,
                          corners^[i].Y + CIRCLE_RADIUS);
  end;
end;
```

4.3.7 Описати процедуру, котра розраховує особливі точки (corners) виходячи з вхідного зображення, а також параметрів. Для цього слід описати заголовок процедури:

```
procedure FindMoravecCorners(const imageArray: PImageArray; const corners: PCorners; const threshold: Integer = 10000; const size: Integer = 3);
```

І звісно ж тіло:

```
procedure FindMoravecCorners(const imageArray: PImageArray;
                             const corners: PCorners;
                             const threshold: Integer;
                             const size: Integer);
  procedure clearCorners();
  begin
    SetLength(corners^,0);
  end;
  procedure addCorner(const x,y: Integer);
  begin
    SetLength(corners^,Length(corners^)+1);
```

```
      corners^[High(corners^)].X := x;
      corners^[High(corners^)].Y := y;
    end;
  const
    wShift: Array [0..8-1] of TPoint = ((X:-1;Y:-1),(X: 0;Y:-1),(X: 1;Y:-1),
  (X:-1;Y: 0),(X: 1;Y: 0),(X:-1;Y: 1),(X: 0;Y: 1),(X: 1;Y: 1));
  var
    aShift: Array of TPoint;
  procedure generateAShift();
  var
    r: integer;
    i,j: Integer;
  begin
    r := size div 2;
    SetLength(aShift,size*size);
    for i:=0 to size-1 do
    begin
      for j:=0 to size-1 do
      begin
        aShift[j+size*i].X := j-r;
        aShift[j+size*i].Y := i-r;
      end;
    end;
  end;
  var
    w,h,x,y,direction,i: Integer;
    radius: Integer;
    minimal: Integer;
    cx,cy: Integer;
    total: Integer;
    p1,p2: TPoint;
    v1,v2: Integer;
  begin
    if (imageArray=nil)or(corners=nil) then raise Exception.Create('Null Pointer
Exception!');
    if (size<3)or(size mod 2 = 0) then Exception.Create('Illegal Argument
Exception!');
```

```
  clearCorners();
  generateAShift();
  radius := SIZE div 2;
  h := Length(imageArray^);
  w := Length(imageArray^[0]);
  for y:=radius to h-radius-1 do
  begin
    for x:= radius to w-radius-1 do
    begin
      minimal := MaxInt;
      for direction:=0 to High(wShift) do
      begin
        cy := y + wShift[direction].Y;
        cx := x + wShift[direction].X;
        if (cy<radius)or(cy>=h-radius)or(cx<radius)or(cy>=w-radius)then Continue;
        total := 0;
        for i:=0 to High(aShift)do
        begin
          p1.X := x + aShift[i].X;
          p1.Y := y + aShift[i].Y;
          v1 := imageArray^[p1.Y][p1.X];
          p2.X := cx + aShift[i].X;
          p2.Y := cy + aShift[i].Y;
          v2 := imageArray^[p2.Y][p2.X];
          total := total + (v1-v2)*(v1-v2);
        end;
        if total<minimal then minimal := total;
      end;
      if minimal>threshold then addCorner(x,y);
    end;
  end;
end;
```

Як видно з лістингу, дана процедура має 3 вкладені процедури. Перша очищує масив з знайденими точкам, друга в залежності від параметрів додає особливу точку. Третя процедура призначена для генерації масиву зміщень.

4.3.8 Реалізувати обробник події по пошуку особливих точок зображення, що завантажується. Для цього слід натиснути двічі по кнопці і у вікні редагування коду описати наступне:

