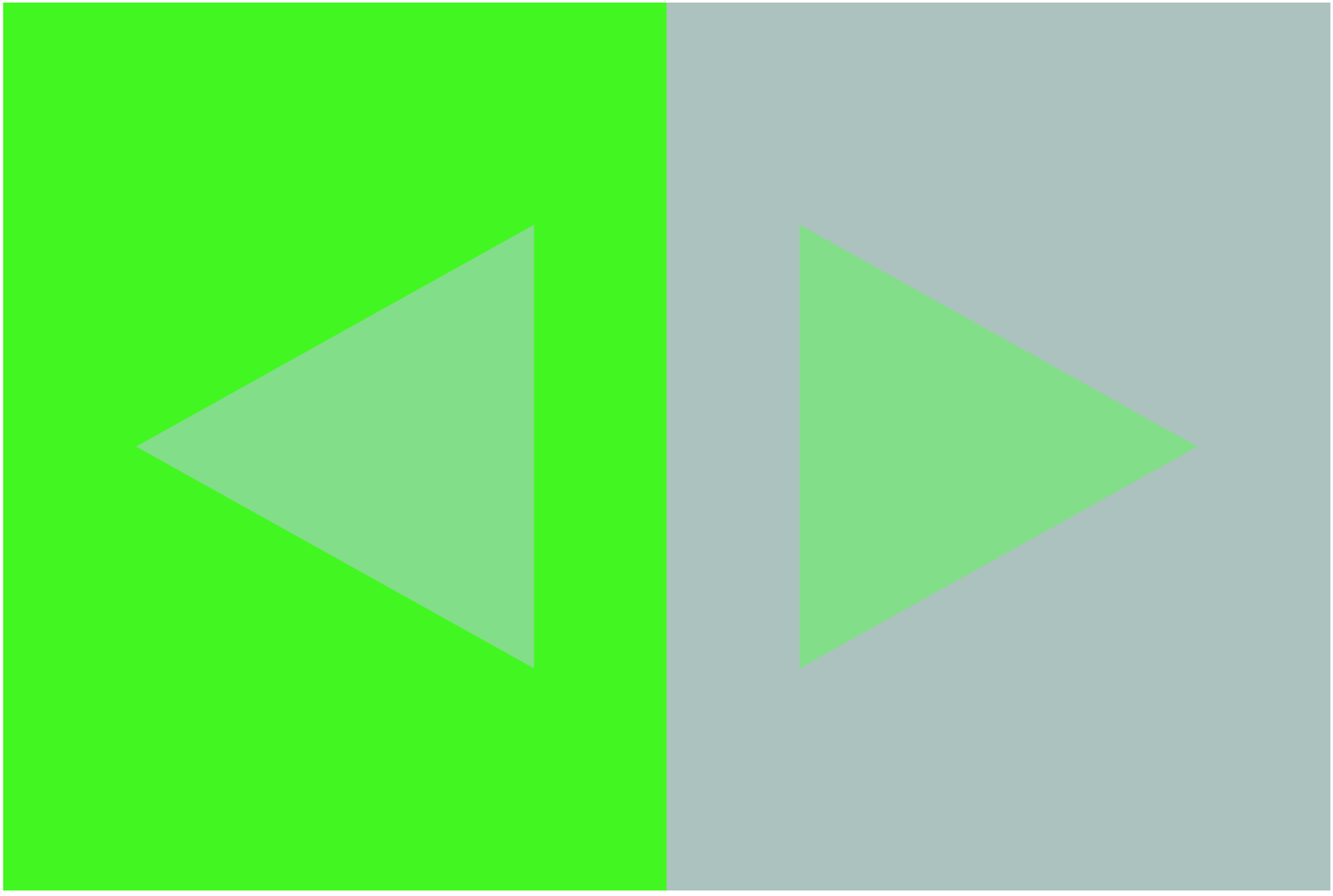
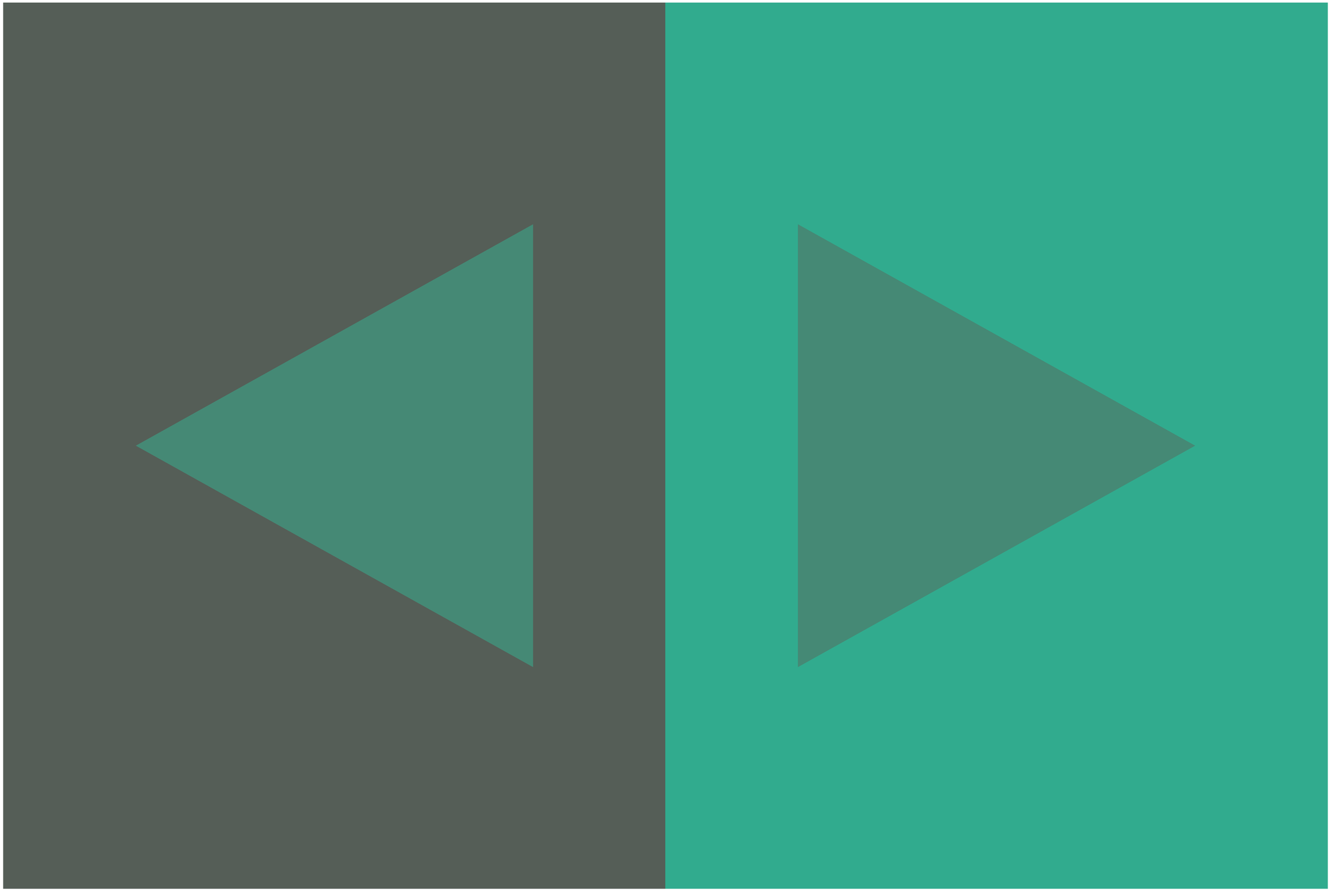
THE COLORIST COOKBOOK

