BITMAP'ın döndərilməsi

Sadiq Məlikov

BITMAP (.bmp) şəkillərinin döndərilmə üsuluna baxmazdan əvvəl BITMAP faylının quruluşunu nəzərdən keçirək.

Basic BMP File Format

Name	Size	Description
Header	14 bytes	Windows Structure: BITMAPFILEHEADER
Signature	2 bytes	'BM'
FileSize	4 bytes	File size in bytes
reserved	4 bytes	unused (=0)
DataOffset	4 bytes	File offset to Raster Data
InfoHeader	40 bytes	Windows Structure: BITMAPINFOHEADER
Size	4 bytes	Size of InfoHeader =40
Width	4 bytes	Bitmap Width
Height	4 bytes	Bitmap Height
Planes	2 bytes	Number of Planes (=1)
BitCount	2 bytes	Bits per Pixel 1 = monochrome palette. NumColors = 1 4 = 4bit palletized. NumColors = 16 8 = 8bit palletized. NumColors = 256 16 = 16bit RGB. NumColors = 65536 (?) 24 = 24bit RGB. NumColors = 16M
Compression	4 bytes	Type of Compression 0 = BI_RGB no compression 1 = BI_RLE8 8bit RLE encoding 2 = BI_RLE4 4bit RLE encoding
ImageSize	4 bytes	(compressed) Size of Image It is valid to set this =0 if Compression = 0
XpixelsPerM	4 bytes	horizontal resolution: Pixels/meter
YpixelsPerM	4 bytes	vertical resolution: Pixels/meter
ColorsUsed	4 bytes	Number of actually used colors
ColorsImportant	4 bytes	Number of important colors 0 = all
ColorTable	4 * NumColors bytes	present only if Info.BitsPerPixel <= 8 colors should be ordered by importance
Red	1 byte	Red intensity
Green	1 byte	Green intensity
Blue	1 byte	Blue intensity
reserved	1 byte	unused (=0)
repeated NumColors times		
Raster Data	Info.ImageSize bytes	The pixel data

“BMP.h” adlı C/C++ dili başlıq faylı yaradaq.Bu başlıq faylında BITMAP ilə işləyə bilmək üçün strukturlar və funksiyalar yaradacağıq.Cədvəldən göründüyü kimi ilk 14 baytlıq hissəni Header (BMP faylının başlığı) təşkil edir. Bu 14 baytın ilk 2 baytı “signature”,yəni faylın tipini (BMP olması informasiyasını) saxlayır.Növbəti 4 bayt faylın baytlarla ölçüsünü,digər 4 bayt “reserved” və sonrakı 4 bayt da “offset data” informasiyalarını saxlayır. Növbəti 40 baytlıq hissə İnfoHeader faylın saxladığı verilənlər barədə informasiya daşıyır.Göründüyü kimi bu 40 baytlıq hissə də müvafiq olaraq 2 və 4 baytlıq hissələrə bölünərək orada en,uzunluq,ölçü,bit sayı kimi müxtəlif informasiyaları özündə saxlayır.Bunu kod şəklinə salaq:

```
1   #pragma once
2   #include <bits/stdc++.h>
3
4   #pragma pack(push, 1)
5   struct BMPFileHeader {
6       uint16_t file_type{ 0x4D42 };
7       uint32_t file_size{ 0 };
8       uint16_t reserved1{ 0 };
9       uint16_t reserved2{ 0 };
10      uint32_t offset_data{ 0 };
11  };
12
13  struct BMPInfoHeader {
14      uint32_t size{ 0 };
15      int32_t width{ 0 };
16      int32_t height{ 0 };
17
18
19      uint16_t planes{ 1 };
20      uint16_t bit_count{ 0 };
21      uint32_t compression{ 0 };
22      uint32_t size_image{ 0 };
23      int32_t x_pixels_per_meter{ 0 };
24      int32_t y_pixels_per_meter{ 0 };
25      uint32_t colors_used{ 0 };
26      uint32_t colors_important{ 0 };
27  };
28
```

Burada “uint16”-lar 16 bitlik (2 baytlıq) verilənləri, “uint32”-lər isə 32 bitlik (4 baytlıq) verilənləri saxlayır. 0x4D42 kodu isə faylın BITMAP olmasını göstərən xüsusi koddur.

