Universidad Nacional Autónoma de México

Posgrado en Ciencia e Ingeniería de la computación

Integración de propiedades de color y textura para Meta-esferas en 3D

T E S I S

QUE PARA OPTAR POR EL GRÁDO DE:

Maestro en Ciencia e Ingeniería de la Computación

P R E S E N T A:

Leonardo Gabriel Martínez Uribarren

TUTOR

Dr. Alfonso Gastelum Strozzi
Instituto de Ciencias Aplicadas y Tecnología

Ciudad de México, Marzo 2023

Dedicado

Esta Tesis está dedicada a todas aquellas personas que estuvieron a mi lado durante este episodio de mi vida. Especialmente a mis padres, que sin su apoyo esto hubiera sido sólo un sueño lejano. A mi gran amigo Ricardo Pestaña Guzmán, que gracias a tu apoyo y ánimo logré terminar este posgrado. A René Dávila, único compañero y amigo que conocí en estos años. A mi hermano, que aunque lejos, me mandabas ánimo. A mi mascota Shanti, que siempre me acompañaste en todo momento.

Índice general

Índice de figuras

Índice de cuadros

Índice de algoritmos

Resumen

Las meta-esferas son útiles para construir superficies libres de mallas, comúnmente utilizadas para la representación y simulación de fluidos, así como en la construcción de objetos de formas complejas gracias a las características que la definen, cuales resultan en bordes suaves y definidos. Sin embargo, el color en meta-esferas no ha sido objeto de estudio hasta el momento, limitando la representación de superficies solamente a la forma resultante. Este trabajo ha hecho una revisión completa de la definición de meta-esferas que ha permitido ampliarla para que sea posible introducir el color en meta-esferas y su mezcla sobre la iso-superficie resultante, el cual permite además incluir técnicas para aplicar color como lo son los mapas de color, texturizado, mapas de normales, entre otros. También ha permitido proponer un método para el calculo de vectores normales en la iso-superficie que en conjunto con la aplicación de color, permite que interactúe con un modelo de iluminación. Implementado en un sistema basado en la arquitectura CUDA.

Capítulo 1

Introducción

1.1. Contexto

Existen varias vías para construir un objeto gráfico. Una de ellas requiere que se modele el objeto partiendo de un espacio vacío, esculpiendo un objeto pre-generado o una combinación de los anteriores, haciendo uso de software de modelado profesional[37][38][39][40]. Otra forma es adquiriendo datos del mundo, utilizando sensores para obtenerlos y posteriormente discretizarlos. Otras formas hacen uso de técnicas de inteligencia artificial y algoritmos para crear información de objetos gráficos.

Para la visualización de objetos, son comunes tanto aquellos métodos que usan mallas para la organización y representación de la información, como aquellos cuyo objetivo es visualizar objetos sin la necesidad de usar polígonos como componentes para su representación, llamados métodos libres de mallas o *mesh-free*. El resultado de construir un objeto gráfico es información de vértices ordenados que forman polígonos[41] o como nube de puntos[42].

Actualmente el uso de mallas triangulares es la técnica dominante, principalmente debido al eficiente uso de sus elementos para representar superficies y facilidad de implementación. Es utilizada en campos como el cine, la televisión, videojuegos, juguetes, entre otros. Una malla está conformada por una o más caras, estas son el área resultante de agrupar tres vértices y tres aristas. Estos elementos permiten que las mallas se visualicen de forma eficaz en conjunto con técnicas de iluminación, motores de física, materiales, etcétera.

Sin embargo, técnicas para obtención de datos como LIDAR[47], fotogrametría[48], estereovisión[49], multivisión[50], entre otros, resultan en información de nubes de puntos. Es por esto que son importantes los métodos de visualización que son libres de mallas.

Una nube de puntos hace referencia a un conjunto de puntos en el espacio que representan un objeto, cuyas características son su posición y, frecuentemente, información de color y de vectores normales. Las nubes de puntos no tienen propiedades que relacionen los elementos que la componen en el espacio. Es por eso que el conjunto de estas características hace que trabajar nubes con un gran número de elementos requiera de técnicas para procesarlas, las cuales buscan reducir el número de elementos, filtrar el conjunto para reducir ruido, buscar vecindades, entre otros. Algunas aplicaciones de las nubes de puntos las podemos encontrar en la reconstrucción 3D[51], animación[52], realidad virtual y aumentada[53][54][55], robótica[56], vehículos autónomos[57], entre otros.

La visualización de una nube de puntos se puede hacer de manera directa dibujando uno a uno cada elemento, realizando una proyección a un espacio de dos dimensiones, transformando la nube a una malla o construyendo una superficie utilizando una técnica como la de meta-esferas.

Las meta-esferas son una técnica para construir superficies libres de mallas a través del uso de nubes de puntos, a partir de una definición matemática implícita, estas superficies son conocidas por producir bordes

suaves sobre las intersecciones de sus elementos. Esta técnica se ha explotado para diferentes fines y dada esta versatilidad, constantemente se están desarrollando implementaciones que las involucran, se enlistan algunos ejemplos a continuación:

- En 1998 Xiaogang Jin[7] presenta una técnica para deformar mallas utilizando meta-esferas.
- En 2002 Jianhua Shen[17] presenta una plataforma para manipulación y modelado de volúmenes utilizando meta-esferas y cuyo resultado es una malla.
- En 2007 Jun-Wei Chang et.al.[11] presenta un método para simular fluidos utilizando meta-esferas en dos dimensiones, la simulación era capaz de interactuar con diferentes objetos.
- En 2008 Yoshihiro Kanamori et.al.[16] implementa un método de *ray tracing* en GPU para representar altas cantidades de meta-esferas.
- En 2018 Junxuan Bai et.al[15] implementa un método en el que utiliza meta-esferas para evitar deformaciones en animación de cuerpos, estas interactúan sobre una malla, específicamente sobre las articulaciones.
- En 2021 Pei Zhang et.al.[12] presenta un método de simulación de partículas con formas realistas, utiliza meta-esferas para definir la forma de los elementos.

En general este tipo de trabajos no está interesado en la implementación de color. Cuando se requería añadirlo, solían aplicarse una única base o usaban técnicas ajenas a las meta-esferas para dotar con color a la superficie resultante. Es decir, no ha existido hasta ahora una implementación nativa en términos de color y textura de esta técnica; a pesar de que Nishimura[2] ya hacía mención de como calcular una correcta mezcla de color a conjuntos de meta-esferas, el resto de las fuentes disponibles no están interesadas en la implementación de tales características.

Sin embargo, podemos encontrar en diversas implementaciones de esta técnica algunos ejemplos en los que se han visto involucrado soluciones de color y que pueden servirnos para profundizar en las intenciones que tenemos al conducir una investigación como la presente.

James Blinn[1] en 1982 presenta la técnica de meta-esferas, un primer modelo para construir las iso-superficies y un algoritmo para su correcta representación, sus resultados son presentados con color plano en cada superficie.

Al año siguiente Hitoshi Nishimura[2] propone un modelo de meta-esferas propio en el que incluye la relación de radio-distancia y además menciona como realizar la mezcla de color en meta-esferas, sin embargo no consiguió aplicar textura a una superficie.

En 1986 Geoff Wyvill[3] introduce el concepto de *soft object*, presenta un nuevo modelo de meta-esferas y un método para representar las superficies en 3D, además, discute la animación de los objetos resultantes. No obstante, su resultado solo cuenta con color plano en la superficie.

En 1996 Daniel Thalmann[14] introduce el concepto de *primitivas deformables* donde construye cuerpos humanos dividiéndolos y agrupándolos en quince diferentes grupos compuestos por meta-esferas, cada grupo hace referencia a una parte del cuerpo. Utilizando la superficie resultante construye una malla triangular sobre la cual aplican textura a cada conjunto individualmente.

En 2010 Dongfang Chen[8] propone un método para visualizar gotas de agua en 2D utilizando la función de densidad como textura de meta-esfera y a esta le aplica otra textura de relieve (*Bump mapping*) para modificar las normales y el color final depende de la textura que se encuentre detrás. Sin embargo no modifica los vectores normales en relación a las meta-esferas que componen la superficie, por lo que resulta en una incorrecta visualización en la unión de las meta-esferas.

En 2016 Junjun Pan[10] describe un sistema para disección de órganos donde el modelo geométrico está compuesto por una superficie de meta-esferas interior, a partir de esta, generan una malla en el exterior construida utilizando un algoritmo de *skinning*. Es en esta última en donde aplican texturizado para su presentación final.

En 2021, Marwan Abdellah et.al.[13] implementa un algoritmo para reconstruir células cerebrales utilizando meta-esferas, obtenida la superficie, obtienen una malla para su presentación, esta se muestra con diferentes colores para visualizar los diferentes elementos de la célula.

1.2. Objetivo

Integrar color y textura como propiedades nativas en la representación final del sistema de graficación libre de mallas llamado meta-esfera a través de la ampliación de la definición de meta-esfera que asocie al color y cuyo resultado permita la influencia de un modelo de iluminación, con la finalidad de mejorar la representación gráfica de las nubes de puntos.

1.3. Estructura de la tesis

En el capítulo 2 hacemos un recorrido por aquellos trabajos que han aportado a la técnica de meta-esferas, analizando los cuatro modelos más importantes, así como algunos de sus derivados, modelos con baja complejidad computacional y trigonométricos; además se presenta una definición formal y una ampliación de la misma que será la utilizada en el desarrollo de este trabajo y con la que será posible introducir el color y textura de forma nativa, así como otros efectos en las superficies. También se presenta cómo se puede utilizar la técnica de renderizado "ray tracing" con superficies de meta-esferas para lograr su representación con alta definición. Finalmente se analiza la complejidad computacional en el tiempo de las meta-esferas para comprender por qué es necesario el uso de la computación en paralelo, así como algún tipo de algoritmos para reducirla.

En el capítulo 3 se muestra el método para calcular color en meta-esferas basado en la propia definición, así como diferentes algoritmos de aplicación como lo son pseudo-color, mapas de color y textura, así como el correcto cálculo de los vectores normales de la superficie y la aplicación de mapa de normales. En el proceso, se desarrolla y presenta un formato para almacenar superficies de meta-esferas.

En el capítulo 4 se presentan los resultados de aplicar color a superficies de meta-esferas tanto con un modelo de iluminación activado como desactivado, así como con una etapa de post-procesamiento aplicada. También se muestra el rendimiento en el espacio y el tiempo para superficies con diferentes cantidades de meta-esferas.

En el capítulo 5 exponen las conclusiones relacionadas a esta investigación, incluyendo un apartado de trabajo a futuro en el que se enuncian el desarrollo que permitiría continuar la investigación presentada en esta tesis.

Capítulo 2

Marco teórico

2.1. Introducción a meta-esferas

Los llamados *Blobby objects* (que después se conocerían como Metaball, en español Meta-esferas) fueron introducidos por primera vez por *James F. Blinn* en el año de 1982 [1] como una forma de dibujar superficies curvas utilizando directamente funciones matemáticas en lugar de dividirlos en un gran número de polígonos, esta publicación tuvo como objetivo modelar estructuras moleculares y mapas de densidad de electrones. Blinn propone un método para representar superficies complejas partiendo de curvas que pertenecen a las *superficies implícitas* para proponer un modelo[1] que muestrea un átomo de hidrógeno como una distribución Gaussiana $D(x, y, z) = \exp^{-a\,d}$.

Y el campo de intensidad sobre un punto $p(x, y, z)$ para una colección de átomos se representa como la suma de la contribución de cada átomo por separado sobre cada punto $p(x, y, z)$ en el espacio.

$$D(x, y, z) = \sum_i b_i \exp^{-a_i\, d_i^2} \tag{2.1}$$

Donde:

- i Índice del átomo.
- c Centro del átomo.
- a Desviación estándar.
- b Altura.
- d Distancia desde $p(x, y, z)$ hasta $c(x_0, y_0, z_0)$.

Finalmente, se define la superficie resultante como todos aquellos puntos $\{p(x, y, z) \in Superficie \mid F_{Blinn}(x, y, z) = D_i(x, y, z) - T = 0\}$. Donde T es un valor de umbral.

En la Figura 2.1 se ilustra del lado izquierdo el átomo (meta-esfera) desde su función de densidad *D(x, y, z)*; del lado derecho se visualiza la superficie resultante de dos meta-esferas donde se observa la unión suave en el centro de la superficie, resultado de la contribución entre ambas.

[1] Un modelo en el contexto de meta-esfera se refiere a una función implícita $f(x, y, z) = 0$, también llamada función de densidad.

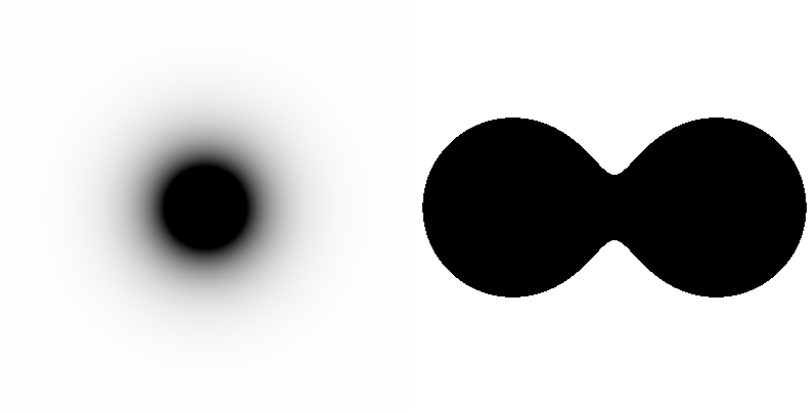


Figura 2.1: Representación 2D de Meta-esfera propuesta por Blinn. *Desviación Estándar* ($a = 16$), *Altura* ($b = 5$) y *Umbral* ($T = 0.15$).

En el año de 1983 Hitoshi Nishimura publicó un modelo propio para la construcción de meta-esferas [2][5], cuya principal aportación fue la relación entre un *Radio de Influencia* (r) y la distancia entre un punto $p(x, y, z)$ y el centro $c(x_0, y_0, z_0)$ de cada meta-esfera en su función de densidad (2.2). Donde el radio de influencia se volvería un parámetro de suma importancia para las ecuaciones polinómicas y la relación entre r y d se usarían posteriormente en la mayoría de modelos de meta-esferas. En la figura 2.2a se muestra representada la meta-esfera de Nishimura, así como la superficie resultante compuesta por dos meta-esferas.

$$F_{Nishimura}(x, y, z) = \sum_i \begin{cases} 1 - 3\left(\frac{d_i}{r_i}\right)^2 - T & 0 \leq d_i \leq \frac{r_i}{3} \\ \frac{3}{2}\left(1 - \left(\frac{d_i}{r_i}\right)\right)^2 - T & \frac{r_i}{3} < d_i \leq r_i \\ 0 & d_i > r_i \end{cases} \tag{2.2}$$

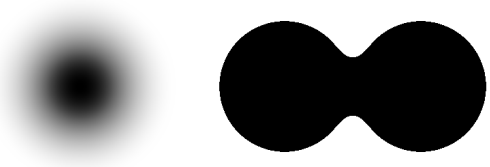
Donde:

- i Índice de la Meta-esfera.
- c Centro de la meta-esfera.
- r Radio de influencia.
- d Distancia desde $p(x, y, z)$ hasta $c(x_0, y_0, z_0)$.
- T Valor de umbral.

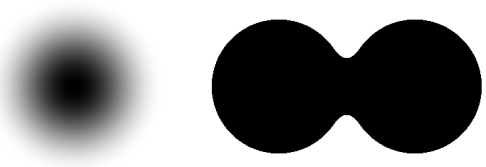
Con la misma composición de elementos, Wyvill en su artículo publicado en el año 1986 [3] formula un modelo propio (2.3) para la creación y calculo de meta-esferas y en el año de 1987, Murakami [5] publica su propia función de densidad (2.4). Podemos observar las meta-esferas construidas con la función de densidad de Wyvill, figura 2.2b y Murakami, figura 2.2c.

$$F_{Wyvill}(x, y, z) = \sum_i -\frac{4}{9}\left(\frac{d_i}{r_i}\right)^6 + \frac{17}{9}\left(\frac{d_i}{r_i}\right)^4 - \frac{22}{9}\left(\frac{d_i}{r_i}\right)^2 + 1 - T \tag{2.3}$$

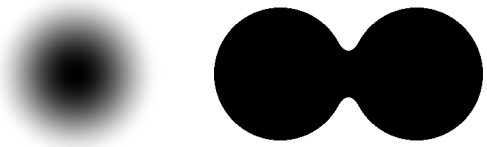
$$F_{Murakami}(x, y, z) = \sum_i \left(1 - \left(\frac{d_i}{r_i}\right)^2\right)^2 - T \tag{2.4}$$



(a) Representación 2D de Meta-esfera propuesta por Nishimura. *Radio de Influencia* ($r = 0.35$) y un *Umbral* ($T = 0.15$).



(b) Representación 2D de Meta-esfera propuesta por Wyvill. *Radio de Influencia* ($r = 0.33$) y un *Umbral* ($T = 0.15$).



(c) Representación 2D de Meta-esfera propuesta por Murakami. *Radio de Influencia* ($r = 0.29$) y un *Umbral* ($T = 0.15$).

Figura 2.2: Propuestas de Meta-esferas. (a) Nishimura, (b) Wyvill y (c) Murakami.

En la figura 2.3 se observan los cuatro modelos descritos con un radio de influencia $r = 1$. En está gráfica se puede visualizar las diferencias entre el modelo de Blinn y aquellos que dependen de un radio de influencia, estos últimos siempre son iguales a cero cuando la distancia es mayor al radio de influencia; mientras que el primero nunca será cero ya que usa una exponencial en su función de densidad.

Las diferencias en los resultados de los diferentes modelos dependen directamente de la distancia calculada, para (2.1) la altura y la desviación estantar son variables clave para calcular la magnitud de la función de densidad (aunque estos, así como el radio de influencia se obtienen a prueba y error), entre más pequeña sea la desviación estandar, menor distancia será necesaria para que un punto pertenezca a la superficie. Por otro lado, para (2.2), (2.3) y (2.4) la variable importante es el radio de influencia, el cual requiere ser mayor para el modelo de Nishimura que los otros dos para que un punto pertenezca a la superficie y el modelo de Wyvill tiene que ser mayor al de Murakami y menor al de Nishimura.

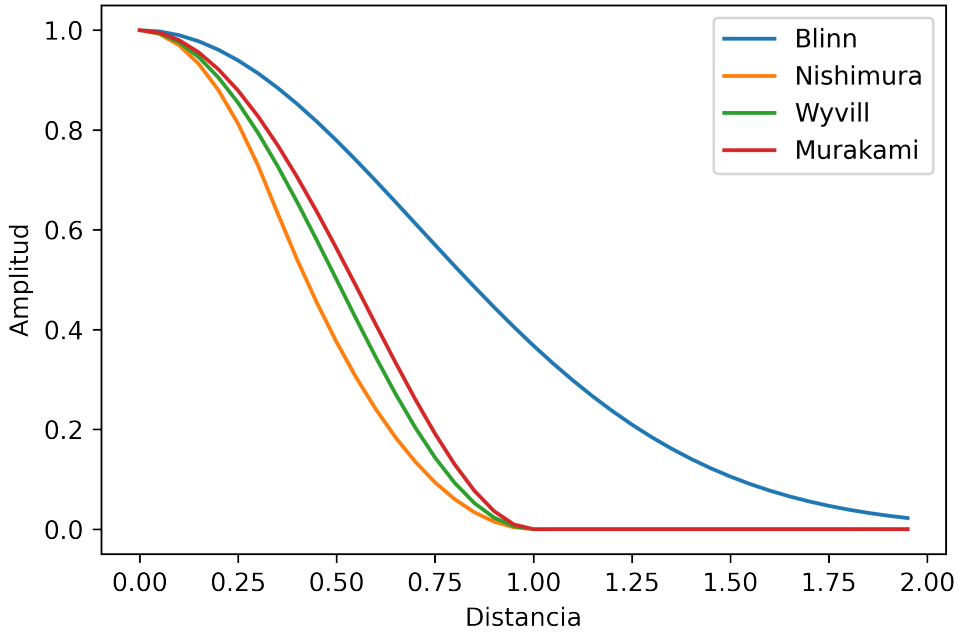


Figura 2.3: Modelos de meta-esferas (Blinn, Nishimura, Wyvill y Murakami).