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

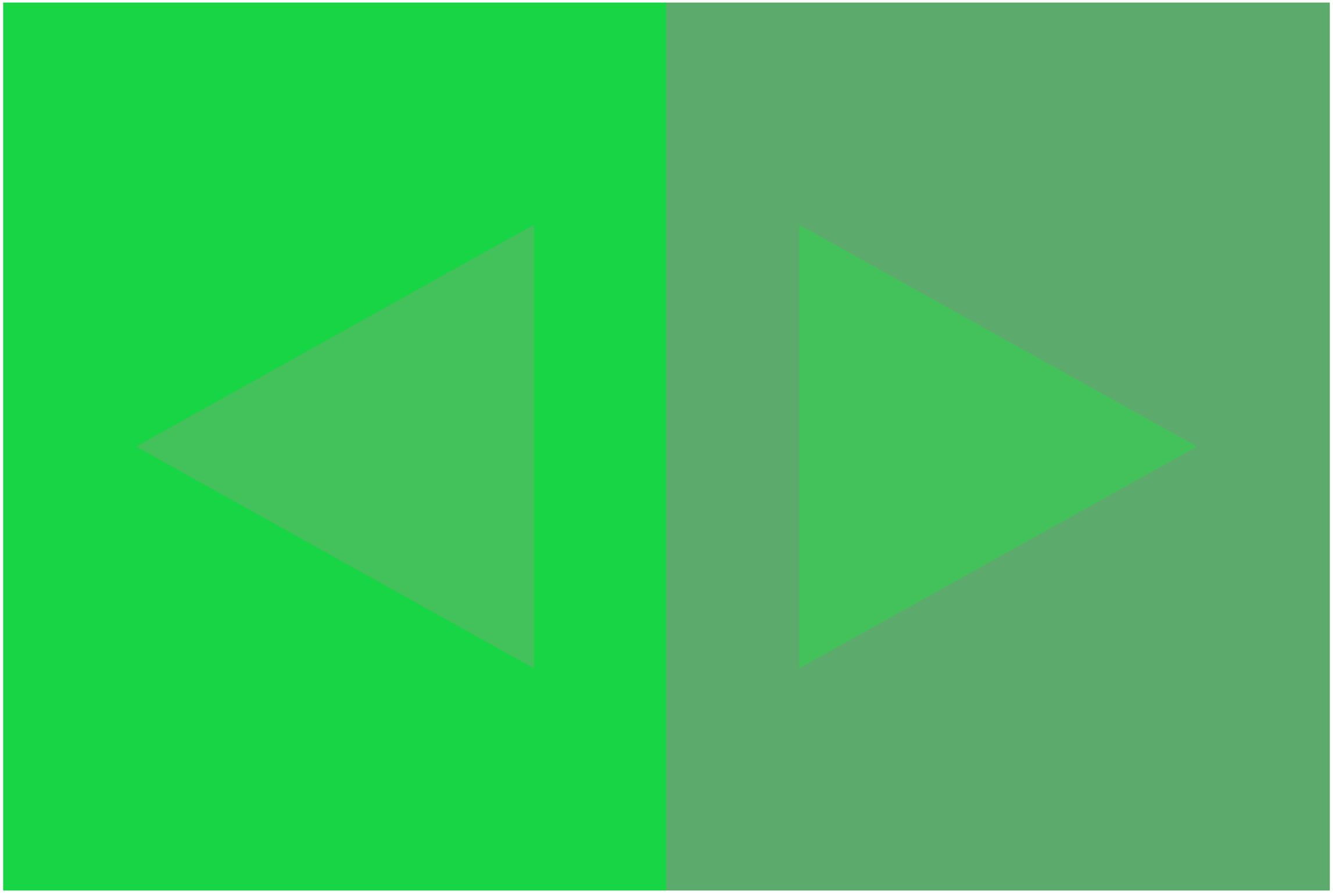
0xFF42F721 0xFFACC2BE 0xFF82DE88



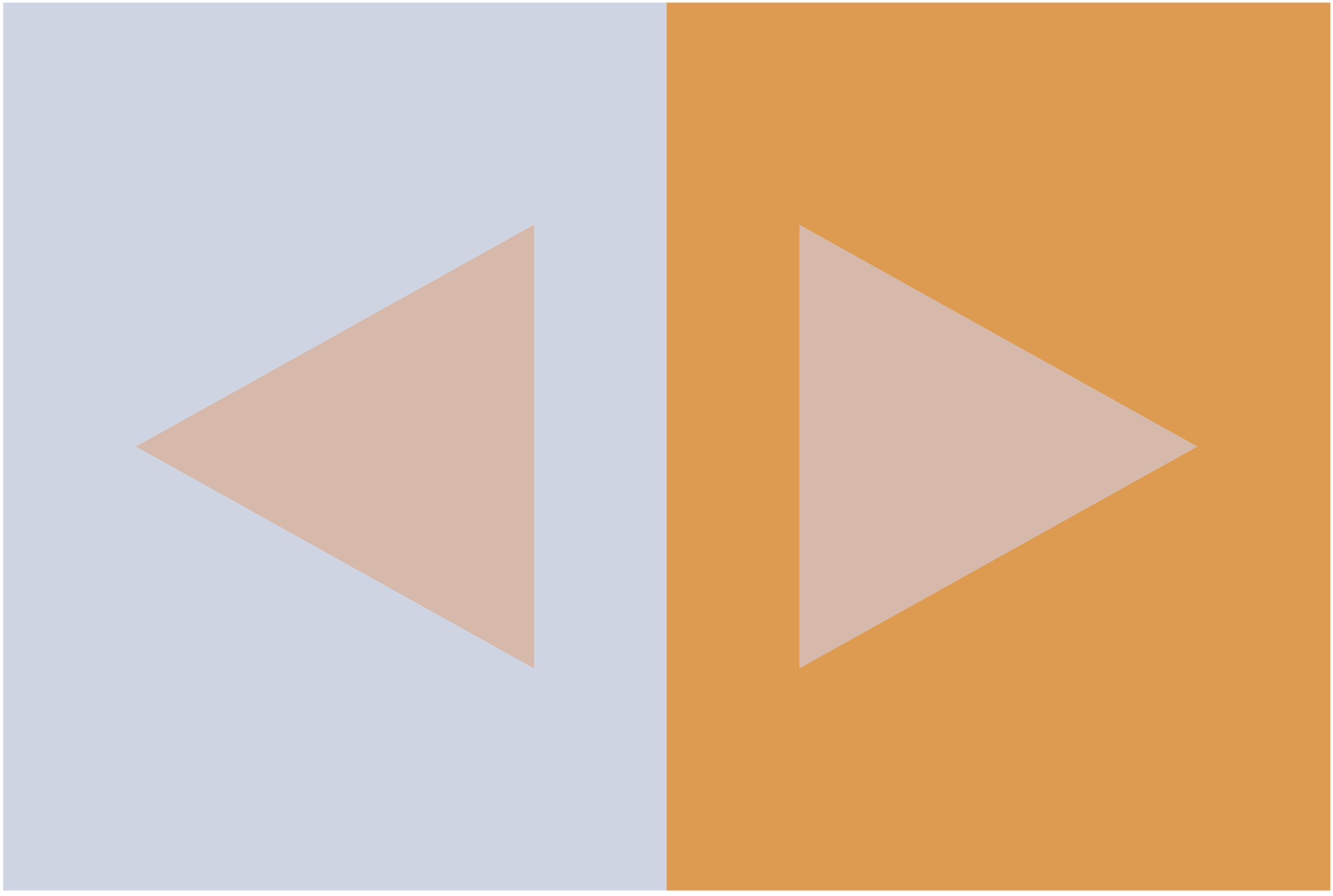
0xFF555E57 0xFF31AB8E 0xFF458975



0xFF17D544 0xFF5DAB6C 0xFF43C15A



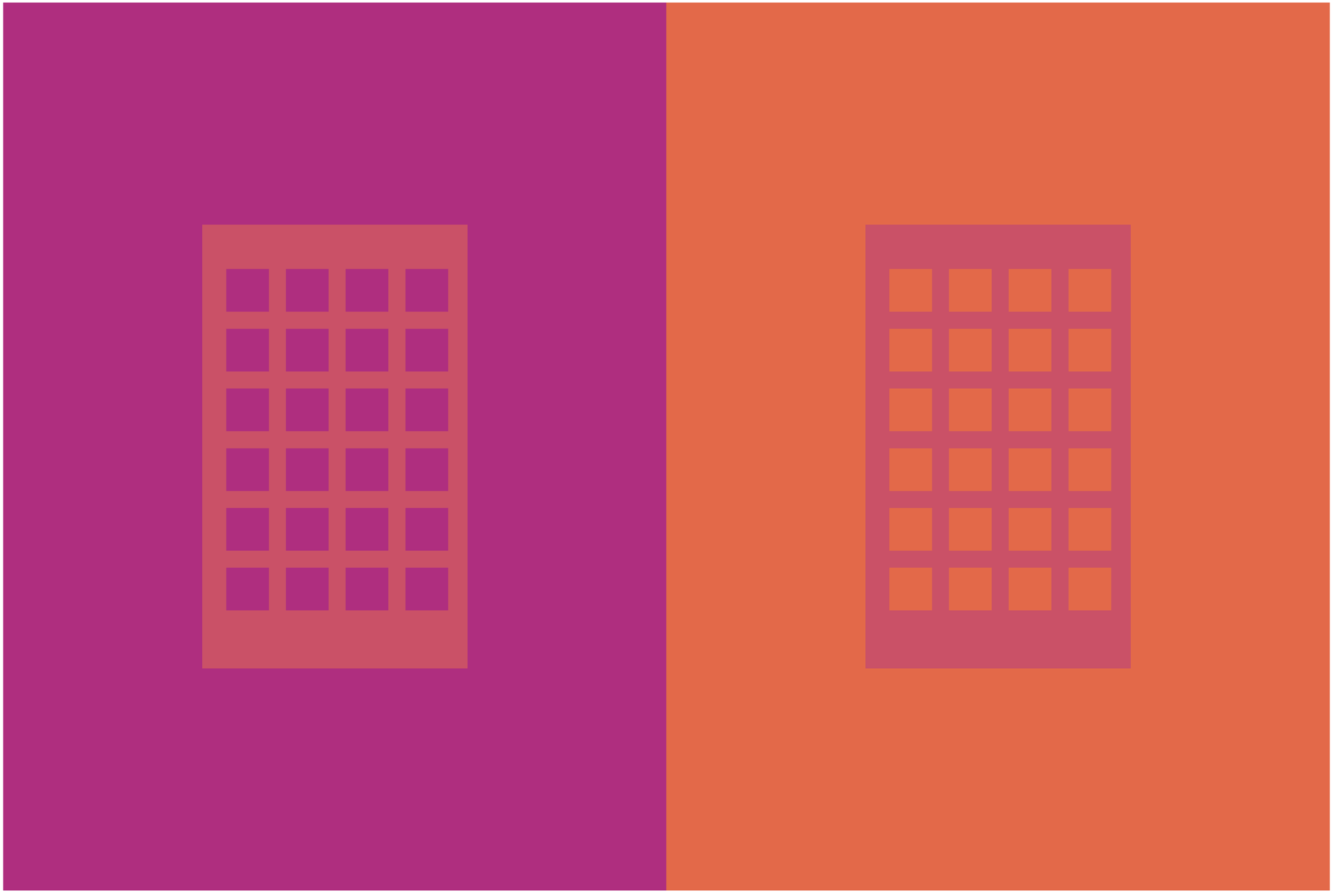
0xFFCFD4E3 0xFFDD9B51 0xFFD6B9AA



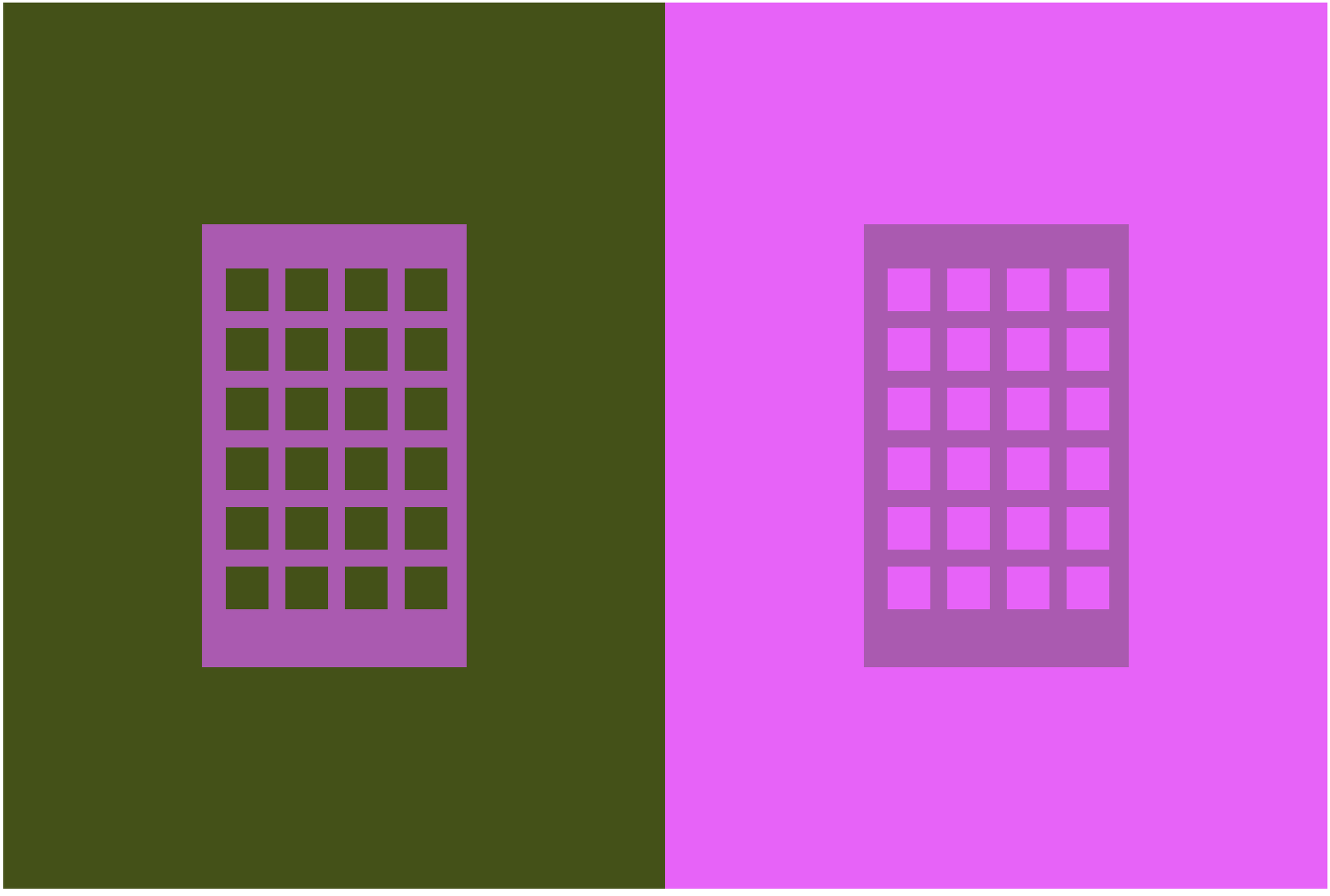
```
// recipe for simple simultaneous contrast

// 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();
```

