Librería de DEC usando Quad-Edge para CGAL

Diego Arturo Bernal Melo

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
MAESTRÍA EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
BOGOTÁ, D.C.
2020

Librería de DEC usando Quad-Edge para CGAL

Autor:

Diego Arturo Bernal Melo

MEMORIA DEL TRABAJO DE GRADO REALIZADO PARA CUMPLIR UNO DE LOS REQUISITOS PARA OPTAR AL TITULO DE

MAGÍSTER EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Director

Leonardo Flórez Valencia

Comité de Evaluación del Trabajo de Grado

Cesar Julio Bustacara Medina

Luis Carlos Diaz Chaparro

Página web del Trabajo de Grado

https://livejaverianaedu.sharepoint.com/sites/Ingsis/TGMISC/203005

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
MAESTRÍA EN INGENIERIA DE SISTEMAS Y COMPUTACIÓN
BOGOTÁ, D.C.
Nov,2020

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
MAESTRÍA EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Rector Magnífico

Jorge Humberto Peláez, S.J.

Decano Facultad de Ingeniería

Ingeniero Lope Hugo Barrero Solano

Director Maestría en Ingeniería de Sistemas y Computación

Ingeniera Angela Carrillo Ramos

Director Departamento de Ingeniería de Sistemas

Ingeniero Efraín Ortíz Pabón

Artículo 23 de la Resolución No. 1 de Junio de 1946

"La Universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado. Sólo velará porque no se publique nada contrario al dogma y la moral católica y porque no contengan ataques o polémicas puramente personales. Antes bien, que se vean en ellos el anhelo de buscar la verdad y la Justicia"

AGRADECIMIENTOS

Quiero dar gracias a mi familia por este apoyo incondicional quienes siempre han estado presentes en todos mis procesos académicos y éste no ha sido la excepción. En especial quiero agradecer a mi esposa que, sin su apoyo moral, económico y espiritual, no habría sacado este proyecto adelante.

Contenido

ABSTRACT

This document explains the process to build a Quad-edge library that works together with a built DEC library, to give a powerful tool to simulate fluid systems using CGAL. Giving a summary of the experience earned in the process and a formal methodology to test the library. At the end of this document some conclusions of this process are given.

RESUMEN

Este documento explica el proceso para construir una librería de Quad-edge que trabaja junto con una librería de DEC construida, para dar a CGAL una poderosa herramienta de simulación sobre sistemas de fluidos. Dando un resumen de la experiencia adquirida en el proceso y una metodología formal para probar la librería. Al final de este documento son dadas algunas conclusiones de este proceso.

RESUMEN EJECUTIVO

Desde el principio de los tiempos el ser humano ha buscado entender el mundo que le rodea y para ello se ha valido de herramientas como las simulaciones. Esto ha mejorado el entendimiento de los fenómenos naturales, comportamientos sociales, comportamiento del mercado, entre otros. Gracias a querer encontrar la verdad mediante simulaciones se han generado una cantidad de aplicaciones que permiten predecir el comportamiento de sistemas antes de ser construidos, para saber de antemano los verdaderos recursos requeridos para su construcción y si realmente es posible crearlos, ahorrando dinero, tiempo y esfuerzo en diferentes áreas. Las simulaciones están descritas en modelos simplificados de la realidad a través de ecuaciones diferenciales parciales (PDE). Para solucionar las PDE se requieren de soluciones numéricas y por lo tanto de la ayuda de un computador. Diferentes métodos han sido inventados para resolver estas ecuaciones y realizar simulaciones mediante interacciones locales para afectar la estructura en general sin embargo estas no generan la suficiente precisión necesaria y por esta razón el método más usado para encontrar estas soluciones aproximadas es el conjunto de métodos de elementos finitos (FEM). Estos métodos son los más usados actualmente por los softwares de simulación por su eficiencia, precisión y estabilidad numérica en la mayoría de los casos. FEM son métodos que requieren de un alto esfuerzo en el momento en que se plantea el problema debido a la formulación de las condiciones de frontera, esto hace que FEM tenga más dificultad a la hora de implementarse. Como una nueva apuesta nace el cálculo exterior discreto (DEC) que resuelve las PDE desde un punto de vista diferente basado en el teorema de Stokes.

DEC resuelve las PDE mediante la construcción de operadores basados en combinatorias discretas y operaciones geométricas aplicadas sobre la frontera de una variedad. Haciendo uso del teorema de Stokes, DEC plantea resolver el problema en la frontera de la variedad para afectarla y por lo tanto afectar a la variedad en sí. DEC solo requiere de la malla y su dual para lograr construir los operadores, generando así la necesidad de tener una estructura de datos topológica que almacene tanto el primal como el dual de una malla. La mejor estructura se llama Quad-edge. Quad-edge permite tener el primal y el dual de una malla, resolviendo cualquier pregunta de adyacencia mediante combinaciones de dos operaciones. Esta estructura tiene la capacidad de lograr esas consultas en tiempo O(1) si la malla es especialmente una triangulación. La idea de DEC es la reescritura de las PDE en términos de operaciones discretas. De esta manera al ejecutar dichas operaciones los resultados estarán discretizados reemplazando la función de FEM en las simulaciones.

Las bases para hacer simulaciones es tener una librería que permita resolver las PDE. Esta investigación detalla la construcción de una librería capaz aplicar los operadores más importantes de DEC sobre una triangulación almacenada en un Quad-edge. Estas dos librerías serán las bases para en el futuro resolver las PDE con DEC usando Quad-edge. Para alcanzar esto la investigación se separó en tres fases: construcción de la estructura de datos Quad-edge, construcción de una librería de DEC y finalmente las pruebas. Para la primera fase, se realiza ingeniería inversa sobre la estructura Half-edge de CGAL para la reescritura en términos de su estructura hermana Quad-edge. Siguiendo los lineamientos y buenas prácticas de la librería CGAL construida en C++. Siendo ésta una librería de altos estándares en términos de geometría

computacional dota a la estructura de la calidad esperada en generalización, optimización de memoria y velocidad de ejecución, mediante el uso de diferentes patrones de diseño de software combinados. Al ser una librería basada en el lenguaje C++ liga la investigación al uso del lenguaje de programación en C++ por lo tanto en la segunda fase se vuelve a hacer uso de la técnica de ingeniería inversa, esta vez sobre la librería PyDEC construida en Python con una estructura de datos topológica diferente a Quad-Edge. La idea detrás es reescribir la librería PyDEC en el lenguaje C++ y diseñando los algoritmos que se reemplazarían en unas pocas líneas de código por el hecho de dotar la librería DEC con una estructura como Quad-edge. Esta característica es el resultado directo de la investigación. En la tercera fase se realizó un plan de pruebas que demuestra que la construcción de las dos librerías se encuentra en un estado aceptable para su uso. El plan de pruebas está más dirigido sobre la librería de Quad-edge debido a que esta librería es la esencia de esta investigación porque al construir bien los elementos de Quad-edge es posible calcular los operadores de DEC descritos en el diseño de la librería PyDEC. Al estar las dos librerías DEC y Quad-edge basadas en librerías ya existentes el resultado esperado es que las dos librerías estén basadas en buenas prácticas y que funcionen como extensión de las mismas.

Al finalizar estas fases de construcción se obtienen diferentes conclusiones ligadas al uso de la estrategia de ingeniería inversa y del resultado conseguido al unir estos dos conceptos. También hay conclusiones independientes por librería y del lenguaje de programación usado. El resultado más directo que esta investigación arrojó es que usar Quad-edge para resolver los principios de DEC es la forma más efectiva en términos de complejidad algorítmica y en el diseño de algoritmos, gracias a la velocidad y las capacidades que ofrece Quad-edge. Al parecer esta sinergia que tienen estos dos componentes es adquirida debido a que los dos conceptos están basados en el teorema de Stokes. Esta combinación tiene el potencial de ser un futuro reemplazo de FEM. Además de mejorar la estabilidad numérica son esos casos donde FEM no la tiene. Ofreciendo además una construcción más sencilla y con la ayuda de Quad-edge la velocidad requerida en las simulaciones.

INTRODUCCIÓN

La mayor motivación que genera este proyecto se postula como una pregunta: ¿Para qué simular?, la respuesta a esta pregunta es porque el ser humano busca entender los fenómenos físicos y naturales que le rodean y para esto se ha valido de diferentes herramientas, especialmente desde la llegada del computador. Esto se debe a que las simulaciones están basadas en modelos simplificados de la realidad descritos en términos de ecuaciones diferenciales parciales (PDE por sus siglas en inglés). Las soluciones de estas ecuaciones diferenciales no se consiguen mediante soluciones analíticas por lo que es requerido de soluciones numéricas para llegar a un resultado aproximado del comportamiento del fenómeno en cuestión. Para hacer simulaciones se han inventado diferentes técnicas que van desde mecánica molecular, mecánica Browniana,

autómatas celulares, métodos de Lattice Boltzmann y los métodos actuales más usados en la mayoría de software de simulaciones los métodos de elementos finitos (FEM por sus siglas en inglés). Cada uno de estos métodos tratan de solucionar los problemas en la simulación desde puntos de vista diferente en especial los métodos FEM, En pocas palabras FEM trata de aproximar la solución de las ecuaciones que Navier-Stokes por medio de subdivisiones de un complejo simplicial en elementos conocidos como cubos, esferas entre otros [1]. Estos métodos son eficientes, pero adquieren una complejidad alta cuando se quiere calcular la frontera del fluido. En estos casos el mayor esfuerzo que se hace es en algo llamado las condiciones de frontera. Estas condiciones tienen la posibilidad de inyectar errores de cálculo y de precisión donde no estén bien definida. Si las condiciones de frontera están bien definidas aún así existe la posibilidad en casos especiales de generar errores numéricos y hacer que una simulación de fluidos consuma más recursos de máquina de los que debería [2].

Como una nueva apuesta en el año 2003, Anil Nirmal Hirani inventa en su tesis doctoral un concepto llamado cálculo exterior discreto (DEC por sus siglas en inglés) [3]. Hirani pretende solucionar las PDE mediante combinatorias discretas y operaciones geométricas. Esto es posible debido a que este cálculo es basado en el teorema de Stokes. DEC plantea resolver el problema en la frontera de la variedad para afectarla y por lo tanto afectar a la variedad en sí [3]. Este aspecto es un punto de vista diferente debido a que FEM plantea todo lo contrario y por lo tanto su punto más complicado es cuando llega a la frontera. DEC ataca el problema directamente sobre la frontera en cuestión. Esto es inteligente porque la frontera además de heredar la orientación de la variedad también es de una dimensión menor. Esto hace que sea más sencillo de tratar al reducir la complejidad del problema en una dimensión. La idea detrás de este cálculo es reescribir las PDE en términos de operaciones discretas planteadas por Hirani. El problema es ejecutar esas operaciones. Hirani plantea que dichas operaciones son construcciones basadas únicamente en la malla y su dual. Por otro lado, DEC se construye a partir de una variedad dada esto implica una ventaja adicional frente a FEM porque no hay necesidad de construir elementos nuevos. Únicamente con la geometría es suficiente para construir todo el cálculo. Permite eliminar dichas condiciones de frontera porque se está calculando todo sobre la misma frontera. Al hacer las operaciones desde este punto de vista se está ofreciendo la posibilidad de corregir defectos en los FEM y especialmente la corrección de posibles errores numéricos [2]. Además, Hirani construye una librería escrita en Python junto con N. Bell [7] con las funcionalidades de DEC llamada PyDEC.

DEC requiere de una estructura para manipular la malla y su dual. Esto hace que sea necesario la búsqueda de una estructura capaz de responder a las necesidades de DEC, para ello se busca sobre la librería CGAL como la librería por excelencia en geometría computacional. Dentro de CGAL se encuentra la estructura Half-edge que a pesar de funcionar bien en otros aspectos, para DEC no es suficiente por lo que a partir de esta estructura se realiza una búsqueda llegando así a la estructura elegida llamada Quad-edge. Esta estructura está basada también en el teorema de Stokes si es usada sobre variedades orientables. Esto es una ventaja porque Quad-edge permite almacenar variedades orientables y no orientables dependiendo de su implementación. Se hace uso de complejos simpliciales hechos a partir de triángulos, la razón crucial para usar triangulaciones fue por su eficacia en las operaciones de adyacencia al ser la malla una triangulación [3], además de su versatilidad para ser extrapolada en otros polígonos, aunque perdiendo tal eficacia en las operaciones. Además DEC ofrece que esta malla no necesita estar condicionada, por lo tanto no es necesario tener una triangulación de Delaunay [19]. Quad-

edge es la topología que juega un papel importante en el uso de DEC debido a que esta estructura permite navegar por la triangulación y por su dual de forma rápida y simple, permitiendo que se puedan realizar los cálculos de DEC. DEC exige que se tenga información del primal y del dual para la construcción de sus operadores.

