University of London

BSc Computer Science

CM2030 Graphics Programming

Final Assignment

Code PDF

By Hristo Stantchev

Legend:

1. Project Title (e.g. 3D Sine Games)
   1. Code extract(e.g. //code)  
      File name (e.g. sketch.js)

***Note: .HTML files not provided as they are the default showcased in the template.***

Final Assignment:

1. 3D Sine Games

//Bonus Additions:

//Materials + Several point lights were implemented.

//There is a Graphics texture for the grid.

//The graphics themselves are a background colour with Red value changing with height

//and there are labeled sliders that control Green and Blue values of the Graphics colour.

//custom fonts were also added

//start of my code

//bools to prevent infinite console logs

var loggedGrid = false;

//vars for cube grid

var startX = -400;

var startZ = -400;

var endX = -startX;

var endZ = -startZ;

var cubeXZ = 50;

var buffer;

var length;

//vars for sliders

var sliderHeight;

var sliderColourG;

var sliderColourB;

var sliderOffsetY = 30;

var sliderOffestX = 150;

var labelOffsetY = 0;

var labelH;

var labelG;

var labelB;

//vars for confetti

var confLocs = [];

var confTheta = [];

//vars for cam

var camPos = {

    x: 800,

    y: -600,

    z: 800

}

//slows down the circling animation of the camera

var camAnimSlower = 0.2;

//zoom out camera during circling animation

var camOffset = 200;

//called on successful font load

function drawLabels(font) {

    let keys = ['font-size', 'font-family', 'color'];

    let values = ['16px', 'Montserrat-Bold', 'white'];

    for (let i = 0; i < keys.length; i++) {

        labelH.style(keys[i], values[i]);

        labelG.style(keys[i], values[i]);

        labelB.style(keys[i], values[i]);

    }

    updateLabels();

}

function updateLabels() {

    labelH.html('Height+: ' + sliderHeight.value());

    labelG.html('Green: ' + sliderColourG.value());

    labelB.html('Blue: ' + sliderColourB.value());

}

//called on failed font load

function throwfontErr(font) {

    console.error("Error Loading font: ", font.toString());

}

//function to set positions for the camera

function setCam() {

    camera(camPos.x, camPos.y, camPos.z, 0, 1, 0);

}

//sets material and draws box

function drawCube(size, length, textureValue) {

    //cube draw options

    buffer.background(textureValue, sliderColourG.value(), sliderColourB.value());

    specularMaterial(180);

    shininess(30);

    texture(buffer);

    box(size, length, size);

}

//helper function to generate random coordinates for confetti

function genRandomConfettiLoc() {

    return { x: random(-500, 501), y: random(-800, 1), z: random(-random(-500, 501)) }

}

function drawConfettiplane() {

    normalMaterial();

    plane(15, 15);

}

//confetti function draws confetti using confLocs and confTheta data

function confetti() {

    for (let i = 0; i < confLocs.length; i++) {

        push();

        //set confeti location

        translate(confLocs[i].x, confLocs[i].y, confLocs[i].z);

        //animate falling by incrementing Y

        confLocs[i].y++;

        if (confLocs[i].y > 0) {

            //res animation after falling too far down

            confLocs[i].y = -800;

        }

        rotateX(confTheta[i]);

        confTheta[i] += 10;

        drawConfettiplane();

        pop();

    }

}

//function for drawing point lights from centre and opposing sides

function drawLights() {

    pointLight(255, 255, 255, 0, -301, 0);

    pointLight(255, 255, 255, startX - 50, length, startZ - 50);

    pointLight(255, 255, 255, endX + 50, length, endZ + 50);

    pointLight(255, 255, 255, startX - 50, 0, startZ - 50);

    pointLight(255, 255, 255, endX + 50, 0, endZ + 50);

}

//draws a Cube Grid based on a starting and ending X and Z positions

//it takes a size argument to determine the cube size

function drawCubeGrid(startX, startZ, endX, endZ, size) {

    //loop through X and Z positions to draw the grid.

