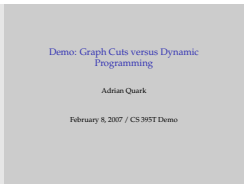
Demo: Graph Cuts versus Dynamic Programming

Adrian Quark

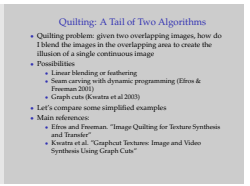
February 8, 2007 / CS 395T Demo



1. Note pages are interleaved with slides. These notes cover some of the verbal content of the talk.

Quilting: A Tail of Two Algorithms

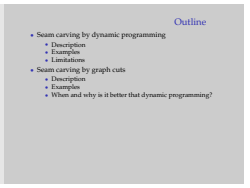
- Quilting problem: given two overlapping images, how do I blend the images in the overlapping area to create the illusion of a single continuous image
- Possibilities
  - Linear blending or feathering
  - Seam carving with dynamic programming (Efros & Freeman 2001)
  - Graph cuts (Kwatra et al 2003)
- Let's compare some simplified examples
- Main references:
  - Efros and Freeman. "Image Quilting for Texture Synthesis and Transfer"
  - Kwatra et al. "Graphcut Textures: Image and Video Synthesis Using Graph Cuts"



1. Feathering: often used in panoramic stitching. May provide adequate results if the patches are well-matched, but it creates blur if high-frequency areas of patches don't match. Not suitable for quilting.
2. I'll ignore more complicated solutions like warping or sampling; we'll assume pixels are fixed and all we get to do is choose which patch owns each pixel.
3. Use contrived "toy" examples of patches only a few pixels wide. This will make it easier to illustrate strengths and weaknesses of these algorithms. I'll try to make it clear how these extrapolate to real images.
4. I'll ignore the problem of positioning patches. The choice of overlap may make the seam-finding problem really easy or effectively impossible. But I'll just assume we've found the best possible overlap already.

Outline

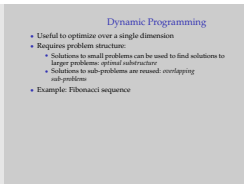
- Seam carving by dynamic programming
  - Description
  - Examples
  - Limitations
- Seam carving by graph cuts
  - Description
  - Examples
  - When and why is it better that dynamic programming?



1. This talk will cover both algorithms and then compare them.

Dynamic Programming

- Useful to optimize over a single dimension
- Requires problem structure:
  - Solutions to small problems can be used to find solutions to larger problems: *optimal substructure*
  - Solutions to sub-problems are reused: *overlapping sub-problems*
- Example: Fibonacci sequence



1. Dynamic programming = recursion + memoization
2. Optimal substructure: new Fibonacci numbers are based on previous ones.
3. Overlapping subproblems: each Fibonacci number is used twice (to compute its successor and its successor's successor).

Seam Carving

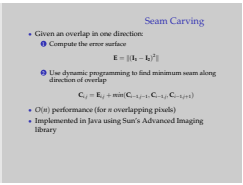
- Given an overlap in one direction:
  1. Compute the error surface

$$\mathbf{E} = \|(\mathbf{I_1} - \mathbf{I_2})^2\|$$

  2. Use dynamic programming to find minimum seam along direction of overlap

$$\mathbf{C}_{i,j} = \mathbf{E}_{i,j} + min(\mathbf{C}_{i-1,j-1}, \mathbf{C}_{i-1,j}, \mathbf{C}_{i-1,j+1})$$

- $O(n)$ performance (for n overlapping pixels)
- Implemented in Java using Sun's Advanced Imaging library



