Noise and User Attention

HyoJeong Choi

December 19, 2014

Abstract

This project explores passive user interaction with video, and how the noise created by the interaction affects user attention. Here passive interaction refers to unintentional input from the user such as ambient sound or eye movement. The user is not making an active decision to influence what he/she sees but rather responding to what he/she is exposed to. Based on the input data the video is manipulated to disrupt the video to interrupt the user attention.

1 Project Description

Video projection has been used as theatrical tool since the early 1920s by Erwin Piscator, a close collaborator of Bertolt Brecht. In Brecht's theory of Epic theater, he states that projection should be used to comment or oppose the event on stage to create the alienation effect.

Projection technology has come a long way since then, and the emergence of film and internet have completely changed the way video is conceived in the entertainment industry. The use of projection on stage is not by itself interesting, if not trite. Perhaps with rapid development in the field of human computer interaction and its design, it would only be natural to follow the trend in designing performing arts.

The problem becomes more sophisticated however when the role of the "user" is not so simply defined as in the case of the audience in theaters. It connects directly with the question of why people go to the theater and what is the fine line for audience participation. The concept of audience participation in performing arts is in itself a debatable topic, in relation to the fact that many audience come to the theater to enjoy the evening. The difference in perception between an "observer" and "participant" is radically different, and this dynamic must be treated with care.

Figure 1: Before and after Gaussian blur is applied.

Thus this project starts from the question of how to disrupt the system(video projection) by audience participation in a way that will not break the audience's perception as an observer of the event. The specific parameter of interest in this project was "attention." And to measure the degree of attention, sound input is used only because it is the most convenient and direct way of describing the environment. The level of disruption depends on the level of noise. Later on with the help of an eye tracking device, more direct manipulation was possible by tracking the eyes. However this is yet fully explored. Ultimately, the goal is to understand how this disruption would affect the user experience.

1.1 Gaussian Blur

As a means of disruption, I have been experimenting with two different methods: Gaussian blur and Coloring. Gaussian blur script calculates a kernel size with radius 9 and sigma depended on the maximum volume every 20 millisecond (see Sec. 3, line 164, and Fig. 1). The radius is set at 9 for practical purposes, and with a simple image, as oppose to a real time recording, larger size kernels were handled. As of now, the code makes the playback video blur more as the noise level increases. I have been experimenting to make where the user is looking at either more blurry or less blurry, but as I went on I have not focused on making the viewing clearer

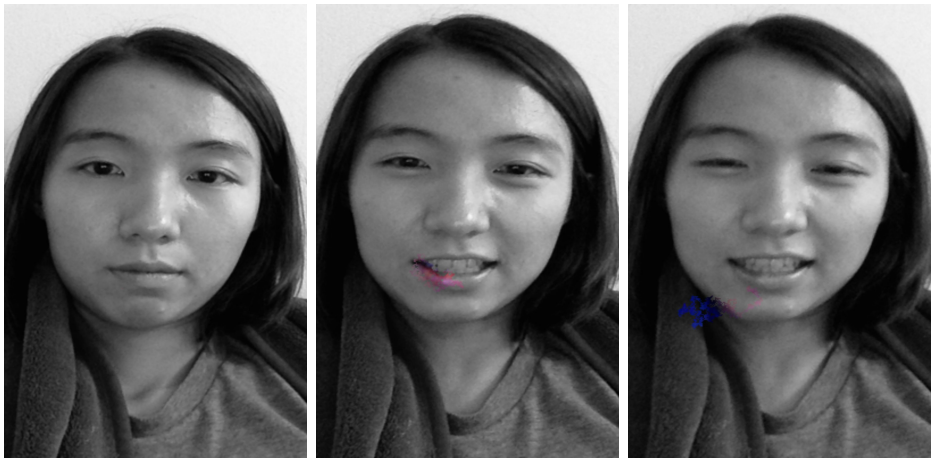

Figure 2: No input sound(left) versus input sound looking at different points(middle and right).

since our vision already functions in a way in which we blur our peripheral vision.

