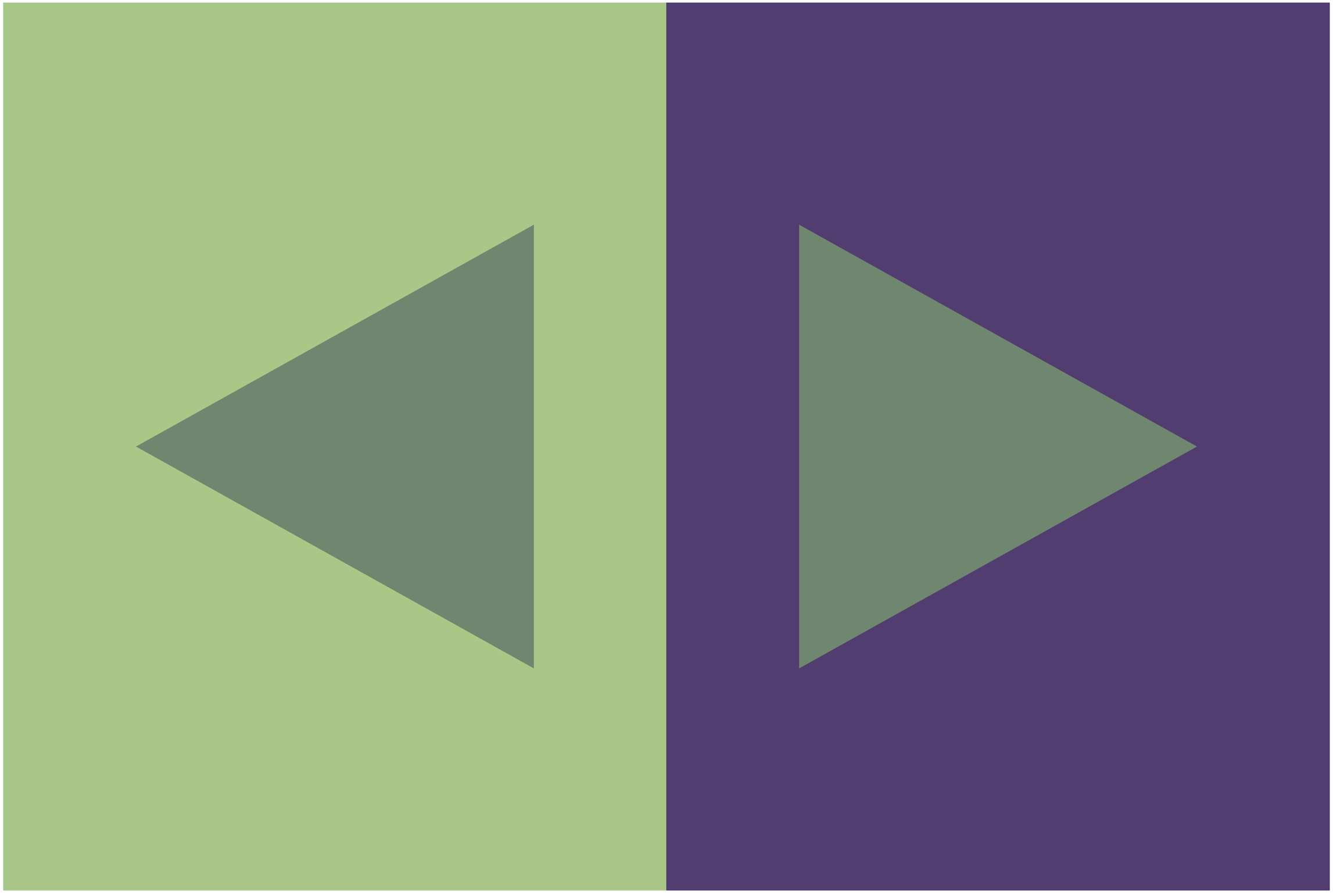
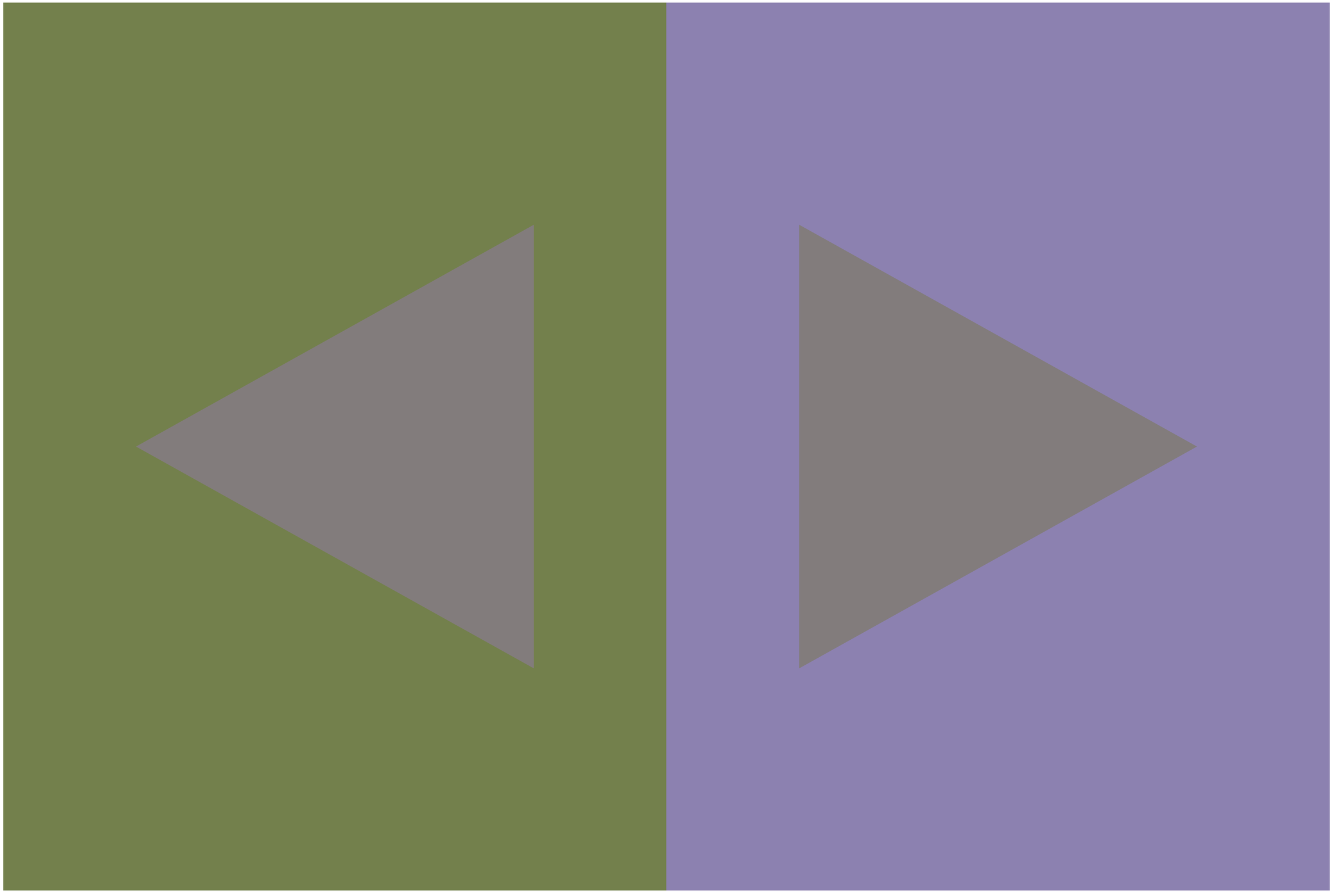
THE COLORIST COOKBOOK

