SOUTH UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA

IMAGE AND VIDEO PROCESSING

CLASS PROJECT 1

Image Resizing

Author:
Jie YANG

Supervisor:
JIANHONG SHI

September 17, 2018

CONTENTS

Image resizing

Introduction

Image resizing is a kind of method to change image size, which can be divided into reduce image size and enlarge image size. It is very useful in image processing. Image reduce usually remove several lines in original image to decrease image size. While image enlargement require to generate new lines, which cause interpolation. Interpolation is a kind method which can generate new pixels base on neighborhood pixels. In this project, bilinear and bicubic interpolation will be discussed.

Method

For each pixel in resized image $f(x,y)$, we can map it to original image by **Eq.2**. Then we can use different interpolation method to determine its value.

$$\begin{cases} \hat{x} = & x/scale \\ \hat{y} = & y/scale \end{cases} \tag{1}$$

Alternative line reduction & pixel replication

For alternative line reduction and pixel replication, they has same calculation process. The difference is that former used to reduce image size while later used to enlarge image size.

$$\begin{cases} \tilde{x} = & ceil(\hat{x}) \\ \tilde{y} = & ceil(\hat{x}) \end{cases} \tag{2}$$

For each $\hat{x}$, we choose the nearest integer as the mapped coordinates by **Eq.1**. Then transfer this pixel intensity to the resized image pixel.**Eq.3**

$$f(x,y) = F(\tilde{x},\tilde{y}) \tag{3}$$

Nearest enlargement

For each $\hat{x}$, we choose the nearest integer as the mapped coordinates by **Eq.4**. Then transfer this pixel intensity to the resized image pixel.**Eq.3**

$$\tilde{x} = \begin{cases} ceil(\hat{x}), & ceil(\hat{x}) - \hat{x} < 0.5 \\ floor(\hat{x}), & ceil(\hat{x}) - \hat{x} \geq 0.5 \end{cases} \quad \tilde{y} = \begin{cases} ceil(\hat{y}), & ceil(\hat{y}) - \hat{y} < 0.5 \\ floor(\hat{y}), & ceil(\hat{y}) - \hat{y} \geq 0.5 \end{cases} \tag{4}$$

Bilinear interpolation

Bilinear interpolation dose not just map one pixel to new image, but find two near pixels in original image **Eq.5** then give weight to each pixel and calculate their weighted sum as the intensity of new pixel.**Eq.6**

$$\begin{cases} \hat{x}_l = & floor(\hat{x}) & \hat{y}_l = & ceil(\hat{y}) \\ \hat{x}_r = & floor(\hat{x}) & \hat{y}_r = & ceil(\hat{x}) \\ p = & \hat{x} - \hat{x}_l & q = & \hat{y} - \hat{y}_l \end{cases} \tag{5}$$

$$\begin{aligned} f(x,y) = \quad & (1-q) * (p * F(\tilde{x}_r, \tilde{y}_l) + (1-p) * F(\tilde{x}_l, \tilde{y}_l)) + \\ & q * (p * F(\tilde{x}_l, \tilde{y}_r) + (1-p) * F(\tilde{x}_r, \tilde{y}_r)) \end{aligned} \tag{6}$$

Bicubic interpolation

This method use cubic polynomial $w(x)$ to approach optimal interpolation function $sin(x)/x$ **Eq.7**.

