CEIG - Spanish Computer Graphics Conference (2022)
J. Posada and A. Serrano (Editors)

Development of a Node-Based Material Editor

L. Suaya Leiro[1] and M. Garrigó[1]

[1]Centre de la Imatge i la Tecnologia Multimèdia - Universitat Politècnica de Catalunya

Abstract

Materials systems are an important element within the development of a renderer for an application such as a video game. Nowadays, the method to build a graphic style for a product involving a real-time engine implies a rendering system supporting a solid and concise materials system, as those well-established in real-time engines such as Unreal or Unity.
This study presents an open-source application to serve as an editor of materials consisting of a modern real-time renderer. The application consists of a basic OpenGL real-time rendering engine to visualise 3D geometry and its appearance through the support of a node-based material editor to assemble materials in an intuitive and simple manner, without the use of programming and little technical knowledge.
The culmination of the project and the achievement of its objectives was satisfactory. We concluded that this work can be used as a reference to understand real-time material systems and renderers and its state of the art in the video games industry.

CCS Concepts

• ***Computing methodologies*** → ***Rendering;*** • ***Computer systems organization*** → ***Real-time system architecture;*** • ***Human-centered computing*** → *Accessibility; Visualization; Human computer interaction (HCI);*

1. Introduction, Motivation and Justification

Nowadays, the method to build a graphic style for a video game in three dimensions or a similar product involving a real-time engine implies a rendering system supporting a solid materials system. The real-time result of the materials created should follow an artistic style marked by the artistic direction. Therefore, an artist must be able to create materials that replicate a desired style without involving programming or technical knowledge.

The problem arises in the difficulty of the process. In essence, a material is a series of coded variables that define how an object should be rendered, so – in the background – a material can be programmed and it is processed by a renderer to end up in the GPU. In this process, there are certain considerations such as optimizations and many others, but in general, this causes an obstacle in the creation of materials: it makes the process complex, it involves deep knowledge in the graphics programming field and it requires programmer to be in constant communication with artists. Besides, the existence of little deep documentation and the fact that there is no specific consensus on how to build a renderer with a material system makes it more difficult, because it depends on the type of engine operated or developed, so there are different methods to solve this problem. For these reasons, this project proposes to build and delineate from the beginning a renderer and a system of materials that is visual and easy to use, with the aim that anyone in the industry can understand how it works.

The motivation behind this project was our passion for videogames and graphics programming, as well as the curiosity to explore and expand our knowledge limits. We aimed to develop a tool to create and understand easily the operation and functioning of materials in a 3D environment, what implications they have, and their usage in real-time frameworks like videogames. At the time of beginning this work, we were aware of the existence of widely known and used software that already solved the problem that we addressed, but our intention was not to expand these, make them better, or search for new or better key findings in the area. Instead, we wanted our contribution to consist of an open-source and instructive project to investigate how they can be done and try to mimic them as well as possible regardless of the evident quality, time, and budget divergence, but also without any additions or features externals to this specific area to keep it as simple and easy as possible.

2. State of the Art

There are several programs and tools in the market addressing this problem, from applications focused on visual effects, animation, or geometry creation such as *3DS Max*, *Blender* or *Houdini* [Autne, Blene, Sidne] to real-time engines like *Unreal*, *Unity* or *Godot* [Epinec, Unined, Linne]. Traditionally, the method to assemble a material in this style of application was to program a shader and expose certain parameters that could define a material's characteristics. Then, through a material's user interface, the user could modify its attributes and assign each material to the objects

DOI: 10.2312/ceig.20221148

in a scene [Uninea]. Any further customization had to be made by the user by programming new shaders and following the same process [Unineb]. However, in the last decades, many of the mentioned applications implemented node-based editors in which the material became a node instead of a plain and limited user interface [Epinea, Epineb, Frene, Uninec]. In this node we could input other nodes, thus giving the ability to make each of the material's characteristics be the result of a set of operations represented by the connections of nodes in the editor, forming a graph. This abstraction removed the potential need for programming when crafting custom materials.

Additionally, there are applications with powerful renderers specifically designed to artistically craft materials or to modify geometry according to the material's traits. Some examples are *Substance Designer* and *Substance Painter* or *Marmoset* [Adoneb, Adonea, Marne], which provide tools thought to optimize the creation process, such as painting materials to texture and shape them.

