WebGL Part1

基本概念

Rong Shen

Why 3D?

- SoC 无处不在 (CPU + GPU + ... Radio Stack)
- 只要是UI，就有3D技术运用在内
- CSS 3D, iPhone UIView, Qt, WebGL, IDE(Sublime Editor), 滤镜，并行计算，神经网络...
- 即使不做3D编程，你也很可能要用到3D建模

http://heinekencitysymphony.com/

PRAHA

14:35:38

SNÍH
-1°

ON ON ON ON ON

14:35:42 Metro B dorazilo do stanice Florenc - B
14:35:41 Bus AE opustil stanici Dejvická
14:35:40 Metro A opustilo stanici Jižní z Poděbrad
14:35:39 Metro A opustilo stanici Malostranská
14:35:39 Metro C opustilo stanici Vyšehrad
14:35:39 Metro B opustilo stanici Karlovo náměstí
14:35:38 Metro B dorazilo do stanice Sokolovská rozcestí

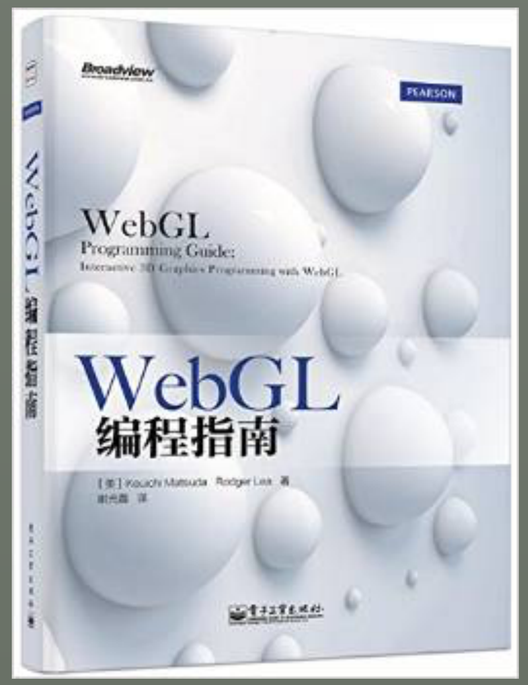
Dosud se ti zobrazilo 5 5 19 391 0

ZAPOJ SE

Share Tweet

Při sdílení této stránky, nebo tweetnutí hashtagu #heinekencitysymphony se zázrakem i ty staneš součástí symfonie.

WebGL Programming Guide

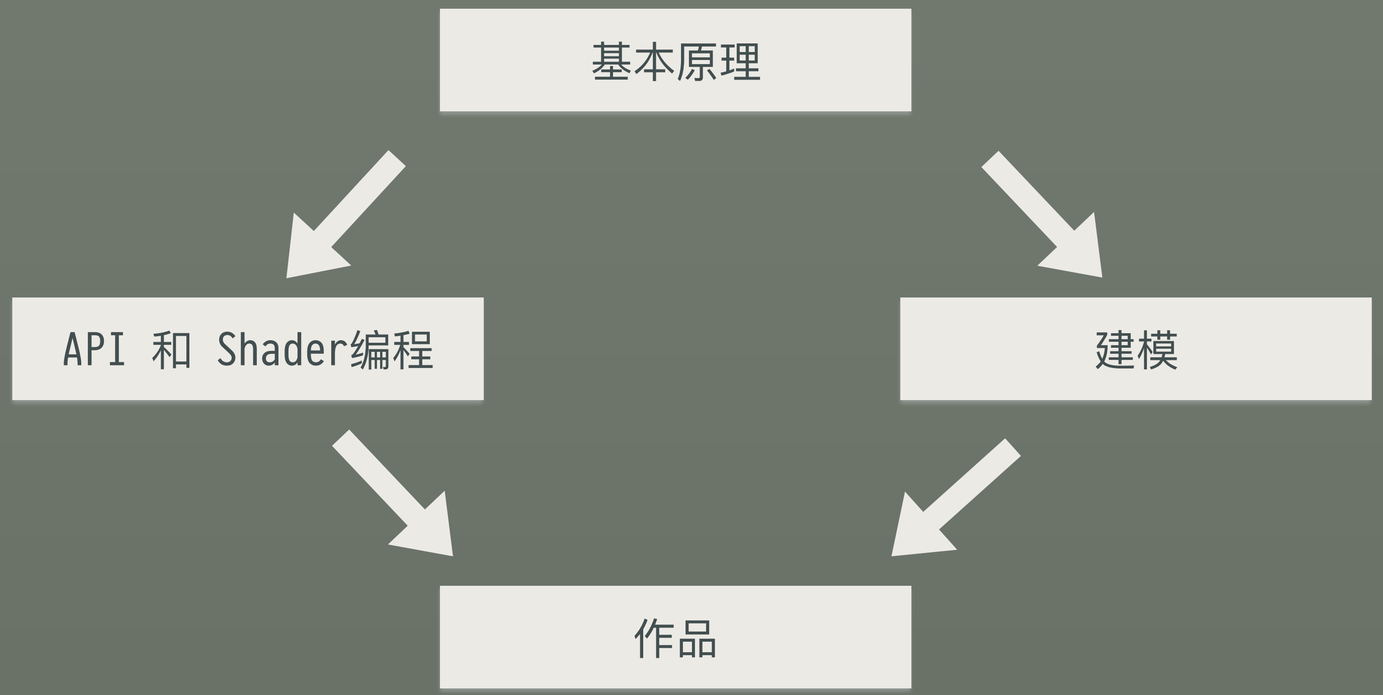


http://www.amazon.cn/WebGL编程指南-松田浩一/dp/B00KXQPR0Y/ref=sr_1_1?ie=UTF8&qid=1423214139&sr=8-1&keywords=WebGL+Programming+Guide

Sample Code

https://sites.google.com/site/webglbook/home/downloads

3D编程和设计



- 我们重点讲解基本原理
- 包括一些初级的矩阵变换😆

课程范围

- 课程样例
- GPU 概述
- WebGL 入门
- 绘制和变换三角形
- 高级变换和动画
- 颜色和纹理
- GLSL ES
- 三维世界

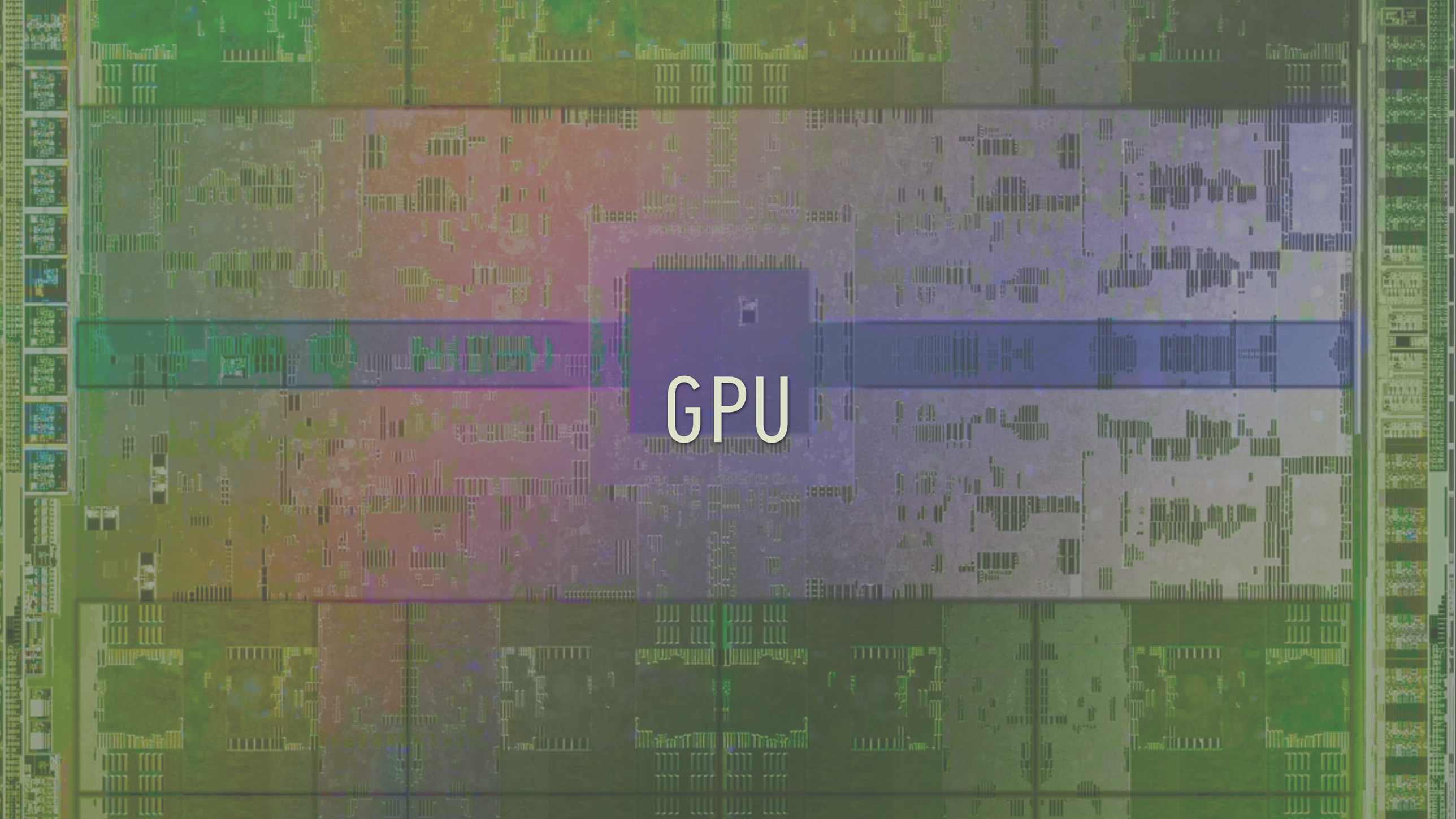
- 光照和阴影
- 其他高级技术

本课程样例的组织形式

- https://github.com/jacobbubu/browserify-beefy-tutorial
- 没有 HTML 页面
- 所有代码都是 CoffeeScript
- CommonJS 规范的 Modules

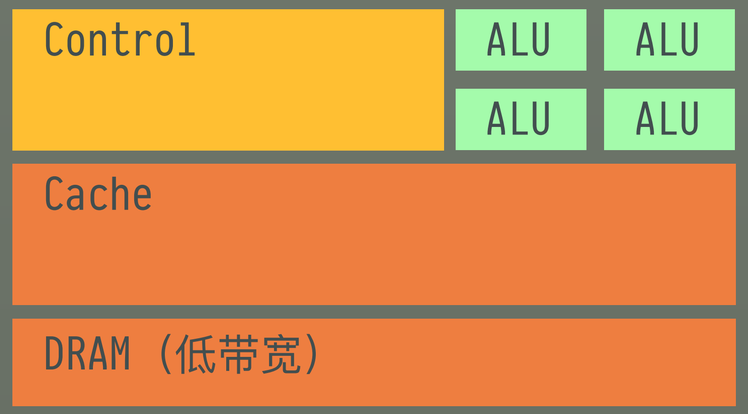
样例

- npm start -- examples/testbed.coffee
- http://localhost:9966

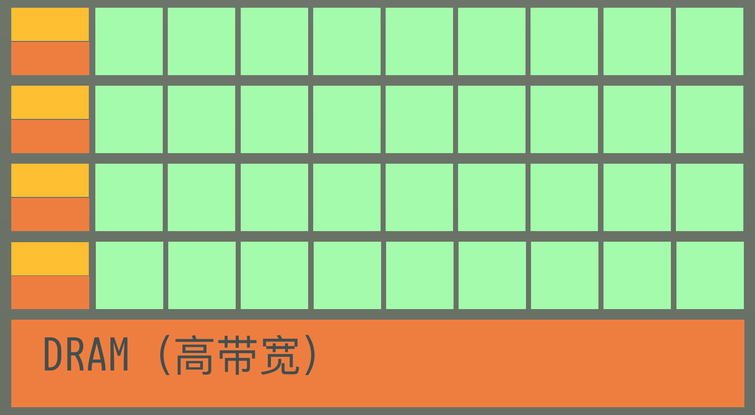

GPU

GPU快在哪里?

- 几十个控制器，数以千记的计算单元
- 每个控制器可以控制几千个计算线程；极小的线程切换成本(无需寄存器切换)
- SIMT(Single-Instruction, Multiple-Thread)
- 对齐的数据结构
- Pipeline处理（Stream）
- 几乎不回写数据



CPU

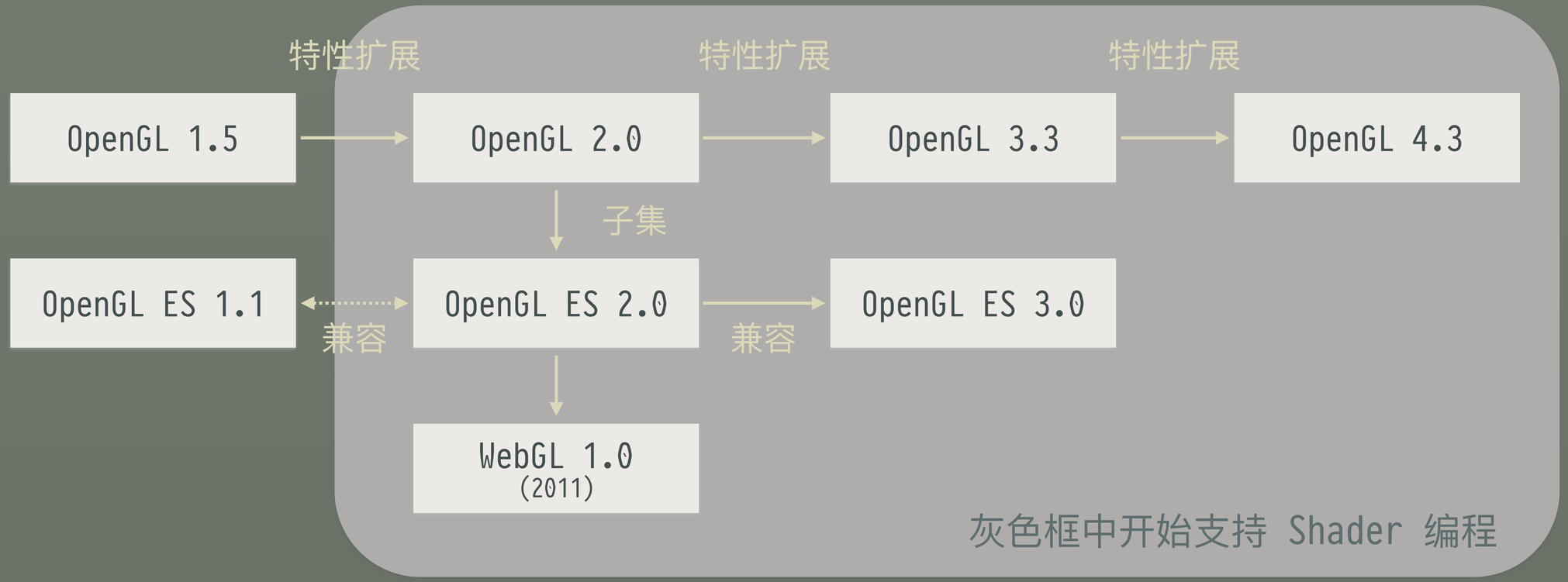


GPU

ALU: Arithmetic logic unit

WebGL 入门

发展路程


特性扩展
特性扩展
特性扩展
OpenGL 1.5
OpenGL 2.0
OpenGL 3.3
OpenGL 4.3
子集
OpenGL ES 1.1
OpenGL ES 2.0
OpenGL ES 3.0
兼容
兼容
WebGL 1.0
(2011)
灰色框中开始支持 Shader 编程

绘制一个点

- helloPoint1.coffee

Vertex Shader(顶点着色器)

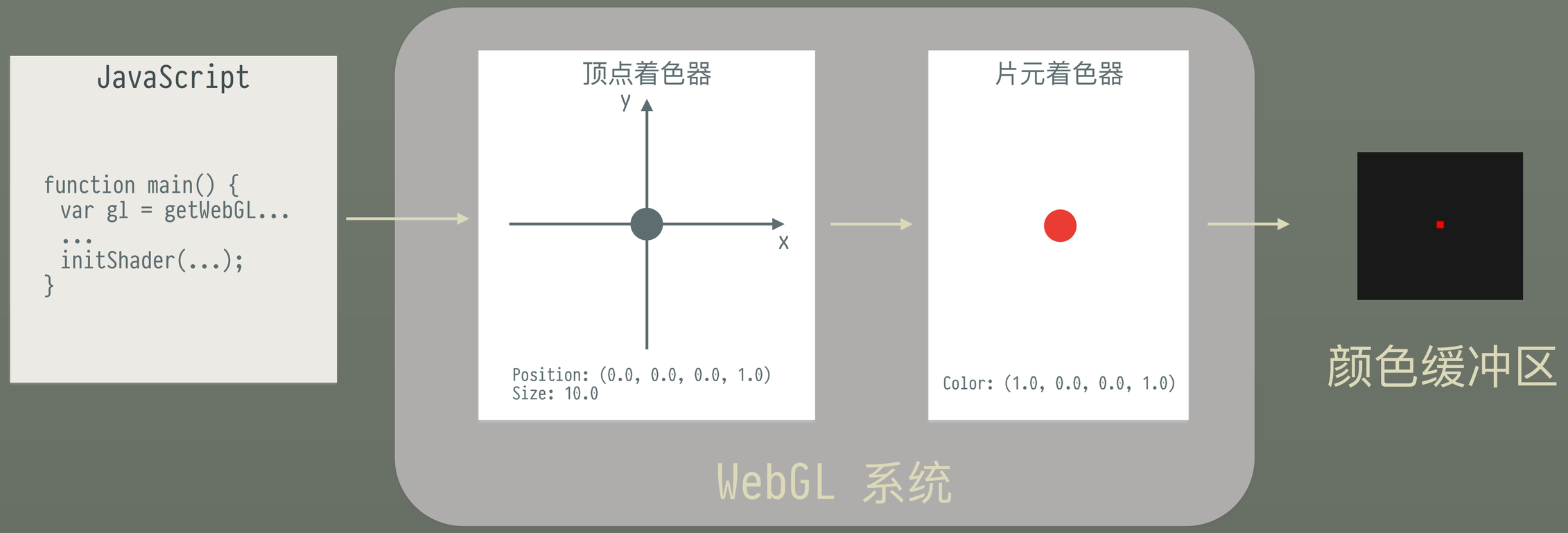
- 用来描述几何形状顶点特性的程序（如位置、颜色等）
- 顶点是指二维或者三维空间中的一个点，例如几何形状的端点或者交点