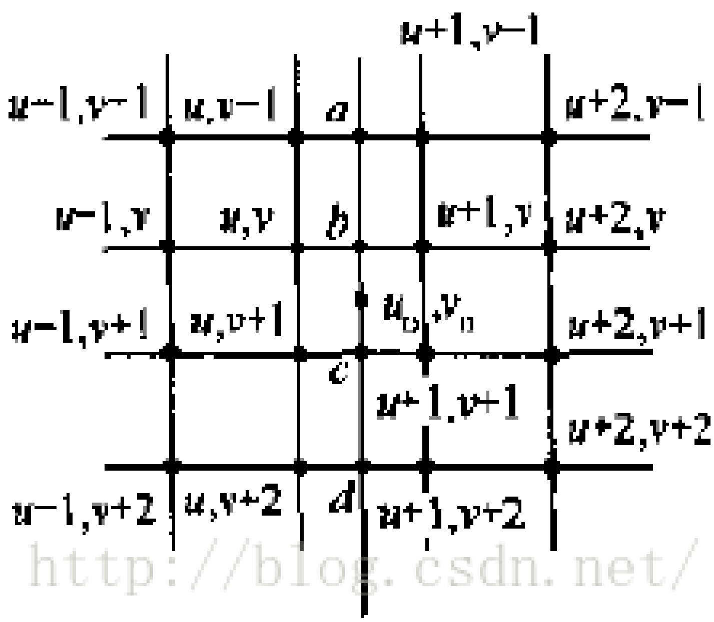


Figure 1: Diagram of bicubic interpolation

$$w(x) = \begin{cases} 1 - 2|x|^2 + |x|^3 & 0 \leq & |x| < 1 \\ 4 - 8|x| + 5|x|^2 - |x|^3 & 1 \leq & |x| < 2 \\ 0 & 2 \leq & |x| \end{cases} \tag{7}$$

First step is map $f(x,y)$ to $F(x,y)$.**Eq.8**. Then calculate map distance for next step.**Eq.9**.

$$\begin{cases} \tilde{x} = & floor(\hat{x}) \\ \tilde{y} = & floor(\hat{x}) \end{cases} \tag{8}$$

$$\begin{cases} \alpha = & \hat{x} - \tilde{x} \\ \beta = & \hat{y} - \tilde{y} \end{cases} \tag{9}$$

Bicubic interpolation do polynomial in $4x4$ neighbourhood.**Fig.1**. It first do cubic interpolation at four horizontal lines to get $g(a), g(b), g(c), g(d)$.**Eq.10** Then do vertical cubic interpolation between $g(a), g(b), g(c), g(d)$.**Eq.11**

$$\begin{aligned} g(u,v) &= w(1+\alpha)F(u-1,v) + w(\alpha)F(u,v) \\ &+ w(1-\alpha)F(u+1,v) + w(2-\alpha)F(u+2,v) \end{aligned} \tag{10}$$

$$\begin{aligned} f(x,y) &= w(1+\beta)g(u,v-1) + w(\beta)g(u,v) \\ &+ w(1-\beta)g(u,v+1) + w(2-\beta)g(u,v+2) \end{aligned} \tag{11}$$

The whole process can be write in matrix form.**Eq.15**

$$A = \begin{bmatrix} w(1+\alpha) & w(\alpha) & w(\alpha) & w(2-\alpha) \end{bmatrix} \tag{12}$$

$$B = \begin{bmatrix} F(u-1,v-1) & F(u-1,v) & F(u-1,v+1) & F(u-1,v+2) \\ F(u,v-1) & F(u,v) & F(u,v+1) & F(u,v+2) \\ F(u+1,v-1) & F(u+1,v) & F(u+1,v+1) & F(u+1,v+2) \\ F(u+2,v-1) & F(u+2,v) & F(u+2,v+1) & F(u+2,v+2) \end{bmatrix} \tag{13}$$

$$C = \begin{bmatrix} w(1+\beta) & w(\beta) & w(\beta) & w(2-\beta) \end{bmatrix}^T \tag{14}$$

