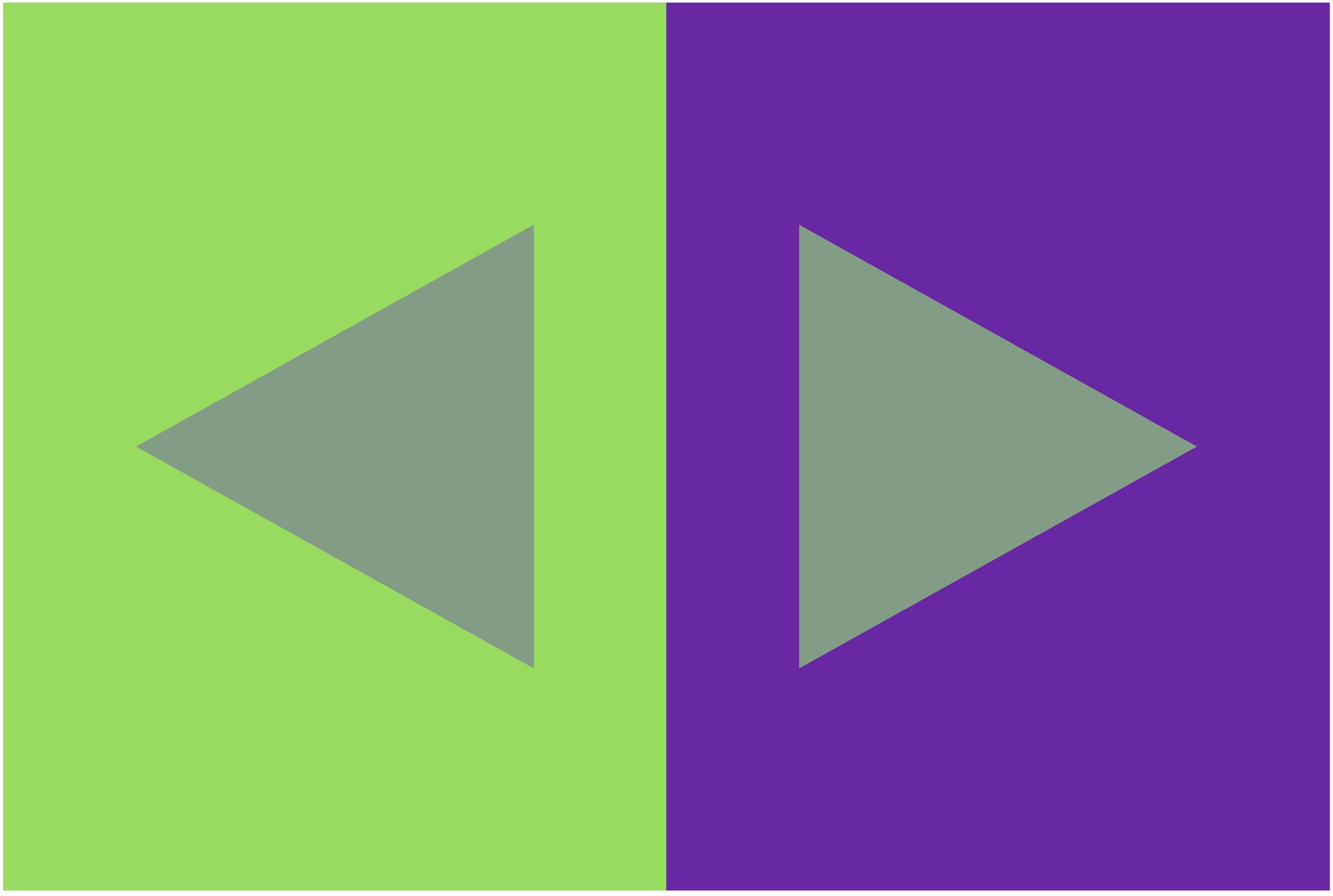
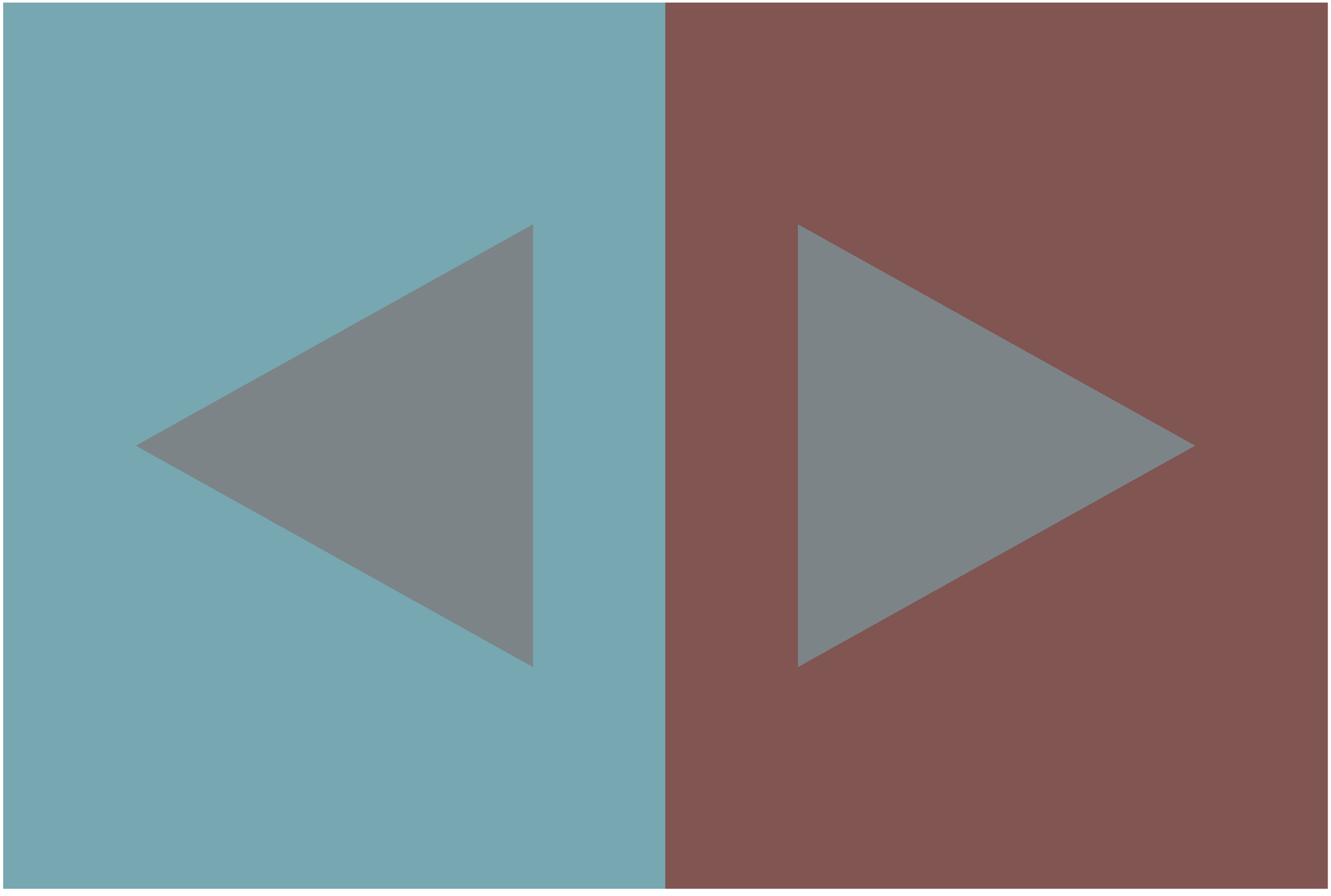
THE COLORIST COOKBOOK

"I TRY TO APPLY COLORS LIKE WORDS THAT SHAPE POEMS, LIKE NOTES THAT SHAPE MUSIC."

