图像缩放之接缝裁剪算法

陈鸿绪　35　PB21000224

2024 年 3 月 17 日

摘要

接缝裁剪算法，通过滤波器计算图像每一点像素的强度，应用动态规划算法找到图中数条强度较低的接缝，通过删除或增加所选择的接缝，实现图像的缩减或者拉伸。在本次实验中，我采用了传统的接缝裁剪算法实现图像缩减，取得良好的实验结果。实验过程中发现单纯运用接缝裁剪算法进行拉伸效果欠佳，图像锯齿化效应非常明显，为了得到更好的实验结果，我采用以下做法：先将原图像等比例扩大至宽度与拉伸后的宽度相同，再旋转 90° 通过接缝裁剪将图像横向缩减，最后旋转至原位。该做法得到良好的实验结果。同时，我也尝试不用以上技巧的情况下，探索改进接缝裁剪算法进行图像拉伸：在局部范围中均匀随机选择强度极小值路径，复制上述路径，最后得到的效果比朴素接缝裁剪算法锯齿化效应减弱，但是效果仍然不比以上技巧。

一、前言（问题的提出）

1.1 问题背景

在图像处理和计算机视觉领域，经常需要对图像进行缩放以适应不同的显示设备和屏幕尺寸。然而，传统的图像缩放方法往往会导致图像内容的扭曲或失真，特别是在缩小图像时，重要的细节可能会被丢失或变形。

1.2 传统图像缩放算法

在接缝裁剪算法（Seam Carving）之前，已经存在多种图像缩放的算法。这些算法主要基于采样和插值的方法，用于改变图像的尺寸以适应不同的显示需求。

最近邻插值（Nearest-neighbor interpolation）：这是一种最简单的插值方法。它选择离待求点最近的像素值作为该点的值。这种方法速度快，但可能导致图像出现锯齿状边缘和马赛克效应。

双线性插值（Bilinear interpolation）：这种方法考虑了待求点周围四个像素点的值，通过在这四个点之间进行线性插值来得到待求点的值。双线性插值比最近邻插值更平滑，但可能仍然会在某些区域产生模糊。

双三次插值（Bicubic interpolation）：这种方法考虑了待求点周围 16 个像素点的值，并使用三次多项式进行插值。双三次插值通常能得到比双线性插值更平滑的结果，但计算复杂度也更高。

尽管这些传统的图像缩放算法在一定程度上能够满足基本的缩放需求，但它们都存在着一些问题。特别是在缩小图像时，这些方法可能会导致图像内容的扭曲和失真，因为它们在缩放过程中没有考虑到图像内容的重要性。

二、相关工作

在*"Seam carving for content-aware image resizing"* [1] 文章中，Shai Avidan 和 Ariel Shamir 提出了接缝裁剪算法。该算法的核心思想是在保持图像内容完整性的前提下，通过删除或增加图像中的某些像素来实现图像的缩放。这些被删除或增加的像素被称为“接缝”，它们的选择是基于像素的强度或能量值，确保在删除或增加接缝后，图像的视觉内容不会被显著影响。

三、问题分析

3.1 分析一

在本次实验中，目标是利用复现传统接缝裁剪算法进行图像缩放。注意到横向缩放和纵向缩放只需要将图像旋转 90° 即等价，所以本次实验只进行横向缩放，加上或减去某个值代表横向拉伸或缩减该值。

3.2 分析二

注意到等比例扩大图像可以使用 matlab 中基于插值算法的 imresize 函数。利用该函数，先将原图像等比例扩大至宽度与拉伸后的宽度相同，再旋转 90° 通过接缝裁剪将图像横向缩减，最后旋转至原位即可利用缩减算法完成图像拉伸。

四、建模的假设

4.1 假设 1

图像不同位置的“能量”不同：这里的“能量”可以理解为像素的重要性或信息量。在采用接缝裁剪的算法前，我们需要假设重要内容所在的区域能量大，而相对不那么重要的区域能量小。

4.2 假设 2

图像内容的连续性：算法假设图像中的内容是连续且相对一致的，这样在寻找接缝时，能够基于相邻像素的相似性进行决策。如果图像内容过于复杂或断裂，可能会影响到接缝的准确寻找。

4.3 假设 3

能量函数的合理性：算法使用能量函数来评估像素的重要性。这个能量函数通常是基于像素的亮度、颜色、纹理等特征来定义的。算法假设这个能量函数能够合理地反映像素的重要性，从而能够指导接缝的寻找。

五、符号说明

表 1: 符号说明

符号	说明	单位
I	输入图像的三通道矩阵表示，其中 I(p) 表示像素 p 的颜色值	-
V	滤波器对应的卷积矩阵，V(p) 用于对像素 p 进行卷积	-
E	能量矩阵，E(p) 用于衡量像素 p 的重要性或信息量	-
dir	指示方向矩阵，其中 $M(i,j)$ 为该点像素所在接缝的上个像素相对位置	-
M	累积能量矩阵，尺寸与原图相同，其中 $M(i,j)$ 为该点像素的累计能量	-
w,h	分别为图像的宽度和高度	-
k	需要裁剪的行数或列数，具体取决于裁剪的方向实验中采取横向	-

六、数学模型建立

6.1 能量函数的定义

实验中我们分别使用了两种不同的滤波器，即对应两种不同的卷积矩阵：

$$V = \begin{pmatrix} 0.5 & 1 & 0.5 \\ 1 & -6 & 1 \\ 0.5 & 1 & 0.5 \end{pmatrix}, \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$

