



南開大學
Nankai University

南开大学
计算机学院
算法导论期末实验报告

图片“内容感知”缩放 (Seam Carving)

https://github.com/iChubai/Seam_Carving

侯嘉栋
年级：2023 级
专业：计算机科学与技术
指导教师：李翔

2025 年 6 月 2 日

摘要

Seam Carving 是一种内容感知的图像缩放技术，能够根据图像的视觉重要性来选择性地缩小或放大图像，避免传统缩放方法中常见的对象变形或背景丢失的问题。本报告介绍了该算法的实现方法及其优化，特别是基于能量图的垂直缝隙搜索算法。通过动态规划寻找最小能量路径，并利用能量图来指导像素删除或插入，从而在保证图像主要内容不受损的情况下调整图像大小。实验结果显示，Seam Carving 在内容保留方面具有显著优势，尤其在图像背景较为简单的情况下，但在主体占据图像较大区域时可能会导致图像变形，特别是在人物或建筑物等主题较为显著的图像中。

关键字：Parallel 图像缩放 Seam Carving 内容感知 动态规划

目录

一、 概述	1
(一) 第一节: 问题描述	1
(二) 第二节: 算法设计与分析	1
1. 算法核心	1
2. 具体任务	2
(三) 第三节: 源代码	3
1. 能量场的计算	3
2. 垂直 Seam 搜索算法	4
3. 移除指定的垂直 seam	5
4. 基于能量图的搜索算法优化	6
(四) 第四节: 效果展示	7
二、 总结	9

一、概述

(一) 第一节: 问题描述

图像缩放是一个常见的图像处理任务，但传统方法如均匀缩放或裁剪往往会导致重要内容的失真或丢失。例如，均匀缩放可能使图像中的对象变形，而裁剪可能直接移除关键区域。Seam carving（接缝雕刻）是一种内容感知的图像缩放技术，旨在通过移除或添加视觉重要性最低的像素路径（称为“接缝”）来调整图像大小，同时最大程度保留重要内容。接缝是从图像顶部到底部（或左侧到右侧）的 8 连通像素路径，其重要性由能量函数定义，通常基于梯度幅值或其他视觉显著性度量。Seam carving 由 Shai Avidan 和 Ariel Shamir 于 2007 年提出 (Seam Carving Paper)，已被广泛应用于图像编辑软件（如 Photoshop 的内容感知缩放和 GIMP 的 Liquid Rescale）。该技术适用于调整宽高比、目标大小调整、对象移除等场景，特别适合将图像适配到不同显示设备（如手机或投影屏幕）或动态网页布局。

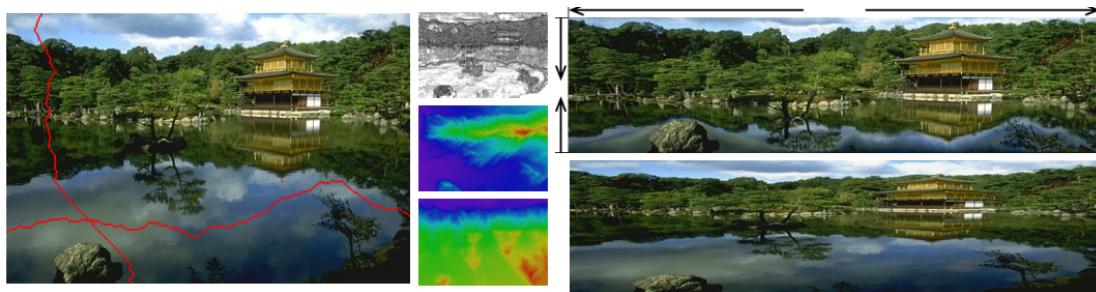


Figure 1: A seam is a connected path of low energy pixels in an image. On the left is the original image with one horizontal and one vertical seam. In the middle the energy function used in this example is shown (the magnitude of the gradient), along with the vertical and horizontal path maps used to calculate the seams. By automatically carving out seams to reduce image size, and inserting seams to extend it, we achieve *content-aware resizing*. The example on the top right shows our result of extending in one dimension and reducing in the other, compared to standard scaling on the bottom right.

