Universidad de la República
Facultad de Ingeniería
Instituto de Computación

Informe de Proyecto de Grado

Supervisor: Andrés Almansa
Co-supervisor: Javier Preciozzi

Cálculo de mapas de elevación de zonas urbanas

Matías Rodríguez

27 de julio de 2007

Resumen

El tema de este proyecto de grado es el cálculo de mapas de elevación de alta precisión de zonas urbanas, a partir de dos imágenes aéreas o satelitales tomadas de distintos puntos de vista.
El principio estereoscópico permite calcular las alturas de los objetos de una escena (usando dos imágenes de distinta perspectiva), debido a que relaciona la profundidad de los mismos con su desplazamiento (disparidad) relativo a los dos puntos de vistas. La precisión en las alturas estimadas depende directamente de la precisión en las disparidades calculadas y del cociente B/H (baseline[1]/ altura de las cámaras). Los algoritmos estereoscópicos clásicos calculan disparidades enteras, pero usando una configuración de cámaras con un baseline grande permiten obtener buenas estimaciones. Aunque esto tiene el inconveniente de que el aumento de la distancia entre las cámaras incrementa zonas que son vistas solo en una imagen y no en ambas (oclusiones). En estas zonas no se puede estimar las alturas debido a que no existen correspondencias entre el par de imágenes. Lo anterior se ve agravado en las imágenes aéreas o satelitales de zonas urbanas a causa de que las construcciones (edificios, casas, etc.) más altas taparían a las más bajas.
Recientemente, se ha empezado a diseñar nuevos algoritmos para tratar pares estéreo con un baseline pequeño. De esta forma, se pueden obtener mapas de elevación densos (evitando las oclusiones), pero teniendo que innovar en técnicas para obtener disparidades subpixelares, amén de compensar la perdida de precisión por la configuración de cámaras usada (un H/B grande).

La primera parte del trabajo realizado consiste en el desarrollo de una aplicación para trabajar con algoritmos estereoscópicos en el cálculo de mapas de elevación.
La misma está enfocada en el uso de pares estéreo con un baseline pequeño y permite calcular mapas de disparidad subpixel con los algoritmos presentados e implementados en [3]. Otra característica importante es que genera a partir de un par estéreo y un mapa de disparidad, nuevos pares estéreo que simulan una variación en el cociente B/H (baseline / altura). Esto tiene como propósito poder comparar algoritmos estereoscópicos usando distintas configuraciones de cámaras para la misma escena.

Como segunda parte se continua con el desarrollo y evaluación de un nuevo algoritmo estereoscópico iniciado como tarea final del curso "Teoría Computacional de la Gestalt".
El algoritmo propuesto calcula disparidades subpixelares basándose en la búsqueda de correspondencias entre las líneas de nivel de las imágenes.

[1]Distancia entre las cámaras

Índice general

Capítulo 1

Introducción

1.1. Motivación

La motivación de este proyecto de grado es poder generar una representación tridimensional de una zona urbana a partir de un par de imágenes aéreas o satelitales tomadas de distintos puntos de vista.
A pesar de que existen una gran variedad de aplicaciones para visualizar ciudades a través de imágenes satelitales (ej. Google Earth, Montevi Map, etc.), por lo general éstas no presentan información tridimensional o si lo hacen es solo parcialmente para algunas zonas calculando manualmente las alturas de los edificios. A medida que se diseñan nuevos y mejores algoritmos para calcular mapas de elevación de forma automática, se hace mucho más interesante la posibilidad de integrar estos algoritmos con las aplicaciones que visualizan ciudades usando imágenes satelitales.
La forma en que se puede obtener información 3D a partir de dos imágenes bidimensionales es a través del principio estereoscópico, que relaciona el desplazamiento relativo (llamado disparidad) de los objetos entre cada punto de vista y la profundidad de los mismos. Más precisamente, si se tiene dos cámaras en paralelo a una distancia B (baseline), una altura H con respecto al suelo y el tamaño de un pixel de la imagen proyectado en el suelo es r_0, la relación entra la altura h de un punto de la escena 3D y su disparidad d es:

$$d = \frac{h}{H} B r_0 \tag{1.1}$$

Por este motivo, uno de los pasos más importantes en el proceso del cálculo de mapas de elevación, es la búsqueda de correspondencias entre pares de imágenes en estéreo. Los algoritmos que resuelven esto son conocidos como algoritmos estereoscópicos, que no hacen otra cosa que encontrar los pares de pixels que representan o proyectan el mismo punto del espacio 3D. Cuando se usan imágenes aéreas o satelitales, el problema tiene una agravante adicional, a causa de que el error en la altura estimada (e_{alt}) depende del error en la disparidad (e_{match}) mediante la siguiente ecuación: $e_{alt} = \frac{H}{B}.r_0 e_{match}$. Como se ve, el error en la altura estimada es directamente

proporcional a $\frac{H}{B}$. Es por esto que los algoritmos estereoscópicos clásicos, usan una configuración de cámaras con un B bastante grande. Pero en las imágenes aéreas o satelitales la altura H es inherentemente grande, y aumentar B para contrarrestar el efecto negativo causado en la precisión por H no es muy viable, debido a que esto incrementa considerablemente las zonas ocluidas. Otra alternativa es calcular disparidades muy precisas con una resolución menor a la de un pixel, pero esto no es contemplado por los algoritmos estereoscópicos clásicos que por lo general se limitan a calcular disparidades enteras. En cambio, recientemente se ha empezado a diseñar nuevos algoritmos (como los presentados en [3]) que innovan en técnicas para calcular disparidades subpixelares en imágenes de zonas urbanas con un baseline pequeño.

1.2. Objetivos

El objetivo principal es desarrollar una herramienta para trabajar con algoritmos estereoscópicos en el cálculo de mapas de elevación de zonas urbanas. Se espera construir una aplicación gráfica que entre otras cosas permita generar a partir de un par estéreo real y un mapa de disparidad, nuevos pares que simulen cambios en la altura de las cámaras. De esta forma, el usuario ingresando una nueva altura puede ver el par estéreo como si se hubiera sacado en esa posición. La importancia de crear nuevos pares que simulen una variación en el valor B/H es que permite comparar algoritmos usando distintas configuraciones de cámaras para la misma escena. Además de lo anterior la aplicación debe brindar las siguientes funcionalidades:

- Calcular mapas de elevación usando los algoritmos presentados e implementados en [3].
- Estimar errores de los mapas de disparidad usando las máscaras devueltas por los algoritmos o las agregadas manualmente por el usuario.
- Integrar dinámicamente nuevos algoritmos para trabajar con imágenes estereoscópicas.
- Generar anaglifos[1].
- Visualizar las distintas imágenes cargadas o calculadas (par estéreo, mapas de disparidad, ground-truth, pares simulados, anaglifos).
- Ver escenarios 3D usando los mapas de disparidad calculados.

Como objetivo secundario se continua con el desarrollo y evaluación de un nuevo algoritmo estereoscópico iniciado como tarea final del curso "Teoría Computacional de la Gestalt 2005". En esta segunda etapa se espera poder concluir si el algoritmo propuesto es válido o no. Para hacer la validación se compara los resultados obtenidos de una implementación del mismo con otro algoritmo estereoscópico.

[1]Un anaglifo es una superposición de dos imágenes que produce al ser miradas con lentes especiales una sensación de relieve.

1.3. Organización del Documento

El documento se organiza de la siguiente manera. En el Capítulo 2 se hace un resumen del estado del arte del cálculo de mapas de elevación de zonas urbanas. En éste se resumen los conceptos básicos de la visión en estéreo, los algoritmos computacionales clásicos que abordan el problema de la búsqueda de correspondencias, y los algoritmos de alta precisión usados para imágenes satelitales presentados en [3].
El Capítulo 3 trata sobre el trabajo realizado en la primera etapa del proyecto de grado, en donde se desarrolla una aplicación gráfica para el tratamiento de imágenes estereoscópicas.
En el Capítulo 4 se presenta la segunda parte que consiste en la continuación y evaluación de un nuevo algoritmo estereoscópico iniciado como tarea final en la materia Teoría Computacional de la Gestalt.
Finalmente en el Capítulo 5 se hacen las conclusiones sobre el trabajo realizado en el proyecto de grado y se plantean posibles extensiones para hacer en futuro.

Capítulo 2

Estado del Arte

El objetivo de la computación en estéreo es obtener una representación tridimensional de una escena a partir de dos o más imágenes tomadas desde puntos de vistas diferentes. En este capítulo se hace una revisión del estado del arte de la misma, enfocada principalmente al cálculo de mapas de elevación de alta precisión de zonas urbanas, usando imágenes aéreas o satelitales. Este capítulo se divide en tres secciones: Visión en Estéreo, Cálculo de Correspondencias y Métodos Sub-Pixelares. La primera trata sobre los conceptos básicos de la visión en estéreo, que entre otras cosas son necesarios para poder comprender el problema de la computación en estéreo en general, luego en la segunda sección se describen los algoritmos computacionales clásicos que abordan el problema de la búsqueda de correspondencias, y por último se describen los algoritmos de alta precisión usados para imágenes satelitales presentados en [3].

2.1. Visión en Estéreo

Cuando se observa un objeto tridimensional desde dos puntos de vistas distintos, se puede ver que el objeto tiene un desplazamiento relativo entre cada punto de vista, por más que el objeto no se mueva realmente. Por ejemplo si se pone el dedo índice delante de la cara y se lo observa con cada ojo por separado, se puede ver como el dedo se desplaza. Este desplazamiento al que se le llama disparidad, depende directamente de la distancia entre el observador y el objeto. Es por eso que los objetos más cercanos al observador tienen un mayor desplazamiento (como el dedo en el ejemplo anterior) y los más lejanos uno menor. Esta relación desplazamiento-profundidad, permite obtener mapas de profundidades a partir de las disparidades calculadas por los algoritmos estereoscópicos. En esta sección se describen los conceptos básicos para entender cómo la visión en estéreo se relaciona con la obtención de profundidades de una escena.

2.1.1. Cámara Pinhole

El modelo geométrico de la cámara es usado para obtener una imagen 2D a partir de una escena 3D. Como se ve en la figura 2.1, la cámara está formada por el plano imagen I en donde se proyecta la escena, el centro de proyección C, y la distancia focal f que es la distancia entre el plano imagen y el centro de proyección. Un punto M cualquiera de la escena 3D, se proyecta en la imagen bidimensional por la intersección entre el plano I y la recta que pasa por M y el centro de proyección C.

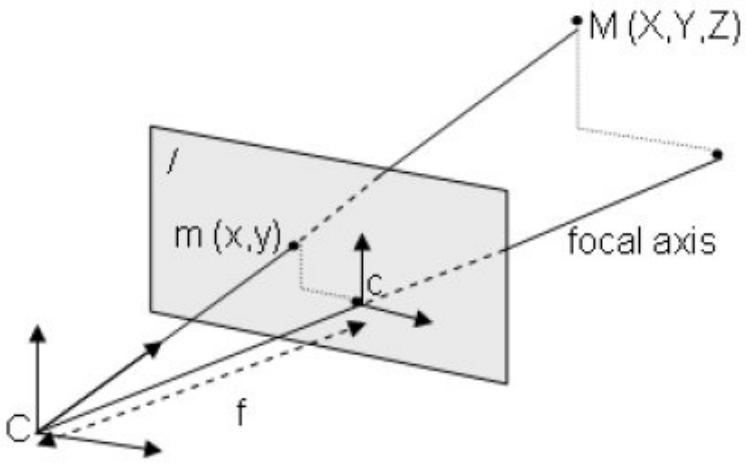


Figura 2.1: Modelo Pinhole

2.1.2. Geometría Epipolar

La geometría epipolar explica cómo se relacionan dos o más cámaras pinhole. Como se ve en la figura 2.2 donde C y $C^{'}$ son los centros de proyección, y I e $I^{'}$ los planos imagen de cada cámara. Dado M un punto cualquiera en el espacio, los puntos C , $C^{'}$ y M definen un plano π llamado plano epipolar. Este plano epipolar intersecta los planos de las cámaras I e $I^{'}$ en las líneas lm y $lm^{'}$ respectivamente. Si m es la proyección de M en I, entonces su correspondiente $m^{'}$ (que es la proyección de M en $I^{'}$) tiene que estar sobre la línea $lm^{'}$ (línea epipolar de $m^{'}$). Variando la posición del punto M, se puede observar que todas las líneas epipolares de $I^{'}$ (análogo para I) tienen un punto $e^{'}$ (e) en común. Estos puntos son la intersección de la línea C y $C^{'}$ con cada plano imagen. Como se cumple que si un punto de la imagen I y un punto de la imagen $I^{'}$ corresponden al mismo punto real M, entonces m, $m^{'}$, C y $C^{'}$ están sobre el mismo plano (restricción co-planar [14]), se tiene una restricción sobre el proceso de macheo, debido a que la correspondencia de un punto de una imagen tiene que estar sobre la línea epipolar en la otra imagen. Esto reduce la búsqueda de correspondencias a una dimensión y es llamado restricción epipolar.

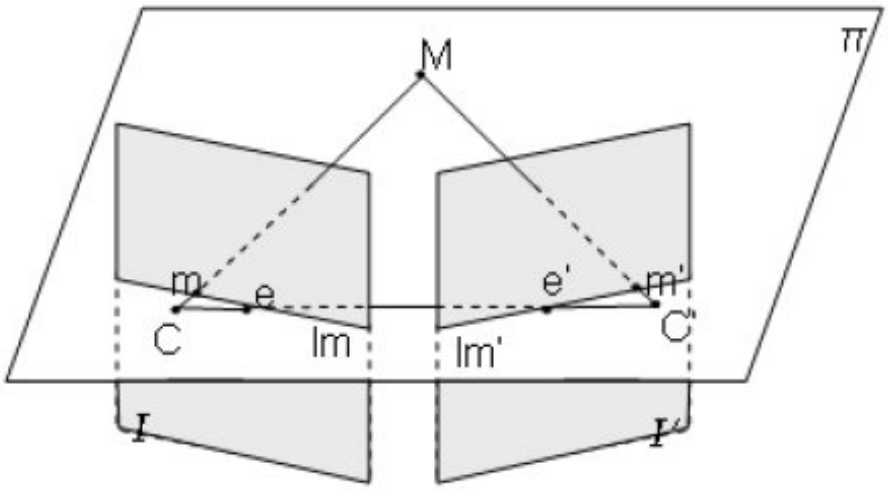


Figura 2.2: Geometría Epipolar

2.1.3. Principio Estereoscópico

Cuando se tiene dos cámaras que tienen la misma distancia focal y los planos imagen y centros de proyección están desplazados horizontalmente (a esto se le llama una configuración en paralelo), se puede obtener una relación entre las disparidades y las profundidades que es llamado Principio Estereoscópico. Aunque las imágenes originales no tengan una configuración en paralelo, se las

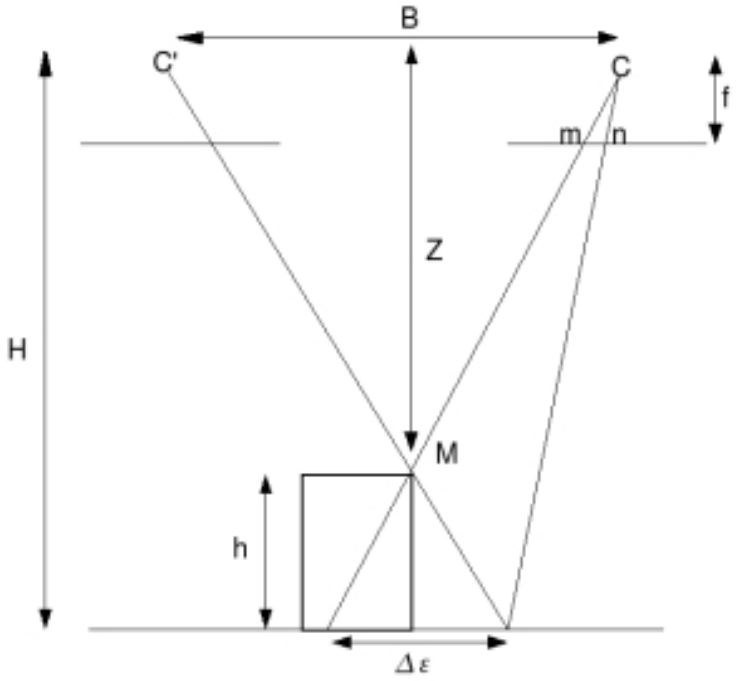


Figura 2.3: Configuración en Paralelo

puede llevar a dicha configuración mediante una rectificación, que es una operación que alinea horizontalmente las líneas epipolares presentadas en la sección anterior. La figura 2.3 muestra un ejemplo de dos cámaras configuradas en paralelo, donde B es la distancia entre los dos centros de proyección conocido como baseline, H es la altura de las cámaras con respecto al suelo, M es un punto observado de la escena, h es la altura del punto observado, y $\Delta\epsilon$ es la disparidad calculada de la proyección del punto M sobre el plano del suelo usando los centros de proyección C y C'.
Si Z es la distancia entre las cámaras y el punto M ($Z = H - h$), usando semejanzas de triángulos se puede llegar a la siguiente relación:

$$\frac{B}{Z} = \frac{\Delta\epsilon}{h} \Rightarrow h = \frac{Z}{B}\Delta\epsilon \Rightarrow h = \frac{H-h}{B}\Delta\epsilon \tag{2.1}$$

Si d es la disparidad $\Delta\epsilon$ proyectada sobre el plano imagen de la cámara $d = x_m - x_n$, usando también semejanzas de triángulos se obtiene la siguiente relación:

$$\frac{f}{d} = \frac{H}{\Delta\epsilon} \Rightarrow \Delta\epsilon = \frac{H}{f}d \tag{2.2}$$

haciendo el cambio de variable $r_0 = H/f$ (conceptualmente r_0 es el tamaño de un pixel de la imagen, proyectado en el suelo) y sustituyendo la ecuación 2.2 en 2.1 se tiene una relación directa entre la disparidad de un punto en la imagen y su altura,

$$d = \frac{h}{H-h} Br_0 \tag{2.3}$$

Dentro del contexto de imágenes aéreas o satelitales de zonas urbanas, la altura H de las cámaras es bastante mayor que la de los objetos de la escena (edificios, casas, etc.). Esto permite aproximar la función $f(h) = \frac{h}{H-h}$, usando el polinomio de Taylor de primer orden ($f(x) = f(0) + f'(0)x + O(|x^2|)$) de la siguiente manera.

$$\Rightarrow f(h) = 0 + \left(\frac{H}{(H-h)^2} | h = 0 \right) h + O(|h^2|) \Rightarrow \frac{h}{H-h} = \frac{h}{H} + O(|h^2|) \tag{2.4}$$

Usando 2.4 se puede aproximar 2.3, como

$$d = \frac{h}{H} Br_0 \tag{2.5}$$

2.1.4. Oclusiones

A causa de la escena y la geometría de las cámaras, cuando se ve una escena desde dos puntos de vistas distintos, hay puntos que son visibles en una cámara pero no en ambas. A estos puntos se le llama puntos de oclusión, en la figura 2.4 se muestra un par de ejemplos de como se puede dar esta situación, donde el punto P_o es visto por el observador O_L pero no por el observador O_R.
El problema fundamental de las oclusiones, es que no se puede calcular las correspondencias en estos puntos debido a que éstas no existen entre el par de imágenes. Esto conlleva a que no sea posible estimar sus alturas con la ecuación 2.5.

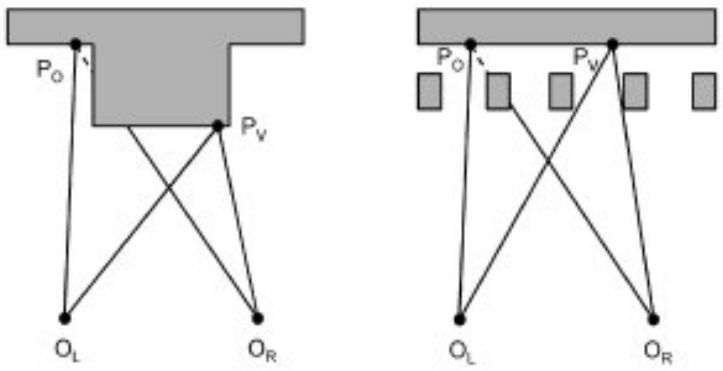


Figura 2.4: Oclusiones

2.1.5. Relación entre la precisión de la disparidad y la altura estimada

La precisión de la altura estimada con la ecuación 2.5, va a depender de qué tan precisa sea la disparidad calculada. Observando la figura 2.5 y haciendo un razonamiento análogo al realizado en la sección 2.1.3, se obtiene la siguiente relación entre la precisión de la disparidad e_{mach} y la de la altura estimada e_{alt}.

$$e_{alt} = \frac{H}{B}.r_0 e_{match} \tag{2.6}$$

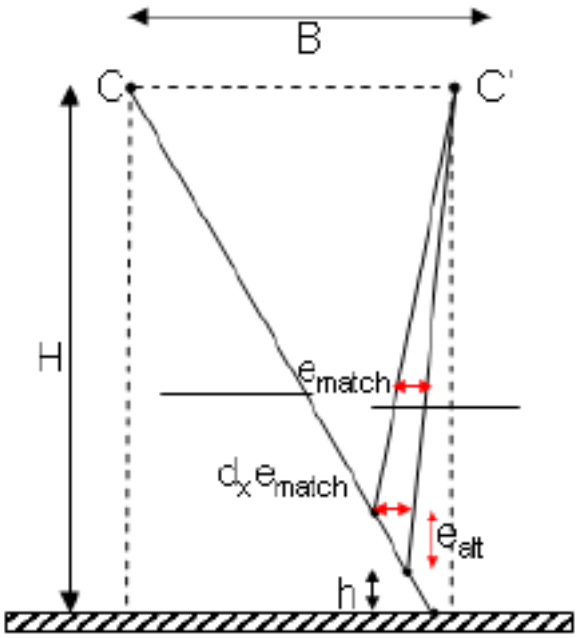


Figura 2.5: Relación entre la precisión de la disparidad y la altura estimada

De la ecuación 2.6 se deduce que al usar un $\frac{B}{H}$ grande, se reduce el error de las alturas estimadas con respecto al de las disparidades calculadas. Es por eso que los algoritmos clásicos de cálculo de disparidades son pensados para usar con un B (distancia entre cámaras) bastante grande. De esta forma, disminuye el error estimado, pero teniendo el inconveniente de que al alejar las cámaras, se incrementa la cantidad de puntos ocluidos como se puede ver en la figura 2.6, provocando de esta forma un aumento de las zonas no detectadas por los algoritmos.

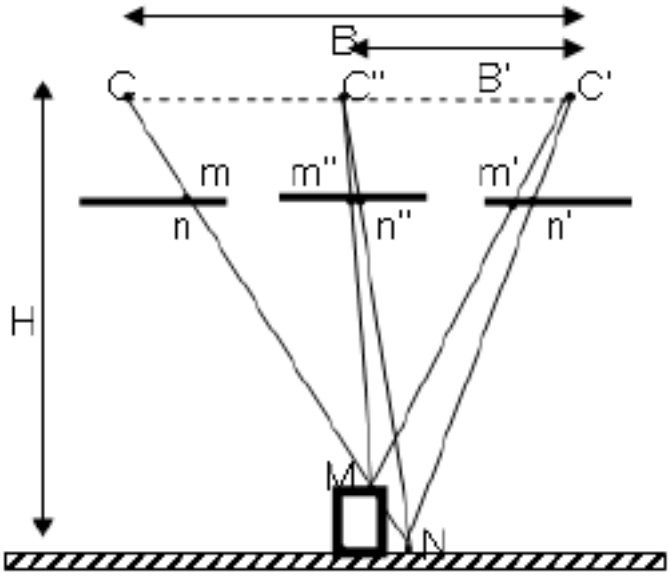


Figura 2.6: Oclusiones

2.2. Cálculo de las correspondencias

El cálculo de las correspondencias, consiste en encontrar para cada punto de la imagen de referencia, el punto que le corresponde en la imagen secundaria. En otras palabras, es encontrar los pares de puntos que representan o proyectan el mismo punto del espacio 3D.
Los algoritmos estereoscópicos calculan estas correspondencias y generan imágenes bidimensionales, conocidas como mapas de disparidad, siendo los valores de sus pixeles las disparidades calculadas del par estéreo. Luego, con estos mapas y usando el principio estereoscópico, se puede obtener una representación tridimensional de la escena. La figura 2.7 muestra un ejemplo de un mapa de disparidad, donde las zonas más claras indican disparidades mayores (por lo tanto más cercanas) y las más oscuras menores. Los mapas pueden ser densos o dispersos, la diferencia entre ambos, es que los primeros tienen calculadas las disparidades para todos sus pixeles, y los segundos solo para algunos puntos, siendo necesario interpolarlos para obtener mapas densos.
Calcular las correspondencias, es un problema abierto y no existe una única solución general que lo resuelva mejor, debido a que hay una gran ambigüedad en los macheos provocada por las oclusiones, uniformidad en las texturas, etc.. Por otro lado hay una serie de restricciones que facilitan la búsqueda de correspondencias (ej. la restricción epipolar que limita la búsqueda de una correspondencia en una dimensión) y se puede hacer varias suposiciones que hacen que el problema sea tratable (ej. constancia en el brillo de las imágenes). Esto hace, que existan varias técnicas que aborden el tema de forma distinta, y difieran tanto en precisión como en perfomance, como se puede ver en [11], donde se hace una comparación de perfomance y precisión de las principales técnicas usadas.
En [10] se clasifican los algoritmos de correspondencia en los siguientes dos grupos, Métodos

(a) Referencia (b) Secundaria (c) Mapa de Disparidad

Figura 2.7: Ejemplo de un par estéreo con su mapa de disparidad

Locales y Métodos Globales. Los Locales, pueden ser muy eficientes, pero son muy sensibles a las ambigüedades locales (como son las regiones de oclusión y regiones con textura uniforme), mientras que los Métodos Globales obtienen una mayor precisión pero a expensa de un costo computacional mayor.

2.2.1. Métodos Locales

Se los denomina métodos locales, porque para calcular la disparidad de un punto, imponen restricciones locales sobre un grupo de puntos que lo rodea (ventana), y no usan restricciones globales que abarquen a toda la imagen. Las restricciones más usadas por estos métodos son la restricción epipolar y la de semejanzas. La primera, implica que el correspondiente de un punto debe estar en la recta epipolar del punto en la otra imagen (por lo que restringe la búsqueda del correspondiente sobre la recta epipolar). Y la segunda, supone que las características (ej. color) de los puntos de una imagen, varían muy poco con respecto a las de sus correspondientes de la otra imagen.
Continuando con la categorización taxonómica que se hace en [10], se dividen los Métodos Locales en tres grupos: Block Matching, Optical Flow y Feature Matching.

