Saccadic Scanpath Simulation on Static Images

Chi-Ning Huang & Xitu Chen

0. Abstract

In this project we implemented an interactive saccadic scanpath simulation with Processing, together with a saliency map generation program and an eyeball saccades animation simulation. The initial goal is to create saccadic scanpaths for a more realistic and an efficient model of eyeball animation, and we have achieved these, though with many possibilities for improvement.

Keywords: Saccadic Scanpath, Saliency, Animation

1. Introduction

1.1. Inspiration

Saccades is one of the three types of movements human eyes are capable of, the other two being vergence shift (change in focal length) and smooth pursuit. It describes the small, rapid and jumpy movements with very brief fixations in between. They are usually inaccurately or arbitrarily portrayed in animations, or ignored entirely, resulting in unconvincing eye animation. Our initial inspiration is to study and implement a system to simulate saccadic movements over static images, and eventually use our results to integrate animation.

1.2. Related Works

1.2.1. Saccadic Scanpath Generation

Eye ball movements are composed of fixations and saccades. A sequence of fixation is called a visual scanpath. We use a widely used scanpath generation method referenced in L. Duan et al. [3] that takes in a static saliency map and generate scanpath by using winner-take-all (WTA) algorithm and inhibition-of-return (IoR) scheme. However we do not take into account of the effect of human knowledge on the fixation point selection as other paper suggested. It will required a certain degree of computer intelligent and a tested database which we think will be out of scope to implement and we couldn't find any existing source online neither. What we have implemented is closer to Levy flights path generation which is a heavy-tailed distribution random walk generation.

1.2.2. Saliency Generation

We came across multiple different approaches in generating saliency maps - a grid of values that represents the importance of each pixel in the image using a certain criteria. [4] proposes an approach that produces saliency maps with sharp and distinct edges. The method first abstracts the input image into like-sized patches, each patch being the average color of the bounded pixels; the patches/elements are then compared globally and locally for their uniqueness and spatial distribution. This approach works very well with well defined elements in natural images that are not too crowded, but dealing with photos with no dominant subject is a limitation.

[5] proposes a more psychologically-based approach, using matrix orthogonalization to detect different features such as line orientation, color, density, etc., and eventually composing all maps generated into one. We like their detailed approach and convincing results, but with little implementation details, their method is beyond the scope of our project.

We decide to use a method proposed by an older paper [6] (we will refer to it as GBVS from now on), which is also used in [1]. It is a three-step algorithm, using a graph as a Markov chain to distribute and concentrate saliency values in more important places. It will be described in more details in Section 2. The approach is organic and It draws an analogy between the Markovian graph and neuronetwork, and the fuzzy results seem convincing, taking nature of saccades into consideration.

1.2.3. Saccades Animation

There has not been too much study on animation. [7] used an anatomically accurate approach to simulate the muscles and rotation of eyeball. We did not need such accuracy in our result, but it was an interesting read. We now know that saccadic movements have no “winding-up” motions and no overshoots.

2. Saliency Map Generation

As mentioned, the GBVS method is a three-step process: extraction, activation and normalization. The first step involves assigning an initial saliency value to each pixel through feature extraction, followed by building a weighted graph based on them, and distributing mass throughout the system along the weighted edges.

2.1. Feature Extraction

Based on implementation details described in [6], we use Gabor filter for the first step. Gabor filter is orientation-based, and gives higher values to pixels that stand out within their neighborhood (kernel size) and aligned with θ.

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\lambda} + \psi\right)$$

eq. (1)

λ: wavelength

θ: orientation

ψ: phase offset

σ: standard deviation of Gaussian envelope

γ: aspect ratio

x': xcosθ + ysinθ

y': -xsinθ + ycosθ

A single image produced by the filter usually produces a hatching effect, and the resulting feature extraction map is produced by clamping values and overlaying multiple filtered images. We iterate through {0°, 45°, 90°, 135°} for orientation, and {0, 90} for the phase offset. It is tricky trying to choose a suitable value for wavelength as it depends on the resolution of the picture. After many trials we have found that for images under 150x150px, lambda = [3, 8], for images going up to 300x300px, lambda = [8, 13], and for up to 600x600px, lambda = [13, 20].

