Propuesta de paradigma para análisis geométricos en SIG independiente del sistema de referencia

Proposal of paradigm for GIS geometric analysis based on a generic algorithm independent from the reference system

Antúnez Romanuel Ramón y Hernández Montero Lidisy[1]

Recibido: octubre, 2012 / Aceptado: enero, 2013

Resumen

La mayor parte de los análisis espaciales realizados en los sistemas de información geográfica (SIG) hacen uso de cálculos geométricos sencillos, a partir de los cuales se construyen algoritmos más complejos. Tradicionalmente estos cálculos se han desarrollado según métricas euclidianas; sin embargo, la información geográfica ha ganado en disponibilidad y exactitud planimétrica, lo que ha provocado que estos cálculos realizados en los SIG sean poco precisos. Estos errores pudieran llegar a ser notablemente importantes en algunos casos particulares, tales como en ámbitos muy extensos o proyecciones poco equidistantes. Por tal motivo en este trabajo se propone un conjunto de algoritmos geométricos apoyados en el diseño de un algoritmo genérico, que tenga en cuenta el sistema de coordenadas y/o proyección con la que se trabaja, que permitan un correcto y eficiente funcionamiento de las funcionalidades para el cálculo de distancia, perímetro y área en aplicaciones SIG.

Palabras clave: Algoritmo; análisis geométrico; geometría computacional; SIG.

Abstract

Most spatial analysis undertaken on Geographic Information Systems (GIS) use simple geometric calculations, from which more complex algorithms are built. Traditionally, these calculations have been developed according to Euclidean metric, but have won geographic information on availability and planimetric accuracy what has caused these calculations in GIS be inaccurate. These errors could become significantly important in some particular cases, such as in very long or little equidistant projections. That is why in this paper, a set of geometric algorithms is proposed, supported by the design of a generic algorithm that takes into account the coordinate system and / or projection on which it is worked, to enable proper and efficient administration of the functions to calculate distance, perimeter and area on GIS applications.

Key words: Algorithm; geometric analysis; computational geometry; GIS.

1 Universidad de las Ciencias Informáticas, Departamento de Geoinformática, Centro de Desarrollo GEYSED, Santiago de Cuba, Cuba. Correo electrónico: rramon@uci.cu

1. Introducción

Gran parte de la información manejada a nivel empresarial y/o gubernamental presenta una estrecha relación con datos espaciales; por tal motivo, la toma de decisiones y la precisión de éstas están condicionadas, en gran medida, por la calidad, exactitud y actualización de esta información espacial, convirtiéndose entonces los sistemas de información geográfica en herramientas vitales para el apoyo en dicha toma de decisiones.

Una de las disciplinas computacionales que presenta sus aportes de una forma bastante inmediata en estos sistemas es la Geometría Computacional. Esta disciplina proporciona criterios para detectar y organizar estructuras geométricas, así como su representación en pantalla. Desarrolla herramientas computacionales para el análisis de problemas geométricos y propone estrategias para implementar algoritmos eficientes –en cuanto a tiempo de ejecución, preferiblemente igual o menor a la clase *O* (*nlogn*)– que faciliten la resolución efectiva de estos problemas computacionales.

Generalmente, en esta disciplina los problemas más tratados y aplicados en los SIG han sido trabajados según métricas euclidianas. Por otro lado, la información geográfica ha ganado en disponibilidad y exactitud planimétrica lo que ha provocado que estos cálculos, realizados en los SIG mediante métricas euclidianas, sean poco precisos (Pesquer *et al.*, 2005).

Estos errores pudieran llegar a ser notablemente importantes en algunos casos particulares, tales como en ámbitos muy extensos o proyecciones poco equidistantes. Por consiguiente, se hace necesario sustituir en aquellas herramientas SIG los clásicos cálculos euclidianos por otros que usen algoritmos geodésicos.

Para un correcto desempeño de estas funcionalidades es esencial analizar la calidad de las bases cartográficas que participan, las características del sistema de referencia –básicamente proyección y modelo elipsoide o esférico–, las dimensiones del ámbito y sus objetivos –resolución de las bases resultantes, precisión planimétrica y temática requerida, entre otras. En función de estos parámetros, las herramientas de análisis SIG deberían facilitar al usuario no experto en estos temas una solución válida de forma automática y, al usuario avanzado, un control adecuado para hacer valer su decisión de usar unos métodos, euclidianos, u otros, geodésicos. Sería absurdo obligar al usuario a usar siempre y en cualquier condición los métodos más precisos, pero mucho más lentos de la geodesia (Pesquer *et al.*, 2005).

En el marco del presente trabajo, se propone un conjunto de algoritmos para cálculos geométricos apoyados en el diseño de un algoritmo genérico, que tenga en cuenta el sistema de coordenadas y/o proyección con la que se trabaja y, específicamente, el desarrollo de un nuevo algoritmo basado en la geometría esférica para determinar el área de polígonos esféricos sin necesidad de utilizar modelos geodésicos para su re-proyección y/o eliminando restricciones a la hora de la digitalización de dichos polígonos.

2. Cálculos geométricos en los SIG

Según Olaya (2010), la mayor parte de los análisis espaciales realizados en los SIG hacen uso de cálculos geométricos sencillos, a partir de los cuales se construyen algoritmos más complejos.

Se plantea que esto se debe a que prácticamente todos los cálculos geométricos que se realizan en estos sistemas –fundamentalmente los realizados sobre capas vectoriales– se basan en la posición y las relaciones topológicas entre objetos.

