Universidad de las Ciencias Informáticas

Facultad 5

Trabajo de Diploma para optar por el Título de Ingeniería en Ciencias Informáticas

"Algoritmo de Niveles de Detalles para la Visualización de Modelos 3D"

Autores: Dailen Vega Infante

Celina Fernández Balbuena

Tutores: Ing. Osvaldo Pereira Barzaga

Ciudad de la Habana

Julio 2010

DECLARACIÓN DE AUTORÍA

Declaramos ser únicas autoras de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

Autores:

_____ _____

Celina Fernández Balbuena

Dailen Vega Infante

Tutor:

_____

Ing. Osvaldo Pereira Barzaga

DATOS DE CONTACTO

Tutor: Ing. Osvaldo Pereira Barzaga
Edad: 25 años
Ciudadanía: cubano
Institución: Universidad de las Ciencias Informáticas (UCI)
Título: Ingeniero en Ciencias Informáticas

Categoría Docente: Profesor Instructor

E-mail:opereira@uci.cu

Graduado de la UCI, con seis años de experiencia en el tema de la Gráfica Computacional y líder de un proyecto de Visualización Médica en la Universidad de las Ciencias Informáticas.

DEDICATORIA

De Dailen

A mis padres, por su amor más noble y total dedicación.

A mi novio, por estar siempre para mí y ser mi guía en muchos aspectos.

A toda mi familia, por su apoyo y cariño.

A todos ellos, quiero dedicarles el resultado de este trabajo y uno de los momentos más felices de mi vida.

De Celina

A mis padres por su total dedicación para hacer posible este triunfo.

AGRADECIMIENTOS

De Dailen

A mis padres Giraldo y Denia, por dármelo todo, por hacerme llegar hasta este momento, por su amor y apoyo de toda la vida, por ser la luz de mis días.

A mi tía Daisy, por ser mi otra madre, por darme tanto cariño y ánimos para continuar.

A mi novio Yerandi, por su amor infinito, por su compañía en los momentos de alegrías y tristezas, por hacer suyas mis preocupaciones con las pruebas u otros motivos, por todos los momentos que hemos compartido juntos, por toda su paciencia todo este tiempo.

A toda mi familia en general, que hoy se sienten muy orgullosos de mí.

A Celina, por ser más que mi amiga, mi hermana, por su cariño incondicional, por soportarme, por ser quien es.

A Osvaldo, por ayudarnos tanto en la realización de este trabajo cuando pensábamos que no lo lograríamos.

A mis compañeros de estudio y mis amigas, las de antes de entrar a la UCI, a las que conocí cuando llegué aquí y a las que se fueron sumando con el paso del tiempo, con las que he compartido tantas cosas buenas y que se quedarán conmigo para siempre, a Ivette, Anna, Vity, Liannet, Leyanis, Arianna, Aniay, Maray, Annierys, por ayudarme en lo que me hiciera falta, Yanet, mi chiquitica, que me daba tantas fuerzas en los momentos más difíciles, en los que hacía falta una buena amiga.

A la UCI, por acogernos a todos y formarnos como profesionales.

A mis profesores por su empeño en prepararnos como hombres y mujeres de ciencia.

A Dios, por estar para todos.

AGRADECIMIENTOS

De Celina

Agradezco de una manera muy especial a esas dos personas que me han dado el ser, mi madre y mi padre, por tenerlos de guía en todo momento y recibir de ellos tanto amor y cariño.

A mis hermanos por tenerlos y sentirse orgullosos de mí.

A mis sobrinos por darme tanta felicidad.

A mi familia por apoyarme y demostrarme que siempre puedo contra con ellos.

A la Revolución por permitirme pertenecer a este maravilloso proyecto y formarme como una profesional.

A mi compañera de tesis Dailen, que más que mi amiga es mi hermana, demostrándomelo en cada momento de estos 5 años que compartimos juntas, pasando buenos y malos momentos.

A Osvaldo por confiar en nosotras y darnos la fuerza que necesitábamos para el desarrollo de este trabajo, sin su apoyo no hubiera sido posible.

A mi amiga Leyanis y su familia por acogerme como una hija más y hacerme sentir parte de ellos.

A la peque del grupo Yanesita por estar siempre.

Al circulo de amigas por ocupar un lugarcito en mi corazón Rossemary, Mara, Vity, Liannet, Aneydis, Arianna, Annierys, Aniays.

A todos los profesores por educarnos de la mejor manera y formarnos como mejores personas.

A Dios por darme las fuerzas necesarias cada vez que se lo pedía.

A todas las personas que tarde o temprano han llegado a mi vida para quedarse.

RESUMEN

El rápido avance de los sistemas de realidad virtual y de la tecnología que le da soporte se ha ido insertando en la sociedad, agilizando aun más su desarrollo. Es por ello que también se ha tenido que trabajar en aras de lograr una mejor visualización de estos sistemas de realidad virtual, donde los modelos tridimensionales deben ser cada vez más realistas e interactivos para lograr que los usuarios perciban las mismas sensaciones que en el mundo real, estos modelos tridimensionales requieren un alto volumen de información.

La aplicación de técnicas que mejoren este proceso de visualización, conjuntamente con el hardware que se utilice para ello, constituyen aspectos fundamentales a tratar en este trabajo; obteniendo como resultado un algoritmo que perfeccione dicho proceso de visualización.

TABLA DE CONTENIDOS

INTRODUCCIÓN.

Como parte del proceso de desarrollo social, actualmente se impone la necesidad de nuevos retos que marquen pautas significativas en aras de un desarrollo sostenible para todos. La Informática es una de las ciencias que ha manifestado un acelerado impacto en las esferas de la vida social y económica, siendo el avance tecnológico y científico técnico de los últimos años; premisas fundamentales de desarrollo de nuevas generaciones.

La tecnología ha acelerado su desarrollo al punto de ser más rápido que la habilidad de las personas para hacer uso de ella, llegando a decir que tanto la creatividad como la imaginación tienen la facilidad de formarse dentro de un mundo inmenso y simulado. Con el fin de eliminar las brechas existentes entre la capacidad de aprendizaje de las personas y el rápido desarrollo de las tecnologías surgen los sistemas de realidad virtual, que según *Aukstakainis* y *Blatner* son: “Una forma en que los humanos visualizan, manipulan e interactúan con computadoras y datos extremadamente complejos”.

La Realidad Virtual ha sido el resultado de la combinación gradual de dos importantes factores, la computación gráfica y la visión por computador. La misma surge como una necesidad de explorar la realidad, permitiéndonos estar donde no estamos y hacer aquello que nunca habíamos hecho. Sus aplicaciones han experimentado un vertiginoso salto en los últimos tiempos y no nos basta solo con observar mundos completamente realistas sino en ser parte de ellos e interactuar dentro del mismo, asumiendo roles y responsabilidades que nos convierta en usuarios completamente activos. Su alcance es hoy mucho mayor que los juegos interactivos, la representación estática de objetos o realización de imágenes.

Unas de las ramas del cocimiento que abarca la Realidad Virtual es la visualización de modelos tridimensionales en tiempo real, la cual se encarga de todo lo relacionado al problema de representar superficies complejas sin afectar el nivel de interactividad y dinamismo de los modelos en la escena virtual. Esta línea de investigación ha experimentado considerables contribuciones científicas con el objetivo de visualizar un elevado número de modelos *3D* en la escena con altos niveles de realismo e interacción en tiempo real; debido a que los usuarios de las aplicaciones de realidad virtual aplicadas a otras esferas como: la biología, la medicina, la arquitectura y otras; demandan cada vez más aplicaciones que representen mayores

volúmenes de información con un nivel de realismo y precisión sin que se vea afectada la interacción en tiempo real del entorno virtual [1].

Con el avance de las prestaciones de las tarjetas gráficas se han incrementado voluminosamente la utilización de las mismas para aumentar el realismo de la visualización en los sistemas de realidad virtual, permitiendo la visualización de cientos de miles de primitivas por segundo. Podría suponerse que el uso de una tarjeta gráfica resolvería los problemas de lograr una mejor visualización, sin embargo, si no se aplican algoritmos o técnicas de optimización, el hardware resulta insuficiente.

Nuestro país no se ha quedado al margen de este proceso de informatización y del incremento de nuevas tecnologías, lo ha utilizado para ampliar sus horizontes y a pesar de sus limitaciones en el ámbito económico, sus esfuerzos se encaminan a adquirir elevados índices de cultura y utilizar esas tecnologías en beneficio de la sociedad. La Universidad de las Ciencias Informáticas tiene como misión promover la producción de software y servicios informáticos en nuestro país, concentrándose principalmente en el desarrollo de proyectos productivos, alcanzando resultados alentadores en diferentes esferas. La misma está compuesta por facultades, cada una de ellas especializadas en la producción de software en un área específica.

En la facultad 5 se encuentran los proyectos relacionados con la Realidad Virtual; donde se hace más exigente modelar objetos virtuales a un mayor nivel de realismo en el cual los usuarios experimenten sensaciones muy similares a las del mundo real. Al interactuar con objetos en el espacio es necesario un proceso de optimización para poder realizar diversas funciones gráficas. La optimización de la visualización, trata de lograr, a través de técnicas y algoritmos, que se visualice la mayor cantidad de información en el menor tiempo posible permitiendo que los entornos virtuales sean lo más fieles posible a la realidad, de manera que les resulte fluida a los usuarios, independientemente de si la carga de objetos en la escena es muy grande, o de la cantidad de procesos que se necesiten hacer durante la visualización.

Uno de los proyectos que integran esta facultad es Paseos Virtuales, donde la visualización de las escenas carece de realismo puesto que el texturizado de los objetos es muy básico y no pueden representar por si solas cualidades de rugosidad, materiales, y otras propiedades de los objetos que atentan con la visualización realista de estos entornos virtuales. Ante este

problema científico se plantea la siguiente interrogante: ¿Cómo obtener escenas virtuales con un elevado nivel de realismo e interacción?

Definiendo como objeto de estudio la Visualización de modelos gráficos, siendo el campo de acción los Algoritmos de multiresolución para la visualización de modelos 3D.

El objetivo general que se propone este trabajo es: Implementar un algoritmo que permita la visualización realista e interactiva de una escena virtual a partir de la combinación de Técnicas de Niveles de Detalles y Texturizado de Objetos.

Con el propósito de satisfacer las necesidades planteadas se proponen las siguientes tareas de la investigación:

- Caracterizar las herramientas utilizadas en proyectos de Realidad Virtual para la interacción con modelos 3D.
- Caracterizar técnicas de optimización y aceleración utilizando Hardware Gráfico para alcanzar una mejor visualización.
- Desarrollar una propuesta detallada de un algoritmo que de solución al insuficiente realismo en la representación de escenas.
- Implementar un Demo que muestre los resultados alcanzados.
- Descripción de los aportes, limitaciones y recomendaciones en el trabajo desarrollado.

CAPÍTULO 1.FUNDAMENTACIÓN TEÓRICA.

En el desarrollo del presente capítulo se abordarán diferentes aspectos, entre los que se destacan una breve reseña sobre el texturizado de objetos, se abordarán algunas de las técnicas que permiten interactuar con modelos tridimensionales mediante el uso de diferentes niveles de detalles, permitiendo mediante la simplificación de polígonos el aumento de los *Frames per second*, (*fps*) y un alto nivel de interactividad. Trayendo como desventaja que se afecte el realismo de los modelos; sin embargo mediante técnicas orientadas específicamente al trabajo con materiales, texturas y otros efectos se alcanzará el realismo necesario para conseguir una mejor visualización. Finalmente se tratarán términos relacionados con la Programación Gráfica.

1.1 MODELADO MULTIRESOLUCIÓN.

Se define el concepto de modelo multiresolución como aquel que permite la representación y manipulación de entidades geométricas con diferentes niveles de detalle [2].

Independientemente de cómo se genere el modelo poligonal, de la aplicación que lo utilice y del campo en el que se trabaje, almacenar, transmitir y visualizar enormes cantidades de datos se hace muy difícil. Con el fin de solucionar estos problemas aparece el modelado multiresolución [3]. Este tipo de técnica almacena ***n*** aproximaciones distintas del objeto, llamadas Niveles de Detalle, siendo ***n*** proporcional al tamaño del propio objeto.