The composite image is then applied a Gaussian blur with radius of 3px to smooth out the result, and downsampled for further processing. Here is a pseudocode for the sampling algorithm I use:

```
input: n x m: image dimensions, w: sampling window size
for i from λ to m-λ in increments of w:
      for j from λ to n-λ in increments of w:
            sum = sum(all pixels within window)
            if the next window size < w/2:
                  add the remaining pixels
            result[i/w][j/w] = average(sum)
```

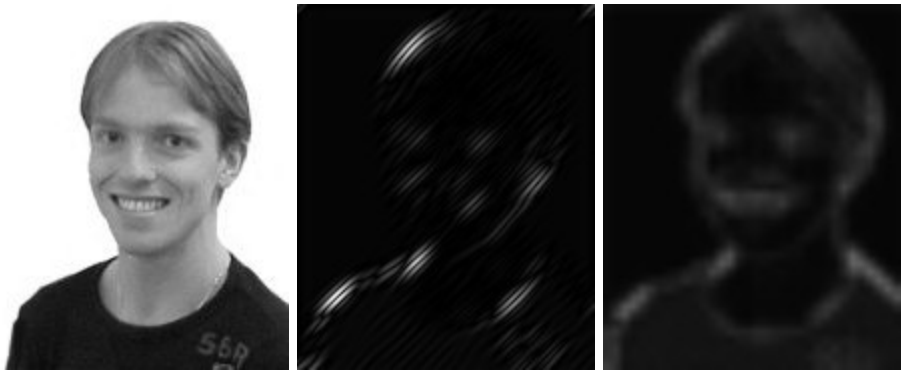

Fig. 1. a) original picture from [8]; b) filtered result with λ=5, θ=135°; c) result of composition and downsampling. Image source: [8]

2.2. Building the Graph

The next part is to generate a weighted graph with the extraction values. After downsampling, We implemented a graph and populated it with nodes holding values from pixels in the image from the previous step. Each node is connected to all other nodes within its neighborhood (5x5 square), and the weight of the edge is calculated with equation (2):

$$w_1((i,j),(p,q)) \triangleq d((i,j)||(p,q)) \cdot F(i-p, j-q), \text{ where}$$
$$F(a,b) \triangleq \exp\left(-\frac{a^2+b^2}{2\sigma^2}\right).$$

eq. (2)

d: difference of map values from previous step between [i,j] and [p,q]

σ: it is set to seventh of the map width, based on [6]

The weight is determined by how different the nodes are and the distance between them. Each node stores its neighbors and edge weights in a hashmap, and the graph keeps a grid of nodes, so weights are not calculated repeatedly. After the construction, each node distributes mass of value 24 proportionally according to edge weights to its neighbors. The mass value could be anything, it is chosen because each node that is not at the edge of the image has 24 neighbors.

The weight distribution process is iterated several times, with edge weight recalculations, and we find that 4 iterations is usually sufficient. The process concentrates saliency values at places with higher initial values within their neighborhood, and eliminates a certain amount of noise.

Fig. 2. a) before distribution; b) after distribution

3. Scanpath Generation

3.1 Overview

The scanpath generation method we use is based on the method described in [3]. The scanpath is guided by three factors: saliency map, visual memory and the winner-takes-all algorithm. The method starts by taking in the saliency map generated by algorithm described in the previous section, and finding a fixation point for the current time frame. The current fixation point will be stored in memory that represent the physiological visual memory which help to eliminate the effects of the current fixation point on selection of the next fixation point. After the generated duration time for the current fixation point had passed, the program will do another calculation to find the next fixation point from the saliency map. The following subsection in this section will be explain the detail of this algorithm.