3. Our Approach

Our approach consisted of building a real-time game engine named *Kaimos* focused on a renderer with a tool embedded for materials assembly. It was programmed with C++ and we used OpenGL and GLSL as a graphics API and shading language respectively.

We referenced this approach on the popular game engine *Unreal Engine 4*, bridging the differences. The objective was to develop the prototype of a real-time engine running at a minimum of 30fps that allowed to load 3D geometry and visualize it in a scene editor. These geometry would use materials to shape appearance, and these materials would be created with a node-based editor. Finally but not less important, the implementation of a lighting system was also required, as it is an important pillar of appearance.

3.1. System Architecture

First, we developed base systems for the engine (elemental life-support, cameras, input, scene, entities-components, serialization, files management, time, mathematics...). We required the usage of external libraries which, simultaneously with the mentioned systems, formed a first engine prototype with which to work.

Afterwards, we built a basic batched renderer to draw simple shapes on the screen. Jointly with it, we defined how *Kaimos* compiled shaders and managed geometry, with which we could draw more complex meshes with a few draw calls. Then we went further and implemented a materials structure and the possibility for each piece of geometry in the scene to have a material attached. The material would transmit its properties to a shader (its surface appearance data) and therefore the shader would know how to render each mesh according to its surface properties. This gave a different appearance to each mesh in the scene with a different material.

With the renderer pillars in place, we implemented mesh files loading, a lighting system based on the *Blinn-Phong* [dVneb, dVnea, Bli77] model with different types of light sources. By means of this, we could improve the renderer a little by adding some techniques like normal or specular mapping [dVned, dVnee].

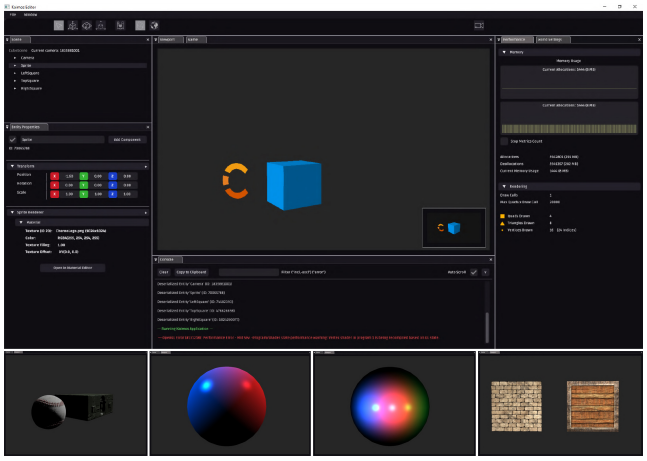

Figure 1: *Images of a primitive Kaimos Editor. On top, a view of the editor with some of its panels (tools, scene hierarchy, entity editor, console and performance), at the bottom, examples of models' loading, basic lighting, normal and specular mapping.*

3.2. Modern Rendering

To modernize the renderer, we expanded it with the techniques of *Physically Based Rendering* (PBR) and *Image-Based Lighting* (IBL) as they significantly increase the visual quality. Previously, we used *Blinn-Phong* [dVneb, dVnea, Bli77] to illuminate and render objects.

PBR is a rendering technique in which the lighting and materials are based on a physical theory that matches closely the real world ones [dVneg] to simulate them in a physically-plausible or realistic approach. The materials and lighting rely on physical parameters that make them look realistically correct. It takes into consideration three concepts:

- **The microfacet surface theory:** describes surfaces as formed by tiny and reflective microfacets whose alignment defines the material's reaction to light.
- **The energy conservation law:** makes the light leaving the material not to be higher than the light entering.
- The employment of **physically-based reflections** based on a physically-realistic equation named the *Reflectance Equation*.