$$f(x,y) = A \times B \times C \tag{15}$$

Results

These are the original images.**Fig.2**

(a) Crosses (b) Circles

Figure 2: Original images

(a) Crosses (b) Circles

Figure 3: Reduce to half size by alternative line reduction

Image reduction

- **Alternative line reduction**

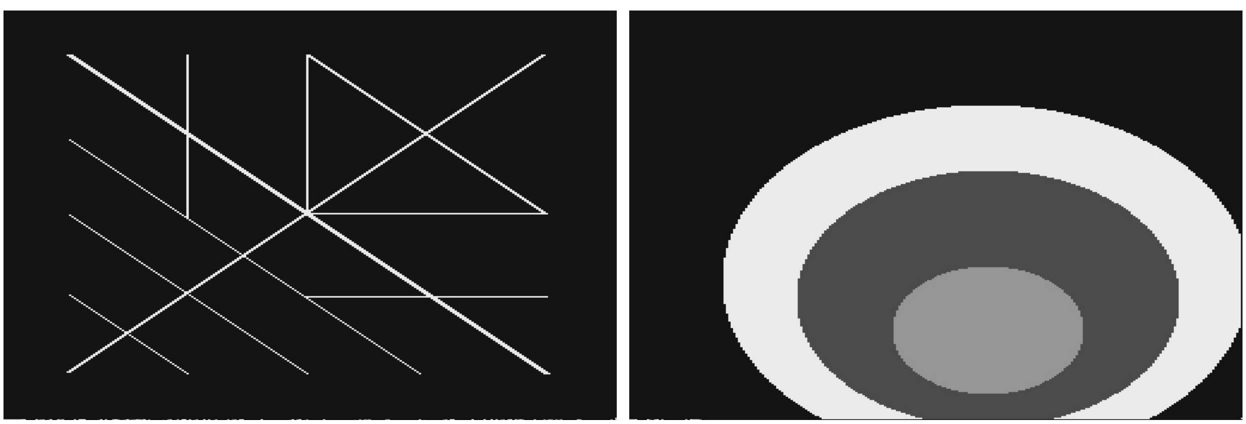
It is clear to see that the image size reduced to half of original image. **Fig.3** However, some horizontal and vertical lines disappeared after reducing. The line which near to diagonal line also disappeared. **Fig.3-a**

- **Fractional linear reduction to reduce images to any smaller size**

It is clear to see that the image size reduced to 0.7 times of original image.**Fig.4** Although there is no line reduced, the image be blurry. The white line turn to gray.**Fig.4-a** For circles image, the edge becomes indistinct and more artifact appear near edge.**Fig.4-b**

(a) Crosses (b) Circles

Figure 4: Reduce to 0.7×0.7 by bilinear interpolation



(a) Crosses (b) Circles

Figure 5: Enlarge to 2×3 by pixel replication

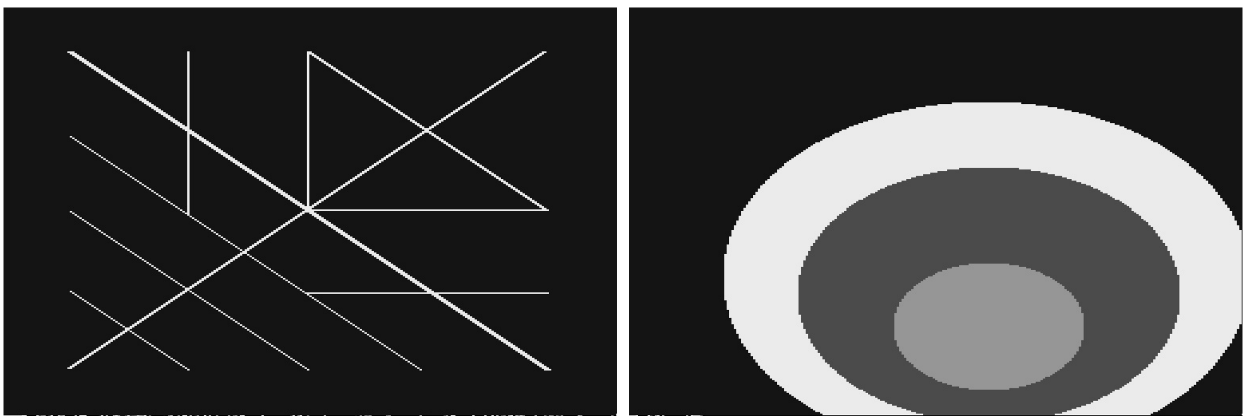
Image enlargement

- **Pixel replication**

It is clear to see that the width and height enlarged to 3 and 2 times of original image, respectively.**Fig.5** The image still be clear and no gray pixel generate.**Fig.5-a** For circles image, the edge still be very sharp but can see clear steps near edge.**Fig.5-b**

