CS 465 Program 2: Resample

out: Monday 11 September 2006
due: Tuesday 26 September 2006

1 Introduction

In this assignment you will implement two different kinds of filtering operations: discrete filtering on an image (to do blurring and sharpening) and image resampling for enlarging and reducing images. We provide a framework that handles loading images, displaying them on the screen, and choosing an operation and filter type. You provide implementations of the filters and the filtering algorithms.

2 Assignment Overview

The user interface provided by the framework allows the user to load a single image, and it has two modes: one for filtering the image while keeping its dimensions fixed and one for resampling the image (enlarging it or reducing it to fit the window as it is resized). In each case, the framework calls the filtering code every time the image needs to be recomputed, giving it the single stored image and displaying the result.

Your work breaks down into two main parts: image filtering and image resampling. They are completely independent (since the code from one never calls the code from the other). For each of these parts you have to implement several filters. This does not mean you have to write separate code for all of them! Aside from point and box resampling, Each of those parts has two subparts: implement the filtering operation for a general filter, and write code to evaluate the individual filters. Once you have the brute-force version of the filtering algorithm working, you will take advantage of the separability of the filter functions to improve performance. You will see a drastic improvement for large filters.

We stress that that this assignment will be conceptually difficult rather than being difficult to program. Our final versions of the general filtering/resampling methods are collectively less than 150 lines of Java. Whether your implementation is longer or shorter is not part of the assignment, but be aware that working without a careful plan can lead you to do more work than necessary. We strongly suggest that you work a couple of pencil and paper examples of resampling and filtering (think Homework 3) before starting to code. If you plan it right you will find that the code for the filtering and resampling operations is quite similar.

3 Requirements

The requirements of the assignment are:

1. Apply discrete filters to images. The framework asks for this by calling the filterImage method on the currently selected subclass of DiscreteFilter. You should be able to implement this with a general algorithm in DiscreteFilter.filterImage.

2. Resample images using continuous filters. The framework asks for this by calling the resampleImage method on the currently selected subclass of ContinuousFilter. You should be able to implement this with a general algorithm in ContinuousFilter.resampleImage, except for the box filter, which is most easily handled by separate code in BoxFilter.resampleImage.

3. Take advantage of separability for improved performance. This applies to both types of filters; to do this, you implement the filterImageSeparable method for discrete filters and the resampleImageSeparable method for continuous filters. Again, all cases other than the continuous box can be handled by single methods shared by all the filters of each type.

4. Implement the following discrete filters:

   (a) Box blur
   (b) Gaussian blur
   (c) Unsharp mask

   For the gaussian and the unsharp mask, which is derived from a gaussian, the support radius should be 3σ where σ is the gaussian's standard deviation. The unsharp mask filter is described in more detail later in this handout.

5. Implement the following continuous filters:

   (a) Box
   (b) Tent
   (c) B-spline
   (d) Catmull-Rom
   (e) Mitchell-Netravali

Chapter 4 of Shirley describes in detail all of the filters that you will implement for this assignment.

4 Framework

In this section, we provide an overview of the framework code for this assignment.

4.1 The GUI

Out of the box, running MainFrame.java opens the assignment GUI application. You can load images and save results using the File menu. The application has two modes, depending on whether you select a continuous or discrete filter from the list of available filters.

For a discrete filter you can set its parameters, if it has them, and apply the filter. If the function has parameters a separate sub-window will appear automatically when the function is selected. You can enter the parameters in this window in text fields next to an Update button. **The fields in the cooresponding filter object will be set after (and only after) the Update button is clicked.** The Filter button will apply the filter and display the result. The Apply button will make the current window contents the source image. Any future filtering and resampling operations will then be performed on that image.

Selecting a continuous filter puts the GUI in Resample mode. The original image will be resampled to fill the current window and will be again resampled from the source every time the window size is changed. The filtering buttons will disappear in this mode. Instead you can check a box to hold the aspect ratio constant during resizing.

Finally, clicking the Separable Mode box will cause the GUI to call the seperable versions of the filtering and resampling methods.

