《基于线裁剪的内容敏感图像缩放》

实验报告

计 23 鲁逸沁 2012011314

【实验内容】

对于一张图片，使用线裁剪（Seam Carving）的方法，从图片中选择出一些从上到下（或从左到右）的裁剪线，通过移除（或复制）这些裁剪线，对整幅图片的宽度（或高度）进行缩小（或放大）。

通过算法选择最优的裁剪线，使得这些裁剪线的变化，对整幅图片内容的影响最小，实现对内容敏感的特性。

如下面的例子中，中图中红色的部分，就是左图中选择若干条从上到下的裁剪线。这些裁剪线的移除（或复制）对图片的内容影响不大，可以对图片进行横向的收缩（或扩大），右图就是收缩的结果。



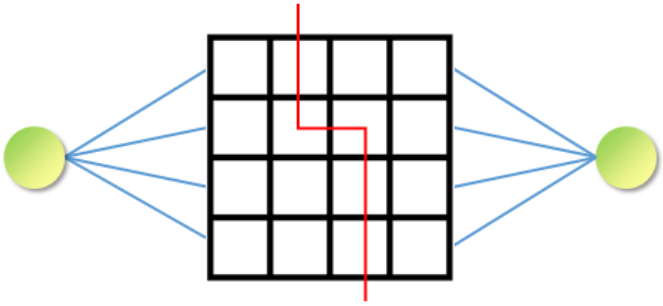
【实验配置】

- 实验语言：C++
- 实验平台：Visual Studio 11
- 实验使用工具：OpenCV-2.4.6.0
- 实验数据：网络图片

【算法描述】

✧ 像素点权重的割算法

算法的核心就是将整幅图建成一个网络流模型，利用类似 Graph Cut 的方法计算出最小割，将最小割中的像素连在一起就是寻找的一条“裁剪线”。



接下来的问题就是如何设计好每个像素的权值，使得最小割的效果最好。

一种方法是计算每个像素的能量，能量的定义为像素点其在横轴和纵轴上的梯度之和，即

$$\text{energy}(x,y)=\left|\frac{\partial I}{\partial x}\right|+\left|\frac{\partial I}{\partial y}\right|$$

其中I为像素值函数，在离散的情况下能量函数就是

$$\text{energy}(x,y)=\frac{1}{2}\left(\left|I_{x+1,y}-I_{x,y}\right|+\left|I_{x-1,y}-I_{x,y}\right|+\left|I_{x,y+1}-I_{x,y}\right|+\left|I_{x,y-1}-I_{x,y}\right|\right)$$

其中$I_{x,y}$为x,y的像素值。

当然，也可以使用其在八连通相邻块的差之和作为能量函数，即

$$\text{energy}(x,y)=\sum_{\substack{-1\le i\le 1\\-1\le j\le 1}}\left|I_{x+i,y+j}-I_{x,y}\right|\cdot w_{i+1,j+1}$$

其中$w=\frac{1}{12}\begin{bmatrix}1&2&1\\2&0&2\\1&2&1\end{bmatrix}$是加权矩阵，描述不同的像素差之间的比例。

也可以采用显著性区域检测里的显著值作为像素的权值

$$S(x,y)=\sum_{\forall I_{i,j}\in I}\left|I_{i,j}-I_{x,y}\right|$$

之后就是采用网络流中经典的最小割算法求解其裁剪线即可。由于平面图最小割算法可以转换成最短路算法求解，因此我在实验中使用的最短路算法。

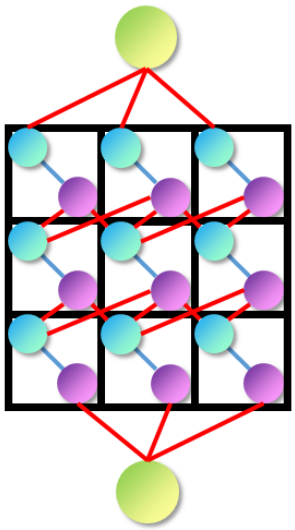
当然，直接求出的最小割并不能直接移除或复制，因为一行或一列中会出现多个像素点。假设我们做的是横向缩放，那一行中只能移除或复制一个像素。而如果一行中在最小割里出现k个像素时，就需要将这k个像素进行融合，按比例缩放成$k-1$（或$k+1$）个像素，使其尽量保持原来的颜色分布。

