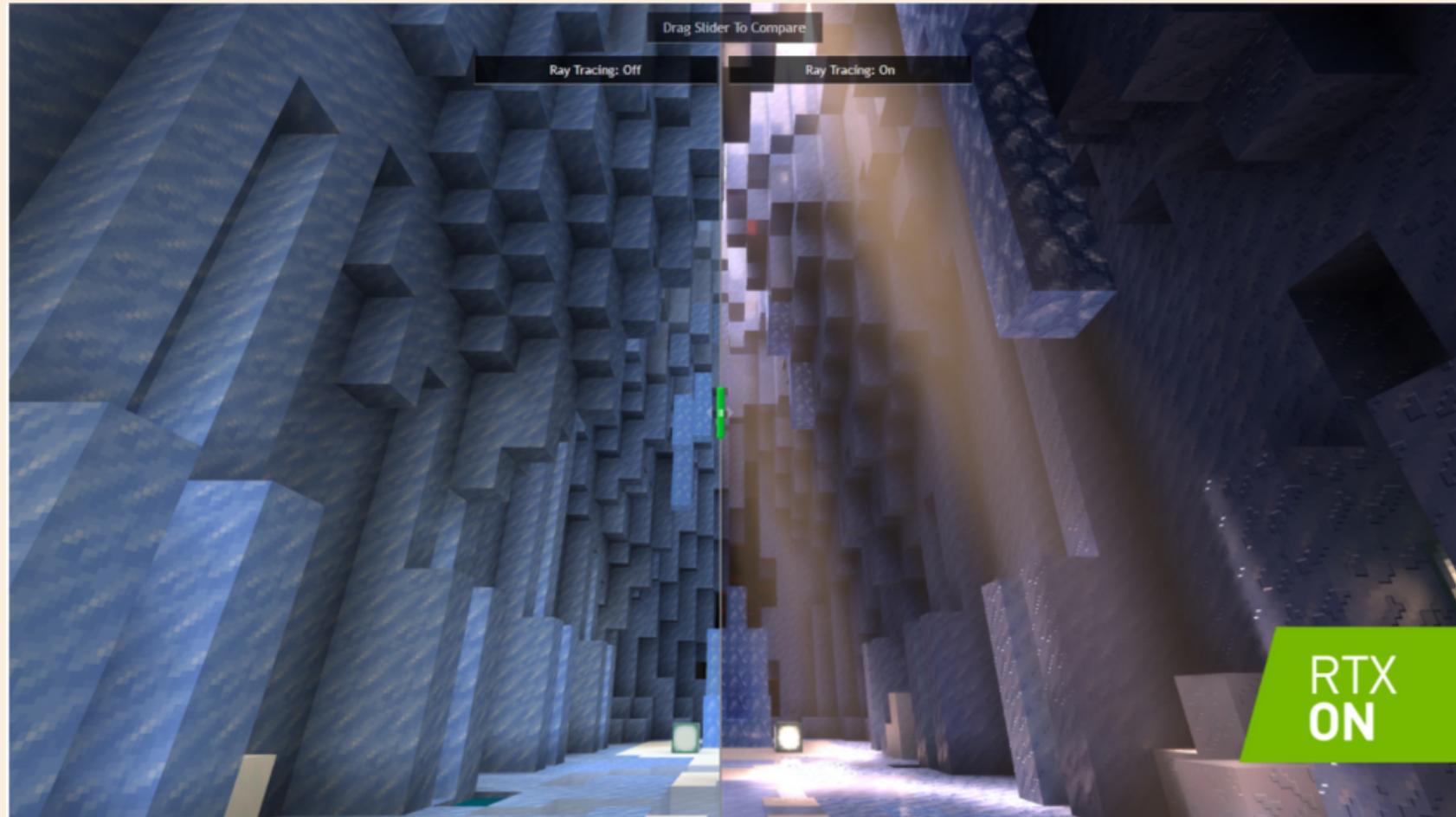


REAL TIME RENDER

Sergio Peñaloza

RASTERIZACIÓN

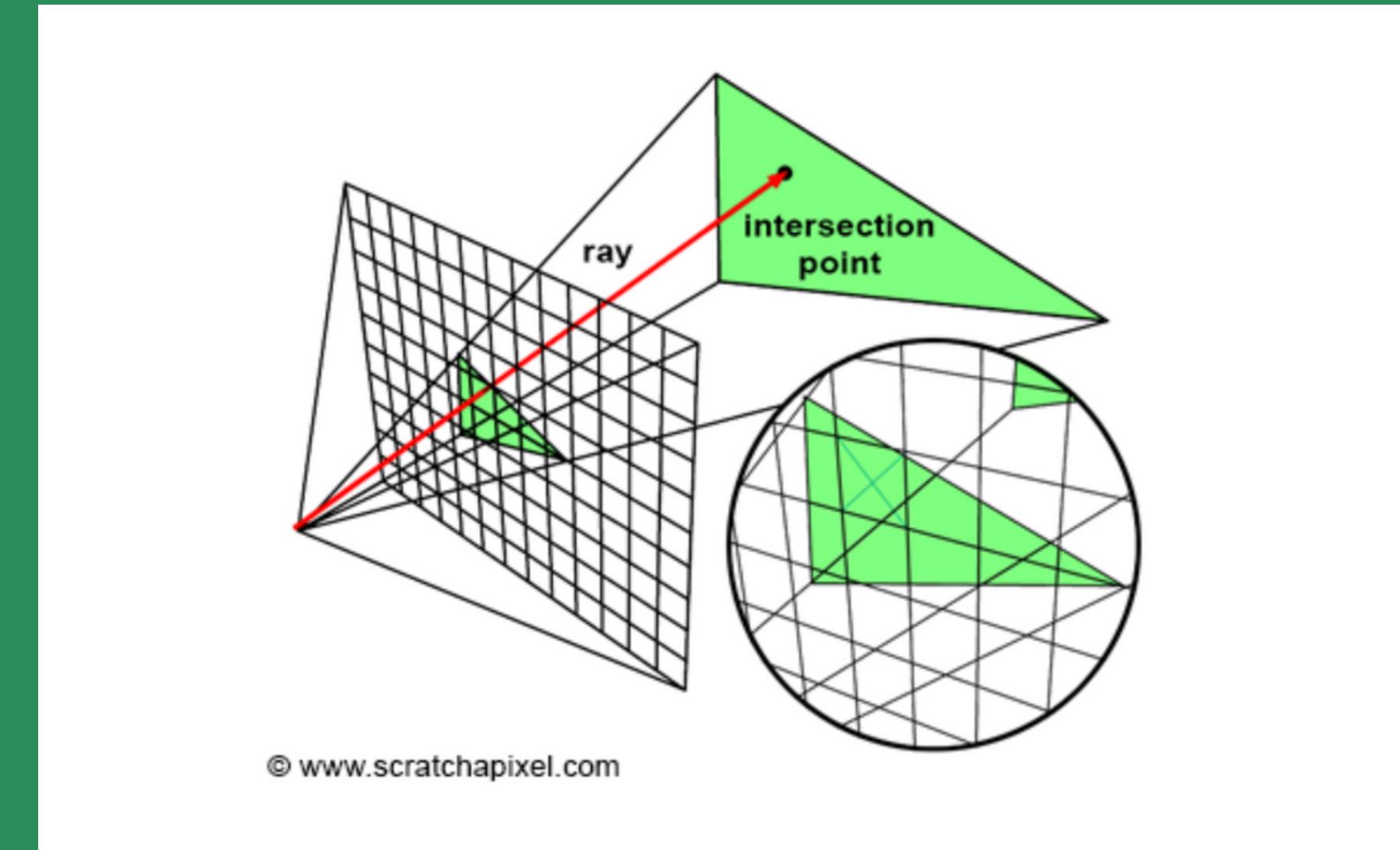




Para simular la luz que incide en una superficie de manera realista, el acercamiento de simular casi una infinidad de rayos de luz rebotando, perdiendo y transfiriendo energía ha resultado durante la mayoría de años de historia del render en tiempo real una tarea computacionalmente imposible.

