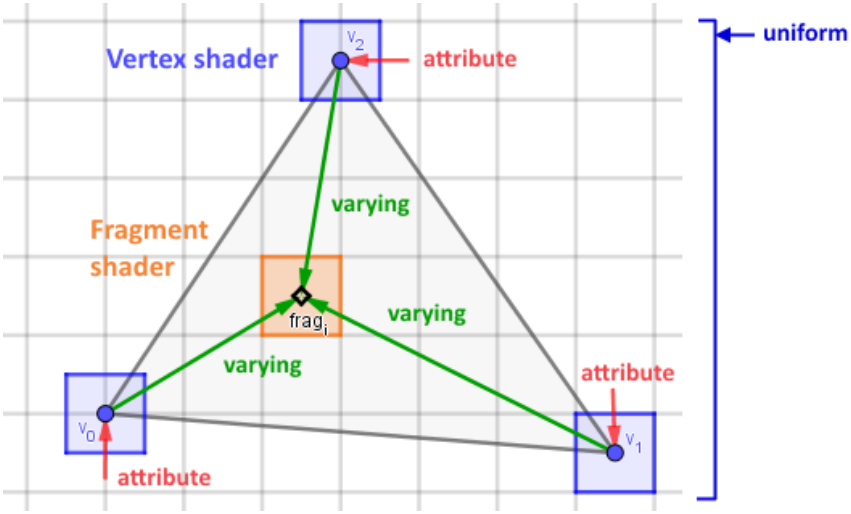
Computer Graphics – The Vertex and Fragment Shaders

Shadres run parallel on the vertices and fragments. The **attribute** variables in the vertex shader get their values from the CPU for each vertex. The **varying** variables get their value in the vertex shader and are **interpolated** to the fragment shader. The **uniform** variables have the same value everywhere and cannot be changed in shaders.



Main purpose of the **vertex shader** is to assign a correct position to the **gl_Position** built-in variable.

Main purpose of the **fragment shader** is to assign a correct color to the **gl_FragColor** built-in variable.

GLSL has a very convenient syntax for matrices, vectors, trigonometric functions etc.

Syntax	Meaning		Syntax	Meaning
vec3(1.0, 2.0, 3.0);	Creates a vector [1, 2, 3]		vec3 v = vec3(1.0, 2.0, 3.0); vec2 u = **v.xx**;	Vector *u* will be [1, 1]
vec3(1.0);	Creates a vector [1, 1, 1]		float d = **length**(v);	Variable *d* will have the length of *v*
vec3 v = vec3(0.0); vec4 u = vec4(**v**, 1.0);	Vector *u* will be [0, 0, 0, 1]		float x = **dot**(u, v);	Variable *x* will have the dot product of *u* and *v*.
vec3 v = vec3(1.0, 2.0, 3.0); float x = **v.x**;	Variable *x* will be 1		vec3 w = **cross**(u, v);	Vector *w* will be the cross product of *u* and *v*
vec3 v = vec3(1.0, 2.0, 3.0); vec2 u = **v.xy**;	Vector *u* will be [1, 2]		vec3 r = **reflect**(v, n)	Vector *r* will be the reflection vector of *v* off a surface with the normal *n*.

