Experiment 02: Living Impressions

Chengkun Li

Computational Media, University of California - Santa Cruz

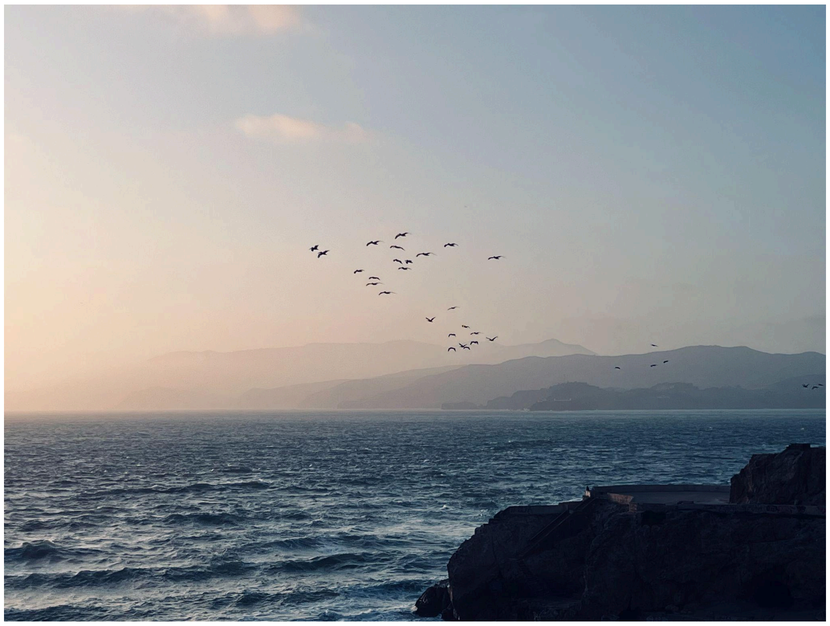
CMPM 147: Generative Design

Prof. Wes Modes

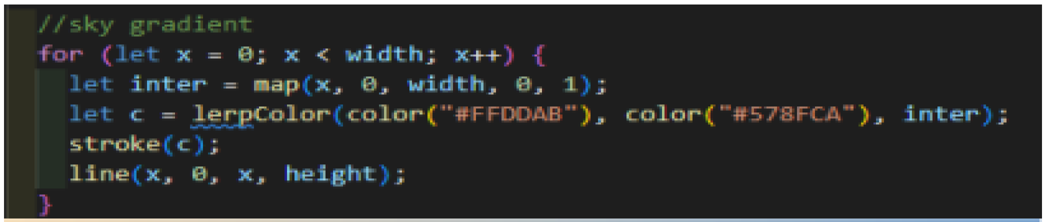
4/14/2025

Step 1 - Imitate:

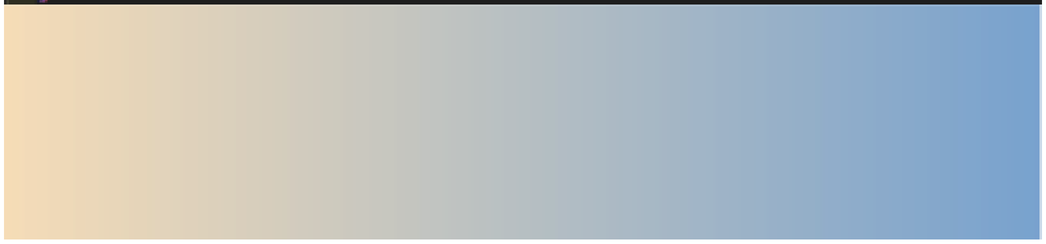
For a person with a choice phobia, it is difficult to choose an item among many targets. So before typing the code, I wandered in my photo album for a long time. Finally, I chose a picture of the seaside. I am not good at using a camera, so many of the photos I took are different from what I saw, and they don't have the feeling I want, so I take fewer and fewer photos. The reason I chose it is that I think it is my favorite, the one that I feel the most, and the one that is closest to what I see with my naked eyes.



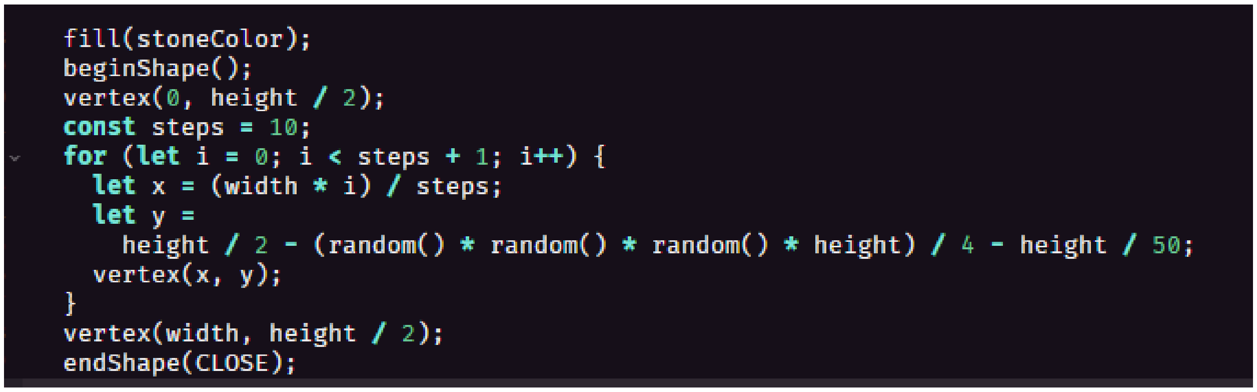
What I like most about this photo is the atmosphere and the lighting from the sun. So I started working with the background colors. I noticed that the image was roughly divided into two colors: orange and gray. I realized that a single-color background would not help me imitate the image. So I looked for a video tutorial on gradient colors. Finally, I found [flanniganable's video](). So I simply copied his code. But I needed a gradient from left to right, not from top to bottom. So I modified his code a little bit. I used https://colorhunt.co/ to find the colors.



