郭隆邦 2023-10-17

3. 第一个WebGL案例

本节课给大家演示一个WebGL案例，就是在web页面上绘制一个方形点，虽然超级简单，但是可以把WebGL整个的代码结构，给你完美展示出来。

创建一个Canvas画布

创建一个Canvas画布，用于显示WebGL的渲染结果，canvas元素和div等元素一样是HTML的一个元素，区别地方在于，Canvas画布具有2D和3D绘图功能。

```js
// 宽高度
<canvas id="webgl" width="500" height="500" style="background: #000;"></canvas>
```

创建一个html文件，在里面body中插入一个canvas元素。

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>使用WebGL绘制一个点</title>
</head>
<body>
    <canvas id="webgl" width="500" height="500" style="background: #000;"></canv
</body>
</html>
```

获取webgl上下文

```html
<canvas id="webgl"></canvas>
```

通过 `getElementById()` 方法获取canvas画布对象

```html
<script>
    const canvas= document.getElementById('webgl')
</script>
```

通过方法`.getContext()`获取WebGL上下文,然后你就可以通过返回的对象`gl`调用WebGL API，实现3D绘图。

```js
const gl = canvas.getContext('webgl');
```

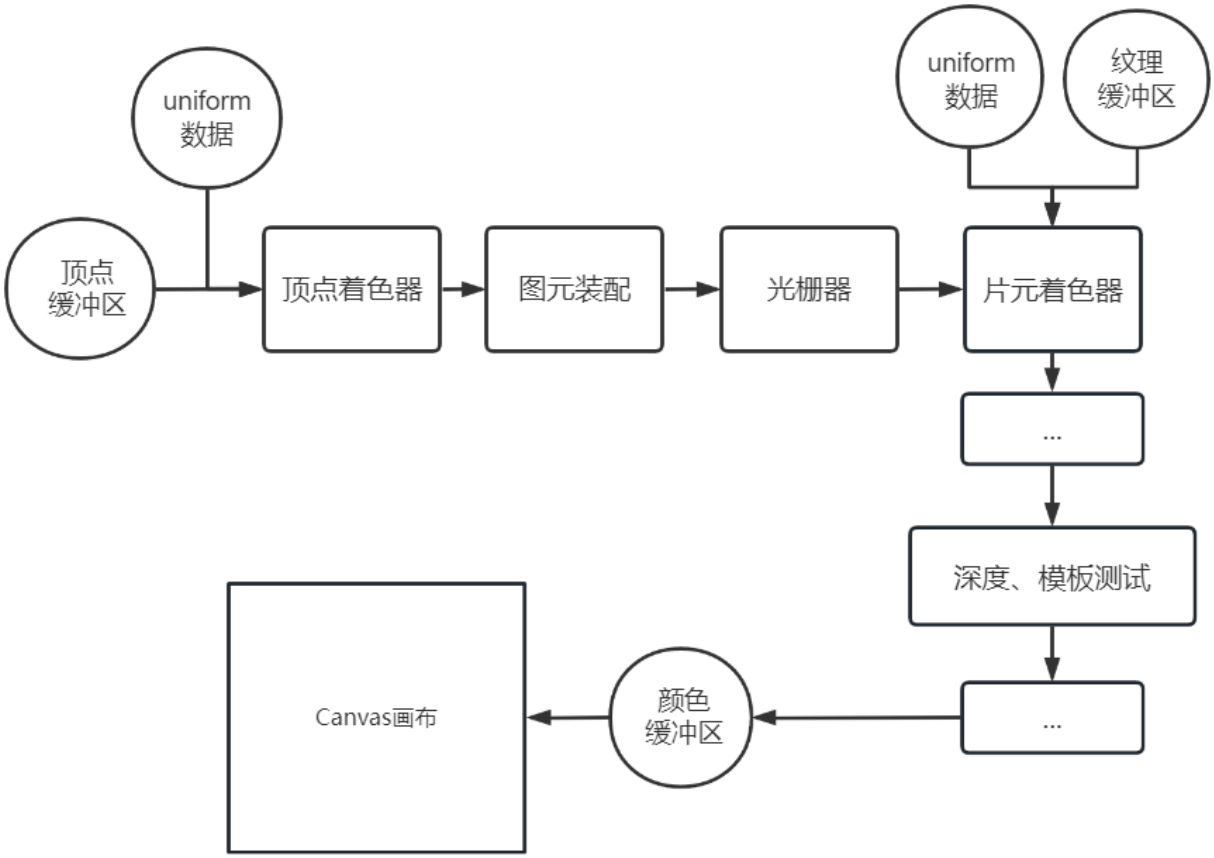
通过gl对象，可以调用各种WebGL API,通过这些WebGL API可以控制显卡GPU绘制3D图案。

