

# 计算机图形学实验报告

## 基于 Seam Carving 算法的图像缩放与对象移除

张钰晖

2015011372, yuhui-zh15@mails.tsinghua.edu.cn, 185-3888-2881

2017 年 6 月 24 日

## 目录

<b>1 问题描述</b>	<b>1</b>
<b>2 算法原理</b>	<b>2</b>
2.1 Seam 的求解 . . . . .	2
2.2 图像缩小 . . . . .	2
2.3 图像放大 . . . . .	2
2.4 对象移除 . . . . .	2
2.5 使用不同的算子进行测试 . . . . .	2
2.6 自选例子的 Fancy 效果 . . . . .	3
<b>3 实验结果</b>	<b>3</b>
3.1 图像缩放 . . . . .	3
3.2 对象移除 . . . . .	18
3.3 算子测试 . . . . .	28
3.4 缩小比例 . . . . .	37
<b>4 实验心得</b>	<b>47</b>
<b>5 程序使用</b>	<b>47</b>
5.1 编译方式 . . . . .	47
5.2 使用方式 . . . . .	47

5.3 文件组织	48
5.4 测试环境	48
<b>6 源代码</b>	<b>48</b>

## 1 问题描述

图像缩放一直是图像领域研究的热门话题，常规的缩放会改变图像比例，导致图像失真。在缩小与放大的过程中保证图像不失真，有着重要而深远的意义。Shai Avidan 和 Ariel Shamir 提出的基于内容感知的算法 Seam Carving 在该领域具有非常显著的效果，其基本原理是使用动态规划选出图像中能量最小的一条线（称为“Seam”）并砍去。基于该算法更可以轻松地实现对象移除等操作，配合上不同的算子更能对特定部分进行筛选。为此，笔者实现了该算法，并对算法进行了一定的讲解与分析。

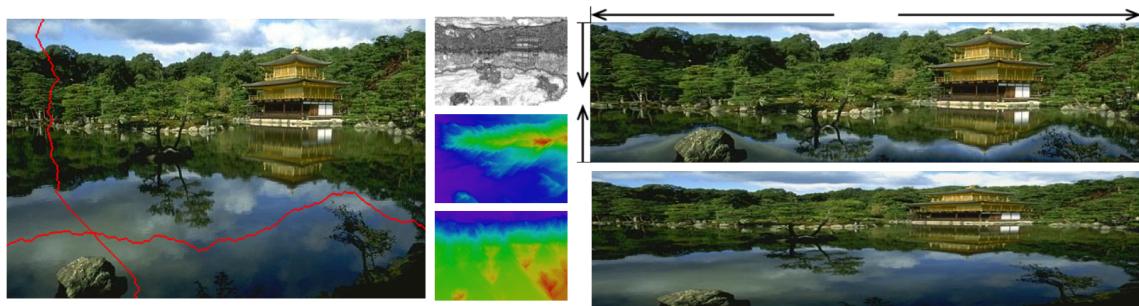


图 1: Seam Carving 介绍

最终选题：Seam 的求解 ( 60% )，使用不同的算子进行测试 ( 15% )，图像放大 ( 10% )，对象移除 ( 20% )，自选例子的 Fancy 效果 ( 10% )，共计 115%。

## 2 算法原理

### 2.1 Seam 的求解

Seam 的求解本质是动态规划算法，大致描述如下：

- ( 1 ) 计算图像中每个像素的能量（使用卷积核加滤波实现）
- ( 2 ) 使用动态规划算法找出能量最低的一条线，这条线被称为 Seam

## 2.2 图像缩小

图像缩小是不断寻找 Seam 的过程，大致描述如下：

- ( 1 ) 在  $I_{W*H}$  找出 Seam
- ( 2 ) 从图中删去这条线，得到  $I_{(W-1)*H}$
- ( 3 ) 如果要得到  $I_{(W-K)*H}$ ，则重复上述过程  $K$  次即可

## 2.3 图像放大

图像放大是图像缩小的逆过程，大致描述如下：

- ( 1 ) 如果要得到  $I_{(W+K)*H}$ ，先将图片缩小到  $I_{(W-K)*H}$ ，记录删去的  $K$  条 Seam
- ( 2 ) 将删除的  $K$  条 Seam 复制一遍插回原图即可，插入时根据周围像素颜色做适度调整

## 2.4 对象移除

对象移除是将图片改变能量后求解 Seam 的过程，大致描述如下：( 1 ) 用户选择要删除和要保留的区域，对删除区域能量值减少 INF，对保留区域能量值增加 INF（其中 INF 是极大值，可设为  $1e5$  左右，若设置太大 double 将会溢出）

- ( 2 ) 在改变能量后的  $I_{W*H}$  找出 Seam
- ( 3 ) 从图中删去这条线，得到  $I_{(W-1)*H}$
- ( 4 ) 上述过程重复进行，直至删除区域全被删除为止

## 2.5 使用不同的算子进行测试

卷积是图像处理的重要方法，也是基于线性代数形成的一套理论。不同的卷积核具有不同的特征提取能力，因此使用不同的卷积核最终计算的能量值也会有所不同，因此，使用不同的卷积核得到的 Seam 图也会有差异。在本次实验中，共使用了以下四种算子，分别是：我的算子（0号算子），Sobel 算子（1号算子），Laplace 算子（2号算子），Roberts 算子（3号算子），算子的矩阵描述如下：

( 0 ) 我的算子：

水平卷积核 : [0, 0, 0; 0, 1, -1; 0, 0, 0]

数值卷积核 : [0, 0, 0; 0, 1, 0; 0, -1, 0]

( 1 ) Sobel 算子

水平卷积核 : [-1, 0, 1; -2, 0, 2; -1, 0, 1]

数值卷积核 : [-1, -2, -1; 0, 0, 0; 1, 2, 1]

( 2 ) Laplace 算子

水平卷积核 : [0, 1, 0; 1, -4, 1; 0, 1, 0]

数值卷积核 : [0, 1, 0; 1, -4, 1; 0, 1, 0]

( 3 ) Roberts 算子

水平卷积核 : [1, 0; 0, -1]

数值卷积核 : [0, 1; -1, 0]

程序运行过程会显示出用算子卷积产生的梯度图, 因为不作为提交要求故没有输出为文件

## 2.6 自选例子的 Fancy 效果

详见结果分析

## 3 实验结果

### 3.1 图像缩放

对提供的 6 张图片缩小原照片大小的 20%, 并执行逆过程放大原照片大小的 20%, 全部采用 Sobel 算子, 结果和 Seam 图如下:



