# Facultad de Informática
# Universidad Nacional de La Plata

## Trabajo de Grado: Un Modelo OO para Manipulación de Datos Espaciales


Diego Cano
diego.cano@lifia.info.unlp.edu.ar

Gisela Trilla
gisela.trilla@lifia.info.unlp.edu.ar

Director: Dra. Silvia Gordillo



*La Plata - Buenos Aires*
*Argentina*
*Abril de 2007*

## AGRADECIMIENTOS

Debo en primer lugar a mi familia el hecho de haber podido realizar este trabajo, el apoyo de mis padres y mi hermana ha sido constante desde el principio; agradezco también a mi abuela Dora por el interés, el apoyo y el millar de velas encendidas en épocas de exámenes.

A Silvia Gordillo, por su predisposición para este trabajo y otros del mismo campo, y por su contención en los momentos en que los mismos se volvían dificultosos o ejercían presión.

A Javier Bazzocco, por todas las correcciones, observaciones y asistencias tan necesarias para el desarrollo de este trabajo.

A los compañeros de facultad y a los de LIFIA, por el excelente ambiente en el que pude estudiar y trabajar.

Agradezco también a quienes desde disciplinas supuestamente distantes de la informática han contribuido de manera indirecta pero concreta en este trabajo, en particular a Enrique Pagani.

Diego Cano

Agradezco a mi mamá y mi papá porque gracias a ellos pude venir a estudiar, y este trabajo está dedicado al esfuerzo que hicieron ellos en poder darme esa posibilidad.

Agradezco a mis hermanas, Yamila y Mara, por ayudarme muchas veces con Sofi, y ahora con Camila, para que pueda ponerme a estudiar y terminar lo que me quedaba de esta carrera.

La ayuda que me dieron Julia y Fido, al igual que mis hermanas con Sofi, y el hacerme sentir que siempre puedo contar con ellos para lo que necesite. A Julia, en particular, porque siempre me insistió para que me pusiera las pilas y terminara la carrera.

A Javier por acompañarme desde el segundo año de la carrera, formando parte de mi vida, junto con los dos solcitos que tenemos. También le agradezco sus correcciones que ayudaron a refinar este trabajo y todo lo que me aguanto desde que nos conocemos.

Al LIFIA, que como ya lo expresó Diego, me permitió trabajar y estudiar, y porque siempre me han escuchado y ayudado, en particular Silvia Gordillo y Andrés Rodríguez.

A Diego, compañero de casi toda la carrera, compañero en el LIFIA y compañero de tesis, por no tener problemas en reunirse y trabajar con los llantos de Camila en su oreja o las interrupciones de Sofi para que juegue con ella un ratito.

Gisela Trilla

## ÍNDICE

# 1. CAPÍTULO I: INTRODUCCIÓN

*Este capítulo describe brevemente los aspectos principales del dominio de aplicaciones SIG, que serán expuestos en mayor detalle en capítulos posteriores. También presenta los objetivos de este trabajo de grado, demarca su alcance y describe la estructura general del mismo, de manera de brindar una visión global de su desarrollo.*

## 1.1. Objetivos del trabajo

Un Sistema de Información Geográfica (SIG) es un conjunto de procedimientos manuales o computarizados utilizados para almacenar y manipular datos referenciados geográficamente. Un SIG no es una aplicación concreta, sino que constituye un núcleo, un conjunto de métodos para la gestión de datos georeferenciados que cualquier aplicación concreta de cualquier dominio puede utilizar. Las aplicaciones concretas en que se utiliza un SIG actualmente son muy numerosas y diversas, como ejemplo pueden citarse: Agricultura y Planeamiento de Uso de la Tierra, Control Forestal, Arqueología, Geología, Planeamiento Urbano, Catastro, Control Epidemiológico, Control Policial, Asistencia en Robos de Vehículos, Marketing.

El presente trabajo de grado consiste en el desarrollo de un modelo genérico que brinde funcionalidad referente a la gestión de información geográfica. Este modelo constituye una capa de SIG y se busca que sobre la misma pueda montarse cualquier aplicación cliente concreta que necesite gestionar información geográfica. Los requisitos de integración entre esta capa genérica y las aplicaciones clientes concretas deben ser mínimos de modo reducir el esfuerzo de integración entre ambas.

Para que un SIG pueda responder a las necesidades de las aplicaciones de forma correcta, debe contar con formas muy particulares de gestionar la información en aspectos tales como su captura (GPS[1], Remote Sensing, Fotografía Aérea o Satelital), procesamiento (Sistemas de Referencia, Topología, Indización) y presentación (Mapas Digitales en diversos formatos, además de las formas tradicionales).

Concretamente, los aspectos que se contemplarán en este trabajo son los siguientes: Sistema de Referencias, Modelo Topológico, Estructuras de Indización Geográfica y Estrategias de Geometría Computacional.

En este trabajo se diseña una capa de SIG mediante un modelo Orientado a Objetos que contempla los aspectos mencionados anteriormente. Este modelo tiene gran flexibilidad en la elección, configuración y extensión de varios de esos aspectos por parte del usuario final. Asimismo no impone implementaciones particulares para estos aspectos, sino que provee un conjunto de ellas de las cuales el usuario final podrá elegir la que más se adapte a su caso particular; incluso podrá agregas otras nuevas que considere más adecuadas que las ofrecidas, sin reparar en problemas de coexistencia e integración con el resto del sistema.

El Sistema de Referencias permite definir la ubicación espacial de los objetos y la realización de operaciones tales como el cálculo de la distancia entre ellos, o el del área ocupada por el perímetro definido por un conjunto de tales objetos. Existen varios Sistemas de Referencias y cada uno de ellos tiene sus ventajas y desventajas dependiendo del dominio de aplicación, es por esto que el modelo contempla algunos de ellos y queda abierto para una fácil extensión con otros.

El Modelo Topológico permite definir la forma de las entidades geográficas y facilitar las operaciones que dependen de ella y de su ubicación relativa, es decir, sin depender de la ubicación exacta en el espacio de tales entidades.

Las Estructuras de Indización Geográfica permiten un rápido acceso a los objetos geográficos y la formulación de consultas con parámetros espaciales, como por ejemplo la obtención de todos los objetos que se encuentran en un área dada como dato. Éstas se definen mediante una interfaz genérica de manera de dejar el modelo abierto a nuevas estructuras de indización que el usuario final desee agregar y utilizar.

Las Estrategias de Geometría Computacional son algoritmos que se utilizan para la realización de operaciones y testeos geométricos, como por ejemplo el cálculo de la unión e intersección de polígonos o testeos de inclusión espacial. Existen varias estrategias alternativas para cada operación, cada una con ventajas dependientes de aspectos particulares, por ejemplo del tamaño de los polígonos. Es por esto que constituye una ventaja el hecho de ofrecer varias de ellas de modo que el usuario seleccione la más apropiada para su dominio de aplicación.

## 1.2. Contenido de capítulos y apéndices

[1] Global Positioning System.

El trabajo está dividido en dos partes. La primera parte contiene tres capítulos, los cuales presentan información de dominio necesaria para poder comprender los principales aspectos de un SIG. La segunda parte consta de cinco capítulos, los cuales muestran el diseño de los aspectos anteriores (Sistema de Referencias, Modelo Topológico, Estructuras de Indización Geográfica, Estrategias de Geometría Computacional) y presentan la integración con una aplicación cliente tomada como ejemplo.

En el **Capítulo II** se describe el dominio y las particularidades que tienen los aspectos topológicos y el sistema de referencias de los sistemas de información geográfica.

Se exponen en el **Capítulo III** los aspectos de indización espacial que deben contemplarse en las aplicaciones del dominio.

En el **Capítulo IV** se analizan las diversas estrategias de Geometría Computacional utilizadas como base en las operaciones que deben llevarse a cabo entre objetos geométricos.

El **Capítulo V** marca el comienzo de la segunda parte de este trabajo, y en él se presenta el diseño del modelo para los aspectos de la topología y del sistema de referencias de la capa de SIG.

En el **Capítulo VI** se presenta el diseño de la parte concerniente a la indización de objetos geográficos. Cuenta además con una aplicación que muestra el funcionamiento de una implementación para una estructura de indización particular: el R-Tree.

En el **Capítulo VII** se describe el diseño de las operaciones geométricas de la capa de SIG. También cuenta con una aplicación que permite observar el funcionamiento real de una implementación concreta para estas operaciones.

En el **Capítulo VIII** se presenta una implementación que muestra la integración de los aspectos presentados en los capítulos V, VI y VII.

En el **Capítulo IX** se detalla como interactúa la capa de SIG con una aplicación cliente en particular. Se presenta un ejemplo de aplicación en el dominio de la agricultura, en particular la gestión de cultivos en cuanto a su relación con lotes y equipos de riego.

Existe información que complementa a los capítulos anteriores, la misma se encuentra dividida en tres apéndices:

El **Apéndice A** complementa al capítulo III. Contiene algoritmos utilizados para la implementación de una estructura de Indización muy utilizada como lo es el R-Tree.

El **Apéndice B** contiene una breve descripción de cada Patrón de Diseño utilizado para desarrollar nuestro modelo.

El **Apéndice C** complementa al capítulo VIII mostrando una descripción de un patrón de diseño utilizado para realizar la aplicación que se encuentra en ese capítulo. Dicho patrón es el Appearance y permite desacoplar la visualización de los elementos de un sistema de información geográfica. Este patrón ha sido creado por los autores y presentado por los mismos como parte de trabajos expuestos en congresos nacionales e internacionales.

# PARTE I: DESCRIPCIÓN DEL DOMINIO

## 2. CAPÍTULO II: DATOS Y MODELOS GEOGRÁFICOS

*En este capítulo se introducen los conceptos básicos del dominio de aplicaciones SIG. Se describen las características esenciales de los datos geo-referenciados y los modelos más utilizados para la representación de entidades geográficas. Con respecto a estos últimos, se presentan algunos modelos de datos geográficos de tipo Raster y de tipo Vector; se muestran sus componentes esenciales y se los compara en cuanto a ventajas y desventajas relacionadas a aspectos tales como riqueza de representación, espacio de almacenamiento y performance en las operaciones más frecuentemente utilizadas.*

### 2.1. Datos geográficos

Un sistema de base de datos de un SIG debe proveer la manera de organizar los datos espaciales y no espaciales de una manera eficaz. Los datos de este tipo de sistemas poseen cuatro características esenciales [Aronoff] que deben ser gestionadas correctamente para alcanzar tal eficacia; las mismas se describen a continuación.

#### 2.1.1. Posición

Para definir la posición geográfica de un objeto no es suficiente un conjunto de coordenadas, sino que además se necesita una referencia a partir de la cual interpretarla. Los distintos modelos que brindan estos contextos referenciales se conocen como Sistemas de Referencias.

Por lo tanto, el sistema de referencias es el encargado de brindar el contexto geográfico necesario para interpretar la coordenada del objeto. Para esto, dicho sistema se compone de otros subsistemas o contextos referenciales de segundo orden, como por ejemplo sistemas de coordenadas y datums, temas que se describirán en mayor nivel de detalle en un capítulo posterior.

A grandes rasgos, el sistema de coordenadas constituye un marco lógico y matemático para describir la posición [Voser98]. Ejemplos de sistemas de coordenadas son el geodésico (latitud, longitud y altura con respecto al elipsoide que representa la tierra) y el ECEFXYZ (coordenadas x, y, z con respecto a los ejes de un datum). Los distintos Sistemas de Coordenadas proveen diferentes marcos referenciales, con lo cual permiten expresar de diversas maneras la posición geográfica de un mismo objeto.

La Tierra es una superficie compleja en cuanto a su regularidad geométrica. Su forma constituye un sólido denominado geoide, sin embargo, dado que éste no puede ser representado mediante fórmulas matemáticas, se lo aproxima mediante un elipsoide [Notes on GPS]. Además de este elipsoide es necesario definir algunas relaciones espaciales entre éste y el sistema de coordenadas. Estas relaciones son la orientación y el desplazamiento respecto al eje de coordenadas XYZ. La primera indica el grado de rotación del elipsoide sobre sí mismo en cada dimensión con respecto a su orientación original; el segundo indica la ubicación del mismo con respecto al origen del Sistema de Coordenadas. El elispoide, junto con la orientación y el desplazamiento, forman lo que llamamos datums [OpenGIS Consortium]. De esta manera, los datums permiten representar la forma de la Tierra con más precisión en algunas partes que otras. Dada la irregularidad innata del geoide, se aproximan distintas partes del mismo mediante diferentes datums de forma de lograr mayor precisión en cada una de ellas. Esto genera la definición de varios datums, uno para cada país, por lo cual debe prestarse atención al hecho de estar usando el datum correcto en cada caso particular.

#### 2.1.2. Atributos no espaciales

Los atributos no espaciales permiten describir al dato geográfico. Por ejemplo, para una ciudad atributos no espaciales podrían ser el nombre, cantidad de habitantes, promedio de edad de sus habitantes. Podríamos decir que en general esta información es la que gestionan los sistemas administrativos tradicionales.

#### 2.1.3. Relaciones espaciales

Se utilizan para mostrar cómo se relacionan los datos geográficos. Por lo general son muchas las relaciones que tiene un dato geográfico con el resto de ellos, por lo cual no es posible gestionarlas a todas de forma simultánea. La selección de aquellas que se decida gestionar dependen del aspecto que se desea analizar en el caso particular de cada aplicación SIG. Por ejemplo, son relaciones espaciales entre datos geográficos: las rutas que se utilizan para llegar de una ciudad a otra, la red de servicio de alumbrado público, la vecindad de países o ciudades en cuanto a su división política del territorio.

### 2.1.4. Tiempo

El dato geográfico puede hacer referencia a un momento o período de tiempo. La disposición geográfica de un conjunto de objetos en una región, como así también sus relaciones, podrían variar en un período determinado. Un lugar despoblado podría poblarse al año siguiente, lo cual demandaría el alta de nuevos objetos en el sistema; para el caso de las relaciones, un convenio entre dos empresas de transporte público podría demandar el alta de una nueva posibilidad de transbordo en algún punto común a dos de sus recorridos pre-existentes.

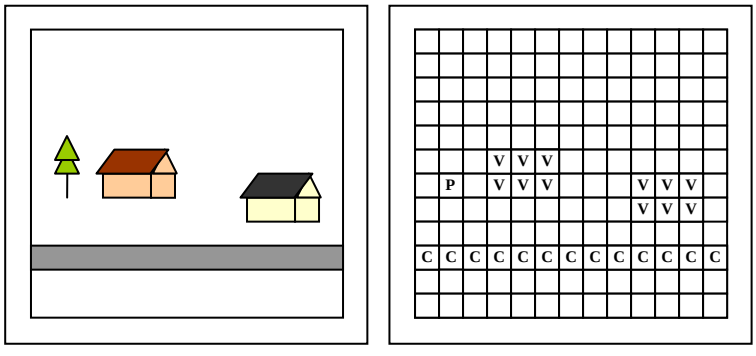
De acuerdo al tipo de análisis que se desee realizar, la información histórica puede ser uno de los componentes más valiosos de un SIG.

## 2.2. Modelo de Datos geográficos

Existen distintos modelos para representar datos geográficos, cada uno de ellos con diferentes características referentes a almacenamiento de relaciones, espacio de almacenamiento, facilidad de actualización y performance de operaciones. En esta sección se describirán dos clasificaciones típicas de estos modelos: los de tipo Raster y los de Vector.

### 2.2.1. Raster

En este tipo de modelos el espacio es dividido en celdas regulares, generalmente cuadradas. La posición de los objetos geográficos queda definida por la posición de la columna y fila de las celdas que ocupan. El área que cada celda representa define la resolución del modelo. El valor guardado en cada celda indica el tipo de objeto que se encuentra en ese lugar.




a b

**Figura 2.1**.- (a) Conjunto de objetos distribuidos espacialmente, (b) Representación raster del conjunto de objetos de (a).

La unidad espacial es la celda, cada una de ellas corresponde a un área en una ubicación específica. Los objetos geográficos quedan representados mediante un conjunto de celdas, sin embargo cada uno de estos conjuntos no conforman una entidad en sí mismos.

Puede verse que se trata de una estructura simple, por lo que pueden implementarse operaciones de superposición de manera fácil y eficiente. La alta variabilidad de espacio es representada eficientemente, es por esto que es muy utilizado para la gestión de imágenes digitales. Tiene como desventaja que la estructura de datos no es compacta, y por esto deben utilizarse técnicas de compresión. Además, las relaciones entre los objetos (tales como conectividad o vecindad) son difíciles de gestionar eficientemente ya que no pueden representarse de forma explícita, sino que se encuentran implícitas en el modelo y deben obtenerse mediante complejas operaciones sobre la matriz.

En cuanto a su forma, la implementación de las celdas no se ve limitada a la figura cuadrada sino que también pueden implementarse modelos Raster con triángulos, hexágonos u otras figuras. Un requisito indispensable es que es el agrupamiento de varias de tales figuras produzca una total cobertura del espacio. Por ejemplo, el círculo no sería una forma de celda válida para un modelo Raster.

Para el almacenamiento físico, se genera un archivo por cada aspecto que se quiera representar (p. ej: jurisdicciones policiales, áreas de mayor delito, ubicación de comisarías u hospitales). Las operaciones sobre varios archivos raster involucran recuperar y procesar los datos desde posiciones iguales de celdas en los diferentes archivos.

Debido a que muchas celdas pueden llegar a tener el mismo valor, se utilizan técnicas de compresión de datos. Algunas de las mismas se explican a continuación:

- Run length encoding: Mediante esta técnica se agrupan las celdas adyacentes de una fila que tienen el mismo valor, esto se denomina run. Entonces un mismo valor es guardado una vez, indicando la fila y la cantidad de posiciones con el mismo valor.

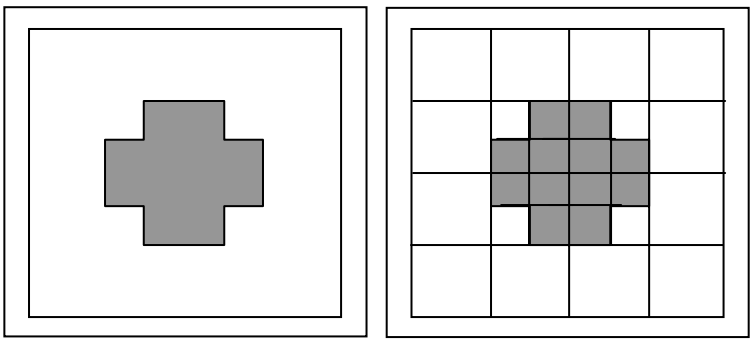
  Existen varias estrategias de codificación run-length. Una de ellas es la estándar, mencionada anteriormente, que guarda el valor, la cantidad de celdas que ocupa y la fila en la que se encuentra. Otra es la codificación de "punto valor": se guarda el valor y el punto (fila, columna) donde termina la secuencia de valores iguales. Esta última implica que la grilla debe seguir una numeración que comienza en la parte superior izquierda y continua recorriendo cada fila.

  El grado de compresión que puede lograrse depende de la complejidad del mapa. Mediante celdas de tamaño más grandes obtenemos archivos de datos más pequeños que utilizan menos espacio de almacenamiento y son más rápidos de procesar, sin embargo los bordes pueden ser posicionados sólo por el tamaño de una celda. Mediante celdas más chicas se logra más posicionamiento sobre los bordes pero incrementa el tamaño del archivo y los tiempos de procesamiento al utilizarlo.

- Quadtree: Con esta técnica se soluciona el tema de la resolución y de la redundancia. También son utilizados como índices, esto se describe en mayor detalle en el capítulo III, correspondiente a Indización.

  La compresión es realizada definiendo celdas de tamaño variable. A pesar de que se puede realizar una dividisión en celdas de un solo tamaño, se utilizan subdivisiones más finas en aquellas áreas que requieren más detalle, así un alto nivel de resolución es provisto solamente cuando es necesario.

  La construcción de un quadtree es un proceso de subdivisión regular en cuatro cuadrantes de igual tamaño. Si cada cuadrante no contiene el mismo valor, entonces se subdivide en otros cuatro, así continúa hasta que todas las celdas contengan el mismo valor.



a b

**Figura 2.2**.- (a) Área a representar, (b) Representación en forma de quadtree del área (a).

Los quadtree pueden hacer más eficientes algunas operaciones entre polígonos (como búsquedas de un punto en un polígono) que con la estructura de un raster común. Son muy útiles cuando los datos son homogéneos y no se requieren actualizaciones frecuentes. Sus ventajas disminuyen si pasa lo contrario, es decir cuando el mapa a representar es muy complejo y se requieren muchas actualizaciones.

### 2.2.2. Vector

En este tipo de modelos los objetos geográficos son representados mediante puntos, líneas y polígonos, que definen sus bordes. La figura siguiente ilustra una situación real y una representación posible mediante el modelo de vector.

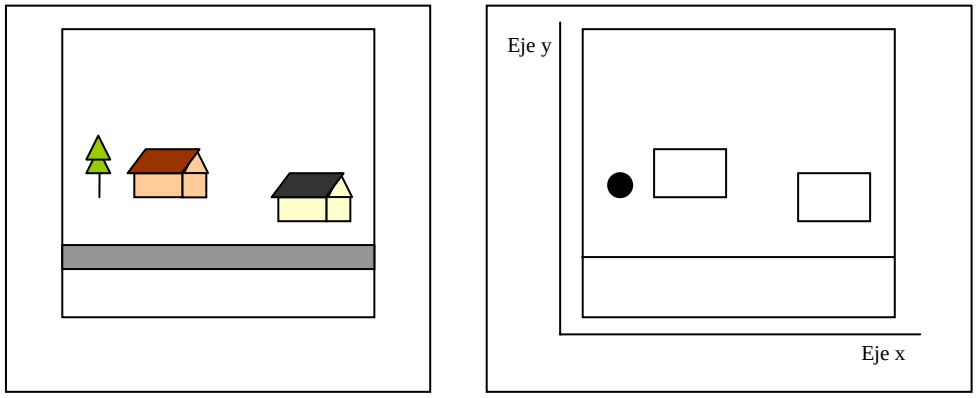



**Figura 2.3**.- (a) Conjunto de objetos distribuidos espacialmente, (b) Representación de vector del conjunto de objetos de (a).

La posición de los objetos geográficos queda definida por su ubicación en el mapa, de acuerdo a un sistema de coordenadas, en general XY. En un SIG, las posiciones son usualmente almacenadas utilizando sistemas de coordenadas geográficos, como el UTM, State Plane o latitud/longitud.

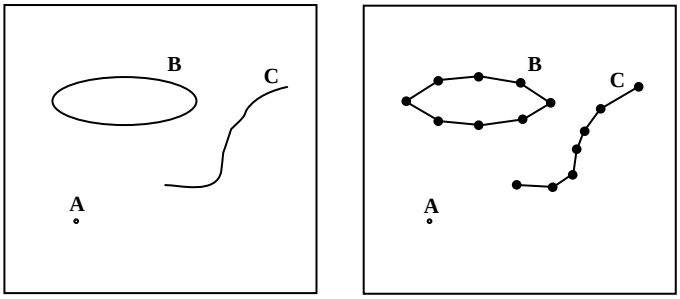
Un punto es guardado como un único par de coordenadas (x,y), una línea (no necesariamente recta) como una serie de pares coordenados (x,y), y un polígono como serie cerrada de pares coordenados (x,y) que definen los bordes de un área.

Cada posición en el mapa tiene un único valor de coordenadas. A diferencia del modelo Raster, cada elemento de este modelo (línea, punto o polígono) representa una unidad del mundo real.

Este modelo utiliza una estructura de datos más compacta que el modelo de raster y una implementación más eficiente de las operaciones que requieren información de relación espacial entre objetos. Es más adecuado para soportar gráficos. Esto trae como desventaja que la estructura de datos es más compleja, las operaciones de superposición son más difíciles de implementar y la representación es ineficiente cuando el espacio presenta mucha variabilidad.

Hay distintos tipos de sistemas que utilizan el modelo de vector para almacenar los objetos geográficos. Algunos se detallan a continuación:

- Spaghetti: Traduce el mapa línea por línea en un conjunto de coordenadas (x,y), el mapa es expresado en coordenadas cartesianas. Un punto es un par de coordenadas, una línea es un string de coordenadas y un área es representada por un polígono, el cual se guarda como una línea cerrada de pares coordenados. Los bordes de polígonos adyacentes son guardados una vez por cada polígono.




a b

**Figura 2.4**.- (a) Conjunto de objetos distribuidos espacialmente, (b) Técnica Spaghetti para representas el conjunto de objetos de (a).

Un archivo de datos espaciales construido por este modelo es simplemente un conjunto de strings de coordenadas, que no tienen una estructura inherente. A continuación se detalla el archivo de datos correspondiente a la Figura 2.4.

| Tipo de figura | Identificación | Ubicación |
|---|---|---|
| Punto | A | (x,y) |
| Línea | C | $(x_1,y_1)$, $(x_2,y_2)$,…, $(x_7,y_7)$ |
| Polígono | B | $(x_1,y_1)$, $(x_2,y_2)$,…, $(x_8,y_8)$, $(x_1,y_1)$ |

El modelo permite registrar todos los objetos geográficos pero no guarda las relaciones entre ellos, por ejemplo el hecho de que un polígono es adyacente a otro. Esta información se genera con operaciones de búsqueda sobre el archivo de datos y determinando si hay o no adyacencia.

Este modelo es ineficiente para la mayoría de los tipos de análisis espaciales que se requieren en un SIG. Sin embargo, es un modelo eficiente para reproducir mapas digitalmente, ya que no guarda la información de las relaciones espaciales.

- Topológico: Este modelo es el más utilizado para almacenar relaciones espaciales en un SIG. La topología es el método matemático utilizado para definir relaciones espaciales.

  La entidad lógica básica es el arco, conformado por una serie de puntos que comienzan y terminan en un nodo. Un nodo es un punto de intersección entre dos o más arcos. Los nodos no conectados a arcos son representados con puntos. Un polígono es una serie cerrada de arcos que no se cruzan y que conforman los bordes del área.

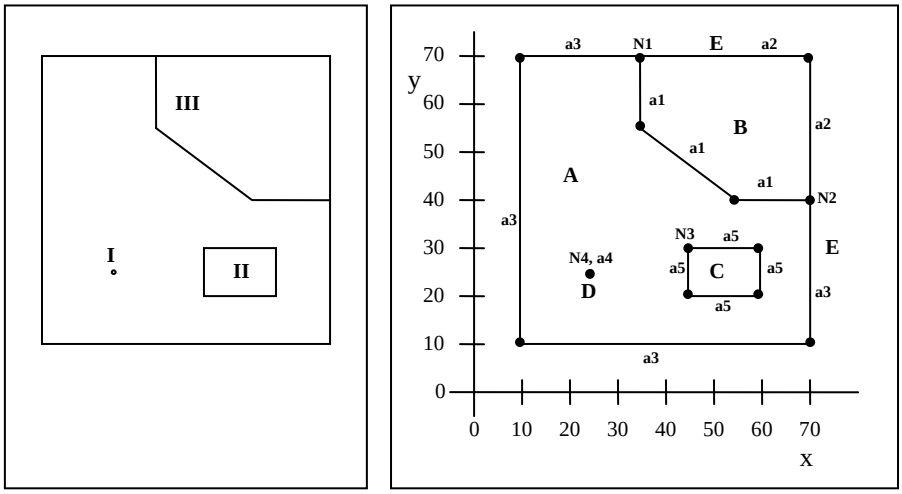



**Figura 2.5**.- (a) Conjunto de objetos distribuidos espacialmente, (b) Representación del conjunto de objetos de (a), mediante topología.

La topología es guardada en tres tipos de tablas, una para cada tipo de elemento espacial, y otra tabla para las coordenadas. A continuación se detallan los archivos correspondientes a la Figura 2.5:

| Topología Polígono | |
|---|---|
| Polígono | Arcos |
| A | a1, a3 |
| B | a2, a1 |
| C | a5 |
| D | a4 |
| E | Área exterior |

| Topología Nodo | |
|---|---|
| Nodo | Arcos |
| N1 | a1, a2, a3 |
| N2 | a2, a1, a3 |
| N3 | a5 |
| N4 | a4 |

| Topología Arco | | | | |
|---|---|---|---|---|
| Arco | Nodo Inicio | Nodo Fin | Polígono derecha | Polígono izquierda |
| a1 | N1 | N2 | B | A |
| a2 | N1 | N2 | E | B |
| a3 | N2 | N1 | E | A |
| a4 | N4 | N4 | A | A |
| a5 | N3 | N3 | A | C |

| Datos de coordenadas | | | |
|---|---|---|---|
| Arco | Inicio x, y | Intermedios x,y | Fin x, y |
| a1 | (35,70) | (35,55), (55,40) | (70,40) |
| a2 | (35,70) | (70,70) | (70,40) |
| a3 | (70,40) | (70,10), (10,10),(10,70) | (35,70) |
| a4 | (25,25) | | (25,25) |
| a5 | (45,30) | (60,30),(60,20), (45,20) | (45,30) |

La tabla de Polígonos registra los arcos que conforman a cada uno de ellos. También se debe definir un polígono para definir los bordes del mapa. Este también es guardado junto a los demás polígonos. La tabla de Nodos registra los nodos junto con los arcos que pertenecen a él. La tabla de Arcos registra los nodos de inicio y fin del arco, y la relación con los polígonos que lo rodean, es decir cual polígono es adyacente a la derecha del arco, y cuál a la izquierda. Esto permite que las relaciones espaciales puedan ser calculadas rápidamente, a diferencia del modelo Spaghetti. La tabla de Coordenadas guarda las coordenadas que son necesarias para representar las posiciones del mundo real. Cada arco puede ser representado por más de un segmento de línea definido por una serie de coordenadas.

Los atributos de los datos son guardados en una base de datos relacional, y uno de sus datos es el código de la entidad espacial que representan.

Con este modelo se pueden realizar análisis de adyacencia y de conectividad de una forma muy fácil, sin necesidad de utilizar los datos de las coordenadas, evitando así caer en complejos cálculos matemáticos. La desventaja de este modelo es que es muy costoso crear la estructura y mantenerla actualizada.

- TIN: Es un modelo topológico basado en el de vector. Representa la superficie de la tierra como un conjunto de triángulos interconectados. Por cada uno de los tres vértices se registran las coordenadas xy de su posición y un valor z correspondiente a la altura.

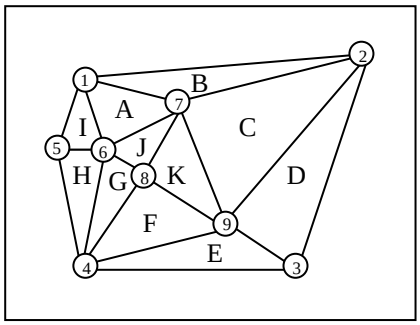



**Figura 2.6**.- Ejemplo de una representación de TIN.

Cada triángulo es identificado con una letra, y los vértices son numerados. Se utilizan cuatro tablas para su almacenamiento. La tabla de nodos, que lista cada triángulo y los nodos que lo definen. La tabla de límites, que almacena los dos o tres triángulos adyacentes a cada triángulo. La tabla de coordenadas XY y la tabla de coordenadas z que guarda el valor de la altura para cada coordenada XY. A continuación se detallan las tablas correspondientes a la Figura 2.6:

| Coordenadas XY | |
|---|---|
| Nodo | Coordenadas |
| 1 | $x_1$,$y_1$ |
| 2 | $x_2$,$y_2$ |
| . | |
| . | |
| 9 | $x_9$,$y_9$ |

| Coordenadas Z | |
|---|---|
| Nodo | Coordenadas |
| 1 | $Z_1$ |
| 2 | $z_2$ |
| . | |
| . | |
| 9 | $z_9$ |

| Límites | |
|---|---|
| Triángulo | Triángulos adyacentes |
| A | B, J, I |
| B | A, C |
| C | B, D, K |
| D | E, C |
| E | D, F |
| F | E, G, K |
| G | F, H, J |
| H | G, I |
| I | A, H |
| J | A, G, K |
| K | C, F, J |

| Nodos | |
|---|---|
| Triángulo | Nodo |
| A | 1, 7, 6 |
| B | 1, 2, 7 |
| C | 7, 2, 9 |
| D | 2, 3, 9 |
| E | 3, 4, 9 |
| F | 9, 4, 8 |
| G | 6, 4, 8 |
| H | 6, 4, 5 |
| I | 1, 6, 5 |
| J | 7, 8, 6 |
| K | 7, 9, 8 |

Usando este modelo, la pendiente y aspectos de un terreno son calculados por cada triángulo y guardados como un atributo de él, de la misma forma que los atributos son guardados para los polígonos. Estos valores pueden ser consultados utilizando el mismo tipo de operaciones de base de datos.

Una de las ventajas del TIN es que la información es codificada para áreas complejas sin necesidad de requerir grandes colecciones de datos, de la misma forma que las áreas simples. Debido a que el tamaño de los triángulos es variable, para obtener una representación detallada se debe aumentar la densidad de puntos y con eso los triángulos quedaran más pequeños.

Un TIN puede representarse utilizando celdas de una grilla, como un raster. La desventaja del TIN contra un modelo raster, es que el TIN necesita más procesamiento para generar su archivo que el de un raster común. Sin embargo, una vez que el TIN es generado se logra una representación más compacta y que puede procesarse más eficientemente.

# 3. CAPÍTULO III: INDIZACIÓN

*Las estructuras de indización más utilizadas por los sistemas administrativos tradicionales son bien conocidas por todos, como así también sus ventajas. Sin embargo, la naturaleza propia de los datos geográficos torna inútiles a la mayoría de tales estructuras para este dominio, por lo que a su vez demanda la creación de otras nuevas.*

*En este capítulo se introducen los tipos de consultas propios del dominio de SIG para luego presentar las principales estructuras de indización de datos geográficos. Al respecto se presentan las estructuras más utilizadas para el almacenamiento de información puntual (Quadtrees, Octrees, K-D-Trees), y una de las más utilizadas para la gestión de entidades que ocupan no ya una posición puntual sino áreas bien definidas: el R-Tree.*

## 3.1. Consultas

Las características propias de los datos geográficos, descriptas al inicio del capítulo anterior, generan la necesidad de realizar consultas que exceden el alcance de las consultas de bases de datos tradiciones. En esta sección se describen los grupos más importantes en que se clasifica a estas nuevas consultas.

### 3.1.1. Tipos de Consultas

Los tipos de consultas detallados a continuación se refieren al almacenamiento y búsqueda de objetos puntuales. Se basa en la idea de que, desde el punto de vista del modelo relacional, una base de datos espacial es un conjunto de registros, donde cada uno de esos registros corresponde a una entidad geográfica, y está formado por varios atributos o claves. La consulta devolverá todos los registros que satisfacen un predicado o tienen valores específicos o de rango para las claves especificadas. Hay tres tipos de consultas básicas a una base de datos espacial [Knuth]:

- Point Query: Es una consulta que determina si algún elemento de la base contiene a un punto dado como parámetro. Por ejemplo, consultar si existe alguna una ciudad que se encuentre ubicada en un punto determinado.

- Range Query: Es una consulta que pregunta por un conjunto de datos cuyas claves tienen valores específicos o valores dentro de los rangos dados. En este caso, podemos dar como ejemplo una consulta sobre las ciudades que se encuentran dentro del área determinada por el rectángulo con vértices opuestos con coordenadas $(x_1,y_1)$ y $(x_2,y_2)$

- Boolean Query: Es una consulta que consiste en la combinación de las anteriores con operaciones booleanas. Un ejemplo será devolver todas las ciudades que tienen coordenada $(x_1,y_1)$ o se encuentran entre los rangos $(x_2,y_2)$ y $(x_3,y_3)$

## 3.2. Estructuras de Indización Puntual

Las estructuras de indización puntual permiten el almacenamiento y operación sobre datos cuyas posiciones se representan mediante un punto. En ocasiones pueden utilizarse para almacenar objetos que tienen área simplemente eligiendo un punto determinado de la misma que los represente (por ejemplo, el de más arriba y a la izquierda). Sin embargo, esta última opción es limitada para tales casos y se estudiará una mejor solución más adelante, en esta sección se tratará el almacenamiento de información esencialmente puntual.

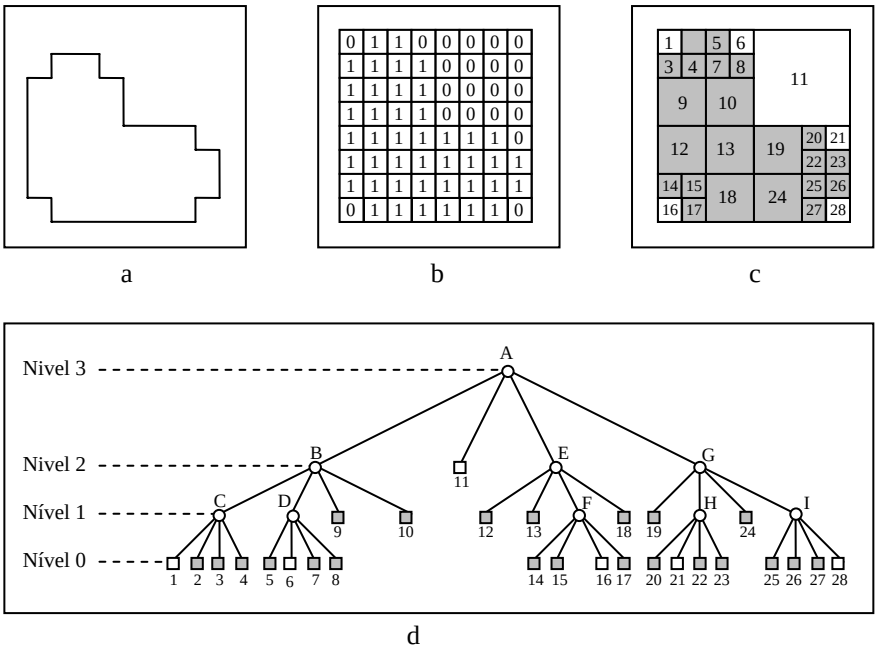
### 3.2.1. Quadtrees y Octrees

Los quadtrees son una de las técnicas más utilizadas para representar datos espaciales. Su crecimiento se debe a que, en parte, ocupan menos lugar de almacenamiento cuando se agregan datos iguales o similares. Representan un tipo de estructura de datos jerárquica basada en el principio de descomposición recursiva del espacio.

Los octrees son la extensión de los quatrees a tres dimensiones. Se basan en la subdivisión sucesiva de la región en octantes.

Se pueden diferenciar por tres elementos principales, los cuales son:

- el tipo de datos que utilizan para representar: puntos, áreas, curvas, superficies y volúmenes.
- el proceso de descomposición: puede ser en partes iguales en cada nivel o definido por la entrada de datos.
- la resolución de la descomposición: es el número de veces que el proceso de descomposición es aplicado, puede ser fijado de antemano o definido por las propiedades de la entrada de datos.




d

**Figura 3.1**.- (a) Región a representar, (b) Raster de la región anterior, (c) Bloques maximales que la región ocupa, (d) Quadtree correspondiente a la región

### 3.2.2. Region QuadTrees

Los quadtrees más estudiados son los region-quadtree, los cuales están basados en la subdivisión sucesiva de la región en cuatro cuadrantes del mismo tamaño. La región tomada es de dos dimensiones y tiene un tipo de datos binario. Si el cuadrante no contiene enteramente ceros o unos, éste se sigue subdiviendo en subcuadrantes hasta que todos los bloques tengan sólo ceros o unos.

Los procesos de construcción de los region quadtree, son del estilo bottom-up, aunque de acuerdo a cómo están definidos impliquen un proceso top-down. Se puede elegir entre los siguientes:

- Morton Order: Este proceso de construcción se utiliza para imágenes chicas. Recorre el array que representa la región en el orden que se muestra en la siguiente figura:

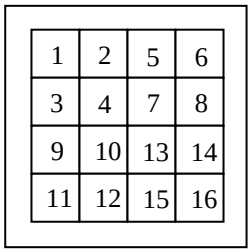



**Figura 3.2**.- Recorrido Morton

En este proceso de construcción no se crea un nodo hoja hasta que su maximal no es conocido, se divide la región en cuatro subregiones y se toma cada elemento de la imagen de izquierda a derecha y de arriba hacía abajo en cada subregión.

- Raster Scan Order: Este proceso de construcción se utiliza para imágenes grandes. Recorre el array que representa la región en el orden que se muestra en la siguiente figura:

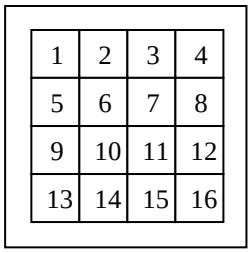



**Figura 3.3**.- Recorrido Raster Scan

En este proceso de construcción se toman los elementos de a uno y de a una fila por vez. Puede llegar a ser más lento que el recorrido Morton debido a que se deben ordenar o insertar nodos para reubicar los elementos de la imagen. Fue mejorado realizando una única inserción por cada nodo en el quadtree final y evitando realizar operaciones de mezclado [Shaf].

- Split and Merge: Este proceso de construcción se utiliza para unir o mezclar regiones adyacentes que son homogéneas. La región resultante puede no llegar a ser un quadtree. El region quadtree sólo se utiliza como paso inicial en el proceso.

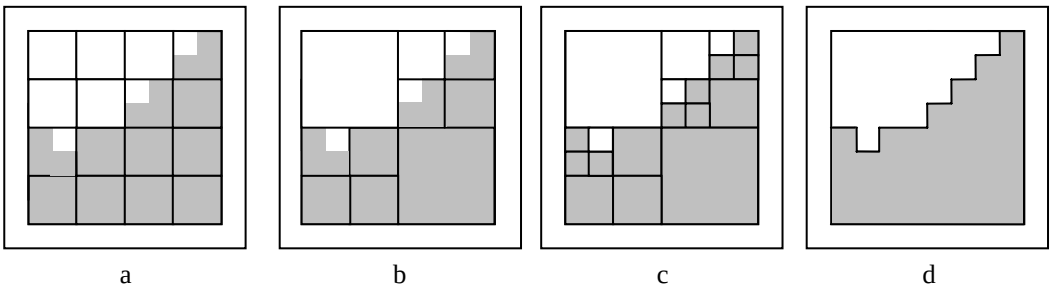



**Figura 3.4**.- Split and Merge. (a) Inicio, (b) Mezclado, (c) Particionamiento, (d) Agrupamiento

En la figura anterior, la imagen se descompone en cuadrantes de igual tamaño (a), y se intenta mezclar recursivamente las cuatro regiones hermanas que sean homogéneas (b). Luego viene el paso de descomponer las regiones que no son homogéneas hasta que se satisface un criterio de homogeneidad particular o se llega a un nivel dado. A continuación, se agrupan todas las regiones negras cuatro-adyacentes (adyacentes en dirección vertical y horizontal) en una única región negra y todas las regiones blancas ocho-adyacentes (adyacentes en dirección vertical, horizontal y diagonal) en una única región blanca.

### 3.2.3. Point QuadTrees

Este tipo de representación se puede confundir fácilmente con los region quadtree. La principal diferencia es que éstos se basan en una descomposición irregular, mientras que los region quadtree tienen una descomposición regular. Es decir, que su armado depende del orden en que se dan los puntos a almacenar y los region quadtree se dividen en sucesivamente en cuatro regiones del mismo tamaño.

Fue inventado por Finkel y Bentley [Fink], es la unión de un método de grilla y un árbol binario de búsqueda que resultó en un directorio, con forma de árbol, con celdas de distintos tamaños que contienen un elemento.

Es implementado como una generalización de un árbol binario de búsqueda. En dos dimensiones, cada punto es representado como un nodo en un quadtree. Ese nodo se representa como un registro con siete campos. Los primeros cuatro campos contienen punteros a los cuatro hijos, luego siguen los dos campos correspondientes a las coordenadas x e y del punto, y un campo para una descripción del nodo que se está representando.

Son muy utilizados para aplicaciones que realizan búsquedas por proximidad, por ejemplo devolver todos los puntos que están a cierta distancia de un punto dado.

Las operaciones principales de un point quadtree son:

- Inserción: Los registros son insertados en un point quadtree en forma similar a la inserción en los árboles binarios de búsqueda. Por cada punto, en cada nodo se realiza una comparación de las coordenadas del nodo con el punto en x e y, y del resultado de esa comparación se define con cuál subárbol se debe continuar para hacer la próxima comparación. Cuando se llega al final del árbol es donde se debe ubicar el nuevo registro. La comparación que se realiza en cada nodo es la siguiente: si la coordenada x del nodo actual del árbol es menor que la coordenada x del nodo a insertar y la coordenada y del nodo actual del árbol es menor que la coordenada y del nodo a insertar devuelve el tercer cuadrante, en cambio si la coordenada y del nodo actual del árbol es mayor que la coordenada y del nodo a insertar devuelve el primer cuadrante. Si la coordenada x del nodo actual del árbol es mayor que la coordenada x del nodo a insertar y la coordenada y del nodo actual del árbol es menor que la coordenada y del nodo a insertar devuelve el cuarto cuadrante, en cambio si la coordenada y del nodo a insertar es mayor devuelve el segundo cuadrante.

  La siguiente figura muestra la ubicación de nueve localidades que se utilizarán para insertar en un quadtree. Las localidades se insertan en el siguiente orden:

| | |
|---|---|
| 1) Olavarría | -36º 55’, -60º 15’ |
| 2) Bolivar | -36º , -61º |
| 3) Gral. Lamadrid | -37º 16’, -61º 16’ |
| 4) Tapalqué | -36º 21’, -60º 1’ |
| 5) Chillar | -37º 17’, -59º 58’ |
| 6) Tandil | -37º 19’, -59º 8’ |
| 7) Benito Juarez | -37º 45’, -59º 50’ |
| 8) Las Flores | -36º 0’, -59º 0’ |
| 9) Crnl. Pringles | -38º 0’, -61º 25’ |

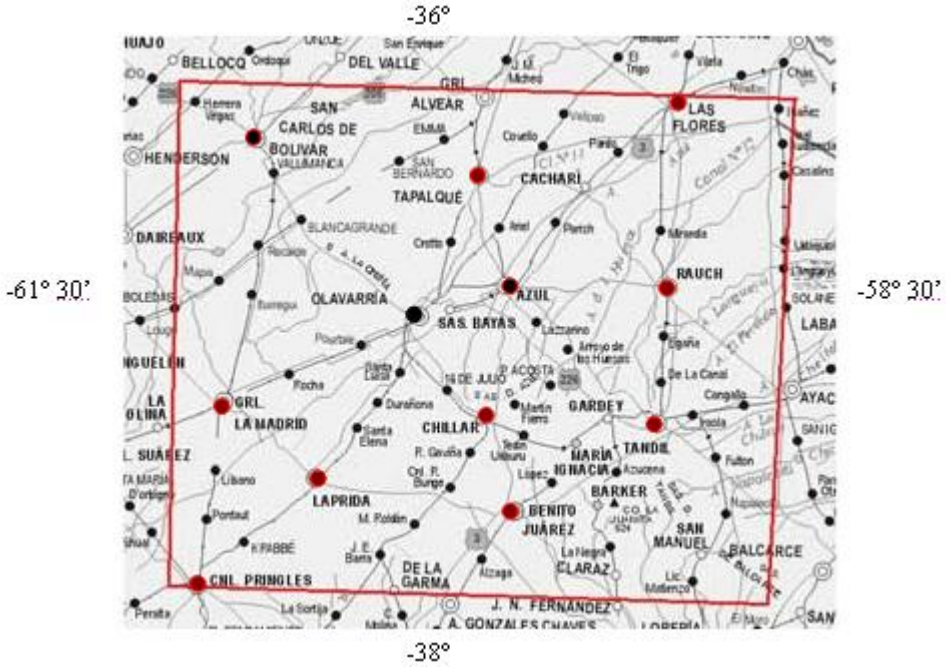



**Figura 3.5**.- Ubicación de los puntos en coordenadas (latitud, longitud)

En la siguiente figura se muestra el point quadtree, correspondiente a las nueve localidades de la Figura 3.5.

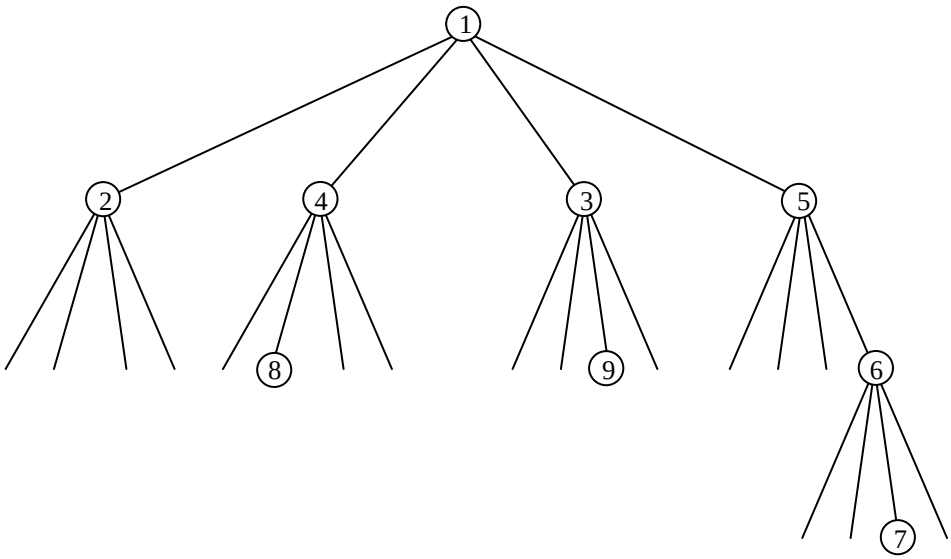



**Figura 3.6**.- Point quadtree con las localidades anteriores insertadas

- Eliminación: La eliminación es un poco más compleja. Finkel y Bentley sugieren que todos los nodos del árbol deben ser reinsertados.

  Luego Samet desarrolló un proceso más eficiente. El algoritmo es similar al de árboles binarios de búsqueda, la diferencia es que hay cuatro nodos candidatos para reemplazar al que se debe borrar. Cada nodo corresponde a un cuadrante, y la manera de encontrarlo es recursiva: por cada hijo llama a una función recursiva que se puede volver a llamar con el hijo que se encuentra en el cuadrante a 180 grados de su cuadrante, así hasta que el hijo es null. Una vez que se encontraron los cuatro candidatos, se intenta encontrar el mejor candidato para el reemplazo, para esto se siguen dos criterios: por el primer criterio el candidato elegido es el más cercano a cada uno de sus ejes de borde, que cualquiera de los tres candidatos restantes. Con este criterio pueden darse dos cosas: que no se pueda encontrar un candidato, o que se encuentre más de uno. Si hay más de un candidato con el primer criterio, entonces se aplica el segundo criterio. Este último criterio permite seleccionar el candidato cuya suma de desplazamientos de sus ejes x e y es la mínima.

- Búsquedas: Las búsquedas que se realizan son las de proximidad a un punto dado, es decir todos los puntos que se encuentran a una cierta distancia del punto. Estas búsquedas tienen un gran poder de poda del espacio de búsqueda, ya que puede ir descartando cuadrantes de manera muy fácil.

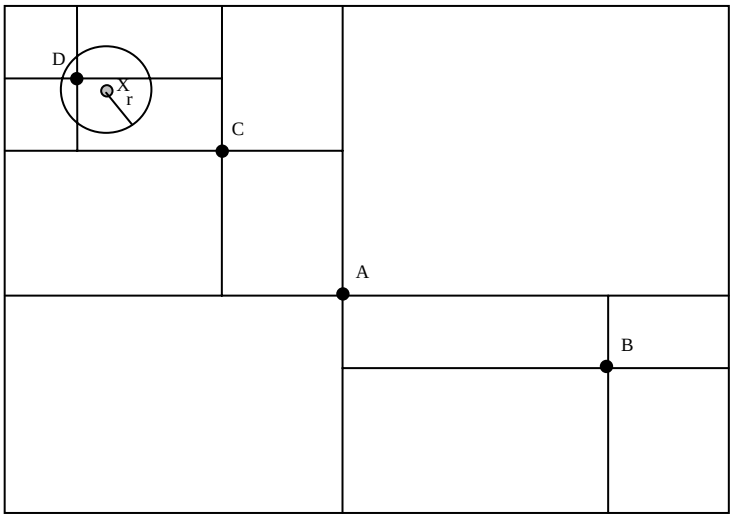



**Figura 3.7**.- Búsqueda por proximidad en un Point quadtree

En la figura anterior se puede visualizar que al buscar todos los puntos con distancia r del punto X, el espacio de búsqueda que se consultará se achica a los hijos de D. Al tomar el nodo A, ya se descartan tres cuadrantes, y se toma sólo el hijo con el nodo C, lo mismo pasa con los hijos de C, que sólo se toma el cuadrante con el hijo D, y los restantes no se examinan. Ya situados en el nodo D deben examinarse los cuatro hijos.

También se pueden utilizar otras técnicas para buscar puntos en una figura conectada, como ser algoritmos de búsqueda de regiones con rectángulos de distinto tamaño. Para manejar regiones complejas como pueden ser polígonos convexos, se definen árboles de polígonos donde el plano x-y es subdivido por J líneas que no necesariamente deben ser ortogonales. Cuando J = 2, el resultado es un point quadtree con ejes no ortogonales.

Con estas búsquedas se pueden manejar cualquiera de los tipos de queries mencionados anteriormente.

### 3.2.4. K-D Trees

Esta estructura fue inventado por Bentley [Bentley], y es una mejora del point quadtree, porque reduce el factor de ramificación por cada nodo y los requerimientos de almacenamiento. Esto se debe a tres factores:

- por cada nodo hay que consultar en todas las claves del quadtree
- los punteros o link ocupan demasiado espacio
- los nodos aumentan mucho de tamaño debido a que requiere muchas palabras

La constante "K" en el K-D Tree indica la dimensionalidad del espacio que está siendo representado. En dos dimensiones, tenemos un árbol binario de búsqueda con la diferencia de que en cada nivel el valor de clave diferente es testeado cuando se determina la dirección en la cual se debe generar la rama. Es decir en un árbol 2-D comparamos el valor de la coordenada x en las profundidades pares y la raíz, y comparamos el valor de la coordenada y en las impares.

Cada punto es representado como un nodo con un registro de seis campos. Los primeros dos campos contienen los punteros a los hijos derechos e izquierdo, luego siguen dos campos correspondientes al valor de la coordenada x y de la coordenada y del punto. En el siguiente se guarda información descriptiva sobre el punto, como ser el nombre de la ciudad que representa. El último campo indica el nombre de la coordenada que el nodo utiliza para testear.

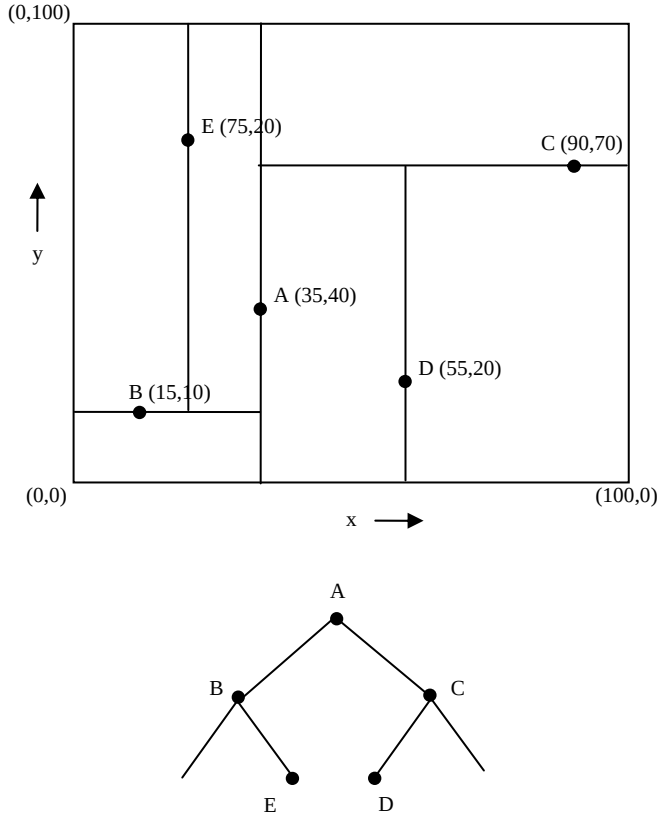



**Figura 3.8**.- Representación de un K-D Tree. En este caso, d = 2.

Se dice que un nodo es x-discriminador si todos los nodos de la izquierda de él tienen un valor menor en la coordenada x que el valor de la coordenada x de él y todos los nodos de su derecha tienen un valor mayor o igual en la coordenada x que su valor. En forma similar se define a un nodo y-discriminador.

Las operaciones principales de un K-D Tree son:

- Inserción: La inserción es similar a la de los árboles binarios de búsqueda. Se busca el registro a insertar basándose en los valores de las coordenadas x e y, comparando la

coordenada x en los niveles pares y la coordenada y en los niveles impares. La comparación que se hace en cada nodo es la siguiente: si el nivel requiere que se compare la x, entonces si la coordenada x del nodo actual es menor al coordenada x del nodo a insertar, continúa la comparación por la derecha, sino por la izquierda. De la misma forma funciona cuando el nivel requiere comparar la coordenada y. Cuando se llega al final del árbol, entonces se realiza la inserción del nuevo nodo. Al igual que los point quadtree la forma del árbol depende del orden en que inserten los nodos. En este proceso cada nodo particiona al plano en dos, de acuerdo a la coordenada que se está comparando.

- Eliminación: El borrado es un proceso recursivo. Si queremos borrar el nodo (a,b) del k-d tree, y sus subárboles son vacíos entonces reemplazamos el nodo (a,b) por el árbol vacío. En otro caso, el borrado implica reemplazar (a,b) por (c,d), y recursivamente borrar (c,d). Una vez que se borró (c,d) reemplazar (a,b) con (c,d). El problema está en cómo elegimos a (c,d), y esto se logra encontrando el nodo con el mínimo valor en la coordenada x en algún subárbol de (a,b). Es importante notar que aunque el nodo con el mínimo valor de x debe ser un subárbol de un x-discriminador, también podría estar en un subárbol de un y-discriminador.

- Búsquedas: Con este tipo de árbol se puede realizar cualquiera de los tres tipos de consultas vistos anteriormente. De la misma forma que los point quadtree, para buscar posiciones que están a una cierta distancia no se necesita recorrer todo el árbol.

## 3.3. Estructuras de Indización de Áreas

En esta sección se describen algunas metodologías relacionadas al almacenamiento y gestión de objetos que requieren ser representados geográficamente mediante toda el área que ocupan, y no sólo mediante una representación puntual. Se describen las estructuras y operatorias normales como así también algunas técnicas de carga masiva que favorecen la performance y el espacio de almacenamiento en bases de datos estáticas o poco dinámicas.

### 3.3.1. Indización de objetos con extensión espacial

Los métodos de indización expuestos hasta ahora, como los Quad-Trees y K-D-Trees, se limitan al manejo de información puntual: pueden tratar con puntos en el espacio de dos dimensiones o más, pero no pueden capturar aspectos tales como la forma de los objetos ni su extensión espacial en las dimensiones en que éste se encuentre representado.

En rigor, estas estructuras pueden utilizarse, aunque en forma limitada, para la indización de objetos cuya extensión sea mayor a cero, es decir, para aquellos que tengan área, volumen e incluso extensión en dimensiones superiores. Esto se logra estableciendo algunas convenciones acerca del modo de almacenamiento y búsqueda de los objetos; en concreto, se toma sólo un punto representativo de cada objeto y es este dato puntual el que se almacena en la estructura suplantando al objeto y mediante el cual se lo busca. Por ejemplo, podría convenirse que se almacena el centroide o la esquina superior derecha de cada uno, siempre, por supuesto, asociado a una referencia al objeto real. En la Figura 3.9 se muestra un conjunto de objetos y su indización mediante un Quad-Tree.

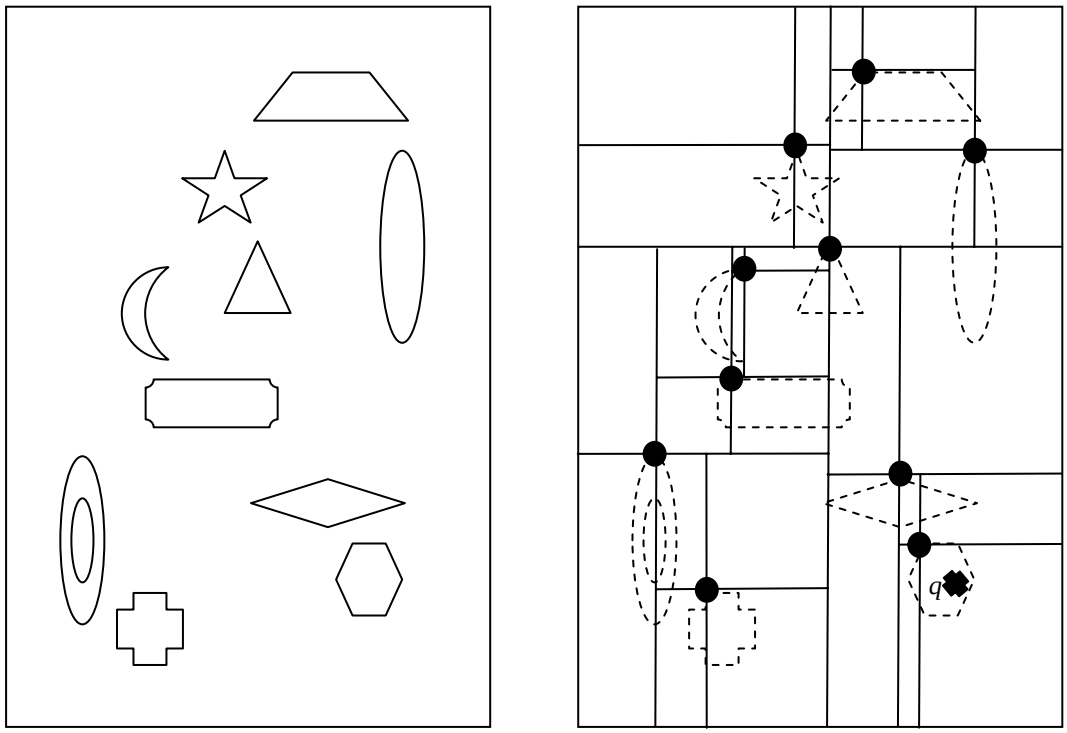



**Figura 3.9**.- (a) Conjunto de objetos distribuidos espacialmente, (b) Quad-Tree que almacena sólo un punto determinado de cada objeto (más una referencia al mismo), *q* representa un punto de consulta.

Para el caso mostrado en la figura anterior se acuerda almacenar el punto más alto de cada objeto, y en caso de haber varios puntos superiores se toma el que más a la izquierda se encuentre. Como podemos observar la estructura sólo permite recuperar objetos ya conocidos y además no permite saber si algún objeto contiene o no contiene a *q*, el punto de consulta.

Esta forma de indización para objetos no puntuales es limitada en cuanto a que sólo permite búsquedas muy específicas y demasiado acotadas. Al no contemplar la extensión espacial, la estructura pierde información usualmente útil a los fines funcionales de una aplicación. Por ejemplo, resulta imposible obtener todos los objetos que se solapan con un área dada como dato, obtener todos los objetos que se encuentran en un punto dado, y tampoco es posible saber, mediante la misma estructura, si un objeto se solapa con algún otro también contenido en ella. Puede verse que esta metodología efectivamente permite el almacenamiento de objetos con extensión espacial, pero no soporta un conjunto mínimo deseable de operaciones de consulta.

Para hacer frente eficazmente a ambos aspectos (almacenamiento y consultas) han surgido varias estructuras que contemplan la extensión espacial de los objetos en dos o más dimensiones, una de las más utilizadas es el R-Tree.