```js
const gl = canvas.getContext('webgl');
gl.drawArrays();
gl.createShader();
gl.shaderSource();
gl.compileShader();
gl.createProgram();
gl.attachShader();
gl.linkProgram();
gl.useProgram();
```

渲染管线概念

入门WebGL，比较重要的一点就是建立**渲染管线**的概念。

你可以把**渲染管线**想象为显卡GPU上的一条流水线，渲染管线上有不同的功能单元。WebGL **渲染管线**上的各个功能单元，可以通过刚刚介绍的通过`WebGL API`进行控制。



整个**渲染管线**，是比较复杂的，本节课只要求知道渲染管线这个概念就行，更多具体细节，在下来WebGL课程中，会一一介绍。

顶点着色器

编写顶点着色器需要用到一门语言，就是前面提到的着色器语言GLSL ES。

在js代码中，着色器GLSL代码，要使用字符串的形式表达。为了方便预览顶点着色器代码，咱们用模板字符串 `` 的形式去写,模板字符串 `` 的按键位于键盘Tab键的上面。

```js
const vertexShaderSource = `
这里面写着色器GLSL ES代码
`
```

按照着色器语言习惯，创建一个名为 `main` 的主函数，前面使用关键字 `void` 类似C语言的语法，表示没有返回值。