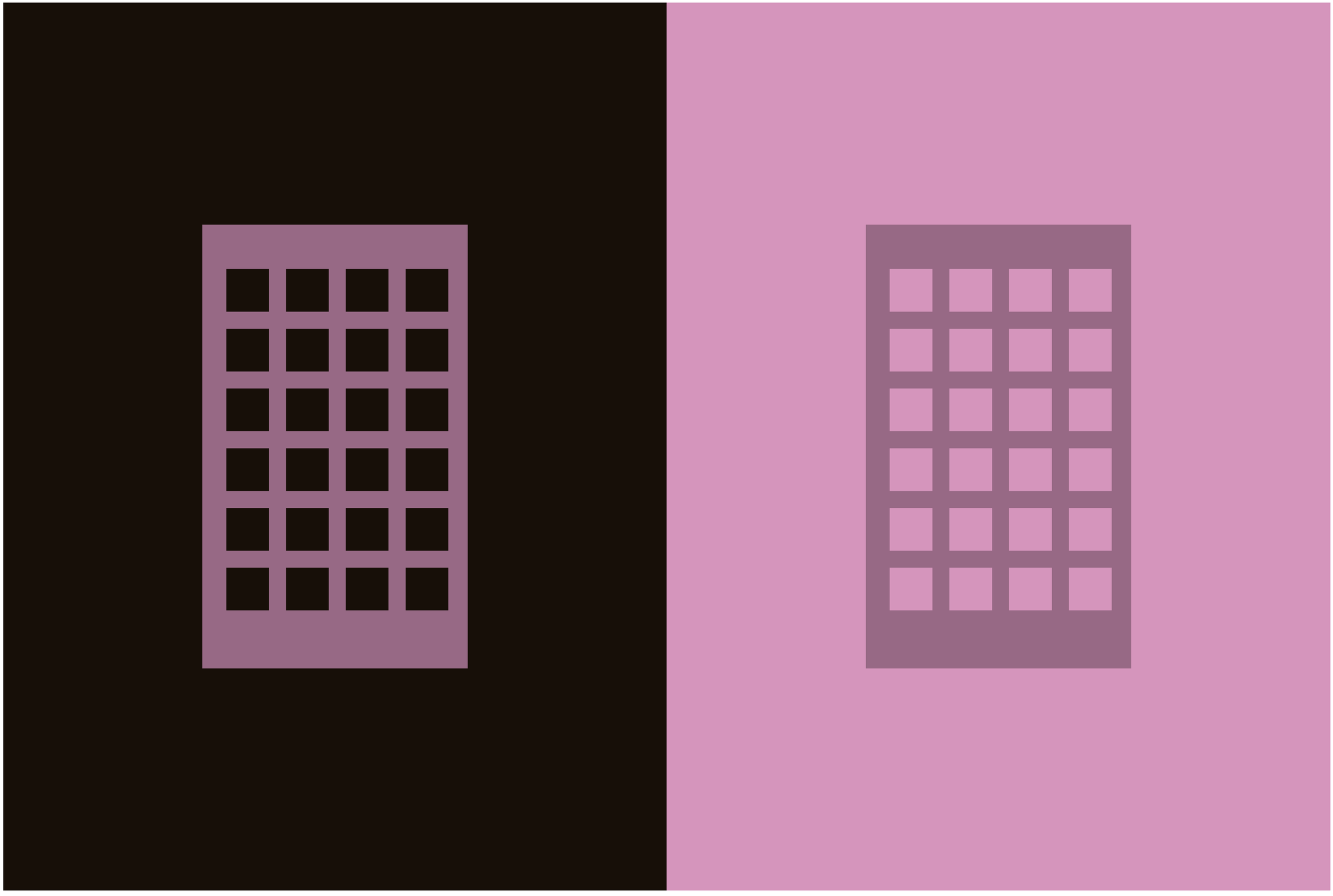
0xFFAF2E7F 0xFFE36949 0xFFCA5167



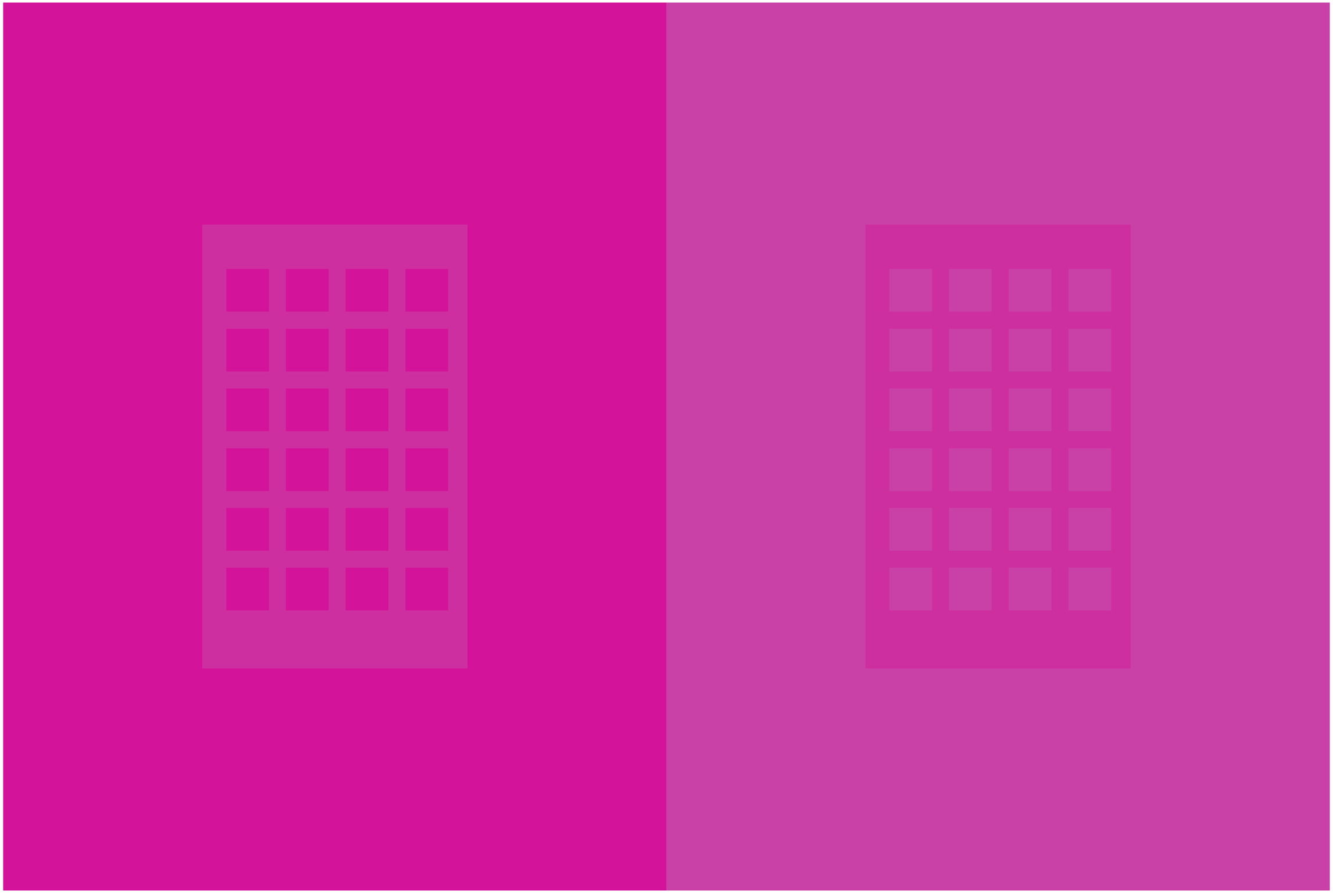
0xFF445118 0xFFE763F8 0xFFAA5AB0



0xFF170F08 0xFFD595BC 0xFF976985



0xFFD31399 0xFFC941A7 0xFFCE2FA0



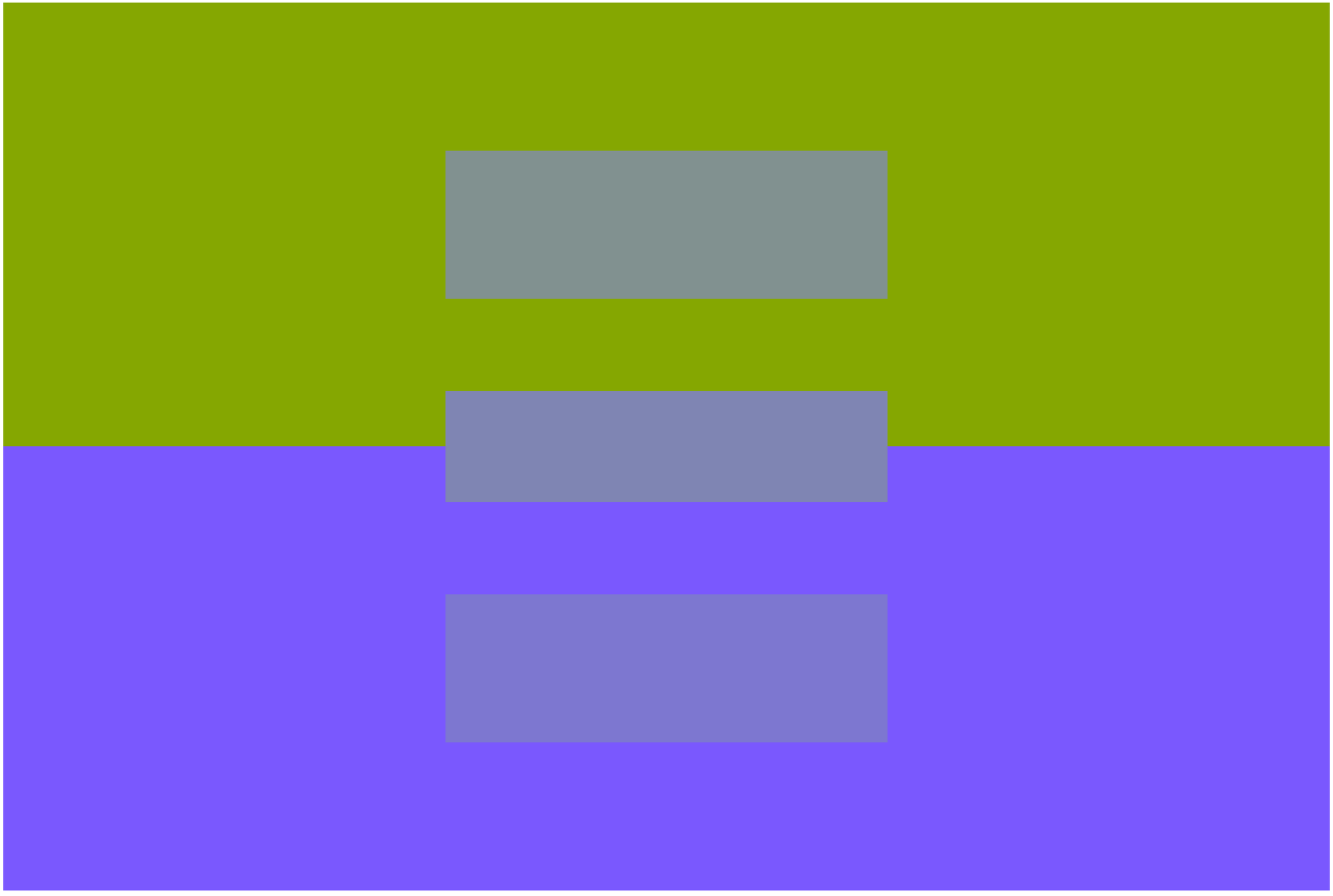
```
// recipe for holed simultaneous contrast

// 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();
```