The *Reflectance Equation* combines these concepts. It is a specialized version of the *Render Equation* [dVneg, Kaj86, ICG86] used in computer graphics to simulate physically-plausible lighting:

$$L_0(p,\omega_0) = \int_\Omega f_r(p,\omega_i,\omega_0)\ L_i(p,\omega_i)\ n\cdot\omega_i\ \mathrm{d}\omega_i$$

This equation is solved for each light source in the scene. We will omit the mathematical details for synthesis and readability purposes, as it is a widely known technique, but it is notable the f_r term, known as the *Bidirectional Reflectance Distribution Function* (BRDF). It calculates the light reflection considering the

material characteristics, its microfacets, the metalness or roughness of the surface, etc. There are many models to define it, but we based it on the *Unreal Engine 4* implementation, published in 2013 [KEne, dVneg, dVnef]. It uses the *Cook-Torrance BRDF* [Karne, CT82], which has a diffuse and a specular term representing respectively the light reflected after its refraction — or scattered among the material's surface particles — and the light directly reflected (that defines the specular lobe of the light impacting on the surface).

To conclude the PBR implementation, we also modified the light attenuation of *Blinn-Phong* to use a more physically accurate version, the inverse-square law [KEne]. The result can be seen in Figure 2.

IBL illuminates the objects in a scene in a physically-plausible manner too, but with the light of the environment. Our implementation was also based on *Unreal Engine 4* [KEne, dVnec, dVneh]. It uses an environment map (a cubemap) as the background environment from which we extract the source of light. With the texture texels, we obtain a global illumination to apply it to the scene objects by resolving the same *Reflectance Equation* used for PBR. Since the IBL light sources are too large to compute them in real-time (potentially every texel), we pre-process the calculations to use them in real-time. The idea consists of producing two maps from the original environment map by convolution. In these, we respectively store the resulting general diffuse and specular irradiance. Then they are accessed in real-time to sample and calculate the ambient irradiance to make it part of the illumination process. The diffuse map (or *Irradiance Map*) is obtained by applying the diffuse term of the *Reflectance Equation* to a sample of texels of the original map. The specular map (or *Pre-filtered Environment Map*) is obtained by applying the specular term of the equation to a sample of directions from the original map, biased to be within the specular reflection lobe. On each mipmap of it, a result for a different roughness value is stored. We also store the possible BRDF values in a LUT 2D texture (as it varies), used in conjuntion with the other maps in real-time to sample the total BRDF of the environment according to the surface lightened.

There are some quality compromises comparing the pre-computation with the real-time calculation, some reflections disappear or may not be the most realistic. However, the real-time processing of IBL is not possible due to the potential number of light sources and the fact of performing a heavy calculation like an integral for each of them. This makes the result an approximation, but it is realistic enough, as it can be seen in Figure 2.

3.3. Kaimos Material Editor

A node editor works based on the connections of nodes that have a specific individual purpose. Programming is not involved and, in this case, there is no static edition of the material variables in the interface. We built a structured inheritance of classes for the different nodes, where each pin of a node stores its atomic data, whilst the node knows what to do with that data. Upon compilation, the calculation of the main node parameters begins on each of its input pins, checking the outputs connected. These make their nodes to process the input data, and this process continues until reaching the final nodes without inputs connected. The result is kept in the

Figure 2: *Rendering outcome in Kaimos Engine.*

material which is attached to meshes in the scene, but the materials dependent on variable vertex attributes need to be re-calculated for each mesh. In summary, the material editor structure has four parts:

- *Graph:* Connected group of nodes, represent the whole material appearance. Operates at a higher level, managing node usage (creation, deletion, linkage...).
- *Nodes:* Graph atomic structure with specific functionality. Represents a defined data type that can result from a constant, an operation...
- *Main Node:* Node that represents the material properties with its inputs and has no output. All graph connections end here.
- *Pins:* Connection points of a node, they can be connected to a compatible data-type pin and holds the atomic data of the node. The outputs represent the node result and the inputs the data needed to compute it.

The *Kaimos Material Editor* is aware of whether a material is PBR or non-PBR to decide which parameters displays in the main node. Both have vertex attributes and color as well as albedo and normal textures to shape appearance. PBR materials require textures for roughness, metallic and ambient occlusion parameters while non-PBR requires a specular map. The textures have a 0-1 numeric value to control their impact/influence.