    for (let posX = startX; posX < endX; posX += size) {

        for (let posZ = startZ; posZ < endZ; posZ += size) {

            //Push and pop reset translate positions to 0,0,0

            push();

            translate(posX, 0, posZ);

            //map out a sine wave for the Y value of the box by using distance from centre

            var distance = dist(posX, 0, posZ, 0, 0, 0);

            length = map(sin(distance + frameCount), -1, 1, 100 + sliderHeight.value(), 300 + sliderHeight.value());

            textureValue = map(sin(distance + frameCount), -1, 1, 0, 255);

            drawCube(size, length, textureValue);

            pop();

            //REMOVE COMMENTS FOR TESTING PURPOSES

            // if (!loggedGrid)

            //     console.log("Cube X: ", posX, "\ncube Z: ", posZ, "\nCube dist: ", distance);

        }

    }

    //REMOVE COMMENTS FOR TESTING PURPOSES

    // loggedGrid = true;

}

function preload() {

    buffer = createGraphics(50, 50);

    sliderHeight = createSlider(0, 400, 0, 5);

    sliderColourG = createSlider(0, 255, 130, 1);

    sliderColourB = createSlider(0, 255, 255, 1);

}

//end of my code

function setup() {

    //start of template code

    createCanvas(900, 800, WEBGL);

    //end of template code

    //start of my code

    //set camera start Position

    setCam();

    //generate confetti locations

    for (let i = 0; i < 200; i++) {

        confLocs.push(genRandomConfettiLoc());

        confTheta.push(random(0, 361));

    }

    //set sliders positions

    sliderHeight.position(sliderOffestX, sliderOffsetY);

    sliderColourG.position(sliderOffestX \* 2, sliderOffsetY);

    sliderColourB.position(sliderOffestX \* 3, sliderOffsetY);

    //set slider labels

    labelH = createP();

    labelB = createP();

    labelG = createP();

    labelH.position(sliderOffestX, labelOffsetY);

    labelG.position(sliderOffestX \* 2, labelOffsetY);

    labelB.position(sliderOffestX \* 3, labelOffsetY);

    //label sliders

    loadFont("assets/Montserrat-Bold.ttf", drawLabels, throwfontErr);

    //end of my code

}

function draw() {

    //start of template code

    background(125);

    angleMode(DEGREES);

    //end of template code

    //start of my code

    //map circular camera movement on X and Z axis

    camPos.x = cos(frameCount \* camAnimSlower) \* (height + camOffset);

    camPos.z = sin(frameCount \* camAnimSlower) \* (height + camOffset);

    setCam();

    //draw lights

    drawLights();

    //draw cube grid

    drawCubeGrid(startX, startZ, endX, endZ, cubeXZ);

    //draw confetti

    confetti();

    //update Labels

    updateLabels();

    //end of my code

}

* 1. sketch.js

1. Average Face

//extensions:

//Key press events trigger the change of the left photo:

//-Implemented using a random number generator (RNG) and the keyPressed() function

//-calls loop() as draw() is executed once before noloop() (applies to next extension)

//faces change dynamically change values following MouseX:

//- Created using mappings to linear interpolation

//- In order to draw updates, mouseMoved() function (called on event when mouse is moved) calls loop()

var imgs = [];

var avgImg;

var numOfImages = 30;

//start of my code

var imageIndexLeft;

//end of my code

//////////////////////////////////////////////////////////

function preload() { // preload() runs once

    //start of my code

    //push all images to imgs array

    for (let i = 0; i < numOfImages; i++) {

        let filename = "assets/" + i + ".jpg";

        imgs.push(loadImage(filename));

    }

    //set index for left image

    setImageIndexLeft();

    //end of my code

}

//////////////////////////////////////////////////////////

function setup() {

    //start of my code

    createCanvas(imgs[0].width \* 2, imgs[0].height);

    //end of my code

    pixelDensity(1);

    //start of my code

    avgImg = createGraphics(imgs[0].width, imgs[0].height);

    //end of my code

}

//////////////////////////////////////////////////////////

function draw() {

    background(125);

    //start of my code

    //draw first image

    image(imgs[imageIndexLeft], 0, 0);