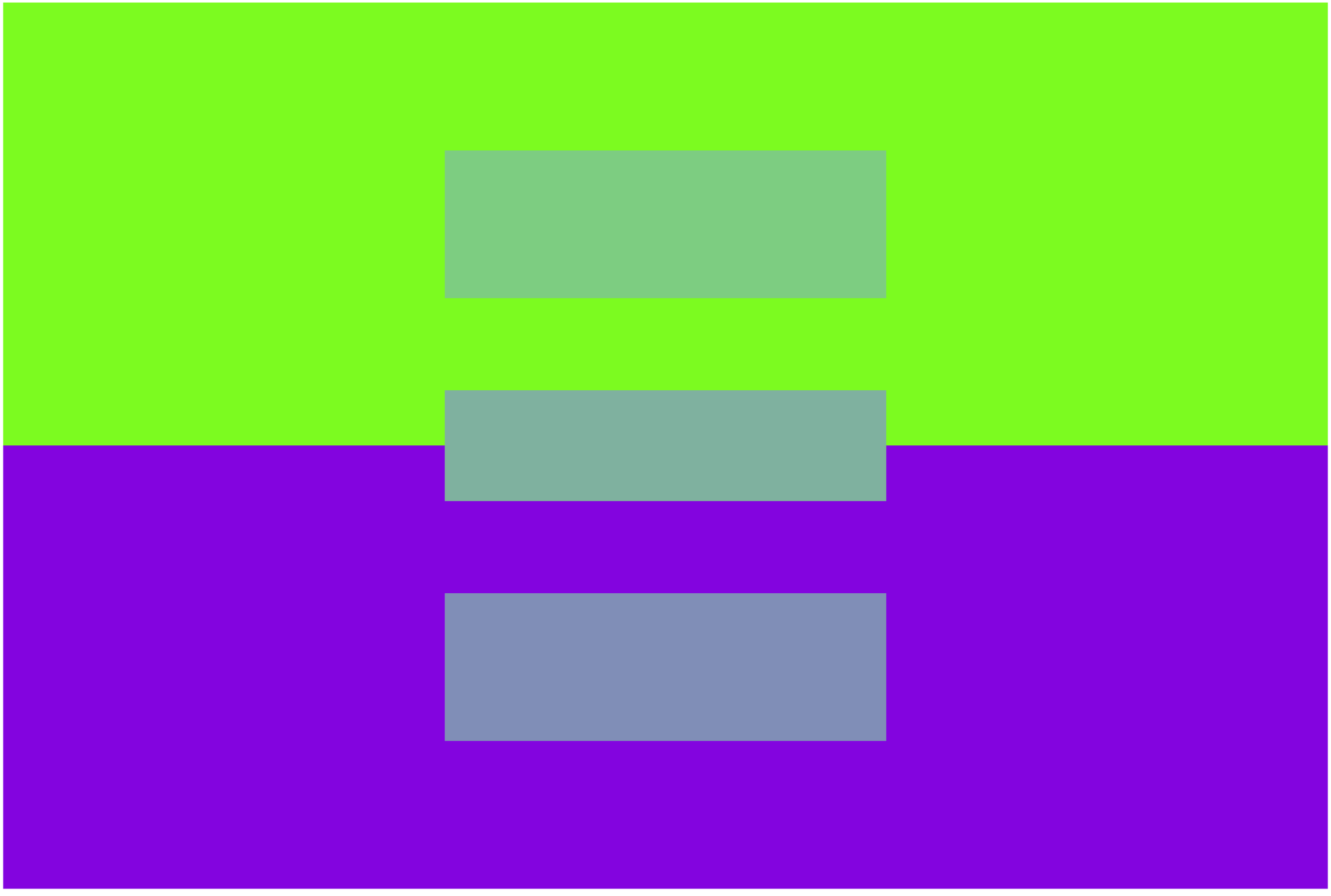
```
  for(int cols = 0; cols < 4; cols++) {
    rect(pgwidth * 0.668 + 140*cols, pgheight * 0.3 + 140*rows,100,100);
  }
}
popMatrix();

popMatrix();
```

0xFF92A07C 0xFF6D5F83 0xFF868D7D 0xFF80837F 0xFF797780

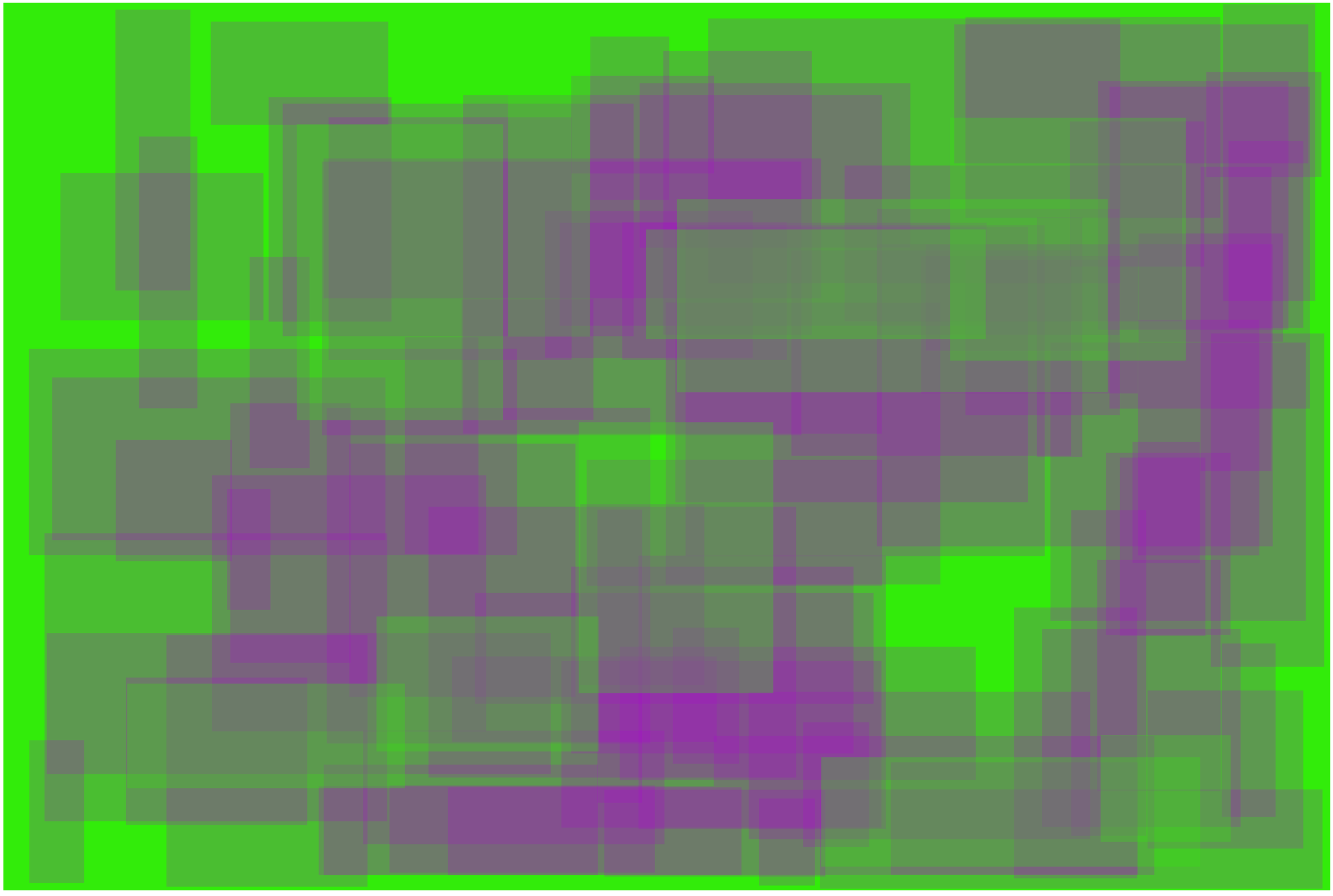
0xFF85A701 0xFF7A58FE 0xFF819190 0xFF7F85B3 0xFF7D77D0