Regarding the interface, there is a panel in the *Kaimos Editor* in which to operate the graph. We used the *ImNodes* library to have nodes' UI functionalities. Designed from the point of view of the user, the ease of its usage had to be granted (the user focus must be on the important tasks of the material edition) and it had to be as graphic as possible. This was given by the nature of the editor itself, nodes converging through connections into the main node. Thus, we shaped the usage to be as fast and as simple so all the actions were at a maximum distance of two clicks. We also wanted the nodes-selector pop-up to display the nodes' categorization while keeping the fast and easy characteristic, so it is a simply structured menu easy to read. We also decided not to add static menus to preserve simplicity, unity, and dynamism. Due to how the library works and renders, we could not implement zooming.

The nodes' categorization reflect the operations and methods:

- *Vertex Attributes Nodes:* Geometry vertex attributes
- *Constants Nodes:* Global or external constants (time, pi...)
- *Camera Nodes:* Related to the scene camera
- *Variables Nodes:* Plain data type variables (int, float, vec2...)
- *Mathematic Nodes:* Mathematical operations (random numbers, simple and advanced operations, conversions, trigonometric and shader operations...).
- *Vectors Nodes* Related to vector operations (break into components, normalize, rotate, reflect, lerp...).

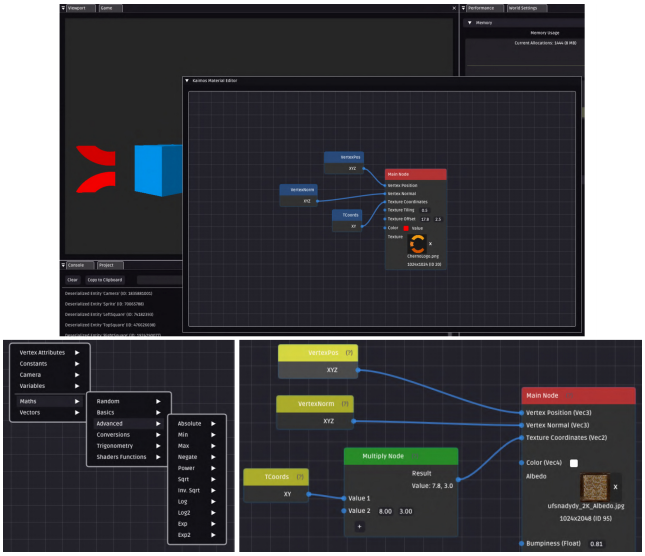

Figure 3: *Images of Kaimos Material Editor (KME). On top, an overview with an example and in the bottom details of the nodes.*

To conclude, it is noteworthy that we considered an alternative to the system consisting of parsing the entire graph into a shader, as if it was a translator from the graph to code. It was a more attractive approach but highly complicated with the available resources, and it would imply significant and complex changes, especially with the API-abstracted nature of *Kaimos* and the shaders' compilation process. As it went much further from our knowledge barrier and from the scope of the project, it was discarded.

3.4. Results

We built three scenes with different complexity setups to test the renderer performance and the material editor. Two consisted of seventy-two spheres (a total of 12500 triangles, approximately), each with a relatively simple material, and thus a simple node graph. One scene was rendered with PBR materials and an IBL environment, the other without PBR and a plain skybox environment without IBL. The third scene consisted of a 3D plane simulating water movement through its material (of 800 triangles to make the animation softer), which required a far more complex PBR material with a complex arrangement of nodes, and with an IBL environment too.
We measured them with *Kaimos Engine* built-in tools executing in a computer with an *AMD Ryzen 5 3500X* CPU, 32GB of RAM and an *Nvidia GeForce RTX 2060* GPU. Furthermore, we also used a profiler to measure the performance of the application and have an independent measurer in which we could rely and compare to be sure of our own measurements. The three scenes maintained a stable framerate of 60fps with Vertical Synchronization enabled and a rendering time taking from 1ms to 5ms. It was beyond expected for the project scope considering that our objective was to keep a minimum of 30fps, and we initially expected the rendering to take more time (we did not consider a reasonable limit, but the measure was better than we ambitioned). Still, as we did not implement any rendering optimization, it is a potential improvement point.

The memory usage was below 2GB. This also considered the *Visual Studio* memory usage and the memory measurer we implemented in *Kaimos* was not considered to be trusted due to the issues with its implementation and its erratic behaviour. Nevertheless, although we cannot determine the proper memory usage of the application and the scenes with all the certainty that we would like, we do know that all the resources were loaded in memory (heavy HDR maps, models, ...) as we did not have resources management or optimizations. Therefore, we can affirm that this is also an improvement point that could potentially improve this measure.

