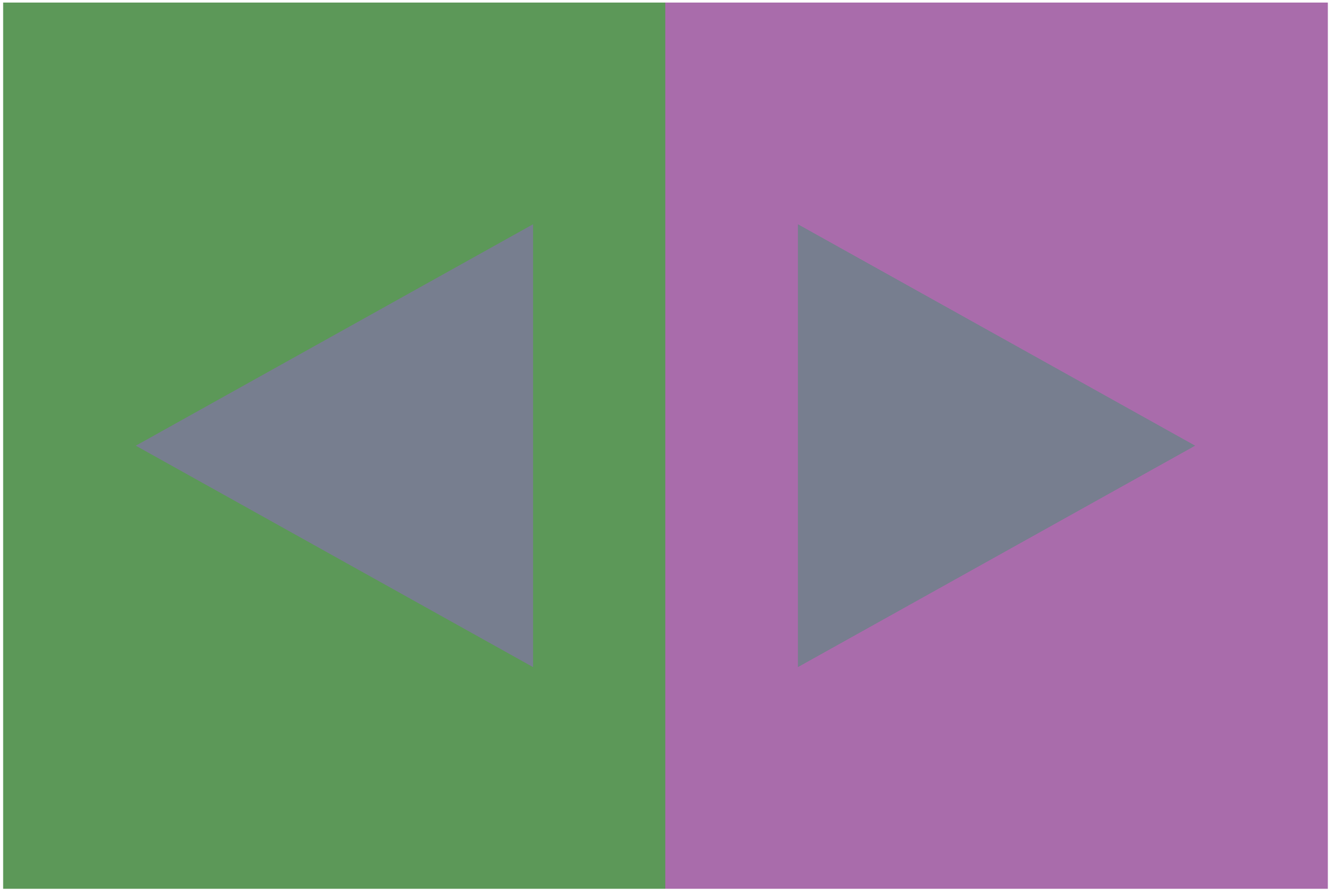
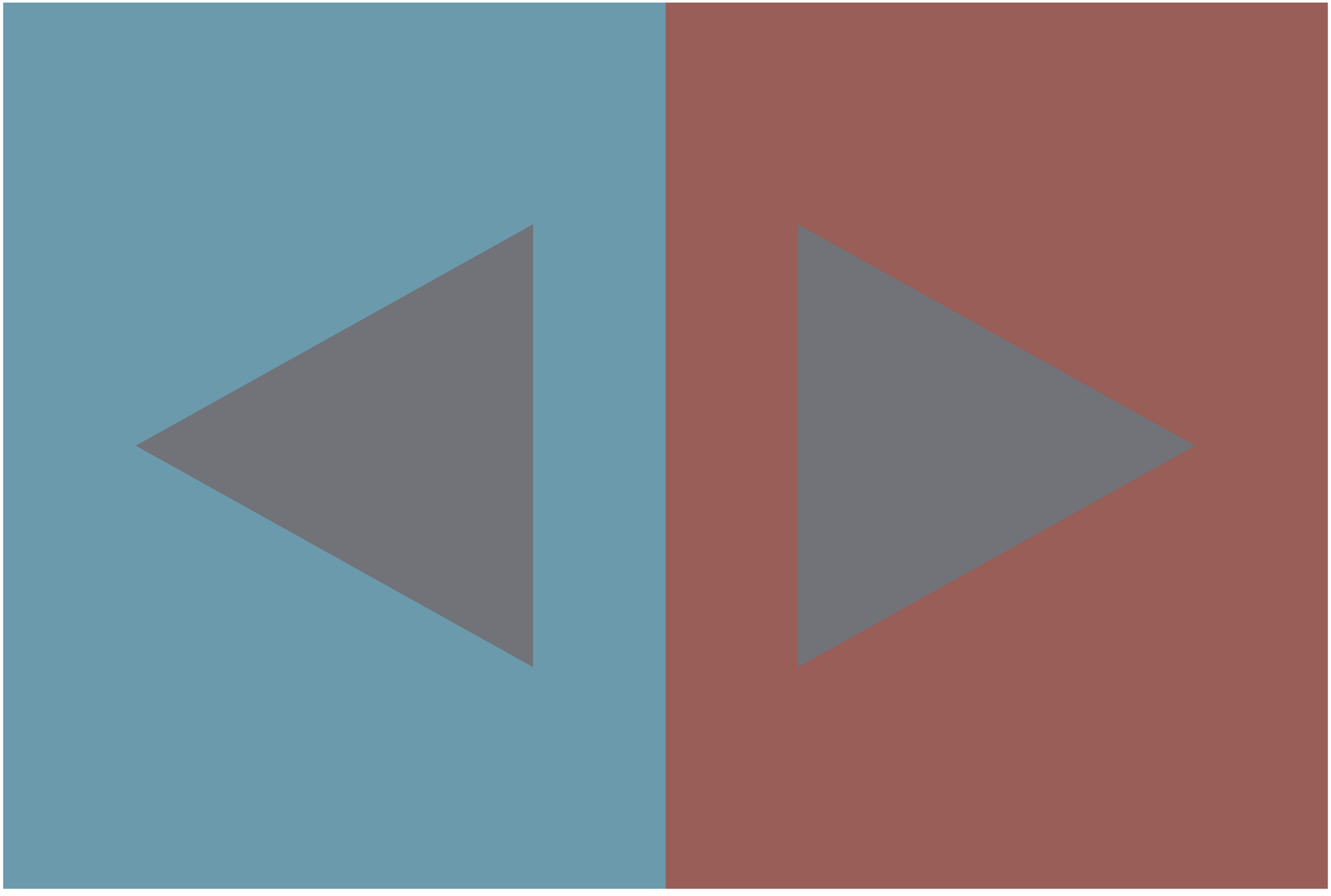
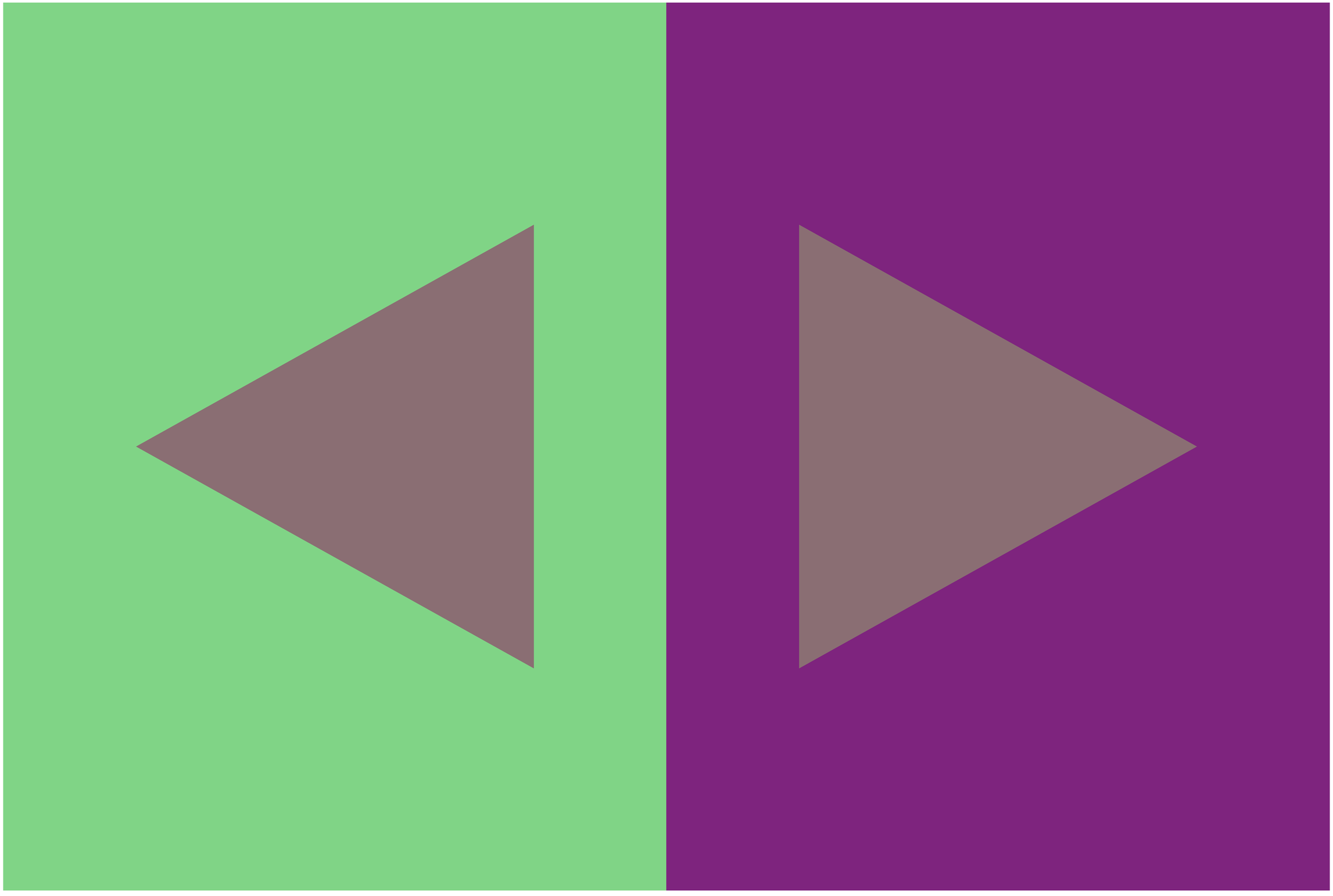
# THE COLORIST COOKBOOK