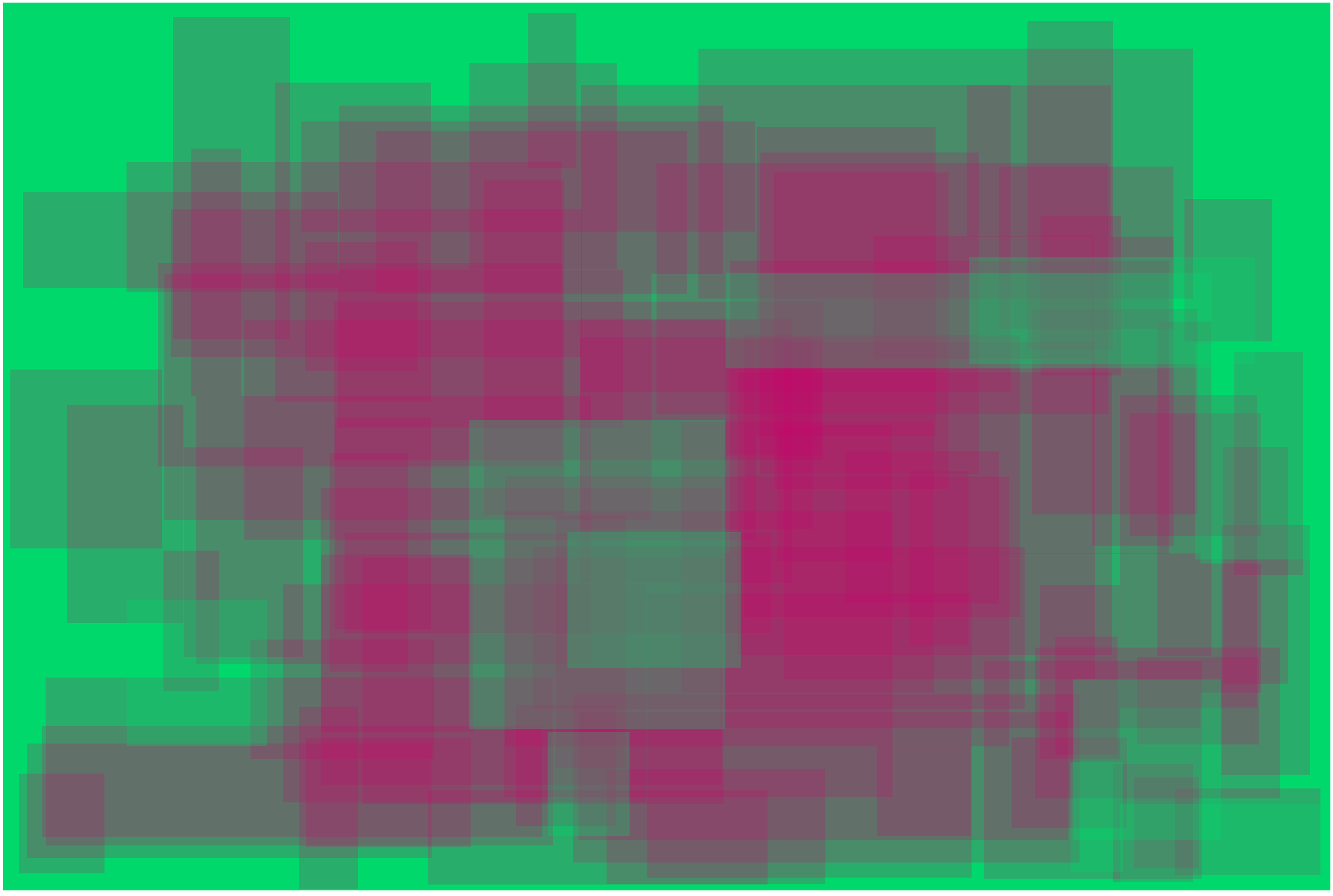
0xFF7CFB20 0xFF8304DF 0xFF7DCD81 0xFF7FB19F 0xFF808EB7



