Compresión de modelos 3D usando Transformadas de Fourier

¿Cómo se pueden utilizar las Transformadas de Fourier para aproximar modelos 3D?

Palabras: 3945

Código Alumno: jsx364

Convocatoria mayo 2022

Índice de contenidos

1. INTRODUCCIÓN

En esta Monografía se explora las series de Fourier desde un punto de vista matemático. Junto con esta investigación teórica también se estudiará las aplicaciones prácticas del Análisis de Fourier, en concreto, siguiendo la pregunta: ¿Cómo se pueden utilizar las Transformadas de Fourier para aproximar modelos 3D?

En un principio, se plantearán las Series de Fourier como concepto matemático, seguido de las Transformadas de Fourier. A continuación, se hará uso de estos conocimientos para explicar sus diferentes aplicaciones, y seguidamente cumplir con una de ellas, la aplicación gráfica de las series de Fourier, haciendo una compresión de un modelo 3D, de formato PLY.

1.1. Objetivos

Los objetivos que quiero lograr a lo largo de este trabajo son los siguientes:

- Responder a la pregunta de investigación: ¿Cómo se pueden utilizar las Transformadas de Fourier para aproximar modelos 3D?, y a la vez, ¿Cómo comparan con métodos tradicionales?
- Responder a las preguntas: "¿Qué es el análisis de Fourier?", "¿Cómo se puede visualizar una Serie de Fourier?" y "¿Cómo se usan las Transformadas de Fourier para aproximar funciones?"
- Obtener los conocimientos necesarios para realizar el trabajo
- Aprender a organizar mi tiempo para hacer un trabajo externo
- Finalmente, hacer un trabajo del que esté satisfecho

1.2. Metodología

He decidido dividir este trabajo en tres capítulos:

1. Marco Teórico: En este primer capítulo hay una introducción teórica de lo necesario para seguir el trabajo y de los conceptos que se usan a lo largo de este. Este apartado empieza con una definición de conceptos y seguidamente las propiedades de funciones que se emplean durante el trabajo y son relevante a este.
2. Análisis de Fourier: En este capítulo se contesta a las preguntas: "¿Qué es el Análisis de Fourier?". Empieza con una breve introducción sobre su historia, seguida de la definición de que son las series y transformadas de Fourier y sus diferencias. En este apartado se intenta que el lector comprenda los fundamentos básicos que serán aplicados en el apartado siguiente.
3. Aplicaciones: El último capítulo de este trabajo empieza listando todas las posibles aplicaciones que tienen las Series de Fourier. Seguidamente, se responde a la pregunta: "¿Cómo se puede utilizar las Transformadas de Fourier para aproximar objetos 3D?", utilizando de un programa informático.

Para poder representar las figuras y toda la información recopilada, se utilizarán una variedad de programas. Matlab, para representar figuras en el plano 3D, y para procesar la información obtenida se utilizarán programas escritos en C++.

1.3. Motivación personal

A principio de mi bachillerato, me encontré con un video en el que se podía ver una animación de la Serie Compleja de Fourier: "*Pure Fourier series animation montage*" de *3Blue1Brown*, uno de los creadores de contenido matemático más conocido en *Youtube*. En el video, se podía ver como la suma de un conjunto de vectores rotando a una frecuencia constante dibujaba una figura con el tiempo (Ilustración 1).

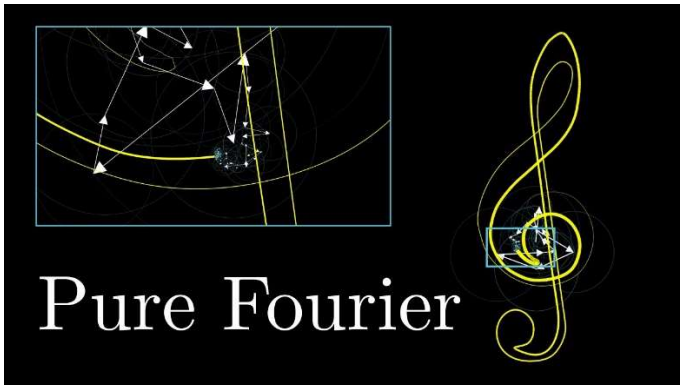


Ilustración 1*:* Video *Pure Fourier series animation montage*

Siendo yo una persona curiosa, decidí hacer investigación sobre el concepto matemático que había detrás de tal animación. Sin saber dónde me metía, entre en el mundo del análisis de Fourier. Además, tengo la suerte que mis dos padres son graduados de ingeniería de telecomunicaciones y están muy bien informados en el tema. En consecuencia, decidí hacer de esta investigación mi Monografía.

2. MARCO TEÓRICO

Seguidamente, se exponen todos los conocimientos previos necesarios para poder comprender lo que se tratara en el trabajo. Antes de empezar el trabajo, se presupone que el lector tiene conocimiento sobre cómo realizar integrales.

2.1. Funciones Periódicas

Diremos que una función es periódica cuando exista un número real T, tal que el valor de la función en el punto x coincida con el valor de la función en el punto $x + T$: $f(x) = f(x + T)$. Llamamos periodo al número real T que nos indica cada cuanto se repite la función.[1]

2.2. Números complejos

Al número $z = a + bi$ se le llama número complejo. En general, cualquier número complejo se denota por la letra z y donde i es la unidad imaginaria: $\sqrt{-1} = i$. Al número a se llama **parte real** del número complejo y se denota $a = Re(z)$, mientras que al número b se llama **parte imaginaria** del número complejo y se denota por $b = Im(z)$.

El conjunto de los números complejos se defino como: $\mathbb{C} = \{a + bi | a, b \in \mathbb{R}\}$

En matemáticas, el **plano complejo** es una forma de visualizar y ordenar el conjunto de los números complejos. Puede entenderse como un plano cartesiano modificado, en el que la parte real está representada en el eje de abscisas y la parte imaginaria en el eje de ordenadas. De esta manera representas

los números complejos como vectores [3]. El número complejo z, es equivalente a un vector en el plano complejo y se puede expresar en forma polar:

$$z = a + bi \Rightarrow \alpha = \tan^{-1}\left(\frac{b}{a}\right) \; y \; |X_n| = \sqrt{a^2 + b^2}$$

2.2.1. Formula de Euler

La fórmula de Euler, atribuida a Leonhard Euler, establece que:

$$e^{i\varphi} = \cos(\varphi) + i\, sin(\varphi)$$

Para todo número real x, que representa un ángulo en el plano complejo. Aquí, e es la constate de Euler y i es la unidad imaginaria [4]. Esta fórmula lo que nos dice es que $e^{i\varphi}$ es un vector de magnitud 1 con inicio en el centro de coordenadas y que tiene un ángulo φ (Figura 1). Así mismo, si $\varphi = \omega t$; donde t va variando, obtenemos un vector que rota sobre el circulo unitario del plano complejo. Las coordenadas del número complejo z que describe son:

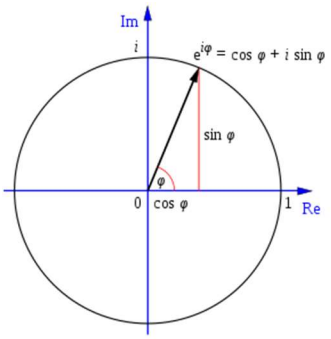


Figura 1: Circulo de la unidad para representar e^{ix}

$$Re(z) = \cos(\varphi) \quad Im(z) = sin(\varphi)$$

2.2.2. Amplitud, Fase y Frecuencia de $e^{i\omega t}$

Nos referimos al ángulo en que empieza a rotar como fase (ω), el módulo del vector como amplitud y a la rapidez del vector cuando gira como frecuencia (como de rápido varía t).

3. ANALISIS DE FOURIER

3.1. Introducción

Durante la extensión de este trabajo, trataremos dos temas principales, las series de Fourier y las transformadas de Fourier. Estos dos conceptos matemáticos se basan en una misma idea, representar un fragmento (o periodo) de una función como la suma de funciones trigonométricas (senos y cosenos). Por un lado, las series de Fourier sirven para analizar funciones periódicas a través de la descomposición de dicha función en una suma infinita de funciones sinusoidales mucho más simples (como combinación de senos y cosenos con frecuencias enteras y su peso apropiado) [5]. En cambio, la transformada de Fourier es usada para funciones aperiódicas.

Estos dos conceptos constituyen una herramienta muy importante en la solución de problemas físicos en las que intervienen ecuaciones diferenciales ordinarias y parciales. Asimismo, el análisis de Fourier, también es utilizado en muchas ramas de la ingeniería, como el análisis vibratorio, acústica, óptica, procesamiento de imágenes y señales, y comprensión de datos.

Antes de empezar, hemos de clarificar que el trabajo estará enfocado en el análisis de Fourier en un tiempo discreto (DT), a diferencia de un tiempo continuo (CT). Esto es relevante de clarificar, ya que existe una variación según el dominio de tiempo en ambos, la serie de Fourier y las transformadas de Fourier.

3.1.1. Historia

La introducción al Análisis de Fourier fue uno de los mayores avances jamás realizados en la física matemática y en sus aplicaciones en la ingeniería, ya que es utilizado en nuestro día a día, aunque no nos demos cuenta [6].

El Análisis de Fourier fue aplicado por primera vez en el siglo XVIII por Jean-Baptiste Joseph Fourier con el propósito de resolver la ecuación de calor en 1807 *Mémoire sur la propagation de la chaleur dans les corps solides* y la publicación de la obra *Théorie analytique de la chaleur* en 1822. Aunque fue inspirado por el trabajo anterior de Daniel Bernoulli. Leonhard Euler o Jean Le Rond d'Alembert.

La ecuación de calor es una ecuación diferencial parcial, a la cual Fourier dio solución. Antes del trabajo de Fourier no se conocía ninguna solución excepto en los casos excepcionales en que la fuente de calor tomaba una forma de una oda sinusoidal. Fourier, puesto de modo simple, decidió expresar cualquier otra función como la superposición o combinación lineal de simple ondas sinusoidales y cosenoidales, y obtener la solución a partir de estas. Esta sobreposición o combinación lineal de ondas recibe el nombre de series de Fourier [7].

