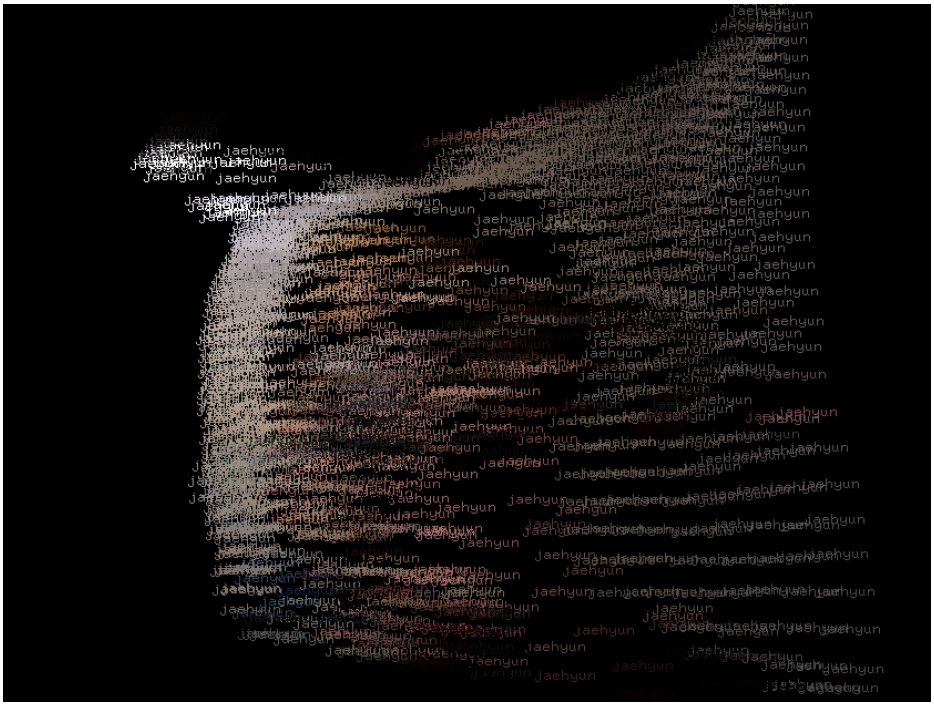
PLAYING WITH ofPIXEL

Mirong Kim and Jaehyun Kim

How to change pixels to typos - by using ofPixel

Before generating 3d effect from 2d video images, let's play with typo and ofPixel to warm up! This tutorial will help you same image here with changing the pixels to text. You can change the text whatever you want!



1. Creating an image by using ofPixel (using ofVideoGrabber class to capture video images). Also, before getting the pixels and text, we should make variable here. The text will change every time you type, so we should make Boolean to append the characters every time key pressed. Then we should make the first typed line when we start the program, so you should also make string variable.

1) in testApp.h

```
#pragma once

#include "ofMain.h"

class testApp : public ofBaseApp{
```

```
    public:
      void setup();
      void update();
      void draw();

      void keyPressed(int key);


    ofPixels pixels;
    ofTexture texture;

    ofEasyCam camera;

    ofVideoGrabber video;

    bool bFirst;
    string typeStr;


};
```

2) in testApp.cpp,

To get the pixels and update them every time when capturing the real-time video image, we should resize the pixels and place the video.update () and part for updating pixels. Also, we should make first typed line here using typeStr variable.

```
#include "testApp.h"

//--------------------------------------------------------------
void testApp::setup(){


    video.initGrabber(40,30);



    pixels.resize(pixels.getWidth()*.05, pixels.getHeight()*.05);
//resize my pixels


    bFirst  = true;
     typeStr = "mirong jaehyun";


    ofEnableDepthTest();
    ofEnableAlphaBlending();


}

//--------------------------------------------------------------
```

```
void testApp::update(){

    video.update();
    if(video.isFrameNew())
    {

        pixels =video.getPixelsRef();

    }

}
```

