Assignment #4: Morphing

Date handed out: March 21, 2014; Due date: April 4, 2014

PDF version

Overview

In this assignment you will implement and experiment with Beier and Neely's morphing algorithm to be discussed in the lecture and the upcoming tutorial.

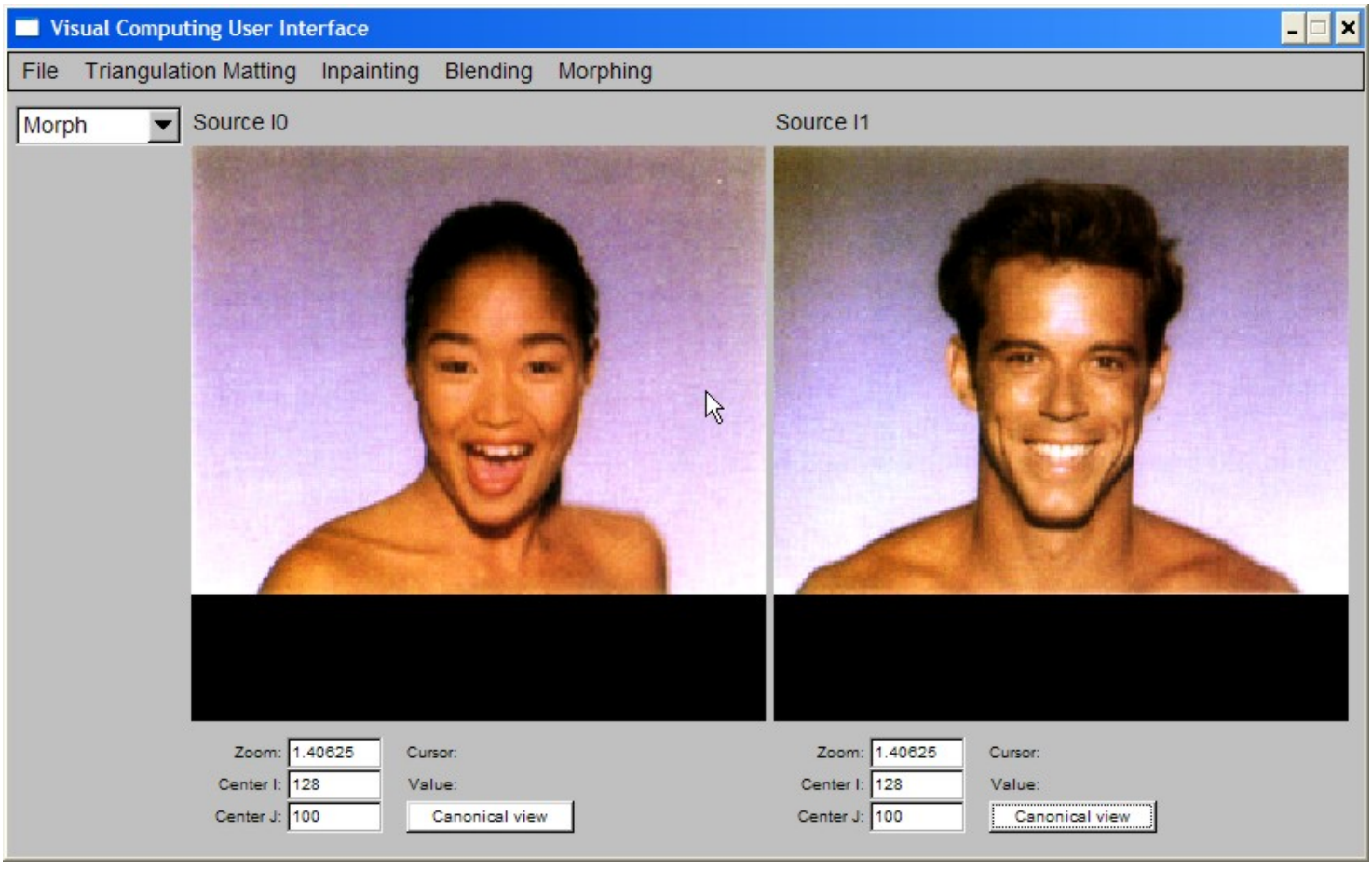
The goals of this assignment are:

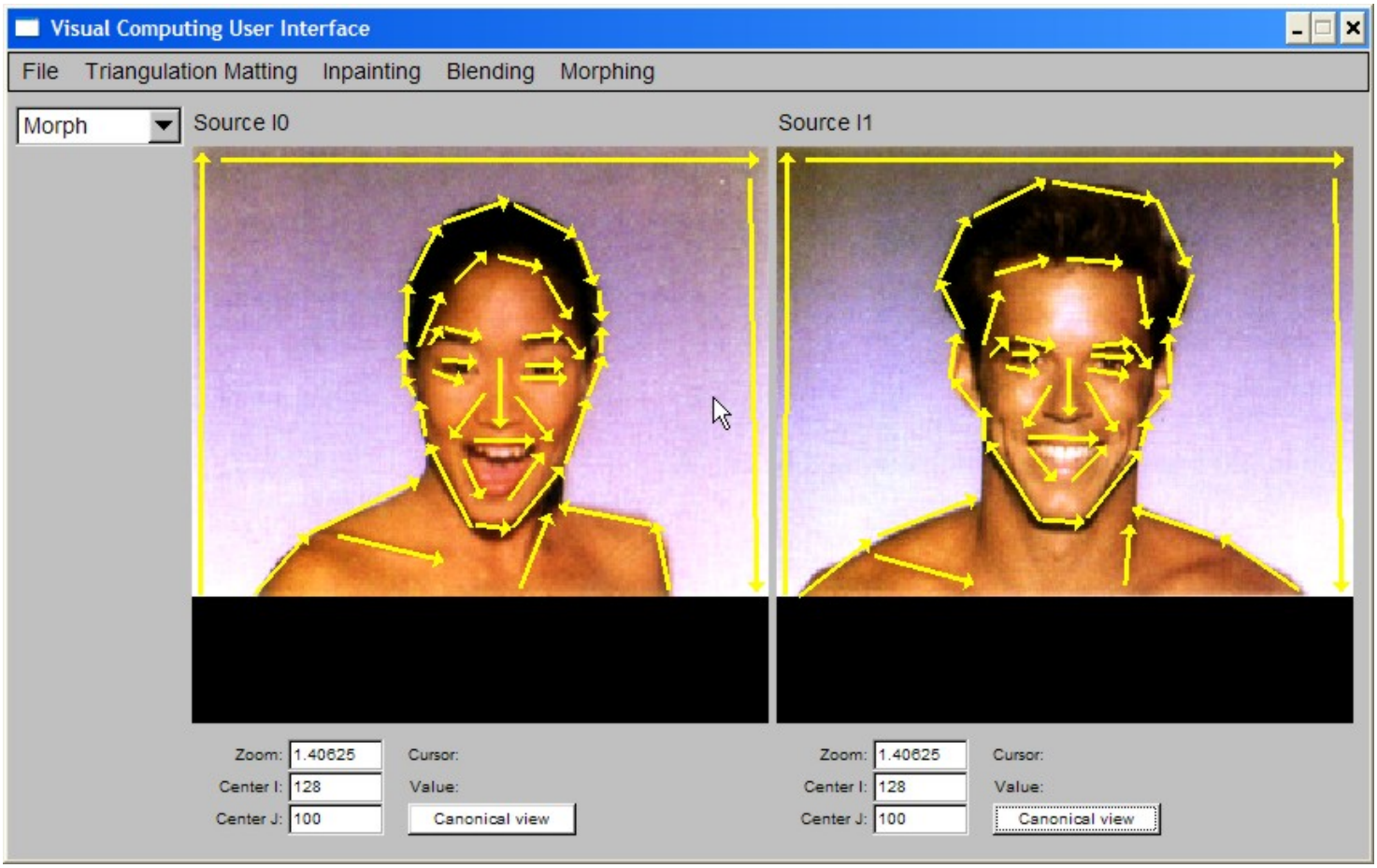
- to become familiar with a basic and very useful type of spatial image transformation (ie. morphing)
- to read and understand one more computer vision research paper
- to read and understand the implementation of one more non-trivial image manipulation techniques
- to run your own experiments, using the tools

While the due date for the assignment is 2 weeks away, it is strongly advised that you read the paper and go through the supplied code in the next 3-4 days, and then begin coding as soon as possible. As usual, plan to spend some time looking at the code already supplied (methods, classes, etc), as you will have to depend on it for your implementation.