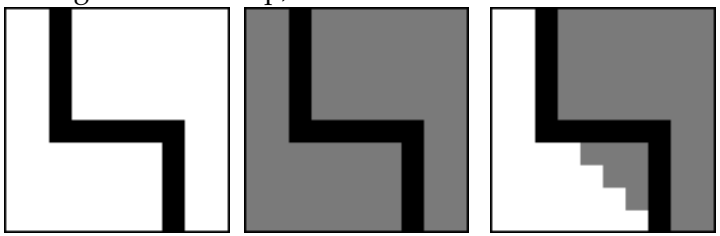
1. This algorithm is described in Efros and Freeman.
2. The algorithm used in Avidan and Shamir is of course quite similar but uses a different error function appropriate to their domain.
3. Assume: somebody else already found a good overlap for us. Overlap must be rectangular and go from one side of the patch to the other. This is obviously a serious limitation if you want to fit patches into irregular spaces.
4. For any pixel, the error is the amount the two images disagree on the value of that pixel. Choosing a path which minimizes error means finding a path where the disagreement on pixels is lowest, on other words the two images are most similar.
5. Working from one end of the overlap to the other, for each pixel compute the minimum cost of all the possible paths ending at that pixel. After doing this for the whole image we can pick the minimum-cost path ending in the last row, and then work backwards to enumerate it.
6. The implementation is simple and requires just two passes over the overlapping area.

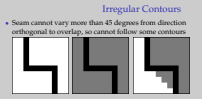
Quilting Example

1. Illustrate dynamic programming applied to quilting two brick images.
2. NEXT: limitations

Irregular Contours

- Seam cannot vary more than 45 degrees from direction orthogonal to overlap, so cannot follow some contours

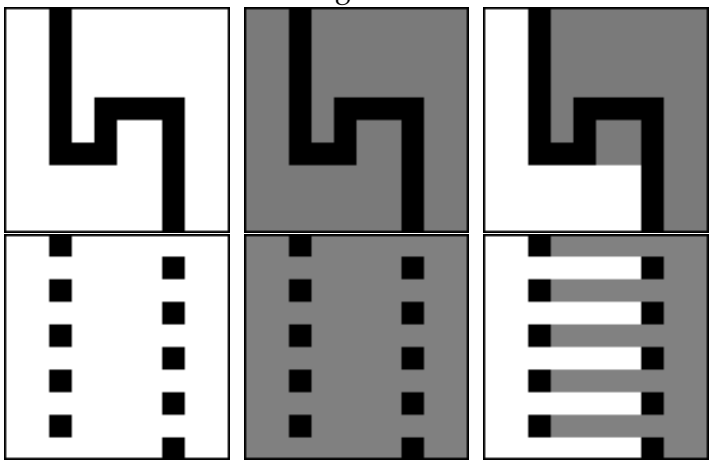


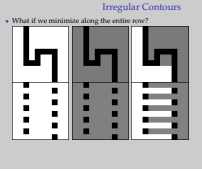


1. The dynamic programming algorithm only minimizes along one dimension, so it can't decide to turn to follow a minimum-error surface orthogonal to the seam.
2. These two images are each 10 pixels square, and we'd like to overlap them completely and combine them, respecting image boundaries. You can imagine that these are white and grey objects, and the black line is a boundary between two objects.
3. The ideal solution respects the object boundaries.
4. Because our dynamic programming algorithm is unable to "turn the corner" we get something less nice.

Irregular Contours

- What if we minimize along the entire row?

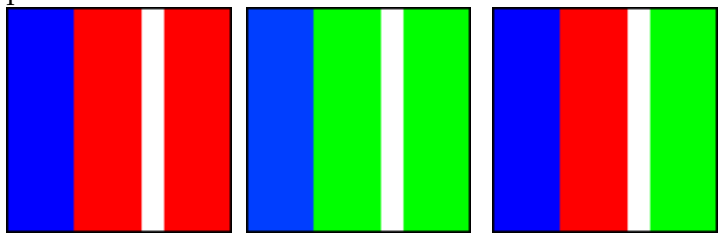


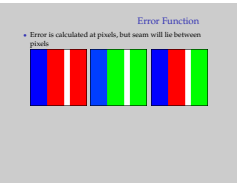


1. We allow the path to jump as far left or right as is necessary to move to the next good pixel. That would work fine for our above example.
2. But in general this doesn't work because it doesn't account for the fact that we're implicitly creating a horizontal seam, and we need to make sure the horizontal seam isn't visible either.
3. NEXT: another limitation

