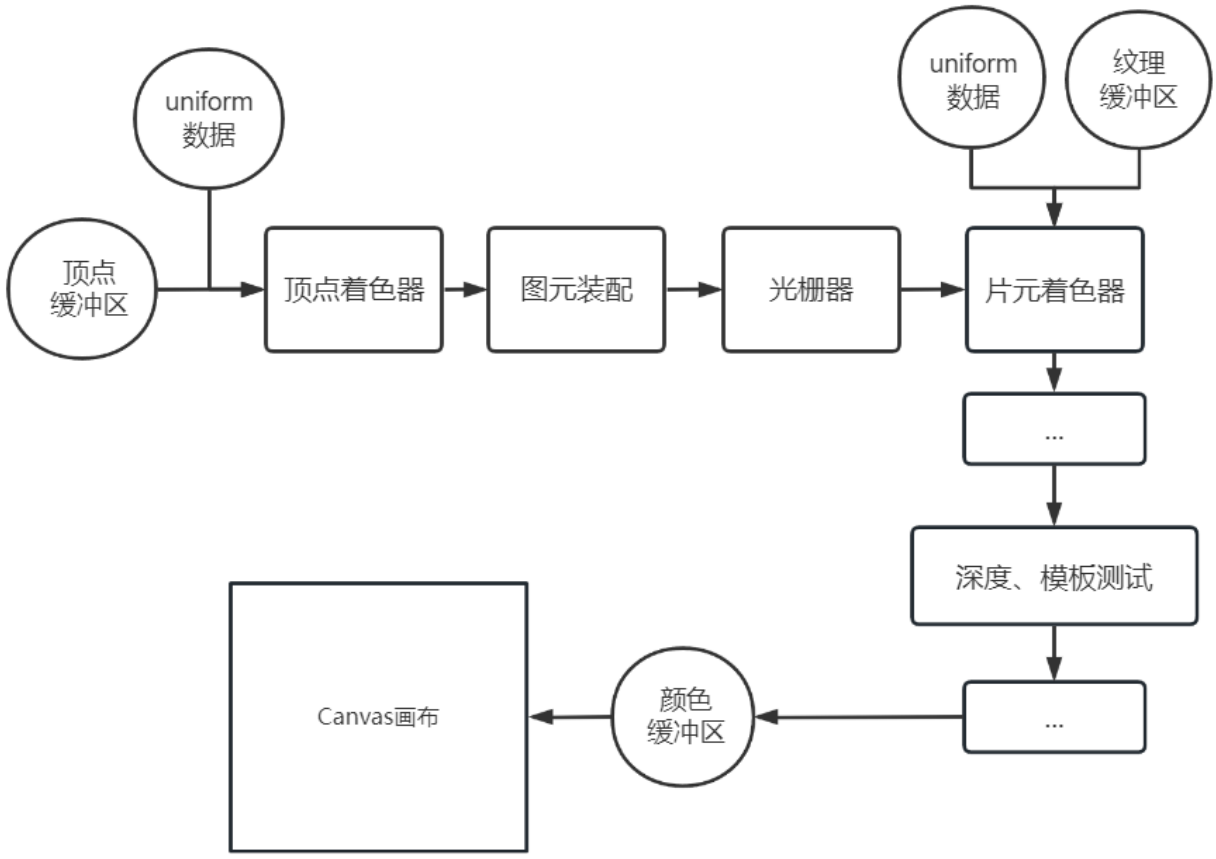
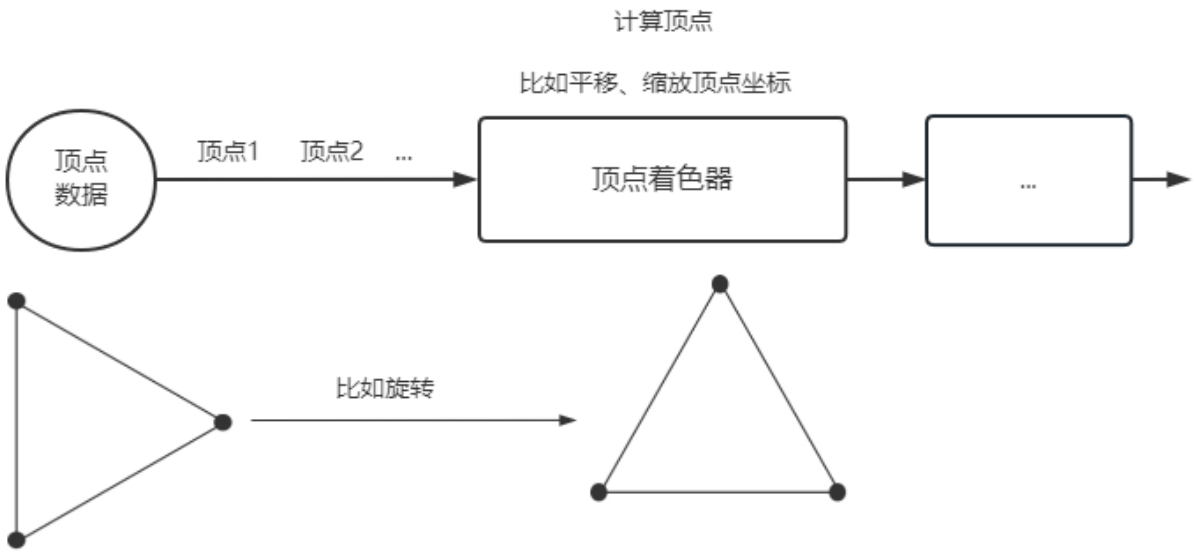
郭隆邦 2023-04-22

