这篇文章介绍图像的变换

**基于函数的变换**

如下，我们使用简单的方法进行变换，原理就是重新计算我们顶点xy的位置

<body style="margin: 0px;">

    <canvas id="canvas"></canvas>

    <!-- 顶点着色器 -->

    <script id="vertexShader" type="x-shader/x-vertex">

        // 导出属性

        attribute vec4 a\_Position;

        attribute float a\_PointSize;

        void main(){

            gl\_Position = a\_Position;

            gl\_PointSize = a\_PointSize;

        }

    </script>

    <!-- 片元着色器 -->

    <script id="fragmentShader" type="x-shader/x-fragment">

        // 导出颜色属性

        precision mediump float;

        uniform vec4 u\_FragColor;

        void main() {

            gl\_FragColor = u\_FragColor;

        }

    </script>

    <script>

        ...

        // 设置从缓存取数据

        function setVertexDatas(context, position) {

            // 生成数据缓存

            context.bindBuffer(context.ARRAY\_BUFFER, context.createBuffer());

            // 之前我们是手动给 position 赋值，示例：context.vertexAttrib2f(position, x, y);

            // 现在我们调用该方法，相对于告诉WebGL，画图时你从缓存中取2个值，调用 context.vertexAttrib2f(position, x, y) 取绘图

            // 2表示2个参数，context.FLOAT为参数类型，false  是否将顶点数据归一，0 相邻两个顶点间的字节数，0 从缓冲区的什么位置开始存储变量

            context.vertexAttribPointer(position, 2, context.FLOAT, false, 0, 0);

            // 开启顶点数据的批处理功能

            context.enableVertexAttribArray(position);

        }

        // 平移

        function pan(x, y) {

            return [x + 0.1, y + 0.1];

        }

        // 缩放

        function zoom(x, y) {

            return [x \* 0.9, y \* 0.9];

        }

        // 旋转

        function rotate(x, y) {

            // 旋转弧度

            let radian = 0.1;

            return [

                x\*Math.cos(radian) - y\*Math.sin(radian),

                y\*Math.cos(radian) + x\*Math.sin(radian),

            ];

        }

        ...

        // 获取着色器属性点

        const a\_Position = context.getAttribLocation(context.program, 'a\_Position');

        // 设置顶点从缓存取数据

        setVertexDatas(context, a\_Position);

        // 顶点数据

        let datas = [

            // x, y

            0.0, 0.1,

            -0.1, -0.1,

            0.1, -0.1

        ];

        // 把数据放到缓存中

        context.bufferData(context.ARRAY\_BUFFER, new Float32Array(datas), context.STATIC\_DRAW);

        // 绘制（绘制时顶点会从缓存中取出数据进行绘制）

        context.drawArrays(context.LINE\_LOOP, 0, 3);

        // 重新绘制图形

        setInterval(() => {

            console.log(datas);

            let [ax, ay, bx, by, cx, cy] = datas;

            // 旋转

            [ax, ay] = pan(ax, ay);

            [bx, by] = pan(bx, by);

            [cx, cy] = pan(cx, cy);

            // 缩放

            [ax, ay] = zoom(ax, ay);

            [bx, by] = zoom(bx, by);

            [cx, cy] = zoom(cx, cy);

            // 平移

            [ax, ay] = rotate(ax, ay);

            [bx, by] = rotate(bx, by);

            [cx, cy] = rotate(cx, cy);

            datas = [ax, ay, bx, by, cx, cy];

            context.bufferData(context.ARRAY\_BUFFER, new Float32Array(datas), context.STATIC\_DRAW)

            context.clear(context.COLOR\_BUFFER\_BIT);

            context.drawArrays(context.LINE\_LOOP, 0, 3);

        }, 1000)

    </script>

</body>

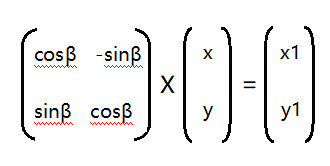
**基于矩阵的变换**

来看一下点旋转的公式

cosβ\*x-sinβ\*y=x1

sinβ\*x+cosβ\*y=y1

该公式可转换为矩阵表示



现在我们使用矩阵去旋转图像

<body style="margin: 0px;">

    <canvas id="canvas"></canvas>

    <!-- 顶点着色器 -->

    <script id="vertexShader" type="x-shader/x-vertex">

        // 导出属性

        attribute vec4 a\_Position;

        // 导出矩阵

        uniform mat4 u\_Matrix;

        attribute float a\_PointSize;

        void main(){

            gl\_Position = u\_Matrix\*a\_Position;

            gl\_PointSize = a\_PointSize;

        }

    </script>

    <!-- 片元着色器 -->

    <script id="fragmentShader" type="x-shader/x-fragment">

        // 导出颜色属性

        precision mediump float;

        uniform vec4 u\_FragColor;

        void main() {

            gl\_FragColor = u\_FragColor;

        }

    </script>

    <script>

        ...

