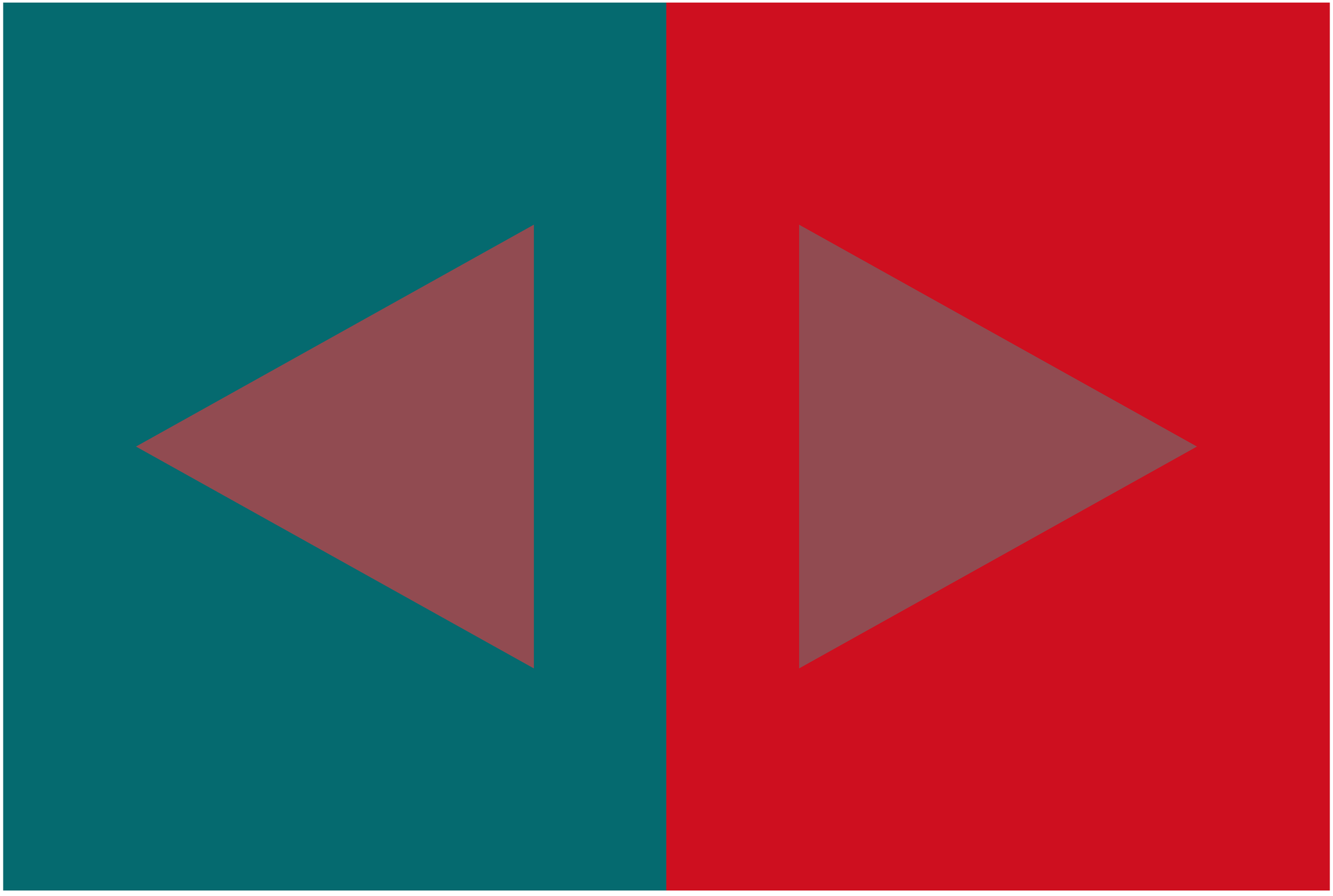
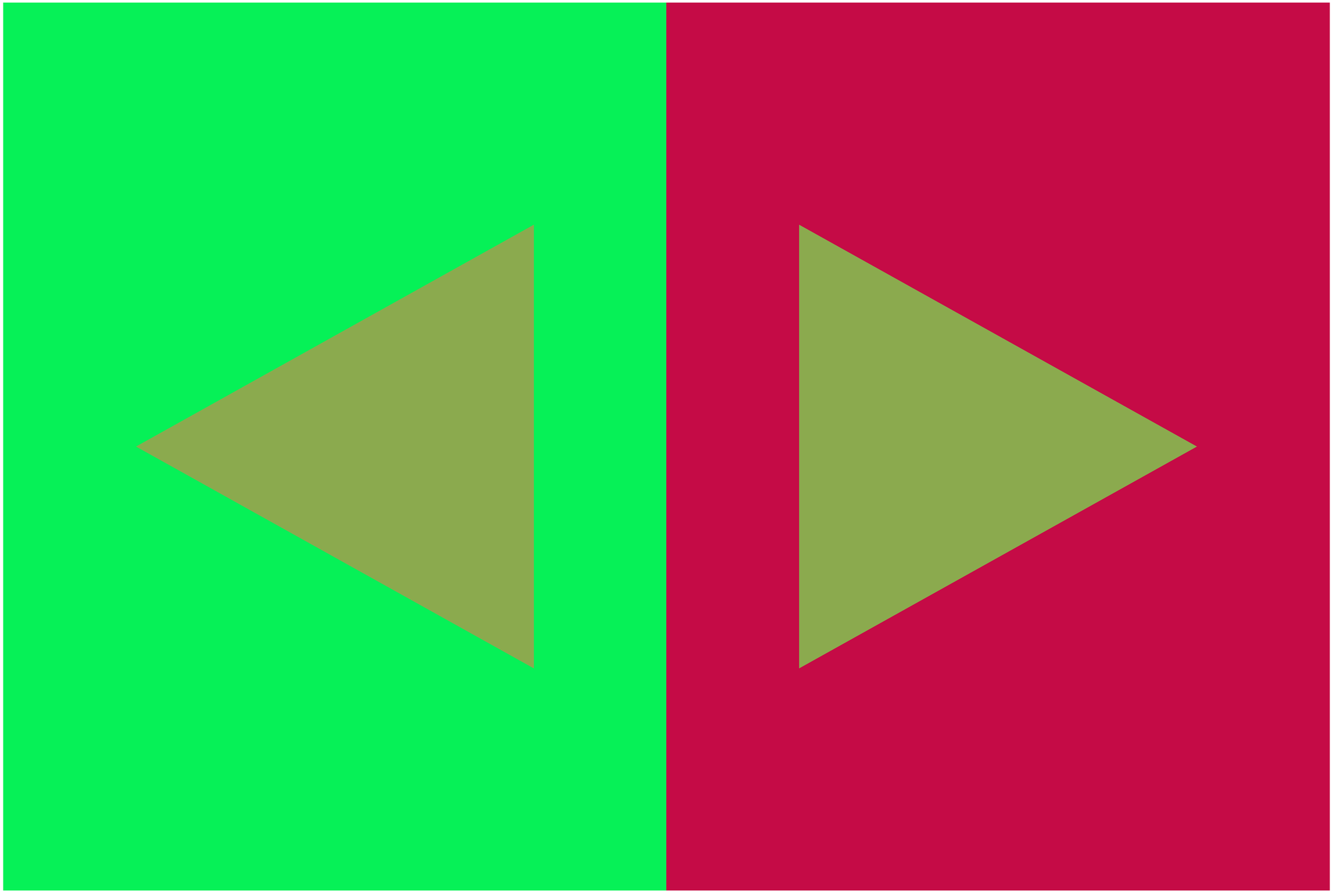
THE COLORIST COOKBOOK

"WOMEN THINK OF ALL COLORS EXCEPT THE ABSENCE OF COLOR. I HAVE SAID THAT BLACK HAS IT ALL. WHITE TOO. THEIR BEAUTY IS ABSOLUTE. IT IS THE PERFECT HARMONY."