Beləliklə BITMAP faylının ilk 54 baytlıq hissəsi ölçü,bit sayı və digər bu tip informasiyaları saxlamaq üçündür. İndi isə bu 54 baytdan sonrakı C×W×H baytlıq hissəsinə baxaq.Burada W şəklin üfüqi ölçüsü,H şaquli ölçüsü,C isə bir pikselə düşən rəng kanalının sayıdır.Məsələn RGB sistemi 3 kanaldan ibarətdir (1 bayt-qırmızı,1 bayt-yaşıl, 1-bayt mavi,ümumi 3 bayt və ya 24 bit).RGBA sistemi 4 kanaldan ibarətdir (1 bayt-qırmızı,1 bayt-yaşıl,1 bayt mavi,1 bayt-şəffaflıq,ümumi 4 bayt və ya 32 bit). Bu hissədə piksellərin rəng verilənləri saxlanır.

```
29  struct BMPColorHeader {
30      uint32_t red_mask{ 0x00ff0000 };
31      uint32_t green_mask{ 0x0000ff00 };
32      uint32_t blue_mask{ 0x000000ff };
33      uint32_t alpha_mask{ 0xff000000 };
34      uint32_t color_space_type{ 0x73524742 };
35      uint32_t unused[16]{ 0 };
36  };
37  #pragma pack(pop)
```

Və bunları ümumiləşdirərək yazıla və oxuna bilən fayl təşkil edən BMP strukturunu yaradırıq.Ümumi başlıq faylını bu linkdən əldə etmək olar:

https://github.com/TiberTesla/BITMAP/blob/main/BMP.h

Bu başlıq faylında elan etdiyimiz hazır struktur və funksiyalardan şəkilləri oxumaq və yazmaq üçün yararlanacağıq.İndi isə əsas məsələyə-şəklin döndərilmə üsuluna baxaq.

“main.cpp” adlı C++ kod faylı yaradaq və işə başlayaq. BMP başlıq faylımızı proqrama daxil edək:

```
1   #include <bits/stdc++.h>
2   #include "BMP.h"
3   using namespace std;
4
```

Mövcud olan “flower.bmp” şəklini BMP1 olaraq açaq və w1 və h1 dəyişənlərinə bu şəklin üfüqi və şaquli ölçüsünü (width,height) mənimsədək.

```
86  //Open bitmap (Call it BMP1)
87  BMP bmp1("flower.bmp");
88
89  //Width and height of BMP1
90  int w1=bmp1.bmp_info_header.width,h1=bmp1.bmp_info_header.height;
```

Dərəcəni radiana çevirən və radianı dərəcəyə çevirən funksiyalar təyin edək. C++ dilindəki triqonometrik funksiyalar radianla işlədiyi üçün,dərəcədən radiana keçmək bizə lazım olacaq.

```
5   //From degree to radian
6   double degtorad(double deg)
7   {
8       return (deg*M_PI/180);
9   }
10
11  //From radian to degree
12  double radtodeg(double rad)
13  {
14      return (rad*180/M_PI);
15  }
```

Daha sonra isə x və y dəyişənlərinə sahib “Point” adlı struktur elan edək. Bu Point strukturu x və y koordinatları parametrlərinə sahib olan nöqtə olacaq.

```
34  //Point (x,y)
35  struct Point
36  {
37      double x,y;
38      Point()
39      {
40
41      }
42      Point(double _x,double _y)
43      {
44          x=_x;
45          y=_y;
46      }
47  };
```

Şəklin mərkəz nöqtəsinin koordinatlarını hesablayaq və şəkli döndərmək istədiyimiz α dərəcəsinin giriş kodunu yazaq.Şəklin sol yuxarı küncünü (0;0) nöqtəsi kimi qəbul etsək,mərkəzi nöqtənin koordinatları uyğun olaraq üfüqi və şaquli ölçülərin yarısına bərabər olacaq.α-nı dərəcə ilə daxil etdikdən sonra kod onu radiana çevirəcək.

```
92  //Center point of BMP1
93  Point center((double)w1/2,(double)h1/2);
94
95  //Rotation degree-alpha,we'll convert to and use as radian
96  double alpha;
97  cin>>alpha;
98  alpha=degtorad(alpha);
```

Nəzərə alsaq ki,şəklin dönməsi üçün onun hər bir pikseli mərkəzi piksel ətrafında fırlanmalıdır,bu zaman bir nöqtənin mərkəz nöqtə ətrafında fırlanması üçün funksiya yazaraq bütün piksellərə növbə ilə tətbiq edərik.