El producto final de esta investigación es el diseño de la librería de DEC usando Quad-Edge para CGAL, esto implica que todo el código sea compatible con CGAL. Para lograr esto se construye la estructura de datos Quad-edge de la misma manera que fue construida su estructura hermana Half-edge dentro de esta librería. Esta librería está construida en C++, permitiendo dotar a la estructura de toda la potencia de cómputo que ofrece este lenguaje. Adicionalmente mejorando la velocidad de cómputo usando una topología más eficiente en tiempos de procesamiento. Y con el diseño de la librería de DEC se expone el pseudocódigo de los operadores principales necesarios para construir la derivada discreta y la estrella de Hodge, gracias a Quad-edge estos pseudocódigos son sencillos y mucho más legibles que los que se encuentran construidos en PyDEC. Esta librería se encuentra en una cuenta de bitbucket en el siguiente enlace: https://bitbucket.org/dabm007/quadedgepuj/src/master/

1. DESCRIPCIÓN GENERAL

Oportunidad y problemática

FEM son métodos eficientes y por esta razón son los métodos más usados en software de simulación de fluidos. Pero los problemas más grandes que se ven en estos métodos son: estabilidad numérica [2], discretización y complejidad de implementación. DEC combinado con Quad-edge corrige estos problemas. Ofreciendo principalmente una forma de discretización distinta, casi al punto de que pareciera que no se hiciera el proceso de discretización.

DEC al estar construido bajo los principios del teorema de Stokes permite que las operaciones se hagan sobre la frontera del fluido y en este caso la frontera del fluido será una triangulación. Por medio del algebra de aristas que otorga Quad-edge se pueda aplicar DEC de una manera eficiente. Si bien esta investigación solo llega hasta la construcción de los operadores básicos de DEC se espera que se pueda usar esta librería en el futuro y otras investigaciones para la construcción de los operadores faltantes y así reescribir las ecuaciones de Navier-Stokes para ser usadas en una simulación. Al usar DEC se están eliminando los problemas mencionados

con FEM, eliminando las condiciones de frontera que son necesarias para las aproximaciones de FEM. Reduciendo el nivel de complejidad de implementación.

Esta construcción podría ser la nueva forma común de hacer simulaciones y dando la parada para que los próximos métodos estén basados en cálculos discretos. Aunque con DEC requiere de altos conceptos de geometría computacional al final la implementación no será tan compleja ni tampoco tan costosa. Se escoge la combinación DEC y Quad-edge porque los dos se complementan en términos computacionales.

Otra problema encontrado en otras estructuras topológicas es que ninguna de ellas ofrece almacenar el primal y el dual dentro de la misma estructura como lo hace Quad-edge. Es una oportunidad de uso de esta topología porque DEC exige operaciones sobre el primal y el dual de un complex. Quad-edge facilita esa comunicación sin tener que hacer procesos adicionales como otras estructuras que si lo tienen que hacer.

2. DESCRIPCIÓN DEL PROYECTO

2.1. Objetivo general

- Construir una librería de DEC usando Quad-edge para CGAL.

2.2 Objetivos específicos

- Construir una librería de Quad-edge en CGAL
- Construir una librería de DEC con operadores básicos
- Usar la librería de DEC con la librería de Quad-edge

2.3 Fases de desarrollo

2.3.1 Levantamiento de requerimientos

En esta etapa se planteó un plan de levantamiento de requerimientos usando un método de investigación evolutiva, el cual requiere de un punto de partida y sobre este realizar la acción de investigación hasta llegar a los elementos necesarios. En este caso el punto de partida fue FEM. Desde aquí se descubren cantidad de trabajo investigativo hasta llegar a una forma creativa de resolver las PDE. De esta manera se descubre DEC que tiene como pilar fundamental, estar basado en el teorema de Stokes. DEC requería de una estructura de datos topológica que tuviera la capacidad de almacenar la malla y su dual. Desde aquí nace como nuevo punto de partida la librería CGAL, por ser una librería de geometría computacional y siendo una de las

más usadas. Se empieza una búsqueda sobre esta librería de una estructura que cubra las necesidades de DEC. Esta búsqueda se realiza sin éxito pero con una pista, pues se encuentra una estructura llamada Half-edge. Al realizar una búsqueda alrededor de esta estructura se encuentran otras dos estructuras hermanas y una de ellas tiene la ventaja de almacenar el dual de la malla sacrificando un poco la eficiencia en memoria pero potenciado por la capacidad de navegar por el primal y el dual de una triangulación en tiempo O(1). Al seguir investigando sobre esta estructura se descubre que no solo almacenaba el dual, sino que también tenía la capacidad de almacenar variedades no orientables si se implementaba una funcionalidad y variedades orientables si no se hacía. Para terminar esta estructura también está basada en el teorema de Stokes. Aquí es donde se atan todos los conceptos. Porque con DEC usando Quad-edge se podría hablar el mismo lenguaje. Además de ser apoyado por trabajos relacionados encontrados hablando de la eficacia de DEC en la solución numérica de las PDE y como podría ser una buena opción usar con Quad-edge.

Como se nota, este plan de investigación evolutiva genera una construcción de conocimiento basado en un tema inicial y a partir de investigar sobre ese tema se puede encontrar diferentes proyectos asociados. Encontrar que el teorema de Stokes es el punto esencial detrás de la investigación fue gracias a que se encontró un cálculo y una estructura que hablaban sobre este teorema. Potente razón para parar la investigación evolutiva y empezar a entender todos los conceptos asociados: DEC, Quad-edge y CGAL.

2.3.2 Diseño

El diseño de esta aplicación se realizó bajo el modelo SCRUM, el cual para esta investigación consistió en entregas semanales de avances de las diferentes etapas del proyecto. En la etapa de levantamiento de requerimientos las entregas semanales estaban basadas en la investigación evolutiva informando de cada avance encontrado. Esta etapa requirió de un semestre para poder entender la idea general y los objetivos específicos necesarios para el proyecto y adicional a eso entender cuales eran los conceptos necesarios para la implementación. Durante la etapa de la implementación se plantearon hitos de desarrollo para las entregas semanales. Estos hitos consistían en entregas parciales de la librería inicialmente de Quad-edge. En estas entregas se analizaban los avances y problemas obtenidos durante la semana. Esto ayudaba a comprender el problema y daba pie para el siguiente hito de la siguiente semana. De esta manera se podía ir entregando fracciones de la librería hasta tenerla terminada. Al realizarse mediante ingeniería inversa, las primeras semanas fueron diseñadas para la construcción de diagramas sobre la librería Half-edge y la reescritura de esta librería renombrándola como Quad-edge. Una vez que la librería Quad-edge funcionaba con CGAL como si fuera Half-edge, se destinó el tiempo para quitar funcionalidades de Half-edge para luego añadir las funcionalidades de Quad-edge. Una vez que la librería quedó terminada las siguientes entregas fueron pruebas sobre la librería. En esta fase se usó una estrategia de realimentación dependiendo del resultado de las pruebas, se inician las pruebas con la construcción de un vértice hasta llegar a elementos más complejos. Estas pruebas se basaron en validar la construcción de un Quad-edge según las reglas que un Quad-edge debe cumplir para cada elemento. De esta manera cada entrega debía tener la prueba su validación. Cuando la librería de Quad-edge estuvo terminada se procedió a hacer lo mismo con la librería de DEC usando la misma estrategia. Lamentablemente el tiempo no alcanzó para

su totalidad sin embargo se deja el diagrama de clases que debe seguirse y los pseudocódigos para la implementación de los operadores basadas en operaciones simples de Quad-edge.

2.3.3 Implementación

De acuerdo con el diseño basado en SCRUM utilizado las entregas se hacen semanales, siguiendo esta idea cada semana se tenía que alcanzar un hito nuevo. Estos hitos fueron divididos en las dos librerías:

- Construir Quad-edge
  - Diagramas
    - Diagrama de clases
    - Diagrama de componentes
    - Diagrama de paquetes
  - Renombrar Half-edge por Quad-edge
  - Quitar funcionalidades de Half-edge
  - Construcción de Vértices
  - Construcción de Caras
  - Construcción Quad-edges aristas
    - Construcción de implementación de plantillas
    - Construcción de operadores básicos rot y onext
    - Construcción de operadores secundarios (Lnext, Dnext, Rnext, Oprev, Sym…)
  - Construcción de operadores de I/O
    - Método In
    - Método Out
  - Construcción de geometrías básicas
    - Construcción y validación de vértice
    - Construcción y validación de aristas
    - Construcción y validación de caras
    - Constricción y validación de triángulos
    - Construcción y validación de triángulos enfrentados
    - Construcción y validación del conejo de Stanford
  - Construcción de pruebas
    - Cargar vértice
    - Cargar arista
    - Cargar cara
    - Cargar triangulo
    - Cargar triángulos enfrentados
    - Cargar el conejo de Stanford
- Construir DEC
  - Diagramas
    - Diagrama de clases
    - Diagrama de colaboración
    - Diagrama de componentes

- Diagrama de paquetes
- Reescribir método de paridades usando Eigen
- Pseudocódigo de método de paridades
- Pseudocódigo de método de fronteras
- Pseudocódigo de la derivada exterior discreta
- Pseudocódigo de la estrella de Hodge

2.3.4 Plan de pruebas

El plan de pruebas se diseñó basado en el artículo de Guibas y Stolfi.

- Pruebas unitarias librería Quad-edge
  - Construcción y validez de un punto
  - Construcción y validez de una arista
  - Construcción y validez de un triángulo
  - Construcción y validez de un complex simple en Quad-edge desde un archivo plano
  - Construcción del conejo de Stanford
  - Validez del conejo de Stanford

2.3.4 Documentación

- Documento de plan de pruebas unitarias librería Quad-edge
- Diagrama de clases librería Quad-edge
- Diagrama de clases librería DEC
- Diagrama de colaboración
- Diagrama de componentes
- Diagrama de paquetes

Toda la documentación se encuentra anexada en el repositorio de esta memoria de grado.

3. MARCO TEÓRICO / ESTADO DEL ARTE

Para entender el lineamiento que se siguió en esta investigación es necesario saber que para poder construir una simulación en especial la de fluidos, se deben seguir las leyes que los rigen. Las ecuaciones de Navier-Stokes son un conjunto de ecuaciones escritas en derivadas parciales usadas para describir el comportamiento de fluidos newtonianos, esto es, fluidos que tienen una viscosidad constante o con un cambio de viscosidad despreciable. Estas ecuaciones por lo general tienen soluciones analíticas con la que se puede saber con total exactitud el comportamiento de un fluido, pero no siempre es posible obtener este tipo de soluciones [4]. Así que para encontrar soluciones a estas ecuaciones se debe aplicar otro tipo de técnicas que darán aproximaciones a la solución. Las aproximaciones son obtenidas generalmente por análisis numérico y cómo estás soluciones requieren de infinidades de cálculo numérico que normalmente son realizados por un computador. Aquí es donde empieza está investigación, la rama de la mecánica de fluidos que requieren de soluciones numéricas.

La mecánica de fluidos computacional que es la rama que encierra este conjunto de soluciones numéricas realizadas sobre un computador. Es una rama ampliamente estudiada, sin embargo, para esta investigación la solución que más se va a detallar es la de métodos de elementos finitos (FEM por sus siglas en inglés). FEM es un conjunto de métodos que consiste en aproximar la solución numérica de las ecuaciones de Navier-Stokes por medio de subsecciones acotadas de una malla [5].

