CSC-301: Seam Carving

Timur Kasimov, Peter Murphy, Krishna Nayar

May 9, 2024

References:

Class textbook, professor Rebelsky

Part 1

14-8a

Suppose we are given a color picture consisting of an m•n array $A[1 : m, 1 : n]$ of pixels. We show that the number of vertical seams in such a picture grows at least exponentially in m assuming that $n > 1$.

Proof. First, note that to start a seam, we first select one of the n pixels in the first row ($m = 0$). Once we have one pixel in the first row, we have three options for the next pixel in the next row, which is a pixel on the left, down, or right (two options on edge cases). As we continue down the picture, one row at a time, we keep multiplying the total number of seams so far by 3 with each additional row for all the remaining $m - 1$ rows. This means that the number of possible seams is given by $n \cdot 3^{m-1}$. This shows the exponential growth of the number of seams with respect to m.

□

Part 2: Seam Carving Algorithm

Intuition:

We use dynamic programming with a bottom up approach to solve this problem. From our calculated *RGBenergymatrix* we construct a *solutionenergymatrix* with values in each entry representing the energy of the optimal seam starting at that entry's corresponding pixel. At the same time we construct a *solutionpathmatrix* of the same dimensions as the rgb energy and solution energy matrices, except the *solutionpathmatrix* will have values entries of -1, 0, and 1 representing down-left, down, and down-right movement which constructs the optimal (lowest energy) seam, one row at a time.

We begin constructing the solution energy matrix by iterating through the bottom-most row of the rgb energy matrix and simply copying the values of that row. We then iterate through the second from bottom row of the rgb energy matrix. For each entry we sum its energy cost and the minimum of the energies to the bottom-left, bottom, and bottom-right, since a seam can only go through these possible pixels, as the two pixels in neighbouring rows need to be adjacent. For the remaining rows, we add a particular entry's energy cost in the rgb energy matrix and the minimum value of the down-left, down, down-right vertices of the solution energy matrix. Thus the solution energy matrix should have varying but increasing values as we move up since we have to cut through more pixels which increases our energy sums. The dynamic programming approach here is clear as we start solving the subproblems and then simply refer to our solution table in *solutionenergymatrix* to solve larger superproblems, without ever solving the same problem again.

We construct the solution path matrix at the same time as the solution energy matrix, excluding the bottom row. When considering the down-left, down, down-right vertices for a particular entry and choose the minimum, we will record the optimal direction. If the smallest solution energy vertex is bottom-left we enter -1; if its bottom-center we enter 0; if its bottom-right we enter 1. Thus we fill the path matrix up with a bunch of -1s, 0s, and 1s.

When the solution energy and path matrices have been filled, finding the path is simple. We look at the top row of the solution energy matrix and consider the vertex with the smallest value. From this vertex coordinate, we look at the path matrix and follow the path from this coordinate all the way to the bottom by considering the -1s, 0s, and 1s in each entry. We store the coordinates of each vertex in this path. This is the path we will carve out.

Pseudocode

Assume e is the energy function which has 3 parameters. The first parameter is the input image. The x and y coordinate of the pixel are the second and third parameters. Assume q is the image. Returns pixels of the minimum energy seam.

```
Optimal-Seam(e,q):
    \\construct energy matrix
    rgb-energy-matrix = empty-2d-array(rows(q),cols(q))
    for row from 1 to rows(q):
        for col from 1 to cols(q):
            rgb-energy-matrix[row,col] = e(q,row,col)

    \\ initialize two auxilllary matrices
    solution-path-matrix = empty-2d-array(rows(q),cols(q))
    solution-energy-matrix = empty-2d-array(rows(q),cols(q))

    \\copy bottom row for solution energy matrix
    solution-energy-matrix[rows(q),1:] = rgb-energy-matrix [rows(q),1:]

    \\loop down to up starting at 2nd to bottom row
    for row from (rows(q)-1) to 1:
        for col from 1 to cols(q):

            \\record left, right, down solutions
            left = infinity
            right = infinity
            if col != 0:
                left = rgb-energy-matrix[row,col] + solution-energy-matrix[row+1, col-1]
            if col != cols(q):
                right = rgb-energy-matrix[row,col] + solution-energy-matrix[row+1, col+1]
            down = rgb-energy-matrix[row,col] + solution-energy-matrix[row+1, col]

            \\find optimal solution
            solution = min(down,left,right)
            solution-energy-matrix[row,col] = solution

            \\figure out the correct path
            if (solution == left):
                path_matrix[row, col] = -1
            elif (solution == right):
                path_matrix[row, col] = 1
            else:
                path_matrix[row, col] = 0

    \\ now construct optimal seam
```

```
optimal-seam = empty-array(rows(q))
\\find starting point from top of solution-matrix
starting-coord-pair = get-coord(minimum-energy-vertex(solution-energy-matrix[1,1:]))
optimal-seam[1] = q[starting-coord-pair.x,starting-coord-pair.y]
\\get the seam by following the solution-path-matrix
lastcoord = starting-coord-pair
for vertex from 2 to rows(q):
    newcoord = lastcoord
    path-value = solution-path-vertex[lastcoord.x,lastcoord.y]
    \\shift down 1
    newcoord.y = newcoord.y + 1
    \\shift left, right, or no shift
    if(path-value == -1):
        newcoord.x = newcoord.x - 1
    else if(path-value == 1)
        newcoord.x = newcoord.x + 1
    optimal-seam[vertex] = q[newcoord.x,newcoord.y]
    lastcoord = newcoord
return(optimal-seam)
```

Proof of Correctness

Loop Invariant: Every entry in the i-th row from the bottom of our solution matrix has the energy associated with the optimal solution that can be created from that entry.

Initialization: Before the first iteration of the loop, we simply look at the bottom row of the solution matrix. Since optimal solutions created from those entries are one element long (that is, they are simply that element), each element's associated energy is, by default, the energy associated with the optimal solution that can be created from that entry.

Maintenance: Before the start of iteration i, each element of row $i-1$ (counting from the bottom) of our energy matrix contains an energy level, and each element of row $i-1$ of our path matrix contains $-1, 0$, or 1 to help indicate the optimal path. During the iteration, each element in row i of the solution matrix becomes the sum of the corresponding element of the energy matrix and the optimal step down-left, down, or down-right below this entry. So, at the end of iteration, each element of row i contains the energy associated with the optimal solution that can be created from that entry - indicating the loop invariant holds after an iteration of the loop.

Termination: The loop variable i decreases by 1, with the loop terminating once $i = 0$. When that is the case, we are at the top row. As entries in the top row contain the optimal energy needed for a seam through the entire image, we select the minimum element of the top row

for our minimum-energy seam, which is indicating that our algorithm is correct.

Runtime:

Assume m, n represent number of rows and columns respectively. Constructing the energy matrix alone takes $\Theta(mn)$ runtime since we iterate through each entry in the matrix, as we can see in the nested for loop at the beginning of the pseudocode. Looping down up to construct solution energy matrix similarly takes $\Theta(mn)$ runtime, as indicated by the nested for loop again. Note that the calls within the nested for loop, such as initializations, or calculating energy of the specific pixel using e function, or the if conditionals all take constant time.

Constructing the optimal seam only takes $\Theta(m)$ runtime, as we iterate through every row and the rest of the inner operations take constant time.

As such, the overall algorithm's runtime is $\Theta(mn)$.