"NO ONE IS BORN A GREAT COOK, ONE LEARNS BY DOING."

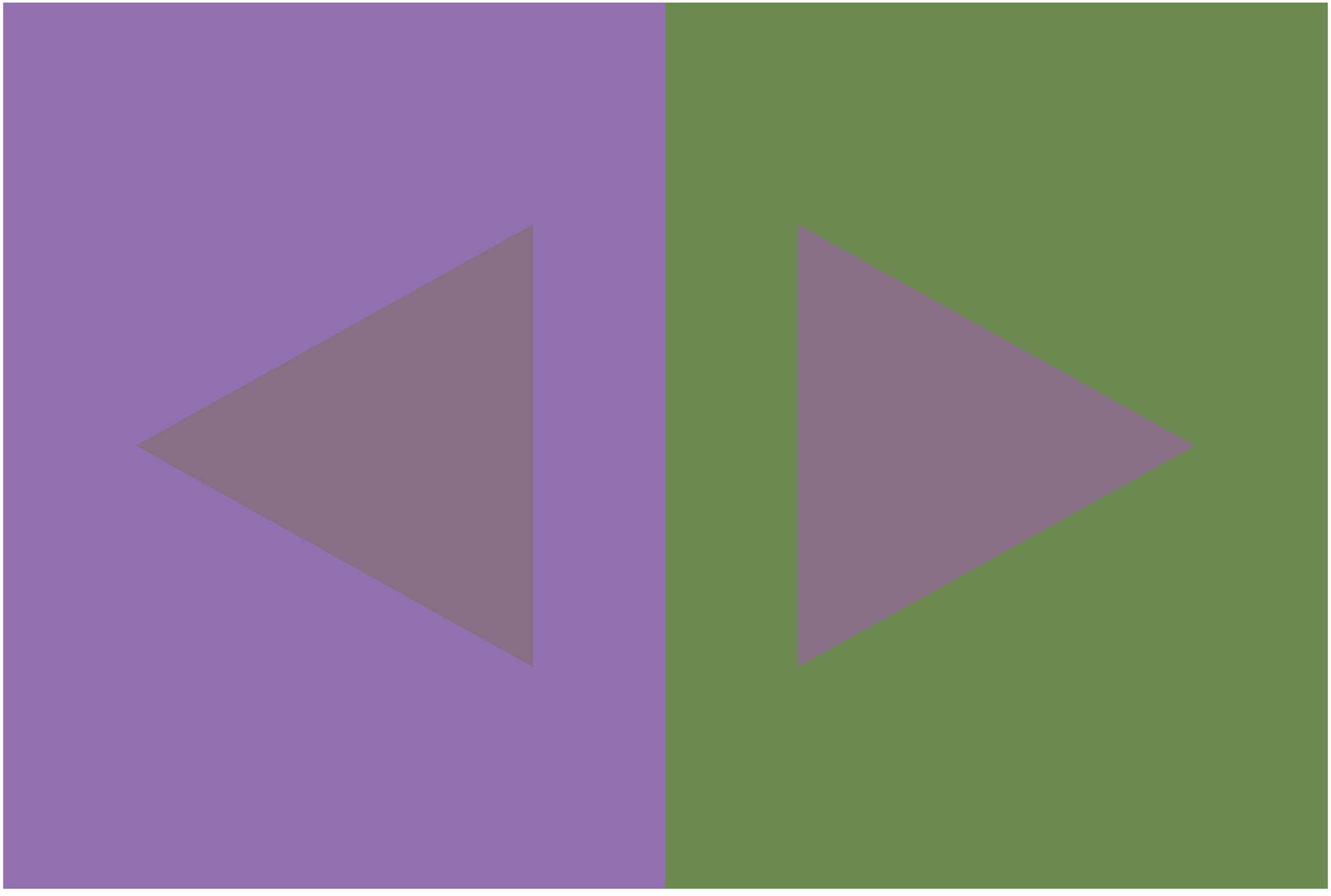
0xFFA9C787 0xFF523D71 0xFF6F876E



0xFF73804C 0xFF8C81B0 0xFF827C7C



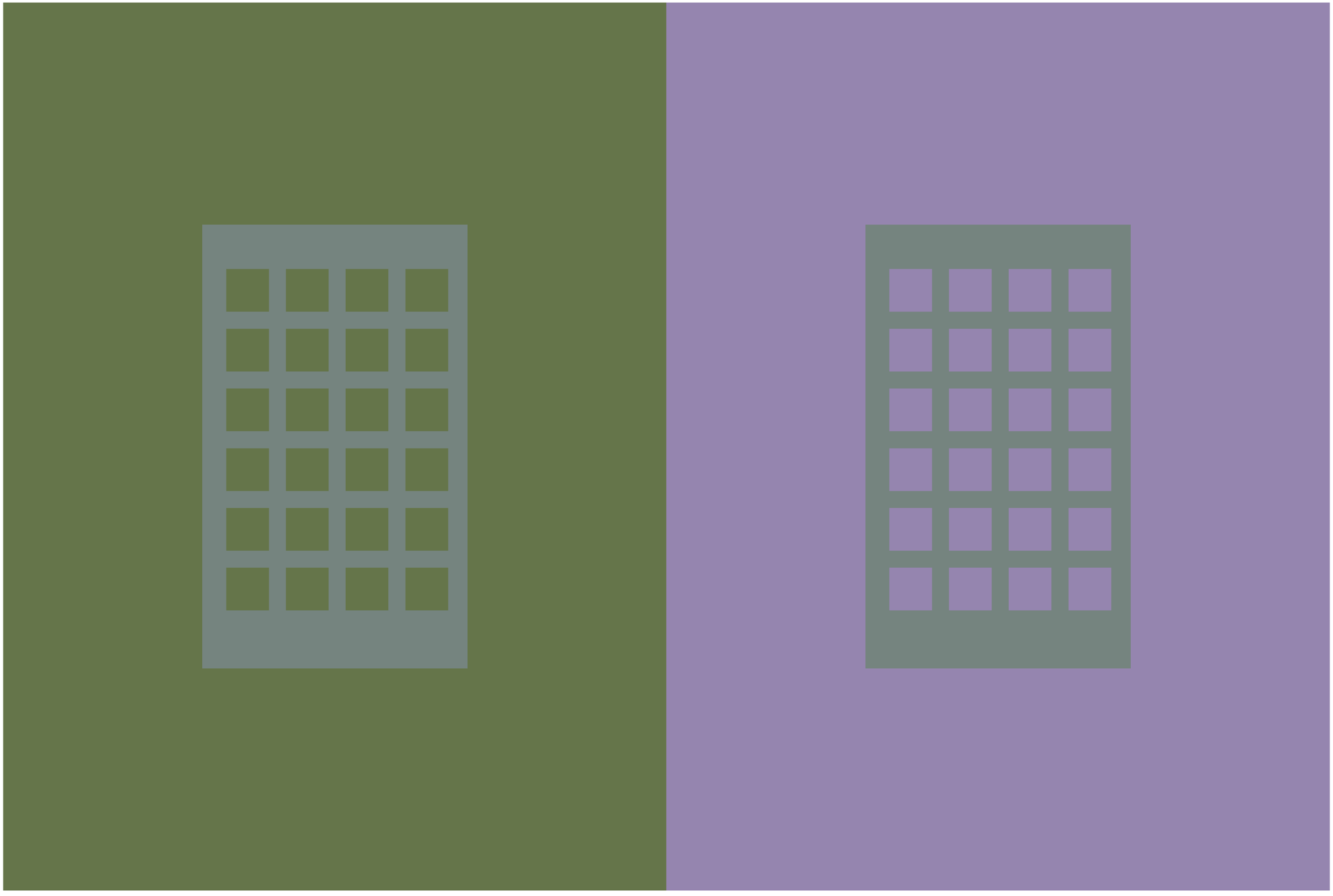
