jromeem / M4RT1N1C0D3

★ Star 0 Fork 0

<> Code

Branch: master ▾ M4RT1N1C0D3 / / M4RT1N1C0D3.pde

Find file Copy path

jromeem M4RT1N1C0D3 0dde7e2 an hour ago

789 lines (688 sloc) 20.7 KB

Raw Reward History

```
// images and fonts
PImage img;
PFont font;

// glitchers and graphers
Glitcher glitch;
Grapher graph;

// a random string of characters
String randomCharacters = "{}|[]/:<>?.,);'~(*&^%$#@!-=_+12340!@#$";

// three x positions
float posx1;
float posx2;
float posx3;

// three y positions
float posy1;
float posy2;
float posy3;

void setup()
{
  // create the canvas
  size(2100, 1400);

  // load an image and a font
  img = loadImage("data/chris.jpg");

  // start a glitcher
  glitch = new Glitcher(img);
  glitch.shiftSegment(240, 285, -200);

  // create random x positions
  posx1 = random(0, width);
  posx2 = random(0, width);
  posx3 = random(0, width);

  // create random y positions
  posy1 = random(0, height);
  posy2 = random(0, height);
  posy3 = random(0, height);
}

void draw()
{
  background(0);
  //image(img, 0, 0, width, height);
  // do some glitches first
  glitch.shiftSegment(240, 285, 3);
  glitch.shiftSegment(415, 540, -3);
  glitch.shiftSegment(615, 640, 2);
  glitch.vshiftSegment(0, width/2, 2);
  glitch.vshiftSegment(width/2, width, -2);

  // start a grapher using the glitch as the image
  graph = new Grapher(glitch);
  color cc = color(255, 255, 255);

  // create a halftone based on the glitched image
  graph.halftone(10, cc);

  // display the background glitched image
  //image(glitch, 0, 0, width, height);
  // display the halftone

  image(graph, 0, 0, width, height);

  // display a smaller version of the background on top
  //image(glitch, 100, 100, 300, 500);

  // create a white unfilled rectangle
  //noFill();
  //stroke(255, 255, 255);
  //rect(120, 60, 300, 500);

  // create strings for text
  //String mm =  nf(millis()%1000, 3); // Values from 0 - 999
  //String d = nf(day(), 2);    // Values from 1 - 31
  //String mo = nf(month(), 2);  // Values from 1 - 12
  //String y = nf(year(), 4);   // 2003, 2004, 2005, etc.
  //String s = nf(second(), 2);  // Values from 0 - 59
  //String m = nf(minute(), 2);  // Values from 0 - 59
  //String h = nf(hour(), 2);    // Values from 0 - 23

  //// build the date string
  //String date = y+"-"+mo+"-"+d+" "+h+":"+m+":"+s+":"+mm;

  //// set the alignment and display the text
  //textAlign(CENTER, BOTTOM);
  //text(date, 0, 0, width, height*7/8);

  // create random letters drawn from the randomCharacters string
  //char randomLetter = randomCharacters.charAt(int(random(randomCharacters.length()-1)));
  //char randomLetter2 = randomCharacters.charAt(int(random(randomCharacters.length()-1)));
  //char randomLetter3 = randomCharacters.charAt(int(random(randomCharacters.length()-1)));
  //char randomLetter4 = randomCharacters.charAt(int(random(randomCharacters.length()-1)));
  //char randomLetter5 = randomCharacters.charAt(int(random(randomCharacters.length()-1)));
  //char randomLetter6 = randomCharacters.charAt(int(random(randomCharacters.length()-1)));

  //// set the alignment again
  //textAlign(CORNER, CORNER);

  //// display three different texts in three different positions
  //text("h"+randomLetter+"llo w"+randomLetter2+"rld", posx1, posy1);
  //text("he"+randomLetter3+"lo worl"+randomLetter4, posx2, posy2);
  //text(randomLetter5+"ello wor"+randomLetter6+"d", posx3, posy3);
}

class Glitcher extends PImage {
  PImage img;
  PImage imgImmutable;
  PGraphics pg;

  Glitcher(PImage i) {
