Image Resizing and Blending

BIL721: Computational Photography
Spring 2015, Lecture 5

Aykut Erdem
Hacettepe University
Computer Vision Lab (HUCVL)
Today

• Image Resizing
  – Seam Carving

• Image Compositing and Blending
  – Alpha compositing
  – Laplacian pyramid blending
  – Poisson blending
Today

• Image Resizing
  – Seam Carving

• Image Compositing and Blending
  – Alpha compositing
  – Laplacian pyramid blending
  – Poisson blending
Display Devices
Content Retargeting
Page Layout
Simple Media Retargeting Operators

Letterboxing

Scaling
Content-aware Retargeting Operators

Content-aware

"Important" content

Content-oblivious
Content-aware Retargeting

Input

Scale

Crop

Content-aware

“less-Important” content
Image Retargeting Objectives

1. Change size

2. Preserve the important content and structures

3. Limit artifacts created
Importance (Saliency) Measures

• A function $S: p \rightarrow [0,1]$

• More sophisticated: attention models, eye tracking (gazing studies), face detectors, ...

Wang et al. 2008

Judd et al. ICCV09 Learning to predict where people look
General Retargeting Framework

1. Define an energy function $E(I)$ (interest, importance, saliency)

2. Use some operator(s) to change the image $I$

- Recompose
  - Setlur et al. [2005]

- Crop
  - Santella et al. [2005]

- Warp
  - Gal et al. [2006]
Previous Retargeting Approaches

- Optimal Cropping Window

- For videos: “Pan and scan”

Still done manually in the movie industry

Cropping
Seam Carving

• Assume $m \times n \rightarrow m \times n'$, $n' < n$

• **Basic Idea: remove unimportant pixels from the image**
  – Unimportant = pixels with less “energy”

$$e_1(I) = |\frac{\partial}{\partial x} I| + |\frac{\partial}{\partial y} I|$$

• **Intuition for gradient-based energy:**
  – Preserve strong contours
  – Human vision more sensitive to edges – so try remove content from smoother areas
  – Simple, enough for producing some nice results
  – See their paper for more measures they have used

“Seam Carving for Content-Aware Image Resizing,”
S. Avidan and A. Shamir, *Proc. SIGGRAPH, 2007*
Pixel Removal

- Optimal (global)
- Least-energy pixels (per row)
- Least-energy columns
A Seam

- A connected path of pixels from top to bottom (or left to right). Exactly one in each row

\[
s^x = \{s^x_i\}_{i=1}^n = \{(x(i), i)\}_{i=1}^n, \text{ s.t. } \forall i, |x(i) - x(i - 1)| \leq 1
\]

\[
s^y = \{s^y_j\}_{j=1}^m = \{(j, y(j))\}_{j=1}^m, \text{ s.t. } \forall j, |y(j) - y(j - 1)| \leq 1
\]
Finding the Seam?
The Optimal Seam

\[ E(I) = |\frac{\partial}{\partial x} I| + |\frac{\partial}{\partial y} I| \Rightarrow s^* = \arg \min_s E(s) \]
The Optimal Seam

- The recursion relation

\[ M(i, j) = E(i, j) + \min(M(i-1, j-1), M(i-1, j), M(i-1, j+1)) \]

- Can be solved efficiently using dynamic programming in \( O(s \cdot n \cdot m) \)

\((s=3 \text{ in the original algorithm})\)
Invariant property:
  \[ M(i,j) = \text{minimal cost of a seam going through } (i,j) \] (satisfying the seam properties)
Dynamic Programming

\[ M(i, j) = E(i, j) + \min(M(i-1, j-1), M(i-1, j), M(i-1, j+1)) \]
Dynamic Programming

\[ M(i, j) = E(i, j) + \min(M(i-1, j-1), M(i-1, j), M(i-1, j+1)) \]
Dynamic Programming

\[ M(i, j) = E(i, j) + \min(M(i - 1, j - 1), M(i - 1, j), M(i - 1, j + 1)) \]
Searching for Minimum

- Backtrack (can store choices along the path, but do not have to)
Backtracking the Seam

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>8</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>9</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
</tbody>
</table>

Slide by Michael Rubinstein
Backtracking the Seam

![Backtracking the Seam Diagram](image-url)
Backtracking the Seam

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>8</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>9</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
</tbody>
</table>