Para generar las distintas aproximaciones es necesario el uso de un método de simplificación, el cual decide que información es menos relevante y la elimina del modelo, creando una aproximación formada por un menor número de vértices y triángulos. Aplicándolo de forma sucesiva se obtienen las distintas aproximaciones. Los primeros modelos multiresolución, sólo almacenaban un conjunto pequeño de niveles de detalle. Sin embargo, esta solución presenta el problema de la transición entre dos niveles que produce un perceptible efecto de salto en la imagen. Por este motivo, los modelos multiresolución han pasado a albergar ***n*** niveles de detalle, donde entre dos representaciones consecutivas la diferencia suele ser de un vértice, una arista o un triángulo.

1.1.1 NIVELES DE DETALLE.

El uso de los niveles detalle o *LOD* surge con el objetivo de incrementar las prestaciones del sistema gráfico. De nada vale visualizar un objeto con múltiples polígonos si el mismo se encuentra a una larga distancia del observador, pudiendo representar el objeto con pocos polígonos, de manera que el hardware gráfico tenga mejor rendimiento. Permitiendo lograr una navegación fluida y continua por toda la escena y mejorar solo lo que estemos viendo. A medida que se vaya acercando el objeto se puede ir aumentando el nivel de detalle hasta lograr una mejor representación.

Los niveles de detalles permiten establecer muy buena relación entre velocidad y complejidad. Utilizan modelos complejos para visualizaciones en tiempo real, lo que implica abordar dos cuestiones: las técnicas de simplificación de la geometría y las técnicas de selección del nivel de detalle. Las técnicas de simplificación se basan en conseguir representaciones de un modelo a menor resolución Fig: 1, mientras que la de selección del nivel de detalle elige el nivel de resolución del modelo según las circunstancias en que se va a utilizar, mediante la *distancia observador – objeto o de la proyección del volumen envolvente* de dicho objeto en pantalla.

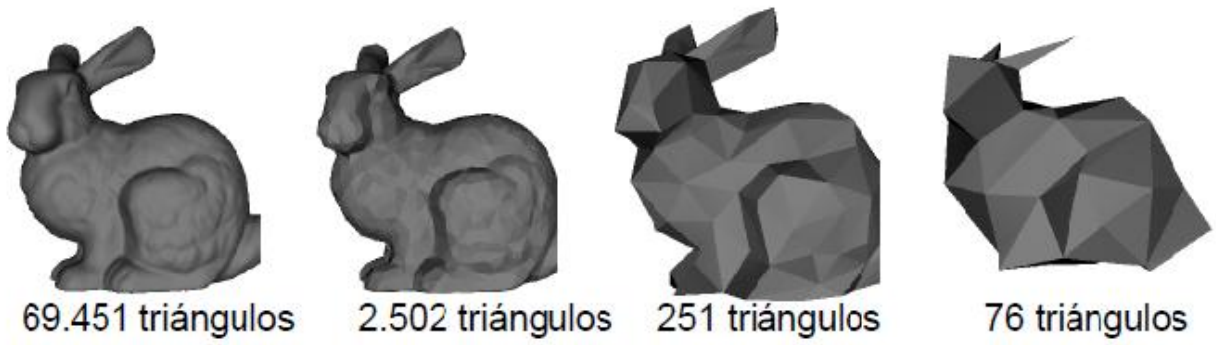

Fig: 1 Simplificación de Polígonos

Aún con estas técnicas de reducción de la resolución de los datos, en dependencia de su distancia al centro de proyección (*COP* del inglés *Center of Projection*), no es suficiente, debido a que es imposible tener cargado en memoria todos los datos. De aquí, que se necesiten estructuras de datos espaciales para el manejo, almacenamiento e intercambio de los datos entre la memoria y el disco de forma eficiente. Estas estructuras permiten conocer eficientemente la topología de los datos, para así tener cargado en memoria sólo una parte y la

otra mantenerla en disco. Para esto, se pueden utilizar estructuras como *Quadtrees*, *Octrees*, *K-dTree y R-Tree* [4].

1.1.1.1 CLASIFICACIÓN DE LOS NIVELES DE DETALLE.

El enfoque de multiresolución se puede clasificar en función del número de aproximaciones que utilizan. Pueden ser **discretos o estáticos, continuos o dinámicos**, o **dependientes del punto de vista**. Aunque el concepto de *LOD* no está circunscrito a una representación concreta del modelo, se suele aplicar tradicionalmente a la simplificación de mallas poligonales, mayoritariamente triangulares.

Discretos o Estáticos: Fue propuesto en 1976 por Clark, también llamado *LOD* basado en rangos. Genera distintas versiones del modelo original con distinto nivel de detalle. Realiza las aproximaciones durante el preproceso de la aplicación. Durante la ejecución no es necesario aplicar algoritmos de simplificación.

Continuos o Dinámicos: Este método, conocido también como *LOD* progresivo, parte de la idea del *LOD* discreto, con la diferencia de que el nivel de detalle de cada objeto es calculado en tiempo real cuando es necesitado, en lugar de ser preprocesado. Se genera una estructura de la cual se pueden extraer aproximaciones con el nivel de resolución deseado. Permite elegir el nivel de detalle sin restringirse a un conjunto de modelos precalculados.

***Dependientes del punto de vista*:** Adapta el nivel de detalle a la situación del punto de vista. Nivel de resolución no homogéneo en todas las zonas de la superficie del modelo. Podría plantearse desde un enfoque discreto o continuo, pero suelen ser continuos. Como es un modelo anisotrópico, un simple objeto puede abarcar múltiples niveles de simplificación; por ejemplo, las partes más cercanas del objeto se pueden representar con mejor detalle que las más lejanas (óptimo para objetos grandes y complejos como terrenos), o la silueta se puede representar con mejor resolución que las regiones interiores.

1.1.1.2 FACTORES DE SELECCIÓN DE LOS NIVELES DE DETALLES.

Los distintos factores que se han utilizado para realizar la selección de un determinado nivel de detalle son los siguientes [5]:

Distancia*:* Dado un objeto, respecto del observador, a mayor distancia menor será su tamaño. Para hacer una rápida estimación se suelen englobar a los objetos con cajas o esferas.

Incidencia: Ciertos objetos, según el ángulo de la vista, pueden verse de formas distintas. Por ejemplo una puerta vista de perfil, aún estando cerca del observador, necesitaría muy pocos polígonos para representarla.

Movimiento: Un objeto animado no es necesario que se represente con grandes detalles ya que estos apenas serán perceptibles.

Visión Periférica: Dibujar con más detalle aquellos objetos que están en el centro del campo de visión y con menos detalle aquellos que quedan alrededor de dicho centro.

Tarea: Utilizar niveles de detalle altos en aquellos objetos que se están utilizando para hacer una determinada tarea, y niveles bajos para aquellos objetos que no se utilicen.

Silueta: Utilizar una mayor resolución de la malla en las partes extremas, que definen la silueta del objeto modelado.

1.1.1.3 REQUISITOS AL APLICAR NIVELES DE DETALLES.

Las propiedades que un modelo multiresolución deberá cumplir, se resumen en los siguientes aspectos [5]:

- El incremento en el tamaño del modelo no sea significativo cuando se aumenta el número de niveles de detalle.
- La extracción de un *LOD* se realice de forma eficiente, es decir, si el modelo contiene ***n*** niveles de detalle la información debe estar organizada de manera que el algoritmo de extracción pueda recuperar los datos en el menor tiempo posible a fin de permitir una visualización interactiva.
- La malla extraída sea continua, entendiendo por continua que no aparezca ningún tipo de rotura o agujero para todas las posibles representaciones.
- La transición de un nivel a otro se realice de forma suave, es decir, al pasar de un nivel de detalle al siguiente (ya sea para aumentar o disminuir el nivel de detalle) no ha de percibirse ningún tipo de salto en la imagen.

1.2 VISUALIZACIÓN REALISTA E INTERACTIVA.

Por estos días los sistemas de realidad virtual requieren que se visualicen escenas lo más reales e interactivas posibles. Mediante el uso de técnicas de niveles de detalles se alcanza un mejor rendimiento, sin embargo trae consigo falta de realismo en los modelos tridimensionales; siendo necesario aplicarle determinadas técnicas de texturizado a los objetos para darle solución a este problema. A continuación se describen algunas de estas técnicas, haciendo énfasis en sus principales características y los resultados que se alcanzan mediante su aplicación y solo tratando de proporcionar una breve reseña de la forma en que se logran tales efectos.

1.2.1 ¿QUÉ ES EL TEXTURIZADO DE OBJETOS?

Una textura es una imagen que se asigna a las superficies poligonales (secuencia de caras), cuando se realiza la acción de aplicarle la imagen al polígono se conoce como mapeo de texturas, en la que no se crea geometría nueva. Las texturas es una forma de dotar drásticamente el nivel de detalle y realismo a los objetos sin necesidad de aumentar su complejidad geométrica. Se pueden definir varios tipos de texturas como son:

- Texturas Unidimensionales (*1D*), las mismas presentan un solo píxel de alto o de ancho.
- Texturas Bidimensionales (*2D*), son imágenes con más de un píxel de alto o de ancho, los mismo suelen ser ficheros de *BMP* o *JPG*.
- Texturas Tridimensionales (*3D*), además de cubrir un espacio en el plano presentan profundidad.

Las texturas pueden aplicarse sobre distintos tipos de primitivas como polígonos o superficies curvas, y éstas pueden repetirse para cubrir toda la superficie del objeto. Finalmente, la textura puede aplicarse de distintas formas: puede utilizarse para colorear una primitiva o para simular los reflejos del entorno.

El texturizado permite también el uso de más de una textura a la vez en un polígono y es llamado multitexturizado. Esta técnica combina dos o más texturas, para ello, en lugar de crear dos versiones de la misma textura, se crea un mapa de iluminación, que no es más que una textura que superpondrá a la anterior, de este modo se consigue crear un nuevo efecto sin

cargar tanto al procesador, ver Fig: 2. En la actualidad las tarjetas gráficas implementan este método, consiguiendo que de una sola pasada se genere la imagen resultante sin tener que renderizar dos veces el polígono. La siguiente imagen muestra como resultado el multitexturizado [6].

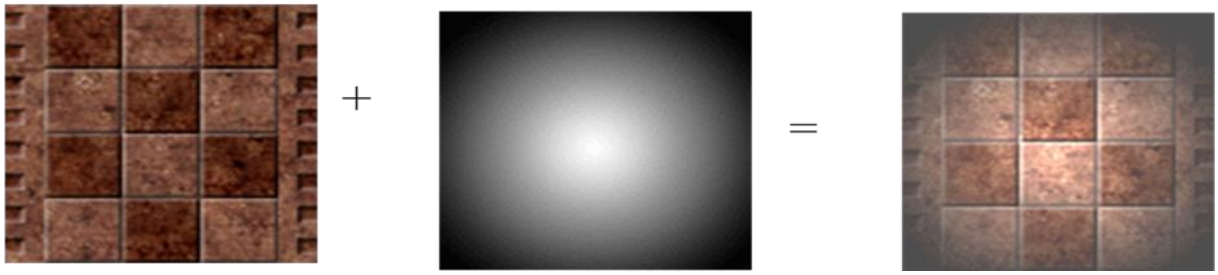

Fig: 2 Proceso de Multitexturizado: a) Textura original b) Mapa de luz c) Resultado final

1.2.2 TÉCNICAS DE TEXTURIZADO DE OBJETOS.

1.2.2.1 MIP MAPPING

Técnica de manejo de texturas, donde se tienen diversas versiones de una misma textura, pero con tamaños diferentes. Se basan principalmente en tener colecciones de imágenes de mapas de bits que acompañan a una textura principal para aumentar la velocidad de renderizado y reducir sus artefactos. Su funcionamiento se fundamenta básicamente en que el renderizador utiliza la textura principal cuando es necesario renderizar a todo detalle y cambia al *mipmap* adecuado cuando renderiza la textura desde cierta distancia o en un tamaño menor.

Al decir que la velocidad de renderizado aumenta se trata de que la cantidad de *píxeles (texels)* procesados puede ser mucho menor que en una textura común. Además se reducen los artefactos dado que luego del procesamiento las imágenes *mipmap* se encuentran sin *aliasing*, es decir, sin bordes dentados o escalonados, evitándose aplicar la técnica de *antialiasing* y aunque resolvería el problema sería más trabajoso [7].

Su aplicación es muy útil cuando se tienen texturas que se repiten una y otra vez a lo largo de una superficie. Sus ventajas se hacen más notables al no tener que contar con texturas tan grandes, además de que aumenta la calidad gráfica. Se evitan cálculos de texturas y

operaciones, como por ejemplo las que hace *OpenGl*, tratando de evitar que al tener una textura lo suficiente grande pero aplicada a un objeto que se encuentra distante se distorsione, perdiendo el realismo. A través de esta técnica los objetos alejados tendrán un nivel de resolución bajo y los cercanos una alta resolución, Fig: 3.