图 2: 1 原图



图 3: 1 缩小图



图 4: 1 放大图



图 5: 1 Seam 图

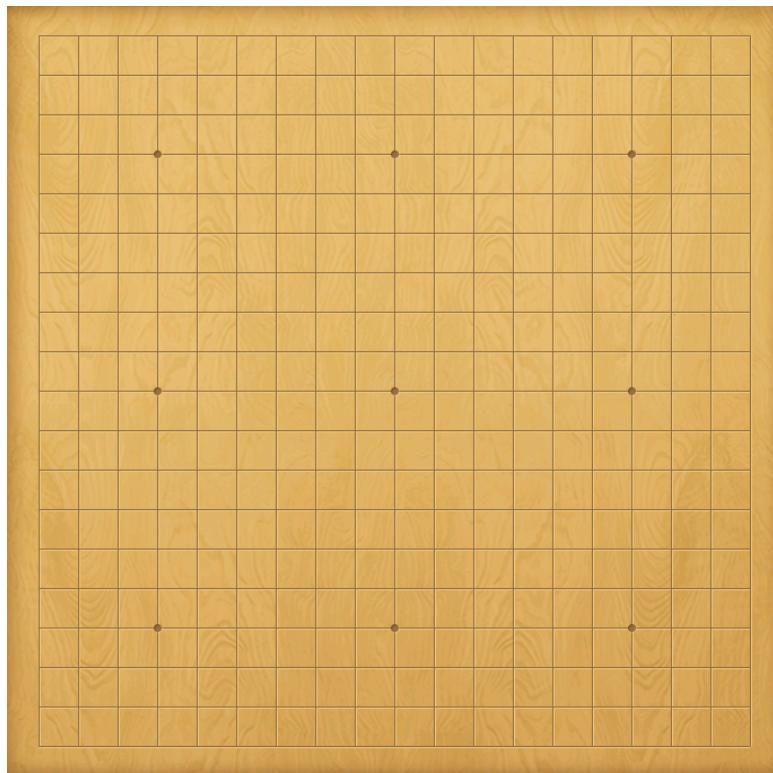


图 6: 2 原图

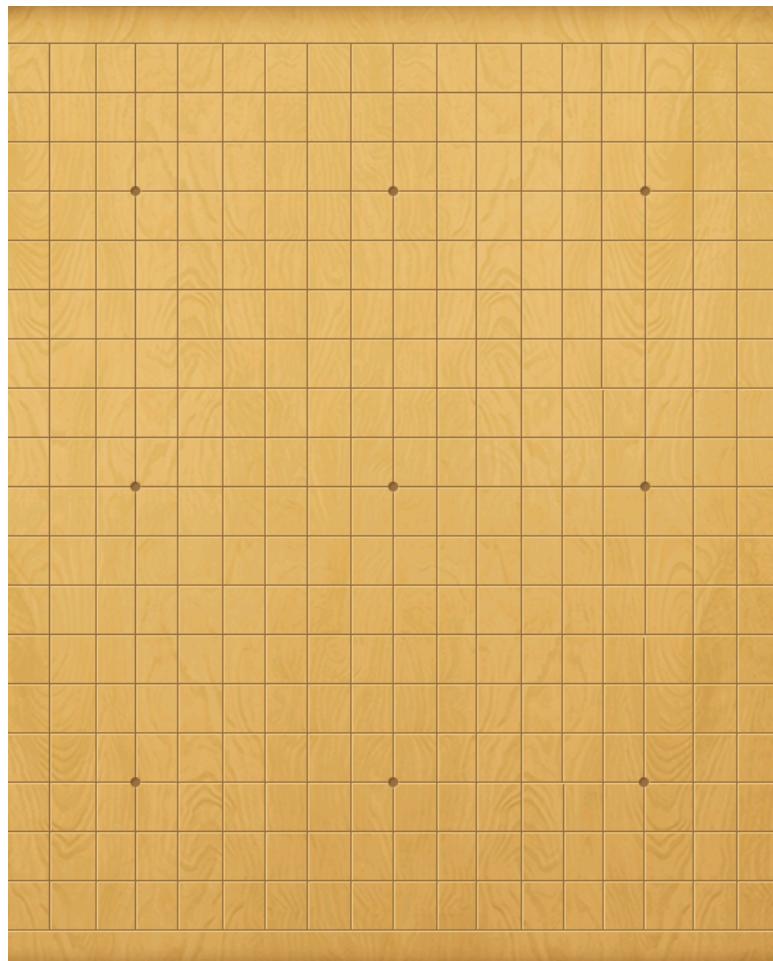


图 7: 2 缩小图

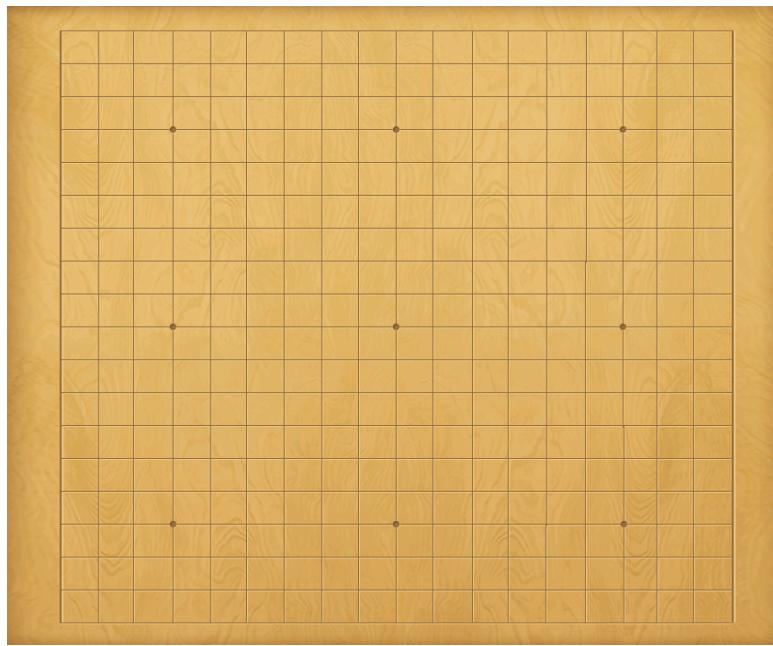


图 8: 2 放大图

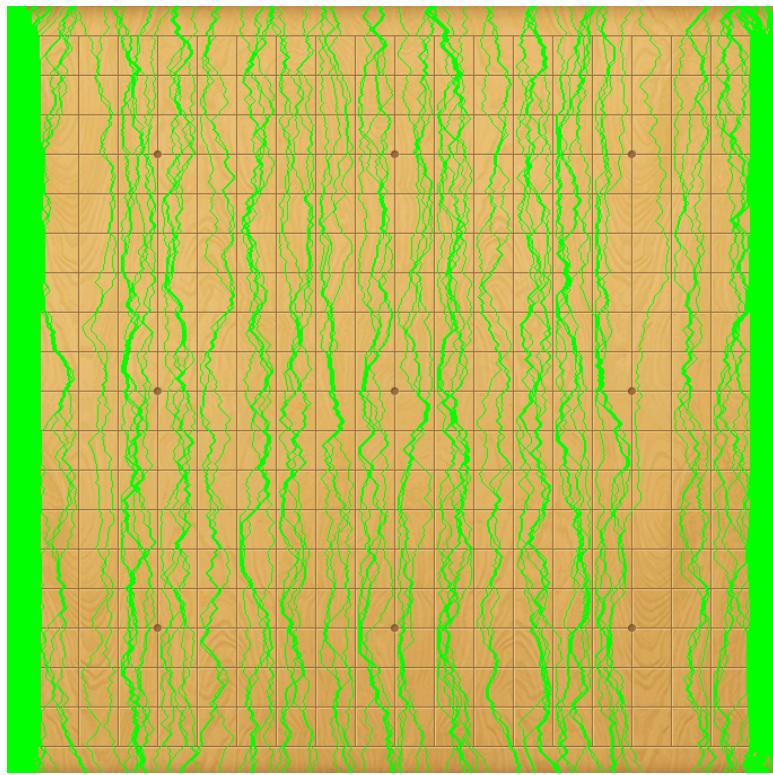


图 9: 2 Seam 图



图 10: 3 原图



图 11: 3 缩小图



图 12: 3 放大图



图 13: 3 Seam 图



图 14: 4 原图



图 15: 4 缩小图



图 16: 4 放大图

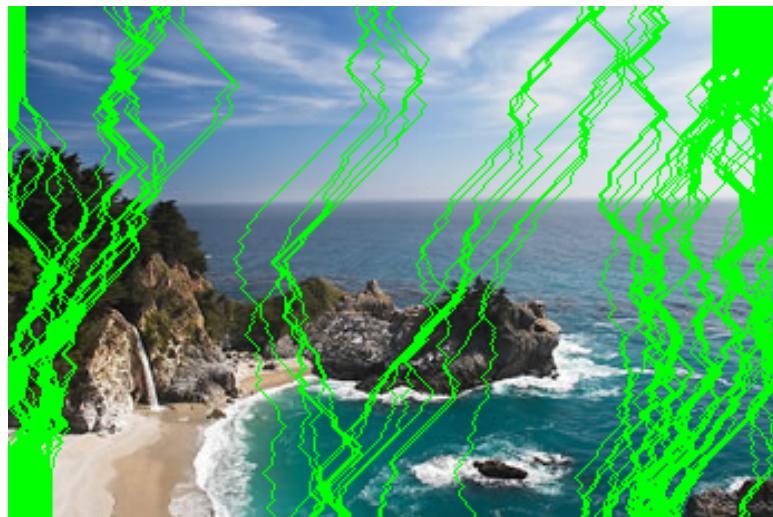


图 17: 4 Seam 图



图 18: 5 原图



图 19: 5 缩小图



图 20: 5 放大图

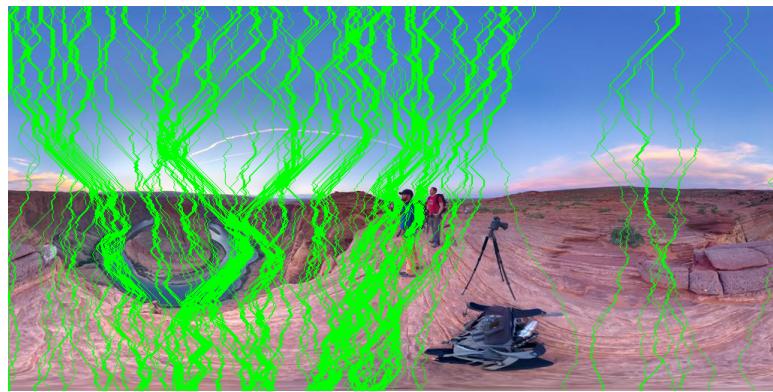


图 21: 5 Seam 图



图 22: 6 原图



图 23: 6 缩小图



图 24: 6 放大图



图 25: 6 Seam 图

从结果我们可以看出，Seam Carving 算法确实是图像缩放的有力武器，可以轻松的实现图片无失真的放大与缩小，效果非常好，几乎看不出来原图被缩放过。