Fərz edək ki, bizim A(x,y) nöqtəmiz var və bu nöqtəni $C(x_m,y_m)$ nöqtəsi ətrafında α dərəcə fırlatmaq istəyirik. Əvvəlcə A(x,y) nöqtəsini $A_T(x_T,y_T)$ nöqtəsinə köçürək.

$x_T=x-x_m$

$y_T=y-y_m$

$A_T(x_T,y_T)$ nöqtəsini (0;0) ətrafında α dərəcə fırlatmaq lazımdır və bunun üçün $\begin{pmatrix} cos\alpha & -sin\alpha \\ sin\alpha & cos\alpha \end{pmatrix}$ matrisini $\begin{pmatrix} x_T \\ y_T \end{pmatrix}$ matrisinə vurmaq lazımdır.Vurmadan sonra $x_T`$ və $y_T`$ alarıq ki,bunlar $A_T`(x_T`,y_T`)$ nöqtəsinin koordinatları olar.

$$\begin{pmatrix} cos\alpha & -sin\alpha \\ sin\alpha & cos\alpha \end{pmatrix}\begin{pmatrix} x_T \\ y_T \end{pmatrix}=\begin{pmatrix} x_T` \\ y_T` \end{pmatrix}$$

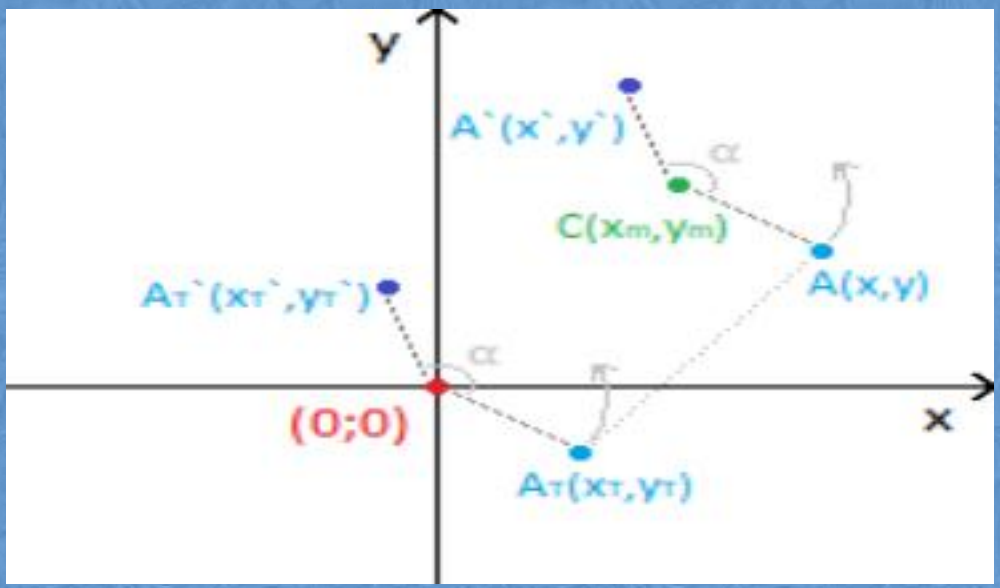
$x_T`=x_T cos\alpha - y_T sin\alpha$

$y_T`=x_T sin\alpha + y_T cos\alpha$

$A_T`$ nöqtəsi $A_T$ nöqtəsinin koordinat başlanğıcı ətrafında α dərəcə fırlanmış vəziyyətidir.$A_T`$ nöqtəsini $C(x_m,y_m)$ nöqtəsi ətrafına köçürsək və yeni nöqtəni $A`(x`,y`)$ ilə işarə etsək,$A`$ nöqtəsi A nöqtəsinin C ətrafında α dərəcə fırlanmış vəziyyəti olacaq.

