Proyecto Parcial

Programación y Algoritmos

Segmentación de curvas con Ramer Douglas Peucker Visvalingam y BFLS en interfaz gráfica de dibujo.

Kevin Emmanuel Soto Hernández, Centro de Investigación en Matemáticas (Cimat).

Abstract.

En el siguiente documento, se presenta los resultados obtenidos por la implementación de los algoritmos de segmentación de líneas: Ramer Douglas y Visvalingam. Ademas de estos dos se incorpora un nuevo algoritmo de refinamiento que puede ser usado con ambos y mejora sustancialmente los estimadores de preservación de la forma. Estos algoritmos ademas son implementados en una interfaz gráfica de dibujo que puede obtener la forma segmentada por cada uno de los métodos.

1. INTRODUCCION.

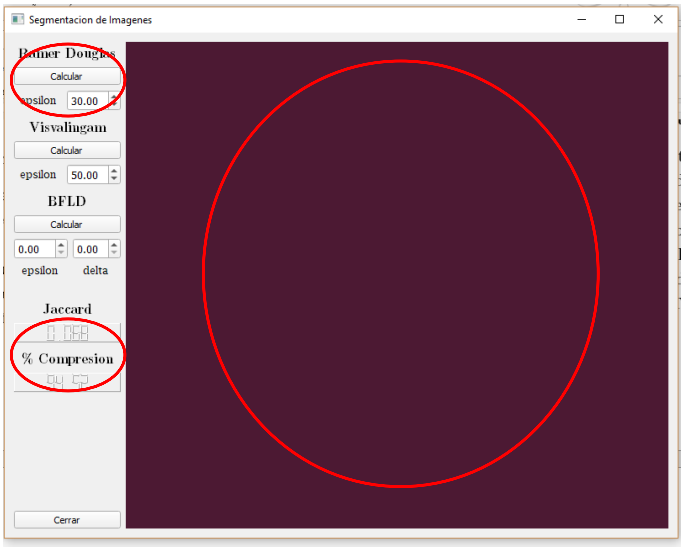
El algoritmo de Ramer–Douglas–Peucker (RDP) es un algoritmo para reducir el número de puntos utilizados en la aproximación de una curva. La forma inicial del algoritmo fue independientemente propuesta en 1972 por Urs Ramer, en 1973 por David Douglas y Thomas Peucker y algunos más en la siguiente década. Este algoritmo también es conocido con el nombre de algoritmo de Ramer-Douglas-Peucker.

El objetivo del algoritmo es, dada una curva compuesta por segmentos, encontrar una curva similar aproximada con menos puntos. El algoritmo define una diferencia basada en la máxima distancia entre la curva original y la curva simplificada. La curva simplificada consiste en una reducción de los puntos que definían la curva original. Se ha observado que algoritmo de Ramer tiene algunos defectos cuando se ele es sometido a ciertos casos de curvas de los cuales se hablará más adelante. Este tipo de problemas es generalmente resuelto por otro algoritmo llamado Visvalingam, del cual hablaremos tambien. El cual toma un enfoque diferente: en lugar de elegir los puntos que mejor se ajusten a la curva, mejor se eliminan aquellos que no representan o que no aportan informacion relevante a la forma local del segmento. Este noble esfuerzo, tambien puede verse truncado por otros inconvenientes por ejemplo que los estimadores de preservación de la forma no son buenos (Jaccard, Dice, etc.); es cuando se propone un nuevo metodo de refinamiento post- segmentación llamado BFLD que usa la línea generada por mínimos cuadrados para obtener más puntos que mejoran estos estimadores.

2. DESARROLLO.

Uso de la Interfaz Gráfica de Dibujo

Comenzando con lo básico, el uso de la interfaz es trivial. Abra el ejecutable. Con el mouse dibuje la curva deseada que se quiere segmentar en el widget (panel morado). Seleccione el metodo de simplificación de líneas (Ramer Douglas, Visvalingam, BFLD). Debajo de cada metodo existe una caja que lee el parámetro ϵ y δ. Teclee el parámetro deseado. Haga click en calcular.



El coeficiente de Jaccard y el porcentaje de compresión se pueden observar en los displays lcd. Para volver a dibujar haga click en cualquier lugar del widget.