    //load pixel arrays for left images and right image

    avgImg.loadPixels();

    imgs.forEach(element => {

        element.loadPixels();

    });

    //loop through X and Y coordinates

    for (let x = 0; x < imgs[0].width; x++) {

        for (let y = 0; y < imgs[0].height; y++) {

            //get pixel index

            let pixelIndex = (x + (y \* imgs[0].width)) \* 4;

            let sumR = 0;

            let sumG = 0;

            let sumB = 0;

            //get sum of all images' RGB values at this pixel

            imgs.forEach(img => {

                sumR += img.pixels[pixelIndex];

                sumG += img.pixels[pixelIndex + 1];

                sumB += img.pixels[pixelIndex + 2];

            });

            //set RGB values to average of all images' RGBs at this pixel, set alpha to max

            avgImg.pixels[pixelIndex] = round(sumR / imgs.length);

            avgImg.pixels[pixelIndex + 1] = round(sumG / imgs.length);

            avgImg.pixels[pixelIndex + 2] = round(sumB / imgs.length);

            avgImg.pixels[pixelIndex + 3] = 255;

        }

    }

    //call linear interpolation function

    lerpMouseXToRightImg();

    avgImg.updatePixels();

    //draw right image and stop loop

    image(avgImg, imgs[0].width, 0);

    noLoop();

    //end of my code

}

//start of my code

//on any key pressed swap left image

function keyPressed() {

    setImageIndexLeft();

    loop();

}

//draws new random image

function drawRandomImg() {

    setImageIndexLeft();

    image(imgs[imageIndexLeft], 0, 0);

}

//sets random index for left image

function setImageIndexLeft() {

    imageIndexLeft = round(random(0, imgs.length));

}

//restarts draw loop

function mouseMoved() {

    loop();

}

//linearly interpolates all pixels on avgImg to pixels of left img based on mouseX movement

function lerpMouseXToRightImg() {

    for (let pixel = 0; pixel < imgs[0].pixels.length; pixel++) {

        avgImg.pixels[pixel] = lerp(avgImg.pixels[pixel], imgs[imageIndexLeft].pixels[pixel], map(mouseX, 0, width, 0, 1));

    }

}

//end of my code

* 1. sketch.js

1. Your Own Instagram Filter

// Image of Husky Creative commons from Wikipedia:

// https://en.wikipedia.org/wiki/Dog#/media/File:Siberian\_Husky\_pho.jpg

// Extensions:

// Pressing any key changes the filters from sepia to greyscale and vice versa:

// - implemented using keyPressed() function to trigger change a bool

// - also calls loop() to update the image on the right

// Slider implemented to control the matrix value:

// - default matrix value is at 64, however can be changed from values 1-128 (determined by testing)

// - these affect the convolution and radial blur of the image

// - genMatrix() function implemented to shorten code for matrix and generate updated visuals on slider change

// - short tutorial written on how to operate both extensions below left image

var imgIn;

//start of my code

//true for sepia, false for grayscale

var sepOrGrey = true;

var matrixValue = 64;

var matrixSlider;

var matrix;

//end of my code

/////////////////////////////////////////////////////////////////

function preload() {

  imgIn = loadImage("assets/husky.jpg");

  //start of my code

  matrixSlider = createSlider(1, 128, matrixValue, 1);

  //end of my code

}

/////////////////////////////////////////////////////////////////

function setup() {

  createCanvas((imgIn.width \* 2), imgIn.height + 30);

  //start of my code

  matrixSlider.position(5, imgIn.height + 70);

  //end of my code

}

/////////////////////////////////////////////////////////////////

function draw() {

  //start of my code

  matrixValue = matrixSlider.value();

  matrix = genMatrix();

  //end of my code

  background(255);

  image(imgIn, 0, 0);

  image(earlyBirdFilter(imgIn), imgIn.width, 0);

  textSize(15);

  text("Press any key to switch between sepia and greyscale.", 0, imgIn.height + 15);

  text("Move Slider to change amount of radial blur and convolution.", 0, imgIn.height + 30);

  noLoop();

