



Escuela
Politécnica
Superior

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



Grado en Ingeniería Multimedia

Trabajo Fin de Grado



Universitat d'Alacant
Universidad de Alicante

Autor:

Santiago Palacio Caro

Tutor/es:

Rafael Molina Carmona

Septiembre 2015

1. Citas.

2. Dedicatoria

3. Agradecimientos

Índices.

Índice de contenidos.

1. Citas.....	3
2. Dedicatoria	4
3. Agradecimientos.....	5
Índices.....	6
4. Resumen	8
5. Introducción.....	9
6. Justificación y Motivación.....	10
7. Objetivos.....	12
General	12
Específicos	12
8. Estado del arte.	14
Altamente integrado con el motor gráfico VS Independientes.....	34
Para renderizado en tiempo real VS renderizado offline.	36
Editor de materiales de UE4.	37
Editor de materiales 3ds Max	39
Shader Forge	43
Marmoset Toolbag	46
Discusión	47
9. Metodología y herramientas.....	48
Metodología de desarrollo de Software.....	48
Control de versiones	48
10. Modelo Propuesto	54
Especificación y Análisis de Requisitos	54

Descripción general del sistema.....	54
Casos de uso.....	¡Error! Marcador no definido.
Requisitos.	67
Marco Teórico	¡Error! Marcador no definido.
Diseño del sistema.....	78
11. Experimentación.....	78
12. Conclusiones y trabajos futuros.	78
Aportaciones	78
Comparación con los objetivos.....	78
Trabajos futuros.....	78
13. Bibliografía y Referencias.....	79

4. Resumen

5. Introducción

En el presente actual 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.

Comentado [SP1]: Revisar (Suenan mal) no pega

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 está 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 qué 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.

6. 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. Pero en definitiva es donde recae la belleza visual y la verdadera esencia artística y tecnológica.

Comentado [SP2]: Se habla en primera persona o en tercera persona (autor)

Es en este punto donde el arte y la tecnología se unen permitiendo 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 estos dos mundo. 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.

Comentado [SP3]: Se habla en presente, pasado o futuro?

7. 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.

Comentado [SP4]: Alarga mucho el párrafo y parece innecesario

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 carrera 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

Comentado [SP5]: Puede que puedan ir juntos estos dos párrafos

ser fácil de integrar en sistemas externos, o bien se le ha de dar usuario algún método para hacerlo.

8. Estado del arte y contexto.

Este trabajo pretende construir modelo de iluminación un no muy complejo fácilmente alimentable por los datos que el usuario genere, datos que finalmente serán los que se le pasen al modelo como parámetros del mismo.

A continuación se hará un análisis a grandes rasgos de los fundamentos teóricos y técnicos necesarios para entender la consecución de un sistema de dibujado 3D en tiempo real, tomando en cuenta los fenómenos físicos y también los procedimientos modernos más comunes en el campo de los gráficos.

Comentado [SP6]: Reescribir

Fenómeno Visual

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

“Like so many challenges in computer science, a great place to start is by investigating how nature Works”.

3D Math primer – Pag 345.

En primera instancia la luz es un fenómeno físico, que se puede entender 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 recta (fotón).

La luz es emitida por cuerpos 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.

La cantidad de luz dispersada junto con la cantidad de luz absorbida son el resultado de la interacción física que tiene el material con el rayo de luz en definitiva, y quien por lo tanto es capaz de determinar el aspecto de las superficies. En pocas

palabras, la cantidad de luz reflejada y absorbida dotan a una superficie de un aspecto visual determinado. Este comportamiento podría ser modelado generando un conjunto de propiedades (en determinados casos, véase tipos de BRFD) que determinan eventualmente la forma en que se manejan las superficies bajo una fuente de luz, a este conjunto de propiedades podemos llamarla a este punto, **Material**. Más adelante hablaremos en profundidad de los materiales.

Radiometría y Colorimetría.

Cabe valorar a este punto la importancia de hacer un análisis sobre la radiometría y la colorimetría. Si bien son dos conceptos que pueden resultar ajenos a la informática y a los gráficos, son fuentes importantes de estudio en términos de medir y modelar el comportamiento de la luz y su percepción sobre nuestros ojos, pues ofrecen valiosas herramientas que facilitan su entendimiento.

“La radiometría es la ciencia que estudia la medición de la radiación electromagnética. Su campo abarca las longitudes de onda del espectro electromagnético (frecuencias entre 3×10^{11} y 3×10^{16} Hz), al contrario de la fotometría que solo se ocupa de la parte visible del espectro, es decir, la que puede percibir el ojo humano.” (Wikipedia)

El concepto popular que todos tenemos sobre el color es que este es una combinación de en alguna medida la luz roja, verde y azul (RGB). No obstante, este concepto no es del todo correcto, pues bien, la luz puede tomar cualquier frecuencia en el ancho de banda del espectro visible, o una combinación de ellas. El color es un fenómeno de percepción humana y no es lo mismo que una frecuencia. De hecho diferentes frecuencias de luz pueden ser percibidas como diferentes colores, a este fenómeno se le conoce como *metamers*. (3D Math Primer pag 353).

“La colorimetría es la ciencia que estudia la medida de los colores y que desarrolla métodos para la cuantificación del color, es decir, obtener valores numéricos del color.” (Wikipedia)

La colorimetría se encarga de proporcionar métodos para la clasificación y la reproducción de los colores. Los distintos sistemas de interpretación del color son llamados **espacios de color**. La reproducción un color depende en gran medida al

Comentado [SP7]: Introducción sobre Radiometría y colorimetría (breve)

dispositivo sobre el cual se quiere reproducir y al espacio de color.

Para describir la distribución espectral de la luz hace falta una función continua. Sin embargo para la percepción humana de la luz es suficiente con los tres números R, G y B. (3D Math Primer pag 353).

Existen varios espacios de color, y no necesariamente el RGB es el mejor, pero por varios motivos resulta el más conveniente en campos como la informática y los gráficos. Principalmente el espacio RGB saca ventaja en el hecho de que la mayoría de dispositivos de visualización hoy por hoy están basados en este estándar.

Al final del día lo que interesa es el poder medir la intensidad de la luz en términos de la radiometría. Así el primer término en entrar a escena es la **energía radiante** (*radiant energy*) que no es más ni menos que la energía electromagnética en radiometría. Su unidad en el SI es el *joule* (J). Además interesa conocer la medida para cantidad de energía por unidad de tiempo, es decir la potencia. En el SI, la potencia viene dada por *watt* que es un *joule* por segundo ($1\text{ W} = 1\text{ J/s}$). La potencia en energía electromagnética es llamada **flujo radiante** (*radiant flux*). De esta forma el flujo radiante mide la cantidad de energía que incide, emite o fluye sobre una superficie. Por otro lado si consideramos una superficie de 2 m^2 que emite una cierta cantidad de flujo radiante y otra superficie de 20 m^2 emitiendo la misma cantidad, la superficie con menos área va a parecer mucho más brillante que la de mayor. La densidad de energía por unidad de área es también conocida como **radiosity** cuyas unidades en el SI son el W/m. En ocasiones el término radiosity puede cambiar en función de la dirección del rayo de luz, así pues, si el rayo está incidiendo sobre la superficie es usado el término **irradiancia** (*irradiance*), si el rayo está siendo emitido por la superficie el término usado es *exitance* o **emitancia radiante** (*radiant emittance*) y por otro lado si el rayo está dejando la superficie (después de una reflexión p. ej.) se usa el término radiosity.

Para medir la cantidad de brillo de un rayo de luz (lo cual es importante a la hora de construir un modelo físico) se debe medir la cantidad de flujo por unidad de área proyectada. El término usado es **radiancia** (*radiance*).

Comentado [SP8]: Sacado del libro 3D math, pero no literal ¿Poner la referencia?

Unidades del SI utilizadas en radiometría				
Magnitud física	Símbolo	Unidad del SI	Abreviación	Notas
Energía radiante	Q	julio (unidad)	J	energía
Flujo radiante	Φ	vatio	W	Energía radiada por unidad de tiempo. Potencia.
Intensidad radiante	I	vatio por estereorradián	$W \cdot sr^{-1}$	Potencia por unidad de ángulo sólido
Radiancia	L	vatio por estereorradián por metro cuadrado	$W \cdot sr^{-1} \cdot m^{-2}$	Potencia. Flujo radiante emitido por unidad de superficie y por ángulo sólido
Irradiancia	E	vatio por metro cuadrado	$W \cdot m^{-2}$	Potencia incidente por unidad de superficie
Emitancia radiante	M	vatio por metro cuadrado	$W \cdot m^{-2}$	Potencia emitida por unidad de superficie de la fuente radiante
Radiancia espectral	L_λ o L_ν	vatio por estereorradián por metro cúbico θ vatio por estereorradián por metro cuadrado por hercio	$W \cdot sr^{-1} \cdot m^{-3}$ $W \cdot sr^{-1} \cdot m^{-2} \cdot Hz^{-1}$	Intensidad de energía radiada por unidad de superficie, longitud de onda y ángulo sólido. Habitualmente se mide en $W \cdot sr^{-1} \cdot m^{-2} \cdot nm^{-1}$
Irradiancia espectral	E_λ o E_ν	vatio por metro cúbico θ vatio por metro cuadrado por hercio	$W \cdot m^{-3}$ $W \cdot m^{-2} \cdot Hz^{-1}$	Habitualmente medida en $W \cdot m^{-2} \cdot nm^{-1}$

(Tabla Wikipedia)

Ley de Lambert

La cantidad de irradiancia que produce un rayo al incidir sobre una determinada superficie es determinada por el ángulo con el cual se está produciendo dicha incidencia. Cuando un rayo de luz incide sobre una superficie este genera una cantidad de irradiancia, si la el ángulo formado por el rayo y la superficie es perpendicular, este generará más irradiancia que el generado por ejemplo por un ángulo más cerrado (glazing angle).

IMAGEN DE LAMBERT

Fuentes de luz

Las fuentes de luz son capaces emitir energía a distintas direcciones. Estas se pueden clasificar según distintas características. A continuación se va a realizar una breve descripción de estas en términos de gráficos por computador.

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 para llevar a cabo el efecto de una luz direccional sobre una escena consiste en mantener una dirección asociada a la luz y una energía generada. Este tipo de luces no representan gran complicación a la hora de ser implementadas, no obstante esto puede variar en función de la complejidad del modelo de iluminación que se haya elegido.

Luces puntuales:

Son luces que se representan en un punto del espacio desde el cual se genera luz en todas las direcciones. Este tipo de luces también son llamadas omni light.

El vector dirección (l) y la contribución de irradiación (E_i) de este tipo de luces depende de la posición de la superficie sobre la que se esté calculando la intensidad de luz:

$$r = |P_l - P_s|$$

$$l = (P_l - P_s) / |P_l - P_s|$$

$$E_i = |P_l - P_s| / r^2$$

Donde P_l es la posición de la luz y P_s 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 más complejo.

Funcion de atenuación de OpenGL y DirectX

Comentado [SP9]: Arreglar con formato

$$F_{dis} = 1 / (sc + scr + \sqrt{r^2})$$

Comentado [SP10]: Meter las otras funciones RTR pag 219

Spotlight

Este tipo de luz genera irradiación en una dirección circular pero acotada por un ángulo que define la amplitud del cono que produce la misma. Históricamente, este tipo de luces se pueden calcular con funciones incorporadas en las APIs de dibujo OpenGL y DirectX.