4.2 MainFrame **and** ScrollingPanel

MainFrame is the root class of the application. It defines and controls the GUI interface. ScrollingPanel defines some of the properties of the image viewing window. You don't need to change either of these classes.

4.3 Color**,** Image **and** ResizableImage

The Color and Image classes are the same as in the ray1 assignment. ResizableImage is a subclass of Image that supports a resize operation that avoids unnecessarily allocating new memory during resizing. Changing the size of an existing image is quicker than discarding the image and allocating a new one. Note that the old data will still be sitting in the image after a resize operation, so don't assume the image is initialized to any particular value.

4.4 FilterFunction

FilterFunction is the base class of all the filter functions used in this assignment. Its purpose is to provide a mechanism for the GUI to learn about a filter's parameters and set them from the text boxes in the window. You shouldn't need to change this class.

4.5 DiscreteFilter **and subclasses**

Discrete filters have an integer radius of support, and they can only be evaluated for integer arguments. The best way to handle filtering by all the required filters is to implement filterImage and

filterImageSeparable in the base class Discrete Filter, and only override getSupportRadius and evaluate in the subclasses.

Note that discrete filters have a variable size: for example, changing the radius parameter of a box filter causes its support radius to change.

4.6 ContinuousFilter and subclasses

Continuous filters have a real-number radius of support, and they can be evaluated for any real argument. The best way to handle resampling images using all the required filters is to implement resampleImage and resampleImageSeparable in the base class ContinuousFilter, and only override getSupportRadius and evaluate in the subclasses.

Note that the continuous filters all have fixed support radii. The radius of the filter relative to the input and output pixel grids depends on the resampling ratio, but the filter subclasses don't need to know this.

The framework contains an implementation of a very simple (and bad!) resampling method in NearestNeighborFilter, so that you can get an image on the screen right off.

5 Implementation Details

5.1 Normalization, or, What to do at the edge?

In class, several methods were described to handle the cases where the filter would fall off the edge of image, such as wrapping around to the other side, reflecting across the border, and using the closest value. For this project we want you to renormalize the convolution operation. You will divide the the final convolution value at a point by the sum of the filter weights used for samples under the filter support at that point.

For example in one dimension, the resulting equation for a convolution is:

$$I'[j] = \frac{\sum_i I[i] f(j-i)}{\sum_i f(j-i)}$$

where $I'[j]$ is one output sample at j, I is the original set of samples, f is the filtering function and r is the radius of the filter.

5.2 Separability or Why are all the filter functions 1D?

It should be clear that to filter images you will need 2D filter functions and all of the filtering function code seems to imply only 1D functions. As noted in class, this is because all the filter functions are seperable. For a seperable function $h_2(x, y)$ in 2D:

$$h_2(x, y) = f_1(x) g_1(y)$$

where f and g are 1D functions that describe the filter's behavior separately in each dimension. For this assignment, each of the FilterFunctions will be used to construct a separable 2D filter by using the function for both f and g in the above equation.

5.3 Coordinate systems, or, How to I make sense of all this sample stuff?

Perhaps the hardest part of this assignment is defining a consistent coordinate system for resampling operations. This will be vital to correctly finding the samples that fall under a particular filter's support and for calculating correct filter values. We suggest the following: always work in the coordinate system of the source image. A good layout for doing this is to imagine that the source image is a grid with samples taken at the centers of each grid cell. Align the lower left corner of the grid with the point $(-0.5, -0.5)$. If you assume the pixel spacing of the source image is the same as the spacing on the axes, the lowest leftmost sample should fall on the point $(0, 0)$ and all of the other samples should lie on integer coordinates (see Shirley Figure 3.1).

Using this grid concept you must work out how to calculate two different pieces of information. First, determine a method to find the source image samples that lie within a rectangle centered an arbitrary point in the source image. You will not want to assume that this point will have integer coordinates. Second, work out a way of converting points in the output image to points in the input image. A good visualization to use here is to imagine the output image as another grid that has exactly the same boundaries as the input grid but with more or fewer pixels in each direction. Of course the output grid cells will be different sizes than the input grid and they will most likely not even be square. However, you should be able to use the ratios of the input and output image sizes to determine how big the cells are relative to the input spacing, and from there you should be able to determine the input coordinates for an arbitrary output image point.