Error Function

- Error is calculated at pixels, but seam will lie between pixels



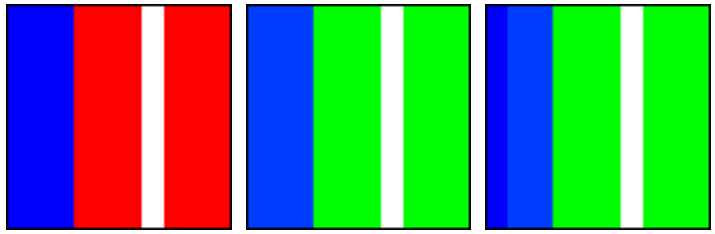


1. We're minimizing the difference at a given pixel, but our seam will actually lie to one side or the other of that pixel. This is annoying theoretically but not usually a problem in practice.
2. Two 10-pixel square images again. Let's call them red and green objects against a blue background. As before, we will overlap these completely and would like our seam to respect object boundaries.
3. We'd like to get the green object against the darker blue background.
4. But because the narrow white line on the objects matches slightly better than the background, our algorithm decides to put the seam there.
5. In real images, this means the algorithm is slightly more "local" than it should be. We'll see this in a more realistic example later.

Error Function

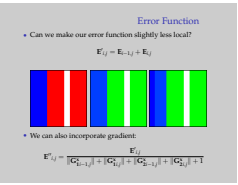
- Can we make our error function slightly less local?

$$\mathbf{E}'_{i,j} = \mathbf{E}_{i-1,j} + \mathbf{E}_{i,j}$$



- We can also incorporate gradient:

$$\mathbf{E}''_{i,j} = \frac{\mathbf{E}'_{i,j}}{\|\mathbf{G}^{\mathbf{x}}_{\mathbf{1}i-1,j}\| + \|\mathbf{G}^{\mathbf{x}}_{\mathbf{1}i,j}\| + \|\mathbf{G}^{\mathbf{x}}_{\mathbf{2}i-1,j}\| + \|\mathbf{G}^{\mathbf{x}}_{\mathbf{2}i,j}\| + 1}$$



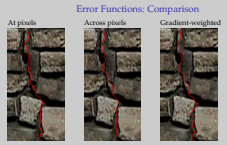
1. We can correct this problem by using an error function which takes into account the neighborhood of the seam rather than just one side of it.
2. This equation says that the error for a seam to the *left* of a pixel is the sum of the error on both sides of that seam. For other kinds of seams or overlap we'd need different functions.
3. With the same example as before, we get a "better" result.
4. NEXT: comparing these error functions

Error Functions: Comparison

At pixels

Across pixels

Gradient-weighted



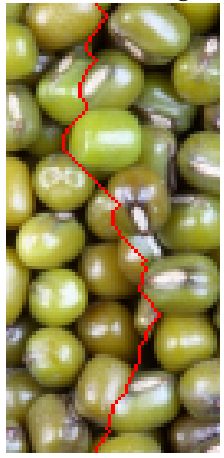
1. Averaging error across seam does seem to improve results.
2. DISCUSS: can we use an error function which accounts for error in more than one direction (not just left-right but up-down)

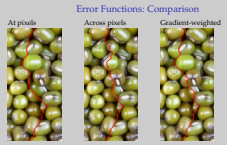
Error Functions: Comparison

At pixels

Across pixels

Gradient-weighted



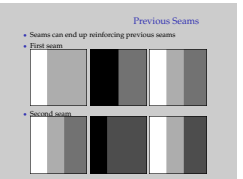


1. Gradient-weighting improves the results slightly by encouraging the seam to follow object boundaries where appropriate. Notice that the simple error function does this naturally but has other problems.
2. NEXT: another limitation