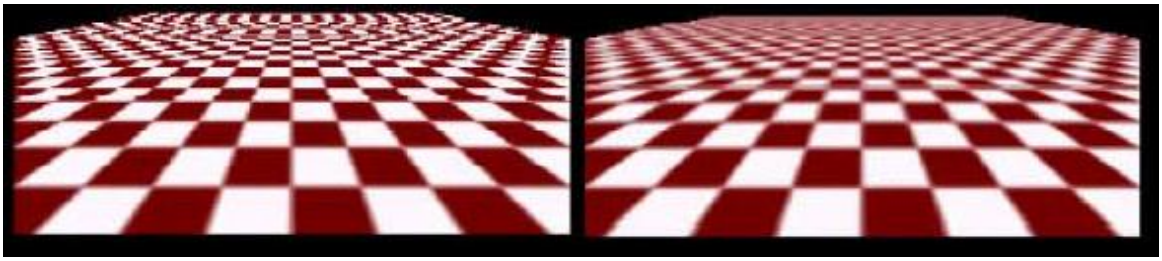

Fig: 3 a) Aplicación de textura sin Mip Mapping b) Aplicación de textura con Mip Mapping

1.2.2.2 LIGHTMAPS O SHADOWMAPS.

Lightmaps o mapa de luz es una técnica basada en la textura, donde no importa la geometría del objeto, siendo una forma más de conseguir efectos de sombra en los gráficos por computadoras, pues son diversas las formas de obtenerlas. En aplicaciones gráficas se basa en añadir una textura a todas las caras existentes en una escena *3D*, proporcionando a la escena un ambiente más sombreado que el anterior [8]. El funcionamiento de esta técnica se observa en la Fig: 4, donde a una textura base se le aplica el *lightmap* correspondiente a un polígono, esta iluminación se pre-calcula mediante cualquier algoritmo y se almacena en una textura [18].

El principio básico para conseguir una sombra es haciendo un render especial desde una fuente de luz (*depth map),* permitiendo calcular los objetos a los que ilumina dicha luz. Luego se hace el render estándar desde la cámara y se comprueba si cada píxel está representado en el *depth map*. Si lo está, la zona se ilumina, si no, se sombrea.

Uno de los factores que afectan la calidad de este tipo de sombra es cuando se estira mucho la sombra o se usan texturas demasiado pequeñas para almacenar el render, posibilitando que haya *aliasing* y que sea necesario aplicar algún tipo de filtrado, otra de las formas de solucionar este problema sería incrementando el tamaño del mapa de sombra, pero se puede ver limitado debido a las limitaciones de cálculo del hardware. Sus efectos al imprimir realismo a las escenas han aumentado considerablemente y sobre todo con la aparición de los *shader*.

Fig: 4 Efecto conseguido por el Lightmaps

1.2.2.3 BUMP MAPPING.

Bump Mapping o mapa de relieve es una técnica propuesta por *James Blinn* en 1978. Fundamentalmente es una técnica de gráficos computacionales *3D* el cual consiste en darle una textura de rugosidad a un objeto para proporcionarle realismo [17]; utilizando una imagen en escala de grises, en la que los colores claros significan protuberancias y los colores oscuros hendiduras [20]. Se basa en algoritmos que captan la textura inicial y la convierten en otra, creando pequeños *bump mapping* en la superficie del objeto para darle texturas, simulando la misma superficie pero con relieves. Los colores cercanos al negro se convertirán en hendiduras y los cercanos al blanco, serán protuberancias.

Se basa en modificar las normales de la superficie sin cambiar su geometría, las normales originales de la superficie seguirán perpendiculares a la misma. El *bump mapping* cambia la perpendicularidad por otras normales para lograr el efecto deseado, todo ello sin modificar la topología ni la geometría del objeto.

Contiene una escala de grises que indica un mapa de altura (*height map*) de la superficie, la forma en que se logra distorsionar el aspecto de la superficie con este mapa es tomando cada píxel y comparándolo con los píxeles adyacentes y de esta forma hallar una normal que será usada para cambiar el brillo generado por las luces en un punto de la superficie. Se utiliza más que todo para hacer relieves básicos de algún material en específico, como la aspereza de un material de concreto, del óxido de un metal o las grietas de una pared. El resultado al aplicar esta técnica es razonablemente rico y detallado, y pueden lograrse grandes parecidos a elementos naturales (como la textura de una naranja), Fig: 5. Tiene una gran ventaja y es que se aplica sobre una textura, por lo que el ahorro poligonal es considerable**;** pero no deja de ser una ilusión óptica pues no hay un relieve real [16].

Fig: 5 a) Sin Bump Mapping b) Con Bump Mapping

1.2.2.4 PARALLAX MAPPING

Parallax mapping (también llamado *"offset mapping" o "virtual displacement mapping"*), es una mejora de las técnicas *Bump mapping* o *Normal mapping*. La misma fue introducida en el año 2001 por *Tomomichi Kaneko*. Para el usuario final el *parallax mapping*, significaría tener texturas más realistas, por ejemplo como las de las rocas donde tendrán una profundidad más clara y un realismo mayor con poca influencia en el desempeño de la simulación [9].

Se logra implementar desplazando las coordenadas de la textura a un punto en el polígono por una función desde el ángulo de vista en el espacio de la tangente (ángulo relativo a la superficie normal) y el valor del mapa de alturas en tal punto. Desde un punto de vista inclinado, las coordenadas de la textura se desplazan más, y así se logra la ilusión de mayor profundidad debido a los efectos *parallax* mientras la vista se mueve.

La aplicación de esta técnica sigue siendo un truco, pues la geometría de la superficie se mantiene intacta, pero se consigue un mejor efecto de profundidad.

Fig: 6 Efecto conseguido por el Parallax Mapping en el juego Splinter Cell 3 - Chaos Theory:

a) Sin Parallax Mapping b) Con Parallax Mapping

1.2.2.5 NORMAL MAPPING.

Normal mapping o mapa de normal es la versión mejorada del *bump mapping* clásico. Los mapas de normales se aplican para simular relieve (detalles), en los objetos que no lo tienen. Su uso se basa en añadir detalles a un modelo de pocos polígonos. Mientras que el *bump mapping* perturba la normal de un modelo en un solo canal de escala de grises, el *normal mapping* la reemplaza completamente, contemplando los canales RGB, indicando la dirección de la normal de la superficie, donde R define la dirección horizontal de la normal, G la dirección vertical y B indica que tan profunda es la normal [19]. De esta forma, se consiguen objetos con el mismo detalle que otros con millones de triángulos más, liberando a la tarjeta gráfica de bastante trabajo.

Utilizar tres canales implica el uso de un mapa de bits común, posibilitando almacenar para cada píxel la información tridimensional de su orientación según la normal que posea. El color rojo (R) representa al eje X, el verde (G) representa al Y, y el azul (B) al Z. Estos mapas de normales pueden ser generados a través de software de desarrollo 3D, como es el caso del *Photoshop*, el cual presenta un *plugin NVIDIA* [15].

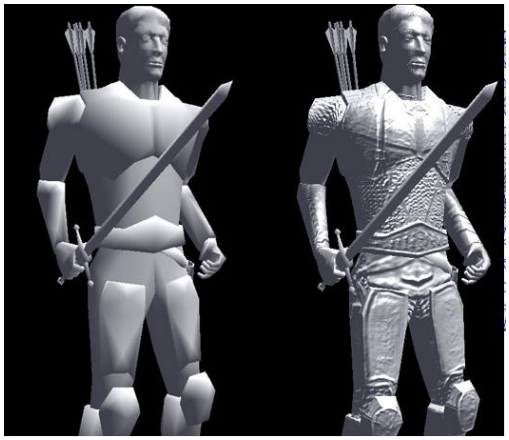

Fig: 7 Efecto del Normal Mapping

1.2.2.6 FILTRADO DE TEXTURAS.

Las texturas son creadas a resoluciones específicas, pero ya que la superficie en donde están aplicadas puede estar a cualquier distancia del observador, estas pueden mostrarse en tamaños arbitrarios en la imagen final. Como resultado, un *píxel* en la pantalla usualmente no corresponde directamente a un *texel*. Alguna técnica de filtrado debe ser aplicada para lograr imágenes claras a cualquier distancia. Se hace posible además con el filtrado, que cada píxel tenga una textura diferente a los de su alrededor. Hay varios métodos, con diferentes relaciones entre calidad de imagen y complejidad computacional. Estos son unos de los métodos más comúnmente usados en orden creciente de coste computacional y calidad de imágenes [17].

Filtrado bilineal: Pone una textura a un *píxel* con una media de las imágenes de los *píxeles* que lo rodean en el eje X y en el eje Y. Se utiliza para que cada *píxel* no tenga la misma textura que los de su alrededor.

Filtrado trilineal: Este tipo de filtrado, además de hacer el filtrado bilineal con las texturas, hace interpolación entre dos texturas empleadas para diferentes distancias, permitiendo que la transición entre *mipmaps* sea más suave; posibilita reducir los parpadeos de las imágenes.

Filtrado anisotrópico: Es un filtro que mejora la textura de la superficie de un objeto, especialmente en superficies oblicuas, o sea, todo lo que se ve en ángulo, perfeccionando la

manera cómo se ve dentro de los objetos, que se refiere a todos los espacios entre los bordes. Siendo ampliamente utilizado en el hardware gráfico.

1.3 HARDWARE GRÁFICO PROGRAMABLE.

Antes de la aparición de la *GPU* (*Unidad de Procesamiento Gráfico*), las tarjetas gráficas únicamente disponían de un limitado espacio de memoria llamado *frame buffer* o memoria de vídeo donde la *CPU* (*Unidad Central de Procesamiento*) escribía directamente los datos que codificaban las imágenes que se querían mostrar por pantalla después de haber realizado todos los cálculos pertinentes [10].

La *GPU* es un procesador dedicado exclusivamente al procesamiento de gráficos, para minimizar la carga de trabajo del procesador central en aplicaciones como los videojuegos y aplicaciones *3D* interactivas. De esta forma, mientras gran parte de lo relacionado con los gráficos se procesa en la *GPU*, la *CPU* puede dedicarse a otro tipo de cálculos (como la inteligencia artificial o los cálculos mecánicos en el caso de los videojuegos) [11].

Una *GPU* implementa ciertas operaciones gráficas llamadas primitivas optimizadas para el procesamiento gráfico. Una de las primitivas más comunes para el procesamiento gráfico en *3D* es el *antialiasing*, que suaviza los bordes de las figuras para darles un aspecto más realista.

Las tarjetas gráficas de última generación son de considerable importancia en la renderización de grandes cantidades de datos. *GeForce* es la denominación que tienen las tarjetas gráficas que cuentan con unidades de procesamiento gráfico (*GPU*) desarrolladas por la empresa estadounidense *NVIDIA*. Su introducción en el mercado posicionó a la entonces casi desconocida firma, a ser la compañía líder del sector. Actualmente, la serie *GeForce* ha conocido diez generaciones [12]. Su última generación, la *GeForce 400*, formada en un inicio por la *GTX 470* y la *GTX 480,* basadas en la arquitectura *Fermi*, representando un salto de calidad y rendimiento con la generación anterior de esta arquitectura y con una cantidad de transistores verdaderamente impresionante (tres mil millones).

1.3.1 SHADERS PROGRAMS.

En el 2000, la serie 2 de tarjetas *GeForce* permitía a la *GPU* hacerse cargo de funciones de transformación e iluminación que hasta ahora debía hacerlas la *CPU*, sin embargo no fue hasta la *GeForce3* que surgió el concepto de *shader* [10]. Permitiendo básicamente programar parte del proceso de visualización que realiza la *GPU*. En esta generación de *GPUs* se podían

programar las transformaciones geométricas. Estos programas reciben el nombre de *vertex shaders* o *vertex program* a los que luego se le sumaron los *píxel shaders* o conocidos también como *fragment shader* y un tiempo después surgió el *geometry shader*.