2. By using 'camera', we can move real-time image every direction by using mouse. Just type camera.begin(). We've already made the variable for ofEasycam. This will also apply to see when we generate 3d-effect from 2d video images.

```
void testApp::draw(){


    ofBackground(0);

    ofSetColor(255);


    camera.begin();
```

3. Then, this part is important. To apply the right color for each pixels, and to make it visible on the screen, we should use ofColor and ofVec3f. Pixels will get the same color from the captured video image. ofVec3f has 3 variables- x, y,and z.

```
   for(std::size_t x =0; x < pixels.getWidth(); ++x)
    {
        for(std::size_t y=0;y<pixels.getHeight(); ++y)
        {
            ofColor color = pixels.getColor(x,y);
            float z =color.getBrightness();
//ofMap(color.getBrightness(),0,255,0,1024);
          ofVec3f position(x * 5 ,y * 5,z);
```

4. Now, using ofDrawBitmapString, place first typed line we made above in the ofVec3f position. Also, you should end camera here. By using ofDrawBitmapString, you do not have use ofTrueTypeFont if there's no preference for font.

```
  ofDrawBitmapString(typeStr,position);
            ofFill();
            ofSetColor(color);


        }
    }
```

```
    camera.end();

}
```

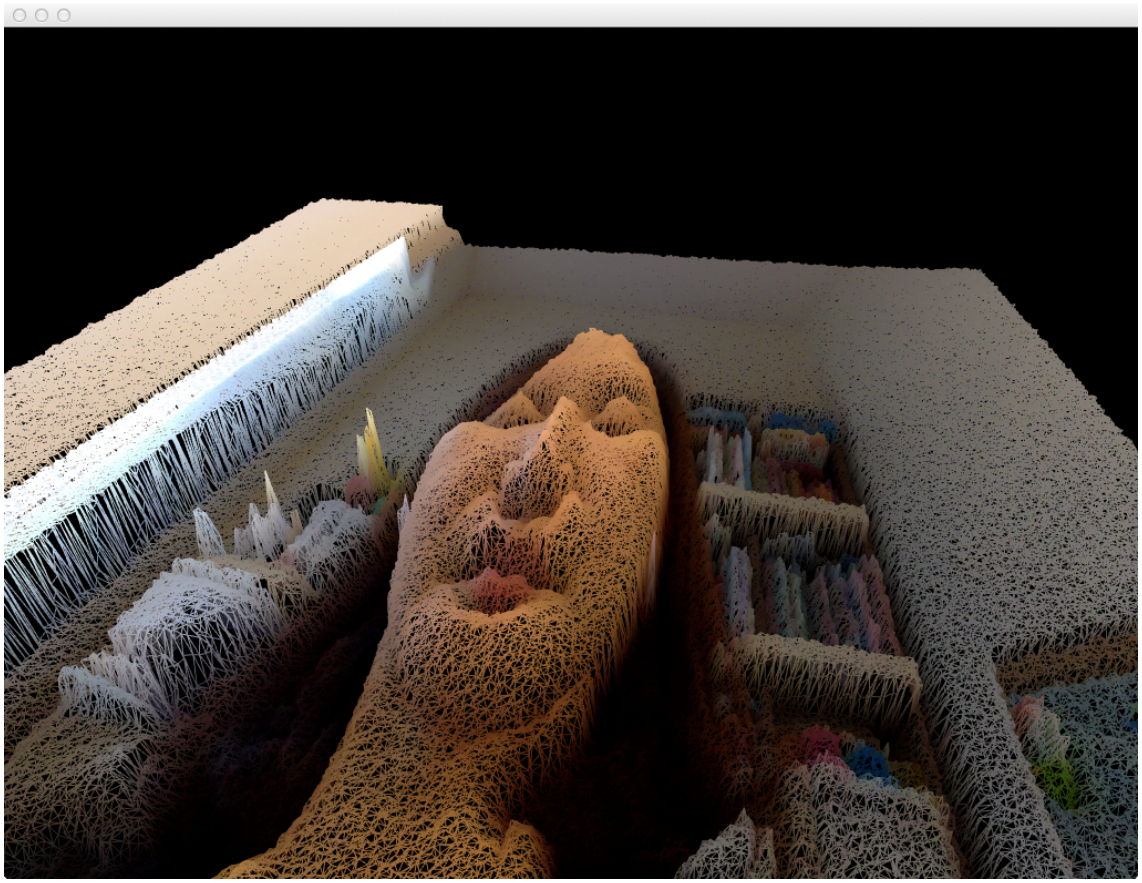
5. Now, in keyPressed part, to make sure to use delete button and append characters whenever you type.

```
void testApp::keyPressed(int key){

    if(key == OF_KEY_DEL || key == OF_KEY_BACKSPACE){
      typeStr = typeStr.substr(0, typeStr.length()-1);
    }
    else if(key == OF_KEY_RETURN ){
      typeStr += "\n";
    }else{
      if( bFirst ){
            typeStr.clear();
            bFirst = false;
      }
      typeStr.append(1, (char)key);
    }


}
```