Fragment Shader(片元着色器)

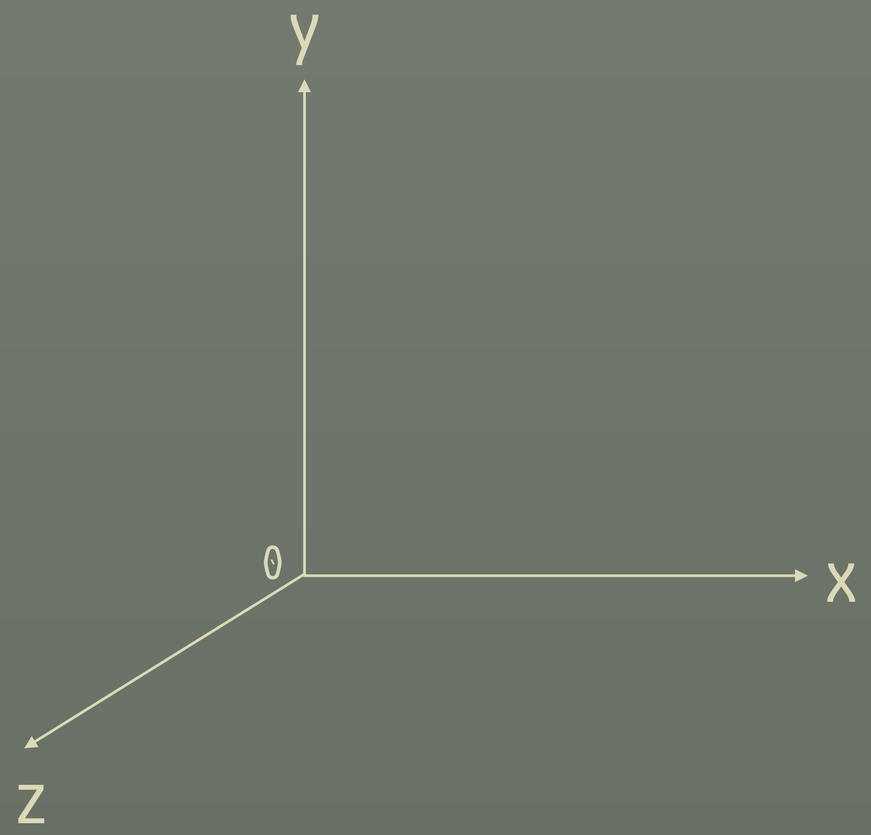
- Fragment = Pixel(像素)
- 计算光栅化之后的每个像素应该具有的颜色

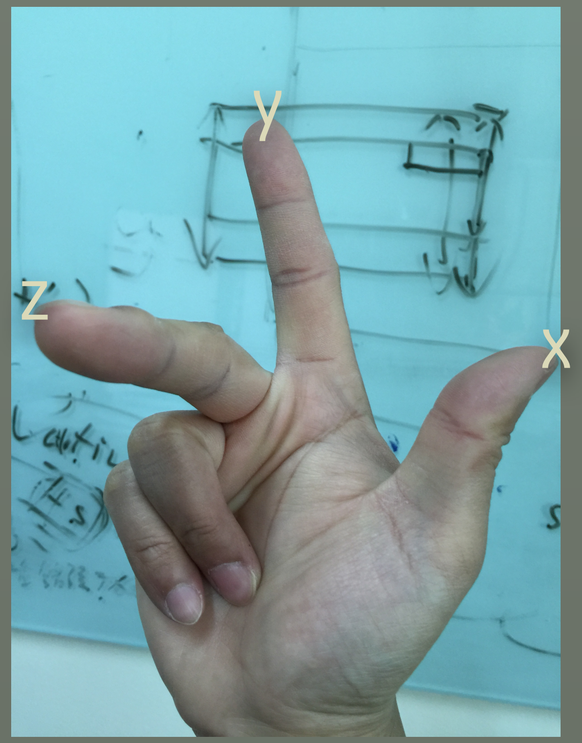
GLSL ES 中的数据类型

- float：浮点数，例如：
  - gl_PointSize = 10.0;
- vec4：四个浮点数组成的矢量，例如：
  - gl_Position = vec4(0.0, 0.0, 0.0, 1.0);
  - gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);


JavaScript
function main() {
var gl = getWebGL...
...
initShader(...);
}
顶点着色器
y
x
Position: (0.0, 0.0, 0.0, 1.0)
Size: 10.0
片元着色器
Color: (1.0, 0.0, 0.0, 1.0)
颜色缓冲区
WebGL 系统

右手坐标系





不是所有的3D API都是右手坐标系

关键点

- WebGL 程序包括，运行在：
  - 浏览器中的 JavaScript
  - WebGL 系统中的着色器程序

绘制一个点(传入坐标)

- helloPoint2.coffee

存储限定符	类型	变量名
attribute	vec4	a_Position;

```
# 获得 Shader 中的变量在 JavaScript 中的存储位置
a_Position = gl.getAttribLocation gl.program, 'a_Position'
```

```
# 在 JS 中设置变量的值
gl.vertexAttrib3f a_Position, 0.0, 0.5, 0.0

gl.vertexAttrib1f location, v0        # v1,v2 缺省为0, v4缺省为1
gl.vertexAttrib2f location, v0, v1   同上
gl.vertexAttrib4f location, v0, v1, v2, v3, v4
```

鼠标点击绘制

- clickedPoint.coffee

```
render = (gl, width, height, dt) ->
    clear gl
    for i in [0...g_points.length] by 2
        gl.vertexAttrib3f a_Position, g_points[i], g_points[i+1], 0.0
        gl.drawArrays gl.POINTS, 0, 1
```

- 每一帧，都对保存的每一个鼠标点击坐标进行一次绘制
- 每一次绘制都会执行顶点着色器(1次)和片元着色器(100次)

改变点的颜色

coloredPoints.coffee

uniform 变量

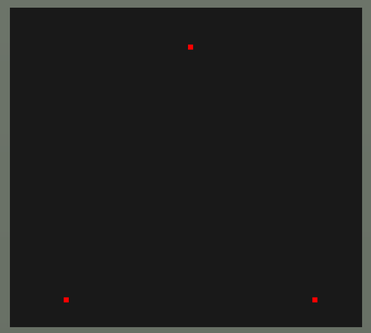
- 只有顶点着色器才能使用 attribute 变量
- 片元着色器可以使用 uniform 和 varying(后面介绍) 变量
- uniform 是指，在一次绘制过程中不变的变量
  - 一次绘制：gl.drawArrays gl.POINTS, 0, 1

绘制和变换三角形

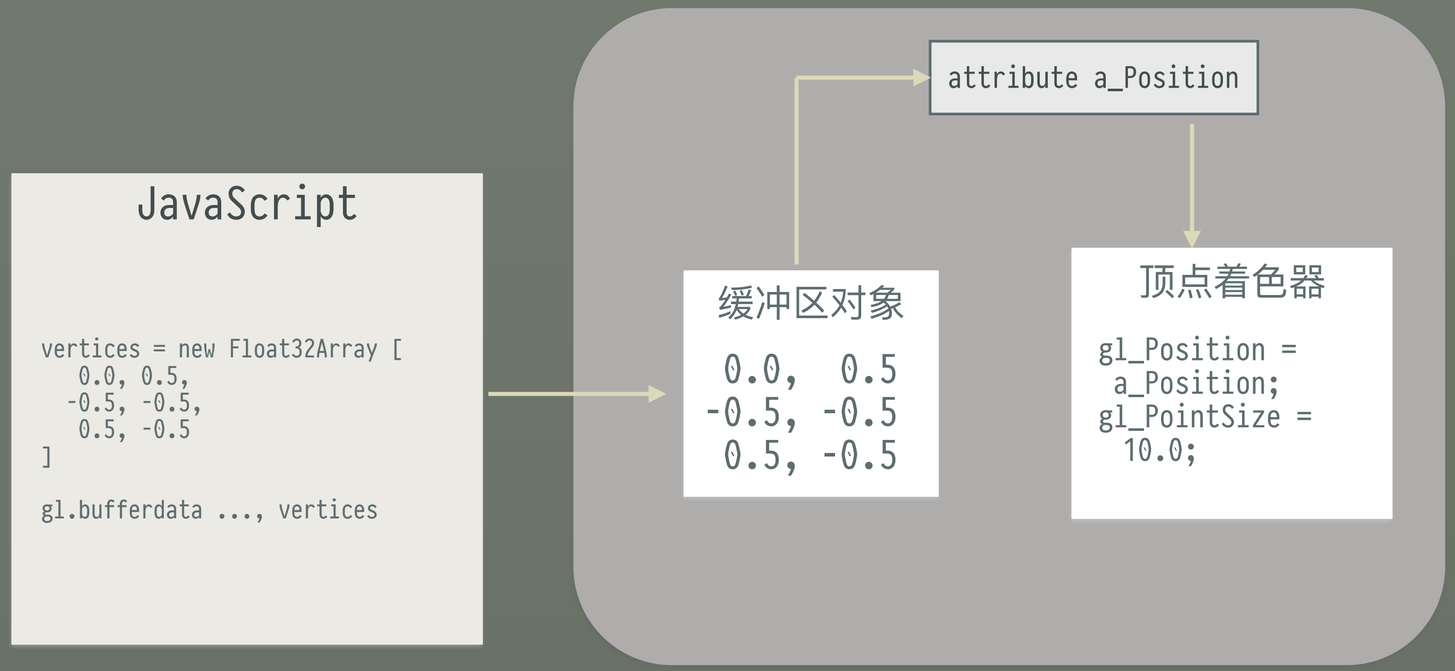
- 一次传入多个顶点以绘制三角形
- 使用多个三角形绘制其他基本图形
- 对三角形进行基本变换，如移动、缩放和旋转
- 利用矩阵变换

绘制多个点

multiPoints.coffee



使用缓冲区对象


JavaScript
vertices = new Float32Array [
0.0, 0.5,
-0.5, -0.5,
0.5, -0.5
]
gl.bufferdata ..., vertices
attribute a_Position
缓冲区对象
0.0, 0.5
-0.5, -0.5
0.5, -0.5
顶点着色器
gl_Position =
a_Position;
gl_PointSize =
10.0;

主要步骤

- 创建缓冲区对象(gl.createBuffer)
- 绑定缓冲区对象(gl.bindBuffer)
- 将数据写入缓冲区(gl.bufferData)
- 将缓冲区对象分配给一个 attribute 变量 (gl.vertexAttribPointer)
- 开启 attribute 变量 (gl.enableVertexAttribArray)

gl.bindBuffer(target, buffer)		
允许使用 *buffer* 表示的缓冲区对象并将其绑定到 *target* 指定的目标上		
参数	target	可以是以下值之一
	gl.ARRAY_BUFFER	纹理参数，见下页
	gl.ELEMENT_ARRAY_BUFFER	纹理参数的值，见下页
	buffer	指定之前由 gl.createBuffer() 返回的待绑定的缓冲区对象 如果为 null, 则禁用对 target 的绑定

gl.bufferData(target, data, usage)		
开辟存储空间，向绑定在 *target* 上的缓冲区对象中写入数据 *data*		
参数	target	gl.ARRAY_BUFFER 或 gl.ELEMENT_ARRAY_BUFFER
	usage	表示程序将如何使用存储在缓冲区对象中的数据。该参数将帮助 WebGL 优化操作，但是就算你传入了错误的值，也不会终止程序(仅仅是降低程序的效率)
	gl.STATIC_DRAW	只会向缓冲区对象中写入一次数据，但需要绘制很多次
	gl.STREAM_DRAW	只会向缓冲区对象中写入一次数据，然后绘制若干次
	gl.DYNAMIC_DRAW	会向缓冲区对象中多次写入数据，并绘制多次

```
vertices = new Float32Array [
        0.0, 0.5,   -0.5, -0.5,   0.5, -0.5
]

...

gl.bufferData gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW
```

JS 类型化数组

数组类型	所占字节数	描述
Int8Array	1	8位整型数
UInt8Array	1	8位无符号整型数
Int16Array	2	16位整型数
UInt16Array	2	16位无符号整型数
Int32Array	4	32位整型数
UInt32Array	4	32位无符号整型数
Float32Array	4	单精度32位浮点数
Float64Array	8	双精度64位浮点数

gl.vertexAttribPointer

gl.vertexAttribPointer(location, size, type, normalized, stride, offset)		
将绑定到 gl_ARRAY_BUFFER 的缓冲区对象分配给由 *location* 指定的 *arrtribute* 变量		
参数	location	指定待分配 attribute 变量的存储位置
	size	指定缓冲区中每个顶点的分量个数（1到4）
	type	指定数据类型，例如 gl.FLOAT
	normalized	true 或 false，表明是否将非浮点型的数据归一化到 [0, 1] 或 [-1, 1] 区间
	stride	指定相邻两个顶点间的字节数，默认为0
	offset	指定缓冲区对象中的偏移量(以字节为单位)，即 attribute 从缓冲区中的何处开始存储。如果是从起始位置开始存储，其值应设为0

```
gl.vertexAttribPointer a_Position, 2, gl.FLOAT, false, 0, 0
```

开启 attribute 变量

- gl.enableVertexAttribArray a_Position
- 对应的 gl.disableVertexAttribArray a_Position
- 一旦 Enable 之后，就不允许使用 gl.vertexAttrib[1234]f 来传送数据了

类型化数组的方法和属性

- get(index): 获取第 index 个元素值
- set(index, value): 设置第 index 个元素的值
- set(array, offset): 从第 offset 个元素开始将 array 内容复制过去
- length: 数组的元素个数
- BYTES_PER_ELEMENT: 数组中每个元素所占的字节数
- 注意，没有 push、pop、unshift 等 JS Array 方法

gl.drawArrays gl.POINTS, 0, vertexNum

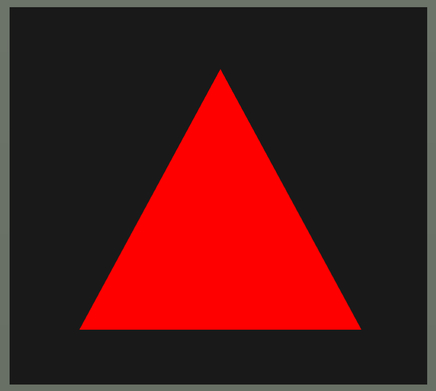
gl.drawArrays(mode, first, count)		
执行顶点着色器，按照 *mode* 指定的方式绘制图形		
参数	mode	指定会脂肪是，可以接收以下常量: gl.POINTS、gl.LINE、gl.LINE_STRIP、gl.LINE_LOOP、gl.TRIANGLES、gl.TRIANGLE_STRIP、gl.TRIANGLE_FAN
	first	指定从哪个顶点开始绘制(整型数)
	count	指定这次绘制需要用到多少个顶点(整型数)

attribute 和 uniform

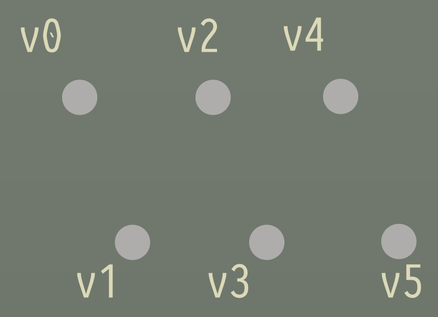
- 可以看到在一次绘制过程中，Vertex Shader 或枚举 attribute
- 而 uniform 在一次绘制中不会变化

绘制三角形

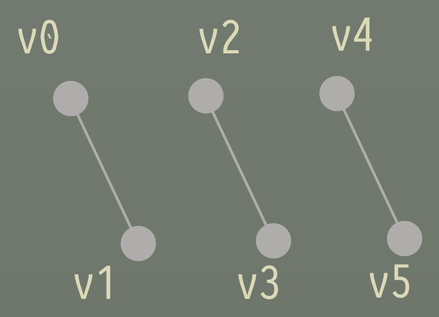
helloTriangle.coffee



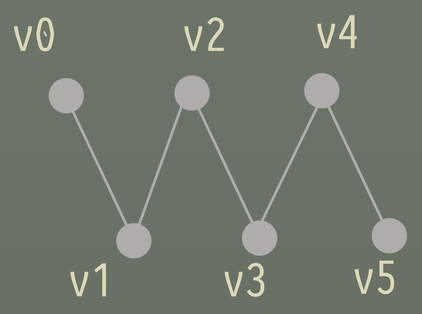
基本图形(gl.drawArrays)