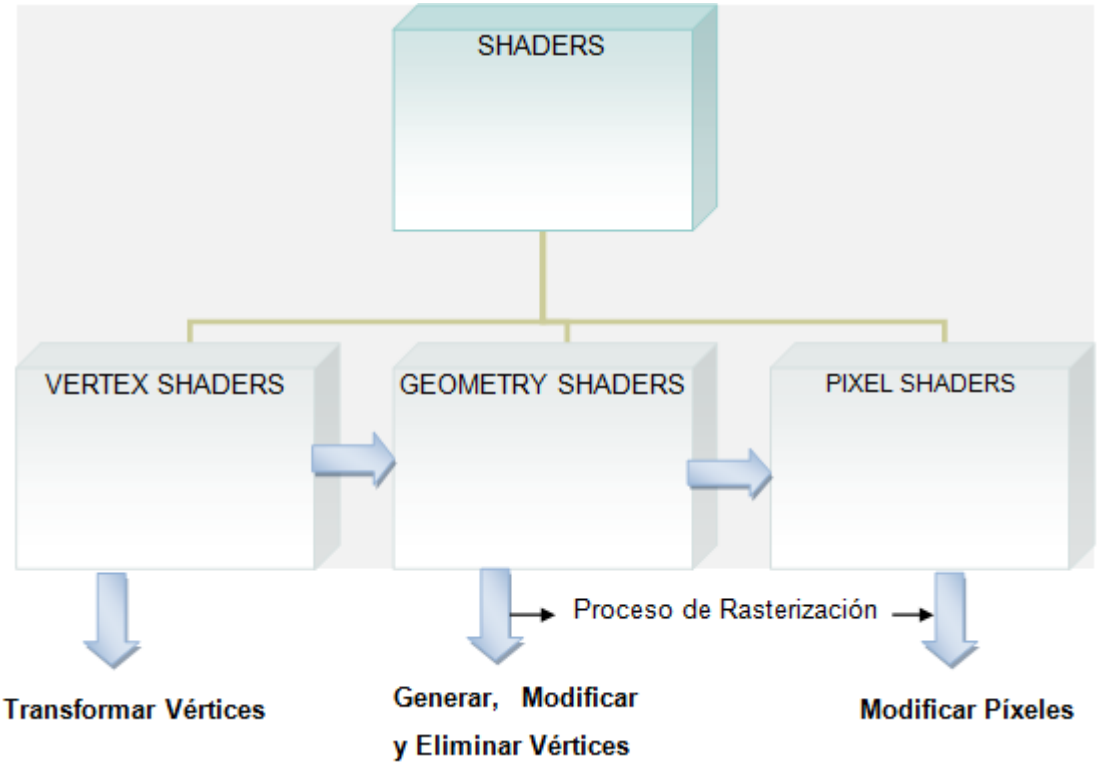

Fig: 8 Esquema de Trabajo de los Shaders.

Proceso de rasterización: proceso en el cual las superficies se dividen en un conjunto de puntos o píxeles y que luego serán desplegados en un medio de salida digital.

La utilización de los *shaders* ha brindado una gran gama de posibilidades a los programadores, entre las que podemos mencionar distintos tipos de efectos que han contribuido al desarrollo de entornos virtuales con un adecuado nivel de realismo e interactividad, entre los que se pueden mencionar: sombras, Iluminación por píxel, *bump mapping*, *parallax bump mapping*, entre otros.

1.3.1.1 VERTEX SHADERS.

Vertex shader es una herramienta capaz de trabajar con la estructura de vértices de los modelos tridimensionales y con ellos realizar operaciones matemáticas modificando estas variables y así definiendo colores, texturas e incidencia de la luz. Esto da libertad a los

programadores para realizar diferentes efectos desde la deformación de un objeto hasta la recreación de las olas del mar o el movimiento de las hojas de los árboles [13]. Lo que en realidad pretende esta herramienta es adicionar a una malla de polígonos elementos que se alojan en los vértices de dichos polígonos o simplemente modificarlos.

El *vertex shader* ha evolucionado con el tiempo encontrándose en la actualidad en la versión 4.0. Incluido en *Direct3D* y *OpenGL*, el *vertex shader* puede reproducir diferentes efectos realistas. Entre las características geométricas sobre las que se obtiene control se encuentran [10]:

- Transformaciones de los vértices.
- Transformaciones de las normales.
- Normalización y escalado.
- Iluminación.
- Generación y transformación de coordenadas de textura.

1.3.1.2 GEOMETRY SHADERS.

Las primeras tarjetas gráficas que dieron soporte a *geometry shaders* fueron las *NVIDIA*, específicamente la serie *GeForce 8800*. El geometry *shaders* se ejecuta luego del *vertex program*, recibiendo vértices, los cuales serían primitivas geométricas, como es el caso de puntos, líneas o triángulos, las cuales se pueden generar, modificar o eliminar, siendo un punto muy importante en su funcionamiento [14].

Un ejemplo clave de la forma en que trabaja sería la opción de poder utilizar o eliminar vértices de una malla según diferentes consideraciones, como por ejemplo según la distancia que pueda existir del observador a la geometría, haciendo uso de la técnica Niveles de detalle.

Los *geometry shaders* pueden ser programados en los siguientes lenguajes: *Ensamblador*, *Cg*, *HLSL* de *Direct3D* (*DirectX 10*) y *GLSL* (*OpenGL*).

1.3.1.3 PÍXEL SHADERS.

Un *píxel shader* es un programa de sombreado, normalmente ejecutado en la unidad de procesamiento gráfico. En *OpenGL* se conoce como fragmento de sombreado. Sirve para manipular un *píxel*, es decir, aplicar un efecto sobre la imagen (realismo*, bump mapping,* sombras*,* explosiones *y* efectos). Se trata de una función gráfica que calcula los efectos sobre

una base *per-píxel*. Dependiendo de la resolución, una cantidad de 2 millones de *píxeles* puede ser necesario para ser renderizado, iluminado, sombreado y dar color para cada marco [13].

Los *píxel shaders* están programados en los siguientes lenguajes: *Ensamblador, Cg, GLSL*. Algunas de las funciones sobre las que se obtiene control o efectos que se pueden crear son [10]:

- Acceso y aplicación de las texturas.
- Control del color de cada fragmento.
- Modificación del buffer de profundidad.
- Niebla.
- Sombras.

1.3.2 LENGUAJES DE PROGRAMACIÓN SHADERS.

La tecnología *shaders* es cualquier unidad escrita en un lenguaje de sombreado que se puede compilar independientemente. Para la escritura de esas instrucciones, los programadores hacen uso de lenguajes de programación diseñados específicamente para ello. Cada uno de estos lenguajes de programación necesita enlazarse mediante una *API*, entre las que podemos mencionar: *DirectX* u *OpenGL*. Existen otros lenguajes pero los siguientes son los más conocidos [13].

- ***HLSL*** (*Lenguaje de Sombreador de Alto Nivel*) es la implementación propiedad de *Microsoft*, la cual colaboró junto a *NVIDIA* para crear un lenguaje de sombreado. Los programas *HLSL* funcionan con estas tres formas, los *Vertex Shaders*, *Geometry Shaders* y *Píxel Shaders.*
- ***GLSL*** (*OpenGL Shading Language*) es el lenguaje desarrollado por el grupo *Khornox*. Está diseñado específicamente para su uso dentro del entorno de *OpenGL*. Sus diseñadores afirman que se ha hecho un gran esfuerzo para lograr altos niveles de paralelismo. Su diseño se basa en *C* y *RenderMan* como modelo de lenguaje de sombreado.
- **CG** (C for Graphics) lenguaje propiedad de la empresa *NVIDIA* resultante de su colaboración con *Microsoft* para el desarrollo de un lenguaje de sombreado. El lenguaje está basado en el lenguaje de programación *C* y comparte varias de sus características aunque existen diferencias notables y elementos añadidos para adaptarlos a cuestiones concretas y especiales de la programación de las *GPU*. Su principal ventaja es que

puede ser usado por las *APIs OpenGL* y *DirectX*. Otra ventaja de este lenguaje es el uso de perfiles, los cuales se encargan de elegir para su ejecución el más adecuado de los programas disponibles para el *hardware*. Estos lenguajes no son totalmente independientes del hardware por lo tanto es recomendable crear programas específicos para diferentes tarjetas gráficas.

1.3.3 CONSIDERACIONES GENERALES.

En la trayectoria de este capítulo se han abordado diferentes temas que aportarán ideas para iniciar el próximo capítulo. El texturizado de objetos, el modelado multiresolución y las técnicas de visualización estudiadas fueron algunos de los temas tratados, que en su combinación darán paso a obtener la solución, alcanzando modelos realistas e interactivos.

CAPÍTULO 2. SOLUCIÓN PROPUESTA.

2.1 TÉCNICAS DE NIVELES DE DETALLES.

En el epígrafe 1.1.1 se caracterizaron diferentes técnicas para definir el nivel de detalle de un objeto en una escena virtual. Una manera de proporcionar dichos detalles es a través de los *LOD* automáticos, los mismos a través de uno o más parámetros de entrada, ajustan la complejidad geométrica de los objetos [6].

En la solución que se propone se define como parámetro de entrada para establecer los niveles de detalles con el cual se visualizará el objeto a la distancia que existen entre el modelo tridimensional visualizado en la escena y el observador (cámara virtual). Se fundamenta la selección de la distancia como parámetro de entrada ya que es el que está definido en la herramienta *CAD* (Diseño Asistido por Computadoras) que se utiliza para el diseño de los modelos de los escenarios que representan los diferentes paseos virtuales. Dicha herramienta exporta en un fichero los diferentes niveles de detalles (malla de triángulos) de acuerdo a los rangos de distancias definidos durante la etapa de diseño de los modelos. Ver Fig: 9.

Es válido destacar que aunque los *LODs* permiten establecer muy buena relación entre velocidad y complejidad no tienen en cuenta el realismo con que se visualizan los modelos en el entorno virtual.

Fig: 9 Niveles de Detalles.

2.2 NIVEL DE REALISMO DE MODELOS.

En el capítulo anterior se estudiaron diferentes técnicas para lograr una visualización realista de un modelo en un entorno virtual, haciendo especial énfasis en las técnicas de visualización basadas en mapas de texturas. Las cuales tienen como idea central otorgar realismo a un objeto a partir de la aplicación de una textura.

Entre las técnicas más usadas en la bibliografía consultada se destacan: *bump mapping, normal mapping* y *parallax mapping*, las cuales presentan como principales desventajas que sacrifican el nivel de interacción y la velocidad de visualización por lograr una representación realista de los modelos; sin embargo, alcanzan una gran importancia debido a que mediante su uso le proporcionan a los objetos un elevado nivel de realismo a través de las características de relieve que le imprimen a estos objetos.

2.3 ALGORITMO PROPUESTO.

Atendiendo a las principales desventajas de los *LODs* y las técnicas de Mapas de Texturas expresadas en los epígrafes 1.1.1 y 1.2.2, se propone una solución que busca mediante la combinación de los conceptos fundamentales de ambos enfoques obtener un algoritmo que permita visualizar de forma realista y en tiempo real los modelos de los paseos virtuales de forma que el usuario posea un elevado nivel de interacción con los mismos.

El algoritmo propuesto cuenta de dos etapas fundamentales. La primera es la etapa donde se definen los diferentes niveles de detalles para los modelos; la cual se realiza en tiempo de diseño, como se explicó en el epígrafe 2.1, por lo que no influye en la complejidad computacional de la solución. Por lo que el enfoque estará dado en la segunda etapa, en la cual se decide que técnica de visualización se deberá aplicar al objeto de acuerdo al nivel de detalle con el que se esté visualizando el modelo en el entorno virtual.

El algoritmo que se propone, ver esquema de la Fig: 10 se basa en cargar el modelo *3D*, utilizando para dicha acción la herramienta conveniente para el usuario, según sus criterios y necesidades. A través de la técnica de Niveles de Detalles, donde se considerará la distancia que existe entre el observador y el objeto se obtendrán tres versiones diferentes del mismo. La primera representación del objeto, siendo el más cercano al observador tendrá un alto nivel de detalle y se le aplicará la técnica de *Shaders Bump Mapping.* En una segunda representación

del objeto y estando a una mediana distancia del observador se le aplicará solo una textura, sin requerir el mismo detalle de la primera representación. La tercera salida del objeto, aun más alejado del observador tendrá un bajo nivel de detalle y se le aplicarán solamente propiedades del material, como sería el caso de colores, no siendo necesaria la representación de un objeto con una geometría compleja, donde los detalles no serían distinguidos.