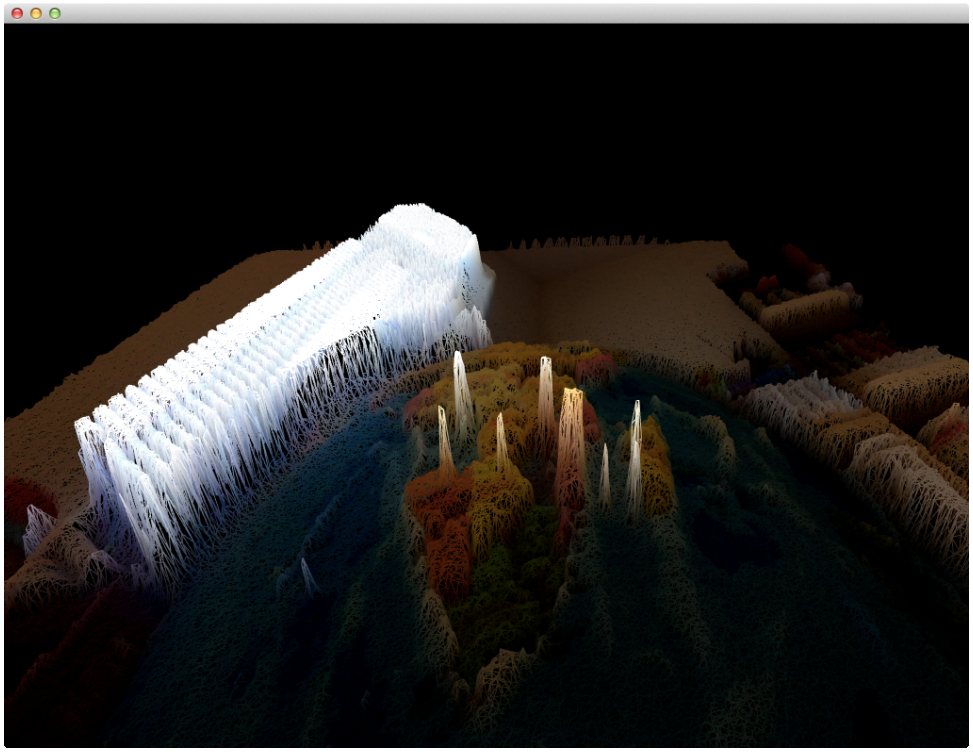
Now, when you run the program, you will get the same image above!