```
if not OpenPictureDialog1.Execute then Exit;
Image1.Picture.Bitmap.LoadFromFile(OpenPictureDialog1.FileName);
Image1.Picture.Bitmap.PixelFormat := pf24bit;
BitmapToImageArray(Image1.Picture.Bitmap,@aimage1);
FindMoravecCorners(@aimage1,@corners1,);
DrawCorners(Image1.Picture.Bitmap,@corners1);
```

4.3.9 Запустити программу на виконання, приклад роботи програми представлений на рис. 4.2. Використовуючи прості зображення (прямокутники на однорідному фоні), дослідити роботу програми при повороті та інвертуванні кольорів. Пояснити отримані результати.

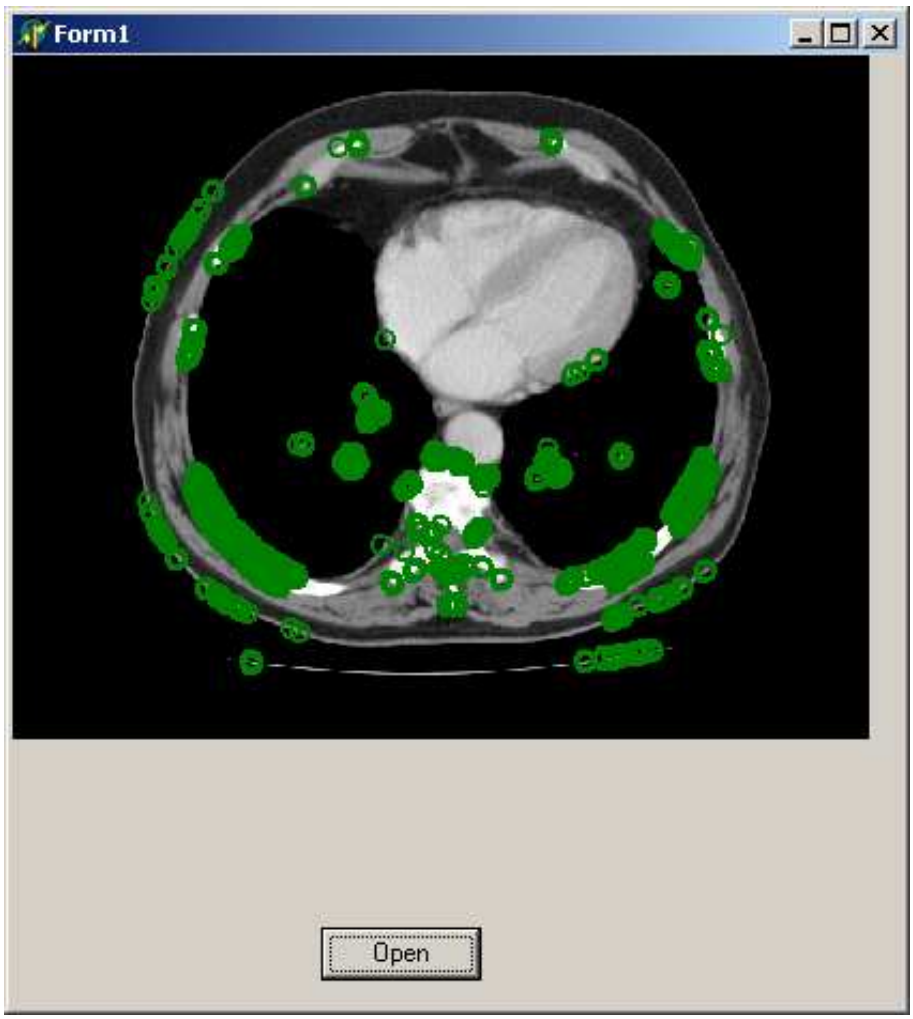

Рисунок 4.2 – Результат роботи програми

4.4 Зміст звіту

У звіті наводять:

– назву роботи;

– мету роботи;

– зображення інтерфейсу програми;

– результати роботи програми для різних зображень та різних його модифікаціях;

– лістинг програми;

– висновки.

4.5 Контрольні питання

1. Що таке точкові особливості зображення?
2. Для чого призначений оператор Моравека?
3. Які алгоритми детектування особливих точок Вам відомі?
4. Опишість можливі області застосування даного алгоритму.
5. Чи до усіх змін пристосований оператор Моравека?
6. Чи має алгоритм Моравека недоліки, якщо має то які?

5 РЕКОНСТРУКЦІЯ ОБ’ЄКТУ МЕТОДОМ ЛАЗЕРНОГО СКАНУВАННЯ

5.1 Мета роботи

Ознайомитися із спрощеним методом лазерного сканування, який призначений для реконструкції об’єктів. Дослідити можливі області застосування даного методу в медицині.

5.2 Підготовка до виконання роботи

Під час підготовки до виконання роботи слід згадати матеріал, що стосується мови програмування Object Pascal – базової мови програмування Delphi. Також слід ознайомитися з бібліотекою OpenGL.

Технології лазерного сканування для реконструкції об’єктів знайшли сьогодні широке використання, у тому числі і в медицині. В найпростішому випадку, робота зводиться до реєстрації зображення профіля об’єкту (рис. 5.1).

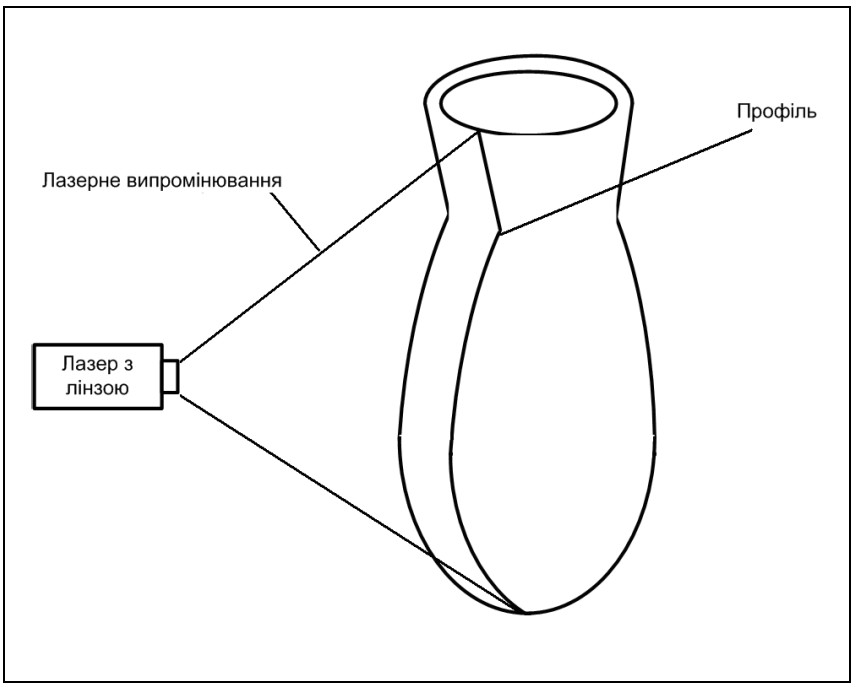

Рисунок 5.1 – Приклад отримання зображення

Таким чином, реєструючи профіль навколо об’єкту (360 °) можна реконструювати об’єкт. На рис. 5.2 зображений приклад проекту David по реконструкції поверхні об’єктів.

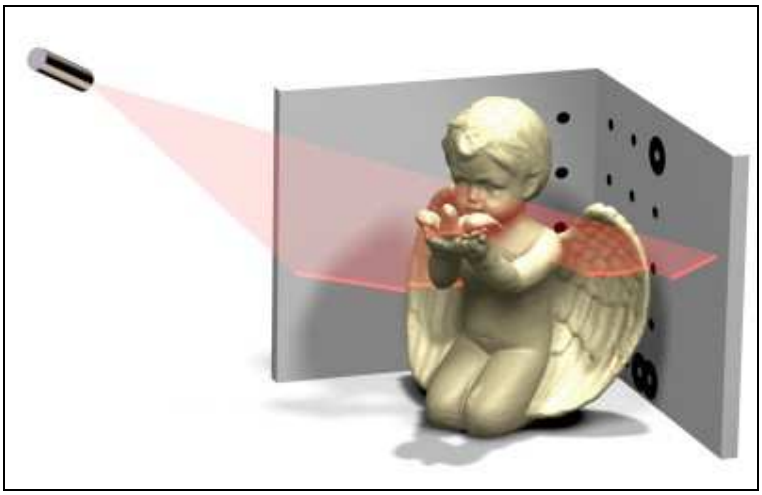

Рисунок 5.2 – Проект David для сканування 3D об’єктів

В лабораторній роботі, у якості вхідних даних, будуть використовуватися згенеровані зображення (одне зображення для усіх 360 °). В дійсності для впуклих об’єктів задача ускладнюється, окрім того, профіль не однорідний та має певний градієнт.

5.3 Порядок виконання роботи

5.3.1 Запустити програму Delphi

5.3.2 На диску у папці зі своїм прізвищем створити нову папку Lab5, у якій зберегти створюваний проект. У цій же папці зберегти зображення отримане у викладача. Також скопіювати заголовочний файл dglOpenGL, котрий містить все, що необхідно для роботи OpenGL.

5.3.3 У вікні Form1 розмістити 1 компонент типу TTimer з вкладки System. Встановити властивість Interval рівну 100, а також Enabled – False. Приклад вигляду інтерфейсу представлено на рис. 5.1.

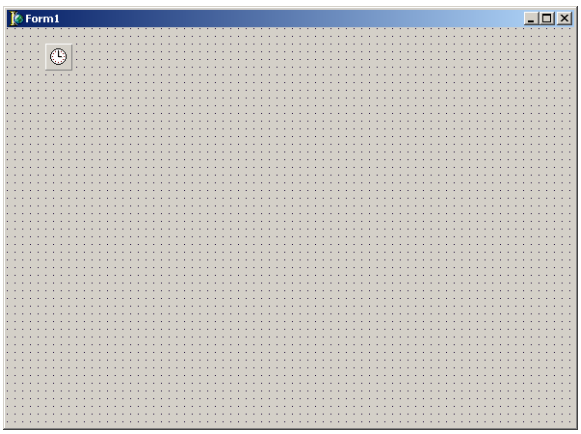

Рисунок 5.1 – Варіант вікна під час проектування інтерфейсу програми

5.3.4 Підключити бібліотеку OpenGL до свого проекту, для цього в модулі головної форми, в розділі uses додати модуль dglOpenGL.pas:

```
uses
  …,
  dglOpenGL;
```

5.3.5 В описі головної форми, головній формі додати наступні поля:

```
  TForm1 = class(TForm)
    …
  private
    FDC: HDC;
    FRC: HGLRC;
  public
    …
  end;
```

В них буде зберігатись інформація про контекст відображення Windows та OpenGL.

5.3.6 Описати змінну в якій будемо зберігати профіль, що зареєстрували (в даному випадку профіль буде ідентичним для усіх 360 °):

```
var
  Form1: TForm1;
  profile: Array of TPoint;
```

5.3.7 Створити обробник події при створенні форми:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  bmp: TBitmap;
  i,j: Integer;
begin
  FDC := GetDC(Handle);
  FRC := CreateRenderingContext(FDC,[opDoubleBuffered],32,24,0,0,0,0);
  ActivateRenderingContext(FDC,FRC);
  glClearColor(0.0,0.0,0.0,1.0);
  glEnable(GL_DEPTH_TEST);
  ReadOpenGLCore();
  SetLength(profile,0);
  bmp := TBitmap.Create();
  bmp.LoadFromFile(ExtractFilePath(Application.ExeName)+'profile.bmp');
  for i:=0 to bmp.Height-1 do
  begin
    for j:=0 to bmp.Width-1 do
    begin
      if bmp.Canvas.Pixels[j,i]=clRed then
      begin
        SetLength(profile,Length(profile)+1);
        profile[High(profile)].X := j;
        profile[High(profile)].Y := bmp.Height-1-i;
      end;
    end;
  end;
  FreeAndNil(bmp);
  Timer1.Enabled := True;
end;
```

Як видно з вищенаведеного лістингу, спочатку отримується контекст Windows для головного вікна (вікна в який буде відбуватися відображення 3D сцени). Далі, для цього контексту створюється контекст OpenGL з відповідними налаштуваннями. І відразу ж цей контекст активується. В наступній частині відбувається встановлення налаштувань для машини станів OpenGL. А також зчитування функцій ядра OpenGL. Після зчитується зображення профілю і за допомогою послідовного сканування будується профіль моделі. І звісно ж дозволяється рендерінг сцени.

5.3.8 Реалізувати обробник події по знищенню форми (OnDestroy).

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  Timer1.Enabled := False;
  DeactivateRenderingContext();
  DestroyRenderingContext(FRC);
  ReleaseDC(Handle,FDC);
end;
```

5.3.9 Реалізувати обробник події OnTimer для таймеру:

```
procedure TForm1.Timer1Timer(Sender: TObject);
var
  i,p: Integer;
begin
  glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if Width>Height then gluPerspective(90, Width/Height, 0.4,1000)
  else gluPerspective(90, Height/Width, 0.4,1000);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glViewport(0,0,Width,Height);
  glTranslatef(0,-20,0);
  glRotatef(90,0,1,0);
  glTranslatef(100,0,0);
  glColor3f(1.0,1.0,1.0);
  glBegin(GL_LINES);
  for i:=-10 to 10 do
  begin
    glVertex3f(i*10,0,-100);
```