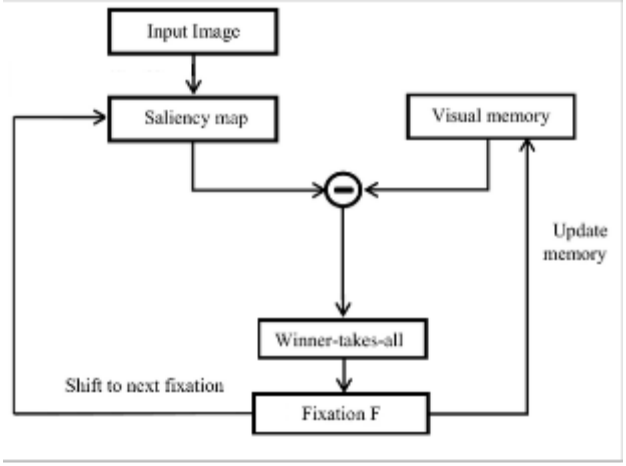

Fig. 3. The proposed framework

3.2 Fixation point selection

Our method takes visual memory into account, but the point selection is not completely based on cognitive psychology. We adopt a semi-deterministic method to do our random selection. After sorting the fixation points in descending order of saliency values, we determine randomly between two selection scheme. The first scheme is selecting the fixation point from one of the top five fixation points over all points. The five points are weight with their saliency value and higher value will have higher chance to be selected. The second scheme we use is randomly choosing points among the top 50% of points. We believe this will give a more controlled randomness for fixation point selection.

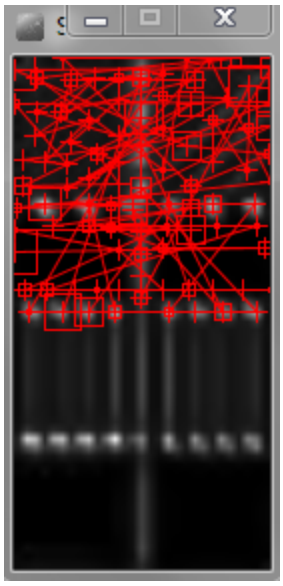

Fig. 4. Blooper picture of a previous bug in point selection. Image source: [8]

3.3 Visual memory representation

The selected fixation point is stored inside memory to represent the effect visual memory has on fixation selection. We will eliminate those points from the selection group once they have been chosen before. We also create a bounding box around the selected point to eliminate points that are few pixels away from the selected point. This ensure the divergence of our selection instead of focusing on the few high saliency points that are very close to each other.

Moreover we only keep track of 7 points that are currently displayed on screen. We take this approach from [1] as a representation of oblivion in human short term memory. This gives the points that been visited another chance to be visited and also ensure we do not run out of points to select from.

3.4 Duration time generation

Once again, our algorithm is not entirely psychologically-based, hence the duration of eye on a fixation point is not determined by the complexity or the information richness of the object the eye is looking at. We determine the fixation time by the distance between the current point and the last chosen point and their saliency value differences. We found this method is very effective and produce satisfying results that are very similar to the real duration time of saccadic eye movement.

4. Eyeball Animation

We implemented a program that takes text input generated by the interactive saccadic scanpath program and animates a pair of eyeballs according to the fixation times and locations. The speed of rotation can be easily tweaked; peak speed of human eyes during saccadic movements is 900°/s.

Fig. 5. Eyeball animation

5. Results

5.1. Saliency Maps (Xitu)

During the first step, feature extraction, we compare our Gabor filtered image to images generated by [8] to make sure our implementation is correct.

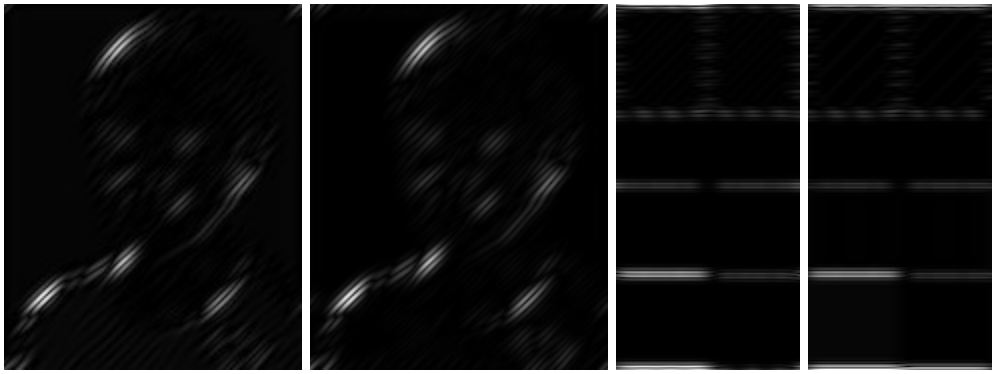

Fig. 6. a) our implementation, λ=5, θ=135°; b) [8]'s result, same attributes; c) our implementation, λ=5, θ=90°; d) [8]'s result, same attributes. Image source: [8]