"ALL COLORS ARE THE FRIENDS OF THEIR NEIGHBORS AND THE LOVERS OF THEIR OPPOSITES."



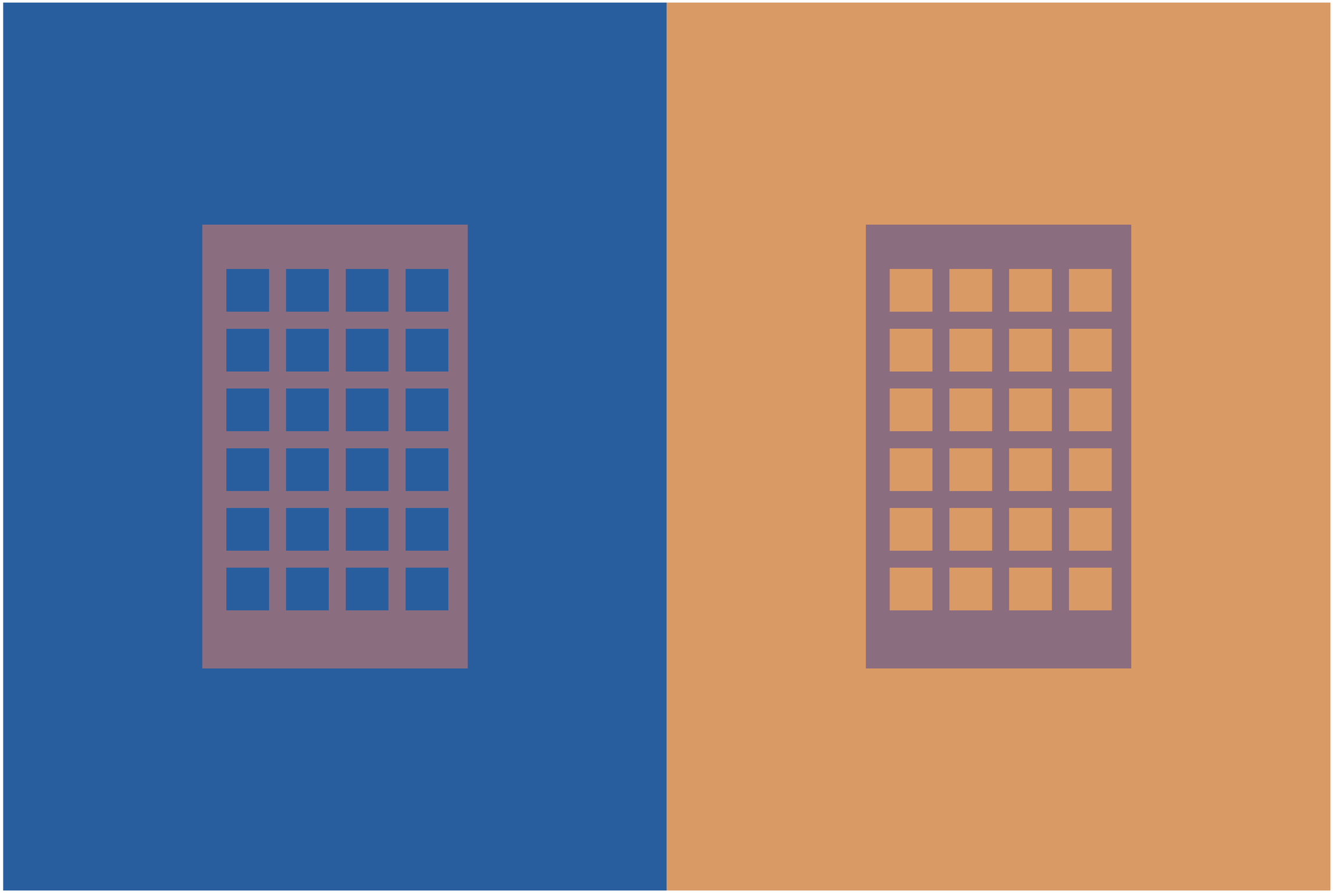
h

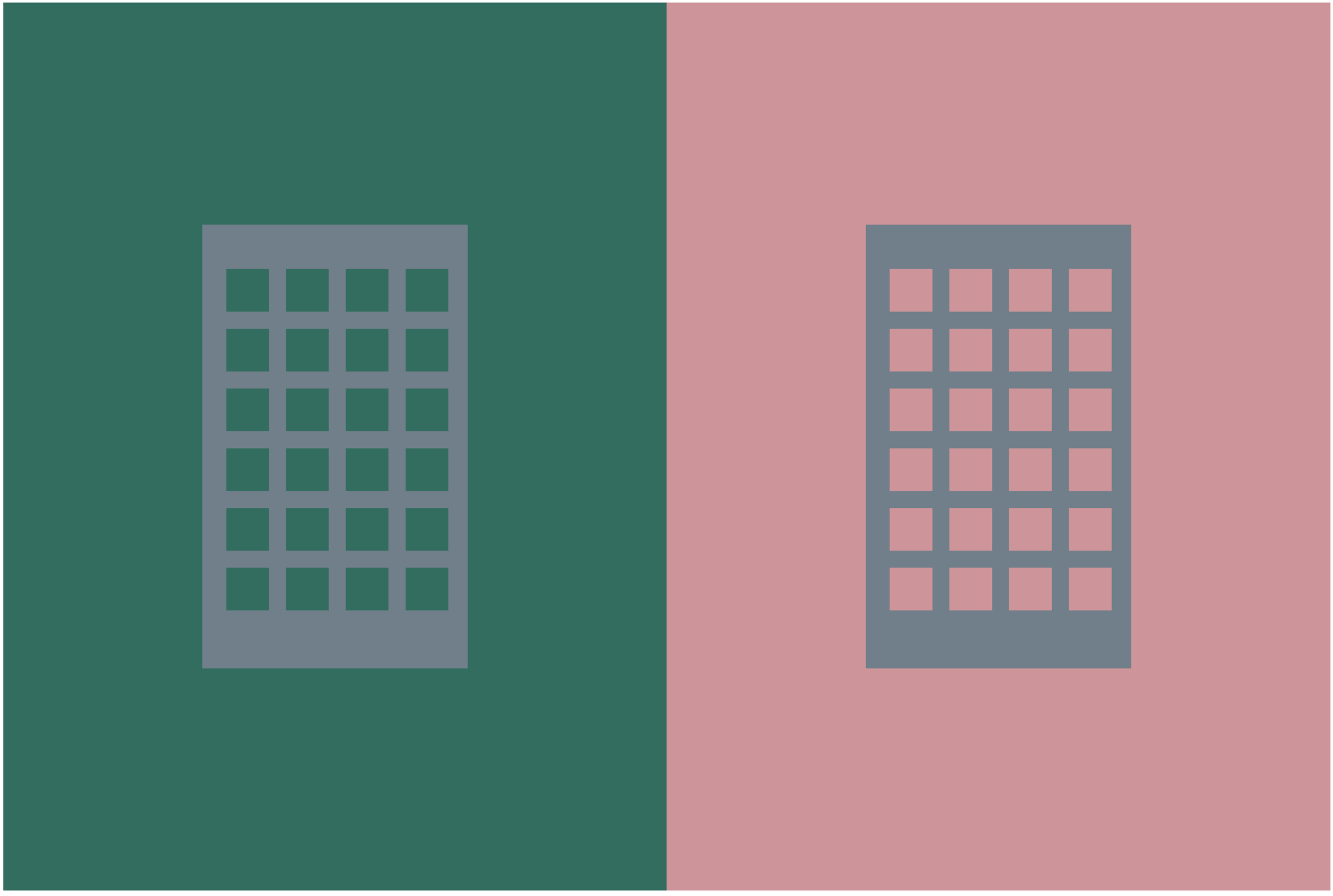


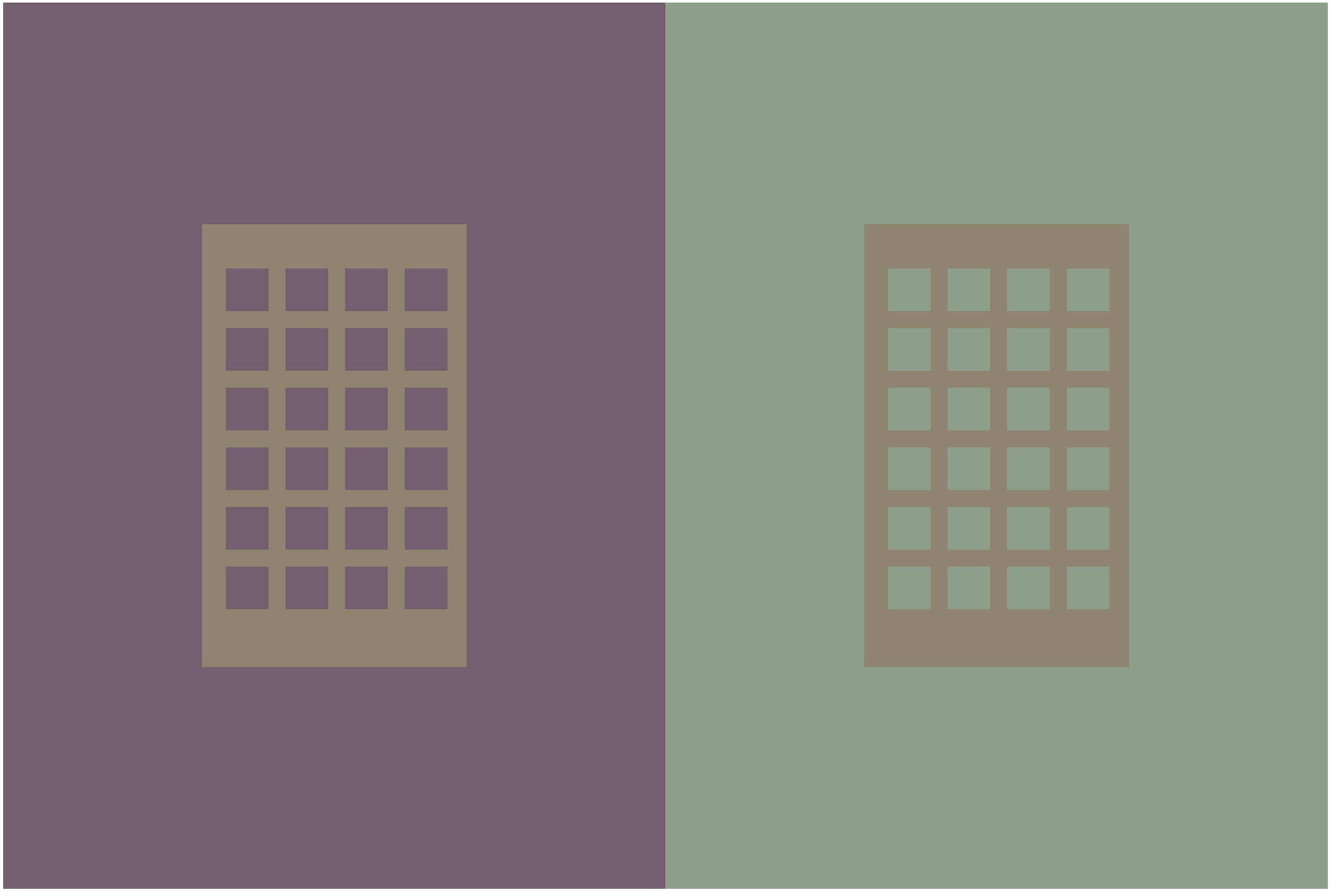


```
// recipe for simple simultaneous contrast

  // 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 '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);