La cantidad de irradiancia es calculada con el vector dirección de la luz \mathbf{s} , y el ángulo θ_s de amplitud. Ya que el cálculo se realiza sobre la superficie de los objetos \mathbf{I} se requiere también del vector dirección opuesto.

$$I_L = \begin{cases} I_{Lmax}(\cos \theta)^{s_{exp}}, & \text{donde } \theta_s \leq \theta_u, \\ 0, & \text{donde } \theta_s > \theta_u, \end{cases}$$

Función de Spotlights de OpenGL

BRDF

Como se ha visto anteriormente, la emitancia radiante es generada por los cuerpos que fuentes de luz. Esta llega en forma de irradiancia a las superficies de los objetos que quedamos dibujar. La manera en que dicha irradiancia es reflejada o tratada por la superficie es definida por lo que se denomina **Bidirectional Reflectance Distribution Function** (BRDF). Una BRDF es una función en términos de radiometría, de cuatro variables reales que definen como la luz es reflejada sobre una superficie opaca.

(Wikipedia).

$$f(x, \hat{\mathbf{w}}_{in}, \hat{\mathbf{w}}_{out}, \lambda)$$

La función toma como parámetros a x como el punto donde se produce la incidencia, a los vectores unitarios $\hat{\mathbf{w}}_{in}$ y $\hat{\mathbf{w}}_{out}$ direcciones desde donde se emite y hacia donde se refleja el rayo respectivamente y a λ como la longitud de onda del rayo de luz en cuestión. Así este último parámetro recuerda que BRDF recibe cualquier tipo de

longitud de onda, es decir, bien sea lo proveniente de fuentes de luz como la reflejada por otros objetos con superficies reflectantes.

Cuando una superficie obedece principios físicos la cantidad de luz reflejada es igual y recibida debe ser igual. De esta forma se establece un principio de reciprocidad entre la luz entrante y la luz saliente:

$$f(x, \hat{w}_1, \hat{w}_2, \lambda) = f(x, \hat{w}_2, \hat{w}_1, \lambda)$$

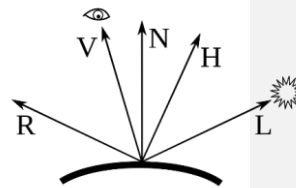
A este principio le llamamos **Reciprocidad de Helmholtz**.

(3D Math primer).

Algunos modelos de BRDF son:

Modelo de Lambertiano: Es aquel que representa superficies perfectamente difusas (matte) mediante una BRDF constante.

Modelo de Phong: Se trata de un modelo empírico propuesto por Bui Tuong Phong en el año 1973. El modelo de Phong define la cantidad de luz saliente de una superficie como la suma de 3 componentes básicas: Componente ambiental, componente difusa y componente especular.

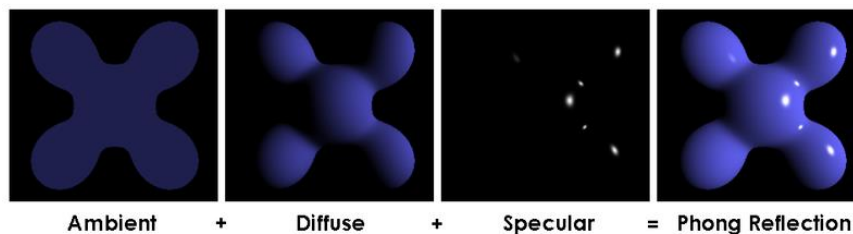


Vectores necesarios para el cálculo del modelo de Phong y de Blinn-Phong.

$$I_p = k_a i_a + \sum_{m \in \text{lights}} (k_d (\hat{L}_m \cdot \hat{N}) i_{m,d} + k_s (\hat{R}_m \cdot \hat{V})^\alpha i_{m,s}).$$

Componente ambiente es calculado como la suma de la contribución ambiental de luz de todas las luces de la escena (i_s). Los componentes i_s y i_d son definidos como intensidades especulares y difusas de las fuentes de luz.

Los valores K son constantes definidas por el material, de modo que: k_s es la constante especular del material que regula la proporción de reflexión especular de la luz entrante, k_d es la constante difusa del material que regula la proporción de luz difusa de la luz entrante y k_a es la constante ambiental del material que regula la cantidad de luz presente en todos los lugares de la escena. Además el exponente α es el término de brillo o *Shininess*. Este exponente es mayor para superficies esmaltadas y menos para superficies más opacas. En definitiva este exponente regula cuan condensado son los brillos especulares del material.



Suma de los componentes del modelos de Phong

(Wikipedia)

Modelo de Blinn-Phong:

El modelo de Blinn-Phong es una optimización en términos de rendimiento del modelo de Phong, por lo tanto se basa en el mismo principio que dicho modelo. La principal peculiaridad del modelo de Blinn-Phong es que este utiliza el vector H para el cálculo del componente especular reemplazando el termino $R \cdot V$ el cual es un por $N \cdot H$ donde N es el vector normal de la superficie y V es el vector que apunta hacia la posición del observador. La mejora radica en que de esta forma no hace falta calcular el vector reflexión $R = 2(L \cdot N)N - L$ lo cual es una operación costosa, en cambio, se calcula el vector $H = \frac{L+V}{|L+V|}$ que es el vector normalizado bisectriz entre L y V .

(Wikipedia)

Este modelo es el elegido por su fácil implementación y por su mencionado rendimiento para satisfacer las necesidades de este proyecto. Así pues, el modelo de

Blinn-Phong fue el implementado aquí en combinación del modelo de Shading de Phong.

Modelo de Cook-Torrance:

Se trata de un modelo más cercano a la realidad física que el modelo de Phong. Es usado para simular la reflectancia especular de distintos materiales. Se trata de un modelo que simula mejor ciertos tipos de materiales del mundo real como son los metales y los materiales con altos índices de reflectividad. El modelo trata cada superficie un conjunto de muchas microfacetas, es decir, pequeñas facetas que reflejan la luz entrante. En superficies ásperas, las pendientes de estas microfacetas varían considerablemente, mientras que en superficies lisas las microfacetas están orientadas en una dirección similar. Este modelo tiene en cuenta fenómenos naturales y muy presentes en la realidad como el factor Fresnel que es un tipo de reflexión que varía en función del ángulo de visión con la superficie.

(<http://ruh.li/GraphicsCookTorrance.html>)

The Rendering Equation

La ecuación de renderizado es una integral en la cual el equilibrio de radiancia saliente de un punto dado es la suma de la radiancia emitida y la radiancia reflejada bajo una aproximación geométrica de la óptica (rol del perceptor de la escena).

(Wikipedia)

Si bien la BRDF nos permite calcular la radiancia saliente de un punto dado en una dirección $\hat{\omega}_{out}$ desde una dirección $\hat{\omega}_{in}$ hacia una dirección $\hat{\omega}_{out}$, la ecuación de renderizado realiza la suma de dicha radiancia más la cantidad de radiancia que es reflejada a cualquier luz que es emitida desde la superficie hacia nuestra dirección.

La forma de la ecuación de renderizado es la siguiente:

$$L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + \int_{\Omega} f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t) L_i(\mathbf{x}, \omega_i, \lambda, t) (\omega_i \cdot \mathbf{n}) d\omega_i$$

Para resolver la ecuación de renderizado hace falta tener en cuenta un tipo de iluminación mucho más compleja y que aporta mucho más realismo al dibujado la cual

es la iluminación global. Si bien la BRFD puede definir la cantidad de radiancia local, que es reflejada en una dirección la cantidad de radiancia es una escena con varios objetos suele ser el resultado de varias radiancias entrantes y salientes producto de las distintas reflexiones que se suceden en la escena. En este se hace insuficiente calcular solo la porción de radiancia que proporciona la BRFD.

Muchos sistemas de renderizado más complejos y realistas intentan resolver esta ecuación y llevar un dibujo más técnico cuantificando de forma real la cantidad total de energía que hay en la escena.

Dado que esta ecuación toca aspectos que este trabajo no puede cubrir, no se hará más énfasis en el tema, pero queda nombrado como algo a considerar si se desea desarrollar sistemas con complejos en un futuro no muy lejano.

Material

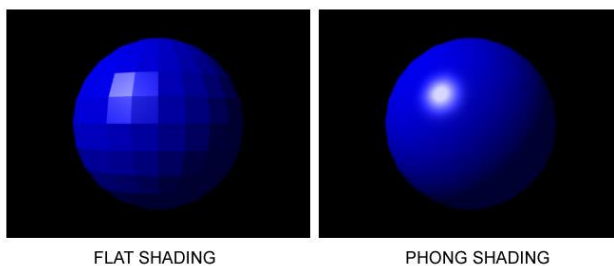
Un material define un conjunto de propiedades que caracterizan y definen un aspecto determinado. Un material es el encargado en generar los datos que luego son transmitidos a la BRFD para computar la cantidad de radiancia de la superficie y posteriormente realizar el cálculo del color de la superficie. La forma en que están modelado los materiales varía en función del modelo de iluminación que se quiera satisfacer. En el caso de este proyecto los materiales están pensados para ser suministrados al modelo de reflexión de Phong. Más adelante se hablará del diseño de los materiales para este proyecto.

Shading

Shading es el proceso de usar una función BRFD para computar la radiance saliente L_0 a lo largo del rayo de visión, \mathbf{v} , basándose en las propiedades del material y de la fuente de luz. Es decir, shading es proceso para darle un color a un objeto basándose en una BRFD y en unas propiedades de materiales.

Existen modelos de shading más comunes que otros, por ejemplo, el modelo usado por este proyecto, el Modelo de Blinn-Phong, que se trata de una versión modificada y optimizada del modelo original de Phong, el cual usa método de interpolación de

normales entre fragmentos, en combinación con el modelo de reflexión de Phong (BRFD). Hablar de Shading y de BRFD no es lo mismo, si bien un BRFD propone matemáticamente una solución para determinados tipos de superficies, el proceso de Shading es el definido como el de obtener un color para un determinado Fragment basándose en una BRFD determinada. También es común confundir el modelo de reflexión de Phong (BRFD) con el Shading de Phong. Si bien el modelo de reflexión de Phong define como se ha de calcular una determinada irradiancia sobre una superficie, el Shading de Phong no es más que una técnica ligada a las condiciones de los gráficos por computador y más a los modelos 3D, en donde mediante interpolación se obtiene valores para la normal en puntos donde no se encuentra un vértice geométrico (los cuales son los únicos que almacenan la información de la normal) y así obtener resultados más continuos visualmente. En contraste a este tipo de Shading se encuentra el Shading Flat (plano) donde la normal es igual para todo un polígono y el resultado es un objeto facetado, o el Shader de Gouraud donde no se interpola la normal pero sí se calcula el color a nivel de Fragment y donde se obtienen resultados menos facetados aparentemente, pero evidentemente facetados para modelos con bajos polígonos y en zonas donde la luz especular está presente. Como ha de esperarse, los modelos con mejor resultado visual son mucho más costosos en términos de tiempo de computación y suelen requerir de un procesado extra en la etapa de Fragment Shader (etapa en el proceso de dibujado donde se calcula el color del pixel a partir de un pequeño programa – Fragment Shader -).



