

## Tarea 11

---

### Programa. Algoritmo de Visvalingam

#### 0.1 Metodología

Se puede definir un polígono a partir de un conjunto de puntos unidos mediante líneas, lo cual asume que los puntos que nos dan siguen un cierto orden.

El algoritmo de Visvalingam consiste en tomar como entrada un conjunto de puntos que representan un polígono y dar como resultado otro polígono con menos puntos pero que resulte ser lo más similar posible al de entrada.

Esto se logra asociando a cada punto  $p_i$  un área definida por el triángulo formado entre  $p_i$  y sus dos puntos vecinos, es decir, el punto a su izquierda  $p_{i-1}$  y a la derecha  $p_{i+1}$ .

Una vez que se ha calculado el área para cada punto del polígono se hará una serie de iteraciones de los siguientes pasos:

1. Identificar el punto  $p_i$  con menor área asociada y quitarlo del polígono.
2. Actualizar el área de los puntos vecinos a  $p_i$ , es decir,  $p_{i-1}$  y  $p_{i+1}$ .

Este ciclo se repite hasta que la menor área asociada a algún punto sea mayor a un threshold definido por el usuario, o hasta que queden 2 puntos en el polígono, ya que con 2 puntos ya no se puede definir el área y el polígono resultante es una línea recta.

Este procedimiento implica que el porcentaje de compresión que se obtendrá estará en función del threshold de área que defina el usuario.

#### 0.2 Implementación

Para implementar este algoritmo se determinó que la estructura más adecuada para acceder al punto del polígono con menor área es una cola de prioridad, con lo que el acceso es de complejidad  $O(1)$ .

Para actualizar el área de los puntos vecinos en el heap, lo que en realidad se guarda en el heap son apuntadores a puntos, los cuales se pueden acceder con un vector mediante complejidad  $O(1)$ , ejecutando después una función de `heapify_up` sobre la cola de prioridad para mantener la estructura en orden, lo cual tiene una complejidad de  $O(\log n)$ .

Y como el algoritmo va iterando sobre los puntos del polígono quitando uno a la vez, la complejidad total del algoritmo será de  $O(n \log n)$  donde  $n$  es el número de puntos del polígono.

La obtención del porcentaje de compresión se calculó de la siguiente manera:

$$C_{per} = (1 - \frac{|P_{simplified}|}{|P|}) * 100 \quad (1)$$

Donde  $|P_{simplified}|$  representa el número de puntos del polígono simplificado y  $|P|$  representa el número de puntos del polígono original.

Para el porcentaje de error se optó por calcular la proporción de áreas entre el polígono simplificado y el polígono original, bajo la misma intuición del algoritmo de que si los polígonos tienen un área bastante parecida implicará que las forma del polígono simplificado no habrá cambiado demasiado. Por lo que se propone la siguiente fórmula:

$$\epsilon_{per} = (1 - \frac{A(P_{simplified})}{A(P)}) * 100 \quad (2)$$

Donde  $A(P)$  representa la evaluación del área del polígono  $P$ .

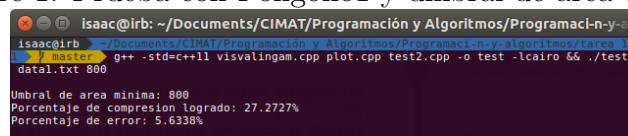
Finalmente, para validar los resultados del algoritmo se implementó una función que hace uso de la biblioteca gráfica Cairo, donde se dibujan 3 cosas:

1. Puntos del polígono original en rojo.
2. Polígono original en azul.
3. Polígono simplificado en verde.

Cabe mencionar que para la graficación se usaron polígonos con coordenadas  $(x, y)$  en el intervalo  $0 < x < 200$  y  $0 < y < 200$ , debido a que al probar la visualización con puntos de datasets geográficos resultó problemático visualizarlos ya que los valores varían muy poco en sus coordenadas haciendo que queden encimados y no se pueda apreciar el resultado. Para la validación del algoritmo con un dataset geográfico se hace uso de gnuplot.

## 0.3 Resultados

Figure 1: Prueba con Polígono1 y umbral de área de 800



```
isaac@irb: ~/Documents/CIMAT/Programación y Algoritmos/Programación-y-algoritmos
$ g++ -std=c++11 visvalingam.cpp plot.cpp test2.cpp -o test -lcairo && ./test data1.txt 800
Umbral de area minima: 800
Porcentaje de compresion logrado: 27.2727%
Porcentaje de error: 5.6338%
```

Figure 2: Polígono1 en azul y Polígono1 simplificado en verde

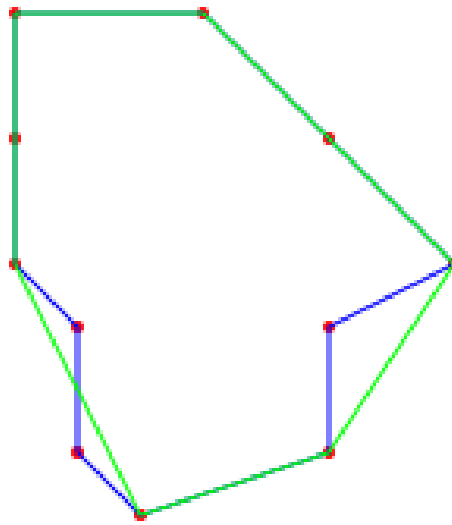


Figure 3: Prueba con Polígono 2 y umbral de área de 910

```
isaac@irb: ~/Documents/CIMAT/Programación y Algoritmos/Programaci-n-y-a  
master g++ -std=c++11 visvalingam.cpp plot.cpp test2.cpp -o test -lcairo && ./  
test data2.txt 910  
Umbral de area minima: 910  
Porcentaje de compresion logrado: 38.4615%  
Porcentaje de error: 3.27869%
```