The main problem with the blur effect was that since the <canvas> paints the blurred image every 20 milliseconds, JavaScript does not seem to be fast enough to blur of video and then manipulating certain points of the pixel. Therefore for further development, the options would be either to make the blur more efficient by using a non-kernel based script and employing a two dimensional Fourier Transform, or to not blur the whole video but to create a blurred ¡canvas¿ only at the position of the user's attention.

1.2 Gray Scale and Color

The second approach was to start from a gray scaled video and then to color the viewer's attention (see Sec. 3, line 89, and Fig. 1.2). This was more of a recent development, from the desire to play more with the colors. Before sound input was creating random video noises, the number proportional to the level of noise. However the "randomness" became too redundant very quickly, so I started to play with just one noise each <canvas>.

The original intention was to be able to color a certain segment of the video possibly by edge detection. However this was never explored due to its feasibility. So right now the size of the noise is again proportional to the noise level, and the shape is determined by a random walk process (the next pixel to color is chosen randomly from the 8 neighboring pixels, and

it continues until it reaches the number of total pixels). The original color of the pixel is retrieved by skipping the gray scale effect and making the transparency of that pixel to 0.

2 Discussion

There are some issues that need to be resolved in order to move forward. One of them being the scaling of the sound input. Currently, the range of decibels for the sound api analyzer is from -90dB to -10dB. And I have yet made an effective scale conversion where it is visible to differentiate from low level noise to high level noise. So for both blur and color effects, after a certain level has passed, the effect seems to be visibly the same. And become more of an issue as when the video source is a real-time web camera because the resolution is not high enough to create a wide range effect.

Then again, the question may be ignored if the direction for development is toward a more efficient and simple effect. If the blurring is only to take place where the eye is at, the only way (that I can think of) to blur the image more than once, i.e. to increase the intensity of the blurring, is to overlay a more blurred image. And this must be done in a way that is not timed with the refreshing of the video or else there is no time to do it anyway.