Slide by Michael Rubinstein
H & V Cost Maps

- Horizontal Cost
- Vertical Cost

Legend:
- High cost
- Low cost

Slide by Michael Rubinstein
Seams

Seams over energy image  Seams over input image
The Seam-Carving Algorithm

SEAM-CARVING(im,n’) // size(im)
1. Do (n-n’) times
   2.1. E ← Compute energy map on im
   2.2. s ← Find optimal seam in E
   2.3. im ← Remove s from im
2. Return im

• For vertical resize: transpose the image

• Running time:
  2.1 \(O(mn)\) 2.2 \(O(mn)\) 2.3 \(O(mn)\)
  \(\Rightarrow O(dmn)\) \(d=n-n’\)
Changing Aspect Ratio
Changing Aspect Ratio

Original

Seam Carving

Scaling
Changing Aspect ratio

Cropping

Seams

Scaling
Changing Aspect Ratio

Original                      Retarget               Scaling
Changing Aspect Ratio

Original

Retarget

Scaling
Image Expansion (Synthesis)
Image Expansion – Take 2

Scaling
Combined Insert and Remove

Insert & remove seams

Scaling
Content Amplification

- Scale the image, this will scale everything, content as well as non-content.
- Shrink the scaled-image by seam carving, this will carve out the non-content part.
Object Removal

- Remove consecutive vertical seams until no “green” pixels were left.
Object Removal
Find the Missing Shoe!
Solution
Limitations

Content

Structure
Old “Backward” Energy

\[ M(i, j) = E(i, j) + \min \left\{ \begin{array}{l}
M(i - 1, j - 1) \\
M(i - 1, j) \\
M(i - 1, j + 1)
\end{array} \right. \]
New Forward Looking Energy

\[ M(i, j) = \min \left\{ \begin{array}{l}
M(i - 1, j - 1) + C_L(i, j) \\
M(i - 1, j) + C_U(i, j) \\
M(i - 1, j + 1) + C_R(i, j)
\end{array} \right\} \]
Adding “Pixel Energy”

\[ M(i, j) = P(i, j) + \min \begin{cases} 
   M(i - 1, j - 1) + C_L(i, j) \\
   M(i - 1, j) + C_U(i, j) \\
   M(i - 1, j + 1) + C_R(i, j) 
\end{cases} \]
Results
Results
Backward vs. Forward

Backward

Forward
Video Resizing

- Resizing each frame independently is bad
- Instead, find 2D surface in 3D $x$-$y$-$t$ video volume
Programming Assignment 2

Due March 31, 2015
Today

• Image Resizing
  – Seam Carving

• Image Compositing and Blending
  – Alpha compositing
  – Laplacian pyramid blending
  – Poisson blending
Image Compositing

How do I put an object from one image into another?
News Composites

Original

“Enhanced” Version

http://www.guardian.co.uk/world/2010/sep/16/mubarak-doctored-red-carpet-picture
News Composites

Original

“Enhanced” Version

Walski, LA Times, 2003
Three methods

1. Cut and paste
2. Laplacian pyramid blending
3. Poisson blending
Method 1: Cut and Paste
Method 1: Cut and Paste

Method:
• Segment using intelligent scissors
• Paste foreground pixels onto target region
Method 1: Cut and Paste

Problems:
• Small segmentation errors noticeable
• Pixels are too blocky
• Won’t work for semi-transparent materials
Feathering

Near object boundary pixel values come partly from foreground and partly from background
Method 1: Cut and Paste (with feathering)
Output = foreground*mask + background*(1-mask)

Alpha compositing
Alpha compositing with feathering

Output = foreground*mask + background*(1-mask)
Another example (without feathering)

Mattes

Composite

Composite by David Dewey
Proper blending is key
Alpha Blending / Feathering

\[ I_{\text{blend}} = \alpha I_{\text{left}} + (1-\alpha)I_{\text{right}} \]
Setting alpha: simple averaging