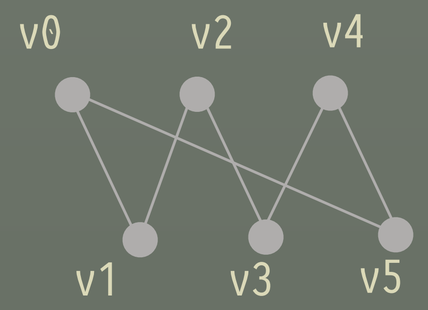
gl.POINTS



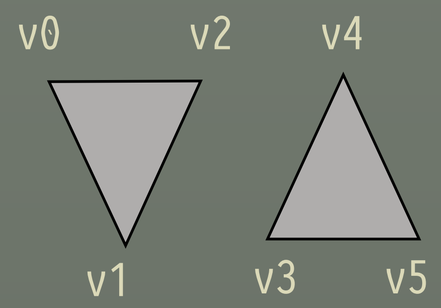
gl.LINES



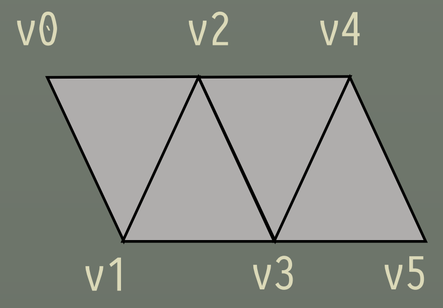
gl.LINE_STRIP



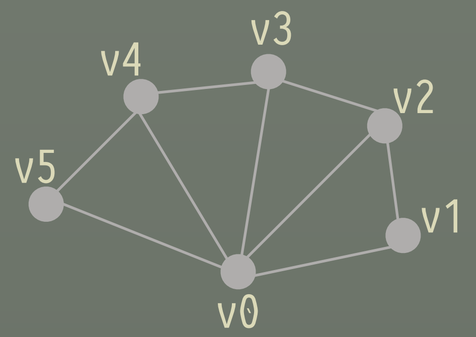
gl.LINE_LOOP



gl.TRIANGLES



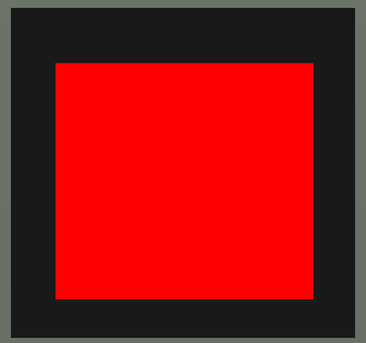
gl.TRIANGLE_STRIP



gl.TRIANGLE_FAN

绘制矩形

helloQuad.coffee



简单变换

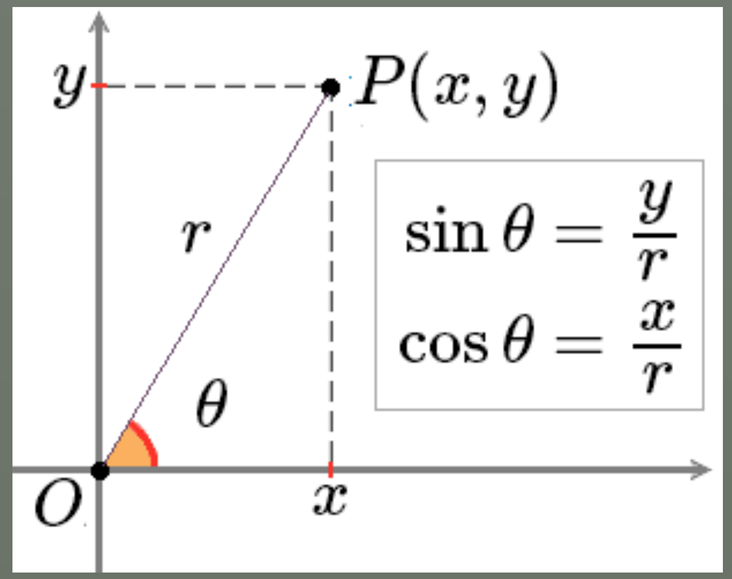
平移

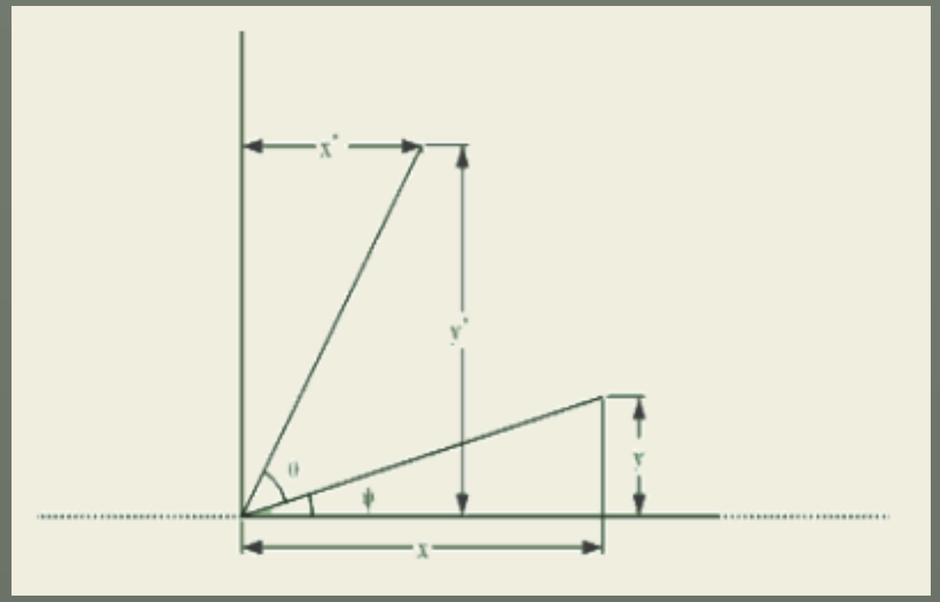
translatedTriangle.coffee

旋转

rotatedTriangle.coffee

旋转





$$x' = x\cos\theta - y\sin\theta$$

$$y' = x\sin\theta + y\cos\theta$$

变换矩阵

矩阵乘以向量

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$x' = ax + by + cz$$

$$y' = dx + ey + fz$$

$$z' = gx + hy + iz$$

旋转方程到旋转矩阵

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

$$\Rightarrow \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

扩展到三维

二维旋转矩阵

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

三维绕 z 轴旋转矩阵

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

其他轴

rotateX

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \bullet \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

rotateY

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} \bullet \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

缩放

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \times s_x \\ y \times s_y \\ z \times s_z \end{pmatrix} \Rightarrow \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix} \bullet \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$x' = s_x \times x + 0 \times y + 0 \times z$$

$$y' = 0 \times x + s_y \times y + 0 \times z$$

$$z' = 0 \times x + 0 \times y + s_z \times z$$

平移

我们需要：

$$x' = x + t_x$$
$$y' = y + t_y$$
$$z' = z + t_z$$

但是既有的3x3矩阵
无法满足

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \bullet \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$x' = ax + by + cz$$

4x4矩阵与齐次坐标

平移齐次变换矩阵

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_z \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \Rightarrow \begin{aligned} x' &= x + t_x \\ y' &= y + t_y \\ z' &= z + t_z \end{aligned}$$

rotateZ齐次变换矩阵

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- rotatedTriangle_Matrix.coffee

行主序和列主序

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix}$$

按行主序

$$\begin{pmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & p \\ d & h & l & p \end{pmatrix}$$

按列主序（WebGL）

缩放齐次变换矩阵

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- `scaledTriangle.coffee`

高级变换和动画

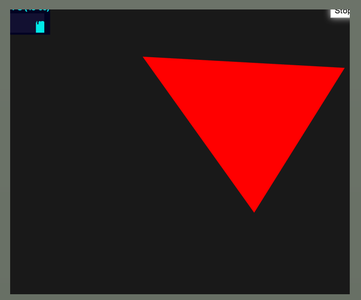
- glMatrix
- 复合变换
- 动起来

glMatrix

https://github.com/toji/gl-matrix

平移后旋转

- rotatedTranslatedTriangle.coffee



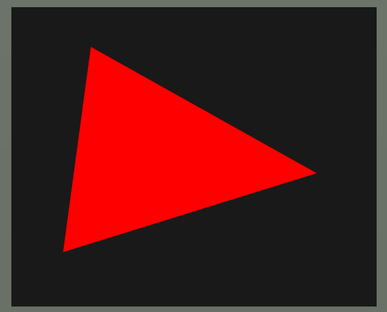
顺序很重要

- 平移后旋转和旋转后平移是不一样的
- 平移后坐标 = 平移矩阵 x 原始坐标
- 平移后旋转坐标 = 旋转矩阵 x 平移后坐标
- 代入：
  - 平移后旋转坐标 = 旋转矩阵 x（平移矩阵 x 原始坐标）
  - 平移后旋转坐标 = 旋转矩阵 x 平移矩阵 x 原始坐标
  - 这就是为什么下面程序先准备旋转矩阵，后乘以平移矩阵的原因

```
...
 modelMatrix = mat4.create()
 mat4.rotateZ modelMatrix, modelMatrix, radian
 mat4.translate modelMatrix, modelMatrix, vec3.fromValues Tx, Ty, 0
...
```

动起来

- rotatingTriangle.coffee(speedUp/Down)

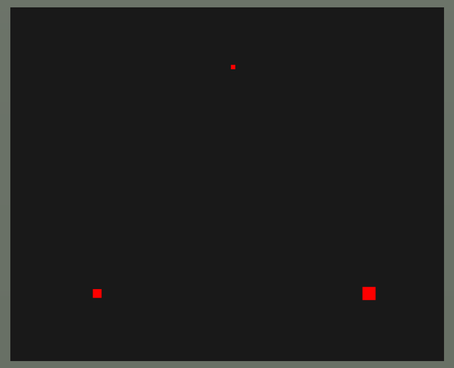


颜色与纹理

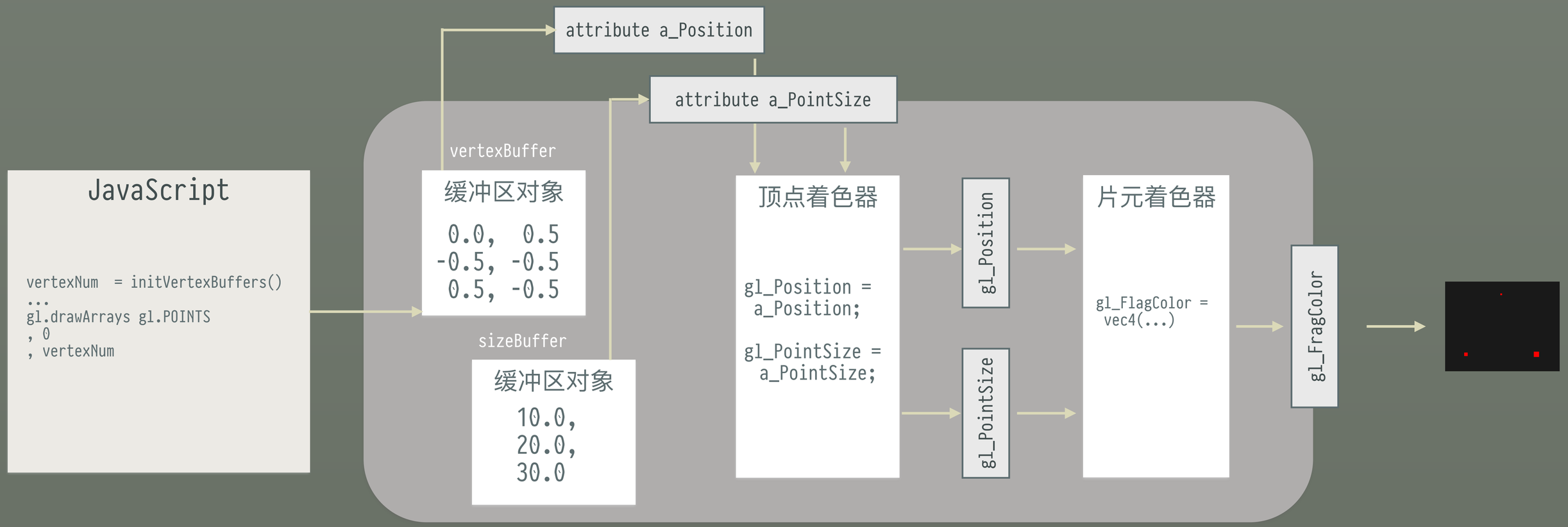
- 将非顶点数据传入 Vertex Shader
- 光栅化
- 将图像(纹理)映射到三维对象表面

非坐标数据传入Vertex Shader

- multiAttributeSize.coffee

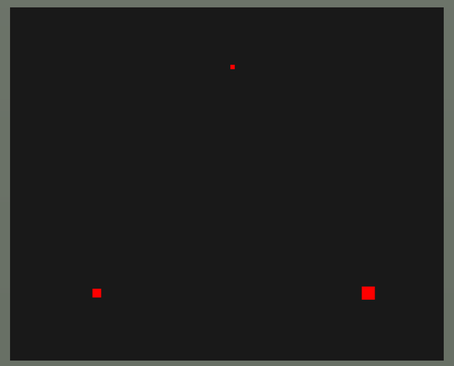


使用两个缓冲区对象


attribute a_Position
attribute a_PointSize
JavaScript
vertexNum = initVertexBuffers()
...
gl.drawArrays gl.POINTS
, 0
, vertexNum
vertexBuffer
缓冲区对象
0.0, 0.5
-0.5, -0.5
0.5, -0.5
sizeBuffer
缓冲区对象
10.0,
20.0,
30.0
顶点着色器
gl_Position =
a_Position;
gl_PointSize =
a_PointSize;
gl_Position
gl_PointSize
片元着色器
gl_FlagColor =
vec4(...)
gl_FragColor

使用一个缓冲区对象

- multiAttribute_Interleaved.coffee



gl.vertexAttribPointer

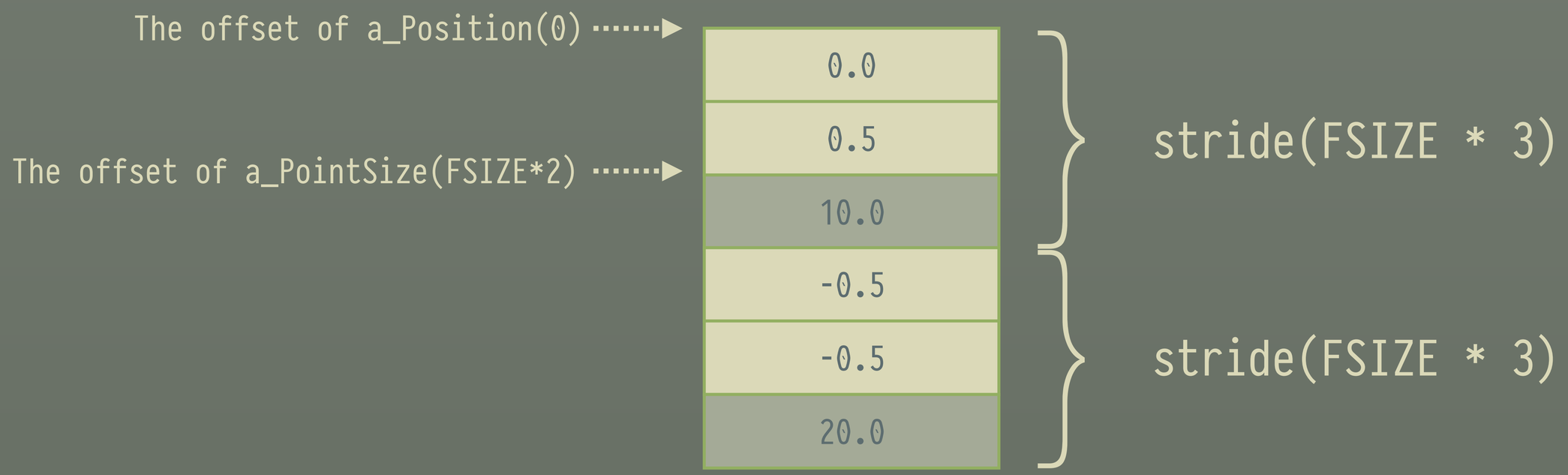
gl.vertexAttribPointer(location, size, type, normalized, stride, offset)		
将绑定到 gl_ARRAY_BUFFER 的缓冲区对象分配给由 *location* 指定的 *arrtribute* 变量		
参数	location	指定待分配 attribute 变量的存储位置
	size	指定缓冲区中每个顶点的分量个数（1到4）
	type	指定数据类型，例如 gl.FLOAT
	normalized	true 或 false，表明是否将非浮点型的数据归一化到 [0, 1] 或 [-1, 1] 区间
	stride	指定相邻两个顶点间的字节数，默认为0
	offset	指定缓冲区对象中的偏移量(以字节为单位)，即 attribute 从缓冲区中的何处开始存储。如果是从起始位置开始存储，其值应设为0