How to generate 3d-effect from 2d video images - by using ofPixel and ofMesh

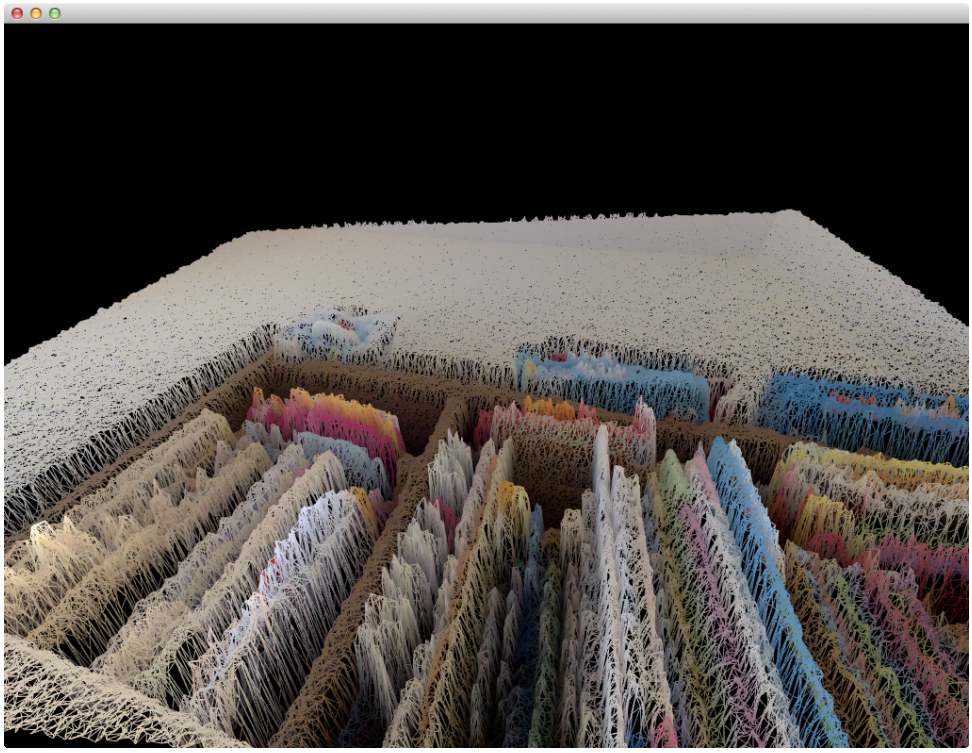
This tutorial is intended to explain how to generate 3d mesh images from 2d images captured from a video camera by using ofPixel and ofMesh classes provided in openFrameworks. Among other information contained in 2d images, this tutorial focuses on how to use the “brightness” information to transform each pixel into 3d points. It is also explained that, after 3d point clouds have been generated how they can be wired as a set of triangular mesh by using ofMesh class. In particular, this tutorial introduces how to enhance the quality of 3d rendering images by employing re-indexing the points in a mesh and smoothing by using the information of the neighborhood pixels.
The following images are obtained by the proposed method. When the brightness of pixels is transformed to 3d coordinates, it can generates an effect of emphasizing the difference between the bright areas and the dark areas.



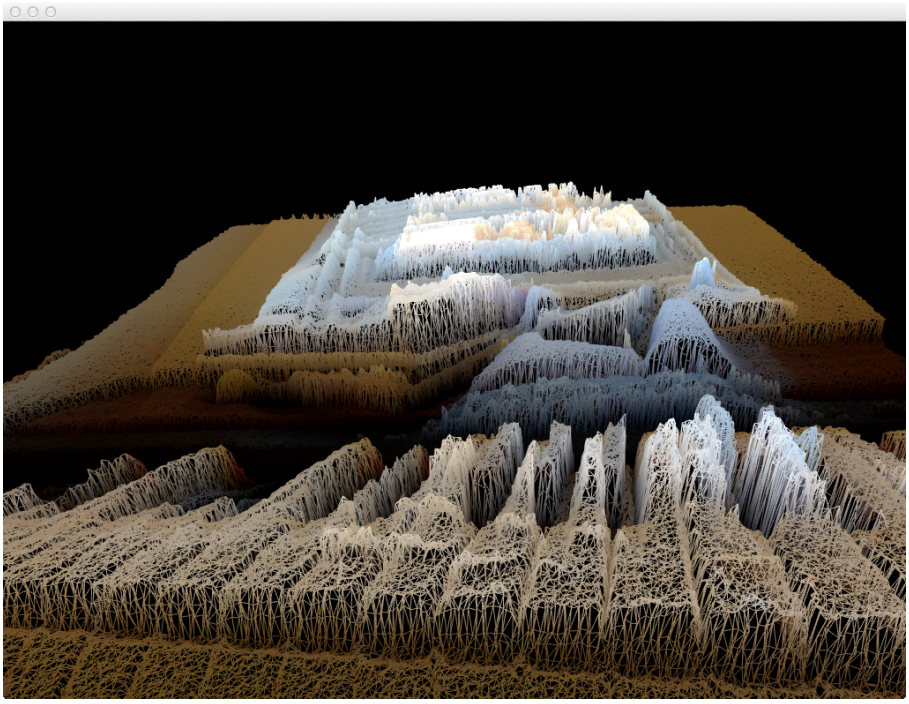
<Fig 1> Wireframe rendering of 3D mesh generated from video image