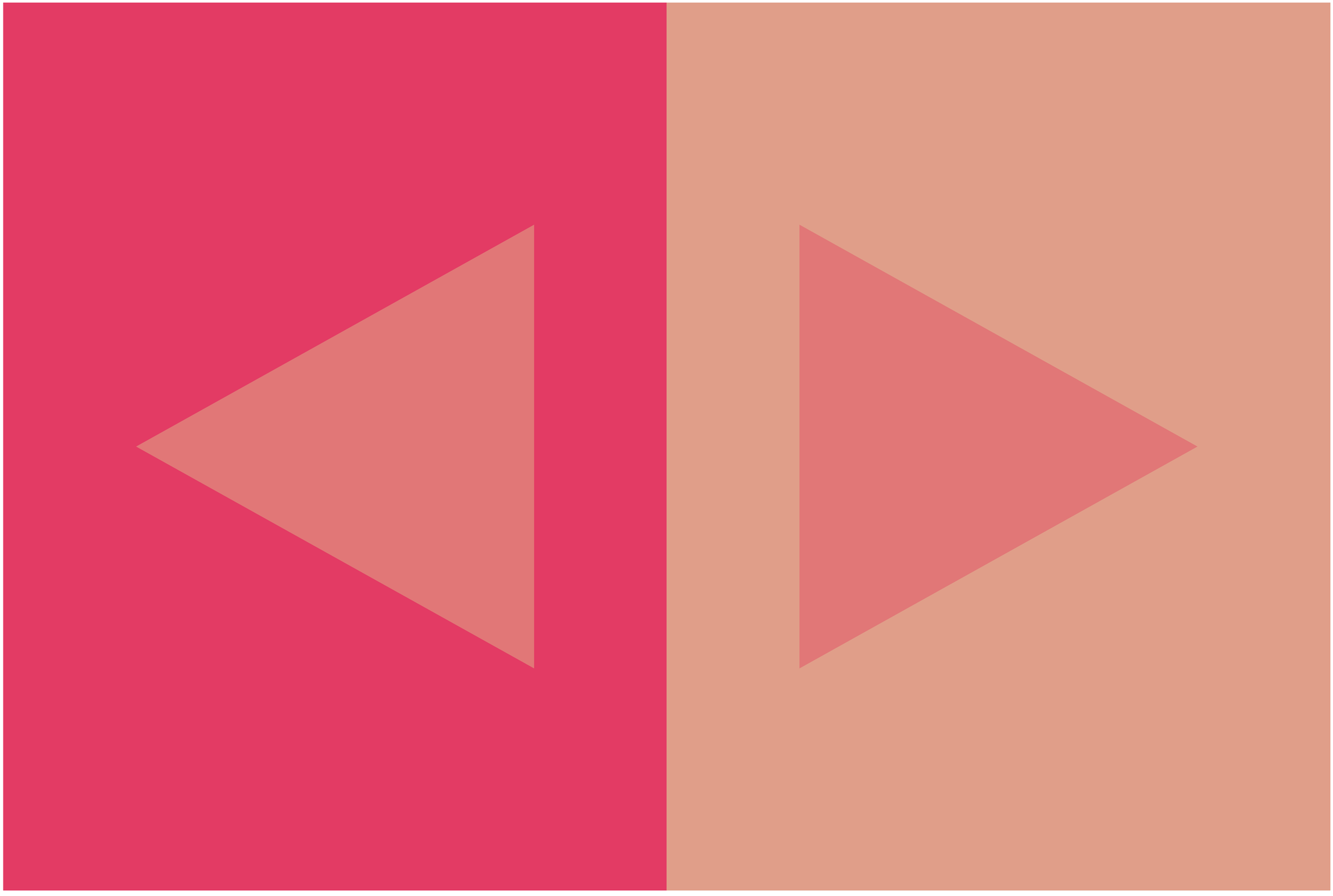
0xFF056A6F 0xFFCE0F1F 0xFF914B51



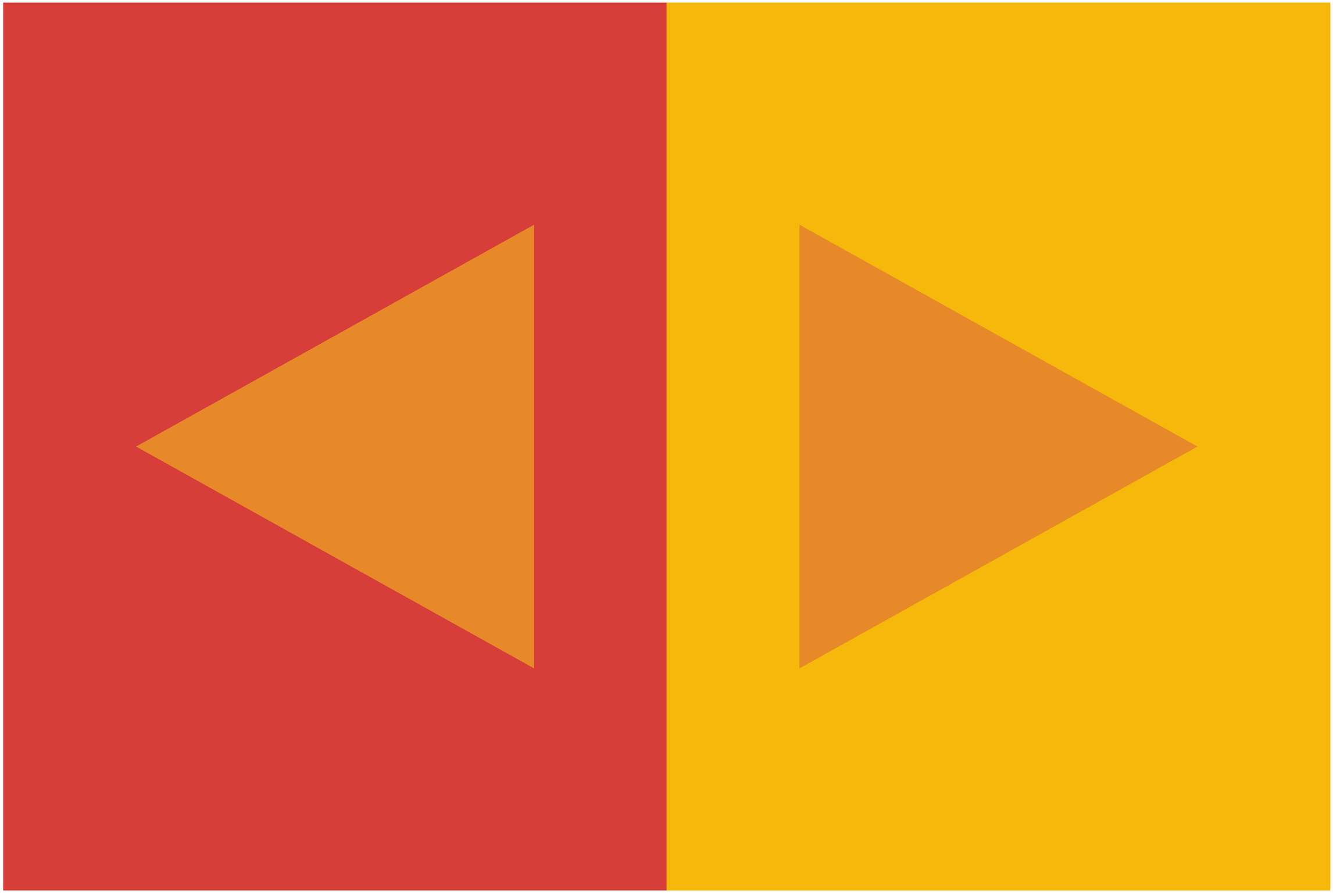
0xFF06F157 0xFFC50B46 0xFF8BAA4E



0xFFE33B64 0xFFE09E89 0xFFE17777