}

/////////////////////////////////////////////////////////////////

function mousePressed() {

  loop();

}

//start of my code

//functions that ensure slider movement with Mouse And Keyboard call draw function

function mouseReleased() {

  loop();

}

function keyPressed(){

  //start of my code

  sepOrGrey = !sepOrGrey;

  //end of my code

  loop();

}

//end of my code

/////////////////////////////////////////////////////////////////

//start of my code

function genMatrix() {

  arr = [1 / matrixValue, 1 / matrixValue, 1 / matrixValue, 1 / matrixValue, 1 / matrixValue, 1 / matrixValue, 1 / matrixValue, 1 / matrixValue];

  matr = [];

  for (let i = 0; i < 8; i++) {

    matr.push(arr);

  }

  return matr;

}

function sepiaFilter(img) {

  var resultImg = createImage(img.width, img.height);

  resultImg.loadPixels();

  img.loadPixels();

  for (let x = 0; x < img.width; x++) {

    for (let y = 0; y < img.height; y++) {

      let index = ((y \* img.width) + x) \* 4;

      let oldRed = img.pixels[index];

      let oldGreen = img.pixels[index + 1];

      let oldBlue = img.pixels[index + 2];

      let newRed = (oldRed \* .393) + (oldGreen \* .769) + (oldBlue \* .189);

      let newGreen = (oldRed \* .349) + (oldGreen \* .686) + (oldBlue \* .168);

      let newBlue = (oldRed \* .272) + (oldGreen \* .534) + (oldBlue \* .131);

      resultImg.pixels[index] = newRed;

      resultImg.pixels[index + 1] = newGreen;

      resultImg.pixels[index + 2] = newBlue;

      resultImg.pixels[index + 3] = 255;

    }

  }

  resultImg.updatePixels();

  return resultImg;

}

function darkCorners(img) {

  var resultImg = createImage(img.width, img.height);

  img.loadPixels();

  resultImg.loadPixels();

  for (let x = 0; x < img.width; x++) {

    for (let y = 0; y < img.height; y++) {

      let index = ((y \* img.width) + x) \* 4;

      let oldRed = img.pixels[index];

      let oldGreen = img.pixels[index + 1];

      let oldBlue = img.pixels[index + 2];

      let distFromCentre = dist(round(img.width / 2), round(img.height / 2), x, y);

      let dynLum;

      if (distFromCentre < 300) {

        dynLum = 1;

      }

      else if (distFromCentre >= 300 & distFromCentre < 450) {

        dynLum = map(distFromCentre, 300, 449, 1, 0.4);

      }

      else {

        dynLum = map(distFromCentre, 450, 600, 0.4, 0);

      }

      let newRed = constrain(oldRed \* dynLum, 0, 255);

      let newGreen = constrain(oldGreen \* dynLum, 0, 255);

      let newBlue = constrain(oldBlue \* dynLum, 0, 255);

      resultImg.pixels[index] = newRed;

      resultImg.pixels[index + 1] = newGreen;

      resultImg.pixels[index + 2] = newBlue;

      resultImg.pixels[index + 3] = 255;

    }

  }

  resultImg.updatePixels();

  return resultImg;

}

function radialBlurFilter(img) {

  var resultImg = createImage(img.width, img.height);

  resultImg.loadPixels();

  img.loadPixels();

  for (var x = 0; x < img.width; x++) {

    for (var y = 0; y < img.height; y++) {

      var index = (x + y \* img.width) \* 4;

      var c = convolution(x, y, matrix, matrix.length, img);

      let oldRed = img.pixels[index];

      let oldGreen = img.pixels[index + 1];

      let oldBlue = img.pixels[index + 2];

      let distFromCentre = dist(img.width / 2, img.height / 2, x, y);

      let dynBlur = 1;

      if (distFromCentre < 100) {

        dynBlur = 0;

      }

      else {

        dynBlur = constrain(map(distFromCentre, 100, 300, 0, 1), 0, 1);