Below are composited saliency maps. The algorithm tends to pick up details with high contrast within the neighborhood, but it sometimes overwhelms the macro structure, as seen in Figure 7 (e). There is also a fringe of whiteness with thickness of λ in the Gabor filtered images, and it is cropped out during down sampling. This problem is present in [8]'s implementation as well.

Unfortunately we were unsuccessful when trying to incorporate color with Gabor filtering, as the overlayering of results from the 4 channels (RGB + grayscale) often causes very low contrast in the final image and grabs too much details. It is something we are trying to tackle. One approach could be applying some noise reduction algorithm before layering them.

Runtime is a concern, as it grows proportionally with k^2, k being the kernel size, which is directly affected by λ the wavelength. The graph creation and distribution has been fast after downsampling. Figure 7 a) (128x256) and c) (149x183) takes around 5 seconds, while d) (600x600) and e) (640x480) takes approximately 5 minutes each. Most time are spent on Gabor filtering images. Fortunately these are precomputed input for the saccadic scanpath program, not real time.

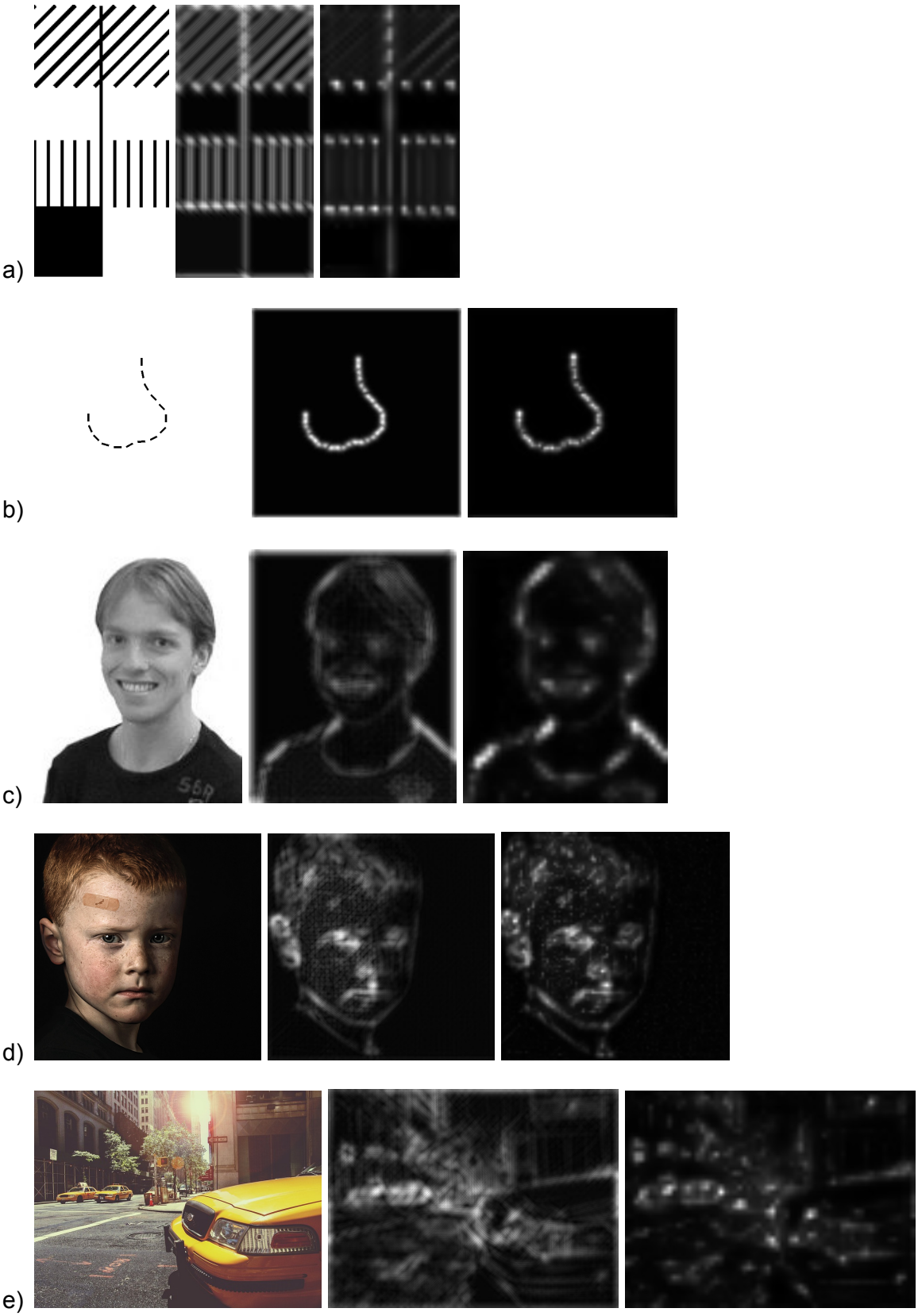

Fig. 7. Saliency images. Left is original, middle is result of composited Gabor filtered images, and right is the final distributed result. Image sources: a) to c): [8], d): [9], e): [10]

5.2. Saccadic Scanpaths (Chi-Ning)

Figure 8 shows three composited images (Xitu) we created by overlaying results of 3 seconds of saccadic movement data. Each image is created by 10 sets of data to show trends of concentration within the 3-second time frame. As a rough testing, it fits the general trend intuitively. Notice in Figure 8(a), the edge of the image still gets some saliency value due to an unexpected thickness of white fringe.