Dibujando en 3D.

Hoy en día las técnicas para obtener una imagen de objetos en 3d están prácticamente depuradas y extendidas, incluso podríamos hablar de estándares. Esta práctica, en gran medida, y como muchas otras cosas en el campo de la informática, ha crecido basándose en las propuestas de personas o grupo de personas que han sabido encarar de la mejor forma posible el problema en cuestión. Resultado de esto, hoy en día tenemos una forma bastante competente de hacer las cosas para conseguir pintar objetos en 3D. Paralelamente, y muy importante, ha sido el rol que los fabricantes de hardware para gráficos en 3d han tenido en esta evolución y bien, en gran medida, s como la tecnología software y hardware se han sabido complementar para crear arquitecturas bien definidas y altamente extendidas.

Pintar objetos en 3d hoy en día (sin tener en cuenta modelos de iluminación) es la una combinación de varios factores. En primer lugar deben existir dispositivos para el dibujo y para los cálculos que se realizan hasta conseguir una imagen plana. Dadas estas dos premisas hoy en día nos encontramos con un elemento fundamental: las APIs de dibujo. OpenGL y DirectX son un conjunto de funciones a bajo nivel que permiten a los programadores el poder abrir contextos visuales y crear imágenes en 3D de objetos, entre otras cosas. OpenGL es un proyecto open source multiplataforma, mientras que DirectX es un software propietario y es desarrollado por la compañía Microsoft e incorporado en dispositivos con sistema operativo Windows. Ambas son herramientas que sirven para dibujar. Dicho esto, estas APIs están altamente cohesionadas con las arquitecturas hardware actuales. Su forma de actuar es razonablemente entendible si se mira desde el punto de vista hardware y se atiende a la premisa de que quien logra altos índices de velocidad de computación – gracias a su arquitectura paralela SIMD o SIMT (NVidia) – son las tarjetas gráficas actuales.

Aprovechando las instrucciones en paralelo de estas tarjetas podemos por ejemplo, realizar el cálculo de millones de píxeles de forma eficaz y así determinar el color que le corresponde dadas ciertas reglas antes comentadas como la luz, el material etc.

Aunque sean dos APIs distintas ambas manejan relativamente el mismo “protocolo” para dibujar en pantalla. A este protocolo se le denomina pipeline (tubería). Tanto el pipeline de OpenGL como el DirectX son similares es varias etapas, incluso en

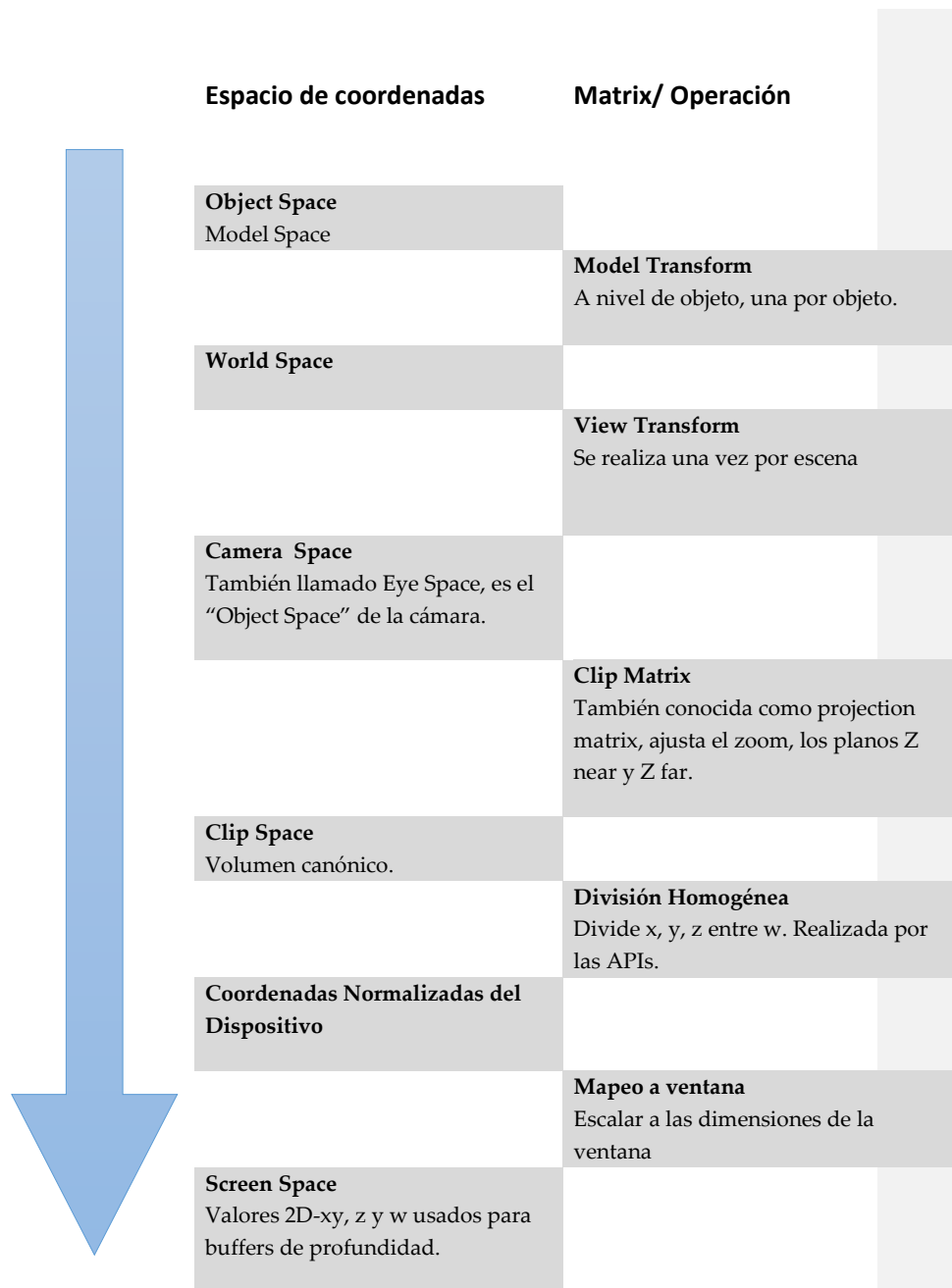
ocasiones usando términos distintos para denominar cosas esencialmente similares (algo nada nuevo). OpenGL es la API usada para el desarrollo de este proyecto.

A continuación se hará una breve descripción del proceso llevado a cabo para dibujar mallas de vértices en pantalla. Esto es así porque se ha considerado que este proceso determinado no representa un hito a superar en este proyecto ya que resulta algo trivial una vez superado. Además en cualquier fuente de información relacionada con el tema de gráficos se presentan de una mejor y más detallada manera las explicaciones referentes a este tópico.

Comentado [SP11]: Pie de página explicando que es

Comentado [SP12]: Suena muy rudo?

Mediante una valiosa herramienta, el álgebra lineal, se tiene la capacidad de generar imágenes planas haciendo uso de distintas transformaciones geométricas. Estas transformaciones constituyen una forma de conseguir proyectar diferentes objetos sobre un plano abstracto como lo es una pantalla o monitor. Para llegar a esto, primero se han de pasar por diferentes etapas y ahí donde los pipelines de cada API entran en juego, dando la posibilidad de controlar lo que pasa en cada etapa. Las transformaciones básicas como la rotación, el escalado, la translación, la proyección etc. son llevadas a cabo mediante operaciones matemáticas basadas en matrices. De esta forma podemos aplicar diferentes transformaciones a, p. ej., un vértice de una malla 3D simplemente haciendo uso de una matriz preparada para dicha transformación.



Conversión de coordenadas de vértices a través del graphic-pipeline.

OpenGL pipeline

OpenGL tiene un pipeline denominado OpenGL Pipeline el cual posee a grandes rasgos las mismas etapas que proceso descrito anteriormente. Tanto DirectX como OpenGL dan la libertad al desarrollador de controlar determinadas fases. Dichas fases son implementadas en términos de programación por pequeños programas (*Shaders*) escritos en un lenguaje de programación diseñado para este tipo de tareas. El lenguaje de programación de OpenGL es llamado GLSL (OpenGL Shading Language). Las etapas programables en el pipeline de OpenGL son:

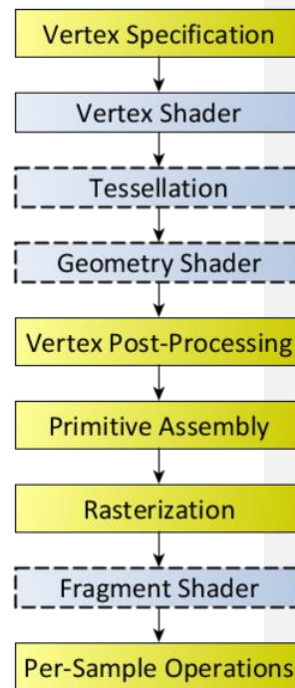
Vertex Shader: Se procesan operaciones básicas sobre cada vértice, estas operaciones suelen ser llevar los vértices a Clip Space mediante transformaciones y también existe la posibilidad de que el usuario genere información de salida que puede ser recibida en posteriores etapas como la dirección de las luces, o las normales.

Tessellation: Las primitivas pueden ser teseladas usando dos Shader, Tessellation Control Shader (TCS) que define la cantidad de teselación y Tessellation Evaluation Shader (TES) que se encarga de la interpolación y de otras operaciones definidas por el usuario.

Geometría Shader: Esta fase recibe un número determinado de primitivas y puede devolver cero (descartar vértices) o más primitivas. El Shader que controla esta fase es llamado Geometry Shader.

Fragment Processing: Los datos de cada fragmento en el estado de rasterización son procesados por el Fragment Shader. La salida de esta etapa da como resultado una lista de colores que se almacenan en el framebuffer, en el depthBuffer (valores de profundidad) y en el stencilBuffer (el stencil value).

Texturas



OpenGL Pipeline, los cuadros con líneas discontinuas representan etapas opcionales

Comentado [SP13]: Definirlos en pie

Las texturas son una poderosa herramienta en el campo de los gráficos, no solo para ser aplicadas sobre los objetos, un uso más común, sino también para realizar tareas más diversas como cálculos de iluminación basada en mapas de luz (texturas con información para el cálculo de luz) o el cálculo de reflexiones con cube maps. Algunos mapas de texturas son creadas de antemano por programas de edición de imágenes como Photoshop mientras que otros son generados mediante procedimientos en tiempo de carga o de edición (precalculados). Ejemplos de esto último son los cube maps generados automáticamente para el cálculo de reflexiones donde el color de la radiancia entrante es una porción calculada proporcionada por el cube map, de esta forma se pueden conseguir efectos como las reflexiones propias de un espejo, o las de un material altamente reflectivo.



Reflexiones calculadas con environment cube maps.

<https://developer.valvesoftware.com/wiki/Cubemaps>

No obstante, el uso de mapas de texturas que más interesa para la finalidad de este proyecto son las que proporcionan información para establecer parámetros de los materiales. Algunos ejemplos de estas texturas son los mapas de textura difusos, mapas de normales, mapas especulares, mapas de valores de transparencia (alpha maps), e incluso Shininess maps.

Mapas de textura difusos.