3.2. Series de Fourier

Empecemos introduciendo la serie de Fourier. Si un intervalo de la función $f(t)$ cumple con las condiciones de Dirichlet (explicadas en el siguiente apartado), entonces se puede obtener el desarrollo de la serie de Fourier convergente de $f(t)$. La serie de Fourier, en forma trigonométrica, asociada a $f(t)$ es [8]:

$$f(t) = \frac{a_o}{2} + \sum_{n=1}^{\infty} (a_n \cdot \cos(\omega_n t) + b_n \cdot \sin(\omega_n t)) ; \qquad \text{Donde } \omega_n = \frac{2n\pi}{T}$$

Donde a_0 es el promedio de la función $f(t)$ y se calcula integrando la función periódica en su periodo T.

$$a_0 = \frac{2}{T}\int_t^{t+T} f(t)\,dt$$

Donde a_n y b_n se denominan coeficientes de Fourier de la serie de Fourier $f(t)$, y se obtienen realizando la integral de la función $f(t)$ multiplicado por el coseno o el seno respectivo, de $\omega_n t$, en un periodo de tiempo y dividiendo por $T/2$.

$$a_n = \frac{2}{T}\int_t^{t+T} f(t)\cdot\cos(\omega_n t)\,dt, \quad b_n = \frac{2}{T}\int_t^{t+T} f(t)\cdot\sin(\omega_n t)\,dt$$

Usando la fórmula trigonométrica (1) de la Serie de Fourier, es necesario calcular los 3 coeficientes para así obtener la Serie de Fourier de la función periódica $f(t)$.

Veamos un ejemplo [9], consideremos la función (Figura 2):

$$f(t) = \begin{cases} 0 & para\ -\pi < t < 0 \\ \pi - t & para\quad 0 \le t < \pi \end{cases}$$

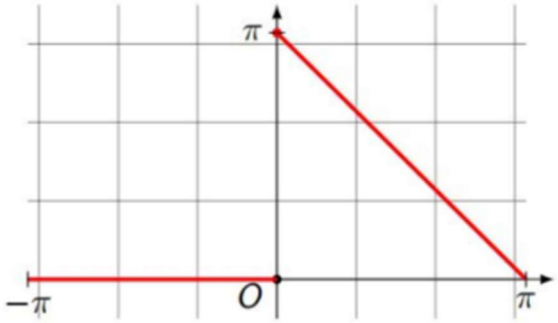


Figura 1: Representación de la función $f(t)$ del ejercicio

Veamos que su está definida en un intervalo cerrado $[-\pi, \pi]$ (consideramos este intervalo como su periodo, como si se fuera repitiendo), $T = 2\pi$. Entonces:

$$\omega_n = \frac{2n\pi}{T} = \frac{2n\pi}{2\pi} = n \implies f(t) = \frac{a_o}{2} + \sum_{n=1}^{\infty}(a_n\cdot\cos(nt) + b_n\cdot\sin(nt))$$

Calculemos los distintos coeficientes, considerando $t = -\pi$:

En un principio calculamos a_0:

$$a_0 = \frac{2}{2\pi}\int_{-\pi}^{\pi} f(t)\,dt = \frac{1}{\pi}\int_{-\pi}^{0} 0\,dt + \frac{1}{\pi}\int_{0}^{\pi} (\pi - x)\,dt = \frac{1}{\pi}\int_{0}^{\pi} (\pi - x)\,dt = \frac{1}{\pi}\left[\pi x - \frac{x^2}{2}\right]_0^{\pi} = \frac{\pi}{2}$$

Ahora se calcula a_n (se han ahorrado algunos cálculos para no hacer pesada la lectura del trabajo):

$$a_n = \frac{2}{2\pi}\int_{-\pi}^{\pi} f(t)\cdot\cos(nt)\,dt = \frac{1}{\pi}\int_{-\pi}^{0} 0\cdot\cos(nt)\,dt + \frac{1}{\pi}\int_{0}^{\pi} (\pi - x)\cdot\cos(nt)\,dt =$$

$$= \frac{1}{\pi}\int_{0}^{\pi} (\pi - x)\cdot\cos(nt)\,dt = \cdots = \frac{1-\cos(n\pi)}{\pi\cdot n^2} = \frac{1-(-1)^n}{\pi\cdot n^2}$$

Lo mismo con b_n:

$$b_n = \frac{2}{2\pi}\int_{-\pi}^{\pi} f(t)\cdot\sin(nt)\,dt = \frac{1}{\pi}\int_{-\pi}^{0} 0\cdot\sin(nt)\,dt + \frac{1}{\pi}\int_{0}^{\pi} (\pi - x)\cdot\sin(nt)\,dt = \cdots =$$

$$= \frac{1}{\pi}\cdot\left[-\frac{\pi}{n}\cos(nx) - \left(\frac{1}{n}\sin(nx) - \frac{x}{n}\cos(nx)\right)\right]_0^{\pi} = \frac{1}{n}$$

Substituyendo a la expresión de $f(t)$ que nos da la serie de Fourier en su forma trigonométrica obtenemos:

$$a_0 = \frac{\pi}{2} \qquad a_n = \frac{1-(-1)^n}{\pi\cdot n^2} \qquad b_n = \frac{1}{n}$$

$$f(t) = \frac{\pi}{4} + \sum_{n=1}^{\infty}\left(\frac{1-(-1)^n}{\pi\cdot n^2}\cdot\cos(nt) + \frac{1}{n}\cdot\sin(nt)\right)$$

Veamos cómo se aproxima la serie de Fourier a la función inicial. Consideremos, en vez de un sumatorio de $n = 1$ hasta infinito, sumatorios finitos (S_1, S_2, S_5, S_{100}) (Figura 3):

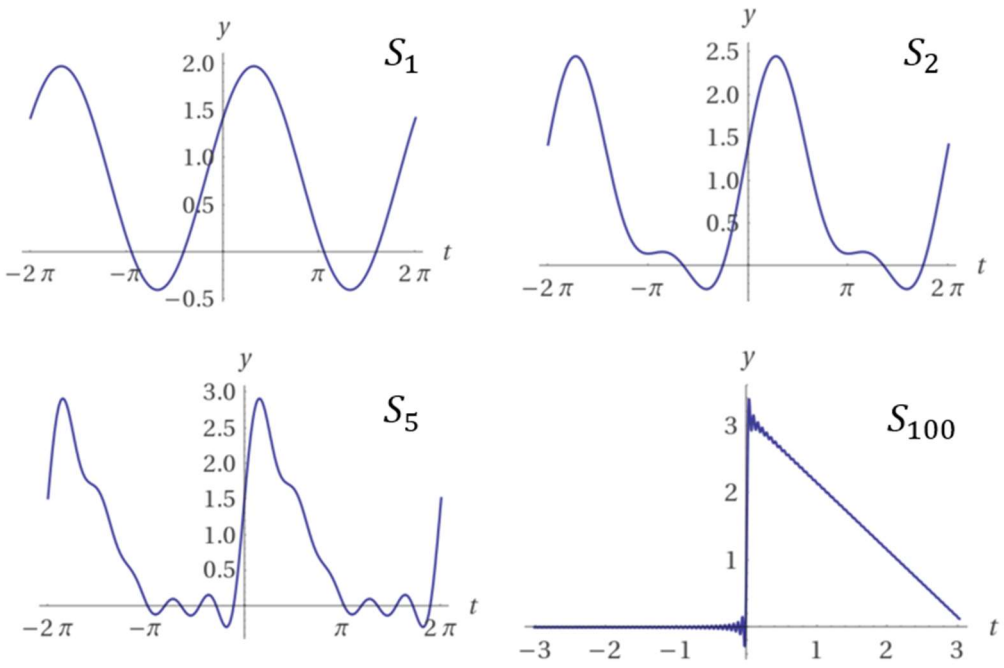


Figura 2: Representación gráfica de las aproximaciones de la función $f(t)$, usando S_n términos del sumatorio

3.2.1. Condiciones de Dirichlet

Anteriormente, hemos dicho que las Series de Fourier se usa para funciones periódicas o definidas en un intervalo finito, pero esta afirmación no siempre es válida. Para que exista una Serie de Fourier convergente, la función ha de primero cumplir con unas condiciones que reciben el nombre de "Las condiciones de Dirichlet" [10]:

1. La función $f(t)$ tiene un número finito de discontinuidades en su periodo
2. La función $f(t)$ debe tener un número finito de máximos y mínimos en el periodo
3. El valor absoluto de $f(t)$ ha de ser integrable en su rango de periodo:

$$\int_0^T |f(t)|\, dt < \infty$$

Si se cumplen, la serie de Fourier de $f(t)$ converge a $f(t)$ en todos los puntos donde la función es continua.

3.3. Formulación en forma compleja

Teniendo en cuenta la fórmula establecida anteriormente, la podemos transformar para obtener una forma de la serie de Fourier en el plano complejo:

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cdot \cos(n\omega_0 t) + b_n \cdot \sin(n\omega_0 t))$$

Usando la fórmula de Euler podemos transformar las funciones trigonométricas al plano complejo:

$$e^{i\varphi} = \cos(\varphi) + i\, sin(\varphi)$$

$$\cos(x) = \frac{e^{ix} + e^{-ix}}{2} \qquad \text{sen}(x) = \frac{e^{ix} - e^{-ix}}{2i}$$

Substituyendo y expandiendo, se llega a lo que es conocido como la serie de Fourier en su forma compleja:

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{i\omega_n t}$$

Donde los coeficientes (c_n) ahora serían:

$$c_n = \frac{1}{T} \int_{-T/2}^{T/2} f(t) \cdot e^{-i\omega_n t}\, dt$$

Ahora notemos que implica la serie de Fourier expresada de esta manera. Si nos fijamos en el término exponencial y en la fórmula de Euler, nos está diciendo que cualquier función $f(t)$ que cumpla con los requisitos, se puede expresar como la

suma de vectores rotando en el plano complejo. Estos vectores son modificados por un coeficiente c_n, este modifica la amplitud y la fase inicial de cada uno de los vectores, que rotan a una velocidad constante determinada por ω_n. Como ejemplo, imaginemos que queremos que uno de estos vectores empiece a unos $\pi/4$ radianes y que tenga una amplitud de 0.5, podemos hacer lo siguiente:

$$c_n e^{i\omega_n t} \rightarrow 0.5(e^{(\pi/4)i})e^{i\omega_n t}$$

En la serie de Fourier en forma compleja, esto se hace para cada uno de los vectores que componen la función.