### 3.3.2. R-Trees

El R-Tree es una estructura de indización espacial en forma de árbol y con propiedades y algorítmica similares a las del B-Tree. Esta estructura contempla la forma de los objetos en dos, tres o cualquier cantidad de dimensiones. No sólo permite la indización de los mismos sino que también permite un conjunto de consultas bastante rico en cuanto a características espaciales: permite, entre otras operaciones, obtener los objetos ubicados en una posición puntual determinada, obtener todos los objetos que se superponen con un segmento, área o volumen ingresado como dato, y obtener todos los objetos que se encuentran dentro de la misma estructura que se superponen.

El concepto fundamental en el que se basa el R-Tree es el de bounding-box. El bounding-box de un objeto espacial en dos dimensiones es el rectángulo más chico que lo contiene. En tres dimensiones, se

define como el cubo que cumpla la misma condición, y en dimensiones superiores, como “figuras” cuyos lados son siempre paralelos a los ejes dimensionales.

En Geometría Computacional, tema que se desarrollará cuando hablemos de operaciones geométricas, el concepto de bounding-box se utiliza como una mejora a la performance de algoritmos tales como el de testeo de inclusión de puntos en polígonos o superposición de polígonos, justamente las consultas que se desean hacer al R-Tree. La idea reside en considerar el bounding-box del objeto para intentar evitar el chequeo de orden lineal (de inclusión o superposición) que considera siempre la forma concreta del mismo: sabemos que si el bounding-box de un objeto no contiene un punto, entonces el objeto tampoco. Un primer chequeo contra el bounding-box puede permitir saber en tiempo constante que un objeto no contiene un punto o no se superpone con otro objeto. En el caso en que se detectara inclusión o superposición con respecto al bounding-box, debe irremediablemente procederse a un chequeo contra la forma particular del objeto, sin obtener ganancia alguna en el tiempo de ejecución.

El R-Tree explota esta facilidad al máximo, considerando el bounding-box no sólo de los objetos indexados sino también el de los nodos que lo componen. Al igual que en el B-Tree, en el R-Tree las referencias a los objetos almacenados se encuentran sólo en los nodos hoja. Cada una de estas hojas tiene un conjunto de entradas de registros de índice, cada uno de los cuales consiste en una tupla de la forma (R,O), donde O es una referencia al objeto almacenado y R es el bounding-box del mismo. Por su parte, los nodos no-hoja tienen exactamente la misma estructura, sólo que el campo O de cada entrada hace referencia a un nodo hijo en el árbol, mientras que el campo R representa al bounding-box que contiene espacialmente a todos los bounding-boxes que figuran en las entradas de registro del hijo referenciado.

Como se verá más adelante, las propiedades de un R-Tree determinan que los nodos pueden tener un número máximo M de elementos, determinado de antemano e igual para todos. En la Figura 3.10 se muestra un conjunto de objetos distribuidos espacialmente y los bounding-boxes de la estructura de R-Tree asociada para M = 3, se muestra también la estructura en términos de nodos y referencias entre ellos.

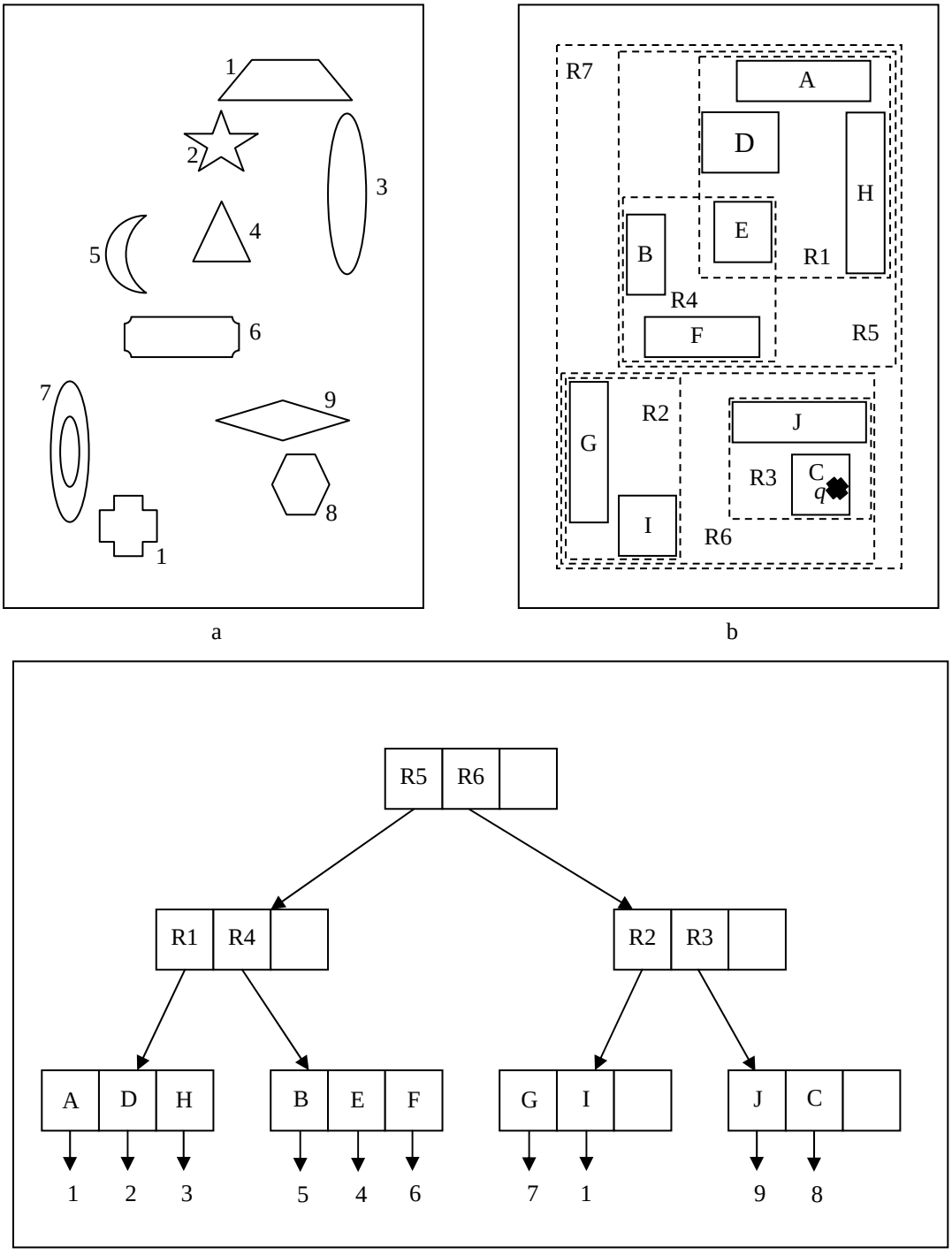



**Figura 3.10**.- (a) Conjunto de objetos distribuidos espacialmente, (b) extensiones espaciales de los bounding-boxes de los objetos y de los nodos del R-Tree que los indexa, más un punto de consulta *q;* y (c) representación de los nodos y referencias del R-Tree.

En esta figura, el bounding-box de cada objeto se nombra con una letra, mientras que para los rectángulos correspondientes a los nodos del R-Tree se utiliza la nomenclatura Ri. R7 constituye el bounding-box de la raíz y se muestra sólo por claridad, ya que generalmente no se incluye en la estructura y, o bien se lo mantiene fuera de ella o bien no se lo utiliza, ambas alternativas presentan ventajas y

desventajas cuyos detalles no serán estudiados aquí. Podemos ver que la estructura permite saber que el hexágono con bounding-box C es el único objeto que podría contener a *q*.

Cabe aclarar que varios de los lados de los rectángulos en realidad se encuentran superpuestos, aunque por razones de legibilidad en la figura se encuentren levemente separados uno del otro, por ejemplo los rectángulos H, R1, R5 y R7 se superponen todos en su lado derecho.

En esta sección se describirán de manera informal los algoritmos de búsqueda, inserción y borrado asociados a la estructura, si se desea un enfoque más riguroso puede consultarse el pseudo-código de los mismos en el Apéndice A. Sí creemos necesario describir formalmente las propiedades que debe cumplir un R-Tree:

> Sea M el número máximo de entradas que pueden caber en un nodo, y m <= M/2 un parámetro que especifica el número mínimo de entradas de un nodo, un R-Tree satisface las propiedades que se listan a continuación:
>
> 1.- Cada nodo hoja contiene entre m y M registros de índice, salvo en el caso de que a su vez este nodo constituya la raíz.
> 2.- Por cada registro de índice (I, identificador de tupla) en un nodo hoja, I es el bounding-box (en n-dimensiones) más chico que contiene espacialmente al objeto dato n-dimensional representado por la tupla indicada.
> 3.- Cada nodo no-hoja tiene entre m y M hijos, salvo para el caso en que también constituya la raíz.
> 4.- Por cada entrada (I, puntero al hijo) en un nodo no-hoja, I es el bounding-box (en n-dimensiones) más chico que contiene espacialmente a los bounding-boxes del nodo hijo al que referencia.
> 5.- El nodo raíz tiene al menos dos hijos, salvo en el caso en que constituya, a su vez, la raíz.
> 6.- Todas las hojas están al mismo nivel.

Se llama orden de un R-Tree a la tupla (m,M) que lo define, de modo que el orden del árbol de la figura anterior es (2,3).

A diferencia del Quad-Tree, esta estructura provee una aproximación bastante cercana a la extensión espacial de cada objeto, lo cual permite un conjunto más grande de consultas que pueden realizarse. De hecho, a partir de la indización propuesta en la figura anterior, podemos saber con certeza que el único objeto que podría contener al punto *q* es el hexágono, ya que es el único cuyo bounding-box contiene al punto. Sin embargo, como éste podría estar contenido en el bounding-box pero no en el objeto, debe recurrirse a algún chequeo de inclusión geométrico (punto-en-polígono) para determinar si realmente este objeto lo contiene, estos métodos de chequeo se describirán más adelante.

Al igual que muchas estructuras de indización en forma de árbol, también en este caso se da que para un mismo conjunto de objetos pueden existir varias estructuras distintas de R-Tree que los indexen. Esto depende, entre otras cosas, del orden en que se inserten los objetos. Además puede darse también la superposición espacial de nodos, y aún la inclusión espacial total de un nodo en el bounding-box de otro que sin embargo no lo contiene en la estructura. Esto puede verse en la Figura 3.10 para el caso del bounding-box E, ya que el mismo se encuentra de forma accidental espacialmente contenido en R1, sin embargo no se encuentra alocado estructuralmente en el nodo al que referencia R1, sino en el referenciado por R4.

El algoritmo de búsqueda toma un parámetro *q* que puede ser un punto, un rectángulo o un segmento y devuelve todas las referencias a objetos cuyos bounding-boxes se superponen parcial o totalmente con *q*. Consiste en un algoritmo recursivo que comienza en el nodo raíz: se toma una de sus entradas, digamos E1, y se evalúa si su bounding-box incluye a *q*, de no ser así se sabe que todo el subárbol referenciado por E1 tampoco lo contiene, por lo tanto se evita la infructuosa búsqueda en él y se pasa a la siguiente entrada del nodo raíz para repetir el mismo proceso.

En el caso de que E1 sí contuviese a *q*, se dispararía recursivamente el algoritmo sobre el nodo que referencia, su hijo. Es importante destacar que al volver de la invocación recursiva, aún en este caso exitoso y al igual que en el caso anterior, el algoritmo debe continuar la búsqueda en la siguiente entrada (digamos E2) ya que puede haber otros objetos en la misma área pero almacenados en otros subárboles, pues los nodos pueden superponerse espacialmente (caso del bounding-box E de la figura). Como resultado, el algoritmo devuelve todas las referencias a objetos reales cuyos bounding-boxes se

superponen con el dato de entrada *q*, es responsabilidad del llamador de este algoritmo determinar si cada objeto devuelto realmente contiene a *q*.

El algoritmo de inserción es similar al utilizado para B-Trees, se trata de un algoritmo recursivo que hace las inserciones en las hojas, y a medida que vuelve de la recursión hace actualizaciones en cada nodo de manera de dejar consistente la estructura con respecto a los cambios realizados en los niveles inferiores.

El nodo hoja en el cual se va a insertar la nueva entrada se determina recorriendo el árbol desde la raíz hacia abajo, eligiendo en cada paso el subárbol cuyo bounding-box haya que agrandar menos para que pueda contener al bounding-box de entrada.

Una vez que se haya localizado el nodo hoja en el cual se lo debe alocar, se realiza un chequeo para determinar si esta inserción causará un overflow en el nodo. Si no es así, se lo inserta y comienza la vuelta de la recursión agrandando los bounding-boxes de cada nivel de manera que cubran a los del nodo inferior, quizá más agrandados por la inserción en cuestión. Por el contrario, si la nueva inserción efectivamente causa overflow, el nodo hoja debe partirse en dos. Esto significa crear un nuevo nodo hoja, repartir las M + 1 entradas entre ambos e insertar el nuevo nodo hoja en el padre del nodo hoja original. Esto último podría causar que el nodo padre también cayera en overflow, lo que demandaría una nueva partición y así sucesivamente; es decir que las particiones pueden propagarse hacia arriba, y la vuelta de la recursión puede no sólo involucrar actualizaciones de bounding-boxes sino también la gestión de particiones.

En el caso de que una partición se propague hasta la raíz causando también la suya, debe crearse una nueva raíz cuyos hijos serán la raíz original y el nuevo hermano de ésta, dando lugar al crecimiento en altura del árbol.

Los algoritmos de partición son algoritmos auxiliares que se utilizan durante el proceso de inserción cada vez que sea necesario partir nodos. Existen varias particiones posibles para un mismo nodo, y cada una básicamente determina la manera en que deben distribuirse los elementos entre los dos nodos. No da lo mismo utilizar cualquiera de ellas ya que existen particiones de distinta calidad. Se considera que una partición es buena cuando la distribución de entradas resulta en dos bounding-boxes que en el futuro evitarán consultas innecesarias. Esto puede conseguirse minimizando el área de superposición de los nodos o bien minimizando el área total de cobertura. En algunos casos estos dos objetivos pueden ser contradictorios, como lo demuestran las dos posibles particiones descriptas en la Figura 3.11.

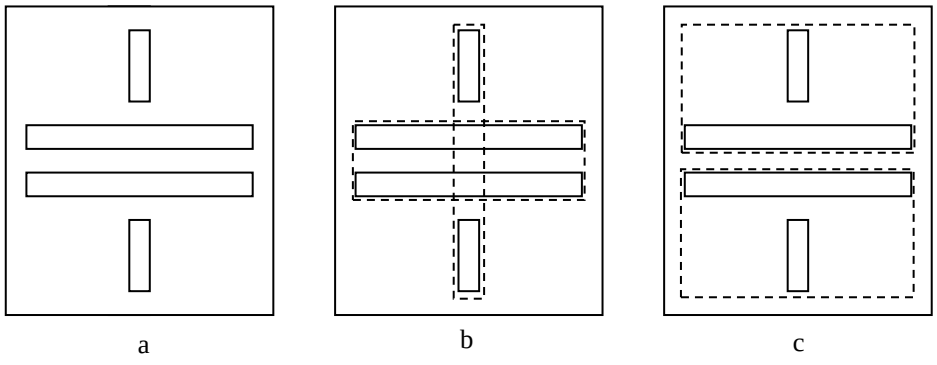



**Figura 3.11**.- (a) Cuatro bounding-boxes y dos formar posibles de distribuirlos en una partición: (b) minimizando el área total de cobertura y (c) minimizando el área de superposición de los nodos generados.

Puede verse que la figura “c” desperdicia un área mayor, por lo que es propensa a demandar consultas con no muy buena probabilidad de éxito. La partición de la figura “b” logra un área mucho más ajustada, aunque las consultas sobre la zona de superposición implicarán que deban consultarse innecesariamente ambos nodos; no obstante, es ésta la alternativa generalmente preferida.

Existen algoritmos para obtener cada tipo de partición. El óptimo es el exhaustivo y puede utilizarse para obtener ambos tipos; el mismo consiste en probar todas las alternativas de distribución de elementos y quedarse con la mejor, pero esto implica 2M – 1 alternativas posibles, y como es de esperarse, su performance no es adecuada para los valores de M que suelen manejarse. Existen alternativas que a costa de la pérdida de algo de calidad permiten particiones con mejor eficiencia en el tiempo: en el Apéndice A pueden consultarse dos algoritmos, uno de orden lineal y otro de orden cuadrático, que generalmente se utilizan con buenos resultados. También existen estudios de particiones de dos a tres nodos que permiten obtener resultados un poco mejores en cuanto a minimización de área total de cobertura, sin embargo esta diferencia no es sustancial, por lo que no se justifica el precio de su mayor orden de ejecución y no se tratan en este trabajo.

El algoritmo de modificación se utiliza cuando cambia la ubicación, forma o tamaño de un objeto con su consecuente cambio en el bounding-box. El algoritmo original [Guttman] consiste simplemente en borrar el objeto de la estructura mediante el algoritmo de borrado normal, efectuar los cambios necesarios y volver a insertarlo de manera que encuentre su lugar correcto en la estructura. Si bien esta metodología puede dejar la sensación de que se realiza trabajo innecesario al no realizar actualizaciones sólo a nivel local, debe pensarse que algunas modificaciones son muy difíciles de tratar de otra manera y pueden traer consecuencias graves en la estructura del R-Tree. Por ejemplo, si ante el cambio considerable en la posición de un objeto agrandáramos todos los bounding-boxes de sus ancestros, estaríamos ocupando demasiado espacio innecesario con sus bounding-boxes, este deterioro de la estructura demandaría una mayor cantidad de consultas posteriores, también innecesarias, obteniendo como resultado final una notable baja en la performance.

Existen metodologías más avanzadas que en ciertos casos permiten reacomodos locales, pero es seguro que la original, que es la que se menciona aquí, contribuye a un mantenimiento saludable de la estructura.

El algoritmo de borrado de un objeto, o mejor dicho, de su referencia, es recursivo y consiste en ubicar el nodo hoja en donde se encuentra la entrada a borrar. Esto se realiza de la misma manera que la búsqueda, pero al llegar a la hoja se debe eliminar la entrada, y luego, mientras se vuelve de la recursión, se deben ir ajustando (achicando lo más posible) los bounding-boxes de cada nodo de manera de contemplar la ausencia espacial del elemento eliminado.

Un aspecto a tener en cuenta es que la eliminación de una entrada puede ocasionar que un nodo hoja caiga en underflow, es decir que la cantidad de entradas restantes no alcancen el número mínimo m requerido por el orden del árbol. En estos casos, estas entradas se almacenan en un conjunto temporal (para tratarlas al final), a continuación se elimina el nodo que las contenía, y continuando con la vuelta de la recursión, se pasa al nodo del nivel superior para ajustar su bounding-box. Esta eliminación de nodos puede causar, a su vez, underflow en los nodos superiores, en cuyo caso se repite con ellos el mismo procedimiento descripto. Luego de la vuelta de la recursión se reinsertan todas las entradas que se han almacenado en el conjunto, contemplando que algunas inserciones deben realizarse en niveles superiores, pues pueden involucrar subárboles completos cuyas hojas deben coincidir en altura con las del árbol original.

Aquí radica una de las diferencias más grandes con respecto al B-Tree, en donde en estos casos se recurre a arreglos locales consistentes en el merging con nodos hermanos. Si bien esta técnica es más difícil de implementar en R-Trees (pues no hay un orden total como en el espacio unidimensional), existen algunas metodologías que permiten hacerlo. Sin embargo los algoritmos originales utilizan la reinserción descripta por algunas razones. Una de ellas es que es más fácil de implementar, pues se reutilizan algoritmos ya existentes. Otra es que no es tan ineficiente como parece, pues las páginas utilizadas en la reinserción suelen ser las mismas que las de la búsqueda anterior, por lo que es altamente probable que se encuentren en memoria. Y por último, siguiendo la misma filosofía del algoritmo de borrado, se considera que la reinserción refina la estructura espacial del árbol, evitando el deterioro que ocasionaría la permanencia de los nodos siempre bajo el mismo nodo padre.

Todo lo expuesto hasta aquí trata de la estructura original del R-Tree, sin embargo existen muchas variantes del mismo, cada una intentando mejorar algún aspecto en particular.

El R+-Tree (“ó R-Tree más”) [Samet] evita la superposición de los nodos no-hojas del árbol, con lo cual ahorra consultas innecesarias en más de una rama. El precio a pagar por esto es que el objeto figura en varios nodos hoja al mismo tiempo, lo cual a su vez brinda la ventaja de poder ser accedido desde cualquiera de ellos. Esto significa que existen varios caminos desde la raíz hasta el bounding-box del objeto, por lo que una consulta sobre el mismo que comience en la raíz siempre llegará hasta él en el

primer intento, evitando consultas infructuosas en ramas que no lo contienen. Si bien esta alternativa utiliza más espacio y la altura del árbol suele ser algo mayor que la original, también se obtiene, en general, una mayor eficiencia en los tiempos de respuesta de las consultas.

Otra alternativa es el R*-Tree (ó "R-Tree estrella") [Beckmann] cuya característica principal es la implementación del concepto de forced-reinsertion. El mismo consiste en que cuando un nodo cae en overflow, algunos de sus hijos son cuidadosamente seleccionados y reinsertados, lo cual contribuye a una mejora progresiva de la estructura del árbol.

Tanto la original como todas estas alternativas tratan con la indización de objetos en el espacio pero no se enfocan sobre aspectos temporales. Todas permiten saber en donde están los objetos ahora, pero no permiten saber en dónde estaban en el pasado ni tampoco saber que objetos había antes y que ya no figuran por haber sido eliminados. Los R-Tree Temporales [Nascimento] permiten una indexación espacio-temporal, es decir que a todas las consultas espaciales que pueden realizarse mediante cualquier alternativa de los R-Trees descriptos hasta ahora, puede agregársele una componente temporal que indique un momento puntual o un rango temporal, es decir, el lapso de tiempo en el que debe considerarse la distribución espacial de los objetos. Pueden realizarse consultas del tipo "obtener todos los objetos que se encontraban en el punto *q* en el momento *t*", y otras más complejas como "obtener todos los objetos que hallan estado ubicados en el área rectangular *R* en el lapso de tiempo desde *t1* hasta *t2*", o bien "obtener el momento *t* en el que por primera vez el objeto *O* se ubicó en la posición *q*". Esta funcionalidad se logra mediante la implementación de la duplicación de ramas de cada nodo. Cada nodo contiene un conjunto de entradas espaciales pero clasificadas según el tiempo en que fue generada o modificada por última vez. Ante cualquier cambio espacial que la afecte (inserción, borrado, modificación) se hace una copia de la rama original y se la ubica en una nueva entrada identificada con el momento *t* actual, y los cambios se realizan sobre esta nueva rama, que a partir de este momento constituirá la "rama actual". De esta manera, se sigue conservando intacta, "congelada", la rama original que es la que contiene la información espacial en el lapso de tiempo transcurrido desde el último cambio que la ha afectado hasta el momento *t*.

Las estructuras descriptas hasta ahora, tanto las espaciales como la espacio-temporal, son eficientes en la gestión de información dinámica (altas y bajas frecuentes). Sin embargo, cuando la información a tratar es estática, puede explotarse esta característica para mejorar la estructura resultante, obteniendo a la vez mejores tiempos de respuesta y menor utilización en el espacio de almacenamiento. Algunas de estas técnicas contemplan la carga masiva (bulk-loading) de un R-Tree, lo que da lugar a los R-Tree Empaquetados o Packed R-Trees.

### 3.3.3. R-Trees empaquetados

Se considera dinámica a una Base de Datos cuando sobre ella se realizan frecuentes operaciones de alta, baja y modificación. Sin embargo existen muchas aplicaciones cuya naturaleza es neta o prácticamente estática, es decir que una vez almacenada la información las únicas operaciones (o casi las únicas) que se realizan sobre ella son las de consulta. Ejemplos de este tipo de BD son las cartográficas, en donde una vez almacenado un mapa sólo se prosigue mediante operaciones de consulta; las bases que almacenan información de censos presentan la misma característica, como así también las que almacenan información meteorológica. Como veremos a continuación, conocer de antemano la calidad de estática de una base de datos espacial permite aprovechar este hecho para obtener una mejor estructura del R-Tree en términos de tamaño y tiempos de respuesta.

Como se ha expuesto anteriormente, a un mismo conjunto de objetos no siempre le corresponde la misma estructura de R-Tree, sino que ésta puede variar dependiendo, entre otras cosas, del orden en que se insertan tales objetos. Las estructuras obtenidas mediante las operaciones de alta, baja y modificación expuestas anteriormente son buenas, pero no son las óptimas. Cabe aclarar que los mismos algoritmos evitarán que se llegue a casos demasiado indeseables o extremos, pero también es preciso notar que las estructuras resultantes podrían ser aún mejores. Una de las causas es, por ejemplo, que un orden de inserción puede inducir a que algunos bounding-boxes de nodos ocupen más área que el que ocuparían mediante la inserción de los mismos objetos en distinto orden; ya sabemos que mayor área de nodos implica mayor número de consultas innecesarias. Otra causa viene ligada al concepto de empaquetamiento del R-Tree, el cual se describe a continuación.

La cantidad de elementos que puede tener un nodo varía entre un número mínimo m y un máximo M, es decir que los nodos pueden tener espacios para entradas inutilizados. El grado de empaquetamiento de un R-Tree es una medida de la cantidad de espacio al que están completos los nodos de la estructura; un empaquetamiento al 100% significa que todos los nodos están llenos al máximo de su

capacidad. Por lo general, cada nodo se mapea a una página de disco, y en cada operación de consulta que lo involucra se lleva la página completa a memoria principal. De manera que un nivel bajo de empaquetamiento deriva, en primer lugar, en un mayor espacio de almacenamiento en memoria secundaria. En segundo término, implica una falta de aprovechamiento de la memoria principal, pues pueden cargarse páginas completas para realizar testeos sobre unas pocas entradas de cada página, lo que demandará nuevos accesos a memoria secundaria para recuperar nuevas páginas a cuyas entradas poder consultar. Por último, y este punto viene asociado al anterior, un bajo nivel de empaquetamiento implica que la indización de un conjunto de objetos se realice mediante una cantidad de nodos mayor que la que podría utilizarse para indexar ese mismo conjunto; mayor cantidad de nodos determina una mayor altura del árbol, lo que redunda, a su vez, en una mayor cantidad de consultas (accesos a disco) de nodos (páginas) para llegar a los datos, que se encuentran en las hojas.

La utilización de espacio, o empaquetamiento, de un R-Tree dinámico está garantizada en un mínimo del 50%, y resultados experimentales muestran que el promedio es de un 70%. El objetivo de los R-Trees empaquetados es alcanzar el 100% de utilización del espacio.

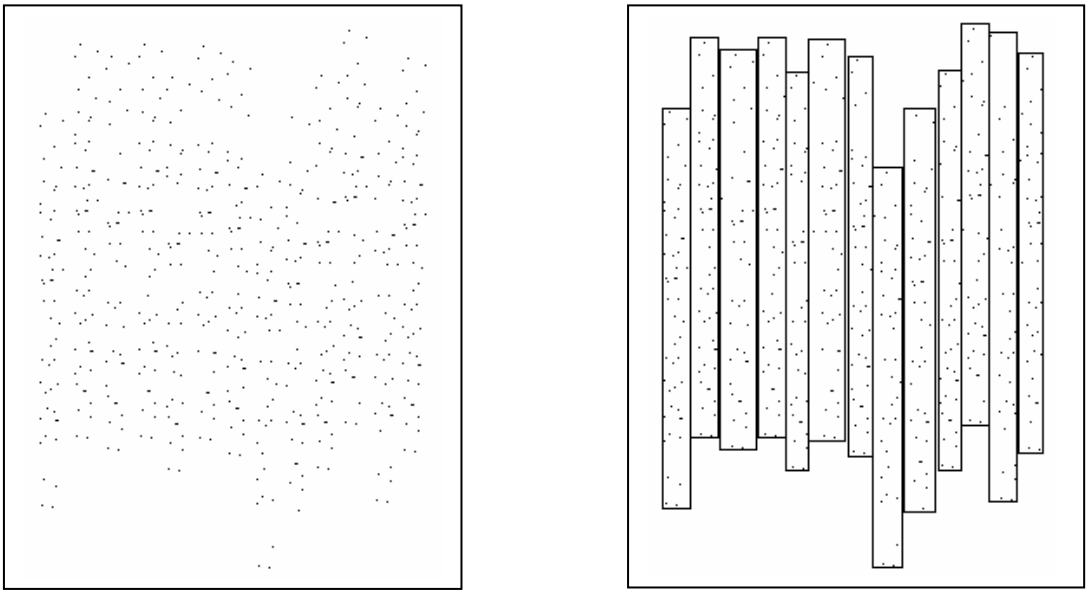
Las técnicas de empaquetamiento se basan en el hecho de que por tratarse de bases estáticas, se conoce toda la información de antemano, de manera que el árbol puede construirse mediante formas alternativas al algoritmo de inserción descripto anteriormente, siempre cuidando que la estructura resultante cumpla con las propiedades de todo R-Tree. Estas técnicas consisten en construir el árbol desde abajo hacia arriba: primero se agrupan los bounding-boxes de los objetos en grupos de M elementos cada uno (siendo M el máximo permitido por nodo) y se asigna cada grupo a un nodo, luego se construye el segundo nivel de la misma manera pero tomando ahora como entrada los nodos del primer nivel. El proceso se repite recursivamente hasta llegar a formar el nodo raíz. A esta metodología se la conoce también como carga masiva (bulk-loading) y permite un empaquetamiento del 100% o muy cercano, puesto que no puede asegurarse que el último nodo de cada nivel pueda llenarse al máximo de su capacidad.

Existen varias estrategias de empaquetamiento y todas cumplen con la secuencia de pasos descripta, la diferencia entre una y otra reside en la forma en que se seleccionan los nodos que conformarán cada grupo. A continuación se describirán dos de tales técnicas: una muy rudimentaria pero útil para la indización de puntos, y una segunda, más compleja, basada en fractales.

### 3.3.4. Empaquetamiento Horizontal o Vertical

Este método [Roussopoulos] establece un orden de los rectángulos considerando sólo la coordenada x ó y de una de las esquinas de los mismos. Una vez ordenados, recorre la lista comenzando por el primero y va asignando los rectángulos al primer nodo hasta colmar su capacidad. Una vez lleno, crea otro nodo vacío y sigue repitiendo el proceso con el resto de la lista, hasta que todos sus elementos han sido asignados a algún nodo. De esta manera se consigue un empaquetamiento del 100% o muy cercano.

Este algoritmo mejora ampliamente la performance de los R-Trees originales al tratar con datos puntuales, sin embargo no ocurre lo mismo al indexar objetos con extensión espacial. Para estudiar de manera informal este hecho puede observarse la Figura 3.12, en donde se muestra un conjunto de puntos y el empaquetamiento (sólo del primer paso) logrado por este algoritmo. Se considera el orden por la coordenada x de la esquina inferior izquierda, que en la figura, al mostrar sólo el primer paso, se degenera en la coordenada x de los puntos.



a b

**Figura 3.12**.- (a) Conjunto de puntos aleatoriamente distribuidos en el espacio y (b) último nivel del *R-Tree empaquetado* generado por la carga masiva mediante el orden de nodos obtenido considerando solamente la coordenada x de cada uno.

Puede verse que la superficie de cada nodo es bastante chica, y además éstos no se superponen salvo el caso en el cual el último punto de un nodo tiene la misma coordenada x que el primero del próximo, y aún en este caso se superpondrían sólo en un lado. La poca área ocupada y la casi nula superposición permiten reducir la cantidad de consultas innecesarias, que junto con la baja altura del árbol logrado por el alto grado de empaquetamiento explican perfectamente la causa de la mejora con respecto a la alternativa original.

Sin embargo esta mejora no ocurre al indexar objetos con extensión espacial. Intuitivamente puede verse que un buen empaquetamiento debería asignar objetos cercanos a un mismo nodo, y esto no ocurre con este algoritmo ya que ignora una coordenada. En el ejemplo, los objetos que se extienden horizontalmente podrían generar nodos de un área muy grande y mucha superposición con otros, debido a que posiblemente se los asociaría al mismo nodo que objetos que se encuentran muy distantes a él pero con una coordenada x cercana a la suya.

El problema aquí es que no se contempla el sentido de vecindad, es este aspecto el que trata de resolver la siguiente metodología de empaquetamiento.

### 3.3.5. Empaquetamiento Basado en Fractales

Mediante esta metodología [Faloustos] se construyen nodos con objetos de buena proximidad espacial a través de curvas de llenado del espacio o fractales. Un fractal es una curva que recorre todos los puntos de una grilla k-dimensional, visitando cada uno una única vez y sin cortarse a sí misma. Existen varias curvas que cumplen esta condición, una de ellas es la Z-Order o Morton-Order, descripta anteriormente en este trabajo. Otra es la curva de Hilbert, que es la que constituye la base de la metodología que se expone aquí.

La curva de Hilbert presenta una forma más compleja que la Morton-Order, y su definición parte de una forma básica y combinaciones a partir de ella. La forma básica corresponde a una grilla de 2x2 y se muestra en la Figura 3.13.a. La curva de Hilbert de orden i se obtiene, reemplazando cada vértice de la curva básica con la curva de orden i – 1, que puede adecuadamente rotada o reflejada. Cuando el orden de esta curva tiende a infinito, la curva resultante es un fractal.

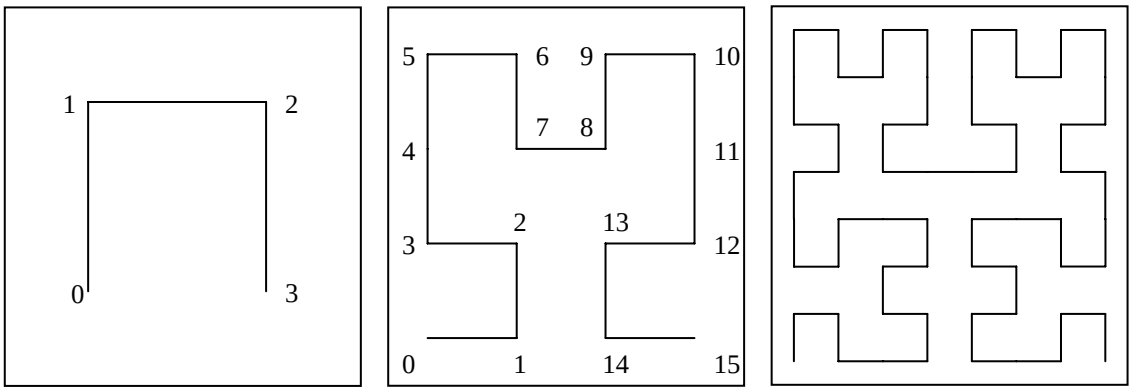



**Figura 3.13**.- (a) Curva de Hilbert de primer orden, (b) de segundo orden y (c) de tercer orden. Los números asociados indican el valor de Hilbert de cada punto.

El camino de esta curva determina un orden de las grillas que recorre, que se obtiene recorriéndola desde un extremo hasta el otro. Las figuras a y b muestran este orden para las curvas de ordenes 1 y 2. El valor que se le asigna a cada grilla según este orden se denomina valor de Hilbert. Mapeando estas grillas a un espacio discreto de puntos, podemos decir que, en la figura b, el punto (0,0) tiene un valor de Hilbert de 0, mientras que al punto (1,1) le corresponde el valor 2.

Intuitivamente puede verse que dos puntos que tienen valores cercanos de Hilbert también se encuentran cercanos en el espacio: el orden de Hilbert implica cercanía espacial (no vale la inversa), lo cual no sucedía con el método anterior. La construcción de nodos se hace de la misma manera que con el empaquetamiento precedente, sólo que ahora se considera el orden de Hilbert. Los nodos generados mediante este método tienen a ser chicos y con formas tendientes al cuadrado, esto implica poca área y poco perímetro. Para obtener buena performance en consultas puntuales se necesita nodos de poca área, mientras que para lograr performance en consultas de mayor rango se necesita, además, que tengan poco perímetro. Un análisis formal de este aspecto queda fuera del alcance de este trabajo pero puede ser consultado en las fuentes referenciadas.

Para concluir con la definición del método de empaquetamiento sólo basta determinar como mapear un rectángulo a un punto, ya que éste es el que determina el valor de Hilbert de aquel. Luego de varios estudios se concluyó que los mejores resultados se obtienen al considerar el centro de cada rectángulo, por lo tanto el valor de Hilbert de un bounding-box es el de su centro y queda así determinado el orden para los rectángulos.

El pseudo-código de este algoritmo de empaquetamiento puede consultarse en el Apéndice A.

# 4. CAPÍTULO IV: GEOMETRÍA COMPUTACIONAL

*La geometría tradicional define representaciones de objetos geométricos y un conjunto de operaciones que puede realizarse con ellos. Sin embargo, estas metodologías no siempre pueden ser llevadas (o al menos no resulta sencillo) a un modelo computacional, es decir, a estructuras de datos y algoritmos. Por ejemplo, el cálculo de la distancia entre dos puntos sobre una superficie se realiza tradicionalmente mediante una integral, sin embargo la obtención de la primitiva que la define es un proceso no muy fácil de implementar mediante un algoritmo. Además, se deben considerar limitaciones como la cantidad de memoria disponible, la precisión numérica que una máquina es capaz de manejar y el tiempo de ejecución tolerable requerido por la aplicación.*

*El objetivo de la Geometría Computacional podría definirse, de manera sintética, como la reformulación (cuando sea necesario) de representaciones y procedimientos de la Geometría Tradicional de manera de adaptarlos al área computacional [Preparata].*

*Las áreas de aplicación de esta disciplina son muy variadas, entre ellas se encuentran el Diseño Asistido por Computadora, Robótica, Sistemas de Información Geográfica y Bases de Datos Multidimensionales y Espaciales.*

*Este capítulo presenta las estrategias de geometría computacional que se encuentran más estrechamente relacionadas al dominio de SIG.*

## 4.1. Clasificación de Polígonos

En este trabajo se hará referencia a polígonos simples, disjuntos y con agujeros, por lo que en esta sección demarcaremos brevemente el alcance de cada definición.

Un polígono simple consiste en una secuencia de segmentos, los vértices de cada uno de ellos coinciden exactamente con un vértice de otro segmento formando así una cadena cerrada, además, en un polígono simple, los segmentos no se superponen entre sí más que en dos puntos de contacto (los vértices) y en cada uno de estos puntos sólo lo hacen con un segmento.

Un polígono simple puede tener zonas que se encuentren contenidas dentro del área delimitada por su perímetro pero que sin embargo no forman parte de él, a estas regiones las llamaremos agujeros y estarán definidas por otros polígonos incluídos en el principal. El área interna de estos polígonos será la que se excluye del polígono principal.

Un polígono disjunto es un conjunto de polígonos simples con o sin agujeros que se toman como una unidad. Esta definición no es tan útil para la geometría convencional como para el área computacional, ya que a menudo debemos modelar aspectos de la realidad que se involucran una disjunción, pero que sin embargo deben ser tratados como una misma unidad por los algoritmos involucrados; el ejemplo típico es el de un archipiélago.

En la Figura 4.1 se muestran ejemplos de cada uno de estos casos.

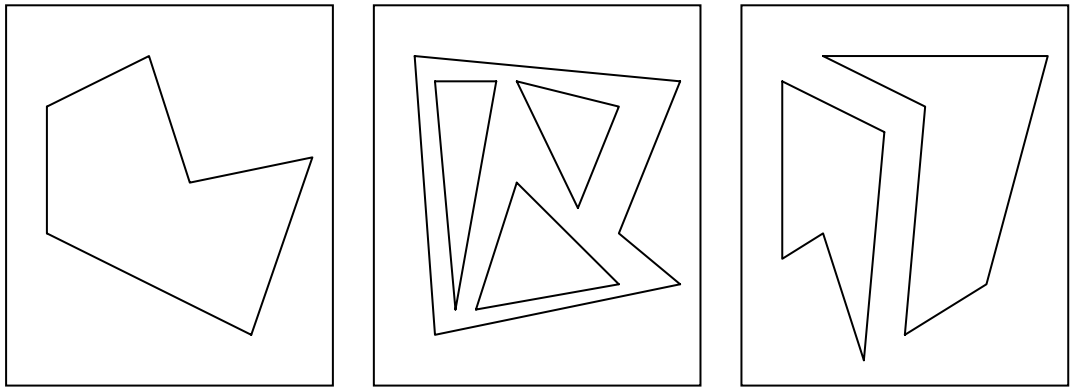



**Figura 4.1**.- (a) Polígono Simple, (b) Polígono Simple con tres agujeros, y (c) Polígono Disjunto.

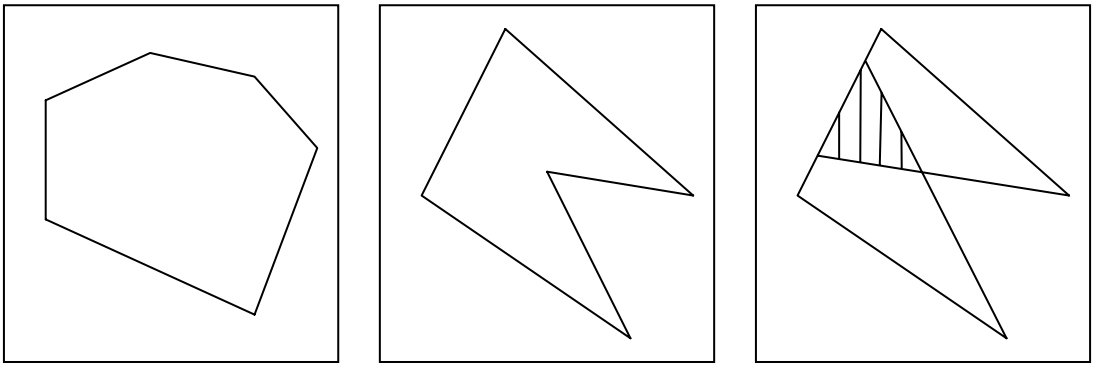
Los polígonos pueden clasificarse también sobre la base de otras características de la distribución de su área (de su extensión espacial), dando lugar a las categorías de polígonos convexos y polígonos estrella.

Un polígono es convexo si para cualquier par de puntos contenidos en el mismo, el segmento que los une también se encuentra íntegramente contenido en él.

Un polígono es estrella cuando existe al menos un punto p contenido en él tal que, para cualquier otro punto q también contenido en el mismo, el segmento que une a p con q también se encuentra contenido completamente dentro del polígono. Un polígono estrella puede tener un conjunto de puntos que cumplan esta condición, a ese conjunto (área) se la llama el kernel del polígono.

Con base en estas definiciones puede verse que un polígono simple puede ser o no convexo (el de la figura 4.1.a no lo es), independientemente de que tenga agujeros o no. También se deduce que tanto los polígonos con agujeros y los disjuntos nunca son convexos ni estrella. Se desprende también que un polígono convexo es siempre estrella, y que su kernel coincide con él mismo.

En la figura 4.2 se muestran ejemplos de polígonos según estas definiciones.



a b c

**Figura 4.2**.- (a) Polígono Convexo, (b) Polígono Estrella, y (c) kernel del polígono estrella

## 4.2. Operaciones y Testeos

Los SIGs hacen bastante uso de algunas operaciones y testeos geométricos, para lo cual suele recurrirse estrategias y algoritmos de geometría computacional. Es esta sección se estudian los algoritmos más utilizados en el campo de los SIGs.

### 4.2.1. Operaciones entre polígonos

Existe un conjunto de operaciones binarias básicas entre polígonos, entre las que consideraremos las de intersección, unión, diferencia y diferencia simétrica. Estas operaciones pueden ser realizadas con polígonos de cualquier tipo: simples (independientemente de su calidad de convexos o estrella), disjuntos o con agujeros. La diferencia sí podrá radicar en el algoritmo utilizado en cada uno de estos casos, ya que o bien no todos los algoritmos pueden utilizarse para todos ellos, o bien pueden conseguirse algunas ganancias al explotar alguna característica conocida de antemano.

- Algoritmo Switching Trip: Tratándose de polígonos convexos, el algoritmo más obvio para el cálculo su unión o intersección es el switching trip. Consiste simplemente en calcular los puntos de intersección entre los dos polígonos, y considerándolos como nuevos vértices de los mismos (que serán vértices compartidos por ambos), realizar un recorrido de los polígonos cambiando de polígono en cada punto de intersección. Esto es: partir de uno de estos puntos y recorrer un polígono almacenando sus vértices hasta que se encuentre un vértice común, a partir de allí, se cambia de polígono de manera de continuar el recorrido sobre el otro, también almacenando sus vértices hasta llegar a otro vértice común, para

cambiar nuevamente al polígono original y continuar el recorrido repitiendo este proceso. El conjunto de vértices almacenados (en ese orden) determinará la unión o intersección de ambos. El hecho de obtener un resultado u otro depende de la elección del polígono a recorrer en el inicio.

En la Figura 4.3 puede verse que partiendo desde p en el sentido indicado, si comenzamos recorriendo P1 obtenemos la intersección de ambos, pero si en cambio lo hacemos por P2 obtenemos la unión. La idea es que, al momento de elegir seguir el camino entre dos segmentos, para la unión siempre debemos elegir los segmentos exteriores, y en la intersección, los interiores. La descripción formal de estas elecciones puede encontrarse en la bibliografía referenciada.

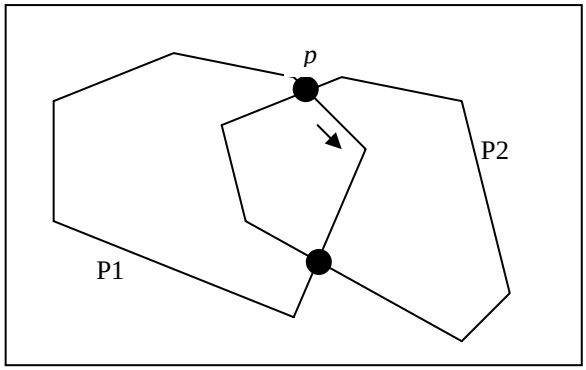



**Figura 4.3**.- Intersección mediante algoritmo Switching Trip.

La desventaja de este algoritmo es su orden de ejecución O(LM), siendo L y M la cantidad de lados de cada polígono, lo cual es inaceptable para L y M grandes. Esto es así debido a que en la fase de detección de puntos de intersección, cada lado de un polígono debe, en principio, chequearse contra todos los lados del otro polígono. Sin embargo, podría conseguirse alguna mejora explotando la convexidad de los polígonos conocida de antemano, por ejemplo, sabemos que sus lados no pueden intersectarse más de dos veces.

Cabe destacar que la intersección de dos polígonos simples convexos también da como resultado un polígono simple. Esto, como puede observarse en la figura 4.4, no ocurre para polígonos generales, en donde el resultado puede estar compuesto por varios polígonos o, según las definiciones precedentes, un polígono disjunto.

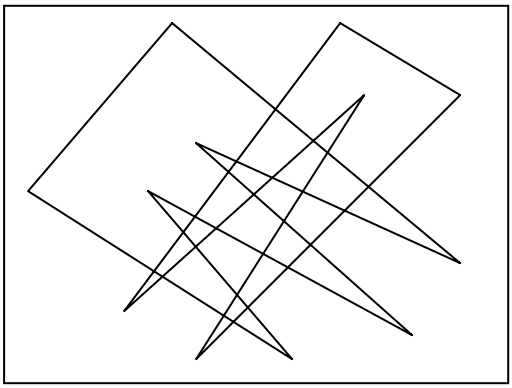


**Figura 4.4**.- Intersección de polígonos generales. El resultado puede ser un conjunto de polígonos simples (o uno disjunto)

Existen métodos que además de la unión e intersección, permiten calcular la diferencia y la diferencia simétrica para cualquier tipo de polígonos con mejores tiempos de ejecución.

- Algoritmo de Franjas o Slabs: Este algoritmo puede utilizarse con cualquier tipo de polígonos y permite el cálculo de su intersección, unión, diferencia y diferencia simétrica. Es un algoritmo que claramente se clasifica dentro de la metodología Divide and Conquer ya que se basa en la idea de tomar el problema de operar con dos polígonos de forma compleja y general, y dividirlo en n problemas más chicos, reduciendo los operandos a formas más simples y conocidas. El resultado final se obtiene combinando estos resultados parciales.

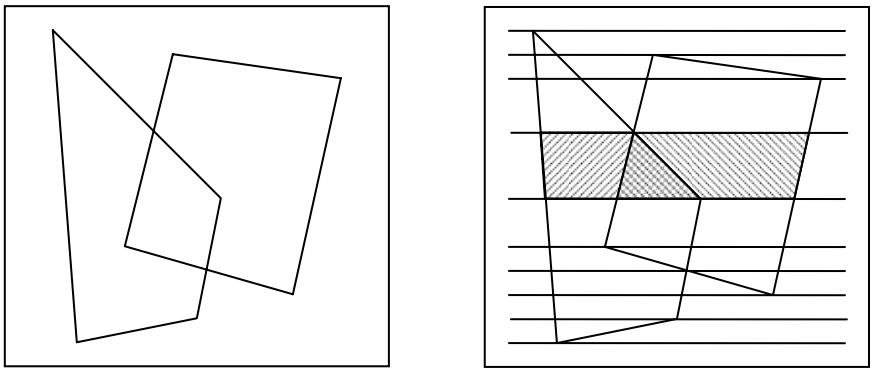
  El algoritmo también tiene una primera fase en la que deben encontrarse todos los puntos de intersección entre los polígonos (para lo que puede utilizarse la técnica de Barrido de Espacio). Luego se calculan las rectas que pasan por cada uno de estos puntos y por los vértices de cada polígono. De esta manera se obtienen franjas (Slabs) que, considerando algunos segmentos de estas rectas como lados, dividen a cada polígono en trapecios, que bien pueden degenerarse en triángulos, cuadrados o rectángulos.

  La segunda fase consiste en realizar la operación (intersección, unión o alguna diferencia) dentro de cada slab entre los trapecios que ésta define. De esta forma se ha reducido el problema de operar con dos grandes polígonos de forma desconocida a operar con n problemas más chicos y operandos conocidos (trapecios), siendo mucho más simple la operación con trapecios al poder ser clasificada en unos pocos casos conocidos.

  La tercera fase consiste en la unión de los resultados parciales de manera de obtener el polígono (o polígonos) con la forma del resultado esperado.

  Suele ser conveniente una cuarta fase de "purificación" en la cual se eliminan vértices innecesarios, es decir, aquellos cuyos dos vecinos forman ya una línea recta.

  En la figura 4.5 puede observarse la división en franjas para dos polígonos:



a b

**Figura 4.5**.- (a) Dos Polígonos y (b) su división en franjas y trapecios determinados en una de ellas.

### 4.2.2. Testeos Básicos

En esta sección se describirán algunos métodos para testear si un punto está contenido en un segmento, si un punto está contenido en un polígono y si dos segmentos se intersectan.

Antes se describirán algunas primitivas básicas que define la Geometría Computacional como soporte para la realización de estos testeos [Lopez].

- Primitivas Básicas

  **Producto Cruzado**:

  El producto cruzado de p1 = (x1,y1) y p2 = (x2,y2) está dado por:

$$\det \begin{vmatrix} i & j & k \\ x1 & y1 & 0 \\ x2 & y2 & 0 \end{vmatrix} = (x1y2 - x2y1)\, k$$

$$p1 \times p2 = \det \begin{vmatrix} x1 & y1 \\ x2 & y2 \end{vmatrix} = x1y2 - x2y1 = -p2 \times p1$$

Propiedades del Producto Cruzado:

Cross (p1,p2):

es > 0 si p1 se ubica en sentido horario con respecto a p2
es < 0 si p2 se ubica en sentido horario con respecto a p1
es = 0 si p1 y p2 son colineales.

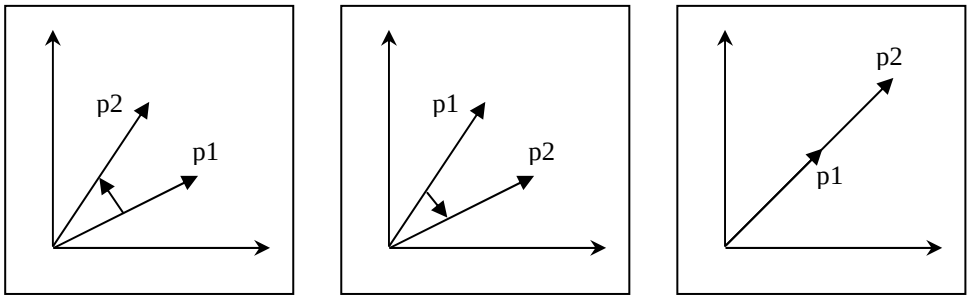



**Figura 4.6**.- (a) cross(p1,p2) > 0, (b) cross(p1,p2) < 0, y (c) cross(p1,p2) = 0

**Primitiva Turn**

Turn(p,q,r):
Dobla a Izquierda si cross(q – p, r - p) > 0
Dobla a Derecha si cross(q – p, r – p ) < 0
Son colineales si cross(q – p, r – p) = 0

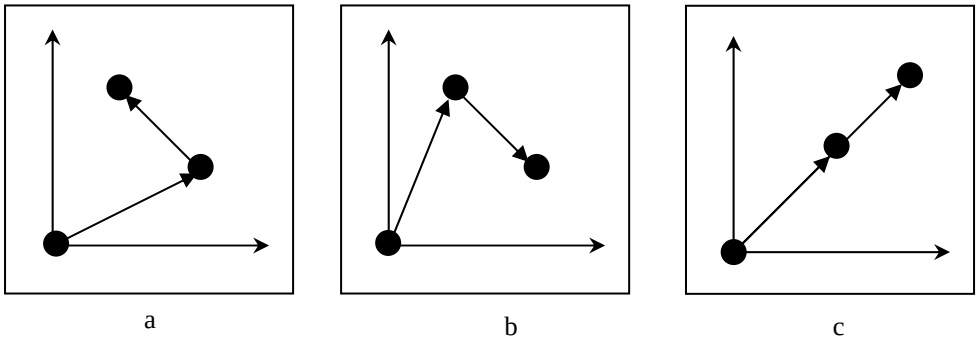



**Figura 4.7**.- Tres puntos y su circuito tal que (a) dobla a izquierda, (b) dobla a derecha y (c) no dobla.

**Bounding Boxes**

Como se ha descripto anteriormente, el bounding box de un objeto es el rectángulo más chico que lo contiene. Para el caso particular de un segmento p1p2 es el rectángulo [p1*,p2*] cuyas esquinas opuestas son p1* y p2* donde x1* = min (x1,x2), x2* = max(x1,x2), y1* = min(y1,y2), y2* = max(y1,y2).

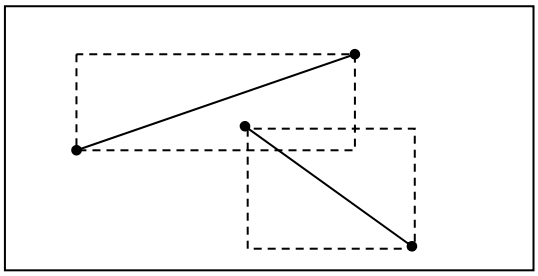


**Figura 4.8**.- Bounding boxes segmentos

Dos bounding boxes [p1*,p2*] y [p3*, p4*] se intersectan si y solo si es verdadero lo siguiente:

(x2 >= x3) ^ (x4 >= x1) ^ (y2 >= y3) ^ (y4 >= y1).

- Testeo de Punto en Segmento

El testeo de si un punto se encuentra contenido en un segmento se realiza mediante bounding boxes y la primitiva turn. Un punto q está contenido en un segmento p1p2 si está contenido en su bounding box y además p1, p2 y q son colineales; esto último se cumple si turn(p1, q, p2) = 0.

- Testeo de Intersección de Segmentos

Los métodos de la geometría tradicional para la representación de rectas y la operación con ellas (como por ejemplo el testeo de intersección entre dos de ellas) presenta ciertas dificultades si se intenta trasladarla directamente a una implementación computacional. En concreto, con respecto a la representación, la ecuación de la recta no permite definir rectas verticales, por lo que debería tratarse continuamente este caso aislado de los cómputos normales que se utilizan para el resto de los casos. En cuanto a la operación de intersección, la resolución del sistema de ecuaciones tradicional puede acarrear problemas con respecto a la precisión numérica que la computadora es capaz de manejar: por ejemplo, si las rectas son casi paralelas, el punto de intersección se hallaría muy distante, con lo cual los números en cuestión podrían ser inmanejables computacionalmente.

En consecuencia, la geometría computacional propone una alternativa a la solución tradicional del testeo de intersección. Ésta se basa en las primitivas anteriormente descriptas y se adapta más fácilmente a nuestros requerimientos. La misma se da a continuación, entendiendo por recta de soporte de un segmento a la recta sobre la cual éste se encuentra enteramente contenido.

Los segmentos p1p2 y p3p4 se intersectan si y solo si:

1) bbox(p1,p2) se intersecta con bbox(p3,p4).
2) el segmento p1p2 se intersecta con la recta de soporte de (p3p4).
3) el segmento p3p4 se intersecta con la recta de soporte de (p1p2).

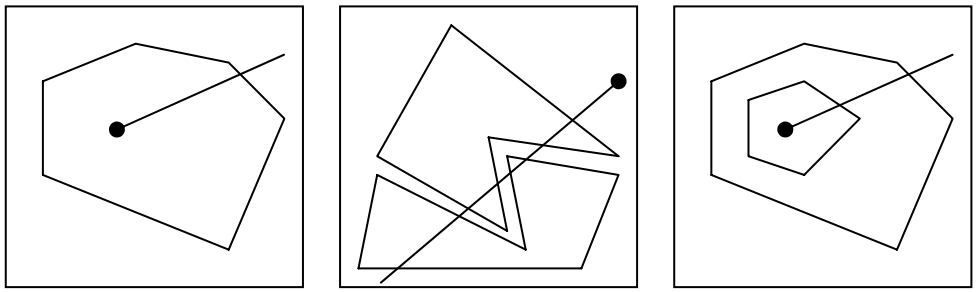
p3p4 straddles support(p1p2) si y sólo si:

cross(p3 – p1, p2 – p1) * cross(p4 – p1, p2 – p1) <= 0

- Testeo de Punto en Polígono

El testeo de inclusión de un punto q en un polígono p es en principio muy simple y directo: se traza una semirrecta con comienzo en el punto q hacia cualquier dirección, si esta semirrecta corta al polígono un número impar de veces, el punto se encuentra incluído en el polígono, de lo contrario se encuentra fuera del polígono. Este método tiene la ventaja de que sirve para todo tipo de polígonos: simples (tanto convexos y estrellas como generales), disjuntos y con agujeros.

En la figura 4.9 se muestran algunos ejemplos de este método.

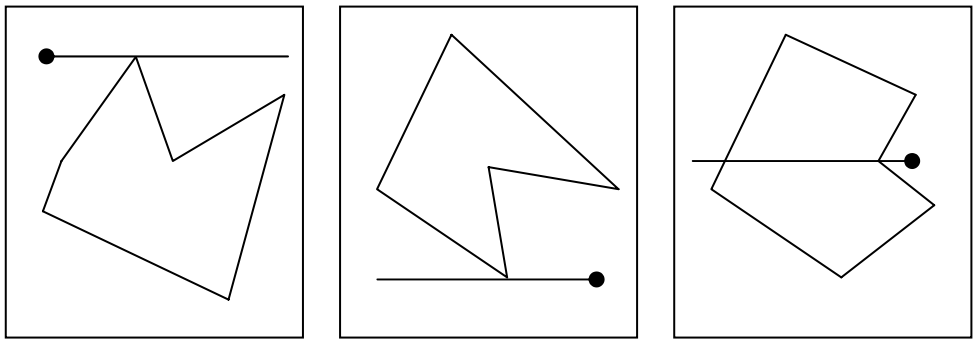


a b c

**Figura 4.9**.- (a) Testeo de inclusión de un punto en: (a) un polígono simple, (b) un polígono disjunto, y (c) un polígono con agujeros.

Esta es la solución para los casos normales, sin embargo existen casos particulares, generalmente llamados “casos degenerados” en los que hay que tomar algunas precauciones y consideraciones.

Uno de estos casos se presenta cuando la semirrecta corta al polígono justo en un vértice, pues como el punto que conforma un vértice se comparte entre dos lados del polígono, podrían considerarse erróneamente dos cruces, y dada la paridad podríamos obtener un resultado erróneo. La solución para este caso problema es muy simple: para cada segmento cuyo vértice es intersectado por la semirrecta, se cuenta el cruce sólo si ese vértice tiene coordenada x más grande que la del vértice del otro extremo del mismo segmento.



a b c

**Figura 4.10**.- Casos particulares de intersección de la semirrecta con vértices

Puede verse en las figura 4.10 a y b que si bien en rigor existe un solo punto de intersección entre la recta y el polígono, en el primer caso no se cuenta ningún cruce y en el

segundo caso se cuentan dos. Si bien queda claro que es errónea la cantidad de cruces computados, lo importante es que este método mantiene la paridad, que es lo que determina la validez del resultado final.

# PARTE II: DESARROLLO DEL MODELO

## 5. CAPÍTULO V: DESARROLLO DEL MODELO DE DATOS GEOGRÁFICOS

*En este capítulo se presenta el modelo orientado a objetos para cada uno de los conceptos que fueron introducidos en el Capítulo II.*

*Se profundiza aún más en el estudio de los Sistemas de Referencias y sus elementos constituyentes, como consecuencia de tal análisis se presenta un modelo OO con suficiente generalidad para adaptarse a diversas situaciones concretas. Lo mismo se realiza con un Modelo de Datos Geográficos de tipo Vector: el Topológico.*

### 5.1. Datos geográficos

Para modelar los datos geográficos tenemos que respetar las cuatro características que poseen y que fueron descriptas en el capítulo II. Es decir, que tienen que tener una posición, atributos no espaciales, mantener las relaciones espaciales, y el tiempo en el que se toman esos datos.

El mayor problema es que tenemos distintos tipos de información: espacial y no espacial. La información espacial se refiere a datos geográficos, como ser la posición, y al comportamiento geográfico de las entidades del mundo real. La información no espacial se refiere a datos no geográficos, como ser su nombre, y al comportamiento de dichas entidades en el dominio de la aplicación.

La solución utilizada para representar a las entidades que poseen una ubicación geográfica permite mantener desacoplados los conceptos anteriores, y plantea el uso del patrón Decorator [Gamma]. La solución fue planteada por [Gordillo] y [Balaguer], y en base a ésta complementaremos la información de un objeto geográfico.

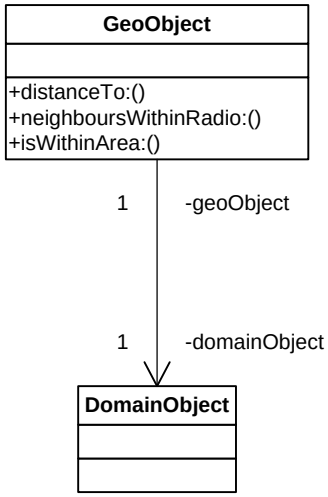



**Figura 5.1**.- Relación entre objetos del dominio y objetos geográficos.

De la solución planteada anteriormente vemos que el objeto encargado de mantener la información geográfica, llamado geoObject, tiene demasiadas responsabilidades que debería delegar en otras entidades. Dentro de las responsabilidades que le son propias se encuentran: la manera de representar la posición y la manera de representar la forma geográfica del objeto geográfico. La primera abarca los conceptos de posición, sistema de referencia y sistema de coordenadas, y es desarrollada a continuación. La segunda abarca la representación del Modelo de Datos Geográficos, tema ya desarrollado en una sección anterior.

### 5.1.1. Posición

La manera de representar la posición de un objeto geográfico es desacoplando esa información y su comportamiento del geoObjeto. Para esto se define una clase Location, la cual será la responsable de administrar todo lo relacionado a la posición del geoObjeto. De esto se deduce que debe conocer el sistema de referencia en el cual fue tomada la posición y la forma geométrica del objeto geográfico.