Estos mapas contienen la información del color difuso del material, en ocasiones los artistas suelen incluir valores de iluminación en este tipo de materiales (sombreado artificial o *Ambient Occlusion*).

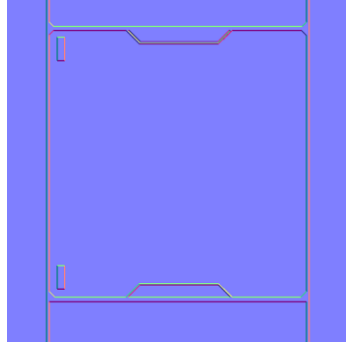


Mapas de textura normales.

La técnica consiste en almacenar la dirección de la normal en una imagen.

Posteriormente el valor mapeado es leído para cada Fragment y ese valor de la normal es usado para el cálculo del color con el modelo de shading que corresponda. Una particularidad de esta técnica es que los valores almacenados en la textura están en términos del espacio tangente (tangent space). Es por este motivo que los cálculos para obtener el color son realizados en dicho espacio de coordenadas, esto obliga a que los vectores dirección de la luz, dirección del observador y todo aquellos que entren en juego a la hora de realizar el cálculo estén en espacio tangente. Para llevar un objeto a espacio tangente lo que se suele realizar es componer una matriz TBN calculada a partir de las componentes normal, tangencial y binormal de los vértices y posteriormente realizar la multiplicación con esta matriz para hacer el cambio de coordenadas.

En definitiva, si se quiere usar esta técnica, los cálculos de la iluminación han de hacerse en espacio tangencial ya que es la forma en la que las estos mapas guardan la información de la normal.

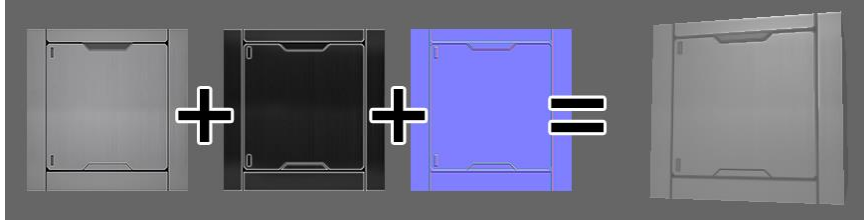


Mapas de texturas Especulares

Este tipo de mapa almacena la intensidad del valor especular del material. Este mapa suele estar en escala de grises, aunque existe la posibilidad que cuente con color lo que produciría brillos de color.



Si asignamos los mapas correspondientes al modelos difuso, especular y ambiental con perturbación de las normales obtenemos un resultado así:



Physically Based Rendering.

Actualmente existe una tendencia por la cual está pasando la industria de los videojuegos. La mayoría de grandes empresas desarrolladoras empezaron a darle una nueva aproximación a sus sistemas de renderizado, o por lo menos, un más enfoque mucho más científico. Propiciado por los avances en temas de hardware, se han podido modelar sistemas mucho más complejos y realistas que los antiguos. Así pues se ha empezado a popularizar el termino **Physically Based Rendering (PBR)**.

El PBR no es algo nuevo, en el mundo del renderizado off-line ha estado presente desde hace ya bastante. No obstante, es un tópico nuevo para renderizado en tiempo real ya que hasta hace muy poco no se podían realizar gastos computacionales considerables sin afectar el frame rate de dibujado, y es ahora cuando al parecer, se han podido generar nuevas técnicas (aun distantes a las del renderizado off-line) que permiten de cierta forma modelar sistemas de renderizado donde se conservan magnitudes físicas como lo son la cantidad de energía radiante que se encuentra en una escena.

A día de hoy no existe un estándar de PBR, inclusive cada desarrollador está llevando los conceptos como bien considera. No obstante en conferencias de renombre como SIGGRAPH (Special Interest Group on GRAPHics and Interactive Techniques) se exponen siempre enunciados y trabajos con el ánimo de reflejar un criterio unificado y conseguir así hacer del PBR un estándar único. Cada quien intenta realizar el sistema más competitivo posible, no obstante, empiezan a surgir determinadas similitudes y conceptos comunes. Un ejemplo de la necesidad de estas semejanzas radica en hechos

como que los artistas que trabajan en diferentes proyectos posean un método de trabajo igual sea cual sea el motor al que se destinará determinado Asset (un objeto o recursos de un proyecto de tipo videojuego). De no ser así, el coste de tener artistas entrenados para determinado motor podría ocasionar problemas.

Características de un sistema PBR.

Un sistema de PBR tiene como objetivo principal crear un dibujado fotorrealista tan preciso como sea posible. En PBR un objeto no puede reflejar más energía de la que recibe (conservación de la energía). A continuación se presentan algunos parámetros que pueden alimentar un PBR y sus diferencias con términos tradicionales.

El término “difuso” es sustituido por un término más preciso y fiel llamado Albedo. Un mapa de albedo define el color de luz difusa. Una de las grandes diferencias entre un mapa Albedo y un tradicional mapa difuso es la falta de luz direccional o “ambient occlusion” (AO. Técnica usada para aproximar un efecto de iluminación ambiental). Esto es debido a que bajo ciertas condiciones la luz direccional que incluye el AO resultan incorrectas. Por eso los valores de AO deben ser añadidos como nuevo parámetro de material y no combinado en el canal difuso como se hacía tradicionalmente.

Microsurface (micro superficie) define cuan áspero es la superficie. A este punto es cuando los principios de la conservación son afectados por la micro superficie del material. Superficies más ásperas mostraran reflexiones especulares más amplias mientras que superficies más lisas mostraran brillos especulares más definidos asemejándose al aspecto de un espejo.

Reflectividad es el porcentaje de luz que una superficie refleja. Todos los tipos de reflectividad están incluidos en este parámetro: especular, metalness (metalicidad), etc. La reflectividad define como de reflectivo es el material cuando es observado de frente, mientras que el parámetro Fresnel define como de reflectiva es la superficie en ángulos cerrados (glazing angles).

Algunos motores gráficos como UE4 nombran este parámetro como Metalness.

Fresnel define la cantidad de reflectividad de la superficie en ángulos cerrados. La mayoría de materiales debería establecer este parámetro a 1, pues la mayoría de objetos tienen reflectividad del 100% en ángulos cerrados.

Otros mapas como mapas de Ambient Occlusion y de Cavity (cavidad) complementan el modelo.

Una de las grandes ventajas de trabajar con PBR es la que la definición de un material resulta mucho más intuitiva para los artistas, viéndose favorecido su flujo de trabajo de forma considerable.

(<http://www.marmoset.co/toolbag/learn/pbr-practice>)

Estado del arte

Editor de materiales

Un editor de materiales es una interfaz que permite de alguna forma editar y suministrar valores para los distintos parámetros del modelos de iluminación que se este empleando. En la mayoría de enfoques un material es una representación abstracta que se hace visualmente evidente cuando es aplicada a una malla.

El término “editor de materiales” es un concepto bastante extendido en la actualidad en contextos de gráficos por computador y en motores gráficos. Si bien existen gran variedad de herramientas que permite suministrar los parámetros que caracterizan un determinado material, 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 como otros efectos más avanzados (desenfoque de movimiento basado en mapeado de movimientos, la iluminación volumétrica etc.). Todos ellos hacen parte del dibujado final de la escena pero no así de las propiedades físicas del material.

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 específico, manejando conceptos y parámetros más genéricos y que se aproximan más al “estándar” más extendidos 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 todo 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.

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 procesos internos son radicalmente distintos, 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.

Una tendencia que se está empezando a extender bastante es la de pasar del clásico editor de materiales con casillas de texto y campos para las texturas, a un nuevo enfoque mucho más diverso y con más potencial. Se trata de las interfaces basadas en nodos. El principio básico en la mayoría de casos, es el aprovechamiento de operaciones realizadas sobre los colores directamente, en combinación con las texturas y las fuentes de datos como los vectores, creando conexiones entre nodos y otros nodos que representan operaciones. En este punto, el editor de materiales hace referencia más a un creador de Fragment Shaders (de alguna forma) que de simplemente asignar unos valores a unas entradas.

La potencia que proporciona es notable. Constituye un sistema VPL (Visual programming Language) que se aprovecha de las bondades de estos sistemas como la capacidad de manipular elementos gráficamente para generar programas, en este caso, para generar un Shader. Los nodos en combinación con operaciones generan nuevos resultados y al final lo que se obtiene es una pieza de código capaz de definir un material.

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.

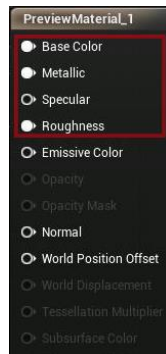
Este editor permite la creación de Shaders para aplicarlos sobre alguna superficie geométrica determinada. Dicho editor viene incorporado en el motor gráfico UE4 y hace parte de uno de sus subsistemas.

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.

El editor de materiales de UE4 permite la creación de una variedad considerable de nodos que bien representan operaciones las cuales encapsulan 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 que no está especializada en programación.

UE4 utiliza un modelo de shading PBR. 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 de aspereza del material. Un material con un alto nivel Roughness refleja luz a más 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 parcialmente en 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 oclusiones o sombreado a pequeña escala, representadas en el mapa de normales (mapa de 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 Occlusion) 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:

$$\text{BaseColor} = \text{Cavity} * \text{OldBaseColor}, \text{Specular} = \text{Cavity} * 0.5$$

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 grande.

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.

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.

Glossiness: 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, 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 de mapas más utilizados 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 UV también están a disposición del usuario para su asignación.

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 píxeles 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 píxel, 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.

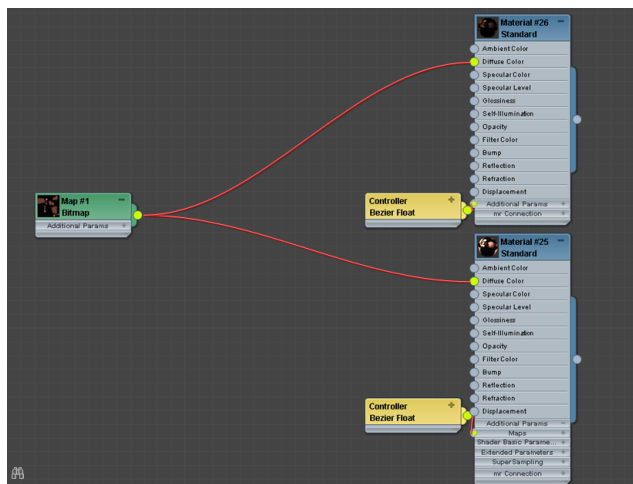
Su interfaz de usuario deja de ser la convencional interfaz compacta para pasar a ser una más actualizada respecto a las tendencias actuales de edición de materiales. Dicha interfaz está basada en nodos, lo cual la hace mucho más adecuada a las demandas y flujos de trabajo actuales de este tipo de edición.

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.



