Optimización del algoritmo Nvdiffrec mediante el uso de métodos para retopología 3D

Para obtener el grado de Maestro en Ciencias de la Computación
Presenta Arturo Tapia Rodríguez

Director de tesis: Dr. Guillermo De Ita Luna.

Co-director: Dr. Luis Carlos Altamirano Robles

Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias Computacionales
Puebla, México
Noviembre 2023

"Veni, vidi, vici"
-Julio César

Agradecimientos

Quiero agradecer a Dios por haberme permitido llegar hasta este punto en mi carrera profesional y por todos los logros a lo largo de mi vida.

A mis padres y a mi hermana por haberme ayudado siempre de manera incondicional y por haber podido contar con ellos durante todo el periodo de la maestría.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por su apoyo a través de la beca de maestría.

De manera especial, quiero agradecer mi amigo Arturo González y su familia por haberme ayudado durante todo mi periodo de estancia en Puebla, ya que su apoyo fue fundamental para poder completar este logro.

Igualmente, agradezco a mis amigos Israel Ortíz y Juan Diego Torres por su apoyo durante las constantes y variadas vicisitudes que acontecieron durante la elaboración de esta tésis.

Y por sobre todo, a la Benemérita Universidad Autónoma de Puebla y a su profesorado por su constante retroalimentación, excelente asesoramiento y aprendizajes brindados.

Resumen

En los últimos años se han propuesto diferentes enfoques para resolver el problema de la reconstrucción 3D. Sin embargo, Nvdiffrec [9] ha demostrado ser particularmente preciso de acuerdo a los estándares de la industria. Esto se debe principalmente a que el algoritmo genera las texturas que se utilizan para el desarrollo de videojuegos de forma predeterminada. Aun así, algunas partes del proceso de reconstrucción 3D se pueden optimizar para obtener más precisión. También es importante tener en cuenta que los modelos 3D producidos por Nvdiffrec no son fáciles de usar para editar, principalmente porque genera sólo mallas trianguladas. En comparación con nuestro algoritmo optimizado, tenemos la intención de generar una malla que sea más fácil de editar para artistas 3D y también para producir una salida más precisa tanto para la malla como para todas las texturas. Intentamos lograr esto mediante la implementación adecuada de algoritmos de retopología 3D, con la intención de evitar la triangulación de la malla, y de ésta manera generar un resultado más preciso antes de proceder a la formación de las texturas.

Palabras clave: Reconstrucción 3D, Retopología, Redes neuronales.

Contenido

Agradecimientos II

Resumen III

Lista de símbolos VI

1. Introducción 1

2. Trabajo relacionado 2
2.1. Reconstrucción 3D . . . 2
2.1.1. Retopología . . . 3

3. Fundamento teórico 4
3.1. Renderizado Basado en Físicas(PBR) . . . 4
3.2. Topología en el modelado 3D . . . 5
3.3. Redes neuronales . . . 8
3.3.1. Redes neuronales convolucionales . . . 8
3.3.2. ResNet 50 . . . 9
3.3.3. Desvanecimiento del Gradiente . . . 10
3.3.4. Graph Convolutional Network . . . 10
3.3.5. Red Neuronal Convolucional Punto-Vóxel . . . 11

4. NVDiffrec 13
4.1. Optimización . . . 13
4.2. Aprendizaje de topología . . . 13
4.2.1. Deep Marching Tetrahedrons . . . 13

5. Octree remeshing 15

6. El estándar de Turbosquid para evaluar modelos 3D 17
6.0.1. Estándares en la inspección del modelo 3D . . . 18

7. Implementación 20

8. Resultados 26
8.1. Precisión del renderizado inverso . . . 26

8.2. Calidad del modelo reconstruido 27
8.3. Análisis de complejidad en tiempo 28
8.3.1. Redes neuronales 28
8.3.2. Octree 30
8.3.3. Ray tracing 32

9. Conclusión y trabajo a futuro **34**
9.1. Trabajo a futuro 34

A. Anexo: Imágenes de la retopología **35**

B. Anexo: Imágenes de los resultados **36**

C. Anexo: Ejemplo de loop-cutting **39**

Bibliografía **42**

Lista de símbolos

Subíndices

Subíndice	Término
O	NVDiffrec Optimizado
N	NVDiffrec
T	Valor de verdad (Ground Truth)

Abreviaturas

Abreviatura	Término
1.ANN	Redes Neuronales Artificiales
2.CNN	Redes Neuronales convolucionales
3. GCN	Red Convolucional para Gráfos
4. $PVCNN$	Red Neuronal Convolucional Pixél-Vóxel
5.RNN	Redes Neuronales recurrentes
6.$ResNet$	Redes Neuronales Residuales
7.$TSDF$	Función truncada de distancia con signo
8.MSE	Error cuadrático medio

1. Introducción

La creación de modelos 3D siempre ha sido una tarea realizada por artistas 3D profesionales, esto se debe principalmente a las complicaciones que pueden surgir durante el proceso de creación. Sin embargo, dentro de la creación 3D a veces se requiere recrear objetos que ya existen en el mundo real, para esto ya se han desarrollado muchas herramientas. Desafortunadamente, la mayoría de estas herramientas brindan resultados que no son óptimos para un uso frecuente (como renderizar animaciones o desarrollar juegos).

Esta es la razón por la que NVdiffrec [9] es un gran avance en el campo de los algoritmos de reconstrucción 3D. A pesar de esto, todavía hay muchas mejoras que se pueden hacer a este algoritmo, por lo que proponemos comenzar con la generación de una mejor topología que le permita al artista 3D trabajar con la malla inmediatamente después de generar la salida.

Para el proceso de mallado, se decidió utilizar el algoritmo de retopología octree debido a su rapidez de ejecución.

Ha habido muchos trabajos para mejorar el rendimiento de la reconstrucción 3D, algunos incluso hacen uso de redes neuronales para resolver este problema. Sin embargo, el método clásico para manejar este problema es la reconstrucción 3D multi-vista donde la principal herramienta utilizada para reconstruir es la correspondencia entre imágenes.

Nvdiffrec [9] aborda este problema como un problema de renderizado inverso en el que dada una imagen (o renderizado) queremos encontrar los datos necesarios para generar dicho renderizado (el modelo 3D y sus mapas de textura correspondientes). Aunque esto produce resultados impresionantes, tiene el problema de que, dado que ahora los mapas y el modelo están estrechamente vinculados, el error en el proceso de reconstrucción de la malla o en la generación de texturas se propaga a lo largo de todos los demás mapas de texturas.

Debido a esto, creemos que tener una mejor topología reducirá el error en todos los mapas y en la malla 3D.

2. Trabajo relacionado

Si bien hay muchos trabajos en la tarea de reconstrucción 3D, no hay tantos orientados a la industria de la animación y el desarrollo de juegos. Esto se debe principalmente a que al ser un problema de inversión, no hay garantía de que la solución obtenida sea única o estable, además de necesitar de grandes cantidades de memoria gráfoca asi como velocidad de procesamiento. Por estas razones, hasta hace algunos años era inviable el proporcionar una solución precisa. Debido a esto, decidimos mencionar algunos de los trabajos más recientes relacionados con la reconstrucción 3D y los algoritmos de mallado, incluso si están destinados a aplicaciones de propósito general.

2.1. Reconstrucción 3D

Como mencionamos antes, el problema de la reconstrucción 3D se ha abordado de muchas maneras diferentes. El enfoque clásico es la reconstrucción 3D multi-vista, generalmente hablando, se genera una nube de puntos para crear los vértices de la malla. Luego, esta nube de puntos se conecta para crear una malla. Debido a esto, los métodos para encontrar la correspondencia son variados, uno de ellos es la selección de vista Pixelwise para imágenes estéreo multivista no estructuradas [11], que utiliza colecciones de imágenes no estructuradas para recuperar la profundidad y la información normal.

Sin embargo, existen otros enfoques para este problema, como el uso de representaciones volumétricas en lugar de una malla para la reconstrucción 3D. NeRF [2, 4, 8, 14] es uno de los principales algoritmos que adoptan este enfoque. Hace uso de campos de radiación que representan cada imagen como un vector 5D para reconstruir la escena completa basándose únicamente en sus imágenes.

También hay muchos algoritmos para la representación explícita de superficies, lo que significa que dan una malla como salida. Uno de los ejemplos más recientes de estos algoritmos es el renderizador diferenciable híbrido [1], que tiene en cuenta los modelos de materiales que varían espacialmente y la iluminación ambiental para crear materiales realistas.

2.1.1. Retopología

Dentro del modelado 3D, siempre ha sido de interés lograr la representación topológica más adecuada de la figura que se está modelando. Por esta razón, ha habido muchos trabajos relacionados con el aprendizaje geométrico y la retopología.

Uno de los enfoques para el aprendizaje geométrico 3D es obtener un conjunto desordenado de proyecciones de la figura 3D y usar esas proyecciones para crear una representación 3D diferente de la misma. Esta metodología se conoce como enfoque Multi-vista para el reconocimiento de formas en 3D utilizando Redes Neuronales Convolucionales. También existen las CNN dinámicas de gráfos (DGCNN) para el aprendizaje en nubes de puntos [16] utiliza este enfoque para su proceso de retopología. Aunque la principal ventaja de este enfoque es el hecho de que podemos usar redes neuronales convolucionales 2D para resolver el problema. De igual manera, es posible usar CNN tridimensionales y procesar directamente los datos como una nube de puntos volumétricos. La reconstrucción de formas 3D texturizadas de alta calidad con redes convolucionales en cascada [6] es un ejemplo de este enfoque.

Con la mejora constante en el Aprendizaje Profundo, también ha habido muchos trabajos relacionados con el Aprendizaje Profundo en mallas. Sin embargo, es importante mencionar que las mallas tienen 3 propiedades geométricas diferentes de las que las computadoras pueden aprender: vértices, aristas y caras. dependiendo de la propiedad elegida, el algoritmo cambiará en consecuencia.

En los algoritmos de aprendizaje profundo basados en vértices, tenemos sistemas como las redes neuronales convolucionales duales de malla [7] que hacen uso de la vecindad de vértices locales para crear una representación diferente de la malla.

Cuando el proceso de aprendizaje profundo se basa en aristas, se requiere una malla triangulada. Esto se debe principalmente a que aprovecha la propiedad de que cada arista es adyacente a dos caras y cuatro aristas para poder realizar una convolución 2D. PD-MeshNet [7] usa esta propiedad para crear una convolución invariable de orden y luego generar la nueva malla como resultado.

Finalmente, tenemos Aprendizaje Profundo basado en las caras de la malla. Este enfoque consiste en obtener la mayor cantidad de información posible de las caras vecinas. La síntesis de texturas geométricas profundas hace uso de este enfoque para realizar una convolución de 3 caras.

3. Fundamento teórico

3.1. Renderizado Basado en Físicas(PBR)

Muchos programas de edición 3D tienen su implementación para el renderizado PBR (Renderizador por Ciclos para Blender, IRay para herramientas de Autodesk, etc.). PBR utiliza muchos mapas de textura para incorporar diferentes propiedades del material. Los más importantes (y comúnmente utilizados) de estos mapas de textura son los mapas difuso, metálico y de rugosidad y también el mapa normal. Enseguida explicaremos la función de cada uno de estos mapas de textura.

El mapa difuso tiene los datos de color del material sin verse afectado por ninguna fuente de luz. También se le llama mapa de color base.

El mapa metálico como su nombre lo indica contiene información sobre que tan metálico es el material. A veces puede usar la misma imagen que utiliza el mapa de rugosidad. Este es el caso de Unity, donde la aspereza está incrustada en el canal alfa de la imagen.

