



Implementación de un método de detección de contornos con precisión sub-píxel en un entorno gráfico y estudio comparativo para aplicaciones en imágenes 2D y 3D

Implementación de un método de detección de contornos con precisión sub-píxel en un entorno gráfico y estudio comparativo para aplicaciones en imágenes 2D y 3D

Proyecto de Fin de Carrera

Autor: Daniel Elías Santana Cedrés

Tutores: Agustín Trujillo Pino

Karl Krissian



Daniel Elías
Santana
Cedrés

Implementación de un método de detección de
contornos con precisión sub-píxel en un entorno
gráfico y estudio comparativo para aplicaciones
en imágenes 2D y 3D

Daniel Elías Santana Cedrés

Escuela de Ingeniería Informática
Universidad de Las Palmas de G.C.

Escuela de Ingeniería Informática. Universidad de Las Palmas de G.C.

Proyecto fin de carrera

Título: Implementación de un método de detección de contornos con precisión sub-píxel en un entorno gráfico y estudio comparativo para aplicaciones en imágenes 2D y 3D

Apellidos y nombre del alumno: Santana Cedrés, Daniel Elías

Fecha: 29 de septiembre de 2011

Tutores:

Dr. Agustín Trujillo Pino

Profesor de la Universidad de Las Palmas de Gran Canaria.

Área de conocimiento: Ciencias de la Computación e Inteligencia Artificial.

Dr. Karl Krissian

Investigador contratado del programa Ramón y Cajal.

Universidad de Las Palmas de Gran Canaria.

Escuela de Ingeniería Informática. Universidad de Las Palmas de G.C.

Agradecimientos

Se terminan las revisiones y empieza a oírse como arranca la impresora, mientras cruzo los dedos para que no se quede atascado ningún papel. Pero, antes de imprimir, llega el momento de dar las gracias a todas las personas que de una manera u otra han intervenido en la consecución de este trabajo. No es algo sencillo, al menos para mí. Durante bastante tiempo no he podido evitar tener en mente cómo redactar los agradecimientos, pero sólo he conseguido tener dos cosas claras: No voy a ser breve y no quiero hacer simplemente una lista de personas. Es por esa razón que he decidido contar tres historias.

La primera de las historias tiene que ver con la familia. La familia es la que nos hace llegar a donde estamos, la que está en todo momento y con la que compartimos muchas de las cosas de nuestra vida. Me gustaría agradecer a mis padres, María y Suso, su paciencia. Sobre todo esas temporadas en las que vivía 12 horas en la facultad e iba a trabajar 8 horas a casa, ¿o era al revés? A mi hermana, Pepa, le agradezco su constante ánimo. Siempre me ha admirado su capacidad de entusiasmarse incluso cuando no entendía una palabra de lo que yo estaba diciendo. De mi familia he aprendido que todo esfuerzo vale la pena, aún más cuando lo haces por alguien que te importa o a quien quieres. Gracias familia.

Lo segundo que quiero contar está relacionado con el entorno académico. Aquí me gustaría dar las gracias a mis tutores, Agustín Trujillo y Karl Krissian. Agustín ha sido capaz de soportar mis porqués en tantas y tantas reuniones, en las que se llenaban folios con dibujos hasta que yo era capaz de entender lo que había detrás de cada algoritmo. Nunca olvidaré su frase cuando íbamos a empezar con la parte 3D *“Bueno, ahora a por 3D. Tranquilo, si esto es fácil, es igual... pero con z”*. Por otro lado, probablemente Karl es la persona sobre la tierra capaz de tener más conversaciones sobre programación y tenis con un café en la mano. Como principal experto en AMILab, sin su ayuda no habría sido posible realizar este proyecto. Desde aquí les agradezco a ambos su tiempo, su dedicación y su paciencia.

Me gustaría aprovechar también, ya que estoy en el plano académico, para dar las gracias a aquellos profesores que han creído en mí a lo largo de los años. Guardo un especial recuerdo de Dña. Carmina y D. Gonzalo (en mis tiempos aún se trataba de esa forma a los profesores). De ella aprendí la importancia del orden, la disciplina y la atención por los pequeños detalles. De él recuerdo su genial forma de enseñarnos mapas en Geografía, su capacidad para hacer esquemas (que en muchas ocasiones he echado de menos) y su deducción matemática. Gracias a él tuve la oportunidad de participar en el concurso local de matemáticas, donde quedé primero... por la cola. Ellos siempre me animaron a seguir adelante y eso es algo que valoro.

Durante mi etapa de estudiante he tenido la oportunidad de trabajar como becario de colaboración en la biblioteca del edificio de Informática y Matemáticas. Desde estas líneas quiero tener un especial recuerdo para Paqui, Tere, Mercedes y María. Probablemente no leerán esto, pero con total seguridad este documento pasará por sus manos.

No quisiera olvidarme de dar las gracias al programa de becas del Ministerio de Educación del Gobierno de España. Sin su financiación, seguramente no habría podido cursar estudios universitarios.

La última de las historias que quería contar va sobre los amigos. Se suele decir que los amigos son esa familia que uno elige y con la que pasas gran parte de tu tiempo y, he de decir, que yo he sido muy afortunado en ese sentido. Guardo recuerdos de mis primeros años de carrera, donde todo era raro o diferente. Aquellos años en los que conocí a Gema, Violeta, Ancor, Yeray Ventura, Fran Ruano, Carlos Martín... Fueron comienzos difíciles, pero todos de una forma u otra seguimos adelante. Recuerdo las tardes en el laboratorio de Sistemas Operativos. Fue allí donde conocí a muchos de los amigos que hoy aún conservo. Por ejemplo, estaban Adaya, Sory y Sabina. Todos teníamos problemas normales para implementar concurrencia: los filósofos comensales, los canarios, el baño mixto... ellas tenían los macarrones concurrentes. Fue en esa época cuando también conocí a Abra, Drew, José, Carlos, Sara, Raúl... Recuerdo de aquellas tardes la anécdota del `bzero()` y el día que todos a contrarreloj escribíamos e imprimíamos la memoria de prácticas para entregarla en fecha, mientras fuera había una fiesta erasmus. Lo nuestro si que era fuerza de voluntad. Los informáticos (porque a veces parece que la gente les da miedo llamarnos ingenieros) somos de esas personas que hemos visto hacer casas en una noche, no miento. En las noches que pasábamos en la facultad de arquitectura, mientras los estudiantes hacían maquetas de los proyectos de sus asignaturas y nosotros escribíamos código. A ellos les quedaba casi una bonita estructura al amanecer y a nosotros, con suerte, lográbamos compilar sin errores.

Fue más tarde cuando conocí a Ibra, Ju, Samu, Orlando, Jose, Pedro... Bases de Datos, Biocibernética Computacional... ah no, que eso son asignaturas, a veces me confundo. En concreto, de Ibra siempre recuerdo su capacidad para reírse de todo y sin Ju no hubiera aprobado muchas de las asignaturas. Creo que muchos recuerdan sus "apuntes mágicos" (cuando apruebas ciertas asignaturas eres capaz de atribuirle características sobrenaturales hasta a pequeños trozos de papel esquemáticos).

En el caso de Sara y Carlos, además de compartir prácticas compartimos laboratorio, ella durante la primera mitad del proyecto (y final del suyo) y él durante la segunda. A ellos les agradezco también su paciencia y su capacidad para aguantar mis bromas y tonterías. Tampoco podía dejar atrás a Sara Illera, que también sigue haciendo su proyecto en el laboratorio. No puedo olvidar que, por adopción, mi segundo laboratorio es Comciencia. Muchas veces he sido acogido allí por mis amigos José y Miguel Ángel. A este último quiero agradecerle su apoyo durante mi primera conferencia.

Además hay otra serie de personas que, aunque no hemos compartido el día a día de la carrera, o que no estén tan cerca como el resto, han servido igualmente de apoyo. En primer lugar me gustaría nombrar a Belén. Parafraseando la novela¹ del difunto Douglas Adams, ella ha sido en muchas veces el 42 del sentido de la vida, el universo y todo lo demás. Pero no quiero olvidarme de otras muchas personas: Elena, Tati, Leti, Silvia, Nere, Sandra, Paula, Mónica, Víctor, Tewise, Lourdes... y tantos otros. Gracias aunque sólo sea por preguntarme como me va y por desearme ánimos.

Tengo el orgullo de aprender cada día algo nuevo de todos y cada uno de ellos, y tengo la suerte de poder llamarles amigos. He tenido la oportunidad de estar en muchas de la fases finales de los proyectos de mis amigos, pero eso no me hace estar más preparado. Sólo espero

¹Douglas Adams. *The Hitchhiker's Guide to the Galaxy*. Octubre de 1979

poder hacerlo con la mitad de profesionalidad y conocimiento que ellos lo han hecho.

Hasta aquí han llegado las historias que quería contar. Espero no haber olvidado a nadie, pero aprovecho para agradecer al conjunto complementario y quedarme tranquilo (ventajas del álgebra). Puede que quien lea esto piense que han sido unos agradecimientos largos o extensos, pero permítame que le diga que nunca serán lo suficientemente grandes.

Escuela de Ingeniería Informática. Universidad de Las Palmas de G.C.

“Una vieja máxima mía dice que, cuando has eliminado lo imposible, lo que queda, por muy improbable que parezca, tiene que ser la verdad.”

Sherlock Holmes en *La Aventura de la Diadema de Berilos*
(*The Beryl Coronet*. **Sir Arthur Conan Doyle**.
The Adventures of Sherlock Holmes. Mayo de 1892)

Escuela de Ingeniería Informática. Universidad de Las Palmas de G.C.

Índice general

| | |
|---|-----------|
| 1. Introducción | 13 |
| 1.1. Objetivos | 13 |
| 1.2. Metodología | 14 |
| 1.3. Recursos | 15 |
| 1.4. Estructura del Documento | 15 |
| 1.4.1. Algunas Convenciones de Estilo | 17 |
| 2. Herramientas Utilizadas | 19 |
| 2.1. Herramientas de Desarrollo | 19 |
| 2.1.1. Code::Blocks | 19 |
| 2.1.2. KDevelop | 20 |
| 2.1.3. Xcode | 21 |
| 2.1.4. AMILab | 25 |
| 2.1.4.1. CMake | 27 |
| 2.1.4.2. ITK | 29 |
| 2.1.4.3. VTK | 29 |
| 2.1.4.4. wxWidgets | 30 |
| 2.1.4.5. Pthreads | 30 |
| 2.1.4.6. Zlib | 31 |
| 2.1.4.7. Boost C++ | 32 |

| | |
|---|-----------|
| 2.2. Herramientas de Edición | 32 |
| 2.2.1. \LaTeX | 32 |
| 2.2.1.1. Texto fuente | 33 |
| 2.2.1.2. Composición o compilación | 33 |
| 2.2.1.3. Visualización o impresión | 34 |
| 2.2.1.4. Estructura de un Documento en \LaTeX | 34 |
| 2.2.1.5. \BIBTeX | 36 |
| 2.2.1.6. El paquete <code>gloss</code> | 38 |
| 2.2.1.7. Editor | 39 |
| 2.2.2. OmniGraffle | 41 |
| 3. Métodos | 45 |
| 3.1. Detección de Bordos en Imágenes Bidimensionales | 45 |
| 3.1.1. Métodos Tradicionales a Nivel Píxel | 45 |
| 3.1.1.1. Detectores Basados en la Primera Derivada (Gradiente) | 47 |
| 3.1.1.2. Aproximaciones Discretas del Operador Gradiente | 48 |
| 3.1.1.3. Operadores de Roberts, Prewit, Sobel y Frei-Chen | 49 |
| 3.1.1.4. Operadores Basados en la Segunda Derivada (Laplaciana) | 50 |
| 3.1.1.5. Operador de Canny | 50 |
| 3.1.2. Detección de Contornos con Precisión Sub-píxel | 52 |
| 3.1.2.1. Método Simplificado de Primer Orden | 53 |
| 3.1.2.2. Método Simplificado de Segundo Orden | 56 |
| 3.1.2.3. Contornos en Imágenes Suavizadas | 58 |
| 3.1.2.4. Restauración | 61 |
| 3.2. Detección de Bordos en Imágenes Tridimensionales | 66 |
| 3.2.1. Métodos Tradicionales a Nivel Vóxel | 66 |

| | | |
|-----------|---|------------|
| 3.2.2. | Detección de Contornos con Precisión Sub-vóxel | 68 |
| 3.2.2.1. | Contornos de Segundo Orden | 69 |
| 4. | Desarrollo | 73 |
| 4.1. | 2D | 73 |
| 4.1.1. | Métodos a Nivel Píxel | 73 |
| 4.1.1.1. | Detectores de Bordos Básicos | 74 |
| 4.1.1.2. | Interfaz para los Detectores de Bordos Básicos | 76 |
| 4.1.2. | Métodos Sub-Píxel | 77 |
| 4.1.2.1. | Método Sub-Píxel de Segundo Orden Básico | 79 |
| 4.1.2.2. | Método Sub-Píxel de Segundo Orden para Imágenes con Ruido | 84 |
| 4.1.2.3. | Método Sub-Píxel de Restauración | 92 |
| 4.1.2.4. | <i>Script</i> en AMILab | 97 |
| 4.2. | 3D | 112 |
| 4.2.1. | Métodos a Nivel Vóxel | 112 |
| 4.2.1.1. | Interfaz para los Detectores de Bordos Básicos en 3D | 113 |
| 4.2.2. | Método Sub-Vóxel de Segundo Orden Básico | 117 |
| 4.2.2.1. | <i>Script</i> en AMILab | 120 |
| 4.3. | Imágenes Sintéticas | 132 |
| 4.3.0.2. | <i>Script</i> en AMILab | 136 |
| 5. | Resultados | 147 |
| 5.1. | Estudio de la Generación de Imágenes Sintéticas | 147 |
| 5.1.1. | 2D | 148 |
| 5.1.2. | 3D | 149 |
| 5.2. | Estudio Comparativo | 150 |
| 5.3. | Estudio de un Caso Práctico | 151 |

| | |
|--|------------|
| 6. Conclusiones y Trabajo Futuro | 155 |
| A. Manuales de Usuario | 157 |
| A.1. Compilación de AMILab en Mac OS X | 157 |
| A.1.1. ITK | 158 |
| A.1.2. VTK | 158 |
| A.1.3. wxWidgets | 158 |
| A.1.4. Pthreads | 159 |
| A.1.5. zlib y bzip2 | 159 |
| A.1.6. Boost | 159 |
| A.1.7. AMILab | 160 |
| A.1.8. Problemas Conocidos | 160 |
| A.2. Ayuda del <i>script</i> para la Generación de Imágenes Sintéticas | 160 |
| A.2.1. Inicio Rápido | 161 |
| A.2.2. Interfaz | 161 |
| A.2.2.1. Pestaña Main | 161 |
| A.2.2.2. Pestaña Func | 161 |
| A.2.2.3. Pestaña Help | 162 |
| A.3. Ayuda del <i>script</i> para la Detección de Bordos con Precisión Sub-Píxel | 162 |
| A.3.1. Inicio Rápido | 162 |
| A.3.2. Interfaz | 162 |
| A.3.2.1. Pestaña Param | 162 |
| A.3.2.2. Pestaña Px Info | 163 |
| A.3.2.3. Pestaña Stat | 164 |
| A.3.2.4. Pestaña Set | 164 |
| A.3.2.5. Pestaña Help | 164 |

| | |
|--|------------|
| A.4. Ayuda del <i>script</i> para la Detección de Bordos con Precisión Sub-Vóxel | 164 |
| A.4.1. Inicio Rápido | 165 |
| A.4.2. Interfaz | 165 |
| A.4.2.1. Pestaña Param | 165 |
| A.4.2.2. Pestaña Set | 165 |
| A.4.2.3. Pestaña Help | 166 |
| B. Algunas mejoras añadidas en AMILab | 167 |
| B.1. Mejoras en el Editor de <i>Scripts</i> de AMILab | 167 |
| B.1.1. Buscar y Reemplazar (Find and Replace) | 167 |
| B.1.1.1. Constructor | 168 |
| B.1.1.2. Buscar siguiente (Next) | 169 |
| B.1.1.3. Buscar anterior (Prev) | 170 |
| B.1.1.4. Reemplazar (Replace) | 170 |
| B.1.1.5. Reemplazar todos (Replace all) | 171 |
| B.1.1.6. Tratamiento de eventos | 171 |
| B.1.2. Ir a la línea (Go to line) | 172 |
| B.1.3. Mejoras de la interfaz | 173 |
| B.2. Mejoras en el Visor | 176 |
| B.2.1. Refactorización del Código | 176 |
| B.2.2. Mejoras en la Interfaz | 179 |
| C. Detalles Implementación | 183 |
| C.1. 2D | 184 |
| C.1.1. Clase <code>borderPixel</code> | 184 |
| C.1.2. Clase <code>SubPixel2D</code> | 184 |
| C.2. 3D | 186 |

Índice general

| | |
|---|-----|
| C.2.1. Clase <code>borderVoxel</code> | 186 |
| C.2.2. Clase <code>SubVoxel3D</code> | 187 |

| | |
|-----------------|------------|
| Glosario | 193 |
|-----------------|------------|

Índice de figuras

| | |
|--|----|
| 2.1. Entorno XCode | 22 |
| 2.2. Depurador para XCode | 24 |
| 2.3. Arquitectura de AMILab | 27 |
| 2.4. Interfaz de BIBDesk | 38 |
| 2.5. Interfaz de Texmaker | 40 |
| 2.6. Interfaz de OmniGraffle | 42 |
| 3.1. Modelo ideal de borde | 46 |
| 3.2. Borde en una imagen real | 47 |
| 3.3. Efecto parcial 2D | 53 |
| 3.4. Coordenadas del píxel | 54 |
| 3.5. Área de las columnas | 55 |
| 3.6. Cálculo orden 2 | 57 |
| 3.7. Ventana en una imagen suavizada | 59 |
| 3.8. Proceso de restauración | 63 |
| 3.9. Efecto parcial 3D | 69 |
| 3.10. Orientación de los ejes en 3D | 70 |
| 4.1. Convolución 2D | 75 |
| 4.2. Interfaz para los detectores 2D a nivel píxel | 76 |

| | |
|--|-----|
| 4.3. Resultados usando métodos a nivel píxel | 78 |
| 4.4. Proceso de <i>wrapping</i> | 79 |
| 4.5. Clase <code>borderPixel</code> | 80 |
| 4.6. Diagrama de flujo del método de detección sub-píxel de orden dos | 81 |
| 4.7. Clase <code>SubPixel2D</code> | 82 |
| 4.8. <code>AMIObject</code> | 83 |
| 4.9. Resultado detección sub-píxel en una circunferencia | 84 |
| 4.10. Detección básica en una imagen con ruido | 84 |
| 4.11. Detección básica en imágenes con ruido | 85 |
| 4.13. <code>AMIObject</code> para imágenes con ruido | 86 |
| 4.12. Diagrama de flujo del algoritmo de orden 2 para imágenes con ruido | 87 |
| 4.14. Resultado en imágenes con ruido usando el método de suavizado | 88 |
| 4.15. Diagrama de flujo en imágenes con ruido y detección de bordes cercanos | 89 |
| 4.16. Detección en imágenes con ruido usando ventana flotante | 91 |
| 4.17. Diagrama de flujo del algoritmo de restauración | 94 |
| 4.18. <code>AMIObject</code> del método de restauración | 95 |
| 4.19. Detección en imágenes con ruido mediante método de restauración | 96 |
| 4.20. Estructura directorio del <i>script</i> 2D | 97 |
| 4.21. Icono <i>script</i> sub-píxel | 98 |
| 4.22. Diagrama de clases del <i>script</i> para detectores de bordes a nivel sub-píxel | 99 |
| 4.23. Pestaña principal de la interfaz 2D | 105 |
| 4.24. Pestaña de información del píxel | 106 |
| 4.25. Pestaña del cálculo de estadísticas | 107 |
| 4.26. Pestaña de opciones | 107 |
| 4.27. Pestaña de ayuda | 108 |

| | |
|--|-----|
| 4.28. Resultado en imágenes sintéticas | 109 |
| 4.29. Resultado en TC de disección aórtica | 109 |
| 4.30. Resultado en angiografía cerebral | 110 |
| 4.31. Resultado en angiografía coronaria | 110 |
| 4.32. Resultado en una imagen de la retina | 111 |
| 4.33. Estructura de clases del <i>script</i> para la detección a nivel vóxel | 112 |
| 4.34. Interfaz para los detectores de bordes a nivel vóxel | 114 |
| 4.35. Arteria carótida. Imagen cortesía de The Visual MD | 114 |
| 4.36. Detección a nivel vóxel | 115 |
| 4.37. Renderizado volumétrico de la detección a nivel vóxel | 116 |
| 4.38. Clase borderVoxel | 117 |
| 4.39. Clase SubVoxel3D | 118 |
| 4.40. Diagrama de flujo del método básico sub-vóxel | 119 |
| 4.41. AMIObject del método básico 3D | 120 |
| 4.42. Estructura del directorio del <i>script</i> 3D | 121 |
| 4.43. Icono <i>script</i> sub-vóxel | 121 |
| 4.44. Diagrama de clases del <i>script</i> 3D | 122 |
| 4.45. Ejemplo de <i>glyphing</i> en una imagen tridimensional | 123 |
| 4.46. Pestaña principal | 126 |
| 4.47. Pestaña de preferencias | 127 |
| 4.48. Pestaña de ayuda | 128 |
| 4.49. Resultado en imágenes sintéticas 3D | 129 |
| 4.50. Resultado en una TC de l parte superior del cuerpo | 130 |
| 4.51. Resultado de una TC de la carótida | 130 |
| 4.52. Resultado de una TC de disección aórtica | 131 |

| | |
|---|-----|
| 4.53. Resultado de una TC de disección aórtica. Vista Axial | 131 |
| 4.54. Resultado de una TC del tronco | 132 |
| 4.55. Diagrama de clases de la generación de imágenes sintéticas | 133 |
| 4.56. Ejemplo gráfico de subdivisión | 135 |
| 4.57. Imágenes sintéticas bidimensionales | 135 |
| 4.58. Imágenes sintéticas tridimensionales | 136 |
| 4.59. Estructura del directorio del <i>script</i> para la generación de imágenes sintéticas . . | 137 |
| 4.60. Icono del <i>script</i> para imágenes sintéticas | 137 |
| 4.61. Diagrama de clases del <i>script</i> de imágenes sintéticas | 139 |
| 4.62. Pestaña principal de la interfaz para la generación de imágenes sintéticas | 142 |
| 4.63. Interfaces para imágenes sintéticas 2D | 143 |
| 4.64. Interfaces para imágenes sintéticas 3D | 144 |
| 4.65. Pestaña de ayuda del <i>script</i> para la generación de imágenes sintéticas | 145 |
| 5.1. Gráficas generación de imagen 2D | 149 |
| 5.2. Gráficas generación de imagen 3D | 150 |
| 5.3. Estructura de la pared de una arteria. Imagen cortesía de A.D.A.M. Inc. | 152 |
| 5.4. Aorta normal y diseccionada | 152 |
| 5.5. TC de una disección aórtica | 153 |
| 5.6. Disección sintética | 154 |
| B.1. Detalle de la interfaz para buscar y reemplazar | 168 |
| B.2. Detalle de la interfaz para ir a una línea determinada | 173 |
| B.3. Diagrama de clases para la gestión de pestañas | 174 |
| B.4. Detalle de la interfaz del editor de <i>scripts</i> de AMILab | 175 |
| B.5. Diagrama de clases de <i>DessinImage</i> | 177 |
| B.6. Diagrama de clases del visor | 178 |

| | |
|--|-----|
| B.7. Visor antiguo | 179 |
| B.8. Visor nuevo | 180 |
| B.9. Características del visor nuevo | 181 |
| C.1. Doxygen | 183 |

Escuela de Ingeniería Informática. Universidad de Las Palmas de G.C.

Capítulo 1

Introducción

La detección de contornos en imágenes es una de las operaciones previas para muchas tareas de más alto nivel como la segmentación, el registrado, la identificación de objetos o rincones, la reducción de ruido, etc. En muchas de estas aplicaciones, la invarianza por rotación y la alta precisión de los bordes son criterios importantes que permiten mejorar el resultado final. Durante su tesis, el profesor Agustín Trujillo Pino, ha desarrollado un método de detección nuevo que se basa en la hipótesis de efecto de volumen parcial para estimar los contornos con más precisión.

En este trabajo se describe la integración de los algoritmos de detección de bordes con alta precisión dentro del entorno de procesado y visualización de código abierto **AMILab**, con el objetivo de compartirlos con la comunidad científica. Asimismo, se han aportado algunas mejoras, así como el diseño de la interfaz dentro de la estructura de *scripts* de **AMILab** e incorporado documentación asociada a la misma.

En [16] se engloba a los proyectos informáticos en alguna de las cinco categorías siguientes: basados en la investigación, desarrollo, evaluación, colaboración con la industria y resolución de problemas.

El presente proyecto no se puede clasificar totalmente dentro de ninguna de las categorías anteriores. No es exactamente un proyecto de desarrollo, porque no existe una fase específica de análisis de requisitos a partir de un cliente, si no que se realiza la adaptación de unos métodos a un entorno software. Además, debido al carácter comparativo entre diferentes métodos, se puede considerar que contempla una fase de evaluación. Asimismo, se ha tocado de manera tangencial la investigación, debido a la participación en la **Thirteen International Conference on Computer Aided Systems Theory (EUROCAST 2011)**.

1.1. Objetivos

Para obtener estos resultados se plantearon como principales objetivos los siguientes:

- Entender el funcionamiento del algoritmo de detección de bordes sub-píxel en 2D y 3D: previamente al desarrollo fue preciso tener claro cómo funcionan los métodos propuestos en [42]. Para ello fue necesario realizar un análisis de los mismos, viendo cuál es la hipótesis de la que parten y cuál es el método propuesto para la detección a partir de ésta.
- Implementar estos algoritmos en el entorno del programa de código libre AMILab: AMILab es un entorno libre de procesamiento y visualización de imágenes, principalmente orientado al tratamiento de imágenes médicas. Pero además, incorpora su propio lenguaje de *scripts*. La idea principal fue incorporar los métodos dentro de alguno de los módulos internos del software. Posteriormente y, a través de un proceso de *wrapping* o encapsulado de código, se puso a disposición del lenguaje de *scripts* estas funcionalidades.
- Incluir una interfaz gráfica que permita al usuario probar el algoritmo y visualizar fácilmente el resultado: partiendo del desarrollo realizado en el punto anterior, es posible generar una interfaz gráfica desde el lenguaje de *scripts* de AMILab, que provee mecanismos para ello. Esto ha hecho que sea sencillo probar el método para un usuario. Es más, esto también es más sencillo para el propio desarrollador, ya que al tratarse de un lenguaje interpretado, no es necesario recompilar, si no que basta con recargar el *script* que se está desarrollando cuando se ha realizado alguna modificación.
- Comparar este algoritmo con otros algoritmos de detección de bordes de la literatura: de esta manera se se ha demostrado la aportación que dan estos métodos respecto a los clásicos de detección de bordes.
- Intentar identificar y proponer posibles mejoras al algoritmo existente.
- Hacer una página web que permita compartir el trabajo realizado con la comunidad científica.

1.2. Metodología

El desarrollo de código y de la interfaz han seguido y usado los entornos de programación de interfaz de usuario de AMILab, es decir, C++, wxWidgets y OpenGL. Además se ha hecho uso de la librería para visualización Visualization Toolkit (VTK), desarrollada por Kitware, Inc.

El desarrollo se ha realizado en una rama subversion de la rama principal de AMILab, incluyendo una documentación generada con la herramienta Doxygen. Se incluyen herramientas para poder probar los algoritmos de detección, como la generación de imágenes sintéticas.

Para la planificación se ha hecho uso de herramientas que usan la filosofía Getting Things Done (GTD) como por ejemplo Wunderlist. La coordinación se ha realizado mediante herramientas online, haciendo especial interés en reflejar en cada momento el estado actual del avance del proyecto, informándose así a todas las personas involucradas en el mismo.

Desde el punto de vista de la ingeniería del software, se ha hecho uso del modelado basado en prototipos. Existen varios modelos para llegar a la construcción final de un producto de software y optimizar el desarrollo del mismo. Es habitual que en un proyecto no se identifiquen

de entrada los requisitos de forma detallada, al igual que no se está seguro de la eficiencia de un algoritmo o de la forma en que se ha de implantar la manera de interactuar con el mismo.

La aproximación mediante el uso de prototipos consiste en realizar la fase de definición de requisitos del sistema en base a tres factores: un alto grado de iteración, un alto grado de participación del usuario y un uso extensivo de prototipos.

Durante la ejecución del proyecto se han realizado diferentes prototipos de manera que se han incluido en cada uno más funcionalidades, a la vez que se ha tratado de ir refinando el producto final.

Este modelo de desarrollo tiene sus pros y contras:

- **Ventajas:** reducción de la incertidumbre y del riesgo, reducción de tiempo y de costos, incrementos en la aceptación del nuevo sistema, mejoras en la administración de proyectos, mejoras en la comunicación entre desarrolladores y clientes, etc.
- **Desventajas:** la dependencia de las herramientas de software para el éxito ya que la necesidad de disminución de incertidumbre depende de las interacciones del prototipo. Entre más iteraciones existan mejor y esto último se logra mediante el uso de mejores herramientas, lo que hace a este proceso dependiente de las mismas. También, no es posible aplicar la metodología a todos los proyectos de software. Finalmente, existe una mala interpretación por parte de los usuarios del prototipo, al cual pueden confundir con el sistema terminado.

1.3. Recursos

El desarrollo del proyecto se ha visto centrado en la arquitectura **Macintosh**, pues existía interés de que haya una línea activa en este tipo de sistemas. Para ello, se ha usado un portátil **MacBook** con un procesador **Core 2 Duo** de 2,4GHz y 4Gb de memoria **RAM DDR2**.

Además se precisa de la información a procesar con los algoritmos que serán desarrollados. Es por ello que se ha dispuesto de distintas imágenes médicas de varias modalidades para hacer las pruebas de calidad.

1.4. Estructura del Documento

Es importante describir a grandes rasgos cuáles serán los puntos que se verán a lo largo del presente documento. En principio se hará un recorrido por todas las herramientas que se han usado a través del proceso de realización del proyecto. Dentro de estas herramientas se consideran aquellas asociadas tanto puramente al desarrollo como a edición y generación de la documentación. En relación a las primeras se hace un breve recorrido por algunos entornos, que fueron estudiados previamente antes de tomar la decisión final. Como es obvio, dentro de

este apartado no puede faltar una descripción del software dentro del que se han integrado los métodos de detección, **AMILab**.

La generación y edición de documentos gira principalmente alrededor de dos entornos: el usado para la creación del propio documento en \LaTeX y el necesario para la realización de gráficos, diagramas, dibujos, etc.

Una vez conocidas y descritas las herramientas, es necesario hacer un pequeño recorrido por el *background* de los métodos de detección a nivel píxel y vóxel. Es también en esta sección donde se introducen y explican los métodos de detección sub-píxel y sub-vóxel propuestos en [42].

Sabiendo el marco de trabajo y los métodos que se van a usar, se pasa a ver el desarrollo propiamente dicho y la integración dentro del software **AMILab**, así como el uso de dichos métodos desde el lenguaje de *scripts*. Se intenta seguir la misma estructura que en la descripción de los métodos, viendo primero el caso 2D y luego el 3D. Dentro de cada uno de estos dos bloques, se habla inicialmente del uso de los métodos tradicionales para luego pasar a los de precisión sub-píxel y sub-vóxel. En la parte final de cada una de estas dos secciones se encuentra la descripción del desarrollo de la interfaz en el lenguaje de *scripts*, así como algunos resultados tanto en imágenes sintéticas como reales.

Además de lo anterior, dentro del apartado de desarrollo se describe la metodología seguida para la generación de imágenes sintéticas. La idea es que, partiendo de un método ya existente en el software, plantear una solución para la creación de imágenes sintéticas con efecto parcial, independientemente del tipo de imagen. Esto se logra a través del uso de funciones que describen las imágenes, denominadas funciones analíticas.

Posteriormente en la sección de resultados se analizan principalmente tres aspectos. En primer lugar se habla de la generación de imágenes sintéticas y de cómo el nivel de subdivisión influye en la obtención de un nivel de intensidad adecuado para un píxel/vóxel borde. Seguidamente se pasa a realizar una comparativa entre los métodos de detección a nivel píxel y sub-píxel. En este apartado, además de lo anterior, se describe brevemente el estudio de un caso práctico, con el cual se participó en la conferencia **EUROCAST**.

Por último, el contenido de la memoria termina con las conclusiones y una serie de posibles líneas futuras que se podrían seguir para la ampliación y mejora del proyecto.

En la parte de anexos se incluyen manuales de usuario, tanto para realizar la compilación de **AMILab** en entornos **Macintosh**, como pequeñas guías para el uso de los *scripts* desarrollados. También se incorpora un apartado con algunas mejoras realizadas en el entorno durante la realización del trabajo, así como una sección con detalles de implementación.

Finalmente, existe un pequeño glosario a disposición del lector, con la definición de algunos de los términos usados a lo largo del documento.

1.4.1. Algunas Convenciones de Estilo

Con el objetivo de dar cierta uniformidad al documento se siguen una serie de criterios que se describen a continuación [31].

El diccionario de la Real Academia de la Lengua Española (R.A.E.) reconoce la palabra *píxel*, acentuada, como sustantivo para la unidad mínima de representación digital de una imagen bidimensional. Su plural se forma añadiendo *-es* al final. Por congruencia con esto, aunque la palabra *vóxel* no aparece en el citado diccionario, se ha considerado utilizar las mismas reglas durante la redacción de la presente memoria.