Ramer Douglas Peucker

El algoritmo construye una aproximación de la curva inicial mediante un proceso recursivo. Se toma como solución inicial el segmento que une los dos puntos extremos de la curva. Entonces, se busca el punto más alejado de dicho segmento (peor punto).

Si el peor punto está más cerca del segmento que el umbral de distancia ε, entonces se termina el proceso. Es seguro que el resto de puntos de la curva están a menor distancia que el umbral ε, y por lo tanto todos los puntos de la curva (salvo los extremos) pueden ser descartados.

Si el peor punto está más alejado que ε, entonces ese punto debe permanecer en la simplificación. El algoritmo hace dos llamadas recursivas a sí mismo para calcular la aproximación de dos curvas de menor longitud. Una con los puntos entre el primer y el peor punto y otra con los puntos entre el peor punto y el punto final de la curva.

Cuando se completa la recursión la nueva curva puede ser generada a partir de los puntos que han permanecido tras haber aplicado el algoritmo. A continuación el psuedocódigo del algoritmo Ramer Douglas Peucker implementado en nuestro código.

Procedure RAMER DOUGLAS PEUCKER (RDP)

$$\textbf{input}: list[\,],\ \epsilon$$

$$dmax, index, end \leftarrow 0, 0, \textbf{size}(list)$$

$$\textbf{for}\ i = 2\ to\ end - 1: \{$$

$$dist \leftarrow \textbf{shortest distance}\ (list_i, list_1, list_{end})$$

$$\textbf{if}\ (dist > dmax): index, dmax \leftarrow i, dist\}$$

$$\textbf{if}\ (dmax > \epsilon): \{$$

$$leftslice, righslice \leftarrow \textbf{rdp}(list_{1\ldots index}, \epsilon)\ , \textbf{rdp}(list_{index+1\ldots end},\ \epsilon)\}$$

$$\textbf{else}: \textbf{return}\ [list_1, list_{end}]$$

$$\textbf{return}\ \textbf{merge}(leftslide, rightslide)$$

Sin embargo, detrás de este sencillo algoritmo se encuentran otros más técnicos como el que realiza la lectura de la imagen, la impresión de la imagen final, la búsqueda de pixeles en blanco y el DFS que se encarga de buscar todos los pixeles conectados. En este último algoritmo, se observó que no era robusto a ciertos tipos de curvas peleadas pues estas contenían características que hacían que DFS se ciclará, no encontrara un punto extremo o encontrara un punto extremo erróneo. Hablamos específicamente del problema de los "surcos", es decir, puntos pixeles en donde la curva parece plagarse y que contienen tres o más vecinos (de los 8 posibles). Es curioso que cuando se trata de recorrer a los ocho vecinos más cercanos de un pixel, pueden ocurrir los errores descritos atrás. Pero si el recorre en forma de cruz (+) o en forma de cruz inclinada (x) siendo necesariamente primero probar con cruz; este bug desaparece e incluso puede trabajar con otras geometrías como los círculos (Véase Fig. 4) e incluso si se cambian unas cuantas línea de código puede trabajar con curvas bifurcadas. Toda este proceso es implementado en la funcion

find_extrems() de la clase *Segmentation.* Véase el código para mejor informacion de cómo es implementado.

Visvalingam

El algoritmo fue propuesto por Visvalingam & Whyatt 1993. El principio del algoritmo es seleccionar los vértices a eliminar (los menos característicos) en lugar de elegir los vértices a mantener (en el algoritmo de Douglas y Peucker). Para seleccionar los vértices a eliminar, hay un proceso iterativo, y en cada iteración, se calculan los triángulos formados por tres vértices consecutivos. Si el área del triángulo más pequeño es más pequeña que un umbral (parámetro ϵ), se elimina el vértice medio y se inicia otra iteración. A continuación el algoritmo:

```
Procedure: VISVALINGAM

input: list[vertex], ϵ

if( size(list) < 3) return

max ← ∞

for i = 1 to end:

   list[i].prev ← list[i-1].vertex

   list[i].next ← list[i+1].vertex

   list[i].area ← triangarea(list[i-1].vertex,
list[i].vertex, list[i+1].vertex)

do{

    min_area  ← max-1;

    for i =1 to end

        if(list[i].area < min_area)

            min_area ← list[i].area

            index_min ← i

    list[index_min].area  ← max

    index_prev ← list[index_min].prev;

    index_next ←list[index_min].next;

    #Actualizar vecinos

    list[index_prev].next ← index_next

    list[index_next].prev ← index_prev

#Actualizar area de vecinos

    list[index_prev].area ← triangarea(
list[list[index_prev].prev],
list[list[index_prev].vertex],
list[list[index_prev].next])

    list[index_next].area ← triangarea(
list[list[index_next].prev],
list[list[index_next].vertex],
list[list[index_next].next])

} while(min_area < ϵ)

#Eliminar los que tienen área más grande que ϵ

for i = 1 to end:

    if ( list[i].area == max): delete list[i]

return list
```