0xFF946FAF 0xFF6A8A4F 0xFF877085



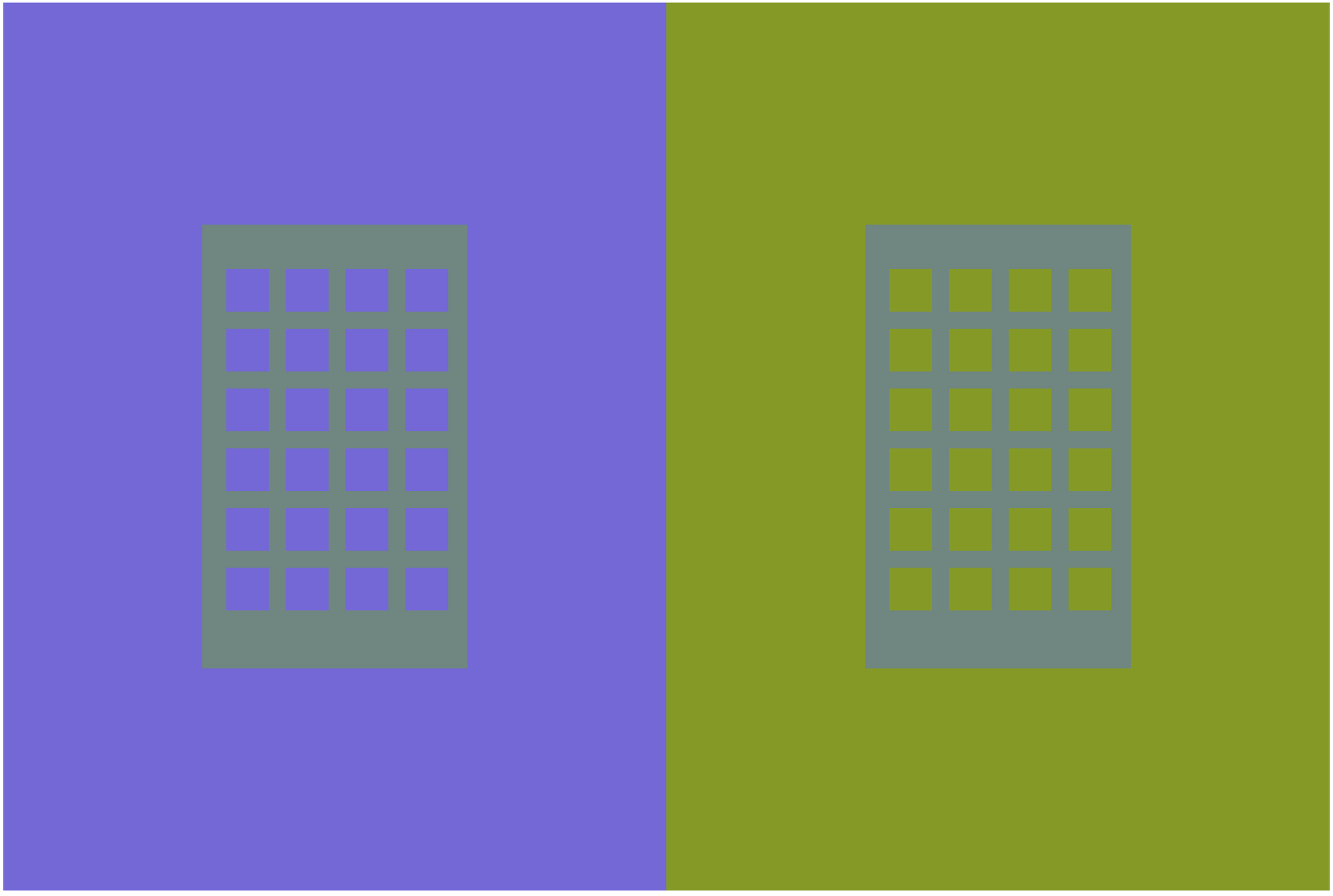
```
// recipe for simple simultaneous contrast

// 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();
```

