Lecture 9

Computer Vision

Edge Detection

Some slides from: S. Seitz, D. Jacobs, D. Lowe, H. Man, K. Grauman, D. Hoiem, S. Lazebnik
Outline

• Edge Detection
  • Edge types
  • Image Gradient
  • Canny Edge Detector

• Application
  • intelligent image resizing: Seam Carving
Edge Detection

- Convert intensity image into binary (0 or 1) image that marks prominent curves
- What is a prominent curve?
  - no exact definition
  - intuitively, it is a place where abrupt changes occur
- Why perform edge detection?
  - most shape and semantic and information is encoded in edges
  - edges are stable to lighting and other changes, makes them good features for object recognition, etc.
  - more compact representation than intensity
Line Drawings

• Artists do it
  • and much better, as they use high level knowledge which edges are more perceptually important
Origin of Edges

- Many **discontinuity** causes:
  - surface color or texture discontinuity
  - depth discontinuity (object boundary)
  - surface normal discontinuity
  - illumination discontinuity (shadows)
Derivatives and Edges

- An edge is a place of rapid change in intensity.

![Image showing an edge, intensity function, and first derivative.](image)

- Edges correspond to extrema of the derivative function.
Derivatives with Convolution

- For 2D function $f(x,y)$, partial derivative in horizontal direction

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\varepsilon \to 0} \frac{f(x + \varepsilon, y) - f(x, y)}{\varepsilon}$$

- For discrete data, approximate

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1, y) - f(x, y)}{1}$$

- Similarly, approximate vertical partial derivative (wrt $y$)

- How to implement as a convolution?
Image Partial Derivatives

Which is with respect to $x$?

\[ \frac{\partial f(x, y)}{\partial x} \]

or

\[ \frac{\partial f(x, y)}{\partial y} \]

\[
\begin{pmatrix}
-1 & 1 \\
+1 & -1
\end{pmatrix}
\]

or

\[
\begin{pmatrix}
-1 & 1 \\
1 & -1
\end{pmatrix}
\]
Finite Difference Filters

- Other filters for derivative approximation

Prewitt: \[ H_x = \frac{1}{6} \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad H_y = \frac{1}{6} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \]

Sobel: \[ H_x = \frac{1}{8} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad H_y = \frac{1}{8} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \]
Image Gradient

- Combine both partial derivatives into vector \( \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \end{bmatrix} \)

- Gradient points in the direction of most rapid increase in intensity

\[
\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x}, 0 \end{bmatrix} \quad \nabla f = \begin{bmatrix} 0, \frac{\partial f}{\partial y} \end{bmatrix} \quad \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \end{bmatrix}
\]

- **Direction** perpendicular to edge:
  \[
  \theta = \tan^{-1}\left(\frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}}\right)
  \]
  gradient orientation

- **Edge strength**
  \[
  \|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}
  \]
  gradient magnitude
Application: Gradient-domain Image Editing

- Goal: solve for pixel values in the target region to match gradients of the source region while keeping background pixels the same.

Simplest Edge Detector

- Compute gradient magnitude at each pixel
  
  \[ g(x, y) = \| \nabla f \| = \sqrt{\left( \frac{\partial f}{\partial x} \right)^2 + \left( \frac{\partial f}{\partial y} \right)^2} \]

- Threshold gradient magnitude to get binary edge map \( e(x,y) \)
  
  \[ e(x, y) = \begin{cases} 
  1 & \text{if } g(x,y) > T \\
  0 & \text{otherwise} 
  \end{cases} \]
Effects of Noise

- Too many pixels with large gradient magnitude due to image noise

\[ \frac{\partial f}{\partial y} \quad \frac{\partial f}{\partial x} \]
Effects of noise

- Consider a single row of the image
- Plot intensity as a function of $x$

\[ f(x) \]

\[ \frac{\partial}{\partial x} f(x) \]

