John Lenhard - 108265150
Lisa Xu - 108059610

527 Final Report: Texture Consistent Shadow Removal

Summary:
http://web.cecs.pdx.edu/~fliu/project/shadrm.htm
http://web.cecs.pdx.edu/~fliu/papers/eccv08.pdf
The aim of this project is to reproduce results given in the paper: remove a shadow from an image that's been fuzzily traced by the user, while maintaining proper detail around the edges of the shadow, and keeping texture consistent within the shadow compared to the lit area.
The basic user steps of interaction will be:
Take in an image that has a shadow that should be removed.
Trace the boundary of the shadow with a wide brush (doesn't have to be pinpoint in accuracy).
Process the image and return the resulting shadowless image.

Resources Required:
Matlab and related image toolboxes (fuzzy tracing)
Sample images for shadow removal

Major Necessary Components:

Fuzzy image tracer
-Supply a black/white image mask traced in Gimp/Photoshop

Estimate illumination change in between the shadowed area and lit area
-use this estimation to both locate the shadow boundary and estimate the change in lighting at the edge simultaneously, in order to preserve detail around the edge

Estimate the effect of the shadow on texture details
-Match gradient characteristics of shadowed area to unshadowed area: for example, shadowed areas may have less noise but more detail. We want these characteristics to match the noisier unshadowed area.

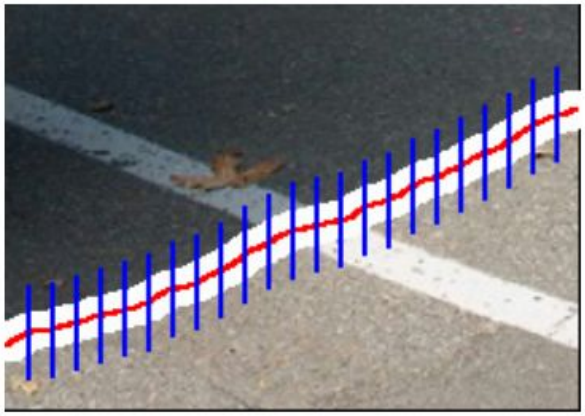
Our approach:
First, we traced the image shadow edge with a large inaccurate brush, using a shade of grey. The area outside the shadow trace was filled with white, and inside the shadow trace is filled with black. This could be done without coloring inside or outside of the shadow, as matlab could easily distinguish the different regions. To keep coding and processing time down, this distinction was made using fill tools ahead of time in an image editor.

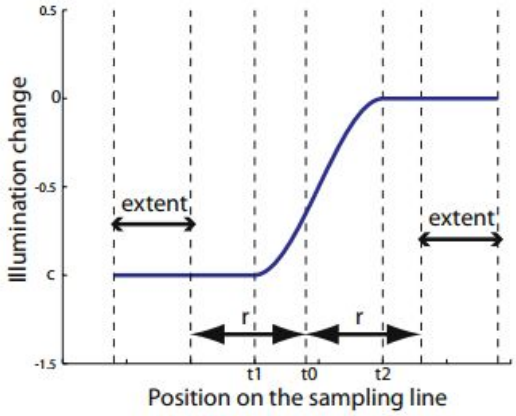
The next step is to load in the image and mask (shadow trace). After everything is loaded, the first step of processing is to calculate the location and direction of X samples and Y samples along the

traced area. These samples are 1 dimensional descriptors of the actual shadow edge model. They are to be used in the next step. These samples are stored in an array within matlab, and directly relate to the pixel locations of the mask, image, and gradients. The initial implementation limits sample size to be between a certain percentage (90% - 150%) of the brush size used for the shadow trace. This causes artifacting on certain angles of shadows (shown in the following image), and has been fixed.

After each shadow sample has been calculated, they are then matched with their immediate left and right neighbors for the next step.

Next, the models are fitted to a generic shadow model that takes into account the smoothness of shadow transition to it's neighbor model, and the best fit for every individual shadow model.





