Assignment #3: Image Blending

Date handed out: Mar 5, 2014; Due date: Mar 19, 2014

PDF version

Overview

In this assignment you will implement and experiment with an image blending technique based on the *Pyramid-Based Blending* algorithm to be discussed in class.

The goal of this assignment is four-fold:

- to become familiar with image pyramid representations and in particular
    - to understand their construction using 1D convolutions
    - to appreciate their usefulness for image editing & manipulation, by decomposing images at multiple levels of detail
- to read and understand another very influential computer vision research paper, one that has withstood the test of time (Laplacian pyramids were introduced 31 years ago!)
- to read, understand and partially implement a non-trivial image manipulation technique
- to run your own experiments

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 2-3 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 own implementation.

Part A1: Pyramid-Based Image Blending

Pyramid-based image blending was introduced by Burt & Adelson in 1983 as an effective way to create seamless mosaics. Most recently, a variation of this technique has been used in AutoStitch, a tool for automatically stitching a set of photos into a large image panorama. Pyramid image blending works by blending the Laplacian pyramids of two input photos:

using the Gaussian pyramid of a "mask" image as the alpha matte:

The result of this blend is a new Laplacian pyramid from which we can reconstruct a full-resolution, blended version of the input photos. In the example above, the blended photo is impossible to capture with a traditional camera in one shot, as it has two objects in focus, one on the foreground and one on the background, but not the objects in between:

Since the actual blending step is trivial (it's just standard alpha matting) the most important computations are (1) computation of the Laplacian pyramid of an image and (2) reconstruction of the Gaussian pyramid of an image given its Laplacian pyramid. These computations, in turn, involve two basic functions: the *REDUCE()* and the *EXPAND()* function discussed (or to be discussed) in class. These two functions are mainly what you have to implement for this part of the assignment. You need to read and understand Burt and Adelson's original paper on the Laplacian and Gaussian pyramids. You should read up to, but not including, the section entitled "Entropy". The actual blending algorithm was introduced in a follow-up paper on image mosaics. The only section from that paper you should look at is a very brief Section 3.2 (Splining regions of arbitrary shape), which lists the 3 steps of the blending algorithm. For your purposes, you can ignore the rest of that paper.

As usual, I am providing you with a full implementation of the blending tool in executable form (see the menu options under "Blending"), along with a suite of test images, so that you can run it yourself and see how it works (see the executables partA/bin/VisComp_full.exe and partA/bin/viscomp_full in the assignment zipfile). The program should work for a variety of image formats (tif, jpeg, etc), and the binary masks you can use are identical to those you used for A2 (eg. they can be in .bmp format). In addition to the basic code, I am providing you with a "packing" routine in the source code that allows you to visualize a photo's entire pyramid in a single image for debugging purposes, as in the examples above.

Interactive vs. Non-Interactive (batch) Operation

The supplied program can be run in interactive mode (ie. UI windows are displayed and user works through menus and/or keyboard accelerators), in batch mode (ie. user supplies a set of command-line options specifying the input and output images names) or both (ie. some or all command-line options are specified and, after images are loaded, the UI windows are displayed as well). Type 'viscomp -help' to see the command line options.

Your Tasks

- Familiarize yourself with the Gaussian & Laplacian pyramid construction algorithms, and the pyramid blending algorithm
- Implement the "missing" components of the blending tool and integrate them into the supplied "starter code"
- Run your own blending experiments, by taking pictures with your digital camera (at home, outside, or anywhere else you'd like) and applying the blending algorithm to them (see Part B of the assignment below)

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

- Read the Laplacian pyramid 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/Blend/partA/README_A1.1st** for a brief description of how to run it (also available here).
- Take a look at the heavily-commented files *src/pyramid/pyramid.h* and *src/pyramid/pyramid.cxx*. Notation (variable names, etc) follows the notation of Burt and Adelson's paper. You should study the *pyramid::build()* and *pyramid::g()* routines very carefully, as they are the top-level functions for building the Gauss and Laplacian pyramids
- 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 *pyramid::reduce()* in file *pyramid.cxx* to allow construction of an image's Gaussian pyramid
  - implement the function *pyramid::expand()* in file *pyramid.cxx* in preparation for the construction of an image's Laplacian pyramid
  - implement the function *pyramid::blend()* in file blend.*cxx*
- 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 use the *vil_save()* routine of VXL to save an intermediate image that you created so that you can view it by running an external image viewing tool (eg. 'xv', 'irfanview', etc), or by loading & browsing it in viscomp.

