

Editor de materiales para modelos en 3D basado en nodos.

Grado en Ingeniería Multimedia





Trabajo Fin de Grado

Autor:

Santiago Palacio Caro

Tutor/es:

Rafael Molina Carmona

Septiembre 2015

# Citas.

# Dedicatoria

# Agradecimientos

# Índices.

**Índice de contenidos.**

[1. Citas. 3](#_Toc425890041)

[2. Dedicatoria 4](#_Toc425890042)

[3. Agradecimientos 5](#_Toc425890043)

[4. Índices. 6](#_Toc425890044)

[5. Resumen 8](#_Toc425890045)

[6. Introducción 9](#_Toc425890046)

[7. Justificación y Motivación. 10](#_Toc425890047)

[8. Objetivos 12](#_Toc425890048)

[**General**. 12](#_Toc425890049)

[Específicos 12](#_Toc425890050)

[9. Estado del arte. 14](#_Toc425890051)

[Altamente integrado con el motor gráfico VS Independientes. 14](#_Toc425890052)

[Para renderizado en tiempo real VS renderizado offline. 15](#_Toc425890053)

[Editor de materiales de UE4. 16](#_Toc425890054)

[Editor de materiales 3ds Max 19](#_Toc425890055)

[Shader Forge 23](#_Toc425890056)

[Marmoset Toolbag 26](#_Toc425890057)

[Discusión 27](#_Toc425890058)

[10. Metodología y herramientas. 28](#_Toc425890059)

[Metodología de desarrollo de Software. 28](#_Toc425890060)

[Control de versiones 28](#_Toc425890061)

[11. Modelo Propuesto 29](#_Toc425890062)

[Especificación y Análisis de Requisitos 29](#_Toc425890063)

[Descripción general del sistema. 29](#_Toc425890064)

[Requisitos. 39](#_Toc425890065)

[Marco Teórico 39](#_Toc425890066)

[Diseño del sistema 41](#_Toc425890067)

[Diseño Grafico 41](#_Toc425890068)

[12. Experimentación 42](#_Toc425890069)

[13. Conclusiones y trabajos futuros. 42](#_Toc425890070)

[Aportaciones 42](#_Toc425890071)

[Comparación con los objetivos 42](#_Toc425890072)

[Trabajos futuros 42](#_Toc425890073)

[14. Bibliografía y Referencias 43](#_Toc425890074)

# Resumen

# Introducción

El presente actual es un momento en el cual pasamos por grandes periodos de cambio, la tecnología nos sorprende cada vez más. Su avance parece no detenerse y su potencial parece no agotarse, así mismo el campo de los gráficos por computador parece uno de los más ricos y prolíferos. Si bien ha crecido de manera notable en las últimas tres o cuatro décadas es evidente el impacto que tiene sobre determinados sectores económicos que manejan volúmenes de dinero considerables como los videojuegos, el cine y el ocio digital en general, han ayudado fuertemente a materializar este avance.

La tecnología avanza, esto implica que los dispositivos son cada vez mejores y las técnicas que antes servían para hacer una cosa, hoy en día son mucho más depuradas, optimizadas y en general mejores que en antaño. No obstante, hay que recalcar el software avanza en función del hardware, así pues, cosas que antes no podíamos hacer, hoy en día con las tarjetas gráficas modernas podemos hacerlas en nuestros propios hogares.

Han sido innumerables los aportes al campo de los gráficos por computador. Se han hecho grandes motores gráficos que son capaces de simular efectos y fenómenos, tan reales como si de una fotografía se tratara. Y es ahí, en el aspecto, en el acabado final de los objetos donde este trabajo pretende versar.

En gran medida, el realismo se consigue aplicando el color correcto –o aproximado- sobre las superficies de los objetos que son expuestos a alguna fuente de iluminación. La forma de determinar dicho color es calculada basándose en distintos parámetros. El material es un concepto que encapsula varios de estos parámetros, y define una forma general de comportarse bajo la incidencia de la luz para diferentes tipos de superficies o bien, **materiales**.

Definir un material en el mundo real puede ser algo sencillo, podríamos decir que con mencionar la palabra “metal”, nos referimos a que algún objeto esta hecho de metal. Lo mismo sucede con el cuero, o la roca. No obstante, para un artista que trabaja en gráficos 3D, estas descripciones resultan planas e insuficientes. Si bien un artista debe tener en mente a que material quiere convertir un objeto (cuero, metal, etc.), también se preocupa por entender cómo funciona la luz sobre una superficie, y así saber que parámetros son los que hacen que un material se comporte como tal. De esta forma el artista puede emular dicho material y conseguir el aspecto más realista posible.

Dicho lo anterior, este proyecto intenta aportar una solución para los artistas que necesitan integrar un material en un entorno 3D usando un sistema fácil, usable y entendible. Además de aportar total libertad al artista ya que el sistema le proporciona diferentes formas de resolver el problema. Todo ello usando las técnicas modernas de gráficos y los estándares actuales.

# Justificación y Motivación.

Diferentes son los motivos por los cuales se ha decidido realizar este trabajo. Cabe volver mencionar que se trata de un trabajo de fin de grado, por lo tanto, es un requisito indispensable para todos los estudiantes del régimen estudiantil, el presentar y superar dicha prueba para obtener el título de ingenieros que este conlleva. No obstante, el enfoque que se tiene de éste no es el de un obstáculo, sino por el contrario, la de una grandísima oportunidad para demostrar las habilidades conseguidas durante todo el periodo estudiantil y que mejor forma de hacerlo que con un trabajo de tema a elegir por el estudiante. Un desafío nunca mejor recibido y que se afrontó con toda la pasión y esfuerzos posibles.

Dicho lo anterior, también se ha de mencionar motivaciones más concretas fuera de lo que representa el trabajo de fin de grado como requisito para la obtención del título.