El mapa de normales contiene detalles geométricos sobre la malla. Este mapa es extremadamente útil cuando se utilizan materiales muy detallados. Principalmente porque no necesita cambiar la topología de la malla para dar la ilusión de detalle (ver Figura **3-1**).

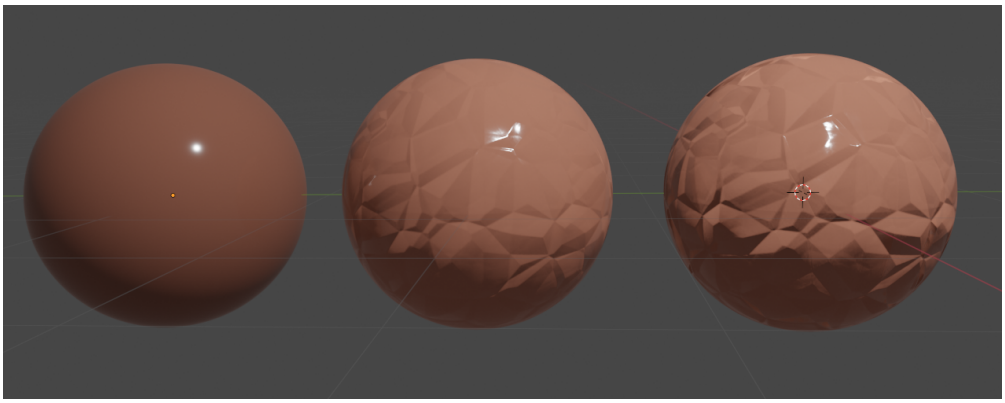

Figura 3-1.: Efecto de los mapas normales (0 intensidad en la izquierda, 0,5 en el medio y 1 en la derecha).

La combinación de estos mapas de texturas nos permite crear modelos 3D fotorrealistas (consulte la Figura **3-2**) que son compatibles con diferentes herramientas de modelado. Debido a esto, se volvió importante automatizar el proceso de diseño de materiales para la industria del modelado 3D. Substance Painter es una de las herramientas más utilizadas que consiguen esto con resultados fotorrealistas y compatibilidad entre todas las herramientas de modelado 3D.

Figura 3-2.: Ejemplo de un objeto 3D fotorrealista.

Aquí podemos ver los resultados del uso de todos los mapas de textura disponibles para lograr renderizados más realistas. Esto ha sido particularmente útil en el campo de la arquitectura y el diseño de interiores, donde es importante una representación precisa del espacio y la iluminación.

3.2. Topología en el modelado 3D

En el modelado 3D, la topología se refiere a la estructura de la malla y cómo se distribuyen los bordes dentro de ella. Esto significa que incluso si dos mallas se ven similares, es posible que tengan una topología completamente diferente (consulte la Figura **3-3**).

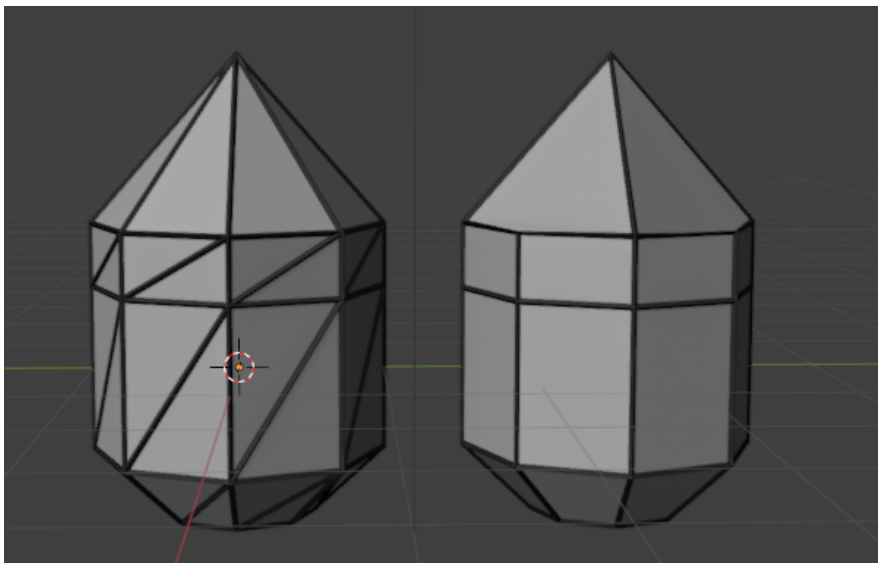

Figura 3-3.: Comparación de diferencia de topología

Como podemos ver, ambas figuras representan casi la misma forma. Sin embargo, la topología (resaltada en negro) es muy diferente. Además, es importante saber que, en términos generales, se prefiere tener cuadriláteros sobre triángulos.

Estas son algunas de las principales razones por las que los profesionales del 3D trabajan principalmente con cuadriláteros:

- La ecuación de interpolación de sombreado suave tiene comportamientos inesperados cuando se trabaja con triángulos.
- Los modificadores de subdivisión funcionan mejor con cuadriláteros.
- Loop-Cutting (véase Apéndice C) solo funciona en cuadriláteros. Esto sucede porque los cuadriláteros tienen 2 direcciones, los triángulos no tienen ninguna.

El uso mayoritario de cuadriláteros también es usado para mantener un buen flujo de las aristas es algo muy importante para el proceso de elaboración de modelos 3D. Esto se debe a la carencia de dirección en los triángulos, ya que al no tener dirección estos rompen el flujo de aristas.

Aunque las mallas triangulares de sombreado plano funcionan correctamente, como podemos ver en la Figura **3-4**, el sombreado suave en mallas trianguladas produce resultados no deseados.

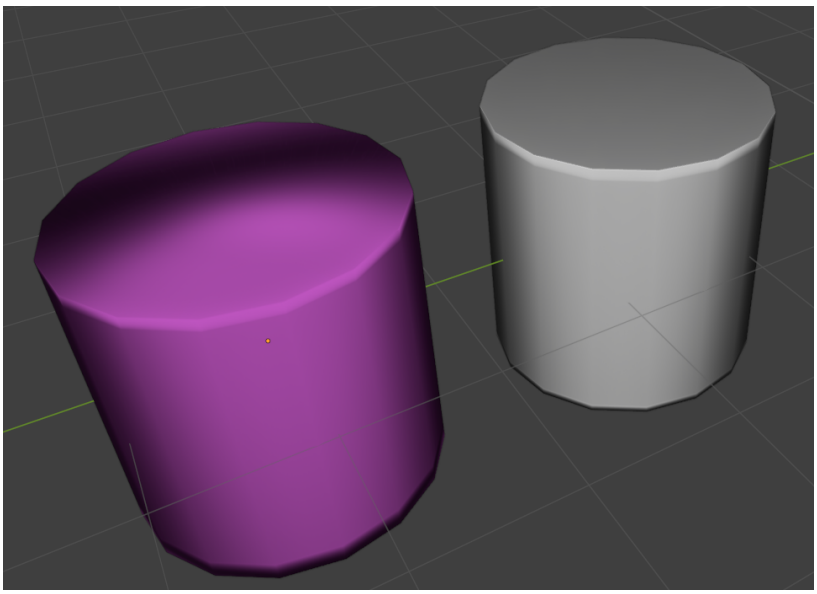

Figura 3-4.: Comparación de sombreado suave de diferencia de topología

Como podemos ver en **3-4**, el sombreado suave tiene resultados de iluminación inesperados cuando usamos mallas trianguladas como la que está en púrpura. También es importante notar que estos problemas tienden a escalar a medida que aplicamos más modificadores sobre mallas con una topología incorrecta.

Sin embargo, con solo abstenerse de utilizar triángulos no es suficiente para asegurar una buena topología. Por esta razón, existe un estándar topológico para la representación de figuras básicas (como planos, cubos, cilindros, esféras y conos).

Este estándar tiene como finalidad principalmente el garantizar que el proceso de subdivisión sea correcto. No obstante, la mayor parte de los modelos 3D diseñados por profesionales están lejos de ser las figuras básicas. Por está razón es importante el desarrollo de una heurística que genere buena topología.

El problema principal al aplicar el modificador de subdivisión a mallas trianguladas es que al efectuarla en triángulos genera cuadrados. Esto significa que a medida que aumentamos los niveles de subdivisión, el comportamiento de la malla cambia. Debido a esto, podemos tener problemas de sombreado (consulte la figura **3-4**) y una conservación deficiente de la forma después de la subdivisión (consulte la figura **3-5**).

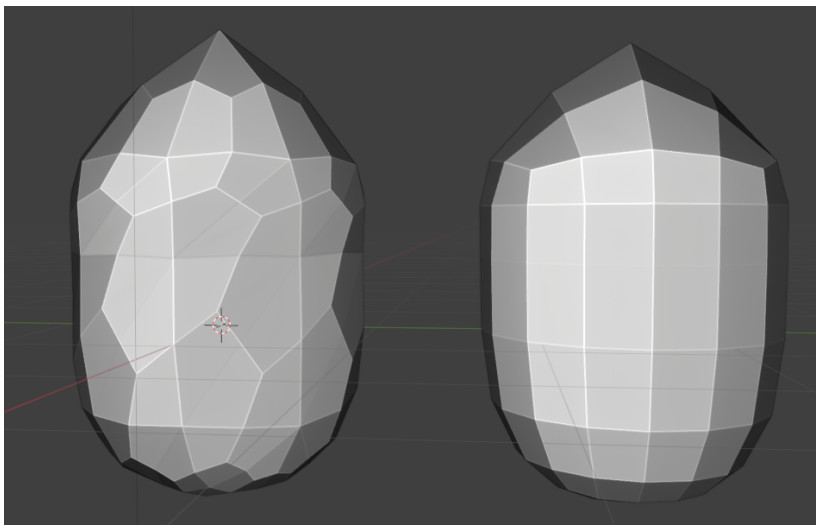

Figura 3-5.: Topología después de la subdivisión

Como podemos ver, la figura de la izquierda comienza presentando algunos artefactos debido a la subdivisión del triángulo. Además, se nota que ya no tenemos bordes rectos en los lados de la forma (en comparación con la subdivisión basada en cuadriláteros).

Por todas estas razones, es preferible tener una malla compuesta mayoritariamente por cuadriláteros. AdemÃ¡s de que facilita el proceso de edición y generar una malla triangulada a partir de una malla cuádruple ya es un proceso automatizado.

3.3. Redes neuronales

Las redes neuronales artificiales (ANNs) son una rama del aprendizaje automático. Las redes neuronales pueden considerarse una herramienta bÃ¡sica para el aprendizaje profundo. Las redes neuronales consisten de una capa de entrada, una o más capas ocultas, y una capa de salida. La primer propuesta de un sistema parecido a lo que son la redes neuronales hoy en día fue el perceptrón de Rosenblatt [10].

3.3.1. Redes neuronales convolucionales

Las redes neuronales convolucionales han sido la mejor opción para el reconocimiento de patrones en imágenes dentro de la rama de aprendizaje profundo.

Las redes neuronales convolucionales reciben su nombre debido a que realizan una operación conocida como convolución a través de sus capas. Las CNNs son una versión regulada de los

perceptrones multicapa.

Las CNNs consisten de una capa de entrada, varias capas ocultas y una capa de salida. En estas capaz se realiza el producto punto con el kernel de la convolución y la capa de entrada. El kernel nos sirve como filtro para el reconocimiento de características en la matriz, por lo general se usa como kernel una matriz de 3×3, esto se debe a que al anidar convoluciones, el campo receptivo de cada kernel aumenta. Por ejemplo, al realizar la convolución de una matriz de 5×5 con un kernel de 3×3 se obtiene como resultado una matriz de 3×3 la cuál tiene como campo perceptivo la mtriz de 5×5 con la cual se generó.