2.2.1.1. Block Matching

El método Block Matching, para calcular la disparidad de un punto de la imagen de referencia compara una pequeña región alrededor del punto (ventana), con otras ventanas del mismo tamaño extraídas de la otra imagen (imagen secundaria). La restricción epipolar, restringe la búsqueda de las regiones en una dimensión y las métricas más usadas para comparar ventanas son: correlación normalizada NCC,

$$\frac{\Sigma_{u,v}\left(I_1(u,v)-\bar{I}_1\right).\left(I_2(u,v)-\bar{I}_2\right)}{\sqrt{\Sigma_{u,v}\left(I_1(u,v)-\bar{I}_1\right)^2.\left(I_2(u,v)-\bar{I}_2\right)^2}} \tag{2.7}$$

suma de las diferencias al cuadrado SSD,

$$\Sigma_{u,v} \left(I_1(u,v) - I_2(u+d,v)\right)^2 \tag{2.8}$$

SSD Normalizado,

$$\Sigma_{u,v} \left(\frac{\left(I_1(u,v) - \bar{I}_1\right)}{\sqrt{\Sigma_{u,v}\left(I_1(u,v) - \bar{I}_1\right)^2}} - \frac{\left(I_2(u,v) - \bar{I}_2\right)}{\sqrt{\Sigma_{u,v}\left(I_2(u+d,v) - \bar{I}_2\right)^2}} \right)^2 \tag{2.9}$$

y suma de los valores absolutos de la diferencia SAD

$$\Sigma_{u,v} \left|I_1(u,v) - I_2(u+d,v)\right| \tag{2.10}$$

2.2.1.2. Optical Flow

Los métodos basados en Optical Flow, asumen que el brillo de una escena es constante entre las dos imágenes, esto hace que se pueda calcular disparidades locales entre las dos imágenes, formulando una ecuación diferencial que relacione el movimiento con el brillo de las dos imágenes. Como la traslación de un punto de una imagen con respecto a la otra es horizontal, se puede representar la relación entre el brillo y el movimiento, como se muestra en [10] mediante la siguiente ecuación diferencial:

$$(\nabla_x E)\vec{v} + E_t = 0, \tag{2.11}$$

Siendo $\nabla_x E$ la componente horizontal del gradiente, E_t representa la diferencia entre el brillo de la imagen de referencia y secundaria, y $\vec{v}$ es la traslación entre las dos imágenes. Si se asume, que la disparidad alrededor de una ventana pequeña varía poco, se podría calcular la disparidad de un punto, aplicando mínimos cuadrados a la ecuación diferencial de la siguiente forma

$$v = A^t A^{-1} A^t b \tag{2.12}$$

donde,

$$A = \begin{vmatrix} \nabla_x E(p_1) \\ \nabla_x E(p_2) \\ \vdots \\ \nabla_x E(p_n) \end{vmatrix}, \quad b = \begin{vmatrix} E(p_1) \\ E(p_2) \\ \vdots \\ E(p_n) \end{vmatrix} \quad y\ p_1, p_2, \ldots, p_n \text{ son los pixeles de la ventana} \tag{2.13}$$

2.2.1.3. Feature Matching

Los métodos Block Matching y Optical Flow tienen la desventaja que son muy sensibles a las discontinuidades, ya que las regiones que las contienen, tienen puntos con más de una profundidad. Además de que también son sensibles a las regiones que tienen textura uniforme. Los

métodos basados en las características, buscan evitar este problema restringiendo la región a las características de los objetos presentes en las imágenes, que pueden ser bordes, curvas, esquinas, etc.. El principal defecto de esta técnica, es que los mapas de disparidad calculados son dispersos, haciendo que sea necesario usar alguna técnica de interpolación para obtener mapas densos.

2.2.2. Métodos Globales

Los métodos globales además de las restricciones locales, usan restricciones globales para reducir la sensibilidad de las regiones locales que fallan al machear debido a las oclusiones, uniformidad de las texturas, etc.. Con la desventaja de aumentar el costo computacional al usar este tipo de restricciones, los dos métodos globales más importantes son la Programación Dinámica y el Graph Cuts.

2.2.2.1. Programación Dinámica

La Programación Dinámica es un método general, que reduce la complejidad computacional de un problema de optimización. Esto lo hace, descomponiendo el problema original en subproblemas donde el costo global se define como la minimización de los costos de los subproblemas. Los algoritmos estereoscópicos que se basan en esta técnica [12, 11], usan la restricción epipolar y la restricción de orden, para descomponer el problema original, y mediante correlación obtienen el costo de los subproblemas.
A pesar de ser un método global, tiene la desventaja que procesa las filas de la imagen por separado, es decir la minimización del costo global se hace independientemente para cada fila sin tomar en cuenta la globalidad total de la imagen. Haciendo que un error local se pueda propagar a lo largo de la fila, eliminado otros buenos macheos y generando líneas horizontales que suelen tener los mapas calculados por estos algoritmos (como se ve en la figura 2.8(b)).

2.2.2.2. Graph Cuts

El problema de los algoritmos estereoscópicos que usan programación dinámica, es que el costo de cada línea lo procesan de forma independiente, por lo que no toman en cuenta las restricciones de continuidad horizontal y vertical a la vez, ya que solo usan la horizontal. En cambio los que se basan en la técnica Graph Cuts [13, 11], minimizan el costo de una función que toma en cuenta todo el espacio al mismo tiempo. Buscando el máximo flujo de un grafo, que se construye de manera que minimice una función de energía como la siguiente.

$$E(L) = E_{data}(L) + \lambda E_{smooth}(L) \tag{2.14}$$

Donde $\lambda > 0$ determina el nivel de suavidad que se quiere obtener en el mapa de disparidad calculado.

(a) Ground-Truth (Distancias Verdaderas)

(b) Usando Programación Dinámica

(c) Usando Graph Cuts

Figura 2.8: Ejemplos de Mapas de Disparidad Calculados usando Programación Dinámica y Graph Cuts

2.3. Métodos Sub-pixelares

Por lo general los métodos descriptos anteriormente, son usados para calcular mapas de disparidad entera, éstos pueden ser una buena solución cuando se está trabajando con una configuración de cámaras que tiene un B/H grande, pero cuando se usa un B chico con respecto a H, el error relativo de las alturas estimadas es muy grande debido a la relación $e_{alt} = \frac{H}{B}.r_0 e_{match}$. Al trabajar con imágenes satelitales, se tiene un valor muy grande de H, que es inherente al problema. Esto dificulta considerablemente la obtención de estimaciones precisas, y aumentar B no es una buena solución, debido que al alejar las cámaras se incorporan más y nuevos puntos de oclusión por lo que se estaría empeorando el problema inicial. Especialmente en imágenes urbanas donde los edificios más altos ocluirían a los más bajos
Otra alternativa, es obtener un mapa de disparidad que tenga una resolución menor a la de un pixel, pero para eso, se tendrían que usar algoritmos que implementen técnicas sub-pixelares que los algoritmos clásicos comentados anteriormente y los que son presentados en [11] no abarcan. A continuación, se describen algunos algoritmos [3, 2, 4] que innovan en técnicas para calcular correspondencias sub-pixelares, en imágenes de zonas urbanas con un baseline pequeño.

2.3.1. MARC

MARC [5, 6] es un algoritmo multiresolución para correlación redefinida, codificado por Nathalie Camlong y Vincent Muron y que forma parte de la patente 0403143 de CNES.
La estrategia de multiresolución consiste en compensar una imagen con la disparidad previamente calculada en otra escala menor. Este proceso primero obtiene con las imágenes referencia (u_0) y secundaria ($\tilde{u}_0$) de escala menor un mapa de disparidad ϵ_0. Luego cuando se pasa a la

siguiente escala mayor, la imagen $\tilde{u}_0$ es corregida con una versión de la interpolada de ϵ_0

$$\tilde{u}_1(x) = \tilde{u}_1(x + \epsilon_0) \tag{2.15}$$

Haciendo que el par de imágenes u_1 y $\tilde{u}_1$, tenga disparidades subpixelares entre el rango que corresponde a la diferencia de la disparidad real ϵ y el resultado anterior. Con lo que se puede aplicar correlación otra vez y así sucesivamente hasta llegar al par de imágenes de la escala original. Esto permite obtener, un mapa de disparidad con una resolución menor a la del pixel. La desventaja del método es que calcula mapas dispersos, por lo que es necesario aplicarles técnicas de interpolación para obtener mapas densos.

2.3.2. Variational approach

Es un algoritmo [3] de interpolación, que genera mapas de disparidad densos a partir de mapas dispersos (ej. los mapas que son calculados con MARC). Este método consiste en regularizar los valores de las disparidades existentes con la interpolación de los faltantes. Esto lo hace, obteniendo un mapa de disparidad que por un lado trate de obtener la mínima cantidad de superficies S, pero que también intente ajustarse lo mejor posible al mapa de disparidad disperso de entrada D. Básicamente, el método minimiza una función de energía como la siguiente, que toma en cuenta la cantidad de superficies S y el ajuste de los datos D de entrada.

$$\min_{\epsilon} S(\epsilon) + \lambda D(\epsilon) \tag{2.16}$$

con,

$$S(\epsilon) = \sum \sqrt{\beta^2 + \left| \nabla\epsilon - c \left\langle \nabla\epsilon, z \right\rangle \frac{z}{|z|} \right|^2} \qquad y \qquad D(\epsilon) = \sum \sqrt{a^2 + (\epsilon - m)} \tag{2.17}$$

2.3.3. Affine region merging algorithm

Este algoritmo [2] también es un método para interpolar mapas de disparidad dispersos, y se basa en la combinación iterativas de regiones. Esto lo hace partiendo de una segmentación inicial R de la imagen de referencia, que consiste en un conjunto de regiones conectadas y disjuntas $R_1, R_2, \ldots, R_n$. Como las imágenes son tomas de zonas urbanas, éstas por lo general están compuestas por azoteas y otros elementos planos. Por lo que las disparidades de las regiones se pueden modelar como una transformación afín de tres parámetros $T(x, y) = ax + by + c$. El algoritmo aprovecha esta propiedad de las regiones para obtener para cada una de ellas la transformación que mejor se ajuste a los datos de entrada, esto lo hace buscando la transformación que minimice la siguiente formula:

$$T_i = \min_{t} \sum_{x \in \hat{R}_i} \rho\left(T(x) - d(x)\right) \tag{2.18}$$

siendo d el mapa de disparidad original que se quiere interpolar, $\hat{R}_i$ son los puntos válidos de la región y ρ es una función robusta[1].

Una vez que se ha obtenido la transformación que mejor se ajusta para cada región, se puede asumir una "coherencia entre regiones" para empezar a hacer una agrupación iterativas de ellas. La "coherencia entre las regiones" significa que las regiones vecinas tienen una transformación afín similar, por lo que pasarían a formar parte de la misma estructura. Por lo que el siguiente paso del algoritmo es decidir cuando dos regiones vecinas forman parte de la misma estructura. Esta decisión se hace midiendo el grado de significatividad con un número de falsas alarmas, a partir de la definición de un modelo a contrario [8, 9].

2.3.4. Region-based Affine Motion Estimation

Este algoritmo [4] al contrario de los dos anteriores, no parte de un mapa de disparidad disperso sino que directamente calcula los mapas densos.
El método que se usa se encuentra dentro de la categoría Optical Flow, y está basado en el macheo de la orientación del gradiente entre las imágenes. Esta técnica se usa para estimar el movimiento de los objetos de una escena en una secuencia de cuadros, por lo que en particular se puede aplicar para encontrar las disparidades entre un par de imágenes en estéreo. La forma en que trabaja es para cada curva de nivel busca el mapa de disparidad ϕ que minimiza la siguiente función de energía:

$$E(\phi) = \int_{\Omega} \rho\left(\left\|Z^1(\phi) - \bar{Z}_{\phi}\right\|^2\right) \delta x \delta y \tag{2.19}$$

siendo $Z^1(x)$ los vectores normales de la curva de nivel de la imagen secundaria y $\bar{Z}_{\phi}$ los vectores normales de la curva de referencia transformada por el mapa de disparidad ϕ.

[1] Una función es robusta si es simétrica, positiva y con un solo mínimo en cero

Capítulo 3

Desarrollo de ICMEZU

El trabajo realizado en esta parte del proyecto de grado, consiste en el desarrollo de una aplicación gráfica, que integra distintas herramientas para trabajar con algoritmos estereoscópicos en el cálculo de mapas de elevación de zonas urbanas.
Este capítulo se divide en dos secciones, en la primera se especifican los requerimientos capturados y en la segunda se hace una descripción del diseño.

3.1. Captura y Análisis de Requerimientos

En esta sección se especifican los requerimientos capturados para usarlos como guía en el diseño e implementación de la aplicación. También sirve para que una vez finalizada la misma se pueda validar además de saber si se cumplen las metas planteadas. Esta sección se divide en cuatro partes, en la primera se hace una descripción general de los requerimientos del sistema, en las siguientes se listan las operaciones necesarias para cumplir con los requerimientos funcionales, luego las restricciones de diseño y por ultimo se comentan los requerimientos de documentación.

3.1.1. Descripción general

Funciones del producto

La aplicación consiste en una interfaz gráfica, que permite trabajar con pares estéreo de imágenes aéreas o satelitales de zonas urbanas, para calcular mapas de elevación y poder testear distintos algoritmos estereoscópicos. Entre otras cosas el producto brinda las siguientes funcionalidades:

- Calcular mapas de elevación usando los algoritmos presentados e implementados en [3].
- Generar nuevos pares que simulen cambios en la altura de las cámaras, por lo que el usuario ingresando una nueva altura puede ver al par estéreo como si se hubiera sacado en esa posición.
- Estimar errores en los mapas de disparidad calculados, usando las máscaras devueltas por los algoritmos o las agregadas manualmente por el usuario.
- Integrar dinámicamente nuevos algoritmos para trabajar con imágenes estereoscópicas.
- Generar anaglifos.
- Visualizar las distantes imágenes cargadas o calculadas (par estéreo, mapas de disparidad, ground-truth, pares simulados, anaglifos).
- Ver escenarios 3D usando los mapas de disparidad calculados.

Interfaz con software

La aplicación se tiene que poder ejecutar bajo el ambiente Linux, además que se recomienda el uso de la librería MegaWave para el tratamiento de imágenes.

Interfaces de usuario

Como el producto está enfocado principalmente a construir una UI, se espera que las interfaces sean lo suficiente amigables como para permitir un uso fácil y práctico de las funcionalidades brindadas por el sistema. Además se requiere que usen la tecnología de ventanas y se opere mediante el mouse en vez del uso de líneas de comando.

Restricciones de memoria

Es importante que se pueda trabajar con imágenes grandes, esto implica que cuando se haga zoom a una imagen, no se haga el zoom de toda la imagen sino de solo la porción que está viendo el usuario, evitando de esta forma un uso excesivo de memoria.

Características de los usuarios

Los usuarios serán personas que les interese trabajar con imágenes estereoscópicas, por lo que contarán con los conocimientos técnicos suficientes para poder entender sin mayor esfuerzo la nomenclatura y funcionalidades de la aplicación.

3.1.2. Requerimientos Funcionales

A continuación se listan las operaciones que tiene que brindar el Sistema para cumplir con los requerimientos funcionales.

3.1.2.1. Crear Project

Crea una nueva sesión de trabajo, especificando la altura de la cámara, el valor del factor b/h (baseline/altura) y las rutas en el file system de las imágenes Referencia, Secundaria y Ground-Truth.

3.1.2.2. Guardar/Abrir Project

Guarda y abre una sesión con su juego de imágenes, mapas de disparidad, máscaras y simulaciones creadas.

3.1.2.3. Simular altura del par estéreo

A partir del par estero y un mapa de disparidad genera pares de imágenes que simulan un cambio en la altura de las cámaras. Esto permite que el usuario ingresando una nueva altura pueda ver como sería el par estéreo si se hubiera sacado en esa posición.

3.1.2.4. Visualización y Zoom

Visualiza las imágenes generadas (par estéreo real, ground-truth, mapas calculados, pares simulados, anaglifos, autocoherencias) y permite aplicarle distintas escalas de zoom.

3.1.2.5. Obtener Mapa de Disparidad

Calcula el mapa de disparidad usando el par estéreo y un algoritmo seleccionado por el usuario.

3.1.2.6. Obtener Error de un Mapa de Disparidad

Obtiene una estimación del error de un mapa de disparidad calculado con respecto al ground-truth, esta estimación se tiene que poder hacer con todos los pixeles de la imagen, y también usando solo los válidos por una máscara, ingresada por el usuario o devuelta por la ejecución de un algoritmo estereoscópico.

3.1.2.7. Ver representación 3D

A partir de un mapa de disparidad calculado o un Ground-Truth, genera una representación 3D de la escena.

3.1.2.8. Generar Anaglifo

Permite generar un anaglifo usando las imágenes referencia y secundaria reales o simuladas.

3.1.2.9. Generar Autocoherecia

Permite generar un anaglifo usando dos imágenes de referencia, una es la real y la otra es simulada con el algoritmo presentado en el apéndice A (como parámetros de entrada usa la imagen secundaria real y un mapa de disparidad).

3.1.2.10. Cargar Módulo de Cálculo de Disparidades

Registra en la aplicación una librería para calcular mapas de disparidad. Para realizar dicha operación el usuario tendrá que especificar una ruta de la ubicación de la librería y el nombre de la función.

3.1.2.11. Borrar Módulo de Cálculo de Disparidades

Permite al usuario borrar el registro de un módulo previamente cargado.

3.1.3. Requerimientos no Funcionales

3.1.3.1. Ambiente de Ejecución

La aplicación debe de correr bajo Linux, por lo que se desarrollará dentro de este ambiente.

3.1.3.2. Formato de las Imágenes

El sistema tiene que permitir trabajar con los siguientes tipos de imágenes que son soportados por el formato Fimage de MegaWave: rim, pm_f,img, pm_c, gif, tiff, pgma, pgmr, bmp, jfif, ps, epsf, inr, mti, bin, pmc_c, tiffc, bmpc, ppm, jfifc.

3.1.3.3. Uso de X-Windows

Al estar la interfaz gráfica orientada al uso de ventanas, se debe usar una librería que interactúe con las librerías de X-windows.

3.1.3.4. Carga dinámica de módulos

Los módulos que calculan mapas de disparidad se deben de poder cargar en la interfaz de forma dinámica y sin ser necesario compilar la misma.

3.1.4. Requerimientos de documentación

3.1.4.1. Manual de Usuario

Con la entrega del sistema se debe incluir un manual de usuario, con el nivel de detalle suficiente para que permita a un usuario hacer usufructo de la totalidad de las funcionalidades del sistema sin ningún mayor inconveniente.

3.1.4.2. Guías de instalación, configuración y archivo Léame

Al igual que el Manual de Usuario las guías de instalación deben permitir al usuario instalar la aplicación fácilmente, no siendo necesario ser un experto en Linux.

3.1.5. Casos de Uso

El documento de Casos de Uso se adjunta en el apéndice G.

3.2. Diseño

3.2.1. Diseño de Requerimientos relevantes

3.2.1.1. Simular altura del par estéreo (Requerimiento 3.1.2.3)

La simulación consiste en emular cambios en la altura en que se ha sacado el par estéreo real, para que el usuario ingresando una nueva altura pueda ver el par estéreo como si se hubiera sacado en esa posición. Una simulación recibe los siguientes parámetros de entrada:

Par Estéreo: Son las imágenes de referencia y secundaria que se usan como base para generar la simulación.

Mapa de Disparidad: Es el mapa de disparidad que se usa para estimar el desplazamiento horizontal a partir de un cambio de altura. El mapa usado puede ser uno calculado por un algoritmo estereoscópico o un ground-truth ingresado por el usuario.

Intervalo de Alturas: Por defecto la simulación se hace entre la altura de las cámaras de la imagen original (altura máxima) y el suelo (altura mínima). Por lo que el usuario en la simulación puede variar la altura entre ese intervalo [Suelo,Posición original de las cámaras]. Pero si el usuario lo desea puede cambiar la altura mínima ingresando un valor alternativo.

Numero de pares (N): Es la cantidad N de pares que se van a calcular dentro del intervalo [altura mínima,altura máxima] usando el algoritmo descrito en el apéndice A.

Para hacer la simulación se divide el intervalo [altura mínima,altura máxima] en N tramos de igual distancia, y para cada tramo se calcula un nuevo par simulado usando el algoritmo descrito en el apéndice A. Como se tiene una simulación discreta dentro del intervalo [altura máxima,altura mínima] (debido a que solo se tiene N pares calculados en todo el intervalo), para hacer la simulación densa, cuando el usuario ingresa una altura arbitraria h, se toman los dos pares de los tramos adyacentes a h, y haciendo una interpolación lineal de éstos se genera un nuevo par que simula la nueva altura h.

3.2.1.2. Representación 3D de la escena (Requerimiento 3.1.2.7)

Esta operación obtiene a partir de un mapa de disparidad (calculado por un algoritmo o el ground-truth ingresado por el usuario) una representación tridimensional de la escena.
La forma en que se hace lo anterior es a través de los siguientes pasos:

1. Usando el mapa de disparidad se obtiene un mapa de elevación, calculando para cada pixel su altura mediante la siguiente ecuación:

$$h = \frac{H.d}{B}$$

   H**:** Es la altura de las cámaras.

   B**:** Es la distancia entre las cámaras (baseline).

   d**:** Es la disparidad del pixel.

   h**:** Es la altura calculada para el pixel.

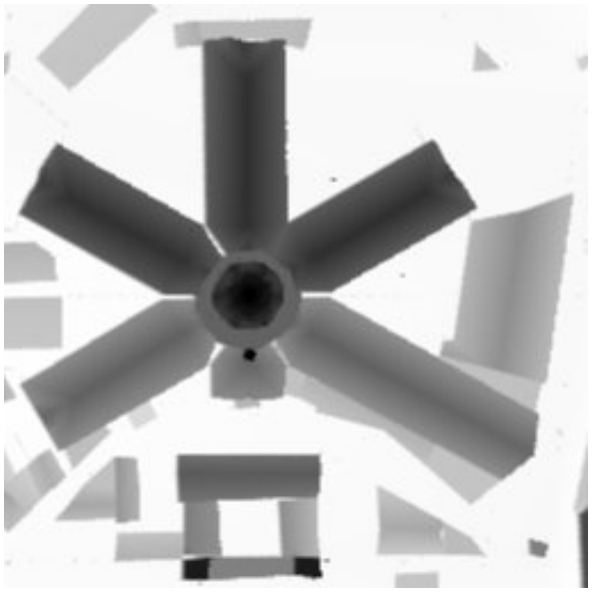
2. Con las alturas del mapa de elevación y las coordenadas bidimensional de los pixeles de la imagen de referencia, se obtiene las coordenadas de éstos en el espacio 3D

3. Se construye una malla tridimensional uniendo los pixeles 3D adyacentes, con triángulos usando la estructura conocida como triangle strip (figura 3.1). Se puede ver un ejemplo de una malla construida de esta forma en la figura 3.2(b).

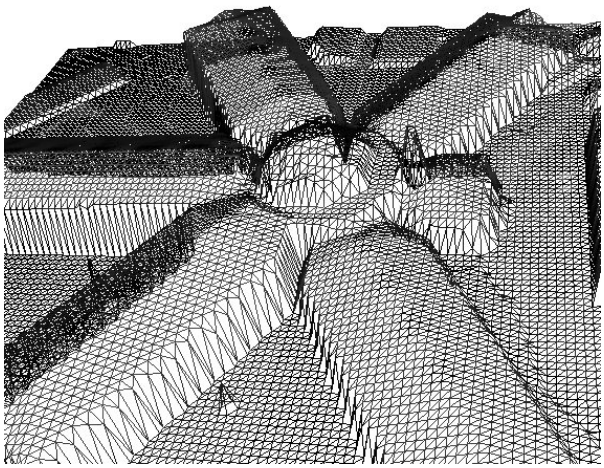
4. Se cubre la malla con la textura de la imagen de referencia, generando de esta forma la representación final de la escena 3D como se ve en la figura 3.2(d). El cubrimiento se hace pintando la superficie de cada triángulo, con una interpolación lineal entre los colores de sus 3 vértices (el color de un vértice 3D es del pixel que le corresponde en la imagen de referencia).

La implementación de los pasos 2, 3 y 4 se hace usando la librería VTK (sección 3.2.3.3), que permite generar el escenario 3D de una manera bastante sencilla mediante el uso de clases implementadas en C++. Para ver en más detalle de como se usa VTK en los pasos 2, 3 y 4 ver el apéndice E.

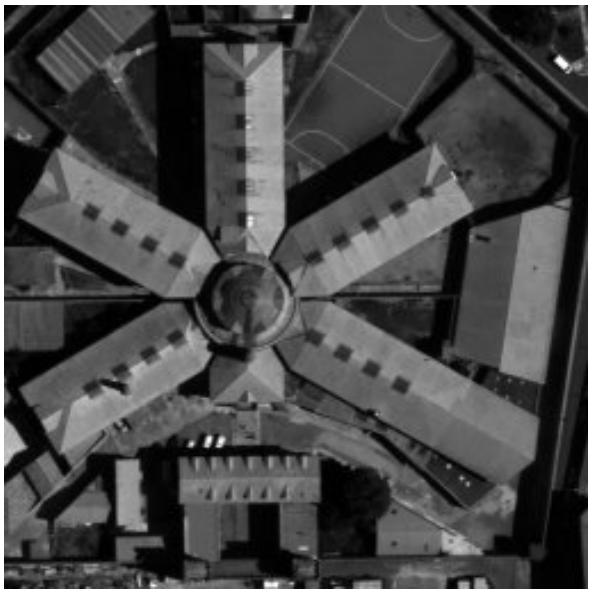
Figura 3.1: Triangle strip



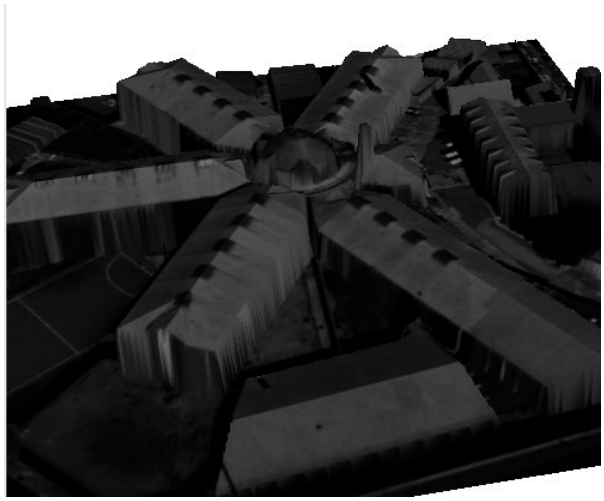
(a) Mapa de Disparidad usado para calcular el mapa de elevación



(b) Malla tridimensional construida con las alturas del mapa de elevación



(c) Imagen de referencia usada como textura



(d) Representación 3D final

Figura 3.2: Ejemplo de la construcción de una escena 3D

3.2.1.3. Integración dinámica de algoritmos estereoscópicos (Requerimiento 3.1.2.10)

El propósito de este requerimiento es incorporar nuevos algoritmos estereoscópicos a ICMEZU sin tener que recompilarlo. Esto se hace, usando librerías dinámicas que contengan las implementaciones de los nuevos algoritmos, y especificando sus parámetros de entrada en archivos de formato XML. Esto último, es para ICMEZU pueda saber que parámetros de entrada le tiene que solicitar al usuario que ingrese, cuando éste quiera ejecutar un algoritmo determinado. Los pasos que el usuario tiene que seguir, para registrar un nuevo algoritmo se detalla en el apéndice C. En forma resumida éstos son:

1. Encapsular el algoritmo en una función implementada en C con un cabezal como el especificado en el apéndice C.
2. Crear una librería dinámica (con el comando ld), que tenga la función anterior.
3. Especificar en un archivo xml los nombres, tipos y valores por defecto de los parámetros de entrada del algoritmo.
4. Registrar el xml creado en el paso anterior en ICMEZU para que ésta pueda usar la nueva librería con el algoritmo implementado.

