HandMade

The final project for @pybae, @MikkelKim, @tsherlock, and @iancai.
Our team name is HandMade!

Our members:

- Paul Bae
- Mikkel Kim
- Tim Sherlock
- Ian Cai

Pitch

There is a growing market in drawing tools, such as Paper and Sketchpad. These tools don't seek to replace the power of heavy applications, such as PhotoShop, but instead seek to provide a simple interface to allow users to quickly sketch up their ideas.

There have been several burgeoning ideas in this region as well. Rocketboard, which allows users to draw on a whiteboard to share thoughts in real time, FlockDraw, and much more.

Our app seeks to do the same, but in a unique way.

Rather than relying on mice or touch surfaces for input, our app will take input from the user by a camera. By using real-time hand tracking, we can track the motions of the hand to simulate drawing on a virtual whiteboard.

We can come up with a set of gestures to represent each action:

- An index finger pointed to the camera will represent the drawing state.
- A palm moving will cause the current whiteboard view to move.
- A closing fist represents zooming out
- A opening fist represents zooming in

And these gestures are hypothetical as of the moment, and will likely changed based on user input.

Our end goal is to create a desktop application that relies on the laptop's camera as input and provides a virtual whiteboard that users can write on, zoom in, zoom out, move around, and share (if time permits) to other users.

Objectives

Objective 1: Preprocessing

- Compute background
- Background subtraction
- Convert to YCrCb
- Threshold on each channel
- Morphology (erode and dilate) on each channel
- Merge channels
- Bitwise and operator with original frame
- Threshold final image
- Morphology (erode and dilate) final image
- Gaussian Blur
- Result: A binary image of the hand with most background noise

Objective 2: Detect hand motion

- Contour extraction
- Polygon approximation
- Max inscribed circle and radius
- Find region of interest
- Convex hull
- Convexity defects
- K-curvature
- Min enclosing circle
- Result: Contours, convexity defects, and circles of the hand from Objective 1

Objective 3: Gesture Recognition (TBA)

- Simple heuristics to detect static gesture
- For example:
    - 0-1 fingers detected, then close palm
    - 4-5 fingers detected, then open palm
    - 1 finger detected and no thumb, then pointing
- More research to be done here
- Result: Coordinates of where the gesture is and a flag corresponding to the gesture

Objective 4: OpenGL Whiteboard

- Design a modular system for gesture recognition
- Link up the x, y coordinates to drawing on an OpenGL canvas
- Implement an interface for multiple brushes, colors, and the like
- Result: A GUI showing an OpenGL whiteboard (built on Qt as a backend)

Objective 5: HTML5 Canvas (optional)

- Link up the OpenGL whiteboard in step 4 to HTML5 Canvas
- Make the updating real time
- Optimize the speed
- Launch on a server (most likely heroku)
- Result: Real time mirroring to a server of the image

Schedule

Our implementation will likely be in C++, OpenCV, and OpenGL. Our goal for the final project is to have a working demo of the whiteboard, with the above gestures implemented, and if time permits, present the whiteboard in real time online.

13 Nov:

- Objective 1

20 Nov:

- Objective 2

2 Dec:

- Objective 3
- Objective 4
- Objective 5 (maybe)

Proposal 1

We quickly realized that our initial goals were quite ambitious in terms of the scheduling of the project.

After doing some research into the field, we've come up with the above scheme for accomplishing the goal.

There are three main objectives that we aim to accomplish.

1. Preprocessing
2. Hand detection
3. Gesture recognition

Preprocessing includes the background subtraction, threshold and finding the hand in binary.
Hand detection represents actually finding the hand using a variety of techniques.
Gesture recognition represents associating a sequence of static frames to the gesture.
As for what we've accomplished so far, we did the entirety of the first objective. The current code is pushed up to the github.

The code itself works for a clean background (such as a whiteboard or a solid color wall), and noise detection will be a recurring focus throughout our project.

We mainly described the process in the objective above.

In terms of the code itself, we have a c++ make build with several classes and unit tests. There is definitely more to come in what we can accomplish.

The trello is the best way to see what we've accomplished.

The paper that we've been following closely is "Hand tracking and gesture recognition system for human computer interaction using low cost hardware".

Proposal 2

Following our schedule from last time, we managed to accomplish hand detection this week.

First using the preprocessed binary image as input, we found contours based on that image.

From that image we find contours using the OpenCV function:

findContours(.., …, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE)

Where the flags CV_RETR_EXTERNAL ensures that OpenCV returns the outer most contour, if finding multiple, and CV_CHAIN_APPROX_NONE returns all the points (without any optimizations, this might be changed later).

From there, we then find a polynomial approximation of the contours using the OpenCV function approxPolyDP.

This benefits us in two ways:

1. Filtering certain contours
    a. This works due to the fact that the approxPolyDP takes in a parameter that represents the minimum likeliness to a polynomial curve. Thus for noise contours, these are removed
2. Ensuring that a ConvexHull can be easily formed and computed
    a. Since we now have our hand as a polynomial curve, it is now both faster and more efficient to compute a ConvexHull in the next step

Once we've done the above, we now calculate a convexHull using the corresponding OpenCV function call.

Note that in our code, however, we call the function twice, one to store the actual points of the convexHull, and another to find the indices of the convexHull in the polyCurve we passed in.

Now we compute the convexity defects.
To do so, we run the function convexityDefects on each of the polyCurves and convex hulls we just found.

Finally, we have to find the two enclosing circles.

To first find the maximum inscribed circle we use the OpenCV function pointPolygonTest, which calculates the minimum length between a point and a contour (a vector of points).

Now we iterate over the entire image by pixels of 10, and find the minimum distance to the contours. This point is the center of the maximum inscribed circle.

We acknowledge the fact that this method is rather inefficient. For our next progress report we intend to find a faster way to compute this circle. We're currently looking into Voronovoi models and designing an algorithm based on that.

Once we've found this maximum inscribed circle, we create a bounding circle with it's radius time 3.5. We found this constant in a paper previously mentioned and it seems to be a logical heuristic for most scenes.

Once we've found the maximum inscribed circle, finding the minEnclosingCircle is rather trivial.

And finally, we find the curvature of a set of points in both directions starting from the valley. If the curvature is below a certain limit, we are able to deduce that this a fingertip.

We are also encountering difficulties in optimizing the above procedure. Again, this is one of our planned goals for the next project.

So to summarize, we've now found the positions of the fingers, the palm, the hand itself, and all the coordinates that represent that code. Using this information, we are now able to detect hand gestures.

All in all, our schedule has not changed significantly. We were able to accomplish the goals outlined last time.

There are however, difficulties arising in this process. The main being that of optimization. Right now, the code appears to run in real time on our machines, but with older processors, we aren't able to make that guarantee.

Videos

- Progress Report 1