      }

      resultImg.pixels[index + 0] = c[0] \* dynBlur + oldRed \* (1 - dynBlur);

      resultImg.pixels[index + 1] = c[1] \* dynBlur + oldGreen \* (1 - dynBlur);

      resultImg.pixels[index + 2] = c[2] \* dynBlur + oldBlue \* (1 - dynBlur);

      resultImg.pixels[index + 3] = 255;

    }

  }

  resultImg.updatePixels();

  return resultImg;

}

function convolution(x, y, matrix, matrixSize, img) {

  var totalRed = 0.0;

  var totalGreen = 0.0;

  var totalBlue = 0.0;

  var offset = floor(matrixSize / 2);

  // convolution matrix loop

  for (var i = 0; i < matrixSize; i++) {

    for (var j = 0; j < matrixSize; j++) {

      // Get pixel loc within convolution matrix

      var xloc = x + i - offset;

      var yloc = y + j - offset;

      var index = (xloc + img.width \* yloc) \* 4;

      // ensure we don't address a pixel that doesn't exist

      index = constrain(index, 0, img.pixels.length - 1);

      // multiply all values with the mask and sum up

      totalRed += img.pixels[index + 0] \* matrix[i][j];

      totalGreen += img.pixels[index + 1] \* matrix[i][j];

      totalBlue += img.pixels[index + 2] \* matrix[i][j];

    }

  }

  // return the new color

  return [totalRed, totalGreen, totalBlue];

}

function borderFilter(img) {

  let resultImg = createGraphics(img.width, img.height);

  resultImg.image(img, 0, 0);

  resultImg.noFill();

  resultImg.stroke(255);

  resultImg.strokeWeight(40);

  resultImg.rect(0, 0, img.width, img.height, 40);

  return resultImg;

}

function greyscaleFilter(img) {

  var resultImg = createImage(img.width, img.height);

  resultImg.loadPixels();

  img.loadPixels();

  for (x = 0; x < resultImg.width; x++) {

    for (y = 0; y < resultImg.height; y++) {

      var index = (x + y \* resultImg.width) \* 4;

      var r = img.pixels[index + 0];

      var g = img.pixels[index + 1];

      var b = img.pixels[index + 2];

      var gray = r \* 0.299 + g \* 0.587 + b \* 0.114; // LUMA ratios

      resultImg.pixels[index + 0] = resultImg.pixels[index + 1] = resultImg.pixels[index + 2] = gray;

      resultImg.pixels[index + 3] = 255;

    }

  }

  resultImg.updatePixels();

  return resultImg;

}

//end of my code

function earlyBirdFilter(img) {

  var resultImg = createImage(img.width, img.height);

  //start of my code

  if (sepOrGrey) {

    resultImg = sepiaFilter(img);

  }

  else {

    resultImg = greyscaleFilter(img);

  }

  //end of my code

  resultImg = darkCorners(resultImg);

  resultImg = radialBlurFilter(resultImg);

  resultImg = borderFilter(resultImg)

  return resultImg;

}

* 1. sketch.js

1. Webcam Piano

//extensions:

//Implemented Audio functionalities:

// - Notes now play sounds from a p5 MonoSynth object

// - p5 audio class implemented

// - Notes data comes from a static notes array (index chosen based on grid X) and a mapped octave (based on grid Y)

//UI Implementations:

// - Audio ON/OFF toggle comes from Backspace key

// - Slider for threshold (defaults at 50)

// - Slider for volume in % (defaults at 50%)

//Visual Implementations(in Grid.js):

// - Sparks fly off on 3% of detections

// - notes no longer circles - now rectangles

// - rectangular notes rotate as well

// - colour of rectangular notes

// - Blue value mix now includes frameCount into map function

var video;

var prevImg;

var diffImg;

var currImg;

var thresholdSlider;

var threshold;

//start of my code

var grid;

var volSlider;

var vol;

//end of my code

function setup() {

    createCanvas(640 \* 2, 480);

    pixelDensity(1);

    //start of my code

    angleMode(DEGREES);

    //end of my code

    video = createCapture(VIDEO);

    video.hide();

    thresholdSlider = createSlider(0, 255, 50);

    thresholdSlider.position(20, 20);