Las palabras pertenecientes a otros idiomas y no reconocidas en el castellano se muestran en cursiva, haciendo uso del entorno `\emph` de L^AT_EX. Ejemplo de esto son *script* o *checkbox*.

Los nombres de ficheros, rutas o directorios, funciones, métodos, tipos, variables, objetos y clases se escriben en **letra de máquina de escribir (typewriter)** cuando son parte del texto. En las figuras y gráficos, por uniformidad, se usa la misma fuente para todas las etiquetas (generalmente una fuente palo seco).


Los nombres de entornos, programas, lenguajes y librerías se escriben en **letra palo seco (sans serif)**.

Todas las figuras (en su escala 100 %) cuentan con etiquetas distanciadas aproximadamente medio centímetro de su borde exterior (salvo problemas de espacio, como es el caso de la figura B.9 que se encuentra en los anexos) en **letra palo seco**. Cada etiqueta está asociada a la imagen a través de una línea que la une a un cuadro relleno de un color, con una opacidad del 25 % para éste.

Las imágenes 2D con resultados generalmente van acompañadas de un *zoom* en alguna región, para permitir ver correctamente la detección.

Capítulo 2

Herramientas Utilizadas

n este capítulo se realizará una descripción de las diferentes herramientas que han sido usadas a lo largo del proyecto. En este sentido, se tienen dos secciones diferentes. Inicialmente, se detallarán cuáles han sido las herramientas de desarrollo que se han usado. Esto engloba tanto aquellas que han sido necesarias para ello como la descripción de la herramienta de software libre **AMILab**, dentro de la cual se ha realizado la implementación.

En la segunda parte se describirán las distintas herramientas usadas para redactar la documentación y el diseño de los gráficos e imágenes que acompañan a la misma.

2.1. Herramientas de Desarrollo

Existía un interés inicial en incorporar una nueva línea de desarrollo en el software **AMILab**. Aunque se contaba con material para el lanzamiento de las versiones *release* en todas las plataformas (tanto **Windows** como **Linux** y **Mac**) no existía un desarrollo como tal dentro del entorno **Mac OS X**. Pero, una vez tomada esta dirección, queda decidir cuál será finalmente el entorno de desarrollo que será usado en dicha plataforma. Es por ello que se hablará de algunas herramientas, tratando de reflejar la decisión que finalmente hiciese utilizar el entorno de desarrollo **Xcode**.

2.1.1. **Code::Blocks**

Code::Blocks es un entorno de desarrollo multiplataforma libre y de código abierto que soporta compiladores de múltiples lenguajes incluidos **GCC** y **MSVC**. Está desarrollado en **C++** usando como herramienta para la interfaz de usuario **wxWidgets**. Al utilizar una arquitectura de *pluggings*, sus capacidades y características vienen definidas por los mismos. Actualmente, **Code::Blocks** está orientado a **C** y **C++**. Puede ser también usado para crear programas y apli-

caciones en ARM, AVR, D, DirectX, FLTK, GLFW, GLUT, GTK+, Irrlicht, Lightfeather, MATLAB, Ogre, OpenGL, Qt (framework), SDL, SFML, STL, SmartWin y wx. Aunque, en algunos casos, el SDK o entorno respectivo requieren de instalación para desarrollar en una tecnología específica. Code::Blocks se está desarrollando para Windows, Linux y Mac OS X [4].

El principal problema con Code::Blocks al usarlo en Mac OS X es que no es totalmente estable. Es por esta razón que en ocasiones tiene un comportamiento anómalo o, al menos, inesperado. Por ejemplo, algo tan simple como cambiar el tipo y tamaño de fuente del editor no estaba permitido (puesto que al darle a dicho botón se abría y cerraba automáticamente la ventana para hacer el cambio). Este hecho se encuentra justificado por parte de los desarrolladores, ya que no se cuenta con personal suficiente para trabajar en compatibilidades con Mac. No obstante, una vez más avanzado el proyecto volvió a probarse por curiosidad una nueva versión de este entorno de desarrollo, y cabe decir que ha sido bastante mejorado. A pesar de ello, quedan ciertos detalles de interfaz que pulir además de pequeños bugs como uno relacionado con la creación de ámbitos de bloque en C++ mediante el uso de llaves, ya que no permite poner abre llave.

2.1.2. KDevelop

KDevelop es un entorno de desarrollo integrado de código libre para sistemas GNU/Linux y otros sistemas Unix, publicado bajo licencia GPL, orientado al uso bajo el entorno gráfico KDE, aunque también funciona con otros entornos, como GNOME. Depende de GCC para crear código binario. KDevelop ofrece todas las comodidades de los IDEs modernos. Para proyectos y aplicaciones grandes, la característica más importante es que KDevelop es capaz de entender C++: parsea todo el código y recuerda qué clases tienen qué funciones miembro, dónde están definidas las variables, cuáles son sus tipos, y muchas otras cosas sobre el código. Pero KDevelop no es sólo un editor inteligente de código; hay otras cosas que hace bien. Obviamente, resalta el código fuente en diferentes colores, tiene un indentado personalizable, tiene una interfaz integrada con el depurador GDB de GNU, puede mostrar la documentación para una función si se mantiene el puntero sobre un uso de ella, puede tratar con diferentes tipos de entornos de construcción y compiladores (por ejemplo con proyectos basados en make y en CMake), y muchas otras funcionalidades [6].

Mac OS X permite la instalación de este tipo de software mediante herramientas que descargan el código fuente, resuelven las dependencias y realizan la compilación del mismo. Dos de estas herramientas son MacPorts o Fink. El problema que surgió con este entorno es que la última versión del sistema operativo para Mac (Snow Leopard¹) daba problemas al intentar realizar la compilación de las librerías de KDE necesarias para la instalación de KDevelop. Estaba previsto que para la siguiente versión (la 4.0), este entorno de desarrollo fuese compatible con más plataformas, a saber, Windows y Mac OS X. Finalmente parece que sólo han sacado una versión compatible con el sistema operativo Windows y para Mac OS X continúa la posibilidad anteriormente comentada y que en este caso se ha visto que no era viable.

¹En el instante en que se escribe este documento ya hay una nueva versión del sistema operativo: Mac OS X Lion

2.1.3. Xcode

A causa de las razones descritas en las dos secciones previas, se tomó la decisión de hacer uso de la herramienta de desarrollo de Apple, Xcode. Xcode es el entorno de desarrollo integrado de Apple Computer usado para escribir, construir, y probar programas que se ejecutan en el sistema operativo Mac OS X. Usando Xcode, se pueden crear una amplia variedad de aplicaciones, complementos, controladores y entornos usando diferentes tecnologías y lenguajes. Incluye herramientas para diseñar, editar, analizar, depurar, probar, empaquetar y distribuir los proyectos. Puede ser usado por un desarrollador independiente o para colaborar con un grupo de desarrolladores.

Un IDE es el “pegamento” que mantiene unido y ayuda a manejar todas las que pequeñas piezas que se necesitan para producir un software moderno. Es completamente posible editar, compilar, enlazar, empaquetar y probar el software sin usar un IDE. De hecho, así ha sido como el software se ha desarrollado y lo sigue siendo, ocasionalmente. Se pueden editar los ficheros con un editor, guardarlos, ejecutar un compilador para compilar el fuente, ejecutar un enlazador para enlazar los ficheros objeto dentro de un programa y entonces arrancar un depurador para probarlo. La mayoría de estas herramientas tienen muy poca o ninguna interfaz de usuario, siendo controladas casi exclusivamente desde la línea de comandos. Como las aplicaciones comenzaron a ser más complejas, lo mismo sucedió con las herramientas necesarias para crearlas. Incluso un proyecto “simple” podría emplear una docena de herramientas diferentes para crear una aplicación. Usando las herramientas correctas, ejecutándolas en el orden correcto con los argumentos correctos, el manejar los cientos de ficheros intermedios producidos durante este proceso, haciéndolo de forma repetida y consistentemente, se convierte en algo tedioso para la mayoría de los desarrolladores.

El entorno de desarrollo Xcode pone un aspecto conveniente, amigable y gráfico en todas las herramientas necesarias para desarrollar un proyecto. En lugar de tratar con carpetas y ficheros individuales, Xcode presenta una ventana de proyecto con los iconos representando las partes de que está compuesto. En lugar de mantener macros y reglas de compilación, Xcode proporciona una lista de preferencias de construcción. En lugar de luchar con la sintaxis de los comandos del depurador, Xcode remarca la línea del código fuente que la aplicación está ejecutando y muestra el estado de sus variables, convenientemente formateadas en una tabla. Simplemente arrastrando iconos se pueden hacer complejos cambios estructurales en los proyectos.

Es preciso tener en cuenta que Xcode no “hace” realmente mucho del trabajo actual en la consecución de un producto final. No compila el código fuente, no construye los ficheros NIB, no enlaza el código, no ensambla la aplicación en un paquete y no tiene el control del código durante la depuración. Lo que Xcode hace es mantener juntas todas las herramientas que actualmente hacen este trabajo y las hace parecer una sola [14].

Por tanto, las herramientas de desarrollo de Xcode se combinan para dar un un entorno de desarrollo de calidad profesional. Contiene todas las herramientas que se necesitan para manejar un proyecto, desarrollo de código, diseño de interfaz de usuario, depuración, manejo de revisiones, pruebas de unidad, monitorización de prestaciones y empaquetado [33].

En la figura 2.1 se puede ver una captura de este entorno de desarrollo. A esa ventana se le

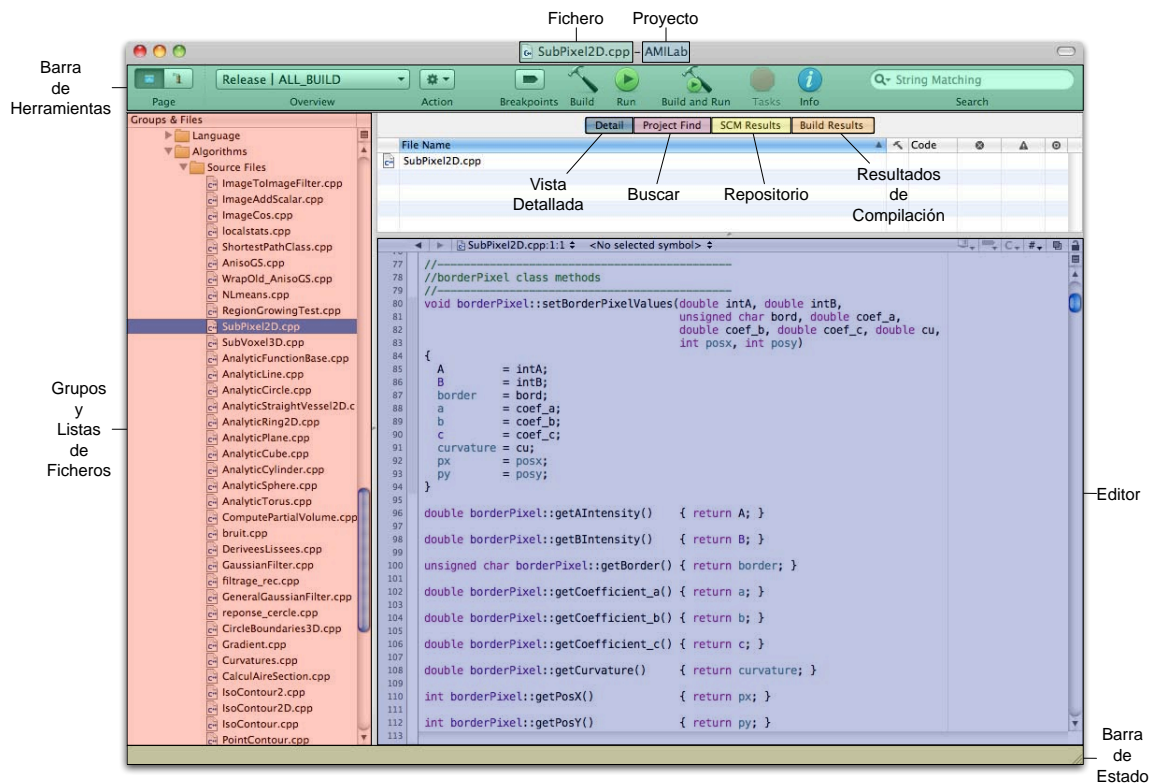


Figura 2.1: Captura de pantalla del entorno de desarrollo de Apple Computer, Xcode

denomina ventana de proyecto y es el control central de un proyecto en Xcode. Viendo más en detalle los componentes de la misma tenemos:

- La barra de herramientas: contiene los botones y otros controles que se pueden usar para realizar tareas comunes. Los dos botones situados más a la izquierda etiquetados con el texto *Page*, permiten seleccionar entre la vista habitual del proyecto y la vista en modo depuración (véase la figura 2.2). El menú desplegable *Overview* permite seleccionar el kit de desarrollo, la configuración, el objetivo, el ejecutable y la arquitectura. El botón *Action* muestra un menú de tareas que se pueden realizar con el ítem seleccionado en el árbol de la izquierda. A continuación aparece el botón *Breakpoints* que permite activar o desactivar los puntos de interrupción que se hayan colocado a lo largo del código ².

Posteriormente se pueden ver los botones de *Build* y de *Run*. Estos botones no aparecen por defecto en la barra de herramientas, si no que aparece directamente *Built and Run*. Se ha personalizado de esta manera para mayor comodidad del desarrollador. Independientemente de ello, la función de los dos primeros es por un lado hacer la compilación y enlazado (*Build*), y la ejecución de la aplicación (*Run*). El botón que aparece por defecto aglutina todas estas operaciones en un sólo elemento de la barra.

²Acerca de los puntos de interrupción o de ruptura: en Xcode colocar un punto de ruptura es tan simple como hacer clic con el botón izquierdo en la región gris izquierda del editor de código. Pueden ponerse tantos como se desee y sólo estarán activos cuando se active el mencionado botón de la barra de herramientas. Para eliminarlos sólo hay que pinchar en ellos y arrastrarlos fuera, o haciendo clic con el botón derecho encima de ellos y seleccionando la opción *Remove breakpoint* del menú contextual.

El botón *Tasks* permite detener cualquier operación que se encuentre en proceso, por ejemplo la compilación o la ejecución del programa. *Info* abre una ventana que muestra información y preferencias para los grupos, los ficheros, y los objetivos en el proyecto. Se pueden cambiar bastantes preferencias desde la ventana *Info*. Por último, pero no menos importante, en la barra de herramientas se encuentra el campo *Search*. Este elemento permite filtrar los ítems que se estén mostrando actualmente en la vista detallada (*detail view*).

Como ya se ha indicado anteriormente, es posible personalizar el contenido de esta barra de herramientas. Para ello, no hay más que hacer clic con el botón derecho en un espacio libre de la misma y seleccionar *Customize Toolbar* en el menú desplegable que aparece. Sólo bastará arrastrar aquellos nuevos elementos que se quieren mostrar hacia la barra.

- Grupos y listas de ficheros: los grupos organizan los ficheros y la información en un proyecto. Su propósito es ayudar a organizar el proyecto y permitir encontrar rápidamente componentes e información. Se pueden personalizar algunos de los grupos que aparecen por defecto en la lista e incluso definir grupos propios.

Xcode provee algunos grupos como *Targets*, *Executables*, *Bookmarks* y *SCM* (*source control management*). El primer grupo lista siempre el grupo del proyecto, donde se organizan todos los componentes necesarios para construir el programa. Normalmente contiene subgrupos con ficheros fuente del proyecto, ficheros de recursos, entornos y productos.

A la derecha aparece la vista detallada. Es una lista detallada de todos los ítems que están actualmente seleccionados en la lista de grupos y ficheros. Se puede buscar y ordenar rápidamente los ítems en la vista detallada, obteniendo un acceso rápido a información importante del proyecto.

- Editor de texto: además de mostrar el contenido que se está editando, el editor de texto proporciona una barra de navegación en su parte superior, además de herramientas para colapsar o expandir el código, o centrar el foco en cierta parte del contenido (oscurece el resto del contenido manteniendo sólo destacada la región delimitada por el ámbito de las llaves -caso de C++-). Se puede ver y editar un fichero fuente sin la ventana de proyecto haciendo doble clic sobre el fichero deseado en la lista de grupos y ficheros. Sin embargo, si se desea verlo dentro de la misma ventana del proyecto, basta con hacer un único clic.

En el margen izquierdo se muestran además los números de línea, así como información sobre la localización en el fichero de puntos de interrupción (*breakpoints*), errores o avisos. La herramienta para centrar el foco ayuda a concentrar la atención en partes del código mediante la identificación del ámbito de un bloque de código, además de permitir ocultar y mostrar bloques del mismo.

La barra que se encuentra en la parte superior del editor contiene varios menús y botones que permiten moverse entre ficheros que han sido vistos, saltar entre símbolos y cambiar a otros ficheros abiertos.

Al igual que otros editores propone sugerencias en el momento en que se está escribiendo código, evitando así que el usuario tenga que escribir los nombres extensos de algunos métodos. Además de ello, permite que una vez que se haya seleccionado una sugerencia (en el caso de llamar a una función o procedimiento) se pueda navegar a través de los argumentos usando la tecla tabulador, para ir completando los parámetros que se usarán para la llamada.

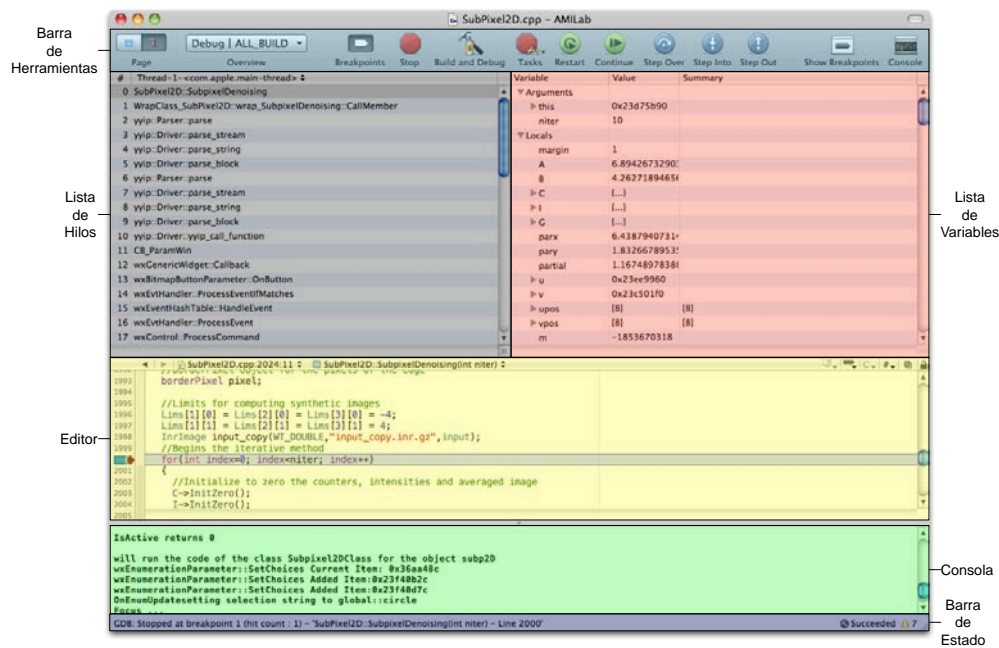


Figura 2.2: Depurador proporcionado por el entorno de desarrollo Xcode

Además de todo ello, en la parte superior de la ventana del proyecto se encuentran en el centro y a la izquierda el nombre del fichero que se está editando en ese momento y, a la derecha separado por un guión, el nombre del proyecto en curso. En la parte inferior de esa misma ventana está la barra de estado, que informa entre otras cosas del avance del proceso de compilación, así como del número de errores y avisos que van surgiendo hasta el momento. Si se hace clic sobre alguno de esos dos números lleva a una lista con la descripción [22].

En la figura 2.2 se encuentra una descripción del depurador que proporciona el entorno de desarrollo Xcode. Dentro de sus principales características pueden encontrarse:

- Barra de herramientas: contiene los ítems usados para manejar el proyecto y controlar la ejecución del programa.
- Lista de hilos: muestra la pila de llamadas del hilo actual. El menú que se encuentra en la parte de encima permite seleccionar diferentes hilos, algo que es bastante útil cuando se intenta depurar un aplicación multihilo.
- Lista de variables: muestra las variables definidas en el ámbito actual así como sus valores. Esta sección muestra además el estado actual de los registros del procesador cuando está activa la vista desensamblada.
- Editor: este panel muestra el código fuente que se está ejecutando. Cuando la ejecución del programa está pausada, el depurador indica la línea en la cual la ejecución se ha parado mostrando el indicador PC, el cual aparece como una flecha roja. La línea de código también se encuentra resaltada. Se puede cambiar el color usado para resaltar el estado actual de ejecución con la instrucción *Pointer Highlight* (en Xcode → *Preferences* → *Debugging*). Además si se mantiene el puntero del ratón quieto sobre alguna variable

aparece un menú emergente que muestra información acerca de la misma (tipo y valor tomado).

- Consola: en la consola se van mostrando las diferentes salidas que se tengan del proceso de depuración, así como los diferentes mensajes que se tengan indicados por la salida estándar en el código fuente.
- Barra de estado: muestra el estado de la sesión de depuración. Por ejemplo, en la figura 2.2 la barra de estado muestra que la ejecución está detenida en un punto de interrupción, así como otra información, como el número de línea.

Se puede cambiar el aspecto de la interfaz del depurador mediante las opciones que se encuentran en el menú *Run* → *Debugger Display*, como por ejemplo que la configuración visual sea vertical en lugar de horizontal.

Es importante destacar que para poder usar el depurador es preciso haber compilado generando símbolos de depuración. Esta generación se realiza por defecto cuando se compila el proyecto en modo depuración (*debug*). Si esta opción está activa, el compilador genera toda la información de depuración necesaria [23].

2.1.4. AMILab

AMILab es un software diseñado para el procesamiento de imagen en general, con especial énfasis en el procesamiento y visualización de imágenes 2D y 3D. Está orientado al uso académico, y puede ser usado para propósitos de investigación y educacionales. Una de las particularidades de AMILab es su capacidad para combinar convenientemente interfaces para la visualización de imágenes y mallas con un potente lenguaje de *scripts* orientado a objetos. Entre otros programas relacionados están: 3D Slicer, Osirix, Matlab, Octave, SCIRun, Mevislab, Gimp, por citar algunos. Cada uno de estos programas proporciona diferentes características, como herramientas de visualización para 2D, 3D o series temporales de imágenes, visualización de un conjunto de datos poligonales 3D, algoritmos de procesamiento de imágenes, la posibilidad de usar lenguajes de *scripts* como *python*, *matlab*, *perl* u otros lenguajes de *scripts*, y características para crear interfaces gráficas fácilmente. En general, cada uno de estos programas tiene sus propias ventajas e inconvenientes y son los más adecuados para ciertas tareas. Véase a continuación una breve descripción de cada uno de ellos.

- 3D Slicer: 3D Slicer es una plataforma flexible que puede ser extendida fácilmente para permitir el desarrollo tanto de herramientas interactivas, como por lotes, para una variedad de aplicaciones. Dentro de sus funcionalidades se encuentran el registro de imágenes, procesamiento de tractografía, una interfaz a dispositivos externos para soporte guiado por imagen, renderizado volumétrico por GPU, entre otras capacidades. 3D Slicer tiene una organización modular que permite la adición de nuevas funcionalidades de forma sencilla y provee un número de características genéricas no disponibles en herramientas de la competencia. Finalmente, 3D Slicer está distribuido bajo una licencia BSD no restrictiva [10].

- **Osirix:** Osirix es un programa gratuito disponible al público en el sitio web de *Apple Inc.* Los visualizadores biomédicos pueden usar este software para visualizar conjuntos de datos anatómicos y extraer información visual de referencia [28]. Osirix ha sido específicamente diseñado para la navegación y visualización de imágenes multimodales y multidimensionales: visor 2D, visor 3D, visor 4D (series 3D con dimension temporal) y visor 5D (series 3D con dimensiones temporales y funcionales). El visor 3D ofrece todos los modos de renderizado modernos: reconstrucción multiplanar, renderizado de superficie, renderizado volumétrico y proyección de máxima intensidad. Osirix es simultáneamente una estación DICOM de sistema de archivado y transmisión de imágenes (PACS) para imágenes médicas y un paquete software de procesamiento de imágenes para investigación médica (radiología e imágenes nucleares), imágenes funcionales, imágenes 3D, microscopía confocal e imágenes moleculares [8].
- **Matlab:** Matlab es un lenguaje interpretado de programación, un conjunto de reglas para escribir programas. Matlab es un lenguaje de programación orientado al Cálculo Numérico (de ahí su nombre Matrix Laboratory). Por motivos de licencias, Matlab se encuentra dividido en paquetes (o *toolboxes*) donde cada uno cumple una función específica y puede ser adquirido aparte [20].
- **Octave:** Octave se describe como un lenguaje de programación de alto nivel orientado al Cálculo Numérico. Proporciona una consola para resolver problemas lineales y no lineales. Puede ser copiado, modificado y redistribuido libremente bajo los términos de la licencia GNU GPL tal como se publica por la Free Software Foundation. Se diseñó inicialmente para ser una herramienta de la línea de comandos, aunque posteriormente ha sido portado a muchos sistemas operativos. Era un lenguaje independiente hasta ir convergiendo a Matlab, llegando a buscar la compatibilidad con él. A diferencia de Matlab que está escrito en C, Octave está escrito en C++ [20].
- **SCIRun:** SCIRun es un entorno de resolución de problemas basado en una programación de flujo de datos modular. Tiene un conjunto de módulos que realizan funciones específicas de un flujo de datos. Cada módulo lee datos desde los puertos de entrada, calcula los datos, y envía nuevos datos hacia los puertos de salida [40].
- **MeVislab:** Mevislab representa una plataforma para la investigación y desarrollo de procesamiento de imágenes enfocado a imágenes médicas. Permite una rápida integración y prueba de nuevos algoritmos y el desarrollo de prototipos de aplicación que puede ser usado en entornos clínicos. Incluye módulos avanzados para la segmentación de imágenes médicas, registrado, volumetría, y morfología cuantitativa y análisis funcional. Varios prototipos clínicos han sido realizados en la base de MeVislab, incluyendo asistentes software para neuro-imágenes, análisis dinámico de imágenes, planificación quirúrgica y análisis de vasos [7].
- **Gimp:** GIMP es una herramienta de manipulación fotográfica multiplataforma. Es el acrónimo de GNU Image Manipulation Program. En él se pueden realizar todo tipo de tareas de manipulación de imágenes, incluyendo retoque fotográfico, composición y creación de imágenes. Es ampliable y extensible. Está diseñado para ampliarse con complementos y extensiones. Se puede automatizar a través de *scripting*, desde las tareas más simples, hasta los procedimientos más complejos de manipulación de imágenes [5].

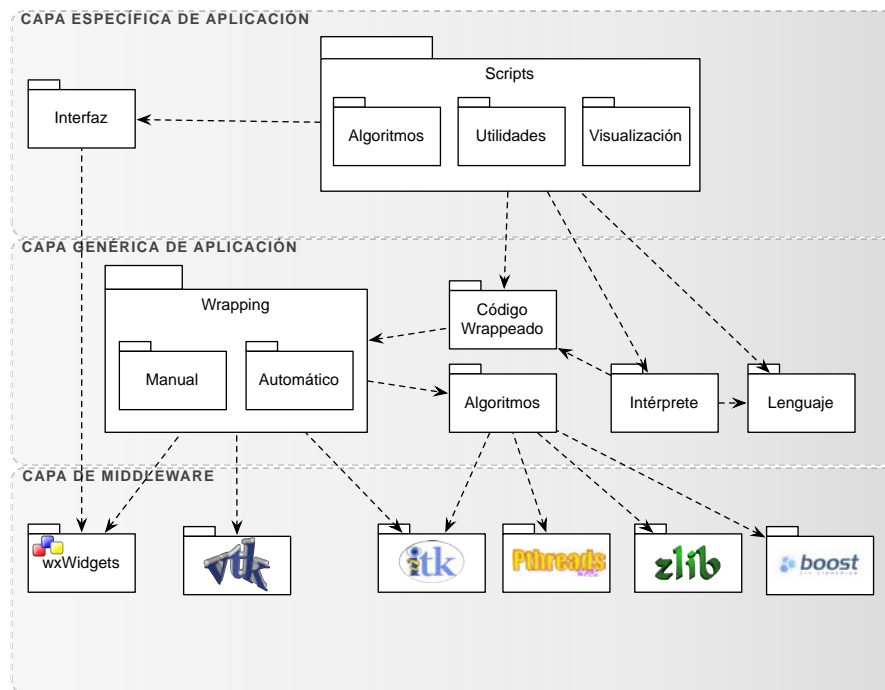


Figura 2.3: Diagrama de las capas de la arquitectura del software AMILab

AMILab ha sido diseñado usando C/C++, Flex y Bison, wxWidgets, OpenGL, VTK e ITK. En la figura 2.3 puede verse una descripción aproximada de la arquitectura del software AMILab. En la *Capa de Middleware* se encuentran los paquetes y subsistemas de la tecnología que son usados en la implementación. En la *Capa Genérica de Aplicación* están los paquetes de servicio y aquellos que son muy genéricos, o de los que dependen bastante subsistemas o paquetes. Destacar que en el caso del *wrapping* automático, éste se produce por demanda. Es decir, este proceso no se produce sistemáticamente a lo largo de todas y cada una de las clases que forman parte de la librería, sino que se hace solamente para las indicadas en el fichero `classes.txt` de la configuración del *wrapping* de cada una de ellas. Por último, en la *Capa Específica de Aplicación* están los paquetes y subsistemas que forman parte de la aplicación. Por simplicidad, en esta última capa se ha intentado hacer una abstracción del total de paquetes y subsistemas que existen en realidad dentro del software. A continuación se describirán brevemente algunas de las librerías utilizadas.

2.1.4.1. CMake

CMake es un sistema multiplataforma para automatizar el proceso compilación. Su nombre es una abreviatura de “cross platform make”. Es una aplicación independiente y de mayor nivel que el sistema `make` comúnmente usado para el desarrollo en Unix. Está constituido por una familia de herramientas diseñadas para compilar, probar y empaquetar software. Se usa para un control del proceso de compilación mediante el uso de una plataforma simple y un compilador de archivos que se configura de forma independiente. CMake genera *makefiles* nativos del sistema en

que se esté y proyectos que pueden ser utilizados en el entorno del compilador que se elija (por ejemplo Code::Blocks, Xcode y otros). Puede compararse con el sistema GNU Unix Autoconf desde la perspectiva de que el proceso de construcción está controlado por los archivos de configuración, en el caso de CMake, los denominados `CMakeLists.txt`. Como GNU Autoconf, no compila directamente el software final, sino que genera los ficheros de compilación estándar (por ejemplo, *makefiles* en Unix y proyectos/áreas de trabajo de Windows en Visual C++ o Eclipse CDT) que se utilizan en la forma estándar, lo que permite un enfoque más sencillo para los desarrolladores familiarizados con un entorno de desarrollo en particular. Este uso del entorno de compilación nativo es lo que distingue a CMake de sistemas como Autoconf.

A diferencia de muchos sistemas multiplataforma, CMake está diseñado para ser utilizado en conjunción con la versión nativa del entorno en el que se trabaja. Para generar los ficheros de compilación estándar utilizados habitualmente se usan archivos simples de configuración localizados en cada directorio del código fuente (llamados ficheros `CMakeLists.txt`). CMake puede compilar el código fuente, crear bibliotecas, generar contenedores y crear ejecutables en combinaciones arbitrarias. Soporta compilación dentro y fuera de lugar (*in-place and out-of-place building*), por lo que puede soportar múltiples construcciones a partir de un mismo árbol de código fuente. También se apoya en la biblioteca estática y dinámica. Otra característica interesante de CMake es que genera un archivo de caché que está diseñado para ser utilizado con un editor gráfico (CMakeGui). Por ejemplo, cuando se ejecuta CMake, se localizan e incluyen archivos, bibliotecas y ejecutables, y se pueden construir directivas opcionalmente. Esta información se recoge en la caché, que puede ser cambiada por el usuario antes de la generación de los archivos nativos. Además esta caché queda guardada, de forma que si es preciso reconstruir los ficheros de compilación, se tiene almacenada cuál fue la última configuración usada para el proyecto.

CMake está diseñado para soportar complejas jerarquías de directorios y aplicaciones que dependen de varias bibliotecas. Por ejemplo, soporta proyectos consistentes en múltiples conjuntos de herramientas (como librerías), donde cada uno puede estar contenido en varios directorios, y la aplicación depende de todo ello más su propio código adicional. CMake también puede manejar situaciones en las que los ejecutables deben ser construidos con el fin de generar código que se compila y enlaza en una aplicación final. Debido a que el código fuente de CMake es libre y abierto, teniendo un diseño simple y ampliable, puede extenderse según sea necesario para apoyar nuevas características.

Usar CMake es simple. El proceso de compilación es controlado por la creación de uno o más ficheros `CMakeLists.txt` en cada directorio (incluyendo subdirectorios) que conforman un proyecto. Cada `CMakeLists.txt` consta de uno o más comandos. Cada uno de ellos tiene la forma `COMMAND (args ...)` donde `COMMAND` es el nombre del comando, y `args` es una lista de argumentos separados por espacios en blanco. CMake proporciona bastantes comandos predefinidos, pero si es necesario, se pueden añadir nuevos. Además, un usuario avanzado puede agregar otros generadores *makefile* para una combinación particular compilador/sistema operativo [3].