Minecraft Rasterized VS RTX(nvidia)

Para lograr calcular una imagen hasta 60 veces en un segundo se ha venido utilizando la Rasterización que consiste principalmente en proyectar las formas poligonales de un espacio 3D visibles a través de un plano (pantalla) sobre el plano mismo.



MVP



Por lo general, la proyección que caracteriza a la rasterización está dada por la multiplicación de 3 matrices para cada vértice del modelo 3D:

Model

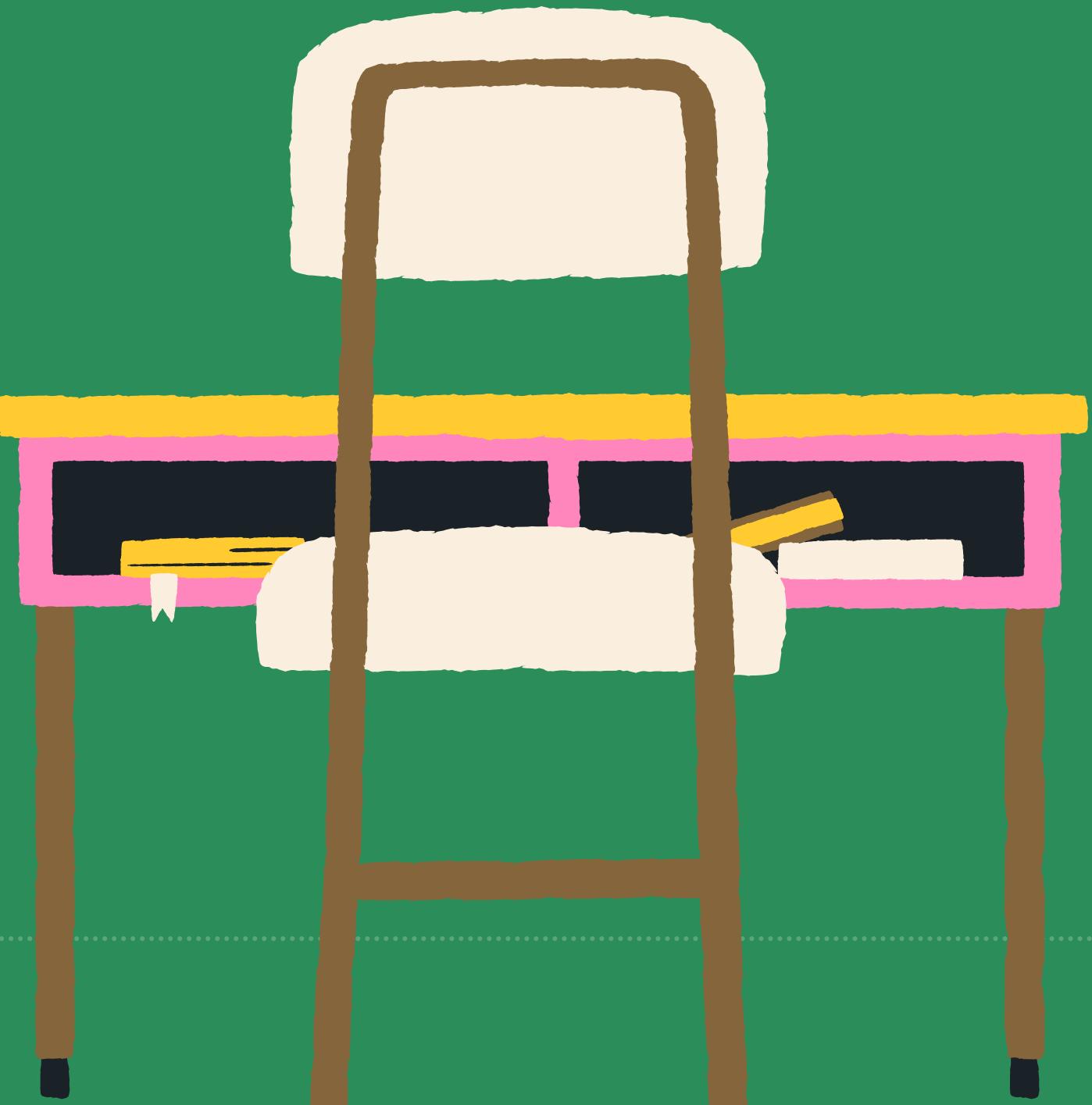
View

Projection

MODEL

Transforma coordenadas de vértices desde un espacio local (pivot del modelo) a un espacio global conocido como World Space

En sí misma es una matriz TRS (Translate, Rotate, Scale) que resuelve una posición en espacio global a partir de una posición en espacio local con punto de referencia de un pivot