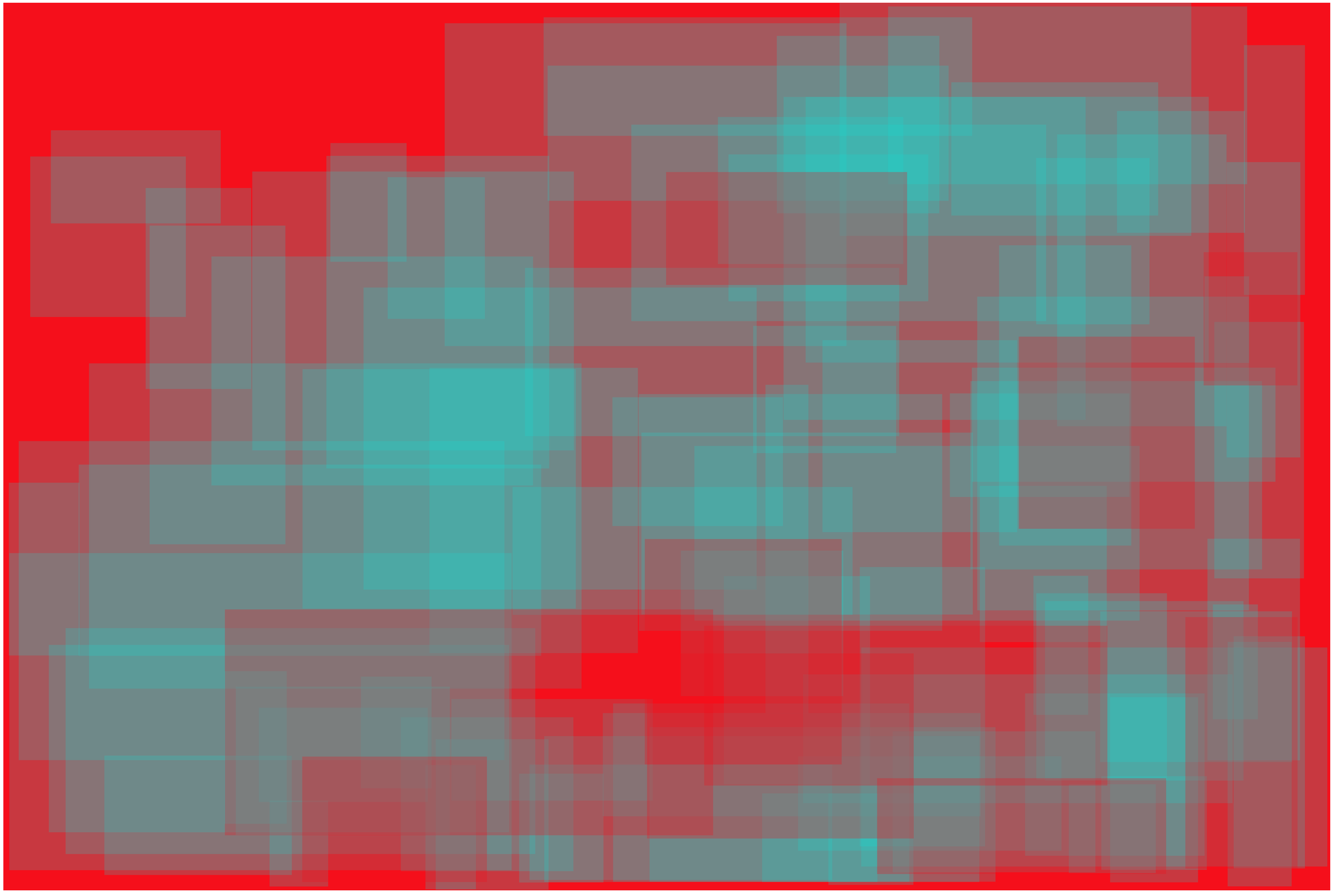
0xFFAF03D4 0xFF32EC0A



0xFFCC0369 0xFF00D86B



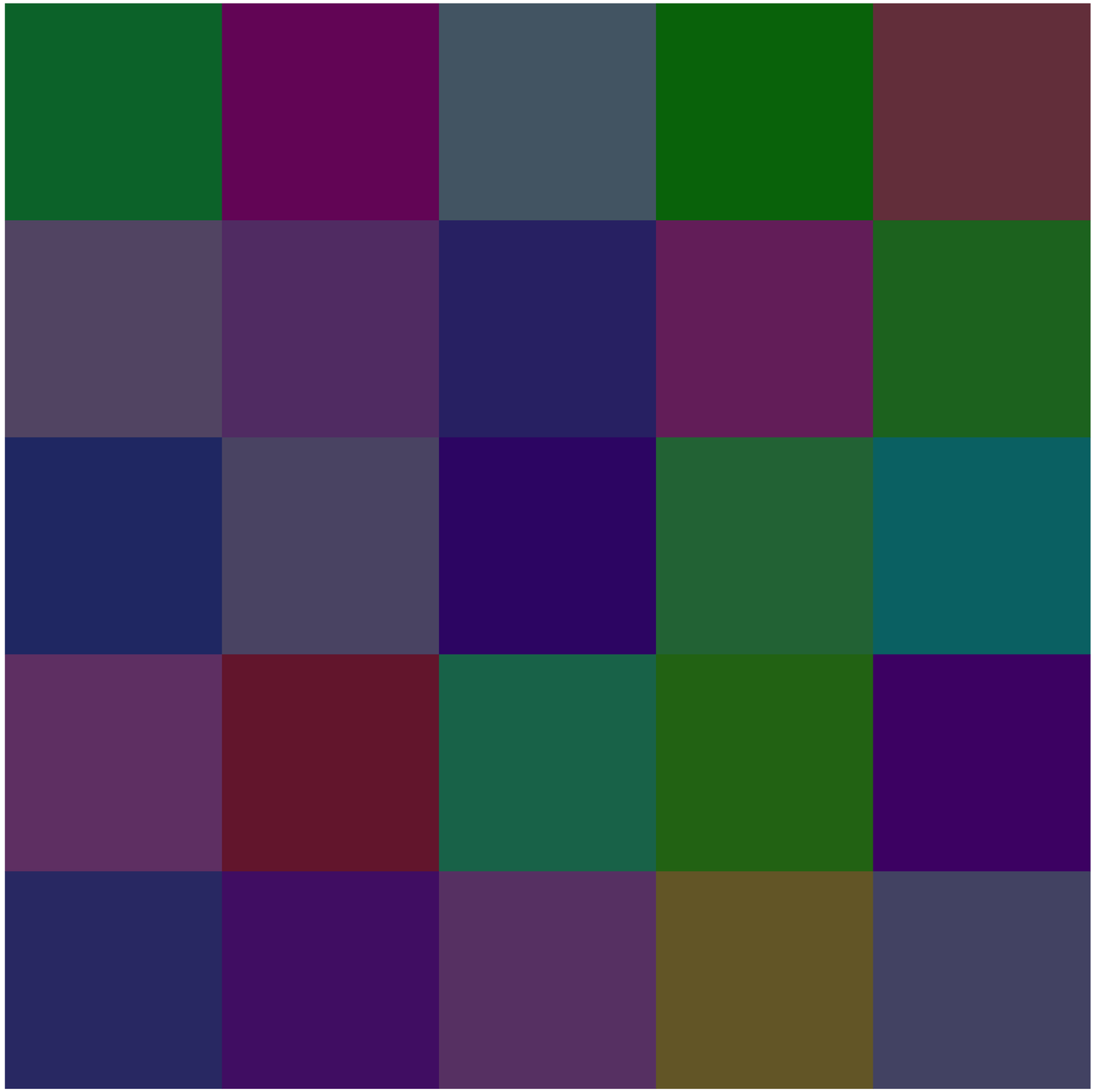
0xFF11E4D9 0xFFF50F1B



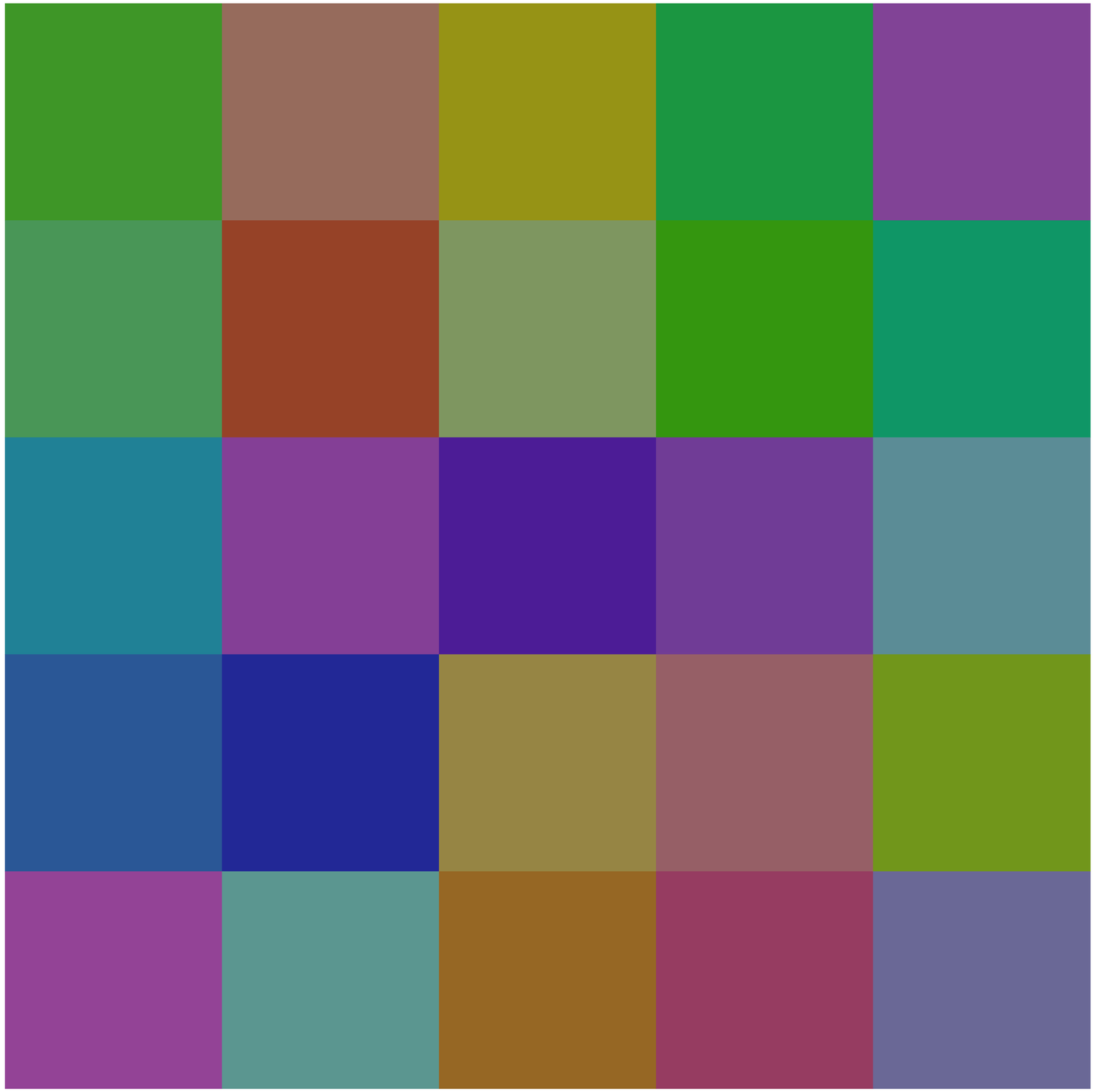
```
// recipe for using bridging colors

// 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();