图 1: 一条 seam（接缝）指的是在图像中由低能量像素构成的连通路径。左侧示意图展示了原始图像，并在其中标出了一个水平 seam 与一个垂直 seam。中间给出的是本例所用的能量函数（即梯度幅值），以及依据该能量函数计算得出的垂直与水平路径图。通过自动移除 seam 来缩小图像尺寸，或插入 seam 来放大图像，就能实现“内容感知”的图像大小调整。右上角的示例展示了在一个方向上扩展、另一个方向上收缩后的效果；右下角则是传统比例缩放的对比结果。

(二) 第二节: 算法设计与分析

1. 算法核心

Seam carving 算法的核心是通过动态规划和能量函数识别并操作接缝。

对内容感知尺寸调整的思路，是有选择地删减像素。因此，首要问题就是：该删掉哪些像素？直观上，我们的目标是移除那些“不起眼”、与周围环境自然融合的像素。：

$$e_1(I) = \left| \frac{\partial I}{\partial x} \right| + \left| \frac{\partial I}{\partial y} \right|. \quad (1)$$

给定能量函数，假设我们需要缩减图像的宽度，可以想出若干策略。例如，为了最大限度保留能量（即保留高能量像素），一种“最优”策略是按能量从低到高依次删除像素。但这样会破坏图像的矩形形状，因为每一行可能被删除的像素数量不同。如果想保持矩形形状，可以在每一行删除相同数量的低能量像素，却会造成锯齿状的视觉伪影。

为了同时保持形状和视觉连贯性，可以采用自动裁剪（Auto-cropping）：在原图中寻找一个与目标尺寸相同的子窗口，使其包含的能量最大。另一种介于删像素和裁剪之间的方法是删除整列能量最低的列，但所得图像仍可能出现明显的瑕疵。因此，我们需要一种比裁剪或整列删除更灵活、又比逐像素删除更能保护内容的尺寸调整算子，这就引出了 Seam Carving 及“内部 seam”的概念。

设图像 I 的尺寸为 $n \times m$ 。垂直 Seam 定义为

$$s_x = \{s_x^i\}_{i=1}^n = \{(x(i), i)\}_{i=1}^n, \quad |x(i) - x(i-1)| \leq 1, \quad (2)$$

其中映射 $x : [1, n] \rightarrow [1, m]$ 。即 seam 是一条自上而下、每行恰含一个像素的 8-连通路径。
水平 Seam 定义为：

$$s_y = \{s_y^j\}_{j=1}^m = \{(j, y(j))\}_{j=1}^m, \quad |y(j) - y(j-1)| \leq 1, \quad (3)$$

其中 $y : [1, m] \rightarrow [1, n]$ 。

设 seam s 的像素集合为 $I_s = \{I(s_i)\}_{i=1}^n$ （例如垂直 seam 时 $s_i = (x(i), i)$ ）。类似删除整行或整列，删除 seam 只会产生局部影响：沿 seam 的像素被移除后，其右侧（或下侧）像素向左（或向上）平移以填补空缺，因此视觉影响仅沿 seam 路径出现，图像其余部分保持不变。若将约束 $|x(i) - x(i-1)| \leq 1$ 放宽为 $|x(i) - x(i-1)| \leq k$ ，则当 $k = 0$ 时退化为整列删除；当 $1 \leq k \leq m$ 时，则得到分段连通甚至完全离散的像素集合。

实现合理的 seam 删除，我们需要搜索最优的 seam。给定能量函数 e ，定义一条 seam $s = \{s_i\}_{i=1}^n$ 的代价为

$$E(s) = \sum_{i=1}^n e(I(s_i)).$$

我们希望找到代价最小的 seam

$$s^* = \arg \min_s E(s) = \arg \min_s \sum_{i=1}^n e(I(s_i)). \quad (4)$$

这个过程需要用到动态规划：

1. 第一步，自第二行起遍历至最后一行，递推累计最小能量

$$M(i, j) = e(i, j) + \min(M(i-1, j-1), M(i-1, j), M(i-1, j+1)).$$

遍历完成后， M 最后一行的最小值给出了最短垂直 seam 的终点。

2. 第二步，从该最小值开始反向回溯，即可得到整条最优 seam

水平方向的 M 递推完全类似，只需对图像做一次转置即可

2. 具体任务

对于长宽比缩放，设图像 I 尺寸为 $n \times m$ ，目标尺寸为 $n \times m_0$ ，其中 $m - m_0 = c$ 。只需连续删除 c 条垂直 seam 即可完成宽度缩减。与简单缩放不同，此操作不会改变高能量区域，从而实现非均匀、内容感知的缩放。若要拉宽图像，也可按比例增加行数到 $n_0 \times m$ ，这种方法不丢失任何信息。