La razón por la cual este tipo de redes dominan en el reconocimiento de patrones en imágenes es debido a que una vez que la imagen es procesada por la primera capa, esta se abstrae en sus rasgos característicos. Este proceso de abstracción es conocido como pooling (Figura **3-6**).

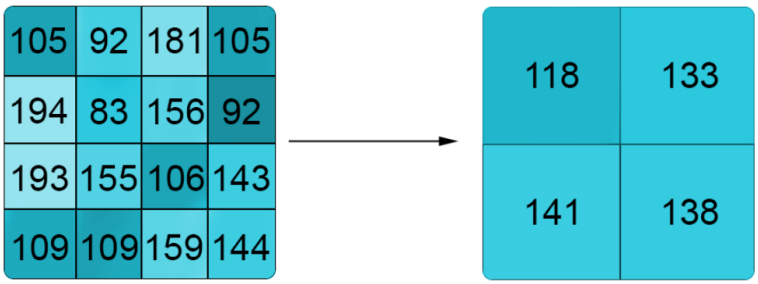

Figura 3-6.: Ejemplo de pooling en redes neuronales convolucionales

Las redes neuronales convolucionales se especializan en el reconocimiento de patrones para entradas en forma matricial. Esto se debe a la naturaleza de la operación de convolución, la cual realiza un barrido por toda la matriz para generar la entrada a la siguiente capa.

3.3.2. ResNet 50

Las redes tipo ResNet son conocidas como redes neuronales residuales, estas son bastante similares en arquitectura a las redes neuronales convolucionales, con la diferencia de que estas tienen una retro alimentación hacia adelante que ayuda a la red a no tener que recordar el vector de entrada (Figura **3-7**).

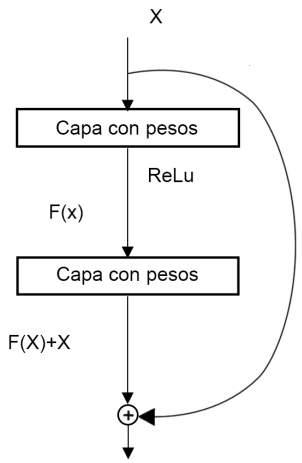

Figura 3-7.: Ejemplo de una sola capa de las redes tipo ResNet

3.3.3. Desvanecimiento del Gradiente

EL principal problema mitigado por las redes tipo ResNet es el del vanishing gradient. Este problema consiste en el hecho de que al retropropagar el gradiente en redes neuronales recurrentes (RNN) cuando el gradiente pasa a través de muchas capas este se vuelve extremadamente pequeño.

Debido a esto el entrenar redes neuronales residuales se vuelve muchísimo mas eficiente en comparación a las RNN tradicionales. Ya que al tener una conexión directa hacia el siguiente grupo de capas el gradiente crece considerablemente.

Entre las redes tipo ResNet la Resnet50 ha sido la dominante en tareas de reconocimiento de imágenes. Esta red esta caracterizada por tener 50 capas como la descrita en la figura **3-7**.

3.3.4. Graph Convolutional Network

Las GCN [5] son redes especializadas en segmentación semántica. En particular en el caso de [5] se utiliza un conjunto de Splines para describir el contorno del objeto sin la necesidad de una gran cantidad de puntos.

Para inicializar la red neuronal se asume que el contorno puede ser descrito de una manera aceptable con N puntos de control.
Sea $cp_i = [x_i, y_i]^T$ la ubicación del i-ésimo punto de control y $V = \{cp_0, cp_1, ..., cp_{N-1}\}$ el conjunto de puntos de control. Se define un grafo con el conjunto de nodos V y el conjunto de aristas E. Entonces la propagación hacia adelante se realiza de la manera siguiente:

$$f_i^{l+1} = w_0^l f_i^l + \sum_{cp_j \in N(cp_i)} w_1^l f_j^l \tag{3-1}$$

Ya que las GCN también son un tipo de red residual (ResNet) tienen una función especial para agregar el residuo durante la propagación hacia adelante.

$$r_i^l = ReLU(w_0^l f_i^l + \sum_{cp_j \in N(cp_i)} w_1^l f_j^l) \tag{3-2}$$

$$r_i^{l+1} = \tilde{w}_0^l r_i^l + \sum_{cp_j \in N(cp_i)} \tilde{w}_1^l r_j^l \tag{3-3}$$

$$f_i^{l+1} = ReLU(r_i^{l+1} + f_i^l) \tag{3-4}$$

Donde r_i^l es el residuo conectado a f_i^{l+1}. Debido a esta conexión, al igual que en las redes tipo ResNet tenemos un mejor desempeño en el tiempo de entrenamiento.

3.3.5. Red Neuronal Convolucional Punto-Vóxel

Las PVCNN (por sus siglas en inglés) son redes neuronales convolucionales dedicadas al proceso de reconstrucción 3D. Son un híbrido de las PCNN y las VCNN. Las principales ventajas de esta red por encima de las mencionadas previamente es mejorar el desempeño tanto en tiempo como en memoria.

Estas redes funcionan representando la información 3D como $x = \{x_k\} = \{(p_k, f_k)\}$ donde p_k son las coordenadas en 3D del k-ésimo voxel.

Debido a la representación utilizada para los datos, se puede utilizar la misma fórmula para la convolución tanto de los puntos como de los vóxeles.

$$y_k = \sum_{x_i \in N(x_k)} K(x_k, x_i) \times F(x_i). \tag{3-5}$$

Donde x_k es el centro de cada vóxel, $N(x_i)$ el centro de los vóxeles vecinos, K el kérnel y F las características de los vecinos.

Además, como parte el proceso de inicialización, se normaliza la nube de puntos para asegurar que las coordenadas (x, y, z) de nuestro objeto estén dentro del rango $[0, 1]$. Para asegurar

esto, se calcula el centro de masa de la figura, y se toma este centro como el origen $(0, 0, 0)$. Después se calcula el valor mas lejano del conjunto de puntos x, y y z de manera independiente para evitar deformaciones y finalmente se hace $\hat{p_k} = p_k / max(||p_k||)$.

Finalmente, se transforma la nube de puntos $\hat{p_k}$ en un conjunto de vóxeles tomando el promedio de las características de cada punto que este dentro del vóxel. Este promedio se realiza de la manera siguiente:

$$V_{u,v,w,c} = \frac{1}{N_{u,v,w}} \sum_{k=1}^{n} \mathbb{I}[\lfloor \hat{x}_k \times r \rfloor = u, \lfloor \hat{y}_k \times r \rfloor = v, \lfloor \hat{z}_k \times r \rfloor = w] \times f_{k,c} \tag{3-6}$$

Donde $\mathbb{I}[\cdot]$ nos indica si la coordenada de $\hat{p}_k$ pertenece o no al grid de vóxeles (en formato binario).

4. NVDiffrec

4.1. Optimización

En general, la optimización se calcula mediante la diferencia entre el renderizado del objeto creado por NVDiffrec I_ϕ y el original I_{ref}. Esta función de optimización por Nvidia es realizado como sigue:

$$\mathbf{argmin}_\phi \mathbb{E}_c[L(I_\phi(c), I_{ref}(c))] \tag{4-1}$$

Donde $\mathbf{argmin}_\phi$ es el valor mínimo del argumento, $\mathbb{E}_c$ la función de riesgo empírico, I_{ref} es la imagen de referencia ϕ es el conjunto de parámetros a optimizar c es la pose de la cámara de la imagen a renderizar L la función de pérdida. A su vez la función de pérdida se define como sigue:

$$L = L_{image} + L_{mask} + \Lambda L_{re} \tag{4-2}$$

Donde, L_{image} es la pérdida en el espacio de la imagen (normalizada), L_{mask} es la pérdida de la imagen al cuadrado y ΛL_{re} que es una función regularizadora para mejorar la geometría. Finalmente, para la optimización se usa el optimizador Adam con el gradiente $\partial L/\partial \phi$ garantizando así que nos acerquemos al mínimo de pérdida posible ajustando el set de parámetros ϕ.

4.2. Aprendizaje de topología

Para el aprendizaje de la topología se realiza un proceso de regresión usando la función SDF mediante el método de los Deep Marching Tetrahedrons (DMTet). Este método es un equivalente al método conocido como Marching Cubes, con la exepción de que el primero nos dará siempre una malla triangulada.

4.2.1. Deep Marching Tetrahedrons

El algoritmo de los DMTet es un método de reconstrucción 3D que usa supervisión en 2D para el proceso de generación de la malla.

Este proceso consiste en generar una malla deformable la cual consiste únicamente de tetraedros. Una vez creada la malla se procede a deformarla usando el renderizado de la figura como método de supervisión en 2D.

El algoritmo DMTet es similar al algoritmo Marching Cubes con la única diferencia de que DMTet utiliza tetraedros en vez de cubos. Esto es importante ya que no existe forma de cortar un tetraedro sin que el corte tenga forma de triangulo. Sin embargo, para el caso de MC (Marching Cubes) los cortes pueden tener distintas formas como resultado.

El hecho de que el corte a los tetraedros siempre tenga forma triangular nos garantiza que se obtendrá una malla triangulada como resultado.

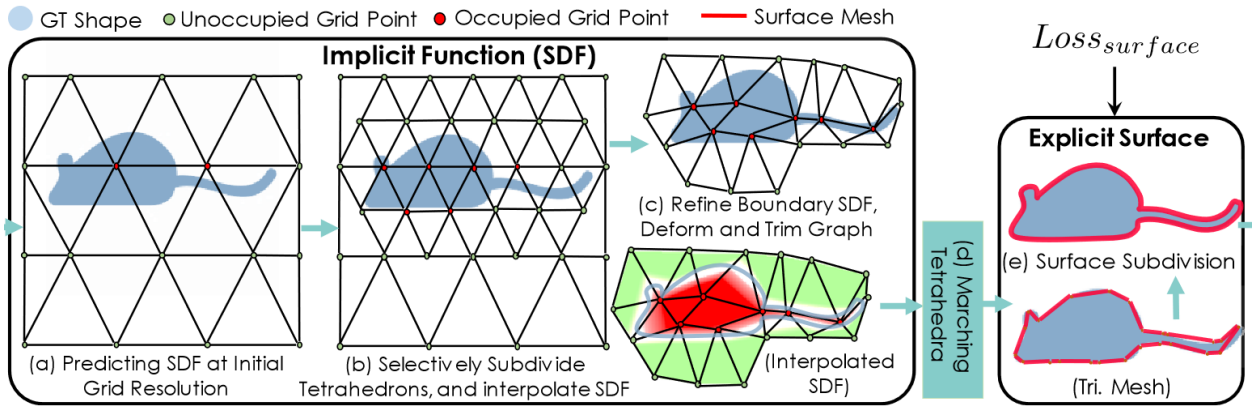

Figura 4-1.: Ejemplo del algoritmo DMTet en 2D (tomado de [13])

El algoritmo DMTet hace uso de un tipo de red convolucional conocida como PVCNN (Point-Voxel Convolutional Neural Network) para generar la entrada del algoritmo como un conjunto de características del volumen 3D.

Una vez obtenido este conjunto de características se realiza una primera estimación de la función de distancia con signo (SDF) con el fin de generar una primera representación implícita de la malla.

Después se utiliza esta estimación inicial para definir cuales tetraedros se subdividirán. Esta predicción es realizada mediante un Perceptrón Multicapa. Para realizar esta predicción, primero se identifica cuáles tetraedros están tocando la superficie de la figura.