Posteriormente, en el trabajo publicado por Jay Jeon Kim en el año 1997 [6] introduce una modificación de la ecuación (2.4) obteniendo una meta-esfera que produce superficies más suaves a las propuesta por Murakami. En el año de 1998, Xiaogang Jin utiliza la ecuación (2.3) para realizar deformaciones a geometrías simples y complejas. Dongfang Chen et. al. en el año 2010 [8] se hace referencia a las funciones de densidad (2.1)(2.2)(2.3)(2.4) previamente descritas como las más comunes, utilizando en su trabajo la propuesta por Blinn, pero se deja abierta la posibilidad de usar otra. Además, Olivier Gourmel et. al. en el año 2010 [9] menciona la función de densidad de Blinn (2.1) y una generalización de la ecuación propuesta por Murakami para así lograr un resultado más suave al variar el último exponente.

Por otro lado, existen otras funciones de densidad dependientes de la distancia y el radio de influencia matemáticamente más simples; no obstante son menos comunes debido a que por su naturaleza no componen superficies suaves. En la ecuación resultado del campo escalar (2.5) solo se evalúa la relación entre ambos componentes. Mientras que la función de densidad descrita en [4] es una ligera modificación de (2.5) que hace uso del recíproco, es decir $1/F$. Para ambos casos, el modelo proviene de la ecuación de la circunferencia en 2D o de la esfera en 3D (ver apéndice A.1). En la figura 2.4a se ejemplifica la meta-esfera de la ecuación (2.5).

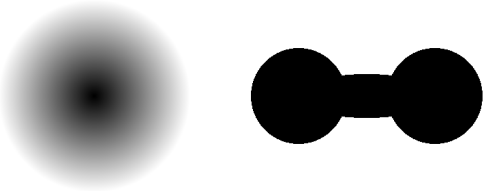
$$F = \sum_i \frac{d_i}{r_i} - T \tag{2.5}$$

La siguiente ecuación (2.6) describe la función de densidad más simple en términos de complejidad computacional, este modelo también viene de la ecuación de la esfera (ver apéndice A.2) y cuya única particularidad es que el valor de umbral no está definido en su función y utilizarlo puede provocar resultados inesperados (y se sugiere utilizar valores de $T \to 0$). Este modelo da un resultado en el que se aprecia como la meta-esfera decrece ligeramente entre mayor sea la distancia. En la figura 2.4b se ejemplifica la meta-esfera de esta ecuación.

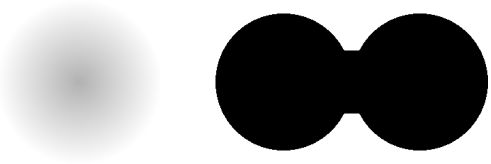
$$F = \sum_i -d_i + r_i \tag{2.6}$$

La ultima ecuación (2.7) ofrece un ejemplo de las posibilidades de un potencial uso de las ecuaciones trigonométricas para la construcción de modelos de meta-esferas que se ha puesto de patente durante el desarrollo de esta investigación y cuya exploración y análisis de sus posibilidades no se han estudiado a la fecha. En la figura 2.4c se observa el resultado de la ecuación propuesta para este ejemplo, donde cte_1 y cte_2 modifican la amplitud y el periodo de la función respectivamente.

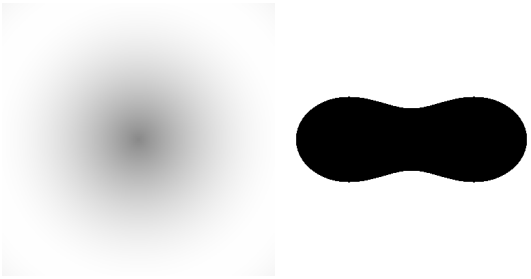
$$F = \sum_i cte_1 \cos(d * cte_2) - T \tag{2.7}$$



(a) Representación 2D de Meta-esfera Simple (2.5). *Radio de Influencia* ($r = 0.35$) y un *Umbral* ($T = 0.5$).



(b) Representación 2D de Meta-esfera Simple (2.6). *Radio de Influencia* ($r = 0.3$) y un *Umbral* ($T = 0.05$).



(c) Representación 2D de Meta-esfera construida con trigonometría. *Umbral* ($T = 0.25$).

Figura 2.4: Propuesta de meta-esferas. (a)(b) meta-esferas computacionalmente simples. (c) meta-esfera función trigonométrica.

En la figura 2.5 Se observa como los modelos (2.5) y (2.6) se comportan de la misma manera si el valor del radio es igual (en este caso $r = 1$), mientras que nuestro modelo trigonométrico puede resultar en valores negativos debido a la naturaleza de la función coseno.

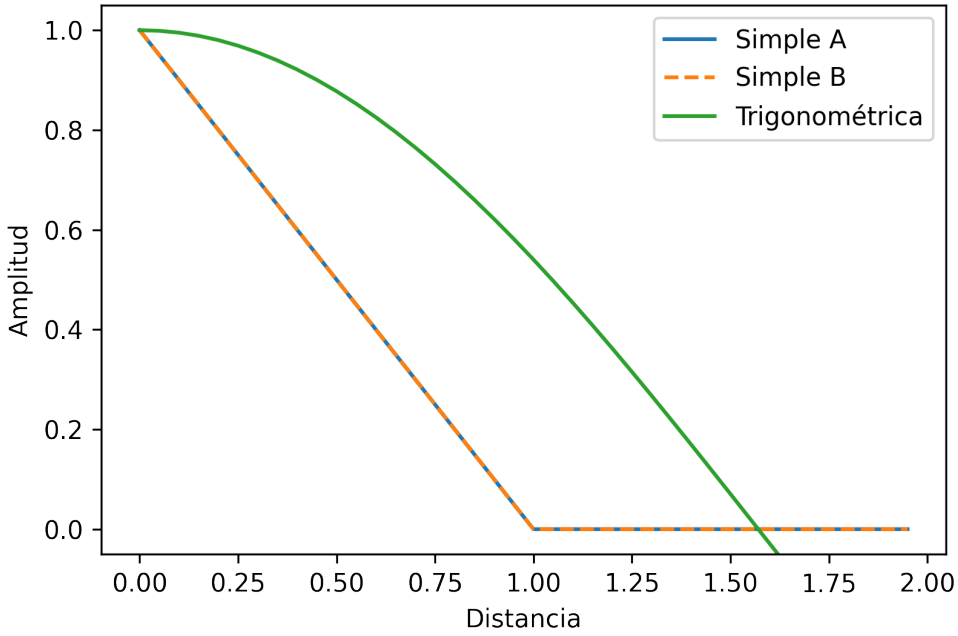


Figura 2.5: Modelos de meta-esferas (Simples y trigonométrica).

2.1.1. Definición de meta-esfera

Una *Meta-esfera* es una *superficie implícita* [18][23] que está definida por los siguientes elementos:

- Un punto central $c = (x_0, y_0, z_0)$
- Una función de densidad f
- Un valor de umbral T

Donde para todo punto $p_i(x, y, z) \in$ a la superficie implícita S_{SI} si y solo si $f(p_i) = 0 \mid i \in \mathbb{N}$.
El resultado de la contribución de k Meta-esferas sobre p_i es una *iso-superficie*[24] (o superficie de meta-esferas).
Donde $\{p_i \in$ a la iso-superficie S_{IS} si y solo si $F(p_i) = T,\ F(p_i) \in (0, T]\}$.

$$F(p_i) = \sum_k f_k(p_i) = T \tag{2.8}$$

La ecuación (2.8) es válida solamente si el valor de T es igual para toda meta-esfera. Para el calculo de la iso-superficie para valores de T diferentes o iguales para cada meta-esfera, T debe incluirse dentro del resultado de f para cada meta-esfera.
Por lo tanto $\{p_i \in$ a la iso-superficie S_{IS} si y solo si $F(p_i) = 0,\ F(p_i) \in [-T, 0]\}$, ecuación (2.9).

$$F(p_i) = \begin{cases} 0 & \sum_k (f_k(p_i) - T_k) \geq 0 \\ -T & \sum_k (f_k(p_i) - T_k) < 0 \end{cases} \tag{2.9}$$

Sin embargo, este trabajo tiene como objetivo la implementación de color sobre superficies de meta-esferas y por lo tanto, al utilizar los elementos disponibles de la definición de meta-esfera para la aplicación de color (como se verá en la sección 3.2), se revelan por si mismas las limitaciones. Por lo cual se hace necesario

incluir nuevos elementos a la definición de meta-esfera con el objetivo de incluir diferentes métodos para la inclusión de color a meta-esferas, como lo son mapas de color, textura, mapas de normales, etc.

Lo que proponemos específicamente aquí es agregar un vector V de n canales con lo cual la meta-esfera podrá almacenar información que a su vez le permita a la iso-superficie resultante ser presentada con color.

Por lo tanto en nuestra nueva definición, una meta-esfera se encuentra definida por los siguientes elementos:

- Un punto central $c = (x_0, y_0, z_0)$
- Una función de densidad $f(x, y, z)$
- Un valor de umbral T
- Un vector información adicional $V(a_0, a_1, ..., a_n) \mid n > 0$

2.2. Ray Tracing

Como ya se mencionó en la sección 2.1, las meta-esferas producen superficies de bordes suaves y el uso de la técnica de renderizado *Ray Tracing*[43][44][45] resulta ser una excelente opción para representar superficies de meta-esfera de manera más precisa, esto debido a que por su definición, la intersección de rayos permite evaluar cuales objetos tendrán interacción con la escena por cada píxel en la imagen final y esta técnica es tan versátil que permite que tal cálculo se pueda adecuar a diferentes tipos de representación de objetos como lo son mallas o superficies implícitas.

El método de *Ray Tracing*, como se muestra en la figura 2.6, funciona calculando vectores (rayos) desde una cámara hacia la escena, se crean tantos rayos como píxeles se quiera la imagen resultante y la dirección de dichos rayos se calcula utilizando la posición de la cámara y cada punto en la imagen. Cuando un rayo interseca con un elemento en la escena, se calcula si el punto de intersección es influenciado por una fuente de luz para establecer a que intensidad se va a aplicar color.

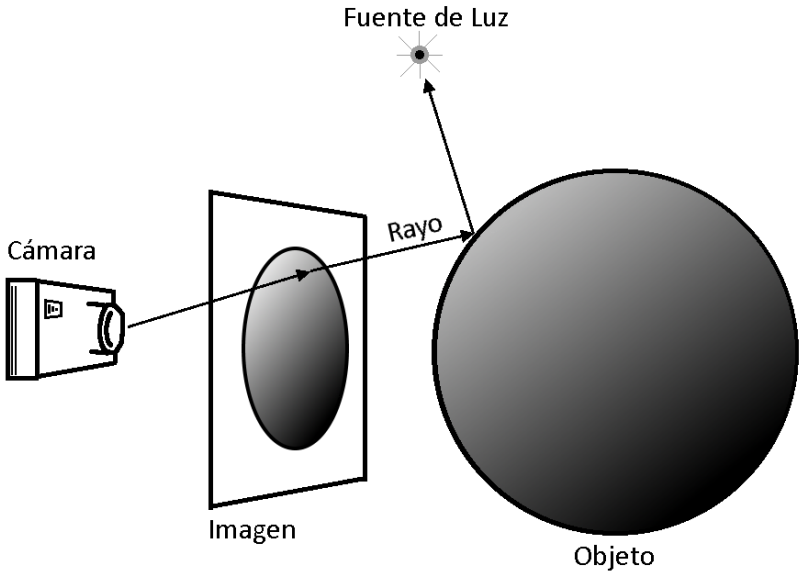


Figura 2.6: Representación gráfica del algoritmo de *Ray Tracing*.

Un rayo está definido por un punto en el espacio (p) y una dirección (d).

$$f(t) = p + td \tag{2.10}$$

Para representar superficies implícitas por medio de *ray tracing* se procede a calcular si existe una intersección entre un rayo y una superficie, para ello, se resuelve la ecuación que defina a la superficie en términos de (2.10). En el caso de una esfera con centro (C), se resuelve el polinomio de segundo orden:

$$\begin{aligned} (X - C)^2 &= r^2 \\ ((p + td) - C)^2 &= r^2 \\ (d^2)t^2 + (2d \cdot (p - C))t + ((p - C)^2 - r^2) &= 0 \end{aligned} \tag{2.11}$$

En la figura 2.7 se visualiza una esfera renderizada (utilizando software ray tracing[2]) desde su definición implícita utilizando (2.11).

[2] *Software Ray Tracing* se refiere a la implementación de la técnica ray tracing sobre algoritmos que se van a ejecutar sobre

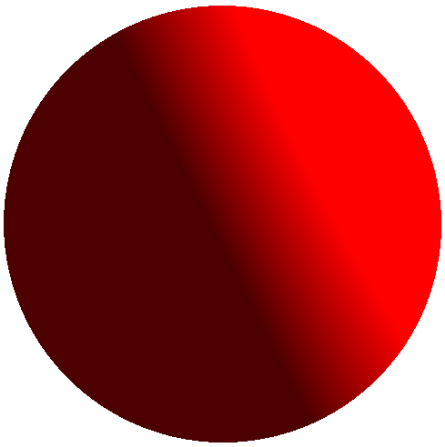


Figura 2.7: Esfera renderizada utilizando *Ray Tracing*.

Intersección rayo meta-esfera

Si estuviéramos evaluando una meta-esfera, basta con utilizar (2.11) para obtener las intersecciones. Aunque hasta la fecha no hay una forma directa de resolver una intersección para un conjunto que forma una iso-superficie; trabajos como los presentados por Nishimura[2], Nishita[5], Jay Jeong Kim[6], Gourmel[9], Thalmann[14], Kanamori[16] muestran diferentes métodos para calcular intersecciones con superficies de meta-esferas.

Sin embargo, César Adrián Victoria Ramírez[19] propone un método que considera una esfera de radio igual al radio de influencia (utilizando (2.2)), dicha esfera, denominada "esfera exterior" será utilizada para descartar aquellas meta-esferas cuya distancia sea mayor al radio. Si se han descartado todas las meta-esferas para un rayo, significa que éste no tiene interacción con la iso-superficie. Si un rayo interseca con una esfera exterior, sólo es necesario calcular el punto p para el cual $f(p) = 0$. Por otro lado si un rayo interseca con dos o más meta-esferas, se avanza el rayo utilizando el método *marching points* propuesto en [46].

2.3. Complejidad computacional en el tiempo para meta-esferas

Las superficies de meta-esferas por definición presentan una elevada complejidad computacional (ver sección 2.1.1), ya que para cada punto a evaluar en el espacio, es necesario calcular la función de densidad de cada meta-esfera sobre dicho punto y esto da como resultado un crecimiento exponencial en el número de meta-esferas $O(p^m)$, donde p es el número de puntos a evaluar y m es el número de meta-esferas en el espacio. Al usar ray tracing, además se incluye el número de rayos creados (r_i) y el número de fuentes de luz (l_j) como elementos que influyen a la complejidad $O(p^{r_i l_j m})$.

Por lo expuesto arriba, se vuelve evidente que tanto el uso de meta-esferas como ray tracing requieren grandes volúmenes de cálculos, dicha complejidad no es deseable en la representación de superficies de meta-esferas ya que se volvería inviable, computacionalmente hablando, visualizar una iso-superficie con alta densidad de meta-esferas en un sistema en tiempo real. Por lo que buscar un medio para reducirla es imperativo.

---

el *Central Processing Unit* (CPU) o en el *Graphic Processing Unit* (GPU). El método más habitual es utilizar una geometría de tres o más vértices que ya haya sido procesado al menos por un shader de vértices y rasterización; y el algoritmo de ray tracing se implementaría sobre el shader de fragmentos.

2.3.1. Paralelismo

Hasta ahora se ha analizado a las meta-esferas como si se ejecutaran secuencialmente en un único proceso en el CPU, donde para cada punto se requiere hacer múltiples operaciones y solamente espera como respuesta un valor de intensidad por punto. Sin embargo, la forma en la que se encuentra definida la meta-esfera es muy conveniente por que es posible dividir cada punto en una tarea independiente, es decir, que es fácilmente paralelizable.

Si bien trabajar paralelismo en el CPU reduce la complejidad dado el número de núcleos con los que se puede trabajar[3], dando como resultado una complejidad $O((p/n_{cpu})^{r_i l_j m})$, donde n_{cpu} es el número de núcleos. Resulta muy limitado si lo comparamos con una GPU[28][29][30] que por su lado contienen en su arquitectura núcleos en orden de miles a los que se le pueden programar en paralelo mediante tecnología que permita ejecutar código ajeno a su propósito principal (computación gráfica) en la GPU como lo es *CUDA*[34][35][36][4] (para más información, ver apéndice B). Permitiendo así que la GPU ejecute segmentos de código únicos para diferentes datos de entrada (conocido como *Single Instruction-Multiple Data* o *SIMD*) y dejando que la CPU ejecute código secuencial y de control para datos de entrada únicos. A esta forma de construcción y organización de código se le denomina *Computación Heterogénea*, ilustrado en la figura 2.8.

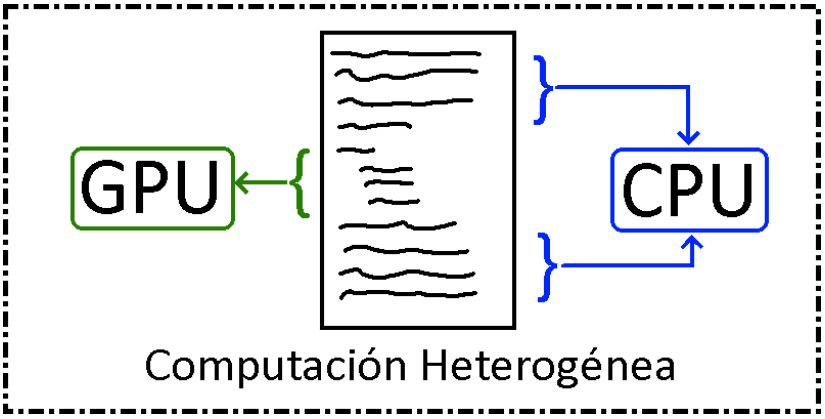


Figura 2.8: Representación de la Computación Heterogénea.

La complejidad en este punto es $O((p/n_{gpu})^{r_i l_j m})$, donde n_{gpu} es el número de procesos que se pueden ejecutar en paralelo y $n_{gpu} > n_{cpu}$. Este valor de complejidad depende directamente de las especificaciones de la tarjeta de video, en este trabajo solo se tomará en cuenta el número de procesos, sin embargo, considerar otros aspectos como la frecuencia o la memoria pueden ser de ayuda para comprender el rendimiento del sistema en comparativa con distintos hardware.