FEM nació como un conjunto de soluciones de varios autores recorriendo desde los años de 1940 hasta 1960, durante ese tiempo se ha refinado, mejorado y le ha dado la fama que actualmente tiene hoy al ser el método más usado por softwares de simulación [16]. Gracias a su alto potencial y estudio ha sido usado en diferentes campos de la física como electromagnetismo y mecánica de fluidos. Además de aplicaciones en simulación de estructuras, medicina, entre otros. Aunque FEM ha sido ampliamente aceptado, no quiere decir que sea la mejor solución. Se ha demostrado que estos métodos tienden a tener problemas numéricos cuando se somete a ciertas situaciones. En este punto nace DEC como una esperanza de solución como un aparente reemplazo de FEM para evitar estos problemas numéricos.

DEC fue inicialmente concebido por Anil Nirmal Hirani, quien en 2003 desarrolla esta teoría como su tesis doctoral. En dicha tesis menciona la posibilidad de generar soluciones numéricas mediante el uso de combinatorias discretas y operaciones geométricas realizadas únicamente a partir de un complex y su dual [3]. Mediante este cálculo se puede reescribir todas las operaciones realizadas en el cálculo tradicional como operaciones discretas. Esta es una forma de discretización desde un punto de vista diferente al usado en FEM, aunque los dos trabajan sobre

la malla en sí, FEM debe hacer operaciones locales sobre la malla para ir discretizando uno a uno los valores resultantes del cálculo. Mientras que DEC siempre busca la manera de realizar el cálculo de manera discretizada.

DEC exige que las ecuaciones que gobiernan el fenómeno a simular sean reescritas en termino de los operadores discretos. Los operadores son calculados a partir de la geometría y topología de la triangulación. Los operadores son cálculos matriciales sobre matrices dispersas. Al ser matrices dispersas hace que estos cálculos y soluciones lineales sean resueltos en O(n). La memoria estaría ligada a la cantidad de elementos diferentes de cero de la matriz dispersa. La librería propuesta en esta investigación hace uso de la librería de algebra lineal Eigen y que resuelve la mayoría de las operaciones que DEC exige en tiempo O(log(n)) [6].

Hirani ha explotado este concepto hasta el punto de construir una librería junto con N. Bell. La librería de DEC que fue escrita en Python detalla la construcción de los operadores que menciona en su tesis doctoral [7]. Adicional hace uso de una estructura de datos topológica con la ventaja de almacenar el dual como una construcción matricial donde cada fila de la matriz es una dimensión del complejo simplicial y cada columna es el primal o el dual, es decir solo habría dos columnas. Esto está contenido en una clase que hereda de la clase lista de Python por lo que adicional a esto también almacena información adicional de la lista en sí. La topología usada permite acceder a los elementos por dimensión y hacer operaciones de dichas dimensiones con DEC. Esto significa que se puede aplicar los operadores de DEC a cada dimensión del complejo simplicial.

A pesar de que el trabajo realizado en esa librería es grandioso, el eslabón débil es la estructura topológica usada. Para resolver esto se busca en la librería de geometría computacional por excelencia CGAL una estructura que pueda adaptarse más al trabajo de PyDEC, sin embargo se encontraron dos problemas. Primero usar CGAL implica usar el lenguaje de programación C++. PyDEC al estar escrita en Python tendría que ser reescrita en C++. El segundo problema es que solo se encontró una estructura de datos topológica que no almacenaba el dual. Esta estructura es llamada Half-edge. Al investigar un poco más sobre esta estructura se descubre que Half-edge es una de tres estructuras de datos hermanas, que permiten relacionar los elementos de un complejo simplicial con una alta eficiencia tanto en memoria como en velocidad de procesamiento. El conjunto de estructuras son: Wing-edge [8], Half-edge [9] y Quad-edge [10] son topologías diseñadas para mejorar el rendimiento al realizar consultas de adyacencia en tiempos de ejecución O(1).

Según esta investigación Quad-edge es la estructura topológica más adaptable con los cálculos de DEC, la razón es simple. DEC exige que los cálculos como derivadas o integrales tengan acceso al dual de un complejo simplicial y además de las otras estructuras, Quad-edge es la única que almacena el dual. Esto se puede ver mejor en la siguiente tabla:

	Complejidad algorítmica	Complejidad en memoria	Tiene acceso al dual?	Implementación en CGAL?	Variedad orientable

Wing-edge [8]	O(1)	Σ(2n)	No	No	Si
Half-edge [9]	O(1)	Σ(2n)	No	Si	Si
Quad-edge [10]	O(1)	Σ(4n) con n la cantidad de aristas	Si	No	Si y No

Como se puede apreciar en la tabla, Quad-edge parece tener una desventaja en cuestión de memoria y que no tenga una implementación en CGAL. Claramente esta estructura consume más memoria que otras estructuras sin embargo lo compensa el hecho de almacenar el dual del complejo simplicial. Almacenar el dual y aun así seguir con la misma complejidad algorítmica es una ventaja sobre las otras estructuras aún más grande que su desventaja [10]. Quad-edge permite almacenar además variedades orientables y no orientables dependiendo de la implementación. DEC requiere acceso constante al dual del complejo simplicial, si bien DEC puede ser desarrollado con cualquier otra estructura eso aumentaría la complejidad algorítmica porque implicaría calcular el dual de la malla en un pasó anterior y almacenarla en cache para que no se vuelva a calcular, pero en el momento en que la malla sufra una deformación se tendría que calcular nuevamente su dual.

Quad-edge parece la estructura ideal para esto debido a su poder de almacenar el dual y el primal en la misma estructura. Como también calcularlo implica recorrer la lista de vértices y caras una sola vez [10]. Si se están leyendo desde un archivo. La idea de la investigación es la construcción de una librería de DEC que use Quad-edge como topología y que esté construida sobre CGAL, es decir que hay que tomar la librería de Hirani como un modelo a seguir para traducirlo al lenguaje de programación C++ por el hecho de que Quad-edge va a ser una estructura extensiva de la librería CGAL y esta librería está en C++.

4. TRABAJOS RELACIONADOS

El trabajo más relacionado a esta librería es la misma librería de Hirani [7] porque es eso lo que se quiere construir para que trabaje sobre CGAL. De hecho, esto hace más sencillas las cosas al momento de probar la librería. Se sabe que la librería trabaja como se espera si arroja los mismos resultados que la librería de Hirani. Pero la idea de esta librería es que sea también una actualización de la librería de Hirani debido a que ésta no usa una estructura topológica convencional y que provoca lentitud de procesamiento en mallas con millones de polígonos. Esto es una de las cosas que Quad-edge entraría en principio a solucionar gracias a su eficiencia.

DEC también ha sido ampliamente investigado al punto de que con este cálculo se han podido hacer simulaciones en distintas áreas de estudio. En el estudio presentado por Mahbod Salmasi

y Michael Potter se puede apreciar de una forma elegante como se traduce las ecuaciones de Maxwell discretizando únicamente sus formas diferenciales a través de DEC mediante el uso de los operadores principales la derivada discreta y la estrella de Hodge [17]. Aunque en este estudio se hace uso de otros métodos numéricos, se puede apreciar la potencia de DEC en electromagnetismo. Otro ejemplo de DEC usado para resolver problemas de mecánica de fluidos como el flujo de Darcy propuesto por Hirani et al. [18] o las mismas ecuaciones de Navier-Stokes en el artículo de M. S. Mohamed et al. [11].

Otro caso publicado en donde DEC fue usado para simular el pulso de radiación Gaussiana sobre antenas [21]. Donde mediante un programa implementado en Java se pudo obtener los resultados esperados. En este caso también se hizo uso de las ecuaciones de Maxwell discretizadas al nivel de DEC. En el pseudocódigo descrito en este caso demuestra la sencillez que hay en la implementación de este tipo de simulaciones aprovechando a DEC.

A pesar de que DEC funciona bien en los casos ya mencionados, se ha evidenciado de que hay limitaciones teóricas en este cálculo a la hora de usarlos para modelos de materiales, está relacionado a problemas propios de este cálculo en el momento de usar los operadores básicos como la derivada discreta o la estrella de Hodge. Estos problemas están descritos para algunos casos especiales en el artículo de P. R. Kotiuga. [20] Donde al final concluye que mediante un método finito se pueden resolver estos problemas.

En otras situaciones se exponen soluciones como las que involucran a las ecuaciones de Navier-Stokes, el tiempo de solución del sistema parece no estar relacionado con el modelo escogido para representar la superficie [22]. Lo que da mucha flexibilidad a DEC sobre fluidos no newtonianos. Por otro lado, el flujo de Darcy parece complicarse un poco más en ciertas situaciones, debido a que las matrices dispersas que necesita DEC tienen más elementos diferentes de cero.

M. S. Mohamed et al. también propone la solución numérica de la ecuación de Navie-Stokes sobre superficies de mallas como triangulaciones [11]. Esto lo hace a partir de la librería que construyó y que es lo que se quiere lograr con esta librería a futuro. Básicamente la librería que se presenta en este documento es una actualización tanto en lenguaje como estructura topológica de la librería de Hirani. Dotando a la librería de CGAL con esta poderosa herramienta.

Otra librería que hace uso de DEC sobre superficies de triangulaciones es la construida por Camilo Rey que presenta una versión de todo DEC sobre una estructura de datos topológica ofrecida por la librería lean-and-mean para Java [12]. Esta investigación arroja como uno de los trabajos futuros, implementar la librería DEC construida por él sobre la estructura de datos Quad-edge y que de hecho es un resultado directo a porqué se eligen estos dos elementos DEC y Quad-edge como candidatos iniciales para esta librería.

Por otro lado, Quad-edge también tiene trabajos relacionados porque es una estructura derivada de las ideas de otras estructuras topológicas sobre todo con Half-edge. Esta última es una estructura que permite tener dos referencias por arista, para ser más concretos guarda la referencia de la arista y la arista en dirección opuesta. Esto facilita recorrer la triangulación desde cualquier vértice, además la librería Half-edge construida en CGAL también almacena las caras dependiendo de la dirección de la arista. La librería de Half-edge en CGAL es la pieza clave

de la construcción de la librería de Quad-edge en CGAL [13]. Porque es adaptar la arquitectura ya propuesta por CGAL en la construcción del Half-edge; en la construcción de Quad-edge.

Otro trabajo relacionado con la estructura Quad-edge fue el realizado por Leonardo Flórez et al. Para expandir la librería ITK para el procesamiento de imágenes [14]. En esa extensión se desarrolló la estructura de datos topológica Quad-edge escrita en C++, la razón por la que no se usó esta librería es porque fue desarrollada con los patrones de diseño de ITK. La forma en la que está construido CGAL es diferente a la de ITK, empezando por que son dos librerías que se usan en contextos diferentes. Sin embargo, esta librería fue una guía muy útil en el diseño de operadores del Quad-edge.

I- TOPOLOGÍA

La topología es la estructura lógica que relaciona los elementos dentro de un complex. A partir de ésta se puede conocer toda la información de la malla por medio de preguntas de adyacencia. Esto es, saber qué arista está a la izquierda de otra, que puntos conectan una arista o que caras comparten una arista. La topología es una herramienta útil a la hora de hacer todo tipo de operación, además dota a la triangulación de elementos adicionales. Al ser una representación lógica no sólo se está tratando a un punto como una coordenada en un espacio de dimensiones, sino también de un elemento que puede o no albergar información adicional. En algunos casos comunes los puntos suelen llevar información de color. Si el problema fuera darle color a las caras se pueden hacer diferentes soluciones, una de ellas sería recorrer todas las caras y preguntar por el color de todos los vértices para sacar un color ponderado y asignar el color resultante a la cara. Pero esto solo sería posible si por debajo de la geometría existe una estructura que permita recorrer las caras de la triangulación y que a partir de las caras se pueda recorrer los vértices que la componen. La topología es la solución a este problema.

Sabiendo la importancia de la topología, ahora hay que saber que existen diferentes estructuras de datos que las representan. Para esta investigación no es necesario conocer todas las estructuras creadas debido a una limitante. La librería CGAL tiene desarrollado una estructura de datos llamada Half-edge [13] y esta estructura funcionaría en combinación de DEC pero implicaría un esfuerzo en términos computacionales. Esto sugiere a usar otra estructura que encaje con el estándar de CGAL. Siguiendo este orden de ideas la investigación sugiere usar estructuras similares a Half-edge.

Se estudiaron topologías bien conocidas, que por su construcción son altamente eficientes en términos computacionales. Que realmente son estructuras hermanas de Half-edge y esta es la única estructura construida en CGAL. Al final la estructura Quad-edge es la que mejor se adapta a las operaciones de DEC, razón por la cual es la estructura en la que se basa más la investigación. Junto a ella se realiza la construcción de la librería con el diagrama de clases usando como base la librería CGAL.