```

```
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();
```



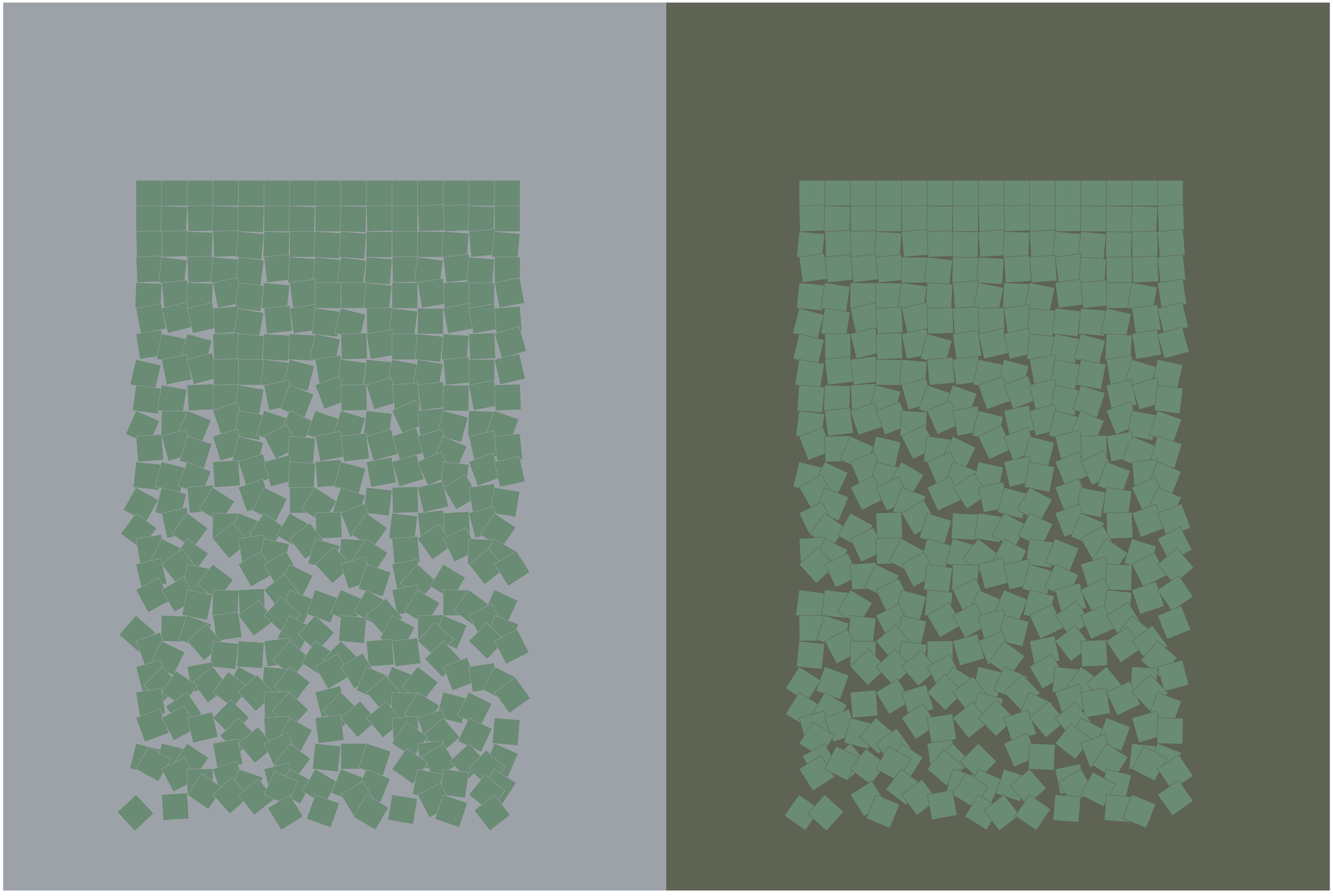




```
// recipe for holed simultaneous contrast

  // 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);
```

```
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
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();
//translate(rand3, rand4);
```