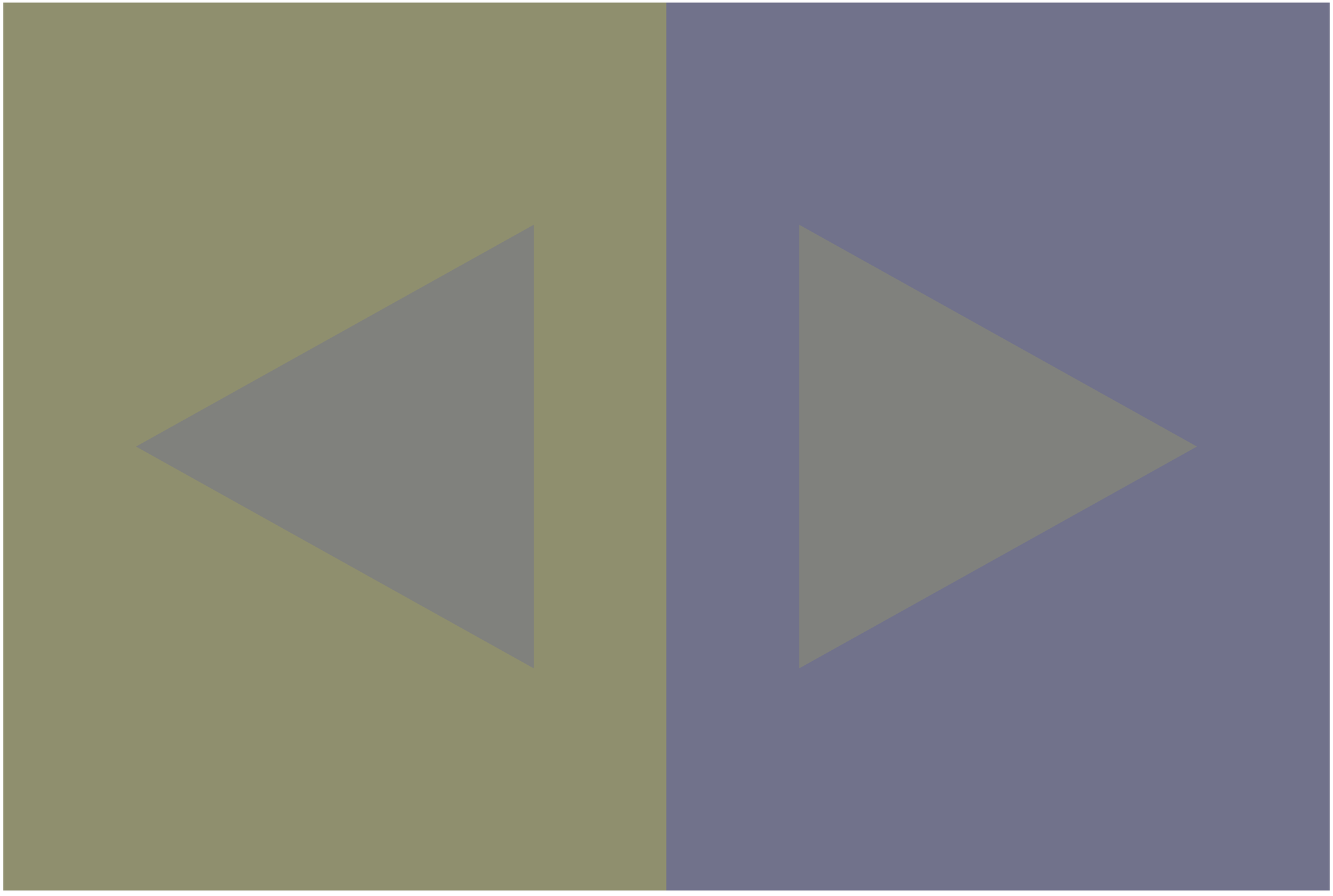
0xFF99DA61 0xFF6827A3 0xFF829C86



0xFF77A7B0 0xFF815551 0xFF7C8488



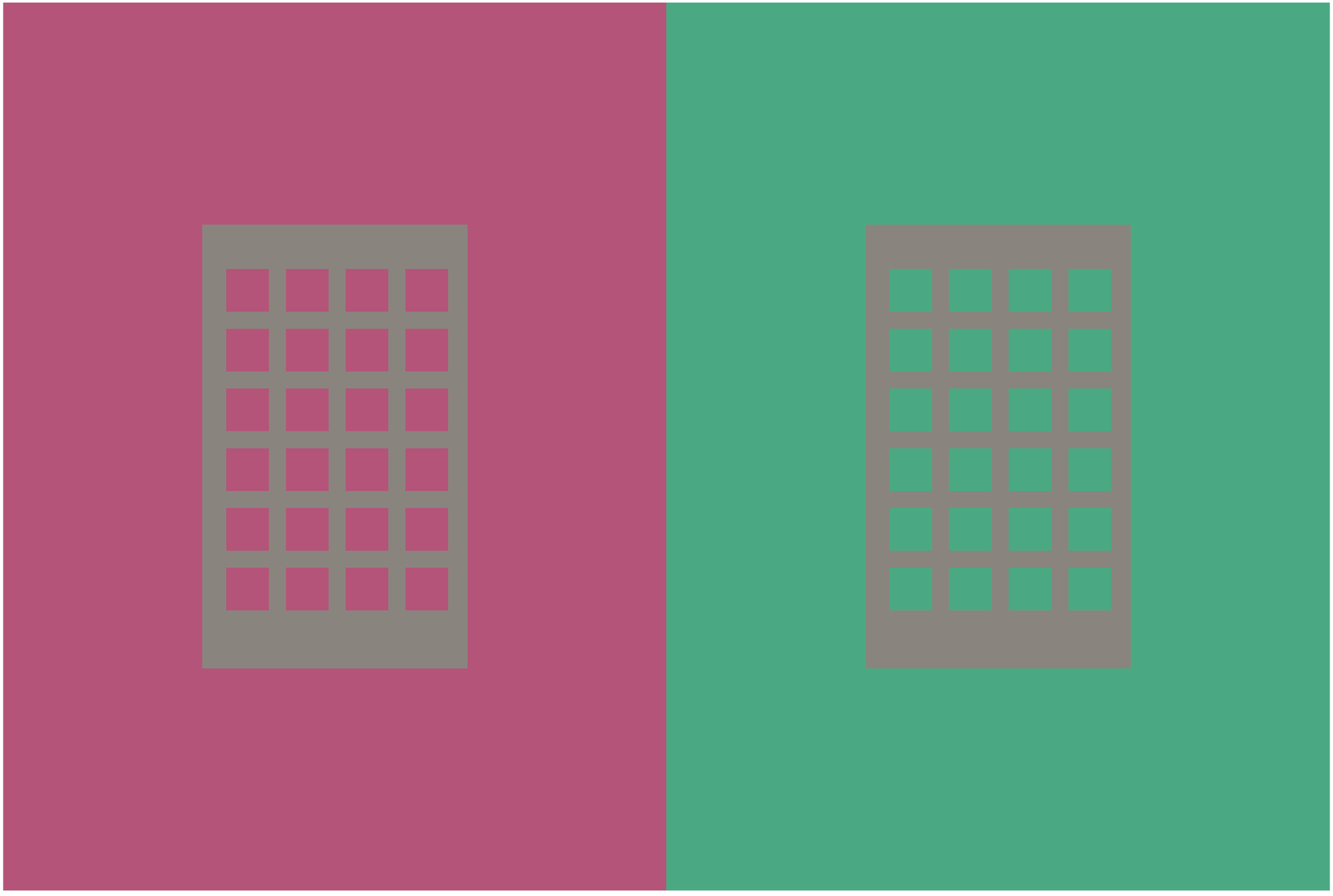
0xFF8F8F6E 0xFF71728B 0xFF80817D



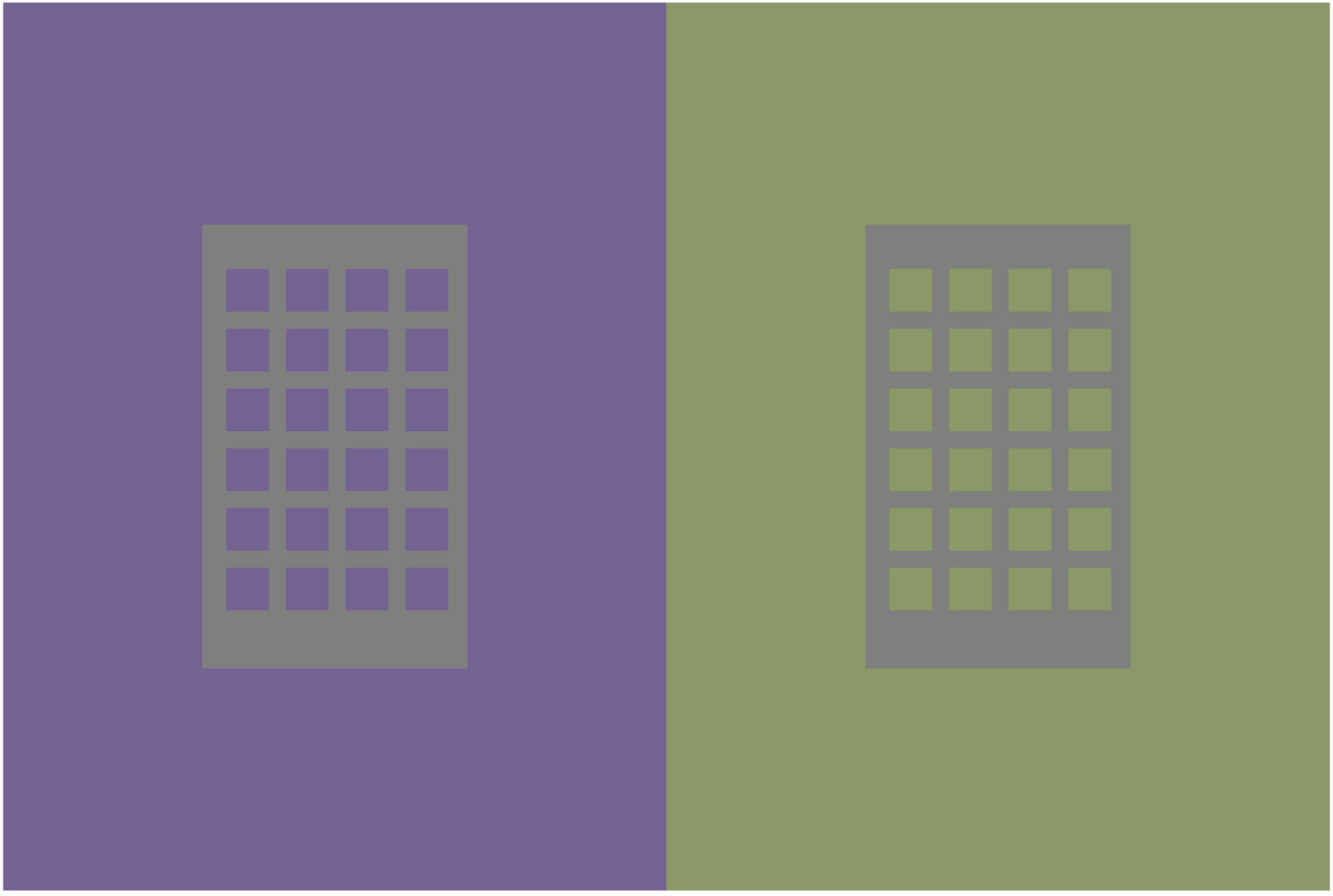
```
// recipe for simple simultaneous contrast

// prepare the first color
float r_col1 = random(2,84) + random(2,84) + random(2,84);
float g_col1 = random(2,84) + random(2,84) + random(2,84);
float b_col1 = random(2,84) + random(2,84) + random(2,84);

color col1 = color(r_col1, g_col1, b_col1);

// prepare the color opposite to the first color
// season with a touch of randomness
color col2 = color(255 - r_col1 + random(-7,7),
                   255 - g_col1 + random(-7,7),
                   255 - b_col1 + random(-7,7));

// evenly mix the first two colors to create
// the 'middle' color
// (recommended) season with randomness
color mid = color((r_col1+red(col2))/2f + random(-15,15),
                  (g_col1+green(col2))/2f + random(-15,15),
                  (b_col1+blue(col2))/2f + random(-15,15)) ;

// pre-translate the transformation matrix
// to the size of your margins
pushMatrix();
translate(margin, margin);

rectMode(CORNER);

// arrange opposing colors beside each other
```

```
fill(col1);
stroke(col1);
rect(0,0,pgwidth/2f,pgheight);
fill(col2);
stroke(col2);
rect(pgwidth/2f,0,pgwidth/2f,pgheight);

// top with the middle color
fill(mid);
noStroke();

triangle(pgwidth*0.1,pgheight/2f,pgwidth*0.4,pgheight/4f, pgwidth*0.4,pgheight*0.75);
triangle(pgwidth*0.9,pgheight/2f,pgwidth*0.6,pgheight/4f, pgwidth*0.6,pgheight*0.75);

popMatrix();
```

