BBM 444 – Week 4
Image Blending, Compositing, Resizing

Aykut Erdem
Computer Vision Lab (CVL)
Hacettepe University
Announcements

- Your 2\textsuperscript{nd} programming assignment will be announced shortly!
Today’s Agenda: Image Blending

How do I put an object from one image into another?
Today’s Agenda: Finding Seams and Boundaries

Segmentation

Slide credit: Derek Hoiem
Today’s Agenda: Finding Seams and Boundaries

Retargeting

http://swieskowski.net/carve/

Slide credit: Derek Hoiem
Image Compositing

Slide credit: Alexei Efros
Compositing Procedure

1. Extract Sprites (e.g. using *Intelligent Scissors* in Photoshop)

2. Blend them into the composite (in the right order)

Composite by David Dewey

Slide credit: Alexei Efros
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)
Alpha compositing

Output = foreground * mask + background * (1-mask)

Slide credit: Derek Hoiem
Alpha compositing with feathering

Output = foreground*mask + background*(1-mask)
Proper blending is key
Alpha Blending / Feathering

$$I_{blend} = \alpha I_{left} + (1-\alpha)I_{right}$$
Alpha Blending / Feathering

\[ I_{\text{blend}} = \alpha I_{\text{left}} + (1-\alpha)I_{\text{right}} \]
Affect of Window Size

Large windows -> lots of ghosting

Slide credit: Alexei Efros
Affect of Window Size

Small window -> seams are too obvious

Slide credit: Alexei Efros
Good Window Size

“Optimal” Window: smooth but not ghosted

Medium window -> just right!

Slide credit: Alexei Efros
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]

Slide credit: Tamara Berg
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

Figure from David Forsyth
Laplacian Pyramid

Lowpass Images
Laplacian Pyramid

Lowpass Images
Laplacian Pyramid

Lowpass Images
Laplacian Pyramid

Lowpass Images

Slide credit: Tamara Berg
Laplacian Pyramid

Lowpass Images
Pyramid Blending

- At low frequencies, blend slowly
- At high frequencies, blend quickly

Slide credit: Alexei Efros
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

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

4. Collapse the pyramid to get the final blended image

Slide credit: Tamara Berg

Burt and Adelson 1983
Laplacian level 4

Laplacian level 2

Laplacian level 0

left pyramid

right pyramid

blended pyramid
Pyramid Blending

Slide credit: Alexei Efros
Blending Regions
Season Blending (St. Petersburg)
Season Blending (St. Petersburg)
Simplification: Two-band Blending

• Brown & Lowe, 2003
  – Only use two bands: high freq. and low freq.
  – Blends low freq. smoothly
  – Blend high freq. with no smoothing: use binary alpha
Simplification: Two-band Blending

- Brown & Lowe, 2003
  - Only use two bands: high freq. and low freq.
  - Blends low freq. smoothly
  - Blend high freq. with no smoothing: use binary alpha
2-band Blending

Low frequency

High frequency

Slide credit: Alexei Efros
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

Slide credit: Derek Hoiem

Perez et al. 2003
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)

- Signal One
- Signal Two
- Direct Blending
- Gradient Blending

Slide credit: Nathan Jacobs
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')

Slide credit: Nathan Jacobs
Gradient Domain Blending (2D)

- Trickier in 2D:
  - Take partial derivatives $dx$ and $dy$ (the gradient field)
  - Fiddle around with them (smooth, blend, feather, etc.)
  - Reintegrate
    - But now integral($dx$) might not equal integral($dy$)
  - Find the most agreeable solution
    - Equivalent to solving Poisson equation
    - Can be done using least-squares

Slide credit: Alexei Efros
Gradient Domain Editing

• General concept: Solve for pixels of new image that satisfy constraints on the gradient and the intensity
  – Constraints can be from one image (for filtering) or more (for blending)
Poisson Blending

• A good blend should preserve gradients of source region without changing the background

• Treat pixels as variables to be solved
  – Minimize squared difference between gradients of foreground region and gradients of target region
  – Keep background pixels constant

$$v = \arg\min_v \sum_{i \in S, j \in N_i \cap S} ((v_i - v_j) - (s_i - s_j))^2 + \sum_{i \in S, j \in N_i \cap -S} ((v_i - t_j) - (s_i - s_j))^2$$

s: source image
t: target image
v: blended image
S: source region
$N_i$: 4-neighborhood of i

[Perez et al. 2003]
Example

Gradient Visualization

Slide credit: Derek Hoiem

Source: Evan Wallace
Other results

Slide credit: Derek Hoiem

[Perez et al., 2003] 52
What do we lose?