要缩放一定比例，就需要找到多条裁剪线进行移除或复制。因为该算法每次只能找到一条裁剪线，要进行整体缩放一定比例时就需要使用贪心法每次找最优的裁剪线，但这并不能保证整体是最优的。下面提到的算法会修复这一问题。

✧ 全局最小费用最大流算法

上面提到每次找一条最小割贪心的方法做不到全局最优，那就可以利用网络流考虑全局的良好特性，将最大流模型改成费用流，利用最小费用最大流的算法解决找k条裁剪线的问题。

这里考虑将每个像素设为一个点，将像素之间的距离设为边权，来构建最小费用最大流模型。以横向缩放为例，为了保证裁剪线连续，让每个像素只和其下面的相邻三个像素相连。由于考虑到一个像素点只能被移除或复制一次，因此还需要把每个像素点拆点，中间加一条流量为1的边，保证其只在一条裁剪线里。



每个像素点和它下面连通的三个像素点连边，边费用是之前定义的像素差。这样，求出这个网络的最小费用最大流，限制最大流的大小，就能求出任意个数条裁剪线，这些裁剪线保证了每个像素点至多被一条裁剪线覆盖，并且所有裁剪线的费用之和是全局最小的。

虽然网络流算法效果很好，但是它有一个致命的缺陷——速度极其慢。因此可能我们需要牺牲一点效果，寻求更快的算法。

✧ 动态规划简化算法

效率慢的原因归根结底是因为我们考虑了全局最优。如果还按照第一种方法一样，每次选取最优的一条道路，迭代若干次获得效果，就可以大大提升效率。

在最小费用最大流的建图中，每个像素之和它上一行相邻的三个像素可能连线，状态分层性明显，因此可以使用动态规划方法求一条裁剪线。转移方程如下

$$f(x,y)=\min_{-1\le i\le 1}\{f[x-1,y+i]+|I_{x,y}-I_{x-1,y+i}|\}$$

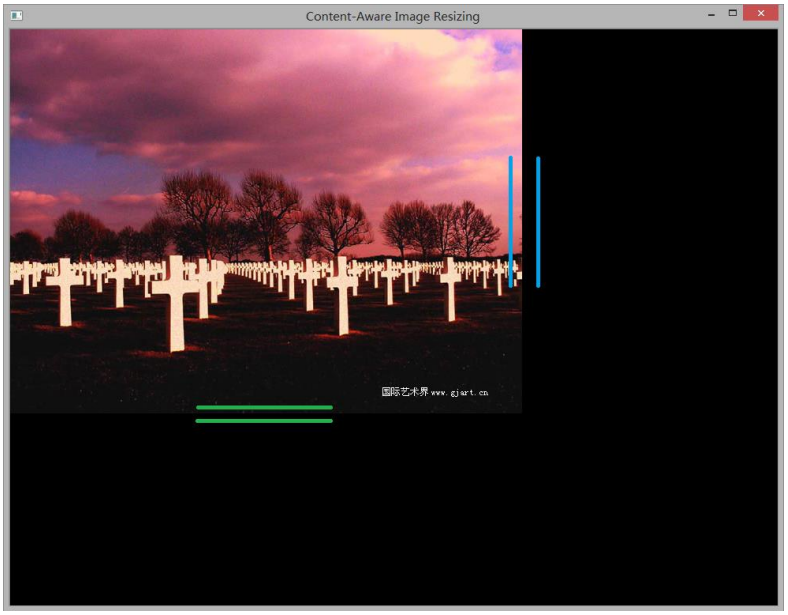
这样，从第一行求到最后一行，就能求出最优的线裁剪方案。这样的算法复杂度是和像素点个数同阶的，做到了理论单次线裁剪的最小复杂度。