La posición de un objeto geográfico en un momento dado quedará representada por la clase Location, como dijimos anteriormente. En principio, hasta que veamos como representar la topología de un objeto geográfico asumiremos que la posición de éste es representada por una o más coordenadas, las cuales se interpretan en base a un sistema de referencias. Debido a que existen diferentes representaciones para una posición, desacoplamos la posición de la representación de la siguiente forma:

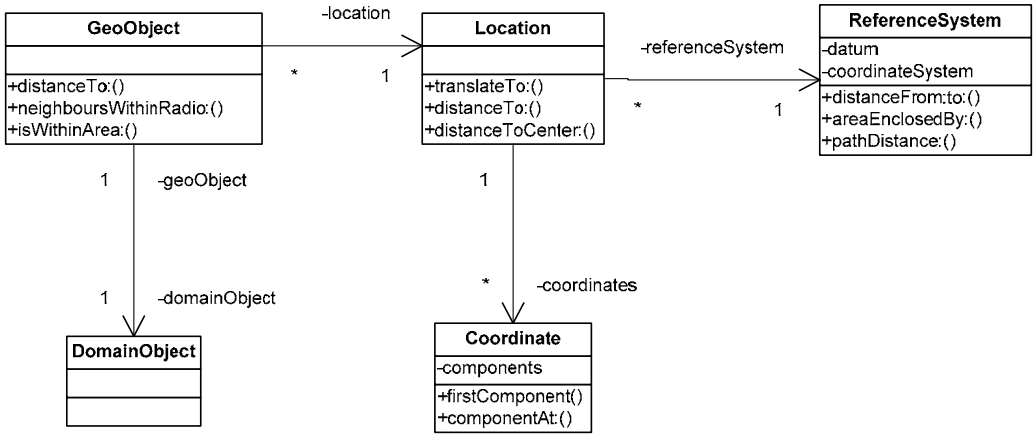



**Figura 5.2**.- Diseño para la posición de un objeto geográfico.

De esta forma, con la solución planteada en la figura anterior se pueden soportar cambios de representación dinámicos sin necesidad de modificar la posición del objeto geográfico.

### 5.1.2. Sistema de Referencias

Para lograr un buen diseño debemos tener en cuenta como está formado el sistema de referencias y cómo opera. Esto, en parte, ya se explicó en el capítulo II. En este capítulo se explican con más detalle aspectos relacionados con los sistemas de referencias y se presenta un diseño que utiliza como base lo presentado por [Lliteras].

Para georeferenciar una entidad geográfica necesitamos un sistema de referencias que contemple el uso de cuatro componentes:

1. Superficie de referencia de la Tierra: es decir, su representación. Como se explicó en el Capítulo II, la figura de la Tierra es difícil de representar debido a que es muy irregular y cambia constantemente (esto último es real, aunque sólo relevante para aplicaciones muy específicas, por ejemplo, algunas relacionadas con la astronomía). Es por este motivo que se utilizan aproximaciones teniendo en cuenta el nivel del mar y su prolongación debajo de los continentes. Una aproximación es la que utiliza la figura de un geoide, pero las anomalías de la tierra pueden hacer que no sea una aproximación precisa en todas partes. Los geodesistas utilizan otra figura geométrica, que es más simple de manipular, llamada elipsoide.

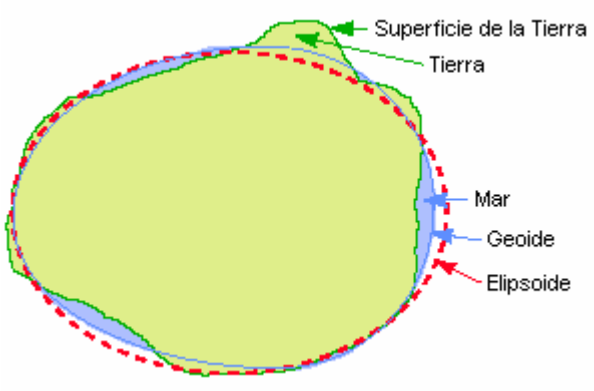



**Figura 5.3**.- La Tierra y sus aproximaciones.

Como se muestra en la figura 5.3, el geoide aproxima en base al promedio de la superficie de los océanos, el elipsoide lo hace sobre la superficie de la tierra entera, es decir superficie y océanos. Se sabe que la mejor aproximación de la tierra con una figura matemáticamente fácil de manipular se obtiene con un elipsoide.

Aproximar al geoide a través de un elipsoide implica definir su tamaño (dimensión de sus semiejes mayor y menor), su forma y excentricidad. Entonces, siendo:

- a: dimensión del semieje mayor
- b: dimensión del semieje menor

La fórmula que define la forma, la nombramos f, y se compone como se muestra a continuación:

$$\frac{(a - b)}{a}$$

La excentricidad se define con la siguiente fórmula, en base a la fórmula f anterior:

$$(2f - f^2)^{1/2}$$

Algunos elipsoides son:

| Nombre | Semieje mayor | 1/ excentricidad |
|---|---|---|
| WGS84 | 6.378.137 | 298,257223563 |
| South American 1969 | 6.378.160 | 298,25 |
| Internacional 1924 | 6.378.388 | 297 |

2. Datum: define la orientación y el desplazamiento desde el lugar donde está posicionada la representación de la Tierra respecto de los ejes de coordenadas iniciales. Significa que además de definir la representación de la Tierra, se debe tener en cuenta un sistema de ejes de coordenadas inicial en donde se centrará el sistema de referencias. Los sistemas de referencias actuales son centrados en el baricentro terrestre, el plano XY es el plano ecuatorial y el eje de las Z apunta al polo Norte [Perdomo]. El término datum es el que se utiliza para agrupar los conceptos explicados anteriormente (orientación y desplazamiento). En la siguiente figura se muestra un eje de coordenadas inicial y un posible desplazamiento para un datum:

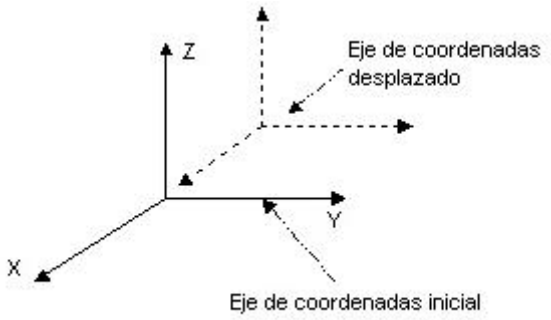



**Figura 5.4**.- Ejes de coordenadas inicial y un posible desplazamiento.

Existen diferentes datums, y cada país debe utilizar el que más se aproxime a su superficie. La siguiente figura muestra ejemplos de diferentes datums:

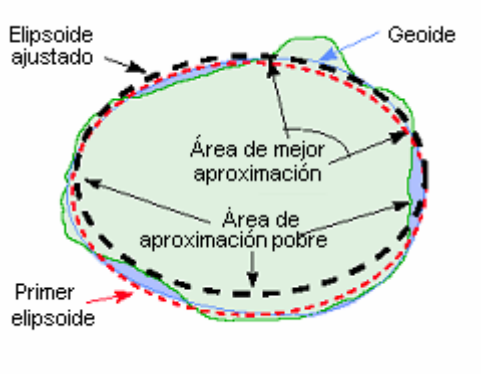



**Figura 5.5**.- Uso de diferentes datums, según la aproximación que se necesite.

Existen cientos de datums, como ejemplo, el Global Positioning System (GPS) está basado en el World Geodetic System 1984.

3. Puntos referenciados: son las ubicaciones de los objetos geográficos.

4. Sistema de coordenadas: como también mencionamos en el Capítulo II, el sistema de coordenadas es el que brinda el marco lógico y matemático para describir la posición de la entidad geográfica. Utiliza coordenadas y una forma de medición para definir la posición relativa de un punto en relación a un origen. Los más comúnmente usados son:
   - Elipsoidal o geodésico: definido usando los conceptos de latitud, longitud y altura.
   - Geográfico: derivado del anterior, se setea la altura en cero.
   - Earth Centered Earth Fixed (ECEF XYZ)
   - Universal Transverse Mercator (UTM)

   Cada sistema de coordenadas tiene sus características. Por ejemplo, el geodésico es utilizado para navegación y el UTM cuando renecesita una representación planar de la superficie de la Tierra.

   Debemos hacer notar que estamos hablando de dos tipos de sistemas de coordenadas. Por un lado existen los proyectados y por el otro los no proyectados. Los primeros utilizan coordenadas planares (como ser el UTM) y necesitan saber cuál es la forma en que se proyectaron (es decir, cómo se mapeó cada punto del modelo en tres dimensiones a un modelo en dos dimensiones: el mapa). Los segundos utilizan tres coordenadas (por ejemplo el Geodésico o el ECEF XYZ), permiten representar posiciones en el espacio.

Un sistema de coordenadas no está ligado a un datum específico. De la misma forma que ellos pueden intercambiarse para obtener mejores aproximaciones según se necesite.

Todos los conceptos de los cuatro componentes de un sistema de referencias detallados anteriormente son necesarios para conocer como funciona el georeferenciamineto de los objetos. La figura 5.6 muestra un punto P, con los componentes necesarios para su georeferenciamiento. Este punto se indica con latitud, longitud y altura, podemos ver que utiliza el sistema de coordenadas geodésico, junto con sus ejes y el elipsoide que mapea la superficie terrestre:

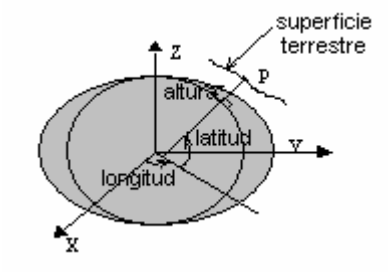



**Figura 5.6**.- Georeferenciamiento del punto P.

Conociendo todas las particularidades explicadas anteriormente, podemos deducir que el diseño del sistema de referencias planteado en la Figura 5.2 no es bueno ya que la clase ReferenceSystem maneja demasiada información. Esta información se puede desacoplar para brindar más flexibilidad al usuario a la hora de configurarlo.

Se utilizará el patrón Mediator [Gamma], el cual permite definir un objeto que encapsula la forma en que un conjunto de objetos interactúa. De esta forma, definimos al sistema de referencias como el mediador de sus componentes: el datum y el sistema de coordenadas. La siguiente figura muestra esta solución:

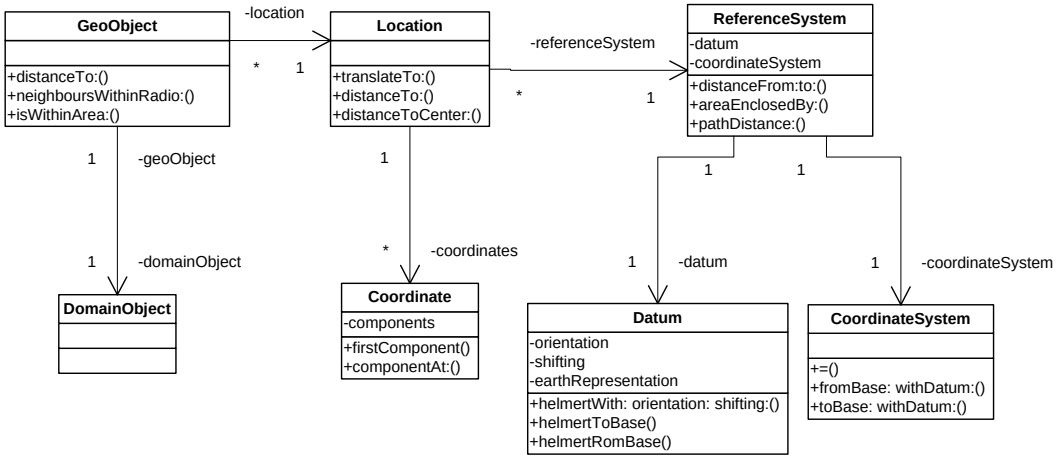



**Figura 5.7**.- Utilización del patrón de diseño Mediator, para desacoplar información del sistema de referencias.

Con este diseño, el sistema de referencias delega las responsabilidades en sus componentes, como ser el cálculo de operaciones matemáticas. También se pueden realizar operaciones en tiempo de ejecución para posiciones con distintos sistemas de referencias.

Como mencionamos, existen distintos tipos de sistemas de coordenadas: los proyectados y no proyectados. La diferencia entre ellos radica en si se necesitan representar las posiciones sobre el plano o en el espacio. Para que la representación pueda hacerse en uno o en otro de manera indistinta, podemos subclasificarlos en dos subclases. Así, los no proyectados referencian una posición en un espacio de tres dimensiones (el geodésico (latitud, longitud y altura), ecefxyz (para coordenadas x,y,z)), y los proyectados representan una posición a través de coordenadas planares (x,y) y la estrategia de proyección de coordenadas de tres dimensiones a dos dimensiones. Las estrategias de proyección más utilizadas son Mercator, Universal Transverse Mercator (UTM), etc.

Las estrategias de proyección proveen una correspondencia entre un punto sobre la tierra y su punto en el plano de la proyección [Snyder]. En el diseño se utiliza el patrón de diseño Strategy [Gamma] para modelar las diferentes estrategias de proyección. Este patrón define una familia de algoritmos, las cuales son encapsuladas y pueden ser intercambiadas. Los algoritmos pueden variar en forma independiente del cliente que los usa, brindando una gran flexibilidad al mismo.

De esta forma, la arquitectura final del sistema de referencias queda como se puede visualizar en la siguiente figura:

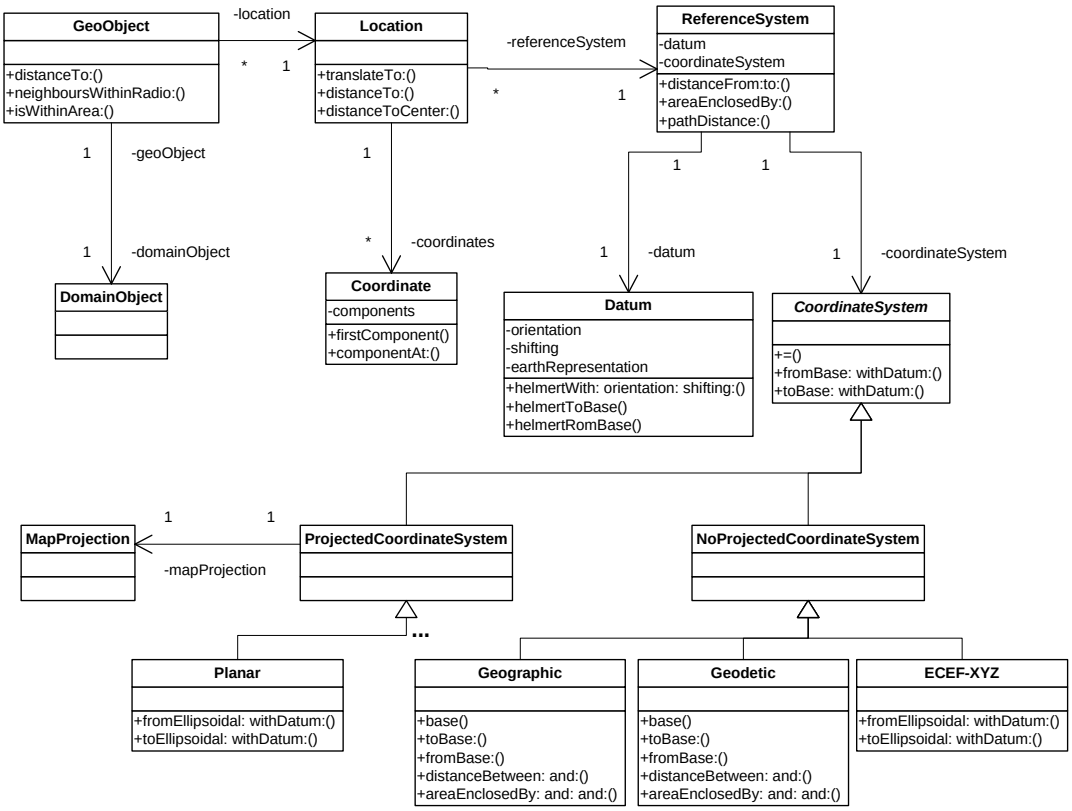



**Figura 5.8**.- Modelo OO final del objeto geográfico y su posición.

A continuación describimos los componentes más importantes involucrados en la arquitectura anterior, a modo de resumen de las características que poseen las clases detalladas en el diseño:

- ReferenceSystem: es el encargado de dar sentido geográfico a las coordenadas. Se relaciona con el datum y el sistema de coordenadas, a los cuales delega responsabilidades para poder llevar a cabo sus operaciones.
- Datum: define la orientación de los ejes y su desplazamiento respecto de un sistema de coordenadas y utiliza la representación de la Tierra. La representación de la Tierra podría separarse en otra clase para darle más generalidad al modelo, ya que podrían utilizarse otras figuras que no fueran elipsoides. Con lo planteado hasta el momento tendríamos una subclase llamada Ellipsoid. Al desacoplarlo del sistema de referencias, el datum puede ser cambiado en tiempo de ejecución.

- CoordinateSystem: el encargado de llevar a cabo las operaciones de translación que se necesiten para dejar la información consistente. Al igual que el datum, puede ser cambiado en tiempo de ejecución. Se subclasifica en proyectados o no, para que cada subclase se encargue de manejar la información que tienen en común. Por ejemplo, a los proyectados se les asocia la estrategia de proyección, situación que no se da con los sistemas de coordenadas no proyectados.

Conocer la distancia entre dos objetos es una operación básica que debe poder realizarse en cualquier SIG. En la figura 5.9 se describe un diagrama de interacción con objetos instanciados de acuerdo a la arquitectura anterior, de forma de ver la manera en que ellos interactúan cuando se quiere conocer la distancia que hay entre ellos. Dicho diagrama sigue los siguientes pasos para el cálculo:

1. Desde una aplicación se dispara el mensaje que le dice al objeto **geoObject1** que calcule la distancia entre él y el objeto **geoObject2**.
2. El objeto **geoObject1** toma la posición del **geoObject2**, la cual es representada por el objeto **location2**.
3. El objeto **geoObject1** toma su posición, representada por el objeto **location1**.
4. El objeto **geoObject1** le dice a su posición que calcule la distancia a la posición del objeto **geoObject2**.
5. El objeto **location1** toma su sistema de referencias, representado por el objeto **referenceSystem1**.
6. El objeto **location1** le dice a su sistema de referencias que calcule la distancia que existe entre él y la posición representada por el objeto **location2**.
7. El objeto **location2** cambia su sistema de referencias al mismo sistema de referencias que el objeto **location1**, para que sean compatibles
8. El sistema de coordenadas del objeto **location1**, denominado **coordinateSystem1**, realiza los cálculos de distancia entre las dos posiciones.

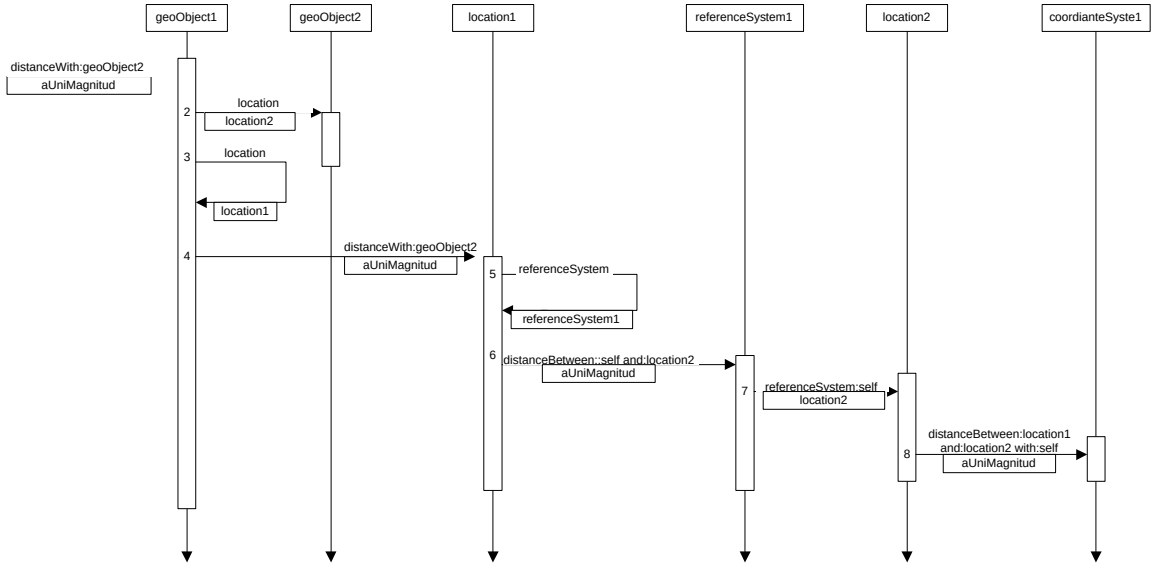



**Figura 5.9**.- Diagrama de interacción para el cálculo de distancia entre dos geoObjetos.

## 5.2. Modelo de datos geográficos

Esta sección describe el diseño orientado a objetos del modelo de datos geográficos expuesto conceptualmente en el Capítulo II. En particular, nos hemos enfocado principalmente en el modelo de vector, ya que es más rico en capacidad de representación y operación.

### 5.2.1. Topología

En la sección anterior vimos que los objetos geográficos conocen su posición como un conjunto de coordenadas interpretadas en un sistema de referencias. El modelo de datos de los objetos geográficos expuesto en el Capítulo II es más complejo de mantener que sólo un conjunto de coordenadas. Pudimos observar que hay distintos modelos para representarlos: modelo de raster y modelo de vector.

En nuestro trabajo decidimos realizar un diseño de objetos para el modelo de vector, en particular el topológico, ya que nos provee mayores ventajas al momento de realizar las operaciones geométricas.

Este diseño consta de una jerarquía cuya clase abstracta se denomina Geometry, la cual permite especificar los aspectos relacionados a la forma de cada objeto en términos de puntos, líneas y polígonos, y también los aspectos referentes a las relaciones espaciales entre ellos, con lo cual se gana perfomance al momento de realizar cálculos, como testeos de inclusión, conectividad e intersección.

La siguiente figura muestra un primer diseño orientado a objetos para este modelo topológico. La clase Geometry reemplaza a la clase Coordinate, que figura en la arquitectura final mostrada en la Figura 5.8.

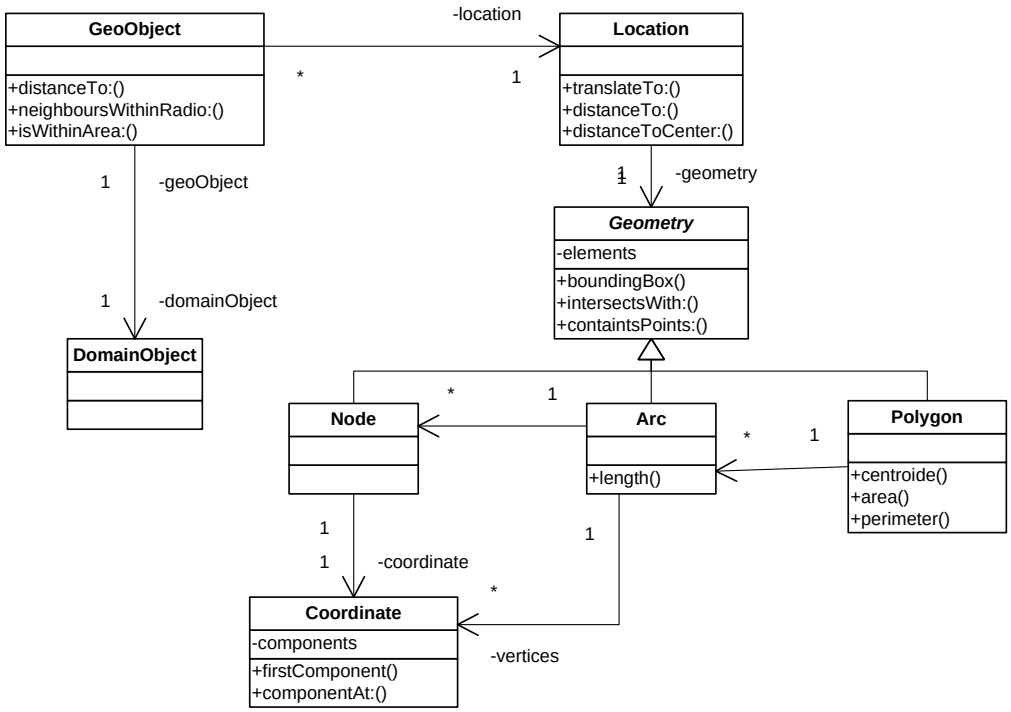



**Figura 5.10**.- Primer diseño del modelo topológico.

La geometría o forma de una entidad geográfica puede ser instanciada con base en las clases Node, Arc o Polygon. De forma análoga a lo explicado en el Capítulo II:

- Node: se puede utilizar para representar una coordenada puntual, la intersección entre dos o más arcos, o el comienzo o fin de éstos. El valor de sus coordenadas se toma desde una instancia de la clase Coordinate, la cual sigue representando una coordenada sobre la tierra, que será interpretada en el sistema de referencia asociado a la Location del GeoObject.
- Arc: está representado por dos nodos, de comienzo y de fin, y un conjunto de coordenadas, llamada vértices, también instancias de la clase Coordinate.
- Polygon: representado por un conjunto de arcos cerrados, que forman el borde del área.

En la siguiente figura se presenta un ejemplo concreto de cómo se instancia la topología de ciertos objetos geográficos, denominados A, B, C y D:

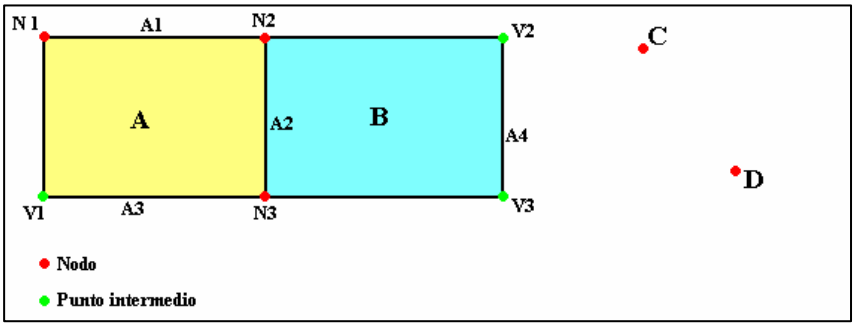



**Figura 5.11**.- Ejemplo de la topología de los objetos geográficos A, B, C y D.

En la figura anterior puede notarse que la entidad A es representada por el polígono A, compuesto por los arcos A1, A2, y A3. De la misma manera, A1 está delimitado por los nodos N1 y N2, A2 por N2 y N3, y A3 por N3 y N1 respectivamente. Además de los nodos de comienzo y fin, A3 tiene también un punto intermedio V1 (representado por una instancia de la clase Coordinate).

Un aspecto importante de esta arquitectura es la capacidad de compartir objetos. Notar que, en la figura anterior, el nodo N2 es compartido por los arcos A1, A2 y A4. De la misma manera, el arco A2 es compartido por los polígonos A y B.

En los sistemas de información geográfica existen fundamentalmente dos tipos de operaciones:

- ***Operaciones geométricas***: aquellas que necesitan puntos georeferenciados para poder realizarse, como ser la distancia entre dos entidades.
- ***Operaciones topológicas***: aquellas que sólo requieren información sobre relaciones espaciales entre entidades, como ser: conectividad o adyacencia entre entidades.

Como un ejemplo del primer tipo de operación, la distancia entre las entidades C y D en la figura anterior es calculada utilizando sus coordenadas interpretadas en los sistemas de referencia correspondientes. Por otro lado, la adyacencia entre las entidades A y B es determinada por la existencia de un arco común a los polígonos que representan cada entidad. Notar que cada polígono está compuesto por un conjunto de arcos y, en este caso, tanto el polígono A como el B tienen al arco A2 como parte de su composición. Como cada uno de estos arcos “conoce” a los objetos compuestos a los que pertenece, esta operación puede resolverse testeando, para cada arco en el polígono A, si también se encuentra en el polígono B.

En conclusión, con este modelo es posible manipular de forma flexible operaciones geométricas, como ser distancia entre dos entidades, y operaciones topológicas, las cuales requieren información sobre relaciones espaciales, como ser adyacencia entre entidades. Las primeras operaciones se pueden computar con las instancias de la clase Coordinate. Las segundas se pueden determinar por la existencia de instancias en común, por ejemplo arcos, entre entidades.

Para terminar de cerrar este modelo topológico debemos resolver la manera de representar entidades geográficas que no son tan fáciles de de ver como un punto, un arco o un polígono. Entre ellas se encuentran los polígonos disjuntos o con agujeros, ya mencionados en el capítulo IV, según se pueden ver en la siguiente figura:

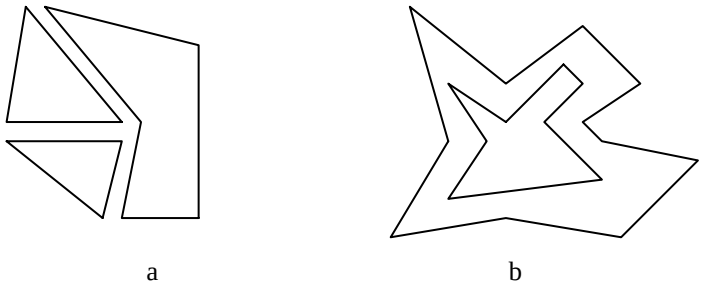



**Figura 5.12**.- (a) Polígono disjunto. (b) Polígono con agujero.

Los objetos geográficos, que posean las características de la figura anterior, deben tratarse como uno sola entidad. Para esto necesitamos componer polígonos, ya que una entidad geográfica está asociada a un solo elemento geométrico. Como ejemplo, podemos mencionar las Islas Malvinas, las cuales deberían ser representadas por dos polígonos disjuntos y referenciar a la misma entidad.

Una primera solución sería aquella que permite asociar más de un elemento topológico a un geoObjeto. Esta solución tiene la desventaja de que se pierde el concepto de composición y el geoObjeto sería el responsable de manejar los aspectos relacionados con su geometría, lo cual conduce a una sobrecarga de responsabilidad en dicha clase. Algunos ejemplos de esta situación pueden ser el cálculo de área de un polígono disjunto o con agujeros. En el primer caso, seguramente el geoObjeto debe hacer la sumatoria de las áreas de los polígonos componentes, mientras que en el segundo debe restar las áreas de los agujeros del área del polígono principal.

Una segunda solución, finalmente adoptada por nuestra arquitectura, permite manejar elementos compuestos como una unidad capaz de resolver los cálculos por si misma. Para esto se utiliza el patrón de diseño Composite [Gamma], el cual nos da la ventaja de evitar dicha sobrecarga, ya que nos permite contar con la capacidad de manejar un elemento compuesto como una unidad capaz de resolver los cálculos por sí misma. Permite tratar de manera uniforme a elementos compuestos y simples. De esta manera, seguimos teniendo asociado un solo elemento topológico al geoObjeto.

En la siguiente figura se visualiza la arquitectura final para la topología de un geoObjeto:

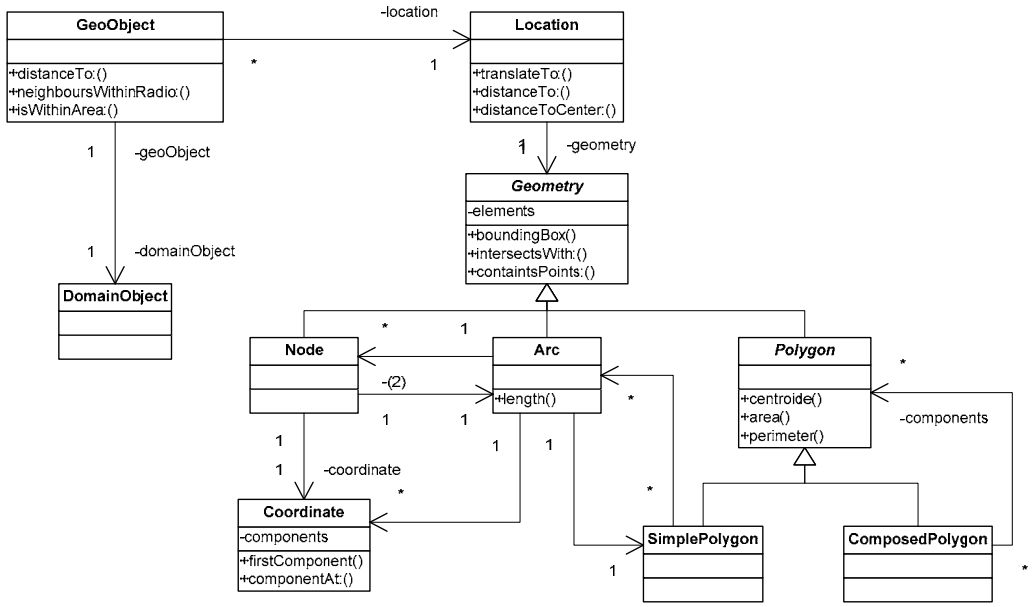



**Figura 5.13**.- Modelo OO final del objeto geográfico y su forma.

La clase Polygon es ahora una clase abstracta, cuyas subclases pueden ser SimplePolygon o ComposedPolygon.

Eventualmente, de la misma forma que se utilizó el patrón de diseño Composite para los polígonos, se podría utilizar el mismo patrón para modelar arcos compuestos.

# 6. CAPÍTULO VI: DESARROLLO DEL MODELO DE INDIZACIÓN

*En este capítulo se muestra el diseño orientado a objetos de algunas de las estructuras descriptas en el Capítulo III. Particularmente se discuten aspectos de diseño de las estructuras de Quad-Tree y R-Tree, por ser algunas de las más representativas de la indización puntual y de áreas, respectivamente. Se presta especial atención a la transparencia en el uso de tales estructuras por parte del cliente, intentando que éste pueda cambiarla por otra de manera flexible; para esto se propone una generalización de las mismas bajo un protocolo común.*

*Por último, se muestra una aplicación construida por los autores exclusivamente para la prueba de la estructura de R-Tree, también implementada para este trabajo. Mediante tal aplicación se intenta mostrar mediante ejemplos prácticos los aspectos teóricos discutidos oportunamente al respecto.*

## 6.1. Modelo de indización

En esta sección se describe el modelo orientado a objetos de las estructuras Quad-Tree y R-Tree ya expuestas anteriormente. Se definen con un gran nivel de detalle las estructuras y la operatoria de las mismas, y luego se busca generalizar el uso por parte del cliente de manera que el mismo pueda optar por diferentes estructuras sin grandes cambios ni impactos en su aplicación.

### 6.1.1. Quad-Tree

En esta subsección se describirá el modelo de objetos para el Quad-Tree desarrollado en el capítulo III. Sobre la base de lo ya expuesto, tanto su estructura como su funcionalidad se han definido en forma recursiva. En esta estructura todo nodo contiene información sobre objetos indexados (a diferencia del R-Tree, en donde la información sólo se encuentra en las hojas). Por lo tanto, cada nodo mantiene una referencia al objeto que indexa; además, cada nodo mantiene hasta cuatro referencias a nodos hijos, que se encuentran ubicados espacialmente en cada uno de los cuatro cuadrantes que la posición de éste define. En la siguiente figura puede observarse el diagrama de objetos para este modelo. Los nodos se representan mediante instancias de la clase *Node*; la clase *QuadTree* define el protocolo de operación (agregar, borrar, buscar objetos) que los clientes deben utilizar para operar con la estructura; éstos no operan directamente con los nodos, lo que permite una abstracción de aspectos de implementación.

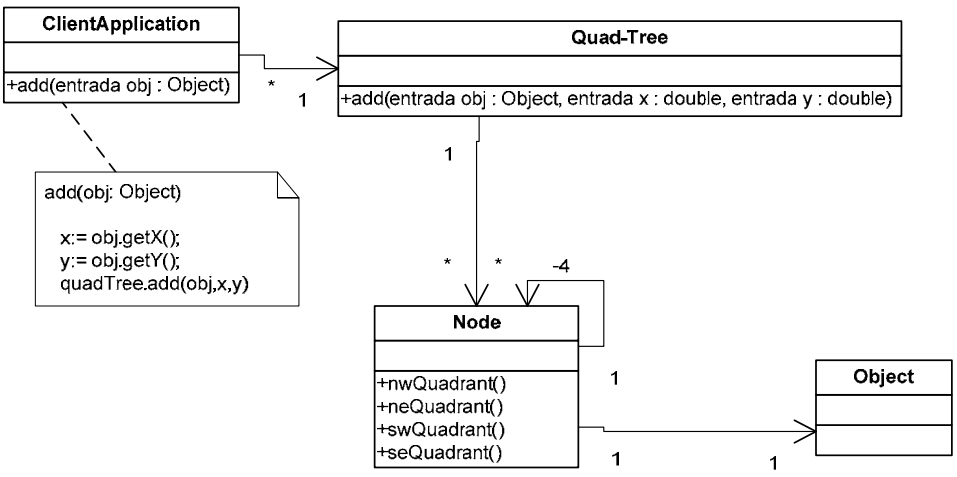



**Figura 6.1**.- Modelo Inicial del Quad-Tree.

Con este modelo, una aplicación cliente podría indexar los objetos que maneja a través de la interfaz que provee las instancias de la clase Quad-Tree.

Como puede observarse, las instancias de la clase Quad-Tree entienden un mensaje *add* con tres parámetros: el objeto que se desea agregar a la estructura de índice, su coordenada *x*, y su coordenada *y*. Esto implica que la aplicación cliente primero deberá obtener las coordenadas del objeto a agregar y recién luego invocar el mensaje *add* de la estructura.

Esta metodología presenta la desventaja de que cada aplicación cliente debe implementar este mismo conjunto de acciones previas; para nuestro ejemplo, todos los clientes deberán implementar en uno o varios lugares un código idéntico al asociado al mensaje *add* mostrado en la Figura 6.1. En este caso las acciones a implementar no son muchas puesto que sólo se trata de obtener las coordenadas. No obstante, debe notarse que la *distribución de responsabilidades* entre los objetos no es la correcta, ya que el cliente se está encargando de detalles particulares concernientes a la estrategia de indización; y esto es responsabilidad exclusiva de la estructura de indización.

Es sabido que una mala distribución de responsabilidades atenta contra los objetivos de flexibilidad de extensión y modificación del diseño Orientado a Objetos [Rebbeca]. En nuestro caso podemos proponer un simple ejemplo para ilustrar este punto: si deseáramos cambiar la metodología de indización mediante Quad-Tree por una mediante R-Tree, las acciones previas a agregar un objeto a esta nueva estructura consistirían en obtener el bounding-box del objeto, y ya no sus coordenadas *x* e *y*. Claramente podemos ver que un cambio en la metodología de indización demandaría cambios en las aplicaciones cliente, lo cual sería incorrecto puesto que en realidad éstas deberían centrarse en los aspectos del dominio y no deberían verse afectadas por cambios en aspectos de estructuras de indización.

Teniendo en mente estas consideraciones, se propone como solución que no sea la aplicación cliente quien obtenga la información previa a la inserción, sino la misma Estructura de Indización. Se entiende que ésta sí es una *buena distribución de responsabilidades* ya que esta información depende exclusivamente de la Estructura de Indización a utilizar, lo que hace que su obtención sea su exclusiva responsabilidad. En términos de implementación, esto deriva en que la clase Quad-Tree ofrece ahora un mensaje *add* con el objeto a agregar como único parámetro. El método asociado a este mensaje obtendrá las coordenadas *x* e *y* del objeto para luego insertarlo en la estructura (delegando en la clase Node). Para esto último, la clase Quad-Tree podría invocar ella misma el mensaje original de tres parámetros, pero ahora éste sería *privado* por tratarse de un mensaje auxiliar.

En la figura siguiente se muestra el diagrama de clases con la redefinición propuesta y un diagrama de interacción que describe los pasos involucrados en la inserción de un objeto en la estructura.

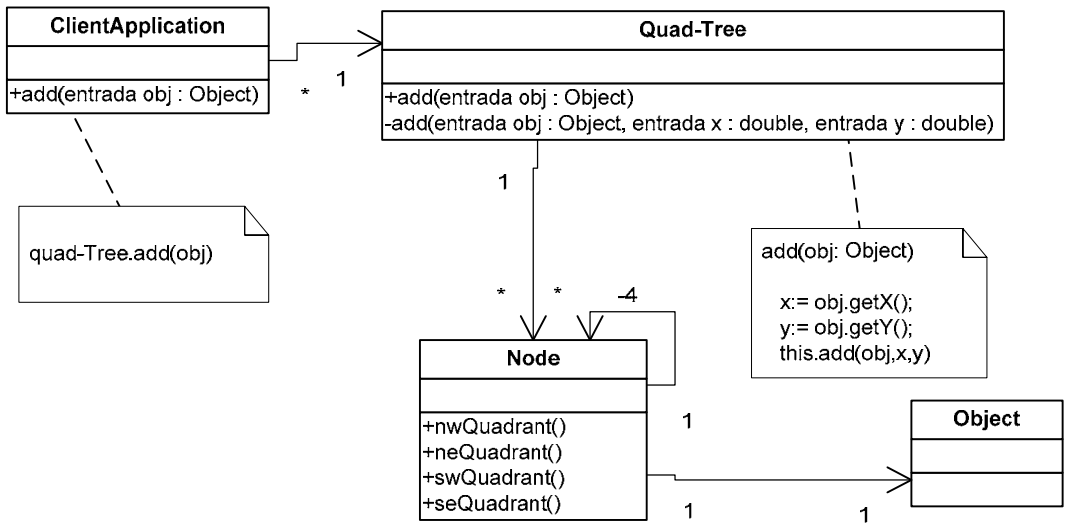



**Figura 6.2**.- Modelo Final del Quad-Tree

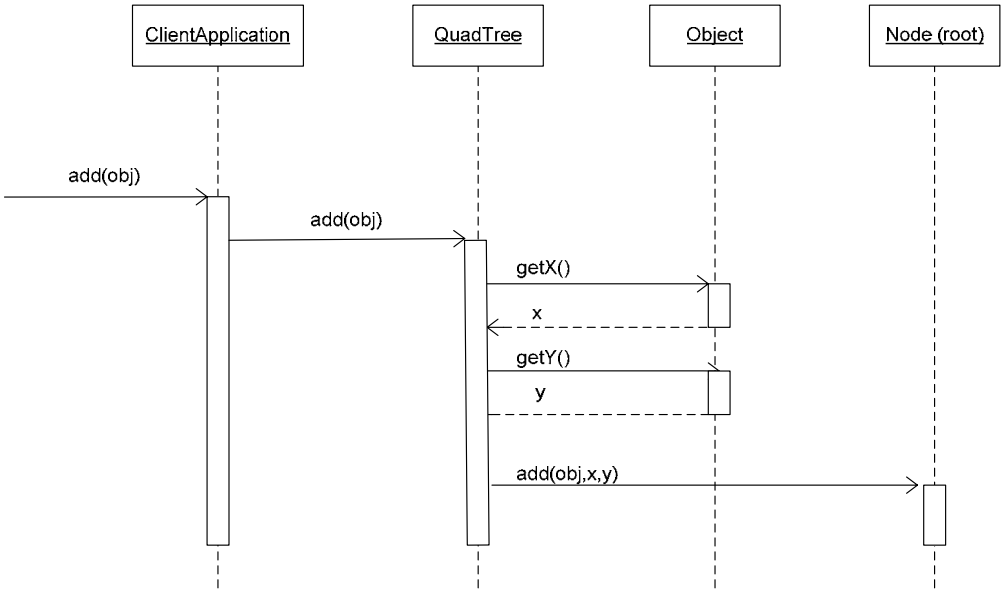



**Figura 6.3**.- Diagrama de Interacción para la Inserción de un Objeto.

Como puede observarse, la estructura envía a los objetos los mensajes *getX()* y *getY(),* esto determina que los objetos (al igual que en la versión original) deben cumplir un *contrato*, es decir, deben definir una *interfaz* que incluya estos dos mensajes y deben implementar un método para cada uno de ellos. Los objetos a almacenar pueden ser heterogéneos y pertenecer a distintas clases, pero siempre cada una debe definir esta interfaz; de esta manera se logra un modelo de alta flexibilidad gracias al *polimorfismo* del Diseño Orientado a Objetos.

### 6.1.2. R-Tree

En esta subsección se describirá el modelo de objetos para el R-Tree expuesto en el capítulo III. Como se ha visto, esta metodología de indización consta de una estructura recursiva, lo que deriva también en una algorítmica del mismo tipo. La estructura consta básicamente de nodos, los cuales contienen un conjunto de entradas, cada una de las cuales contiene, a su vez, un bounding-box y un puntero. Al parecer no habría diferencia entre nodos hoja y nodos internos (no-hoja), sin embargo, aunque en rigor la estructura es la misma para ambos, la semántica de las entradas es distinta: ambos definen bounding-boxes y referencias, pero los nodos internos lo hacen para sus nodos hijos, mientras que los nodos hoja lo hacen para los objetos indexados. Esta diferencia semántica estructural deviene en una diferencia algorítmica, ya que en muchos casos el trato de los nodos internos es distinto al de los nodos hoja, puesto que éstos constituyen el caso base de los algoritmos recursivos (a diferencia del Quad-Tree, aquí las referencias a datos reales se encuentran sólo en los nodos hoja). Con base en esta diferencia de comportamiento más que en la estructural, es que se ha decidido modelar estos dos tipos de nodos con dos clases diferentes, *LeafNode* e *InternalNode*, cada una contemplando los aspectos algorítmicos que le son propios. Tanto la estructura, que es la misma para ambos, y los algoritmos o parte de ellos que son comunes, se modelan en una clase abstracta: *RTreeNode*. De esta manera se obtiene un modelo flexible en cuanto a extensión y mantenimiento mediante la *herencia* del Paradigma Orientado a Objetos.

Se ha visto también que un R-Tree define una cantidad mínima y máxima, *m* y *M,* que determina los límites de entradas que deben estar llenas, fuera de ellos un nodo se encontrará en *underflow* u *overflow*. Estos valores son los mismos para todos los nodos, por lo tanto no tiene sentido definir variables para *m* y *M* en las clases de los nodos; por otro lado tampoco tiene sentido fijar sus valores en la estructura de los nodos y en los algoritmos, pues determinaríamos un *orden* del árbol para cualquier instanciación del mismo. Por estas razones, estos valores deben ser dinámicos (configurables por el usuario) y además deben definirse fuera de los nodos. Resulta claro que son un parámetro de la estructura

por lo que se los define como variables de instancia en la clase que la representa: *RTree.* De esta manera el tamaño del conjunto de entradas de los nodos es dinámico y su tamaño se ajustará siempre a los límites impuestos por las estructura (a su vez impuestos por el usuario).

Como puede observarse en la siguiente figura, la clase *RTree* conoce al nodo raíz de la estructura y determina la interfaz del cliente con la misma, ya que es la única clase que éste conoce y con la que interactúa; el resto de las clases le resulta transparente. De esta manera se ocultan aspectos de implementación permitiendo flexibilidad en futuras extensiones o modificaciones.

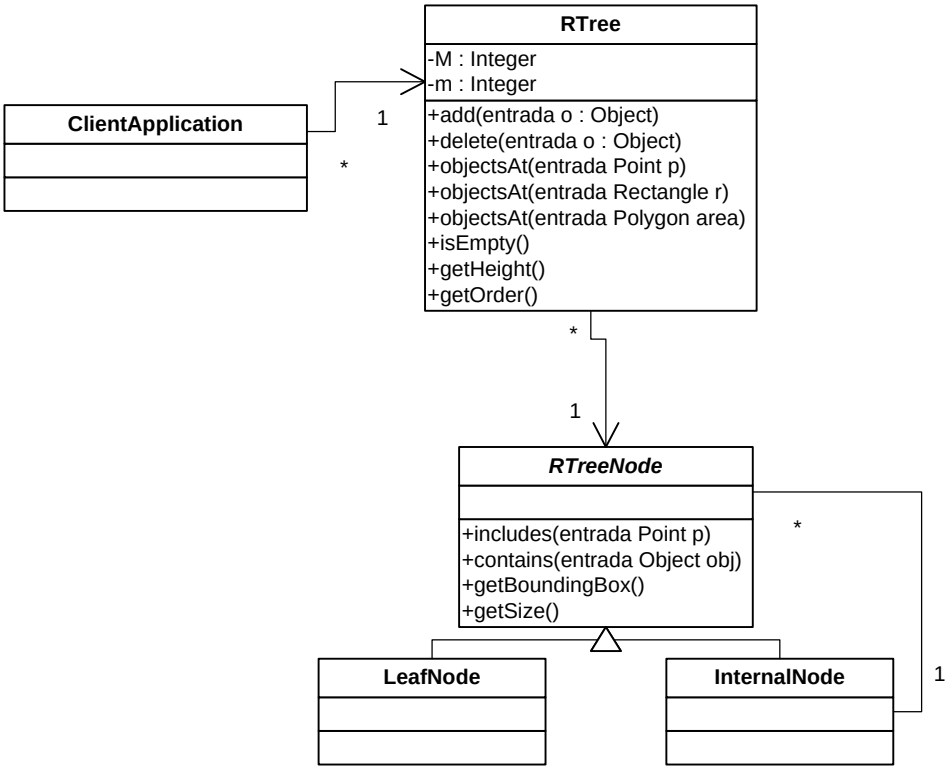



**Figura 6.4**.- Modelo básico del R-Tree.

Anteriormente se ha visto que existen distintos algoritmos de partición de nodos. Este modelo tiene implementado en alguna de sus clases alguno de estos algoritmos, lo que significa que al estar definido como un *método*, la metodología de partición será siempre la misma y no podrá variar para diferentes instanciaciones del árbol. Sin embargo, sería deseable que fuese el usuario quien elija la estrategia de partición de nodos acorde a sus criterios y al contexto en el que utilizará el RTree.

Para lograr este dinamismo en la elección de estrategias de partición puede utilizarse el patrón de diseño *Strategy* [Gamma]. En contraposición a la tendencia normal de implementar una estrategia en un método, este patrón sugiere modelar cada estrategia en una clase distinta y dedicada únicamente a tal fin. El *mensaje* que en cada clase dispare la ejecución del algoritmo que ésta implementa debe ser exactamente el mismo, aunque, obviamente, el *método* que cada una tenga asociada será distinto ya que constituye la esencia de cada estrategia en particular. De esta manera se logra una solución flexible y dinámica gracias el *polimorfismo* del Paradigma Orientado a Objetos.

Una *superclase abstracta* común a todas estas estrategias puede abstraer la definición de parámetros (en forma de constantes o variables de instancia), la estructura y el comportamiento común a todas sus subclases; además define la interfaz común que se deberá utilizar para comunicarse con cualquiera de las instancias que definen las estrategias concretas.

Entendemos que la Estrategia de Partición es un parámetro más de configuración del árbol, y además es la misma para todos los nodos que se deseen partir, por estas razones se ha decidido que sea el *RTree* quien la conozca, es decir, se ha definido como uno de sus atributos. Una instancia de la clase *RTree* deberá colaborar con los nodos en los momentos de partición de manera de brindarle soporte metodológico a partir de la Estrategia de Partición particular que se le haya definido.

En la figura siguiente se muestra la evolución del modelo anterior, ahora incluyendo las Estrategias de Partición de nodos como un atributo de configuración dinámico.

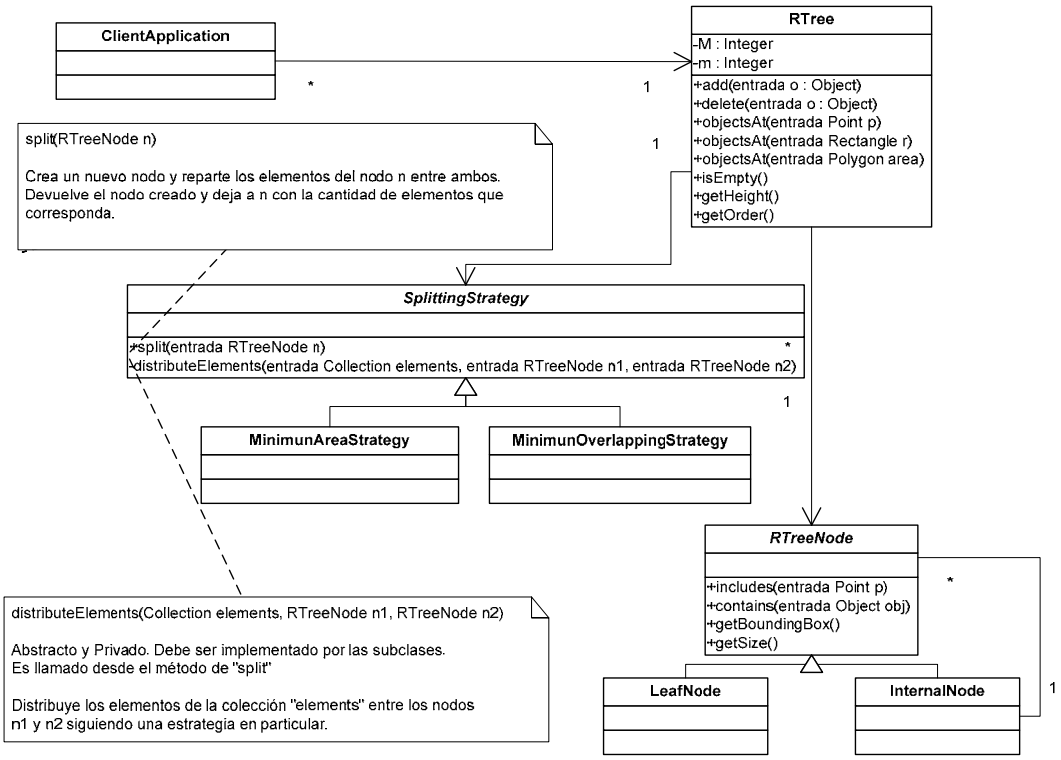



**Figura 6.5**.- Modelo del R-Tree con Estrategias de Partición.

Al igual que para el caso del Quad-Tree y por las mismas razones, es también aquí la Estructura de Indización (R-Tree) la encargada de las acciones previas a la inserción, que en este caso consisten en el cálculo y construcción del bounding-box del objeto a insertar.

En la siguiente figura se describe, mediante un Diagrama de Interacción, el proceso de búsqueda de los objetos que se encuentran en un punto ingresado por el usuario.

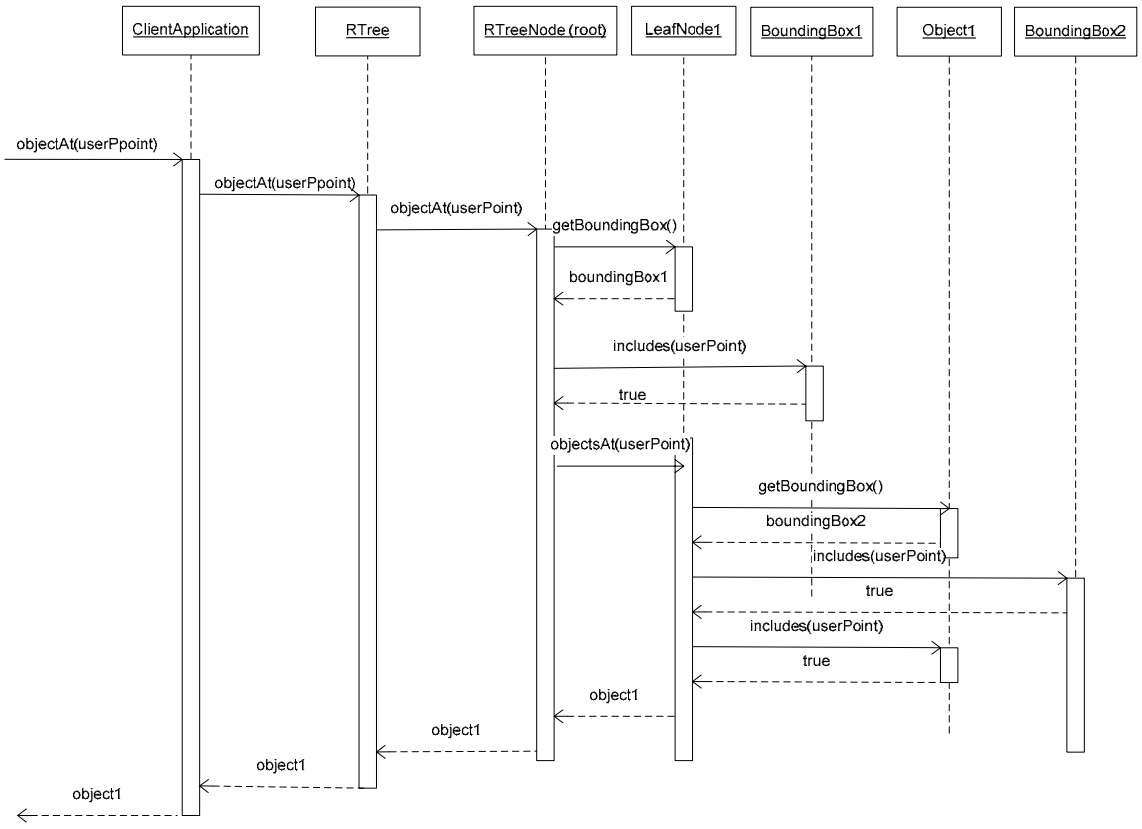



**Figura 6.6**.- Diagrama de interacción para la búsqueda en un RTree.

Con el modelo de clases descripto hasta aquí se tiene toda la funcionalidad necesaria para la operación normal del R-Tree. Sin embargo, se ha mencionado anteriormente que cuando la base de datos es *estática* o está muy próxima a serlo pueden utilizarse técnicas para una carga masiva inicial. Éstas permiten mejorar el empaquetamiento del árbol y como consecuencia la performance, además de reducir el espacio de almacenamiento de la estructura. Al árbol resultante de esta carga se lo conoce como *R-Tree Empaquetado*.

Para incorporar una de estas técnicas en el modelo descripto se podría agregar un mensaje a la clase RTree, este mensaje tomaría como parámetro la colección de objetos a cargar (o una referencia a ella) y como postcondición se obtendría el R-Tree Empaquetado. De la misma manera que como se describió para las Estrategias de Partición, programar un método de esta manera para una Estrategia de Carga Masiva determinaría estáticamente a la misma y el usuario no podría cambiarla por otra que considerara más apropiada para su caso particular. Otra alternativa es la de programar varios métodos en el R-Tree, uno para cada estrategia. Sin embargo, el hecho de intentar agregar o eliminar estrategias implicaría agregar o quitar métodos, con las consecuentes recompilaciones de la estructura ya existente y quizá también hasta instanciada. Además, y no menos importante, debido a la ya mencionada cuestión de *Distribución de Responsabilidades* [Rebbecca], la clase *R-Tree* estaría encargándose de demasiados detalles de aspectos de carga masiva que no le competen, lo que afectaría directamente a la flexibilidad del modelo. En consecuencia, sería ideal que ésta delegara (tercerizara) toda la funcionalidad relacionada a la carga masiva en otro objeto que se dedicara únicamente a tal tarea. La figura de este nuevo objeto corresponde a la de un *Loader* o *Cargador*.

Dado en esencia el mismo problema que para las Estrategias de Partición, se propone la misma solución que para ellas: la utilización del patrón de diseño *Strategy* [Gamma]. En este sentido se implementa cada estrategia en una clase concreta. Estas clases ofician de *cargadores* de estructuras y son subclases de una clase abstracta que define el comportamiento y estructura común a todas ellas. Además define la interfaz que también es común a todas las estrategias particulares, de esta manera el cliente (R-Tree) podrá comunicarse con cualquiera de ellas de una misma y única manera, lo que permite el

agregado y borrado de nuevas estrategias sin tener que tocar en absoluto a las clases que las utilizan (en este caso, el R-Tree).

En la siguiente figura se muestra el diagrama completo para el R-Tree incluyendo las estrategias de carga masiva. Éstas se encuentran en la jerarquía con raíz en la clase abstracta *BulkLoader*, y cuyas subclases son, en este caso, *HilbertLoader* y *SweeperLoader* que representan las dos estrategias que se han desarrollado en el capítulo III, pero pueden ser agregadas nuevas estrategias de carga de manera totalmente flexible con el único requisito de cumplir con la interfaz definida por la clase *BulkLoader*.

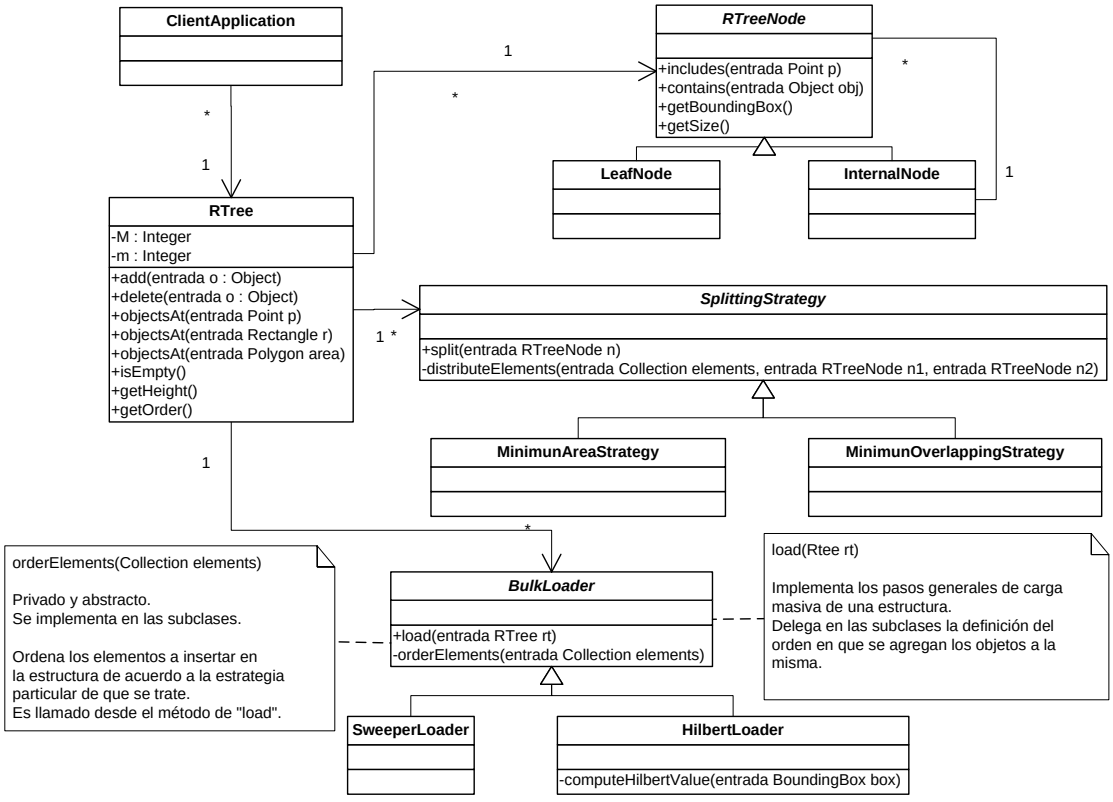



**Figura 6.7**.- Modelo final del RTree.

### 6.1.3. Generalización de las Metodologías de Indización

Como se ha visto en el capítulo III, un mismo juego de datos puede ser indexado mediante distintas estructuras. Por ejemplo, un conjunto de puntos puede ser indexado tanto por un R-Tree, como por un Quad-Tree o un K-D-Tree. En rigor, la aplicación cliente debería encargarse de resolver aspectos del dominio del problema, y no deberían afectarle los detalles referentes a temas de indización. En particular, el cambio o elección de una estructura de indización no debería demandar cambios en tal aplicación. Aún más: es el usuario final el que debería determinar cuál es la estructura de indización más adecuada al dominio, por lo que es necesario que ésta pueda ser elegida dinámicamente, en tiempo de ejecución. Esto, en consecuencia, determina que la aplicación cliente no debe tratar con detalles particulares de ninguna estructura; es decir, su forma de comunicación con ella debe ser lo suficientemente genérica de manera de que sea la misma para cualquiera de las demás estructuras, pues en tiempo de programación no se sabe cuál se utilizará. Además, y por lo mismo, en lenguajes tipados como Java, en la aplicación cliente no debería tiparse con la clase de ninguna estructura en particular a la variable de instancia que la referencia.