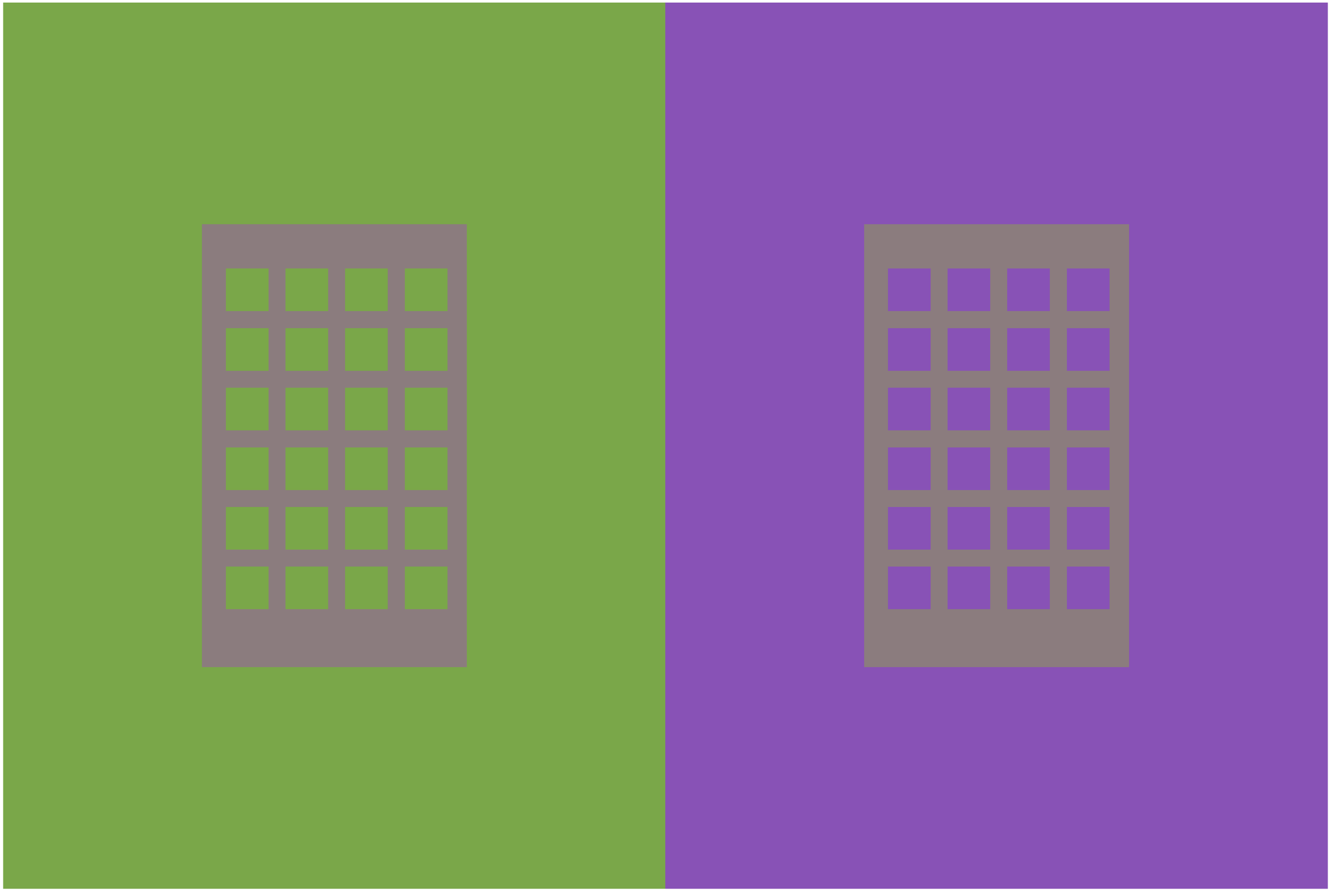
0xFF65754A 0xFF9585AF 0xFF75847F



0xFF7368D6 0xFF859A26 0xFF6F8681



0xFF7AA749 0xFF8852B6 0xFF8B7C7E

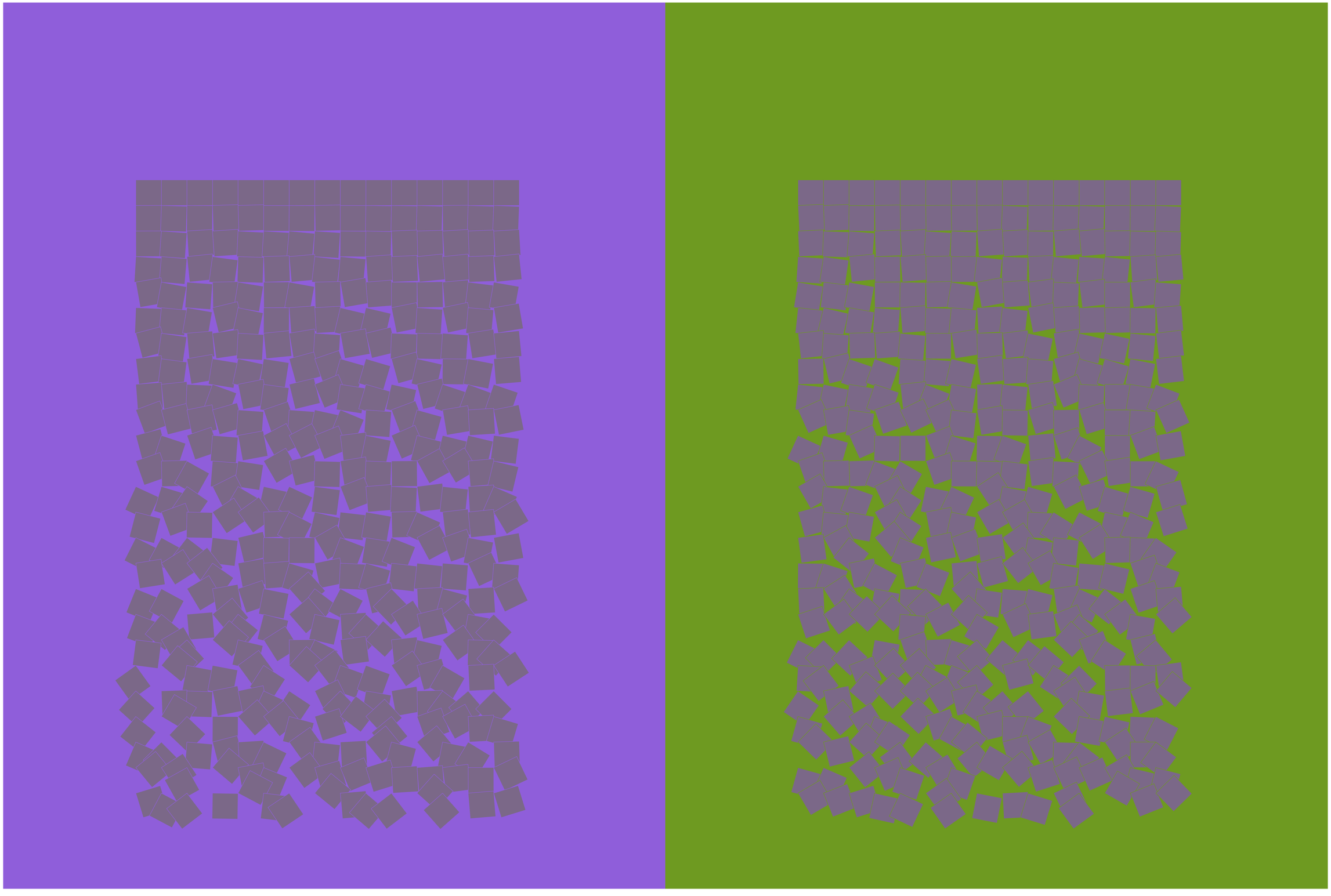


```
// recipe for holed simultaneous contrast