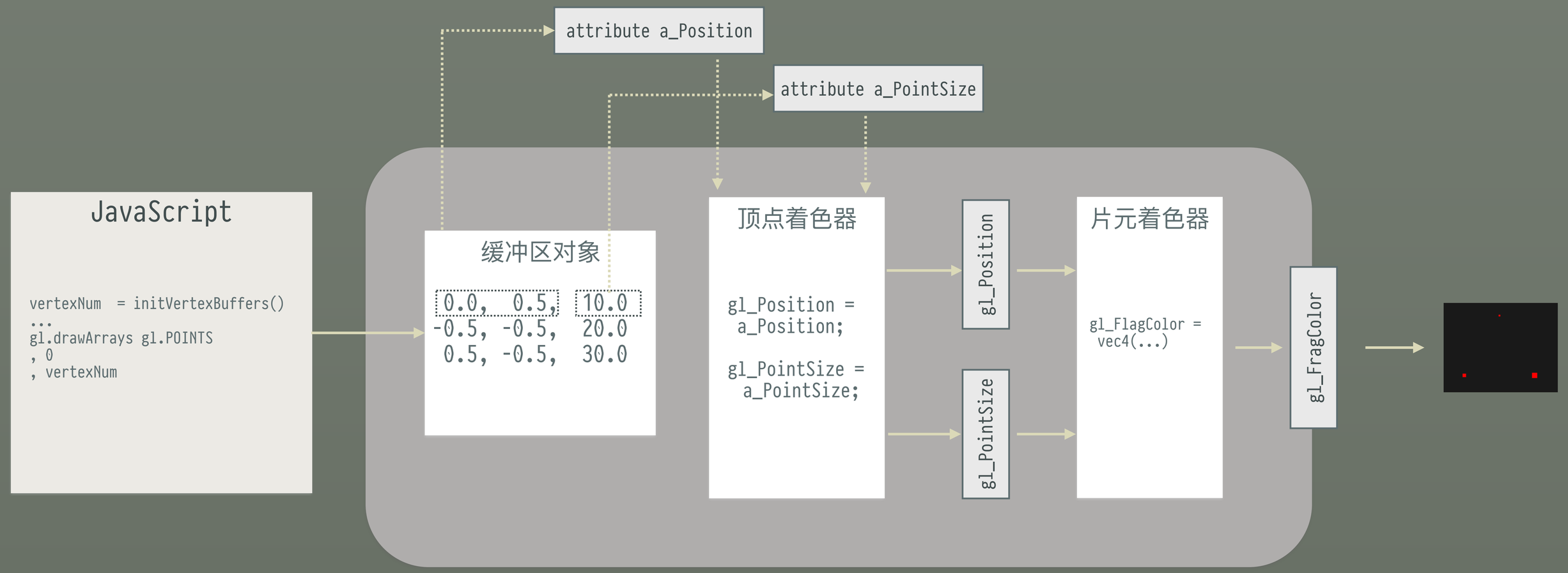
```
gl.vertexAttribPointer a_Position, 2, gl.FLOAT, false, 0, 0
```

Stride 和 Offset

FSIZE = verticesSizes.BYTES_PER_ELEMENT

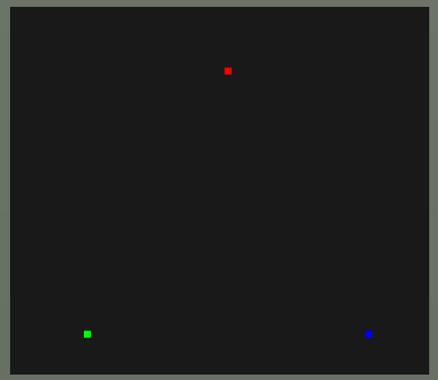
对于浮点数 FSIZE = 4




attribute a_Position
attribute a_PointSize
JavaScript
vertexNum = initVertexBuffers()
...
gl.drawArrays gl.POINTS
, 0
, vertexNum
缓冲区对象
0.0, 0.5, 10.0
-0.5, -0.5, 20.0
0.5, -0.5, 30.0
顶点着色器
gl_Position =
a_Position;
gl_PointSize =
a_PointSize;
gl_Position
gl_PointSize
片元着色器
gl_FlagColor =
vec4(...)
gl_FragColor

修改颜色

- multiAttributeColor.coffee

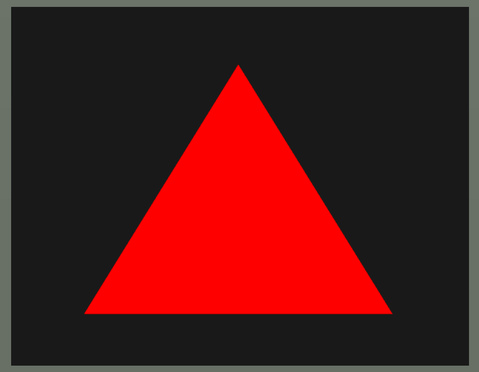


varying

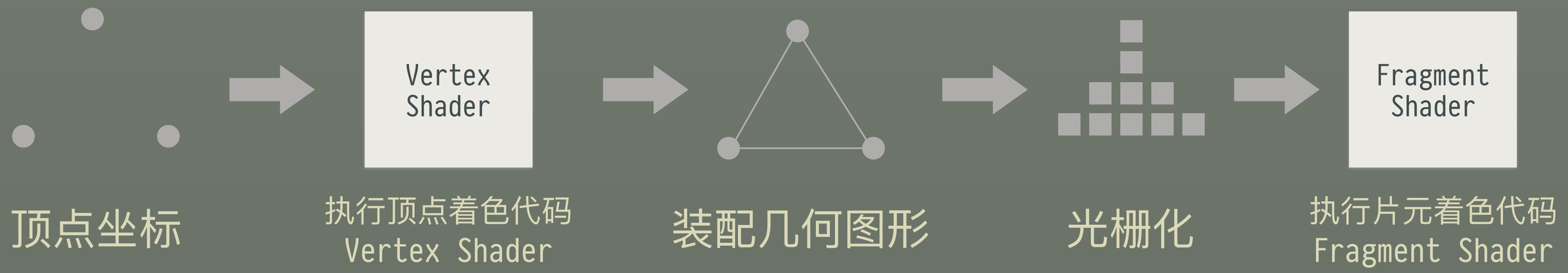
- 颜色值通过 attribute 传递给顶点着色器
- 但是却是以 varying 方式被传递给片元着色器
- 后面详细讲解其原理

集合形状的装配与光栅化

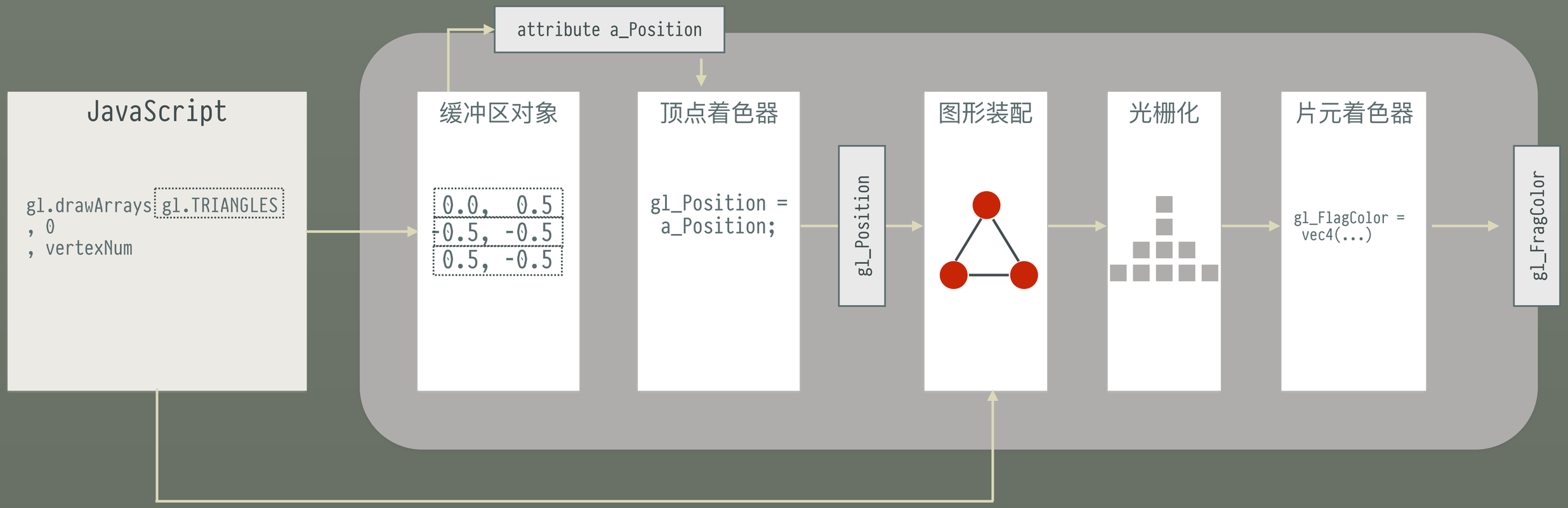
- Back to helloTriangle.coffee



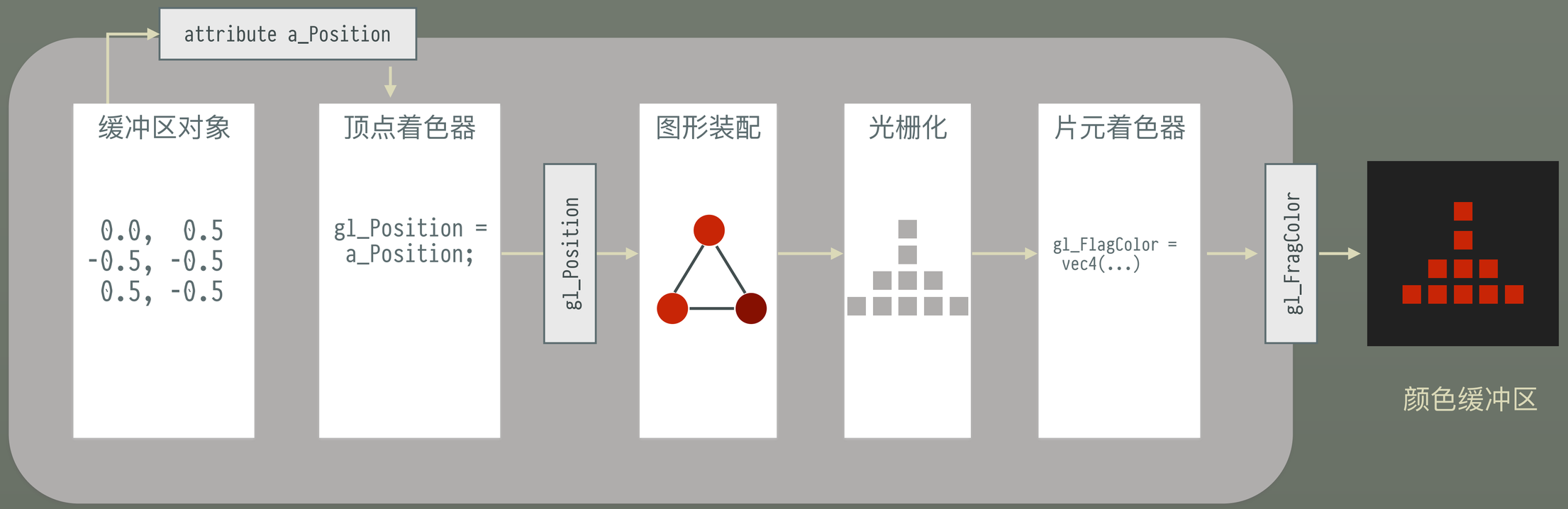
顶点坐标、图形装配、光栅化、片元着色



图形装配和光栅化

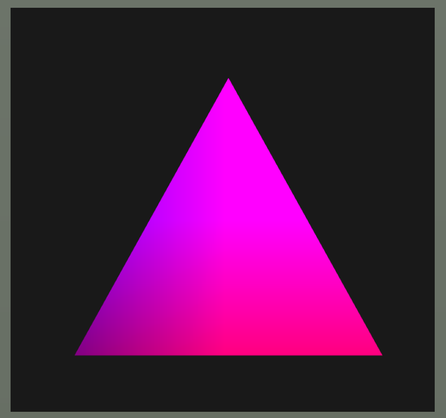

attribute a_Position
JavaScript
gl.drawArrays gl.TRIANGLES
, 0
, vertexNum
缓冲区对象
0.0, 0.5
-0.5, -0.5
0.5, -0.5
顶点着色器
gl_Position =
a_Position;
gl_Position
图形装配
光栅化
片元着色器
gl_FlagColor =
vec4(...)
gl_FragColor

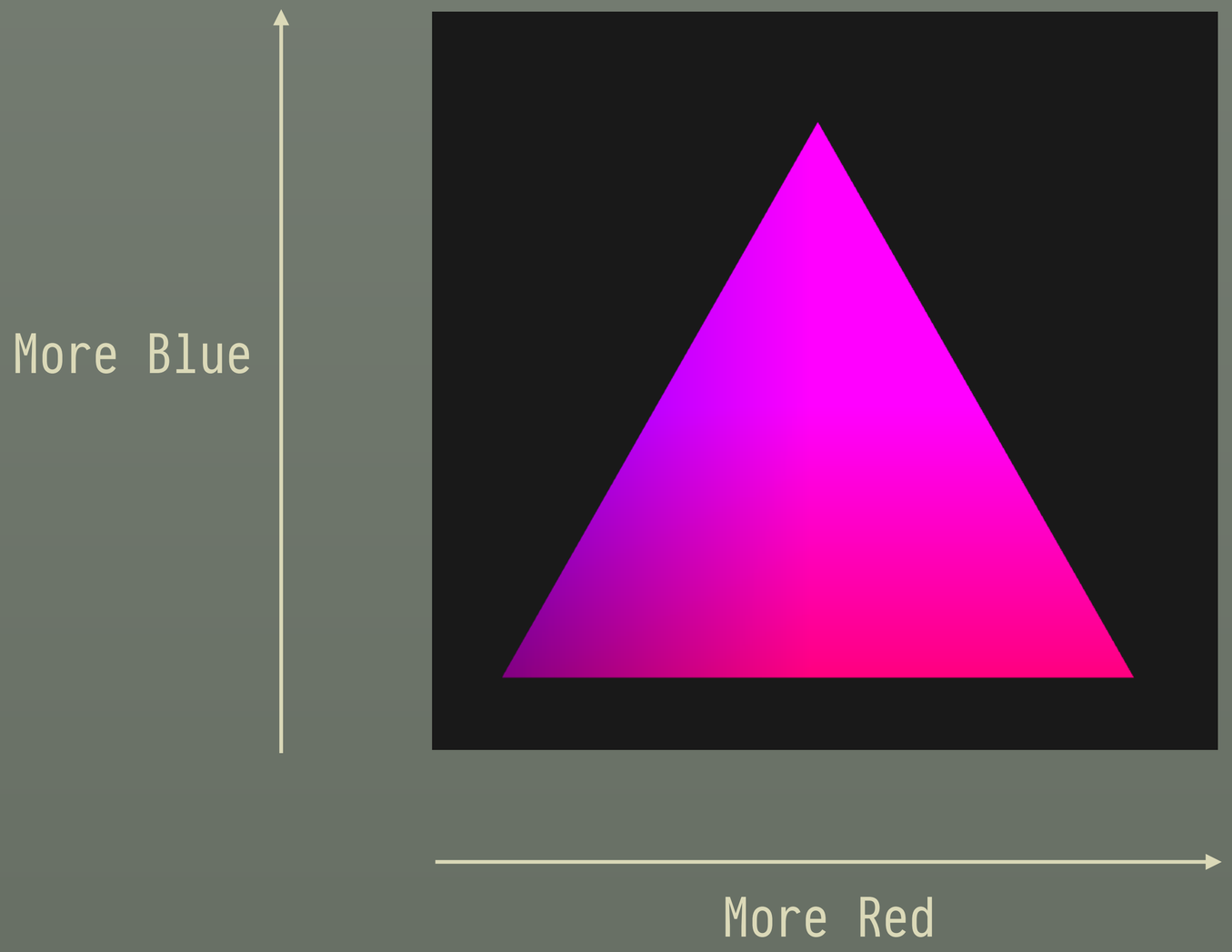
Fragment Shader



varying 变量和内插过程

- helloTriangle_FragCoord.coffee

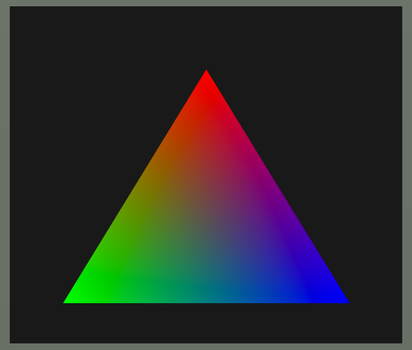




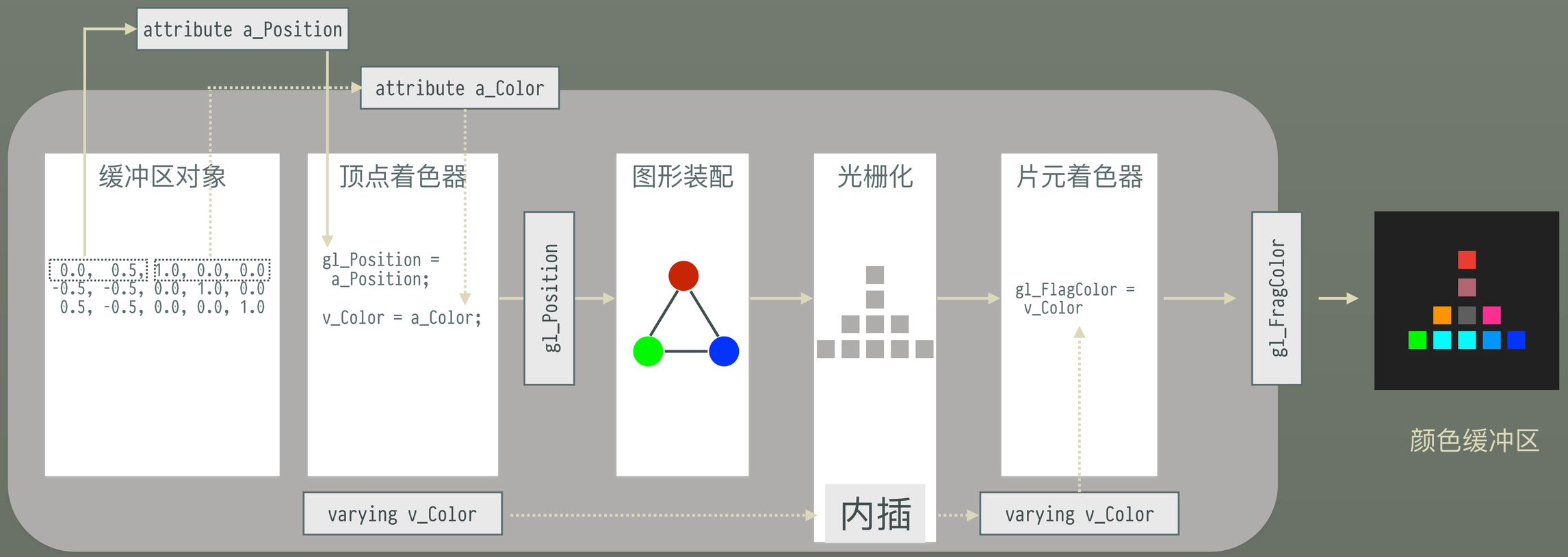

```
gl_FragColor = vec4(
   gl_FragCoord.x/u_Width,
    0.0,
    gl_FragCoord.y/u_Height, 1.0
 );
```