1. Wing-edge

Winged-Edge es una de las estructuras más antigua para hacer representaciones de una malla de polígonos. Esta estructura de datos contiene la información de vértices, aristas y caras

mediante tres tablas de información. La primera para conocer los "winged" de cada arista, esto es, las aristas adyacentes a una arista dada. La segunda tabla para mantener la información de todos los vértices dentro de la malla de polígonos y una tercera para conocer todas las caras de la malla en cuestión [8].

Esta representación es bastante buena en cuanto la malla esté compuesta por polígonos simples. Esto no quiere decir que no se pueda representar una malla compuesta por polígonos con huecos. Aunque si lo puede hacer, para ello se deben hacer algunas adiciones o consideraciones que aumentan el tamaño de las tablas nombradas arriba, lo cual tiene complicaciones de memoria. Ahora si se tiene en cuenta que en una simulación la malla de polígonos va cambiando su forma con respecto al tiempo, en la cual esta malla se va deformando y quizás cambiando también el tipo de polígono por la cual está compuesta. Esto hace pensar que Winged-Edge no sería una buena estructura de datos para la simulación de fluidos. Si bien Winged-Edge no es la mejor topología para simulación de fluidos, ésta le va a dar una propiedad única a sus dos estructuras hermanas, que es la capacidad de responder una gran mayoría de preguntas de adyacencia en tiempo constante, además también le da las piezas claves en la construcción de la estructura de datos.

2. Half-edge

Half-Edge es una adaptación de la estructura de datos anterior, en el cual se tiene la mitad de la información ocupando menos espacio que el Winged-Edge, sin embargo, esto también limita su uso drásticamente. Al perder información también pierde el alcance que ya tenía su estructura padre. Esto limita al Half-Edge a solo funcionar en geometrías de variedad 3 sin ninguna clase de huecos vacíos.

Esta estructura de datos es valiosa en términos de tiempo debido a su herencia de su estructura antecesora pero no permitiría una representación geométrica del tipo de simulación que se quiere lograr [9].

Esta estructura esta implementada y ampliamente usada en la librería de datos de CGAL. Esta estructura está implementada mediante el uso del patrón de rasgos, la cual dota a la estructura la capacidad de ser usada por cualquier tipo de dato que le entre a la topología, y en CGAL estos tipos de datos están almacenados en algo llamado un Kernel. El Kernel define los elementos que se van a usar geométricamente hablando y también los tipos de datos que se va a almacenar en las estructuras geométricas. Gracias a esto CGAL puede alternar entre diferentes Kernel y aún así poder utilizar la estructura Half-edge sin ningún inconveniente. Esto es clave porque la librería de Quad-edge sigue este mismo patrón de diseño haciendo que también esta estructura tenga tal flexibilidad.

3. Quad-edge

Quad-edge es una estructura de datos creada por Leonidas Guibas y Jorge Stolfi [10], usada para almacenar la información topológica de un complejo simplicial y su dual. La construcción

de esta estructura está descrita en una serie de pasos lógicos. Describiendo desde cual sería la topología de un punto hasta la de una triangulación. Como su nombre lo indica, en Quad-edge se involucran cuatro Quad-edge por cada arista: la arista en sí, el rotacional, su inverso y el inverso del rotacional. Con estas referencias es posible viajar a través de una estructura.

Todo esto es posible mediante dos operaciones básicas de adyacencia el Rot (rotacional) y el ONext. La operación Rot permite viajar entre el primal y su dual y viceversa. Si se le aplica Rot a una arista, esa función deberá devolver la referencia al dual de dicha arista. Mientras que el ONext permite moverse en algo llamado el anillo de aristas, que básicamente es el conjunto de aristas incidentes en un vértice dado. Una forma de ver esta estructura es imaginarse a la referencia ONext como paradas de un bus inter dimensional dentro de una ciudad. Viajar por el bus de la ciudad permite recorrer la ciudad entera. Por otro lado, la referencia Rot es una dimensión alterna de la ciudad. El bus seguirá existiendo y parando en los ONext de esa nueva dimensión y viajará sobre la misma ciudad en una dimensión distinta. Al usar nuevamente la referencia rot regresaría a la dimensión original. Eso quiere decir que con esas dos referencias es posible recorrer toda la triangulación y todo el dual.

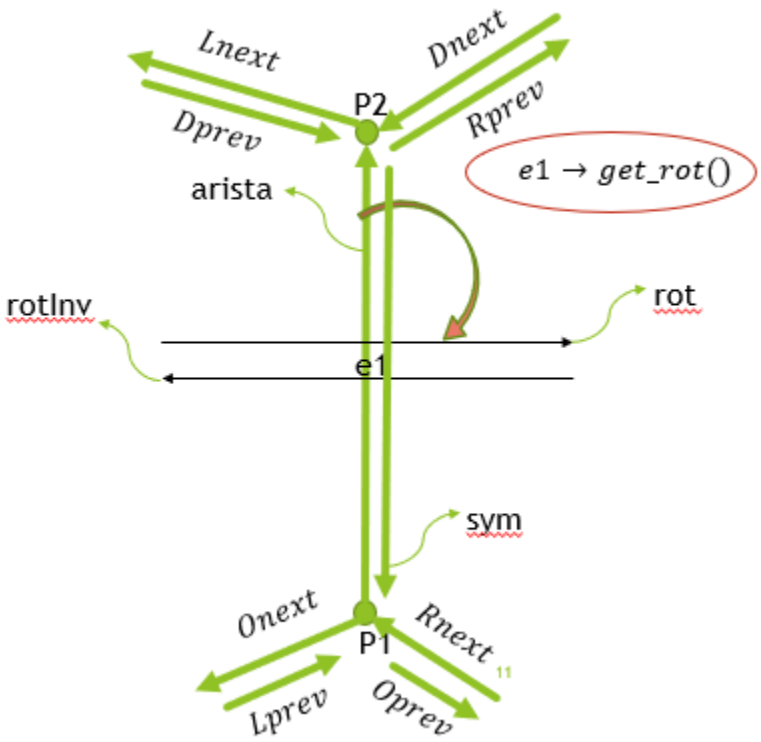

Ilustración 1. Esquema de operaciones de Quad-edge.

La estructura Quad-edge tiene además unas operaciones secundarias para poder circular alrededor de la estructura en un sentido u otro. Lo interesante es que estas operaciones adicionales son construidas como combinaciones de las dos operaciones básicas. Como mencioné el

rotacional sirve para cambiar de una dimensión a otra y por lo tanto en la Ilustración 1 se muestra como aplicando la función get_rot sobre "e1" se puede llegar al rotacional, en este diagrama se representan en flechas de color verde las aristas primales y en flechas negras las dos aristas duales de "e1". Si se quiere construir la operación del simétrico Sym, se debe aplicar sobre "e1" dos veces la función get_rot de esta manera ser transportado del primal al dual en la primera ejecución y del dual al primal en la siguiente. Por consiguiente, el Quad-edge resultante es el que va en sentido contrario a la arista "e1". Si se quiere cambiar de arista ya sea dentro del dual o del primal solo es necesario aplicar alguna de las operaciones secundarias como Lnext o Rnext dependiendo de hacia dónde se quiere trasladar. Se puede dar cuenta el lector que aplicar Lnext recorrería las aristas de una cara dentro de una triangulación en sentido antihorario de las manecillas del reloj.

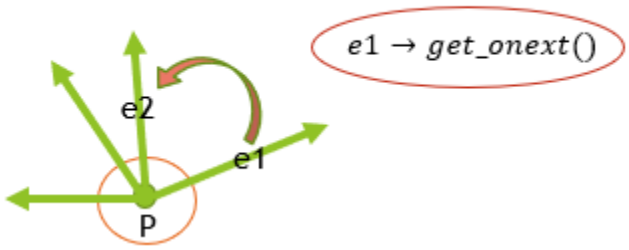

Ilustración 2. Anillo de aristas. Uso de get_onext para viajar de una arista a la siguiente.

Por otro lado, se tiene la operación get_onext que permite recorrer en sentido contrario a las manecillas del reloj. Tal y como se muestra en la Ilustración 2. Así mismo este concepto ayuda a que la estructura no tenga aristas repetidas si una nueva arista es insertada es posible recorrer todas las aristas del anillo y preguntar si esa nueva arista ya se encuentra. De esta forma aplicar Onext no solo sirve para trasladarse por la estructura, sino que también sirve para generar unicidad en la estructura de datos topológica. Con esta función se puede saber en tiempo O(1) todas las aristas que son incidentes a un vértice.

La estructura de datos Quad-edge permite almacenar variedades orientables si en la implementación no se programa un componente adicional llamado flip. Este componente dota a la estructura con la capacidad de almacenar variedades no orientables. Para cumplir con el teorema de Stokes es necesario implementar esta estructura sin el componente flip. Siendo así la estructura debe asegurar que cada arista cumpla con cinco enunciados:

E1. Aplicar el rotacional cuatro veces sobre una arista da como resultado la misma arista.

E2. Aplicar la secuencia Rot Onext Rot Onext sobre una arista da como resultado la misma arista.

E3. Aplicar el rotacional dos veces sobre una arista es diferente a la arista original.

E4. Aplicar el rotacional sobre una arista primal cualquiera, el Quad-edge resultante será una arista dual.

E5. Aplicar el Onext sobre una arista primal cualquiera, el Quad-edge resultante será una arista primal.

Si cada una de las aristas dentro de un complejo simplicial cumple con esta condición entonces la estructura de datos topológica Quad-edge quedó construida de manera correcta. Esto da un punto de partida para poder probar cualquier construcción de simplices. Adicionalmente, para que esto pueda ser posible es necesario actualizar las referencias de los diferentes Quad-edge involucrados cuando se desea unir dos aristas. Se necesitan una manera de unir las aristas topológicamente. Esta función es llamada splice [10], permite que dadas dos aristas se puedan unir mediante la actualización de las referencias Rot y Onext de los Quad-Edge existentes. Teniendo una triangulación implica que para construir una cara es necesario unir mediante el splice al menos tres aristas.

II - DEC.

DEC se un cálculo discreto basado en el cálculo exterior. Hirani propone resolver las PDE por medio de combinatorias discretas y operaciones geométricas en su cálculo exterior discreto. Para usar este cálculo es necesario que las PDE que rigen una simulación sean reescritas en términos de los operadores de DEC. Los operadores son construidos a partir de la triangulación y su dual y la idea es construir dos operadores básicos: la derivada discreta y la estrella de Hodge, estas dos operaciones son las operaciones homólogas de la derivada y la integral. A partir de esos dos operadores se construyen el resto de los operadores. Para la construcción de la derivada discreta se necesitan el arreglo de paridades y el arreglo de frontera. Estos dos arreglos representan el flujo de dos dimensiones. Si se calcula el arreglo de paridades en una cara de la triangulación se obtiene un arreglo binario con la orientación de la cara, si se calcula para todas las caras este arreglo tendría tantos valores binarios como caras hay en la triangulación. En cambio, conseguir el arreglo de fronteras de una cara es conseguir la orientación de cada arista que conforman la frontera de esa cara, también en un valor binario. Siguiendo ese orden de ideas, el arreglo de paridades tendría la orientación de las caras y el arreglo de frontera va a contener la orientación de las aristas. La multiplicación ordenada de estos dos arreglos da una relación de flujo o de orientación entre las caras y las aristas. Este cálculo es posible hacerlo para cada dimensión salvo por la última dimensión de la variedad. A continuación, se hace una explicación de la derivada discreta:

Suponga la siguiente triangulación:

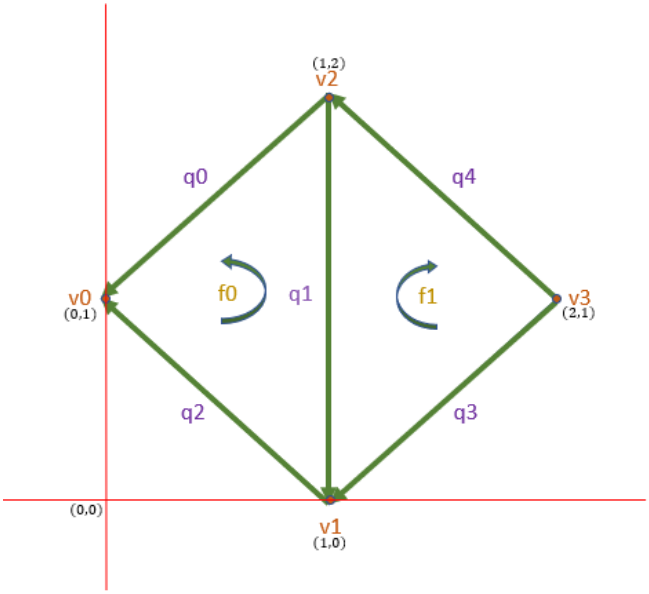

Ilustración 3 Triangulación construida con dos triángulos enfrentados.

