Custom shapes

Although it's a good exercise to design only with simple shapes, complex shapes offer more possibilities. In a manual design process, complex shapes often take a long time to draw, as every detail of a design will need to be created by hand. Although efforts have been made to automate such tasks, some designs are still tedious to create in current digital design tools like Adobe Illustrator or Sketch. This is particularly true for designs that require the use of repetition or randomization, like a pattern of Sine curves with changing amplitudes. In code, we have the ability to procedurally generate very complex shapes in an instant, and the code required can be quite simple. On the other hand, shapes drawn randomly with a pen can be hard to recreate in code, especially if there is no underlying rule to explain the outline of the shape.

In the following chapters, I will introduce a range of techniques to procedurally draw custom shapes. However, we must first understand the basic concepts of drawing shapes in code, which means looking at the `beginShape()` function, as well as the many vertex functions that can be used to define the outline of a shape.

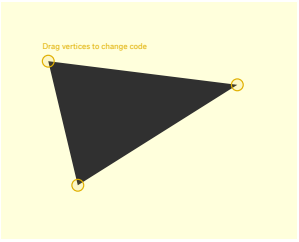
Programming custom shapes

Most graphics programming languages allow you to draw custom shapes like a Connect the Dots drawing: You define a series of points – which we will refer to as vertices – that are connected via lines to form the outline of a shape.

Each vertex in a shape determines how it is connected to the vertex before it. If it's a simple vertex, it will be connected with a straight line. If it's a curved vertex, it will be connected with a curved line. The shape can optionally become a closed shape by connecting the last vertex to the first vertex. P5.js follows this same concept. Use the `beginShape()` function to start a new custom shape, define the vertices of the shape with the desired vertex functions, and finally connect the lines in the shape by calling the `endShape()` function with an optional argument to close the shape. In the following, we will examine these vertex functions.

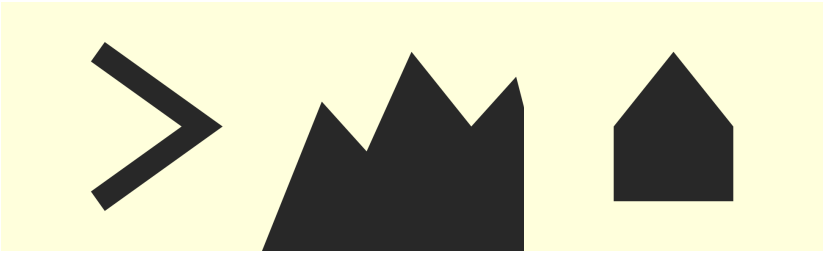
Straight lines

The `vertex()` function creates a simple vertex that connects to the vertex before it with a straight line. This is the simplest of the vertex functions, and all shapes created with `beginShape()` must start with a `vertex()` function call to define the starting point of the shape. This is illustrated in the example below. Try dragging the vertices to see the resulting code.




```
beginShape();
  vertex(80, 100);
  vertex(400, 140);
  vertex(130, 310);
endShape(CLOSE)
```

The following examples are all created with simple vertices, but use strokes and fills to achieve very different designs.



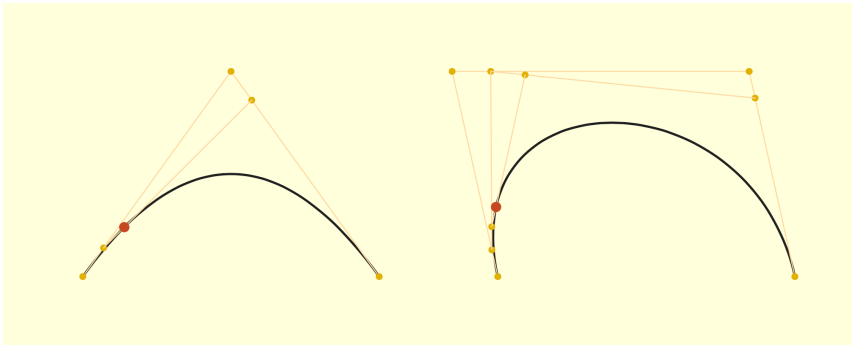
See Code See Code See Code

Bézier curves

To create a vertex that is connected to the vertex before it with a curved line,

we use the `quadraticVertex()` and `bezierVertex()` functions. These are a bit more complex than the `vertex()` function, because they need several `x` and `y` coordinates to control the curve of the line. To understand how this works, let's have a brief look at the concept of Bézier curves.

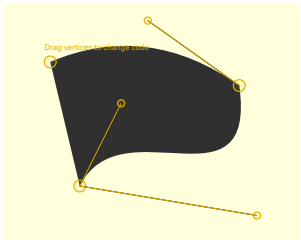
The Bézier curve algorithm was popularized by Pierre Bézier in the 1960's as a solution to a common problem in computational geometry: Drawing curved lines that can scale to any size. The Bézier curve algorithm solves this problem in a very elegant way by introducing the idea of control points: Invisible gravity points that attract the line to bend it into a curve. A Bézier curve with a single control point is called a quadratic Bézier, while a Bézier curve with two control points is called a cubic Bézier. If you have ever used the Pen tool in Adobe Illustrator, you are already familiar with this concept.



