**Rīgas Tehniskā universitāte**

**Datorzinātnes un informācijas tehnoloģijas fakultāte**

Datorvadības, automātikas un datortehnikas institūts



**Atskaite par VIII. praktisko darbu**

priekšmeta "Scēnu analīze"

**Izstrādāja: Igors Šemels**

**Pārbaudīja: O. Krutikova**

**2012./2013.m.g.**

**Teorijas apraksts:**

1)Veidojam scēnu.

2)Nolasam informāciju no scēnas un ierakstām dinamiskajā masīvā.

3)Scēnas segmentācija. (Apgabalu pieaugšanas algoritms).

4)Objektu kontūru atrašana un izdalīšana. (Kukaiņa algoritms).

5)Objektu virsotņu atrašana. Adaptīvais kontūrs. Tiek iezīmētas un saskaitītas visas objekta virsotnes.

6)Virsotņu leņķu izskaitļošana. (Kosinusu teorēma).

7)Objektu klasifikācija pēc leņķu grādiem.

Pēc kukaiņa algoritma izpildīšanas tika saglabātas ārējā kontūra koordinātes.

1)Var salīdzināt katru kontūra koordināti pēc attāluma ar nulli un tādejādi meklēt vistālāko attālumu. Tas būs pirmais stūris.

2)Atrastais stūra attālums tiek salīdzināts pēc kārtas ar katru nākamo koordināti. Tā tiek meklēts otrais vistālākais attālums.

3)Kad šie divi stūri ir atrasti, starp vieņiem tiek meklēts vel viens vistālākais attālums gan pa pulksteņrādītāja virzienu, gan pret pulksteņradītāja virzienu.

Lai varētu klasificēt objektus, nepieciešams atcerēties būtiskās pazīmes ar ko atšķirās šie objekti. Piemēram, pie klases trijstūri pieder objekti, kuriem stūru skaits ir 3, un lai tālāk noteikt pie kāda veida pieder šie trijstūri, nepieciešams salīdzināt lēņķus.

Ar trapecem un taisnstūriem ir mazliet atšķirīga situācija. Piemēram, taisnstūrim un kvadratam leņķi ir vienādi ar 90°, bet malu garumi atšķirās vai ir vienādi. Trapecēm divas malas un diagonāles var sakrist (vienādsānu trapeces gadījumā), piemēram, ja ir taisnleņķa trapece, tad divi leņķi vienādi ar 90 grādiem un tt.

**Programmas pirmkods ar komentāriem**

procedure TMain\_menu.findAllVertixes();

var i,j,k, maxX, maxY, jD, vertCt: integer;

distance1, distance2, maxDist, vertDist: real;

begin

maxDist := 0;

maxX:=0;

maxY:=0;

for i:=0 to length(objects)-1 do

begin

distance1:=0;

distance2:=0;

vertexSet[i].is4Vert := false;

jD:=0;

for j:=0 to length(objects[i])-1 do

begin

distance1:= getDistanceBetween(objects[i][0].x,objects[i][0].y,objects[i][j].x, objects[i][j].y);

if (distance1 > maxDist) then

begin

maxDist:=distance1;

maxX:=objects[i][j].x;

maxY:=objects[i][j].y;

jD:=j;

end;

end;

img[maxX,maxY].segment:=2;

vertexSet[i].vertex[1].X:=maxX;

vertexSet[i].vertex[1].Y:=maxY;

maxDist:=0;

vertCt:=0;

vertDist:=0;

vertDist:= getDistanceBetween(vertexSet[i].vertex[0].X,vertexSet[i].vertex[0].Y, vertexSet[i].vertex[1].X, vertexSet[i].vertex[1].Y);

for k:=0 to jD do

begin

if (jD<length(objects[i])) then

begin

distance1:= getDistanceBetween(vertexSet[i].vertex[0].X,vertexSet[i].vertex[0].Y, objects[i][k].x, objects[i][k].y);

distance2:= getDistanceBetween(vertexSet[i].vertex[1].X,vertexSet[i].vertex[1].Y, objects[i][k].x, objects[i][k].y);