En la Ilustración 3 se detalla una triangulación simple donde cada arista es representada con la letra "q", las caras con la letra "f", los vértices con la letra "v". Cada cara y arista tiene una orientación. Esta orientación determinará el resultado de los operadores básicos la estrella de Hodge y la derivada discreta. En este ejemplo se tiene tres conjuntos claramente identificables describiendo los simplices que componen el complex:

$$P = [p0, p1, p2, p3] \qquad F = [f0, f1]$$

$$Q = [q0, q1, q2, q3, q4]$$

Estos elementos están relacionados mediante el flujo que hay de una dimensión a otra. Note que la circulación de una cara no necesariamente va en dirección de la orientación de las aristas. Cada arista puede ser representada como una combinación lineal de los vértices involucrados. De esta forma las aristas pueden ser reescritas de la siguiente forma:

$$-q2 = -(v0 - v1)$$
$$q0 = v2 - v0$$
$$-q1 = -(v1 - v2)$$
$$q3 = v3 - v1$$
$$-q4 = -(v2 - v3)$$

Claro algunas aristas son negativas porque están apuntando en dirección contraria a la orientación de la cara asociada. Este conjunto de ecuaciones forma un sistema de ecuaciones lineales que pueden ser representadas mediante una matriz de coeficientes que relacione las dimensiones de vértices y de aristas:

$$\begin{bmatrix} d0 & q2 & q0 & q1 & q3 & q4 \\ v0 & -1 & -1 & 0 & 0 & 0 \\ v1 & 1 & 0 & -1 & -1 & 0 \\ v2 & 0 & 1 & 1 & 0 & -1 \\ v3 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Esta matriz es la denominada derivada externa. La matriz representa la derivada externa de la dimensión 0. Según la definición de este operador es una relación de una dimensión sobre su dimensión superior. La dimensión 0 sería los vértices de la triangulación, la dimensión 1 son las aristas y así sucesivamente. Quiere decir que la derivada discreta está definida a una dimensión menor que la dimensión de un complex de dimensión n. Hasta este punto se entiende cual es la razón de usar elementos de la librería Eigen. Esta librería contiene las operaciones necesarias sobre matrices dispersas. Estas matrices están representadas en la librería DEC_PUJ en un formato CSR debido a la representación natural de tener todos los valores almacenados como arreglos en filas. Debido a que la construcción de esta derivada se realiza una vez obtenido todos los valores es fácil almacenar en memoria los resultados de estas matrices. Esta matriz contiene información adicional, se puede apreciar que las filas representan los vértices y qué relación tiene con las diferentes aristas, así como también la orientación del vértice en sí. Cada columna representa la relación inversa, cuales puntos están involucrados en la construcción de una arista y los signos dirán cuál es la combinación lineal de dicha arista.

III - LIBRERÍA.

Antes de explicar la librería es bueno dejar en claro que es necesario la construcción de dos librerías una para DEC y otra para Quad-edge, Sin estas dos librerías no sería posible hacer simulaciones. Como se muestra en esta investigación solo se construyó la librería de Quad-edge y se realizó el diseño de la librería de DEC junto con su diseño de sus algoritmos. Estas serán las bases para una futura simulación, se recomienda empezar con la ecuación de Navier-Stokes.

Para construir la estructura topológica Quad-edge en la librería CGAL se utilizó la estrategia de ingeniería inversa sobre la librería de Half-edge. La intención fue replicar el código de Half-edge y luego adaptar la funcionalidad propia del Quad-edge. Al replicar el código se mantuvo el estándar y los patrones de diseños usados por CGAL en la construcción de estructuras de datos. También se hereda el manejo de memoria y de datos usados por CGAL. De esta manera Quad-edge sigue el estándar de CGAL.

Siguiendo los lineamientos del desarrollo de la estructura Half-edge de CGAL y teniendo en cuenta de que la estructura Half-edge no contiene la información del dual hace que se replanteé parte de la estrategia usada en la implementación de Half-edge. Para resolver esto usé la especialización de plantillas. Permitiendo tener flexibilidad de escritura de funciones por dimensión. Esto significa que si las preguntas de adyacencia son realizadas en la plantilla del primal los resultados estarán acordes a la triangulación, pero si en cambio la pregunta es hecha sobre la plantilla del dual los resultados estarán acordes al diagrama de Voronoi. Como se muestra en la Ilustración 4.

Lo primero que se construyó en la librería fueron los elementos básicos de la topología estos son los vértices, las caras y el mismo Quad-edge, donde los vértices tienen acceso al Quad-edge principal de la arista. El Quad-edge tiene acceso a la cara que está a la izquierda de la arista y al vértice incidente. Los elementos son clases dentro de la librería y normalmente CGAL pide que esos componentes tengan una clase base que permita darle generalidad a la clase más importante.

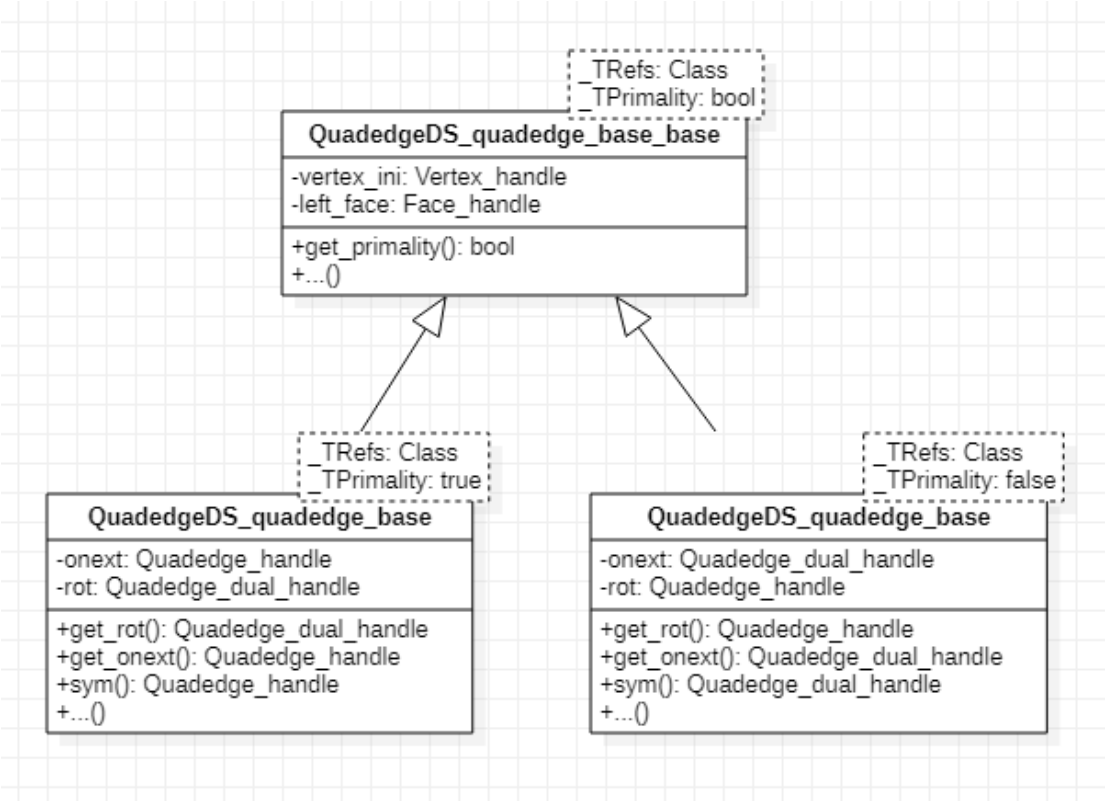

Ilustración 3 Diagrama de clases parcial que representa la estructura Quad-edge como arista dentro de la librería

Especialmente en la clase que representa al Quad-edge como se muestra en la imagen de arriba, la clase base tiene la información de la cara a la izquierda y el vértice incidente por medio de atributos. Adicional a eso tiene dos parámetros de plantilla _TRefs que es la clase por la cual se representan toda la estructura topológica y un booleano _TPrimality que permite al elemento saber en que diagrama se encuentra. Por medio de la especialización de plantillas [15] se generan dos clases diferentes cuando el booleano _TPrimality dice estar en la triangulación o en el diagrama de Voronoi, Aunque los atributos de estas dos especializaciones son realmente los mismos onext y rot, sus tipos no lo son. En efecto, los tipos están trocados permitiendo a la estructura escoger que diagrama usar dependiendo de su primalidad. Se le dice primal a la triangulación y dual al diagrama de Voronoi. En esta librería si la primalidad está en true se está tratando con el primal y si está en false con el dual.

Los atributos onext y rot son iteradores de una clase más alta llamada QuadedgeDS_list, que hereda de una clase de CGAL llamada In_place_list. Esta lista funciona de manera similar a una lista de la librería estándar std de C++. El tipo de dato del onext es Quadedge_handle que al ser un iterador de la estructura principal y que a su vez es una referencia al siguiente Quad-edge en el anillo de aristas, permite que la siguiente referencia también sea posible acceder al próximo onext o rot de esa nueva arista. Para acceder a esas referencias se usan las funciones get_rot y get_onext, de esta manera se podrían hacer operaciones como el sym. Sym es tan sencillo como ejecutar dos veces la función get_rot sobre un Quad-edge, al hacer esto se cambia la dimensión dos veces por eso el tipo de retorno debería ser el mismo del Quad-edge original. En este caso sym devolverá la arista que va en el otro sentido de la arista original. Así como el sym se definen el resto de las operaciones que permiten navegar de cualquier manera por el primal o el dual descritas en el artículo de Leonidas Guibas y Jorge Stolfi [10].

Cada elemento cara, arista y vértice son envueltos en estructuras para poder condensarlas todas en una clase general más grande llamada QuadedgeDS_items_3. Esta clase almacena todas los elementos para hacer uso de sus tipos como rasgos de la clase. Las envolturas terminan con el sufijo _wrapper como se muestra en la ilustración 5. La función de la clase Quaded-geDS_items_3 es ser usada como anunciante de cuáles son los elementos que se van a usar en la clase más grande ya mencionada anteriormente Quadedge_list.

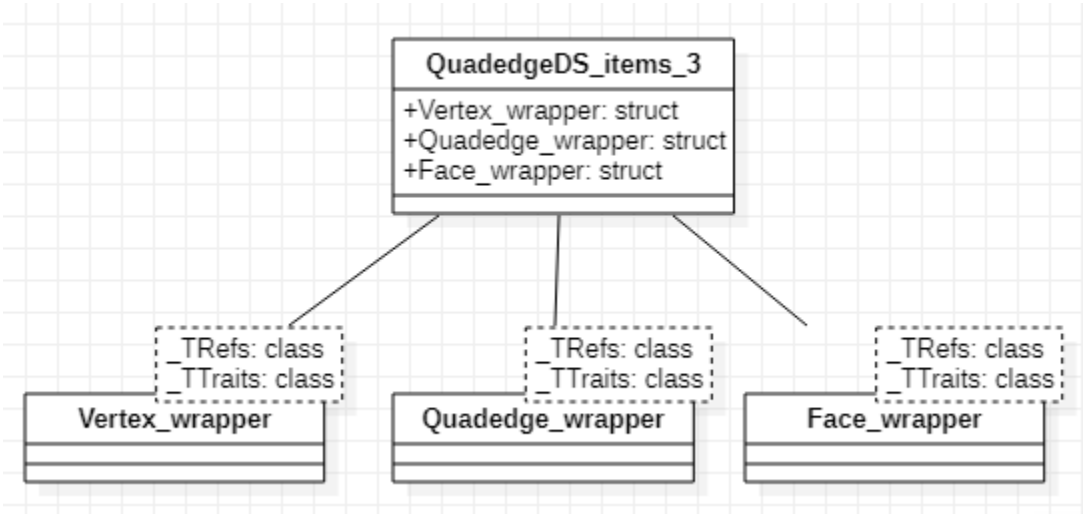

Ilustración 4 Diagrama de clases parcial estructura de ítems.