5. 顶点着色器

你把**渲染管线**想象为工厂的一条**流水线**，**顶点着色器**想象为流水线上一个的**工位。**



GPU渲染管线上提供的**顶点着色器单元**的功能就是**计算顶点**，所谓计算顶点，简单点说，就是对顶点坐标x、y、z的值进行平移、旋转、缩放等等各种操作。



顶点着色器代码

GPU渲染管线上的顶点着色器功能单元，可以执行WGSL着色器语言编写的代码。

所有顶点数据经过顶点着色器这个工位时候，都会执行顶点着色器代码中顶点计算的函数，比如平移顶点坐标，比如放大顶点坐标，具体怎么改变顶点数据，看你怎么写的顶点着色器代码。

```js
@vertex
fn main(@location(0) pos: vec3<f32>) -> @builtin(position) vec4<f32> {
    var pos2 = vec4<f32>(pos,1.0);//pos转齐次坐标
    pos2.x -= 0.2;//偏移所有顶点的x坐标
    return pos2;
}
```

WGSL着色器代码形式

在JavaScript或Typescript写的WebGPU代码时候，按照语法要求，WGSL着色器的代码，要以字符串的形式存在。

如果你直接在**单引号**或**双引号**表示的字符串里面写WGSL代码，实现字符串的多行书写，需要用+号码连接，不是很方便的。

```js
const str = '@vetex'
    + 'fn main(@location(0) pos: vec3<f32>) -> @builtin(position) vec4<f32> {'
    + '    return vec4<f32>(pos,1.0);'
```

```
    + '}'
```

使用ES6的语法模板字符串 `` (反引号)，实现字符串的多行书写很方便。

```js
// 顶点着色器代码
const vertex = `
@vertex
fn main(@location(0) pos: vec3<f32>) -> @builtin(position) vec4<f32> {
    return vec4<f32>(pos,1.0);
}
`
```

反引号里面写顶点着色器代码

Tab键上面的一个按键输入反引号,实现JavaScript模板字符串 `` 语法

```js
const vertex = `以字符串形式写WGSL代码`
```

@vertex

`@vertex` 表示字符串vertex里面的代码是顶点着色器代码，在GPU渲染管线的顶点着色器单元上执行。

```js
const vertex = `
@vertex
`
```

为了方便单独管理WGSL着色器代码，你可以创建一个 `shader.js` 文件，在里面写着色器代码。

```js
const vertex = `
@vertex
`
export { vertex }
```

`fn` 关键字声明一个函数

`fn` 关键字声明一个函数，命名为main，作为顶点着色器代码的入口函数。 `fn` 关键字类似JavaScript语言的 `function` 关键字，用来声明一个函数

```js
@vertex
fn main(){
}
```

vscode插件 可视化WGSL语法

搜索关键词WGSL，安装插件WGSL和WGSL Literal。

着色器代码之前设置 `/* wgsl */` ，可以使用不同颜色来显示WGSL不能的部分，更方便预览学习。

```js
// 顶点着色器代码
const vertex = /* wgsl */`
@vertex
fn main(@location(0) pos: vec3<f32>) -> @builtin(position) vec4<f32> {
    return vec4<f32>(pos,1.0);
}
`
```

