The Book of Shaders by *Patricio Gonzalez Vivo & Jen Lowe*

Bahasa Indonesia - *Tiếng Việt* - 日本□語 - 中文版 - 한국어 - *Español* - *Portugues* - *Français* - *Italiano* - *Deutsch* - *Русский* - *English*

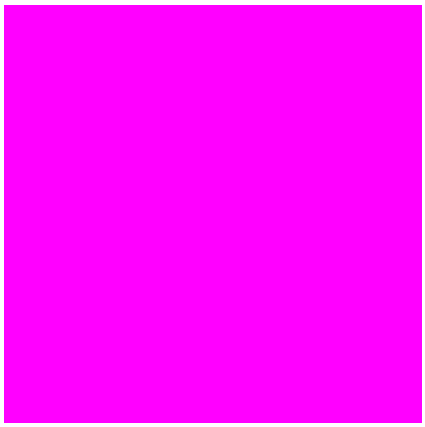
Hello World

“Hello world!”通常都是学习一个新语言的第一个例子。这是一个非常简单，只有一行的程序。它既是一个热情的欢迎，也传达了编程所能带来的可能性。

然而在 GPU 的世界里，第一步就渲染一行文字太难了，所以我们改为选择一个鲜艳的欢迎色，来吧躁起来！

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

△



如果你是在线阅读这本书的话，上面的代码都是可以交互的。你可以点击或者改动代码中任何一部分，尽情探索。多亏 GPU 的架构，shader 会飞速地编译和更新，这使得你的改动都会立刻出现在你眼前。试试改动第 8 行的值，看会发生什么。

尽管这几行简单的代码看起来不像有很多内容，我们还是可以据此推测出一些知识点：

1. shader 语言 有一个 `main` 函数，会在最后返回颜色值。这点和 C 语言很像。
2. 最终的像素颜色取决于预设的全局变量 `gl_FragColor`。
3. 这个类 C 语言有内建的变量（像`gl_FragColor`），函数和数据类型。在本例中我们刚刚介绍了`vec4`（四分量浮点向量）。之后我们会见到更多的类型，像 `vec3`

（三分量浮点向量）和 vec2 （二分量浮点向量），还有非常著名的：float（单精度浮点型）， int（整型） 和 bool（布尔型）。

4. 如果我们仔细观察 vec4 类型，可以推测这四个变元分别响应红，绿，蓝和透明度通道。同时我们也可以看到这些变量是规范化的，意思是它们的值是从0到1的。之后我们会学习如何规范化变量，使得在变量间**map**（映射）数值更加容易。

5. 另一个可以从本例看出来的很重要的类 C 语言特征是，预处理程序的宏指令。宏指令是预编译的一部分。有了宏才可以 #define （定义）全局变量和进行一些基础的条件运算（通过使用 #ifdef 和 #endif）。所有的宏都以 # 开头。预编译会在编译前一刻发生，把所有的命令复制到 #defines 里，检查#ifdef 条件句是否已被定义， #ifndef 条件句是否没有被定义。在我们刚刚的“hello world!”的例子中，我们在第2行检查了 GL_ES 是否被定义，这个通常用在移动端或浏览器的编译中。

6. float类型在 shaders 中非常重要，所以精度非常重要。更低的精度会有更快的渲染速度，但是会以质量为代价。你可以选择每一个浮点值的精度。在第一行（precision mediump float;）我们就是设定了所有的浮点值都是中等精度。但我们也可以选择把这个值设为“低”（precision lowp float;）或者“高”（precision highp float;）。

7. 最后可能也是最重要的细节是，GLSL 语言规范并不保证变量会被自动转换类别。这句话是什么意思呢？显卡的硬件制造商各有不同的显卡加速方式，但是却被要求有最精简的语言规范。因而，自动强制类型转换并没有包括在其中。在我们的“hello world!”例子中，vec4 精确到单精度浮点，所以应被赋予 float 格式。但是如果你想要代码前后一致，不要之后花费大量时间 debug 的话，最好养成在 float 型数值里加一个 . 的好习惯。如下这种代码就可能不能正常运行：

```
void main() {
    gl_FragColor = vec4(1,0,0,1);   // 出错
}
```

现在我们已经基本讨论完了“hello world!”例子中所有主要的内容，是时候点击代码，检验一下我们所学的知识了。你会发现出错时程序会编译失败，只留一个寂寞

的白屏。你可以试试一些好玩的小点子，比如说：

- 把单精度浮点值换成整型数值，猜猜你的显卡能不能容忍这个行为。
- 试试把第八行注释掉，不给函数赋任何像素的值。
- 尝试另外写个函数，返回某个颜色，然后在 `main()` 里面使用这个函数。给个提示，这个函数应该长这样：

```
vec4 red(){
    return vec4(1.0,0.0,0.0,1.0);
}
```

- 有很多种构造 `vec4` 类型的方式，试试看其他方式。下面就是其中一种方式：

```
vec4 color = vec4(vec3(1.0,0.0,1.0),1.0);
```

尽管这个例子看起来不那么刺激，它却是最最基础的 —— 我们把画布上的每一个像素都改成了一个确切的颜色。在接下来的章节中我们将会看到如何用两种输入源来改变像素的颜色：空间（依据像素在屏幕上的位置）和时间（依据页面加载了多少秒）。

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