但是有一个问题，在做图片放大的时候，如果每次都用同样的方法选取一条最优裁剪线，并将其复制，那接下来每次寻找的最优裁剪线必定和第一次相同。这样经过多次复制后，一样的复制内容就会连续的叠加在一起，造成失真，影响整体效果。如下图所示

为了避免这样的问题，进行放大操作时，需要在每次线裁剪后将最优裁剪线及其复制的裁剪线上的像素标记为不可达，在下次动态规划时直接忽略这些像素，这样就能保证每次选取的路径没有重合部分，也就不影响整体效果了。

虽然动态规划方法没有做到全局最优，不过最后实现的结果也相当不错，程序运行速度也属于可容忍范围。

【Demo 说明】



Demo 在一开始会弹出一个选择文件对话框，要求你导入一张图片。待图片导入后会出现如上图所示的操作画面。

如果图片是$W \times H$大小的话，整个窗口会是$1.5W \times 1.5H$大小的，这样用户可以在宽度$0 \sim 1.5W$，高度$0 \sim 1.5H$的范围里随意调整。

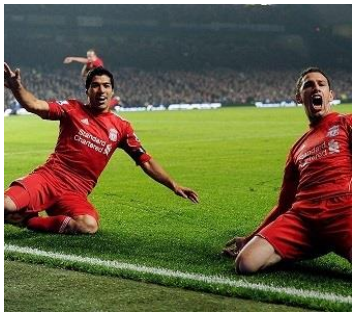
图片的边界是可以鼠标操作的部分。如果拖动图片的右边界（蓝色线条中间部分），图片就能将宽度缩放至用户指定的范围。同理拖动图片的下边界（绿色线条中间部分），图片就能将高度缩放至用户指定的范围。

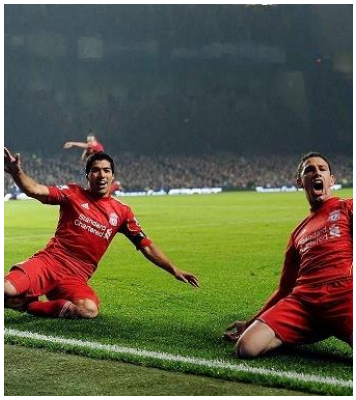
【实验结果】













【算法比较】

✧ 三种算法的效果比较

原图

像素点权重的割算法

全局最小费用最大流算法

动态规划简化算法

✧ 三种算法性能比较

缩放处理：将宽度减半

时间单位：毫秒

	像素点权重的割算法	全局最小费用最大流算法	动态规划简化算法
400×300	4702	233587	4168
620×428	12019	162393	12784
768×480	20657	834356	21158

【实验总结】

通过本次实验，我了解了内容敏感图像缩放的原理，学习了基于 Seam Carving 的内容敏感图像缩放，了解它的可行性和算法流程。

本次实验中，我实现了两种不同的 Seam Carving 类型的方法，一种是基于像素点权值的割算法，一种是基于像素点之间差值的割算法。其中第二种算法可以做到全局最优，但是效率上比较差，因此我通过分析和推测优化了第二种算法，提出了非全局最优但是效率不错的第三种算法。事后查资料得知第三种算法确实也有人尝试过。

我比较了三个算法的实验效果，尝试了很多不同的图片，发现算法效果几乎一样，也就是说权衡效率和效果，第三种算法是明显比第二种算法好的。

这些算法在一些内容比较突出的图片中表现不错，但是在一些内容并不是特别突出的图片中表现一般，可能这就是 Seam Carving 原理本身的缺陷，是无论如何优化算法都无法逾越的困难。

同时通过本次实验，我也掌握了基本的 OpenCV 用法，并写了近 20K 的代码，锻炼了自己的代码能力，受益匪浅！

【参考文献】

[1] Zargham P, Nassirpour S. Content-Aware Image Resizing[J].

[2] Avidan S, Shamir A. Seam carving for content-aware image resizing[C]//ACM Transactions on graphics (TOG). ACM, 2007, 26(3): 10.

[3] 雷励星. 基于混合能量的内容敏感图像缩放新方法[J]. 计算机学报, 2010, 33(10): 2016-2021.