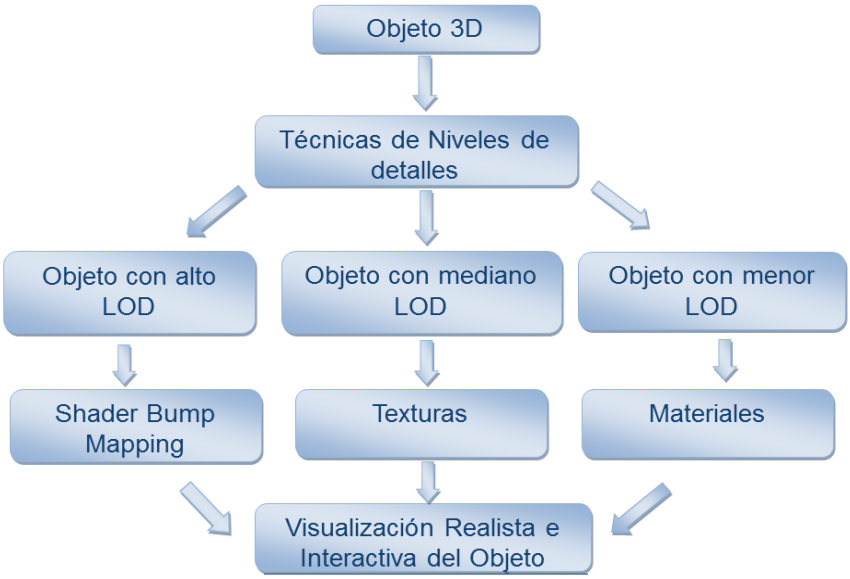

Fig: 10 Representación del Algoritmo de Niveles de Detalles para la Visualización de Modelos 3D

2.3.1 SHADERS BUMP MAPPING.

Después del estudio realizado en el capítulo anterior se llegó a la conclusión que la técnica *Bump Mapping* aumentará considerablemente su calidad visual sin necesidad de aumentar el nivel de detalle de los modelos incorporando más polígonos. Su uso es considerado como una de las técnicas más manejables por los usuarios, proporcionándoles efectos de profundidad y rugosidad a una superficie plana.

La técnica de *bump mapping* se basa en la implementación de un modelo de iluminación donde las normales de la superficie de los objetos son modificadas a partir de un mapa de normales especificado por el usuario.

2.3.1.1 MODELO DE ILUMINACIÓN.

En el artículo "*Bump Mapping with GLSL*" se propone el uso del modelo de iluminación *Blinn-Phong* [28]; teniendo en cuenta que el color ambiente en la mayoría de los casos será una constante, además de que el color difuso depende de la amplitud del ángulo formado por la dirección de la fuente de luz y la normal de la superficie y donde el color especular estará dado por la amplitud del ángulo establecido entre el ángulo de visión y la normal de la superficie; considerándose las siguientes ecuaciones:

```
pixelColor= Ambient + (Diffuse + Specular) * Shadow

Donde:
Ambient = ambientMaterial * ambientLight


Diffuse = diffuseMaterial * diffuseLight * lamberFactor
lamberFactor = max (dot (lightVec, normal), 0.0)

Specular = specularMaterial * specularLight * speculatCoef
speculatCoef = pow (max (dot (halfVec, normal), 0.0), shininess)
```

Fig: 11 Modelo de Iluminación de Blinn-Phong.

Partiendo del modelo anteriormente explicado se propone realizar una simplificación del mismo que consiste solo en considerar la amplitud del ángulo definido entre la normal de la superficie y la dirección de la fuente de luz como el valor de la intensidad de iluminación de la superficie en ese punto. De modo que el mismo quedaría definido por la siguiente ecuación:

(1) $I(x, y, z) = \max(\text{bumpNormal(x, y, z)} * \text{lightDir}, 0.0)$

Donde I es la intensidad de la luz en el vértice con coordenadas *x,y,z*; ***bumpNormal*** es la normal modificada definida por el usuario en el mapa de normales y ***lightDir*** es el vector dirección de la luz de la escena.

Luego, al multiplicar la intensidad de luz por el color del vértice entonces se obtendrá el color final de mismo, de forma que aquellos vértices cuyas normales forman un ángulo muy grande

con respecto a la luz recibirán menos influencia de la fuente de luz y por ende se verán oscuras en la escena final. Ver Fig: 12.

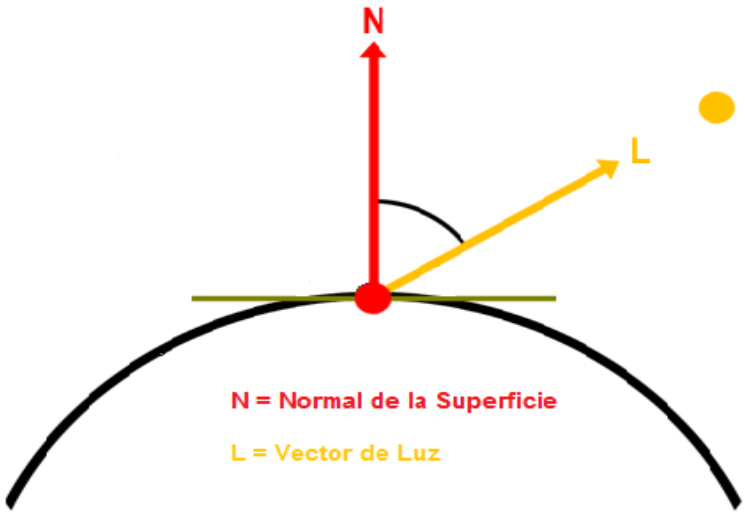

Fig: 12 Simplificación del Modelo de Iluminación de Blinn-Phong.

2.3.1.2 ESPACIO TANGENTE.

Una vez seleccionado el modelo de iluminación solo queda transformar la normales definidas por el usuario en el mapa de normales a un sistema de coordenadas que sea relativo a la superficie del modelo, esta transformación se le conoce en la bibliografía consultada como *espacio tangente,* el cual consiste en encontrar los vectores *binormal, tangente y normal* a la superficie del modelo, ver Fig: 13 y definiendo como matriz la que se muestra en la Fig: 14:

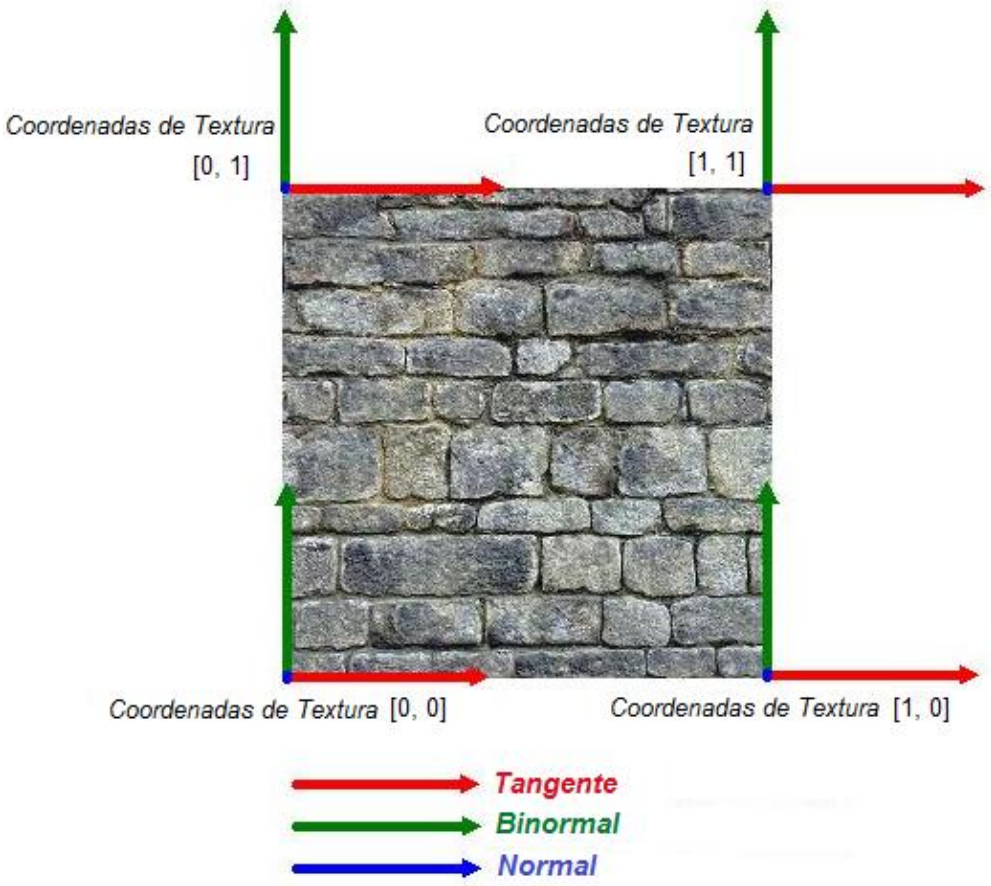

Fig: 13 Representación del espacio tangente.

```
[   Tangent.x   Tangent.y   Tangent.z   ]
[   BiNormal.x  BiNormal.y  BiNormal.z  ]
[   Normal.x    Normal.y    Normal.z    ]
```

Fig: 14 Matriz TBN.

A través de las siguientes ecuaciones se calcularon los diferentes vectores de la matriz *TBN* en la implementación de la técnica *bump mapping* propuesta, mediante las coordenadas de los vectores, como se ilustra en la Fig: 15.

```
CVector T, B, N;
T = CVector((c3c1_B * v2v1.x - c2c1_B * v3v1.x) * fScale1,
    (c3c1_B * v2v1.y - c2c1_B * v3v1.y) * fScale1,
    (c3c1_B * v2v1.z - c2c1_B * v3v1.z) * fScale1);

B = CVector((-c3c1_T * v2v1.x + c2c1_T * v3v1.x) * fScale1,
    (-c3c1_T * v2v1.y + c2c1_T * v3v1.y) * fScale1,
    (-c3c1_T * v2v1.z + c2c1_T * v3v1.z) * fScale1);

N = T.CrossProduct(B);
```

Fig: 15 Cálculo de los vectores: tangente, binormal y normal.

A partir de lo mencionado anteriormente se propone la implementación de este modelo de *bump mapping* sobre la *GPU,* la cual se detalla en las siguientes figuras, Fig: 16 y Fig: 17:

```
attribute vec3 tangent;
attribute vec3 binormal;

uniform vec3 lightPosition;

varying vec2 texCoord;
varying vec3 lightDirection;

void main(void)
{
   texCoord = gl_MultiTexCoord0.xy;
   lightDirection = normalize(lightPosition - gl_Vertex.xyz);

   mat3 tbnMatrix = mat3(tangent, binormal, gl_Normal);
   lightDirection = mul(tbnMatrix, lightDirection);

   gl_Position = ftransform();
}
```

Fig: 16 Vertex Shaders

El siguiente fragmento de programa muestra la implementación de la etapa de asignación de colores a los *píxeles*, esta etapa recibe como parámetros la textura original (*srcTexture*), la cual

da el color del vértice con las coordenadas de texturas especificadas por la variable (*textCoord*) y el mapa de normales enviado al *fragment shader* en forma de textura (*bumpTexture*) la cual contiene en la posición (*texCoord*) la normal del vértice modificada o definida por el usuario. Luego con estos dos parámetros y aplicando el modelo de iluminación simplificado y descrito anteriormente se obtiene el color final del fragmento y por ende el efecto visual de relieve en la textura (del inglés *bump mapping*).

```
uniform sampler2D srcTexture;
uniform sampler2D bumpTexture;

varying vec2 texCoord;
varying vec3 lightDirection;

void main(void)
{
    vec4 decalColor = texture2D(srcTexture, texCoord);
    vec4 bumpNormal = texture2D(bumpTexture, texCoord);

    float NdotL = max( dot(bumpNormal.xyz,lightDirection), 0.0);

    gl_FragColor = decalColor * NdotL;
}
```

Fig: 17 Fragment Shaders.

2.3.2 TEXTURIZADO DE OBJETOS.

Las texturas pueden ser unidimensionales, bidimensionales o tridimensionales. Matemáticamente se ha definido que el hecho de asignarle a un objeto una textura consiste en definir una función univoca, de mapeado o *mapping*, donde a cada punto del espacio ocupado por la superficie o sólido se le hace corresponder un punto en el espacio donde está definida la textura. No pudiéndose definir de manera inversa (como una función del espacio de textura al espacio de objetos), debido a que es usual que un mismo punto de la textura pueda estar representado varias veces en el espacio objeto.

El avance de las aplicaciones *3D* ha traído consigo un mayor uso de texturas *2D*. Cuando una porción de la textura *2D* es mapeada sobre la superficie del objeto *3D*, se le llama mapeo de

texturas, tal y como se muestra en la Fig: 18, y donde se han definido tres etapas en el proceso de texturización.