Part A:

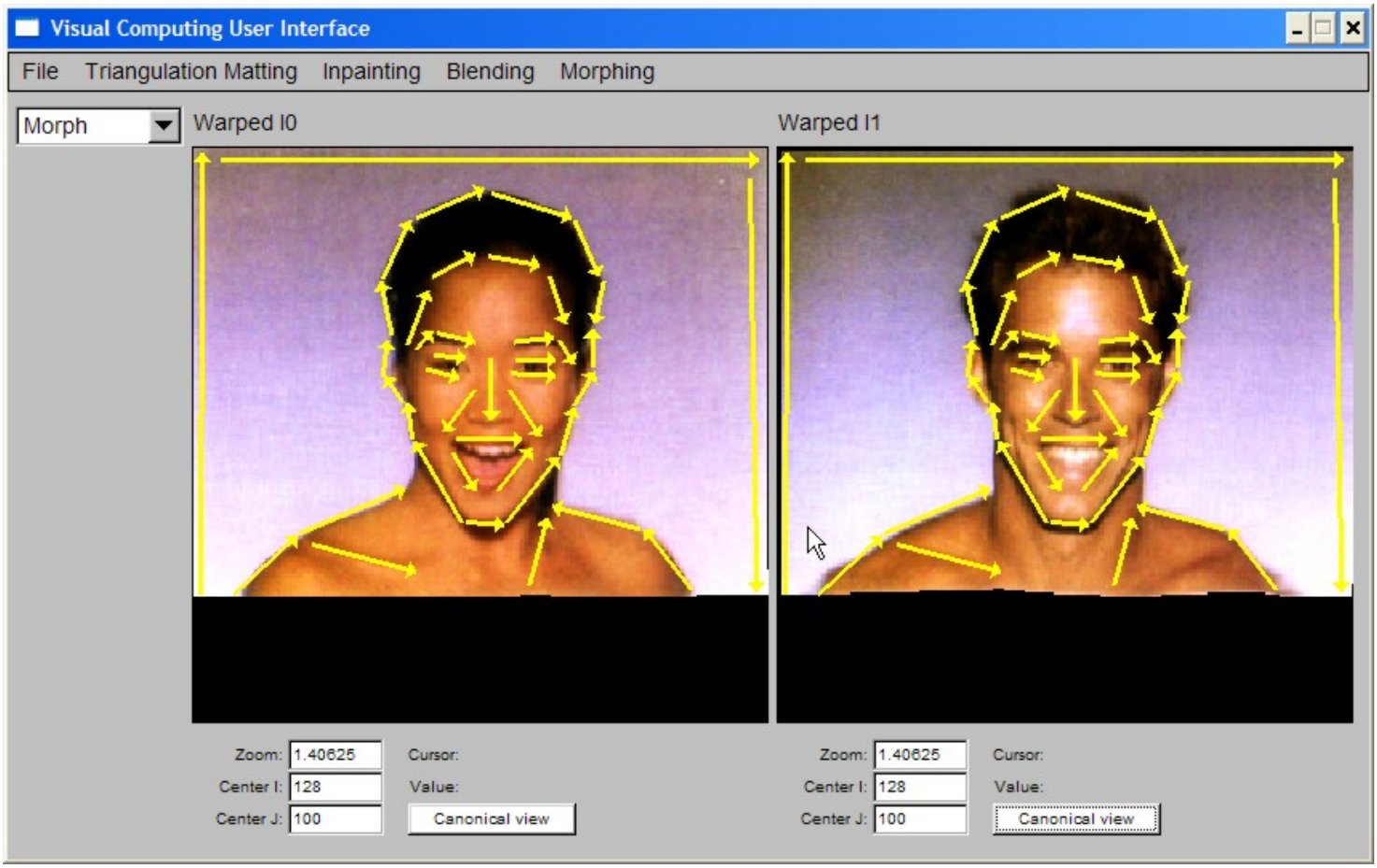
For this part of the assignment, you will be implementing the method as described in the paper by Beier & Neely's 1992 paper on "Feature Based Image Metamorphosis" covered in class next week. Given two source images, *I0* and *I1*, the user specifies a set of corresponding lines in them to establish correspondence between "features" in one image and "features" in the other:





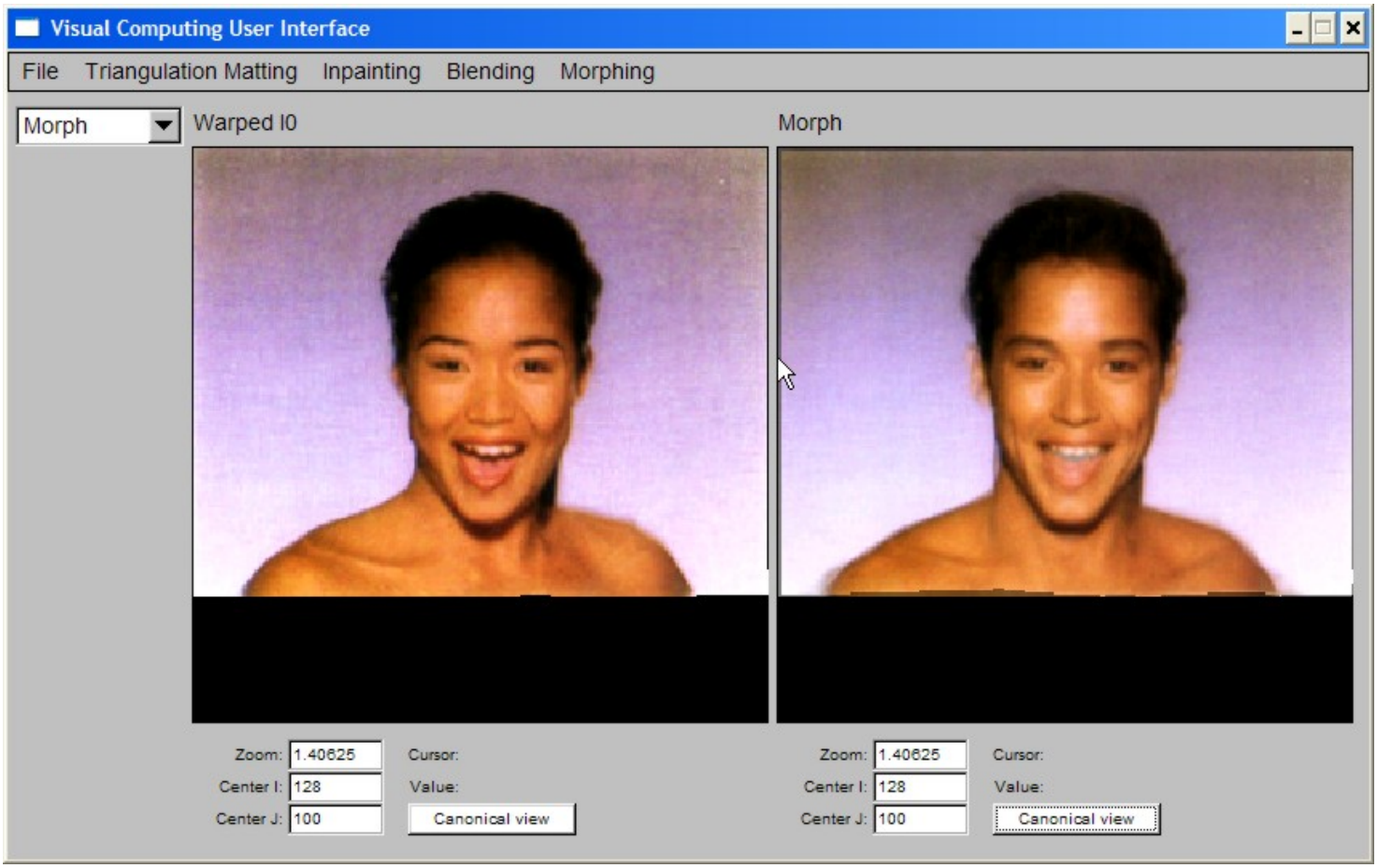
These corresponding "line pairs" are then used to warp the input images so that pixels "follow" the lines closest to them. To produce an intermediate image between *I0* and *I1*, the line pairs are interpolated to create an in-between position for the lines. This is done using linear interpolation with a single interpolation parameter *t*. In this interpolation, *t=0* produces a set of lines whose position is identical to that of image *I0* while for *t=1* they are identical to image *I1*.

