Geometría Computacional 2015-2
Practica 02

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

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

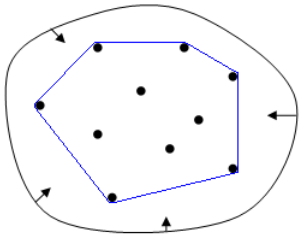
24 Febrero 2015

1. Objetivos:

- Que implementes dos algoritmos para calcular el cierre convexo: Incremental y Divide y vencerás.

2. Marco teórico:

El cierre convexo de un conjunto S de puntos, denotado por $\mathcal{CH}(\mathcal{S})$, es el polígono convexo que contiene a todo S y cuyos vértices son puntos de S, tal que tiene el perímetro mínimo, el área mínima y es mínimo por contención.



3. Descripción:

En clase has visto algoritmos para calcular el $\mathcal{CH}(\mathcal{S})$ de un conjunto de puntos S, algunos de los cuales ya has visto programados en el laboratorio. Es tu turno de explorar la implementación de estos algoritmos y lidiar con sus particularidades. En esta práctica implementarás los algoritmos Incremental y el Divide y vencerás.

El algoritmo Incremental, como su nombre lo dice, consiste en ir agregando uno a uno los puntos del conjunto S para formar el $\mathcal{CH}(\mathcal{S})$. En el paso k del algoritmo se tiene el cierre convexo de los primeros $k-1$ puntos $\mathcal{H}_{k-1}$ y se busca integrar el punto p_k para obtener el cierre de los primeros k puntos $\mathcal{H}_k$. Por lo tanto, el problema principal del algoritmo es resolver cómo integrar un punto a un cierre convexo, tal que se tenga una solución completa.

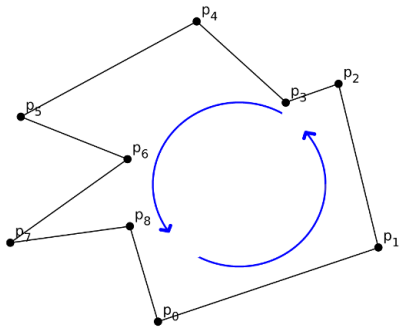
El algoritmo por Divide y Vencerás, ocupa una técnica que ya conoces (recuerda el *mergesort* que viste en tus primeros semestres), y que consiste en dos pasos: primero se divide un problema en varios más pequeños y más fáciles de resolver, y después se unen estas pequeñas soluciones para construir una solución completa.

En términos del cierre convexo, el primer paso consiste en ir dividiendo el conjunto S en mitades hasta que $|S| = 3$, ya que en este caso el $\mathcal{CH}(\mathcal{S})$ es un triángulo. El segundo paso consiste en unir las soluciones, es decir, encontrar el cierre convexo de la unión de dos cierres convexos, y así formar una solución completa.

Entonces, el problema de fondo es tener una manera de mezclar dos cierres convexos.

4. Implementación:

¿Qué estructura ocuparías para guardar un Cierre Convexo? Un polígono.
Un polígono se puede representar con una secuencia de vértices $p_0, p_1, ..., p_n$ donde las aristas están dadas por la posición de los vértices en la secuencia, es decir, se tiene la arista $p_i p_{i+1}$ en el polígono para toda $i \in \{0, 1, ..., n\}$.
Por convención, el recorrido del polígono se hace en sentido contrario a las manecillas del reloj (*counterclockwise*), y el vértice inicial podría ser cualquiera.



Los algoritmos que implementes regresarán, por lo tanto, el cierre convexo representado en un polígono.

