Angela Lin
6.865 Pset 10 Writeup: "Non-photorealistic Rendering: Painterly Rendering"

Motivation

For pset 10, I chose to implement the old "Non-photorealistic Rendering: Painterly Rendering" pset. I chose to implement this pset because I saw the resulting images in the pset handout and thought that they looked really neat! As someone with absolutely no artistic abilities, I thought that it would be especially exciting to use computational photography to create images that look as if they were hand-drawn or hand-painted.

Background

As described in the pset handout, "Painterly Rendering" refers to the process of taking source images and rendering them such that they look as if a painter had created them. This process is primarily introduced and discussed in the "Painterly Rendering with Curved Brush Strokes of Multiple Sizes" by Hertzmann, who presented a method for painting an image using a series of spline brush strokes, the color of which are chosen to match colors in the source image. His painted image uses a multiscale algorithm and is comprised of multiple layers, where each layer is painted using a different brush size to express various levels of detail.

In a related "Image and Video Based Painterly Animation," Hays and Essa introduce an automatic image-based rendering approach to generate painterly animations by using video analysis to manipulate brush strokes. The algorithm creates the first frame of the video sequence using the same algorithm used for still images, and then for each successive frame, paints over the previous frame using difference masking, or painting only in portions of the video which contain significant motion. This allows us to obtain an animation where unchanging regions of the video frame will be left unchanged, producing a video with the appearance of a painting that has been repeatedly painted over and photographed in a "paint-on-glass" style.

Another similar method that seeks to automatically transform video segments into animations that have a hand-painted look is presented in "Processing Images and Video for An Impressionist Effect" by Litwinowicz. While Litwinowicz has an arguably more difficult problem to solve by dealing with temporal changes and has a larger focus on how to move, add, and delete brush strokes from frame to frame using optical flow fields to push strokes in the direction of pixel movement, his technique of rendering an impressionist painting is similar to that described by Hertzmann: strokes could either be entirely generated from scratch or rendered using a textured brush image, as we do in the pset. Then, the orientation of the brush stroke is determined using the value of the gradient direction when the magnitude of the gradient is not close to zero.

Algorithm

The algorithm given in the pset handout for creating a painterly rendition of a source image is as follows:

1. **Sharpness Map:** First, determine the sharpness map of the image. The sharpness map of the image determines where the image has strong high frequencies.
2. **Orientation Extraction:** Using the eigenvector of the structure tensor (used for Harris corner detection) of our image, we can determine the direction of maximum and minimum variation due to edges and corners. Since we want our strokes to align with edges and oriented structures, we must find the smallest eigenvector. Compute the angle between this eigenvector and the horizontal line to determine the local edge orientation and thus the angle at which a stroke painted at this pixel should be drawn.
3. **Paint Brush Strokes:**
    a. First, rescale our paint texture to the desired size.
    b. If an importance map is specified, then we will vary the density of the strokes according to the map, such that the density of strokes is proportional to the map. For each pixel, we draw a random number and reject the sample if the number is less than the value of the importance map at that pixel.
    c. For each of the number of samples desired, randomly generate a (x,y) coordinate and determine what color our brush should be at that coordinate by finding the color of the image at those coordinates. Optionally, we can modulate the color with some noise for a more natural effect. Splat a paint brush stroke centered at (x,y).
4. **Two-scale:** Repeat step 3, but this time, also decrease the size of our paint texture by a factor and only paint in areas that are specified as high frequency areas by our sharpness map. This allows us to emphasize and paint important details within the source image, bringing the image more to life.

Implementation

I implemented the algorithm as described above and in the pset handout by implementing single-stroke paint splattering, single-scale painting, two-scale painting, and finally, oriented two-scale painting (for 6.865).

As an extension, I also implemented light-to-dark and dark-to-light ordering, where we paint the strokes in order of increasing (or decreasing) luminance. This was inspired by Northam's paper, and the idea that a common technique artists use is to paint starting with the lightest colors, and work towards the darkest because objects that disappear into the horizon are often lighter colors, and items at greater depth are generally painted before items that are near. Furthermore, it is much easier to darken an image than it is to lighten it since lighter paint colors are not usually opaque, so if lighter color is painted over a dark, more than one coat may be required to get the desired color. This extension involved storing all of the brush strokes that would be generated from our original painting algorithm, sorting them by luminance value, and applying each stroke based on its luminance value.

Another natural extension was to increase the number of scales we use to paint. I implemented a multi-scale oriented painting algorithm by changing the value of sigma and truncate for our sharpness map to correspond to the level of detail we want. As I narrow in on the higher frequency areas of the image, I decrease the size of my brush stroke accordingly, to generate a multi-scale painting.