```
  glVertex3f(i*10,0, 100);
 end;
 for i:=-10 to 10 do
 begin
  glVertex3f(-100,0,i*10);
  glVertex3f( 100,0,i*10);
 end;
 glEnd();
 glPushMatrix();
 glColor3f(1.0,0.0,0.0);
 for i:=0 to 360 do
 begin
  glRotatef(1.0,0,1,0);
  glBegin(GL_LINES);
  for p:=0 to High(profile)do
  begin
    glVertex3f(profile[p].X*0.3,profile[p].Y*0.3,0.0);
  end;
  glEnd();
 end;
 glPopMatrix();
 SwapBuffers(FDC);
end;
```

5.3.10 Запустити программу на виконання, приклад роботи програми представлений на рис. 5.3. Змінити крок сканування з 1 градусу до більшої величини. Описати це. Модифікувати програму таким чином, щоб для однієї четверті використовувався один профіль, для інших трьох – зовсім інший (Для цього необхідно попередньо завантажити ще один профіль перед його відображенням).

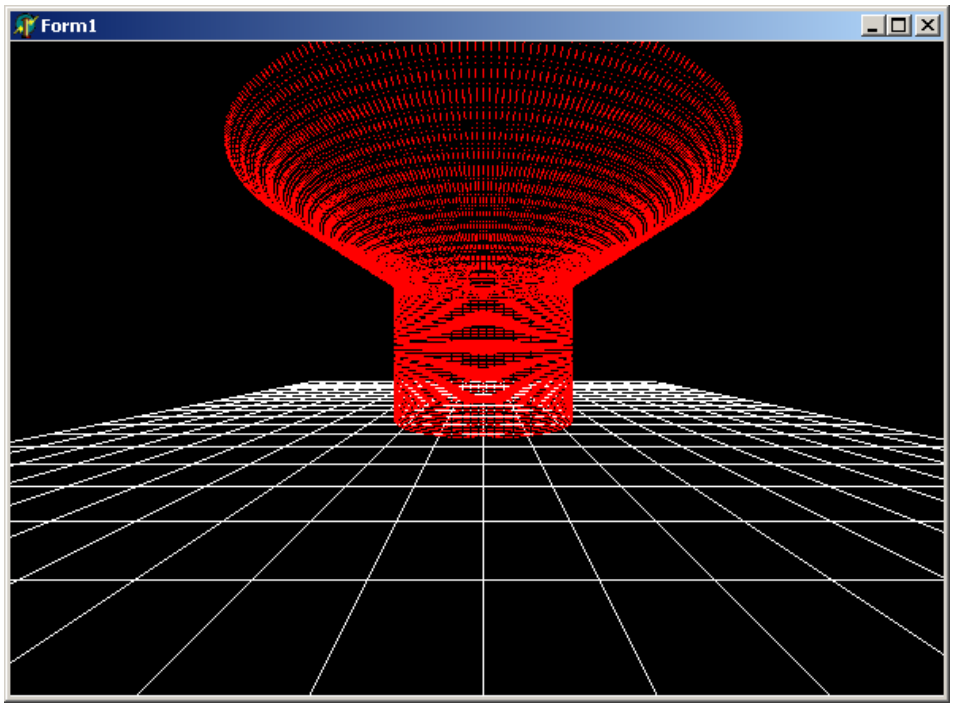

Рисунок 5.3 – Результат роботи програми

5.4 Зміст звіту

У звіті наводять:

– назву роботи;
– мету роботи;
– зображення інтерфейсу програми;
– результати роботи програми для різних умов;
– лістинг програми;
– висновки.

5.5 Контрольні питання

1. Поясніть роботу методу лазерного сканування?
2. Де застосовується даний метод?
3. Наведіть приклади застосування лазерного сканування в медицині.
4. Що таке OpenGL.
5. Які функції OpenGL Ви використовували.
6. Яким чином Ви вдосконалили програму для того щоб отримати необхідні результати?
7. Опишіть наступні кроки по створенню більш вдосконаленого 3D сканеру.

6 ПІДБІР ПОРОГУ СЕГМЕНТАЦІЇ ЗОБРАЖЕННЯ З ВИКОРИСТАННЯМ ГЕНЕТИЧНОГО АЛГОРИТМУ

6.1 Мета роботи

Ознайомитися із можливістю використання генетичних алгоритмів у обробці медичних зображень. Розробити програмне забезпечення по підбору порогів сегментації в просторі HSV по зображенню та прикладу його сегментації (маски).

6.2 Підготовка до виконання роботи

Генетичні алгоритми – це клас алгоритмів, які базуються на еволюційному пошуці. Дані алгоритми знаходять своє застосування в задачах оптимізації шляхом почергового підбору результату, використовуючи комбінування елементів, що чимось нагадують еволюцію. Звідси і пішла назва.

У генетичних алгоритмах виділяють наступні етапи:

– генерація початкових особин;

– розрахунок їх пристосованості;

– видалення не пристосованих;

– схрещення особин + мутація;

В даній роботі використовуватиметься наступні зображення в якості вхідних даних (рис. 6.1)

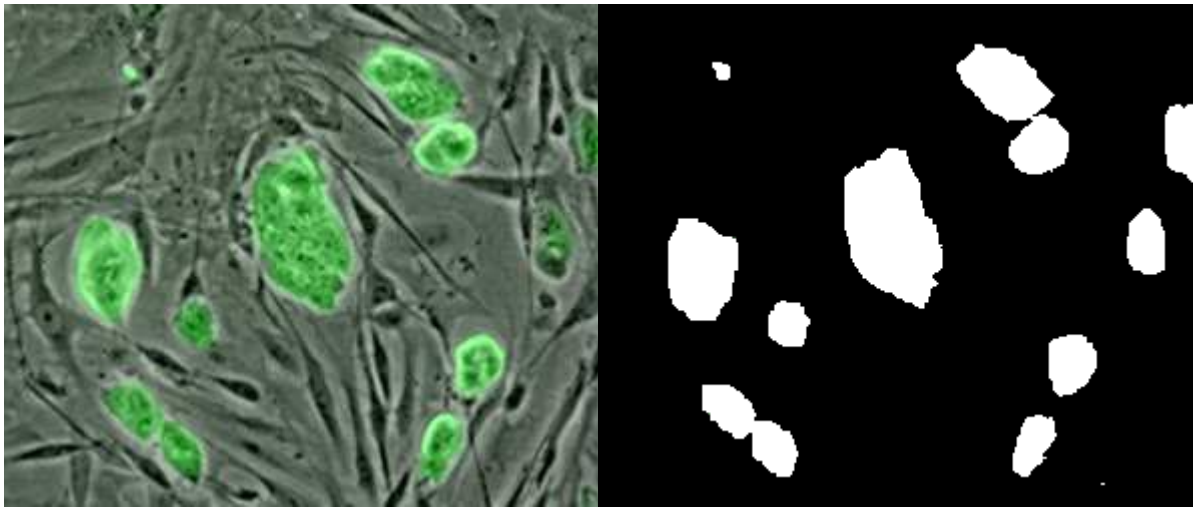

Рисунок 6.1 – Вхідне зображення та його маска

Суть задачі полягає в обчисленні порогів сегментації зображення у просторі HSV, маючи приклад зображення та його сегментації.

6.3 Порядок виконання роботи

6.3.1 Запустити програму Delphi

6.3.2 На диску у папці зі своїм прізвищем створити нову папку Lab6, у якій зберегти створюваний проект. У цій же папці зберегти зображення отримані у викладача.

6.3.3 У вікні Form1 розмістити 3 компонент типу. Встановити властивість їм усім властивість Stretch та Proportional рівну в True. Назвати ці зображення наступним чином – iSrc, iMask, iDst. Встановити два компоненти типу TOpenPictureDialog з вкладки Dialogs і назвати їх opdOpenSrc та opdOpenMask. Розмістити два компоненти TButton та назвати їх bRun і bApply. Приклад вигляду інтерфейсу представлено на рис. 6.2.

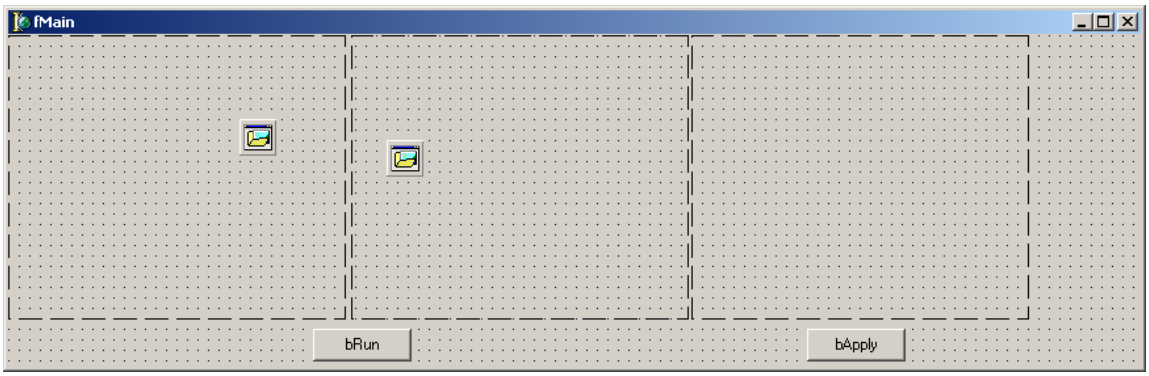

Рисунок 6.2 – Варіант вікна під час проектування інтерфейсу програми

6.3.4 Створити новий котрий реалізує генетичний алгоритм, для цього натиснути пункт меню File->New->Unit. Зберегти модуль в папці з програмою під іменем uGenetic. Лістинг модулю наведено нижче:

```
unit uGenetic;
interface
uses
  SysUtils,
  Classes,
  Graphics;
type
```

```
  TGen = class
  private
    hmin,hmax,
    smin,smax,
    vmin,vmax: Byte;
  public
    constructor Create(const Ahmin,Ahmax,Asmin,Asmax,Avmin,Avmax: Byte);
    function Segmentation(const H,S,V: Byte): Boolean;
  end;
  TPerson = class
  private
    gen: TGen;
    live: Boolean;
  protected
    function GetDifference(const original,mask: TBitmap): Integer;
  public
    constructor Create(AGen: TGen);
    destructor Destroy(); override;
    property IsLive: Boolean read live;
  end;
  TGeneticAlgorithm = class
  private
    persons: Array of TPerson;
    currIteration: Integer;
    maxIteration: Integer;
    mortalityRate: Integer;
    Result: TGen;
  public
    constructor Create(const personCount: Integer = 1000;
                       const mortalityRate: Integer = 30;
                       const maxiteration: Integer = 1000);
    procedure Run(const original,mask: TStringList); overload;
    procedure Run(const original, mask: String); overload;
    procedure Segmentation(const Src: TBitmap; const Dst: TBitmap);
    destructor Destroy();override;
  end;
function BringToTheWorldGen(const g1,g2: TGen): TGen;
```