$x`=x_T`+x_m$

$y`=y_T`+y_m$

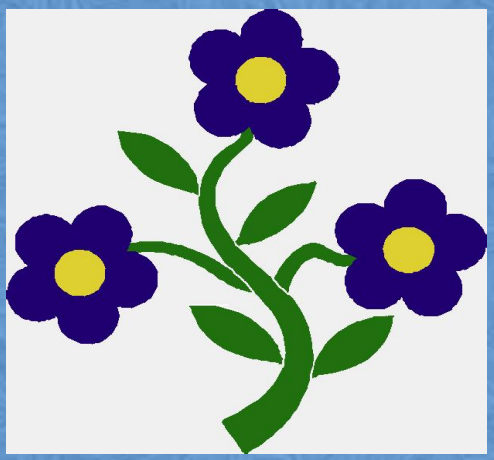


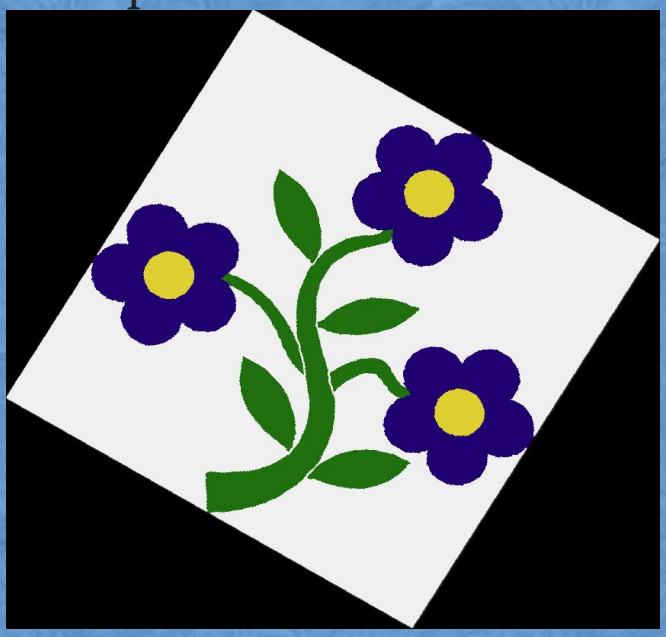
Məsələnin riyaziyyatını araşdırdığımıza görə onu koda çevirə bilərik:

```
49    //Rotate point around another point
50    Point rotaround(Point cen,double alpha,Point p)
51    {
52        double s=sin(alpha);
53        double c=cos(alpha);
54        p.x-=cen.x;
55        p.x-=cen.y;
56        double xnew=p.x*c-p.y*s;
57        double ynew=p.x*s+p.y*c;
58        p.x=xnew+cen.x;
59        p.y=ynew+cen.y;
60        return p;
61    }
62
```

Kod parçasında qaytarılan nöqtə funksiyaya verilən p nöqtəsinin dönmüş şəkli olacaq.

Nəzərə alsaq ki, şəkli döndərdikdən sonra kənarda boşluqlar (üçbucaqlar) qalacaq:





Yeni şəklin üfüqi və şaquli ölçüləri fərqli olacaq.Bu ölçüləri təyin etməyin üsuluna baxaq.Şəklin ölçüləri w2

və h2 olarsa,bu zaman künc nöqtələrin koordinatı A(0;0), B(w2;0),C(0;h2) və D(w2;h2) olar.Şəkli döndərdikdən sonra bu künc nöqtələrin koordinatları $A(x_1;y_1)$,$B(x_2;y_2)$, $C(x_3;y_3)$ və $D(x_4;y_4)$ olsun.Məlumdur ki,yeni şəklin üfüqi ölçüsü künc nöqtələrin x-ləri fərqinin maksimumuna bərabər olar.x-lər fərqi maksimum olması üçün azalan ən böyük,çıxılan ən kiçik olmalıdır.Buna görə də:

w2=max(x1,x2,x3,x4)-min(x1,x2,x3,x4)

Eyni qaydanı h2-ni tapmaq üçün y-lərə tətbiq etsək:

h2=max(y1,y2,y3,y4)-min(y1,y2,y3,y4)