<Earth>



<Bookshelf>



<piano>

1. Creating an image by using ofPixel

The following code is to setup ofVideoGrabber class to capture video images, which will be transformed to 3d mesh for 3d rendering.

```
void ofApp::setup(){
...
ofEnableAlphaBlending();
ofEnableDepthTest();
pixel.allocate(900, 900, OF_PIXELS_BGRA);
grabber.initGrabber(900,900);
...
}


void ofApp::update(){
grabber.update();

if(grabber.isFrameNew())
{
pixels = grabber.getPixelsRef();
}
}
```

2. Transforming pixels to 3d points

Each pixel has its 2d coordinate, that is (x, y). In order for each pixel to become a point in 3d space, it needs one more dimension that is z-coordinate. Although any information contained in the pixels can be used to be generated various 3d effects, we focus on the brightness of each pixel. Specifically, brightness of each pixel is changed into the z coordinate values. The brighter a pixel is, the more it will protrude than other points in 3d space.

1) Generating a mesh of 3d points from pixels

At first, we need to initialize a mesh with OF_PRIMITIVE_TRANGLES mode. Then for each pixel in the captured image, we get ofColor from which we can obtain the brightness by calling the ofColor.getBrightness() function.

Then the brightness of each pixel is used to create the z coordinate values of the corresponding 3d points. The 3d points then is fed into ofMesh class. The point here is that each added point has index according to the sequence of addition. In the code below, an image is scanned vertically, from the left-top corner to the right-bottom, that is the index structure of points will look like the figure shown in page 6.

```
mesh.clear();
mesh.setMode(OF_PRIMITIVE_TRIANGLES);
mesh.enableColors();
int height = pixels.getHeight();
int width = pixels.getWidth();

for(std::size_t x = 0; x < width - 1; x = x + 2)
{
for(std::size_t y = 0; y < height - 1; y = y + 2)
{
ofColor color = pixels.getColor(x, y);
float z = ofMap(color.getBrightness(), 0, 255, 0, 100);

ofVec3f position(x, y, z);
mesh.addVertex(position);
mesh.addColor(color);

}
}
```

<코드1> Adding point to mesh

2) Smoothing

Depending on the captured images, the transformed 3d images can have very rough and rugged surface. In order to reduce this unintended effect, we can employ a technic of smoothing. The idea is to incorporate the information of neighboring pixels, instead of using only its own information.

```
ofColor color = pixels.getColor(x, y);
ofColor color_L = pixels.getColor(x-1, y);
ofColor color_R = pixels.getColor(x+1, y);
ofColor color_U = pixels.getColor(x, y-1);
ofColor color_D = pixels.getColor(x, y+1);

float avg = (color.getBrightness() + color_L.getBrightness() + color_R.getBrightness() +
color_U.getBrightness() + color_D.getBrightness()) / 5;
```