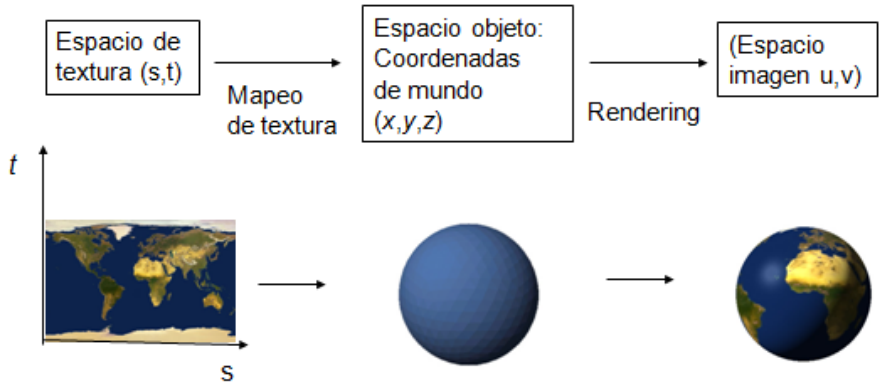

Fig: 18 Proceso de Texturización

Etapas del proceso de Texturizado:

1. Mapear las coordenadas (se calculan los *texels* del espacio textura (s,t) y los píxeles del espacio objeto (x,y,z) a texturizar. Consiguiendo determinar que *texels* de la textura deben ser dibujados y cuál es su píxel correspondiente dentro del objeto.
2. Aplicar la textura y corregirla en caso necesario (mediante un filtrado de textura).
3. Modificar valores de Iluminación (aplicando efectos de mapas de textura).

2.3.3 MATERIAL.

El concepto de los materiales puede estar dado por la información de propiedades de un objeto, color ambiente, difuso y especular, la emisión y el brillo, así como la combinación del color ambiente, difuso y especular. La declaración de un material se propone mediante este algoritmo:

1. Habilitar función del material.
2. Pasar a la función los parámetros (cara, tipo_propiedad)

Donde:

Cara: determina la cara del objeto en la que se aplican las propiedades, se pueden definir los valores de la siguiente manera: al frente, atrás o ambas.

Tipo Propiedad: indica la propiedad a utilizarse para la cara del objeto seleccionada, las mismas se nombran:

- Color ambiente: establece la forma en que el objeto refleja la luz ambiental que recibe.
- Color difuso: establece el color del objeto.
- Color especular: establece el color del brillo del objeto.
- Emisión: hace que parezca que los objetos emiten una luz de un color determinado, independientemente de la luz recibida.
- Brillo: constituye la exponente de brillo del objeto.
- Colores indexados: se hace una combinación del color ambiente, difuso y especular.

3. Modificar valores de la propiedad que trae por defecto para obtener la deseada.
4. Deshabilitar la función del material.

2.4 ENTORNO DE DESARROLLO.

Herramienta de desarrollo.

La herramienta de desarrollo utilizada es *Visual Studio,* siendo un entorno de desarrollo integrado *(IDE)* para sistemas operativos *Windows*. La misma soporta varios lenguajes de programación tales como *Visual C++*, *Visual C#*, *Visual J#*, *ASP.NET* y *Visual Basic.NET*, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros. Esta herramienta cuenta con pequeñas herramientas auxiliares de completamiento de código como es el *Visual Assist* la cual ayuda para la codificación de la aplicación y otras, así como el *VTune* que le permite al desarrollador ir comprobando el rendimiento en cuanto a memoria y uso del *CPU* de la aplicación que se está codificando.

Lenguaje de programación.

El lenguaje de programación utilizado es C++, creado con la intensión de extender el famoso lenguaje de programación *C* con un mecanismo que permita la manipulación de objetos. Suele

decirse que el *C++* es un lenguaje de programación multiparadigma, ya que este lenguaje de programación abarca tres paradigmas de la programación, la programación estructurada y la programación orientada a objetos y posteriormente se añadió la programación genérica. Las principales características *del C++* son las facilidades que proporciona para la programación orientada a objetos y para el uso de plantillas o programación genérica *(templates)*. *C++* está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel. Por parte de la programación del *GPU* se utilizó *GLSL (OpenGL Shading Language),* siendo este compatible con la *API* gráfica *OpenGL.*

CAPÍTULO 3.ANÁLISIS DE LOS RESULTADOS OBTENIDOS.

En el presente capítulo dada una selección de los artículos más referenciados en la bibliografía consultada se exponen métodos, algoritmos o avances con el propósito de establecer comparaciones entre los mismos y entre el algoritmo en cuestión que se desarrolla en el presente trabajo a través de sus principales resultados y contribuciones. Estos están relacionados con el tema de las Técnicas de Texturizado de Objetos y las de Niveles de Detalles.

Tradicionalmente, las superficies de los objetos virtuales se han caracterizado por ciertos parámetros como su color, opacidad o reflectividad. A estas características también se le añaden texturas, adicionando matices a objetos que presentan un bajo detalle geométrico. Los dispositivos de aceleración 3D actuales pueden utilizar varias imágenes de forma simultánea para un polígono. Siendo este soporte tecnológico un gran paso de avance para simular efectos complejos mediante técnicas de texturizado, consiguiendo materiales con imperfecciones, reflejos, suciedad o relieve, y efectos de iluminación avanzados como reflexión, refracción o radiosidad [18]; de aquí la importancia que las técnicas de texturizado tienen al conseguir los resultados finales con grandes dotes de realismo.

Una manera de mejorar el rendimiento del render y la calidad visual es asignándole más polígonos y mayor textura a los modelos de mayor nivel de detalle, y a un modelo de menor detalle menos polígonos y una textura pequeña. De manera que si se le dibuja menos polígono cuando los objetos están más lejos de la cámara aumenta la velocidad y aplicándole mayor detalle a los objetos que se encuentran cerca de la cámara se tiene una mayor calidad visual.

3.1 ARTÍCULOS MÁS REFERENCIADOS. BASADOS EN TÉCNICAS DE TEXTURIZADO DE OBJETOS.

1. *Aplicaciones Avanzadas del uso de Texturas Precomputadas en Entornos de Simulación en Tiempo Real.*

La publicación de este artículo se realizó en el año 2003 en España, en una Revista de Ingeniería Industrial por los autores *Iñaki Ayucar*, *Alejandro García Alonso* y *Luis Matey* [18].

Planteándose como objetivos fundamentales del estudio realizado el análisis detallado y las posibles aplicaciones de una serie de técnicas de última generación que posibilitan la representación de objetos entiempo real con dotes de realismo añadidos mediante el uso de multitexturas. Entre las técnicas abordadas en este artículo se presentaron los *Lightmaps* (mapas de luz) y el *Bump Mapping*.

Como resultados del estudio realizado por este grupo de desarrolladores acerca de estas y otras técnicas presentes en el artículo; implementaron un sistema, el cual apoyándose en el uso de texturas precomputadas (calculadas con anterioridad) se desarrolló satisfactoriamente, sin que la complejidad de almacenamiento en texturas de la iluminación concerniente a una escena afectara su representación en tiempo real.

Fig: 19 Resultado de una escena iluminada mediante lightmaps.

2. *Real-Time Bump Map Synthesis.*

La realización de este artículo está dado por los autores *JanKautz*, *Wolfgang Heidrich* y *Hans-Peter Seidel*, pertenecientes a una Universidad de Canadá, en el año 2001 [21].

En dicho artículo se presenta un método que sintetiza automáticamente mapas de relieve en niveles de detalle arbitrarios en tiempo real. Requiriendo como únicos datos de entrada una Función de Densidad Normal (*FDN*), la cual describe la distribución de probabilidad de las microfacetas de diferentes orientaciones, usada para generar el mapa de relieve y también para dar sombra al mapa de relieve generado.

Mediante la aplicación de este método se pretende que en lugar de dibujar un mapa de relieve, que sólo sería válido para un nivel de detalle, se logre sintetizar un mapa de relieve en niveles de detalles arbitrarios de un modelo de reflexión dado. La técnica permite hacer zoom infinito en la superficie, debido a que más detalles se pueden crear cuando esté más cerca, donde el sombreado será más consistente y bajo la suposición de que la superficie rugosa es fractal y que la reflexión del modelo se basa en microfacetas (muchos pequeños parches de superficie especular).

Cuando el observador se acerca al mapa de relieve, se crean más arrugas en frecuencias más altas, manteniendo la distribución de la normal del modelo de sombreado; véase la Fig: 20 y donde no importa lo cerca de un zoom, las microfacetas siempre tendrán la misma distribución.

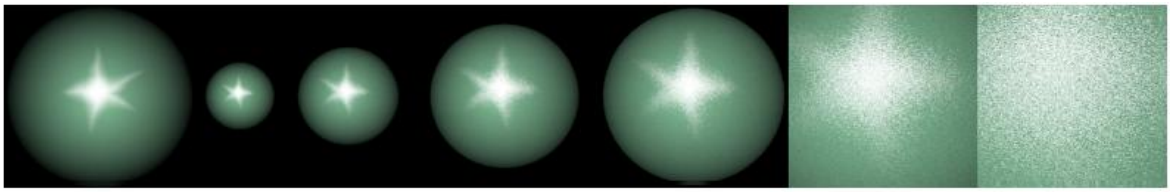

Fig: 20 Representación del Bump mapping a diferentes niveles de detalles y sombreado generados con la distribución de la normal como se muestra a la izquierda.

Una característica muy específica y particular de este método es que sintetiza un detalle más fino de una descripción muy gruesa; siendo muy diferente a la mayoría de los métodos existentes de síntesis de textura, los cuales se basan fundamentalmente en crear de una imagen pequeña una imagen grande [22]; entre los que se pueden mencionar: Método de Efros y Leung [23], de Ashikhmin [24], de Efros y Freeman [25].

Destacando las principales contribuciones de este método se puede decir que sintetiza de forma interactiva los mapas de relieve en niveles de detalle arbitrario de una función de densidad normal. Además de ser un algoritmo de renderizado que realiza el sombreado en la generación de mapas de relieve generado, con un modelo de reflexión basado en la misma función de densidad de la normal. Algunos de los resultados que se plasman en este artículo, obtenidos mediante la aplicación de esta técnica se muestran a continuación.

En la Fig: 21 se visualiza una tetera, mediante un software que utiliza la técnica de *Síntesis del Bump Mapping en Tiempo Real* con una *FDN* anisotrópica. A la izquierda se observa la

representación de la *FDN*, las tres imágenes centrales muestran representaciones a diferentes distancias. La imagen de la derecha muestra lo que sucede si las normales se distribuyen de manera uniforme y no en función de la *FDN*.

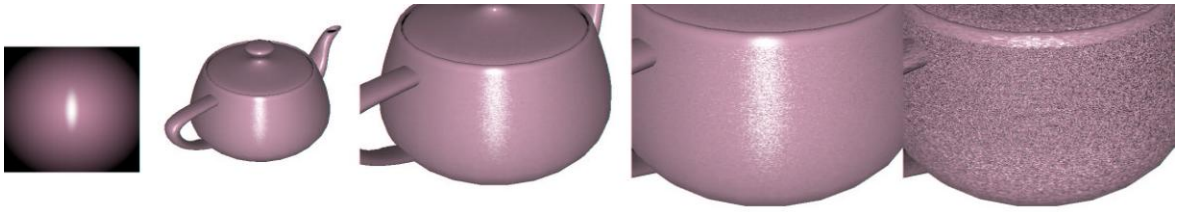

Fig: 21 Tetera con generación automática de mapas de relieve usando *FDN*, comenzando por la izquierda, a continuación tres imágenes vistas a diferentes distancias y la última muestra cuando no se utiliza FDN.

La Fig: 22 representa una tetera, realizada con una *FDN* isotrópica y usando una tarjeta gráfica *NVIDIA GeForce 3*. Este modelo cuenta con 17000 triángulos. Donde la generación del mapa de relieve y el sombreado se puede hacer en un solo paso, siendo posible hacer render en alrededor de 30 *frames* por segundo, evidenciándose la calidad del resultado.

Fig: 22 Imagen generada a razón de 30 frames por segundo a través de una tarjeta gráfica *NVIDIA GeForce 3.*

3. Generación en Tiempo Real de Pequeñas Deformaciones Debido a Colisiones Utilizando Bump y Normal Mapping.

Este artículo fue publicado a raíz de la tesis de grado anteriormente realizada por los autores *Erislandys Igarza Castro* y *Yury Antonio Rodríguez Cruz* en la Universidad de las Ciencias Informáticas (UCI), Cuba en el año 2009 [19].