Alpha = 0.5 in overlap region
Setting alpha: center seam

Distance Transform
bwdist

Alpha = logical(dtrans1 > dtrans2)
Setting alpha: blurred seam

Distance transform

Alpha = blurred
Setting alpha: center weighting

\[ \text{Alpha} = \frac{\text{dtrans1}}{\text{dtrans1} + \text{dtrans2}} \]
Affect of Window Size

slide by Alexei Efros
Affect of Window Size
Good Window Size

Optimal Window: smooth but not ghosted
How much should we blend?
Image Pyramids

Idea: Represent NxN image as a “pyramid” of 1x1, 2x2, 4x4,…, \(2^k \times 2^k\) images (assuming \(N=2^k\))

Known as a **Gaussian Pyramid** [Burt and Adelson, 1983]
- In computer graphics, a *mip map* [Williams, 1983]
A bar in the big images is a hair on the zebra’s nose; in smaller images, a stripe; in the smallest, the animal’s nose.

Gaussian Pyramid
Laplacian Pyramid

Lowpass Images
Laplacian Pyramid

Lowpass Images
Laplacian Pyramid

Lowpass Images
Laplacian Pyramid

Lowpass Images
Laplacian Pyramid

Lowpass Images

slide by Tamara Berg
Pyramid Blending

Left pyramid

blend

Right pyramid
Method 2: Laplacian Pyramid Blending

Implementation:

1. Build Laplacian pyramids for each image
2. Build a Gaussian pyramid of region mask
3. Blend each level of pyramid using region mask from the same level

\[ L_{12}^i = L_1^i \cdot R^i + L_2^i \cdot (1 - R^i) \]

- \( L_1^i \): Image 1 at level \( i \) of Laplacian pyramid
- \( R^i \): Region mask at level \( i \) of Gaussian pyramid

4. Collapse the pyramid to get the final blended image

Burt and Adelson 1983
Pyramid Blending

How this works? Blends high frequencies less and low frequencies more!
Blending Regions
Simplification: Two-band Blending

- Brown & Lowe, 2003
  - Only use two bands: high freq. and low freq.
  - Blends low freq. smoothly
2-band Blending

Low frequency

High frequency
Linear Blending
2-band Blending
Related idea: Poisson Blending

- A good blend should preserve gradients of source region without changing the background
Related idea: Poisson Blending

- A good blend should preserve gradients of source region without changing the background
Gradient Domain

• In Pyramid Blending, we decomposed our image into 2\textsuperscript{nd} derivatives (Laplacian) and a low-res image

• Let us now look at 1\textsuperscript{st} derivatives (gradients):
  – No need for low-res image
    • captures everything (up to a constant)
  – Idea:
    • Differentiate
    • Blend / edit / whatever
    • Reintegrate
Gradient Domain blending (1D)
Code for 1D gradient blending

1. mask = double(im1 > 15);
2.
3. im1 = 1:30; im2 = 5*sin(linspace(0,15,30));
4.
5. im1g = gradient(im1);
6. im2g = gradient(im2);
7.
8. imBlend = mask.*im1 + (1-mask).*im2;
9. imBlendGradient = mask.*im1g + (1-mask).*im2g;
10. imBlendGradient(1) = imBlend(1); % fix one value (should really fix both ends)
11. imBlendGradient = cumsum(imBlendGradient); % reintegrate
12.
13. figure(1);
14. subplot(2,2,1), plot(im1), title('signal one')
15. subplot(2,2,2), plot(im2), title('signal two')
16. subplot(2,2,3), plot(imBlend), title('direct blending')
17. subplot(2,2,4), plot(imBlendGradient), title('gradient blending')
Problems with direct copy/paste

From Perez et al. 2003
Solution: paste gradient

hacky visualization of gradient

seamless cloning
What is a gradient?

• derivative of a multivariate function
• for example, for \( f(x, y) \)

\[ \nabla f = \left( \frac{df}{dx}, \frac{df}{dy} \right) \]

• For a discrete image, can be approximated with finite differences

\[ \frac{df}{dx} \approx f(x + 1, y) - f(x, y) \]
\[ \frac{df}{dy} \approx f(x, y + 1) - f(x, y) \]
Gradient: intuition
Color images

- 3 gradients, one for each channel.
- We’ll sweep this under the rug for this lecture
- In practice, treat each channel independently
Seamless Poisson cloning

- Paste source gradient into target image inside a selected region
- Make the new gradient as close as possible to the source gradient while respecting pixel values at the boundary
Seamless Poisson cloning