```js
const vertexShaderSource = `
void main(){
```

```
}
`
```

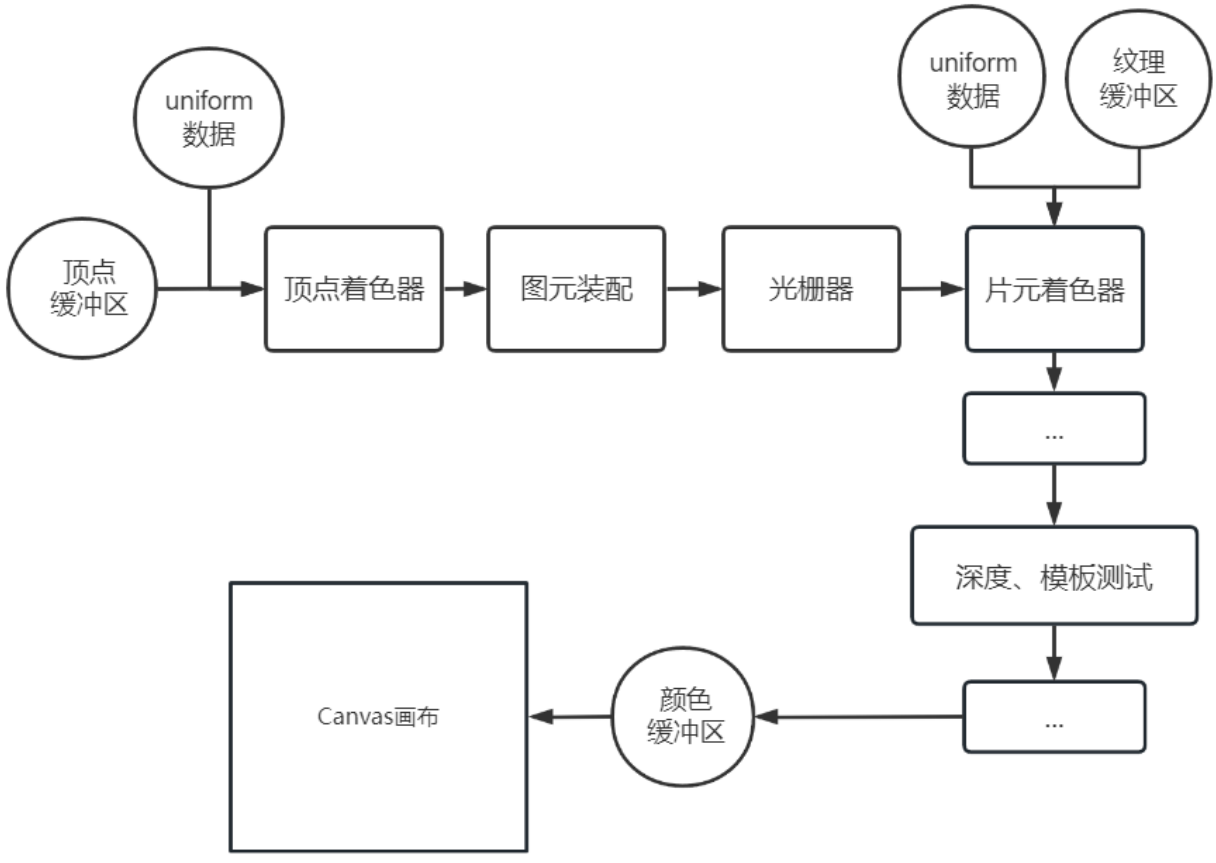
`gl_PointSize` 和 `gl_Position` 是内置变量，所谓内置变量，就是不用声明就可以使用。

`gl_PointSize` 表示渲染点的像素大小，注意用小数(浮点数)表示。

`gl_Position` 表示顶点的位置，值是四维向量vec4,比如表示 `(x,y,z)` 坐标，书写形式是 `vec4(x,y,z,1.0)` ,按照语法规定，前面三个参数表示xyz坐标，最后一个参数是1.0。

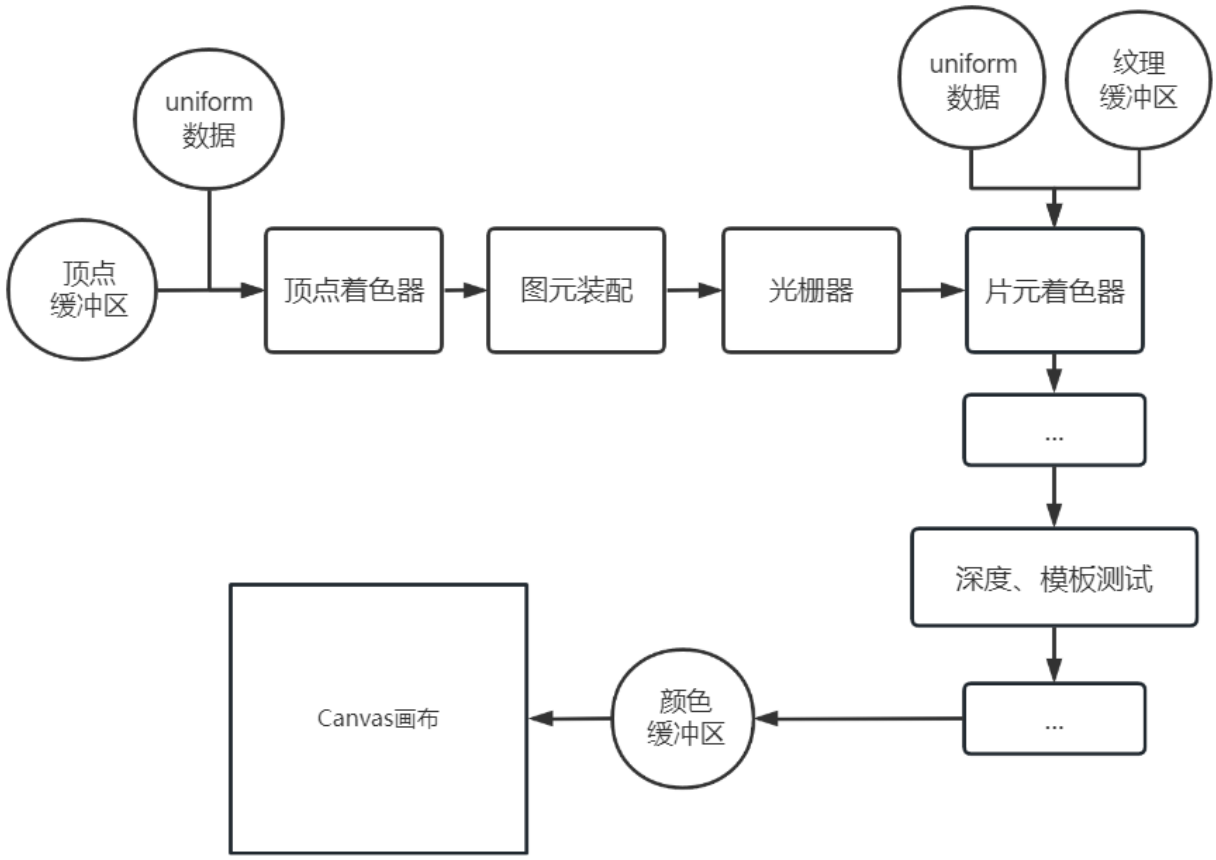
```js
//顶点着色器源码
const vertexShaderSource = `
void main(){
    gl_PointSize = 20.0;
    gl_Position = vec4(0.0,0.0,0.0,1.0);
}
`
```

顶点着色器代码就是在GPU上的**顶点着色器**功能单元执行。



片元着色器

片元着色器代码在GPU上的片元着色器功能单元执行。



内置变量`gl_FragColor`用来设置片元(像素)颜色，以为本节课为例就是方形点的像素值。

`gl_FragColor`的值是四维向量vec4，前面三个参数是颜色RGB值，第四个参数是透明度值，第一次学习先给一个1.0表示不透明就行。