    //start of my code

    volSlider = createSlider(0, 100, 50, 1);

    volSlider.position(20, 50);

    grid = new Grid(640, 480);

    //end of my code

}

function draw() {

    background(0);

    image(video, 0, 0);

    currImg = createImage(video.width, video.height);

    currImg.copy(video, 0, 0, video.width, video.height, 0, 0, video.width, video.height);

    //start of my code

    currImg.resize(video.width / 4, video.height / 4);

    currImg.filter(BLUR, 3);

    //end of my code

    diffImg = createImage(video.width, video.height);

    diffImg.loadPixels();

    //start of my code

    diffImg.resize(video.width / 4, video.height / 4);

    //end of my code

    threshold = thresholdSlider.value();

    //start of my code

    vol = volSlider.value();

    //end of my code

    if (typeof prevImg !== 'undefined') {

        prevImg.loadPixels();

        currImg.loadPixels();

        for (var x = 0; x < currImg.width; x += 1) {

            for (var y = 0; y < currImg.height; y += 1) {

                var index = (x + (y \* currImg.width)) \* 4;

                var redSource = currImg.pixels[index + 0];

                var greenSource = currImg.pixels[index + 1];

                var blueSource = currImg.pixels[index + 2];

                var redBack = prevImg.pixels[index + 0];

                var greenBack = prevImg.pixels[index + 1];

                var blueBack = prevImg.pixels[index + 2];

                var d = dist(redSource, greenSource, blueSource, redBack, greenBack, blueBack);

                if (d > threshold) {

                    diffImg.pixels[index + 0] = 0;

                    diffImg.pixels[index + 1] = 0;

                    diffImg.pixels[index + 2] = 0;

                    diffImg.pixels[index + 3] = 255;

                } else {

                    diffImg.pixels[index + 0] = 255;

                    diffImg.pixels[index + 1] = 255;

                    diffImg.pixels[index + 2] = 255;

                    diffImg.pixels[index + 3] = 255;

                }

            }

        }

    }

    diffImg.updatePixels();

    image(diffImg, 640, 0);

    prevImg = createImage(currImg.width, currImg.height);

    prevImg.copy(currImg, 0, 0, currImg.width, currImg.height, 0, 0, currImg.width, currImg.height);

    console.log("saved new background");

    noFill();

    stroke(255);

    text(threshold, 160, 35);

    //start of my code

    text(vol+'%',160,65);

    text("Press BACKSPACE to toggle audio ON/OFF", currImg.width / 2 + 100, 10);

    text(setAudioText(), currImg.width / 2 + 150, 25);

    grid.run(diffImg,vol);

    //end of my code

}

function keyPressed() {

    //start of my code

    if (keyCode === BACKSPACE) {

        grid.toggleAudio();

    }

    //end of my code

}

//start of my code

function setAudioText() {

    let audioText = "Audio is OFF";

    stroke(255, 0, 0);

    if (grid.audioEnabled) {

        stroke(0, 255, 0);

        audioText = "Audio is ON";

    }

    return audioText;

}

//end of my code

// faster method for calculating color similarity which does not calculate root.

// Only needed if dist() runs slow

function distSquared(x1, y1, z1, x2, y2, z2) {

    var d = (x2 - x1) \* (x2 - x1) + (y2 - y1) \* (y2 - y1) + (z2 - z1) \* (z2 - z1);

    return d;

}

* 1. sketch.js

class Grid {

  /////////////////////////////////

  constructor(\_w, \_h) {

    this.gridWidth = \_w;

    this.gridHeight = \_h;

    this.noteSize = 40;

    this.notePos = [];

    this.noteState = [];

    this.audioEnabled = false;

    this.userAudioOn = false;

    this.notes = ['C', 'Db', 'D', 'Eb', 'D', 'F', 'Gb', 'G', 'Ab', 'A', 'Bb', 'B'];

    this.synth;

    this.volume;