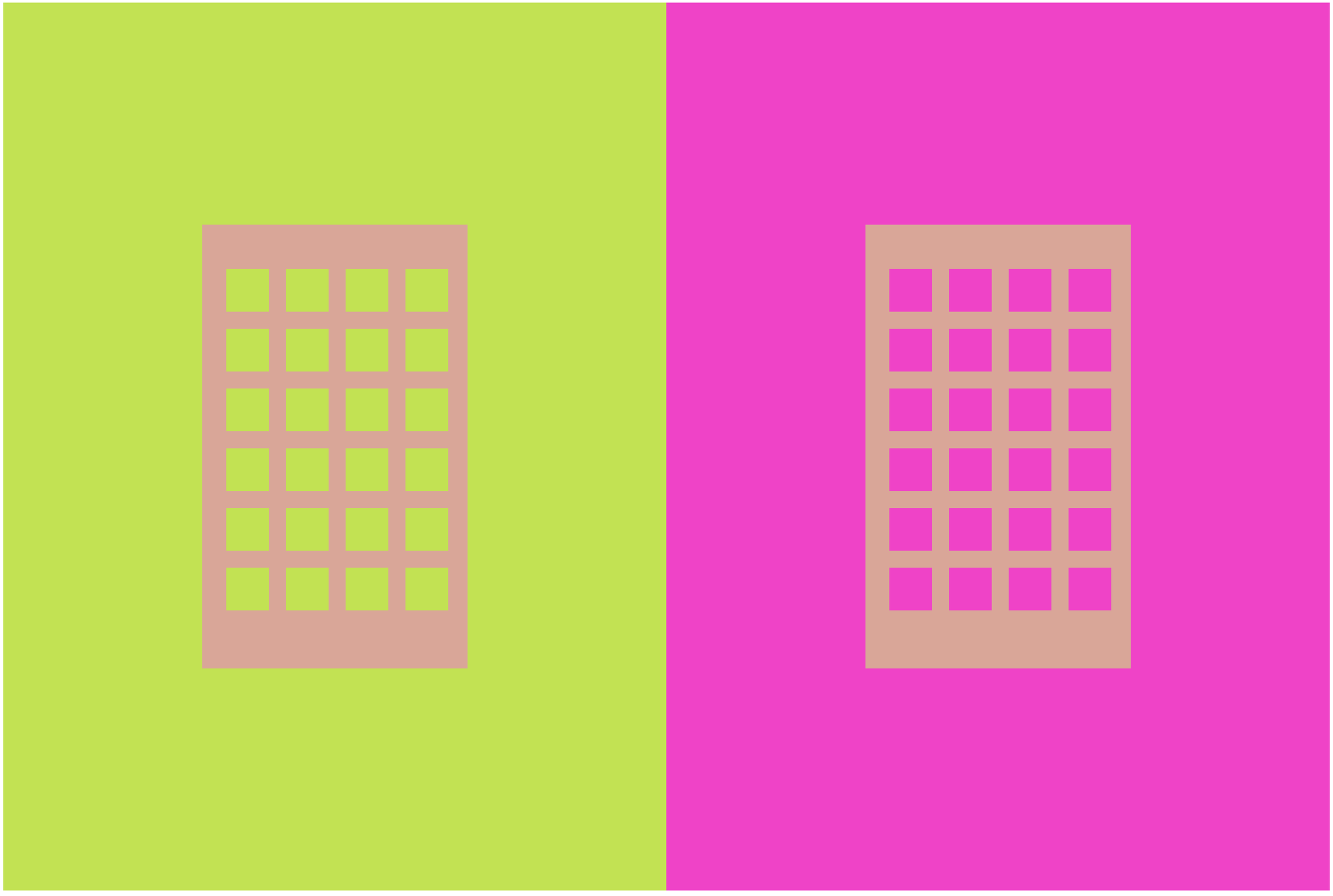
0xFFD73D39 0xFFF6B80A 0xFFE78928



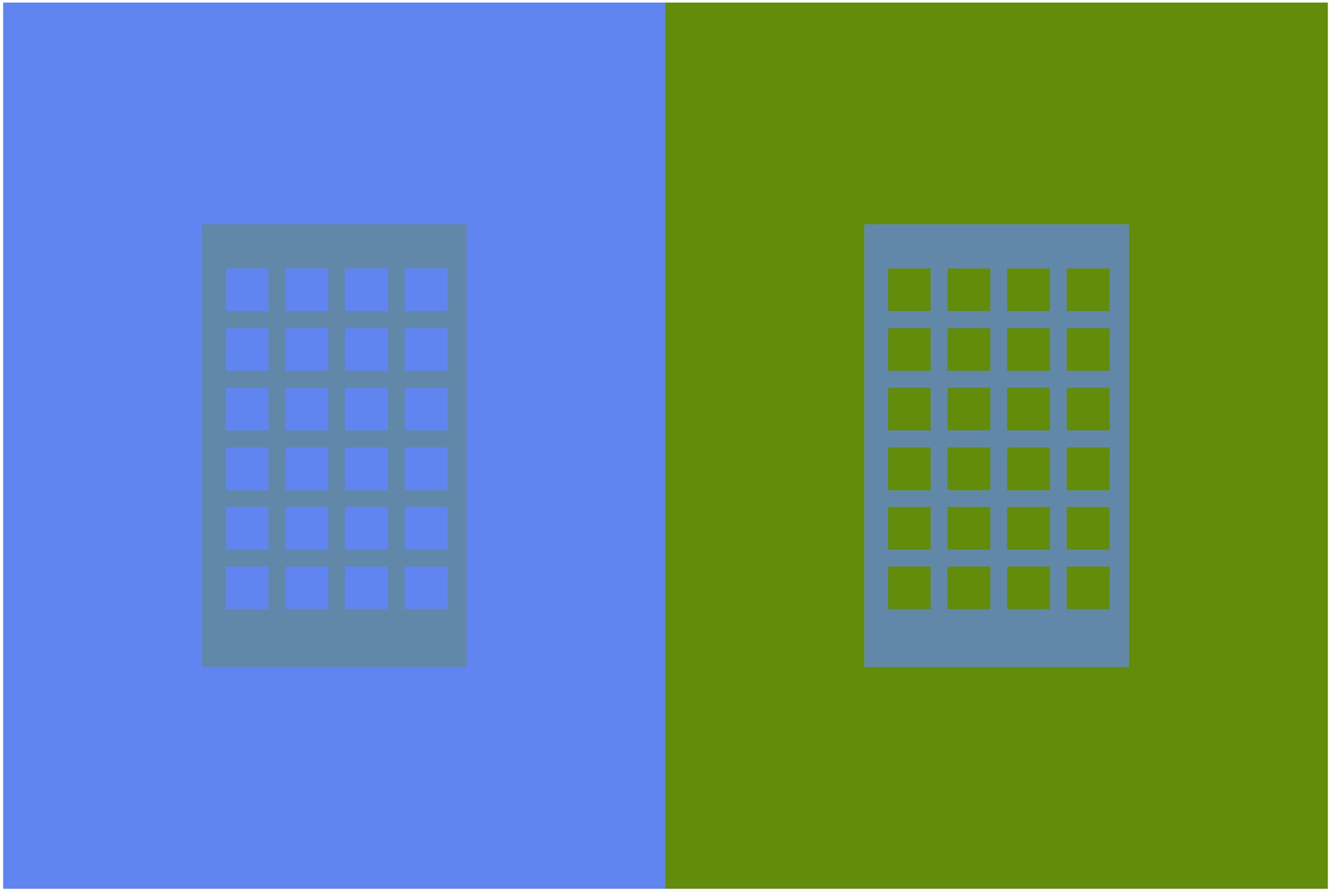
```
// recipe for simple simultaneous contrast

// 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();
```

