# A Brief Primer on GLSL - The GL Shader Language

Vertex and Fragment shaders are written in GLSL, a domain-specific mini-language for writing shaders. The syntax is C/C++-like, and basic shaders tend to look like very simple C programs, such as:

```
// some declarations here

void main() {
    // main program here
}
```

## Shader Responsibilities

The **vertex shader's job** is to:

- assign a value to the built-in magic variable `gl_Position`, which is a `vec4` that contains the vertex's position.
- assign values to any `varying` parameters needed for the Fragment shader

The **fragment shader's job** is to:

- assign a value to the built-in magic variable `gl_FragColor`, which is a `vec4` containing the fragment's color.

## GLSL Features

Here are some things about GLSL that are not C-like:

- Types: `vec2`, `vec3`, `vec4`, `mat2`, `mat3`, `mat4` are built-in vector and matrix types; their values are floats.
- Multiplication involving these types is matrix/vector multiplication
- Colors are 4 channels. The first 3 are RGB, and the 4th is transparency; 0 is fully transparent, 1 is fully opaque.
- `varying`s are declared in both the Vertex shader and in the Fragment shader. By convention, their names are usually chosen to begin with `v`, such as `vColor` or `vNormal`.
- Vector types support some neat shorthand. For example:
    - In the following line `Position` is a `vec3`, and its entries become the first three entries of the `vec4` being set to `gl_Position`:

```
gl_Position = vec4(Position, 1.0);
```

- Accessing elements of a vector using attribute names:

```
vec4 a = vec4(1, 2, 3, 4);
float x = a.x; // x = 1.0
float y = a.y; // x = 2.0
float z = a.z; // x = 3.0
float w = a.w; // x = 4.0
```

This works for shorter vectors too, within bounds; you can also use `r,g,b,a`, and `s,t,p,q` as alternative accessors.

- **Swizzling** allows you to easily construct one vector from elements of another:

```
vec4 a = vec4(1, 2, 3, 4);

vec3 b = a.xyz;  // b = (1,2,3)
vec2 c = a.qp;   // c = (4,3)
vec4 d = a.xxyy; // d = (1,1,2,2)
```

## Examples of working with vector and matrix types

- `vec2`

```
vec2 a;
a.x = 0.0;
a.y = 1.0; // a = (0,1)

vec2 b;
b.s = 10.0;
b.t = 12.5; // b = (10,12.5)

vec2 c;
c[0] = 9.0;
c[1] = 8.0; // c= (9,8)
```

- `vec3`

```
vec3 a;
a.x = 10.0; a.y = 20.0; a.z = 30.0; // a = (10,  20,  30)
a.r = 0.1;  a.g = 0.2;  a.b = 0.3;  // a = (0.1, 0.2, 0.3)
a.s = 1.0;  a.t = 2.0;  a.p = 3.0;  // a = (1, 2, 3)

```

```
vec3 b = vec3(4.0, 5.0, 6.0);

vec3 c = a + b;       // c = (5, 7, 9)
vec3 d = a - b;       // d = (-3, -3, -3)
vec3 e = a * b;       // e = (4, 10, 18)
vec3 f = a * 3;       // e = (3, 6, 9)
float g = dot(a, b); // g = 32
vec3 h = cross(a,b); // h = (-5,6,-3)
float i = length(a); // i = 3.742
```

- `vec4`

```
vec4 a;
a.x = 10.0; a.y = 20.0; a.z = 30.0; a.w = 40.0; // a = (10, 20, 30, 40)
a.r = 0.1;  a.g = 0.2;  a.b = 0.3;  a.a = 0.4;  // a = (0.1, 0.2, 0.3, 0.4)
a.s = 1.0;  a.t = 2.0;  a.p = 3.0;  a.q = 4.0;  // a = (1, 2, 3, 4)

vec4 b = vec4(5, 6, 7, 8);

vec4 c = a + b;       // c = (6, 8, 10, 12)
vec4 d = a - b;       // d = (-4, -4, -4, -4)
vec4 e = a * b;       // e = (5, 12, 21, 32)
vec4 f = a * 3;       // f = (3, 6, 9, 12)
float g = length(a); // g = 5.477
```

- `mat2`

```
mat2 A = mat2(1.0,2.0,3.0,4.0); // in column-major order

vec2 x = vec2(1.0, 0.0);
vec2 y = vec2(0.0, 1.0);

vec2 a = A * x; // a = (1,2)
vec2 b = A * y; // b = (3,4)
```

- `mat3`

```
mat3 A = mat3(1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0); // in column-major order

vec3 x = vec3(1.0, 0.0, 0.0);
vec3 y = vec3(0.0, 1.0, 0.0);
vec3 z = vec3(0.0, 0.0, 1.0);

vec3 a = A * x; // a = (1,2,3)
vec3 b = A * y; // b = (4,5,6)
vec3 c = A * z; // c = (6,7,8)
```

- `mat4`

```
mat4 A = mat4(
    1.0,  2.0,  3.0,  4.0,
    5.0,  6.0,  7.0,  8.0,
    9.0,  10.0, 11.0, 12.0,
    13.0, 14.0, 15.0, 16.0); // in column-major order

vec4 x = vec4(1.0, 0.0, 0.0, 0.0);
vec4 y = vec4(0.0, 1.0, 0.0, 0.0);
vec4 z = vec4(0.0, 0.0, 1.0, 0.0);
vec4 w = vec4(0.0, 0.0, 0.0, 1.0);

vec4 a = A * x; // a = (1,2,3,4)
vec4 b = A * y; // b = (5,6,7,8)
vec4 c = A * z; // c = (9,10,11,12)
vec4 d = A * w; // d = (13,14,15,16)
```

- There are also integer vectors (`ivec2`, `ivec3`, `ivec4`) and boolean vectors (`bvec2`, `bvec3`, `bvec4`).
- You can declare fixed-size arrays whose sizes are known at compile time:

```
float A[4];
A[0] = 5; A[3] = 10;

vec4 B[10];
B[3] = vec4(1,2,3,4);
B[8].y = 10.0;
```