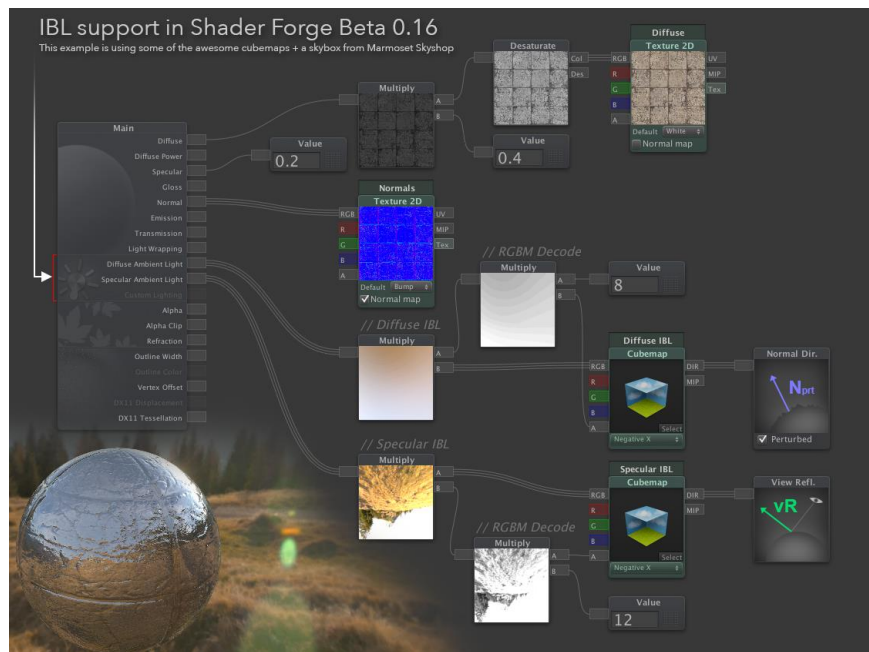
Dos materiales compartiendo y reutilizando una textura representada por un nodo de tipo Bitmap

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.

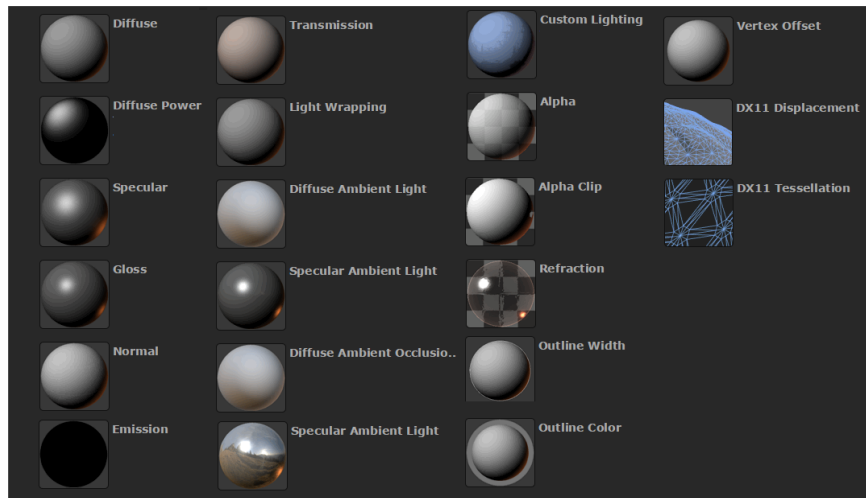
Además Shader Forge tiene la capacidad de poder trabajar con *Image Based Lighting* lo que le permite trabajar con técnicas como la de *Cube Maps* para visualizar la forma en que se iluminan los objetos, y generar valores para *Diffuse Ambient Light* y *Specular Ambient Light*. Gracias a esto Shader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

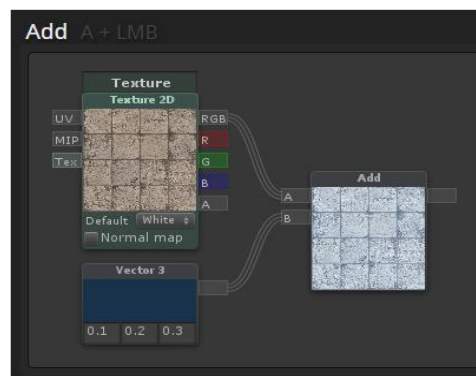
Su flujo de trabajo intuitivo es perfecto para artistas quienes no está muy habituados a programar ni a generar shaders mediante programación de texto.

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 Substances
- Dithered Supersampled Transparency

Discusión

Comentado [SP14]: Comentar los pros y cons, y decir que se mejora con este trabajo. Y una tabla para comparar

9. 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 es una similar a una metodología basada en Sprints pequeños de unos 7 días por lo general, en donde se presentaban nuevas funcionalidades cada vez.

Estrictamente no se ha llevado a cabo el seguimiento de una metodología de desarrollo como tal. Desarrollo en algunos momentos conto con periodos de revisión más prolongados debido a inconvenientes que surgieron para acudir a las visitas de evaluación y revisión.

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 en un principio fue la de elegir una plataforma académica y libre como lo era Google Code. Una buena ventaja de esta decisión es que Google Code se podía basar en Subversion (un tipo de control de versiones) lo cual era muy bien recibido debido a la experiencia adquirida en pasado proyectos sobre esta plataforma.

El desarrollo se llevó a cabo sobre este repositorio durante la primera etapa de existencia. Durante esta primera etapa de desarrollo (mediados de Marzo de 2015) Google anuncio el cierre de Google Code y el final del servicio de reposición. Debido a esto la necesidad de migrar el proyecto se hizo casi obligatoria.

Tras un tiempo de investigación y consulta, se decidió usar GitHub. El servicio prestado por GitHub es gratuito, lo que se ajusta perfectamente a las mis condiciones económicas. Su herramienta de control de versiones es más que suficiente para llevar a cabo el desarrollo del proyecto de forma íntegra. El pequeño coste es la curva de aprendizaje para usarlo, aunque, como todo, con práctica se consiguió. Además GitHub es realmente fácil de usar, con lo que la curva de aprendizaje se reduce bastante.

La aplicación para realizar los commits es el **cliente de GitHub de Windows**. En él es fácil crear un commit y subir los cambios sincronizando la versión local con la del servidor.

Al final de todo no se sabrá cuanto tiempo vaya a estar el proyecto bajo el repositorio. Se hará todo lo posible para que dure el máximo de tiempo online, pero como todo puede pasar no se puede asegurar que el repositorio este vigente siempre. No obstante, si algo llegase a ocurrir, se procurará dejar toda la información del nuevo paradero del proyecto en caso de que se mueva o se migre.

A día de hoy (Agosto de 2015) la dirección para encontrar el proyecto en GitHub es:

<https://github.com/s4ntia60/spc-editor/>

Herramientas de desarrollo

Visual Studio

Como entorno de desarrollo se utilizó Visual Studio (VS) en su versión 2013. Visual Studio permite el desarrollo de aplicaciones bajo sistemas operativos Windows con lenguajes de programación como C++ y con la posibilidad de integrar librerías y frameworks de forma fácil. Cabe también destacar que la potencia ofrecida, el número de herramientas y más que todo la facilidad para integrar desarrollo de otro tipos de proyectos (como scripts para motores gráficos o desarrollos web) hacen de VS uno de los más potentes –y universales - IDEs del mercado.

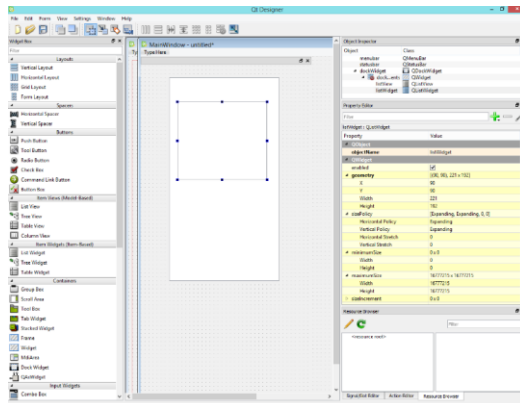
Qt

Qt es un frameworks multiplataforma de desarrollo de aplicaciones ampliamente usado para desarrollo de aplicaciones que pueden ser ejecutadas en diferentes sistemas sin la necesidad de cambiar el código o cambiarlo de una manera radical.

Qt fue la herramienta elegida para crear la interfaz gráfica del proyecto. Qt permite crear contextos para el dibujado 3D en OpenGL, permite crear interfaces de usuario típicas como controles de texto e inputs numéricos, pero además, lo más significativo para este proyecto es que permite generar un espacio donde a partir de elementos visuales como cajas y líneas, se puede crear el editor basado en nodos requerido.

La instalación típica incluye un IDE propio de Qt y también herramientas para el diseño de ventanas y formularios. Esta instalación fue evitada dado que el IDE de desarrollo es Visual Studio. Los creadores de Qt dejan a disposición de los desarrolladores la opción de crear las mismas aplicaciones que se pueden crear desde su propio IDE, en Visual Studio. Para ello se ha publicado un plugin para VS. Este plugin realiza las configuraciones pertinentes para crear un proyecto Qt. Para usar configurar un proyecto Qt en VS principalmente se debe seleccionar el lugar de instalación de Qt en el disco para establecer la versión con la que se compilara y para generar el código enlazando desde esta ubicación. Al momento de compilar, el plugin crea toda una rutina para generar archivos necesarios para la ejecución. Estos archivos pueden ser ficheros de interfaces de usuario (.ui) o ficheros *moc_* donde se definen propiedades de algunos controles y meta elementos que Qt gestiona para funcionalidades como Signals/Slots (se trata de una aproximación de gestión eventos mediante funciones).

Además, para contribuir con la tarea de trabajar con VS, Qt permite la instalación del diseñador de ventanas y formularios (*QtDesigner*) de forma independiente, donde lo único que se generan son archivos .ui.



Qt Designer

Como consecuencia de usar Qt para el desarrollo, la curva de aprendizaje fue un nuevo desafío a superar. El mayor dolor de cabeza de todo el proyecto fue la implementación del editor de nodos como tal. Afortunadamente en la etapa de investigación y búsqueda de un elemento similar se encontró con el editor de bloques de **Stanislaw Adaszewski**. Este editor era una implementación de Qt usando las clases de dibujado como QGraphicsView y QGraphicsScene. La implementación cuenta con los elementos básicos como nodos, inputs y conexiones. Además la captura de eventos p. ej. para realizar las conexiones mediante el uso del ratón o el cambio de posiciones mediante la acción de clic y arrastre del ratón. En definitiva este pequeño editor fue de gran ayuda en ausencia de experiencia con Qt y como ilustración para a partir de la misma desarrollar y modelar el sistema como se requiere.

El código de Stanislaw está bajo una licencia de tipo BSD. Las reproducción, alteración y publicación están permitidas por lo que usar/modificar este código no supone ningún problema. Son pocas las restricciones que esta licencia obliga, como incorporar un cabecero en cada fichero que contenga su código donde se especifica los estatutos de dicha licencia.

<http://alcoholic.eu/qnodeseditor-qt-nodesports-based-data-processing-flow-editor/>

GLEW (OpenGL Extension Wrangler Library)

GLEW es una librería multiplataforma que permite llamar a las funciones de OpenGL. Aunque OpenGL no es una librería como tal, pues está instalado y habita en las tarjetas gráficas, es necesario realizar la inclusión de una librería como GLEW para realizar consultas y cargar las extensiones de OpenGL. Sin GLEW no podríamos llamar a las funciones de OpenGL.

Wikipedia

Assimp.

Se trata de una librería que permite la importación de varios de los elementos usados en gráficos. Principalmente se ha usado para importar las mallas usadas en la escena 3D del proyecto. Su principal ventaja es que puede importar mallas desde varios tipos de ficheros como .obj, .3ds, etc. Además assimp es de uso libre.

