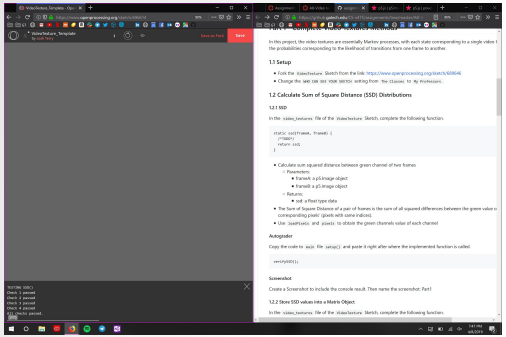
Computational Photography
Assignment #8
Video Textures

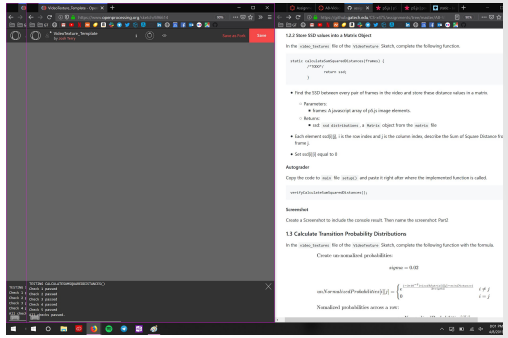
Full Name: Josh Terry

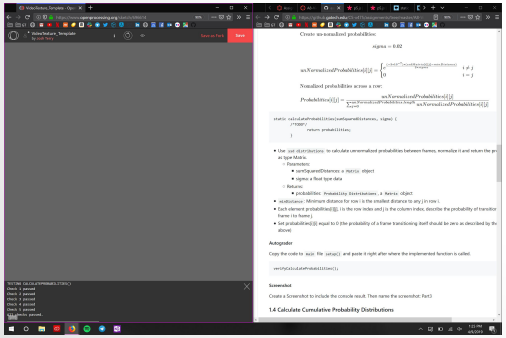
GTID: 903067329

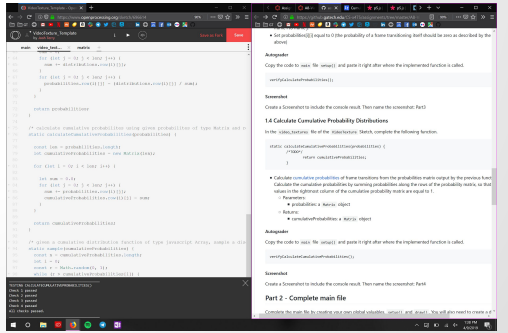
Spring 2019

Autograder Screenshots









Draw Function - SSD

```
function draw() {
    if (Math.random(0,1) < 0.2 || frame == cp.length - 1) {
            frame = VideoTexture.sample(cp.row(frame));
    } else {
            frame++;
    }
    image(gFrames[frame], 0, 0);
    runVisualizers(25.0);
}

function runVisualizers(size) {
    visualizeSSD(size);
    visualizeP(size);
}

function visualizeSSD(size) {
    for (let i = 0; i < ssd.row(frame).length; i++) {
            fill(200 - ssd.row(frame)[i] / 400000.0);
            if (i == frame) {
                    fill(255, 0, 0);
            }
            square(i*size, gFrames[frame].height + size/2, size);
    }
}
```

To the left are my draw(), runVisualizers(size), and visualizeSSD(size) functions.

I visualize the SSD distributions by passing in a “size” number that is the size of each square that represents a frame in the video.

All but one of these squares scale in brightness relative to the current frame, as brighter squares are more similar to the currently displayed frame.

The current frame is represented by a red square.