更一般地，将图像 I 从 $n \times m$ 重新定向到 $n_0 \times m_0$ （设 $m_0 < m$, $n_0 < n$ ）时，需要决定删除 seam 的先后次序：先垂直？先水平？还是交替进行？我们把最优顺序表述为最小化下列目标函数：

$$\min_{s_x, s_y, \alpha_k} \sum_{i=1}^k E(\alpha_i s_x^i + (1 - \alpha_i) s_y^i), \quad (4)$$

其中 $k = r + c$, $r = m - m_0$, $c = n - n_0$, $\alpha_i \in \{0, 1\}$ 指示第 i 步删除垂直 ($\alpha_i = 1$) 或水平 ($\alpha_i = 0$) seam。

利用 **传输图** T 可用动态规划求解。条目 $T(r, c)$ 记录得到尺寸 $(n - r) \times (m - c)$ 图像的最小总代价:

$$T(r, c) = \min \left\{ T(r-1, c) + E(s_x(I_{(n-r-1) \times (m-c)})), T(r, c-1) + E(s_y(I_{(n-r) \times (m-c-1)})) \right\}. \quad (5)$$

将连续删除 seam 视为“时间演化”过程, 记 $I^{(t)}$ 为删除 t 条 seam 后的图像。若要放大, 可“逆向”插入人工 seam:

1. 在当前图像 I 上找最优垂直 (或水平) seam s
2. 复制 s 两侧像素并取平均, 插入到 s 位置, 得到 $I^{(-1)}$

若保持尺寸不变而增强主体, 可先放大再裁剪: 先用常规缩放整体放大, 再用 seam carving 将图像裁回原尺寸。被删除的像素实际上是原图的“子像素”, 因而主体得到放大。

(三) 第三节: 源代码

在实现上我基于 Opencv 写了一个 SeamCarver 类, 成员包括 `current_image`, `energy_map`, 分别为现在的图像和其能量图, 除了一些可视化函数之外, 核心函数有 (由于水平和竖直的对称性, 以下分析基于竖直情况)

- `calculate_energy_map` 用于计算能量图
- `find_vertical_seam` 用于找到一条能量最低的垂直 seam
- `find_multiple_vertical_seams` 用于找到多条能量最低的垂直 seam
- `remove_vertical_seam` 用于移除指定的垂直 seam

1. 能量场的计算

Algorithm 1 计算能量图

Input: 当前图像 `current_image`
Output: 能量图 `energy_map`

```

1: if current_image 为空 then
2:   return
3: end if
4: 将 current_image 转为灰度图 gray_image
5:  $grad_x \leftarrow \text{Sobel}(\text{gray\_image}, dx = 1, dy = 0)$ 
6:  $grad_y \leftarrow \text{Sobel}(\text{gray\_image}, dx = 0, dy = 1)$ 
7:  $abs\_gx \leftarrow |grad_x|$ 
8:  $abs\_gy \leftarrow |grad_y|$ 
9:  $energy\_map \leftarrow 0.5 \times abs\_gx + 0.5 \times abs\_gy$ 

```

其中 Sobel 算子通过对灰度图像 I 分别应用

$$G_x = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix}, \quad G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{pmatrix}$$

