Centro de Investigación y
de Estudios Avanzados
del IPN

Departamento de Ingeniería Eléctrica

Sección de Computación

Reconocimiento, Interpretación y Visualización Espacial de Superficies Terrestres en Base a Fotografías Estereoscópicas

Tesis de Maestría

Luis Cruz Romo

Supervisada por
Dr. Luis Gerardo de la Fraga

Octubre 2003

Este trabajo fue escrito en LaTeX[20][8]

Resumen

En la rama de la Ingeniería Civil conocida como "fotogrametría" se obtienen las curvas de nivel de cierta parte de la superficie terrestre mediante una serie de fotografías estereoscópicas; esta actividad se efectúa en forma manual, quedando involucrados operadores humanos, hecho que hace a este procedimiento tardado, por consiguiente caro, y poco preciso.

En este trabajo se presenta un desarrollo para la obtención automática de las curvas de nivel a partir de pares de fotografías estereoscópicas. Se describe el proceso de obtención de las curvas y el resultado obtenido al procesar imágenes reales.

Dentro del sistema desarrollado se presenta una interfaz de usuario amigable para que el ingeniero responsable de la obtención de las curvas de nivel pueda observarlas en tercera dimensión.

Abstract

There is a field in the Civil Engineering area known as “photogrammetry”; its purpose is to get the Earth surface’s digital elevation model (DEM) using only a series of stereoscopic photographs. This activity is manually done by human operators, this fact makes this process error prone and expensive, and as such their results are low precise.

This work presents a system to automatically get those DEM by means of a stereoscopic photographic pair. The process of getting that DEM and some results of processing a couple of real pairs are shown.

Inside the system is a friendly graphical user interface in which the engineer responsible of drawing the DEM could see it in a 3D model.

Agradecimientos

Al CINVESTAV

sin cuya existencia no se hubiera completado esta etapa en mi crecimiento profesional.

Al CONACyT

por haber proporcionado los recursos necesarios que concluyen con este trabajo.

A mi esposa: **Alicia Martínez Calderón**

quien me ha demostrado que el amor puede romper las barreras del pensamiento.

A mi hijo: **Aarón Cruz Martínez**

de quien he aprendido que no es posible aprender ni mejorar sin divertirse.

A mis padres: **Luis Cruz Ruvalcaba**
y **Carmen Romo Reynoso**

por que sin ellos nada de esto sería posible.

A todas aquellas personas de las que he aprendido algo y me ha permitido mejorar como persona y profesional.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La rama de la ingeniería civil encargada de la obtención de las curvas de nivel mediante pares de fotografías estereoscópicas se le conoce como *fotogrametría*[1]. Esta disciplina utiliza fotografías tomadas por aviones que recorren el terreno, en el que se desea realizar la obra civil, sobre una línea determinada del mismo tal y como puede apreciarse en la Fig. 1.1

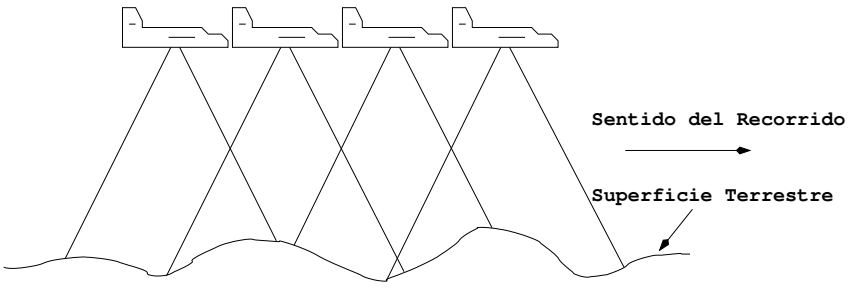

Figura 1.1: Recorrido del avión sobrevolando la superficie del terreno de interés

Las fotografías son tomadas dos a dos para formar un par estereoscópico con el cual, mediante aparatos ópticos, podrán ser obtenidas las curvas de nivel. Fundamentalmente, el proceso seguido para llevar a cabo esta tarea se divide en tres partes:

- Visualización óptica del par estereoscópico.
- Trazado de las curvas de nivel.
- Dibujo de las curvas obtenidas en CAD.

Visualización óptica del par estereoscópico

El operador humano se coloca en la máquina óptica (llamada estereoscopio), en donde se ha colocado el par estereoscópico y con ella se le permite ver una de las imágenes con el ojo izquierdo y la otra con el derecho. De esta forma el operador puede *ver* la superficie del terreno en tercera dimensión como se muestra en la Fig. 1.2

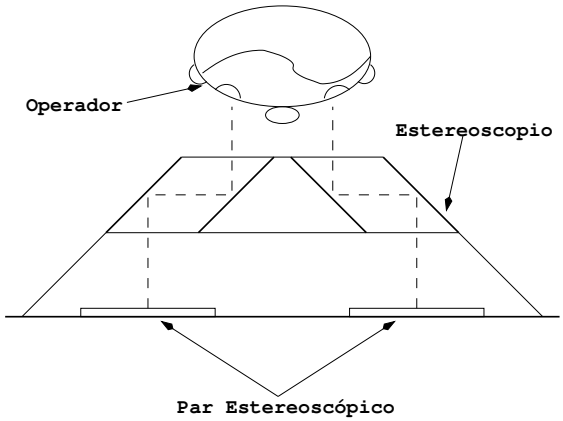

Figura 1.2: Visualización óptica del par estereoscópico por un operador

Trazado de las curvas de nivel

Una vez que el operador humano *ve* la superficie del terreno en tercera dimensión, procede a trazar las curvas de nivel por donde *él cree conveniente*, es decir, las traza *a ojo* sobre algún papel especial, o, posiblemente, en una tableta digitalizadora o incluso sobre una de las fotografías del par, este trazado puede observarse en la Fig. 1.3

Dibujo de las curvas obtenidas en CAD

El proceso final de la obtención de las curvas de nivel es el traslado de las mismas al sistema de CAD que se esté utilizando, si el trazado de las mismas fue hecho sobre una tableta digitalizadora este último paso no es necesario, pero si fue hecho sobre papel u otro medio semejante, sí lo es. En la Fig. 1.4 puede observarse una representación en un sistema CAD de las curvas que

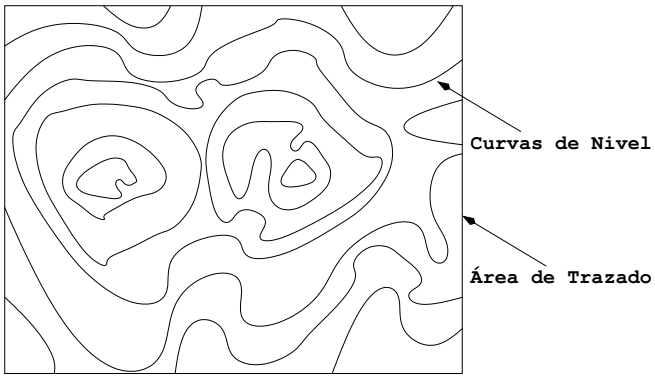

Figura 1.3: Trazo de las curvas de nivel

se muestran en la Fig. 1.3

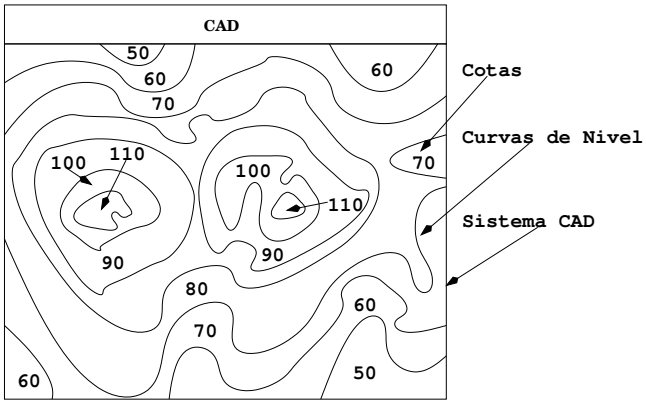

Figura 1.4: Representación en CAD de las curvas de nivel

1.1. Objetivo

El presente trabajo pretende sustituir muchas de las actividades *manuales* del proceso actual, debido a que es muy dependiente del operador humano que traza las curvas de nivel y por ende se encuentra propenso a errores, puede resultar extremadamente lento y, por lo mismo, caro.

El sistema a desarrollar pretende sustituir las operaciones desde el trazado de las curvas de nivel hasta su representación en CAD, utilizando el procesamiento digital del par estereoscópico, el cálculo de las curvas de nivel mediante geometría computacional y la representación tridimensional mediante una interfaz en *Qt*[17] utilizando *OpenGL*[14].

El sistema deberá ser relativamente amigable al usuario, puesto que será utilizado por ingenieros civiles que lo que pretenden es obtener una representación de las curvas de nivel en CAD para su posterior manipulación.

Se cuenta con un conjunto de pares de imágenes de prueba proporcionadas por la *Escuela Superior de Ingeniería y Arquitectura*.

1.2. Descripción de la tesis

El trabajo presentado aquí se divide en los pasos mostrados en la Fig. 1.5. La organización de la tesis se realizó en correspondencia a los pasos de la Fig. 1.5 de la siguiente manera:

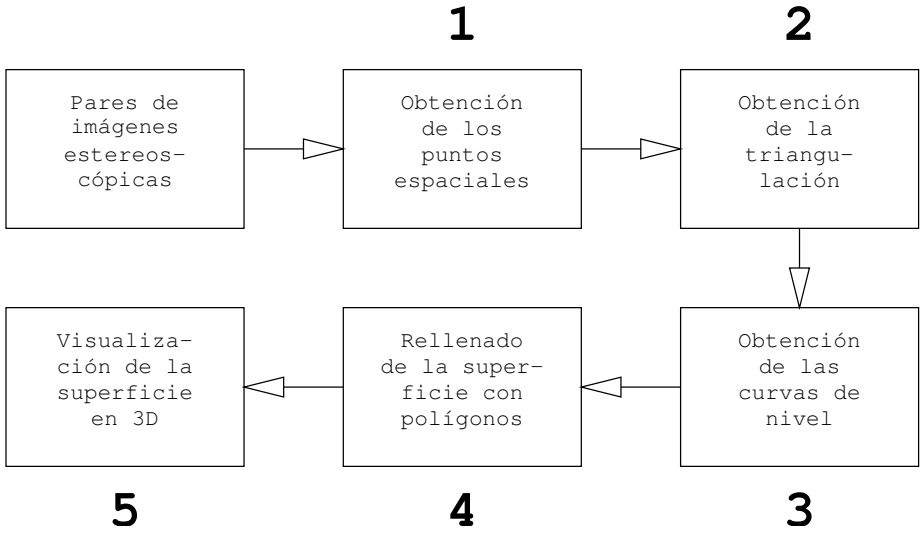

Figura 1.5: Pasos propuestos para la solución del problema

- **Capítulo 2. (Paso 1 en la Fig. 1.5)**: *Obtención de los puntos espaciales mediante la correlación cruzada normalizada rápida de la imagen*

derecha sobre la izquierda

Se realiza una subdivisión del par estereoscópico en bloques cuadrados, a partir de los cuales se realiza la correlación cruzada normalizada rápida considerando el bloque de la imagen derecha y una región mayor en la imagen izquierda centrada en el bloque correspondiente para determinar el lugar de la mejor correspondencia del bloque derecho y de esta forma obtener todos los puntos que se corresponden entre ambas imágenes tal y como se puede apreciar en la Fig. 1.6

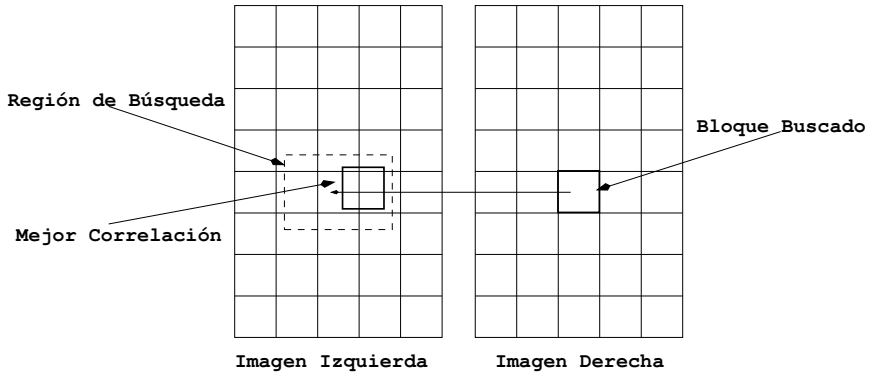

Figura 1.6: Selección de bloques entre ambas imágenes, su relación y la obtención de la mejor correlación entre ellos

- **Capítulo 3, (Paso 2 en la Fig. 1.5)**: *Obtención de la triangulación de Delaunay sobre los puntos tridimensionales reales obtenidos considerando la altura y escala de las fotografías*

  Una vez que se han obtenido los puntos correspondientes se procede al cálculo de las coordenadas tridimensionales reales de cada uno de ellos mediante los datos de campo (altura de la toma de las fotografías, la escala de las imágenes, las posiciones del avión sobre éstas y la distancia del avión entre ambas tomas) y se procede a realizar una triangulación de Delaunay especial sobre dichos puntos (esto es, no se considera la altura de éstos), la cual, además de obtener los triángulos respectivos, también obtiene los dos triángulos que una arista comparte; este esquema puede apreciarse en la Fig. 1.7

- **Capítulo 4. (Pasos 3 y 4 en la Fig. 1.5)**: *Obtención de las curvas de nivel utilizando la triangulación ya generada y retriangulación de la*

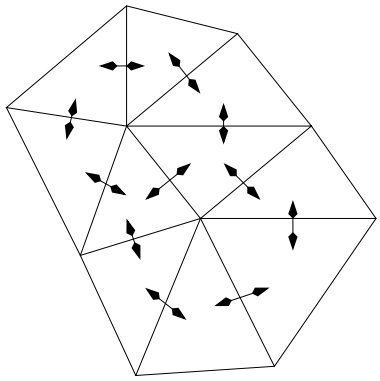

Figura 1.7: Triángulos opuestos a cada arista como son obtenidos por el algoritmo de triangulación propuesto

superficie para delimitar las regiones entre curva y curva

Con la triangulación ya generada se procede a la obtención de las curvas de nivel; éstas se obtienen comenzando desde cualesquier triángulo procediendo sucesivamente a cada triángulo opuesto a la arista que dicha curva atraviesa hasta que se regrese al triángulo desde donde se comenzó (en cuyo caso se tendrá una curva cerrada) o, bien, cuando se salga de la triangulación (en cuyo caso se tendrá una curva abierta); este proceso puede apreciarse en las Figs. 1.8 y 1.9

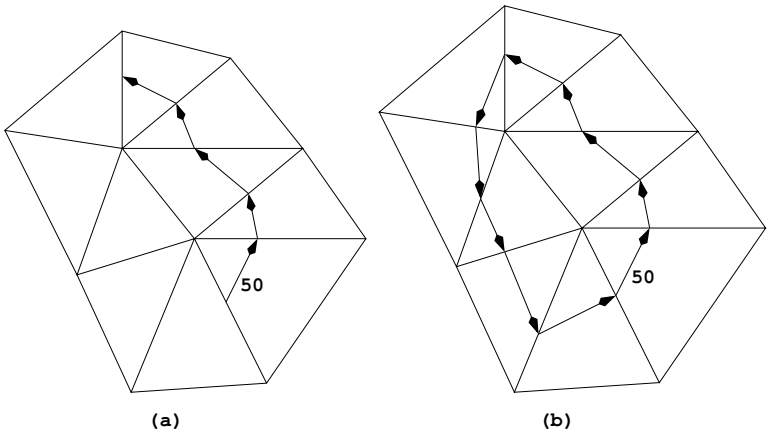

Figura 1.8: Obtención de una curva de nivel cerrada; (a) inicio del algoritmo, (b) final de la curva

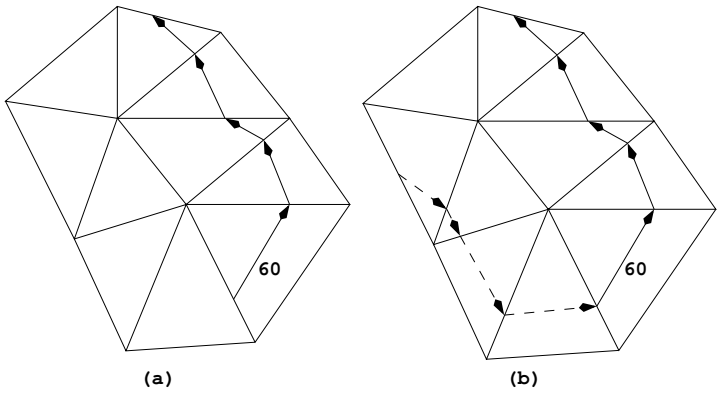

Figura 1.9: Obtención de una curva de nivel abierta; (a) inicio del algoritmo, (b) final de la curva

Mediante las curvas de nivel ya obtenidas es posible, para cada uno de los triángulos que se vayan visitando, ir retriangulando los mismos de forma tal que la nueva triangulación conserve los límites entre curva y curva (impuestos por la diferencia de altura entre las mismas), lo cual permite la obtención de la representación tridimensional de la superficie tal y como se puede ver en la Fig. 1.10

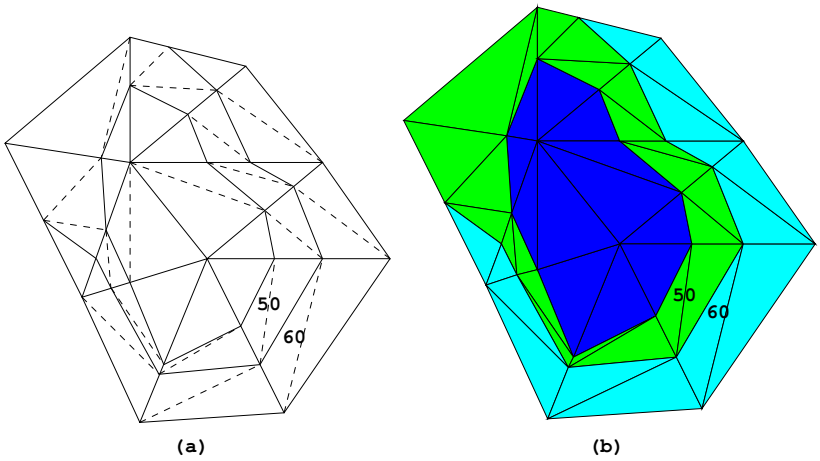

Figura 1.10: (a) Re-triangulación de las curvas de nivel, (b) límites entre curvas de nivel

- **Capítulo 5.**: Aquí se presentan dos ejemplos de las primeras cuatro

etapas del procesamiento; etapas que corresponden a los primeros cuatro capítulos de esta tesis y que están descritos en los puntos anteriores.

- **Apéndice A. (Paso 5 en la Fig. 1.5)**: *Visualización de la superficie triangulada en una interfaz* **Qt** *mediante* **OpenGL**

  Una vez que se tiene la representación tridimensional de la superficie se procede a su visualización mediante una interfaz en *Qt* utilizando *OpenGL*.

- **Capítulo 6**: *Conclusiones del trabajo realizado*

  Se presentan aquí las conclusiones que se consideran más importantes así como las consideraciones del trabajo realizado.

  Se describe, además, el posible trabajo futuro que se puede realizar tomando como base lo que se ha hecho dentro de este trabajo

La biblioteca de funciones de que se compone este trabajo se encuentra desarrollada en el lenguaje de programación C[9]

Capítulo 2

Concordancia

Dentro de este trabajo se utiliza la Correlación Cruzada Normalizada (CCN) para la determinación de la concordancia entre las características de la imagen derecha en la imagen izquierda; tales características no son más que la subdivisión de la imagen derecha en bloques cuadrados de tamaño fijo; dependiendo de la forma en la que se desee realizar la obtención de la concordancia, es posible que estos bloques puedan traslaparse o no; esta subdivisión puede apreciarse en la Fig. 2.1

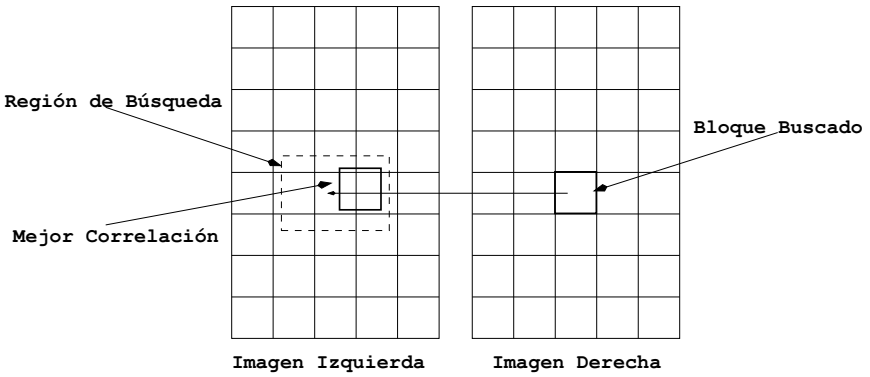

Figura 2.1: Selección de bloques entre ambas imágenes, su relación y la mejor correlación obtenida

La ecuación fundamental para el cálculo de la CCN es la Ec. (2.1), la cual

se ha tomado de [2] y de [3]

$$\text{ccn} = \frac{\sum_{x,y}[f(x,y) - \overline{f}][t(x,y) - \overline{t}]}{\sqrt{\sum_{x,y}[f(x,y) - \overline{f}]^2 \sum_{x,y}[t(x,y) - \overline{t}]^2}} \tag{2.1}$$

en donde ccn es la CCN de los bloques considerados, $f(x,y)$ es la función que representa la luminosidad de cada píxel del bloque de la imagen izquierda, $\overline{f}$ es la media de f, $t(x,y)$ es la función que representa la luminosidad de cada píxel del bloque de la imagen derecha que se desea correlacionar y $\overline{t}$ es la media de t.

El numerador de la Ec. (2.1) se conoce como convolución y es igual a la Ec. (2.2)

$$\text{conv} = \sum_{x,y}[f(x,y) - \overline{f}][t(x,y) - \overline{t}] \tag{2.2}$$

en donde conv es la convolución entre f y t.

Si se pre-normalizan los valores de t a media cero (lo cual se puede hacer en un tiempo $O(l^2)$, con l el tamaño del bloque de búsqueda) es decir

$$t' = t - \overline{t}$$

la Ec. (2.2) se reduce a

$$\begin{aligned}\text{conv} &= \sum_{x,y}[f(x,y) - \overline{f}][t(x,y) - \overline{t}] = \\ &= \sum_{x,y}[f(x,y) - \overline{f}]t'(x,y) = \\ &= \sum_{x,y} f(x,y)t'(x,y) - \overline{f}\sum_{x,y} t'(x,y)\end{aligned}$$

pero como t' tiene media cero, también tiene suma cero, así que $\sum_{x,y} t'(x,y) = 0$, por lo que la ecuación anterior se reduce a la Ec. (2.3)

$$\text{conv} = \sum_{x,y} f(x,y)t'(x,y) \tag{2.3}$$

El denominador de la Ec. (2.1) representa la energía contenida en ambas imágenes, tanto de f como de t y es igual a la Ec. (2.4)

$$\text{ener} = \sqrt{\sum_{x,y}[f(x,y) - \overline{f}]^2 \sum_{x,y}[t(x,y) - \overline{t}]^2} \tag{2.4}$$

si se sustituye t' por t en la Ec. (2.4) se reduce a la Ec. (2.5)

$$\text{ener} = \sqrt{\sum_{x,y}[f(x,y) - \overline{f}]^2 \sum_{x,y} t'(x,y)^2} \tag{2.5}$$

Si durante la pre-normalización a media cero se realiza el ajuste de la energía de t' a la unidad ($\sum_{x,y} t'(x,y)^2 = 1$; la cual se hace en un tiempo $O(l^2)$) se tiene que la Ec. (2.5) se reduce a la Ec. (2.6)

$$\text{ener} = \sqrt{\sum_{x,y}[f(x,y) - \overline{f}]^2} \tag{2.6}$$

desarrollando la Ec. (2.6), se obtiene la Ec. (2.7)

$$\text{ener} = \sqrt{\sum_{x,y} f(x,y)^2 - \frac{(\sum_{x,y} f(x,y))^2}{n}} \tag{2.7}$$

en donde n representa la cantidad total de píxels de la imagen f.

Sustituyendo las Ecs. (2.3) y (2.7) en la Ec. (2.1) se obtiene la Ec. (2.8) que representa la Correlación Cruzada Normalizada Rápida (CCNR)

$$\text{ccnr} = \frac{\sum_{x,y} f(x,y)t'(x,y)}{\sqrt{\sum_{x,y} f(x,y)^2 - \frac{(\sum_{x,y} f(x,y))^2}{n}}} \tag{2.8}$$

en donde ccnr es la correlación cruzada normalizada rápida entre f y t.

2.1. Convolución

Para el cálculo de la convolución se utiliza la relación entre ésta y la Transformada de Fourier mostrada en la Ec. (2.9)

$$\text{conv} = \mathcal{F}^{-1}(\mathcal{F}(f) \cdot \mathcal{F}^*(t)) \tag{2.9}$$

en donde $\mathcal{F}$ es la transformada de Fourier bidimensional, $\mathcal{F}^{-1}$ es la transformada inversa y $\mathcal{F}^*$ es el conjugado de la transformada de Fourier bidimensional.