彩色三角形

- coloredTriangle.coffee



Varying 变量与内插



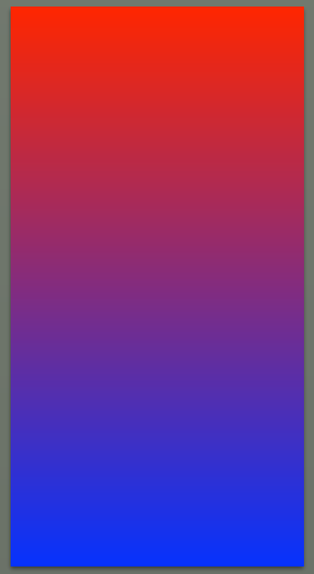
颜色值内插

红色

(1.0, 0.0, 0.0)

蓝色

(0.0, 0.0, 1.0)



红色 (1.0, 0.0, 0.0)

内插颜色 (0.75, 0.0, 0.25)

内插颜色 (0.5, 0.0, 0.5)

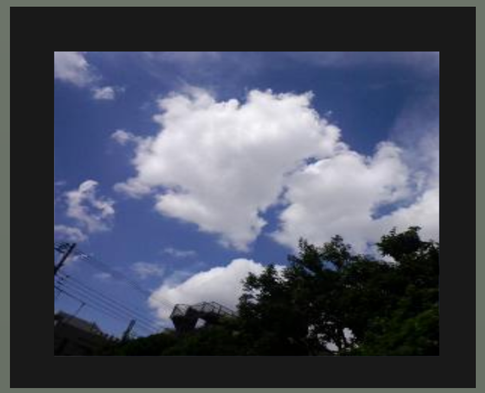
内插颜色 (0.25, 0.0, 0.75)

内插颜色 (1.0, 0.0, 0.0)

v_Color

纹理映射(贴图)

- texturedQuad.coffee



纹理映射步骤

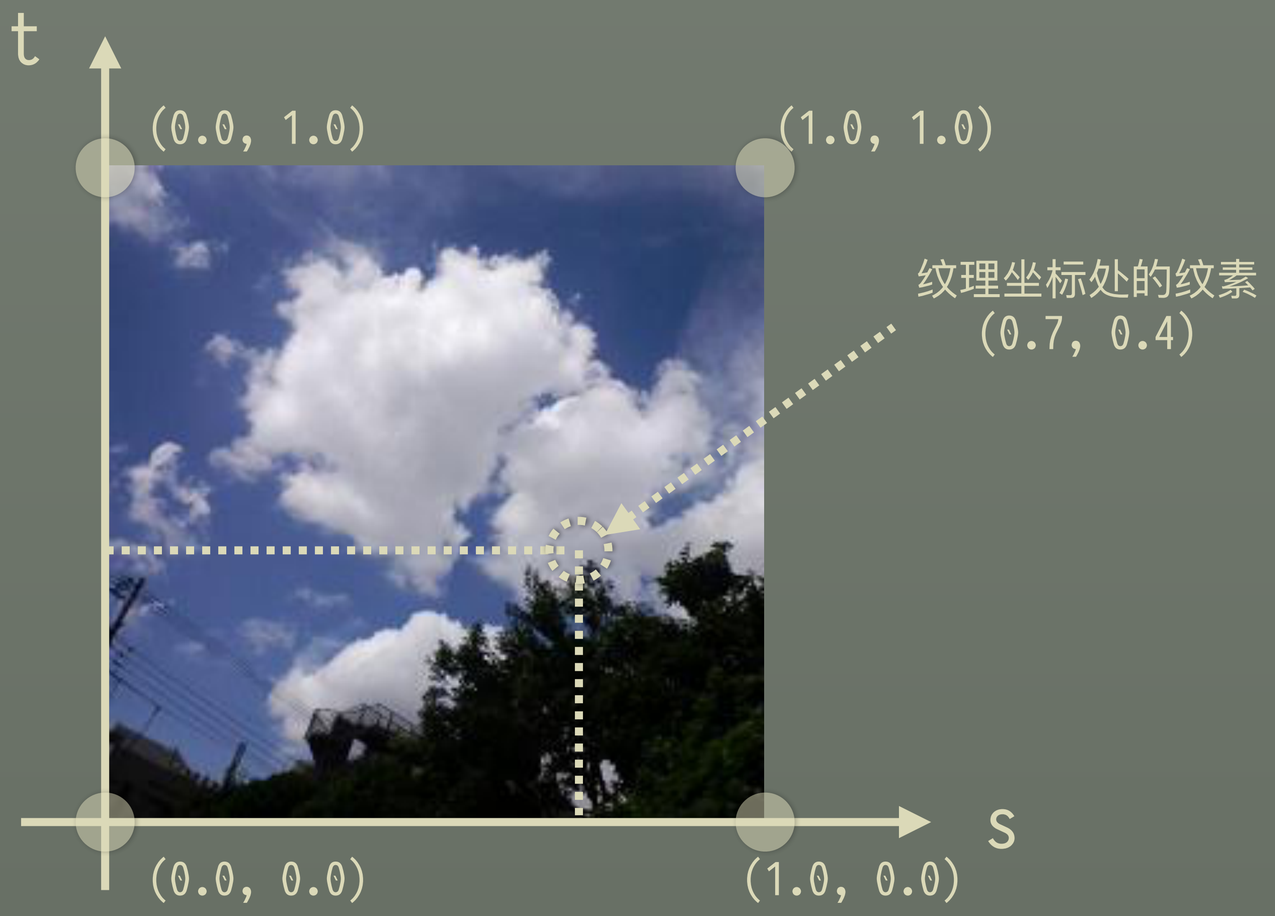
- 准备好纹理图像
- 为几何图形配置纹理映射方式
- 加载纹理图像，进一步配置
- 在片元着色器中将对应的纹素从纹理中提取出来，并且将纹素赋给片元

纹理图像与纹素

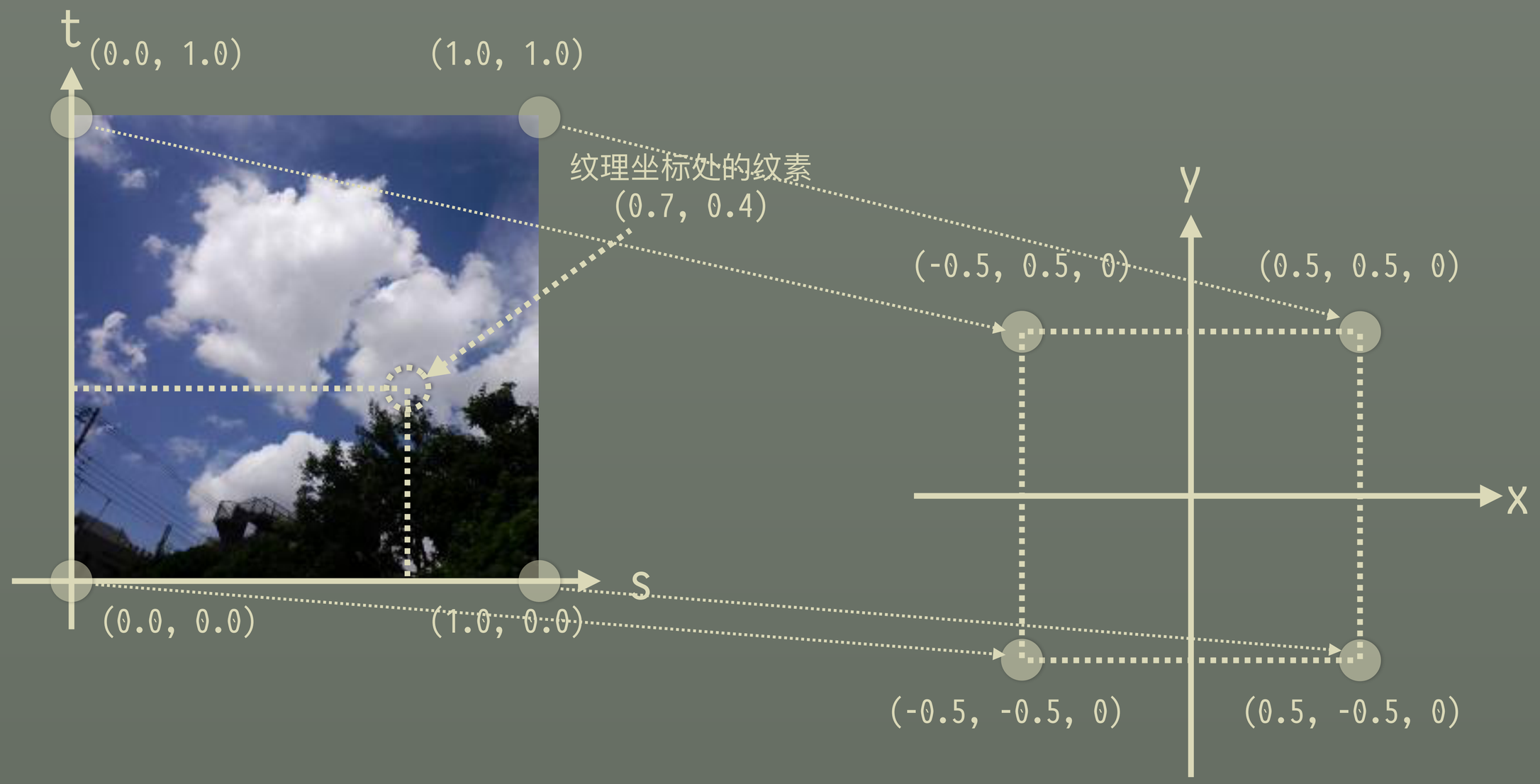
纹理图像

每一个方格(像素)就是纹素

WebGL 纹理坐标系统



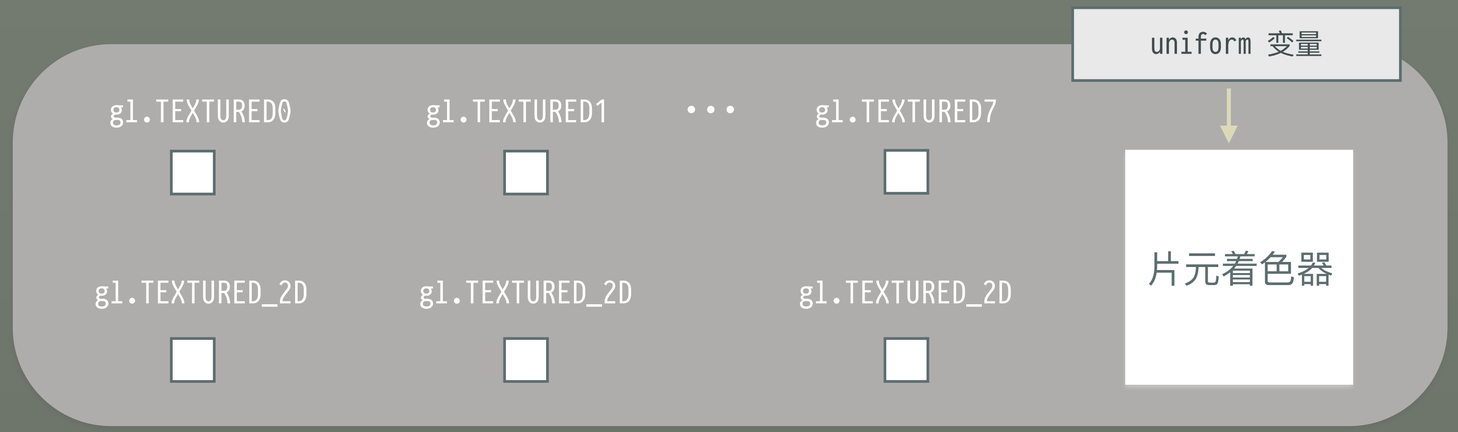
纹理坐标到顶点的映射


t
(0.0, 1.0)
(1.0, 1.0)
纹理坐标处的纹素
(0.7, 0.4)
s
(0.0, 0.0)
(1.0, 0.0)
y
x
(-0.5, 0.5, 0)
(0.5, 0.5, 0)
(-0.5, -0.5, 0)
(0.5, -0.5, 0)

创建纹理对象

JavaScript

```
function main() {
  var gl=getWebGL...
}
```



JavaScript

```
function main() {
  var gl=getWebGL...
  ...
  gl.createTexture(...);
}
```

- gl.TEXTIRE0~7 是管理纹理对象的8个纹理单元
- 每一个都与 gl.TEXTURE_2D 关联
- gl.deleteTexture()

Y 轴翻转

- gl.pixelStorei gl.UNPACK_FLIP_Y_WEBGL, 1

图片坐标系统

纹理坐标系统

gl.pixelStorei(pname, param)		
使用 *pname* 和 *param* 指定的方式处理价在得到的图像		
参数	pname	可以是以下二者之一
	gl.UNPACK_FLIP_Y_WEBGL	指定缓冲区中每个顶点的分量个数（1到4）
	gl.UNPACK_PREMULTIPLY_ALPHA_WEBGL	如过 params 是 true，那么将图像的每一个分量乘以 aplha 通道的值(如果有 alpha)
	param	非 0 (true) 或者 0 (false)。必须为整数
返回值	无	
错误	INVALID_ENUM	pname 不是合法的值

激活纹理单元

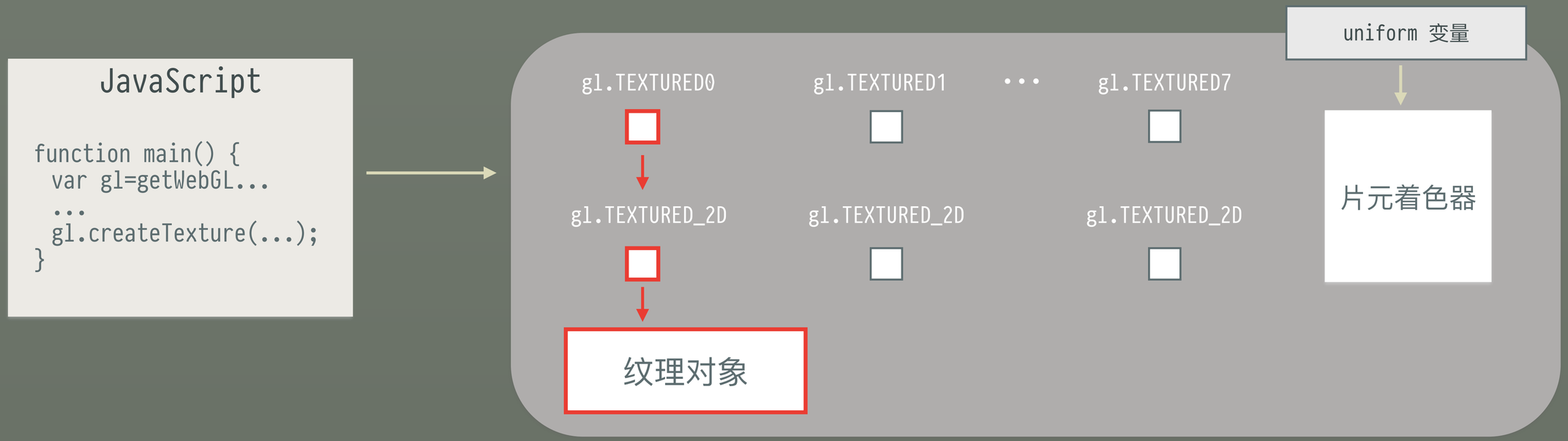
gl.activeTexture(gl.TEXTURE0)

JavaScript

```
function main() {
  var gl=getWebGL...
  ...
  gl.createTexture(...);
}
```

绑定纹理对象

gl.bindTexture(gl.TEXTURE_2D, texture)