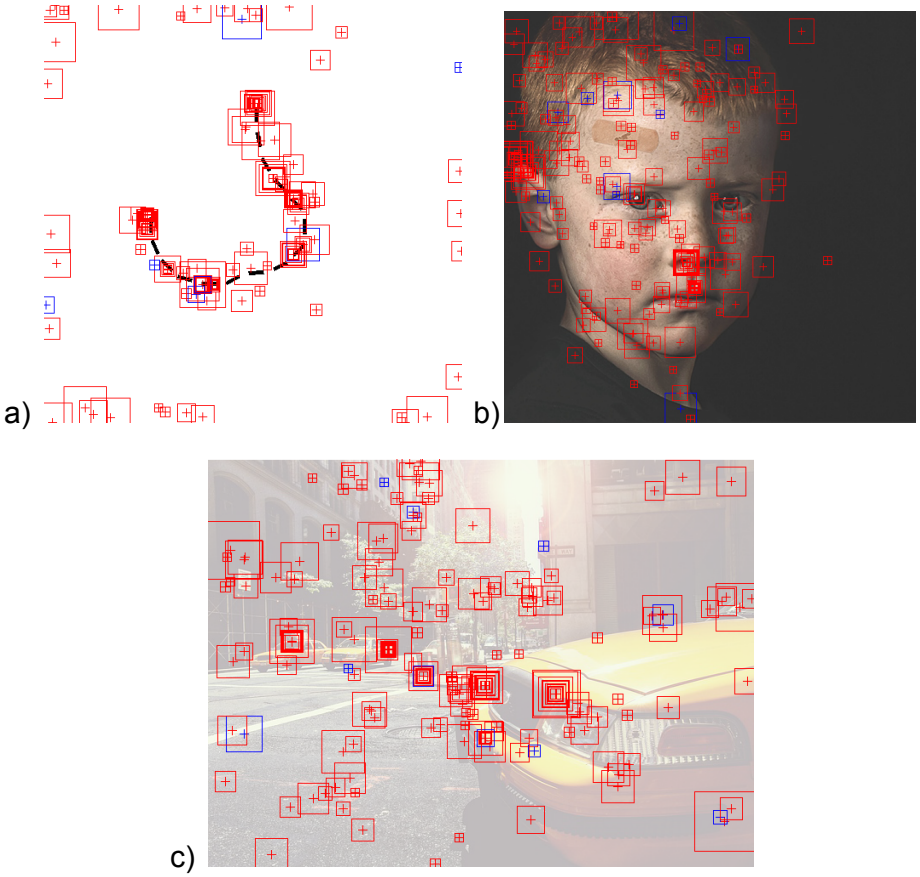

Fig. 8. Composite images from 10 data sets of 3 seconds for each image. Image source: a): [8]; b): [9]; c): [10].

We then compared our results with online database CAT2000 from Massachusetts Institute of Technology (MIT) with real human fixation datasets. As we can see in Figure 9, Figure 10 and Figure 11, the distribution of fixation points generated by our program matches the fixation point distribution graph of the CAT2000 dataset (the right image, where the bright part represent high concentration of fixation point and vice versa). However since our saliency map generation does not take object recognition into consideration, and people tend to gravitate their views towards familiar objects (e.g. image with warning sign, iconic sculpture, people's face...etc), there is a significant portion of mismatches. As shown in Figure 12, our program tends to focus on parts with higher contrast rather than where people normally look, such as eyes in this case. Nevertheless our program is sufficient

enough, and we think we have successfully achieve the goal we set for this paper: a program that creates saccadic scanpaths for a more realistic and efficient model of eyeball animation.

5.3. Animation (Xitu)