if ((distance1+distance2 > maxDist)) then

begin

maxDist:=distance1+distance2;

maxX:=objects[i][k].x;

maxY:=objects[i][k].y;

end; end; end;

if (abs(maxDist-vertDist)> 5) then

begin

img[maxX,maxY].segment:=2;

if (length(vertexSet) > i) then

begin

vertexSet[i].vertex[2].X:=maxX;

vertexSet[i].vertex[2].Y:=maxY;

vertCt:=vertCt+1;

end;

end;

maxDist:=0;

for k:=jD to length(objects[i])-1 do

begin

if (jD<length(objects[i])) then

begin

distance1:= getDistanceBetween(vertexSet[i].vertex[0].X,vertexSet[i].vertex[0].Y, objects[i][k].x, objects[i][k].y);

distance2:= getDistanceBetween(vertexSet[i].vertex[1].X,vertexSet[i].vertex[1].Y, objects[i][k].x, objects[i][k].y);

if ((distance1+distance2 > maxDist)) then

begin

maxDist:=distance1+distance2;

maxX:=objects[i][k].x;

maxY:=objects[i][k].y;

end;

end;

end;

if (abs(maxDist-vertDist)> 5) then

begin

img[maxX,maxY].segment:=2;

if (length(vertexSet) > i) then

begin

if (vertCt=0) then

begin

vertexSet[i].vertex[2].X:=maxX;

vertexSet[i].vertex[2].Y:=maxY;

end else

begin

vertexSet[i].vertex[3].X:=maxX;

vertexSet[i].vertex[3].Y:=maxY;

end;

vertCt:=vertCt+1;

end;

end;

maxDist:=0;

if(vertCt=2) then vertexSet[i].is4Vert := true;

end;

calculateAngles();

end;

*//Procedura kura rekina katru lenki.*

procedure TMain\_menu.calculateAngles();

var i :integer;

AB,AC,BC,AD,DC,DB :array of real;

begin

SetLength(AB,length(vertexSet));

SetLength(AC,length(vertexSet));

SetLength(BC,length(vertexSet));

SetLength(AD,length(vertexSet));

SetLength(DC,length(vertexSet));

SetLength(DB,length(vertexSet));

for i:=0 to length(vertexSet)-1 do

begin

if (vertexSet[i].is4Vert) then

begin

AB[i] := getDistanceBetween(vertexSet[i].vertex[0].x,vertexSet[i].vertex[0].y,vertexSet[i].vertex[1].x,vertexSet[i].vertex[1].y);

AC[i] := getDistanceBetween(vertexSet[i].vertex[0].x,vertexSet[i].vertex[0].y,vertexSet[i].vertex[2].x,vertexSet[i].vertex[2].y);

BC[i] := getDistanceBetween(vertexSet[i].vertex[1].x,vertexSet[i].vertex[1].y,vertexSet[i].vertex[2].x,vertexSet[i].vertex[2].y);

AD[i] := getDistanceBetween(vertexSet[i].vertex[0].x,vertexSet[i].vertex[0].y,vertexSet[i].vertex[3].x,vertexSet[i].vertex[3].y);

DC[i] := getDistanceBetween(vertexSet[i].vertex[3].x,vertexSet[i].vertex[3].y,vertexSet[i].vertex[2].x,vertexSet[i].vertex[2].y);

DB[i] := getDistanceBetween(vertexSet[i].vertex[3].x,vertexSet[i].vertex[3].y,vertexSet[i].vertex[1].x,vertexSet[i].vertex[1].y);

*// rekinam katru lenki*

vertexSet[i].vertex[3].angle := arccos((sqr(DB[i])+sqr(AD[i])-sqr(AB[i]))/(2\*DB[i]\*AD[i]))\*180/Pi;

vertexSet[i].vertex[1].angle := arccos((sqr(DB[i])+sqr(BC[i])-sqr(DC[i]))/(2\*DB[i]\*BC[i]))\*180/Pi;

vertexSet[i].vertex[0].angle := 180 - vertexSet[i].vertex[3].angle;

vertexSet[i].vertex[2].angle := 180 - vertexSet[i].vertex[1].angle;

end else

begin

AB[i] := getDistanceBetween(vertexSet[i].vertex[0].x,vertexSet[i].vertex[0].y,vertexSet[i].vertex[1].x,vertexSet[i].vertex[1].y);