2.3.2. Octree

El valor de complejidad obtenido hasta el momento es bastante eficiente cuando la densidad de meta-esferas no es muy alta, incluso puede ser lineal si es menor o igual al número de procesos que puede ejecutar el GPU. Sin embargo, es importante considerar algoritmos que reduzcan el número de evaluaciones para

[3] Las CPUs Intel® [26] en sus procesadores Intel® Core™ (que es la línea comercial más popular de la compañía) con lanzamiento en el primer cuarto del año 2022 tienen entre cuatro y diez núcleos en sus procesadores Intel® Core™ i3, entre seis y doce núcleos en sus procesadores Intel® Core™ i5, entre diez y dieciséis en sus procesadores Intel® Core™ i7 y entre catorce y dieciséis en sus procesadores Intel® Core™ i9. Por otro lado AMD[27] ofrece entre seis y dieciséis núcleos en sus procesadores de la línea AMD Ryzen™ serie 7000.

[4] *Compute Unified Device Architecture* (CUDA) desarrollada por la empresa NVIDIA Corporation y publicada en el año 2007, se utiliza para ejecutar código de propósito general en la *Unidad de Procesamiento Gráfico* (GPU), es decir, que dicho código no tiene como propósito obtener un resultado gráfico y no es necesario conocer el *pipeline* gráfico para hacer uso de la GPU.

grandes cantidades de meta-esferas[6][9]. La plataforma utilizada en este trabajo utiliza una estructura de datos *octree*.

Un octree (árbol octal) es una estructura de datos tipo árbol donde cada nodo tiene hasta ocho nodos hijos. Cuando todos los nodos tienen exactamente ocho nodos hijos, se denomina como octree completo. Los componentes de una estructura octree son un nodo padre, nodos intermedios y nodos hoja. El nodo padre es único y es la base de la estructura y se encuentra a profundidad cero, los nodos intermedios son aquellos que a su vez tienen nodos hijos y su profundidad es mayor a cero y los nodos hoja son nodos que no contienen nodos hijos. La profundidad de un nodo se refiere al nivel, dentro del árbol, en donde se encuentra el nodo.

Las estructuras de árbol permiten una organización jerárquica de los elementos de interés, en el caso de las meta-esferas, lo que se busca es relacionar grupos de estas con los diferentes nodos de la estructura para poder recuperarlas posteriormente. El criterio de selección de meta-esferas es la división espacial que proporciona la estructura y la lectura de los datos se realiza a través del recorrido del árbol con algoritmos como *breath-first-search* o *depth-first-search*[58][59]. Sin embargo, para estos últimos, su complejidad es logarítmica y los accesos no son aleatorios.

Otro camino es el establecido por Victoria Ramírez[19] y Soriano Valdez[20][22], que proponen la implementación de una estructura de datos *octree* almacenada en un arreglo unidimensional, donde el nodo padre se almacena en la localidad cero del arreglo, seguido de los ocho nodos hijos que ocupan las localidades uno a nueve y continuando así hasta la profundidad máxima del árbol. Dividido el espacio, es simple descartar meta-esferas que no se encuentran en el espacio del nodo y donde la relación entre las meta-esferas y la estructura de datos está dada por una "clave Morton"[21][25] codificada para la posición (x, y, z) de cada meta-esfera y cada uno de los nodos ya mencionados.

La codificación en el árbol completo requiere una cadena de bits que sea divisible entre el número de valores de la estructura de datos, en el caso del octree se utilizan grupos de 3 bits que permiten hacer combinaciones de ocho valores para representar el octante. Y cada grupo de bits va a representar un estrato de profundidad, considerando que el primer nivel se encuentra en el grupo de bits más significativos y el último en los menos significativos.

Esta implementación utiliza un valor entero sin signo, el cual nos permite un máximo de diez niveles de profundidad utilizando los 30 bits menos significativos (figura 2.9) y los dos bits restantes se utilizarán como indicador de pertenencia al espacio ocupado por el octree, donde 00_2 indica que no pertenece y 01_2 si pertenece (siempre será 01_2 en el caso de los nodos).

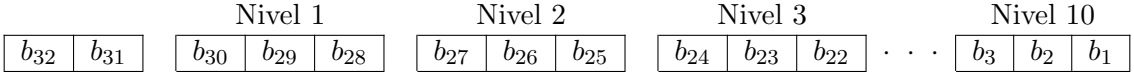


Figura 2.9: Representación de los niveles de profundidad de un octree codificado en una variable entera sin signo.

Para las meta-esferas, se codifica su posición en el espacio. Primero se mapea la posición al rango $([0, 1023], [0, 1023], [0, 1023])$ que es el máximo valor al utilizar 10 bits, es decir, un valor entero sin signo y asignando 30 bits a la posición. El código se forma al ordenar los bits $X_i Y_i Z_i$ del menos significativo al mas significativo en la cadena final como se muestra en la figura 2.10. Los dos bits más significativos funcionan igual que con los nodos.

$$(X_{10}\,X_9\,X_8\,X_7\,X_6\,X_5\,X_4\,X_3\,X_2\,X_1,\ Y_{10}\,Y_9\,Y_8\,Y_7\,Y_6\,Y_5\,Y_4\,Y_3\,Y_2\,Y_1,\ Z_{10}\,Z_9\,Z_8\,Z_7\,Z_6\,Z_5\,Z_4\,Z_3\,Z_2\,Z_1)$$

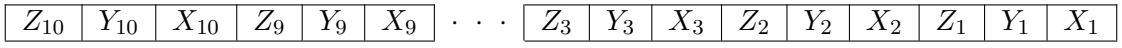


Figura 2.10: Representación de la posición codificada en una variable entera sin signo.

La relación entre la clave Morton del octree y de la meta-esfera es directa y nos permite conocer en que nodo se encuentra esta última a diferentes niveles. Si se comparan los bits 3 al 5 más significativos, se localiza la meta-esfera en un nodo de primer nivel, si se utilizan 3 bits más, se está hallando la meta-esfera en un nodo de segundo nivel y así sucesivamente. En la figura 2.11 se visualiza gráficamente un *quadtree*[5] con coordenadas de sus nodos para los primeros tres niveles. Para comprobar la pertenencia de la meta-esfera al espacio de un nodo comparamos la clave Morton de la meta-esfera con la del octree $MK_{me}\ and\ MK_{octree} = MK_{octree}$, si resulta verdadero, la meta-esfera se encuentra en ese nodo. Por ejemplo, si se quiere localizar una meta-esfera con coordenadas $(67, 768)_{10} = (00\,01\,00\,00\,11, 11\,00\,00\,00\,00)_2$, con una clave Morton $MK_{me} = 10\,10\,00\,01\,00\,00\,00\,00\,01\,01$ y una clave Morton de nodo $MK_{octree} = 10\,10\,00\,00\,00\,00\,00\,00\,00\,00$, tenemos:

$$10\,10\,00\,01\,00\,00\,00\,00\,01\,01\ and\ 10\,10\,00\,00\,00\,00\,00\,00\,00\,00 = 10\,10\,00\,00\,00\,00\,00\,00\,00\,00$$

Por lo tanto, la meta-esfera se encuentra en el nodo superior izquierdo.

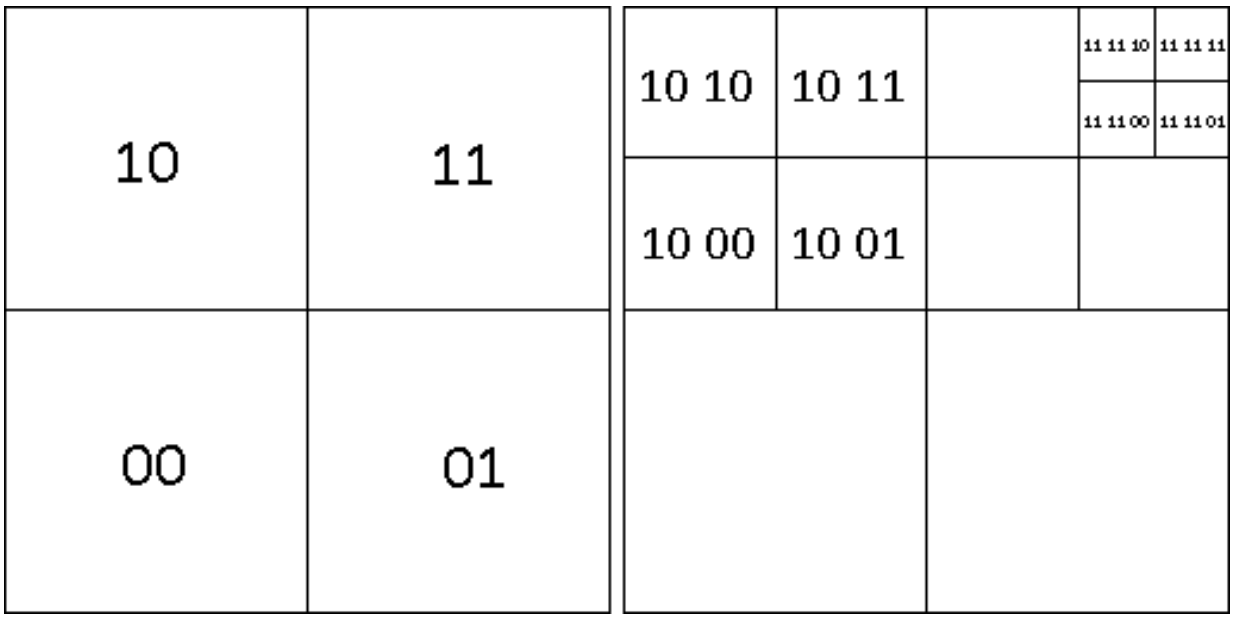


Figura 2.11: Representación gráfica de una estructura de datos *quadtree* con ejemplos de claves Morton para los tres primeros niveles.

La codificación tiene una complejidad lineal tanto en el número de nodos como en el número de meta-esferas. El número de nodos es constante y está definido desde el inicio (por ejemplo, 73 nodos para una profundidad de tres). Y esta adición implica que no se evaluarán todas las meta-esferas, por lo tanto, la complejidad será:

$$O\left(k_{nodo} + m + \left(\frac{p}{n_{gpu}}\right)^{r_i l_j (m - m_k)}\right) \tag{2.12}$$

Donde k_{nodo} es el número de nodos y m_k es el número de meta-esferas descartadas para cada punto.

La complejidad (2.12) es correcta si solamente se va a evaluar a las meta-esferas contenidas en un solo nodo. No obstante, en los nodos adyacentes también pueden existir elementos que deberían influir sobre la superficie. Es por esto que es necesario un algoritmo de búsqueda de vecindades, se utilizará el descrito en

[5]Un *quadtree* es similar a un *octree*, con la diferencia de que este árbol solo puede tener hasta cuatro nodos hijos.

la sección 2.4, el cual modifica la complejidad de la siguiente manera:

$$O\left(3d + k_{nodo} + m + \left(\frac{p}{n_{gpu}}\right)^{r_i l_j (m - m_k)}\right) \tag{2.13}$$

Donde d es el número de niveles del octree.

2.4. Búsqueda de vecindades

Cuando se ha calculado un nodo en el cual se espera encontrar una superficie, ese nodo nos permite conocer las meta-esferas que se encuentran en su espacio, por lo que es posible calcular una iso-superficie con estos elementos. No obstante, suele ser insuficiente la evaluación de un solo nodo para este propósito y utilizar en conjunto las meta-esferas que se encuentran en los nodos vecinos resulta en una superficie homogénea y con mejor definición.

En [20] se desarrolla una algoritmo para la búsqueda de vecindades de forma radial, utilizando la clave Morton para este propósito. Esto permite recuperar la información de las meta-esferas que se encuentran en los veintiséis nodos vecinos del nodo central. Estas vecindades las define en tres tipos: vecindad por cara, vecindad por arista y vecindad por vértice.

El primero es el conjunto de nodos que se encuentran a la derecha (R), izquierda (L), arriba (U), abajo (D), enfrente (F) y atrás (B) del nodo central.

$Vecino_{cara} = \{N_R,\ N_L,\ N_U, N_D,\ N_F,\ N_B\}$

El segundo es el conjunto de nodos que combinan dos direcciones en el espacio (por ejemplo: derecha-arriba o abajo-atrás).

$Vecino_{arista} = \{N_{RU}\ N_{RD}\ N_{RF}\ N_{RB}\ N_{LU}\ N_{LD}\ N_{LF}\ N_{LB}\ N_{UF}\ N_{UB}\ N_{DF}\ N_{DB}\}$

El tercero es el conjunto de nodos que combinan tres direcciones en el espacio (por ejemplo: derecha-abajo-enfrente).

$Vecino_{vertice} = \{N_{RUF}\ N_{RUB}\ N_{RDF}\ N_{RDB}\ N_{LUF}\ N_{LUD}\ N_{LDF}\ N_{LDB}\ \}$

La propiedad espacial de la clave Morton permite calcular las vecindades con operaciones binarias. El algoritmo utilizado busca encontrar a los nodos vecinos partiendo de localizar la posición del nodo central en el octante de su nodo padre, con una operación lógica *and* se evalúa el termino del algoritmo o se repite el proceso en un nivel inferior y con la operación lógica *xor* se construye la clave Morton del nodo vecino.

En el algoritmo 1 se muestra la implementación del cálculo de vecindades por cara. Y hacemos las siguientes consideraciones: Para las direcciones "izquierda" y "derecha" se utilizará una máscara $Mask = 001_b$, para "arriba" y "abajo" $Mask = 010_b$ y para "enfrente" y "atras" $Mask = 100_b$. Si la dirección es positiva, es decir, "derecha", "arriba" y "enfrente", se usará un valor de signo $SignDir = 000_b$ y si la dirección es negativa, $SignDir = Mask$. La profundidad del *octree* ($Depth$) debe ser constante para toda la búsqueda.

Algoritmo 1: Algoritmo que calcula la clave Morton de los vecinos por cara de un nodo.

```
[nMK, Flag] ← getVecindad (MortonKey, Mask, SignDir, Depth)
    Limit ← 10 − Depth
    Test ← (MortonKey << 2) >> 2
    Flag ← False
    nMK = MortonKey
    for i ← Limit to 10 do
        iMask ← Mask << (i ∗ 3)
        iRes ← SignDir << (i ∗ 3)
        nMK ← nMK ⊕ iMask
        if (Test ∧ iMask) == iRes then
            Flag ← True
            break
        end
    end
end
```

Cabe reiterar que este método solo permite encontrar vecinos por cara. Para calcular la clave Morton de un vecino por arista, es imperativo utilizar el algoritmo dos veces, en dos direcciones distintas y utilizando como clave de entrada para la segunda ocasión el resultado de la primera. Y para obtener la clave Morton de un vecino por vértice, hay que usar el algoritmo tres veces, utilizando el resultado de la segunda iteración como parámetro de entrada para la tercera.

Otra fortaleza que nos ofrece este algoritmo es la posibilidad de elegir la profundidad como parámetro, de esta forma es posible buscar nodos vecinos a diferentes niveles.

Capítulo 3

Metodología

La definición de meta-esfera establecida en la sección 2.1.1 dice que toda meta-esfera va a generar influencia sobre un punto y en el caso de que esta sea mayor a un valor establecido, se considerará como parte de la iso-superficie. A grandes rasgos, el resultado de la función de densidad para cada elemento va a contribuir (de acuerdo a factores como el radio y la distancia) equitativamente a verificar si pertenece o no a la superficie. Esta interpretación va a ser la que se va a seguir durante el desarrollo de este capítulo y nos va a servir de guía para desarrollar los métodos de color, textura y vectores normales.

3.1. Color en Meta-esfera

Para añadir color a una meta-esfera como objeto independiente, basta con definir los valores RGB con los que se va a dibujar la superficie implícita. Sin embargo, cuando dos o más meta-esferas están contribuyendo para formar una superficie, se debe calcular la mezcla de color dependiente de la intensidad del campo de cada meta-esfera sobre cada punto $p(x, y, z)$. Esto se consigue con la sumatoria del calculo del valor porcentual de influencia de cada meta-esfera $f_i(x, y, z)$ multiplicado por el color de la meta-esfera[2], ecuación (3.1). Resultando en una correcta mezcla de color entre los elementos como se muestra en la figura 3.1.

Cabe aclarar que para aplicar color se debe retirar la restricción de $\{F(p) \in [-T, 0]\}$ para valores de T diferentes o iguales. De lo contrario el color resultaría en una división sobre cero para todo punto en el espacio que pertenece a la superficie. Por lo tanto $\{F(p) \in [-T, \infty),\ p(x, y, z) \in S_{IS} \ \leftrightarrow\ F(p) > 0\}$.

$$Color_f = \sum_i \frac{f_i(x, y, z)}{F(x, y, z)} \cdot Color_i(R, G, B) \tag{3.1}$$

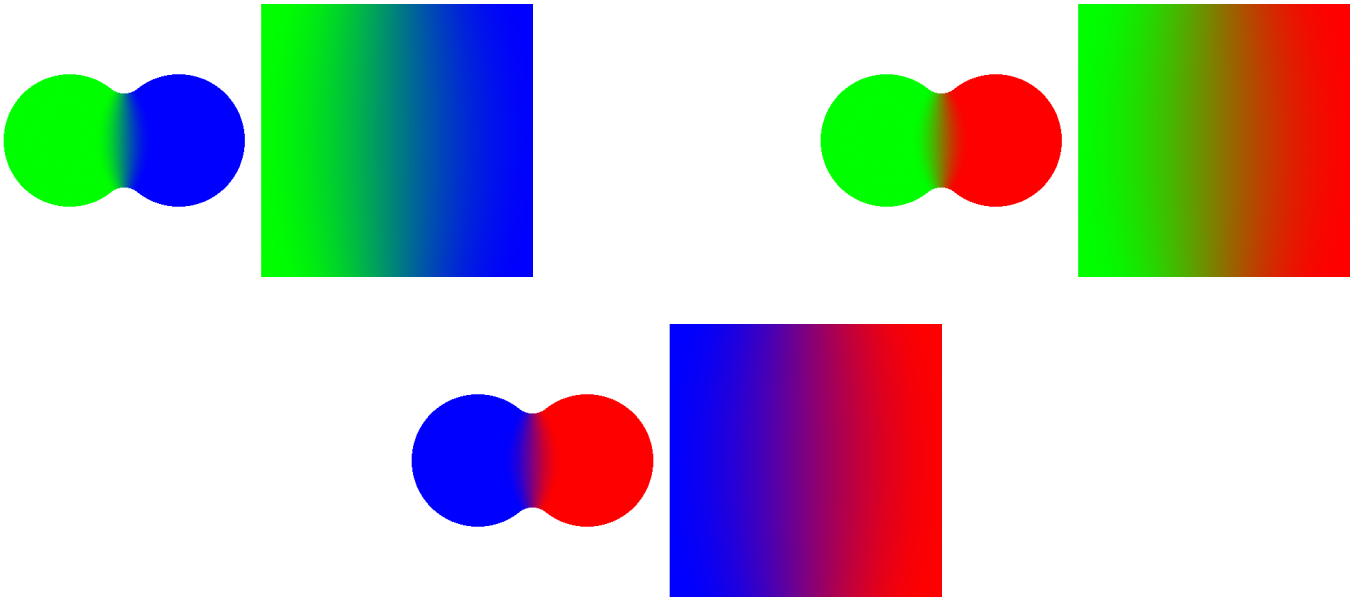


Figura 3.1: Mezcla de color en meta-esferas.