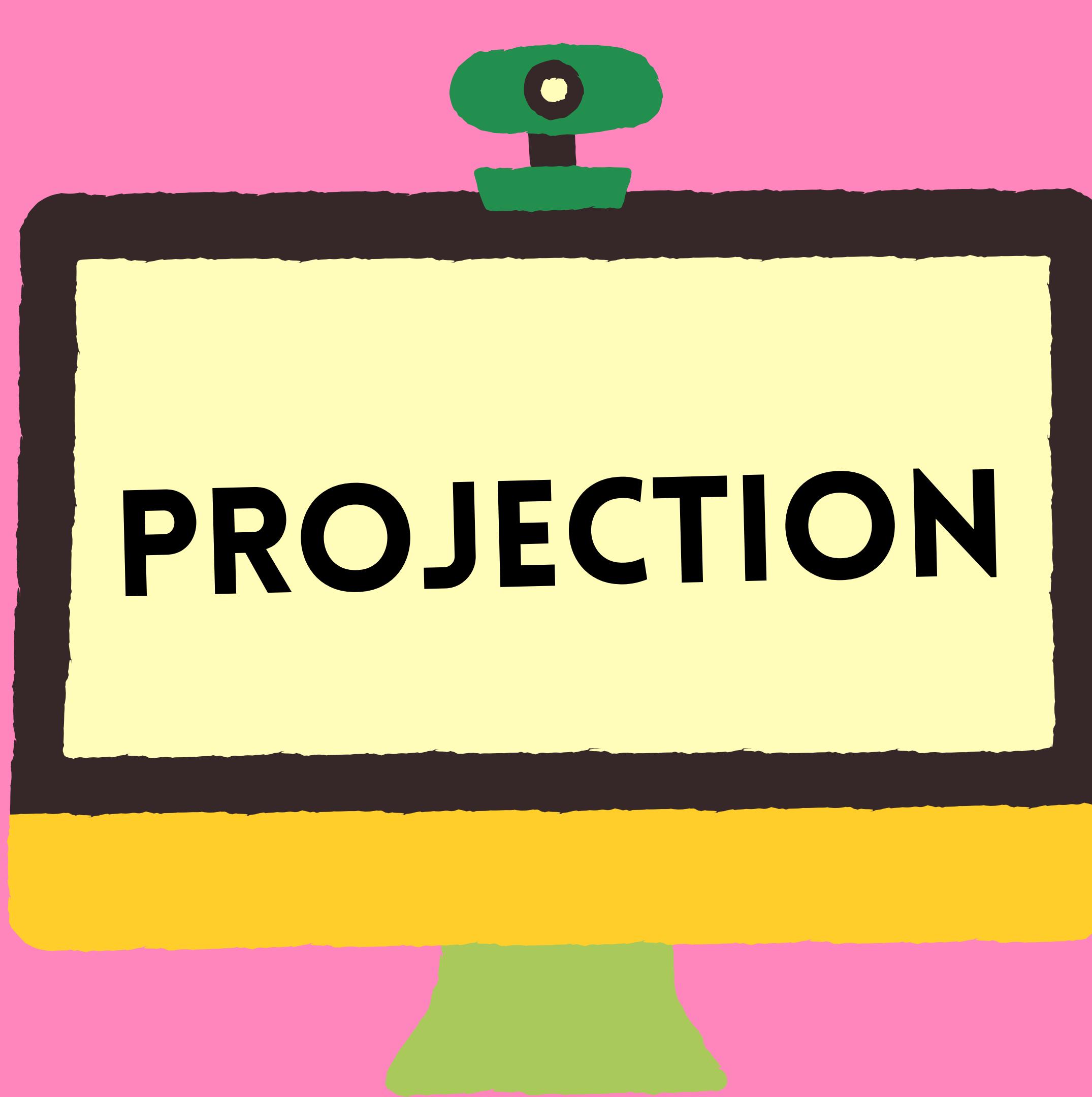


VIEW

Transforma coordenadas de un espacio global a un espacio local con punto de referencia de la posición y orientación de una cámara conocido como View Space o Eye Space.

Así como la matriz M, es una matriz TRS donde la matriz de escala siempre es I





PROJECTION

Dependiendo del tipo de proyección de la cámara (ortografico o perspectiva), transforma las coordenadas de View Space a un espacio local cartesiano unitario donde la unidad puede ser 1 (Normalized Device Coordinates Space) o W (Homogeneous Clip Space)