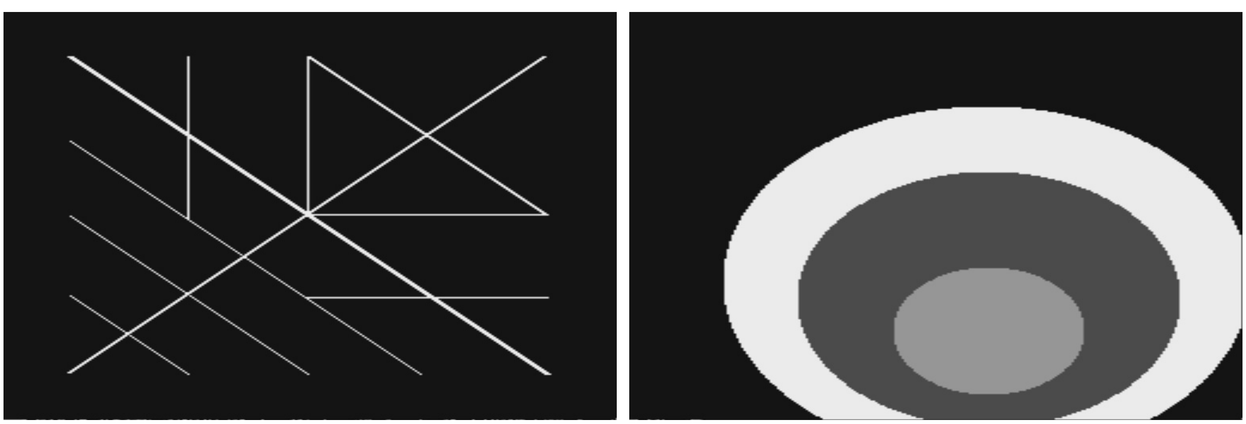
- **Nearest enlargement**

It is clear to see that the width and height enlarged to 3 and 2 times of original image, respectively.**Fig.6** There is no gray pixel generate but the line be discontinuous. There are several black line inserted into white vertical lines and the edge of bias also be etched by back line.**Fig.6-a** For circles image, the edge also be etched by near pixels, which induce blurry edge.**Fig.6-b**



(a) Crosses (b) Circles

Figure 6: Enlarge to 2×3 by nearest enlargement



(a) Crosses (b) Circles

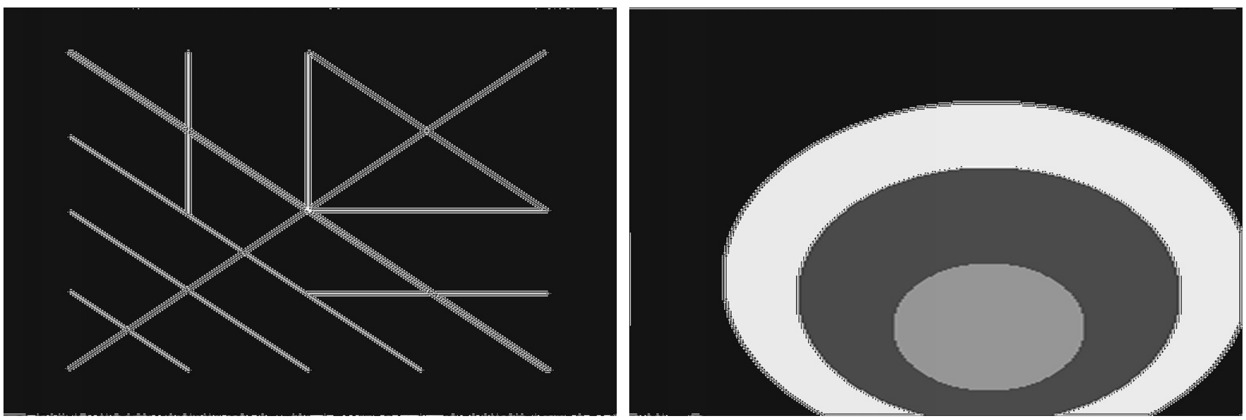
Figure 7: Enlarge to 2×3 by bilinear interpolation

- **Bilinear interpolation**

It is clear to see that the width and height enlarged to 3 and 2 times of original image, respectively.**Fig.7** There are gray pixels generated near edge, which induce to smooth edge.**Fig.7-a** For circles image, the edge also be smooth.**Fig.7-b**

