Jacqui De Sa
December 10th, 2015

6.865 Final Project Deliverables

Project Name: Painterly Rendering

(Extension: Non-Anisotropic Gaussians and “Cross Stitching” Effect)

Stanford Old Student Union (with cross stitch oriented brush strokes)

Implementation

For my project, I implemented the non-painterly rendering pset from Fall 2013 (https://stellar.mit.edu/S/course/6/fa13/6.815/homework/assignment11/) with the graduate version extension. As an add-on to the project, I also implemented the fast anisotropic gaussian filtering method described and provided an option of rendering images in a cross-stitching style by painting brush strokes perpendicular to edges in the image.

The aim of this pset is to be able to take an input photograph and output a version of the photograph styled as if it were painted. The algorithm is composed of a few different functions. The fundamental underpinning of the algorithm is the brush function, which takes a png of a white-on-black brush stroke and a given color and applies the brush stroke with that color to a given image. The brush algorithm is then extended by the function singleScalePaint, which randomly draws brush strokes at different locations, choosing the brush color based on the color of the original image at the new brush stroke location (with some variation due to noise to make the painted version more interesting). The density of these brush strokes is varied based on an importance map which can be used, for instance, to indicate areas of the painted image which should have more dense, detailed strokes (such as edges) by mapping each pixel to a threshold for how likely brush strokes should be painted or rejected at that given location. By doing edge detection (either by using a sobel filter, an extended difference of gaussians, or a sharpnessMap as suggested in the pset), we can create an edge importance map and create a painterly rendering of an image by first using large brush strokes to paint the whole image (using a constant importance map) and then finer brush strokes to go back over the edges as indicated by the edge importance map.

Painterly rendering of Stanford Old Student Union

The graduate portion of the assignment, Oriented Painterly rendering, uses the eigenvectors of structure tensor to calculate which direction a given brush stroke should be to follow edges and orientations within an image and then paints in that direction. The effect produced ended up looking much more convincing than the simpler painterly method above (see comparison below) since the edge-preserving strokes make the result seem much more natural.

Oriented Painterly rendering of Stanford Old Student Union

One minor extension I made to the pset at this point was to create two different brush effects -- the one suggested by the pset (in which strokes follow edges and orientations) and another in which brush strokes are perpendicular to edges and orientations, creating a stitching effect as seen below. This effect was simply created by rotating selecting an angle 90° to the angle to the horizontal corresponding to the minimum eigenvector of the structure tensor (example image below contrasted with the normal brush effect).

Left: Sephiroth (Kingdom Hearts 2 version), with stitching effect (especially noticeable on the outlines of the hair) vs edge-preserving brush stroke effect (right)

Another effect I experimented with was trying to use anisotropic gaussians. I was first thinking of implementing an extended difference of gaussians as described by Winnemoller et all in my background material, but found it much easier to implement the anisotropic gaussian filtering. I approached anisotropic gaussian filtering by first 1-D gaussian filtering my input image, then rotating it an arbitrary acute angle, than filtering it again, then subtracting the result from a rotated version of the original image, then rotating the final high frequency rotated image back to the original orientation. I had some issues with the rotation step cutting off portions of the image, however (see below) and did not finish implementing an extension to the rotation to fix this so I didn't integrate it into my painterly rendering.

Ansiotropic Filtering Example — see the cut off on the right boot

Background Section

XDoG: Advanced Image Stylization with eXtended Difference-of-Gaussians -- Holger
http://dl.acm.org/citation.cfm?id=2024700
Winnemoller
2011

This Explores the Difference-of-Gaussians operator and artistic applications of it compared to other techniques. The Difference-of-Gaussians (DoG) can be used as an edge-detection function, with possible extensions such as variable thresholding (adding additional parameters to control sensitivity, tone-mapping, and thresholding). The paper goes on to define the flow-aligned difference of gaussians, which locally adapts the DoG to image content (similar to a single-step recursion with one step weakly estimating edges by using image gradients, and a second edge-detection pass), which can potentially be combined with the extended DoG. The DoG is not capable of abstraction/indication (prioritizing which edges are more important) but is well suited to conveying motion lines, ghosting, and negative edges. The paper additionally describes the use of the DoG to mimic stylistic rendering (including hatching).