- Where is the edge?
Effects of Noise

• How do we deal with noise?
• We already know, filter the noise out using Gaussian kernel
• First convolve image with a Gaussian filter
• Then take derivative
Derivative Theorem of Convolution

\[
\frac{\partial}{\partial x} (H \ast f) = \left( \frac{\partial}{\partial x} H \right) \ast f
\]

- This saves us one step

\[
f
\]

\[
\frac{\partial}{\partial x} H
\]

\[
\left( \frac{\partial}{\partial x} H \right) \ast f
\]

(edge)
Derivative of Gaussian

\[ G_\sigma \quad \frac{\partial}{\partial x} G_\sigma \quad \frac{\partial}{\partial y} G_\sigma \]

- Which finds horizontal, which vertical edges?

white is positive values, dark negative, gray zero

• Which finds horizontal, which vertical edges?
Derivative of Gaussian: Example

• Ignoring constant:

\[ G_\sigma(x, y) = e^{-\frac{(x^2 + y^2)}{2\sigma^2}} \]

• Differentiate:

\[ \frac{\partial}{\partial x} G_\sigma(x, y) = -\frac{x}{\sigma^2} \cdot e^{-\frac{(x^2 + y^2)}{2\sigma^2}} \]

\[ \frac{\partial}{\partial y} G_\sigma(x, y) = -\frac{y}{\sigma^2} \cdot e^{-\frac{(x^2 + y^2)}{2\sigma^2}} \]

• Plug in \( \sigma = 5 \), and take 5×5 window

\[
\begin{array}{cccccc}
(-2,-2) & (-1,-2) & (0,-2) & (1,-2) & (2,-2) \\
(-2,-1) & (-1,-1) & (0,-1) & (1,-1) & (2,-1) \\
(-2,0) & (-1,0) & (0,0) & (1,0) & (2,0) \\
(-2,1) & (-1,1) & (0,1) & (1,1) & (2,1) \\
(-2,2) & (-1,2) & (0,2) & (1,2) & (2,2) \\
\end{array}
\]

coordinates in window

\[
\begin{array}{cccccc}
0.04 & 0.08 & 0 & -0.08 & -0.04 \\
0.16 & 0.37 & 0 & -0.37 & -0.16 \\
0.27 & 0.61 & 0 & -0.61 & -0.27 \\
0.16 & 0.37 & 0 & -0.37 & -0.16 \\
0.04 & 0.08 & 0 & -0.08 & -0.04 \\
\end{array}
\]

\[
\begin{array}{cccccc}
-0.04 & -0.16 & -0.27 & -0.16 & -0.04 \\
-0.08 & -0.37 & -0.61 & -0.37 & -0.08 \\
0 & 0 & 0 & 0 & 0 \\
0.08 & 0.37 & 0.61 & 0.37 & 0.08 \\
0.04 & 0.16 & 0.27 & 0.16 & 0.04 \\
\end{array}
\]

\( H_x \) \hspace{1cm} \( H_y \)
**Example Continued**

<table>
<thead>
<tr>
<th></th>
<th>0.04</th>
<th>0.08</th>
<th>0</th>
<th>-0.08</th>
<th>-0.04</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_x$</td>
<td>0.16</td>
<td>0.37</td>
<td>0</td>
<td>-0.37</td>
<td>-0.16</td>
</tr>
<tr>
<td></td>
<td>0.27</td>
<td>0.61</td>
<td>0</td>
<td>-0.61</td>
<td>-0.27</td>
</tr>
<tr>
<td></td>
<td>0.16</td>
<td>0.37</td>
<td>0</td>
<td>-0.37</td>
<td>-0.16</td>
</tr>
<tr>
<td></td>
<td>0.04</td>
<td>0.08</td>
<td>0</td>
<td>-0.08</td>
<td>-0.04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>-0.04</th>
<th>-0.16</th>
<th>-0.27</th>
<th>-0.16</th>
<th>-0.04</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_y$</td>
<td>-0.08</td>
<td>-0.37</td>
<td>-0.61</td>
<td>-0.37</td>
<td>-0.08</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0.08</td>
<td>0.37</td>
<td>0.61</td>
<td>0.37</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>0.04</td>
<td>0.16</td>
<td>0.27</td>
<td>0.16</td>
<td>0.04</td>
</tr>
</tbody>
</table>