WebGL 可以处理 gl.TEXTURE_2D 和 gl.TEXTURE_CUBE_MAP（立方体纹理）

https://www.opengl.org/wiki/Cubemap_Texture

配置纹理参数

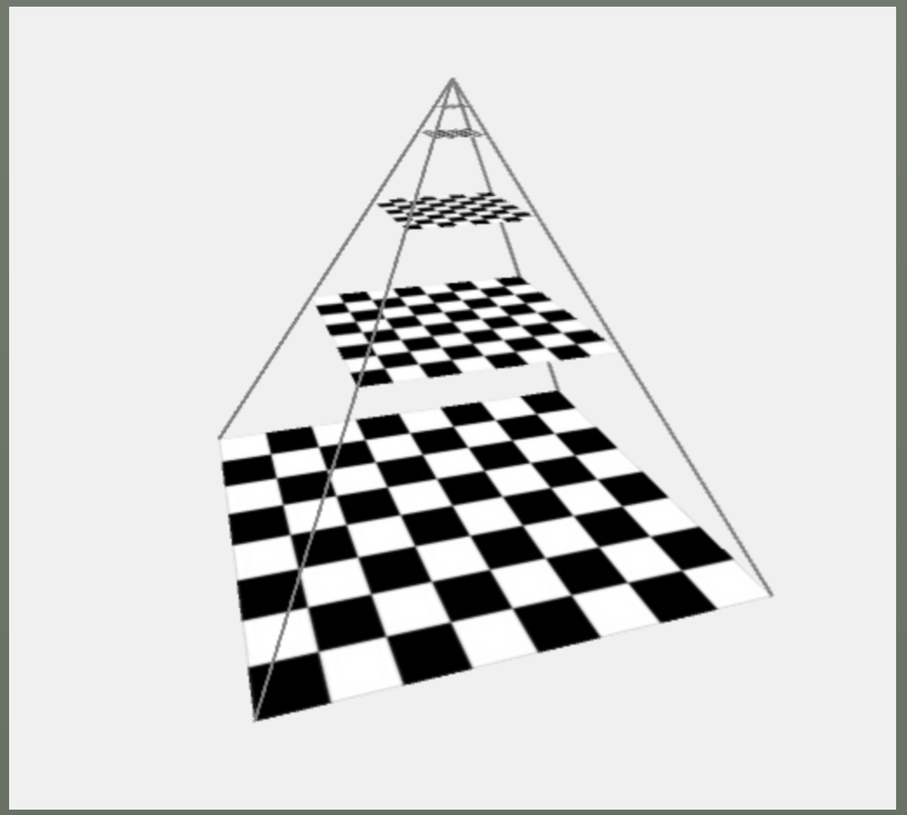
gl.texParameterti(target, name, param)		
将 *param* 的值赋给绑定到目标纹理对象的 *pname* 参数上		
参数	target	gl.TEXTURE_2D 或 gl_TEXTURE_CUBE_MAP
	pname	纹理参数，见下页
	param	纹理参数的值，见下页
返回值	无	
错误	INVALID_ENUM INVALID_OPERATION	target 不合法或者没有绑定到纹理对象

pname 取值

纹理参数	描述	可能的取值	默认值
gl.TEXTURE_MAG_FILTER	纹理放大	gl.NEAREST: 使用纹理上距离映射后像素(新像素)中心最近的那个像素的颜色值 * gl.LINEAR: 距离新像素中心最近的四个像素的颜色的加权平均	gl.LINEAR
gl.TEXTURE_MIN_FILTER	纹理缩小	同上	gl.NEAREST_MIPMAP_LINEAR (见下页)
gl.TEXTURE_WRAP_S	纹理水平填充	gl.REPEAT: 平铺式的重复纹理 gl.MIRRORED_REPEAT: 镜像对称式的重复 gl.CLAMP_TO_EDGE: 用纹理图像的边缘值来重复	gl.REPEAT
gl.TEXTURE_WRAP_T	纹理垂直填充	同上	gl.REPEAT

*: 距离计算采用的是"曼哈顿举例", |x1-x2| + |y1-y2|

MIPMAP

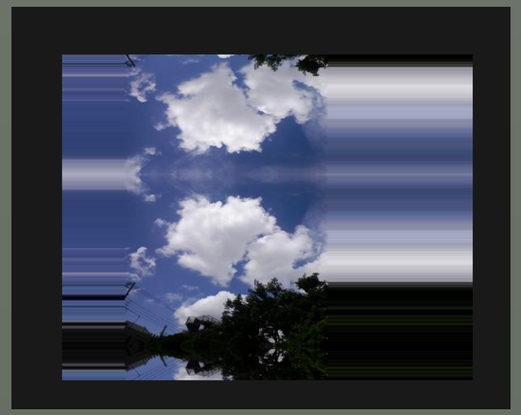


拉丁语的 *“multum in parvo”*，大意为英语的*“much in little”*-“小中见大”

你可以预先生成好 MIPMAP，也可以让 WebGL 帮你生成。每一层纹理尺寸缩小一半，最小到 1x1

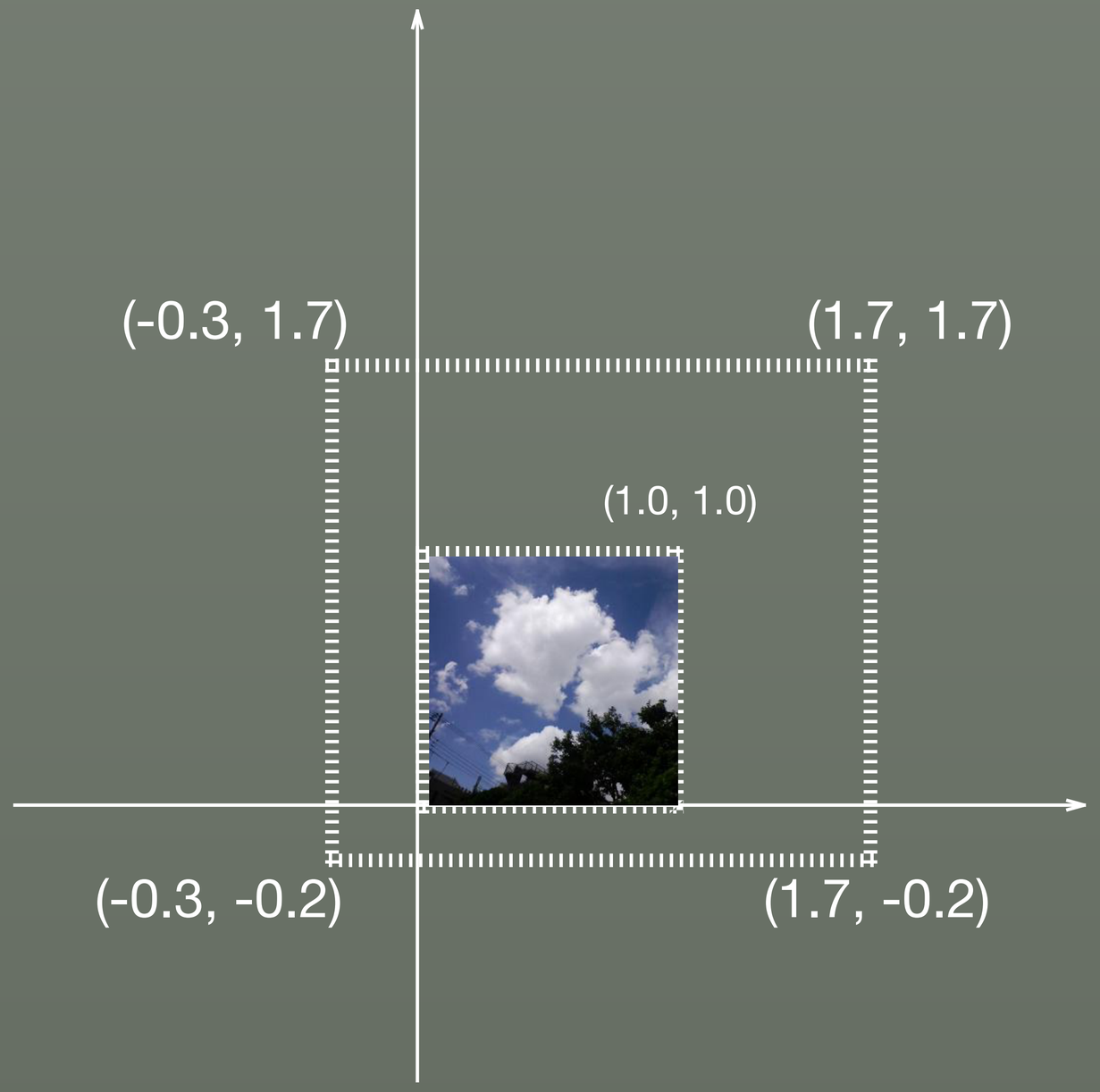
纹理参数

- texturedQuad_Clamp_Mirror.coffee



纹理的扩展

```
[
    -0.5,  0.5,   -0.3, 1.7,
   -0.5, -0.5,   -0.3, -0.2,
    0.5,  0.5,   1.7, 1.7,
    0.5, -0.5,   1.7, -0.2
]
```



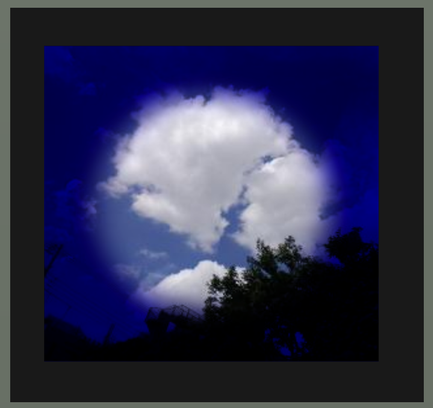
将纹理图像分配给纹理对象

gl.texImage2D(target, level, internalFormat, format, type, image)			
将 *image* 指定的图像分配给绑定到目标上的纹理对象			
参数	target		gl.TEXTURE_2D 或者 gl.TEXTURE_CUBE_MAP
	level	表示传入图像的金字塔层数	
	internalFormat	图像的内部格式	gl.RGB: 红、绿、蓝 gl.RGBA: 红、绿、蓝、透明度 gl:.ALPHA: (0.0, 0.0, 0.0, 透明度) gl.LUMINANCE: (L, L, L, 1L) 流明 *
	format	取值类型同上	同上
	type	纹理数据的类型	gl.UNSIGNED_BYTE: 无符号整形，每个颜色分量1字节 gl.UNSIGNED_SHORT_5_6_5: RGB 分别占,5,6,5 bits gl.UNSIGNED_SHORT_4_4_4_4: RGBA 分别占 4 bits gl.UNSIGNED_SHORT_5_5_5_1: RGB分别占 4 bits, A 占 1 bit
	image	包含纹理图像的 Image 对象	

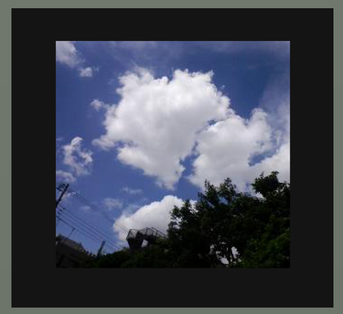
*：流明表示我们感知的物体表面的亮度，亮度纹理可以让我们用比较简单的方法制作出凹凸感

多幅纹理

- multiTexture.coffee



纹理的简单叠加



×

=

```
vec4 color0
     ×
vec4 color1
```

r1	b1	c1	a1
r2	b2	c3	a4
r1*r2	b1*b2	c1*c2	a1*a2

GLSL ES

概述

Vertex Shader

```
attribute   vec4 a_Position;
attribute   vec4 a_Color;
varying     vec4 v_Color;
void main() {
    gl_Position = a_Position;
    v_Color = a_Color;
}
```

Fragment Shader

```
#ifdef GLSL_ES
precision mediump float;
#endif
uniform vec4 u_FragColor;
void main() {
    gl_FragColor = u_FragColor;
}
```

http://www.khronos.org/registry/gles/specs/2.0/GLSL_ES_Specification_1.0.17.pdf

基本特性

- 类 C 语言
- 大小写敏感
- 语句 ';' 分号结束
- 从 `main()` 函数开始执行（有且只有一个，无参）
- // 和 */ */ 注释
- 不能有递归函数

基本类型

- float: 单精度浮点类型
- int: 整型数
- bool: 布尔值

赋值和类型转换

- `int i = 8;  // 正确`
- `float f1 = 8;  // 错误`
- `float f2 = 8.0; // 正确`
- `float f3 = 8.0f;// 错误`
- `int i = 8; float f1 = float(i); //将整型转为float`

矢量和矩阵类型

- vec2, vec3, vec4, ivec2, ivec3, ivec4, bvec2, bvec3, bvec4
- mat2, mat3, mat4

赋值和构造

- vec4 position = vec4(1.0, 2.0, 3.0, 4.0);
- vec3 v3 = vec3(1.0, 0.0, 0.5);
- vec2 v2 = vec2(v3); //v2被设置为v3的前两个元素(1.0, 0.0)
- vec4 v4 = vec4(1.0); //v4被设置为(1.0, 1.0, 1.0, 1.0)
- vec4 v4b = vec4(v2, v4); //v4b被设置为(1.0, 0.0, 1.0, 1.0)

矩阵构造

- mat4 = m4 = mat4(1.0, 2.0, ... 16.0); //列主序

- vec2 v2_1 = vec2(1.0, 3.0);
- vec2 v2_2 = vec2(2.0, 4.0);
- mat2 m2_1 = mat2(v2_1, v2_2);

$$\begin{pmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \end{pmatrix}$$

- mat2 m2 = mat2(1.0, 3.0, v2_2);
- mat4 m4 = mat4(1.0);

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- mat4 m4 = mat4(1.0, 2.0, 3.0); //错误

访问元素

- 分量访问
  - x, y, z, w
  - r, g, b, a
  - s, t, p, q

分量访问

```
vec3 v3 = vec3(1.0, 2.0, 3.0);
float f;
f = v3.x; //f = 1.0
f= v3.r; //f = 1.0
f= v3.t; //f = 1.0
f = v3.w;  //出错
vec2 v2;
v2 = v3.xy; //v2 = (1.0, 2.0);
v2 = v3.yz; //v2 = (2.0, 3.0);
v2 = v3.xx; //v2 = (1.0, 1.0);
vec4 position = vec4(1.0, 2.0, 3.0, 4.0);
position.xw = vec2(5.0, 6.0);   //聚合分量也可用于赋值
```

取样器(Sampler)

- uniform sampler2D u_Sampler;
- 用于访问纹理数据
- 片元着色器最少可以有8个取样器变量
- 而顶点着色器则不能使用取样器变量

运算、程序流和函数

- if else
- for:
  - 只能有一个循环变量，其必须是 int 或 float
  - 循环体内不能修改循环变量

精度限定符

```
precision   mediump float;
varying     vec4    v_Color;
void main() {
    gl_FragColor = v_Color;
}
```

精度限定字

精度限定字	描述	默认数值范围和精度	
		float	int
highp	高精度，顶点着色器的默认精度	(-2^62, 2^62) 精度 2^-16	(-2^16, 2^16)
mediump	中精度，片元着色器的最低精度	(-2^14, 2^14) 精度 2^-10	(-2^10, 2^10)
lowp	低精度，可以表示所有颜色	(-2, 2)　精度 2^-8	(-2^8, 2^8)

IEEE 754：单精度浮点数（-2^127，2^127），精度 2^-23

数据类型的默认精度

着色器类型	数据类型	默认精度
顶点着色器	int	highdp
	float	highdp
	sampler2D	lowp
	sampleCube	lowp
片元着色器	int	mediump
	float	无
	sampler2D	lowp
	sampleCube	lowp

进入3D世界

观察目标点

(atX, atY, atZ)

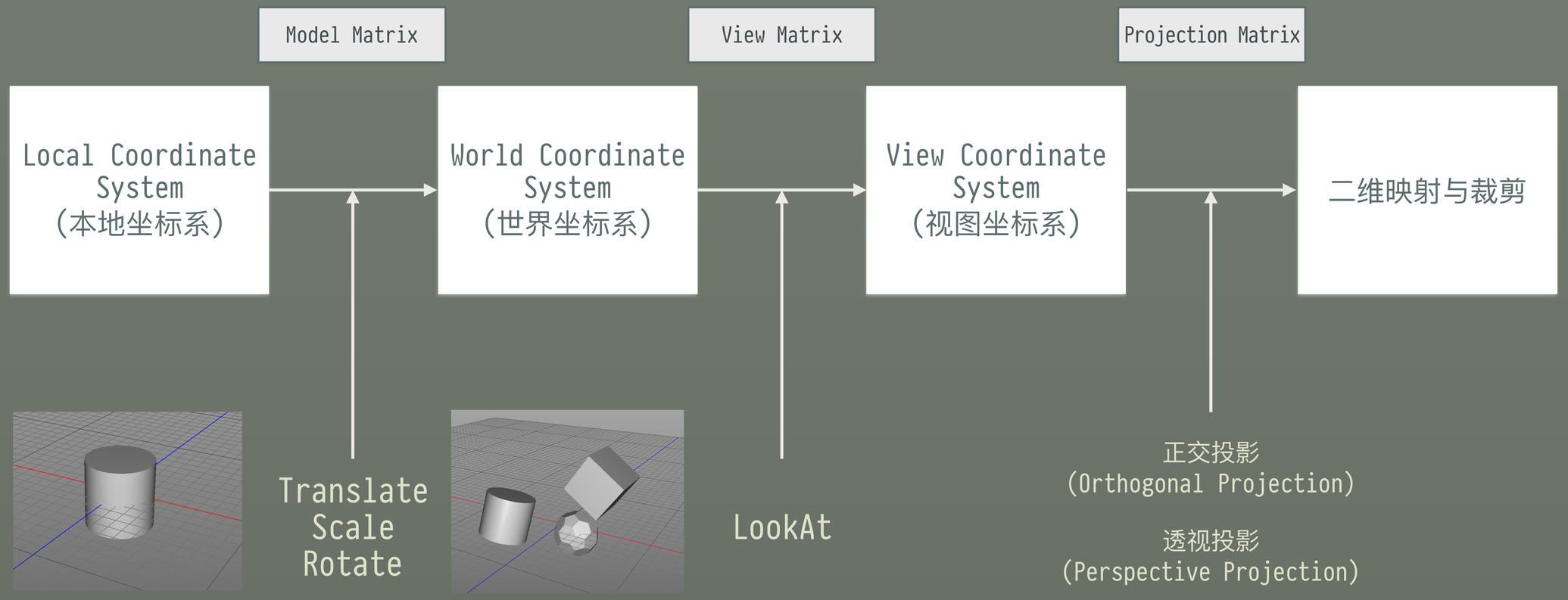
上方向 (upX, upY, upZ)

视点

(eyeX, eyeY, eyeZ)

- 视点：观察者所在的三维空间位置
- 观察目标点：被观察者所在的点
- 上方向： 最终你希望屏幕上的上方向，在空间中的方向

从几何物体到你看到的


Model Matrix
View Matrix
Projection Matrix
Local Coordinate System (本地坐标系)
World Coordinate System (世界坐标系)
View Coordinate System (视图坐标系)
二维映射与裁剪
Translate
Scale
Rotate
LookAt
正交投影 (Orthogonal Projection)
透视投影 (Perspective Projection)