Al igual que en la práctica anterior ya 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
  |     |-- convexHull
  |         |-- IncrementalCH.java
  |         |-- DivideAndConquerCH.java
  |-- structures
  |     |-- Polygon2D.java
  |     |-- Vector2D.java
  |-- visualization
  |-- Main.java
```

En la carpeta *structures* están las clases Polygon2D y Vector2D que representan un polígono y un vector en dos dimensiones, tus algoritmos devolverán y trabajarán con estas clases. La clase Vector2D es una envoltura para la clase *processing.core.PVector* y puedes ocupar todos los métodos de esta clase.

En la carpeta *algorithms/convexHull* están las clases llamadas IncrementalCH y DivideAndConquerCH, que representan los algoritmos a implementar.

En la carpeta *visualization* debes colocar los archivos relacionados a los *sketches* de Processing.

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* de pruebas: una que prueba tu implementacion del polígono, y las otras dos que prueban la implementación de los dos algoritmos para el cierre convexo.
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.

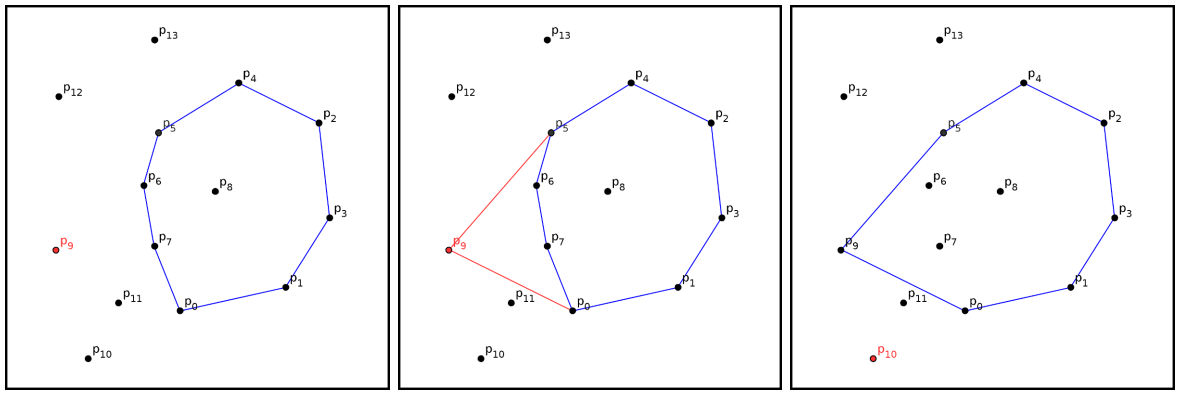
6. Visualizacion:

En esta ocasión harás dos *sketches* de Processing: uno por cada algoritmo.

Para el algoritmo Incremental, el sketch presentaá una animación donde muestres cada paso del algoritmo, es decir, que sea evidente cómo es que se va construyendo el cierre convexo en el paso k.
Los requerimientos son los siguientes:

- El usuario debe de poder agregar puntos mediante el click izquierdo, o generar un nuevo conjunto de 100 puntos con el click derecho.
- Al presionar la tecla *backspace* se borrará el conjunto de puntos actual.
- Con la tecla *Enter*, empezará la animación del algoritmo.
- Mientras corra el algoritmo, no se pueden generar nuevos puntos ni generar aleatorios.
- Se debe notificar al usuario cuando el algoritmo termine.
- Genera un archivo *incrementalCH.gif* donde se vea cómo fue una ejecución de tu sketch.

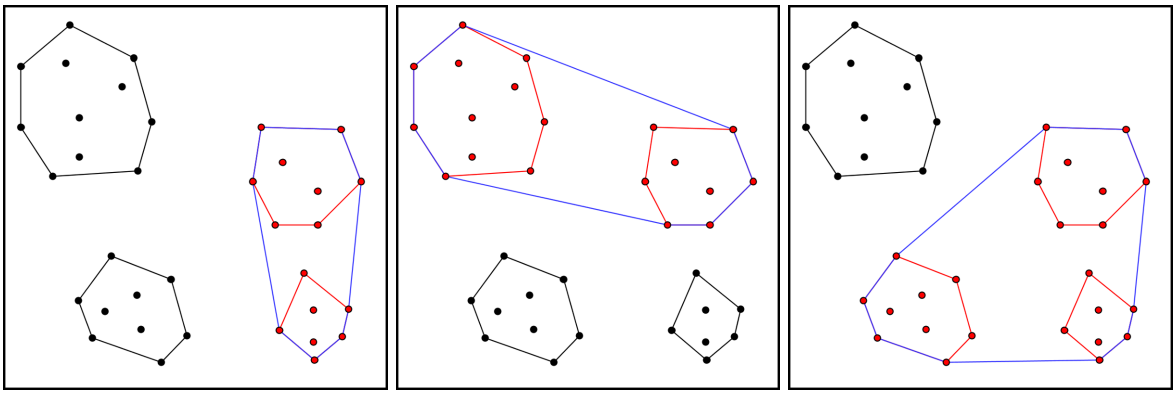
Ejemplo de cómo se podría ver los *frames* de la animación:



Para el algoritmo de Divide y Vencerás, el sketch generará 4 conjuntos de puntos S_0, S_1, S_2, S_3, tales que sus cierres convexos $\mathcal{CH}(S_0), \mathcal{CH}(S_1), \mathcal{CH}(S_2), \mathcal{CH}(S_3)$ no deben de intersectarse y calcular todas las posibles uniones de todos estos cierres convexos. Esto es, la union del $\mathcal{CH}(S_0)$ con $\mathcal{CH}(S_1)$, el $\mathcal{CH}(S_0)$ con $\mathcal{CH}(S_2)$, etc.
El usuario podrá ir cambiando la unión que desea ver. Cada que se muestre la unión, se debe pintar de un color los cierres convexos involucrados y de otro color el cierre convexo de la unión.
Los requerimientos son los siguientes:

- El usuario puede generar 4 nuevos conjuntos de puntos con el click derecho.
- Al presionar la tecla *backspace* se borrarán los 4 conjuntos de puntos actuales.
- Con la tecla *Enter*, el usuario puede ir cambiando la unión que desea visualizar.
- Una vez que se han visto todas las uniones, se regresa a la primera.

Ejemplo de cómo se podría ver el sketch cada vez que el usuario cambie la unión que desea ver:



Las visualizaciones que hagas se ejecutan con la clase Main.
La manera de elegir el *sketch* a ejecutar será mediante el paso del argumento sketch.
Es decir, si se desea ejecutar la visualización para el algoritmo incremental, el comando a ejecutarse es *$ ant run -Dsketch=incremental*; y si se desea ejecutar la visualización para el divide y vencerás, el comando a ejecutarse es *$ ant run -Dsketch=divide*.

7. Preguntas (punto extra)

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

- Para la implementación de tus algoritmos se asume que no hay tres puntos colineales. ¿Cómo afecta a tu algoritmo si los hay? ¿Le tendrías que modificar algo?
  Explica.
- En el algoritmo incremental, se puede alcanzar una mejor complejidad si los puntos están ordenados por el eje X. ¿Cómo podrías hacerlo? ¿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 Martes 10 de Marzo.