We have achieved quite convincing results according to feedback from our presentation in class, and our Github repository holds gifs to our animations under /misc_pics (link at the end of Section 6). eyeball_fast_233.gif was produced by feeding the program 3 seconds of saccadic movement input for image from Figure 7(d), rotation speed set to 900°/s; eyeball_slow_233.gif is generated from the same input, rotation speed set to 180°/s. The previous is more convincing, the latter has smoother animation. The face of the mouse (or bunny) is an image that was added on later for better visual effect.

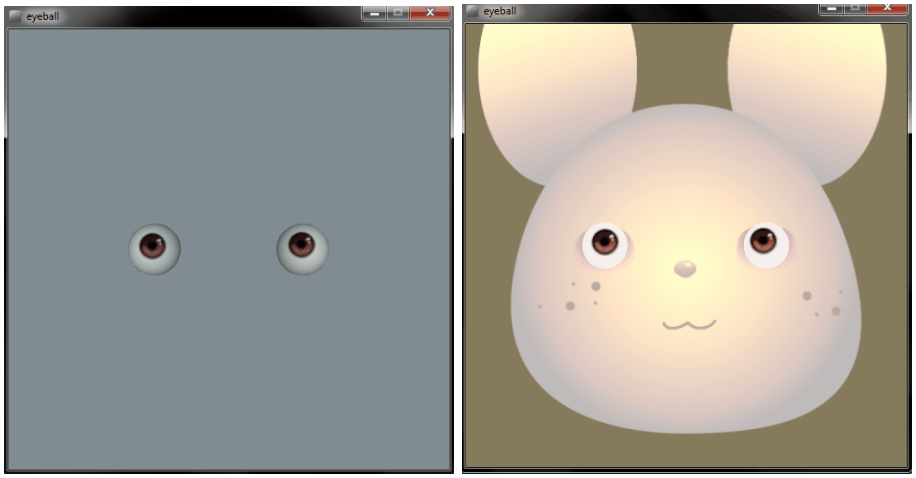

Fig. 13. a) before adding face; b) after adding face.

6. Conclusion and Future Possibilities

The work of this project is divided into saliency map generation (Xitu) and scanpath generation (Chi-Ning) and we both got relatively satisfying results. The saliency maps successfully describe the important locations in some images and the scanpath is correctly generated from the saliency map. Major limitation for saliency generation is that the algorithm focuses too much on local contrast instead of global importance, and can ignore some features. It also only considers luminance values instead of including colors. Though through comparing with the CAT 2000 dataset, our resulting scanpaths are not far off. We could try out different saliency approaches next time, such as [4] with sharper images, or a more systematic and psychologically-based method described in [5] given more time.