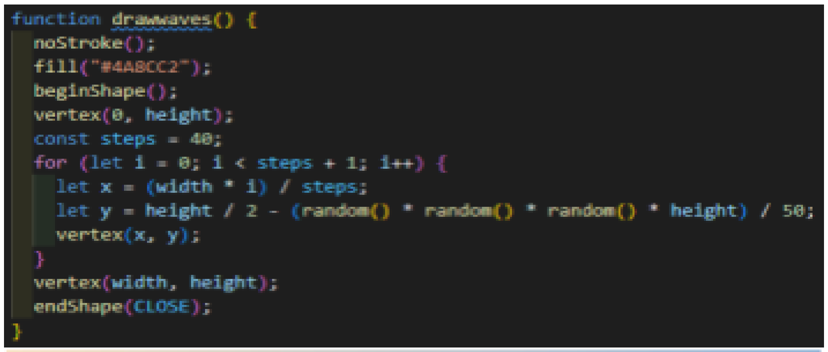
```
//sky gradient
for (let x = 0; x < width; x++) {
  let inter = map(x, 0, width, 0, 1);
  let c = lerpColor(color("#FFDDAB"), color("#578FCA"), inter);
  stroke(c);
  line(x, 0, x, height);
}
```



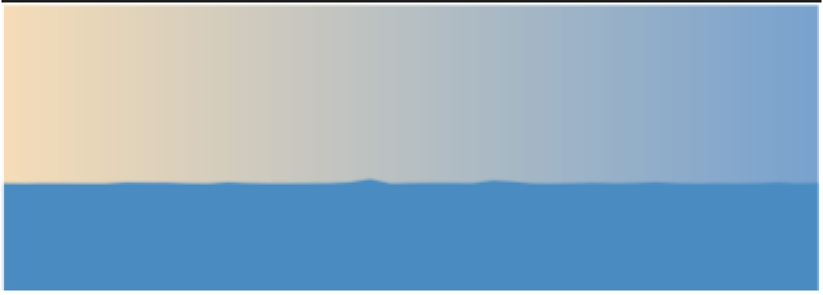
I like the ocean very much because I think the wind at the seaside is very comfortable. It can temporarily blow away my messy troubles. Therefore, the next step I going to do is make the waves. For this part, I copied the code from Prof. Modes. He used this part of the code to create the mountains in the background, but I think the ocean waves can also be created with this code.



```
fill(stoneColor);
beginShape();
vertex(0, height / 2);
const steps = 10;
for (let i = 0; i < steps + 1; i++) {
  let x = (width * i) / steps;
  let y =
    height / 2 - (random() * random() * random() * height) / 4 - height / 50;
  vertex(x, y);
}
vertex(width, height / 2);
endShape(CLOSE);
```

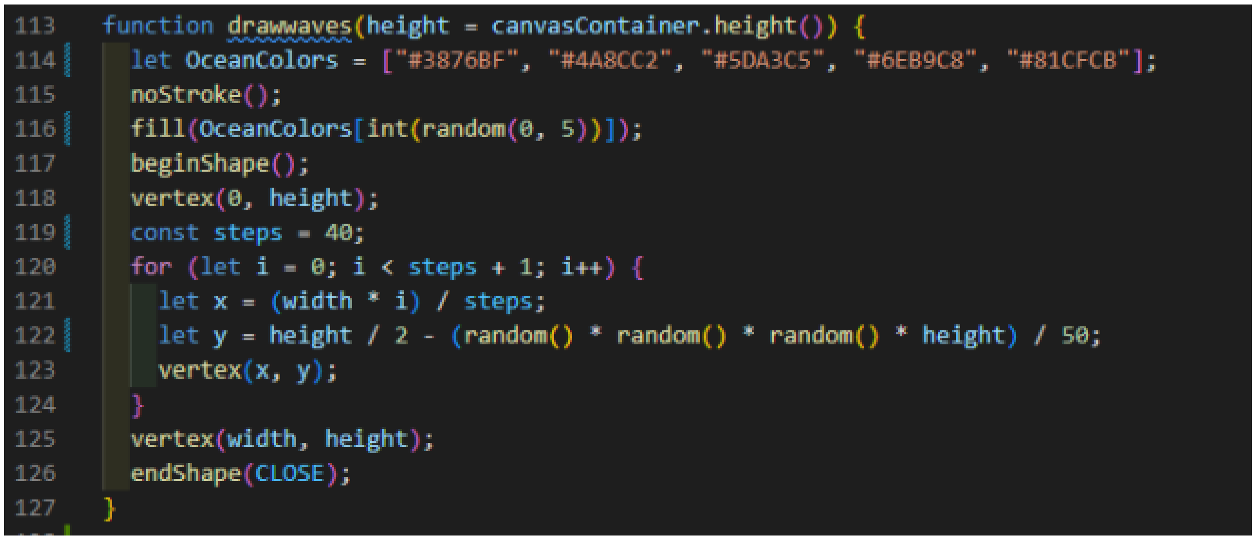


```
function drawwaves() {
  noStroke();
  fill("#4A8CC2");
  beginShape();
  vertex(0, height);
  const steps = 40;
  for (let i = 0; i < steps + 1; i++) {
    let x = (width * i) / steps;
    let y = height / 2 - (random() * random() * random() * height) / 50;
    vertex(x, y);
  }
  vertex(width, height);
  endShape(CLOSE);
}
```