- **Bicubic interpolation**

It is clear to see that the width and height enlarged to 3 and 2 times of original image,respectively. **Fig.8** The edge is shaper than bilinear interpolation, but also insert some back pixels near to edge, which let the lines seems like cracked**Fig.8-a** For circles image, the edge also be smooth but there are some misplaced pixels near to edge.**Fig.8-b**

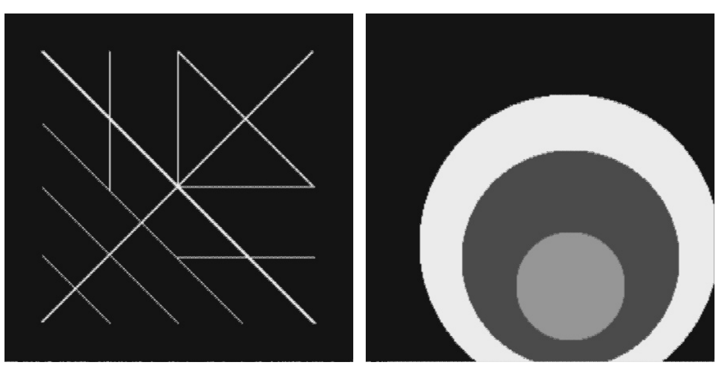


(a) Crosses (b) Circles

Figure 8: Enlarge to 2×3 by Bicubic interpolation

- **Fractal linear expansion to expand images to any larger size**

It is clear to see that the size becomes 1.7 times of original image.**Fig.9** There are gray pixels generated near edge, which induce to smooth edge but also blur the image.**Fig.9-a** For circles image, the edge also be smooth.**Fig.9-b**



(a) Crosses (b) Circles

Figure 9: Enlarge to 1.7×1.7 by bilinear interpolation

Discussion

In this experiment, we can find that when use alternative line reduction to reduce image size, it is easy to calculate but it may remove some horizontal or vertical lines, especially when these lines are very narrow. This may cause information drop.

When use pixel replication to enlarge image size, it is also very easy to calculate. But it will cause steps at edge when edge is not horizontal or vertical.

When use nearest enlargement to enlarge image size, it is easy to calculate and will reduce the step, but may cause intensity step, especially when pattern is very narrow.

When use bilinear interpolation to enlarge image size, the edge will be smooth , but it will blur the image, reduce the resolution.

When use bilinear interpolation to enlarge image size, the edge will be shaper than bilinear interpolation, but the calculation is more complex and takes longer times to do interpolation. When there is intensity step at edge, it may induce pixel misplace.

Reference

https://blog.csdn.net/zhangfuliang123/article/details/76659467

Supplementary

This is the code used in this project.

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <string>
#include<math.h>
using namespace cv;
using namespace std;


struct ImgProp{
  string img_fold;
  //string save_fold;
  string type;
  string img_name;
  string mark;
  Mat img;



};

//enum functype{ ALR, FLR, Pixen, NEE, Bilinear, Bicubic, FLE };
enum interptype{ PXR,NEE, Bilinear, Bicubic };
```

```
double PXRenlargement(int x, int y, double Xscale, double Yscale, Mat img){
  //imshow("cedhihfji" ,img);
  //cout << "Nearest enlargement" << endl;
  double val;
  double xhat = x / Xscale;
  double yhat = y / Yscale;

  int xproj = ceil(xhat);
  int yproj = ceil(yhat);
  val = (double)(*(img.data + img.step[0] * xproj + img.step[1] * yproj));
  //cout << x << "," << y << "-->" << xhat << "," << yhat << "\t\t" << val << endl...
    ;
  return val;

}
double Nearestenlargement(int x, int y, double Xscale, double Yscale,Mat img){
  //imshow("cedhihfji" ,img);
  //cout << "Nearest enlargement" << endl;
  double val;
  double xhat = x / Xscale;
  double yhat = y / Yscale;

  int xproj = (ceil(xhat)-xhat)<0.5 ? floor(xhat) :ceil(xhat);
  int yproj = (ceil(yhat) - yhat)<0.5 ? floor(yhat) : ceil(yhat);
  val = (double)(*(img.data + img.step[0] * xproj + img.step[1] * yproj));
  //cout << x << "," << y << "-->" << xhat << "," << yhat << "\t\t" << val << endl...
    ;
  return val;

