

endShape()

Stops adding vertices to a custom shape.

The `beginShape()` and `endShape()` functions allow for creating custom shapes in 2D or 3D. `beginShape()` begins adding vertices to a custom shape and `endShape()` stops adding them.

The first parameter, `mode`, is optional. By default, the first and last vertices of a shape aren't connected. If the constant `CLOSE` is passed, as in `endShape(CLOSE)`, then the first and last vertices will be connected.

The second parameter, `count`, is also optional. In WebGL mode, it's more efficient to draw many copies of the same shape using a technique called `instancing`. The `count` parameter tells WebGL mode how many copies to draw. For example, calling `endShape(CLOSE, 400)` after drawing a custom shape will make it efficient to draw 400 copies. This feature requires [writing a custom shader](#).

After calling `beginShape()`, shapes can be built by calling `vertex()`, `bezierVertex()`, `quadraticVertex()`, and/or `curveVertex()`. Calling `endShape()` will stop adding vertices to the shape. Each shape will be outlined with the current stroke color and filled with the current fill color.

Transformations such as `translate()`, `rotate()`, and `scale()` don't work between `beginShape()` and `endShape()`. It's also not possible to use other shapes, such as `ellipse()` or `rect()`, between `beginShape()` and `endShape()`.

Examples



```
function setup() {
  createCanvas(100, 100);

  background(200);

  // Style the shapes.
  noFill();

  // Left triangle.
  beginShape();
  vertex(20, 20);
  vertex(45, 20);
  vertex(45, 80);
  endShape(CLOSE);

  // Right triangle.
  beginShape();
  vertex(50, 20);
  vertex(75, 20);
  vertex(75, 80);
  endShape();

  describe(
    'Two sets of black lines drawn on a gray background. The three lines on the left form a right triangle. The two lines on the right form a right angle.'
  );
}
```

```
function setup() {
  createCanvas(200, 100);

  background(240);

  noFill();
  stroke(0);

  // Open shape (left)
  beginShape();
  vertex(20, 20);
  vertex(80, 20);
  vertex(80, 80);
  endShape(); // Not closed

  // Closed shape (right)
  beginShape();
  vertex(120, 20);
  vertex(180, 20);
  vertex(180, 80);
  endShape(CLOSE); // Closed

  describe(
    'Two right-angled shapes on a light gray background. The left shape is open with three lines. The right shape is closed, forming a triangle.'
  );
}
```

```
// Note: A "uniform" is a global variable within a shader program.

// Create a string with the vertex shader program.
// The vertex shader is called for each vertex.
let vertSrc = `#version 300 es

precision mediump float;

in vec3 aPosition;
flat out int instanceID;

uniform mat4 uModelViewMatrix;
uniform mat4 uProjectionMatrix;

void main() {

  // Copy the instance ID to the fragment shader.
  instanceID = gl_InstanceID;
  vec4 positionVec4 = vec4(aPosition, 1.0);

  // gl_InstanceID represents a numeric value for each instance.
  // Using gl_InstanceID allows us to move each instance separately.
  // Here we move each instance horizontally by ID * 23.
}
```

Syntax

```
endShape([mode], [count])
```

Parameters

`mode`

Constant: use `CLOSE` to close the shape

`count`

Integer: number of times you want to draw/instance the shape (for WebGL mode).

This page is generated from the comments in [src/core/shape/vertex.js](#). Please feel free to edit it and submit a pull request!

Related References

[beginContour](#)

Begins creating a hole within a flat shape.

[beginShape](#)

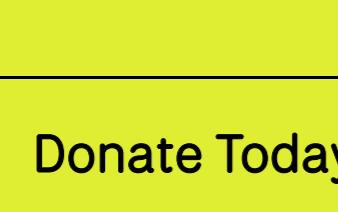
Begins adding vertices to a custom shape.

[bezierVertex](#)

Adds a Bézier curve segment to a custom shape.

[curveVertex](#)

Adds a spline curve segment to a custom shape.



Donate Today! Support p5.js and the Processing Foundation.



Donate Today! Support p5.js and the Processing Foundation.