```

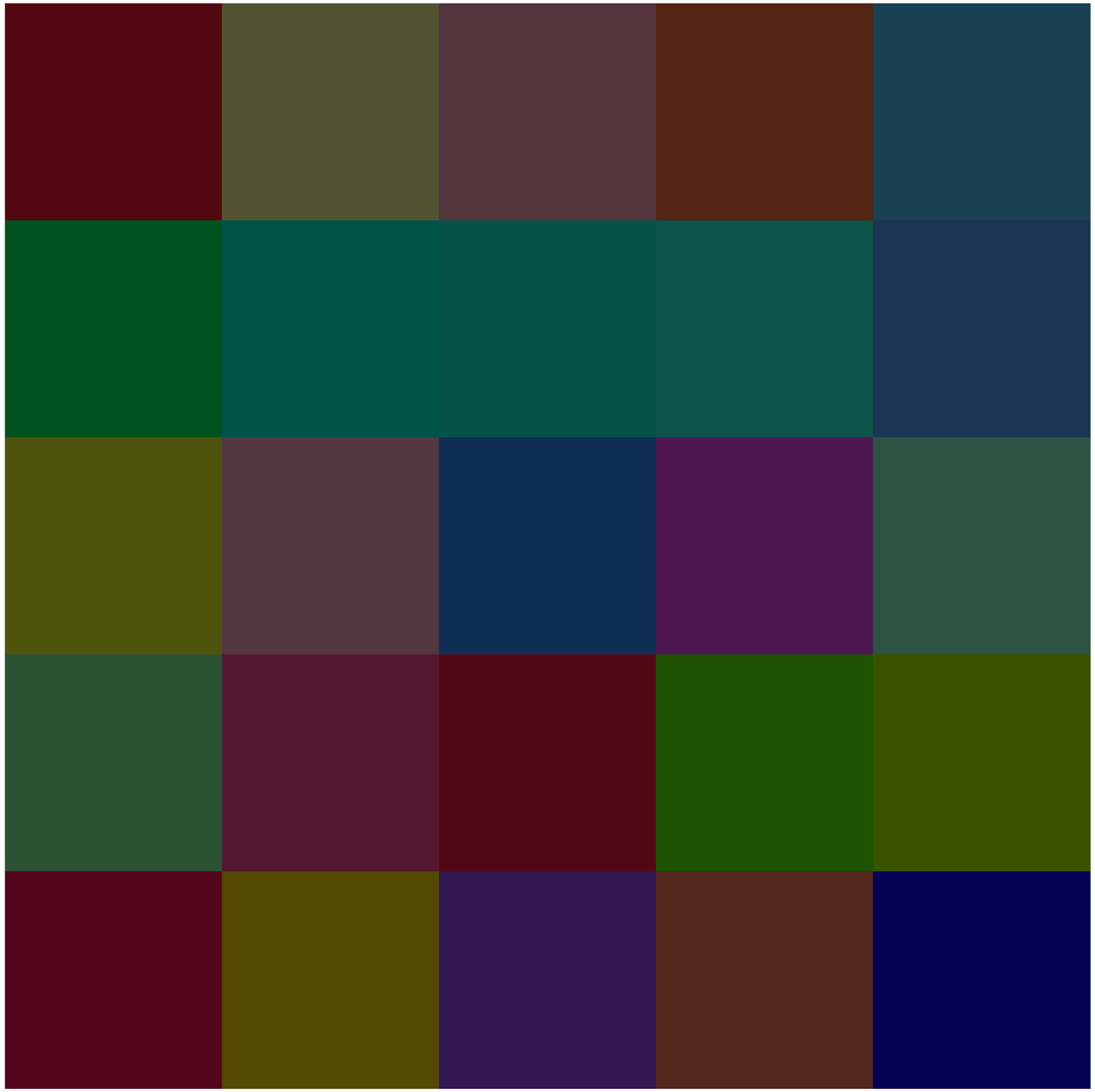
value: 38.636837



value: 59.013824



value: 32.883232



artist:
Sara Cwynar

0xFFF4AC9E
0xFFD0725F
0xFFB84C3B
0xFF8D3526
0xFFED7964
0xFFD14C3D

artist:
Paul Klee

0xFF724F33
0xFF53442B
0xFF383923
0xFFAA241F
0xFFC33919
0xFFCE8104
0xFF8E857A
0xFFD50E23