El estado actual del arte y de la industria involucrada en mantener sistemas en 3D es cada vez más complejo, aportando soluciones y métodos únicos que permiten obtener resultados cada vez más interesantes y logrados. La realización del trabajo aquí descrito es también motivada por el gran interés que me despiertan las mencionadas tecnologías. Así pues, uno de los principales focos de atención dentro de un entorno de gráficos en 3D es la posibilidad que dichos sistemas tienen de aportar a cada modelo en 3D un aspecto único, un material. Un material que le dota de un aspecto en ocasiones realista, estilizado o abstracto, según la finalidad.

Es en este punto donde el arte y la tecnología se unen en una simbiosis inmejorable y es la que le permite a artistas e ingenieros gráficos el lograr grandes creaciones a través de su trabajo en combinación. Es aquí donde este trabajo busca conseguir unificar de una forma nueva y más sencilla esa combinación entre los artistas y los ingenieros. Es por eso que la decisión tomada fue la de crear una herramienta para crear materiales para objetos en 3D, sencilla y fácil de usar, que permita al usuario que lo use, entender el proceso por el cual pasan los modelos en 3D hasta ser dibujados de una forma más intuitiva y más aún, entender el estado de Shading en donde se le asignan los colores a las superficies de los modelos teniendo en cuenta tanto parámetros del material como parámetros de la luz puntual dada.

Una vez tomada la decisión de desarrollar una herramienta para crear materiales y por consiguiente **Shaders** (pequeño programa capaz de controlar determinados estados del proceso de dibujado o *graphic pipeline*), se detectó un modelo frecuente de este tipo de aplicaciones en los principales productos de la industria actual – motores gráficos, herramientas de modelado, etc.-. Dichas aplicaciones tienen en común que todas son capaces de generar materiales usando una interfaz basada en nodos. Esto simplifica de una manera considerada la forma en que se generan dichos materiales, entre otros motivos, porque se sustenta de las virtudes de un VPL (visual programming language) como el fácil entendimiento del funcionamiento del programa debido a que está basado en gráficos y grafos intuitivos, entre otros.

Una aplicación basada en nodos y que genere las operaciones necesarias para un determinado material en función del grafo de nodos que se haya diseñado de antemano. Es aquí donde el artista sin conocimientos de programación, gráficos en profundidad, y HLSL/GLSL puede aportar todo su talento sin problema alguno.

En definitiva, éste trabajo es considerado como un buen – y desafiante- proyecto a desarrollar como parte del TFG ya que implica tener los suficientes conocimientos que conlleva este tipo de aplicaciones. Conocimientos que ha sido adquiridos durante los últimos años de carrera universitaria y que se han puestos a prueba en dicho proyecto.

# Objetivos

### General.

El objetivo general de este proyecto es el de realizar una herramienta capaz de generar Shaders que definan el aspecto de objetos en 3D. La aplicación debe funcionar bajo una interfaz basada en nodos. Los aspectos definidos para los modelos serán considerados materiales y será posible definir materiales usando operaciones frecuentes en la creación de Fragment Shaders. Dicha herramienta deberá satisfacer las necesidades básicas de alguien con un perfil profesional involucrado en gráficos 3D y Shading.

## Específicos

* Desarrollar un modelo de iluminación parametrizable y capaz de ser alimentado por la información que produzca el usuario usando el grafo de nodos.
* Obtener la capacidad para hacer uso de tecnologías nunca antes vistas haciendo uso del sentido de investigación y de aprendizaje que los años de carreara han aportado como parte de la experiencia educativa.
* Construir un sistema de nodos manipulable y fácil de usar en un espacio preparado para ello.
* Implementar una aplicación usable e intuitiva que permita al usuario llevar a cabo su tarea de la forma más sencilla posible proporcionándole soluciones para problemas típicos como la redundancia, las repeticiones y en definitiva mejorar el flujo de trabajo.
* Implementar una aplicación capaz de generar un Shader bajo los estándares de GLSL basado en el grafo de nodos definido por el usuario. Dicho Shader debe ser fácil de integrar en sistemas externos, o bien se le ha de dar usuario algún método para hacerlo.

# Estado del arte.

El término “editor de materiales “es un concepto bastante extendido en la actualidad (2015) en contextos de gráficos 3d por computador. Si bien existen gran variedad de herramientas que permiten de alguna forma proveer, modificar y administrar los parámetros que caracterizan un determinado material sobre una superficie o una malla geométrica, cabe destacar algunas características que son diferenciadoras entre unos y otros editores de materiales.

### Altamente integrado con el motor gráfico VS Independientes.

En determinados entornos, los editores de materiales son parte de un sistema mucho más complejo y funcionan como sub-sistema dentro de un motor gráfico o sistema más generales. Dichos editores funcionan de forma complementaria al sistema de renderizado de sus motores, estableciendo integridad en la parametrización de valores de cara a su ecuación de renderizado a varios niveles de complejidad. En otras palabras, existe una alta cohesión entre lo ofrecido por el editor de materiales y su puesta en escena por parte del motor, donde otros procesos entran en juego como la iluminación global, el sombreado de la escena y otros efectos más avanzados como el desenfoque de movimiento basado en mapeado de movimientos o la iluminación volumétrica etc. Todos ellos hacen parte del dibujado final de la escena pero no así de las propiedades físicas del material.

Hacer referencia a la rendering equation como función de dibujado final, y a la BDRF como intento por definir la reflectividad y color del objeto 3d ¿Aquí o en otro sitio anterior mejor?

Un ejemplo claro de este tipo de herramientas son las ya incorporadas en motores comerciales de alto desempeño como el motor gráfico Unreal Engine 4 (UE4) de Epic Games Inc. o el motor gráfico CryEngine de Crytek.

Por otra parte, si bien hemos dicho que existen editores pensados para un determinado motor de renderizado en tiempo real, también se da el caso de editores de materiales que por el contrario no están diseñados para un sistema de renderizado especifico, manejando conceptos y parámetros más genéricos y que se aproximan más al “estándar” de gráficos por computador permitiendo dibujar los materiales producidos bajo distintos motores de renderizado. Un caso claro de esto son los editores de materiales incluidos en las aplicaciones de edición 3d. En estas es común realizar el trabajo de edición de los materiales y luego elegir el motor de renderizado. Un buen ejemplo se da en el programa 3ds Max, donde podemos elegir por defecto entre el motor de renderizado Scanline o el motor de renderizado MentalRay. Además existen motores de terceros como el motor V-Ray que pueden ser añadidos y usarse para el renderizado dentro de 3ds Max.

