p5.js Activity Guide

SPLIT / REPEAT / PAINT

by Murat Atimtay

An introduction to creating abstract city silhouettes using recursion.

01 / THE TECHNIQUE

Each rectangle splits into two, and the smaller pieces repeats the same rule.

More vertical cuts create tall, narrow columns, while occasional horizontal cuts break them into different heights.

Together with colours that darken towards the bottom, these divisions suggest an abstract city skyline.

Steps means: how many more times should rectangle split?

Imagine starting with steps = 1. The whole rectangle splits into two.

Each call has its own steps value. steps - 1 gives a smaller number to a new call; and the function calls itself. This is called recursion.

LETS START ->

02 / START HERE

Open editor.p5js.org. Replace sketch.js. Add the code on pages 3-7 in order.

```
const steps = 11;
const cityColor = 0;

function setup()
{
  createCanvas(600, 800);
  colorMode(HSB, 360, 100, 100);
  noStroke();
  noLoop();
}
function draw() {
  splitRect(0, 0, width, height, steps, cityColor);
}
```

steps variable sets the number of splits along each branch. cityColor sets the starting hue in HSB mode.

noLoop() makes one image. Press Play again to generate another silhouette. Play with steps and cityColor variable

*Increasing steps too much may cause your computer to freeze

03 / WHEN TO STOP

```
function splitRect(x, y, w, h, steps,hu)
{
  // Stop splitting and draw this
  rectangle.

  if (steps === 0) {

    colourRect(x, y, w, h,hu);

    return;
  }
  .
  .
  .
```

x, y: top-left corner of the cell to split
w, h: width and height of the cell to split
hu: starting hue.

This function calls itself until steps reach to 0

At 0, colour this rectangle and finish this call. Otherwise, continue to the split.

04 / LEFT AND RIGHT

```
let ratio = random(0.6, 0.9);
.
.
.
if (random()<0.72) {
  // Split into left and right rectangles.
  let cut = floor(w * ratio);

  splitRect(x, y, cut, h, steps - 1,hu);
  splitRect(x + cut, y, w - cut, h,
            steps - 1,hu);
}
.
.
.
```

ratio is the amount of the split that the rectangle gets. Keeping it between 0.6 and 0.9 means one part of the split is always bigger than the other.

if(random() < 0.72) condition chooses vertical cuts about 72% of the time:

let cut = floor(w * ratio); splits the rectangle into two parts

Using steps - 1, splitRect calls itself for both the left and right rectangles, repeating until each call reaches steps = 0.

05 / TOP AND BOTTOM

```
} else {
    // Split into top and bottom rectangles.
    let cut = floor(h * ratio);

    splitRect(x, y, w, cut, steps - 1,hu);
    splitRect(x, y + cut, w, h - cut, steps -
1,hu);
  }
}
```

floor(h*ratio) ensures that the cut position is pixel perfect to prevent blurry edges

The top piece has height cut. The bottom piece starts at y + cut and has height h - cut.

hu passes unchanged to every piece. The last brace closes splitRect()

06 / PAINT

```
function colourRect(x, y, w, h,c=23)
{
  let middle = y + h / 2;

  let hue = map(middle, 0, height, c, c+23 );
  let light = map(middle, 0, height, 125,
10);
  let sat = map(middle, 0, height, 10, 255);

  fill(hue, sat, light);
  rect(x, y,w, h);
}
```

This is where the magic happens. Decreasing brightness and increasing saturation from top to bottom creates atmospheric perspective: pale, hazy cells recede into the distance, while darker, richer colours bring the lower forms into the foreground.

The center of each rectangle is calculated by :
let middle = y + h / 2;

map() function uses the centroid position of each cell to change the hue, saturation and brightness.

c+23 is the amount of of change in hue. You can change 23 to something bigger or smaller and see how it changes

07 / CHANGE A RULE

Change the **step** variable and see how the subdivision changes

```
1 step = 2 final rectangles
2 steps = 4 final rectangles
3 steps = 8 final rectangles
11 steps = 2048 rectangles
```

Replace random(0.6, 0.9) with 0.5 to make equal halves.

if (random()<**0.72**)

Replace **0.72** with something else and see how subdivision style changes

Play with hue, saturation and lightness

Change one rule at a time. Predict what will happen, run the code, and compare.