Warping the source images toward this interpolated set of lines produces two images, *WarpedI0* and *WarpedI1,* shown here with the interpolated lines overlayed:



Here are the same images without the interpolated lines:

The final morph is just a blend between the two images using *t* as the blending parameter:



The main component of the morphing technique is an image warping algorithm called **Field Warping** that you will have to implement. First, read through Beier & Neely's paper to understand the overall method and the context. Again, as in Assignments 2 and 3, an important skill that you will need to develop while reading research papers such as this one is to focus only on those parts of the paper that you must fully understand to complete the assignment---you can ignore parts of the paper that require background knowledge you don't have, or you can ask me or the TAs to explain those parts of the paper in more detail. Specifically, read Sections 1 & 2 for background and motivation behind the work. The Sections you are asked to fully understand and implement are Sections 3.2 and 3.3.

Your Tasks

For Part A:

For this part you will have to do the following:

- Familiarize yourself with the Field Morphing algorithm
- Implement the missing components of the morphing tool and integrate them into the supplied starter code as described below
- Run your own morphing experiments, by taking pictures with your digital camera (at home, outside, or anywhere else you'd like) and applying the morphing algorithm to them (see Part B of the assignment below)

I suggest you use the above order in tackling these tasks. Specifically:

- Read Beier & Neely's paper right away and, in parallel, run the full implementation to visualize exactly how the algorithm is supposed to work. See the file **320/Assignments/Warp/partA/README_A.1st** for a brief description of how to run it (also available here).
- Take a look at the heavily-commented files *src/morphing/morphing.h* and *src/morphing/morphing_algorithm.cxx*. For easier reference, the notation (variable names, etc) follows the notation of the paper. You should study the classes in *morphing.h* and *linepairs.h* very carefully, as well as the *morphing::compute()* and *morphing::morph_iteration()* routines which are the top-level functions of the morphing implementation.
- When writing your code, you should implement and debug each component piece by piece. I suggest the following order for your

implementation (see Part A.1 below for details):
    - implement the method *morphing::field_warp()* in file *morphing_algorithm.cxx,* that allows a single image to be warped according to a supplied set of corresponding line pairs.
    - test your implementation by first supplying a single pair of lines as input to the algorithm (you can do this easily through the viscomp interface). Try scaling, moving the line in image *I1* to see how the warp behaves.
    - once you are convinced that the single-line algorithm works, test your implementation on a pair of lines. Again, reposition and rescale the lines in image *I1* to test that the algorithm behaves as expected.
    - implement the function *morphing::compute_morph()* in file *morphing_algorithm.cxx* which computes the morph for the current setting of the *t* parameter.
- There are no additions to the UI that you will have to do for this assignment; for the purposes of visualizing/debugging intermediate results of your computations, your best option is to modify the *morph_iteration()* routine in *morphing_algorithm.cxx* to call *field_warp()* directly so that you can save your warped results through the supplied code, without having to finish your entire imlpementation first.

Part A: Programming Component (80 Points)

Part A.0: Unpack the Helper Code

The helper code is packaged into the tarfile warp.tar.gz. The following sequence of commands will add files to your existing CS320 directory under your home directory on CDF and will unpack the code:

```
> cd ~
> tar xvfz warp.tar.gz
> rm warp.tar.gz
```

This will create the directory ~/CS320/Assignments/Warp along with files and subdirectories needed for the assignment. All the code that you turn in should be in those directories as well, exactly as specified in the details below. The relevant pieces of the code are in the directory *src/morphing*.