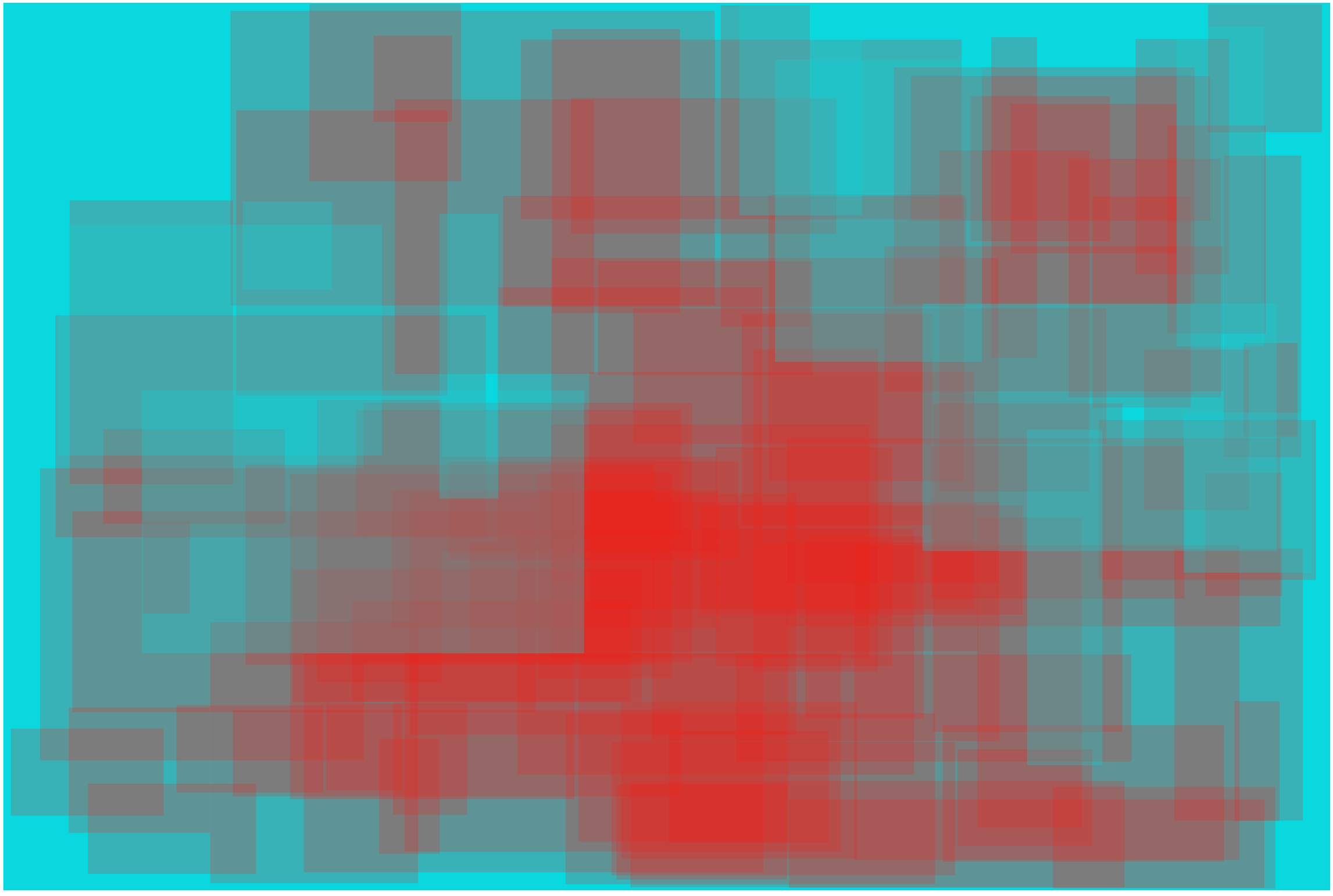
```
    rect(pgwidth * 0.668 + 140*cols, pgheight * 0.3 + 140*rows,100,100);
  }
}
popMatrix();

popMatrix();
```