通过对应卷积矩阵 V 的滤波器 (imfilter) 作用在图像三通道矩阵 I 上计算能量：

$$\mathrm{E} = \underset{3channel}{\mathrm{sum}}((\mathrm{imfilter}(\mathrm{I}, V)^2), 3) \tag{1}$$

得到能量矩阵 E。

6.2 累计能量初始化

为了下面动态规划可以进行，我们需要额外定义一行像素的累计能量用来初始化：

$$M(0, i) = 0 \tag{2}$$

其中 $0 \leq i \leq w$。

6.3 动态规划计算累计能量

我们需要得到一条累计能量最小的接缝，为了找到该接缝，需要利用动态规划算法计算出累计能量矩阵 M。动态规划方程为：

$$M(i, j) = E(i, j) + \min\left(M(i-1, j-1), M(i-1, j), M(i-1, j+1)\right), \tag{3}$$

其中 $0 \leq i \leq w$, $0 \leq j \leq h$。同时记录每一个像素点所在接缝连接上一个的像素点的相对位置，左上角、上方、右上角分别记为-1、0、1，得到指示方向矩阵 dir。

6.4 接缝寻找与删除

得到累计能量矩阵之后，最后一行即为所有可能能量最小接缝的累计能量值，记需要删除的像素点在该层的索引为 delete_index。我们需要寻找最小的一条接缝，首先从第一行开始：

$$\mathrm{delete\_index} = \underset{1 \leq i \leq w}{\arg\min} M(i, h) \tag{4}$$

其次，回溯寻找所在接缝的所有像素点位置：

$$\mathrm{delete\_index} = \mathrm{delete\_index} + dir(textdelete\_index) \tag{5}$$

每一次回溯寻找的同时将该层索引为delete_index的像素点删除。

6.5 循环 k 次操作

由于需要删除的接缝总共是 k 个，只需要重复上述步骤 k 次即可完成图像横向缩小 k 个像素点。

6.6 变体：图像横向拉伸

采取三种不同的方法进行图像横向拉伸。第一种采用原始接缝裁剪算法直接将接缝复制加入进图像，但效果欠佳所以不予展示; 第二种是基于接缝裁剪的优化，回溯寻找第一层 delete_index 的时候不再全局寻找累计能量最小，而是随机某个固定范围内寻找极小值的 delete_index。即第一层：

$$\text{delete\_index} = \underset{i\in[randi-C,randi+C]}{\arg\min} M(i,h) \tag{6}$$

其中 C 为正整数，$randi$ 是均匀随机在 $[1+C, w-C]$ 的整数; 第三种即采用之前所述的 trick，利用插值函数等比例放大，再通过旋转操作和上述图像缩减的接缝裁剪算法得到图像拉伸结果。

七、结果（与对比）

下面展示相对优的算法结果，对于算法欠佳的结果将在结果对比中展示。

7.1 缩减结果展示

图 1: peppers.png 横向缩小 100 像素，卷积核二

图 2: 自选图像横向缩小 200 像素，卷积核二

7.2 拉伸结果展示

图 3: peppers.png 横向拉伸 100 像素，卷积核二

图 4: 自选图像横向拉伸 200 像素，卷积核二

7.3 对比结果展示

图 5: peppers.png 横向拉伸 100 像素，左侧基于接缝裁剪优化算法，右侧基于等比例放大旋转 trick

八、结论

8.1 结论一

接缝裁剪算法能够地识别图像中的低能量区域，这些区域通常对应着图像中的背景或者非关键细节。通过在这些区域进行裁剪，算法能够在缩减图像时最大程度地保留关键内容，从而避免扭曲或变形。从上图缩减结果我们可以看出所有图片中的主体内容都得以良好保留，peppers 的右侧红色辣椒还有自选图像中的房子都得以完好保留。

8.2 结论二

接缝裁剪算法在缩放图像时对图像中物体边缘识别敏感，这主要取决于卷积核的形式，对于实验中所选取的卷积核，我们发现删除的能量较小的接缝位置总是集中在物体边缘，比如在 peppers 横向缩减 100 像素的时候，图像结果的左侧，即图中物体与暗色背景的交接处，锯齿化严重，且损失信息较多。实验中同时也发现两个卷积核差别不是很明显，所以这里全部使用卷积核二得到的图像结果。

8.3 结论三

在本次实验中，朴素接缝裁剪算法和经过优化的接缝裁剪算法拉伸效果均欠佳，这主要因为在往图像中添加能量较低的接缝时，下一次寻找能量较低的接缝很容易会陷入之前加入过的接缝附近。从 7.3 节左图可以观察出，拉伸的接缝明显集中在某些特定区域。如果采用朴素算法会陷入特定一个区域，经过随机优化的算法接缝不会陷入特定一个区域，而是会有多个。然而基于接缝裁剪缩减的、等比例放大旋转 trick 的算法（7.3 节右图）效果是最好的。

九、问题

笔者认为对于本次实验构建的算法主要的问题是算法时间代价较大，而且随着图像像素的增多运行时间也会相应增大，所以我认为在进行动态规划的时候可以引入堆优化，累计能量数组可以用堆进行存储，在每次更新的时候只需要进行堆维护，寻找并取出最小能量的索引只需要 O(1) 的时间。

参考文献

[1] S. Avidan and A. Shamir, “Seam carving for content-aware image resizing,” in *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pp. 609–617, 2023.