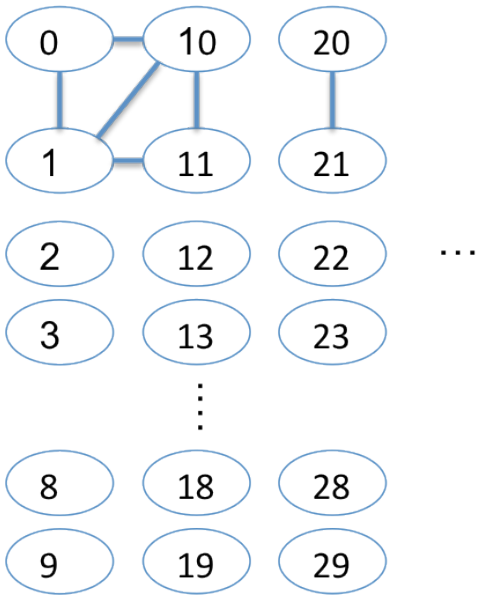
The code above introduces a variable avg indicating the average brightness of 5 pixels, a pixel at (x, y) and its 4 neighboring pixels (left, right, upper, and down). By using this averaged brightness instead of using individual value for each pixel, we can expect that the rendered 3d-mesh has smoother surfaces, enhancing the quality of result images. Furthermore, by extending the range of neighbor for averaging, the smoothing effect can be enhanced much more.

3) Re-indexing points in ofMesh

Another important technique required to enhance image quality is to re-index points added to ofMesh. As explained earlier, the sequence of adding point determines how each point is wired with other to form a triangular mesh. The way of forming triangles from points has significant effect on the look and feel of the rendered images.

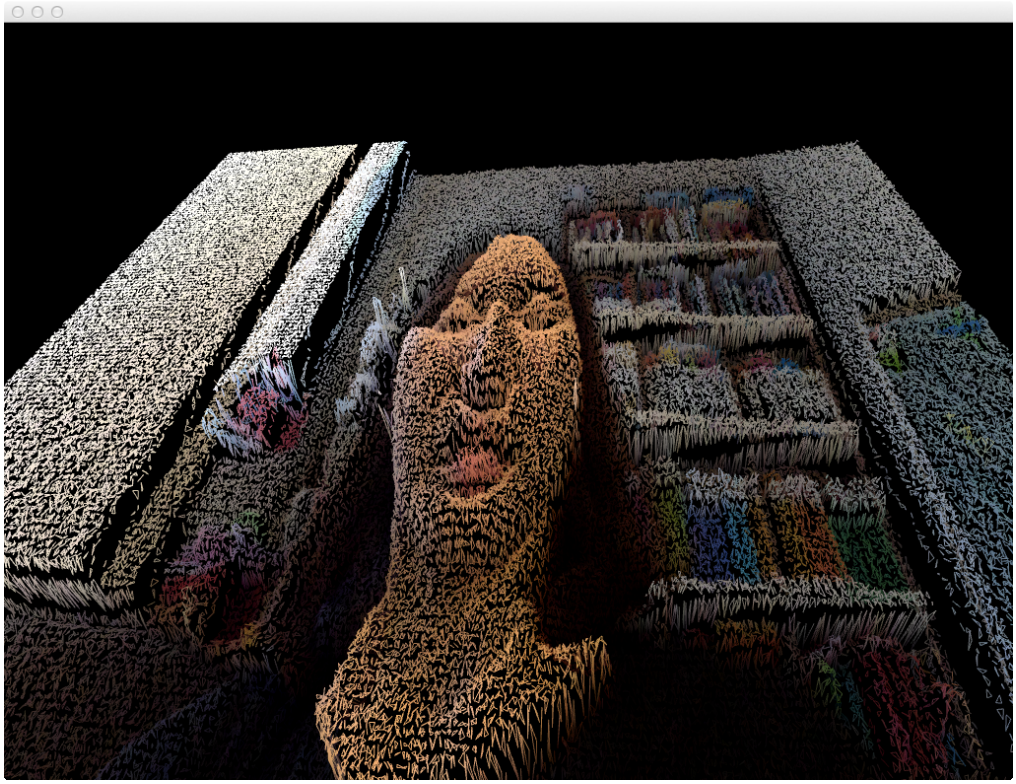
In 2d images, each pixel has 4 neighbors around them; the most natural approach is to have them connected with each other in the same triangle. This can be done by adding a new index to the points in the mesh as shown in the code below. The logic of this code can be explained as shown in Figure 6. Consider 10x10 image as in Figure 6. In the previous code the sequence of adding 3d points into a mesh was determined by the sequence of scanning pixels. As a result, the initial index of each point in the ofMesh is the number indicated in the circle (which correspond to pixel and 3d point added). This initial index structure is now transformed by the following code, which enables ofMesh to generate triangles by connecting points transformed from the neighboring pixels in the original images. The first 3 line in the code connect the points labeled as 0, 1, 10, and the following 3 line then generate a triangle by linking points 1, 11, 10.

```
for (int x = 0; x<r_width-1; x++){
for (int y=0; y<r_height-1; y++){
mesh.addIndex(x+y*r_height);            // 0
mesh.addIndex((x+1)+y*r_height);        // 1
mesh.addIndex(x+(y+1)*r_height);        // 10

mesh.addIndex((x+1)+y*r_height);        // 1
mesh.addIndex((x+1)+(y+1)*r_height);    // 11
mesh.addIndex(x+(y+1)*r_height);        // 10
}
}
```