La transformada de Fourier permite la evaluación de la convolución entre el bloque de la imagen derecha y el bloque de búsqueda, de tamaño mayor, de la imagen izquierda; esto debido a la forma de operar de la misma. En este trabajo se hace uso de esta característica para la obtención de la convolución de una forma rápida, lo cual permite la obtención de ésta en un tiempo $O(nm \log r)$, en donde $r = \text{máx}(n, m)$ (utilizando la Transformada Rápida de Fourier -FFT-), donde n y m representan el tamaño de la imagen. Es sabido que la complejidad de la FFT para un arreglo de n elementos es $O(n \log n)$, por lo que el orden de la convolución es $O(mn \log n) + O(nm \log m) = O(nm \log r)$. El algoritmo programado para el cálculo de esta transformada se adaptó de [6]

2.2. Cálculo de la energía

Para el cálculo de la energía se hace uso de las Ecs. (2.10) y (2.11)

$$\sum_{x_0}^{x_1} f(x) = \sum_{x_0-1}^{x_1-1} f(x) - f(x_0 - 1) + f(x_1) \tag{2.10}$$

$$\sum_{x_0}^{x_1} f^2(x) = \sum_{x_0-1}^{x_1-1} f^2(x) - f^2(x_0 - 1) + f^2(x_1) \tag{2.11}$$

en las cuales x_0 y x_1 representan el intervalo del tamaño del bloque (si t es el tamaño del bloque, entonces $x_1 = x_0 + t - 1$) y f es la función a la cual se le desea obtener la energía.

Utilizando estas relaciones es posible calcular las sumatorias necesarias, primero en la dirección horizontal sobre la fotografía izquierda (véase el proceso simplificado en la Fig. 2.2) y una vez determinadas, se procede al cálculo en la dirección vertical (esta parte del proceso se muestra de forma simplificada en la Fig. 2.3) y, con estas sumas, la energía del bloque considerado es igual a la Ec. (2.12); las regiones de las sumas que se calculan pueden

apreciarse en la Fig. 2.4

$$\text{ener} = \sqrt{\sum_{x_0}^{x_1} f^2(x) - \frac{(\sum_{x_0}^{x_1} f(x))^2}{n}} \tag{2.12}$$

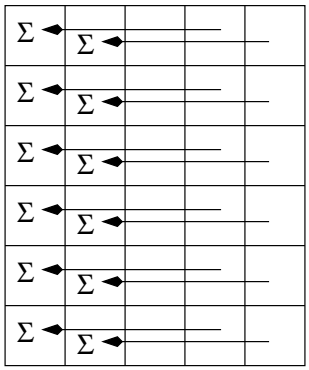

Figura 2.2: Sumas horizontales, cuando $t = 4$; las flechas indican los píxels que se han de sumar sobre las filas de la imagen

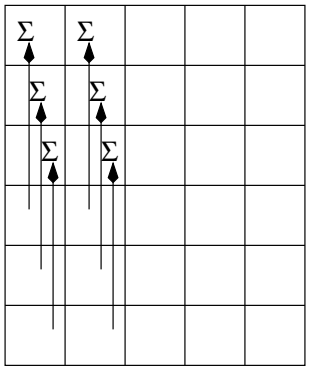

Figura 2.3: Sumas verticales, cuando $t = 4$; las flechas indican los píxels que se han de sumar sobre las columnas de la imagen

Con esto es posible la obtención de la energía de f en un tiempo $O(nm)$.

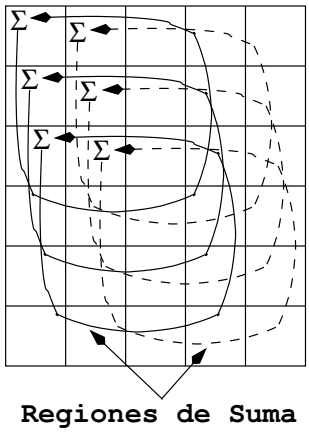

Figura 2.4: Cálculo de la energía, cuando $t = 4$; las regiones cuadradas corresponden a los píxels que se suman para la obtención de la energía del bloque

2.3. Correlación cruzada normalizada rápida

Una vez que se tienen los cálculos de las Ecs. (2.9) y (2.12) junto con la normalización del bloque de búsqueda y utilizándolas en la Ec. (2.8), se tiene un algoritmo que se ejecuta en:

$$O(\mathcal{A}) = O(l^2 nm \log r) + O(nm) = O(l^2 nm \log r)$$

en donde $\mathcal{A}$ es el algoritmo descrito.

Mediante este algoritmo se obtiene el mapa de conconrdancia entre ambas imágenes que se observa en la Fig. 2.5

2.4. Descripción del proceso

Para la obtención del mapa de desplazamientos se sigue el siguiente proceso general (las palabras mostradas al comienzo de cada punto en letra `monoespaciada` representan los nombres de las funciones programadas. Una descripción de estas funciones se haya en el apéndice D)

1. (`Concordancia`) División de la imagen derecha en bloques cuadrados de igual tamaño, los cuales se pueden traslapar o no tal y como se aprecia en la Fig. 2.6

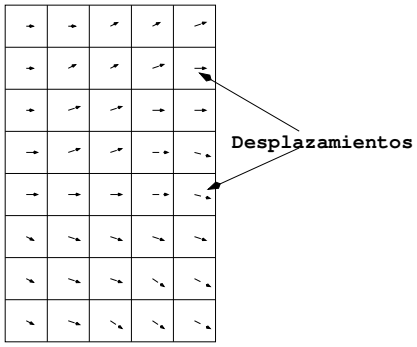

Figura 2.5: Mapa de concordancia entre las imágenes del par estereoscópico una vez que han sido procesadas por el algoritmo $\mathcal{A}$. Las flechas representan el desplazamiento entre el centro del bloque derecho y el centro del bloque mejor correlacionado izquierdo

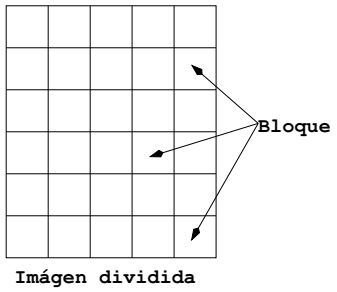

Figura 2.6: Cada cuadrado representa un bloque de la imagen derecha ya dividida

2. (`Concordancia`) División de la imagen izquierda en los mismos bloques de la imagen derecha, para efectos de concordancia entre ellas

3. (`ObtenEnergias`) Obtención de las energías de todos y cada uno de los bloques de que se compone la imagen izquierda (píxel a píxel, sin que necesariamente se correspondan a los bloques de la imagen derecha) como puede apreciarse en la Fig. 2.7

4. (`Correlacion`) Se procede a la obtención de la correlación entre el bloque de la imagen derecha y el bloque de búsqueda de la imagen izquierda

    4.1. (`Correlacion`) Cada bloque de la imagen izquierda se expande dependiendo de los datos proporcionados para efectos de la búsqueda

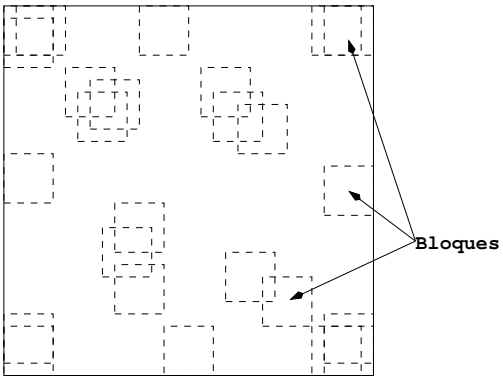

Figura 2.7: Bloques de la imagen izquierda a los que se les calcula la energía; nótese que es para cada bloque del tamaño establecido que se obtiene píxel a píxel

del respectivo bloque de la imagen derecha como se aprecia en la Fig. 2.8

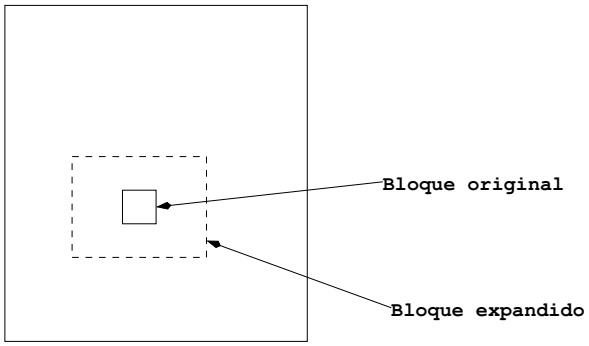

Figura 2.8: Bloque original de la imagen izquierda que se ha expandido para su posterior convolución

4.2. (`NormalizaImagen`) Se normaliza a media cero los valores de píxel del bloque de la imagen derecha y se obtiene el valor de su energía

4.3. (`ObtenPotenciaDeDos`) Ambos bloques (el izquierdo y el derecho) se hacen del mismo tamaño, el cual será una potencia de dos

4.4. (`Convolucion`) Se procede a la obtención de la convolución de ambas imágenes (en la Fig. 2.9 se puede observar el proceso simplificado de la obtención de dicha convolución)

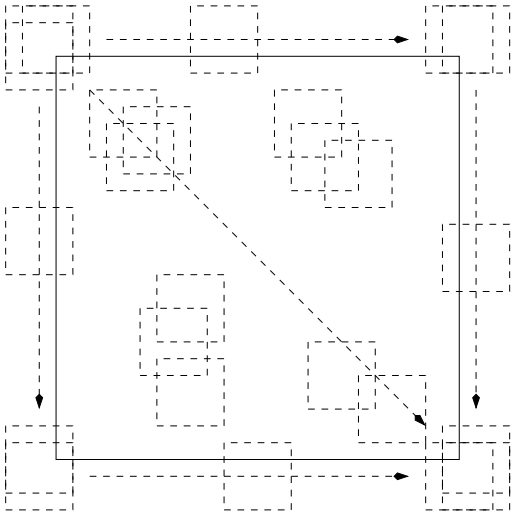

Figura 2.9: El cuadrado con línea punteada representa el bloque de la imagen derecha y el cuadrado con línea continua el bloque expandido de la imagen izquierda. Para cada posible posición del bloque derecho sobre el izquierdo se calcula la convolución

4.4.1. (`FFT2`) Se procede a obtener la FFT bidimensional de ambas imágenes

4.4.2. (`Convolucion`) Se multiplican uno a uno los elementos de ambas imágenes, tomando en cuenta que de la imagen derecha se toma el conjugado de cada uno de los valores complejos obtenidos por la FFT

4.4.3. (`FFT2`) Se obtiene la FFT inversa de la imagen producto

4.5. (`Correlacion`) Una vez con la convolución se procede al cálculo de la correlación entre el bloque de la imagen derecha y todos los posibles bloques de la imagen izquierda (tomando en cuenta que este se agrandó) utilizando las energías obtenidas en los pasos *3* y *4.2*

4.6. (`Correlacion`) Se toma como el mejor aquel bloque que tenga el valor de la correlación más cercano a la unidad

4.7. (`Correlacion`) Se almacenan los resultados obtenidos de la correlación en la entrada del arreglo correspondiente, formando así el mapa de conconrdancia, como el mostrado en la Fig. 2.5

5. (`Concordancia`) Se repite el paso *4* para cada bloque de la imagen derecha

Capítulo 3

Triangulación

Una vez que se han obtenido los puntos semejantes en ambas fotografías estereoscópicas, es necesario realizar una representación de la superficie con esos puntos semejantes. La "mejor" representación se consigue con la triangulación de Delaunay [24].

En este capítulo se van a obtener las coordenadas reales –las coordenadas tridimensionales, con altura, y corregidas debido al paralaje– de los puntos semejantes y cómo se implantó la triangulación de Delaunay.

3.1. Cálculo de los puntos reales

Para poder obtener la triangulación, lo primero que se necesita es un conjunto de puntos en el espacio. Con el mapa de desplazamientos obtenido con el procedimiento explicado en el capítulo 2, Pág. 15, se obtienen las coordenadas en el plano de los puntos semejantes en ambas fotografías. Ahora, para la obtención de las coordenadas tridimensionales de estos puntos coincidentes es necesaria la utilización de la información del paralaje, es decir, la información de los desplazamientos entre ambas imágenes (la distancia relativa de la posición de un objeto entre lo que ve el ojo derecho y lo que ve el ojo izquierdo), la cual se muestra en la Fig. 3.1

Con la información de paralaje, los desplazamientos relativos entre ambas imágenes y el mapa de desplazamientos de los puntos se puede (por triángulos semejantes) obtener la altura de todos ellos, tal y como se muestra en

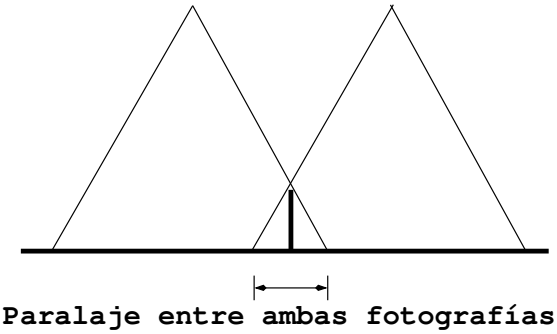

Figura 3.1: Información de paralaje que se obtiene al momento de la toma del par estereoscópico

las Figs. 3.2 y 3.3; para estas figuras se tienen: O posición del avión en el momento de la toma de la fotografía, A vertical (azimuth) del avión sobre el terreno, A' vertical del avión sobre la fotografía, D es la posición *real* de la característica sobre el terreno, D' es la posición *real* de la característica sobre la fotografía, B es la *proyección* de la característica sobre el terreno, B' es la *proyección* de la característica sobre la fotografía, C es la posición *observada* de la característica del terreno, C' es la posición *observada* de la característica sobre la fotografía, f es la altura a la que fueron tomadas ambas fotografías, f' es la distancia focal de la lente de la cámara, h es la altura *real* de la característica, O_0 es el punto en el cual el avión toma la primer fotografía, O_1 es el punto en el que se toma la segunda, $d = \overline{A_0A_1}$ es la distancia sobre el terreno de las verticales del avión y $d' = \overline{A'_0A'_1}$ es la distancia sobre las fotografías de las verticales del avión.

De la relación entre los triángulos semejantes BCD y ACO en la Fig. 3.2 se tiene

$$\frac{h}{\overline{BC}} = \frac{f}{\overline{AC}}, \qquad h = f\frac{\overline{BC}}{\overline{AC}}$$

para ambos triángulos (en las dos fotografías) de la Fig. 3.3 y la ecuación anterior se tiene

$$h = f\frac{\overline{BC_0}}{\overline{A_0C_0}} = f\frac{\overline{BC_1}}{\overline{A_1C_1}} \tag{3.1}$$

pero $\overline{BC} = \overline{AC} - \overline{AB}$ en la Fig. 3.2, de lo cual

$$\frac{\overline{A_0B}}{\overline{A_0C_0}} = \frac{\overline{A_1B}}{\overline{A_1C_1}}$$

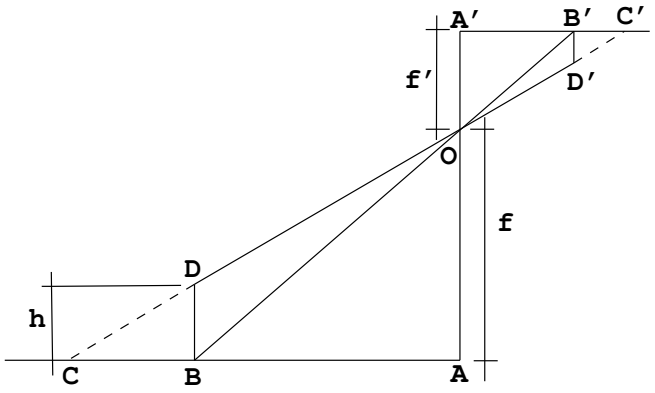

Figura 3.2: Cálculo de la altura, una sola fotografía; en donde se muestran los triángulos que representan la característica sobre el terreno y los correspondientes dentro de la cámara

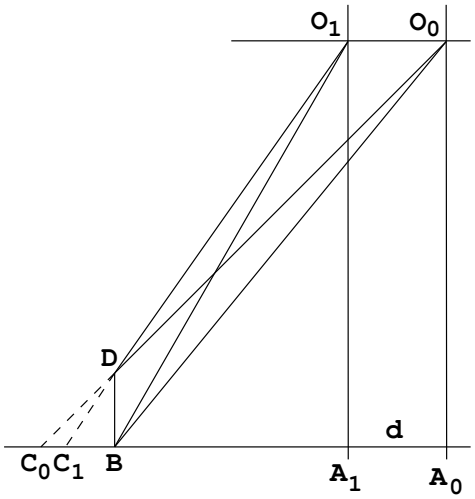

Figura 3.3: Cálculo de la altura para las dos fotografías; en donde se muestran solamente los triángulos correspondientes en el momento de la toma de éstas

si se hace $b_0 = \overline{A_0B}$, $c_0 = \overline{A_0C_0}$, $b_1 = \overline{A_1B}$ y $c_1 = \overline{A_1C_1}$ se tiene

$$\frac{b_0}{c_0} = \frac{b_1}{c_1} \tag{3.2}$$

de la cual

$$\frac{c_0 - b_0}{c_0} = \frac{c_1 - b_1}{c_1} \tag{3.3}$$

sustituyendo la Ec. (3.3) en la Ec. (3.1) se obtienen

$$h = f\left(\frac{c_0 - b_0}{c_0}\right) = f\left(1 - \frac{b_0}{c_0}\right) \tag{3.4}$$

$$h = f\left(\frac{c_1 - b_1}{c_1}\right) = f\left(1 - \frac{b_1}{c_1}\right) \tag{3.5}$$

de la Ec. 3.2 se obtiene

$$b_1 = \frac{c_1 b_0}{c_0}, \qquad b_0 - b_1 = \frac{b_0(c_0 - c_1)}{c_0}$$

pero $d = b_0 - b_1$ así que

$$\frac{b_0}{c_0} = \frac{d}{c_0 - c_1}$$

sustituyendo esta última ecuación en la Ec. (3.4) se obtiene la altura real de la característica del terreno en la Ec. (3.6)

$$h = f\left(1 - \frac{d}{c_0 - c_1}\right) = f\left(1 - \frac{d'}{c'_0 - c'_1}\right) \tag{3.6}$$

de las Ecs. (3.4) y (3.5) se obtienen

$$\frac{h}{c_0 - b_0} = \frac{f}{c_0}, \qquad b_0 = c_0\left(1 - \frac{h}{f}\right)$$

$$\frac{h}{c_1 - b_1} = \frac{f}{c_1}, \qquad b_1 = c_1\left(1 - \frac{h}{f}\right)$$

sustituyendo la Ec. (3.6) en estas últimas ecuaciones se obtienen las Ecs. (3.7) y (3.8) que representan las coordenadas reales en la dirección del movimiento del avión.

$$b_0 = \frac{c_0 d'}{c'_0 - c'_1} \tag{3.7}$$

$$b_1 = \frac{c_1 d'}{c'_0 - c'_1} \tag{3.8}$$

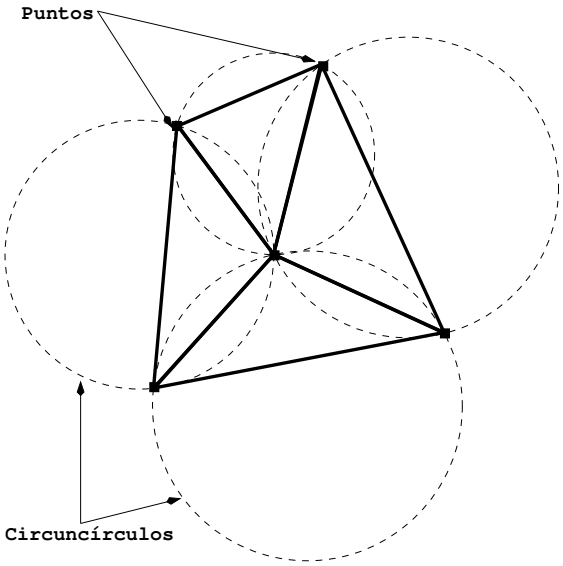

Figura 3.4: La triangulación de Delaunay de los cinco puntos mostrados está representada mediante la línea continua

3.2. Triangulación de Delaunay

La definición de la triangulación de Delaunay es: *Para cualesquier triángulo de la triangulación, el círculo que lo inscribe no contiene ningún otro punto, del conjunto dado, en su interior.* En la Fig. 3.4 se puede observar la triangulación de Delaunay de un conjunto de cinco puntos.

Para ésta se ha implementado el algoritmo descrito en [4], el cual contiene los siguientes pasos:

1. Cálculo de la coordenada absoluta mayor
2. Obtención del triángulo inicial
3. Para cada punto del mapa:
   - 3.1. Identificación del triángulo que contiene el punto a añadir
   - 3.2. Creación de los nuevos triángulos que dividen al original
   - 3.3. Verificación y re-establecimiento de la condición de la triangulación de Delaunay

4. Numeración de los triángulos finales

5. Obtención del arreglo de triángulos

Dentro del algoritmo descrito se menciona el hecho que es muy importante el manejo de los casos particulares, es decir, de triángulos degenerados que pudieran encontrarse dentro del proceso.

3.3. Manejo del triángulo original

El procesamiento de los triángulos que contienen alguno de los puntos que forman el triángulo original (dicho triángulo se crea de forma tal que todos los puntos del conjunto que se triangularán estarán contenidos en él) se hace de la siguiente forma (en la Fig. 3.5 se muestra el diagrama que ejemplifica el proceso):

- Si el vértice 2 o el vértice 3 pertenecen al triángulo original y el vértice 1 o el vértice 4 pretenecen al triángulo original y se viola la condición de la triangulación, hay que intercambiar las diagonales
- Si ni el vértice 1 ni el vértice 4 pertenecen al triángulo original y se viola la condición de la triangulación, hay que intercambiar las diagonales

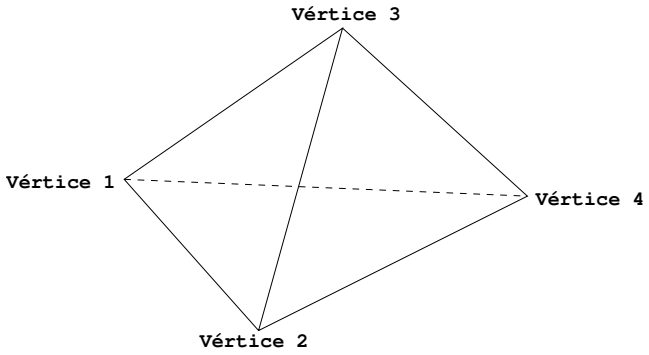

Figura 3.5: Manejo de los vértices del triángulo original, la diagonal original es la dibujada con línea continua y la que la substituirá es la punteada

Una vez que se han incluído todos los puntos del conjunto y, por ende, obtenido la triangulación deseada, es necesario eliminar todos los triángulos en los que por lo menos uno de sus vértices sea vértice del triángulo original.

3.4. Cuándo un triángulo no está bien formado

Dentro del proceso que se sigue para ir añadiendo los puntos a la triangulación ya existente, es posible que se vayan obteniendo triángulos degenerados, entre los cuales podemos encontrar de tres tipos; el primero cuando se tienen tres puntos colineales no coincidentes entre sí, el segundo caso cuando se tienen dos puntos con las mismas coordenadas y, el tercer caso, cuando los tres puntos coinciden entre sí, se puede ver una representación esquemática de ellos en la Fig. 3.6

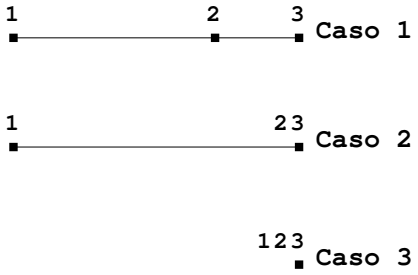

Figura 3.6: Los tres tipos de triángulos degenerados que se pueden obtener dentro del proceso de la triangulación

3.4.1. Puntos repetidos

De los tres casos de triángulos degenerados de la Fig. 3.6, el único que interesa es cuando existe más de un punto con las mismas coordenadas; en éste si se deja que el algoritmo inserte un punto en donde ya hay uno será creada una arista con longitud cero y, por ende, un vector de longitud cero, por lo que el vector perpendicular al plano que forma este triángulo no existe, con lo cual la parte del algoritmo que trata de determinar si se ha de modificar la diagonal del cuadrilátero formado falla y el algoritmo no puede determinar el lugar en donde deberá de caer dicho punto.

El algoritmo evita la inserción de más de un punto con las mismas coordenadas ordenando primero la lista de puntos de forma lexicográfica (primero el valor de la coordenada y y, posteriormente, la coordenada x) revisando si

el punto que se está tratando de insertar coincide con el previamente insertado y, si son iguales, no se realiza su inserción, pero, si son diferentes, sí se inserta.

3.4.2. Triángulos degenerados

En el caso de un triángulo degenerado cuyos vértices son colineales pero cuyas coordenadas no son las mismas, el algoritmo lo considera como cualquier otro triángulo y procede a la inserción del punto que se esté procesando en ese momento. Aquí lo que se obtiene es un triángulo que en realidad es una línea recta, su vector perpendicular es un vector cero, por lo que en este caso el punto se considera dentro del triángulo solamente si sus coordenadas se encuentran dentro de las coordenadas de los extremos de la recta, tal y como puede verse en la Fig. 3.7

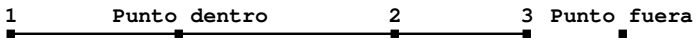

Figura 3.7: Triángulo degenerado, con inserción de punto

3.5. Descripción del proceso

Para la obtención de la triangulación se sigue el proceso general descrito a continuación (las palabras mostradas al comienzo de cada punto en letra `monoespaciada` representan los nombres de las funciones programadas. Una descripción de estas funciones se haya en el apéndice D):

1. (`OrdenaLexicograficamente`) Se ordenan los puntos lexicográficamente (primero la coordenada y y luego la coordenada x)

2. (`ValorMaximo`) Obtención de la coordenada absoluta máxima, no importando si es en el eje x o en el eje y.

