Geometría Computacional 2015-2
Proyecto Final

Prof. Adriana Ramírez Vigueras
adriana.rv@ciencias.unam.mx

Ayudante Lab. Diego Andrés Gómez Montesinos
diegoMontesinos@ciencias.unam.mx

14 Mayo 2015

1. Objetivos:

- Que extiendas algunos de los conceptos vistos en clase, como la orientación de vectores y *DCEL*, del plano (2 dimensiones) al espacio (3 dimensiones).
- Que seas capaz de visualizar objetos tridimensionales usando *Processing*.
- Que implementes un algoritmo para calcular el Cierre Convexo $\mathcal{CH}(S)$ de un conjunto S de puntos en $\mathbb{R}^3$.
- Que obtengas un archivo *.stl* para poder imprimir el poliedro resultante del algoritmo de Cierre Convexo.

2. Marco teórico:

Poliedro

Como recordarás, el Cierre Convexo $\mathcal{CH}(S)$ de un conjunto de puntos S en el plano es un polígono simple y convexo. En el espacio, el Cierre Convexo de un conjunto de puntos S es el mínimo poliedro convexo que contiene a todos los puntos de S.

En tu implementación del algoritmo que calculaba el Cierre Convexo en $\mathbb{R}^2$, el resultado era almacenado como una lista de vértices (puntos), ya que esta es una forma sencilla y directa de guardar un polígono. Por tanto, debemos encontrar una forma sencilla de guardar un poliedro.

Un poliedro es una generalización natural de un polígono, es decir, lo podemos definir como el conjunto de puntos en $\mathbb{R}^3$ cuya frontera está formada por un conjunto finito de caras planas que se unen mediante aristas y vértices; y es convexo si para cualquier par de puntos en el poliedro, el segmento que los uno queda dentro.

Al igual que con un polígono, nos interesa tener una representación de la frontera del poliedro, es decir, la superficie de este objeto.

De la definición anterior, tenemos que esta frontera está formada por tres tipos de objetos: vértices (puntos), aristas (segmentos de recta) y caras (poligonos planos).

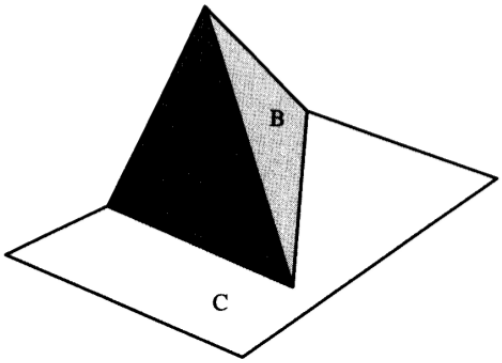
Para descartar ambiguedades, consideraremos como válidos a los poliedros que cumplan las siguientes condiciones englobadas en tres tipos:

1. *Incidencia en componentes*

   Dadas dos caras, se debe de cumplir cualquiera de estos casos:

   - Son disjuntas.
   - Tienen un solo vértice en común.

- Tienen dos vértices en común y una arista las une.



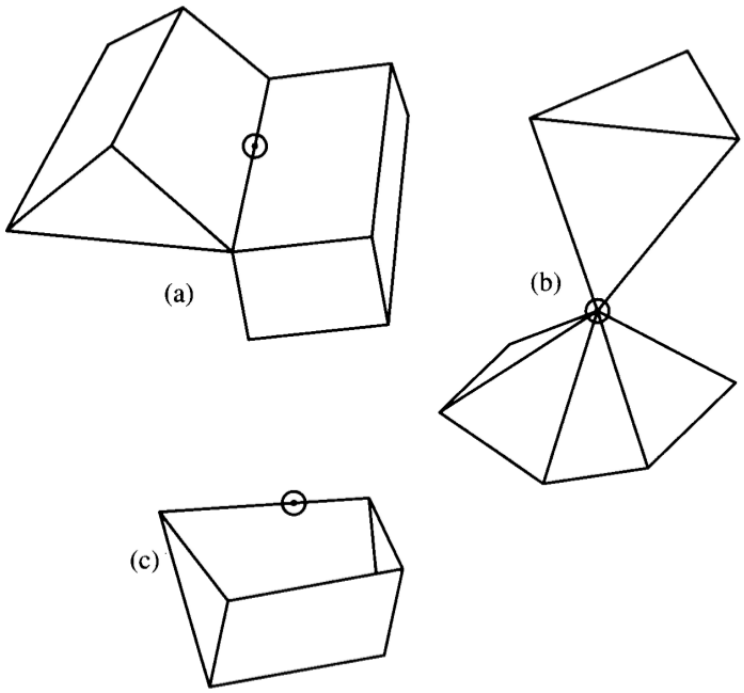
La cara B viola la condición de incidencia con la cara C.