}
double Bilinearinterpolation(int x, int y, double Xscale, double Yscale, Mat img){
  //cout << "Bilinear interpolation" << endl;
  double val;
  double xhat = x / Xscale;
  double yhat = y / Yscale;
  int x_left = floor(xhat);
  int x_right = ceil(xhat);
  int y_left = floor(yhat);
  int y_right = ceil(yhat);
  double ll = (double)(*(img.data + img.step[0] * x_left + img.step[1] * y_left));
  double lr = (double)(*(img.data + img.step[0] * x_left + img.step[1] * y_right))...
    ;
  double rl = (double)(*(img.data + img.step[0] * x_right + img.step[1] * y_left))...
    ;
  double rr = (double)(*(img.data + img.step[0] * x_right + img.step[1] * y_right)...
    );
  double p = xhat - x_left;
  double q = yhat - y_left;

  val = (1 - q)*(p*rl + (1 - p)*ll) + q*(p*lr + (1 - p)*rr);
  return val;
```

```

}


double wcubic(double x,double a){
  double w = 0;
  if (x < 0) x = -x;
  if (x < 1) w = (a + 2)*pow(x, 3) - (a + 3)*pow(x, 2) + 1;
  else if (x<2)  w = a *pow(x, 3) - 5*a*pow(x, 2) + 8*a*x+4;
  return w;
}
Mat get_S_mat(double v){
  double a = -1;
  Mat S(4, 1, CV_64FC1);
  S.at<double>(0, 0) = wcubic(1+v, a);
  S.at<double>(1, 0) = wcubic( v,  a);
  S.at<double>(2, 0) = wcubic(1-v, a);
  S.at<double>(3, 0) = wcubic(2-v, a);
  return S;
}


double Bilcubicinterpolation(int x, int y, double Xscale, double Yscale, Mat img){
  //cout << "Bicubic interpolation" << endl;
  double val;

  double xhat = x / Xscale;
  double yhat = y / Yscale;
  int i = floor(xhat);
  int j = floor(yhat);
  double u = xhat - i;
  double v = yhat - j;

  Mat B(4, 4, CV_64FC1);
  for (int k = 0; k < 4;++k)
  for (int m = 0; m < 4; ++m)
    B.at<double>(k, m) = (double)(*(img.data + img.step[0] * (i + k - 1) + img....
    step[1] * (j + m - 1)));
    //cout << k << m << endl;

  Mat A = get_S_mat(u);
  Mat C= get_S_mat(v);
  Mat ABC= A.t()*B*C;


  val = ABC.at<double>(0, 0);


  /*
  cout << "test\n" << test<< endl;

```

```
  cout << "yiyiyi" << (double)(*(B.data + B.step[0] * 0 + B.step[1] * 0)) << endl;
  cout << "onoeone" << (double)(*(B.data + B.step[1] * 0 + B.step[1] * 1)) << endl...
    ;
  cout << "jdfksfjdlk" << (double)(*(B.data + B.step[1] * 2 + B.step[1] * 3)) << ...
    endl;
  cout << "A\n" << A << "\nB\n" << B << "\nC\n" << C << "\nABC\n" << ABC << "\nval...
    \t" << val << endl;
  */
  return val;
}






Mat img_resize(Mat img, double Xscale, double Yscale, interptype intype){
  int xsize = img.rows;
  int ysize = img.cols;
  int xlen = floor(xsize*Xscale);
  int ylen = floor(ysize*Yscale);
  Mat ima(xlen, ylen, CV_8UC1);

  for (int x = 0; x <xlen; ++x){
    uchar * p = ima.ptr<uchar>(x);
    for (int y = 0; y <ylen; ++y){

      switch (intype){
      case PXR:p[y] = PXRenlargement(x, y, Xscale, Yscale, img); break;
      case NEE:p[y] = Nearestenlargement(x, y, Xscale, Yscale, img); break;
      case Bilinear:p[y] = Bilinearinterpolation(x, y, Xscale, Yscale, img); break...
    ;
      case Bicubic:p[y] = Bilcubicinterpolation(x, y, Xscale, Yscale, img); break;
      default: cout << "Interpoletation type error !" << endl; break;
      }