<http://assimp.sourceforge.net/>

GLM (OpenGL Mathematics)

GLM es una librería matemática cuya principal ventaja es que permite el uso de tipos como vectores y funciones lineales de una forma muy parecida a la que GLSL los implementa. Además está ampliamente extendida y aunque no es un estándar, suele ser un elemento esencial en el desarrollo cotidiano de OpenGL.

NotePad++

Se trata de un editor de texto plano para desarrollo de software bastante ligero. Su uso principalmente fue para retocar el código los shaders y su edición.

Photoshop y Gimp.

El uso de Photoshop fue principalmente para realizar los gráficos y texturas del proyecto, no solo para mapas de texturas sino también para crear esquemas y cuadros conceptuales. El uso de Gimp se basó en a partir del plugin *Normalmap* crear algunos mapas de normales usados en los ejemplos y en las pruebas del proyecto.

3D Max 2016

Se trata de una herramienta para el modelado en 3D. Se usó principalmente para generar mallas y modelos en formato .obj las cuales se podían importar al proyecto para su visualización. Además en este programa se pueden modificar las coordenadas de textura de los modelos aplicando funciones de proyección distintas en caso de que las predeterminadas no satisficieran las necesidades eventuales.

10. Modelo Propuesto

Especificación y Análisis.

A continuación se realizará una descripción completa del sistema, repasando aspectos como requisitos funcionales y no funcionales, interfaz gráfica y otros comportamientos. Además se hará un estudio de viabilidad y una planificación que se seguirá durante el desarrollo.

Antes de empezar a describir en detalle el sistema se hará una definición del mismo en términos generales.

Descripción general del sistema.

El Sistema es una aplicación de escritorio que permite al usuario poder manipular una serie de nodos para crear un árbol, las conexiones entre los nodos alimentan unas entradas básicas, especificadas como parámetros del modelo de iluminación a desarrollar. Una vez hecho esto, el sistema debe resolver las conexiones del árbol y generar un Shader (Fragment) que se compilará y se usará para dibujar. Por consiguiente, debe existir un visor en 3d en el cual los resultados de las modificaciones que se van realizando se hagan notorias.

Cada nodo del árbol representa una operación o una fuente de datos. Todos tienen al menos una salida y los nodos que requieren de algún argumento para realizar operaciones tienen entradas para dichas operaciones. Todos los nodos terminan alimentando de alguna forma u otra alimentando el nodo principal **MainNode**. Los nodos que no hacen parte del árbol (aislados) no se toman en cuenta para la creación del Shader.

El sistema proporciona todas las interfaces necesarias para el suministro de datos por parte del usuario como son datos de vectores, imágenes y valores de los distintos nodos.

Además de las interacciones con los nodos, también están contempladas las realizadas sobre el visor 3D. Estas incluyen girar el modelo de muestra, acercar o alejar la cámara y abrir o cerrar el zoom de la cámara. Además se pueden ajustar parámetros de la cámara como el color y la intensidad. Otra opción a disposición del usuario es la de poder cambiar el modelo a una serie de modelos proporcionados como muestras de visualización.

A continuación se hace un análisis a más profundidad de cada apartado del sistema.

Editor de nodos.

El editor de nodos será un área de trabajo donde se pueden manipular cajas que representan los nodos de funciones dentro del Fragment Shader que se generará.

Nodos

Los nodos son abstracciones visuales de operaciones existentes en OpenGL o creadas de forma personalizada. Cada nodo devolverá un valor capaz de ser interpretado por el nodo que solicite dicha salida (esté conectado a él). Además de devolver el resultado principal de sus operaciones, los nodos deben devolver de forma individual las componentes de color que lo forman. Así si un nodo tiene una salida del tipo vec4, éste contendrá una salida para obtener el valor compuesto del vec4 pero además tendrá otras 3 salidas para las componentes R, G, B y A. La principal motivación para esto es que devolviendo cada componente se puede lograr efectos muchísimos más variados como la emascaración de colores usando texturas (Texture Masking) o la obtención de nuevos materiales sin nuevas fuentes de datos. Esta característica dota de potencial a cualquier editor de materiales basado en nodos.

Al iniciar el sistema, se contará con un nodo principal el cual contiene solo inputs (MainNode). Este nodo contiene los inputs básicos a definir para construir un material nuevo. Estos inputs son:

Color:

Suministra el componente difuso del modelo.

SpecularColor

Suministra el componente especular del modelo.

Shininess

Regula el exponente de especularidad dentro del modelo.


Normal

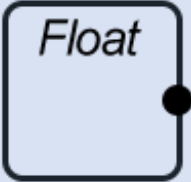
Suministra la dirección de la normal para realizar el Shading.

Alpha

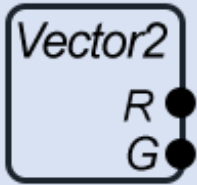
Sirve como mascara para modificar el valor de alpha. Y asignar transparencia.

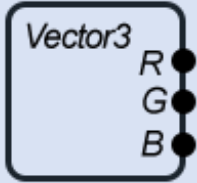
A continuación se hace una descripción de cada nodo disponible en el sistema donde se especifica el diseño que debe tener, las entradas y salidas que poseen y para qué sirven además de la operación que realizan:

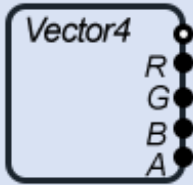
Nombre:	Const Value Node
Diseño	
Descripción:	Recibe los valores finales para los parámetros del modelo de iluminación.
Inputs:	Color, specular, shininess, normal y alpha.
Outputs:	No posee outputs

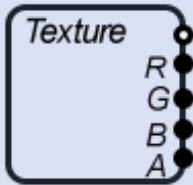
Nombre:	Const Value Node
Diseño	

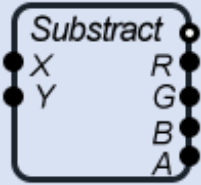
Descripción:	Almacena un valor único en forma de float. Este nodo debe tener relacionado un campo numérico para la introducción del dato como tal.
Inputs:	Ninguno.
Outputs:	Devuelve un número en coma flotante (float).

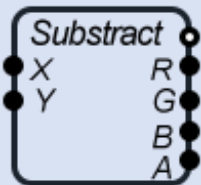
Nombre:	Vector 2 Node
Diseño	
Descripción:	Almacena un vector 2D con componentes RG.
Inputs:	Ninguno.
Outputs:	R: Devuelve el componente de rojo del vector. G: Devuelve el componente de verde del vector

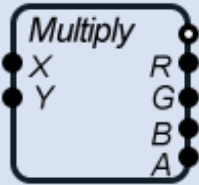
Nombre:	Vector 3 Node
Diseño	
Descripción:	Almacena un vector 3D con componentes RGB.
Inputs:	Ninguno.
Outputs:	R: Devuelve el componente de rojo del vector. G: Devuelve el componente de verde del vector B: Devuelve el componente azul del vector.

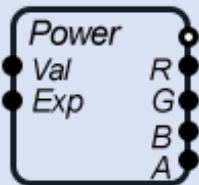
<i>Nombre:</i>	Vector 4 Node
<i>Diseño</i>	
<i>Descripción:</i>	Almacena un vector 4D de componentes RGBA.
<i>Inputs:</i>	Ninguno.
<i>Outputs:</i>	R: Devuelve el componente de rojo del vector. G: Devuelve el componente de verde del vector B: Devuelve el componente azul del vector. A: Devuelve el componente alpha del vector.

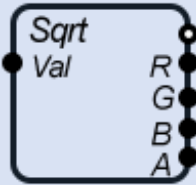
<i>Nombre:</i>	Texture Node
<i>Diseño</i>	
<i>Descripción:</i>	Almacena de una textura.
<i>Inputs:</i>	Ninguno (Se ha de introducir la ruta de la textura)
<i>Outputs:</i>	O - Devuelve el vector 4D que componen los valores RGBA de la textura. R: Devuelve el componente de rojo del vector. G: Devuelve el componente de verde del vector B: Devuelve el componente azul del vector. A: Devuelve el componente alpha del vector.

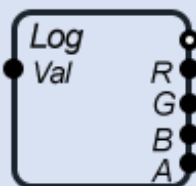
<i>Nombre:</i>	Add
<i>Diseño</i>	
<i>Descripción:</i>	Calcula la operación de suma.
<i>Inputs:</i>	X y Y: Operandos para realizar la suma.
<i>Outputs:</i>	<p>O - Devuelve el vector 4D que componen los valores RGBA de la operación.</p> <p>R: Devuelve el componente de rojo del vector.</p> <p>G: Devuelve el componente de verde del vector</p> <p>B: Devuelve el componente azul del vector.</p> <p>A: Devuelve el componente alpha del vector.</p>

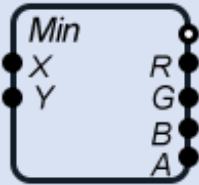
<i>Nombre:</i>	Subtract
<i>Diseño</i>	
<i>Descripción:</i>	Calcula la operación de resta.
<i>Inputs:</i>	X y Y: Operandos para realizar la resta.
<i>Outputs:</i>	<p>O - Devuelve el vector 4D que componen los valores RGBA de la operación.</p> <p>R: Devuelve el componente de rojo del vector.</p> <p>G: Devuelve el componente de verde del vector</p> <p>B: Devuelve el componente azul del vector.</p> <p>A: Devuelve el componente alpha del vector.</p>

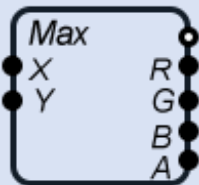
<i>Nombre:</i>	Multiply
<i>Diseño</i>	
<i>Descripción:</i>	Calcula la operación de multiplicación.
<i>Inputs:</i>	X y Y: Operandos para realizar la multiplicación.
<i>Outputs:</i>	<p>O - Devuelve el vector 4D que componen los valores RGBA de la operación.</p> <p>R: Devuelve el componente de rojo del vector.</p> <p>G: Devuelve el componente de verde del vector</p> <p>B: Devuelve el componente azul del vector.</p> <p>A: Devuelve el componente alpha del vector.</p>

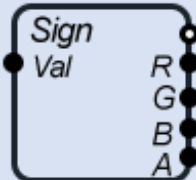
<i>Nombre:</i>	Power
<i>Diseño</i>	
<i>Descripción:</i>	Eleva un valor a la potencia especificada.
<i>Inputs:</i>	<p>Val: Base de la potencia.</p> <p>Exp: Exponente.</p>
<i>Outputs:</i>	<p>O - Devuelve el vector 4D que componen los valores RGBA de la operación.</p> <p>R: Devuelve el componente de rojo del vector.</p> <p>G: Devuelve el componente de verde del vector</p> <p>B: Devuelve el componente azul del vector.</p> <p>A: Devuelve el componente alpha del vector.</p>

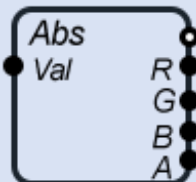
<i>Nombre:</i>	Sqrt
<i>Diseño</i>	
<i>Descripción:</i>	Realiza la operación de raíz cuadrada.
<i>Inputs:</i>	Val: Valor por el que se calculara la raíz.
<i>Outputs:</i>	<p>O - Devuelve el vector 4D que componen los valores RGBA de la operación.</p> <p>R: Devuelve el componente de rojo del vector.</p> <p>G:Devuelve el componente de verde del vector</p> <p>B: Devuelve el componente azul del vector.</p> <p>A: Devuelve el componente alpha del vector.</p>

