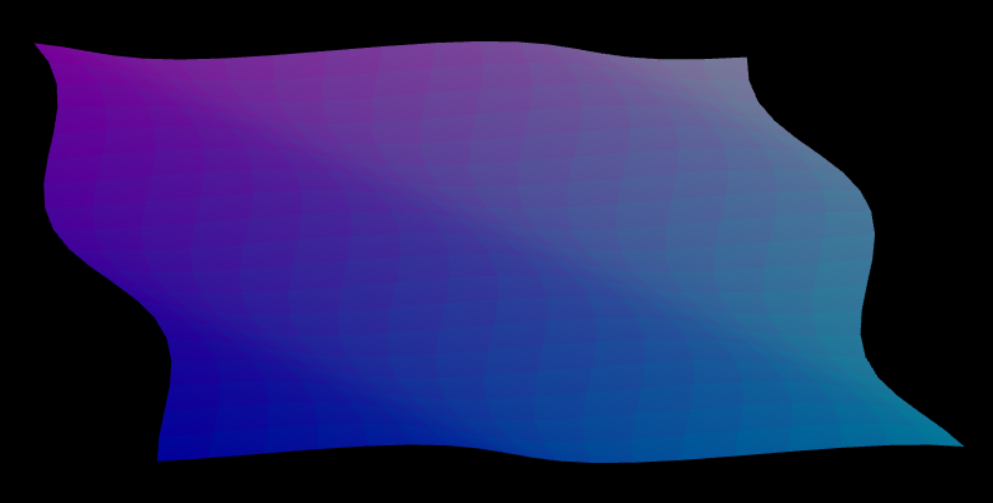
今天做一下如下的波浪效果



1. 改变视角

首先我们改变一下我们的视角

        // 视图矩阵

        const viewMatrix = new THREE.Matrix4().lookAt(

            new THREE.Vector3(-0.2, 1.0, 0.3),

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

            new THREE.Vector3(0, 1, 0)

        )

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

1. 绘制点

接着我们绘制一个20\*20的点

        function buildSource() {

            let points = [];

            let startx = -0.5;

            let startz = -0.5;

            let step = 0.05;

            for (let row = 0; row < 20; row++) {

                let z = startz + row \* step;

                for (let col = 0; col < 20; col++) {

                    let x = startx + col \* step;

                    points = points.concat([

                        x, 0, z,

                        0, 0, 0.6, 1]);

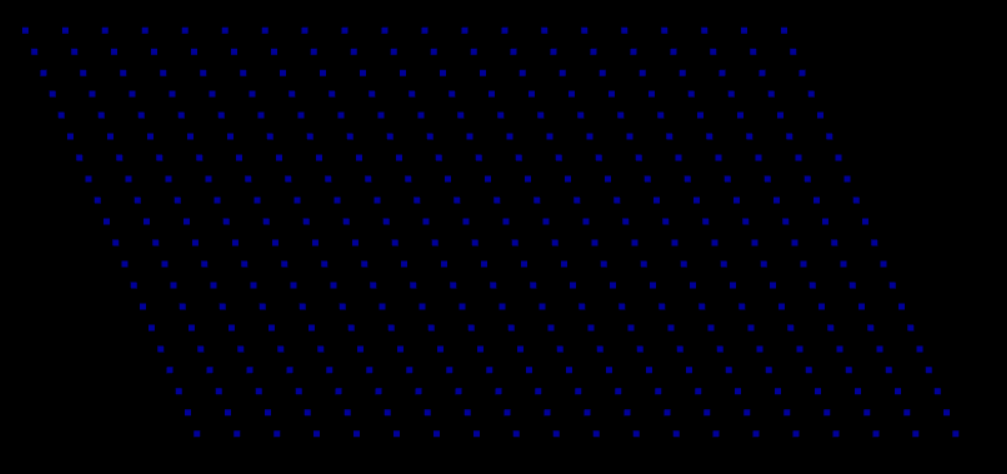
                }

            }

            return points;