AC[i] := getDistanceBetween(vertexSet[i].vertex[0].x,vertexSet[i].vertex[0].y,vertexSet[i].vertex[2].x,vertexSet[i].vertex[2].y);

BC[i] := getDistanceBetween(vertexSet[i].vertex[1].x,vertexSet[i].vertex[1].y,vertexSet[i].vertex[2].x,vertexSet[i].vertex[2].y);

if ((round(AB[i])=0) or (round(AC[i])=0) or (round(BC[i])=0)) then

begin

vertexSet[i].vertex[0].angle:=0;

vertexSet[i].vertex[1].angle:=0;

vertexSet[i].vertex[2].angle:=0;

end

else begin

vertexSet[i].vertex[0].angle := arccos((AC[i]\*AC[i]+AB[i]\*AB[i]-BC[i]\*BC[i])/(2\*AC[i]\*AB[i]))\*180/Pi;

vertexSet[i].vertex[1].angle := arccos((BC[i]\*BC[i]+AB[i]\*AB[i]-AC[i]\*AC[i])/(2\*BC[i]\*AB[i]))\*180/Pi;

vertexSet[i].vertex[2].angle := 180-vertexSet[i].vertex[0].angle-vertexSet[i].vertex[1].angle;

end;

end;

end;

classifyObjects(AC,BC,DB,AD);

end;

//ši procedura atbild par objektu definešanu

procedure TMain\_menu.classifyObjects(AC,BC,DB,AD :array of real);

var i,j :integer;

angle :array of array of integer;

objectIndex :array of integer;

begin

SetLength(angle,length(vertexSet), 4);

SetLength(objectIndex,length(vertexSet));

for i:=0 to length(vertexSet)-1 do

for j:=0 to 3 do

begin

if (j=3) then

begin

if (vertexSet[i].is4Vert) then

angle[i,j]:= round(vertexSet[i].vertex[j].angle)

else angle[i,j]:= 0;

end else

begin

angle[i,j]:= round(vertexSet[i].vertex[j].angle);

end;

end;

for i:=0 to length(vertexSet)-1 do

begin

if (vertexSet[i].is4Vert) then

begin

if ((abs(angle[i,0]-90)<3) and (abs(angle[i,1]-90)<3) and (abs(angle[i,2]-90)<3)) then

begin */ / nosacijumi pec kuriem tiek defineti objekti*

if ((abs(AC[i]-BC[i])<4) and (abs(AC[i]-DB[i])<4) and (abs(AC[i]-AD[i])<4)) then

objectIndex[i] := 5 else objectIndex[i] := 6;

end else

if (((abs(angle[i,0]-90)<3) and (abs(angle[i,3]-90)<3)) or ((abs(angle[i,1]-90)<3) and (abs(angle[i,2]-90)<3))) then objectIndex[i] := 8

else

if ((abs(AD[i]-BC[i])<3) or (abs(AC[i]-DB[i])<3)) then objectIndex[i] := 7

else objectIndex[i] := 9;

end else

begin

objectIndex[i] := 4;

if ((abs(angle[i,0]-90)<2) or (abs(angle[i,1]-90)<2) or (abs(angle[i,2]-90)<2)) then objectIndex[i] := 1 else

begin

if ((abs(angle[i,0]-angle[i,1])<6) and (abs(angle[i,0]-angle[i,2])<6)) then objectIndex[i] := 3 else

begin

if ((abs(angle[i,0]-angle[i,1])<2) or (abs(angle[i,0]-angle[i,2])<2) or (abs(angle[i,1]-angle[i,2])<2)) then objectIndex[i] := 2;

end;

end;

end;

end;

segmentToImage();

Image1.Canvas.Font.Name := 'Tahoma';

Image1.Canvas.Font.Size := 10;

Image1.Canvas.Font.Color := clRed;

Image1.Canvas.Font.Style := [fsItalic, fsBold];

