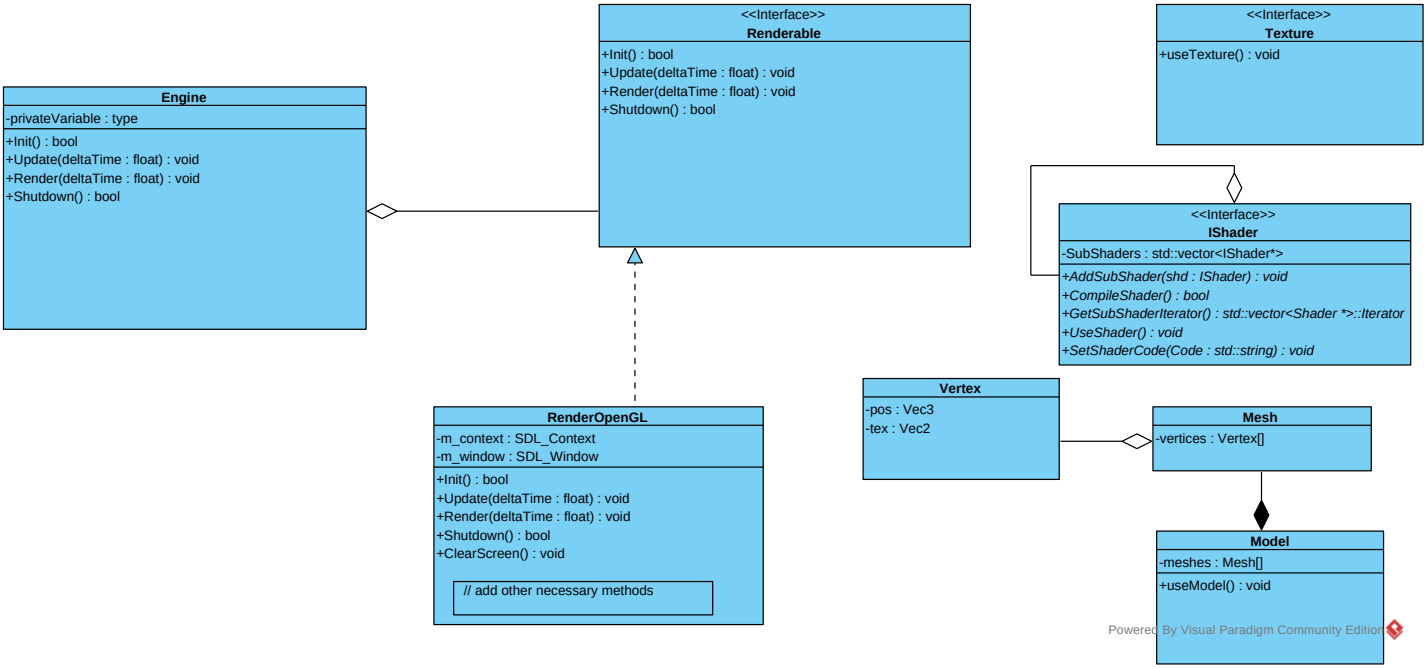
Engine
-privateVariable : type
+Init() : bool
+Update(deltaTime : float) : void
+Render(deltaTime : float) : void
+Shutdown() : bool
<<Interface>>
Renderable
+Init() : bool
+Update(deltaTime : float) : void
+Render(deltaTime : float) : void
+Shutdown() : bool
<<Interface>>
Texture
+useTexture() : void
<<Interface>>
IShader
-SubShaders : std::vector<IShader*>
+AddSubShader(shd : IShader) : void
+CompileShader() : bool
+GetSubShaderIterator() : std::vector<Shader *>::Iterator
+UseShader() : void
+SetShaderCode(Code : std::string) : void
RenderOpenGL
-m_context : SDL_Context
-m_window : SDL_Window
+Init() : bool
+Update(deltaTime : float) : void
+Render(deltaTime : float) : void
+Shutdown() : bool
+ClearScreen() : void
// add other necessary methods
Vertex
-pos : Vec3
-tex : Vec2
Mesh
-vertices : Vertex[]
Model
-meshes : Mesh[]
+useModel() : void
Powered By Visual Paradigm Community Edition

Renderable Protocol

This section is the renderable protocol class diagram that holds the interface for all the other classes to implement. The Interface class IRenderable will be a pointer by the Engine. The RenderOpenGL is a class that can implement IRenderable but can also be used by another graphics API like DirectX. The other interface classes are for each member of the protocol (Texture, Shader, Vertex and Mesh) all of those are used on the Model class to be rendered therefore.

Shader tree

We decided to use a shader tree to achieve a rich dynamic shader system. Essentially the root shader would have the main GLSL function and inside that function, forward declare several common GLSL functions like GetPos() GetColor(). In your leaf node include a GLSL shader that define the previously forward declared shader. The process can end here but one can extend this ShaderA uses ShaderB and ShaderC and ShaderB uses ShaderD and shaderE and etc