    // initalise grid structure and state

    for (var x = 0; x < \_w; x += this.noteSize) {

      var posColumn = [];

      var stateColumn = [];

      for (var y = 0; y < \_h; y += this.noteSize) {

        posColumn.push(createVector(x + this.noteSize / 2, y + this.noteSize / 2));

        stateColumn.push(0);

      }

      this.notePos.push(posColumn);

      this.noteState.push(stateColumn);

    }

  }

  /////////////////////////////////

  run(img, vol) {

    img.loadPixels();

    this.findActiveNotes(img);

    this.drawActiveNotes(img);

    //start of my code

    this.volume = vol \* 0.01;

    //end of my code

  }

  /////////////////////////////////

  drawActiveNotes(img) {

    // draw active notes

    fill(255);

    noStroke();

    //start of my code

    this.synth = new p5.MonoSynth();

    //end of my code

    for (var i = 0; i < this.notePos.length; i++) {

      for (var j = 0; j < this.notePos[i].length; j++) {

        var x = this.notePos[i][j].x;

        var y = this.notePos[i][j].y;

        if (this.noteState[i][j] > 0) {

          var alpha = this.noteState[i][j] \* 200;

          var c1 = color(0, 0, 255, alpha);

          var c2 = color(255, 255, 0, alpha);

          var mix = lerpColor(c1, c2, map(i + //start of my code

            sin(frameCount), -1,//end of my code

            this.notePos.length, 0, 1));

          fill(mix);

          var s = this.noteState[i][j];

          //start of my code

          push();

          translate(x, y);

          if (random(0, 100) > 97) {

            this.drawSparks();

          }

          rotate(map(sin(frameCount), -1, 1, 0, 360));

          rect(0, 0, this.noteSize \* s, this.noteSize \* s);

          pop();

          if (this.audioEnabled) {

            let octave = round(map(y, 0, this.gridHeight, 0, 11));

            let noteIndex = round(map(x, 0, this.gridWidth, 0, this.notes.length - 1))

            this.playSynth(noteIndex, octave);

          }

          //end of my code

        }

        this.noteState[i][j] -= 0.05;

        this.noteState[i][j] = constrain(this.noteState[i][j], 0, 1);

      }

    }

  }

  /////////////////////////////////

  findActiveNotes(img) {

    for (var x = 0; x < img.width; x += 1) {

      for (var y = 0; y < img.height; y += 1) {

        var index = (x + (y \* img.width)) \* 4;

        var state = img.pixels[index + 0];

        if (state == 0) { // if pixel is black (ie there is movement)

          // find which note to activate

          var screenX = map(x, 0, img.width, 0, this.gridWidth);

          var screenY = map(y, 0, img.height, 0, this.gridHeight);

          var i = int(screenX / this.noteSize);

          var j = int(screenY / this.noteSize);

          this.noteState[i][j] = 1;

        }

      }

    }

  }

  //start of my code

  drawSparks() {

    let sparkCount = random(3, 12);

    for (let i = 0; i < sparkCount; i++) {

      push();

      rotate(map(random(i, i + 50), 0, sparkCount, 0, 360));

      fill(255, 255, 0);

      let sparkX = map(sin(frameCount + i), -1, 1, 0, 40);

      let sparkY = map(sin(frameCount + i), -1, 1, 0, 50);

      let sparkH = map(sin(frameCount + i), -1, 1, 15, 0);

      let sparkW = sparkH / 4;

      rect(sparkX, sparkY, sparkW, sparkH);

      pop();

    }

  }

  toggleAudio() {

    if (this.audioEnabled) {

      outputVolume(0);

      this.audioEnabled = false;

    }

    else {

      if (!this.userAudioOn) {

        userStartAudio();

        this.userAudioOn = true;

      }

      outputVolume(this.volume);

      this.audioEnabled = true;

    }

  }

  playSynth(indexNote, indexOctave) {

    outputVolume(this.volume);

    let noteOct = this.genNote(this.notes[indexNote], indexOctave);

    this.synth.play(noteOct, 0, 0.001);

  }

  genNote(note, octave) {

    return note + octave.toString();

  }

  //end of my code

}

* 1. Grid.js

End of Paper

Thank you for your time and attention!