Model Matrix

计算空间中的点在经历变换后在世界坐标系下的位置

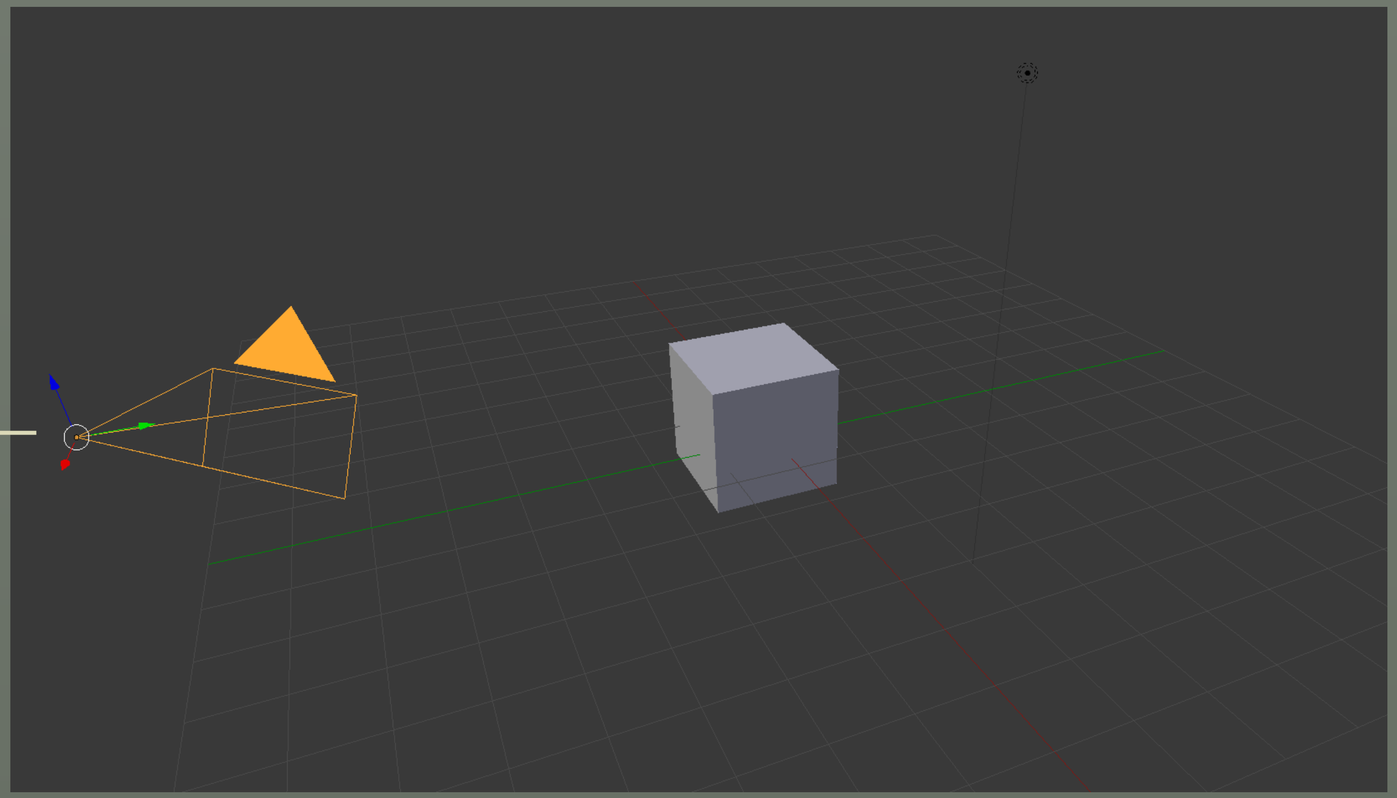
观察者(Camera)

Camera

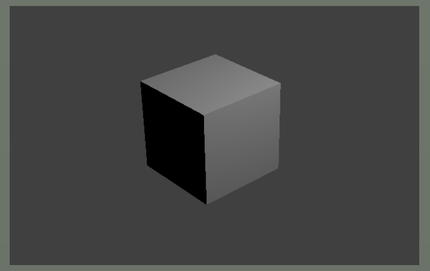
包括视线源点

视线方向

上方向



从 Camera 看到的



观察者和世界是相对变换的

观察者平移了(tx, ty, tz),
相当于整个世界平移了(-tx, -ty, -tz)

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

观察者绕z轴旋转了θ度,
相当于整个世界绕z轴旋转了-θ度

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

观察者缩小了s倍(视野缩小了),相当于世界放大了s倍

$$\begin{pmatrix} \frac{1}{s} & 0 & 0 & 0 \\ 0 & \frac{1}{s} & 0 & 0 \\ 0 & 0 & \frac{1}{s} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

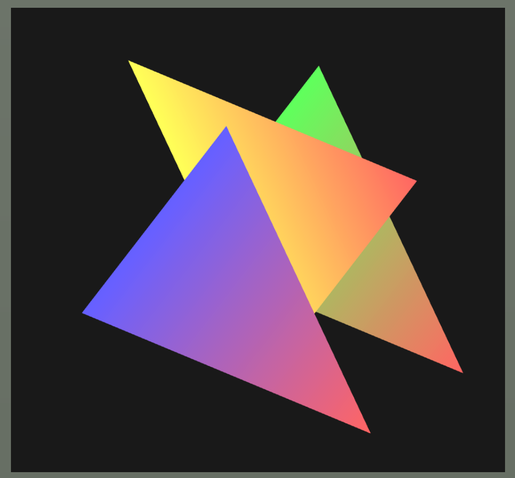
视图矩阵的推导

- 如何根据使用习惯来推导视图矩阵
- 视点：eyeX, eyeY, eyeZ
- 目标点：atX, atY, atZ
- 上方向：upX, upY, upZ 例如：
  - (upX, upY, upZ) = (0, 1, 0)

http://www.cnblogs.com/mikewolf2002/archive/2012/11/25/2787636.html

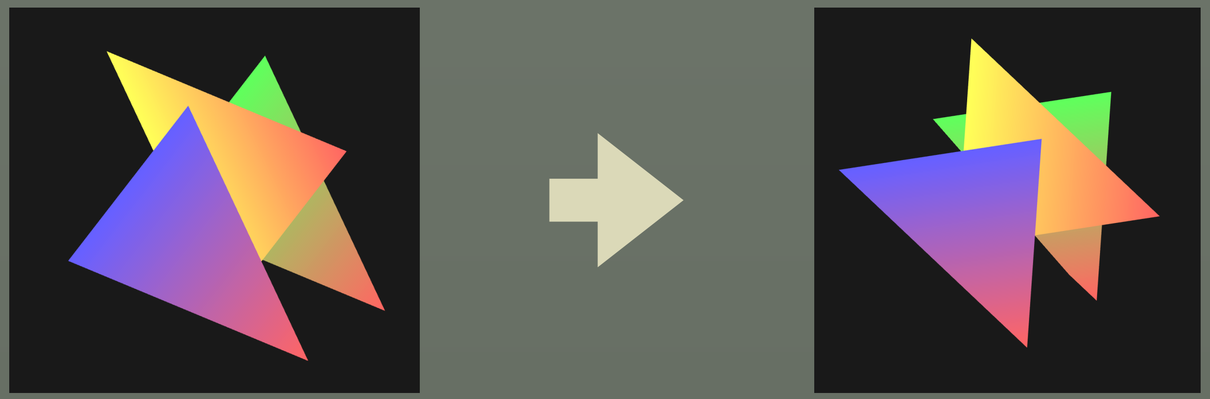
观测者

- lookAtTriangles.coffee



添加变换矩阵

- lookAtRotatedTriangles.coffee



一次性提供计算矩阵

- lookAtRotatedTriangles_mvMatrix.coffee

控制视点移动

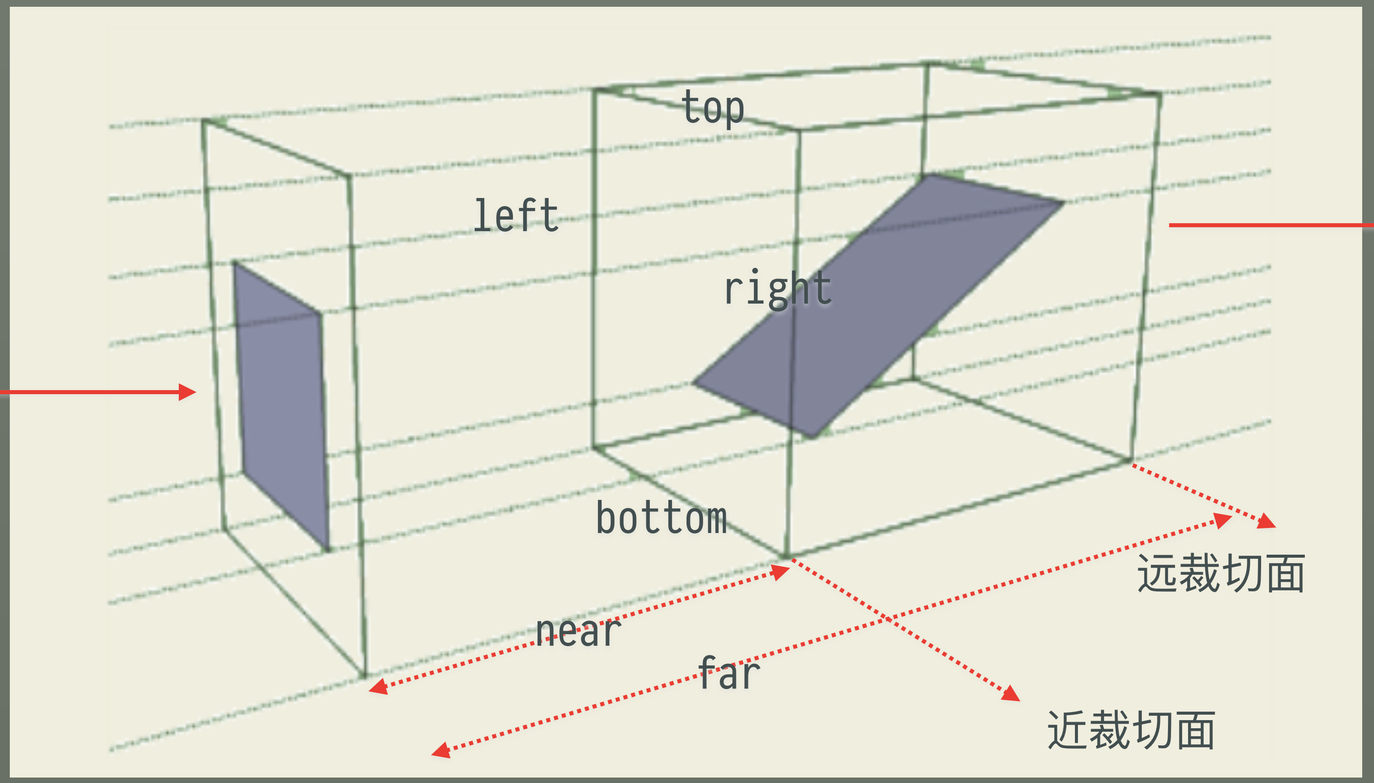
- lookAtTrianglesWithKeys.coffee(Left/Right)

Projection Matrix

- 投影变换是在模型变换和视图变换都完成之后，映射到二维平面用到的变换矩阵

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} x' \\ y' \end{pmatrix}$$

正交投影(orthogonal projection)



投影面

可视空间
(相机空间)
(视景体)
(View Volume)

setOrtho left, right, bottom, top, g_near, g_far

正交投影矩阵的推导

$$\begin{pmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

http://www.codeguru.com/cpp/misc/misc/graphics/article.php/c10123/Deriving-Projection-Matrices.htm#page-2

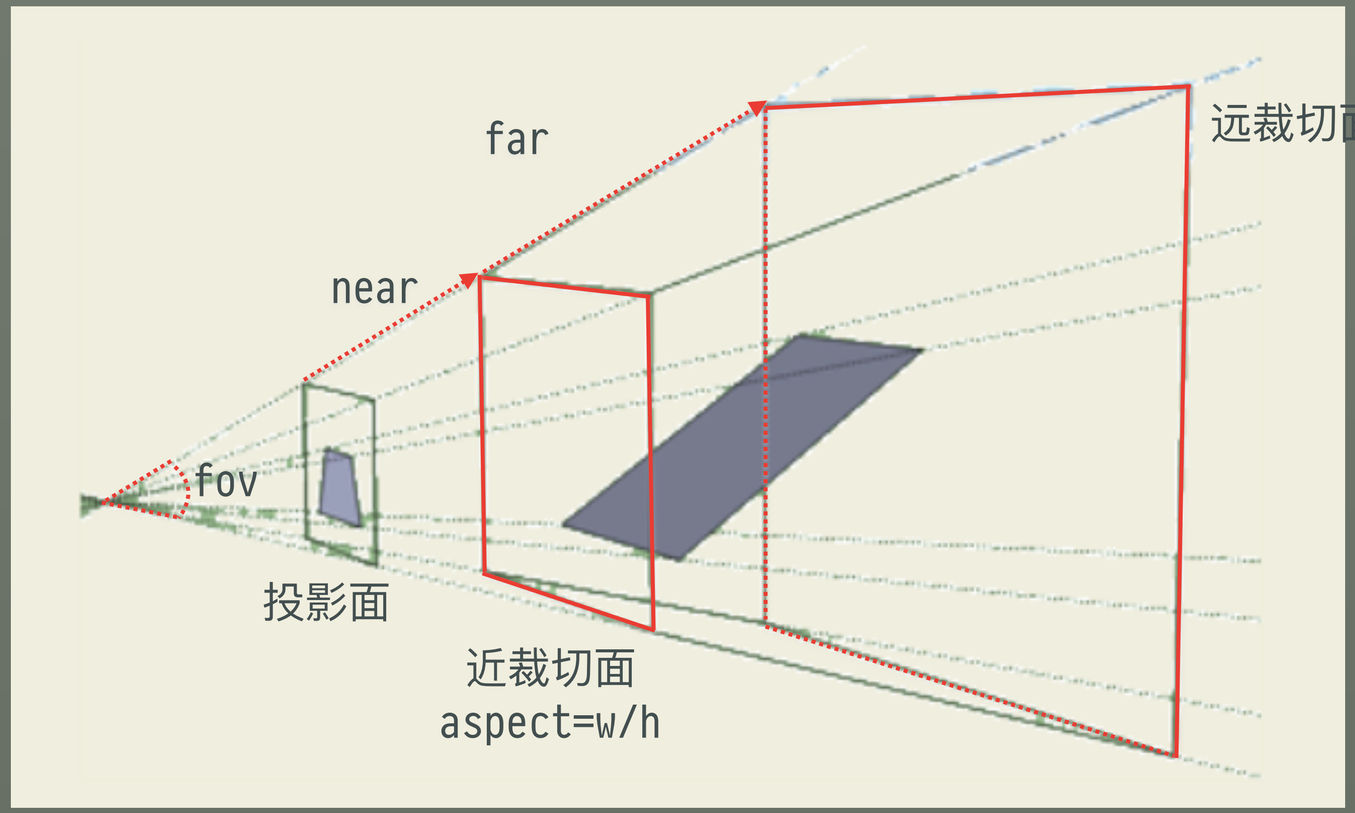
正交投影例子

- orthoView.coffee

补上缺掉的角

- lookAtTrianglesWithKeys_ViewVolume.coffee (Left/Right)

透视投影可视空间



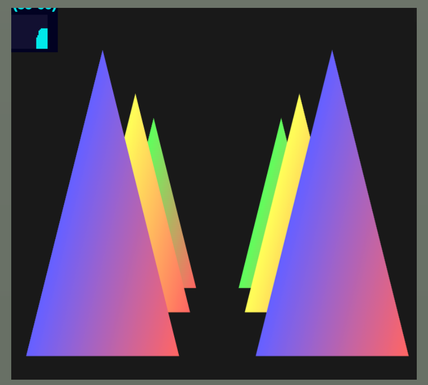
透视投影矩阵的推导

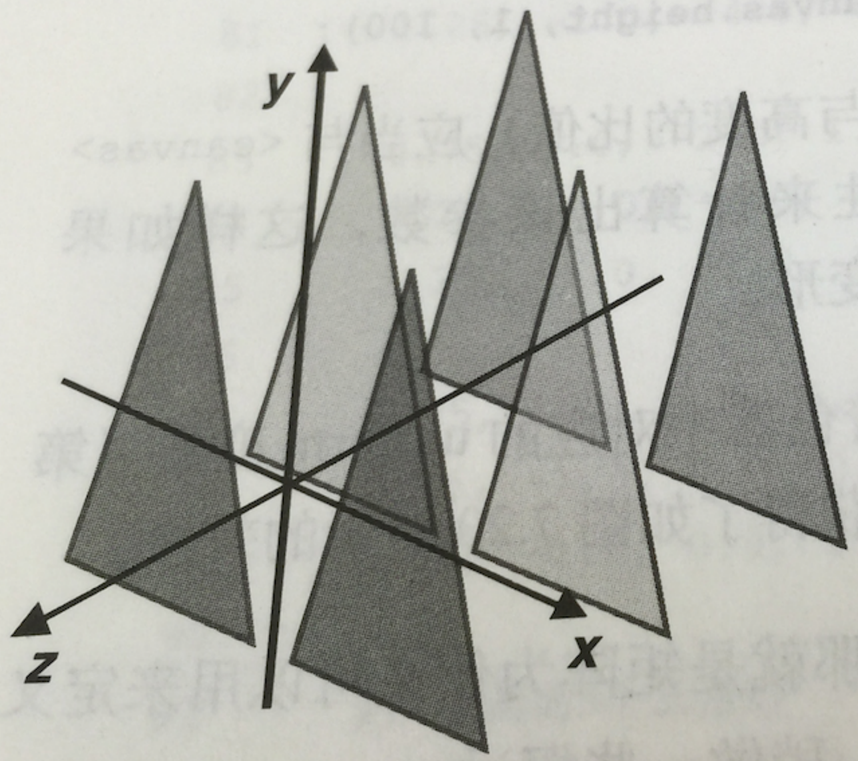
$$\begin{pmatrix} \frac{1}{aspect \times \tan(\frac{fov}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{fov}{2})} & 0 & 0 \\ 0 & 0 & -\frac{far + near}{far - near} & -\frac{2 \times far \times near}{far - near} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