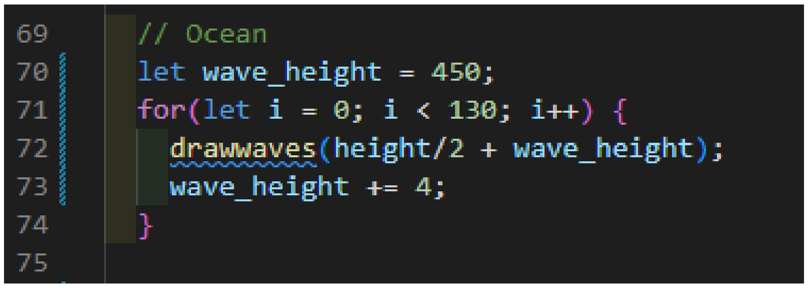


In the beginning, I didn't understand how the code works. So I kept changing the parameters, the number of steps, height, width, and also tried removing random(). Then I noticed the vertex() function, so I searched for its use in p5js. Then I saw this video. After watching it, I figured out how to use the vertex() function. After some tinkering, I got something like this.

However, it is not really what I want. It can simply represent the sea surface, but it has not yet achieved the effect I want, that is, the sparkling waves in the picture. Then I thought about whether I could add multiple outlines to achieve the effect of waves. With this idea in mind, I first looked for some blue so that I could see the differences. The second thing, I am thinking about is the height. I wanted to test whether I could use height differences to express layers of waves. I added a parameter to the drawwaves() function for different heights and then added a different blue color to give the illusion of ocean waves.

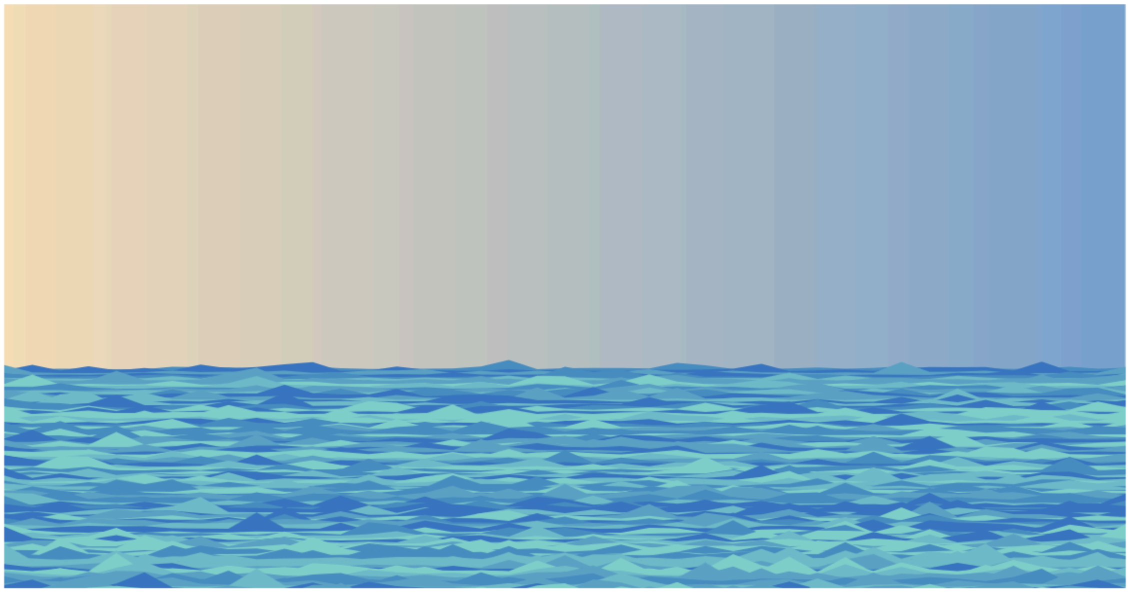


```
function drawwaves(height = canvasContainer.height()) {
    let OceanColors = ["#3876BF", "#4A8CC2", "#5DA3C5", "#6EB9C8", "#81CFCB"];
    noStroke();
    fill(OceanColors[int(random(0, 5))]);
    beginShape();
    vertex(0, height);
    const steps = 40;
    for (let i = 0; i < steps + 1; i++) {
        let x = (width * i) / steps;
        let y = height / 2 - (random() * random() * random() * height) / 50;
        vertex(x, y);
    }
    vertex(width, height);
    endShape(CLOSE);
}
```



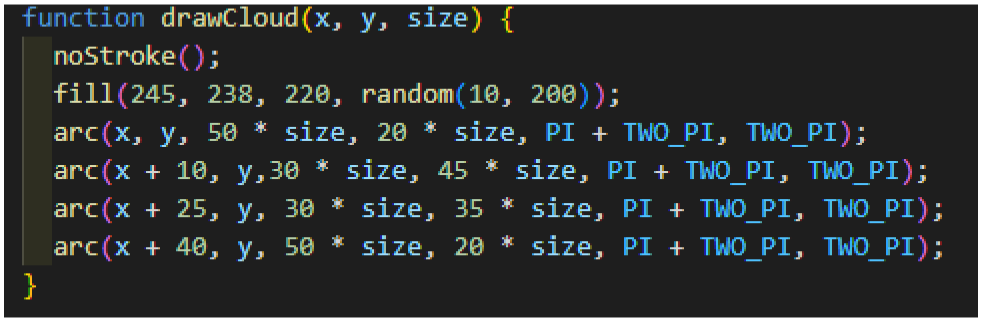
```
    // Ocean
    let wave_height = 450;
    for(let i = 0; i < 130; i++) {
        drawwaves(height/2 + wave_height);
        wave_height += 4;
    }
```

And the for loop, The for loop is used to generate wave outlines of different heights, extending all the way to the bottom of the canvas. And then, I got this.