Estos aspectos demandan una *interfaz* común a todas las estructuras mediante la cual la aplicación cliente pueda comunicarse con cualquiera de ellas. Se ha decidido crear una superclase abstracta, *TreeIndex*, que defina esta interfaz y también el comportamiento común a las estructuras de

árboles particulares, sus subclases. En la figura siguiente se muestra esta jerarquía con dos clases concretas que representan las estructuras vistas hasta ahora. Las clases concretas que figuran en el diagrama son las que conoce la aplicación cliente y con las que interactúa; para mayor claridad no se muestran las clases por las que está compuesta cada estructura, por ejemplo las que representan a los nodos.

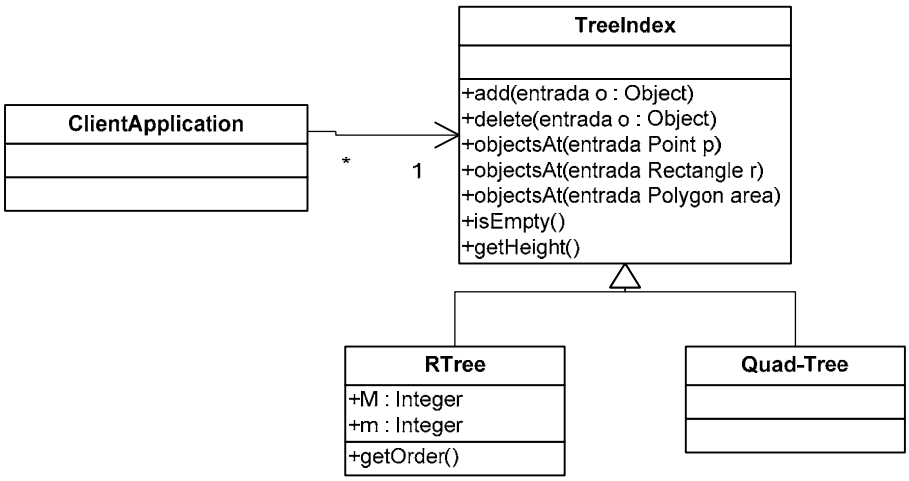



**Figura 6.8**.- Generalización a varias estructuras de árbol.

Siguiendo la misma línea de razonamiento, no hay razones para limitar al usuario a utilizar una estructura de árbol. Deberían poder utilizarse otras, como por ejemplo algunas estructuras lineales o estrategias de *Hashing*. Para lograr esto se ha creado una superclase *Index* que define el comportamiento y sobre todo la interfaz con la que el cliente puede comunicarse con cualquier estructura. Dada la diversidad de estructuras, quizá no exista mucho comportamiento común para abstraer en esta clase, por lo que en lenguajes tipados como Java podría crearse una interfaz (de las que pueden definirse en ese lenguaje) de manera de definir sólo un protocolo común sin necesidad de crear una clase que tenga poco o ningún comportamiento.

En la figura siguiente se muestra el diagrama de indización propuesto.

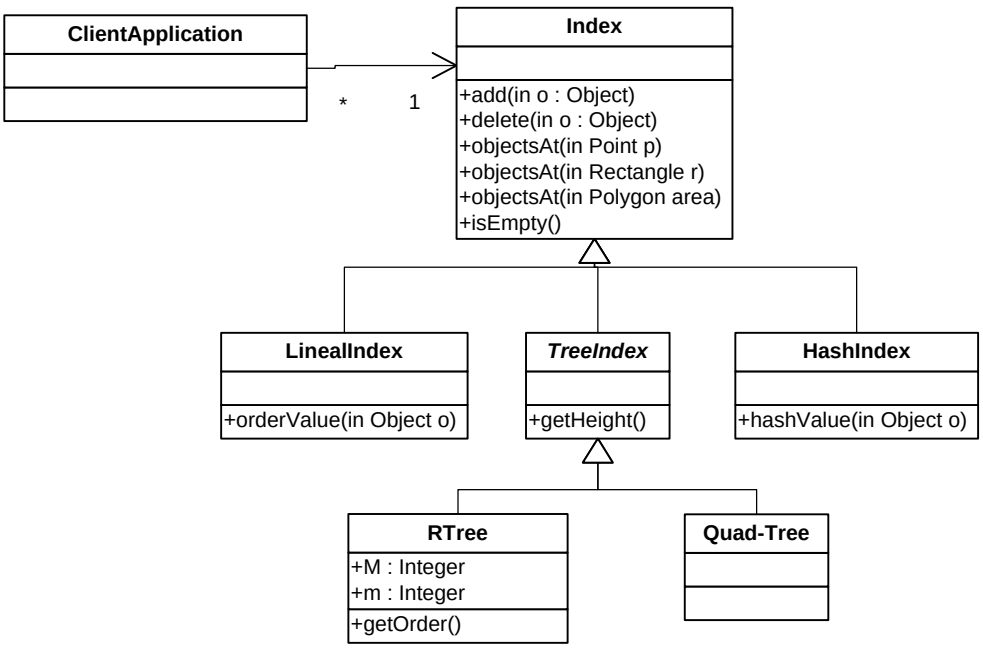


**Figura 6.9**.- Generalización a varias estructuras de índices.

Así como existen estrategias de carga masiva para R-Trees, también existen otras para otros tipos de estructuras. Por ejemplo, al utilizar una estrategia lineal podrían ubicarse al principio de la secuencia a los elementos que se sabe serán los más frecuentemente accedidos. De la misma manera también puede lograrse mayor empaquetamiento en una estructura de Hashing, que como bien sabemos, también ocupa espacio que no en todo momento se utiliza para el almacenamiento.

Dado que las estrategias de carga masiva no constituyen un aspecto exclusivo del R-Tree sino que son compartidas por todas las estructuras, se ha decidido definirlas como atributo de la clase *Index* (y ya no como atributo en la clase R-Tree) de manera de que todas las estructuras puedan utilizarla. Por supuesto, no cualquier estructura podrá utilizar cualquier estrategia de carga, pero esta restricción no se impone a nivel estructural en el modelo.

A continuación se muestra el diagrama final de indización:

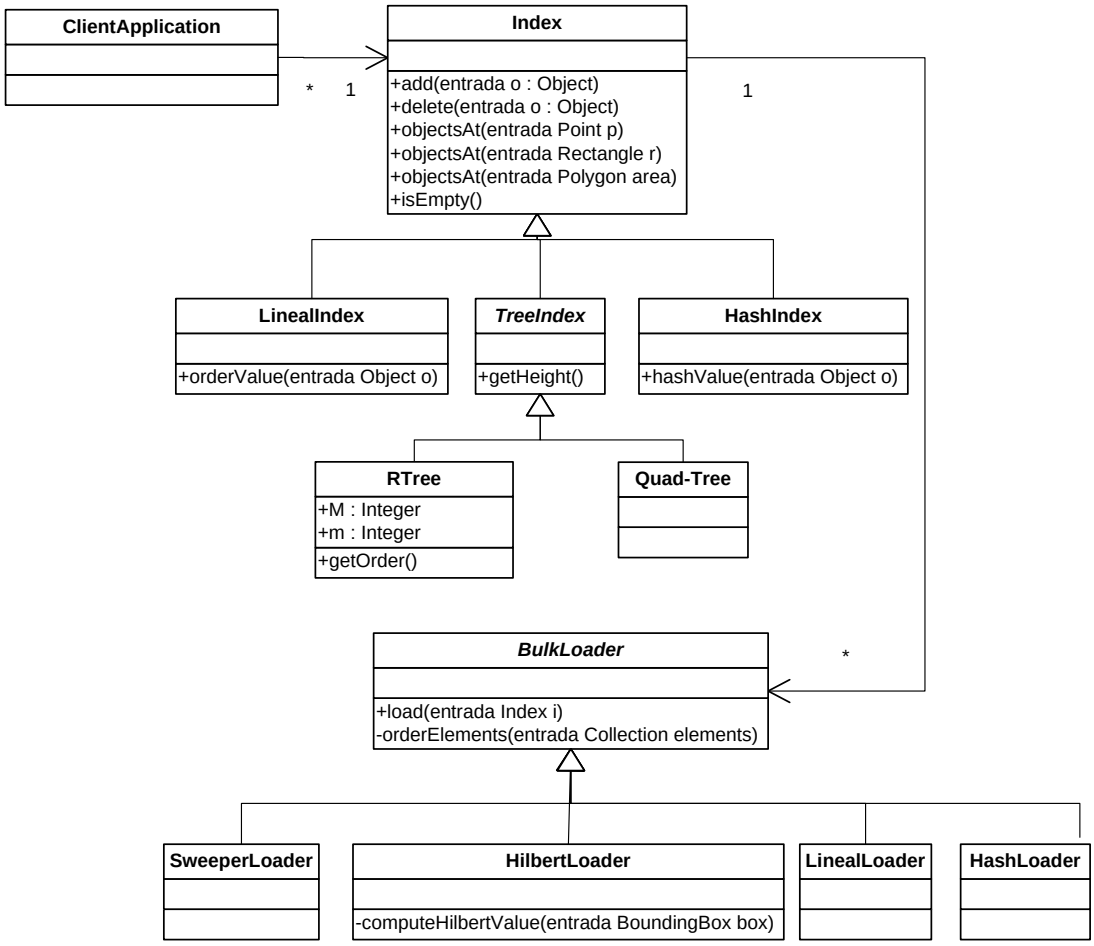



**Figura 6.10**.- Modelo final de Indización.

## 6.2. Implementación del RTree e Interfaz Gráfica de Prueba

En esta sección se describe brevemente una aplicación gráfica construída y utilizada como prueba de implementación de la estructura RTree ya descripta anteriormente. Esta aplicación ha sido desarrollada por los autores de este trabajo con el único fin de testear y mostrar la estructura implementada. Sin embargo, cabe destacar que la estructura se encuentra totalmente desacoplada de la interfaz gráfica, de esta manera se alcanza la suficiente flexibilidad para poder reutilizar la misma estructura en otras aplicaciones más complejas, como la descripta en el capítulo VIII. Esta flexibilidad se

logra a través de la utilización de interfaces (definición de protocolos), tal como se ha expuesto en este capítulo.

A continuación se describirá la pantalla principal y sus componentes, luego se mostrarán algunos ejemplos de uso de la misma.

### 6.2.1. Descripción de la Aplicación

La aplicación está compuesta por un panel que permite al usuario la inserción, borrado y búsqueda de distintos objetos geométricos. A medida que éstos son agregados o borrados del panel también son agregados o borrados automáticamente del RTree sobre el que se monta la aplicación. El panel permite observar en todo momento tanto a los objetos agregados como a la estructura de nodos que va adoptando el RTree a lo largo de su desarrollo.

Esta aplicación se muestra en la siguiente figura:

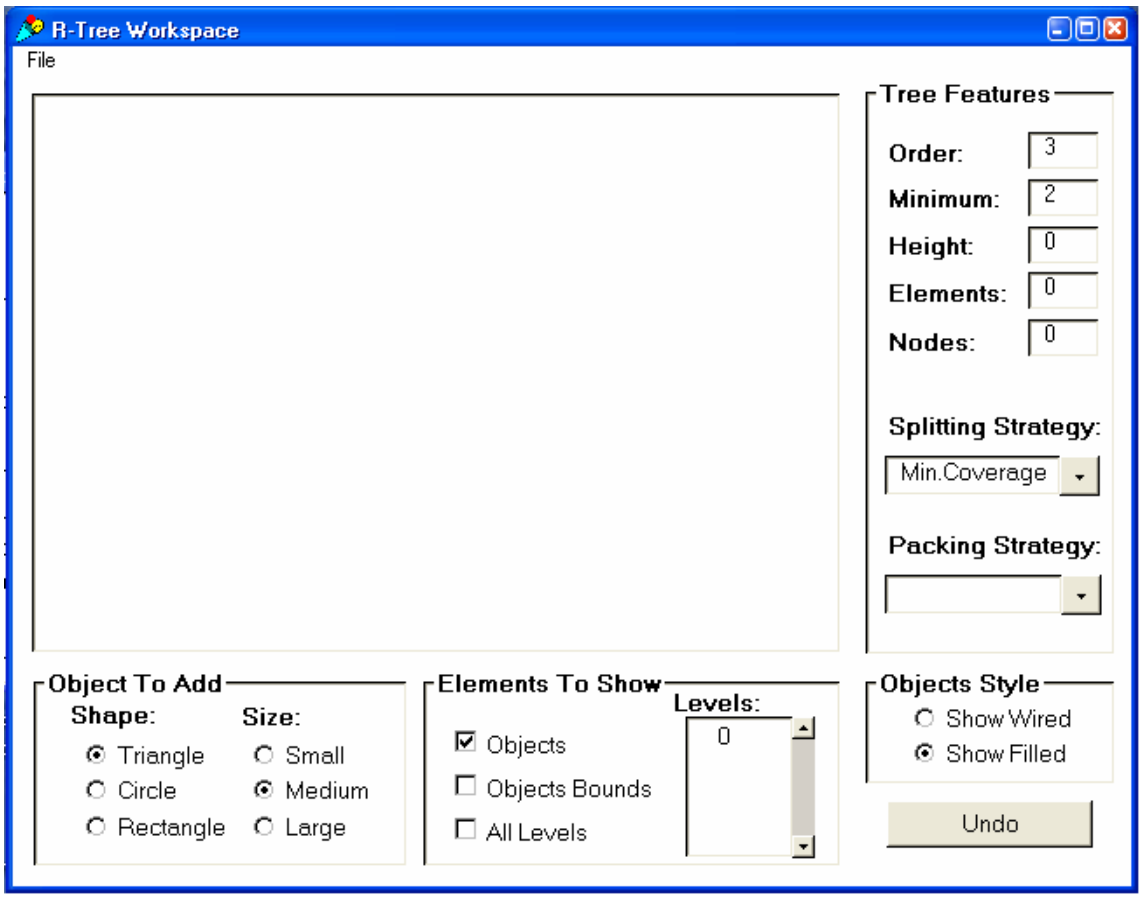



**Figura 6.11**.- Aplicación de Gestión del RTree

La aplicación cuenta con bastante flexibilidad en cuanto al tipo de objetos a agregar, la forma de visualizarlos a ellos y a la estructura de nodos, y también la estrategia de partición de nodos a utilizar. A continuación se describe la información que provee la interfaz como así también la manera en que pueden configurarse las características anteriormente enumeradas.

- **Tree Features:** Esta área permite la visualización y configuración del RTree sobre el que se monta la interfaz. Estos conceptos han sido discutidos detalladamente en el capítulo correspondiente.
  - **Order:** Orden del árbol.

- **Minimum:** Cantidad mínima de elementos (objetos u otros nodos) que puede contener un nodo. Depende directamente del orden del árbol.
- **Height:** Altura del árbol (variable conforme se agreguen o borren elementos). Es la cantidad de aristas entre cualquier nodo hoja (un RTree es siempre nivelado) y el nodo raíz, sin contar a los bounding boxes de los objetos como nodos.
- **Elements:** Cantidad de elementos agregados por el usuario.
- **Nodes:** Cantidad total de nodos del árbol (sin contabilizar los bounding boxes de los objetos).
- **Splitting Strategy:** Estrategia de partición de nodos. Es dinámicamente configurable por el usuario, puede cambiarse en cualquier momento.
  - **Minimum Coverage:** Estrategia de Cobertura Mínima. Realiza particiones de manera que el área total cubierta por los dos nodos resultantes sea mínima.
  - **Minimum Overlapping:** Estrategia de Superposición Mínima. Realiza particiones de manera que el área de superposición entre ambos nodos sea mínima.
- **Packing Strategy:** Estrategia de Empaquetamiento. Estrategia de carga masiva utilizada para generar el árbol.

- **Object To Add:** Permite configurar el objeto a agregar en cada operación de inserción. La misma se realizará en el lugar del panel en donde el usuario cliquee luego de realizar esta configuración.
  - **Shape:** Forma del objeto a agregar.
  - **Size:** Tamaño del objeto a agregar.

- **Elements To Show:** Permite especificar qué objetos o nodos desean visualizarse en el panel.
  - **Objects:** Permite mostrar u ocultar los objetos agregados por el usuario.
  - **Objects Bounds:** Permite mostrar u ocultar los bounging boxes de los objetos agregados por el usuario.
  - **Levels:** Permite mostrar u ocultar los nodos de niveles particulares seleccionados por el usuario.
  - **All Levels:** Facilidad o atajo de la aplicación: permite seleccionar todos los niveles de la lista anterior. Cuando este check-box se encuentra seleccionado el usuario no tiene habilitada la selección de niveles particulares en la lista Levels.

- **Objects Style:** Define el estilo en que pueden mostrarse los objetos agregados por el usuario.
  - **Show Wired:** Muestra sólo el contorno de los objetos.
  - **Show Filled:** Muestra a los objetos con su área rellena en negro.

- **Undo:** Permite realizar undo de operaciones.

### 6.2.2. Ejemplo de Operación

En esta subsección, haciendo uso de la aplicación presentada, se ilustrará mediante un ejemplo el funcionamiento del RTree implementado. Se agregará un conjunto de objetos y se mostrarán los pasos relevantes de la evolución de la estructura.

En la figura siguiente se muestra la aplicación abierta sobre un RTree de orden 3, lo cual implica que el máximo de elementos (objetos o nodos hijos) que puede tener un nodo es 3, mientras que el mínimo es 2. Acorde a lo explicado oportunamente, cuando un nodo de este árbol particular supere los 3 elementos caerá en overflow, por lo que deberá procederse a su partición y redistribución de sus elementos. Esta estrategia de partición puede ser elegida e incluso cambiada por el usuario en cualquier momento, para el caso de nuestro ejemplo se ha seleccionado la Estrategia de Cobertura Total Mínima. En la figura se muestra el estado del árbol inmediatamente después del agregado del tercer elemento. Pueden observarse estos elementos junto con sus bounding boxes, todos ellos contenidos dentro de un nodo de la estructura: el nodo raíz.

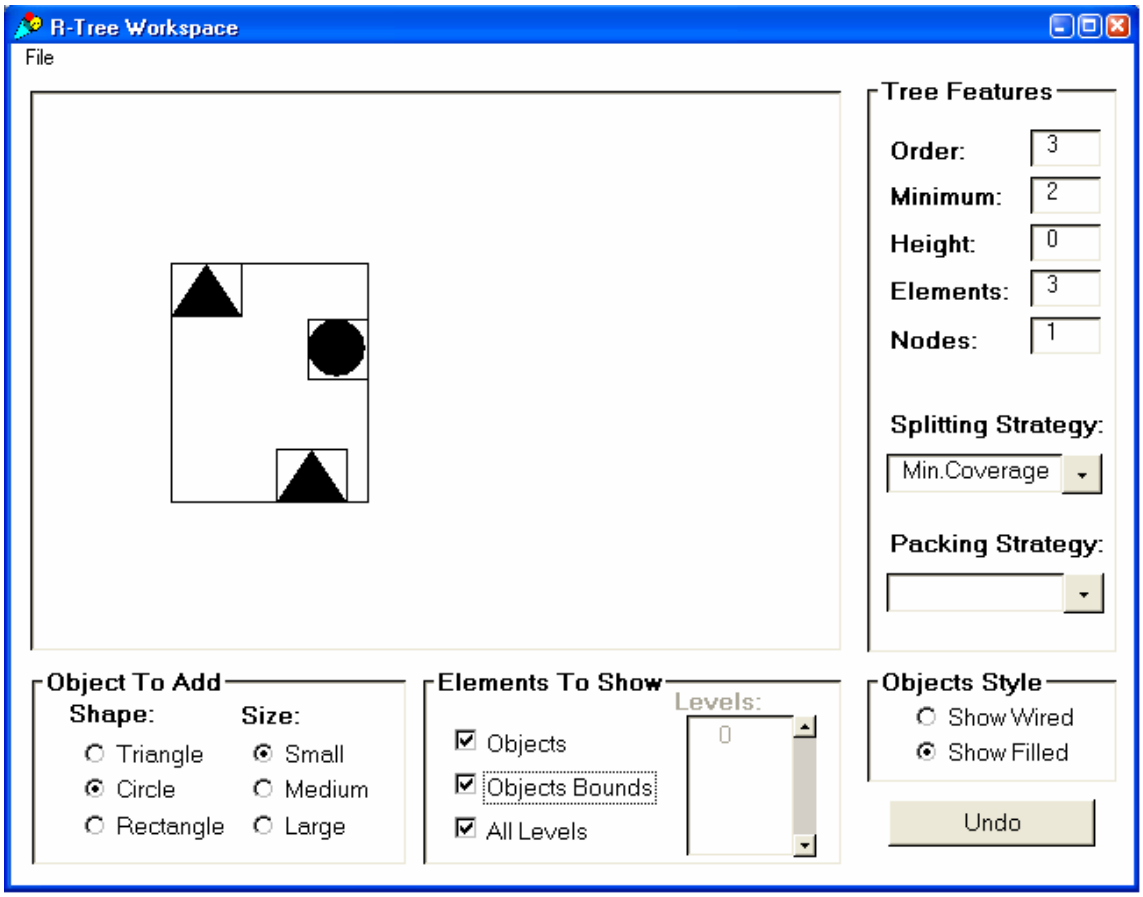



**Figura 6.12**.- Nodo raíz con tres elementos

El único nodo de la estructura se encuentra lleno hasta su capacidad máxima (tres), por lo que el agregado de un nuevo elemento causará overflow y deberá procederse a una partición del mismo. La situación resultante causada por este nuevo agregado se muestra en la siguiente figura.

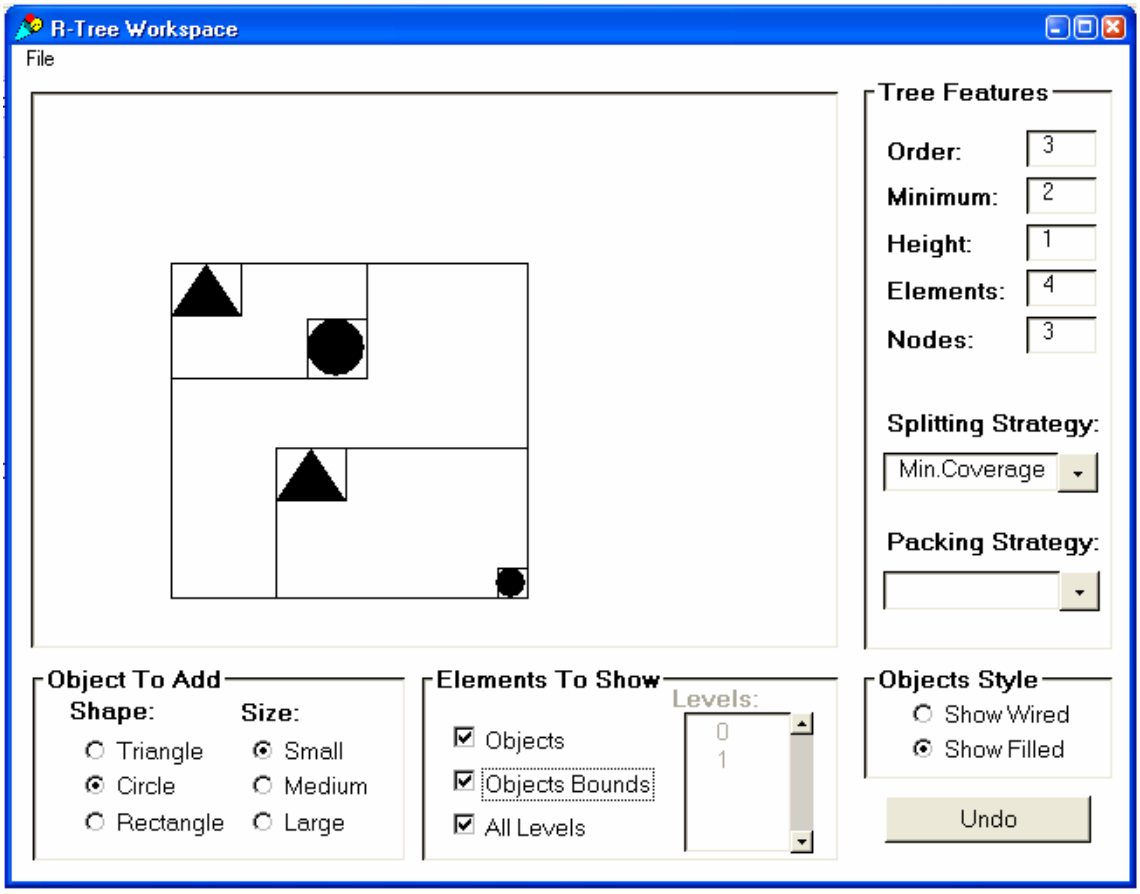



**Figura 6.13**.- Partición causada por overflow del nodo raíz original

Como puede observarse se ha generado una partición: se ha creado un nodo hermano al original, y sus elementos han sido distribuidos entre ambos (mediante la estrategia de partición seleccionada). A su vez, en este caso, la partición ha generado un nuevo nodo raíz al que se la ha asignado como hijos a los dos nodos mencionados. La altura del árbol se ha incrementado en 1, tal como puede observarse en el campo Height.

A continuación se agrega otro elemento que, por su ubicación espacial, se inserta en uno de los dos nodos hoja haciendo extender los límites de su bounding box, lo que por recursividad también produce el agrandamiento del bounding box del nodo raíz. El nodo hoja que ha recibido al nuevo elemento ha alcanzado el máximo de su capacidad. Tal situación se muestra en la siguiente figura.

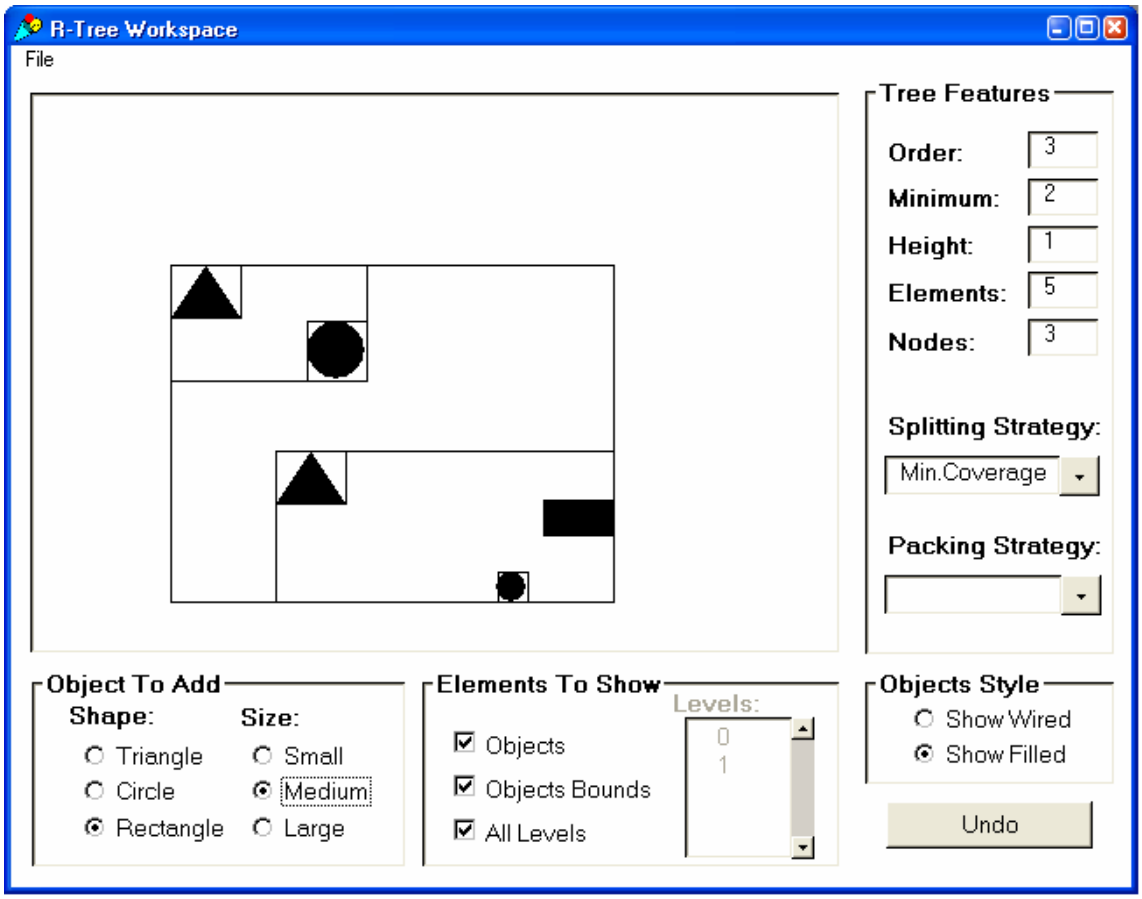



**Figura 6.14**.- Agregado de nuevo elemento y consecuente reestructuración de los bounding box de los nodos

En la siguiente figura se muestra cómo el agregado de un nuevo elemento causa la partición de uno de los nodos hoja, generando ahora un nuevo hermano para los dos previamente existentes. En este momento el nodo raíz tiene tres nodos hijos, por lo que ha alcanzado el máximo de su capacidad.

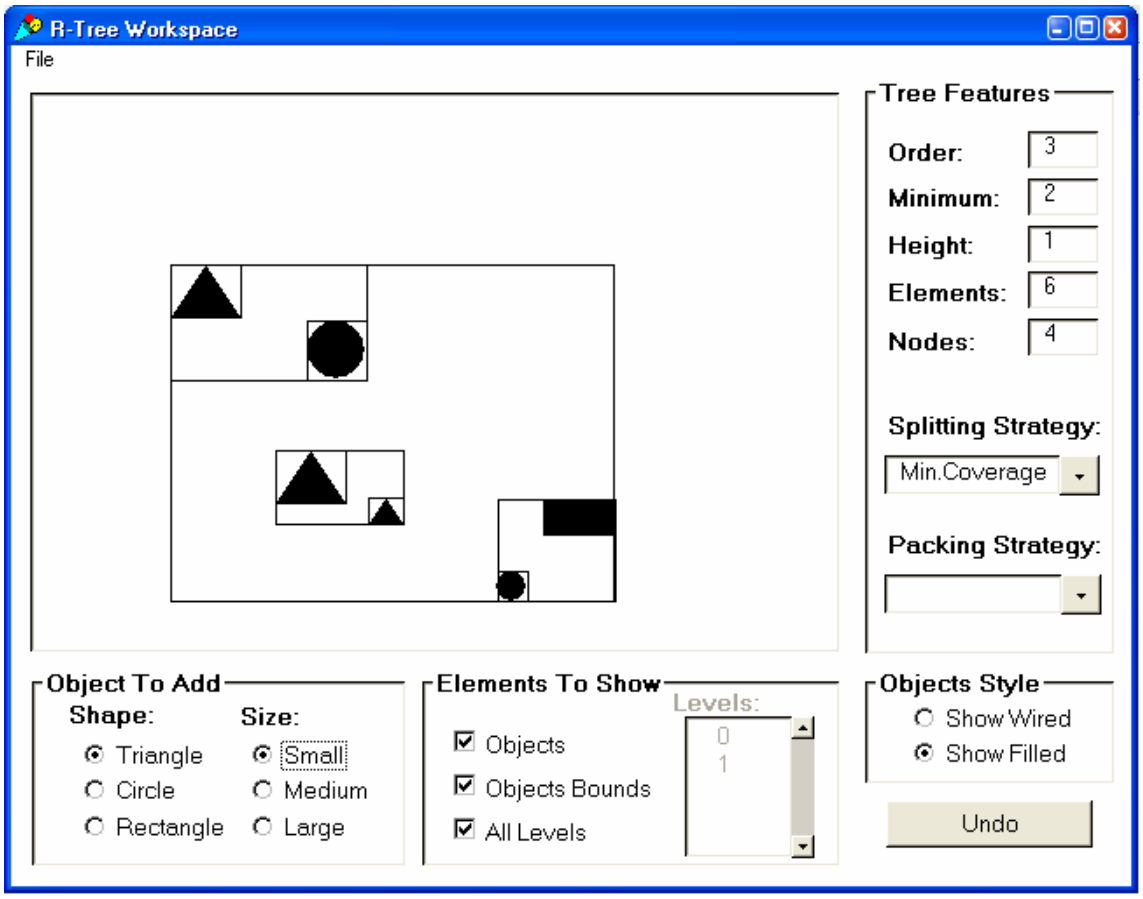



**Figura 6.15**.- Agregado de nuevo elemento y consecuente partición. Nodo raíz al máximo de capacidad.

En la figura 6.15 se observa que el agregado de un nuevo objeto lleva a uno de los nodos hoja hasta su cantidad límite de elementos soportados, por lo que el agregado de otro nuevo objeto produce la partición mostrada en la figura 6.16.

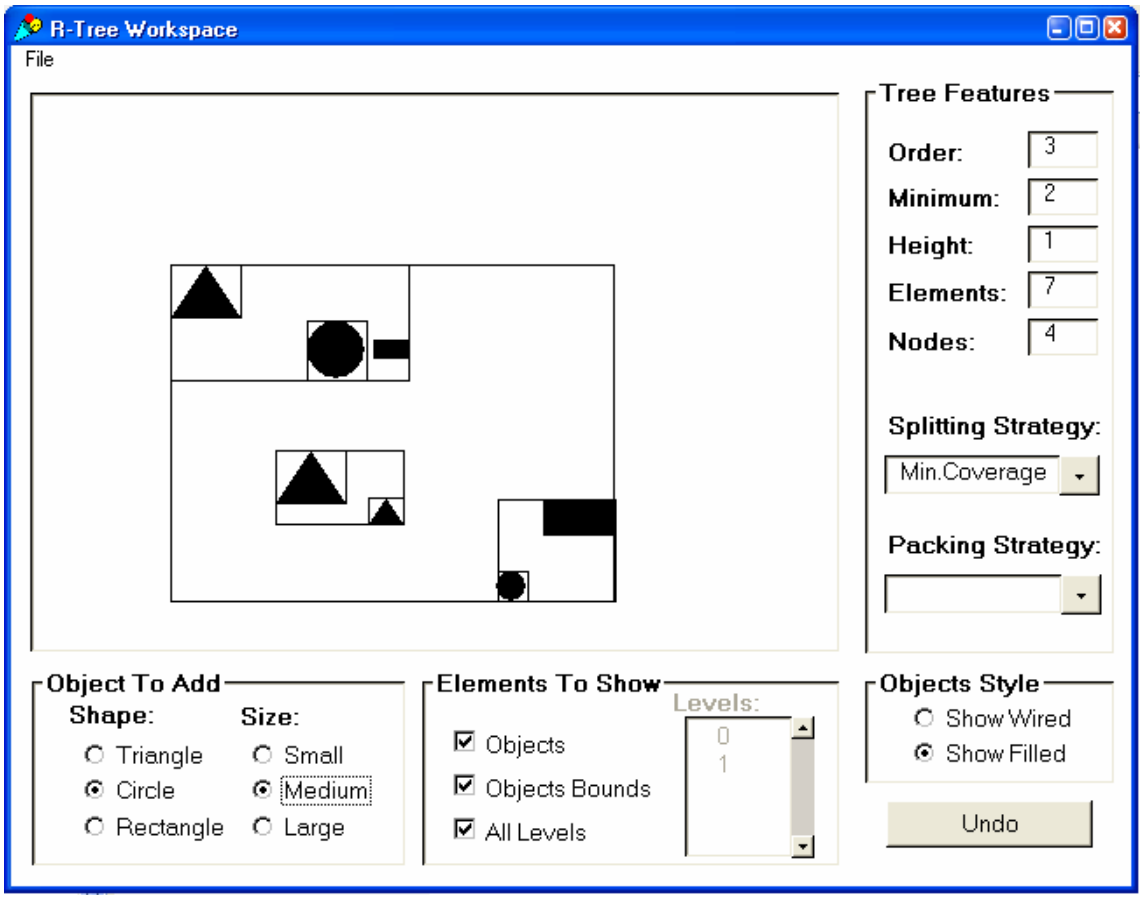



**Figura 6.16**.- Agregado de nuevo elemento. Nodo hoja al máximo de capacidad

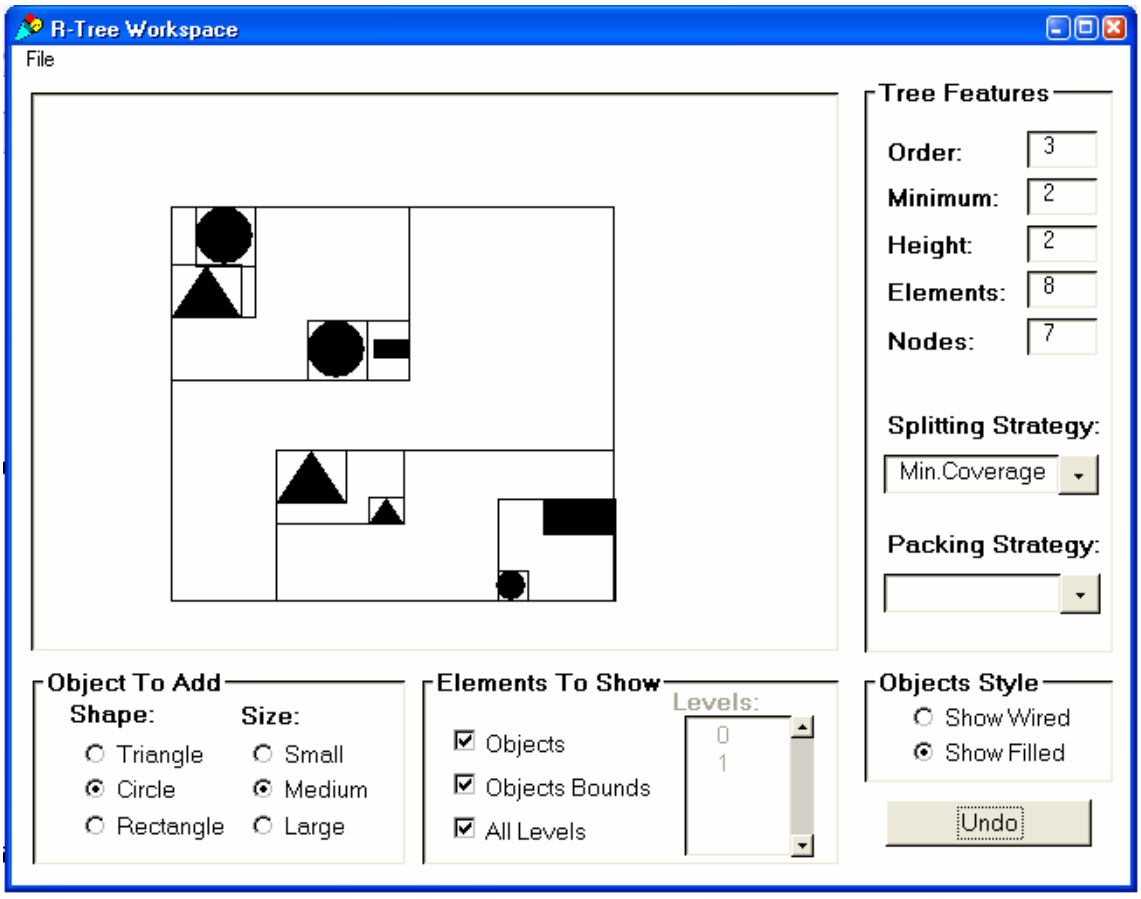



**Figura 6.17**.- Agregado de nuevo elemento. Partición recursiva desde nodo hoja hasta la raíz

Como se observa en la figura inmediatamente superior, el overflow del nodo hoja produjo una partición, con la generación del nuevo nodo hermano que tal operación implica. A su vez, por tener ahora cuatro hijos, también la raíz ha entrado en overflow, lo que causó también su partición y la generación de una nueva raíz, incrementando nuevamente la altura del árbol.

Consideramos que éste ha sido un ejemplo lo suficientemente significativo para mostrar el funcionamiento de la estructura implementada y el de la aplicación de prueba. A continuación se muestran distintas formas de observar la última situación según algunas de las alternativas que permite la aplicación, mediante las mismas pueden verse distintos aspectos de forma aislada para una mejor comprensión.

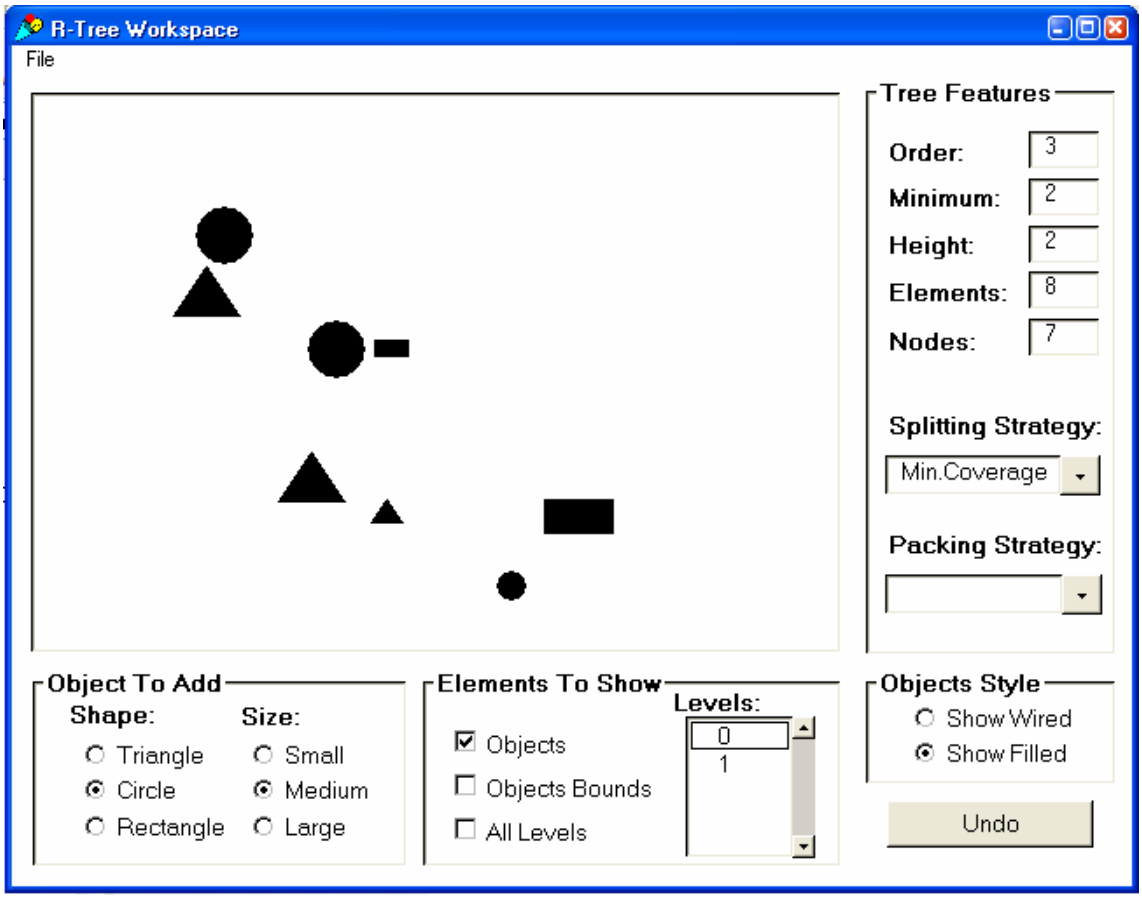



**Figura 6.18**.- Vista sólo de los objetos agregados

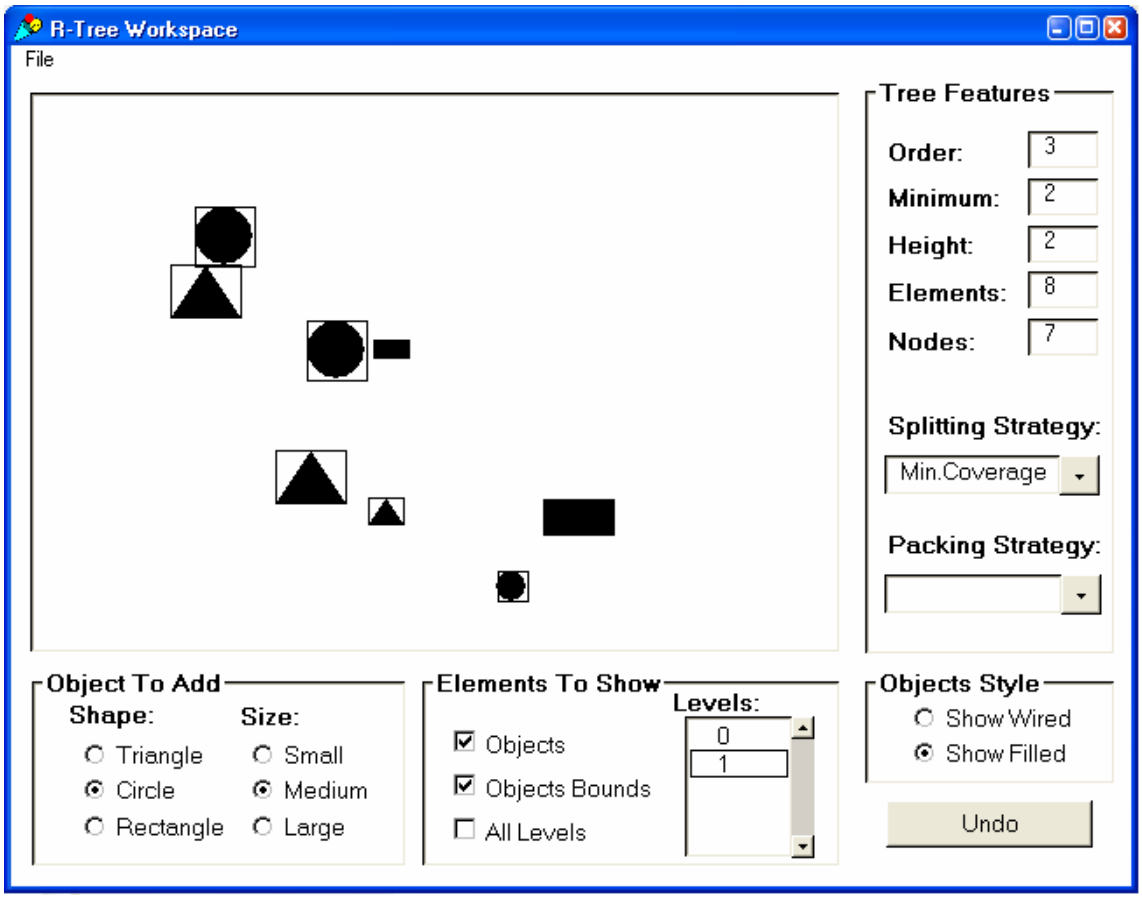



**Figura 6.19**.- Vista de los objetos agregados y sus bounding boxes

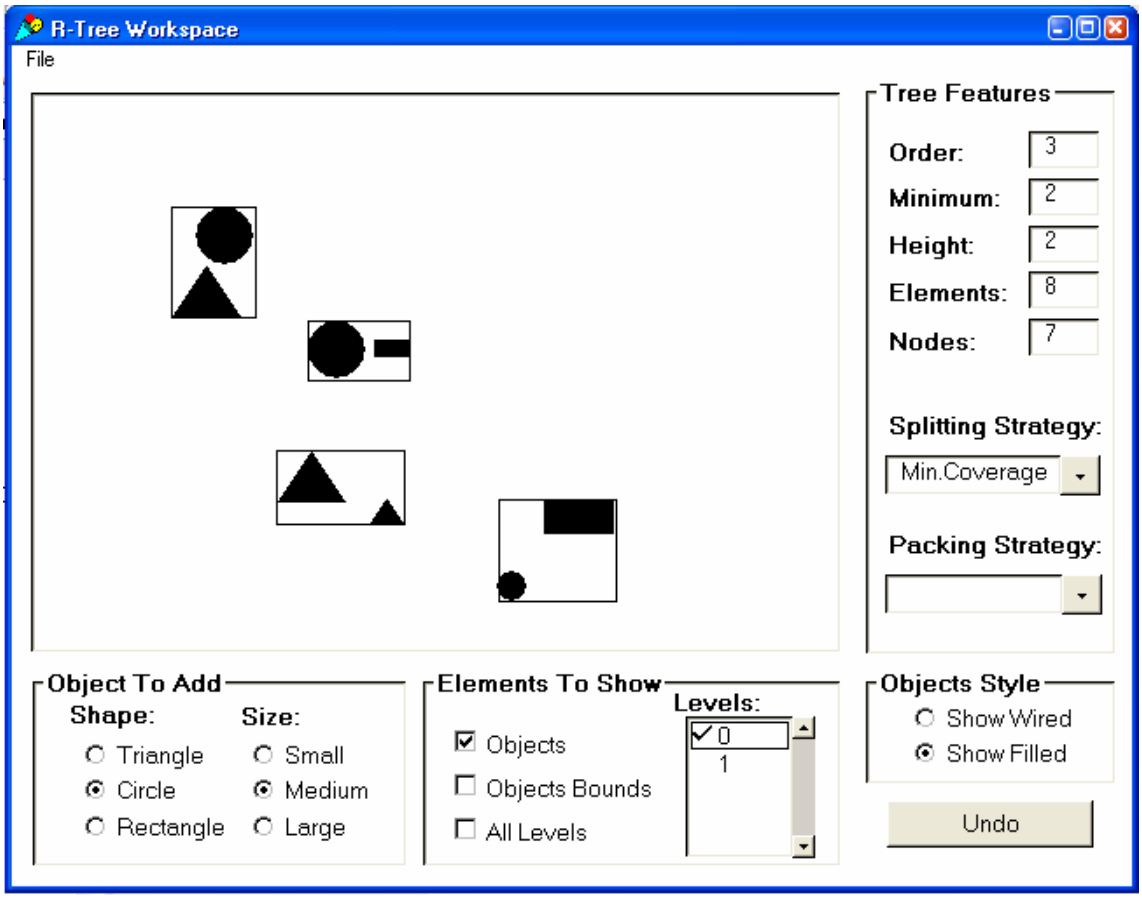



**Figura 6.20**.- Objetos (sin sus bounding-boxes) y nivel inferior de nodos del árbol

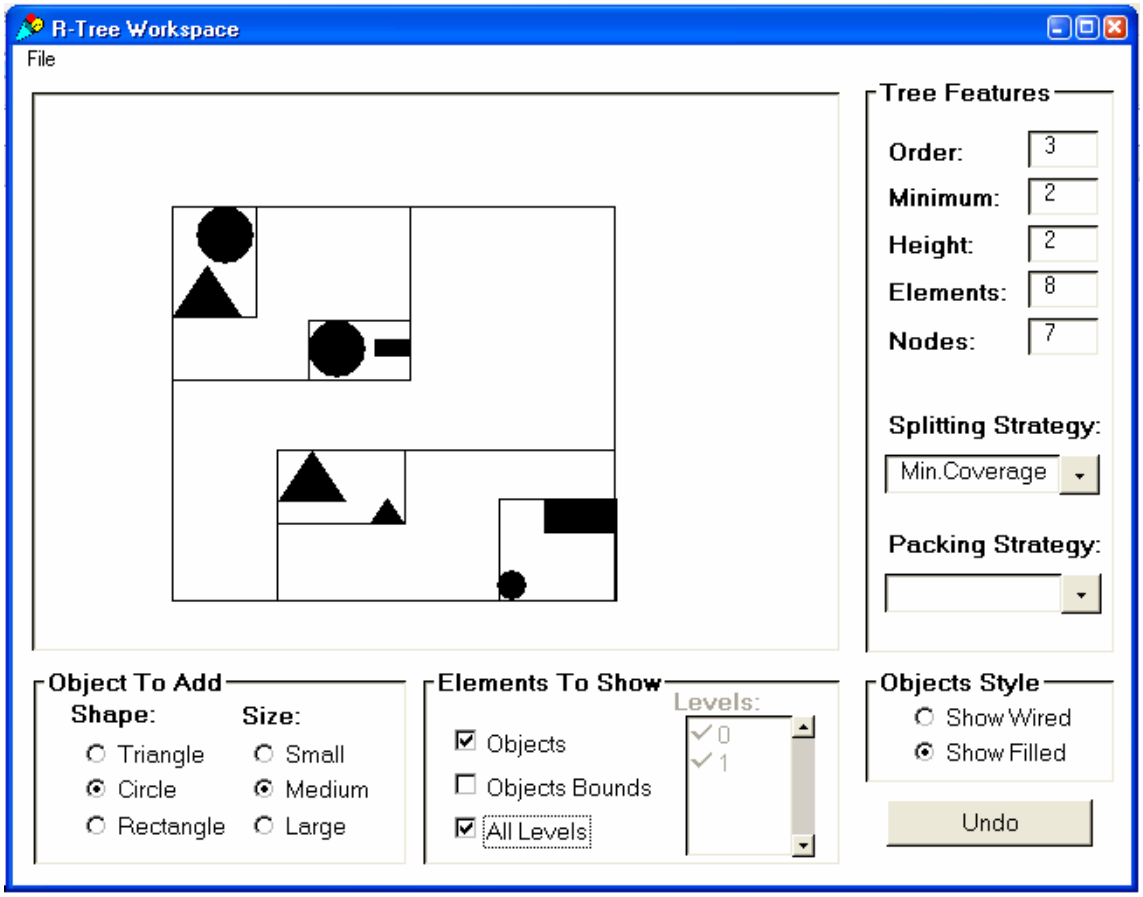



**Figura 6.21**.- Objetos (sin sus bounding-boxes) y todos los niveles de nodos del árbol

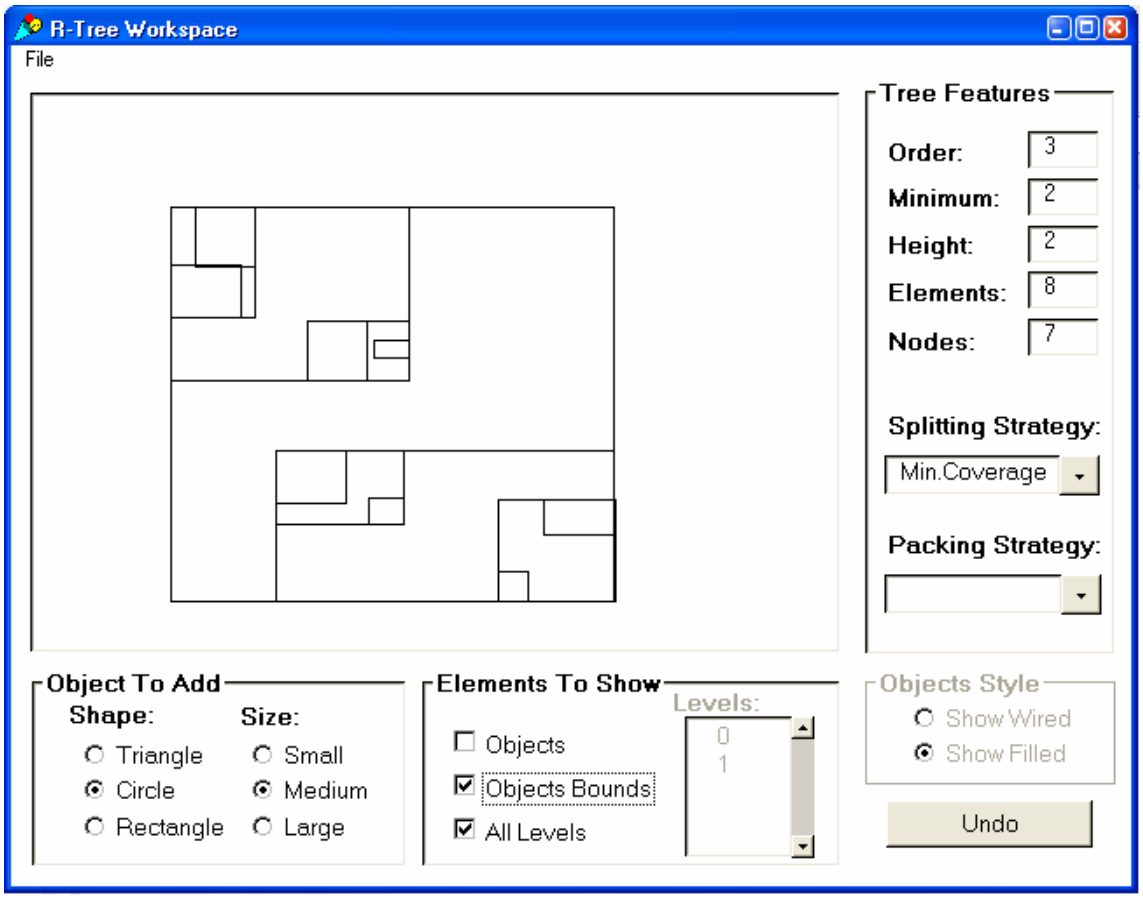



**Figura 6.22**.- Estructura resultante del RTree incluyendo los bounding boxes de los objetos

La aplicación permite también operaciones de búsqueda y borrado de elementos. La aplicación abre un menú contextual (pop-up menu) sobre el punto del panel en el que el usuario cliquea; tal menú permite ordenarle al RTree subyacente que borre o busque todos los elementos que se encuentran en tal posición.

Las dos figuras siguientes muestran el caso de la búsqueda de un elemento.

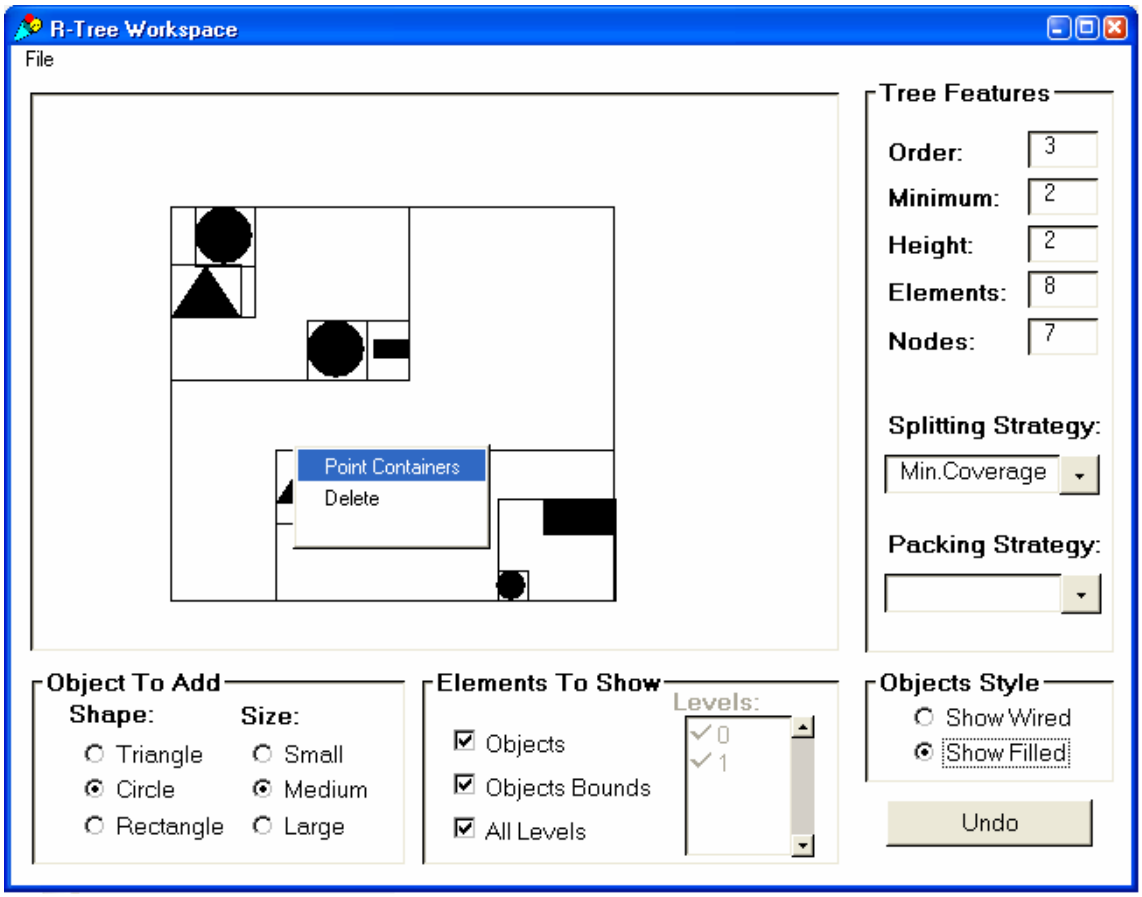



**Figura 6.23**.- Menú de Borrado y Búsqueda de objetos en un punto espacial

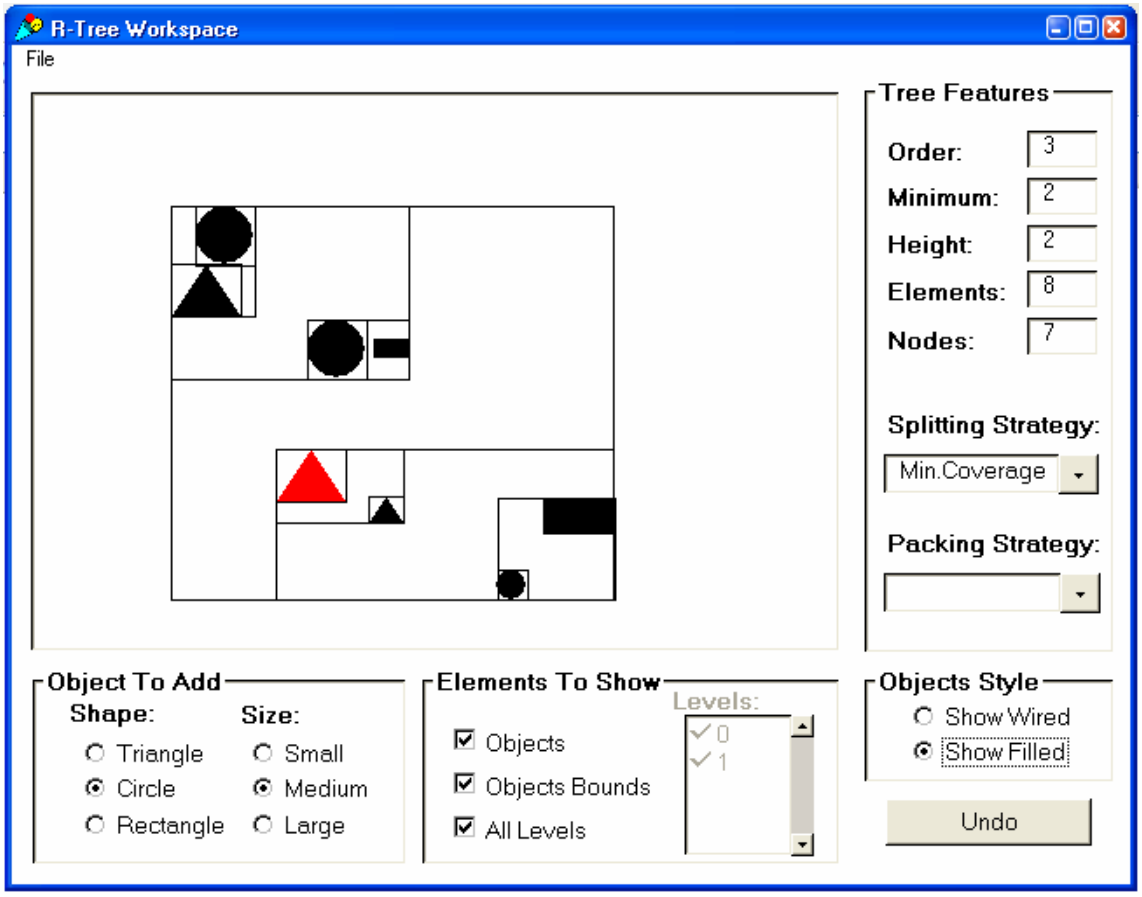



**Figura 6.24**.- Elemento devuelto por la estructura ante la consulta de búsqueda en un punto dado

### 6.2.3. Estrategias de Partición

Como se ha expuesto anteriormente, el usuario puede seleccionar dinámicamente la estrategia de partición e incluso crear y agregar nuevas.