The image to the right shows what the generic smoothed shadow model will look like. Where the first derivative = 0, is the external unshadowed and internal shadowed area. Where the first derivative does not equal zero is the shadow edge. This model is used to cancel the effect on intensity of the shadow from the edge of the shadow.

To get the piecewise function, using the t1, t2, and calculated c value (calculated as the average illumination change from lit area to t1 vs t2 to the umbra) and the derivatives at t1 and t2 (being 0), I ran a simple set of 4 linear equations to solve for the coefficients of the cubic function generated. I then plug in t (if t is between t1 and t2) into the cubic generated by the linear system, and I am also able to return the derivative in the same fashion as the derivative of a cubic function is easy to find and plug into directly.

I set up a function that finds the value of E_{fit} ,taking parameters of the set of pixels in each sampling line, the t values on each sampling line, t1 and t2 relative to the set of t values, the normal distribution fitted to the gradient of the set of pixels on the sampling line (passed in from the minimization brute search function to reduce running time), and the gradient. I find doing a simple diff here works better than passing in the global image gradient, so I do that.

In the main loop, I get the loaded in image, in order to preserve color, I divided the image into its 3 channels, and ran a loop for each (for some reasons the timer I set up showed a much slower total time when I ran the 3 together, so that's why I separated them) to get the total time. This part is a bit time consuming; It could be because I have too many loops around because this is the first class I've ever used matlab and I'm not quite familiar with all the shortcuts or something, but it does take a while on a medium sized image. I promise your computer is not crashing here. I also initialize some empty arrays to store some values like $C_{li}(t)$ and its derivative. I find it's easier to just cache them and then do a simple matrix subtraction later.

Anyway I run the min_efit function, which is just a brute force search as described in the paper, and this is what takes the bulk of the time. Basically, this loops through values of t1 and t2 and finds one that's optimal. Then it returns the t1 and t2. Then at that sampling line, for each pixel value, go through the $C_{li}(t)$ function to get the values of $C_{li}(t)$ and $C'_{li}(t)$. I add the values of $C_{li}(t)$ to the averagec arrays (per color) and the derivatives to the derivative arrays. I will apply them in the next script, which is also applying gradient changes. I did not manage to do energy minimization, because I couldn't understand how to parameterize it into a sparse matrix enough to run pcg on. There is an unused function to calculate each E_{sm} value, though. Since I'm running on fairly limited time and using the raw optimized set of t1,t2s worked decently enough for the most part, I decided to do the texture reconstruction.

The next part is fairly straightforward. I used the Wavelet library (http://www.mathworks.com/matlabcentral/fileexchange/48066-wavelet-based-image-reconstruction-from-gradient-data) here, since it's one of the first ones that pops up on a google search and I don't trust myself to write gradient reconstruction of images by myself, anyway, and the details of which are not the focus of the paper. I get the gradients of everything in the shadow, then get the gradients of the lit area and the umbra around the brush (up to 3 px width, depending on if there's room), and fit it through normal distribution. I originally split it up into the 3 different rgb channels like I did with the illumination change model, but found that merging them have no adverse effect whatsoever and makes the code a lot faster, so that's what I did. I also originally applied the derivatives before reconstruction as the $G_{final}(t) = G_{orig}(t) - C'_{li}(t)$ Equation demands, but found ultimately leaving it out in the way I did things ended up giving a smoother transition. After I apply reconstruction, having transformed the gradients using the formulas specified in the paper, I subtracted the $C_{li}(t)$ values directly, per channel. This works fairly smoothly on the eagle picture shown below, though there's still a bit of artifacting with the bricks picture. The coloration is not entirely perfect, and I suspect if I got the Energy Minimization with pcg to work it'd work better, but it's fairly close.

The bricks are a little weird, the reconstructed colors are entirely off, though there's some differences in the bird picture as well it's a lot harder to notice. The penumbra detection is a bit more obvious here, though the colors compared to the reconstructed lit area isn't too far off.