

Seam Carving

Step 0: 介绍

原论文：Avidan and Shamir, “Seam Carving for Content-aware Image Resizing,” SIGGRAPH 2007.

这篇文章讲了一种通过自动寻找到的缝隙，用最佳的操作方式对图像的尺寸进行缩放的方法，它跟别的图像尺寸变换方法不一样的地方是，它会通过对图像内容的感知来保持图像的重要内容信息不受损失，我用下图来做一个例子：

下面是一张需要被处理的原始图像（猜猜这个建筑是？）：



下面是用普通的缩放方法对图像进行处理的结果，你可以看到由于图像的高被压缩，因此图像中的关键内容都发生了形变：



下面这张则是Seam Carving后的结果，你可以看到图像的重要内容的形变要远小于传统的方法：



你的任务是根据相关材料，按照步骤实现该算法。在实验开始前，请确保你的编程环境是NOI Linux 2.0。

下发材料包括：

- **Seam Carving.pdf** 也就是这篇文章。
- **CImg库参考手册中文版.pdf** 注意该手册是比较老的版本，不一定准确，你需要对照VSCode内的提示（当你在VSCode里写了一个CImg库的函数，把鼠标移到函数上等一等，会有VSCode生成的提示）得到准确写法。
- **imret.pdf** 原论文。
- **1.bmp** 待处理的图片。
- **CImg.h** 图像处理库。
- **1.cpp, 2.cpp, 3.cpp** 示例代码文件。

Step 1: 图片预处理 10pts

本实验使用CImg库对图片进行处理。CImg库只包含了一个名为 **CImg.h** 的头文件，你可以在下发文件中找到。

我们将使用CImg库加载一张图片，你后续的seam carving也会在该图片上进行。

```
// 1.cpp
#include "CImg.h"
using namespace cimg_library;
int main()
{
    CImg<unsigned char> image("1.bmp");
    // 输出图像的宽、高、三维图像的深度（由于这是二维图像所以深度为1）、色彩通道数
    （RGB图像通道为3）
    printf("%d %d %d
    %d\n",image.width(),image.height(),image.depth(),image.spectrum());
    // 显示原图
    image.display("Original Image");
    return 0;
}
```

然后使用命令 `g++ 1.cpp -o 1 -O2 -L/usr/X11R6/lib -lm -lpthread -lX11` 即可编译。
可以 `./1` 运行一下该程序观察运行效果。

然后，我们将这幅彩色的图片灰度化，灰度化是将一幅彩色图像转换为灰度化图像的过程。彩色图像通常包括R、G、B三个分量，分别显示出红绿蓝等各种颜色，灰度化就是使彩色图像的R、G、B三个分量相等的过程。灰度图像中每个像素仅具有一种样本颜色，其灰度是位于黑色与白色之间的多级色彩深度，灰度值大的像素点比较亮，反之比较暗，像素值最大为255（表示白色），像素值最小为0（表示黑色）。

一种常见的方法是将RGB三个分量求和再取平均值，但更为准确的方法是设置不同的权重，将RGB分量按不同的比例进行灰度划分。比如人类的眼睛感官蓝色的敏感度最低，敏感最高的是绿色，因此将RGB按照0.299、0.587、0.114比例加权平均能得到较合理的灰度图像。

在CImg中，我们可以使用 `cimg_forXY(img, x, y)` 循环遍历图片上的每个像素，这其实是一个宏，原定义是 `for (int y = 0; y<(int)((img)._height); ++y) for (int x = 0; x<(int)((img)._width); ++x)`。然后用 `img(x,y,z,c)` 访问图片在 (x, y, z, c) 处的像素（ z 表示图像的深度， c 表示图像的色彩通道， $c = 0$ 表示红色通道， $c = 1$ 表示绿色通道， $c = 2$ 表示蓝色通道，注意，下标都是从0开始的）。