    this.img = i;
    this.pg = createGraphics(this.img.width, this.img.height);

    // make a immutable copy
    PGraphics pg2 = createGraphics(this.img.width, this.img.height);
    pg2.beginDraw();
    pg2.image(this.img, 0, 0);
    this.imgImmutable = pg2.copy();
    pg2.endDraw();
  }

  Glitcher(Grapher g) {
    this.img = g.img;
    this.pg = createGraphics(this.img.width, this.img.height);

    // make a immutable copy
    PGraphics pg2 = createGraphics(this.img.width, this.img.height);
    pg2.beginDraw();
    pg2.image(this.img, 0, 0);
    this.imgImmutable = pg2.copy();
    pg2.endDraw();
  }

  // overriding method from PImage
  void resize(int w, int h) {
    this.pg = createGraphics(w, h);
    this.pg.beginDraw();
    this.img.resize(w, h);
    this.pg.image(this.img, 0, 0);
    this.img = this.pg.copy();
    this.pg.endDraw();
  }

  // defer PImage methods -- get PImage first then use PImage methods
  PImage getPImage() {
    return this.img;
  }

  Glitcher shift() {
    return this.shift(int(random(this.img.width)));
  }

  Glitcher shift(float shiftAmount) {
    return this.shiftSegment(0, this.img.height, int(shiftAmount));
  }

  Glitcher shift(int shiftAmount) {
    return this.shiftSegment(0, this.img.height, shiftAmount);
  }

  Glitcher shiftSegment() {
    int y1 = int(random(this.img.height));
    int y2 = int(random(this.img.height));
    int randShiftAmount = int(random(this.img.width));
    int minY = min(y1, y2);
    int maxY = max(y1, y2);
    return this.shiftSegment(minY, maxY, randShiftAmount);
  }

  Glitcher shiftSegment(float y1, float y2, float shiftAmount) {
    return this.shiftSegment(int(y1), int(y2), int(shiftAmount));
  }

  Glitcher shiftSegment(int y1, int y2, int shiftAmount) {
    y1 = constrain(y1, 0, this.img.height);
    y2 = constrain (y2, 0, this.img.height);
    shiftAmount = int(shiftAmount);
    this.pg.beginDraw();
    this.pg.image(this.img, 0, 0);
    this.pg.loadPixels();
    for (int i = y1*this.img.width; i < y2*this.img.width; i+=this.img.width) {
      color[] pixelsDest = new color[this.img.width];
      arrayCopy(this.pg.pixels, i, pixelsDest, 0, this.img.width);
      for (int w = 0; w < this.img.width; w++) {
        if (shiftAmount < 0) {
          int absShiftAmount = abs(shiftAmount);
          this.pg.pixels[i+w] = pixelsDest[(w+absShiftAmount)%this.img.width];
        } else {
          this.pg.pixels[i+((w+shiftAmount)%this.img.width)] = pixelsDest[w];
        }
      }
    }
    this.pg.updatePixels();
    this.img = this.pg.copy();
    this.pg.endDraw();
    return this;
  }

  Glitcher vshift() {
    return this.vshiftSegment(0, this.img.width, int(random(this.img.height)));
  }

  Glitcher vshift(float shiftAmount) {
    return this.vshiftSegment(0, this.img.width, int(shiftAmount));
  }

  Glitcher vshift(int shiftAmount) {
    return this.vshiftSegment(0, this.img.width, shiftAmount);
  }

  Glitcher vshiftSegment() {
    int x1 = int(random(this.img.width));
    int x2 = int(random(this.img.width));
    int randShiftAmount = int(random(this.img.height));
    int minX = min(x1, x2);
    int maxX = max(x1, x2);
    return this.vshiftSegment(minX, maxX, randShiftAmount);
  }

  Glitcher vshiftSegment(float x1, float x2, float shiftAmount) {
    return this.vshiftSegment(int(x1), int(x2), int(shiftAmount));
  }

  Glitcher vshiftSegment(int x1, int x2, int shiftAmount) {
    x1 = constrain(x1, 0, this.img.width);
    x2 = constrain (x2, 0, this.img.width);
    this.pg.beginDraw();
    this.pg.image(this.img, 0, 0);
