EXCERSIZES
INTERACT WITH CAMERAS

creative
scripting
m e a l s

06

exercises

BLACK AND WHITE CAMERA

INTERACTIVE PIXELATE LIVE VIDEO_DRAW WHITE PIXEL

PATTERN 2D LIVE VIDEO TEXT BINARY

PATTERN 2D LIVE VIDEO TEXT BINARY2

PIXELATE LIVE VIDEO SCALED RECTS

REMOVE BACKGROUND

MOTION TRACKING

COLOR TRACKING

sketches

Black And White Camera

DESCRIPTION

Using this sketch try to develop the following projects

Pattern2D Live Video Text Binary and Pattern 2D Live Video Text Binary 2
Compine previous projects and apply color effects to every object that holds the correct color.

CODE

```
import processing.video.*;

int video_width  = 320;
int video_height = 240;
int num_pixels = (video_width * video_height);
int previous_frame[];

Capture video;

void setup()
{
  //set up size of window and video
  size(320, 240);
  frameRate(60);
  loadPixels();
  previous_frame = new int [num_pixels];

  video = new Capture(this, video_width, video_height, 30);
  video.start(); // Start capturing the images from the camera
}

void captureEvent(Capture video) { video.read(); }

void draw()
{
  background(0);

  for (int x = 0; x < video.width; x++) {
    for (int y = 0; y < video.height; y++ ) {
      //calculate the 1D location from a 2D grid
      int loc = x + y*video.width;

      int threshold = 100; // check brightness
      //calculate a threshold from 0-255 based on mouseX
      // int threshold = int(((float) mouseX / width) * 255);
      color c;
      //do a threshold test based on brightness
      //the resulting pixel is only white or black
      if (brightness(video.pixels[loc]) > threshold) {
        c = color(255, 255, 255);
      }
      else {
        c = color(0, 0, 0);
      }
      pixels[loc] = c;
    }
  }
  updatePixels();
}
```

Pixelate Live Video with Scaled Rects

DESCRIPTION

This projects uses a patterns of black rects to represent the captured image. It transforms the size of the rects according to the brightness of the pixels.

CODE

```
PImage bg;

import processing.video.*;
Capture video;

int cellSize = 15;
int cols, rows;


void setup()
{
  size(640, 480);
  cols = width / cellSize;
  rows = height / cellSize;
  colorMode(RGB, 255, 255, 255, 100);
  rectMode(CENTER);
  video = new Capture(this, width, height, 15);
  video.start();
  frameRate(30);
}

void draw()
{

  if (video.available()) {
    video.read();
    video.loadPixels();
    background(255);

    for (int i = 0; i < cols;i++) {
      for (int j = 0; j < rows;j++) {
        int x = i * cellSize;
        int y = j * cellSize;
        int loc = (video.width - x - 1) + y*video.width;
        color c = video.pixels[loc];
        float sz = (brightness(c) / 255.0) * cellSize;
        fill(0);
        noStroke();
        rect(x + cellSize/2, y + cellSize/2, sz, sz);
      }
    }
  }
}
```

Remove background

CODE

```
Capture video; // Variable for capture device
PImage backgroundImage; // Saved background

float threshold = 10; // How different must a pixel be to be a foreground pixel

void setup() {
  size(320,240);
  video = new Capture(this, width, height, 30);
  video.start();
  // Create an empty image the same size as the video
  backgroundImage = createImage(video.width,video.height,RGB);
}

void draw() {
  // Capture video
  if (video.available()) {
    video.read();
  }

  // We are looking at the video's pixels, the memorized backgroundImage's pixels,
as well as accessing the display pixels.
  // So we must loadPixels() for all!
  loadPixels();
  video.loadPixels();
  backgroundImage.loadPixels();

  // Begin loop to walk through every pixel
  for (int x = 0; x < video.width; x ++ ) {
    for (int y = 0; y < video.height; y ++ ) {
      int loc = x + y*video.width; // Step 1, what is the 1D pixel location
      color fgColor = video.pixels[loc]; // Step 2, what is the foreground color

      // Step 3, what is the background color
      color bgColor = backgroundImage.pixels[loc];

      // Step 4, compare the foreground and background color
      float r1 = red(fgColor);
      float g1 = green(fgColor);
      float b1 = blue(fgColor);
      float r2 = red(bgColor);
      float g2 = green(bgColor);
      float b2 = blue(bgColor);
      float diff = dist(r1,g1,b1,r2,g2,b2);
```

```
// Learning Processing
// Daniel Shiffman
// http://www.learningprocessing.com

// Example 16-12: Simple background
removal

// Click the mouse to memorize a
current background image
import processing.video.*;
```

```
      // Step 5, Is the foreground color different from the background color
      if (diff > threshold) {
        // If so, display the foreground color
        pixels[loc] = fgColor;
      } else {
        // If not, display green
        pixels[loc] = color(0,255,0);
        // We could choose to replace the background pixels with something other
than the color green!
      }
    }
  }
  updatePixels();
}

void mousePressed() {
  // Copying the current frame of video into the backgroundImage object
  // Note copy takes 5 arguments:
  // The source image
  // x,y,width, and height of region to be copied from the source
  // x,y,width, and height of copy destination
  backgroundImage.copy(video,0,0,video.width,video.height,0,0,video.
width,video.height);
  backgroundImage.updatePixels();
}
```