Llegados a este punto cabe puntualizar que aunque estos editores manejen parámetros más generales, también pueden disponer de otros más avanzados en función del motor de renderizado que se va a utilizar, pero nunca abandonando los valores estándar. En el caso de 3ds Max, es frecuente ver como el editor de materiales se ve modificado añadiéndole nuevas interfaces que permiten el poder trabajar con dicho motor gráfico y sus parámetros específicos.

### Para renderizado en tiempo real VS renderizado offline.

Una importante característica más a destacar, es también el hecho de que el destino de algunos de estos editores sean sistemas de renderizado en tiempo real o bien de renderizado offline. Esto puede ser importante ya que ambos sistemas difieren sustancialmente en aspectos clave como la calidad del resultado o el tiempo de respuesta final, aun así, aunque ambos manejen aspectos claves en común, se debe más a una herencia de modelos de iluminación clásicos ligados a las condiciones físicas de la naturaleza, más que a su forma de procesar el dibujado en sí.

Un claro ejemplo de esto se da en los parámetros básicos como de factor de especularidad de una superficie, o su valor de componente difusa. Si bien la forma en que un editor para renderizado en tiempo real los manejan respecto a otro offline puede resultar similar, a la hora de realizar el dibujado final, sus proceso internos son radicalmente distinto, ya que en la mayoría de los casos, un sistema offline usa métodos basados en Raytracing, los cuales son inviables para renderizado en tiempo real debido a su gran costo computacional.

Tipo de interfaz: Basada en Nodos VS Clásica.

Cons y pros de una y de otra

A continuación se hará un análisis de los editores de materiales que más han influido este trabajo y sus principales características.

### Editor de materiales de UE4.

Este editor permite la creación de Shaders para aplicarlas sobre alguna superficie geométrica determinada. Dicho editor viene incorporado en el motor gráfico UE4 y hace parte de uno de sus subsistemas.

Su interfaz gráfica basada en nodos, es una de sus características más destacables. Es frecuente en UE4 ver dicho tipo de interfaces basada en nodos. Si bien sirve como interfaz de usuario para el desarrollo de materiales, también entra en la categoría lenguaje de programación visual (VPL) el cual permite programar y generar Shaders mediante la manipulación de elementos visuales, en este caso entre nodos y sus conexiones. El editor de materiales de UE4 permite la creación de una variedad considerable de nodos que bien representan una operación la cual encapsula una función determinada, o bien nodos que representan algún tipo de información como valores de color, o de mapas de texturas. Cada nodo está provisto de entradas y salidas a las cuales se les puede conectar otros nodos ya bien sea como entradas o como salidas. El número de entradas depende del tipo de nodo.

Este tipo de interfaz es común en otros subsistemas del UE4 como lo son los ***Blueprints***, en donde la finalidad no es crear materiales sino modificar o crear características del gameplay, cámaras y otros tipos de elementos de una forma más intuitiva y rápida, más aun para gente no programadora.

#### Physically-based shading

UE4 utiliza un modelo de shading basado en físicas (Physically-based shading model). Los resultados finales son más precisos y con un aspecto típicamente más natural. El salto gráfico respecto a su predecesor Unreal Engine 3 (UE3) es considerable.

Los modelos basados en físicas (Physically-based), son la característica de moda en cuanto a gráficos en tiempo real de nueva generación.

Su condición de physically-based le permite establecer propiedades de los materiales más intuitivas, más cercanas a la realidad y al cómo funcionan los fenómenos físicos de iluminación en la naturaleza. UE4 define un material basándose en 4 propiedades básicas: BaseColor, Metallic, Roughness y Specular.



*Nodo principal del panel de grafo en UE4. Por defecto este nodo está siempre presente. A él son conectados los inputs creados en el editor para generar el resultado final*.

La propiedad ***BaseColor*** define un color total para el material del objeto. Recibe como parámetro de entrada un vector de tres dimensiones donde cada componente representa los valores de intensidad para el estándar de color RGB. Estos valores son restringidos al rango [0-1].

Por su parte, la propiedad ***Roughness*** controla el nivel es aspereza del material. Un material con un alto nivel Roughnessrefleja luz a mas direcciones que uno con poco nivel dejando una luz especular más difusa y borrosa, alternativamente un material con un nivel de Roughness más bajo refleja luz en parcialmente una misma dirección y tiene una luz especular mucho más definida tendiendo a imitar el comportamiento de un espejo.

La propiedad ***Metallic*** controla el nivel de metalizado de la superficie. Fenómenos como la cantidad de luz que es absorbida por el material son controladas por este parámetro contribuyendo a definir el nivel de metalizado del material. Un material con un valor de 1 se comporta como totalmente como un metal mientras que un material con un valor de 0 no poseería ninguna característica metálica (rocas, madera, etc.).

La propiedad ***Specular*** recibe un valor entre 0 y 1 y se utiliza para escalar la cantidad actual de especularidad en superficies no metálicas. Es decir, no tiene ningún efecto sobre los metales. Por defecto, si no se le pasa un valor, el valor inicial es de 0.5.

Si se modifica el valor de especularidad, se hace para añadir micro colusiones o sombreado a pequeña escala, representadas en el mapa de normales. Esto en ocasiones es referido como cavidades.