2.1 Cálculos geométricos según métricas euclidianas

La idea de distancia es fundamental para todo análisis espacial, pues en casi todos los procedimientos se incluye este concepto. En el plano, la distancia euclídea entre dos puntos viene dada por (Alonso Sarría, 2006):

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Además de calcular la distancia entre dos puntos puede calcularse entre geometrías. Es el caso entre 2 rectas en el plano es igual a la distancia entre un punto cualquiera sobre la primera hasta otro punto ortogonal a él en la segunda, siempre y cuando sean paralelas, en caso contrario la distancia es nula, pues siempre existirá un punto en que ambas se encuentran (Olaya, 2010).

Sin embargo, en los SIG no suele trabajarse con rectas de longitud infinita en el sentido matemático, sino con segmentos de éstas.

La distancia de un segmento definido por sus extremos (x_1, y_1) y (x_2, y_2) a un punto (x_3, y_3) se calcula como la distancia de este punto hasta el punto sobre el segmento en el que se intercepta la recta que pasa por el punto (x_3, y_3) y es perpendicular al segmento en cuestión (Alonso Sarría, 2006).

De esta manera, se puede obtener igualmente la distancia de un punto a un polígono, siendo la distancia entre el punto y la línea que define el perímetro de dicho polígono.

Para el caso de los polígonos, son dos magnitudes principales las que se analizan, área y perímetro. Para el caso del área se calcula de la siguiente forma:

$$A = \left| \frac{1}{2} \sum_{i=1}^{n} x_i y_{i+1} - x_{i+1} y_i \right|$$

Para aquellos polígonos que contengan huecos basta con restar al área calculada, la correspondiente al hueco y ésta se calcula de igual manera, pues los huecos se definen al igual que los polígonos, como una polilínea cerrada (Olaya, 2010).

El perímetro de un polígono no es más que la suma de las distancias entre vértices adyacentes, es decir:

$$P = \sum_{i=1}^{n} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

2.2 Cálculos geodésicos

Los cálculos presentados hasta el momento por Alonso Sarría (2006) y Olaya (2010) son referidos al plano, y es como tradicionalmente se han realizado en los SIG. Sin embargo, cuando se trabaja con un sistema de coordenadas diferente a

las cartesianas estos resultados son completamente erróneos. En Pesquer *et al.* (2005) se plantean tres algoritmos para solucionar estos problemas que pueden generar los cálculos euclidianos.

El **Problema Inverso de la Geodesia** consiste en determinar el acimut y la longitud de la línea geodésica que separe dos puntos sobre el elipsoide. Existen múltiples métodos para su resolución, pero uno de los más empleados en la literatura científica es el método iterativo de Bessel, el cual se explica en Zakatov (1981).

El **Problema Directo de la Geodesia** consiste en determinar un punto destino problema a partir de un punto origen, una distancia y un acimut conocidos. Para éste también existen diversos métodos conocidos para su resolución siendo uno de los más utilizados la integración de Runge-Kutta de 4to orden, también explicado en Zakatov (1981).

El tercer algoritmo planteado por Pesquer *et al.* (2005) es referido al cálculo de áreas, donde se propone transformar las coordenadas de los vértices del polígono a analizar a una proyección equivalente y se propone, específicamente, una Cilíndrica Equal-Área, donde luego se puede aplicar la ecuación planteada en Olaya (2010).

Para la transformación de las coordenadas se emplean las siguientes ecuaciones (Pesquer *et al.*, 2005):

$$X = ak_0(\lambda - \lambda_0) \quad Y = aq/(2k_0)$$

$$k_0 = \frac{\cos(\phi_s)}{\sqrt{1 - e^2 \sin^2(\phi_s)}}$$

$$q = (1 - e^2)\left[\frac{\sin(\phi)}{1 - e^2 \sin^2(\phi)} - \left(\frac{1}{2e}\right)\ln\left(\frac{1 - e\sin(\phi)}{1 + e\sin(\phi)}\right)\right]$$

2.3 Cálculos geométricos sobre la esfera

Además de los algoritmos propuestos en Pesquer *et al.* (2005), otros autores han hecho referencia a métodos de naturaleza más simple para determinar la distancia entre dos puntos y el área de un polígono sobre una superficie esférica.

Entre los métodos para determinar la distancia entre dos puntos sobre la superficie terrestre se tiene el caso de la ley de cosenos para trigonometría esférica (Iglesias *et al.*, 1996).

$$a = \sin(lat_1) * \sin(lat_2)$$

$$b = \cos(lat_1) * \cos(lat_2) * \cos(lon_1) * \cos(lon_2)$$

$$c = \cos^{-1}(a + b)$$

$$d = R * c$$

Aunque esta fórmula es matemáticamente exacta, varios autores plantean que es poco fiable computacionalmente para distancias pequeñas, pues el arco coseno puede introducir considerables errores.

Otro método es el empleo de la fórmula en coordenadas polares para el modelo de tierra plana:

$$a = \frac{\pi}{2} - lat_1$$

$$b = \frac{\pi}{2} - lat_2$$

$$c = \sqrt{a^2 + b^2 - 2 * a * b * cos\,(lon_2 - lon_1)}$$

$$d = R * c$$

Esta nueva fórmula proporciona errores máximos más pequeños que la aproximación pitagórica para latitudes elevadas y grandes distancias.

Una tercera propuesta para calcular la distancia es la fórmula de Haversine (Mencia, 2006):

$$dlon = lon_2 - lon_1$$

$$dlat = lat_2 - lat_1$$

$$a = sin^2\left(\frac{dlat}{2}\right) + \cos(lat_2) * \cos(lat_2) * sin^2\left(\frac{dlon}{2}\right)$$

$$c = 2 * sin^{-1}(\min(1.0, \sqrt{a}))$$

$$d = R * c$$

La fórmula de Haversine proporciona resultados matemática y computacionalmente exactos. El resultado intermedio *c* es la distancia en radianes sobre el gran círculo. Mientras la distancia *d* estará en las mismas unidades que *R*.