**Given vector field** $\nu$ (pasted gradient), find the value of $f$ in unknown region that optimize:

$$\min_f \iint_\Omega |\nabla f - \nu|^2 \text{ with } f|_{\partial \Omega} = f^*|_{\partial \Omega}$$

Figure 1: **Guided interpolation notations.** Unknown function $f$ interpolates in domain $\Omega$ the destination function $f^*$, under guidance of vector field $\nu$, which might be or not the gradient field of a source function $g$. 
Discrete 1D example: minimization

- Copy to

**orange**: pixel outside the mask
**red**: source pixel to be pasted
**blue**: boundary conditions (in background)
Discrete 1D example: minimization

- Copy

\[
\text{Min } \left[ (f_2-f_1)-1 \right]^2 + \left[ (f_3-f_2)-(-1) \right]^2 + \left[ (f_4-f_3)-2 \right]^2 + \left[ (f_5-f_4)-(-1) \right]^2 + \left[ (f_6-f_5)-(-1) \right]^2
\]

With

\[f_1=6\] \[f_6=1\]
1D example: minimization

- Copy

\[ \text{Min } [(f_2-f_1)-1]^2 \quad \Rightarrow \quad f_2^2+49-14f_2 \]
\[ + [(f_3-f_2)-(-1)]^2 \quad \Rightarrow \quad f_3^2+f_2^2+1-2f_3f_2 +2f_3-2f_2 \]
\[ + [(f_4-f_3)-2]^2 \quad \Rightarrow \quad f_4^2+f_3^2+4-2f_3f_4 -4f_4+4f_3 \]
\[ + [(f_5-f_4)-(-1)]^2 \quad \Rightarrow \quad f_5^2+f_4^2+1-2f_5f_4 +2f_5-2f_4 \]
\[ + [(f_6-f_5)-(-1)]^2 \quad \Rightarrow \quad f_5^2+4-4f_5 \]
1D example: big quadratic

- Copy to

- Min \((f_2^2 + 49 - 14f_2 + f_3^2 + f_2^2 + 1 - 2f_3f_2 + 2f_3 - 2f_2 + f_4^2 + f_3^2 + 4 - 2f_3f_4 - 4f_4 + 4f_3 + f_5^2 + f_4^2 + 1 - 2f_5f_4 + 2f_5 - 2f_4 + f_5^2 + 4 - 4f_5)\)

Denote it \(Q\)
1D example: derivatives

- Copy

\[
\begin{align*}
\text{Min} (f_2^2 &+ 49 - 14 f_2 \\
+ f_3^2 &+ f_2^2 + 1 - 2 f_3 f_2 + 2 f_3 - 2 f_2 \\
+ f_4^2 &+ f_3^2 + 4 - 2 f_3 f_4 - 4 f_4 + 4 f_3 \\
+ f_5^2 &+ f_4^2 + 1 - 2 f_5 f_4 + 2 f_5 - 2 f_4 \\
+ f_5^2 &+ 4 - 4 f_5)
\end{align*}
\]

Denote it \( Q \)

\[
\begin{align*}
\frac{dQ}{df_2} &= 2 f_2 + 2 f_2 - 2 f_3 - 16 \\
\frac{dQ}{df_3} &= 2 f_3 - 2 f_2 + 2 + 2 f_3 - 2 f_4 + 4 \\
\frac{dQ}{df_4} &= 2 f_4 - 2 f_3 - 4 + 2 f_4 - 2 f_5 - 2 \\
\frac{dQ}{df_5} &= 2 f_5 - 2 f_4 + 2 + 2 f_5 - 4
\end{align*}
\]
1D example: set derivatives to zero

- Copy

\[
\frac{dQ}{df_2} = 2f_2 + 2f_2 - 2f_3 - 16 = 0
\]

\[
\frac{dQ}{df_3} = 2f_3 - 2f_2 + 2 + 2f_3 - 2f_4 + 4 = 0
\]

\[
\frac{dQ}{df_4} = 2f_4 - 2f_3 - 4 + 2f_4 - 2f_5 - 2 = 0
\]

\[
\frac{dQ}{df_5} = 2f_5 - 2f_4 + 2 + 2f_5 - 4 = 0
\]

\[
\begin{pmatrix}
4 & -2 & 0 & 0 \\
-2 & 4 & -2 & 0 \\
0 & -2 & 4 & -2 \\
0 & 0 & -2 & 4
\end{pmatrix}
\begin{pmatrix}
f_2 \\
f_3 \\
f_4 \\
f_5
\end{pmatrix}
= 
\begin{pmatrix}
16 \\
-6 \\
6 \\
2
\end{pmatrix}
\]
1D example recap