3.2. Pseudo-color en Meta-esfera

El primer paso para dar un propósito a establecer color sobre una superficie construida con meta-esferas es que el color añadido tenga un significado y usar pseudo-color parece la aproximación más natural para comenzar a añadir color a una meta-esfera, de lo contrario, solamente se obtendría una superficie con carácter estético como se muestra en la figura 3.2. Similar al uso de esta técnica en procesamiento digital de imágenes, en meta-esferas, uno de los componentes resultantes es un campo escalar de intensidades, además, las meta-esferas cuentan con otras propiedades con las cuales establecer un color bajo uno o más criterios según el objetivo requerido.

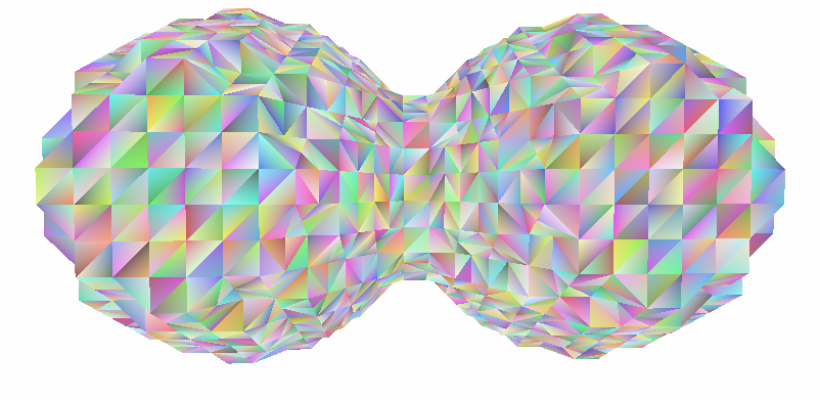


Figura 3.2: Color aleatorio en meta-esferas.

La adición de color por campos de intensidad muestra cuanta influencia recibe un punto $p(x, y, z)$ de todas las meta-esferas, en la figura 3.3 se observa la aplicación de color por campos de intensidad sobre el canal de color *rojo*, en el cual las zonas con menor influencia recibida se colorean más oscuras, donde $\{Color_f \in [T, 1]\}$ como un rango normalizado dependiente del valor de T, tomando en cuenta que la influencia de cada meta-esfera va a aportar color a la superficie, podemos sustituir cada valor de color RGB por la influencia f_i,

ecuación (3.2).

$$\begin{aligned} Color_f.r &= \sum_i \frac{f_i(x,y,z)^2}{F(x,y,z)} \\ Color_f.g &= 0 \\ Color_f.b &= 0 \end{aligned} \tag{3.2}$$

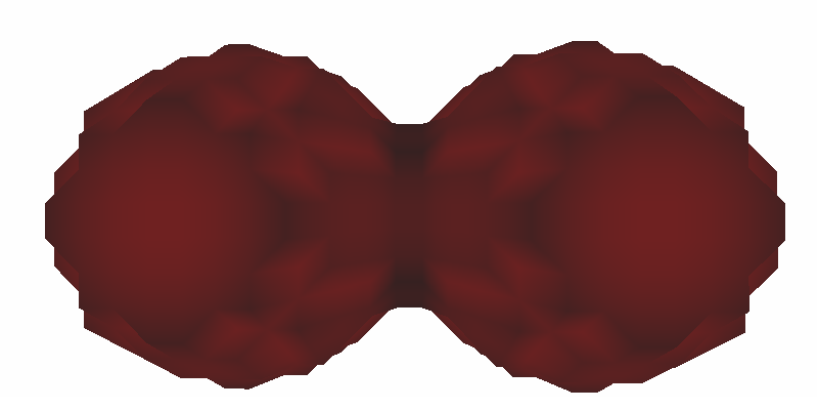


Figura 3.3: Aplicación de color por campo de intensidad.

En el algoritmo 2 se muestra la implementación de la aplicación de color por campos de intensidad, cabe aclarar que la función $Calcula_f$ se refiere al cálculo de la función de densidad, dentro de esta, pueden estar implementados uno o más modelos de meta-esferas.

Algoritmo 2: Algoritmo que establece color a superficies de meta-esferas de acuerdo al campo de intensidad.

```
. . .
foreach Punto do
    foreach Meta-esfera do
        f ← Calcula_f(meta-esfera)
        if f > 0 then
            F ← F + f
            Col_r ← Col_r + f^2
        end
    end
    if F > 0 then
        Col_r ← Col_r/F
    end
end
. . .
```

Otro parámetro de la definición de meta-esfera con el que podemos aplicar color es la posición en el espacio de la propia meta-esfera. Interpretando las coordenadas (x, y, z) como el color a una escala de $[0, 1]$ o $[0, 255]$ o personalizada, según las necesidades. En la figura 3.4 se presenta un ejemplo en el que se toma

la coordenada Y como banda de color G en una escala de $[0, 255]$, ecuación 3.3.

$$\begin{aligned} Color_f.r &= 0 \\ Color_f.g &= \sum_i \frac{f_i(x,y,z) \cdot c(0,y_0,0)}{F(x,y,z) \cdot 255} \\ Color_f.b &= 0 \end{aligned} \tag{3.3}$$

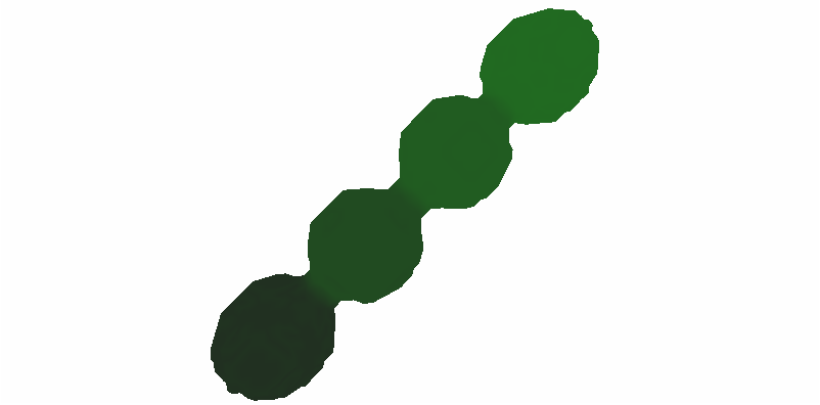


Figura 3.4: Aplicación de color por posición (altura).

Algoritmo 3: Algoritmo que establece color a iso-superficies de acuerdo a la posición en el espacio de las meta-esferas.

```
. . .
foreach Punto do
    foreach Meta-esfera do
        f ← Calcula_f(meta-esfera)
        if f > 0 then
            F ← F + f
            Col_g ← Col_g + (f * c_y)
        end
    end
    if F > 0 then
        Col_g ← Col_g/F
    end
end
. . .
```

Además de los elementos de la definición con los cuales se puede hacer pseudo-color, otros valores con los que es posible aplicar color pueden ser la distancia (usualmente calculada en la función de densidad), los vectores normales, la clave Morton, etcétera.

3.3. Mapas de color en Meta-esferas

El uso de mapas de color es conveniente cuando se busca una representación personalizada del color en la superficie (a diferencia de la aplicación de color por pseudo-color). No obstante, los elementos utilizados para la aplicación del color son los mismos que los ya discutidos en esta sección, es decir, utilizando el campo de intensidad, la posición en el espacio u otros.

Para aplicar un mapa de color a una superficie por campos de intensidad basta con sustituir la influencia de (3.2) cuyo significado era color, por los valores RGB apropiados del mapa de color, ecuación (3.4).

$$Color_f = \sum_i \frac{f_i(x,y,z)}{F(x,y,z)} \cdot ColorMap.Color \tag{3.4}$$

En la figura 3.5 se aplica un mapa de color de cinco elementos a una superficie en la que solo se muestran los primeros dos colores. Esto es debido a la definición de la meta-esfera, es decir, lo correcto sería que toda la superficie se mostrara solo con el primer color del mapa, sin embargo, esto no ocurre debido al algoritmo de selección de color (dividir F sobre el número total de colores disponibles), a la resolución con la que se están calculando las meta-esferas y la técnica de visualización (en este caso *marching cubes*). Ver el segundo color del mapa en la superficie es resultado de que la influencia previa al uso de *marching cubes* es mayor a la influencia en la superficie después de aplicarlo.

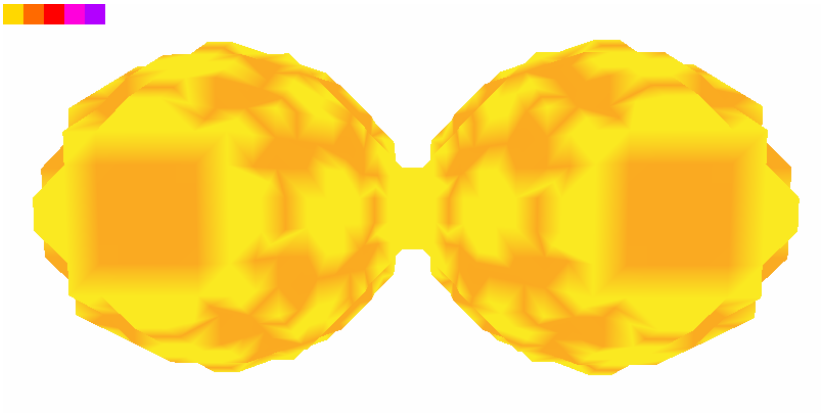


Figura 3.5: Aplicación de mapa de color por intensidad.

También es posible generar artefactos si la influencia calculada para el color en un punto $p(x,y,z)$ es menor o igual a cero, es decir, el calculo del color no pertenece a la superficie que si existe. En la figura 3.6 se visualizan artefactos (color negro) reduciendo el valor de umbral ($T = 0.01$). Si $T \to 0$ mayor es la proporción de artefactos en la superficie. Nuevamente la superficie solo usa dos colores del mapa, esto es por las misma razones ya expuestas.

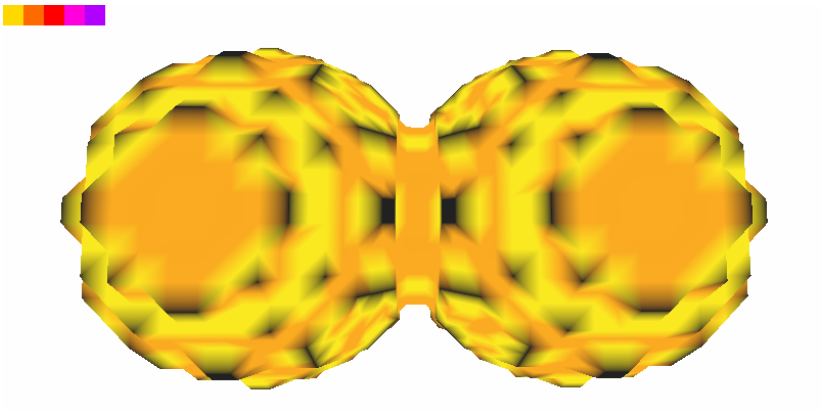


Figura 3.6: Artefactos en la visualización de mapa de color por intensidad.

Por otro lado, para aplicar mapas de color a una superficie por coordenadas en el espacio basta con evaluar el o los ejes de coordenadas de la meta-esfera que influyen sobre el punto a evaluar. En la figura 3.7 se observan cuatro meta-esferas separadas por diez unidades sobre el eje XY empezando desde la posición

$(5, 5)$, a las cuales se les aplicó un mapa de color evaluando únicamente la posición en el eje Y en una escala de $[0, 40]$. Se visualiza que se aplicaron los colores solo omitiendo el tercero, esto es correcto debido a que el resultado de la altura de las meta-esferas nunca es igual al índice 2.

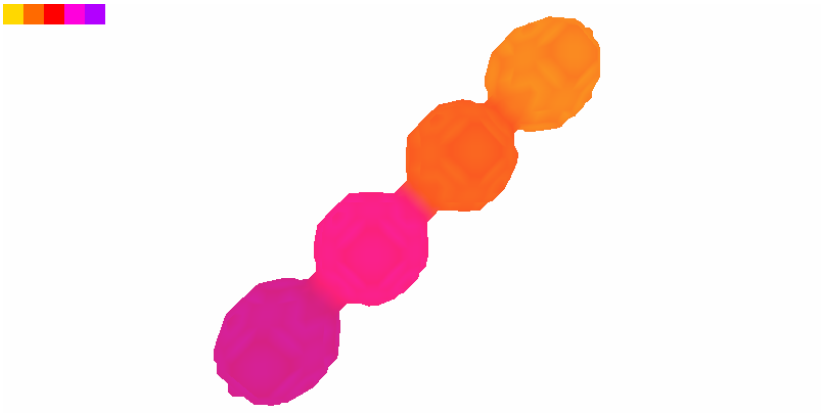


Figura 3.7: Aplicación de mapa de color por coordenada en el espacio.

En el algoritmo 4 se muestra la implementación de la aplicación de color por mapas de color. Donde la función $GetColorMap(var)$ divide el la magnitud máxima de la intensidad (f) o de la posición (p) entre el tamaño del mapa para obtener color.

Algoritmo 4: Algoritmo que establece color a superficies de meta-esferas por intensidad o posición de acuerdo a un mapa de color.

```
. . .
foreach Punto do
    foreach Meta-esfera do
        f ← Calcula_f(meta-esfera)
        if f > 0 then
            F ← F + f
            Cmap ← GetColorMap(f)/* Color por intensidad                    */
            //Cmap ← GetColorMap(p)/* Color por posición                    */
            Col_r ← Col_r + (f * Cmap_r)
            Col_g ← Col_g + (f * Cmap_g)
            Col_b ← Col_b + (f * Cmap_b)
        end
    end
    if F > 0 then
        Col_rgb ← Col_rgb/F
    end
end
. . .
```

3.4. Textura en Meta-esferas

Para aplicar una textura a una superficie compuesta por meta-esferas hay que relacionar el color final de una meta-esfera con los valores RGB de una textura utilizando coordenadas UV.

Esta relación de una superficie con coordenadas UV depende de la forma que tenga la misma superficie, de otra manera, resultaría en una superficie con color (tomado de la textura) aleatorio.

Si tomamos una superficie que este formando un plano sobre el eje XY, XZ o YZ, es decir, que las meta-esferas que lo componen tienen un valor $z = 0$, $y = 0$ o $x = 0$ respectivamente, las coordenadas de textura y/o color final se obtienen mapeando la posición de cada meta-esfera sobre la textura. En la figura 3.8 se presenta una textura de $50p\,x\,50p$ aplicada a una superficie de $7\,x\,7$ meta-esferas, mostrando transiciones suaves de color como se visualiza en la textura asociada. El color se puede almacenar como valores RGB o como coordenadas UV en V, dependiendo de las circunstancias.

Figura 3.8: Aplicación de textura a meta-esferas.

Cabe destacar que entre mayor sea la cantidad de meta-esferas que componen la superficie, mayor será la definición del color resultante, como se muestra en la figura 3.9, donde se presenta una superficie de $9\,x\,9$ en la que se aprecia una mayor cantidad de colores obtenidos de la textura.

Figura 3.9: Aplicación de textura a meta-esferas.

Una superficie de meta-esferas puede representar formas complejas, las cuales pueden ser resultado de una nube de puntos o de una geometría. Para la primera, si el conjunto de puntos no contiene información de color o se busca modificar los datos de la nube, el uso de un algoritmo de *UV Unwrapping*[1] es necesario para obtener coordenadas de textura que sean coherentes con la superficie de meta-esferas. Una vez obtenida la nube de puntos desenrollada, se procede a crear una textura adecuada (Cabe resaltar que crear una textura es un procedimiento artístico e independiente del texturizado de superficies de meta-esferas) y se asocia a cada punto un valor RGB o directamente su correspondiente coordenada UV para su posterior calculo y despliegue ya mencionados anteriormente.

[1] *UV Unwrapping* es una técnica que consiste en desplegar información en 3D a un plano en 2D, siendo esta proyección las coordenadas de textura asociada a los datos de entrada. Este proceso es usado para facilitar la tarea de relacionar una textura en 2D a un objeto en 3D. Los algoritmos de unwrapping son variados y se pueden encontrar desde proyecciones cúbicas, cilíndricas o esféricas, métodos geométricos para desenrollar, métodos qué usan redes neuronales, entre otros. Sin embargo, profundizar en estos métodos se encuentra fuera del alcance de este trabajo.

En la figura 3.10 se presenta una nube de puntos (primera imagen) a la cual se le aplicó una algoritmo de unwrapping de proyección cilíndrica (segunda imagen), que sirvió como guía para crear una textura (tercera imagen) para obtener color utilizando coordenadas UV y finalmente desplegar una superficie de meta-esferas texturizada (cuarta imagen).

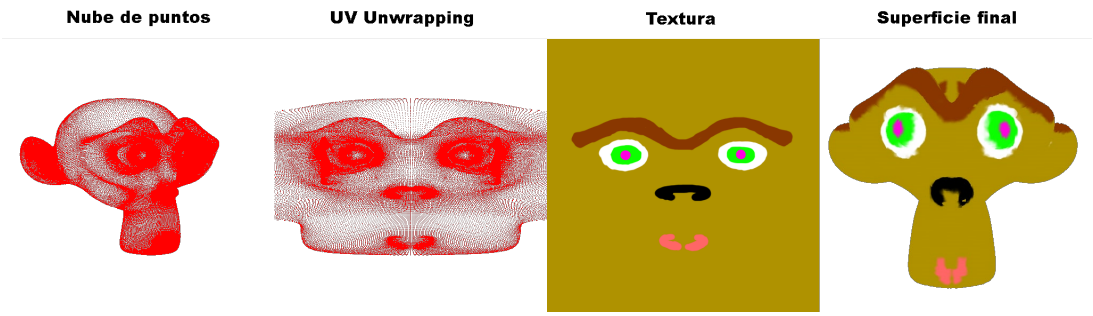


Figura 3.10: Procedimiento UV Unwrapping sobre el modelo de Suzanne.

Similar a la función $GetColorMap(var)$, en el algoritmo 5 la función $GetColorTex(var, var)$ utiliza las coordenadas de textura para para obtener el color final.

Algoritmo 5: Algoritmo que establece textura a superficies de meta-esferas.

```
. . .
foreach Punto do
    foreach Meta-esfera do
        f ← Calcula_f(meta-esfera)
        if f > 0 then
            F ← F + f
            Cmap ← GetColorTex(uv, Tex)
            Col_r ← Col_r + (f * Cmap_r)
            Col_g ← Col_g + (f * Cmap_g)
            Col_b ← Col_b + (f * Cmap_b)
        end
    end
    if F > 0 then
        Col_rgb ← Col_rgb/F
    end
end
. . .
```

Para construir una superficie partiendo de una geometría ya construida lo necesario es transformar la información del modelo tridimensional a un modelo de meta-esferas. Considerando que el objeto de entrada ya fue previamente trabajado en un motor para modelado como *Blender*[37], *3D Max*[38], entre otros, es decir, que el trabajo de calculo de coordenadas de textura está resuelto. Un modelo de meta-esferas requiere que la información que la define más los elementos de color e información adicional sean considerados en un archivo final. Este trabajo de investigación propone el representado en la figura 3.11:

(uint)	Número total de meta-esferas								
(uint)	Bandera de características								
(array)	Nombre de archivos de Textura, Normales, etc.								
MB_1	x_0	y_0	z_0	r	T	Normales	Color	NormalMap	...
MB_2	x_0	y_0	z_0	r	T	Normales	Color	NormalMap	...
...	...			...			...		...
MB_n	x_0	y_0	z_0	r	T	Normales	Color	NormalMap	...

Figura 3.11: Formato de archivo de meta-esfera.

