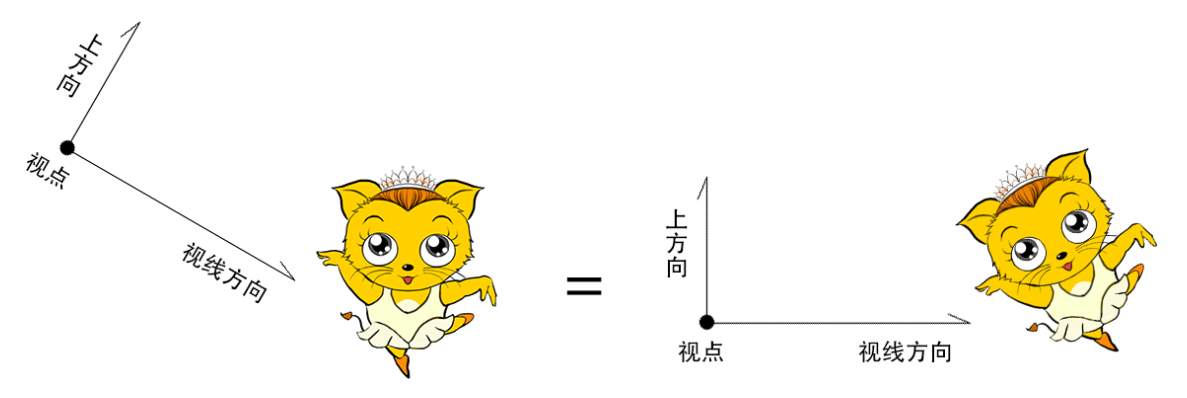
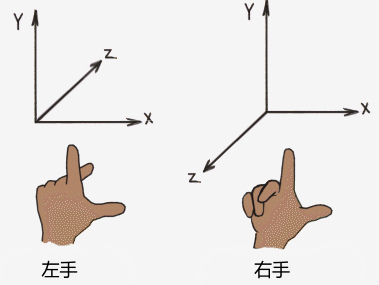
本片文章需要先了解视图矩阵知识，如果不了解可以到 线性代数 -> 视图矩阵 查看文档

WebGL有2个坐标系，一个是世界坐标系xyz，表明了物体所在的位置，一个则是观察者坐标系（就是你电脑屏幕的中心点为原点，向右为x`，向上为y`，向电脑里面为z`）

如下，再世界坐标系中，这个小人是站立的，但在观察者坐标系中，这个小人是倾斜的

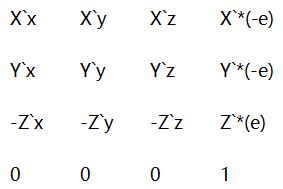


世界坐标系使用右手坐标系，观察者坐标系使用左手坐标系



程序渲染图像是根据观察者坐标系进行渲染的，也就是说我们要将世界坐标系的物体坐标转为观察者坐标系的物体坐标

我们知道视角转换矩阵如下



现在问题是如何算出单位向量X`，Y`，Z`的坐标点，直接给出X`，Y`，Z`的坐标不合理，正常情况下我们给出的是，e（视点坐标，就是原点坐标），t（所看向的目标点坐标），u（上方向坐标），这3个坐标表示的就是一个游戏角色所在的位置e，看下目标t，u指明角色是正着看还是斜着看还是倒着看，根据这3个坐标我们可以求出X`，Y`，Z`的坐标点

我们不用自己求，直接使用Threejs的方法计算

        // 获取视图转换矩阵

        // e (eye)为视点坐标

        // t (target)为目标点坐标

        // u (up)为上方向坐标

        function getViewMatrix(e, t, u) {

            //基向量z

            const z = new THREE.Vector3().subVectors(e, t).normalize()

            //基向量x

            const x = new THREE.Vector3().crossVectors(u, z).normalize()

