

Seam Carving 实验报告

计 75 班 赵成钢 2017011362

2021 年 1 月

目录

1 代码项目说明	2
1.1 采分点	2
1.2 目录结构	2
1.3 编译运行方式	2
2 代码思路	2
2.1 GUI	2
2.2 缩放函数	3
2.2.1 删除缝隙	3
2.2.2 增加缝隙	3
2.2.3 最佳缝隙查找	4
2.3 加速	5
3 实验结果和分析	5
3.1 缩小	5
3.2 放大	6
3.3 不好的结果	8
4 视频演示	9
5 感想和感谢	9

1 代码项目说明

1.1 采分点

1. 完成了基础算法
2. 通过接缝插入实现了图像扩展
3. 实现了前向能量公式
4. 实现了简单的 GUI (可拖拽缩放)
5. 其他: OpenMP 多线程加速

1.2 目录结构

1. images/originals: 一些示例的原图
2. images/outputs: 一些示例对应的输出
3. videos: 图形界面展示的视频
4. compile.sh: 一键编译脚本
5. CMakeLists.txt: CMake 配置文件
6. main.cpp: 程序入口, 参数处理, 新建窗口
7. scaling.cpp/h: 基于前向能量的缩放实现
8. window.cpp/h: 可智能缩放的窗口类

1.3 编译运行方式

本项目采用 CMake 标准写法构建, 新建 build 文件夹后直接进行 cmake 和 make 命令即可, 或者运行 compile.sh 脚本。

编译完成之后, 以 “seam_carving <image_path> <output_path>” 的格式运行即可, 其中 image 为对应要缩放的输入, 而 output 为最后退出时图片状态的保存路径。

2 代码思路

2.1 GUI

本项目从功能性上出发, 先考虑了图形界面的运作方式, 使用了 Qt 为整体架构, 整个界面只用一个 QLabel 内嵌 QPixmap 显示图片, 同时当 QLabel 被 resize 时, 我们可以通过 override QLabel 的 resizeEvent 来实现对里面的 QPixmap 智能缩放 (调用缩放函数), 大体展示效果如下图。



图 1: GUI 设计

同时，为了防止用已经被缩放之后的图片再次缩放，每次缩放的起点都是用户输入的原图，而不是已经失真的正在展示的图片。

至此，我们只需要实现缩放函数。

2.2 缩放函数

为了简单起见，我没有实现预处理，每次都是重头进行缩放运算；我也没有实现求解最佳增删顺序，做法是简单先增删高度，再增删宽度。那么进入缩放函数之后，只先需要判断高度是变小还是变大，一个缝一个缝地增删即可，之后再判断宽度。

下面，我们以宽度为例（高度同），分别讨论删除和增加。

2.2.1 删除缝隙

删除缝隙和增加缝隙的前提都是找到最佳的缝，具体实现会在后面讲解。在找到缝隙之后，我们直接在图上去掉这条线即可。如此迭代每次去掉一条缝直到宽度符合最后的要求。

2.2.2 增加缝隙

就像论文中所说，如果每次都找到能量最低的缝，然后在缝的左侧进行增加，同时像素取两侧的平均，如此迭代，会出现每次都扩展同一个裂缝的情况，会让结果如下图所示，非常不自然。



图 2: 每次找能量最低的缝进行扩展

那么，我们假设要增加 x 长度的宽度，我们就维护一个减少 x 长度的图片，每次减少 x 长度的图片少了一个缝隙，我们就对应扩展到增加的上面，相当于求解了近似的前 x 低能量的缝隙进行扩展，以至于不会重复扩展一个地方。

2.2.3 最佳缝隙查找

于是，上述的增加或者删除的一个前提是找到最佳缝隙，为了实现这个函数，我们采用文中的 DP 方法，假设 $f(x, y)$ 是以 (x, y) 为结尾的最佳竖直缝隙的能量和（以增删宽度为例），那么我们为了继续向下扩展宽度，只需要从上一行的相邻三个进行转移：

$$f(x, y) = \min \{f(x - 1, y - 1), f(x, y - 1), f(x + 1, y - 1)\} + e(x, y) \quad (1)$$

其中 $e(x, y)$ 为能量。如此转移并记录转移的来源，找到最后一行的最小 f ，之后从这里回溯回去既可以找到能量最小的缝隙。

进一步的，我们使用前向能量公式，考虑作为缝隙去掉之后的代价，而不是缝隙本身的代价，把三种转移的代价重新定义为去掉之后产生新拼接的像素的差异和，可以得到下面三个情况（其中红线对应情况需要统计的像素代价差异）：