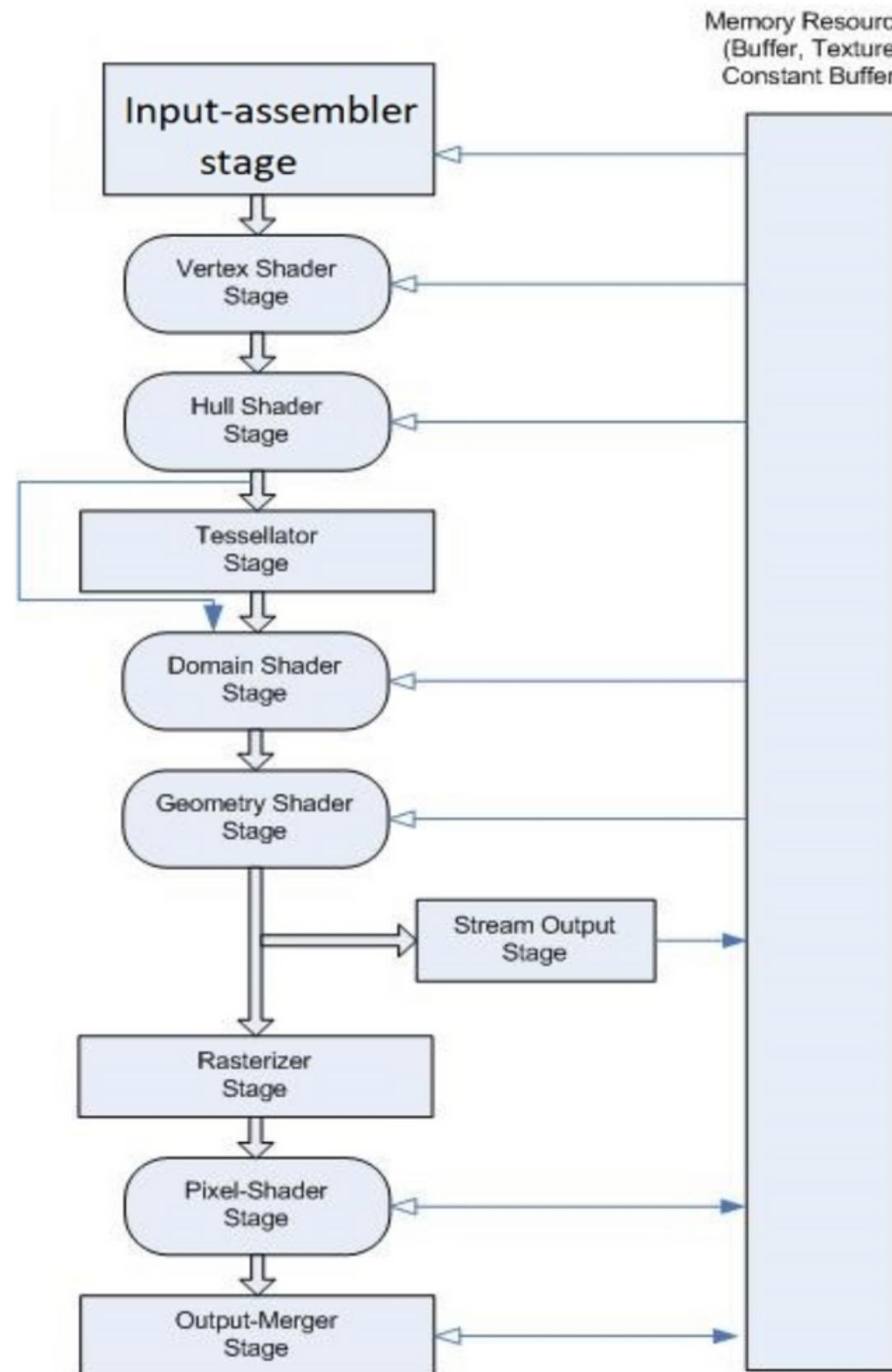
SHADERS

Para poder describir distintas cualidades en las superficies se utilizan Shaders, pequeños programas ejecutados en la gpu que describen las propiedades tanto geométricas como lumínicas de una superficie. Dependiendo de la etapa de render en la que se ejecute el shader, este se evaluará para cada Vértice, Primitivo(Triangulos o Quads) o Pixel del modelo 3D que se esté procesando en un momento dado. Para que un objeto pueda ser renderizado se necesita por lo menos un **Vertex Shader** (Cada vertice) y un **Fragment Shader** (Cada pixel o sub-pixel).