- apply $H_x$ to pixel in red: -0.78
- apply $H_y$ to pixel in red: 0.46

- apply $H_x$ to pixel in red: 217
- apply $H_y$ to pixel in red: 0.69
Example Continued

![Matrix Image]

\[
H_x = \begin{bmatrix}
0.04 & 0.08 & 0 & -0.08 & -0.04 \\
0.16 & 0.37 & 0 & -0.37 & -0.16 \\
0.27 & 0.61 & 0 & -0.61 & -0.27 \\
0.16 & 0.37 & 0 & -0.37 & -0.16 \\
0.04 & 0.08 & 0 & -0.08 & -0.04 \\
\end{bmatrix}
\]

\[
H_y = \begin{bmatrix}
-0.04 & -0.16 & -0.27 & -0.16 & -0.04 \\
-0.08 & -0.37 & -0.61 & -0.37 & -0.08 \\
0 & 0 & 0 & 0 & 0 \\
0.08 & 0.37 & 0.61 & 0.37 & 0.08 \\
0.04 & 0.16 & 0.27 & 0.16 & 0.04 \\
\end{bmatrix}
\]

apply $H_x$ to pixel in red: **217**
apply $H_y$ to pixel in red: **0.69**

apply $H_x$ to pixel in red: **-0.69**
apply $H_y$ to pixel in red: **-217**

Mask looks like the pattern it is trying to detect!
What does this Mask Detect?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>-4</th>
<th>-4</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>-4</td>
<td>-4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-4</td>
<td>-4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-4</td>
<td>-4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-4</td>
<td>-4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-4</td>
<td>-4</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

- strong negative response
- strong positive response
What Does this Mask Detect?

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>-2</td>
<td>-2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>-2</td>
<td>-2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

**strong negative response**

**strong positive response**
Canny Edge Detector

input image
Canny Edge Detector

gradient magnitude
Canny Edge Detector

thresholding
• Why we get thick regions after thresholding?

\[ f(x) \]

\[ \|\nabla f(x)\| \]

\[ \text{threshold} \]

\[ \text{edge} \]
Edge Thinning: non-maximum suppression

- Check if pixel $q$ is local maximum along gradient direction
  - take two neighbors in $p$ and $r$ in the gradient direction
    - requires checking interpolated pixels $p$ and $r$
  - turn off edge at pixel $q$ if $g(q) < g(p)$ or $g(q) < g(r)$
Another problem: some weak edge pixels do not survive thresholding after thinning
The Canny Edge Detector

- Try a smaller threshold?
  - too many weak edges
Hysteresis Thresholding

- Specify a **high** and **low** thresholds
- Use **high** threshold to start edge curves
  - Continue edge in the gradient direction
  - Use **low** threshold for continuation
The Canny Edge Detector

low threshold  

high threshold

hysteresis with low and high thresholds
Effect of Kernel Size and Spread

- Smaller $\sigma$ / mask size detects fine scale edges
- Larger $\sigma$ / mask detects large scale edges
Still Far From Human Vision

- Berkeley segmentation database:
  
  http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/
State-of-the-Art in Contour Detection

- Canny
- Learned with combined features
- Human

Source: Jitendra Malik: http://www.cs.berkeley.edu/~malik/malik-talks-ptrs.html
Illusory Contours

- impossible detect the “illusory” contours using only local image gradients
Application of Gradients: Intelligent Resizing

- In traditional image resizing, all dimensions change by the same ratio.
Application of Gradients: Intelligent Resizing

- What if need to fit to a mobile device? Or resize to a web page?
  - often need different ratio in different dimensions

- Change width, height unchanged

- Object proportions are not preserved:
Application of Gradients: Intelligent Resizing

- Intelligent resizing “seam carving”
- Shai & Avidan, SIGGRAPH 2007

**seam carving**

**naïve resizing**
Seam Carving: Main Idea

- Preserve the most “interesting” content
  - large gradient magnitude = interesting
  - small gradient magnitude = uninteresting