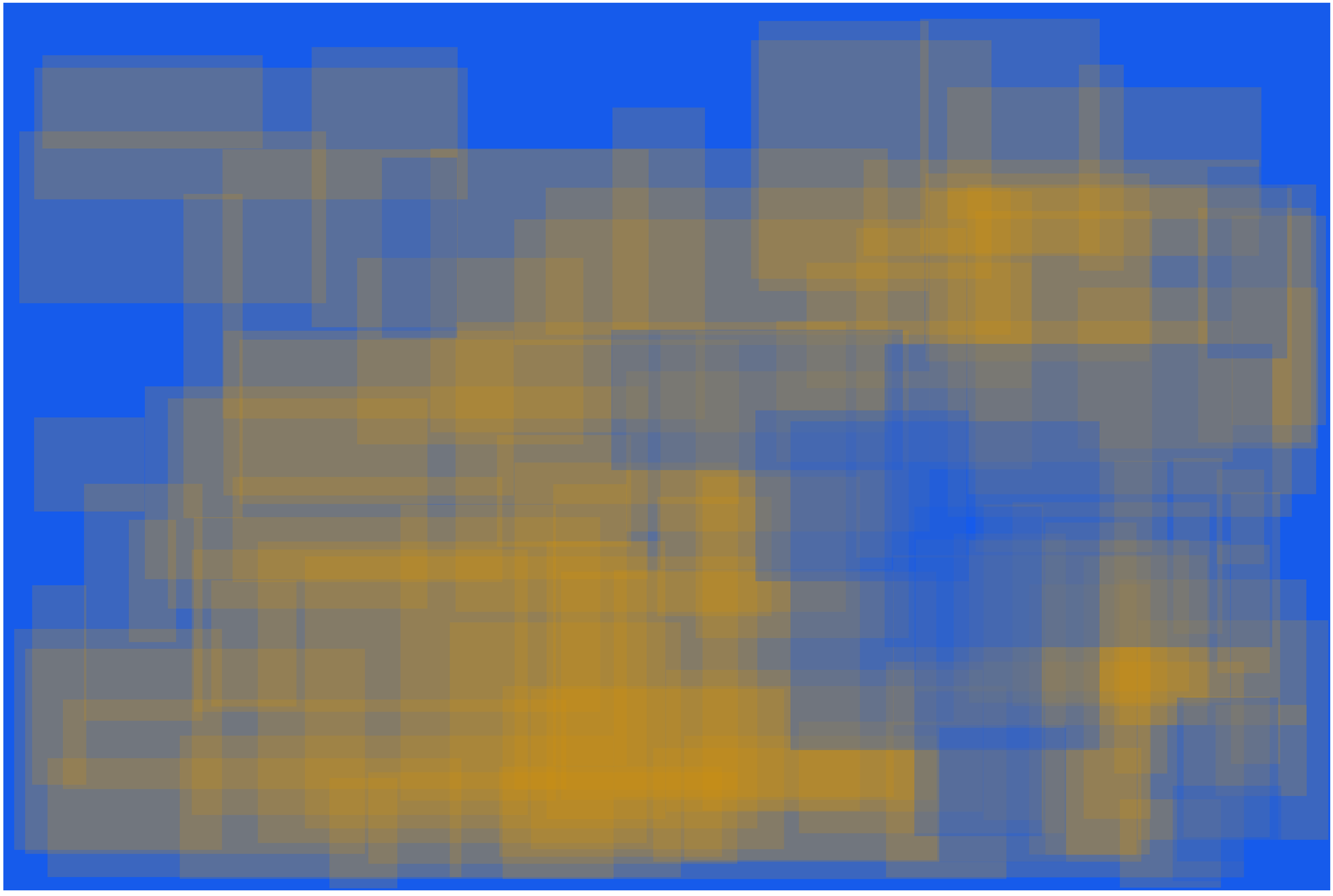


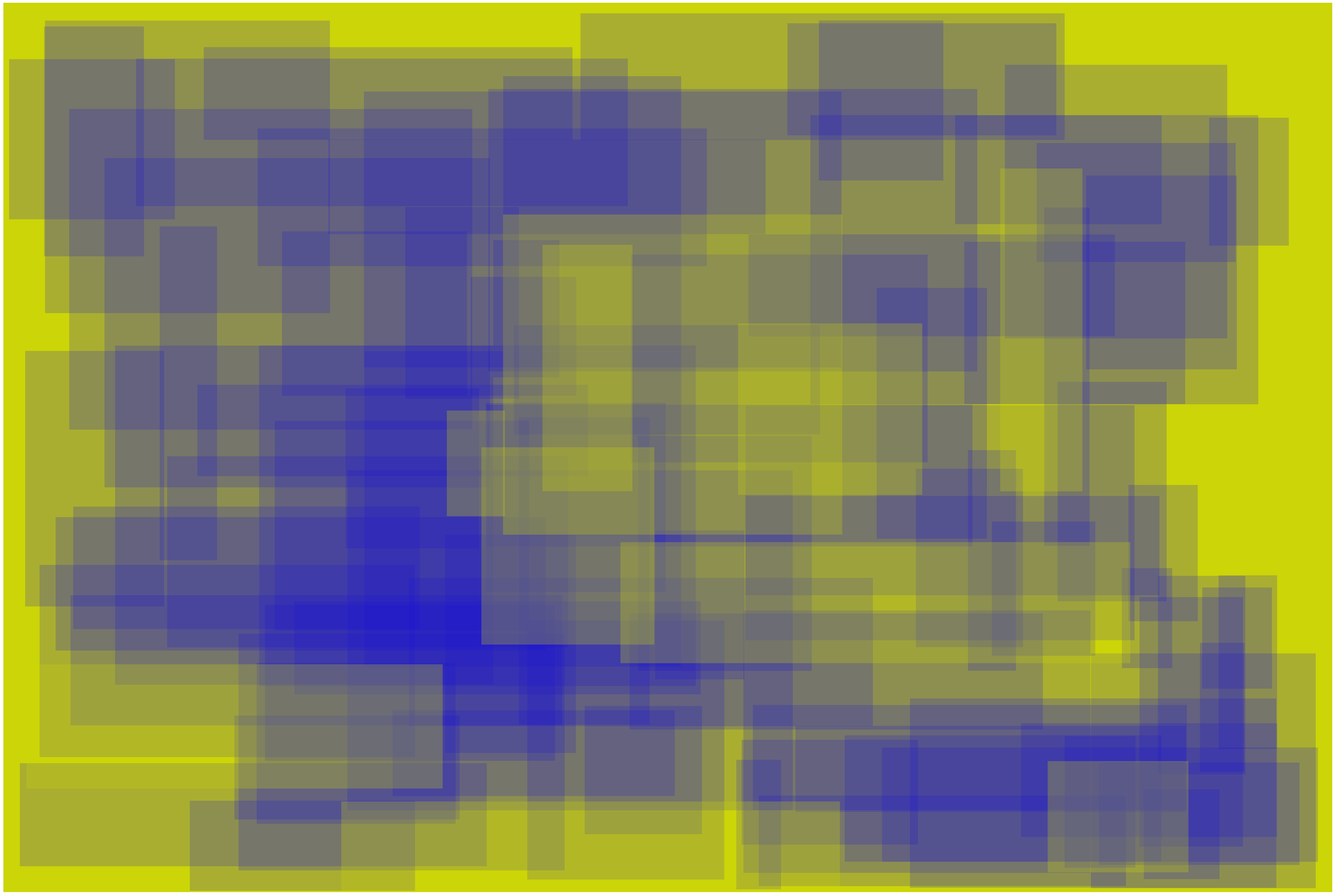
```
// recipe for schotter simultaneous contrast

  // 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);
```

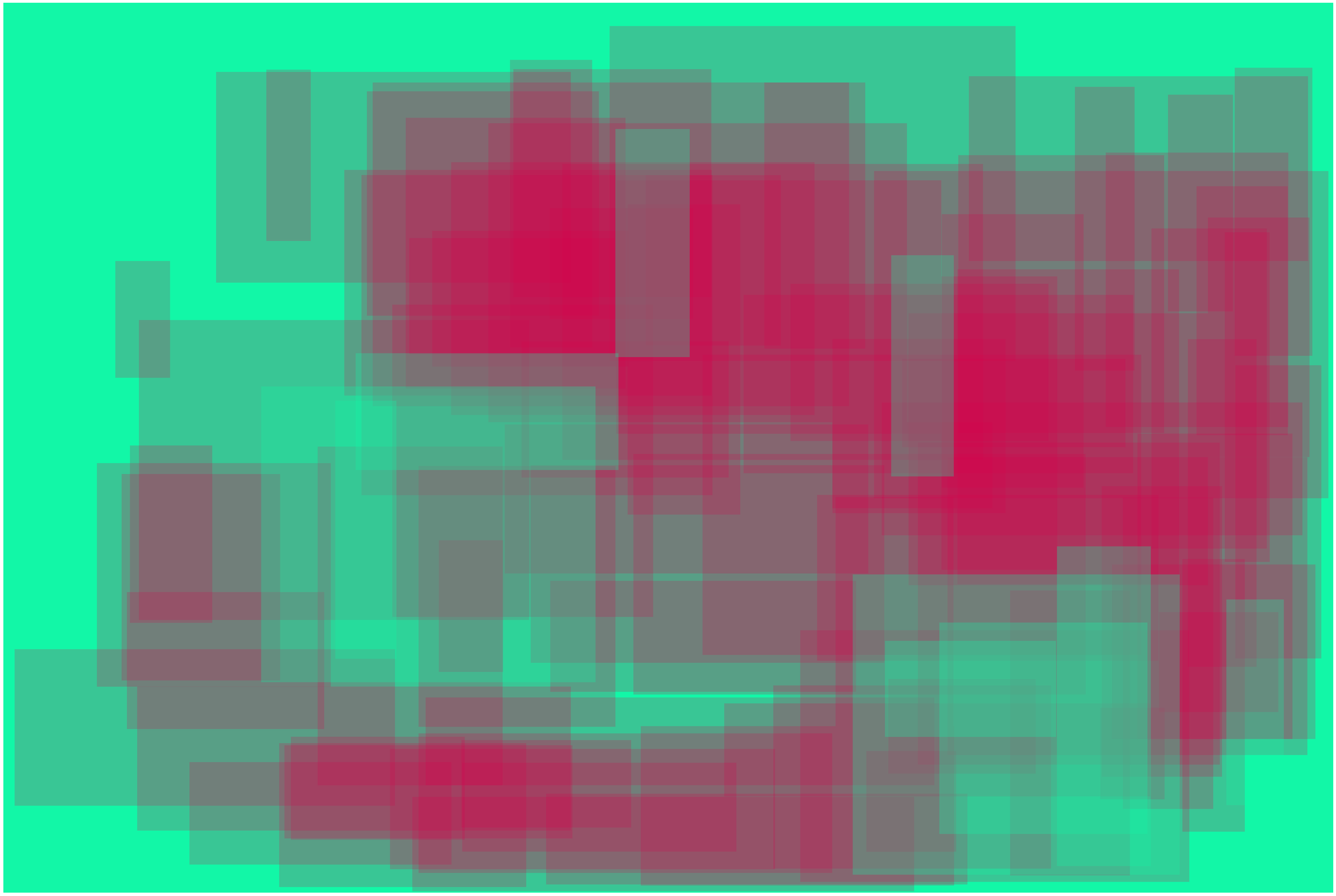
```
fill(col1);
stroke(col1);
rect(0,0,pgwidth/2f,pgheight);
fill(col2);
stroke(col2);
rect(pgwidth/2f,0,pgwidth/2f,pgheight);

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);
    float rotamnt = random(-rows*0.05,rows*0.05);
    rotate(rotamnt);
    rect(0,0,60,60);
    popMatrix();

  }
}
rectMode(CORNER);
fill(mid);
stroke(col2);
for(int rows = 0; rows < 25; rows++) {
  for(int cols = 0; cols < 15; cols++) {
    pushMatrix();
    translate(pgwidth * 0.6 + 60*cols, pgheight * 0.2 + 60*rows);
```