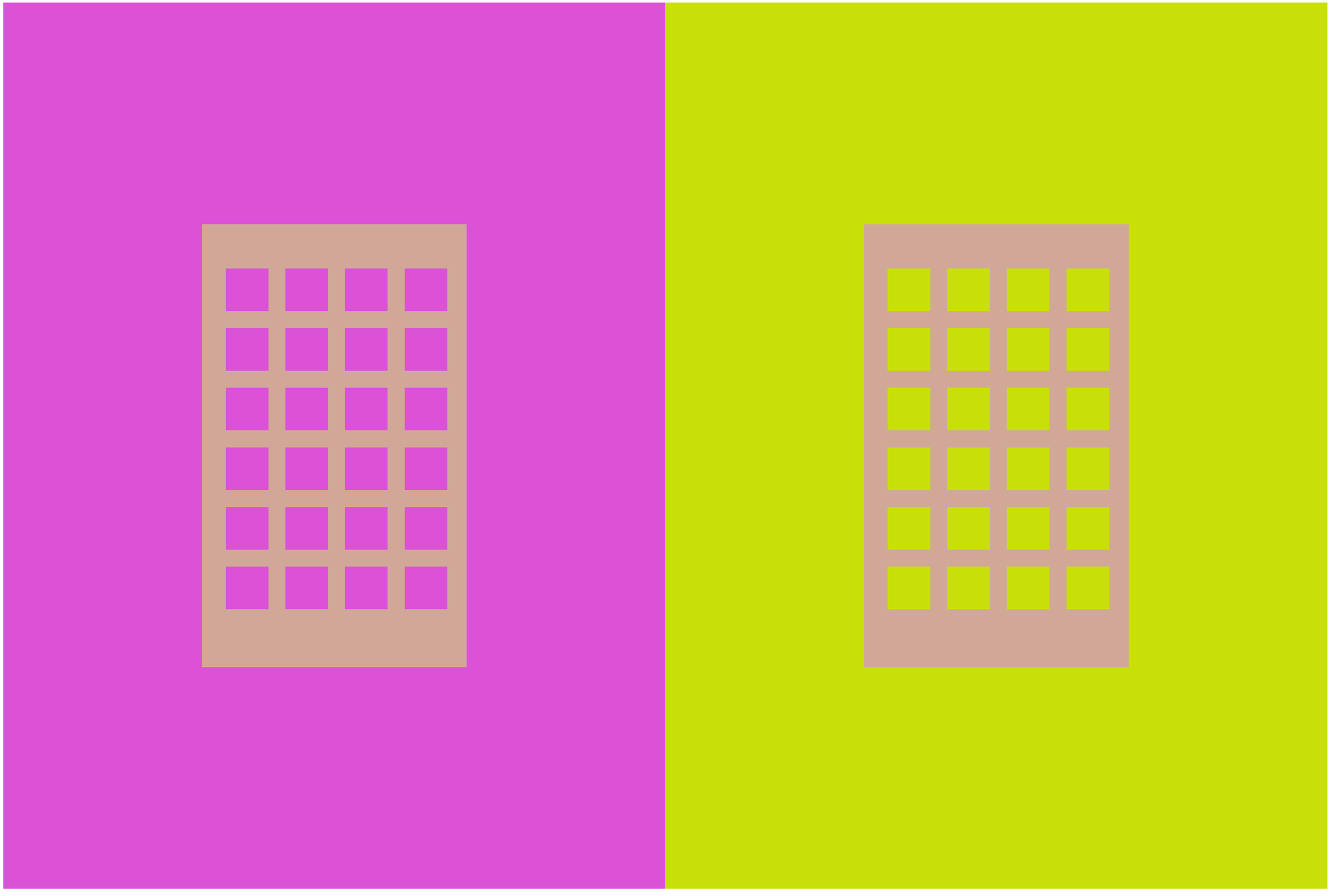
0xFFC2E253 0xFFEF43C7 0xFFD9A698



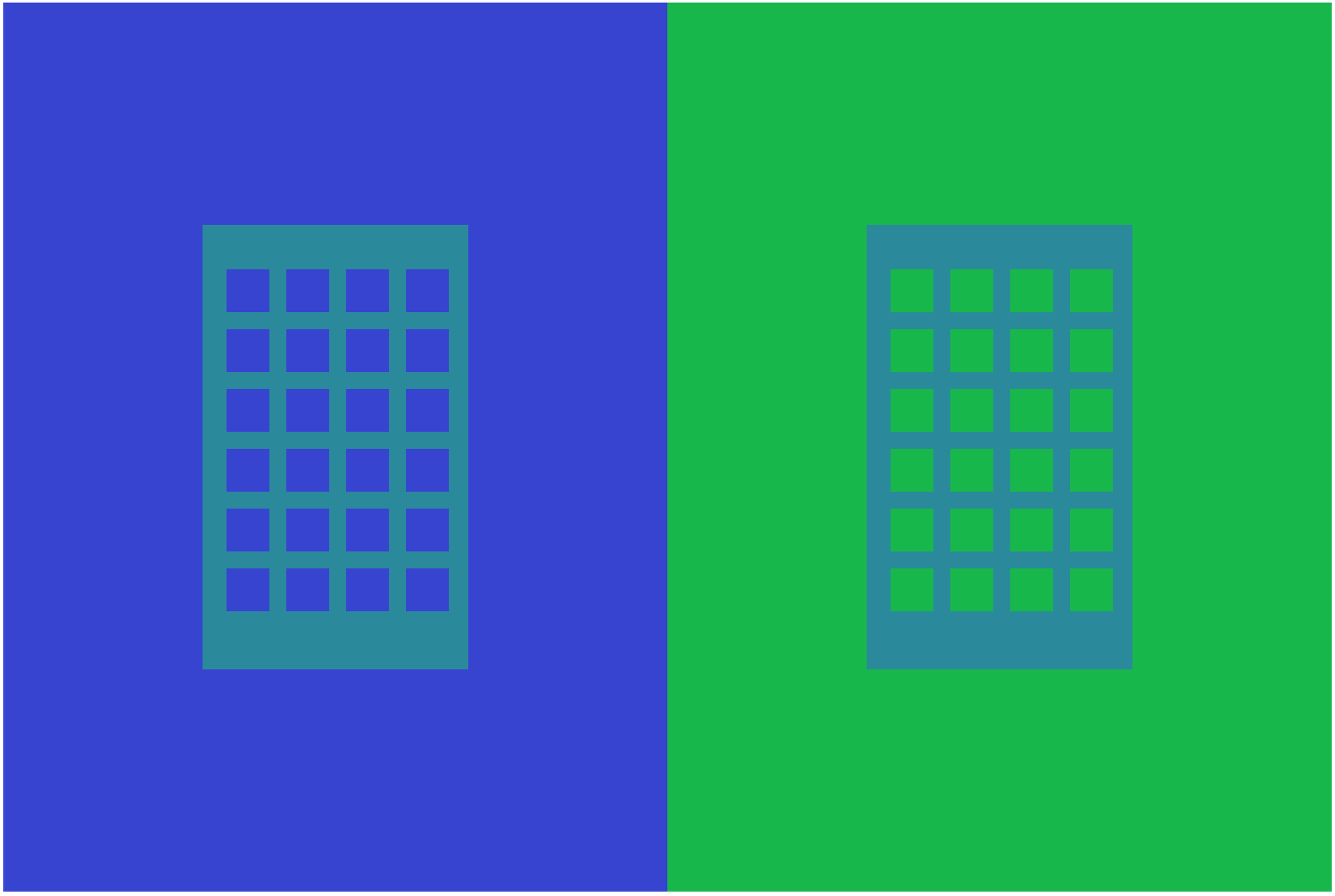
0xFF6085F0 0xFF628C09 0xFF6188A9



0xFFDB52D7 0xFFC8E008 0xFFD1A898



0xFF3744D0 0xFF18B74C 0xFF2A8A9C

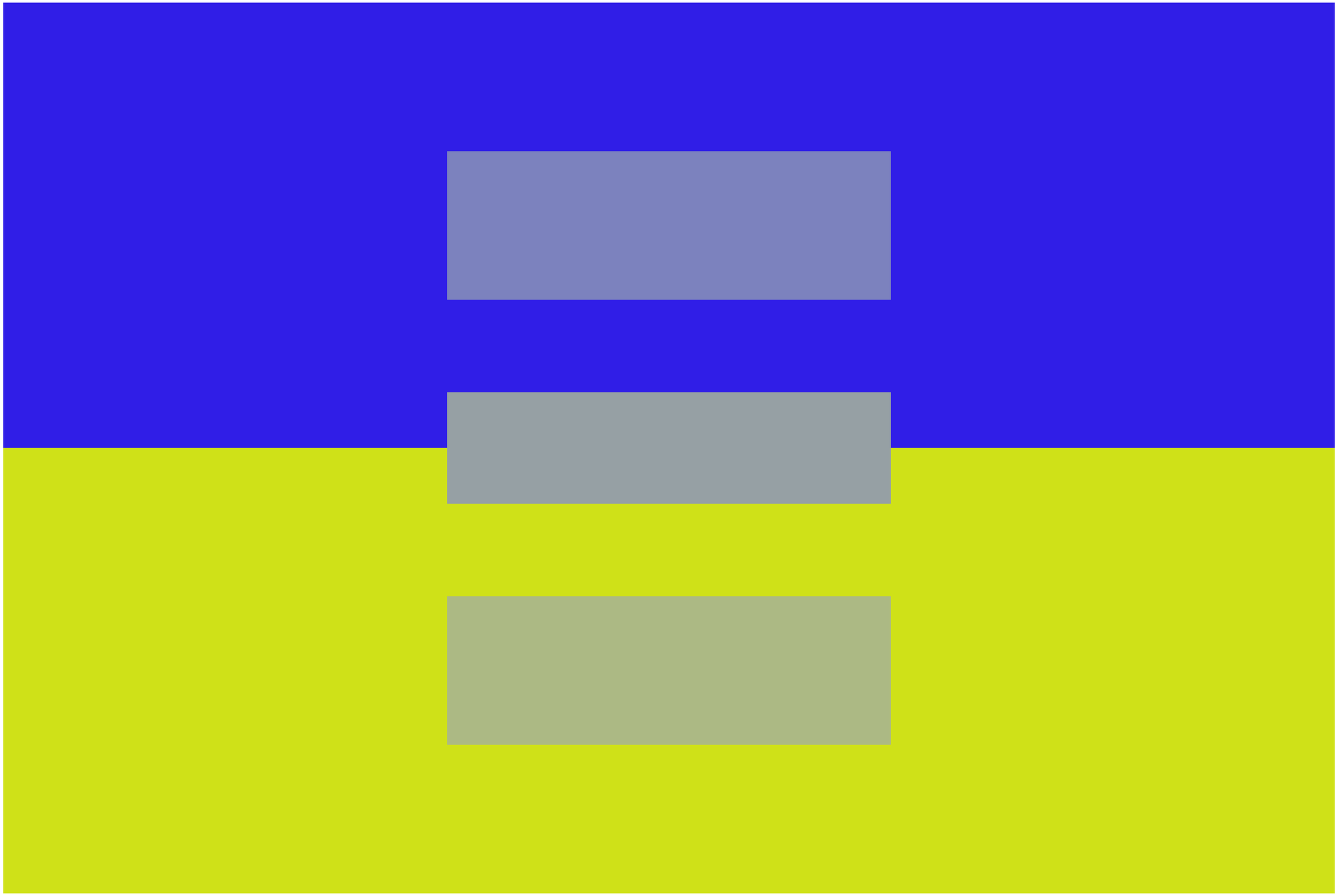


```