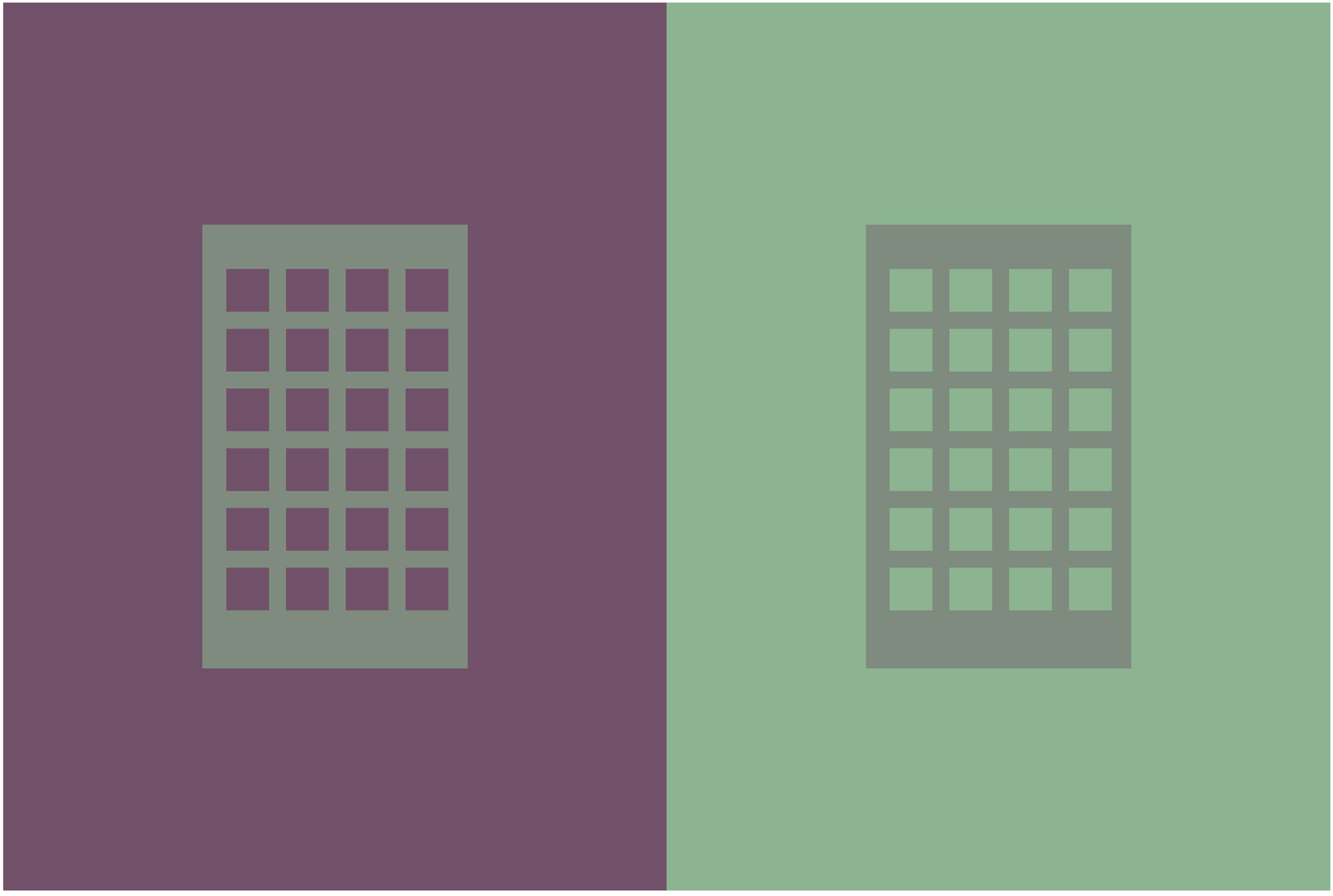
0xFFB45479 0xFF4AA983 0xFF89857E



0xFF736393 0xFF8A9966 0xFF7F807E



0xFF715169 0xFF8CB491 0xFF7F8B7E

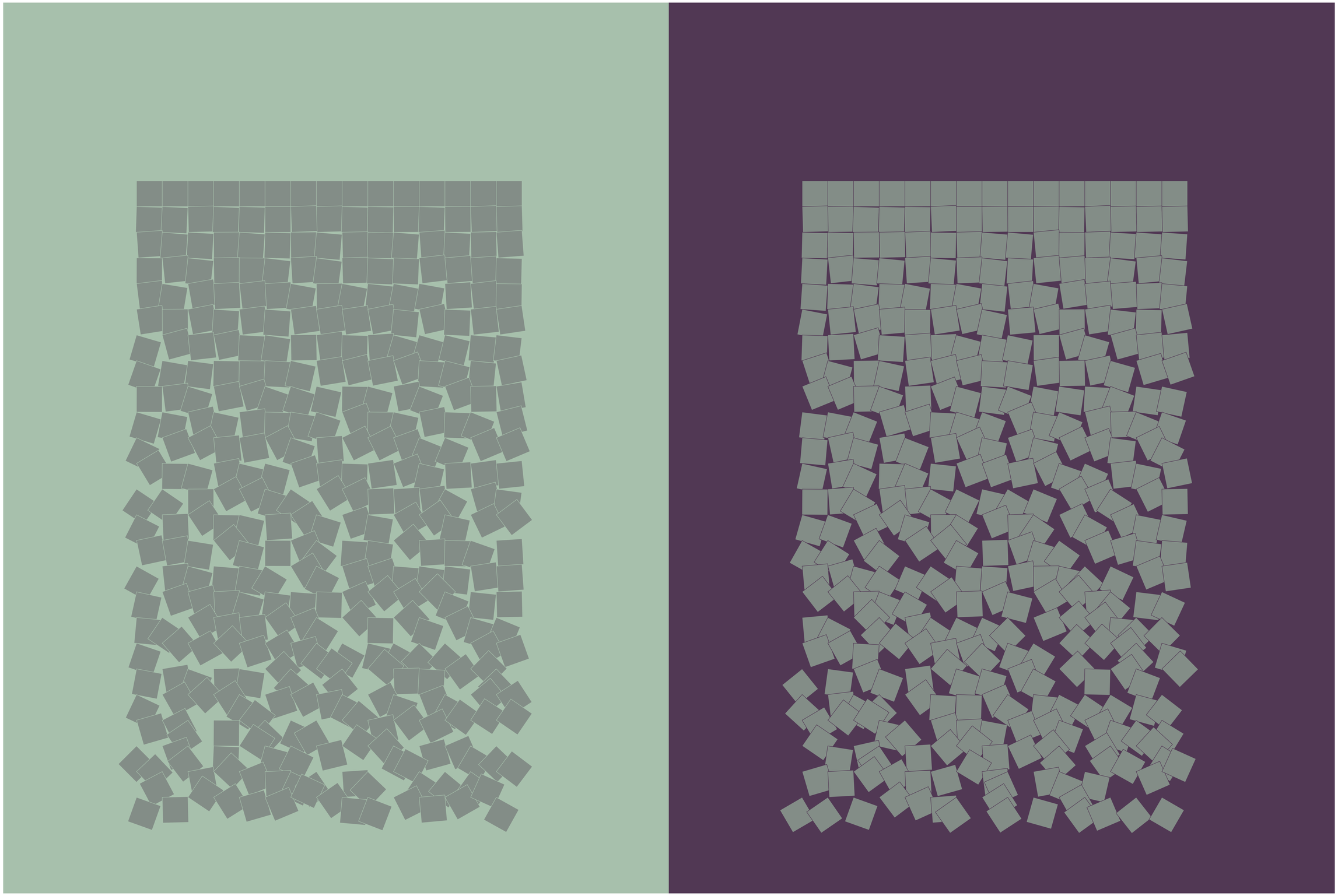


```
// recipe for holed simultaneous contrast

// prepare the first color
float r_col1 = random(2,84) + random(2,84) + random(2,84);
float g_col1 = random(2,84) + random(2,84) + random(2,84);
float b_col1 = random(2,84) + random(2,84) + random(2,84);

color col1 = color(r_col1, g_col1, b_col1);

// prepare the 'opposite' color to the first color
// season with a touch of randomness
color col2 = color(255 - r_col1 + random(-7,7),
                   255 - g_col1 + random(-7,7),
                   255 - b_col1 + random(-7,7));

  // evenly mix the first two colors to create
  // the 'middle' color
  // (recommended) season with randomness
  color mid = color((r_col1+red(col2))/2f + random(-15,15),
                    (g_col1+green(col2))/2f + random(-15,15),
                    (b_col1+blue(col2))/2f + random(-15,15)) ;

  // pre-translate the transformation matrix
  // to the size of your margins
  pushMatrix();
  translate(margin, margin);

  rectMode(CORNER);

  // arrange opposing colors beside each other
```

```
fill(col1);
stroke(col1);
rect(0,0,pgwidth/2f,pgheight);
fill(col2);
stroke(col2);
rect(pgwidth/2f,0,pgwidth/2f,pgheight);

// top with the middle color
rectMode(CENTER);
fill(mid);
noStroke();
rect((pgwidth)/4f,pgheight/2f,pgwidth/5f,pgheight/2f);
rect((pgwidth*3f)/4f,pgheight/2f,pgwidth/5f,pgheight/2f);

// slice holes in the middle for a more dramatic effect
// waffle aesthetic
rectMode(CORNER);
fill(col1);
for(int rows = 0; rows < 6; rows++) {
  for(int cols = 0; cols < 4; cols++) {
    rect(pgwidth * 0.168 + 140*cols, pgheight * 0.3 + 140*rows,100,100);
  }
}

fill(col2);
float rand3 = random(-350,350);
float rand4 = random(-250,250);
pushMatrix();
```

```
  for(int cols = 0; cols < 4; cols++) {
    rect(pgwidth * 0.668 + 140*cols, pgheight * 0.3 + 140*rows,100,100);
  }
}
popMatrix();

popMatrix();
```