La aplicación del mínimo protege de posibles errores de redondeo que podrían afectar la computación del arcoseno si los dos puntos son antipodales, en lados opuestos de la Tierra. Bajo estas condiciones, la fórmula de Haversine tiene resultados adversos, pero el error que puede ser de hasta dos kilómetros está en el contexto de una distancia de cerca de 20.000 kilómetros, es decir cuatro órdenes de diferencia.

Para el caso del cálculo del área de polígonos simples sobre la esfera, Bevis y Cambareri (1987) proponen un algoritmo basado en la formulación del área de polígonos esféricos.

$$S = \frac{\pi R^2}{180}(A_1 + A_2 + \cdots + A_n - (n-2) * 180)$$

En radianes quedaría:

$$S = R^2(A'_1 + A'_2 + \cdots + A'_n - (n-2) * \pi)$$

Dicha fórmula solo depende de la amplitud de los ángulos interiores del polígono y es precisamente esta amplitud la que intenta calcular el algoritmo presentado por Bevis y Cambareri (1987). Sin embargo, para poder garantizar que se midan exactamente los ángulos interiores se impone una restricción en este algoritmo, y es la necesidad de que el polígono se haya digitalizado en sentido horario, en caso contrario deberían ordenarse los vértices del polígono a analizar obteniéndose una cota superior de complejidad para el algoritmo de *nlogn*.

2.4 Aplicación de métodos geométricos en SIG actuales

La mayoría de los sistemas de información geográfica libre actualmente disponibles para la comunidad, tales como *QuantumGIS*, posibilitan herramientas para cálculos geométricos básicos, principalmente en herramientas para el cálculo de distancias entre puntos y áreas de polígonos. Estas funcionalidades se basan en los métodos planteados en el epígrafe 2.1, por lo que al trabajar en coordenadas geográficas las distancias y áreas quedan expresadas en grados y grados cuadrados respectivamente, unidades que no son válidas para estas mediciones.

Otras herramientas libres como *Postgis* –extensión de soporte espacial para

PostgreSQL– presentan algunas alternativas a este problema. Tal es el caso de las funciones de *Postgis ST_Distance*, para el cálculo de distancia cartesiana entre geometrías basado en el método para la distancia presentado en el epígrafe 2.1, *ST_Distance_Sphere*, para el cálculo de la distancia entre geometrías sobre la esfera basado en el método de Haversine, además de las funciones *ST_Azimuth* para determinar el acimut entre dos puntos y *ST_Area* para determinar el área de un polígono. Esta última función está basada en el método presentado en el epígrafe 2.1 para el área de polígonos en el plano, por lo que al trabajar con coordenadas geográficas debe utilizarse la función *ST_Transform* para proyectar estas coordenadas a una proyección equivalente (Refractions Research, 2008).

En el ámbito propietario la mayoría de los sistemas, como es el caso de *MapInfo*, sí proveen herramientas capaces de solucionar estos problemas, pues posibilitan funciones para el cálculo de la distancia tanto cartesiana como esférica, opción que puede escoger el usuario siempre que sea posible. Para el caso del cálculo del área de polígonos proceden de forma similar pues implementan funciones para el área en el plano y el área en la esfera. Pero, al ser aplicaciones de código cerrado no se puede tener certeza si se basan en algunos de los métodos planteados anteriormente o en métodos propios.

3. Descripción de la solución

Una proyección cartográfica se puede definir como una biyección; luego por teorema se tiene que si f es una función biyectiva entonces f^{-1} existe y es también biyectiva (Weisstein, 2011).

Partiendo de este teorema es evidente que si se trabaja con coordenadas proyectadas es posible obtener las coordenadas elipsoidales correspondientes en el elipsoide de referencia del cual se partió.

El algoritmo genérico propuesto se basa en esta propiedad y teniendo en cuenta que en un sistema de información geográfica siempre es conocida la proyección y/o sistema de coordenadas en el que se está trabajando, la idea que se presenta es la siguiente:

- Si se está trabajando en un sistema proyectado entonces se puede trabajar con algoritmos para coordenadas planas o algoritmos para coordenadas elipsoidales con una previa transformación de las coordenadas planas rectangulares a elipsoidales.
- Esta decisión puede ser tomada por el usuario en caso de ser experto; por defecto se trabajaría convirtiendo las coordenadas a elipsoidales.
- En caso de no estar trabajando en un sistema proyectado entonces se trabajaría con los algoritmos para las coordenadas elipsoidales.

El algoritmo genérico en pseudocódigo quedaría de la siguiente forma:

Algoritmo Genérico

```
function X (x: type)
if projection do
        if cartesian do
                return CartesianX(x: type)
        end if
        return SphericalX(Transform(x: type))
end if
      return SphericalX(x: type)
end function
```

La variable *projection* es un booleano que tomaría valor verdadero en caso de estar trabajándose con un mapa proyectado y falso en caso distinto, mientras la variable *cartesian* sería otro booleano que se inicializaría en falso para trabajar por defecto con el algoritmo para las coordenadas elipsoidales, pero que puede cambiar su valor a petición del usuario.

3.1 Algoritmo para el cálculo de distancias

Una vez diseñado el algoritmo genérico para el análisis geométrico en SIG, se procede entonces a proponer y diseñar en consecuencia al mismo, los algoritmos para el cálculo de distancias, perímetros, acimut y áreas.

Partiendo de que el determinar la distancia entre dos puntos será vital para los demás algoritmos se comienza entonces por la propuesta para este problema particular.

Para el diseño del algoritmo teniendo en cuenta el algoritmo genérico anteriormente planteado se deben proponer dos métodos para determinar la distancia entre dos puntos; un primer método para trabajar con coordenadas planas y un segundo método para trabajar con coordenadas elipsoidales.