Finalmente, mediante una GCN (Graph Convolutional Network) se calcula hacia donde se tiene que deformar la malla de tetraedros para encajar mejor con la figura original.

5. Octree remeshing

Los octree son un tipo de árbol en el cual cada nodo tiene exactamente 8 hijos. debido a que un cubo tiene exactamente 8 vértices, los octrees son mayormente usados para subdividir figuras tridimensionales.

Para hacer remeshing con octrees se requiere de dos cosas. El objeto original al que se le realizará el proceso y la profundidad que tendrá el octree.

Después para decidir como se subdividirá el objeto, se divide en 3 casos distintos:

- Los 8 vértices se encuentran dentro de la figura.
- Algunos vértices se encuentran adentro y algunos afuera.
- Los 8 vértices se encuentran afuera

Dados estos 3 casos el algoritmo funciona de la manera siguiente:

1. Se genera un cubo que encierre a la figura que se va a someter al proceso de remeshing.
2. Luego este cubo se divide en 8 cubos
3. Se evalúa en que caso queda cada uno de los cubos.
4. Todos los cubos que tienen el caso 2 se subdividen
5. Se aumenta el contador de profundidad del árbol en 1. En caso de no haber llegado a la profundidad esperada ir al paso 2.
6. Se mueven los vértices de los cubos que estén en el caso dos lo mas cercano a la figura original mediante un proceso de interpolación.

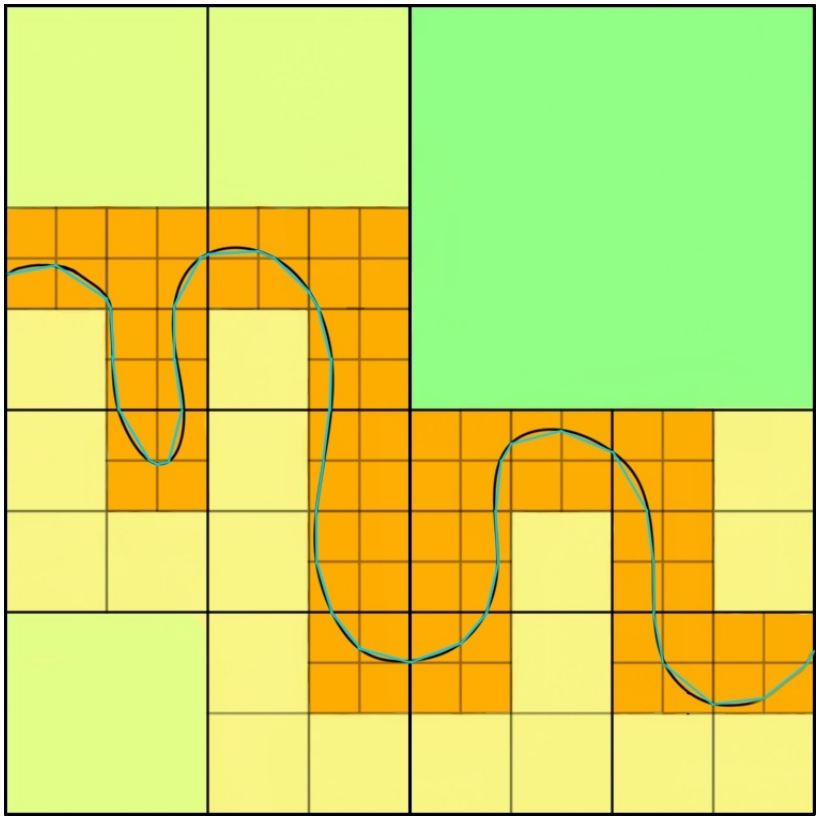

Figura 5-1.: Ejemplo del algoritmo octree en 2D (quadtree)

Uno de los beneficios mas importantes del método octree para retopología es que su complejidad es independiente a la cantidad de nodos con los que se forma la figura original y depende únicamente de la profundidad del árbol.

6. El estándar de Turbosquid para evaluar modelos 3D

Debido a la naturaleza creativa del trabajo de modelado 3D no existe a la fecha un estándar en el procedimiento adecuado para realizar un modelo 3D.

Sin embargo, el estándar más prolífero actualmente para este proceso es el estándar Checkmate Pro de Turbosquid. Algunas de las compañías que utilizan este estándar actualmente son:

- Ford
- General Motors
- Ethan Allen
- Google
- Saleen
- Walmart
- Zamboni

El estándar Checkmate pro se divide en tres secciones principales:

- Limitaciones de la certificación Checkmate Pro
- Estándares en la inspección del modelo 3D
- Estándares para presentación del producto

Las optimizaciones realizadas en este proyecto se enfocan en el segundo bloque de esta certificación.

Esto se debe a que el bloque uno es el conjunto de reglas y características mínimas del modelo para poder aplicar a la certificación.

Además, el último bloque cubre toda la parte de marketing y los estándares de presentación establecidos por Turbosquid. Debido a que ambas secciones son específicas de turbosquid nos enfocamos en la segunda sección en la cuál se especifican las características de un modelo de calidad.

6.0.1. Estándares en la inspección del modelo 3D

Para que un modelo apruebe esta sección, es necesario satisfacer los puntos que se mencionan a continuacíon:

1. Geometría
   - No tener vértices aislados
   - No tener vértices coincidentes (en la misma ubicación)
   - No tener caras coincidentes o coplanares
   - Normales apuntando hacia afuera
   - No tener objetos vacíos
   - La geometría no debe tener aberturas en las que se pueda ver a través
2. Topología
   - Cuadriláteros y triángulos solamente
   - Uso mayoritario de Cuadriláteros. Un uso máximo de 20 % de triángulos es considerado aceptable.
   - Buen flujo de aristas
   - El modelo debe ser subdividible: Esto significa que al subdividir el objeto no se pierde la forma original y se adquiere un mayor detalle.
3. Escala 1:1 con el objeto real
4. Posición y orientación
5. La matriz de transformación (posición, rotación y escala) debe estar en sus valores predeterminados.
6. Nombrado y organización
7. Texturas y materiales
   - Todos los objetos deben de tener un material.
   - Las texturas deben estar aplicadas donde sea necesario
8. UVs

- Las texturas no deben presentar un estiramiento notable.
- Las divisiones del mapa de UVs deben estar en las zonas menos visibles del modelo

9. Rigging
10. Animación

Las optimizaciones realizadas al trabajo de Nvidia estan enfocadas en su mayor parte a la sección de topología del estándar Checkmate Pro.

7. Implementación

Dado que el objetivo principal de esta optimización es hacer que el algoritmo sea más útil para la industria 3D, decidimos ejecutar ambos algoritmos en una computadora con especificaciones que coincidían con las utilizadas por los profesionales en el campo. Dado que PyTorch fue la herramienta utilizada para ambos algoritmos, la consideración más importante fue la GPU. La PC que usamos para compilar estos algoritmos tenía una Nvidia Geforce RTX 2070 con Max-Q Design y 8 gigabytes de memoria GPU.

De forma predeterminada, NVdiffrec es un algoritmo que utiliza mucha memoria. Por esta razón, aunque originalmente se usaron 8 epoch con fines de entrenamiento, decidimos limitar el algoritmo a 2 epoch. El algoritmo normalmente utiliza 8 epoch, pero esto haria que se necesitara un total de 24Gb de GPU lo cual excede por mucho las especificaciones de un equipo promedio (8 Gb de GPU). Sin embargo, debido a esto, nuestros resultados son más ruidosos que los originales (ver Figura **7-1**). Aún así, este ruido también se reduce a través del proceso de remallado.

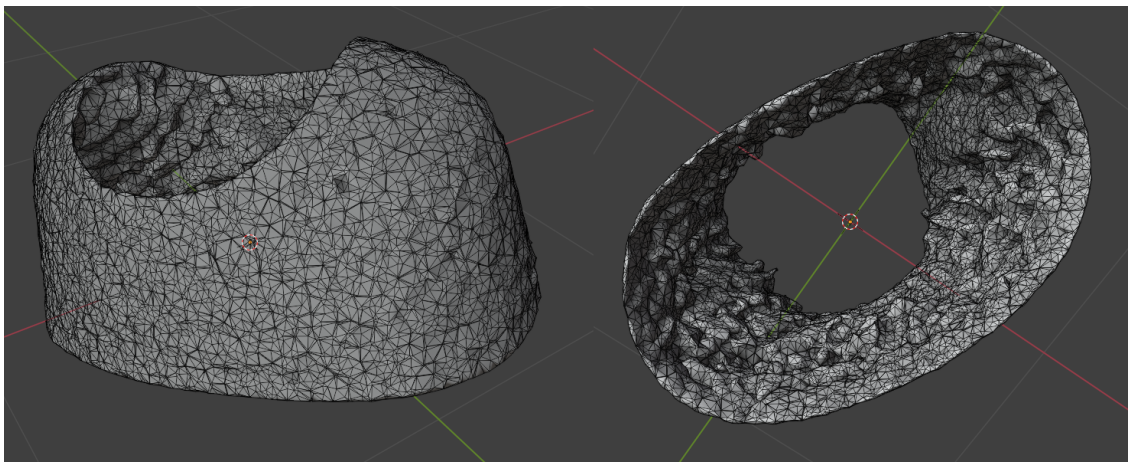

Figura 7-1.: Resultados ruidosos usando 2 epoch

Para la implementación, primero se obtiene una malla subdividida del algoritmo de reconstrucción de NVDiffrec. Después, insertamos esta malla en CNN de ajuste utilizando siluetas de la figura como supervisión.

Una vez obtenida la malla, mediante el algoritmo retopología Octree se obtiene una nueva malla retopologizada. Finalmente, mediante ray tracing obtenemos las propiedades del material de NVdiffrec [9] y las traspasamos a nuestra malla optimizada. La principal ventaja de este enfoque es que no tenemos que preocuparnos por el resto del algoritmo, ya que no tenemos que hacer grandes cambios en la forma en que funciona.

Para minimizar los cambios realizados en ambos algoritmos, decidimos insertar un nuevo bloque dedicado a la retopología (Figura **7-2**) inmediatamente después de que NVdiffrec [9] realice el proceso de reconstrucción de la malla. El enfoque es conveniente porque el bloque anterior proporciona una malla como salida, que es la entrada esperada del octree. Además, ya que el algoritmo de octree también proporciona una malla como salida, que es la entrada esperada del siguiente paso en el algoritmo NVdiffrec [9].