En esta subsección se mostrará brevemente cómo se comporta la estructura ante un mismo juego de datos con una y otra de las estrategias implementadas.

A continuación se muestran dos figuras. En la primera de ellas se presenta un juego de datos de tres elementos, y en la siguiente se muestra cómo los mismos son agrupados en un RTree de orden tres. En este caso consta sólo de un nodo que, por tener tres elementos, se encuentra lleno hasta el límite de su capacidad, de manera que el agregado de un cuarto elemento generaría una partición.

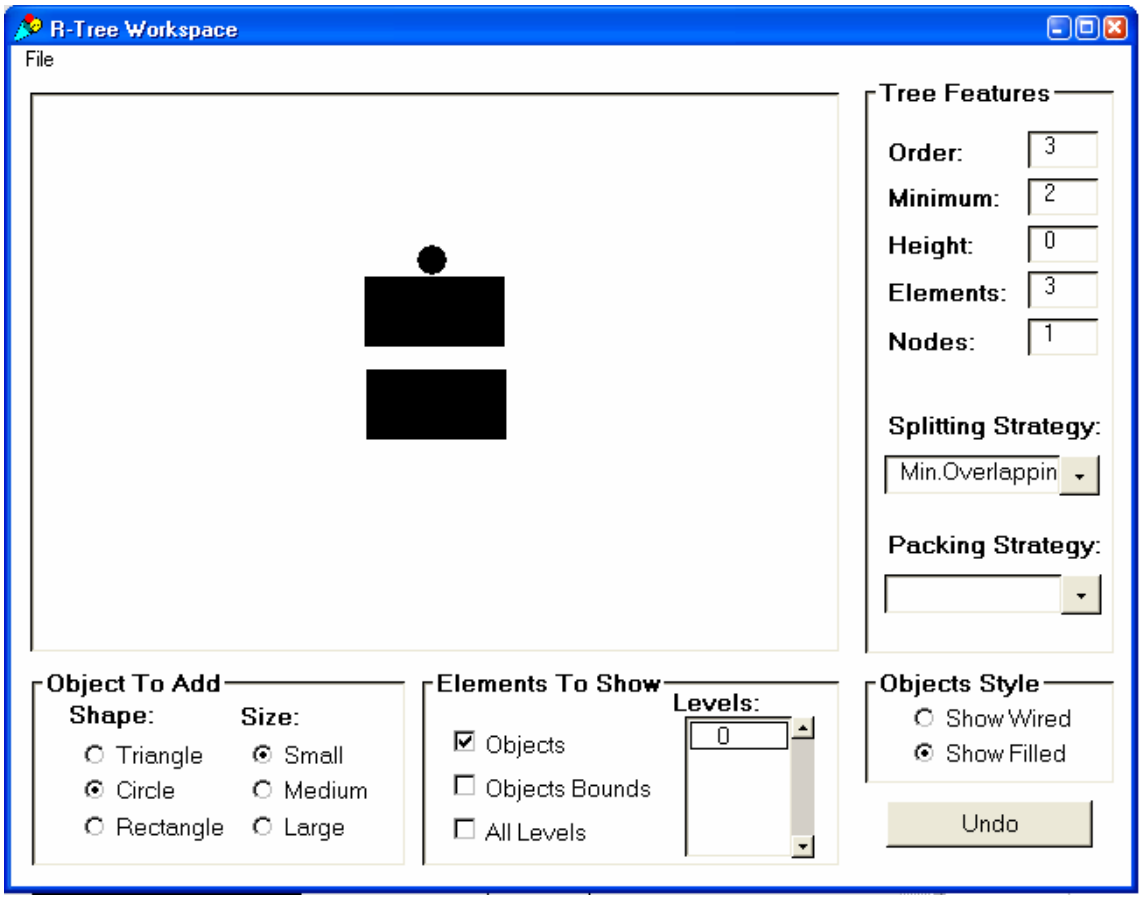



**Figura 6.25**.- Juego de datos de tres elementos iniciales

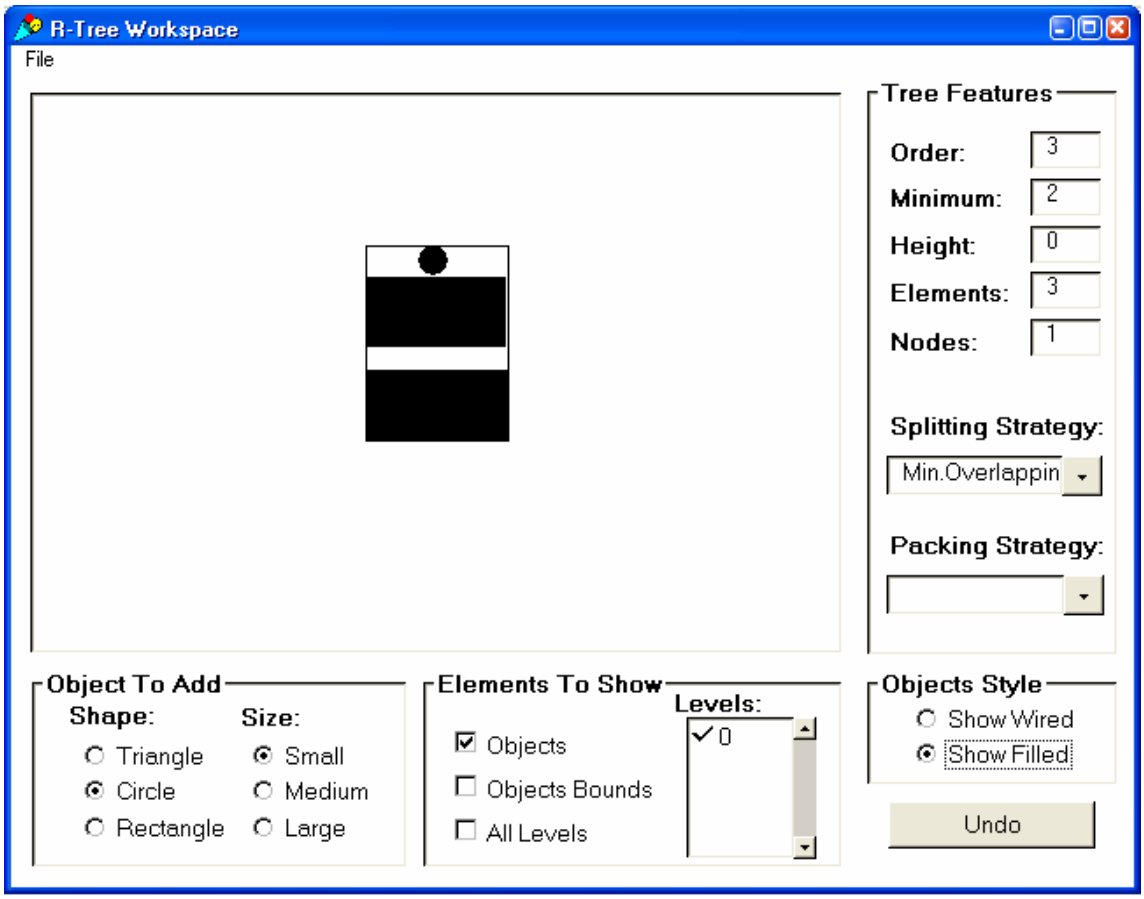



**Figura 6.26**.- Agrupación de los datos iniciales en el nodo raíz del RTree

En las dos figuras siguientes se muestra la diferencia entre agregar un mismo elemento en una misma posición pero utilizando dos estrategias distintas de partición. En el primer caso se utiliza la estrategia de Superposición Mínima, ésta distribuye los elementos entre los dos nodos de manera de reducir al mínimo el área de superposición entre ambos. Por otro lado, en el segundo caso, se utiliza la estrategia de Cobertura Total Mínima, ésta distribuye los elementos entre los dos nodos de manera que el área total de cobertura de los mismos sea mínima.

Las ventajas y desventajas de cada estrategia han sido analizadas oportunamente por lo que no se volverá a hacerlo aquí. Sólo se destaca que estos dos objetivos (reducir área total o reducir área de superposición) no siempre son contradictorios, sin embargo, para una mejor apreciación del funcionamiento de la estructura implementada en relación a este aspecto, se ha elegido un ejemplo en el que sí lo son, lo que implica diferencias en cuanto a la agrupación resultante.

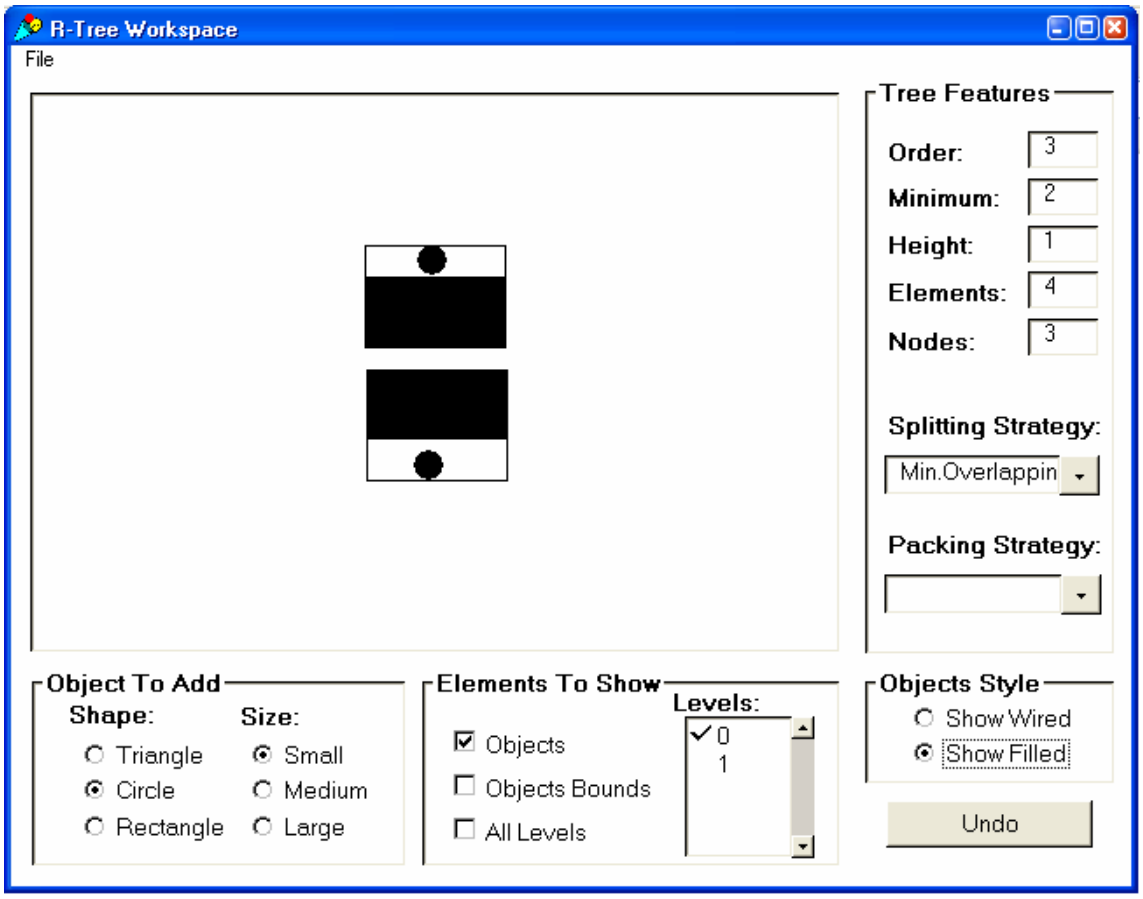



**Figura 6.27**.- Agregado de nuevo elemento. Distribución según la Estrategia de Superposición Mínima

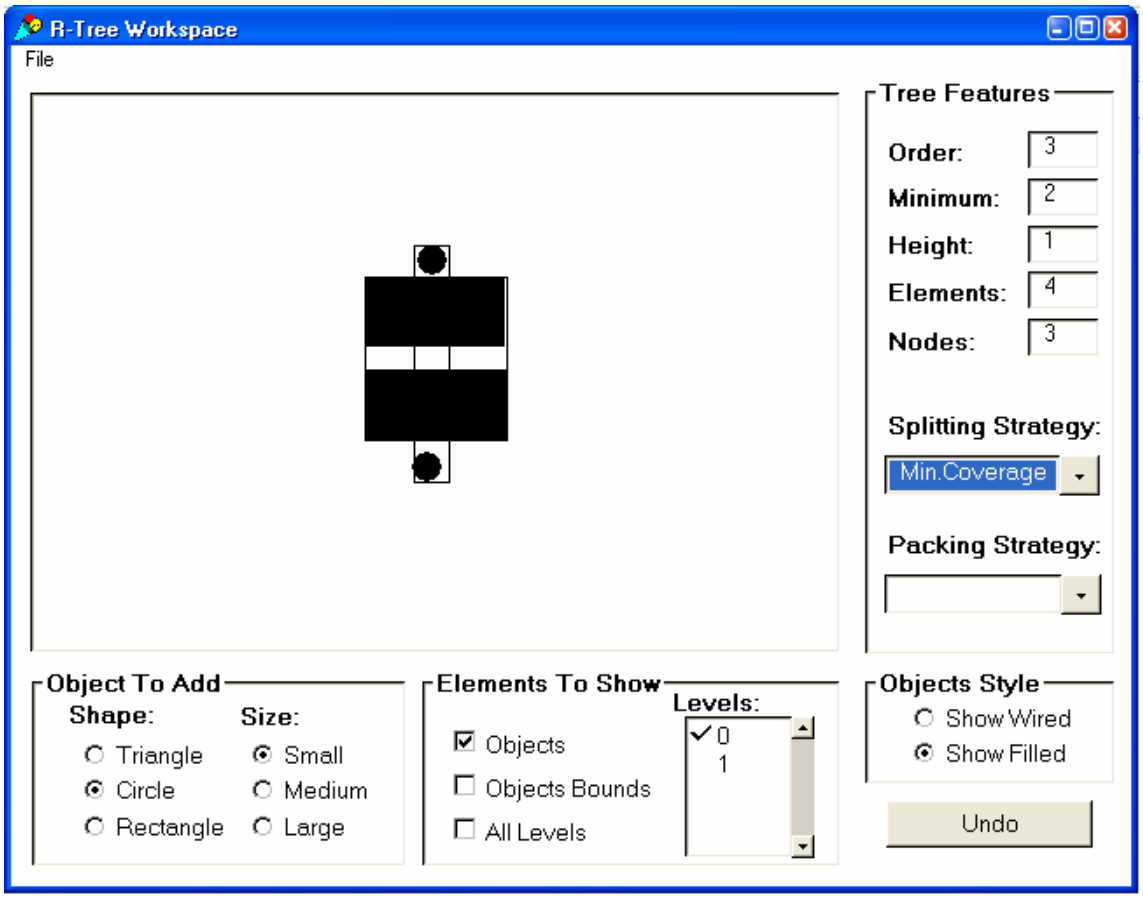



**Figura 6.28**.- Agregado de nuevo elemento. Distribución según la Estrategia de Cobertura Total Mínima

En la siguiente figura puede observarse la misma situación de la figura anterior pero con los objetos en estilo transparente (wired), de manera de apreciar más claramente el área abarcada por los nodos que los contienen.

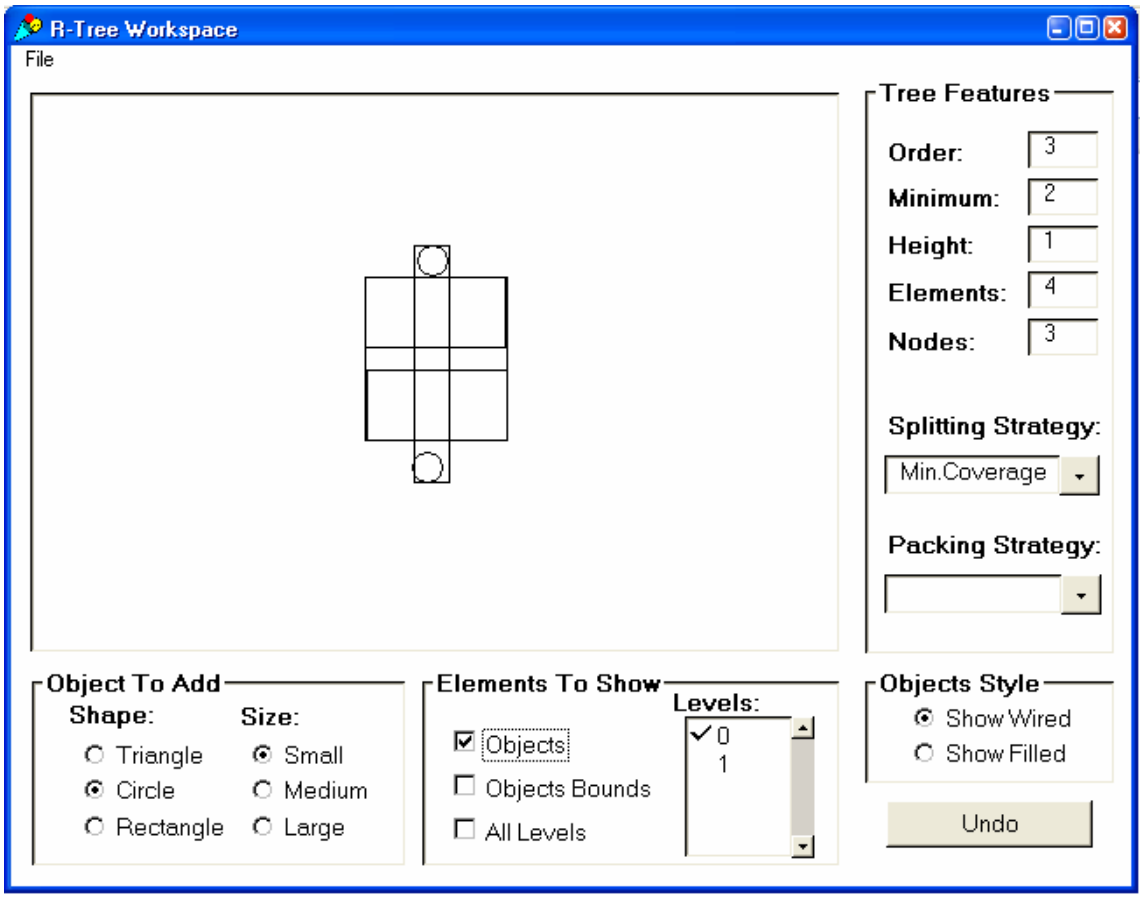



**Figura 6.29**.- Distribución según la Estrategia de Cobertura Total Mínima. Vista con objetos transparentes

# 7. CAPÍTULO VII: DESARROLLO DEL MODELO DE GEOMETRÍA COMPUTACIONAL

*En este capítulo se describe la manera en que se introducen en el modelo OO desarrollado las estrategias de Geometría Computacional presentadas en el Capítulo IV. Se intenta en todo momento lograr flexibilidad en cuanto a la elección de las mismas por parte del usuario, e incluso se propone un modelo abierto a extensión con nuevas estrategias. Se discuten también algunos aspectos relacionados a la utilización del espacio de almacenamiento y de mejoras de performance mediante algunas técnicas reconocidas.*

*Finalmente se muestran algunos ejemplos concretos de una implementación del algoritmo de Slabs mediante una interfaz de prueba desarrollada por los autores.*

## 7.1. Modelo de Geometría Computacional

Si bien suele relacionarse a los algoritmos de geometría computacional con implementaciones en lenguajes de bajo nivel, en esta sección se muestra cómo los mismos pueden integrarse a un modelo OO, incluso mediante estrategias OO. Tales estrategias constituyen la aplicación de algunos patrones de diseño que favorecen la flexibilidad y la performance, permitiendo al usuario seleccionar los algoritmos más adecuados para su caso particular manteniendo un buen tiempo de ejecución y un razonable espacio de almacenamiento.

### 7.1.1. Operaciones básicas del modelo

Anteriormente se ha descripto el modelo Topológico, que incluye a los polígonos entre sus elementos principales. Además, también se han desarrollado algunos algoritmos provistos por la Geometría Computacional para operaciones entre polígonos como la intersección, unión y diferencia. En esta sección se describirá la manera en que estos algoritmos son incluídos en el modelo Topológico explotando las ventajas del Paradigma Orientado a Objetos.

La alternativa más directa de implementación es la definición de un mensaje para cada operación en la clase Polygon del modelo topológico, cada mensaje toma como parámetro el polígono que constituye el segundo operando. El método asociado a cada mensaje consiste en la implementación de alguno de los métodos de Geometría Computacional ya descriptos (Slabs, SwitchingTrip).

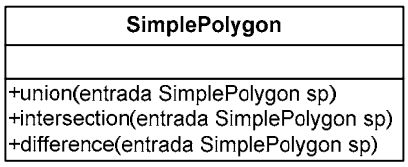



**Figura 7.1**.- Simple Polygon.

Las operaciones sobre polígonos más complejos, como los disjuntos o con agujeros, no incluyen en sí mismas metodologías de Geometría Computacional sino que se basan en la delegación de las operaciones en los polígonos más simples por los que están compuestos. Por ejemplo, la intersección entre dos polígonos disjuntos se obtiene mediante varias operaciones de intersección entre los polígonos simples que los conforman.

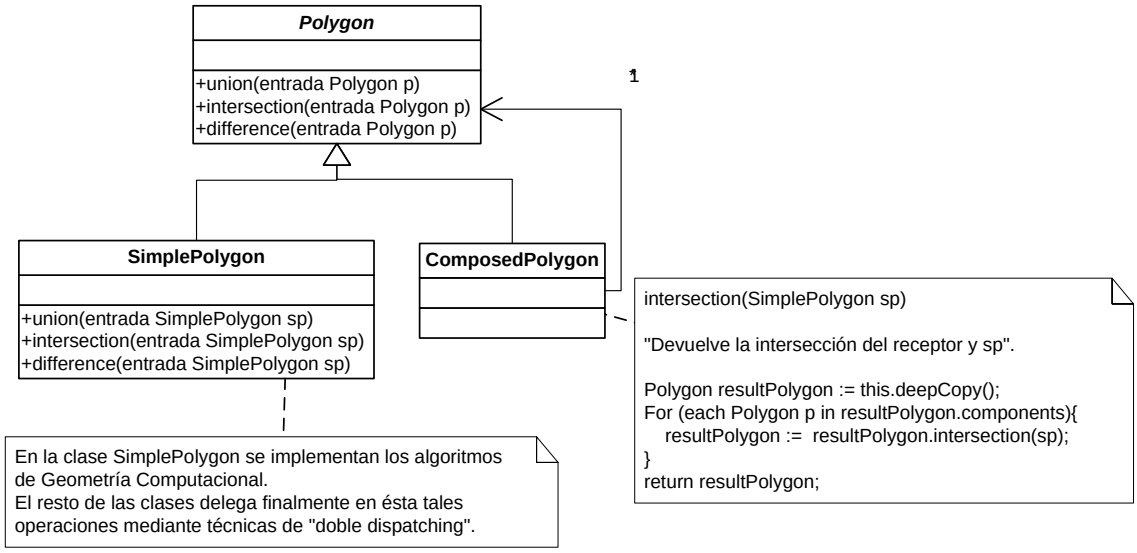



**Figura 7.2**.- Jerarquía de Polígonos.

Dado que esta sección trata de la implementación de los algoritmos geométricos y que, en virtud de la delegación, éstos sólo se encuentran implementados en la clase SimplePolygon, a partir de ahora reduciremos la discusión a esta clase, ignorando a las demás sin pérdida de generalidad.

De acuerdo al contexto y dominio de una aplicación SIG en particular, puede resultar más ventajoso utilizar uno u otro algoritmo de intersección, unión o diferencia. Por ejemplo, la cantidad de vértices de los polígonos o la cantidad estimada de puntos de intersección entre los mismos pueden ser factores que claramente determinen la conveniencia a favor de la utilización de un algoritmo determinado. Sin embargo, en este modelo, el hecho de implementar el algoritmo directamente en un método de la clase SimplePolygon, fija definitivamente una única metodología de operación y el usuario no podrá cambiarla por otra en tiempo de ejecución.

Para lograr mayor dinamismo en el cambio y agregado de nuevos algoritmos se propone la utilización del patrón de diseño *Strategy* [Gamma]. Este patrón permite desacoplar a los algoritmos geométricos de las clases que los utilizan. Esto se consigue implementándolos ya no en la clase SimplePolygon, sino en clases separadas (una clase por cada algoritmo), de manera que las instancias de éstas puedan ser dinámicamente referenciadas por las de aquella. En la figura siguiente se muestra este desacoplamiento para el caso de la operación de intersección.

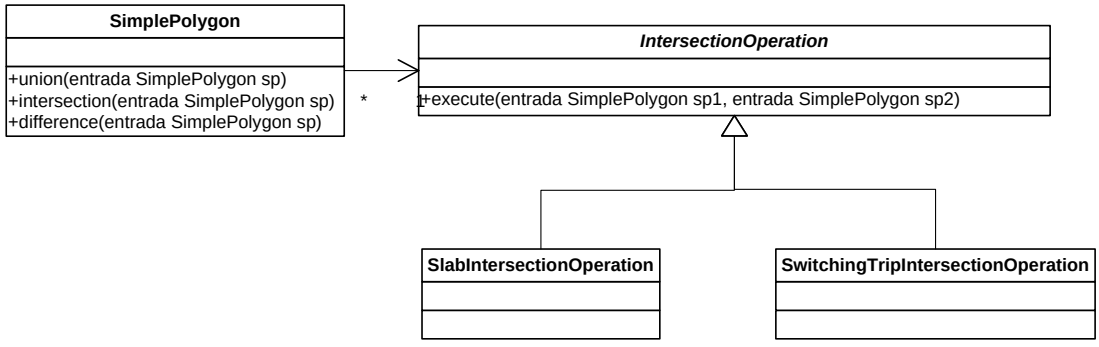



**Figura 7.3**.- Jerarquía de Operaciones.

Con este modelo, la instancia de un polígono puede utilizar en un momento determinado el algoritmo de intersección mediante Slabs (referenciando y colaborando con una instancia de la clase SlabIntersectionOperation), pero puede, por ejemplo, cambiarlo en cualquier momento por el de

intersección mediante la estrategia de Switching Trip (referenciando y colaborando con una instancia de la clase SwitchingTripIntersectionOperation). Este esquema definitivamente resuelve el problema de la flexibilidad, pero introduce otro en cuanto que también deben crearse jerarquías similares para las otras operaciones (unión, diferencia, diferencia simétrica). Consecuentemente, cada polígono del sistema debe referenciar a varios de estos objetos: uno objeto por cada operación. Esto significa, en mayor detalle, que cada instancia de la clase SimplePolygon deberá tener n referencias extras (siendo n el número de operaciones que implemente, por ejemplo: intersección, unión, diferencia) lo que implica n nuevas variables de instancia en cada uno y que, además, por cada nueva instancia de la clase SimplePolygon que se cree se deberán instanciar n nuevos objetos (una instancia de estrategia por cada operación). Puede observarse que la proliferación de nuevos objetos es muy grande, y además la clase SimplePolygon se hace más grande por las nuevas referencias que debe contener.

La solución a estos dos problemas se consigue mediante la aplicación del patrón de diseño *Singleton* [Gamma]. Este patrón sugiere que en algunos casos determinados no es necesario crear muchas instancias de una misma clase, sino que alcanza con crear una sola y que todos los objetos del sistema que necesiten una instancia de tal clase compartan la misma (la única en todo el sistema). Esta solución sólo puede utilizarse cuando el estado de los objetos a los que se hace referencia es independiente de los objetos cliente (que son los que lo referencian). Esto es generalmente cierto para los objetos cuya única responsabilidad en el sistema es proveer comportamiento, funcionalidad; tal es exactamente el caso de las instancias de cada una de las clases que definen estrategias de operaciones en nuestro modelo. Por ejemplo, de haber muchas instancias de la clase SlabIntersectionStrategy todas serían exactamente iguales, por lo tanto es lo mismo que haya una sola y todos los polígonos la referencien. Esto reduce drásticamente la cantidad de objetos del sistema sin alterar en lo más mínimo la funcionalidad deseada y queda resuelto el problema de la proliferación de objetos. En la figura siguiente se muestra un caso concreto de esta solución.

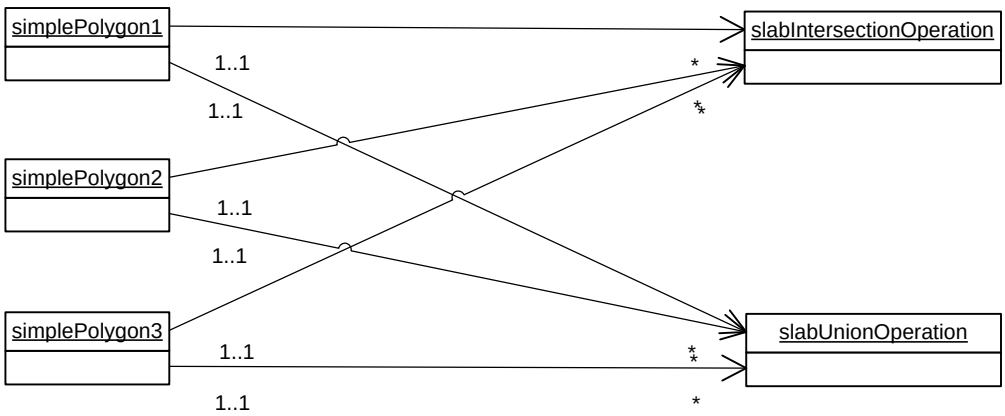



**Figura 7.4**.- Singleton de Operaciones

Aún queda por resolver el problema de la cantidad de referencias por parte de cada polígono a las instancias de estrategias de operaciones. De hecho, puede observarse en la figura que todos los polígonos deben mantener referencias a cada una de las estrategias, aunque éstas sean únicas en el sistema. Sin embargo resulta importante notar el hecho de que esto no constituye necesariamente un problema, sino que por el contrario, podría redundar en una ventaja en cuanto a que cada polígono (cada instancia) puede utilizar su propia estrategia para cada operación; por ejemplo, algunos podrían intersectarse utilizando el método de Slabs y otros mediante el método de SwitchingTrip. Es perfectamente implementable y para nada descabellado un esquema en el cual cada polígono tenga asociada una determinada estrategia de intersección de acuerdo a aspectos muy particulares e inherentes a sí mismo, como por ejemplo su convexidad, si es estrella, o su cantidad de vértices. Esta posibilidad se muestra en la figura siguiente.

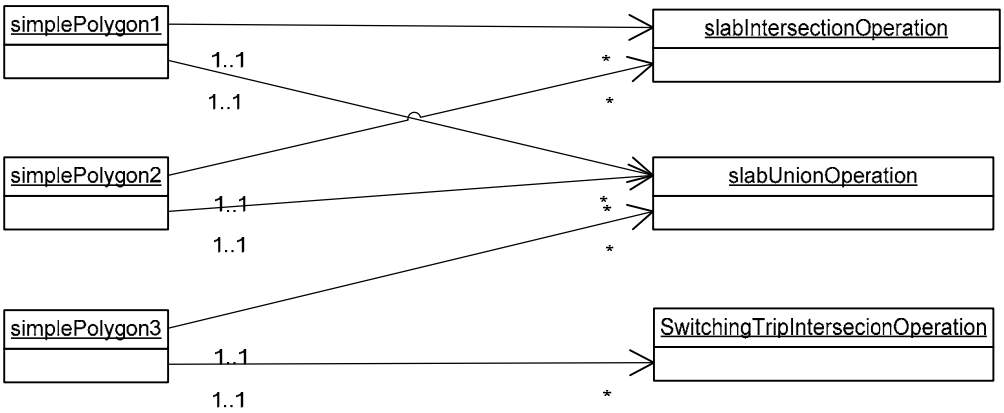



**Figura 7.5**.- Configuración particular de operaciones.

Entendemos que para la mayoría de los casos, aunque debe posibilitarse que las estrategias puedan variar, serán las mismas para todos los polígonos; es decir, en todo momento todos los polígonos utilizan las mismas estrategias, y de variar, variarán para todos. Para estos casos, que constituyen la gran mayoría, no se justifica pagar el precio de mantener las referencias en cada polígono, tanto por razones de diseño como por razones de espacio físico.

Como solución, se opta por almacenar estas referencias no a nivel local (en cada polígono) sino a nivel global, de manera tal que todos lo polígonos puedan accederlas. Sabemos que cualquier objeto en principio puede referenciar cualquier clase (cualquier clase, no cualquier instancia); como el ejemplo más común de esto tenemos el momento en que una instancia referencia a una clase para instanciarla. Por lo tanto, una de las maneras de lograr globalidad consiste en que una clase determinada (no una instancia, sino una clase en sí) se haga cargo de almacenar y devolver las distintas instancias de los objetos Strategy. Esta clase sabe qué estrategia se debe utilizar para cada operación. También implementa mensajes del tipo “getIntersectionStrategy”, “getUnionStrategy”, de manera tal que cada uno de ellos devuelve la estrategia de intersección o unión que se está utilizando en todo el sistema en ese momento (pues es ella quien las centraliza). De este modo, cambiar la estrategia de intersección del sistema consiste sólo en reemplazar la instancia de IntersectionStrategy en esa clase, de manera que a partir de ese momento el mensaje “getIntersectionStrategy” devuelva la nueva instancia; los polígonos no notarán el cambio debido a que todas las instancias de estrategias son polimórficas, tal como lo especifica el patrón *Strategy*.

De esta manera ha quedado resuelto tanto el problema de proliferación de objetos de estrategias como el de la cantidad de referencias por parte de los objetos polígono. A continuación se muestra un diagrama de instancias que describe un caso particular de lo expuesto, y un diagrama de interacción que permite observar el proceso de intersección entre dos polígonos.

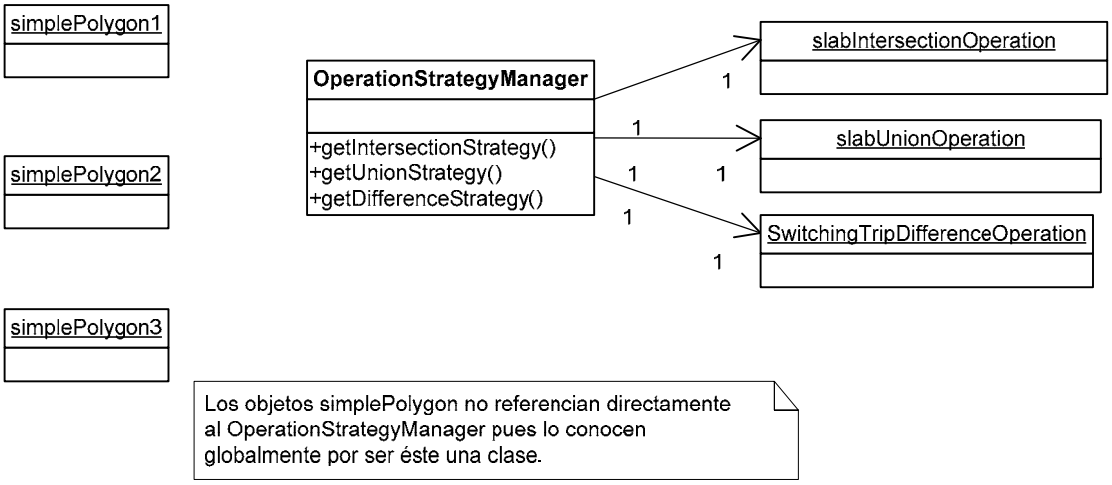



**Figura 7.6**.- Centralización de Estrategias de Operación.

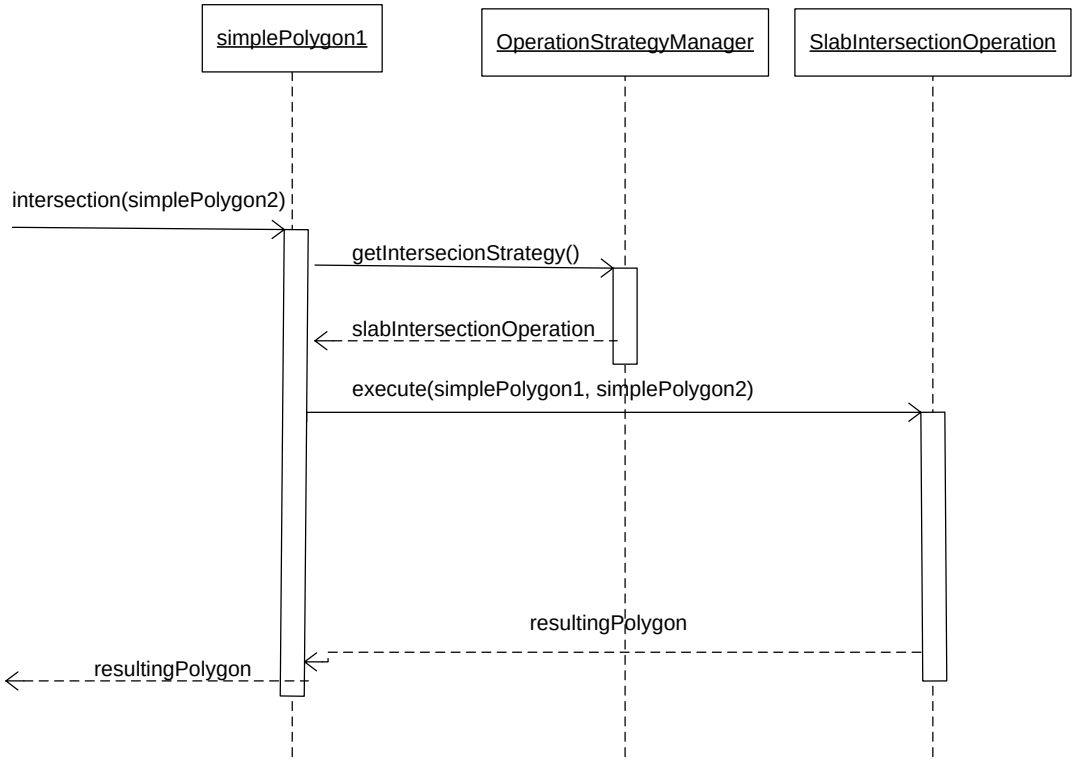



**Figura 7.7**.- Ejecución de una operación.

### 7.1.2. Revisión de las jerarquías de operaciones

Anteriormente se ha visto una jerarquía de estrategias para la operación de intersección, pero también se ha mencionado la necesidad de jerarquías similares para otras operaciones, por ejemplo para la unión y diferencia. Esto básicamente presenta el problema de la repetición de jerarquías muy similares: de hecho, cada estrategia (Slab, Switching Trip) se repite en la jerarquía de cada operación. El problema reside en la repetición de comportamiento común (y su consecuente problema de flexibilidad en cuanto a mantenimiento y extensión), ya que estas estrategias varían muy poco de una operación a otra. Por ejemplo, a continuación se muestra el pseudocódigo del algoritmo de slabs para intersección, unión y diferencia. Estos algoritmos constituirían el código de la clase SlabIntersectionOperation, SlabUnionOperation y SlabDifferenceOperation respectivamente; puede observarse que ambos varían en un solo paso, éste se marca en negrita.

1- Obtener los vértices de los dos polígonos
2- Obtener los puntos de intersección de los dos polígonos
3- Generar líneas horizontales que pasen por cada uno de los puntos obtenidos.
4- Seccionar a los polígonos mediante las líneas obtenidas, de manera de generar franjas de trapecios.
5- Por cada franja:
    a. **[Hacer la intersección, unión o diferencia]** entre los trapecios de un polígono y los del otro.
6- Unir los trapecios resultantes de las distintas franjas.
7- Eliminar vértices innecesarios (por encontrarse ya en medio de una línea recta.)

**Figura 7.8**.- Pseudocódigo del Agoritmo de Slabs.

Para solucionar este problema se propone un cambio de perspectiva: no clasificar por operaciones (unión, intersección), sino clasificar por estrategias (Slabs, Switching Trip). De esta manera seguiremos teniendo varias jerarquías, ahora una por estrategia, pero haciendo uso del patrón de diseño *TemplateMethod* [Gamma] podremos abstraer el comportamiento común a cada una en una superclase (template method) y dejar que sus subclases (las operaciones implementadas mediante esa estrategia) definan sólo el paso que necesitan (hook method). En la figura siguiente se muestra esto para el caso de la estrategia de Slabs para las operaciones de intersección, unión y diferencia.

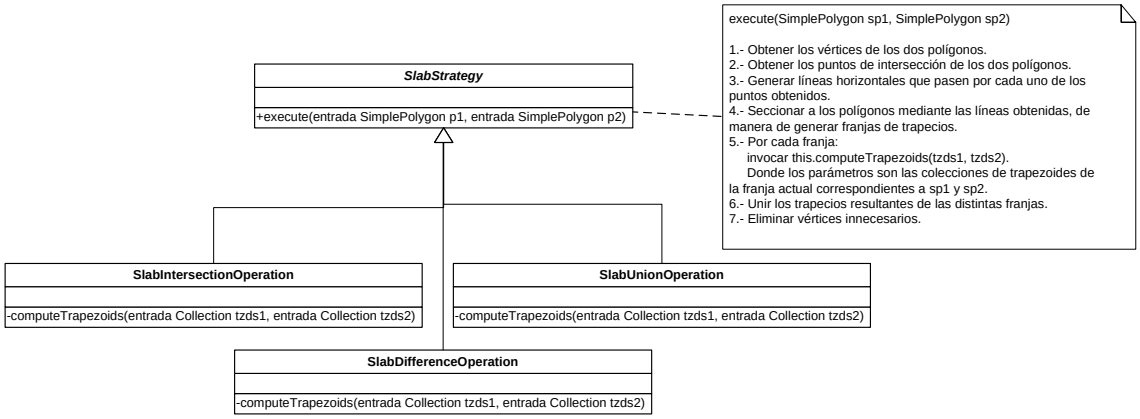



**Figura 7.9**.- Template Method de estrategia de slabs.

De manera similar puede definirse una jerarquía para la estrategia mediante SwitchingTrip, en donde el paso diferente en cada subclase consiste en la elección del camino a seguir cada vez que en el recorrido se llega a un punto de intersección entre los polígonos.

Es cierto que ahora es la operación la que debe tener una subclase que la represente en cada jerarquía, pero antes el comportamiento común era muy poco o nulo y ahora es mucho mayor, lo que justifica esta elección. De todos modos, aún puede seguir depurándose este modelo de manera de eliminar estas jerarquías; esto puede conseguirse, nuevamente, mediante la utilización del patrón de diseño *Strategy*.

El patrón *Strategy* encapsula una operación en una clase de modo que una instancia de la misma pueda ser creada y configurada dinámicamente con determinados parámetros para luego ser ejecutada mediante la invocación de un mensaje polimórfico. Esto permite que el cliente pueda ignorar los aspectos concretos de cada una, lo cual facilita los intercambios. En nuestro modelo, podrían crearse clases *Strategy* cuyas responsabilidades se limiten a calcular la unión, intersección y diferencia entre dos trapecios, podrían llamarse, por ejemplo: *TrapezoidUnionStrategy*, *TrapezoidIntersectionStrategy* y *TrapezoidDifferenceStrategy*. A su vez, las tres clases deben entender un único *mensaje* (polimorfismo) que reciba como parámetro los dos trapecios con los que debe operar, podría llamarse, por ejemplo: *execute(Trapezoid t1, Trapezoid t1).* El *método* asociado a este *mensaje* sería diferente en cada clase, ya que en una calcularía la unión, en otra la intersección y en la tercera la diferencia. Con este nuevo esquema, ya no existiría una clase del algoritmo de Slabs para cada operación, sino que existiría una sola que se utilizaría para todas las operaciones ya que en cualquier momento podría ser configurada con cualquiera de las clases Strategy. De esta manera, el paso del algoritmo que difiere de una operación a otra quedaría abstraído para todas ellas, y la variación en el comportamiento dependería solamente de la clase Strategy con la que se la haya configurado.

### 7.1.3. Mejora de performance al algoritmo de Slabs

Sobre la base de las pruebas que hemos realizado, el algoritmo de Slabs ha demostrado ser muy robusto. Sin embargo, tal como se lo ha descripto presenta problemas de performance para polígonos con gran cantidad de vértices (por ejemplo, los que definen con buen detalle el contorno de un país). La causa de este problema radica en la gran cantidad de trapecios que deben crearse, puesto que cada vértice y cada punto de intersección determina una franja (slab), la que a su vez determina uno o varios trapecios. A

nivel técnico, la raíz del problema se encuentra en el *ciclo de vida* de cada objeto trapecio que se crea, y para gran cantidad de los mismos la ineficiencia es considerable.

El ciclo de vida de un objeto consta de la creación, inicialización, uso y destrucción. Tanto en la creación como en la destrucción deben manejarse aspectos que conciernen a la alocación y liberación de memoria. Para esto deben hacerse llamadas al sistema operativo, las cuales demandan un tiempo determinado. Este tiempo es despreciable e imperceptible cuando la frecuencia de creación de objetos es media o baja y el tiempo de la etapa de uso supera ampliamente al de creación o destrucción. Sin embargo, representa un problema considerable cuando debe crearse gran cantidad de objetos, usarse durante un período muy corto de tiempo y luego destruirse. En estos casos se invierte la mayor cantidad de tiempo en el manejo de memoria propio de las etapas de creación y destrucción en detrimento de la etapa de uso, que es la que realmente nos interesa. Tal es exactamente el caso de los objetos Trapecio de nuestro modelo.

Este problema se repite en muchas situaciones y en aplicaciones muy diversas: se da también en el manejo de conexiones a una base de datos para aplicaciones con muchos usuarios (por ejemplo en sitios web) y en la creación masiva y repetitiva de threads para resolver concurrentemente un problema, entre muchos otros casos. En vistas de esta recurrencia del problema, se lo ha estudiado y se ha definido una solución genérica para el mismo, lo que ha resultado en la aparición de un nuevo patrón de diseño, el *Object Pool*.

La solución propuesta por este patrón consiste en alterar el ciclo de vida de los objetos. En concreto, propone no destruir el objeto una vez utilizado sino mantenerlo en memoria de manera de poder utilizarlo nuevamente en otro momento configurándolo con otros valores. Esto evita la repetición de la etapa de creación para cada nueva utilización. Para la implementación de esta metodología debe crearse una nueva clase, el *ObjectPool*, que se encarga de mantener almacenados (referenciados) un buen número de objetos, que deben ser creados en un momento previo a la utilización. En nuestro caso particular tenemos un *TrapezoidPool* que almacena un conjunto de objetos Trapecio; la clase TrapezoidPool provee dos mensajes: *getTrapezoid*() y *retrieveTrapezoid(Trapezoid t)*, el primero sirve para obtener un trapecio cualquiera de los que el pool ya tiene creados y almacenados, el segundo sirve para devolver el trapecio al pool luego de haberlo utilizado, de manera tal que pueda ser reutilizado en otra oportunidad. De esta manera, la secuencia que antes seguía el algoritmo de Slabs (crear trapecios, usarlos, destruirlos) se convierte ahora a: pedir trapecios al pool, usarlos, devolverlos al pool.

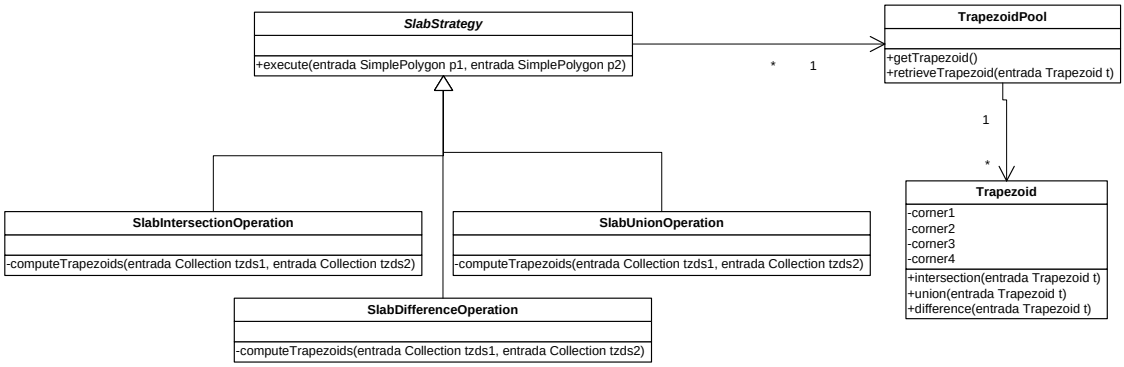



**Figura 7.10**.- Utilización de un Pool en algoritmos de Slab.

## 7.2. Implementación de Operaciones de Geometría Computacional e Interfaz Gráfica de Prueba

En esta sección se describe una aplicación gráfica construída y utilizada para la prueba de la implementación de los algoritmos de Geometría Computacional utilizados para las operaciones geométricas, tema descripto en este capítulo. Esta aplicación ha sido desarrollada por los autores de este trabajo con el único fin de testear y mostrar el funcionamiento de tales operaciones. También cabe destacar que los algoritmos y estructuras se encuentran desacoplados de la interfaz gráfica, de esta manera se alcanza la suficiente flexibilidad para poder reutilizarlos en otras aplicaciones más complejas.

No sólo se mostrará la aplicación de algunas operaciones sobre elementos y sus resultados, sino que también se mostrarán algunos pasos intermedios del algoritmo, de manera de tener una buena

apreciación práctica de los conceptos teóricos expuestos en el capítulo como así también una idea de ciertas magnitudes y complejidades.

En este sentido, se expondrán cinco ejemplos de operaciones entre dos polígonos utilizando el algoritmo de Slabs, cuyos pormenores ya han sido descriptos. Se recuerda brevemente que tal algoritmo es de tipo “Divide and Conquer” en el cual se divide a los polígonos en varias franjas formándose así trapezoides en cada una de ellas (figuras de forma más restringida sobre las que es más fácil operar); posteriormente se opera sobre estos trapezoides en cada franja y por último se los une para obtener el resultado final.

En cada ejemplo se mostrarán los resultados de las operaciones de unión e intersección entre ellos como así también los pasos intermedios más relevantes del algoritmo: la generación de las líneas que determinan las franjas (slabs) de partición y la posterior división de los polígonos en trapezoides a partir de tales líneas.