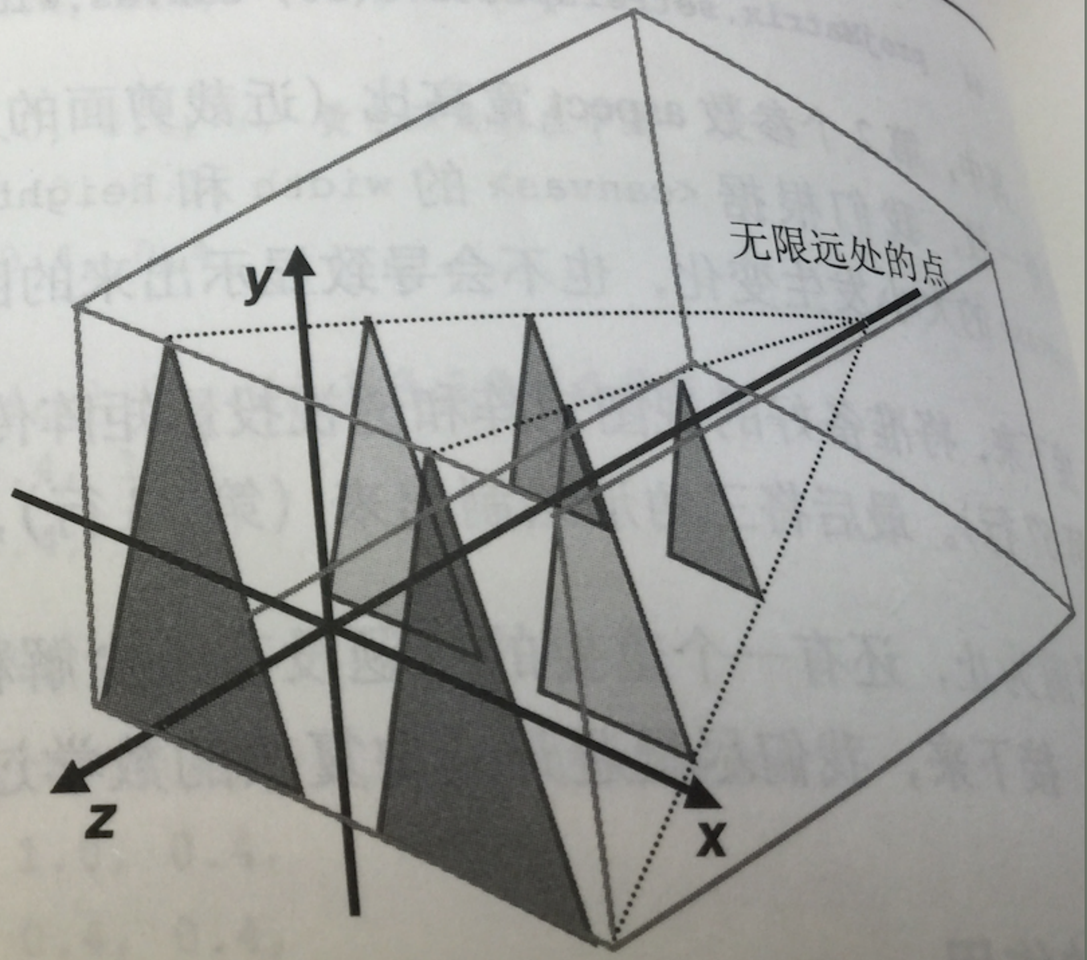
http://www.cnblogs.com/graphics/archive/2012/07/25/2582119.html

透视投影

- perspectiveView.coffee

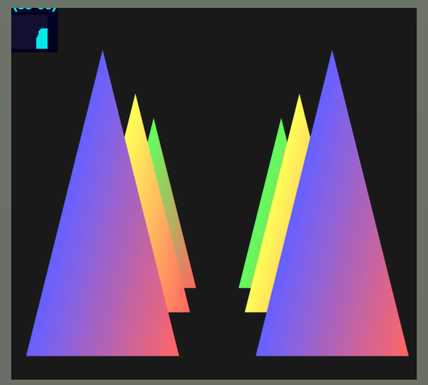



y
z
x

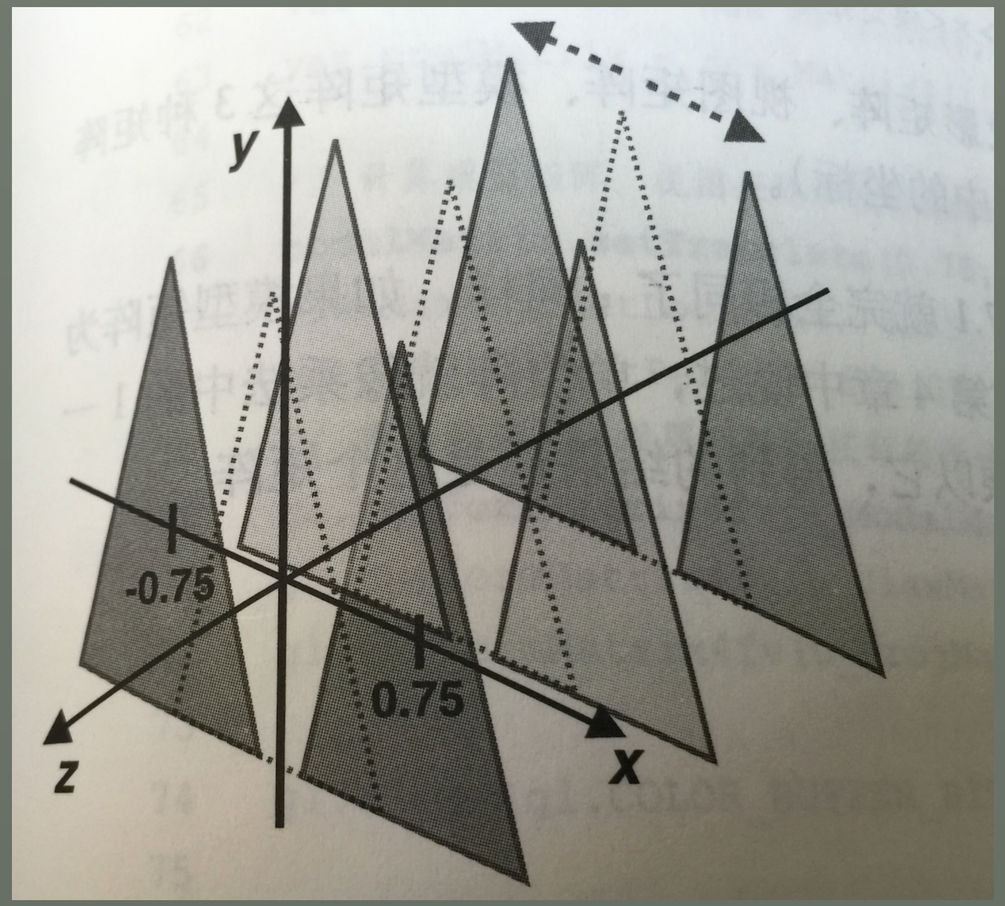

无限远处的点
y
z
x

模型矩阵、视图矩阵和投影矩阵

- perspectiveView-mvpMatrix.coffee

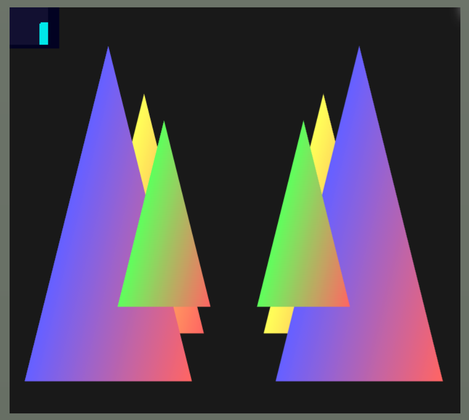


用平移创建矩阵



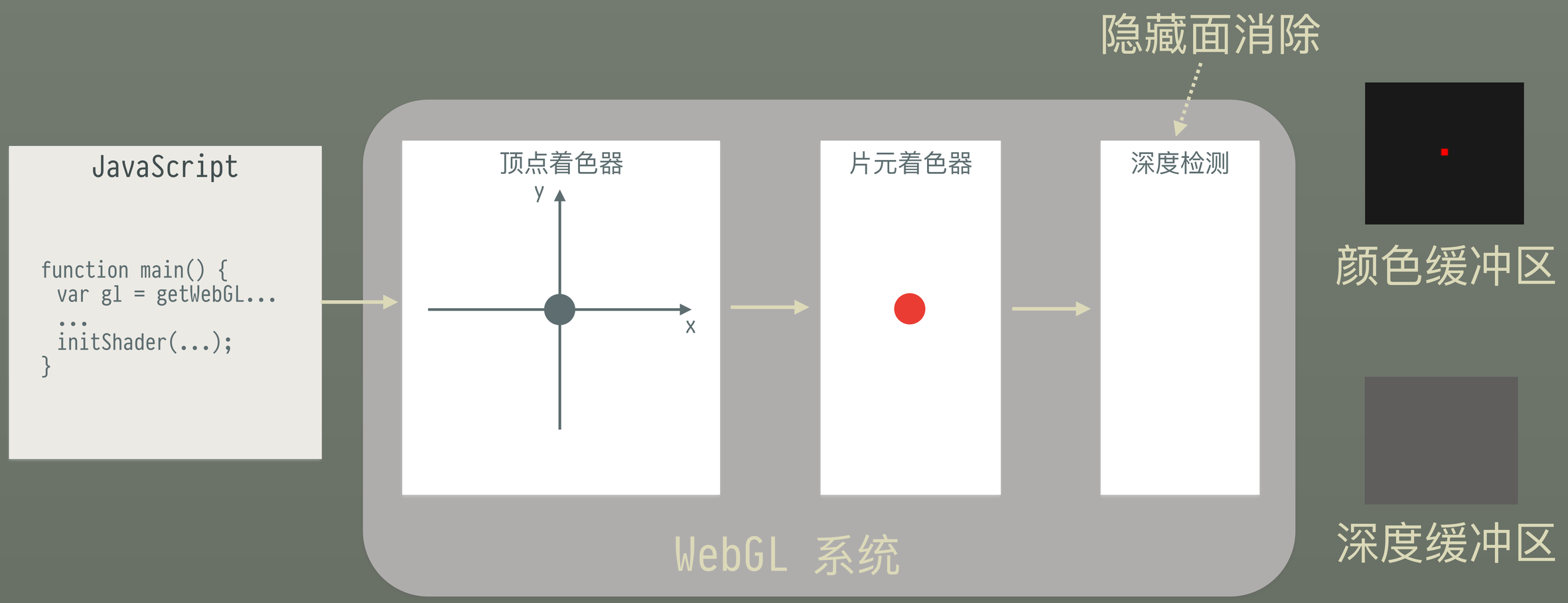
正确处理前后关系

- perspectiveView-mvpReverse.coffee



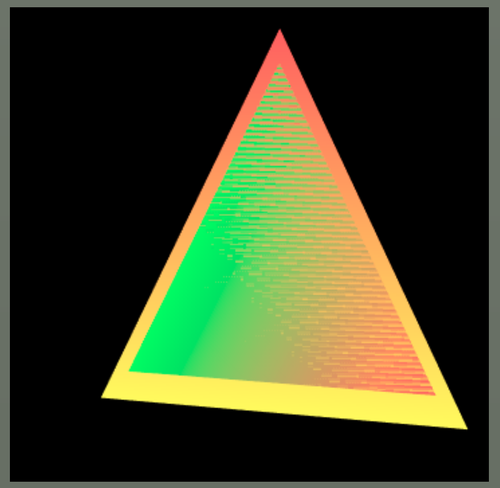
添加深度测试

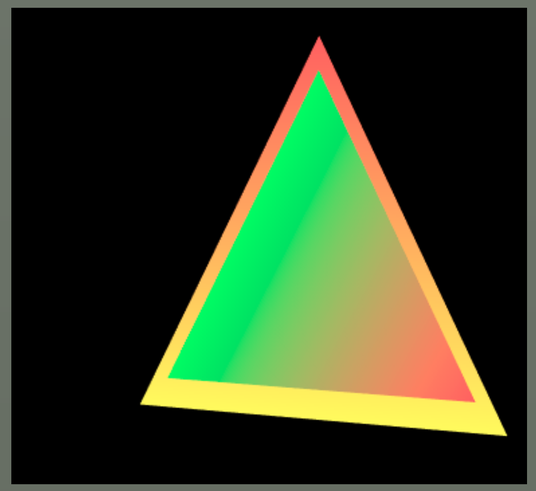
- gl.enable gl.DEPTH_TEST
- gl.clear gl.DEPTH_BUFFER_BIT


隐藏面消除
JavaScript
function main() {
var gl = getWebGL...
...
initShader(...);
}
顶点着色器
y
x
片元着色器
深度检测
WebGL 系统
颜色缓冲区
深度缓冲区

Z-Fighting

- zfighting.coffee





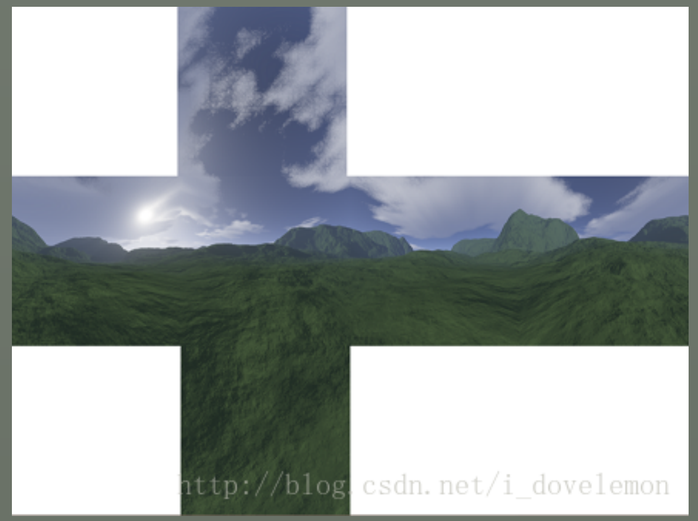
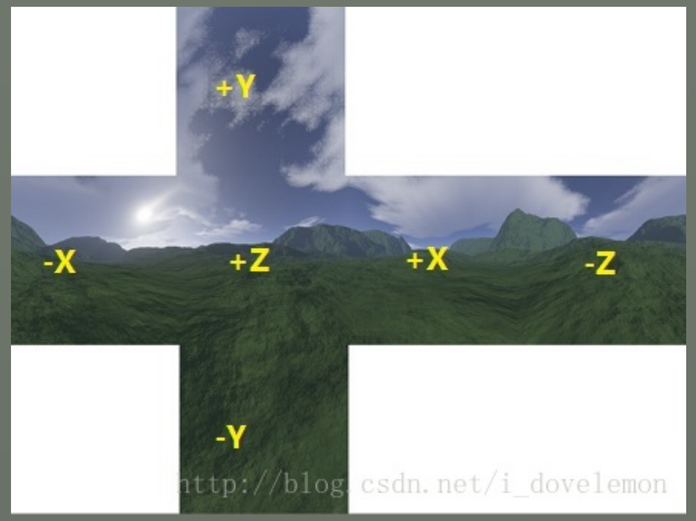
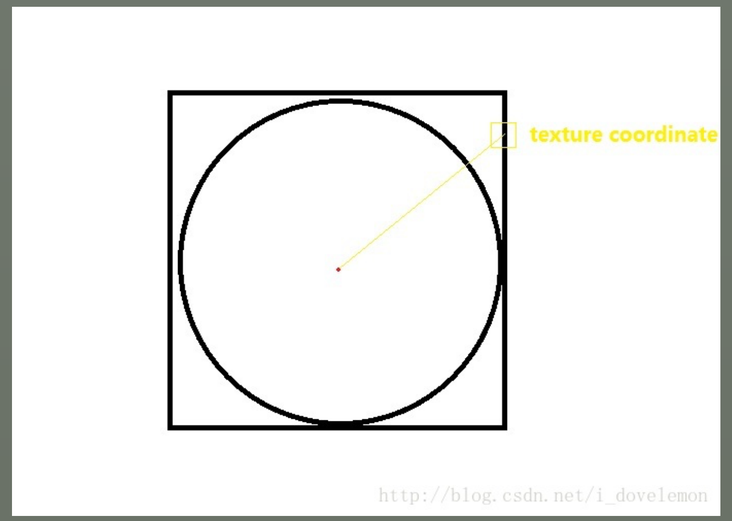
Polygon Offset

- 自动在 z 轴加一个偏移量，偏移量的值由物体表面相对于观察者视线的角度来确定
- 启用该机制：
  - gl.enable(gl.POLYGON_OFFSET_FILL);
- 绘制前：
  - gl.polygonOffset(1.0, 1.0)

gl.polygonOffset(factor, units)

- 指定加到每个顶点绘制后 z 轴上的偏移量，偏移量按照公式
  - m * factor + r * units
  - m 表示顶点所在的表面相对于观察者的视线的角度
  - r 表示硬件能够区分两个 z 值之差的最小值
- 你可以在绘制每一个多边形之前提供不同的 factor, units
  - 内部计数器、时钟变量等

漫长的一课?!

附录

gl.TEXTURE_CUBE_MAP
立方体纹理