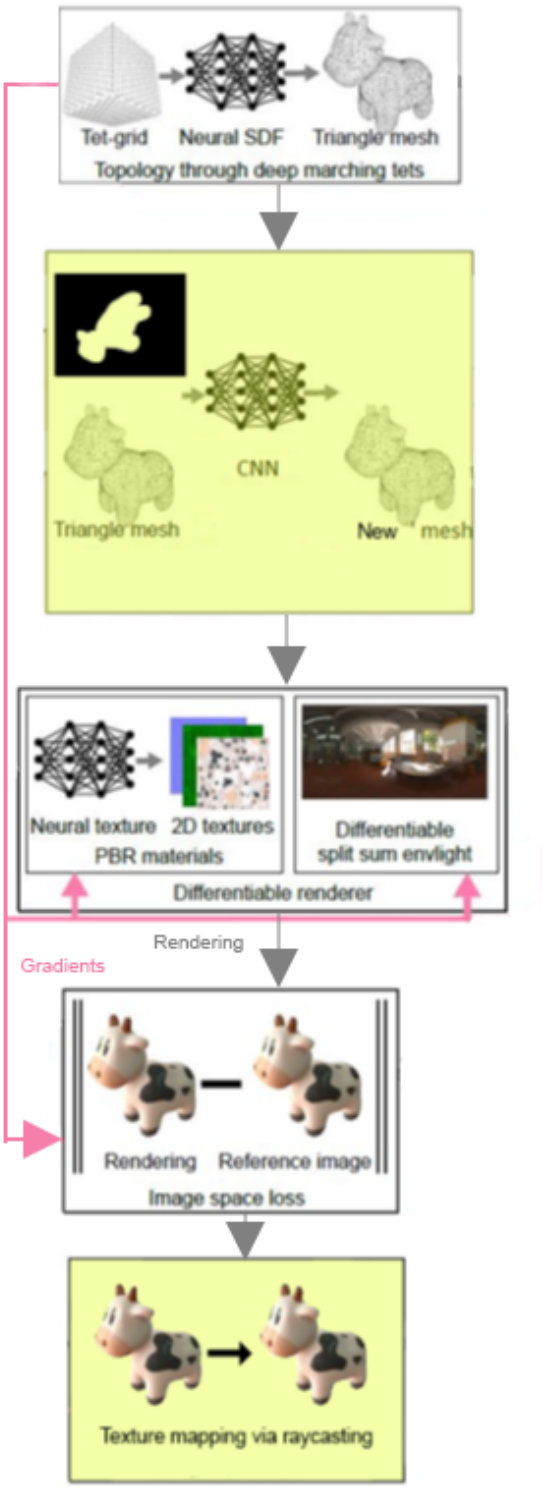

Figura 7-2.: Optimización de NVDiffrec (resaltado en amarillo)

Como podemos ver, los únicos cambios necesarios con este enfoque son la forma en que co-

nectamos ambos algoritmos y hacemos que la ecuación de gradiente funcione con la salida de SubDivNets en lugar del bloque original de NVdiffrec. [3].

El objetivo principal de este paso es adquirir una representación más precisa de la malla y también transformar las caras de los triángulos en cuadrados (este proceso se conoce como destriangulación). Esto es importante porque la mayoría del software de edición 3D prefiere los cuadrados para fines de edición, incluso si se triangulan una vez que se genera el resultado final. Como podemos ver en la Figura **7-3**, después del proceso de remallado se logra el propósito principal de este paso.

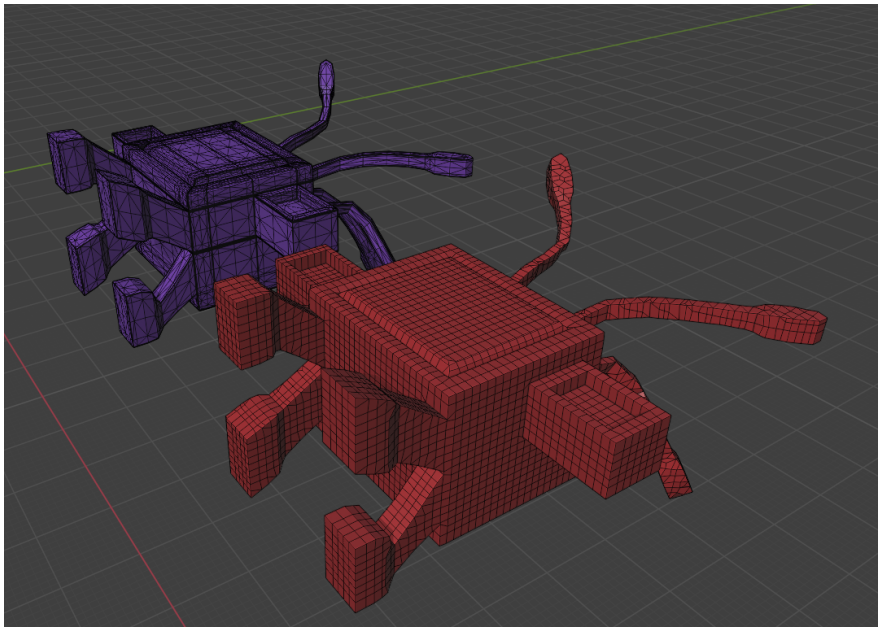

Figura 7-3.: Remallado y destriangulación

La nueva malla ahora se procesará con el resto del algoritmo NVdiffrec para generar los mapas difuso, normal y de rugosidad del material.

Para hacer la malla original mas precisa, se utilizó la red neuronal para detección de profundidad del algoritmo NVDiffrec.

Al utilizar la malla generada por NVDiffrec en combinación con las imágenes con profundidad nos permite realizar de manera muy precisa un proceso conocido como fusión de mapas de profundidad.

Figura 7-4.: Ejemplo de una imagen con profundidad

Luego utilizamos una red neuronal convolucional para realizar la regresión de la función de distancia truncada con signo (TSDF por sus siglas en inglés) y obtenemos una nube de puntos del objeto con mayor resolución a la que obtendríamos directamente de NVDiffrec.

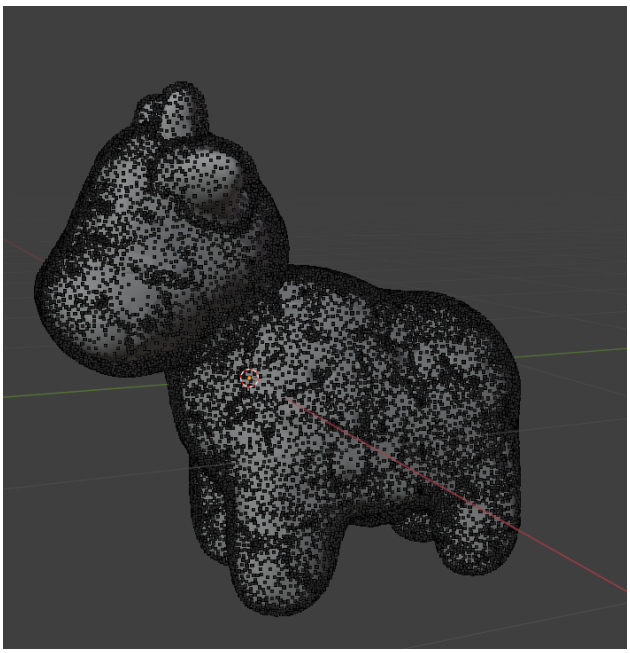

Figura 7-5.: Ejemplo de la nube de puntos

Una vez generada la nube de puntos hacemos el pareo con las imágenes de la silueta para reacomodar los vértices de la nueva figura. Una vez realizado este proceso se obtiene la figura con mayor resolución y una mejor aproximación a la imagen real.

Al procesar nuestro conjunto de imágenes con NVDiffrec garantizamos que nuestra red neuronal converja a uno de los mínimos de la función de error entre las imágenes de entrada y la renderización de la figura reconstruida. Este algoritmo nos ayuda a mejorar la precisión de la topología debido a que después de habernos asegurado que converge a uno de los mínimos del error, se incrementa la resolución de la malla y se realiza otro proceso de optimización con la red neuronal propuesta, esto nos sirve como refinamiento para acercarnos a un más al mínimo encontrado por NVDiffrec.

8. Resultados

8.1. Precisión del renderizado inverso

Se eligió el error cuadrático medio (MSE) para calcular la precisión del algoritmo optimizado.

La manera de calcular el error cuadrático medio es como sigue:

Sea I_O el conjunto de C imágenes renderizadas del modelo por nuestro algoritmo optimizado ,I_N el conjunto de imágenes de NVDiffrec y I_T el conjunto original de renderizaciones. Además, ya que todas las imágenes del mismo tamaño $m \times n$. La ecuación para el error cuadrático medio se define como:

$$MSE_{Opt} = \sum_{i=0}^{C} \sum_{j=0}^{mn} \left(\frac{(I_{O_{j_R}} - I_{T_{j_R}})^2 + (I_{O_{j_G}} - I_{T_{j_G})^2} + (I_{O_{j_B}} - I_{T_{j_B}})^2}{3mn} \right) \tag{8-1}$$

Para la versión optimizada, y para NVDiffrec queda como sigue:

$$MSE_{Opt} = \sum_{i=0}^{C} \sum_{j=0}^{mn} \left(\frac{(I_{N_{j_R}} - I_{T_{j_R}})^2 + (I_{N_{j_G}} - I_{T_{j_G}})^2 + (I_{N_{j_B}} - I_{T_{j_B}})^2}{3mn} \right) \tag{8-2}$$

NVDiffrec	NVDiffrec optimizado
3.55	1.88

Además, se generó la vizualización del error cuadrático haciendo la resta entre el resultado esperado y el obtenido en cada pixel.

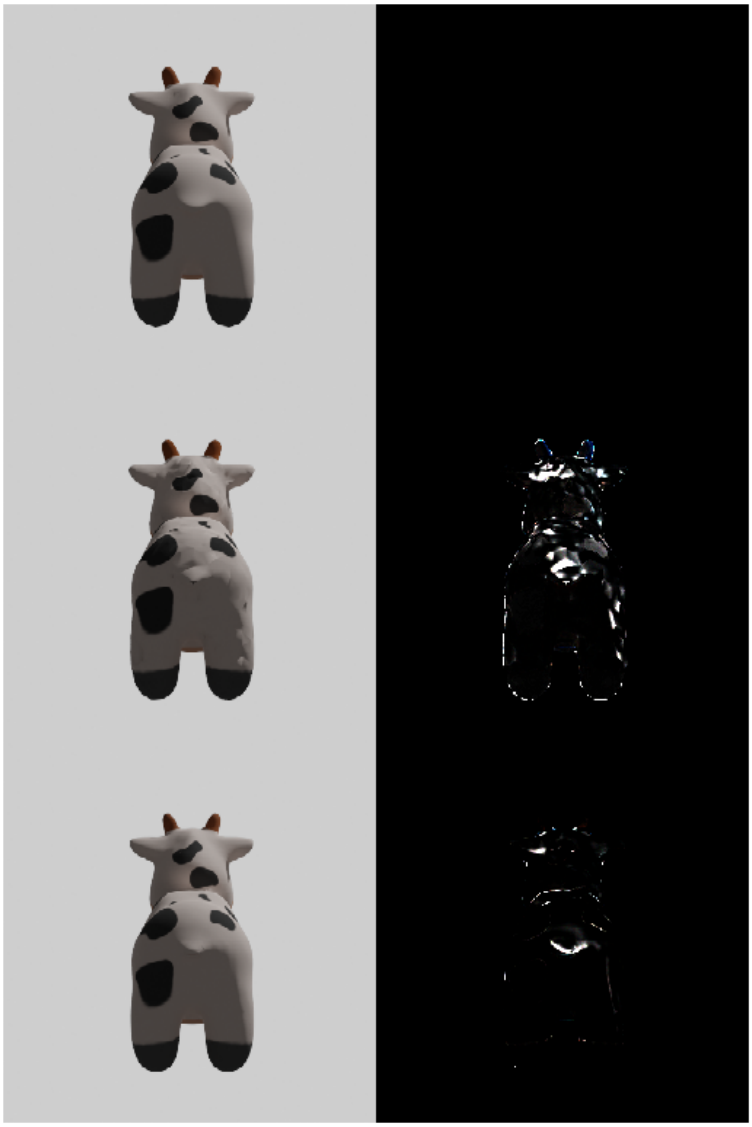

Figura 8-1.: Resultados obtenidos. Renderizado original, NVDiffrec, NVDiffrec optimizado, respectivamente)

8.2. Calidad del modelo reconstruido

Para medir la calidad del modelo reconstruido se utilizó el estándar de certificación de Checkmate Pro, en el cual se obtuvieron los Resultados siguientes:

Características	NVDiffrec	NVDiffrec optimizado
No tener vértices aislados		✓
No tener vértices coincidentes	✓	✓
No tener caras coincidentes o coplanares	✓	✓
Normales apuntando hacia afuera		✓
No tener objetos vacíos	✓	✓
20 % de triángulos o menos		✓
Buen flujo de aristas		✓
Escala 1:1		
Posición y orientación		
Modelo subdividible		
Texturas sin estiramiento notable		
Las divisiones del mapa de UVs no visibles		

8.3. Análisis de complejidad en tiempo

Debido a que el algoritmo consta de tres partes diferentes, en análisis en tiempo del algoritmo esta dividido en cada uno de sus subsistemas.