3. (`PrimerTriangulo`) Creación de un triángulo inicial con vértices cuyas coordenadas son $(\text{máx}, 0)$, $(0, \text{máx})$ y $(-\text{máx}, -\text{máx})$, con lo cual se garantiza que todos los puntos del mapa estarán contenidos dentro de este triángulo inicial (véase la representación esquemática de dicho triángulo en la Fig. 3.8) Este triángulo inicial crea un nodo dentro de

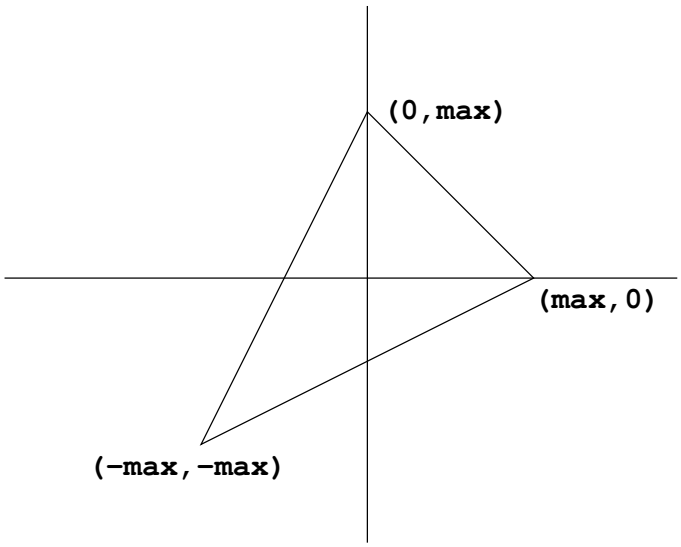

Figura 3.8: Triángulo inicial creado a partir de la coordenada máxima de los puntos max, el cual comienza el proceso de triangulación

una estructura especial que se va almacenando en memoria, en la cual se tienen:

3.1. (`tArbolTrianguloD`) Una lista simplemente ligada que contiene la totalidad de los triángulos que se han creado; esta estructura es necesaria para cuando se están extrayendo los triángulos terminales finales de la triangulación y no se desea que se revise más de una vez cada triángulo; también es importante para establecer la numeración final de los mismos

3.2. (`tArbolTrianguloD`) Un árbol en el cual se establece la relación de contención entre todos los triángulos creados, de padres a hijos, definida en el algoritmo [4]; esto es para facilitar el proceso de la identificación de la contención, de los puntos que se están insertando, en los triángulos de una forma eficiente

4. Para cada punto del mapa se procede como sigue:

4.1. (`UbicaPunto` y `DentroDeTriangulo`) Se identifica el triángulo que lo contiene, utilizando la estructura de árbol, de la forma descrita a continuación y mostrada en las Figs. 3.9, 3.10 y 3.11:

4.1.1. Se comienza desde el nodo raíz del árbol tal y como se muestra en la Fig. 3.11(a)

4.1.2. Se identifica sobre cuál triángulo hijo se encuentra el punto y se desplaza hacia allá (mostrado en la Fig. 3.11(b))

4.1.3. Si el triángulo se encuentra en un nodo hoja del árbol, ya se llegó al triángulo terminal que lo contiene, en caso contrario se repite el proceso desde *3.1.2.* (el triángulo hoja final se muestra en la Fig. 3.11(c))

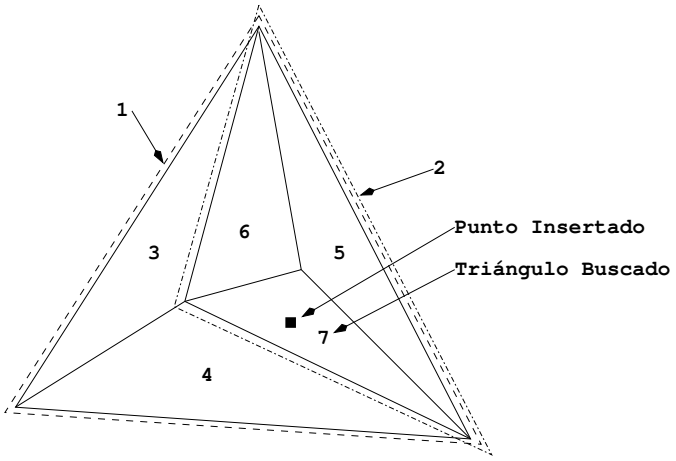

Figura 3.9: Punto al que se le desea determinará cuál es el triángulo que lo contiene

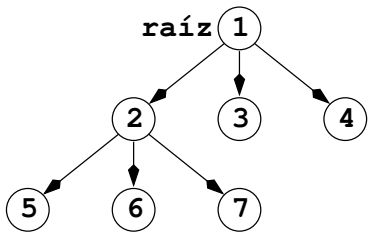

Figura 3.10: Árbol que representa la triangulación de la Fig. 3.9

4.2. (`CreaNodoPuntos`) Se divide el triángulo, en el que el punto a insertar se encuentra contenido, en sus triángulos componentes (se obtienen 3) y se marca el original como un triángulo no terminal; este proceso puede observarse en la Fig. 3.12

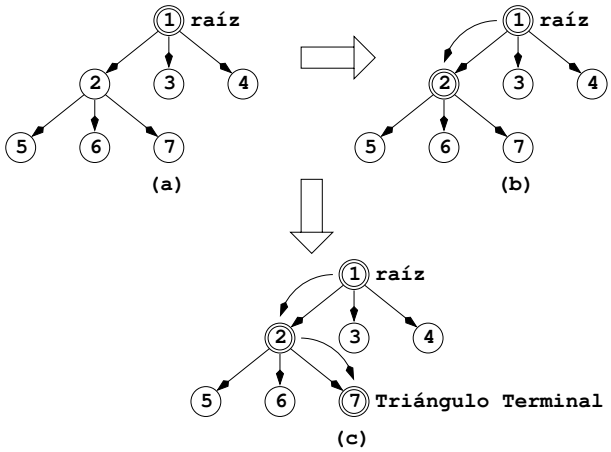

Figura 3.11: Recorrido dentro del árbol hasta el triángulo que contiene el punto que se desea insertar (ver Fig. 3.9) el cual va del nodo *1* (raíz) al *2* y, finalmente, al *7*

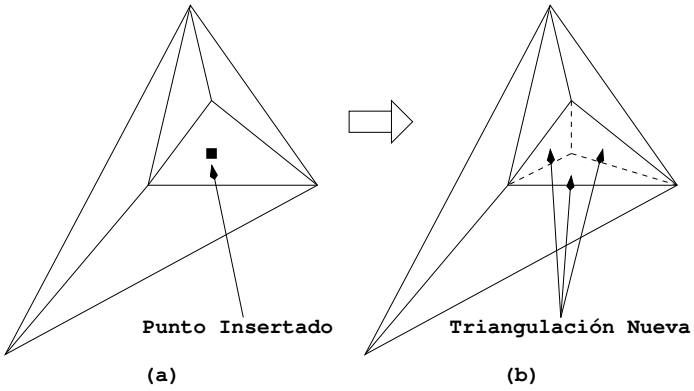

Figura 3.12: División de un triángulo en sub-triángulos por el punto que se está insertando

4.3. (`InsertaTrianguloListaD`) Se enfilan los nuevos triángulos generados para verificar si se viola la condición de la triangulación de Delaunay dentro de los cuadriláteros nuevos formados; dichos cuadriláteros se muestran en la Fig. 3.13

4.4. (`VerificaOpuestos`) Para cada triángulo en la cola de procesamiento de violaciones, se realizan los siguientes pasos:

4.5.1. (`ExtraeTrianguloListaD`) Se extrae el siguiente triángulo de

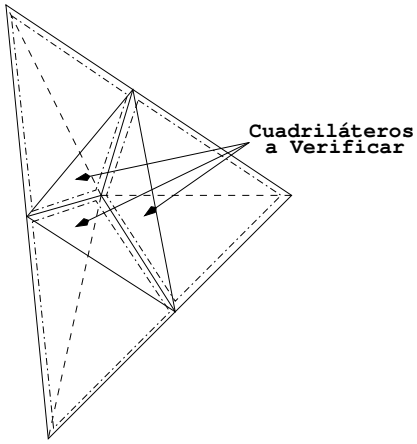

Figura 3.13: Verificación de cada nuevo triángulo y su cuadrilátero asociado, las diagonales con línea continua son las que posiblemente sean substituídas por las de línea punteada

la cola

4.5.2. (`VerificaOpuestos`) Se pregunta si es un triángulo terminal, caso de que no, se va al paso *3.5.1.*

4.5.3. (`AngulosNoValidos` e `IntercambiaAristas`) Se revisa la condición del círculo circunscrito y, si se viola, se procede al cambio de las diagonales del cuadrilátero que se forma con las aristas de estos dos triángulos, tal y como se muestra enla Fig. 3.14 y se enfilan los nuevos triángulos formados para su verificación

4.5.3.1. (`AngulosNoValidos`) Si los puntos son vértices del triángulo original, se hacen las siguientes consideraciones (véase la numeración de los vértices en la Fig. 3.5); si el vértice 2 o el vértice 3 pertenecen al triángulo original y el vértice 1 o el vértice 4 pretenecen al triángulo original y se viola la condición de la triangulación, hay que intercambiar las diagonales, pero, si ni el vértice 1 ni el vértice 4 pertenecen al triángulo original y se viola la condición de la triangulación, hay que intercambiar las diagonales tal y como se muestra en la Fig. 3.15

4.5.3.2. (`InsertaTrianguloListaD`) Se enfilan los nuevos triángulos para la revisión de la condición de la triangulación de

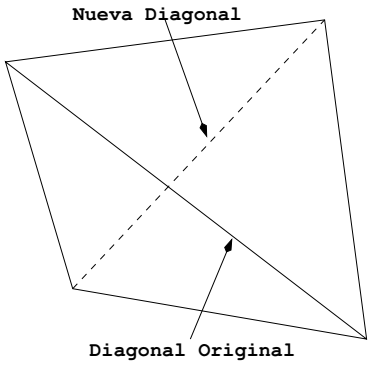

Figura 3.14: Intercambio de diagonales en un par de triángulos que violan la condición del círculo cinrcunscrito

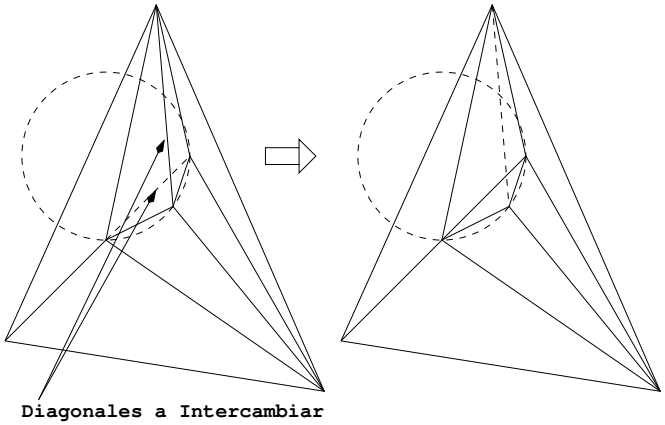

Figura 3.15: Consideraciones del punto original. En este caso, aunque no se está violando la condición de la triangulación de Delaunay, es necesario el cambio de diagonales para conservar la cubierta convexa de los puntos del mapa

Delaunay

5. (`ObtenTriangulosFinalesD`) Una vez con todos los puntos añadidos a la estructura, se obtienen los triángulos que sí forman parte de la triangulación definitiva:

   5.1. (`ObtenTrianguloTerminal` y `CuantosTriangulosTerminales`) Se numeran los triángulos terminales comenzando desde el 0 y se establecen las relaciones de los triángulos opuestos por aristas como se muestra en la Fig. 3.16

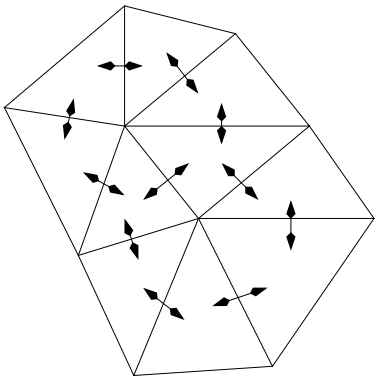

Figura 3.16: Triángulos opuestos a todas y cada una de las aristas que conforman la triangulación de Delaunay obtenida

6. (`ObtenTriangulosFinalesD`) Se copian los triángulos finales al arreglo que será devuelto

Durante todo el proceso se van actualizando las estructuras de datos (lista y árbol) para que éste se vaya realizando de una forma óptima; en las Figs. 3.17, 3.18, 3.19, 3.20 y 3.21 se muestra una serie de inserciones de puntos dentro de una triangulación con el objeto de ilustrar el comportamiento, tanto de la lista como del árbol, dentro del desarrollo del algoritmo.

En la Fig. 3.17 se muestra solamente el triángulo inicial y el único nodo existente dentro de las estructuras correspondientes (árbol y lista)

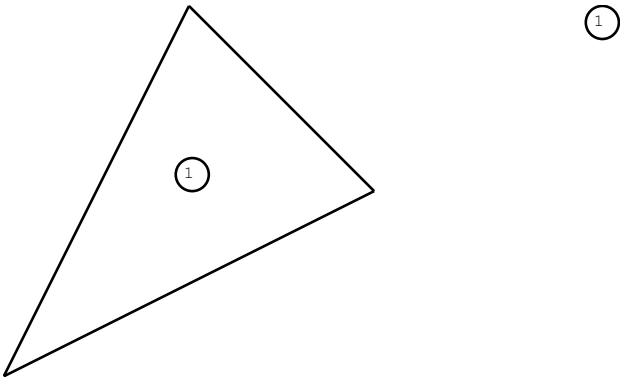

Figura 3.17: Inicio del algoritmo

En la Fig. 3.18 se muestra la inserción del primer punto y la división del triángulo inicial en sus tres sub-triángulos. Dentro de la estructura se puede apreciar en línea punteada las ramas del nodo original dentro del árbol y en continua la lista que se va formando con todos los triángulos creados hasta el momento.

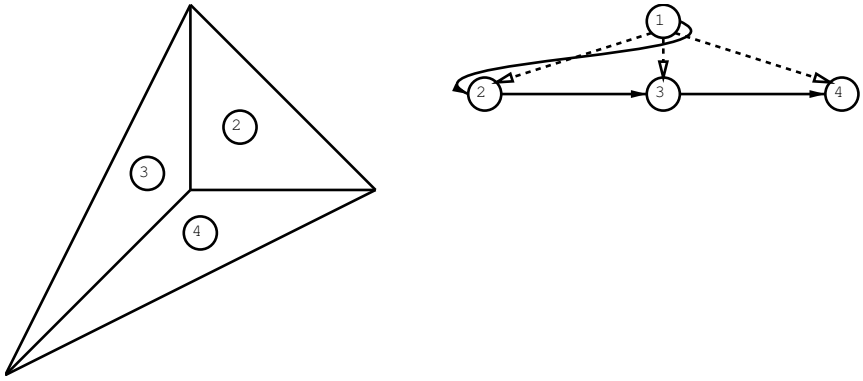

Figura 3.18: Inserción del primer punto; en línea punteada se aprecia la estructura de árbol y en continua la lista

En la Fig. 3.19 se muestra la inserción del segundo punto y la división del triángulo número 2, hasta este momento no se ha creado ningún triángulo propiamente dicho dentro de los puntos insertados.

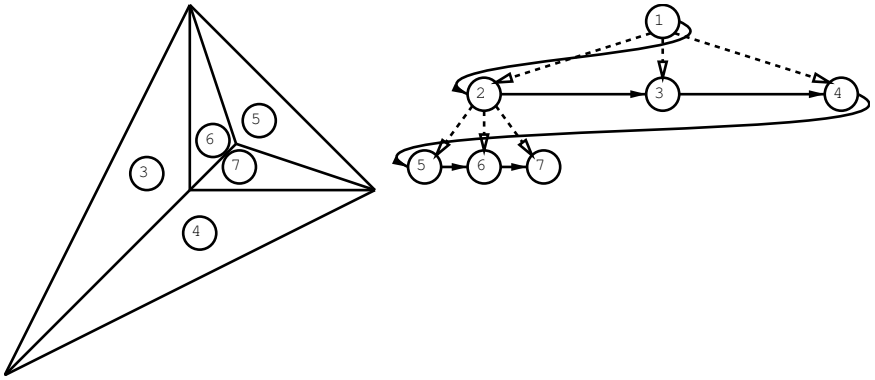

Figura 3.19: Inserción del segundo punto

En la Fig. 3.20 se ha insertado un punto dentro del triángulo número 3, el cual crea tres nuevos sub-triángulos. En la estructura correspondiente se

puede apreciar la formación de más hojas dentro de la estructura de árbol y las ligas correspondientes a la lista. Una vez aquí se detecta que es necesario un intercambio de diagonales para la obtención del primer triángulo formado por los puntos insertados (triángulos 4 y 8)

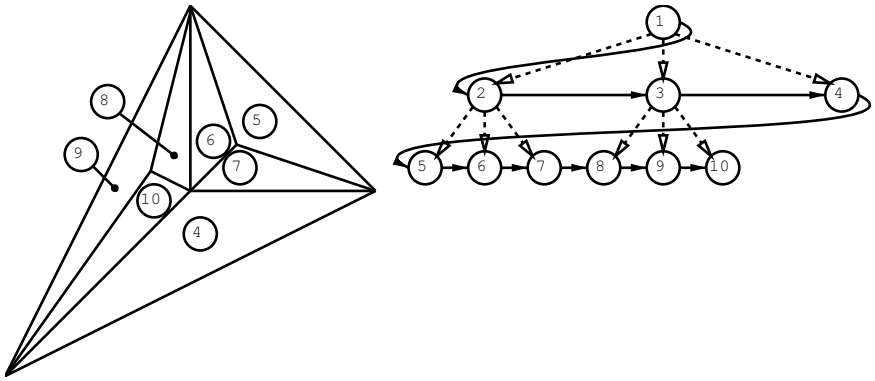

Figura 3.20: Inserción del tercer punto

En la Fig. 3.21 se muestra el intercambio de las diagonales entre los triángulos 4 y 8, formándose 2 nuevos triángulos, el 11 y 12. Del lado de las estructuras se puede observar que existen un par de triángulos que tienen los mismos hijos (el 4 y 8) para cuestiones de búsqueda, para la lista simplemente se han añadido los nuevos triángulos a ésta.

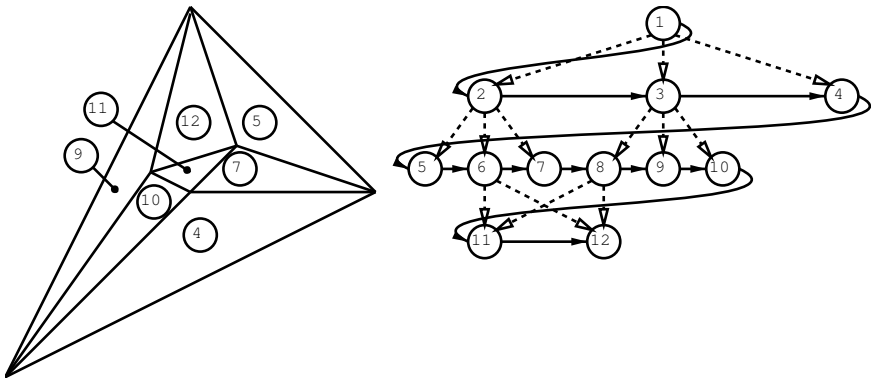

Figura 3.21: Intercambio de diagonales

3.6. Prueba de la triangulación

Para la verificación del algoritmo de triangulación programado se hizo una comparativa con el *qhull* [29]; se tomó la decisión de realizar el algoritmo de triangulación ya que con el *qhull* solamente se obtienen los vértices de los triángulos que conforman la triangulación y, dentro del procesamiento posterior para este trabajo, es necesario que para todos y cada uno de los triángulos se obtengan los correspondientes opuestos a sus aristas. Esto es, para nuestro procesamiento necesitamos una estructura ordenada de los vértices.

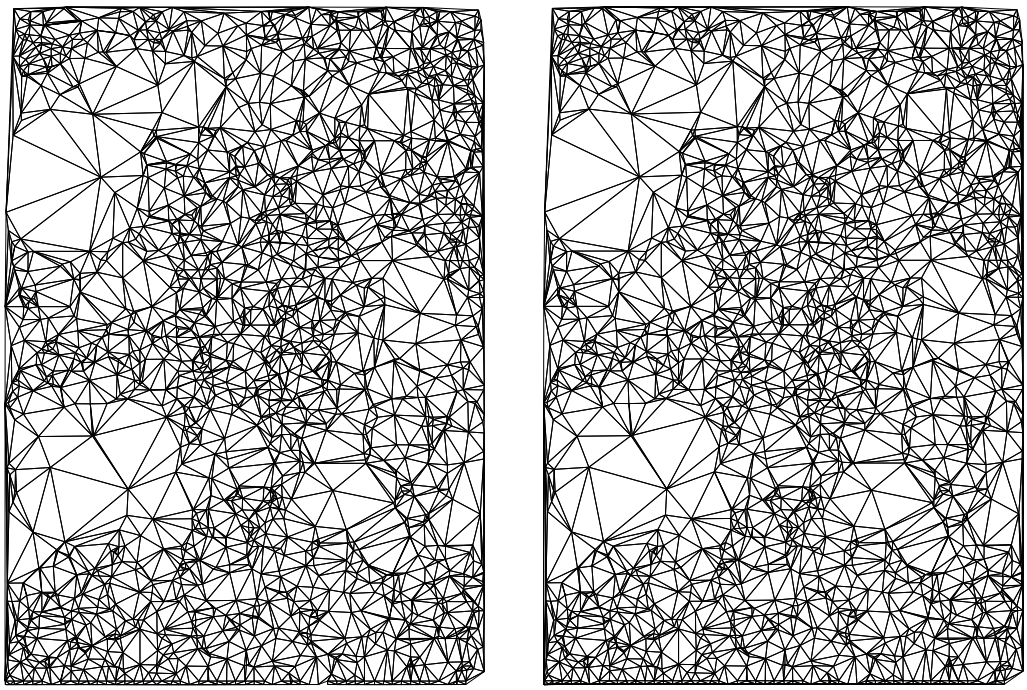

Figura 3.22: En la imagen izquierda se muestra la triangulación con el algoritmo desarrollado, en la derecha el resultado del *qhull*, en baja densidad

Se tomó uno de los pares estereoscópicos, se obtuvo su mapa de concordancias y, precisamente, este mapa se trianguló utilizando el algoritmo desarrollado y el *qhull* en dos densidades diferentes; en el caso de la densidad baja se puede observar en la Fig. 3.22 en donde se observa que prácticamente son iguales ambas triangulaciones.

En las Figs. 3.23 y 3.24 se pueden observar las triangulaciones obtenidas con una densidad alta de puntos; es posible darse cuenta de pequeñas diferencias entre ambas triangulaciones, sin embargo, son mínimas comparadas con la gran cantidad de triángulos resultantes. Una diferencia apreciable es un par de triángulos en el borde derecho muy cerca del centro de la figura.

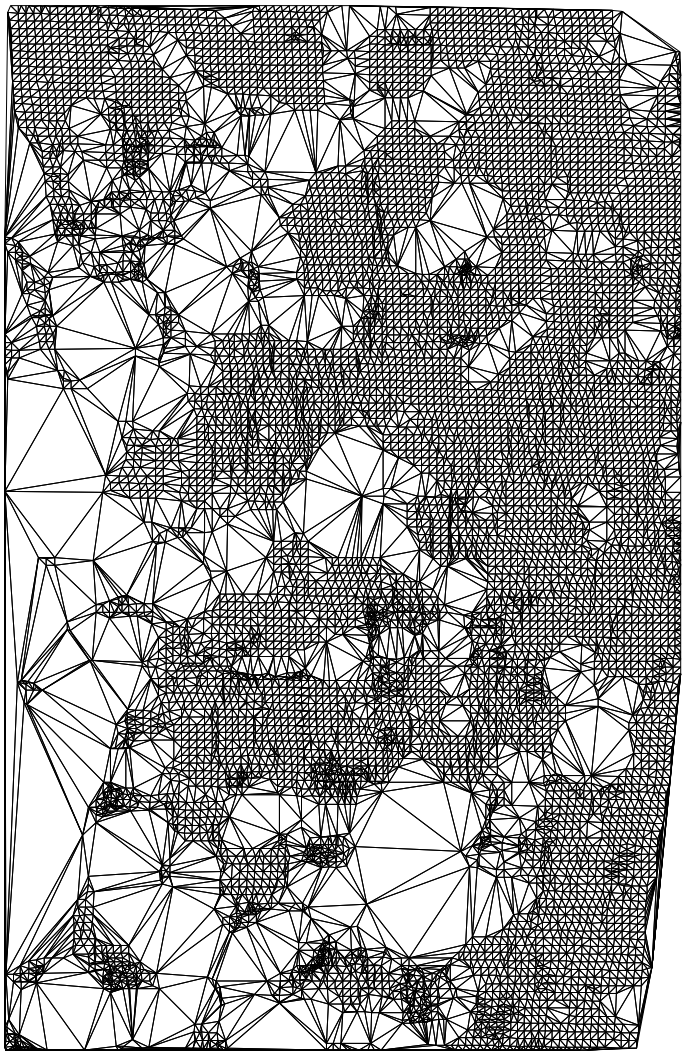

Figura 3.23: Triangulación con el algoritmo desarrollado, en alta densidad

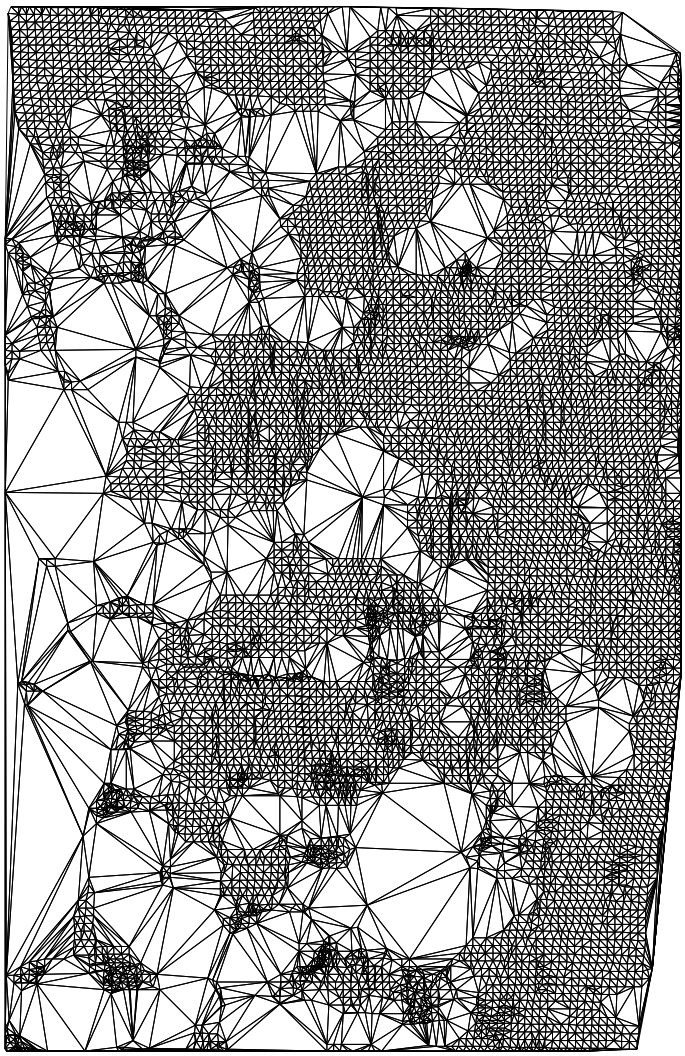

Figura 3.24: Triangulación con el *qhull*, en alta densidad

Capítulo 4

Curvas de nivel

Una vez que se ha triangulado la superficie se procede a la obtención de las curvas de nivel. Esta parte del proceso se basa en la información directa de la triangulación, utilizando el hecho de que se conocen los dos triángulos adyacentes a una arista.

