

*The Book of Shaders by Patricio Gonzalez Vivo & Jen Lowe**Bahasa Indonesia - Tiếng Việt - 日本語 - 中文版 - 한국어 - Español - Português - Français - Italiano - Deutsch - Русский - English*

Uniforms

现在我们知道了 GPU 如何处理并行线程，每个线程负责给完整图像的一部分配置颜色。尽管每个线程和其他线程之间不能有数据交换，但我们能从 CPU 给每个线程输入数据。因为显卡的架构，所有线程的输入值必须统一（uniform），而且必须设为只读。也就是说，每条线程接收相同的数据，并且是不可改变的数据。

这些输入值叫做 uniform（统一值），它们的数据类型通常为：float, vec2, vec3, vec4, mat2, mat3, mat4, sampler2D and samplerCube。uniform 值需要数值类型前后一致。且在 shader 的开头，在设定精度之后，就对其进行定义。

```
#ifdef GL_ES
precision mediump float;
#endif

uniform vec2 u_resolution; // 画布尺寸（宽，高）
uniform vec2 u_mouse;      // 鼠标位置（在屏幕上哪个像素）
uniform float u_time;       // 时间（加载后的秒数）
```

你可以把 uniforms 想象成连通 GPU 和 CPU 的许多小的桥梁。虽然这些 uniforms 的名字千奇百怪，但是在这一系列的例子中我一直有用到：u_time（时间），u_resolution（画布尺寸）和 u_mouse（鼠标位置）。按业界传统应在 uniform 值的名字前加 u，这样一看即知是 uniform。尽管如此你也还会见到各种各样的名字。比如 [ShaderToy.com](https://shadertoy.com) 就用了如下的名字：

```
uniform vec3 iResolution; // 视口分辨率（以像素计）
uniform vec4 iMouse;      // 鼠标坐标 xy：当前位置，zw：点击位置
uniform float iTime;       // shader 运行时间（以秒计）
```

好了说的足够多了，我们来看看实际操作中的 uniform 吧。在下面的代码中我们使用 u_time 加上一个 sin 函数，来展示图中红色的动态变化。

```
1 | #ifdef GL_ES
```

```

2   precision mediump float;
3   #endif
4
5   uniform float u_time;
6
7   void main() {
8       gl_FragColor = vec4(abs(sin(u_time)), 0.0, 0.0, 1.0);
9   }
10

```

△



GLSL 还有更多惊喜。GPU 的硬件加速支持我们使用角

度，三角函数和指数函数。这里有一些这些函数的介绍：[sin\(\)](#)，[cos\(\)](#)，[tan\(\)](#)，[asin\(\)](#)，[acos\(\)](#)，[atan\(\)](#)，[pow\(\)](#)，[exp\(\)](#)，[log\(\)](#)，[sqrt\(\)](#)，[abs\(\)](#)，[sign\(\)](#)，[floor\(\)](#)，[ceil\(\)](#)，[fract\(\)](#)，[mod\(\)](#)，[min\(\)](#)，[max\(\)](#) 和 [clamp\(\)](#)。

现在又到你来玩的时候了。

- 降低颜色变化的速率，直到肉眼都看不出来。
- 加速变化，直到颜色静止不动。
- 玩一玩 RGB 三个通道，分别给三个颜色不同的变化速度，看看能不能做出有趣的效果。

gl_FragCoord

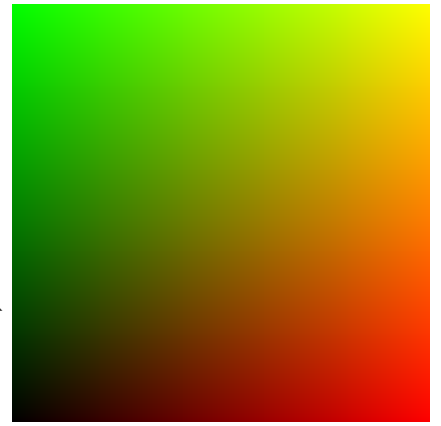
就像 GLSL 有个默认输出值 `vec4 gl_FragColor` 一样，它也有一个默认输入值（`vec4 gl_FragCoord`）。`gl_FragCoord` 存储了活动线程正在处理的像素或屏幕碎片的坐标。有了它我们就知道了屏幕上的哪一个线程正在运转。为什么我们不叫 `gl_FragCoord` **uniform**（统一值）呢？因为每个像素的坐标都不同，所以我们把它叫做 **varying**（变化值）。

```

1   #ifdef GL_ES
2   precision mediump float;
3   #endif
4
5   uniform vec2 u_resolution;
6   uniform vec2 u_mouse;
7   uniform float u_time;
8
9   void main() {
10      vec2 st = gl_FragCoord.xy/u_resolution;

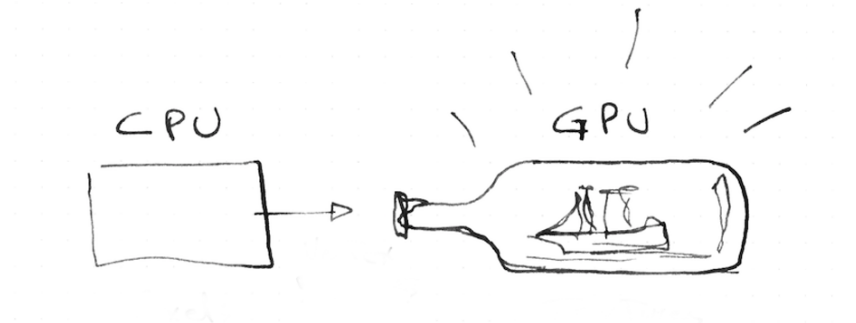
```

```
11 |         gl_FragColor = vec4(st.x, st.y, 0.0, 1.0);  
12 |     }  
13 |
```



上述代码中我们用 `gl_FragCoord.xy` 除以 `u_resolution`，对坐标进行了规范化。这样做是为了使所有的值落在 0.0 到 1.0 之间，这样就可以轻松把 X 或 Y 的值映射到红色或者绿色通道。

在 `shader` 的领域我们没有太多要 `debug` 的，更多地是试着给变量赋一些很炫的颜色，试图做出一些效果。有时你会觉得用 `GLSL` 编程就像是把一艘船放到了瓶子里。它同等地困难、美丽而令人满足。



现在我们来检验一下我们对上面代码的理解程度。

- 你明白 `(0.0, 0.0)` 坐标在画布上的哪里吗？
- 那 `(1.0, 0.0)`, `(0.0, 1.0)`, `(0.5, 0.5)` 和 `(1.0, 1.0)` 呢？
- 你知道如何用未规范化（normalized）的 `u_mouse` 吗？你可以用它来移动颜色吗？
- 你可以用 `u_time` 和 `u_mouse` 来改变颜色的图案吗？不妨琢磨一些有趣的途径。

经过这些小练习后，你可能会好奇还能用强大的 `shader` 做什么。接下来的章节你会知道如何把你的 `shader` 和 `three.js`, `Processing`, 和 `openFrameworks` 结合起来。

< < *Previous* *Home* *Next* > >