Previous Seams

- Seams can end up reinforcing previous seams
- First seam

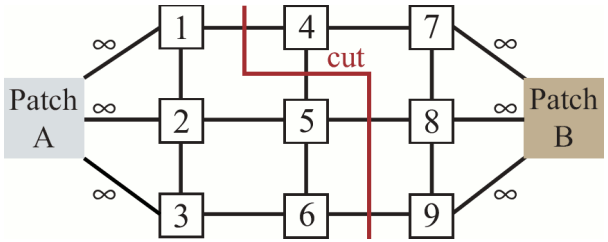
- Second seam

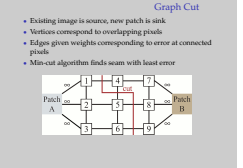


1. When new seams are chosen assuming that old seams are part of the original image, you can get "drift" which reinforces the old seams.
2. Not clear how to fix this in a dynamic programming algorithm.
3. Probably not an issue in practice if old-seam overlaps are limited and regular, as in Efros & Freeman
4. NEXT: graph cuts

Graph Cut

- Existing image is source, new patch is sink
- Vertices correspond to overlapping pixels
- Edges given weights corresponding to error at connected pixels
- Min-cut algorithm finds seam with least error





1. Translating our image problem into a graph problem lets us make use of well-understood algorithms.

Min-cut Algorithms

- *The maximum amount of flow is equal to the capacity of a minimal cut.* (Elias, Feinstein & Shannon 1956)
- "Augmenting flow"

1956	Ford & Fulkerson	$O(E \cdot maxflow)$
1972	Edmonds & Karp	$O(VE^2)$
2004	Boykov & Kolmogorov	$O(VE^2 \cdot mincut)$

- "Push-relabel"

1970	Dinic	$O(V^2E)$
1980	Sleator & Tarjan	$O(VE\log(V))$
1988	Goldberg & Tarjan	$O(VE\log(\frac{V^2}{E}))$

- I used Edmonds & Karp's $O(n^3)$ algorithm as implemented by JUNG (Java Universal Network/Graph Framework)

Min-cut Algorithms
- *The maximum amount of flow is equal to the capacity of a minimal cut.* (Elias, Feinstein & Shannon 1956)
- "Augmenting flow"
  1956 Ford & Fulkerson $O(E \cdot maxflow)$
  1972 Edmonds & Karp $O(VE^2)$
  2004 Boykov & Kolmogorov $O(VE^2 \cdot mincut)$
- "Push-relabel"
  1970 Dinic $O(V^2E)$
  1980 Sleator & Tarjan $O(VE\log(V))$
  1988 Goldberg & Tarjan $O(VE\log(\frac{V^2}{E}))$
- I used Edmonds & Karp's $O(n^3)$ algorithm as implemented by JUNG (Java Universal Network/Graph Framework)

1. The max-flow min-cut theorem intuitively means that the capacity of a network is limited by its bottlenecks.
2. Augmenting flow: find a path with available capacity and send flow along it, until there are no more. Improvements come from finding augmenting paths more efficiently.
3. Push-relabel: each node gets a height: push sends water from high ground to low, relabel raises the height of a node. Repeat as necessary. Improvements come from changing the rules for when to relabel and using better data structures.

