

## **Homework 2**

**Due Tuesday, September 15, 2008 at 11:59pm**

In this homework, you will get your hands dirty with 2D raster graphics, sampling and that kind of thing, like we have been talking about in class.

### **1. Bilinear Resampling (30 pts)**

You have been hired at Adobe to work on the new version of Photoshop. They want a complete re-write of their codebase, and you have been assigned the task of writing the new bilinear resampling for scaling images. This is the algorithm that will run on an image when the user selects “Image->ImageSize” and changes the size of an image using bilinear resampling.

You are to write a C or C++ program called `LinearResample.exe` that takes in an image in various possible formats and outputs it at a different size by using bilinear resampling. The executable should take in the following arguments:

```
LinearResample.exe <input> <new width> <new height> <output>
```

Where `<input>` and `<output>` are the input and output images, respectively. The output image should have the resolution given by `<new width> x <new height>` and the images could be in the `jpg`, `tif`, or `bmp` formats as specified by their extension. So for example, I would run the following to scale my `test1.tif` image into a 50 x 25 image which is saved as `out.tif`:

```
LinearResample.exe test1.tif 50 25 out.tif
```

Remember, the output image can be bigger or smaller than the original. The user can also change the aspect ratio. For example, one dimension can be increased while the other is decreased (see the examples below). Your program should produce an output that is visually comparable to that of using the bilinear resampling filter in Photoshop.



To help you get started, I provide you with a Visual Studio solution file called `ImageProc` which reads in an image and saves it back out as a different filename. This code uses the Windows `CImage` class to read and write image, but feel free to write your own if you want (although writing a function to read/write jpegs is probably not a valuable use of your time). You can also use any standard image I/O library if there is another you prefer

or if you want to do your coding on something other than Windows. Just make sure that your final submission compiles and runs on the Windows machines in the lab, so please test it there before you submit and include any of the libraries or dll's you need. Also please note in the comments any code you borrow. You are not to use someone else's code to do the actual filtering. In terms of the filtering, remember that you can use different convolution kernels for upsampling and downsampling of the image.

**To submit:** Just zip up your code and an .exe version and call it LinearResample.zip

## 2. Seam Carving (30 pts)

A popular paper from SIGGRAPH 2007 (and revisited again in 2008) was “Seam Carving for Content Aware Image Resizing” by Shai Avidan and Ariel Shamir. They describe a new way to change the resolution of images without scaling. The basic idea is that you can compute low energy “seams” that extend from one end of the image to the other. Since these seams contain little energy, you can remove them or duplicate them without affecting the appearance of the image. To change the aspect ratio, you can alternate between vertical and horizontal seams.

Adobe wants this to be a new feature in Photoshop CS4. Your task is to write the program SeamCarving.exe which allows you to resize images through the seam carving method. It should have the arguments as follows:

```
SeamCarving.exe <input> <new width> <new height> <output>
```

which are similar to that of the previous function. You can use the ImageProc code to get started.

To begin, you might want to read Avidan and Shamir's seam carving paper, which is available off the class webpage. I recommend you compute the energy function by computing the image differential at every pixel as they recommend. Also be sure to implement the algorithm as a dynamic program since it needs to be fast. Users will not wait for a long time for their image to be rescaled! I will link some images to try on your algorithm from the class webpage. I also want you to try a few of your own images and see if seam carving works for them. It doesn't work for everything so play around a bit with it.

**Extra credit (20 pts):** Write an editor to allow you to adjust the “energy” of images like I showed in class so that you can remove objects from a scene. Basically, create a mouse driven interface that lowers the energy of a particular region, forcing all the seams to pass through it. You might want to add the option of saving out this energy map into a file so another user can try it. Demonstrate that it works by editing a few images.

**Extra credit (20 pts):** Read their new paper (“Improved Seam Carving for Video Retargeting” by Rubinstein et al.) and implement their “forward energy” approach to minimize artifacts.

**To submit:** Just zip up your code and an .exe version and call it SeamCarving.zip. Also include any test images and the extra credit stuff if you worked on it. If you do any of the extra credit exercises, be sure to document what you did in a write-up so that the TA can understand what you accomplished.

### 3. PhotoMosaic (40 pts)

Photomosaics are images composed out of smaller images. In this part of the assignment, you will write a program that creates photomosaics using the database that you helped build in the mini-assignment. The program takes in the following arguments:

```
MosaicMaker.exe <target image> <src image template> <num images>  
<cell width> <cell height> <mini image width> <mini image height>  
<output image>
```

Here, <target image> is the larger image which the mosaic is trying to duplicate. <src image template> is the template for the miniature images that will be used to fill in the mosaic. An example of an appropriate template is Data/image\_%06d.jpg. The next argument <num images> tells the program the number of source images available. <cell width> and <cell height> specifies the size of the cell in the original image that will be replaced with a miniature image. So if these values are 4 and 3, then every 4 x 3 set of pixels in the original image will be replaced by a “mini” image of resolution <mini image width> and <mini image height>. Finally, output image is specified by <output image>.