8.3.1. Redes neuronales

El sistema final consta de 3 redes neuronales convolucionales distintas. Nvdiffrec utiliza una para la generación de la topología y otra para el mapeo de las distintas texturas que utiliza el modelo Phong de renderizado.

Además de las redes neuronales utilizadas por NVDiffrec, se agregó una mas para el proceso de optimización del algoritmo, la cual se encarga de mejorar la regresión de la función SDF.

Para el análisis de complejidad de tiempo de las redes neuronales convolucionales se utilizo

la fórmula obtenida en [12] que nos permite saber la complejidad en tiempo en base a los elementos que componen a nuestra CNN.

$$\sum_{l=1}^{f} D \cdot W \cdot H \cdot N \tag{8-3}$$

Siendo f la profundidad de la red neuronal, D las dimensiones de la entrada, W el ancho de la entrada, H el alto de la entrada y N el número de salidas.

Red Neuronal SDF (NVDiffrec)

NVDiffrec implementa una red neuronal para su algoritmo DMTet (Deep Marching Tetrahedrions). Esta red neuronal usa como entrada un grid de tetraedros de tamaño fijo.

Sabemos que la profundidad de la red neuronal no cambia en ningún momento del proceso, por lo que podemos asumir a f como un valor constante.

Como la red neuronal genera un nuevo grid de tetraedros que es del mismo tamaño que el de la entrada, tanto los dimensiones de la entrada como de la salida son constantes.

Entonces, al tomar la ecuación 8-3, podemos ver que las 5 variables son constantes en la forma en la cual utilizamos nuestra red, resultando en una complejidad computacional de $O(1)$.

Podemos saber con certeza que la complejidad de $O(1)$ es correcta, porque aun si la figura requiere muchos detalles, la resolución del resultado final no cambia. Esto significa que se mantiene una complejidad de $O(1)$ a costo de perder resolución entre mas compleja sea la figura a reconstruir.

Red Neuronal para texturizado (NVDiffrec)

NVDiffrec además de utilizar una red para la regresión de la SDF, implementa una red neuronal para la reconstrucción de las texturas del material.

Las texturas entregadas por el algoritmo NVDiffrec consisten en 3 mapas diferentes de 2048×2048. Por lo tanto, N es constante.

También sabemos que como entrada se recibe la malla de tetraedros generada en la iteración pasada, por lo cual W, D y H también son constantes.

Finalmente, la red neuronal designada para esta tarea al igual que la red neuronal anterior es de estructura fija. Esto causa que también tengamos una f constante.

Debido a que la ecuación 8-3 en este caso esta también compuesta únicamente de constantes la complejidad es de $O(1)$.

Red Neuronal de optimización SDF

Finalmente, tenemos la red neuronal que se implementó como proceso de optimización para generar una topología más adecuada.

Esta red al igual que las anteriores es de estructura fija, con entrada y salida constantes. Sin embargo, esta red neuronal esta colocada dentro de un ciclo de optimización.

Para evitar que el algoritmo se cicle de manera permanente, este se detiene al alcanzar las 5000 iteraciones. En caso contrario, se detendrá al estar dentro de cierto umbral de error.

Debido a que para datos muy complejos el algoritmo se mantendrá constante en 5000 iteraciones obteniendo así una complejidad de $O(1)$ en nuestra optimización.

8.3.2. Octree

Como se mencionó en el marco teórico el algoritmo de octree para remeshing implica como mínimo la visita en cada uno de los hijos para un árbol octal, además de eso para cada hoja se lleva acabo un proceso de interpolación para ubicar los vértices en la mejor ubicación del cuadrante.

Este proceso de interpolación se realiza de la manera siguiente: Sea el V_n el conjunto de los últimos 8 nodos del árbol de forma tal que las aristas son V_1V_2, V_1V_3, V_1V_4, V_2V_5, V_2V_6, V_5V_7, V_6V_7, V_7V_8, V_3V_6, V_4V_5,V_3V_8, V_4V_8.

Debido a que en una malla triangulada todas las caras están compuestas de triángulos basta con considerar el caso donde un plano descrito por 3 puntos atraviesa el cubo formado por el ultimo conjunto de nodos. Siendo estos puntos P_1, P_2, P_3 se quiere averiguar a que altura las aristas del cubo se intersectan con el plano.

Para resolver este problema, primero se obtiene la ecuación que describe al plano mediante las ecuaciones siguientes:

$$A_1 = P_{2_x} - P_{1_x} \tag{8-4}$$
$$B_1 = P_{2_y} - P_{1_y} \tag{8-5}$$
$$C_1 = P_{2_z} - P_{1_z} \tag{8-6}$$
$$A_2 = P_{3_x} - P_{1_x} \tag{8-7}$$
$$B_2 = P_{3_y} - P_{1_y} \tag{8-8}$$
$$C_2 = P_{3_z} - P_{1_z} \tag{8-9}$$
$$a = B_1C_2 - B_2C_1 \tag{8-10}$$
$$b = A_2C_1 - A_1C_2 \tag{8-11}$$
$$c = A_1B_2 - B_1A_2 \tag{8-12}$$
$$d = (-aP_{1_x} - bP_{1_y} - cP_{1_z}) \tag{8-13}$$

Después, obtenemos la ecuación que describe las aristas que se intersectan con el plano. Por lo tanto, para cada una de ellas se resuelven las ecuaciones siguientes:

$$dx = V_{P_x} - V_{Q_x} \tag{8-14}$$
$$dy = V_{P_y} - V_{Q_y} \tag{8-15}$$
$$dz = V_{P_z} - V_{Q_z} \tag{8-16}$$

Finalmente, se encuentra el punto de intersección entre la recta y el plano:

$$nu = \frac{P_1}{|P_1|} \cdot \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} \tag{8-17}$$
$$w = V_P - P_1 \tag{8-18}$$
$$si = -\frac{P_1}{|P_1|} \cdot wPsi = w + si * \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} + P_1 \tag{8-19}$$

Debido a que estas operaciones se repiten una vez por cada hoja del árbol la complejidad final es $O(8^n)$ dónde n es el nivel de profundidad del octree.

La mayor ventaja de este algoritmo es que no depende de la malla de entrada. Por esta razón, la complejidad en tiempo del algoritmo octree en referencia a la entrada es de $O(1)$. Esto se debe a que no importa que tan grande sea la entrada, el algoritmo siempre se detendrá al nivel n independientemente del tamaño de la entrada.

8.3.3. Ray tracing

El algoritmo de ray tracing es el método mas común para renderizado utilizado hoy en día. Este algoritmo funciona emitiendo haces de luz desde la cámara (el punto elegido para visualizar la escena). Luego se calcula la dirección del haz de luz usando el pixél como referencia. Una vez que se realizó lo anterior, se emite el haz de luz y rebota en la escena un número predeterminado de veces

El algoritmo funciona de la manera siguiente:

1. Se define la cantidad de rebotes que tendrá el haz
2. Se selecciona un pixél de la imagen.
3. Se obtiene la dirección del haz dependiendo del pixél asignado.
4. Se rebota el haz en el primer material.
5. Si el material es vidrio el haz se divide en refracción y difracción. Y se realiza otro rebote hasta que rebote en un material difuso o se exceda la cantidad máxima de rebotes.
6. Si el objeto es un material difuso, se computa la iluminación.
7. Si faltan pixéles por procesar se pasa al siguiente pixél y se vuelve al paso 3.

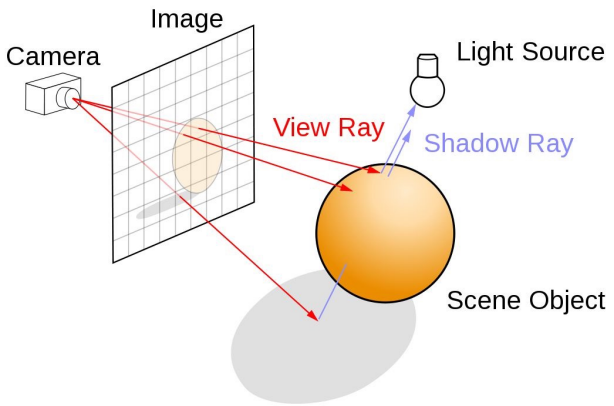

Figura 8-2.: Visualización del algoritmo de Ray Tracing (Imagen tomada de la documentación de desarrollo de Nvidia [15]

Por lo tanto, sea n la cantidad total de píxeles y k la cantidad máxima de rebotes. La complejidad computacional del mejor caso es de $O(n)$ (cuando todos los materiales son difusos) y el peor de los casos $O(2^k n)$ (cuando todos los materiales son vidrio).

Sin embargo, ya que en nuestro caso tanto la cantidad de rebotes como el tamaño de la imagen final son constantes. La imagen conserva sus dimensiones para ser compatible con las distintas capas de la red neuronal. Por otro lado, la cantidad de rebotes se trunca a un número predefinido, esto se debe a que la cantidad de rebotes que hace la luz en la vida real es demasiado grande como para poder computarla. Debido a esto, se obtiene una complejidad de $O(1)$.

9. Conclusión y trabajo a futuro

Se logró incrementar la precisión del algoritmo de reconstrucción de NVDiffrec y al mismo tiempo mejorar la topología resultante del algoritmo. Además, se obtuvo una complejidad final de $O(1)$.

Además de tener una mejor precisión se obtuvo una figura final que encaja mejor con los estándares de calidad utilizados por los profesionales en la industria del modelado 3D. Nuestro algoritmo está un paso más cerca de ser implementado en la industria, debido a que nuestro resultado final se acerca mas a satisfacer las exigencias de TurboSquid.

El haber hecho uso de herramientas mas tradicionales del renderizado de modelos 3D (como el algoritmo de Raycasting y el Octree) en conjunto con las redes neuronales nos sirvió para poder incrementar considerablemente la precisión sin sacrificar la complejidad en tiempo del algoritmo.

9.1. Trabajo a futuro

Como se pudo observar en la sección de resultados, a pesar de que se logró una mejoría sobre el algoritmo original de NVDiffrec, aún faltan muchos requisitos a satisfacer para automatizar el proceso de reconstrucción 3D al nivel de un artista 3D profesional.

Los principales puntos a mejorar es la disminución del estiramiento y la generación de las divisiones de UVs en las partes mas ocultas de la malla.

Además falta mejorar también que la reconstrucción sea automaticamente en escala 1:1 con el objeto real y la posición y orientación correcta.

Finalmente, también sería importante la demostración o refutación de si el algoritmo Octree para retopología nos va a generar que no pierda resolución al subdividirse si se usa la profundidad adecuada para el mismo.