2.1.4.2. ITK

Insight Toolkit (ITK: Insight Segmentation and Registration Toolkit) es un conjunto de herramientas software de código abierto para la realización de registrado y segmentación. La segmentación es el proceso de identificación y clasificación de datos que se encuentran en una representación digital muestreada. Típicamente, la representación muestreada es una imagen adquirida a través de dispositivos médicos como escáneres CT o MRI. El registrado es la tarea de alinear o desarrollar correspondencias entre datos. Por ejemplo, en entornos médicos, un escáner CT puede ser alineado con un escáner MRI con el fin de combinar la información contenida en ambos.

ITK está implementado en C++. Es multiplataforma, usando un entorno de construcción conocido como CMake para manejar el proceso de compilación de manera que sea independiente este proceso de la plataforma. Además, un proceso de *wrapping* automático (Cable) genera interfaces entre C++ y lenguajes de programación interpretados como Tcl, Java y Python. Esto permite a los desarrolladores crear software usando una variedad de lenguajes de programación. Al estilo de implementación C++ de ITK se le denomina como programación genérica, esto quiere decir que utiliza *templates* (plantillas) de manera que el mismo código pueda ser aplicado genéricamente a cualquier clase o tipo que soporte las operaciones usadas. El uso de *templates* C++ hace que el código sea altamente eficiente, ya que la mayoría de problemas software son descubiertos en tiempo de compilación, en lugar de en tiempo de ejecución.

Ya que ITK es un proyecto de código abierto, desarrolladores de todo el mundo pueden usarlo, depurarlo, mantenerlo y ampliarlo. ITK usa un modelo de software conocido como **Programación Extrema**. La **Programación Extrema** colapsa la metodología habitual de desarrollo de software en un proceso simultáneo e iterativo de diseño-implementación-prueba-lanzamiento (*design-implement-test-release*). Las características clave de la **Programación Extrema** son la comunicación y la prueba. La comunicación entre los miembros de la comunidad de ITK es lo que permite ayudar a mantener la rápida evolución del software. La prueba es lo que mantiene el software estable. En ITK, un proceso extenso de prueba (usando un sistema conocido como Dart) se encarga de medir la calidad sobre una base diaria. Los resultados de las pruebas en ITK son publicados continuamente (a través del ITK Testing Dashboard), reflejando la calidad del software en cualquier momento [21].

2.1.4.3. VTK

VTK (Visualization Toolkit) es un proyecto de código abierto bajo licencia BSD, para gráficos 3D por ordenador, procesado de imagen y visualización. VTK consta de una librería de clases C++ y algunas capas de interfaz interpretadas, incluyendo lenguajes como Tcl/Tk, Java y Python. Ha sido creado por Kitware y continúa extendiendo el conjunto de herramientas, ofreciendo soporte profesional y servicios de consulta. VTK soporta una amplia variedad de algoritmos de visualización incluyendo: métodos escalares, vectoriales, tensoriales, de texturas y volumétricos. Además de ello, también incluye técnicas de modelado avanzadas como: modelado implícito, reducción de polígonos, suavizado de malla, corte, contorneado y triangulación de Delaunay. Cuenta con un amplio entorno de visualización de la información, incluyendo una suite de *wid-*

gets 3D interactivos. Soporta procesamiento paralelo y se integra con varias bases de datos sobre la interfaz gráfica de usuario, tales como Qt y Tk. Es multiplataforma y funciona en Linux, Windows, Mac y Unix. Internamente VTK está implementado en C++, y requiere a los usuarios crear aplicaciones mediante la combinación de varios objetos en una aplicación. El sistema también soporta *wrapping* automático del núcleo C++ en Python, Java y Tcl, de modo que las aplicaciones de VTK también pueden ser escritas usando estos lenguajes de programación interpretados.

Para VTK se emplea el **Proceso de Calidad del Software de Kitware** (CMake, CTest, CDash y CPack) para compilar, probar y empaquetar el sistema. Hacer que VTK sea una aplicación multiplataforma depende de: un desarrollo basado en pruebas, la programación extrema y permitir los mecanismos necesarios para producir un código de alta calidad y robusto. VTK es usado en todo el mundo para aplicaciones comerciales, de investigación y desarrollo, y es la base de muchas aplicaciones de visualización avanzadas, tales como: ParaView, Visita, VisTrails, 3DSlicer, Mayavi y OsiriX [36].

2.1.4.4. wxWidgets

wxWidgets es un conjunto de herramientas de programación para desarrollar aplicaciones de escritorio o móviles con interfaz gráfica de usuario (GUIs). Es un sistema, en el sentido que se ocupa de la gestión interna y provee por defecto características de funcionamiento típicas de las aplicaciones. La librería wxWidgets contiene un gran conjunto de clases y métodos que el programador puede usar y personalizar. Las aplicaciones muestran típicamente ventanas con controles estándar, quizás imágenes y gráficos especializados y responden a eventos de ratón, teclado u otras fuentes. Además se comunican con otros procesos o conducen a otros programas. En otras palabras, wxWidgets hace relativamente fácil al programador desarrollar una aplicación que hace todas las cosas que habitualmente hacen las aplicaciones modernas.

Mientras wxWidgets es categorizada a menudo como un conjunto de herramientas para el desarrollo de interfaces de usuario, es de hecho mucho más que esto y sus características son útiles para muchos aspectos del desarrollo de aplicaciones. Ésta ha de ser la razón por la que todo lo que forma parte de una aplicación de wxWidgets ha de ser portable a diferentes plataformas, no sólo por la parte de interfaz. wxWidgets proporciona clases para ficheros y flujos, múltiples hilos, preferencias de las aplicaciones, comunicación entre procesos, ayuda en línea, acceso a bases de datos, y mucho más [37].

2.1.4.5. Pthreads

POSIX Threads o Pthreads es un estándar POSIX (Portable Operating System Interface, la X viene de UNIX como seña de identidad de la API) para hilos. Define una API para la creación y manipulación de hilos.

Pthreads define un conjunto de tipos desarrollados en C, funciones y constantes. Está implementado en el fichero cabecera `pthread.h` y una librería `thread`. Hay alrededor de cien pro-

cedimientos `Pthreads`, todos con el prefijo “`pthread_`” y pueden clasificarse en cuatro grupos diferentes:

1. Manejo de hilos: creación, unión de hilos, etc.
2. Mutex.
3. Variables condición.
4. Sincronización entre hilos usando cerrojos de lectura/escritura y barreras.

La API de los semáforos POSIX funciona con `Pthreads`, pero no es parte del estándar de hilos, viniendo definida dentro de `POSIX.1b` (estándar de extensiones de tiempo real). Consecuentemente los procedimientos de semáforos tienen el prefijo “`sem_`” en lugar de “`pthread_`” [9].

2.1.4.6. Zlib

`Zlib` es una librería software usada para la compresión de datos. Fue escrita por *Jean-Loup Gailly* y *Mark Adler*, y es una abstracción del algoritmo de compresión `DEFLATE` usado por su programa de compresión `gzip`. La primera versión pública, la 0.9, fue lanzada el 1 de Mayo de 1995 y fue originalmente diseñada para utilizarse con la biblioteca de imágenes `libpng`. Es software libre, distribuido bajo la licencia `zlib`.

Los datos comprimidos con `zlib` son escritos normalmente en paquetes `gzip` o `zlib`. El paquete encapsula los datos `DEFLATE` mediante la adición de una cabecera y un preámbulo. Esto proporciona una identificación y detección de errores que no son prestados por los datos `DEFLATE` en bruto.

El encabezado de `gzip` es mayor que el encabezado de `zlib`, ya que almacena un nombre de archivo y otros datos de sistema de archivos. Este es el formato utilizado en la cabecera de los archivos ubicuos `gzip`.

Actualmente `zlib` sólo admite un algoritmo llamado `DEFLATE`, que es una variación de `LZ77`. Este algoritmo proporciona una buena compresión en una amplia variedad de datos con un uso mínimo de recursos del sistema. Este es también el algoritmo usado hoy en día, casi invariablemente, en el formato de archivo `ZIP`. Es poco probable que el formato `zlib` tienda a utilizar cualquier otro algoritmo de compresión, aunque la cabecera tiene en cuenta esa posibilidad.

La biblioteca dispone de funcionalidades para el control del procesador y el uso de memoria. Puede suministrarse un valor de nivel de compresión, aunque eso variará la velocidad de procesamiento. También hay funcionalidades para la conservación de la memoria, pero probablemente sólo sea útil en entornos con memoria restringida, como algunos sistemas embebidos. `Zlib` permite además que la compresión pueda ser optimizada para tipos de datos específicos [12].

2.1.4.7. Boost C++

Boost C++ es una colección de bibliotecas de código abierto que amplían las funcionalidades de C++. Su licencia permite que sea utilizada en cualquier tipo de proyectos, ya sean comerciales o no. Varios fundadores de Boost pertenecen al comité ISO de Estándares C++.

Su diseño e implementación permiten que sea utilizada en un amplio espectro de aplicaciones y plataformas. Abarca desde librerías de propósito general (como la librería `smart_ptr`) hasta abstracciones del sistema operativo (como `Boost FileSystem`). Con el objetivo de alcanzar el mayor rendimiento y flexibilidad se hace un uso intensivo de plantillas (*templates*).

La versión actual de Boost contiene más de 80 bibliotecas individuales, incluidas bibliotecas para álgebra lineal, generación de números pseudoaleatorios, multihilos, procesamiento de imágenes, expresiones regulares, pruebas unitarias y muchas otras. La mayoría de las librerías de Boost están basadas en cabeceras, consistentes en funciones *inline* y plantillas, y como tal no necesita ser compilado antes de su uso [2].

2.2. Herramientas de Edición

2.2.1. L^AT_EX

Una de las ideas detrás de L^AT_EX es la separación entre la disposición y la estructura (en la medida de lo posible), en la cual se permite al usuario concentrarse en el contenido en lugar de preocuparse acerca de problemas de diseño [27].

L^AT_EX es un conjunto de sentencias escritas en un lenguaje de programación llamado T_EX, nombre que se adopta también para denominar al intérprete de dicho lenguaje o compilador. Pero T_EX, el verdadero motor creado por Donald Ervin Knuth entre los años 1977 y 1978, no es un lenguaje de programación usual, sino uno orientado a la escritura de textos de excelente calidad. No se trata, naturalmente, de un editor de textos al estilo de otros bien conocidos.

T_EX no es un editor de la familia WYSIWYG, término empleado para denominar a los editores que sólo trabajan sobre la pantalla, dando un formato visual al texto, y en los que cuáles “lo que ves es lo que tienes”. En T_EX se escribe el texto acompañado de órdenes que el compilador, posteriormente, interpreta y ejecuta para proporcionar un texto perfectamente compuesto.

Leslie Lamport creó L^AT_EX en 1982, con la intención de simplificar la tarea de aquéllos que desean utilizar L^AT_EX.

T_EX es lo que se podría denominar un “maquetador” o “compositor” de textos. Para poder comprender lo que esto significa, entender su portabilidad entre las diferentes plataformas (una cualidad sumamente interesante) y poder actuar eficazmente sobre él, es necesario conocer su funcionamiento y los ingredientes básicos del mismo. T_EX posee tres etapas básicas (texto

fuente, composición y visionado o impresión) que configuran el proceso.

2.2.1.1. Texto fuente

En cualquier plataforma informática, con cualquier editor de textos capaz de producir un fichero “sólo texto”, se escribe un documento en el que, además de texto propiamente dicho, se introduce, de acuerdo con una determinada sintaxis, información sobre la estructura final que se desea para el mismo. Supondremos almacenado este documento en un fichero de nombre `MiDoc.tex`, al que se le ha asignado la extensión `tex`, que si bien no es obligatoria, si es la usual y aconsejable. Estos ficheros “sólo texto” son transportables entre sistemas informáticos, están perfectamente adaptados a Internet y pueden ser leídos y modificados por los diferentes editores.

2.2.1.2. Composición o compilación

El texto fuente se procesa para darle formato y componerlo. El sistema realiza la tarea que correspondería a un impresor de una imprenta Gutenberg, es decir, aprende del autor lo que éste quiere hacer y se ocupa de componer líneas y páginas, dar formato a capítulos, secciones, notas a pie, índice general, etc. La composición de los documentos descansa en los siguientes dos elementos principales:

- El núcleo central de $\text{T}_{\text{E}}\text{X}$, llamado compilador en argot informático, que es un programa capaz de ejecutar tareas que le son ordenadas mediante un “lenguaje” lo suficientemente rico y flexible para poder resolver casi cualquier situación que se puede presentar en esta materia.

El contrapunto a esta flexibilidad es, en opinión de algunos, una cierta complejidad de manejo. Por ello, al poco tiempo de crearse la herramienta se comenzaron a desarrollar intérpretes con la finalidad de facilitar la comunicación del que escribe el texto fuente con el compilador $\text{T}_{\text{E}}\text{X}$.

- El formato $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$, creado por Lesli Lamport, es, en su actual versión, el intérprete más versátil porque, a sus ya importantes capacidades, ha incorporado las herramientas más destacadas de otro intérprete famoso, $\text{AMS-T}_{\text{E}}\text{X}$, desarrollado por la Sociedad Matemática Americana (American Mathematical Society) y descrito en el libro de Michael Spivak [38]. $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$, además de facilitar la comunicación con el motor central, tiene un enorme conocimiento sobre la forma de presentar un texto con la mejor calidad profesional y evita al autor del texto fuente tener que tomar decisiones sobre el formato preciso con el que quiere que aparezca su trabajo, permitiéndole concentrarse en el contenido del documento y las estructuras básicas del mismo.

Al compilar el texto fuente `MiDoc.tex` se produce un nuevo fichero, llamado `MiDoc.dvi`, que contiene toda la información “cajística” para imprimir el texto final, pero en el cual los tipos (las fuentes) podría decirse que están todavía sin entintar, son “blancos”. Este fichero

tiente también un cierto carácter de transportabilidad entre plataformas informáticas como ocurría con el fichero fuente; cualquier programa de visionado o impresión para \TeX ha de permitir obtener de él una copia en la pantalla del ordenador o en la impresora, utilizando la “tinta” adecuada, sin importar cuál sea la plataforma informática o los modelos de pantalla o impresora. La extensión `dvi` hace referencia a este carácter de independencia respecto del periférico (**d**evice **i**ndependent).

Durante la compilación con \LaTeX se producen también otros ficheros que son herramientas auxiliares para conseguir el objetivo final (`MiDoc.aux`), o bien contienen información técnica sobre el propio proceso de compilación (`MiDoc.log`).

2.2.1.3. Visualización o impresión

Para poder ver en pantalla e imprimir el resultado de la composición que \LaTeX ha realizado es necesario “entintar” los tipos del fichero `dvi` de salida. De ello se ocupan programas específicos diseñados con esa finalidad, de los que existe una gran variedad en cada plataforma informática. El mismo fichero (por ejemplo, `MiDoc.dvi`), y, por tanto, la misma composición, puede imprimirse (o verse en pantalla) con diferentes niveles de acabado, dependiendo de las capacidades de la impresora (o monitor) y de la “finura de entintado” o resolución que el programa de impresión (o visualización) utilice [34].

2.2.1.4. Estructura de un Documento en \LaTeX

\LaTeX puede ser usado para varios propósitos, como para escribir un artículo o una carta, o para producir transparencias. Obviamente, documentos para diferentes propósitos necesitan diferentes estructuras lógicas, diferentes comandos y entornos. Se dice que un documento pertenece a una clase de documentos que tienen la misma estructura general (pero no necesariamente la misma apariencia tipográfica). En un documento \LaTeX se especifica a que clase pertenece un documento empezando con el comando `\documentclass`, donde el parámetro obligatorio especifica el nombre de la clase del documento. La clase del documento define los comandos lógicos disponibles y los entornos (por ejemplo, `\chapter` para la clase `report`) así como un formato por defecto para esos elementos. Un argumento opcional permite modificar el formato de esos elementos a través una lista de opciones de la clase. Por ejemplo, `11pt` es una opción reconocida por la mayoría de las clases de documentos y le dice a \LaTeX de seleccionar once puntos como el tamaño básico de la fuente para el documento.

Muchos comandos no son específicos de una sola clase de documento, pudiendo usarse en clases diferentes. A una colección de estos comandos se le denomina paquete y se puede informar a \LaTeX del uso de ciertos paquetes en el documento poniendo uno o más comandos `\usepackage` después de `\documentclass`.

Al igual que `\documentclass`, `\usepackage` tiene un argumento obligatorio consistente en el nombre del paquete y un argumento opcional que puede contener una lista de opciones del paquete que modifican su comportamiento.

Las clases de documentos y los paquetes residen en ficheros externos con las extensiones `cls` y `sty`, respectivamente. El código para las opciones está a veces guardado en ficheros externos (en el caso de ficheros de clases con la extensión `.clo`) pero está normalmente especificado de manera directa en el fichero de la clase o del paquete. Sin embargo, en caso de opciones, el nombre de fichero puede ser diferente al nombre de la opción. Por ejemplo, la opción `11pt` está relacionada con el fichero `size11.co` cuando se usa la clase artículo (`article`) y al fichero `bk11.co` cuando se trata de la clase libro (`book`).

Los comandos situados entre `\documentclass` y `\begin{document}` se les denomina **preámbulo del documento**. Todos los parámetros de estilo deben ser definidos en este preámbulo, tanto en los paquetes como en los ficheros de clases, o directamente en el mismo documento antes del comando `\begin{document}`, donde se asignan los valores de algunos de los parámetros globales.

Generalmente, los paquetes \LaTeX no estándar contienen modificaciones, extensiones o mejoras con respecto al estándar \LaTeX , mientras que los comandos en el preámbulo definen cambios del documento de actual. En consecuencia, para modificar el aspecto de la salida de un documento, se tienen varias posibilidades:

- Cambiar las preferencias estándar de los parámetros en un fichero de clase por opciones definidas para esa clase.
- Añadir uno o más paquetes al documento y hacer uso de ellos.
- Cambiar las preferencias estándar de los parámetros en un fichero de paquete por opciones definidas por ese paquete.
- Escribir sus propios paquetes locales conteniendo preferencias de parámetros especiales y cargarlos con `\usepackage`.
- Hacer ajustes finales dentro del preámbulo [29].

Por tanto, queda claro que se denomina “preámbulo” del documento a la parte del documento fuente que precede a la orden de inicio del texto ordinario `\begin{document}`, y cuerpo a la parte comprendida entre `\begin{document}` y la orden `\end{document}` con la que finaliza el texto que el compilador considerará. De hecho, todo lo que siga a `\end{document}` será ignorado.

En resumen, en el “preámbulo” se deben escribir todas las declaraciones que afectan a todo el documento y su primera línea debe ser la declaración de la clase de documento. Además, junto a la declaración de la clase de documento se especifican opciones para el tamaño del papel a utilizar y el tamaño de la fuente. Así mismo, como se citó con anterioridad, han de incluirse además en esta región las declaraciones de los paquetes que se quieren utilizar [34].

2.2.1.5. BIB_TE_X

BIB_TE_X es un programa diseñado originariamente por Oren Patashnik ³ para generar de forma automática, en combinación con L^AT_EX, el entorno `thebibliography` a partir de los comandos `\cite` de nuestro documento y de bases de datos y estilos de bibliografía creados para BIB_TE_X. De esta manera se logra:

- **Consistencia** en la presentación de la lista de referencias. Por ejemplo, los nombres de los autores serán todos abreviados o mantenidos completos simultáneamente; algo similar para el uso de la fuente itálica en los títulos de artículos o libres, etc.
- **Facilidad** para poder cambiar el formato de las citas en el texto y la ordenación de la lista bibliográfica. Por ejemplo, se puede elegir el orden alfabético, el orden de aparición en el texto, etc., con sólo cambiar una línea en el documento L^AT_EX.
- **Economía** en el mantenimiento de nuestras referencias bibliográficas. Una base de datos creada para BIB_TE_X podrá ser utilizada en tantos documentos L^AT_EX como se quiera.

Una base de datos para BIB_TE_X es un fichero **BIB** en formato ASCII en el que se utilizan determinados convenios para indicar lo que va a ser un **registro** (libro, artículo, etc.) de la base de datos y cuáles son los **campos** (autor, título, año, etc.) dentro de los registros.

Si el documento fuente que se tiene es `MiDoc.tex` y la base de datos de bibliografía es `MiBase.bib`, para producir de forma automática las citas bibliográficas con L^AT_EX y BIB_TE_X es suficiente con seguir los siguientes cinco pasos que se indican a continuación:

1. Incluir en el documento fuente los comandos `\cite{Etiqueta}` donde se quiere hacer referencia a algún documento contenido en `MiBase.bib`. *Etiqueta* es el identificador utilizado para cada registro (registros diferentes no pueden tener la misma *Etiqueta*). Se puede hacer uso del comando `\nocite` con una o más etiquetas separadas por comas, para que se haga una entrada en el registro de la bibliografía sin hacer referencia a ella.
2. Incluir en el documento, en el lugar en que se desea que aparezca la bibliografía, las líneas `\bibliographystyle{Estilo}` y `\bibliography{NombreBase}`. El argumento *NombreBase* es el nombre de la base de datos para BIB_TE_X que se quiere utilizar (ha de tener extensión `bib`). Nótese que al usar el nombre de la base en el comando indicado anteriormente no ha de incluirse la extensión en el nombre. Si la base no se encuentra en el mismo directorio que el fichero fuente, ha de indicarse la ruta. Pueden utilizarse varias bases de datos, para lo cual basta con introducir los nombres de éstas separados por comas.

Estilo es uno de los estilos que BIB_TE_X utilizará para configurar la presentación de la lista de bibliografía. Los ficheros de estilo de la bibliografía tienen extensión `bst`, que no hay que incluir en el argumento, al igual que antes. Los estilos estándar son:

- **plain**: orden alfabético.

³Existe otra versión más actualizada desarrollada por Niel Kempson y Alejandro Aguiar-Sierra, conocida con el nombre de BIB_TE_X8

- **unsrt**: aparecen en el orden en que fueron citadas.
 - **alpha**: en vez de numerar las entradas, se las identifica con parte del nombre del autor y del año de publicación.
 - **abbrv**: igual que **plain** excepto que las entradas son más breves porque los nombres de autores, meses y revistas se abrevian.
3. Compilar el documento `MiDoc.tex`. Una vez utilizados los comandos anteriores la compilación de `MiDoc.tex` hace que en el fichero `MiDoc.aux` se guarde la información correspondiente a las citas realizadas en el documento, junto con el nombre del fichero de base de datos para `BIBTEX` que se quiere utilizar y con el nombre del estilo que se ha elegido para presentar la bibliografía.
 4. Generar la lista de bibliografía. Para procesar la información referente a citas bibliográficas guardada en el fichero `MiDox.aux` y producir el entorno `thebibliography` para `LATEX`, ha de ejecutarse el programa `BIBTEX` pasándole como argumento `MiDoc.tex`.
 5. Volver a compilar el documento `MiDoc.tex`. En esta compilación se utilizan los datos guardados en los ficheros generados por el anterior proceso, para escribir la lista de bibliografía y establecer las oportunas citas en el texto principal del documento.

El fichero de base de datos para `BIBTEX` es un fichero de texto que contiene **registros**, que a su vez contienen **campos**. Los registros empiezan por el carácter '@' seguido de una cadena que identifica el tipo (libro, artículo, etc.), luego entre llaves van los diferentes campos del registro separados por comas. Cada campo se compone de un nombre, un carácter '=' y, a continuación, su contenido. Dicho contenido puede ir entre comillas o llaves.

Los campos de un registro se dividen en tres clases:

- **Campos necesarios**. Si se omite alguno de los campos que tienen esta consideración para un tipo de registro, se produce un mensaje de advertencia al ejecutar `BIBTEX`. En general, eso es indicativo de que se está utilizando un tipo de registro inadecuado.
- **Campos optativos**. La información contenida en los campos de esta clase se utilizará si está presente, pero puede ser omitida sin causar problemas.
- **Campos ignorados**. De esta clase forman parte el resto de los campos de un registro. `BIBTEX` los ignora, pero pueden ser utilizados para poner información complementaria en el fichero `BIB` [34].

Actualmente existen programas que ofrecen una interfaz amigable para la creación de estas bases de datos bibliográficas. En nuestro caso se ha hecho uso de `BIBDesk` (fig. 2.4). `BIBDesk` es un editor gráfico y un administrador de referencias para `Mac OS X`. Es libre y de código abierto. Proporciona de una manera sencilla todas las opciones necesarias para la creación de una base de datos `BIBTEX`, así como herramientas de gestión (etiquetado mediante palabras clave, acceso a repositorios web externos con información `BIBTEX`, etc.).

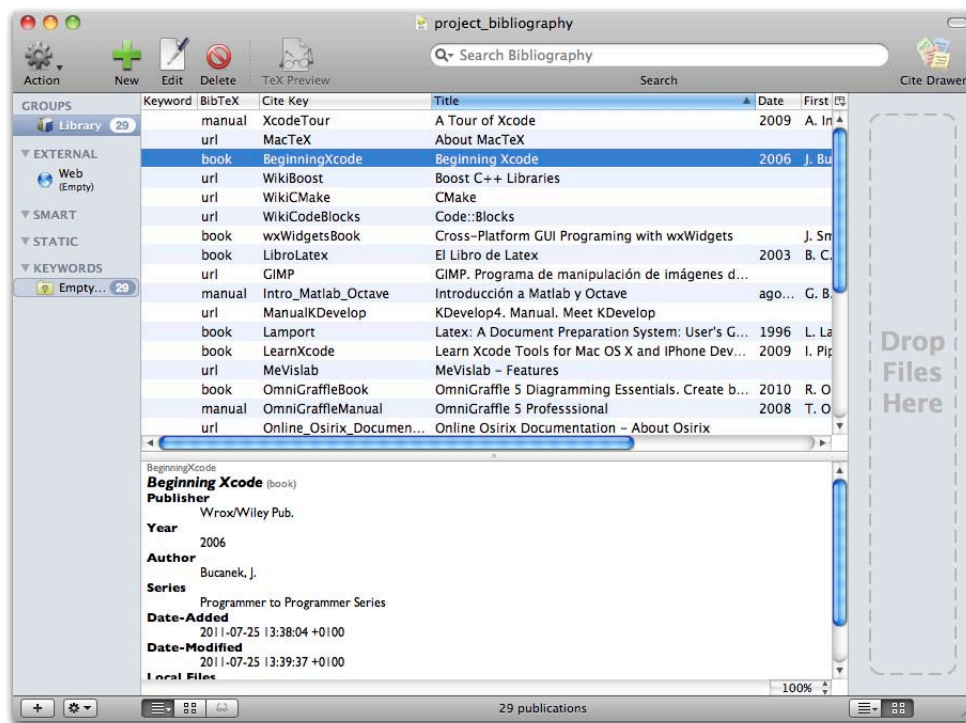


Figura 2.4: Interfaz del programa BIBDesk para la gestión de bases de datos bibliográficas BIB_TE_X

2.2.1.6. El paquete gloss

En ocasiones es útil poder tener una relación de los conceptos que se manejan a lo largo de un texto y quizás tener una descripción “relativamente extensa” de ellos, que, por otra parte, puede reutilizarse en nuevos documentos que se realicen sobre la misma temática. A esta relación de términos es a lo que se denomina **glosario**. L^AT_EX proporciona entornos para la generación de índices terminológicos, mediante el uso de MAKEINDEX y la adición de algún comando en el concepto que posteriormente se quiere incluir en la lista. No obstante, este mecanismo no nos permite poner una descripción, algo que sería un poco más interesante.

En este sentido, José Luis Díaz y Javier Brezos han desarrollado el paquete **gloss** con el objetivo de crear glosarios utilizando BIB_TE_X (descrita en el apartado anterior) en lugar de MAKEINDEX, dando mejores funcionalidades para abordar el problema descrito.

La herramienta **gloss** consta esencialmente de: un paquete para la comunicación con L^AT_EX, de nombre **gloss.sty** y un fichero de estilo para BIB_TE_X, de nombre **glsplain.bst**. El fichero de estilo ha sido desarrollado para gestionar bases de datos con un estilo específico para manejar un glosario:

```

@GD{
  Etiqueta
  word          = {Concepto},
  definition    = {DescripciónDelConcepto}
}

```

Es decir, únicamente hay un tipo de registro, @GD, con los siguientes campos obligatorios:

- **word:** donde el **Concepto** debe aparecer tal y como figura en el texto fuente. En el glosario que se genere, las entradas con la misma inicial aparecerán agrupadas, encabezadas por dicha letra y ordenadas alfabéticamente.
- **definition:** el argumento **DescripciónDelConcepto** puede ser tan largo como permita la longitud máxima soportada en los campos por BIB_{TEX}. Puede contener varios párrafos (realizados con `\par`) y cualquier construcción de L^AT_EX.

Los comandos básicos para la generación de un glosario, supuesto que se dispone de una base de datos BIB con el formato antes descrito, son:

```
\makegloss      \gloss[Opciones]{Etiqueta}      \printgloss{BaseDeDatos}
```

El comando `\makegloss` debe colocarse en el preámbulo, junto con el correspondiente paquete `gloss`. Es el encargado de crear el fichero auxiliar, `MiDoc.gls.aux`, en el que se irá anotando cada *Etiqueta* existente en el documento fuente que hayan sido seleccionadas para ese fin mediante el comando `\gloss`. El comando `\gloss` es análogo al comando `\cite`. Finalmente `\printgloss` es un comando similar a `\bibliography`: se ocupa de imprimir el glosario en el documento, utilizando para ello el fichero `MiDoc.gls.bbl` generado por BIB_{TEX} a partir de `MiDoc.gls.aux`, el estilo `glsplain.bst` y la base de datos declarada en el argumento *BaseDeDatos*.

Se puede observar que el flujo de funcionamiento es paralelo al que se emplea para construir la bibliografía de forma automática [34, 17]. Al igual que sucedía con el manejo de ésta, BIBDesk (fig. 2.4) proporciona soporte para manejar *registros* de tipo `gloss`.

2.2.1.7. Editor

A la hora de realizar la documentación se tuvieron en cuenta diferentes editores para tratar los documentos, o específicamente en nuestro caso, el código T_EX. Independientemente de la elección del editor, es indispensable instalar un compilador para dicho código. En el caso del sistema operativo Mac OS X se ha realizado la instalación de MacT_EX, cuya última versión del 4 de julio de 2011 es compatible con las versiones 10.5 y superiores del SO. MacT_EX es un producto desarrollado por el grupo T_EXnico de MacT_EX del grupo de usuarios de T_EX (TUG - T_EX Users Group). MacT_EX está compuesto de dos partes: MacT_EX-2011 y MacT_EXtras. MacT_EX-2011 es un paquete de instalación que instala todo lo necesario para poder ejecutar T_EX en Mac OS X. El paquete usa el instalador estándar de Apple. MacT_EXtras es una colección de extras opcionales: editores adicionales, correctores ortográficos, documentación y ejemplos [1].

Para la edición del código fuente del documento se seleccionó el editor Texmaker (fig. 2.5). Texmaker es un editor libre, moderno y multiplataforma de L^AT_EX, disponible para Linux, Mac

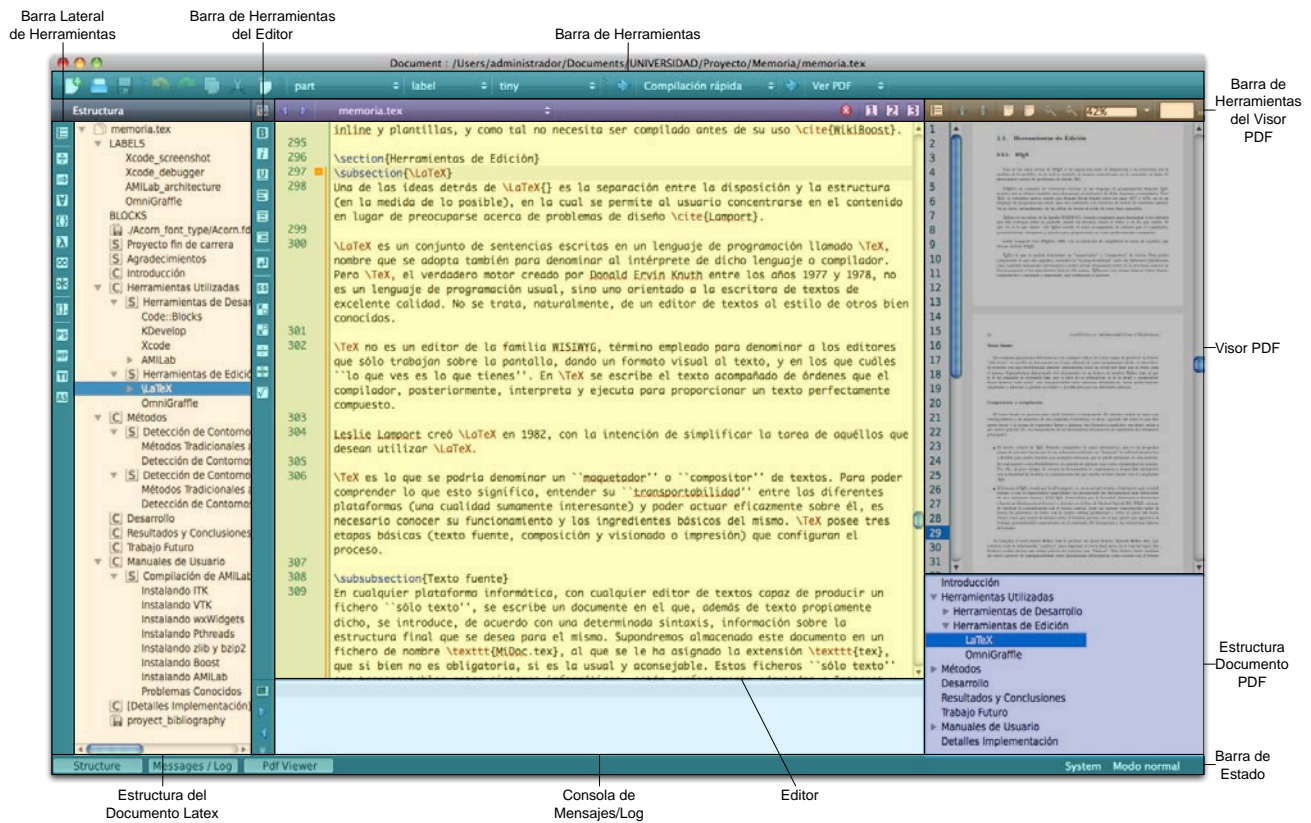


Figura 2.5: Captura de pantalla del software Texmaker

OS X y Windows. Integra todas las herramientas necesarias para desarrollar documentos con L^AT_EX en una sola aplicación.