Aquí termina nuestro estudio teórico de las Series de Fourier, pero para las computaciones, ya que un ordenador no puede realizar una integral, usaremos lo que es conocido como las Transformadas de Fourier.

3.4. Transformadas de Fourier

Como mencionado anteriormente, las Series de Fourier solo sirven para unas funciones que cumplen las condiciones de Dirichlet. En consecuencia, cuando se aplica los conceptos establecidos en el Análisis de Fourier, normalmente se utiliza la Transformada de Fourier, que sirve para funciones definidas por valores. Igual que la Serie de Fourier en su forma compleja, la Transformada de Fourier es una transformada de carácter complejo. De forma similar que la Serie de Fourier, la Transformada de Fourier en su forma discreta se trata de una transformación lineal:

$$f: \mathbb{C}^N \rightarrow \mathbb{C}^N$$

Donde $\mathbb{C}$ denota el conjunto de números complejos. Entonces, la transformada discreta de Fourier (DFT), transforma una secuencia de señales separadas de manera uniforme (los valores de nuestra función) $x_n \coloneqq x_0, x_1, \dots, x_{N-1}$ a coeficientes de naturaleza compleja (similares a c_n) $X_k \coloneqq X_0, X_1, \dots, X_{N-1}$, que incluyen información de tanto de amplitud como de fase de los vectores. Entonces, la DFT transforma una señal (equivalente a una función) en el dominio del tiempo a dominio de frecuencia. La transformada discreta de Fourier se define como [11] [12]:

$$\boxed{X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi kn}{N}i} = \sum_{n=0}^{N-1} x_n \left[\cos\left(\frac{2\pi kn}{N}\right) - i \sin\left(\frac{2\pi kn}{N}\right)\right]}$$

$$N = número\ de\ muestras$$

$$n = muestra\ actual$$

$$k = frecuencia\ actual\ k \in [0, N-1]$$

$$x_n = valor\ de\ la\ señal\ en\ la\ muestra\ n$$

$$X_k = coeficientes\ de\ la\ DFT$$

La transformada de Fourier, a veces se puede denotar con la $\mathcal{F}$, así que $X = \mathcal{F}\{x\}$. Así mismo, existe otra transformada de Fourier conocida como la transformada inversa (IDFT). Esta hace lo contrario que la DFT, usa los mismos coeficientes que en la DFT para obtener la función inicial.

$$\boxed{x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k \cdot e^{\frac{2\pi kn}{N}i} = \frac{1}{N}\sum_{k=0}^{N-1} X_k \cdot \left[\cos\left(\frac{2\pi kn}{N}\right) + i \sin\left(\frac{2\pi kn}{N}\right)\right]}$$

Si aplicamos una DFT a una señal y después usas los coeficientes y se realiza una IDFT, obtendremos la misma señal que el inicio.

4. APLICACIONES

La transformada discreta de Fourier (DFT) se aplica ampliamente al procesamiento de datos discretos en ciencia e ingeniería. Por ejemplo, algunas aplicaciones de las transformadas y series de Fourier en el mundo real son [13] [14]:

1. **Procesamiento de señales:** Puede que sea la mejor aplicación del análisis de Fourier. Se usa en casi todo dispositivo electrónico que recibe señales.
2. **Teoría de la aproximación:** Empleamos series de Fourier para escribir una función como un polinomio trigonométrico.
3. **Teoría del control**: La serie de funciones de Fourier en la ecuación diferencial a menudo proporciona alguna predicción sobre el comportamiento de la solución de la ecuación diferencial. Son útiles para conocer la dinámica de la solución.
4. **Ecuaciones diferenciales parciales**: Se utilizan para resolver ecuaciones diferenciales parciales de orden superior mediante el método de separación de variables.

La lista es casi infinita porque las transformadas de Fourier son presentes en todo dispositivo electrónico que se utiliza hoy en día. Esto es debido a que cuando trabajamos con señales, la transformada de Fourier es imprescindible.

4.1. Computación

En este apartado vamos a usar la Transformada de Fourier para comprimir un tipo de archivo que describe un objeto 3D, un archivo PLY.

4.1.1. Archivo PLY

Un archivo PLY describe un modelo 3D a partir de su representación como un polígono. Entonces, son necesarios los vértices y sus distintas caras [15]. Consiste en una lista de puntos (x, y, z) que describen los vértices y una lista de sus caras. También se pueden crear y adjuntar nuevas propiedades a los elementos del polígono, algunos ejemplos son:

- Textura
- Color
- Transparencia
- Tango de confianza de datos
- Propiedades para el anverso y el reverso de un polígono

Se puede formato binario, así como con texto ASCII, nosotros usaremos archivos (.ply) en formato ASCII.

La estructura de tal archivo es la siguiente:

Encabezado de archivo: Está al inicio del archivo y describe todo el archivo. Empieza con la palabra: "ply", para reconocer el formato ply y termina con la palabra: "end_header". En el encabezamiento se encuentran todos los elementos que se van a describir (vértices, caras, textura, color …) y su cantidad. Un ejemplo es el siguiente:

element vertex 8	{*define el elemento* vertex, *y el número que hay*}
property float x	{"vertex" *tiene una propiedad "x" de tipo "float"*}
property float y	{"vertex" *tiene una propiedad "y" de tipo "float"*}
property float z	{"vertex" *tiene una propiedad "z" de tipo "float"*}

En un archivo PLY siempre se describen dos características, "element vertex" y "element face", los vértices y caras del polígono.

4.1.2. Computación en 2D y 3D

Antes de tratar con objetos en 3D, averiguaremos como tratar con un dibujo en 2D. Ya que el trabajo está orientado en objetos 3D, usaremos como ejemplo contenido de fuente no propia [16].

En un principio, queremos obtener la transformada de Fourier de un dibujo. Para esto consideremos que nuestro dibujo es un recorrido cerrado de coordenadas (x, y) (Figura 4).

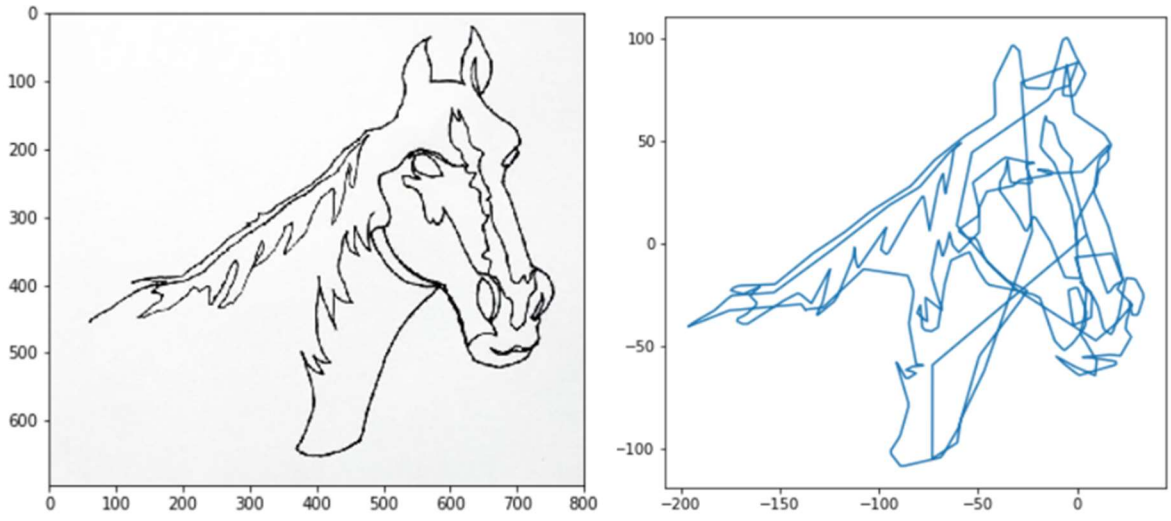


Figura 4: Dibujo expresado en coordenadas de puntos (x, y) y su recorrido cerrado respectivo

Seguidamente, podemos parametrizar el recorrido de cada una de las coordenadas de nuestro dibujo en función del tiempo (Figura 5). Así, obtenemos que si dibujo inicial es: $f(t)$, este compuesto de dos funciones: $f(x) = (x(t),\ y(t))$. Así obtenemos dos funciones de muestras que están separadas de manera uniforme, a las que podemos aplicar una DFT. De esta manera hemos obtenido la transformada de Fourier de nuestro dibujo inicial. Es decir, si realizamos una transformada Inversa, a las dos funciones transformadas, obtenemos nuestro dibujo inicial.

Figura 5: Parametrización de las funciones f(x) y f(y)

Esto se puede mejorar, ya que si en vez de considerar $f(x)$ como la composición de dos funciones, considerando que: $f: \mathbb{C}^N \to \mathbb{C}^N$, se puede obtener directamente la DFT del dibujo. De esta manera, las funciones $x(t), y(t)$ se juntan y $x(t)$ pasa a ser la parte real de $f(t)$ y $y(t)$ la imaginaria.

En conclusión, considerando que el dibujo se sitúa en el plano complejo podemos usar sus coordenadas como función para obtener la transformada que queremos.