0xFFA7C0AC 0xFF513854 0xFF838D87

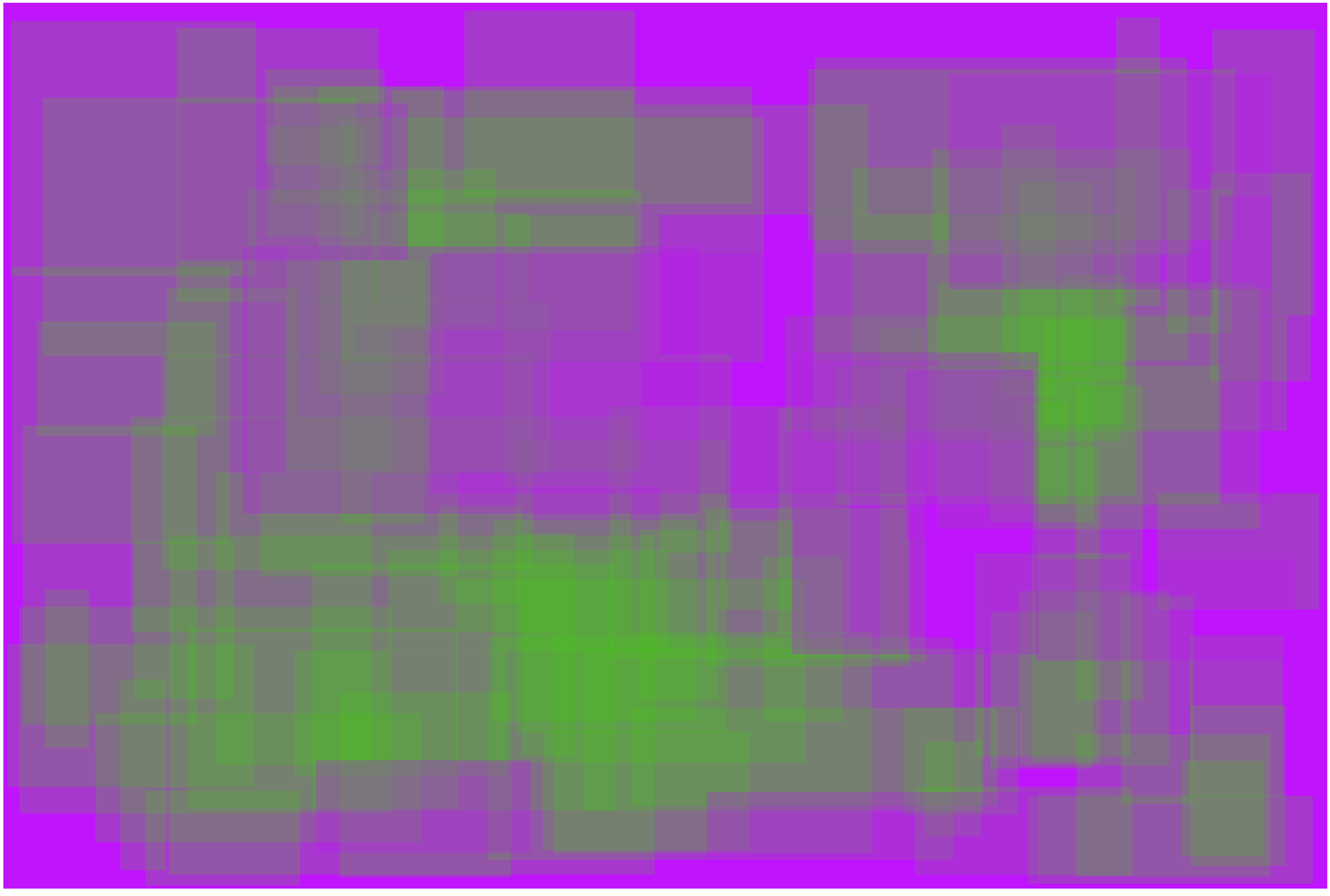


```
// recipe for schotter simultaneous contrast

// prepare the first color
float r_col1 = random(2,84) + random(2,84) + random(2,84);
float g_col1 = random(2,84) + random(2,84) + random(2,84);
float b_col1 = random(2,84) + random(2,84) + random(2,84);

color col1 = color(r_col1, g_col1, b_col1);

// prepare the 'opposite' color to the first color
// season with a touch of randomness
color col2 = color(255 - r_col1 + random(-7,7),
                   255 - g_col1 + random(-7,7),
                   255 - b_col1 + random(-7,7));

  // evenly mix the first two colors to create
  // the 'middle' color
  // (recommended) season with randomness
  color mid = color((r_col1+red(col2))/2f + random(-20,20),
                    (g_col1+green(col2))/2f + random(-20,20),
                    (b_col1+blue(col2))/2f + random(-15,15)) ;

  // pre-translate the transformation matrix
  // to the size of your margins
  pushMatrix();
  translate(margin, margin);

  rectMode(CORNER);

  // arrange opposing colors beside each other
```

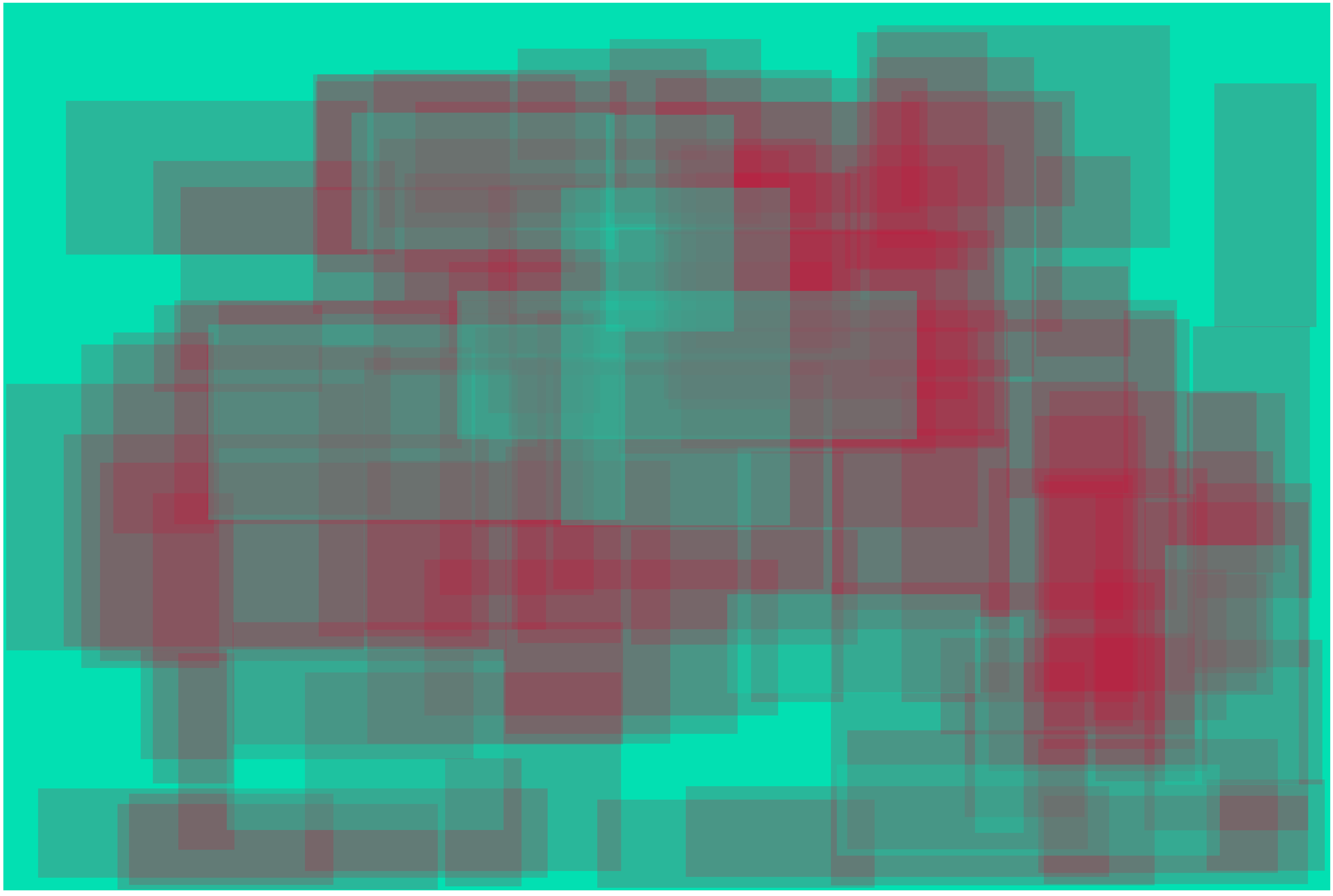
```
fill(col1);
stroke(col1);
rect(0,0,pgwidth/2f,pgheight);
fill(col2);
stroke(col2);
rect(pgwidth/2f,0,pgwidth/2f,pgheight);

// arrange squares in a Georg Nees fashion
rectMode(CORNER);
fill(mid);
stroke(col1);
for(int rows = 0; rows < 25; rows++) {
  for(int cols = 0; cols < 15; cols++) {
    pushMatrix();
    translate(pgwidth * 0.1 + 60*cols, pgheight * 0.2 + 60*rows);
    // let squares scatter more near the bottom
    float rotamnt = random(-rows*0.05,rows*0.05);
    rotate(rotamnt);
    rect(0,0,60,60);
    popMatrix();

  }
}
// repeat previous step
rectMode(CORNER);
fill(mid);
stroke(col2);
for(int rows = 0; rows < 25; rows++) {
  for(int cols = 0; cols < 15; cols++) {
```

```
    translate(pgwidth * 0.6 + 60*cols, pgheight * 0.2 + 60*rows);
    float rotamnt = random(-rows*0.05,rows*0.05);
    rotate(rotamnt);
    rect(0,0,60,60);
    popMatrix();

  }
}

popMatrix();
```