Texmaker incluye soporte para unicode, corrector ortográfico, auto-completado, plegado de código y un visor pdf incorporado, con soporte para la sincronización con el texto y visualización en modo continuo. Es sencillo de usar y se distribuye bajo licencia GPL [11].

2.2.2. OmniGraffle

OmniGraffle es quizás uno de los programas más fáciles para hacer diagramas disponible para entornos Macintosh. De la misma forma que ocurre con otras herramientas de productividad, éste programa puede ser usado para más propósitos de los previstos. OmniGraffle puede utilizarse para escribir una carta o un informe de negocios, de la misma forma que con Microsoft Word se pueden crear diagramas.

Hay una buena razón de por qué se debe usar OmniGraffle para hacer diagramas (y consecuentemente usar un procesador de textos para escribir informes), y la razón más simple es que OmniGraffle está especializado en diagramas. OmniGraffle es excepcionalmente bueno cuando se trata de hacer gráficos con un aspecto profesional. Pero que los diagramas tengan un buen aspecto no es la única razón para usarlo, sino que además son fáciles de hacer, manipular y reutilizar.

Afortunadamente, existen herramientas de productividad como PagesTM de iWork y Microsoft WordTM que son excelentes para escribir informes, libros y otros textos; así que OmniGraffle no es necesario para hacer este tipo de tareas.

Existen algunas razones importantes de por qué debe intentarse evitar la creación de diagramas con Microsoft Word, o en menor grado, con OpenOffice Writer:

- Diseñar formas es engorroso, porque hay que hacer muchos clics.
- Se está limitado al tamaño de la página y es necesario planear cuidadosamente el diagrama, ya que cambiar algo puede causar que después haya que hacer mucho trabajo.
- No es posible conectar entre sí las formas, de forma que al remodelar el diagrama hay que estar moviendo cada cosa a mano.
- No se tienen preferencias automáticas de diseño, cada alineamiento, cada ajuste debe ser hecho a mano (y la medición se debe hacer a ojo).
- Se tiene un número limitado de formas a disposición del usuario.
- Es muy complicado ajustar características como la sombra, el grosor, el relleno, la geometría... a más de una figura a la vez.
- Las opciones de exportación de los diagramas son bastante limitadas [30].

En la figura 2.6 se ilustra de forma detallada la interfaz de este software. En él se puede encontrar:

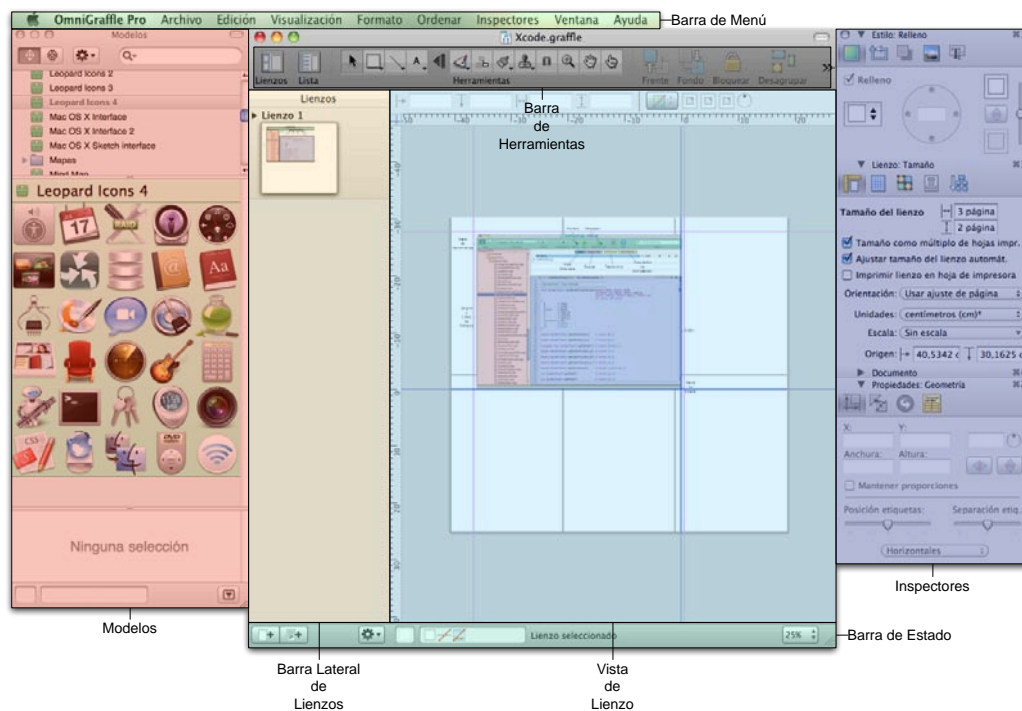


Figura 2.6: Detalle de la interfaz del programa OmniGraffle

- Barra de menú: al igual que en el resto de aplicaciones de Mac OS X, desde aquí se puede acceder a toda la variedad de comandos que se encuentren a disposición del usuario.
- Barra de herramientas: proporciona un acceso rápido a los controles. Se puede personalizar dicha barra desde el menú **Visualización**. Además, en la barra de herramientas se encuentra la paleta de herramientas que se usa para la creación de diagramas.
- Barra lateral de lienzos: muestra todos los lienzos que se encuentran en el documento, así como las capas que contienen. Se puede hacer clic sobre alguno de los lienzos y trabajar en él en la vista de lienzo.
- Modelos: contiene muchos objetos predefinidos para una amplia diversidad de propósitos. OmniGraffle viene por defecto con bastantes modelos, pero se pueden obtener más en [Graffletopia](#), que es un magnífico repositorio de terceros para modelos creados por usuarios de OmniGraffle (es preciso destacar que este servicio es completamente gratuito). En este repositorio además de obtener esos modelos, se pueden agregar a favoritos o dejar comentarios acerca de los mismos. Otro plus a recalcar es que se ocupan de indicar con qué software de la compañía Omni Group es compatible cada modelo. La búsqueda puede realizarse a través de un buscador, por categorías o incluso por etiquetas (*tags*).
- Vista de lienzo: es en la vista que actualmente se está dibujando. Se pueden usar las herramientas de dibujo para crear formas y conectarlas entre ellas usando líneas. Se pueden arrastrar objetos alrededor, agruparlos, hacer tablas... Para usar un modelo predefinido simplemente hay que arrastrarlo desde los **Modelos** hasta el lienzo.
- Inspectores: contienen todos los controles necesarios para modificar los objetos seleccionados en el lienzo, el propio lienzo y todo el documento. Existen un total de dieciséis

inspectores, que se encuentran organizados basándose en los tipos de cosas a las que afectan: estilo, propiedades, lienzo y documento [19].

Capítulo 3

Métodos

Nuna vez conocidas cuáles son las herramientas que se van a utilizar durante el desarrollo del proyecto, es el momento de mostrar el estudio de los métodos. Este capítulo estará dividido en dos partes. Por un lado se tratará el caso 2D y por otro el 3D. Dentro de cada uno de ellos se comenzará viendo los detectores clásicos de la literatura (a nivel píxel y vóxel), para posteriormente introducir los de alta precisión (sub-píxel y sub-vóxel).

En relación a los métodos sub-píxel, se hará un recorrido que lleva desde el método básico de detección para contornos de segundo orden hasta llegar al método de restauración, capaz de detectar contornos muy cercanos dentro de una imagen afectada por ruido.

En la parte dedicada a la precisión sub-vóxel, se explicará el método básico para la detección de contornos tridimensionales de segundo orden, como extensión de lo relatado en 2D.

3.1. Detección de Bordos en Imágenes Bidimensionales

3.1.1. Métodos Tradicionales a Nivel Píxel

La primera de las tareas a realizar es recabar y estudiar los detectores de contornos básicos, a fin de conseguir un cierto adiestramiento de cara al posterior análisis del método de detección con precisión sub-píxel. En esta sección serán descritos dichos detectores, su fundamento, así como sus características fundamentales.

No obstante, previamente a ello, sería preciso describir a qué es lo que se considera como borde o contorno. Los bordes de una imagen se pueden definir como transiciones entre dos regiones de niveles de gris significativamente distintos. Éstos suministran una valiosa información sobre las fronteras de los objetos que puede ser utilizada para la segmentación de la imagen, reconocimiento de objetos, visión estéreo, etc.

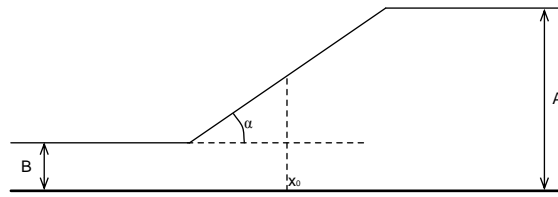


Figura 3.1: Modelo unidimensional y continuo de un borde ideal [24]

En la figura 3.1 se muestra un modelo unidimensional y continuo de un borde. Este modelo representa una rampa desde un nivel de gris bajo “B” a uno alto “A”, con los siguientes parámetros:

- Variación de intensidad $H = A - B$
- Ángulo de inclinación de la rampa “ α ”
- Coordenada horizontal “ x_0 ”

Un operador que proporciona los valores de “ x_0 ” y “H” daría unos datos valiosísimos sobre la imagen, ya que proporcionaría la amplitud del borde, y lo localizaría con exactitud dentro de la imagen.

En las imágenes reales los bordes nunca se ajustan totalmente al modelo anterior (fig. 3.2). Las causas para ello son diversas, destacándose las siguientes:

- Las imágenes son discretas.
- Están afectadas por ruido diverso.
- El origen de los bordes puede ser muy diverso: bordes de oclusión, superficies de diferente orientación, distintas propiedades reflectantes, distinta textura, efectos de la iluminación (sombras y/o reflejos), etc.

Las circunstancias anteriores introducen una gran complejidad a la hora de detectar los bordes de una imagen. En particular, en este proceso se consideran tres tipos de errores:

1. **Error en la detección:** un operador es un buen detector si la probabilidad de detectar el borde es alta cuando éste realmente existe en la imagen, y baja cuando éste no existe.
2. **Error en la localización:** un operador localiza bien un borde cuando la posición que proporciona coincide con la posición real del borde en la imagen. Ambos errores están estrechamente ligados a los problemas anteriormente señalados, y muy especialmente a la presencia de ruido en la imagen. En la práctica, la calidad en la detección y localización están en conflicto.
3. **Respuesta múltiple:** varios píxeles son detectados en un único borde.

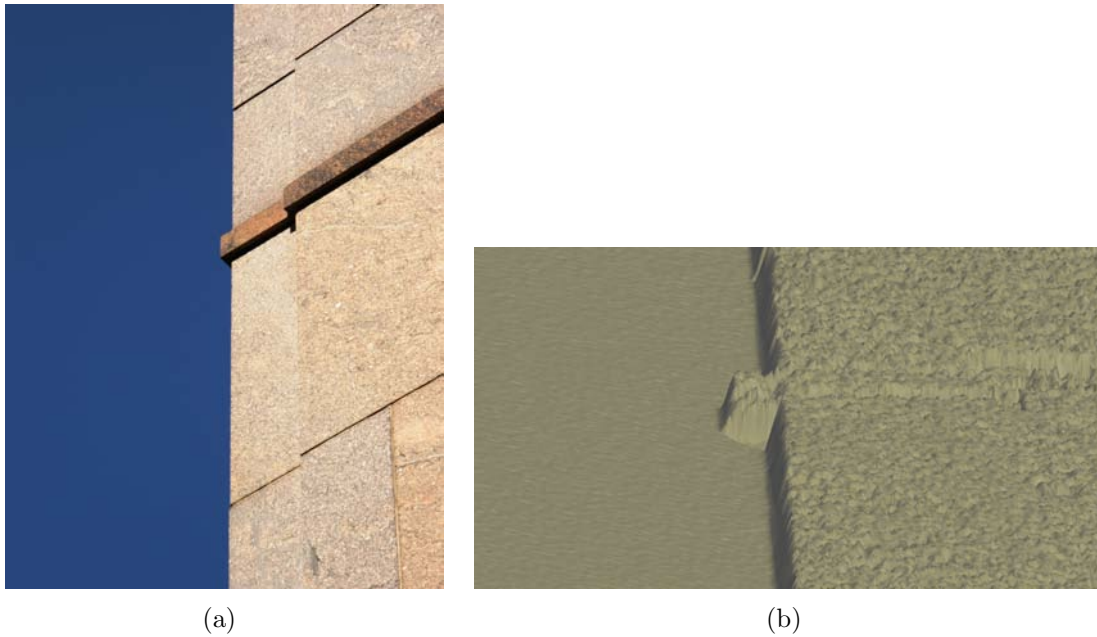


Figura 3.2: Representación de un borde en una imagen real: (a) Imagen 2D (b) Imagen del borde visto como una superficie

La mayoría de las técnicas para detectar bordes emplean operadores locales basados en distintas aproximaciones discretas de la primera y segunda derivada de los niveles de grises de la imagen, si bien existen otras posibilidades para ello, como el empleo de patrones de bordes ideales. A continuación se describirán algunas de las técnicas.

3.1.1.1. Detectores Basados en la Primera Derivada (Gradiente)

La derivada de una señal continua proporciona las variaciones locales con respecto a la variable, de tal forma que el valor de la derivada es mayor cuanto más rápidas son estas variaciones.

En el caso de funciones bidimensionales $f(x, y)$, la derivada es un vector que apunta en la dirección de máxima variación de $f(x, y)$ y cuyo módulo es proporcional a dicha variación. Este vector, denotado como $\nabla f(x, y)$, se denomina gradiente y se define como:

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial}{\partial x} f(x, y) \\ \frac{\partial}{\partial y} f(x, y) \end{bmatrix} = \begin{bmatrix} f_x(x, y) \\ f_y(x, y) \end{bmatrix} \quad (3.1)$$

Obsérvese que, de acuerdo con la expresión 3.1, el gradiente en un punto (x, y) viene dado por las derivadas de $f(x, y)$ a lo largo de los ejes coordenados ortogonales “x” e “y”. El módulo y la dirección del gradiente vienen dados por:

$$|\nabla f(xy)| = \sqrt{(f_x(x, y))^2 + (f_y(x, y))^2} \quad (3.2)$$

$$\alpha(x, y) = \arctan\left(\frac{f_y(x, y)}{f_x(x, y)}\right) \quad (3.3)$$

3.1.1.2. Aproximaciones Discretas del Operador Gradiente

En el caso bidimensional discreto, las distintas aproximaciones del operador gradiente se basan en diferencias entre los niveles de grises de la imagen. Por ejemplo, la derivada parcial $f_x(x, y)$ (gradiente de fila) puede aproximarse por la diferencia de píxeles adyacentes de la misma fila, es decir:

$$f_x(x, y) \approx G_F(i, j) = [F(i, j) - F(i, j - 1)] / T \quad (3.4)$$

o también por la diferencia de píxeles separados:

$$f_x(x, y) \approx G_F(i, j) = [F(i, j + 1) - F(i, j - 1)] / 2T \quad (3.5)$$

donde se ha supuesto, sin pérdida de generalidad, que el índice de filas “ i ” crece de arriba a abajo, y el de columnas “ j ” de izquierda a derecha. Puesto que las derivadas serán comparadas más tarde con un umbral ajustable, los factores de escala $1/T$ y $1/2T$ de 3.4 y 3.5, pueden ser omitidos.

El gradiente de fila G_F y de columna G_C en cada punto se obtienen mediante la convolución de la imagen con sendas máscaras H_F y H_C , esto es:

$$G_F(i, j) = F(i, j) \otimes H_F(i, j) \quad (3.6)$$

$$G_C(i, j) = F(i, j) \otimes H_C(i, j) \quad (3.7)$$

donde $H_F(i, j)$ y $H_C(i, j)$ son las respuestas impulsionales del gradiente de fila y de columna respectivamente.

La magnitud y orientación del vector gradiente se obtienen de acuerdo con las ecuaciones 3.2 y 3.3, respectivamente; si bien, dado el elevado coste computacional que conlleva el cálculo de la magnitud, ésta suele aproximarse por la expresión:

$$|G(i, j)| = \sqrt{(G_F)^2 + (G_C)^2} = |G_F(i, j)| + |G_C(i, j)| \quad (3.8)$$

3.1.1.3. Operadores de Roberts, Prewitt, Sobel y Frei-Chen

La idea del operador de **Roberts** (Robert, 1965) es obtener una buena respuesta ante bordes diagonales a partir de la diferencia entre pares diagonales de píxeles. Nótese que, en este caso, las máscaras no representan las derivadas a lo largo de filas y columnas, sino en dos direcciones diagonales (45°) perpendiculares entre sí. Aunque esta circunstancia no afecta a la expresión para determinar el módulo del gradiente 3.8, sí afecta a la orientación, que en este caso viene dada por la expresión:

$$\alpha(i, j) = \frac{\pi}{4} + \arctan\left(\frac{G_2(i, j)}{G_1(i, j)}\right) \quad (3.9)$$

donde

$$G_1(i, j) = F(i, j) - F(i + 1, j + 1) \quad (3.10)$$

$$G_2(i, j) = F(i + 1, j) - F(i, j + 1) \quad (3.11)$$

El operador de Roberts ofrece unas buenas prestaciones en cuanto a localización. El gran inconveniente es su extremada sensibilidad al ruido, y por tanto, con unas pobres cualidades de detección. Los operadores de Prewitt, Sobel y Frei-Chen intentan aliviar esta deficiencia involucrando en la convolución un mayor número de píxeles del entorno de vecindad. Estos tres detectores pueden formularse de forma conjunta mediante las máscaras de convolución mostradas en 3.12. Obsérvese que los gradientes de fila y columna están normalizados para proporcionar una ganancia unidad sobre la zona de pesos positivos y negativos.

$$H_F(i, j) = \frac{1}{2 + K} \begin{pmatrix} 1 & 0 & -1 \\ K & 0 & -K \\ 1 & 0 & -1 \end{pmatrix}, \quad H_V(i, j) = \frac{1}{2 + K} \begin{pmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & K & 1 \end{pmatrix} \quad (3.12)$$

El operador de **Prewitt** ($K = 1$) es, en cierta forma, similar al de diferencia de píxeles separados introducido anteriormente, donde se involucran a los vecinos de filas/columnas adyacentes para proporcionar mayor inmunidad al ruido.

El operador de **Sobel** se distingue del operador de Prewitt en que se duplican los valores de los píxeles al norte, sur, este y oeste, es decir, $K = 2$. De esta forma, cada píxel del entorno de vecindad es ponderado de acuerdo con la distancia de éste al central (píxel evaluado). Como consecuencia de ello, el Sobel es más sensible que el Prewitt a los bordes diagonales, mientras que éste lo es para los horizontales y verticales. En la práctica, sin embargo, no se aprecia gran diferencia entre ellos.

Por otra parte, **Frei** y **Chen** propusieron un valor de $K = \sqrt{2}$ con el fin de que el gradiente sea el mismo para bordes horizontales, verticales y diagonales.

Estos operadores 3×3 tienen unas prestaciones bastante similares. Particularmente, los dos primeros (Prewitt y Sobel) son muy populares, debido a que los valores de los coeficientes hacen que las operaciones aritméticas se puedan programar de una manera eficiente o implementar fácilmente en hardware [24].

3.1.1.4. Operadores Basados en la Segunda Derivada (Laplaciana)

Siempre que en la imagen se presenta un cambio de intensidades a lo largo de una determinada dirección, existirá un máximo en la primera derivada a lo largo de dicha dirección y, consecuentemente, un paso por cero en la segunda derivada. La dirección de interés será aquella ortogonal a la orientación local de los cruces por cero.

Puesto que, generalmente esta dirección no es conocida, parece claro que la utilización de un operador independiente de la orientación como la Laplaciana presenta evidentes ventajas computacionales. Asimismo, al ser una derivada segunda, tiene la ventaja de facilitar la localización precisa del borde.

En el dominio continuo la Laplaciana del borde de una función bidimensional $f(x, y)$ se define como:

$$\nabla^2 f(x, y) = \frac{\partial^2}{\partial x^2} f(x, y) + \frac{\partial^2}{\partial y^2} f(x, y) \quad (3.13)$$

La Laplaciana vale cero si $f(x, y)$ es constante o cambia linealmente su amplitud. El cambio de signo de la función resultante nos indica que en ese lugar existe un cruce por cero (cambio de positivo a negativo o viceversa) y, por tanto, indica la presencia de un borde. Nótese que, a diferencia del gradiente, la Laplaciana no es un vector.

Las características principales de la Laplaciana son:

- Al ser un operador de segunda derivada, exhibe una pobre detección de bordes ya que es muy sensible al ruido. Para evitarlo, en lugar de aplicar la Laplaciana directamente, se realiza primero un suavizado (usando operadores como **LoG** - Laplaciana de la Gaussiana).
- Si los bordes de la imagen original $F(i, j)$ verifican ciertas condiciones de suavidad, los cruces por cero de la imagen Laplaciana resultante son curvas cerradas (Torre y Poggio, 1986).
- Proporciona un borde de espesor unidad (un píxel) [24].

3.1.1.5. Operador de Canny

A diferencia de los anteriores operadores, el operador de Canny (1986) está basado en un desarrollo analítico de optimización, partiendo de un modelo continuo unidimensional de un escalón.

Considérese una función escalón con amplitud “ h_E ” afectada por un ruido blanco gaussiano de desviación típica σ_n . Supóngase que la detección de bordes se va a llevar a cabo mediante la convolución de una función continua unidimensional $f(x)$ con una función respuesta impulsional

antisimétrica $h(x)$, que tiene amplitud cero fuera del intervalo $[-W, W]$. El borde buscado de la función $f(x)$ se marcará donde aparezca el máximo local del gradiente, obtenido tras la convolución de $f(x)$ con $h(x)$. Para determinar la función $h(x)$ buscada se exige que ésta satisfaga los siguientes criterios:

- **Buena Detección.** Se maximiza la amplitud de la relación señal-ruido (snr) del gradiente para obtener una alta probabilidad de marcarlo donde no lo hay. La relación señal-ruido para el modelo considerado es:

$$snr = \frac{h_E}{\sigma_n} S(h) \quad (3.14)$$

con

$$S(h) = \frac{\int_{-W}^0 h(x) dx}{\int_{-W}^W [h(x)]^2 dx} \quad (3.15)$$

- **Buena Localización.** Los puntos del borde marcados por el operador deben estar tan cerca del centro del borde como sea posible. El factor de localización se define como:

$$LOC = \frac{h_E}{\sigma_n} L(h) \quad (3.16)$$

con

$$L(h) = \frac{h'(0)}{\int_{-W}^W [h(x)]^2 dx} \quad (3.17)$$

donde $h'(0)$ es la derivada de $h(x)$.

De acuerdo con estos dos criterios, el detector óptimo de bordes de tipo escalón es un escalón truncado (diferencia de escalones). Este operador, sin embargo, tiende a generar muchos máximos locales en su respuesta a bordes ruidosos de tipo escalón. Aunque estos bordes se deberían considerar erróneos de acuerdo con el primer criterio, sin embargo, no se ha considerado la interacción entre las respuestas en varios puntos próximos. Si se examina la salida del operador de diferencia de escalones, se ve que la respuesta a un escalón con ruido es un pico triangular con varios máximos en la vecindad del borde. Por ello es necesario incorporar el siguiente criterio que corrija este problema.

- **Respuesta Única.** Debe haber una única respuesta para cada borde. La distancia entre picos del gradiente cuando sólo el ruido está presente, denotada por x_m , se establece como una fracción de k del ancho del operador, es decir:

$$x_m = kW \quad (3.18)$$

Canny combina los tres criterios minimizando el producto $S(h)L(h)$ sujeto a la restricción dada por 3.18. Debido a la complejidad de esta formulación no existe solución analítica de este problema.

Todo el desarrollo expuesto hasta ahora se refiere a una señal unidimensional continua. En el caso de imágenes digitales (bidimensionales y discreto), el operador propuesto por Canny se aproxima mediante la derivada de la Gaussiana en la dirección perpendicular al borde. En la práctica, aunque existen diferentes implementaciones, los pasos a seguir serían:

1. Calcular el módulo y la dirección del gradiente de la imagen suavizada aplicando el operador DroG (Derivada de la Gaussiana).
2. En la dirección del gradiente, eliminar puntos que no sean máximos locales del módulo (equivalente a encontrar el paso por cero en el operador LoG). Desechando los píxeles que no son máximos locales se mejora la localización y se evitan detecciones falsas.

El proceso de eliminación de no-máximos locales se suele implementar siguiendo el borde en la dirección perpendicular al gradiente, considerando los 8-vecinos.

Normalmente se utiliza una función de histéresis tal que el primer píxel de un segmento del borde debe tener un módulo del gradiente que supere un valor *gradiente_alto*. Entonces se comienza a añadir píxeles vecinos en la dirección del borde (perpendicular al gradiente) mientras que el valor del módulo de éste no caiga por debajo de un valor *gradiente_bajo* (menor que *gradiente_alto*) [24].

Aquí termina un breve repaso por los detectores de bordes tradicionales a nivel píxel. En secciones posteriores se describirá cómo se ha usado esta parte como adiestramiento para realizar un ejemplo introductorio en el software AMILab. A continuación se pasan a ver los métodos con precisión sub-píxel para imágenes bidimensionales.

3.1.2. Detección de Contornos con Precisión Sub-píxel

Antes de comenzar a describir los métodos, sería interesante decir cuál es la hipótesis inicial. Es preciso indicar que, a pesar de que el término PVE (Partial Volume Effect - Efecto de Volumen Parcial) está más asociado a imágenes 3D, se va a introducir aquí, ya que en dos dimensiones sucede algo similar a este efecto.

Como se ha descrito con anterioridad, un borde se puede definir como una línea virtual entre dos regiones con diferentes niveles de intensidad. Normalmente, los métodos estándar sólo indican cuál es el píxel borde. Para dicho fin se basan en información numérica de una cierta vecindad, y así se decide si se considera borde o no. Además, se asume para ello que la imagen es una función continua y derivable en todo punto. Es decir, aunque la imagen que se use sea una matriz de valores, se estima que es el resultado del muestreo de una determinada señal continua.

El método propuesto por el profesor Agustín Trujillo Pino en [42] plantea algo diferente. En lugar de considerarse que se está ante una función continua y derivable, se plantea que un borde es una discontinuidad en los valores de intensidad de la función, delimitando la frontera entre dos objetos. Cuando un píxel forma parte de un borde tiene una intensidad intermedia, que se encuentra entre la intensidad del objeto y la intensidad del fondo (ver fig. 3.3). Por tanto, se afirma que la intensidad de un píxel borde es la suma de la contribución de cada color, pesada por el área relativa que ocupa cada uno en el interior de dicho píxel. Es este efecto parcial en el interior del píxel (vóxel en tres dimensiones) el que permitirá al método localizar de forma precisa el contorno, así como todas sus características: posición sub-píxel, orientación, curvatura e intensidad a ambos lados. En las siguientes secciones se analizarán cada uno de los

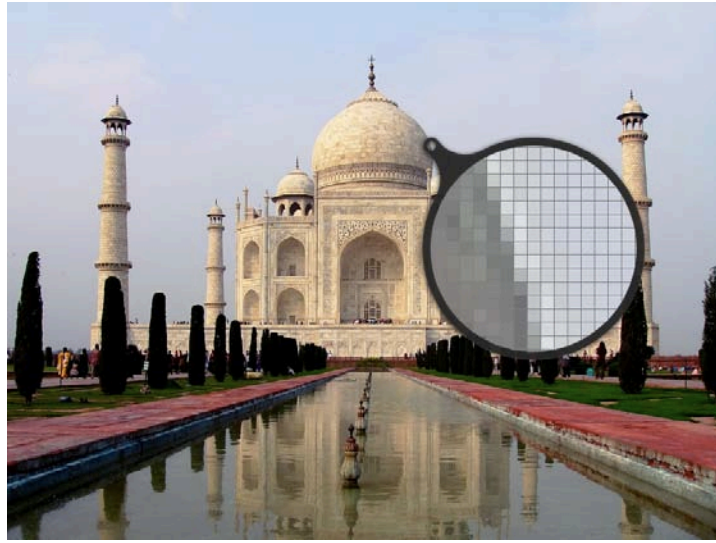


Figura 3.3: Detalle del efecto parcial producido en un borde

métodos que posteriormente serán incorporados a AMILab, pero ha de tenerse en cuenta que se describe el caso en el que el valor de la derivada en y es mayor que en x ($|G_y(i, j)| > G_x(i, j)$), es decir, caso en el que se tiene un borde horizontal. Es por ello que se verá como se detalla el procedimiento donde el método se aplica por columnas, aunque es simple extrapolarlo al caso contrario.

3.1.2.1. Método Simplificado de Primer Orden

Considérese que $F_{i,j}$ es una imagen compuesta por un único contorno recto que separa dos zonas de intensidades A y B. Dicho contorno recto atraviesa el píxel central de la imagen y posee una pendiente entre 0 y 45° . A esto se le denomina imagen ideal con un contorno de primer orden.

En un caso real, el contorno no va a ser exactamente recto, con lo cual suponer que lo es no es más que una aproximación que se asume en la vecindad del píxel.

Formalmente, lo que se hace puede definirse de la siguiente manera: se sitúa el origen de coordenadas en el centro geométrico del píxel central, y considérese los ejes x e y como se ven en la figura 3.4. Nótese que esta vez se ha tomado la dirección de las y positivas hacia arriba, simplemente para seguir la forma habitual de representar funciones en el plano.

Sea $y = r(x)$ la curva que representa el contorno, cuya expresión exacta es desconocida. Lo que si se sabe es que la pendiente de la curva en $x = 0$ está entre 0 y 1 . Una aproximación lineal a la curva alrededor del punto $x = 0$ puede obtenerse como $y = a + bx$, al estilo de cuando se hace una aproximación a través del polinomio de Taylor de grado 1 de una función

$$P_1(x) = r(0) + r'(0)x = a + bx$$

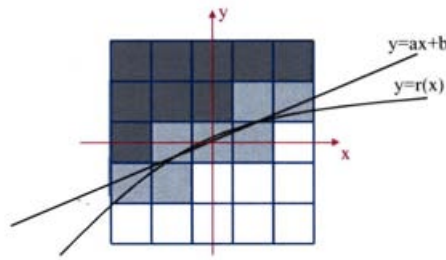


Figura 3.4: Ejes de coordenadas en la zona central del píxel [42]

Realmente no se hace como en Taylor, pues éste obtiene como aproximación la recta cuya posición y primera derivada coincide con la curva del contorno. En nuestro caso, la recta que se busca es la que mejor se ajusta a los valores de intensidad de los píxeles adyacentes, y no es necesariamente la misma que la de Taylor. Por lo tanto, lo que el método trata de hacer puede interpretarse como **encontrar los valores de a y b que mejor se ajustan a las intensidades de los píxeles** en un cierto contorno centrado en el píxel central. Una vez que se hayan localizado los coeficientes de la recta, la posición de ésta puede interpretarse como la posición del contorno dentro del píxel, y la pendiente de la recta considerarse como la derivada de la curva del contorno sobre la vertical central del píxel. Por lo tanto, el vector normal a dicha recta será una buena aproximación del vector normal al contorno en dicho punto.

Para la realización del cálculo se tomará una única ventana, y se calculan con los valores de los píxeles interiores los coeficientes a y b, a partir de los cuales se podrá estimar la orientación y localización exacta de la recta que se está buscando. Tómese por ejemplo un entorno de 5×3 , centrado en el píxel central. Es preciso recordar que se está en el caso para pendientes entre 0 y 1, y de ahí que la ventana sea más larga en la dirección y que en la x . Para contornos con pendiente mayor que 1, la ventana sería lógicamente de 3×5 .

Por otro lado, también se va a trabajar directamente con los valores de la imagen de entrada $F_{i,j}$, sin tener que usar los valores de las imágenes de parciales más que para detectar los máximos locales y el octante en el que se encuentra. De esta forma, en las expresiones que se usen para detectar los parámetros del contorno sólo aparecerán los valores de F y no los de F_x y F_y .

Cálculo de los coeficientes de la recta

Tómese por tanto dicho entorno, centrado en el píxel central (i, j) . Dentro de esta ventana, se asume que el contorno se comporta como una línea recta $y = a + bx$. Eso significa que, según muestra la hipótesis inicial, la intensidad de cada píxel viene dada por la siguiente expresión:

$$F(i, j) = \frac{P_{i,j}}{h^2} A + \frac{h^2 - P_{i,j}}{h^2} B = B + \frac{A - B}{h^2} P_{i,j} \quad (3.19)$$

donde en este caso $P_{i,j}$ es el área interior al píxel (i, j) que se encuentra bajo la recta. Por tanto, $0 \leq P_{i,j} \leq h^2$.