```js
//片元着色器源码
const fragShaderSource = `
void main(){
    // 红色
    gl_FragColor = vec4(1.0,0.0,0.0,1.0);
}
`
```

编译着色器，并创建程序对象

顶点着色器、片元着色器代码如果想在GPU上执行，需要先通过WebGL API进行编译处理，并创建一个程序对象 `program` 。

学习建议：对于大部分同学学习的重点是**着色器语言GLSL和渲染管线。**如果你不是直接使用原生WebGL API做项目，或者封装3D引擎，你完全不需要记住这些API，只需要有个印象就行。

对于下面代码，你只需要跟着课程过一遍就行，不要求记住，甚至复制粘贴一遍也无所谓。

```js
//创建顶点着色器对象
const vertexShader = gl.createShader(gl.VERTEX_SHADER);
//创建片元着色器对象
const fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
//引入顶点、片元着色器源代码
gl.shaderSource(vertexShader, vertexShaderSource);
gl.shaderSource(fragmentShader, fragmentShaderSource);
//编译顶点、片元着色器
gl.compileShader(vertexShader);
gl.compileShader(fragmentShader)
//创建程序对象program
const program = gl.createProgram();
//附着顶点着色器和片元着色器到program
gl.attachShader(program, vertexShader);
gl.attachShader(program, fragmentShader);
//链接program
gl.linkProgram(program);
//使用program
gl.useProgram(program);
```

绘制 `gl.drawArrays()`

通过程序对象program处理好上面着色器代码，你在网页上还不能看到渲染效果，还需要通过一个绘制的API `gl.drawArrays()` 来执行绘制。

`gl.drawArrays()` 在后面会经常用到，因为本节课例子太过简单，只能简单解释下 `gl.drawArrays()` 的参数。

文档gl.drawArrays()

```js
gl.drawArrays(mode, first, count);
```

参数1 `mode` 表示绘制模式，有多种模式， `gl.POINTS` 表示绘制形式是点，后面还会介绍线和三角形。

参数2 `first` 表示从第几个点开始绘制，本节课案例比较简单，只有一个点，设置0即可

参数3 `count` 表示总共有多少点，本节课案例只提了一个点的坐标，所以设置为1。

```js
//开始绘制，显示器显示结果
gl.drawArrays(gl.POINTS, 0, 1);
```

封装一个着色器初始化函数

初始化着色器的代码，可以封装在一个函数 `initShader` 中，因为后面每节课的案例，都会用到这段固定的代码。

```js
//初始化着色器
const program = initShader(gl, vertexShaderSource, fragShaderSource);
//开始绘制，显示器显示结果
gl.drawArrays(gl.POINTS, 0, 1);
//声明初始化着色器函数
function initShader(gl, vertexShaderSource, fragmentShaderSource) {
    //创建顶点着色器对象
    const vertexShader = gl.createShader(gl.VERTEX_SHADER);
    //创建片元着色器对象
    const fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
    //引入顶点、片元着色器源代码
    gl.shaderSource(vertexShader, vertexShaderSource);
    gl.shaderSource(fragmentShader, fragmentShaderSource);
    //编译顶点、片元着色器
    gl.compileShader(vertexShader);
    gl.compileShader(fragmentShader);
    //创建程序对象program
    const program = gl.createProgram();
    //附着顶点着色器和片元着色器到program
    gl.attachShader(program, vertexShader);
    gl.attachShader(program, fragmentShader);
    //链接program
    gl.linkProgram(program);
    //使用program
    gl.useProgram(program);
    //返回程序program对象
    return program;
```

```
}
```

练习测试

你可以通过改变WebGL着色器代码内置变量`gl_PointSize`、`gl_Position`、`gl_FragColor`测试WebGL渲染效果的变化。

`gl_PointSize=20.0`改为`gl_PointSize=10.0`，观察canvas画布上点的大小变化

`gl_Position =vec4(0.0,0.0,0.0,1.0)`改为`gl_Position =vec4(0.5,0.0,0.0,1.0)`，观察canvas画布上点的位置变化

`gl_FragColor=vec4(1.0,0.0,0.0,1.0)`更改为`gl_FragColor = vec4(0.0,0.0,1.0,1.0)`，观察屏幕canvas画布上点的颜色变化

← 2. 着色器GLSL ES语言　　1. 数学基础(平移、旋转、缩放矩阵)→