`location` 关键字

`location` 是WGSL语言的一个关键字，通用用来指定顶点缓冲区相关的顶点数据，使用 `location` 的时候需要加上 `@` 符号前缀， `@location()` 小括号里面设置参数。

main函数的参数 `@location(0)` 表示你GPU显存中标记为0的顶点缓冲区中顶点数据。

```
@vertex
fn main(@location(0)){
}
```

执行 `@location(0) pos` 给main函数参数 `@location(0)` 表示的顶点数据设置一个变量名pos。

```
@vertex
fn main(@location(0) pos){
}
```

顶点变量的数据类型

可以用三维向量 `vec3` 的三个分量表示顶点的x、y、z坐标。

执行 `@location(0) pos: vec3<f32>` 给main函数参数 `pos` 设置数据类型， `vec3` 表示pos变量的数据类型是三维向量 `vec3` ， `<f32>` 表示三维向量x、y、z四个属性的值都是32位浮点数。

```
@vertex
fn main(@location(0) pos: vec3<f32>){
}
```

注意@location(0)对应WebGPU传过来的顶点是三个为一组，所以顶点着色器代码中pos变量的数据类型，用三维向量表示，如果WebGPU中传过来的顶点数据，两个为一组，比如只有x和y坐标，没有z坐标，书写形式就是 `@location(0) pos: vec2<f32>` 。

vec3顶点坐标转vec4齐次坐标

在WGSL顶点着色器代码中，很多时候会用四维向量 `vec4` 表示顶点的位置坐标， `vec4` 第四个分量默认值一般是1.0， `vec4` 相比 `vec3` 多了一个分量，你可以把 `vec4` 形式的坐标称为齐次坐标，是WGSL内部一个常用语法形式格式。

```
@vertex
fn main(@location(0) pos: vec3<f32>){
```

```
    var pos2 = vec4<f32>(pos,1.0);//pos转齐次坐标
}
```

顶点计算后，`return`返回顶点数据

实际开发，一般会在main函数中，进行顶点计算，具体说就是，对顶点的坐标进行几何变换，比如平移、缩放、旋转等操作。

```js
@vertex
fn main(@location(0) pos: vec3<f32>){
    var pos2 = vec4<f32>(pos,1.0);
    pos2.x -= 0.2;//偏移所有顶点的x坐标
}
```

渲染管线是一条流水线，顶点着色器处理好的顶点数据，最后需要通过关键字`return`返回，这样渲染管线的下个环节，就可以使用了。

```js
@vertex
fn main(@location(0) pos: vec3<f32>){
    var pos2 = vec4<f32>(pos,1.0);
    pos2.x -= 0.2;
    return pos2;//返回顶点数据，渲染管线下个环节使用
}
```

如果你不需要在GPU顶点着色器中对顶点坐标进行变换，可以直接return返回即可

```js
@vertex
fn main(@location(0) pos: vec3<f32>){
    return vec4<f32>(pos,1.0);//返回顶点数据，渲染管线下个环节使用
}
```

函数返回值数据类型

main函数`return`返回的变量，需要通过`->`符号设置函数返回值的数类类型，`-> vec4<f32>`表示函数返回的变量是浮点数构成的四维向量`vec4`。

```js
@vertex
fn main(@location(0) pos: vec43<f32>) -> vec4<f32>{
    return vec4<f32>(pos,1.0);//返回顶点数据，渲染管线下个环节使用
```

```
}
```