两个 3×3 卷积核，得到 $\text{grad}_x = G_x * I$ 和 $\text{grad}_y = G_y * I$ ，其梯度幅值

$$E(x, y) = \sqrt{\text{grad}_x(x, y)^2 + \text{grad}_y(x, y)^2}$$

或近似为

$$E(x, y) \approx |\text{grad}_x(x, y)| + |\text{grad}_y(x, y)|.$$

Sobel 卷积核在计算一阶导数的同时对邻域像素加权平滑，既能有效抑制噪声，又能高效提取水平和垂直边缘，且计算量小，易于硬件或 SIMD 并行加速。

2. 垂直 Seam 搜索算法

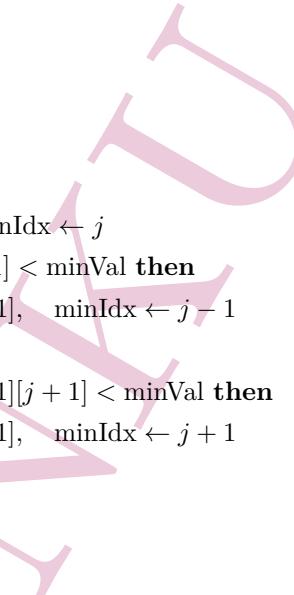
Algorithm 2 垂直 Seam 搜索（动态规划）

Input: 能量图 $E[0:rows - 1][0:cols - 1]$
Output: 垂直 seam 路径 $seam[0:rows - 1]$

```

1: 初始化矩阵  $M \leftarrow +\infty$ ,  $P \leftarrow 0$ 
2: for  $j = 0$  to  $cols - 1$  do
3:    $M[0][j] \leftarrow E[0][j]$ 
4: end for
5: for  $i = 1$  to  $rows - 1$  do
6:   for  $j = 0$  to  $cols - 1$  do
7:      $minVal \leftarrow M[i - 1][j]$ ,  $minIdx \leftarrow j$ 
8:     if  $j > 0$  and  $M[i - 1][j - 1] < minVal$  then
9:        $minVal \leftarrow M[i - 1][j - 1]$ ,  $minIdx \leftarrow j - 1$ 
10:    end if
11:    if  $j < cols - 1$  and  $M[i - 1][j + 1] < minVal$  then
12:       $minVal \leftarrow M[i - 1][j + 1]$ ,  $minIdx \leftarrow j + 1$ 
13:    end if
14:     $M[i][j] \leftarrow E[i][j] + minVal$ 
15:     $P[i][j] \leftarrow minIdx$ 
16:  end for
17: end for
18:  $bestIdx \leftarrow \arg \min_{0 \leq j < cols} M[rows - 1][j]$ 
19:  $seam[rows - 1] \leftarrow bestIdx$ 
20: for  $i = rows - 2$  down to 0 do
21:    $seam[i] \leftarrow P[i + 1][seam[i + 1]]$ 
22: end for
23: return  $seam$ 

```



基于动态规划思想，在已计算好的能量图 E 上寻找从图像顶端到底端、能量总和最小的像素路径。首先，令矩阵

$$M[i, j] = \begin{cases} E(0, j), & i = 0, \\ \infty, & i > 0 \end{cases}$$

表示通过位置 (i, j) 的最小累计能量；同时维护回溯矩阵 $P[i, j]$ ，记录达到 (i, j) 的最优前驱索引。随后，自第 2 行开始，对每个像素 (i, j) 考察其上方三个候选位置 $(i - 1, j - 1), (i - 1, j), (i -$

$1, j + 1)$ 的累计能量，取最小者加上当前能量更新 $M[i, j]$ ，并在 $P[i, j]$ 中记录该最小前驱。这样，矩阵 M 在第 i 行填充结束后即包含了以该行每个像素为终点的最小路径能量。

当最后一行计算完毕后，遍历第 $rows - 1$ 行，找到能量最小的列索引 j^* 。从该位置开始，利用回溯矩阵 P 自底向上依次查找前驱，依次得到从第 0 行到第 $rows - 1$ 行的最小能量路径，即所求垂直 seam。该算法时间复杂度为 $O(rows \times cols)$ ，空间复杂度同样为 $O(rows \times cols)$ 。

3. 移除指定的垂直 seam

在已获得垂直 seam 路径后，本算法在原图像上逐行删除对应列的像素，并将其余像素向左平移以填补空洞。首先，对输入进行合法性检查：保证图像与 seam 非空且行数匹配，并且图像宽度大于 1。随后，构造一个宽度减一的新图像 new_image，大小为 $rows \times (old_cols - 1)$ 。对于每一行 i ，若列索引 j 在 seam 之前，则直接复制原像素；否则复制原像素右侧一列的值。完成所有行的像素重映射后，用 new_image 替换 current_image，并调用 calculate_energy_map() 重新计算能量图。该算法时间复杂度为 $O(rows \times cols)$ ，空间复杂度同样为 $O(rows \times cols)$ 。