3.1.1 Método para coordenadas planas

Para el caso de coordenadas planas el problema es de resolución trivial y fue presentado en el capítulo anterior donde se plantea que viene dada por la fórmula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Por lo que el algoritmo en pseudocódigo sería bastante simple como se muestra en el algoritmo uno:

Algorithm 1. Cálculo de distancia en el plano

```
function CartesianDistanceCalculation
        (_pto1, _pto2: Point)
return sqrt
(pow((_pto2.x – pto1.x), 2) + pow((_pto2.y – pto1.y), 2))
end function
```

Donde la función ***sqrt*** es la encargada de devolver el valor de la raíz cuadrada del número que recibe como parámetro, y la función ***pow*** es la encargada de elevar el primer número que recibe como parámetro a la potencia indicada por el segundo parámetro.

3.1.2 Método para coordenadas elipsoidales

Para esta instancia del problema se vieron diversos métodos, tanto iterativos como directos en el apartado anterior. Sin embargo, en esta investigación se hará mayor énfasis en los métodos directos, pues por su naturaleza suelen ser más eficientes en cuanto a tiempo de ejecución que los iterativos.

Luego de analizar las variantes presentadas en el apartado anterior se propone emplear la fórmula de Haversine, debido a que presenta una mejor estabilidad computacional respecto a las otras, además de ser uno de los métodos más ampliamente usados y citados en la literatura especializada. En el algoritmo dos se muestra su pseudocódigo.

Algorithm 2. Cálculo de distancia sobre la esfera

```
function SphericalDistanceCalcutation
          (_pto1, _pto2: Point, _radian: bool)
          _radio ← 6 378 400
          _dlon ← _pto2.x - _pto1.x
          _dlat ← _pto2.y - _pto1.y
          _intcal ← pow(sin(_dlat/2),2) +
          cos(_pto1.y) * cos(_pto2.y)
          * pow(sin(_dlon/2),2)
          _intd ←  2 * arcsin(min(1.0,sqrt(z)))
          if _radian do
                    return _intd
          end if
          return _radio * _intd
end function
```

3.1.3 Pseudocódigo de algoritmo genérico para la distancia

Finalmente ya se tienen los métodos tanto para trabajar con coordenadas planas como en coordenadas elipsoidales; por lo que se está en condiciones de presentar el diseño completo del algoritmo.

Algorithm 3. Algoritmo genérico para el cálculo de distancia

```
function CalculateDistance (_pto1, _pto2: Point)
     if projection do
          if cartesian do
     return CartesianDistanceCalculation (_pto1, _pto2)
     end if
     return  SphericalDistanceCalcutation(Transfo
     rm(_pto1), Transform(_pto2), true)
     end if
return SphericalDistanceCalculation(_pto1, _pto2, true)
end function
```

3.2 Algoritmo para el cálculo de perímetros

Una vez elaborados las propuestas y el diseño del algoritmo genérico para determinar la distancia entre dos puntos es posible confeccionar un algoritmo similar para el perímetro de un polígono o la longitud de una polilínea.

La solución a este problema es bastante trivial partiendo de la propuesta anterior, pues solo sería iterar en la propuesta anterior cada dos vértices adyacentes de la figura geométrica analizada.

Algorithm 4. Algoritmo Genérico para el cálculo del perímetro

```
Procedure PolygonPerimeter ( _p: Polygon)
    if projection do
          if cartesian do
               for i ← 0 to n do

 perimeter ←  CartesianDistanceCalculation (_p [i],
_p[i+1])
               end for
          else
               for i ← 0 to n do
               perimeter ← SphericalDistanceCalc
utation(Transform(_p[i]),  Transform(_p[i+1]), true)
          end for
          end if
     else
          for i ← 0 to n do
          perimeter ←  SphericalDistanceCalcutat
ion(_p[i]), _p[i+1]), true)
          end for
     end if
     return perimeter
End procedure
```

3.3 Algoritmo para el cálculo de áreas

Para completar la propuesta de solución solo resta diseñar el algoritmo genérico para determinar el área de un polígono simple. Para ello primero se presenta el algoritmo para el caso de polígonos pla-

nos, se prosigue con el diseño de un algoritmo para el caso de polígono esféricos y, finalmente, se presenta el diseño general integrando ambos en el algoritmo genérico propuesto al inicio del trabajo.

3.3.1 Algoritmo para el área de polígonos simples en el plano

Uno de los algoritmos más difundidos para hallar el área de un polígono en el plano está basado en la técnica incremental. Consiste en, a partir de un vértice –vértice inicial–, trazar diagonales a los restantes vértices, obteniendo una triangulación de dicho polígono; posteriormente, el área del polígono en cuestión sería la sumatoria de las áreas de los triángulos resultantes.

Hallar el área de un triángulo en el plano es un problema de resolución trivial, pues estaría dada por el producto vectorial dividido entre dos de $p_1\, p_2\, x_1$ y $p_2\, p_3$, siendo p_1, p_2 y p_3 los vértices del triángulo, por lo que solo quedaría resolver el determinante de la matriz y dividir el resultado (Chen, 1996).

$$\begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix}$$

Esto daría el área resultante con signo positivo si los vértices se toman en sentido anti horario y expresada en correspondencia con las unidad de medida en que estén expresados p_1, p_2 y p_3.