Donde las coordenadas (x_0, y_0, z_0) se refiere al centro de la meta-esfera, r es el radio de influencia, T es el valor de umbral, las *Normales* son los vectores normales asociados a cada meta-esfera, el *Color* puede estar dado por sus valores en *RGB* o por sus coordenadas de textura *UV*, las *NormalMap*, si no se utiliza las coordenadas UV de textura o si el mapeado no es idéntico, es un vector de tres dimensiones que almacena el valor correspondiente a un mapa de normales o sus correspondientes coordenadas de textura y por ultimo, la *Bandera de características* es un valor con el que se pretende conocer qué características componen a cada meta-esfera y se va a evaluar de la siguiente manera:

$(xxxx\,xxxx)\ and\ (0000\,0001) = (0000\,0001) \rightarrow$ Existe centro
$(xxxx\,xxxx)\ and\ (0000\,0010) = (0000\,0010) \rightarrow$ Existe radio de influencia
$(xxxx\,xxxx)\ and\ (0000\,0100) = (0000\,0100) \rightarrow$ Existe valor de umbral
$(xxxx\,xxxx)\ and\ (0000\,1000) = (0000\,1000) \rightarrow$ Existe valor de Normales
$(xxxx\,xxxx)\ and\ (0001\,0000) = (0001\,0000) \rightarrow$ Existe valor de color
$(xxxx\,xxxx)\ and\ (0010\,0000) = (0010\,0000) \rightarrow$ Existe valor de mapa de normales
... ...

Donde si no existe centro para cada meta-esfera, se debe interpretar que la posición se asignará utilizando otro criterio. Si el radio de influencia no existe, significa que se aplicará un valor por defecto a toda meta-esfera. Si el valor de umbral no existe, significa que se utilizará un mismo valor de T para cada meta-esfera. Si no existen valores de normales, se puede considerar que la escena no tiene iluminación o estas se establecerán en otro momento. Si no existen valores de color, normalMap y otros, se omitirá el calculo para cada característica individualmente. Las rutas para los archivos de cada característica deberán estar ordenados de acuerdo a la bandera de características y tomando en cuenta si estas están activas o si no es necesario cargar archivos externos.

Para el arreglo de nombres de archivos, si existe una cadena que haga referencia a un archivo de textura y el color existe en la bandera de características, se interpreta que lo que está almacenado son coordenadas de textura. Si la cadena solo es la palabra *"Color"*, significa que lo que está almacenado son los valores *RGB*.

Si existe el valor de color y de mapa de normales y existen cadenas que sean archivos de textura y mapa de normales respectivamente, se interpretará como que el valor de normal es idéntico a las coordenadas de textura almacenadas en color. Si la cadena del archivo de normales tiene la preposición *"UV"*, significa que se espera que existan coordenadas de textura particulares para el mapa de normales y si en la lectura del color no fueron establecidas coordenadas de textura, deberán considerarse estas coordenadas como los valores *UV* principales. De forma anidada, para las características de la misma naturaleza subsecuentes deberá aplicarse igualmente este procedimiento.

Finalmente, para convertir un objeto tridimensional con un formato como ".obj", ".3ds" u otro, se tienen que tomar los valores que definen una meta-esfera y escribirlos en el formato propuesto.

3.5. Vectores normales

Los vectores normales son un elemento importante para que una superficie pueda interactuar con una fuente de iluminación. Si partimos de una geometría ya construida utilizando un motor de modelado, basta con calcular los vectores normales por vértice. En caso contrario hay que calcularlos, Victoria Ramírez[19] en su trabajo implementa un método para la obtención de normales a través del calculo del cambio de la función de densidad. Sin embargo, en esta investigación se propone un método para su calculo basado en la definición de meta-esfera, es decir, en la influencia ejercida de cada meta-esfera sobre un punto.

El vector normal ($\overrightarrow{N}$) en una esfera ordinaria se obtiene calculando el vector entre en el centro de la esfera (c) y un punto (p) de interés (el vector resultante debe estar normalizado). Si en una escena solo hay una meta-esfera, el calculo de los vectores normales es el mismo ya descrito.

$$\overrightarrow{N}_{IS}(p) = \frac{p - c}{|p - c|}$$

Por lo tanto, si dos o más meta-esferas están influyendo sobre un punto (p) de interés, el vector normal se calcula con la sumatoria del valor porcentual de la influencia de cada meta-esfera sobre el punto de interés, ecuación (3.5).

$$\overrightarrow{N}_{IS}(p) = \sum_i \left(\frac{f_i}{F} \cdot \frac{p - c_i}{|p - c_i|} \right) \tag{3.5}$$

3.5.1. Mapas de normales en Meta-esferas

Para aplicar un mapa de normales a una superficie de meta-esferas, al igual que la mezcla de color, se utiliza la definición de meta-esfera, para así modificar el vector normal de acuerdo a un mapa de normales asociado, ecuación (3.6). Tomando en consideración los límites del mapa de normales ($X : [-1, 1]$, $Y : [-1, 1]$ y $Z : [0, 1]$).

$$\overrightarrow{N_f} = \left(\overrightarrow{N}_{IS} + \sum_i \frac{f_i}{F} \cdot NormalMap.\overrightarrow{N} \right) \cdot 0.5 \tag{3.6}$$

Algoritmo 6: Algoritmo que establece textura, calcula vectores normales y aplica mapa de normales a superficies de meta-esferas.

```
. . .
foreach Punto do
    foreach Meta-esfera do
        f ← Calcula_f(meta-esfera)
        if f > 0 then
            F ← F + f
            /* Color                                              */
            Cmap ← GetColorTex(uv, Tex)
            Col_r ← Col_r + (f * Cmap_r)
            Col_g ← Col_g + (f * Cmap_g)
            Col_b ← Col_b + (f * Cmap_b)

            /* Normal                                             */
            N ← N + (f * cp)

            /* Mapa Normal                                        */
            Cmap ← GetColorTex(uv, NormalMap)
            nm_x ← nm_x + (f * Cmap_x)
            nm_y ← nm_y + (f * Cmap_y)
            nm_z ← nm_z + (f * Cmap_z)
        end
    end
    if F > 0 then
        Col_rgb ← Col_rgb/F
        N ← N/F
        nm_xyz ← nm_xyz/F
    end
end
. . .
```

3.6. Iluminación en superficies de Meta-esferas

La iluminación sobre una superficie nos permite conocer con cuanta intensidad se va a representar un punto. A diferencia de lo abordado en este capítulo, la iluminación no depende de la técnica de meta-esferas, por lo que tras calcular el color final y el vector normal sobre un punto en el espacio, el siguiente paso es utilizar estos valores, junto con el respectivo vector de una fuente de luz para aplicar una técnica de iluminación como *Diffuse Shading*[43].

$$\begin{aligned} Color_f.r &= (amb * Color.r) + (diff * Color.r * (\overrightarrow{N} \cdot \overrightarrow{l})) \\ Color_f.g &= (amb * Color.g) + (diff * Color.g * (\overrightarrow{N} \cdot \overrightarrow{l})) \\ Color_f.b &= (amb * Color.b) + (diff * Color.b * (\overrightarrow{N} \cdot \overrightarrow{l})) \end{aligned} \tag{3.7}$$

Donde *amb* y *diff* son los componentes ambiental y difuso respectivamente, $\overrightarrow{l}$ es el vector de la fuente de luz y $\overrightarrow{N} \cdot \overrightarrow{l} > 0$.

Cabe aclarar que el color se encuentra dentro del rango $[0, 1]$, por lo que si las constantes ambiental

y/o difusa son grandes, pueden provocar saturación de color o artefactos. Por eso es importante limitar el resultado de la ecuación (3.7). En el algoritmo 7 se detalla con más detalle esta técnica.

En la figura 3.12 Se observa la misma superficie mostrada en la figura 3.8, a la cual se le ha aplicado un mapa de normales, ilustrado del lado izquierdo de la imagen. La superficie se visualiza sombreada tanto del lado izquierdo, como del derecho e iluminado en la parte central, que es el resultado esperado de modificar los vectores normales con el mapa propuesto.

Figura 3.12: Aplicación de textura y mapa de normales a meta-esferas.

Algoritmo 7: Algoritmo *Diffuse Shading.*

$[R_f, G_f, B_f] \leftarrow$ **DiffuseShading** *(R, G, B, Normal, Light, cteAmb, cteDiff)*

$R_f \leftarrow R * cteAmb$
$G_f \leftarrow G * cteAmb$
$B_f \leftarrow B * cteAmb$

$dir \leftarrow Dot(Normal, Light)$
if $dir > 0$ **then**
  $R_f \leftarrow R_f + (R * cteDiff * dir)$
  $G_f \leftarrow G_f + (G * cteDiff * dir)$
  $B_f \leftarrow B_f + (B * cteDiff * dir)$
end

$R_f \leftarrow min(R_f, 1.0)$
$G_f \leftarrow min(G_f, 1.0)$
$B_f \leftarrow min(B_f, 1.0)$
end

Capítulo 4

Resultados

Este trabajo se desarrolló utilizando una computadora portátil con un procesador Intel® Core^{TM} i7-9750H y una tarjeta de video NVIDIA GeForce GTX 1660Ti. Las iso-superficies mostradas fueron calculadas utilizando el modelo de Murakami, ecuación (2.4). La profundidad del octree es de siete niveles para todos los ejemplos mostrados en este capítulo.

4.1. Color y pseudo-color en iso-superficies

Como se discutió en el capítulo 3, la forma más básica de agregar color a iso-superficies es partiendo con el pseudo-color, por lo que en los siguientes ejemplos se muestran superficies que han sido generadas por computadora para facilitar la visualización de la aplicación del color. En la figura 4.1 se presenta una superficie con pseudo-color por intensidad aplicado, visualizándose en color más oscuro aquellas zonas con menor influencia de las meta-esferas vecinas, cabe destacar que para este ejemplo, entre mayor sea el valor de umbral mayor superficie se va a visualizar con menor intensidad y por el contrario, si el valor de umbral es menor, la superficie se va a observar con color plano, para crear esta superficie se utilizaron los valores $r = 0.003$ y $T = 1$.

Este método resulta tener un uso muy particular de las superficies de meta-esferas, ya que como se puede observar la intensidad, es posible notar sectores con menor densidad de elementos o si los valores como el umbral no son los adecuados. Si bien el radio de influencia y el valor de umbral no dependen uno del otro, estos deben tener coherencia con el resultado que se está buscando y este método permite que estos se puedan ajustar en tiempo real.

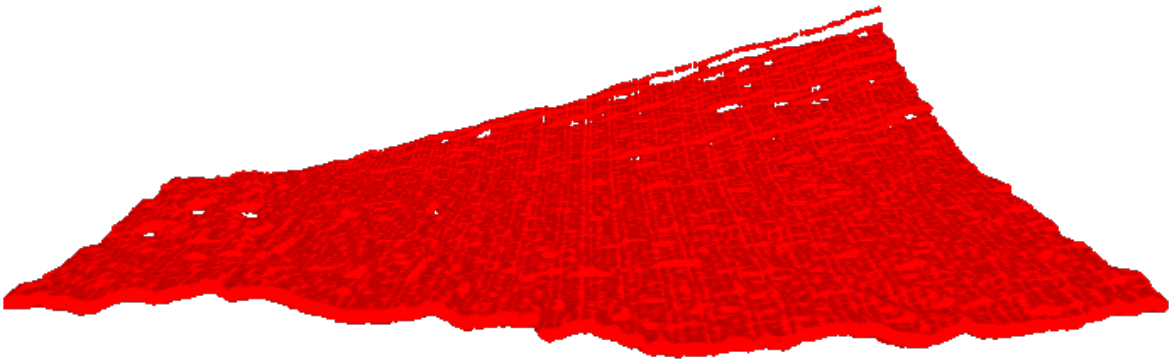


Figura 4.1: Iso-superficie conformada por 10 000 meta-esferas con pseudo-color por intensidad aplicado.

En la figura 4.2 muestra una superficie con pseudo-color por posición, aplicando una banda a cada eje coordinado ($R \to x, G \to y, B \to z$) y con origen en $(0, 0, 0)$, con valores de $r = 0.015$ y $T = 0.01$, donde el único color que no se aprecia es el verde y eso es por que la superficie no tiene un valor en y suficientemente alto para apreciarlo o una mezcla del mismo. Por otro lado, en la figura 4.3 se aplica un mapa de color a una superficie basado en la posición de las meta-esferas, donde el color más claro se muestra si la superficie es cercana al cero en el eje y, con valores de $r = 0.015$ y $T = 0.01$, esta superficie hace uso de todo el mapa, por lo que se puede apreciar diez diferentes colores.

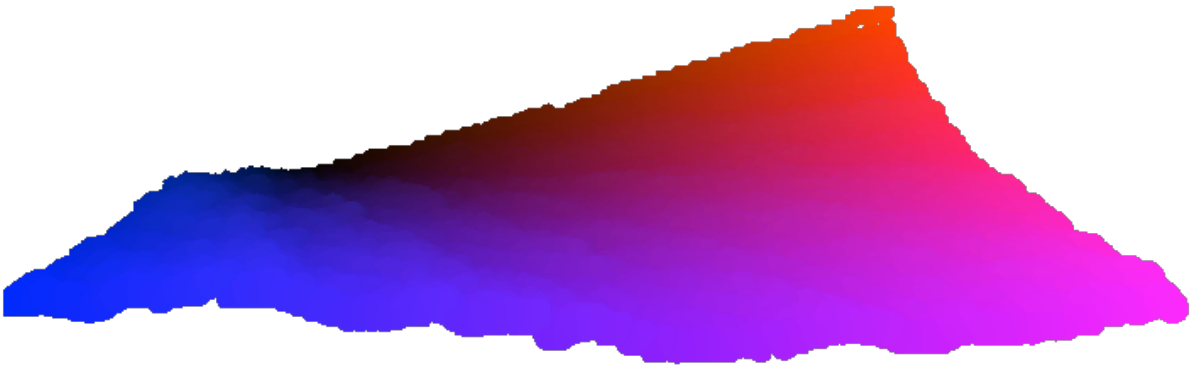


Figura 4.2: Iso-superficie conformada por 10 000 meta-esferas con pseudo-color por posición en el espacio aplicado.

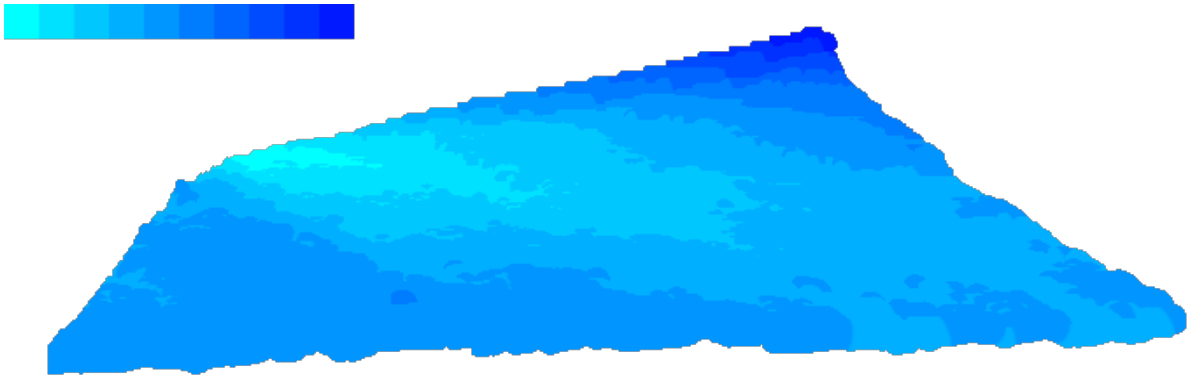


Figura 4.3: Iso-superficie conformada por 10 000 meta-esferas con un mapa de color aplicado.

4.2. Aplicación de textura en meta-esferas

La memoria utilizada en una superficie de meta-esferas depende directamente de la cantidad y tipo de características utilizadas, por ejemplo, si se carga en memoria una superficie que solo contiene posición y valor de umbral, esta utiliza 16 *bytes* por cada meta-esfera; si la superficie está compuesta por 1 000 meta-esferas, el total de memoria es igual a $16 \times 1\,000 = 16\,000$ *bytes*.

Sólo tomando en cuenta lo referente a meta-esferas, la complejidad espacial es lineal y depende de las características que las constituyen. En la tabla 4.1 se muestra la memoria utilizada por diferentes superficies de meta-esferas utilizando todas las características mencionadas en la sección 3.4, donde se está considerando que el color y el mapa de normales comparten las mismas coordenadas de textura. Y se incluye un valor entero para almacenar la clave Morton asociada.

El método implementado tiene un bajo consumo de memoria en el almacenamiento de meta-esferas, esto considerando que el equipo utilizado tiene una capacidad de $16GB$ de memoria RAM y la tarjeta de video tiene $6GB$ de memoria GDDR6, por lo que apenas con $10MB$ es posible guardar 250 000 meta esferas.

En la tabla 4.2 se muestra el rendimiento en el tiempo medido en cuadros por segundo (fps) para las mismas superficies ya mencionadas. Las tres primeras columnas hacen referencia a la iluminación sobre la superficie, mientras que la última columna indica si hay activo un algoritmo de post-procesamiento, específicamente un filtro paso-bajas de bloque de tamaño 5×5. Aquí podemos observar que el rendimiento no se ve afectado si la escena tiene iluminación o esta es dinámica y lo mismo ocurre si pasamos por una etapa de post-procesamiento, aunque está última depende del algoritmo implementado.

Número de meta-esferas					Memoria (kilo bytes)
10	×	10	=	100	4
20	×	20	=	400	16
50	×	50	=	2 500	100
100	×	100	=	10 000	400
200	×	200	=	40 000	1 600
500	×	500	=	250 000	10 000

Cuadro 4.1: Memoria utilizada por iso-superficies con diferentes cantidades de meta-esferas.

	Cuadros por segundo (FPS)			
Número de meta-esferas	Sin iluminación	Con iluminación	Iluminación dinámica	Post-procesamiento
100	142-144	142-144	142-144	140-142
400	139-142	139-142	139-142	124-130
2 500	95-100	95-100	90-94	90-92
10 000	40-43	40-41	40-41	40
40 000	35-37	34-35	34-35	33-34
250 000	22-23	21-22	21-22	21-22

Cuadro 4.2: Rendimiento en FPS del sistema visualizando iso-superficies con diferentes densidades de meta-esferas. Utilizando una tarjeta de video NVIDIA GeForce GTX 1660Ti.

Cabe destacar que el rendimiento en el tiempo depende de más factores que la pueden alterar. En este caso sólo se tomó en consideración el número de meta-esferas mostradas en pantalla y que la superficie ocupe todo el espacio del *octree*. Otro parámetro a tomar en cuenta es cuando la superficie es más pequeña que el *octree*, esto resulta en rayos que no requieren realizar cálculos y por lo tanto el rendimiento aumenta. Por otro lado, si la densidad de meta-esferas es muy alta en espacios pequeños, va a resultar en un exceso de operaciones y por lo tanto el rendimiento disminuye. El rendimiento puede variar si existen elementos en todos los nodos en la vecindad. También se puede ver afectado el rendimiento al utilizar diferente hardware.

En las figuras 4.4 y 4.5 se muestran la imagen y las superficies utilizadas para elaborar la tabla 4.1. En estas, debido a que la resolución de la imagen original es alta ($4160x3120$ píxeles), aplicar una textura a superficies de meta-esferas con menor densidad de meta-esferas que píxeles resulta en una perdida de información, es decir, la iso-superficie se observa con una textura de menor resolución.

