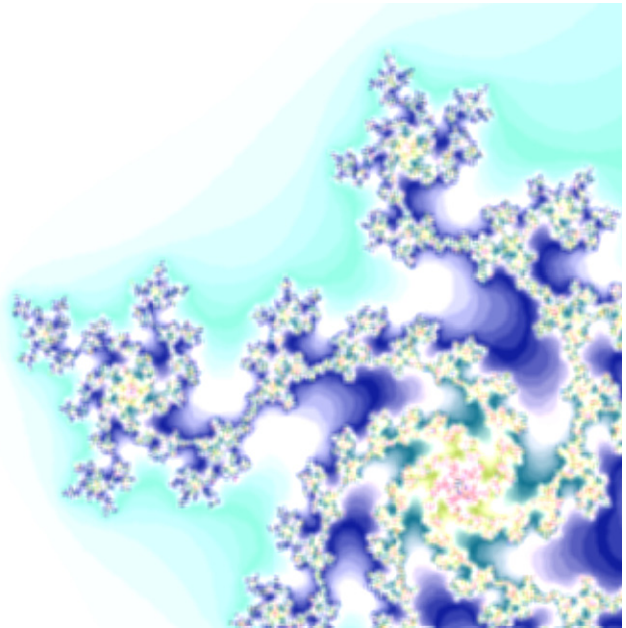
Graphcut Textures

Image and Video Synthesis Using Graph Cuts

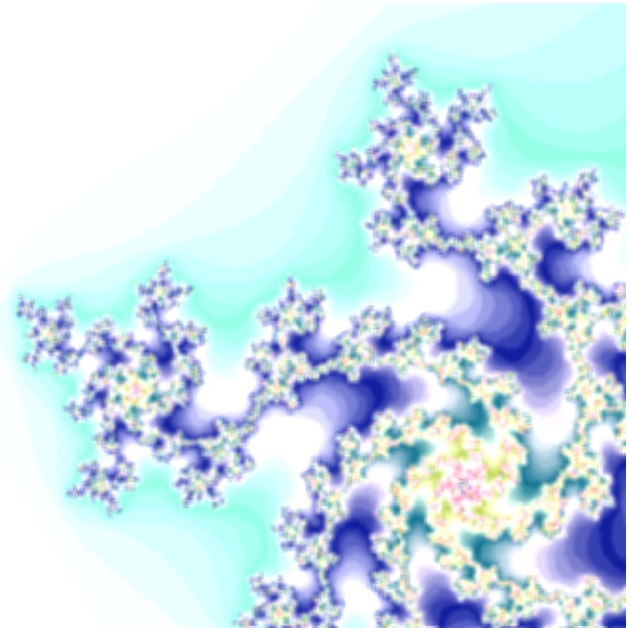
Vivek Kwatra
Arno Schödl
Irfan Essa
Greg Turk
Aaron Bobick



Presented by Benjamin Sigg

Topics

- Whats the goal?
- Graph Cut
- Patch placement
- Extensions, Video
- Authors conclusions
- My conclusions

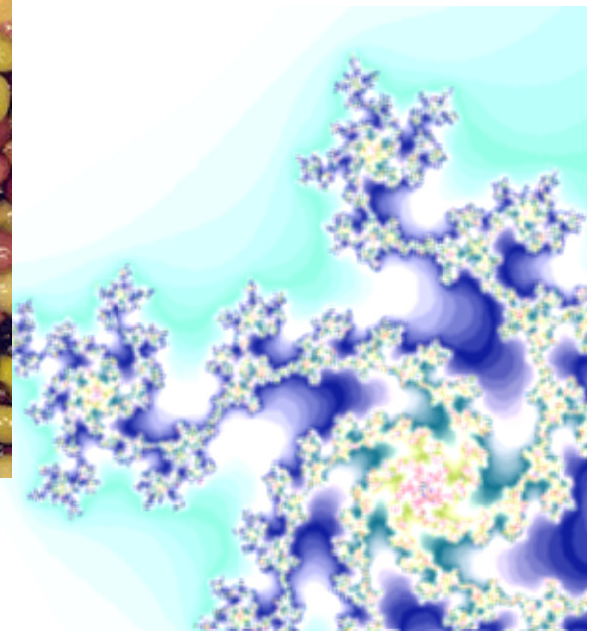


Creating Textures

- Input is a small image
- Output is a large image

Input

Output



Idea behind

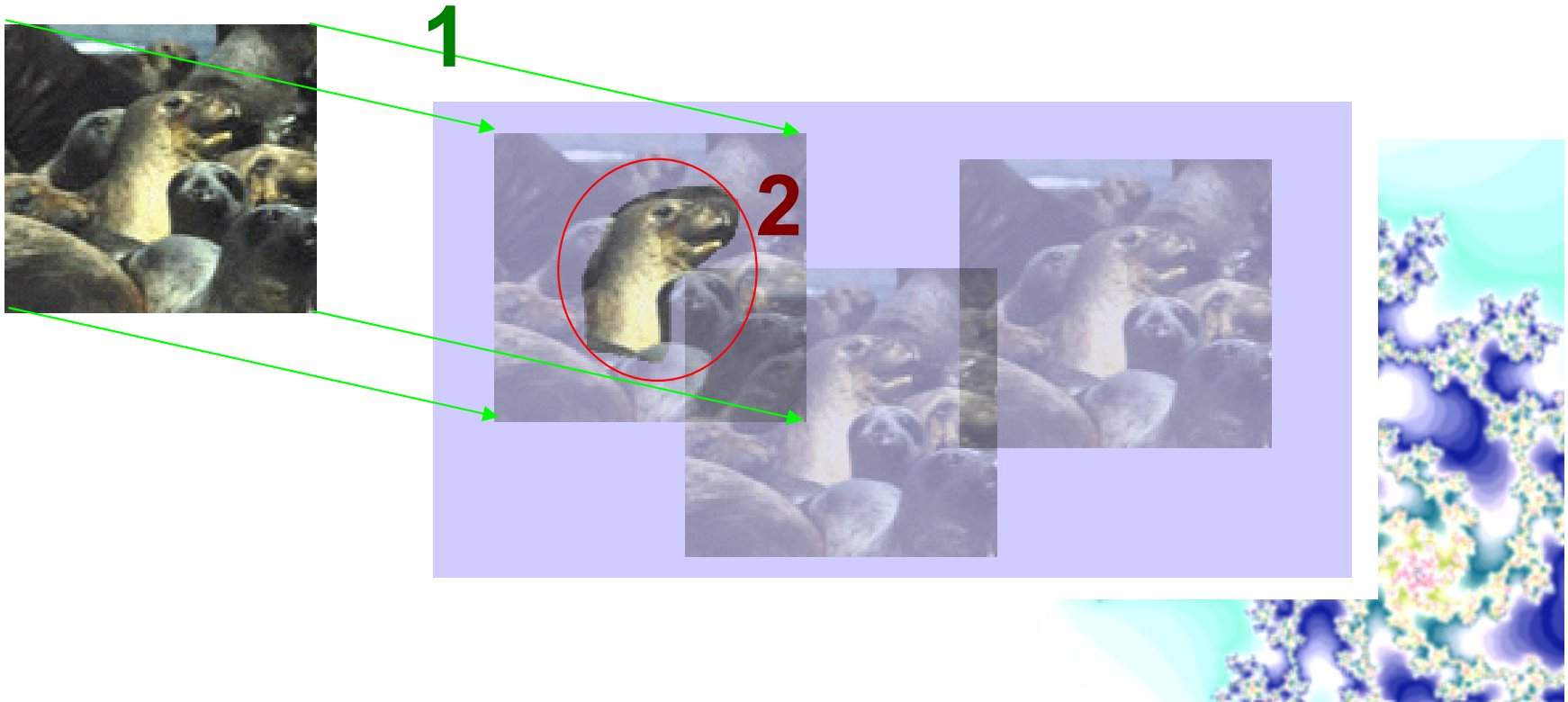
- Take a patch from the input
- Place the patch somewhere on the output such that no seams are visible