Geometría a pequeña escala, especialmente detalles solo presentes en el modelos high poly y luego generados (baked) sobre el mapa de normales, no son recogidas por las sombras del renderizador en tiempo real. Para capturar estas sombras, el motor UE4, genera un mapa de cavidades (Cavity), el cual es típicamente un mapa AO (Ambient Oclusion) con poca distancia de trazado. Este mapa es multiplicado por el *BaseColor* y posteriormente multiplicado por 0.5 (especularidad por defecto) como la salida final de la propiedad ***Specular***. Es decir:

BaseColor = Cavity \* OldBaseColor, Specular = Cavity \*0.5

#### Interfaz gráfica.

Aquí se describe la interfaz grafica

### Editor de materiales 3ds Max

El editor de materiales de 3ds Max ofrece dos versiones en su interfaz de usuario: La versión compacta y la versión de pizarra (Slate editor). Ambas opciones producen la misma calidad de resultados en términos técnicos. No obstante la diferencia entre ambos es considerable.

#### Versión compacta:

La versión compacta es representada en una pequeña ventana modal, invocada desde el la interfaz principal. Los materiales generados se ven en la parte superior de la ventana en forma de grid. Por defecto 3ds Max ofrece una serie de materiales a partir de los cuales se pueden empezar a alterar propiedades. Sin embargo el más típicamente usado en términos generales es el llamado “Standard”. Dicho perfil de material posee las propiedades más típicamente usadas y entendidas por gráficos por computador.

#### Análisis de la interfaz gráfica

La interfaz gráfica de 3ds Max, en su mayoría, está estructurada por rollouts\*. El modelo estándar tiene como principales rollouts a:

* Shader basic Parameters: Aquí se puede definir el mediante que función de reflexión se ha de interpretar el material. Entre sus opciones más destacadas se encuentran: Blinn, Phong, Metal y etc.
* Basic Parameters: Sus opciones cambian según el modelo de reflexión elegido en el rollout anterior. Por lo general presenta los valores más típicos del material:

**Ambient**: Constante de color que intenta simular la contribución lumínica de las reflexiones globales de la escena. Recibe un color rgb como parámetro de entrada.

**Diffuse**: Se trata del color base del material. La cantidad de luz que es reflejada en direcciones uniformes. Recibe un color rgb como parámetro de entrada.

**Specular** (Color): Define el color con el que las luces especulares son interpretadas. Recibe un color rgb como parámetro de entrada.

**Specular Level:** Indica el nivel de especularidad del material. Un material con un valor especular igual a 0 no reflejara luces especulares hacia la cámara, mientras que un material con un valor especular mucho más alto tendrá mucha riqueza de dichas luces en su superficie.

**Glossines:** Define el la fuerza de las luces especulares. De ello depende cuán grande son las luces especulares que son reflejadas. Si el valor es muy bajo, se producen luces más grandes con gradientes mayores. Si el valor es alto, la luz especular se decrementa al igual que lo hace el gradiente, dejando luces mucho más definidas y puntuales.

* Maps: En este rollout están presentes los slots a los cuales se pueden introducir distintos tipos de mapas para dotar al material de diferentes características. Por lo general, a cada mapa se le es asignado un componente de tipo Bitmap, pues es una acción típica el importar mapas para diferentes acabados en ficheros de tipo imagen y que son generados, generalmente, en otro tipo de programas como Photoshop. Sin embargo a dichos slots se les puede asignar otro tipo de componente. 3ds Max ofrece gran variedad de mapas, algunos por ejemplo, más procedurales capaces de generar mapas de ruido aleatoriamente como el componente “Noise”, u otros capaces de generar gradientes como el componente “Gradient”. Por lo tanto, estos valores no tiene por qué estar enteramente ligados a un mapa imagen, aunque suele ser lo más frecuente en el flujo de trabajo de la industria de gráficos en tiempo real como los videojuegos.

Entre los canales típicamente más reconocibles se encuentran:

**Diffuse Color:** Recibe un mapa (por lo general una textura), que aporta el color base a la superficie del material. Los valores de coordenadas UW también están a disposición del usuario para su manipulación.

**Specular Level:** Recibe un mapa de valores especulares.Este tipo de mapa, típicamente está en escala de grises y controla el comportamiento del índice especular a lo largo de todo el modelo.

**Opacity:** Recibe un mapa de opacidad donde el valor de los pixeles define la visibilidad de la superficie.

**Bump:** Recibe un mapa que, típicamente está en escala de grises y que define micro detalles y rugosidades a lo largo del material. Para ello, el mapa es capaz de modificar la dirección de las normales encapsulando valores numéricos sobre cada pixel, normalmente, en espacio de coordenadas tangencial.

#### Versión Slate.

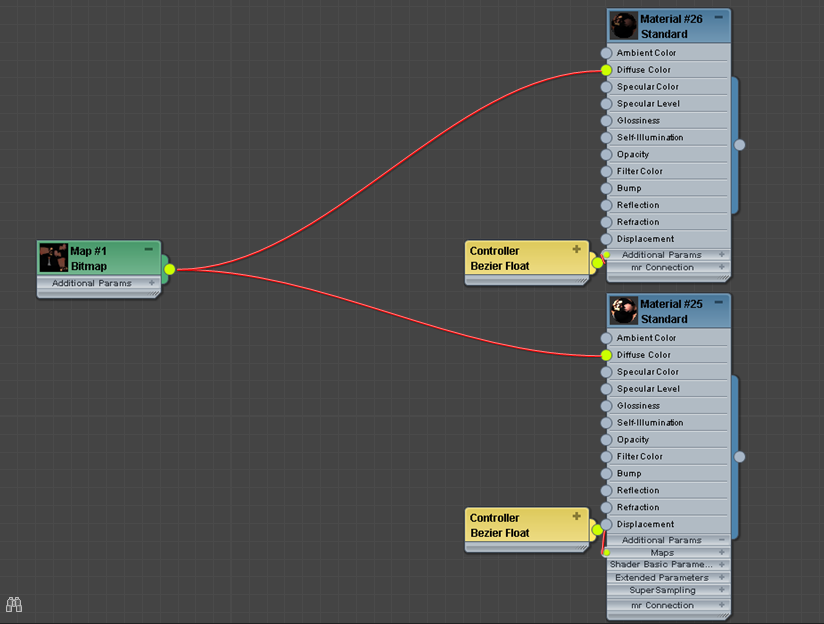
Esta versión es una aproximación más cercana a la propuesta en este proyecto en su interfaz gráfica.