        // canvas画布

        const canvas = document.getElementById('canvas');

        canvas.width = document.body.clientWidth;

        canvas.height = document.body.clientHeight;

        // webgl上下文

        const context = canvas.getContext('webgl');

        // 初始化上下文

        init(context);

        // 指定将要用来清理绘图区的颜色

        context.clearColor(0.0, 0.0, 0.0, 1.0);

        // 清理绘图区

        context.clear(context.COLOR\_BUFFER\_BIT);

        // 获取着色器属性点

        const a\_Position = context.getAttribLocation(context.program, 'a\_Position');

        const a\_PointSize = context.getAttribLocation(context.program, 'a\_PointSize');

        const u\_FragColor = context.getUniformLocation(context.program, 'u\_FragColor');

        // 导出矩阵属性

        const u\_Matrix = context.getUniformLocation(context.program, 'u\_Matrix');

        // 设置顶点大小

        context.vertexAttrib1f(a\_PointSize, 0);

        // 设置绘图颜色

        context.uniform4f(u\_FragColor, 1, 1, 1, 1.0);

        // 设置顶点从缓存取数据

        setVertexDatas(context, a\_Position);

        // 顶点数据

        let datas = [

            // x, y

            0.0, 0.1,

            -0.1, -0.1,

            0.1, -0.1

        ];

        // 把顶点数据放到缓存中

        context.bufferData(context.ARRAY\_BUFFER, new Float32Array(datas), context.STATIC\_DRAW);

        // 旋转角度

        let angle = 0;

        // 旋转矩阵

        let matrix = [

            1.0, 0, 0, 0,

            0, 1.0, 0, 0,

            0, 0, 1.0, 0,

            0, 0, 0, 1.0

        ];

        // 重新绘制图形

        setInterval(() => {

            // 旋转角度

            angle = angle + 0.1;

            // 设置二维旋转矩阵

            const sin = Math.sin(angle);

            const cos = Math.cos(angle);

            // 为何这里的矩阵反转过来了？

            // 因为WebGL使用列主序的矩阵，所以这里我们需要转置一下矩阵

            matrix = [

                cos, sin, 0, 0,

                -sin, cos, 0, 0,

                0, 0, 1.0, 0,

                0, 0, 0, 1.0

            ]

            // 设置旋转的矩阵

            context.uniformMatrix4fv(u\_Matrix, false, matrix);

            context.clear(context.COLOR\_BUFFER\_BIT);

            context.drawArrays(context.LINE\_LOOP, 0, 3);

        }, 1000)

    </script>

</body>

- 顶点着色器代码分析

    <!-- 顶点着色器 -->

    <script id="vertexShader" type="x-shader/x-vertex">

        // 导出属性

        attribute vec4 a\_Position;

        // 导出矩阵

        uniform mat4 u\_Matrix;

        attribute float a\_PointSize;

        void main(){

            gl\_Position = u\_Matrix\*a\_Position;

            gl\_PointSize = a\_PointSize;

        }

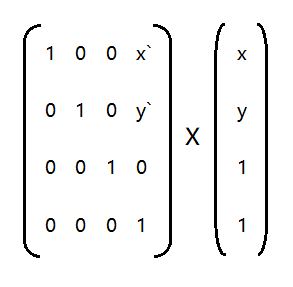
    </script>

这里导出了矩阵属性

并且顶点的坐标等于矩阵\*原坐标（gl\_Position = u\_Matrix\*a\_Position;）

- 移动点的“矩阵”该怎么写

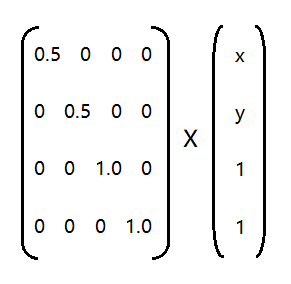
将点p(x,y) 的x移动x`，y移动y`，这个矩阵该怎么写呢？如下



注：记得把矩阵转置

- 缩放矩阵该怎么写？

如下我们将x，y缩放0.5



**基于Three.js的矩阵**

我们不一定要自己手动创建矩阵，如下我们通过threejs生成矩阵

- 引入js

<script src="http://www.yanhuangxueyuan.com/versions/threejsR92/build/three.js"></script>

- 修改代码

        // 旋转角度

        let angle = 0;

        // 旋转矩阵

        let matrix;

        // 重新绘制图形

        setInterval(() => {

            // 旋转角度

            angle = angle + 0.1;

            // 创建一个4维矩阵

            matrix = new THREE.Matrix4();

            // 将矩阵沿着z轴旋转angle弧度

            matrix.makeRotationZ(angle);

            // 设置旋转的矩阵

            context.uniformMatrix4fv(u\_Matrix, false, matrix.elements);

            context.clear(context.COLOR\_BUFFER\_BIT);

            context.drawArrays(context.LINE\_LOOP, 0, 3);

        }, 1000)

- 旋转、平移、缩放函数

// 旋转

matrix.makeRotationX(angle);

matrix.makeRotationY(angle);

matrix.makeRotationZ(angle);

// 移动

matrix.makeTranslation(x,y,z);

// 缩放

matrix.makeScale(x,y,x);

**复合变换**

我们将点p先旋转，再平移，该怎么实现呢？

旋转矩阵\*p=旋转后的p1

平移矩阵\*p1=平移后的p2

所以这个问题的解为 平移矩阵\*旋转矩阵\*p（注：不要把位置搞混了）

现在我们使用threejs实现先旋转后移动后的图形

        // 旋转角度

        let angle = 0;

        // 平移x

        let x = 0;

        // 旋转矩阵

        let mr;

        // 平移矩阵

        let mt;

        // 重新绘制图形

        setInterval(() => {

            // 旋转角度

            angle = angle + 0.1;

            x = x + 0.1;

            // 将矩阵沿着z轴旋转angle弧度

            mr = new THREE.Matrix4();

            mr.makeRotationZ(angle);

            // 将矩阵平移

            mt = new THREE.Matrix4();

            mt.makeTranslation(x, 0, 0);

            // 矩阵相乘，先旋转再平移

            const matrix = mt.multiply(mr);

            // 设置旋转的矩阵

            context.uniformMatrix4fv(u\_Matrix, false, matrix.elements);

            context.clear(context.COLOR\_BUFFER\_BIT);

            context.drawArrays(context.LINE\_LOOP, 0, 3);

        }, 1000);