MotionTracking

CODE

```
import processing.video.*;

int numPixels;
int[] previousFrame;
Capture video;

void setup() {
  size(640, 480);

  // This the default video input, see the GettingStartedCapture
  // example if it creates an error
  video = new Capture(this, width, height);

  // Start capturing the images from the camera
  video.start();

  numPixels = video.width * video.height;
  // Create an array to store the previously captured frame
  previousFrame = new int[numPixels];
  loadPixels();
}

void draw() {
  if (video.available()) {

    // When using video to manipulate the screen, use video.available() and
    // video.read() inside the draw() method so that it's safe to draw to the screen

video.read(); // Read the new frame from the camera
    video.loadPixels(); // Make its pixels[] array available

    int movementSum = 0; // Amount of movement in the frame
    for (int i = 0; i < numPixels; i++) { // For each pixel in the video frame...
      color currColor = video.pixels[i];
      color prevColor = previousFrame[i];

      // Extract the red, green, and blue components from current pixel
      float currR = red(currColor); //(currColor >> 16) & 0xFF; // Like
red(), but faster
      float currG = green(currColor); //(currColor >> 8) & 0xFF;
      float currB = blue(currColor); //currColor & 0xFF;

      // Extract red, green, and blue components from previous pixel
      float prevR = red(prevColor); //(prevColor >> 16) & 0xFF;
      float prevG = green(prevColor); //(prevColor >> 8) & 0xFF;
      float prevB = blue(prevColor); //prevColor & 0xFF;

      // Compute the difference of the red, green, and blue values
      float diffR = abs(currR - prevR);
      float diffG = abs(currG - prevG);
      float diffB = abs(currB - prevB);
```

```
/**
 * Frame Differencing
 * by Golan Levin.
 *
 * Quantify the amount of movement
in the video frame using frame-
differencing.
 */
```

```
      // Add these differences to the running tally
      //  movementSum += diffR + diffG + diffB;
      // Render the difference image to the screen

      if (diffR + diffG + diffB>8) {
      pixels[i] = color(0, 0, 0);
      } else {
         pixels[i] = color(255, 0, 0);
      }

      // The following line is much faster, but more confusing to read
      //pixels[i] = 0xff000000 | (diffR << 16) | (diffG << 8) | diffB;
      // Save the current color into the 'previous' buffer
      previousFrame[i] = currColor;
    }

    // To prevent flicker from frames that are all black (no movement),
    // only update the screen if the image has changed.
    // if (movementSum > 0) {
        updatePixels();
    // println(movementSum); // Print the total amount of movement to the console

  }
}
```

Color Tracking

CODE

```
import processing.video.*;

// Variable for capture device
Capture video;

// A variable for the color we are searching for.
color trackColor;

void setup() {
  size(320,240);
  video = new Capture(this,width,height,15);
  video.start();
  // Start off tracking for red
  trackColor = color(255,255,255);
  smooth();
}

void draw() {
  // Capture and display the video
  if (video.available()) {
    video.read();
  }
  video.loadPixels();
  image(video,0,0);

  // Before we begin searching, the "world record" for closest color is set to a high number that is easy for the first pixel to beat.
  float worldRecord = 500;

  // XY coordinate of closest color
  int closestX = 0;
  int closestY = 0;

  // Begin loop to walk through every pixel
  for (int x = 0; x < video.width; x ++ ) {
    for (int y = 0; y < video.height; y ++ ) {
      int loc = x + y*video.width;

      // What is current color
      color currentColor = video.pixels[loc];
      float r1 = red(currentColor);
      float g1 = green(currentColor);
      float b1 = blue(currentColor);
      float r2 = red(trackColor);
      float g2 = green(trackColor);
      float b2 = blue(trackColor);

      // Using euclidean distance to compare colors
      float d = dist(r1,g1,b1,r2,g2,b2);
      // We are using the dist( ) function to compare the current color with the color we are tracking.
```

```
// Learning Processing
// Daniel Shiffman
// http://www.learningprocessing.com

// Example 16-11: Simple color tracking
```

```
      // If current color is more similar to tracked color than
      // closest color, save current location and current difference
       if (d < worldRecord) {
         worldRecord = d;
         closestX = x;
         closestY = y;
       }
     }
   }

   // We only consider the color found if its color distance is less than 10.
   // This threshold of 10 is arbitrary and you can adjust this number depending on
how accurate you require the tracking to be.
   if (worldRecord < 10) {
     // Draw a circle at the tracked pixel
     fill(trackColor);
     strokeWeight(4.0);
     stroke(0);
     ellipse(closestX,closestY,16,16);
   }
}

void mousePressed() {
   // Save color where the mouse is clicked in trackColor variable
   int loc = mouseX + mouseY*video.width;
   trackColor = video.pixels[loc];
}
```