En el mismo se presentó un método que generaba pequeñas deformaciones sobre superficies de objetos sólidos causados por colisiones, sin modificar la geometría de los mismos y explotando para ello las técnicas *bump mapping* y *normal mapping*. Resultando adecuado este método para su aplicación en los videos juegos, donde se requieren importantes cotas de realismo, haciéndolos más creíbles y dinámicos; realizando los cálculos en tiempo real utilizando la Unidad de Procesamiento Gráfico.

Al hacer uso en este trabajo de las técnicas de texturizado, específicamente el *bump mapping* se puso en práctica teniendo en cuenta que para la representación de cada objeto sería necesario un *bump map* o mapa de relieve, tal y como exige el algoritmo, encontrándose el mismo en varios puntos de la geometría. Donde para el correcto funcionamiento del método que se presenta, es preciso tener un *bump map* y una textura para cada polígono. De ahí que se tendría un inconveniente, dado que si los diseñadores aplican la reutilización de texturas y por consiguiente el mismo *bump map* en varios puntos del modelo, se tendrían varios puntos de la geometría asociada a un único *texel*, como se muestra en la Fig: 23a; de esta forma en caso de colisión, al modificar el *bump map*, se afectarían visualmente todos los lugares donde ha sido mapeado, obteniendo un resultado insatisfactorio. Ante este problema se planteó como solución la realización del mapeado de textura, pero utilizando una única textura y un único *bump map* para cada geometría, como se muestra en la Fig: 23b.

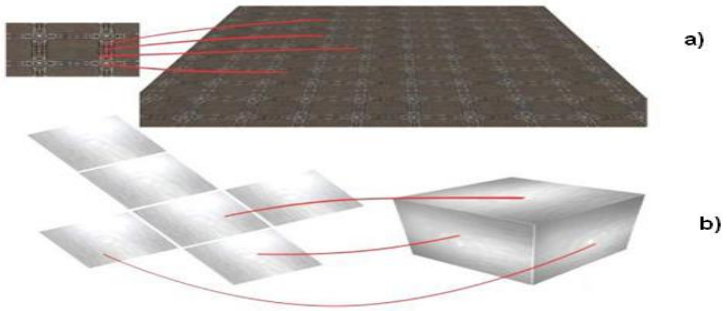

Fig: 23 Texturizado para el algoritmo a) incorrecto b) correcto.

Al haber calculado la deformación que sufre un objeto al ocurrir una colisión en el *GPU*, se actualiza el *bump map* de los objetos, ejecutándose esta actualización en el *CPU*, la cual se puede realizar de variadas formas, en dependencia del motor grafico con que se esté trabajando. Los resultados que se obtuvieron en la deformación de objetos debido a una colisión y sin que fuese necesario alterar la malla del mismo para lograr esta acción se demuestran en la Fig: 24.

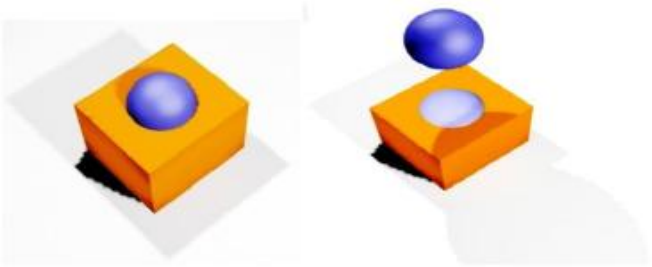

Fig: 24 Deformación de un objeto causada por una colisión.

La técnica de *normal mapping* se introduce en este trabajo por la necesidad que la visualización de la deformación en tres dimensiones cuente con el realismo esperado, done no es suficiente actualizar el *bump map*; donde habría que tomar cada píxel y compararlo con los píxeles adyacentes, para de esta manera hallar una normal que pueda ser utilizada para cambiar el brillo generado por las luces en un punto de la superficie, incorporando el detalle necesitado. Pero como la técnica de *normal mapping*, como evolución de la de *bump mapping* ha venido a suplir estas deficiencias, entonces su aplicación es muy efectiva, contando con tres canales *RGB*. Así, solo resulta necesario ejecutar el cálculo una sola vez para una escena, o al menos solamente cuando esta cambie; por lo que esta técnica fue utilizada en la fase de visualización de la deformación.

Como el proceso de cálculo puede incluso ser costoso si las imágenes son de alta resolución se utiliza la técnica de los mapas de direcciones para actualizar el *normal map*, permitiendo que solo se actualice el *normal map* solo en aquellos *texels* donde haya ocurrido la deformación.

Fig: 25 Actualización del normal map solo en los texels deformados.

Algunas de las fortalezas que se destacan mediante la aplicación de este método es que se pueden lograr deformaciones sin necesidad de transformar la geometría, logrando así eficiencia computacional. Así como que la utilización del *GPU* para el cálculo de la deformación permitiendo aligerar la carga de la *CPU*.

Es válido mencionar que la propuesta de utilizar el *bump mapping* puede tener privaciones, tal es el caso de que una deformación muy grande que implique cambio en el nivel de la macro-estructura no debería ser modelada por el método de deformación antes propuesto, pues existe un límite teórico marcado por la máxima deformación que se puede representar con un *bump map*.

Sin duda alguna se destaca que el principal aporte de este método fue la inclusión de la técnica de *normal mapping* al algoritmo propuesto por *Morgan McGuire*, con la cual se evita el constante cálculo de las normales luego de la actualización del *bump map*. Resultando un refinamiento al algoritmo con una evolución de la propia técnica utilizada por *Morgan*.

3.2 ARTÍCULOS MÁS REFERENCIADOS. BASADOS EN TÉCNICAS DE NIVELES DE DETALLE.

En la simplificación poligonal la idea es reducir el número de triángulos, vértices, bordes, huecos, túneles o cavidades, normalmente agrupando puntos y haciendo necesario casi siempre una nueva triangulación. Utilizando el colapso de bordes o el colapso de triángulos no es necesaria una nueva triangulación debido a que trabaja directamente sobre la malla poligonal. La idea es remover puntos pertenecientes al mismo triángulo y conservar la triangulación de los triángulos vecinos. En los artículos que se mencionarán a continuación se verán los resultados obtenidos mediantes algoritmos de multiresolución.

1. *Multiresolución Adaptativa de Mallas Triangulares Basado en Criterio de Curvatura.*

Este artículo fue publicado en el 2007 por *Alexander Ceballo*, *Jorge Hernández* y *Flavio Prieto*, en el se aplica *Colapso de Bordes* a los objetos con el objetivo de simplificar la malla [26].

El colapso de bordes ofrece una variedad de ventajas como son:

- La posición del nuevo vértice puede ser escogida con libertad, el nuevo vértice puede ser ubicado en el medio del borde, en un extremo e incluso en la mitad de un triángulo.
- La posibilidad de optimización escogiendo qué vértices son removidos dadas las características deseadas.

- No es necesaria una nueva triangulación, la malla varía sobre si misma al removerse los puntos con los bordes y triángulos ligados a ellos.
- Simplicidad, rapidez y efectividad, no se requiere gran cantidad de cálculos para escoger la nueva posición del punto, se trabaja directamente sobre los datos originales sin necesidad de transformaciones ni funciones complejas.

Para colapsar un borde se tienen cuenta dos vértices que forman un borde (puntos vecinos en el espacio), uniéndolos en uno solo. Además que se borran los dos triángulos adyacentes a cada borde. Obteniendo reducir el número total de puntos en uno y el de triángulos en dos. El colapso de bordes posee tres versiones, en las cuales la idea es remover puntos pertenecientes al mismo triángulo conservando la triangulación de los triángulos vecinos, ya sea eliminando uno de los dos vértices y dejando la posición original del otro o dándole una nueva posición en el medio de ambos vértices, como se puede apreciar en la Fig: 26.

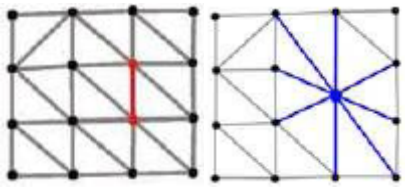

Fig: 26 a) Malla original b) Malla resultante

Después de haber aplicado varias iteraciones a un modelo se puede observar que en la cuarta iteración y con el 10% de triangulación del modelo original, la calidad visual del mismo sigue siendo aceptable. Pero sin embargo se va degradando notablemente, como se muestra en la Fig: 27.

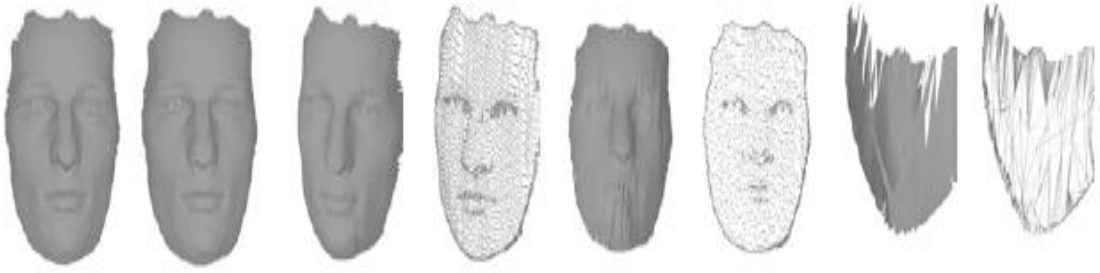

Fig: 27 Colapso de bordes

(a) Modelo original, 152970 triángulos	(b) Cuarta iteración, 15653 triángulos, 10%	(c) Sexta iteración, 6563 triángulos, 4%	(d) Octava iteración, 1300 triángulos, 0.85%

2. *Multiresolución Adaptativa de Mallas Triangulares Basado en Criterio de Texturas.*

Este artículo fue publicado en el 2007 por los mismos autores del artículo anterior *Alexander Ceballo*, *Jorge Hernández* y *Flavio Prieto*, en el se aplica *Colapso de Triángulos* a los objetos con el objetivo de optimizar la malla sin necesidad de una nueva triangulación, la malla varía sobre sí misma al removerse los puntos con los bordes y triángulos ligados a ellos, logrando simplicidad, rapidez y efectividad [27].

La manera de colapsar un triángulo es uniendo sus tres vértices en uno, Al desaparecer el triángulo, lo hacen también los tres triángulos adyacentes a sus lados, es decir que el número total de puntos decrece en dos, mientras el de triángulos lo hace en cuatro; como se puede ver en la siguiente representación.

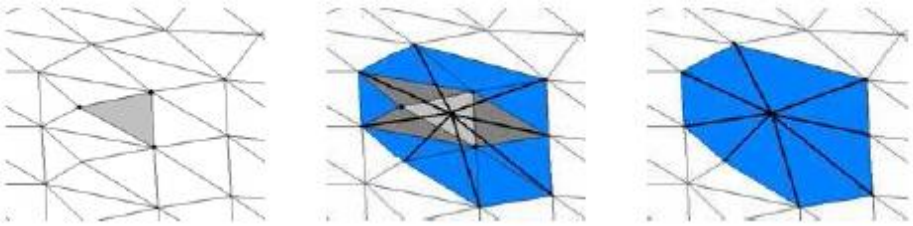

Fig: 28 a) Triángulo a remover b) Nuevo vértice c) Triángulo resultante

Para colapsar un triángulo se tiene que tener en cuenta cuáles puntos pueden ser unidos, considerando los siguientes casos:

- Colapsando todos los vecinos en un solo ciclo, y teniendo en cuenta que cada punto pertenece a más de un triángulo, no se simplifica la malla, lo único que se consigue es cambiar la posición espacial de los puntos; sin embargo, la complejidad de la malla es similar.
- Colapsando iterativamente todos los triángulos, se obtiene un punto al final, debido a que cada punto nuevo pertenece a nuevos triángulos; si no se mantiene información de cuáles ya han sido removidos se obtiene un resultado parecido al de la Fig: 29.

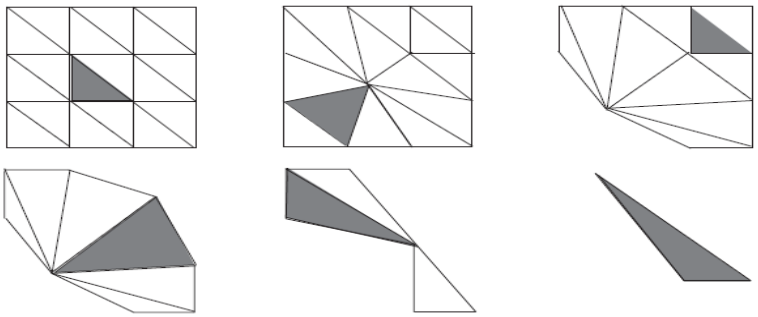

Fig: 29 Colapso iterativo de triángulos.