Este algoritmo presenta una complejidad lineal $O(n)$ como se puede apreciar en el pseudocódigo del algoritmo cinco:

Algorithm 5. Algoritmo para el cálculo del área de un polígono en el plano

```
function SurfaceCartesianPolygon (p: Polygon)
   area ← 0
   for i ← 0 to p.vertex – 2
            area ← area + LeftTurn(p.vertex[0],
p.vertex[i], p.vertex[i+1])
   end for
   if area < 0 do
      return (area * -1) / 2
   end if
   return area / 2
end function

function LeftTurn( pto1, pto2, pto3: Point)
      return (pto1.x * pto2.y) – (pto1.y * pto2.x) +
(pto2.x * pto3.y) – (pto2.y * pto3.x) +     (pto1.x *
pto3.y) – (pto1.y *pto3.x)
end function
```

En el algoritmo presentado se calculan todas las áreas de los triángulos dobles y solo se divide al final para, de esta forma, disminuir los errores por truncamiento que pueden existir al realizar los cálculos en un ordenador debido al redondeo.

3.3.2 Algoritmo para el área de polígonos simples esféricos

Para el caso de polígonos simples sobre la superficie de la esfera se presentó en el capítulo anterior la propuesta de Bevis y Cambareri (1987), explicándose la restricción implícita en el mismo a la hora de digitalizar los polígonos y la cota superior de complejidad de no tenerse en cuenta dicha restricción.

Por tal motivo se diseña otra variante para determinar el área de estos polígonos eliminando esta restricción en la digitalización y de forma computacional menos costosa. El nuevo diseño mantiene la misma filosofía incremental del caso para polígonos en el plano con modificaciones en la forma de calcular el área de los triángulos.

El algoritmo tiene como única entrada el polígono al cual se le quiere determinar el área. Primeramente se selecciona un vértice como inicial y, a partir de éste, se trazan líneas geodésicas a los vértices restantes. De esta forma, se obtienen un conjunto de triángulos esféricos. El próximo paso sería sumar o restar el área de cada uno de estos triángulos en dependencia del giro realizado al recorrer sus vértices.

Para realizar el cálculo del área de estos triángulos, se calculan las longitudes de sus lados mediante Haversine. Luego, por la fórmula del coseno para geometrías esféricas, se hallan cada uno de los ángulos y finalmente con estos valores se puede obtener el área de dichos triángulos.

Algorithm 6. Algoritmo para el cálculo del área de un polígono sobre la esfera

```
function SurfaceSphericalPolygon (p: Polygon)
  area ← 0
  for i ← 0 to p.vertex – 2
      area← area + SurfaceSphericalTriangle(p.vertex[0],p.
vertex[i],p.vertex[i+1])
  end for
  if area < 0 do
    return area * -1
  end if
  return area
end function
function SurfaceSphericalTriangle (pto1, pto2, pto3:
Point)
  radio ← 6 378 100
  l1 ← SphericalDistanceCalcutation(pto1,pto2,false)
  l2 ← SphericalDistanceCalcutation(pto2,pto3,false)
  l3 ← SphericalDistanceCalcutation(pto1,pto3,false)
  a1 ← Fcoseno(l1,l2,l3)
  a2 ← Fcoseno(l2,l3,l1)
  a3 ← Fcoseno(l3,l1,l2)
  epsilon ← (a1 + a2 + a3) – Pi
  area ← epsilon * pow(radio, 2)
  if LeftTurn(pto1, pto2, pto3) < 0 do
     area ← area * -1
  end if
  return area
end function
function Fcoseno (l1, l2, l3: double)
    cose ← (cos(l1) – cos(l2) * cos(l3)) / (sin(l2) * sin(l3))
   return arccos(cose)
end function
```

De esta forma, se mantiene el mismo orden de complejidad -lineal- que para el caso de polígonos planos.

3.3.3 Pseudocódigo de algoritmo genérico para el área

Una vez que se cuenta con los algoritmos para el área de polígonos, tanto planos como esféricos, se puede pasar al diseño genérico vinculando ambos algoritmos. Inicialmente, el algoritmo en pseudocódigo quedaría como sigue en el algoritmo siete:

Algorithm 7. Variante 1 del algoritmo genérico para el cálculo del área de un polígono

```
function  CalculateSurface(p: Polygon)
        if projection do
            if cartesian do
                return SurfaceCartesianPolygon(p)
            end if
      return SurfaceSphericalPolygon(Transform(p))
      end if
      return SurfaceSphericalPolygon(p)
end function
```

4. Descripción y análisis de resultados

Una vez elaborada la propuesta en su totalidad se procede a analizar los resultados obtenidos con la misma y es este el objetivo del presente apartado. Para realizar las pruebas, la propuesta fue implementada en la Plataforma Soberana *GeneSIG*, que tiene como objetivo facilitar el desarrollo de sistemas de información geográfica en entornos web.

Estas pruebas se centran principalmente en el algoritmo genérico para el cálculo del área de un polígono debido a que es el aporte particular del trabajo

y en él se utiliza el algoritmo propuesto para determinar la distancia entre dos puntos. Algoritmo que es básico para los demás propuestos.

Primeramente se presenta el análisis de la complejidad del algoritmo para el área y luego se plasman una serie de resultados obtenidos tras varias corridas del algoritmo para diferentes polígonos.

4.1 Análisis de la complejidad del algoritmo para el área

En el algoritmo genérico propuesto para el cálculo del área se llevan a cabo dos comparaciones de complejidad $O(1)$ y en dependencia del resultado, se ejecutan funciones auxiliares que serían los que dicten la complejidad del algoritmo en general. Por lo que entonces se procede a analizar la complejidad de cada una de estas funciones.

4.1.1 Análisis de la complejidad de la función SurfaceCartesianPolygon

La función *SurfaceCartesianPolygon* implementa un ciclo que repite $n - 2$ veces la función *LeftTurn*, y en esta función a su vez solo se ejecutan operaciones elementales de suma, resta y multiplicación, por lo que tiene una complejidad $O(1)$. Teniéndose entonces el siguiente teorema:

Teorema 4.1.1.1: El área de un polígono en el plano puede ser computado con una complejidad $O(n)$. Dónde n es la cantidad de vértices del polígono.