A diferencia de las envolturas wrapper usadas en CGAL en la estructura de Half-edge, Quad-edge permite tener dos tipos. Los dos tipos son usados para diferenciar el primal del dual y ya fueron mencionados antes como Quadedge_handle para el primal Quadedge_dual_handle para

el dual. La clase de ítems o elementos es usada en la clase QuadedgeDS_list para informarle a la lista quienes son los tipos de elementos a usar.

La clase QuadedgeDS_list es la clase fundamental de toda la topológia porque almacenará toda la información global de la estructura, como por ejemplo la cantidad de caras o cantidad de vétices que tiene la triangulación. Además de poder interactuar con los elementos de memoria usados por CGAL llamados Alloc que permitirán reservar el espacio de memoria definido por el tamaño de la clase a crear. De esta manera si se quiere insertar un nuevo vértice dentro de la triangulación, la clase misma reservará el espacio de memoria dependiendo del tipo de vértice que se está usando mediante el método vertices_push_back() como se muestra en la Ilustración 4.

Los parámetros de entrada de este tipo de clases son normalmente asociadas a la referencia del objeto entrante. La razón de porqué se hace esto es para evitar hacer copias de elementos ya existentes, así solo se manejarán direcciones de memoria en estos métodos. Este patrón es una herencia de haber construido la estructura de datos Quad-edge sobre la del Half-edge. Gracias a esta implementación Half-edge es eficiente en su uso de memoria cuando se construyen triangulaciones.

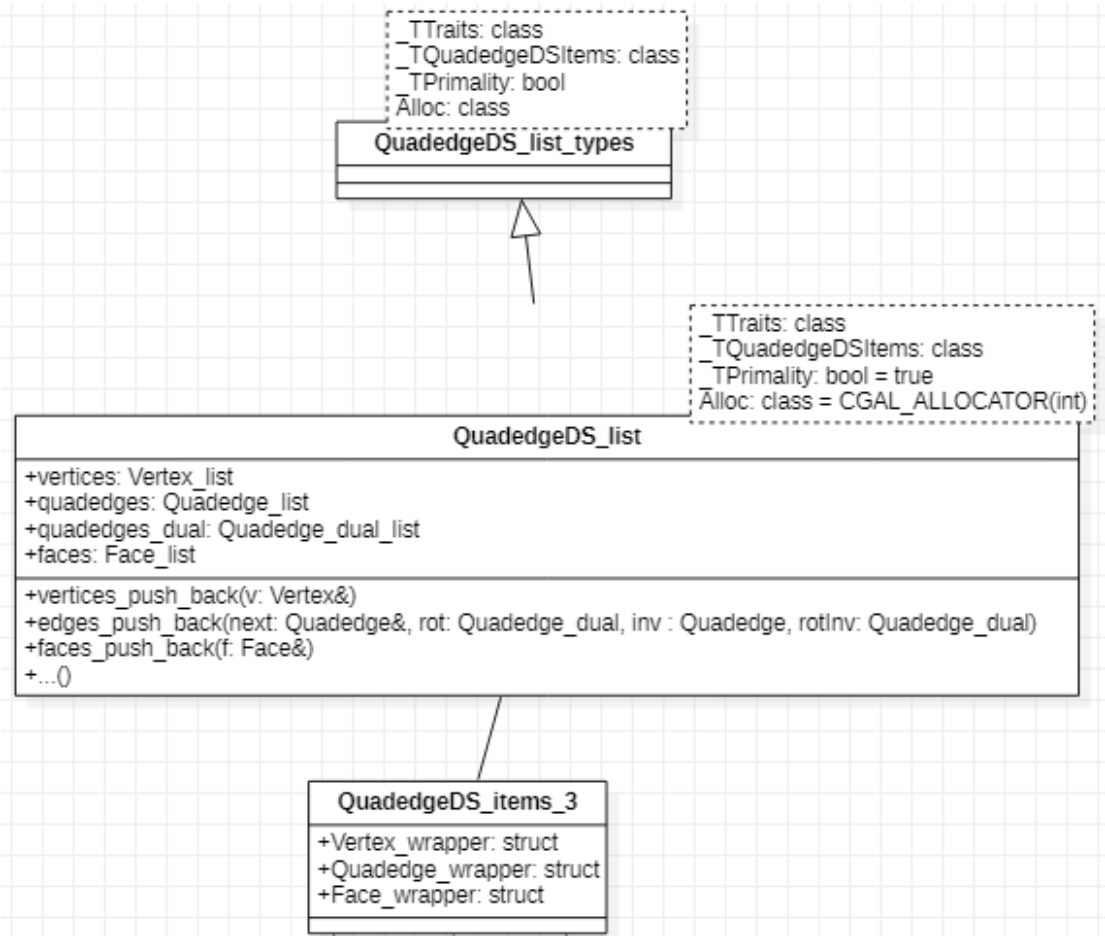

Ilustración 5 Diagrama de clases parcial, ilustra la clase principal QuadedgeDS_list.

La función más especial en el diagrama de arriba es la función edges_push_back, es en esta función donde se encuentra el almacenamiento del propio Quad-edge. Esta función tiene como parámetros de entrada cuatro Quad-edges haciendo referencia a las cuatro referencias que se crean cuando una arista es introducida a la triangulación. Dentro de esta función se actualizan todas las referencias necesarias para que una arista esté bien formada, esto es que cumpla con los cinco enunciados.

La clase QuadedgeDS_list contiene además la información en forma de listas de los vértices, caras, aristas primales y aristas duales. Para tener un acceso directo a la cantidad de simplices que hay por dimensión. Los tipos de esas listas son conseguidas por la clase padre QuadedgeDS_list_types. Esta clase padre es una clase usada con dos propósitos, el primero para darle generalidad al tipo de lista que se use y para definir todos los tipos en común. En especial las listas de los ítems a usar.

El uso de parámetros de plantilla en la clase QuadedgeDS_list permiten aún mayor flexibilidad, al usuario que desee usar la librería. Por medio del parámetro de plantilla _TQuadedgeDSItems se puede introducir una estructura de ítems que se desee siempre y cuando esa estructura tenga

los elementos básicos de los vértices, los Quad-edges y las caras. Cumpliendo con esa condición un usuario puede crear su propia estructura para añadir más información adicional a cualquiera de los ítems.

Finalmente, la topología en sí termina con la clase QuadedgeDS_default que hereda de la clase QuadedgeDS_list, que oculta parte de la implementación y desenvuelve tipos necesarios para ser fácilmente usados en la declaración de tipos de la estructura. Siendo esta la clase que da cara al usuario de la librería y permitiendo usar esta estructura de datos tal y como se usa en Half-edge. Teniendo toda la potencia descrita de Quad-edge.

La topología termina hasta la clase QuadedgeDS_default, pero aún falta poder almacenar la geometría. En este punto se hablará de clases enfocadas en la geometría, para esto comenzaré hablando de la clase Quadedge_triangulation_3. Esta clase tiene como propósito controlar los elementos geométricos de la triangulación y por esta razón van a haber algunos parecidos con los diagramas de clases de la topología. En efecto es necesario tener también elementos como vértices, aristas y caras. También una clase que los encapsule llamada Quadedge_triangulation_items_3 que permita acceder a estos elementos de manera sencilla y a su vez dando la flexibilidad para generar estructuras propias y usarlas con ciertos mínimos elementos.

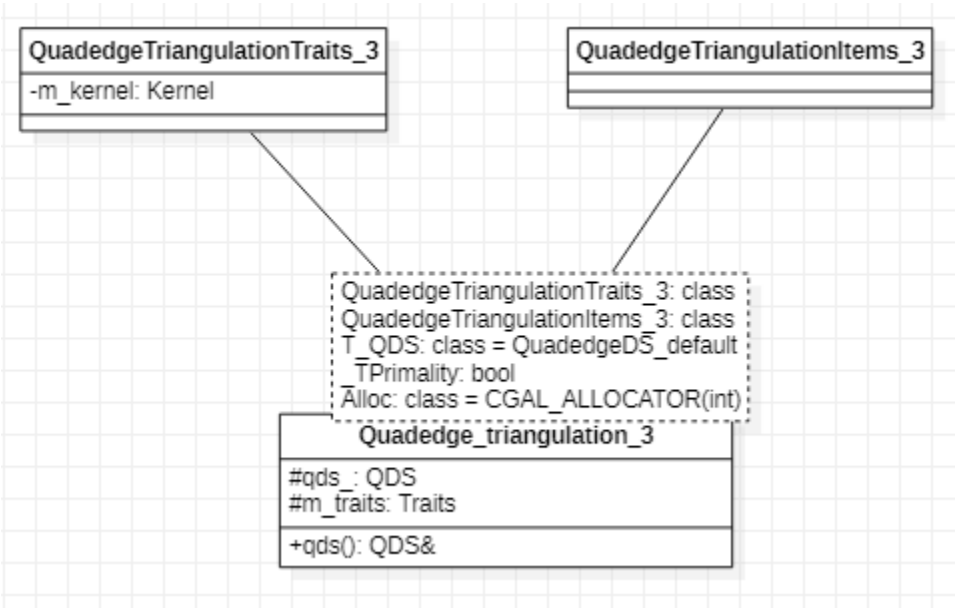

Ilustración 6 Diagrama de clases parcial, detalle Quadedge_triangulation_3

Lo diferente que tiene la clase Quadedge_triangulation_3 es que está relacionada con una clase que controla los rasgos usados entro de la estructura. Esta clase se llama QuadedgeTriangulationTraits_3 como se muestra en la Ilustración 5, permitiendo dejar claro que elementos gráficos son los que se van a usar. Para esta librería se generalizó un poco más los elementos gráficos con respecto al Half-edge. Estos elementos son tridimensionales sin embargo las pruebas de esta librería están enfocadas en dos dimensiones. Sin embargo, se deja la posibilidad del uso tridimensional en la geometría.

La clase Quadedge_triangulation_3 también es altamente flexible y más que todo en el parámetro de plantilla T_QDS que por defecto está la clase QuadedgeDS_default que es la clase que alberga la estructura de datos topológica. Esto quiere decir que algún usuario de la librería podría crear su propia estructura de datos Quad-edge y usarla sobre esta triangulación. Claro, siempre y cuando se respeten los requisitos mínimos de la clase default. Además la triangulación tiene acceso a esa estructura de datos, haciendo que se pueda acceder a los componentes de la topología mediante el método qds(). Teniendo control total tanto de la geometría como de la topología.

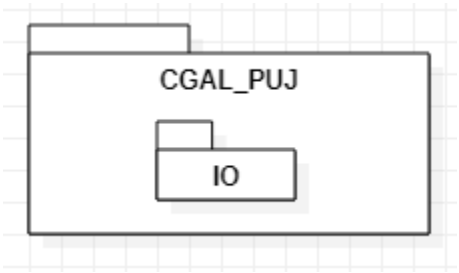

Ilustración 8 Diagrama de paquetes, Librería Quad-edge.

Como se muestra en el diagrama de la ilustración 8. El diagrama de paquetes de la librería Quad-edge tienen dos paquetes definidos. Uno llamado CGAL_PUJ que contiene todos los elementos conseguidos en la ingeniería inversa para la construcción del Quad-edge y también de los elementos geométricos de la triangulación. Dentro de este paquete hay otro llamado IO que se encarga de hacer todas las operaciones de entrada y salida de esta estructura. Para más detalle puede consultar los archivos adjuntos a esta memoria de grado.

Por otro lado, bajo la misma estrategia de ingeniería inversa se construye una estructura de clases similar a las dispuestas en la librería de Hirani, esto implica tener una clase que permita conseguir el arreglo de paridades y el arreglo de simplices con la frontera de la triangulación. Esa clase está construida con métodos estáticos dentro de la clase para que sean accesibles desde el nombre de la clase. Como parámetro de entrada la referencia a la topología de la estructura. Esto es una ventaja de la construcción de la topología con los lineamientos de CGAL. Accediendo directamente al método qds() de una triangulación quad-edge se puede conseguir la topología y con ella lo suficiente para la construcción del arreglo de paridades y la frontera de la triangulación.