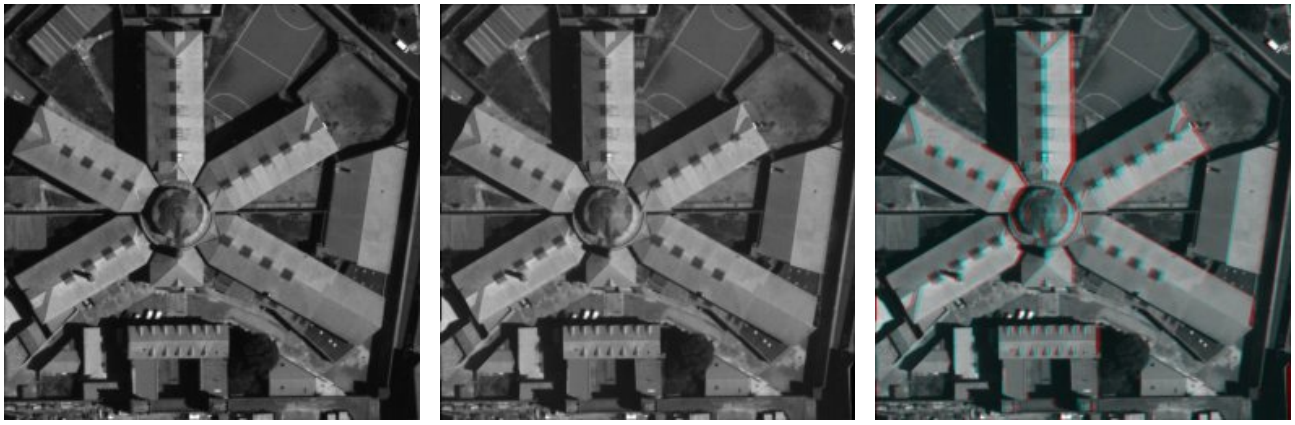
Por otro lado, cuando el usuario desde la aplicación selecciona un algoritmo para ejecutar, ICMEZU hace lo siguiente:

1. Primero obtiene desde el archivo XML que corresponde al algoritmo seleccionado, la especificación de sus parámetros de entrada, para luego solicitarle al usuario que ingrese sus valores.
2. Una vez que los valores son ingresados por el usuario, ICMEZU verifica que éstos cumplan con sus tipos (entero, punto flotante, string, enumerado y rango) especificados en el XML.
3. Si la verificación anterior pasa, se carga en la memoria la librería que implementa el algoritmo seleccionado. La ubicación de la librería en el file system y el nombre de su función principal, también se encuentran en el XML asociado al algoritmo seleccionado.
4. Se ejecuta la función principal de la librería cargada, pasándole como parámetros de entrada las imágenes de referencia y secundaria, y una lista de strings que contiene los valores ingresados por el usuario en el paso 2. Como parámetros de salida se le pasa el mapa de disparidad y la máscara.
5. Cuando la función finaliza, se libera de la memoria la librería cargada en el paso 3.

Para ver en más detalle de como se ejecuta dinámicamente las funciones (pasos 3, 4 y 5) ver el apéndice D.

3.2.1.4. Generación de Anaglifos (Requerimiento 3.1.2.8)

Un anaglifo se construye superponiendo la imagen de referencia con la secundaria, provocando una sensación tridimensional si se observa esta imagen con unos lentes de colores (lente izquierdo rojo y el derecho azul). Para hacer la superposición, se crea una imagen de tres canales RGB, donde en el canal R va la imagen izquierda del par y en los canales G y B la imagen de la derecha. De esta forma, cuando se la observa con lentes de colores, cada ojo ve la imagen que le corresponde (el ojo izquierdo ve la imagen izquierda del par y el derecho la derecha), y luego el cerebro une las dos imágenes en una sola imagen tridimensional.



(a) Referencia (b) Secundaria (c) Anaglifo

Figura 3.3: Ejemplo de un anaglifo generado con las imágenes de referencia y secundaria

3.2.1.5. Generación de Autocoherencias (Requerimiento 3.1.2.9)

Una imagen de autocoherencia se crea de una manera similar a lo de un anaglifo, salvo que no se usa para ver una imagen con una sensación de relieve, sino que sirve para ver las zonas de no coherencia de un mapa de disparidad calculado. Para crear esta imagen, primero se genera una imagen de referencia simulada usando un mapa de disparidad y la imagen secundaria (como se muestra en el apéndice A). Luego la imagen de referencia original y la calculada anteriormente, se superponen de la misma manera que un anaglifo. Si el mapa de disparidad fuera exacto, las dos imágenes de referencia (la real y simulada) serían idénticas. De esta forma, el anaglifo generado con éstas sería una imagen de escala de gris debido a que los tres canales RGB tendrían el mismo valor para cada pixel. Pero como el mapa de disparidad contiene errores, existirán zonas donde las dos imágenes de referencia son distintas, y esto se verá reflejado en las pixels de color del anaglifo, a causa de que los tres canales RGB tendrán distintos valores.

3.2.1.6. Zoom en Imágenes (Requerimiento 3.1.2.4)

Para hacer grandes escalas de zoom sobre las imágenes visualizadas sin un uso excesivo de la memoria RAM, se utiliza la librería gtkimageviewer. Debido a que ésta lo resuelve haciendo zoom solo a las zonas de la imagen que cabe en la pantalla, y a medida que el usuario va haciendo panning para ver el resto de la imagen, la librería le va aplicando zoom a las nuevas zonas expuestas.

3.2.1.7. Estimación de Errores en los mapas calculados (Requerimiento 3.1.2.6)

La estimación de los errores de los mapas de disparidad, se pueden hacer usando el 100 % de sus pixeles o solo los válidos por las máscaras (que son ingresadas por el usuario o devueltas por los algoritmos). Para calcular los errores, dado un mapa e, un ground-truth ϵ y una máscara M se usan los siguientes estimadores:

$$L1(e, \epsilon, M) = \sum_{x \in M} \frac{1}{|M|} |\epsilon(x) - e(x)|$$

$$L2(e, \epsilon, M) = \sum_{x \in M} \frac{1}{|M|} (\epsilon(x) - e(x))^2$$

$$Linf(e, \epsilon, M) = \max_{x \in M} |\epsilon(x) - e(x)|$$

3.2.1.8. Guardar/Abrir Project (Requerimiento 3.1.2.2)

Un Project es una sesión de trabajo que abarca un par estéreo real y las imágenes (mapas, máscaras, simulaciones y autocoherencias) que se fueron generando a partir de éste. El propósito de este requerimiento es que el usuario pueda persistir en el file system distintas sesiones de trabajo, para recuperarlas en otro momento y seguir trabajado en ellas. La información comprendida por éstos es:

- Datos generales:
  - Nombre del Project.
  - Imágenes del par estéreo y el ground-truth.
  - Imagen de autocoherencia generada con el ground-truth.
  - Altura de las cámaras.
  - B/H (baseline/altura de las cámaras).
- Mapas de disparidad que se calcularon con el par estéreo real, que para cada mapa se tiene la siguiente información:

    - Nombre del mapa de disparidad.
    - Nombre del algoritmo con que se calculó.
    - Parámetros de entrada con que se corrió el algoritmo.
    - Fecha en que se hizo la ejecución.
    - Imagen del mapa de disparidad.
    - Imagen de autocoherencia generada con el mapa de disparidad.
- Simulaciones generadas
    - Nombre del mapa de disparidad que se usó para generar la simulación.
    - Cantidad de pares que se generaron.
    - Altura mínima del rango que se simuló
    - Imágenes de los pares simulados.
- Máscaras:
    - Nombre de la máscara.
    - Imagen de la máscara.

Para persistir la información anterior (excepto para las imágenes) se usa el formato XML, debido a que se tomó en cuenta las ventajas que tiene dicho formato:

- El analizador sintáctico (parser) del formato XML es un componente estándar y existen varias implementaciones del mismo, por lo que no es necesario crear un nuevo parser.
- Debido a su forma estructurada en etiquetas, es fácil de entender y hacer modificaciones.

3.2.2. Arquitectura del Sistema

Al ser una aplicación gráfica con persistencia de la información, se eligió una arquitectura en capas con las capas de presentación, lógica y acceso a los datos. La primera capa se encarga de la visualización gráfica e interacción con el usuario, la segunda agrupa los distintos módulos que implementa la lógica de las funcionalidades brindadas, y la ultima es la que se encarga de persistir los datos en el file system.

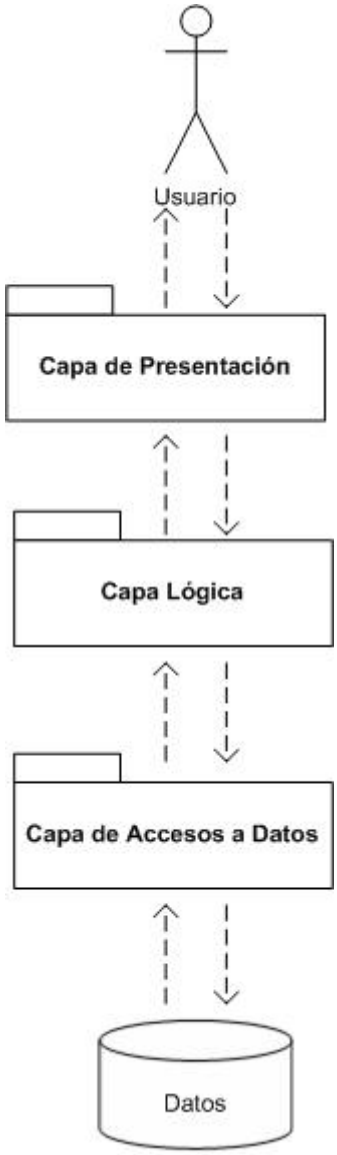


Figura 3.4: Arquitectura del Sistema

3.2.2.1. Capa de Presentación

La capa de presentación está formada por módulos, donde cada uno de ellos implementa un diálogo o una ventana que interactúa con el usuario para que éste pueda realizar una operación determinada. Estos módulos no procesan las órdenes realizadas por el usuario, sino que solo son puntos de entrada que las reciben para mandárselas a la capa lógica y luego devolverle los resultados de las mismas. A continuación se listan los módulos contenidos en esta capa:

- V-Menu : Implementa la ventana principal que se carga al iniciar ICMEZU, recibe las órdenes realizadas por el usuario, y luego las delega a los otros diálogos o se comunica directamente con la capa lógica.
- ViewTests: Es una tabla contenida en la ventana principal, que muestra los algoritmos estereoscópicos ejecutados al par estéreo del Project actual.
- ViewErrors: Es una tabla contenida en la ventana principal, que muestra los errores de los mapas de disparidad calculados.
- V-NewProject: Es una ventana que solicita la información necesaria para crear un nuevo Project.
- V-EditProject: Es una ventana mediante la cual se puede editar la información del Project creado.
- V-CalcDisparidades: Permite crear una nueva ejecución de un algoritmo, mediante esta ventana el usuario tendrá que seleccionar el algoritmo que quiere ejecutar e ingresar los valores de los parámetros de entrada del mismo.
- V-AgregarMask: Permite al usuario seleccionar una nueva máscara para calcular los errores de los mapas de disparidad.
- V-NewSimulacion: Solicita la información para crear una nueva simulación de pares estéreo.
- V-Visor: Permite visualizar las distintas imágenes del Project: par estéreo real, ground-truth, simulaciones y mapas de disparidad calculados.
- V-AgregarModulos: Dialogo para un nuevo registro de un algoritmo estereoscópico a la aplicación.
- V-EliminarModulos: Dialogo para eliminar el registro de un algoritmo previamente registrado.

3.2.2.2. Capa Lógica

En esta capa se maneja la lógica de las funcionalidades del sistema a través de los siguientes módulos:

- ModelProjects: Maneja en memoria el estado del Project actual con que está trabajando el usuario (información del par estéreo, algoritmos ejecutados, máscaras agregadas, simulaciones creadas, etc.).
- SimuladorParEstereo: Módulo que implementa los algoritmos de simulación de pares estéreo.
- IAlgoritmo: Interfaz que se comunica dinámicamente con los algoritmos estereoscópicos integradas a la aplicación.
- ToolFimage: Implementa operaciones básicas que tengan relación con las imágenes (calcular el error de un mapa de disparidad, crear un anaglifo, etc.).

3.2.2.3. Capa de Persistencia

La información que se persiste en el file system consiste en la relacionada a los Projects (dividida en datos e imágenes) y la que se refiere a los algoritmos cargados en la aplicación (de estos últimos se guardan el nombre, ubicación de la librería, tipo y valores por defecto de los parámetros de entrada, etc.).

- ProjectPersist: Se encarga de guardar en y recuperar desde el file system los Projects creados por el usuario.
- ImagePersist: Maneja la persistencia de las imágenes.
- AlgoritmosPersist: Se encarga de guardar y recuperar desde el file system los algoritmos registrados en la aplicación.

3.2.3. Plataforma de Desarrollo

3.2.3.1. Sistema Operativo

Linux Fedora Core 3

3.2.3.2. Lenguaje de Programación

C y C++

3.2.3.3. Principales Librerías Usadas

- GTK+-2.0 [19]: The GIMP Toolkit, es un conjunto de herramientas multi-plataforma para crear interfaces gráficas de usuario.
  Los motivos por lo que se ha seleccionado esta herramienta son:
  - Brinda completamente las funcionalidades necesarias para el desarrollo de la interfaz gráfica.
  - Cuenta con una muy buena y extensa documentación gratuita [19].
  - Es software libre y con licencia GNU LGPL, teniendo todas las ventajas que esto conlleva.
  - Es multi-plataforma, haciendo más flexible la migración a otras plataformas en caso que sea necesario.
  - Soporta los lenguajes C/C++.
  - Viene instalada en la mayoría de las distribuciones de Linux. Facilitando de esta forma la implantación en distintas distribuciones la aplicación desarrollada. Este punto fue el decisivo de por que se eligió esta librería y no otras (ej. wxWidgets) con características similares pero que no vienen por defecto en las distribuciones de Linux.
- GtkImageViewer-0.3.4 [21]: Es una librería para visualizar imágenes con la característica especial que cuando se hace zoom a una imagen, solo se le aplica esta operación a las zonas expuestas y no a toda la imagen. Para su selección se tomó en cuenta lo siguiente:
  - Es muy fácil integrarla a la interfaz gráfica debido a que está implementada usando C y GTK.
  - Tiene la característica especial de hacer zoom solo a las áreas que están expuestas y no a toda la imagen.
  - Forma parte del software libre.

- VTK-5.0.3 [20]: The Visualization ToolKit, es una herramienta para generar gráficos 3D y fue la que se usó para crear los escenarios 3D por los siguientes motivos:
  - Permite crear los escenarios 3D de una forma sencilla.
  - Es soportada por múltiples plataformas (basadas en Unix, Windows 98/ME/NT/2000/XP y Mac OSX Jaguar)
  - Cuenta con documentación gratuita [20].
  - Forma parte del software libre.
  - Soporta C++.
- MegaWave2-2.30 [18]: Es un software libre para el procesamiento de imágenes, que brinda en una librería en C módulos con algoritmos escritos por investigadores.
  - Cuenta con una gran variedad de algoritmos para el tratamiento de imágenes.
  - Los algoritmos estereoscópicos con los que son trabajados en este proyecto usan esta librería.
  - Está implementada en C.
  - Es software Libre.
- libxml-2.0: Debido a que la información persistida de los Projects es bastante simple y no es requerido hacer consultas sobre ésta. Se decidió usar el formato de archivos xml para realizar la misma. Esto se hace mediante la librería libxml a causa de que está implementada en C y viene en la mayoría de las distribuciones de Linux.

Capítulo 4

Algoritmo de líneas de nivel afines

El trabajo realizado en esta segunda parte del proyecto de grado, consiste en la evaluación de un nuevo algoritmo estereoscópico, iniciado como tarea final del curso "Teoría Computacional de la Gestalt 2005". El objetivo del mismo es calcular mapas de disparidad subpixel para pares de imágenes satelitales urbanas con un b/h (baseline / altura) pequeño.
Para hacer la búsqueda de correspondencias, el modelo propuesto se basa en las semejanzas entre las curvas de nivel del par estéreo, por lo que éste se puede catalogar como un algoritmo del tipo Feature Matching. Las curvas de nivel interpoladas permiten conseguir las disparidades sub-pixelares. Aunque tiene la desventaja de generar mapas de disparidad no densos, haciendo que sea necesario aplicar técnicas de interpolación a los mapas devueltos por él.
Este capítulo se divide en tres secciones principales, en la primera se enuncian las definiciones principales de la Teoría Computacional de la Gestalt en la cual está basado este método, luego en la siguiente sección se presenta el algoritmo propuesto y en la última sección se estudian los resultados obtenidos con su implementación.

4.1. Enunciados de la Teoría Computacional de la Gestalt

Para hacer las correspondencias entre las curvas de nivel, el algoritmo usa cierto marco teórico perteneciente a la Teoría Computacional de la Gestalt, por lo que a continuación se enuncian algunas de sus definiciones principales extraídas de [1, 8, 9].

Principio de Helmholtz Dado n objetos $O_1, O_2, \ldots, O_n$ presentes en una imagen, donde k de ellos tienen una característica en común (por ejemplo, el mismo color, el mismo tamaño, etc.), se desea saber si esta característica está presente por casualidad en los k objetos o si no lo está y forman parte de una misma entidad. Entonces se asume que la característica ha estado aleatoriamente distribuida en los objetos $O_1, O_2, \ldots, O_n$, por lo que se supone que los objetos observados son una realización al azar de este proceso. Finalmente se hace la siguiente pregunta: ¿es probable en este modelo la distribución observada o no?. Si la respuesta es no, se prueba por modelo a contrario que estos objetos se pueden agrupar como parte de la misma entidad.

Definición 1 (evento ϵ-significativo) *Se dice que un evento del tipo "Una configuración de objetos geométricos tienen cierta propiedad", es ϵ-significativa si el numero de ocurrencias esperadas del evento es menor a ϵ asumiendo una distribución aleatoria.*

Definición 2 (Número de Falsas Alarmas - NFA) *Dado un evento de la forma "Una configuración de objetos que tienen cierta propiedad en común", su numero de falsas alarmas (NFA) es el número esperado de ocurrencias del evento asumiendo una distribución aleatoria.*

Usando el NFA se pude redefinir la Definición 1

Definición 3 (evento ϵ-significativo) *Un evento E del tipo "Una configuración de objetos geométricos tienen cierta propiedad", es ϵ-significativo si el numero de falsas alarmas es menor a ϵ.*

$$NFA(E) < \epsilon \tag{4.1}$$

Si H_1 es la hipótesis "Dada una configuración de objetos con cierta propiedad que se produce al azar", y se define la variable aleatoria ξ para analizar la observación del evento E, se define el NFA como

Definición 4 (Número de Falsas Alarmas - NFA) *El número de falsas alarmas (NFA) del evento E se define como:*

$$NFA(E) = N.P[\xi \geq E|H_1] \tag{4.2}$$

donde N es el número de todas las posibles configuraciones de H_1

4.2. Presentación del Algoritmo

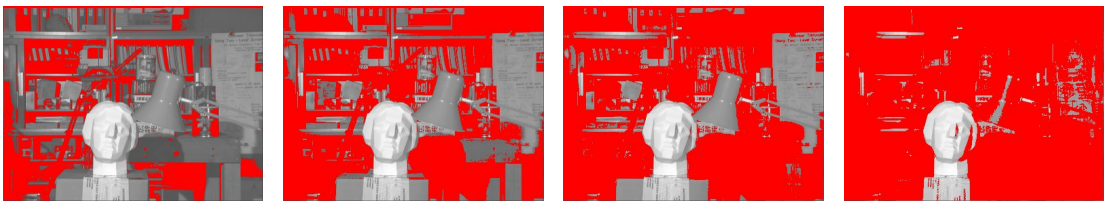
La idea general del algoritmo consiste en detectar correspondencias entre las curvas de nivel de ambas imágenes para luego obtener las correspondencias entre los píxeles contenidas en ellas. Las curvas de nivel son las fronteras de los conjuntos de nivel de una imagen, estos últimos se definen como:

Definición 5 (Conjuntos de Nivel) *Los conjuntos de nivel de una imagen en tonos de gris* $u : R^2 -> R$ *se define como*

$$\xi_\lambda = \{x \in R^2 : u(x) \geq \lambda\} \tag{4.3}$$

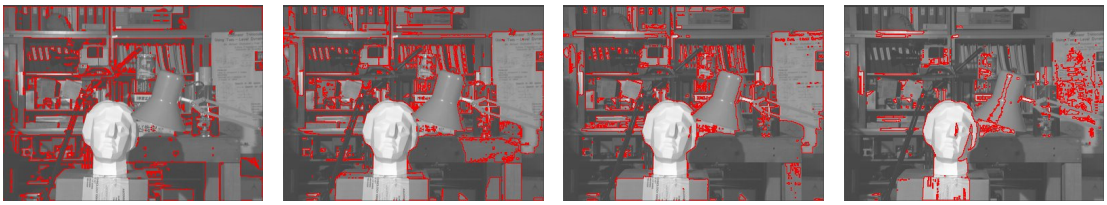
donde λ *toma valores en todo* R

La estructura de una imagen se puede descomponer de forma reversible usando los conjuntos de nivel, o sea que a partir de estos últimos se puede reconstruir la imagen. Su importancia en la detección de correspondencias es que en las líneas de nivel hay suficiente información para obtener los contornos de los objetos [7, 8]. En las figuras 4.1 y 4.2 se muestran los conjuntos y líneas de nivel para algunos valores de λ.



(a) $\lambda = 25$ (b) $\lambda = 60$ (c) $\lambda = 80$ (d) $\lambda = 120$

Figura 4.1: Conjuntos de nivel para valores de $\lambda = 25, 60, 80, 120$. Las zonas rojas identifican las áreas donde la tonalidad de gris es menor a λ



(a) $\lambda = 25$ (b) $\lambda = 60$ (c) $\lambda = 80$ (d) $\lambda = 120$

Figura 4.2: Líneas de nivel para valores de $\lambda = 25, 60, 80, 120$. Las curvas rojas son las líneas de nivel que limitan los conjuntos de nivel

Al trabajar con imágenes aéreas de zonas urbanas, la mayoría de los objetos observados son planos (azoteas, paredes, suelo, etc.), por lo que se puede suponer que la correspondencia de los objetos entre el par de imágenes (como se muestra en [1]) está dada por una transformación de la forma

$$T(x, y) = ax + by + c \tag{4.4}$$

donde ax representa un estiramiento horizontal del objeto, by un sesgo vertical y c una traslación horizontal. Por la suposición anterior, para poder comparar curvas y obtener una medida de similitud entre ellas, se define un sistema de coordenadas normalizado invariante, al grupo de las transformaciones mencionadas anteriormente.
Básicamente el algoritmo se divide en estas dos etapas que se describen en las secciones siguientes:

1. Se obtiene para cada pixel de las dos imágenes, un trozo de la curva de nivel que pasa por él y luego se la normaliza en un sistema de coordenadas invariantes a las transformaciones mencionadas anteriormente.

2. Se define un modelo a contrario para obtener las correspondencias entre los trozos de curvas afines calculadas en el paso anterior, y de esta forma calcular las correspondencias entre los pixeles.

4.2.1. Etapa I - Normalización de Curvas

Como se mencionó anteriormente, el primer paso del algoritmo consiste en obtener para cada pixel de las dos imágenes, el trozo de curva de nivel que pasa por él, y luego normalizarla en un espacio invariante a las transformaciones afines. Para realizar lo anterior se hace para cada pixel la siguiente secuencia de pasos

1. Se obtiene la curva de nivel C que pasa por el pixel P.

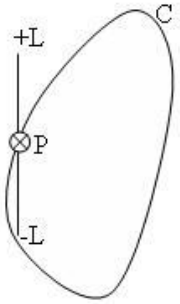


Figura 4.3: Paso 1

2. De la curva C, solo se queda con el trozo de curva que está dentro de los L píxeles arriba y abajo del pixel P.

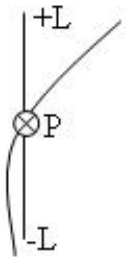


Figura 4.4: Paso 2

3. Se calcula un eje de coordenadas para hacer la normalización, el **eje X** es la recta horizontal que pasa por el pixel, y el **eje Y** es la recta que mejor se aproxima al trozo de curva (para calcular el **eje Y** se usa la regresión lineal especificada en el apéndice B) como se ve en la figura 4.5. El centro del eje es la intersección entre el **eje X** y el **eje Y**.

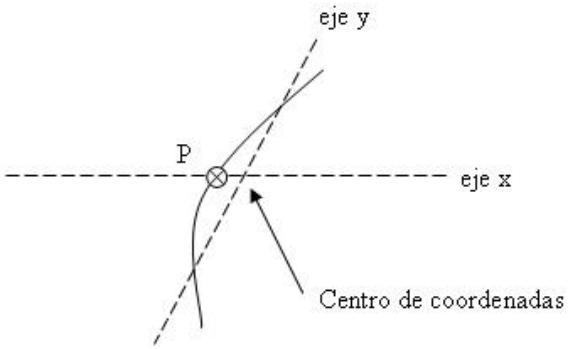


Figura 4.5: Paso 3

4. Finalmente se calcula las nuevas coordenadas del trozo de curva con respecto al nuevo eje de coordenadas, y se normaliza la curva dividiendo las coordenadas entre la distancia del punto de la curva que está más alejado del **eje Y**.

Una vez que se ha realizado lo anterior, se pasa a la siguiente etapa del algoritmo que consiste en la búsqueda de correspondencias de las curvas entre el par de imágenes.

4.2.2. Etapa II - Búsqueda de correspondencias entre trozos de curvas

En esta segunda etapa del algoritmo, se empieza a buscar para cada pixel de la imagen de referencia, el pixel de la secundaria que tiene el trozo de curva más parecido al que pasa por él. Por la geometría epipolar y la suposición de que las imágenes han sido rectificadas (secciones 2.1.2 y 2.1.3) la traslación de las curvas es sobre el **eje X**. Entonces la búsqueda se restringe en una dimensión sobre el eje horizontal entre un intervalo $\pm D$ de la posición del pixel original, siendo D la disparidad máxima. Cuando se encuentra una "muy parecida" se tiene una correspondencia entre curvas y por lo tanto una correspondencia entre píxeles. Para decidir cuando el parecido entre dos curvas C_1 y C_2 es significativo se define el NFA de curvas dispares como:

Definición 1 *El número de falsas alarmas (NFA), de dos curvas a una distancia d, se define como:*

$$NFA(C_1, C_2) = N_{tests} P(dist(C_x, C_y) < d) \tag{4.5}$$

donde N_{tests} es el número de configuraciones que puede haber entre todos los pares de curvas de las dos imágenes, y $P(dist(C_x, C_y) < d)$ es la probabilidad que la distancia entre un par de curvas sea menor a d (siendo d la distancia entre las curvas C_1 y C_2).

Dado un umbral ϵ, se dice que el par de curvas C_1 y C_2 es ϵ-significativo si el número de falsas alarmas es menor que ϵ.
El N_{tests} es el número total de pares de curvas que se pueden formar. Este número está dado por el producto entre la cantidad de trozos de curvas de la imagen de referencia y el número de curvas de la imagen secundaria usados para comparar una curva de la imagen de referencia. Por lo que queda definido como:

$$N_{tests} = \frac{|I_1|}{2L} 2D \tag{4.6}$$

- $\frac{|I_1|}{2L}$ estima la cantidad de curvas de la imagen de referencia, ya que $|I_1|$ es la cantidad de píxeles de la imagen de referencia (cada pixel tiene un trozo de curva de nivel propio) y se lo divide por la constante $2L$ para disminuir la dependencia entre las curvas.
- $2D$ es el número de trozos de curvas de la imagen secundaria usadas para comparar un trozo de curva de la imagen de referencia, ya que D es la disparidad máxima y la búsqueda se restringe en un intervalo $\pm D$.

El cálculo de la probabilidad que dos curvas tengan una distancia menor a d, se resuelve de manera empírica mediante la construcción de un histograma de distancias, este histograma se construye de la siguiente forma:

$$H(d) = \frac{\#\{\, (C_x, C_y) \,/\, C_x \in I_1,\ C_y \in I_2 \ y\ dist(C_x, C_y) < d\}}{|I_1||I_2|} \tag{4.7}$$

Usando lo anterior se puede reescribir la Definición 1 como:

Definición 2 *El número de falsas alarmas (NFA), de dos curvas C_1 y C_2 de distancia d se define como:*

$$NFA(C_1, C_2) = D\frac{|I_1|}{L}H(d) \tag{4.8}$$

Como es posible que las imágenes contengan curvas simples pero muy frecuentes (ej. segmentos de recta), puede suceder que usando un solo histograma las distancias pequeñas fueran bastantes probables y llevaría a no encontrar curvas significativas. Para evitar lo anterior, se clasifican las curvas en distintos grados de complejidad y luego para cada uno de estos grupos se construye un histograma distinto. De esta forma, cuando se calcula la probabilidad que dos curvas C_1 y C_2 tengan una distancia menor a d, se usa el histograma construido con las curvas del mismo grado de complejidad de C_1.
Actualmente el algoritmo clasifica el grado de complejidad de una curva mediante el redondeo de la distancia del punto de la curva que está más alejado del eje Y (que se calcula en el paso 3 de la sección 4.2.1). En la figura 4.6 se puede ver un ejemplo de una curva C con un grado de complejidad round(k), debido a que el punto más alejado de C con respecto al eje Y tiene una distancia horizontal k.