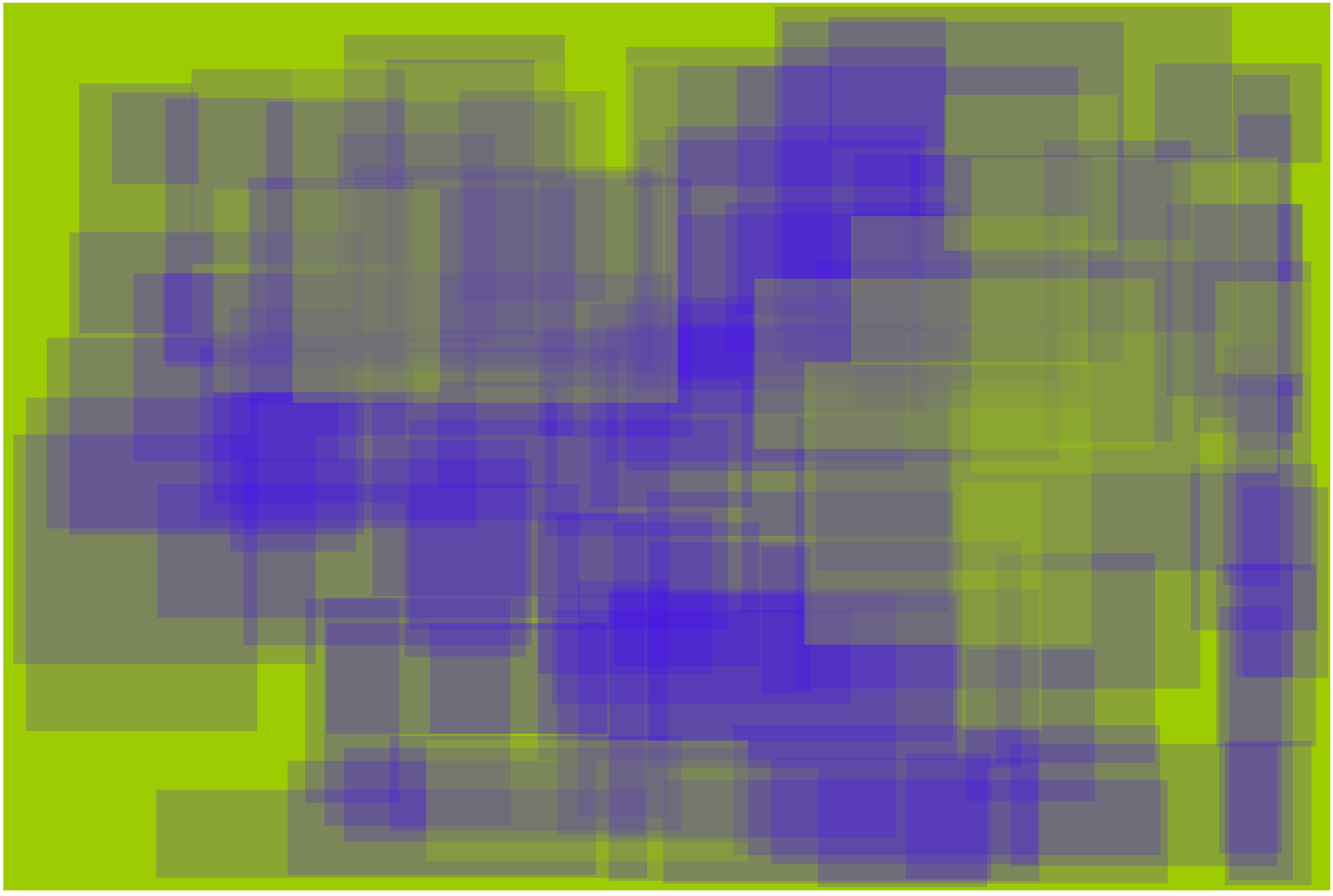
```
    rect(0,0,60,60);
    popMatrix();

  }
}


popMatrix();
```