Obtenidas las curvas de nivel es necesario re-triangular la superficie. Esto se necesita para llenar con triángulos el espacio entre las curvas, lo que a su vez permitirá la visualización de la superficie con pseudocolores, para cada curva en dos dimensiones en una representación tridimensional. Con esta representación en tres dimensiones se ha visto que es posible una mejor retroalimentación de la configuración de la superficie terrestre para el usuario del sistema.

4.1. Obtención de las curvas de nivel

Para la obtención de las curvas de nivel se determinan, en primera instancia, todas las curvas de nivel que pasarán por todos y cada uno de los triángulos que conforman la superficie. Una vez que se ha obtenido dicha información se toma el triángulo siguiente y para todas las curvas de nivel que pasan sobre él, *y que no han sido procesadas ya*, se determina una de las aristas que cruza (que deberán ser dos) pero, la que interesa es la arista por la que la curva sale del triángulo, sobre esta arista se obtiene el punto sobre el que la curva la corta, este punto se añade a la lista de puntos que van conformando la curva de nivel. Acto seguido se obtiene el triángulo ad-

yacente a la arista procesada del actual y se sigue este proceso por todos los triángulos por los que se vaya pasando hasta que ocurra una de dos cosas; la primera, que se regrese al triángulo del que se partió, en cuyo caso la curva de nivel es cerrada y se ha finalizado su procesamiento; en el segundo caso se tiene que llega el momento en el que el procesamiento hace que no haya ningún triángulo adyacente a la arista, en cuyo caso se tiene una curva de nivel abierta y es necesario procesar en sentido opuesto desde el triángulo del que se partió para la obtención de la otra parte de ésta.

En la Fig. 4.1 se puede observar el procesamiento seguido para la obtención de una curva de nivel cerrada y en el caso de la Fig. 4.2 el procesamiento de una curva de nivel abierta.

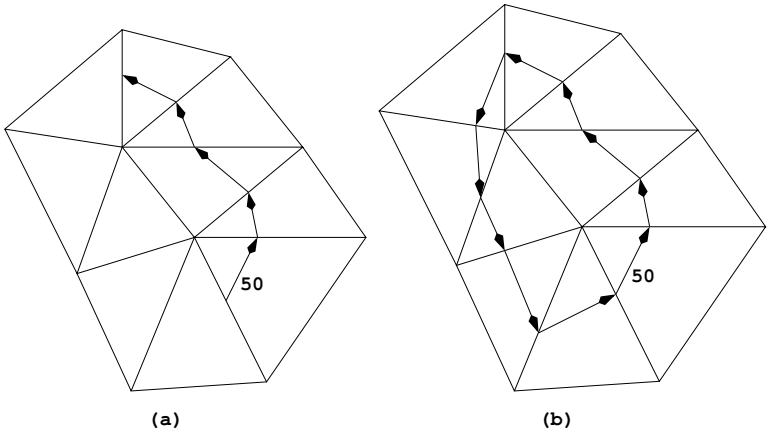

Figura 4.1: Obtención de una curva de nivel cerrada; (a) inicio del algoritmo, (b) final de la curva

En la Fig. 4.3(b) se muestran un ejemplo de las curvas de nivel obtenidas del procesamiento de la triangulación mostrada en la Fig. 4.3(a)

4.2. Re-triangulación de la superficie

Durante la obtención de las curvas de nivel se va obteniendo la nueva triangulación de la superficie de forma tal que se van conservando los límites entre curva y curva de una forma clara; esto se logra haciendo que solamente los triángulos originales sean divididos, no mezclados, en nuevos triángulos

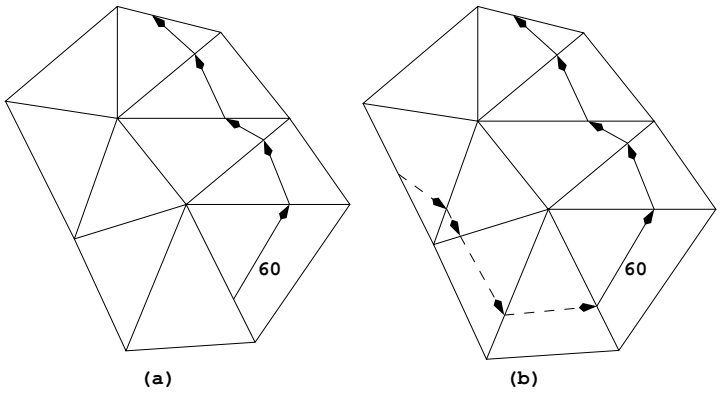

Figura 4.2: Obtención de una curva de nivel abierta; (a) inicio del algoritmo, (b) final de la curva

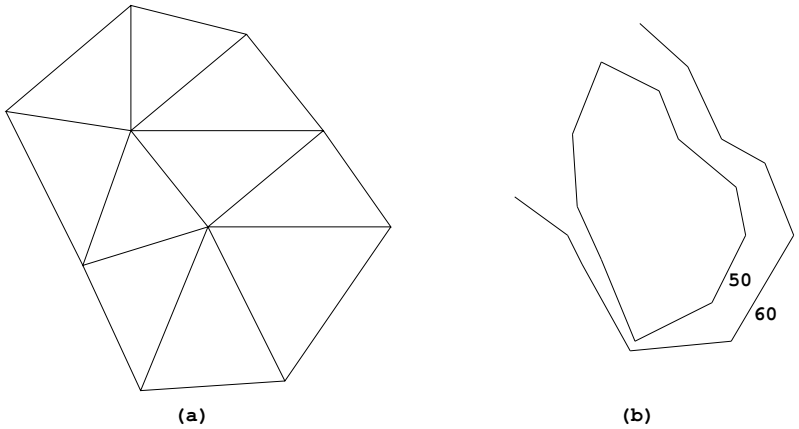

Figura 4.3: Obtención de las curvas de nivel; (a) triangulación de la superficie, (b) curvas de nivel obtenidas

que conformen la superficie; para lograr esto es necesario que sean consideradas las formas en las que las curvas de nivel pueden cortar las aristas de dichos triángulos.

Para cada forma de corte entre la curva de nivel y el triángulo correspondiente se forman nuevos triángulos que han de ser considerados para la triangulación de la superficie; en la Fig. 4.4 se muestran 3 de las 4 posibilidades de esta nueva triangulación (la primera es cuando ninguna curva de nivel corta las aristas del triángulo y, por ende, éste no se tiene que re-triangular

y pasa intacto a la triangulación de la superficie)

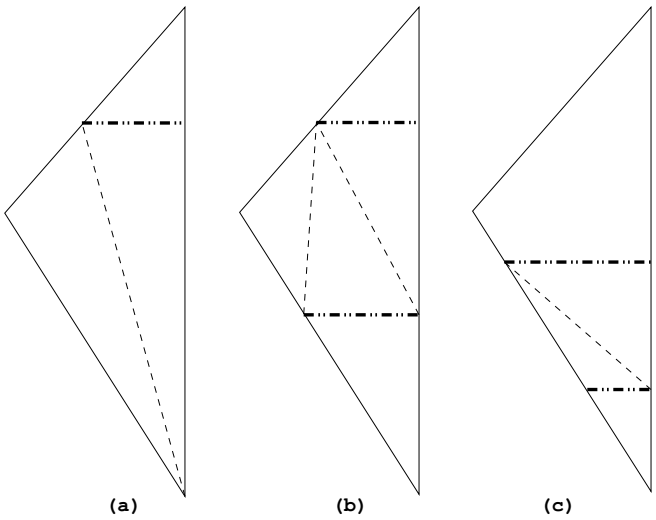

Figura 4.4: Presentación de las tres formas de corte (línea delgada punteada) entre la curva de nivel (línea gruesa punteada); (a) primer caso, (b) segundo caso, (c) tercer caso

En la Fig. 4.4(a) se muestra el caso cuando la curva de nivel corta una arista por encima (considerando la altura de cada uno de los vértices del triángulo) del vértice intermedio sin ninguna otra curva entre ésta y el vértice inferior, en este caso la re-triangulación se reduce a dividir el cuadrilátero formado en dos triángulos.

En la Fig. 4.4(b) se muestra el segundo caso, en el cual se tienen un par de curvas de nivel una por debajo del vértice intermedio y otra por encima, lo cual nos crea un pentágono que hay que dividir en tres nuevos triángulos, tal y como se aprecia en dicha figura.

Finalmente, en la Fig. 4.4(c) se puede observar el último caso, el cual consiste en dos curvas de nivel, ya sea ambas por debajo del vértice intermedio o ambas por encima, en este caso se tiene un cuadrilátero que hay que dividir en dos para obtener la nueva re-triangulación.

En la Fig. 4.5(a) se puede observar el resultado de la re-triangulación de

la Fig. 4.3(a) por efecto de las curvas de nivel de la Fig. 4.3(b), aquí pueden observarse en línea punteada las nuevas aristas que fueron creadas para la re-triangulación y en cuál de los tres casos caen cada una de ellas; y en la Fig. 4.5(b) se muestra una representación en color de la región que cubre cada una de las curvas de nivel así obtenidas.

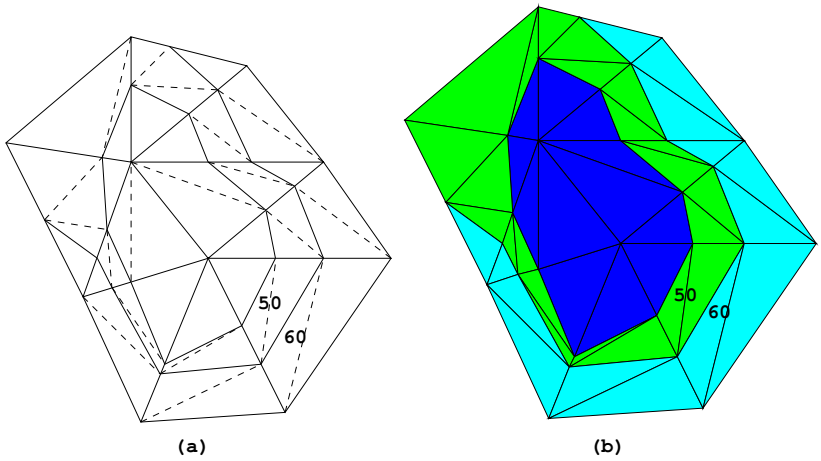

Figura 4.5: (a) Re-triangulación de las curvas de nivel, (b) límites entre curvas de nivel

4.3. Tratamiento de los triángulos degenerados

Durante la obtención de las curvas de nivel se pueden encontrar tres tipos de triángulos, en cuanto a las alturas de sus vértices se refiere (los cuales están mostrados en la Fig. 4.6):

(a) Se observa un triángulo con sus tres vértices a alturas diferentes, en este caso el procesamiento no tiene nada de especial puesto que las curvas de nivel que lo cruzan claramente entran por una arista y salen por otra

(b) En este caso se tiene un triángulo con dos de sus vértices a la misma altura, en el caso dado de que dicha altura coincida con la altura de la curva de nivel que se está obteniendo, se considera que dicha curva *no* corta esta arista

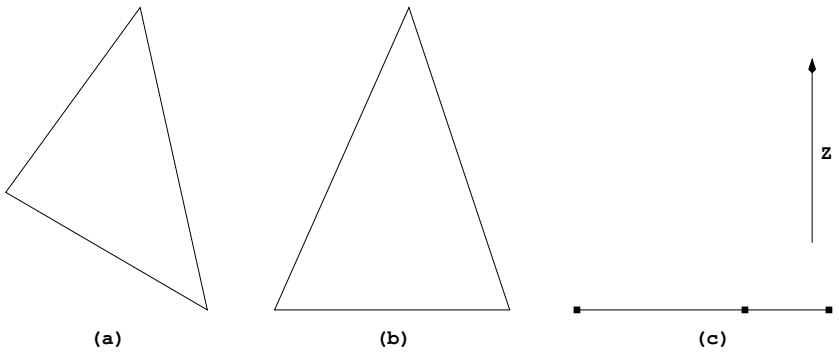

Figura 4.6: Las tres posibles situaciones de las alturas de los vértices de un triángulo; (a) sus vértices a alturas diferentes, (b) dos de sus vértices a la misma altura, (c) los tres vértices a la misma altura

(c) En este caso se tiene un triángulo completamente horizontal y se considera que ninguna curva de nivel pasa por él

4.4. Visualización tridimensional

Una vez que se han obtenido la re-triangulación de la superficie se procede a realizar la representación tridimensional de la misma en una interfaz *Qt* mediante *OpenGL*; en la Fig. 4.7 se puede apreciar la reconstrucción tridimensional de la superficie de la Fig. 4.5(b)

En ella puede apreciarse claramente la división de la superficie por las curvas de nivel; éstas dividen a dicha superficie en tres partes que se encuentran identificadas con los mismos colores.

4.5. Descripción del proceso

Para la obtención de las curvas de nivel y la re-triangulación de la superficie se sigue el proceso general a continuación descrito (las palabras mostradas al comienzo de cada punto en letra `monoespaciada` representan los nombres de las funciones programadas. Una descripción de estas funciones se haya en el apéndice D):

1. (`CreaArregloAlturas`) Se crea el arreglo de alturas para todos y cada

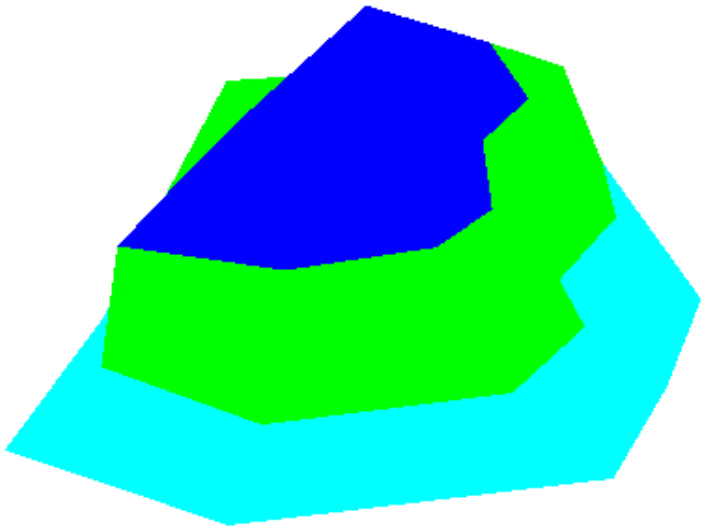

Figura 4.7: Imagen tridimensional de la superficie de la Fig. 4.5(b)

uno de los triángulos, en la Fig. 4.8 se muestra un ejemplo del arreglo de alturas creado para un triángulo

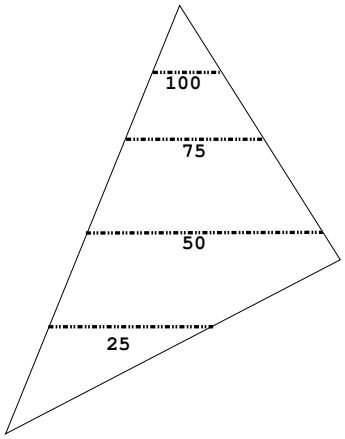

Figura 4.8: Arreglo de altura s para un triángulo: 25-50-75-100

2. (`ObtenCurvasDeNivelTriangulacion`) Cada triángulo de la triangulación original se procesa de la siguiente forma:

2.1. (`ReTriangula`) Se retriangula dependiendo de las curvas de nivel que lo cruzan tal y como se muestra en la Fig. 4.9

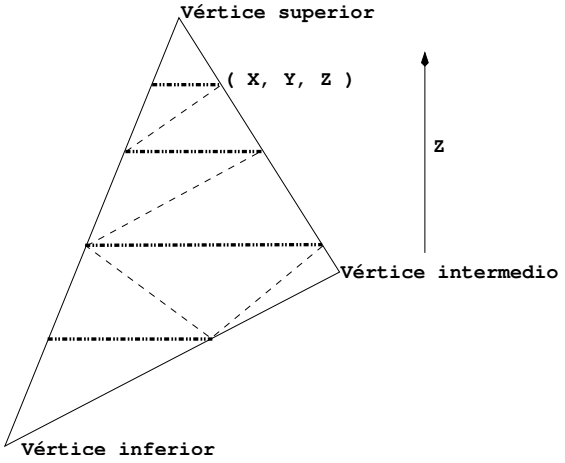

Figura 4.9: Re-triangulación de uno de los triángulos existentes; en línea gruesa punteada las curvas de nivel y en delgada punteada las aristas creadas para la nueva triangulación

2.1.1. (`ObtenOrdenTrianguloD`) Se ordenan los vértices del triángulo de tal forma que el primero sea el de menor altura

2.1.2. (`ObtenPuntoAltura3D`) Se obtienen las coordenadas de corte entre las aristas del triángulo y las curvas de nivel que lo cortan

2.1.3. (`InsertaTrianguloLista`) Dependiendo de la forma en la que las curvas cortan las aristas se van insertando los nuevos triángulos en la lista correspondiente

2.2. (`ObtenCurvasDeNivelTriangulacion`) Para cada curva de nivel no procesada

2.2.1. (`CreaCurvaDeNivel`) Se crea la curva de nivel

2.2.1.1. (`VerificaAlturaArista`) Se obtiene una de las aristas que corta la curva de nivel

2.2.1.2. (`AnadePuntosCurva`) Añade los puntos correspondientes a esa curva a la lista, este caso se puede apreciar en la Fig. 4.1 un ejemplo de curva de nivel cerrada

2.2.1.3. (`CreaCurvaDeNivel`) En caso de que se haya obtenido una curva de nivel abierta se procede a la obtención de la parte restante de la misma, este caso se puede apreciar en la Fig. 4.2

2.2.2. (`InsertaCurvaLista2D`) Se inserta en la lista de las curvas de nivel ya obtenidas en una estructura de lista de listas; este esquema puede apreciarse en la Fig. 4.10

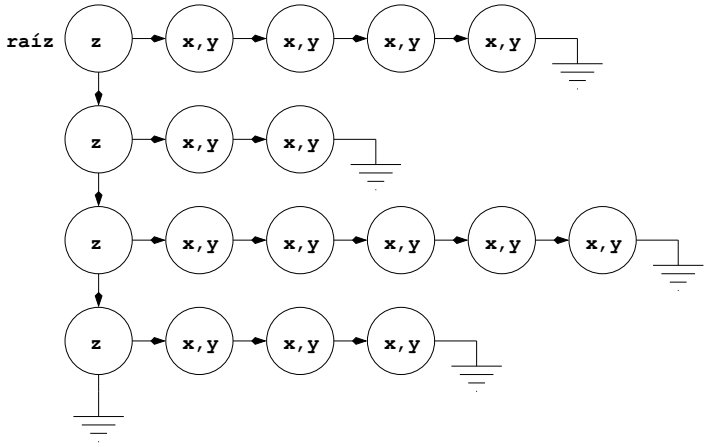

Figura 4.10: Estructura en la que se almacenan las curvas de nivel

2.2.2.1. (`CreaNodoCurva`) Se crea el nodo para guardar la curva en la lista de curvas

2.2.2.2. (`InsertaNodoCurvaLista`) Se inserta este en la lista de curvas

3. (`ObtenCurvaLista2D`) Se regresa la lista de las curvas de nivel ya obtenidas

Capítulo 5

Ejemplos del procesamiento

Para efectos de presentación del trabajo realizado se presentan aquí dos ejemplos de pares estereoscópicos, el primer par se muestra en la Fig. 5.1, el cual representa una montaña y el segundo par se muestra en la Fig. 5.2, que en este caso representa un llano.

En ambos casos los parámetros para el cálculo de la correlación fueron:

- **Tamaño del bloque**: 64 píxels
- **Desplazamiento del bloque**: 64 píxels
- **Incremento de la ventana de búsqueda en X**: 128 píxels
- **Incremento de la ventana de búsqueda en Y**: 32 píxels

Para la obtención de la triangulación de Delaunay se utilizaron:

- **Umbral de validez**: 0.3
- **Altura a la que fueron tomadas las fotografías**: 100.0 metros
- **Escala de las fotografías**: 10.0 metros por píxel
- **Desplazamiento azimutal entre las fotografías**: 300 píxels
- **Posición del azimuth en la fotografía izquierda**: 100 píxels
- **Posición del azimuth en la fotografía derecha**: 300 píxels

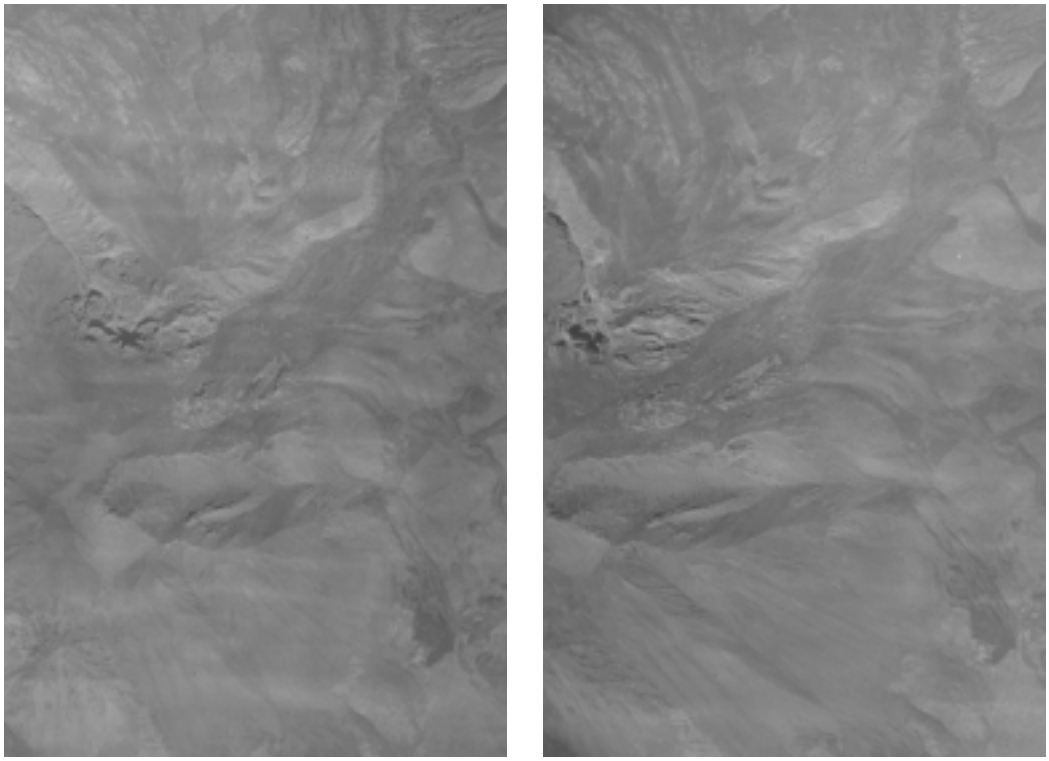

Figura 5.1: Par estereoscópico que contiene una montaña

Y para la obtención de las curvas de nivel:

- **Equidistancia para las curvas de nivel**: 25.0 metros

Los diagramas y dibujos presentados a lo largo del presente apéndice se llevaron a cabo utilizando el programa de dibujo vectorial *XFig*[19], así como con el sistema desarrollado.

5.1. Montaña

En el caso del par estereoscópico mostrado en la Fig. 5.1 se obtuvo el mapa de desplazamientos mostrado en la Fig. 5.3 en el cual se puede apreciar claramente que en la zona de la montaña los desplazamientos entre la imagen izquierda y la derecha presentan un mayor desplazamiento entre los bloques correspondientes.