Tómese cualquiera de las tres columnas que forman la ventana, y súmese los valores de los 5 píxeles que comprenden. En ese caso, los valores obtenidos deberían corresponder a las siguientes expresiones:

$$\begin{aligned}
 S_L &= \sum_{n=j-2}^{j+2} F_{i-1,n} = 5B + \frac{A-B}{h^2} \sum_{n=j-2}^{j+2} P_{i-1,n} = 5B + \frac{A-B}{h^2} L \\
 S_M &= \sum_{n=j-2}^{j+2} F_{i,n} = 5B + \frac{A-B}{h^2} \sum_{n=j-2}^{j+2} P_{i,n} = 5B + \frac{A-B}{h^2} M \\
 S_R &= \sum_{n=j-2}^{j+2} F_{i+1,n} = 5B + \frac{A-B}{h^2} \sum_{n=j-2}^{j+2} P_{i+1,n} = 5B + \frac{A-B}{h^2} R
 \end{aligned} \tag{3.20}$$

donde L , M y R indican el área interior a cada columna que se encuentra bajo la recta, tal y como se ve en 3.5.

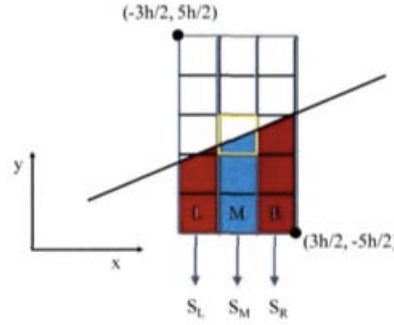


Figura 3.5: Área interior de cada columna (L, M, R) [42]

Tratándose de una línea recta que atraviesa el píxel central, con pendiente entre 0 y 1, se deduce que dicha recta siempre atravesará los límites de la ventana de izquierda a derecha, y nunca cortará el borde superior e inferior de la ventana. Esto significa que dichas áreas pueden ser calculadas analíticamente de forma sencilla, mediante una integral iterada, de la siguiente forma:

$$\begin{aligned}
 L &= \int_{-3h/2}^{-h/2} (a + bx + 5h/2) dx = ah - bh^2 + \frac{5}{2}h^2 \\
 M &= \int_{-h/2}^{h/2} (a + bx + 5h/2) dx = ah + \frac{5}{2}h^2 \\
 R &= \int_{h/2}^{3h/2} (a + bx + 5h/2) dx = ah + bh^2 + \frac{5}{2}h^2
 \end{aligned}$$

Se ve que la segunda ecuación correspondiente a M sólo depende de a y no de b , con lo cual se puede usar para hallar una expresión para a , de la manera siguiente:

$$S_M = 5B + \frac{A-B}{h^2} M = 5B + \frac{A-B}{h^2} \left(ah + \frac{5}{2}h^2 \right) \Rightarrow$$

$$\Rightarrow a = \frac{2S_M - 5(A + B)}{2(A - B)}h$$

Una vez conocido a , podría hallarse b utilizando cualquiera de las columnas izquierda o derecha, pero para que el esquema sea más simétrico y poder así usar información de ambas columnas simultáneamente, se usará la resta de ambas expresiones, ya que el término donde se encuentra b tiene el signo cambiado. De esta manera:

$$\begin{aligned} S_R - S_L &= \frac{A - B}{h^2}(R - L) = \frac{A - B}{h^2}(2bh^2) \Rightarrow \\ \Rightarrow b &= \frac{S_R - S_L}{2(A - B)} \end{aligned}$$

Como los valores de S_L , S_M y S_R son conocidos, ya que se calculan a partir de los valores de los píxeles, faltaría conocer los valores de A , B y h . Para h , su valor exacto es irrelevante, con lo cual lo usual es darle el valor $h = 1$. Para A y B , y ciñéndose al interior de la ventana que se está utilizando, se puede encontrar estimaciones fácilmente usando los píxeles que se sabe con seguridad que no tocan la recta. De esta forma, con los tres píxeles de la esquina superior izquierda se estimaría el valor de A , y con los tres de la esquina inferior derecha el valor de B . De esa forma quedaría:

$$\begin{aligned} A &= \frac{F_{i,j+2} + F_{i+1,j+2} + F_{i+1,j+1}}{3} \\ B &= \frac{F_{i-1,j-1} + F_{i-1,j-2} + F_{i,j-2}}{3} \end{aligned} \quad (3.21)$$

Con esto se concluye que, usando una ventana de 5×3 centrada en el píxel, se puede obtener de forma sencilla la expresión para la recta que mejor aproxima la curva del contorno. Ahora se usará la expresión de esta recta para obtener los datos que se querían, que son: posición de la recta en $x = 0$ y vector normal a la recta. El salto de intensidad a ambos lados del contorno se calcula directo, ya que es $A - B$.

Con todo lo anterior, se pueden obtener los parámetros de la recta, sabiendo que el corte con la vertical central del píxel está en $(0, a)$ y que el vector normal se calcula como [42]:

$$N = \frac{A - B}{\sqrt{1 + b^2}} [b, -1] \quad (3.22)$$

3.1.2.2. Método Simplificado de Segundo Orden

Para desarrollar un método que permita estimar la curvatura del contorno en un píxel a partir del método anterior, simplemente habrá que cambiar de una aproximación lineal a una cuadrática. Esto es lógico ya que, si realmente se está reconociendo que la línea del contorno no es recta sino que puede tener una ligera curvatura, entonces no tiene sentido aproximar dicha línea por una recta. En su lugar habrá que aproximar por una parábola. Es decir, volviendo al

polinomio de Taylor, pero esta vez de grado 2, de una función $r(x)$ alrededor del punto $x = 0$, éste tiene la expresión:

$$P_2(x) = r(0) + r'(0)x + \frac{r''(0)}{2}x^2 = a + bx + cx^2 \quad (3.23)$$

Por lo tanto, lo que se hará en esta ocasión será tratar de encontrar los coeficientes de la parábola que mejor se ajustan a los valores de intensidad de los píxeles cercanos al píxel en cuestión, y obtener los valores de posición, pendiente y curvatura de dicha parábola, los cuales serán las estimaciones que se darán para los parámetros del contorno.

Se procederá de la misma manera que en el apartado anterior. La situación será la que se muestra en la figura 3.6. Se va a tratar de encontrar la parábola que pasa por el píxel central. Se usará una ventana de iguales dimensiones que la anterior, 5×3 . Sin embargo, en este caso aparece una diferencia con respecto al método lineal. Ahora no se puede garantizar que la parábola cruce la ventana de izquierda a derecha. Por ahora se asume que la parábola no corta los límites inferior y superior de la ventana.

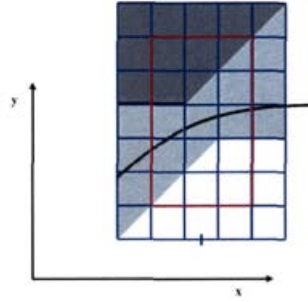


Figura 3.6: Situación para el cálculo del método de segundo grado [42]

Cálculo de los coeficientes de la parábola Si se realizan las sumas de los valores de los píxeles en las tres columnas, se encuentran idénticas expresiones que en el caso lineal. La diferencia radica ahora en las expresiones para las áreas L , M y R . Dichas expresiones quedarían como:

$$\begin{aligned} L &= \int_{-3h/2}^{-h/2} (a + bx + cx^2 + 5h/2) dx = ah - bh^2 + \frac{13}{12}ch^3 + \frac{5}{2}h^2 \\ M &= \int_{-h/2}^{h/2} (a + bx + cx^2 + 5h/2) dx = ah + \frac{1}{12}ch^3 + \frac{5}{2}h^2 \\ R &= \int_{h/2}^{3h/2} (a + bx + cx^2 + 5h/2) dx = ah + bh^2 + \frac{13}{12}ch^3 + \frac{5}{2}h^2 \end{aligned} \quad (3.24)$$

En esta ocasión, se tiene un sistema lineal de tres ecuaciones y tres incógnitas, (a, b, c) , fácil de resolver, cuya solución es la siguiente:

$$a = \frac{26S_M - S_L - S_R}{24(A - B)}h + \frac{(5 - 125h)A - (5 + 115h)B}{48(A - B)}$$

$$b = \frac{S_R - S_L}{2(A - B)} + \frac{5(h - 1)}{4h}$$

$$c = \frac{S_L + S_R - 2S_M}{2h(A - B)} + \frac{5(h - 1)}{4h^2}$$

Los valores de A y B se estiman igual que en el caso lineal y a h se le asigna valor 1, quedando expresiones más simples.

Con esto se concluye que, de forma similar al caso anterior, usando una ventana de 5×3 centrada en el píxel, se puede obtener de forma sencilla la expresión para la parábola que mejor aproxima la curva del contorno (aunque hasta el momento se ha fijado que la parábola cruza la ventana de izquierda a derecha). Ahora se usará la expresión de esta parábola para obtener los datos que se quería, que son: posición de la curva en $x = 0$, vector normal a la curva en ese punto, y su curvatura. El salto de intensidad a ambos lados del contorno se calcula de forma directa, ya que es $A - B$.

En cuanto al cálculo de la normal al contorno, la expresión es la misma que en 3.22. Pero, debido a que se está en el caso de segundo orden, hay que añadir además la expresión para el cálculo de la curvatura [42]:

$$K = \frac{2c}{(1 + b^2)^{\frac{3}{2}}} \tag{3.25}$$

3.1.2.3. Contornos en Imágenes Suavizadas

Las imágenes obtenidas por dispositivos de captación nunca son totalmente ideales, puesto que de un modo u otro se ven afectadas por algún tipo de ruido. Este fenómeno puede afectar a una correcta detección de los contornos. Es necesario procesar la imagen previamente para tratar de eliminarlo en la medida de lo posible. Por ello, en esta sección se describirá una nueva variante del método para este caso en el que la imagen de entrada se vea afectada por ruido. El objetivo es que el método sea capaz de detectar correctamente un borde en una imagen que previamente ha sido suavizada.

Hasta el momento, si se tiene una imagen ideal a la que se le aplica el método, el resultado que se obtiene es el esperado: una correcta detección de los parámetros del contorno. Sin embargo, si se trata de aplicarlo a una imagen con ruido, el resultado será bastante más caótico, producto de los cambios de intensidad introducidos por el ruido en la imagen. Lo más lógico sería realizar un proceso de suavizado previo para poder posteriormente llevar a cabo la detección. Pero, a pesar de ello, la detección tampoco es del todo correcta. Esto es debido a que el suavizado produce que al intentar aplicar el detector no se cumpla la hipótesis de partida, es decir, un píxel borde con diferentes niveles de intensidad a ambos lados. El objetivo será encontrar una expresión que permita encontrar los parámetros del borde de forma exacta en una imagen suavizada.

Método de Suavizado

El método de suavizado aplicado en la fase previa a la detección es un suavizado de tipo gaussiano, un filtro espacial paso-bajo (a menudo también son denominados como *promediado*

en el entorno [18]). Es un método de los denominados isotrópicos (difusión isotrópica), esto es, que el efecto de suavizado se produce de manera uniforme en todas direcciones. La desventaja que causan este tipo de filtros es que no se preservan los contornos (a diferencia de los métodos anisotrópicos (difusión anisotrópica) como los trabajos realizados por Perona y Malik en [32] o por Krissian y Aja-Fernández en [26], por poner dos ejemplos, que se encuentran basados en esquemas de ecuaciones diferenciales en derivadas parciales). No obstante, la solución pasará por plantear las ecuaciones necesarias para que sea posible estimar la posición exacta del contorno, así como el resto de sus características, a pesar de este tipo de filtrado.

En el caso que nos ocupa, para el suavizado se va usar una máscara de radio 1. Dicha máscara posee la siguiente estructura:

$$H_1 = \begin{pmatrix} a_{11} & a_{01} & a_{11} \\ a_{01} & a_{00} & a_{01} \\ a_{11} & a_{01} & a_{11} \end{pmatrix} \quad (3.26)$$

donde se cumple que

$$\begin{aligned} 1 &= a_{00} + 4a_{01} + 4a_{11} \\ a_{00} &\geq a_{01} \geq a_{11} > 0 \end{aligned} \quad (3.27)$$

Al realizar la convolución de la imagen de entrada F con la máscara H_1 se obtiene la imagen G tal que

$$G = F * H_1 \quad (3.28)$$

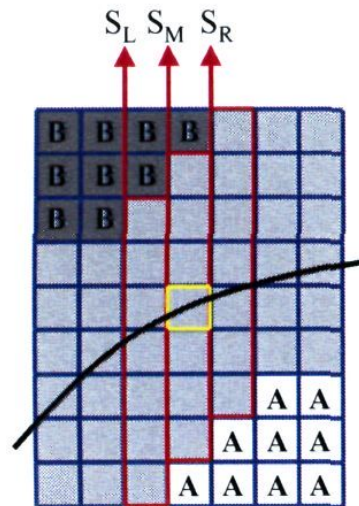


Figura 3.7: Ventana para el cálculo de los parámetros del contorno en una imagen suavizada [42]

Se demuestra que usando solamente la información de la imagen suavizada G , independientemente de los valores de a_{ij} usados en la máscara de suavizado (aunque sabiendo que se trata de una máscara con radio 1), se pueden obtener de forma exacta los parámetros del contorno a

través de las ecuaciones que aparecen a continuación. A diferencia del método original, y debido al efecto de difuminado provocado por la convolución, la información de vecindad de cada píxel necesaria para aplicar el método es la incluida en una ventana 9×3 centrada en el píxel, en lugar de una de tamaño 5×3 . No obstante, se consideran los límites de la ventana para el cálculo de las sumas, puesto que si no los valores que se cogen para realizar la estimación de A y B podrían estar muy alejados y tener valores que no serían los deseados para el cálculo. Estos límites serán diferentes en función de la pendiente, por ello se podría decir que se tendría una especie de ventana escalonada (fig. 3.7). S_L está desplazada un píxel hacia abajo con respecto a S_M , y S_R está desplaza un píxel hacia arriba. Esto se hace para pendientes entre 0 y 1. Sin embargo, para pendientes entre 0 y -1, la situación es la simétrica, S_L un píxel más arriba que S_M , y S_R un píxel más abajo.

Teniendo en cuenta todo lo anterior, a partir de los datos de la imagen suavizada G se pueden estimar los parámetros del contorno de la imagen original F (posición, pendiente, magnitud del salto de intensidad y curvatura, medidos sobre la vertical central del píxel), llevando a cabo los siguientes pasos:

En primer lugar se obtienen las sumas de las columnas izquierda, central y derecha de la siguiente manera:

$$\begin{aligned} S_L &= \sum_{k=-3+m}^{3+m} G_{i-1,j+k} \\ S_M &= \sum_{k=-3}^3 G_{i,j+k} \\ S_R &= \sum_{k=-3-m}^{3-m} G_{i+1,j+k} \end{aligned} \tag{3.29}$$

donde

$$m = \begin{cases} 1 & \text{si } G_x(i, j)G_y(i, j) > 0 \\ -1 & \text{si } G_x(i, j)G_y(i, j) < 0 \end{cases} \tag{3.30}$$

Posteriormente, se obtiene una estimación para los valores de intensidad a ambos lados del borde:

$$\begin{aligned} B &= \frac{G_{i-m,j-3} + G_{i-m,j-4} + G_{i,j-4}}{3} \\ A &= \frac{G_{i,j+4} + G_{i+m,j+4} + G_{i+m,j+3}}{3} \end{aligned} \tag{3.31}$$

De esta manera se pueden obtener los coeficientes de la ecuación $y = a + bx + cx^2$ como:

$$\begin{aligned} c &= \begin{cases} 0 & \text{si se buscan contornos rectos} \\ \frac{S_L + S_R - 2S_M}{2(A - B)} & \text{si se buscan contornos de segundo grado} \end{cases} \\ b &= m + \frac{S_R - S_L}{2(A - B)} \\ a &= \frac{2S_M - 7(A + B)}{2(A - B)} - \frac{1 + 24a_{01} + 48a_{11}}{12}c \end{aligned} \tag{3.32}$$

Finalmente, se calculan los parámetros del contorno. Al igual que en los anteriores casos, se considera que el corte con la vertical está en $(0, a)$ y el cálculo del vector normal se realiza como en 3.22. Sin embargo, ahora la curvatura se obtiene como [42]:

$$K = \frac{2cn}{(1 + b^2)^{\frac{3}{2}}} \quad (3.33)$$

donde

$$n = \begin{cases} 1 & \text{si } G_y(i, j) > 0 \\ -1 & \text{si } G_y(i, j) < 0 \end{cases} \quad (3.34)$$

3.1.2.4. Restauración

El método propuesto en la sección anterior para la localización de contornos en imágenes afectadas por ruido a través de un preproceso de suavizado, funciona con imágenes que no se encuentran afectadas por mucho ruido. Es decir, si se da como entrada una imagen que tiene un nivel de ruido grande, el algoritmo fallará al intentar realizar la detección de los contornos. Esto se debe principalmente a que el suavizado no ha sido suficiente para eliminar el ruido que afecta a la señal (en nuestro caso la imagen) y que hace que sigan existiendo perturbaciones en los niveles de intensidad. La persistencia de este fenómeno hace que no sea posible localizar con exactitud las características de los bordes.

Lo más lógico inicialmente es tratar de aplicar algún tipo de esquema iterativo que permita lograr eliminar este ruido que afecta a la imagen. Con anterioridad se comentó que el tipo de método aplicado es de los denominados filtros de difusión isotrópica. Es decir, cuando se aplica el filtro sobre la imagen, la intensidad de cada píxel queda promediada por los de una cierta vecindad, en función de los valores que contengan los elementos de la máscara gaussiana de suavizado. Aplicar de forma reiterada un método como éste causa una difusión generalizada y, por tanto, una pérdida total o parcial de los bordes; haciendo que nuestro objetivo no sea posible.

Por otro lado, también se habló muy por encima de otro tipo de métodos de filtrado de difusión anisotrópica. Este tipo de filtros consiguen difuminar el ruido preservando los bordes. Para dicho fin cuentan en su esquema diferencial con un término que posibilita no suavizar los bordes, término al que normalmente se le suele denominar energía (Φ). La idea principal de este tipo de métodos es respetar el contorno bien realizando en la dirección normal al mismo o bien suavizando a lo largo de la línea de éste. Describamos brevemente los dos ejemplos citados con anterioridad:

- En [32] Perona y Malik proponen un método que logra preservar los contornos a partir de un coeficiente de difusión que es una función decreciente de la norma del gradiente, haciendo que en las zonas donde el valor del gradiente sea alto no se realice suavizado.
- En [26] Krissian y Aja-Fernández plantean algo similar. En el método anterior sucede que en cada iteración se produce la misma difusión independientemente de la cantidad de ruido persistente en la imagen tras el proceso de suavizado. Además, que el término de

difusión sea dependiente del gradiente produce que sea necesaria la selección de un valor de umbral (típico en la detección de contornos mediante el gradiente). Por ello, los autores proponen usar un región de interés en la imagen, y a partir de estadísticas locales en ella (varianza global del ruido), hacer que el proceso de difusión pare (si ya se ha suavizado lo suficiente, que no se siga). También el método propuesto hace uso de un modelo del ruido (estimador del ruido).

En esta sección se propone un método iterativo de difusión del ruido pero que sea capaz de preservar los contornos, de forma que sea posible posteriormente localizarlos con exactitud. En [42] se presenta el método dividido en dos partes. Aquí se tratará en la medida posible ser más simples y hacer una descripción más reducida.

Debido a que el principal interés es obtener de manera exacta todos las características de los bordes, el uso de alguno de los métodos como los descritos no es lo adecuado. Esto se debe a que el proceso de difusión modifica la imagen de entrada de tal manera que ya no cumple la hipótesis de partida y, por ello, no se obtendrán los resultados esperados.

Algoritmo Básico

Sea F_0 la imagen original a la cual se le aplica una máscara de suavizado H_1 para obtener la imagen suavizada G_0 . A esta nueva imagen se le puede aplicar el algoritmo descrito en 3.1.2.3. Dicho algoritmo devolverá información sobre los píxeles en los que existe un contorno, así como sus parámetros (posición, orientación y salto de intensidad). Con esta información, podría generarse una nueva imagen, F_1 , tal que en las zonas alejadas de los píxeles pertenecientes a un borde, mantengan el valor de la imagen suavizada, G_0 , y en las zonas cercanas a los píxeles borde, asignarles un valor calculado a partir de los parámetros obtenidos por el método para dicho píxel.

La forma de calcular esos valores sería: sea el píxel (i,j) un píxel que ha sido etiquetado por el método como un píxel borde. Esto significa que, a partir de los valores de una ventana 9×3 centrada en dicho píxel, el método ha deducido que por él atraviesa una curva que se ajusta a la ecuación $y = a + bx + cx^2$, dejando las intensidades A y B a cada lado. Por tanto, podría generarse una imagen sintética con las mismas dimensiones (9×3) a partir de los parámetros estimados por el método.

Para calcular la intensidad de cada uno de los 27 píxeles de esa ventana, simplemente se aplica la hipótesis de partida pero a la inversa. Es decir, dado un píxel centrado en el punto (x_k, y_k) , dada una curva $y = a + bx + cx^2$, y dado dos valores de intensidad a cada lado del borde, A por debajo y B por encima, se calcula el área relativa interior al píxel bajo la parábola, S , tal que $0 \leq S \leq 1$, y deducir su intensidad como $I = SA + (1 - S)B$.

Aplicando este esquema, en el caso en que la imagen original fuera ideal, se obtendría una pequeña imagen sintética que coincidiría exactamente con la original. Pero si la imagen original tuviese algo de ruido, es entonces cuando mayor interés habría, ya que la pequeña imagen sintética generada seguiría dando como resultado los mismos parámetros para el contorno que la original, pero con una calidad visual superior.

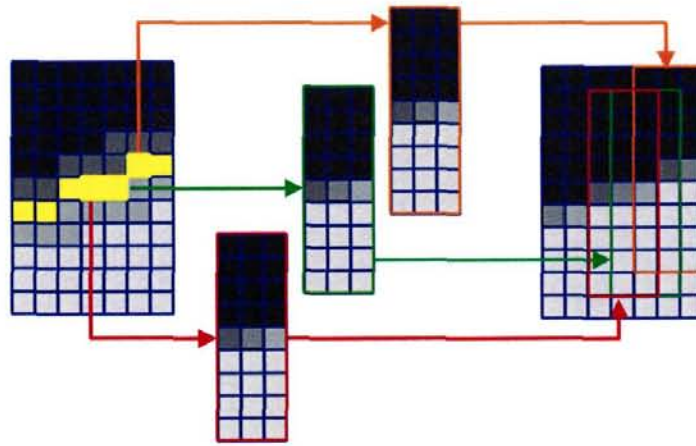


Figura 3.8: Combinación de imágenes sintéticas para obtener la imagen restaurada [42]

Llevando a cabo este mismo proceso para todos y cada uno de los píxeles borde, ocurrirá que a lo largo de un contorno, se irán generando diferentes sub-imágenes con solapes entre ellas (fig. 3.8). Para conseguir un resultado que combine la información en una sola imagen F_1 (entrada para la siguiente iteración) se plantea lo siguiente:

Se crean dos imágenes de la misma resolución que la imagen original. Una de ellas es una imagen de contadores, C , donde el valor de cada píxel indica el número de ventanas que han sido generadas y en las que dicho píxel estaba incluido. La segunda imagen es una imagen de intensidades, I , donde el valor de cada píxel indica la suma acumulada de las intensidades que se han ido calculando para cada píxel. Así, cuando se haya procesado todos los píxeles borde y generado todas las sub-imágenes necesarias, los píxeles donde $C = 0$ indicarán que están lejos de cualquier borde, y por lo tanto tendrán el mismo valor que en la imagen suavizada G_0 . Por el contrario, los píxeles donde $C > 0$ indicarán que están incluidos en una o más ventanas sintéticas, y por lo tanto su color será el cociente $\frac{I}{C}$.

Con ello, partiendo de una imagen F_0 , se consigue una nueva imagen F_1 donde las zonas en las que no hay borde están suavizadas, y en las zonas cercanas a bordes, éstos aparecen más nítidos. La imagen F_1 tiene por tanto una calidad mayor, por lo cual se le puede aplicar el método para el cálculo de los parámetros de los bordes y se obtendrían resultados más regulares. Así se puede aplicar sucesivamente este esquema teniendo una progresión del tipo $\{F_0 \dots F_i \dots F_n\}$, siendo n el número de iteraciones, y donde cada imagen tendría una mejor calidad que la anterior.

Mejoras

- **Aceleración de las Iteraciones:** se puede acelerar el suavizado aumentando el peso de la columna central de las sub-imágenes sintéticas que se van generando. Hay que tener en cuenta que el píxel del que se calculan los parámetros del borde es el píxel central de dicha ventana. Es por ello que la columna central es la que más importancia tiene. En cada columna i de la imagen se solapan tres sub-imágenes diferentes (las centradas en las

columnas $i - 1$, i e $i + 1$). Según lo expuesto antes, la intensidad calculada para la columna i será entonces un promedio de las tres, cuando realmente la que más importancia debería tener sería la centrada en la columna i . Como solución el método propone que se de un cierto peso (mayor que 1) a la columna central.

- **Tratamiento de los Márgenes:** en el caso de los márgenes hay que hacer un tratamiento tanto en la etapa de suavizado como en la etapa de detección de los bordes. En relación al suavizado, como esta operación se realiza con una máscara de radio 1 (3×3), pueden existir tanto en la primera como en la última fila, al igual que en la primera y última columna. Una solución que se da en ocasiones a este problema es considerar que la imagen tiene una fila/columna más con el mismo valor que la última (es decir, añadir un borde exterior a la imagen de tamaño un píxel y con los valores que tenía en los márgenes). Desde el punto de vista de la detección de contornos esto no es lo más adecuado. Esto es debido a que si se tiene un borde que llegue hasta el margen no es correcto que éste cambie bruscamente de dirección y que no continúe con la que ya tenía. Otra solución es promediar sólo con los 5 vecinos adyacentes en lugar de los 8, pero tampoco es aconsejable. Lo que se realiza en las filas y columnas es promediar teniendo en cuenta sólo los píxeles vecinos en esa misma fila o columna y no las adyacentes. En relación a las cuatro esquinas sólo se tienen en cuenta los vecinos que están al lado y encima/debajo.

En cuanto a la detección de contornos y generación de sub-imágenes, se propone que en el caso en el que borde toque algún margen se estima solamente usando el valor de la intensidad que se encuentra disponible, es decir, la parte de la ventana que queda en el interior de la imagen, haciendo la generación de la imagen sintética de la misma manera, dejando el resto de píxeles igual que en la imagen original.

Restauración con Límites Variables

En lo descrito hasta el momento, sólo se ha considerado que se cuenta con un único borde dentro de la ventana usada para la detección. Si dentro de la misma hubiera más de uno, fallaría la detección, además de verse afectado por el proceso de suavizado previo. La solución propuesta en [42] para resolver este problema es el uso de límites variables en la ventana. El cálculo alternativo que se propone para las sumas es el siguiente:

$$\begin{aligned}
 S_L &= \sum_{k=l_1}^{l_2} G_{i-1,j+k} \\
 S_M &= \sum_{k=m_1}^{m_2} G_{i,j+k} \\
 S_R &= \sum_{k=r_1}^{r_2} G_{i+1,j+k}
 \end{aligned} \tag{3.35}$$

Esos límites $l_1, l_2, m_1, m_2, r_1, r_2$ se calculan para cada caso. La sumatoria de cada columna se llega hasta un píxel en que se estima que se ha alcanzado el valor de la intensidad de la zona que separaba el contorno, es decir, hasta donde el valor de la derivada alcance un mínimo.

Para estimar las intensidades A y B se utilizarán los píxeles extremos de las sumatorias. Las ecuaciones serían:

$$\begin{aligned} B &= \frac{G_{i-1,j+l_1} + G_{i,j+m_1}}{2} \\ A &= \frac{G_{i,j+m_2} + G_{i+1,j+r_2}}{2} \end{aligned} \quad (3.36)$$

para el caso de pendiente positiva, y

$$\begin{aligned} B &= \frac{G_{i,j+m_1} + G_{i+1,j+r_1}}{2} \\ A &= \frac{G_{i-1,j+l_2} + G_{i,j+m_2}}{2} \end{aligned} \quad (3.37)$$

para los casos de pendiente negativa.

Sólo restaría saber las expresiones para el cálculo de los coeficientes de la ecuación $y = a + bx + cx^2$:

$$\begin{aligned} c &= \begin{cases} 0 & \text{si se buscan contornos rectos} \\ \frac{S_L + S_R - 2S_M}{2(A-B)} + \frac{A(2m_2 - l_2 - r_2) - B(2m_1 - l_1 - r_1)}{2(A-B)} & \text{si se buscan contornos de segundo grado} \end{cases} \\ b &= \frac{S_R - S_L}{2(A-B)} + \frac{A(l_2 - r_2) - B(l_1 - r_1)}{2(A-B)} \\ a &= \frac{2S_M - (1 + 2m_2)A - (1 - 2m_1)B}{2(A-B)} - \frac{1 + 24a_{01} + 48a_{11}}{12}c \end{aligned} \quad (3.38)$$

El cálculo de la normal y la curvatura es exactamente igual que en el caso anterior.

Además de para la detección del contorno, los límites de las sumatorias son también un dato importante para la generación de la ventana sintética sobre la nueva imagen que se está generando. Ahora no serán siempre ventanas fijas de tamaño 9×3 , sino que solamente se genera los valores sintéticos para aquellos píxeles que han sido incluidos en la sumatoria [42].

Contornos muy Cercanos

Cuando dos contornos están muy cercanos entre sí, se está en una situación en la que el valor de la intensidad B entre los dos contornos se ha perdido en la imagen suavizada. Por tanto, aunque se extienda los límites de la sumatoria hasta alcanzar un mínimo de la derivada, en ese punto el valor de la intensidad no será el correcto, y ello hará que se produzca un error. La forma de poder detectar que se está en esta situación es la siguiente: cuando los contornos están muy próximos, antes de que la derivada llegue a un valor casi nulo, la subida empieza con fuerza muy pronto. Es decir, la condición para detectar este caso podría ser mirar el valor de la parcial uno o dos píxeles más allá de aquél donde se hace mínima, y ver si supera un cierto umbral.

Una vez se ha detectado esta situación, la solución para estimar el verdadero valor de B está en la imagen original. Suponiendo un caso ideal, aunque ese píxel tenga un valor diferente de B en la imagen suavizada, es exactamente B en la imagen original. Esto será así siempre y

cuando el grosor no esté muy por debajo de 2 píxeles, porque si el grosor es 1 píxel o menos, la intensidad del interior no podría recuperarse, ya que no existiría ningún píxel que cayera por completo en el otro color. Por lo tanto, el cálculo de B sería:

$$B = \frac{F_{i-1,j+l_1} + F_{i,j} + m_1}{2} \quad (3.39)$$

donde F es la imagen original. Para el valor de A , sólo se reescribirá su expresión cuando se esté en ese mismo caso (la expresión es similar a la mostrada para el caso de la intensidad B). En caso contrario, no será necesario.

Pero existe otro problema. Al intentar estimar los parámetros de la parábola, se obtendrán valores erróneos. Esto viene justificado porque las expresiones existentes se encuentran diseñadas para la imagen sobre la que se ha aplicado el suavizado y no la original. La solución pasa por generar una sub-imagen intermedia F' , de tamaño máximo 11×5 , centrada en el píxel borde cuyo contorno se está estimando. Dicha imagen coincidirá con los píxeles de la imagen original F excepto la zona situada sobre los píxeles que se usaron para estimar la intensidad B . En dicha zona, el valor será igual a B para todos los píxeles. Visualmente, dicha imagen es en realidad una versión idéntica a la original pero donde el segundo contorno ha sido eliminado.

De esta forma, esta imagen F' puede ser suavizada para obtener una nueva imagen G' , de tamaño máximo 9×3 , centrada en el mismo píxel. Dicha imagen G' será sobre la que se la que se aplica el método de límites variables [42].

3.2. Detección de Bordes en Imágenes Tridimensionales

Se ha visto que en el caso 2D una imagen puede representarse como una cierta función bidimensional $f(x, y)$ que mide la intensidad en cada punto del plano. También ahora se puede representar una imagen 3D como una cierta función tridimensional $f(x, y, z)$ que mide la intensidad en cada punto del espacio. De esta manera, se seguirá considerando borde a aquella zona en la que haya una variación brusca de la intensidad. Ahora los píxeles borde forman superficies en lugar de líneas.

3.2.1. Métodos Tradicionales a Nivel Vóxel

Al igual que sucediera en el caso bidimensional, existen una serie de métodos a nivel vóxel, los cuáles van a ser descritos brevemente.

La técnica más básica para detectar bordes sigue siendo calcular la derivada de la función y buscar máximos locales en la dirección del gradiente. Para estimar el vector gradiente en un vóxel se usarán esquemas parecidos al caso 2D. Es decir, primero se obtiene de forma discreta el valor de las tres derivadas parciales en dicho vóxel y, finalmente, se combinan para producir el vector gradiente.

Este vector indica la dirección de máxima variación en cada punto, y su módulo representa la magnitud de dicha variación. Esto significa que el vector gradiente tendrá magnitud pequeña sobre los puntos de las zonas homogéneas, y tendrá un valor muy alto justo en la zona donde se produce la transición de una intensidad a otra. Además, la dirección de dicho vector indicará en cada punto en qué dirección se produce ese cambio.