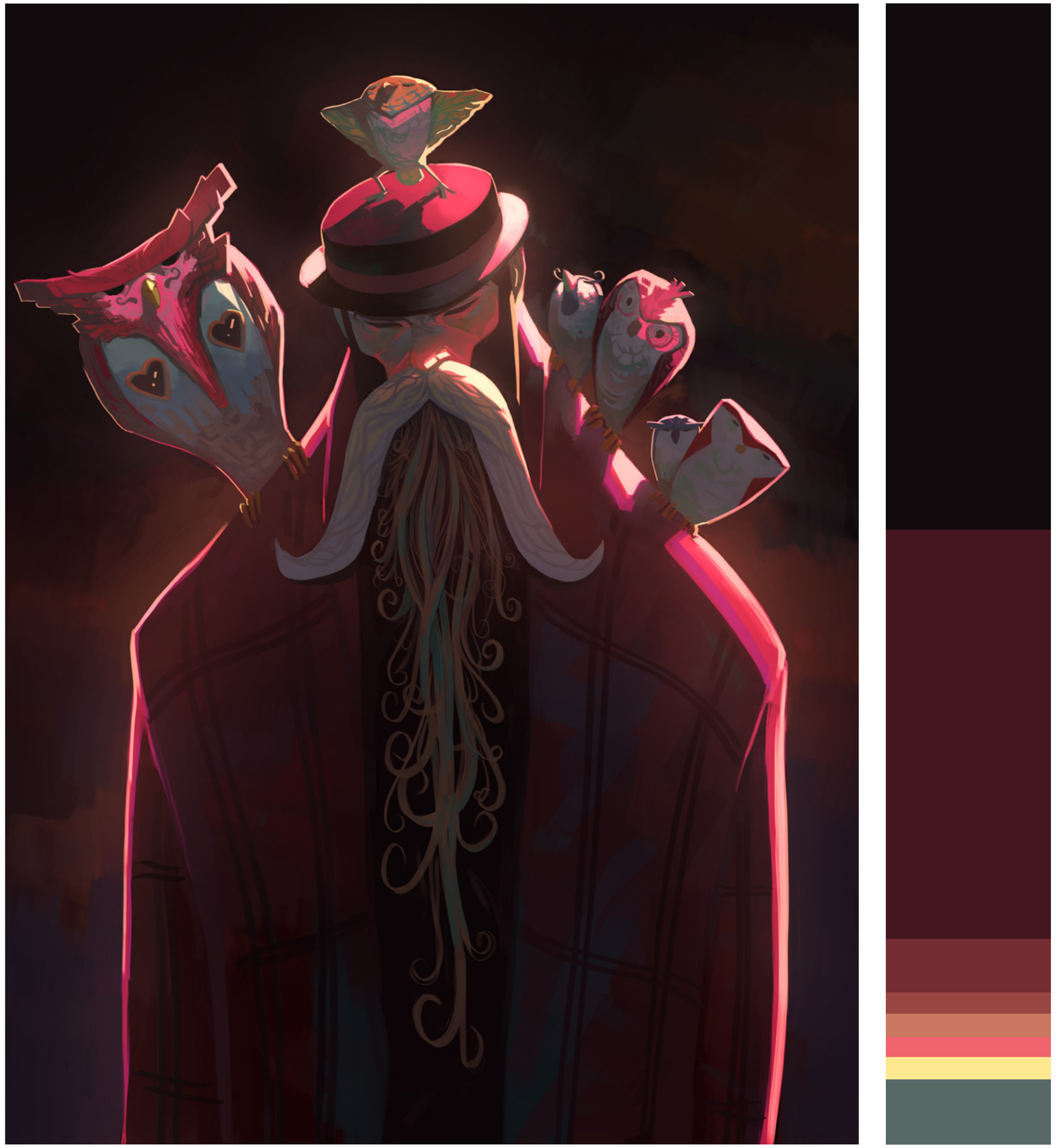




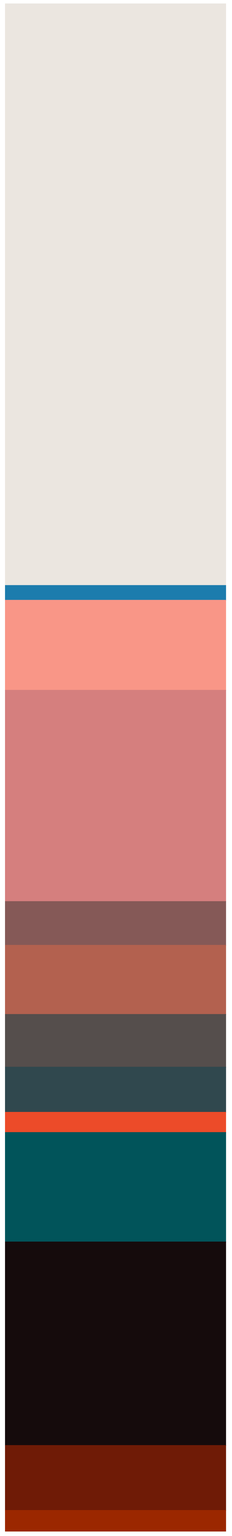


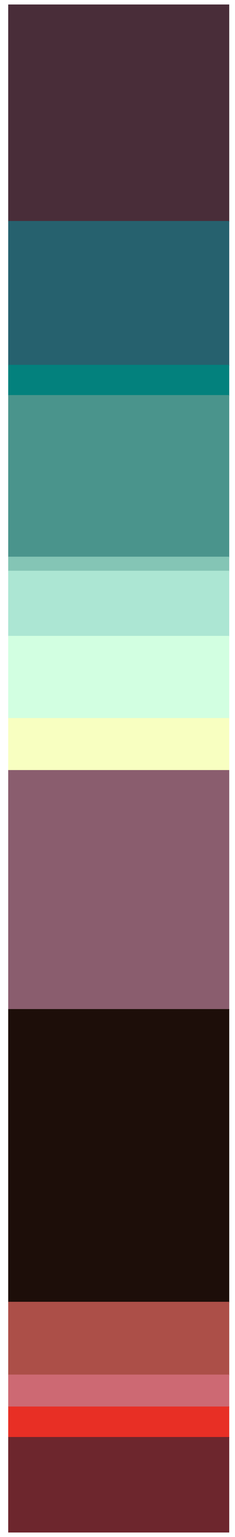
```
colorMode(HSB,360,100,100);
color col1 = color(random(0,360), random(90,100), random(80,100));

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

color bg = color((hue(col1)+180)%360, random(90,100), random(80,100));
fill(bg);
rect(0, 0, pgwidth,pgheight);

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));

  fill(col1, 50);
  rect(posX, posY, rectw,recth);

}

for(int i = 0; i < 10; i++) {
  float posX = random(0, pgwidth-200);
  float posY = random(0, pgheight-200);
```

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

}

popMatrix();
colorMode(RGB);
```




OM





```
// for the palette
ArrayList<Pixel> colors = new ArrayList<Pixel>();
float tolerance = 50.0;
int totalCount = 0;
float proptolerance = 0.005;
PImage[] images = new PImage[7];

public class Pixel {
  public color pixelcolor;
  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();
  p.pixelcolor = currpix;
  p.count = 1;
  colors.add(p);
  totalCount++;
}
```

```
}

void palettePage(int imgindex) {
  //
  totalCount = 0;
  // colors is an ArrayList of pixel objects
  // it is initialized as: ArrayList<Pixel> colors = new ArrayList<Pixel>();
  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;

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 {
         int idx;
         boolean foundMatch = false;
```

```
for(idx = 0; idx < colors.size(); idx++) {
  float currDist = colorDist(currpix, colors.get(idx).pixelcolor);
  if(currDist < tolerance) {
        if(foundMatch && (currDist < minDist)) {
          colors.get(idx).count++;
          colors.get(prevIdx).count--;

          prevIdx = idx;
          minDist = currDist;
        }
        else if(!foundMatch && (currDist < minDist)) {
          colors.get(idx).count++;
          totalCount++;
          foundMatch = true;
          prevIdx = idx;
          minDist = currDist;
         }
       }
    }

    if(!foundMatch) {
      // couldn't find a matching color
      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;
color prevColor= colors.get(0).pixelcolor;

pushMatrix();
translate(margin + 700, margin);
image(images[imgindex], 0,0);

for(int idx = 0; idx < colors.size(); idx++) {
  float prop = colors.get(idx).count / ((float)netCount);
  if(prop >= proptolerance) {
    float rheight = pgheight*prop;

    stroke(colors.get(idx).pixelcolor);
    fill(colors.get(idx).pixelcolor);

    rect(palettepos, ypos, 300, rheight);
    ypos += rheight;
    prevColor = colors.get(idx).pixelcolor;
```

```
  popMatrix();
}
```