Best Fitting Line δ

El proceso de simplificación de puntos con Visvalingam y RDP en si mismos, son eficientes, robustos, rápidos, adaptativos a la forma de la curva, etc. Mientas que el porcentaje de compresión es alto para un épsilon pequeño, es cierto que algunas ocasiones los estimadores de la conservación de la forma (Jaccard, Dice, etc.) no son los ideales si se habla de precisión. Es decir, lo que indican estos estimadores es que existen ciertos pedazos de la curva que no estan encajando perfectamente con la recta de los puntos de simplificación. Por ejemplo, con RDP es posible que al tratar de simplificar hundimientos en la frontera de la curva, las rectas trazadas por los puntos de simplificación se puedan intersectar perdiendo la continuidad.

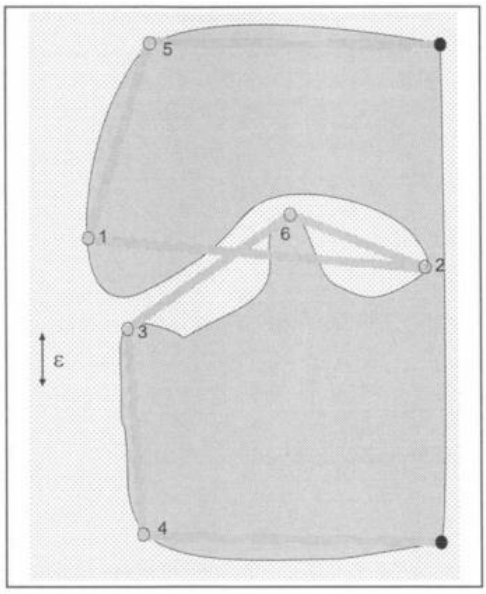


Fig. El algoritmo Ramer Douglas podría producir auto intersección.

Otro evento que se puede presentar es cuando en algunos tramos del segmento existen ondulaciones pequeñas donde, para un cierto ϵ grande, es pasado por alto y causar una pérdida importante de la informacion. La forma trivial de corregir este percance es disminuir el parámetro ϵ pero esto tambien significa decaer el porcentaje de reducción drásticamente de manera global pues no importa si la curva es suave en todo el resto del segmento, RDP ya no obtendrá una reducción notable.

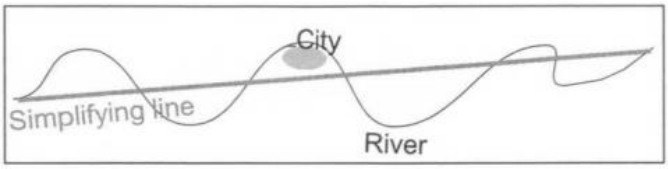


Fig. Un algoritmo de simplificación podría localizar un punto en el sitio incorrecto.

Por eso y otras razón **The National Oceanic Service** (Estados Unidos) prohibió el uso de RDP en la reducción de los mapas cartográficos de la costa del pacifico y atlántico.

Visvalingam, por su parte, ofrece mejores resultados pero como veremos más adelante, no será suficiente. Surgió entonces la idea de un refinamiento post-reducción con algunos de los algoritmos anteriores.

La idea a priori del refinamiento es sencilla: Sobre un segmento se ejecuta la reducción con Visvalingam/RDP para un ϵ considerable (i.e. la épsilon elegida debe de tener la sutilidad de dar un alto porcentaje de compresión por arriba de 90% por ejemplo; y que esté en equilibrio con un buen coeficiente de Jaccard: arriba de 0.5 por ejemplo). Después se busca entre aquellos puntos de reducción, todos los sub-segmentos que tengan un coeficiente de Jaccard menor que un δ (evidentemente $\delta \in [0,1]$). Sobre cada sub-segmento se calcula la mejor línea de encaje, esta línea es la obtenida con mínimos cuadrados. A continuación un ejemplo visual de como debería de funcionar el algoritmo con la funcion seno.