Other Applications

- Generally useful for energy minimization
- First applied to computer vision (image restoration) by Greig et al 1989
- Also applied to:
  - Segmentation
  - Stereo correspondence
  - Object recognition
  - Shape reconstruction
  - Augmented reality
  - Texture generation (of course)
  - Video textures (of course)

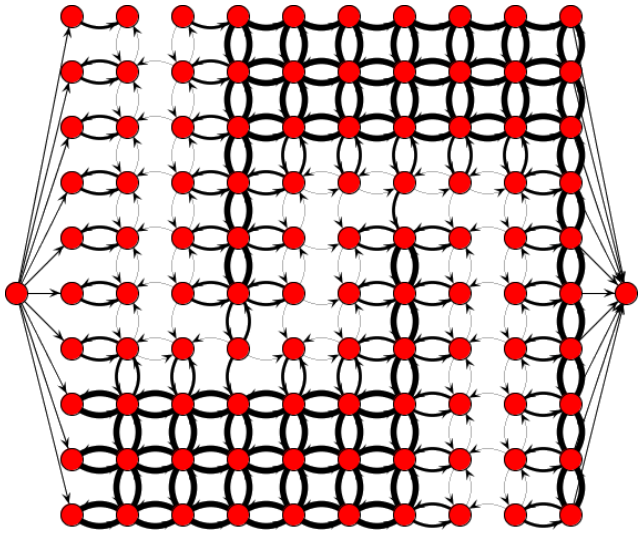
(Boykov & Kolmogorov 2004)

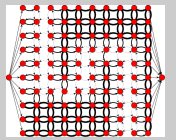
Other Applications
- Generally useful for energy minimization
- First applied to computer vision (image restoration) by Greig et al 1989
- Also applied to:
  - Segmentation
  - Stereo correspondence
  - Object recognition
  - Shape reconstruction
  - Augmented reality
  - Texture generation (of course)
  - Video textures (of course)

(Boykov & Kolmogorov 2004)

1. The ability of graph cuts to handle graphs with arbitrary structure, not just 2D image grids, makes them very powerful.

Example

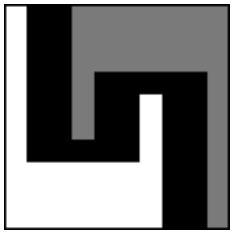




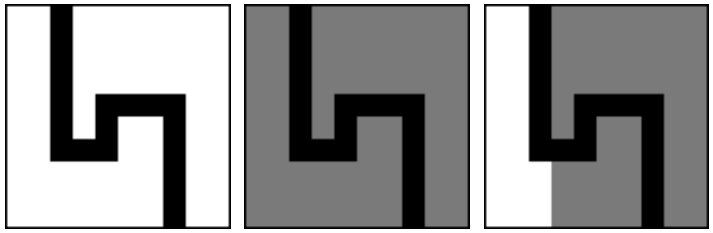
1. Two 10-pixel images will be completely overlapped.
2. A graph is constructed for the overlapping area, with edge weights corresponding to error across a seam cutting that edge.
3. The graph is cut. Graph cut algorithms are for directional graphs so we introduce reverse edges. Only one edge will be cut (the one pointing towards the new image).

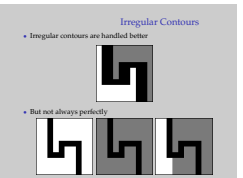
Irregular Contours

- Irregular contours are handled better



- But not always perfectly

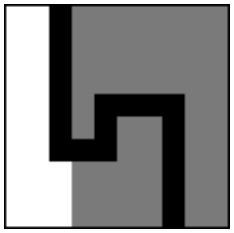




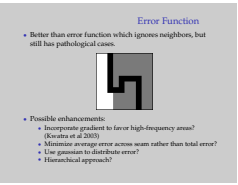
1. Because we consider total cost of the cut, not average cost, the algorithm prefers a shorter cut with some high-error segments to a longer one with uniform error.
2. Real images tend to be more continuous so this may not matter in practice.

Error Function

- Better than error function which ignores neighbors, but still has pathological cases.