    this.pg.loadPixels();
    for (int i = x1; i < x2; i++) {
      color[] pixelsDest = new color[this.img.height];
      for (int c = 0; c < this.img.height; c++) {
        int index = c * this.img.width + i;
        pixelsDest[c] = this.pg.pixels[index];
      }
      for (int j = 0; j < this.img.height; j++) {
        if (shiftAmount >= 0) {
          int shifter = (j+shiftAmount)%this.img.height;
          int sindex = shifter * this.img.width + i;
          this.pg.pixels[sindex] = pixelsDest[j];
        } else {
          int absShiftAmount = abs(shiftAmount);
          this.pg.pixels[j*this.img.width+i] = pixelsDest[((j+absShiftAmount)%this.img.height)];
        }
      }
    }
    this.pg.updatePixels();
    this.img = this.pg.copy();
    this.pg.endDraw();
    return this;
  }

  PImage pixelate(float pixelAmount) {
    return this.pixelate(0, 0, this.img.width, this.img.height, int(pixelAmount));
  }

  PImage pixelate(float xOffset, float yOffset, float w, float h, float pixelAmount) {
   return this.pixelate(int(xOffset), int(yOffset), int(w), int(h), int(pixelAmount));
  }

  PImage pixelate(int xOffset, int yOffset, int w, int h, int pixelAmount) {
    this.pg.beginDraw();
    this.pg.image(this.img, 0, 0);
    if (pixelAmount <= 0) {
      this.img = this.pg.copy();
      return this;
    }
    this.pg.loadPixels();
    for (int j = 0; j < this.img.width*h; j+=this.img.width*pixelAmount) {
      for (int i = 0; i < w; i+=pixelAmount) {
        // find pixel average
        int collectionTotal = int(pixelAmount*pixelAmount);
        float totalR, totalG, totalB;
        totalR = totalG = totalB = 0;
        for (int m = 0; m < pixelAmount*this.img.width; m+=this.img.width) {
          for (int p = 0; p < pixelAmount; p++) {
            color c = this.pg.pixels[min(p+m + i+j, this.pg.pixels.length-1)];
            totalR += red(c);
            totalG += green(c);
            totalB += blue(c);
          }
        }
        float thisR = totalR / collectionTotal;
        float thisG = totalG / collectionTotal;
        float thisB = totalB / collectionTotal;
        color newColor = color(thisR, thisG, thisB);
        this.pg.fill(newColor);
        this.pg.noStroke();

        // draw new pixel rect
        int px = (i+j)%this.img.width + xOffset;
        int py = j/this.img.width + yOffset;
        this.pg.rect(px, py, pixelAmount, pixelAmount);
      }
    }
    this.img = this.pg.copy();
    this.pg.endDraw();
    return this;
  }

  PImage offset(int x, int y, int w, int h, int randAmount) {
    PGraphics pgg = createGraphics(this.img.width, this.img.height);
    pgg.beginDraw();
    pgg.image(this.img, 0, 0);
    int rAmount = int(random(-randAmount, randAmount));
    pgg.copy(this.imgImmutable,x,y,w,h,x+rAmount,y+rAmount,w,h);
    this.img = pgg.copy();
    pgg.endDraw();

    this.pg = pgg;
    this.pg.beginDraw();
    this.pg.image(this.img, 0, 0);
    this.pg.endDraw();

    return this;
  }

  PImage stretch() {
    return this.img;
  }

  PImage spin() {
    return this.img;
  }

  PImage channel_shift() {
    return this.img;
  }

  PImage data_bend() {
    return this.img;
  }

  PImage pixelSort(int mode, float threshold) {
    PixelSort p = new PixelSort(this.img, mode, threshold);
    this.pg.beginDraw();
    this.pg.image(p.ps_draw(), 0, 0);
    this.pg.endDraw();
    return this;
  }
}

class PixelSort {
  /*
   ASDF Pixel Sort
   Kim Asendorf | 2010 | kimasendorf.com
   Sorting modes:
   0 = black
   1 = brightness
   2 = white
  */