- Copy

\[
\begin{pmatrix}
4 & -2 & 0 & 0 \\
-2 & 4 & -2 & 0 \\
0 & -2 & 4 & -2 \\
0 & 0 & -2 & 4
\end{pmatrix}
\begin{pmatrix}
f_2 \\
f_3 \\
f_4 \\
f_5
\end{pmatrix}
= 
\begin{pmatrix}
16 \\
-6 \\
6 \\
2
\end{pmatrix}
\]

\[
\begin{pmatrix}
f_2 \\
f_3 \\
f_4 \\
f_5
\end{pmatrix}
= 
\begin{pmatrix}
6 \\
4 \\
5 \\
3
\end{pmatrix}
\]
In 2D

• More complex
• Of course, we do not need to worry about it, it’s all handled naturally by the least square approach
In 2D

- **Variational minimization** (integral of a functional) with boundary condition
  \[ \min_f \iint_{\Omega} |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial \Omega} = f^*|_{\partial \Omega}, \]

- **Euler-Lagrange equation:**
  \[ \Delta f = \text{div} \mathbf{v} \text{ over } \Omega, \text{ with } f|_{\partial \Omega} = f^*|_{\partial \Omega} \]
  where \( \text{div} \mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \) is the divergence of \( \mathbf{v} = (u, v) \)

- (Compared to Laplace, we have replaced \( \Delta = 0 \) by \( \Delta = \text{div} \))
Discrete Poisson solver

- Two approaches:
  - Minimize variational problem \( \min \int \int_{\Omega} |\nabla f - v|^2 \) with \( f|_{\partial \Omega} = f^*|_{\partial \Omega} \),
  - Solve Euler-Lagrange equation \( \Delta f = \text{div} v \) over \( \Omega \), with \( f|_{\partial \Omega} = f^*|_{\partial \Omega} \)

In practice, variational is best

- In both cases, need to discretize derivatives
  - Finite differences over 4 pixel neighbors
  - We are going to work using pairs
    - Partial derivatives are easy on pairs
    - Same for the discretization of \( v \)
Discrete Poisson solver

- Minimize variational problem

\[
\min_{f|\Omega} \sum_{\langle p,q \rangle \cap \Omega \neq \emptyset} (f_p - f_q - v_{pq})^2, \text{ with } f_p = f^*_p, \text{ for all } p \in \partial \Omega
\]

Discretized gradient

\[
\text{Boundary condition}
\]

• Rearrange and call \(N_p\) the neighbors of \(p\)

\[
|N_p| f_p - \sum_{q \in N_p \cap \Omega} f_q = \sum_{q \in N_p \cap \partial \Omega} f_q^* + \sum_{q \in N_p} v_{pq}
\]

• Big yet sparse linear system

Only for boundary pixels

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>p</th>
<th>q</th>
<th></th>
</tr>
</thead>
</table>
Result (eye candy)

[Image of a face with yellow outlines indicating source/destination, cloning, and seamless cloning]
What do we lose?

- Foreground color changes
- Background pixels in target region are replaced
Blending with Mixed Gradients

- Use foreground or background gradient with larger magnitude as the guiding gradient

since the source has no texture, it looks flat

mix the gradients from the source and background
Blending with Mixed Gradients

source

destination

Transparent Objects
Summary

• Three ways to blend/composite
  1. Alpha compositing
     • Need nice cut (intelligent scissors)
     • Should **feather**
  2. Laplacian pyramid blending
     • **Smooth blending at low frequencies, sharp at high frequencies**
     • Usually used for stitching
  3. Gradient domain editing
     • Also called **Poisson Editing**
     • Explicit control over what to preserve
     • Changes foreground color (for better or worse)
     • Applicable for many things besides blending