### 3.2 对象移除

对提供的 6 张图片，前 5 张选取部分成分作为移除对象，全部采用 Sobel 算子，结果和 Seam 图如下：



图 26: 1 原图



图 27: 1 移除图

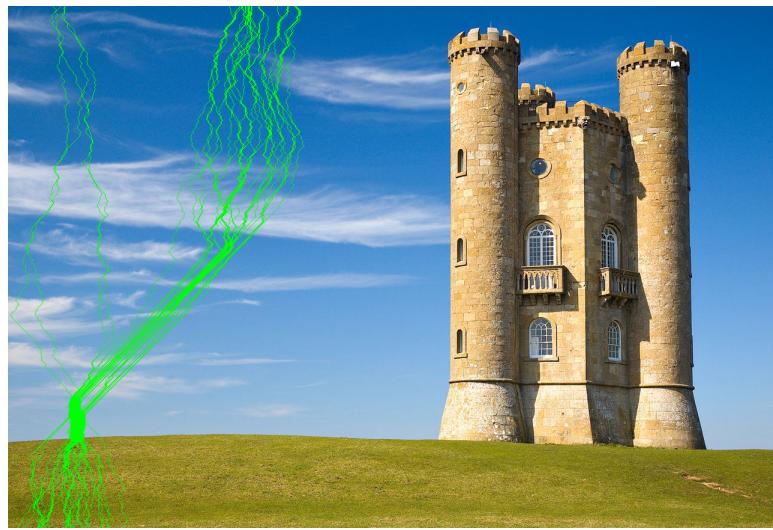


图 28: 1 Seam 图

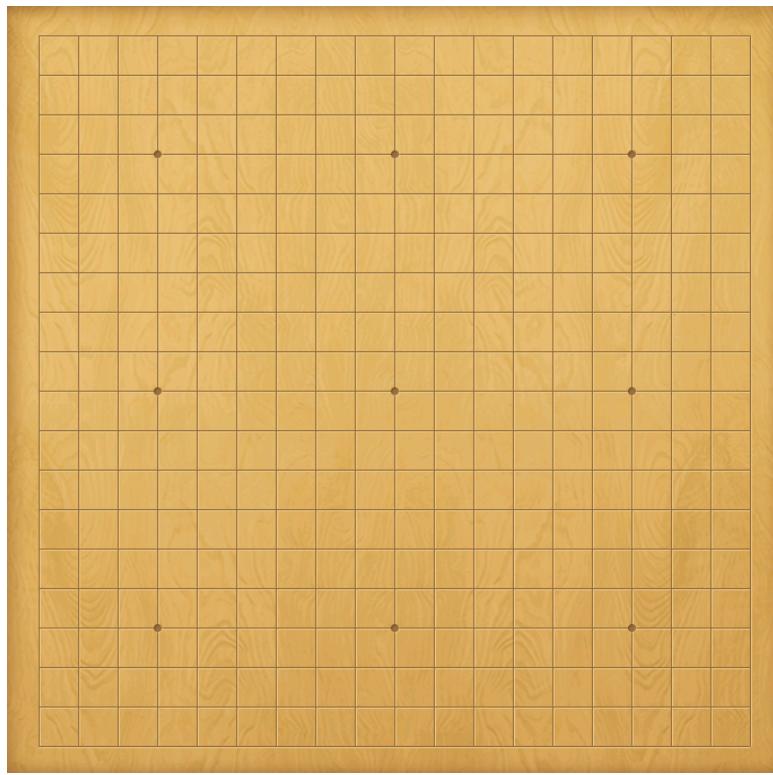


图 29: 2 原图

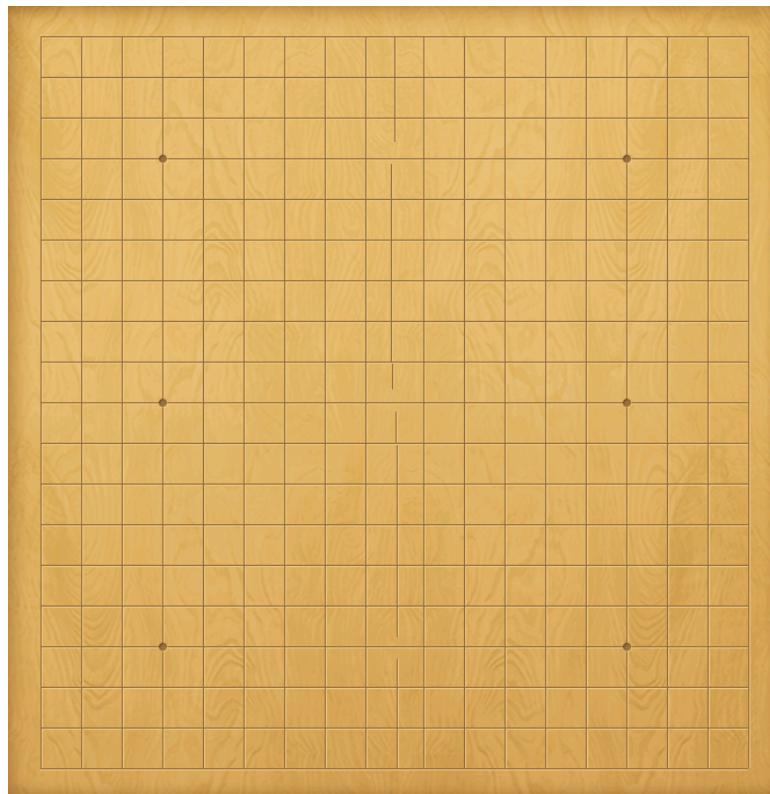


图 30: 2 移除图

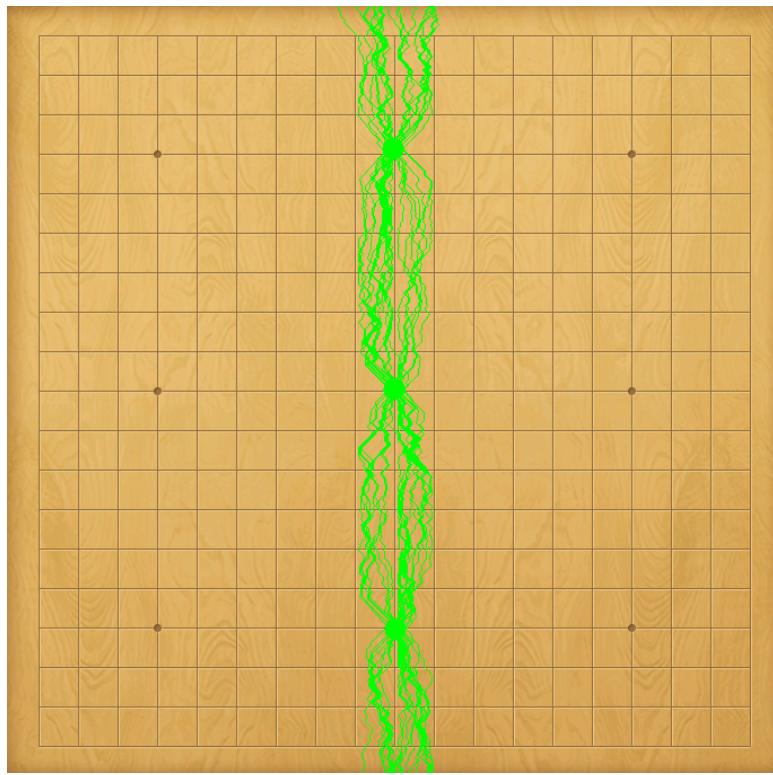


图 31: 2 Seam 图



图 32: 3 原图



图 33: 3 移除图

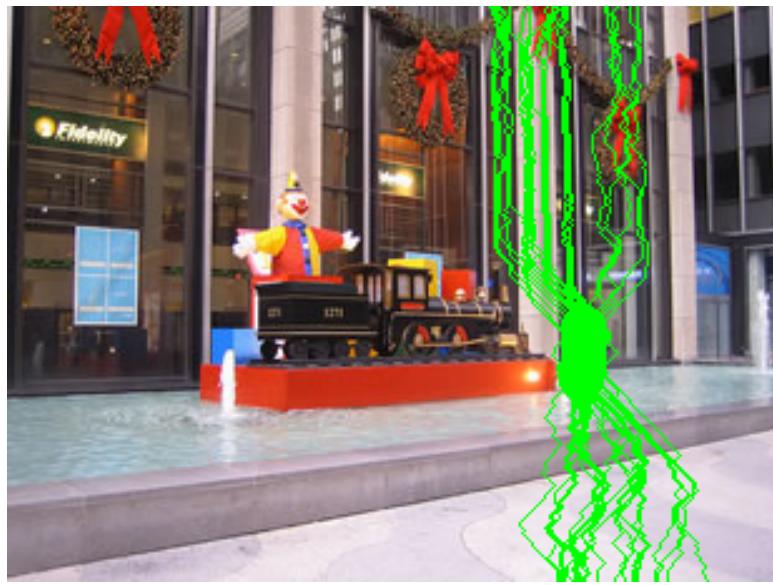


图 34: 3 Seam 图



图 35: 4 原图



图 36: 4 移除图

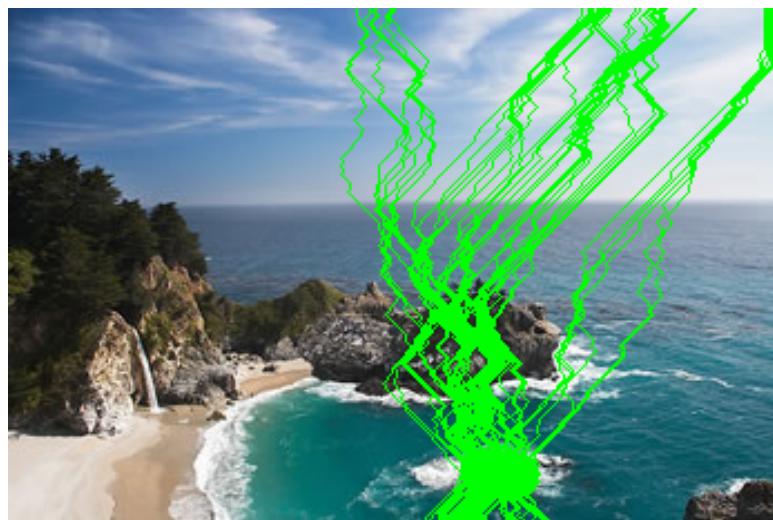


图 37: 4 Seam 图



图 38: 5 原图



图 39: 5 移除图

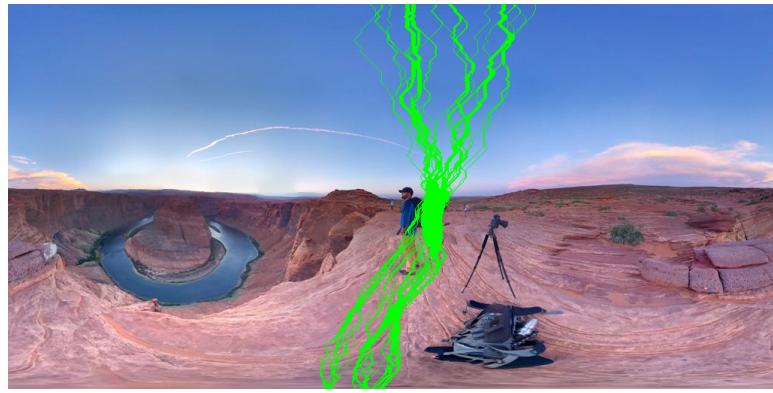


图 40: 5 Seam 图

从结果我们可以看出，由于对删除区域的能量值减去了一个很大的常数（ INF ），使得该区域的能量值很低，动态规划得到的 Seam 总是经过删除区域；由于保留区域的能量值加上了一个很大的常数（ INF ），使得该区域的能量值很高，动态规划得到的 Seam 总是不经过保留区域。综合以上两点，可以轻松的实现对象的移除。