Fast Anisotropic Gauss Filtering
https://ivi.fnwi.uva.nl/isis/publications/2003/GeusebroekTIP2003/GeusebroekTIP2003.pdf
Jan-Mark Geusebroek
2003

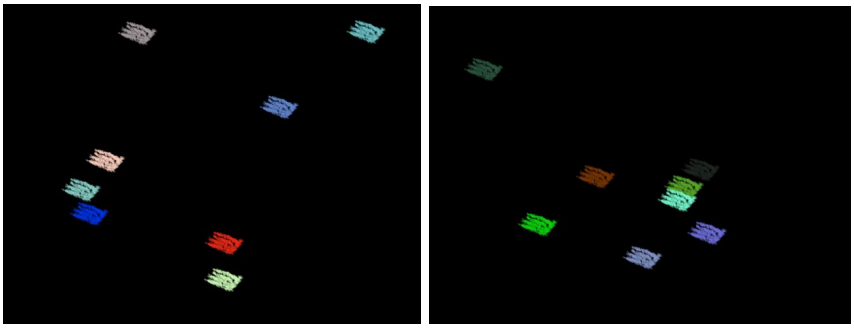
This paper describes a method for fast anisotropic gaussian filtering by combining a 1-D gaussian filter in the x direction direction and in then another 1-D filter in some other direction which is non-orthogonal to the first one. An anisotropic Gaussian filter enables fast creation of edge/ridge maps with good spatial and angular accuracy, and is better at preserving information along edges/lines compared with isotropic gaussians. Since adapting the traditional method of orientation analysis (using steerable filters) to approximating anisotropic gaussians requires a large number of non-separable basis filters, the traditional method is highly computationally intense and so the method described in the paper (of two separable 1-D gaussians) is preferred. Two methods of modelings the anisotropic gaussian are presented, one based on a convolution filter (which is more useful in locally steered filtering), and one based on a recursive filter (which is faster than the convolution filter implementation but produces larger error, and more useful for smoothing or application to the whole image rather than a small portion).

Customizing Painterly Rendering Styles Using Stroke Processes
http://www.stat.ucla.edu/~mtzhao/research/stroke-processes/
Mintian Zhao
https://stellar.mit.edu/S/course/6/fa13/6.815/homework/assignment11/
This paper discusses modifications to painterly rendering which allow customization of the styles used for painterly rendering by mapping perceptual characteristics (like how vibrant an image is) to parameters used for rendering. The paper breaks down perceptual dimensions into eight different rendering parameters: stroke density, non-uniformity (strokes very dense in some places but very sparse elsewhere), local isotropy (the similarity of stroke orientations within an image region), coarseness (stroke size), size contrast (size differences between each stroke and its neighboring strokes), lightness contrast (color lightness change between each stroke and its neighbors), chroma contrast (color change between each stroke and its neighbors), and hue contrast (hue change between each stroke and its neighbors). The paper focuses on local isotropy, size contrast, lightness contrast, chroma contrast, and hue contrast through a novel pipeline of stroke processes. The stoke process pipeline follows a two level approach -- one that uses an interactive segmentation model to break a picture into different regions to paint, and another level which allows user customization of the eight parameters based on sliders. After user customization, strokes are computed first by running a layout process for stroke positions (based on the density and non-uniformity parameters), building a stroke neighborhood graph (where each stroke is a node, and continually changes), and running three local attribute processes on this graph relating to stroke orientations, sizes, and colors (as per the parameters).

Test Cases

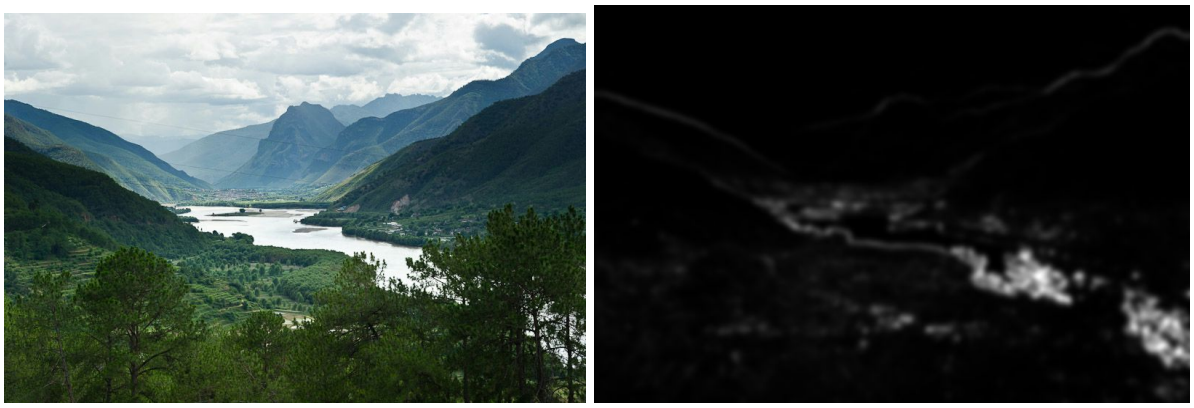
testBrush()

Test brush checks the fundamental underpinning of whether the brush function (described in the intro) works. Given a brush texture, this test should produce an image that should be visually confirmed to have randomly placed different brush strokes of the provided brush textures, in different colors, in the same orientation, and should be in different places when rerun. The images below are two examples of this test being run successfully.



testSharpnessMap()

This tests checks functionality of the edge detection used to create the importance map in the second level pass of the painterly function. The result should appear as a somewhat blurred edges of the image. The image below is the result for the china image (original image, left, to sharpness map, right).