Las superficies de meta-esferas ilustradas en la figura 4.5 fueron creadas tomando en cuenta el total de la dimensión del octree en el espacio, el cual se mapea de $(0, 1023)$ a $(0, 1)$ en los tres ejes (meta-esferas fuera el octree no son consideradas dentro de la iso-superficie). Cada meta-esfera se encuentra separada 0.01 unidades entre si, por lo tanto, una superficie de $100x100$ meta-esferas ocupa todo el espacio del octree y superficies con menor densidad se visualizaran de menor tamaño. Para superficies con mayor densidad, las meta-esferas se encuentran separadas $1/w$ y $1/h$ unidades en los ejes x y y respectivamente.

Figura 4.4: Imagen "Avión.png" de tamaño 4160x3120 píxeles (Imagen original).

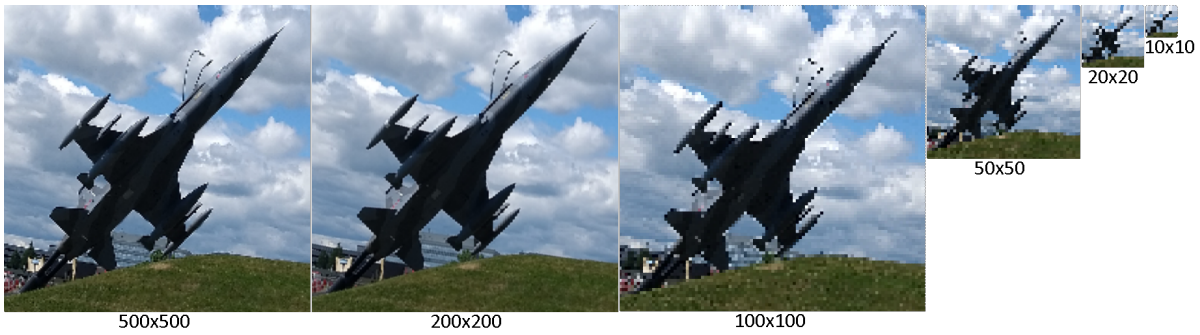


Figura 4.5: Iso-superficies compuestas por 500x500, 200x200, 100x100, 50x50, 20x20 y 10x10 meta-esferas respectivamente.

Similar al ejemplo anterior, en la figura 4.6 se muestra la imagen representando texto con una resolución de 500x500 píxeles, donde cada renglón presenta letras con diferentes tamaños, utilizada para aplicar textura a las superficies mostradas en la figura 4.7, las cuales están compuestas por 10 000, 40 000, 100 000 y 1 000 000 meta-esferas respectivamente. Debido a la resolución de la textura aplicada, la tercera imagen (inferior izquierda) al estar compuesta por un número de meta-esferas igual al número de píxeles, no pierde calidad hasta el último renglón, el cual no es legible. En la primera imagen (superior izquierda) es ilegible a partir del quinto renglón. La segunda y cuarta imagen (superior derecha e inferior derecha) es legible hasta el quinto renglón.

Para las superficies con menor densidad de meta-esferas que píxeles de la textura aplicada, lo esperado es que el resultado se observe con perdida de información, como ya se mencionó anteriormente. Para aquellas superficies con un mayor número de meta-esferas, se esperaría un resultado con mayor detalle. Sin embargo, no es así, esto se debe a que existe un mayor número de meta-esferas influyendo sobre un mismo punto, por lo que detalles finos en la imagen se van a ver afectados, provocando perdida de información.

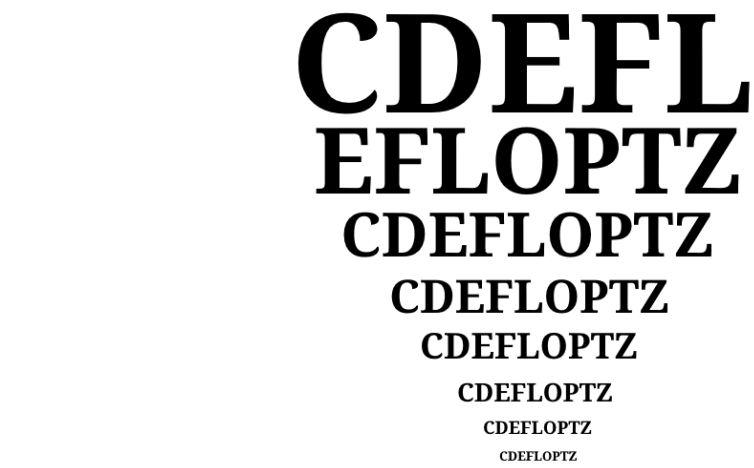


Figura 4.6: Imagen "Letras.png" de tamaño 500x500 píxeles (Imagen original).

Figura 4.7: Iso-superficies compuestas por 100x100 (superior izquierda), 200x200 (superior derecha), 500x500(inferior izquierda) y 1000x1000 (inferior derecha) meta-esferas respectivamente.

En los siguientes ejemplos se visualizan superficies de 200x200 meta-esferas distribuidas sobre el plano XY, influenciadas por una fuente de luz, a las cuales se les aplicó una textura de 200x200 píxeles con su respectivo mapa de normales. Cabe destacar que el radio y el umbral utilizados fueron $r = 0.015$ y $T = 0.01$ respectivamente, lo que resulta en una superficie completamente plana donde podemos destacar la correcta aplicación del mapa de normales.

En la figura 4.8 se presenta la imagen "Renata.png" como textura, en esta, hay pocos colores y hacen contraste entre la perra y el fondo, el pelo del animal no contiene mucho detalle. Esto resulta en un mapa de normales con bordes bien marcados y sólo un poco de ruido detrás del marco blanco. En la superficie final los vectores normales se encuentran apuntando hacia el observador ($\vec{N} = (0, 0, 1)$) y son modificados por el mapa de normales.

Por otro lado, en la figura 4.9 se muestra la imagen "Shanti.png", la gata en cuestión, por su color, hace buen contraste con el fondo. Sin embargo, la abundancia de las plantas ha generado mucho ruido sobre el mapa de normales, generando abundantes artefactos como contornos blancos sobre las plantas más expuestas a la fuente de luz. Otro detalle a destacar es que debido al color del animal no se aprecia la luz sobre de él, e incluso pierde detalle sobre los ojos.

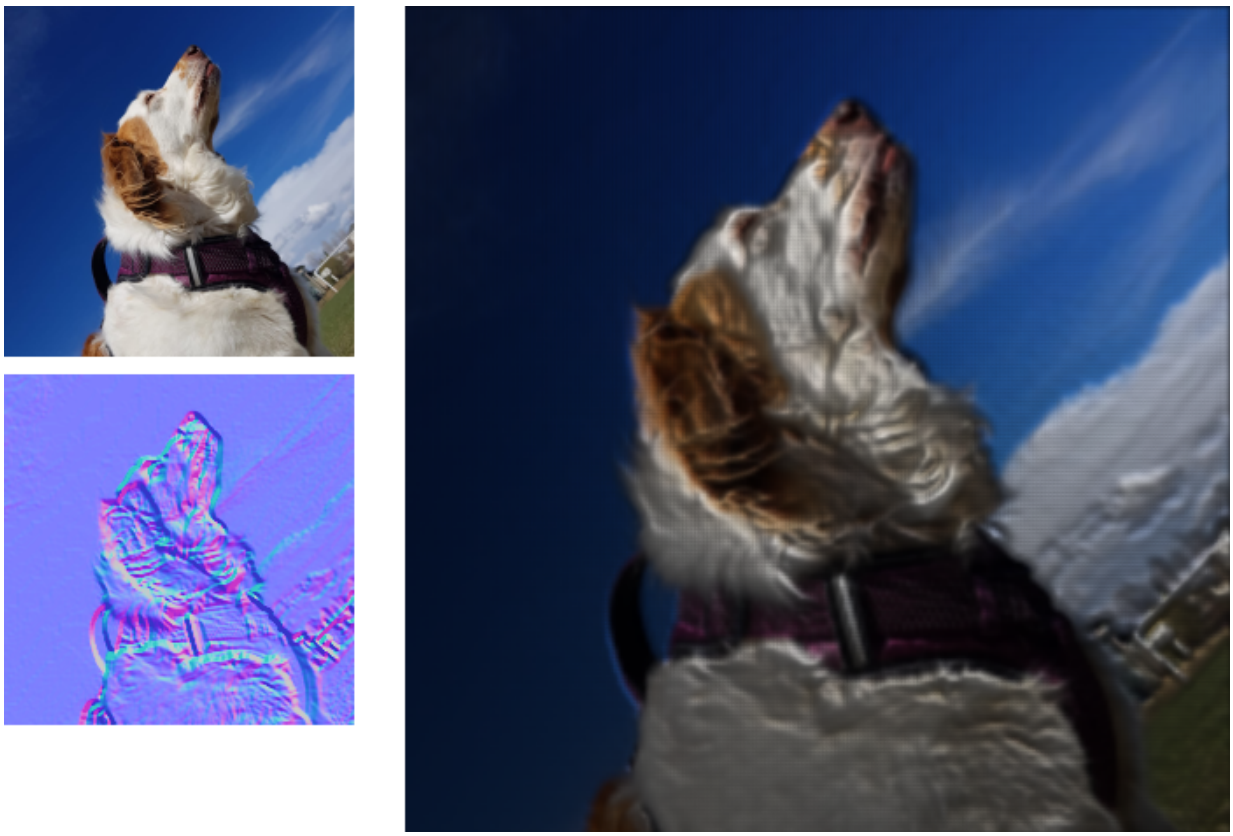


Figura 4.8: Imagen "Renata.png" con su respectivo mapa de normales aplicados a una iso-superficie.

Figura 4.9: Imagen "Shanti.png" con su respectivo mapa de normales aplicados a una iso-superficie.

4.3. Aplicación de textura y color a objetos con volumen

Los resultados de los ejemplos anteriores fueron creados para formar una superficie plana, esta es una manera adecuada de representarlas con cualquier tipo de textura. Sin embargo, las nubes de puntos también contienen profundidad, es decir, las superficies de meta-esferas resultantes deberían formar volúmenes como en la figura 4.10, donde se muestra una iso-superficie esférica texturizada compuesta por 270 977 meta-esferas con un radio de influencia $r = 0.008$ y un umbral $T = 0.7$, con un desempeño en el tiempo de 18 cuadros por segundo, la imagen aplicada busca darle una apariencia de un ojo. La superficie presenta rugosidad sobre toda su área, se ve de esta manera por la distancia que existe entre meta-esferas, no hay suficiente intensidad para construirla uniforme y lisa. También se observan artefactos en forma de pequeñas esferas que sobresalen sobre la superficie, esto aparece por como consecuencia de que un rayo está encontrando una o más meta-esferas en la frontera de dos nodos y esto tiene como resultado que el sistema considera el el espacio del *octree* más cercano, cuando el que se quisiera es el más alejado.

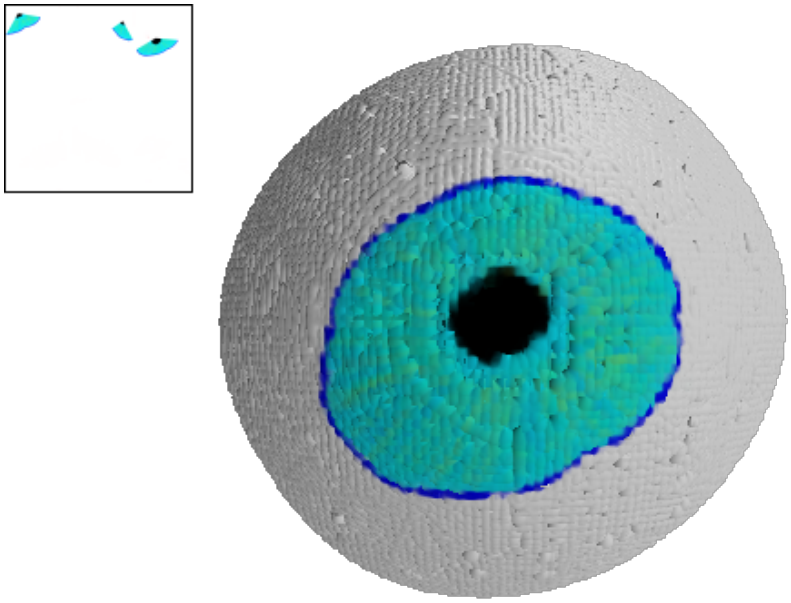


Figura 4.10: Iso-superficie esférica texturizada compuesta por 270 977 meta-esferas, con rendimiento en el tiempo de 18 fps.

Con un total de 251 716 meta-esferas en la figura 4.11 se presenta una iso-superficie del modelo "Suzanne" [37] con los mismos parámetros y características descritas en la figura anterior, con un rendimiento de $18 - 19$ cuadros por segundo. La superficie también presenta artefactos como los ya descritos en la figura anterior y se deben a las mismas razones ya mencionadas, estos se encuentran en menor cantidad y eso es por la distribución de las meta-esferas en el espacio. Por otro lado, de igual forma se aprecia rugosidad sobre todo el volumen, sobre todo debajo de la nariz, esto se debe a que no hay suficiente cantidad de meta-esferas en esa zona. El texturizado también presenta errores, por ejemplo, en los labios el color que se debería ver es marrón oscuro, esto se puede deber a una combinación de dos factores, el primero, que ese color se encuentre dentro de la boca (donde no hay meta-esferas) y la segunda, que hay más densidad de meta-esferas sobre el color café claro.

En los ejemplos anteriores se dio prioridad al número de meta-esferas visibles, por lo que ambas nubes de puntos fueron creadas a partir de modelos huecos y sin geometría en la parte posterior. En la primera superficie (Ojo) no se ve afectada en su visualización ni en su rendimiento por este hecho, no obstante, en la segunda superficie (Suzanne), como su modelo original, los ojos se encuentran separados del resto del modelo, alrededor de estos se observa color negro, esto es resultado de que el fondo del espacio de visualización también es negro. Mas adelante se podrá apreciar este detalle con diferente color.

Figura 4.11: Iso-superficie del modelo "Suzanne" texturizado compuesto por 251 716 meta-esferas, con rendimiento en el tiempo de $18 - 19$ fps.

La arquitectura propuesta ha resultado versátil por lo que añadir módulos de procesamiento solo requiere que sean ejecutados en secuencia (como se define la computación heterogénea).

El siguiente experimento se presenta debido a que esta tesis busca mostrar despliegues en tiempo real mayores a treinta cuadros por segundo, por lo que para mantener este rendimiento, no es viable calcular numerosos rayos dentro del algoritmo de *ray tracing* y en su lugar se utilizan técnicas de procesamiento digital de imágenes sobre el resultado final[31][32][33]. En los siguientes ejemplos se observan los volúmenes de las superficies anteriores, donde la primera imagen solamente tiene una fuente de iluminación activa y al resto se les ha aplicado además un filtro paso bajas en una etapa de post-procesamiento, dicho filtro cuenta con un kernel de tamaño 3x3, 5x5, 7x7, 9x9 y 11x11 respectivamente. El rendimiento no se ve afectado para ningún tamaño de kernel utilizado, por lo que para todos los casos el sistema se ejecuta a 18 fps.

En la figura 4.12, la superficie se comienza a ver uniforme utilizando un kernel de tamaño 5x5, a pesar de que los artefactos se sigan visualizando; pero es cuando se utiliza un filtro de 9x9 cuando dichos artefactos comienzan a desaparecer. Por otro lado, la textura utilizada tiene poco detalle, lo que beneficia el uso de un kernel de mayor tamaño.

A diferencia de la figura anterior, en la 4.13 aquellos defectos descritos para este ejemplo comienzan a desaparecer al usar un kernel de 7x7. Sin embargo, como la superficie tiene una forma más compleja y tiene una textura con más detalle, el uso de filtros de mayor tamaño puede resultar en perdida de información. En esta figura, dicha perdida se puede visualizar en las orejas y en la cabeza, el uso de un filtro que suaviza, hace que el color se funda y se pierda la información de profundidad.

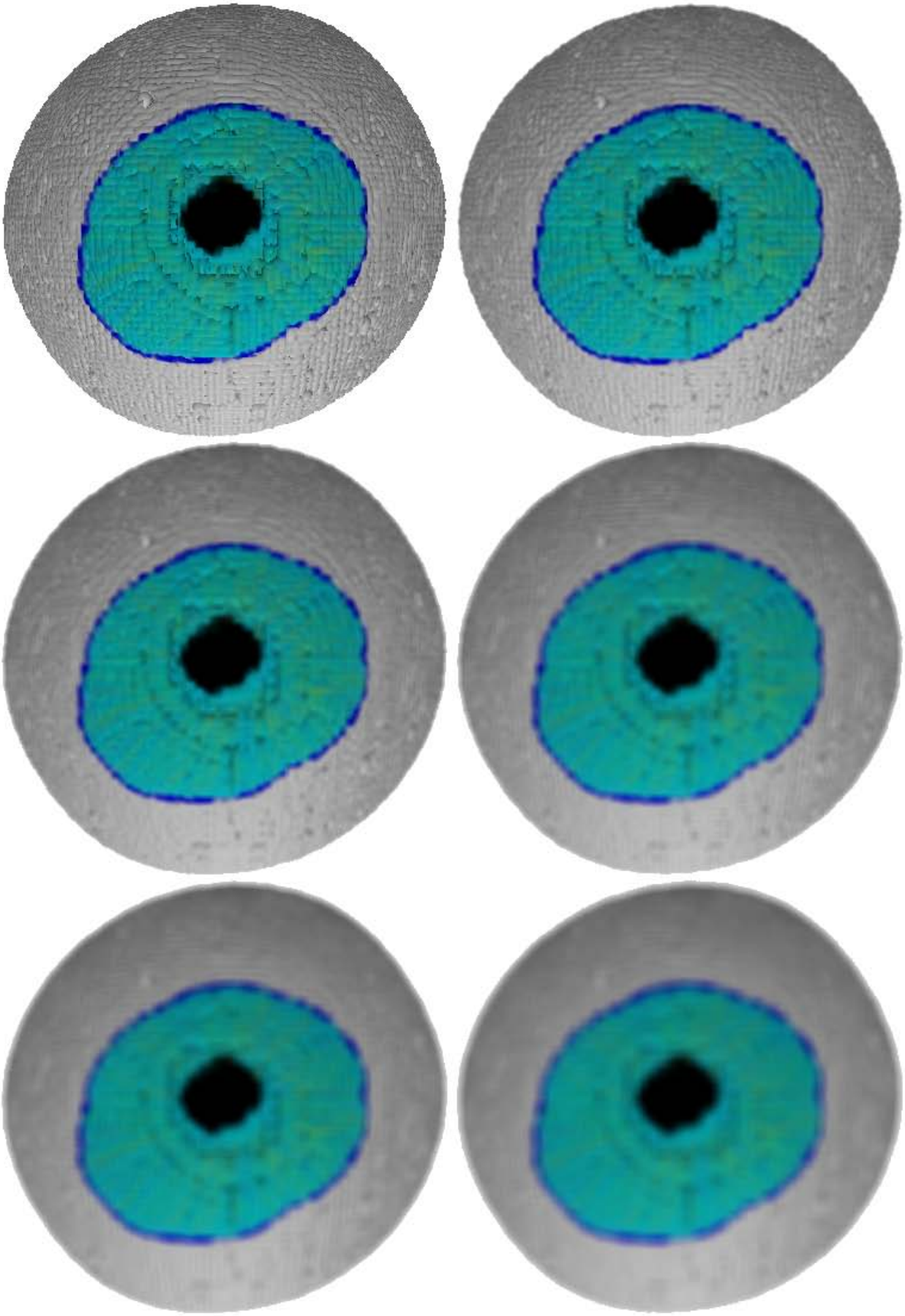


Figura 4.12: Aplicación de una etapa de post-procesamiento a iso-superficies. Filtro paso-bajas de 3x3, 5x5, 7x7, 9x9 y 11x11.