### 3.3 算子测试

由于棋盘（ 2.png ）非常典型，图像内容均为重要内容，故以此作为测试图片。使用 4 种算子，分别为我的算子、 Sobel 算子、 Laplace 算子、 Roberts 算子，进行图像缩小与放大，结果和 Seam 图如下：

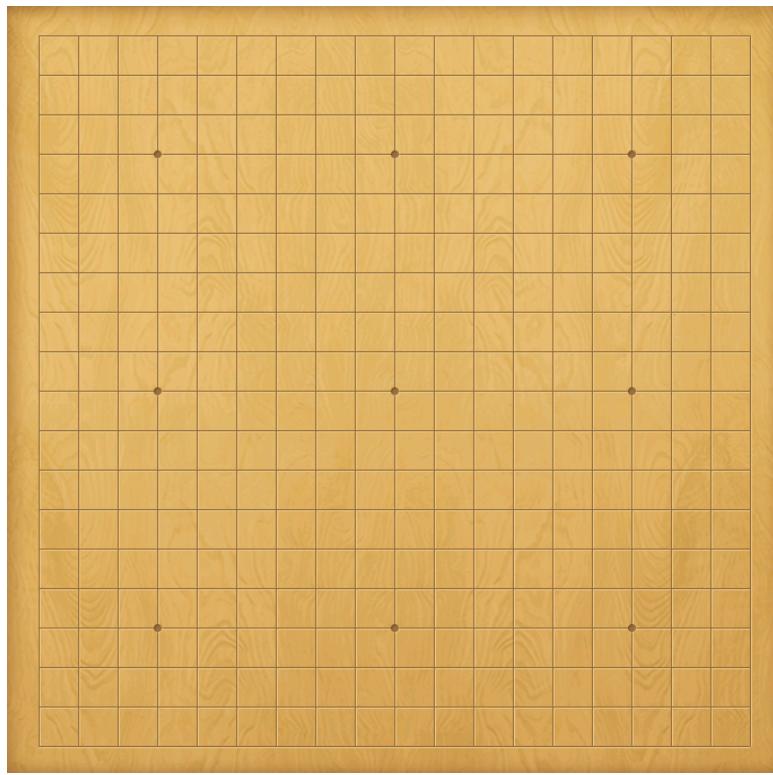


图 41: 原图

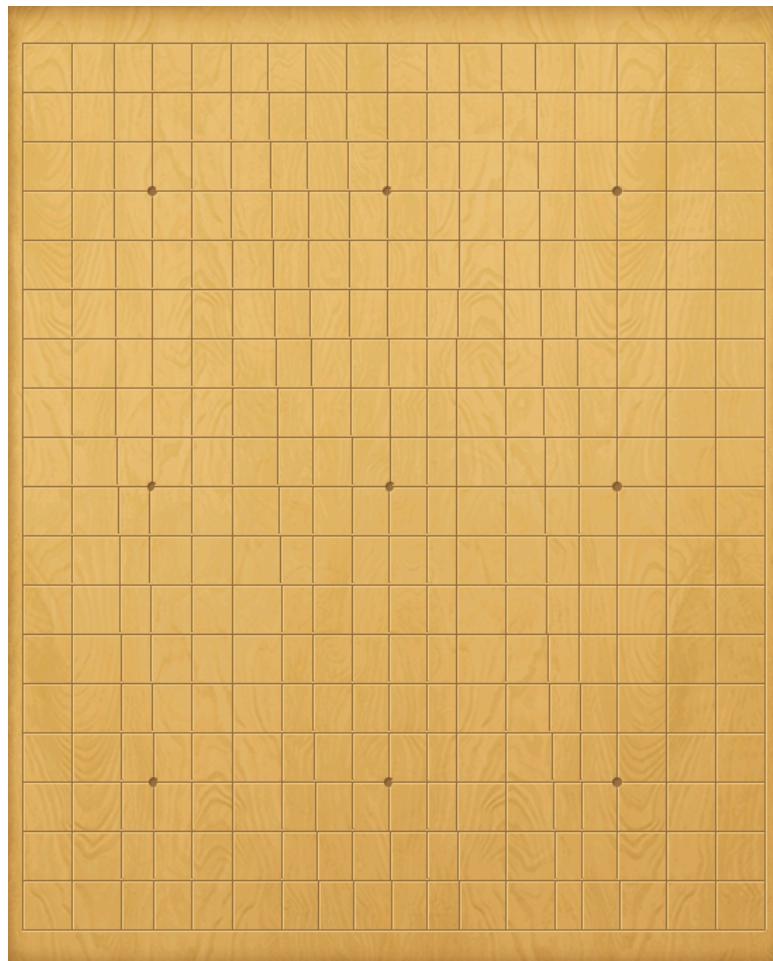


图 42: 0 我的算子缩小图

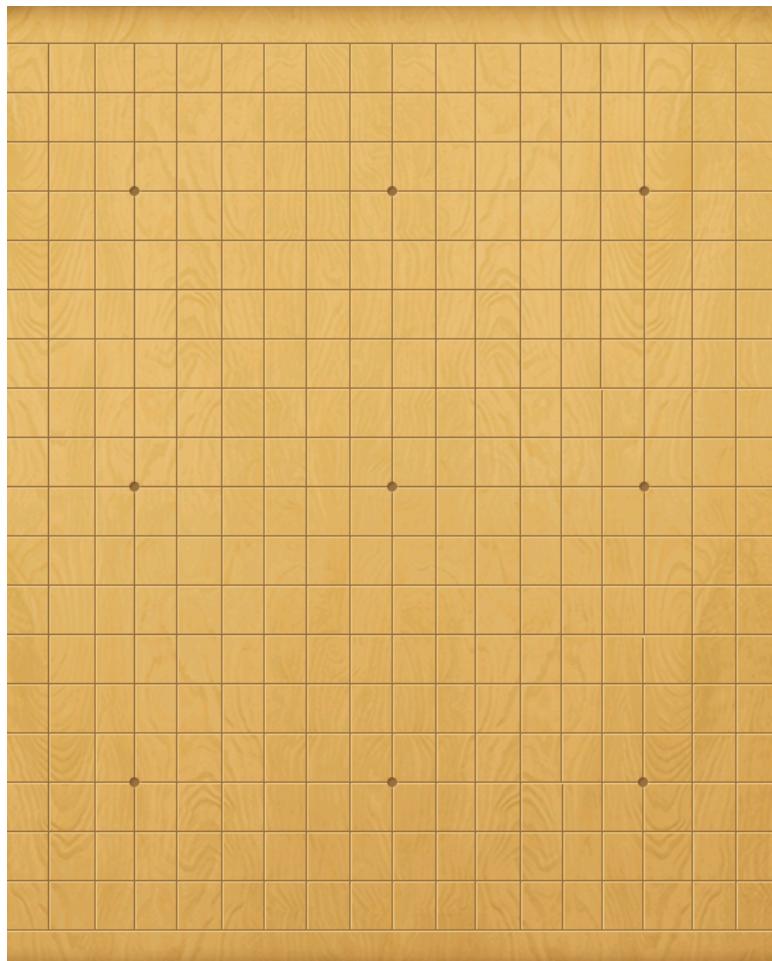


图 43: 1 Sobel 算子缩小图

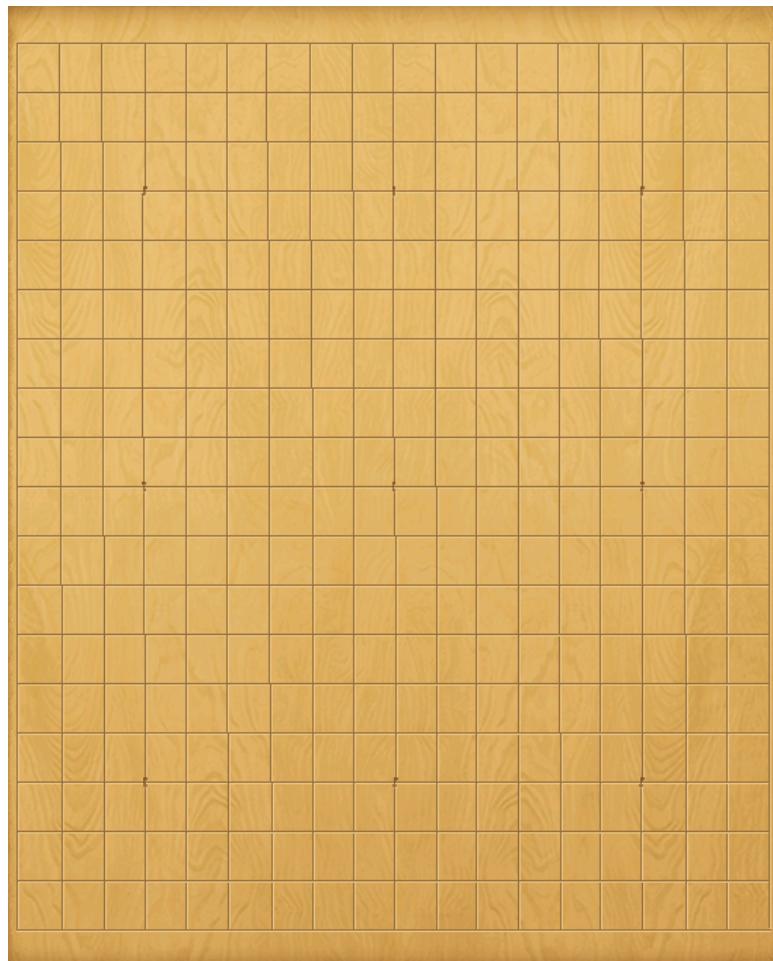


图 44: 2 Laplace 算子缩小图

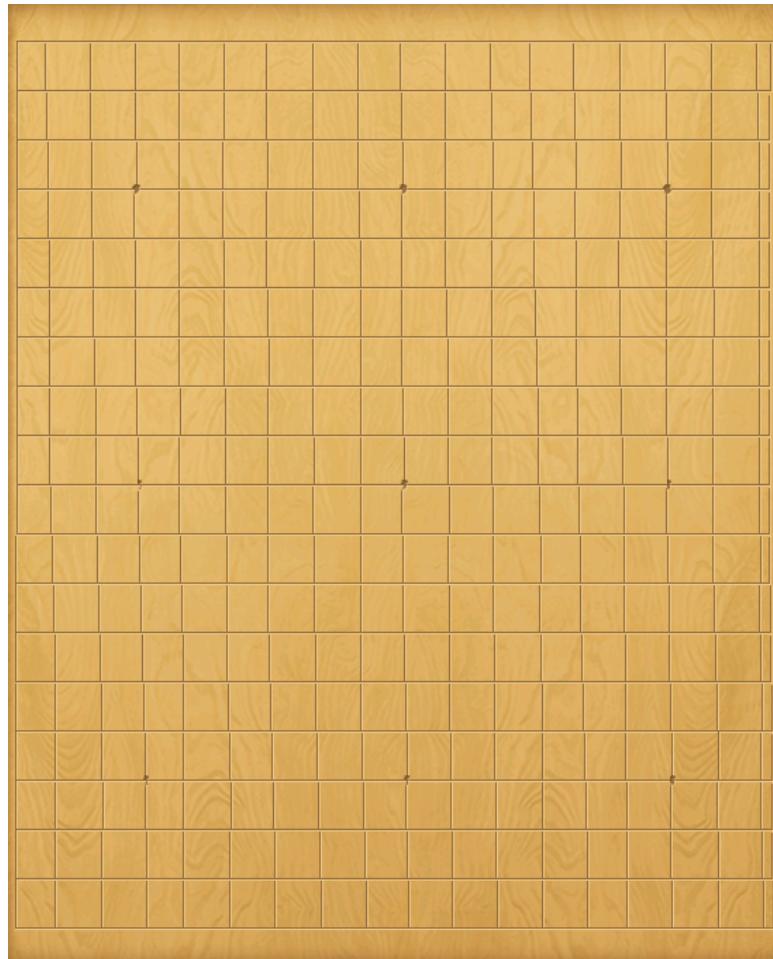