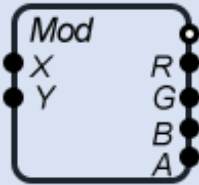
<i>Nombre:</i>	Log
<i>Diseño</i>	
<i>Descripción:</i>	Realiza la operación de logaritmo.
<i>Inputs:</i>	Val: Valor con el que se calculara el logaritmo.
<i>Outputs:</i>	<p>O - Devuelve el vector 4D que componen los valores RGBA de la operación.</p> <p>R: Devuelve el componente de rojo del vector.</p> <p>G:Devuelve el componente de verde del vector</p> <p>B: Devuelve el componente azul del vector.</p> <p>A: Devuelve el componente alpha del vector.</p>

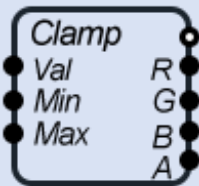
<i>Nombre:</i>	Min
<i>Diseño</i>	
<i>Descripción:</i>	Compara dos valores y devuelve el menor.
<i>Inputs:</i>	X y Y: Valores a comparar.
<i>Outputs:</i>	<p>O - Devuelve el vector 4D que componen los valores RGBA de la operación.</p> <p>R: Devuelve el componente de rojo del vector.</p> <p>G: Devuelve el componente de verde del vector</p> <p>B: Devuelve el componente azul del vector.</p> <p>A: Devuelve el componente alpha del vector.</p>

<i>Nombre:</i>	Max
<i>Diseño</i>	
<i>Descripción:</i>	Compara dos valores y devuelve el mayor.
<i>Inputs:</i>	X y Y: valores a comparar.
<i>Outputs:</i>	<p>O - Devuelve el vector 4D que componen los valores RGBA de la operación.</p> <p>R: Devuelve el componente de rojo del vector.</p> <p>G: Devuelve el componente de verde del vector</p> <p>B: Devuelve el componente azul del vector.</p> <p>A: Devuelve el componente alpha del vector.</p>

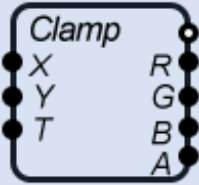
<i>Nombre:</i>	Sign
<i>Diseño</i>	
<i>Descripción:</i>	Devuelve el signo del valor entrante, -1 si es negativo, 1 si es positivo, y 0 si es 0.
<i>Inputs:</i>	Val: Valor a calcular el signo.
<i>Outputs:</i>	<p>O - Devuelve el vector 4D que componen los valores RGBA de la operación.</p> <p>R: Devuelve el componente de rojo del vector.</p> <p>G: Devuelve el componente de verde del vector</p> <p>B: Devuelve el componente azul del vector.</p> <p>A: Devuelve el componente alpha del vector.</p>

<i>Nombre:</i>	Abs
<i>Diseño</i>	
<i>Descripción:</i>	Devuelve el valor absoluto de un valor.
<i>Inputs:</i>	Val: valor a calcular su valor absoluto.
<i>Outputs:</i>	<p>O - Devuelve el vector 4D que componen los valores RGBA de la operación.</p> <p>R: Devuelve el componente de rojo del vector.</p> <p>G: Devuelve el componente de verde del vector</p> <p>B: Devuelve el componente azul del vector.</p> <p>A: Devuelve el componente alpha del vector.</p>

<i>Nombre:</i>	Mod
<i>Diseño</i>	
<i>Descripción:</i>	Calcula el modulo (resto)
<i>Inputs:</i>	X: especifica el valor a evaluar Y: dividendo.
<i>Outputs:</i>	O - Devuelve el vector 4D que componen los valores RGBA de la operación. 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.

<i>Nombre:</i>	Clamp
<i>Diseño</i>	
<i>Descripción:</i>	Acota un valor a un rango. Si el valor es inferior al mínimo, éste devuelve el mínimo, si es mayor que el máximo, éste devuelve el máximo, de lo contrario devuelve el valor original.
<i>Inputs:</i>	Val: valor a restringir. Min: valor mínimo. Max: valor máximo.
<i>Outputs:</i>	O - Devuelve el vector 4D que componen los valores RGBA de la operación.

	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:	Lerp
Diseño	
Descripción:	Realiza una interpolación lineal entre dos valores basándose es un porcentaje.
Inputs:	X: valor inicial Y: valor final T: porcentaje.
Outputs:	O - Devuelve el vector 4D que componen los valores RGBA de la operación. 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.

Conexiones

Las conexiones se realizaran haciendo clic sobre un input u output y manteniendo pulsado el botón del ratón mientras 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 puertos.

Restricciones de conexión:

- Un input solo puede tener una conexión asignada. Si se intenta conectar un nodo hacia un input que ya estaba previamente establecido, la conexión no se realizará.
- Un output puede proveer más de una conexión, lo que facilita la reutilización de nodos, y evita redundancias.
- Se ha de poder eliminar un nodo, en tal caso, todas las conexiones adyacentes a este nodo se deben eliminar.
- 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.
- Se han de poder eliminar conexiones más no inputs ni outputs.

Motor de dibujado y controles del visor 3D.

Además de la implementación del editor de nodos, el sistema debe contar con un sistema para dibujar el resultado del Shader que está siendo generado desde el árbol de nodos.

El sistema de dibujado será incluido en una parte de la interfaz de usuario. Desde allí se dibujara una malla que será cargada desde un fichero. Además se dispondrá una interfaz para cambiar la malla que está siendo dibujada por una nueva malla elegida de una selección. También habrá controles para establecer las propiedades de la luz, así como para girar la malla que se visualiza.

Modelo de iluminación

El sistema debe contar con un modelo de iluminación. Este modelo es importante no solo porque sirve para dibujar, evidentemente, sino porque a partir de él se modela el sistema entero y se generan los Shader desde el editor de nodos. Esto quiere que es el modelo de iluminación es quien afecta en la forma y el comportamiento a al resto de la aplicación de alguna manera.

Modelo de Blinn-Phong.

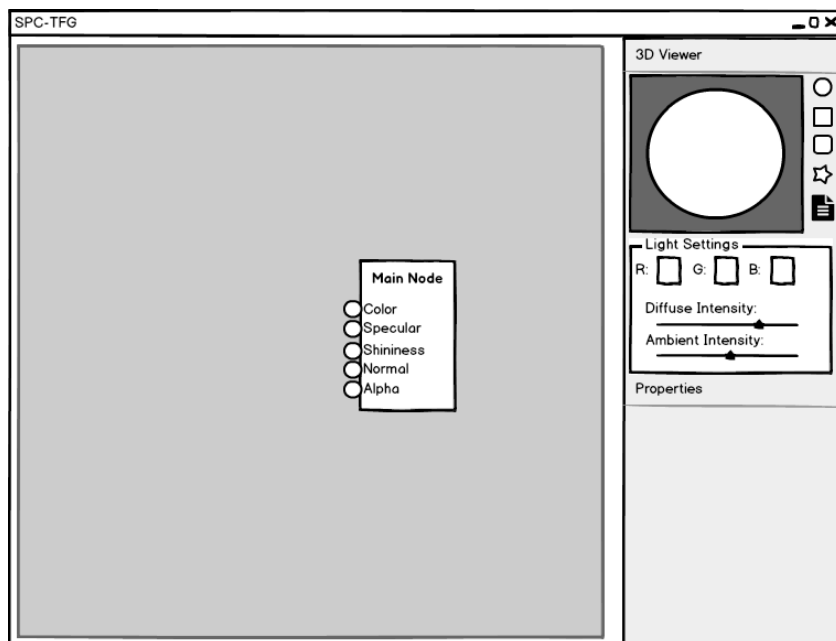
El modelo a implementar es el modelo de Blinn-Phong. Esto es debido a que es un modelo altamente usado, rápido, fácil de entender y también de implementar. Además él se puede obtener resultados bastante realistas o competitivos (depende de factores como el arte y el modelado).

Como bien se ha mencionado en los objetivos del proyecto, uno de las metas es la de dar a entender de una forma más intuitiva como se crean los gráficos y por computador (en especial en la etapa del Fragment Shader) y el modelo de Blinn-Phong es perfecto por su sencillez para tal objetivo.

Además es un modelo clásico y se ha considerado que se merece estar incluido en este proyecto por el valor histórico.

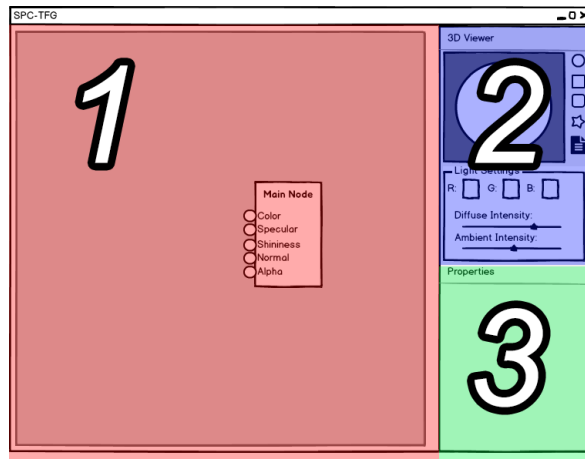
Interfaz de usuario y uso.

Presentación y descripción de interfaz de usuario:



Interfaz gráfica del programa.

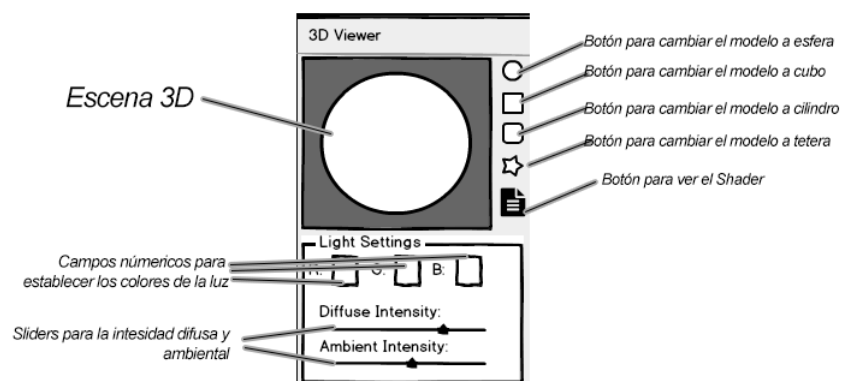
La interfaz de usuario se presenta en forma de ventana de aplicación de escritorio. Está constituida por 3 secciones principales:



Secciones de la interfaz gráfica.

La **primera sección** es donde se haya el editor de nodos, en ella aparecen inicialmente el nodo principal. En este espacio es donde se trabajará con los nodos.