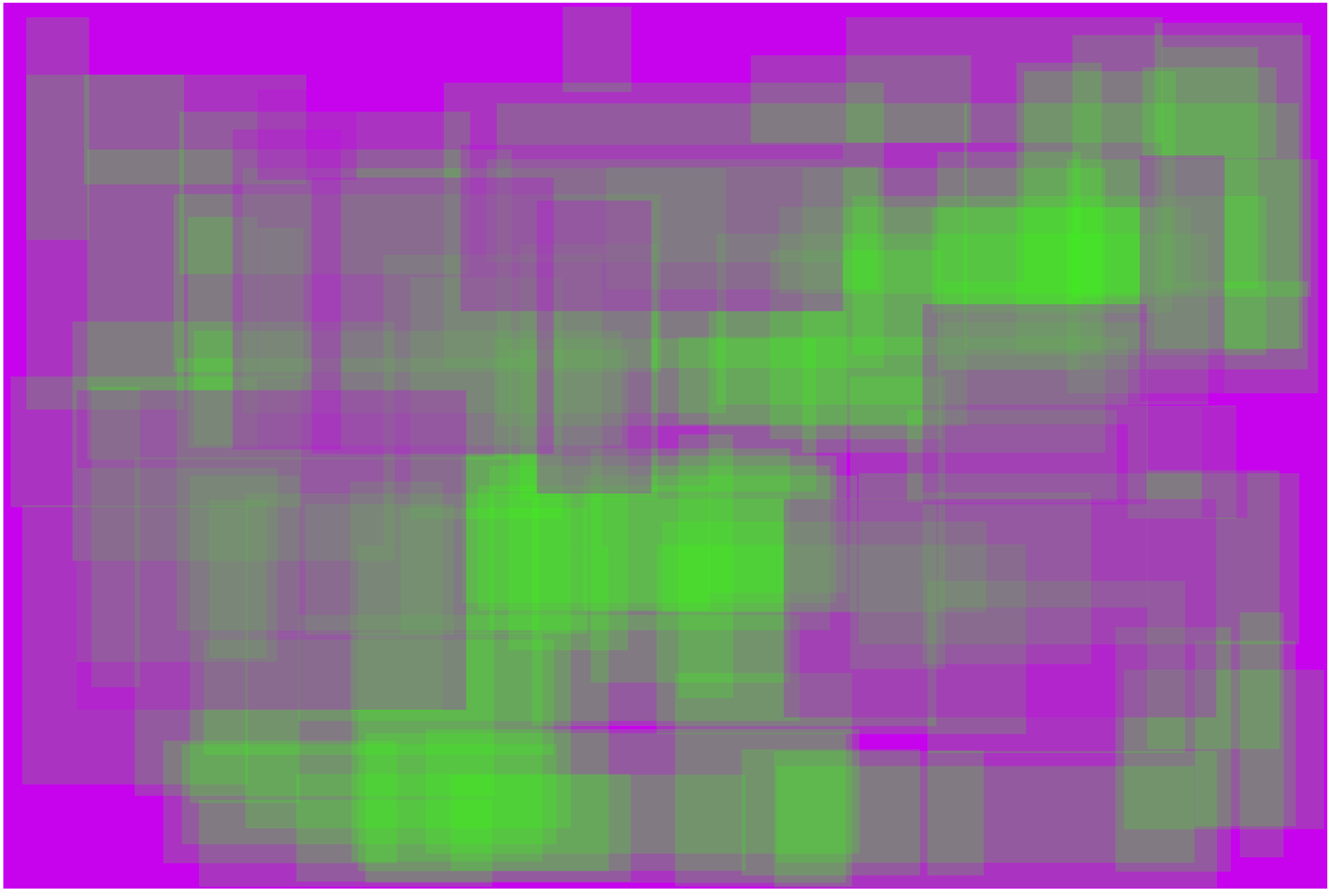
0xFF41D00F 0xFFC014FC



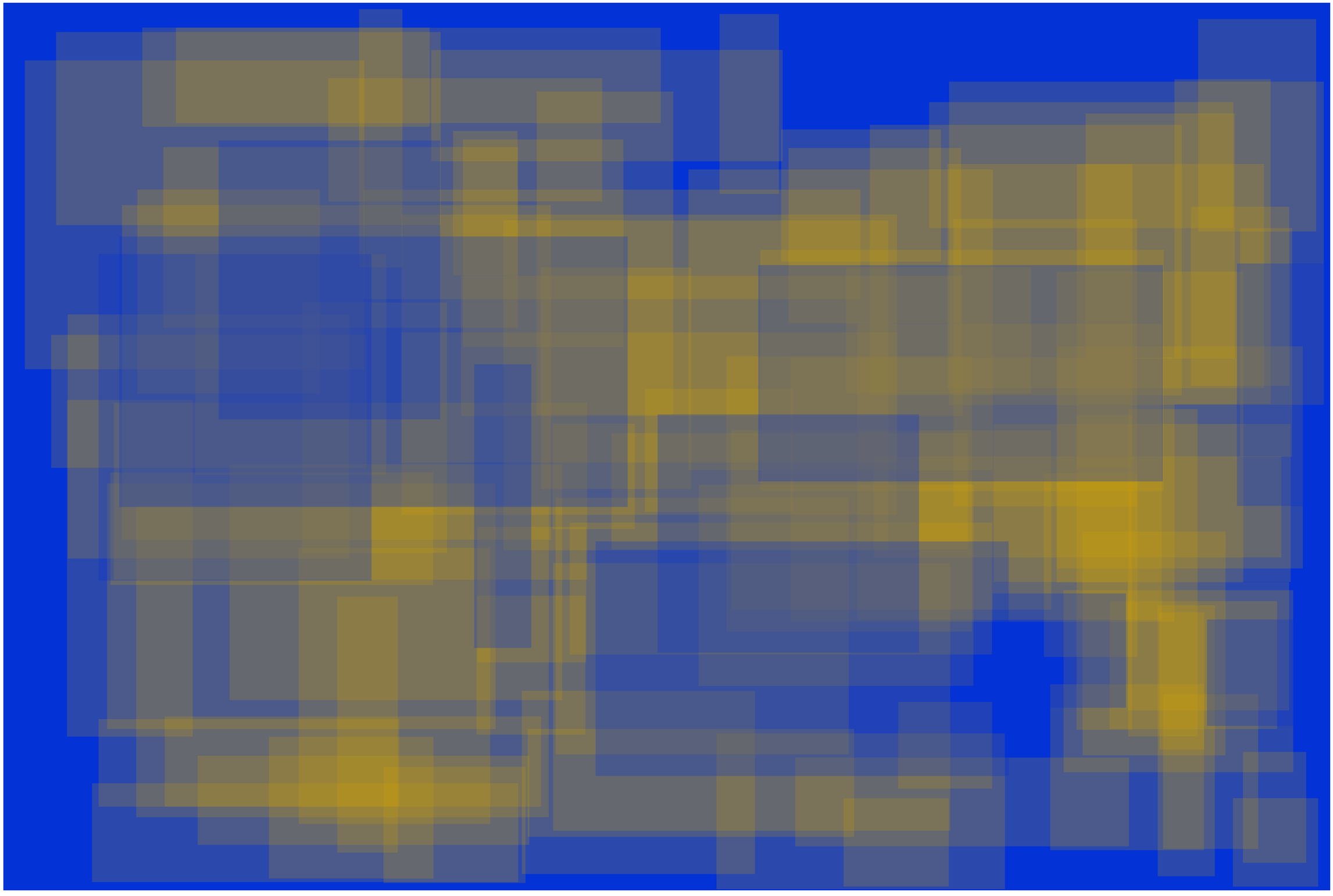
0xFFCD1138 0xFF02E0B2



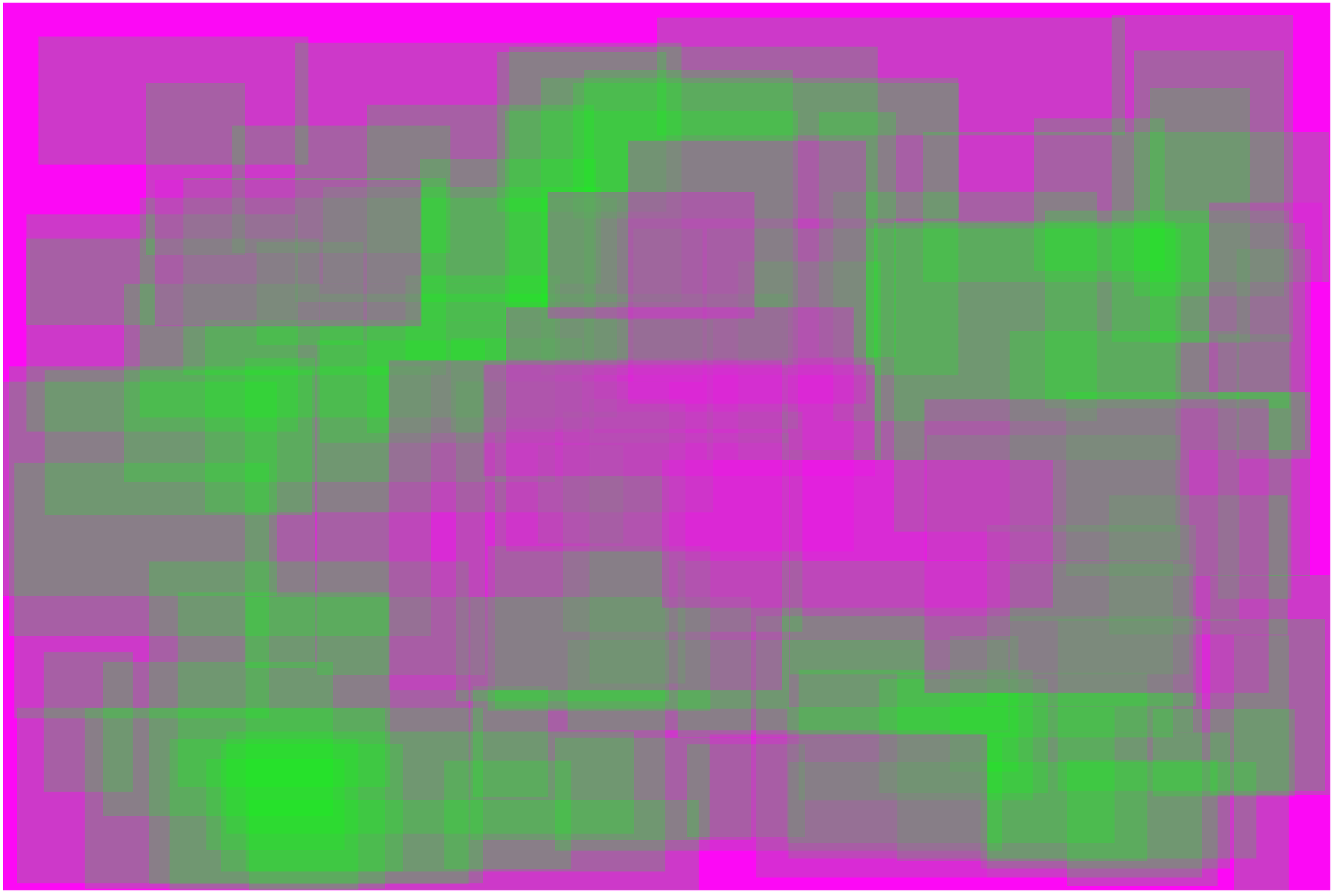
0xFF3AFE14 0xFFC702ED



0xFFD2A402 0xFF0332D6



0xFF0EFE15 0xFFFC09F5



```
// recipe for using bridging colors

// switch working color mode to HSB
// before preparing the foreground color
colorMode(HSB,360,100,100);
color col1 = color(random(0,360), random(90,100), random(80,100));

rectMode(CORNER);
pushMatrix();
translate(margin, margin);
noStroke();

// glaze with the background color, which is opposite from the foreground color
// add a touch of randomness
color bg = color((hue(col1)+180)%360, random(90,100), random(80,100));
fill(bg);
rect(0, 0, pgwidth,pgheight);

// randomly add 90 slices of the foreground color
for(int i = 0; i < 90; i++) {
  float posX = random(0, pgwidth-200);
  float posY = random(0, pgheight-200);

  float rectw = random(100, min(1200, pgwidth-posX));
  float recth = random(200, min(800, pgheight-posY));

  // the slices can overlap with each other,