3 Source Code

```
var WIDTH;
var HEIGHT;
var video;

var canvas;
var context;
var analyser;

document.addEventListener('DOMContentLoaded',function(){

  navigator.getUserMedia = (navigator.getUserMedia ||
                            navigator.webkitGetUserMedia ||
                            navigator.mozGetUserMedia ||
                            navigator.msGetUserMedia);

  var audioCtx = new (window.AudioContext || window.
      webkitAudioContext)();
  var sourse;

  analyser = audioCtx.createAnalyser();
  analyser.minDecibels = -90;
  analyser.maxDecibels = -10;
  analyser.smoothingTimeConstant = 0.85;

  video = document.getElementById('video');
  canvas = document.getElementById('canvas');
  context = canvas.getContext('2d');

  if(navigator.getUserMedia){
    console.log('getUserMedia supported.');
    navigator.getUserMedia(
      {
        audio: true,
        video:true,
      },
      function(stream){
        video.src = window.URL.createObjectURL(stream);
        video.controls = true;
        video.muted = true;
        localMediaStream = stream;

        source = audioCtx.createMediaStreamSource(stream);
        source.connect(analyser);
        console.log("audio connected");
        Filter();
        grayScale();
```

```
46        },
47        function(err){
48              console.log('The following gUM error occured: ' +
                    err);
49          }
50        );
51
52
53    }else {
54       console.log('getUserMedia not supported on your browser!')
              ;
55    }
56 },false);
57
58 function updateVideoDim() {
59    WIDTH = video.videoWidth;
60    HEIGHT = video.videoHeight;
61
62    canvas.width = WIDTH;
63    canvas.height= HEIGHT;
64 }
65
66 function getMaxOfArray(numArray) {
67     return Math.max.apply(null, numArray);
68 }
69 function angle(n){
70    var dir =0;
71    if (n==1){dir= -((WIDTH+1)*4);}
72    else if (n==2){dir = -(WIDTH*4);}
73    else if (n==3){dir = -((WIDTH-1)*4);}
74    else if (n==4){dir = 4;}
75    else if (n==5){dir = -4;}
76    else if (n==6){dir = (WIDTH-1)*4;}
77    else if (n==7) {dir = WIDTH*4;}
78    else if (n==8){dir = (WIDTH+1)*4}
79    else return false;
80    return dir;
81 }
82 function match(array, num){
83    var bool = false;
84    for(var i=0; i<array.length;++i){
85      if(array[i]==num) return true;
86    }
87    return bool;
88 }
89 function grayScale(){
90
91    if (!video.videoWidth) {
92      setTimeout(grayScale,20);
```

```
    return;
  }

  if (video.videoWidth!=WIDTH) {
    updateVideoDim();
  }

  var bc = canvas.getContext('2d');
  bc.drawImage(video,0,0,WIDTH,HEIGHT);
  var imageData=bc.getImageData(0,0,WIDTH,HEIGHT);
  var data = imageData.data;
  var imglen=WIDTH*HEIGHT;

  analyser.fftSize = 2048;
    var bufferLength = analyser.frequencyBinCount;
    freq = new Uint8Array(bufferLength);
    analyser.getByteFrequencyData(freq);

     //set variables
  var max = getMaxOfArray(freq);
  var radius =9;
  var sigma = Math.pow(6,max/200)-0.99;

  if(sigma ==0) sigma =0.01;

  var kernel = Gaussian(radius,sigma);
  var rad = kernel.length;
  var px = new Array(rad);

  var brightness;
  for(var j=0; j<HEIGHT; ++j) {
    for(var i=0; i<WIDTH; ++i) {

      var k=(j*WIDTH+i)*4;

          var r = data[k];
          var g = data[k+1];
          var b = data[k+2];
          brightness = (3*r+4*g+b)>>>3;

          if (lasteyedata) {
          var eyei=Math.floor(lasteyedata.scoords.x*WIDTH);
        var eyej=Math.floor(lasteyedata.scoords.y*HEIGHT);
            var eye = [];

            if(k==4*(eyei+WIDTH*eyej)){
              var d= 0;
              //var ln = Math.random()*500;Math.floor(Math.pow
                  (1.5,max)
```

```
              for(var l=0; l<10*max; ++l){
            //if(px) data[k] = dotProduct(px,kernel);
            //data[k+d] = r;
            //data[k+d+1] = g;
            //data[k+d+2] = b;
            data[k+d+3] =0;
            if(data[k+d]==null){d = 0;}
            eye.push(k);
            k = k+d;
            d = angle(Math.ceil(Math.random()*8));
          }
        }else if(match(eye,k)==false){
              data[k] = brightness;
              data[k+1] = brightness;
              data[k+2] = brightness;
        }
      }