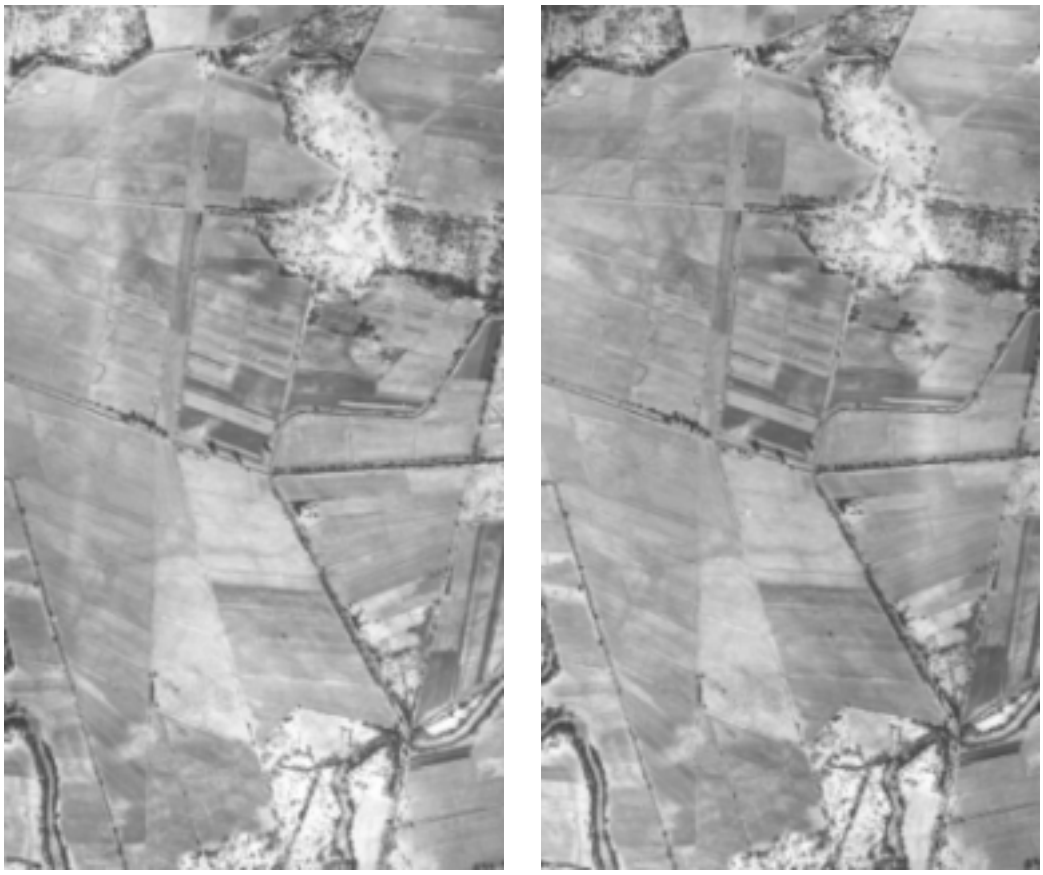

Figura 5.2: Par estereoscópico que contiene un llano

Con este mapa de desplazamientos y los parámetros respectivos (ya presentados) se obtiene la triangulación de Delaunay que se muestra en la Fig. 5.4, en donde se puede apreciar la forma en la que los valores de la altura de la toma de las fotografías y la escala de las mismas logran que los puntos cerca del borde de las mismas se desplacen de una manera que no es la adecuada, es decir, estos puntos, aunque tienen un umbral alto no coinciden de forma tal que permitan una representación adecuada del terreno.

Una vez obtenida la triangulación, se procede a la obtención de las curvas de nivel y la re-triangulación, para su representación tridimensional. Las curvas de nivel ya obtenidas (con la equidistancia ya establecida) se muestran en la Fig. 5.5 y la re-triangulación en la Fig. 5.6 en donde puede apreciarse la

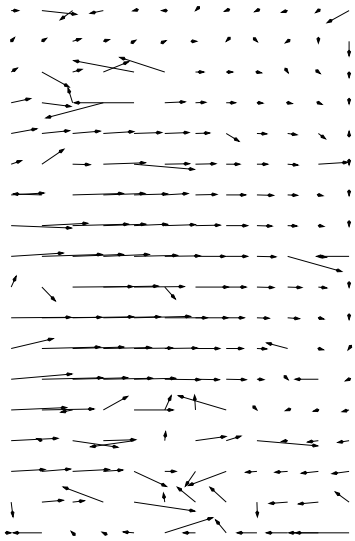

Figura 5.3: Mapa de desplazamientos después de ejecutar la correlación

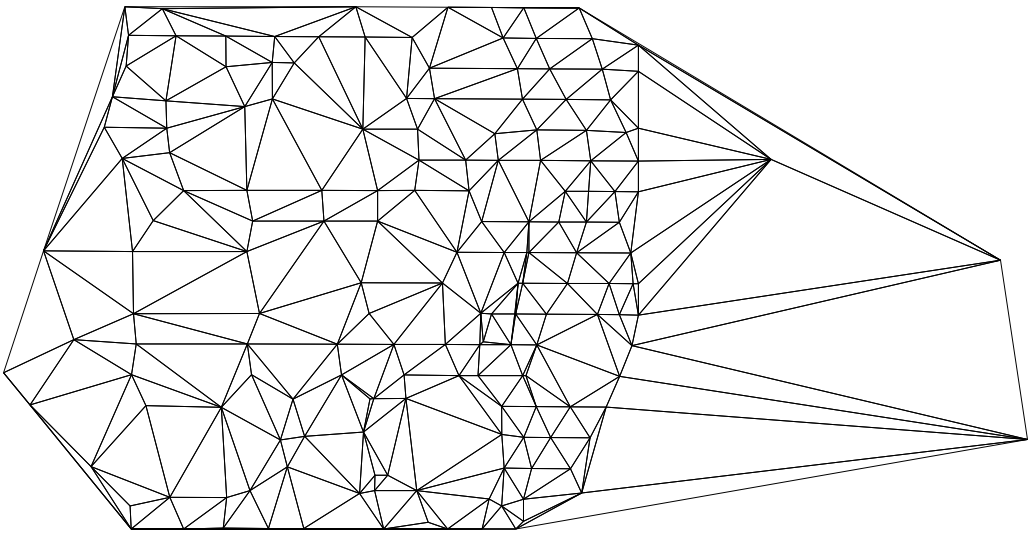

Figura 5.4: Triangulación de Delaunay obtenida con el mapa de desplazamientos

forma en la que las curvas de nivel cortan los triángulos existentes, creando nuevos que permiten su representación tridimensional.

En la Fig. 5.7 se muestra una representación en colores falsos de las regio-

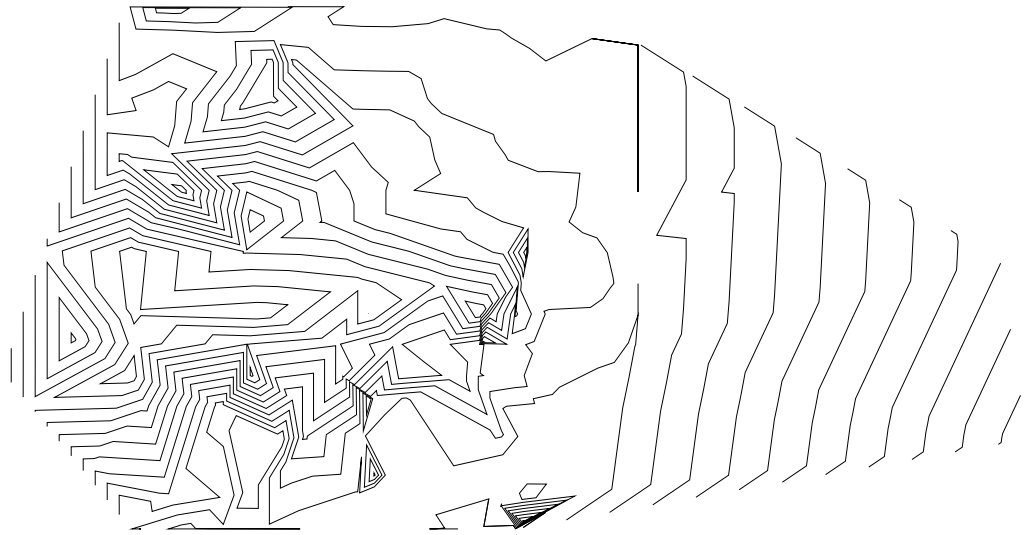

Figura 5.5: Curvas de nivel que se obtienen de la triangulación de la Fig. 5.4

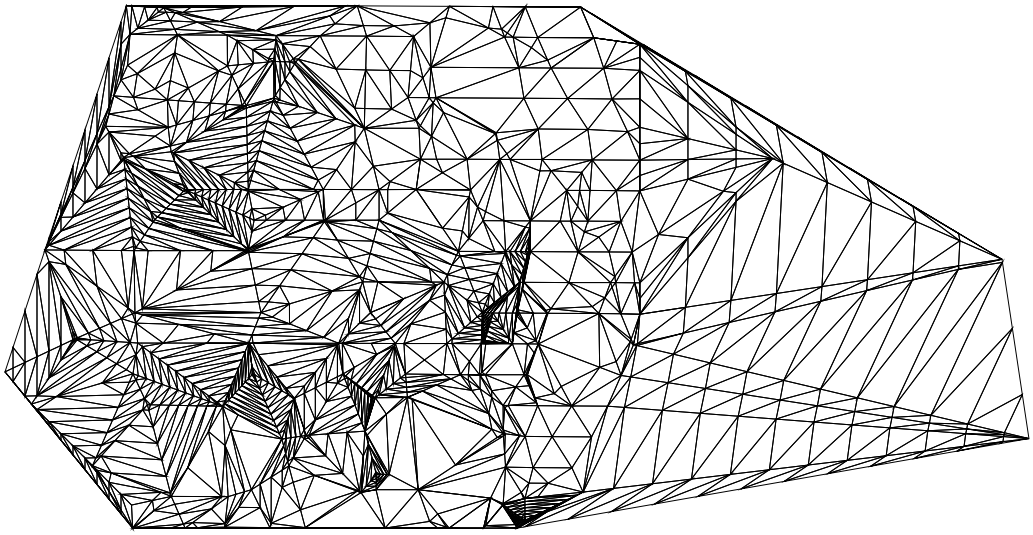

Figura 5.6: Re-triangulación de los triángulos mostrados en la Fig. 5.4 basada en las curvas de nivel de la Fig. 5.5

nes que se cubren entre curva y curva. De igual forma en esta imagen puede observarse la forma en la que los puntos que se encuentran cerca del borde hacen que las curvas de nivel y, por consiguiente, la representación tridimensional de las mismas, se vean deformadas en esos lugares.

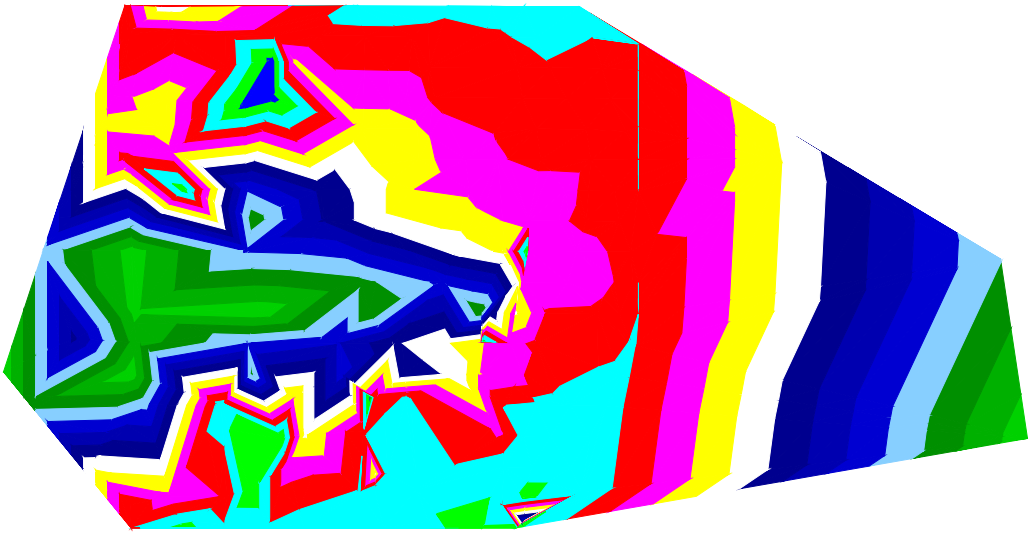

Figura 5.7: Áreas contenidas entre curva y curva de la Fig. 5.5 en colores falsos

La representación tridimensional (en el mapa de colores que puede observarse en la Fig. 5.8 y cuyos códigos en formato *RGB* (Red -rojo-, Green -verde- y Blue -azul-) pueden obtenerse de la Tabla 5.1) de estas curvas de nivel se puede observar en la Fig. 5.9 en el cual se pueden apreciar las zonas bajas en azul y las zonas altas en rojo y blanco.

Figura 5.8: Mapa de colores utilizado para la representación tridimensional de las superficies terrestres generadas

Finalmente, es necesaria la generación del archivo respectivo en formato DXF para su posterior procesamiento en el sistema de CAD. En la Fig. 5.10 se muestran las curvas de nivel en una ventana de *AutoCAD*; compárense éstas con las que se ven en la Fig. 5.5.

Nivel	Rojo	Verde	Azul
0 (más bajo)	0.0	0.4	1.0
1	0.0	0.5	0.5
2	0.6	0.8	0.3
3	0.6	0.5	0.4
4	1.0	0.6	0.0
5	0.6	0.1	0.1
6	0.8	0.4	0.4
7 (más alto)	0.8	0.8	0.8

0.0 es ausente y 1.0 es completamente saturado

Tabla 5.1: Mapa de colores utilizado para la representación tridimensional de las superficies terrestres

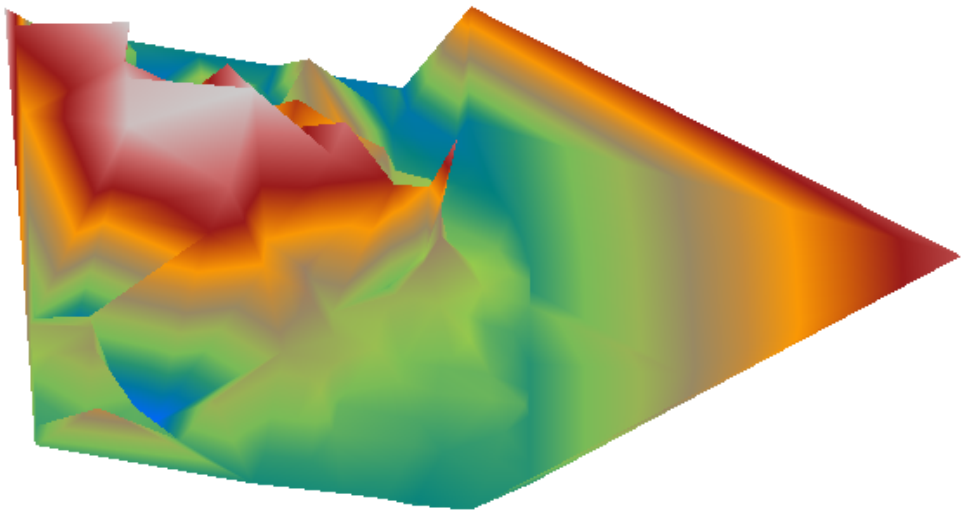

Figura 5.9: Representación tridimensional de las regiones mostradas en la Fig. 5.7

5.2. Llano

Para el caso del par estereoscópico mostrado en la Fig. 5.2 se obtuvo el mapa de desplazamientos mostrado en la Fig. 5.11 en el cual puede apreciarse claramente la zona casi completamente plana que se tiene en él; solamente se pueden observar seis puntos en los cuales los desplazamientos son altos y

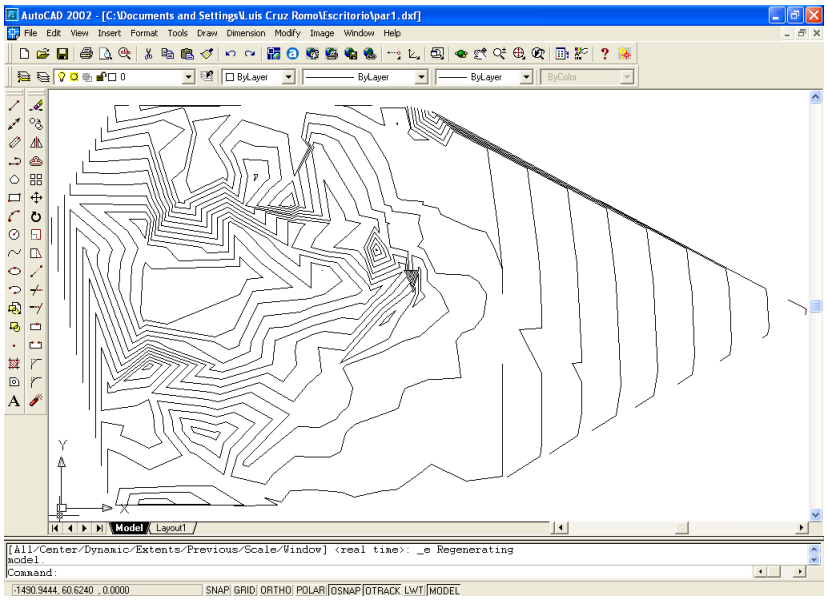

Figura 5.10: Visualización de las curvas de nivel en *AutoCAD*

que por consiguiente darán lugar a una zona elevada.

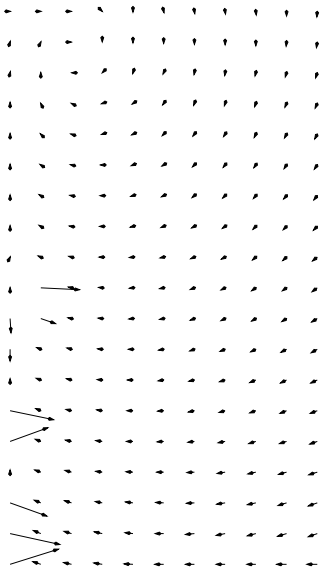

Figura 5.11: Mapa de desplazamientos después de ejecutar la correlación

Con este mapa de desplazamientos y los parámetros respectivos se obtiene la triangulación de Delaunay de la Fig. 5.12; en ésta se puede observar el patrón propio de los puntos que conforman un llano: una malla cuadrada de puntos, excepto en la zona donde los puntos obtenidos en el mapa de concordancia tienen un desplazamiento alto.

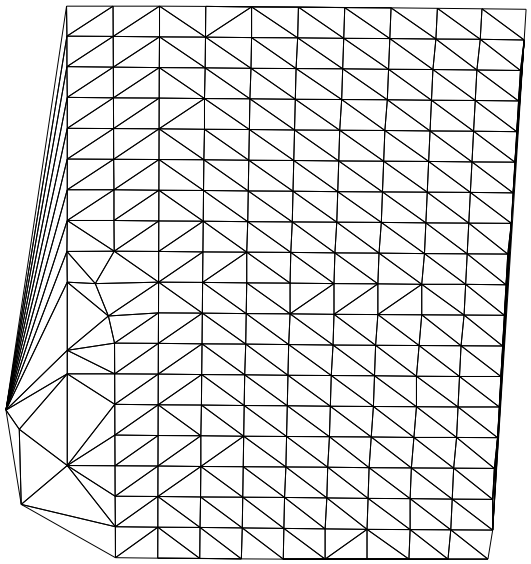

Figura 5.12: Triangulación de Delaunay obtenida con el mapa de desplazamientos

Una vez obtenida la triangulación, se procede a la obtención de las curvas de nivel y la re-triangulación para su representación tridimensional. Las curvas de nivel ya obtenidas se muestran en la Fig. 5.13 y la re-triangulación en la Fig. 5.14 en donde puede observarse que los triángulos divididos debido a las curvas de nivel son muy pocos.

En la Fig. 5.15 se muestra una representación en colores falsos de las regiones que se cubren entre curva y curva. En ella se puede observar la parte baja (plana) en color azul y las partes altas en color violeta; además, se aprecia cerca del borde una degeneración de la misma que hace que esa zona sea también alta.

La representación tridimensional de estas curvas de nivel se puede observar en la Fig. 5.16 en donde puede apreciarse el cerro de la parte izquierda

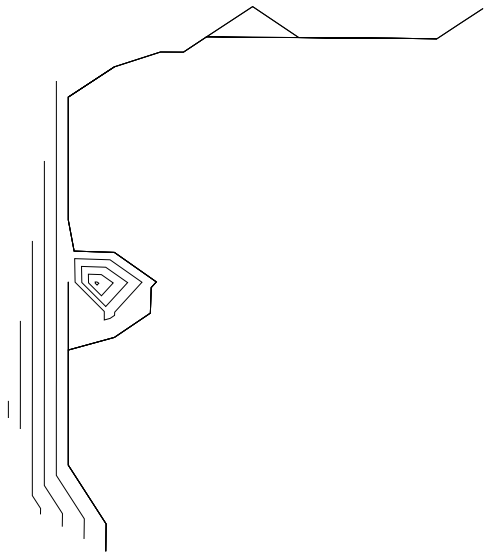

Figura 5.13: Curvas de nivel que se obtienen de la triangulación de la Fig. 5.12

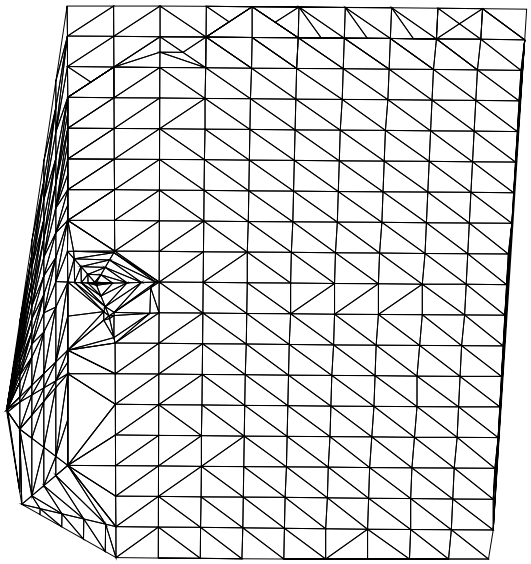

Figura 5.14: Re-triangulación de los triángulos mostrados en la Fig. 5.12 basada en las curvas de nivel de la Fig. 5.13

de la imagen; éste es el resultado de los puntos de alto desplazamiento que se muestran en la Fig. 5.11 (el mapa de colores utilizado en esta imagen es

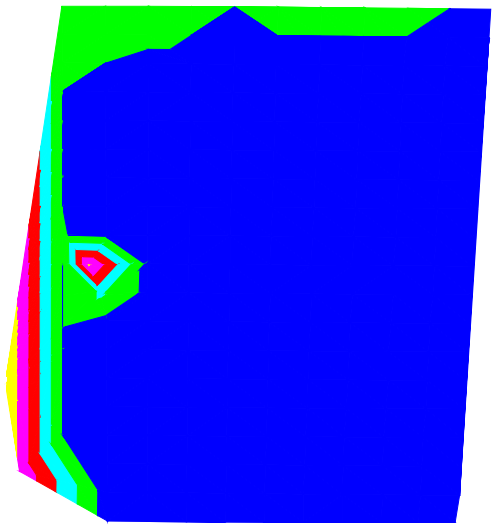

Figura 5.15: Áreas contenidas entre curva y curva de la Fig. 5.13 en colores falsos

el que se puede ver en la Fig. 5.8)

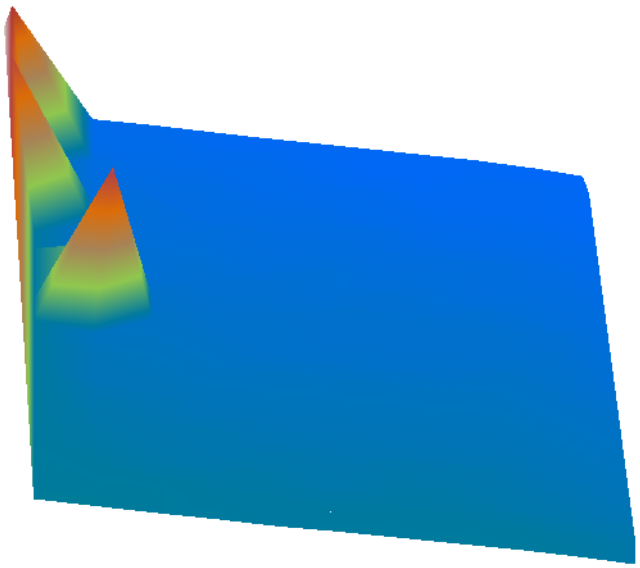

Figura 5.16: Representación tridimensional de las regiones mostradas en la Fig. 5.7

Finalmente, es necesaria la generación del archivo respectivo en formato DXF para su posterior procesamiento en el sistema de CAD, en la Fig. 5.17 se muestran las curvas de nivel en una ventana de *AutoCAD*; compárense éstas con las que se ven en la Fig. 5.13.

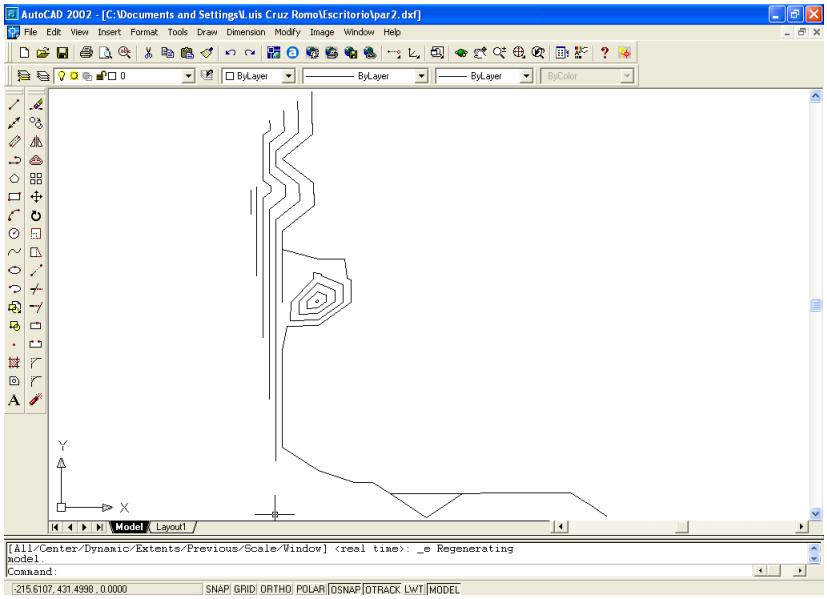

Figura 5.17: Visualización de las curvas de nivel en *AutoCAD*

Capítulo 6

Conclusiones

Del desarrollo del trabajo de tesis se pueden concluir los siguientes puntos:

1. Se ha desarrollado un sistema para la obtención automática y visualización de curvas de nivel a partir de pares de fotografías estereoscópicas.