```
function BringToTheWorldPerson(const p1,p2: TPerson): TPerson;
function BringToTheWorldPersonRandom(): TPerson;
implementation
function BringToTheWorldGen(const g1,g2: TGen): TGen;
  function IntegrateParams(const p1,p2: Byte): Byte;
  begin
    //        mother & father & mutation
    Result := Trunc((p1 + p2)/2 + 100*(-0.1+Random(200)/1000));
  end;
var
  hmin,hmax,
  smin,smax,
  vmin,vmax: Byte;
  t: Byte;
begin
  if (g1=nil)or(g2=nil)then raise Exception.Create('Null Pointer Exception');
  hmin := IntegrateParams(g1.hmin,g2.hmin);
  hmax := IntegrateParams(g1.hmax,g2.hmax);
  if hmin>hmax then
  begin
    t := hmin;
    hmin := hmax;
    hmax := t;
  end;
  smin := IntegrateParams(g1.smin,g2.smin);
  smax := IntegrateParams(g1.smax,g2.smax);
  if smin>smax then
  begin
    t := smin;
    smin := smax;
    smax := t;
  end;
  vmin := IntegrateParams(g1.vmin,g2.vmin);
  vmax := IntegrateParams(g1.vmax,g2.vmax);
  if vmin>vmax then
  begin
    t := vmin;
```

```
    vmin := vmax;
    vmax := t;
  end;
  Result := TGen.Create(hmin,hmax,smin,smax,vmin,vmax);
end;
function BringToTheWorldPerson(const p1,p2: TPerson): TPerson;
begin
  Result := TPerson.Create(BringToTheWorldGen(p1.gen,p2.gen));
end;
function BringToTheWorldPersonRandom(): TPerson;
var
  hmin,hmax,
  smin,smax,
  vmin,vmax: Byte;
  t: Byte;
begin
  hmin := Random(256);
  hmax := Random(256);
  if hmin>hmax then
  begin
    t := hmin;
    hmin := hmax;
    hmax := t;
  end;
  smin := Random(256);
  smax := Random(256);
  if smin>smax then
  begin
    t := smin;
    smin := smax;
    smax := t;
  end;
  vmin := Random(256);
  vmax := Random(256);
  if vmin>vmax then
  begin
    t := vmin;
```

```
    vmin := vmax;
    vmax := t;
  end;
  Result := TPerson.Create(TGen.Create(hmin,hmax,smin,smax,vmin,vmax));
end;
{ TGen }
constructor TGen.Create(const Ahmin, Ahmax, Asmin, Asmax, Avmin,
  Avmax: Byte);
begin
  hmin := Ahmin;
  hmax := Ahmax;
  smin := Asmin;
  smax := Asmax;
  vmin := Avmin;
  vmax := Avmax;
end;
function TGen.Segmentation(const H, S, V: Byte): Boolean;
begin
  Result := (H in [hmin..hmax])and(S in [smin..smax])and(V in [vmin..vmax]);
end;
{ TPerson }
constructor TPerson.Create(AGen: TGen);
begin
  gen := AGen;
  live := True;
end;
destructor TPerson.Destroy;
begin
  FreeAndNil(gen);
  inherited;
end;
  function Min(const v1,v2: Byte): Byte;overload;
  begin
    if v1>v2 then Result := v2
    else Result := v1;
  end;
  function Max(const v1,v2: Byte): Byte;overload;
```

```
begin
  if v1>v2 then Result := v1
  else Result := v2;
end;
function Min(const v1,v2: Single): Single;overload;
begin
  if v1>v2 then Result := v2
  else Result := v1;
end;
function Max(const v1,v2: Single): Single;overload;
begin
  if v1>v2 then Result := v1
  else Result := v2;
end;
procedure RGB2HSV(const r,g,b: Single; var h,s,v: Single);
var
  maxv,minv: Single;
begin
  maxv := Max(r,Max(g,b));
  minv := Min(r,Min(g,b));
  if maxv=minv then
  begin
    h := 0;
  end else
  if (maxv = r)and(g>=b) then
  begin
    h := 60*(g-b)/(maxv-minv);
  end else
  begin
    if (maxv = r)and(g<b) then
    begin
      h := 60*(g-b)/(maxv-minv)+360;
    end else
    begin
      if maxv = g then
      begin
        h := 60*(b-r)/(maxv-minv)+120;
```

```
        end else
        begin
          h := 60*(r-g)/(maxv-minv)+240;
        end;
      end;
    end;
    v := maxv;
    if maxv=0 then
    begin
      s := 0;
    end else begin
      s := 1-minv/maxv;
    end;
    h := h/360;
  end;
type
  TBGR = record
    b,g,r: Byte;
  end;
  PArrayBGR = ^TArrayBGR;
  TArrayBGR = Array [0..0]of TBGR;
function TPerson.GetDifference(const original, mask: TBitmap): Integer;
var
  x,y,w,h: Integer;
  prow1,prow2: PArrayBGR;
  th,ts,tv: Single;
  handssegmentation: Boolean;
begin
  w := original.Width;
  h := original.Height;
  Result := 0;
  for y:=0 to h-1 do
  begin
    prow1 := original.ScanLine[y];
    prow2 := mask.ScanLine[y];
    for x:=0 to w-1 do
    begin
```

```
        RGB2HSV(prow1[x].r/255,prow1[x].g/255,prow1[x].b/255,th,ts,tv);
        handssegmentation := (prow2[x].b = 255)and(prow2[x].g = 255)and(prow2[x].r = 255);
        if gen.Segmentation(Trunc(255*th),Trunc(255*ts),Trunc(255*tv))<>handssegmentation then Inc(Result);
       end;
     end;
   end;
   { TGeneticAlgorithm }
   constructor TGeneticAlgorithm.Create(const personCount, mortalityRate,
    maxiteration: Integer);
   var
    i: Integer;
   begin
    curriteration := 0;
    Self.maxiteration := maxiteration;
    SetLength(persons,personCount);
    for i:=0 to High(persons)do
    begin
     persons[i] := BringToTheWorldPersonRandom();
    end;
    Self.mortalityRate := mortalityRate;
    if Self.mortalityRate>95 then Self.mortalityRate := 95;
    if Self.mortalityRate<5 then Self.mortalityRate := 5;
   end;
   destructor TGeneticAlgorithm.Destroy;
   var
    i: Integer;
   begin
    for i:=0 to High(persons) do
    begin
     FreeAndNil(persons[i]);
    end;
    SetLength(persons,0);
    inherited;
   end;
```