4.1.2 Análisis de la complejidad de la función GetBox

La función *GetBox* también implementa un ciclo que ejecuta n veces un conjunto de comparaciones de complejidad $O(1)$ y asignaciones de complejidad $O(1)$. Resulta entonces el siguiente lema:

Lema 4.1.2.1: El menor rectángulo que contiene a un polígono puede ser computado con una complejidad Q (n). Dónde n es la cantidad de vértices del polígono.

4.1.3 Análisis de la complejidad de la función SphericalDistance-Calcutation

Como se puede apreciar, la función *SphericalDistanceCalcution* no implementa ciclo y sí solo un conjunto de asignaciones y cálculos matemáticos, por lo que su complejidad se puede decir a priori que sería $O(1)$. Sin embargo, se debe analizar el caso de las funciones raíz cuadrada y trigonométrica que se emplean en dicha función.

Brent (1976) se plantea que todas estas funciones pueden ser calculadas en tiempo constante en dependencia del nivel de precisión que se quiera obtener. Llegándose entonces al lema:

Lema 4.1.3.1: La distancia entre dos puntos sobre la esfera puede ser computada con una complejidad $O(h)$. Dónde h es una constante que varía en dependencia de la precisión que se quiere obtener.

4.1.4 Análisis de la complejidad de la función SurfaceSphericalPolygon

En la función *SurfaceSphericalPolygon* se sigue el mismo diseño que el planteado

en la función *SurfaceCartesianPolygon*, por tal motivo a priori se podría decir que esta función presenta una complejidad $O(n)$ por el Teorema 4.1.1.1.

Sin embargo, en esta función se hacen uso de funciones trigonométricas por lo que esta cota sufre modificaciones por el lema 4.1.3.1 quedando entonces:

Teorema 4.1.4.1: El área de un polígono esférico puede ser computado con una complejidad $O(nh)$. Dónde n el número de vértices de dicho polígono.

4.1.5 Análisis de la complejidad de la función SurfacePolygonHibrid-HeronSpherical

En el caso de la función *SurfacePolygon-HibridSpherical* sucede algo similar a la función *SurfaceSphericalPolygon* por lo que su complejidad es $O(nh)$.

4.1.6 Análisis de la complejidad del algoritmo genérico para el área

Una vez analizada la complejidad de las funciones auxiliares utilizadas en la confección del algoritmo genérico para el cálculo del área –Algorithm 7– se puede proceder a analizar la complejidad del mismo.

Si se está trabajando con coordenadas proyectadas y se quiera utilizar un método para el plano, se tendría una complejidad $O(n)$ –por el Teorema 4.1.1.1– y en caso de que se quiera usar un método para coordenadas elipsoidales se tendría una complejidad $O(nh)$.

Si se está trabajando con coordenadas elipsoidales se tendría una complejidad $O(nh)$.

Finalmente se tiene que:

Teorema 4.1.6.1: El área de un polígono se puede computar, teniendo en cuenta el sistema de coordenadas en que se encuentra, con una complejidad lineal.

4.2 Pruebas

Una vez analizada la complejidad del algoritmo genérico propuesto para el área, se procede a realizar un conjunto de pruebas para ver el comportamiento real del mismo en la Plataforma Soberana *GeneSIG*, o sea qué tan próximos están los resultados obtenidos con la propuesta de los resultados arrojados con otras herramientas.

Estas pruebas fueron realizadas con geometrías en diferentes escalas, pues como bien plantea Olaya (2010), la información geográfica puede analizarse a distintos niveles y en dependencia del nivel empleado, los resultados pudieran ser de naturaleza diferentes.

El primer grupo de pruebas se realizó con un conjunto de municipios de Cuba representados en una escala 1:500 000, comparándose los resultados arrojados por el algoritmo propuesto y los resultados obtenidos para los mismos polígonos, utilizando *MapInfo* y las función *ST_Area de Postgis* transformando las coordenadas a las proyecciones EPSG:2085 y EPSG:2086 respectivamente. Los resultados de estas mediciones se muestran en el cuadro 1 y figura 1 expresados en kilómetros cuadrados respectivamente.

Para el segundo grupo de prueba se utilizaron los mismos municipios, pero

Cuadro 1. Valores de las mediciones de áreas de polígonos en escala 1:500 000

Región	MapInfo	Postgis (2085)	Postgis (2086)	Algoritmo Propuesto
San Juan y Martínez	405,009	403,9143283	404,2283507	404,2779931
San Antonio de los Baños	130,43	130,1017484	130,2797945	130,2089398
Madruga	447,984	446,869987	447,510497	446,9620815
Los Arabos	748,965	747,0036781	747,8447752	749,5081552
Jovellanos	504,829	503,5480719	504,2183746	507,3098716
Colón	595,895	594,3571673	595,0894715	597,5460233
Cruces	196,655	196,1230512	196,2762825	198,6428112
Placetas	603,122	601,4902318	601,9473072	602,4329133
Manicaragua	1.062,70	1059,8104	1060,479165	1063,362983
Santo Domingo	875,7	873,4060305	874,3914672	875,3329034
Ranchuelo	549,672	548,1981547	548,6906695	551,6447314
Quemado de Güines	316,486	315,6772997	316,0841249	318,3506661
Fomento	465,461	464,1972602	464,4477088	466,0971347
La Sierpe	1.006,74	1004,059181	1004,222441	1009,323201
Sagua de Tánamo	706,746	709,5722342	705,3414425	704,6596852
Frank País	499,753	498,7042331	498,2829088	497,8296829