### 7.2.1. Ejemplo 1

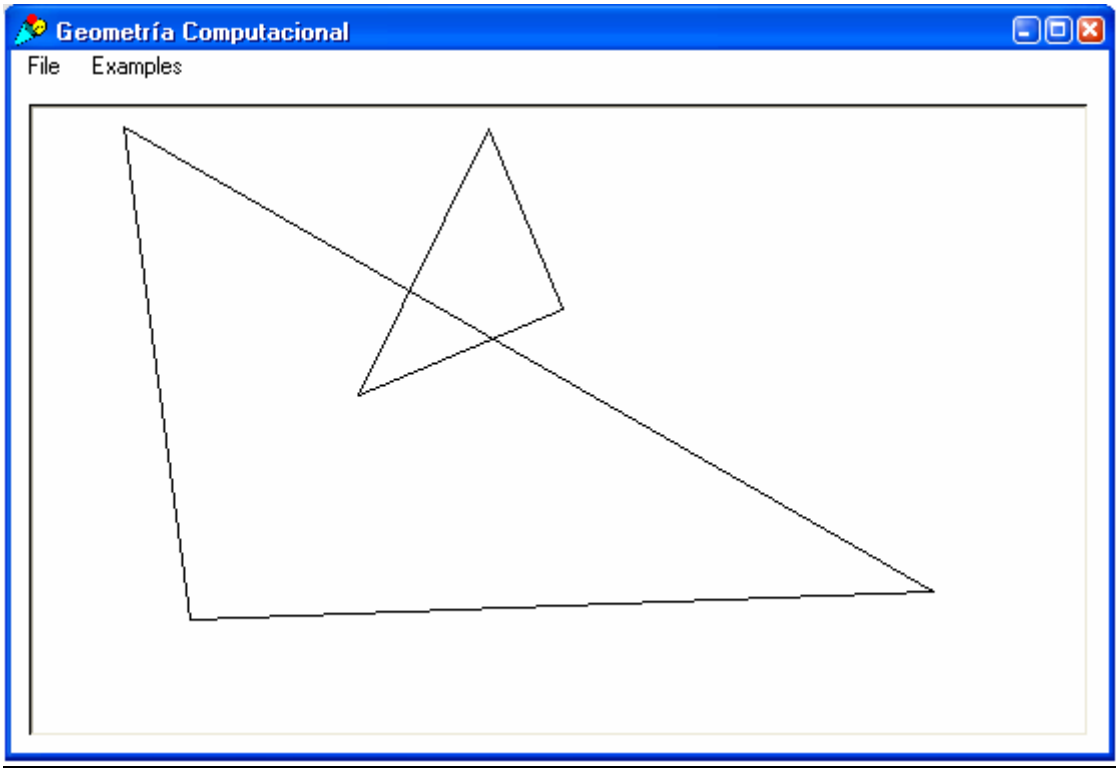



**Figura 7.11**.- Polígonos Operando.

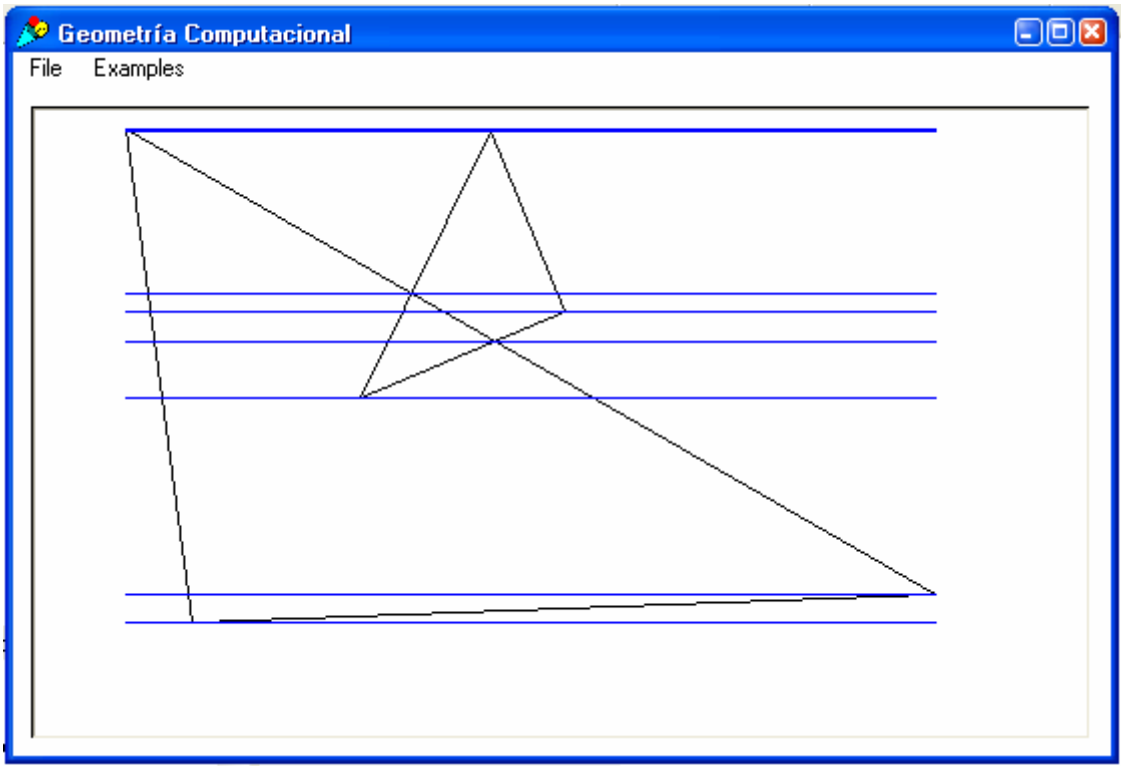



**Figura 7.12**.- Líneas de Slabs Generadas.

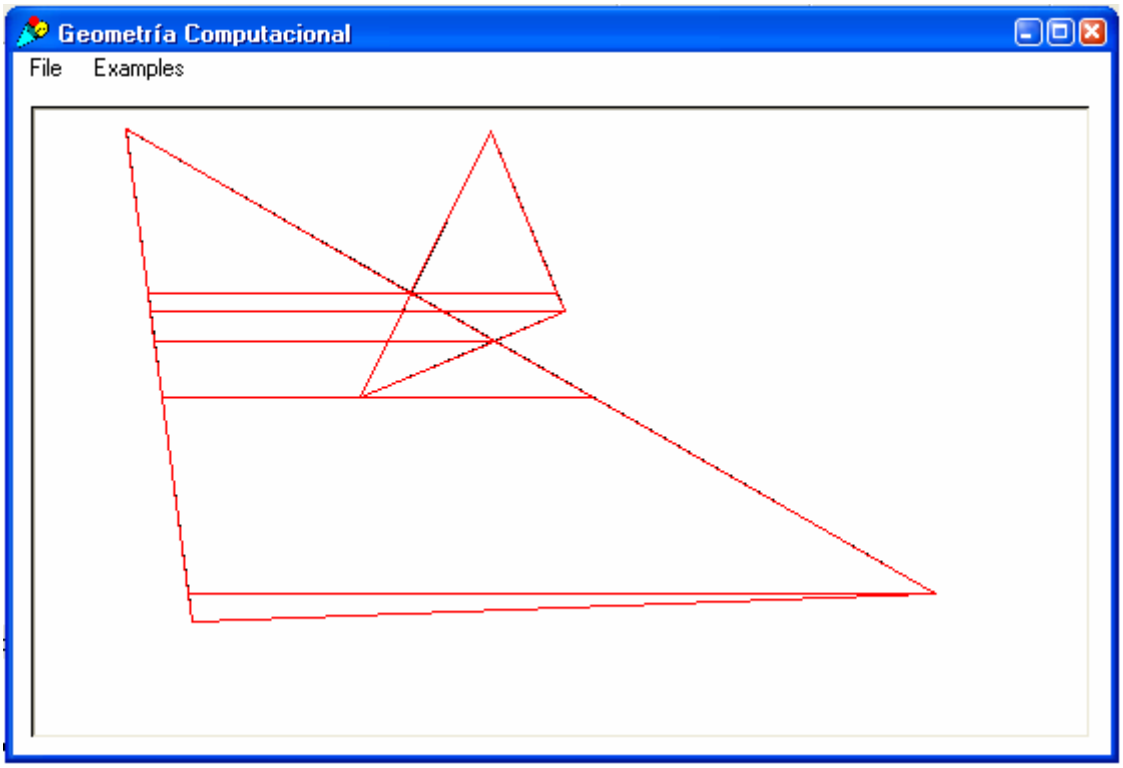



**Figura 7.13**.- Trapezoides Generados.

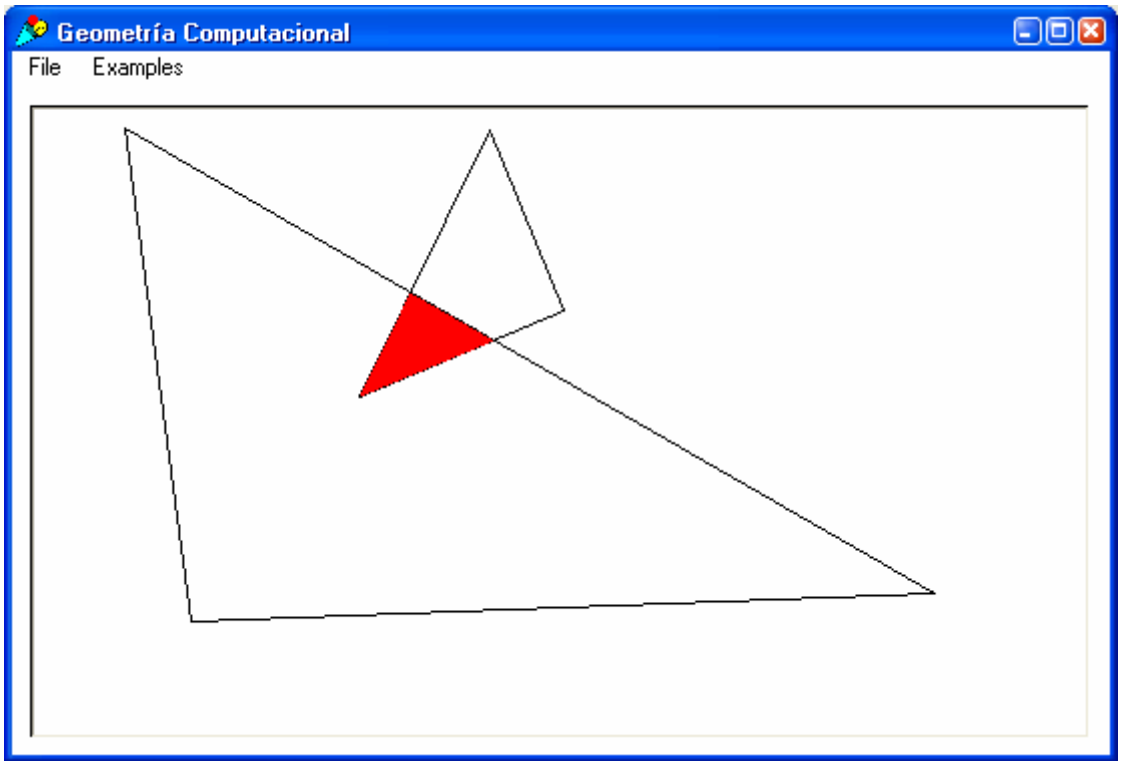



**Figura 7.14**.- Intersección.

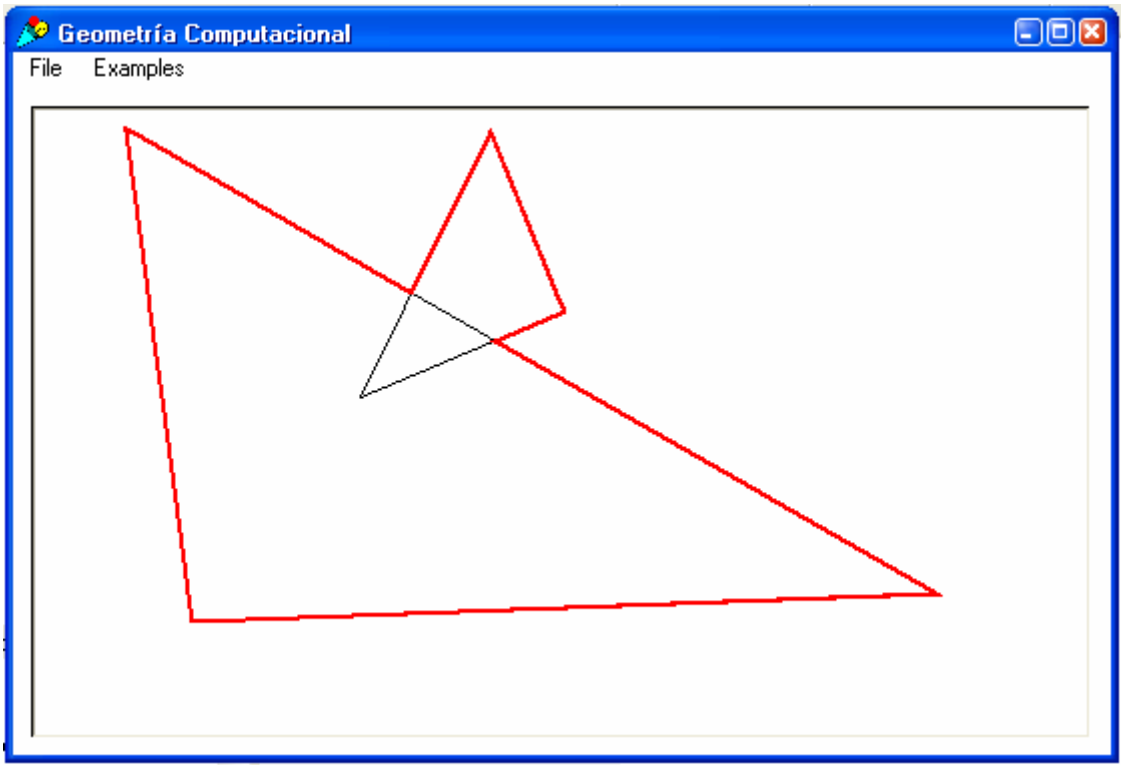



**Figura 7.15**.- Unión.

### 7.2.2. Ejemplo 2

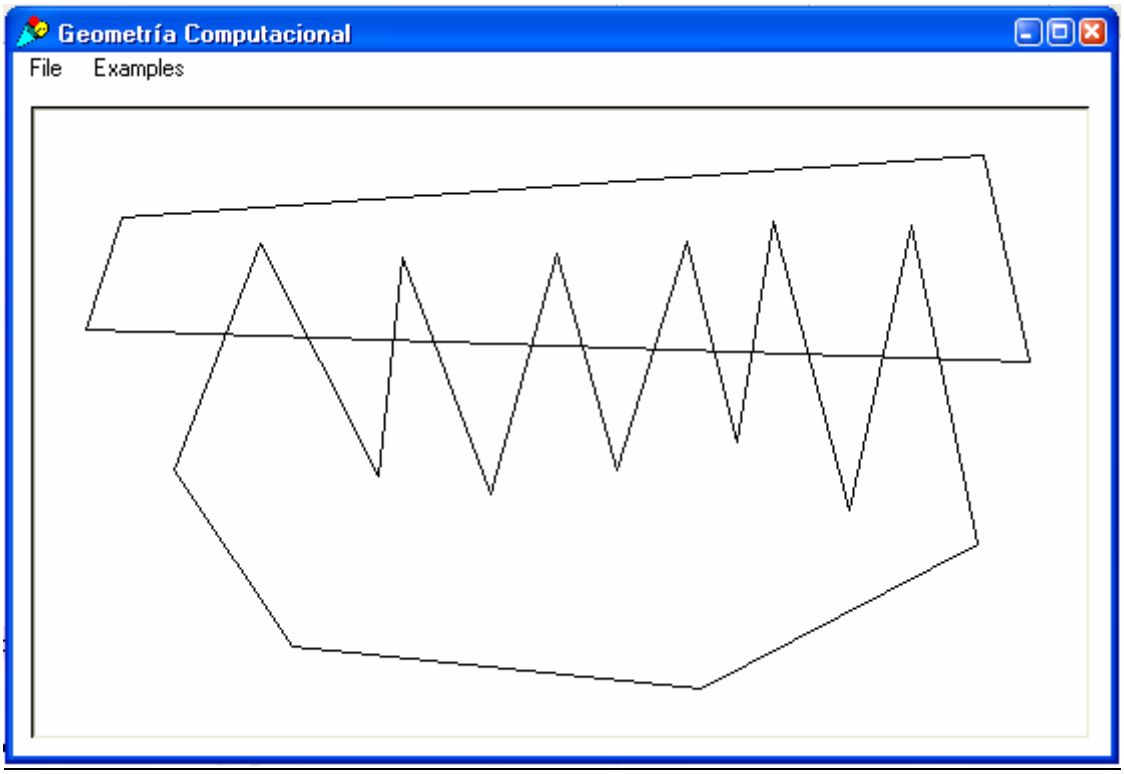



**Figura 7.16**.- Polígonos Operando.

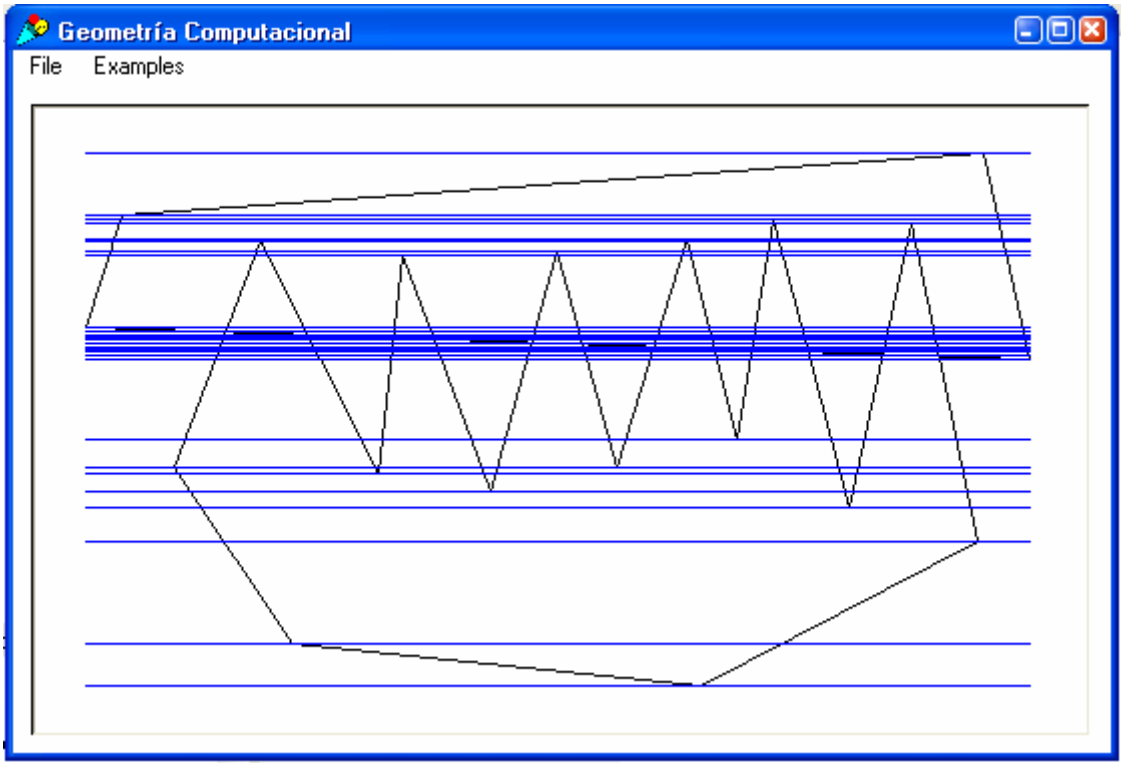



**Figura 7.17**.- Líneas de Slabs Generadas.

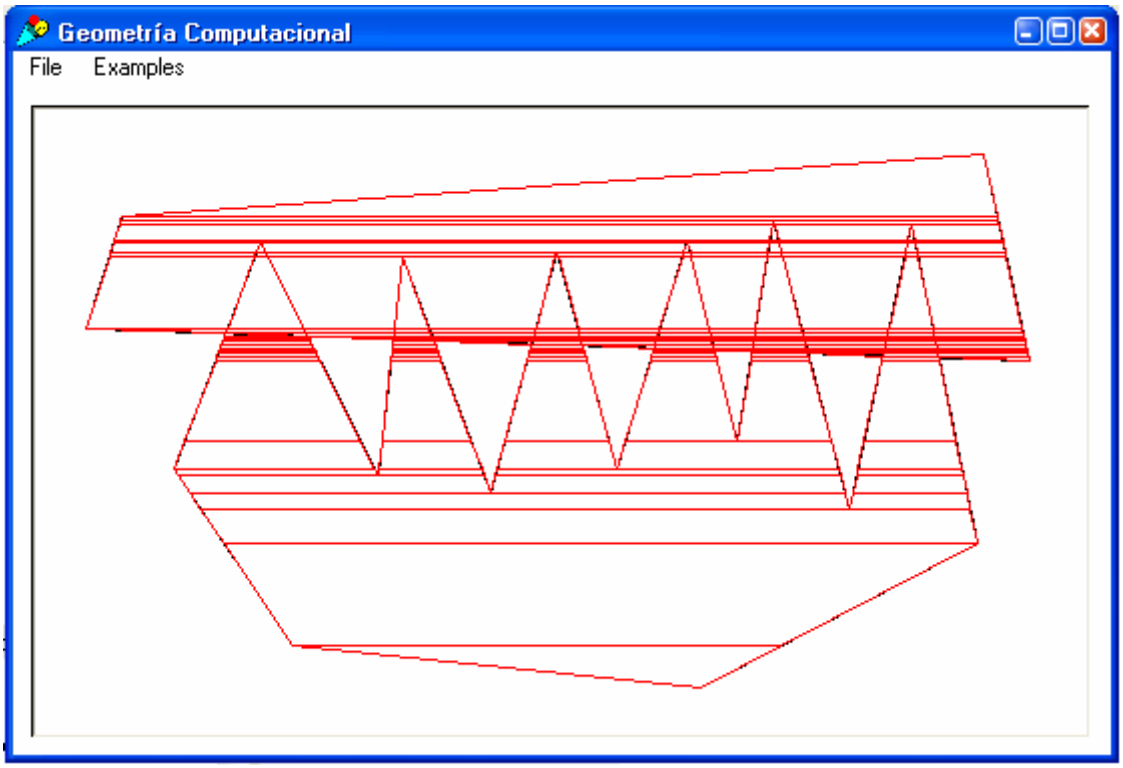



**Figura 7.18**.- Trapezoides Generados.

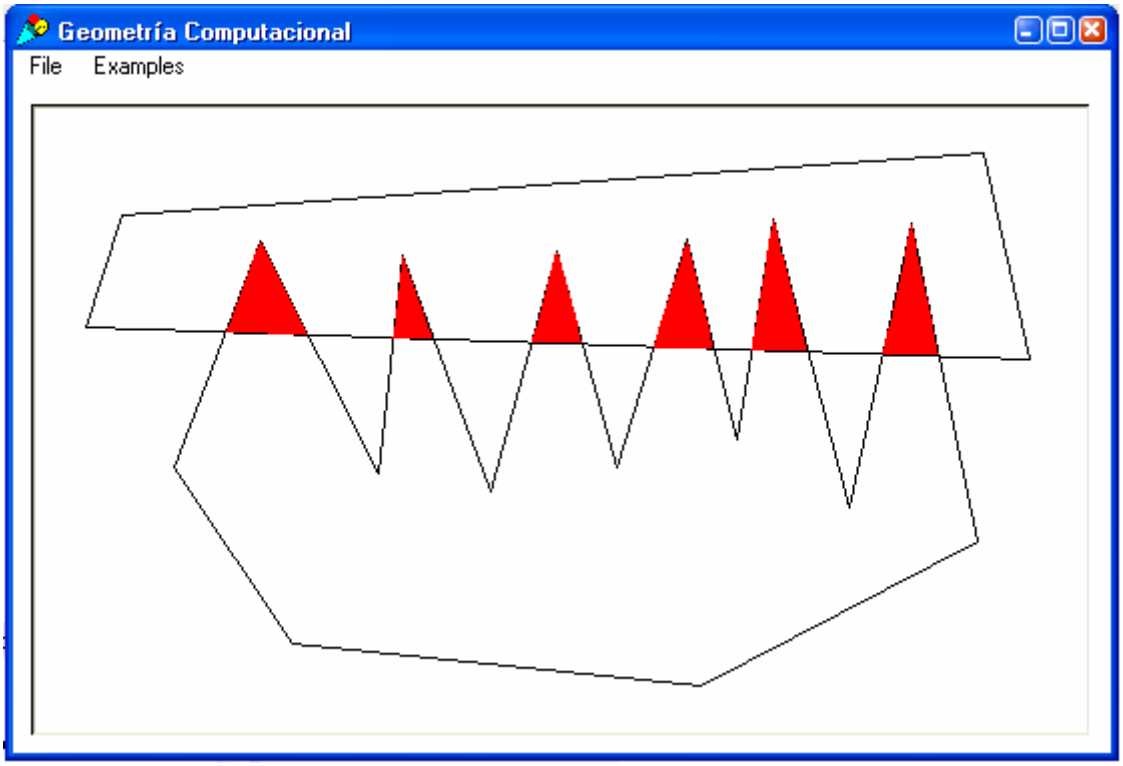



**Figura 7.19**.- Intersección.

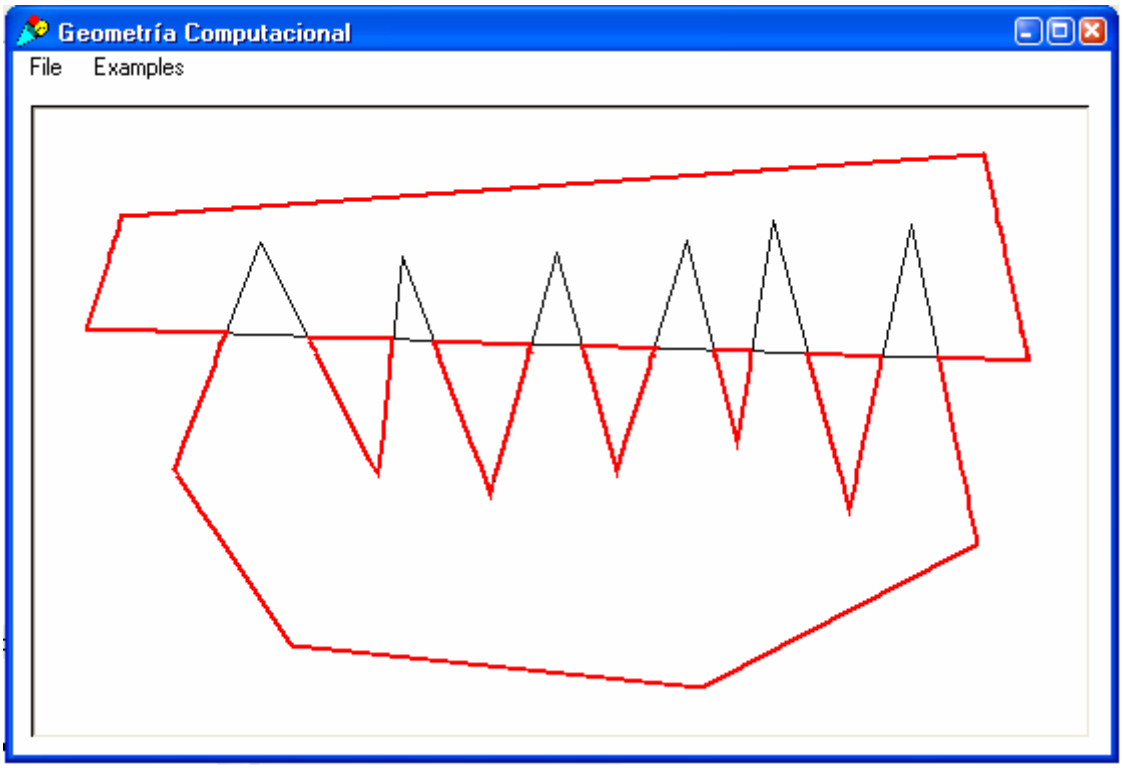



**Figura 7.20**.- Unión.

### 7.2.3. Ejemplo 3

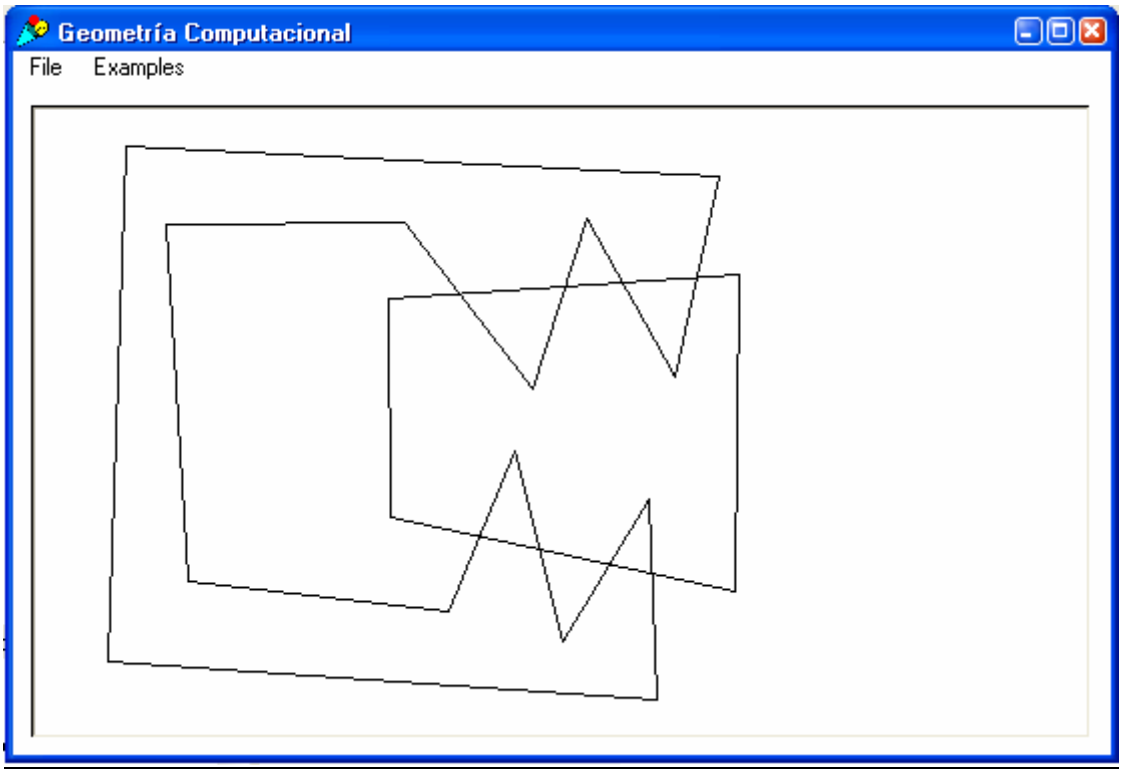



**Figura 7.21**.- Polígonos Operando.

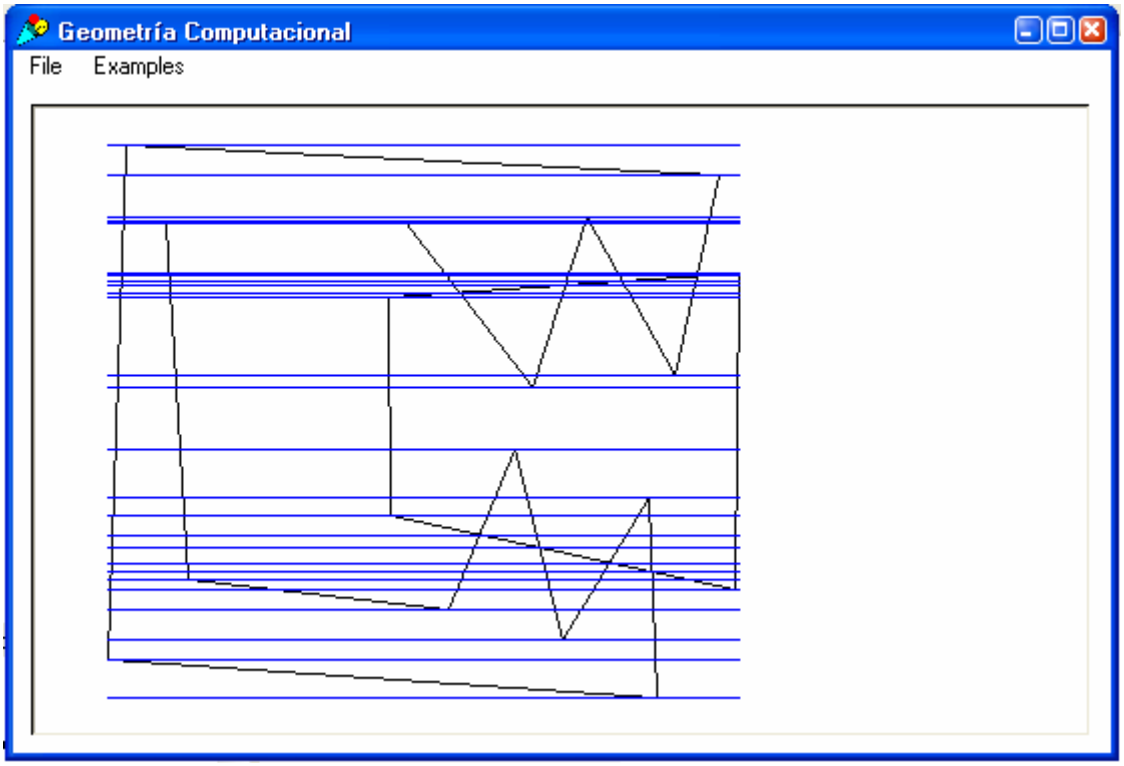



**Figura 7.22**.- Líneas de Slabs Generadas.

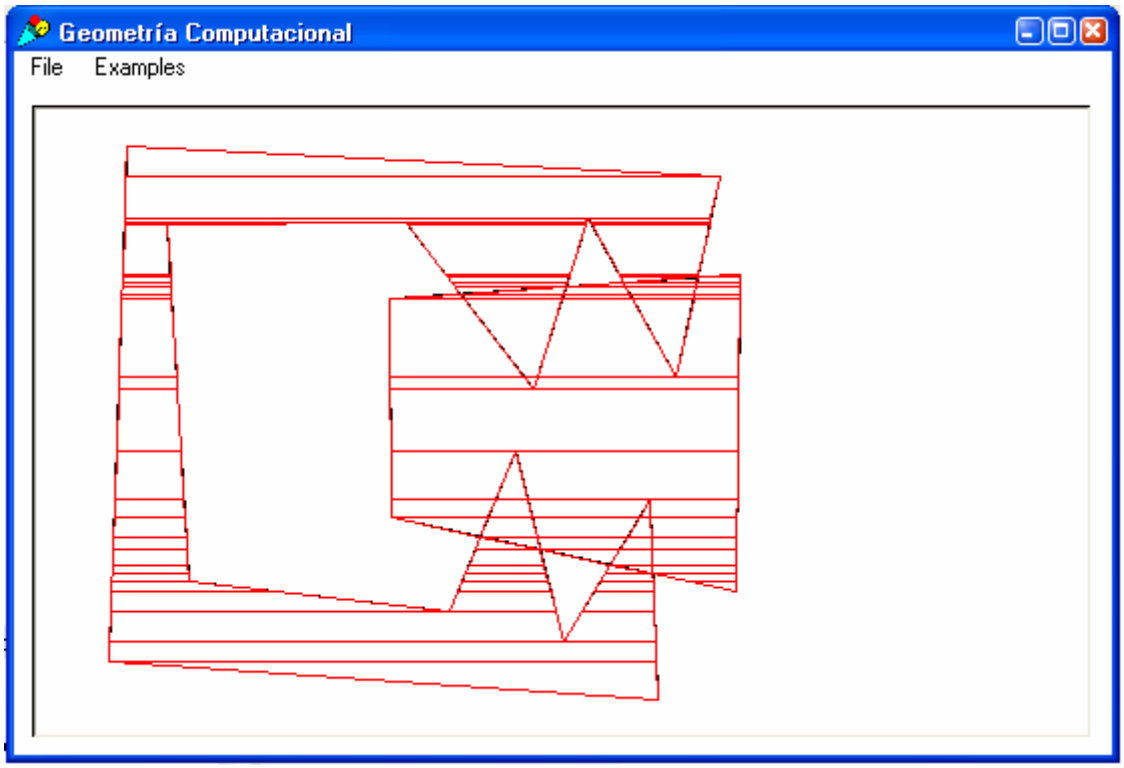



**Figura 7.23**.- Trapezoides Generados.

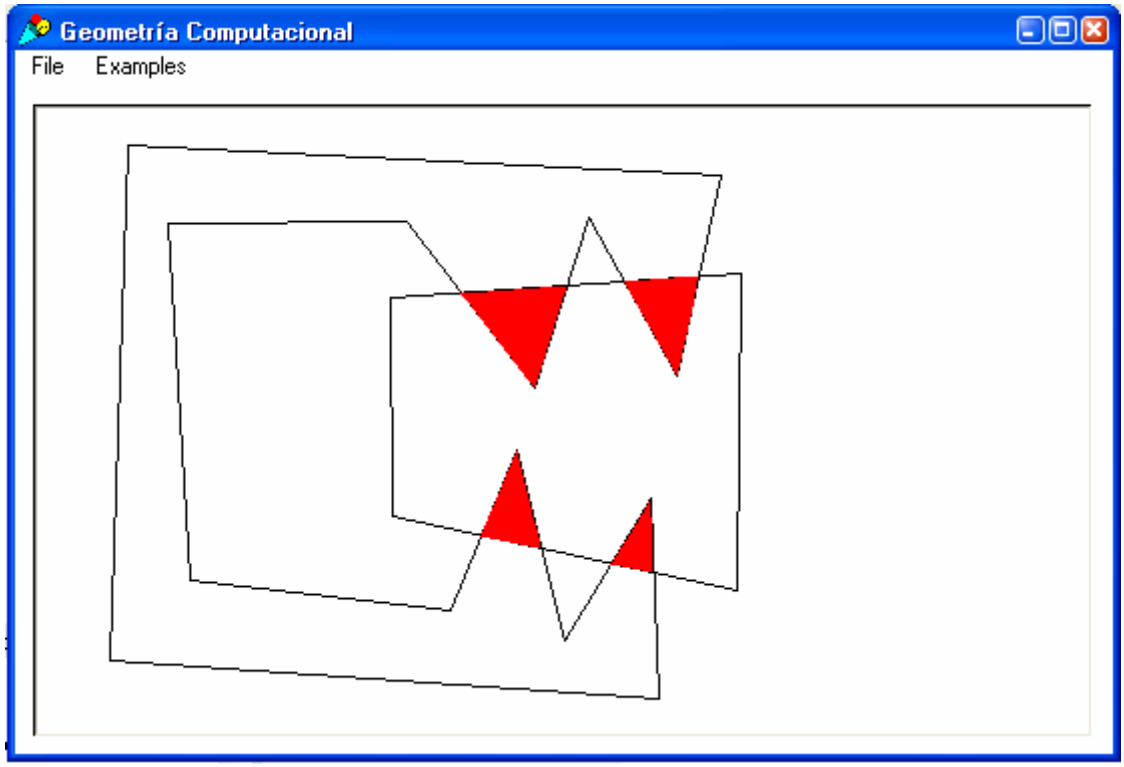



**Figura 7.24**.- Intersección.

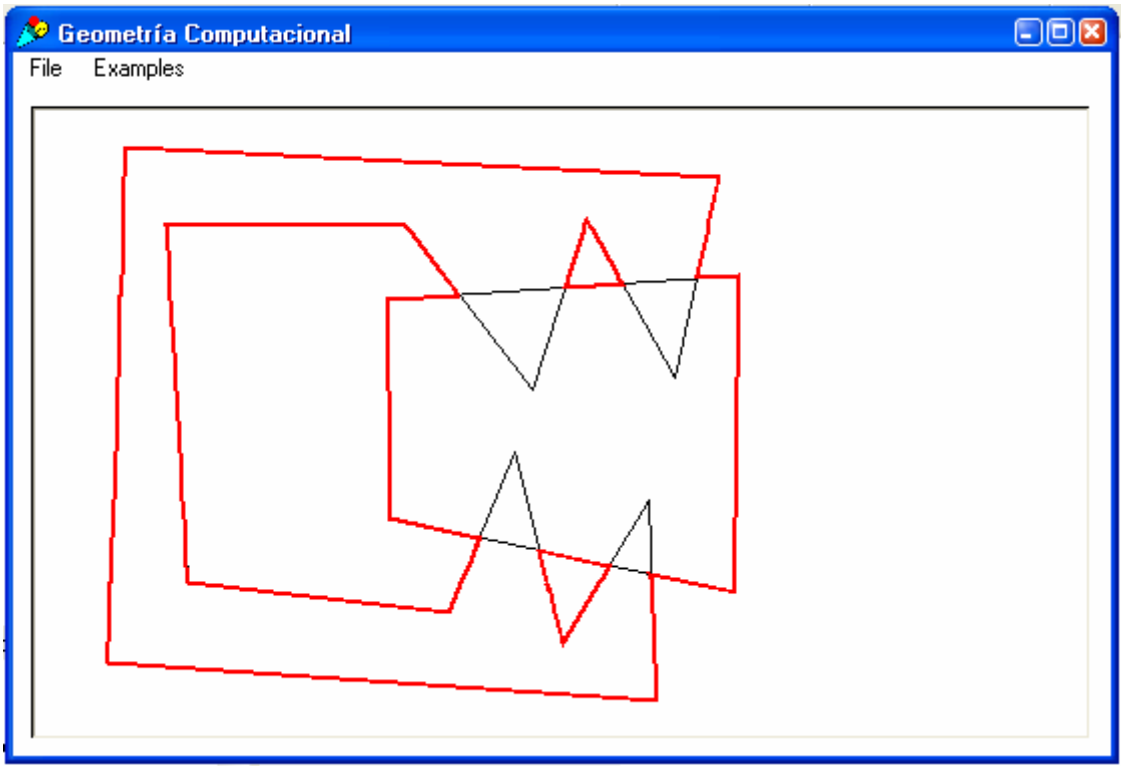



**Figura 7.25**.- Unión.

### 7.2.4. Ejemplo 4

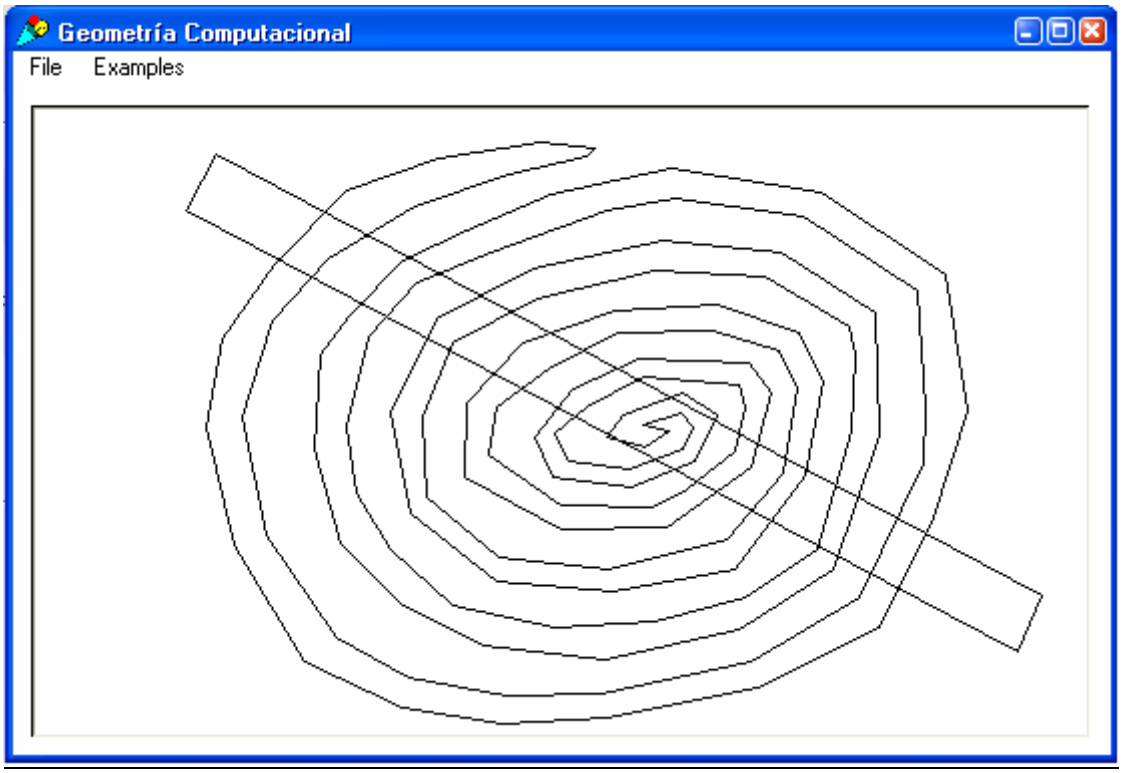



**Figura 7.26**.- Polígonos Operando.

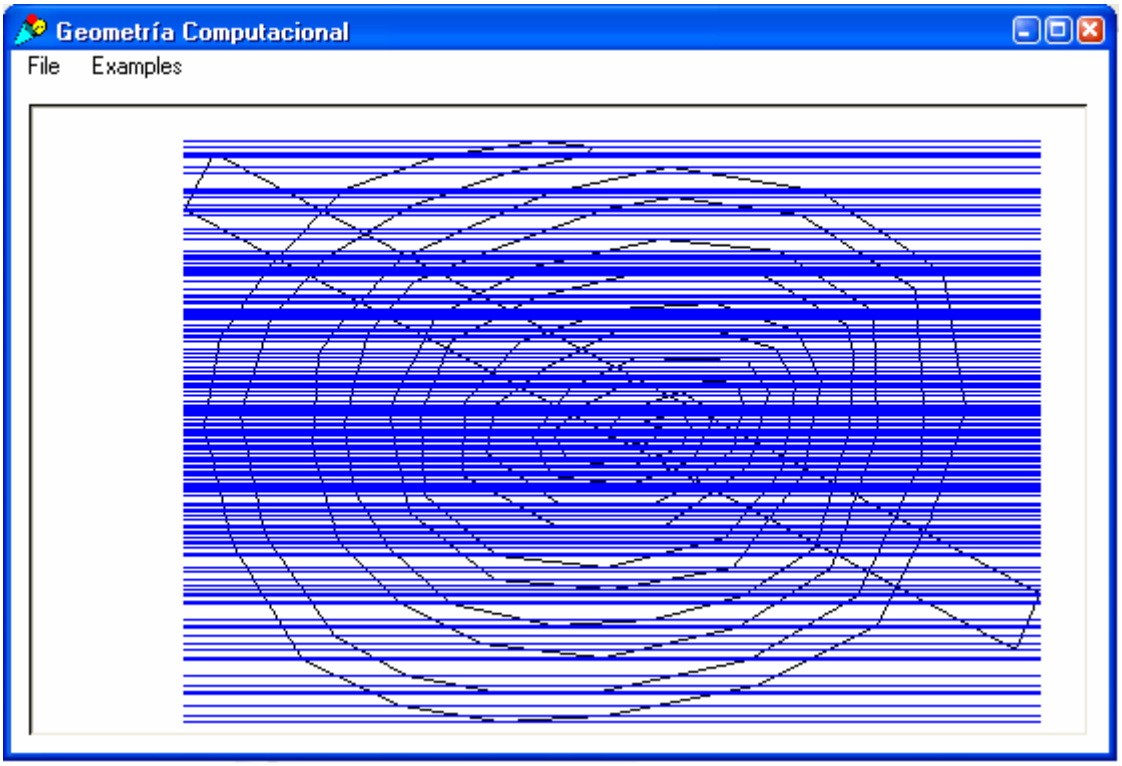



**Figura 7.27**.- Líneas de Slabs Generadas.

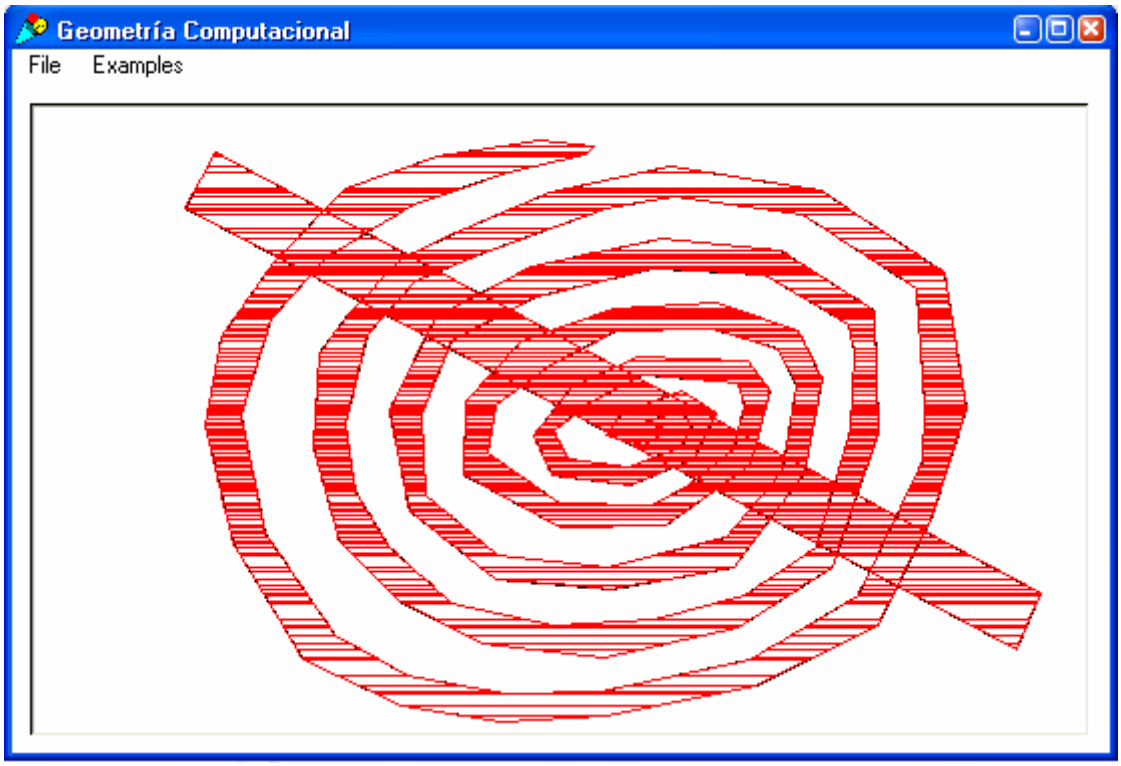



**Figura 7.28**.- Trapezoides Generados.

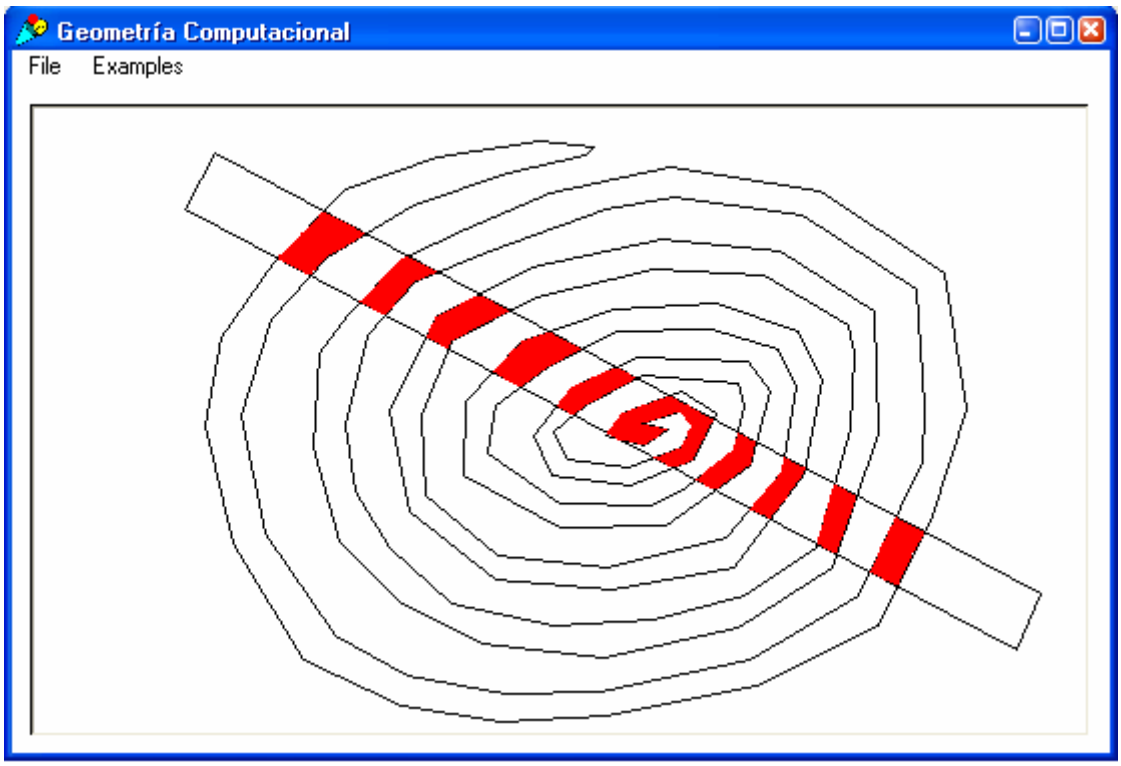



**Figura 7.29**.- Intersección.

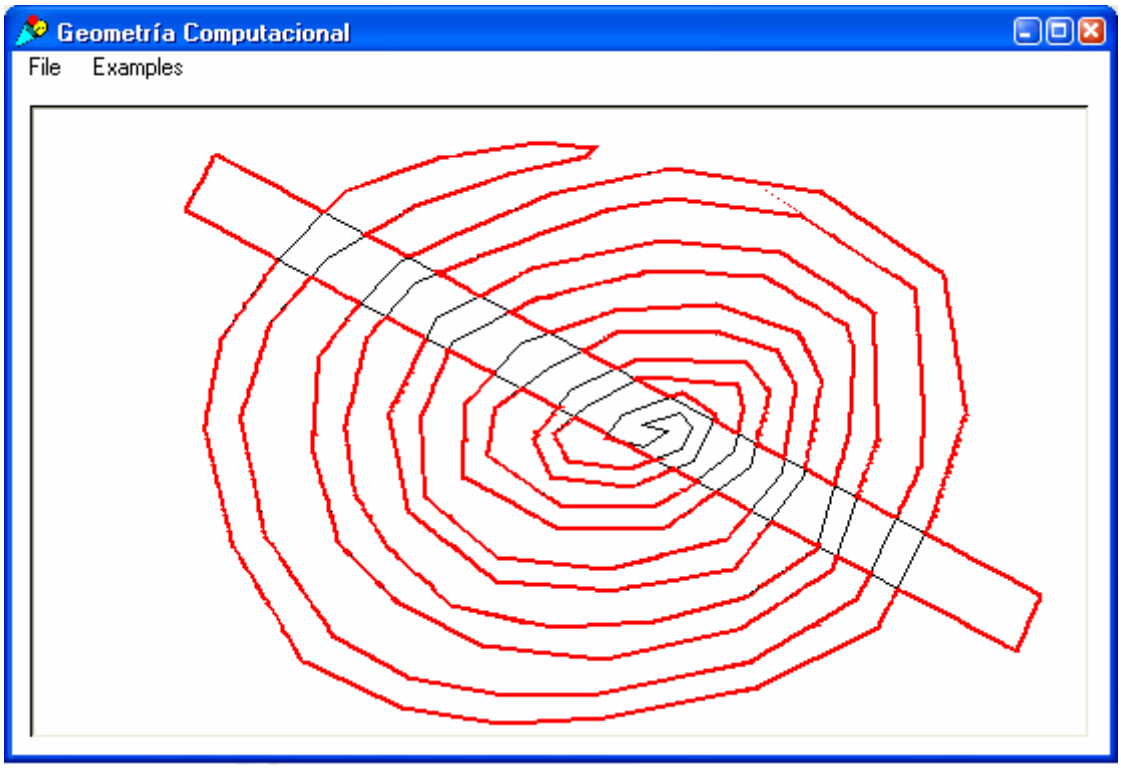



**Figura 7.30**.- Unión.

### 7.2.5. Ejemplo 5

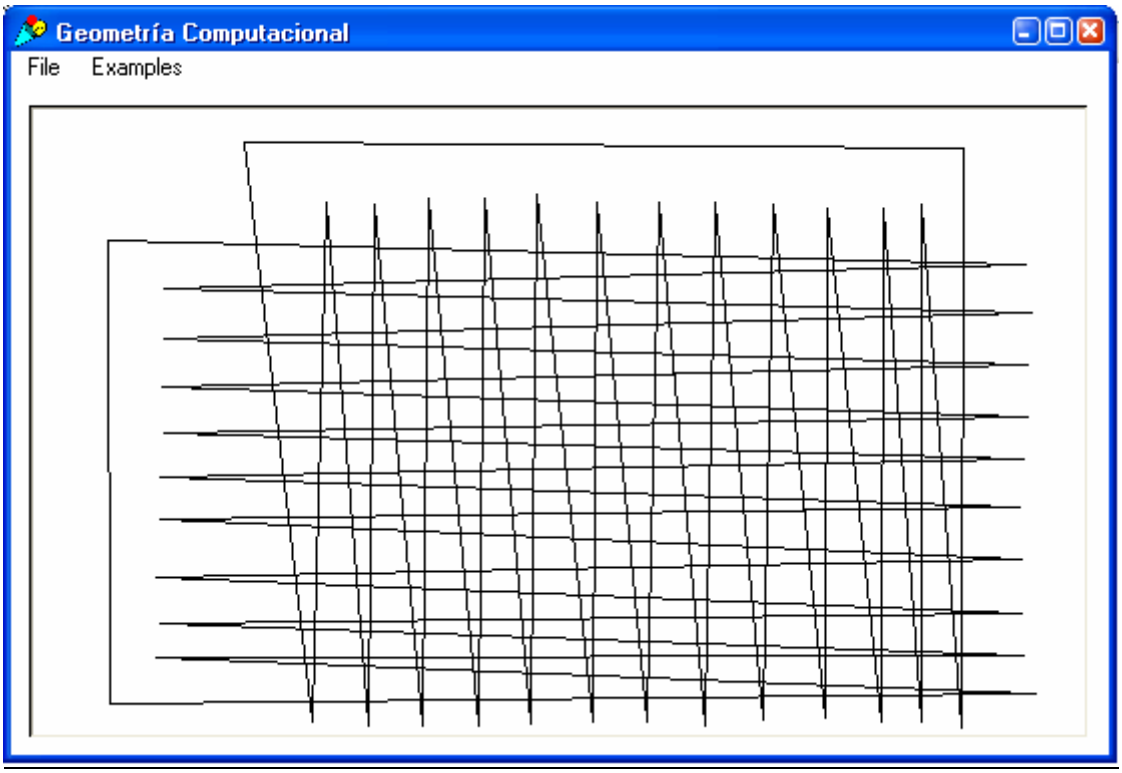



**Figura 7.31**.- Polígonos Operando.

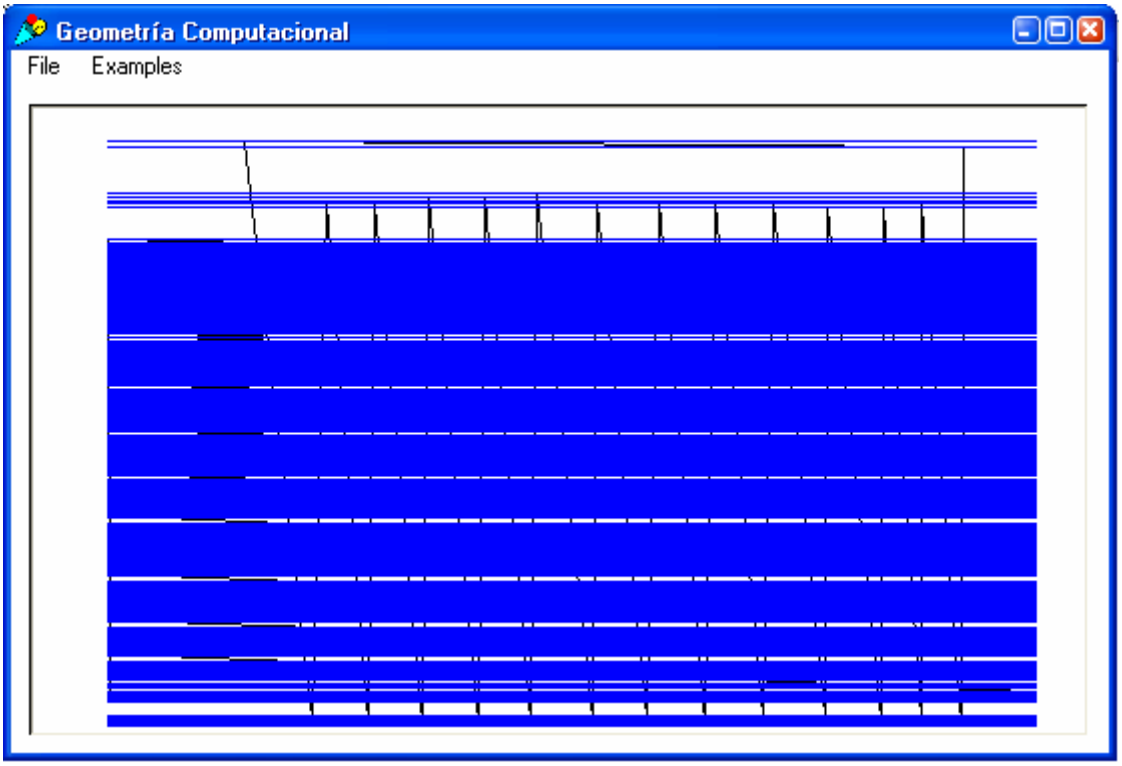



**Figura 7.32**.- Líneas de Slabs Generadas.

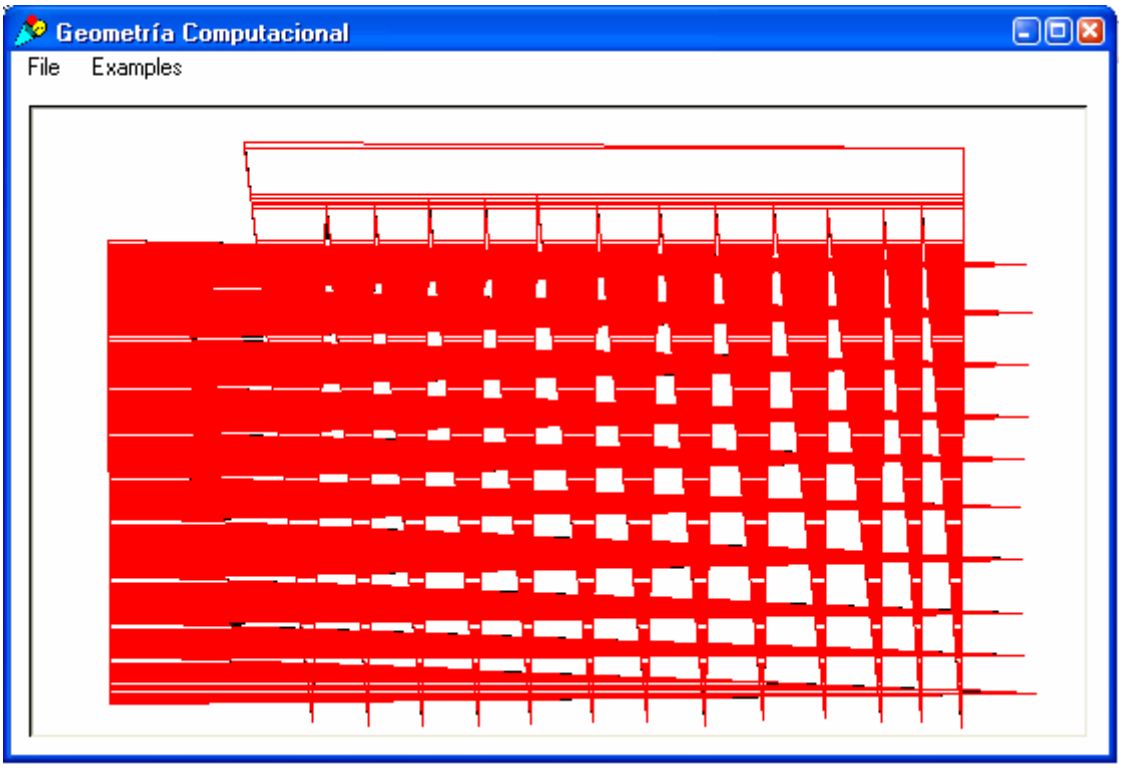



**Figura 7.33**.- Trapezoides Generados.

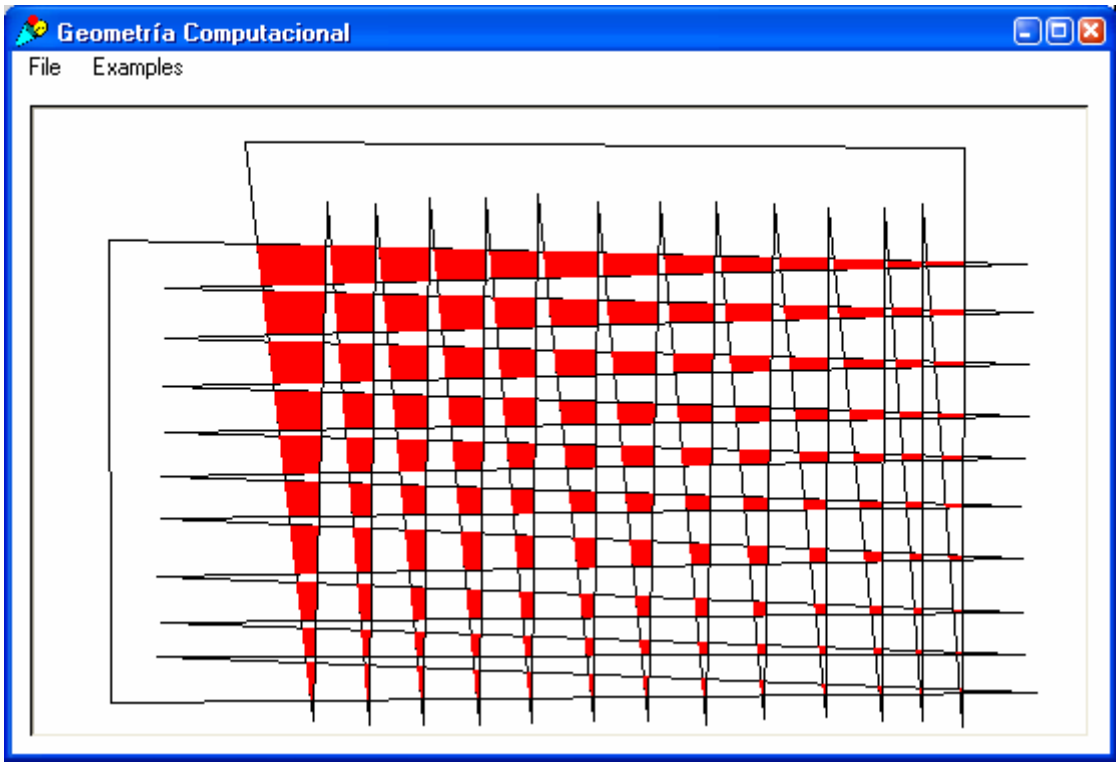



**Figura 7.34**.- Intersección.

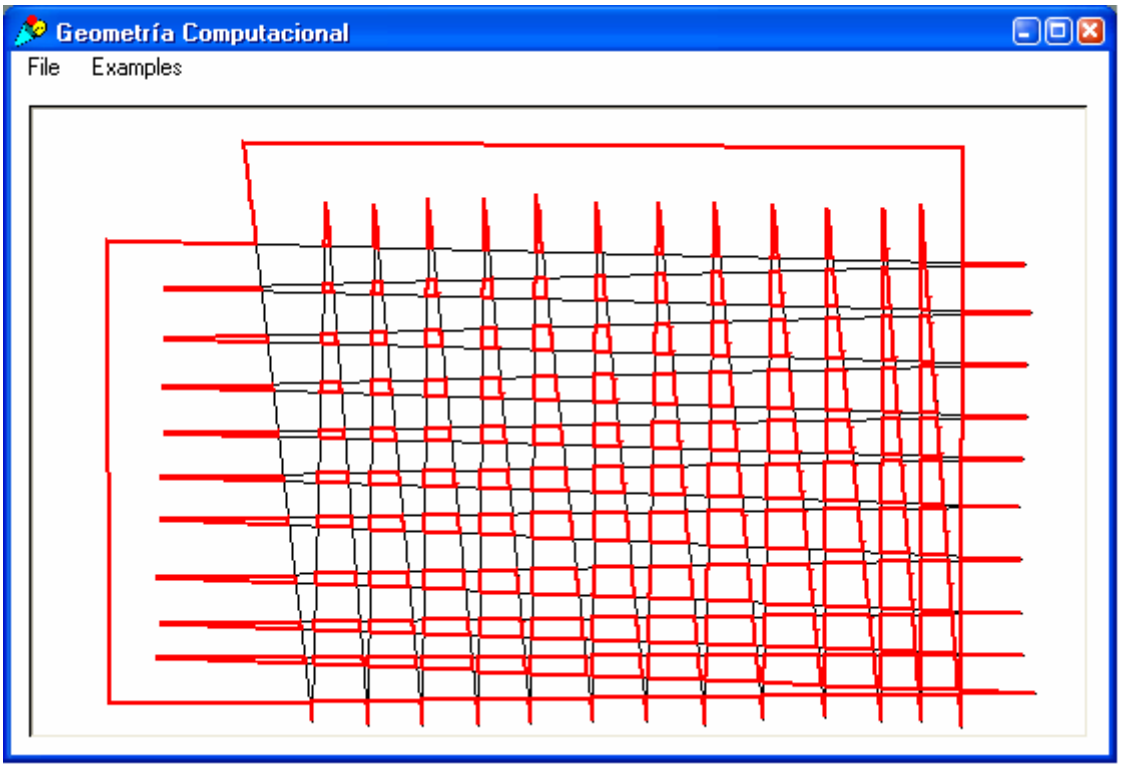



**Figura 7.35**.- Unión.

# 8. CAPÍTULO VIII: IMPLEMENTACIÓN DE LA TOPOLOGÍA Y OPERACIONES GEOMÉTRICAS

*En este capítulo se describe una aplicación gráfica construida y utilizada como prueba para la implementación del diseño presentado para la Topología de los geoObjetos, ya descripto anteriormente. Esta aplicación ha sido desarrollada por los autores de este trabajo con el único fin de testear y mostrar la estructura diseñada y presentada.*

*Cabe destacar que la estructura se encuentra totalmente desacoplada de la interfaz gráfica, de esta manera se alcanza la suficiente flexibilidad para poder reutilizar la misma estructura en otras aplicaciones más complejas. Para desacoplar la información que necesitan los elementos topológicos para ser visualizados se utiliza el patrón de diseño* ***Appearance****, detallado en las publicaciones realizadas por los autores. Una descripción de este patrón de diseño se anexa en el apéndice C.*

## 8.1. Descripción de la Aplicación

A continuación se brinda una descripción básica de la aplicación. En particular se presentan sus interfaces principales, el agregado de elementos topológicos básicos y se describe la utilización de layers (capas temáticas).

La pantalla principal de la aplicación se muestra en la siguiente figura:

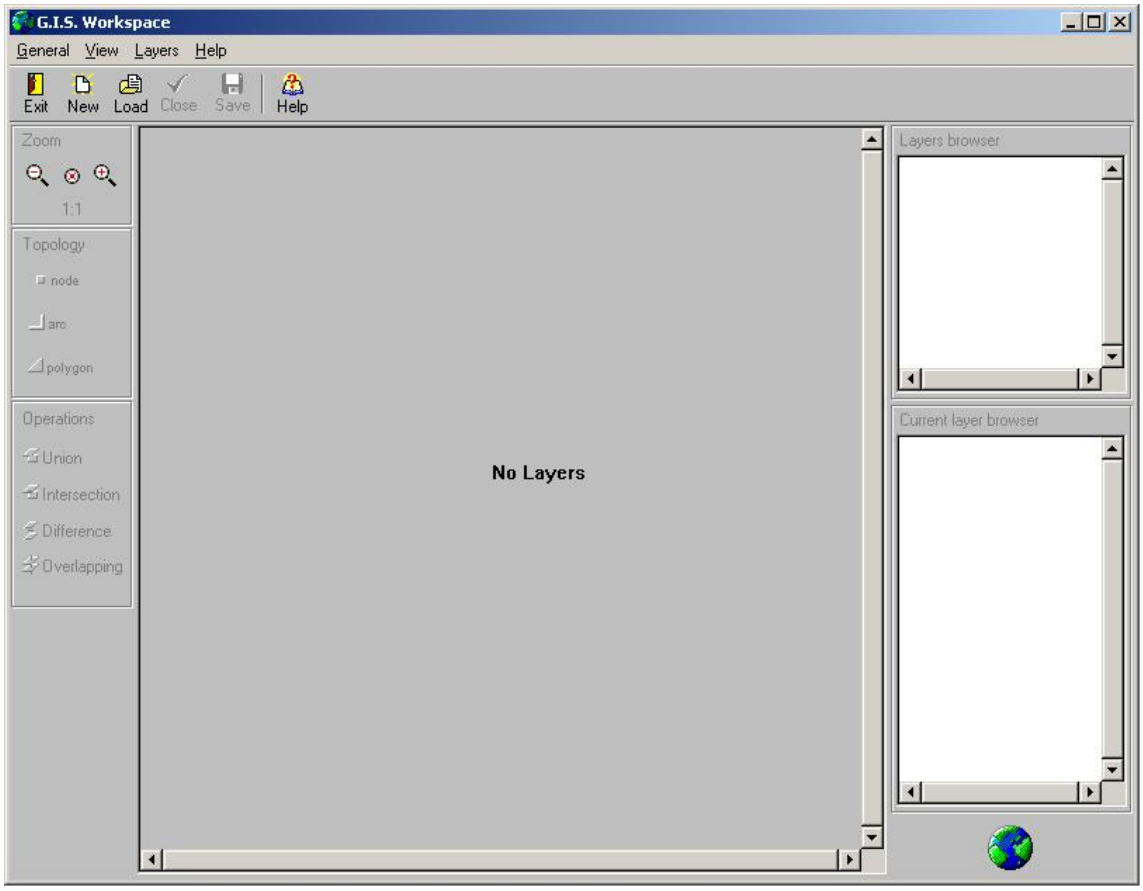



**Figura 8.1**.- Aplicación de Gestión de la Topología

La aplicación está compuesta por un panel que permite al usuario manejar layers o capas, en las cuales agregan elementos correspondientes a nodos, arcos y polígonos. Una vez creado cada elemento, permite asociarle o cambiarle la apariencia visual del mismo, gracias a la aplicación del pattern Appearance, y también realizar operaciones geométricas de unión, intersección y diferencia entre ellos. Estos elementos son instancias de la clase Geometry, y se definió un único sistema de referencia para toda la aplicación, aunque como ya dijimos anteriormente se podría contar con más de un sistema de referencias.

Cada layer puede ser persistido junto con sus elementos. Esto es, que se pueden guardar como un tipo de archivo especial (.vl) en los discos físicos de la computadora, para luego poder volver a abrirlos y utilizarlos desde la aplicación.

Además, se proveen funciones de zoom para cada layer y superposición de elementos en un mismo layer.

En la parte derecha y superior de la pantalla se puede observar un componente, llamado Layers Browser, que permite seleccionar los layers que desea ver el usuario en el panel central. Desde este componente también se pueden eliminar layers, a través de un menú contextual que se accede con el botón de la derecha del Mouse.

En la parte derecha inferior de la pantalla se encuentra otro componente, llamado Current Layer Browser, que permite visualizar e interactuar con los elementos del layer seleccionado en el componente anterior. Desde este componente, a través de un menú contextual que se accede con el botón de la derecha del Mouse, se puede:

- eliminar elementos del layer
- cambiarle las propiedades de visualización al elemento (color del elemento y color al seleccionarlo, ancho de las líneas).
- cambiar el formato de su apariencia (arbitraria, literal, derivada de su posición)
- en el caso de polígonos, se puede agregar agujeros o unirlos para generar polígonos compuestos.