Image1.Canvas.Brush.Color := clWhite;

for i:=0 to length(vertexSet)-1 do

begin

Case objectIndex[i] of *//izvada nosaukumu katram objektam*

1 : image1.Canvas.TextOut(vertexSet[i].vertex[0].x-20, vertexSet[i].vertex[0].y+40, 'Taisnlenka trijsturis');

2 : image1.Canvas.TextOut(vertexSet[i].vertex[0].x-20, vertexSet[i].vertex[0].y+40, 'Vienadsanu trijsturis');

3 : image1.Canvas.TextOut(vertexSet[i].vertex[0].x-20, vertexSet[i].vertex[0].y+40, 'Vienadmalu trijsturis');

4 : image1.Canvas.TextOut(vertexSet[i].vertex[0].x-20, vertexSet[i].vertex[0].y+40, 'Dazhadmalu trijsturis');

5 : image1.Canvas.TextOut(vertexSet[i].vertex[0].x-20, vertexSet[i].vertex[0].y+30, 'Kvadrats');

6 : image1.Canvas.TextOut(vertexSet[i].vertex[0].x-20, vertexSet[i].vertex[0].y+30, 'Taisnsturis');

7 : image1.Canvas.TextOut(vertexSet[i].vertex[0].x-20, vertexSet[i].vertex[0].y+30, 'Vienadsanu trapece');

8 : image1.Canvas.TextOut(vertexSet[i].vertex[0].x-20, vertexSet[i].vertex[0].y+30, 'Taisnlenka trapece');

9 : image1.Canvas.TextOut(vertexSet[i].vertex[0].x-20, vertexSet[i].vertex[0].y+30, 'Dazhadsanu trapece');

end;

end;

end;

*// procedura veido objektus scene*

procedure TMain\_menu.Button7Click(Sender: TObject);

Var pol:array[1..3] of TPoint;

polK:array[1..4] of TPoint;

i,j, ran:integer;

begin

if (image1.Height=bmp.Height) then button6.Click();

bmp.Width:=777;

bmp.Height:=425;

bmp.PixelFormat:=pf24bit;

for j:=0 to bmp.height-1 do

for i:=0 to bmp.width-1 do

begin

bmp.Canvas.Pixels[i,j]:=clwhite;

end;

bmp.Canvas.pen.Color:=clgreen;

bmp.Canvas.brush.Color:=clgreen;

ran:=150+Random(40);

pol[1].X:=50; pol[1].Y:=50;

pol[2].X:=ran; pol[2].Y:=50;

pol[3].X:=round((ran+50)/2);

pol[3].Y:=round((pol[1].Y + pol[2].Y + (pol[2].X-pol[1].X)\*sqrt(3))/2);

bmp.Canvas.Polygon(pol);

bmp.Canvas.pen.Color:=clpurple;

bmp.Canvas.brush.Color:=clpurple;

pol[1].X:=400; pol[1].Y:=100+Random(100);

pol[2].X:=300; pol[2].Y:=120;

pol[3].X:=Round(Cos(radtodeg(60))\*(pol[2].X-pol[1].X)-(pol[2].y-pol[1].y)\*Sin(radtodeg(60))+pol[1].x);

pol[3].Y:=Round(Cos(radtodeg(60))\*(pol[2].y-pol[1].y)+(pol[2].x-pol[1].x)\*Sin(radtodeg(60))+pol[1].y);

bmp.Canvas.Polygon(pol);

bmp.Canvas.pen.Color:=clblue;

bmp.Canvas.brush.Color:=clblue;

pol[1].X:=150; pol[1].Y:=250+Random(50);

pol[2].X:=50; pol[2].Y:=350;

pol[3].X:=200; pol[3].Y:=400;

bmp.Canvas.Polygon(pol);

bmp.Canvas.pen.Color:=clgreen;

bmp.Canvas.brush.Color:=clgreen ;

pol[1].X:=300; pol[1].Y:=250+Random(50);

pol[2].X:=300; pol[2].Y:=400;

pol[3].X:=400; pol[3].Y:=400;

bmp.Canvas.Polygon(pol);