- Foreground color changes
- Background pixels in target region are replaced

Slide credit: Derek Hoiem
Blending with Mixed Gradients

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

Figure 3: Insertion. The power of the method is fully expressed when inserting objects with complex outlines into a new background. Because of the drastic differences between the source and the destination, standard image cloning cannot be used in this case.

Figure 4: Feature exchange. Seamless cloning allows the user to replace easily certain features of one object by alternative features. In the second example of texture swapping multiple broad strokes (not shown) were used.

Figure 5: Monochrome transfer. In some cases, such as texture transfer, the part of the source color remaining after seamless cloning might be undesirable. This is fixed by turning the source image monochrome beforehand.

Figure 6: Inserting objects with holes. (a) The classic method, color-based selection and alpha masking might be time consuming and often leaves an undesirable halo; (b-c) seamless cloning, even averaged with the original image, is not effective; (d) mixed seamless cloning based on a loose selection proves effective.

Figure 7: Inserting transparent objects. Mixed seamless cloning facilitates the transfer of partly transparent objects, such as the rainbow in this example. The non-linear mixing of gradient fields picks out whichever of source or destination structure is the more salient at each location.

This mixed seamless cloning is also useful when adding one object from a source image very close to another object in the destination image, see Fig. 8.

[Note: Textual explanations are omitted for brevity.]

[Slide credit: Derek Hoiem]
Blending with Mixed Gradients

Transparent Objects

Slide credit: Tamara Berg
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
Compositing with moving objects

Moving objects become ghosts

• So far we only tried to blend between two images. What about finding an optimal seam?

Slide credit: Alexei Efros
Cutting & Stitching

- Segment the mosaic
  - Single source image per segment
  - Avoid artifacts along boundaries
    - Dijkstra’s algorithm
Cutting: Finding Seams and Boundaries

Segmentation

Slide credit: Derek Hoiem
Cutting: Finding Seams and Boundaries

compositing from different scenes

Slide credit: Derek Hoiem
Cutting: Finding Seams and Boundaries

- Fundamental Concept: The Image as a Graph
  - Intelligent Scissors: Good boundary = short path
  - Graph cuts: Good region has low cutting cost
Semi-automated segmentation

User provides imprecise and incomplete specification of region – your algorithm has to read his/her mind.

Key problems
1. What groups of pixels form cohesive regions?
2. What pixels are likely to be on the boundary of regions?
3. Which region is the user trying to select?
What makes a good region?

- Contains small range of color/texture
- Looks different than background
- Compact
What makes a good boundary?

• High gradient along boundary
• Gradient in right direction
• Smooth
The Image as a Graph

Node: pixel

Edge: cost of path or cut between two pixels

Slide credit: Derek Hoiem
Intelligent Scissors

Mortenson and Barrett (SIGGRAPH 1995)
Intelligent Scissors
Mortenson and Barrett (SIGGRAPH 1995)

A good image boundary has a short path through the graph.
Intelligent Scissors

• Formulation: find good boundary between seed points

• Challenges
  – Minimize interaction time
  – Define what makes a good boundary
  – Efficiently find it
Intelligent Scissors: method

1. Define boundary cost between neighboring pixels
2. User specifies a starting point (seed)
3. Compute lowest cost from seed to each other pixel
4. Get path from seed to cursor, choose new seed, repeat
Intelligent Scissors: method

1. Define boundary cost between neighboring pixels
   a) Lower if edge is present (e.g., with edge(im, ‘canny’))
   b) Lower if gradient is strong
   c) Lower if gradient is in direction of boundary
Gradients, Edges, and Path Cost