Input

Output

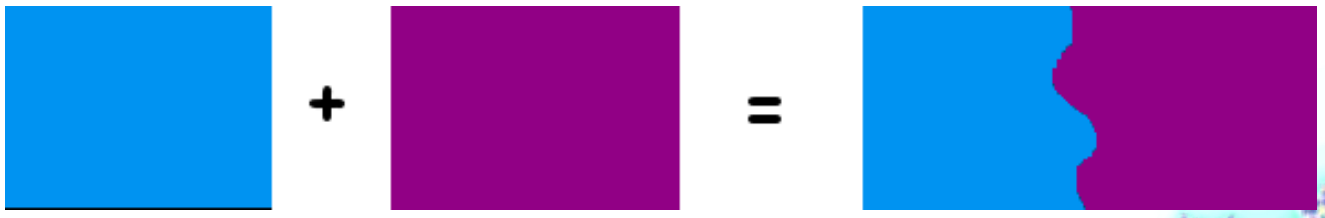
Idea behind

- Iterate over
  - Step 1: Patch matching & placement
  - Step 2: Graph cut



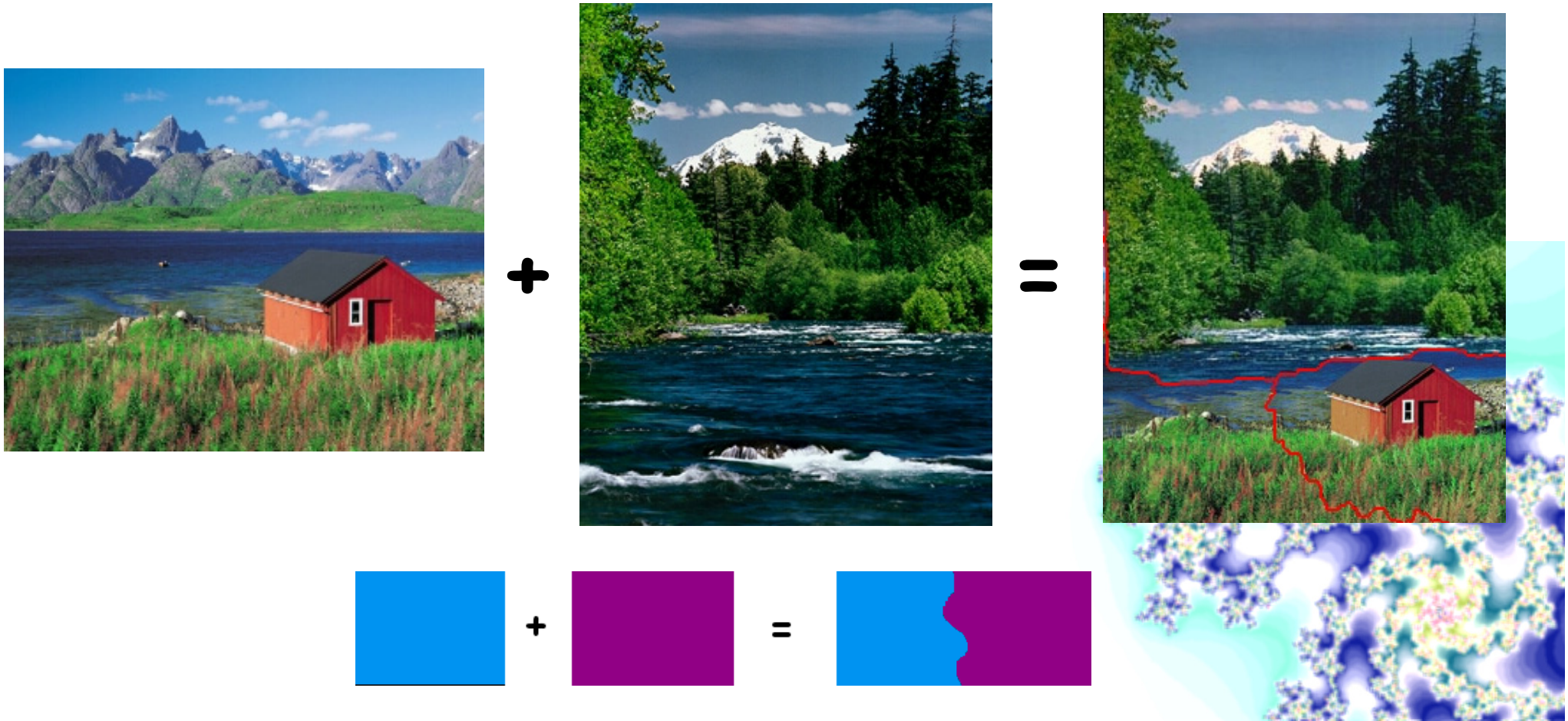
Graph Cut

- Cut two images, such that they can be seamless merged
- Minimize a costfunction for cut and merge



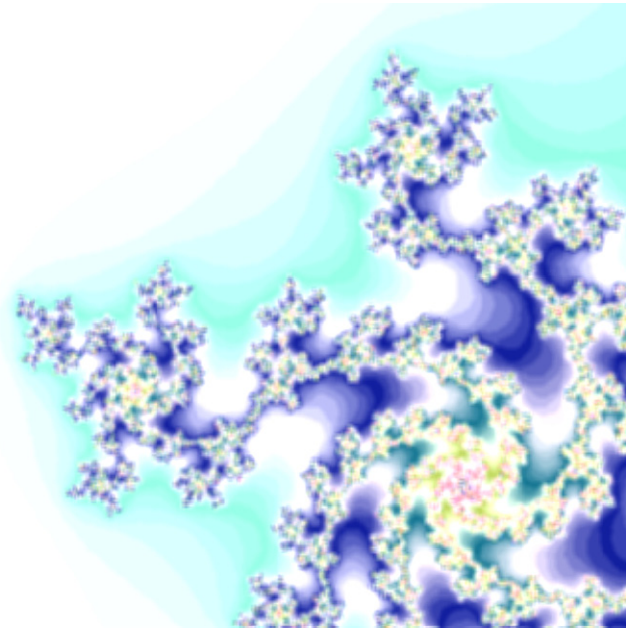
Graph Cut

- A cut is more than a line



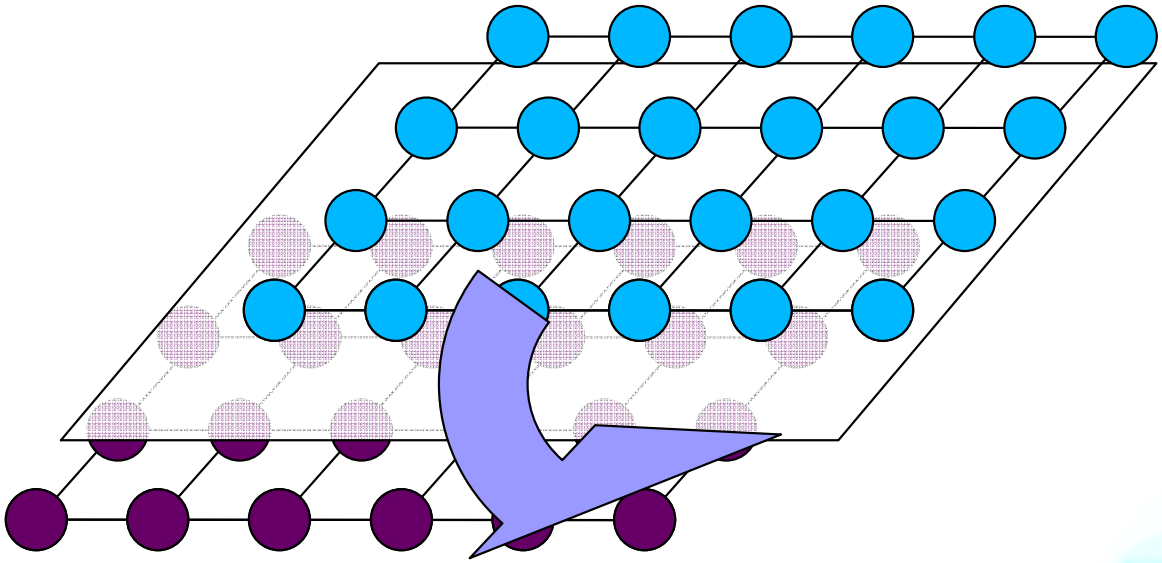
Graph Cut

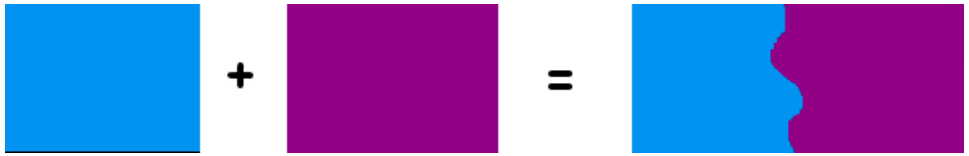
- Input
  - One graph
  - One costfunction
- Output
  - The cut with minimal costs

