6.087 Final Project Proposal

Nathan W Brei

The inspiration for my project came from a real-life problem I have faced– to convert photographs of technical drawings into a reasonably clean CAD file. The original plan provided for a full-featured set of tools to achieve this; however, the time frame only allowed for one part to be completed. However, this part was completed fairly thoroughly.

In its final form, to be handed in later today, my project does the following: It allows a user to convert graphic data back into a numeric form suitable for CAD. The user specifies an algorithm, and relevant parameters for that algorithm. Depending on command-line switches, the program either computes silently, shows the image as it undergoes filtering /binarizing /contourizing, or allows the user to play with the algorithm parameters in real time.

The 'output' of my program is a victim to the time crunch; DXF is not intuitive enough to learn in a couple of hours, and I chose to focus more on the algorithms and user interface instead. However, the data is nicely stored in a CvSeq of CvPoints just waiting for someone to do something with it.

Instructions for compiling my code can be found in the Makefile. I also included hyperlinks to the instructions for installing OpenCV. Although it was difficult to get OpenCV up and running at first, I strongly believe that there are no profound difficulties, and that the code should compile just fine from any computer with OpenCV already installed. I strongly encourage everyone to run my code— I worked hard on it, and I believe that it is fun to look at. Particularly fun is to visualize all of the different algorithms on `images/test0.jpg`:

```
./image2dxf -v -a 0 images/test0.jpg
./image2dxf -v -a 1 images/test0.jpg
./image2dxf -v -a 2 images/test0.jpg
./image2dxf -v -a 3 images/test0.jpg
```

Tests 1,2,3,5, and 10 are for the most part quite pretty. More intriguing is to open up an image in manual mode,

```
./image2dxf -m -a 0 images/test0.jpg
```

and see how the results change with varying threshold levels. I am sorry that the sliders aren't better documented (ran out of time, etc.) but the descriptions aren't particularly meaningful anyway unless you spend 48 hours reading books and articles on image processing. If you want a vague idea what the different 'vals' are about, type

```
./image2dxf
```

to see my POSIX-style help screen. Interestingly, the best algorithms for my program are probably the simplest, as algorithms 2 and 3 are really designed for determining the boundaries of solids. Playing with filters messes with the results, of course, but the pattern holds.

All source code is my own work. Some code skeletons came from the GNU C manual or the official OpenCV documentation; those were either trivial demonstrations, or 'best practice' pointers. The majority of the code here was written in the last 48 hours.

I broke up my code into the following modules. Main handles all of the command-line interface, and launches Brain into one of three states, which will persist for the duration of the program. These states are 'automatic', 'automatic visualized', and 'manual', referring to the amount of insight and control the user has over the process. 'Visualize' gives the user a glimpse of how the filtering/binarization is going, whereas 'manual' allows the user to make fine-tuned real-time adjustments to the algorithm's parameters. Brain controls the actual image filtering and edge detection, and launches the quasi-graphical windows. The algorithms for the filtering and edge detection are in kept in Toolbox, and the file output methods are kept in Scribe.

To facilitate communication between the different modules, I made use of global variables, possibly more than would seem prudent. While this made coding much easier, it also makes it far less safe. When this was taken into consideration, I concluded that the likelihood of witless modification causing catastrophic failure was extremely high regardless, and that the OpenCV does sufficient assertion checking to prevent problems from going undetected for an unreasonable amount of time. I consider the global variables to be a shared set of program parameters. Given the single-state nature of the program flow, I consider the utter lack of encapsulation to be acceptable, though not entirely desirable.

Just kidding, it's fine. You'll love it.