Part A.1: Image Morphing (80 points)

The goal of this part of the assignment is to implement the two core routines of the Beier-Neely morphing algorithm. The supplied source code adds a lot of "machinery" for interactively manipulating and editing lines, and visualizing the morphing results. The good news is that this code contains all the necessary graphical user interface that is necessary, so you can focus only on what really matters: the morphing algorithm itself. In this respect, the supplied code mirrors the implementation of the inpainting code from Assignment 2 and 3, the file morphing.cxx handles all the mundane interface-related code and the basic defintion of the *morphing* class, and you can pretty much avoid studying it too closely as long as you know the "specs" of the various methods in file morphing/morphing.h. The three files to look at closely are morphing/morphing.h, morphing/linepairs.h and morphing/morphing_algorithm.cxx (which is where your code should go). The top-level routine that calls your code is morphing.morph_iteration(), also located in that file, which runs the morphing algorithm for a single setting of the parameter *t* and saves the results to disk. Your task is to implement two routines: the *morphing.field_warp()* routine that implements basic field warping as described in the paper and *morphing.compute_morph()* which combines two warping steps and a cross-dissolve to compute the final morph for a given value of t. Compared to Assignment 3, there is somewhat less code to write, although you should try and take efficiency into consideration because the *field_warp()* routine can be very slow if not implemented carefully. There may be points deducted for unreasonably ineficient implementations.

- **Your starting point**: The file **320/Assignments/Warp/partA/README_A.1st** gives a fairly complete description of the starter code related to morphing.
- Run the full implementation. Be sure to open the morphing control panel which allows control of the algorithm's parameters and provides the tools to draw lines and to modify them interactively.
- By now, you should be familiar with a lot of the viscomp code already and the new additions mirror some of the code you've seen in Assigment 2 and 3. Three files are the most important, which implement parts of the *morphing* class. These files are under the src/morphing subdirectory:
    - First look at src/morphing/morping.h. This file is where the class methods are defined, which provide the interface between the interactive UI and the morphing algorithm, as well as the interface between the starter code and your implementation.
    - Second, look at morphing_algorithm.cxx, which contains the top-level routines of the morphing implementation (which also call your code).

- Third, look at linepairs.h and linepairs.cxx. These files implement the *linepairs* class which provides the tools for manipulating sets of pairs of corresponding lines (interpolation, addition, copy into a matrix, etc). You will need to use methods of this class in your code, so you should become quite familiar with them.
- The only components missing from the implementation are the functions *morphing.field_warp()*, and *morphing.compute_morph()* methods . **Your task for this part of the assignment is to write the code for these methods.** The methods are called by the routines in morphingl_algorithm.cxx. Detailed specs for these functions are given in that file and in morphing.h. You should be able to know exactly what you have to implement from the comments, parameter lists, and callers of these methods.
- **(50 points)** Implement *morphing.field_warp()* in file morphing_algorithm.cxx
  - A detailed description of the input and output of this method can be found in the file morphing.h. To receive full points you MUST implement it exactly as described in the paper.
  - To compute pixel intensities at non-integer positions in the source image, you must do some form of interpolation. Although many interpolation kernels are possible, you will receive full marks as long as you implement some form of 2D interpolation for this step (bilinear interpolation between the 4 nearest integer pixels is the simplest of all). This interpolation produces reasonable (but not great) results and is what I implemented in the full-blown implementation.
- **(30 points)** Implement *morphing.compute_morph()* in file morphing_algorithm.cxx

- To help debug your warping implementation, you are being provided with two test images called *checker1.tif* and *checker2.tif*. These images show a checker board pattern and should help in visualizing how specific parts of the image plane are warped by your field warping implementation.
- You should be able to run the algorithm in batch mode and save its results using a command-line invocation. The following flags provide the input to the morphing algorithm:

  viscomp -no_gui -morphing -msource0 <I0> -msource1 <I1> -mlines <lines file>

  to save the results to disk, use the following additional flags

  -mwarp -mbase <basefilename for results>

  the first option above makes the program output the *warpedI0* and *warpedI1* images as well.