Algorithm 3 移除垂直 Seam

Input: 原图像 $current_image$, 垂直 seam 列索引数组 $seam[0:rows - 1]$

Output: 更新后的图像 $current_image$

```

1: if  $current\_image$  为空 or  $seam$  为空 or  $rows \neq |seam|$  then
2:   输出错误并 return
3: end if
4: if  $cols \leq 1$  then
5:   输出错误并 return
6: end if
7:  $new\_cols \leftarrow cols - 1$ 
8: 构造空图像  $new\_image[rows][new\_cols]$ 
9: for  $i = 0$  to  $rows - 1$  do
10:    $sc \leftarrow seam[i]$ 
11:   for  $j = 0$  to  $new\_cols - 1$  do
12:     if  $j < sc$  then
13:        $new\_image[i][j] \leftarrow current\_image[i][j]$ 
14:     else
15:        $new\_image[i][j] \leftarrow current\_image[i][j + 1]$ 
16:     end if
17:   end for
18: end for
19:  $current\_image \leftarrow new\_image$ 
20: 调用  $calculate\_energy\_map()$  重新计算能量图
21: return

```

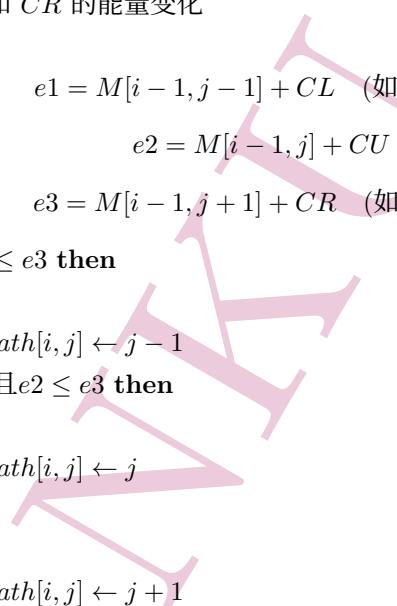
4. 基于能量图的搜索算法优化

Algorithm 4 寻找垂直缝隙 (Forward)

Input: 当前图像 $current_image$, 能量图 $energy_map$

Output: 垂直缝隙的列索引数组 $seam[0:rows - 1]$

```

1: if  $current\_image$  为空 or  $energy\_map$  为空 then
2:   输出错误并 return
3: end if
4:  $rows \leftarrow energy\_map.rows$ 
5:  $cols \leftarrow energy\_map.cols$ 
6: 构造矩阵  $M$  和回溯矩阵  $backtrack\_path$ , 并初始化为零
7: 将能量图的第一行复制到  $M$  的第一行
8: for  $i = 1$  to  $rows - 1$  do
9:   for  $j = 0$  to  $cols - 1$  do
10:    计算  $CL$ ,  $CU$ , 和  $CR$  的能量变化
11:    计算能量值:
           $e1 = M[i - 1, j - 1] + CL$  (如果有效)
           $e2 = M[i - 1, j] + CU$ 
           $e3 = M[i - 1, j + 1] + CR$  (如果有效)
          
12:    if  $e1 \leq e2$  且  $e1 \leq e3$  then
13:       $M[i, j] \leftarrow e1$ 
14:       $backtrack\_path[i, j] \leftarrow j - 1$ 
15:    else if  $e2 \leq e1$  且  $e2 \leq e3$  then
16:       $M[i, j] \leftarrow e2$ 
17:       $backtrack\_path[i, j] \leftarrow j$ 
18:    else
19:       $M[i, j] \leftarrow e3$ 
20:       $backtrack\_path[i, j] \leftarrow j + 1$ 
21:    end if
22:  end for
23: end for
24:  $min\_loc \leftarrow minMaxLoc(M[rows - 1])$ 
25:  $last\_row\_min\_j \leftarrow min\_loc.x$ 
26: for  $i = rows - 2$  to  $0$  do
27:    $seam[i] \leftarrow backtrack\_path[i + 1, seam[i + 1]]$ 
28: end for
29: return  $seam$ 

```

首先，算法会判断输入图像 $current_image$ 和能量图 $energy_map$ 是否为空，如果是，则返回一个空的路径。在确认图像有效后，算法获取能量图的行数 $rows$ 和列数 $cols$ ，并初始化两个动态规划表： M 表示到达某个像素的最小累积能量， $backtrack_path$ 用于记录每个像素的最优前驱像素。