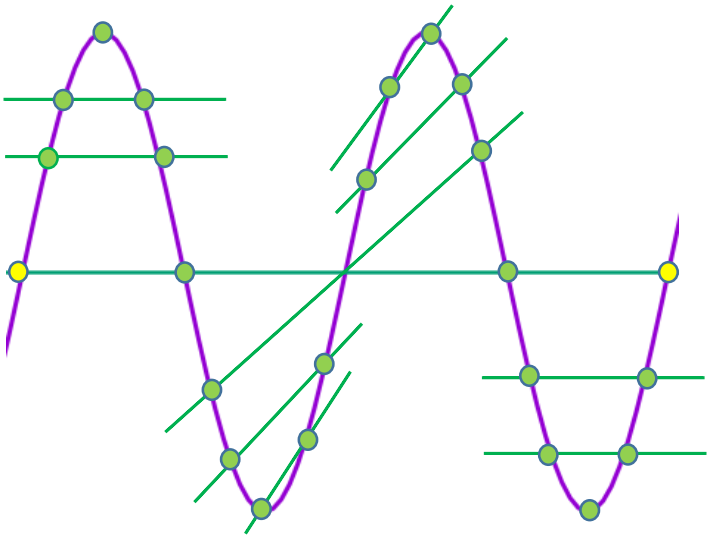


Fig. Ejemplo visual de la segmentación de BFLD con la funcion seno

Observe que no importa la forma del segmento, la recta debe de cortar en minimo dos puntos siempre y cuando los puntos del segmento sean continuos (esta condición es muy importante ya que determina si el algoritmo funciona o no. Como veremos, BFLD no funciona en la interfaz gráfica debido a esto).

Los punto de corte más cercanos a los extremos, serán los nuevos extremos de tres sub-segmentos porque intuitivamente la recta que pasa por ellos es tiene el mayor coeficiente de Jaccard. El proceso de corte se hace manera recursiva hasta que todos los segmentos tienen un coeficiente de Jaccard mayor o

igual a δ; o en su defecto cuando el segmento tiene 4 o menos puntos (en este caso se retorna el punto del medio). A continuación el algoritmo:

Procedure BEST FITTING LINE DELTA (BFLD)

input: $list[\,], \epsilon, \delta$

$dmax, index, end \leftarrow 0, 0, \textbf{\textit{size}}(list)$

if $(jaccard(list) \geq \delta)$:

  return $(list[1], list[end])$

else if $(size(list) \leq 4)$:

  return $list[mid]$

else:

  $m, b \leftarrow regression(list)$

  for $i = 2$ *to* $end - 1$:

    if $list[i] \rightarrow y - \frac{1}{2} \leq m * list[i] \rightarrow x + b \leq list[i] \rightarrow y + \frac{1}{2}$:

      $index1 = i, \quad break$

  for $i = end - 1$ *to* 1:

    if $list[i] \rightarrow y - \frac{1}{2} \leq m * list[i] \rightarrow x + b \leq list[i] \rightarrow y + \frac{1}{2}$:

      $index2 = i, \quad break$

  $slice1 \leftarrow list[0: index1]$

  $slice2 \leftarrow list[index1: index2]$

  $slice3 \leftarrow list[index2: end]$

  $out1 \leftarrow bfld(slice1, \epsilon, \delta)$

  $out2 \leftarrow bfld(slice2, \epsilon, \delta)$

  $out3 \leftarrow bfld(slice3, \epsilon, \delta)$

  return ***merge***$(out1, out2, out3)$

3. RESULTADOS.

La siguiente imagen fue creada en GIMP, cargada y segmentada con ambas Visvalingam y BFLD + Visvalingam con $\epsilon = 5$ y $\delta = 0.5$. Debajo de cada imagen se muestra el porcentaje de reducción y el coeficiente de Jaccard.

