applyMatrix()

Applies a transformation matrix to the coordinate system.

Transformations such as translate(), rotate(), and scale() use matrix-vector multiplication behind the scenes. A table of numbers, called a matrix, encodes each transformation. The values in the matrix then multiply each point on the canvas, which is represented by a vector.

`applyMatrix()` allows for many transformations to be applied at once. See Wikipedia and MDN for more details about transformations.

There are two ways to call `applyMatrix()` in two and three dimensions.

In 2D mode, the parameters `a`, `b`, `c`, `d`, `e`, and `f`, correspond to elements in the following transformation matrix:

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

The numbers can be passed individually, as in `applyMatrix(2, 0, 0, 0, 2, 0)`. They can also be passed in an array, as in `applyMatrix([2, 0, 0, 0, 2, 0])`.

In 3D mode, the parameters `a`, `b`, `c`, `d`, `e`, `f`, `g`, `h`, `i`, `j`, `k`, `l`, `m`, `n`, `o`, and `p` correspond to elements in the following transformation matrix:

$$\begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{bmatrix}$$

The numbers can be passed individually, as in `applyMatrix(2, 0, 0, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 0, 1)`. They can also be passed in an array, as in `applyMatrix([2, 0, 0, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 0, 1])`.

By default, transformations accumulate. The push() and pop() functions can be used to isolate transformations within distinct drawing groups.

Note: Transformations are reset at the beginning of the draw loop. Calling `applyMatrix()` inside the draw() function won't cause shapes to transform continuously.

Examples

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

  describe('A white circle on a gray background.');
}

function draw() {
  background(200);

  // Translate the origin to the center.
  applyMatrix(1, 0, 0, 1, 50, 50);

  // Draw the circle at coordinates (0, 0).
  circle(0, 0, 40);
}
```

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

  describe('A white circle on a gray background.');
}

function draw() {
  background(200);

  // Translate the origin to the center.
  let m = [1, 0, 0, 1, 50, 50];
  applyMatrix(m);

  // Draw the circle at coordinates (0, 0).
  circle(0, 0, 40);
}
```

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

  describe("A white rectangle on a gray background. The
rectangle's long axis runs from top-left to bottom-right.");
}

function draw() {
  background(200);

  // Rotate the coordinate system 1/8 turn.
  let angle = QUARTER_PI;
  let ca = cos(angle);
  let sa = sin(angle);
  applyMatrix(ca, sa, -sa, ca, 0, 0);

  // Draw a rectangle at coordinates (50, 0).
  rect(50, 0, 40, 20);
}
```

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

  describe(
    'Two white squares on a gray background. The larger square
appears at the top-center. The smaller square appears at the
top-left.'
  );
}

function draw() {
  background(200);

  // Draw a square at (30, 20).
  square(30, 20, 40);

  // Scale the coordinate system by a factor of 0.5.
  applyMatrix(0.5, 0, 0, 0.5, 0, 0);

  // Draw a square at (30, 20).
  // It appears at (15, 10) after scaling.
  square(30, 20, 40);
}
```

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

  describe('A white quadrilateral on a gray background.');
}

function draw() {
  background(200);

  // Calculate the shear factor.
  let angle = QUARTER_PI;
  let shearFactor = 1 / tan(HALF_PI - angle);

  // Shear the coordinate system along the x-axis.
  applyMatrix(1, 0, shearFactor, 1, 0, 0);

  // Draw the square.
  square(0, 0, 50);
}
```

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

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

  describe('A white cube rotates slowly against a gray
background.');
}

function draw() {
  background(200);

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

  // Rotate the coordinate system a little more each frame.
  let angle = frameCount * 0.01;
  let ca = cos(angle);
  let sa = sin(angle);
  applyMatrix(ca, 0, sa, 0, 0, 1, 0, 0, -sa, 0, ca, 0, 0, 0,
0, 1);

  // Draw a box.
  box();
}
```

Syntax

```
applyMatrix(arr)
```

```
applyMatrix(a, b, c, d, e, f)
```

```
applyMatrix(a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p)
```

Parameters

arr	Array: an array containing the elements of the transformation matrix. Its length should be either 6 (2D) or 16 (3D).
a	Number: an element of the transformation matrix.
b	Number: an element of the transformation matrix.
c	Number: an element of the transformation matrix.
d	Number: an element of the transformation matrix.
e	Number: an element of the transformation matrix.
f	Number: an element of the transformation matrix.
g	Number: an element of the transformation matrix.
h	Number: an element of the transformation matrix.
i	Number: an element of the transformation matrix.
j	Number: an element of the transformation matrix.
k	Number: an element of the transformation matrix.
l	Number: an element of the transformation matrix.
m	Number: an element of the transformation matrix.
n	Number: an element of the transformation matrix.
o	Number: an element of the transformation matrix.
p	Number: an element of the transformation matrix.

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

Related References

applyMatrix
Applies a transformation matrix to the coordinate system.

resetMatrix
Clears all transformations applied to the coordinate system.

rotate
Rotates the coordinate system.

rotateX
Rotates the coordinate system about the x-axis in WebGL mode.