接着，算法复制能量图的第一行到动态规划表 M 中。然后，对于从第二行开始的每一行，算

法逐列计算该像素的三个可能的能量变化：左侧（CL）、上方（CU）和右侧（CR）像素的能量变化。这些变化的计算基于当前像素与相邻像素的颜色差异。对于每一个像素，选择具有最小能量变化的方向，并将该方向的累积能量存入 M 中，同时在 backtrack_path 中记录该像素的前驱位置。

完成动态规划表的填充后，算法通过在最后一行中寻找最小能量值的像素，确定路径的起始点。接下来，算法从该起始点开始回溯，通过 backtrack_path 表找出最优的垂直缝隙路径。最终返回的 seam 向量包含了该路径的列索引，表示在每一行中应删除的像素位置。

(四) 第四节：效果展示

我们选一张可以凸显 seam-carving 优势的图片，这张图片大小是 960×1031 个像素，我们在水平垂直各删除 350 个 pixel.



图 2: seam-carving 效果对比

相比于传统缩放，可以看出 seam-carving 更多的凸显了主体上的优势。

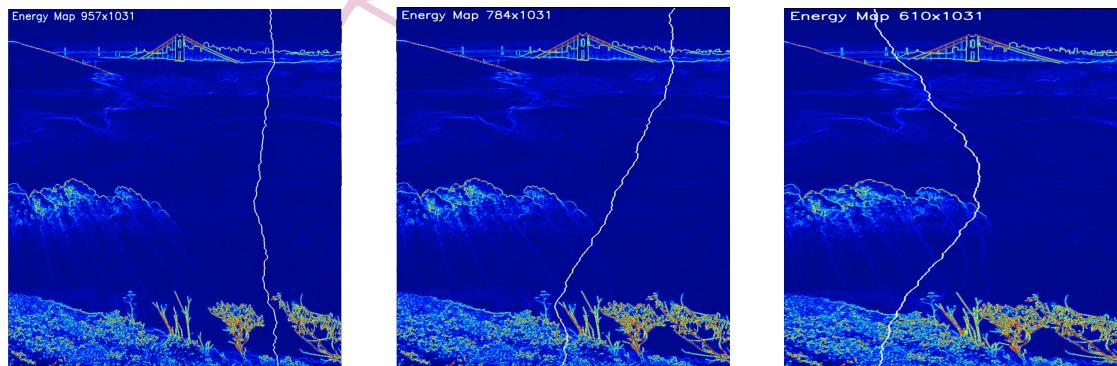


图 3: 能量图可视化

通过能量图的变化，我们可以直观的感受到 seam-carving 的优势，在背景是低能量区域时，传统缩放往往无法凸显主体，而 seam-carving 非常好的重新调整了主题和背景的相对大小关系。其中 Basic algorithm 即是传统 seam-carving 算法，而 Optimized algorithm 是使用一维数组存储 DP 表，以提高缓存命中率的优化结果，可以看到时间的缩短，Forward energy 是我们提出的基于能量图的搜索算法。

