Iran University Of Science And Technology
Faculty of Computer Engineering

Seam Carving Project

Prof.Entezari

Keyvan Booshehri

How did i calculate pixels energies?

A simple Computing model is used, the borders energy values are set to 1.000, because they're uncomputable via the function.

$$energy(x,y) = \sqrt{\Delta_x^2(x,y) + \Delta^2 y(x,y)}$$
$$\Delta_x^2(x,y) = R_x(x,y)^2 + G_x(x,y)^2 + B_x(x,y)^2$$
$$R_x(x,y) = R(x+1,y) - R(x-1,y)$$
$$R(x,y) : Red\ value\ at\ pixel\ x,y.$$

```
public static double[][] CalculateEnergyArray(Color[][] colors)
{
    double[][] Array = new double[colors.Length][];
    //interating inside pixels:
    for (int i = 0; i < colors.Length; i++)
    {
        Array[i] = new double[colors[i].Length];
        for (int j = 0; j < Array[i].Length; j++)
        {
            double energy;
            //implementing given energy function:
            if (0 < i && i < colors.Length - 1 && 0 < j && j < colors[i].Length - 1)
            {
                double deltaXRed = colors[i + 1][j].R - colors[i - 1][j].R;
                double deltaXGreen = colors[i + 1][j].G - colors[i - 1][j].G;
                double deltaXBlue = colors[i + 1][j].B - colors[i - 1][j].B;
                double deltaX = Math.Pow(deltaXRed, 2) + Math.Pow(deltaXGreen, 2) + Math.Pow(deltaXBlue, 2);

                double deltaYRed = colors[i][j + 1].R - colors[i][j - 1].R;
                double deltaYGreen = colors[i][j + 1].G - colors[i][j - 1].G;
                double deltaYBlue = colors[i][j + 1].B - colors[i][j - 1].B;
                double deltaY = Math.Pow(deltaYRed, 2) + Math.Pow(deltaYGreen, 2) + Math.Pow(deltaYBlue, 2);

                energy = Math.Round(Math.Pow(deltaX + deltaY, 0.5), 3);
            }
            //setting the borders energy to 1.000(becasue theyre not calculatable using the function):
            else
                energy = 1_000;

            Array[i][j] = energy;
        }
    }
    return Array;
}
```

How does the algorithm for finding min seam work?

The program is based on Dynamic programming.We calculate the minimum energy possible considering we start from top pixels and we should reach the bottom pixels.So the last row will contain amount of energy we traverse for the possible seams.So we should find the minimum of them , that will be the last pixel of the desired seam.To track the pixels of the route , we backtrack each pixels predecessor while we construct the array N , and store them in array M.

$$N(i,j)=E(i,j)+\min(N(i-1,j-1),N(i-1,j),N(i-1,j+1))$$

Important Note:For the edge pixels there will not be any j-1 or j+1 pixels so we only compare the two available pixels and the first row must be set to their own energy (1.000).