图 45: 3 Roberts 算子缩小图

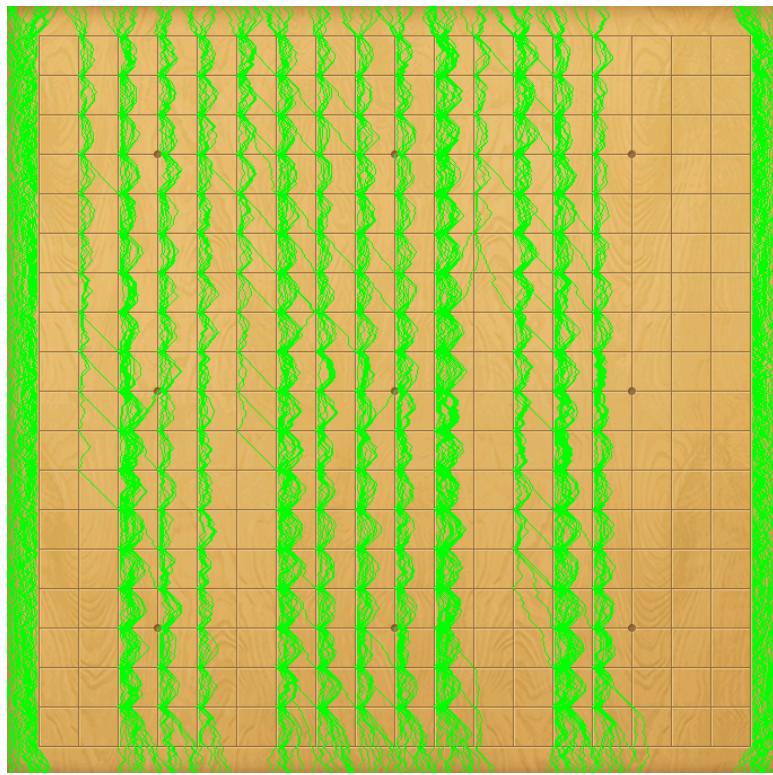


图 46: 0 我的算子 Seam 图

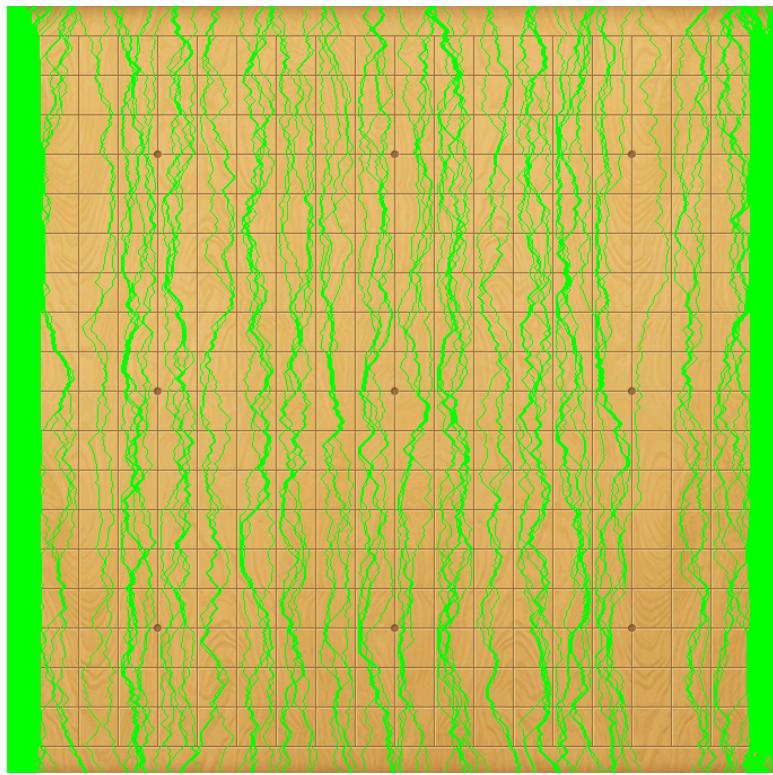


图 47: 1 Sobel 算子 Seam 图

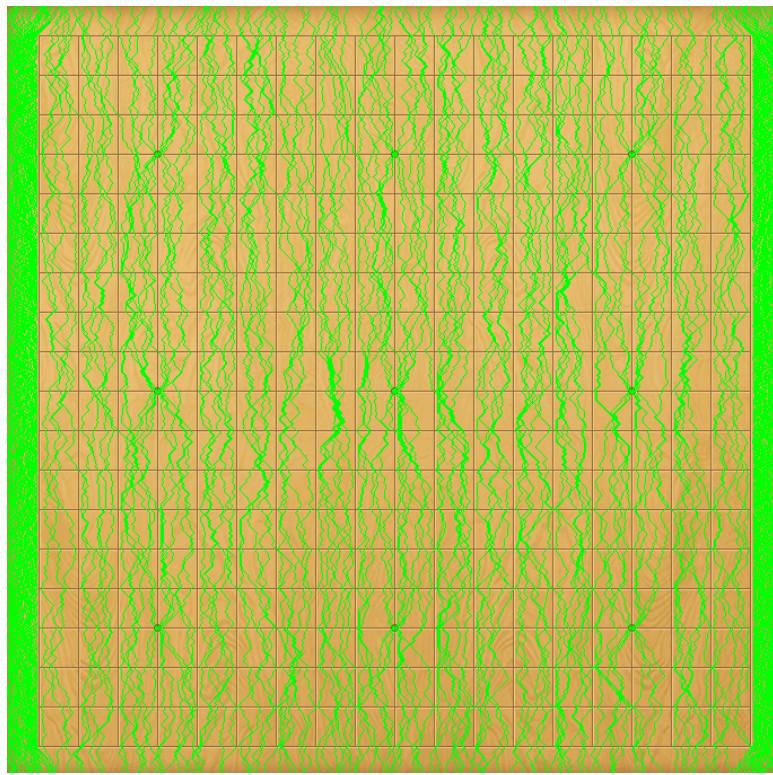


图 48: 2 Laplace 算子 Seam 图

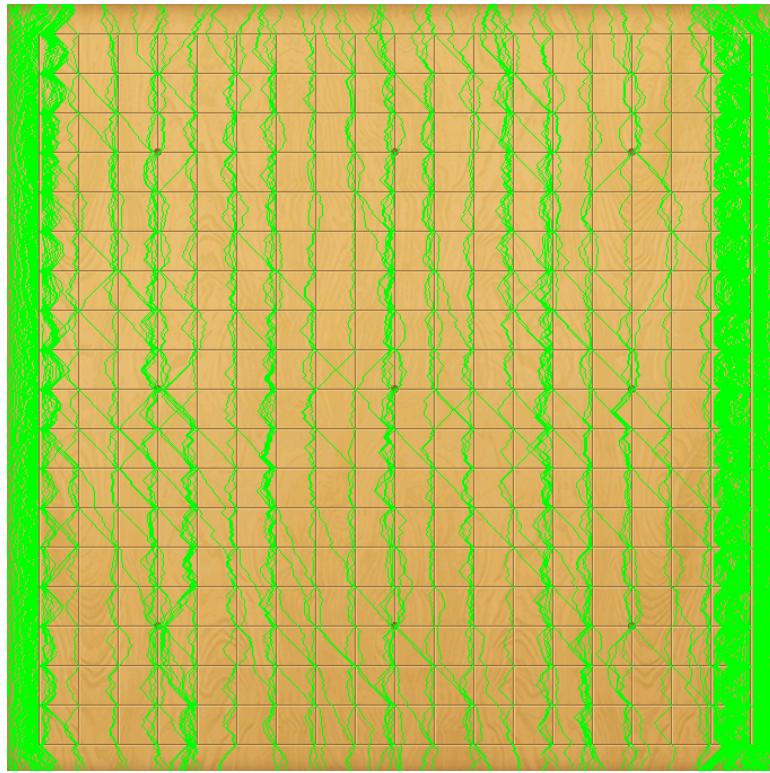


图 49: 3 Roberts 算子 Seam 图

不同的算子具有不同的特征提取能力，因此使用不同的卷积核最终计算的能量值也会有所不同，因此，使用不同的卷积核得到的 Seam 图也会有差异，可以从 Seam 图中清晰地看出来这样的差异。因此针对不同的图像，使用针对性的算子，可以更好的实现图像的缩放。