//kvadrats

bmp.Canvas.pen.Color:=clred;

bmp.Canvas.brush.Color:=clred;

ran := 50+Random(40);

polk[1].X:=190; polk[1].Y:=150;

polk[2].X:=polk[1].X+ran; polk[2].Y:=polk[1].Y;

polk[3].X:=polk[1].X; polk[3].Y:=polk[1].Y+ran;

polk[4].X:=polk[2].X; polk[4].Y:=polk[3].Y;

bmp.Canvas.Rectangle(polk[1].x, polk[1].y, polk[4].x, polk[4].y);

//taisnsturis

bmp.Canvas.pen.Color:=clyellow;

bmp.Canvas.brush.Color:=clyellow;

ran := 50+Random(50);

polk[1].X:=480; polk[1].Y:=30;

polk[2].X:=polk[1].X+ran; polk[2].Y:=polk[1].Y;

polk[3].X:=polk[1].X; polk[3].Y:=polk[1].Y+ran+60;

polk[4].X:=polk[2].X; polk[4].Y:=polk[3].Y;

bmp.Canvas.Rectangle(polk[1].x, polk[1].y, polk[4].x, polk[4].y);

// taisnlenka trapece

bmp.Canvas.pen.Color:=clMaroon;

bmp.Canvas.brush.Color:=clMaroon;

ran := 100+Random(20);

polk[1].X:=450 + Random(20); polk[1].Y:=270;

polk[2].X:=520; polk[2].Y:=polk[1].Y;

polk[3].X:=630; polk[3].Y:=polk[1].Y+ran;

polk[4].X:=polk[1].X; polk[4].Y:=polk[1].Y+ran;

bmp.Canvas.Polygon(polk);

// vienadsanu trapece

bmp.Canvas.pen.Color:=clactivecaption;

bmp.Canvas.brush.Color:=clactivecaption;

ran := 100+Random(20);

polk[1].X:=660; polk[1].Y:=30;

polk[2].X:=polk[1].X + 30+Random(30); polk[2].Y:=polk[1].Y;

polk[3].X:=polk[2].X + 30; polk[3].Y:= 100;

polk[4].X:=polk[1].X - 30; polk[4].Y:= 100;

bmp.Canvas.Polygon(polk);

// dazadsanu trapece

bmp.Canvas.pen.Color:=clAqua ;

bmp.Canvas.brush.Color:=clAqua;

ran := 50+Random(20);

polk[1].X:=660; polk[1].Y:=200;

polk[2].X:=polk[1].X + ran; polk[2].Y:=polk[1].Y;

polk[3].X:=polk[2].X + 30; polk[3].Y:= 270;

polk[4].X:=polk[1].X - 60; polk[4].Y:= 270;

bmp.Canvas.Polygon(polk);

ReadFromBMP();

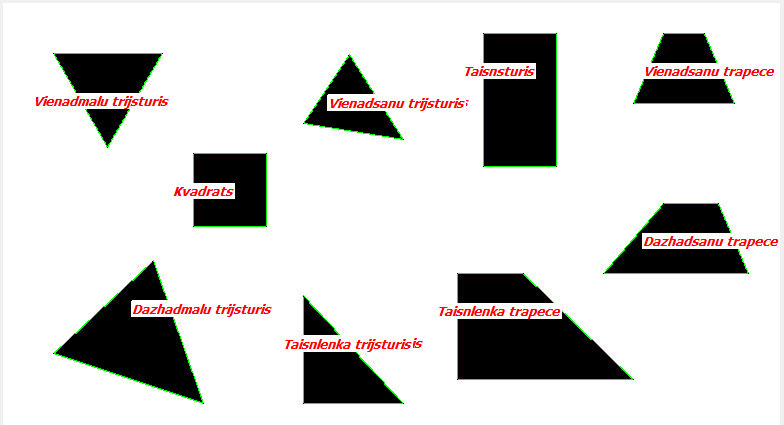
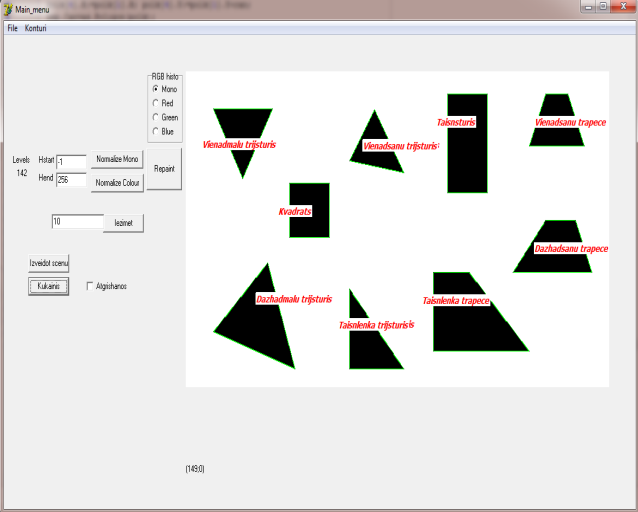
imgToImage1(0);

