Himanshu Bhatia  
Goran Svorcan  
Ningchuan Wan  

“Toon It”  
Real Time Cartoon Filtering  

**Introduction:**  
Our goal in this project was to implement a cartoon filter, both on single images on the PC and in real time on the phone’s viewfinder. We wanted a way to make colors look more general and drawn while at the same time defining and darkening edges to make people and objects look as if they were outlined and colored in. We also wanted to provide a way to capture video with our filter turned on as well as a way to capture still images with it. We based our work on the paper “Real-Time Video Abstraction” by Holger Winnemoller, Sven C. Olsen, and Bruce Gooch that can be found at [http://www.cs.northwestern.edu/~holger/Research/papers/videoabstraction.pdf](http://www.cs.northwestern.edu/~holger/Research/papers/videoabstraction.pdf).  

**Implementation:**  
We started off the project by trying to implement the algorithms described in the paper. It was Goran and Ningchuan’s job to get familiar with these and implement them on the PC, while Himanshu would be working on setting up a framework for the UI and the real-time filtering. Then we all got together and worked on getting the filtering to work on the phone as close to real time as we could make it, as well as getting the video and image capture components to work. The filter algorithm involved putting the image through a bilateral filter to smooth out the colors and accentuate the edges. Then this filtered image was put through a difference of Gaussian filter to just get the dark edges back. The bilateral filtered image was also quantized to limit the amount of colors in the image, giving it a more drawn look. Finally the quantized image was combined with the difference of Gaussian image to create the final cartoon-like image.  

Originally we used OpenCV’s built in bilateral filter function in the PC version of our program, but once we got familiar with it we decided to write the bilateral filter ourselves. On the phone, to filter the images we converted the images to textures as they were coming in, and then applied a shader to them in which we implemented our cartooning function. We found that implementing the bilateral filter was too taxing on the phone and would lead to results that were too choppy, so we decided to try something else. The lack of bilateral filtering made the difference of Gaussians work poorly as well so we settled on applying the sobel filter in both the x and y direction, and then combining the results to get our edges. Then we quantized the original image and added the new edges to get our final image. This considerably improved the processing time and produced fairly good images as well.  

**Results:**  
Our results are in the results PDF in this folder. However a sample of our video capture is available in this YouTube video:  
[http://www.youtube.com/watch?v=a3m50FJiSv8](http://www.youtube.com/watch?v=a3m50FJiSv8)
Review of Results:

We feel we were able to get fairly good results with our code. The PC code was difficult to implement at first, mostly because we were unfamiliar with the algorithms and because it required frequent type conversions between IplImages and Mats in order to use the built-in OpenCV filter functions. Once we got a handle on it however implementing our own version of the bilateral filter for example turned out to actually be simpler albeit involved more lines of code and ended up slowing down the program slightly. Our phone code however was a different story.

As stated previously, we had difficulty implementing the bilateral filter without slowing down the frame rate to unacceptable levels. Part of the slowdown came from us needing to convert floats to ints at points for array indexing. For loops as well slowed down the frame rate, which mean that we couldn’t have large kernel sizes, even kernels of size 11 for example were too large. A small kernel size however meant that the effects of our bilateral filter could hardly be seen and thus ended up not being worth the slowdown. This caused problems for our difference of Gaussian filter and thus we decided on using two sobel filters. Using quantization did not cause too much of a slowdown, however we were able to optimize it. Instead of looping through a certain amount of bins, if we just multiplied the color by 10, found the ceiling, and then divided by 10, we could get the nearest of 10 bins, and we ended up doing this for each color channel. This did not work as well as finding which actual color they were closest to as that would have required too many color declarations, but it produced a nice look. We eventually took an average of the ceiling and floor values to take away the washed out look we were getting in certain lighting conditions.

Finally, our image capture ended up working well, however our video capture, while implemented, was quite laggy. Since now we had to save each frame as soon as the shader was done being applied to it and pass it to the movie writer, we had the computationally taxing save operation on top of the already taxing shader, which led to frames being missed if the camera was moved too fast. We also ran into a problem with the black bar on the right appearing in our saved images. Since the image size is 640x480 while the camera viewfinder size is 800x480, a black bar on the side can be seen. So whenever we go to grab an image to capture from the frame buffer, the black bar ends up there. This is an unfortunate circumstance of the fact that we aren’t able to grab images directly from the camera as those would have no shaders applied to them. In addition, because the movie writer can only do 20 fps at its lowest setting and we cannot provide that many frames, we end up with very fast movies and about 20 seconds of filming results in about 1 second of footage.

Future Work:

If we had had more time, we could have tried other ways of applying our filters to the image, perhaps investigating how other mobile devices apply their filters.