### 3.4 缩小比例

以海景图 ( 4.jpg ) 作为测试图片，采用 Sobel 算子缩小原图大小的 10%， 20%， 30%， 40%， 50%， 60%， 结果和 Seam 图如下：



图 50: 原图



图 51: 缩小 10% 结果图



图 52: 缩小 20% 结果图



图 53: 缩小 30% 结果图



图 54: 缩小 40% 结果图



图 55: 缩小 50% 结果图



图 56: 缩小 60% 结果图



图 57: 缩小 10%Seam 图

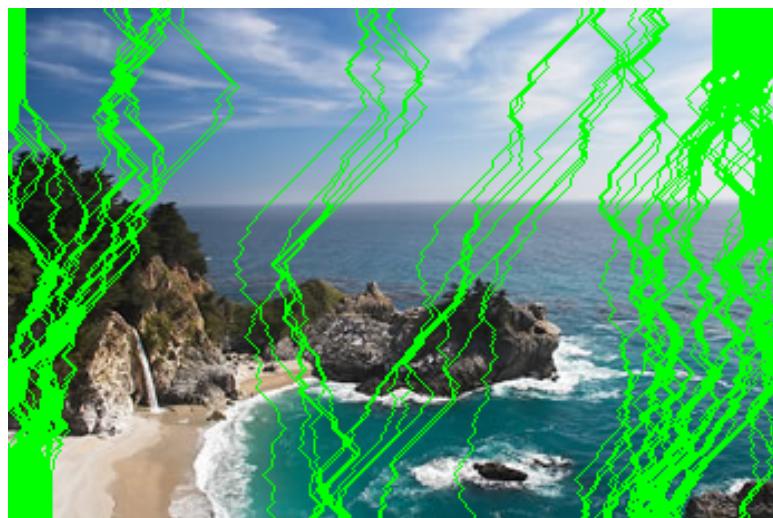


图 58: 缩小 20%Seam 图

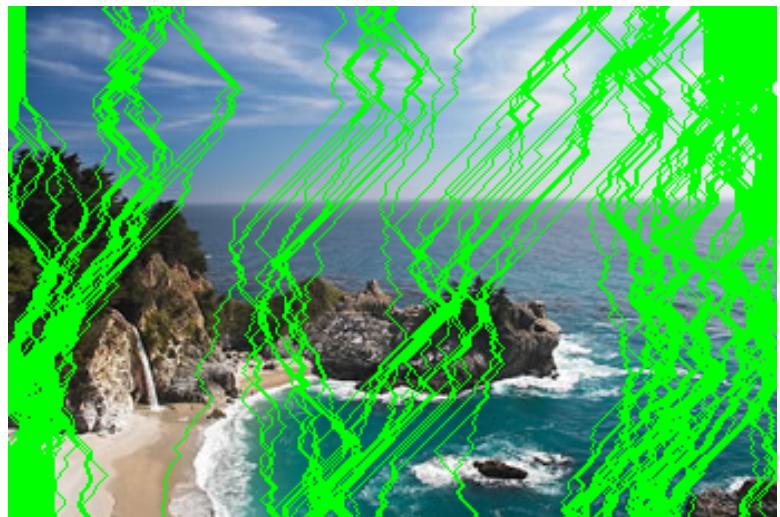


图 59: 缩小 30%Seam 图

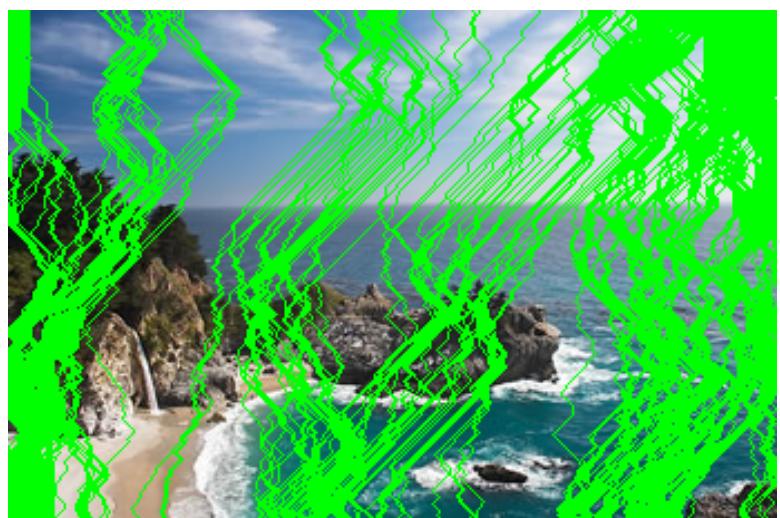


图 60: 缩小 40%Seam 图



图 61: 缩小 50%Seam 图

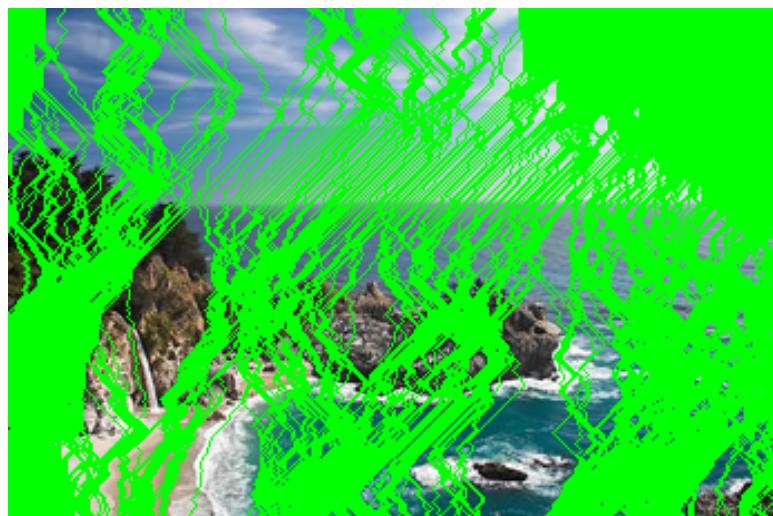


图 62: 缩小 60%Seam 图

即使是缩小到 60%，我们可以看出图像仍然没有失真，效果表现良好，甚至好的超乎预期。

## 4 实验心得

这次大作业非常有意思，图像的无失真缩放一直是困扰我多年的问题，简单的拉伸和缩小会导致图像失真，也没有较好的方法可以解决这个问题。

Seam Carving 通过动态规划和能量的转换关系，实现 Seam 的求解，从而在图片中一步步的“砍”掉能量最小的边，从而神奇的实现了图像的缩放，并以此为基础可以极其简单的实现图片放大与对象移除。但在实现图像放大的时候，如何记录清楚图像像素点的对应关系，是一个很有技巧的问题，我在这里调试了很久，最终采用非常巧妙的方式解决了，感兴趣的读者可以参见代码。

通过本次实验，我切实感受到 Seam Carving 真的是一个强大的算法。这个算法无论是原理还是实现均相对较为简单，我在一个文件中 (main.cpp) 用了短短 400 行代码便实现了这么多功能，并且效果非常好，超乎我的预料，让我感受到算法在图像处理中的力量。

为了清晰明了的展示该算法，我在每步求解 Seam 的时候，都在当前图中标出了 Seam，并延时 50ms，这样便实现了一步一步“砍”去 Seam 的动画效果，可以更清晰的被使用者理解。同时在对象移除过程中实现了画笔功能，可以让读者用鼠标标出删除区域（绿色）与保留区域（红色），从而执行对象移除算法，在此感谢提供该 GUI 的同学。但是由于时间精力有限，我没有实现双向缩放的处理，略有遗憾，希望今后有机会可以继续尝试。

## 5 程序使用

### 5.1 编译方式

在终端中输入 `g++ main.cpp -O3 -std=c++11 `pkg-config --cflags --libs opencv``

### 5.2 使用方式

在终端中输入 `./a.out`，依次输入三个参数：图片文件名，缩放像素数（若为正则为先缩小后扩大，若为负则为缩小），使用算子（0 为我的算子，1 为 Sobel 算子，2 为 Laplace 算子，3 为 Roberts 算子），执行图片缩放过程，执行过程中会在窗口中显示实时动画，执行完成后将显示缩放图和 Seam 图。

缩放执行完成后，在新窗口涂出删除区域（显示为绿色），按 Esc 结束填涂，继续涂出保留区域（显示为红色），按 Esc 结束填图，执行对象移除过程，执行过程中会在窗口中显示实时动画，执行完成后将显示缩放图和 Seam 图。程序进入暂停模式。

输出的四个文件为：`result.png`、`seam.png`、`result_remove.png`、`seam_remove.png`，分别为缩放图、缩放 Seam 图、对象删除图、对象删除 Seam 图。

### 5.3 文件组织

- main.cpp 为源代码
- report.pdf 为实验报告
- 缩放测试内含原始图 1-6 使用 Sobel 算子缩小放大 20% 的结果图和 Seam 图
- 对象移除测试内含原始图 1-5 使用 Sobel 算子对象移除的结果图和 Seam 图
- 算子测试内含原始图 2 和使用不同算子缩小 20% 的结果图和 Seam 图
- 缩小比例测试内含原始图 4 和使用不同缩放比例的结果和 Seam 图