        }

效果如下

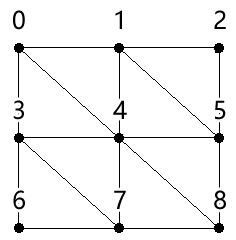


1. 绘制面

绘制面我们使用的是TRIANGLES（独立三角形）进行绘制，当你将类型改为TRIANGLES是会发现只有线，没有面，为什么呢？因为TRIANGLES是根据顶点1、顶点2、顶点3绘制为一个3角形，但顶点123都在同一条直线上，所以就变成了直线



现在我们思考一下如何绘制面，如何绘制(0,3,4,1)这个4边形面呢？要绘制这个4变形需要2个三角形，(0,3,4)和(0,4,1)，也就是说，在0这个位置，实际上我们应该插入6个点数据，这分别是(0,3,4,0,4,1)。同理，在1这个位置我们需要插入6个点数据(1,4,5,1,5,2)。位置2不需要插入数据，位置3、4同理6个点



所以我们的算法变成如下

        function buildSource() {

            let points = [];

            let startx = -0.5;

            let startz = -0.5;

            let step = 0.05;

            for (let row = 0; row < 19; row++) {

                let z = startz + row \* step;

                for (let col = 0; col < 19; col++) {

                    let x = startx + col \* step;

                    points = points.concat([

                        x, 0, z,

                        0, 0, 0.6, 1]);

                    points = points.concat([

                        x, 0, z + step,

                        0, 0, 0.6, 1]);

                    points = points.concat([

                        x + step, 0, z + step,

                        0, 0, 0.6, 1]);

                    points = points.concat([

                        x, 0, z,

                        0, 0, 0.6, 1]);

                    points = points.concat([

                        x + step, 0, z + step,

                        0, 0, 0.6, 1]);

                    points = points.concat([

                        x + step, 0, z,

                        0, 0, 0.6, 1]);

                }

            }

            return points;

        }

注：为什么这么奇葩，根据3个点绘制3角形，为什么不能根据4个点绘制4边形？因为3角形的3个点一定在一个面上，但4边形则不是

1. 颜色渐变

如下改进算法

        function buildSource() {

            let points = [];

            let startx = -0.5;

            let startz = -0.5;

            let step = 0.05;

            // 颜色步进

            let colorStep = 0.025;

            for (let row = 0; row < 19; row++) {

                let z = startz + row \* step;

                // row 控制颜色 r

                let colorR = row \* colorStep;

                for (let col = 0; col < 19; col++) {

                    let x = startx + col \* step;

                    // col 控制颜色 g

                    let colorG = col \* colorStep;

                    points = points.concat([

                        x, 0, z,

                        colorR, colorG, 0.6, 1]);

                    points = points.concat([

                        x, 0, z + step,

                        colorR, colorG + colorStep, 0.6, 1]);

                    points = points.concat([

                        x + step, 0, z + step,

                        colorR + colorStep, colorG + colorStep, 0.6, 1]);

                    points = points.concat([

                        x, 0, z,

                        colorR, colorG, 0.6, 1]);

                    points = points.concat([

                        x + step, 0, z + step,

                        colorR + colorStep, colorG + colorStep, 0.6, 1]);

                    points = points.concat([

                        x + step, 0, z,

                        colorR + colorStep, colorG, 0.6, 1]);

                }

            }

            return points;

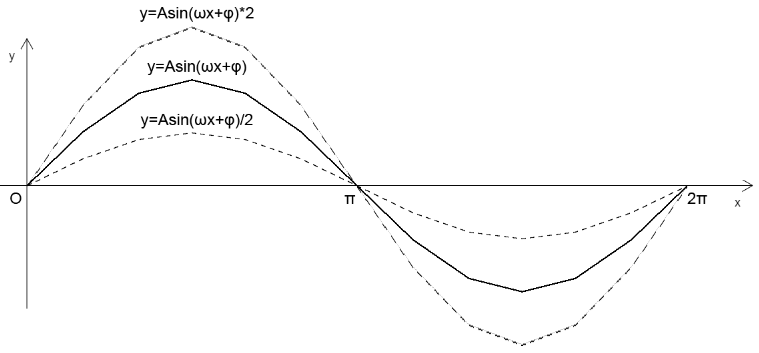
        }

效果如下



1. 波动效果

我们通过控制点在y上的坐标就可以实现波动效果，波函数就是sin函数，sin函数对弧度有一个上线波动



如下是我们计算y的方式

y = 0.06 \* Math.sin(radian + 10 \* (x + z))

radian：基础弧度

10 \* (x + z)：控制xz坐标对弧度的影响

0.06：控制波动的高度

所以最终我们的的函数变成了

function buildSource(radian) {

            let points = [];

            let startx = -0.5;

            let startz = -0.5;

            let step = 0.05;