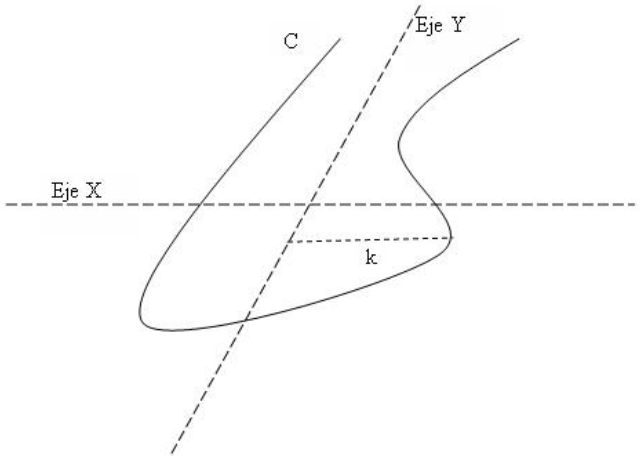


Figura 4.6: Ejemplo de curva de complejidad k

4.2.3. Cálculo de la disparidad subpixelar usando dos curvas significativas

Cuando se tiene una correspondencia significativa entre dos trozos de curvas C_1 y C_2, donde el primero pertenece a un pixel P de coordenadas (x, y) de la imagen de referencia (I_1) y el segundo a otro pixel de la imagen secundaria (I_2). Se puede estimar la disparidad del pixel P de la siguiente manera.

1. Primero se obtiene las coordenadas del pixel P relativas al eje de coordenadas de la curva C_1. En la figura 4.7 se puede ver un ejemplo de las coordenadas relativas (x_r, y_r), el componente vertical y_r es siempre 0 por la forma en que se construye el eje horizontal (descripto en la sección 4.2.1).

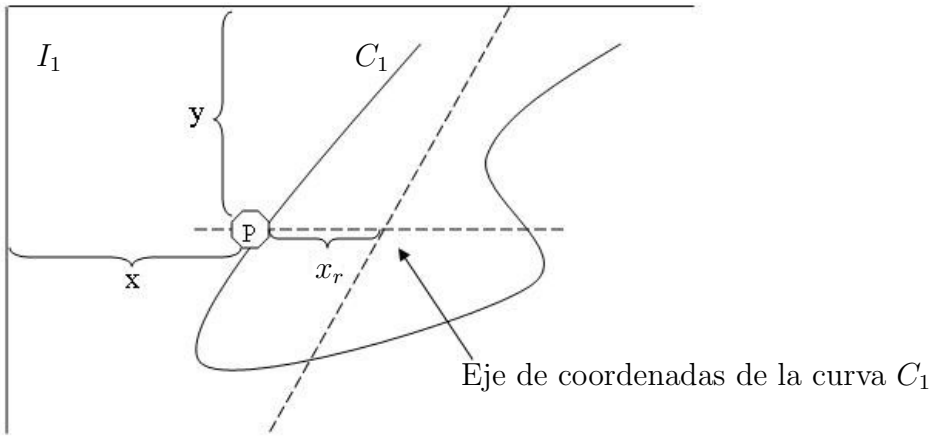


Figura 4.7: Paso 1

2. Luego se aproxima $T(P)$ en I_2, siendo T la transformación afín tal que $T(C_1) = C_2$. Esto se hace calculando el punto $Q = T(P)$ usando las coordenadas relativas (x_r, y_r) con el eje de coordenadas de la curva C_2 en I_2 (como se puede ver en la figura 4.8).

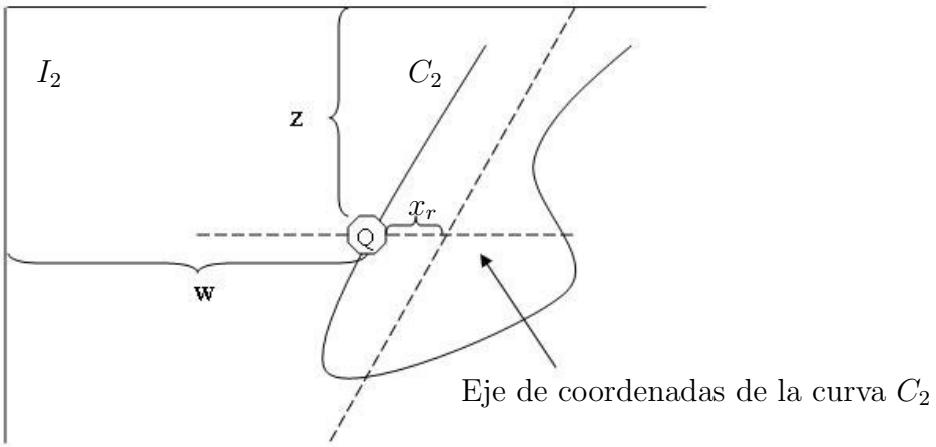


Figura 4.8: Paso 2

3. Finalmente se obtiene las coordenadas absolutas (w, z) con respecto a I_2 de Q , y la disparidad se calcula como $w - x$.

4.2.4. Bosquejo del Algoritmo

Sintetizando las secciones anteriores, se hace un bosquejo del algoritmo que recibe como parámetros de entrada las imágenes de referencia (I_1) y secundaria (I_2), la disparidad máxima D y el umbral ϵ usado.

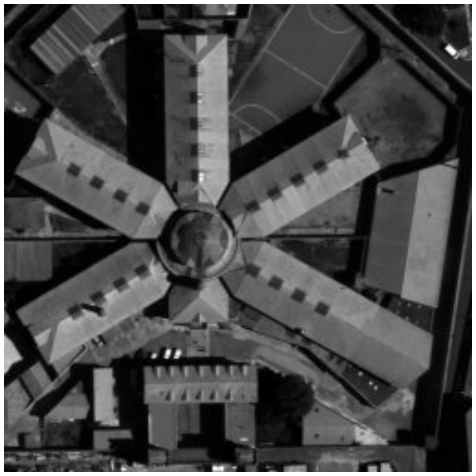
1. Para cada pixel de la imagen de referencia I_1 y de la imagen secundaria I_2 se calcula su trozo de curva normalizado como se detalle en la sección 4.2.1.
2. Con un muestreo de los píxeles de I_1 y I_2, se estima el histograma H de la ecuación 4.7.
3. Para cada pixel $P_{(x,y)} \in I_1$ se hace la siguiente secuencia de pasos:
   - *a*) Se obtiene la curva $C_{x,y}$ de $P_{(x,y)}$ calculada en el paso 1
   - *b*) $NFA_{min} = \epsilon + 1$.
   - *c*) FOR $i = x - D$ TO $x + D$
     1) Se obtiene la curva $C_{i,y}$ del pixel $P_{(i,y)} \in I_2$ calculada en el paso 1
     2) Se calcula el NFA de las curvas $C_{x,y}$ y $C_{i,y}$ usando la ecuación 4.8.
     3) Si $NFA(C_{x,y},C_{i,y}) < NFA_{min}$
        - $NFA_{min} = NFA(C_{x,y},C_{i,y})$
        - $x_{min} = i$
   - *d*) Si $NFA_{min} \leq \epsilon$
     - Se calcula la disparidad d de los píxeles $P_{(x,y)}$ y $P_{(x_{min},y)}$ usando las curvas $C_{x,y}$ y $C_{x_{min},y}$ (como se describe en la sección 4.2.3).
     - $Mask(x, y) = VALIDO$
     - $Mapa(x, y) = d$
   - *e*) Si no
     - $Mask(x, y) = INVALIDO$
4. El algoritmo retorna la máscara Mask con sus puntos válidos y el Mapa con las disparidades calculadas.

4.3. Resultados Experimentales

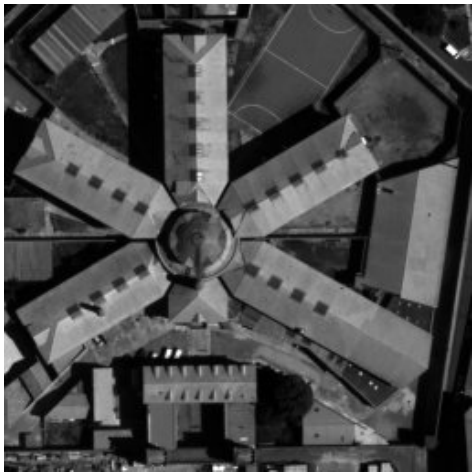
En esta sección del capítulo se analiza los resultados obtenidos con una implementación realizada del algoritmo presentado anteriormente. Para hacer las pruebas se usa un juego de imágenes real, y se comparan los resultados con los del algoritmo MARC. Se ha seleccionado MARC para hacer la comparación de los resultados experimentales debido a que ambos algoritmos tienen en común el objetivo de detectar disparidades sub-pixelares sin la necesidad de generar mapas densos, y además de que ambos difieren en las técnicas usadas para calcular dichas disparidades. Otro de los principales motivos para esta elección, es que MARC ha sido uno de los mejores algoritmos encontrado en la bibliografía consultada para el cálculo de disparidades sub-pixelares.
Esta sección está formada por tres subsecciones, en la primera se presentan el juego de datos y estimadores para analizar los resultados, luego en la siguiente se compara el error de los mapas no densos devueltos por los algoritmos, y en la última se hacen las comparaciones entre interpolaciones de los mapas.

4.3.1. Juego de Datos y Estimadores

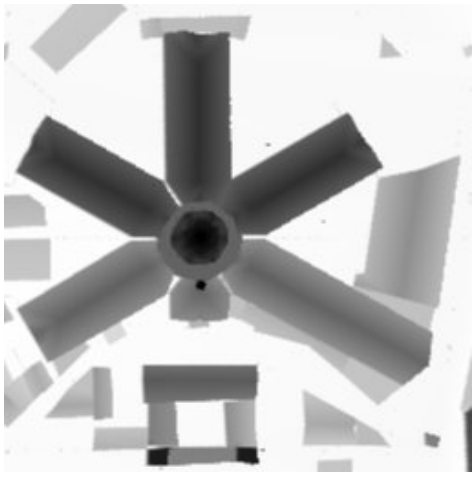
El juego de imágenes usado es un par estéreo real (figura 4.9(a) y 4.9(b)), con un tamaño de 512x512 píxeles, un b/h (baseline / altura) de 0.045 y una disparidad máxima de ± 3 píxeles. Para calcular los errores de los mapas de disparidad, se usa un ground-truth (figura 4.9(c)) que tiene las disparidades reales. Como el par estéreo se ha sacado con una diferencia mayor a 20 minutos, también se usa una máscara (figura 4.9(d)) que elimina los objetos en movimientos (autos, sombras, etc.). La forma en que se usa la máscara, es descartando los pixeles negros de ésta para calcular el error.



(a) Imagen de Referencia



(b) Imagen Secundaria



(c) Ground-Truth

(d) Máscara

Figura 4.9: Juego de Imágenes usado en los Tests

Los estimadores definidos para calcular el error, dado un mapa de disparidad e, usando un ground-truth g y una máscara M son :

- El promedio del valor absoluto de las diferencias de g y e usando solos los puntos válidos (puntos blancos) de M.

$$L1(e, g, M) = \sum_{x \in M} \frac{1}{|M|} |g(x) - e(x)|$$

- El máximo de las diferencias entre g y e usando los puntos válidos de M

$$Linf(e, g, M) = \max_{x \in M} |g(x) - e(x)|$$

- El promedio de las diferencias al cuadrado de ϵ y e usando los puntos válidos de M.

$$L2(e, g, M) = \sum_{x \in M} \frac{1}{|M|} (g(x) - e(x))^2$$

- Al igual que el anterior este estimador calcula el promedio de las diferencias al cuadrado, pero usando los parámetros a y b que minimicen la ecuación, además que en vez de calcular las diferencias punto a punto se compara con el mínimo dentro de una región $r = 1$, esto se hace para tener un mejor estimador en caso que el ground-truth tenga una pequeña deformación geométrica.

$$LP(e, g, M) = \sum_{x \in M} \frac{1}{|M|} \min_{|x' - x| \leq r} (e(x')a + b - g(x))^2$$

4.3.2. Resultados no densos

Usando el algoritmo de líneas de nivel afines (LNA) con el juego de imágenes presentado anteriormente, se obtiene el mapa de disparidad y la máscara que se ven en las figuras 4.10(a) y 4.10(b). La cantidad de puntos detectados por el algoritmo es del 22 %, y los errores estimados por $L1$ y $L2$ son 0,65 y 0,74 respectivamente. En cambio con MARC, el mapa y la máscara calculada (figuras 4.11(a) y 4.11(b)), tiene una densidad de 15,7 % y los errores estimados por $L1$ y Lp son 0,44 y 0,28. Éstos y los otros errores calculados se pueden ver en la siguiente tabla.

Estimador	LNA	MARC
densidad	22.83 %	15.15 %
L1	0.65	0.44
L2	0.74	0.28
Lp	0.43	0.21
Linf	4.89	3.46

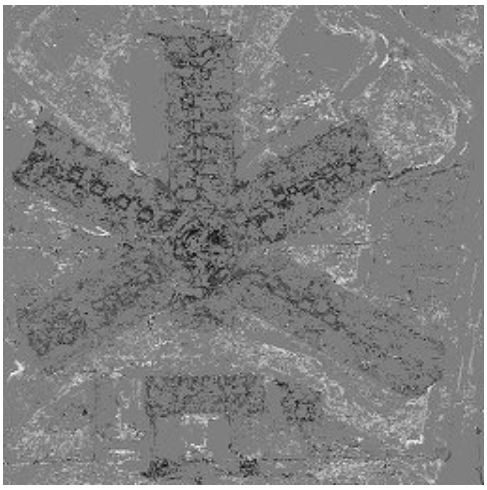
Cuadro 4.1: Comparación de resultados no densos de LNA y MARC

Como se muestra en la tabla anterior, los estimadores dan una amplia superioridad de MARC sobre LNA, por lo que si se interpreta estos resultados de una forma bastante superficial, se puede tender a rechazar el modelo propuesto en este capítulo.
Al dividir el mapa devuelto por LNA en dos mapas, uno formado por los puntos que tengan el error absoluto menor a uno ($L1 < 1$) y el otro que lo tengan mayor o igual a 1 ($L1 \geq 1$), que se llamaran LNA1 y LNA2 respectivamente, se obtiene la siguiente tabla de errores.

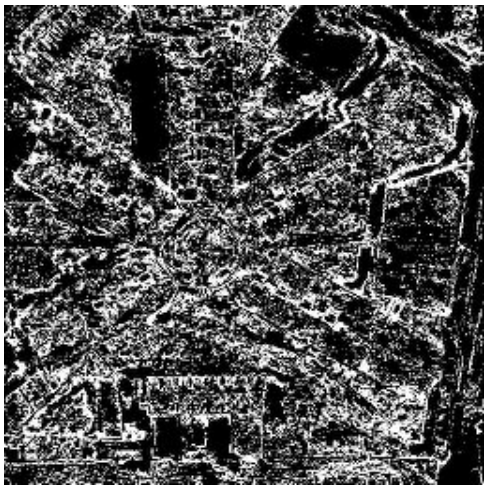
Estimador	LNA1	LNA2	LNA	MARC
densidad	18.45 %	4.37 %	22.84 %	15.15 %
L1	0.43	1.58	0.65	0.44
L2	0.25	2.82	0.74	0.28
Lp	0.25	1.06	0.43	0.21
Linf	0.99	4.89	4.89	3.46

Cuadro 4.2: Comparación de los resultados obtenidos con LNA y MARC

De la tabla anterior se puede ver como se distribuyen los errores dentro de LNA, un 81 % de las disparidades calculadas por LNA (que son los que pertenecen a LNA1) tienen una muy buena precisión ya que el resultado de los estimadores son un poco mejor (excepto Lp) a los de MARC. En cambio, el restante 19 % de disparidades calculadas por LNA (que son las que pertenecen a LNA2), tienen una precisión bastante mala que hacen que el promedio general de LNA sea bastante impreciso. Es por esto que para hacer una mejor comparación de los algoritmos MARC y LNA, en la subsección siguiente se compara los mapas interpolados para saber como el 19 % de las distancias imprecisas afectan al restante 81 % en el resultado final.

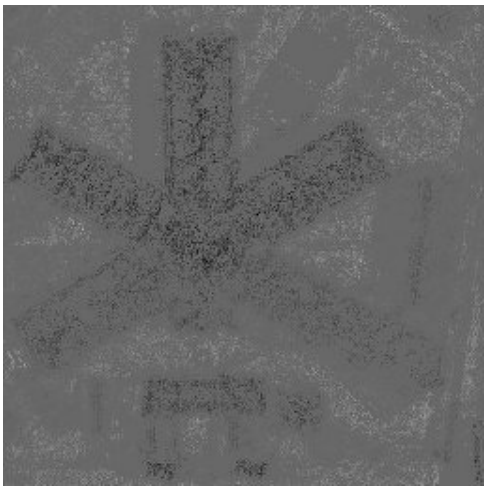


(a) Mapa de Disparidad

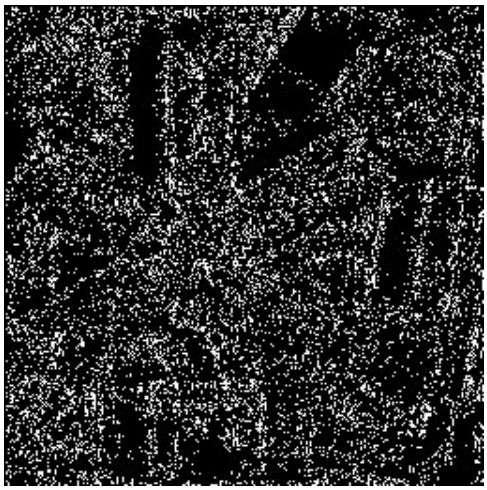


(b) Máscara (los puntos blancos indican los válidos)

Figura 4.10: Imágenes devueltas por LNA



(a) Mapa de Disparidad

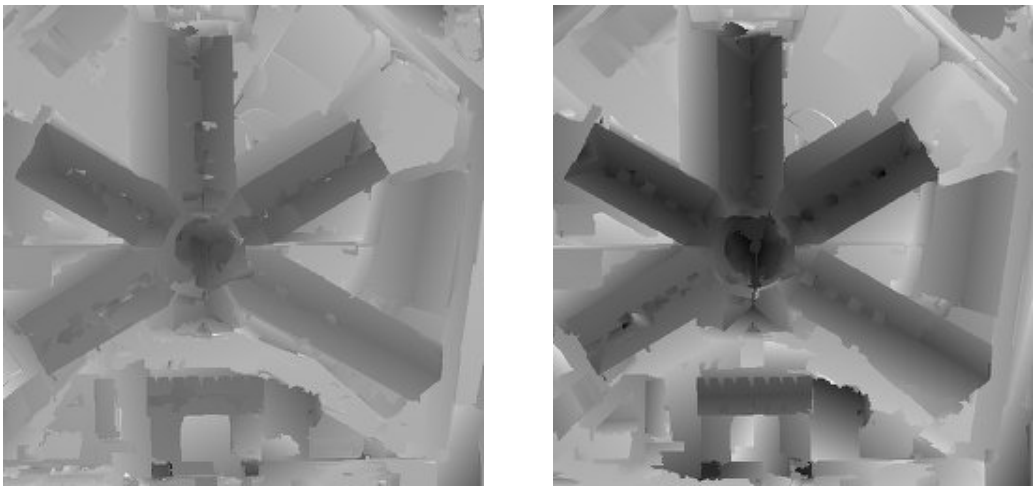


(b) Máscara (los puntos blancos indican los válidos)

Figura 4.11: Imágenes devueltas por MARC

4.3.3. Resultados interpolados

En esta parte se comparan los resultados de LNA y MARC, interpolándolos con los algoritmos Merging y Diffuse Anisotropic presentados en [1, 3]. Estos algoritmos reciben como entrada un mapa de disparidad no denso y devuelven uno denso. Los mapas LNA y MARC interpolados con Merging se pueden ver en las figuras 4.12(a) y 4.12(b).



(a) LNA-Merging (b) MARC-Merging

Figura 4.12: Interpolación de los mapas LNA y MARC usando Merging

Al contrario de los resultados no densos, las versiones interpoladas de LNA y MARC tienen casi la misma precisión como se puede ver en las siguientes tablas, donde el Cuadro 4.3 usa el 100 % de los puntos de la imagen original, y el Cuadro 4.4 usa solo lo los puntos válidos de la máscara Shadow (figura 4.9(d)).

Estimador	LNA-Merging	MARC-Merging
L1	0.435	0.435
L2	0.261	0.272
Lp	0.237	0.246
Linf	3.919	2.689

Cuadro 4.3: Errores de LNA-Merging y MARC-Merging usando el 100 % de los puntos

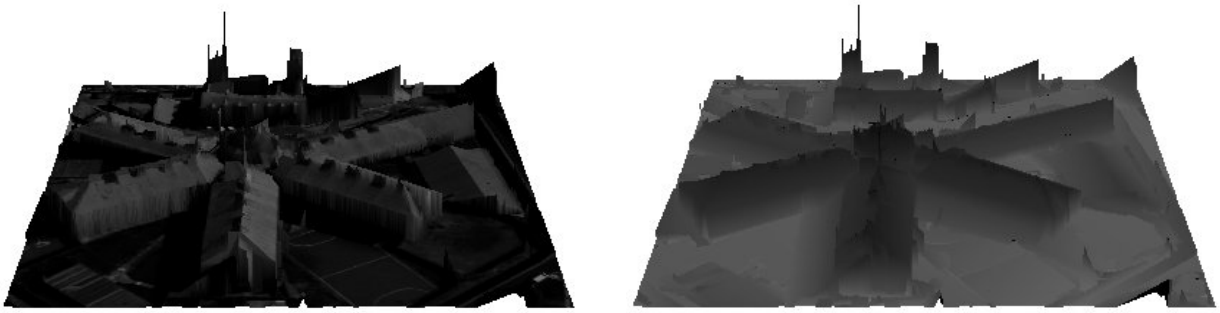
Estimador	LNA-Merging	MARC-Merging
L1	0.420	0.419
L2	0.234	0.229
Lp	0.164	0.160
Linf	3.919	2.689

Cuadro 4.4: Errores de LNA-Merging y MARC-Merging, usando los puntos válidos de Shadow (figura 4.9(d))

Otro resultado importante es que los errores en LNA disminuyeron más de un 50 % con su versión interpolada (ej. L2 bajó de 0.74 a 0.234), esto se puede ver mejor en el Cuadro 4.5. Por otro lado en las figuras 4.13 y 4.14 se muestran las representaciones 3D de la escena usando las alturas calculadas con LNA-Merging y MARC-Merging.

Estimador	LNA	LNA-Merging	MARC-Merging
densidad	22.83 %	100 %	100 %
L1	0.65	0.435	0.435
L2	0.74	0.261	0.272
Lp	0.43	0.237	0.246
Linf	4.89	3.919	2.689

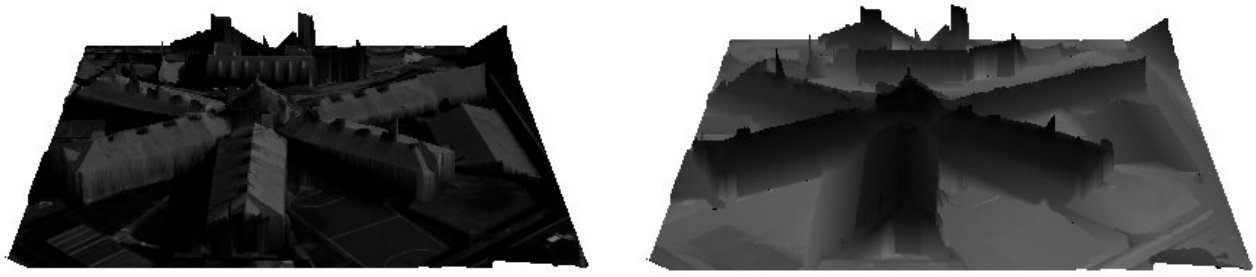
Cuadro 4.5: Comparación de LNA, LNA-Merging y MARC-Merging



(a) Usando como textura la imagen original

(b) Usando como textura el mapa de disparidad de LNA-Merging

Figura 4.13: Representación 3D con las alturas calculadas por LNA-Merging

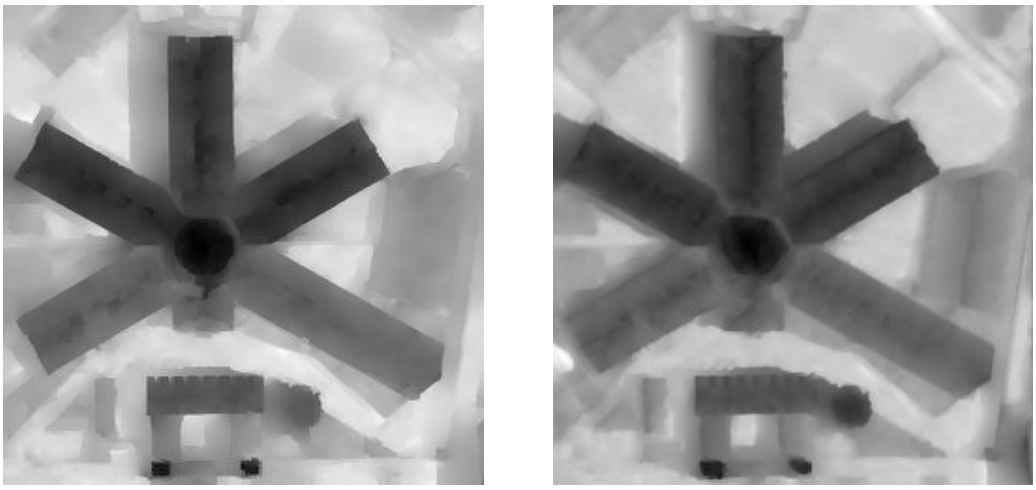


(a) Usando como textura la imagen original

(b) Usando como textura el mapa de disparidad de MARC-Merging

Figura 4.14: Representación 3D con las alturas calculadas por MARC-Merging

Con Diffuse Anisotropic al igual que Merging, el error en LNA baja más de un 50 % haciendo que las precisiones entre las versiones interpoladas de LNA y MARC sean muy parecidas. Además que en estas versiones se obtiene una mejor precisión que las calculadas con Merging. Los mapas de disparidad interpolados con Diffuse Anisotropic, se muestran en las figuras 4.15(a) y 4.15(b), y la representación 3D usando las alturas calculadas de estos mapas se pueden ver en las figuras 4.16 y 4.17.



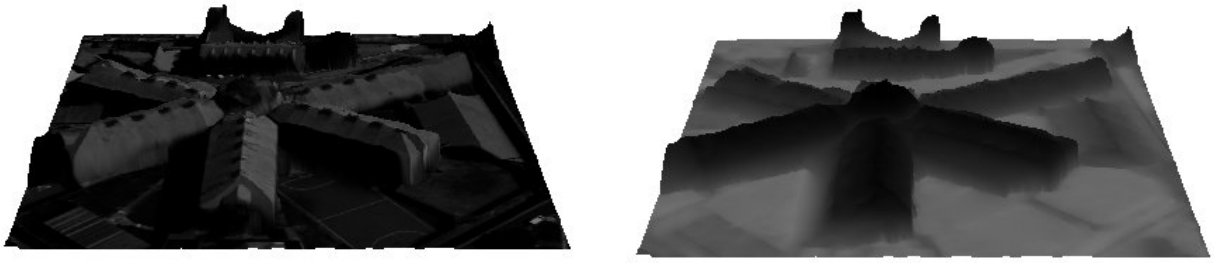
(a) LNA-Diffuse Anisotropic

(b) MARC-Diffuse Anisotropic

Figura 4.15: Imágenes de LNA y MARC interpoladas con Diffuse Anisotropic

(a) Usando como textura la imagen original (b) Usando como textura el mapa de disparidad

Figura 4.16: Representación 3D con las alturas calculadas por LNA-Diffuse Anisotropic



(a) Usando como textura la imagen original (b) Usando como textura el mapa de disparidad

Figura 4.17: Representación 3D con las alturas calculadas por MARC-Diffuse Anisotropic

Los errores estimados con estos mapas, usando todos los puntos se muestran en el Cuadro 4.6 y usando solo los puntos válidos de la máscara Shadow se muestran en 4.7.