A. Anexo: Imágenes de la retopología

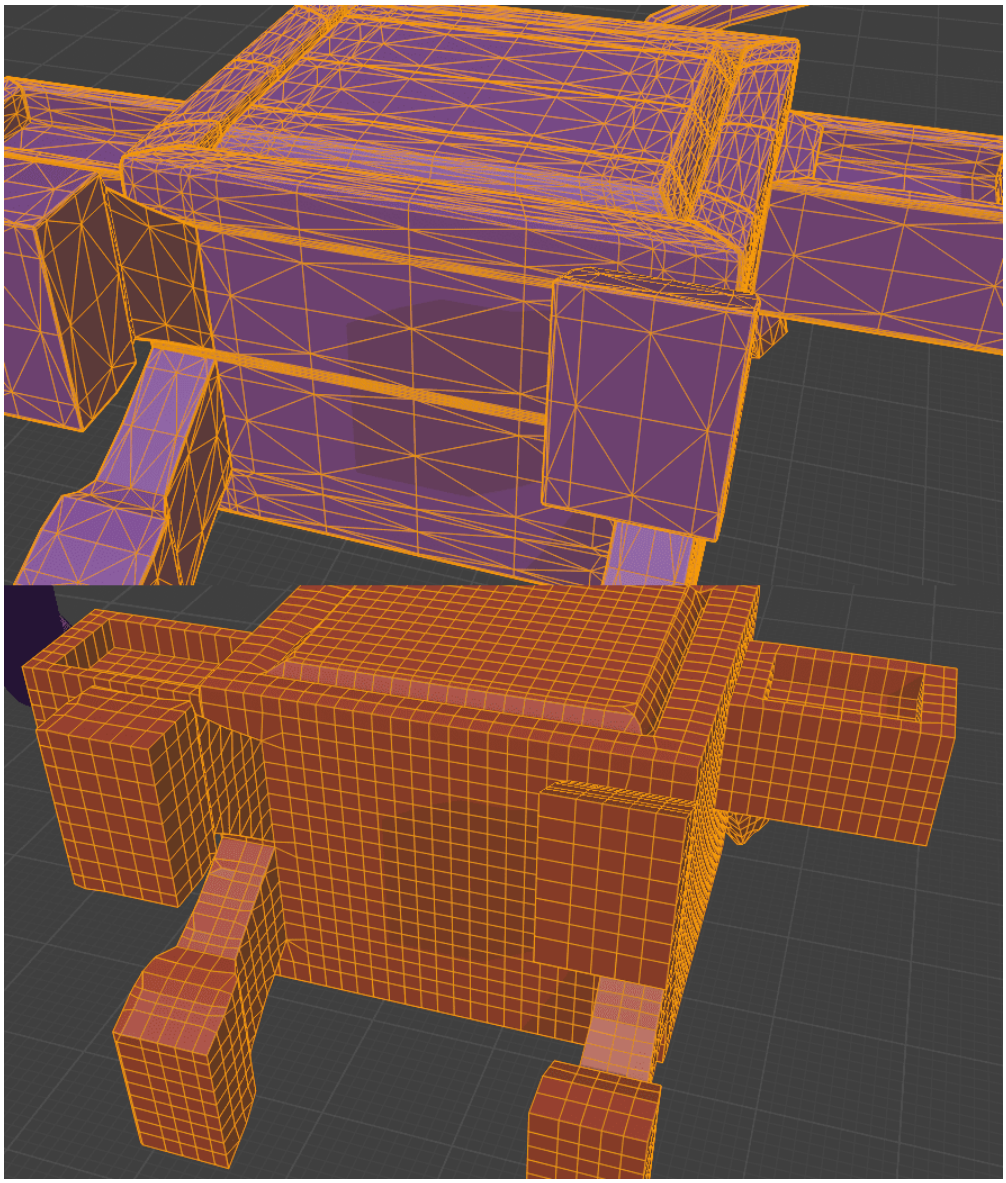

Figura A-1.: Topología original (Morado), topología optimizada (Rojo)

B. Anexo: Imágenes de los resultados

Este anexo contiene las imágenes ampliadas de los resultados.

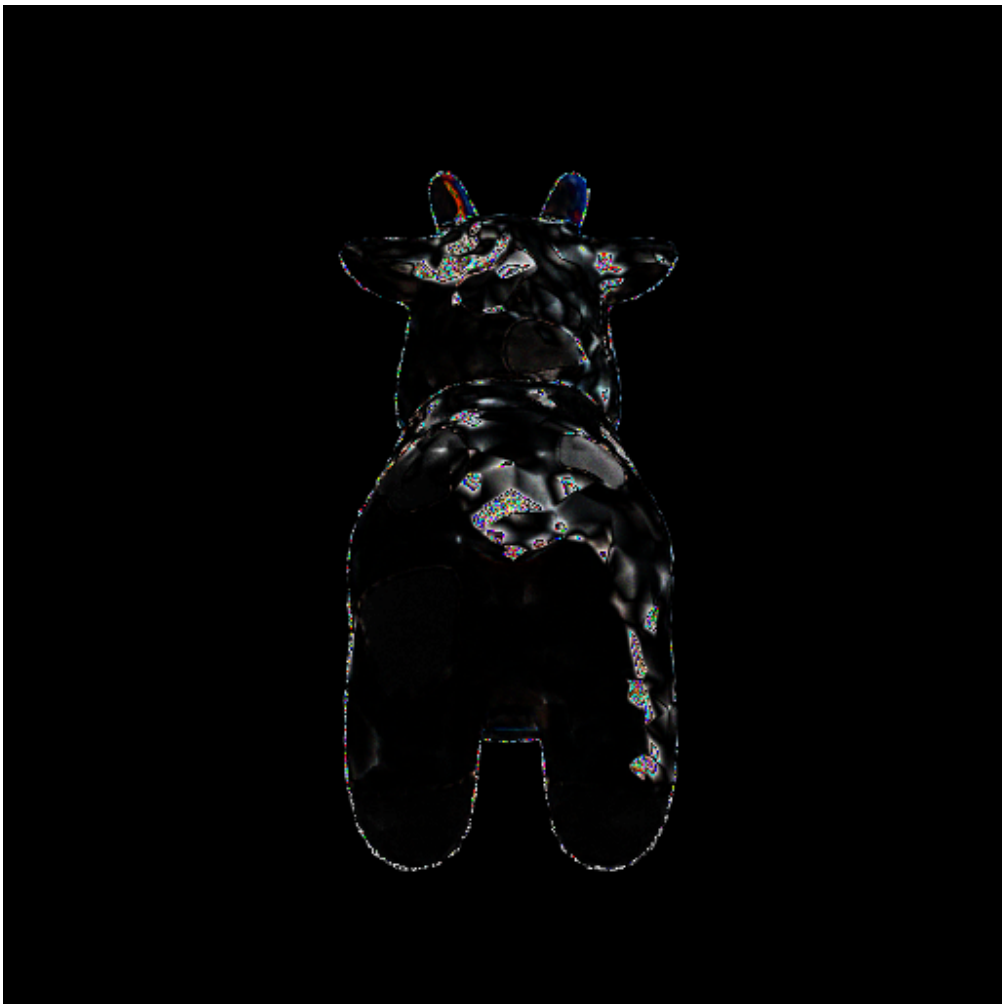

Figura B-1.: Resultado antes de optimizar

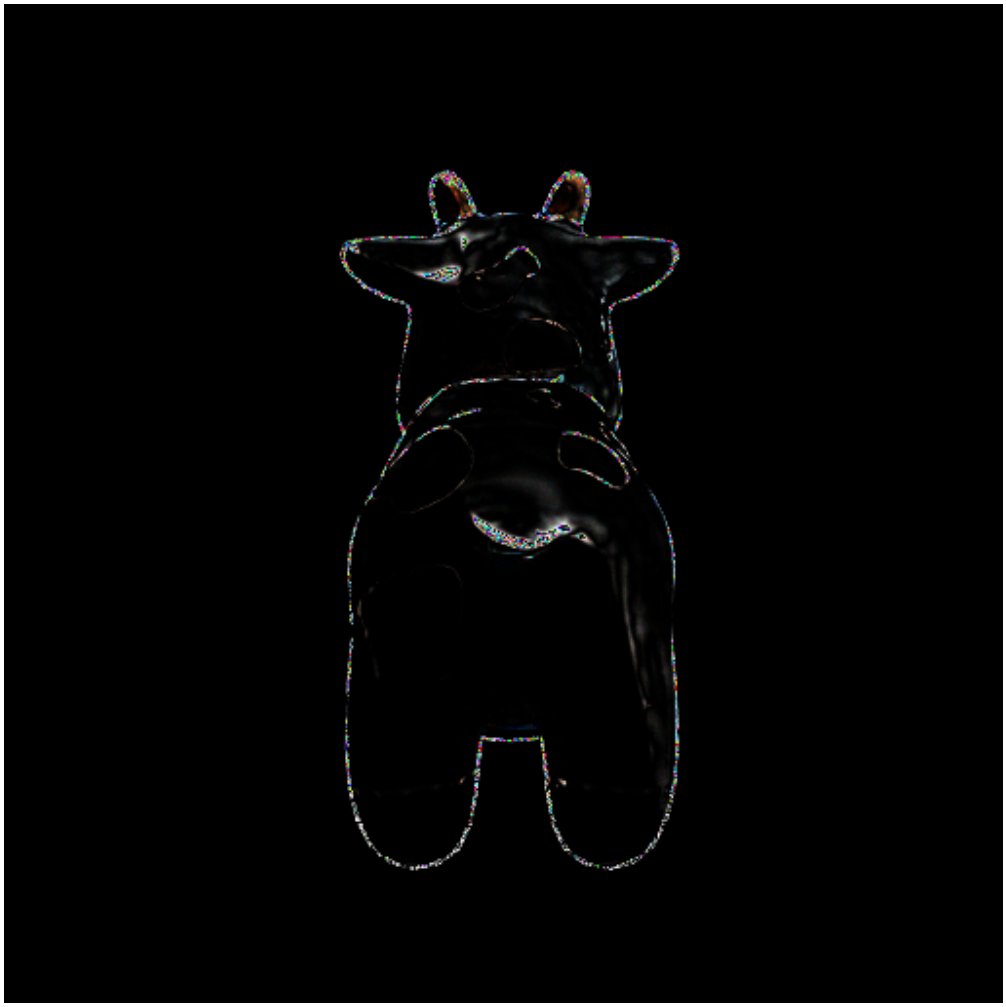

Figura B-2.: Resultado antes de optimizar

Figura B-3.: Mejoría de nuestro algoritmo en comparación con NVDiffrec (calculado mediante B-1)

Sea $H(x)$ la función escalón de Heaviside, MSE_{Delta} la mejoía de nuestro algoritmo en comparacioń con el original, MSE_{Opt} la matriz $(512 \times 512 \times 3)$ del error cuadrático medio del algoritmo optimizado y MSE_{Orig} la matriz $(512 \times 512 \times 3)$ del error cuadrático medio del algoritmo original.

$$MSE_{Delta} = H(MSE_{Opt} - MSE_{Orig})(MSE_{Opt} - MSE_{Orig}) \tag{B-1}$$

C. Anexo: Ejemplo de loop-cutting

Como se mencionó en el documeto, los cuadrilateros tienen dos direcciones en los motores de edición de modelos 3D.

