Geometría Computacional 2015-2
Practica 03

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

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

23 Abril 2015

1. Objetivos:

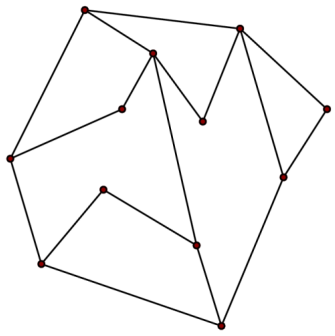
- Que implementes algunas funciones de consulta para una *DCEL*.
- Que implementes una función para pintar una *DCEL* ocupando Processing.
- Que implementes el algoritmo de barrido de línea para dividir un polígono simple en piezas monótonas.
- Que obtengas la triangulación de un polígono simple usando el algoritmo de piezas monótonas.

2. Marco teórico:

DCEL

Una *subdivisión planar*, es una gráfica plana que se embebe, *a.k.a. embedding*, en el plano.
Para embeber una gráfica en el plano, los nodos de la gráfica se mapean a vértices (puntos) en el plano y las aristas a líneas que unen vértices. Se tienen además caras, que son conjuntos maximales conexos del plano que no contienen un vértice, o una arista.

Por simplicidad, vamos a considerar aquellas subdivisiones donde las aristas son líneas rectas y las caras son regiones poligonales, es decir, no hay vértices de grado 1.



Ejemplo de una subdivisión planar.

Para almacenar una subdivisión planar existe una estructura de datos, llamada *Double-Edge Connected List (DCEL)*. En esta estructura se basa en el hecho que una arista limita dos caras, por tanto, una arista se almacena en dos registros llamados *half-edges*.

Los registros que se hacen en una DCEL son:

- Vértice:

- x, y: Sus coordenadas en el plano.
- incidentEdge: Una arista de la cual es origen.

- Half-edge:

- origin, end: Vertices de origen y destino.
- prev, next: La arista previa y siguiente.
- incidentFace: La cara que bordea.
- twin: Su arista gemela.

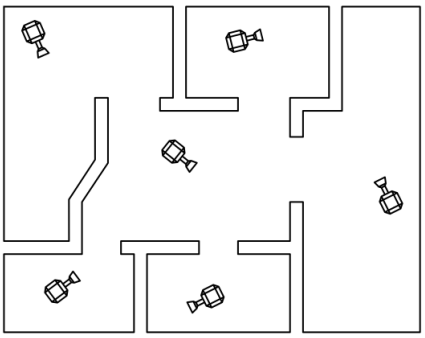
- Cara:

- outerComponent: Es un apuntador a una arista (cualquiera) que la limita.
- innerComponents: Es una lista de apuntadores a aristas que limitan a la cara interiormente, esto es en caso que la cara tenga hoyos.

Con esta estructura se pueden hacer distintas consultas de la información topológica y geométrica de la gráfica, o se puden hacer recorridos sobre las caras.

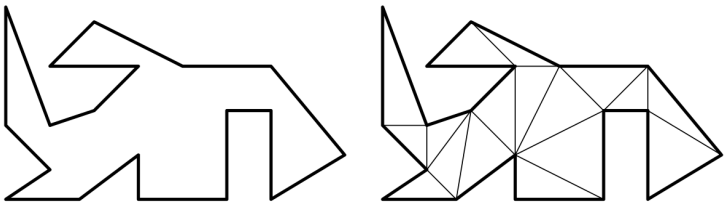
TRIANGULACIÓN DE UN POLÍGONO

El problema de la Galería de Arte consiste en buscar el número de videocamaras suficientes para vigilar una galería de arte vista en su plano de planta, es decir, como un polígono.



Sabemos que si dividimos el polígono en piezas convexas podemos obtener la posición y número de las videocamaras, ya que basta con una cámara para vigilar una pieza convexa.

La estrategia, por lo tanto, es dividir al polígono en piezas más sencillas de tratar. Especialmente podríamos dividir al polígono en triańgulos. A este proceso se le llama triangulación de un polígono.



Un polígono es *monotono* con respecto a una línea l, si cualquier línea perpendicular a l corta dos veces al polígono, o lo intersecta en un punto, o no lo intersecta.

Una manera de triangular un polígono consiste en primero dividirlo en piezas monotonas y después triangular cada una de estas piezas.

Hay un algoritmo para dividir en piezas monotonas que ocupa una técnica que ya has visto en clase llamada Barrido de línea, que consiste en imaginar una línea que barre en un sentido el plano y efectua operaciones dependiendo de lo que se vaya encontrando a su paso.