Estimador	LNA-Diffuse Anisotropic	MARC-Diffuse Anisotropic
L1	0.402	0.437
L2	0.216	0.268
Lp	0.205	0.256
Linf	2.52	2.569

Cuadro 4.6: Resultados de NLA-Diffuse Anisotropic y MARC-Diffuse Anisotropic usando el 100 % de los puntos

Estimador	LNA-Diffuse Anisotropic	MARC-Diffuse Anisotropic
L1	0.404	0.407
L2	0.207	0.203
Lp	0.151	0.157
Linf	2.519	2.540

Cuadro 4.7: Resultados de LNA-Diffuse Anisotropic y MARC-Diffuse Anisotropic usando los puntos válidos de la máscara que no calcula los puntos de las sombras de la imagen (figura 4.9(d))

De las cuatros combinaciones (LNA-Merging, MARC-Merging, LNA-Diffuse Anisotropic, y MARC-Diffuse Anisotropic) con la que se ha obtenido una mejor precisión en la mayoría de los estimadores ha sido LNA-Diffuse Anisotropic, y luego MARC-Diffuse Anisotropic. Esto se puede ver mejor en los Cuadros 4.8 y 4.9

Estimador	LNA-Merging	MARC-Merging	LNA-Dif. Aniss.	MARC-Dif. Aniss.
L1	0.435	0.435	**0.402**	0.437
L2	0.261	0.272	**0.216**	0.268
Lp	0.237	0.246	**0.205**	0.256
Linf	3.919	2.689	**2.520**	2.569

Cuadro 4.8: Comparación final usando el 100 % de los puntos

Estimador	LNA-Merging	MARC-Merging	LNA-Dif. Aniss.	MARC-Dif. Aniss.
L1	0.420	0.419	**0.404**	0.407
L2	0.234	0.229	0.207	**0.203**
Lp	0.164	0.160	**0.151**	0.157
Linf	3.919	2.689	**2.519**	2.540

Cuadro 4.9: Comparación final usando los puntos válidos de la máscara Shadow (figura 4.9(d))

Capítulo 5

Conclusiones y Trabajo Futuro

Como primera parte del proyecto de grado se desarrolló una aplicación (llamada Icmezu) para trabajar con algoritmos estereoscópicos en el cálculo de mapas de elevación de zonas urbanas. La misma está enfocada en el uso de pares estéreo con un baseline pequeño y permite calcular mapas de disparidad subpixel con los algoritmos presentados e implementados en [3]. Otra característica importante es que genera a partir de un par estéreo y un mapa de disparidad, nuevos pares que simulan una variación en la altura de las cámaras. También entre otras cosas se pueden visualizar las imágenes (pares estéreo, ground-truth, mapas de disparidad, simulaciones, anaglifos y autocoherencias), estimar errores en los mapas de disparidad, incorporar dinámicamente nuevos algoritmos estereoscópicos y ver los escenarios 3D construidos con las alturas calculadas.
Además de cumplir con las funcionalidades esperadas, la aplicación fue presentada por Andrés Almansa a los potenciales usuarios (investigadores del CNES), y actualmente se está comenzando un nuevo proyecto para continuar con el desarrollo de Icmezu. El propósito de esto último es realizar mejoras y objetivos que no pudieron ser abarcados en el proyecto de grado por razones de tiempo, siendo los siguientes algunos de los puntos en que se va a trabajar en un futuro:

- Es bastante importante que se pueda usar imágenes muy grandes y por el momento Icmezu no funciona con imágenes de resolución mayor a 1200x1200 pixeles. El problema actual es que se carga completamente en la memoria RAM una imagen cuando se la quiere visualizar. Aunque solo sea una región de ésta la que pueda ser observada por el usuario, en caso que la imagen sea mayor que la pantalla o el visor. De esta forma, para poder visualizar imágenes de gran tamaño y que la memoria RAM no sea una limitante, se tendría que incorporar una librería (ej. OpenJpeg, Kakadu, DjVu, etc.) que permita cargar en memoria solo las zonas de la imagen que están siendo visualizadas por el usuario. La idea es que a medida que el usuario hace las operaciones de zoom y panning, se carguen y liberen las zonas necesarias para realizar dichas operaciones.
- Estudiar alternativas en el algoritmo de simulación de alturas, a causa que su

implementación actual genera artefactos en las zonas de desoclusión y su perfomance es bastante limitada.

- Mejorar la interfaz gráfica para que sea más ergonómica al estilo del Google Earth.

Como segunda parte se evaluó un nuevo algoritmo estereoscópico basado en las líneas de nivel afines (LNA). Usando un par estéreo real se comparó los resultados de LNA con los de otro algoritmo llamado MARC, siendo éste último uno de los mejores de la bibliografía consultada. Ambos algoritmos difieren en las técnicas usadas pero tienen en común que calculan mapas de disparidad subpixel y no densos.
Los resultados obtenidos con las interpolaciones de los mapas de LNA y MARC fueron muy parecidos y esto nos permite concluir que el modelo propuesto es válido.
No obstante, en los mapas sin interpolar no se obtuvo un desempeño similar. Esto se debe a que dentro de las disparidades calculadas por LNA se encuentran falsas detecciones que afectan negativamente en el promedio general del error (pero que luego éstas son eliminadas por los algoritmos de interpolación usados).
A continuación se describen posibles extensiones a hacer en un futuro:

- Investigar las causas de las falsas detecciones, que pude ser tanto un bug en la implementación como una incompletitud en el modelo propuesto.

- La implementación actual del algoritmo usa líneas de nivel interpoladas con el método bilineal. Es posible que interpolaciones de mayor complejidad mejoren el desempeño por lo que se podría tomar en cuenta en futuras implementaciones, el uso de líneas de nivel con interpolación sinc. Además éstas se pueden aproximar haciendo un fftzoom de un factor dos o tres, y luego calculando las líneas de nivel con interpolación bilineal.

- Mejorar la perfomance para poder usar imágenes grandes debido a que esto no fue abarcado en el trabajo realizado en este proyecto de grado. Uno de los problemas principales es que si la imagen es muy grande podría no alcanzar la memoria disponible para trabajar con todas las líneas de nivel a la vez.

Capítulo 6

Glosario

Algoritmo Estereoscópico: Es un algoritmo que calcula mapas de disparidad a partir de un par de imágenes en estéreo.

Anaglifo: Es una superposición de dos imágenes que produce al ser miradas con lentes especiales una sensación de relieve.

Baseline: Distancia entre las cámaras del par estéreo.

Configuración en Paralelo: Una configuración de cámaras en paralelo, es cuando los dos planos retinales (donde se forman las imágenes) están desplazados horizontalmente y las cámaras tienen la misma distancia focal (el par de ojos de una persona es un ejemplo clásico de un par de cámaras con una configuración en paralelo).

Conjuntos de Nivel: Conjuntos que permiten descomponer de forma reversible la estructura de una imagen, éstos se definen usando umbrales sobre la luminosidad de la imagen.

Disparidad: Cuando se observa un objeto tridimensional desde dos puntos de vistas distintos, se le llama disparidad al desplazamiento relativo del objeto entre cada punto de vista. Si los puntos de vista tienen una configuración en paralelo, el desplazamiento del objeto es horizontal.

Ground-Truth: Es un mapa de disparidad con las disparidades reales y no con las estimadas por algún algoritmo estereoscópico.

Imagen de Autocoherencia: Es un anaglifo generado con dos imágenes de referencia, una es la real y la otra es una simulación que se crea usando la imagen secundaria y un mapa de disparidad.

Imagen de Referencia: Es la imagen del par estéreo a la que se le calcula las disparidades.

Imagen Secundaria: Es la imagen del par estéreo que se usa para buscar las disparidades de la imagen de referencia.

Líneas de Nivel: Las líneas de nivel de una imagen, son las fronteras o curvas de Jordan, que delimitan los conjuntos de nivel.

Mapa de Disparidad: Es una imagen que mediante los valores de sus pixeles se indica la disparidad estimada para cada unos de ellos.

Mapa de Elevación: Es una imagen donde los valores de sus pixels representan la altura tridimensional de los mismos. Un mapa de elevación se obtiene usando el principio estereoscópico y su mapa de disparidad correspondiente.

Par Estéreo: Es un par de imágenes de una misma escena tridimensional (en el contexto de este proyecto la escena es una zona urbana), sacada desde dos puntos de vistas distintos.

Principio Estereoscópico: El principio estereoscópico es una ecuación que relaciona la disparidad de un objeto con la profundidad del mismo con respecto al observador.

Representación 3D: Es una reconstrucción tridimensional de una escena, usando un mapa de elevación.

Apéndice A

Algoritmo de simulación

En este capítulo se describe el algoritmo usado para generar pares que simulan un cambio en la altura de las cámaras del par original, pero manteniendo el baseline (distancia entre las cámaras) constante.
Los parámetros de entrada del algoritmo son las imágenes referencia (u) y secundaria ($\widetilde{u}$), el ground-truth (ϵ) del par estéreo y la disminución k de la altura a simular.
Se supone que la imagen de referencia ha sido sacada desde el cenit, esto conlleva que en la simulación no se vea alterada. Pero en cambio como la imagen secundaria no ha sido tomada desde esa posición se va ver afectada por el cambio de altura.
Este algoritmo se basa en el método de simulación presentado en [3] y en la relación entre las disparidades de pares de distintas alturas.

A.1. Relación entre las disparidades de pares de distintas alturas

Si se tiene un par estéreo a una altura H con respecto al suelo, la relación entre la disparidad d y la altura h de un punto dado en la imagen de referencia, se puede expresar [2] de la forma:

$$d[pixels] = \frac{B}{H}\frac{1}{r_o}h[metros] \tag{A.1}$$

siendo B el baseline y r_o[metros/pixel] el tamaño de un pixel proyectado en el suelo.
Como h es constante (debido a que las alturas de los objetos reales es constante independientemente de la posición de las cámaras), y r_o también lo es (si se usa la misma cámara). Al simular una disminución k en la altura H (H/k) manteniendo el mismo B, se obtiene la siguiente relación para la disparidad $d^{'}$ de un punto simulado.

$$d^{'}[pixels] = \frac{B}{H/k}\frac{1}{r_o}h[metros] \tag{A.2}$$

Usando las ecuaciones A.1 y A.2, se obtiene la siguiente relación entre la disparidad de un punto en el par estéreo real y en el par simulado:

$$d^{'} = dk \tag{A.3}$$

De lo anterior se deduce fácilmente que si se tiene un ground-truth ϵ de un par estéreo a una altura H, entonces el ground-truth ϵ_k del mismo par a la altura H/k se puede obtener mediante la siguiente ecuación:

$$\epsilon_k(x) = k\epsilon(x) \tag{A.4}$$

A.2. Simulación usando un muestreo irregular

Cuando se tiene un par estéreo con un baseline pequeño, se puede usar el modelo propuesto en [15, 16], en donde cada pixel de la imagen de referencia u puede ser macheado a pixeles de la imagen secundaria $\widetilde{u}$ sobre la línea epipolar a una distancia ϵ (disparidad):

$$\widetilde{u}(x + \epsilon(x)) = u(x) \tag{A.5}$$

Si $\widetilde{u}_k$ simula una disminución k en la altura de la imagen secundaria, usando las ecuaciones A.4 y A.5 se tiene la siguiente relación.

$$\widetilde{u}_k(x + \epsilon_k(x)) = u(x) \ \ , \text{ siendo } \epsilon_k(x) = k.\epsilon(x) \tag{A.6}$$

Si se le llama Φ a la función de mapeo $\Phi = x + \epsilon_k(x)$, entonces se puede reconstruir $\widetilde{u}_k$ de la siguiente forma

$$\widetilde{u}_k(x) = u(\Phi^{-1}(x)) \tag{A.7}$$

La relación anterior no es del todo cierta, debido a que la ecuación A.5 se basa en la suposición no realista, de que el terreno se puede modelar con una función suave y no hay áreas ocluidas. Además en nuestro caso esto se ve agravado, porque a medida que k crece las disparidades también lo hacen, aumentando de esta manera las oclusiones.
En sí van a ver dos tipos de estos fenómenos que va a ser necesario simular:

Oclusiones: Son los puntos que se ven en la imagen de referencia u pero no en la secundaria $\widetilde{u}_k$. Esto sucede cuando Φ es decreciente[1] ($\epsilon_k^{'}(x) \leq -1$).

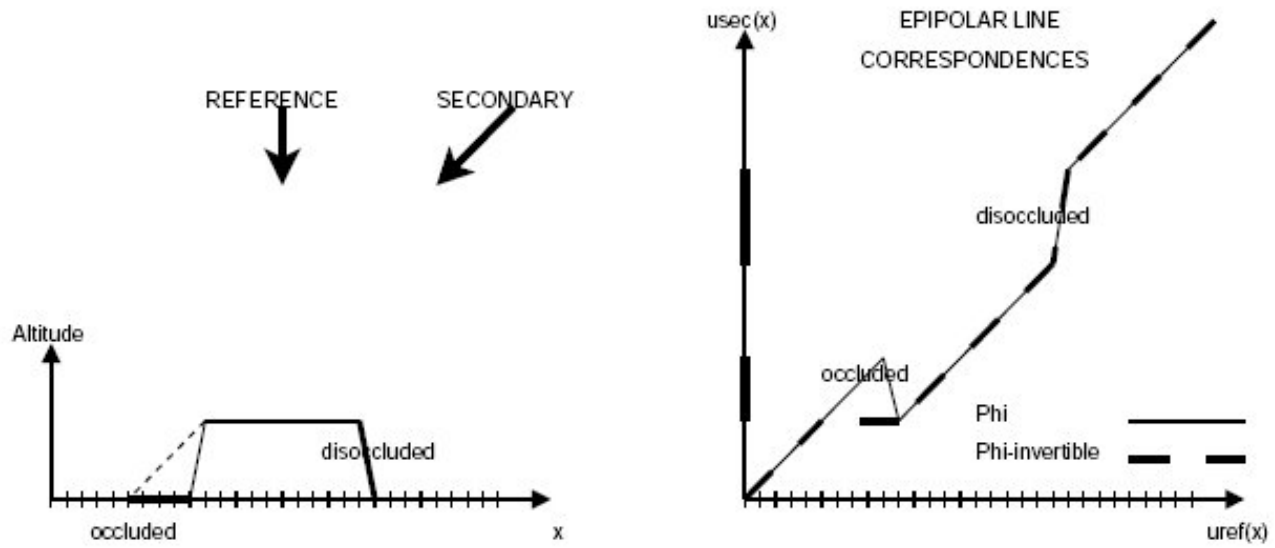
Desoclusión: Si $|\Phi^{'}| > 1$ (si $\epsilon_k^{'}(x) \geq 0$) la frecuencia de sampleo de la imagen secundaria $\widetilde{u}_k$ es mayor que la de la imagen de referencia. Esto significa que en la imagen secundaria hay más detalles representados que en la imagen de referencia. El problema es que cuando se simula u_k desde u hay información que no está en u que es necesaria para hacer la simulación.

[1] Φ es decreciente en $[a, b]$ si $\Phi^{'}(x) < 0 \ \forall x \in (a, b)$

Simulación de Oclusiones: Para generar $\widetilde{u}_k$ a partir de u usando $\widetilde{u}_k(x) = u(\Phi^{-1}(x))$, es necesario calcular la inversa de Φ. En los puntos de oclusión Φ no es invertible debido a que ésta es decreciente. Además que los puntos de oclusión ocurren cuando un punto en la imagen secundaria se corresponde con más de uno en la de referencia. Entonces para simular los puntos de oclusión (y obtener una "inversa" de Φ), en las áreas donde la inversa tenga múltiples valores funcionales, se queda solo con el punto de valor más alto (figura A.2). Definiéndose de esta manera una pseudo-inversa como la siguiente:

$$\Phi^{+}(y) = \max_{x}(\Phi(x) = y) \tag{A.8}$$

La forma en que está implementada la pseudo-inversa, es haciendo una interpolación lineal de Φ. Para construir una lista de intervalos $[\Phi(x), \Phi(x+1)]$ con los valores funcionales conocidos de Φ. Finalmente para obtener el valor de la inversa en un entero i se queda con el t tal que $\Phi(t) = i$ siendo t el mayor de todos los valores que cumplan la condición anterior.



Simulación de Desoclusiones: En la versión original del algoritmo presentado en [3], para simular las desoclusiones se sustituyen estas zonas en la imagen secundaria simulada ($\widetilde{u}_k$), por los puntos de la imagen secundara real $\widetilde{u}$. En la versión de este capítulo no se puede hacer esto porque las imágenes secundaria real y simulada ($\widetilde{u}$ y $\widetilde{u}_k$) tienen distintos ground-truth (ϵ y ϵ_k).
Quedando las siguientes alternativas distintas para simular las desoclusiones:

1. Usar ruido blanco para rellenar los puntos de desoclusión.
2. Cada intervalo horizontal de desoclusión sustituirlo por el próximo intervalo (izquierdo ó derecho) que no tenga puntos de desoclusión.
3. Cada intervalo horizontal de desoclusión sustituirlo por una interpolación lineal usando los dos pixeles adyacentes que no son de desoclusión.

De las tres alternativas anteriores, la primera es la mejor que se adapta para ejecutar algoritmos de correlación en el par simulado, a causa que éstos no detectan macheos en las zonas de ruido blanco.
En cambio, las otras dos alternativas ofrecen una mejor apariencia en la textura de la imagen simulada (por la ausencia de ruido blanco). Pero con el costo de presentar información falsa en esta imagen, especialmente la segunda que con la copia de los intervalos puede afectar negativamente en la perfomance de los algoritmos de correlación. La interpolación de los pixeles adyacentes permite que la simulación se aproxime mejor a la realidad, además que no influye tan negativamente en los algoritmos de correlación como la anterior.

Soporte Espectral Una vez calculada la pseudo-inversa, se quiere reconstruir $\widetilde{u}_k$, usando $u(\Phi^{-1}(x)), x \in Z^2$, esto hace que sea necesario interpolar u en la grilla irregular $\Phi^{-1}(Z)$. Para evitar que en el sampleo irregular las altas frecuencias de la imagen sobrepasen al soporte espectral de la misma, se hace un zoom a la imagen de referencia usando Interpolación de Fourier (zero-padding), y luego del sampleo irregular, para hacer el subsampleo se reduce el espectro con un filtro prolate para evitar el aliasing.

A.3. Simulación de la imagen de referencia

Siendo ϵ el mapa de disparidad de las imágenes de referencia u y secundaria $\widetilde{u}$.
Usando la ecuación A.7 ($\widetilde{u}(x + \epsilon(x)) = u(x)$), se puede generar una imagen de referencia simulada (u_{sim}), haciendo un resampleo sobre la grilla irregular $u_{sim} = \widetilde{u}(\Phi(x))$, con $\Phi(x) = x + \epsilon$.

A.4. Algoritmo Final

Resumiendo lo anterior se muestran los pasos del algoritmo, que recibe como parámetros de entrada la imagen de referencia (que puede ser la real u o la simulada u_{sim}), el ground-truth ϵ y la disminución $1/k$ de la altura a simular.

1. Se le aplica un zoom 2x a la imagen de referencia usando la interpolación de fourier (zero-padding), y se calcula el ground-truth $\epsilon_k = k\epsilon$ para hacerle zoom mediante una interpolación spiline de primer orden.

2. Se calcula la pseudo-inversa de $\Phi(x) = x + \epsilon_k(x)$ (usando pseudoinverse desarrollada por Gabriele Facciolo), y se obtienen las áreas ocluidas $(\Phi(x)^{'} \leq 0)$y desocluidas $(\Phi(x)^{'} > 2)$.

3. Se resamplea la imagen de referencia acorde a la grilla irregular $\widetilde{u}_k = u(\Phi^{-1}(x))$ (usando nfft_test desarrollada por Gloria Haro).

4. Se sustituyen los puntos de desoclusión, usando algunas de las técnicas comentadas anteriormente.

5. Por último se reduce el espectro con un filtro prolate a las imágenes $\widetilde{u}_k$ y u, para luego hacer un subsampleo en ambas imágenes y finalmente agregarle ruido gaussiano.

Apéndice B

Regresión Lineal

En el algoritmo presentado en el capítulo 4, se usa la regresión lineal para calcular la recta $\hat{X} = a + bY$ que mejor se aproxima a una curva de puntos $(x_1, y_1), (x_2, y_2), ...(x_n, y_n)$, minimizando la suma de los errores al cuadrado de las cantidades $x_i - \hat{x}_i$. De esta forma, se hallan los coeficientes a y b de la recta $\hat{X} = a + bY$ que minimicen la siguiente sumatoria:

$$\sum_{i=1}^{N} (x_i - \hat{x}(y_i))^2 = \sum_{i=1}^{N} (x_i - \hat{x}_i)^2 = \sum_{i=1}^{N} e_i^2$$

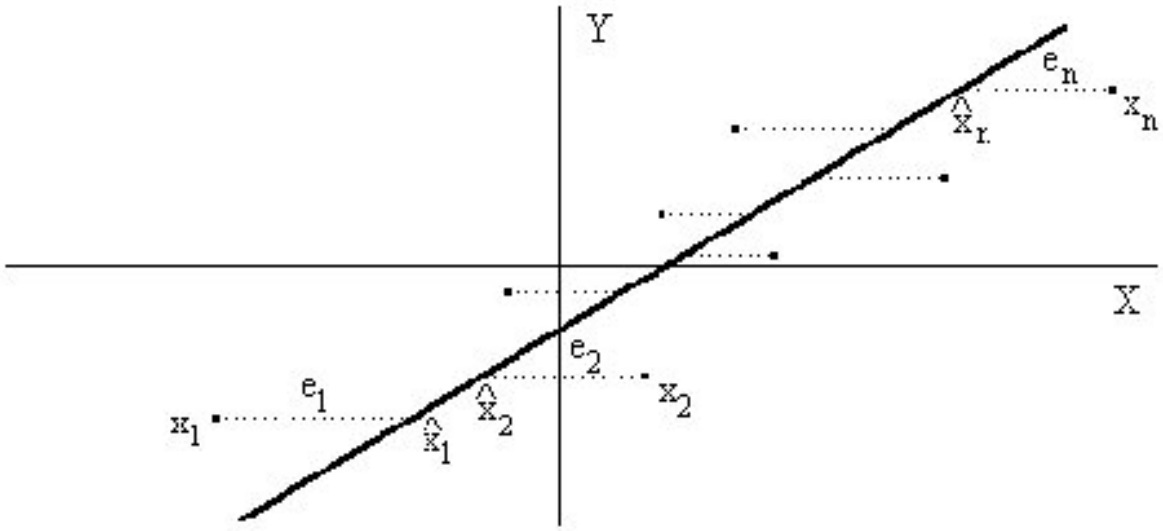


Figura B.1: Los errores a minimizar son las cantidades $e_i = (x_i - \hat{x}_i)^2$

Para hacer lo anterior se define la función $E_{error}(a, b) = \sum_{i=1}^{N} (x_i - \hat{x}_i)^2 = (x_i - a - by_i)^2$, y se

obtiene el mínimo mediante sus derivadas parciales de la siguiente manera:

$$(a,b) \text{ minimizan } E_{error}(a,b) \iff \begin{cases} \frac{E_{error}}{\delta a}(a,b) &= 0 \\ \frac{E_{error}}{\delta b}(a,b) &= 0 \end{cases} \tag{B.1}$$

$$\iff \begin{cases} \frac{E_{error}}{\delta a}(a,b) &= -2\sum_{i=1}^{N}(x_i - a - by_i) &= 0 \\ \frac{E_{error}}{\delta b}(a,b) &= -2\sum_{i=1}^{N}(x_i - a - by_i)y_i &= 0 \end{cases} \tag{B.2}$$

Usando la primera condición de la ecuación B.2 se despeja el coeficiente a:

$$\sum_{i=1}^{N}(x_i - a - by_i) = 0 \iff \sum_{i=1}^{N} x_i - aN - b\sum_{i=1}^{N} y_i = 0 \iff a = \bar{x} - b\bar{y} \tag{B.3}$$

Sustituyendo el resultado anterior en la segunda condición de la ecuación B.2 se obtiene el coeficiente b:

$$\sum_{i=1}^{N}(x_i - a - by_i)y_i = 0 \iff \sum_{i=1}^{N} x_i y_i - a\sum_{i=1}^{N} y_i - b\sum_{i=1}^{N} y_i^2 = 0 \iff$$

$$\sum_{i=1}^{N} x_i y_i - (\bar{x} - b\bar{y})\sum_{i=1}^{N} y_i - b\sum_{i=1}^{N} y_i^2 = 0 \iff \frac{1}{N}\sum_{i=1}^{N} x_i y_i - (\bar{x} - b\bar{y})\bar{y} - b\frac{1}{N}\sum_{i=1}^{N} y_i^2 = 0$$

$$\iff S_{XY} - bS_Y^2 = 0 \iff b = \frac{S_{XY}}{S_Y^2}$$

siendo,

$$\begin{aligned} \bar{x} &= \frac{1}{N}\sum_{i=1}^{N} x_i \\ \bar{y} &= \frac{1}{N}\sum_{i=1}^{N} y_i \\ S_{XY} &= \frac{1}{N}\sum_{i=1}^{N} x_i y_i - \bar{x}\bar{y} \\ S_Y^2 &= \frac{1}{N}\sum_{i=1}^{N} y_i^2 - \bar{y}^2 \end{aligned}$$

En resumen, los coeficientes de la recta $\hat{X} = a + bY$ que mejor se aproxima a la curva de puntos $(x_1, y_1), (x_2, y_2), ...(x_n, y_n)$ se calculan de la siguiente manera:

$$\begin{aligned} b &= \frac{S_{XY}}{S_Y^2} \\ a &= \bar{x} - b\bar{y} \end{aligned}$$

Apéndice C

Incorporación de Algoritmos

Para incorporar nuevos algoritmos en ICMEZU, el usuario tiene que hacer la siguiente secuencia de pasos que se detallan en las secciones siguientes.

1. Crear una librería dinámica (con el comando ld), que tenga una función implementada en C que encapsule el algoritmo.
2. Especificar en un archivo xml cuales son los nombres, tipos y valores por defecto de los parámetros de entrada del algoritmo.
3. Registrar el xml en ICMEZU.

C.1. Crear la Librería Dinámica

La librería es necesaria para que ICMEZU pueda ejecutar dinámicamente los algoritmos estéreos sin tener que compilar la misma.
Para crear la librería el algoritmo tiene que estar encapsulado en una función implementada en C, y que cumpla con la siguiente interfaz.

```
typedef struct Parametros_IN_
        {
           int num_parametros;
           char ** valor_parametros;
        } * Parametros_IN;


void nombre_del_algoritmo(Fimage ref, Fimage sec, Parametros_IN entrada,
                          Fimage *mapa, Fimage *mask)
```

- ref: Imagen de referencia en formato Fimage.
- sec: Imagen secundaria en formato Fimage.
- entrada: Parámetros de entradas adicionales, que se ingresan en una array con tope del tipo ParametrosIN. Que está formada por la cantidad de parámetros de entrada (Parámetros_IN.num_parametros), y una lista de string (char **) que son los valores de dichos parámetros de entrada.
- mapa: Mapa de disparidad en formato Fimage calculado por el algoritmo.
- mapa: Máscara que indica los puntos válidos (VALIDOS=1) del mapa devuelto (puede ser NULL en caso que sea un mapa denso).