Ahora, en vez de descomponer $f(t)$ en 2 señales $(x(t), y(t))$, $f(t)$ se descompone en 3 señales $f(t) = (x(t), y(t), z(t))$. Teniendo en cuenta que sabemos realizar una transformada cuando $f(t)$ se descompone en dos señales podemos separar estas 3 señales en dos funciones descritas por 2 señales: XY(t) y YZ(t). De esta manera obtenemos 2 transformadas que describen nuestra función $f(t)$. (Figura 6)

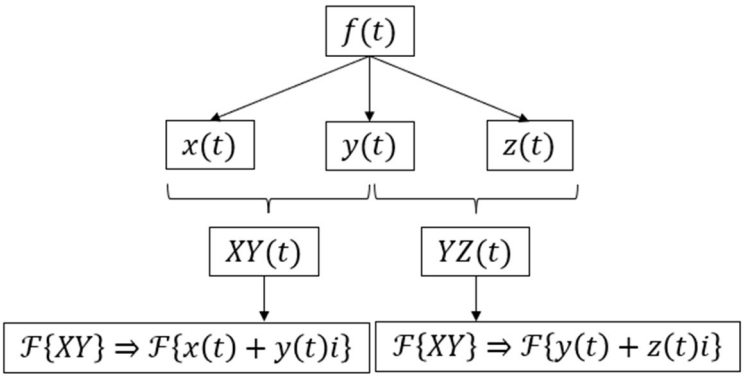


Figura 6: Esquema de la computación 3d de la serie de Fourier

4.1.3. Compresión de un archivo PLY

Como hemos dicho durante todo el trabajo, la transformada de Fourier transforma una función del dominio temporal al dominio de frecuencia. (Figura 7). Esto la hace descomponiendo la función inicial en diversas ondas senoidales con una frecuencia, fase y amplitud determinada.

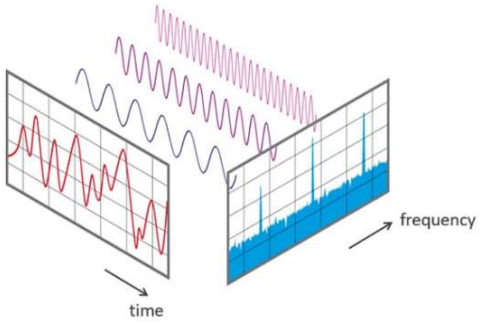


Figura 7: Representación gráfica del domino de frecuencia i el domino temporal

En consecuencia, para poder comprimir nuestros modelos 3D, primero hemos de averiguar cómo obtener la amplitud y fase de nuestro coeficiente de Fourier X_k

obtenido con la DFT (la frecuencia es la k). Ya que el coeficiente de Fourier está en forma binómica ($a + bi$), para obtener lo que buscamos, necesitamos pasar los coeficientes a forma polar.

$$X_n = a + bi \Rightarrow fase = \alpha = \tan^{-1}\left(\frac{b}{a}\right) \; y \; amplitud = |X_n| = \sqrt{a^2 + b^2}$$

Así entonces después de aplicar la transformada de Fourier Discreta y pasar los coeficientes a forma polar, obtenemos la amplitud, frecuencia y fase. Como más grande es la amplitud de una onda, más afecto tiene sobre la transformada inversa. Así mismo, si eliminamos las amplitudes más bajas (Figura 8), y realizamos la transformada inversa, obtenemos nuestra función inicial, pero con valores un poco distorsionados. En esto se basa nuestro proceso de compresión.

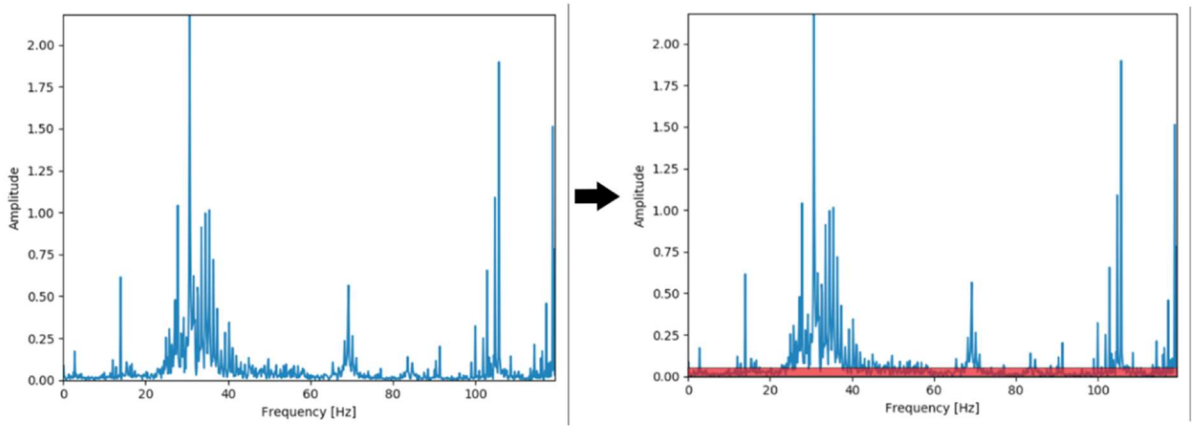


Figure 8: Representación del proceso de compresión eliminando la frecuencia de menor amplitud

Entonces, nuestro proceso de compresión consta de los pasos que se pueden observar en la Figura 9:

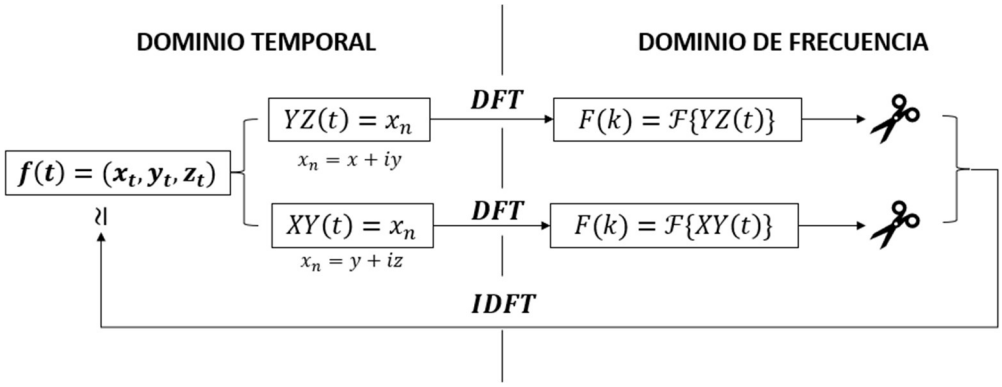


Figura 9: Esquema de todo el proceso de compresión

Para ver como de parecidos son los diferentes modelos que obtenemos usaremos la siguiente fórmula. Sean $\{(x_i, y_i, z_i): i = 1, \dots n\}$ una colección de puntos. Sea $(x, y, z) = \frac{1}{n}\sum_{i=1}^{n}(x_i, y_i, z_i)$, la desviación estándar de todos los puntos és:

$$\sigma^2 = \frac{\sum_{i=1}^{n}((x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2)}{n}$$

Considerando (x, y, z) como las coordenadas de la señal inicial y (x_F, y_F, z_F) las coordenadas obtenidas después de la compresión, obtenemos que la desviación estándar en un instante t es:

$$\sigma_t^2 = \frac{(x - x_F)^2 + (y - y_F)^2 + (z - z_F)^2}{4}$$

Para calcular la diferencia entre todos los puntos de ambos modelos se hace la mediana aritmética de todas las desviaciones estándares.

$$\sigma_{F(x)} = \frac{\sum \sigma_i}{n}$$

4.2. Resultados y análisis de la compresión

En este apartado realizaremos la compresión haciendo uso de 3 programas informáticos que se encuentran en el Anexo (DFT, IDFT, ERROR). Asimismo, todos los modelos que analizaremos son de las siguientes dos fuentes:

[17] "PLY Files an ASCII Polygon Format" [última consulta: 22/02/2022]

web: https://people.sc.fsu.edu/~jburkardt/data/ply/ply.html

[18] "3D Model Formats" [última consulta: 22/02/2022]

web: graphics.im.ntu.edu.tw/~robin/courses/cg03/model/

En el archivo generado después de la compresión se guarda el coeficiente en su forma polar. Entonces, pasamos de haber de guardar en el archivo PLY las tres coordenadas de todos los vértices a dos coeficientes (de naturaleza compleja) y sus frecuencias (6 números). Pues, en un principio la compresión no es favorable, no obstante, podemos eliminar esos coeficientes con amplitud más alta, obteniendo una aproximación de nuestra función inicial. En las tablas que siguen, se puede observar el espacio que ocupa todo eso que no son coordenadas cuando se eliminan todos los coeficientes (100% en las tablas) (no se puede calcular la desviación porque no hay figura). Seguidamente, se puede observar la compresión de cuatro modelos: Turbine.ply, Bunny.ply, Tenis_shoe.ply y Teapot.ply. Las imágenes de la compresión se pueden parecía en los anexos. El espacio que ocupa el modelo se puede apreciar en el título de las tablas.