2. *Topología local*

   Esta condición se refiere a cómo se ve un punto cualquiera de la superficie del poliedro localmente. Esto se puede describir intuitivamente: imagina que vuelas sobre el poliedro y observas un punto cualquiera, lo que ves alrededor del punto debe de poderse aplanar.

   Formalmente hablando, se dice que para cualquier punto de la superficie, una vecindad cualquiera suficientemente pequeña debe ser *homeomorfa* a un disco. [1]

   Una consecuencia de esta condición es que cada arista es compartida por exactamente dos caras.



Los siguientes objetos violan esta condición. En el caso (a) la vecindad del punto señalado no es homeomorfa a un disco ya que la arista se comparte por 4 caras, en (b) no existe una tapa, y por lo tanto la vecindad del punto es como un disco doblado y no será uno-a-uno, y en (c) es evidente que el mapeo no será continuo.

[1] Dos conjuntos son *homeomorfos* si existe un mapeo entre ellos que es uno-a-uno y continuo.

3. *Topología global*

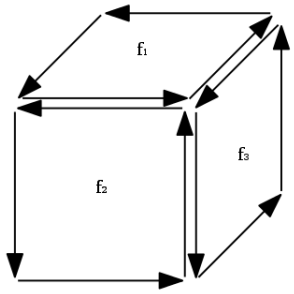
   Pediremos que la superficie sea conexa, cerrada y acotada. Que sea conexa significa que no habrá cavidades internas, y que podemos llegar de cualquier punto en la superficie a otro de manera continua.

   Descartaremos también hoyos que atraviesen la superficie de lado a lado, es decir, túneles. Por ejemplo, la superficie de un toro (dona) no es un poliedro.

Fórmula de Euler y *Dcel3D*

El resultado que nos servirá para idear una representación, es que la fórmula de Euler ($V - E + F = 2$) se cumple para poliedros válidos bajo las condiciones anteriores. Esto implica que la complejidad de un poliedro es de orden $O(n)$ y que lo podemos aplanar. Dado que lo podemos aplanar, entonces podemos verlo como una subdivisión planar. ¿Qué estabamos usando para representar subdivisiones planares que también guardan caras, vértices y aristas? Una *DCEL*.

Por lo tanto, podemos guardar un poliedro válido en una versión tridimensional de nuestra *DCEL*.



Idea de hexaedro (cubo) guardado en una DCEL (solo f_1, f_2 y f_3 son visibles).

Algoritmo Incremental

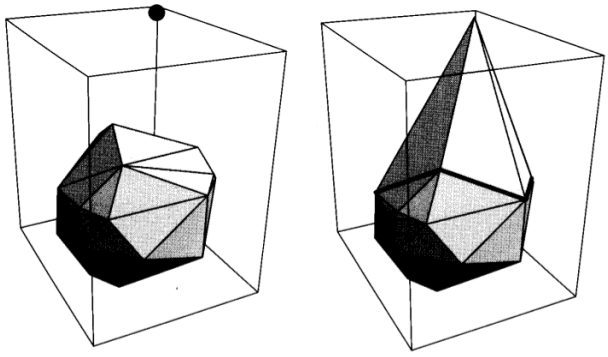
Uno de los algoritmos de Cierre Convexo cuya generalización al espacio es muy sencilla de entender es el incremental.