### 5.4 测试环境

系统 : macOS Sierra

CPU : Intel Core i5 2.7GHz

内存 : 8GB

## 6 源代码

```
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <cstdio>
#include <iostream>

#define MAX_COLS 2000
#define INF 1e5

struct MouseArgs // 用于和回调函数交互
{
    cv::Mat& img; // 显示在窗口中的那张图
    cv::Mat& mask; // 用户绘制的选区 (删除/保留)
    cv::Vec3b& color; // 用来高亮选区的颜色
    MouseArgs(cv::Mat& img, cv::Mat& mask, cv::Vec3b& color): img(img), mask(mask), color(color)
    {}
};

void CalculateEnergy(const cv::Mat& srcMat, cv::Mat& dstMat, cv::Mat& traceMat)
{
    //srcMat是能量矩阵, dstMat是累计能量矩阵 (用于DP), traceMat是轨迹矩阵
    dstMat = srcMat.clone(); //不用“=”, 防止两个矩阵指向的都是同一个矩阵, 现在只需要传里面的数值
    for (int i = 1; i < srcMat.rows; i++) { //从第2行开始计算
        //第一列
        if (dstMat.at<float>(i - 1, 0) <= dstMat.at<float>(i - 1, 1)) {
```

```

        dstMat.at<float>(i, 0) = srcMat.at<float>(i, 0) + dstMat.at<float>(i - 1, 0);
        traceMat.at<float>(i, 0) = 1; //traceMat记录当前位置的上一行应取那个位置, 上左为0, 上中1, 上右
        //为2
    }
    else {
        dstMat.at<float>(i, 0) = srcMat.at<float>(i, 0) + dstMat.at<float>(i - 1, 1);
        traceMat.at<float>(i, 0) = 2;
    }
    //中间列
    for (int j = 1; j < srcMat.cols - 1; j++) {
        float k[3];
        k[0] = dstMat.at<float>(i - 1, j - 1);
        k[1] = dstMat.at<float>(i - 1, j);
        k[2] = dstMat.at<float>(i - 1, j + 1);
        int index = 0;
        if (k[1] < k[0])
            index = 1;
        if (k[2] < k[index])
            index = 2;
        dstMat.at<float>(i, j) = srcMat.at<float>(i, j) + dstMat.at<float>(i - 1, j - 1 +
            index);
        traceMat.at<float>(i, j) = index;
    }
    //最后一列
    if (dstMat.at<float>(i - 1, srcMat.cols - 1) <= dstMat.at<float>(i - 1, srcMat.cols - 2))
    {
        dstMat.at<float>(i, srcMat.cols - 1) = srcMat.at<float>(i, srcMat.cols - 1) + dstMat.at<float>(i - 1, srcMat.cols - 1);
        traceMat.at<float>(i, srcMat.cols - 1) = 1;
    }
    else {
        dstMat.at<float>(i, srcMat.cols - 1) = srcMat.at<float>(i, srcMat.cols - 1) + dstMat.at<float>(i - 1, srcMat.cols - 2);
        traceMat.at<float>(i, srcMat.cols - 1) = 0;
    }
}
}

// 找出最小能量线
void GetMinEnergyTrace(const cv::Mat& energyMat, const cv::Mat& traceMat, cv::Mat& minTrace)
{
    //energyMat是累计能量矩阵, traceMat是轨迹矩阵, minTrace是最小能量路径
    int row = energyMat.rows - 1;// 取的是energyMat最后一行的数据, 所以行标是rows-1
    int index = 0; // 保存的是最小那条轨迹的最下面点在图像中的列标
    // 获得index, 即最后那行最小值的位置
    for (int i = 1; i < energyMat.cols; i++) {

```

```

        if (energyMat.at<float>(row, i) < energyMat.at<float>(row, index)) {
            index = i;
        }
    }

    // 以下根据traceMat, 得到minTrace, minTrace是多行一列矩阵
    minTrace.at<float>(row, 0) = index;
    int tmpIndex = index;
    for (int i = row; i > 0; i--) {
        int temp = traceMat.at<float>(i, tmpIndex); // 当前位置traceMat所存的值
        if (temp == 0) { // 往左走
            tmpIndex = tmpIndex - 1;
        }
        else if (temp == 2) { // 往右走
            tmpIndex = tmpIndex + 1;
        } // 如果temp = 1, 则往正上走, tmpIndex不需要做修改
        minTrace.at<float>(i - 1, 0) = tmpIndex;
    }
}

// 删掉一列
void DeleteOneCol(const cv::Mat& srcMat, cv::Mat& dstMat, const cv::Mat& minTrace, cv::Mat& deletedLine)
{
    for (int i = 0; i < dstMat.rows; i++) {
        int k = minTrace.at<float>(i, 0);
        for (int j = 0; j < k; j++)
            dstMat.at<cv::Vec3b>(i, j) = srcMat.at<cv::Vec3b>(i, j);
        for (int j = k; j < dstMat.cols; j++)
            dstMat.at<cv::Vec3b>(i, j) = srcMat.at<cv::Vec3b>(i, j + 1);
        deletedLine.at<cv::Vec3b>(i, 0) = srcMat.at<cv::Vec3b>(i, k);
    }
}

// 恢复一列
void RecoverOneCol(const cv::Mat& srcMat, cv::Mat& dstMat, const cv::Mat& minTrace, const cv::Mat& deletedLine)
{
    cv::Mat recorvedImage(srcMat.rows, srcMat.cols + 1, CV_8UC3);
    for (int i = 0; i < srcMat.rows; i++) {
        int k = minTrace.at<float>(i);
        for (int j = 0; j < k; j++)
            recorvedImage.at<cv::Vec3b>(i, j) = srcMat.at<cv::Vec3b>(i, j);
        recorvedImage.at<cv::Vec3b>(i, k) = deletedLine.at<cv::Vec3b>(i, 0);
        for (int j = k + 1; j < srcMat.cols + 1; j++)
            recorvedImage.at<cv::Vec3b>(i, j) = srcMat.at<cv::Vec3b>(i, j - 1);
    }
}

```

```

dstMat = recorvedImage.clone();

//显示恢复的轨迹
cv::Mat tmpImage = recorvedImage.clone();
for (int i = 0; i < tmpImage.rows; i++) {
    int k = minTrace.at<float>(i, 0);
    tmpImage.at<cv::Vec3b>(i, k)[0] = 0;
    tmpImage.at<cv::Vec3b>(i, k)[1] = 255;
    tmpImage.at<cv::Vec3b>(i, k)[2] = 0;
}
cv::imshow("Seam Carving...", tmpImage);
}

void ShowSeam(const cv::Mat& srcMat, cv::Mat& dstMat, const cv::Mat& minTrace, const cv::Mat&
deletedLine) {
cv::Mat recorvedImage(srcMat.rows, srcMat.cols + 1, CV_8UC3);
for (int i = 0; i < srcMat.rows; i++) {
    int k = minTrace.at<float>(i);
    for (int j = 0; j < k; j++)
        recorvedImage.at<cv::Vec3b>(i, j) = srcMat.at<cv::Vec3b>(i, j);
    recorvedImage.at<cv::Vec3b>(i, k)[0] = 0;
    recorvedImage.at<cv::Vec3b>(i, k)[1] = 255;
    recorvedImage.at<cv::Vec3b>(i, k)[2] = 0;
    for (int j = k + 1; j < srcMat.cols + 1; j++)
        recorvedImage.at<cv::Vec3b>(i, j) = srcMat.at<cv::Vec3b>(i, j - 1);
}
dstMat = recorvedImage.clone();
}

void Shrink(const cv::Mat& image, cv::Mat& outImage, cv::Mat& outMinTrace, cv::Mat&
outDeletedLine, const cv::Mat& removeMask, const cv::Mat& remainMask, int kernel)
{
    cv::Mat image_gray(image.rows, image.cols, CV_8U, cv::Scalar(0));
    cv::cvtColor(image, image_gray, CV_BGR2GRAY); //彩色图像转换为灰度图像

    cv::Mat gradient_H(image.rows, image.cols, CV_32F, cv::Scalar(0)); //水平梯度矩阵
    cv::Mat gradient_V(image.rows, image.cols, CV_32F, cv::Scalar(0)); //垂直梯度矩阵

    cv::Mat kernel_mine_H = (cv::Mat_<float>(3, 3) << 0, 0, 0, 0, 1, -1, 0, 0, 0); //求水平梯度所
    使用的卷积核 ( 赋初始值 )
    cv::Mat kernel_mine_V = (cv::Mat_<float>(3, 3) << 0, 0, 0, 0, 1, 0, 0, -1, 0); //求垂直梯度所
    使用的卷积核 ( 赋初始值 )

    cv::Mat kernel_Sobel_H = (cv::Mat_<float>(3, 3) << -1, 0, 1, -2, 0, 2, -1, 0, 1);
    cv::Mat kernel_Sobel_V = (cv::Mat_<float>(3, 3) << -1, -2, -1, 0, 0, 0, 1, 2, 1);

    cv::Mat kernel_Laplace_H = (cv::Mat_<float>(3, 3) << 0, 1, 0, 1, -4, 1, 0, 1, 0);
}