    }
    //cout << x << ":\t" << endl;
  }
  return ima;

}
void img_save(ImgProp imgprop){
  string save_path = imgprop.img_fold + imgprop.img_name + "_" + imgprop.mark + ...
    imgprop.type;
  imwrite(save_path, imgprop.img);
  imshow(imgprop.mark + "      " + imgprop.img_name, imgprop.img);
}


int main()
```

```
{
  ImgProp imgprop = {
    "D://graduated//Image_process//lab//PGM_images//",
    ".pgm" };
  ImgProp imgsave = { "D://graduated//Image_process//lab//lab_report//lab2//...
    imagesave//",
    ".jpg" };
  for (int i = 1; i < 3; i++)
  {
    if (i == 1)imgprop.img_name = "crosses";
    else imgprop.img_name = "circles";
    imgsave.img_name = imgprop.img_name;
    string img_path = imgprop.img_fold + imgprop.img_name + imgprop.type;
    Mat img = imread(img_path);

    imshow("original    " + imgprop.img_name, img);
    imgsave.img = img;
    imgsave.mark = "original";
    img_save(imgsave);


    // enlage int times
    imgsave.img = img_resize(img, 2, 3, PXR);
    imgsave.mark = "PXR_2x3";
    img_save(imgsave);
    imgsave.img = img_resize(img, 2, 3, NEE);
    imgsave.mark = "NEE_2x3";
    img_save(imgsave);

    imgsave.img = img_resize(img, 2, 3, Bilinear);
    imgsave.mark = "Bilinear_2x3";
    img_save(imgsave);

    imgsave.img = img_resize(img,2, 3, Bicubic);
    imgsave.mark = "Bicubic_2x3";
    img_save(imgsave);
    // enlage fravtal  times
    imgsave.img = img_resize(img, 1.7, 1.7, PXR);
    imgsave.mark = "PXR_1_7x1_7";
    img_save(imgsave);
    imgsave.img = img_resize(img, 1.7, 1.7, NEE);
    imgsave.mark = "NEE_1_7x1_7";
    img_save(imgsave);

    imgsave.img = img_resize(img, 1.7, 1.7, Bilinear);
    imgsave.mark = "Bilinear_1_7x1_7";
    img_save(imgsave);

    imgsave.img = img_resize(img, 1.7, 1.7, Bicubic);
    imgsave.mark = "Bicubic_1_7x1_7";
    img_save(imgsave);
```

```
        // reduce fractal times
        imgsave.img = img_resize(img, 0.7, 0.7, PXR);
        imgsave.mark = "PXR_0_7x0_7";
        img_save(imgsave);
        imgsave.img = img_resize(img, 0.7, 0.7, NEE);
        imgsave.mark = "NEE_0_7x0_7";
        img_save(imgsave);

        imgsave.img = img_resize(img, 0.7, 0.7, Bilinear);
        imgsave.mark = "Bilinear_0_7x0_7";
        img_save(imgsave);

        imgsave.img = img_resize(img, 0.7, 0.7, Bicubic);
        imgsave.mark = "Bicubic_0_7x0_7";
        img_save(imgsave);
        // reduce int times
        imgsave.img = img_resize(img, 0.5, 0.5, PXR);
        imgsave.mark = "PXR_0_5x0_5";
        img_save(imgsave);
        imgsave.img = img_resize(img, 0.5, 0.5, NEE);
        imgsave.mark = "NEE_0_5x0_5";
        img_save(imgsave);

        imgsave.img = img_resize(img, 0.5, 0.5, Bilinear);
        imgsave.mark = "Bilinear_0_5x0_5";
        img_save(imgsave);

        imgsave.img = img_resize(img, 0.5, 0.5, Bicubic);
        imgsave.mark = "Bicubic_0_5x0_5";
        img_save(imgsave);



    }


    waitKey(0);
    return 0;
}
```

C++ code for image processing