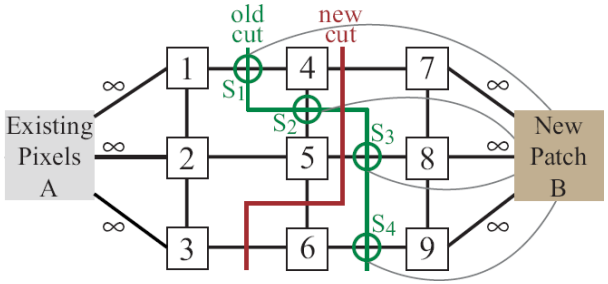
- Possible enhancements:
  - Incorporate gradient to favor high-frequency areas? (Kwatra et al 2003)
  - Minimize average error across seam rather than total error?
  - Use gaussian to distribute error?
  - Hierarchical approach?

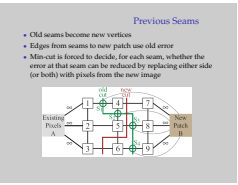


1. DISCUSS: what makes a good error function for quilting?
2. DISCUSS: how can we improve the min-cut error function

Previous Seams

- Old seams become new vertices
- Edges from seams to new patch use old error
- Min-cut is forced to decide, for each seam, whether the error at that seam can be reduced by replacing either side (or both) with pixels from the new image





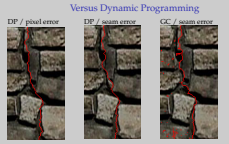
1. I find this easiest to understand by abandoning any geometric intuition and just thinking in terms of: what is the min-cut forced to optimize at a given area of the image?

Versus Dynamic Programming

DP / pixel error

DP / seam error

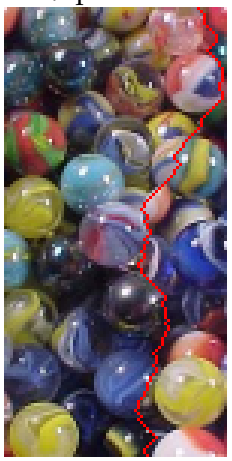
GC / seam error



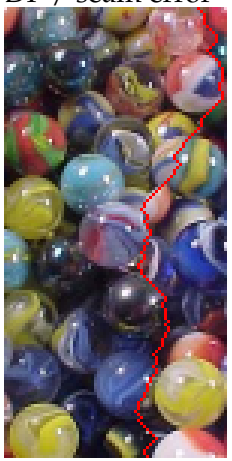
1. The error function makes a difference in eliminating the tiny "double edge" near the middle.
2. The algorithm allows graph cuts to find a slightly better path, but the difference is minimal.

Versus Dynamic Programming

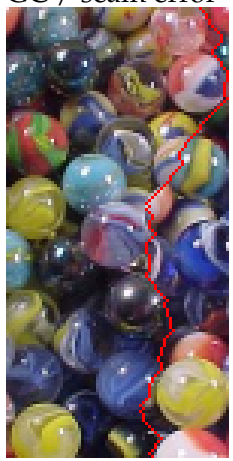
DP / pixel error



DP / seam error



GC / seam error



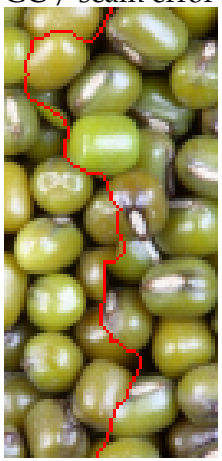
1. The results are surprisingly close.