2. Se construyó una interfaz gráfica para el sistema, esta herramienta se programó en *Qt* y *Mesa* para la visualización.

3. Se desarrolló una biblioteca de funciones en la cual se contienen rutinas para la transformada rápida de Fourier, tanto en una dimensión como en dos.

4. Se ha implementado el algoritmo de la *Correlación Cruzada Normalizada Rápida*, con el cual ha sido posible obtener la concordancia y, por ende, el mapa de desplazamientos de una forma *relativamente* eficiente.

5. Con relación a la triangulación, se modificó el algoritmo de la triangulación de Delaunay para que se conservaran para todos y cada uno de los triángulos de la misma, los triángulos con los que comparte sus aristas, esto para que en el momento de obtener las curvas de nivel se haga de una forma continua y su representación en DXF también.

6. La visualización de las curvas de nivel se obtienen por medio de colorear cada nivel de un solo color y lo que permite también una visualización tridimensional. Esta visualización se programó usando primitivas de *OpenGL*.

7. El sistema funciona de forma consistente en los casos en los que los cambios de altura son pequeños, es decir, en los que la pendiente del terreno no sufre cambios bruscos tales como los que se tendrían en una región montañosa o muy accidentada.

8. En los casos en los que el terreno incluye montañas muy pronunciadas el proceso presentado obtiene buenos resultados pero no son los adecuados a este tipo de trabajo, debido principalmente a las pendientes muy pronunciadas y, por ende, la existencia de muchas regiones ocultas o semi-ocultas dentro del par estereoscópico.

9. Para las superficies muy accidentadas, el proceso obtiene una serie de puntos que, apesar de que coinciden con la superficie terrestre, no permiten la obtención de las curvas de nivel del terreno y, por ende, a una representación tridimensional adecuada de las mismas.

   En este caso es posible que utilizando un desplazamiento de bloque más pequeño se pueda obtener una serie de puntos mucho más estrecha y de esta forma poder identificar los puntos de una mejor forma. En contra de esto se encuentra el hecho de que sería necesario un mayor tiempo de proceso puesto que la cantidad de bloques a correlacionas se incrementaría de forma cuadrática con esta reducción en el desplazamiento.

10. La identificación de las curvas de nivel para el caso de superficies muy planas es un poco complicado puesto que podrían tenerse muchos triángulos cuyas tres aristas se encontraran a la misma altura y posiblemente, lo que sería peor, que coincidieran con alguna curva de nivel, en este caso el sistema los eliminaría a todos ellos.

Se hizo una demostración a los ingenieros responsables del área de fotogrametría de la *Escuela Superior de Ingeniería y Arquitectura* (los cuales amablemente proporcionaron varios pares estereoscópicos de prueba) y los comentarios fundamentales que se hicieron son:

- El sistema en general funciona de manera adecuada, dándole los parámetros correctos.

- Según los pares estereoscópicos de demostración, el sistema no se encuentra listo para su uso en problemas reales puesto que los resultados

obtenidos no han sido homologados con curvas de nivel reales (tales como las del INEGI) sobre fotografías reales.

- El sistema es un paso adelante a la automatización de la obtención de las curvas de nivel.

- La academia de vías terrestres solicitó una copia del sistema para que los alumnos puedan comparar su trabajo contra los resultados del sistema.

Durante el desarrollo del presente trabajo se han notado diferentes situaciones susceptibles de ser mejoradas, modificadas o, incluso, cambiadas completamente; en este capítulo se expondrán algunas de ellas.

Algunas de estas mejoras se encuentran motivadas por las conclusiones que se han obtenido de la utilización del sistema.

6.1. Posibles mejoras

Dentro de las posibles mejoras al sistema elaborado se pueden mencionar las siguientes:

- En el caso de la evaluación de la transformada de Fourier, es *posible* el mejorar el algoritmo considerando que se tiene una subdivisión en bloques de tamaño menor en ambas imágenes (hay que recordar que para la aplicación de la FFT se va calculando ésta en bloques de tamaño $1, 2, 4 \ldots$); esto podría hacerse de forma tal que se evaluara la FFT en los bloques más grandes posibles dependiendo de los desplazamientos y tamaños de los bloques del algoritmo tal y como puede apreciarse en la Fig. 6.1

  Implementar esta mejora implica un mayor movimiento (léase añadir, borrar y cambiar más líneas de código) dentro del sistema.

- Para el caso del cálculo de la energía se puede mejorar el rendimiento en un pequeño porcentaje pre-calculando la matriz de energías de la imagen derecha tal y como se hace con la imagen izquierda.

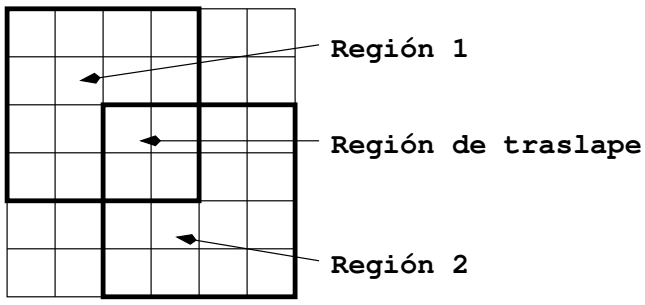

Figura 6.1: La FFT de la región 2 puede obtenerse utilizando la FFT de la región de traslape, la cual ya ha sido obtenida al calcular la FFT de la región 1

- Dentro de la triangulación, el manejo de los puntos que *caen* sobre una de las aristas ya existentes; en este caso el sistema crea los tres triángulos que subdividen al actual (de estos triángulos uno es degenerado, siendo solamente una recta) y cuando se revisa la condición de Delaunay, se subdivide el triángulo opuesto, lo que causa que se generen más triángulos temporales innecesarios.

  La mejora en este caso consiste en determinar si el punto cae sobre una arista y, en ese caso, subdividir los dos triángulos que la comparten; de esa forma, el sistema no genera triángulos degenerados y las verificaciones son más rápidas puesto que no se crean tantas ramificaciones innecesarias dentro del árbol de la triangulación; este esquema se puede apreciar en la Fig. 6.2

- Otra posible mejora dentro de la triangulación es la de conservar un arreglo de los vértices de los triángulos que son opuestos a las aristas que comparten con el triángulo actual, es decir, en vez de calcular el vértice opuesto a la arista elegida, conservarlo en un campo dentro de la misma estructura que representa el triángulo dentro del árbol.

En el caso de la interfaz de usuario lo que se puede mejorar incluye:

- Mejora de la parte de la interfaz en la cual se muestran las curvas de nivel para poder observar directamente las alturas a las que pasa cada una de ellas.

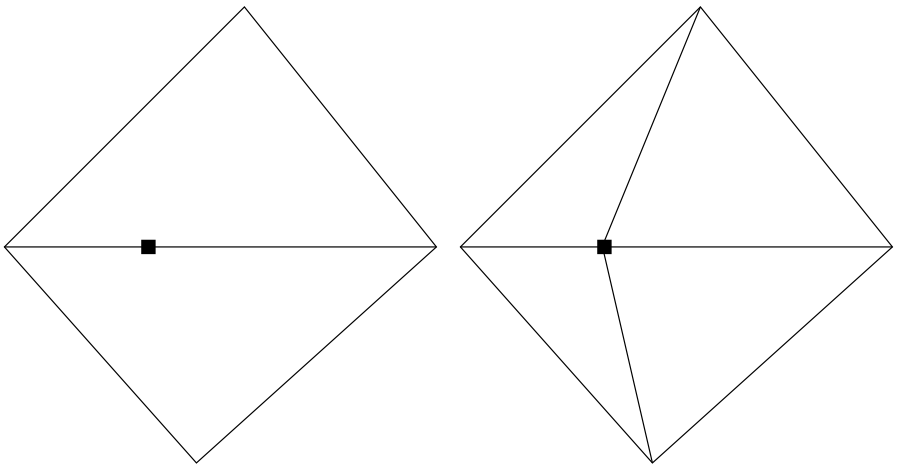

Figura 6.2: Cuando el punto que se está añadiendo cae sobre una arista, es viable la creación de los cuatro triángulos en que se dividen ambos opuestos a dicha arista

- Posibilidad de edición previa a la exportación en DXF de las curvas de nivel.
- Mejora en el algoritmo de dibujado de la representación tridimensional de las curvas de nivel para utilizar listas de dibujo de *OpenGL* con lo cual se mejora el rendimiento del mismo.

6.2. Trabajo futuro

En la siguiente lista se muestran algunos puntos en los cuales es necesaria una re-escritura mayor dentro del sistema, pero que, sin embargo, podrían llevar a una mejora significativa del mismo:

- Utilización de una modificación del algoritmo descrito en [12], en el cual se describe la forma de ir obteniendo la correspondencia entre partes cada vez más estrechas entre ambas imágenes; la modificación podría incluir la forma de determinar las correspondencias globales y su aplicación dentro del problema que se ha planteado aquí.

  También sería necesaria la modificación del algoritmo de triangulación para que vaya considerando las triangulaciones previas.

- Identificación de las regiones en donde la correlación no sea muy fuerte y, en esos casos, realizar un *estiramiento* de las mismas (en base a la altura correspondiente) tratando de identificar si de esta forma la correlación mejora o no.

  En este caso el tiempo de procesamiento se incrementaría de forma significativa puesto que es necesario hacer el *estiramiento* de alguna forma *inteligente* para trarar de conservar el detalle de ambos bloques.

- Mejora del algoritmo de triangulación, tal vez utilizando un algoritmo paralelo del tipo *divide y vencerás*; por ejemplo el que se describe en [11].

- Idear un algoritmo para identificar las partes de la triangulación que se encuentran muy cerca del borde de las imágenes, ya que los puntos en estos bordes causan problemas en el momento del cálculo de las coordenadas reales de dichos puntos.

Apéndice A

Interfaz de usuario

Dentro del desarrollo del sistema es necesaria una interfaz que le permita al usuario interactuar con los resultados del proceso realizado sobre el par estereoscópico, ya sea para poder observar las curvas de nivel o la representación tridimensional de éstas. Es en esta interfaz en la que precisamente puede obtener alguna información extra que le permita una mejor toma de decisiones acerca del proyecto que esté actualmente evaluando.

Esta interfaz de usuario se encuentra desarrollada en *KDE*[18] con la biblioteca de objetos *Qt*[16] mediante el estándar *OpenGL*[13], lo cual permite la presentación gráfica de las imágenes a procesar, la representación esquemática de las curvas de nivel y la presentación tridimensional de la superficie terrestre de una forma fácil y, sobre todo, suficientemente amigable al usuario que la va utilizar, para esta parte del desarrollo se utilizó [7].

Se hicieron estas elecciones puesto que el ambiente de trabajo (*framework*) de *Qt* es suficientemente flexible y fácil de utilizar para el desarrollo de interfaces gráficas, además el de ser multiplataforma, lo cual conlleva muchas ventajas, una de ellas la de poder ejecutar el sistema tanto en Linux como en Windows y, por ende, solamente se necesita recompilar la aplicación.

En el caso de *OpenGL* se eligió por el hecho de ser una tecnología madura que se ejecuta también en las plataformas antes mencionadas de una forma confiable y eficiente, sobre todo si se tiene hardware que permita la aceleración de sus primitivas; la interfaz de *OpenGL* es clara y fácil de usar, además permite la visualización de objetos en tercera dimensión de una manera sen-

cilla.

Dicha interfaz de usuario está desarrollada en una máquina con sistema operativo GNU/Linux[22] de la distribución de RedHat[23] versión 8, la cual incluye la versión 3.0.5 de *Qt* y la versión 4.2.1 de *Mesa*[15] que es la biblioteca de funciones de *OpenGL* libre para Linux/UNIX (ver las referencias [5] y [10])

En la Fig. A.1 se puede apreciar la pantalla inicial que indica que la aplicación se está cargando.

Figura A.1: Pantalla inicial de la carga de la aplicación en donde se muestra el nombre de la aplicación y los autores

En la Fig. A.2 se puede observar la interfaz de usuario sin ninguna imagen cargada en ella tal y como se le presenta una vez cargado el programa, en ella se pueden apreciar las pestañas de las que se compone éste: *Imágenes*, *Datos*, *Curvas de nivel* y *Visualización 3D*, donde cada una de ellas será explicada en su momento.

En la barra de herramientas se tiene un botón que permite mostrar la pantalla de información del sistema, dicho botón se muestra en la Fig. A.3 y la pantalla en la Fig. A.4

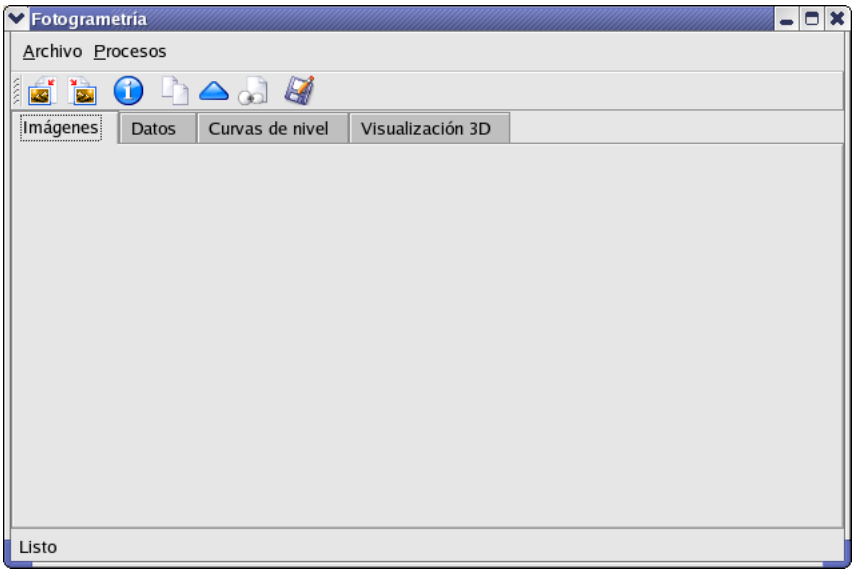

Figura A.2: La interfaz que se le muestra al usuario cuando se acaba de cargar el sistema

Figura A.3: Botón de la barra de herramientas que muestra la pantalla de información del sistema

A.1. Especificación de las imágenes

Para el procesamiento de las curvas de nivel es necesaria la especificación y carga de las imágenes que serán utilizadas para este fin; dentro de la interfaz existen un par de botones (estos se pueden observar en la Fig. A.5) y opciones del menú que permiten especificarlas, éstas opciones se muestran en la Fig. A.6.

Cuando se selecciona una de las opciones de carga de imágenes se muestra la interfaz que se observa en la Fig. A.7 para que el usuario pueda seleccionar las fotografías necesarias.

En la Fig. A.8 se muestra la interfaz del usuario con un par de fotografías

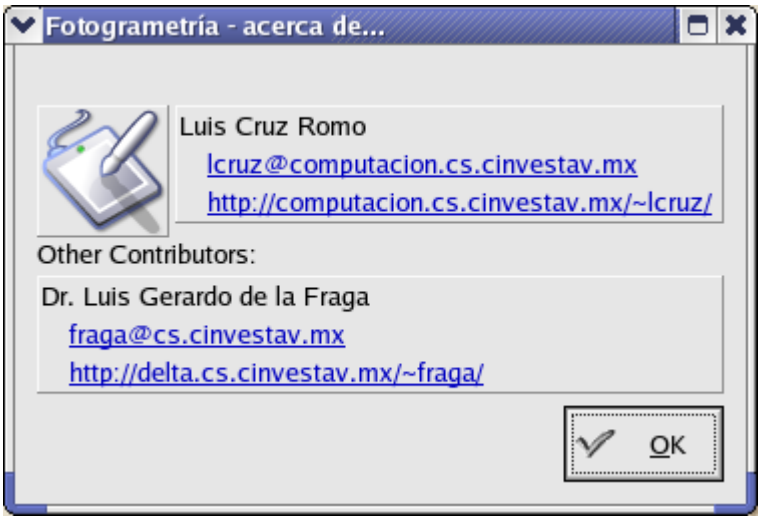

Figura A.4: Pantalla de información de la interfaz de usuario

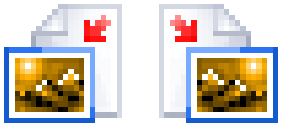

Figura A.5: Botones de la barra de herramientas para especificar las imágenes a utilizar en la obtención de las curvas de nivel; izquierda y derecha

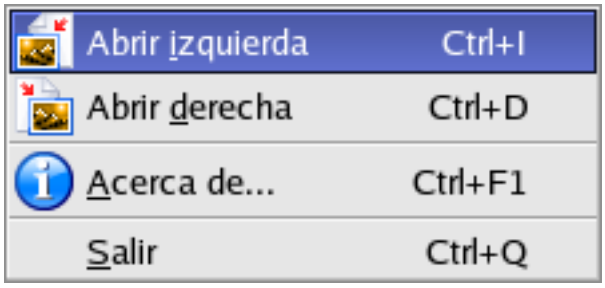

Figura A.6: Menú con las opciones para cargar las imágenes a ser procesadas

estereoscópicas ya cargadas y listas para su procesamiento.

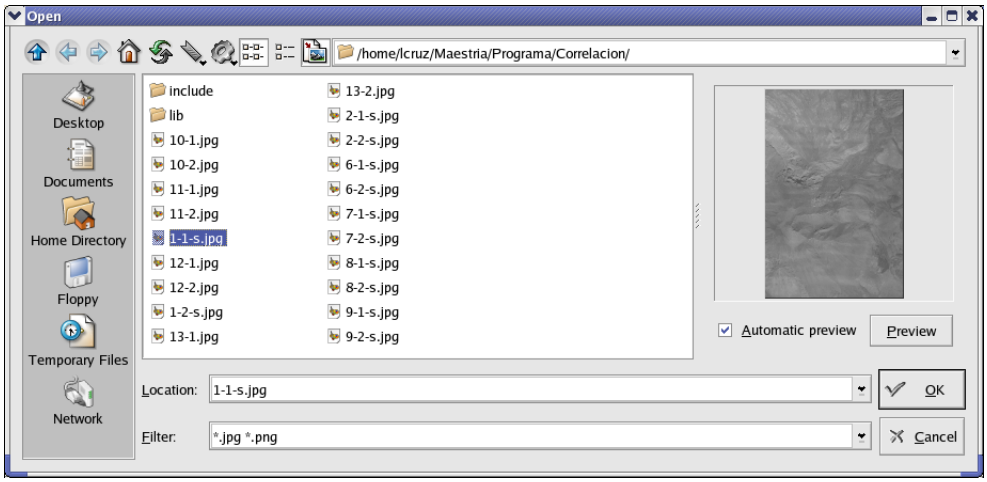

Figura A.7: Pantalla para la selección de las imágenes a procesar

A.2. Parámetros para el cálculo

En la Fig. A.9 se muestra la pestaña de la interfaz en donde se dan de alta los parámetros para el cálculo; claramente se observan las tres partes en las que se divien éstos:

- *Correlación*: Los primeros cuatro parámetros se utilizan para el cálculo de la correlación entre las imágenes tal y como se ve en la Fig. A.10:
  - *Tamaño del bloque (píxels)* Aquí se especifica el tamaño del bloque que se utilizará para la correlación (64 por omisión)
  - *Desplazamiento del bloque (píxels)* En este campo se especifica el desplazamiento del bloque de correlación; puede observarse que si este valor es menor que el anterior ocurrirá un traslape entre ellos (64 por omisión)
  - *Incremento de la ventana de búsqueda en X (píxels)* Aquí se especifica el incremento al tamaño del bloque, en dirección X, para hacer la ventana de búsqueda (128 por omisión)
  - *Incremento de la ventana de búsqueda en Y (píxels)* Aquí se especifica el incremento al tamaño del bloque, en dirección Y, para hacer la ventana de búsqueda (32 por omisión)

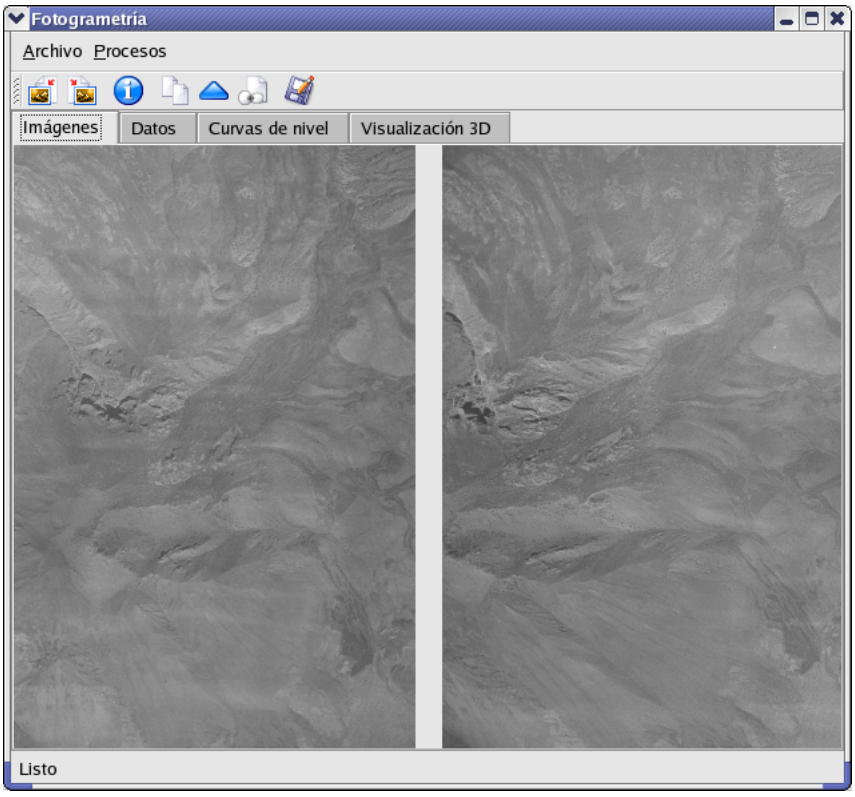

Figura A.8: La interfaz que se le muestra al usuario cuando se acaban de cargar las imágenes

- *Triangulación*: Los siguientes seis parámetros se utilizan para la determinación de la triangulación de los puntos del mapa de desplazamientos, éstos se dividen en dos partes, la primera (el primer parámetro) para la determinación de los puntos que se considerarán válidos del mapa y la segunda parte (los otros cinco parámetros) para la determinación de las coordenadas tridimensionales de dichos puntos; dichos parámetros se muestran en la Fig. A.11:

  - *Umbral de validez (0.0 a 1.0)* En este campo se especifica el umbral del valor de la correlación para determinar si se incluye un punto o no en el mapa de concordancia (0.3 por omisión)
  - *Altura a la que fue tomada la foto (metros)* Aquí se introduce la altura (en metros) a la que fueron tomadas ambas fotografías

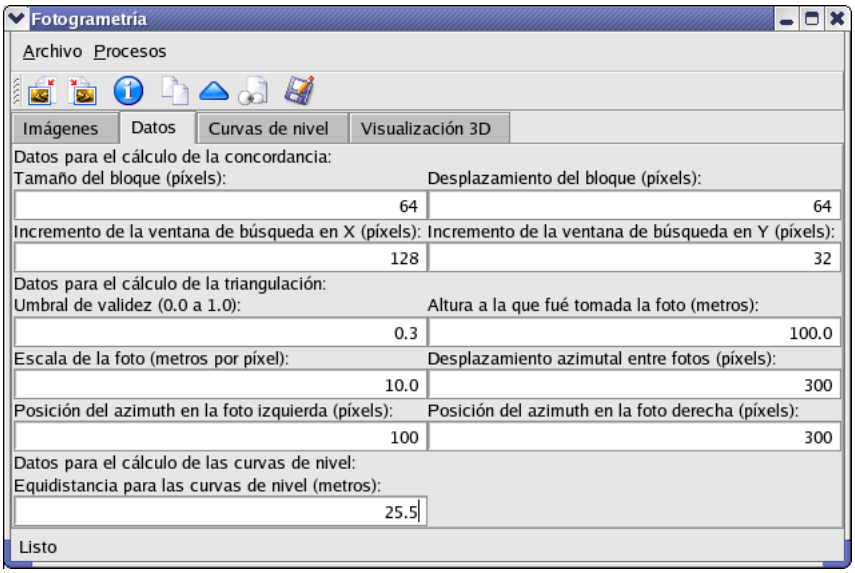

Figura A.9: La interfaz que se le muestra al usuario para la especificación de los parámetros para el cálculo

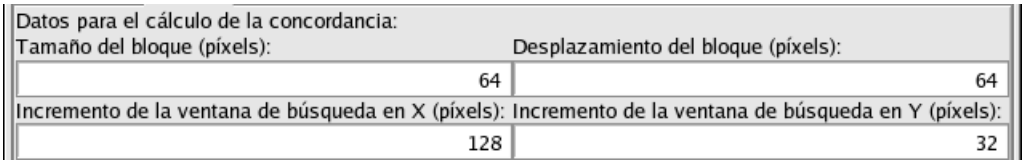

Figura A.10: Parámetros que se utilizan en la correlación de las imágenes

- *Escala de la foto (metros por píxel)* Aquí se introduce la escala de las imágenes de las fotografías, en metros por píxel
- *Desplazamiento azimutal entre fotos (píxels)* En este campo se especifica el desplazamiento azimutal entre fotografías, es decir, el desplazamiento de los azimuths del avión entre la toma de una fotografía y la otra, medido en las imágenes (en píxels)
- *Posición del azimuth en la foto izquierda (píxels)* En este campo se especifica la posición del azimuth del avión en la fotografía izquierda, en píxels, medido desde el margen izquierdo
- *Posición del azimuth en la foto derecha (píxels)* Aquí se introduce la posición del azimuth del avión en la fotografía derecha, en píxels, medido desde el margen izquierdo

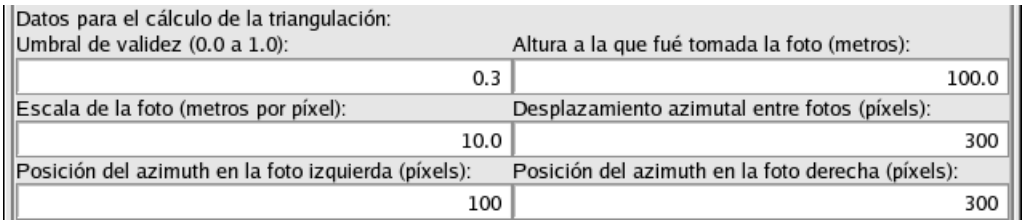

Figura A.11: Parámetros que se utilizan en la triangulación del mapa de desplazamientos

- *Curvas de nivel*: El último parámetro que es necesario proporcionar se utiliza como equidistancia para la obtención de las curvas de nivel; este se puede observar en la Fig. A.12:

  - *Equidistancia para las curvas de nivel (metros)* En este campo se introduce la equidistancia (distancia vertical) entre cada curva de nivel y su inmediara superior o su inmediata inferior, en metros

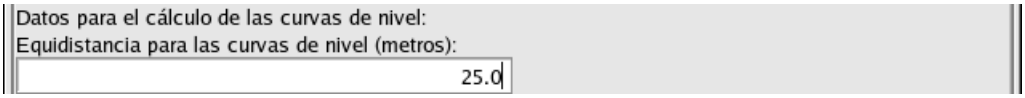

Figura A.12: Parámetro que se utiliza para la determinación de las curvas de nivel

Una vez que se han especificado los parámetros que se utilizarán para el procesamiento del par estereoscópico, se puede proceder a realizar éste. Es posible realizar el procesamiento de forma parcial en cada etapa (*concordancia*, *trignaulación* y *curvas de nivel*) introduciendo solamente los parámetros que en ellas se utilicen.