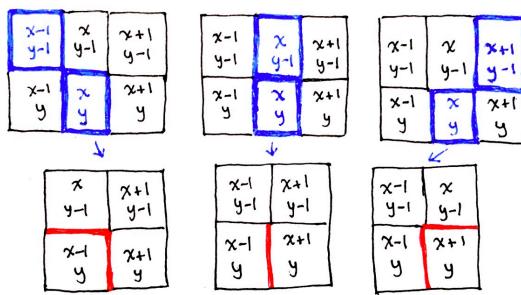


图 3: 前向能量

我们定义 $L = D((x, y - 1), (x - 1, y)) + D((x - 1, y), (x + 1, y))$, $U = D((x - 1, y), (x + 1, y))$, $R = D((x, y - 1), (x + 1, y)) + D((x - 1, y), (x + 1, y))$ 。其中的 D 为对应的像素差异，在我的实

现中是像素灰度的差的绝对值。

如此，我们可以重写 f 转移函数为：

$$f(x, y) = \min \{f(x - 1, y - 1) + L, f(x, y - 1) + U, f(x + 1, y - 1) + R\} \quad (2)$$

如此进行了前向能量的优化，并完成了最佳缝隙的查找，查找一次的复杂度为 $O(wh)$ ，整个算法的整体复杂度为 $O(xwh)$ ，其中 x 是要增加的宽度或者长度。

2.3 加速

由于本实现较为简单，实时缩放相对卡顿，于是我用了 OpenMP 进行加速，我对所有没有循环间依赖的循环前面都加入了编译指导语句来用多线程加速，并有一定的效果（虽然还是比较卡顿）。