TABLA 1: Compresión de Turbine.ply (220KB)

% de X_n descartados	Memoria Ocupada (KB)	Desviación estándar
0	264	0
10	248	1.834×10^{-3}
20	231	3.742×10^{-3}
30	215	5.740×10^{-3}
40	198	7.844×10^{-3}
50	182	1.834×10^{-3}
60	166	1.225×10^{-2}
70	150	1.498×10^{-2}
80	135	1.983×10^{-2}
90	119	3.205×10^{-2}
100	104	

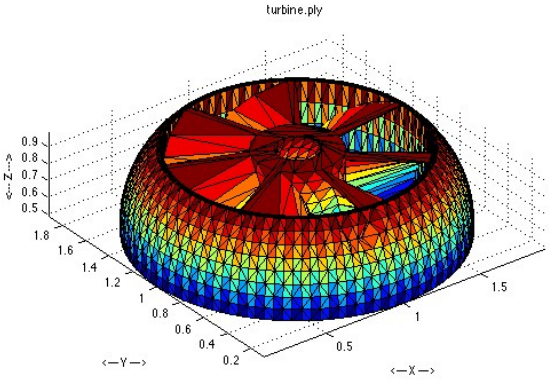


Figura 10: Imagen de Matlab de Turbine.ply

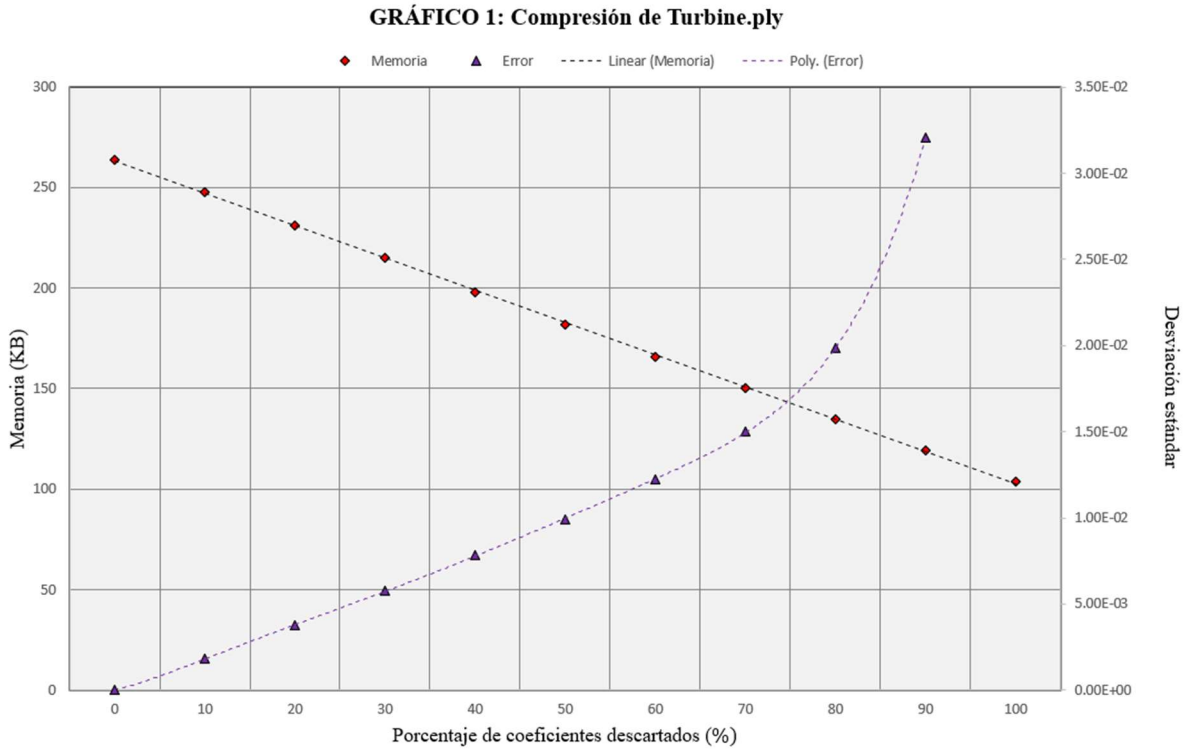


TABLA 2: Compresión de Shoe.ply (125KB)

% de X_n descartados	Memoria Ocupada (KB)	Desviación estándar
0	140	0
10	131	4.330×10^{-3}
20	122	8.772×10^{-3}
30	114	1.363×10^{-2}
40	105	1.857×10^{-2}
50	96	2.370×10^{-2}
60	88	2.894×10^{-2}
70	79	3.433×10^{-2}
80	71	4.148×10^{-2}
90	63	5.579×10^{-2}
100	55	

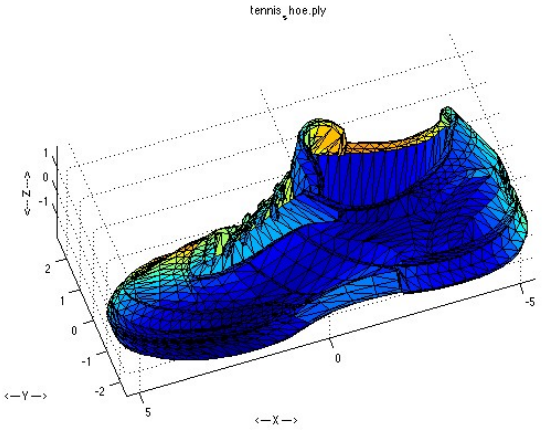


Figura 11: Imagen de Matlab de Shoe.ply

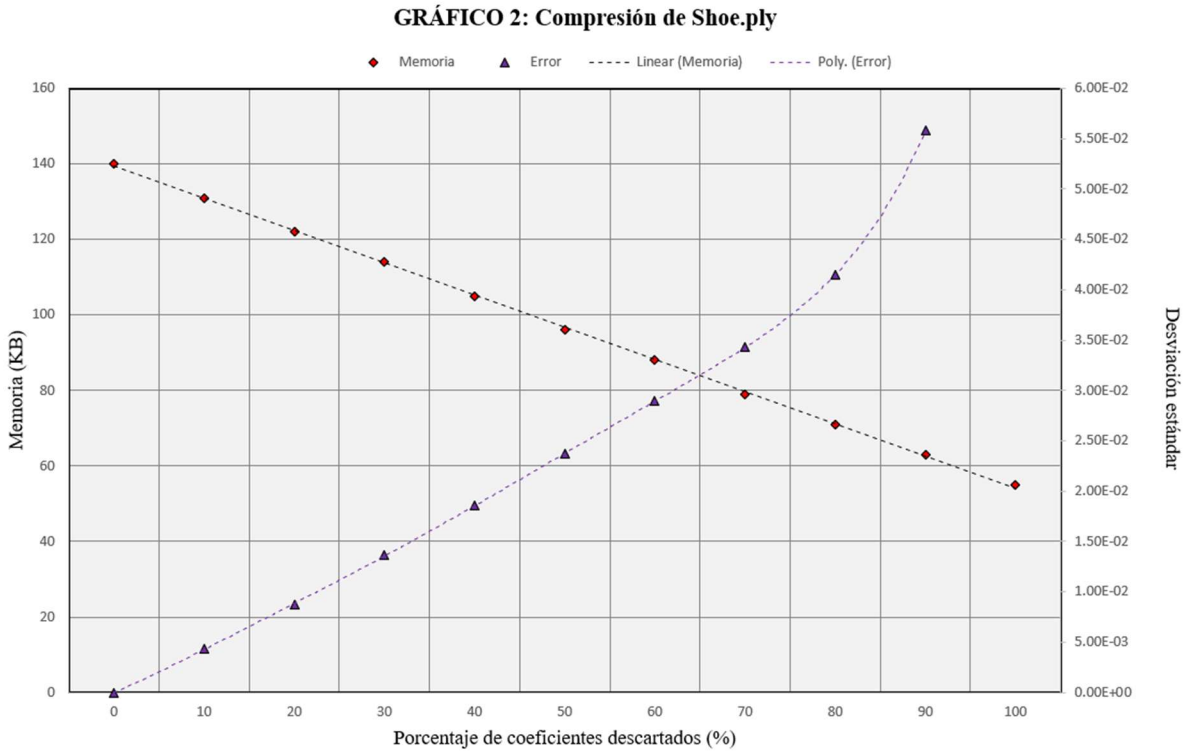


TABLA 3: Compresión de Teapot.ply (77 KB)

% de X_n descartados	Memoria Ocupada (KB)	Desviación estándar
0	86	0
10	80	1.834×10^{-3}
20	75	3.742×10^{-3}
30	70	5.740×10^{-3}
40	64	7.844×10^{-3}
50	59	1.834×10^{-3}
60	53	1.225×10^{-2}
70	48	1.498×10^{-2}
80	42	1.983×10^{-2}
90	37	3.205×10^{-2}
100	32	