Su interfaz de usuario deja de ser la convencional interfaz compacta para pasar a ser una más actualizada respecto a las tendencias actuales de edición de materiales. Dicha interfaz está basada en nodos, lo cual la hace mucho más adecuada a las demandas y flujos de trabajo actuales de este tipo de edición.

El espacio de trabajo tiene como componente principal un área donde se pueden crear nuevos materiales bien sea arrastrándolos de una lista de perfiles o haciendo clic derecho sobre dicha área y eligiendo el perfil del material a iniciar. En ese momento se crea un nodo, este nodo principal contiene una lista de inputs a los cuales se les pueden conectar nuevos nodos. Los inputs representan las propiedades expuestas para dicho material en dicho perfil.

*Nodo principal correspondiente a un material de tipo* ***Standard.***

En el mismo espacio de trabajo se pueden generar más materiales permitiendo al usuario reutilizar nodos o resultados de nodos entre distintos materiales que conviven en el espacio de trabajo. Lo cual ofrece mucha más potencia a la herramienta al igual que escalabilidad.



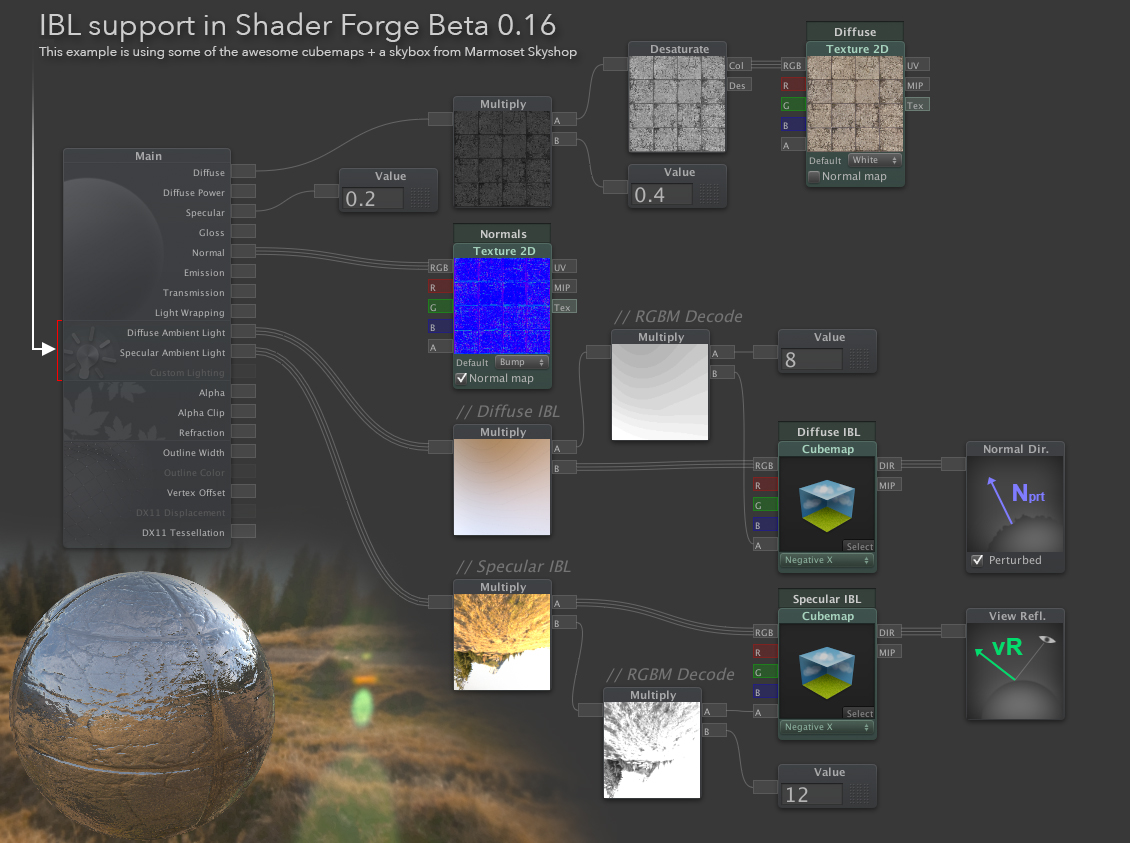
*Dos materiales compartiendo y reutilizando una textura representada por un nodo de tipo Bitmap*

### Shader Forge

Se trata de un editor de Shaders basado en nodos. Está diseñado para que funcione bajo el motor gráfico Unity, de tal forma se presenta como una extensión de pago del motor. La licencia es adquirida a través de la Unity Asset Store – tienda de contenidos y extensiones de Unity-.

Shader Forge trabaja con modelos de iluminación basado en físicas (PBR) y shading de conservación de la energía haciendo uso de los modelos Blinn-Phong o Phong.

Además Shader Forge tiene la capacidad de poder trabajar con *Image Based Lighting* lo que le permite trabajar con técnicas como la de *Cube Maps* para visualizar la forma en que se iluminan los objetos, y generar valores para *Diffuse Ambient Light* y *Specular Ambient Light*. Gracias a esto Shader Forge es compatible con shaders y herramientas de la familia Marmoset (Marmoset Skyshop).



*(Ejemplo de un Shader creado por Shader Forge usando Skyboxes de Marmoset Skyshop)*

Su flujo de trabajo intuitivo es perfecto para artistas quienes no está muy habituados a programar ni a generar shaders mediante programación de texto.

El nodo principal de Shader Forge está compuesto por 21 inputs a los que se les puede pasar un valor que modificara los parámetros del material y de la iluminación del objeto al que se le aplique el Shader. Entre los 21 inputs se encuentran algunos clásicos como Diffuse, Specular y Normal, así como también algunos más personalizados como Custom Lighting el cual le permite al usuario definir un comportamiento de iluminación personalizado sin atender a las físicas de iluminación preestablecidas por el modelo de Shader Forge.