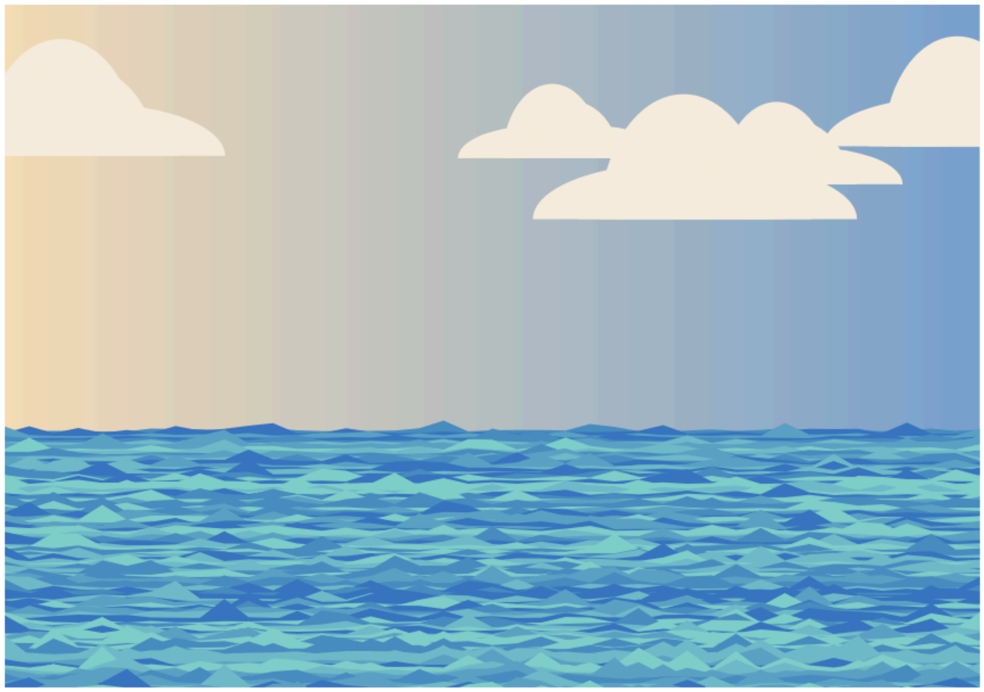


Step 2 - Integrate:

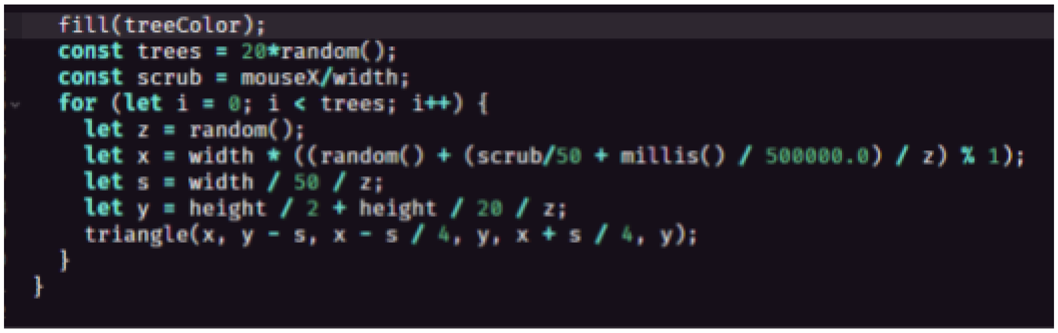
I don’t think the clouds and rocks in the picture are very important, and can even be said to be dispensable. Therefore, I want to choose a simpler way to implement them. At the beginning of At first, I searched how to draw clouds in JS and p5js, and then I saw Stack Up's video and this p5js example by Jackiezen. I combined the code from the video with the code from the p5js example and came up with this.



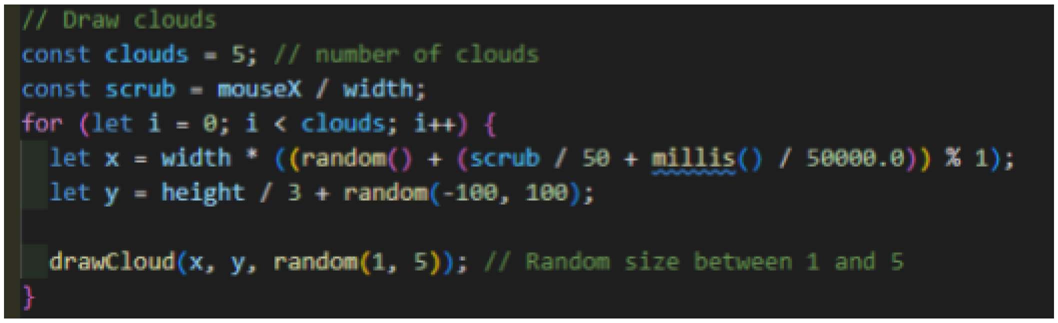
```
function drawCloud(x, y, size) {
  noStroke();
  fill(245, 238, 220, random(10, 200));
  arc(x, y, 50 * size, 20 * size, PI + TWO_PI, TWO_PI);
  arc(x + 10, y,30 * size, 45 * size, PI + TWO_PI, TWO_PI);
  arc(x + 25, y, 30 * size, 35 * size, PI + TWO_PI, TWO_PI);
  arc(x + 40, y, 50 * size, 20 * size, PI + TWO_PI, TWO_PI);
}
```



However, the cloud won't move. Then I thought of the code from Prof. Modes. In his code, the trees can be moved slowly, so I wanted to apply it to my clouds. I copied and modified his code and got my cloud moving successfully.