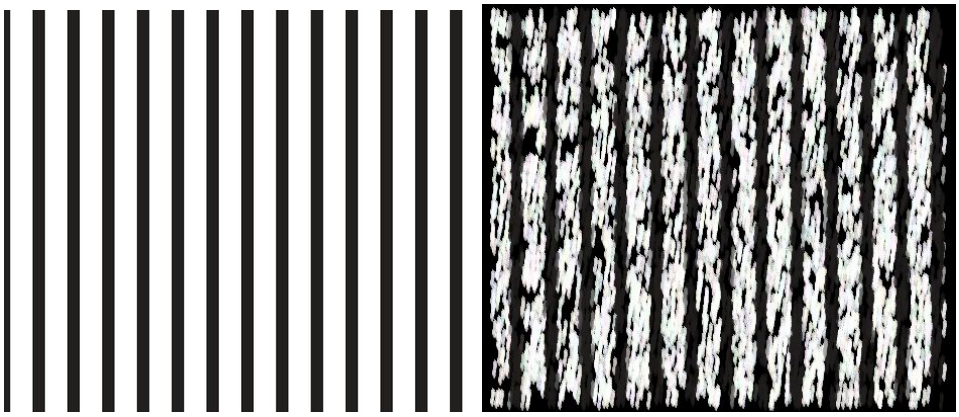
testPainterly_Sephiroth()

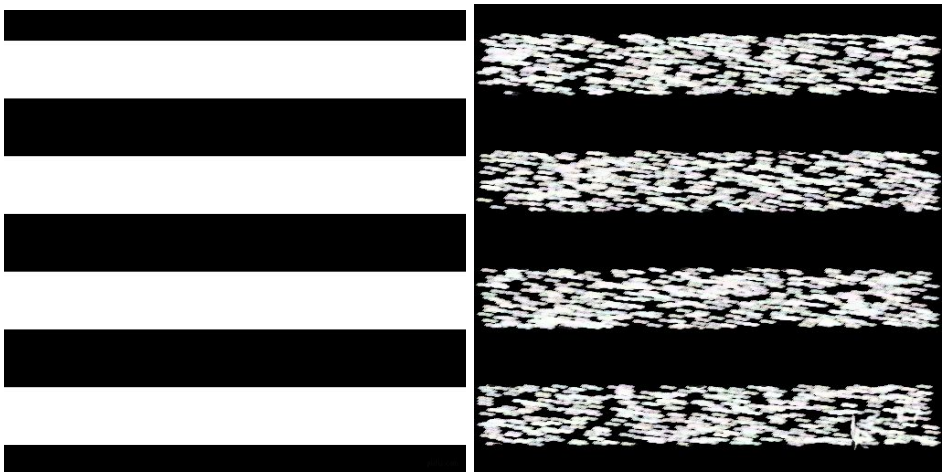
This tests checks that the painting algorithm is painting as expecting with a single brush orientation -- that is, for a given brush stroke, brushes in the output image should align with the original brush stroke orientation, and some noise/differentiation in brush stroke colors occurs in the image. The image I picked for this test has a significant portion which has a white-ish background, making it easy to see if the “noisy” strokes are actually occurring in that area in addition to visually confirming that brush strokes are following the right orientation and color

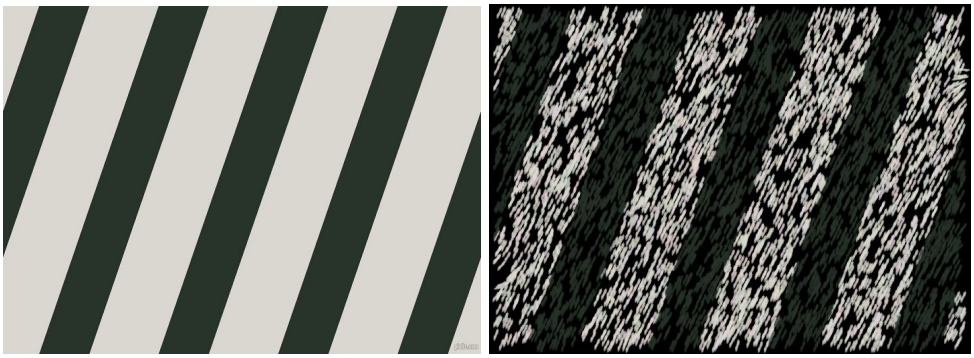
.

testOrientedPaint_EdgeAlignment()