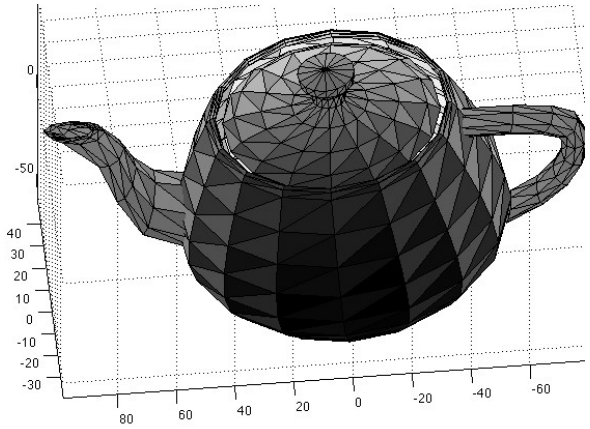


Figura 12: Imagen de Matlab de Teapot.ply

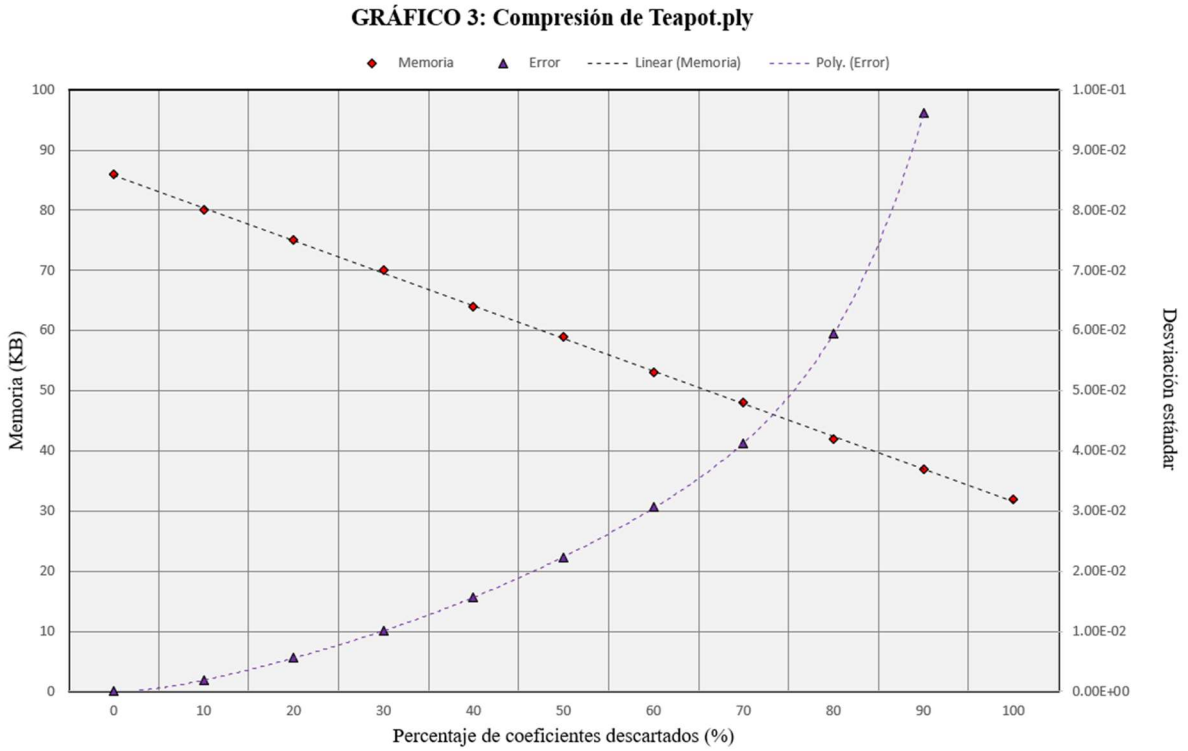
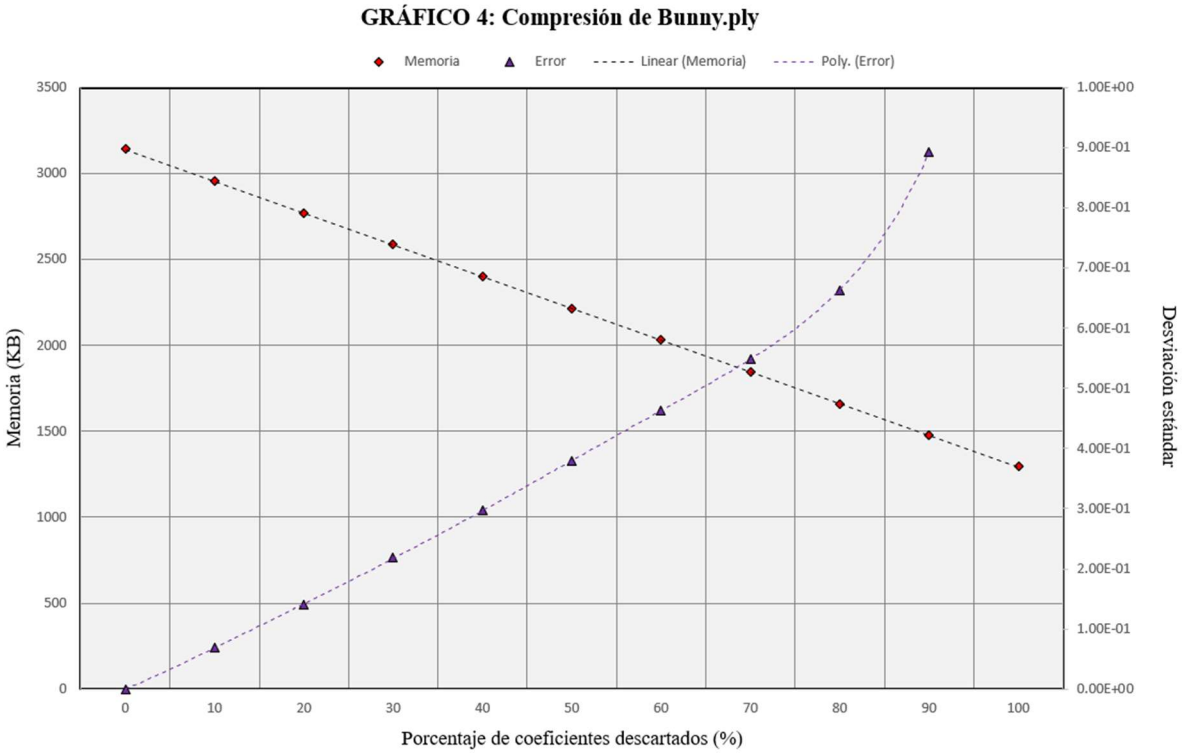


TABLA 4: Compresión de Bunny.ply (2576 KB)

% de X_n descartados	Memoria Ocupada (KB)	Desviación estándar
0	3142	0
10	2956	6.930×10^{-2}
20	2771	1.402×10^{-1}
30	2586	2.181×10^{-1}
40	2400	2.971×10^{-1}
50	2215	3.792×10^{-1}
60	2030	4.630×10^{-1}
70	1845	5.394×10^{-1}
80	1660	6.637×10^{-1}
90	1476	8.927×10^{-1}
100	1295	

Figura 13: Imagen del modelo Bunny.ply



Aunque se ha demostrado que se puede hacer esta compresión, no creo que sea una forma efectiva de realizar una compresión, ya que encuentro que la precisión que se pierde es mucho más valiosa que los kilobytes que se comprimen. También podemos observar que la compresión carece cuando la cantidad de puntos es muy grande. Además, los métodos tradicionales (como el ZIP) tienen un mucho mejor rendimiento (Tabla 5). No obstante, puesto que se pueden combinar ambos métodos, sigue siendo una manera de compresión válida en casos extremos.

TABLA 5: Comparación con métodos tradicionales

Modelo	Memoria Inicial (KB)	Memoria ZIP (KB)
Turbine.ply	220	53
Shoe.ply	125	39
Bunny.ply	2576	886
Teapot.ply	77	16

5. CONCLUSIÓN

Con esto llegamos al final de la Monografía y, al cabo de todos estos meses puedo afirmar que con mi trabajo he conseguido llegar a todos los objetivos que me propuse.

El camino no ha sido nada fácil, durante el trabajo me he encontrado con problemas constantes que he intentado superar al mejor de mis habilidades. Aun así, muchos de estos errores me han ayudado a crear un trabajo del cual estoy orgulloso y a terminar de comprender el concepto del análisis de Fourier.

No obstante, creo que aún hay lugar de mejora en mi trabajo. En un principio, en el archivo comprimido, no sería necesario guardar todas las frecuencias. En vez de ordenar los coeficientes por amplitud en el archivo comprimido, se podrían ordenar por frecuencia y eliminar esos con una amplitud más pequeña. De esta forma, podríamos prescindir de 2 de los 6 números que guardamos por vértice. También, se podría encontrar una manera de ordenar la señal (coordenadas de los vértices) de manera que las frecuencias más altas sean negligibles, y en vez de recortar por amplitud, recortar por frecuencia. Ordenar las coordenadas haría que la señal obtenida de este archivo sea más suave, y, en consecuencia, hacer que las frecuencias más altas sean más negligibles. Esto es muy parecido a lo que pasa con las series de Fourier. Cuando más términos de sumatorio uses, más aproximada será la función, porque se usan las frecuencias más altas (Esto se puede observar en el ejemplo realizado en el trabajo (Figura 4)). Para solucionar esto creo que se pudiese haber aplicado una variación en 3D de un algoritmo conocido como TSP (Traveling Salesman Problem), en el cual se calcula el camino óptimo que se puede hacer dados un conjunto de puntos en el espacio. Modificando este algoritmo para trabajar en puntos 3D y crear un gráfico con las arestas de los polígonos descritos hubiese sido una buena solución para obtener una señal más suave. El problema con este algoritmo es que la forma más optimizada de este algoritmo tiene complexidad (número de operaciones que ha de ejecutar el ordenador) $O(n^2 \cdot 2^n)$ con una cantidad grande de puntos es imposible computar tal algoritmo. Otra propuesta de mejora que me hago es encontrar una mejor manera de cuantificar el

error entre el archivo comprimido y el original, ya que la fórmula que presentada hace una aproximación muy pagana y no nos da un valor realmente significativo. No obstante, todo esto, estoy contento del trabajo e investigación que he hecho.

6. BIBLIOGRAFÍA Y WEBGRAFÍA

[1] "Pure Fourier series animation montage" *3Blue1Brown,* 2020.

Web: https://www.youtube.com/watch?v=-qgreAUpPwM&t=89 [Última consulta: 22/02/2022]

[2] "Periocidad de una función". *La Guía,* 2013.

Web: https://matematica.laguia2000.com/general/periodicidad-de-una-funcion [Última consulta: 30/9/2021]

[3] "Plano Complejo". *Wikipedia,* 2021.

Web: https://es.wikipedia.org/wiki/Plano_complejo [Última consulta: 30/9/2021]

[4] "Fórmula e identidad de Euler" Khan Academy, 2022.

Web: https://es.khanacademy.org/math/ap-calculus-bc/bc-series-new/bc-10-14/v/euler-s-formula-and-euler-s-identity [Última consulta: 22/02/2022]

[5] "Serie de Fourier". *Wikipedia,* 2021.

Web: https://es.wikipedia.org/wiki/Serie_de_Fourier [Última consulta: 30/9/2021]

[6] "Series y Transformadas de Fourier". *Universitat Politécnica de Cartagena,* 2016.

Web:www.dmae.upct.es/~paredes/am_ti/apuntes/Tema%202.%20Series%20y%20transformadas%20de%20Fourier.pdf [Última consulta: 30/9/2021]

[7] "Historia de la Serie de Fourier". *Wikipedia,* 2021.

Web: https://en.wikipedia.org/wiki/Fourier_series#History [Última consulta: 30/9/2021]

[8] "Serie de Fourier". *Wikipedia,* 2021.

Web: https://es.wikipedia.org/wiki/Serie_de_Fourier [Última consulta: 30/9/2021]

[9] "Series de Fourier #3 | Ejercicio resuelto 1.1" *FísicayMates,* 2004.

Web:

https://drive.google.com/file/d/1bCzcntldbCzOcCZBamSVgdPNXIkJAHZB/view

[10] "Rendering 2D and 3D bodies with Fourier Transforms". *Shultz, C.* 2016.

Trabajo: https://app.box.com/s/thpam69f04u5ygntpnemzxtme8hcig1b [Última consulta: 30/9/2021]

[11] "Transformada de Fourier". *Wikipedia,* 2021.