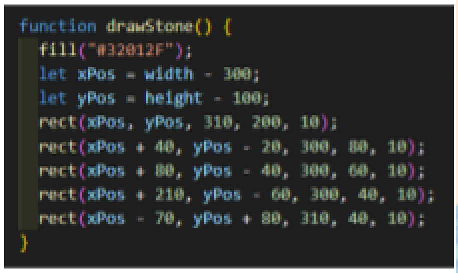


```
  fill(treeColor);
  const trees = 20*random();
  const scrub = mouseX/width;
  for (let i = 0; i < trees; i++) {
    let z = random();
    let x = width * ((random() + (scrub/50 + millis() / 500000.0) / z) % 1);
    let s = width / 50 / z;
    let y = height / 2 + height / 20 / z;
    triangle(x, y - s, x - s / 4, y, x + s / 4, y);
  }
}
```

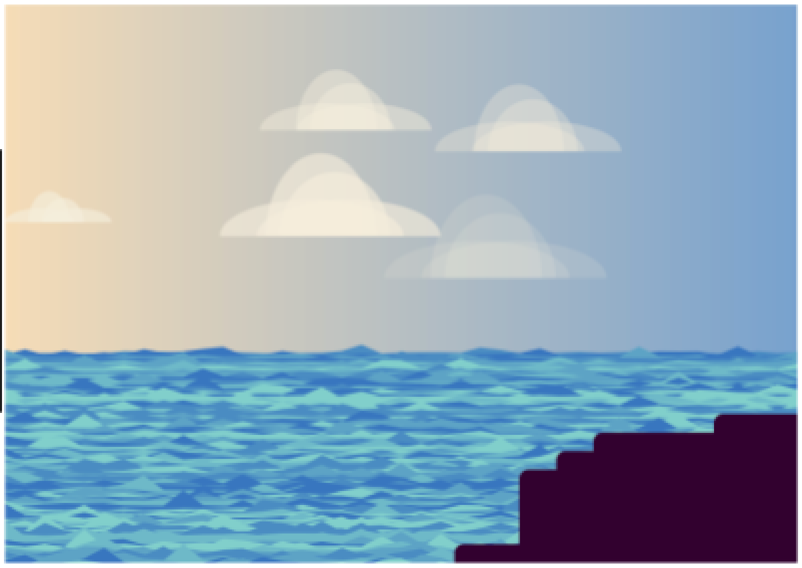


```
// Draw clouds
const clouds = 5; // number of clouds
const scrub = mouseX / width;
for (let i = 0; i < clouds; i++) {
  let x = width * ((random() + (scrub / 50 + millis() / 50000.0)) % 1);
  let y = height / 3 + random(-100, 100);

  drawCloud(x, y, random(1, 5)); // Random size between 1 and 5
}
```

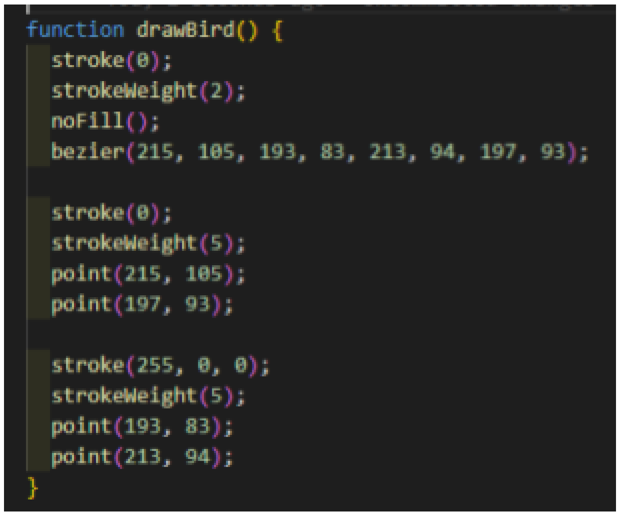
However, I still felt that something was missing from the clouds. Then I thought of the haziness. Then I searched and changed the transparency, and then I saw this post. From AngelLeatherist's comment, I learned that the last number of "fill(0,0,0,0)" represents transparency. So I changed the "fill(245, 238, 220);" in the drawCloud() equation in the code to "fill(245, 238, 220, random(10, 200));" For the stone, I simply used a few rectangles to represent it. Overall, the below images are what I got.



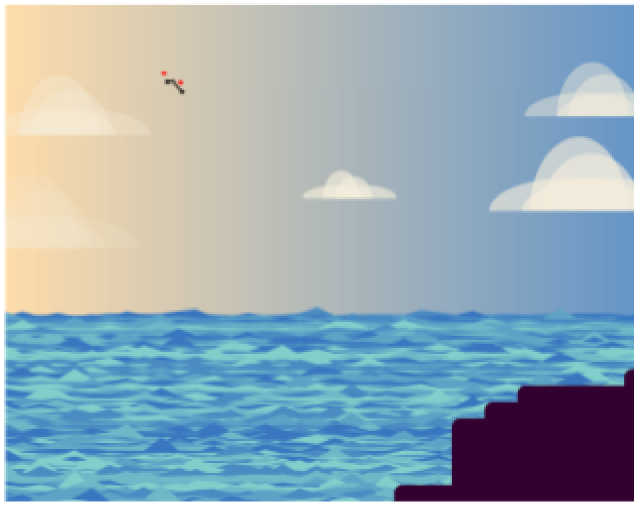
```
function drawStone() {
  fill("#32012F");
  let xPos = width - 300;
  let yPos = height - 100;
  rect(xPos, yPos, 310, 200, 10);
  rect(xPos + 40, yPos - 20, 300, 80, 10);
  rect(xPos + 80, yPos - 40, 300, 60, 10);
  rect(xPos + 210, yPos - 60, 300, 40, 10);
  rect(xPos - 70, yPos + 80, 310, 40, 10);
}
```