Para crear la librería se usan los siguientes comandos:

- gcc -c nombre_algoritmo.c
- ld -o lib_algoritmo.so nombre_algoritmo.o FLAG_LIBS -shared

FLAG_LIBS = es una lista de las librerías que depende nombre_algoritmo.o

A continuación se muestra un ejemplo del uso de los comandos anteriores para crear una librería con un algoritmo que depende de MegaWave2.

- gcc -c algoritmo.c -I$(MEGAWAVE2)/kernel/lib/include
- ld -o algoritmo.o $(MEGAWAVE2)/lib/ix86/libmw.a -L/usr/X11R6/lib -lm -ltiff -ljpeg -lX11 -L$(MEGAWAVE2)/kernel_obj/lib/lib/ix86 -L$(MEGAWAVE2)/sys/lib/ix86 -lsysmw -lW_X11R4

C.2. Crear el archivo de configuración XML

El archivo XML especifica los nombres, tipos y valores por defecto de los parámetros de entrada del algoritmo y la ubicación de la librería en que se halla. Es necesario especificar los parámetros de entrada para que ICMEZU sepa qué tipo de parámetros tiene que ingresar el usuario para ejecutar el algoritmo.
La especificación en el XML se hace de la siguiente forma:

```
<Algoritmo>
        <path> PATH_LIBRERIA   </path>
        <nombre>  NOMBRE_FUN </nombre>
        <num_param>   N      </num_param>
        DEF_PARAMETRO1
        DEF_PARAMETRO2
        ...........
        DEF_PARAMETRON
 </Algoritmo>
```

- PATH_LIBRERIA = Ubicación en el file system de la librería que contiene la implementación del algoritmo.
- NOMBRE_FUN = Nombre de la función que calcula el mapa de disparidad
- N = Número que indica la cantidad de parámetros de entrada que tiene el algoritmo, sin contar el de las imágenes de referencia y secundaria
- DEF_PARAMETROX = Descripción del parámetro de entrada número X, y se define a continuación:

```
<param Valor_Defecto="VAL_DEFECTO">
<nombre>NOM_PARAM</nombre>
<tipo>TIPO</tipo>
DEF_ENUM
DEF_RANGO
</param>
```

  - VAL_DEFECTO = Valor por defecto del parámetro, o sea que va a ser el valor del parámetro que se va a usar cuando el usuario ejecute el algoritmo y no ingrese ningún valor para ese parámetro.
  - NOM_PARAM = Nombre del parámetro o la descripción que se le muestra al usuario para que ingrese un valor.

- TIPO = Indica el tipo de parámetro que puede ser INT FLOAT o CHAR. Esto sirve para hacer un chequeo de los parámetros ingresados por el usuario antes de ejecutar el algoritmo. Por ejemplo si el parámetro es del tipo INT y el usuario ingresa el valor "s", ICMEZU le indicará que el valor tiene que ser un entero y el algoritmo no se ejecuta hasta que el usuario lo corrija.
- DEF_ENUM = Se usa para definir los parámetros que son de tipo enumerado, o sea que están formado por una lista de valores válidos. En caso que el parámetro no sea un enumerado DEF_ENUM se sustituye por

  ```
  <enum Cantidad ="0"/>
  ```

  si se quiere usar la lista de valores válidos hay que definirla de la siguiente forma:

  ```
  <enum Cantidad ="N">
      <enumi> VAL_1 </enumi>
      <enumi> VAL_2 </enumi>
      ..........
      <enumi> VAL_N </enumi>
  </enum>
  ```

  - N = es la cantidad de valores enumerados.
  - VAL_I = es el valor I de la lista de enumerados.
- DEF_RANGO = se usa para acotar el valor de un parámetro superior e inferior, por ejemplo cuando el valor válido de un parámetro es un entero entre 1 y 10. Si el parámetro no tiene rango DEF_RANGO se define de la siguiente forma:

  ```
  <rango/>
  ```

  En caso que se quiera poner un rango se define de la siguiente forma:

  ```
  <rango Inf="COT\_INF" Sup="COT\_SUP" />
  ```

  COT_INF = es la cota inferior del parámetro.
  COT_SUP = es la cota superior del parámetro.

A continuación se muestran un par de ejemplos:

Ejemplo 1: El algoritmo tiene las siguientes características:

- El algoritmo se encuentra en la librería /home/ejemplo/icmezu-data/algoritmos/ejemplo.so.
- El nombre de la función es calcular_mapa.
- Tiene los siguientes dos parámetros de entrada.
  - **ejemplo_enumerado** de tipo enumerado con los valores Enum1, Enum2, Enum3, Enum4, Enum5.
  - **ejemplo_rango** es un entero dentro del intervalo [0,13].

```
<Algoritmo>
  <path>/home/ejemplo/icmezu-data/algoritmos/ejemplo.so</path>
  <nombre>calcular_mapa</nombre>
  <num_param> 2  </num_param>
  <param Valor_Defecto="0.5">
        <nombre>ejemplo_enumerado</nombre>
        <tipo>CHAR</tipo>
        <enum Cantidad="5">
               <enumi> "Enum 1" </enumi>
               <enumi> "Enum 2" </enumi>
               <enumi> "Enum 3" </enumi>
               <enumi> "Enum 4" </enumi>
               <enumi> "Enum 5" </enumi>
        </enum>
        <rango/>
  </param>
  <param Valor_Defecto="2">
        <nombre>ejemplo_rango</nombre>
        <tipo>INT</tipo>
        <enum Cantidad="0"/>
         <rango Inf="0" Sup="13" />
  </param>
</Algoritmo>
```

Ejemplo 2: El algoritmo tiene las siguientes características:

- El algoritmo se encuentra en la librería /share/icmezu-data/algoritmos/marc/libmarc.so.
- El nombre de la función es calcular_disparidades.
- Tiene los siguientes seis parámetros de entrada.
  - **rapport_compare_masse** de tipo float.
  - **rapport_masse_1** de tipo float.
  - **rapport_masse_1** de tipo float.
  - **deca_max_lig** de tipo float.
  - **deca_max_col** de tipo float.
  - **nb_filtre_interpol_trou** de tipo float.

```
<Algoritmo>
  <path>/share/icmezu-data/algoritmos/marc/libmarc.so</path>
  <nombre>calcular_disparidades</nombre>
  <num_param> 6  </num_param>
  <param Valor_Defecto="0.5">
        <nombre>rapport_compare_masse</nombre>
        <tipo>FLOAT</tipo>
        <enum Cantidad="0"/>
        <rango/>
  </param>
  <param Valor_Defecto="0.25">
        <nombre>rapport_masse_1</nombre>
        <tipo>FLOAT</tipo>
        <enum Cantidad="0"/>
        <rango/>
  </param>
  <param Valor_Defecto="0.25">
        <nombre>rapport_masse_2</nombre>
        <tipo>FLOAT</tipo>
        <enum Cantidad="0"/>
        <rango/>
  </param>
  <param Valor_Defecto="0.1885">
        <nombre>deca_max_lig</nombre>
        <tipo>FLOAT</tipo>
        <enum Cantidad="0"/>
```

```
        <rango/>
  </param>
  <param Valor_Defecto="7.4">
        <nombre>deca_max_col</nombre>
        <tipo>FLOAT</tipo>
        <enum Cantidad="0"/>
        <rango/>
  </param>
  <param Valor_Defecto="9">
        <nombre>nb_filtre_interpol_trou</nombre>
        <tipo>INT</tipo>
        <enum Cantidad="0"/>
        <rango/>
  </param>
</Algoritmo>
```

C.3. Registrar el archivo XML en ICMEZU

Una vez creada la librería y especificado los parámetros de entrada en el archivo XML, el último paso consiste simplemente en registrar el archivo XML en la aplicación haciendo lo siguiente.

1. Ejecutar ICMEZU
2. En la pantalla principal (figura C.1) seleccionar desde el menú Tool/Modules/Add..

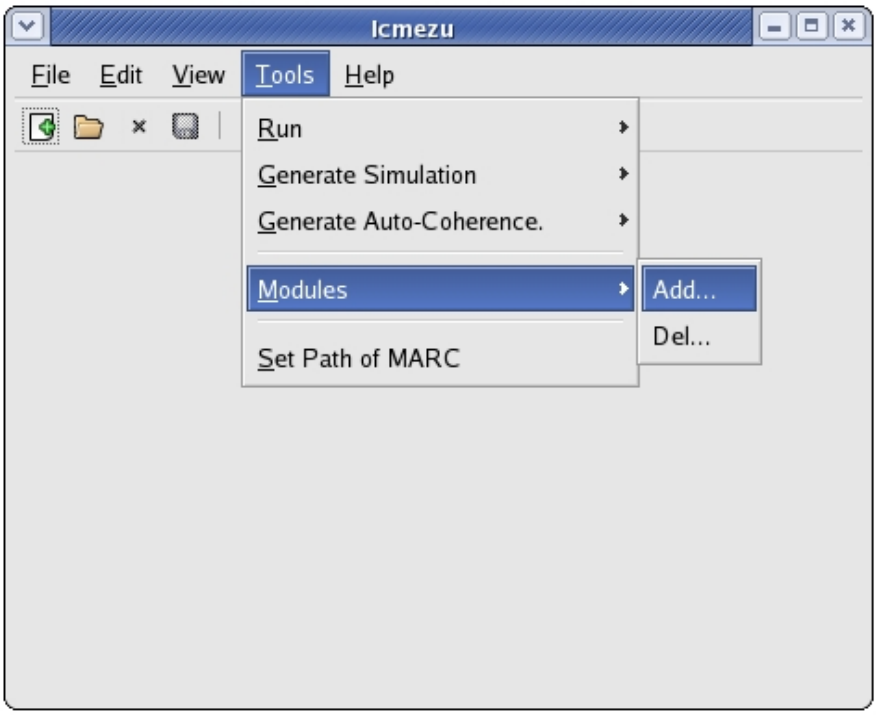


Figura C.1: Pantalla Principal

3. En el diálogo (figura C.2) especificar la ubicación del archivo XML y el nombre con el que se quiere identificar al algoritmo en la aplicación.

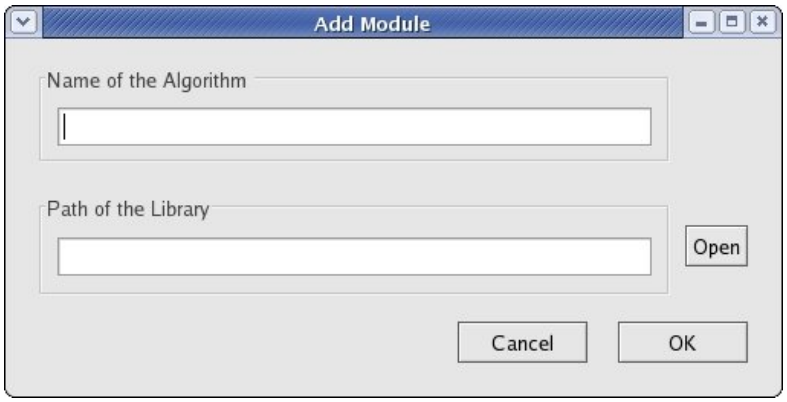


Figura C.2: Dialogo para agregar Módulo

Apéndice D

Ejecución dinámica de funciones

La llamada dinámica de funciones posibilita a ICMEZU ejecutar los algoritmos estereoscópicos incorporados por el usuario. Esto se hace mediante la librería dlfcn (implementada en C) que permite cargar dinámicamente una librería en la memoria (dlopen), para luego obtener punteros a sus funciones (dlsym) y ejecutarlas. A continuación se describe cómo se usa dlopen y dlsym para ejecutar los algoritmos agregados por el usuario, siendo los parámetros de entrada para hacer dicha operación:

- pathLib: Ubicación en el file system de la librería dinámica que tiene la implementación del algoritmo estereoscópico que se va a ejecutar.
- nomFun: Nombre de la función de la librería pathLib, que ejecuta el algoritmo.
- ref y sec: Imágenes de referencia y secundaria respectivamente
- listaString: Lista de strings, que representa los valores ingresados por el usuario para pasarle como parámetros de entrada al algoritmo.

Los parámetros de salida son

- map: Mapa de disparidad calculado
- mask: Máscara que contiene las disparidades válidas de map.

Los pasos de cómo se usan dlopen y dlsym:

1. ```
   void * l = dlopen(pathLib,RTLD_LAZY);
   ```

   Usando la función dlopen se carga la librería que tiene la implementación del algoritmo. Los parámetros de entrada de dlopen son la ubicación en el file system de la librería y el modo en que se va a usar.
   ```

2. `void (*pfn)() = dlsym(l,nomFun)`

   Se obtiene un puntero a la función que ejecuta el algoritmo, se le pasa como parámetros de entrada la librería cargada en el paso anterior y el nombre de la función.

3. `pfn(ref,sec,listaString,&map,&mask);`

   Se ejecuta el algoritmo pasándole como parámetros de entrada las imágenes de referencia y secundaria (ref y sec), los parámetros de entrada ingresados por el usuario (listaString). Como parámetros de salida se pasan el mapa de disparidad y su máscara.

4. `dlclose(l);`

   Cuando la ejecución termina, se tiene el mapa de disparidad y la máscara calculadas en los parámetros de salida (map y mask), y finalmente se libera de la memoria la librería cargada usando dclose.

Apéndice E

Generación del Escenario 3D

Para generar el escenario 3D a partir de un mapa de elevación (ME) y una imagen usada como textura (REF), se usa la librería VTK que permite a través de sus clases implementadas en C++, hacerlo de una manera bastante simple. Esto se debe a que con solo representar el escenario geométrico en un objeto del tipo VtkPolyData, VTK luego se encarga automáticamente de graficarlo en la pantalla y de manejar la interacción con el usuario (rotación, zoom y traslación de la escena). De esta forma, la parte más compleja desde el punto de vista de la implementación, es representar la escena usando la clase VtkPolyData. A continuación se describe los pasos de cómo se usan las clases principales para construir el objeto de tipo VtkPolyData.

1. Primero se obtiene para cada pixel del mapa de elevación sus coordenadas en el espacio 3D, obteniendo de esta forma las coordenadas en la estructura vtkFloatArray.

```
vtkFloatArray* pcoords = vtkFloatArray::New();
pcoords->SetNumberOfComponents(3); /*Setea la dimensión de las
                                     coordenadas*/
pcoords->SetNumberOfTuples(nx*ny); /*Setea la cantidad de coordenadas
                                     nx número de filas,
                                     ny número de columnas*/
for(i=0;i<nx;i++)
   for(j=0;j<ny;j++)
      {
       pts[0] = (float)i;
       pts[1] = (float)j;
       pts[2] = (float)ME[i][j];
       pcoords->SetTuple(k, pts); /*Para cada pixel se crea
                                    una coordenada 3D*/
       k++;
      }
```

2. Luego usando las coordenadas anteriores (pcoords) se crean un objeto del tipo vtkPoints, que representa los puntos ubicados en esas posiciones.

```
vtkPoints* points = vtkPoints::New();
points->SetData(pcoords);
```

3. Para cada par de filas i y i+1, se construye una estructura triangle strip como la de la figura E.1, que consiste en unir los puntos adyacentes mediante triángulos.

```
vtkCellArray* strips = vtkCellArray::New();
for(j=0;j<ny;j=j+1)
   {
    strips->InsertNextCell(2*nx);
    for(i=0;i<ny;i++)
     {
      strips->InsertCellPoint(j+i*nx);
      strips->InsertCellPoint((i+1)*nx+j);
     }
   }
```

Figura E.1: Triangle Strip

4. A cada punto se le asigna el color que le corresponde en la imagen usada como textura (REF).

```
vtkIntArray* newscalars = vtkIntArray::New();
newscalars->SetNumberOfTuples(nx*ny);
for(i=0;i<nx;i++)
  for(j=0;j<ny;j++)
   {
    c = (float)REF[i][j];
    newscalars->InsertComponent(k,0,c);
    k++;
   }
```

5. Finalmente, con los puntos, triangles strip y colores creados en los pasos 2, 3 y 4 respectivamente, se crea un objeto del tipo vtkPolyData que representa el escenario 3D.

```
vtkPolyData* polydata = vtkPolyData::New();
polydata->SetPoints(points);
polydata->SetStrips(strips);
polydata->GetPointData()->SetScalars(newscalars);
```

Apéndice F

Manual de Usuario

F.1. Introducción

ICMEZU (Interfaz para el cálculo de mapas de elevación de zonas urbanas), es una aplicación gráfica que integra distintas herramientas para trabajar con imágenes estereoscópicas de zonas urbanas. Este manual es una guía para instalar, configurar y usar las siguientes funcionalidades:

- Calcular mapas de elevación a partir de pares estéreo.
- Estimar errores usando las máscaras devueltas por los algoritmos o agregadas manualmente.
- Generar nuevos pares que simulen cambios en la altura de las cámaras.
- Integrar a Icmezu dinámicamente nuevos algoritmos para trabajar con imágenes estereoscópicas
- Ver anaglifos.
- Guardar y Cargar proyectos de trabajo.
- Ver escenarios 3D usando los mapas de disparidad calculados.

F.2. Requerimientos Especiales

Para instalar y usar icmezu es necesario cumplir con ciertos requerimientos tanto de software como de hardware que se listan a continuación.

F.2.1. Requerimientos de Software

- Sistema Operativo: Linux
- Librería gtk+2.4 o superior (La mayoría de las distribuciones de linux actuales cumplen con este requerimiento)
- Megawave2 (solo si se quiere compilar el código fuente)

F.2.2. Requerimientos mínimos de Hardware

- Procesador Intel Pentium III.
- Memoria RAM: 128 MB.
- Espacio Libre en el Disco 100MB.

F.3. Instalación

La instalación se puede hacer compilando el código fuente o ejecutando un paquete de instalación. Con esta última opción el proceso se hace de forma automática y funciona para cualquier distribución Linux (que cumpla con los requerimientos de software mencionados anteriormente). Aunque solo se ha testeado en las siguientes distribuciones: Fedora Core 3, Fedora Core 4 y, Suse 4.

F.3.1. Instalación usando autopackage

Para hacer la instalación usando autopackage siga los siguientes pasos:

1. Copiar el archivo icmezu-package.tar.gz en el disco
2. Abrir una Shell y en el directorio donde se copió el archivo anterior ejecutar:
   a) tar -xvzf icmezu-package.tar.gz
   b) cd icmezu-package
   c) ./icmezu-1.0x86.package

F.3.2. Compilando el código fuente

Para poder compilar el código fuente es necesario que se encuentre instalada la aplicación megawave en el sistema operativo y tener seteada correctamente la variable MEGAWAVE2 con la ruta del directorio donde está instalado megawave. Luego los pasos para la compilación son:

1. Copiar el archivo icmezu-scr.tar.gz en el directorio donde se quiere instalar
2. Abrir una Terminal y en el directorio donde se ha copiado el archivo ejecutar los siguientes comandos:
3. tar -xvzf icmezu-src.tar.gz
4. cd icmezu-src
5. sh install.sh

F.3.3. Ejecución

Para ejecutar la aplicación hay que abrir una terminal y escribir icmezu

F.4. Usando el Sistema

F.4.1. Projects

Para empezar a trabajar con un par de imágenes estéreo es necesario crear o abrir un Project existente. A continuación se listan los pasos para crear, abrir y guardar un Project.

F.4.1.1. Nuevo Project

1. En la pantalla principal, seleccionar desde el menú "File/New Project" (figura F.1)

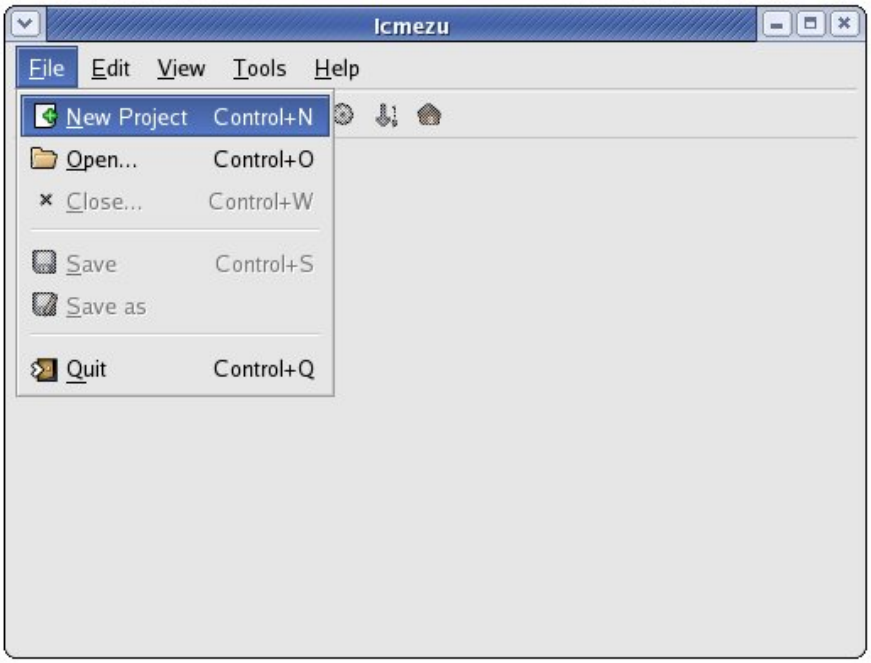


Figura F.1: Pantalla Principal

2. Luego el sistema solicita en una ventana como la de la figura F.2 la siguiente información que debe ser ingresada

   - Nombre del Project
   - Path de las imágenes referencia, secundaria y ground-truth
   - Altura y B/h de las cámaras.

3. Finalmente hacer click en el botón OK:

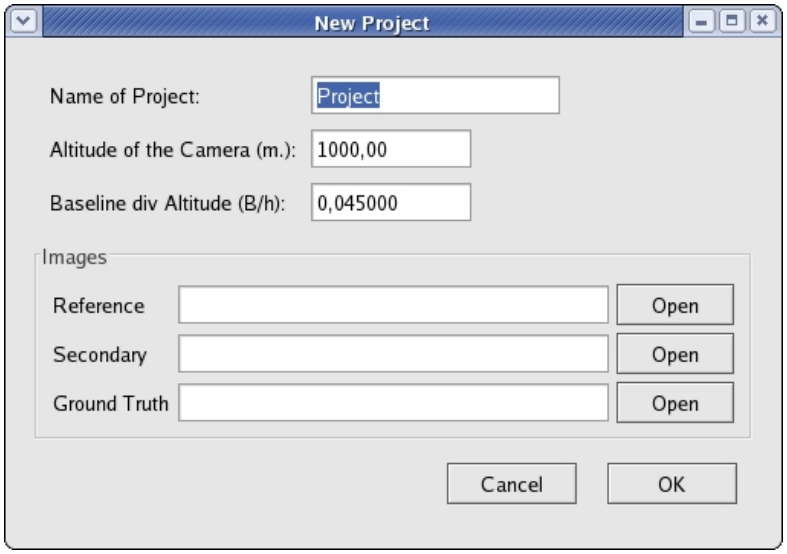


Figura F.2: Información del nuevo Project

F.4.1.2. Guardar Project

1. En la pantalla principal, seleccionar desde el menú “File/Save As...”
2. Ingresar el nombre y la ruta solicitada.

F.4.1.3. Abrir Project

1. En la pantalla principal, seleccionar desde el menú “File/Open”
2. Ingresar la ruta del proyecto que se quiere abrir.

F.4.1.4. Ventana inicial del Project

Cuando se abre o se crea un nuevo Project, en la pantalla principal (ej. figura F.3) se muestra la información general del mismo (nombre, altura, juego de imágenes, etc.). Seleccionado las fichas "Tests" y "Masks", se puede acceder a la información restante (mapas calculados, simulaciones realizadas, errores estimados, etc.).

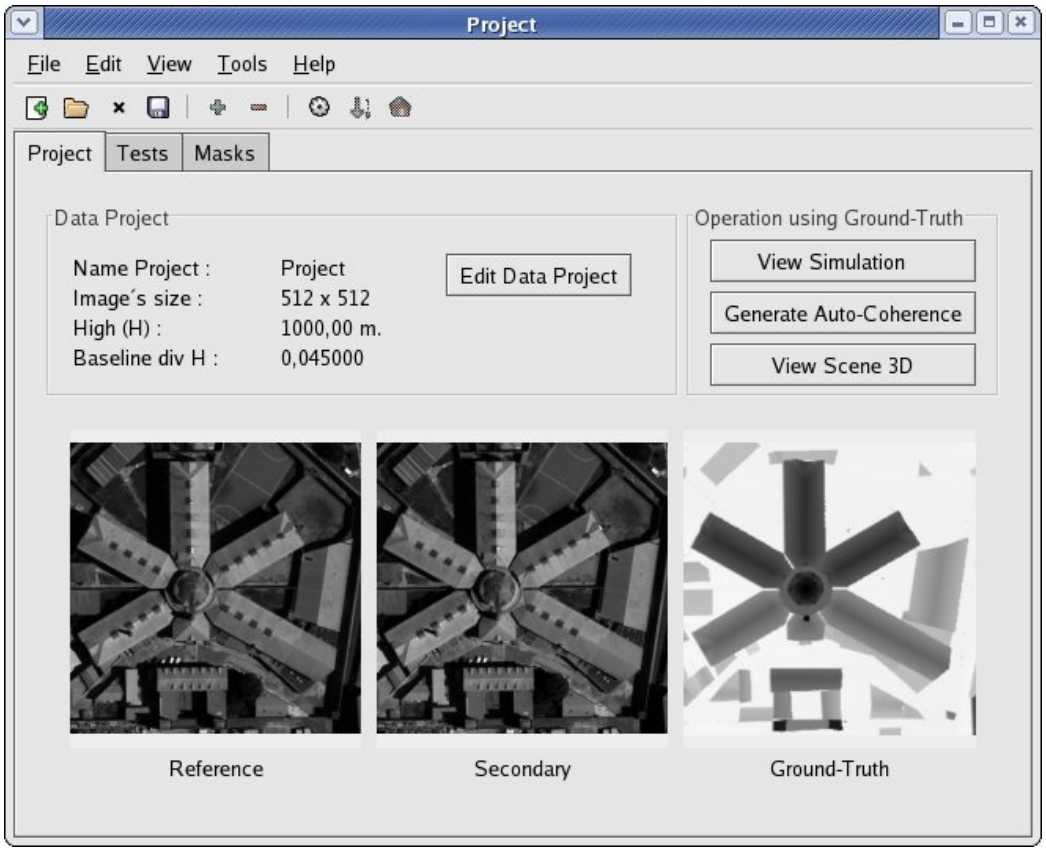


Figura F.3: Ventana inicial del Project

F.4.2. Tests

En la ficha "Tests" se listan las instancias de los algoritmos estereoscópicos agregadas al Project. Para calcular un mapa de disparidad, primero el usuario tiene que agregar un nuevo Test (seleccionando un algoritmo e ingresando los valores de sus parámetros de entrada), y luego ejecutarlo. A continuación se describen los pasos para hacer las siguientes operaciones

- Agregar un Test
- Ejecutar un Test
- Eliminar un Test
- Ver un mapa de disparidad calculado
- Generar el escenario 3D usando un mapa de disparidad.