Figura 4.13: Aplicación de una etapa de post-procesamiento a iso-superficies. Filtro paso-bajas de 3x3, 5x5, 7x7, 9x9 y 11x11.

4.4. Aplicación del método a iso-superficies adquiridas por visión por computadora

En la figura 4.14 se observa la reconstrucción tridimensional de una superficie adquirida con métodos de visión por computadora (utilizando fotogrametría) y exportada a diferentes resoluciones. Proporcionando nubes de puntos de una excavación arqueológica a las que se aplicó la técnica de meta-esferas, resultando en cuatro superficies con 63 334 (Superior izquierda), 126 140 (Superior derecha), 251 521 (Inferior izquierda) y 502 006 (Inferior derecha) meta-esferas respectivamente, utilizando las texturas mostradas en la figura 4.15.

En las iso-superficies se aprecian las diferentes profundidades de la excavación. Por otro lado, se aprecia en este ejemplo como el texto localizado en el objeto color blanco es ilegible, así como otros detalles finos donde se aprecia la perdida de información, ubicados principalmente en la parte más elevada de la excavación.

Figura 4.14: Comparación de superficies con diferente cantidad de meta-esferas. (Superior izquierda) con 63 334 meta-esferas, $r = 0.009$ y $T = 0.7$. (Superior derecha) con 126 140 meta-esferas, $r = 0.005$ y $T = 0.7$. (Inferior izquierda) con 251 521 meta-esferas, $r = 0.005$ y $T = 0.7$. (Inferior derecha) con 502 006 meta-esferas, $r = 0.005$ y $T = 0.7$.

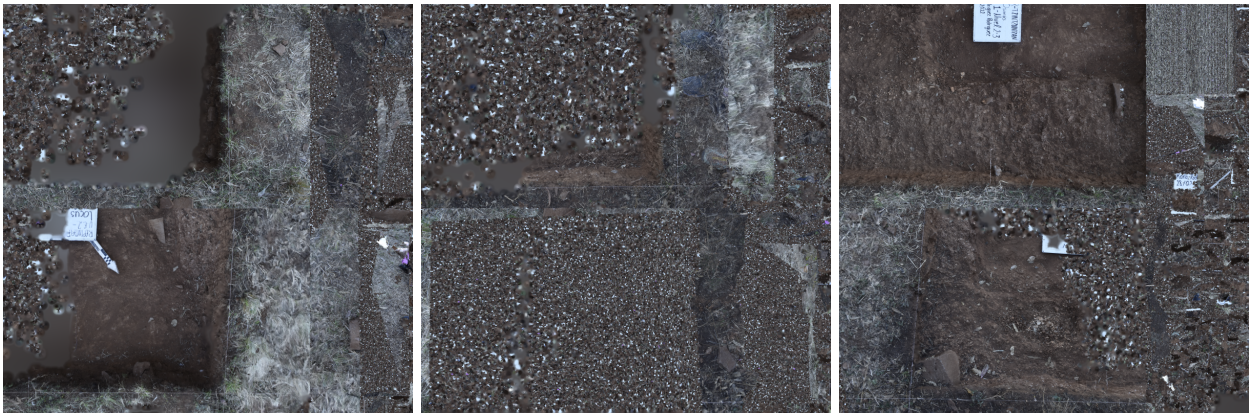


Figura 4.15: Texturas utilizadas para para dar color a las superficies representadas en la figura 4.14. La resolución de las imágenes es de $8\,000x8\,000$ píxeles.

Capítulo 5

Conclusiones

A través de esta investigación se desarrolló e implemento un método para añadir color y textura a superficies de meta-esferas. La necesidad de ello se comprende al reconocer el hecho de que la mayoría de las implementaciones qué había en la bibliografía disponible utilizaban otros métodos no nativos para ese fin, complicando con ello su uso, sin mencionar el alto costo computacional que de por si implica el uso de meta-esferas. Por lo que se hizo necesario dividir nuestro problema de investigación en la resolución de cinco grandes temas: el análisis de la definición de meta-esferas, la aplicación de color y textura, la iluminación sobre la superficie, el rendimiento y la implementación.

En primera instancia, al hacer la revisión de toda la bibliografía relativa a este tema, aquellos trabajos interesados en el avance de esta tecnología, principalmente los citados en la sección 2.1, fueron indispensables para comprender lo que las meta-esferas son y sus descripciones proporcionaron la definición que ha servido de base para investigar las posibles modificaciones que serían necesarias para alcanzar el objetivo. La revisión sistemática específica probó muy pronto la inviabilidad de esta primera definición al no poder aplicar color aparte de pseudo-color. Entonces se tornó necesario ampliar aquella definición, permitiendo así incluir información que se resolvería como mapas de color o textura. De esta misma forma también se podría incluir otros datos, como lo son los mapas de normales, que junto a sus vectores normales, dieron lugar a la representación de superficies de meta-esferas influidas por un modelo de iluminación.

Se desarrollaron tres modelos para la aplicación de color a superficies de meta-esferas, todos ellos tomando como base el cálculo de la mezcla de color. Los casos de pseudo-color y mapas de color han probado proporcionar información visual útil y con cualquier modelo se puede lograr un resultado estéticamente versátil.

Por definición, en meta-esferas se calcula una intensidad sobre un punto, donde cada elemento va a contribuir al resultado. Tomando esto en consideración, cada meta-esfera también aportará una cantidad de color sobre un punto. Este fue el razonamiento que se siguió y dio lugar al cálculo de la mezcla de color (en [2] se presentó una aproximación similar) y posteriormente al cálculo de los vectores normales sobre la superficie. Por lo tanto, en el caso que se requiera integrar otra propiedad, ésta debería seguir esta base para su cálculo.

Por otro lado la implementación de textura se desarrolló considerando que la nube de puntos tiene que ser invariante, por lo que para trabajos como [10], donde la superficie de meta-esferas puede cambiar de forma, esta implementación resulta insuficiente. Sin embargo, la adición de texturas tridimensionales al algoritmo daría solución a este problema y en el ejemplo específicamente se omitiría el trabajo de construir una malla para su visualización.

Una limitante a considerar es que a diferencia de la visualización utilizando mallas, al aplicar una textura a una iso-superficie hay que tomar en consideración el tamaño de la imagen, este debe ser coherente con el número de meta-esferas, de lo contrario se perderá información y el resultado se observara con menor

definición de la esperada, como se puede ver en las superficies presentadas en las figuras 4.14 y 4.7. En el caso de que la densidad de meta-esferas sea menor, lo esperado es que la superficie resultante no se encuentre completamente cerrada, es decir, que existan huecos o que la superficie se muestre con perdida de información. En el caso contrario, también se espera perdida de información (principalmente en detalles finos) debido a la saturación de meta-esferas en una región. En [2], el autor menciona no haber logrado aplicar textura, con este método ahora es posible hacerlo de forma nativa.

Una desventaja de utilizar texturas obtenidas del mundo real es que la imagen misma puede contener información no deseada como iluminación y sombras, la lente puede ser inadecuada o pueden existir elementos pequeños como en los ejemplos de las figuras 4.7 y 4.9. Si a eso le agregamos que la superficie utilizada carece de profundidad, los resultados en primera instancia sólo tendrían un carácter estético. Sería necesario el análisis e implementación de algoritmos, como por ejemplo, de segmentación, para darle utilidad.

A diferencia de nubes de puntos que resultan en superficies planas, aquellas que forman volúmenes son más propensas a generar artefactos si existen meta-esferas en las fronteras del *octree* o a visualizarse de forma incorrecta si no hay una adecuada distribución de meta-esferas. Además, es necesario dedicar más esfuerzo en establecer los parámetros de las meta-esferas (los cuales se realizan a prueba y error) adecuados para su representación. Por otro lado, el rendimiento se ve afectado al utilizar volúmenes, principalmente debido a la densidad e inadecuada distribución de meta-esferas, pero también por el cálculo de vecindades con elementos válidos dentro de ellas.

En la actualidad el uso de filtros 2D es una practica habitual para visualizar objetos realistas tiempo real como videos, animaciones o sistemas interactivos basados en ray tracing. Aun con la tecnología moderna, realizar esta tarea no es una opción.

De la necesidad de visualizar diferentes iso-superficies, se propuso también un formato de archivo para el almacenamiento de meta-esferas. Esto facilitó la tarea de analizar superficies con diferentes propiedades como color o coordenadas de textura, aplicándose a diferente modelos de color.

El rendimiento del sistema fue un tema muy importante durante el desarrollo, gracias a avances en el estado del arte como el uso del octree y la clave Morton en nube de puntos[19][20], fue posible optimizar el rendimiento considerablemente. La complejidad computacional expuesta en la sección 2.3, diseñada para simulación de partículas, está basada en dichos avances y esta se puede mejorar siempre y cuando la nube de puntos no sea sujeto de variaciones. Así pues, la combinación de técnicas para reducir el coste computacional, impactando notablemente en este, ya que calcular una superficie de dieciséis, veinticinco o cincuenta meta-esferas (como en las figuras 3.8 y 3.12) podía tardar hasta veinte segundos en completarse. Sin embargo, al usar computación en paralelo, octree y clave Morton, se logró representar superficies de meta-esferas con cientos de miles de éstas en tiempo real. Otro punto a mencionar es que el equipo utilizado (descrito en el capítulo 4) influye directamente en los resultados, por lo que éstos pueden mejorar utilizando hardware con mejores especificaciones.

Una limitación del espacio del *octree* es que solamente existe en el rango de $[0, 1]$, por lo que cualquier superficie debe pertenecer también a este espacio, el cual, reduce la distancia en la que se puede representan o simular meta-esferas.

Debido a la arquitectura de la implementación, es simple adicionar nuevos módulos al algoritmo, ya sean de pre-procesamiento o de post-procesamiento. En esta tesis se implementó un filtro paso-bajas en una etapa de post-procesamiento (figuras 4.12 y 4.13) y sin impacto en el rendimiento comparándolo con el algoritmo original; aunque este depende del método implementado. Sin embargo, la futura implementación de filtros más robustos proporcionará un panorama más amplio del funcionamiento del sistema y sus posibles limitantes.

5.1. Trabajo futuro

Como ya se mencionó en la sección 5, una textura se carga una única ocasión para asociar las coordenadas de textura a una nube de puntos o esta información ya se encuentra almacenada en un archivo de meta-esferas. Por lo que para esta última, provocaría artefactos si la nube de puntos se ve alterada. Pero para la situación mencionada al inicio, un primer paso sería realizar el calculo de las coordenadas de textura para aquellos elementos que presenten modificaciones. Esto puede aplicarse a simulación de fluidos y a volúmenes que presenten poco cambio ya que difícilmente generaría artefactos significativos. Por otro lado, la implementación de texturas tridimensionales permitiría la visualización de objetos de formas complejas que cambien en el tiempo.

Apéndice A

Modelos de meta-esfera creados a partir de la ecuación de la esfera

A.1. Modelo A

Como se mencionó en la sección 2.1, uno de los modelos más simples matemáticamente proviene directamente de la ecuación de la circunferencia en un espació 2D o de la esfera en un espacio 3D:

$$f(x,y,z) = (x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2 = r^2 \mid r > 0$$

Donde el punto (x_0, y_0, z_0) es el centro de la esfera y r es el radio de la esfera.

Si sacamos raíz cuadrada en ambos lados de la ecuación obtenemos:

$$f(x,y,z) = \sqrt{(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2} = \sqrt{r^2}$$

$$f(x,y,z) = \sqrt{(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2} = r$$

Dividiendo ambos lado de la ecuación sobre r:

$$f(x,y,z) = \frac{\sqrt{(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2}}{r} = 1$$

$$f(x,y,z) = \frac{\sqrt{(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2}}{r} - 1 = 0$$

Si consideramos que (x, y, z) es un punto cualquiera en el espacio, el dividendo de la fracción es igual a la distancia desde el centro de la esfera hacia un punto (x, y, z).

$$f(x,y,z) = \frac{d}{r} - 1 = 0$$

Por ser una función implícita, sabemos que aquellos valores de (x, y, z) que den como resultado $f < 0$ se encuentran en el interior de la esfera y aquellos que den como resultado $f = 0$ son parte del iso-contorno o interfaz. Por lo tanto:

$$f(x, y, z) = \frac{d}{r} - 1 \leq 0$$

Por lo dicho anteriormente, el número 1 en la ecuación es el máximo número posible en el que f pertenece a la esfera. Si tomamos este valor como un valor de umbral T que sea variable y $T \leq 1$:

$$f(x, y, z) = \frac{d}{r} - T \leq 0 \mid T \leq 1,\ r > 0 \tag{A.1}$$

A.2. Modelo B

Este segundo modelo es el más simple matemáticamente, como en la sección anterior, partimos desde la ecuación de la esfera:

$$f(x, y, z) = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2 \mid r > 0$$

Donde el punto (x_0, y_0, z_0) es el centro de la esfera y r es el radio de la esfera.

Si sacamos raíz cuadrada en ambos lados de la ecuación obtenemos:

$$f(x, y, z) = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2} = \sqrt{r^2}$$

$$f(x, y, z) = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2} = r$$

Hasta este punto desarrollamos igual que el modelo anterior, sin embargo, ahora solamente despejamos la distancia restando d (que ya sabemos que es la raíz cuadrada) en ambos lados de la ecuación

$$f(x, y, z) = -d + r = 0 \tag{A.2}$$

La ecuación (A.2) tiene como particularidad que se parece a una función rampa de la forma $y = mx + b$, donde m es la pendiente y b es el radio. Por lo que es fácilmente modificable y adaptable a diferentes circunstancias como se muestra en la figura A.1.

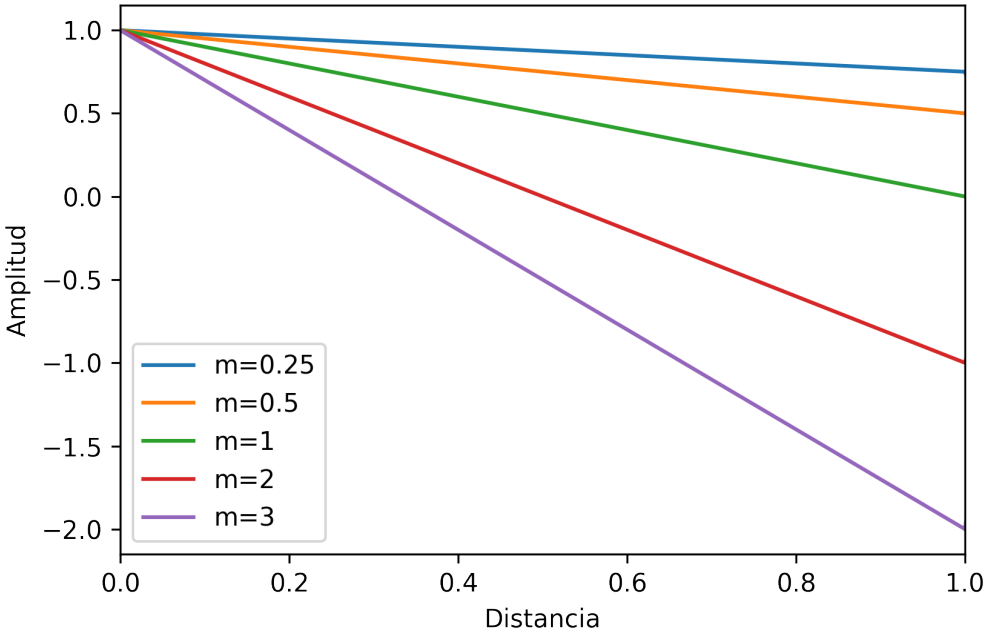


Figura A.1: Modelos de meta-esferas con diferentes pendientes.

También lo podemos extender a funciones de mayor grado de la siguiente manera, (figura A.2).

Para una ecuación de segundo orden tenemos:

$$f(x, y, z) = (-d + r) \cdot (d + r) = 0$$

Para una ecuación de tercer orden:

$$f(x, y, z) = (-d + r) \cdot (d + r) \cdot (-d + r) = 0$$

Para una ecuación de cuarto orden:

$$f(x, y, z) = (-d + r) \cdot (d + r) \cdot (-d + r) \cdot (d + r) = 0$$

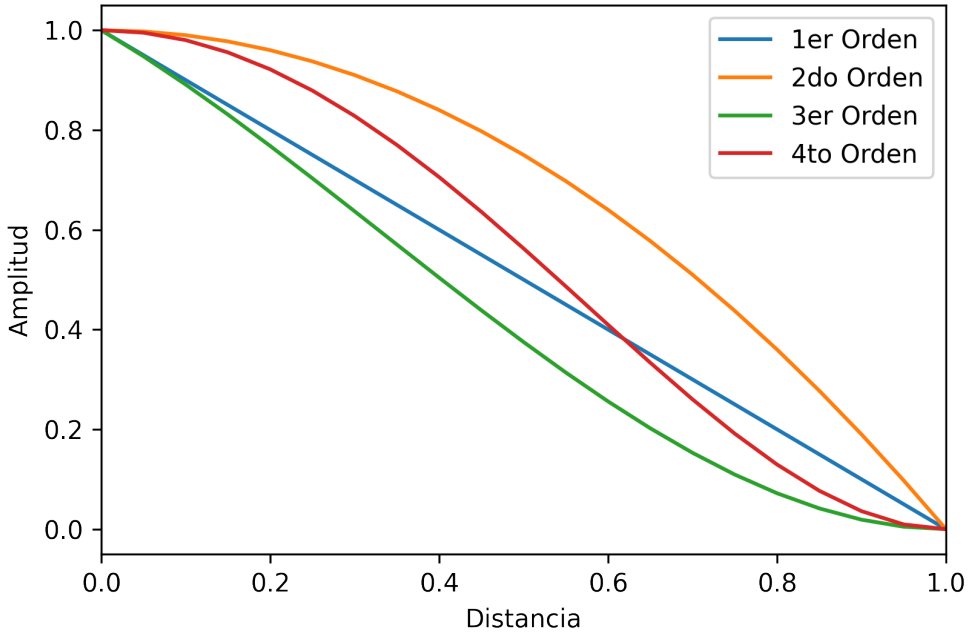


Figura A.2: Modelos de meta-esferas con diferentes grados de polinomio.

Apéndice B

CUDA

Compute Unified Device Architecture (CUDA)[34][35][36] desarrollada por la empresa NVIDIA Corporation y cuya primera versión fue publicada en el año 2007. Es una tecnología que permite la ejecución de código de propósito general en la *Unidad de Procesamiento Gráfico* (GPU) para obtener soluciones en paralelo a problemas computacionales sin la necesidad de conocer un *pipeline* de gráficos por computadora. A continuación se mencionan algunas de las ventajas que el uso de CUDA proporciona[21].

- Permite la transferencia de datos entre el CPU y el GPU. De esta manera, el CPU ejecuta código secuencial y el GPU ejecuta segmentos de código para múltiples datos de entrada (conocido como *Single Instruction-Multiple Data* o SIMD). A esta arquitectura se le denomina computación heterogénea.
- El número de bloques e hilos (componentes de la GPU) pueden ser adaptados a cada problema en particular, permitiendo la implementación de soluciones óptimas. Una vez establecidos para un hardware específico, la solución puede ser ejecutada utilizando hardware más moderno o adaptada para mejorar su rendimiento sólo adaptando los bloques e hilos a utilizar.
- Lenguaje de programación utilizado en CUDA está basado en el lenguaje de programación C, por lo que su estructura es similar a la utilizada en C/C++, siendo compatible con esta ultima de forma nativa. Además, es posible utilizar otros lenguajes como Python, Java, Julia, entre otros, utilizando un adaptador adecuado.