In conclusion, the performance was better than expected, and we did not have any relevant or crucial issues in this field. Some improvements can be developed such as decreasing the GPU queries and textures load, geometry, rendering, and resources optimizations (such as space-partitioning, camera culling...) amongst others.

4. Conclusions and Future Works

We presented a real-time engine with a scene editor centered in a 3D geometry renderer with modern and advanced real-time rendering features. It features a node-based material editor that allows assembling materials without programming or having complex technical knowledge as traditionally, so we can consider it a positive result. It performs better than expected, and we developed it using modern C++, using techniques that were unknown to us before. We satisfactorily accomplished the project objectives; additionally, we have learned beyond the domains and disciplines in which we wanted or expected to learn.

In future works, besides the optimizations considered – which should be a priority – we would recommend to address compatibility with other software. In other words, to implement the usage of the materials in *Kaimos* in another software, for instance by means of exportation, in order to synchronize the engine with widely-used software like *Unreal Engine* or *Unity* and observe the result in other renderers. Additionally, although we can consider a good improvement the creation materials through a node-based editor without requiring any programming, it would be critical to execute a user study and test, by asking artists to use *Kaimos*, to evaluate the performance and user experience.

We also recommend the implementation of rendering improvements like transparencies, IBL probes, and techniques like displacement maps, bloom, anti-aliasing, or shadowing. Another interesting investigation would be renderering systems like *Forward+*, *Deferred*, or the one used by *Unreal Engine 5*, as well as the implementation of better graphic APIs like *Vulkan* or *DirectX*.

References

[Adonea] ADOBE: Adobe Substance 3D Designer, 3D Assets Creation. Limitless. Adobe Substance 3D Designer Frontpage, Website, 2022 [Online]. URL: https://www.adobe.com/products/substance3d-designer.html. 2

[Adoneb] ADOBE: Adobe Substance 3D Painter, Paint in 3D. In real time. Adobe Substance 3D Painter Frontpage, Website, 2022 [Online]. URL: https://www.adobe.com/products/substance3d-painter.html. 2

[AMHH18] AKENINE-MÖLLER T., HAINES E., HOFFMAN N.: *Real-Time Rendering, Fourth Edition*, 4th ed. A. K. Peters, Ltd., USA, 2018.

[Autne] AUTODESK: 3ds Max: Massive Worlds and High-Quality Designs. 3DS Max Frontpage, Website, 2022 [Online]. URL: https://www.autodesk.eu/products/3ds-max/overview?term=1-YEAR&tab=subscription. 1

[Blene] BLENDER FOUNDATION: Blender, Reach new lights. Blender Frontpage, Website, 2022 [Online]. URL: https://www.blender.org/. 1

[Bli77] BLINN J. F.: Models of light reflection forccomputer synthesized pictures. *SIGGRAPH Computer Graphics 11*, 2 (Jul 1977), 192–198. URL: https://doi.org/10.1145/965141.563893, doi:10.1145/965141.563893. 2

[CT82] COOK R. L., TORRANCE K. E.: A Reflectance Model for Computer Graphics. *ACM Trans. Graph. 1*, 1 (Jan 1982), 7–24. URL: https://doi.org/10.1145/357290.357293, doi:10.1145/357290.357293. 3

[dVnea] DE VRIES J.: Advanced Lighting. Learn OpenGL, Blog, 2014 [Online]. URL: https://learnopengl.com/Advanced-Lighting/Advanced-Lighting. 2

[dVneb] DE VRIES J.: Basic Lighting. Learn OpenGL, Blog, 2014 [Online]. URL: https://learnopengl.com/Lighting/Basic-Lighting. 2

[dVnec] DE VRIES J.: IBL Diffuse Irradiance. Learn OpenGL, Blog, 2014 [Online]. URL: https://learnopengl.com/PBR/IBL/Diffuse-irradiance. 3

[dVned] DE VRIES J.: Lighting Maps. Learn OpenGL, Blog, 2014 [Online]. URL: https://learnopengl.com/Lighting/Lighting-maps. 2