F.4.2.1. Agregar un Test

1. Desde el menú seleccionar “Edit/Add Test” (figura F.4)

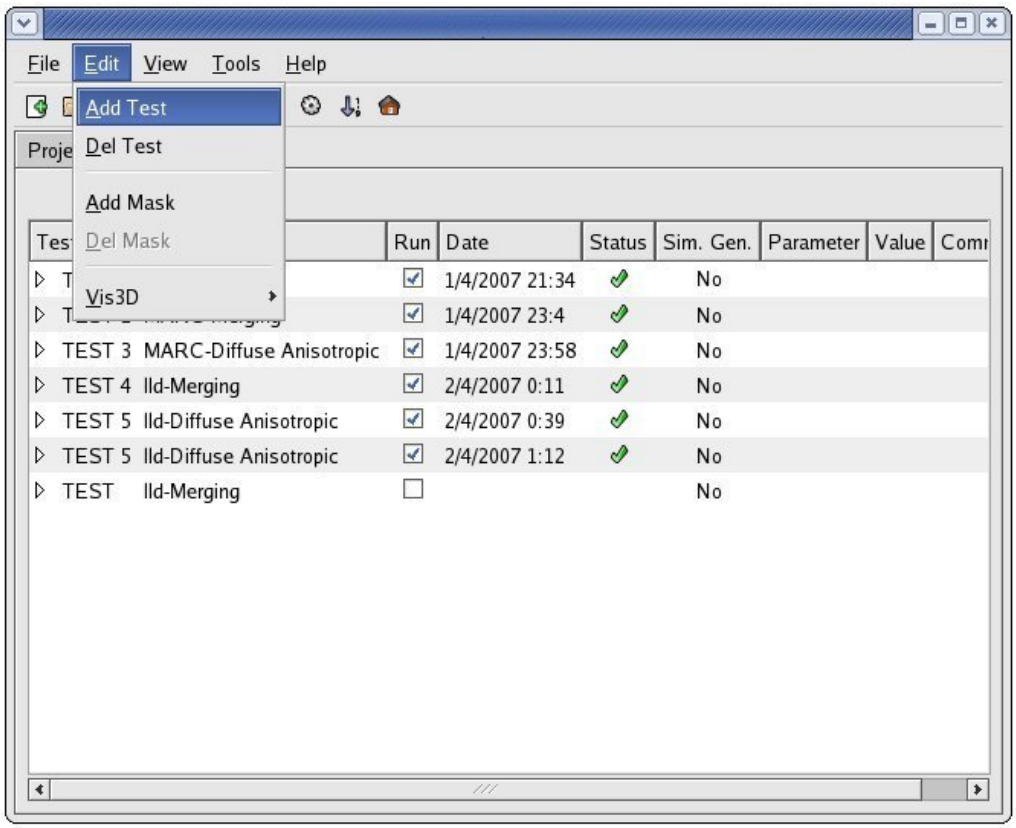


Figura F.4: Paso 1

2. En la siguiente pantalla ingresar un nombre para el test y seleccionar el algoritmo estereoscópico que se quiere agregar (figura F.5).

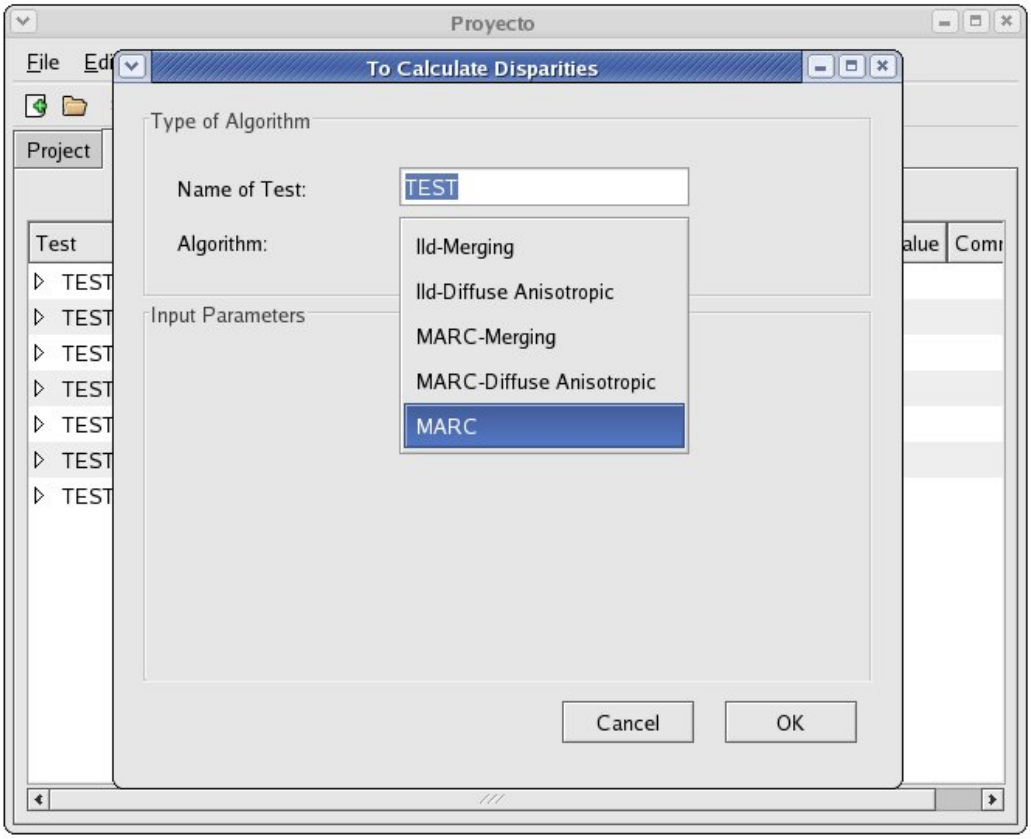


Figura F.5: Paso 2

3. Luego en la ventana se listan los parámetros de entrada del algoritmo seleccionado, ingresar sus valores y finalmente hacer click en OK (figura F.6).

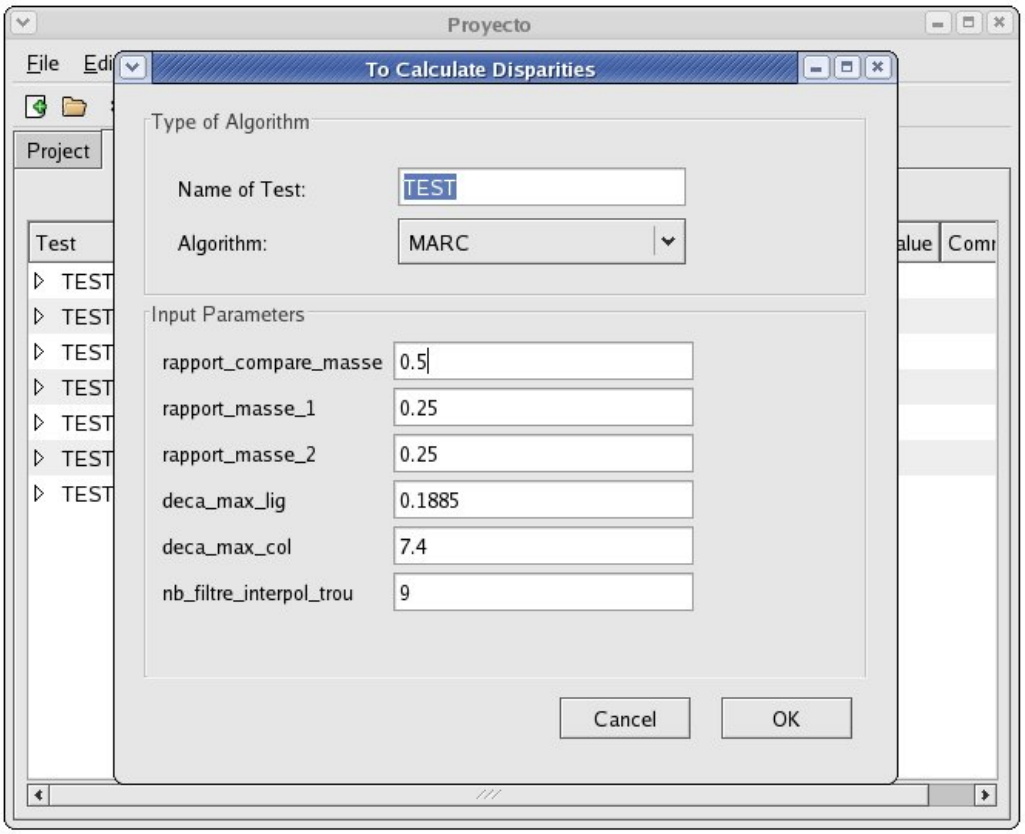


Figura F.6: Paso 3

F.4.2.2. Ejecutar un Test

1. En la ficha “Tests” seleccionar el Test que se quiere ejecutar (figura F.7).

2. Seleccionar desde el menú “Tools/Run/Test Selected” y esperar a que termine la ejecución (En el campo Status se muestra un tick verde para los Tests que ya han sido ejecutados).

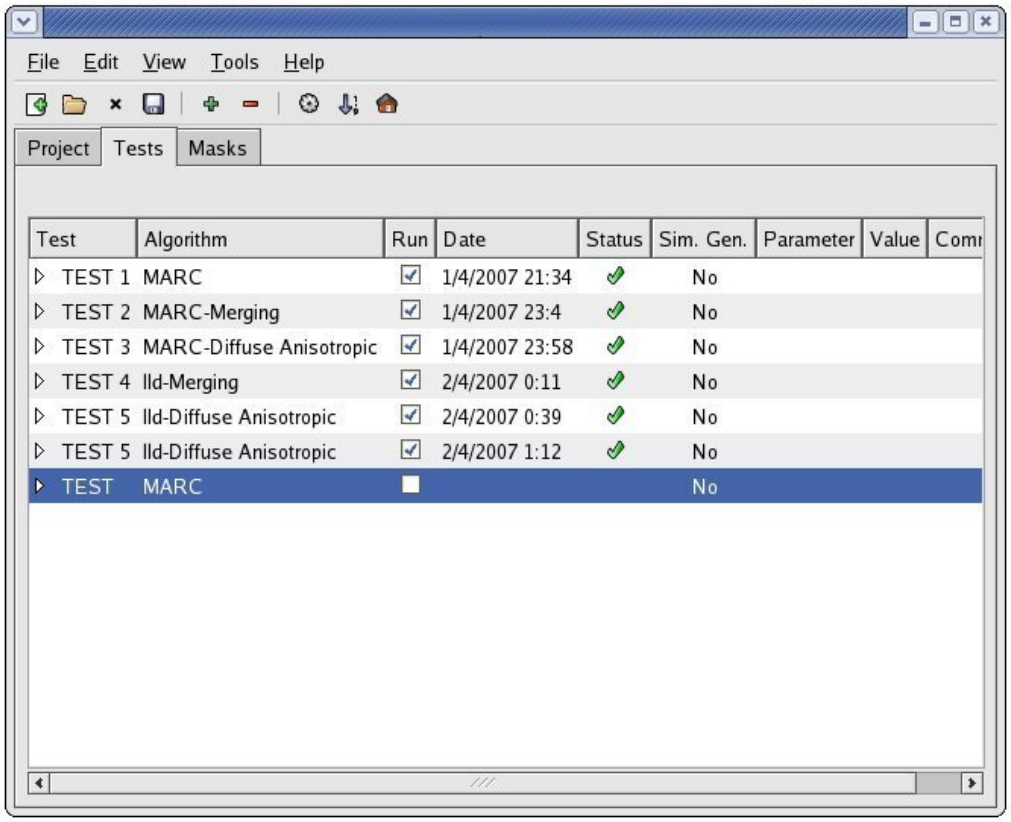


Figura F.7: Listado de Tests

F.4.2.3. Eliminar un Test

1. En la ficha “Tests” seleccionar el Test que se eliminar.
2. Seleccionar desde el menú “Edit/Del Test”.

F.4.2.4. Visualizar Mapa de Disparidad calculado

1. En la ficha “Tests” seleccionar al Test que se quiere ver el mapa de disparidad.
2. Seleccionar desde el menú “View/Disparity Map” (o hacer doble click sobre el Test).
3. Se muestra el Visor de Imágenes con el mapa de disparidad (figura F.8).

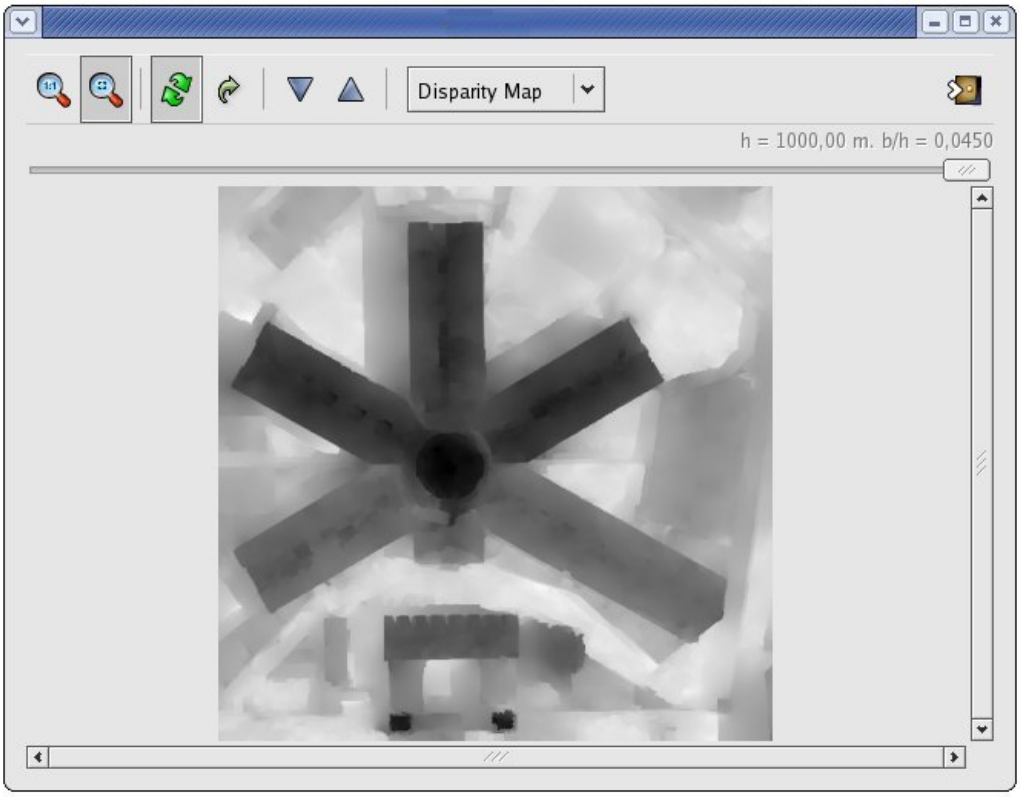


Figura F.8: Visor

F.4.2.5. Visualización 3D

1. En la ficha “Tests” seleccionar el Test con el que se quiere ver la escena 3D.
2. Seleccionar desde el menú “View/Using Disparity Map/Scene 3D”.
3. Se despliega una ventana con la escena generada (figura F.9). Pudiendo hacer las siguientes operaciones con el mouse:
   - Dejando apretado el botón izquierdo y moviendo el mouse se rota la imagen.
   - Dejando apretado el botón del medio y moviendo el mouse se mueve la cámara hacia delante o atrás.
   - Dejando apretado el botón derecho y moviendo el mouse se mueve la cámara hacia los costados.

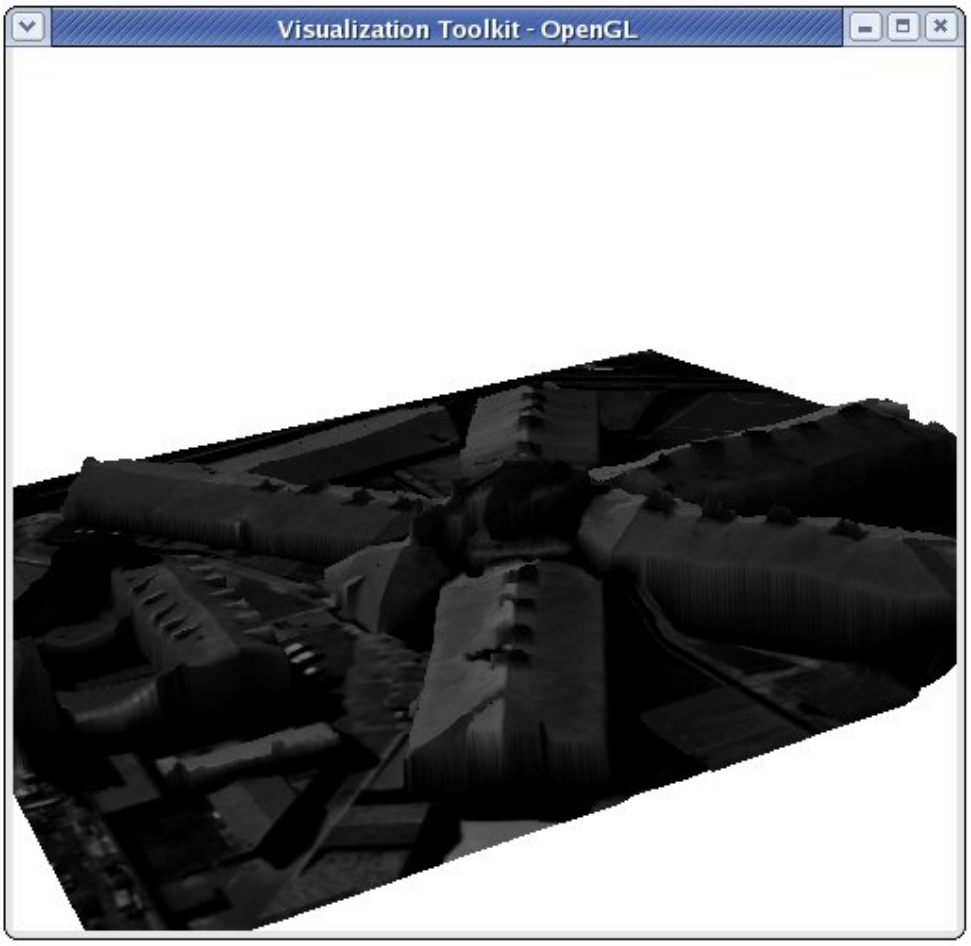


Figura F.9: Escenario 3D

F.4.3. Simulaciones

Las simulaciones consisten en emular cambios en las alturas de las cámaras del par estéreo original (dentro del rango de la altura original y una altura mínima).
Éstas se generan a partir de un mapa de disparidad (que puede ser el ground-truth o uno calculado por el usuario), la altura mínima del rango de alturas a simular y la cantidad N de niveles a generar (debido de que dentro del intervalo solo se generan una cantidad finita N de pares simulados y los restantes se aproximan median una interpolación lineal). A continuación se describen los pasos para hacer las siguientes operaciones

- Generar una simulación
- Visualizar una simulación

F.4.3.1. Generar una Simulación

1. En la ficha "Tests" seleccionar al Test que se quiere usar para generar la simulación (si se quiere usar el Ground-Truth saltear este paso).
2. Seleccionar desde el menú "Tool/Generate Sim/Test Selected" (o "Tool/Generate Sim/Using Ground-Truth" si se quiere usar el Ground-Truth).
3. En la pantalla que se despliega (figura F.10) ingresar el número de niveles que se quiere generar, la altura mínima a simular y la disparidad máxima del par estéreo original.
4. Hacer click en OK y esperar a que termine la simulación.

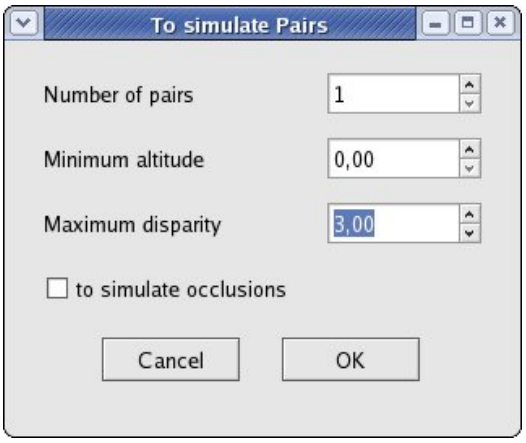


Figura F.10: Generar simulación

F.4.3.2. Visualizar Simulación

1. En la ficha “Tests” seleccionar al Test que se quiere ver la simulación (si se usa el Ground-Truth saltear este paso).
2. Seleccionar desde el menú “View/Using Map Disparity/Simulation” (o “View/Using Ground-Truth/Simulation” si se usa el Ground-Truth).
3. En el visor de imágenes (figura F.11) se despliega el par estéreo. Pudiendo hacer las siguientes operaciones:
   - Con la barra o las flechas se cambia la altura de las cámaras.
   - Con el combo box se selecciona otras imágenes para visualizar (Mapa de Disparidad, Ground-Truth, Anaglifo generado con el par simulado, solo la imagen secundaria o la derecha).
   - Click izquierdo y derecho sobre la imagen hacen un zoom in y un zoom out respectivamente.

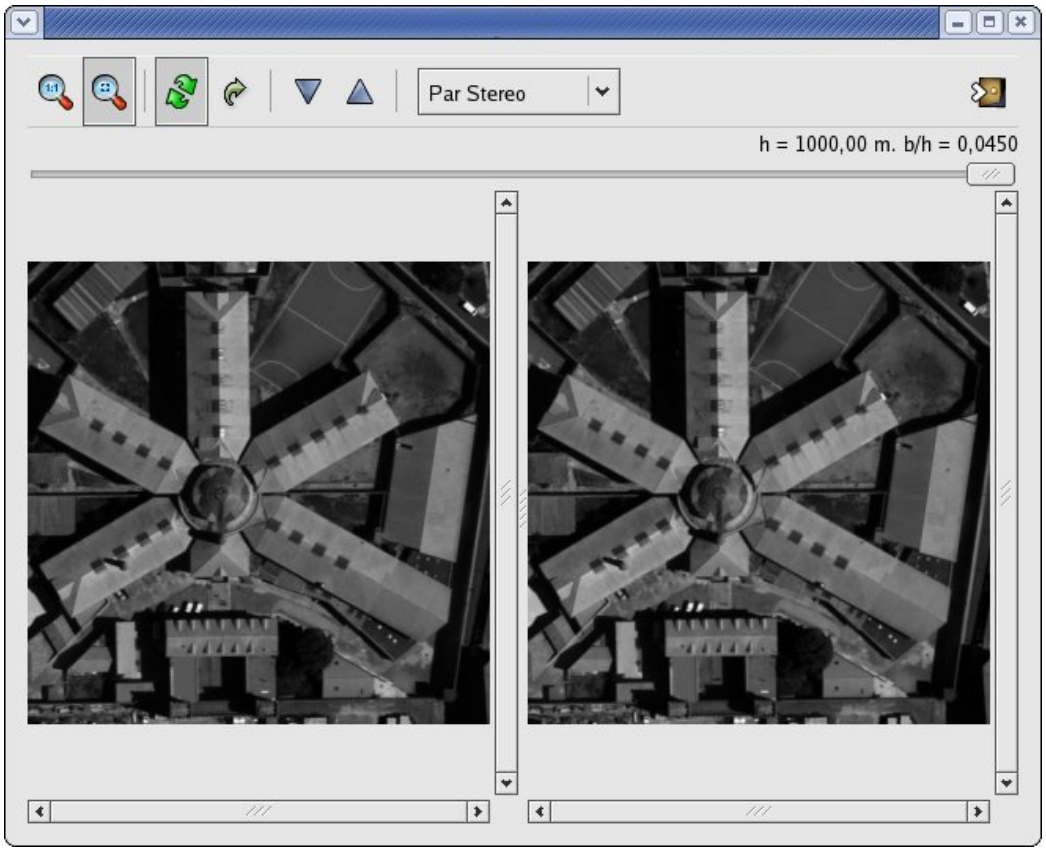


Figura F.11: Visor de Imágenes

F.4.4. Máscaras

En la ficha "Mask", se listan los errores estimados usando todo los pixeles o los que son válidos por las máscaras que son ingresadas por el usuario o agregadas automáticamente por los algoritmos estereoscópicos. A continuación se describen los pasos para hacer las siguientes operaciones:

- Agregar Nueva Máscara.
- Eliminar Máscara.
- Ver Errores estimados usando una Máscara.

F.4.4.1. Agregar Nueva Máscara

1. Ir a la ficha Masks, y seleccionar desde el menú "Add/Mask".
2. En la nueva pantalla (figura F.12) ingresar el nombre y ubicación de la nueva Máscara.
3. Hacer click en OK.

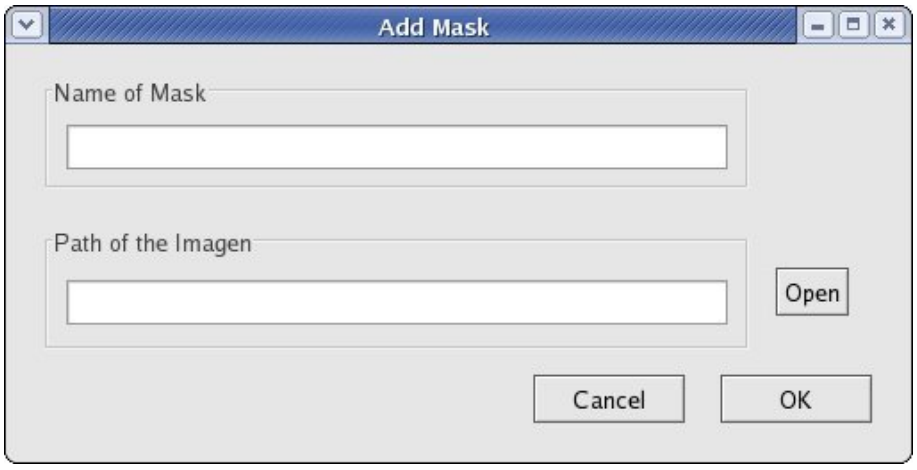


Figura F.12: Ventana: Nueva Máscara

F.4.4.2. Eliminar Máscara

1. Ir a la ficha Masks, y seleccionar en el listado (figura F.13) la máscara que se quiere eliminar.

2. Desde el menú seleccionar "Edit/Del Mask".

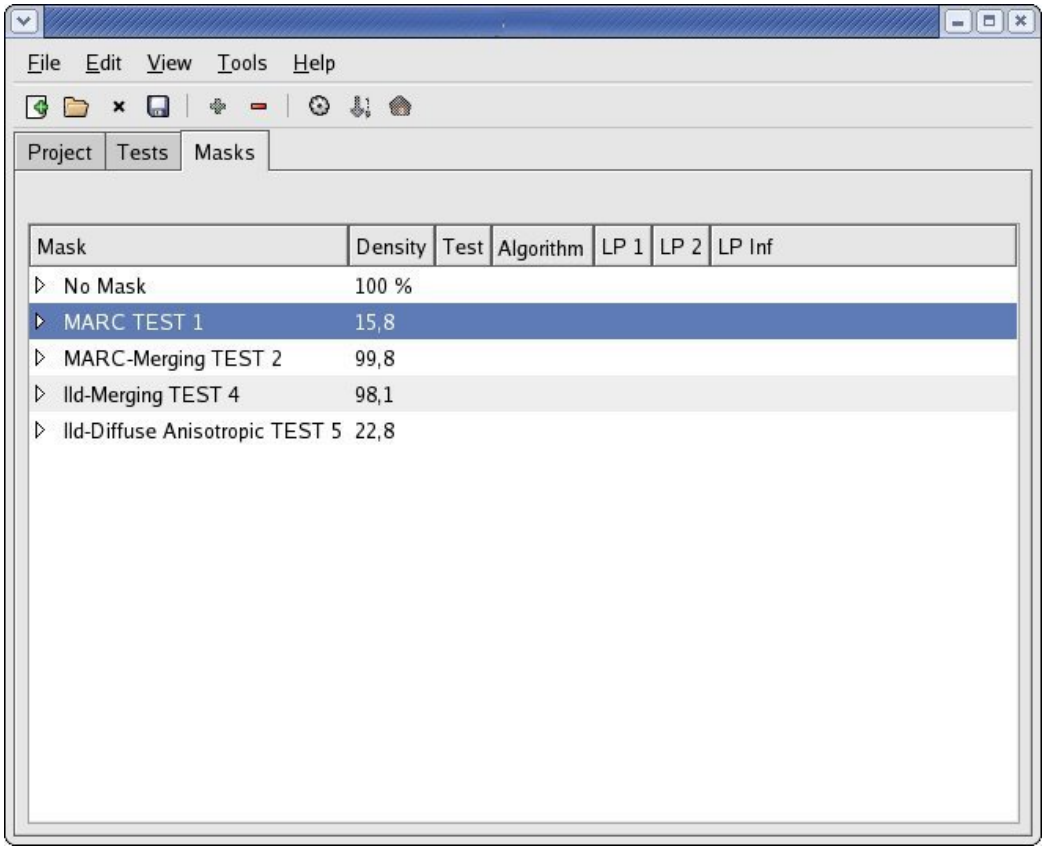


Figura F.13: Listado de las Máscaras

F.4.4.3. Ver Errores estimados con una Máscara

1. Ir a la ficha Masks, y expandir la máscara que se quiere usar para ver los errores estimados.

2. Debajo de la máscara se despliegan los errores (figura F.14).

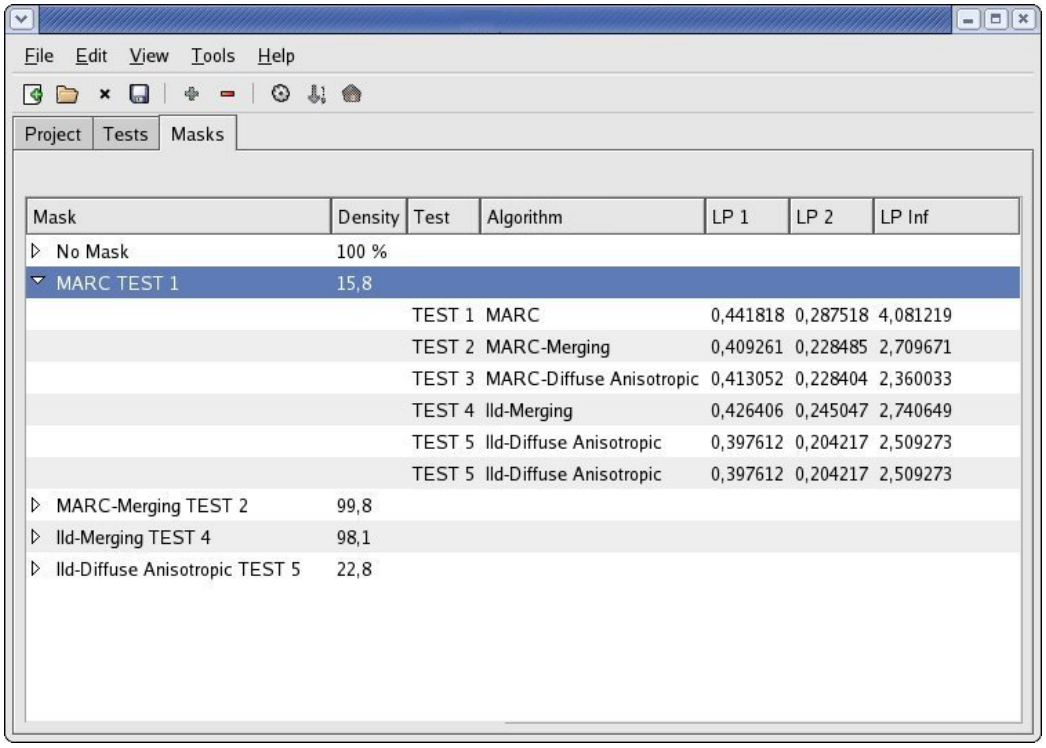


Figura F.14: Listado de los Errores Estimados

F.4.5. Módulos

Para incorporar un nuevo algoritmo estereoscópico a icmezu es necesario crear una librería que tenga la implementación del algoritmo, y especificar en un archivo XML los parámetros de entrada del mismo. Una vez hecho lo anterior (que en C se describe como hacerlo), hay que registrar el archivo XML en icmezu. A continuación se describen los pasos para hacer las siguientes operaciones.