- Prefer changes around low gradient magnitude pixels
Reducing Width by One Pixel

- Traditional resizing
  - works on regular seams
  - through random pixels
- Seam carving
  - find irregular seams
  - through low gradient (uninteresting) pixels

\[
\begin{align*}
&\text{add two seams}\quad \text{w} \\
&\text{remove seam}\quad \text{w}
\end{align*}
\]

\[
\begin{align*}
&\text{w} - 1 \\
&\text{w} - 1
\end{align*}
\]
Seam Carving: Main Idea

- Prefer changes around low gradient magnitude pixels
  - to reduce size in one dimension, *remove* irregular seams
  - to enlarge size in one dimension, *insert* irregular seams
- Many “uninteresting” seams
  - find the best (most boring) seam
  - with dynamic programming
Seam Carving: Main Idea

- Measure **energy** as gradient magnitude
- Removing low energy seam makes change less visible
- Choose seam based on **minimum total energy path**
- Path is **8-connected**
Example

Original Image

Gradient Energy

blue = low energy
red = high energy
Example
Example
Example
Example
Seam Carving: Algorithm

- Vertical seam $s$ consists of $n$ positions that form a path
  - $s = (s_1, s_2, ..., s_n)$: one pixel in every row
- Seam cost $\text{Energy}(s) = \text{Energy}(s_1) + \text{Energy}(s_2) + ... + \text{Energy}(s_n)$
  - red seam has cost $0 + 6 + 3 + 6 + 1 = 16$
  - green seam has cost $4 + 4 + 2 + 4 + 6 = 20$
- Optimal seam minimizes this cost
  $$s^* = \text{argmin}_s \text{Energy}(s)$$
How to Find the Minimum Cost Seam?

- First, consider a **greedy** approach on a small image
  - smaller number corresponds to smaller gradient

![Image of a grid with numbers and highlighted paths]

- Greedy seam cost: $0 + 1 + 3 + 0 + 5 = 9$
- Is this the best vertical seam?
• Dynamic programming can find the best seam
• Work from the top row to the bottom row
• $M(r,c)$ is best seam cost that starts anywhere in row 0 end ends at position $(r,c)$
• After $M$ is computed, the best cost path is the smallest value of $M$ in the last row
• Also keep track of the parent on the path, $P(r,c)$
Seam Carving Algorithm: Initialization Step

Compute Energy image $E$

for $c = 1$ to $maxCol$

$M(1, c) = E(1, c)$

$P(1, c) = \text{null}$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

$E$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

$M$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

$P$
Seam Carving Algorithm: Iteration Step

- Computed $M, P$ for rows 0 to $k$
- How to compute $M, P$ for row $k+1$?

$M(r+1, c) = E(r+1, c) + \text{smallest in}\{M(r, c-1), M(r, c), M(r, c+1)\}$

$P(r+1, c)$ stores corresponding column
  - either $c-1$, or $c$, or $c+1$
for $r = 1$ to $\text{maxRow}$
for $c = 1$ to $\text{maxCol}$

    option1 = $M(r-1,c-1)$
    option2 = $M(r-1,c)$
    option3 = $M(r-1,c+1)$

    if option1 $\leq$ option2 and option1 $\leq$ option3
        $M(r,c) = E(r,c) + M(r-1,c-1)$
        $P(r,c) = c-1$
    elseif option2 $\leq$ option1 and option2 $\leq$ option3
        $M(r,c) = E(r,c) + M(r-1,c)$
        $P(r,c) = c$
    else
        $M(r,c) = E(r,c) + M(r-1,c+1)$
        $P(r,c) = c+1$

!!!Note: have to implement matrix out of bounds check!!!
Example: Initialization

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>0</th>
<th>6</th>
<th>4</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

\(E\)

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>0</th>
<th>6</th>
<th>4</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(M\)

<table>
<thead>
<tr>
<th></th>
<th>null</th>
<th>null</th>
<th>null</th>
<th>null</th>
<th>null</th>
</tr>
</thead>
</table>