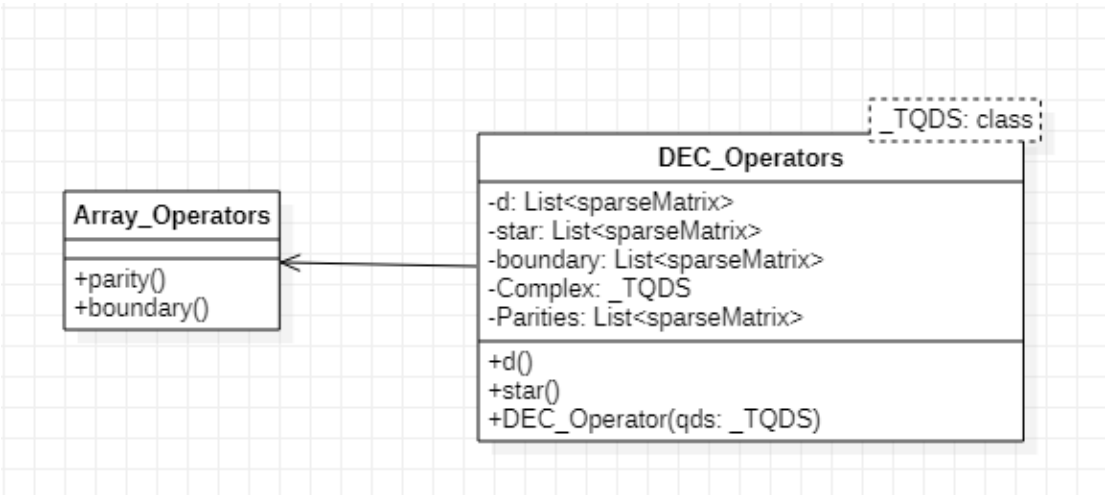

Ilustración 9 Diagrama de clases parcial de la librería DEC, operadores básicos d y estrella de Hodge

La construcción de los operadores d (derivada discreta) y star (estrella de Hodge) es un proceso que requiere de alto procesamiento dependiendo del tamaño de la triangulación. Por esa razón se propone hacer el procesamiento en el momento que se crea una instancia de la clase DEC_Operator. De esta manera es posible tener en la memoria del computador el cálculo de la derivada discreta y la estrella de Hodge. Como estas operaciones se pueden obtener por cada dimensión, son guardadas en la posición de la dimensión de cada una de los atributos de la clase DEC_Operator. Es decir, en el momento de calcular la derivada discreta de la dimensión cero se guardará en la primera posición del arreglo.

Gracias al parámetro de plantilla _TQDS se puede pasar una topología cualquiera como clase y de ahí extraer los elementos para la construcción de los métodos d y star. Este parámetro de plantilla separa la funcionalidad de Quad-edge y la de DEC. Siendo dos librerías distintas para que la librería DEC pueda ser usada con diferentes estructuras de datos topológicas Quad-edge como si fuera un componente plug and play.

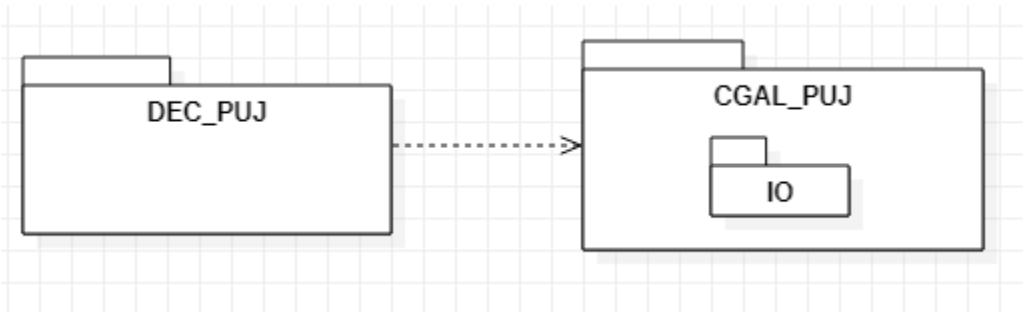

Ilustración 10 Diagrama de paquetes DEC y Quad-edge.

En el diagrama de paquetes se puede ver que las dos librerías se encuentran en dos paquetes distintos. De manera que el paquete CGAL_PUJ pueda ser intercambiado por otro paquete que contenga una estructura de datos con los accesos básicos a los elementos del complejo

simplicial. Como se ilustra en la imagen de arriba, Se detalla que aun así existe tal dependencia porque DEC no puede calcular nada sin una estructura topológica.

El diagrama de colaboración que se muestra en la Ilustración 11. Permite establecer la comunicación que hay en los tres paquetes de la librería cuando un obj es leído desde el paquete de IO. La idea es que esta librería se comunica a través del método addFace para añadir caras a la triangulación. De esta manera el paquete CGAL_PUJ tiene la información suficiente para construir los vértices, aristas (quad-edges) y caras que se requieran para forma la cara, manejando la duplicidad de elementos. Finalmente, con una triangulación ya formada se debe proceder a calcular la derivada discreta y la estrella de Hodge.

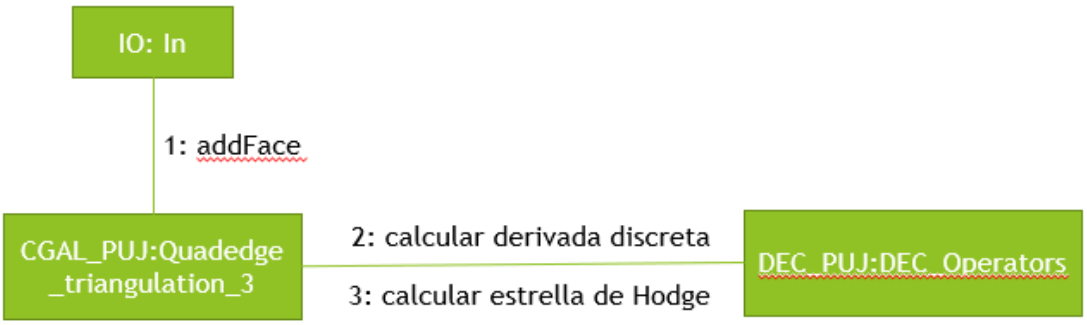

Ilustración 11 Diagrama de colaboración DEC, Quad-edge y IO.

La ventaja de haber combinado una estructura de datos Quad-edge con un cálculo como DEC es la versatilidad que tiene Quad-edge para acceder a todo lo que DEC exige. Esto se puede detallar en la construcción del algoritmo para conocer la derivada discreta. Inicialmente se necesita del arreglo de paridades:

```
Vector v
Foreach face :
    q < −face.quadEdgeOfFace()
    temp < −q.get_Lnext()
    green < −q.getPoint.x ∗ q.getEndPoint.y − q.getPoint.y ∗ q.getEndPoint.y
    while(q! = temp):
        green < −green + temp.getPoint.x ∗ temp.getEndPoint.y − temp.getPoint.y ∗
temp.getEndPoint.y
        temp < −temp.get_Lnext()
    if green > 0:
        v.push(1)
    else:
        v.push(−1)
return v
```

Calcular el arreglo de paridades es aplicar a cada cara de la triangulación un recorrido Lnext para calcular el teorema de Green. El signo del resultado dirá cual es la orientación de dicha cara. Al finalizar el ciclo de caras de la triangulación el vector v deberá contener la orientación de todas las caras. Aplicando la misma idea se procede al cálculo del arreglo de frontera:

```
Vector v
Foreach face :
    q < −face.quadEdgeOfFace()
    temp < −q.get_Lnext()
    while(q! = temp):
        if temp.getIniPoint < temp.getEndPoint:
                v.push(1)
            else:
                v.push(−1)
        temp < −temp.get_Lnext()
    return v
```

Recorriendo las caras por medio de un ciclo, con cada cara se consigue un Quad-edge incidente. Con ese Quad-edge se recorre toda la frontera con la función Lnext y se analiza el orden lexicográfico de los puntos que la conforman. Dependiendo de ese orden se da una orientación u otra. Una vez obtenido los dos arreglos anteriores solo basta de una multiplicación de estos dos arreglos para obtener la derivada discreta:

```
Vector parites
Vector boundary

Matrix m
Foreach face :
    Foreach edge:
        m[face.index][edge.index] =
                        parities[face.index] ∗ boundary[edge.index]
return m
```

Se recorre las caras y por cada arista de dicha cara se consigue la orientación en los dos diferentes arreglos y se realiza la multiplicación la matriz debe comenzar en ceros y tiene dimensión de cantidad de caras multiplicado por cantidad de aristas. Solo los índices involucrados en el cálculo serán llenados y el resto de la matriz quedará en ceros generando así una matriz dispersa.

Para finalizar el diagrama de componentes relaciona todas las librerías en juego para la construcción de la librería DEC-Quad-edge. En la Ilustración 12 se puede ver que las librerías están compartiendo información de manera separada ofreciendo su uso como librerías independientes, si bien el resultado de esta investigación es una librería que usa DEC junto con la estructura de datos Quad-edge, al final se construyó dos librerías independientes para dar flexibilidad a la librería. Por un lado poder usar DEC con estructuras de datos topológicas distintas y por el otro el uso de Quad-edge como estructura de datos independiente con la flexibilidad que ofrece la implementación de CGAL.

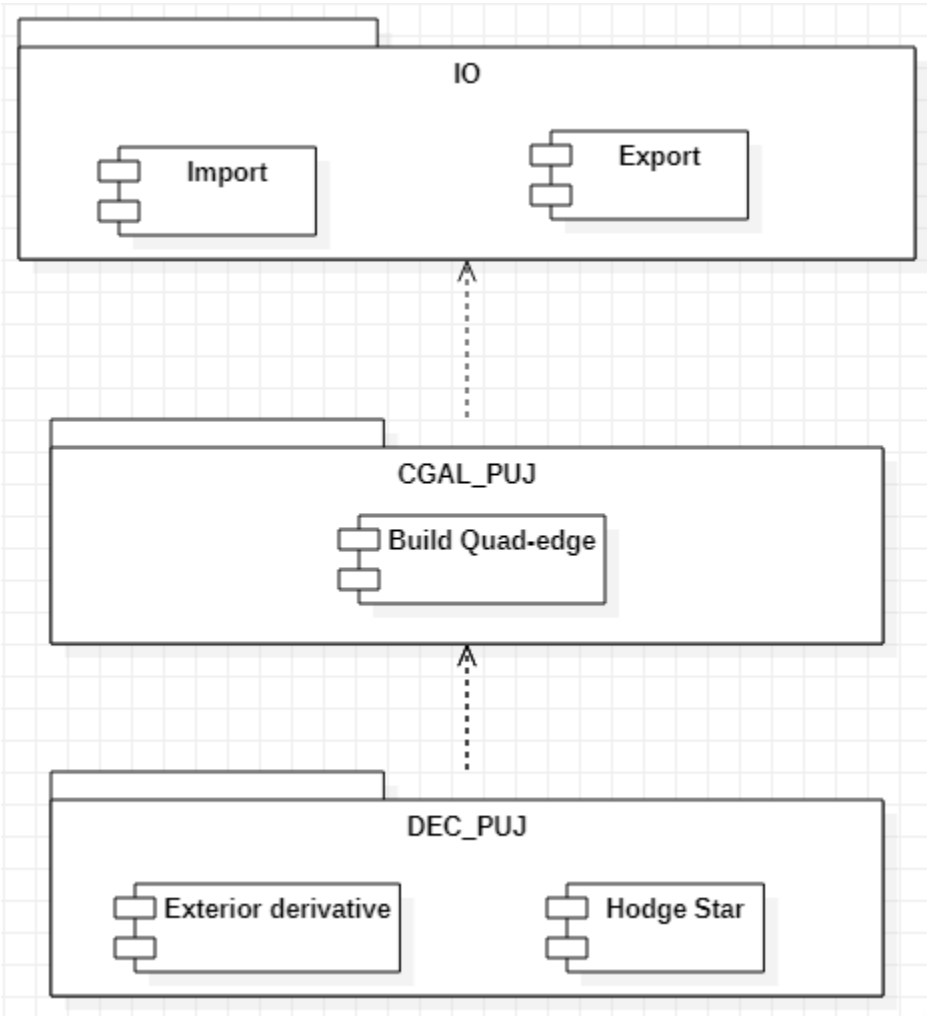

Ilustración 12 Diagrama de componentes de la solución.