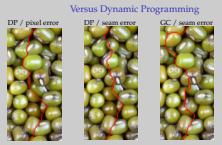
Versus Dynamic Programming

DP / pixel error

DP / seam error

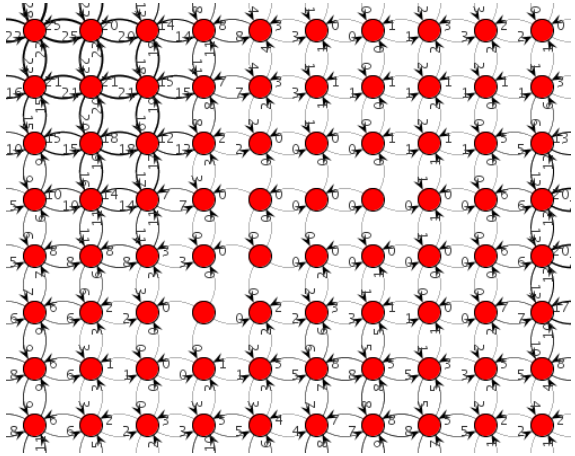
GC / seam error

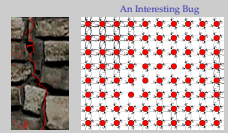




1. The results are surprisingly close.

An Interesting Bug





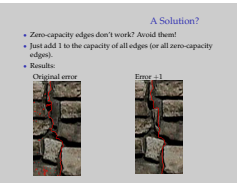
1. I noticed that some isolated regions of the foreground image were being included in the quilt.
2. Closer investigation reveals that these are regions surrounded by zero-error seams.
3. Is this bad? Not as long as the regions are small.
4. NEXT: obvious solution which doesn't work.

A Solution?

- Zero-capacity edges don't work? Avoid them!
- Just add 1 to the capacity of all edges (or all zero-capacity edges).
- Results:

Original error

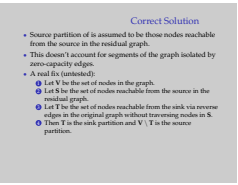
Error +1



1. Adding error to all nodes adds bias in favor of shorter paths.
2. This can have surprisingly bad results.
3. In general I noticed that graph cuts are much more sensitive to the error function than dynamic programming, because the error function affects how much the algorithm penalizes longer paths.

Correct Solution

- Source partition of is assumed to be those nodes reachable from the source in the residual graph.
- This doesn't account for segments of the graph isolated by zero-capacity edges.
- A real fix (untested):
  1. Let **V** be the set of nodes in the graph.
  2. Let **S** be the set of nodes reachable from the source in the residual graph.
  3. Let **T** be the set of nodes reachable from the sink via reverse edges in the original graph without traversing nodes in **S**.
  4. Then **T** is the sink partition and **V** $\setminus$ **T** is the source partition.



1. I haven't yet implemented this fix but hopefully it's obvious that it's correct.

Inherent Limitations

- Depends on good placement of images
- Strongly constrained by features of original image
- Doesn't always agree with human perception

Inherent Limitations
- Depends on good placement of images
- Strongly constrained by features of original image
- Doesn't always agree with human perception

1. DISCUSS: which is more important, finding seams or placing patches?
2. DISCUSS: how important is human perception in the quality of a seam?

Conclusion

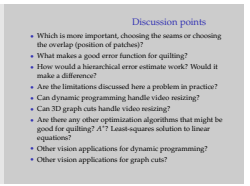
- Dynamic programming: fast but restricted
- Graph cuts: slow but flexible
- What is the best approach for quilting? Results are surprisingly inconclusive
- Future work: Implement Kwatra et al's multi-pass texture generation using dynamic programming to find seams instead of graph cuts
- Questions?

Conclusion
- Dynamic programming: fast but restricted
- Graph cuts: slow but flexible
- What is the best approach for quilting? Results are surprisingly inconclusive
- Future work: Implement Kwatra et al's multi-pass texture generation using dynamic programming to find seams instead of graph cuts
- Questions?

1. I think the reasons Kwatra et al's results look better, in order of importance, are: multiple iterations with larger patch size, better error functions, splining near seams, and last, maybe, graph cuts

Discussion points

- Which is more important, choosing the seams or choosing the overlap (position of patches)?
- What makes a good error function for quilting?
- How would a hierarchical error estimate work? Would it make a difference?
- Are the limitations discussed here a problem in practice?
- Can dynamic programming handle video resizing?
- Can 3D graph cuts handle video resizing?
- Are there any other optimization algorithms that might be good for quilting? A^*? Least-squares solution to linear equations?
- Other vision applications for dynamic programming?
- Other vision applications for graph cuts?



1. Questions intended to stimulate post-talk discussion.