  int mode = 1;
  float thresholdValue;

  int loops = 1;

  // threshold values to determine sorting start and end pixels
  float blackValue = -16000000;
  float brightnessValue = 60;
  float whiteValue = -13000000;

  int row = 0;
  int column = 0;

  boolean saved = false;
  boolean columnDone = false;
  boolean rowDone = false;

  PImage _img;
  PGraphics pgg;
  PImage imgImmutable;

  PixelSort(PImage psImg, int mode, float thresholdValue) {
    this._img = psImg;
    this.mode = mode;

    // map the values based on the mode
    if (this.mode == 0) {
      this.thresholdValue = map(thresholdValue, 0, 1000, 0, -16000000);
    } else if (this.mode == 1) {
      this.thresholdValue = map(thresholdValue, 0, 1000, 255, 0);
    } else if (this.mode == 2) {
      this.thresholdValue = map(thresholdValue, 0, 1000, -16000000, 0);
    } else {
      this.mode = 0;
      this.thresholdValue = 1000;
    }

    // set the values to the threshold value
    this.blackValue = this.thresholdValue;
    this.brightnessValue = this.thresholdValue;
    this.whiteValue = this.thresholdValue;

    this.pgg = createGraphics(_img.width, _img.height);
        // make a immutable copy
    PGraphics pg2 = createGraphics(this._img.width, this._img.height);
    pg2.beginDraw();
    pg2.image(this._img, 0, 0);
    this.imgImmutable = pg2.copy();
    pg2.endDraw();
  }

  PGraphics ps_draw() {
    // loop through columns
    pgg.beginDraw();
    while(column < _img.width-1) {
      _img.loadPixels();
      sortColumn();
      column++;
      _img.updatePixels();
    }
    // loop through rows
    while(row < _img.height-1) {
      _img.loadPixels();
      sortRow();
      row++;
      _img.updatePixels();
    }
    // load updated image onto surface and scale to fit the display width,height
    pgg.image(_img, 0, 0, width, height);
    if (column >= _img.width-1) {
      columnDone = true;
    }
    if (row >= _img.height-1) {
      rowDone = true;
    }
    if (columnDone && rowDone) {
      column = 0;
      row = 0;
      columnDone = false;
      rowDone = false;
    }
    _img = imgImmutable;

    pgg.endDraw();
    return pgg;
  }

  void sortRow() {
    // current row
    int y = row;
    // where to start sorting
    int x = 0;
    // where to stop sorting
    int xend = 0;
    while(xend < _img.width-1) {
      switch(mode) {
        case 0:
          x = getFirstNotBlackX(x, y);
          xend = getNextBlackX(x, y);
          break;
        case 1:
          x = getFirstBrightX(x, y);
          xend = getNextDarkX(x, y);
          break;
        case 2:
          x = getFirstNotWhiteX(x, y);
          xend = getNextWhiteX(x, y);
          break;
        default:
          break;
      }
      if(x < 0) break;
      int sortLength = xend-x;
      color[] unsorted = new color[sortLength];
      color[] sorted = new color[sortLength];
      for(int i=0; i<sortLength; i++) {
        unsorted[i] = _img.pixels[x + i + y * _img.width];
      }
      sorted = sort(unsorted);
      for(int i=0; i<sortLength; i++) {
        _img.pixels[x + i + y * _img.width] = sorted[i];
      }
      x = xend+1;
    }
  }

  void sortColumn() {
    // current column
    int x = column;
    // where to start sorting
    int y = 0;
    // where to stop sorting
    int yend = 0;
    while(yend < _img.height-1) {
      switch(mode) {
        case 0:
          y = getFirstNotBlackY(x, y);
          yend = getNextBlackY(x, y);
          break;
        case 1:
          y = getFirstBrightY(x, y);
          yend = getNextDarkY(x, y);