请自行实现把彩色图片灰度化的相关代码。代码结构可以参考下面给出的代码，但关键代码需要自己补全。

```
// 1.cpp
#include "CImg.h"
using namespace cimg_library;
int main()
```

```

{
    CImg<unsigned char> image("1.bmp");
    // 创建灰度图（注意灰度图只有一个颜色通道）
    CImg<unsigned char> gray(image.width(),image.height(),1,1);
    // 灰度图像素清零
    gray.fill(0);
    /*
        从原图计算得到灰度图，此处代码需要自己实现
    */
    // 显示灰度图
    gray.display("Gray");
    return 0;
}

```

到这里，我们就完成了图片的预处理工作。

Step 2: 图像梯度计算 20pts

在高等数学中，梯度定义如下：对于连续的二维函数 $f(x, y)$ ，其在点 (x, y) 处的梯度是下面的二维向量：

$$\nabla f(x, y) = \begin{pmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{pmatrix}$$

其中， $\frac{\partial f(x, y)}{\partial x} = \lim_{\epsilon \rightarrow 0} \frac{f(x+\epsilon, y) - f(x, y)}{\epsilon}$ ，即 f 在 (x, y) 处对 x 求偏导；
 $\frac{\partial f(x, y)}{\partial y} = \lim_{\epsilon \rightarrow 0} \frac{f(x, y+\epsilon) - f(x, y)}{\epsilon}$ ，即 f 在 (x, y) 处对 y 求偏导。

梯度的幅值作为变化率大小的度量，其值为梯度向量的L2范数（二维向量 (a, b) 的L2范数定义为 $\sqrt{a^2 + b^2}$ ，高维向量以此类推），即 $|\nabla f(x, y)| = \sqrt{\left(\frac{\partial f(x, y)}{\partial x}\right)^2 + \left(\frac{\partial f(x, y)}{\partial y}\right)^2}$ 。

对于离散的二维离散函数 $f(x, y)$ ，可以用有限差分作为梯度幅值的一个近似，如下式所示。

$$|\nabla f(x, y)| = \sqrt{(f(x+1, y) - f(x, y))^2 + (f(x, y+1) - f(x, y))^2}$$

尽管梯度幅值和梯度两者之间有着本质的区别，但在数字图像处理中提到梯度时，往往不加区分，即将上式的梯度幅值称为梯度。

上式中包括平方和开方，不方便计算，因此可近似为绝对值的形式，实际上就是用L1范数代替了L2范数：

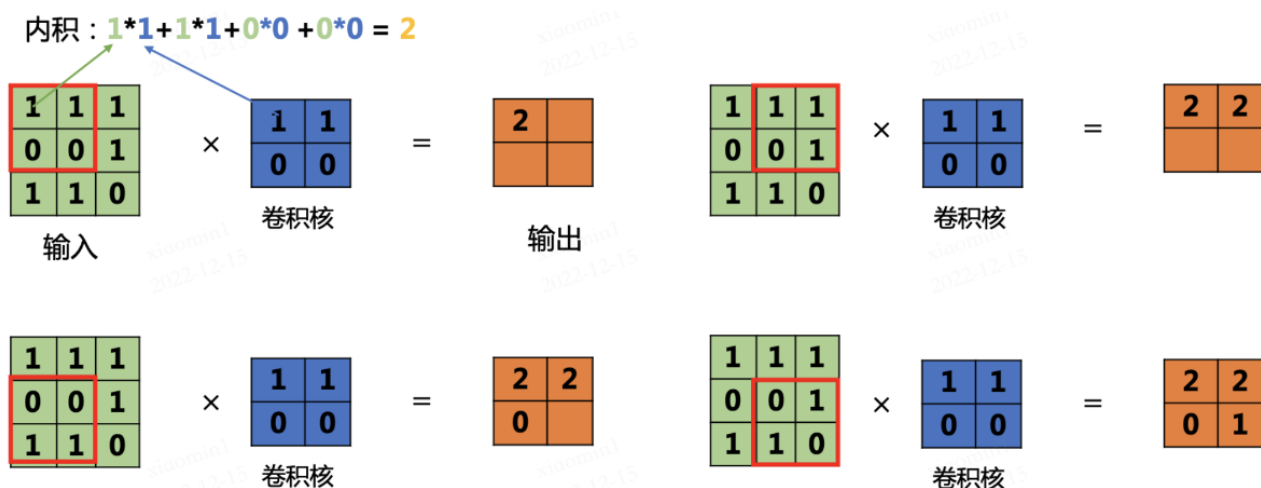
$$|\nabla f(x, y)| = |f(x + 1, y) - f(x, y)| + |f(x, y + 1) - f(x, y)|$$