Web: https://en.wikipedia.org/wiki/Discrete_Fourier_transform [Última consulta: 30/9/2021]

[12] "Applications of the DFT". *Science Direct,* 2021.

Web: www.sciencedirect.com/topics/engineering/discrete-fourier-transform [Última consulta: 30/9/2021]

[13] "Application of DFt to electronic mesurements". *ResearchGate,* 1997.

Web:https://researchgate.net/publication/3730641_Application_of_discrete_Fourier_transform_to_electronic_measurements [Última consulta: 30/9/2021]

[14] "Real world aplications of Foureir seires". *StackExchange,* 2013.

Web: https://math.stackexchange.com/questions/579453/real-world-application-of-fourier-series [Última consulta: 30/9/2021]

[15] "What is a PLY file?". *Fileformat,* 2012.

Web: https://docs.fileformat.com/3d/ply/ [Última consulta: 30/9/2021]

[16] "Rendering 3D objects with epicycles". *StackOverflow,* 2019.

Web: https://stackoverflow.com/questions/57117182/drawing-rendering-3d-objects-with-epicycles-and-fourier-transformations-animati/57127276 [Última consulta: 30/9/2021]

[17] “PLY Files an ASCII Polygon Format” John Burkardt 2012 [última consulta: 22/02/2022]

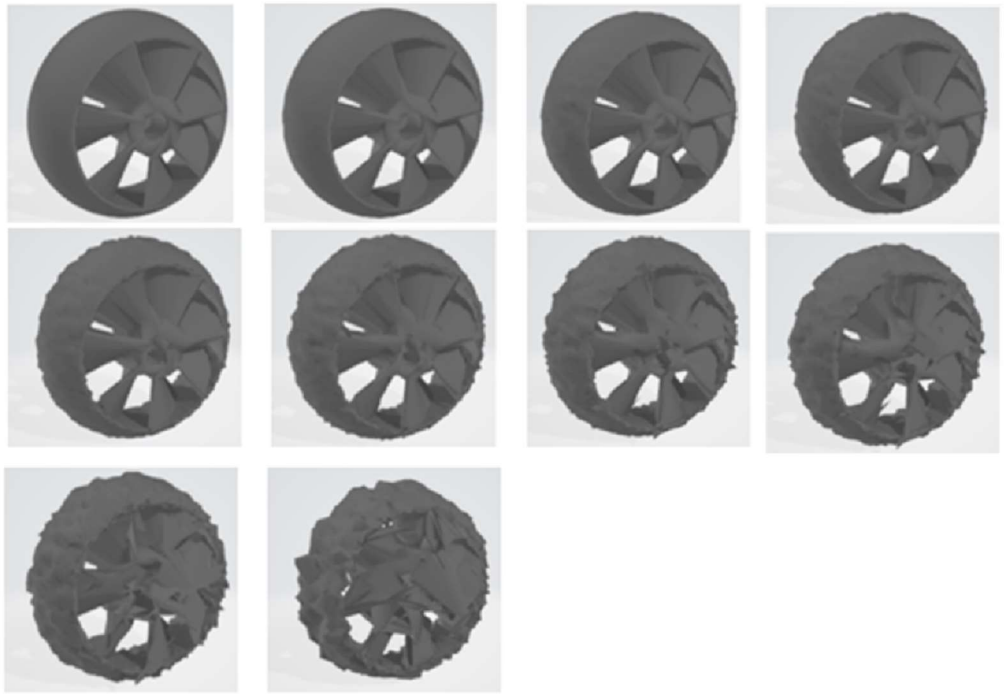
Web: https://people.sc.fsu.edu/~jburkardt/data/ply/ply.html

[18] “3D Model Formats” Princeton University 2005 [última consulta: 22/02/2022]

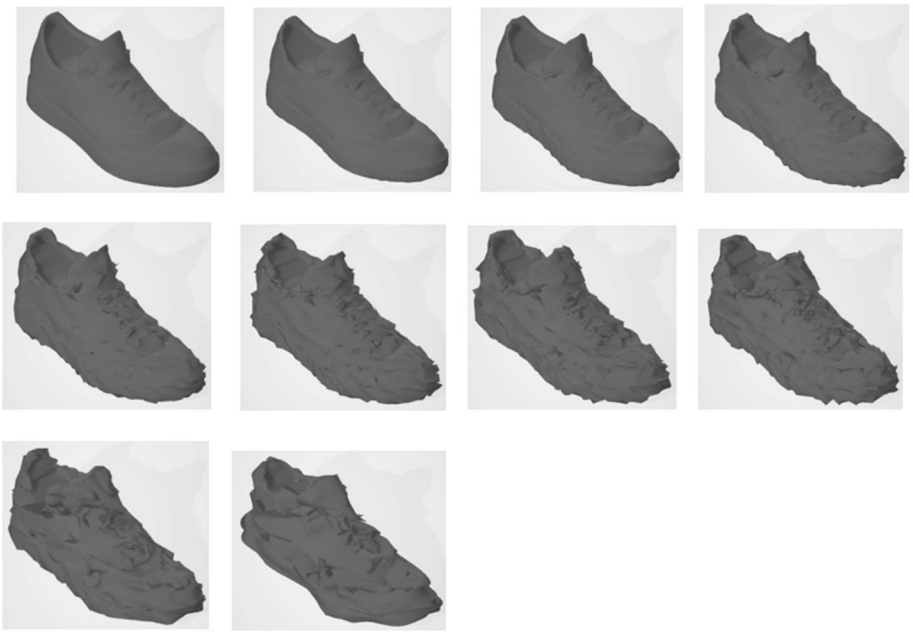
Web: graphics.im.ntu.edu.tw/~robin/courses/cg03/model/

7. ANEXO

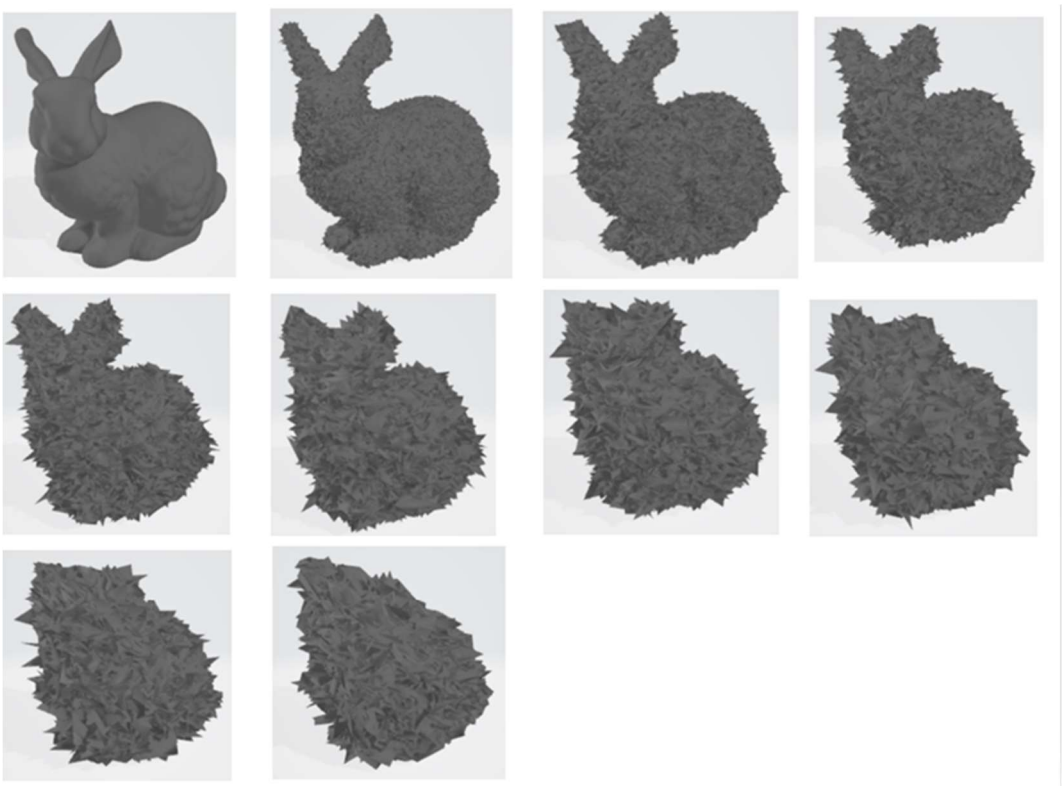
7.1. Compresión: Turbine.ply



7.2. Compresión: Shoe.ply



7.3. Compresión: Bunny.ply



7.4. Compresión: Teapot.ply

7.5. Programa 1: DFT

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  #include <bits/stdc++.h>
5  using namespace std;
6
7  #define real first
8  #define img second
9  #define amp first.first
10 #define phase first.second
11
12 int elementVertex;
13 int elementFace;
14
15
16 pair<double, double> Complex(double re, double im){
17     pair<double, double> ComplexNumber;
18     ComplexNumber = {re, im};
19     return ComplexNumber;
20 }
21
22 pair<double, double> ComplexAdd(pair<double, double>num1,
23 pair<double, double>num2){
24     pair<double, double> result;
25     result.real = num1.real + num2.real;
26     result.img = num1.img + num2.img;
27     return result;
28 }
29
30 pair<double, double> ComplexProduct(pair<double, double>num1,
31 pair<double, double>num2){
32     pair<double, double> result;
33     result.real = num1.real * num2.real - num1.img * num2.img;
34     result.img = num1.real * num2.img + num1.img * num2.real;
35     return result;
36 }
37
```

```
38
39 vector<pair<pair<double, double>, int>>dft(vector<pair<double,
40 double>>f){
41     vector<pair<pair<double, double>, int>>X(elementVertex);
42     pair<double, double>sum;
43
44     for(int k = 0; k < elementVertex; ++k){
45         sum = Complex(0.0, 0.0);
46         for(int n = 0; n < elementVertex; ++n){
47             double w = (M_PI * 2 * k * n)/elementVertex;
48             pair<double, double> phi = Complex(cos(w), -sin(w));
49             sum = ComplexAdd(sum, ComplexProduct(f[n], phi));
50         }
51
52         sum.real /= elementVertex;
53         sum.img /= elementVertex;
54
55
56         X[k].second = k;
57         X[k].amp = sqrt(sum.real * sum.real + sum.img * sum.img);;
58         X[k].phase = atan2(sum.img, sum.real);;
59     }
60
61     return X;
62 }
63
64 int main() {
65     string name;
66     cin >> name;
67     ifstream file(name);
68     string word;
69
70
71     while(file >> word) {
72         if(word == "vertex"){
73             file >> elementVertex;
74             cout << elementVertex << endl;
75         }
```