// recipe for holed simultaneous contrast

// 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();
```

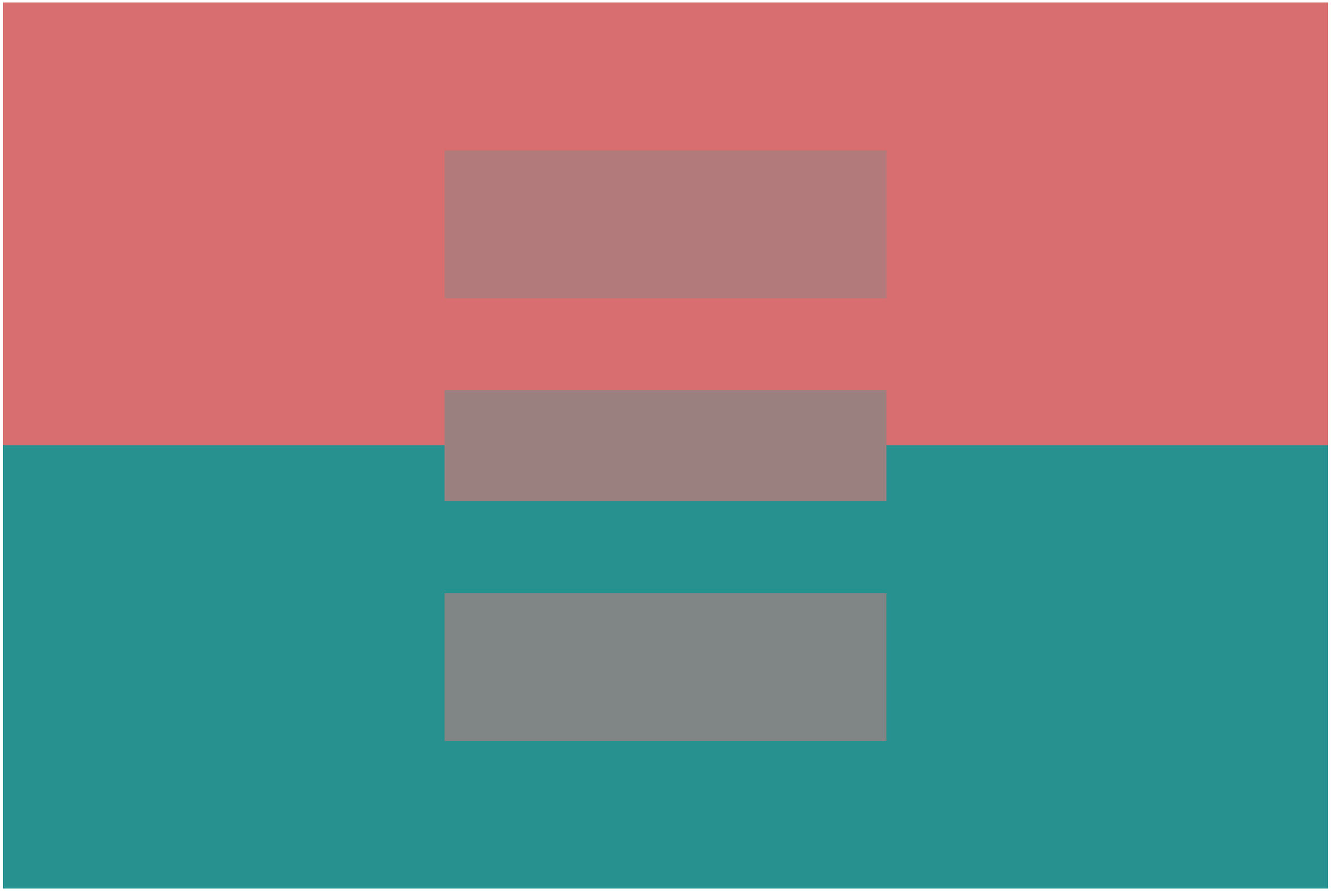
```
  for(int cols = 0; cols < 4; cols++) {
    rect(pgwidth * 0.668 + 140*cols, pgheight * 0.3 + 140*rows,100,100);
  }
}
popMatrix();

popMatrix();
```