            // 颜色步进

            let colorStep = 0.025;

            for (let row = 0; row < 19; row++) {

                let z = startz + row \* step;

                // row 控制颜色 r

                let colorR = row \* colorStep;

                for (let col = 0; col < 19; col++) {

                    let x = startx + col \* step;

                    // col 控制颜色 g

                    let colorG = col \* colorStep;

                    let y;

                    // 计算点1的y值

                    y = 0.06 \* Math.sin(radian + 10 \* (x + z));

                    points = points.concat([

                        x, y, z,

                        colorR, colorG, 0.6, 1]);

                    // 计算点2的y值

                    y = 0.06 \* Math.sin(radian + 10 \* (x + z + step));

                    points = points.concat([

                        x, y, z + step,

                        colorR, colorG + colorStep, 0.6, 1]);

                    y = 0.06 \* Math.sin(radian + 10 \* (x + step + z + step));

                    points = points.concat([

                        x + step, y, z + step,

                        colorR + colorStep, colorG + colorStep, 0.6, 1]);

                    y = 0.06 \* Math.sin(radian + 10 \* (x + z));

                    points = points.concat([

                        x, y, z,

                        colorR, colorG, 0.6, 1]);

                    y = 0.06 \* Math.sin(radian + 10 \* (x + step + z + step));

                    points = points.concat([

                        x + step, y, z + step,

                        colorR + colorStep, colorG + colorStep, 0.6, 1]);

                    y = 0.06 \* Math.sin(radian + 10 \* (x + step + z));

                    points = points.concat([

                        x + step, y, z,

                        colorR + colorStep, colorG, 0.6, 1]);

                }

            }

            return points;

        }

1. 波动动画

最后我们加上一个setInterval让它动起来，这样波浪就完成了

        let radian = 1;

        setInterval(() => {

            radian = radian + 0.05;

            const points = buildSource(radian);

            const source = new Float32Array(points);

            //写入数据

            context.bufferData(context.ARRAY\_BUFFER, source, context.STATIC\_DRAW);

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

            context.drawArrays(context.TRIANGLES, 0, points.length / 7);

        }, 50);

**整体代码**

如下是整体代码

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

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

</head>

<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;

        // 导出大小

        attribute float a\_PointSize;

        // 导出颜色

        attribute vec4 a\_Color;

        // 指定一个全局变量

        varying vec4 v\_Color;

        void main(){

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

            gl\_PointSize = a\_PointSize;

            // 将全局变量指向a\_Color

            v\_Color = a\_Color;

        }

    </script>

    <!-- 片元着色器 -->

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

        // 导出颜色属性

        precision mediump float;

        // 指定一个全局变量

        varying vec4 v\_Color;

        void main() {

            // 将gl\_FragColor指向v\_Color

            gl\_FragColor = v\_Color;

        }

    </script>

    <script>

        // 初始化

        function init(context) {

            //创建程序对象

            const program = context.createProgram();

            //创建顶点着色器

            const vertexShader = context.createShader(context.VERTEX\_SHADER);

            // 获取顶点着色器代码

            const vsSource = document.getElementById('vertexShader').innerText;

            //将着色器源文件传入着色器对象中

            context.shaderSource(vertexShader, vsSource);

            //编译着色器对象

            context.compileShader(vertexShader);

            //把顶点着色对象装进程序对象中

            context.attachShader(program, vertexShader);

            //创建片元着色器

            const fragmentShader = context.createShader(context.FRAGMENT\_SHADER);

            // 获取片元着色器代码

            const fsSource = document.getElementById('fragmentShader').innerText;

            //将着色器源文件传入着色器对象中

            context.shaderSource(fragmentShader, fsSource);

            //编译着色器对象

            context.compileShader(fragmentShader)