Una implementación en CUDA utiliza ambos procesadores. El CPU y el GPU, nombrados como *host* y *device* respectivamente. El primero es responsable de llamar a las funciones en el *device* llamadas *kernel*. Una de las diferencias más significativas entre el CPU y el GPU es que la primera está pensada para ejecutar código secuencial de baja latencia, mientras que la segunda es eficiente para ejecutar código en paralelo. En la figura B.1 se observa la diferencia entre las capacidades de paralelismo entre un CPU y un GPU, donde una CPU depende del número de núcleos para ejecutar procesos en paralelo, mientras que el GPU puede ejecutar cada hilo dentro de cada bloque en paralelo.

En CUDA un bloque es una entidad lógica conformada por un conjunto de hilos. La dimensión de un bloque es definida por el usuario y es almacenada en *blockDim*. Cada hilo y bloque tiene los identificadores *threadIdx* y *blockIdx* respectivamente. Utilizando estas variables es posible saber exactamente con que hilo se está trabajando. Además, hilos dentro de un mismo bloque tienen mecanismos de comunicación y sincronización mediante un tipo de memoria llamada *shared memory*. Los bloques no pueden sincronizarse entre si ya que estos pueden ser resueltos tanto en serie, como en paralelo.

Para ejecutar código en el GPU, primero hay que establecer los datos que van a servir como parámetros dentro de un *kernel*. Para ello hay que transferirlos desde el *host* hacia el *device*, reservando su respectivo espacio de memoria. Una vez con los datos en la memoria de video se procede a llamar a un *kernel* para ejecutarlo en el GPU. En el estado de ejecución, la forma de conocer el estado de los procesos es sincronizando

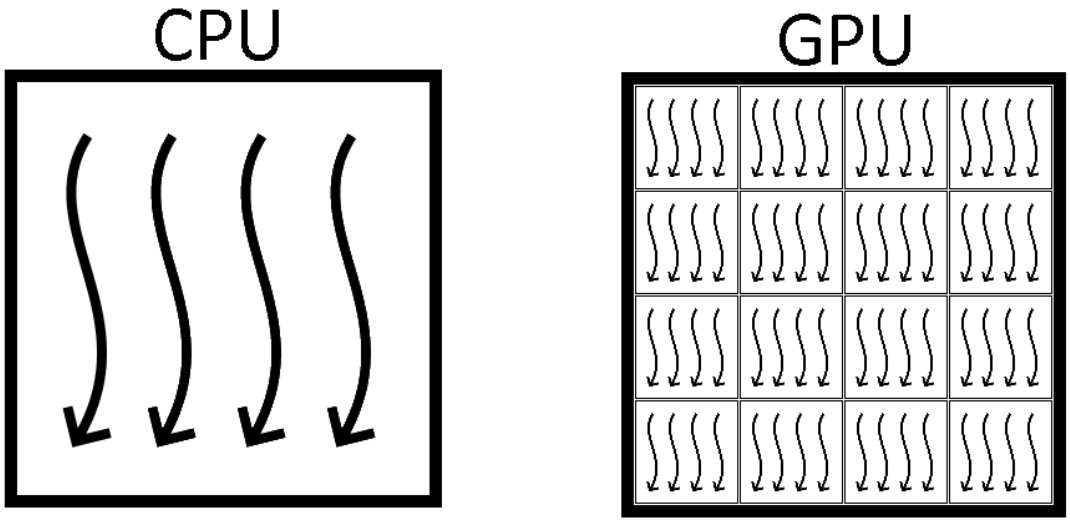


Figura B.1: Comparación de capacidad de ejecución en paralelo entre una CPU y una GPU.

los hilos, de esta forma se puede controlar y llevar un orden de ejecución. Finalmente, para hacer el análisis de los resultados, es imperativo transferirlos desde *device* hacia el *host*.

En el algoritmo 8 se muestra el código necesario para calcular la suma de dos arreglos en paralelo utilizando N bloques. Donde se utiliza el identificador *blockIdx* para para especificar que bloque va a tomar que elemento del arreglo para sumar. Es decir, el bloque con índice cero va a sumar los elementos con índice cero del arreglo a y b, el bloque uno con los elementos de índice uno y así sucesivamente. $\{Bloque[0] \rightarrow c[0] = a[0] + b[0]\}$, $\{Bloque[0] \rightarrow c[1] = a[1] + b[1]\}$, etcétera.

Algoritmo 8: Ejemplo de suma de vectores en CUDA.

```
#include<stdio.h>
#include<stdlib.h>
#define N 512

void fill_array(int *data)
{
    for (int i = 0; i < N; i ++)
        data[i] = i;
}

void print_output(int *a, int *b, int*c)
{
    for (int i = 0; i < N; i ++)
        printf(``%d + %d = %d\n'', a[i], b[i], c[i]);
}

__global__ void device_add(int *a, int *b, int *c)
{
    c[blockIdx.x] = a[blockIdx.x] + b[blockIdx.x];
}

int main(void)
{
    int *a, *b, *c; // Host-Array
    int *d_a, *d_b, *d_c; // Device-Array
    int size = N * sizeof(int);

    // Host-Space allocation of a, b, and c
    a = (int *)malloc(size); fill_array(a);
    b = (int *)malloc(size); fill_array(b);
    c = (int *)malloc(size);
    // Device-Space allocation for d_a, d_b and d_c
    cudaMalloc((void *)&d_a, N * sizeof(int));
    cudaMalloc((void *)&d_b, N *sizeof(int));
    cudaMalloc((void *)&d_c, N * sizeof(int));
    // Device-Copy a and b from host to d_a and d_b in device
    cudaMemcpy(d_a, a, N * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b, N* sizeof(int), cudaMemcpyHostToDevice);
    // Device-Call add vector function
    device_add<<<N,1>>>(d_a,d_b,d_c);
    // Host-Copy d_c from device to c in host
    cudaMemcpy(c, d_c, N * sizeof(int), cudaMemcpyDeviceToHost);
    // Host-Print result
    print_output(a, b, c);
    // Host-Device-Free memory
    free(a); free(b); free(c); cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
    return 0;
}
```

Bibliografía

[1] Blinn, J. F. (n.d.). A Generalization of Algebraic Surface Drawing. ACM Transactions on Graphics (TOG), 1(3), 235–256. https://doi-org.pbidi.unam.mx:2443/10.1145/357306.357310

[2] HITOSHI NISHIMURA, ISAO SHIRAKAWA, KOHICHI OHMURA, MAKOTO HIRAI, TOHRU KAWATA, 大村 皓一, 平井 誠, 河田 亨, 白川 功, & 西村 仁志. (1983). An object modeling with distribution function and a method for efficient image creation. テレビジョン学会技術報告, 7(24), 21. https://doi-org.pbidi.unam.mx:2443/10.11485/tvtr.7.24_21

[3] Wyvill, G. (1), McPheeters, C. (2), & Wyvill, B. (2). (n.d.). Data structure for soft objects. The Visual Computer, 2(4), 227–234. https://doi-org.pbidi.unam.mx:2443/10.1007/BF01900346

[4] Zhang, P., Dong, Y., Galindo-Torres, S. A., Scheuermann, A., & Li, L. (2021). Metaball based discrete element method for general shaped particles with round features. Computational Mechanics, 67(4), 1243.

[5] Nishita, T. and Nakamae, E. (1994), A Method for Displaying Metaballs by using Bézier Clipping. Computer Graphics Forum, 13: 271-280. https://doi.org/10.1111/1467-8659.1330271

[6] Jay Jeong Kim, En-Seok Kim, & Seung-Ki Park. (1997). An automatic description of volumetric objects using metaballs. Proceedings Computer Graphics International, Computer Graphics International, 1997. Proceedings, 65–73. https://doi-org.pbidi.unam.mx:2443/10.1109/CGI.1997.601274

[7] Xiaogang Jinq, Li, Y. F., & Qunsheng Peng. (1998). General constrained deformations based on generalized metaballs. Proceedings Pacific Graphics '98. Sixth Pacific Conference on Computer Graphics and Applications (Cat. No.98EX208), Computer Graphics and Applications, 1998. Pacific Graphics '98. Sixth Pacific Conference On, 115–124. https://doi-org.pbidi.unam.mx:2443/10.1109/PCCGA.1998.732061

[8] Dongfang Chen, & Jiahuan Zhang. (2010). The Merging of Water Droplets Base-on Metaball. 2010 International Conference on Digital Manufacturing & Automation, Digital Manufacturing and Automation (ICDMA), 2010 International Conference On, 2, 716–719. https://doi.org/10.1109/ICDMA.2010.153

[9] Gourmel, O., Pajot, A., Paulin, M., Barthe, L., & Poulin, P. (2010). Fitted BVH for Fast Raytracing of Metaballs. Computer Graphics Forum, 29(2), 281–288. https://doi.org/10.1111/j.1467-8659.2009.01597.x

[10] Junjun Pan, Shizeng Yan, Hong Qin, & Aimin Hao. (2018). Real-time dissection of organs via hybrid coupling of geometric metaballs and physics-centric mesh-free method. Visual Computer, 34(1), 105–116. https://doi.org/10.1007/s00371-016-1317-x

[11] Chang, J.-W. (1), Lei, S. I. E. (1), Chang, C.-F. (2,4), & Cheng, Y.-J. (3). (n.d.). Real-time rendering of splashing stream water. Proceedings - 3rd International Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIHMSP 2007, 1, 337–340. https://doi-org.pbidi.unam.mx:2443/10.1109/IIH-MSP.2007.255

[12] Zhang, P., Dong, Y., Galindo-Torres, S. A., Scheuermann, A., & Li, L. (2021). Metaball based discrete element method for general shaped particles with round features. Computational Mechanics, 67(4), 1243–1254. https://doi-org.pbidi.unam.mx:2443/10.1007/s00466-021-02001-9

[13] Abdellah, M., Foni, A., Zisis, E., Guerrero, N. R., Lapere, S., Coggan, J. S., Keller, D., Markram, H., & Schurmann, F. (2021). Metaball skinning of synthetic astroglial morphologies into realistic mesh models for visual analytics and in silico simulations. BIOINFORMATICS, 37, I426–I433. https://doi-org.pbidi.unam.mx:2443/10.1093/bioinformatics/btab280

[14] Thalmann, D., Jianhua Shen, & Chauvineau, E. (1996). Fast realistic human body deformations for animation and VR applications. Proceedings of CG International '96, Computer Graphics International, 1996. Proceedings, 166–174. https://doi-org.pbidi.unam.mx:2443/10.1109/CGI.1996.511798

[15] Bai, J. (1), Pan, J. (1,2), Yang, Y. (1), & Qin, H. (3). (n.d.). Novel metaballs-driven approach with dynamic constraints for character articulation. Science China Information Sciences, 61(9). https://doi-org.pbidi.unam.mx:2443/10.1007/s11432-018-9470-4

[16] Kanamori, Y., Szego, Z., & Nishita, T. (2008). GPU-based Fast Ray Casting for a Large Number of Metaballs. Computer Graphics Forum, 27(2), 351–360. https://doi-org.pbidi.unam.mx:2443/10.1111/j.1467-8659.2008.01132.x

[17] Shen, Jianhua & Thalmann, Daniel. (2002). Interactive Shape Design Using Metaballs and Splines.

[18] Parent, R. (2002). Computer Animation: Algorithms and Techniques (The Morgan Kaufmann Series in Computer Graphics) (1st ed.). Morgan Kaufmann.

[19] César Adrián Victoria Ramírez. 2021. Partículas Suavizadas Optimizada Mediante Octree Dinámico. México. [Tesis de doctorado no publicada].

[20] Soriano Valdez, David Arturo. 2021. Simulación libre de mallas por partículas suavizadas optimizado mediante octree dinámico en cuda. [Tesis de Doctorado, Universidad Nacional Autónoma de México]. https://ru.dgb.unam.mx/handle/DGB_UNAM/TES01000813657

[21] Gastelum-Strozzi, Alfonso & Delmas, Patrice & Marquez, Jorge & Gimel'farb, Georgy. (2011). Smoothing particle hydrodynamics parallel implementation for numerical modelling of solid-fluid interactions. 10.13140/RG.2.2.13937.22883.

[22] Valdez D.A.S. et al. (2020) CUDA Implementation of a Point Cloud Shape Descriptor Method for Archaeological Studies. In: Blanc-Talon J., Delmas P., Philips W., Popescu D., Scheunders P. (eds) Advanced Concepts for Intelligent Vision Systems. ACIVS 2020. Lecture Notes in Computer Science, vol 12002. Springer, Cham. https://doi.org/10.1007/978-3-030-40605-9_39

[23] Osher, S., & Fedkiw, R. (2003). Level set methods and dynamic implicit surfaces (1st ed.). Springer.

[24] Wenger, R. (2013). Isosurfaces: geometry, topology, and algorithms. CRC Press.

[25] Morton, G.M. (1966) A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing. IBM Ltd., Ottawa.

[26] Processors. Intel. from https://www.intel.la/content/www/xl/es/products/details/processors.html

[27] Procesadores AMD. from https://www.amd.com/es/processors

[28] Compare GeForce Graphics Cards. NVIDIA. from https://www.nvidia.com/en-us/geforce/graphics-cards/compare/

[29] Radeon Rx Graphics AMD. from https://www.amd.com/es/graphics/radeon-rx-graphics

[30] NVIDIA TURING GPU ARCHITECTURE. NVIDIA. from https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

[31] Textures best practices for Unreal Engine. from https://developer.arm.com/documentation/102696/0100/Texture-filtering

[32] What is Anisotropic Filtering?. from https://www.intel.com/content/www/us/en/gaming/resources/what-is-anisotropic-filtering.html

[33] Texture Support ans Settings — Unreal Engine 4.27 Documentation. from https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/Textures/SupportAndSettings/

[34] NVIDIA. (May 20, 2021). CUDA Toolkit Documentation v11.3.1. Website. https://docs.nvidia.com

[35] Han, J., & Sharma, B. (2019). Learn Cuda Programming: A beginner's guide to Gpu programming and Parallel Computing with Cuda 10.x and C/C++. Packt Publishing Limited.

[36] Cook, S. (2013). CUDA programming: a developer's guide to parallel computing with GPUs. Morgan Kaufmann.

[37] Blender. blender.org - Home of the Blender project - Free and Open 3D Creation Software. [online] blender.org. Available at: https://www.blender.org.

[38] Autodesk.mx. 2022. Características de 3ds Max | Características de 2023, 2022 y 2021 | Autodesk. [online] Available at: https://www.autodesk.mx/products/3ds-max/overview?term=1-YEAR&tab=subscription.

[39] Autodesk.mx. 2022. Software AutoCAD. [online] Available at: https://www.autodesk.mx/products/autocad.

[40] FreeCAD. 2022. FreeCAD: Your own 3D parametric modeler [online] Available at: https://www.freecadweb.org/.

[41] Gordon V. S. & Clevenger J. (2019). Computer graphics programming in opengl with c++. Mercury Learning and Information.

[42] Liu, S., Zhang, M., Kadam, P., & Kuo, C.-C. J. (2021). 3D Point Cloud Analysis: Traditional, Deep Learning, and Explainable Machine Learning Methods (1st ed. 2021.). Springer International Publishing.

[43] Guha, S. (2019). Computer Graphics Through OpenGL® From theory to experiments.

[44] Lengyel, E. (2012). Mathematics for 3D game programming and computer graphics. Boston, Mass: Cengage Learning.

[45] Foley, J. D., van Dam, A., Feiner, S.,, Hughes, J. (1990). Computer Graphics: Principles and Practice. Reading, MA: Addison-Wesley. ISBN: 978-0-201-12110-0

[46] Singh, J. M., & Narayanan, P. J. (2010). Real-Time Ray Tracing of Implicit Surfaces on the GPU. IEEE Transactions on Visualization and Computer Graphics, Visualization and Computer Graphics, IEEE Transactions on, IEEE Trans. Visual. Comput. Graphics, 16(2), 261–272. https://doi-org.pbidi.unam.mx:2443/10.1109/TVCG.2009.41

[47] McManamon, P. F. (2019). LiDAR technologies and systems. SPIE Press.

[48] Linder, W. (2016). Digital photogrammetry: a practical course (4th edition). Springer.

[49] Yang, X. (1), Dawant, B. M. (1,2,4), Clements, L. W. (2), Luo, M. (2), Narasimhan, S. (2), Miga, M. I. (2,3,4), & Thompson, R. C. (3). (2017). Stereovision-based integrated system for point cloud reconstruction and simulated brain shift validation. Journal of Medical Imaging, 4(3). https://doi-org.pbidi.unam.mx:2443/10.1117/1.JMI.4.3.035002

[50] Mingyou Chen, Yunchao Tang, Xiangjun Zou, Kuangyu Huang, Lijuan Li, & Yuxin He. (2019). High-accuracy multi-camera reconstruction enhanced by adaptive point cloud correction algorithm. Optics and Lasers in Engineering, 122, 170–183. https://doi-org.pbidi.unam.mx:2443/10.1016/j.optlaseng.2019.06.011

[51] Fu, Y., Jia, T., Song, Z., & Peng, B. (2018). Commodity 3D display based on point cloud reconstruction. 2018 Chinese Control And Decision Conference (CCDC), Chinese Control And Decision Conference (CCDC), 2018, 5617–5622. https://doi-org.pbidi.unam.mx:2443/10.1109/CCDC.2018.8408111

[52] Park, S. I., & Lim, S.-J. (2014). Template-Based Reconstruction of Surface Mesh Animation from Point Cloud Animation. ETRI Journal, 36(6), 1008–1015.

[53] Yu, S., Sun, S., Yan, W., Liu, G., & Li, X. (2022). A Method Based on Curvature and Hierarchical Strategy for Dynamic Point Cloud Compression in Augmented and Virtual Reality System. Sensors (Basel, Switzerland), 22(3). https://doi-org.pbidi.unam.mx:2443/10.3390/s22031262

[54] Zhu, L., Xiang, Y., & Song, A. (2022). Visible Patches for Haptic Rendering of Point Clouds. IEEE Transactions on Haptics, 15(3), 497–507. https://doi-org.pbidi.unam.mx:2443/10.1109/TOH.2022.3165119

[55] Franzluebbers, A., Li, C., Paterson, A., & Johnsen, K. (2022). Virtual Reality Point Cloud Annotation. 2022 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), 2022 IEEE Conference on, VRW, 886–887. https://doi-org.pbidi.unam.mx:2443/10.1109/VRW55335.2022.00294

[56] Ji, S., Li, W., Zhang, Z., Zhou, S., Cai, Z., & Tian, J. (2021). Robotic arm grasping through 3D point clouds recognition. 2021 IEEE International Conference on Real-Time Computing and Robotics (RCAR), Real-Time Computing and Robotics (RCAR), 2021 IEEE International Conference On, 347–352. https://doi-org.pbidi.unam.mx:2443/10.1109/RCAR52367.2021.9517680

[57] Shih, Y., Liao, W., Lin, W., Wong, S., & Wang, C. (2022). Reconstruction and Synthesis of Lidar Point Clouds of Spray. IEEE Robotics and Automation Letters, Robotics and Automation Letters, IEEE, IEEE Robot. Autom. Lett, 7(2), 3765–3772. https://doi-org.pbidi.unam.mx:2443/10.1109/LRA.2022.3148443

[58] Kleinberg, J., Tardos, & E. (2006). Algorithm Design. Addison Wesley.

[59] Cormen T. H. (2001). Introduction to algorithms (2nd ed.). MIT Press.