    }
  }
    imageData.data = data;
    context.putImageData(imageData,0,0);
  setTimeout(grayScale,60);
}
function Filter(){

  if (!video.videoWidth) {
    setTimeout(Filter,60);
    return;
  }

  if (video.videoWidth!=WIDTH) {
    updateVideoDim();
  }


  var bc = canvas.getContext('2d');
  bc.drawImage(video,0,0,WIDTH,HEIGHT);
  var imageData=bc.getImageData(0,0,WIDTH,HEIGHT);
  var data = imageData.data;
  var imglen=WIDTH*HEIGHT;

  //get audio data
  analyser.fftSize = 2048;
    var bufferLength = analyser.frequencyBinCount;
    freq = new Uint8Array(bufferLength);
    analyser.getByteFrequencyData(freq);

    //set parameters for Gaussian kernel
  var max = getMaxOfArray(freq);
```

```
  var radius =11;
  var sigma = Math.pow(6,max/200)-0.99;

  if(sigma ==0) sigma =0.01;

  var kernel = Gaussian(radius,sigma);
  var rad = kernel.length;
  var px = new Array(rad);

  //blur...
  for(var i=0; i<rad;i++){
    px[i]=new Array(rad);
  }

  var size = data.length-4*WIDTH*radius;
  var noisestrength=max/200.0;

  for(var hc=0; hc<size; hc++){
    if(hc%4==3){continue;}
    //if(hc>size - WIDTH*4*radius)
    for(var i=0; i<rad;i++) {
      for(var j=0; j<rad;j++){
        //if(!px[i][j]){px[i][j]=data}
        px[i][j] = data[hc+i*4+WIDTH*j*4];

      }
    }
    data[hc]=dotProduct(px,kernel);
  }
  //draw on canvas
  imageData.data = data;
  context.putImageData(imageData,0,0);

  working=false;
  setTimeout(Filter,80);
}

function dotProduct(a,b){
  var sum=0;
  if (a.length==b.length) {
    for(var i=0;i<a.length;i++){
      for(var j=0; j<b.length; j++){
        sum += a[i][j]*b[i][j];
      }
    }
  return sum;
  };
}
function Gaussian(r, sigma){
```

```
239   var kernel = new Array(r);
240   var uc;
241   var vc;
242   var sum=0.0;
243   var g;
244
245   for(var i =0; i<kernel.length; i++){
246     kernel[i] = new Array(r);
247     for(var j=0; j<kernel.length; j++){
248       uc = i -(kernel.length-1)/2;
249       uv = j - (kernel.length-1)/2;
250       g=Math.exp(-(uc*uc+uv*uv)/(2*sigma*sigma));
251       sum += g;
252       kernel[i][j]=g;
253     }
254   }
255
256   for(var i=0; i<kernel.length;i++){
257     for(var j=0; j<kernel[0].length; j++){
258       kernel[i][j] /= sum;
259     }
260   }
261   return kernel;
262 }
263 //...
264 //
      ..............................................................
     socket.io
265 var socket=io();
266 //
      ..............................................................
     Eye Tracking
267 var paused=false;
268
269 var eyedatahistory=[];
270 var lasteyedata=null;
271
272 socket.on("eyeData", function(data){
273     if (paused) return;
274
275     var mx=(data.GazeL[0]+data.GazeR[0])*0.5;
276     var my=(data.GazeL[1]+data.GazeR[1])*0.5;
277
278     //main = document.getElementById("main");
279     data.scoords={x:mx, y:my};
280     data.ncoords=getElementNCoordsFromGaze(video, mx, my);
281     data.vcoords={x:data.ncoords.x*WIDTH, y:data.ncoords.y*
          HEIGHT};
282     data.time=new Date();
```

```
283     lasteyedata=data;
284     eyedatahistory.push(data);
285     if (eyedatahistory.length>2000) {
286       eyedatahistory.splice(0,1);
287     }
288 });
289 
290 function getElementNCoordsFromGaze(trackElement, nx, ny) {
291     //screen size in pixels [maybe points in MAC]
292     var screenW=window.screen.width;
293     var screenH=window.screen.height;
294 
295     //window in screen pixels
296     var wx=window.screenLeft;
297     var wy=window.screenTop;
298 
299     var ww=window.outerWidth;
300     var wh=window.outerHeight;
301 
302     //client area in screen pixels
303     var cw=window.innerWidth;
304     var ch=window.innerHeight;
305 
306     var cx=wx+(ww-cw)/2; //assuming no side bars
307     var cy=wy+(wh-ch); //assuming no bottom status bar or
        debugging console open
308 
309     //canvas properties
310     var trackRect = trackElement.getBoundingClientRect();
311     var elx=trackRect.left+cx;
312     var ely=trackRect.top+cy;
313     var elw=trackRect.width;
314     var elh=trackRect.height;
315 
316     var xx=nx*screenW;
317     var yy=ny*screenH;
318 
319     var ex=(xx-elx)/elw;
320     var ey=(yy-ely)/elh;
321 
322     return {x:ex, y:ey};
323 }
```