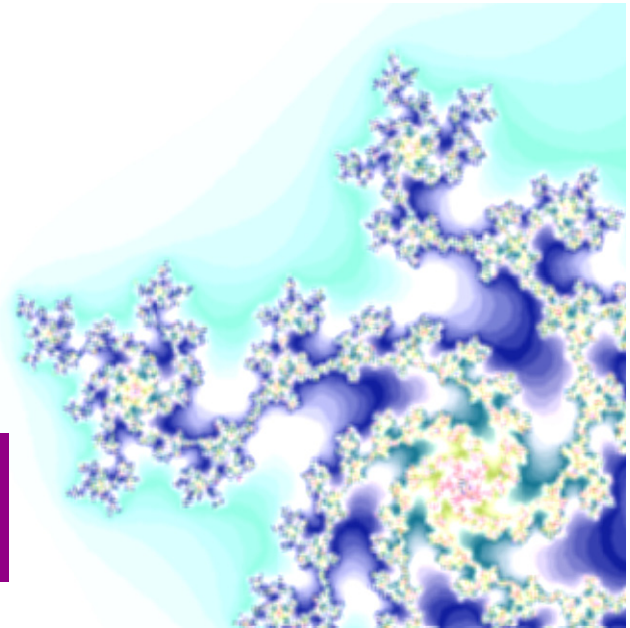


Graph Cut: Input

- Two images seen as graphs

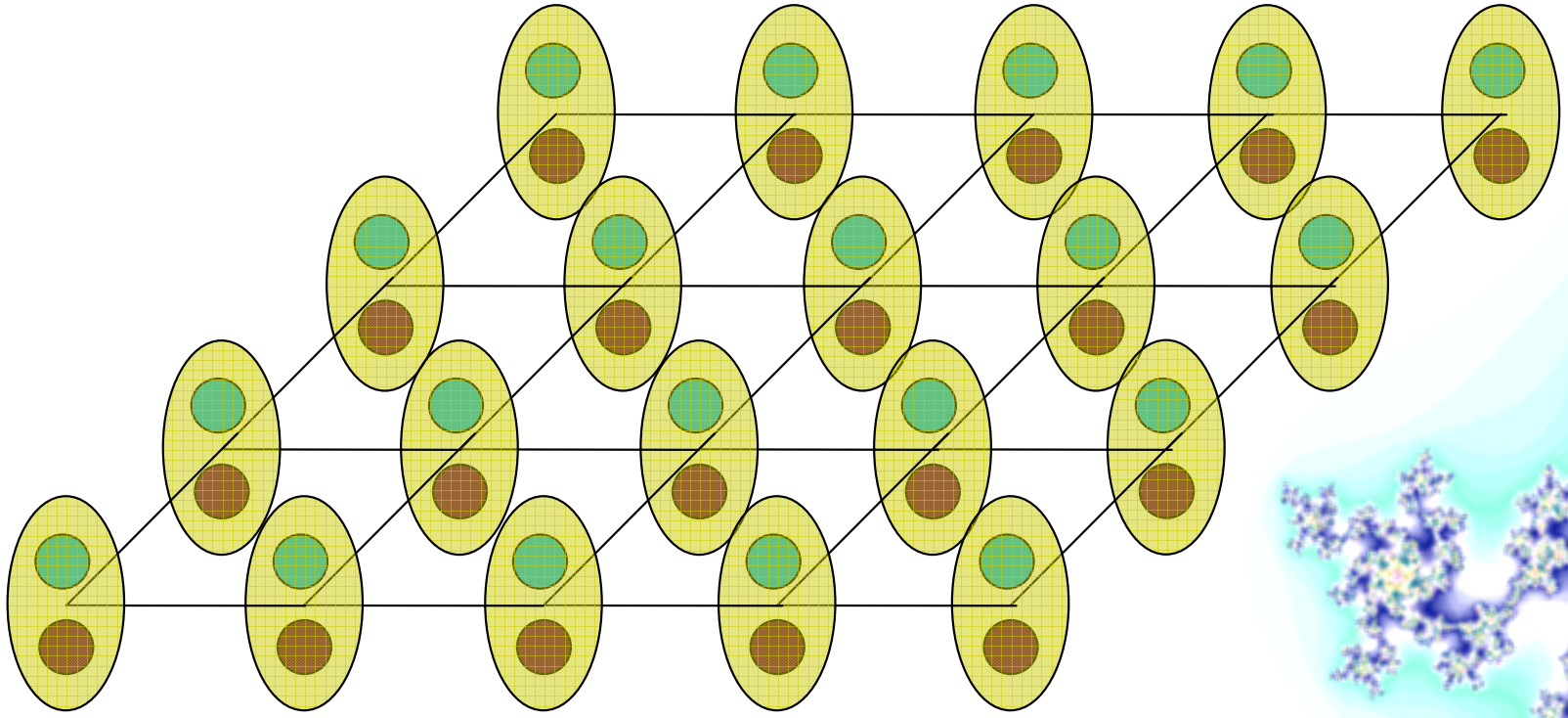


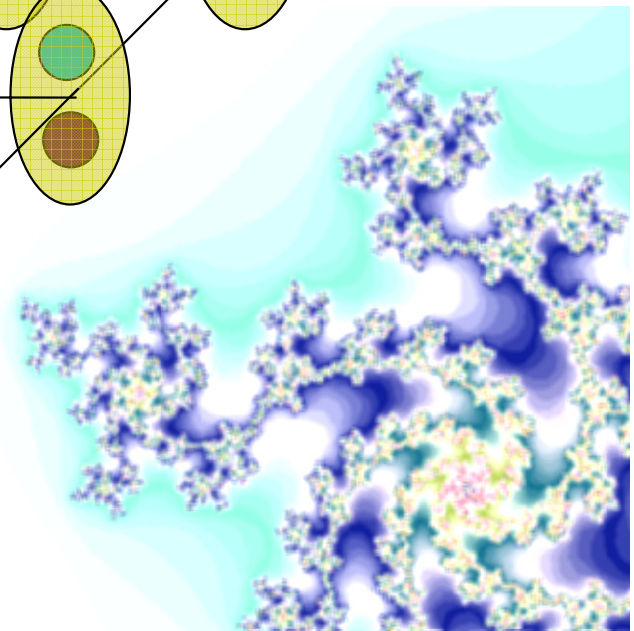




Graph Cut: Input

- But only one graph required
- Use an “embracing” graph

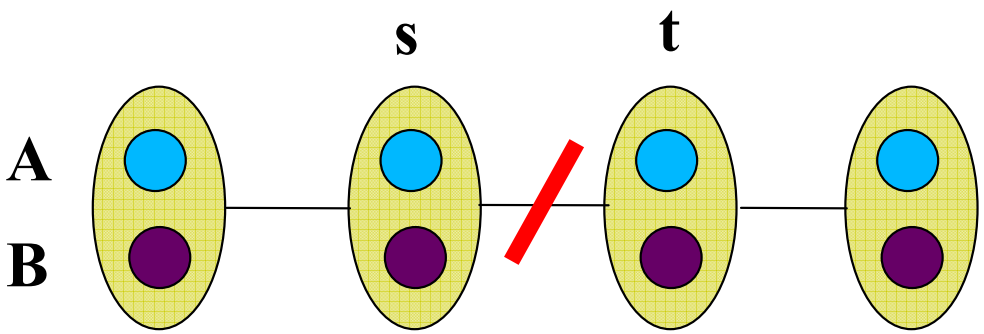


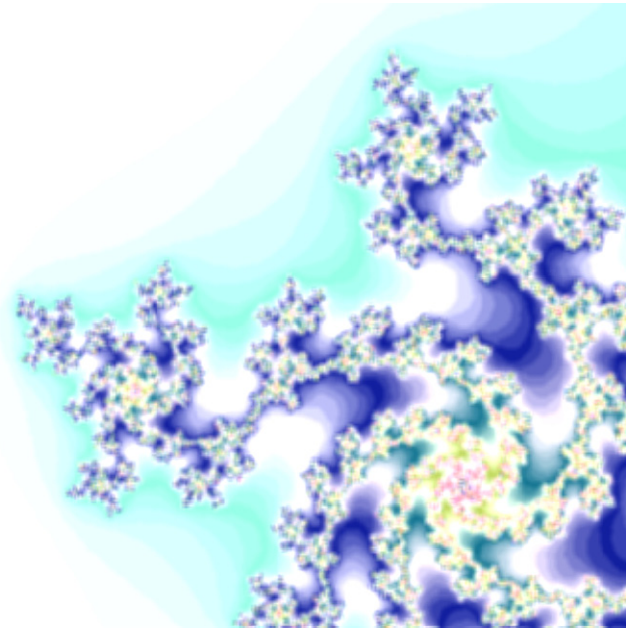


Graph Cut: Input

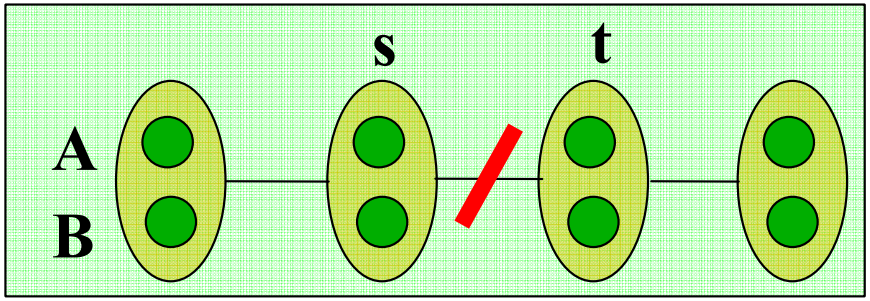
- Costfunction

$$M(A,B,s,t) = |A(s)-B(s)| + |A(t)-B(t)|$$

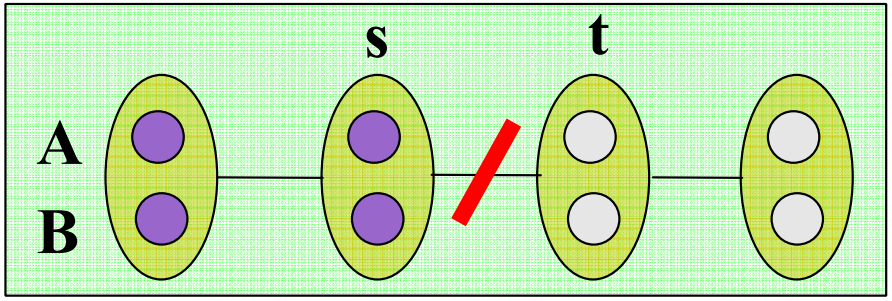


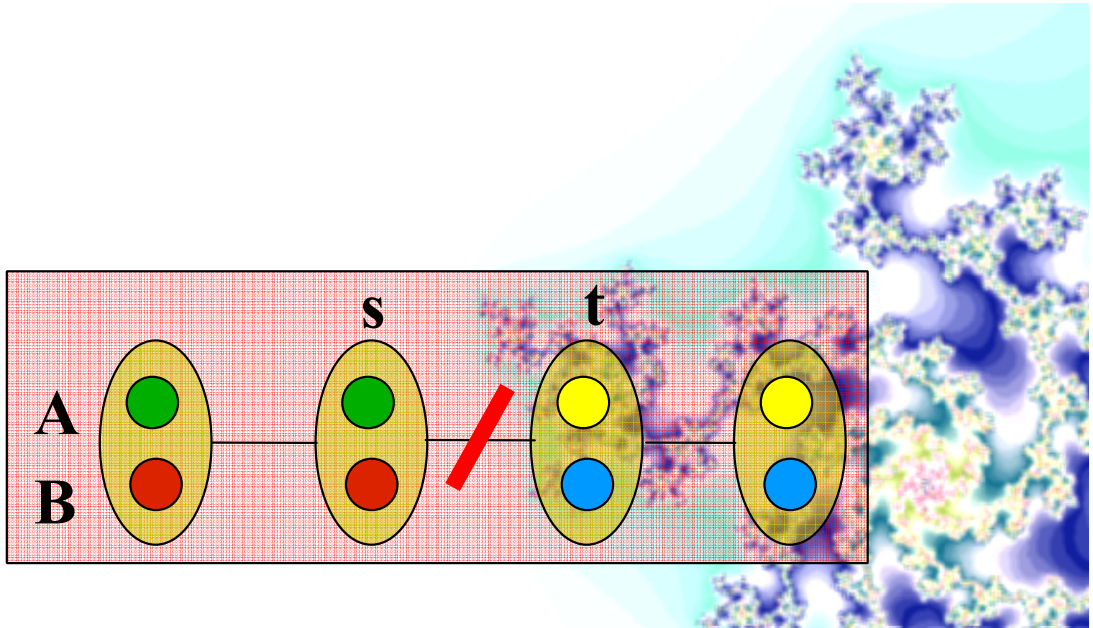


Graph Cut - Input



