textureWrap()

Changes the way textures behave when a shape's uv coordinates go beyond the texture.

In order for `texture()` to work, a shape needs a way to map the points on its surface to the pixels in an image. Built-in shapes such as `rect()` and `box()` already have these texture mappings based on their vertices. Custom shapes created with `vertex()` require texture mappings to be passed as uv coordinates.

Each call to `vertex()` must include 5 arguments, as in `vertex(x, y, z, u, v)`, to map the vertex at coordinates `(x, y, z)` to the pixel at coordinates `(u, v)` within an image. For example, the corners of a rectangular image are mapped to the corners of a rectangle by default:

```
// Apply the image as a texture.
texture(img);

// Draw the rectangle.
rect(0, 0, 30, 50);
```

If the image in the code snippet above has dimensions of 300 x 500 pixels, the same result could be achieved as follows:

```
// Apply the image as a texture.
texture(img);

// Draw the rectangle.
beginShape();

// Top-left.
// u: 0, v: 0
vertex(0, 0, 0, 0, 0);

// Top-right.
// u: 300, v: 0
vertex(30, 0, 0, 300, 0);

// Bottom-right.
// u: 300, v: 500
vertex(30, 50, 0, 300, 500);

// Bottom-left.
// u: 0, v: 500
vertex(0, 50, 0, 0, 500);

endShape();
```

`textureWrap()` controls how textures behave when their uv's go beyond the texture. Doing so can produce interesting visual effects such as tiling. For example, the custom shape above could have u-coordinates are greater than the image's width:

```
// Apply the image as a texture.
texture(img);

// Draw the rectangle.
beginShape();
vertex(0, 0, 0, 0, 0);

// Top-right.
// u: 600
vertex(30, 0, 0, 600, 0);

// Bottom-right.
// u: 600
vertex(30, 50, 0, 600, 500);

vertex(0, 50, 0, 0, 500);
endShape();
```

The u-coordinates of 600 are greater than the texture image's width of 300. This creates interesting possibilities.

The first parameter, `wrapX`, accepts three possible constants. If `CLAMP` is passed, as in `textureWrap(CLAMP)`, the pixels at the edge of the texture will extend to the shape's edges. If `REPEAT` is passed, as in `textureWrap(REPEAT)`, the texture will tile repeatedly until reaching the shape's edges. If `MIRROR` is passed, as in `textureWrap(MIRROR)`, the texture will tile repeatedly until reaching the shape's edges, flipping its orientation between tiles. By default, textures `CLAMP`.

The second parameter, `wrapY`, is optional. It accepts the same three constants, `CLAMP`, `REPEAT`, and `MIRROR`. If one of these constants is passed, as in `textureWrap(MIRROR, REPEAT)`, then the texture will `MIRROR` horizontally and `REPEAT` vertically. By default, `wrapY` will be set to the same value as `wrapX`.

Note: `textureWrap()` can only be used in WebGL mode.

Examples

```
let img;

function preload() {
  img = loadImage('/assets/rockies128.jpg');
}

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

  describe(
    'An image of a landscape occupies the top-left corner of a square. Its edge colors smear to cover the other thre quarters of the square.'
  );
}

function draw() {
  background(0);

  // Set the texture mode.
  textureMode(NORMAL);

  // Set the texture wrapping.
  // Note: CLAMP is the default mode.
  textureWrap(CLAMP);

  // Apply the image as a texture.
  texture(img);

  // Style the shape.
  noStroke();

  // Draw the shape.
  // Use uv coordinates > 1.
```

```
let img;

function preload() {
  img = loadImage('/assets/rockies128.jpg');
}

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

  describe('Four identical images of a landscape arranged in a grid.');
}

function draw() {
  background(0);

  // Set the texture mode.
  textureMode(NORMAL);

  // Set the texture wrapping.
  textureWrap(REPEAT);

  // Apply the image as a texture.
  texture(img);

  // Style the shape.
  noStroke();

  // Draw the shape.
  // Use uv coordinates > 1.
  beginShape();
  vertex(-30, -30, 0, 0, 0);
  vertex(30, -30, 0, 2, 0);
  vertex(30, 30, 0, 2, 2);
```

```
let img;

function preload() {
  img = loadImage('/assets/rockies128.jpg');
}

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

  describe(
    'Four identical images of a landscape arranged in a grid. The images are reflected horizontally and vertically, creating a kaleidoscope effect.'
  );
}

function draw() {
  background(0);

  // Set the texture mode.
  textureMode(NORMAL);

  // Set the texture wrapping.
  textureWrap(MIRROR);

  // Apply the image as a texture.
  texture(img);

  // Style the shape.
  noStroke();

  // Draw the shape.
  // Use uv coordinates > 1.
  beginShape();
```

```
let img;

function preload() {
  img = loadImage('/assets/rockies128.jpg');
}

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

  describe(
    'Four identical images of a landscape arranged in a grid. The top row and bottom row are reflections of each other.'
  );
}

function draw() {
  background(0);

  // Set the texture mode.
  textureMode(NORMAL);

  // Set the texture wrapping.
  textureWrap(REPEAT, MIRROR);

  // Apply the image as a texture.
  texture(img);

  // Style the shape.
  noStroke();

  // Draw the shape.
  // Use uv coordinates > 1.
  beginShape();
  vertex(-30, -30, 0, 0, 0);
```

Syntax

```
textureWrap(wrapX, [wrapY])
```

Parameters

wrapX Constant: either CLAMP, REPEAT, or MIRROR
wrapY Constant: either CLAMP, REPEAT, or MIRROR

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

Related References

copyToContext
Copies the shader from one drawing context to another.

inspectHooks
Logs the hooks available in this shader, and their current implementation.

modify
Returns a new shader, based on the original, but with custom snippets of shader code replacing default behaviour.

setUniform
Sets the shader's uniform (global) variables.