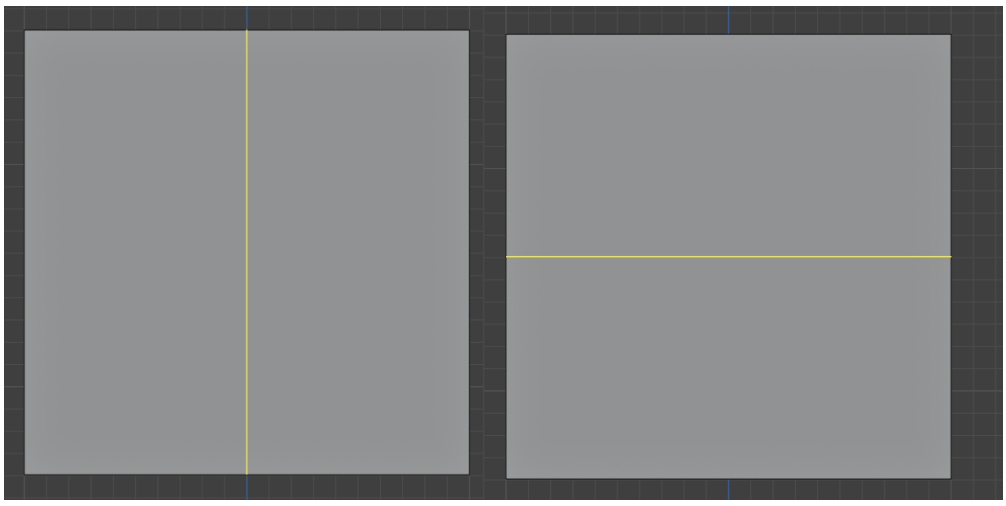

Figura C-1.: Ejemplo de las dos direcciones de corte que posee un cuadrilátero

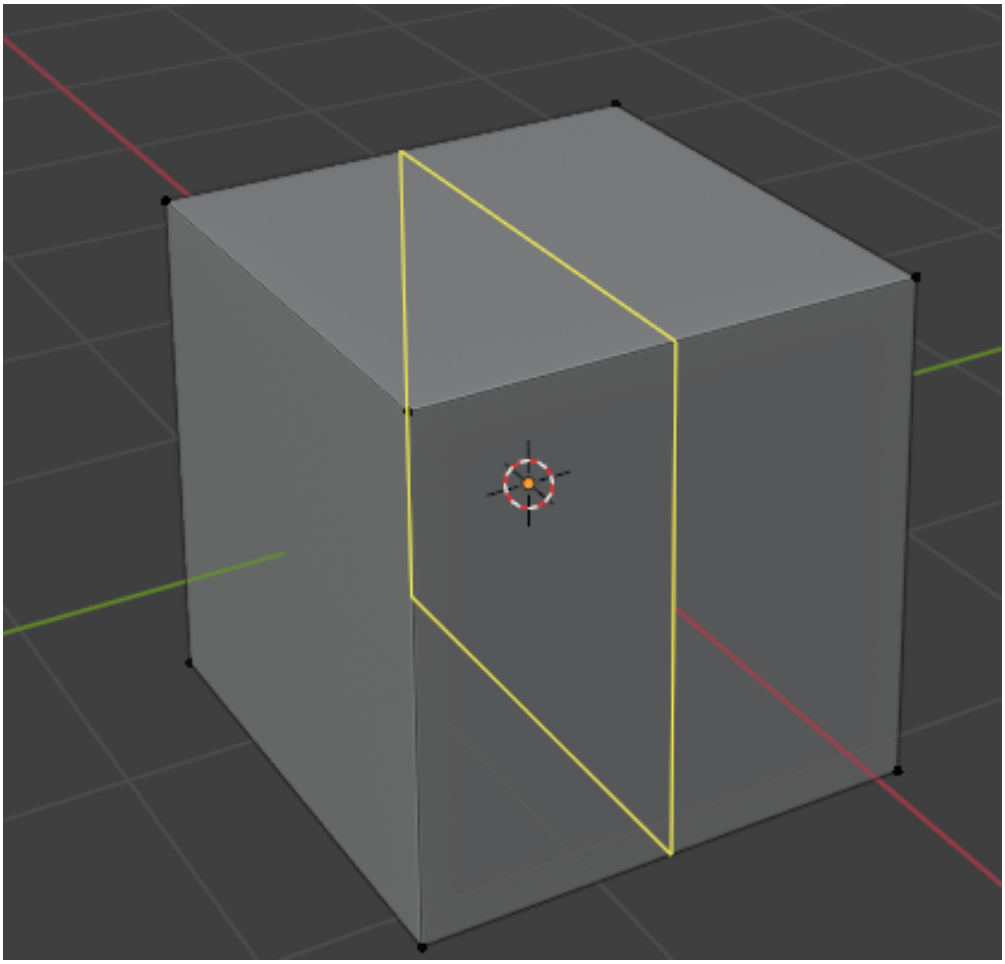

Figura C-2.: Como se puede observar el corte se propaga automáticamente sobre la dirección del cuadrilátero

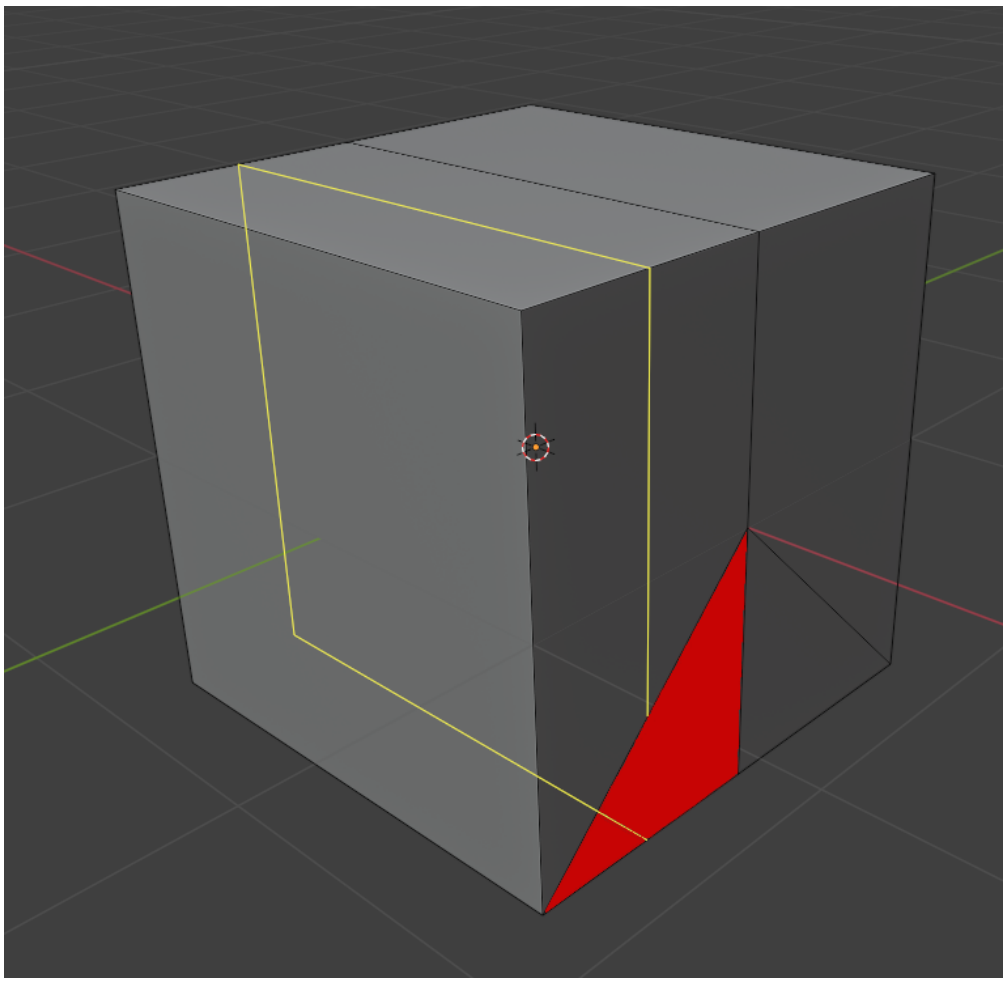

Figura C-3.: Como se puede observar al aparecer triángulos en la figura la propagación se interrumpe debido a la carencia de dirección de los mismos

Bibliografía

[1] Chen, Wenzheng ; Litalien, Joey ; Gao, Jun ; Wang, Zian ; Tsang, Clement F. ; Khamis, Sameh ; Litany, Or ; Fidler, Sanja: DIB-R++: Learning to Predict Lighting and Material with a Hybrid Differentiable Renderer. En: *CoRR* abs/2111.00140 (2021)

[2] Gao, Kyle ; Gao, Yina ; He, Hongjie ; Lu, Denning ; Xu, Linlin ; Li, Jonathan: NeRF: Neural Radiance Field in 3D Vision, A Comprehensive Review. (2022)

[3] Hu, Shi-Min ; Liu, Zheng-Ning ; Guo, Meng-Hao ; Cai, Junxiong ; Huang, Jiahui ; Mu, Tai-Jiang ; Martin, Ralph R.: Subdivision-based Mesh Convolution Networks. En: *ACM Trans. Graph.* 41 (2022), Nr. 3, p. 25:1–25:16

[4] Lee, Dogyoon ; Lee, Minhyeok ; Shin, Chajin ; Lee, Sangyoun: DP-NeRF: Deblurred Neural Radiance Field with Physical Scene Priors. (2022)

[5] Ling, Huan ; Gao, Jun ; Kar, Amlan ; Chen, Wenzheng ; Fidler, Sanja: Fast Interactive Object Annotation with Curve-GCN. (2019)

[6] Liu, Zheng-Ning ; Cao, Yan-Pei ; Kuang, Zheng-Fei ; Kobbelt, Leif ; Hu, Shi-Min: High-Quality Textured 3D Shape Reconstruction with Cascaded Fully Convolutional Networks. En: *IEEE Transactions on Visualization and Computer Graphics* 27 (2021), Nr. 1, p. 83–97

[7] Milano, Francesco ; Loquercio, Antonio ; Rosinol, Antoni ; Scaramuzza, Davide ; Carlone, Luca: Primal-Dual Mesh Convolutional Neural Networks. En: *CoRR* abs/2010.12455 (2020)

[8] Mildenhall, Ben ; Srinivasan, Pratul P. ; Tancik, Matthew ; Barron, Jonathan T. ; Ramamoorthi, Ravi ; Ng, Ren: NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. (2020)

[9] Munkberg, Jacob ; Hasselgren, Jon ; Shen, Tianchang ; Gao, Jun ; Chen, Wenzheng ; Evans, Alex ; Mueller, Thomas ; Fidler, Sanja: Extracting Triangular 3D Models, Materials, and Lighting From Images. En: *arXiv:2111.12503* (2021)

[10] Rosenblatt, F.: The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. En: *Psychological Review* 65 (1958), Nr. 6, p. 386–408

[11] Schõnberger, Johannes ; Zheng, Enliang ; Pollefeys, Marc ; Frahm, Jan-Michael: Pixelwise View Selection for Unstructured Multi-View Stereo. 9907 (2016), 10. ISBN 978–3–319–46486–2

[12] Shah, Bhoomi ; Bhavsar, Hetal: Time Complexity in Deep Learning Models. En: *Procedia Computer Science* 215 (2022), p. 202–210. – 4th International Conference on Innovative Data Communication Technology and Application. – ISSN 1877–0509

[13] Shen, Tianchang ; Gao, Jun ; Yin, Kangxue ; Liu, Ming-Yu ; Fidler, Sanja: Deep Marching Tetrahedra: a Hybrid Representation for High-Resolution 3D Shape Synthesis. (2021)

[14] Wang, Peng ; Zhao, Lingzhe ; Ma, Ruijie ; Liu, Peidong: BAD-NeRF: Bundle Adjusted Deblur Neural Radiance Fields. (2022)

[15] Wang, Peng ; Zhao, Lingzhe ; Ma, Ruijie ; Liu, Peidong: NVidia Developer Documentation. (2023)

[16] Wang, Yue ; Sun, Yongbin ; Liu, Ziwei ; Sarma, Sanjay E. ; Bronstein, Michael M. ; Solomon, Justin M.: Dynamic Graph CNN for Learning on Point Clouds. En: *CoRR* abs/1801.07829 (2018)