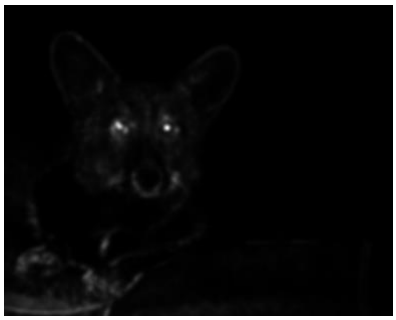
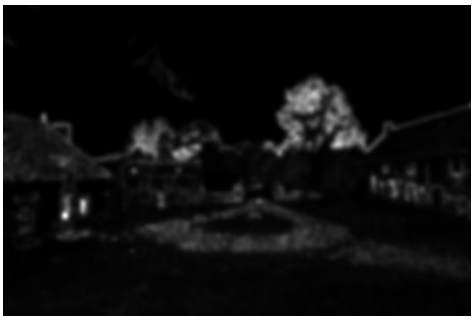
Overall, the most difficult part in implementing this pset was having to translate the Python code as I do not understand numpy very well; there were many incidents where I had bugs because I misunderstood the numpy syntax! It was also very difficult trying to verify whether or not my implementation was producing the correct results because I was used to the thoroughness of previous psets that provided expected outputs for each step of the algorithm.

Test Cases

Single scale painting: simplistic rendering by splatting a paintbrush of an uniform size at random locations:

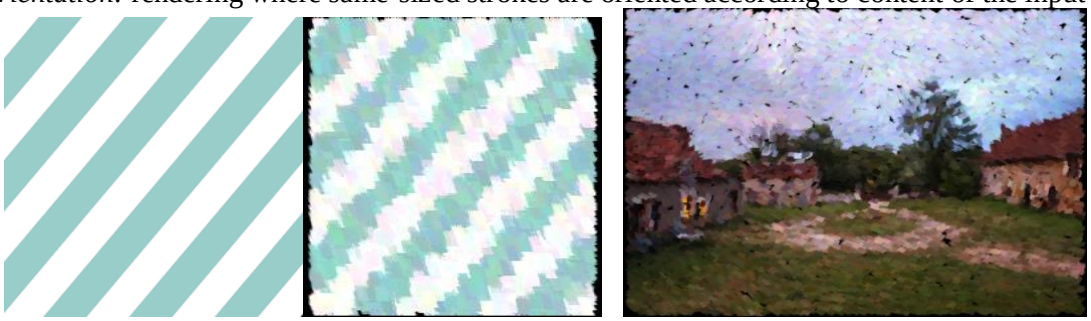



Sharpness mask: determine areas of the image with strong high frequencies

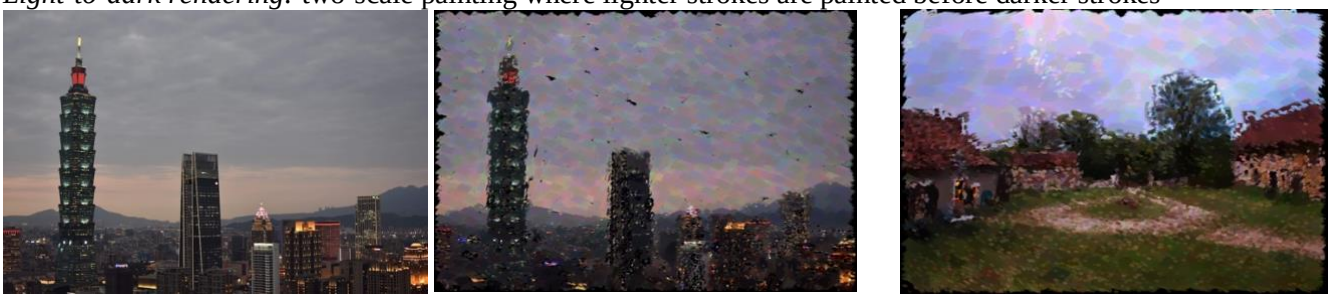
Two-scale rendering: rendering where first layer of coarse strokes is used, followed by a refinement with smaller strokes in regions of high detail

Single-scale Orientation: rendering where same-sized strokes are oriented according to content of the input image



Oriented painterly rendering: two-scale painting where strokes are oriented according to the content of the input image

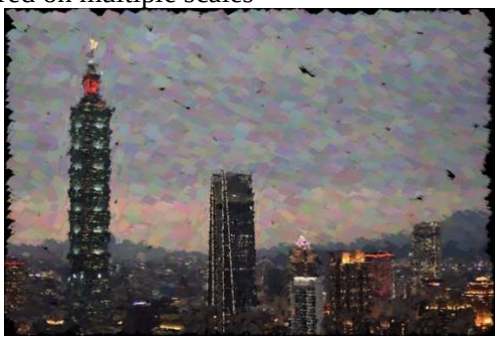
Light-to-dark rendering: two-scale painting where lighter strokes are painted before darker strokes



Dark-to-light rendering: two-scale painting where darker strokes are painted before lighter strokes



Multi-scale rendering: painting where brush strokes are rendered on multiple scales

Final Results

Here are some my favorite results that were not previously included:

References

Aaron Hertzmann and Ken Perlin. Painterly Rendering for Video and Interaction. In *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering*, ACM SIGGRAPH / Eurographics, 7–12. June 2000.

Aaron Hertzmann. Painterly Rendering with Curved Brush Strokes of Multiple Sizes. In *Proceedings of SIGGRAPH 98 Conference*, pages 453–460, July 1998.

Peter Litwinowicz. Processing Images and Video for an Impressionist Effect. In *Proceedings of SIGGRAPH 97 Conference*, pages 407–414, August 1997.

Northam, Lesley. Painterly Rendering. http://www.cgl.uwaterloo.ca/lanortha/CourseProjects/Painterly/painterly.html