computeNormals()

Calculates the normal vector for each vertex on the geometry.

All 3D shapes are made by connecting sets of points called *vertices*. A geometry's surface is formed by connecting vertices to create triangles that are stitched together. Each triangular patch on the geometry's surface is called a *face*. `myGeometry.computeNormals()` performs the math needed to orient each face. Orientation is important for lighting and other effects.

A face's *orientation* is defined by its *normal vector* which points out of the face and is normal (perpendicular) to the surface. Calling `myGeometry.computeNormals()` first calculates each face's normal vector. Then it calculates the normal vector for each vertex by averaging the normal vectors of the faces surrounding the vertex. The vertex normals are stored as p5.Vector objects in the myGeometry.vertexNormals array.

The first parameter, `shadingType`, is optional. Passing the constant `FLAT`, as in `myGeometry.computeNormals(FLAT)`, provides neighboring faces with their own copies of the vertices they share. Surfaces appear tiled with flat shading. Passing the constant `SMOOTH`, as in `myGeometry.computeNormals(SMOOTH)`, makes neighboring faces reuse their shared vertices. Surfaces appear smoother with smooth shading. By default, `shadingType` is `FLAT`.

The second parameter, `options`, is also optional. If an object with a `roundToPrecision` property is passed, as in `myGeometry.computeNormals(SMOOTH, { roundToPrecision: 5 })`, it sets the number of decimal places to use for calculations. By default, `roundToPrecision` uses 5 decimal places.

Examples

```
// Click and drag the mouse to view the scene from differe
angles.

let myGeometry;

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

  // Create a p5.Geometry object.
  beginGeometry();
  torus();
  myGeometry = endGeometry();

  // Compute the vertex normals.
  myGeometry.computeNormals();

  describe(
    'A white torus drawn on a dark gray background. Red lines
extend outward from the torus' vertices.'
  );
}

function draw() {
  background(50);

  // Enable orbiting with the mouse.
  orbitControl();

  // Turn on the lights.
  lights();

  // Rotate the coordinate system.
  rotateX(1);

  // Style the helix.
```

```
// Click and drag the mouse to view the scene from differe
angles.

let myGeometry;

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

  // Create a p5.Geometry object using a callback function.
  myGeometry = new p5.Geometry();

  // Create p5.Vector objects to position the vertices.
  let v0 = createVector(-40, 0, 0);
  let v1 = createVector(0, -40, 0);
  let v2 = createVector(0, 40, 0);
  let v3 = createVector(40, 0, 0);

  // Add the vertices to the p5.Geometry object's vertices
array.
  myGeometry.vertices.push(v0, v1, v2, v3);

  // Compute the faces array.
  myGeometry.computeFaces();

  // Compute the surface normals.
  myGeometry.computeNormals();
```

```
// Click and drag the mouse to view the scene from differe
angles.

let myGeometry;

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

  // Create a p5.Geometry object.
  myGeometry = buildGeometry(createShape);

  // Compute normals using default (FLAT) shading.
  myGeometry.computeNormals(FLAT);

  describe('A white, helical structure drawn on a dark gray
background. Its faces appear faceted.');
}

function draw() {
  background(50);

  // Enable orbiting with the mouse.
  orbitControl();

  // Turn on the lights.
  lights();

  // Rotate the coordinate system.
  rotateX(1);

  // Style the helix.
  noStroke();
```

```
// Click and drag the mouse to view the scene from differe
angles.

let myGeometry;

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

  // Create a p5.Geometry object.
  myGeometry = buildGeometry(createShape);

  // Compute normals using smooth shading.
  myGeometry.computeNormals(SMOOTH);

  describe('A white, helical structure drawn on a dark gray
background.');
}

function draw() {
  background(50);

  // Enable orbiting with the mouse.
  orbitControl();

  // Turn on the lights.
  lights();

  // Rotate the coordinate system.
  rotateX(1);

  // Style the helix.
  noStroke();

  // Display the helix.
```

```
// Click and drag the mouse to view the scene from differe
angles.

let myGeometry;

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

  // Create a p5.Geometry object.
  myGeometry = buildGeometry(createShape);

  // Create an options object.
  let options = { roundToPrecision: 5 };

  // Compute normals using smooth shading.
  myGeometry.computeNormals(SMOOTH, options);

  describe('A white, helical structure drawn on a dark gray
background.');
}

function draw() {
  background(50);

  // Enable orbiting with the mouse.
  orbitControl();

  // Turn on the lights.
  lights();

  // Rotate the coordinate system.
```

Syntax

```
computeNormals([shadingType], [options])
```

Parameters

shadingType	String: shading type. either FLAT or SMOOTH. Defaults to `FLAT`.
options	Object: shading options.

This page is generated from the comments in src/webgl/p5.Geometry.js. Please feel free to edit it and submit a pull request!

Related References

calculateBoundingBox
Calculates the position and size of the smallest box that contains the geometry.

clearColors
Removes the geometry's internal colors.

computeFaces
Computes the geometry's faces using its vertices.

computeNormals
Calculates the normal vector for each vertex on the geometry.