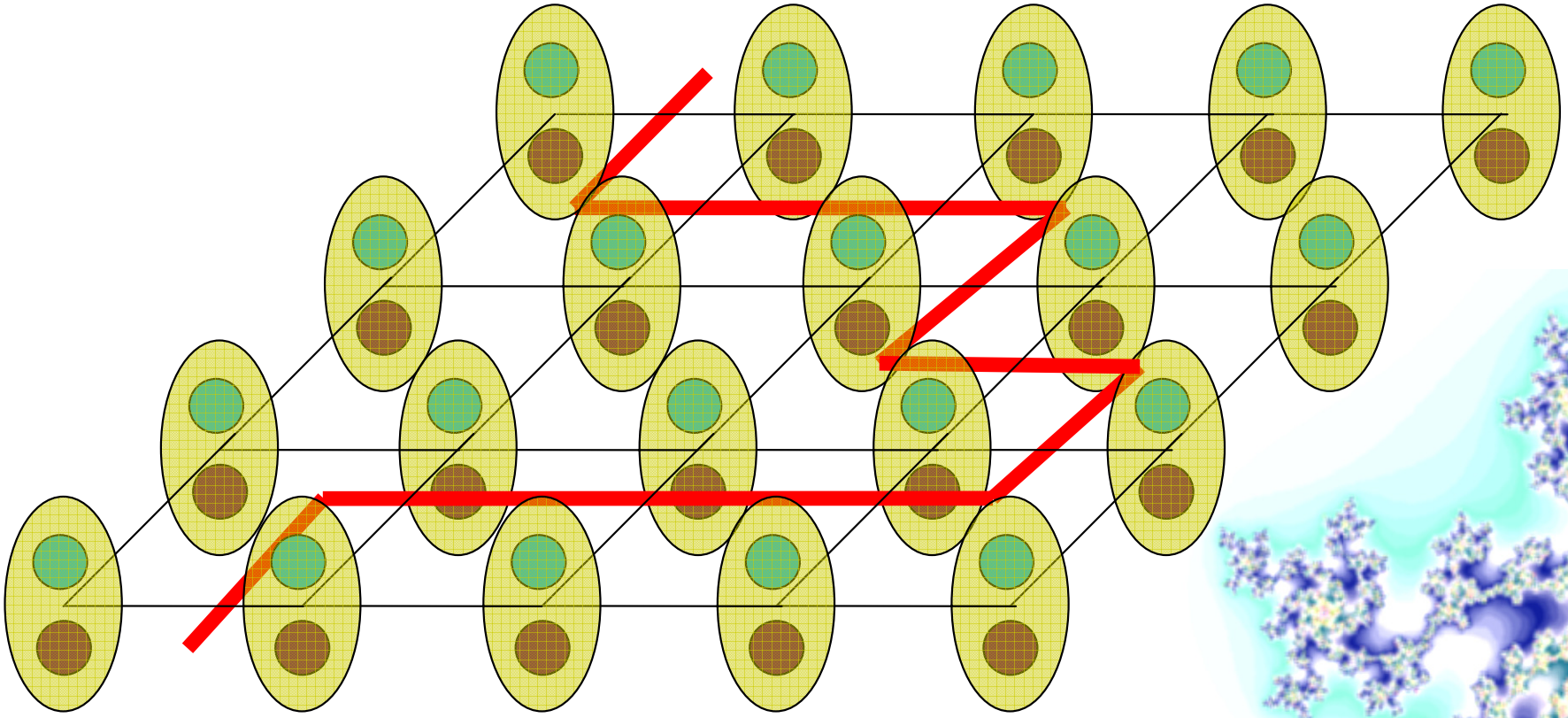
$$M(A,B,s,t) = |A(s)-B(s)| + |A(t)-B(t)|$$





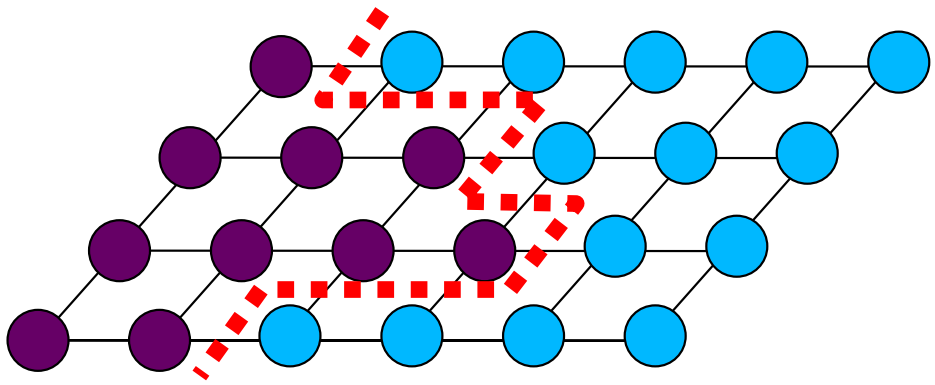
Graph Cut - Output

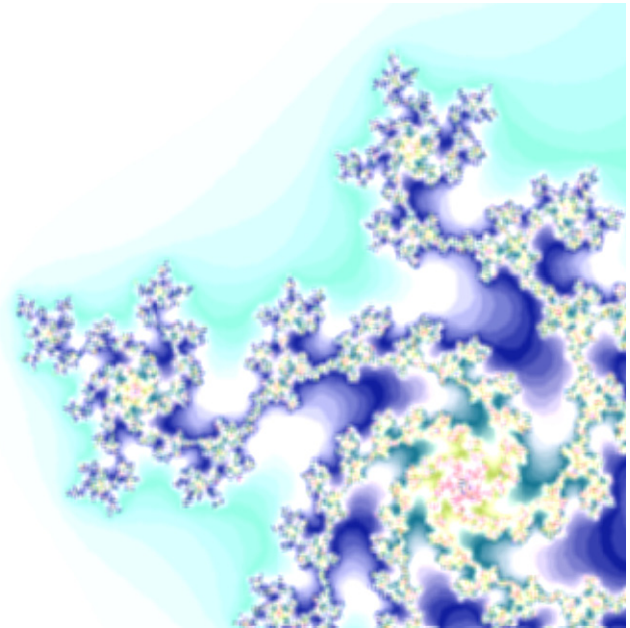
- The minimal cut of the graph



Graph Cut - Output

- Merge the two images along the cut
- (Hopefully) no visible seams



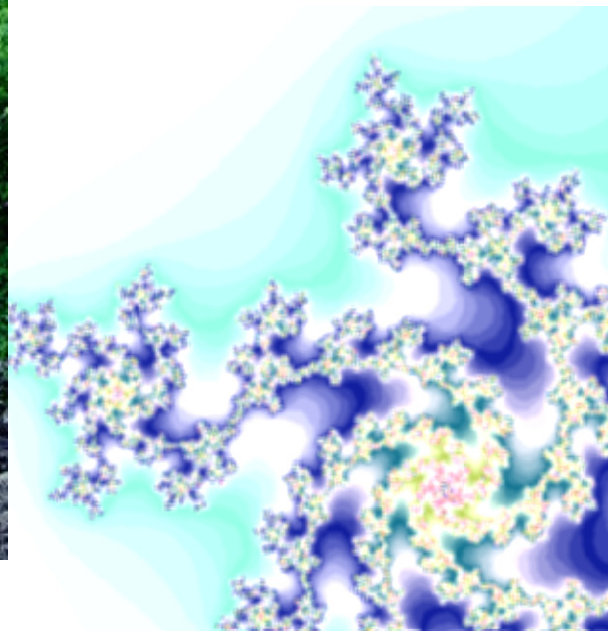


Graph Cut: Example

- Two merged images
- No seams are visible

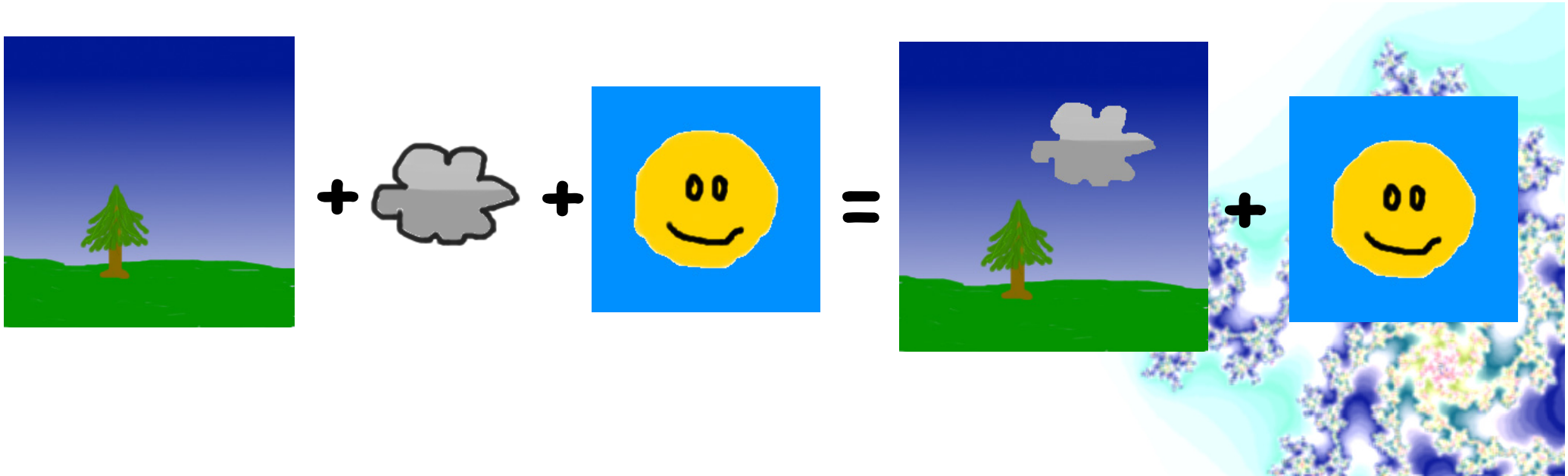
Graph Cut

- Algorithms are well known
- Many possible Costfunctions
- What about many images?