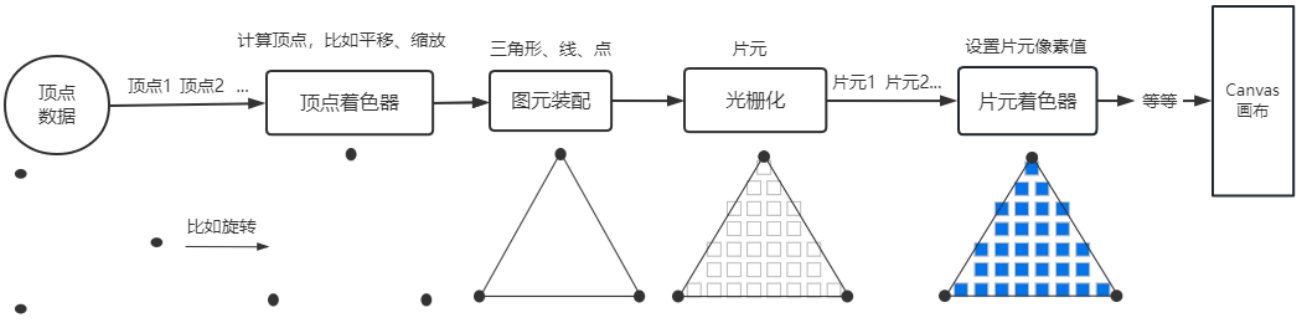
内置变量 `position` 和 `@builtin` 关键字

`position` 是WGSL语言的一个内置变量，所谓内置变量，就是说WGSL默认提供的变量，你不通过关键字 `var` 声明就可以使用。WGSL有很多内置变量，不同的内置变量有不同的含义，内置变量 `position` 表示顶点数据。

`builtin` 是WGSL语言的一个关键字，使用 `location` 的时候需要加上 `@` 符号前缀，`@location()` 小括号里面设置参数，参数一般是WGSL的某个内置变量，换句话就是当你使用内置变量的时候，一般需要通过 `@location()` 标记。

main函数的返回是顶点数据，这时候除了设置返回值数据类型，还需要设置 `@builtin(position)` ,表明返回值是顶点位置数据。

```js
@vertex
fn main(@location(0) pos: vec3<f32>) -> @builtin(position) vec4<f32>{
    return vec4<f32>(pos,1.0);//返回顶点数据，渲染管线下个环节使用
}
```



本节课完成的一个最简单顶点着色器代码

后面课程讲解，会经常在此代码基础上增删代码，第一次学习，没有记住顶点着色器全部代码也没关系，初学者会在本代码基础增删代码即可。

```js
const vertex = `
@vertex
fn main(@location(0) pos: vec3<f32>) -> @builtin(position) vec4<f32>{
    return vec4<f32>(pos,1.0);
}
```

```
`
```

7小节代码体验测试

通过上面学习，你对顶点着色器代码的功能也有了一定了解，你可以在7小节完整代码基础上，改变顶点位置坐标，体验测试，这样印象更加深刻。

着色器代码块方法 `.createShaderModule()`

`shader.js` 文件中顶点着色器代码。

```js
// 顶点着色器代码
const vertex = `
@vertex
fn main(@location(0) pos: vec3<f32>) -> @builtin(position) vec4<f32> {
    return vec4<f32>(pos,1.0);
}
`
export { vertex }
```

通过GPU设备对象的 `.createShaderModule()` 方法，把顶点着色器代码转化为GPU着色器代码块对象。

```js
// 引入顶点着色器vertex代码对应字符串
import { vertex } from './shader.js'
// 字符串形式的顶点着色器代码作为code属性的值
device.createShaderModule({ code: vertex })
```

渲染管线参数 `vertex.module` 属性

把顶点着色器代码块对象 `device.createShaderModule({ code: vertex })` 作为渲染管线参数 `vertex.module` 属性的值，这样就可以配置好渲染管线上顶点着色器功能单元，要执行的顶点着色器代码。

```js
import { vertex } from './shader.js'
const pipeline = device.createRenderPipeline({
    vertex: {
        // 设置渲染管线要执行的顶点着色器代码
        module: device.createShaderModule({ code: vertex }),
```

```
        entryPoint: "main"
    },
});
```

`entryPoint` 属性

实际开发中，一般需要通过 `entryPoint` 属性指定顶点着色器代码的入口函数，入口函数名字你可以自定义,课程中习惯性设置为 `main` 。

```js
const pipeline = device.createRenderPipeline({
    vertex: {
        module: device.createShaderModule({ code: vertex }),
        entryPoint: "main"//指定入口函数
    },
});
```

```js
const vertex =  /* wgsl */`
@vertex
fn main(@location(0) pos: vec3<f32>) -> @builtin(position) vec4<f32> {
    return vec4<f32>(pos,1.0);
}
`
```

← 4. 着色器语言WGSL快速了解　　6. 片元着色器、图元装配→