Draw Function - Probabilities

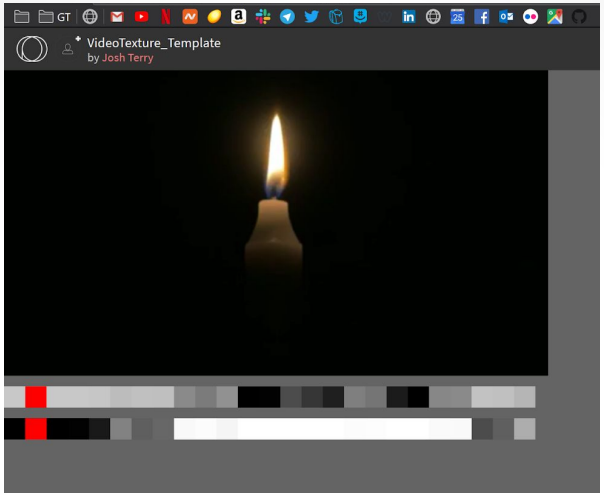
```
function visualizeP(size) {

    let max = 0.0;
    for (let i = 0; i < p.row(frame).length; i++) {
            if (p.row(frame)[i] > max) {
                    max = p.row(frame)[i];
            }
    }
    let scale = 255.0/max;

    for (let i = 0; i < p.row(frame).length; i++) {
            fill(255 - scale * p.row(frame)[i]);
            if (i == frame) {
                    fill(255, 0, 0);
            }
            square(i*size, gFrames[frame].height + size*2, size);
    }
}
```

To the left is my visualizeP(size) function.

Similar to the visualizeSSD(size) function, this takes in a size variable to draw a series of squares that vary in color based on probability.

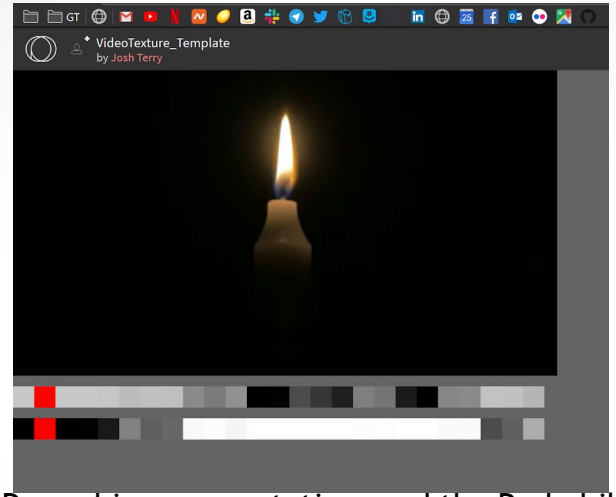
First, I iterate through each image that might come next, and I establish which one is most likely to come next. Since all images' likelihoods range between that max value and 0, I divide 255.0 by that max value to find a scale factor that images may be multiplied by for some value between 0 and 255. I then set the fill color to 255 - scale * probability, resulting in less likely images' squares being lighter and the most likely image's square to be black.
I set the current image's square to red and draw all images' squares accordingly.

SSD Graphic Representation



Above are the current frame, the SSD graphic representation, and the Probability graphic representation, from top-to-bottom.
In the SSD graphic, the red square represents the current frame, and the lightness of each other square to its left and right represents those frames' similarities to the current frame. Brighter squares represent higher similarities to the current frame.
Notice how frames closer to the current frame, or more visually similar to the current frame, have smaller SSD values than those that are farther away or more different from the current frame.

Probabilities Graphic Representation



Above are the current frame, the SSD graphic representation, and the Probability graphic representation, from top-to-bottom.

In the Probability graphic, the red square represents the current frame, and the lightness of each other square to its left and right represents those frames' likelihoods of being displayed after the current frame. The darkest square is the most likely to be displayed next, and the brightest square is the least likely.

Notice how frames more similar to the current frame tend to have far greater likelihoods of being displayed after the current frame.

SSD Graphic Representation - Dynamic Preserve

When the SSD code has been modified to allow for a greater degree of Dynamic Preservation, the SSD representations are generally smoother than before, greatly decreasing the likelihood of a random jump elsewhere in the sequence.

Probabilities Graphic Representation - Dynamic Preserve

Similarly, the Probability distribution graphic representation indicates that the video is far less likely to make a drastic jump to a less-similar frame.

Dynamic Preserve Analysis

- The Dynamic Preservation functionality within my video texture program did notably improve motion, despite being imperfect. Whereas the clock video texture used to appear to jump around fairly randomly, the video texture now looks like the clock is moving in a more-believable fashion than before. While the clock's pendulum does not perfectly swing left and right, it does appear to move with more purpose, and it does not jitter around as randomly after implementing the third function proposed in the assigned research paper.

Video Texture Results

- Link to your video texture sketch - https://www.openprocessing.org/sketch/696614

Resources

- https://github.gatech.edu/CS-x475/assignments/tree/master/A8-Video_Textures
- http://cs.colby.edu/courses/F07/cs397/papers/schodl-videoTextures-sig00.pdf