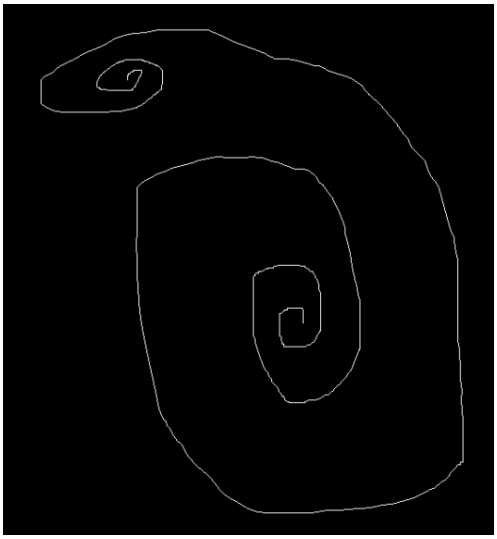


Fig 1. Figura de curva simple con surcos

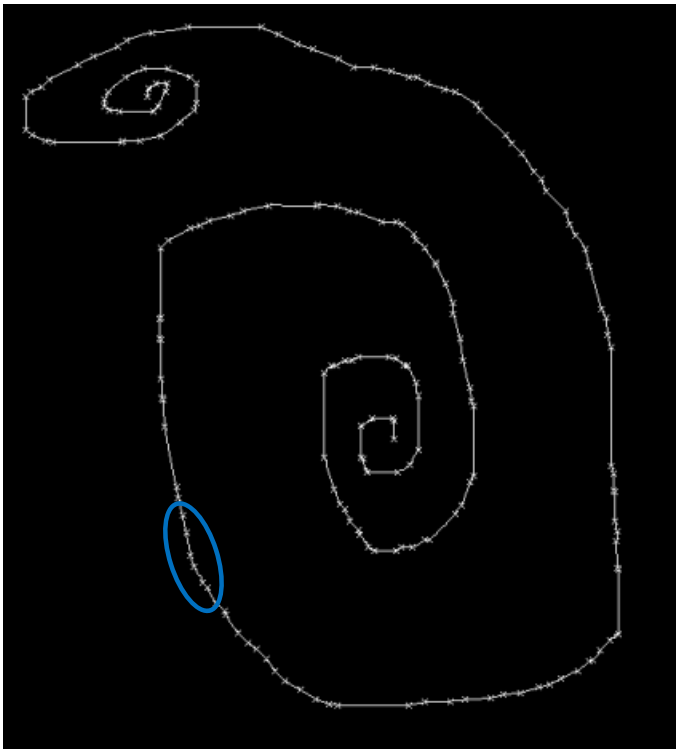


Fig 2. Figura de curva simple con surcos, segmentada y reducida

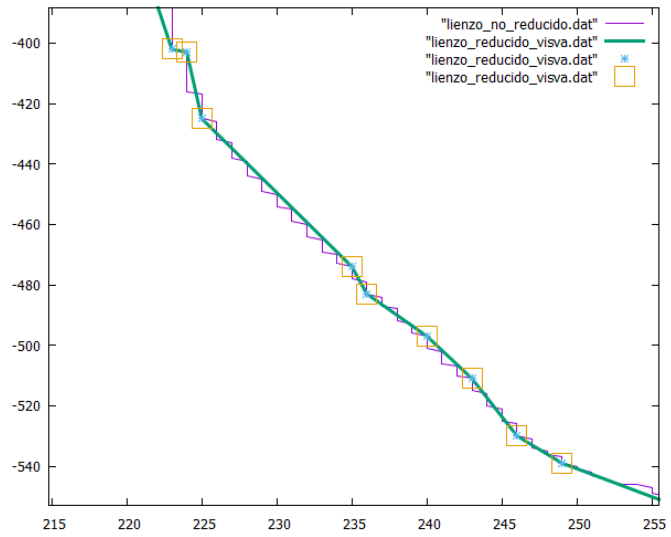


Fig 3. Zoom in del circulo azul de la curva reducida con Visvalingam. Note que la recta verde no encaja con los surcos en zic-zac