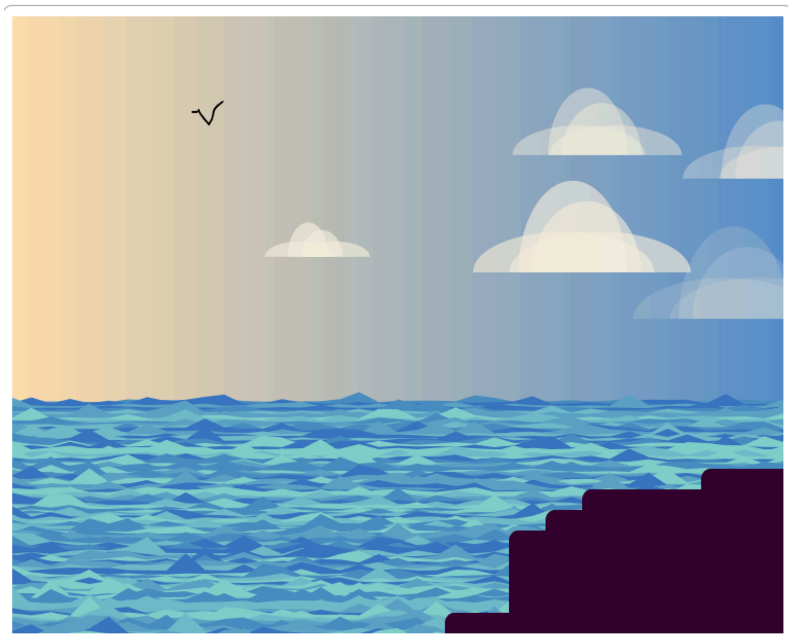


For the seagulls/birds, I wanted to express them with lines/curves. And then I found this video. In the video, he used bezier() to draw a whole seagull, but that is not what I want. So I just simply copied his first step of the code, which is “bezier(215, 105, 193, 83, 213, 94, 197, 93);”, and based this documentation to draw the points and curves.

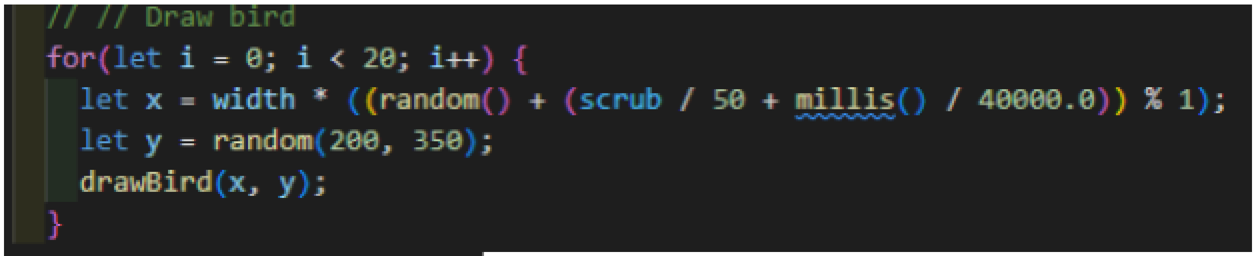


```
function drawBird() {
  stroke(0);
  strokeWeight(2);
  noFill();
  bezier(215, 105, 193, 83, 213, 94, 197, 93);

  stroke(0);
  strokeWeight(5);
  point(215, 105);
  point(197, 93);

  stroke(255, 0, 0);
  strokeWeight(5);
  point(193, 83);
  point(213, 94);
}
```

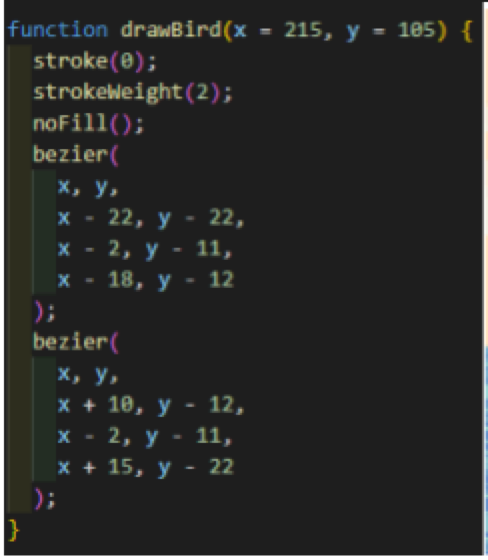




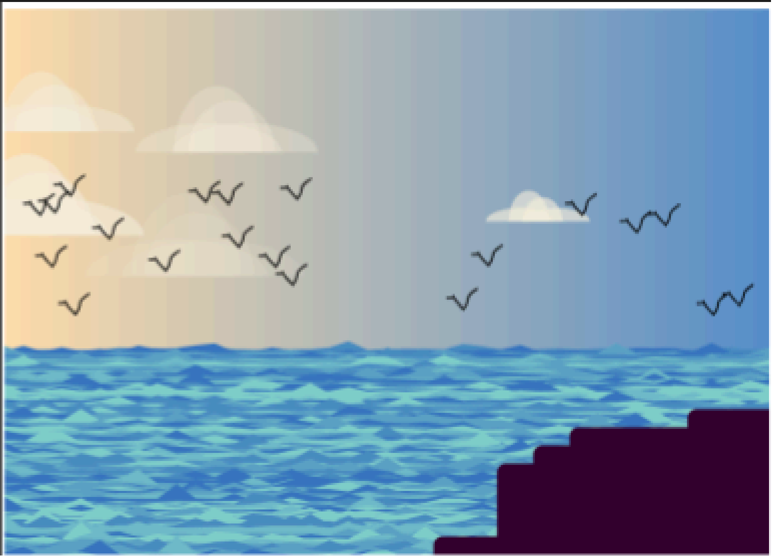
The left side image is after I figured out how bezier() worked, and tried drawing another curve, which just looked like the shape of a seagull in flight. Based on this simple seagull, I continued to modify the drawBird() function so that it can receive two parameters, which is the bottom point. Based on that point, I can modify the number in the bezier. At the same time, I also want it to move like the clouds. And the below images are what I got.



```
// // Draw bird
for(let i = 0; i < 20; i++) {
  let x = width * ((random() + (scrub / 50 + millis() / 40000.0)) % 1);
  let y = random(200, 350);
  drawBird(x, y);
}
```