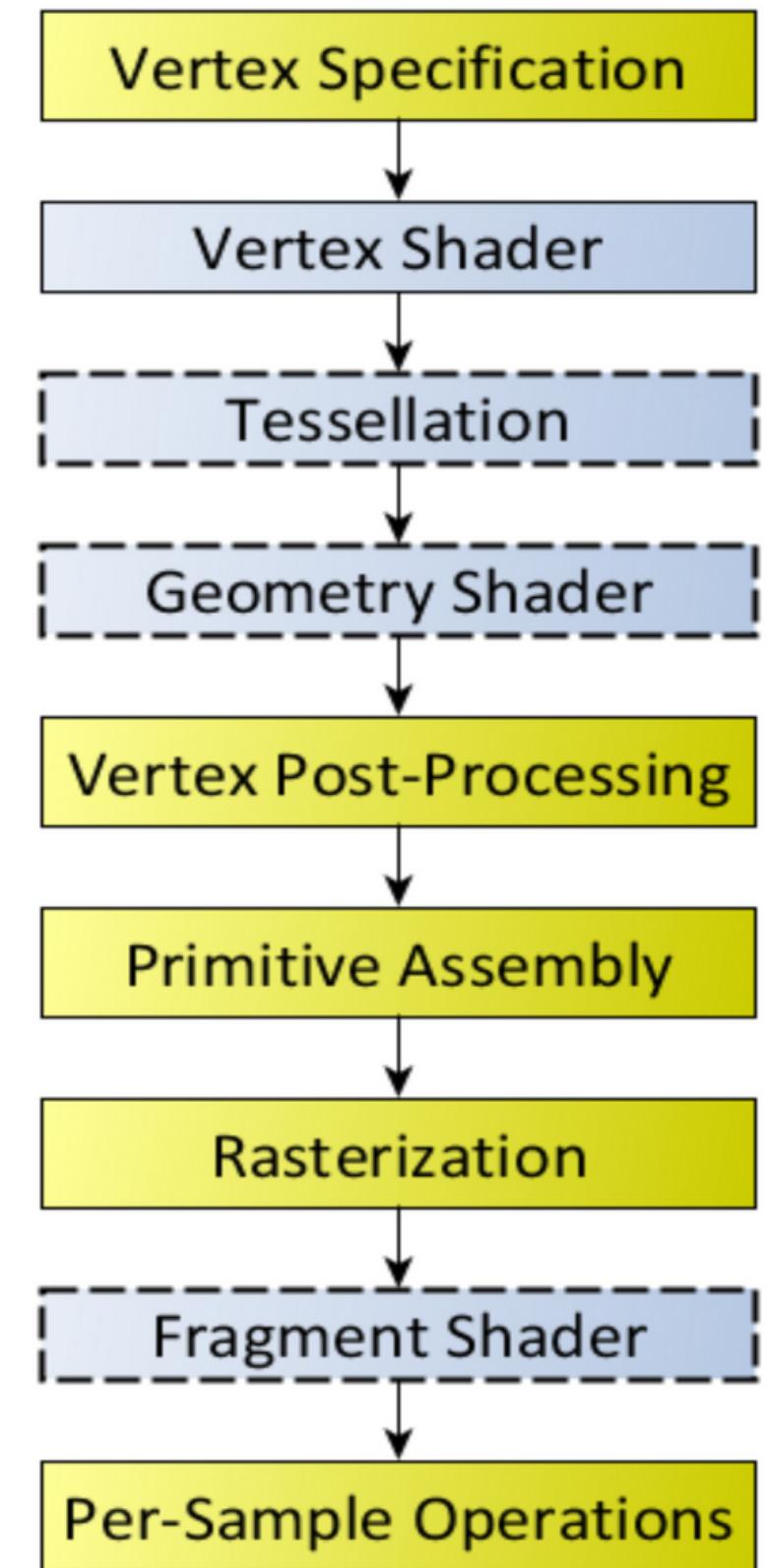


SHADING LANGUAGES

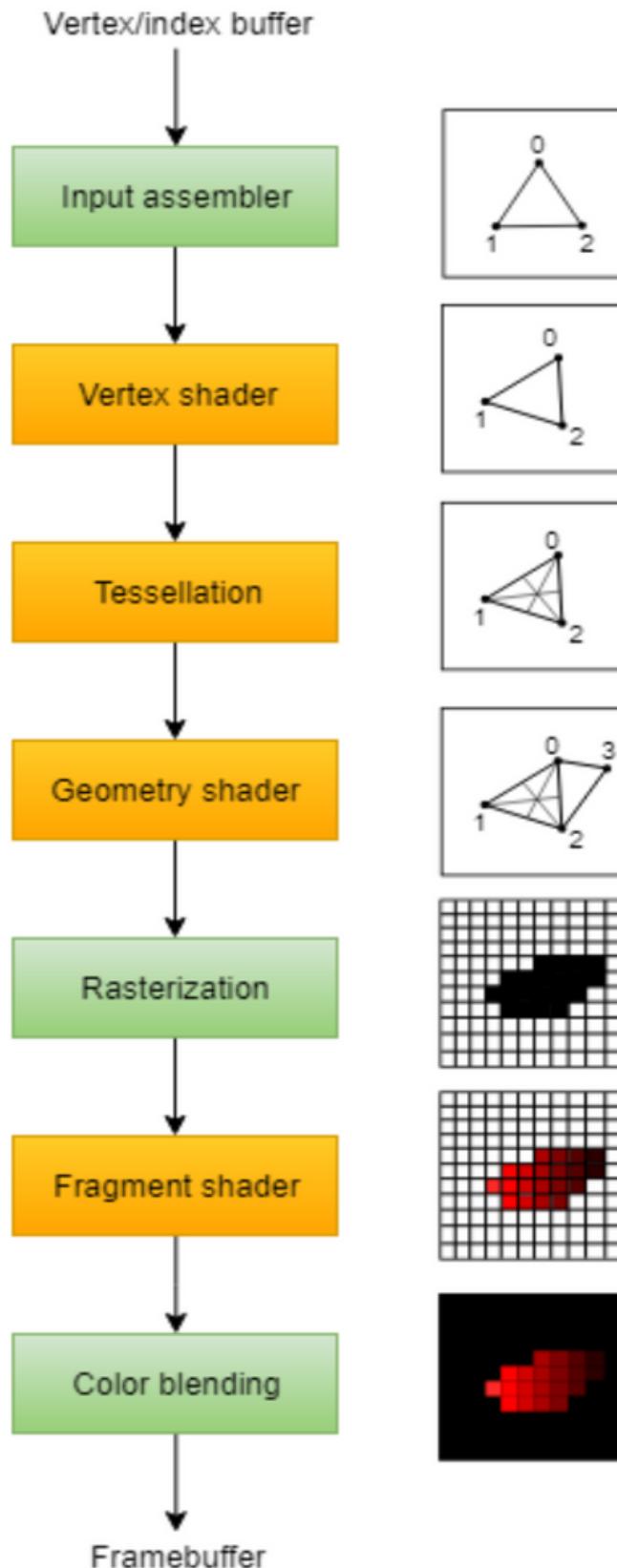
Los Shaders se programan en un Lenguaje de sombreadores o “Shading Languages”, dependiendo de la Api de renderizado que se utilice, el lenguaje cambia, pero la mayoría son muy similares entre sí