```
76      if(word == "face"){
77          file >> elementFace;
78          cout << elementFace << endl;
79      }
80      if(word == "end_header") break;
81  }
82
83
84  vector<double>x(elementVertex);
85  vector<double>y(elementVertex);
86  vector<double>z(elementVertex);
87
88  for(int i = 0; i < elementVertex; ++i){
89      file >> x[i] >> y[i] >> z[i];
90  }
91
92  vector<vector<int>>face(elementFace);
93  int size, inp;
94  for(int i = 0; i < elementFace; ++i){
95      file >> size;
96      face[i].push_back(size);
97      for(int r = 0; r < size; ++r){
98          file >> inp;
99          face[i].push_back(inp);
100     }
101 }
102
103
104 vector<pair<double, double>>complexSignalXY(elementVertex);
105
106 for(int i = 0; i < elementVertex; ++i){
107     complexSignalXY[i].real = x[i];
108     complexSignalXY[i].img = y[i];
109 }
110
111 vector<pair<pair<double, double>, int>>dftXY;
112 dftXY = dft(complexSignalXY);
113
```

```
114
115     vector<pair<double, double>>complexSignalYZ(elementVertex);
116     for(int i = 0; i < elementVertex; ++i){
117         complexSignalYZ[i].real = y[i];
118         complexSignalYZ[i].img = z[i];
119     }
120
121     vector<pair<pair<double, double>, int>>dftYZ;
122     dftYZ = dft(complexSignalYZ);
123
124
125 for(int num = 1; num <= 10; ++num){
126     int elementCoeficient = elementVertex*num/10;
127     string numS = to_string(num);
128     ofstream newFile("dft_"+numS+'_'+name);
129     newFile <<
130     "ply" << endl <<
131     "format ascii 1.0" << endl <<
132     "comment made by Pol de los Santos" << endl <<
133     "comment VCGLIB generated" << endl <<
134     "element vertex " << elementVertex  << endl <<
135     "element coeficient " << elementCoeficient << endl <<
136     "property float frequency" << endl <<
137     "property float aplitud" << endl <<
138     "property float phase" << endl <<
139     "element face " << elementFace << endl <<
140     "property list uchar int vertex_index" << endl <<
141     "end_header" << endl;
142
143
144     sort(begin(dftXY), end(dftXY));
145     reverse(begin(dftXY), end(dftXY));
146     sort(begin(dftYZ), end(dftYZ));
147     reverse(begin(dftYZ), end(dftYZ));
148
149
150     for(int i = 0; i < elementCoeficient ; ++i){
151
```

```
152         newFile << dftXY[i].second << ' ' << dftXY[i].amp << ' ' <<
153 dftXY[i].phase << endl;
154     }
155     for(int i = 0; i < elementCoeficient ; ++i){
156         newFile << dftYZ[i].second << ' ' << dftYZ[i].amp << ' ' <<
157 dftYZ[i].phase << endl;
158     }
159 
160     for(int i = 0; i < elementFace; ++i){
161         newFile << face[i][0];
162         for(int t = 1; t < face[i].size(); ++t){
163             newFile << ' ' << face[i][t];
164         }
165         newFile  << endl;
166     }
167 
168     newFile.close();
    }
        return 0;

    }
```

7.6. Programa 2: IDFT

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  #include <bits/stdc++.h>
5  using namespace std;
6
7  #define real first
8  #define img second
9  #define real first
10 #define img second
11 #define amp first.first
12 #define phase first.second
13 int elementVertex;
14 int elementFace;
15 int elementCoeficients;
16
17
18 vector<pair<double, double>> idft(vector<pair<pair<double, double>,
19 int>>plano_2D){
20     vector<pair<double, double>>f;
21     pair<double, double>sum;
22     double time = 0.0;
23
24     for(int n = 0; n < elementVertex; ++n){
25         sum = {0, 0};
26         for(int k = 0; k < elementCoeficients; ++k){
27             double freq = plano_2D[k].second;
28             double amplitud = plano_2D[k].amp;
29             double pha = plano_2D[k].phase;
30
31             double x = amplitud * cos(freq*time+pha);
32             double y = amplitud * sin(freq*time+pha);
33             sum.real+=x;
34             sum.img+=y;
35
36         }
37         double dt = (2*M_PI)/elementVertex ;
```

```
38          time += dt;
39          f.push_back(sum);
40
41      }
42
43      return f;
44  }
45
46  int main() {
47      string name;
48      cin >> name;
49      ifstream file(name);
50      string word;
51
52
53      while(file >> word) {
54          if(word == "vertex"){
55              file >> elementVertex;
56              cout << elementVertex << 'y' << endl;
57          }
58          if(word == "coeficient"){
59              file >> elementCoeficients;
60              cout << elementCoeficients << endl;
61          }
62          if(word == "face"){
63              file >> elementFace;
64              cout << elementFace << endl;
65          }
66          if(word == "end_header") break;
67      }
68
69
70      vector<pair<pair<double, double>, int>>XY(elementCoeficients);
71      vector<pair<pair<double, double>, int>>YZ(elementCoeficients);
72
73      for(int i = 0; i < elementCoeficients; ++i){
74          file >> XY[i].second >> XY[i].amp >> XY[i].phase;
75      }
```

```
76    for(int i = 0; i < elementCoeficients; ++i){
77       file >> YZ[i].second >> YZ[i].amp >> YZ[i].phase;
78    }
79
80    vector<vector<int>>face(elementFace);
81    int size, inp;
82    for(int i = 0; i < elementFace; ++i){
83       file >> size;
84       face[i].push_back(size);
85       for(int r = 0; r < size; ++r){
86          file >> inp;
87          face[i].push_back(inp);
88       }
89    }
90
91
92    vector<double>x(elementVertex);
93    vector<double>y1(elementVertex);
94    vector<double>y2(elementVertex);
95    vector<double>z(elementVertex);
96
97    vector<pair<double, double>>XY_pair(elementVertex);
98    XY_pair = idft(XY);
99    for(int i = 0; i < elementVertex; ++i){
100      x[i] = XY_pair[i].real;
101      y1[i] = XY_pair[i].img;
102   }
103
104
105
106   vector<pair<double, double>>YZ_pair(elementVertex);
107   YZ_pair = idft(YZ);
108   for(int i = 0; i < elementVertex; ++i){
109      y2[i] = YZ_pair[i].real;
110      z[i] = YZ_pair[i].img;
111   }
112
113
```

```
114    string num;
115    num.push_back(name[4]);
116
117    ofstream newFile("idft"+num+".txt");
118    newFile <<
119    "ply" << endl <<
120    "format ascii 1.0" << endl <<
121    "comment made by Pol de los Santos" << endl <<
122    "comment VCGLIB generated" << endl <<
123    "element vertex " << elementVertex  << endl <<
124    "property float x" << endl <<
125    "property float y" << endl <<
126    "property float z" << endl <<
127    "element face " << elementFace << endl <<
128    "property list uchar int vertex_index" << endl <<
129    "end_header" << endl;
130
131
132
133    for(int i = 0; i < elementVertex ; ++i){
134        newFile << x[i] << ' ' << (y1[i] + y2[i])/2 << ' ' << z[i] << endl;
135    }
136
137    for(int i = 0; i < elementFace; ++i){
138        newFile << face[i][0];
139        for(int t = 1; t < face[i].size(); ++t){
140            newFile << ' ' << face[i][t];
141        }
142        newFile  << endl;
143    }
144
145    newFile.close();
146    return 0;
147 }
```

7.6. Programa 3: ERROR

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  #include <bits/stdc++.h>
5  using namespace std;
6
7  int elementVertex;
8  string word;
9
10 int main() {
11     string name1;
12     cin >> name1;
13     ifstream file1(name1);
14     while(file1 >> word) {
15         if(word == "vertex"){
16             file1 >> elementVertex;
17         }
18         if(word == "end_header") break;
19     }
20
21     vector<double>x1(elementVertex);
22     vector<double>y1(elementVertex);
23     vector<double>z1(elementVertex);
24     double Mx = 0, My = 0, Mz = 0;
25
26     for(int i = 0; i < elementVertex; ++i){
27         file1 >> x1[i] >> y1[i] >> z1[i];
28         Mx += x1[i]; My += y1[i]; Mz += z1[i];
29     }
30     Mx /= elementVertex;
31     My /= elementVertex;
32     Mz /= elementVertex;
33
34     double Fd=0;
35
36     for(int i = 0; i < elementVertex; ++i){
37         double x = (Mx-z1[i]);
```

```
38        double y = (My-y1[i]);
39        double z = (Mz-z1[i]);
40
41        if(Fd < sqrt(x*x+y*y+z*z)) Fd = sqrt(x*x+y*y+z*z);
42    }
43
44    string name2;
45    cin >> name2;
46    ifstream file2(name2);
47
48    vector<double>x2(elementVertex);
49    vector<double>y2(elementVertex);
50    vector<double>z2(elementVertex);
51
52    while(file2 >> word) {
53        if(word == "end_header") break;
54    }
55    for(int i = 0; i < elementVertex; ++i){
56        file2 >> x2[i] >> y2[i] >> z2[i];
57    }
58
59    double sum =0;
60    for(int i = 0; i < elementVertex; ++i){
61        double x = (x2[i]-x1[i]);
62        double y = (y2[i]-y1[i]);
63        double z = (z2[i]-z1[i]);
64        sum += sqrt((x*x+y*y+z*z)/4);
65    }
66
67    sum /= elementVertex;
68    cout << Fd << endl;
69    cout << sum/Fd << endl;
70    return 0;
71 }
```