A.3. Procesamiento

Para el procesamiento de las imágenes ya cargadas y establecidos los parámetros para el cálculo, se procede a realizar el procesamiento de éstas para su posterior presentación.

Dentro del menú se tienen las opciones correspondientes para el cálculo de las curvas de nivel (estas se muestran en la Fig. A.13) así como botones

en la barra de herramientas (estas se muestran en la Fig. A.14), desde la obtención de la correlación, la triangulación hasta las curvas de nivel y su representación tridimensional.

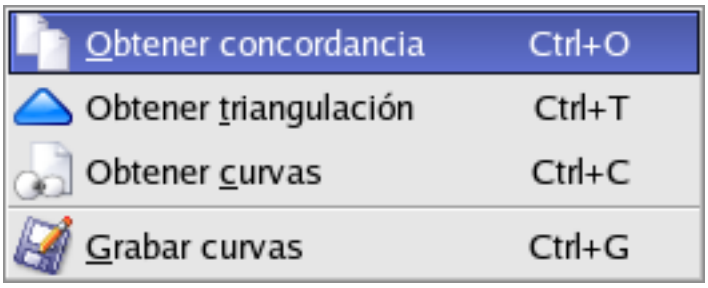

Figura A.13: Menú con las opciones para el procesamiento de las imágenes

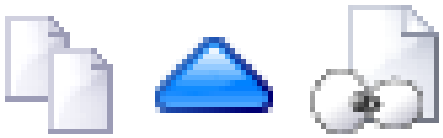

Figura A.14: Botones de la barra de herramientas para el procesamiento; concordancia, triangulación y curvas de nivel

Cuando se comienza el procesamiento de la concordancia entre las imágenes se abre la ventana mostrada en la Fig. A.15 para indicar el porcentaje del cálculo que se ha completado, ya que esta parte del proceso es la más lenta.

A.4. Las curvas de nivel

Dentro de la interfaz de usuario se tiene una pestaña para la presentación de las curvas de nivel como se hayan obtenido según los parámetros que dicho usuario haya introducido; un ejemplo de dicha representación se muestra en la Fig. A.16

Dichas curvas de nivel se obtuvieron con los datos introducidos como se observan en la Fig. A.9 y procediendo a aplicar los procesos *Procesos* →

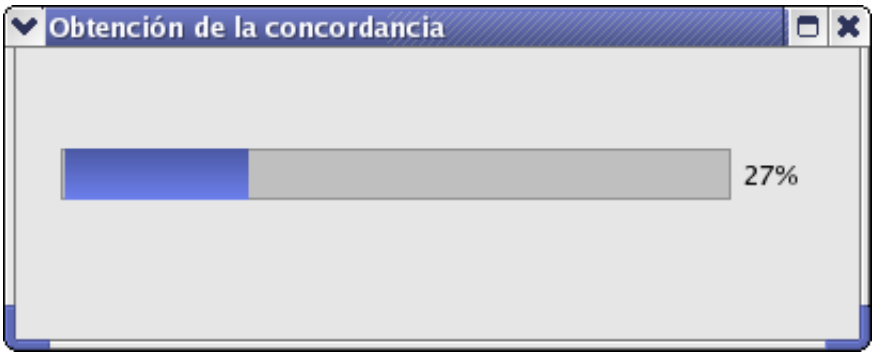

Figura A.15: Ventana en donde se muestra el porcentaje completado del cálculo de la concordancia

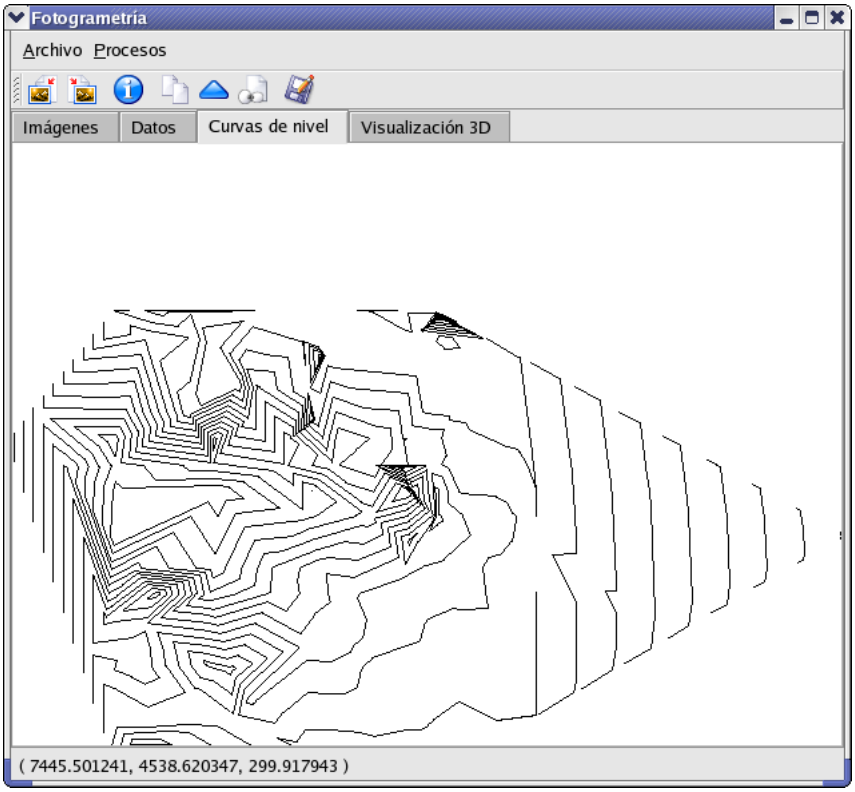

Figura A.16: La interfaz en donde se le muestran al usuario las curvas de nivel ya obtenidas

Obtener concordancia, *Obtener triangulación* y *Obtener curvas*, en ese orden.

En la Fig. A.17 se puede observar que en la barra de estado se muestran las coordenadas tridimensionales del punto sobre el que se encuentra el apuntador del ratón, siempre y cuando se haye dentro del área cubierta por la triangulación original.

Figura A.17: Parte de la interfaz en donde se muestran las coordenadas del punto sobre el que se encuentra el apuntador del ratón

Una vez que se han obtenido las curvas de nivel es necesario exportarlas a un archivo *DXF* (Drawing eXchange Format, formato de intercambio de dibujo, definido por AutoCAD[25]) para su posterior procesamiento en un sistema de CAD (tal como AutoCAD o QCad[26]). En la Fig. A.18 se muestra el botón de la barra de herramientas que permite guardar las curvas de nivel.

Figura A.18: Botón de la barra de herramientas para grabar las curvas de nivel en formato DXF

En la Fig. A.19 se muestra la pantalla que permite especificar el archivo en donde se guardarán dichas curvas.

A.5. Representación tridimensional

Dentro de la interfaz de usuario se tiene también una pestaña para la representación tridimensional de las curvas de nivel y la superficie terrestre tal y como se hayan obtenido después de calcularlas. El par estereoscópico que se muestra en la Fig. A.8, una vez procesado con los prámetros anteriores,

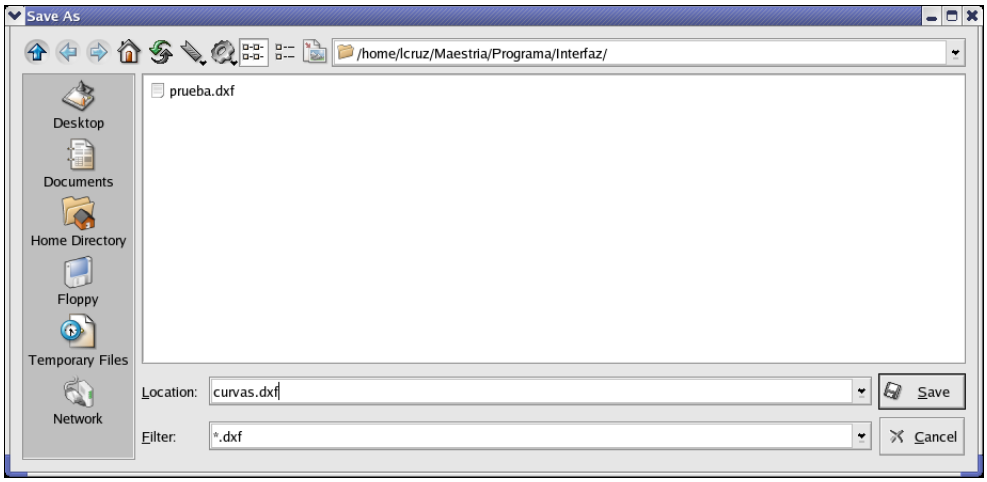

Figura A.19: Pantalla para la selección del archivo *DXF* en donde se grabarán las curvas de nivel obtenidas

se puede observar en la Fig. A.20 en colores que dependen de la altura de la curva de nivel (la escala de colores puede verse en la Fig. A.21)

En la Tabla A.1 se pueden observar los códigos en formato *RGB* del mapa de colores de la Fig. A.21.

Nivel	Rojo	Verde	Azul
0 (más bajo)	0.0	0.4	1.0
1	0.0	0.5	0.5
2	0.6	0.8	0.3
3	0.6	0.5	0.4
4	1.0	0.6	0.0
5	0.6	0.1	0.1
6	0.8	0.4	0.4
7 (más alto)	0.8	0.8	0.8

0.0 es ausente y 1.0 es completamente saturado

Tabla A.1: Mapa de colores utilizado para la representación tridimensional de las superficies terrestres

La forma en la que el sistema determina el color de cada uno de los

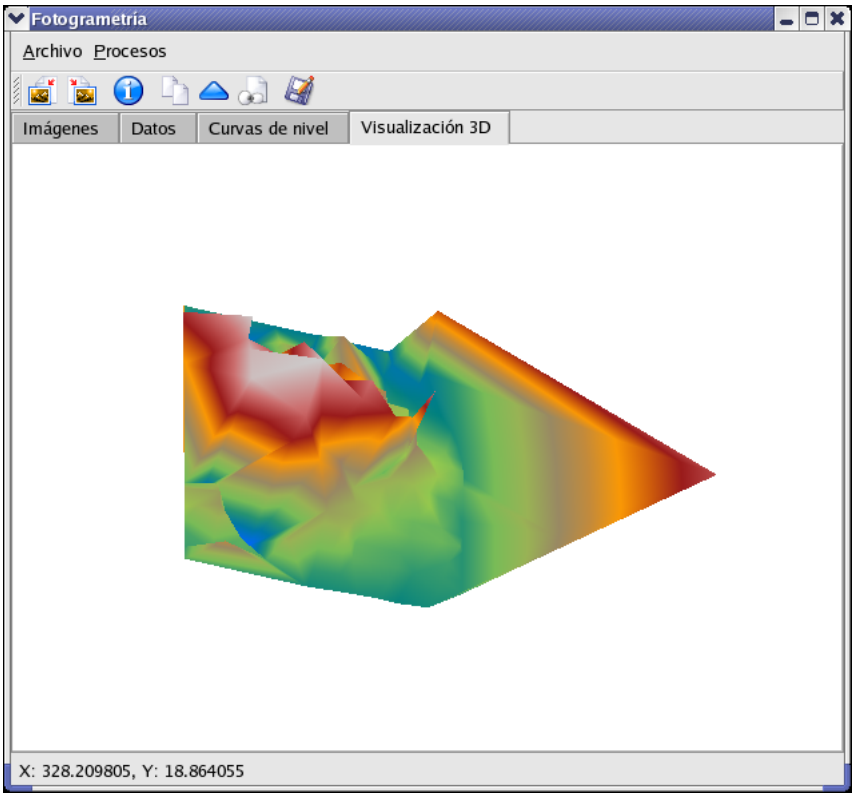

Figura A.20: Imagen de una pantalla en donde se puede apreciar la representación tridimensional del terreno cuyas curvas de nivel se han obtenido

Figura A.21: Mapa de colores utilizado para la representación tridimensional de las superficies terrestres generadas

vértices de la re-triangulación es la siguiente:

- Obtiene las alturas mínima y máxima de toda la superficie y les asigna los colores del nivel **0** y **7**, respectivamente
- Determina entre qué niveles cae el vértice que se está especificando en ese momento (supóngase que se cae entre los colores **3** y el **4** con una proporción de 0.4 y 0.6 de cada color)
- Con esas proporciones se procede a calcular las correspondientes com-

ponentes por separado mediante una regla de tres; para el ejemplo:

$$rojo = 0.6 + 0.4(1.0 - 0.6) = 0.76$$

$$verde = 0.5 + 0.4(0.6 - 0.5) = 0.54$$

$$azul = 0.4 + 0.4(0.0 - 0.4) = 0.24$$

por lo que el color para ese vértice (en formato RGB) es (0.76, 0.54, 0.24)

En esta pestaña de la interfaz es posible la rotación de la imagen tridimensional mostrada, de la siguiente manera:

- **Botón izquierdo del ratón**: Giro en el eje x desplazando el ratón de arriba (giro negativo) a abajo (giro positivo)
- **Botón izquierdo del ratón**: Giro en el eje y desplazando el ratón de izquierda (giro negativo) a derecha (giro positivo)

Estos ángulos serán conservados en todas las visualizaciones tridimensionales de los pares estereoscópicos, a menos que se pulse sobre el botón en la barra de herramientas (éste se muestra en la Fig. A.22) o hasta que el programa sea terminado y se vuelva a cargar, en cuyo caso los ángulos volverán a ser de cero.

Figura A.22: Botón de la barra de herramientas para poner en ceros los ángulos de giro de la representación tridimensional de las curvas de nivel

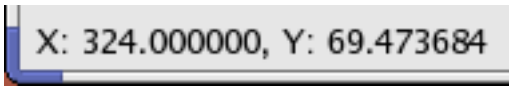

Figura A.23: Parte de la interfaz en donde se muestran los ángulos de giro con los que se han dibujado las curvas de nivel (sobre el terreno)

Mientras se efectua este movimiento, en la barra de estado pueden apreciarse los ángulos actuales con los que ha sido dibujada la imagen tridimensional del terreno (ésta puede observarse en la Fig. A.23) tanto en el eje x como en el y.

Apéndice B

Dibujo en XFig

Durante el desarrollo de este trabajo se ha usado de forma extensiva la interfaz gráfica que proporciona el programa XFig; se han generado archivos que este puede leer y procesar.

Aquí se describe la forma en la que se han generado los diferentes objetos gráficos que se han utilizado:

- Puntos
- Líneas
- Flechas
- Poli-líneas
- Áreas

B.1. Preliminares

En el cuerpo del archivo *.fig* se tiene un encabezado con la siguiente información:

- `#FIG 3.2`: Indica la versión del XFig que se utilizó
- `Landscape`: Indica la orientación de la página
- `Center`: Este campo no se utilizó durante las pruebas realizadas

- `Inches`: Especifica que se han de utilizar medidas en pulgadas
- `Letter`: El tamaño de la página
- `100.00`: La escala del dibujo como un porcentaje
- `Single`: Este campo no se utilizó durante las pruebas realizadas
- `-2`: Este campo no se utilizó durante las pruebas realizadas
- `1200 2`: Este campo no se utilizó durante las pruebas realizadas

B.2. Puntos

```
(1)  2 1 0 20 0 7 50 0 -1 0.000 0 0 -1 0 0 1
(2)       200 300
```

Un punto se encuentra constituído por un par de coordenadas planas, por lo que éstas se especifican en la línea siguiente `(2)` a la del elemento geométrico `(1)` En este caso se tiene un punto en las coordenadas `(200, 300)` según la unidad de medida establecida.

B.3. Líneas

```
(1)  2 1 0 1 0 7 50 0 -1 0.000 0 0 -1 0 0 2
(2)       200 300 400 500
```

En el caso de una línea, ésta se extiende entre dos puntos por lo que es necesario especificar éstos, en `(1)` se tiene una entrada al final, la cual indica que ese elemento geométrico consta de dos puntos y en `(2)` se colocan las coordenadas de esos dos puntos, según la unidad de medida establecida.

B.4. Flechas

```
(1)  2 1 0 1 0 7 50 0 -1 0.000 0 0 -1 1 0 2
(2)       1 1.00 60.00 120.00
(3)       200 300 400 500
```

En el caso de las flechas es necesario especificar el tipo y dimensiones de la punta de flecha, los datos mostrados en (1) y (3) son los mismos que en el caso de una línea, excepto que en (1) los campos 14 (1) y 15 (0) representan los lugares de la línea en donde se dibujarán las puntas de flecha, que, en este caso es al comienzo de la misma; en (2) se tiene que especificar el tipo de punta de flecha (1), del primer (1) tipo, el campo siguiente (1.00) representa la escala de la misma y los campos tercero (60.00) y cuarto (120.00) son la escala (expresada en porcentaje) transversal y longitudinal de la punta de flecha.

B.5. Poli-líneas

```
(1)  2 1 0 1 0 1 50 0 -1 0.000 0 0 -1 0 0 n
(2)       200 300 400 500 600 700
(3)       800 900 1000 1100...
```

Una poli-línea es un objeto geométrico representado por medio de una sucesión de puntos; en este caso se tiene una poli-línea de n puntos, cuya primer coordenada es (200, 300), la segunda (400, 500) y así sucesivamente. Si la poli-línea se compone de muchos puntos, éstos pueden incluírse en líneas separadas, siempre y cuando dichas líneas comienzen con un caracter de tabulación, tal y como se puede observar en (3)

B.6. Áreas

```
(1)  2 1 0 1 c c 50 0 20 0.000 0 0 -1 0 0 n
(2)       200 300 400 500 600 700...
```

Para poder especificar una región cerrada es necesario indicar el color con el cual se hará el relleno tal y como se muestra en (1) con la indicación c c en los campos quinto y sexto; posteriormente se deberán especificar los puntos de los que se compone el perímetro (poli-línea) que la contiene como en (2).

Los colores utilizados por XFig para el trazo de las figuras geométricas y, en particular para el relleno de las regiones cerradas se muestra en la Tabla B.1

Código	Color	Código	Color	Código	Color	Código	Color
0		1		2		3	
4		5		6		7	
8		9		10		11	
12		13		14		15	
16		17		18		19	
20		21		22		23	
24		25		26		27	
28		29		30		31	

Tabla B.1: Paleta de colores utilizada por XFig para el manejo del color

Apéndice C

Generación del formato DXF

Dentro del sistema es necesaria la generación de las curvas de nivel en formato DXF para su posterior procesamiento en algún sistema de CAD (tales como *AutoCAD* o *QCad*), esta generación se realiza tomando en cuenta que la única parte estrictamente indispensable dentro del archivo DXF es la de entidades, es decir, un archivo DXF mínimo tiene la siguiente estructura[27][28]:

```
  0
SECTION
  2
ENTITIES
...
  0
ENDSEC
  0
EOF
```

en donde los tres puntos indican que en esa posición se deberán de insertar las entidades correspondientes, que en este caso son solamente líneas tridimensionales; estas líneas tienen el formato siguiente:

```
  0
LINE
  8
DEFAULT
 10
```

```
Coordenada inicial en X
 20
Coordenada inicial en Y
 30
Coordenada inicial en Z
 11
Coordenada final en X
 21
Coordenada final en Y
 31
Coordenada final en Z
```

De esta manera están generados los archivos en DXF. Como ejemplo de archivo mínimo con dos líneas se tiene:

```
  0
SECTION
  2
ENTITIES
  0
LINE
  8
DEFAULT
 10
100.0
 20
150.0
 30
200.0
 11
120.0
 21
160.0
 31
200.0
  0
LINE
```

```
  8
DEFAULT
 10
125.0
 20
175.0
 30
150.0
 11
345.0
 21
215.0
 31
150.0
  0
ENDSEC
  0
EOF
```

Apéndice D

Curvatura de la Tierra

Para poder considerar las imágenes que serán procesadas por el algoritmo ya descrito es necesario realizar el *aplanamiento* de las mismas, ya que aunque las fotografías son planas, la superficie de la Tierra no lo es, así que se hace necesario un preprocesamiento de ésas imágenes para poder considerarlas *planas*.

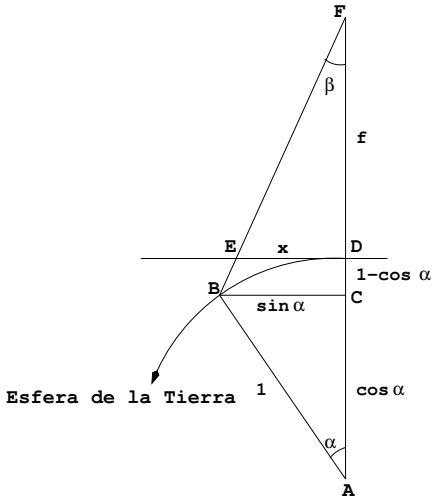

Figura D.1: Esquema representativo de la curvatura de la Tierra

En la Fig. D.1 se muestra la representación de la curvatura de la Tierra, en ella se tiene: A el centro de la Tierra, B el límite de la superficie que se capta desde la cámara, C la proyección de B sobre el radio que une el centro

con el avión, D el punto del azimuth del avión, E la posición vista de B, F la posición del avión, α el ángulo que se forma entre B, A y C y β la mitad del ángulo focal de la cámara.

En esta figura se supone que el radio de la Tierra es de la unidad, además $\overline{FD} = f$ es la altura del avión, $\overline{DE} = x$ es la distancia observada de la superficie terrestre, $\overline{AC} = \cos\alpha$, $\overline{BC} = \sin\alpha$ y $\overline{CD} = 1 - \cos\alpha$

De la relación de los triángulos semejantes FCB y FDE se tiene

$$\frac{1 + f - \cos\alpha}{\sin\alpha} = \frac{f}{x}$$

de la cual se obtiene:

$$x = \frac{f \sin\alpha}{1 + f - \cos\alpha} \tag{D.1}$$

Por otro lado, se tiene:

$$x = f \tan\beta \tag{D.2}$$

donde 2β ángulo focal de la cámara.

Tomando en cuenta las Ecs. (D.1) y (D.2) se obtiene

$$\tan\beta = \frac{\sin\alpha}{1 + f - \cos\alpha} \tag{D.3}$$

Para la obtención entre el tamaño real entre el observado se utiliza

$$r = \frac{\alpha}{x} = \frac{\alpha}{f \tan\beta} \tag{D.4}$$

En el apéndice E se muestra el código utilizado para el cálculo de la Tabla D.1, en el cual se puede observar la relación entre el ángulo focal de la cámara, la altura a la que se toman las fotografías y el incremento del tamaño a aplicar en las imágenes para poder considerar éstas como realmente *planas*, se ha considerado que el radio de la Tierra es de 6378 kilómetros. En dicho cálculo se utilizaron la Ec. (D.3) para la obtención del ángulo α y la Ec. (D.4) para la obtención del incremento en el tamaño de las imágenes.

Al analizar los resultados mostrados en la Tabla D.1 se puede concluir que la curvatura de la Tierra juega un papel despreciable en este problema puesto que la cantidad máxima de píxels que hay que *estirar* una imagen de 4000 píxels es de solo 3, lo que conlleva a que el esfuerzo para hacerlo no lleve un aumento de la exactitud de la solución obtenida.