          break;
        case 2:
          y = getFirstNotWhiteY(x, y);
          yend = getNextWhiteY(x, y);
          break;
        default:
          break;
      }
      if(y < 0) break;
      int sortLength = yend-y;
      color[] unsorted = new color[sortLength];
      color[] sorted = new color[sortLength];
      for(int i=0; i<sortLength; i++) {
        unsorted[i] = _img.pixels[x + (y+i) * _img.width];
      }
      sorted = sort(unsorted);
      for(int i=0; i<sortLength; i++) {
        _img.pixels[x + (y+i) * _img.width] = sorted[i];
      }
      y = yend+1;
    }
  }

  // black x
  int getFirstNotBlackX(int x, int y) {
    while(_img.pixels[x + y * _img.width] < blackValue) {
      x++;
      if(x >= _img.width)
        return -1;
    }
    return x;
  }

  int getNextBlackX(int x, int y) {
    x++;
    while(_img.pixels[x + y * _img.width] > blackValue) {
      x++;
      if(x >= _img.width)
        return _img.width-1;
    }
    return x-1;
  }

  // brightness x
  int getFirstBrightX(int x, int y) {
    while(brightness(_img.pixels[x + y * _img.width]) < brightnessValue) {
      x++;
      if(x >= _img.width)
        return -1;
    }
    return x;
  }

  int getNextDarkX(int _x, int _y) {
    int x = _x+1;
    int y = _y;
    while(brightness(_img.pixels[x + y * _img.width]) > brightnessValue) {
      x++;
      if(x >= _img.width) return _img.width-1;
    }
    return x-1;
  }

  // white x
  int getFirstNotWhiteX(int x, int y) {
    while(_img.pixels[x + y * _img.width] > whiteValue) {
      x++;
      if(x >= _img.width)
        return -1;
    }
    return x;
  }

  int getNextWhiteX(int x, int y) {
    x++;
    while(_img.pixels[x + y * _img.width] < whiteValue) {
      x++;
      if(x >= _img.width)
        return _img.width-1;
    }
    return x-1;
  }

  // black y
  int getFirstNotBlackY(int x, int y) {
    if(y < _img.height) {
      while(_img.pixels[x + y * _img.width] < blackValue) {
        y++;
        if(y >= _img.height)
          return -1;
      }
    }
    return y;
  }

  int getNextBlackY(int x, int y) {
    y++;
    if(y < _img.height) {
      while(_img.pixels[x + y * _img.width] > blackValue) {
        y++;
        if(y >= _img.height)
          return _img.height-1;
      }
    }
    return y-1;
  }

  // brightness y
  int getFirstBrightY(int x, int y) {
    if(y < _img.height) {
      while(brightness(_img.pixels[x + y * _img.width]) < brightnessValue) {
        y++;
        if(y >= _img.height)
          return -1;
      }
    }
    return y;
  }

  int getNextDarkY(int x, int y) {
    y++;
    if(y < _img.height) {
      while(brightness(_img.pixels[x + y * _img.width]) > brightnessValue) {
        y++;
        if(y >= _img.height)
          return _img.height-1;
      }
    }
    return y-1;
  }

  // white y
  int getFirstNotWhiteY(int x, int y) {
    if(y < _img.height) {
      while(_img.pixels[x + y * _img.width] > whiteValue) {
        y++;
        if(y >= _img.height)
          return -1;
      }
    }
    return y;
  }

  int getNextWhiteY(int x, int y) {
    y++;
    if(y < _img.height) {
      while(_img.pixels[x + y * _img.width] < whiteValue) {
        y++;
        if(y >= _img.height)
          return _img.height-1;
      }
    }
    return y-1;
  }
}

// keyboard interactions
void keyPressed() {
  if (key == 'g' || key == 'G') {
    clear();
    setup();
  }
  if (key == ' ') {
    saveFrame("GLITCH-####.png");
  }
}

// overloading image() methods
void image(Glitcher g, float a, float b) {
  image(g.img, a, b);