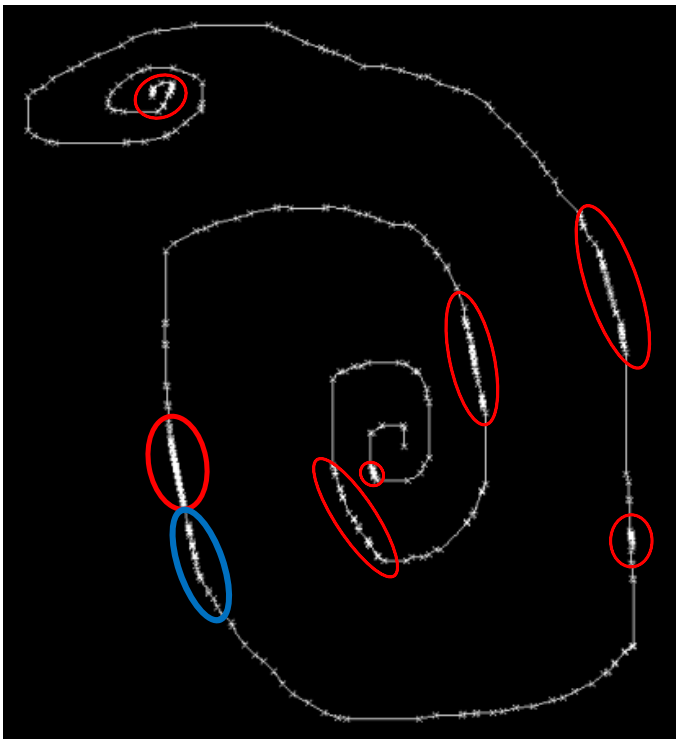


Fig 4. Figura de curva reducida con BFLD + Visvalingam.

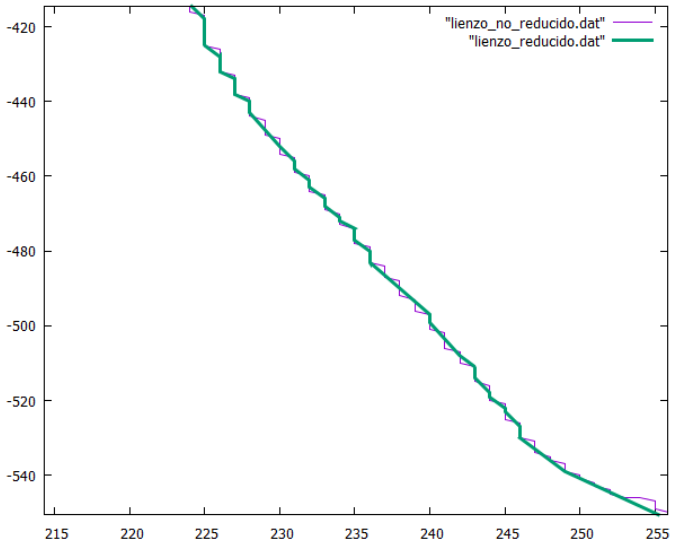


Fig 5. Zoom in del circulo azul de la curva reducida con BFLD - Visvalingam. Observe que la sobreposición de la línea verde esta casi por encima de la morada.

Para otro ejemplo con $\epsilon = 3, \delta = 0.5$ se obtuvo:

Visvalingam:

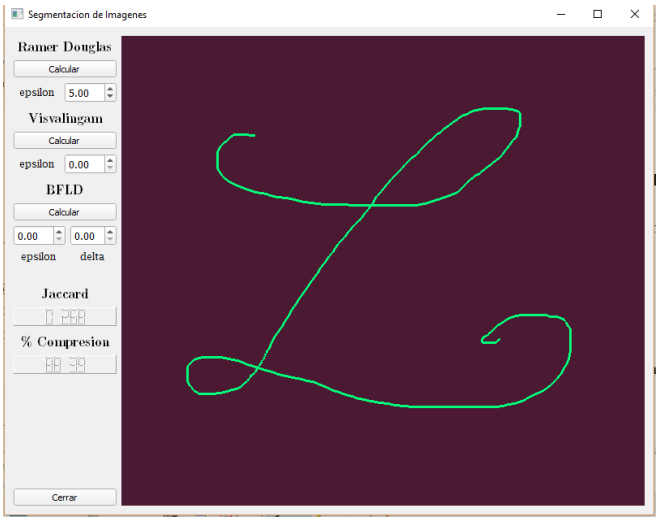
```
El porcentaje de compresion del segmento es 88.3873 %
El coeficiente de jaccard es: 0.820167
```