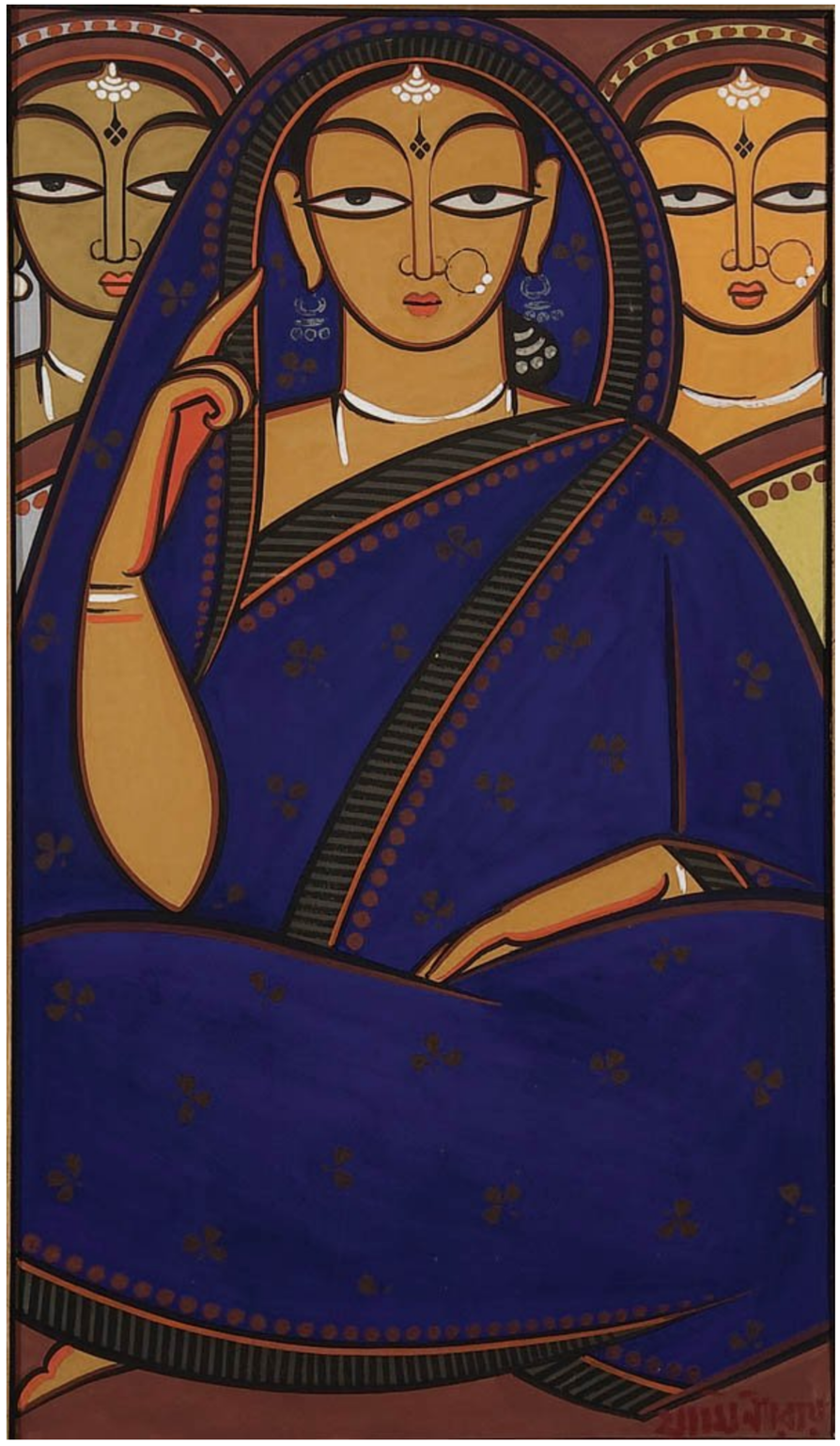
artist:
Shintaro Ohata

0xFF9F9893
0xFFD7D3CA
0xFF867C79
0xFFE7E2D8
0xFF5D6AA3
0xFF3A4083
0xFF314DA4
0xFF212B6D
0xFF859FD7
0xFFA1CAF4
0xFF6E5D48

artist:
Alphonse Mucha

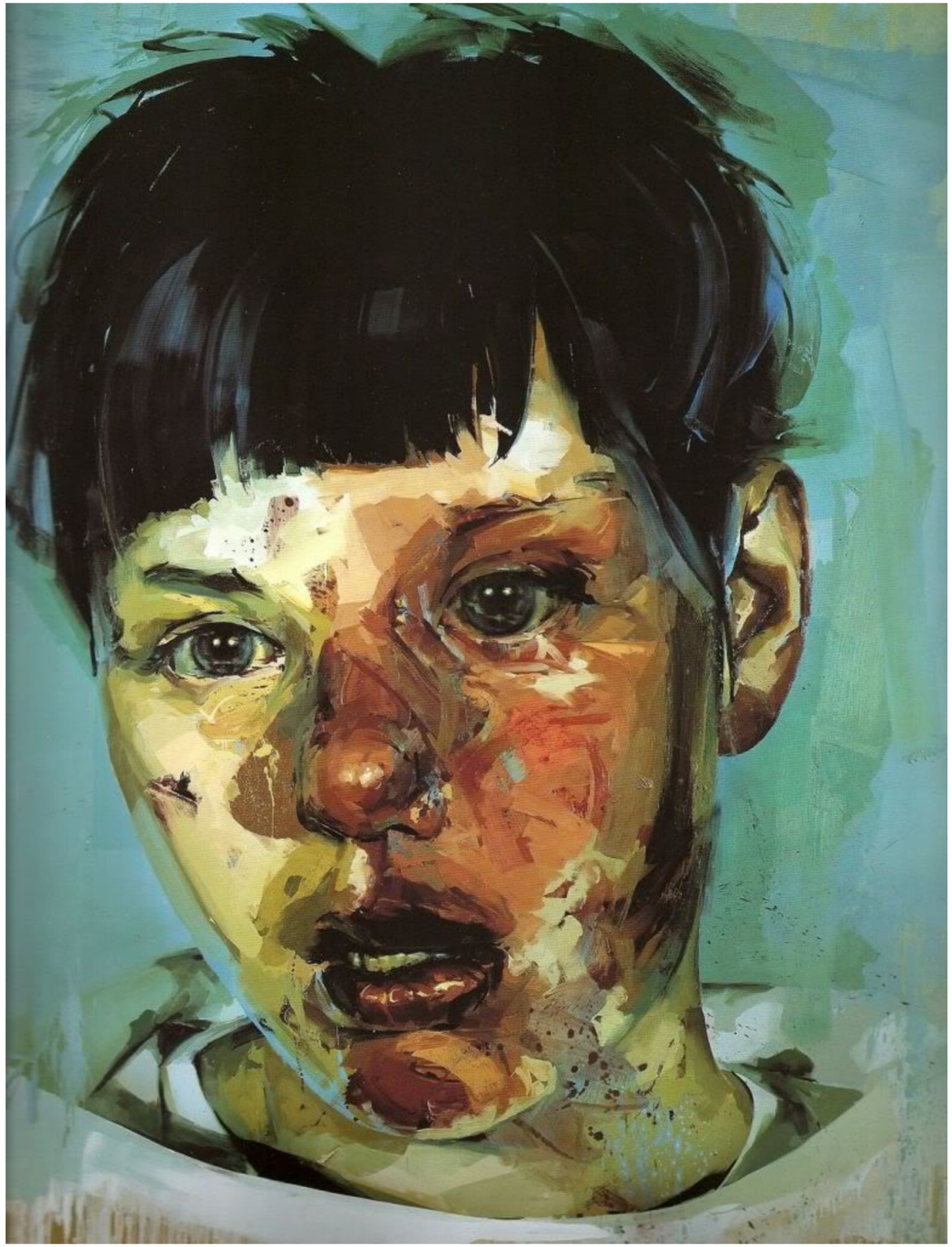
0xFFEAC6A9
0xFFCE996F
0xFFB16E44
0xFF6F2A0D
0xFF40140A
0xFE1A0806
0xFF933812
0xFFB55527
0xFFD17C4A
0xFFE79469

artist:
Jamini Roy



0xFF231D38
0xFF3A282F
0xFF693F30
0xFF936843
0xFFB28149
0xFFCB985A
0xFFBC803E
0xFFD38E43

artist:
Jenny Saville



0xFF546B5D
0xFF242019
0xFF6F9385
0xFF8CA488
0xFF191710
0xFFA77F4C
0xFFCBAE72
0xFFE9DDAA
0xFF805937
0xFF5E402C
0xFFBB7040

```
// 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();

}
```