// 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();
```

0xFF8F5EDA 0xFF6E9A21 0xFF7B6888

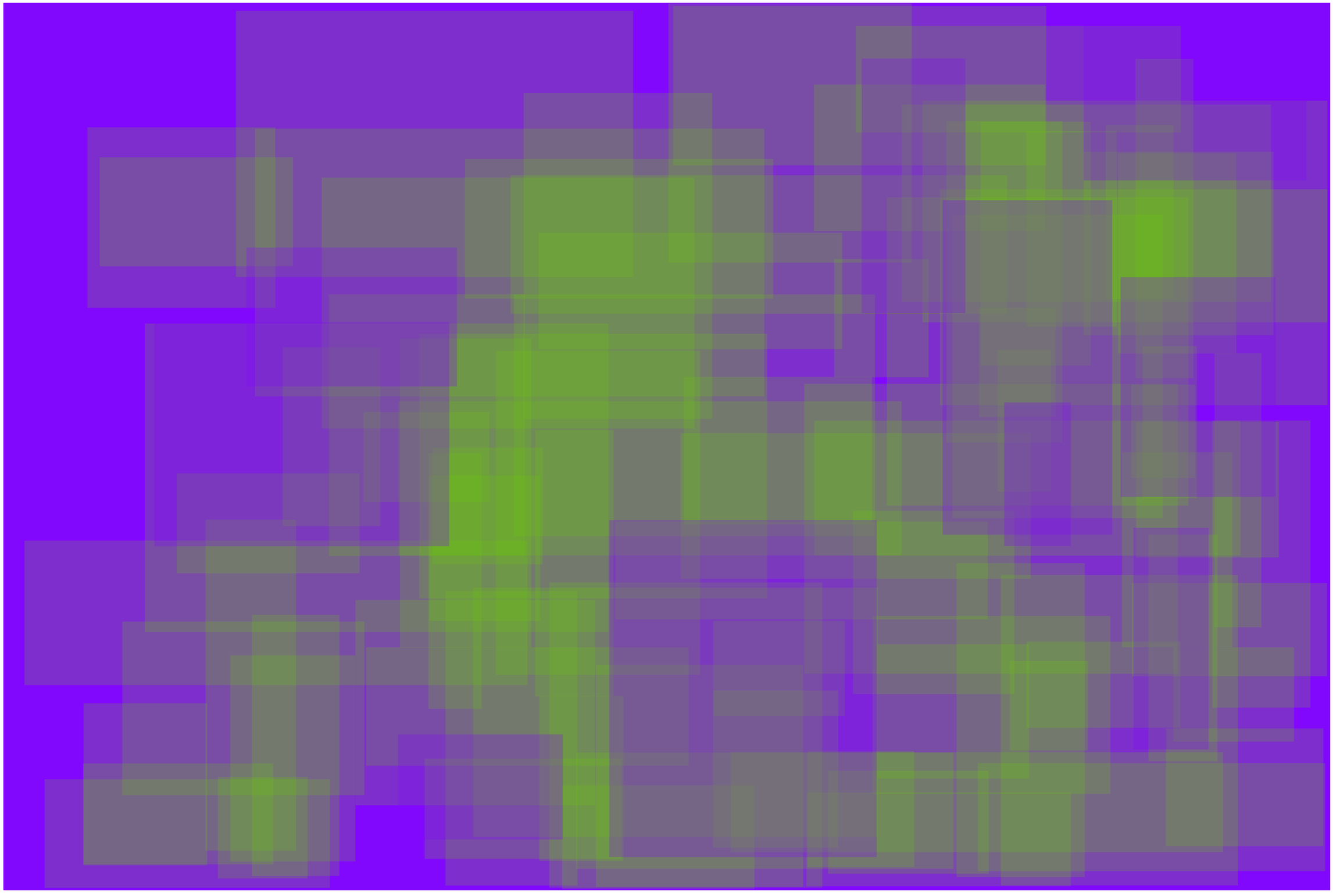


```
// recipe for schotter simultaneous contrast

// 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
```