For instance, N[5,7] holds the minimum amount of energy traversed to reach pixel [5,7] starting from the top and M[5,7] holds coordinates of the pixel which is before pixel [5,7] in the minimal route.
So we can recursively determine the path and make its pixels red.And then delete them(we do this in two different copies of pixels array to have a remaining record of red pixels).

```
for (int i = 1; i < colors.Length; i++)
{
    for (int j = 0; j < colors[i].Length; j++)
    {
        if (j == 0)
        {
            if (N[i - 1, j + 1] < N[i - 1, j])
            {
                x = i - 1;
                y = j + 1;
            }
            else
            {
                x = i - 1;
                y = j;
            }


        }
        else if (j == colors[i].Length - 1)
        {
            if (N[i - 1, j] < N[i - 1, j - 1])
            {
                x = i - 1;
                y = j;
            }
            else
            {
                x = i - 1;
                y = j - 1;
            }

        }
        else
        {
            x = i - 1;
            y = j - 1;
            if (N[i - 1, j - 1] < N[i - 1, j] && N[i - 1, j - 1] < N[i - 1, j + 1])
            {
                x = i - 1;
                y = j - 1;
            }

            if (N[i - 1, j] < N[i - 1, j - 1] && N[i - 1, j] < N[i - 1, j + 1])
            {
                x = i - 1;
                y = j;
            }

            if (N[i - 1, j + 1] < N[i - 1, j] && N[i - 1, j + 1] < N[i - 1, j - 1])
            {
                x = i - 1;
                y = j + 1;
            }
        }
        M[i, j] = new Point(x, y);

        N[i, j] = EnergyArray[i][j] + N[x, y];
    }

}
```

```
List<(double, int)> Dic = new List<(double, int)>();
for (int z = 0; z < colors[0].Length; z++)
{
    Dic.Add((N[colors.Length - 1, z], z));
}
Dic.Sort((x, y) => y.Item1.CompareTo(x.Item1));
Console.WriteLine(Dic[Dic.Count - 1]);
int index = 0;

index = Dic[(Dic.Count - 1)].Item2;
int myx = colors.Length - 1;
int myy = index;
Point myp = new Point();
myp.X = myx;
myp.Y = myy;
colors[myx][myy] = Color.Red;
colors[M[myx, myy].X][M[myx, myy].Y] = Color.Red;
colors2[myx][myy] = Color.Red;
colors2[M[myx, myy].X][M[myx, myy].Y] = Color.Red;


for (int z = 0; z < colors.Length - 2; z++)
{
    myx = M[myx, myy].X;
    myy = M[myx, myy].Y;
    colors[M[myx, myy].X][M[myx, myy].Y] = Color.Red;
    colors2[M[myx, myy].X][M[myx, myy].Y] = Color.Red;
}


SavePhoto(colors2, redFilePath);
List<Color[]> list2 = new List<Color[]>();
for (int i = 0; i < colors.Length; i++)
{
    List<Color> list1 = new List<Color>();
    for (int j = 0; j < colors[i].Length; j++)
    {
        Point mytp = new Point();
        mytp.X = i;
        mytp.Y = j;
        if (colors[i][j] != Color.Red)
            list1.Add(colors[i][j]);
    }
    list2.Add(list1.ToArray());
}
colors = list2.ToArray();
```

How did i delete Pixels?

To delete pixels we make two linked lists.We add every row to the list 1 and add them as arrays to list2 and then make list2 an array.So we'll have a 2D array of arrays containing new pixels.During this process we simply ignore red pixels.

What about other seams?

We loop with length of the seams count and do the process with the pixel arrays produced at the previous state of the loop.

Important Note:If we do all of the process on the same array , minimum routes may overlap, indexes will differ and the picture ratio will change, the time complexity would be better but the result wont be satisfying.

What about horizontal seams?

To find and remove horizontal seams we simply rotate 90 deg the picture, remove the vertical seams and rotate 270 deg to rotate it back :) .

```
var image = Utilities.LoadImage(inputFilePath);

var colors = Utilities.ConvertImageToColorArray(image);
var colors2 = Utilities.ConvertImageToColorArray(image);

colors = Utilities.FandR_V_MinSeam_DP(colors, colors2, verticalSeamsCount, redFilePath);
Utilities.SavePhoto(colors, outputFilePath);


var theimage = Utilities.LoadImage(redFilePath);
colors2 = Utilities.ConvertImageToColorArray(theimage);

var bit = Utilities.ConvertToBitmap(colors);
bit.RotateFlip(RotateFlipType.Rotate90FlipNone);
bit.Save(outputFilePath);

var bit2 = Utilities.ConvertToBitmap(colors2);
bit2.RotateFlip(RotateFlipType.Rotate90FlipNone);
bit2.Save(redFilePath);

image = Utilities.LoadImage(outputFilePath);
var image2 = Utilities.LoadImage(redFilePath);

var mycolors = Utilities.ConvertImageToColorArray(image);
var mycolors2 = Utilities.ConvertImageToColorArray(image2);

mycolors = Utilities.FandR_V_MinSeam_DP(mycolors, mycolors2, horizontalSeamsCount, redFilePath);
Utilities.SavePhoto(colors, outputFilePath);

theimage = Utilities.LoadImage(redFilePath);
colors2 = Utilities.ConvertImageToColorArray(theimage);

bit2 = Utilities.ConvertToBitmap(colors2);
bit2.RotateFlip(RotateFlipType.Rotate270FlipNone);
bit2.Save(redFilePath);

bit = Utilities.ConvertToBitmap(mycolors);
bit.RotateFlip(RotateFlipType.Rotate270FlipNone);
bit.Save(outputFilePath);
```

How does the program run?