```

```

cv::Mat kernel_Laplace_V = (cv::Mat_<float>(3, 3) << 0, 1, 0, 1, -4, 1, 0, 1, 0);

cv::Mat kernel_Roberts_H = (cv::Mat_<float>(2, 2) << 1, 0, 0, -1);
cv::Mat kernel_Roberts_V = (cv::Mat_<float>(2, 2) << 0, 1, -1, 0);

cv::Mat kernel_H;
cv::Mat kernel_V;

//选择kernel
if (kernel == 0) {
    kernel_H = kernel_mine_H;
    kernel_V = kernel_mine_V;
}
else if (kernel == 1) {
    kernel_H = kernel_Sobel_H;
    kernel_V = kernel_Sobel_V;
}
else if (kernel == 2) {
    kernel_H = kernel_Laplace_H;
    kernel_V = kernel_Laplace_V;
}
else if (kernel == 3) {
    kernel_H = kernel_Roberts_H;
    kernel_V = kernel_Roberts_V;
}

cv::filter2D(image_gray, gradiant_H, gradiant_H.depth(), kernel_H);
cv::filter2D(image_gray, gradiant_V, gradiant_V.depth(), kernel_V);

cv::Mat gradMag_mat(image.rows, image.rows, CV_32F, cv::Scalar(0));
cv::add(cv::abs(gradiant_H), cv::abs(gradiant_V), gradMag_mat); //水平与垂直滤波结果的绝对值相加,
可以得到近似梯度大小

for (int i = 0; i < image.rows; i++) {
    for (int j = 0; j < image.cols; j++) {
        if (removeMask.at<char>(i, j) == 1)
            gradMag_mat.at<float>(i, j) -= INF;
        if (remainMask.at<char>(i, j) == 1)
            gradMag_mat.at<float>(i, j) += INF;
    }
}

// 如果要显示梯度大小这个图, 因为gradMag_mat深度是CV_32F, 所以需要先转换为CV_8U
cv::Mat testMat;
gradMag_mat.convertTo(testMat, CV_8U, 1, 0);
cv::imshow("Gradient", testMat);

```

```

//计算能量线
cv::Mat energyMat(image.rows, image.cols, CV_32F, cv::Scalar(0)); //累计能量矩阵
cv::Mat traceMat(image.rows, image.cols, CV_32F, cv::Scalar(0)); //能量最小轨迹矩阵
CalculateEnergy(gradMag_mat, energyMat, traceMat);

//找出最小能量线
cv::Mat minTrace(image.rows, 1, CV_32F, cv::Scalar(0)); //能量最小轨迹矩阵中的最小的一条的轨迹
GetMinEnergyTrace(energyMat, traceMat, minTrace);

//显示最小能量线
cv::Mat tmpImage = image.clone();
for (int i = 0; i < image.rows; i++)
{
    int k = minTrace.at<float>(i, 0);
    tmpImage.at<cv::Vec3b>(i, k)[0] = 0;
    tmpImage.at<cv::Vec3b>(i, k)[1] = 0;
    tmpImage.at<cv::Vec3b>(i, k)[2] = 255;
}
cv::imshow("Seam Carving...", tmpImage);

//删除一列
cv::Mat image2(image.rows, image.cols - 1, image.type());
cv::Mat deletedLine(image.rows, 1, CV_8UC3); //记录被删掉的那一列的值
DeleteOneCol(image, image2, minTrace, deletedLine);
// cv::imshow("Image Show Window", image2);
outImage = image2.clone();
outMinTrace = minTrace.clone();
outDeletedLine = deletedLine.clone();
}

void Resize(const cv::Mat& image, int delta, int kernel) {
    cv::Mat tmpMat = image.clone();
    cv::Mat traces[MAX_COLS];
    cv::Mat deletedLines[MAX_COLS];
    cv::Mat outImage;
    cv::Mat removeMask(image.rows, image.cols, CV_8U, cv::Scalar(0));
    cv::Mat remainMask(image.rows, image.cols, CV_8U, cv::Scalar(0));

    bool shrink = (delta < 0);
    if (shrink) delta = -delta;

    for (int i = 0; i < delta; i++) {
        Shrink(tmpMat, outImage, traces[i], deletedLines[i], removeMask, remainMask, kernel);
        tmpMat = outImage;
        cv::waitKey(50);
    }
}

```

```

cv::Mat tmpMat2 = outImage.clone();
cv::Mat tmpMat3 = outImage.clone(); // 显示Seam
for (int i = delta - 1; i >= 0; i--) {
    ShowSeam(tmpMat3, outImage, traces[i], deletedLines[i]);
    tmpMat3 = outImage;
}
cv::imshow("Seam", tmpMat3);
cv::imwrite("seam.png", tmpMat3);

// 放大图片，这点处理非常巧妙
if (!shrink) {
    for (int i = delta - 1; i >= 0; i--) {
        cv::Mat tracetmp = traces[i].clone();
        for (int j = i + 1; j < delta; j++) {
            for (int k = 0; k < image.rows; k++) {
                if (traces[i].at<float>(k, 0) > traces[j].at<float>(k, 0)) {
                    tracetmp.at<float>(k, 0) += 1;
                }
                else {
                    traces[j].at<float>(k, 0) += 1;
                }
            }
        }
        cv::waitKey(50);
        RecoverOneCol(tmpMat2, outImage, tracetmp, deletedLines[i]);
        tmpMat2 = outImage;
        RecoverOneCol(tmpMat2, outImage, tracetmp, deletedLines[i]);
        tmpMat2 = outImage;
    }
}
cv::imshow("Result", tmpMat2);
cv::imwrite("result.png", tmpMat2);
}

void onMouse(int event, int x, int y, int flags, void *param)
{
    MouseArgs *args = (MouseArgs *)param;
    // 按下鼠标左键拖动时
    if ((event == CV_EVENT_MOUSEMOVE || event == CV_EVENT_LBUTTONDOWN) && (flags &
        CV_EVENT_FLAG_LBUTTON)) {
        int brushRadius = 10; // 笔刷半径
        int rows = args->img.rows, cols = args->img.cols;

        // 以下的双重for循环遍历的是半径为10的圆形区域，实现笔刷效果
        for (int i = std::max(0, y - brushRadius); i < std::min(rows, y + brushRadius); i++) {
            int halfChord = sqrt(pow(brushRadius, 2) - pow(i - y, 2)); // 半弦长
            for (int j = std::max(0, x - halfChord); j < std::min(cols, x + halfChord); j++) {

```

```

        if (args->mask.at<char>(i, j) == 0) {
            // 高亮这一笔
            args->img.at<cv::Vec3b>(i, j) = args->img.at<cv::Vec3b>(i, j) * 0.7 + args->
                color * 0.3;
            // 将这一笔添加到选区
            args->mask.at<char>(i, j) = 1;
        }
    }
}
}

bool IsRemove(const cv::Mat& image, const cv::Mat& removeMask) {
    for (int i = 0; i < image.rows; i++)
        for (int j = 0; j < image.cols; j++)
            if (removeMask.at<char>(i, j) == 1)
                return false;
    return true;
}

void ObjectRemove(const cv::Mat& image, int kernel) {
    cv::Mat showImg = image.clone(); // 拷贝一张图用于显示(因为需要在显示的图上面高亮标注,从而造成修改)
    cv::namedWindow("Remove & Remain", CV_WINDOW_AUTOSIZE); // 新建一个窗口
    cv::Mat removeMask(image.rows, image.cols, CV_8U, cv::Scalar(0)); // 希望获取的待删除选区
    cv::Mat remainMask(image.rows, image.cols, CV_8U, cv::Scalar(0)); // 希望获取的待保留选区

    cv::Vec3b red(0, 0, 255);
    cv::Vec3b green(0, 255, 0);
    MouseArgs *args1 = new MouseArgs(showImg, removeMask, green);
    MouseArgs *args2 = new MouseArgs(showImg, remainMask, red);
    cv::setMouseCallback("Remove & Remain", onMouse, (void*)args1); // 给窗口设置回调函数
    while (1) {
        cv::imshow("Remove & Remain", args1->img);
        // 按 esc 键退出绘图模式,获得选区
        if (cv::waitKey(100) == 27)
            break;
    }
    cv::setMouseCallback("Remove & Remain", onMouse, (void*)args2); // 给窗口设置回调函数
    while (1) {
        cv::imshow("Remove & Remain", args2->img);
        // 按 esc 键退出绘图模式,获得选区
        if (cv::waitKey(100) == 27)
            break;
    }
    cv::setMouseCallback("Remove & Remain", NULL, NULL); // 取消回调函数
    delete args1; args1 = NULL; // 垃圾回收
    delete args2; args2 = NULL; // 垃圾回收
}

```

```

cv::Mat tmpMat = image.clone();
cv::Mat traces[MAX_COLS];
cv::Mat deletedLines[MAX_COLS];
cv::Mat outImage;

int i = 0;
while (!IsRemove(tmpMat, removeMask)) {
    Shrink(tmpMat, outImage, traces[i], deletedLines[i], removeMask, remainMask, kernel);
    for (int j = 0; j < tmpMat.rows; j++) {
        for (int k = traces[i].at<float>(j, 0); k < tmpMat.cols - 1; k++) {
            removeMask.at<char>(j, k) = removeMask.at<char>(j, k + 1);
            remainMask.at<char>(j, k) = remainMask.at<char>(j, k + 1);
        }
    }
    tmpMat = outImage;
    i++;
    cv::waitKey(50);
}
cv::imshow("Result Remove", tmpMat);
cv::imwrite("result_remove.png", tmpMat);
cv::Mat tmpMat3 = outImage.clone(); //显示Seam
for (i--; i >= 0; i--) {
    ShowSeam(tmpMat3, outImage, traces[i], deletedLines[i]);
    tmpMat3 = outImage;
}
cv::imshow("Seam Remove", tmpMat3);
cv::imwrite("seam_remove.png", tmpMat3);
}

int main(int argc, char** argv)
{
    int delta, kernel;
    char filename[100];
    std::cout << "Please input filename." << std::endl;
    std::cin >> filename;
    std::cout << "Please input delta." << std::endl;
    std::cin >> delta;
    std::cout << "Please input kernel." << std::endl;
    std::cin >> kernel;

    cv::Mat image = cv::imread(filename);
    cv::namedWindow("Original Image");
    cv::imshow("Original Image", image);
    cv::waitKey(200);

    Resize(image, delta, kernel);
}

```

```
ObjectRemove(image, kernel);

cv::waitKey(50000);
return 0;
}
```

## 参考文献

- [1] Seam Carving for Content-Aware Image Resizing, *Shai Avidan, Ariel Shamir*, 2007.
- [2] 计算机图形学第二次习题课, 杨晟, 2017.
- [3] 计算机图形学第二次习题课, 胡事民, 2017.