Gradient Magnitude

Edge Image

Path Cost

Slide credit: Derek Hoiem
Intelligent Scissors: method

1. Define boundary cost between neighboring pixels
2. User specifies a starting point (seed)
   – Snapping
Intelligent Scissors: method

1. Define boundary cost between neighboring pixels
2. User specifies a starting point (seed)
3. Compute lowest cost from seed to each other pixel
   - Djikstra’s shortest path algorithm
Djikstra’s shortest path algorithm

Initialize, given seed $s$:

- Compute $\text{cost}_2(q, r)$ % cost for boundary from pixel $q$ to neighboring pixel $r$
- $\text{cost}(s) = 0$ % total cost from seed to this point
- $A = \{s\}$ % set to be expanded
- $E = \emptyset$ % set of expanded pixels
- $P(q)$ % pointer to pixel that leads to $q$

Loop while $A$ is not empty

1. $q =$ pixel in $A$ with lowest cost
2. Add $q$ to $E$
3. for each pixel $r$ in neighborhood of $q$ that is not in $E$
   a) $\text{cost} \_\text{tmp} = \text{cost}(q) + \text{cost}_2(q,r)$
   b) if ($r$ is not in $A$) OR ($\text{cost} \_\text{tmp} < \text{cost}(r)$)
      i. $\text{cost}(r) = \text{cost} \_\text{tmp}$
      ii. $P(r) = q$
      iii. Add $r$ to $A$

Slide credit: Derek Hoiem
Intelligent Scissors: method

1. Define boundary cost between neighboring pixels
2. User specifies a starting point (seed)
3. Compute lowest cost from seed to each other pixel
4. Get new seed, get path between seeds, repeat
Intelligent Scissors: improving interaction

1. Snap when placing first seed
2. Automatically adjust to boundary as user drags
3. Freeze stable boundary points to make new seeds

Slide credit: Derek Hoiem
Where will intelligent scissors work well, or have problems?

Slide credit: Derek Hoiem
Grab cuts and graph cuts

User Input

Result

Magic Wand
(198?)

Intelligent Scissors
Mortensen and Barrett (1995)

GrabCut

Regions

Boundary

Regions & Boundary

Source: Rother

Slide credit: Derek Hoiem
Segmentation with graph cuts

\[ \text{Energy}(y; \theta, \text{data}) = \sum_i \psi_1(y_i; \theta, \text{data}) \sum_{i,j \text{ edges}} \psi_2(y_i, y_j; \theta, \text{data}) \]
Segmentation with graph cuts

\[ \text{Energy}(y; \theta, \text{data}) = \sum_i \psi_1(y_i; \theta, \text{data}) \sum_{i, j \text{ edges}} \psi_2(y_i, y_j; \theta, \text{data}) \]
Colour Model

Gaussian Mixture Model (typically 5-8 components)
Graph cuts

Boykov and Jolly (2001)

**Cut:** separating source and sink; **Energy:** collection of edges

**Min Cut:** Global minimal energy in polynomial time

Slide credit: Derek Hoiem
Graph cuts segmentation

1. Define graph
   - usually 4-connected or 8-connected

2. Set weights to foreground/background
   - Color histogram or mixture of Gaussians for background and foreground
   \[
   \text{unary\_potential}(x) = -\log \left( \frac{P(c(x); \theta_{\text{foreground}})}{P(c(x); \theta_{\text{background}})} \right)
   \]

3. Set weights for edges between pixels
   \[
   \text{edge\_potential}(x, y) = k_1 + k_2 \exp \left( -\frac{\|c(x) - c(y)\|^2}{2\sigma^2} \right)
   \]

4. Apply min-cut/max-flow algorithm

5. Return to 2, using current labels to compute foreground, background models

Slide credit: Derek Hoiem
What is easy or hard about these cases for graphcut-based segmentation?
Easier examples
More difficult Examples

Camouflage & Low Contrast

Initial Rectangle

Initial Result

Fine structure

Harder Case

Slide credit: Derek Hoiem
Lazy Snapping [Li et al., 2004]
Limitations of Graph Cuts

• Requires associative graphs
  – Connected nodes should prefer to have the same label