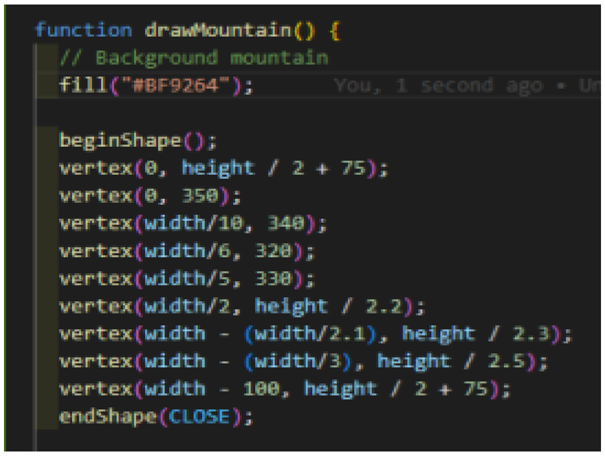


```
function drawBird(x = 215, y = 105) {
  stroke(0);
  strokeWeight(2);
  noFill();
  bezier(
    x, y,
    x - 22, y - 22,
    x - 2, y - 11,
    x - 18, y - 12
  );
  bezier(
    x, y,
    x + 10, y - 12,
    x - 2, y - 11,
    x + 15, y - 22
  );
}
```

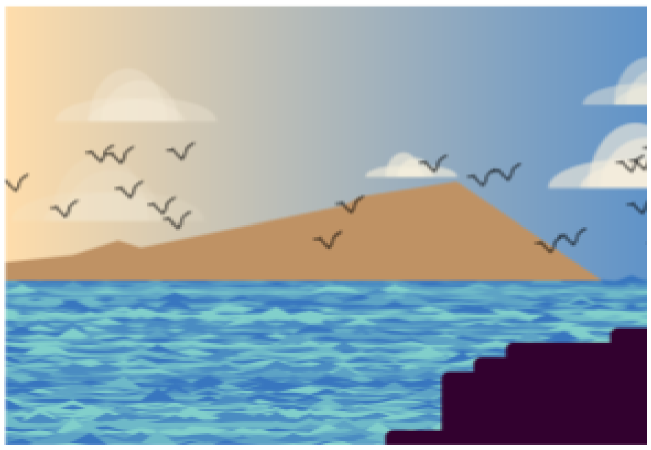


Step 3 - Innovate:

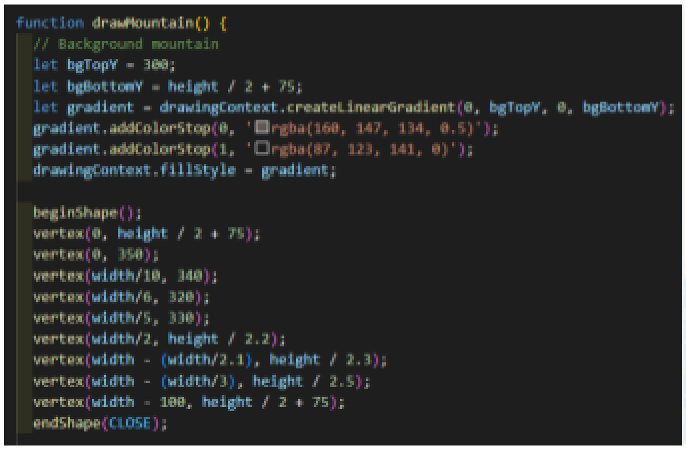
I think the most successful part of the photo is the looming mountains, which have the feel of a Chinese ink painting. Initially, I used the way that I implemented the waves to implement it, but I found that it was not what I wanted, because this approach could not reflect the situation of the mountain slowly rising from left to right. So I went about it in the most primitive and basic way possible: by marking each mountaintop. The image below shows what I have done.



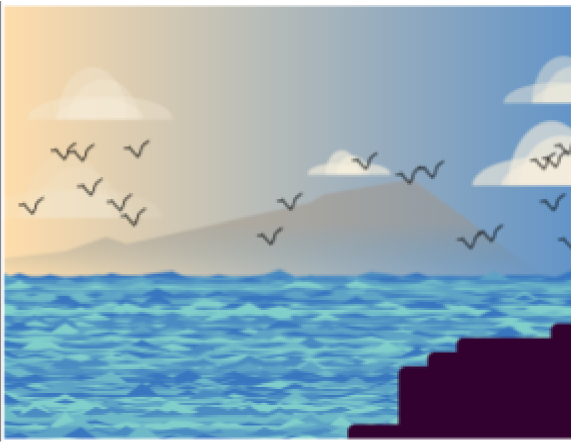
```
function drawMountain() {
  // Background mountain
  fill("#BF9264");        You, 1 second ago • Un

  beginShape();
  vertex(0, height / 2 + 75);
  vertex(0, 350);
  vertex(width/10, 340);
  vertex(width/6, 320);
  vertex(width/5, 330);
  vertex(width/2, height / 2.2);
  vertex(width - (width/2.1), height / 2.3);
  vertex(width - (width/3), height / 2.5);
  vertex(width - 100, height / 2 + 75);
  endShape(CLOSE);
```