  // but they must be thin
  // like prosciutto
```

```
    rect(posX, posY, rectw,recth);

  }

  // for balance, top with a few thin slices of the background color
  for(int i = 0; i < 10; i++) {
    float posX = random(0, pgwidth-200);
    float posY = random(0, pgheight-200);

    float rectw = random(100, min(1200, pgwidth-posX));
    float recth = random(200, min(800, pgheight-posY));

    fill(bg,70);
    rect(posX, posY, rectw,recth);
  }

  popMatrix();
```

0xFF494D5B

0xFF224258

0xFF0B818C

0xFF4D717B

0xFF8C8A88

0xFFBEA9AA

0xFFD2CCC9

0xFFFAF8BF

0xFF90595D

0xFF6B5760

0xFF188E9F

0xFF160E2A

0xFFBA3B3C

0xFFDD3A3A

0xFFD32F2F

0xFF8B383A

0xFFCCD6D0

0xFF9CBBB8

0xFF72A2A6

0xFFF2EEE9

0xFF4C8791

0xFE1B637C

0xFF2B494F

0xFF5F6E67

0xFFE78A5A

0xFF8C634C

0xFFAE6F50

0xFEC8623D

0xFF435854

0xFFD9D2CA

0xFEB87B7A
0xFF551E34

0xFF3C102E

0xFFD56465

0xFFAA3D45

0xFE230B2A
0xFF782533

0xFFE67071

0xFFDF393E

0xFFF1E8E4

artist:
Piotr Jablonski

0xFF896921
0xFFB1910C
0xFFC3A404
0xFFF6F007
0xFF4F3826
0xFF352316
0xFF100A06
0xFF9A8463
0xFF255B51