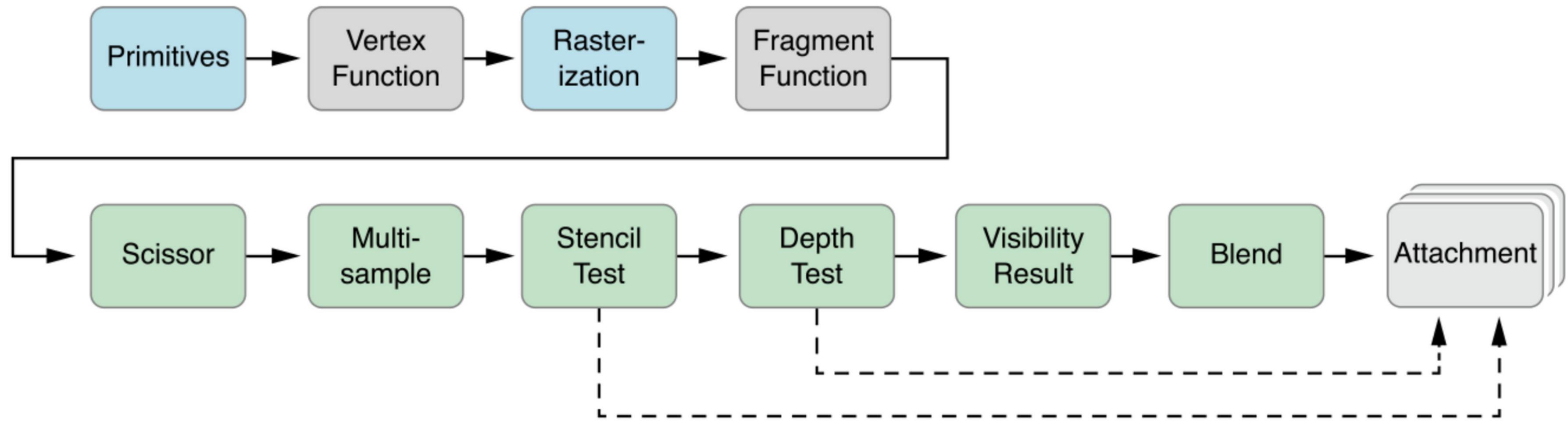
DIRECTX (MICROSOFT)



OPENGL(KHRONOS)



VULKAN(KHRONOS)



METAL(APPLE)

SHADERLAB

Unity implementa un lenguaje de programación intermedio denominado Shaderlab el cual se compone tanto de comandos exclusivos del sistema de render de unity como de código valido de shading languages

```
Shader "Examples/ShaderSyntax"
{
    CustomEditor = "ExampleCustomEditor"

    Properties
    {
        // Material property declarations go here
    }
    SubShader
    {
        // The code that defines the rest of the SubShader goes here

        Pass
        {
            // The code that defines the Pass goes here
        }
    }

    Fallback "ExampleFallbackShader"
}
```