而在实际使用中，经常被采用的是另外一种近似梯度——Robert交叉梯度：

$$|\nabla f(x, y)| = |f(x + 1, y + 1) - f(x, y)| + |f(x, y + 1) - f(x + 1, y)|$$

下面我们介绍图像上的二维卷积（其实是二维相关），便于用卷积的语言来阐述图像梯度的计算。

卷积是通过一定大小的卷积核作用于图像的局部区域，将局部图像区域的像素值与卷积核中的数据做内积运算，如下图：



假设输入是一个 3×3 的单通道图像（可以将图像看作一个矩阵），卷积核为 2×2 矩阵。首先，从左上角开始在输入矩阵上选择一个与卷积核大小一致（ 2×2 ）的“窗口”，然后将该“窗口”中的数值与卷积核中的数值做内积运算（将对应位置的数据相乘，之后相加）。最后，依次向右、向下滑动窗口，覆盖整个输入矩阵，获得输出矩阵。

那么，用卷积的语言来考虑Robert交叉梯度，会发现Robert交叉梯度的卷积核有两个，分别是：

$$\omega_1 = \begin{Bmatrix} -1 & 0 \\ 0 & 1 \end{Bmatrix}, \quad \omega_2 = \begin{Bmatrix} 0 & -1 \\ 1 & 0 \end{Bmatrix}$$

其中， ω_1 对接近 45° 边缘有较强响应； ω_2 对接近 -45° 边缘有较强响应。

有了前面学习的卷积知识，只要分别以 ω_1 和 ω_2 为卷积核，对原图像进行卷积就可得到输出矩阵 G_1 和 G_2 ，而根据公式，最终的Robert交叉梯度为： $G = |G_1| + |G_2|$ 。

由于计算卷积时奇数尺寸的卷积核更常用（因为奇数尺寸的卷积核有中心点），下面再介绍一种大小为 3×3 的Sobel卷积核。

$$\omega_1 = \begin{Bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{Bmatrix}, \quad \omega_2 = \begin{Bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{Bmatrix}$$

其中， ω_1 对水平边缘有较强响应； ω_2 对竖直边缘有较强响应。

同样的，分别以 ω_1 和 ω_2 为卷积核，对原图像进行卷积就可得到输出矩阵 G_1 和 G_2 ，而根据公式，最终的Sobel梯度为： $G = |G_1| + |G_2|$ 。

下面，请在Step 1计算得到的灰度图的基础上，利用Sobel卷积核计算灰度图的Sobel梯度。

注意：根据上文描述，一幅 $n \times m$ 的图片在计算Sobel梯度后分辨率会变成 $(n - 2) \times (m - 2)$ ，即最外层一圈的像素不会被循环遍历到。但是实际运算中我们也对最外层一圈的像素计算梯度，此时超出原图像边界的像素定义为与它最近的有效像素的值，这也被称为诺依曼边界条件(Neumann border condition)。