0xFF301EE7 0xFFCFE118 0xFF7C82BE 0xFF96A0A4 0xFFACB984

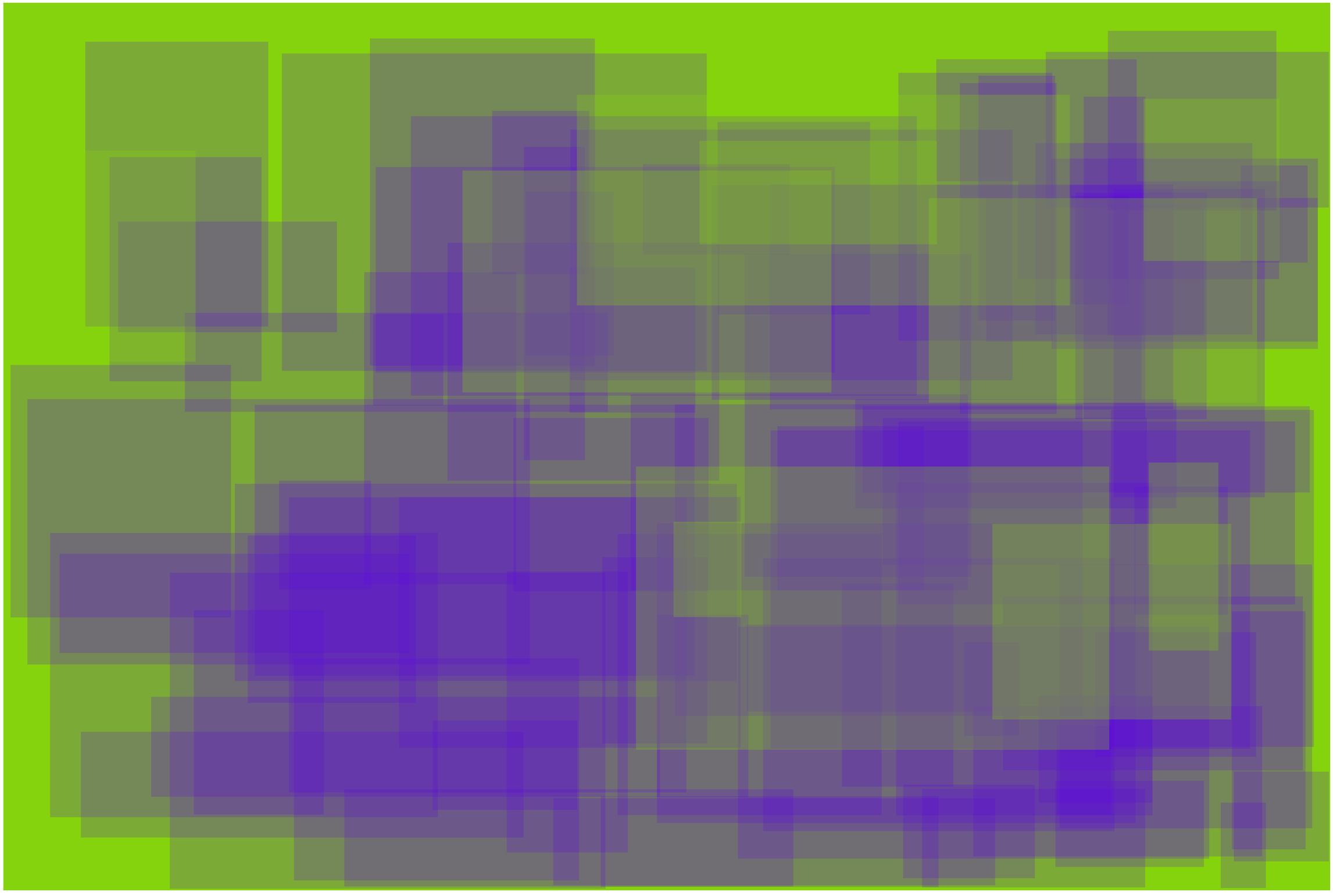


0xFF23AF2A 0xFFDC50D5 0xFF80967D 0xFF9D8899 0xFFB577B0

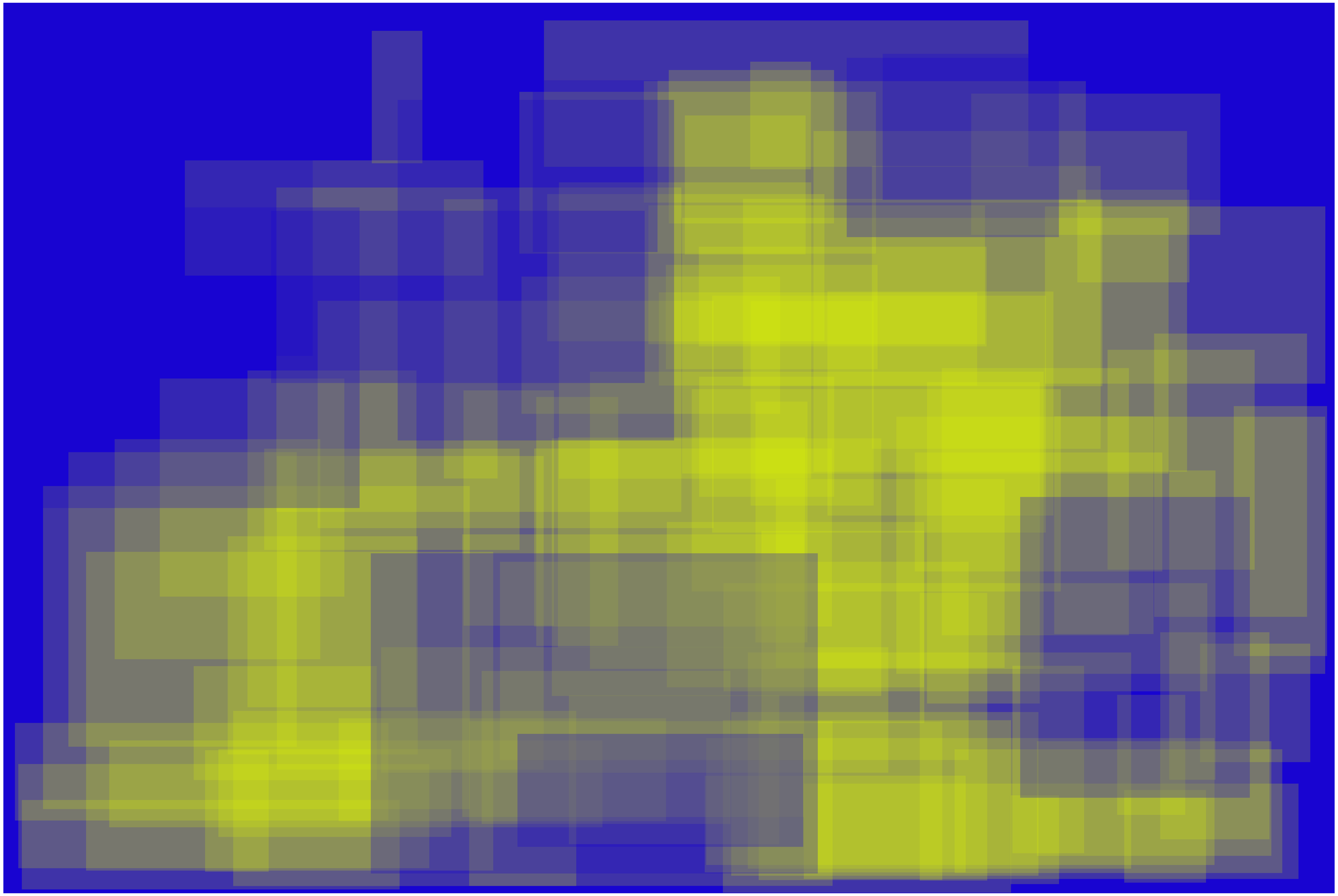