The limitation for scanpath is that it does not take the psychology aspect into account in fixation point selection. We think we can improve this by building a shape recognition system suggested in [1] that identify shapes specify by users. This will allow us to make certain object more stand out than other (e.g. eyes) to give us a closer result to real human saccadic scanpath data. This is one of the future goal for this program.

In terms of the animation (Xitu), due to time constraints we were not able to integrate it with our main scanpath generation program as much, since it requires us to shift Chi-Ning's work entirely to a 3D scene, which proved to be quite a pain in Processing. However further integration is definitely something we would like to do. A Maya script plugin that takes a pair of joints rigged to eye geometries, and an image texture, and outputs animation sequence is one way to extend it. It would be very helpful towards realistic eye animation.

Link to Github repository: https://github.com/huangc8/ACGFinal

7. References

1. O. Le Meur, Z. Liu, 2015. “Saccadic model of eye movements for free-viewing condition”, Vision Research
2. W. Wang, C. Chen, Y. Wang, T. Jiang, F. Fang, Y. Yao, 2011, “Simulating Human Saccadic Scanpaths on Natural Images”, IEEE
3. L. Duan, H. Qiao, C. Wu, Z. Yeng, W. Ma, 2013. “Modeling of Human Saccadic Scanpaths Based on Visual Saliency”, ICGEC 2013
4. F. Perazzi, P. Kraehenbuehl, Y. Pritch, A. Hornung, Disney Research Zurich, Stanford University, 2012, “Saliency Filters: Contrast Based Filtering for Salient Region Detection”, IEEE
5. X. Sun, H. Yao, R. Ji, X. Liu, 2014, “Towards Statistical Modeling of Saccadic Eye Movement and Visual Saliency”, IEEE
6. Harel., Koch., Perona. 2007. “Graphic-Based Visual Saliency”, NIPS*2007
7. Papapavlou C., Moustakas K., “Physics-based modelling and animation of saccadic eye movement”, 2014
8. Free Online Gabor filter open source Matlab implementation by N. Petkov and M.B. Wieling, University of Groningen, matlabserver.cs.rug.nl/edgedetectionweb/web/
9. “Bo”, http://1x.com/photo/22666/category/portrait/popular-ever/bo
10. “Taxi”, RyanMcGuire, http://pixabay.com/en/taxi-cab-taxicab-taxi-cab-new-york-238478/