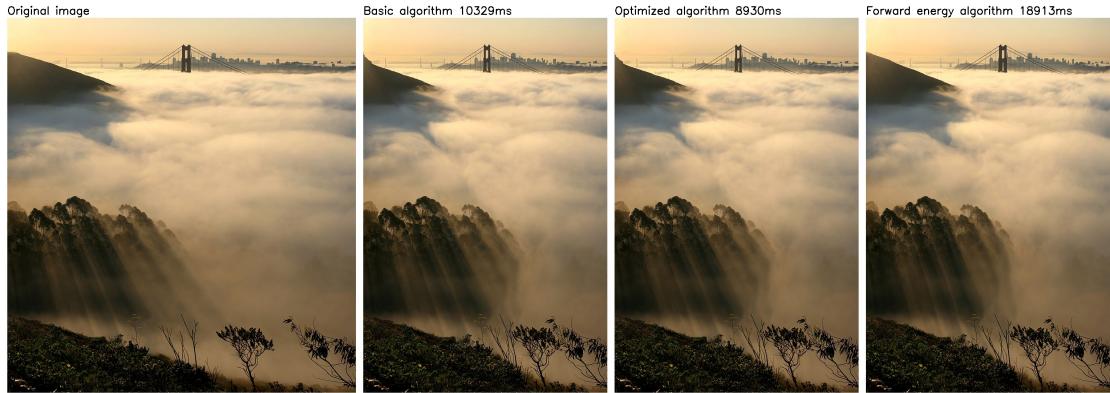


图 4: 不同优化可视化

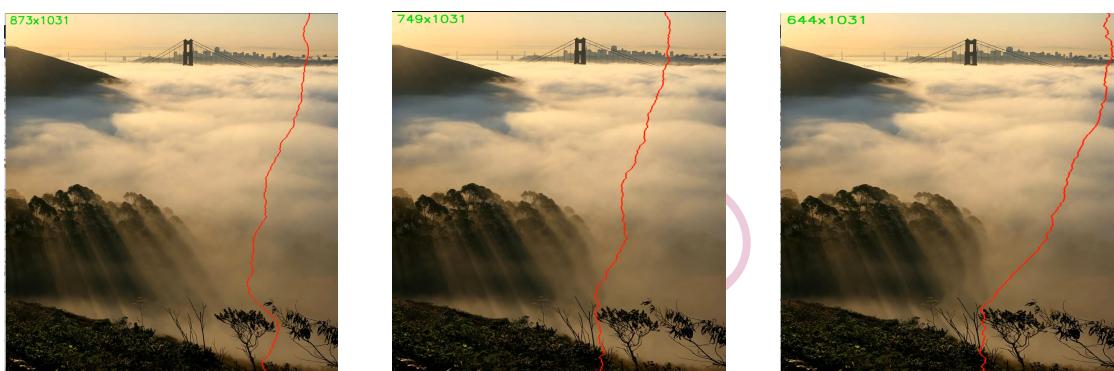


图 5: 能量图可视化

但很遗憾的是 seam-carving 也存在缺陷，尤其在于主体占满整个图像时，可能会导致主体的变形，在人脸方面尤其明显。



比如最开始我使用南开大学的图片，建筑的主体充满图像，会造成建筑主题的变形。

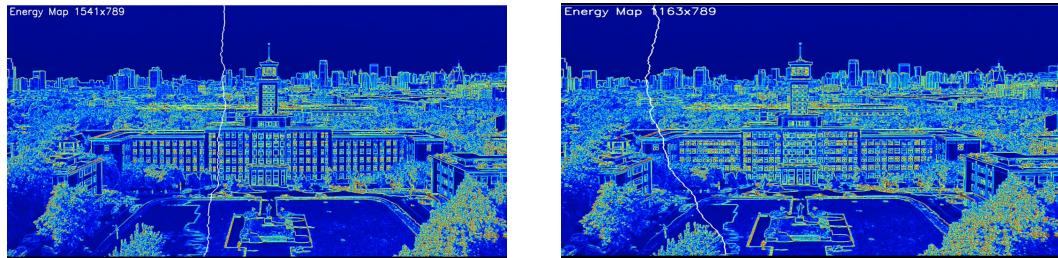


图 6: 失败的例子

可以看到左图为刚开始的 seam 线，其关注点在主楼的中心，不断删除 seams，造成主楼的变形，之后才关注到两侧的背景区域，但主楼的变形已无法改变。类似的 failure cases 可在 [Image Resizing with Seam Carving](#) 中找到

二、总结

本报告中介绍了 Seam Carving 算法的核心原理和实现方式，重点分析了基于动态规划的垂直缝隙搜索算法。在实际应用中，Seam Carving 算法能够在图像尺寸调整的过程中，通过移除或插入能量最低的缝隙，显著保留图像的主要内容，避免传统均匀缩放带来的变形问题。实验结果表明，Seam Carving 在大多数场景下表现出色，尤其在图像背景低能量的情况下，能够有效保留主体。然而，当图像的主体区域占据较大空间时，算法可能会引起主体的形变，尤其是人脸和建筑物等重要结构的失真。为了应对这一问题，未来的研究可以进一步优化缝隙选择策略，探索更加智能的区域自适应删除或插入方法。此外，结合并行计算和 GPU 加速技术，未来的算法执行速度和处理能力也有很大的提升空间。