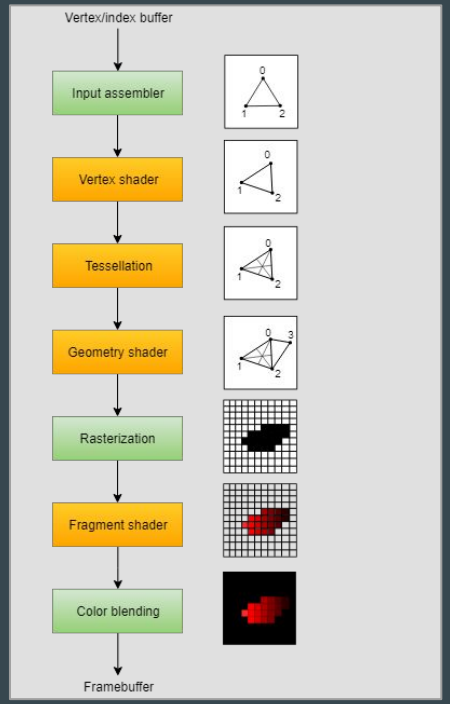
GL Shading Language

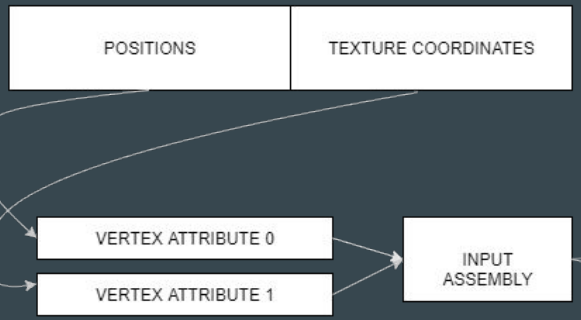
Carlos Fuentes

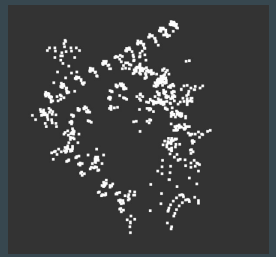
GPU pipeline



- **Input assembler** → collects vertex data (positions, normals, colors) from vertex and index buffer
- **Vertex shader** → Runs for every vertex. Generally applies transforms from model space to screen space
- **Tessellation shader** → allows to subdivide geometry to add quality
- **Geometry shader** → Runs for every primitive (triangle) and discards or adds new primitives.
- **Rasterization** → Discretizes each primitive into fragments/píxels .
- **Fragment shader** → For each survive fragment determines its color and depth.
- **Color blending** → mix color of different fragments that maps to the same pixel (various primitives that overlap) in frame buffer.

GPU pipeline: input assembler to vertex shader





VERTICES

VERTEX SHADER

```
layout(location = 0) in vec3 vertex_position;
layout(location = 1) in vec2 vertex_uv0;

uniform mat4 proj;
uniform mat4 view;
uniform mat4 model;

out vec2 uv0;

void main()
{
    position    = (model*vec4(vertex_position, 1.0)).xyz;
    gl_Position = proj*view*vec4(position, 1.0);
    uv0 = vertex_uv0;
}
```

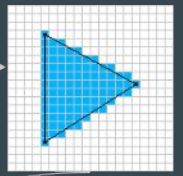
GPU Pipeline: vertex shader to fragment shader

TRIANGLES

POSITIONS

TEXTURE COORDIANTES

RASTERIZER



FRAGMENTS

INTERPOLATED TEXTURE COORDIANTES

VERTEX SHADER

```
layout(location = 0) in vec3 vertex_position;
layout(location = 1) in vec2 vertex_uv0;

uniform mat4 proj;
uniform mat4 view;
uniform mat4 model;

out vec2 uv0;

void main()
{
    position    = (model*vec4(vertex_position, 1.0)).xyz;
    gl_Position = proj*view*vec4(position, 1.0);
    uv0 = vertex_uv0;
}
```

FRAGMENT SHADER

```
uniform sampler2D texture0;

in vec2 uv0;
out vec4 color;
void main()
{
    color = texture(texture0, uv0);
}
```

GLSL reference

- GLSL is a **C like language** with some features from **C++**:
  - **Variables** can be declared everywhere
  - **Functions** can be overloaded
- The **preprocessor** and preprocessor directives can be used preceded by #:
  - #define, #ifdef, #error, #defined, etc..
- **Comments** are C++ /* */ and also //

GLSL reference : Data types

- List of **types**:
  - bool (true, false), int, float
  - vec2, vec3, vec4 (float)
  - bvec2, bvec3, bvec4 (boolean)
  - ivec2, ivec3, ivec4 (integer)
  - uvec2, uvec3, uvec4 (unsigned)
  - mat2, mat3, mat4
  - mat2x3, mat2x4, mat3x2, mat3x4, mat4x2, mat4x3
  - sampler1D, sampler2D, sampler3D, samplerCube, sampler1DShadow, sampler2DShadow, and more...