5 inputs are taken from the user in the terminal.
in path
out path
red path
vertical seams count
horizontal seams count

```
0 references
public static void Main(string[] args)
{
    Console.WriteLine("Enter inputFilePath:");
    string my_in1 = Console.ReadLine();
    string inputFilePath = @my_in1;

    Console.WriteLine("Enter outputFilePath:");
    string my_in2 = Console.ReadLine();
    string outputFilePath = @my_in2;


    Console.WriteLine("Enter redoutputFilePath:");
    string my_in3 = Console.ReadLine();
    string redFilePath = @my_in3;

    Console.WriteLine("Enter vertical seams count:");
    long verticalSeamsCount = Convert.ToInt32(Console.ReadLine());

    Console.WriteLine("Enter horizental seams count:");
    long horizontalSeamsCount = Convert.ToInt32(Console.ReadLine());
```

How images are represented and read by computer?

Using c# drawing bitmap ,images are converted to color(pixel) arrays.

```
//A method to load an image from a path:
5 references
public static Image LoadImage(string path)
{
    Image img = Image.FromFile(path);
    return img;
}

//A method to save an array of colors(pixels) as a bitmap and then as an image to a path:
3 references
public static void SavePhoto(Color[][] colors, string path)
{
    var bmp = ConvertToBitmap(colors);
    bmp.Save(path);
}

//A method to convert an array of colors(pixels) to bitmap object:
5 references
public static Bitmap ConvertToBitmap(Color[][] colors)
{
    Bitmap bmp = new Bitmap(colors[0].Length, colors.Length, System.Drawing.Imaging.PixelFormat.Format32bppArgb);
    for (int y = 0; y < colors.Length; y++)
    {
        for (int x = 0; x < colors[y].Length; x++)
            bmp.SetPixel(x, y, colors[y][x]);
    }
    return bmp;
}

//A method to convert image object to array of colors(pixels) using bitmap:
6 references
public static Color[][] ConvertImageToColorArray(Image image)
{
    Color[][] colors = new Color[image.Height][];
    using (Bitmap bmp = new Bitmap(image))
    {
        for (int y = 0; y < image.Height; y++)
        {
            colors[y] = new Color[image.Width];
            for (int x = 0; x < colors[y].Length; x++)
                colors[y][x] = bmp.GetPixel(x, y);
        }
    }
    return colors;
}
```

Time and storage complexity?

Im more of a python and java programmer than a c# programmer, but i chose c# for this project because it is much faster.

At first i used to hold seams in linked lists and checked to remove them using list.contains() function which made the code very slow because of the search order.

But i improved the codes performance when i was writing the red seams abstraction part.Instead-of checking and holding the list every time, i set the desired pixels colors to read and used it as the condition of deletion which made the code much faster.

Now the code runs smoothly and for 100x100 seams works under a minute for a medium size picture of test cases.

As array N and M must be computed every-time the complexity becomes:

O(Vsc*Hsc*H*W)~=O(n*m)

We could make it so much faster with greedy approach but with the **given energy** function this is the fastest approach which produces **satisfying** images.

A demonstration?

All of the pictures have been tested with 100*100 seams count and available in the test result folder included in the project.

```
(base) keyvans-MacBook-Air:pro
/j4/tnx2hb1s1lg8f67pmx7z9x7800
Enter inputFilePath:
sample.png
Enter outputFilePath:
sampleout.png
Enter redoutputFilePath:
samplered.png
Enter vertical seams count:
20
Enter horizental seams count:
20
```