De hecho, el algoritmo en general es identico al del plano: Se procesan los puntos uno a uno, cuando va a procesar el i-ésimo punto se ha construido el Cierre Convexo de los puntos anteriores $\mathcal{H}_{i-1}$ y se calcula la unión del Cierre Convexo con el i-ésimo punto para tener una solución de los primeros i puntos $\mathcal{H}_i = append(\mathcal{H}_{i-1}, p_i)$. De nuevo, la operación más importante del algoritmo es *append*.

Tenemos dos casos, p_i está o no dentro de $\mathcal{H}_{i-1}$. Calcular esta decisión debe observar si la posición de p_i con cada cara, similar a la decisión que se hace con cada arista en el caso del plano usando el área del paralelogramo. Esto toma $O(n)$.

Si no esta dentro, entonces tenemos que calcular caras triangulares que son tangentes al cierre convexo $\mathcal{H}_{i-1}$, unir estas caras al cierre formando $\mathcal{H}_i$ y por último eliminar las caras que ya no forman parte del cierre. Estas caras forman un cono cuya cúspide es p_i.

Calcular las caras tangentes y eliminar las que ya no forman parte del cierre se calcula en $O(n)$, por lo tanto el algoritmo tiene una complejidad de $O(n^2)$.



Ejemplo de un punto agregado a un cierre convexo.

3. Descripción:

Como primera parte del proyecto debes implementar los métodos que están pendientes en la DCEL, estos incluyen el método para visualizar el poliedro en *Processing*.

Posteriormente, debes implementar el algoritmo incremental para calcular el $\mathcal{CH}(S)$ en $\mathbb{R}^3$.

Al igual que en tu práctica de cierre convexo de dos dimensiones, primero debes implementar el método que agrega un punto a un cierre convexo hecho, pues este es el corazón del algoritmo incremental.

Como sugerencia, empieza por leer las secciones 4.1, 4.2 y 4.3 del libro *Computational Geometry in C, Josheph O'Rourke* (págs 101-144) donde viene una implementación de este algoritmo en lenguaje C, así como una descripción de los detalles importantes en la implementación y cómo resolverlos.

Una vez que hayas implementado el algoritmo y haciendo uso de la biblioteca *toxiclibs* (`http://toxiclibs.org/`) obtendrás un archivo *.stl* que contenga la información de un poliedro resultante para después obtener una versión física usando impresión 3D.

4. Implementación:

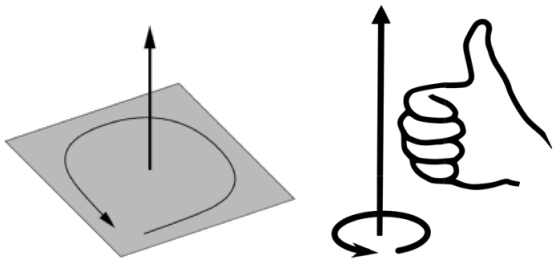
Ya se te proporciona el código base para tu implementación. El código está bien documentado para que entiendas el comportamiento y descripción de cada método.
La carpeta *src* tiene la siguiente estructura:

```
src
  |-- algorithms
  |     |-- convexHull3D
  |         |-- IncrementalCH.java
  |-- structures
  |     |-- dcel3D
  |         |-- Dcel3D.java
  |         |-- DCELUtils.java
  |         |-- Face3D.java
  |         |-- HalfEdge3D.java
  |         |-- Vertex3D.java
  |     |-- vector
  |         |-- Vector3D.java
  |         |-- VectorUtils.java
  |-- visualization
  |-- Main.java
```

En la carpeta *structures/dcel3D*, se te proporciona la implementación de la DCEL através de la clase Dcel3D similar a la que ya has ocupado en la última práctica.

Hay dos observaciones importantes sobre esta versión de la DCEL. La primera es que las caras no pueden tener componentes internos dado que no estamos incluyendo hoyos ni tuneles en los poliedros, esto implica que ya no existe una cara externa.