This animation shows how a quadratic Bézier curve is calculated.

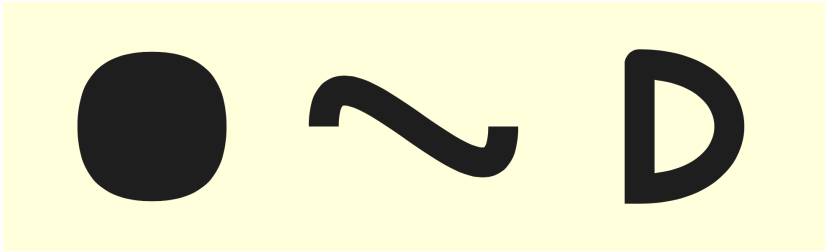
This animation shows how a cubic Bézier curve is calculated.

You can draw a quadratic bezier curve with the `quadraticBezier()` function, passing the coordinates for the single control point and the vertex itself. Likewise, you can draw a cubic Bézier curve with the `bezierVertex()` function, passing coordinates for the two control points and the vertex itself. The only difference between the two functions is the addition of an extra control point in the `bezierVertex()` function, which allows you to draw more sophisticated curves. This is illustrated below where both types of curves are used to draw a custom shape. Try dragging the vertices and control points to see the resulting code.



```
beginShape();
  vertex(80, 100);
  quadraticVertex(245, 30, 400, 140);
  bezierVertex(430, 360, 200, 170, 130, 310);
endShape(CLOSE)
```

It takes a bit of practice to master the Bézier functions, and knowing how many Béziers you need to draw a specific shape can be hard in the beginning. It doesn't help that control points are invisible, so it can be helpful to spend some time playing around with the example above before diving into the code. Below are three examples that all use the Bézier functions to create custom shapes.



See Code

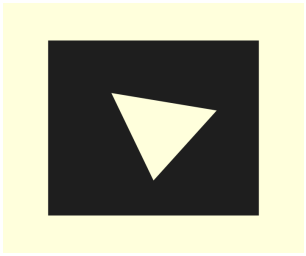
See Code

See Code

Contours

While we can draw most shapes with `vertex()`, `quadraticVertex()`, and `bezierVertex()`, these functions won't allow us to create shapes with holes. In P5.js, a hole is called a contour, and you can draw shapes with contours using the `beginContour()` and `endContour()` functions. In essence, the `beginContour()` function instructs P5 that you a starting a

new shape that be subtracted from your main shape. Like `beginShape()`, you use the vertex functions to draw your contour, and use `endContour()` to end the contour.

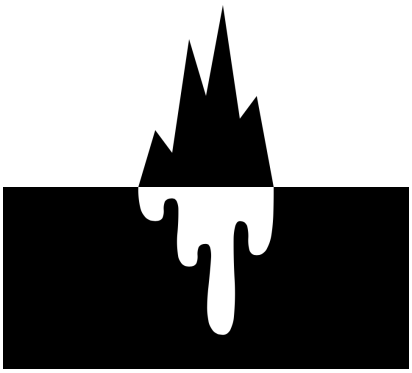


```
beginShape();
  // draw rectangle here
  beginContour();
    // draw triangle here
  endContour();
endShape();
```

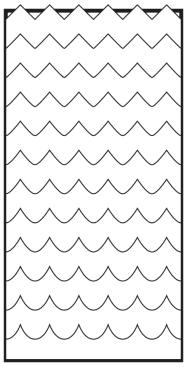
See Code

Wet and Sharp

You can practice designing custom shapes by continuing the 'design a word' exercise from the previous chapters. My assignment to students sound something like this: Make a design with two shapes in black and white that represents the words 'wet' and 'sharp'. There are several reasons why this is a challenging assignment. First of all, the student has to consider how the outline of a shape can help communicate either of those words. Most designs end up using curved vertices to represent wet and simple vertices to represent sharp, but some designs cleverly achieve the goal by doing the opposite. Also, the fact that these shapes exist in the same canvas encourages the student to consider how the shapes can interact with each other to achieve a more dramatic effect. Pointing a knife-like shape directly at a smooth shape will create a certain tension which would not exist if the shape pointed in the other direction.



By Luna Chen. See Code



By Sean McIntyre. See Code

The examples in this chapter have a lot of vertices meticulously defined in code, exactly like you would draw them with the mouse. This is of course not the ultimate promise of algorithmic design. Why make shapes in code when they are faster to draw with a mouse? In the following chapters, we will look at a number of techniques that can be used to draw shapes in a more procedural way.

EXERCISE

Create a design with two shapes in black and white representing the words 'wet' and 'sharp'. The shapes have to be created with the `beginShape()` and `endShape()` functions.

Previous
Basic shapes