  Type 'viscomp -help' to see the other options, that allow setting the algorithm's parameters, and creating a whole sequence of morphs between images *I0* and *I1*. You can try converting such a sequence into a movie with a sequence-to-movie converter program (e.g. the program *mpeg_encode* on CDF allows you to create mpeg movies from image sequences). This is not a requirement, but it would make you appreciate your results (and any potential warping/morphing bugs you may have) more clearly.

Part B: Non-Programming Component (20 points)

- (20 points) Grab some pictures of your own to test the capabilities of your morphing implementation (you must not use images submited for previous assignments).

See PartB/README_B.txt for details.

Part C: Bonus Marks (0-20 points)

This is your chance to run wild :) and try to do something creative with the 4 implementations you did in this class (alpha matting, inpainting, blending, warping).

You have two mutually exclusive options (i.e. choose either, but not both):

- **(10 points max):** You may choose to NOT implement anything new, but just be as creative as possible with capturing photos on your own and manipulating them with the tools you created during the course. Your "submission" for this part should consist of one or more "visual effect" images, plus the source image(s), a brief description of how you created the effect and any other information/data needed to re-create the images using the viscomp program (ie. line files you used for morphing, if any, masks, if any, etc). **Only the best 5 entries received will get any bonus marks**, and the only criterion for selection and bonus point assignment will be creativity (I will entertain the possibility of ties).
- **(up to 20 points):** To get more points, and to get these independently of what other students did, you can try to implement a visual effect by utilizing your existing implementations as subroutines. There are almost no rules here, except that

- The submission must involve some kind of implementation (ie. coding) that builds upon 2 or more of the "tools" we covered in class.
- The effect should be non-trivial (ie. creating a routine that just pipes an image through 3 tools one after the other for no good reason will not receive bonus points).
- The image manipulation you create does not need to be general (ie. it does not need to work for all kinds of input images). In fact, it could be specific to whatever sets of photos you chose to apply it on.
- The end result should also be one or more "visual effect" images
- The final marks determination will be based on creativity, visual appeal of the final result, and level of effort.

Important: The bonus part is no substitute for the actual assignment! Submitting a half-finished Morphing implementation with a really cool visual effect does not mean that you will automatically get bonus points to substitute for deficiencies in your Morphing algorithm. The final determination of how many points you will receive will be made in light of what you implemented for the Morphing section. **My suggestion is that you do not tackle this part until you have the Morphing part in reasonably good shape!**

If you make a claim for bonus points, you must provide good documentation/evidence (eg. images, comments and details that explain level of effort in image acquisition or coding) to support your claims. Put all of this in the partC directory.

Part D: Packing Everything Up and Turning It In

Once you are done with the above, edit the file 320/Assignments/Warp/CHECKLIST.txt (also available here) to specify which components of the assignment you have completed, along with notes about parts that you were not able to complete, if any.

Pack up **your portion of the code** with the following commands:

```
> cd ~/CS320/Assignments
> tar cvfz assign4.tar.gz Warp/CHECKLIST.txt Warp/partB/{README_B.txt,*.jpg} Warp/partA/bin/viscomp
Warp/partA/src/Makefile Warp/partA/src/morphing Warp/partC
```

Finally, you should use CDF's assignment submission system to submit your assignment:

```
> submit -c csc320h -a Assign4 assign4.tar.gz
```

Note that the system has been configured (1) to accept only files whose name is *assign4.tar.gz* and (2) to not accept submissions that are more than 4 days late. Just do 'man submit' at the unix prompt to get more info on this process.

In evaluating your assignment, the first thing we will look for are the files CHECKLIST.txt and README_B.txt.

Then we will make sure that your code compiles (just by typing "make" in the partA/src directory).

Then we will run your code on test examples in both the interactive and the non-interactive modes.

Finally, we will look at your code. **It must be well commented:** if the TA has doubts about the correctness of a specific implementational feature and there is not enough documentation/comments for the TA to decide, you will lose points.
Good luck, and I hope you learn a lot and enjoy doing this assignment!