El colapso de triángulos es el método de multiresolución implementado que mejores resultados arroja como se puede ver en la Fig: 30; en la cuarta iteración con el 6% de los triángulos la calidad visual sigue siendo relativamente buena. La malla se degrada de forma regular al mantenerse en proporción el ancho y alto de los triángulos, lo que hace que se logre un mayor grado de decimación con mejores resultados [27].

Fig: 30 Colapso de triángulos.

(a) Modelo original, 152970 triángulos (b) Cuarta iteración, 8859 triángulos, 6% (c) Sexta iteración, 3045 triángulos, 2% (d) Octava iteración, 1946 triángulos, 1.3%

Luego de haberle aplicado ambos algoritmos a la imagen a un rostro que posee 152970 triángulos, mantiene una buena visualización de la imagen, pero a medida que este proceso de iteración va aumentando la calidad decae, sin embargo al aplicarle el mismo procedimiento a la imagen pero con el colapso de triángulos y la misma cantidad de iteraciones se puede apreciar que la calidad se mantiene bastante estable en comparación con el algoritmo anterior.

3.3 RESULTADOS DEL ALGORITMO DE NIVELES DE DETALLES PARA LA VISUALIZACIÓN DE MODELOS 3D.

Una vez implementado el algoritmo propuesto pasamos al proceso de comparación con el propósito de hacer una valoración al respecto de cuan aplicable es la solución que se presenta al problema planteado.

La Tab: 1 muestra una comparación entre las principales características de los artículos más referenciados anteriormente en los epígrafes 3.1 y 3.2; siguiendo los siguientes aspectos, calidad visual al aplicar dichos algoritmos o técnicas, además de la interactividad resultante de los modelos visualizados, así como el hardware utilizado para realizar el proceso de visualización.

Técnicas / Algoritmos	Calidad Visual	Nivel de Interacción	Hardware Utilizado
Técnicas de Niveles de Detalle.	Baja	Alto	CPU
Técnicas de Texturizado de Objetos.	Alta	Bajo	GPU
Algoritmo Propuesto	Alta	Alta	CPU y GPU

Tab: 1 Comparación entre las Técnicas de Niveles de Detalle, las Técnicas de Texturizado de Objetos y el Algoritmo Propuesto.

Los resultados que se obtuvieron al realizar la demostración del algoritmo propuesto fueron satisfactorios; obteniéndose un equilibrio entre las Técnicas de Niveles de Detalles y las Técnicas de Texturizado de Objetos. Mediante este algoritmo se consigue que el modelo visualizado esté dotado de realismo e interactividad.

En la siguiente figura se muestra una representación del resultado obtenido con el algoritmo propuesto, donde se visualiza el modelo en tres versiones, indicando la distancia existente entre cada una de estas representaciones y la disminución de polígonos, así como la técnica

de texturizado aplicada de acuerdo a la distancia que se encuentra el observador (cámara virtual).

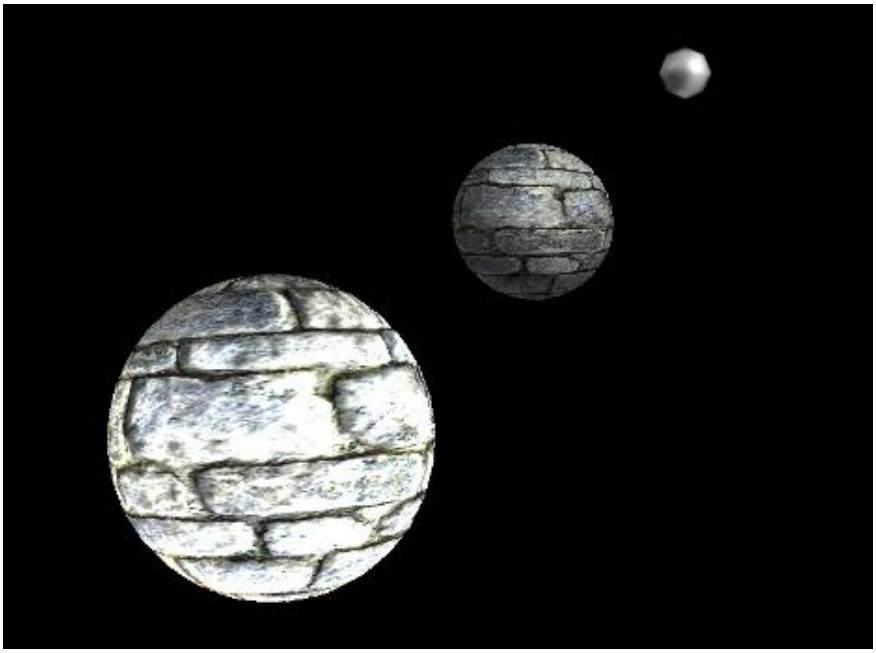

Fig: 31 Resultados del Algoritmo Propuesto.

CONCLUSIONES

Con el desarrollo de este trabajo se alcanzaron los objetivos formulados en el mismo, obteniendo las siguientes conclusiones:

Las Técnicas de Niveles de Detalle proporcionan en la visualización de las escenas tridimensionales un alto rendimiento e interactividad, pero no garantizan el realismo de la misma. Sin embargo las Técnicas de Texturizado de Objetos responden a la carencia de realismo que presentan las técnicas *LOD,* necesitando para esto una mayor cantidad de polígonos, lo que trae consigo un bajo rendimiento de la aplicación. Obteniéndose con la combinación de ambas técnicas un alto nivel de realismo e interactividad en la escena tridimensional.

RECOMENDACIONES

Toda investigación siempre requiere una continuación, por lo que se recomienda:

- Proponer a los interesados de esta investigación científica utilizar otras técnicas de texturizado de objetos que contribuyen al realismo de los modelos tridimensionales.
- Aumentar el número de versiones de un modelo para evitar los posibles saltos bruscos entre una y otra representación.
- Extender el uso del algoritmo propuesto a proyectos y áreas de trabajo que visualicen grandes cantidades de datos en una escena tridimensional y pudiéndose adaptar y mejorar según sus necesidades y facilidades para la interacción con el mismo.

BIBLIOGRAFÍA

[1] Robb, R.A. Three- Dimensional Visualization in Medicine and Biology. San Diego, CA : s.n., 2000.

[2] Andujar, C. Simplificación de Modelos Poliédricos. Catalunya : s.n., 1998.

[3] Heckbert, P.S., Garland, M. Multiresolution Modeling for Fast Rendering. Banff Alberta Canada : s.n., 1994.

[4] Sánchez, Y. A. (Junio de 2008). Visualización 3D de datos geoespaciales.Ciudad de La Habana.

[5] Crespo, Javier LLuch. Modelos multiresolución para la aceleración de la visualización de entornos naturales. Valencia : s.n., 2002.

[6] Mario Ezequiel, Guzmán García. Uso de Tecnologías de Hardware Gráfico en el Apoyo al Realismo en Entornos Virtuales Arquitectónicos. 2004.

[7] http://sabia.tic.udc.es/gc/teoria/texturas/Mapeado_de_Texturas-Filtros.htm. [Online]

[8] http://www.ogre3d.org. [Online]

[9] http://graphics.cs.brown.edu/games/SteepParallax/index.html. [Online]

[10] Gustems, Javier Boó. Visualización interactiva de entornos muy complejos. 2007.

[11] http://es.wikipedia.org/wiki/Unidad_de_procesamiento_gr%C3%A1fico. [Online]

[12] http://es.wikipedia.org/wiki/GeForce#Actualizaci.C3.B3n. [Online]

[13] http://es.wikipedia.org/wiki/Shader. [Online]

[14] http://www.opengl.org/registry/specs/EXT/geometry_shader4.txt. [Online]

[15] http://www.optimizacion3d.info/libro-3d/texturas/mapeado-con-mapas-de-normales. [Online]

[16] http://www.paulsprojects.net/tutorials/simplebump/simplebump.html. [Online]

[17] Graficación y Realidad Virtual, capítulo 2.

[18] Ayucar, Iñaki, Alonso, Alejandro García and Matey, Luis. Aplicaciones Avanzadas del uso de Texturas Precomputadas en Entonos de Simulación en Tiempo Real.

[19] Yuri, Antonio, Rodriguez Cruz and Igarza Castro, Erislandys. Generación en Tiempo Real de Pequeñas Deformaciones Debido a Colisiones Utilizando Bump y Normal Mapping. 2009.

[20] Galván Zald, Julio Antonio and Romero Naranjo, Maydelis. Aplicación de Shaders de Relieve a objetos 3D en entornos de Realidad Aumentada.

[21] Kautz, Jan, Heidrich, Wolfgang and Seidel, Hans-Peter. Real-Time Bump Map Synthesis. British Columbia : s.n.

[22] http://www.alu.ua.es/e/elf2. [Online].

[23] http://www.alu.ua.es/e/elf2/efros%20y%20leung/index.htm. [Online]

[24] http://www.alu.ua.es/e/elf2/Ashikhmin/index.htm. [Online]

[25] http://www.alu.ua.es/e/elf2/efros%20y%20freeman/index.htm. [Online]

[26] Ceballos, Alexánder, Hernández, Jorge and Prieto, Flavio. Multiresolución Adaptativa de Mallas Triangulares Basado en Criterio de Curvatura. Colombia : s.n., 2007.

[27] Ceballos, Alexánder, Hernández, Jorge and Prieto, Flavio. Multirresolución adaptativa de mallas triangulares basado en criterios de textura. 2007.

[28] Singlard, Fabien. http://www.fabiensanglard.net/bumpMapping/index.php. [Online] Abril 2010.

ÍNDICE DE FIGURAS Y TABLAS

ÍNDICE DE FIGURAS Y TABLAS

GLOSARIO DE TÉRMINOS

A

***API*:** A*pplication Programming Interface,* del español Interfaz de Programación de Aplicaciones, son métodos específicos prescritos para un determinado sistema operativo o cualquier otro programa y mediante el cual un programador escribe un programa de aplicación, pudiendo formular peticiones a ese sistema operativo o a ese otro programa.

D

***DirectX*:** colección de *APIs* creadas para facilitar las complejas tareas relacionadas especialmente para la programación de video juegos a través de la plataforma *Microsoft Windows*.

***Direct3D*:** es parte de *DirectX*. Tiene como objetivo principal facilitar el manejo y trazado de entidades gráficas elementales, como líneas, polígonos y texturas, de cualquier aplicación gráfica *3D*, además de efectuar transformaciones geométricas sobre dichas entidades y proveer también una interfaz transparente con el hardware de aceleración gráfica.

E

Espacio tangente: ángulo relativo a la superficie normal y el valor del mapa de alturas en tal punto.

F

***Frame buffer*:** memoria usada para retener uno o más *frames* para su posterior uso.

***Frames per second*:** velocidad a la que se muestran las imágenes animadas en una película o un vídeo.

M

***Mapas de bits*:** conocida también como imagen matricial o *bitmap*, es una estructura o fichero de datos que representa una rejilla rectangular de píxeles, denominada raster.

Modelo anisotrópico: representación de un objeto que presenta múltiples niveles de simplificación dependiendo del punto de vista del observador.

Modelo poligonal**:** colección de vértices y de polígonos que conectan a esos vértices.

N

Normal**:** vector tridimensional, perpendicular a la cara de un objeto, determinando la dirección en que ella apunta.

O

OpenGL: *Open Graphics Library* (Librería de Gráficos Abierta). *API* para diseñar gráficos en *2D* y *3D* creada por *Silicon Graphics en 1992*. La sorporta cualquier sistema operativo y sólo hace falta una tarjeta gráfica con soporte para *OpenGL*. Es el principal competidor de *Direct3D* de *Microsoft*.

P

Píxeles**:** contracción del inglés *textureelement*, o también *texture píxel*, constituye la unidad mínima de una textura aplicada a una superficie y usada en gráficos por computador.

Primitiva: modelos 3D que representan formas geométricas básicas y que se modifican cambiando sus parámetros preestablecidos. Elemento básico de un programa (gráfico). En programas 2D éstas son círculo, línea., en programas 3D son esfera, superficie.

Plugin**:** programas externos al programa principal, que pueden controlar todas las fases del proceso de generación de imágenes, incluyendo el modelado, la representación, la posproducción de vídeo, etc.

R

Render**:** proceso en el que la computadora interpreta todos los datos de luces y objetos y en el que crea una imagen final en el visor seleccionado. La imagen resultante puede ser fija o un cuadro de una animación.

S

***Shader*:** término que abarca las diferentes imágenes y parámetros que se pueden asignar a la superficie de un objeto geométrico 3D.