请自行实现计算灰度图的梯度的相关代码。可能你会用到 3×3 邻域遍历，请参考CImg库参考手册的2.6.4节，由于版本更新，新版的 `cimg_for3x3` 宏在使用时增加了一个参数 `type`，即 `cimg_for3x3(img, x, y, z, v, I, type)`。代码结构可以参考下面给出的代码，但关键代码需要自己补全。


```
// 2.cpp
#include "CImg.h"
using namespace cimg_library;
int main()
{
    CImg<unsigned char> image("1.bmp");
    // 创建灰度图（注意灰度图只有一个颜色通道）
    CImg<unsigned char> gray(image.width(), image.height(), 1, 1);
    // 灰度图像素清零
    gray.fill(0);
    /*
        从原图计算得到灰度图，此处代码需要自己实现
    */
    // 创建一幅和灰度图分辨率相同的梯度图
    CImg<int> grad(gray);
    /*
        从灰度图计算得到梯度图，此处代码需要自己实现
    */
    grad.display("Grad");
    return 0;
}
```

Step 3: Seam Carving 40pts

先介绍Seam Carving的算法思想。由于梯度越大，表示图像内容变化越大，如果把这些像素删掉，对图像内容影响就很大。所以为了缩小图片大小，应该删去那些梯度较小的像素。但是如果删去若干个梯度最小的像素，这些像素在图上分布是分散的，删去以后会导致图像不连续，所以删去的像素应该本身具有一定的连续性，比如，形成一条“狭缝”（seam）。

再介绍Seam Carving的算法流程。假设原图的宽为 n 像素，高为 m 像素，要把原图的宽度减少 k 像素。

首先，计算图片的灰度图。

然后循环 k 次，每次对图片的灰度图计算梯度图 G ，然后从梯度图 G 上从上至下找一条长度为 m 的路径 (x_i, y_i) ，满足 $x_i = i, |y_{i+1} - y_i| \leq 1$ ，且最小化 $\sum_{i=1}^m G(x_i, y_i)$ 。然后把这条路径对应的像素（就是刚才所说的“狭缝”）在原图和灰度图上删掉。在删去狭缝的时候，你可能会用到函数`CImg<T> img.get_crop(int x1, int y1, int x2, int y2)`，该函数从原图的 $[x_1, x_2] \times [y_1, y_2]$ 区域截取一块返回作为新的图像。

输出删去 k 条路径后的图片，此时原图的宽度就减少了 k 像素。

请自行实现Seam Carving算法，并且输出宽度减少了128像素以后的图片。代码结构可以参考下面给出的代码，但关键代码需要自己补全。

```
// 3.cpp
#include "CImg.h"
using namespace cimg_library;
int k=128;
int main()
{
    CImg<unsigned char> image("1.bmp");
    // 创建灰度图（注意灰度图只有一个颜色通道）
    CImg<unsigned char> gray(image.width(),image.height(),1,1);
    // 灰度图像素清零
    gray.fill(0);
    /*
        从原图计算得到灰度图，此处代码需要自己实现
    */

    while(k-->0) {
        // 创建梯度图（注意梯度图只有一个颜色通道）
        CImg<int> grad(gray);
        /*
            从灰度图计算得到梯度图，此处代码需要自己实现
        */
    }
}
```

```

    */
    /*
        从梯度图上从上至下找一条“狭缝”，此处代码需要自己实现
    */
    /*
        在梯度图和原图上删除这条“狭缝”，此处代码需要自己实现
    */
}
//保存处理后的图像为2.bmp
image.save("2.bmp");
image.display("Seam Carving");
return 0;
}

```

Step 4: 其他应用 30pts

自行从下面两个选题中选择一个实现（可以自己想，也可以阅读原论文作为参考）：

- 运用Seam Carving算法为原图的宽度增加 $k = 128$ 像素。
- 人为指定一块原图的 $[x_1, x_2] \times [y_1, y_2]$ 区域，这块区域被保护起来，“狭缝”不优先从该区域中经过，起到人为保护图像中某块内容的效果，如下图：