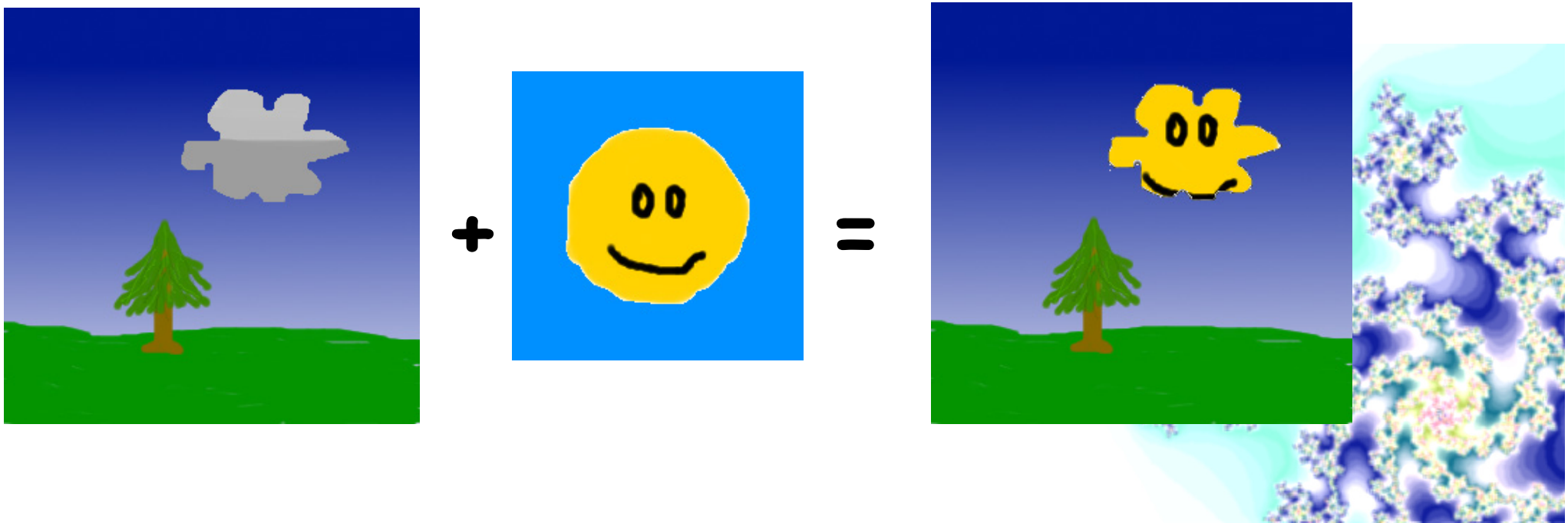
Many Images

- Merge first two images
- Merge resulting image with a third image…



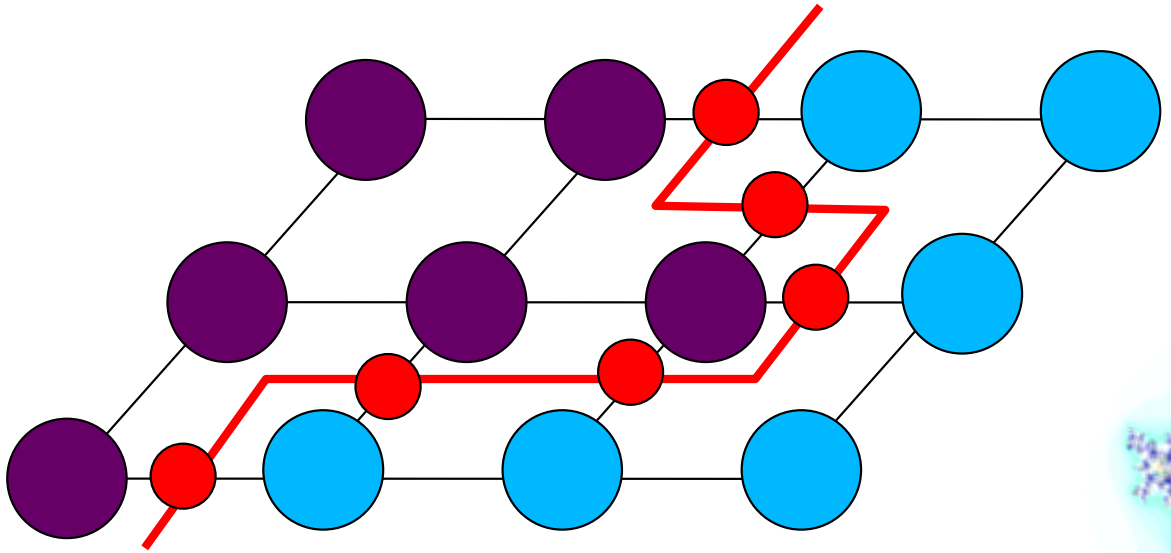
Many Images

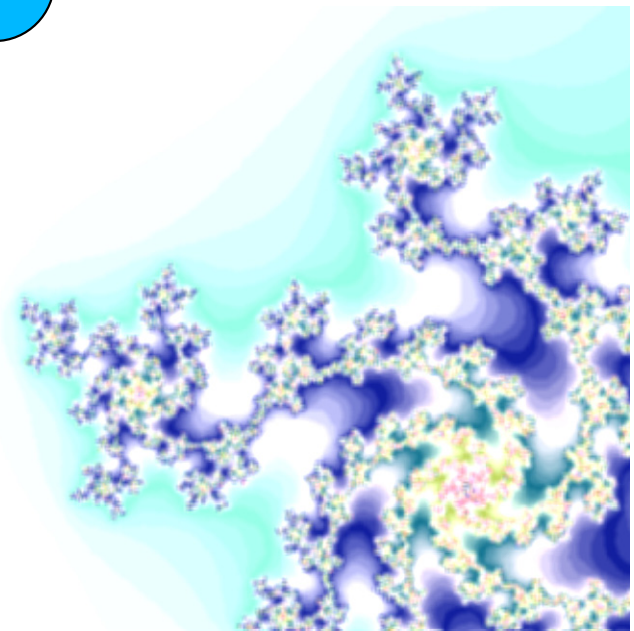
- Merge with a third image
- Result is very bad
- Don't throw away information!