3 实验结果和分析

3.1 缩小



图 4: 沙漠



图 5: 画

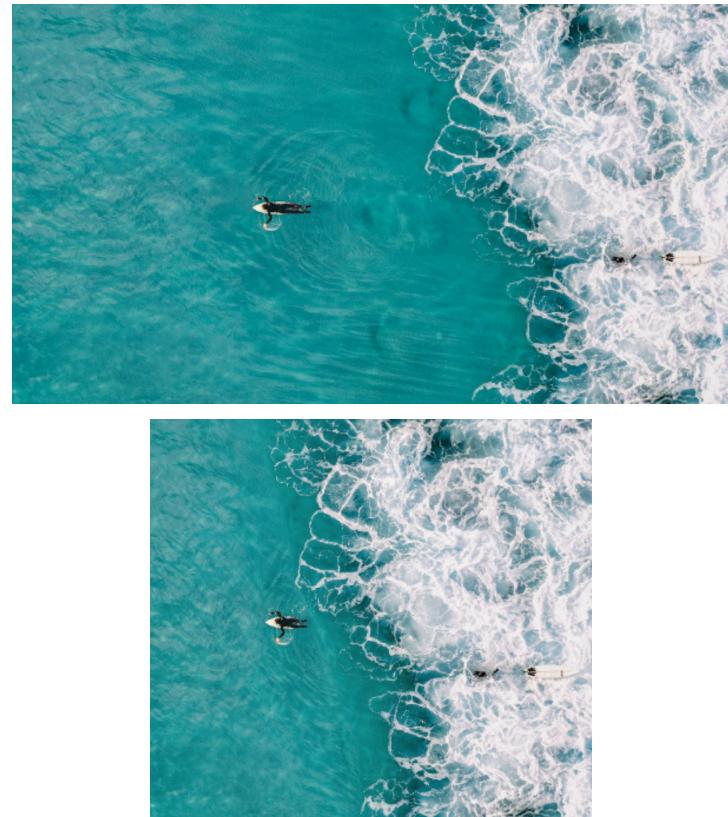


图 6: 海洋

3.2 放大



图 7: 海豚



图 8: 画

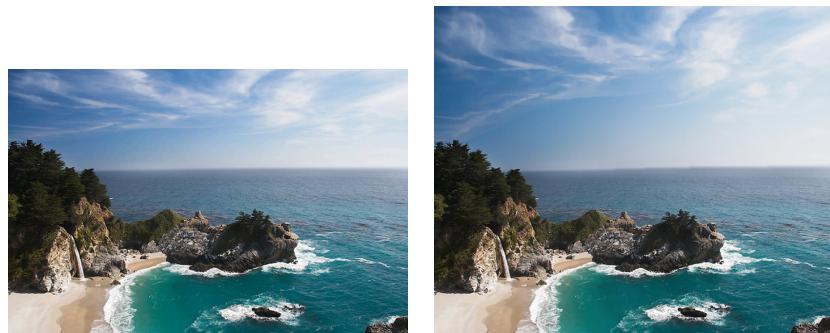


图 9: 海边

3.3 不好的结果

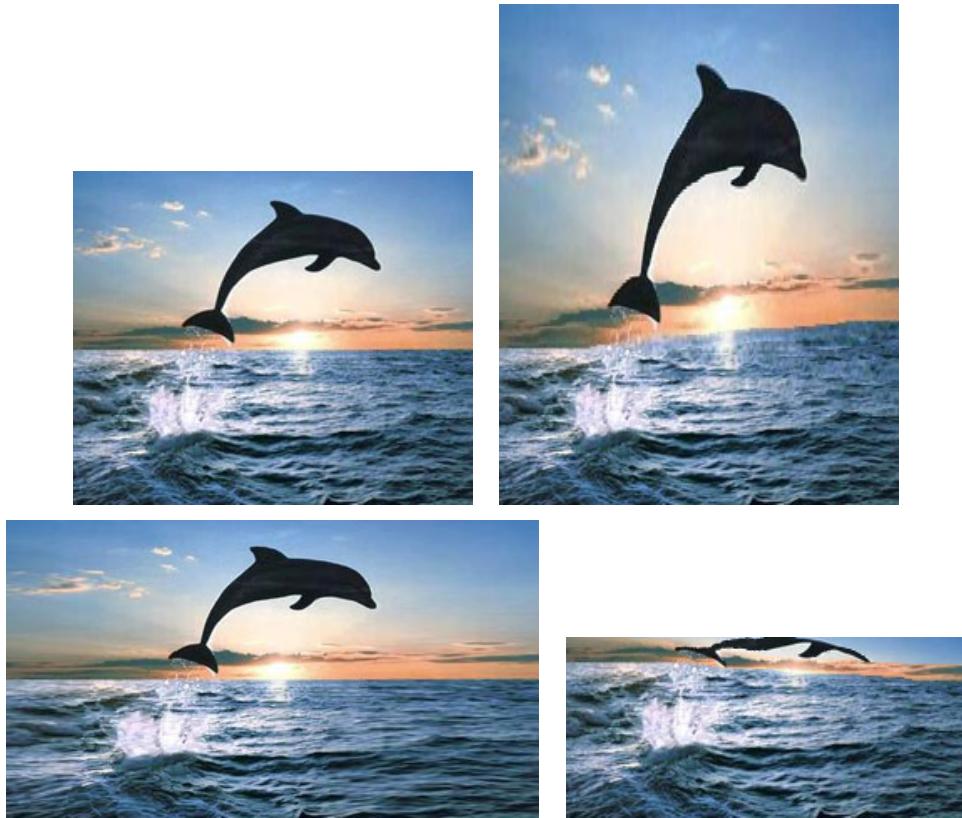


图 10: 海豚

上面以海豚为例给出了几个不好的结果，其中从上到下从左到右排列，第一张是原图，第二章是纵向拉长，第三张是横向拉长，第四张是纵向缩小。可以看到第二张中，大部分选择扩展的裂缝不是本应该重复的海水，而是简单图形的海豚，揪其原因也是因为海面因为非常不规律，虽然视觉上不是重点，但其能量值非常高，超过了只有在边缘处能量高的海豚，这个效应在第四张对偶处理的图里面更明显，可以看到海豚几乎没了。而要解决这个问题，可以参考目标移除等做法，人工标注或通过其他方式识别一些想要的区域，然后标记为高能量的区域。

而对于纵向拉长，因为海水平均分布在图片的下部，所以在图三中会选择天空加海水而不是海豚的部分，效果相对较好，但是毕竟海水是越复杂越有波浪越真实，一味地增加扩大像素，就有一种拉伸的不真实感，这点可以和上面好的效果中的画作为对比，画的大部分都是留白，就可以用这种填充的手法来放大。



图 11: 我

另一个不好的例子同上面海豚的分析，我进一步拿我的照片进行了实验，从上到下从左到右，分别是原图，横向缩小和纵向放大。可以看到第二张本应该缩小裁剪衣服的部分，却因为脸部的能量较低而选择了压缩了脸部，解决这个问题也需要更好的适应特征的能量函数。而第三张，为了横向放大，目前的算法选择了尽量不同的缝隙，而这样反而导致了和正常拉长一样的不好的效应，这时候一直选择前几个低能量的缝隙反复重复（比如衣服左边或者右边的极小的缝）效果肯定会更好，但是我们为了重复不同的地方反而让肩膀变得非常大而且不自然，而为了解决这个问题我们需要更好的填充放大的做法，问题的核心就是选择和重复哪些缝隙。

4 视频演示

为了展示本实验的 GUI 可以运行，我录了几段视频放到了 videos 文件夹中。

5 感想和感谢

本次作业中我认为非常有意思，不同于其他课程的作业，本课程可以真正看到自己在做什么，让我感受到了以媒体为载体和目标的计算的乐趣，也感慨前辈们在目前生活中电影、影像等技术中的贡献。

感谢徐老师的讲解和助教的批改！