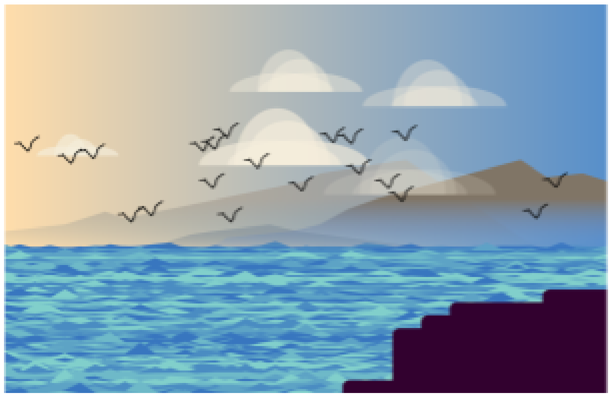
However, the mountains drawn in this way are not quite what I imagined, because they lack a certain feeling, the fairy-like feeling in ink painting. So, I thought of adjusting the transparency, but not the overall transparency, but the transparency that changes gradually from top to bottom, which is, that the top of the mountain is opaque and the bottom of the mountain is transparent. Based on this idea, I searched for how to adjust the gradient transparency in p5js. And then I found this video. From this video, I got a general idea of how gradient colors work. The below image is what I got.

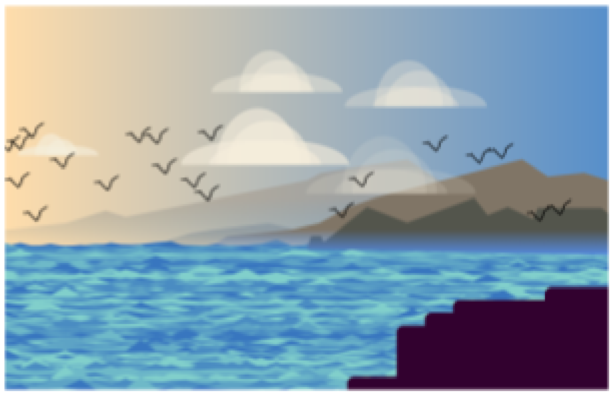


```
function drawMountain() {
  // Background mountain
  let bgTopY = 300;
  let bgBottomY = height / 2 + 75;
  let gradient = drawingContext.createLinearGradient(0, bgTopY, 0, bgBottomY);
  gradient.addColorStop(0, 'rgba(160, 147, 134, 0.5)');
  gradient.addColorStop(1, 'rgba(87, 123, 141, 0)');
  drawingContext.fillStyle = gradient;

  beginShape();
  vertex(0, height / 2 + 75);
  vertex(0, 350);
  vertex(width/10, 340);
  vertex(width/6, 320);
  vertex(width/5, 330);
  vertex(width/2, height / 2.2);
  vertex(width - (width/2.1), height / 2.3);
  vertex(width - (width/3), height / 2.5);
  vertex(width - 100, height / 2 + 75);
  endShape(CLOSE);
```



Based on this way, I used it to draw other mountains. However, there was a new problem when I drew the third mountain out. The low transparency of the bottom of the mountain will expose the mountain behind it, which is a bit contrary to the visual sense. (the left-side image below)





But then I found out that it can be solved by just modifying one value, which is the vertex of the y-axis where the gradient needs to occur. After finding the problem, I drew the remaining mountains. At this point, I consider my project complete (the right-side image below). The whole process can be said to be very torturous, but also very interesting. Sometimes, I am always confused by some values, and then keep modifying them to get what I want. Later on, I might go back and tweak some values to get it just how I want it. And you are welcome to take a look at my code, site, and Glitch.

Reflection:

First of all, I will say this assignment is challenging and enjoyable. I am proud to see that I can implement the photo. I had never heard about p5js before I got into this class, so I am kind of worried about it because it means I have to read and test a lot of stuff, which I really hate.

I started this assignment pretty late, so I didn't have much time to think about the code. The examples in class helped me a lot, I think 50% ideas are from the examples. When making the mountains, I originally just entered some numbers as the vertex values. But then I found that when I changed the size of the canvas, the shape of the mountains would also change. So I had to spend a lot of time looking for a more appropriate way to replace the vertex values. In the end, I used the some values with width and height to replace those numbers. After looking at the projects made by other students, I feel that mine still needs to be improved. Overall, my evaluation of this implementation is qualified because it has not yet reached what I wanted in my mind. It does not achieve the hazy feeling of ink painting.

Self Evaluation Rubric						
Completion: Did you complete the assignment and did you complete it on time?	Submitted on time	Up to 1 day late	Up to 2 days late	Up to 3 days late	4 days late or more	Do you need to clarify? I complete it on time
	X	☐	☐	☐	☐	
Effort: Did you put in earnest effort in execution and attention to detail?	Excellent	Pretty good	About average	Could be improved	Not this time	What supports this? During the execution, I watched several videos and documents to learn and use p5js. However, I often forgot to take screenshots of the code, so I missed some details.
	☐	☐	X	☐	☐	
Results: How well did you accomplished the assignment goals? Was it complete, with minimal errors, correct output, and good style?	Excellent	Pretty good	About average	Could be improved	Not this time	What supports this? I would say the assignment is completed, with no errors, correct output and it has a good style because the site is working when the code runs and it doesn't show any error information.
	☐	X	☐	☐	☐	
Reflection: How much did you reflect on the strengths and weaknesses of the work and what you'd like to improve?	Excellent	Pretty good	About average	Could be improved	Not this time	What supports this? I don't think my code has any strengths, but the weaknesses of the code are that it is too long and the way I implement them is kind of stupid. I'm also still not satisfied with the shape of the mountains in the background. Because it doesn't reflect that natural curve, I meant the mountaintop. I might find a way to make a better mountaintop if I have time later.
	☐	☐	X	☐	☐	
Reflection in section above.						