0xFF867B72

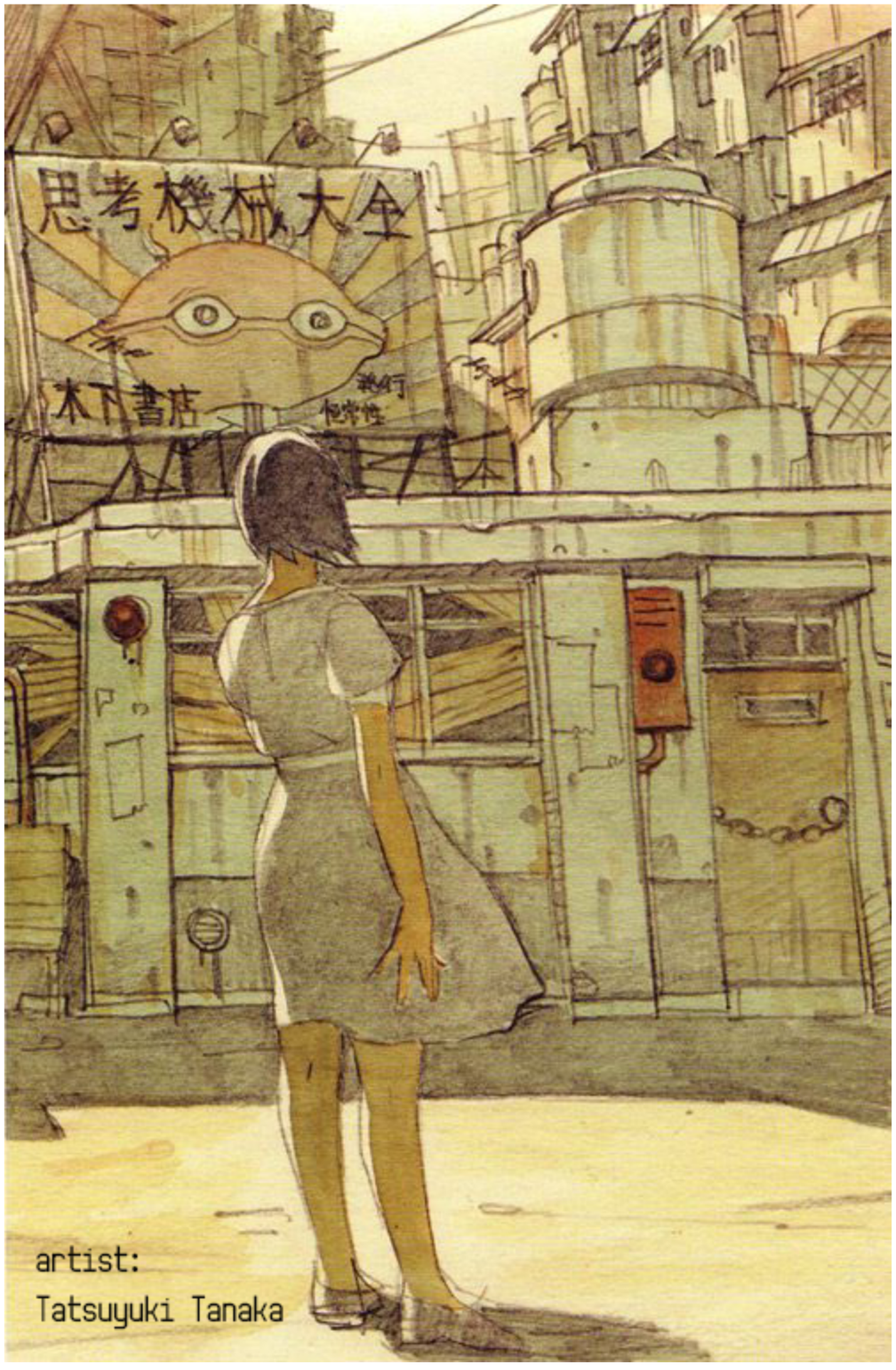
0xFFAC947E

0xFF655D50

0xFF473B32

0xFFC2A87D

0xFF75592E



0xFFBAA873

0xFF9D8657

0xFF775D3C

0xFF543A27

0xFF371F16

0xFFE9DAA9

0xFFFAF3CB

```
// prepare initial ingredients
ArrayList<Pixel> colors = new ArrayList<Pixel>();
float tolerance = 50.0;
int totalCount = 0;
float proptolerance = 0.005;
PImage[] images = new PImage[6];

public class Pixel {
  public ArrayList pixelgroup;
  public int count;
}

float colorDist(color c1, color c2) {
  float r = red(c1) - red(c2);
  float g = green(c1) - green(c2);
  float b = blue(c1) - blue(c2);

  return sqrt(sq(r) + sq(g) + sq(b));
}

void addPixel(color currpix) {
  Pixel p = new Pixel();
  // head of  the color group
  p.pixelgroup = new ArrayList();
  p.pixelgroup.add(currpix);
  p.count = 1;
  colors.add(p);
  totalCount++;
}
```

```
boolean notBlackorWhite(color col) {
  return ((col != 0.0) && (col != 255.0));
}

void palettePage(int imgindex) {
  totalCount = 0;

  // rinse palette before use
  for(int i = colors.size()-1; i >= 0 ; i--) {
     colors.remove(i);
  }

  int iwidth = images[imgindex].width;
  int iheight = images[imgindex].height;
  images[imgindex].resize(0, (int)pgheight);
  float palettepos = images[imgindex].width + 50;

// extract colors from image
// and add them to the palette
for(int i = 0; i < iheight; i++) {
  for(int j = 0; j < iwidth; j++) {
    color currpix = images[imgindex].get(i,j);

    if(notBlackorWhite(currpix)) {
      if(colors.size() == 0) {
        addPixel(currpix);
      }

      else {
```

```
boolean foundMatch = false;
int prevIdx = 0;
float minDist = 0.0;

// look through existing colors in the palette
for(idx = 0; idx < colors.size(); idx++) {
  float currDist = colorDist(currpix, (Integer) colors.get(idx).pixelgroup.get(0));
  // if the color is in the palette
  if(currDist < tolerance) {
        // increase the count for the palette color closest to
        // the current pixel
        if(foundMatch && (currDist < minDist)) {
          colors.get(idx).count++;
          colors.get(prevIdx).count--;
          colors.get(idx).pixelgroup.add(currpix);
          prevIdx = idx;
          minDist = currDist;
        }
        else if(!foundMatch) {
          colors.get(idx).count++;
          colors.get(idx).pixelgroup.add(currpix);
          totalCount++;
          foundMatch = true;
          prevIdx = idx;
          minDist = currDist;
        }
      }
}
```

```
        if(!foundMatch) {
          addPixel(currpix);

        }

      }
    }
  }
}

int netCount = totalCount;
for(int idx = 0; idx < colors.size(); idx++) {
  float prop = colors.get(idx).count / ((float)totalCount);

  if(prop < proptolerance) {
    netCount -= colors.get(idx).count;
  }
}
rectMode(CORNER);
float ypos = 0f;

pushMatrix();
// place image a little to the right of your margins
translate(margin + 700, margin);
image(images[imgindex], 0,0);

// place palette beside image
for(int i = 0; i < colors.size(); i++) {
  float prop = colors.get(i).count / ((float)netCount);
```

```
// if there is enough of the color in the image, display it
if(prop >= proptolerance) {
  float rheight = pgheight*prop;
  float totred = 0;
  float totgreen = 0;
  float totblue = 0;

  int groupsize = colors.get(i).pixelgroup.size();
  for(int j = 0; j < groupsize; j++) {
    Pixel thepixel = colors.get(i);
    color currcolor = (Integer) thepixel.pixelgroup.get(j);
    totred += red(currcolor);
    totgreen += green(currcolor);
    totblue += blue(currcolor);
  }
  color paletteColor = color(totred/groupsize, totgreen/groupsize,totblue/groupsize);

  stroke(paletteColor);
  fill(paletteColor);
  rect(palettepos, ypos, 300, rheight);

  // top it off with the hex value for the colors
  textFont(mainFont, 40);
  fill(paletteColor);
  text("0x"+hex(paletteColor), palettepos + 350, ypos+30);

  ypos += rheight;

}
```

```
    popMatrix();

}
```