            //基向量y

            const y = new THREE.Vector3().crossVectors(z, x).normalize()

            //旋转矩阵

            const mr = new THREE.Matrix4().set(

                x.x, x.y, x.z, 0,

                y.x, y.y, y.z, 0,

                -z.x, -z.y, -z.z, 0,

                0, 0, 0, 1

            )

            //位移矩阵

            const mt = new THREE.Matrix4().set(

                1, 0, 0, -e.x,

                0, 1, 0, -e.y,

                0, 0, 1, -e.z,

                0, 0, 0, 1

            )

            return mr.multiply(mt);

        }

下面我们改变视角去查看我们的三角型吧，代码如下

<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;

        // 导出视图矩阵

        uniform mat4 u\_ViewMatrix;

        attribute float a\_PointSize;

        void main(){

            gl\_Position = u\_ViewMatrix\*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>

        ...

        // 获取视图转换矩阵

        // e (eye)为视点坐标

        // t (target)为目标点坐标

        // u (up)为上方向坐标

        function getViewMatrix(e, t, u) {

            //基向量z

            const z = new THREE.Vector3().subVectors(e, t).normalize()

            //基向量x

            const x = new THREE.Vector3().crossVectors(u, z).normalize()

            //基向量y

            const y = new THREE.Vector3().crossVectors(z, x).normalize()

            //旋转矩阵

            const mr = new THREE.Matrix4().set(

                x.x, x.y, x.z, 0,

                y.x, y.y, y.z, 0,

                -z.x, -z.y, -z.z, 0,

                0, 0, 0, 1

            )

            //位移矩阵

            const mt = new THREE.Matrix4().set(

                1, 0, 0, -e.x,

                0, 1, 0, -e.y,

                0, 0, 1, -e.z,

                0, 0, 0, 1

            )

            return mr.multiply(mt);

        }

        // 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');

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

        // 视图矩阵

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

        // 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);

        // 物体运动矩阵

        let objm = new THREE.Matrix4();

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

        // 生成视图矩阵

        let viewMatrix = getViewMatrix(

            new THREE.Vector3(0.3, 0.2, 0),

            new THREE.Vector3(0.0, 0.1, 0),

            new THREE.Vector3(0, 1, 0)

        );

        // 设置视图矩阵

        context.uniformMatrix4fv(u\_ViewMatrix, false, viewMatrix.elements);

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

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

    </script>

</body>

知识点

- 导出视图矩阵

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

        // 导出属性

        attribute vec4 a\_Position;

        // 导出矩阵

        uniform mat4 u\_Matrix;

        // 导出视图矩阵

        uniform mat4 u\_ViewMatrix;

        attribute float a\_PointSize;

        void main(){

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

            gl\_PointSize = a\_PointSize;

        }

    </script>

- 根据e、t、u 获取X`Y`Z`坐标

            //基向量z

            const z = new THREE.Vector3().subVectors(e, t).normalize()

            //基向量x

            const x = new THREE.Vector3().crossVectors(u, z).normalize()

            //基向量y

            const y = new THREE.Vector3().crossVectors(z, x).normalize()

subVectors(e, t) 向量e减向量t

normalize() 向量的归一化

crossVectors(u, d) 向量u 和向量d的叉乘

- Three生成视图矩阵方法

如下是我们自定义的生成视图矩阵的方法

        // 生成视图矩阵

        let viewMatrix = getViewMatrix(

            new THREE.Vector3(0.3, 0.2, 0),

            new THREE.Vector3(0.0, 0.1, 0),

            new THREE.Vector3(0, 1, 0)

        );

但实际上Three有提供对应的方法，我们不用自己定义（但好像Three算出的视图和预想的不一样）

        const viewMatrix = new Matrix4().lookAt(

            new THREE.Vector3(0.3, 0.2, 0),

            new THREE.Vector3(0.0, 0.1, 0),

            new THREE.Vector3(0, 1, 0)

        )