}

void image(Glitcher g, float a, float b, float c, float d) {
  image(g.img, a, b, c, d);
}

class Grapher extends PImage {
  PImage img;
  PImage imgImmutable;
  PGraphics pg;

  Grapher(PImage i) {
    this.img = i;
    this.pg = createGraphics(this.img.width, this.img.height);

    // make a immutable copy
    PGraphics pg2 = createGraphics(this.img.width, this.img.height);
    pg2.beginDraw();
    pg2.image(this.img, 0, 0);
    this.imgImmutable = pg2.copy();
    pg2.endDraw();
  }

  Grapher(Glitcher g) {
    this.img = g.img;
    this.pg = createGraphics(this.img.width, this.img.height);

    // make a immutable copy
    PGraphics pg2 = createGraphics(this.img.width, this.img.height);
    pg2.beginDraw();
    pg2.image(this.img, 0, 0);
    this.imgImmutable = pg2.copy();
    pg2.endDraw();
  }

  Grapher drawGrid(float stepSize, color c) {
    this.drawRows(stepSize, c);
    this.drawColumns(stepSize, c);
    return this;
  }

  Grapher drawRows(float rowWidth, color c) {
    this.pg.beginDraw();
    this.pg.stroke(c);
    for (int i = 0; i < this.pg.height; i+=rowWidth) {
      this.pg.line(0, i, this.pg.width, i);
    }
    this.pg.endDraw();
    return this;
  }

  Grapher drawColumns(float columnWidth, color c) {
    this.pg.beginDraw();
    this.pg.stroke(c);
    for (int i = 0; i < this.pg.width; i+=columnWidth) {
      this.pg.line(i, 0, i, this.pg.height);
    }
    this.pg.endDraw();
    return this;
  }

  Grapher halftone(float stepSize, color cc) {
    PGraphics imgpg = createGraphics(this.img.width, this.img.height);
    imgpg.beginDraw();
    imgpg.image(this.img, 0, 0);
    imgpg.loadPixels();
    this.pg.beginDraw();
    this.pg.noStroke();
    this.pg.fill(cc);
    float offset = stepSize;
    for (int i = 0; i < this.img.width*this.img.height; i+=this.img.width*(offset*1.5)) {
      for (int j = 0; j < this.img.width; j+=offset) {
        float brightVal = brightness(imgpg.pixels[i+j]);
        float ellipseSize = map(brightVal, 0, 255, 0, offset);
        float xLoc = (i+j) % this.img.width;
        float yLoc = i / this.img.width;
        this.pg.fill(red(imgpg.pixels[i+j]), green(imgpg.pixels[i+j]), blue(imgpg.pixels[i+j]));
        this.pg.ellipse(xLoc, yLoc, ellipseSize, ellipseSize);
      }
    }

    this.pg.endDraw();
    imgpg.endDraw();

    return this;
  }

}

// overloading image() methods
void image(Grapher g, float a, float b) {
  image(g.pg, a, b);
}

void image(Grapher g, float a, float b, float c, float d) {
  image(g.pg, a, b, c, d);
}

class Typer {
  PFont font;

  Typer(String s, int i) {
    this.font = createFont(s, i);
    textFont(font);
  }

  // overloading text methods
  void typeText(char c, float  x, float y) { text(c,x,y); }
  void typeText(char c, float x, float y, float z) { text(c,x,y,z); }
  void typeText(String str, float x, float y) { text(str,x,y); }
  void typeText(char[] chars, int start, int stop, float x, float y) { text(chars,start,stop,x,y); }
  void typeText(String str, float x, float y, float z) { text(str,x,y,z); }
  void typeText(char[] chars, int start, int stop, float x, float y, float z) { text(chars,start,stop,x,y,z); }
  void typeText(String str, float x1, float y1, float x2, float y2) { text(str,x1,y1,x2,y2); }
  void typeText(float num, float x, float y) { text(num,x,y); }
  void typeText(int num, float x, float y) { text(num,x,y); }
  void typeText(float num, float x, float y, float z) { text(num,x,y,z); }
  void typeText(int num, float x, float y, float z) { text(num,x,y,z); }
}
```