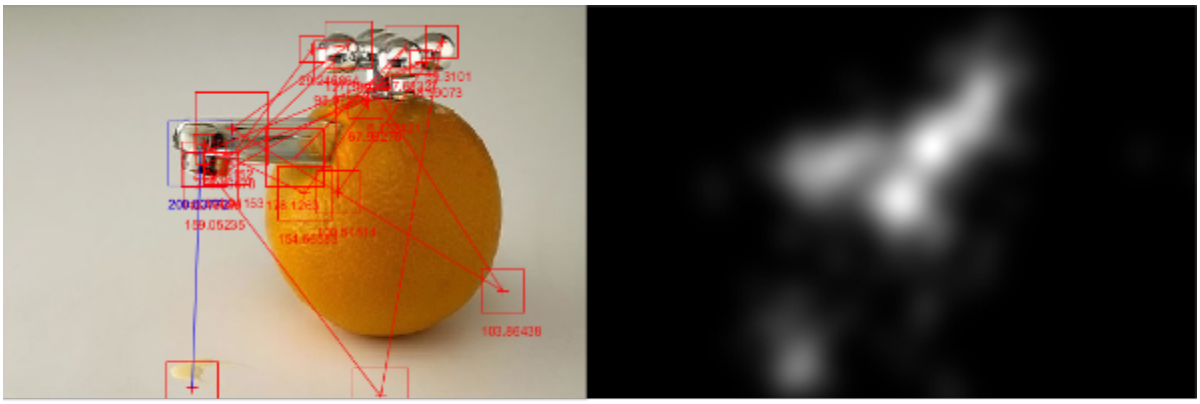

Fig. 9. Fixation point comparison with CAT 2000

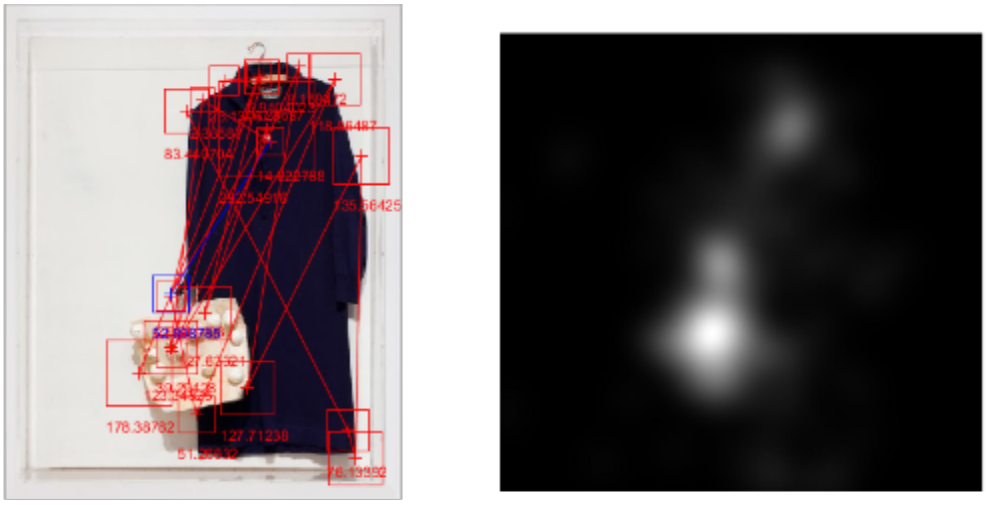

Fig. 10. Fixation point comparison with CAT 2000

Fig. 11. Fixation point comparison with CAT 2000

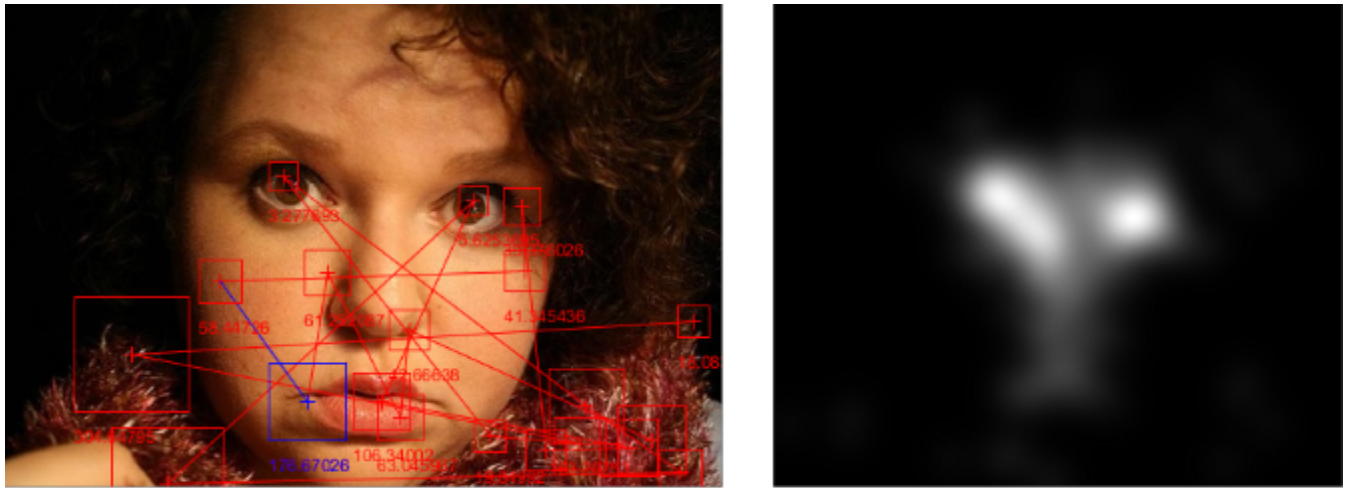

Fig. 12. Fixation point comparison with CAT 2000