GLSL reference: Data types

- **Vectors usage**:
  - vec4 v ⇒ access by index v[0], v[1], v[2], v[3]
  - vec4 v => access to each euclidean coordinate v.x, v.y, v.z, v.w
  - vec4 v ⇒ access to each texture coordinate v.s, v.t, v.p, v.q
  - vec4 v ⇒ access to each color component v.r, v.g, v.b, v.a
  - Sizzling ⇒ vec4 v ⇒ v.xy is a new vec2 and v.xx also (different access types can´t be used)
- **Samplers usage:**
  - Samplers are linked to texture units
  - They can be only uniform or function parameters

GLSL reference: Data types

- **Structs** like C
- Fixed size **arrays**
- **Type qualifiers:**
  - **const** (function parameters or local variables)
  - **uniform** ⇒ linked between application and shader ⇒ The same for all draw call
  - **in** ⇒ link between a variable and previous stage (previous shader stage or vertex attribute)
  - **out** ⇒ link between a variable and next stage
  - out and in can also have the following qualifiers: **smooth** (perspective correction interpolation), **flat** (no interpolation), **noperspective** (linear interpolation).

GLSL reference: Functions

- **Functions**:
  - **Parameters** can be in, out, inout or const
  - Must have a **return type**
  - Array parameters are allowed but **array returns not**
  - **Structs** are allowed as parameter and also as return value
  - **main** ⇒ entry point of shader

GLSL reference: Flow control

- **Flow like C**:
  - if, if- else
  - while, for, do - while
  - continue, break
  - return
  - **discard**

GLSL reference: Built-in vertex shader variables

- Built-in vertex shader variables used for some operations that occurs in fixed functionality between vertex and fragment shader:
  - **int gl_VertexID;** ⇒ The index of current vertex
  - **int gl_InstanceID;** ⇒ The instance of the draw (when using instancing)
  - **out gl_PerVertex { vec4 gl_Position; float gl_PointSize; float gl_ClipDistance[]; }**
  - **gl_Position** must be written with the clipping coordinates of the vertex

GLSL reference: Built-in fragment shader variables

- in vec4 **gl_FragCoord**; ⇒ the fragment coordinates
- in float **gl_ClipDistance**[]; ⇒ linearly interpolated user clip distance
- in vec2 **gl_PointCoord**; ⇒ 2D coordinates within a point for point rendering
- in bool **gl_FrontFacing**; > is front face ?
- **in** int **gl_PrimitiveID**; ⇒ index of the primitive (triangle)
- **out** float **gl_FragDepth**; ⇒ for writing fragment depth (usually depth is auto-computed)

GLSL reference: Built-in constants

- Accessible from **all shaders**. Values depends on **implementation** :
  - const int gl_MaxVertexAttribs = 16;
  - const int gl_MaxVertexUniformComponents = 1024;
  - const int gl_MaxVertexOutputComponents = 64;
  - const int gl_MaxFragmentInputComponents = 128;
  - const int gl_MaxVertexTextureImageUnits = 16;
  - const int gl_MaxCombinedTextureImageUnits = 48;
  - const int gl_MaxTextureImageUnits = 16;
  - const int gl_MaxFragmentUniformComponents = 1024;
  - const int gl_MaxDrawBuffers = 8;
  - const int gl_MaxClipDistances = 8;

GLSL reference: Built-in functions

- **Trigonometry**: sin, cos, tan, asin, acos, atan, radians, degrees
- **Exponent**: pow, exp, log, exp2, log2, sqrt, inversesqrt
- **Math general purpose**: abs, sign, floor, ceil, clamp, mix, etc
- **Geometry**: length, distance, dot, cross, normalize, reflect, refract
- **Matrix and vector**: lessThan, greaterThan, equal, any, all, noct
- **Texture**: texture1D, texture2D, texture3D, textureCube, shadow, etc.
- **Fragment derivatives functions:** dFdx, dFdy, fwidth
- **Noise**: noise1, noise2, noise3, noise4 (1D, 2D, 3D, 4D).