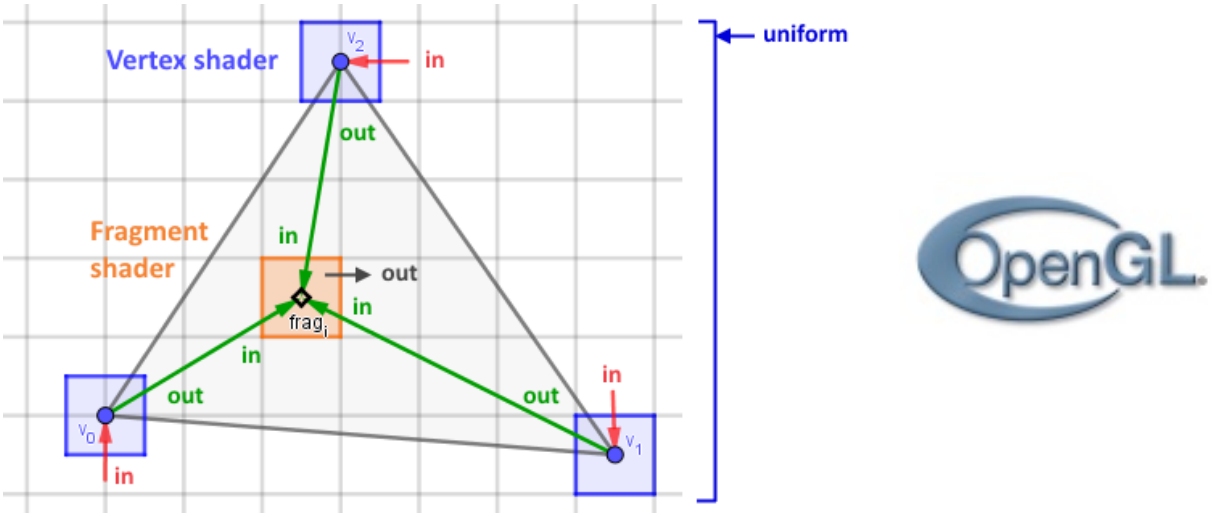
Google: "**WebGL API quick reference card**", it is an official reference card, pages 3 and 4 give a great overview of WebGL-s GLSL syntax.

The **uniform**, **attribute** and **varying** variables should be declared before the void main() function in the shaders. Three.js library already sets and sends a lot of **common variables automatically**. For example the **uniform matrices** projectionMatrix, modelViewMatrix. Here are some of them in the **vertex shader**:

Variable	Meaning		Variable	Meaning
uniform mat4 modelMatrix	Transforms coordinates from local space to world space.		uniform vec3 cameraPosition	Camera's coordinates in world space.
uniform mat4 viewMatrix	Transforms coordinates from world space to view space.		attribute vec3 position	Vertex coordinates in local space.
uniform mat4 modelViewMatrix	Transforms coordinates from local space to view space.		attribute vec3 normal	Normal vector of the vertex.
uniform mat4 projectionMatrix	Transforms coordinates from view space to clip space.		attribute vec2 uv	UV-coordinates of the vertex
uniform mat3 normalMatrix	Transforms normal vectors from local space to view space.			

Computer Graphics – The Vertex and Fragment Shaders

Shadres run parallel on the vertices and fragments. The **in** variables in the vertex shader get their values from the CPU for each vertex. The **out** variables get their value in the vertex shader and are **interpolated** to the **in** variables in fragment shader. The **uniform** variables have the same value everywhere and cannot be changed in shaders. The **uniform**, **in** and **out** variables should be declared before the void main() function in the shaders.



Main purpose of the **vertex shader** is to assign a correct position in the **gl_Position** built-in variable.

Main purpose of the **fragment shader** is to assign a correct color to its first and only **out** variable.

GLSL has a very convenient syntax for matrices, vectors, trigonometric functions etc.

Syntax	Meaning	Syntax	Meaning
vec3(1.0, 2.0, 3.0);	Creates a vector [1, 2, 3]	vec3 v = vec3(1.0, 2.0, 3.0); vec2 u = **v.xx**;	Vector *u* will be [1, 1]
vec3(1.0);	Creates a vector [1, 1, 1]	float d = **length**(v);	Variable *d* will have the length of *v*
vec3 v = vec3(0.0); vec4 u = vec4(**v**, 1.0);	Vector *u* will be [0, 0, 0, 1]	float x = **dot**(u, v);	Variable *x* will have the dot product of *u* and *v*.
vec3 v = vec3(1.0, 2.0, 3.0); float x = **v.x**;	Variable *x* will be 1	vec3 w = **cross**(u, v);	Vector *w* will be the cross product of *u* and *v*
vec3 v = vec3(1.0, 2.0, 3.0); vec2 u = **v.xy**;	Vector *u* will be [1, 2]	vec3 r = **reflect**(v, n)	Vector *r* will be the reflection vector of *v* off a surface with the normal *n*.

One 3D object in OpenGL is defined using a *vertex array object* (VAO). It is a collection of buffers called *vertex buffer objects* (VBO). Buffers define the data for the **in** variables of the vertices or the index (the faces).