Many Images

- Store old seam costs

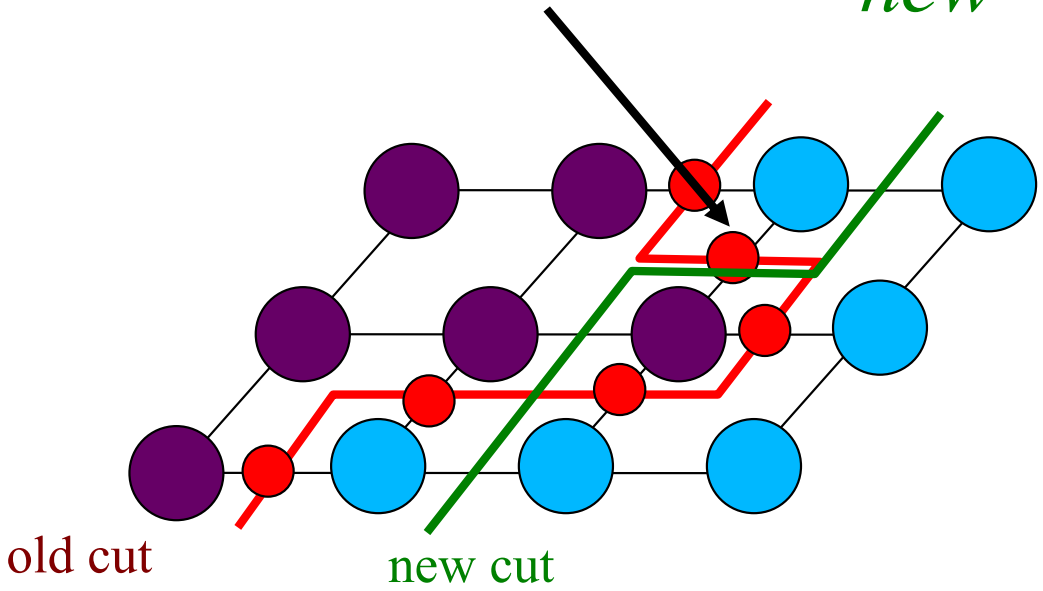


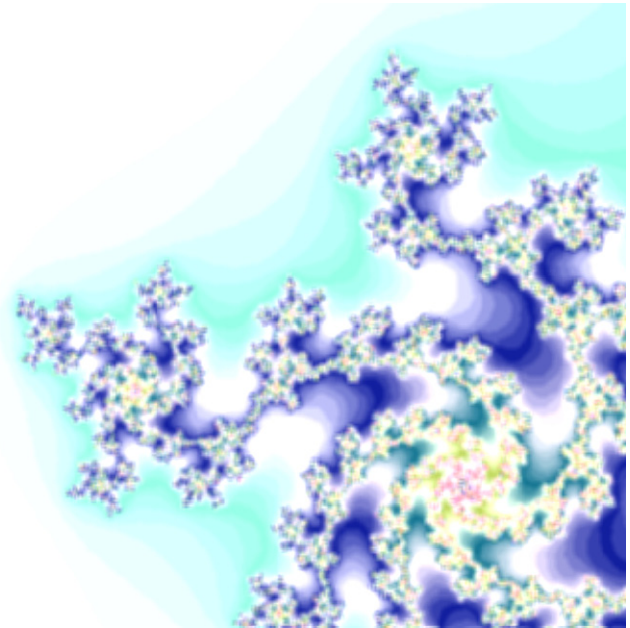


Many Images

- Add the old seam costs to the new cut

$$cost = cost_{new} + cost_{old}$$

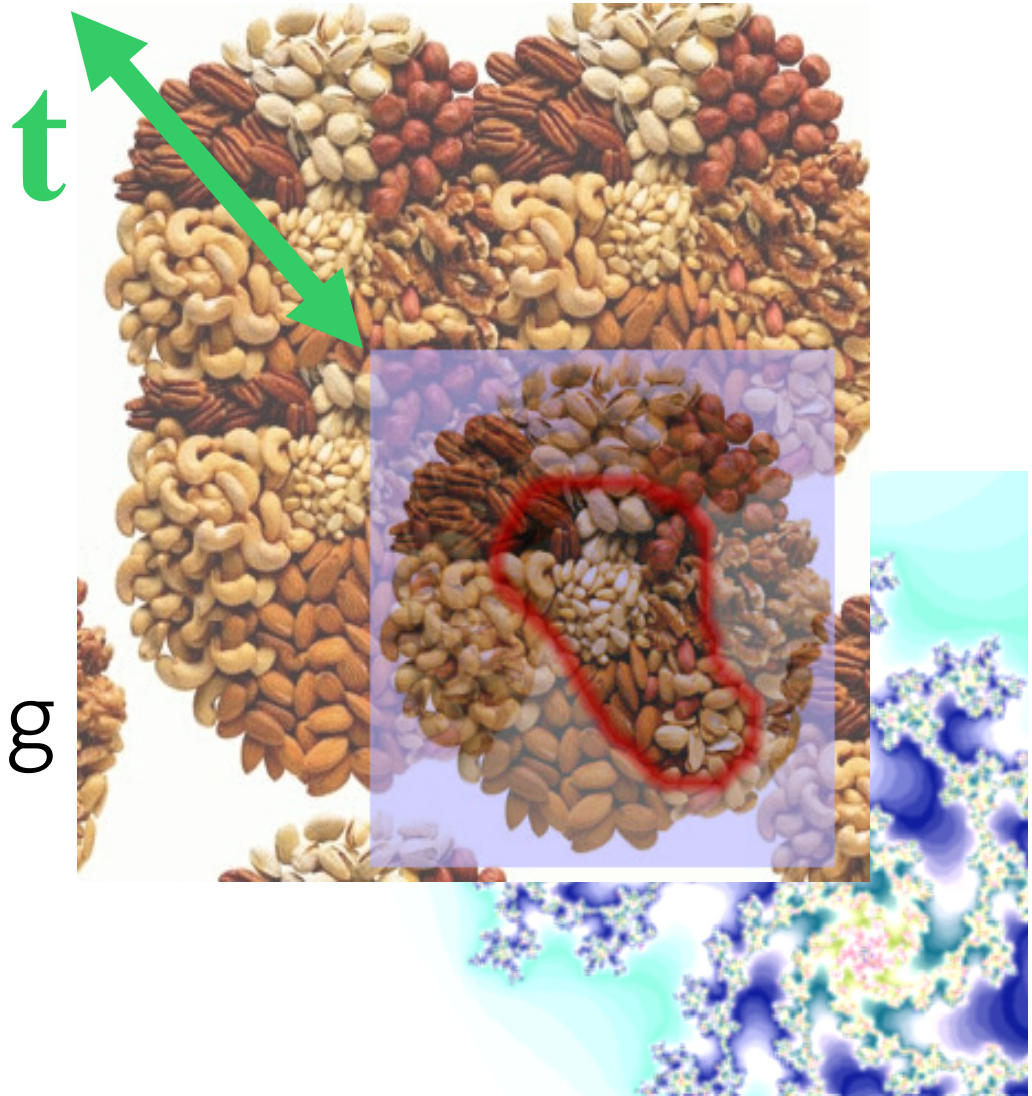




Patch Placement

- Place input somewhere on the output
- Cut out a patch

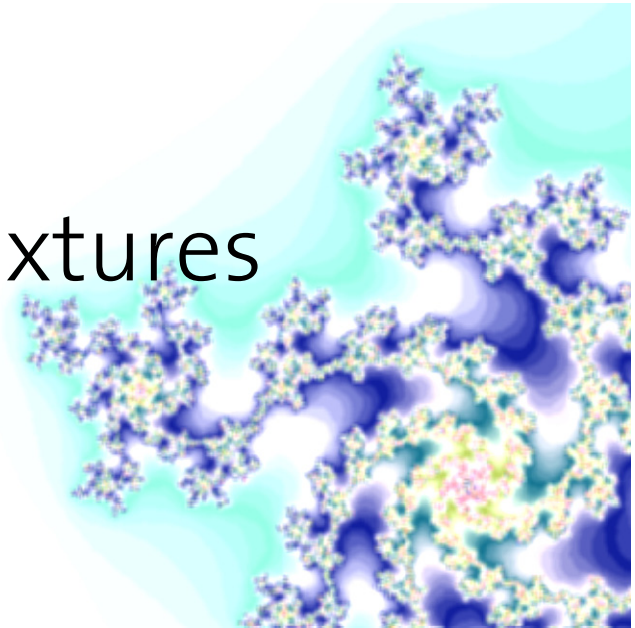
- Random placement
- Entire patch matching
- Sub-patch matching



Random placement

- Place the input randomatically on the output
- Graph cut decides, which part of the input will be seen

- The fastest approach
- Results are good for random textures



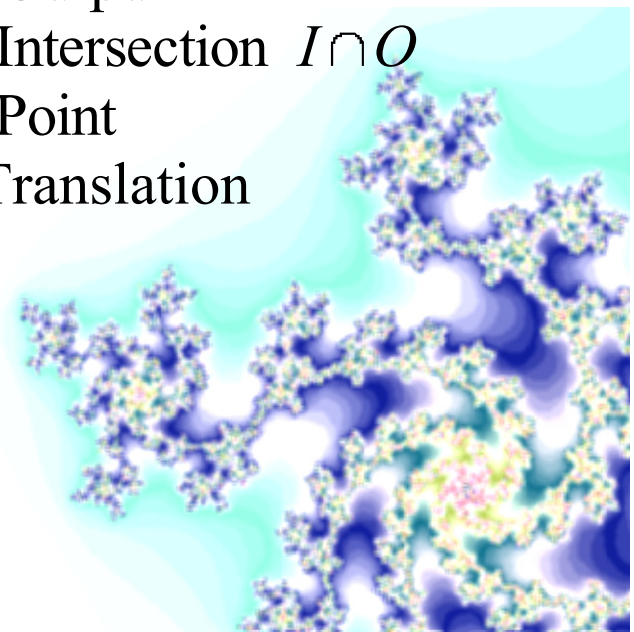
Entire patch matching

- Place the input such that it matches the output best
- Costfunction for Placement (SSD):

$$C(t)=\frac{1}{|A_t|}\sum_{p\in A_t}|I(p)-O(p+t)|^2$$

I: Input
O: Output
A: Intersection $I\cap O$
p: Point
t: Translation

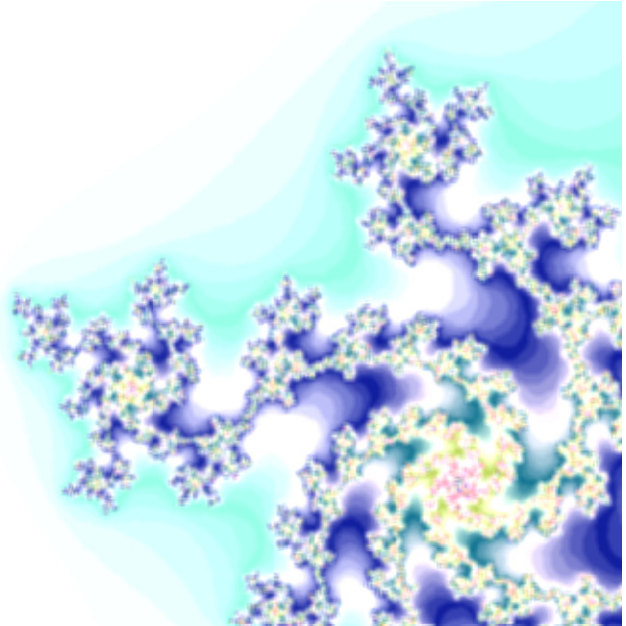
- Results are good for (semi-) structured textures



Sub patch matching

- Choose a small patch in the output
- Search a patch in the input that fits well
- Same Costfunction as “entire patch matching”

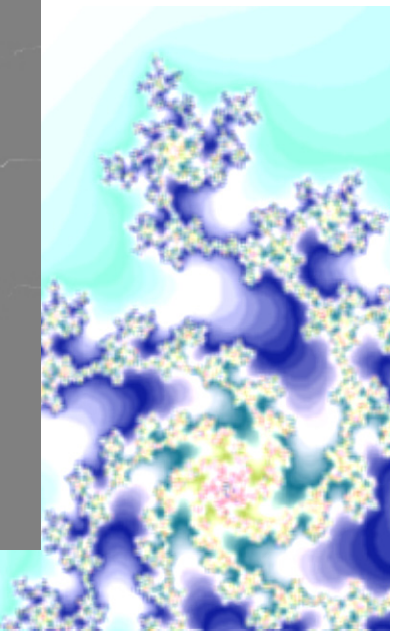
- Most general technique
- Good for videos