*(Conjunto de inputs de Shader Forge)*

Shader Forge dispone de una número de operaciones considerable que permiten la creación de nuevos valores resultado de operaciones realizadas sobre otros mismos. Operaciones como suma, resta, interpolación, mínimo, máximo, valor absoluto, clamp, blend (fundido), etc. son típicas en el desarrollo mediante Shader Forge.



*(Ejemplo de una operación de suma entre un nodo de tipo textura y un vector representando un color del tipo RGB)*

### Marmoset Toolbag

Marmoset Toolbag es una herramienta principalmente de renderizado de alta calidad basado en físicas (PBR).

Marmoset Toolbag permite a artistas visualizar su trabajo en un render de alta calidad sin necesidad de utilizar un motor gráfico para lograrlo. Esto proporciona un estado de estandarización más amplio ya que muchos artistas pasan antes por Marmoset Toolbag que por el motor y empieza a convertirse en un referente de testeo para materiales en PBR.

La calidad de su render es la principal característica de Marmoset Toolbag. Está pensado para visualizar escenas generalmente pequeñas, con iluminación basada en mapas (Cubemaps y Skyboxes) para modelar el comportamiento en el que la luz refleja determinado modelo. Además de poseer una gran gama de características de renderizado que le dotan de gran vistosidad y realismo en sus escenas.

******

*(Modelo 3D renderizado con Marmoset Toolbag en tiempo real.*

*Imagen por: Colt Python by Benjamin Turner | turnededges.com)*

Entre los parámetros que define un material en Marmoset Toolbag se encuentran: **Albedo,** (define el color base del material), **Specular, Gloss y Normales.** Otras características de materiales que se pueden encontrar en Marmoset Toolbag son:

* Dynamic Tessellation
* PN Triangle mesh smoothing
* Height & Vector Displacement modes
* Detail Normal Maps
* Parallax Occlusion Maps
* "Metalness" Maps
* GGX, Blinn-Phong, & Anisotropic Reflections
* Secondary Reflections
* Skin Diffusion
* Microfiber ("Fuzz") Diffusion
* Occlusion and Cavity Maps
* Emissive Maps
* [**Allegorithmic**](http://www.allegorithmic.com/) Substances
* Dithered Supersampled Transparency

## Discusión

# Metodología y herramientas.

### Metodología de desarrollo de Software.

El proyecto software se ha desarrollado bajo una planificación donde el estudiante visitaba periódicamente al tutor. En dichas visitas se acordaba qué nuevas tareas se habrían de realizar para la siguiente visita. Las visitas se realizaban semanalmente en la mayoría de los casos.

Es por esto que la metodología seguida fue una basada en Sprints de 7 días por lo general, en donde se presentaban nuevas funcionalidades cada vez.

Como tal no se ha seguido una metodología de desarrollo. Incluso el desarrollo en algunos momentos conto con periodos de evaluación más prolongados debido a inconvenientes por parte del autor para acudir a las visitas de evaluación.

### Control de versiones

Es de vital importancia contar con una herramienta que permita llevar a cabo un control de versiones por diferentes motivos. Para controlar el desarrollo, para publicar, y más importante aún, para tener un respaldo siempre presente ante cualquier imprevisto.  
Dada las características del proyecto, y tratándose de un trabajo principalmente académico, la decisión para elegir una herramienta para el control de versiones dio como resultado el uso de la herramienta GitHub.

El servicio prestado por GitHub es gratuito, lo que se ajusta perfectamente al bolsillo de todo el público. Su herramienta de control de versiones más que suficiente para llevar a cabo el desarrollo del proyecto de forma íntegra.

La dirección para encontrar el proyecto es: <https://github.com/s4ntia60/spc-editor/>

Meter Herramientas

# Modelo Propuesto

## Especificación y Análisis de Requisitos

### Descripción general del sistema.

El proyecto es una aplicación de escritorio. El sistema operativo sobre el cual correrá pertenece a las plataformas Windows y es desarrollada y testada específicamente sobre Windows 8.

La aplicación debe contar con una interfaz gráfica que posibilitará al usuario el desarrollar todas las tareas necesarias y ofrecidas. La interfaz se divide en X número de secciones.

#### Ventana.

Contenedor principal donde se alojara la aplicación. Este debe contar con los elementos típicos de una ventana en Windows como botones para cerrar, maximizar y minimizar, así como también controles para redimensionar la aplicación.

#### Barra de Menús.

Aquí se contendrán los principales botones de menú.

Describir los menús aquí

#### Editor de nodos.

El editor de nodos es un área de trabajo donde se pueden manipular cajas que representan los nodos de funciones dentro del Fragment Shader. Al iniciar el sistema, se contara con un nodo principal el cual contiene solo inputs. Este nodo contiene los inputs básicos a definir para construir un material nuevo. Estos inputs son:

**Color:**

Aquel valor que se conecte a este input será el asignado al componente difuso del modelo.

**SpecularColor**

Se le asigna al componente especular del modelo.

**Shininess**

Regula el exponente de especularidad dentro del modelo.

**Normal**

Modifica la dirección de la normal para realizar el Shading.

**Alpha**

Sirve como mascara para modificar el valor de alpha.

Para generar materiales se han de concatenar distintos nodos que representan una función, cada nodo contiene sus propios inputs y outputs, en ocasiones pudiendo no tener alguno de ellos, como en el caso de determinados nodos que solo aportan información constante como por el ejemplo el caso de los nodos de texturas y los nodos de vectores.