• Is optimal only for binary problems
Other applications: stitching

Graphcut Textures – Kwatra et al. SIGGRAPH 2003

Ideal boundary:
1. Similar color in both images
2. High gradient in both images
Other applications: stitching

Graphcut Textures – Kwatra et al. SIGGRAPH 2003

Slide credit: Derek Hoiem
Putting it all together

• Compositing images
  – Have a clever blending function
    • Feathering
    • Center-weighted
    • blend different frequencies differently
  
  – Choose the right pixels from each image
    • Graph-cuts
Interactive Digital Photomontage

[Agarwala et al., 2004]

Slide credit: Derek Hoiem
Challenges

• Find good seams between parts of images so they can be joined with few visible artifacts

• Blend along seams to reduce or remove any artifacts remaining after joining
Image Objectives

• User specified global image objectives for composite:
  – Designated color
  – Luminance
  – Contrast
  – Eraser (color most different from current composite)
  – Source Image,
  – ...

* Or Brush interface
Seam Objectives

• Measures suitability of a seam between two image regions
  – Match colors across seams
  – Match colors and color gradients across seams
  – Match colors across seams, but prefer seams that lie along edges
Graph Cut

\[ C(L) = \sum_p C_d(p, L(p)) + \sum_{p,q} C_i(p, q, L(p), L(q)) \]

Labeling \( L(p) \) gives the source image for each pixel \( p \).

Cost of a labeling \( (L) \) is data penalty \( (C_d) \) and interaction penalty \( (C_i) \).
- Data penalty - distance to image objective
- Interaction penalty - distance to seam objective.

**Want to minimize the cost function so that seams are minimized and image objectives are met as well as possible.**

Slide credit: Derek Hoiem
\[ C_D = \sum_x D(x, l(x)) \quad \text{data penalty associated with choosing image } l \text{ at pixel } x \]

\[ C_S = \sum_{(x,y) \in N} S(x, y, l(x), l(y)) \]

\[ S(x, y, l_x, l_y) = \| \tilde{l}_x(x) - \tilde{l}_y(x) \| + \| \tilde{l}_x(y) - \tilde{l}_y(y) \| \]

\text{seam objective that penalizes differences in labelings between adjacent images}
User Selection
Result of Pasting
Gradient Smoothing

• Remove any remaining artifacts after merging source image parts
• View input images as sources of gradient information rather than color.
• Using graph cut labeling, form composite whose gradients best match source gradient vectors.
Final Result after smoothing
Extended Depth of Field

Input Images

Slide credit: Derek Hoiem
Source Map
Extended Depth of Field

Slide credit: Derek Hoiem
Relighting

Input Images
Source Map

Slide credit: Derek Hoiem
Relighted Result

Slide credit: Derek Hoiem
Visualizing Movement

Slide credit: Derek Hoiem
Visualizing Movement

Slide credit: Derek Hoiem
Selective Composites

Slide credit: Derek Hoiem
Selective Composites

Slide credit: Derek Hoiem
Selective Composites
Selective Composites
Background reconstruction

Slide credit: Derek Hoiem
Background reconstruction
Relighting

Slide credit: Derek Hoiem
Compositing and Obstruction removal
Fictional Composites

Slide credit: Derek Hoiem
Interactive Digital Photomontage

Aseem Agarwala, Mira Dontcheva
Maneesh Agrawala, Steven Drucker, Alex Colburn
Brian Curless, David Salesin, Michael Cohen

Slide credit: Derek Hoiem
Summary

• Treat image as a graph
  – Pixels are nodes
  – Between-pixel edge weights based on gradients
  – Sometimes per-pixel weights for affinity to foreground/background

• Good boundaries are a short path through the graph (Intelligent Scissors, Seam Carving)

• Good regions are produced by a low-cost cut (GrabCuts, Graph Cut Stitching)
Image Resizing

Retargeting

http://swieskowski.net/carve/

Slide credit: Derek Hoiem
Display Devices
Content Retargeting
Simple Media Retargeting
Operators

Scaling
Letterboxing

Slide credit: James Hays
Content-aware Retargeting Operators

Content-aware

“Important” content

Content-oblivious
Content-aware Retargeting

Input

Scale

Crop

Content-aware

“less-Important”
content

Slide credit: James Hays
Image Retargeting

• Problem statement:
  – Input Image $nxm$, and new size $n'xm'$
  – Output Image of size $n'xm'$ which will be “good representative” of the original image

• To date, no agreed definition, or measure, as to what a good representative is in this context!
Image/Video Retargeting

In large, we would expect:
1. Adhere to the geometric constraints (display/aspect ratio)
2. Preserve the important content and structures
3. Limit artifacts
4. Perhaps a new representation that will support different sizes?