\(P\)
Example: Iteration 1

\[ E = \begin{bmatrix} 3 & 0 & 6 & 4 & 2 \\ 1 & 3 & 6 & 6 & 4 \\ 4 & 3 & 4 & 6 & 2 \\ 4 & 6 & 0 & 0 & 4 \\ 0 & 6 & 5 & 6 & 6 \end{bmatrix} \]

\[ M = \begin{bmatrix} 3 & 0 & 6 & 4 & 2 \\ 1 & 3 & 6 & 8 & 6 \end{bmatrix} \]

\[ P = \begin{bmatrix} \text{null} & \text{null} & \text{null} & \text{null} & \text{null} \\ 2 & 2 & 2 & 5 & 5 \end{bmatrix} \]
### Example: Iteration 2

#### Example Matrix

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

#### Example Matrix

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>7</td>
<td>12</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

#### Example Matrix

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Example: Iteration 3

<table>
<thead>
<tr>
<th>E</th>
<th>M</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Example: Iteration 4

- Best seam has cost 8, better than what greedy algorithm finds
  - end of the best seam is in column 0

\[
\begin{array}{c|c|c|c|c|c}
3 & 0 & 6 & 4 & 2 \\
\hline
1 & 3 & 6 & 6 & 4 \\
\hline
4 & 3 & 4 & 6 & 2 \\
\hline
4 & 6 & 0 & 0 & 4 \\
\hline
0 & 6 & 5 & 6 & 6 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c}
3 & 0 & 6 & 4 & 2 \\
\hline
1 & 3 & 6 & 8 & 6 \\
\hline
5 & 4 & 7 & 12 & 8 \\
\hline
8 & 10 & 4 & 7 & 12 \\
\hline
8 & 10 & 9 & 10 & 13 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c}
null & null & null & null & null \\
\hline
2 & 2 & 2 & 5 & 5 \\
\hline
1 & 1 & 2 & 3 & 5 \\
\hline
2 & 2 & 2 & 3 & 5 \\
\hline
1 & 3 & 3 & 3 & 4 \\
\end{array}
\]
### Example: Finishing Up

#### $M$

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>0</th>
<th>6</th>
<th>4</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>7</td>
<td>12</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>4</td>
<td>7</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

#### $P$

<table>
<thead>
<tr>
<th></th>
<th>null</th>
<th>null</th>
<th>null</th>
<th>null</th>
<th>null</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

**original image**

**new smaller image**
Other notes on seam carving

- Can also insert seams to *increase* size of image
  - duplicate optimal seam, averaged with neighbors
- Analogous procedure for horizontal seams
- Other energy functions may be plugged in
  - e.g., color-based
- Can remove (or keep, or enlarge) marked objects
Some Results

brings friends closer or draws them apart
Include Color in Energy

- Want to remove objects of red color
  - \( R_f, G_f, B_f \) are red, green blue color channels of image \( f \)

\[
\| \nabla f \| - 2R_f + G_f + B_f = \text{energy}(f)
\]

input image \( f \)  
carving out red
Include Color in Energy

- That hat is too big - get rid of some green

\[
\text{energy}(f) = ||\nabla f|| - 2G_f + R_f + B_f
\]

input image \( f \)  
carved
Some Results
Removal of a Marked Object

- Mask image $M$ is 1 for object, 0 otherwise
- remove vertical and horizontal seams

\[
\|\nabla f\| - 1000 \cdot M = \text{energy}(f)
\]
Insert More Marked Object

- Same energy, now insert vertical seams

\[
\begin{align*}
\|\nabla f\| - 1000 \cdot M &= \text{energy}(f)
\end{align*}
\]
Sometimes Remove Mask not Enough
Remove and Preserve Mask

- $M$ is 1 for pixels to remove, -1 for pixels to keep, 0 for neutral

$$-1000 \cdot \nabla f \cdot M = \text{energy}(f)$$
Carving a Caricature
Sometimes it Fails