Part A: Programming Component (90 Points)

Part A.0: Unpack the Helper Code

The helper code is packaged into the tarfile blend.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 blend.tar.gz
> rm blend.tar.gz
```

This will create the directory ~/CS320/Assignments/Blend 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/pyramid.*

Part A.1: Image Pyramid Construction and Image Blending (55 points)

The goal of this part of the assignment is to implement the three core routines for Gauss/Laplacian pyramid construction (the pyramid.reduce(), pyramid.expand() methods) as well as the pyramid::blend() routine for image blending. The first three routines take a single image as input and they assume that the image is square and has 2^N+1 rows/columns. The supplied source code pads images that do not conform to this constraint with zeros, so you can safely assume that any image passed to these routines does have these dimensions. Overall, there is much less code to go over, compared to Assignment 2, since the technique itself is very compact and has few steps.

- ***Your starting point**: The file **320/Assignments/Blend/partA/README_A1.1st** gives a fairly complete description of the starter code related to image pyramids.*
- *By now, you should be familiar with a lot of the code already and the new additions mirror some of the code you've seen. Three files*

are the most important, which implement the pyramid class. These files are under the src/pyramid subdirectory:

- *First look at src/pyramid/pyramid.h. This file is where the class methods are defined, which provide the interface between the interactive UI and the pyramid algorithm.*
- *Second, look at pyramid.cxx. This file is where the entire pyramid implementation is located. I purposely did not separate out the pieces that you have to implement into a separate file as it is important to understand how pyramids are represented and manipulated in the code. Also, you should look at some of the exisiting routines for guidance/examples of how to implement the parts that are left for you to do (eg. look at the pyramid.g() method). The implementation, variable names, and comments follow the terminology in the Burt & Adelson paper very closely so you should be able to read the code in parallel with the paper, and* ***refer to the paper if there is something you don't understand in the code****.*
- *The only components missing from the pyramid implementation are the functions the pyramid.reduce(), pyramid.expand() 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 pyramid.cxx. Detailed specs for these functions are given in file pyramid/pyramid.h and pyramid/pyramid.cxx. You should be able to know exactly what you have to implement from the comments, parameter lists, and callers of these methods.*
- *The actual blending routine, called blend() is in its own file, pyramid/blend.cxx. The only code in this file right now is a 'dummy' blending routine called blend2() which takes two images and a mask and blends them without constructing the pyramids. You can try running this routine yourself on pairs of images to compare to the results of the pyramid blending algorithm.*
- ***(15 points)** Implement pyramid.reduce() in file pyramid/pyramid.cxx*
  - *A detailed description of the input and output of this method can be found in the file pyramid/pyramid.h. To receive full points you MUST implement it through a series of 1D convolutions (ie. the w_hat kernel in the paper). Implementing the routine using the equivalent 2D kernel will cause reduction of points. Similarly, pay attention to boundary handling, i.e., when pixels of the convolution kernel lie outside the image boundary. The specific way to handle this (ie. using a reduced kernel that considers only the pixels inside the image boundary) will be discussed in class and is detailed in Sam Hasinoff's "Gauss-Laplacian Pyramid" notes.*
- ***(20 points)** Implement pyramid.expand() in file pyramid/pyramid.cxx*
  - *Same instructions apply as in the reduce() routine---you must implement using a 1D kernel and handle boundaries in an identical fashion.*
- ***(20 points)** Implement blend(). Calling conventions can be found in file pyramid/blend.h and pyramid/pyramid.h*

- *Your starter code contains "dummy" versions of the above methods, which allow the implementation to always produce a "pyramid" image or a "blended" image as output. The only difference your implemenation will make is to produce a marked increase in the* ***quality*** *of the results.*
- *You should be able to run the algorithm in batch mode and save its results using the following command-line invocations:*

  to blend two images:

  viscomp -no_gui -blending -bsource0 <Source0> -bsource1 <Source1> -bmask <Mask> -bblend <Blended>

  to save the gaussian pyramid and/or laplacian pyramids of these image, supply the additional options:

  -bgpyr0 -bgpyr1 -bgpyrm -bgpyrb

  to output the gaussian pyramids of source0, source1, mask and blended image, respectively, and

  -blpyr0 -blpyr1 -blpyrb

  to save their laplacian pyramids of source0, source1, and the blended image.

Part B: Non-Programming Component (10 points)

- *(5 points) Grab some pictures of your own to test the capabilities of your blending implementation, and supply them along with your code.*

See PartB/README.txt for details.

Part C: Packing Everything Up and Turning It In

Once you are done with the above, edit the file 320/Assignments/Blend/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 assign3.tar.gz Blend/CHECKLIST Blend/partB/{WRITTEN,.jpg} Blend/partA/bin/viscomp Blend/partA/src/{Makefile,ADDITIONS} Blend/partA/src/pyramid Blend/partA/src/morphing Blend/partC*

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

> submit -c csc320h -a Assign3 assign3.tar.gz

Note that the system has been configured (1) to accept only files whose name is assign3.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 and WRITTEN.

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.*