• Very ill-posed!
  – How do we define important? Is there a universal ground truth?
  – Would different viewers think the same about a retargeted image?
  – What about artistic impression in the original content?
Importance (Saliency) Measures

• A function $S: \mathbb{R} \to [0,1]$

Judd et al. ICCV09 *Learning to predict where people look*

Wang et al. 2008
Importance (Saliency) Measures

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

See their paper for more measures they have used.
Pixel Removal

Optimal (global)

Least-energy pixels (per row)

Least-energy columns

Slide credit: James Hays
A Seam

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

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

$$s^y = \{s^y_j\}_{j=1}^m = \{(j, y(j))\}_{j=1}^m, \text{ s.t. } \forall j |y(j) - y(j - 1)| \leq 1$$
Seams in Images

- Efros & Freeman [2001] – Texture synthesis
- Kwatra et al. [2003] – Image and video synthesis
- Agarwala et al. [2004] – Digital Photomontage
- Perez et al. [2002] – Poisson Image Editing
- Jia et al. [2006] – Drag-and-Drop Pasting
- Rother et al. [2006] – Auto-Collage

Mostly used for composition of two (or more) images or patches…
Finding the Seam?
The Optimal Seam

\[ E(I) = \left| \frac{\partial}{\partial x} I \right| + \left| \frac{\partial}{\partial y} I \right| \Rightarrow s^* = \arg \min_s E(s) \]
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 \( O(s \cdot n \cdot m) \)

\( s = 3 \) in the original algorithm

Slide credit: James Hays
H & V Cost Maps

Horizontal Cost

Vertical Cost

Slide credit: James Hays
Seam Carving
The Seam-Carving Algorithm

SEAM-CARVING(im, n') // size(im) = mxn
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)
  \( \Rightarrow O(dmn) \quad d=n-n' \)
Changing Aspect Ratio
Changing Aspect Ratio

Original

Seam Carving

Scaling

Slide credit: James Hays
Changing Aspect ratio

Cropping  Seams  Scaling

Slide credit: James Hays
Changing Aspect Ratio

Original

Retarget

Scaling

Slide credit: James Hays
Changing Aspect Ratio

Original

Retarget

Scaling

Slide credit: James Hays
Seam Carving in the Gradient Domain

- Combine seam carving with Poisson reconstruction [Perez et al. 2003]

Slide credit: James Hays
Combined Insert and Remove

Insert & remove seams

Scaling

Slide credit: James Hays
• Remove consecutive vertical seams until no “green” pixels were left.
Object Removal

Input  Retargeted  Pigeon Removed  Girl Removed
Limitations

Content

Structure

Slide credit: James Hays
Old “Backward” Energy

\[ M(i, j) = E(i, j) + \min \left\{ M(i - 1, j - 1), M(i - 1, j), M(i - 1, j + 1) \right\} \]
New **Forward Looking Energy**

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

\[ M(i, j) = P(i, j) + \min \left\{ M(i - 1, j - 1) + C_L(i, j), M(i - 1, j) + C_U(i, j), M(i - 1, j + 1) + C_R(i, j) \right\} \]
Results

Input  Backward  Forward

Backward

Input  Forward

Slide credit: James Hays
Backward vs. Forward

Backward

Forward

Slide credit: James Hays
Results
Reading Assignment


  [http://www.faculty.idc.ac.il/arik/SCWeb/imret/](http://www.faculty.idc.ac.il/arik/SCWeb/imret/)

Programming Assignment #2

• Image Resizing by Seam Carving