El proceso para calcular las derivadas parciales es equivalente a convolucionar la imagen usando tres máscaras tridimensionales. Las tres máscaras serían:

$$\begin{aligned}
 H_x &= \frac{1}{2h_x} \left[\begin{bmatrix} -\beta/4 & 0 & \beta/4 \\ -\alpha/4 & 0 & \alpha/4 \\ -\beta/4 & 0 & \beta/4 \end{bmatrix}, \begin{bmatrix} -\alpha/4 & 0 & \alpha/4 \\ -(1-\alpha-\beta) & 0 & (1-\alpha-\beta) \\ -\alpha/4 & 0 & \alpha/4 \end{bmatrix}, \begin{bmatrix} -\beta/4 & 0 & \beta/4 \\ -\alpha/4 & 0 & \alpha/4 \\ -\beta/4 & 0 & \beta/4 \end{bmatrix} \right] \quad (3.40) \\
 H_y &= \frac{1}{2h_y} \left[\begin{bmatrix} -\beta/4 & -\alpha/4 & -\beta/4 \\ 0 & 0 & \alpha/4 \\ \beta/4 & \alpha/4 & \beta/4 \end{bmatrix}, \begin{bmatrix} -\alpha/4 & -(1-\alpha-\beta) & -\alpha/4 \\ 0 & 0 & 0 \\ \alpha/4 & (1-\alpha-\beta) & \alpha/4 \end{bmatrix}, \begin{bmatrix} -\beta/4 & -\alpha/4 & -\beta/4 \\ 0 & 0 & 0 \\ \beta/4 & \alpha/4 & \beta/4 \end{bmatrix} \right] \\
 H_z &= \frac{1}{2h_z} \left[\begin{bmatrix} -\beta/4 & -\alpha/4 & -\beta/4 \\ -\alpha/4 & -(1-\alpha-\beta) & -\alpha/4 \\ -\beta/4 & -\alpha/4 & -\beta/4 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} \beta/4 & \alpha/4 & \beta/4 \\ \alpha/4 & (1-\alpha-\beta) & \alpha/4 \\ \beta/4 & \alpha/4 & \beta/4 \end{bmatrix} \right]
 \end{aligned}$$

Muchas de las máscaras utilizadas tradicionalmente utilizan estos esquemas, siendo extensiones de las máscaras usadas en 2D. De esta manera se tiene la máscara de Prewitt, donde todas las filas tienen el mismo peso, siendo $\alpha = \beta = 4/9$, cuya expresión es:

$$H_x = \frac{1}{18h_x} \left[\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \right] \quad (3.41)$$

Sobel usa $\alpha = 8/15$ y $\beta = 4/15$ lo cual produce la máscara:

$$H_x = \frac{1}{30h_x} \left[\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -2 & 0 & 2 \\ -3 & 0 & 3 \\ -2 & 0 & 2 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \right] \quad (3.42)$$

donde las filas vecinas en las direcciones principales tienen el doble de peso que las vecinas diagonales, y la fila central el triple.

Por último, L. Álvarez y otros [13] proponen una extensión de la máscara de Frei Chen para mantener la condición de que los bordes diagonales perfectos tengan la misma magnitud del gradiente que los bordes en las direcciones principales, aunque hay que mencionar que su interpretación de “bordes perfectos” sigue siendo diferente a la que se propone en la hipótesis de trabajo. Para ellos, un borde viene dado por una función:

$$F(x, y, z) = \begin{cases} 1 & \text{si } ax + by + cz \leq 0 \\ 0 & \text{si } ax + by + cz > 0 \end{cases} \quad (3.43)$$

donde (x, y, z) son las coordenadas enteras de un vóxel, y el plano $ax + by + cz = 0$ indica la orientación del borde. Para obtener el valor de los coeficientes α y β imponen que la norma del gradiente calculado con las máscaras sea la misma en el punto central $(0, 0, 0)$ para cualquier elección de a, b, c donde $a, b, c \in \{-1, 0, 1\}$. Los valores obtenidos son:

$$\begin{aligned}\alpha &= 2\sqrt{2} - \frac{4}{3}\sqrt{3} \simeq 0,519 \\ \beta &= 2 - 2\sqrt{2} + \frac{2}{3}\sqrt{3} \simeq 0,326\end{aligned}\tag{3.44}$$

El módulo del gradiente se calcularía como la raíz cuadrada de cada componente al cuadrado, es decir:

$$G(i, j, k) = \sqrt{F_x^2(i, j, k) + F_y^2(i, j, k) + F_z^2(i, j, k)}\tag{3.45}$$

donde $F_x(i, j, k)$, $F_y(i, j, k)$ y $F_z(i, j, k)$ son las tres imágenes resultantes de aplicar la convolución con las máscaras H_x , H_y y H_z respectivamente [42].

3.2.2. Detección de Contornos con Precisión Sub-vóxel

En la sección dedicada a la descripción de la detección de contornos con precisión sub-píxel se introdujo el término PVE (Partial Volume Effect - Efecto de Volumen Parcial). Redefiniendo el término para el caso en el que nos encontramos, cuando un vóxel forma parte de un borde tiene una intensidad intermedia, que se encuentra entre la intensidad del objeto y la intensidad del fondo (ver fig. 3.9). Por tanto, se afirma que la intensidad de un vóxel borde es la suma de la contribución de cada color, pesada por el área relativa que ocupa cada uno en el interior de dicho vóxel. Este efecto es muy conocido dentro del área de imagen médica. Hay que tener en cuenta que tanto en imágenes de resonancia como en tomografía, cada tejido orgánico produce un cierto valor de intensidad en la imagen. Esto significa que cuando en el interior del volumen correspondiente a un vóxel coexisten más de un tipo de tejido diferente, el valor obtenido para dicho vóxel será un promedio de los valores individuales producidos por cada tejido.

El uso de una herramienta matemática como es el vector gradiente aporta bastante información sobre la existencia de bordes en una imagen. Sin embargo, de nuevo hay que mencionar que para poder aplicar esta herramienta con rigor es preciso que nuestra función sea derivable (y por tanto continua) en todo su dominio (o al menos en aquellos puntos donde queramos evaluar el vector gradiente). Es por ello que todos los trabajos toman como premisa fundamental el hecho de que, aunque la imagen de entrada sea una señal con valores discretos, dicha señal proviene del muestreo de una cierta función continua y derivable, a la cual se le pretende calcular los bordes.

Al igual que en el caso bidimensional, la interpretación de este método es diferente. Se asume que justo en las zonas donde hay un borde en la imagen, es decir, en las superficies de

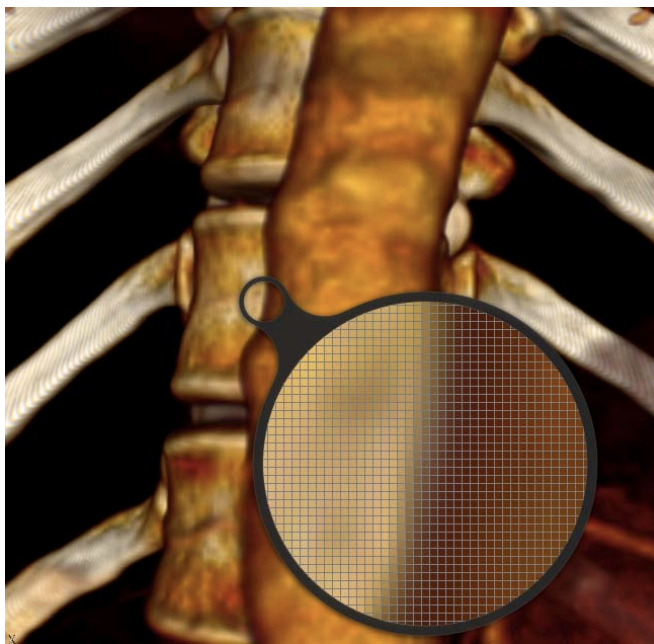


Figura 3.9: Efecto de volumen parcial en una reconstrucción volumétrica de la región toracoabdominal

los objetos presentes en ella, habrá una discontinuidad. Dicho modelo hace que ya no tenga sentido hablar de “gradiente de un vóxel”, sino del “vector normal a la superficie que pasa por ese vóxel”.

En este trabajo sólo se ha incluido la versión más simple del detector con precisión sub-vóxel, que será descrito a continuación. Asimismo, se describirá el caso en el que la derivada de mayor valor es en y , ya que los otros dos casos x, z son análogos a este.

3.2.2.1. Contornos de Segundo Orden

Primero hay que definir claramente el sentido y la orientación de las variables, como se ve en la figura 3.10. El origen de coordenadas se colocará en el centro geométrico del vóxel al que se quiere estudiar (el vóxel central en la figura), el cual tendrá por coordenadas $(0, 0, 0)$. El valor de y crece hacia arriba. El valor de z crece hacia la derecha. el valor de x crece hacia atrás.

Si se llama F a la imagen 3D de vóxeles, se estará diciendo que para el caso de la figura se cumple que $|F_y| > |F_x|, |F_z|$, y por lo tanto, para este caso, la superficie puede considerarse como una función explícita, con la y despejada en función de las variables x, z . Una vez resuelto este caso, los demás se resolverán de forma análoga intercambiando las variables.

En cuanto al tamaño de lo que se considera “vecindad local”, en la figura 3.10 se ha usado un entorno o sub-imagen de tamaño $3 \times 5 \times 3$, de forma similar al caso bidimensional, donde se usaba una ventana 3×5 . Es preciso recordar que estas dimensiones se debían a la imposición de que, al menos en el caso lineal para pendientes entre 0 y 1, la línea de contorno cruzara

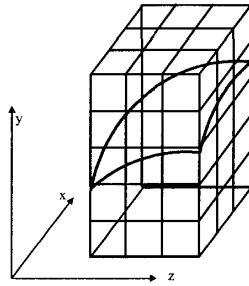


Figura 3.10: Orientación de los ejes de coordenadas (aunque realmente están en el centro del vóxel central)[42]

siempre el entorno de izquierda a derecha, sin tocar nunca los límites inferior ni superior de la sub-imagen.

Sin embargo, al trabajar en 3D, y aún considerándose el caso lineal, no se puede garantizar que la superficie del contorno no toque el suelo o el techo de dicho entorno. Es decir, considerando que la superficie fuese un plano (aproximación lineal), y que en el caso peor las pendientes de x y de z fuesen cercanas a 1 (45°), no se puede garantizar que el plano quede completamente dentro de la sub-imagen, por lo que se necesitaría un entorno con al menos 7 vóxeles de altura para asegurarse.

La importancia de que la superficie no toque el suelo de la sub-imagen es para que, después, cuando se realice la integración de cada una de las columnas, resulten expresiones lineales, puesto que de lo contrario el sistema a resolver para hallar los coeficientes del paraboloides sería bastante complejo. Sin embargo, aumentar la altura de la sub-imagen a 9 vóxeles significaría estar usando vóxeles muy alejados para deducir los parámetros del vóxel central, y en realidad esos casos extremos no se darán con mucha frecuencia. Por tanto, se usarán para los cálculos sub-ímagenes (o ventanas) de $3 \times 7 \times 3$.

Llegando a expresiones similares que en el caso 2D, el método podría resumirse en lo siguiente: sea $F(i, j, k)$ una imagen 3D por cuyo vóxel $(0, 0, 0)$ pasa por un contorno que separa dos zonas de intensidad A (debajo del contorno) y B (por encima), y que cumple que $|F_y|h_y > |F_x|h_x, |F_z|h_z$. Si se supone que en la vecindad de ese vóxel, el contorno se comporta como una superficie de segundo grado, se pueden estimar los parámetros de dicho contorno (posición, vector normal y curvatura) calculando previamente el paraboloides que mejor se aproxima a dicho borde. Los pasos son los siguientes:

- Obtener las sumas de las columnas de vóxeles de la vecindad, de la siguiente manera:

$$\begin{aligned}
 S_1 &= \sum_{j=-3}^3 F_{1,j,0} & S_2 &= \sum_{j=-3}^3 F_{0,j,-1} & S_3 &= \sum_{j=-3}^3 F_{0,j,0} \\
 S_4 &= \sum_{j=-3}^3 F_{0,j,1} & S_5 &= \sum_{j=-3}^3 F_{-1,j,0} & S_6 &= \sum_{j=-3}^3 F_{1,j,-m} + F_{-1,j,m}
 \end{aligned} \tag{3.46}$$

donde

$$m = \begin{cases} 1 & \text{si } F_x F_z \geq 0 \\ -1 & \text{si } F_x F_z < 0 \end{cases} \quad (3.47)$$

- Se obtiene una estimación para los valores de intensidad a ambos lados del borde:

$$\begin{aligned} A &= \frac{1}{9} \left(\begin{array}{l} F_{-\alpha,-3,\beta} + F_{-\alpha,-3,0} + F_{-\alpha,-3,-\beta} + F_{0,-3,0} + F_{0,-3,-\beta} + \\ F_{\alpha,-3,-\beta} + F_{-\alpha,-2,0} + F_{-\alpha,-2,-\beta} + F_{0,-2,-\beta} + F_{-\alpha,-1,-\beta} \end{array} \right) \\ B &= \frac{1}{9} (F_{\alpha,3,-\beta} + F_{\alpha,3,0} + F_{\alpha,3,\beta} + F_{0,3,0} + F_{0,3,\beta} + F_{-\alpha,3,\beta} + F_{\alpha,2,0} + F_{\alpha,2,\beta} + F_{0,2,\beta} + F_{\alpha,1,\beta}) \end{aligned} \quad (3.48)$$

donde

$$\begin{aligned} \alpha &= \begin{cases} 1 & \text{si } F_x F_y \geq 0 \\ -1 & \text{si } F_x F_y < 0 \end{cases} \\ \beta &= \begin{cases} 1 & \text{si } F_z F_y \geq 0 \\ -1 & \text{si } F_z F_y < 0 \end{cases} \end{aligned} \quad (3.49)$$

- Se detectan los coeficientes del paraboloido $y = a + bx + cz + dx^2 + fxz + gz^2$ como sigue:

$$\begin{aligned} a &= \frac{h_y}{24(A-B)} (28S_3 - S_1 - S_2 - S_4 - S_5 - 84A - 84B) \\ b &= \frac{h_y}{2h_x(A-B)} (S_1 - S_5) \\ c &= \frac{h_y}{2h_z(A-B)} (S_4 - S_2) \\ d &= \frac{h_y}{2(A-B)h_x^2} (S_1 + S_5 - 2S_3) \\ f &= \frac{mh_y}{2(A-B)h_x h_z} (S_1 + S_2 + S_4 + S_5 - S_6 - 2S_3) \\ g &= \frac{h_y}{2(A-B)h_x^2} (S_2 + S_4 - 2S_3) \end{aligned} \quad (3.50)$$

- La posición y el vector normal se calcula a partir de la expresión del paraboloido

$$\begin{aligned} \text{corte con la vertical central del v6xel: } & (0, a, 0) \\ \text{vector normal: } & N = \frac{A-B}{\sqrt{1+b^2+c^2}} [b \quad -1 \quad c] \end{aligned} \quad (3.51)$$

Existen otras extensiones del m6todo para la detecci3n en im6genes tridimensionales suavizadas y para la detecci3n de contornos muy cercanos, pero que no han sido incluidos en este trabajo.

Capítulo 4

Desarrollo

Armadados con las herramientas y habiendo adquirido el conocimiento de los métodos, llega el instante de afrontar la integración de los detectores de borde de alta precisión dentro del software **AMILab**. Al igual que se hiciese en el capítulo anterior, aquí también se encontrará una sección dedicada a 2D y otra a 3D.

Dentro de cada una de estas dos secciones se describirá cómo se ha realizado la integración. Se comienza por describir la adición de los métodos tradicionales, para pasar posteriormente a los sub-píxel y sub-vóxel. Además del proceso, se muestra el diagrama de clases, así como algunos ejemplos puntuales de aplicación. En su parte final cada uno de ellos tiene un apartado dedicado al uso de los métodos desde el lenguaje de *scripts* de **AMILab**, así como una detallada descripción de la interfaz. Cada sección, tanto en el caso bidimensional como tridimensional, termina con una serie de ejemplos de aplicación de los métodos, tanto sintéticos como de imágenes reales.

En la tercera sección del capítulo se describe cuál ha sido el mecanismo adoptado para la generación de las imágenes sintéticas, partiendo de lo existente dentro del software **AMILab** y añadiendo la noción de funciones analíticas. Con todo ello se consiguen generar las imágenes con efecto parcial que permiten probar los métodos.

4.1. 2D

4.1.1. Métodos a Nivel Píxel

Como se indicó en el capítulo dedicado a la descripción de los métodos, inicialmente se parte por hacer un repaso a los métodos más significativos a nivel píxel (descritos en la sección 4.3). Así mismo, se hace uso de esta revisión para hacer una primera toma de contacto con la experiencia de programar un código nuevo dentro del software **AMILab**. Ha de considerarse que esta parte se va a dividir en dos fases principales:

1. Implementación del código fuente de los detectores de bordes básicos.
2. Implementación de la interfaz que permite hacer uso de los mismos de una forma amigable.

4.1.1.1. Detectores de Bordes Básicos

En esta primera fase se realiza la implementación de algunos de los detectores de bordes básicos vistos en 4.3. Este desarrollo contará de dos ficheros: un fichero cabecera `.h` y un fichero de implementación `.cpp`, con el nombre `wrapBasicEdgeDetection` en ambos casos. Estos archivos serán incluidos dentro de la carpeta `WrapFilters` del proyecto de `AMILab`, que es donde se localizan todos los filtros que irán por *wrapping*. Es preciso decir que este proceso no siempre es así. Es decir, siendo fieles a la filosofía del entorno, la organización correcta sería la existencia de un fichero cabecera e implementación para el filtro, pero además, otros dos análogos pero dedicados al proceso de *wrapping*. Aquí, por simplicidad, se ha realizado de la manera descrita, aunque no obstante quería dejarse claro cuál es el proceso habitual que ha de seguirse. En relación a esto comentar que actualmente la versión de la rama *trunk* del software cuenta con un sistema de *wrapping* automático aún en desarrollo, pero que es funcional para la amplia mayoría de situaciones que pueden encontrarse. Éste último sistema sería el encargado de crear automáticamente los ficheros de *wrapping* asociados a los ficheros fuente bajo demanda. Es decir, sería preciso indicar que se quiere realizar ese proceso para las clases indicadas a través de un fichero al uso.

En el fichero cabecera antes descrito aparecerán todas las declaración prototipo de las funciones. En este caso se han codificado cuatro funciones que devuelven un objeto tipo `InrImage`:

- `wrapRoberts`: se encargará de la aplicación del método de Roberts.
- `wrapPSF`: esta función agrupa los métodos de Prewitt, Sobel y Frei-Chen; ya que los operadores de los mismos simplemente varían en función del valor de un parámetro `k`.
- `wrapGradient`: aplica el método del gradiente a una imagen.
- `wrapLaplace`: similar a todas las anteriores, pero en este caso se aplica el Laplaciano.

Nótese que todas las funciones llevan el prefijo `wrap-`. Esto es así para mantener cierta homogeneidad con la notación que se sigue a lo largo del resto de funciones que se encuentran en los otros ficheros de `WrapFilters`. Asimismo, cada función recibe por parámetro una estructura del tipo `ParamList`, que permite pasar un número variable de parámetros. Cada uno de ellos tiene asociado un método de extracción en función de su tipo. Esto último era así en la implementación antigua. Actualmente se ha pasado al uso más extendido de punteros inteligentes (*smart pointers*) que permite un método *template* de obtención del parámetro indicado entre `<>` en `get_val_ptr_param<>(InrImage*, ParamList*, int)`, aunque siguen existiendo la obtención de parámetros básicos como enteros o reales. No obstante, independientemente de esto último, entre paréntesis han de indicarse: la variable sobre la que se devuelve el resultado (un `InrImage` en el ejemplo anterior), un puntero a la lista de parámetros y un número entero que indica su posición relativa dentro de la lista (el cual al principio ha de ser inicializado a cero y que se irá incrementando de manera automática por los métodos `get`) respectivamente.

Cada una de estas funciones sigue un flujo básico de funcionamiento común a todas ellas. Inicialmente, se procede a una inicialización de las diferentes variables, entre las que se encuentran los operadores (comúnmente denominados máscaras) que se van a aplicar sobre la imagen. Dicha inicialización depende en cada caso del método que se vaya a aplicar. Tras ello, se procede a la extracción de los parámetros de la forma anteriormente descrita. Después, se invoca a la función `convolve2DImage`, que será descrita posteriormente. Con el resultado de dicha llamada, se realiza el cálculo del módulo del gradiente (si procede), colocando el resultado de cada punto en la imagen de salida, la cual será retornada al finalizar la función. Ha de tenerse en cuenta que, en el caso del Laplaciano, no se realizará el cálculo del módulo.

La función `convolve2DImage` citada anteriormente, realiza el proceso de convolución de una máscara sobre una imagen 2D. Recibe por parámetro dos objetos de tipo `InrImage*` (entrada y salida respectivamente) y una matriz 3×3 que representa el operador que se va a aplicar sobre la imagen de entrada. Es preciso tener en cuenta que durante este proceso hay que tener cierto cuidado en los valores límite o frontera que define la imagen, es decir, los bordes y esquinas. Para solventar esta cuestión se ha tomado la decisión descrita en 4.1. En ella puede observarse que, de manera general, la convolución será la suma de los productos de cada elemento del operador con los valores de los píxeles correspondientes de la imagen. De forma especial se tratan los límites, ya que en esos casos se copia el valor adyacente de la imagen para la realización del cálculo.

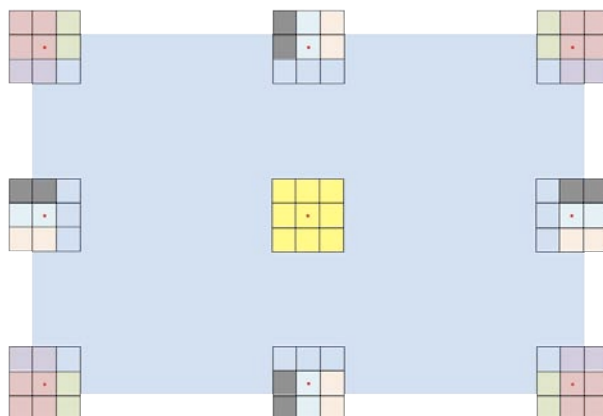


Figura 4.1: Convolución de una imagen 2D con una máscara 3×3

Con ello queda descrita la implementación del código que se encarga de llevar a cabo los filtros básicos de detección de bordes. Sólo resta tener en cuenta una serie de detalles. En primer lugar, es necesario añadir una entrada mediante el método `AddVar`, por cada función que se ha desarrollado, en el fichero `wrapFilters.cpp`. De esta forma se realiza el *wrapping* para luego poder invocar a las funciones desde el lenguaje de *scripts*. Por último, ha de añadirse el nuevo fichero `.cpp` dentro del archivo `CMakeLists.txt` del directorio, para que de esa forma `CMake` construya las dependencias que sean necesarias. Ha de tenerse en cuenta que, hasta el momento, `XCode` no realiza las modificaciones de los `CMakeLists.txt` de directorios superiores. Así que es necesario reconstruirlo con la aplicación gráfica de `CMake` o desde consola (terminal) con el comando `“cmake .”`.

4.1.1.2. Interfaz para los Detectores de Bordes Básicos

Una vez realizado el código de los detectores de bordes básicos, es preciso realizar una interfaz gráfica para que el usuario pueda interactuar con ellos de una forma cómoda. Las interfaces en **AMILab** se realizan a través de su lenguaje de *scripts*, en un fichero de extensión “.amil”, que será incluido dentro del directorio de *scripts* de **AMILab**. En su interior se definen los distintos botones y herramientas con las que contará el usuario. Indicar que en este momento aún no existía el concepto de herencia en el lenguaje de *scripts* (no obstante, sí la noción de clase y objeto), así que más adelante se describirá la herencia básica para la realización de una interfaz de usuario, así como las funcionalidades que proporciona.

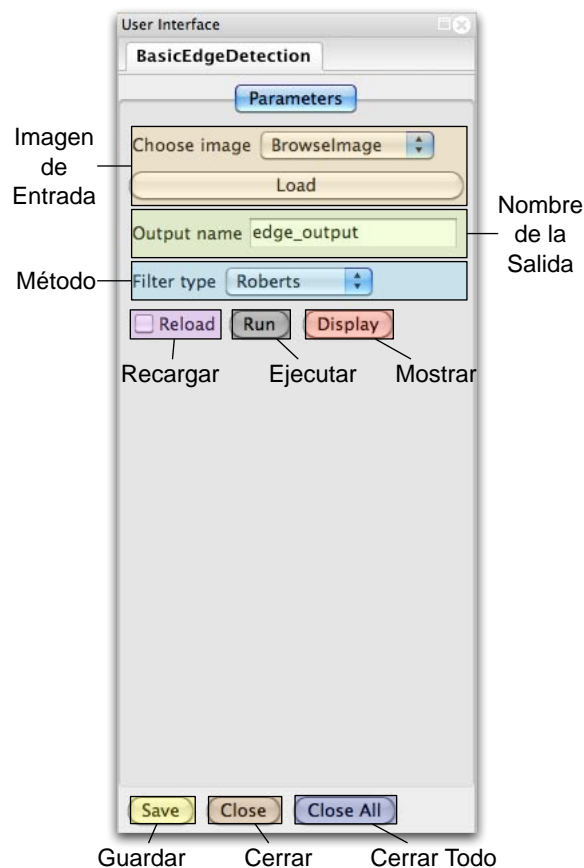


Figura 4.2: Interfaz para los detectores 2D a nivel píxel

El resultado de la interfaz realizada puede verse en la figura 4.2. En ella se muestra una pestaña con el nombre de dicha interfaz, en este caso **Basic Edge Detection** y una serie de parámetros. En primer lugar se encuentra el botón **Load**. Éste abre el cuadro de diálogo que permite realizar la carga de una imagen. Una vez se haya realizado la carga, la imagen aparecerá en la lista **Choose image**.

La caja de texto **Output name** permite escribir el nombre que se desea para la imagen de salida tras la aplicación del filtro. Además, aparece una lista desplegable, **Filter type**, que permite seleccionar qué tipo de filtro será aplicado. Una vez se ha seleccionado el filtro, sólo

bastaría darle a **Run** para ejecutar y a **Display** para mostrar el resultado en modo *compare* (en comparación con la imagen de entrada). Esto permitiría ir viendo el resultado de la aplicación de cada uno de los filtros.

AMILab permite el uso de **Callbacks** mediante el método **SetCallback(f)** del lenguaje de *scripts*, donde **f** es el nombre de una función dentro del *script*, aquella que realiza las operaciones que se desea se vuelvan a ejecutar tras cambiar algún parámetro (en general al realizar alguna acción sobre el objeto al que se le asigna el **callback**). El control de esta funcionalidad se ve reflejado en la interfaz mediante la casilla de verificación **Reload** de forma que, si ésta se encuentra inactiva, el funcionamiento será el anteriormente descrito. Si se encuentra seleccionada, se refrescará automáticamente la vista de los resultados en función del tipo de filtro que se elija.

El botón **Save** permite almacenar el resultado de manera global, de forma que tras su ejecución se pueda cerrar el *script* y la imagen se mantendrá dentro del conjunto de imágenes globales para algún procesado o uso posterior. **Close** cerrará la interfaz y **Close All** cerrará la interfaz y borrará todas las variables que haya creado la misma.

En la figura 4.3 se puede ver el resultado de aplicar diversos métodos a nivel píxel. En todas las imágenes se aplica un *zoom* en la misma región para ver el detalle de la detección.

4.1.2. Métodos Sub-Píxel

En esta sección será descrito el desarrollo de los métodos bidimensionales con precisión sub-píxel dentro del software **AMILab**. Es preciso decir que se parte del desarrollo existente realizado para el entorno **XMegaWave**. No obstante, hay que tener en cuenta cuestiones a la hora de realizar la integración del método en **AMILab**. Una cosa importante a tener en cuenta es que, como se ha dicho con anterioridad, existen dos aspectos diferenciados dentro de **AMILab**. Por un lado, se encuentra el código **C++** donde se encuentran implementados los métodos que se quieren añadir. Normalmente, siguiendo la estructura típica en dicho lenguaje, se cuenta con un fichero cabecera **.h** donde se indica el nombre de la clase, así como los atributos y definición prototipo de sus métodos. Hasta este momento es la típica estructura que cualquier programa tiene. Sin embargo, aquí se encuentra además la noción de lenguaje de *scripts*. **AMILab** posee su propio lenguaje de *scripts*, que permite crear una pequeña interfaz para probar el código que se ha desarrollado, creando ficheros con extensión **.amil**. Se tienen los elementos típicos de cualquier interfaz: botones, cajas de texto, *checkbox*, *combobox*, etc. Pero, ¿cómo es posible que desde el lenguaje de *script* se pueda acceder a las funciones desarrolladas en **C++**? Esto se logra a través de un mecanismo denominado *wrapping* (fig. 4.4). El *wrapping* o encapsulado de código permite acceder desde el lenguaje de *script* a funcionalidades que se encuentran implementadas en el código fuente. Simplemente ha de crearse un objeto de la clase *wrappeada* e invocar al método en cuestión con los parámetros apropiados (esto puede observarse en la figura 4.4 donde se ejemplifica a través de `object->functionName(parameters)`).

El mecanismo de *wrapping* es muy potente, puesto que todo aquello que requiera de capacidad de cómputo será realizado en **C++** (permitiendo así que sea lo más rápido posible) y a

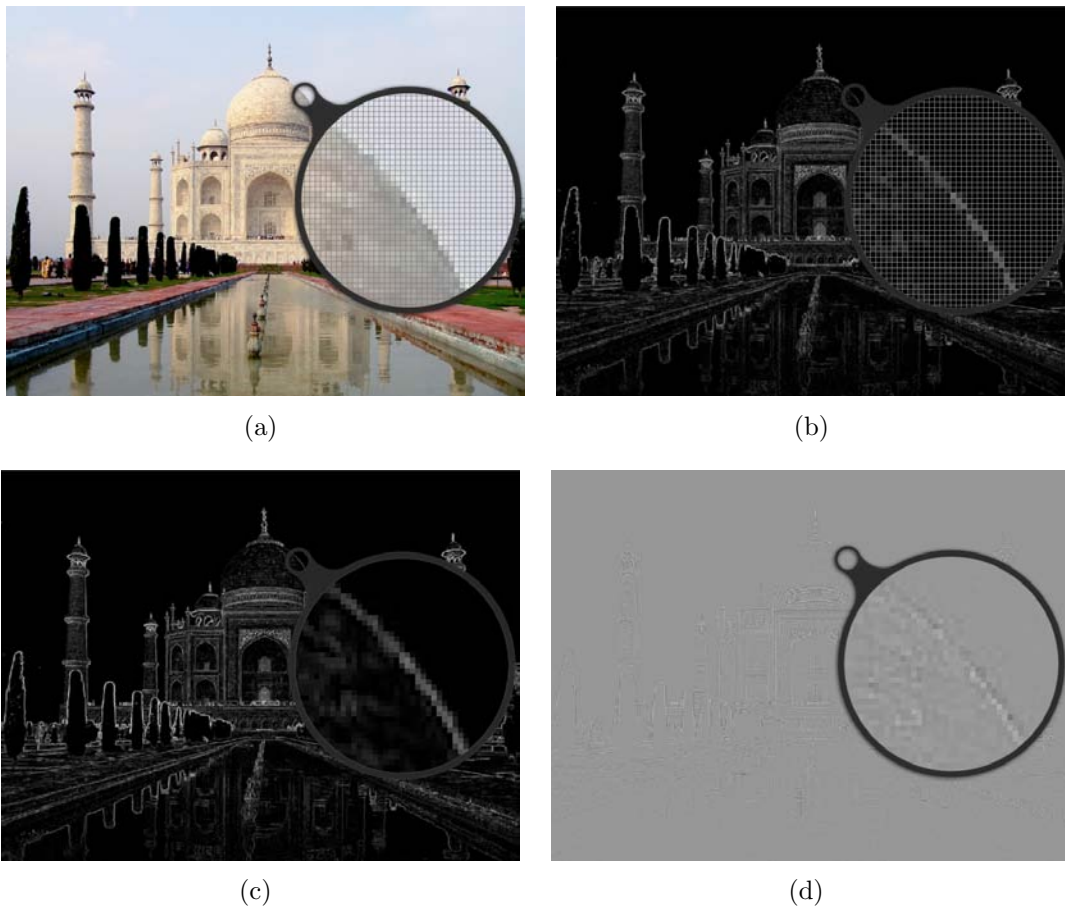


Figura 4.3: Diferentes resultados de la aplicación de detectores de bordes a nivel píxel: (a)Imagen original, (b)resultado del método de Roberts, (c) resultado del método de Frei-Chen y (d)resultado de aplicar el Laplaciano

su vez podrá invocarse desde el lenguaje de *scripts*, para aplicar dicho método a una imagen de entrada. En definitiva, se simplifica la manera de crear una interfaz y se pone a disposición del usuario a través del *wrapping* toda la potencia de los métodos implementados en C++. Aunque en la versión actual del software se cuenta con un sistema de *wrapping* automático, el relativo al trabajo realizado en este proyecto ha sido a través de un procedimiento manual. Esto deja abierto el camino a realizar una adaptación al procesamiento automático.

A continuación se describirán los pasos llevados a cabo para incluir la detección de contornos con alta precisión para imágenes bidimensionales dentro del software AMILab. Antes de continuar, es necesario decir dónde está ubicado el código de la solución adoptada. El código fuente se encuentra situado en `src/Algorithms/Segmentation` dentro del fichero denominado `SubPixel2D.cpp` (mismo nombre que la clase que contiene). El fichero cabecera se encuentra en la misma ruta, pero en lugar de dentro de la carpeta `Segmentation` se encuentra en `include` (de manera genérica, cada sección tiene un directorio `include` donde se encuentran todos los ficheros cabecera).