Results

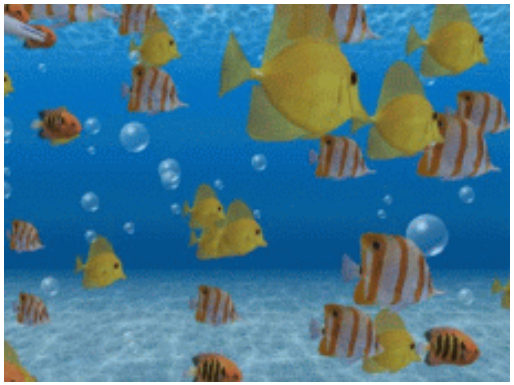
Results

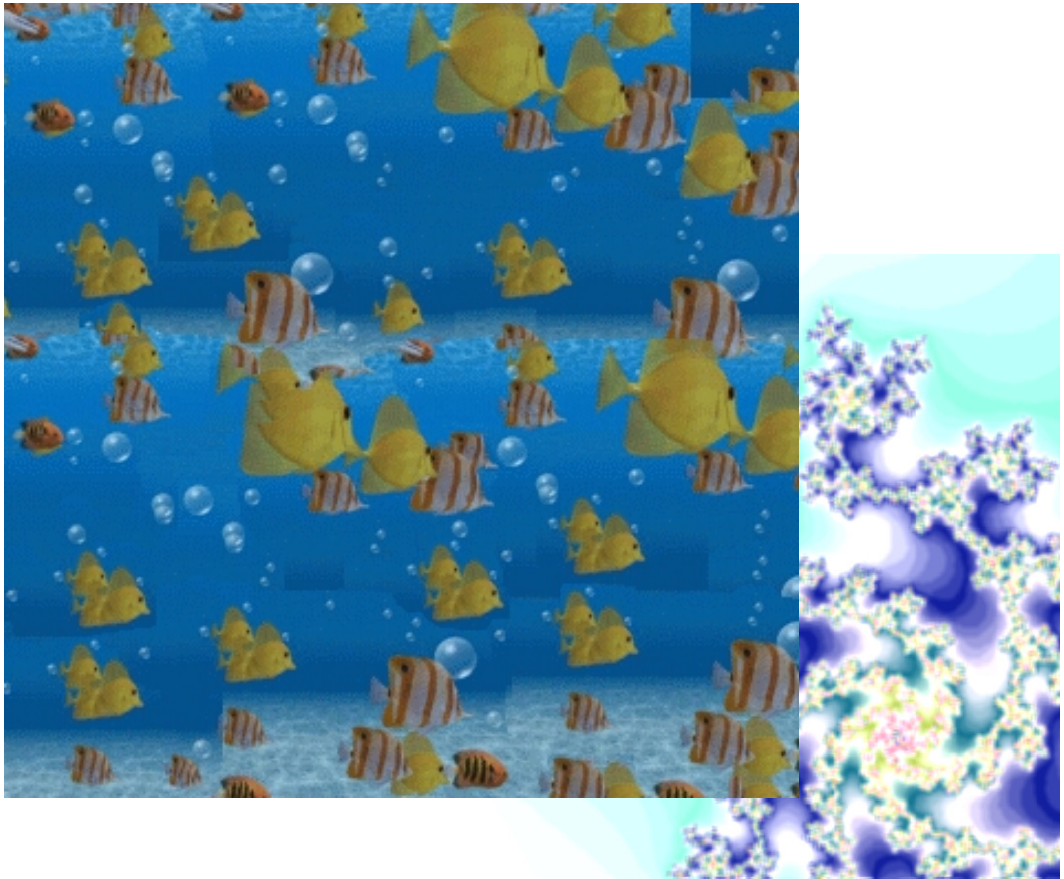
- Input, Output and Error



Results

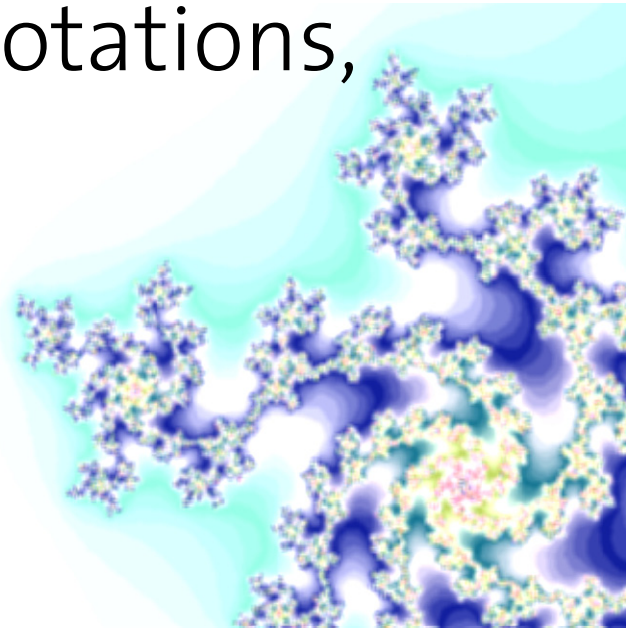
- Not every image is a good input





Extensions

- Other costfunctions
- Blending techniques to omit seams
- FFT for patch placing, dramatically fastens the algorithm
- Use not only translations, use rotations, mirroring and scaling.

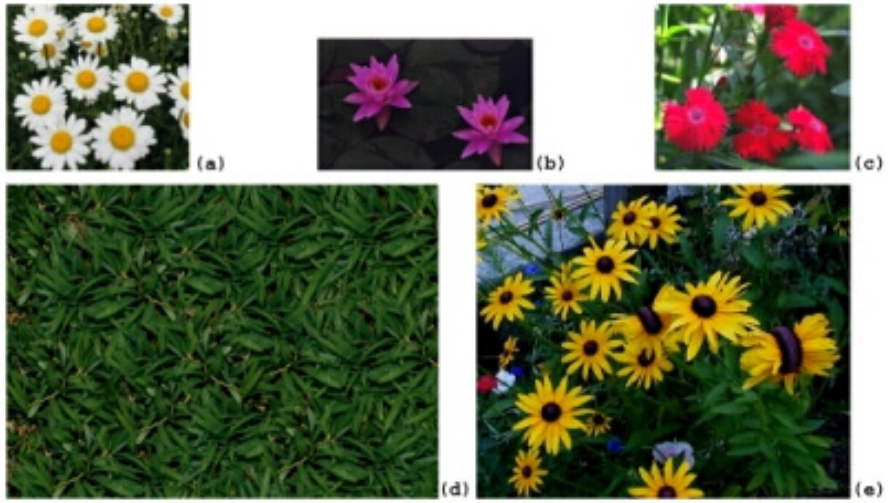


Extensions

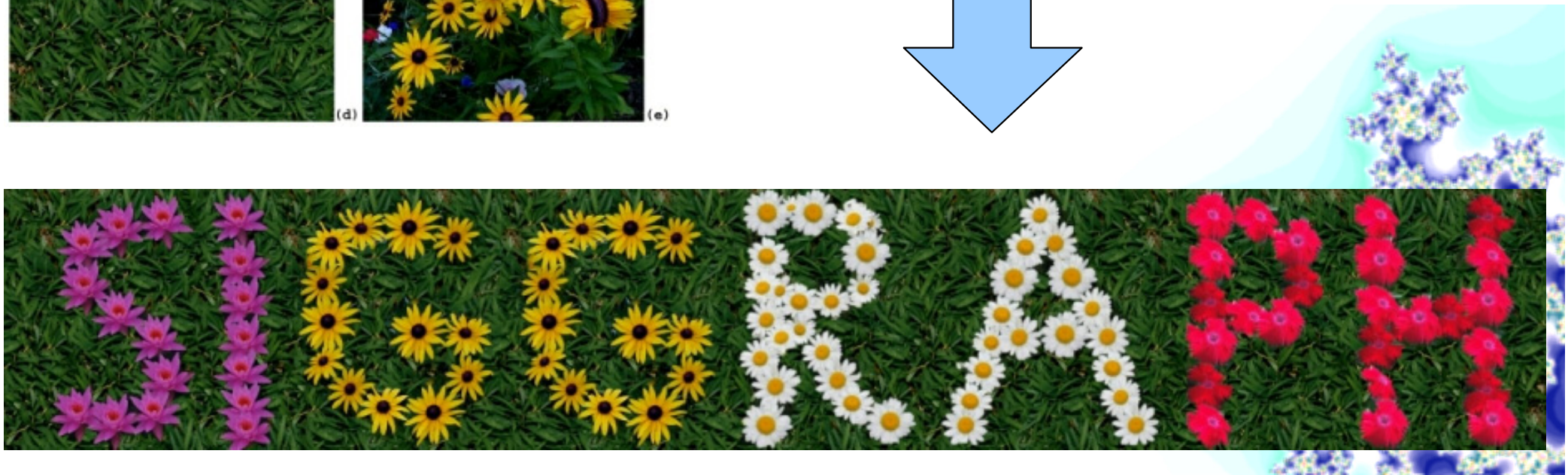
- Perspective by scaling the input
- Additional Constraints for scaling

Extensions

- Interactive Merging and Blending

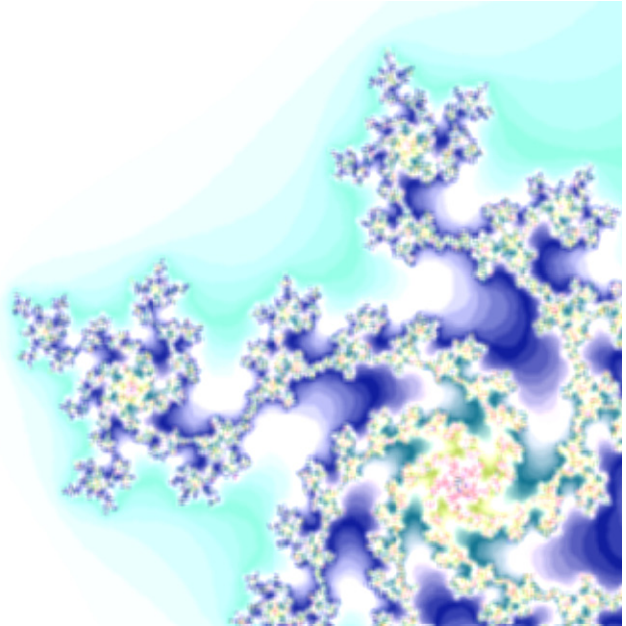


1-2 hours of work



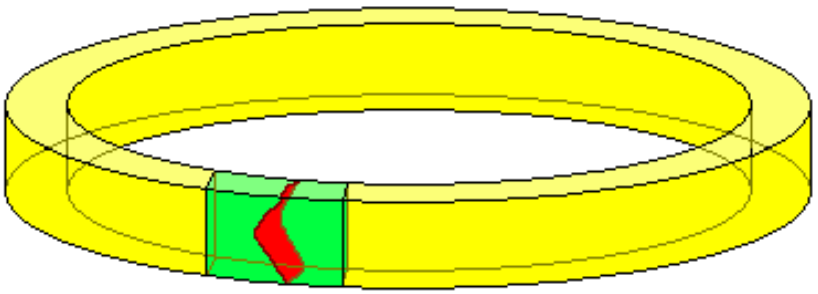
Video

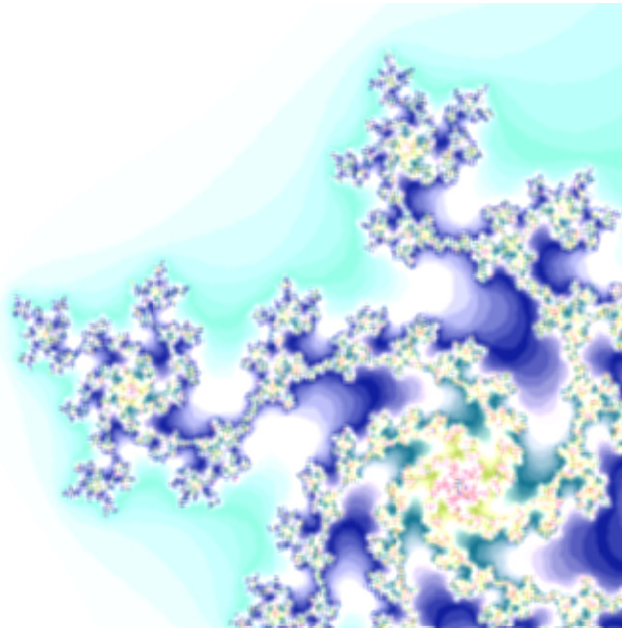
- Can be used to generate infinite videos
- Just see a video as 3-dimensional graph with axis: x, y, t.
- Loop the video