A continuación, en la Figura 8.2, se muestra la aplicación con distintos elementos cargados. Para esto, se creó un layer que contiene:

- un nodo, ubicado en la coordenada (x,y) = (100,100).
- un arco, con su nodo inicial ubicado en la coordenada (x,y) = (150,200) y con su nodo final ubicado en la coordenada (x,y) = (150,250)
- un polígono, con cuatro arcos:
  - un primer arco, con su nodo inicial ubicado en la coordenada (x,y) = (300,300) y con su nodo final ubicado en la coordenada (x,y) = (300,400)
  - un segundo arco, con su nodo inicial ubicado en la coordenada (x,y) = (300,400) y con su nodo final ubicado en la coordenada (x,y) = (400,400)
  - un tercer arco, con su nodo inicial ubicado en la coordenada (x,y) = (400,400) y con su nodo final ubicado en la coordenada (x,y) = (400,300)
  - un cuarto arco, con su nodo inicial ubicado en la coordenada (x,y) = (400,300) y con su nodo final ubicado en la coordenada (x,y) = (300,300)

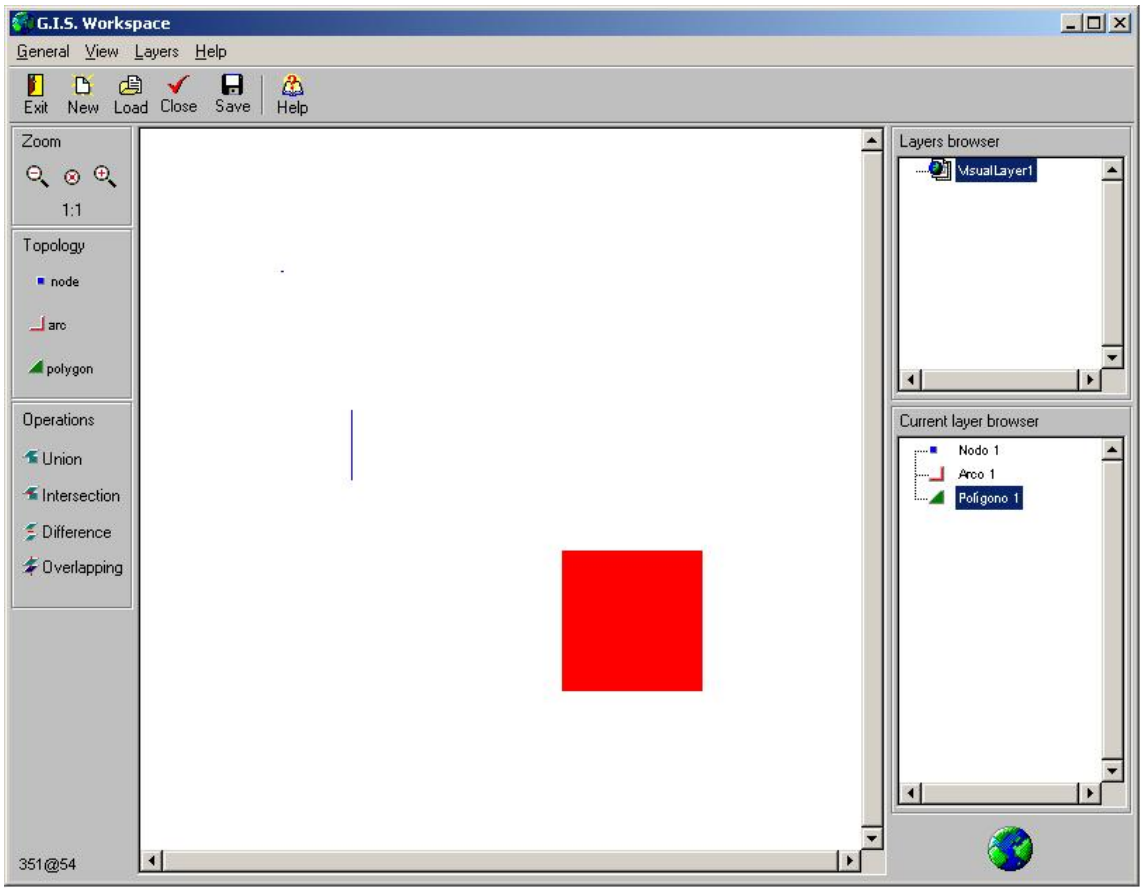



**Figura 8.2**.- Layer con tres elementos básicos de la Topología

Los elementos se agregan clickeando en el tipo de Topología, que se encuentra ubicado en la izquierda de la pantalla. De acuerdo a eso, se pueden abrir las siguientes interfaces:

- Nodo: desde la cual el usuario debe ingresar la coordenada en la cuál se ubicará el nodo y una descripción para el mismo.


**Figura 8.3**.- Agregado de nodos

- Arco: desde la cual el usuario debe ingresar la coordenada en la cuál se ubicará el nodo inicial y el nodo final, junto con una descripción.

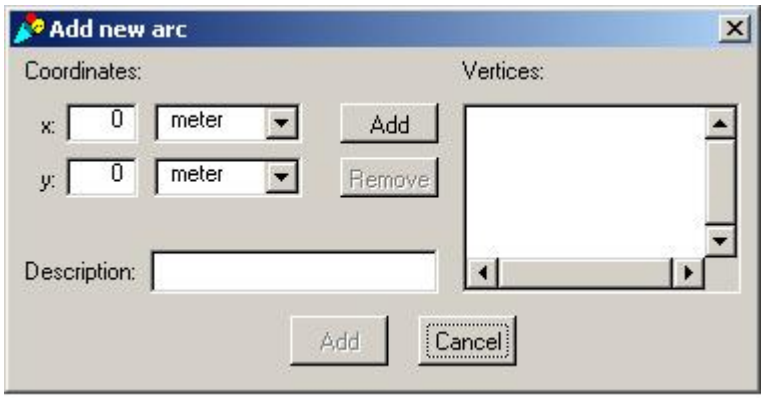



**Figura 8.4**.- Agregado de arcos

- Polígono: desde la cual el usuario debe ingresar las coordenadas correspondientes a los nodos de los arcos que lo forman, junto con una descripción. Los polígonos compuestos, ya sean disjuntos o con agujeros se forman a partir del menú contextual como indicamos anteriormente.

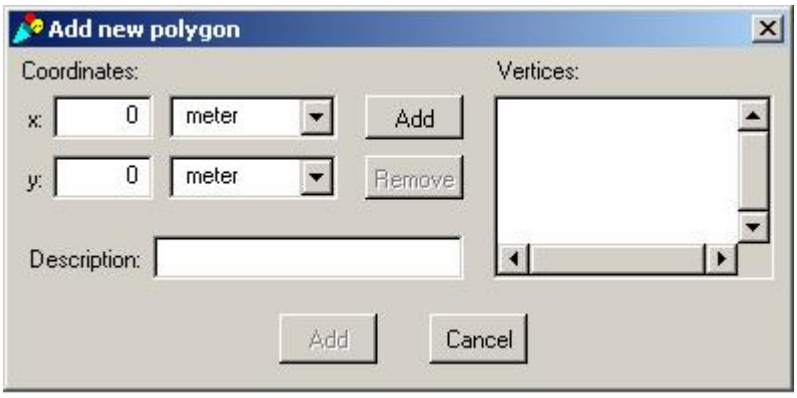



**Figura 8.5**.- Agregado de polígonos

## 8.2. Ejemplos de Operaciones

A continuación mostramos las operaciones que se pueden realizar entre los elementos de los layers. Primero utilizaremos polígonos simples, y luego se mostrarán las mismas operaciones con polígonos complejos.

Para visualizar las operaciones con polígonos simples, partimos de un layer que tiene dos rectángulos, con la siguiente topología:

- un polígono, llamado Polígono 1, con cuatro arcos:
  - un primer arco, con su nodo inicial ubicado en la coordenada (x,y) = (20,20) y con su nodo final ubicado en la coordenada (x,y) = (60,20)
  - un segundo arco, con su nodo inicial ubicado en la coordenada (x,y) = (60,20) y con su nodo final ubicado en la coordenada (x,y) = (60,40)
  - un tercer arco, con su nodo inicial ubicado en la coordenada (x,y) = (60,40) y con su nodo final ubicado en la coordenada (x,y) = (20,40)
  - un cuarto arco, con su nodo inicial ubicado en la coordenada (x,y) = (20,40) y con su nodo final ubicado en la coordenada (x,y) = (20,20)

- un polígono, llamado Polígono 2, con cuatro arcos:
  - un primer arco, con su nodo inicial ubicado en la coordenada (x,y) = (50,30) y con su nodo final ubicado en la coordenada (x,y) = (100,30)

- un segundo arco, con su nodo inicial ubicado en la coordenada (x,y) = (100,30) y con su nodo final ubicado en la coordenada (x,y) = (100,60)
- un tercer arco, con su nodo inicial ubicado en la coordenada (x,y) = (100,60) y con su nodo final ubicado en la coordenada (x,y) = (50,60)
- un cuarto arco, con su nodo inicial ubicado en la coordenada (x,y) = (50,60) y con su nodo final ubicado en la coordenada (x,y) = (50,30)

La siguiente figura muestra el layer con los dos polígonos anteriores agregados, en la cual se encuentra seleccionado el segundo polígono:

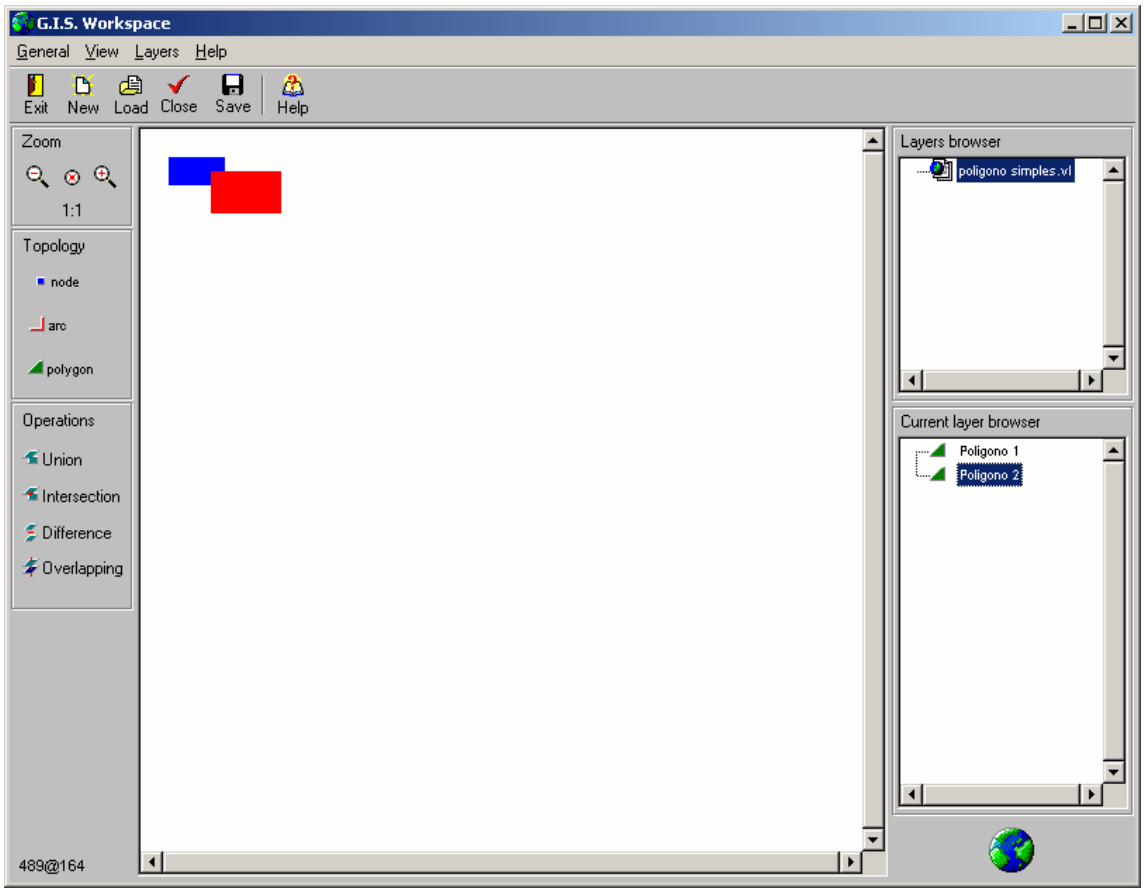



**Figura 8.6**.- Layer con polígonos simples sobre los cuales se realizarán las operaciones

Para realizar la operación de unión entre los polígonos anteriores, se realiza un click en el tipo de operación ‘Union’, ubicado a la izquierda de la pantalla. A partir de ese click se visualiza la siguiente ventana:

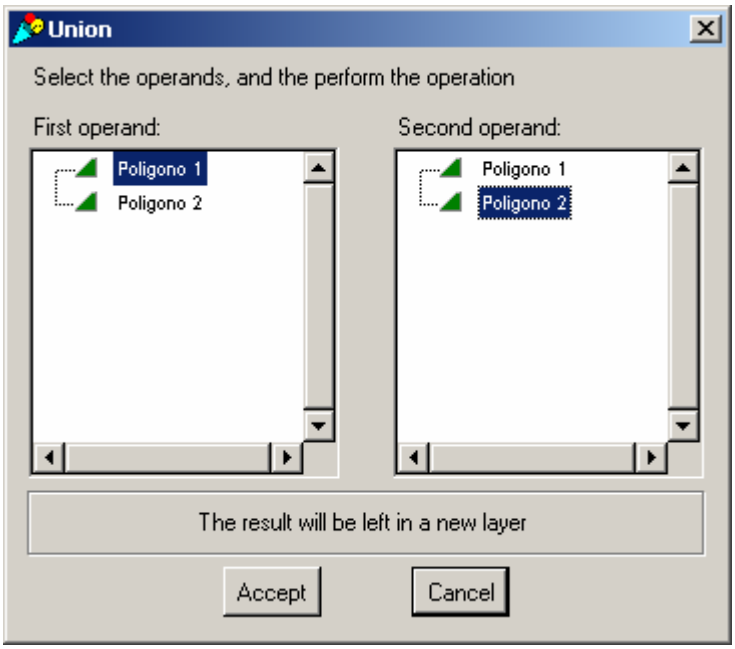



**Figura 8.7**.- Selección de polígonos a unir

Una vez visualizada la ventana anterior se seleccionan los polígonos a unir y al confirmar la operación, se crea un nuevo layer con el elemento topológico, otro polígono, resultante de la operación. La siguiente figura muestra el resultado de esta operación:

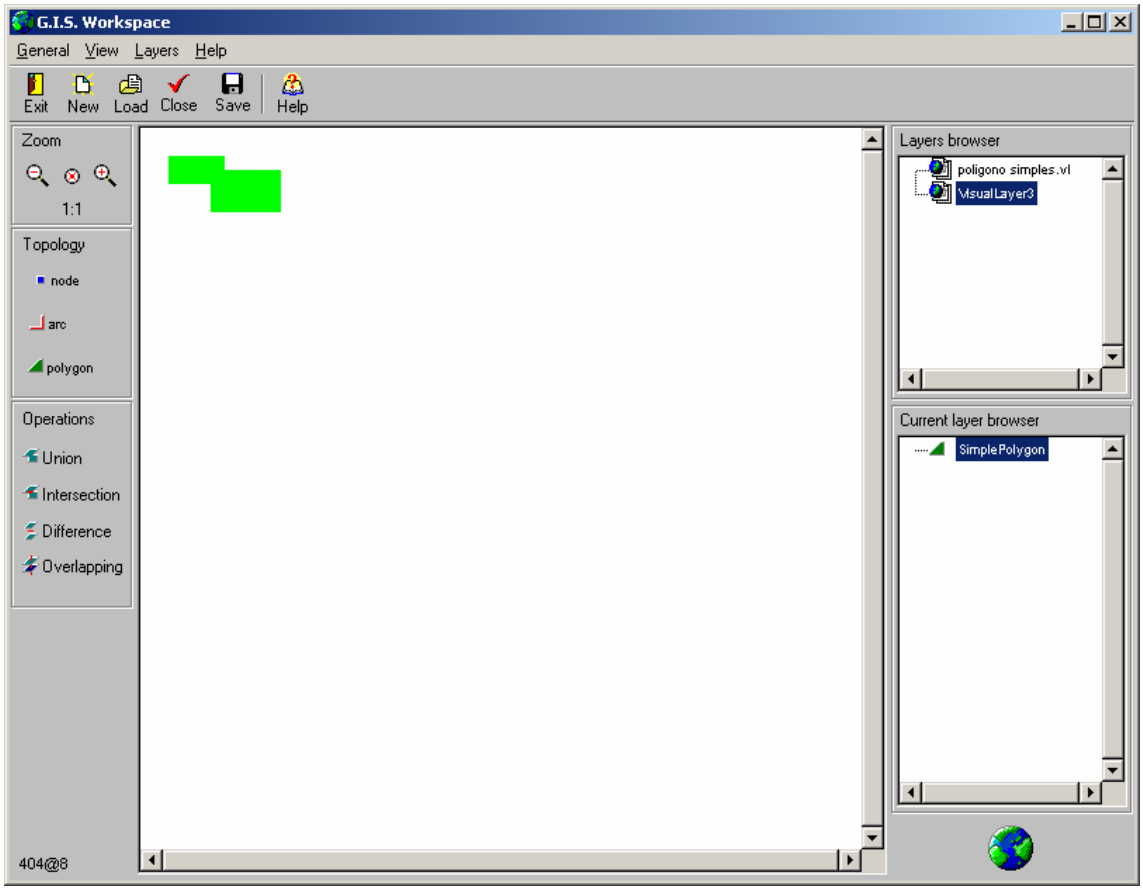



**Figura 8.8**.- Resultado de la unión de los polígonos anteriores

Para realizar la operación de intersección entre los polígonos anteriores, se realiza un click en el tipo de operación 'Intersection', ubicado a la izquierda de la pantalla. A partir de ese click se visualiza la siguiente ventana:

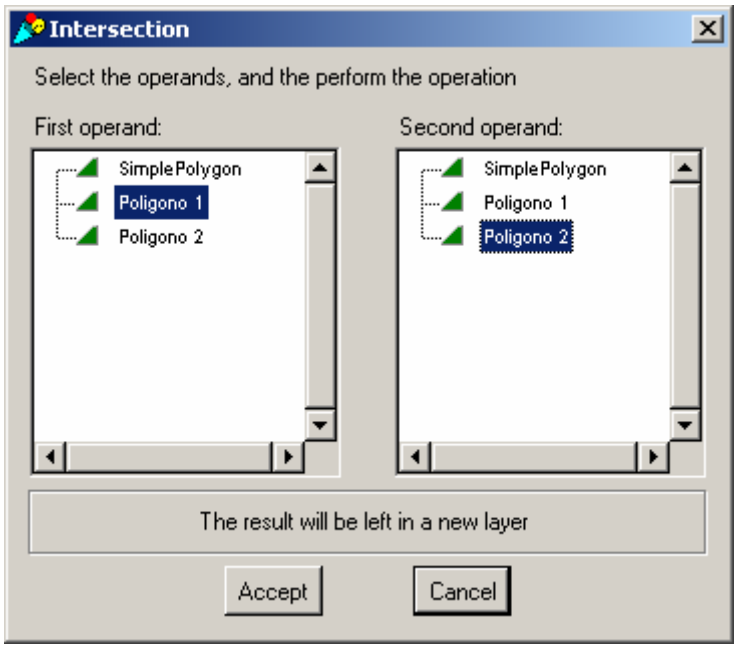



**Figura 8.9**.- Selección de polígonos a intersectar

Como podemos ver en la figura anterior, se agregó un polígono que es el que corresponde a la unión anterior. Siempre se podrá seleccionar cualquier polígono de cualquier layer que esté cargado en la aplicación. Una vez visualizada la ventana anterior se seleccionan los polígonos a intersectar y al confirmar la operación, se crea un nuevo layer con el elemento topológico, otro polígono, resultante de la operación. La siguiente figura muestra el resultado de esta operación:

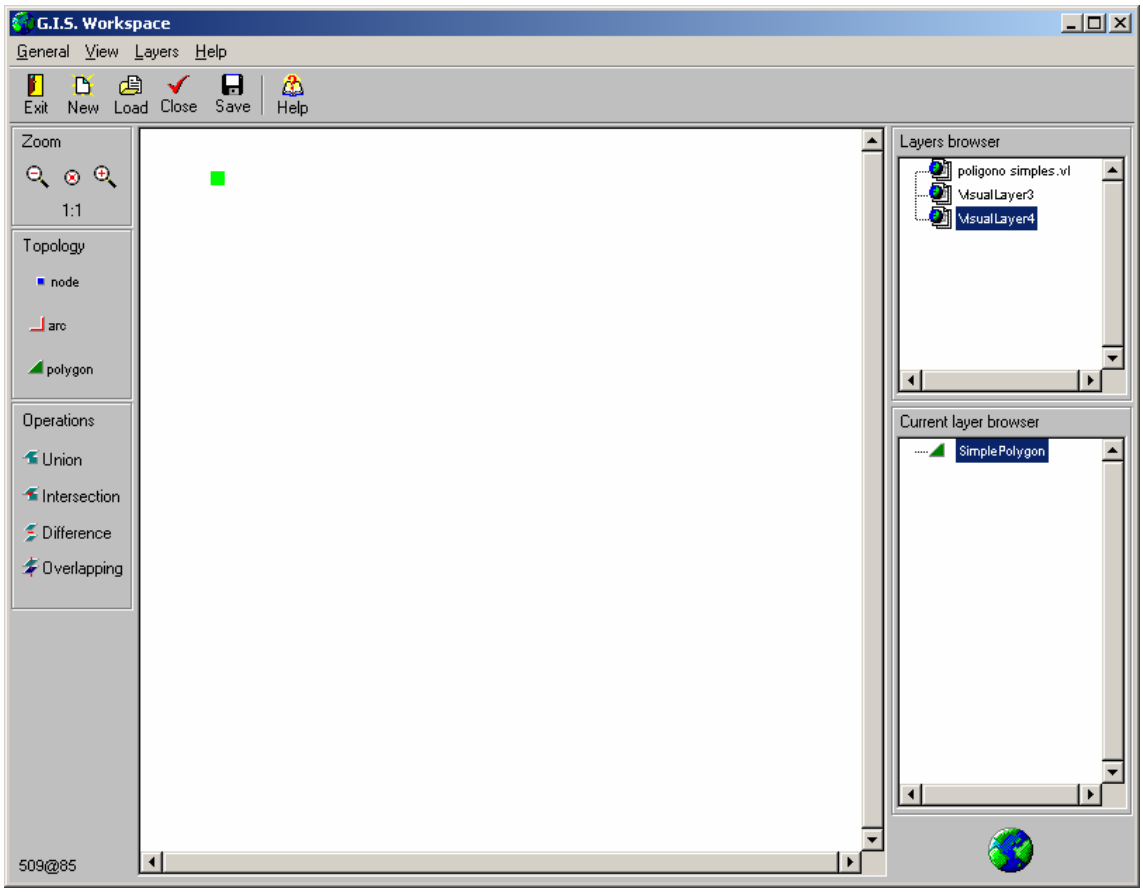



**Figura 8.10**.- Resultado de la intersección de los polígonos anteriores

Para realizar la operación de diferencia entre los polígonos anteriores, se realiza un click en el tipo de operación 'Difference', ubicado a la izquierda de la pantalla. A partir de ese click se visualiza la siguiente ventana:

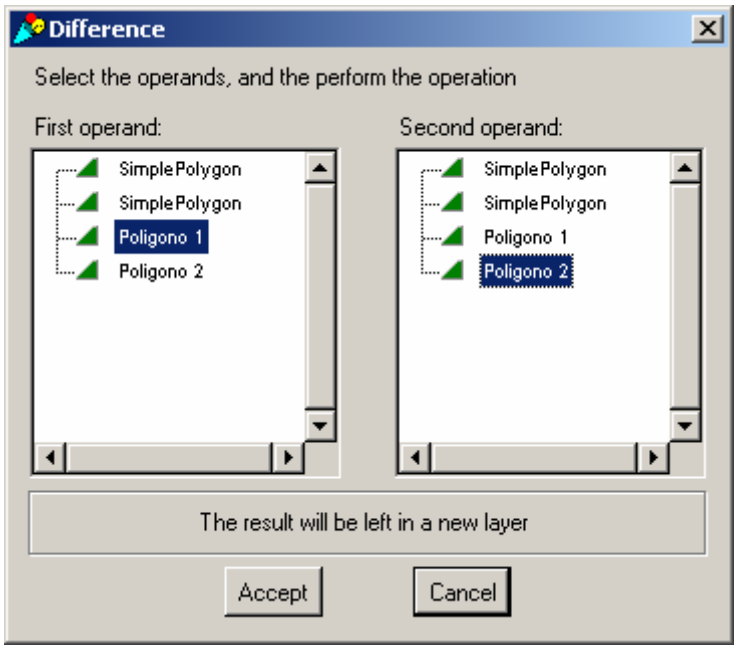



**Figura 8.11**.- Selección de polígonos entre los cuales se hará la diferencia

Una vez visualizada la ventana anterior se seleccionan los polígonos sobre los cuales se realizará la diferencia y al confirmar la operación, se crea un nuevo layer con el elemento topológico, otro polígono, resultante de la operación. La siguiente figura muestra el resultado de esta operación:

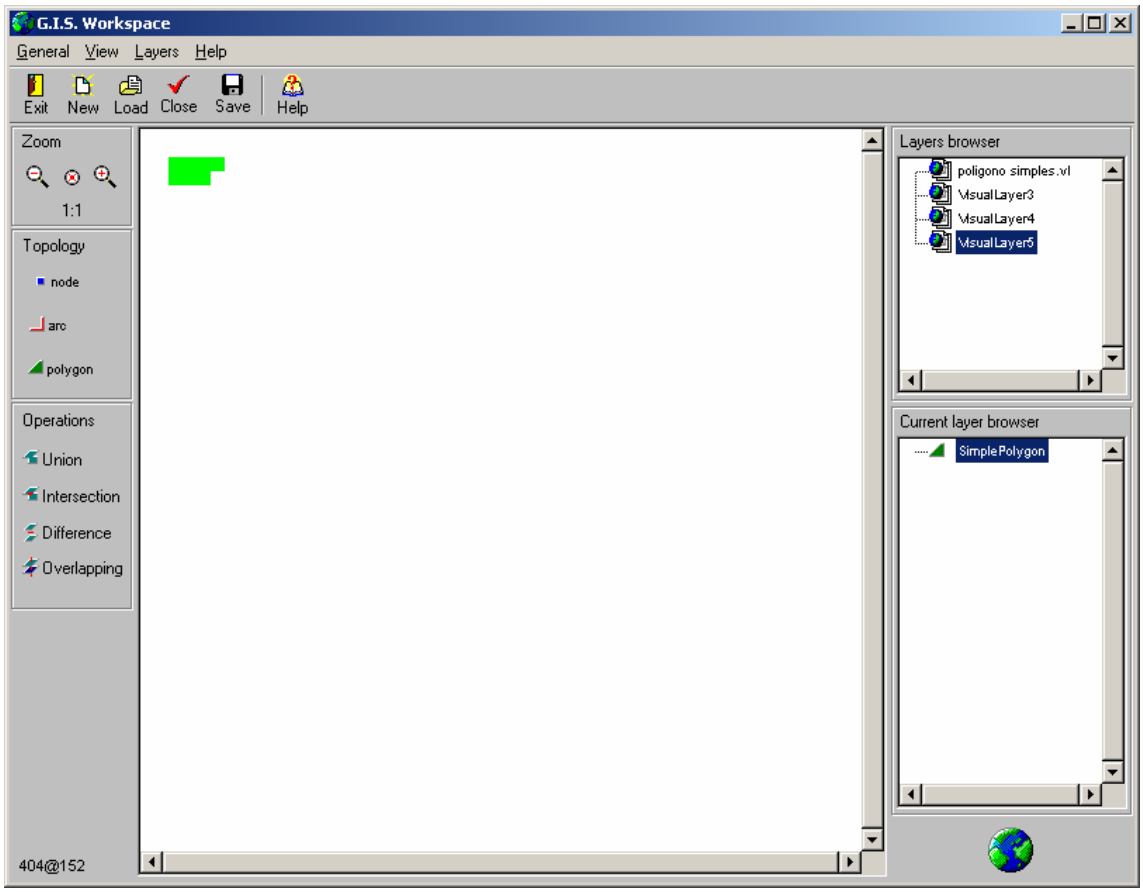



**Figura 8.12**.- Resultado de la diferencia de los polígonos anteriores

Para realizar la operación de superposición, se realiza un click en el tipo de operación 'Overlapping', ubicado a la izquierda de la pantalla. Esta no es una operación geométrica entre elementos, sino que consiste en poner todos los elementos en un solo layer superpuestos. A partir del click en la operación Overlapping, se visualiza la siguiente ventana:

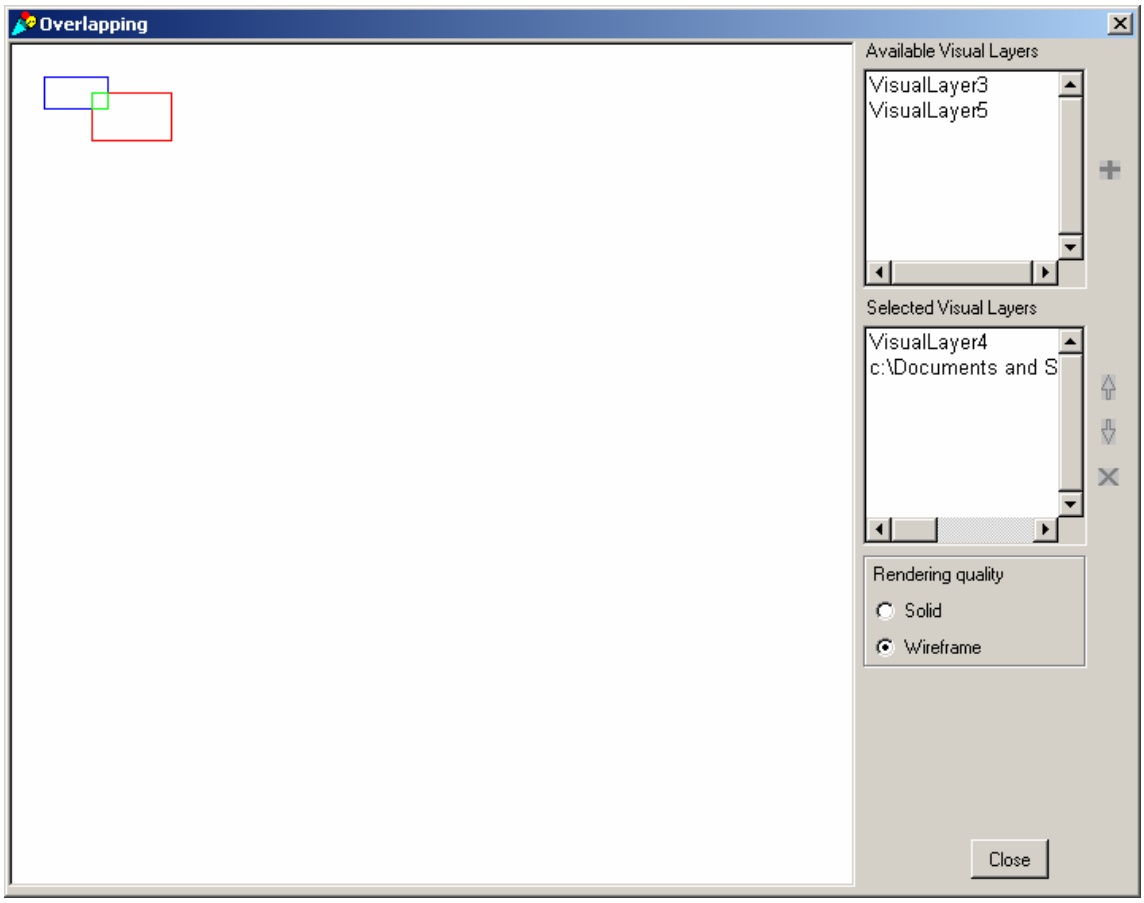



**Figura 8.13**.- Ejemplo de la operación 'Overlapping'

Para la operación anterior se seleccionan los layers para los cuales se desea visualizar sus elementos superpuestos en un solo layer, y se realiza la operación entre los elementos de todos los layers seleccionados. Esta superposición se puede visualizar de dos formas de acuerdo a la calidad de renderización seleccionada.

Los ejemplos anteriores fueron realizados con polígonos simples para mostrar de una manera más fácil las operaciones implementadas. Pero la aplicación es flexible en cuanto a la creación de los polígonos, éstos pueden poseer la forma que se desee. A continuación mostramos las operaciones de intersección y diferencia sobre los polígonos que se ven en la siguiente figura:

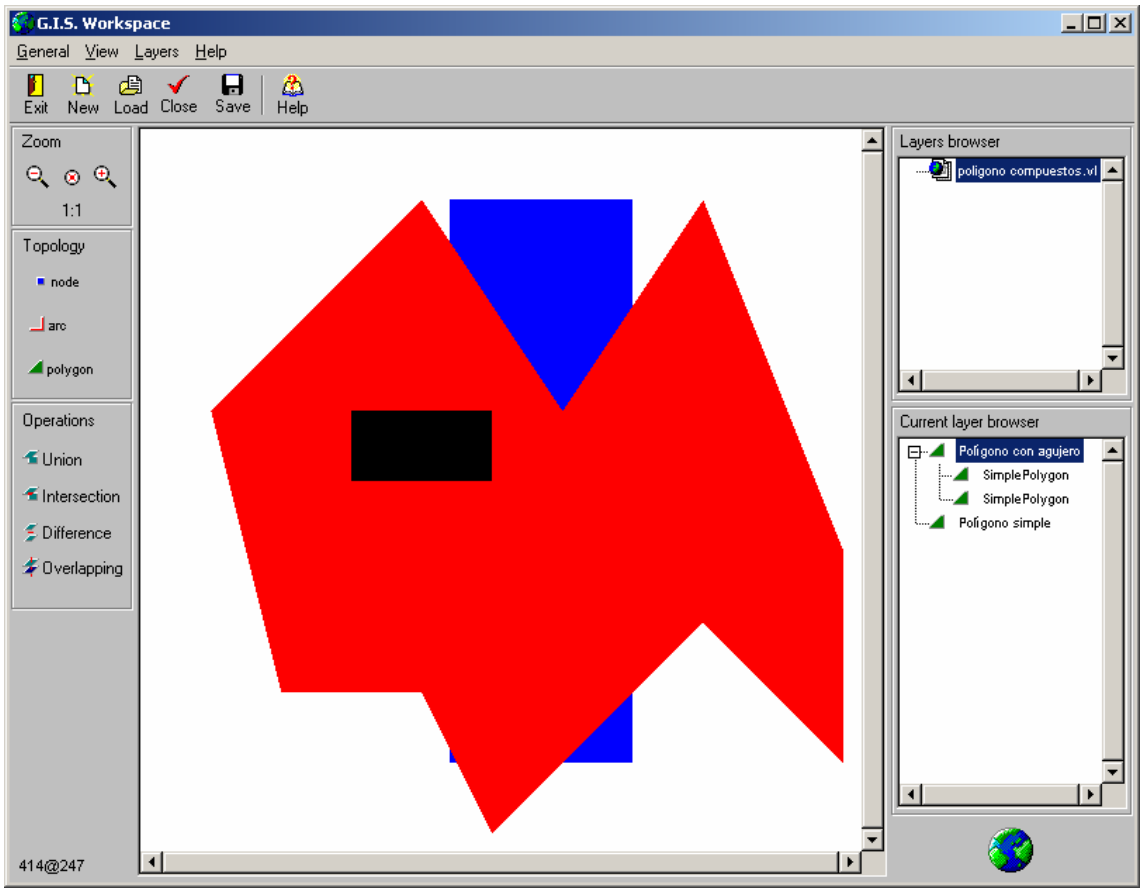



**Figura 8.14**.- Layer con un polígono con agujeros y uno simple

En la figura anterior vemos seleccionado un polígono con un agujero rectangular. Este es el motivo por el cual en el componente 'Current layer browser' se visualiza como formado por dos polígonos. Y se puede visualizar un polígono simple, de color azul, similar a los creados anteriormente.

La operación de intersección entre estos dos polígonos da como resultado el polígono que se puede visualizar en la siguiente figura:

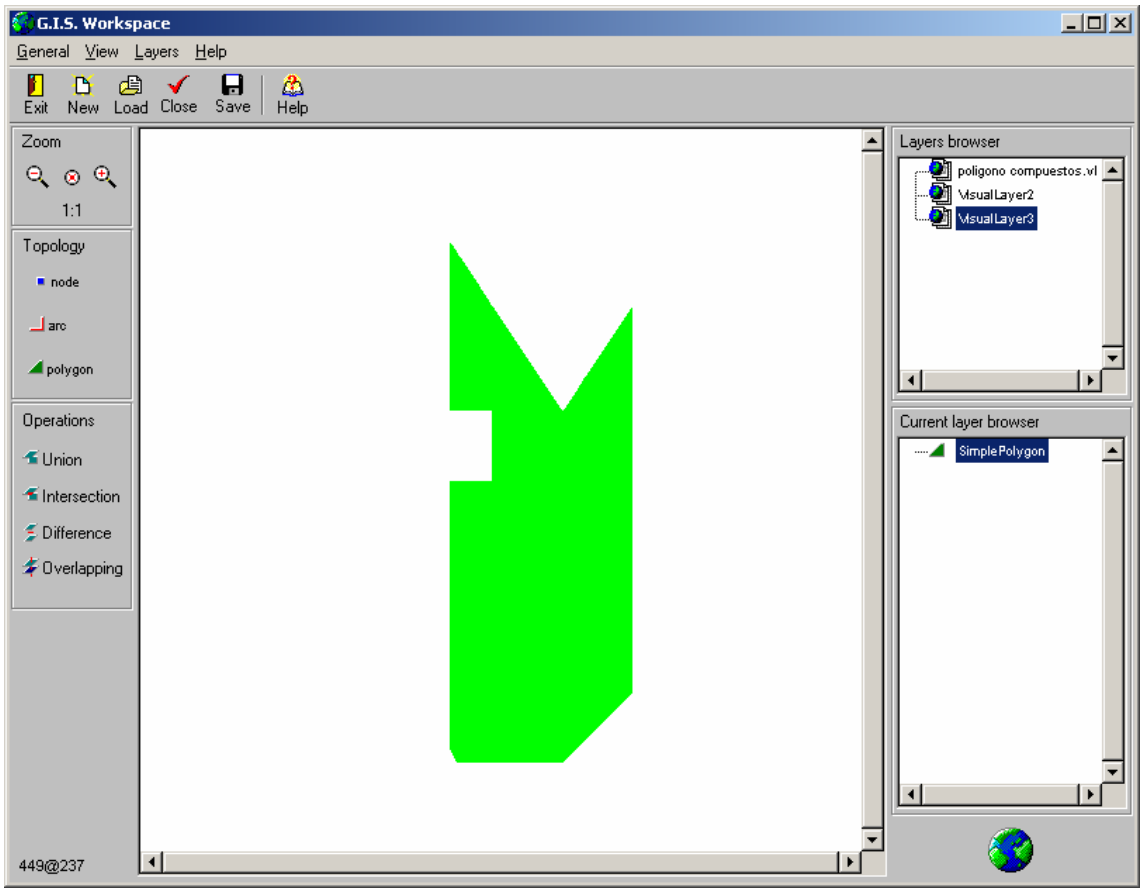



**Figura 8.15**.- Layer con el resultado de la intersección entre los polígonos anteriores

La operación de diferencia entre estos dos polígonos da como resultado el polígono que se puede visualizar en la siguiente figura:

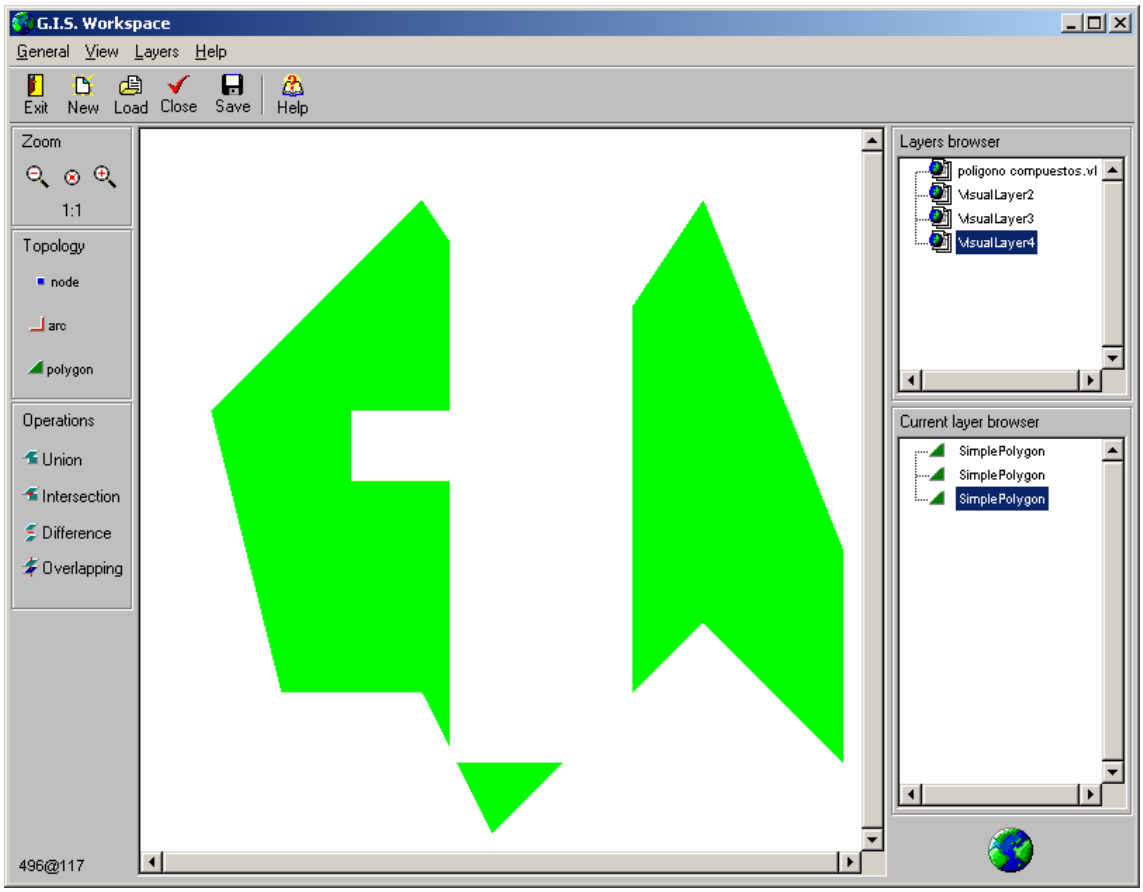



**Figura 8.16**.- Layer con el resultado de la diferencia entre los polígonos anteriores

Las operaciones geométricas realizadas entre los elementos topológicos, como ser la unión, intersección y diferencia, están totalmente desacopladas de la aplicación. La aplicación sólo dispara la acción y visualiza los resultados. La responsabilidad de realizar estas operaciones está a cargo de los elementos que conforman la Topología. Detallamos los pasos que se llevan a cabo para realizarlas:

1. Al clickear en alguna de las operaciones (Union, Intersection, Difference) se setea la interfaz correspondiente para seleccionar los polígonos sobre los cuales se realizará la operación.
2. Una vez seleccionados los polígonos, y confirmada la operación a realizar, se delega a la aplicación principal (G.I.S. Workspace) la responsabilidad para realizar la operación con los polígonos seleccionados.
3. La aplicación principal delega la responsabilidad de llevar a cabo la operación a los elementos topológicos seleccionados, esperando que le devuelva el polígono resultante de la operación para poder visualizarlo.
4. Cuando a un elemento topológico le llega el mensaje de que realice tal operación con otro elemento topológico recibido por parámetro, entran en juego las operaciones geométricas descriptas en el capítulo IV y diseñadas en el capítulo VII. Las operaciones geométricas implementadas para esta aplicación se llevaron a cabo utilizando el algoritmo de Slabs.
5. Una vez que la aplicación principal recibe el polígono resultante de la operación crea un nuevo layer para ese polígono y le asocia al mismo una apariencia, derivada de su forma para poder visualizar el resultado.

A modo de ejemplo, mostramos parte del código en la siguiente figura, es decir el esquema principal del algoritmo, para la operación de unión de polígonos:

```
unionWithPolygon: aPolygon

	"Answer a Polyline which represents the	union between the receiver and aPolyline.It is
computed with the slabing method."


	| slabLines myTrapezoids aPolygonTrapezoids slabsUnions rotationAngle aux1 aux2
finalPolylines |

	(self hasHorizontalEdge or: [aPolygon hasHorizontalEdge])
		ifTrue: [
			rotationAngle := 30.
			aux1 := self rotate: rotationAngle.
			aux2 := aPolygon rotate: rotationAngle.
			[aux1 hasHorizontalEdge or: [aux2 hasHorizontalEdge]]
				whileTrue:
					[rotationAngle := rotationAngle / 2.
					aux1 := self rotate: rotationAngle.
					aux2 := aPolygon rotate: rotationAngle
					].
			aux1 := aux1 verticesAsFixedPoints.
			aux2 := aux2 verticesAsFixedPoints.
			finalPolylines := aux1 unionWithPolygon: aux2.
			^finalPolylines collect: [:each | each rotate: rotationAngle negated]]

		ifFalse:[
		slabLines := self slabLinesWithPolygon: aPolygon.
		myTrapezoids := self slabWith: slabLines.
		aPolygonTrapezoids := aPolygon slabWith: slabLines.
		slabsUnions := Polygon trapezoidUnions: myTrapezoids with: aPolygonTrapezoids
			^Polygon trimTrapezoids: slabsUnions]
```

**Figura 8.17**.- Algoritmo de unión, implementado con Slabs, en la clase Polygon

# 9. CAPÍTULO IX: INTERACCIÓN ENTRE UNA APLICACIÓN CLIENTE Y LA CAPA DE SIG

*Si bien se suele denominar SIG a cualquier aplicación que trate con datos geográficos, cabe hacer la distinción entre dos capas. Una Capa de SIG, que es la capa que provee el soporte con funcionalidad geográfica genérica (por ejemplo indización espacial, soporte de operaciones geográficas, facilidades de ingreso y presentación de información georeferenciada); esta capa por sí misma sólo brinda soporte y no tiene un fin concreto directamente útil al usuario final. Por otro lado existe la Capa de Aplicación, que es la que contempla el objetivo concreto del usuario final; esta capa se apoya sobre la Capa de SIG para obtener soporte de funcionalidad geográfica cada vez que lo necesite [Medeiros].*

*La Capa de SIG debe ser lo suficientemente genérica como para brindar soporte a una variada gama de aplicaciones geográficas concretas. Lograr esto ha sido el objetivo principal del presente trabajo.*

*En este capítulo se describirá la manera en que se integran los distintos aspectos descriptos en los capítulos anteriores de manera de conformar la Capa de SIG, luego se describirá la forma en que la Capa de Aplicación se monta sobre ésta, y por último se describirá un ejemplo de una aplicación concreta para ilustrar el uso con un ejemplo completo y real, perteneciente al dominio de la agricultura.*

## 9.1. Integración de la Aplicación Cliente con la Capa de SIG

En los capítulos anteriores se han desarrollado varios aspectos de un SIG, a saber:

- Datos geográficos, que incluyen el Sistema de referencias y su Topología
- Indización
- Operaciones de Geometría Computacional

El desarrollo de cada uno de estos aspectos se realizó tanto desde el punto de vista conceptual como desde el punto de vista del diseño. En esta sección se mostrará cómo cada uno de estos diseños individuales se unen e interactúan entre sí de manera de formar el diseño de un SIG integral.

### 9.1.1. Integración de objetos del dominio con el Sistema de Referencias y la Topología

En el Capítulo V mostramos la forma en la que se unían los objetos del dominio de aplicación con los objetos de un sistema de información geográfica. Utilizamos la solución planteada por [Gordillo] y [Balaguer], la cual propone utilizar el patrón de diseño Decorator [Gamma]. La ventaja de esta solución radica en que podemos seguir interactuando con aplicaciones existentes ya que el decorador actúa como wrapper del objeto existente. De esta forma, no se contamina a los objetos de la aplicación con protocolo que debe conocer y saber responder sólo un objeto de la capa correspondiente al sistema de información geográfica.

Este objeto geográfico debe ser capaz de poder representar la forma y posición de un objeto del dominio de la aplicación. Para esto debe interactuar con la Topología y con un Sistema de Referencias, tal como se los ha definido y diseñado anteriormente. El hecho de que sea este objeto el que conozca su Sistema de Referencias permite que distintos objetos del dominio de la aplicación puedan referenciar su posición con el sistema más apropiado, y no sea el mismo para todos ellos.

Recordando el diseño propuesto anteriormente, la unión de los Objetos del Dominio con los Objetos Geográficos quedaría como se muestra en la siguiente figura.

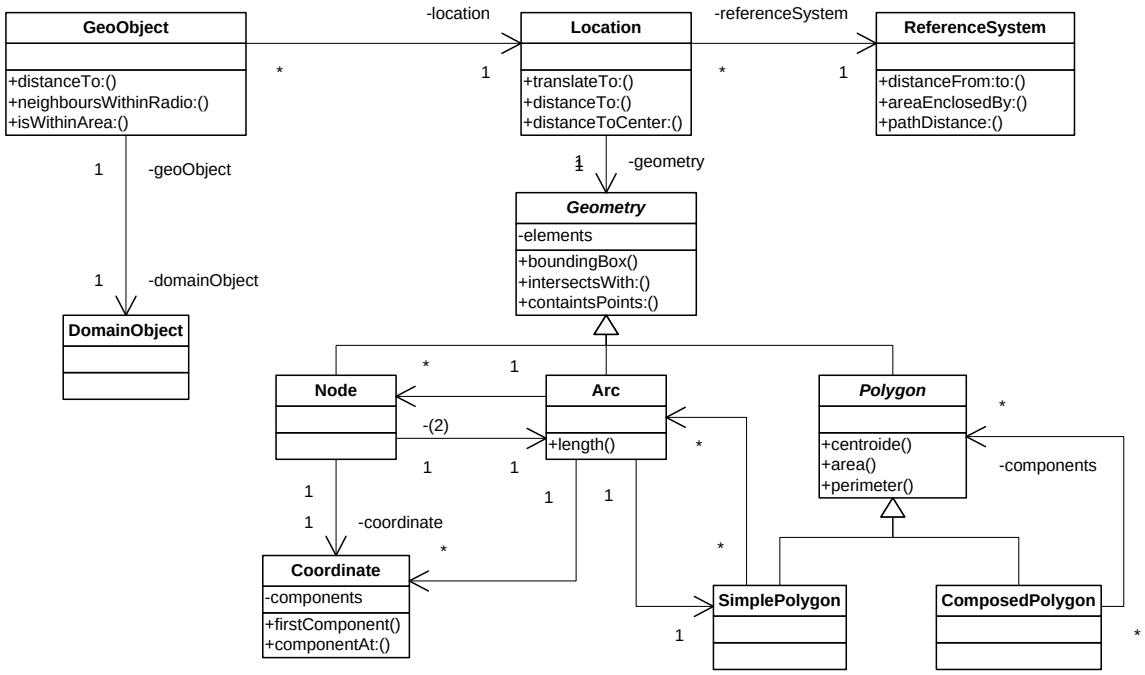



**Figura 9.1**.- Modelo final de la unión de los Objetos del Dominio con los Objetos Geográficos.

A continuación detallamos las responsabilidades que brindan las clases que vemos en el diseño de la figura anterior:

- ***DomainObject***: se utiliza para representar los objetos del dominio. El protocolo dependerá de la Aplicación Cliente que se integre a esta Capa SIG.
- ***GeoObject***: es el encargado de manejar de manejar los aspectos geográficos del objeto del dominio, como ya dijimos anteriormente se utiliza para representar la forma y posición del objeto, y para esto colabora con la *Location*, a su vez ésta colabora con instancias de las subclases de *Geometry* y con una instancia de la clase *ReferenceSystem*.
- ***Location***: representa la posición de un objeto sobre la Tierra.
- ***ReferenceSystem***: provee el contexto en el cual serán interpretadas las coordenadas de los objetos topológicos y se harán los cálculos entre éstos. Sólo se detalla esta clase en la figura 9.1, pero abarca el modelo final para un sistema de referencias presentado en la figura 5.8 del Capítulo V.
- ***Geometry***: es una clase abstracta que permite manejar el comportamiento común de los tipos de elementos que se utilizan para representar los tres elementos del modelo de vector (nodo, arco y polígono).
- ***Node***: representa la posición puntual de un objeto o la intersección entre dos arcos. Realiza operaciones geométricas entre él y otros objetos topológicos.
- ***Arc***: representa la posición longitudinal de un objeto. Realiza operaciones geométricas entre él y otros objetos topológicos.
- ***Polygon***: representa la posición de un objeto que ocupa un área. Define operaciones generales entre polígonos y otros objetos topológicos.
- ***SimplePolygon***: representa la posición de un objeto que ocupa un área continua (no disjunta y sin agujeros).
- ***ComposedPolygon***: representa la posición de un objeto que ocupa un área disjunta y/o con agujeros.
- ***Coordinate***: representa la coordenada de la posición de un objeto, su valor numérico. Sólo es interpretada en el contexto provisto por algún sistema de referencia.

### 9.1.2. Integración de la Indización

En esta sección se tratarán básicamente dos aspectos relacionados a la Estructura de Indización: su independencia del sistema de referencias en que se encuentren representados los objetos que almacena, y la forma en que la misma debe ser notificada para ser actualizada cuando algún geo-objeto de los que almacena cambia de posición o de forma.

Con respecto al primer punto, una estructura de indexación referenciará, desde los nodos correspondientes, a los geo-objetos, los cuales decoran a los objetos del dominio. De esta forma, la estructura de indexación está totalmente desacoplada de estos últimos y del sistema de referencia en que se encuentran representados.

Lo único que necesita la estructura de indexación es que el geo-objeto al que referencia le devuelva su Location, la cual puede responder a los mensajes que devuelven sus coordenadas o su bounding box (depende de la estructura de índice en cuestión). Es gracias a este aspecto que se logra la independencia y desacoplamiento mencionado. Esto se debe a que las Locations pueden compararse y operar entre sí en varios aspectos sin importar el Sistema de Referencias concreto en el que se encuentra representada cada una. Esta ventaja se evidencia y se explota en este caso concreto, el de la Estructura de Indización, ya que la extensión o ubicación espacial de sus nodos se representa mediante Locations en lugar de coordenadas de un sistema específico (por ejemplo x-y o latitud-longitud), esto permite que la estructura pueda indizar geo-objetos que se encuentran representados en distintos sistemas de referencias. Por ejemplo, en la operación de agregado de un geo-objeto, se buscará la posición del mismo en la estructura mediante sucesivas comparaciones entre su Location y la Location de los nodos de la estructura, lo cual permite absoluta independencia entre la Estructura de Indización y el Sistema de Referencia de cada geo-objeto.

El segundo problema importante que se debe solucionar es el de detectar cambios de posición o forma en los objetos para actualizar en consecuencia la estructura de indización. La estructura por sí misma no es capaz de detectar los cambios que sufren los objetos que almacena, por lo tanto debe implementarse algún circuito de notificación de forma de que cada cambio relevante en un geo-objeto resulte en una actualización de la estructura.

En la búsqueda de la solución debe evitarse a toda costa un alto acoplamiento desde el geo-objeto hacia la estructura o la aplicación. Por ejemplo, una solución directa sería que el geo-objeto conociera explícitamente a la estructura de indización, sin embargo debe tenerse en cuenta que la indización es sólo un aspecto más del SIG que puede incorporarse opcionalmente, con lo cual no tendría sentido tal referencia en un escenario sin indización; por otro lado, el geo-objeto tiene la responsabilidad de aportar características geográficas a objetos del dominio, y no características de indización. Una mala distribución de responsabilidades entre los objetos deriva en la falta de flexibilidad del modelo [Rebecca].

Una solución adecuada es la utilización del patrón de diseño Observer [Gamma]. Este patrón propone básicamente dos roles: Observador y Observado. Puede haber varios objetos observadores que observan a mismo objeto (observado). Cada vez que el observado cambia en algún aspecto relevante notifica a los observadores sobre el suceso, de manera que éstos decidan cómo actuar en consecuencia. Para esto, el observado mantiene referencias a los observadores, pero estas referencias son débiles. Esto último significa que no los conoce explícitamente y que no sabe a priori (estáticamente) qué tipo de objetos conocerá, esto permite dinamismo en el alta y baja de objetos que desean ser notificados (observadores) y además permite que éstos sean de cualquier clase. Ésta es una diferencia fundamental con respecto a la alternativa de conocimiento explícito mencionada anteriormente pues de ninguna manera impone algún tipo de acoplamiento, por lo que resulta apta como solución al problema de actualización de la estructura de indización.

En nuestro caso, los geo-objetos tendrán el rol de observados y la Aplicación Cliente (que conoce a la estructura de indización) tendrá el rol de observador. De esta manera, cada vez que un geo-objeto cambie su forma notificará este hecho (esto es importante: sólo notifica este hecho, pero no dice qué debe hacerse al respecto ni quién debe hacerlo, ambas cosas devendrían en un alto grado de acoplamiento) a la Aplicación Cliente (oservadora). Una vez notificada, la Aplicación Cliente sabría que debería considerar la actualización de la estructura de indización con respecto a ese geo-objeto; dado que la aplicación conoce explícitamente a la estructura, la actualización de la misma le resulta trivial, ya que se limita al uso del protocolo que ésta provee.

### 9.1.3. Integración de las Operaciones de Geometría Computacional

Las operaciones están estrechamente ligadas a la topología de los geo-objetos ya que ellos deben saber compararse y operar con otros geo-objetos de manera de poder responder si se intersectan o si se incluyen, como así también calcular su intersección o unión, entre otros testeos y operaciones.

La integración de estas operaciones ya ha sido explicada anteriormente en la sección correspondiente al diseño de las operaciones Geometría Computacional, en el Capítulo VII. Cabe recordar que existen varias técnicas para realizar cada operación, y que el modelo no impone el uso de una determinada, sino que permite al usuario elegir la que considere más acorde a su caso particular. Incluso puede también desarrollar otros algoritmos e integrarlos al modelo de manera directa y totalmente flexible.

## 9.2. Adaptación de la Aplicación Cliente a la capa de SIG

Hasta aquí se ha descripto la manera en que se integran los distintos componentes y aspectos de la Capa de SIG: estructuras de indización, operaciones de geometría computacional, sistema de referencias y topología. En esta sección se describirá la manera en que se monta una aplicación sobre esta capa, es decir, la forma de integración entre la Capa de la Aplicación y la Capa de SIG.

Los objetos del dominio, "decorados" por objetos geográficos, son manejados por una Aplicación Cliente. Esta aplicación es la encargada de manejar la funcionalidad que se brinde al usuario: por ejemplo agregar objetos, buscarlos, modificarlos y relacionarlos mediante operaciones determinadas.

Cuando se agrega un objeto del dominio a la aplicación, ésta debe:

- Crear un Geo-objeto y asociarlo al Objeto del Dominio.
- Asociar una Location al Geo-objeto. Y a su vez:
  - Configurar la Location con el *Sistema de Referencias* que el usuario haya elegido.
  - Crear una *Topología* según la forma especificada para el objeto ingresado y asociarla a la Location.
- Agregar el geo-objeto configurado a las *Estructuras de Indización* correspondientes.

Una vez hecho esto, la aplicación podrá acceder a los objetos a partir de las estructuras de índices. Se asume que un objeto puede estar almacenado en distintas estructuras a la vez, al igual que en las aplicaciones tradicionales. El mantenimiento de las estructuras de indización en base a los cambios de posición y forma de los geo-objetos ya ha sido descripto anteriormente en este mismo capítulo.

El usuario podrá seleccionar de una sola vez las técnicas de geometría computacional que se utilizarán en todo el sistema, aunque podrá cambiarlas por otras en cualquier momento que lo desee. Esto se realiza de la manera especificada en el capítulo en que se trata el diseño de las operaciones de geometría computacional.

Es importante notar el bajo grado de acoplamiento entre la Capa de Aplicación y la Capa de SIG. Si bien la Capa de Aplicación debe conocer algunos elementos básicos de la Capa de SIG (Location, Sistema de Referencia, Topología), esto sucede sólo en momentos muy puntuales e inevitables, como por ejemplo en la creación, tal como se acaba de describir. Una vez creado e incorporado el geo-objeto a la aplicación, todas las operaciones geográficas son resueltas íntegramente por los elementos de la Capa de SIG, siendo éstas totalmente transparentes a la Capa de la Aplicación; esta última jamás interviene directamente en testeos geométricos, operaciones de intersección, unión, cálculo de distancias entre objetos, actualización de topologías y demás aspectos geográficos.

## 9.3. Ejemplo Concreto de la Integración

En esta sección desarrollaremos un ejemplo de una aplicación que utilizará la Capa de SIG desarrollada en los capítulos anteriores. La idea es mostrar con un ejemplo cómo se uniría la Aplicación Cliente a los elementos diseñados como parte de la Capa de un SIG.

Esta aplicación trata sobre una empresa agrícola, la cual necesita conocer el estado de los lotes de su campo para mejorar la productividad, aprovechando las ventajas de tener un sistema que le permita conocer el estado de dichos lotes.

Con este ejemplo mostraremos el diseño que debe tener la Aplicación Cliente y cómo interactúa con el diseño ya desarrollado. Se detallaran diagramas de clases de la Aplicación Cliente, diagramas de instancias para mostrar la integración con la Capa de SIG y diagramas de interacción para mostrar el comportamiento de los objetos ante ciertas situaciones de ejemplo.

### 9.3.1. Resumen de la funcionalidad de la aplicación

La Aplicación Cliente contemplará la funcionalidad necesaria para llevar a cabo las tareas de una empresa agrícola, como mencionamos anteriormente. Esta empresa cuenta con distintos campos y estancias, los cuales se encuentran divididos en lotes. Estos lugares están ubicados en distintas regiones de nuestro país.

Su principal actividad consiste en la siembra/cosecha de cereales en los distintos lotes. Debido a que los lotes se encuentran dispersos por todo el país, en cada lugar se siembran productos acordes a la región. Como ejemplo de esto podemos mencionar que en el sur de la provincia de Buenos Aires predomina el girasol, y en el centro de la misma soja, maíz y trigo, entre otros.

La necesidad de esta empresa radica en conocer ciertos datos relacionados con la superficie de los lotes y el clima característico de los lugares donde se encuentran, de modo de poder aprovechar al máximo las ventajas de cada uno, brindando al usuario la información necesario para que éste pueda anticiparse a determinados eventos climáticos, en particular sequías.

La empresa cuenta con maquinarias para realizar las tareas involucradas en esta actividad agrícola, y cuenta con tecnología que le permite obtener información geográfica, como ser dispositivos gps y fotos satelitales. Esta tecnología podría ser utilizada por la Aplicación Cliente para cargar la información necesario de una manera más cómoda y flexible al usuario.

Además de manejar los aspectos relacionados con los suelos de los lotes, se deberán administrar las tareas relacionadas con el riego de estos lotes, de manera de controlar las posibles sequías. De esta manera deben monitorearse los equipos de riego que tiene asignado cada lote. Estos equipos pueden ser reasignados a nuevos lotes de acuerdo a las necesidades de la empresa.

En conclusión, la Aplicación Cliente deberá ser capaz de:

- manejar lotes, esto incluye información de dominio y geográfica. Como información de dominio, podemos detallar su nombre o identificación, promedio de productividad, promedio de humedad, tipos de semillas más adecuadas, etc. Como información geográfica, se detalla humedad del suelo en distintos sectores del lote, tipo de suelo en distintos lugares, si contiene ríos, arroyos o lagunas, etc.
- mantener los equipos de riego asignados a cada lote. Estos serán rotados de acuerdo a la humedad del suelo de cada lote y al pronóstico indicado para cada área.
- permitir visualizar cada lote, administrando la información antes mencionada.

Para poder llevar a cabo las tareas relacionadas con información geográfica, la Aplicación Cliente deberá interactuar con la Capa de SIG, ya que será ésta la que maneje dichos aspectos.

### 9.3.2. Detalles acerca del dominio de la aplicación

Uno de los principales temas en los cuales tuvimos que ahondar para poder entender las necesidades de la Aplicación Cliente es el conocimiento sobre el suelo y sus relaciones con el agua y las plantas. La información fue recolectada según charlas mantenidas con personas que poseen experiencia en este tema [Martin] y de material sobre cursos de métodos de riego [Prieto]. A continuación se realiza una descripción sintética de la información que se consultó para desarrollar el modelo de esta aplicación.