image1.Height:=bmp.Height;

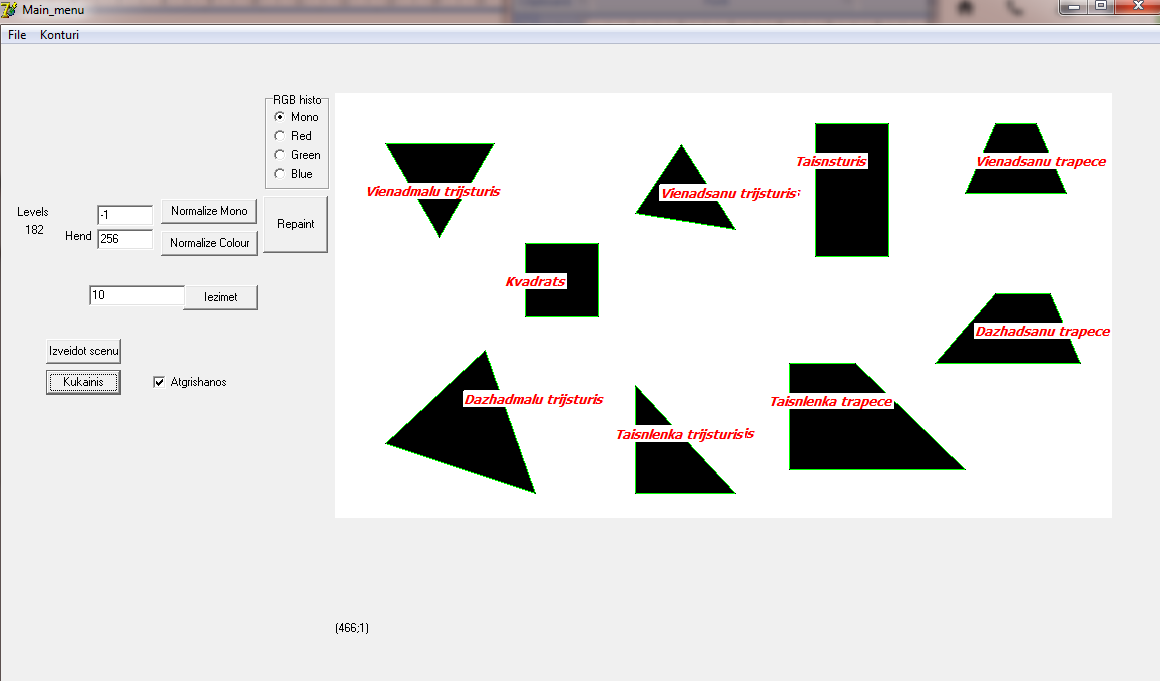
image1.Width:=bmp.Width;

end;

**Programmas darbības rezultāti:**



1.attels Visi atšķirīgi objekti.



**Secinājums:**

Tika izstrādāts 8 laboratorijas darbs, kura laika tika realizēts objektu noteikšana, kura laika tika nedaudz pilnveidots kukaiņa algoritms un tagad var scēnu ģenerēt nevis izvelieties katru reizi jaunu attēlu, kas atvieglo programmas testēšanu. Laboratorijas darbs tika izstrādāts sekmīgi.