La tercera observación es la orientación de las aristas en cada cara: la orientación seguirá una convención llamada *regla de la mano derecha*, esto es, si se orientan los dedos de la mano derecha igual que las aristas el dedo pulgar debe quedar apuntando hacia afuera del poliedro (al vector inducido se le llama *vector normal*). Otra forma de pensarlo es ver la cara de frente y el sentido de las aristas es *counterclockwise* (tal como la versión en dos dimensiones).



Explicacion de la orientación de aristas en las caras.

La clase DCELUtils contiene varios métodos de utilería para manejo de la DCEL.

En la carpeta *structures/vector* se encuentran las clases Vector3D y VectorUtils. En esta última debes implementar la generalización del área con signo de un paralelogramo usado en orientación de vectores como el volumen con signo de un tetraedro.

También se incluye una función para leer un conjunto de puntos a partir de un archivo.

En la carpeta *algorithms/convexHull3D* está la clase IncrementalCH en la cual irá tu implementación del algoritmo.

5. Visualizacion:

Para la visualización de este proyecto deberás hacer un *sketch* de *Processing* similar al de la práctica de Cierre Convexo en $\mathbb{R}^2$: presentará una animación donde se muestre cada paso del algoritmo, es decir, que sea evidente cómo es que se va construyendo el cierre convexo en el paso i.
Debe cumplir las siguientes especificaciones:

- Al principio debe obtener el arreglo de puntos guardados en el archivo pointCloud.txt que se encuentra en la carpeta *data*, usando el método *readVectors*.

  Una vez hecho esto se visualizará el arreglo de puntos en *Processing* en 3D.

- Con la tecla *Enter*, empezará la animación del algoritmo.

- Una vez que el algoritmo termine, se debe notificar al usuario y se debe obtener un archivo *.stl* ocupando la biblioteca *toxiclibs*.

Hint: se le puede especificar a *Processing* que se van a dibujar objetos en tercera dimensión agragando a la función *size* el modo de render P3D. Por ejemplo:

```
size(500, 500, P3D);
```

6. Toxiclibs y archivo STL:

El formato STL (*STereo Lithography*) es un formato creado por 3D Systems para guardar información de sólidos y es ocupado por impresoras 3D.

Para *Processing* existe la biblioteca toxiclibs que, entre otras cosas, contiene un objeto para representar mallas en tres dimensiones (parecida a nuestra DCEL) llamada *TriangleMesh* cuyas caras son triangulos. Este objeto tiene el método *saveAsSTL* que ocuparemos para obtener el archivo STL.

En la clase Dcel3D, debes implementar el método *toTriangleMesh* que transforme una Dcel3D en un objeto *TriangleMesh* para posteriormente generar el archivo STL. Para tal propósito todas las caras de la Dcel3D deben ser triángulos.
Puedes consultar la documentación en `http://toxiclibs.org/docs/core/`.

La biblioteca toxiclibs ya está incluida en la carpeta *libraries* del proyecto.

7. Pruebas:

Solo hay una *suite* de pruebas para asegurar que el algoritmo incremental está funcionando correctamente. Para correr las pruebas: abre la consola y estando en el directorio de la práctica, ejecuta el comando *$ ant test*. Esto genera un directorio llamado *report* donde se encontrará el archivo correspondiente a el resultado de la prueba.

8. IMPORTANTE:

No olvides hacer un buen reporte de proyecto: hacer comentarios de tu implementación, del código base, anotar la complejidad de tus algoritmos, etc.
Recuerda modificar la ruta de la instalación de *Processing* en el archivo build.xml.
CUALQUIER DUDA MANDA UN CORREO AL AYUDANTE.

9. Entrega:

El proyecto se debe entregar el Jueves 4 de Junio del 2015 antes de las 23:59:59. El viernes 5 de Junio del 2015 habrá una sesión (con hora aún por definir) para ponernos de acuerdo cómo se va a llevar a cabo la impresión de sus trabajos.