Estas son las bases de la librería trabajando en conjunto. Dando como resultados la implementación de dos herramientas poderosas para simulación. Si bien no se realiza una simulación debido al tiempo de investigación. Para más detalle puede consultar los archivos adjuntos a esta memoria de grado.

IV - PRUEBAS.

La librería de Quad-edge es capaz de cargar los archivos obj y validar que la estructura esté bien siguiendo los enunciados que validan que la estructura Quad-edge quedó bien construida. Se realiza ese proceso para cada triangulo dentro de la triangulación. Para este esquema de pruebas se cargan diferentes archivos obj de prueba. Uno de ellos es el conejo de Stanford. La validez de esta estructura se calcula en cuestión de milisegundos.

- Triangulo:
  - Tiempo medido en segundos:

```
Triangle
Elapsed time building data structure: 0.0112626
Elapsed time validating structure: 6.3e-07
Data structure is valid.
```

- Triángulos enfrentados:
  - Tiempo medido en segundos:

```
Faced Triangles
Elapsed time building data structure: 0.00699375
Elapsed time validating structure: 9.64e-07
Data structure is valid.
```

- Estructura con forma de armadillo:
  - Tiempo medido en segundos:

```
Armadilo
Elapsed time building data structure: 3.74063
Elapsed time validating structure: 0.0755296
Data structure is valid.
```

- Conejo de Stanford:
  - Tiempo medido en segundos:

```
Stanford Bunny
Elapsed time building data structure: 3.97843
Elapsed time validating structure: 0.0925011
Data structure is valid.
```

Se puede notar que tanto la carga de las estructuras como la validación de estas no exceden más de 5 segundos ni individualmente ni en conjunto. La construcción del Quad-edge involucra operaciones Onext para saber si una arista existe aún así el tiempo consumido en el conejo de Stanford es de tan solo 3.97 segundos. Mientras que la validación son operaciones de Quad-edge para cada enunciado. Al ser estas operaciones para triangulación en tiempo O(1) el tiempo

consumido en validar la estructura completa es de 0.09 segundos. Demostrando la velocidad de ejecución de esta estructura.

V - CONCLUSIONES.

A pesar de que Quad-edge es una estructura topológica poderosa, se descubrió durante el tiempo de investigación de que no ha sido usada en contexto de simulación especialmente en la mecánica de fluidos. Si bien es una estructura inteligente ha sido usada más para la construcción de modelos. Esta estructura es la que más encaja sobre las operaciones de DEC. La derivada discreta y la estrella de Hodge generan matrices dispersas mediante información de la malla que es obtenida de la topología. Quad-edge obtiene esa información en tiempo O(1), tanto al primal como al dual de la triangulación.

Quad-edge es una estructura que tiene más potencial que los hallados en el marco teórico. Esta estructura permite conocer demasiadas propiedades de una triangulación. Por ejemplo, conocer el convex hull de una triangulación 2D es simplemente aplicar la operación Rnext sobre una arista de frontera y aplicar ese Rnext en la siguiente hasta llegar a la arista original. Aplicar conceptos de geometría computacional sería demasiado sencillos con esta estructura.

DEC sin duda es un cálculo mucho más simple y directo que los método FEM, trabajar sobre la frontera de la triangulación es abordar el problema desde un punto de vista diferente. Usar el teorema de Stokes como fuente principal en la construcción de este cálculo es una idea brillante. Al trabajar sobre la frontera se está evitando trabajar sobre campos vectoriales como lo exige los FEM.

Por el lado de la librería, CGAL permite una flexibilidad única cuando se piensa en estructuras de datos, esto solo es posible gracias a sus módulo independientes de Kernel dentro de su librería. Desarrollar Quad-edge dentro de CGAL es una idea grandiosa, al heredar cada requerimiento no funcional que ofrece CGAL está compartiendo la potencia de esta librería con Quad-edge. Dotando a la estructura de ser usada en otros ambientes incluso sin tener que pensar en DEC.

VI – TRABAJO FUTURO.

La librería DEC-Quad-edge no está completa, por el lado de la estructura Quad-edge solo fueron implementadas las operaciones de adyacencia y la creación de estructuras sencillas. Pero hay muchas funciones de deformación de la triangulación que hacen falta, debido al tiempo de la investigación fue necesario simplificar la librería para un uso más limitado pero funcional. Se invita en investigaciones futuras a realizar estos métodos de deformación.

Desde el lado de Quad-edge es una librería construida a partir de los lineamientos de CGAL, esta librería debería ser puesta a prueba con diferentes implementaciones de CGAL que usen estructuras de datos topológicas, la idea de construcción de la estructura Quad-edge es que

fuera usable independientemente del propósito. Como un ejemplo sería interesante hacer un ejercicio básico de geometría diferencial sobre esta estructura para ver cómo se comporta.

Por el lado de DEC solo fue posible el diseño de los operadores básicos de este cálculo, la derivada discreta y la estrella de Hodge. Sin embargo, existen más operadores que no fueron implementados y que hacen parte de este cálculo. Aunque los operadores construidos en esta librería son los fundamentales, algunas ecuaciones requieren del resto de operadores. La ventaja es que estos operadores son escritos en términos de los operadores esenciales así que la implementación de éstos no debería ser costosa en tiempo.

A gran escala esta librería fue construida como las bases para realizar simulaciones bajo un nuevo paradigma. Después de una ampliación de las librerías DEC y Quad-edge será posible programar las ecuaciones de la simulación en términos de operadores discretos, para poder obtener las soluciones discretas aproximadas y tener una simulación al menos en datos de dicho fenómeno. Una vez se tenga eso se abren dos caminos más una realizar comparaciones sobre simulaciones que usen FEM para comparar la estabilidad numérica, precisión, entre otros. Para comprobar la eficiencia del uso de estos dos elementos. El otro camino es plasmar esos datos en una interfaz gráfica para ver la simulación en tiempo real.

La estrategia de ingeniería inversa es excelente para reconstruir un sistema del que uno no tiene conocimiento alguno. La desventaja es que se requiere de la construcción total del sistema para entenderlo. Esta desventaja se ve más grande cuando no hay una buena documentación del sistema. Además, Seguir los lineamientos de CGAL hace que las implementaciones tengan un alto componente de flexibilidad. Estudiarlos dan un amplio panorama del poder de C++.

REFERENCIAS

[1] R. Eymard, T. Gallouët, and R. Herbin, “Finite volume methods,” in Handbook of Numerical Analysis, vol. 7, Elsevier, 2000, pp. 713–1018 [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570865900070058.

[2] On some techniques for approximating boundary conditions in the finite element method - ScienceDirect.” [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377042795000577.

[3] Hirani, Anil Nirmal (2003) Discrete exterior calculus. Dissertation (Ph.D.), California Institute of Technology. doi:10.7907/ZHY8-V329. https://resolver.caltech.edu/CaltechETD:etd-05202003-095403

[4] G. K. Batchelor, “Equations Governing the Motion of a Fluid,” in An Introduction to Fluid Dynamics, Cambridge: Cambridge University Press, 2000, pp. 131–173.

[5] H. K. Versteeg and W. Malalasekera, An Introduction to Computational Fluid Dynamics: The Finite Volume Method. Pearson Education Limited, 2007.

[6] Eigen: Eigen::SparseMatrix< _Scalar, _Options, _StorageIndex > Class Template Reference.[Online].Available:https://eigen.tuxfamily.org/dox/classEigen_1_1SparseMatrix.html#a0700cd0b8658962d742fa51a5e594a2f.

[7] N. Bell and A. N. Hirani, “PyDEC: Software and Algorithms for Discretization of Exterior Calculus[12],” ACM Transactions on Mathematical Software, vol. 39, no. 1, pp. 3-3:41, Jan. 2013, doi: 10.1145/2382585.2382588.

[8] B. G. Baumgart, “Winged edge polyhedron representation.,” Stanford University, Stanford, CA, USA, Technical Report, 1972.

[9] D. E. Muller and F. P. Preparata, “Finding the intersection of two convex polyhedra,” Theoretical Computer Science, vol. 7, no. 2, pp. 217–236, Jan. 1978, doi: 10.1016/0304-3975(78)90051-8. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0304397578900518.

[10] “Primitives for the manipulation of general subdivisions and the computation of Voronoi | ACM Transactions on Graphics.” [Online]. Available: https://dl.acm.org/doi/10.1145/282918.282923.

[11] M. S. Mohamed, A. N. Hirani, and R. Samtaney, “Discrete exterior calculus discretization of incompressible Navier–Stokes equations over surface simplicial meshes,” Journal of Computational Physics, vol. 312, pp. 175–191, May 2016, doi: 10.1016/j.jcp.2016.02.028. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0021999116000929.

[12] R. Torres and C. Camilo, “Discrete exterior calculus : a framework for computer simulation and geometric analysis based on advanced mathematical concepts,” 2016 [Online]. Available: http://repository.javeriana.edu.co/handle/10554/19636.

[13] “CGAL 5.0 - Halfedge Data Structures: User Manual.” [Online]. Available: https://doc.cgal.org/latest/HalfedgeDS/index.html.

[14] A. Gouaillard, L. Florez-Valencia, and E. Boix, "ItkQuadEdgeMesh: A Discrete Orientable 2-Manifold Data Structure for Image Processing," The Insight Journal, p. 122, Jul. 2007 [Online]. Available: http://hdl.handle.net/1926/306.

[15] "explicit (full) template specialization - cppreference.com." [Online]. Available: https://en.cppreference.com/w/cpp/language/template_specialization.

[16] K. k. (1) Gupta and J. l. (2) Meek, "A brief history of the beginning of the finite element method," International Journal for Numerical Methods in Engineering, vol. 39, no. 22, pp. 3761–3774, 01 1996, doi: 10.1002/(SICI)1097-0207(19961130)39:22<3761::AID-NME22>3.0.CO;2-5. [Online]. Available: https://login.ezproxy.javeriana.edu.co/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-0030294525&lang=es&site=eds-live.

[17] M. Salmasi and M. Potter, "Discrete exterior calculus approach for discretizing Maxwell's equations on face-centered cubic grids for FDTD," Journal of Computational Physics, vol. 364, pp. 298–313, Jul. 2018, doi: 10.1016/j.jcp.2018.03.019.

[18] A. n. (1) Hirani, K. b. (2) Nakshatrala, and J. h. (3) Chaudhry, "Numerical Method for Darcy Flow Derived Using Discrete Exterior Calculus," International Journal of Computational Methods in Engineering Science and Mechanics, vol. 16, no. 3, pp. 151–169, 04 2015, doi: 10.1080/15502287.2014.977500. [Online]. Available: https://login.ezproxy.javeriana.edu.co/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edselc&AN=edselc.2-52.0-84937566387&lang=es&site=eds-live.

[19] M. s. Mohamed, R. Samtaney, and A. n. Hirani, "Numerical convergence of discrete exterior calculus on arbitrary surface meshes," International Journal of Computational Methods in Engineering Science and Mechanics, pp. 1–13, 12 2018, doi: 10.1080/15502287.2018.1446196.

[20] P. r. Kotiuga, "Theoretical Limitations of Discrete Exterior Calculus in the Context of Computational Electromagnetics," IEEE Transactions on Magnetics, Magnetics, IEEE Transactions on, IEEE Trans. Magn., vol. 44, no. 6, pp. 1162–1165, Jun. 2008, doi: 10.1109/TMAG.2007.915998. [Online]. Available: https://login.ezproxy.javeriana.edu.co/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.4526921&lang=es&site=eds-live.

[21] X. Zheng, Y. Zheng, and M. Yu-Jie, "Numerical Simulation of Antennae by Discrete Exterior Calculus," Communications in Theoretical Physics, vol. 52, no. 6, p. 1, Dec. 2009.

[22] M. S. Mohamed, A. N. Hirani, and R. Samtaney, “Comparison of discrete Hodge star operators for surfaces,” Computer-Aided Design, vol. 78, pp. 118–125, Sep. 2016, doi: 10.1016/j.cad.2016.05.002. [Online]. Available: https://login.ezproxy.javeriana.edu.co/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edselp&AN=S0010448516300227&lang=es&site=eds-live.