Video Transition

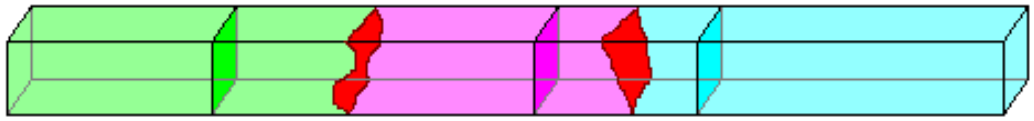
- Search two good matching frames
- Make the cut through this two frames
- Cut is a plane

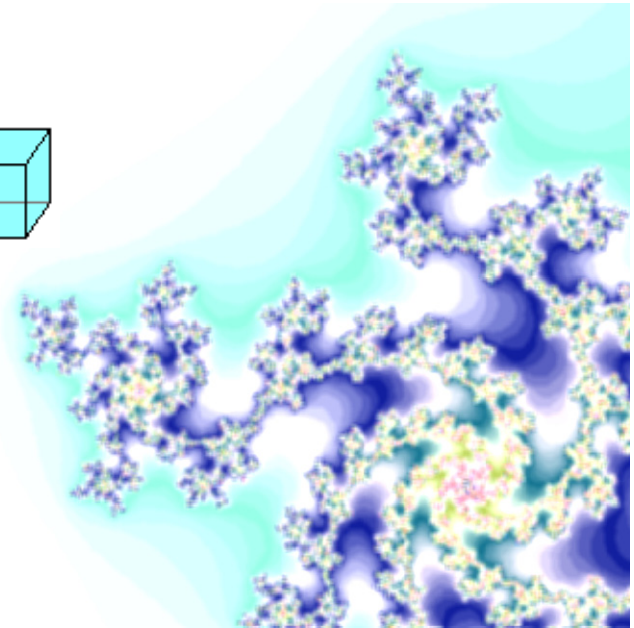




Random Temporal offset

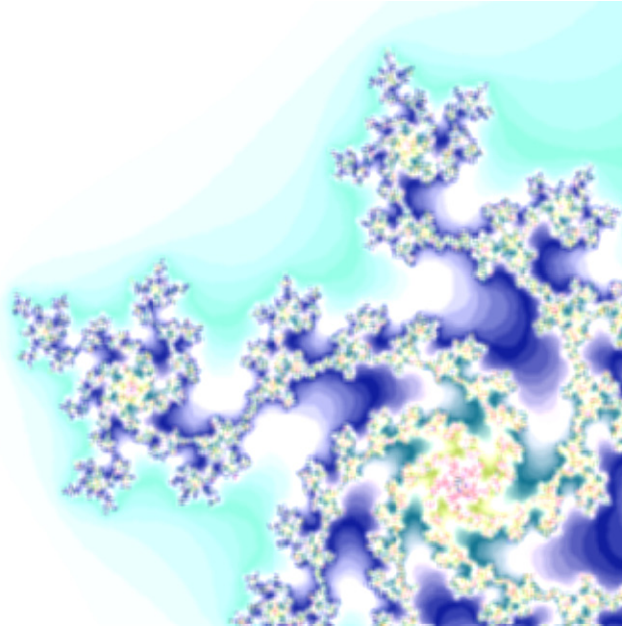
- For small input
- No periodicity
- Requires continous part in time





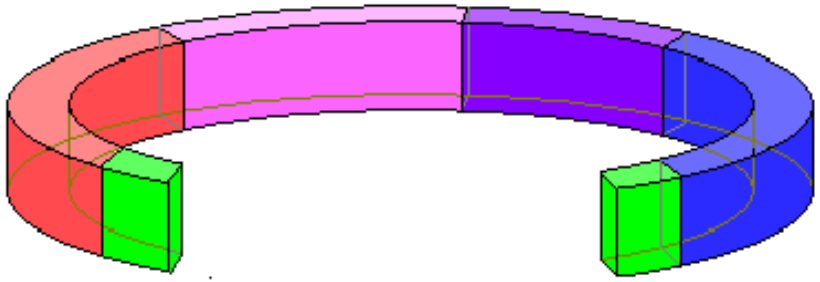
Spatio-Temporally Stationary Videos

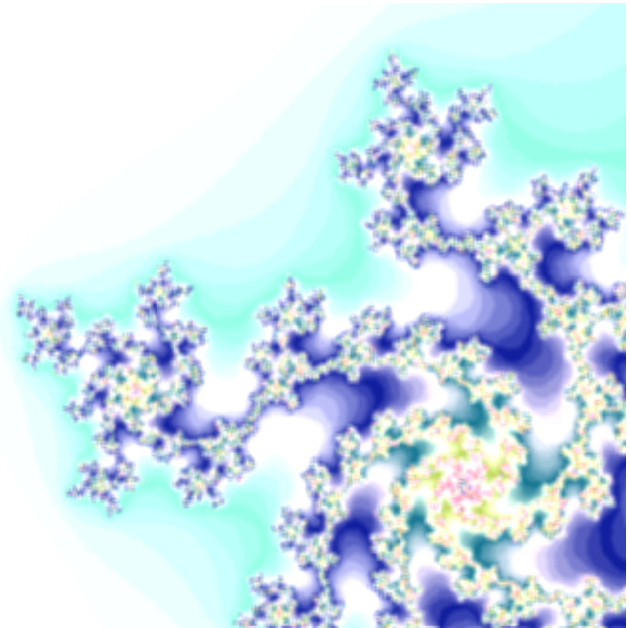
- For videos with a one-directional movement (e.x. Smoke)
- Pixels moved in time and space



Temporal Constraints

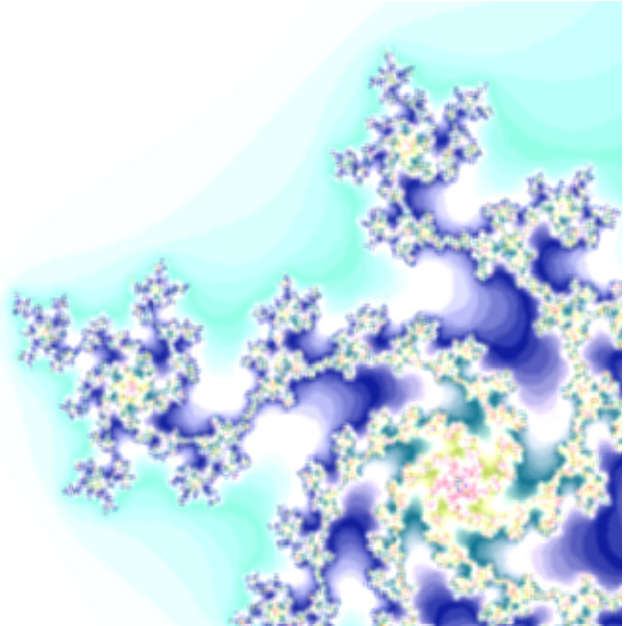
- Moving pixels in time and place causes problems, continuity is lost
- First and last n frames fixed by input
- Last n frames removed when output created





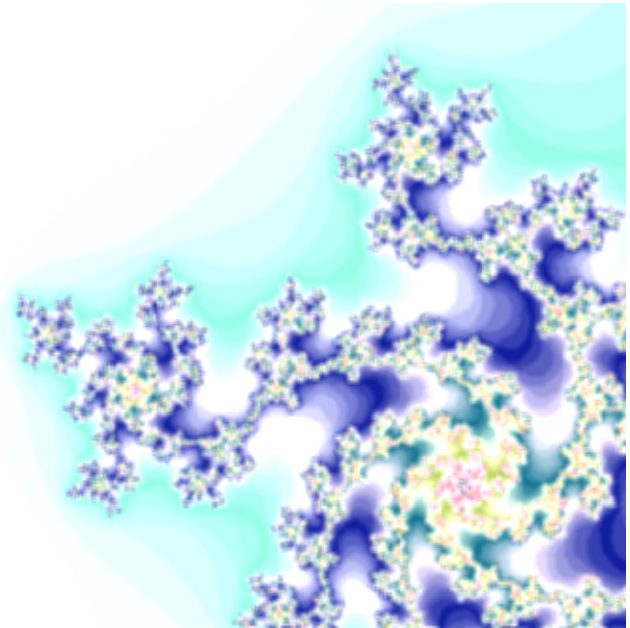
Authors conclusions

- Algorithm is fast
- No restrictions for the shape
- Method for adding constraints
- Easely generalized



My conclusions

- Very flexible technique:
  - User interaction is possible
  - Different costfunctions applicable
  - Different pach placing/matching algorithms available
- Good for random images
- Still problems with videos



The End