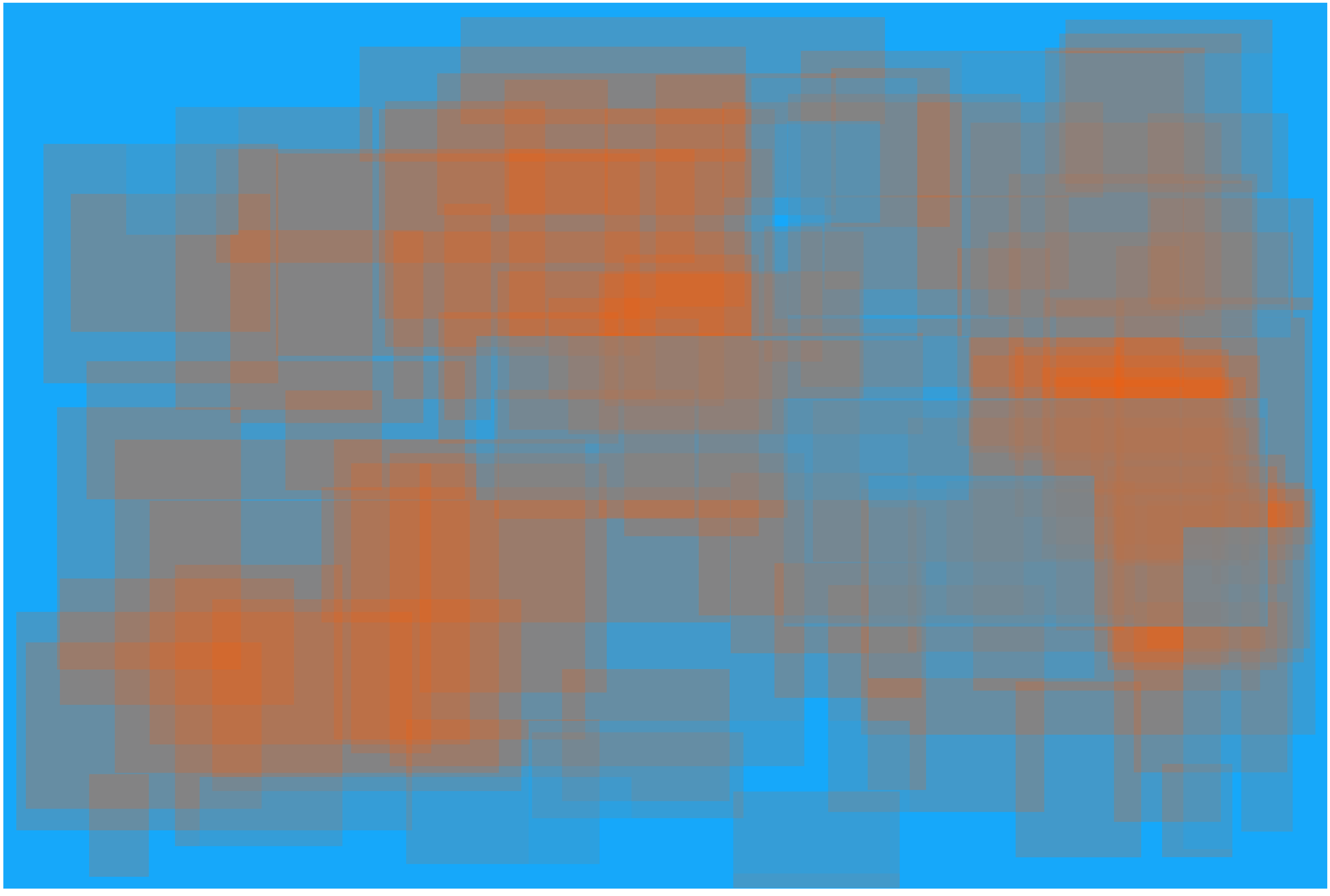
```
fill(col1);
stroke(col1);
rect(0,0,pgwidth/2f,pgheight);
fill(col2);
stroke(col2);
rect(pgwidth/2f,0,pgwidth/2f,pgheight);

// 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();
```