Qeyd: w2 və h2 dəyişənlərinin kəsr qiymətlər də ala biləcək,lakin şəklin ölçüləri (x və y boyu piksellərin sayı) tam olmalıdır.Buna görə w2 və h2 kəsr qiymət alırsa,onları növbəti tama qədər yuvarlaqlaşdıracağıq. (Məsələn: 2.61≈3).Ədəd tamdırsa,ona toxunmuruq.

```
100     //Size of rotated image
101     Point _bef[4],_aft[4];
102     _bef[0]=Point(0,0);
103     _bef[1]=Point(w1,0);
104     _bef[2]=Point(0,h1);
105     _bef[3]=Point(w1,h1);
106     for (int i=0;i<4;i++) _aft[i]=rotaround(center,alpha,_bef[i]);
107
108     int w2=0,h2=0;
109     double xmin=INT_MAX,ymin=INT_MAX;
110     for (int i=0;i<4;i++)
111     {
112         double dw,dh;
113         for (int j=0;j<4;j++)
114         {
115             dw=abs(_aft[i].x-_aft[j].x);
116             dh=abs(_aft[i].y-_aft[j].y);
117             if (dw>(int)dw) dw++;
118             if (dh>(int)dh) dh++;
119             w2=max((double)w2,dw);
120             h2=max((double)h2,dh);
121         }
122     }
```

_bef massivi künc nöqtələrin əvvəlki,_aft isə dönmədən sonrakı koordinatlarını saxlayır.

Daha sonra isə BMP2 BITMAP faylı yaradacağıq.BMP1-dəki piksel verilənlərini və kanal sayını vector və 32 bit unsigned int tipindən ibarət pair-ə köçürəcəyik. BMP1-in piksel verilənləri ilə işləyə bilmək üçün onu tərs çevirəcəyik.(Çünki şəklin piksel verilənləri massivə sol-aşağı küncdən başlayaraq sağa və yuxarı doğru köçürülür) Tərs çevirmə əməliyyatını vrev adlı funksiya elan edib onu istifadə etməklə yerinə yetirəcəyik.24 bitlik (3 baytlıq) kanallı (RGB olan) BMP2 faylı və bu səbəbdən ölçüsü 3×w2×h2 olan piksel massivi yaradacağıq. Dönmədən sonra alınan kənardakı boşluqları qara rənglə rəngləyəcəyik və şəkilin özündəki qara rənglə qarışmaması üçün pikselin boş olub olmaması informasiyasını daşıyan w2×h2 ölçülü bool massivi elan edəcəyik.

```
134     //Create bitmap (Call it BMP2) with size w2 width and h2 height
135     BMP bmp2(w2,h2,false);
136
137     //Get data from BMP1
138     pair<vector<uint8_t>,uint32_t> p=bmp1.getregion(0,0,w1,h1);
139
140     //We'll reverse the pixel data of BMP1 for work on it
141     vector<uint8_t> rv1=vrev(p.first,p.second,w1,h1),rv2;
142
143     //Declare array for store pixel data of BMP2
144     rv2.resize(3*w2*h2);
145
146     //Information about emptiness of pixel data array of BMP2
147     vector<bool> rv_empty(w2*h2);
148
149     //Set emptiness information of all pixels true
150     for (int i=0;i<h2;i++)
151     {
152         for (int j=0;j<w2;j++)
153         {
154             rv2[3*(i*w2+j)]=0;
155             rv2[3*(i*w2+j)+1]=0;
156             rv2[3*(i*w2+j)+2]=0;
157             rv_empty[i*w2+j]=true;
158         }
159     }
```

vrev funksiyasının elan:

```
17  //Reverse pixel data of bitmap
18  vector<uint8_t> vrev(vector<uint8_t> v,uint32_t c,uint32_t w,uint32_t h)
19  {
20      vector<uint8_t> rv;
21      rv.resize(c*w*h);
22      for (int i=0;i<h;i++)
23      {
24          for (int j=0;j<w;j++)
25          {
26              rv[c*(i*w+j)+2]=v[c*((h-(i+1))*w+j)];
27              rv[c*(i*w+j)+1]=v[c*((h-(i+1))*w+j)+1];
28              rv[c*(i*w+j)]=v[c*((h-(i+1))*w+j)+2];
29          }
30      }
31      return rv;
32  }
```

Burada v piksel massivi,c kanal sayı,w və h üfüqi və şaquli ölçülərdir.