Figure 4: Polígono2 en azul y Polígono2 simplificado en verde

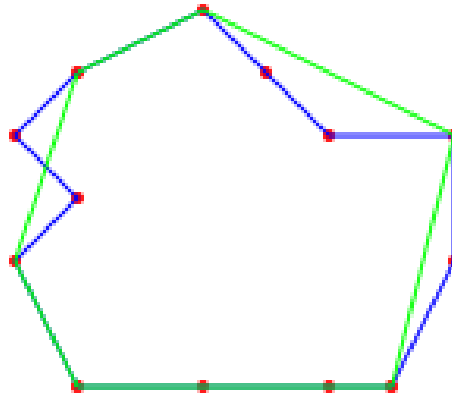


Figure 5: Prueba con Polígono3 y umbral de área de 1000

```
isaac@irb: ~/Documents/CIMAT/Programación y Algoritmos/Programaci-n-y-  
isaac@irb: ~/Documents/CIMAT/Programación y Algoritmos/Programaci-n-y-  
master g++ -std=c++11 visvalingam.cpp plot.cpp test2.cpp -o test -lcairo && ./te  
st data3.txt 1000  
Umbral de area minima: 1000  
Porcentaje de compresion logrado: 25%  
Porcentaje de error: 10.3448%
```

Figure 6: Poligono3 en azul y Poligono3 simplificado en verde

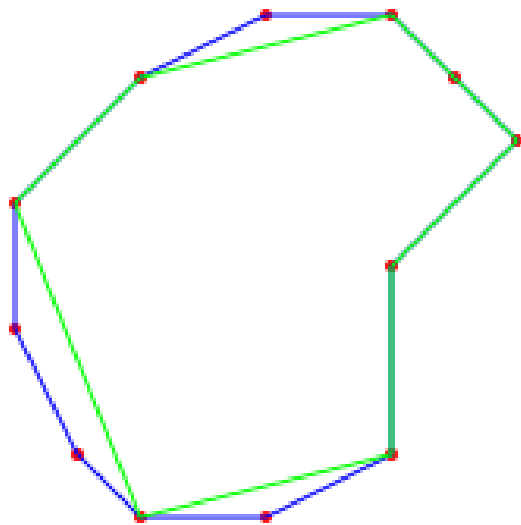


Figure 7: Mapa original del estado de Illinois y umbral de área de 0.003

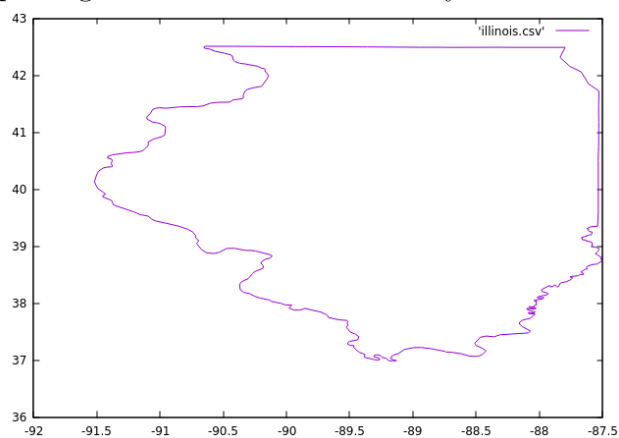


Figure 8: Mapa del estado de Illinois simplificado

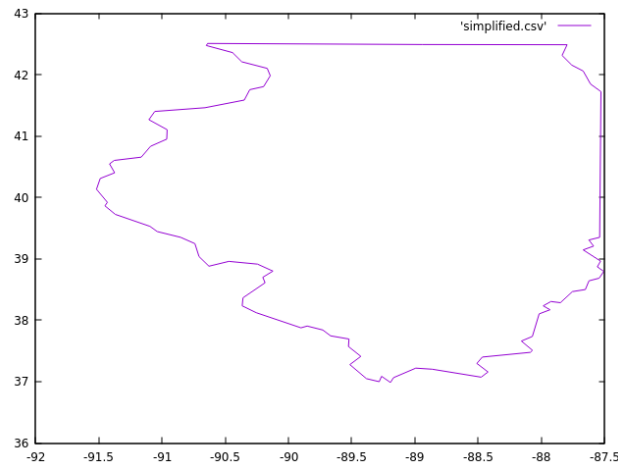
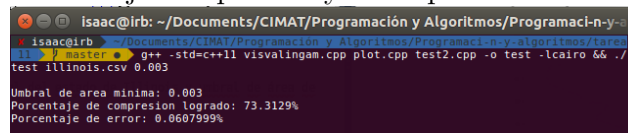


Figure 9: Porcentaje compresión y error para del estado de Illinois



## 0.4 Conclusión

A pesar de ser un algoritmo sencillo me parece muy interesante que se puedan obtener resultados muy buenos como los observados en los ejemplos presentados.

Uno de los puntos del algoritmo implementado que se podría mejorar es el cálculo del error ya que de la manera en que se realizó se asume que el área del polígono simplificado es menor a la del original lo cual no tiene porque cumplirse como se observa en los ejemplos del Poligono1 y Poligono2, donde hay partes del polígono simplificado que quedan fuera del original.