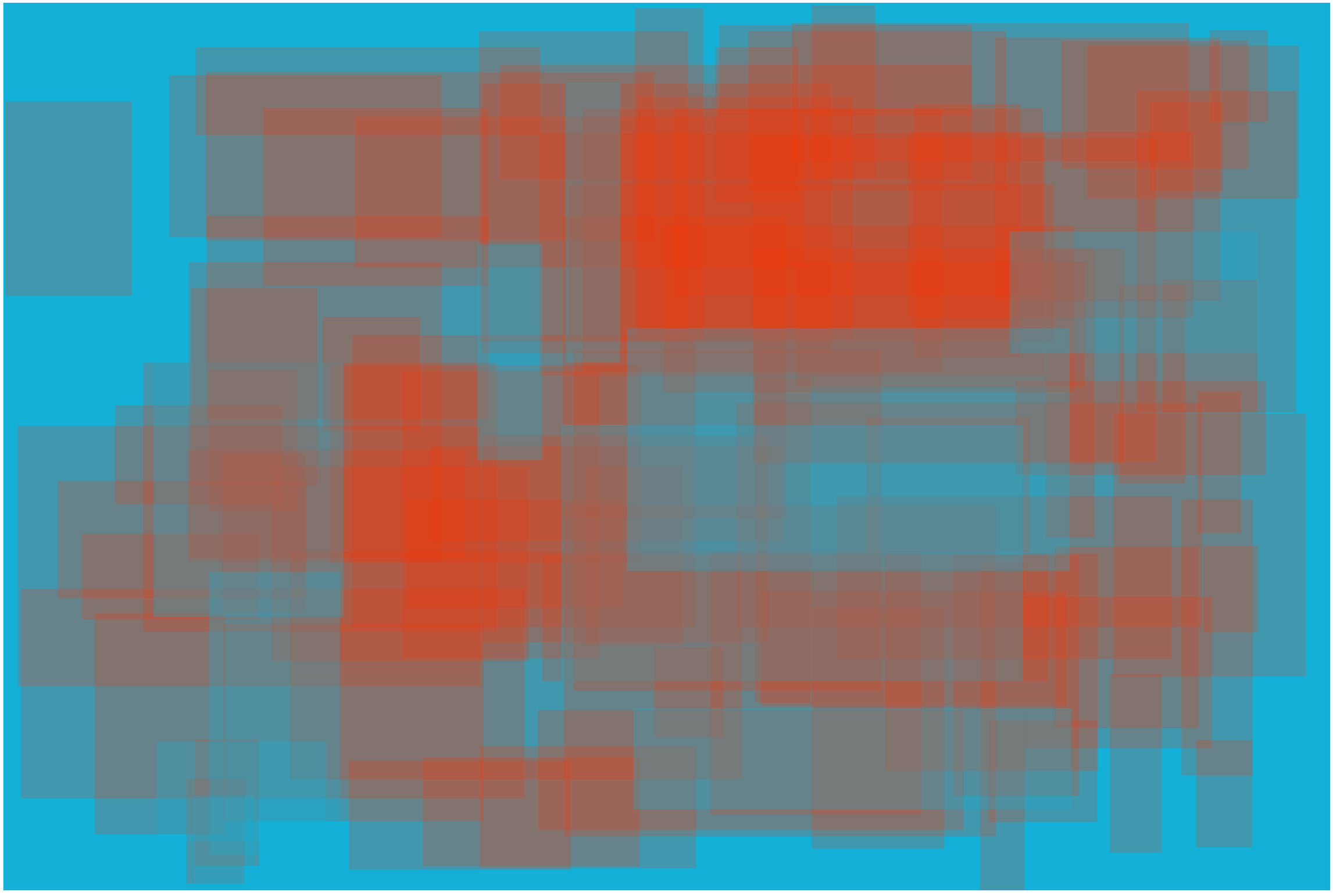
0xFF6BCD07 0xFF8208FD



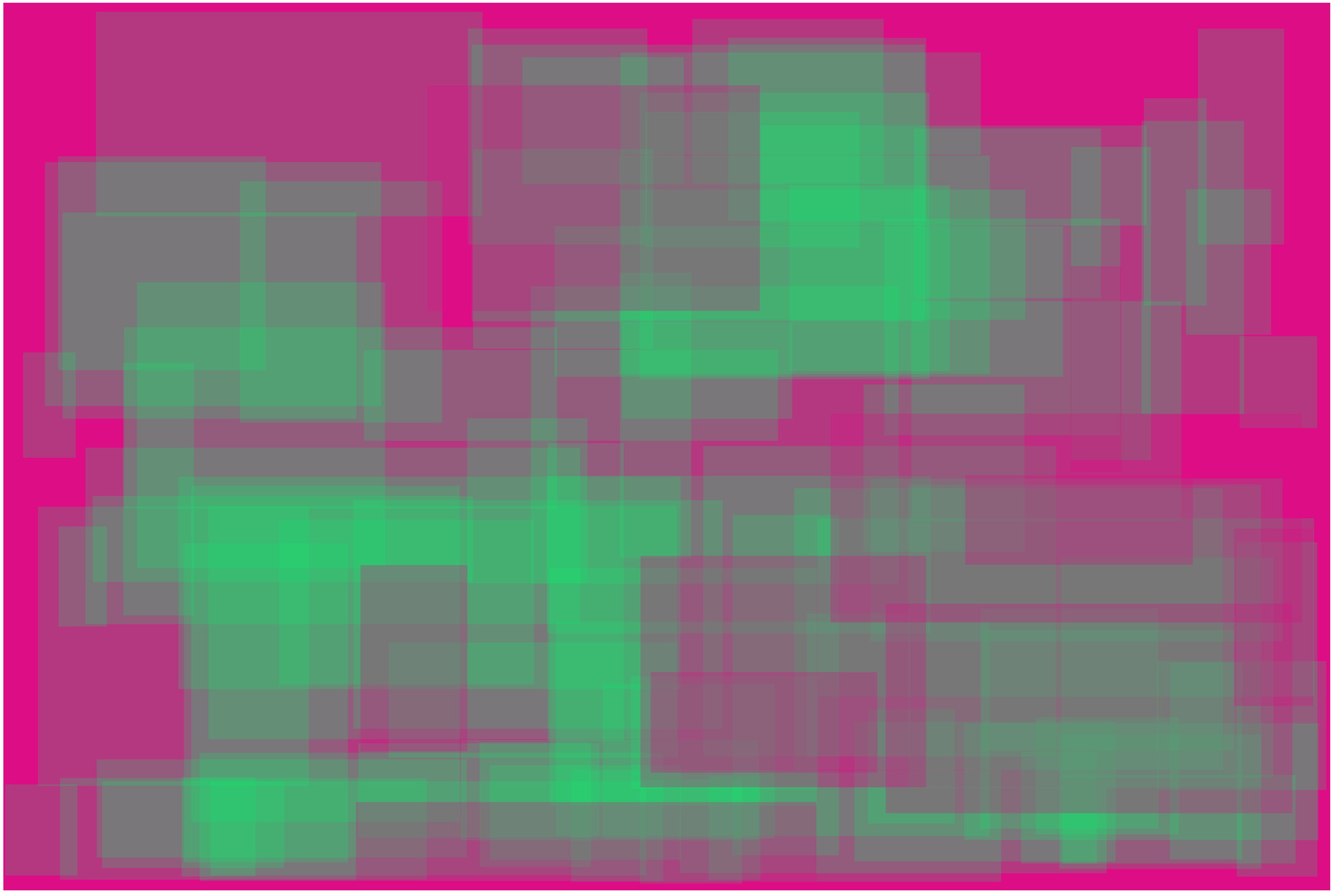
0xFFFB5E06 0xFF16A8FA



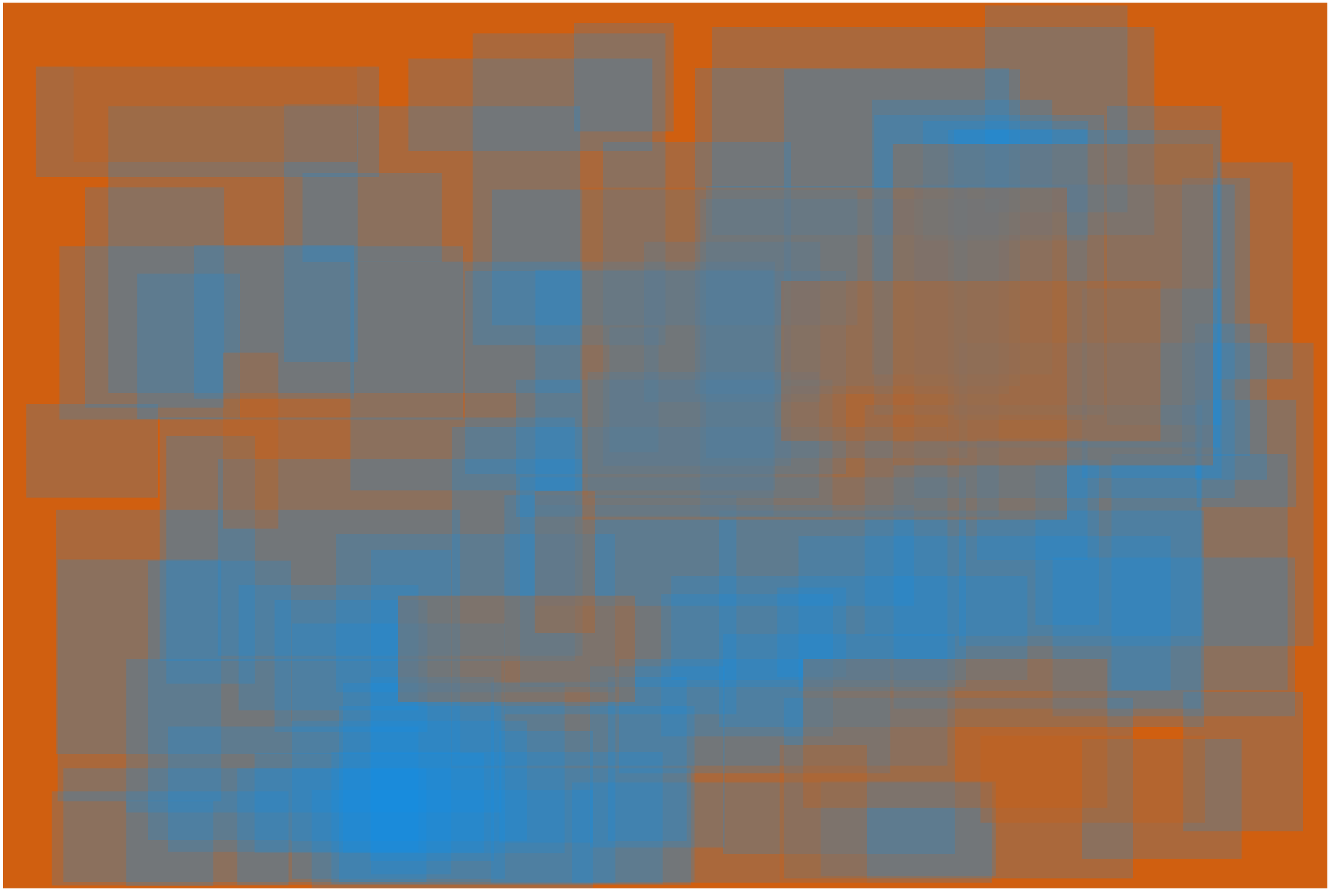
0xFFFE3301 0xFF14B0D7



0xFF0FED6D 0xFFDD0D85



0xFF0F91EC 0xFFD05F10



```
// 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();
```

0xFF1E1111

0xFF422727

0xFF6C3E3D

0xFF9D414A

0xFFCD4960

0xFFF57981

0xFFFDC6B8

0xFF574748

0xFF99B2AB
0xFFBCD5CF
0xFFEABEBF
0xFF6DA9A7
0xFF8CEAD8
0xFFEE92A2
0xFFF5F5EB
0xFF776668
0xFF44373E
0xFF1B1523
0xFFCC7C90
0xFF120F1A

0xFFCFDBD8

0xFF9ABCBC

0xFF71A4AA

0xFFE8ECE9

0xFF5A8D95

0xFF0067EB5

0xFF4C5757

0xFFF6E6059

0xFFAE6E5F

0xFF8F6055

0xFFD6C8C2

0xFFB0968A

0xFF8A6B5A

0xFF603F32

0xFF391E19

0xFF1C0D0D

0xFFEDE6E3

0xFF8A6A1C
0xFFA68519
0xFFD0B705
0xFFF6F007
0xFF4D3D30
0xFF342919
0xFF1A170D
0xFF917C66
0xFF335D43
0xFF6D644D

0xFF464858

0xFF24465B

0xFF0E7C87

0xFF506F78

0xFE8E8488

0xFFB3ABAB

0xFFD5CECB

0xFFFAF8C0

0xFF855F64

0xFF160F29

0xFFB14749

0xFFD14243

0xFFE23433

0xFF6B4551

```
// 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();

}
```