To start, download the database of images that was built up from the mini-assignment that I will post off the class webpage. If you cannot download the entire database, then use the images you submitted for the mini-assignment for testing until the last step. Our final photomosaics will be limited by the number of images in our database, which isn't too large. However, it is reasonable for the purposes of this assignment.

Then follow these simple steps:

1. Write a program that loops over all the possible locations and finds the best image for each position. You can use an L2 norm to measure the “best” image. The output of this section should be a working photomosaic, although it might not look very good because some of the colors might be incorrect and there will be lots of duplicates. (20 pts)
2. Improve the photomosaic by having your program adjust the color of the mini images to best match the local color of the image at that region. (5 pts)
3. Modify your code so that it takes the input flag `-MAX <max_num>` which will limit the maximum number of times a source image can be used. Of course, you should give an error if `max_num x num_images < num cells in image`. (5 pts)

5. Run a check to make sure that duplicate images are not neighboring each other in the 8 cells adjacent to the current one. (5pts)

4. Make a nice final image for submission into the contest, see below. (5 pts)

**The final mosaic:** To generate your final mosaic, set the size of your miniature images to 256 x 192 pixels. Also, make sure that your cells are big enough so that you don't have too many of them in a single image. For example, in my test image below had 52 x 88 mini images, which yielded a final photomosaic of 13,312 x 16,896 pixels in resolution. To save this out, I had to use the saveToBMP() function call provided in the Image class. This image is simply too big for CImage to handle! Also, if your image gets much larger than this you will start running into memory problems, so be careful. Note that all submissions become property of AGL.

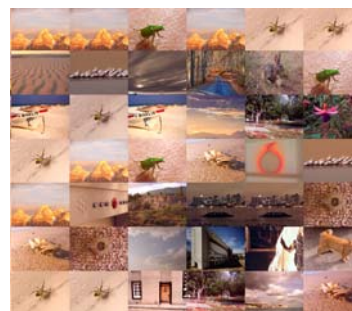
**The contest:** I will judge the final mosaics and select the best one to create a poster for display outside the AGL. Check out last year's winner outside our door!

**Notes:**

The source images are 1024 x 768, so be sure to preserve the 4 x 3 aspect ratio when generating the mosaic. Give an error when the mosaics contains source images of a different aspect ratio.

**Example:**

Here is the photomosaic I generated with a program I wrote in a couple of hours using the data set you all provided. The magnified portion shown to the right is one of my eyes. Can you see it when you squint?



**Extra credit:**

For those of you who breezed through this homework assignment, there are a couple of extra credit options. Implement any you want for bonus points! Be sure to document any extra credit you do in a write-up so that the TA can understand what you did.

1. Non-uniform grid: Instead of rendering the mosaic in a grid as a uniform grid of cells, make the cells staggered in “brick” style or in any other style you can experiment with. Be creative! (10 pts)

2. Scale invariant matching: Instead of having every mini-image at the same size, wouldn't it be cool if some images were larger than others? To do this, you will need to compare the mini images at various scales and translations. You will probably need some fast algorithms to do this, since the “brute-force” approach might be too slow. (20 pts)

**To submit:** Just zip up your code and an .exe version and call it MosaicMaker.zip. Also include your final mosaic submission!

**Submission Instructions**

You are to write your code for these three problems in C or C++ using the Visual Studio environment available on the PCs in the computer lab ECE 211 (or on your own machine, as long as it runs on the machines in ECE 211). The TA will compile your project and run your code and if the code does not compile in Visual Studio you risk getting a 0. You may not use Matlab for the final submission for this project, but you may use it to prototype things if you think it's faster. Matlab is wonderful for quick prototyping, but I ask you to submit your final code in C or C++ because you should be able to write “shipping” code in this class, not just prototype code. Remember: **what you submit must compile!** This homework is due at 11:59pm on Tuesday September 15. This is a hard deadline. Read the syllabus regarding late submissions.

**Tips and advice**

Get started soon – don't wait until the last minute. Also be sure to ask me (and the TA) questions if you have them! **If you don't get everything fully working, please write up some documentation that explains what you have done and what works and what doesn't to help the TA understand your code.** This will save us a lot of time and ensure you get a fair grade. Keep an eye on the website for any corrections announcements regarding this homework. Good luck!