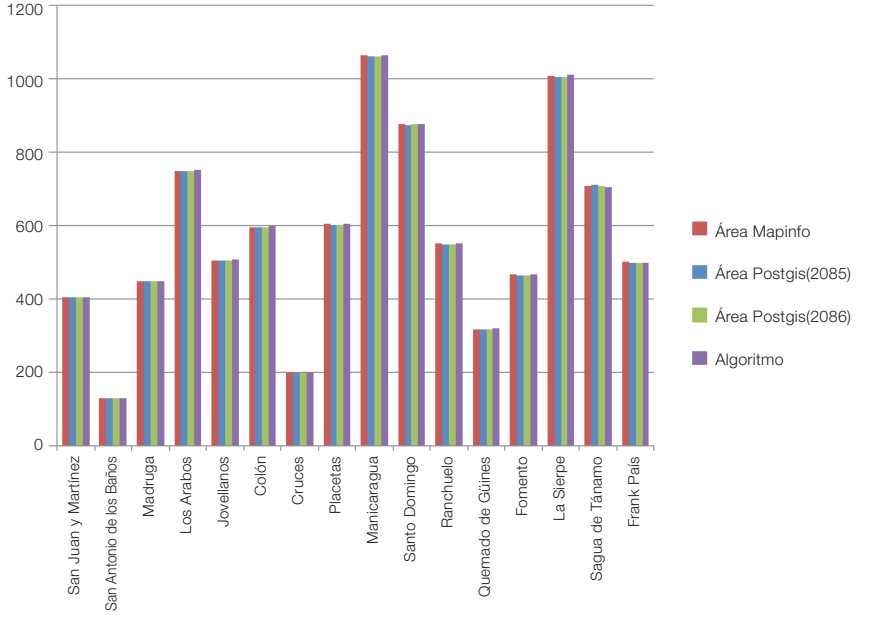

Figura 1. Comparación gráfica de los resultados obtenidos para la escala 1:500 000

en esta ocasión en una representación en escala 1:250 000. Los resultados de estas mediciones se muestran en el cuadro 2 y figura 2 expresados en kilómetros cuadrados respectivamente.

Y para un tercer y último grupo de prueba se utilizó otro conjunto diferente de polígonos –capitales provinciales– en una escala menor que 1:100 000. Los resultados en el cuadro 3 y figura 3.

Cuadro 2. Valores de las mediciones de áreas de polígonos en escala 1:250 000

Región	MapInfo	Postgis (2085)	Postgis (2086)	Algoritmo Propuesto
San Juan y Martínez	406,645	405,5461961	405,861339	400,8808058
San Antonio de los Baños	130,265	129,936429	130,1142107	135,5689788
Madruga	447,808	446,6944101	447,3346966	442,8680584
Los Arabos	747,423	747,0036781	747,8447752	753,5555376
Jovellanos	505,313	504,0306926	504,7013864	503,6723832
Colón	597,879	596,3366151	597,0712043	602,6732356
Cruces	196,411	195,8798178	196,0328505	196,7046385
Placetas	602,52	600,8902879	601,3469224	601,0251069
Manicaragua	1.063,43	1060,540882	1061,210065	1057,31177
Santo Domingo	875,458	873,4060305	874,3914672	877,7535266
Ranchuelo	549,981	548,3702718	548,505862	548,3702718
Quemado de Güines	315,899	315,0914413	315,4974845	310,8600063
Fomento	464,363	463,1020192	463,3519641	459,1251257
La Sierpe	1.005,29	1002,6077	1002,770431	1010,147319
Sagua de Tánamo	707,502	706,0961694	705,4135695	706,6139609
Frank País	501,984	500,930477	500,507247	501,2789209

Cuadro 3. Valores de las mediciones de áreas de polígonos en escala menor que 1:100 000

Región	MapInfo	Postgis (2085)	Postgis (2086)	Algoritmo Propuesto
Bayamo	10,0381	10,01960538	10,00846604	10,08275112
Las Tunas	16,5926	16,55381367	16,54491293	16,12287383
Camagüey	63,1462	62,983798	62,97622072	62,68977622
Ciego de Ávila	12,9559	12,92088205	12,92511921	12,16746572
Pinar del Río	13,3728	13,33696663	13,34908336	12,06496844
Cienfuegos	18,8189	18,76778513	18,77975989	18,47725493
Sancti Spíritus	11,6092	11,57767856	11,58256003	11,10096355
Santiago de Cuba	40,1734	40,11536241	40,05706149	40,62707222

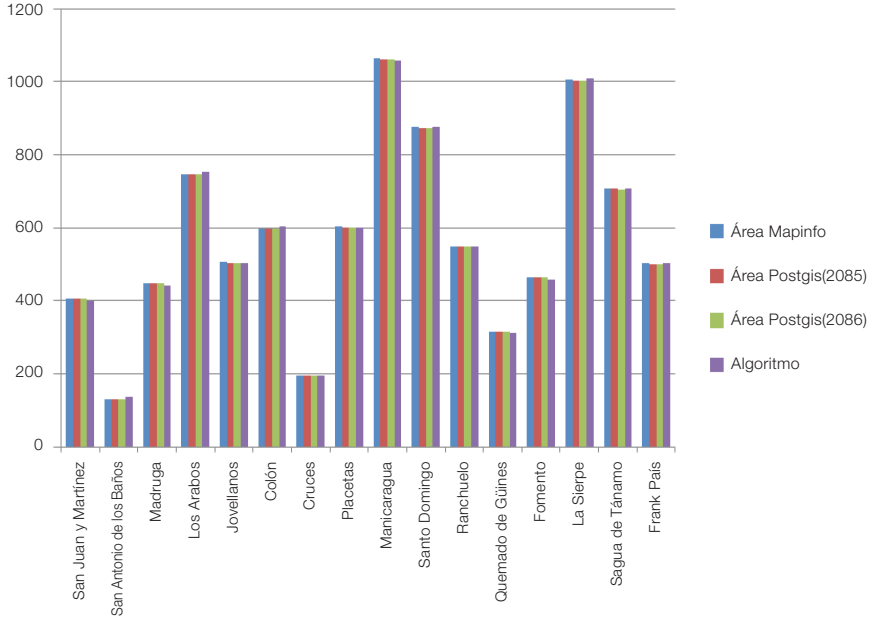

Figura 2. Comparación gráfica de los resultados obtenidos para la escala 1:250 000