İndi isə hər bir pikseli mərkəz ətrafında döndərərək BMP2-nin piksel massivinə yazaq. Şəklin sol yuxarı küncünü (0;0) kimi qəbul etsək, müəyyən sayda piksel döndərildikdən sonra mənfi koordinatlar ala bilər.Lakin piksel massivi mənfi indeks ala bilməz.Ona görə də hər bir nöqtəni (x_{min},y_{min}) qədər sürüşdürsək,bu problem aradan qalxar.x_{min} və y_{min} günc nöqtələrin x-lərinin və y-lərinin minimumudur.Bu zaman hər bir sürüşdürülmüş nöqtənin koordinatı 0-a bərabər və ya 0-dan böyük olacaq.x_{min} və y_{min} qiymətlərini hesablayaq:

```
124     //Get minimum x and minimum y
125     for (int i=0;i<4;i++)
126     {
127         for (int j=0;j<4;j++)
128         {
129             xmin=min(xmin,_aft[i].x);
130             ymin=min(ymin,_aft[i].y);
131         }
132     }
```

Və pikselləri fırladaraq yeni massivə yazmaq üçün kodu yazaq.

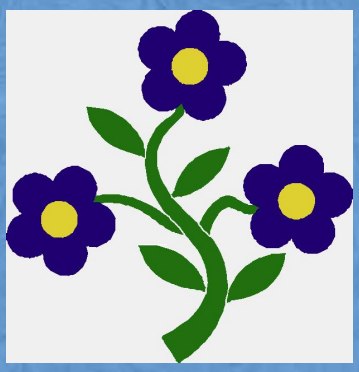
```
161     //Copy each pixel of BMP2 to BMP1 as rotated around center
162     for (int i=0;i<h1;i++)
163     {
164         for (int j=0;j<w1;j++)
165         {
166             Point pt=rotaround(center,alpha,Point(j,i));
167             pt.x-=xmin;
168             pt.y-=ymin;
169             int pos=(int)pt.y*w2+(int)pt.x;
170             if (pos<w2*h2)
171             {
172                 rv2[3*pos]=rv1[p.second*(i*w1+j)];
173                 rv2[3*pos+1]=rv1[p.second*(i*w1+j)+1];
174                 rv2[3*pos+2]=rv1[p.second*(i*w1+j)+2];
175                 rv_empty[pos]=false;
176             }
177         }
178     }
```

Piksel massivini BMP2-yə köçürək və nəticəsinə baxaq. Köçürməzdən əvvəl piksel massivini tərs çevirək.

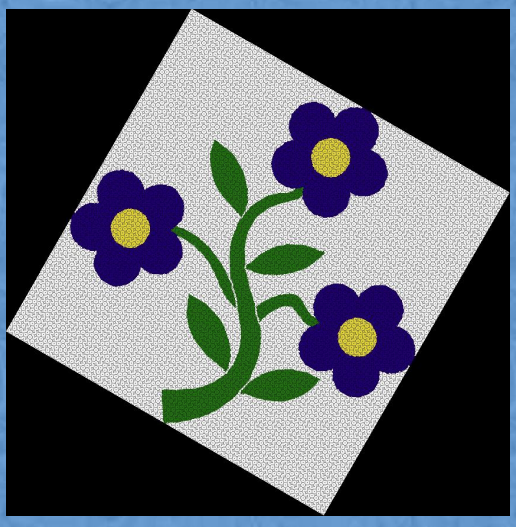
```
212     //Reverse pixel data for write to BMP2
213     rv2=vrev(rv2,3,w2,h2);
214
215     //Write pixels to BMP2
216     for (int i=0;i<h2;i++)
217         for (int j=0;j<w2;j++)
218             bmp2.set_pixel(j,i,rv2[3*(i*w2+j)],rv2[3*(i*w2+j)+1],rv2[3*(i*w2+j)+2],0);
219
220     //Save BMP2
221     bmp2.write("image.bmp");
```

Girişə dərəcə qiyməti verək və nəticəyə baxaq:

Əvvəlki:



Sonrakı:



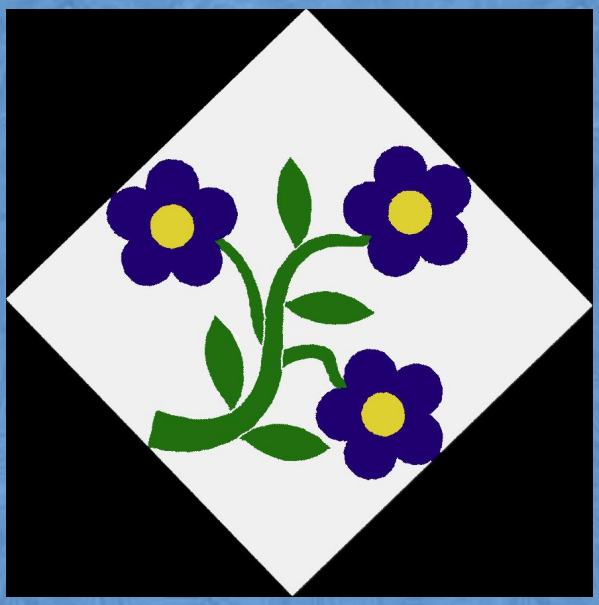
Dönmüş şəklə yaxından baxsaq qara nöqtələri asanlıqla görə bilərik.Bunun səbəbi dönmədən sonra piksellərin tam koordinat alaraq köçürülməsidir.Yəni bəzi xanalar boş qalır,onlara rəng köçürülmür.Həmin boş qalmış piksellərə qonşuluqda yerləşən 4 pikselin (yuxarı,aşağı, sağ,sol)RGB qiymətlərinin ədədi ortasını mənimsətməklə bu problemi aradan qaldırmaq olar.

```
180     //Filling the gaps in the image caused by the completion of the fraction
181     for (int i=1;i<h2-1;i++)
182     {
183         for (int j=1;j<w2-1;j++)
184         {
185             if (rv2[3*(i*w2+j)]==0 && rv2[3*(i*w2+j)+1]==0 && rv2[3*(i*w2+j)+2]==0 && rv_empty[i*w2+j])
186             {
187                 //Four neighbour principle
188                 Color c[4];
189                 c[0]=Color(rv2[3*((i-1)*w2+j)],rv2[3*((i-1)*w2+j)+1],rv2[3*((i-1)*w2+j)+2]);
190                 c[1]=Color(rv2[3*(i*w2+j+1)],rv2[3*(i*w2+j+1)+1],rv2[3*(i*w2+j+1)+2]);
191                 c[2]=Color(rv2[3*((i+1)*w2+j)],rv2[3*((i+1)*w2+j)+1],rv2[3*((i+1)*w2+j)+2]);
192                 c[3]=Color(rv2[3*(i*w2+j-1)],rv2[3*(i*w2+j-1)+1],rv2[3*(i*w2+j-1)+2]);
193
194                 //Set RGB of empty pixel equal to numerical average of neighbours
195                 int r=0,g=0,b=0;
196                 for (int k=0;k<4;k++)
197                 {
198                     r+=c[k].r;
199                     g+=c[k].g;
200                     b+=c[k].b;
201                 }
202                 r/=4;
203                 g/=4;
204                 b/=4;
205                 rv2[3*(i*w2+j)]=r;
206                 rv2[3*(i*w2+j)+1]=g;
207                 rv2[3*(i*w2+j)+2]=b;
208             }
209         }
210     }
```

Burda istifadə edilmiş Color strukturunun elanı:

```
63      //Color as RGB/RGBA
64      struct Color
65      {
66          uint8_t r,g,b,a;
67          Color()
68          {
69              r=0;
70              g=0;
71              b=0;
72              a=false;
73          }
74          Color(uint8_t _r,uint8_t _g,uint8_t _b,bool _a=false)
75          {
76              r=_r;
77              g=_g;
78              b=_b;
79              a=_a;
80          }
81      };
```

Nəticə:



Beləliklə boş qalmış pikselləri doldurma problemini də aradan qaldırdıq.Şəkli istənilən α dərəcə döndərmək üçün C++ proqramını yazdıq.
https://github.com/TiberTesla/BITMAP/blob/main/main.cpp