- Agregar un Nuevo Módulo (registrar el XML de un algoritmo).
- Eliminar un Módulo.

F.4.5.1. Agregar un Nuevo Módulo

1. Seleccionar desde el menú "Tools/Modules/Add..".
2. Ingresar el nombre del algoritmo y la ubicación del archivo XML con la especificación (figura F.15).
3. Seleccionar OK.

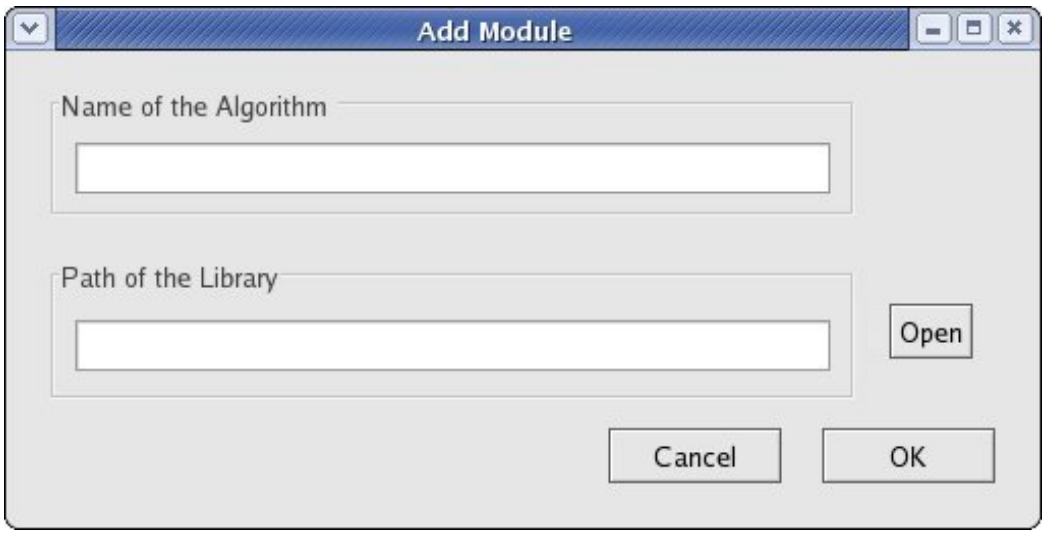


Figura F.15: Ventana: Agregar Módulo

F.4.5.2. Eliminar Módulo

1. Seleccionar desde el menú “Tools/Modules/Del..”.
2. Seleccionar el algoritmo que se quiere eliminar (figura F.16) y hacer click en OK.

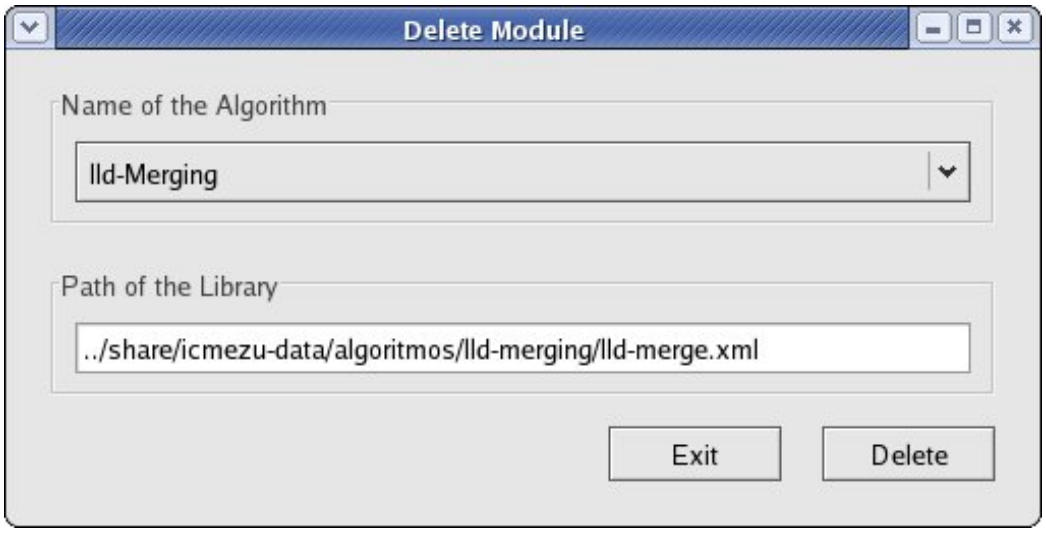


Figura F.16: Venta: Eliminar Módulo

Apéndice G

Casos de Uso del Sistema

En esta sección se listan los casos de usos que especifican el diálogo entre los actores y el sistema para realizar los requerimientos funcionales capturados.

G.1. Actores

G.1.1. Usuario

El usuario es el único actor que interactúa con el sistema y puede acceder a la totalidad de las funcionalidades del sistema.

G.2. Casos de Uso

G.2.1. Diagramas de Casos De Uso

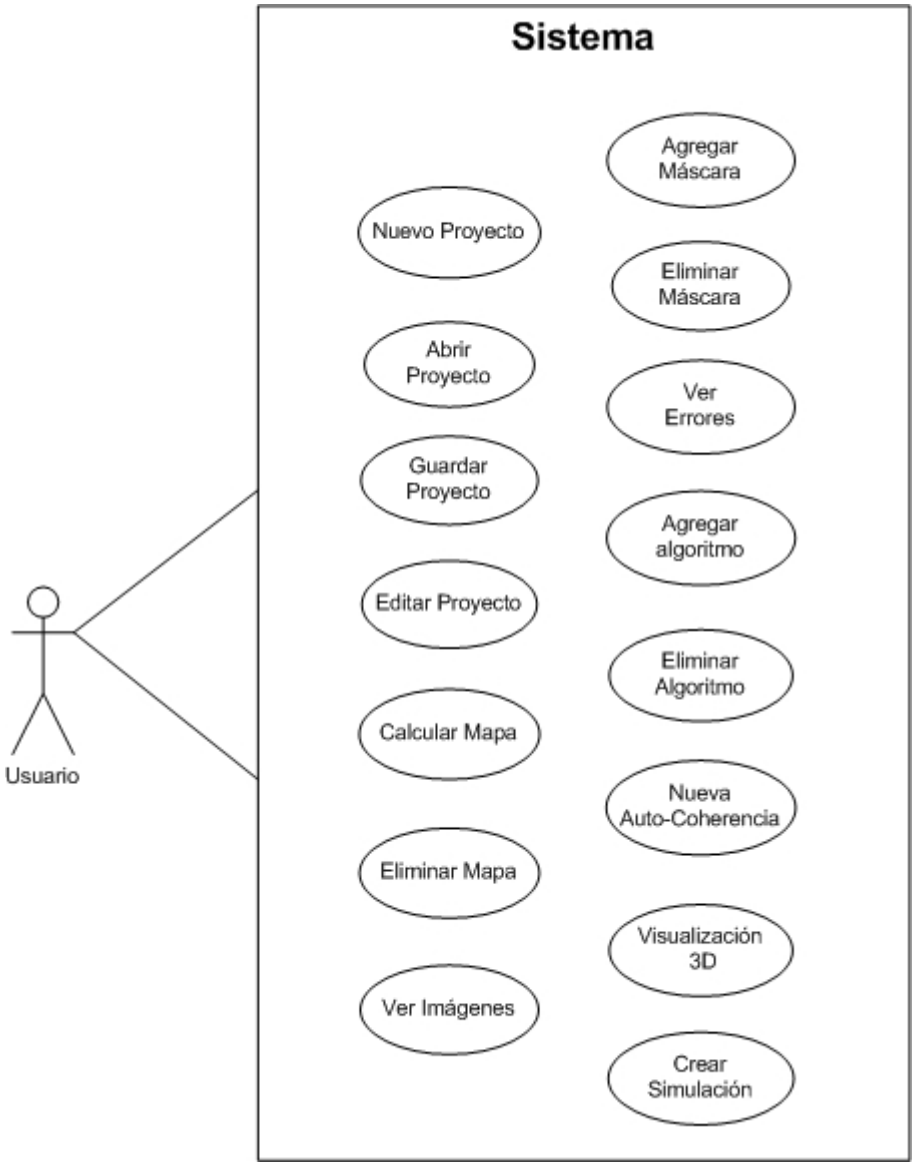


Figura G.1: Diagrama de Casos de Uso

G.2.2. Nuevo Proyecto

G.2.2.1. Descripción

Crea un nuevo proyecto para trabajar con las imágenes estereoscópicas, para crearlo el usuario tiene que especificar la ruta de las imágenes referencia, secundaria y ground-truth e ingresar la configuración de las cámaras del par estéreo (altura y baseline).

G.2.2.2. Pre-condiciones

El usuario se tiene que encontrar en la pantalla principal.

G.2.2.3. Flujo de eventos principal

1 Usuario: Selecciona la opción Nuevo Proyecto.
2 Sistema: Solicita ubicación de las imágenes de Referencia, Secundaria y Ground Truth, e información sobre la configuración de las cámaras.
3 Usuario: Especifica la información solicitada.
4 Usuario: Confirma la creación del nuevo proyecto.
5 Sistema: Vuelve a la pantalla principal.

G.2.2.4. Flujos de eventos alternativos

G.1 El usuario cancela la operación.

G.2.2.5. Post-condiciones

El Sistema carga en memoria el proyecto creado por el usuario. En caso de existir previamente otro proyecto el sistema lo libera de la memoria.

G.2.2.6. Requerimientos especiales

El formato de las imágenes cargada es Fimage. No es obligatorio que el usuario ingrese toda la información solicitada por el Sistema (por ejemplo si el usuario quiere no tiene por que especificar la dirección del ground-truth).

G.2.3. Editar Proyecto

G.2.3.1. Descripción

Permite al usuario cambiar el ground-truth o la configuración de las cámaras del proyecto con el que se está trabajando.

G.2.3.2. Pre-condiciones

El usuario se tiene que encontrar en la pantalla principal y tiene que haber un proyecto cargado.

G.2.3.3. Flujo de eventos principal

1 Usuario: Selecciona la opción Editar Proyecto.
2 Sistema: Despliega la configuración actual del proyecto.
3 Usuario: Hace cambios en la configuración.
4 Usuario: Confirma los cambios.
5 Sistema: Vuelve a la pantalla principal.

G.2.3.4. Flujos de eventos alternativos

G.1 El usuario cancela la operación.

G.2.3.5. Post-condiciones

El Sistema hace los realizados por el usuario en el Proyecto actual.

G.2.3.6. Requerimientos especiales

No Aplica.

G.2.4. Calcular Mapa de Disparidad

G.2.4.1. Descripción

Se calcula un nuevo mapa de disparidad usando un algoritmo seleccionado por el usuario.

G.2.4.2. Pre-condiciones

G.2.4.3. Flujo de eventos principal

1 Usuario: Selecciona la opción nuevo Mapa de Disparidad.
2 Sistema: Lista los algoritmos disponibles para calcular disparidades.
3 Usuario: Selecciona uno
4 Sistema: Muestra los parámetros de entrada para el algoritmo seleccionado.
5 Usuario: Ingresa los valores para los parámetros de entrada y luego selecciona Aceptar.
6 Sistema: Ejecuta el algoritmo seleccionado con los valores ingresados al par estéreo.

G.2.4.4. Flujos de eventos alternativos

G.1 El usuario cancela la operación.

G.2.4.5. Post-condiciones

El resultado del algoritmo es un Mapa de Disparidad y una Máscara que son agregados al Proyecto actual.

G.2.4.6. Requerimientos especiales

El formato de las imágenes generadas es Fimage.

G.2.5. Agregar Máscara

G.2.5.1. Descripción

Agrega una nueva máscara con una imagen y un nombre seleccionado por el usuario.

G.2.5.2. Pre-condiciones

G.2.5.3. Flujo de eventos principal

1 Usuario: Selecciona la opción nueva Máscara.
2 Sistema: Solicita nombre de la máscara y path en el file system de la máscara a agregar.
3 Usuario: Ingresa la información solicitada
4 Sistema: Crea la nueva máscara.

G.2.5.4. Flujos de eventos alternativos

G.1 El usuario cancela la operación.

G.2.5.5. Post-condiciones

Agrega la máscara ingresada manualmente por el usuario al Proyecto actual.

G.2.5.6. Requerimientos especiales

El formato de la máscara debe ser soportado por Fimage.

G.2.6. Eliminar Mapa de Disparidad

G.2.6.1. Descripción

Se elimina del proyecto actual un mapa de disparidad seleccionado por el usuario.

G.2.6.2. Pre-condiciones

G.2.6.3. Flujo de eventos principal

1 Usuario: Selecciona la operación listar mapas de disparidad.
2 Sistema: El sistema muestra los mapas del proyecto actual.
3 Usuario: Elije uno y selecciona eliminar mapa
4 Sistema: Elimina el mapa de disparidad seleccionado.

G.2.6.4. Flujos de eventos alternativos

G.1 El usuario cancela la operación.

G.2.6.5. Post-condiciones

El sistema borra del proyecto actual el mapa de disparidad seleccionado por el usuario.

G.2.6.6. Requerimientos especiales

No Aplica.

G.2.7. Eliminar Máscara

G.2.7.1. Descripción

Se elimina del proyecto actual una máscara seleccionado por el usuario.

G.2.7.2. Pre-condiciones

G.2.7.3. Flujo de eventos principal

1 Usuario: Selecciona la operación listar máscaras.
2 Sistema: El sistema muestra las máscaras del proyecto actual.
3 Usuario: Elije una y selecciona la operación eliminar máscara
4 Sistema: Elimina la máscara seleccionada.

G.2.7.4. Flujos de eventos alternativos

G.1 El usuario cancela la operación.

G.2.7.5. Post-condiciones

El sistema borra del proyecto actual la máscara seleccionada por el usuario.

G.2.7.6. Requerimientos especiales

No Aplica.

G.2.8. Ver Errores

G.2.8.1. Descripción

Se lista los errores estimados para los mapas de disparidad calculados y la máscara seleccionada por el usuario.

G.2.8.2. Pre-condiciones

G.2.8.3. Flujo de eventos principal

1 Usuario: Selecciona la operación listar máscaras.
2 Sistema: El sistema muestra las máscaras del proyecto actual.

3 Usuario: Elije una y selecciona la operación ver errores estimados
4 Sistema: Muestra los errores estimados para los mapas de disparidad usando la máscara seleccionada.

G.2.8.4. Flujos de eventos alternativos

G.1 El usuario cancela la operación.

G.2.8.5. Post-condiciones

No Aplica.

G.2.8.6. Requerimientos especiales

No Aplica.

G.2.9. Generar Pares Simulados

G.2.9.1. Descripción

Genera una simulación a partir del par estéreo real, y un Ground-Truth o un Mapa de disparidad calculado.

G.2.9.2. Pre-condiciones

G.2.9.3. Flujo de eventos principal

1- Usuario: Selecciona la opción Generar Par Simulado y el mapa de disparidad que se va a usar que puede ser el ground-truth o uno calculado.
2 Sistema: Solicita la cantidad de pares que se van a crear y el rango de las alturas que se van a simular.
3 Usuario: Ingresa la información solicitada y selecciona aceptar.
4 Sistema: Corre el algoritmo de simulación y genera los pares simulados.

G.2.9.4. Post-condiciones

Los pares de imágenes son agregados al Proyecto actual.

G.2.9.5. Requerimientos especiales

El formato de las imágenes simuladas es Fimage.

G.2.10. Visualizar Imágenes

G.2.10.1. Descripción

El usuario puede visualizar las imágenes cargadas en el proyecto (Par estéreo Real y Simulados, Ground-Truth, Mapas de Disparidad calculados, anaglifos).

G.2.10.2. Pre-condiciones

El usuario ha ingresado al Sistema y se encuentra en la pantalla principal.

G.2.10.3. Flujo de eventos principal

1 Usuario: Selecciona la opción Ver Imágenes.
2 Sistema: Despliega las imágenes que puede ver en la pantalla (Par estéreo, Ground-Truth, Mapa de Disparidad, Anáglifo, Auto-Coherencia y Simulación)
3 Usuario: El usuario selecciona la imagen que quiere ver.
4 Sistema: El sistema despliega en la pantalla principal la imagen seleccionada por el usuario.
5 Usuario: El usuario cierra el visor y vuelve a la pantalla principal.

G.2.10.4. Flujos de Eventos Alternativos

5A1 Usuario: Selecciona la opción ver otra imagen 5A2 Sistema: Se va al paso 2.
5B1 Usuario: Selecciona la operación de zoom in o zoom out sobre la imagen visualizada.
5B2 Sistema: Aplica el zoom realizado por el usuario en la imagen visualizada y se va al paso 5.
5C1 Usuario: Si las imágenes visualizadas es una simulación, el usuario puede ingresar una nueva altura.
5C2 Sistema: Genera un nuevo par simulado para la altura ingresada por el usuario, se vuelve al paso 5.

G.2.10.5. Post-condiciones

No Aplica.

G.2.11. Generar Auto-coherencia

G.2.11.1. Descripción

Genera una imagen de auto-coherencia usando una imagen de referencia real y otra simulada que es creada usando la imagen secundaria y un mapa de disparidad seleccionado por el usuario.

G.2.11.2. Pre-condiciones

El usuario ha ingresado al Sistema y se encuentra en la pantalla principal.

G.2.11.3. Flujo de eventos principal

1 Usuario: Selecciona la opción Generar Auto-coherencia.
2 Sistema: Solicita que elija el mapa de disparidad para realizar la operación.
3 Usuario: Ingresa la información solicitada.
4 Sistema: Calcula la imagen de auto-coherencia.

G.2.11.4. Flujos de eventos alternativos

G.1 El usuario cancela la operación.

G.2.11.5. Post-condiciones

Cuando se termina de calcular la imagen de auto-coherencia, ésta es agregada al Proyecto actual.

G.2.11.6. Requerimientos especiales

No Aplica.

G.2.12. Visualización 3D

G.2.12.1. Descripción

Muestra la escena 3D generada con la imagen de referencia y un mapa de disparidad que puede ser el ground-truth o uno calculado.

G.2.12.2. Pre-condiciones

El usuario ha ingresado al Sistema y se encuentra en la pantalla principal.

G.2.12.3. Flujo de eventos principal

1 Usuario: Selecciona el mapa de disparidad que quiere usar para la representación 3D y ejecuta la operación "Visualización 3D".
2 Sistema: Muestra en 3D la escena generada con el mapa de disparidad seleccionado.

G.2.12.4. Post-condiciones

No Aplica.

G.2.12.5. Requerimientos especiales

No Aplica.

G.2.13. Guardar Proyecto

G.2.13.1. Descripción

El usuario guardad el Proyecto actual en el file system.

G.2.13.2. Pre-condiciones

No hay.

G.2.13.3. Flujo de eventos principal

1 Usuario: Selecciona la opción Guardar Proyecto.
2 Sistema: Solicita el nombre y ubicación donde se quiere guardar el proyecto
3 Usuario: Ingresa la información solicitada
4 Usuario: Confirma la operación.
5 Sistema: Vuelve a la pantalla principal.

G.2.13.4. Flujos de eventos alternativos

G.1 El usuario cancela la operación.

G.2.13.5. Post-condiciones

El Sistema guarda el proyecto en la dirección y con el nombre ingresado por el usuario.

G.2.13.6. Requerimientos especiales

No Aplica.

G.2.14. Abrir Proyecto

G.2.14.1. Descripción

El usuario carga en memoria un Proyecto guardado en el file system.

G.2.14.2. Pre-condiciones

No hay.

G.2.14.3. Flujo de eventos principal

1 Usuario: Selecciona la opción Abrir Proyecto.
2 Sistema: Solicita el nombre y ubicación del proyecto que se quiere abrir
3 Usuario: Ingresa la información solicitada
4 Usuario: Confirma la operación.
5 Sistema: Vuelve a la pantalla principal.

G.2.14.4. Flujos de eventos alternativos

G.1 El usuario cancela la operación.

G.2.14.5. Post-condiciones

El Sistema carga el proyecto indicado.

G.2.14.6. Requerimientos especiales

No Aplica.

G.2.15. Registrar algoritmo para calcular mapas de disparidad

G.2.15.1. Descripción

Se registra en la aplicación un nuevo algoritmo para calcular disparidades que puede ser usado en cualquier proyecto.

G.2.15.2. Pre-condiciones

No hay.

G.2.15.3. Flujo de eventos principal

1 Usuario: Selecciona la opción Cargar Módulo.
2 Sistema: Solicita la ubicación del archivo de configuración del algoritmo y el nombre con el que se quiere identificarlo.
3 Usuario: Ingresa la información solicitada.
4 Usuario: Confirma la operación.
5 Sistema: Vuelve a la pantalla principal.

G.2.15.4. Flujos de eventos alternativos

G.1 El usuario cancela la operación.

G.2.15.5. Post-condiciones

El Sistema registra el nuevo algoritmo en el sistema, y puede ser usado en cualquier proyecto para calcular mapas de disparidad.

G.2.15.6. Requerimientos especiales

La librería tiene que estar implementada en C o C++.

G.2.16. Borrar registro

G.2.16.1. Descripción

El usuario borra el registro de un algoritmo estereoscópico.

G.2.16.2. Pre-condiciones

No hay.

G.2.16.3. Flujo de eventos principal

1 Usuario: Selecciona la opción Borrar registro.
2 Sistema: Despliega una lista de los módulos registrados en el Sistema.
3 Usuario: Selecciona el que quiere eliminar.
4 Usuario: Confirma la operación.
5 Sistema: Vuelve a la pantalla principal.

G.2.16.4. Flujos de eventos alternativos

G.1 El usuario cancela la operación.

G.2.16.5. Post-condiciones

El Sistema borra el registro del módulo seleccionado por el usuario, por lo que el algoritmo no se va a poder usar más en ningún proyecto al menos que se vuelva a registrar con el caso de uso Registrar Algoritmo.

G.2.16.6. Requerimientos especiales

No Aplica.

Bibliografía

[1] J. Preciozzi: *Dense Urban Elevation Models from Stereo Images by an Affine Region Merging Approach.*, Montevideo, September 18, 2006

[2] L.Igual, J. Preciozzi, L. Garrido, A. Almansa, V. Caselles and B. Rouge. *Automatic Low Baseline Stereo in Urban Areas*, 2007

[3] A. Almansa, G. Facciolo, L. Igual, A. Pardo, J. Preciozzi, *Small baseline stereo for Urban Digital Elevation Models using variational and region-merging techniques*, May 10, 2006

[4] V. Caselles, L. Garrido and L. Igual, *A contrast invariant approach to motion estimation*, International Conference on Scale Space, 2005.

[5] N. Camlong, *Report cssi/111-1/cor-et-marc-2, description de l'algorithme multiresolution algoritme to refine correlation*, Technical report, CNES, 2001.

[6] V. Muron, *Report cssi/111-1/cor-et-marc-5, manuel utilisateur de la chaine de calcul de d'ecalages entre images par l'algorithme marc*, Technical report, CNES, 2003.

[7] P. Musé, F. Sur, F. Cao, Y. Gousseau, J. Morel *Accurate estimates of false alarm number in shape recognition*

[8] A. Desolneux, L. Moisan, J. Morel: *A theory of digital image analysys*, July 8, 2004

[9] L. Moisan: *Modeling and Image Processing*, December 2003

[10] Myron Z. Brown,Darius Burschka, and Gregory D. Hage: *Advances in Computational Stereo*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 25, No. 8, August 2003

[11] Daniel Scharstein and Richard Szeliski: *A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms*, November 2001 Technical Report MSR-TR-2001-81

[12] S.S. Intille and A.F. Bobick. *Incorporating intensity edges in the recovery of occlusion regions.* Proc. Intl Conf. Pattern Recognition, 1:674–677, 1994.

[13] V. Kolmogorov and R. Zabih. *Computing visual correspondence with occlusions via graph cuts.*

[14] O. Faugeras, Q.-T. Luong, and T. Papadopoulo. *The Geometry of Multiple Images : The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications.* MIT Press, 2001. FAU o 01:1 1.Ex.

[15] J. Delon. *Fine comparison of images and other problems.* PhD thesis, Ecole Normale Superieure de Cachan, 2004.

[16] J. Delon and B. Rougé. Analytic study of the stereoscopic correlation, 2004.

[17] Daniel Scharstein and Richard Szeliski. Stereo Vision Research Page. www, 15 de julio de 2007. http://www.middlebury.edu/stereo/

[18] Megawave Home Page. www, 15 de julio de 2007. http://www.cmla.ens-cachan.fr/Cmla/Megawave/

[19] The GIMP Toolkit (GTK+). www, 15 de julio de 2007. http://www.gtk.org/

[20] The Visualization ToolKit (VTK). www, 15 de julio de 2007. http://www.vtk.org/

[21] Dov Grobgeld. GtkImageViewer. www, 15 de julio de 2007. http://giv.sourceforge.net/gtk-image-viewer/