IMAGE RESIZING & SEAM CARVING

CS16: Introduction to Algorithms & Data Structures
Outline

1) Image Resizing
2) Seamcarving Algorithm
3) Digression: argmin
Image Resizing

• Problem
  • resize image without affecting proportions

• Intuition
  • Resize by removing *uninteresting* areas
  • Keep *interesting* areas the same
Image Resizing

- Important elements of the photo should be preserved
- Repetitive areas of the image (water, sand) can be reduced
Image Resizing
Image Resizing

• To shrink image
  • remove unimportant pixels

• Quantify pixel importance by
  • How much it varies from neighbors
  • Sum of differences in intensity with neighbors
# Image Resizing

- Grayscale 3x3 image with the following pixel intensities
- What’s the importance of the center pixel?

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>
**Image Resizing**

- Grayscale 3x3 image with the following pixel intensities:
- What’s the importance of the center pixel? \( 1 + 3 + 2 + 3 = 9 \)

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>
```
Image Resizing

- Apply this method to every pixel of an image to determine the most and least “important” pixels.

Low                  High
Image Resizing: Approach 1

- Remove the least important pixels in order
- Looks terrible! Not removing the same amount from each row, causing jagged right side
Image Resizing: Approach 2

- Remove $n$ least important pixels in each row
- Still not great, too much shifting between adjacent rows
Image Resizing: Approach 3

- Remove the column whose total importance is smallest, and repeat
- Much better! But not perfect…
Image Resizing

• Problem: removing an entire column (or an entire row) distorts the image

• What pixels would we want to remove to resize this image?
Image Resizing: Seamcarve

• Solution: remove **seams** not columns
• Seam: path from top to bottom that moves left or right by at most one pixel per row (vertical seam)
Image Resizing: Seamcarve

- Using the seamcarving approach.
- Near perfection!
Seamcarve: Another Application

- User marks the object to be removed as “unimportant” by artificially deflating the importance of the target pixels
- These pixels are automatically removed by the algorithm
The Seamcarve Algorithm

Function: `find_least_important_seam(vals)
Input: `vals`, a 2D array of importance values
Output: sequence of column-indices, each differing from the last by at most 1, with corresponding pixels being the least-total-importance vertical seam

```
[[ - S - - ]],
[ S - - - ],
[ - S - - ],
[ - - S - ]
```

→ `[1, 0, 1, 2]`
Data Structures needed

- **costs** - 2D array, filled in from bottom to top
  - `costs[row][col]` holds the total importance of the lowest cost seam starting from the bottom row and ending at `costs[row][col]`

- **dirs** - 2D array, filled in at the same time as `costs`
  - `dirs[row][col]` holds the direction (-1, 0, or 1) of the previous pixel in the lowest cost seam ending at `costs[row][col]`

\[
 costs[row][col] = \min(costs[row+1][col-1 \text{ to } col+1]) + \text{vals}[row][col] 
\]
Data Structures illustrated

original image

vals

costs

high

low
Finding the least important seam

- Once costs has been completely filled in, the cell with the minimum value in the top row of costs will be the first pixel in the least important seam of the entire image.

- Starting from that pixel, we can use the dirs array to backtrack our way through the rest of the seam and build the final list of column indices.
Seamcarve Pseudocode

```
function find_least_important_seam(vals):
    dirs = 2D array with same dimensions as vals
    costs = 2D array with same dimensions as vals
    costs[height-1] = vals[height-1] // initialize bottom row of costs

    for row from height-2 to 0:
        for col from 0 to width-1:
            costs[row][col] = vals[row][col] +
            min(costs[row+1][col-1],
                costs[row+1][col],
                costs[row+1][col+1])
            dirs[row][col] = -1, 0, or 1 // depending on min

    // Find least important start pixel
    min_col = argmin(costs[0]) // Returns index of min in top row

    // Create vertical seam of size ‘height’ by tracing from top
    seam = []
    seam[0] = min_col
    for row from 0 to height-2:
        seam[row+1] = seam[row] + dirs[row][seam[row]]

    return seam
```
Digression: What’s \texttt{argmin()}?

- What does the \texttt{min()} function do?
  - Returns the minimum output value of a function
- So what about the \texttt{argmin()} function?
  - Given a function \( f(x) \), returns the value of \( x \) for which \( f(x) \) is minimal.
- Examples:
  - Consider the function \( f(x) = -1 + x^2 \)
    - \( \text{min}(f) = -1 \)
    - \( \text{argmin}(f) = 0 \) \( \leftarrow \) the value of \( x \) for which \( f(x) = -1 \)
  - Consider the array \( L = [5, 4, 1, 3, 9] \)
    - \( \text{min}(L) = 1 \)
    - \( \text{argmin}(L) = 2 \) \( \leftarrow \) The index of the minimum value
Readings

• You might want to check out the original seamcarve paper
  • http://www.eng.tau.ac.il/~avidan/papers/imret Final.pdf
• Don’t expect to understand all of it but it has a lot of great examples and is a worthwhile read