0xFFD86E6F 0xFF279190 0xFFB27A7A 0xFF9B8080 0xFF7F8685



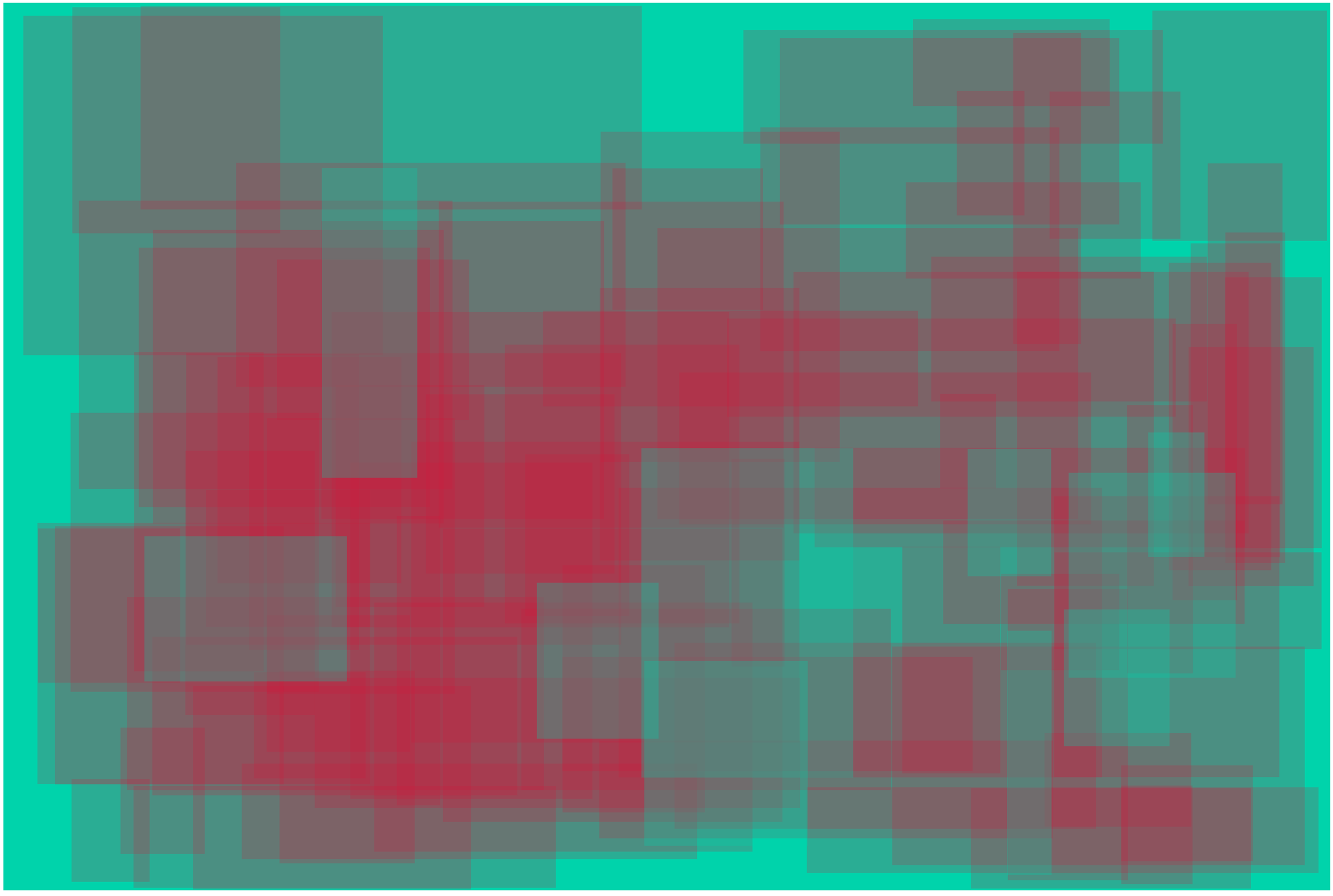
0xFF5C02E6 0xFF85D30C



0xFFE0F704 0xFF1804D1



0xFFD7153A 0xFF00D3AB



```
// 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();
```