A continuación se hace una descripción de cada uno los posibles nodos con los que se puede trabajar en la aplicación:

**Nodos disponibles.**

|  |  |
| --- | --- |
| Nombre: | Const Value Node |
| **Diseño** | \* |
| **Descripción:** | Almacena un valor único en forma de float. |
| **Inputs:** | Ninguno. |
| **Outputs:** | Devuelve un único vector donde todos sus componentes son el valor almacenado. |

|  |  |
| --- | --- |
| Nombre: | Vector 2 Node |
| **Diseño** |  |
| **Descripción:** | Almacena un vector 2D con componentes RG. |
| **Inputs:** | Ninguno. |
| **Outputs:** | R: Devuelve el componente de rojo del vector.  G:Devuelve el componente de verde del vector |

|  |  |
| --- | --- |
| Nombre: | Vector 3 Node |
| **Diseño** |  |
| **Descripción:** | Almacena un vector 3D con componentes RGB. |
| **Inputs:** | Ninguno. |
| **Outputs:** | R: Devuelve el componente de rojo del vector.  G:Devuelve el componente de verde del vector  B: Devuelve el componente azul del vector. |

|  |  |
| --- | --- |
| Nombre: | Vector 4 Node |
| **Diseño** |  |
| **Descripción:** | Almacena un vector 4D con componentes RGBA. |
| **Inputs:** | Ninguno. |
| **Outputs:** | R: Devuelve el componente de rojo del vector.  G:Devuelve el componente de verde del vector  B: Devuelve el componente azul del vector.  A: Devuelve el componente alpha del vector. |

|  |  |
| --- | --- |
| Nombre: | Texture Node |
| **Diseño** |  |
| **Descripción:** | Almacena los valores mapeados de una textura para un modelo. |
| **Inputs:** | Ninguno (Se ha de introducir la ruta de la textura) |
| **Outputs:** | V: Devuelve el vector 4D que componen los valores RGBA de la textura.  R: Devuelve el componente de rojo del vector.  G:Devuelve el componente de verde del vector  B: Devuelve el componente azul del vector.  A: Devuelve el componente alpha del vector. |

|  |  |
| --- | --- |
| Nombre: | Add |
| **Diseño** |  |
| **Descripción:** | Calcula la operación de Suma. |
| **Inputs:** | A y B: Operandos para realizar la suma. |
| **Outputs:** | V: Devuelve el resultado de la Suma. |

|  |  |
| --- | --- |
| Nombre: | Subtract |
| **Diseño** |  |
| **Descripción:** | Calcula la operación de Resta. |
| **Inputs:** | A y B: Operandos para realizar la resta. |
| **Outputs:** | V: Devuelve el resultado de la resta. |

|  |  |
| --- | --- |
| Nombre: | Multiply |
| **Diseño** |  |
| **Descripción:** | Almacena un valor único en forma de float. |
| **Inputs:** | Ninguno. |
| **Outputs:** | Devuelve un único vector donde todos sus componentes son el valor almacenado. |

|  |  |
| --- | --- |
| Nombre: | Power |
| **Diseño** |  |
| **Descripción:** | Almacena un valor único en forma de float. |
| **Inputs:** | Ninguno. |
| **Outputs:** | Devuelve un único vector donde todos sus componentes son el valor almacenado. |

|  |  |
| --- | --- |
| Nombre: | Sqrt |
| **Diseño** |  |
| **Descripción:** | Almacena un valor único en forma de float. |
| **Inputs:** | Ninguno. |
| **Outputs:** | Devuelve un único vector donde todos sus componentes son el valor almacenado. |

|  |  |
| --- | --- |
| Nombre: | Log |
| **Diseño** |  |
| **Descripción:** | Almacena un valor único en forma de float. |
| **Inputs:** | Ninguno. |
| **Outputs:** | Devuelve un único vector donde todos sus componentes son el valor almacenado. |

|  |  |
| --- | --- |
| Nombre: | Min |
| **Diseño** |  |
| **Descripción:** | Almacena un valor único en forma de float. |
| **Inputs:** | Ninguno. |
| **Outputs:** | Devuelve un único vector donde todos sus componentes son el valor almacenado. |

|  |  |
| --- | --- |
| Nombre: | Max |
| **Diseño** |  |
| **Descripción:** | Almacena un valor único en forma de float. |
| **Inputs:** | Ninguno. |
| **Outputs:** | Devuelve un único vector donde todos sus componentes son el valor almacenado. |

|  |  |
| --- | --- |
| Nombre: | Abs |
| **Diseño** |  |
| **Descripción:** | Almacena un valor único en forma de float. |
| **Inputs:** | Ninguno. |
| **Outputs:** | Devuelve un único vector donde todos sus componentes son el valor almacenado. |

|  |  |
| --- | --- |
| Nombre: | Sign |
| **Diseño** |  |
| **Descripción:** | Almacena un valor único en forma de float. |
| **Inputs:** | Ninguno. |
| **Outputs:** | Devuelve un único vector donde todos sus componentes son el valor almacenado. |

|  |  |
| --- | --- |
| Nombre: | Mod |
| **Diseño** |  |
| **Descripción:** | Almacena un valor único en forma de float. |
| **Inputs:** | Ninguno. |
| **Outputs:** | Devuelve un único vector donde todos sus componentes son el valor almacenado. |

|  |  |
| --- | --- |
| Nombre: | Clamp |
| **Diseño** |  |
| **Descripción:** | Almacena un valor único en forma de float. |
| **Inputs:** | Ninguno. |
| **Outputs:** | Devuelve un único vector donde todos sus componentes son el valor almacenado. |

|  |  |
| --- | --- |
| Nombre: | Lerp |
| **Diseño** |  |
| **Descripción:** | Almacena un valor único en forma de float. |
| **Inputs:** | Ninguno. |
| **Outputs:** | Devuelve un único vector donde todos sus componentes son el valor almacenado. |

##### Conexiones

Las conexiones se realizan haciendo clic sobre un input o output, manteniendo pulsado el botón del mouse se arrastra hasta el puerto donde se quiera conectar estableciendo así una nueva conexión. Para realizar una conexión se ha de tener en cuenta ciertas restricciones de conexión entre nodos y puerto.

Restricciones de conexión:

* + Un input solo puede tener una conexión asignada. Si se intenta conectar hacia un input que ya estaba previamente establecido, la conexión se realizará, pero, la anterior conexión se eliminara para dar paso a la nueva.
  + Un output puede proveer más de una conexión, lo que facilita la reutilización de nodos, y evita redundancias.
  + Para eliminar una conexión se debe pulsar clic derecho sobre la misma o sobre el puerto lo que eliminaría el conjunto de conexiones ya sean salientes o entrante.
  + Un nodo no puede ser fuente de alimentación de sí mismo, por lo tanto, si se intenta usar un miembro de un mismo nodo para conectar un puerto de dicho nodo, la conexión no se realizará.