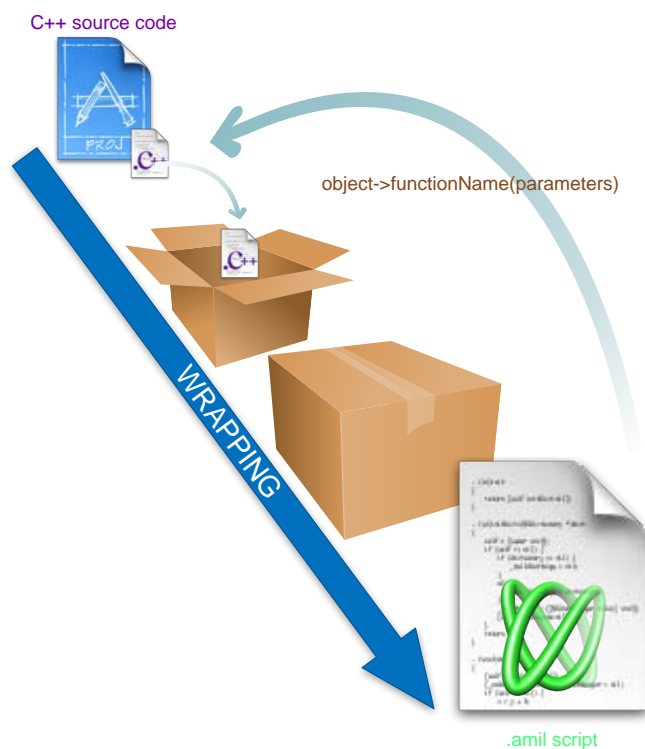


Figura 4.4: Proceso de *wrapping*: desde C++ hasta el lenguaje de *scripts* de AMILab

4.1.2.1. Método Sub-Píxel de Segundo Orden Básico

El primer hecho diferencial que ha de tenerse en cuenta respecto al código desarrollado para el entorno *XMegaWave* está relacionado con las imágenes. Mientras que en éste son consideradas como un vector (puntero a la dirección constante del primer elemento de un vector), en *AMILab* existe el tipo *InrImage* para el manejo y tratamiento de las imágenes. Por tanto, es preciso hacer una adaptación del diseño existente en la anterior implementación. Por tanto, el manejo de la estructura de datos que representa la imagen pasa de ser vectorial a matricial, ya que los métodos asociados a la clase *InrImage* son del tipo *InrImageObject->method(x,y,z)*, donde (x, y, z) representan las coordenadas espaciales dentro de la imagen. Nótese que en el caso de dos dimensiones la tercera componente será ignorada. Es decir, siempre se la asignará a z el valor cero ¹, mientras que x e y irán variando. Ambas componentes bidimensionales crecen de izquierda a derecha (x) y de arriba a abajo (y) respectivamente (se considera por tanto que el origen de coordenadas se encuentra en la esquina superior izquierda de la imagen).

Además de lo anterior, ha de tenerse en cuenta que el proceso de *wrapping* ha de ser lo más limpio posible. Para intentar mantener esta filosofía, es preciso realizar la adaptación adecuada del antiguo desarrollo, teniéndolo como una función interna, de forma que el *wrapping* haga de interfaz entre el lenguaje de *scripts* y el cómputo necesario para la aplicación del método. De esta forma se tendrá un diseño compacto.

¹La estructura de datos representada en la clase *InrImage* varía sus dimensiones en el rango $0 \dots n - 1$, al igual que sucede con los vectores y matrices en C++.

En [42] se comienza explicando el método para contornos de primer orden, para luego pasar a orden dos. Aquí se ha hecho una generalización entre ambos. Esto es posible a través de la adaptación de uno de los coeficientes de la curva que mejor se adapta al contorno. En el caso en el que se esté localizando un contorno de primer orden, el coeficiente c de la ecuación $y = a + bx + cx^2$ pasa a ser cero. En caso contrario, se calcula según lo descrito en la sección 3.1.2.2. Con ello es posible hacer que este método sea igual de factible tanto para primer como para segundo orden. Esto simplifica el proceso, ya que se cuentan con ambas funcionalidades en el mismo diseño. Esto se manifiesta en la interfaz en el *checkbox* denominado “1st order” (véase figura 4.23).

Una de las principales mejoras aportadas al método está relacionada con la gestión de memoria. En la antigua implementación, se reservaba memoria dinámica para cada uno de los parámetros del contorno que se querían calcular (intensidades, coeficientes de la curva, curvatura, etc.) a través de una operación del tipo `*v = (type*)malloc(size*sizeof(type))`. Esto tiene el principal problema de que independientemente de la cantidad de píxeles borde que haya, se hace una reserva de memoria igual al tamaño de la imagen multiplicado por el número de parámetros que se quieran calcular ². El enfoque aquí será un poco diferente. Se va hacer uso de una clase que represente la información referente a un píxel borde que debería ser guardada (figura 4.5). Dicha información sería: los valores de intensidad a ambos lados del borde, el tipo de borde (horizontal o vertical), los coeficientes de la parábola con el mejor ajuste al contorno, el valor de la curvatura y la posición (x, y) del píxel en cuestión. Como suele ser normal, la clase cuenta además con los métodos `set` y `get` precisos para asignar un valor o acceder al mismo de cada uno de los atributos. Dentro de estas funciones una de las más interesantes es `setBorderPixelValues`, pero será descrita posteriormente.

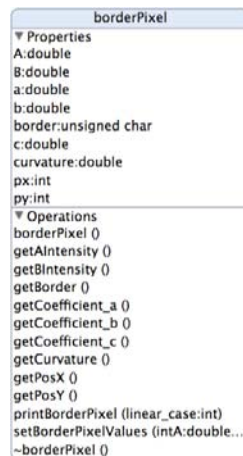


Figura 4.5: Clase `borderPixel`, encargada de guardar todos los parámetros relativos a un píxel borde

Quizás esta situación no es tan apremiante para el caso bidimensional pero si puede ser más grave en 3D. Estadísticamente, el porcentaje de píxeles borde dentro de una imagen son

²Para ser estrictos, el cálculo correcto sería la dimensión de la imagen multiplicada por el número de bytes necesarios para cada tipo de dato en memoria. La sumatoria de esos productos sería la cantidad total de memoria necesaria para albergar los datos.

minoritarios frente al resto. Es por ello que, si se representa sólo la información referente a éstos, el consumo de recursos durante la ejecución del método será menor, y evidentemente será más notable en el método tridimensional.

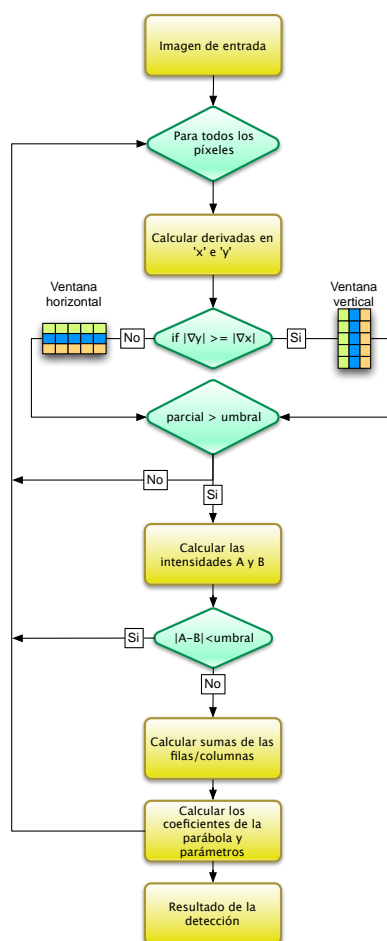


Figura 4.6: Diagrama de flujo del método de detección sub-píxel de orden dos

En la figura 4.6 se encuentra una descripción simplificada del diagrama de flujo que realiza el algoritmo de detección. A partir de la imagen de entrada se va a ir recorriendo todos y cada uno de los píxeles para calcular los contornos. Es necesario decir que en esta primera versión no se hace un tratamiento de los límites superior, inferior, izquierdo y derecho de la imagen. Como se hace uso de una ventana de 5×3 (en el caso horizontal 3×5) se deja un margen externo para hacer los cálculos sin salirse.

Una vez calculadas las parciales en x e y , se ve cuál es la mayor de las dos, para saber si hay que usar una ventana horizontal o vertical. Nótese que aunque aparece el símbolo matemático usado para el gradiente (el operador nabla) no se trata de estimar el valor de éste, sino del valor de la derivada parcial.

Por simplicidad, el resto de pasos se han puesto en común, ya que independientemente de la orientación, los cálculos son los mismos (simplemente se opera o por filas o por columnas). Aclarado esto, en primer lugar es necesario ver si el valor absoluto de la parcial supera el umbral,

ya que en caso contrario no se harán los cálculos y se pasará a tratar el siguiente píxel de la imagen. Además de ello, el método intenta ver si esa parcial es máxima en la fila/columna, para determinar que realmente se está seleccionando el píxel candidato correcto.

El cálculo de las intensidades se hará usando los píxeles que se encuentran en las esquinas de la ventana. La selección de los mismos viene determinado por el valor del producto de las parciales en x e y . En función de si la diferencia de intensidades en valor absoluto es menor que el umbral ($|A - B| < threshold$), se pasan a hacer los siguientes cálculos o se pasa a procesar el siguiente píxel candidato. En caso negativo, se hacen las sumas de las filas/columnas, para posteriormente calcular los parámetros del contorno, es decir, los coeficientes de la parábola con el mejor ajuste al borde y la curvatura del mismo. Es en este instante cuando se hace uso de la nueva funcionalidad para una mejor gestión de la memoria.

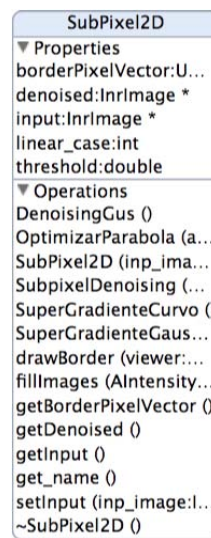


Figura 4.7: Diagrama de la clase SubPixel2D

Anteriormente se había descrito la existencia de la clase `borderPixel` como representación de la información que se ha de guardar respecto de un píxel borde (fig. 4.5). Esta clase es usada por la clase `SubPixel2D` (fig. 4.7) a través de un atributo de la clase de tipo `vector` de la STL de C++. De esta manera, cada vez que se lleva a cabo la detección de un píxel borde, se inserta en un objeto del tipo `pixelBorder` la información relativa al mismo a través del método `setBorderPixelValues`. Finalmente, este objeto se inserta en el vector `borderPixelVector` mediante la función `push_back`. Con ello se tendrá única y exclusivamente la información relativa de los píxeles considerados como borde, sin estar reservando tanta memoria como en la implementación anterior. Con ello, se obtendría el resultado final de la detección haciendo uso del método de orden dos básico.

Pero, aparte de realizar la detección, es necesaria ponerla a disposición del lenguaje de *scripts* de `AMILab` a través de un procedimiento descrito anteriormente, el *wrapping*. Es en este momento cuando surge un problema y es que el *wrapping* no está preparado para devolver un vector de la STL de la clase `borderPixel`. Se toma por tanto la siguiente solución: en `AMILab` existe la clase `AMIObject`. Ésta puede ser creada con o sin un *link* a un objeto de una clase. Si el objeto de la clase está presente, creará un contexto de variables ejecutando el código presente

en el objeto de clase. En otro caso será considerado como un espacio de nombres (*namespace*). La idea es encapsular el resultado dentro de un `AMIObject`, y de esta manera poder devolverlo al lenguaje de *scripts*. Es por eso que en el *wrapping* del método actual, una vez que se tiene el resultado se crea una estructura como la que puede verse en la figura 4.8.

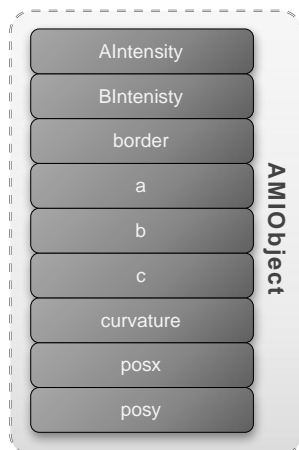


Figura 4.8: Estructura de tipo `AMIObject` que encapsula el resultado de la detección de bordes

redibujado conveniente.

En la figura 4.9 se puede ver un ejemplo de resultado sobre una imagen sintética de un círculo de radio 20 centrado en el píxel central de una imagen de 100×100 píxeles. Sobre la generación de estas imágenes para las pruebas se habla en la sección 4.3.

En ella puede observarse la precisión del método y como el contorno es dibujado dentro del píxel correspondiente. Pero se está hablando del caso de una imagen ideal sintética. ¿Qué sucedería si esta imagen se ve afectada por ruido?

Un ejemplo de esto puede observarse en la figura 4.10. En ella se puede ver el resultado de aplicar el algoritmo básico de detección a una circunferencia de radio 20, centrada en el píxel central de una imagen de 100×100 píxeles a la que se le ha añadido un ruido gaussiano de desviación estándar 5. Se puede ver que con poco ruido se ve afectada la detección tanto en orientación como en la posición sub-píxel. En la siguiente sección se describirá cómo el método es capaz de resolverlo.

Dicha estructura encapsula el resultado de la detección en un conjunto de *smart pointers* a imágenes (`InrImage`) unidimensionales. De esta manera se tienen las intensidades, el tipo de borde, los coeficientes de la parábola, la curvatura y posición del píxel borde. Para el rellenado de esta estructura la clase `SubPixel2D` cuenta con el método `fillImages` (clase de la figura 4.7).

Solventado el problema de devolver el resultado al lenguaje de *scripts*, quedaría el dibujado de los bordes. Inicialmente se consideró hacer dicha operación directamente sobre el lenguaje de *scripts*, pero teniendo en cuenta que el proceso de refresco (cuando se hace zoom por ejemplo) podía ser lento, se llegó a una solución de compromiso entre el lenguaje de *scripts* y C++. Debido a este hecho, la operación será descrita en profundidad en el apartado 4.1.2.4. Sólo se dirá por ahora que básicamente se hace uso del *wrapping* de la función `DrawLine` del *device context* de `wxWidgets`, renombrada como `DrawLineZ` en `AMILab`. Dando en *callback* nuestra función al visor se consigue que, ante cualquier operación (*zoom* o traslación), se vuelva a llamar a la operación haciendo el

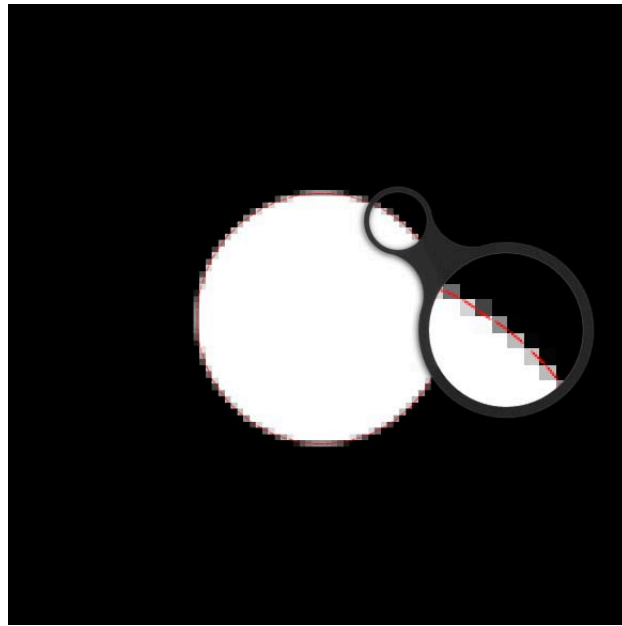


Figura 4.9: Resultado de aplicar el algoritmo básico de detección a una circunferencia de radio 20 y centrada en el píxel central de una imagen de 100×100 píxeles

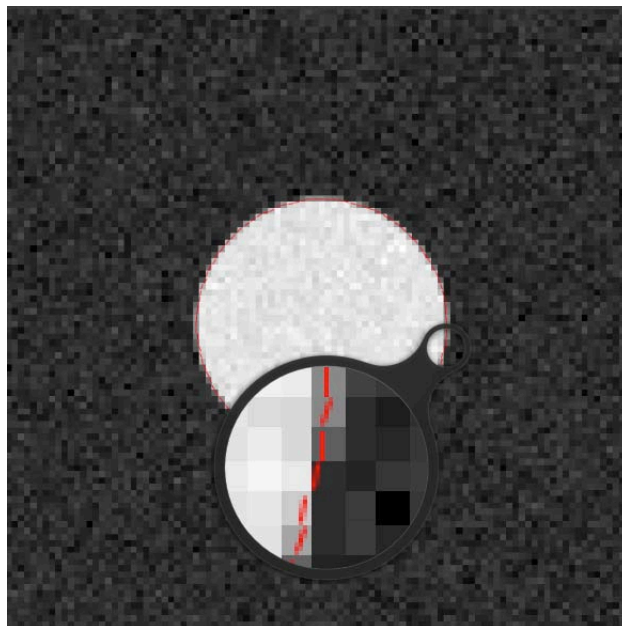


Figura 4.10: Resultado de aplicar el método básico de orden dos a una imagen con ruido

4.1.2.2. Método Sub-Píxel de Segundo Orden para Imágenes con Ruido

Anteriormente se ha visto que el método de detección básico con precisión sub-píxel de segundo orden es adecuado para imágenes ideales con bordes aislados. Pero, desde que se tiene la existencia de un cierto nivel de ruido, la detección se ve afectada en posición y orientación.

De hecho, a medida que aumenta el nivel de ruido, comienzan a aparecer detecciones espurias (*outliers*) en diversas zonas de la imagen (ver fig. 4.11).

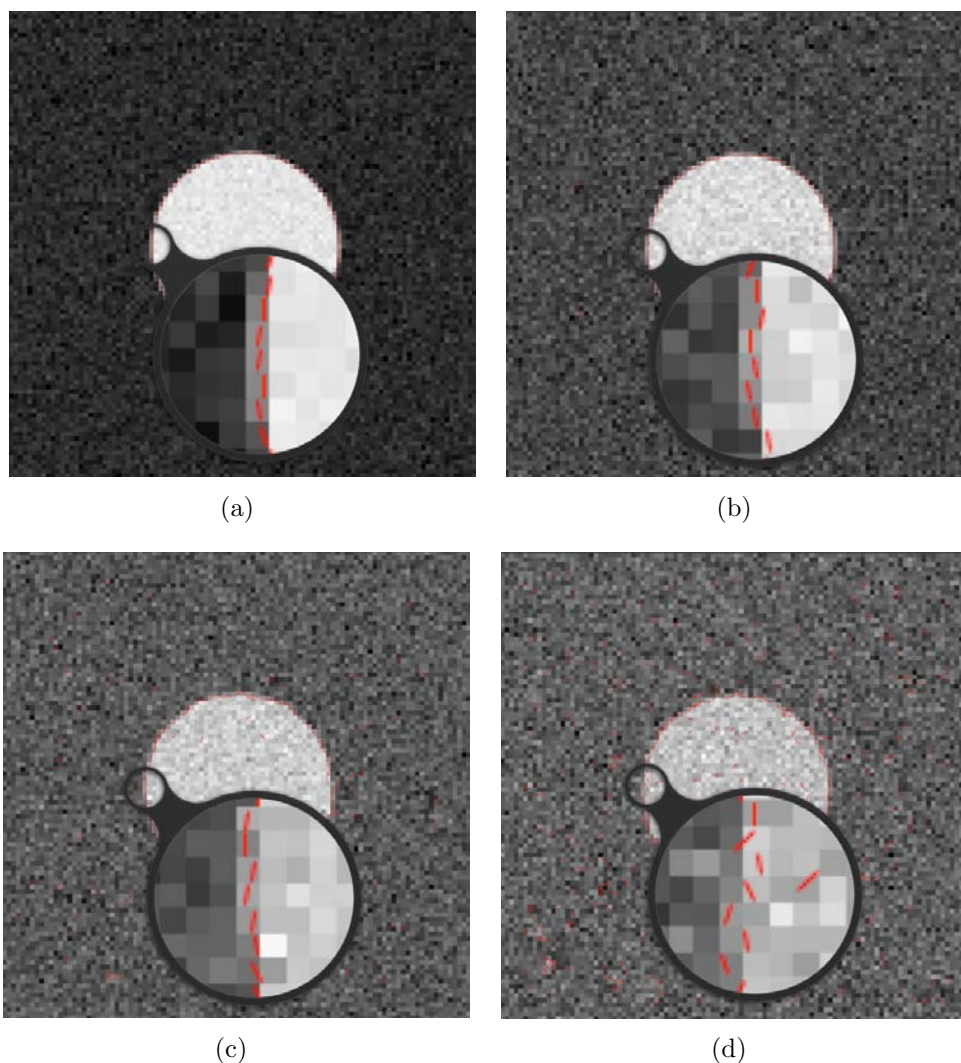


Figura 4.11: Detección algoritmo básico de segundo orden en imágenes con ruido gaussiano: (a) $\text{std}=5$, (b) $\text{std}=10$, (c) $\text{std}=15$ y (d) $\text{std}=20$

En dicha figura se puede ver la detección usando el método descrito en la sección anterior frente a niveles de ruido gaussiano diferentes (con una desviación estándar que va de 5 a 20). Como es de esperar, a un mayor nivel de ruido la detección es peor.

En esta sección se describirá el desarrollo del método explicado en la sección 3.1.2.3. Básicamente consiste en aplicar un suavizado previo a la detección, aunque esto afecta al tamaño de la ventana, que pasa de ser de 5 a 9 píxeles (hablando del caso de ventana vertical -parcial máxima en y -).

Las consideraciones contempladas en el anterior apartado son totalmente válidas en este, en relación con adaptar el tratamiento de la imagen a la clase `InrImage` de `AMILab`. Previamente a realizar la detección, se invoca a la función para aplicar el suavizado, denominada `Average3x3`.

Esta función recibe cinco parámetros. Los dos primeros son de tipo de `InrImage` y corresponden a la imagen de entrada y la de salida, respectivamente. Los tres siguientes son de tipo `double` y están relacionados con la máscara de suavizado que se va a usar (ver ecuación 3.26 y siguientes). Inicialmente se realiza un suavizado de la zona central de la imagen, es decir, dejando un margen de un píxel alrededor de toda la región que se está procesando, ya que es la zona en la que no hay que tener ningún tipo de consideración especial. Posteriormente, se realiza un tratamiento pormenorizado de todas las esquinas y límites ³. Para ello se sigue lo descrito en el apartado *Mejoras* de la sección 3.1.2.4.

En la figura 4.12 se puede observar el flujo de ejecución que sigue el algoritmo. Inicialmente, como ya se ha dicho, se ejecuta el suavizado de la imagen de entrada a través del método descrito. Posteriormente, al igual que en la sección anterior, se entra en un bucle que se encarga de recorrer todos y cada uno de los píxeles de la imagen (dejando un margen). Los primeros pasos son exactamente iguales que en el método anterior, es decir, se calculan las parciales en x y en y , para luego ver cuál de las dos es mayor en valor absoluto. En función del resultado de dicha operación lógica se toma la ventana en vertical o en horizontal. Nótese que en esta ocasión la dimensión de la ventana es superior que en el caso anterior, ya que al haberse realizado el proceso de suavizado para eliminar el ruido, se necesita una ventana mayor para realizar una correcta detección. No obstante, hay que tener en cuenta que las dimensiones no se corresponden totalmente a una estructura rectangular. Esto viene del hecho descrito en la sección 3.1.2.3, y es que en el cálculo de las intensidades se correría el peligro de tomar valores demasiado alejados que podrían dar un resultado erróneo. Es así por tanto que se tiene una estructura escalonada (puede verse a ambos lados de la figura 4.12).

Si sucede que la parcial supera el valor del umbral se realizarán el resto de los cálculos. En caso contrario, se pasará a procesar el siguiente píxel de la imagen. En adelante, el resto de pasos son los mismos que los vistos en la sección anterior: cálculo de las intensidades y comprobación de que su diferencia es menor que el umbral, sumas de las filas/columnas y cálculo de los parámetros del contorno. De esta manera, al terminar se tendrá toda la información relativa a los píxeles borde de la imagen. De igual forma, se hace uso de una estructura vectorial de la STL de C++ del tipo `borderPixel` para almacenar los resultados de la detección.

En relación al *wrapping*, la filosofía es la misma que la mostrada en la explicación del método básico de detección de segundo orden. No obstante, en esta ocasión el `AMIOject` contendrá un elemento más, que será la imagen suavizada (ver figura 4.13). Ésta será la única de tipo bidimensional, ya que el resto de componentes continúan siendo imágenes 1D que contienen los parámetros del contorno: intensidades, tipo de borde, coeficientes de la parábola, curvatura y posición del

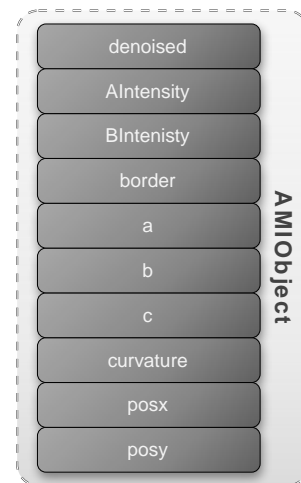


Figura 4.13: Estructura de tipo `AMIOject` que encapsula el resultado de la detección de bordes en imágenes con ruido

³Entiéndase como límite la región lineal (fila o columna) de grosor un píxel situada a la zona izquierda, derecha, superior o inferior de la imagen

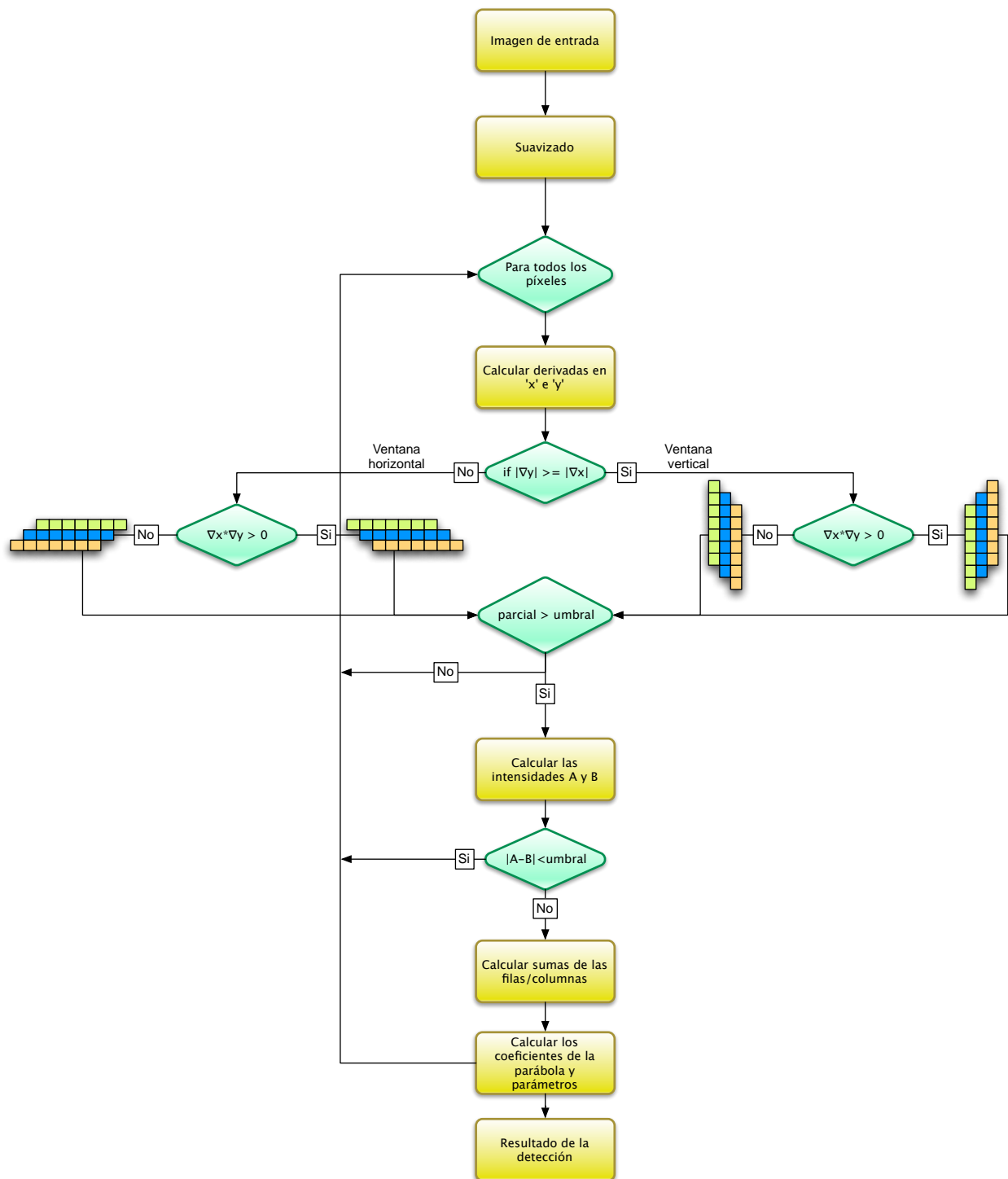


Figura 4.12: Diagrama de flujo del algoritmo de orden 2 para imágenes con ruido

píxel borde en la imagen.

En la figura 4.14 se puede observar el resultado de aplicar este nuevo método a una circunferencia centrada en el píxel central de una imagen de 100×100 píxeles y radio 20, que se ha visto afectada por un ruido gaussiano de desviación estándar 5, 10, 15 y 20 respectivamente.

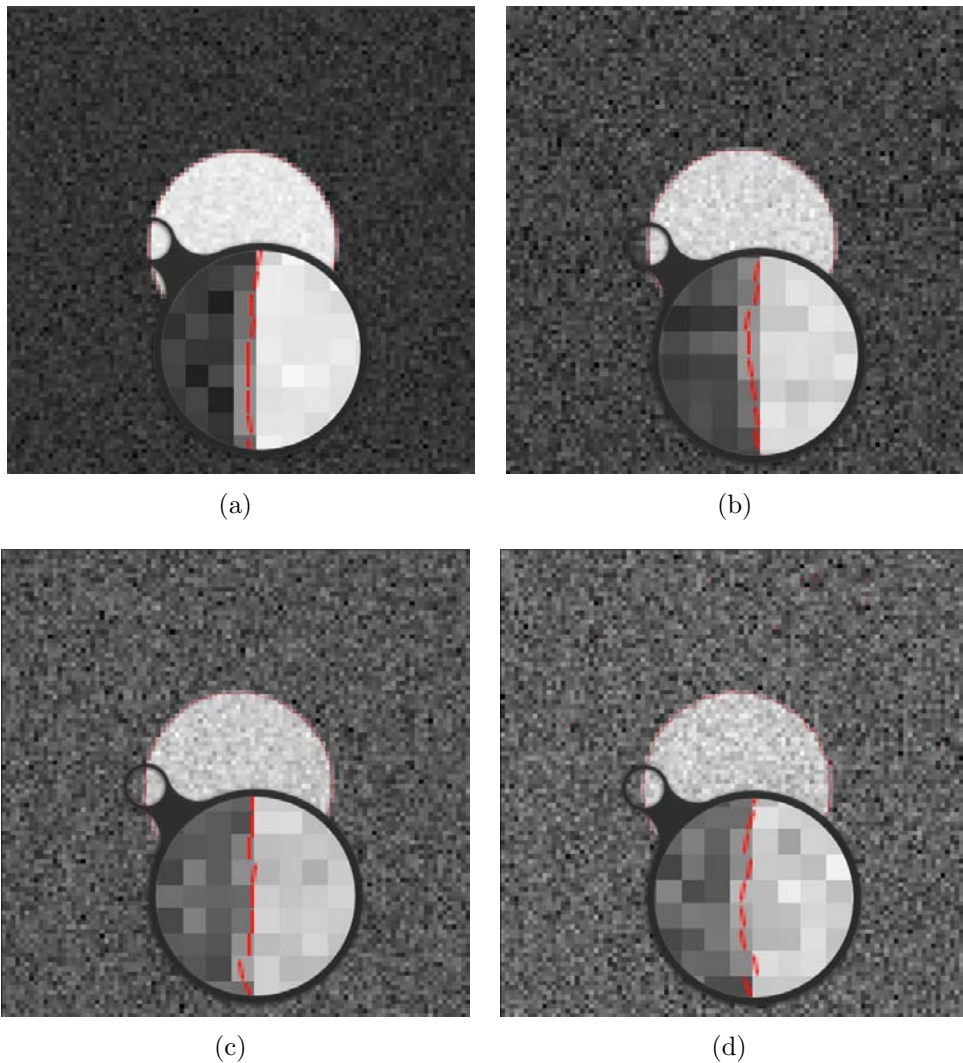


Figura 4.14: Detección algoritmo de segundo orden para imágenes con ruido gaussiano: (a) $\text{std}=5$, (b) $\text{std}=10$, (c) $\text{std}=15$ y (d) $\text{std}=20$

Puede verse como la detección en la imagen afectada por un nivel relativamente bajo de ruido es bastante bueno. Pero, a medida que el nivel de ruido aumenta, la detección se ve afectada tanto en posición como en orientación, así como aparecen detecciones espurias que en principio no tienen nada que ver con el contorno. Por tanto, si la imagen está afectada por un nivel bajo de ruido, el método funciona correctamente. Pero para niveles altos, será preciso implementar un esquema que de un mejor resultado.

Límites Variables

Antes de pasar al método que es capaz de hacer una correcta detección en imágenes afectadas por un nivel superior de ruido, se ha introducido una pequeña mejora en esta sección. Esta mejora es la descrita en la sección 3.1.2.4, concretamente en el apartado de *Mejoras*.