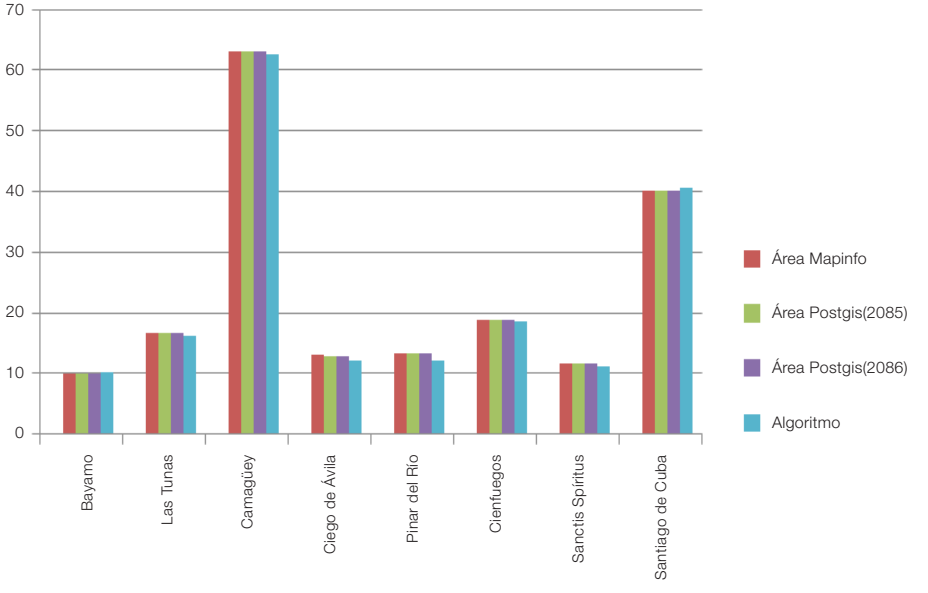

Figura 3. Comparación gráfica de los resultados obtenidos para la escala menor que 1:100 000

5. Conclusiones

- El análisis geométrico en los SIG no es un problema simple si se quieren obtener resultados válidos. El usuario debe ser consciente de la naturaleza de los datos con los que está trabajando, qué tipo de proyección y algoritmos utilizar en dependencia de los resultados que se quieran obtener.
- La aproximación del esferoide/elipsoide a una esfera permite obtener algoritmos sencillos y válidos, con un mínimo de sacrificio en la exactitud de los resultados, para cálculos geométricos sobre su superficie.
- Se elimina la utilización de algoritmos geodésicos cuando se trabaja en coordenadas geográficas, obteniéndose algoritmos eficientes -complejidad lineal- sin la necesidad de imposición de restricciones en la digitalización de las geometrías.
- La combinación de estos algoritmos con los clásicos euclidianos, teniendo en cuenta las características de los datos, permite darles a usuarios avanzados la opción de elegir con qué tipo de método desea trabajar. Y a usuarios inexpertos en temas de cartografía y geodesia les garantiza obtener resultados válidos independientes del sistema de coordenadas con el que trabaje.

6. Referencias citadas

ALONSO SARRÍA, F**.** 2006. *Sistemas de Información Geográfica***.** Universidad de Murcia. Disponible en: http://www.um.es/geograf/sigmur/sigpdf/temario.pdf.

BEVIS, M. y G. CAMBARERI. 1987. *Computing the Area of a Spherical Polygon of Arbitrary Shape*. **Mathematical Geology**. 19(4): 335-346.

BRENT, R. 1976. *Fast Multiple-Precision Evaluation of Elementary Functions*. **Journal of the Association for Computing Machinery**. 23(2): 242-251.

CHEN, J. 1996. *Computational Geometry-Methods and Applications*. A&M University, Computer Science Departament. Texas-USA.

IGLESIAS, M. y M. ASUNCIÓN. 1996. *Trigonometría esférica. Breve Introducción a la navegación*. 8475857876. Universidad del País Vasco. Servicio Editorial. España.

MENCIA, J**.** 2006. *Nociones de trigonometría esférica*. Universidad del País Vasco-Euskal Herriko.

OLAYA, V. 2010. *Sistemas de Información Geográfica*. s.l. OSGEO. Disponible en: http://www.bubok.es/libros/191920/Sistemas-de-Informacion-Geografica.

PESQUER, L.; PONS, X. y J. MASÓ. 2005. Necesidad de cálculos geodésicos para las herramientas SIG de análisis de distancias. *6th Geomatic Week proceedings*. Barcelona-España (8-10 de febrero).

REFRACTIONS RESEARCH, INC. 2008. *Postgis Manual 1.5.0*. Refractions Research Inc. Victoria. British Columbia-Canada.

WEISSTEIN, E. 2011. *Bijective. Wolfram-MathWorld*. Disponible en: http://mathworld.wolfram.com/Bijective.html

ZAKATOV, P. S. 1981. **Curso de Geodesia Superior.** Editorial Mir. Rusia.