El suelo actúa como una reserva de agua y sales, y es un medio poroso permeable al flujo de líquidos y gases. En consecuencia, para planificar las tareas de riego es de interés la estructura del suelo, su textura, el contenido de materia orgánica, porosidad, materiales constituyentes y otros aspectos vinculados con las propiedades físicas del mismo. También se pueden detallar los estados posibles que puede tomar el suelo, según la humedad:

- Saturado: con poros llenos de agua

- Capacidad de campo: el agua que retienen las partículas después del exceso se ha drenado por gravedad
- Punto de marchitamiento permanente: el agua envuelve las partículas, pero éstas las retienen y no pueden ser absorbidas por las plantas

El flujo de agua a través de las plantas se vincula a la velocidad de circulación desde el suelo hacía las hojas y su transpiración y evaporación directa del suelo. La necesidad de agua de las plantas se basa en eso, y es esta necesidad la que debe ser atendida mediante las técnicas de riego, utilizando métodos que resulten adecuados y convenientes desde el punto de vista económico.

El riego está identificado desde hace mucho tiempo como un medio para aumentar la producción de cultivos, y en consecuencia conforman la base de la alimentación y del desarrollo de los pueblos. Para planificar su aplicación se debe tener en cuenta:

- Duración del riego: debe realizarse el primero antes o después de la siembra, y el último antes de la cosecha
- Volumen estacional del riego: cantidad de agua que debe agregarse para producir un cultivo
- Lámina de riego: cantidad de agua que se repondrá al suelo en cada riego
- Momento óptimo del riego: momento en que debe efectuarse el riego, cuando el agua almacenada en el suelo ha disminuido más de un umbral que no debería pasarse
- Momento efectivo del riego: momento en que se realiza el riego
- Intervalo de riego: tiempo entre dos riegos
- Velocidad de infiltración: capacidad del suelo de dejar penetrar la lámina de agua.
- Tiempo de riego: duración del riego definida por el tiempo que tarda la lámina de agua en infiltrar en los riegos de superficie, o del tiempo en cada posición para los riegos por aspersión
- Dotación específica continua: caudal continuo que debería requerirse para satisfacer el volumen estacional requerido para toda el área.
- Disponibilidad de agua: se debe conocer el volumen, calidad, entrega y costo del agua para el riego
- Caudal de perforaciones: volumen de agua entregada cuando se aplica bombeo por perforaciones
- Caudal de manejo: caudal que se entregará en el tiempo de riego

La clasificación de los métodos de riego que detallamos a continuación, y que nos será de utilidad para asociarle la topología correspondiente a cada equipo de riego, se refiere a los criterios de la fuente de energía que se emplea (fuerza de gravedad o presión hidráulica) y a la posición en la que se coloca el agua (que moje todo el terreno o sólo las raíces de los cultivos):

- Gravitacionales: entre ellos se encuentran inundación, parcelas bordeadas o melgas, surcos, corrugación. Son los más utilizados, el agua se ubica sobre la superficie del terreno. En la siguiente figura podemos visualizar un sistema de riego por melgas alimentadas con sifones plásticos:

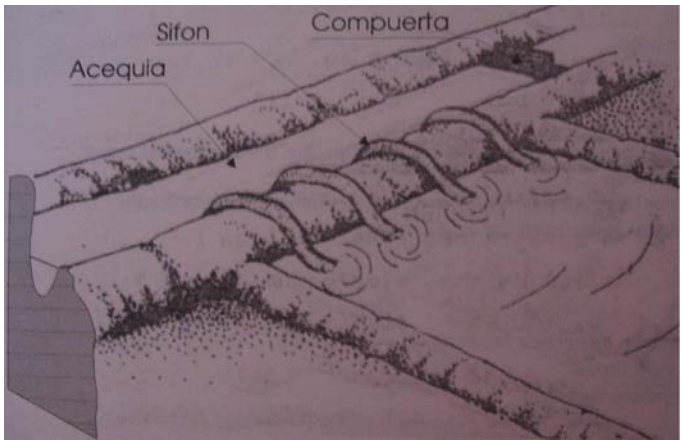



**Figura 9.2**.- Ejemplo de un equipo de riego gravitacional.

- A presión: entre ellos se encuentran surcos o melgas desde tuberías con válvulas o microcompuertas, aspersión (fijos, semifijos, móviles), localizado (goteo, microaspersión, etc):
  - Los primeros son similares a los gravitacionales sólo que varía la forma en hacer llegar el agua y la forma de introducirla en la parcela.
  - Los segundos, por aspersión consisten en conducir el agua a presión desde la fuente hacía conductos cerrados, por ellos se traslada hasta las alas que lanzan el agua al aire como si fuera lluvia. Son útiles cuando el suelo no tiene la suficiente pendiente. Dentro de los de aspersión móvil podemos visualizar al cañón viajero, el cual se muestra en la siguiente figura:

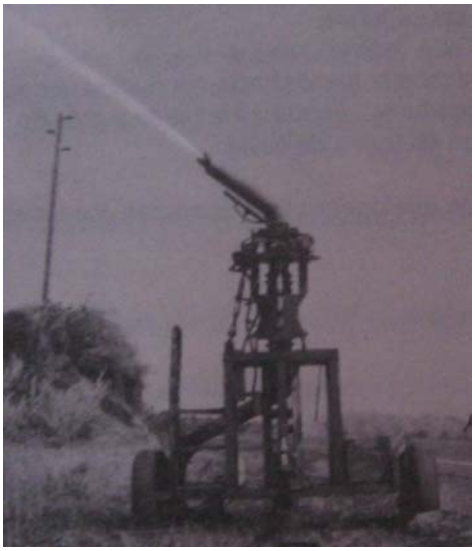


**Figura 9.3**.- Ejemplo de un equipo de riego por aspersión.

  - Los del tercer tipo, localizados, son de gran precisión, operan a baja presión con gran eficiencia. Los de goteo son similares a los de aspersión con sistema fijo. Los de microaspersión son una mezcla entre los de aspersión con sistema fijo y los de goteos mencionados anteriormente. A continuación se muestra un ejemplo de riego por goteo:

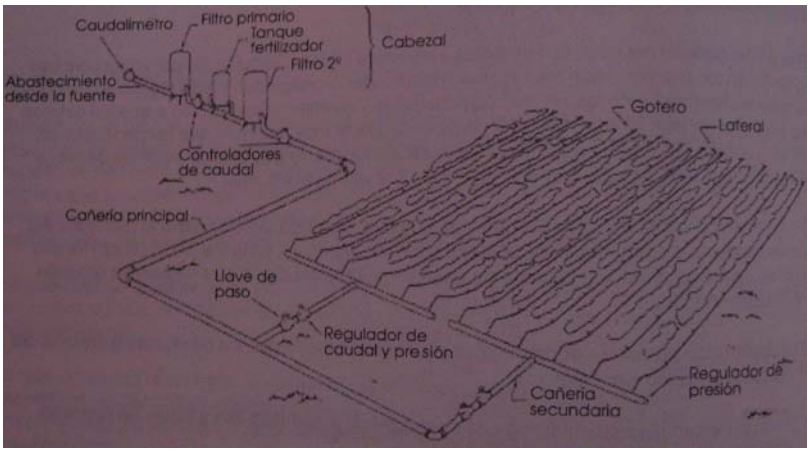



**Figura 9.4**.- Ejemplo de un equipo de riego por goteo.

- Subirrigación: podemos mencionar alimentación de la capa freática, tupos perforados, recipientes enterrados. Se aplican en casos muy especiales, donde ya se cuente con instalaciones de drenaje, y la capa freática (primer acuífero que se encuentra en profundidad, a partir de la superficie del terreno) es poco salina. Muy utilizado en Holanda.

Los criterios para elegir uno u otro método de riego varían de acuerdo a factores agronómicos (características del terreno, exigencias de los cultivos, factores climáticos, disponibilidad de agua, calidad del agua), factores económicos (costos, conveniencia económica), factores sociales (mano de obra) y factores ambientales.

### 9.3.3. Decisiones de diseño para relacionar aspectos del dominio de la aplicación con la Capa SIG

En base a lo detallado anteriormente podemos tomar ciertas decisiones en cuanto a la forma en que representaremos la información geográfica de los elementos de la Aplicación Cliente.

Para modelar los lotes y poder conocer la humedad del suelo, utilizaremos elementos geográficos a los cuales les asociaremos una topología de polígono. Para los ríos y arroyos que puedan existir en cada lote utilizaremos una topología de arco.

En cuanto a los equipos de riego, que se encuentran en cada lote, podemos utilizar distintas topologías de acuerdo al tipo de equipo de riego. A continuación se brindan algunos ejemplos:

- para los equipos de riego gravitacionales o a presión con surcos o melgas, es conveniente utilizar una topología de arco compuesta por arcos simples. De esta forma se puede representar muy fácilmente los lugares donde se está regando, ya que queda marcado por los arcos que representan los surcos o melgas.
- para los equipos de riego a presión por aspersión, es conveniente utilizar una topología de nodo, para indicar el punto en donde se encuentra ubicado el equipo

### 9.3.4. Diseño Orientado a Objetos de la Aplicación Cliente

Asumiendo que la Aplicación Cliente es implementada bajo el paradigma de la programación orientada a objetos, a continuación se detalla un diagrama que muestra las clases necesarias para poder modelar el *dominio* de la Aplicación Cliente:

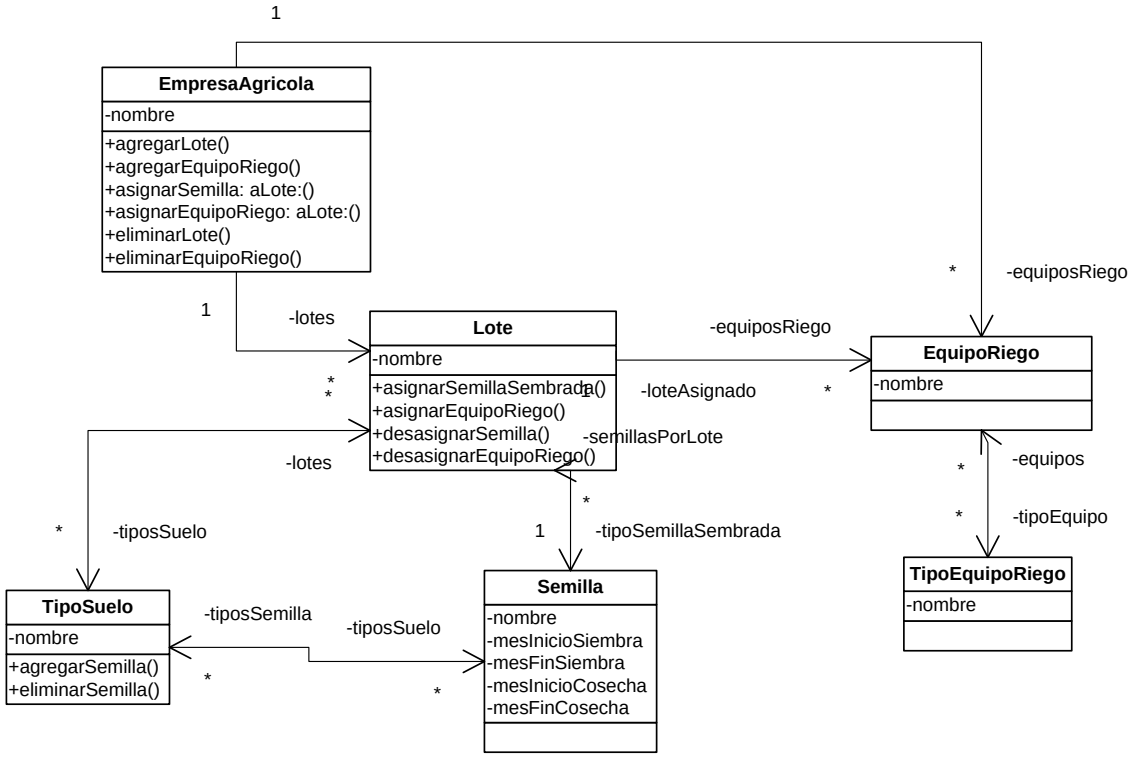



**Figura 9.5**.- Modelo, orientado a objetos, de la Aplicación Cliente.

Las responsabilidades de cada clase, detallada en la figura anterior, son:

- ***EmpresaAgrícola***: es la encargada de administrar toda la información asociada a los lotes y las actividades que se desarrollan en cada uno.
- ***Lote***: administra la información relacionada a él, como ser sus tipos de suelo, semilla sembrada, equipos de riego asignados a él.
- ***EquipoRiego***: representa los equipos de riego con que cuenta la empresa, y maneja la información del lote en él que se encuentran ubicados, junto con información característica de cada tipo de equipo.
- ***TipoEquipoRiego***: administra los tipos existentes de equipos de riego. Instancias de esta clase serían los objetos que representen los tipos gravitacionales, por subirrigación, etc.
- ***TipoSuelo***: administra los tipos de suelo y las semillas que pueden asociarse para esos tipos de suelo.
- ***Semilla***: mantiene información relacionada a cada tipo de semilla.

### 9.3.5. Integración de la Aplicación Cliente con la Capa de SIG

Para realizar la integración entre la Aplicación Cliente y la Capa de SIG debemos asociarle a los objetos del dominio que necesitan ser georeferenciados los objetos que manejarán toda la información geográfica de los mismos.

A continuación se visualiza un diagrama de clases que muestra como se modela la integración entre la Aplicación Cliente y la Capa de SIG:

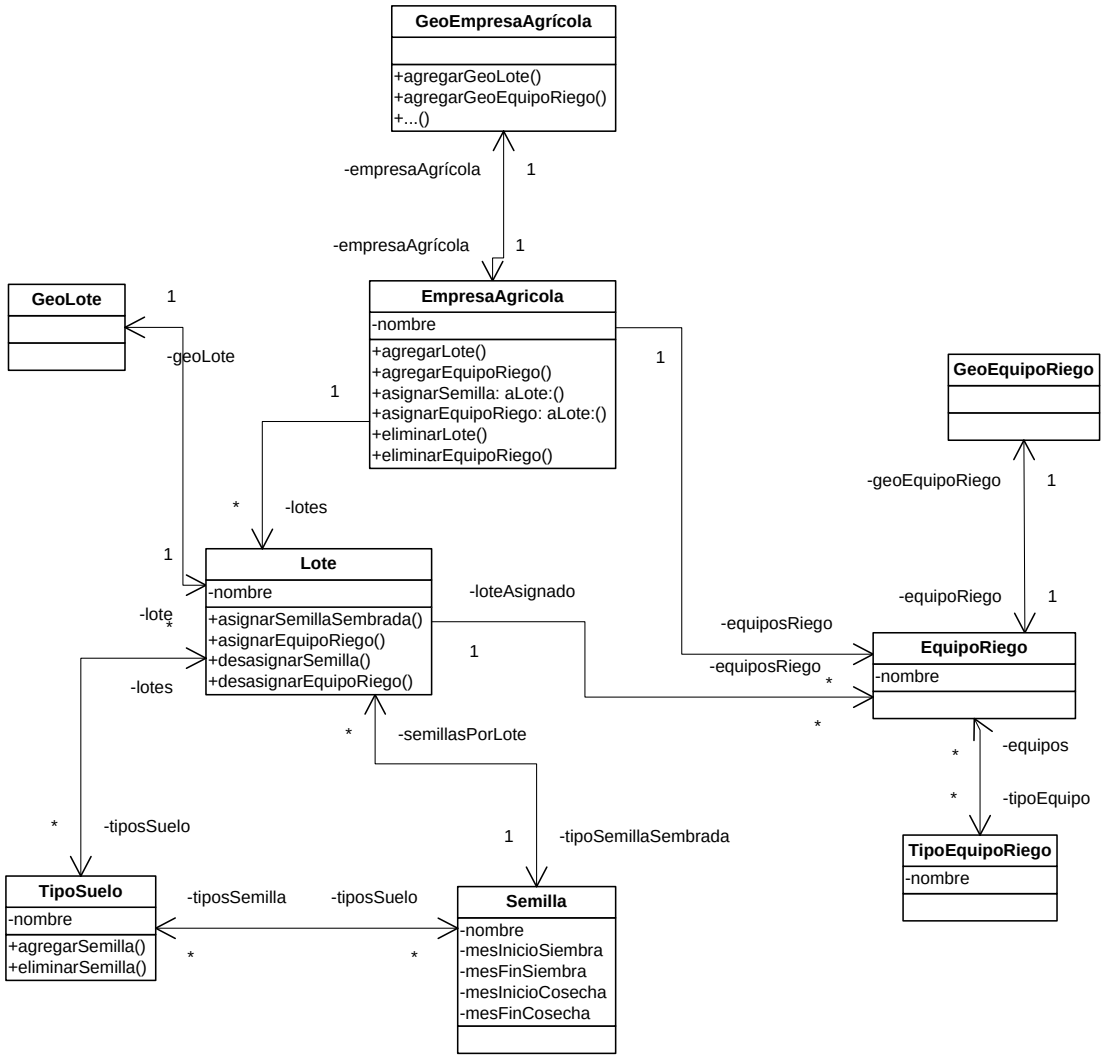



**Figura 9.6**.- Modelo, orientado a objetos, de la integración entre la Aplicación Cliente y la capa de SIG.

A cada clase que se utiliza para representar los elementos de la Aplicación Cliente y se desea georeferenciar se les asocia una clase que los decora para agregarle la información geográfica correspondiente a la Capa SIG. Estas clases que decoran a los objetos del dominio son: GeoempresaAgrícola, GeoLote y GeoEquipoRiego. De esta forma se puede notar la flexibilidad con que se cuenta para poder decidir cuáles elementos georeferenciamos y cuáles no.

Como vimos anteriormente, cada "geoObjeto" tendrá asociada una Location, encargada de dar la posición, y una Topología, encargada de dar la forma de los objetos en el espacio. Cada geobjeto puede definir su posición de acuerdo al sistema de referencias que sea más conveniente.

La clase GeoEmpresaAgrícola tendrá asociadas diferentes estructuras de indización, de acuerdo a la información que se necesite mantener en la Aplicación Cliente. Entonces, para mantener indizados los geoObjetos se debe agregar cada uno a las estructuras de indización correspondientes.

### 9.3.6. Interacción de la Aplicación Cliente con la Capa de SIG

En esta sección mostraremos ejemplos de cómo interactúan los objetos de la Aplicación Cliente con los objetos de la Capa de SIG.

El primer ejemplo muestra la manera de agregar un lote a la empresa. Para entender el mismo, enumeramos los pasos necesarios para llevar a cabo dicha tarea:

1. El Usuario elige el Sistema de Referencias, en el cual ingresará las coordenadas del lote.
2. El Usuario ingresa los puntos que conforman el perímetro del lote.
3. El Usuario ingresa la información de negocio asociado al lote.
4. El Sistema delega al geoObjeto asociado a la empresa agrícola la creación de todo lo relacionado al lote. Para esto la geoEmpresaAgrícola debe:
    a. Delegar al objeto que representa la empresa:
        i. la creación del lote asociándole la información de negocio
        ii. su agregación a la colección de lotes pertenecientes a la empresa
    b. Crear la geometría asociada. En este caso una instancia de la clase Polygon.
    c. Crear la posición, instancia de la clase Location, y asociarle la geometría y el sistema de referencias
    d. Crear el objeto georeferenciado asociado al lote, instancia de GeoLote, asociándole la posición construida anteriormente.
    e. Agregar el geoObjeto creado anteriormente a la estructura de indización, en este caso una instacia de RTree

A continuación se muestra un diagrama de secuencia que permite visualizar la secuencia de pasos anteriormente detallada:

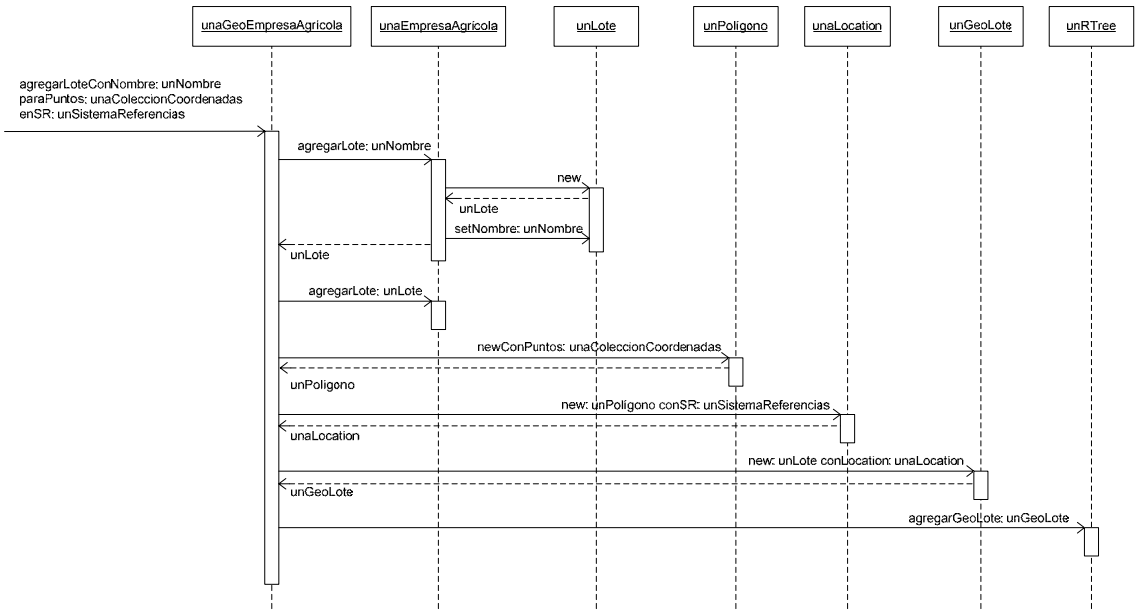



**Figura 9.7**.- Diagrama de secuencia para el agregado de un lote en la Aplicación Cliente.

El mensaje inicial es enviado desde la interfaz de la Aplicación Cliente, con la cual interactúa el usuario. A partir de ahí comienza la interacción para crear los objetos. Los cuales luego se podrán visualizar desde esa aplicación.

De la misma forma que se agregó la información de dominio y geográfica de un lote se puede agregar la información relacionada a un equipo de riego.

Una vez que todos los elementos de la empresa agrícola fueron creados y son visualizados, se pueden realizar operaciones de búsqueda y geométricas entre ellos.

El siguiente ejemplo muestra la búsqueda de objetos que contienen espacialmente a un punto ingresado por el usuario. Para entender el mismo, enumeramos los pasos necesarios para llevar a cabo tal tarea:

1. El Usuario ingresa la coordenada de consulta.
2. El Usuario ingresa el Sistema de Referencias en el cual debe interpretarse la coordenada.
3. El Sistema delega al geoObjeto asociado a la empresa agrícola (geoEmpresaAgrícola) la búsqueda de los objetos que se encuentren en el punto ingresado.
4. La geoEmpresaAgrícola crea un objeto Location a partir de la coordenada ingresada de modo de representarla en términos de los elementos del modelo.
5. La geoEmpresaAgrícola delegar al índice (en este caso un RTree) la búsqueda de los objetos en la Location creada en el paso anterior.
6. El Rtree consulta su estructura interna, delegando la consulta en su nodo raíz (se asume una estructura de altura 1, para mayor simplicidad).
7. El nodo raíz testea si la Location de consulta está contenida en el bounding box de alguno de sus elementos, encontrando que un bounding box la contiene.
8. Se realiza un segundo testeo para determinar si el elemento cuyo boundix box contiene el punto, también la contiene o no (pues puede ocurrir que esté incluido en el bbox y no en el elemento geográfico). Este testeo de inclusión es delegado al elemento, en este caso geoLote1.
9. geoLote1 delega el testeo en su topología, en este caso, un SimplePolygon. Este último responde verdadero a tal testeo.
10. El retorno de los envíos de mensajes termina devolviendo al usuario el geoLote1 como un elemento que contiene espacialmente a la coordenada ingresada.

A continuación se muestra un diagrama de secuencia que permite visualizar la secuencia de pasos anteriormente detallada:

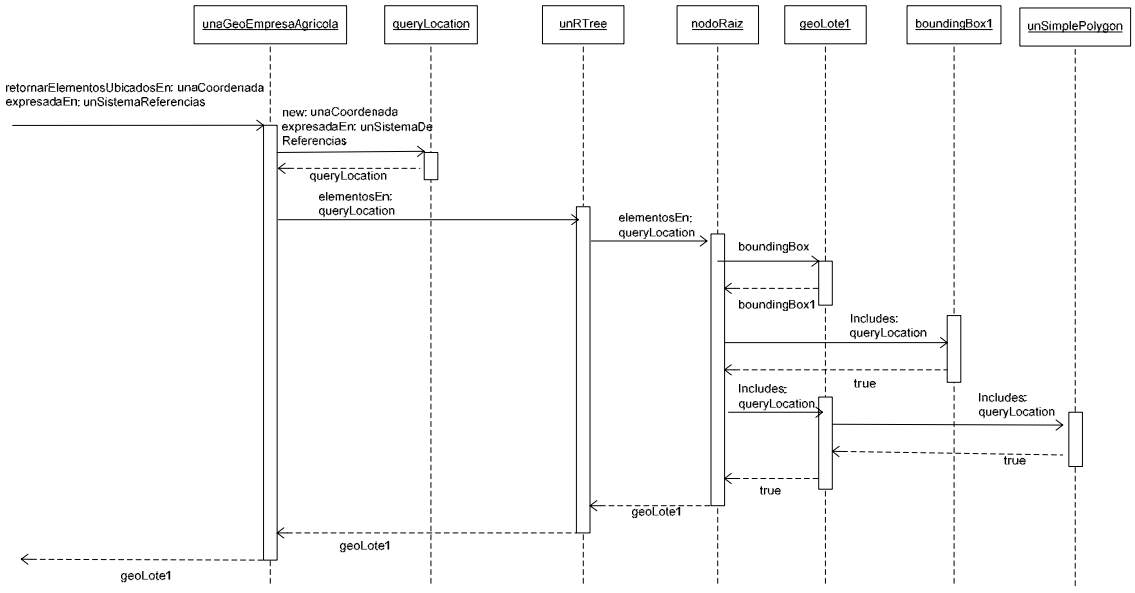



**Figura 9.8**.- Diagrama de secuencia para la búsqueda de objetos en la Aplicación Cliente.

Las operaciones geométricas son llevadas a cabo por la topología de cada geoObjeto. Entonces cuando se seleccionan los dos operandos desde la aplicación, se les delega a éstos la realización de dicha operación.

## 10. CONCLUSIONES Y TRABAJOS FUTUROS

Los Sistemas de Información Geográfica tienen un espectro de aplicación muy amplio. Diversas disciplinas, incluso muchas que poseen muy poca relación entre sí, hacen uso de SIG para enriquecer sus aplicaciones con procesamiento de información geográfica. Esta necesidad debe ser satisfecha para todas ellas, sin embargo, la gran diversidad de dominios de aplicación y los diferentes requerimientos que éstos demandan, dificultan la tarea de definición de un modelo de gestión geográfico que cubra todas las expectativas. Cada dominio, e incluso cada aplicación, tienen aspectos que le son propios y al que no se ajustan soluciones generales, por lo que deberían ser abordados con soluciones muy determinadas e incluso únicas, definidas especialmente para ese caso en particular.

Mediante este trabajo hemos definido el modelo de una capa de SIG genérica que pueda proveer funcionalidad de gestión de información geográfica a aplicaciones de diversos dominios. Para esto, primero hemos intentado detectar cuáles son los puntos en común que tienen todas las aplicaciones que gestionan información geográfica como parte de su operatoria, y para cada uno de ellos un objetivo específico, claro y determinado. Una vez detectadas y clasificadas estas partes, hemos estudiado las diferentes maneras en que se implementa cada una de ellas, es decir, las diferentes maneras en que puede lograrse un mismo objetivo. Por último, se logró abstraer una definición general y la funcionalidad de cada parte de modo de lograr una interfaz común a varias implementaciones de modo de que éstas sean intercambiables.

Estas partes o módulos detectados, que constituyen la capa de SIG definida, son: Sistema de Referencias, Topología, Índices de Datos Geográficos y Operaciones de Geometría Computacional. La gran mayoría de las aplicaciones SIG necesitan de estos módulos, pero cada una de ellas puede requerir una implementación particular de algunos de ellos para cumplir sus objetivos. Además, debe asegurarse que cualquier combinación de módulos entre sí funcione correctamente más allá de la forma en que se haya implementado cada uno. Dado esto último y que, además, estos módulos deben dar soporte a aplicaciones de dominios muy diferentes, es que surge la necesidad de un muy alto grado de abstracción. Es precisamente debido a esto último que hemos utilizado el Paradigma de Orientación a Objetos; éste ha demostrado ser particularmente útil para el diseño de esta capa SIG, que presenta tales características.

El diseño Orientado a Objetos asegura, mediante la definición de contratos y polimorfismo, que cualquier combinación de implementación de módulos funcione correctamente; asimismo permite, mediante herencia de clases, la definición de comportamiento común a varias implementaciones, lo que permite una rápida extensión de la capa propuesta con otras nuevas. Además se han aplicado varios Patrones de Diseño, lo que ha contribuido aún más a lograr un buen grado de flexibilidad en cuanto a configuración y extensión.

En cuanto a los módulos concretos, el Sistema de Referencia provee una manera genérica de posicionar y localizar objetos geográficamente además de permitir ciertos cálculos como ser distancias y áreas. El modelo permite la convivencia de distintos sistemas de referencias, es decir que puede operarse con objetos cuya posición se expresa en distintos sistemas de referencias.

La Topología define una manera flexible de operar con las formas y relaciones geográficas entre los objetos, quedando abierto el modelo a la definición e incluso la convivencia con otros modelos muy utilizados, como el de raster.

En cuanto al modelo de Indización, se han definido los casos concretos de Quad-Tree y RTree, este último incluso ha sido implementado. Estas estructuras de indización se han definido siempre dentro de un marco genérico (interfaces, contratos) que permite la definición e inclusión de otras estructuras de indización por parte del usuario final.

La capa SIG cuenta con un conjunto de Operaciones de Geometría computacional que se utiliza para realizar muchos cálculos geográficos complejos (intersección de polígonos, testeos de inclusión, cálculos de centroides o kernels). Estas operaciones también han sido diseñadas mediante un modelo de objetos que permite cambiar entre distintas implementaciones de acuerdo a la conveniencia del caso particular. La performance lograda es bastante buena, e incluso ha sido mejorada mediante técnicas diseñadas para tal fin. En cualquier caso, de ser necesaria mayor performance, estas operaciones podrían ser implementadas como primitivas en algún lenguaje de bajo nivel y ser invocadas desde el modelo de objetos; esto no chocaría de ninguna manera con la flexibilidad del paradigma OO, gracias a que la definición de interfaces actual oculta los aspectos de implementación de tales operaciones.

En cuanto a trabajos futuros o a la forma en que podría extenderse este modelo, existen varias líneas de acción. Quizá una de las más importantes es la inclusión de funcionalidad de gestión de información temporal. La capa de SIG realizada en este trabajo permite la representación geográfica de

objetos sólo para un momento determinado (generalmente el presente), pero no permite realizar consultas sobre la historia geográfica de tales objetos; por ejemplo, se puede saber en dónde se encuentra un objeto en este momento pero no en dónde se encontraba en un momento anterior determinado. Al respecto hemos estudiado algunas implementaciones de estructuras de indización temporales, por ejemplo el R-Tree Temporal [Nascimento], que podrían incluirse en el modelo adaptando las interfaces del mismo de manera de que queden lo suficientemente genéricas para lograr flexibilidad en cuanto a extensión con otras nuevas.

## 11. BIBLIOGRAFÍA

[Aronoff] S. Aronoff. *Geographic Information Systems*. WDL Publications, Canada, 1989

[Balaguer] F. Balaguer, S. Gordillo, F. Das Neves. *Patterns for GIS Applications Design*. Proceedings of Patterns Language of Programming 1997

[Beckmann] N. Beckman, H.P. Kriegel, R. Schneider y B. Seeger. *The R*-Tree: An efficient and Robust Access Method for Points and Rectangles.* Proc. ACM SIGMOD, páginas 322-331, Mayo de 1990.

[Bentley] J. L. Bentley. *Multidimensional binary search trees used for associative searching*. Communications of the ACM 18, 1975.

[Faloustos] Ibrahim Kamel y Christos Faloustos. *On Packing R-Trees.*

[Fink] R.A. Finkel, J.L. Bentley. *Quad trees: a data structure for retrieval on composite keys.* Acta Informática 4, 1974.

[Gamma] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design patterns. Elements of reusable Object-Oriented Software*. Adison Wesley, 1995

[Gordillo] S. Gordillo, F. Balaguer, F. Das Neves. *Generating the Architecture of GIS Applications with Design Patterns.* 5th ACM Workshop on Geographic Information Systems. Las Vegas, USA.November 1997.

[Guttman] Antonin Guttman. *R-Trees: A Dynamic Index Structure for Spatial Searching*. Proc. ACM SIGMOD, páginas 47-57, Junio de 1984.

[Lliteras] A. Lliteras; S. Gordillo. *Sistemas de referencias para aplicaciones SIG´s*. VI Congreso Argentino de Ciencias de la Computación CACIC 2000, Octubre de 2000.

[Knuth] The Art of Computer Programming, vol 3, Sorting ans Searching, Adison Weslwey, 1973

[Lopez] Mario A. Lopez. Mathematics and Computer Science, Universidad de Denver, Estados Unidos. *Curso sobre Geometría Computacional para alumnos e investigadores de Informática*, dictado en la Facultad de Ciencias Exactas de la Universidad Nacional de La Plata.

[Martin] Charla con Ing. Agr. E. Martin. *Cediagro, Salto, Pcia. Bs As., noviembre de 2006*

[Medeiros] Dra. C. Bauzer Medeiros. *"Curso de Bases de datos geográficas: aspectos teóricos y prácticos"*, dictado en la 15ta. Escuela de Ciencias Informáticas. Departamento de Computación de la Facultad de Ciencias Exactas y Naturales de la Universidad de Buenos Aires, Julio 2001

[Nascimento] M.A. Nascimento and J.R.O. Silva. *Towards Historical R-trees*, Proceedings 13th ACM SAC Symposium, páginas.235-240, Atlanta, GA, 1998.

[Notes on GPS] Notes on GPS, Geodetic Datums and Gris\Lat & Long Convertions. http://www.aerobaticsweb.org/SOARING//BGA/os-note.html

[OpenGIS Consortium] http://www.opengeospatial.org/resource/glossary/d

[Perdomo] R. Perdomo. *Notas sobre Coordenadas y Sistemas de Referencia.* Facultad de Ciencias Astronomicas y Geodesicas, Universidad Nacional de La PLata, Oct. 1994.

[Preparata] Franco P. Preparta y Michael Ian Shamos, Computacional Geometry. Springer-Verlag, 1985.

[Prieto] Ing. Agr. Daniel Prieto, Ing. Agr.Carlos Yañez. *Curso de Métodos de riego a distancia.* ISBN: 950-9853-64-X, 1996

[Rebecca] Rebecca Wirs-Brock, Brian Wilkerson, Lauren Wiener. *Designing Object – Oriented Software*. Prentice Hall PTR, Englewood Cliffs, New Jersey. ISBN 0-13-629825-7

[Roussopoulos] Roussopoulos y D. Leiker. *Direct Spatial Search on Pictorial Data Bases using Packed R-Trees.* Proc. ACM SIGMOD, Mayo de 1985.

[Samet] H. Samet. *The Design and Analisis of Spatial Data Structures*. Addison-Wesley, 1989.

[Shaf] C. A.. Shaffer. *An empirical comparison of vectors, arrays, and quadtrees for representing geographic data*. Proceedings of the International Colloquium on the Construction and Display of Geoscientific Maps Derived from Databases, Alemania, 1986.

[Snyder] John P. Snyder. *Map Projections, A working Manual*. DC: US Govt

[Voser98] Coordinate Reference System Management within GIS -A perspective. Stefan A. Voser

## 12. PUBLICACIONES RELACIONADAS

***OCTANTE: Una herramienta para el cálculo interactivo de distancias y áreas.***
Presentado en JAIIO 98 como trabajo estudiantil de pregrado. Buenos Aires, Argentina.

***OCTANTE: a Tool for Interactive Calculations of Distance and Áreas.***
S. Gordillo, F. Balaguer, J. Bazzocco, D. Cano, G. Trilla.
Presentado en URISA'98. Carolina del Norte, EE.UU.

***Construcción de una Herramienta para el Cálculo de Distancias y Áreas en el Ámbito Geográfico con Tecnología de Objetos***.
D. Cano, J. Bazzocco, G. Trilla.
Presentado en 27 JAIIO. Buenos Aires, Argentina.

***An Object Oriented Approach to Geo-referencing and Visualization.***
J. Bazzocco, D. Cano, G. Trilla, S. Gordillo, F. Balaguer.
Presentado en SIRC '98. Dunedin, Nueva Zelanda.

***The Appearance: a Design Pattern for Continuous Field Interfaces.***
G. Trilla, J. Bazzocco, S. Gordillo.
Presentado en DEXA '99. Florencia, Italia.

***Una Arquitectura Orientada a Objetos para Sistemas de Información Geográfica.***
A. Lliteras, D. Cano, G. Trilla, J. Bazzoco, A. Zambrano, L. Polasek, S. Gordillo, C. Mostaccio.
Presentado en el Workshop de Investigadores en Ciencias de la Computación. Mayo 2000. La Plata, Argentina.

***Taking advantage of Design Patterns in GIS domain.***
L. Polasek, A. Zambrano, S. Gordillo, J. Bazzocco, G. Trilla, D. Cano, A. Lliteras.
Presentado en OOPSLA 2000. Minneapolis, USA.

***Tilcara: An OO Perspective to Handle Continuous Fields in GIS.***
A. Zambrano, L. N. Polasek, S. Gordillo, J. Bazzocco, G. Trilla, D. Cano, A. Lliteras.
Presentado en OOPSLA 2000. Minneapolis, USA.

# 13. APÉNDICE A: ALGORITMOS PARA LA GESTIÓN DE R-TREES

En este apéndice se describen en detalle los algoritmos necesarios para la gestión de un R-Tree, a saber: búsqueda, inserción, borrado y modificación de elementos. Además se incluye uno referente a la generación de carga masiva que da por resultado un R-Tree empaquetado.

Los algoritmos que aquí se presentan constituyen las versiones originales y pertenecen al paradigma estructurado. Para la versión de R-Trees incluída en el modelo desarrollado, se han adaptado estos algoritmos al paradigma Orientado a Objetos.

## 13.1. Terminología y Notación

Para un mejor desarrollo y claridad de exposición de estos algoritmos, se ha establecido una terminología que se explica a continuación. Cabe recordar que a nuestros efectos, una Base de Datos espacial consiste en un conjunto de *tuplas*, cada una con un objeto espacial y un identificador único utilizado para retornar tal objeto.

Los *nodos hoja* de un R-Tree contienen entradas de registros de índice del tipo *( I, identificador de tupla)*, donde el *identificador de tupla* corresponde al identificador de tupla de la Base de Datos que contiene al objeto espacial que el registro de índice representa; por su parte, el atributo *I* conforma el bounding-box n-dimensional del objeto espacial que el registro índice representa. Este atributo *I* se define de la siguiente manera:

$$I = (I_0, I_1, \ldots , I_{n-1})$$

en donde n es el número de dimensiones y cada $I_i$ es un intervalo cerrado [a,b] que representa la extensión del bounding-box en la dimensión i.

Los *nodos no hoja* de un R-Tree contienen entradas de registros de índice del tipo *( I, identificador de hijo)*, donde el *identificador de hijo* corresponde al identificador de un nodo perteneciente al nivel inmediatamente inferior en el R-Tree, e *I*, definido de la manera antes descripta, representa el bounding-box n-dimensional que cubre espacialmente a todos los bounding-boxes de las entradas del nodo hijo.

Las propiedades que debe cumplir un R-Tree se han descripto en el capítulo correspondiente por lo que no serán repetidas aquí.

Establecidas estas definiciones y propiedades, en los algoritmos siguientes se utilizarán las siguientes denotaciones:

- *M:* cantidad máxima de entradas que puede tener un nodo.
- *m:* cantidad mínima de entradas que puede tener un nodo.
- EI: atributo I de una tupla E = (I,P).
- Ep: atributo P de una tupla E = (I,P).

## 13.2. Algoritmos de Búsqueda

### 13.2.1. Algoritmo Search

Dado un R-Tree cuyo nodo raíz es T, encuentra todos los registros índices (que referencian a tuplas de la Base de Datos) cuyos rectángulos se superponen con el rectángulo S dado como parámetro.

S1[Buscar subárboles] Si T no es una hoja, chequear cada entrada E para determinar si EI se superpone con S. Para cada entrada que se superpone con S, invocar a Search recursivamente con el árbol cuyo nodo raíz es referenciado por Ep.

S2[Buscar nodo hoja] Si T es una hoja, chequear todas las entradas E para determinar si EI se superpone con S. En ese caso devolver E, pues es uno de los registros buscados.

## 13.3. Algoritmos de Inserción

### 13.3.1. Algoritmo Insert

Inserta una nueva entrada de índice, E, en el R-Tree.

I1[Encontrar la posición para el nuevo registro] Invocar **ChooseLeaf** para seleccionar un nodo hoja L en el cual ubicar a E.

I2[Agregar registro al nodo hoja] Si L tiene lugar para una entrada más, poner a E en L. En caso contrario, invocar a **SplitNode** para obtener L y LL conteniendo a E y a todas las viejas entradas de L.

I3[Propagar los cambios hacia arriba] Invocar **AdjustTree** con L, también pasando a LL como argumento si es que ha habido una partición.

I4[Agrandar árbol en altura] Si alguna eventual propagación de partición de nodos ha llegado a la raíz causando también su partición, crear una nueva raíz y asociarle como hijos a los dos nodos resultantes de la partición.

### 13.3.2. Algoritmo ChooseLeaf

Selecciona un nodo hoja en el cual ubicar una nueva entrada E.

CL1[Inicializar] Setear N como el nodo raíz.

CL2[Chequear si es hoja] Si N es una hoja, devolver N.

CL3[Elegir subárbol] Si N no es un nodo hoja, sea F la entrada en N cuyo rectángulo FI necesita el menor crecimiento para incluir a EI. Los empates se resuelven eligiendo el rectángulo de menor área.

CL4[Descender hasta una hoja] Setear a N para que sea el nodo hijo apuntado por Fp y repetir desde CL2.

### 13.3.3. Algoritmo AdjustTree

Asciende desde un nodo hoja L hasta la raíz, ajustando los rectángulos que cubren a sus hijos y propagando particiones de nodos cuando sea necesario.

AT1[Inicializar] Setear N = L. Si L fue partido previamente, setear NN para representar el segundo nodo resultante.

AT2[Chequear terminación] Si N es la raíz, terminar.

AT3[Ajustar ractángulo cobertor en la entrada padre] Sea P el nodo padre de N, y sea $E_N$ la n-ésima entrada en P. Ajustar $E_N$I de manera que encierra de la manera más ajustada posible a todos los rectángulos de las entradas en N.

AT4[Propagar partición de nodo hacia arriba] Si N tiene un nodo NN "hermano" resultante de una partición anterior, crear una nueva entrada $E_{NN}$ con $E_{NN}p$ apuntando a NN y $E_{NN}I$ abarcando espacialmente a todos los rectángulos en NN. Agregar $E_{NN}$ a P si es que hay lugar. De lo contrario, invocar a **SplitNode** para producir P y PP conteniendo a $E_{NN}$ y a todas las viejas entradas de P.

AT5[Pasar al nivel inmediato superior] Setear N = P, y también NN = PP si hubo alguna partición. Repetir desde AT2.

## 13.4. Algoritmos de Partición de Nodos

Como se describe en la sección correspondiente, existen varias maneras de partir un nodo redistribuyendo sus entradas con otro/s. Aquí se listan sólo dos de las mencionadas: la de orden cuadrático y la de orden lineal, ambas son particiones de un nodo a dos (recordar que también existen estrategias de partición de dos a tres).

### 13.4.1. Algoritmo Quadratic Split

Divide un conjunto de M + 1 entradas en dos grupos, con orden de ejecución cuadrático.

QS1[Escoger las dos primeras entradas de cada grupo] Aplicar el algoritmo **PickSeeds** para elegir dos entradas de manera de que cada una constituya el primer elemento de cada grupo. Asignar cada una a un grupo.

QS2[Chequear terminación] Si todas las entradas han sido asignadas, terminar. Si un grupo tiene tan pocas entradas al punto tal que todo el resto de las entradas deben asignársele de manera que alcance la cantidad mínima m, entonces asignárselas y terminar.

QS3[Seleccionar la entrada a asignar] Invocar el algoritmo **PickNext** para seleccionar la próxima entrada a asignar. Agregarla al grupo cuyo rectángulo cobertor debe ser agrandado en menor área al incorporarla. Resolver los empates agregándola al rectángulo de menor área, luego al que tenga menos entradas, luego a cualquiera de los dos. Repetir desde QS2.

### 13.4.2. Algoritmo PickSeeds

Selecciona dos entradas que constituyan, cada una, el primer elemento de cada grupo.

PS1[Calcular la ineficiencia de agrupar las entradas juntas] Para cada par de entradas $E_1$ y $E_2$, armar un rectángulo J que incluya $E_1I$ y $E_2I$. Calcular d = area(J) – area($E_1I$) – area($E_2I$).

PS2[Selecciona el par que más área desperdicie] Elegir el par que presente el d máximo.

### 13.4.3. Algoritmo PickNext

Selecciona una de las entradas del conjunto de las aún no asignadas para ser asignada a algún grupo.

PN1[Determina el costo de asignar cada entrada en cada grupo] Para cada entrada E aún no asignada a ningún grupo, calcula $d_1$ = crecimiento de área necesario en el rectángulo que cubre al Grupo 1 para incluir EI. Calcular $d_2$ de la misma manera para el Grupo 2.

PN2[Encontrar la entrada con mayor preferencia para un grupo] Elegir cualquier entrada con la máxima diferencia entre $d_1$ y $d_2$.

### 13.4.4. Algoritmo Linear Split

Divide un conjunto de M + 1 entradas en dos grupos, con orden de ejecución lineal. Este algoritmo es lineal con respecto a M y al número de dimensiones. Es idéntico a QuadraticSplit pero utiliza una versión diferente de PickSeeds. Por su parte, PickNext sólo selecciona cualquiera de las entradas restantes. Por lo tanto sólo se describe la versión lineal de PickSeeds: LinearPickSeeds.

### 13.4.5. Algoritmo LinearPickSeeds

Selecciona dos entradas que constituyan, cada una, el primer elemento de cada grupo.

LPS1[Encontrar los dos rectángulos extremos considerando todas las dimensiones] Por cada dimensión, encontrar la entrada cuyo rectángulo tiene el lado inferior más grande, y el que tiene el lado mayor más chico. Almacenar esta distancia.

LPS2[Ajustar a la forma del cluster del rectángulo] Normalizar las separaciones dividiendo por el ancho del conjunto completo sobre la dimensión correspondiente.

LPS3[Seleccionar el par extremo] Elegir el par con mayor separación normalizada sobre cualquier dimensión.

## 13.5. Algoritmos de Borrado

### 13.5.1. Algoritmo Delete

Elimina el registro E del R-Tree.

D1[Encontrar el nodo que contiene el registro] Invocar **FindLeaf** para localizar el nodo hoja L que contiene a E. Terminar si el registro no fue encontrado.

D2[Borrar el registro] Borrar E de L.

D3[Propagar cambios] Invocar **CondenseTree** pasando L como argumento.

D4[Achicar el árbol] Si el nodo raíz tiene sólo un hijo luego de que el árbol haya sido ajustado, poner como nueva raíz del árbol a ese hijo, descartando la anterior.

### 13.5.2. Algoritmo FindLeaf

Dado un R-Tree con nodo raíz T, encuentra el nodo hoja que contiene la entrada E.

FL1[Buscar en subárboles] Si T no es hoja, chequea cada entrada F en T para determinar si FI se superpone con EI. Por cada una de esas entradas, invocar **FindLeaf** en el árbol cuya raíz es referenciada por Fp hasta que E sea encontrado o todas las entradas hayan sido chequeadas.

FL2[Buscar el registro en el nodo hoja] Si T es hoja, chequear cada entrada para ver si coincide con E. Si se ha encontrado E, retornar T.

### 13.5.3. Algoritmo CondenseTree

Dado un nodo hoja L del cual ha sido borrada una entrada, elimina el nodo si es que ha caído en underflow y reubica sus entradas. Propaga la eliminación de nodos hacia arriba según sea necesario. Ajusta todos los rectángulos en el camino hacia la raíz, reduciendo su cobertura de ser posible.

CT1[Inicializar] Setear N = L. Setear Q, el conjunto de nodos eliminados, como vacío.

CT2[Encontrar la entrada padre] Si N es la raíz, ir a CT6. En caso contrario, sea P el padre de N, y sea $E_N$ la entrada número N en P.

CT3[Eliminar nodo en underflow] Si N tiene menos de m entradas, borrar $E_N$ de P y agregar N al conjunto Q.

CT4[Ajustar el rectángulo de cobertura] Si N no ha sido eliminado, ajustar $E_N$I para que contenga de la manera más ajustada a todas las entradas en N.

CT5[Moverse al nivel próximo superior del árbol] Setear N = P y repetir desde CT2.

CT6[Reinsertar las entradas huérfanas] Reinsertar todas las entradas de los nodos en el conjunto Q. Las entradas de los nodos hoja eliminados son reinsertadas en las hojas del árbol según lo descripto por el algoritmo **Insert**, pero las entradas de nodos de niveles superiores deben ser reinsertadas en niveles más altos del árbol, de manera que las hojas de sus subárboles dependientes estén al mismo nivel que las hojas del árbol principal.

## 13.6. Algoritmos de Modificación

La modificación de un objeto se limita sólo a eliminar su entrada en el R-Tree mediante el método de borrado descripto, y luego a volver a insertarlo con su nueva forma mediante el método usual de inserción. De esta manera encuentra su lugar correcto en el R-Tree.

## 13.7. Algoritmo de Empaquetamiento mediante la curva de Hilbert

El siguiente algoritmo realiza la carga masiva de un R-Tree con los bounding-boxes de un conjunto de objetos mediante una técnica de ordenamiento de nodos basada en fractales, en particular, la curva de Hilbert. El resultado es un *R-Tree empaquetado*.

**Algoritmo Hilbert –Pack:** Empaqueta rectángulos en un R-Tree.

Paso 1: Calcular el valor de Hilbert para cada rectángulo *dato*.
Paso 2: Ordenar los rectángulos ascendentemente según sus valores de Hilbert.
Paso 3: /* *Crear nodos hoja (nivel L = 0)* */
Mientras (hay más rectángulos)
    Generar un nuevo nodo.
    Asignar los siguientes *M* rectángulos a este nodo.
Paso 4: /* *Crear nodos a niveles superiores (L + 1)* */
Paso 5: Mientras (hay más de un nodo en el nivel L)
Ordenar los nodos del nivel L >= 0 en tiempo de *creación* ascendente.
Repetir paso 3.
Paso 6: FIN.

# 14. APÉNDICE B: PATRONES DE DISEÑO

En este apéndice se describe la intención de cada patrón de diseño utilizado para realizar este modelo orientado a objetos. Para más información se puede consultar según lo indicado en [Gamma].

## 14.1. Patrones utilizados

### 14.1.1. Decorator

Agrega responsabilidades adicionales a un objeto en forma dinámica. Los decoradores proveen una alternativa flexible a la subclasificación como forma de extender la funcionalidad.

### 14.1.2. Mediator

Define un objeto que encapsula como interactúan un conjunto de objetos. Este patrón promueve el bajo acoplamiento al evitar que los objetos se refieran entre sí explícitamente, y permite variar la interacción en forma independiente.

### 14.1.3. Strategy

Define una familia de algoritmos, los encapsula y los hace intercambiables entré sí. Este patrón permite a los algoritmos variar independientemente de los clientes que los utilizan.

### 14.1.4. Singleton

Asegura que una clase tenga una sola instancia, y provee un punto global de acceso a ésta.

### 14.1.5. Template Method

Define un esqueleto de un algoritmo en una operación, delegando algunos pasos a las subclases. Este patrón permite a las subclases redefinir algunos pasos sin cambiar la estructura del algoritmo.

### 14.1.6. Composite

Compone objetos en estructuras de árboles para representar jerarquías “contenedor-parte”. Este patrón permite a los clientes tratar a los objetos individuales y a las composiciones de objetos en forma uniforme.

### 14.1.7. Command

Encapsula un requerimiento como un objeto, por lo que permite parametrizar a los clientes con varios requerimientos, encolar o almacenar estos requerimientos y dar soporte a operaciones que se pueden volver atrás.

### 14.1.8. Observer

Define una dependencia uno a muchos entre objetos de manera que cuando un objeto cambia, todos sus dependientes son notificados y actualizados automáticamente.

# 15. APÉNDICE C: PATRÓN DE DISEÑO "APPEARANCE"

En este apéndice se detalla el patrón de diseño utilizado para realizar la visualización de la aplicación mostrada en el Capítulo VIII.

## 15.1. Descripción del patrón de diseño

La siguiente figura muestra la arquitectura que fue desarrollada para implementar el patrón Appearance:

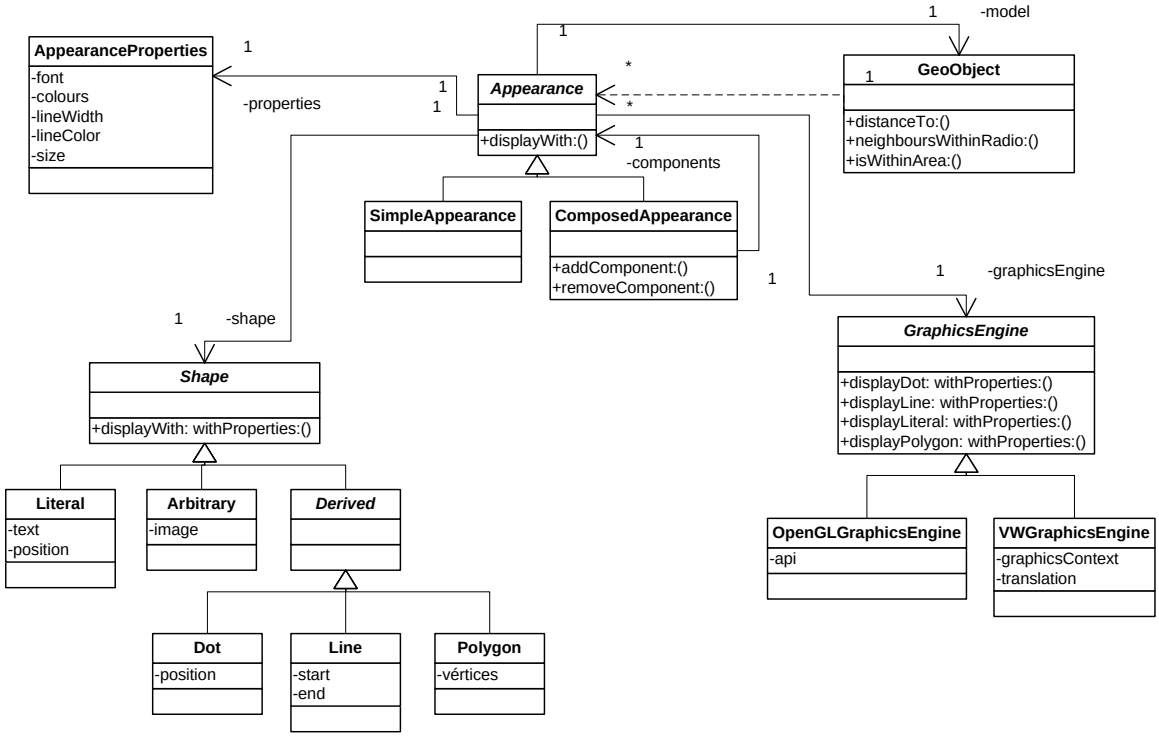



**Figura 1**.- Modelo, orientado a objetos, del patrón Appearance

A continuación detallamos las principales características de este modelo:

- Con el fin de obtener un diseño para la visualización lo suficientemente flexible, se desacopla una entidad geográfica de la responsabilidad de representarse gráficamente. La clase **Appearance** es la responsable de manejar todos los aspectos relacionados con la visualización de un **GeoObject**.
- Para mantener consistente la apariencia ante los cambios en la entidad geográfica, es necesario que la apariencia y ésta se mantengan relacionadas. Esto se logra mediante la aplicación del patrón de diseño Observer [Gamma]. Es la relación entre la clase Appearance y GeoObject que se encuentra en la figura anterior, con línea de forma punteada.
- Se utiliza el patrón de diseño Composite [Gamma] para permitir considerar tanto a las apariencias como una unidad, cuando esto sea necesario, y que también brinde la posibilidad de tratar a cada una de las partes componentes individualmente. Gracias a este diseño, podemos asignar una apariencia compuesta a un determiando **GeoObject**, que representa una isla (considerando la isla como una unidad) y una apariencia simple a cada uno de los polígonos que la componen (en este caso triángulos). Ver figura 2.

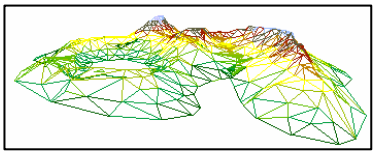


**Figura 2**.- Apariencia compuesta para una entidad geográfica

- Una apariencia deberá mostrar al **GeoObject** de acuerdo a ciertas propiedades como ser color, ancho de línea, nivel de definición, etc. Para muchas entidades geográficas estas propiedades tendrán un valor por defecto y no cambiarán muy seguido. Debido a la gran cantidad de objetos que pueden llegar a manejarse en un SIG, resulta sumamente útil separar la apariencia de las propiedades antes mencionadas. Así, todas las apariencias que deben ser mostradas con las mismas propiedades compartirán la misma instancia de la clase **AppearanceProperties**, y sólo aquellas apariencias que deban tener alguna distinción especial tendrán su propia instancia. Esto conduce a un importante ahorro en almacenamiento y en requerimientos de memoria.
- Una apariencia también debe contar con información acerca de la "forma" que posee. Mantener dicha información será responsabilidad de la clase **Shape**. Cabe destacar que en principio existen tres tipos fundamentales de "formas", dichos tipos son:
  - *Literal*: en este caso se trata de un texto descriptivo asignado por el usuario.
  - *Arbitraria*: se asigna una imagen arbitraria a la forma de un **GeoObject** sin considerar aspectos tales como la forma real que posea, la escala en la que se encuentra, etc.
  - *Derivada*: la forma se deriva de la topología del **GeoObject**, con lo cual es necesario definir tres tipos de formas derivadas: punto, línea y polígono.

  Las siguientes figuras muestran un ejemplo de cada tipo de forma posible:


(a) (b) (c)

**Figura 3**.- Formas posibles de una apariencia. (a) Literal. (b) Arbitraria. (c) Derivada de la Topología

- Con el objetivo de que una apariencia sea totalmente independiente del sistema encargado de dibujarla, se eliminó de las responsabilidades de una apariencia la responsabilidad de conocer los detalles relativos a las implementaciones de los distintos sistemas de graficación existentes, como ser *OpenGL*. Para esto se aplicó el patrón de diseño Strategy [Gamma], definiendo así la clase **GraphicsEngine**. Esta clase es la responsable de mostrar las apariencias en un sistema en particular. Cada sublclase de **GraphicsEngine** será una implementación de los algoritmos de renderizado para un sistema dado.
- Como consecuencia de la gran cantidad de objetos que coexisten en un SIG, es necesaria una manera de automatizar la creación de apariencias a partir de un conjunto de **GeoObjects**. Precisamente ésta será la responsabilidad de la clase **AppearanceBuilder**. Dicha clase no forma parte de este patrón pero permite definir múltiples políticas de creación de apariencias, como ser definir qué atributo del **GeoObject** considerar para asignar colores.

### 15.1.1. Integración del patrón de diseño con la Topología

El geoObject, actuando como modelo de una apariencia, es el que establece el nexo entre las arquitecturas citadas para la Topología y para la Apariencia.

Los siguientes diagramas de interacción muestran la manera en que colaboran las apariencias y las topologías, al momento de seleccionar un objeto desde la interface, por ejemplo mediante un click. Cabe destacar que el testeo de inclusión del punto de selección es a nivel de interface, es decir dicho testeo es llevado a cabo sobre la representación visual del objeto. Las topologías no se verán involucradas en este proceso de selección salvo en el caso en que la representación del geoObject sea derivada de la topología (Derived). En los otros casos, es el shape (Literal o Arbitrary) el encargado de responder si contiene el punto o no.

En la siguiente figura se muestra un diagrama de secuencia que muestra los pasos seguidos al testear si la apariencia de un geobjeto contiene un punto dado. En este caso la apariencia tiene un shape que no es derivado de la topología.

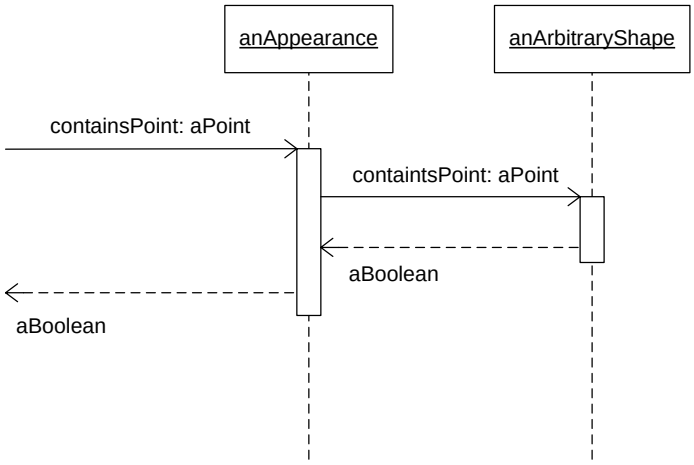



**Figura 4**.- Diagrama de secuencia para el testeo de inclusión de punto en una apariencia no derivada de su posición

El siguiente diagrama de secuencia muestra los pasos seguidos al testear si la apariencia de un geobjeto contiene un punto dado. Pero, en este caso la apariencia tiene un shape que es dependiente de la topología del geobjeto.

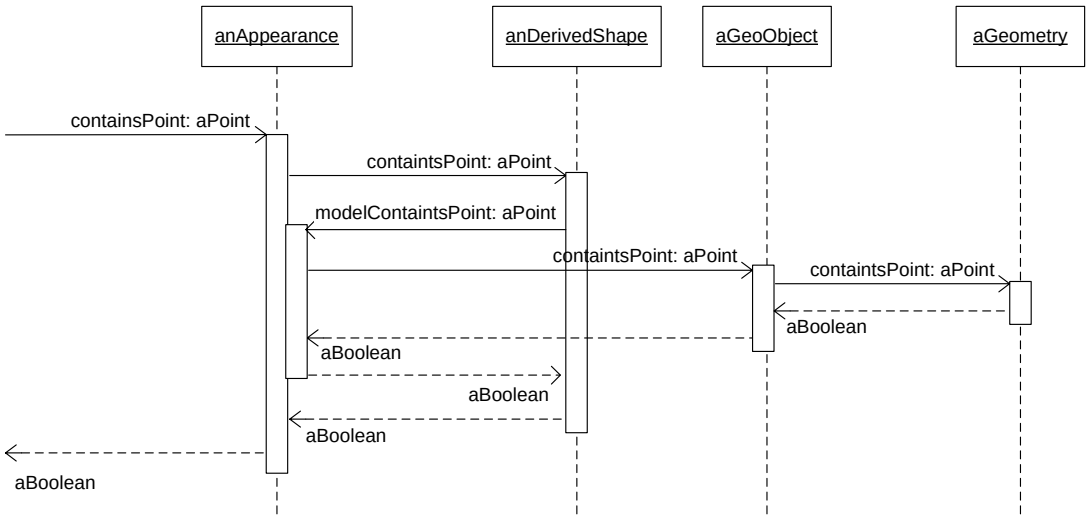



**Figura 5.-** Diagrama de secuencia para el testeo de inclusión de punto en una apariencia derivada de su posición