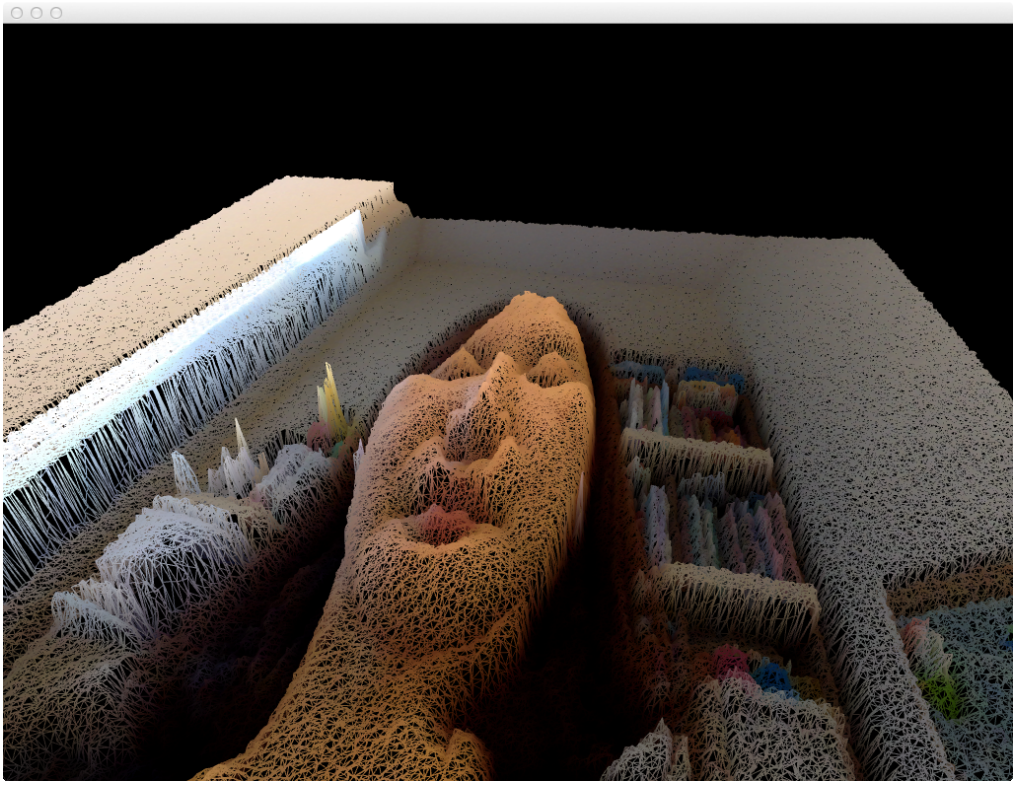


<Figure 3> Triangulation by adding indexes to points

The result of applying new indexes to points is significant as compared in the following two rendered images.



<Fig 4-1> 3D rendering of video image without smoothing and re-indexing



<Fig 4-2> 3D rendering of video image after smoothing and re-indexing of points in mesh