La **segunda sección** es el visor 3d y sus controles. Aquí es donde se encuentra la escena en 3D y alrededor los controles para las propiedades de la luz y el cambio de malla, además del botón para ver el Shader generado:



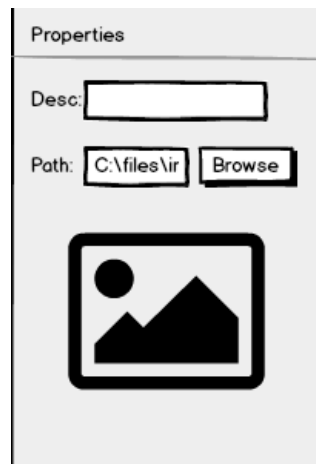
Descripción de los controles:

- **Botones para cambiar el modelo:** Según se pulse un botón de este grupo, se pasará a dibujar el modelo seleccionado.
- **Botón para ver el Shader:** Al darle clic abrirá una ventana modal donde se podrá ver el Shader con el que se está dibujando la escena actual.
- **Componentes de color de la luz:** Estos controles solo aceptan números, sirven para almacenar la información del color de la luz.
- **Sliders para el control de las intensidades:** Controlan la intensidad de la luz para el componente difuso y especular, el hecho de tener dos componentes en lugar de uno (como típicamente se realiza) da pie a realizar ajustes más diversos y obtener resultados mucho más completos (ya que el componente ambiental y el difuso no comparten la misma intensidad de luz).

La **tercera sección** es la sección de las propiedades de los nodos. En ella aparecerán diferentes controles visuales en función del nodo seleccionado. Algunos nodos no contendrán más que un campo para una descripción o una nota, mientras que otros como los nodos Const Float, Vector2, Vector3, Vector4 y Textura tendrán campos para establecer sus correspondientes valores.

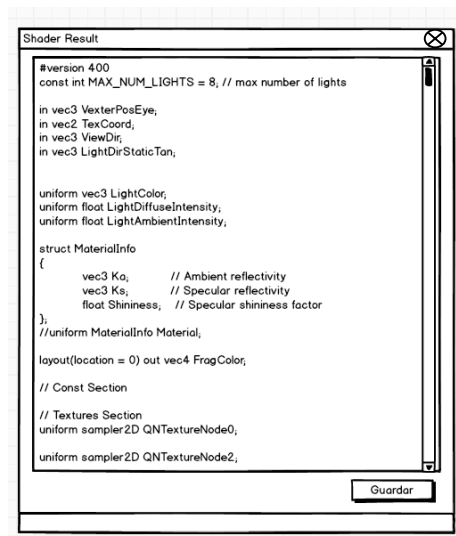
Properties	Properties	Properties	Properties
Desc: <input type="text"/>	Desc: <input type="text"/>	Desc: <input type="text"/>	Desc: <input type="text"/>
R: <input type="text"/>	R: <input type="text"/>	R: <input type="text"/>	V: <input type="text"/>
G: <input type="text"/>	G: <input type="text"/>	G: <input type="text"/>	
B: <input type="text"/>	B: <input type="text"/>		
A: <input type="text"/>			

Diseño de las propiedades de los nodos Vector4, Vector3, Vector2 y Float respectivamente.



Propiedades del nodo Texture

Las propiedades del nodo Texture contienen un botón que al ser pulsado abre una ventana modal para buscar archivos de tipo imagen. Al seleccionar una imagen pasará a mostrarse en miniatura justo debajo.



Ventana modal donde se enseña el Shader resultado

La ventana modal que se despliega cuando se pulsa el botón de “Ver Shader” tiene como finalidad el poder mostrar el código generado por el programa y posibilitar la opción de ser guardado en disco.

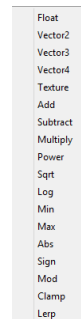
Uso e interacción

A continuación se hará una descripción de las posibles acciones que se pueden realizar en el programa.

Interacción con el editor de materiales:

Las interacciones con el editor de materiales son abundantes y cabe listarlas para su mejor documentación:

- **Seleccionar nodos:** Para seleccionar un nodo se debe realizar clic sobre el mencionado nodo. Al seleccionar un nodo se mostraran sus propiedades la sección de “Propiedades” del panel derecho de la interfaz gráfica.
- **Manipular nodos:** Los nodos se puede mover de un sitio a otro. Para realizar esta acción se deberá pulsar clic sobre un nodo y mientras se mantiene presionado arrastrar el ratón hasta la posición donde se quiere dejar.
- **Crear nodos:** Para crear nodos se debe pulsar clic derecho sobre el editor de nodos, en ese momento un menú desplegable aparecerá. En el menú se encuentran opciones para cada uno de los nodos disponibles, para crear un nodo se ha de elegir una de las opciones y el nodo aparecerá en lugar donde se encuentra el cursor.
- **Eliminar nodos:** A excepción del nodo principal, el resto de nodos se pueden eliminar. Para eliminar un nodo primero se deberá pulsar la tecla Ctrl, mientras se mantiene pulsada se hará clic derecho sobre el nodo a eliminar.



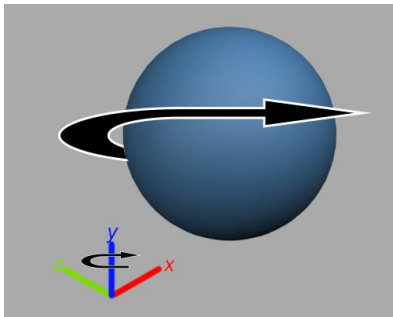
*Menú desplegable
para la creación de
nuevos nodos*

- **Crear conexiones:** Para crear conexiones se debe hacer clic sobre un input o un output y mientras se mantiene pulsado arrastrar hasta un nuevo input u output. Se creara la conexión si es una conexión legal (ver restricciones de conexión).
- **Eliminar conexiones:** Para eliminar una conexión se debe pulsar la tecla Ctrl y mientras se mantiene, hacer clic derecho sobre la conexión que se desea eliminar.

Interacción con el editor el viewport 3D:

Las acciones que se pueden realizar desde el *viewport* de dibujar de OpenGL son principalmente para manipular y transformar el modelo que se dibujar:

Rotar modelo:



Requisitos.

A continuación se realiza un análisis y enunciado de los requisitos funcionales y no funcionales del sistema.

Requisitos funcionales

Comentado [SP15]: Definir RF

Identificador	RF-1
Título	Iniciar la aplicación
Necesidad	Esencial
Descripción	El sistema deberá iniciarse sin errores y desplegarse con normalidad.

Identificador	RF-2
Título	Mostrar interfaz de usuario
Necesidad	Esencial
Descripción	El sistema deberá mostrar en la pantalla la interfaz de usuario.

Identificador	RF-3
Título	Interactuar con nodos
Necesidad	Esencial
Descripción	El sistema deberá permitir al usuario manipular los nodos del editor de nodos.

Identificador	RF-4
Título	Eliminar nodos
Necesidad	Esencial
Descripción	El sistema deberá permitir al usuario el poder eliminar los nodos que desee nodos. Para eliminar un nodo se deberá mantener pulsada la tecla Ctrl y pulsar clic derecho sobre el nodo en cuestión.

Identificador	RF-5
Título	Crear nodos
Necesidad	Esencial
Descripción	El sistema deberá permitir al usuario el crear nuevos nodos. Para crear un nuevo nodo el usuario deberá pulsar clic derecho sobre el área de edición de nodos y seleccionar una opción de un menú desplegable que contiene el nombre de todos los nodos.

Identificador	RF-6
Título	Eliminar conexiones
Necesidad	Esencial
Descripción	El sistema deberá permitir al usuario eliminar conexiones entre nodos. Para eliminar una conexión se deberá mantener pulsada la tecla Ctrl y pulsar clic derecho sobre la conexión en cuestión.

Identificador	RF-7
Título	Crear conexiones
Necesidad	Esencial
Descripción	El sistema deberá permitir al usuario crear nuevas conexiones entre nodos. Para crear una conexión se deberá hacer clic sobre un input o un output de algún nodo y se arrastrar hasta llegar a la posición de un input/output.

Identificador	RF-8
Título	Dibujar una escena 3D
Necesidad	Esencial
Descripción	El sistema debe dibujar una escena en 3D.

Identificador	RF-9
Título	Modificar los parámetros de luz
Necesidad	Esencial
Descripción	El sistema deberá permitir mediante controles visuales cambiar los parámetros de la luz: Color (tres campos numéricos para valores R, G y B), intensidad (un campo numérico), intensidad difusa e intensidad ambiental (un campo numérico).

Identificador	RF-10
Título	Cambiar de modelo a dibujar
Necesidad	Deseable
Descripción	El sistema deberá permitir cambiar el modelo que se está dibujando por uno del sistema, en ningún caso un modelo elegido por el usuario.

Identificador	RF-11
Título	Refrescar escena 3D
Necesidad	Escencial
Descripción	El sistema deberá redibujar la escena 3D cada vez que se genere un Shader nuevo o que se interactúe con la misma.

Identificador	RF-12
Título	Interactuar con el visor 3D
Necesidad	Escencial
Descripción	El visor 3D debe permitir al usuario el poder girar el modelo en los ejes Y (vertical) y X (horizontal) mediante una acción de clic y arrastre del ratón.

	Además se debe permitir alejar o acercar la cámara con una acción de clic derecho y arrastre vertical del ratón y de hacer zoom in o zoom out con el movimiento de la rueda del ratón. Todas las acciones deben producirse dentro del visor 3D.
--	---

Identificador	RF-13
Título	Generar Shader
Necesidad	Escencial
Descripción	El sistema deberá permitir generar un Shader a partir de las conexiones hechas sobre el nodo principal. En caso de no tener conexiones se asignaran valores por defecto. Los Shader nuevos se generarán cada vez que surja un cambio en el árbol de nodos del editor de nodos o en las propiedades de los nodos.

Identificador	RF-14
Título	Visualizar el Shader
Necesidad	Escencial
Descripción	El sistema deberá permitir ver el código fuente del Shader en una ventana modal.

Identificador	RF-15
Título	Guardar el Shader
Necesidad	Escencial
Descripción	El sistema deberá permitir guardar el código del Shader como fichero de texto.

Identificador	RF-16
---------------	-------

Título	Salir del programa
Necesidad	Escencial
Descripción	El sistema deberá permitir salir de la aplicación.

Requisitos no funcionales

Descripción de los requisitos no funcionales.

Identificador	RNF-1
Título	Escalabilidad
Necesidad	Escencial
Descripción	El sistema debe permanecer abierto a posibles mejoras en aspectos como el número de operaciones o el modelo de Shading empleado.

Identificador	RNF-2
Título	Desempeño
Necesidad	Deseable
Descripción	El sistema no deberá en ningún momento superar un tiempo estimado de espera de más de 2 segundos. Este es un tiempo aproximado de lo que puede tardar en resolverse la operación más costosa de la aplicación (Crear el Shader)

Identificador	RNF-3
Título	Facilidad de uso
Necesidad	Seguridad

Descripción	El sistema debe ser de fácil uso y no hará falta entrenamiento básico para su utilización. (No incluye entendimiento del proceso de Shading y de los Shaders)
-------------	---

Diseño del sistema

Diseño

Arquitectura

.
.
.

11. Experimentación

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

Comentado [SP16]: TODO

12. Conclusiones y trabajos futuros.

Aportaciones

Comparación con los objetivos

Trabajos futuros

13. Bibliografía y Referencias

<http://acegikmo.com/shaderforge/>

<http://www.marmoset.co/toolbag/learn/pbr-theory>

<https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/PhysicallyBased/index.html>

[Real Time Rendering](#)

[3D Math Primer For Graphics and Game Development](#)