5.4 Canonical Sizes, or, How do I deal with different filter sizes?

When you use a filter for resampling, you need to scale its size to match either the input pixel grid (for upsampling, or enlarging) or the output pixel grid (for downsampling, or reducing). To see why this is, think about what would happen if your filter was sized according to the output grid when upsampling by a factor of 100 (it would only occasionally see an input sample fall within its support) or if it was sized according to the input grid when downsampling by a factor of 100 (it would see only a few input samples for each output sample, ignoring everything in between the output samples).

Here is how we like to deal with these varying radii in the code. We think of ContinuousFilters as having a "canonical" radius. The canonical radius tells you the size of the filter only *relative to other filters*. Thus a box filter has a canonical radius of 0.5 while a Catmull-Rom Cubic Filter has a canonical radius of 2. However, the canonical radius *does not* tell you how big the filter is relative to the image sample spacing, since this depends on the resampling ratio. The getSupport method should return the canonical radius of the filter and the evaluate method should assume its input values are relative to the canonical radius. When you call evaluate in your resampling code, you will need to scale the offset appropriately depending on the size you need the filter to be.

5.5 Sharpening filters, or, Why do Unsharp Masks sharpen images?

One common image processing operation is to sharpen the image. This is often done by subtracting a blurred version of the image from the original. Because of how this was once implemented in analog photography, filters that do this are generally called Unsharp Masks.

The discrete filter that corresponds to this blur-then-subtract process is the difference of an impulse

(a filter with a single nonzero entry at $[0, 0]$) and a gaussian filter. It has two parameters, σ, which is the standard deviation of the gaussian component, and α, which is a weight that determines the balance between the original image and the blurred one:

$$\text{unsharp}[i] = (1 + \alpha)d[i] - \alpha g_\sigma[i]$$

where $d[0] = 1$ and $d[i] = 0$ for all nonzero i, and $g_\sigma[i]$ is the vaule of the gaussian filter with standard deviation σ.

5.6 Rounding, Integer math, and Java, or, Why does Java hate me?

At many points in this assignment, you will have to compute floating point values from integers and vice versa. Java has some quirky aspects that can make it easy to make simple mistakes. It is good to remember that:

- Java uses integer division for integer operands. So, 3 / 7 is 0 to Java but 3.0 / 7 or 3 / 7.0 is 0.4285714.
- The precedence of a cast is higher than the numerical operations. So (int) 1.51 * 2 = 2 because the cast occurs before the multiplication. It is good to always get in the habit of explicitly using parenthesis to define the range of the cast, (int) (1.51 * 2) = 3.
- Java uses truncation to cast values to int. So 4.9999999 and 4.0000001 both cast to the integer 4. It is very common for numerical accuracy to affect the cast value one way or another.

5.7 Final notes

As part of the assignment, you must to determine the correct scaling factor to use when resampling an image. As a rule of thumb, one canonical filter unit should be as large as one pixel spacing in the input image or the output image whatever is larger. This means that in the enlarging or upsampling case where output pixels are smaller than input pixels, one canonical unit will equal one input pixel spacing and in the reducing or downsampling case where output pixels are larger than input pixels, one canonical unit will be as large as one output pixel spacing. Also even though we think of the filter size both in terms of input and output pixel spacing, it will be convenient to consistently measure only according to the input spacing. So in this latter case, you would want to convert the one output pixel space into input pixel units.

Finally, for filters with some negative parts, it is entirely possible for the convolution value at a point to be greater than 1 or less than 0. In order to avoid strange artifacts, you will have to clamp the output pixel values to this range just before finally storing them in the image.

6 Submission and FAQ

Submission will be through CMS. A FAQ page will be kept on the course web site detailing any new questions and their answers brought to the attention of the course staff.