            //把片元着色对象装进程序对象中

            context.attachShader(program, fragmentShader);

            //连接webgl上下文对象和程序对象

            context.linkProgram(program);

            //启动程序对象

            context.useProgram(program);

            //将程序对象挂到上下文对象上

            context.program = program;

        }

        // 生成顶点数据

        // radian 弧度

        function buildSource(radian) {

            let points = [];

            let startx = -0.5;

            let startz = -0.5;

            let step = 0.05;

            // 颜色步进

            let colorStep = 0.025;

            for (let row = 0; row < 19; row++) {

                let z = startz + row \* step;

                // row 控制颜色 r

                let colorR = row \* colorStep;

                for (let col = 0; col < 19; col++) {

                    let x = startx + col \* step;

                    // col 控制颜色 g

                    let colorG = col \* colorStep;

                    let y;

                    // 计算点1的y值

                    y = 0.06 \* Math.sin(radian + 10 \* (x + z));

                    points = points.concat([

                        x, y, z,

                        colorR, colorG, 0.6, 1]);

                    // 计算点2的y值

                    y = 0.06 \* Math.sin(radian + 10 \* (x + z + step));

                    points = points.concat([

                        x, y, z + step,

                        colorR, colorG + colorStep, 0.6, 1]);

                    y = 0.06 \* Math.sin(radian + 10 \* (x + step + z + step));

                    points = points.concat([

                        x + step, y, z + step,

                        colorR + colorStep, colorG + colorStep, 0.6, 1]);

                    y = 0.06 \* Math.sin(radian + 10 \* (x + z));

                    points = points.concat([

                        x, y, z,

                        colorR, colorG, 0.6, 1]);

                    y = 0.06 \* Math.sin(radian + 10 \* (x + step + z + step));

                    points = points.concat([

                        x + step, y, z + step,

                        colorR + colorStep, colorG + colorStep, 0.6, 1]);

                    y = 0.06 \* Math.sin(radian + 10 \* (x + step + z));

                    points = points.concat([

                        x + step, y, z,

                        colorR + colorStep, colorG, 0.6, 1]);

                }

            }

            return points;

        }

        // 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\_Color = context.getAttribLocation(context.program, 'a\_Color');

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

        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, 5);

        //缓冲对象

        const sourceBuffer = context.createBuffer();

        //绑定缓冲对象

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

        // 设置顶点从缓存取数据

        const points = buildSource(1);

        const source = new Float32Array(points);

        //修改attribute 变量

        context.vertexAttribPointer(

            // 要设置的指针

            a\_Position,

            // 每次取数据的个数

            3,

            // 数据的类型

            context.FLOAT,

            // 是否归一化

            false,

            // 一行数据的字节数，我们一行又7个数据，Float32是4个字节，所以一行数据的字节数为 7\*4

            7 \* 4,

            // 每行数据跳过的自己数

            // a\_Position是从第0个数据开始取的，所以这里填0\*4

            0 \* 4

        );

        //赋能-批处理

        context.enableVertexAttribArray(a\_Position);

        //修改attribute 变量

        context.vertexAttribPointer(

            a\_Color,

            4,

            context.FLOAT,

            false,

            7 \* 4,

            3 \* 4

        )

        //赋能-批处理

        context.enableVertexAttribArray(a\_Color);

        //写入数据

        context.bufferData(context.ARRAY\_BUFFER, source, context.STATIC\_DRAW);

        // 物体运动矩阵

        let objm = new THREE.Matrix4();

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

        // 视图矩阵

        const viewMatrix = new THREE.Matrix4().lookAt(

            new THREE.Vector3(-0.2, 1.0, 0.3),

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

            new THREE.Vector3(0, 1, 0)

        )

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

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

        context.drawArrays(context.TRIANGLES, 0, points.length / 7);

        let radian = 1;

        setInterval(() => {

            radian = radian + 0.05;

            const points = buildSource(radian);

            const source = new Float32Array(points);

            //写入数据

            context.bufferData(context.ARRAY\_BUFFER, source, context.STATIC\_DRAW);

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

            context.drawArrays(context.TRIANGLES, 0, points.length / 7);

        }, 50);

    </script>

</body>

</html>