3. Descripción:

Como inicio, en esta práctica deberás implementar algunas búsquedas sobre una DCEL, para obtener la siguiente información:

- La lista de aristas que son adyacentes a un vértice dado.
- La lista de las caras adyacentes a una cara dada.
- La lista de aristas que son adyacentes a una cara.

También debes implentar una función para pintar una DCEL de tal manera que las aristas no se pinten dos veces (como lo hace la implementación vista en clases).

Después vas a implementar la triangulación de un polígono dado como una DCEL. Para tal propósito, primero debes implementar el algoritmo de barrido de línea para dividir un polígono en piezas monótonas con respecto al eje Y.

Una vez obtenida esta división, que debe ser guardada en la misma DCEL, implementarás el algoritmo que triangule cada pieza monotona. Ver el Capítulo 3 del libro *Computational Geometry, algorithms and applications. Mark de Berg.*

4. Implementación:

Al igual que las prácticas anteriores se te proporciona el código base para tu implementación, así como una descripción del comportamiento de cada uno de los métodos.
La carpeta *src* de esta práctica tiene la siguiente estructura:

```
src
  |-- algorithms
  |     |-- triangulation
  |         |-- PolygonTriangulation.java
  |         |-- MonotoneDivision.java
  |-- structures
  |     |-- dcel2D
  |         |-- Dcel2D.java
  |         |-- DCELUtils.java
  |         |-- Face2D.java
  |         |-- HalfEdge2D.java
  |         |-- Vertex2D.java
  |     |-- vector
  |         |-- Vector2D.java
  |         |-- VectorUtils.java
  |-- visualization
  |-- Main.java
```

En la carpeta *structures/dcel2D*, se te proporciona la implementación de la DCEL através de la clase Dcel2D junto con cada uno de sus registros Face2D, HalfEdge2D y Vertex2D. La clase DCELUtils contiene varios métodos de utilería que ayudan a construir una DCEL.

Los métodos que representan las búsquedas mencionadas como la primera parte de esta práctica, así como el método para pintar, se encuentran en la clase Dcel2D. Asegúrate que entiendes como está estructurada la clase antes de empezar a escribir código.

En la carpeta *structures/vector* se encuentran las clases Vector2D y VectorUtils que implementan vectores en el plano con algunas de sus operaciones.

En la carpeta *algorithms/triangulation*, están las clases MonotoneDivision y PolygonTriangulation.
En la primera debes implementar la división de piezas monotonas con respecto al eje Y, y en la segunda la triangulación de un polígono. Como ya se menciono anteriormente la triangulación debe hacer uso de la primera clase.

Recuerda que si tienes duda de los requerimientos de un método o no te queda claro que es lo que tienes que implementar, envía lo antes posible un correo al ayudante.

5. Pruebas:

Hay tres *suites*: una que prueba tu implementacion de las búsquedas mencionadas en la DCEL, una que prueba tu implementación de la división de piezas monotonas y otro más que prueba el paso de la triangulación.
Para correr las pruebas en la consola estando en el directorio de la práctica, ejecuta el comando *$ ant test*. Esto genera un directorio llamado *report* donde se encontrarán tres archivos correspondientes a los resultados de las pruebas.

6. Visualizacion:

En esta ocasión solo deberás hacer un *sketch* de Processing, el cual debe seguir las siguientes especificaciones:

- Al principio se deberán crear dos DCEL a partir de los archivos exampleA.svg y examplePolygon.svg que se encuentrab en la carpeta *data*. Por default, la DCEL correspondiente al archivo exampleA.svg deberá visualizarse.
- Con click derecho se deberá visualizar la DCEL correspondiente al archivo exampleA.svg.
- Con click izquierdo se deberá visualizar la DCEL correspondiente al archivo examplePolygon.svg.
- Si se está visualizado la DCEL correspondiente al archivo examplePolygon.svg:
  - Con la tecla m se deberá ver la división en piezas monótonas.
  - Con la tecla t se deberá visualizar la triangulación del polígono.

Las visualizaciones que hagas se ejecutan con la clase Main, usando el comando *$ ant run*.

7. Preguntas (punto extra)

Contesta las siguientes preguntas (incluye la respuesta en tu reporte):

- El algoritmo que implementaste para triangular supone que el polígono dado no tiene hoyos. ¿Qué modificarías en tu algoritmo para lidiar con hoyos? ¿Cómo lo implementarías?

8. IMPORTANTE:

No olvides hacer un buen reporte de práctica: 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, si tienes dudas de cómo hacerlo manda un correo al ayudante.

9. Entrega:

Esta práctica se entrega el Jueves 7 de Mayo.