```
procedure TGeneticAlgorithm.Run(const original, mask: TStringList);
type
  TChunk = record
    rate: Integer;
    position: Integer;
  end;
var
  bmp1,bmp2: TBitmap;
  balls: Array of TChunk;
  temp: TChunk;
  i,j: Integer;
  image: Integer;
  parentCount: Integer;
  CountOfDeath: Integer;
  bmps: Array of record
    o,m: TBitmap;
  end;
begin
  bmp1 := TBitmap.Create();
  bmp2 := TBitmap.Create();
  SetLength(bmps,original.Count{-1});
  for image:=0 to High(bmps)do
  begin
    bmps[image].o := TBitmap.Create();
    bmps[image].o.LoadFromFile(original[image]);
    bmps[image].o.PixelFormat := pf24bit;
    bmps[image].m := TBitmap.Create();
    bmps[image].m.LoadFromFile(mask[image]);
    bmps[image].m.PixelFormat := pf24bit;
  end;
  while (curriteration<=maxiteration) do
  begin
    Writeln('CurrIteration: ',currIteration,' - start.');
    Writeln('   count of persons: ',Length(persons));
    SetLength(balls,Length(persons));
    for i:=0 to High(balls)do
    begin
```

```
        balls[i].rate := 0;
        balls[i].position := i;
      end;
      for i:=0 to High(balls)do
      begin
        for image:=0 to High(bmps) do
        begin
          Writeln('image - '+IntToStr(image));
          balls[i].rate := balls[i].rate +
persons[i].GetDifference(bmps[image].o,bmps[image].m);
        end;
      end;
      for i:=0 to High(balls)-1 do
      begin
        for j:=0 to High(balls)-1 do
        begin
          if balls[j].rate>balls[j+1].rate then
          begin
            temp := balls[j+1];
            balls[j+1] := balls[j];
            balls[j] := temp;
          end;
        end;
      end;
      CountOfDeath := Trunc(Length(persons)*mortalityRate/100);
      //Sort
      for i:=High(balls)-CountOfDeath to High(balls)do
      begin
        persons[balls[i].position].live := false;
      end;
      //Death...
      for i:=High(persons) downto 0 do
      begin
        if not persons[i].IsLive then
        begin
          FreeAndNil(persons[i]);
          for j:=i+1 to High(persons) do
```

```
          begin
            persons[j-1] := persons[j];
          end;
        end;
      end;
      SetLength(persons,Length(persons)-CountOfDeath-1);
      parentCount := Length(persons);
      for i:=0 to CountOfDeath do
      begin
        SetLength(persons,Length(persons)+1);
        persons[High(persons)]                                                    :=
BringToTheWorldPerson(persons[Random(parentCount)],persons[Random(parentCo
unt)]);
      end;
      Writeln('   best result: ',balls[0].rate,' in person: ',balls[0].position);
      Writeln('   param of this person: ');
      Writeln('                hmin = ', persons[balls[0].position].gen.hmin);
      Writeln('                hmax = ', persons[balls[0].position].gen.hmax);
      Writeln('                smin = ', persons[balls[0].position].gen.smin);
      Writeln('                smax = ', persons[balls[0].position].gen.smax);
      Writeln('                vmin = ', persons[balls[0].position].gen.vmin);
      Writeln('                vmax = ', persons[balls[0].position].gen.vmax);
      Writeln('CurrIteration: ',currIteration,' - finish.');
      Inc(curriteration);
      if balls[0].rate=0 then
      begin
        Break;
      end;
    end;
    Result := TGen.Create(persons[balls[0].position].gen.hmin,
                          persons[balls[0].position].gen.hmax,
                          persons[balls[0].position].gen.smin,
                          persons[balls[0].position].gen.smax,
                          persons[balls[0].position].gen.vmin,
                          persons[balls[0].position].gen.vmax);
    SetLength(balls,0);
    bmp1.Free();
```

```
  bmp2.Free();
  SetLength(bmps,original.Count-1);
  for image:=0 to High(bmps)do
  begin
    bmps[image].o.Free();
    bmps[image].m.Free();
  end;
  SetLength(bmps,0);
end;
procedure TGeneticAlgorithm.Run(const original, mask: String);
var
  o,m: TStringList;
begin
  o := TStringList.Create();
  o.Add(original);
  m := TStringList.Create();
  m.Add(mask);
  Run(o,m);
  FreeAndNil(o);
  FreeAndNil(m);
end;
procedure TGeneticAlgorithm.Segmentation(const Src, Dst: TBitmap);
var
  prow1,prow2: PArrayBGR;
  x,y,w,h: Integer;
  th,ts,tv: Single;
begin
  Src.PixelFormat := pf24bit;
  Dst.PixelFormat := pf24bit;
  Dst.Width := Src.Width;
  Dst.Height := Src.Height;
  w := Src.Width;
  h := Src.Height;
  for y:= 0 to h-1 do
  begin
    prow1 := Src.ScanLine[y];
    prow2 := Dst.ScanLine[y];
```

```
      for x:= 0 to w-1 do
      begin
        RGB2HSV(prow1[x].r/255,prow1[x].g/255,prow1[x].b/255,th,ts,tv);
        if Result.Segmentation(Trunc(255*th),Trunc(255*ts),Trunc(255*tv))then
        begin
          prow2[x].r := 255; prow2[x].g := 255; prow2[x].b := 255;
        end
        else
        begin
          prow2[x].r :=   0; prow2[x].g :=   0; prow2[x].b :=   0;
        end;
      end;
    end;
  end;
  end.
```

6.3.5 Підключити модуль до модуля головної форми. Для цього у модулі головної форми додати:

```
  uses
    …,
    uGenetic;
```

6.3.6 Створити обробник події при натисненні на перше зображення:

```
    if not opdOpenSrc.Execute then Exit;
    iSrc.Picture.Bitmap.LoadFromFile(opdOpenSrc.FileName);
```

6.3.7 Створити обробник події при натисненні на друге зображення:

```
    if not opdOpenMask.Execute then Exit;
    iMask.Picture.Bitmap.LoadFromFile(opdOpenMask.FileName);
```

6.3.8 Додати глобальну змінну, що описує генетичний алгоритм:

```
    ga: TGeneticAlgorithm;
```

6.3.9 Реалізувати обробник події OnCreate для форми, що має наступний лістинг:

```
    ga := TGeneticAlgorithm.Create(50,30,10);
```

6.3.10 Також створити обробник події OnDestroy для форми:

```
    FreeAndNil(ga);
```

6.3.11 Створити обробник події OnClick для першої кнопки:

```
    ga.Run(opdOpenSrc.FileName,opdOpenMask.FileName);
```

6.3.12 Створити обробник події OnClick для другої кнопки:

```
    ga.Segmentation(iSrc.Picture.Bitmap,iDst.Picture.Bitmap);
```

6.3.13 Дозволити програмі консольний вивід інформації. Для цього перейти в меню: Project->View Source та дописати наступне:

{$APPTYPE CONSOLE}

6.3.14 Запустити программу на виконання, приклад роботи програми представлений на рис. 6.3.

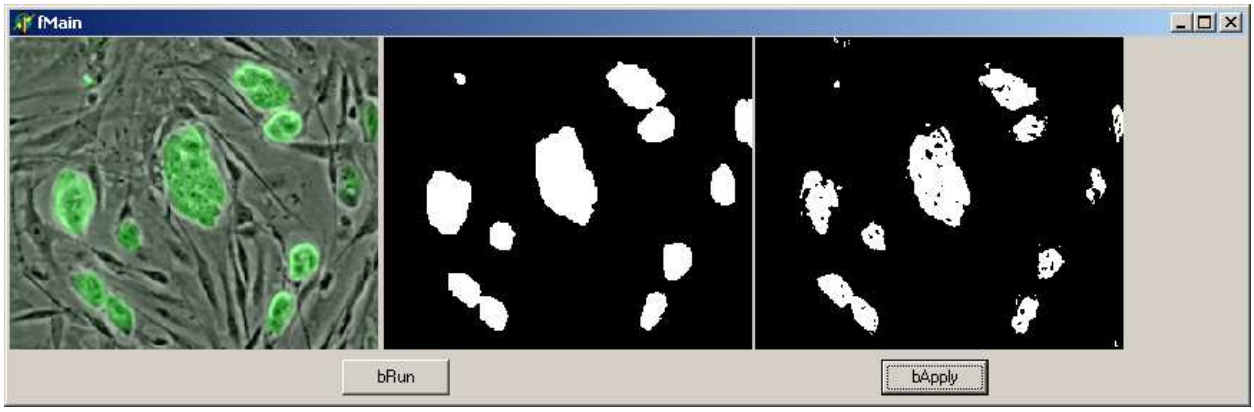

Рисунок 6.3 – Результат роботи програми

6.4 Зміст звіту

У звіті наводять:

– назву роботи;
– мету роботи;
– зображення інтерфейсу програми;
– результати роботи програми;
– лістинг програми;
– висновки.

6.5 Контрольні питання

1. Поясніть роботу генетичних алгоритмів.
2. Для вирішення яких задач використовуються генетичні алгоритми?
3. Наведіть приклади застосування генетичних алгоритмів в медицині.
4. Які Ви знаєте недоліки генетичних алгоритмів?
5. Із яких етапів складаються генетичні алгоритми?
6. Для чого використовується мутація в генетичних алгоритмах?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Девід А. Форсайт, Жан Понс. Комп’ютерний зір. Сучасний підхід. – Вільямс, 2004. – 928 с

2. Красильников Н. Цифрова обробка 2D- та 3D-зображень. – БХВ-Петербург, 2011. – 608 с.

3. Бондарев В., Трьостер Г., Черненга В. Цифрова обробка сигналів: методи та засоби. Навч. посібник для вузів. – Х.: Конус, 2001. – 398 с.

4. Краснов М. В. Open GL графіка в проектах Delphi. – СПб.: БХВ-Петербург, 2001. – 352 с.

5. Візільтер Ю., Желтов С., Бондаренко А., Ососков М., Моржин А. Обробка та аналіз зображень в задачах машинного зору. – Фізматкнига, 2010. – 688 с.

6. Фолі Дж., Вен Дем А., Основи інтерактивной машинної графіки. Кн. 2 / Пер. з англ. В.А. Галактіонова, Ю.М. Лазутіна, О.Н. Родінко. – М.: Мир, 1985. – 368 с.

7. Претт У. Цифрова обробка зображень. Кн. 2 / Пер. з англ. Під ред. Д.С. Лебедєва. – М.: Мир, 1982. – 310 с.

8. Шліхт Г. Цифрова обробка кольорових зображень. – М.: Вид. ЕКОМ, 1997. – 336 с.

Електронне навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт

з дисципліни
“МЕТОДИ ТА ЗАСОБИ АНАЛІЗУ ЗОБРАЖЕНЬ”

для студентів усіх форм навчання
спеціальності 7.05140201, 8.05140201 «Біомедична інженерія»,
7.05140203, 8.05140203 «Інформаційні технології в біомедицині»

Упорядник АВРУНІН Олег Григорович

Відповідальний випусковий А.І. Бих

Авторська редакція