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*

Turn off the lights

Hello World

Usually the "Hello world!" example is the first step to learning a new language. It's a simple one-line program that outputs an enthusiastic welcoming message and declares opportunities ahead.

In GPU-land rendering text is an overcomplicated task for a first step, instead we'll choose a bright welcoming color to shout our enthusiasm!

```
1   #ifdef GL_ES
2   precision mediump float;
3   #endif
4
5   uniform float u_time;
6
7   void main() {
8       gl_FragColor = vec4(0.295,0.385,0.265,1.000);
9   }
10
```

If you are reading this book in a browser the previous block of code is interactive. That means you can click and change any part of the code you want to explore. Changes will be updated immediately thanks to the GPU architecture that compiles and replaces shaders *on the fly*. Give it a try by changing the values on line 8.

Although these simple lines of code don't look like a lot, we can infer substantial knowledge from them:

1. Shader Language has a single `main` function that returns a color at the end. This is similar to C.

2. The final pixel color is assigned to the reserved global variable `gl_FragColor`.

3. This C-flavored language has built in *variables* (like `gl_FragColor`), *functions* and *types*. In this case we've just been introduced to `vec4` that stands for a four dimensional vector of floating point precision. Later we will see more types like `vec3` and `vec2` together with the popular: `float`, `int` and `bool`.

4. If we look closely to the `vec4` type we can infer that the four arguments respond to the RED, GREEN, BLUE and ALPHA channels. Also we can see that these values are *normalized*, which means they go from `0.0` to `1.0`. Later, we will learn how normalizing values makes it easier to *map* values between variables.

5. Another important *C feature* we can see in this example is the presence of preprocessor macros. Macros are part of a pre-compilation step. With them it is possible to `#define` global variables and do some basic conditional operation (with `#ifdef` and `#endif`). All the macro commands begin with a hashtag (`#`). Pre-compilation happens right before compiling and copies all the calls to `#defines` and check `#ifdef` (is defined) and `#ifndef` (is not defined) conditionals. In our "hello world!" example above, we only insert the line 2 if `GL_ES` is defined, which mostly happens when the code is compiled on mobile devices and browsers.

6. Float types are vital in shaders, so the level of *precision* is crucial. Lower precision means faster rendering, but at the cost of quality. You can be picky and specify the precision of each variable that uses floating point. In the second line (`precision mediump float;`) we are setting all floats to medium precision. But we can choose to set them to low (`precision lowp float;`) or high (`precision highp float;`).

7. The last, and maybe most important, detail is that GLSL specs don't guarantee that variables will be automatically casted. What does that mean? Manufacturers have different approaches to accelerate graphics card processes but they are forced to guarantee minimum specs. Automatic casting is not one of them. In our "hello world!" example `vec4` has floating point precision and for that it expects to be assigned with `floats`. If you want to make good consistent code and not spend hours debugging white screens, get used to putting the point (`.`) in your floats. This kind of code will not always work:

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

Now that we've described the most relevant elements of our "hello world!" program, it's time to click on the code block and start challenging all that we've learned. You will note that on errors, the program will fail to compile, showing a white screen. There are some interesting things to try, for example:

- Try replacing the floats with integers, your graphic card may or may not tolerate this behavior.

- Try commenting out line 8 and not assigning any pixel value to the function.

- Try making a separate function that returns a specific color and use it inside `main()`. As a hint, here is the code for a function that returns a red color:

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

- There are multiple ways of constructing `vec4` types, try to discover other ways. The following is one of them:

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

Although this example isn't very exciting, it is the most basic example - we are changing all the pixels inside the canvas to the same exact color. In the following chapter we will see how to change the pixel colors by using two types of input: space (the place of the pixel on the screen) and time (the number of seconds since the page was loaded).

< < Previous *Home* *Next > >*