This tests checks that the oriented painting algorithm is indeed painting along the edges of an image. For this test, I provided three test images with different line contours, a small stroke size, a long brush and relatively few strokes in order to make sure my brushes were actually aligning with the edges. The three test images and their correspondences are below:

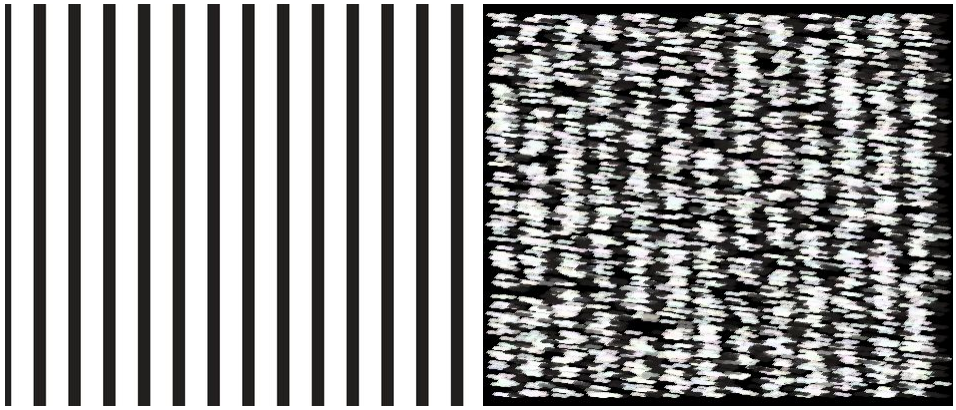


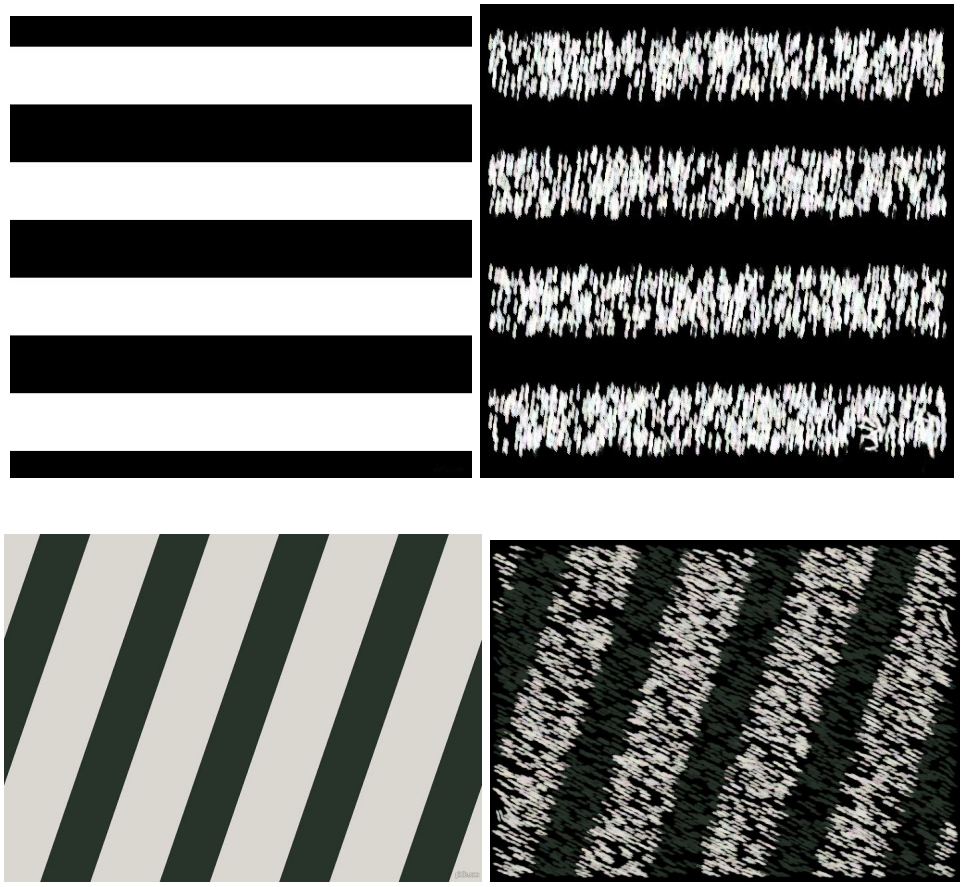




testOrientedPaint_CrossStitchEdgeAlignment()

This test has the same function as the previous one, just checks that that outputted strokes run orthogonal to the edges as expected.





testOrientedPaint_Stanford()

Putting it all together, testOrientedPaint_Stanford() runs the orientedPaint algorithm both for regular and cross stitch brush strokes. I chose the stanford image to run this test on since it is easy to see the effect of the stokes on the roof of the building (a clean edge) and on the palm trees.

Stanford, Cross Stitch

Stanford, Original