[dVnee] DE VRIES J.: Normal Mapping. Learn OpenGL, Blog, 2014 [Online]. URL: https://learnopengl.com/Advanced-Lighting/Normal-Mapping. 2

[dVnef] DE VRIES J.: PBR Lighting. Learn OpenGL, Blog, 2014 [Online]. URL: https://learnopengl.com/PBR/Lighting. 3

[dVneg] DE VRIES J.: PBR Theory. Learn OpenGL, Blog, 2014 [Online]. URL: https://learnopengl.com/PBR/Theory. 2, 3

[dVneh] DE VRIES J.: Specular IBL. Learn OpenGL, Blog, 2014 [Online]. URL: https://learnopengl.com/PBR/IBL/Specular-IBL. 3

[Epinea] EPIC GAMES: Materials Compendium. Unreal Engine 3 Documentation, Online User Manual, 2006 [Online]. URL: https://docs.unrealengine.com/udk/Three/MaterialsCompendium.html. 2

[Epineb] EPIC GAMES: Materials. Unreal Engine 4.26 Documentation, Online User Manual, 2020 [Online]. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/Materials/. 2

[Epinec] EPIC GAMES: Unreal Engine, The world's most open and advanced real-time 3D creation tool. Unreal Engine Frontpage, Website, 2022 [Online]. URL: https://www.unrealengine.com/en-US. 1

[Frene] FREYA HOLMÉR: The ultimate visual node-based shader editor for Unity. Shader Forge Documentation, Online User Manual, 2013 [Online]. URL: https://acegikmo.com/shaderforge/. 2

[ICG86] IMMEL D. S., COHEN M. F., GREENBERG D. P.: A radiosity method for non-diffuse environments. *SIGGRAPH Comput. Graph. 20*, 4 (Aug 1986), 133–142. URL: https://doi.org/10.1145/15886.15901, doi:10.1145/15886.15901. 2

[Kaj86] KAJIYA J. T.: The rendering equation. *SIGGRAPH Computer Graphics 20*, 4 (Aug 1986), 143–150. URL: https://doi.org/10.1145/15886.15902, doi:10.1145/15886.15902. 2

[Karne] KARIS B.: Specular BRDF Reference. Graphic Rants, Blog, Aug. 03, 2013 [Online]. URL: http://graphicrants.blogspot.com/2013/08/specular-brdf-reference.html. 3

[KEne] KARIS B., EPIC GAMES: Real Shading in Unreal Engine 4. Epic Games. URL: https://blog.selfshadow.com/publications/s2013-shading-course/karis/s2013_pbs_epic_notes_v2.pdf. 3

[Linne] LINIETSKY, JUAN AND MANZUR, ARIEL: Godot, The game engine you waited for. Godot Engine Frontpage, Website, 2022 [Online]. URL: http://www.godotengine.org/. 1

[Marne] MARMOSET: Marmoset Toolbag, REAL-TIME RENDERING, TEXTURING, & BAKING TOOLS. Marmoset Toolbag Frontpage, Website, 2022 [Online]. URL: https://marmoset.co/toolbag/. 2

[Sidne] SIDEFX: Houdini. Houdini Frontpage, Website, 2022 [Online]. URL: https://www.sidefx.com/products/houdini/. 1

[Uninea] UNITY TECHNOLOGIES: Material. Unity Documentation, Online User Manual, 2015 [Online]. URL: https://docs.unity3d.com/520/Documentation/Manual/class-Material.html. 2

[Unineb] UNITY TECHNOLOGIES: Writing Vertex and Fragment Shaders. Unity Documentation, Online User Manual, 2015 [Online]. URL: https://docs.unity3d.com/520/Documentation/Manual/SL-ShaderPrograms.html. 2

[Uninec] UNITY TECHNOLOGIES: Introduction to Shader Graph: Build your Shaders with a Visual Editor. Unity Blog, Blog, 2018
[Online]. URL: https://blog.unity.com/technology/introduction-to-shader-graph-build-your-shaders-with-a-visual-editor. 2

[Unined] UNITY TECHNOLOGIES: Unity Engine, The world's leading platform for real-time content creation. Unity Engine Frontpage, Website, 2022 [Online]. URL: https://unity.com/. 1