#### Visor 3D

Se trata de un panel localizado en la parte lateral de la ventana. En él se ubica un visor OpenGL con una escena 3D. En esta escena se dibujara el resultado actual de las distintas operaciones por las que está pasando el modelo. Como ya se ha dicho, las operaciones reflejadas en los nodos son analizadas convertidas en un Shader quien es el que permite su dibujado.  
Si hay algún error, el Shader se descarará, y se pasará a dibujar el modelo con un material de error fácilmente reconocible.

En definitiva, el visor 3D es donde se muestra el resultado del estado del editor de nodos.

#### Panel de propiedades Lateral.

Este panel tiene como función principal mostrar propiedades del nodo seleccionado. Aquí se podrán definir o modificar propiedades que los distintos nodos poseen. Este panel es importante, porque la introducción de datos típica no está muy presente es la aplicación y hay determinados momentos donde conviene (y se necesita) introducir algunos datos manualmente.

##### Explicación de los menús de propiedades:

**…**

#### Interacción y uso.

El editor de nodos es un poderoso medio a la hora de crear flujos de funciones concatenadas. La interacción es fluida y los elementos visuales responden de forma correcta.

Interacciones:

Los nodos se pueden manipular y mover de sitio a cualquier parte del área del editor de nodos lo que le aporta al usuario un método de organizar su trabajo personalizado.

Cada sección de la ventana se puede redimensionar pudiéndose ensanchar o acortar partes que se deseen.

Desde el visor 3D, el usuario podrá tener control de la cámara de la escena mediante las teclas y botones:

Para crear un nuevo nodo se ha de dar clic derecho sobre un espacio vacío en el editor de nodos, en ese momento un menú contextual con los nuevos nodos se desplegará y se podrá elegir cual será el nodo a crear.

### Casos de uso

Definir

### Requisitos.

#### Requisitos funcionales

#### Requisitos no funcionales

### Marco Teórico

#### Introducción a gráficos por PC

Este trabajo pretende construir un modelo de iluminación alimentable por los datos que el usuario genere y se pase como parámetros al modelo. Debido a esto, se ha visto que lo más conveniente sea que continuación se haga una breve explicación sobre el marco teórico, físico, y tecnológico sobre el cual se fundamenta este trabajo.

#### Fenómeno Visual

Cuando se estudia el dibujado por ordenador es útil primero entender algunos comportamientos que caracterizan las imágenes que percibimos y su posterior técnica para simularla en sistemas computacionales.

En primera instancia la luz es un fenómeno físico, que se puede estudiar como una onda que se mueve en el espacio, o bien, se puede entender como un cuerpo minúsculo que viaja a gran velocidad en línea receta llamado Fotón. Esta la aproximación de “una partícula que viaja en línea recta “es que más interesa de momento.

La luz es emitida por un cuerpo como el sol o las luces artificiales que encontramos en nuestro entorno. Esta luz interactúa con objetos, una parte de dicha luz es absorbida y otra parte es dispersada. Son estos dos fenómenos los que alteran el aspecto de un objeto cuando son recogidos por un sensor, bien sea el ojo humano, una cámara fotográfica, un sensor eléctrico etc.

#### Fuentes de luz

Las fuentes de luz son capaces emitir partículas de luz a distintas direcciones, esta luz es energía radiante electromagnética, es decir, energía que viaja por el espacio.

Radiometria y Colorimetria.

Las fuentes de luz pueden ser representadas de diferentes formas para propósitos de renderizado:

##### Luces direccionales:

Las luces producidas por objetos como el sol u objetos que se perciben como generadores de luz masivos, son fácilmente representadas con una luz direccional. El método consiste en mantener un vector dirección que indica la dirección hacia la cual viaja la luz.

##### Luces puntuales:

Son luces que se representan en un punto del espacio desde el cual se genera luz en todas las direcciones. El vector dirección (l) y la contribución de irradiación (El) de este tipo de luces depende de la posición de la superficie sobre la que se esté calculando la intensidad de luz:

r || Pl – Ps||

l = Pl – Ps / || Pl – Ps||

El = || Pl – Ps|| / r2

Donde Pl es la posición de la luz y Ps la posición de la superficie.

Típicamente este tipo de luz es simulada bajo un valor de atenuación que le dota de realismo y que hace que no se iluminen todos los objetos de una escena con la misma luz e intensidad que los demás. Para calcular el valor de atenuación se pueden usar diferentes funciones. Es común que distintos desarrolladores usen cada uno su función ya que simular físicamente estos valores puede llegar a ser mas complejo.

Funcion de atenuación de OpenGl y DirectX

Fdis® = 1 / sc + scr + sqr2

##### Sporlight

Explicarla

The Shading Equation

BRDF

#### Shading

Shading es el proceso de usar una ecuación para computar la radiance saliente *L*0 a lo largo del rayo de visión, **v**, basándose en las propiedades del material y de la fuente de luz.

##### Modelos de Shading

Shading de Phong, Blinn-Phong, Coork-Torrance etc.

## Diseño del sistema

Diseño

Arquitectura

.

.

.

# Experimentación

Pequeña introducción a lo realizado y luego pasar a comparar resultados con otros productos. Ademas mostrar nuevas características como la de incluir una operación personalizada que engloba otra.

# Conclusiones y trabajos futuros.

## Aportaciones

## Comparación con los objetivos

## Trabajos futuros

# Bibliografía y Referencias

[***http://acegikmo.com/shaderforge/***](http://acegikmo.com/shaderforge/)

[***http://www.marmoset.co/toolbag/learn/pbr-theory***](http://www.marmoset.co/toolbag/learn/pbr-theory)

[***https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/PhysicallyBased/index.html***](https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/PhysicallyBased/index.html)

***Real Time Rendering***

***3D Math Primer For Graphics and Game Development***