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

0xFF494D5B

0xFF224258

0xFF0B818C

0xFF4D717B

0xFF8C8A88

0xFFBEA9AA

0xFFD2CCC9

0xFFFAF8BF

0xFF90595D

0xFF6B5760

0xFF188E9F

0xFF160E2A

0xFFBA3B3C

0xFFDD3A3A

0xFFD32F2F

0xFF8B383A

0xFFE2E5DA

0xFFB1AB97

0xFF90866D

0xFF716449

0xFFB78F2F

0xFF4E3B1F

0xFF352207

0xFFE55B08

0xFFD9D2CA

0xFEB87B7A
0xFF551E34

0xFF3C102E

0xFFD56465

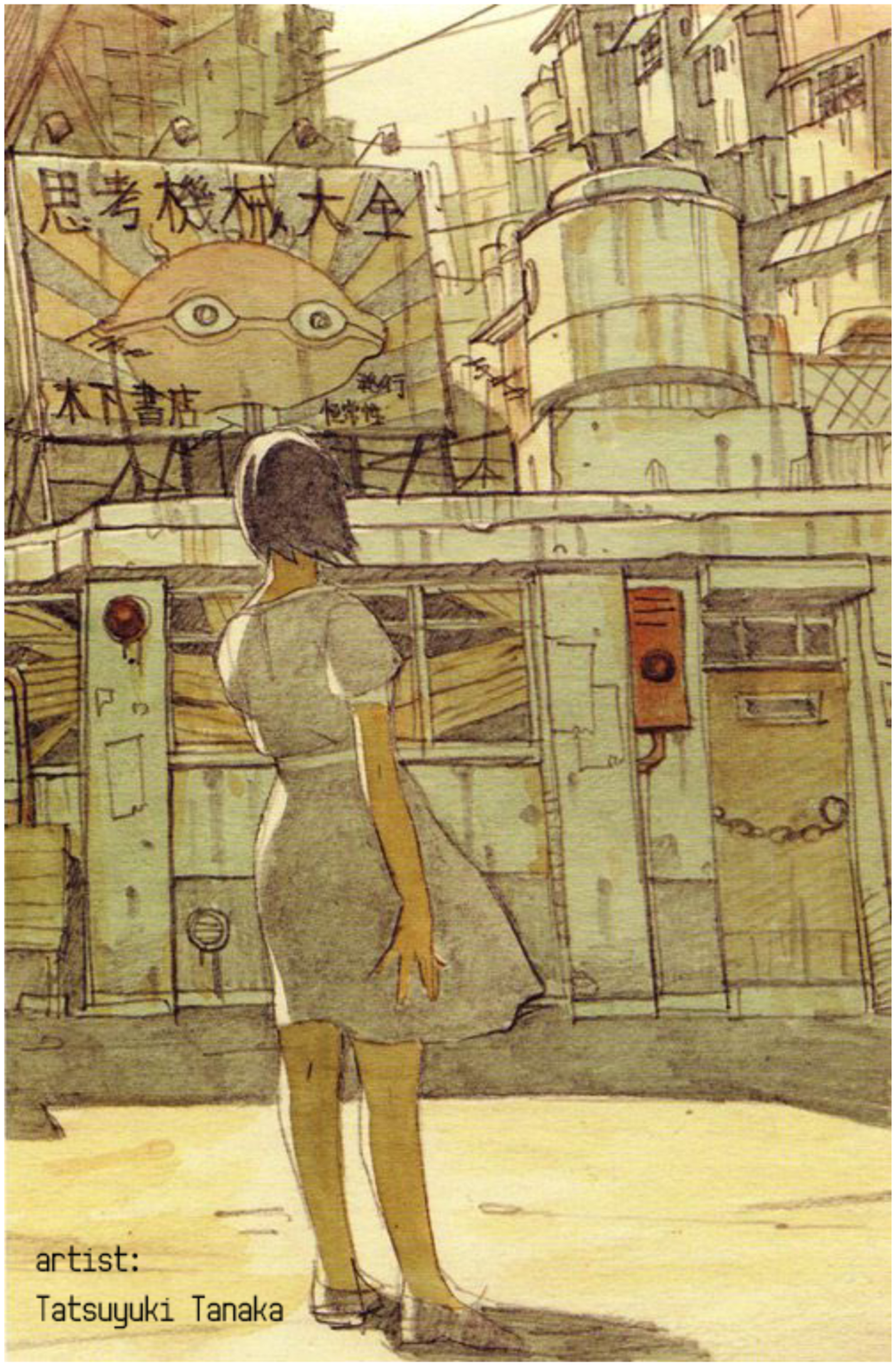
0xFFAA3D45

0xFE230B2A
0xFF782533

0xFFE67071

0xFFDF393E

0xFFF1E8E4



0xFFBAA873

0xFF9D8657

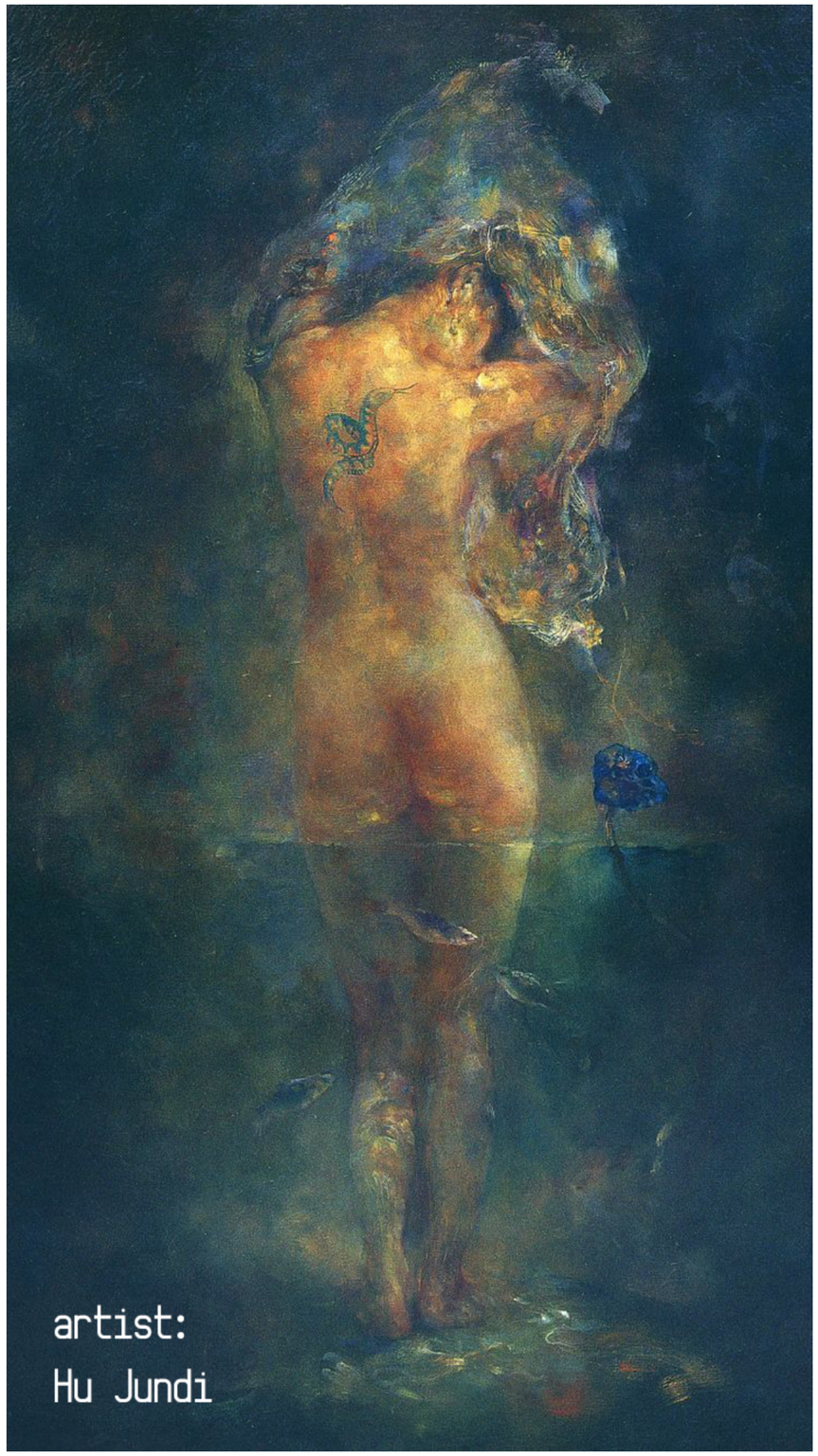
0xFF775D3C

0xFF543A27

0xFF371F16

0xFFE9DAA9

0xFFFAF3CB



0xFF253B44

0xFF505955

0xFF122D3A

0xFF756957

0xFF9D825D

0xFF8A8C9F

0xFFB48755

0xFFD6A663

0xFF695544

```
// 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();

}
```