BFLS + Visvalingam

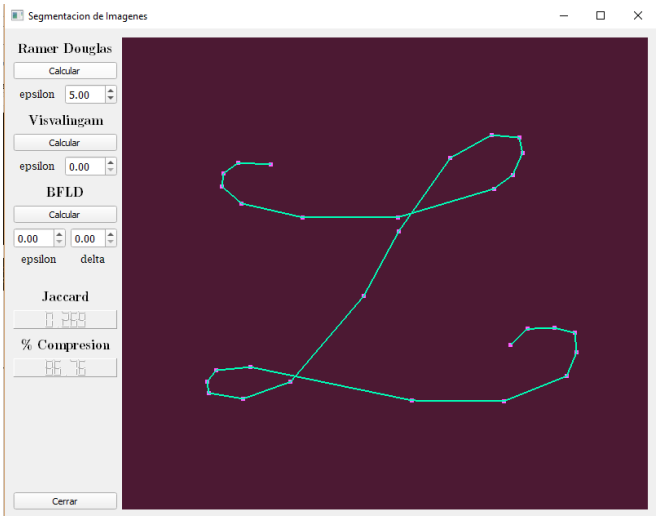
```
El porcentaje de compresion del segmento es 84.0189 %
El coeficiente de jaccard es: 0.867856
```

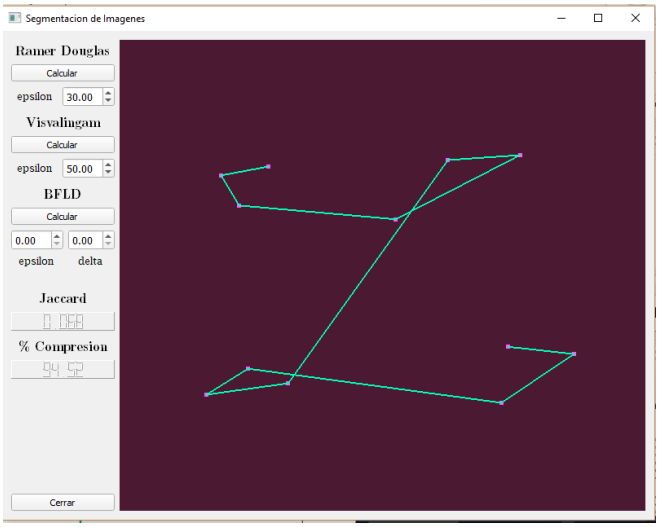
Interfaz Grafica

Dibujo sencillo de curva



Segmentación con los tres algoritmos





Se observó que el algoritmo BFLD no funciona en muchas ocasiones debido a la discontinuidad en el dibujo de la curva. El problema radica en cuando se trata de encontrar las intersecciones de la recta generada por medio de mínimos cuadrados con el sub-segmento. Cuando no haya una de estas intersecciones, la aplicación se cierra inesperadamente.

Para correr BFLD dibuje una línea continua con GIMP y desde la terminal ejecute el archivo main_seg.cpp con el nombre de su imagen.

4. CONCLUSIONES.

Como se venía advirtiendo, BFLD puede mejorarse en muchas cosas, un de ellas es la robustez frente a las curvas bifurcadas. Que no pasa de tener unas cuantas diferencias con respecto al código entregado. Otra mejora, más interesante en complejidad, sería que el algoritmo fuera robusto frente a curvas de diferente grosor.

En adición, se observó en diversos experimentos que BFLD tenía un mejor coeficiente de Jaccard comparado con simplemente implementar Visvalingam para el mismo valor de épsilon. Sin embargo, el porcentaje de compresión se ve afectado por este proceso, aunque no drásticamente (alrededor de un 7% en el peor de los casos estudiados).La elección entre Visvalingam y BFLD depende totalmente en la aplicación donde es implementado. Si se busca una alta preservación de la forma, por ejemplo al segmentar la frontera de los mapas cartográficos, es conveniente usar BFLD. Pero para otros casos en donde lo que se busca es compresión, se debería de usar Visvalingam.

5. Referencias

[1] Charles E. Leiserson, Clifford Stein, Ronald Rivest y Thomas H. Cormen (2001), Introduction To Algorithms, 3th Edition

[2] Steven Halim & Felix Halim (2013), Competitive Programming 3.

[3] Alan Saalfeld (2013). Topologically Consistent Line Simplification with the Douglas-Peucker Algorithm. Cartography and Geographic Information Science