2β	f (km)				
	6.378	12.756	19.134	25.512	31.89
30°	1.000036	1.000072	1.000108	1.000144	1.000180
	4000.143	4000.287	4000.431	4000.575	4000.719
35°	1.000050	1.000100	1.000150	1.000200	1.000249
	4000.199	4000.398	4000.598	4000.798	4000.997
40°	1.000066	1.000133	1.000199	1.000265	1.000332
	4000.265	4000.530	4000.795	4001.061	4001.327
45°	1.000086	1.000172	1.000258	1.000344	1.000430
	4000.343	4000.687	4001.031	4001.375	4001.720
50°	1.000109	1.000218	1.000327	1.000436	1.000545
	4000.435	4000.870	4001.306	4001.743	4002.180
55°	1.000136	1.000271	1.000407	1.000544	1.000680
	4000.542	4001.085	4001.629	4002.174	4002.719
60°	1.000167	1.000334	1.000502	1.000668	1.000836
	4000.668	4001.337	4002.007	4002.673	4003.344

Tabla D.1: Cálculo del incremento en tamaño debido a la curvatura de la Tierra, el valor superior de cada fila representan la relación r de la Ec. (D.4) y el valor inferior el tamaño al que habría que *estirar* una imagen de 4000 píxels

Apéndice E

Funciones selectas del sistema

E.1. Funciones de la concordancia

```
/*

   Funci'on que obtiene la concordancia entre dos im'agenes de escala gris.

   Argumentos:
   const pImagenGris imgo - La imagen origen.
   const pImagenGris imgc - La imagen a correlacionar.
   int t                  - El tama~no de los bloques a buscar.
   int d                  - El desplazamiento en X de cada bloque.
   int tx                 - El tama~no en X de la ventana origen a buscar.
   int ty                 - El tama~no en Y de la ventana origen a buscar.
   pppConcordancia conc   - La matriz de concordancias encontrada.
   pInt nf                - La cantidad de filas de la matriz de concordancia.
   pInt nc                - La cantidad de columnas de la matriz de concordancia.

   Regresa:
   true si todo estuvo bien y false si hubo alg'un error.

*/
bool Concordancia( const pImagenGris imgo, const pImagenGris imgc, int t, int d, int tx, int ty,
                   pppConcordancia conc, pInt nf, pInt nc );

/*

   Funci'on que obtiene las energ'ias de toda la ventana especificada.

   Argumentos:
   const pImagenGris img - La imagen a obtenerle la energ'ia.
   int tx                - El tama~no en X de los bloques.
   int ty                - El tama~no en Y de los bloques.
   ppMatriz energia      - El arreglo en donde quedar'a la energ'ia de cada bloque.

   Regresa:
   true si todo estuvo bien y false si hubo alg'un error.
```

```
*/
bool ObtenEnergias( const pImagenGris img, int tx, int ty, ppMatriz energia );

/*

   Funci'on que obtiene la correlaci'on de dos imagenes grises.

   Argumentos:
   const pImagenGris imgI - La imagen inicial.
   const pMatriz energiaI - La matriz de energ'ia de la imagen inicial.
   const pImagenGris imgB - La imagen que se buscar'a en la inicial.
   int x                  - La coordenada X en la imagen inicial donde se inicia la b'usqueda.
   int y                  - La coordenada Y en la imagen inicial donde se inicia la b'usqueda.
   int tx                 - El tama~no del desplazamiento en X.
   int ty                 - El tama~no del desplazamiento en Y.
   pInt px                - La posici'on en X de la m'axima correlaci'on.
   pInt py                - La posici'on en Y de la m'axima correlaci'on.
   pDouble corr           - El resultado de la correlaci'on.

   Regresa:
   true si todo estuvo bien y false si hubo alg'un error.

*/
bool Correlacion( const pImagenGris imgI, const pMatriz energiaI, const pImagenGris imgB,
                  int x, int y, int tx, int ty, pInt px, pInt py, pDouble corr );

/*

   Funci'on que normaliza la imagen, haci'endola real.

   Argumentos:
   const pImagenGris imgo - La imagen gris origen.
   int tx                 - El tama~no en X de la matriz ya normalizada.
   int ty                 - El tama~no en Y de la matriz ya normalizada.
   ppMatriz imgd          - La imagen normalizada destino.
   pDouble energia        - La energ'ia de la imagen.

   Regresa:
   true si todo estuvo bien y false si hubo alg'un error.

*/
bool NormalizaImagen( const pImagenGris imgo, int tx, int ty, ppMatriz imgd, pDouble energia );

/*

   Funci'on que regresa la menor potencia de dos que es menor o igual al valor especificado.

   Argumentos:
   int n - El valor a verificar.

   Regresa:
   La potencia de dos respectiva.

*/
int ObtenPotenciaDeDos( int n );

/*
```

```
   Funci'on que obtiene la convoluci'on de dos matrices de n'umeros reales.

   Argumentos:
   int nf             - La cantidad de filas de la matriz.
   int nc             - La canDirectoriotidad de columnas de la matriz.
   const ppDouble re1 - La primer matriz.
   const ppDouble re2 - La segunda matriz.
   ppDouble re        - El resultado de la convoluci'on.

   Regresa:
   true si todo estuvo bien y false si hubo alg'un error.

   Modifica el arreglo 're' en l'inea.

*/
bool Convolucion( int nf, int nc, const ppDouble re1, const ppDouble re2, ppDouble re );

/*

   Funci'on que eval'ua la FFT de la matriz especificada.

   Argumentos:
   const ppDouble vRe - La parte real del arreglo al que hay que evaluarle la FFT.
   const ppDouble vIm - La parte imaginaria del arreglo al que hay que evaluarle la FFT.
   ppDouble fRe       - La parte real de la FFT (cos)
   ppDouble fIm       - La parte imaginaria de la FFT (sen)
   int nf             - La cantidad de filas de la matriz.
   int nc             - La cantidad de columnas de la matriz.
   bool tipo          - Indica si se ha de efectuar la transformada directa o inversa.

   Regresa:
   true si todo estuvo bien y false si hubo alg'un error.

*/
bool FFT2( const ppDouble vRe, const ppDouble vIm, ppDouble fRe, ppDouble fIm, int nf, int nc,
           bool tipo );
```

E.2. Funciones de la triangulación de Delaunay

```
/*

   Funci'on que ordena los puntos de forma lexicogr'afica para evitar los duplicados.

   Argumentos:
   int *orden          - Los puntos ya ordenados.

   Regresa:
   Nada.

*/
```

```
void OrdenaLexicograficamente( int *orden );

/*

   Funci'on que obtiene el valor de la coordenada m'axima de todos los puntos.

   Argumentos:
   Ninguno.

   Regresa:
   El valor de la coordenada m'axima.

*/
double ValorMaximo( void );

/*

   Funci'on que crea el primer tri'angulo del algoritmo.

   Argumentos:
   Ninguno.

   Regresa:
   true si todo bien y false si hubo alg'un error.

*/
bool PrimerTriangulo( void );

/* Estructura para el manejo del 'arbol de tri'angulos */
typedef struct __tArbolTrianguloD
{
   pPunto3D punto[ 3 ]; /* Los puntos que conforman el tri'angulo */
   bool terminal; /* Si el nodo representa un tri'angulo terminal */
   int indice; /* El 'indice que le corresponde en el arreglo */
   struct __tArbolTrianguloD *sig; /* El nodo siguiente en la lista de nodos */
   struct __tArbolTrianguloD *opuesto[ 3 ]; /* Los nodos cuyos tri'angulos son opuestos */
   struct __tArbolTrianguloD *hijo[ 3 ]; /* Los nodos hijos */
} tArbolTrianguloD; /* El tipo para los nodos del 'arbol de tri'angulos */

/*

   Funci'on que busca el tri'angulo en el que se colocar'a el punto especificado.

   Argumentos:
   const pPunto3D punto   - El punto a ubicar.
   pArbolTrianguloD arbol - El 'arbol donde se har'a la b'usqueda.

   Regresa:
   El nodo en donde el punto se ubica.

*/
pArbolTrianguloD UbicaPunto( const pPunto3D punto, pArbolTrianguloD arbol );

/*

   Funci'on que indica si un punto se encuentra dentro del tri'angulo especificado.
```

```
   Argumentos:
   const pPunto3D punto             - El punto a verificar.
   const pArbolTrianguloD triangulo - El tri'angulo a verificar.

   Regresa:
   true si se encuentra dentro del tri'angulo y false si se encuentra fuera.

*/
bool DentroDeTriangulo( const pPunto3D punto, const pArbolTrianguloD triangulo );

/*

   Funci'on que crea un nodo de un tri'angulo.

   Argumentos:
   const pPunto3D punto1  - El primer punto del tri'angulo.
   const pPunto3D punto2  - El segundo punto del tri'angulo.
   const pPunto3D punto3  - El tercer punto del tri'angulo.
   ppArbolTrianguloD nodo - El nodo a ser creado.

   Regresa
   true si todo bien y false si hubo alg'un error.

   El valor nodo es modificado dentro de esta funci'on.

*/
bool CreaNodoPuntos( const pPunto3D punto1, const pPunto3D punto2, const pPunto3D punto3,
                     ppArbolTrianguloD nodo );

/*

   Funci'on que inserta un tri'angulo en la cola de tri'angulos.

   Argumentos:
   const pArbolTrianguloD triangulo - El tri'angulo a ser insertado.
   int vertice                      - El v'ertice que se deber'a procesar.

   Regresa:
   true si todo bien y false si hubo alg'un error.

   El valor apuntado por final se modifica dentro de esta funci'on.

*/
bool InsertaTrianguloListaD( const pArbolTrianguloD triangulo, int vertice );

/*

   Funci'on que verifica los 'angulos opuestos y los intercambia en caso necesario.

   Argumentos:
   Ninguno.

   Regresa:
   true si todo bien y false si hubo alg'un error.

*/
bool VerificaOpuestos( void );
```

```
/*

   Funci'on que regresa el siguiente elemento de la cola de tri'angulos.

   Argumentos:
   ppArbolTrianguloD triangulo - El tri'angulo a procesarse.
   pInt vertice                - El valor del v'ertice a ser procesado.

   Regresa:
   true si s'i lo extrajo y false si no pudo.

   El elemento extra'ido es eliminado de la cola.

*/
bool ExtraeTrianguloListaD( ppArbolTrianguloD triangulo, pInt vertice );

/*

   Funci'on que revisa los 'angulos entre los punto1 y punto4 del nuevo tri'angulo formado.

   Argumentos:
   const pPunto3D punto1 - El primer punto y v'ertice del primer 'angulo.
   const pPunto3D punto2 - El segundo punto.
   const pPunto3D punto3 - El tercer punto.
   const pPunto3D punto4 - El cuarto punto y v'ertice del segundo 'angulo (opuesto al primero)

   Regresa:
   true si los 'angulos no son v'alidos, false si son v'alidos.

*/
bool AngulosNoValidos( const pPunto3D punto1, const pPunto3D punto2, const pPunto3D punto3,
                       const pPunto3D punto4 );

/*

   Funci'on que establece los opuestos de los tri'angulos creados y elimina los no terminales.

   Argumentos:
   ppTrianguloD triangulo - El arreglo de los tri'angulos finales.
   pInt cantidad          - La cantidad de tri'angulos en el arreglo.

   Regresa:
   true si todo bien y false si hubo alg'un error.

*/
bool ObtenTriangulosFinalesD( ppTrianguloD triangulo, pInt cantidad );

/*

   Funci'on que obtiene la lista de tri'angulos finales.

   Argumentos:
   ppTrianguloD triangulo - Para regresar el arreglo de tri'angulos finales.
   pInt cantidad          - La cantidad de tri'angulos en el arreglo

   Regresa:
```

```
   true si todo bien y false si hubo alg'un error.

*/
bool ObtenTrianguloTerminal( ppTrianguloD triangulo, pInt cantidad );

/*

   Funci'on que cuenta la cantidad de tri'angulos finales.

   Argumentos:
   Ninguno.

   Regresa:
   La cantidad de tri'angulos finales.

*/
int CuantosTriangulosTerminales( void );
```

E.3. Funciones de las curvas y re-triangulación

```
/*

   Funci'on que solicita la memoria para el arreglo de alturas.

   Argumentos:
   const pTrianguloD triangulo    - El arreglo de tri'angulos.
   int cantidad                   - La cantidad de alturas.
   ppAlturaCurvaTriangulo alturas - El arreglo de alturas.
   double dist                    - La distancia entre curva y curva.

   Regresa:
   true si todo bien y false si hubo alg0'un error.

*/
bool CreaArregloAlturas( const pTrianguloD triangulo, int cantidad, ppAlturaCurvaTriangulo alturas,
                         double dist );

/*

   Funci'on que obtiene la lista de curvas de nivel a partir de la triangulaci'on.

   Argumentos:
   const pTrianguloD triangulo   - El arreglo de tri'angulos.
   int cantidad                  - La cantidad de tri'angulos en el arreglo
   ppListaCurva2D curvas         - La lista de las curvas de nivel.
   ppListaTriangulo3D triangulos - La lista de tri'angulos resultado de la retriangulaci'on.
   double dist                   - La distancia entre curva y curva.

   Regresa:
   true si todo bien y false si hubo alg'un error.

*/
bool ObtenCurvasDeNivelTriangulacion( const pTrianguloD triangulo, int cantidad, ppListaCurva2D curvas,
                                      ppListaTriangulo3D triangulos, double dist );
```

```
/*

   Funci'on que retriangula un tri'angulo dadas las alturas por las que pasa la curva.

   Argumentos:
   const pTrianguloD triangulo        - El tri'angulo a retriangular.
   const pAlturaCurvaTriangulo altura - Las alturas de ese tri'angulo.
   ppListatriangulo3D triangulos      - La lista de los tri'angulos.

   Regresa:
   true si todo bien y false si hubo alg'un error.

*/
bool ReTriangula( const pTrianguloD triangulo, const pAlturaCurvaTriangulo altura,
                  ppListaTriangulo3D triangulos );

/*

   Funci'on que obtiene el orden de los v'ertices desde el menor al mayor.

   Argumentos:
   const pTrianguloD triangulo - El tri'angulo a determinar el orden.
   pInt orden                  - El orden de los v'ertices.

   Regresa:
   Nada.

*/
void ObtenOrdenTrianguloD( const pTrianguloD triangulo, pInt orden );

/*

   Funci'on que obtiene el punto que representa en esa arista la altura especificada.

   Argumentos:
   pDouble x                   - La coordenada en X del punto.
   pDouble y                   - La coordenada en Y del punto.
   const pTrianguloD triangulo - El tri'angulo que contiene el punto.
   int vertice                 - El v'ertice opuesto a la arista que contiene el punto.
   double altura               - La altura del punto.

   Regresa:
   Nada.

*/
void ObtenPuntoAltura3D( pDouble x, pDouble y, const pTrianguloD triangulo, int vertice,
                         double altura );

/*

   Funci'on que inserta un nuevo tri'angulo en la lista de tri'angulos.

   Argumentos:
   double x1, y1, z1 - Coordenadas del primer v'ertice del tri'angulo.
   double x2, y2, z2 - Coordenadas del segundo v'ertice del tri'angulo.
   double x3, y3, z3 - Coordenadas del tercer v'ertice del tri'angulo.
```

```
   ppListaTriangulo3D triangulos - La lista de tri'angulos.

   Regresa:
   true si todo bien y false si hubo alg'un error.

*/
bool InsertaTrianguloLista( double x1, double y1, double z1, double x2, double y2, double z2,
                            double x3, double y3, double z3, ppListaTriangulo3D triangulos );

/*

   Funci'on que crea una curva de nivel nueva y la inserta en la lista de curvas.

   Argumentos:
   pAlturaCurvaTriangulo alturas - Las alturas de cada tri'angulo.
   const ppTrianguloD triangulo  - El arreglo de tri'angulos.
   int indice                    - El tri'angulo desde el que se inicia la curva.
   double altura                 - La altura de la curva que se est'a creando.
   ppListaPunto2D curva          - La lista de las curvas.

   Regresa:
   true si todo bien y false si hubo alg'un error.

*/
bool CreaCurvaDeNivel( pAlturaCurvaTriangulo alturas, const pTrianguloD triangulo,
                       int indice, double altura, ppListaPunto2D curva );

/*

   Funci'on que indica si la altura se encuentra en la arista opuesta al v'ertice especificado.

   Argumentos:
   const pTrianguloD triangulo - El tri'angulo a verificar.
   int vertice                 - El v'ertice a considerar.
   double altura               - La altura a verificar.

   Regresa:
   TRUE si si se encuentra y FALSE si no.

*/
bool VerificaAlturaArista( const pTrianguloD triangulo, int vertice, double altura );

/*

   Funci'on que obtiene la curva hasta donde le es posible, a~nadiendo los puntos seg'un corresponda.

   Argumentos:
   pAlturaCurvaTriangulo alturas - Las alturas de cada tri'angulo.
   const ppTrianguloD triangulo  - El arreglo de tri'angulos.
   pInt indice                   - El tri'angulo desde el que se inicia la curva.
   int vertice                   - El v'ertice por el cual su arista opuesta entr'o la curva.
   double altura                 - La altura de la curva que se est'a creando.
   ppListaPunto2D cabeza         - La cabeza de la lista de las curvas.
   ppListaPunto2D cola           - La cola de la lista de las curvas.
   bool (*InsertaPunto)( const pPunto2D, ppListaPunto2D, ppListaPunto2D )
                                 - La funci'on de inserci'on del punto en la lista.
```

```
   Regresa:
   true si todo bien y false si hubo alg'un error.

*/
bool AnadePuntosCurva( pAlturaCurvaTriangulo alturas, const pTrianguloD triangulo, pInt indice,
                       int vertice, double altura, ppListaPunto2D cabeza, ppListaPunto2D cola,
                       bool (*InsertaPunto)( const pPunto2D, ppListaPunto2D, ppListaPunto2D ) );

/*

   Funci'on que inserta una curva en la lista de curvas.

   Argumentos:
   const pListaPunto2D curva - La curva a ser insertada.
   double altura             - La altura de la curva a insertar.

   Regresa:
   true si todo bien y false si hubo alg'un error.

   El valor apuntador por lista se modifica dentro de esta funci'on.

*/
bool InsertaCurvaLista2D( const pListaPunto2D curva, double altura );

/*

   Funci'on que crea un nodo para ser a~nadido a la lista de curvas.

   Argumentos:
   const pListaPunto2D curva - La curva a ser a~nadida a la lista.
   double altura             - La altura de la curva.
   ppListaCurva2D nodo       - El nodo a ser creado.

   Regresa:
   true si todo bien y false si hubo alg'un error.

   El valor nodo es modificado dentro de esta funci'on.

*/
bool CreaNodoCurva( const pListaPunto2D curva, double altura, ppListaCurva2D nodo );

/*

   Funci'on que inserta un nodo en la lista de curvas.

   Argumentos:
   pListaCurva2D nodo - El nodo a ser insertado.

   Regresa:
   Nada.

*/
void InsertaNodoCurvaLista( pListaCurva2D nodo );

/*

   Funci'on que regresa la lista de las curvas obtenidas.
```

```
   Argumentos:
   Ninguno.

   Regresa:
   Un apuntador a la lista de curvas.

*/
const pListaCurva2D ObtenCurvaLista2D( void );
```

E.4. Consideración de la curvatura de la Tierra

```
/*

   Programa para determinar la relaci'on entre el tama~no real y el visible del terreno.

*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

const double M_PI = 3.14159265358979323846; /* El valor de PI */
const double ERROR = 0.000001; /* El error m'aximo en la obtenci'on de alfa */

/*

   Funci'on que representa la funci'on a encontrarle la ra'iz.

*/
double func( double x, double b, double f )
{
   return sin( x ) - b * ( 1.0 + f - cos( x ) ); /* La regresa */
} /* func */

/*

   Funci'on que representa la derivada de la funci'on a encontrarle la ra'iz.

*/
double funcd( double x, double b )
{
   return cos( x ) + b * sin( x ); /* La regresa */
} /* funcd */

/*

   Funci'on que encuentra la ra'iz de una funci'on mediante el m'etodo de Newton.

*/
double newton( double x, double b, double f )
```

```
{
   double d; /* Temporal para el incremento */

   do /* Haz... */
   {
      d = func( x, b, f ) / funcd( x, b ); /* Obten el incremento */
      x -= d; /* Actualiza */
   } while ( fabs( d ) > ERROR ); /* ...mientras no la encuentres */
   return x; /* Regresa la ra'iz */
} /* newton */

/*

   Principal.

*/
int main( int argc, char *argv[] )
{
   double b, tanb; /* El 'angulo en grados y su tangente */
   double f, s; /* La altura a la que fu'e tomada la fotograf'ia y el tama~no de la im'agen */
   double a, r; /* El 'angulo alfa y la raz'on que se necesita */

   if ( argc < 4 ) /* Si no hay suficientes par'ametros */
   {
      printf( "USO: curvatura altura angulo tamano.\n" );
      printf( "     donde altura - La altura a la que fu'e tomada la fotograf'ia (en Km)\n" );
      printf( "           angulo - El 'angulo focal de la c'amara (en grados)\n" );
      printf( "           tamano - El tama~no de la imagen (en pixels)\n" ); /* Mensajes de uso */
      exit( -1 ); /* Termina */
   }
   f = atof( argv[ 1 ] ) / 6378.0; /* Obtiene la raz'on de la altura contra el radio de La Tierra */
   b = M_PI * atof( argv[ 2 ] ) / 360.0; /* Obtiene la mitad del 'angulo focal en radianes */
   s = atof( argv[ 3 ] ); /* El tama~no de la imagen */
   tanb = tan( b ); /* La tangente del semi 'angulo focal */
   a = newton( b, tanb, f ); /* Obtiene al 'angulo alfa */
   r = a / ( f * tanb ); /* Obtiene la raz'on */
   printf( "               'Angulo al centro (grados): %f\n", a * 180.0 / M_PI );
   printf( "    Tama~no de la fotograf'ia real (en Km): %f\n", 6378.0 * f * tanb );
   printf( "       Raz'on de estiramiento de la imagen: %f\n", r );
   printf( "Tama~no real de la fotograf'ia (en pixels): %f\n", s * r ); /* Resultados */
   return 0; /* OK */
} /* main */
```

Bibliografía

[1] Lillesand, Thomas M and Kiefer, Ralph W. *Remote sensing and image interpretation*, 3rd edition, Chapter 3, John Wiley & Sons, 1994.

[2] J. P. Lewis, *Fast Template Matching*, Vision Interface, pp 120-123, 1995.

[3] J. P. Lewis, *Fast Normalized Cross-Correlation*, Industrial Light & Magic. http://www.idiom.com/∼zilla/Work/nvisionInterface/nip.pdf

[4] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, *Computational Geometry, Algorithms and Applications, Second Edition*, Springer-Verlag, pp 183-210, 2000.

[5] David Sweet, et al, *Desarrollo de aplicaciones con KDE 2.2*, SAMS, 2001.

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms, Second Edition*, MIT Press, 2001.

[7] Mason Woo, Jackie Neider, Tom Davis, Dave Shreiner, *OpenGL Programming Guide, Third Edition*, Addison Wesley, 1999.

[8] Gabriel Valiente Feruglio, *Composición de Textos Científicos con LaTeX*, Alfaomega, 2001.

[9] Herbert Schildt, *C/C++ Programmer's Reference, Second Edition*, Osborne, McGraw Hill, 2000.

[10] Kurt Wall, et al. *Programación en Linux al Descubierto, 2a edición*, Prentice Hall, 2001.

[11] Min-Bin Chen, Tyng-Ruey Chuang, Jan-Jan Wu, *A Parallel Divide-and-Conquer Scheme for Delaunay Triangulation*, Proceedings of the Ninth

International Conference on Parallel and Distributed Systems, IEEE, pp 571-576, 2002.

[12] Jianbo Ma and Narendra Ahuja, *Region Correspondence by Global Configuration Matching and Progressive Delaunay Triangulation*, Computer Vision and Pattern Recognition (CVPR'00)-Volume 2, IEEE, pp 2637-2643, 2000.

[13] Estándar *OpenGL* para la visualización de objetos tridimensionales: http://www.opengl.org

[14] Luis Gerardo de la Fraga, *Diseño de una Interfaz Gráfica para el Problema del Caballo de Ajedrez*, Congreso Nacional de Ingeniería Electrónica del Golfo. 13-17 noviembre del 2000. Instituto Tecnológico de Orizaba. pp. 1-6.

[15] Biblioteca de funciones de *OpenGL* libre para Linux: http://www.mesa3d.org

[16] Biblioteca de objetos para la creación de interfaces gráficas, tanto para Linux como para Windows: http://www.trolltech.com/products/qt/index.html

[17] Luis Gerardo de la Fraga, Feliú Sagols Troncoso, *Scubes: A Program to Visualize Vox-Solids*, VII Conferencia de Ingeniería Eléctrica, 5, 6 y 7 de septiembre del 2001. CINVESTAV.

[18] Escritorio libre de Linux basado en *Qt*: http://www.kde.org

[19] Programa de dibujo vectorial: http://www.xfig.org

[20] Sistema de tipografía LaTeX: http://www.ctan.org

[21] R. Marabini, I. M. Mesagosa and San Martin, M. en C. and J. J. Fernández, S. Marco, L. G. de la Fraga, C. Vaquerizo and J. M. Carazo, *Xmipp: An image processing package for electron microscopy*, Journal of Structural Biology volume 116, pp 237-240, 1996. http://biocomp.cnb.uam.es

[22] R. Stallman, *Linux and the GNU Project*, http://www.gnu.org/gnu/linuxandgnu.html

[23] Distribución muy difundida de GNU/Linux, http://www.redhat.com

[24] F. P. Preparate and M.I. Shamos, *Computational geometry: an introduction*, Springer-Verlag, 1985.

[25] Sistema para la edición de dibujos vectoriales en el plano y tercera dimensión muy utilizado, http://www.autodesk.com/autocad

[26] Sistema de CAD de libre distribución que utiliza de forma nativa el formato DXF de AutoCAD, http://www.ribbonsoft.com/qcad.html

[27] Cómo generar un archivo DXF mínimo y que se pueda leer desde AutoCAD, http://www.autodesk.com/techpubs/autocad/acad2000/ /dxf/writing_a_dxf_file_dxf_aa.htm

[28] Referencia del formato DXF, http://usa.autodesk.com/adsk/servlet/ /item?siteID=123112&id=752569

[29] C. B. Barber and D. P. Dobkin and H. Huhdanpaa, *The quickhull algorithm for convex hull*, ACM Transactions on Mathematical Software, Volume 22, Number 4, pp. 469-483, December 1996.