Hasta el momento, los métodos desarrollados sólo consideraban la existencia de un único

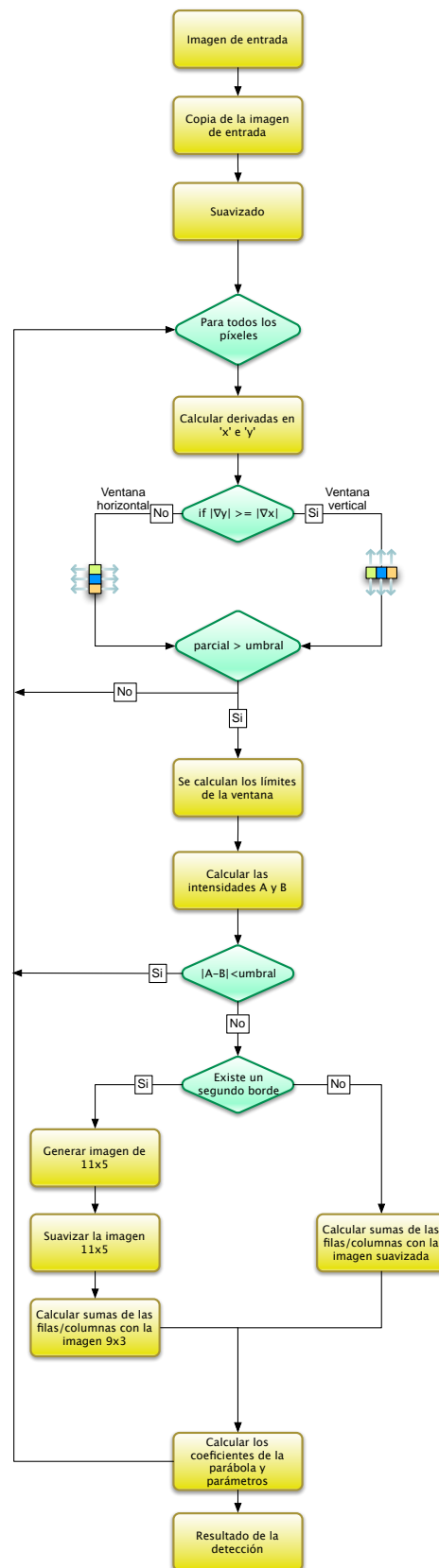


Figura 4.15: Diagrama de flujo del algoritmo para imágenes con ruido. Tratamiento de bordes cercanos y muy cercanos

borde en el interior de la ventana usada para realizar los cálculos. Pero ¿cómo habría que actuar en el caso de que dentro de la misma hubiese más de uno? Esto puede verse en el diagrama de flujo de la figura 4.15. Inicialmente es preciso hacer una copia de la imagen de entrada, ya que es necesaria posteriormente para poder procesar los bordes cercanos. Una vez hecho esto, al igual que en el diagrama anterior, se procede a hacer un suavizado de la imagen de entrada. Con ella se hará el procesado de cada píxel. Como en casos anteriores, se realiza el cálculo de las derivadas parciales, mirando cual de las dos es mayor en valor absoluto. Además ha de superarse el valor de umbral para continuar con el proceso. La parcial de mayor valor será lo que determine la orientación, sin embargo, en esta ocasión la ventana no tiene aún unas dimensiones concretas. Como se describió en la sección citada anteriormente, se hará un cálculo para los límites de cada fila o columna de manera independiente, siendo esto lo que determine sus dimensiones. Con esto queda reflejado el caso en el que existan bordes cercanos. Entonces se pasa a calcular los valores de intensidad a ambos lados y se verifica que el salto sea mayor al umbral. En caso contrario se pasaría a realizar los cálculos con el siguiente píxel.

En este instante se pasa a estudiar si existe un segundo borde muy cercano (segundo caso que se quería tratar). En caso de no existir un segundo borde cercano, se pasa directamente a realizar las sumas de las filas/columnas usando la imagen suavizada, para después calcular los coeficientes de la parábola y los parámetros del contorno, antes de pasar a procesar el siguiente píxel. Por contra, si sucede que se encuentra ese segundo borde, se procede de la siguiente manera: se genera una imagen sintética de tamaño 11 que es rellenada usando los valores de la imagen original (antes de ser suavizada, de ahí que precisáramos la copia) y en su parte superior e inferior con los valores de B y A respectivamente, en función de si la existencia del borde es superior o inferior. Una vez construida esa imagen, se procede a realizar un suavizado. La justificación de tener que realizarlo se explicó en la sección citada, pero simplemente se recuerda que es porque las ecuaciones que se usan para estimar los coeficientes se basa en que se está procesando una imagen suavizada. Por tanto, si se aplicase sin hacer este paso, el resultado no sería correcto. Nótese que en el suavizado no se procesan los límites inferior, superior, izquierdo y derecho de la sub-imagen sintética, puesto que sólo será necesario usar su parte interior, es decir, lo que corresponde a una ventana de 9×3 (no obstante recuérdese que los límites de la ventana en este caso no son fijos y vienen determinados por el cómputo hecho anteriormente). Usando esa parte interna de la sub-imagen se calculan las sumas de las filas/columnas y se obtienen los coeficientes de la parábola y los parámetros del contorno. Con ello, al igual que en el otro caso, ya se podría pasar a procesar el siguiente píxel.

Se continúa con la misma filosofía en relación a cómo guardar la información relativa a la detección de los píxeles borde, de manera que se hace uso de un vector de la STL de C++ de tipo `borderPixel` para guardarla.

En cuanto al mecanismo de *wrapping* o encapsulado de código, es exactamente el mismo que el comentado anteriormente, teniéndose un objeto del tipo `AMIObject` igual al mostrado en la figura 4.13.

Si observamos ahora la figura 4.16 se puede ver como actúa el detector ante una imagen sintética con bordes cercanos afectada por diferentes niveles de ruido. La imagen en cuestión tiene unas dimensiones de 100×100 píxeles y está formada por un anillo centrada en el píxel central, que tiene un radio de 30 y un grosor de 2 píxeles. Se puede ver que ante un ruido

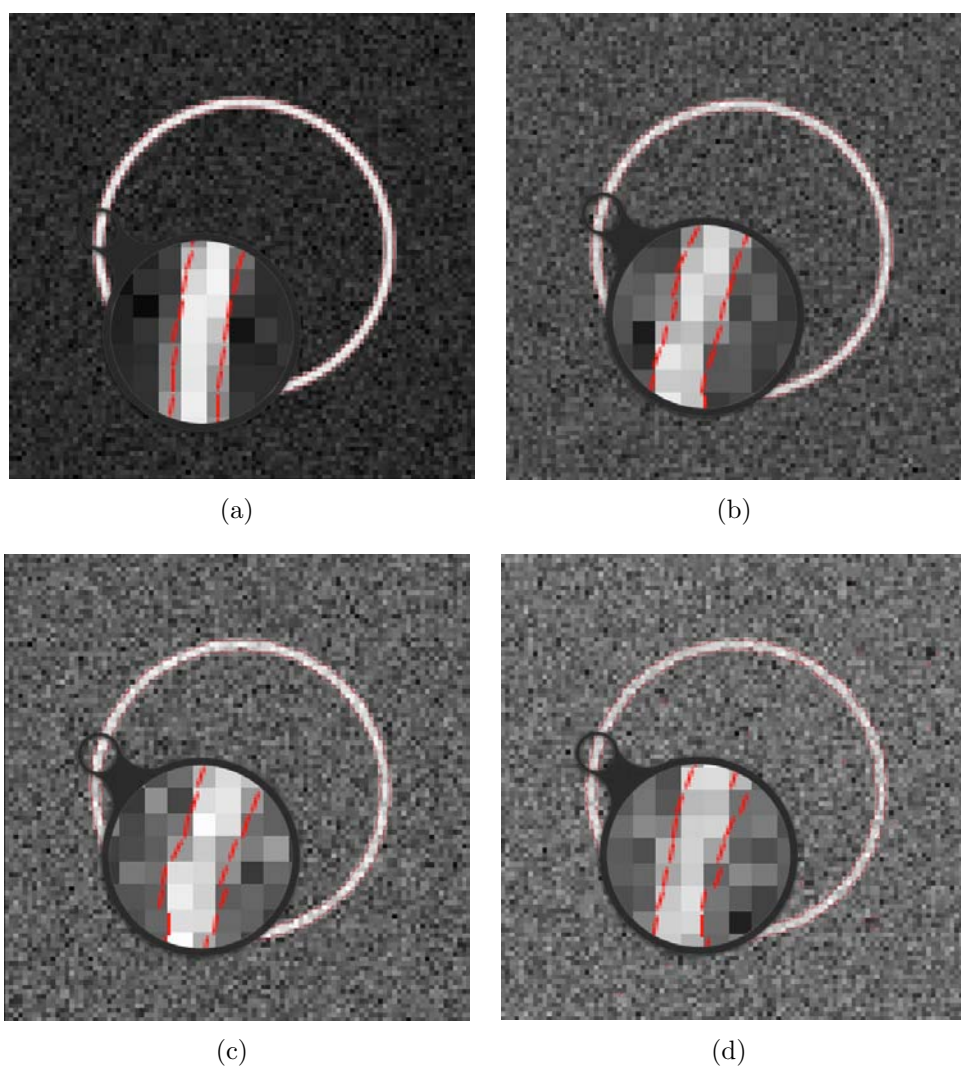


Figura 4.16: Detección algoritmo de segundo orden para imágenes con ruido gaussiano: (a) $\text{std}=5$, (b) $\text{std}=10$, (c) $\text{std}=15$ y (d) $\text{std}=20$

de desviación estándar 5, el algoritmo se comporta de una forma similar al caso que se había estudiado con anterioridad (el caso de una circunferencia a la que se le había añadido ruido). Los bordes cercanos son detectados con bastante precisión. Es necesario recordar que la separación mínima debe ser al menos de dos píxeles, pues si fuese de uno solo no se tendría una de las dos intensidades que se encuentran a ambos lados del borde y, por tanto, el resultado del método no sería correcto.

De igual forma que sucedía anteriormente, al aumentar el nivel de ruido, la detección no es precisa, habiendo errores en posición y orientación. Por tanto, se tiene un método capaz de detectar bordes muy cercanos en imágenes que se encuentran afectadas ligeramente por ruido. Para lograr detectar los bordes con niveles de ruido superior, será necesario desarrollar otro método, que será el que se describe en la siguiente sección.

Es necesario tener en cuenta que, aunque estas mejoras se encuentran descritas respecto al

método de restauración, se han incorporado en la fase de desarrollo en este detector. Además, la incorporación de las mejoras viene justificada por el uso del este algoritmo concreto en la participación en la conferencia EUROCAST 2011. La descripción de ésta, se encuentra en la sección 5.3.

4.1.2.3. Método Sub-Píxel de Restauración

Como se ha descrito en la sección anterior, cuando una imagen se encuentra afectada por altos niveles de ruido, el método que consiste en suavizar la imagen previamente a la detección, no es capaz de encontrar los contornos con precisión (ver fig. 4.16). Por tanto, es necesario que se realice un desarrollo que permita que se puedan detectar los bordes cuando el nivel de ruido es más alto. Por ahora, al menos, al método se le ha incorporado la capacidad de detectar bordes muy cercanos, mediante el uso de límites variables en la ventana que se usa para los cálculos (también denominada ventana flotante en [42]) y el estudio de los valores de las parciales más allá del límite de la ventana, para estimar la posible existencia de un segundo borde muy cercano.

El esquema propuesto y descrito en la sección 3.1.2.4, plantea el uso de un esquema iterativo. Dicho de una manera simple, este método en cada iteración realiza un suavizado de la imagen, a la vez que se van reconstruyendo las zonas en las que se encuentran bordes. De esta manera se elimina el ruido y se intenta lograr preservar los contornos que se detecten.

Se han tomado las mismas consideraciones que en la adaptación de los métodos anteriores, esto es, aplicar el uso de las estructuras de datos propias de **AMILab** como el tipo usado para manejar las imágenes, **InrImage**. En esta ocasión además el método debe tratar algunos aspectos más, pero veámoslo usando su diagrama de flujo. Este diagrama puede verse en la figura 4.17.

Inicialmente, a partir de las dimensiones de la imagen de entrada, se crean tres nuevas imágenes: contadores, intensidades y la imagen suavizada. La primera de ellas va a tener los valores acumulados relativos al número de veces que un píxel ha sido actualizado en la imagen. La segunda será la que contenga los valores de intensidad de cada píxel. La última de ellas es donde se almacena el resultado de realizar el suavizado de la imagen de entrada. Estas tres imágenes se mantendrán a lo largo de las iteraciones y serán actualizadas de forma adecuada.

En ese momento comienzan las iteraciones. Es preciso recordar que esta estructura repetitiva no es la que se encarga de recorrer los píxeles, sino la que controla el número de veces sucesivas que se aplica el método sobre la imagen de entrada. Al principio de éste se inicializan a cero la imagen de contadores y de intensidades, ya que éstas serán calculadas en cada iteración. Acto seguido se hará una copia de la imagen de entrada, ya que como se describió en el anterior método, esto es necesario de cara a poder detectar bordes muy cercanos. De esta manera, se tendrá en cada iteración: la imagen de contadores, la de intensidades, la imagen suavizada i (actual) y la imagen $i - 1$ (de la iteración anterior). Con la copia hecha, ya se puede proceder a realizar el suavizado. Además de ello, en esta etapa se realiza un vaciado del vector de píxeles borde a través del método `clear()`. Esto es así porque lo que realmente interesa es quedarse con la última detección que se haga, con la final.

Superada esta fase comienza la estructura repetitiva anidada que recorre todos los píxeles de la imagen. Es totalmente igual a lo descrito en el método anterior, dado que ya se introdujeron las mejoras en la detección de bordes cercanos (ventana con límites variables, ventana dinámica o ventana flotante) y de bordes muy cercanos (estudio de los valores de las parciales más allá de los límites de la ventana). Ya que este bloque es el mismo, se obviará su descripción con el fin de no ser repetitivos.

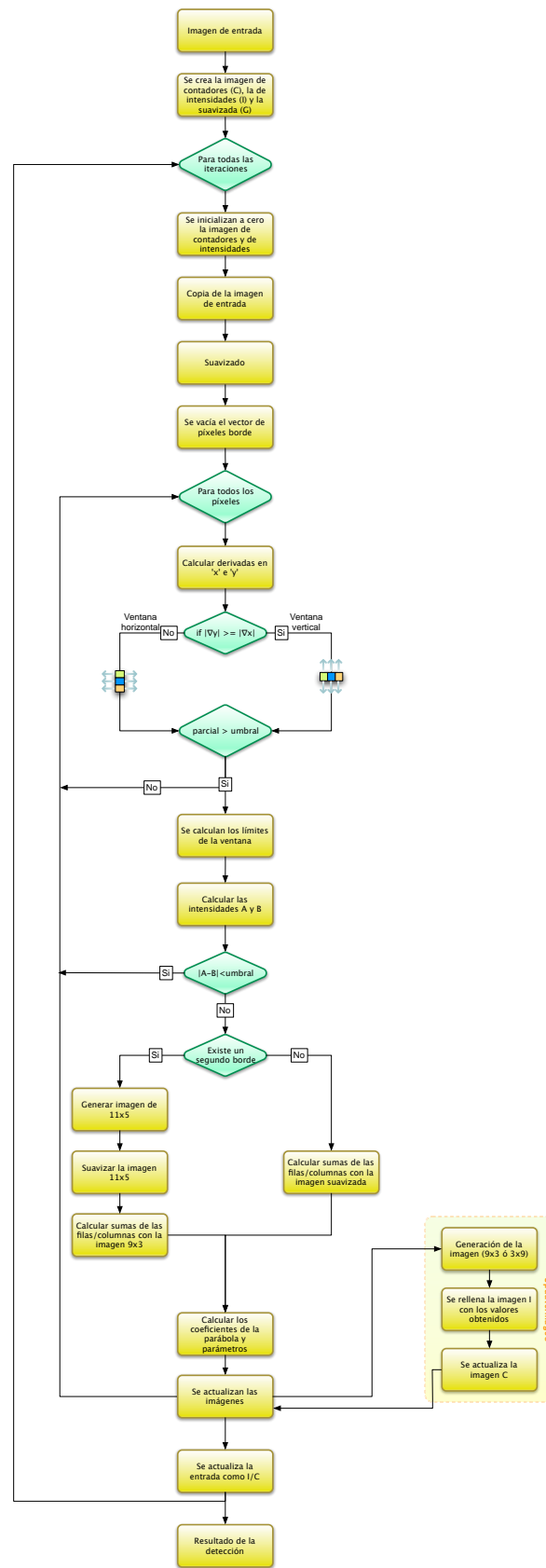


Figura 4.17: Diagrama de flujo del algoritmo de restauración

Una vez se tienen los valores de las intensidades A y B , así como la sumas de las filas/columnas; se pasa a obtener los coeficientes de la parábola, así como el resto de parámetros del contorno. Con todo ello, se pasa a actualizar las imágenes de intensidades y contadores. Para ello se hace uso de la función interna `UpdateImages`, que aparece como un recuadro amarillo en la zona inferior derecha de la figura 4.17. Esta operación se divide básicamente en tres pasos. El primero de ellos es el que se ocupa de generar la imagen sintética de 9×3 ó 3×9 , dependiendo de la orientación del borde. Dicho de una manera sencilla, lo que hace es aplicar el principio del método pero a la inversa. Es decir, si el método lo que hace es tratar de hallar los coeficientes de la parábola que mejor se ajustan al contorno, en esta función lo que se hace es que, a partir de los valores de dichos coeficientes, se generan los valores de intensidad de una pequeña imagen sintética. Con dichos valores se actualizan la imagen de intensidades, asignando en la posición adecuada el valor de intensidad correspondiente. La imagen de contadores simplemente incrementa los valores de la región que constituiría aquella que se está procesando y que corresponde a la pequeña sub-imagen. Dicha actualización se hace teniendo en cuenta el principio propuesto en la sección 3.1.2.4, que no es otro que darle más peso a la fila/columna central, que es la que contiene el píxel sobre el que se está calculando el contorno.

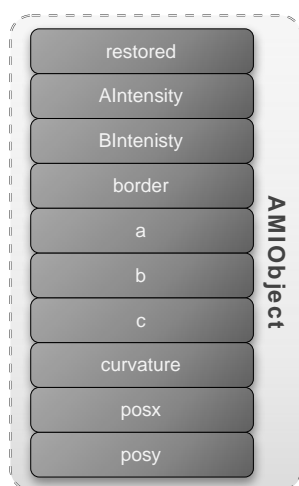


Figura 4.18: Estructura de tipo `AMIObject` que encapsula el resultado de la detección de bordes en imágenes donde se aplica restauración

Después de ello, es necesario actualizar la imagen para la siguiente iteración. Dicha actualización se realiza usando las imágenes de intensidades y de contadores. Ya que la primera indica el valor de intensidad correspondiente y la segunda el número de veces que se ha actualizado dicho valor (siempre en función al convenio de convergencia descrito), la actualización será sencilla. Sólo será preciso ver en qué posiciones de la imagen de contadores (C) se encuentran valores superiores a cero. En ese caso, se actualiza dicha posición con el cociente entre el valor de la intensidad y el valor del contador (I/C). En caso contrario se deja el valor tal cual. De esa forma, las regiones en las que se encuentran bordes, serán actualizadas a través de este proceso y aquellas en las que no, se irán suavizando en sucesivas iteraciones.

En relación al proceso de *wrapping* o encapsulado de código, se procede de manera similar que en anteriores métodos. Sin embargo la estructura `AMIObject` difiere ligeramente. En la figura 4.18 se puede observar que el único cambio es que en lugar de almacenarse la imagen resultado del suavizado, se almacena la resultante de la restauración. El resto de valores devueltos siguen siendo los mismos, introducidos en el interior de imágenes unidimensionales.

Si aplicamos nuevamente ahora este método al conjunto de imágenes mostradas anteriormente, formadas por un anillo afectado por diversos niveles de ruido, se deberían obtener mejores resultados.

En la figura 4.19 se puede ver el resultado de aplicar el algoritmo de restauración a una imagen de 100×100 píxeles, con un anillo centrado en su píxel central, de radio 30 y de 2 píxeles

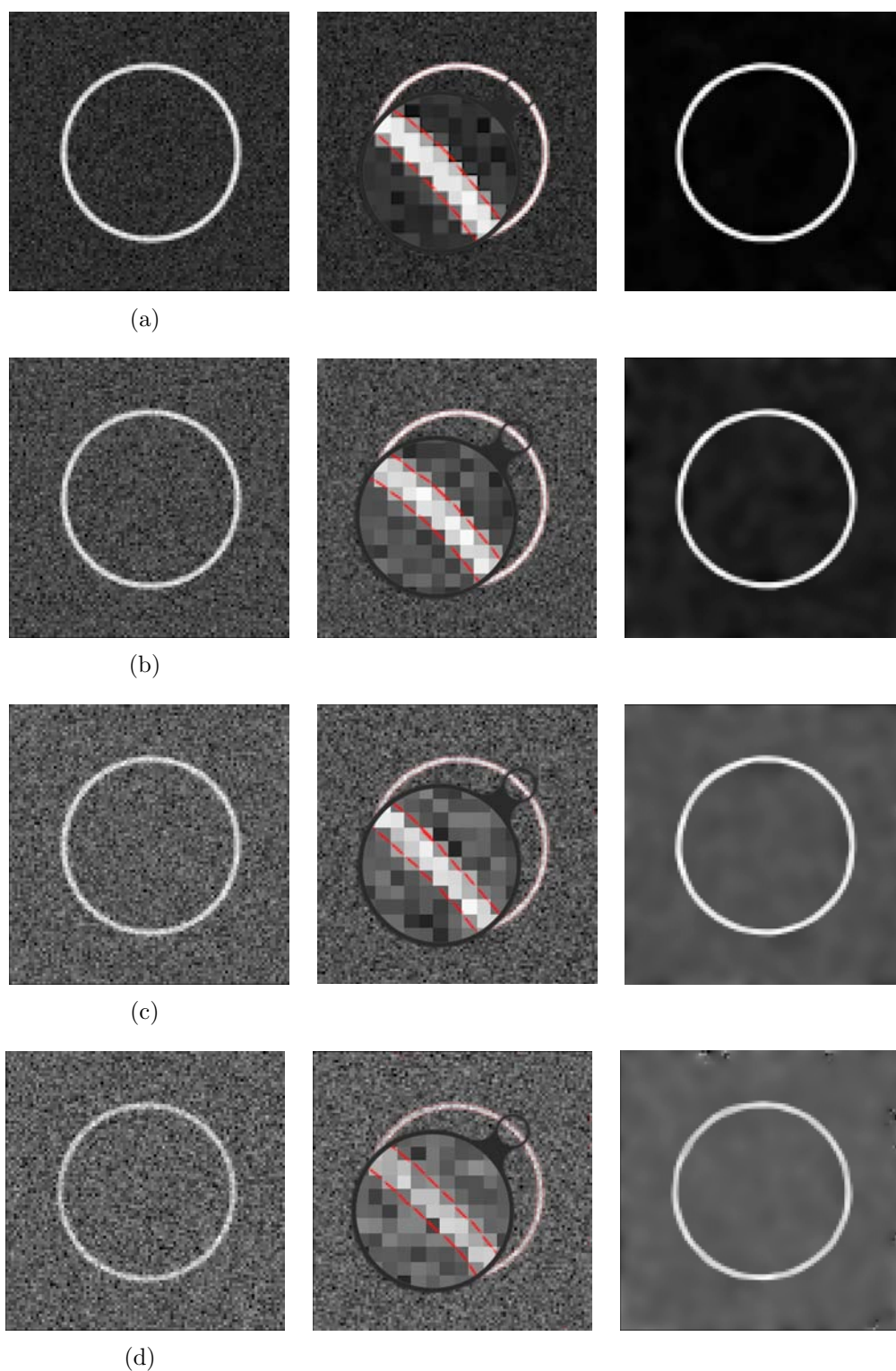


Figura 4.19: Detección algoritmo de restauración para imágenes con ruido gaussiano: (a) $\text{std}=5$, (b) $\text{std}=10$, (c) $\text{std}=15$ y (d) $\text{std}=20$

de grosor. De izquierda a derecha, las columnas representan la imagen original, la detección

realizada y la imagen restaurada resultante del proceso iterativo. Se observa que en casi todos los casos la detección es bastante buena, tanto en orientación como en posición sub-píxel del contorno. Es más, si lo comparamos con el método anterior, es bastante mejor, ya que tiene un mejor comportamiento ante niveles de ruido altos, cosa que no sucedía anteriormente, ya que se carecía de la estructura repetitiva de suavizado.

Existen más extensiones de estos métodos, como la detección para altas curvaturas, pero que no se han incluido en este trabajo.

4.1.2.4. *Script* en AMILab

El mecanismo de *wrapping* es aquel que permite poner a disposición del lenguaje de *scripts* de AMILab toda la potencia de los métodos implementados en C++, pero además, en él se realiza el desarrollo de la interfaz que el usuario podrá manejar para interactuar con el método.

De manera ideal, un *script* en AMILab debe tener su propia carpeta. En ella se deben incluir, como mínimo, el fichero con la clase, el que carga la interfaz y una breve documentación del método realizada en \LaTeX y convertida con \LaTeX2html . Esto último es así, ya que el lenguaje de *scripts* incorpora un pequeño visor de `html`, permitiendo navegar a través de la documentación que se haya generado.

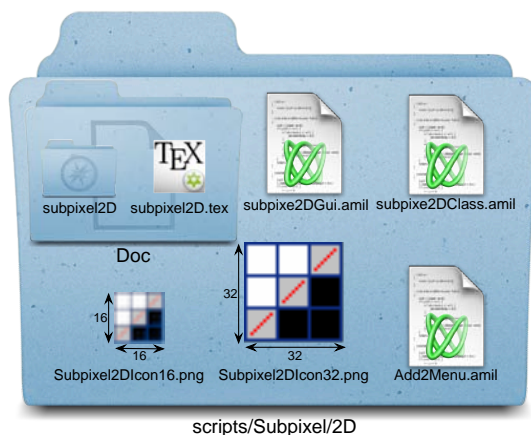


Figura 4.20: Estructura del directorio 2D del *script* para la detección con precisión sub-píxel

En la figura 4.20 se puede ver la estructura que se encuentra dentro del directorio que contiene el *script* para la detección de contornos con precisión sub-píxel. En ella puede observarse que dentro del directorio se encuentra una carpeta denominada “Doc”. En su interior se encuentra el fichero \LaTeX con el nombre `subpixel2D.tex`, así como el directorio del mismo nombre. Éste se ha obtenido de aplicar \LaTeX2html al archivo \LaTeX . Además, están los *scripts* de la clase y que cargan la interfaz (`subpixel2DClass.amil` y `subpixel2DGui.amil` respectivamente), el *script* que añade la funcionalidad al menú y a la barra de herramientas (`Add2Menu.amil`),

así como los iconos 16×16 y 32×32 que aparecerán en el menú y en la barra de herramientas respectivamente.

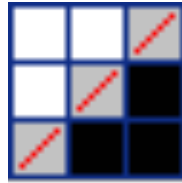


Figura 4.21: Diseño del icono que aparece en la barra de menú y la barra de herramientas de AMILab para el algoritmo de detección con precisión sub-píxel

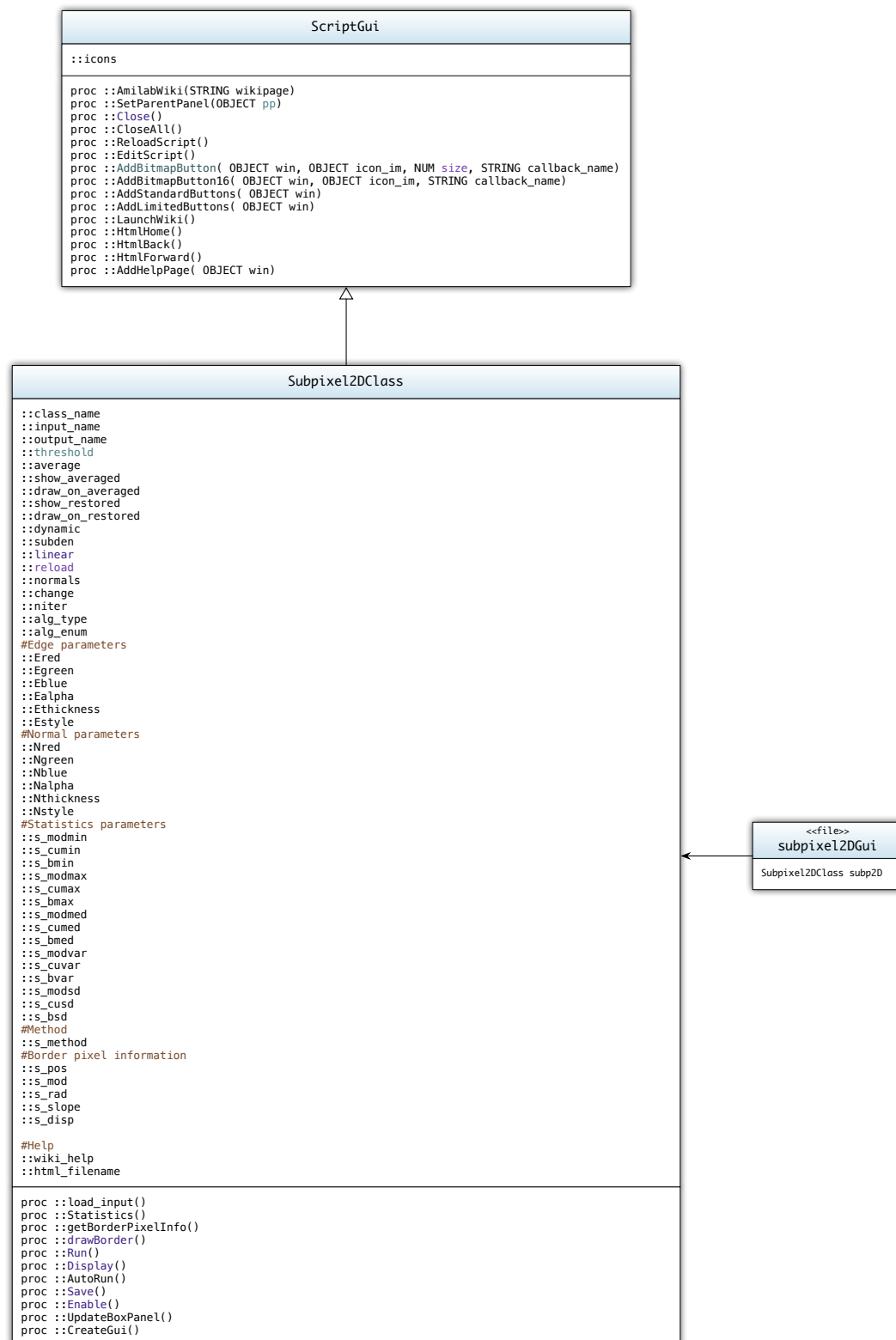
En el diseño del icono (fig. 4.21) se ha partido de la imagen de una rejilla presente en la colección libre que se encuentra dentro del directorio de *scripts*. Se ha intentado reflejar el efecto parcial de pasar de una zona con una intensidad a otra región que posee otros valores, con una transición intermedia. Es dentro de la misma donde se ha dibujado la correspondiente detección sub-píxel.

Anteriormente, cuando se habló del método básico de detección a nivel píxel, se describió que el *script* que se había desarrollado para su manejo no hacía uso de la herencia. El uso de esta herencia es bastante útil, ya que aporta algunas funcionalidades básicas de interfaz, como son las opciones de editar, recargar, cerrar y cerrar todo. Normalmente, lo que se hacía con anterioridad era implementar en cada *script* las funcionalidades de cerrar y cerrar todo. Pero, ahora, gracias al concepto de herencia, sólo es necesario heredar de la clase `ScriptGui` e incluir al final del método que genere nuestra interfaz la operación `::AddStandardButtons(&::win)`, donde `&::win` es la referencia al panel actual de nuestra interfaz. Esta función será la encargada de cargar los botones indicados anteriormente. Además, se añade la opción de poder insertar botones con imágenes (*bitmap buttons*), a través del método `::AddBitmapButton(&::win, &::icon, size, "function")`. Los parámetros que recibe son: `&::win` la referencia al panel actual de la interfaz, `&::icon` la referencia de la imagen que se quiere poner en el botón ⁴, `size` que es el tamaño que se desea para la imagen del botón y `"function"` que es el nombre de la función entre comillas, y que se va a invocar cuando se haga clic.

En la figura 4.22 se puede ver la estructura general del *script* desarrollado para los métodos de detección sub-píxel. El *script* desarrollado está constituido principalmente en la clase `Subpixel2DClass` que hereda de la clase `ScriptGui` las características generales de interfaz y la capacidad de poder añadir botones con imágenes, como se ha descrito anteriormente. El fichero `subpixel2DGui` hace uso de la clase `Subpixel2DClass`. Básicamente crea un objeto del tipo de esta clase e invoca al método de la clase que crea la interfaz (`CreateGui()`).

La clase `Subpixel2DClass` tiene un gran número de atributos. Los primeros de ellos son generales y están asociados con elementos de visualización y de la propia interfaz, como son la variable para el valor del umbral, si se muestra o no la imagen suavizada o la restaurada,

⁴Si no se ha cargado por defecto la imagen para el botón, se puede cargar a través de la operación `::icons.LoadIconPNG("Name")` a la que se le pasa el nombre de la imagen que se quiere cargar y que está presente en el directorio `scripts/Icons/png/32 × 32`

Figura 4.22: Diagrama de clases del *script* para detectores de bordes a nivel sub-píxel

etc. Luego se pasa a ver una serie de variables de configuración, dirigidas al control de las características del borde y las normales (color, transparencia, grosor y tipo de línea). Otro grupo de atributos están dirigidos a almacenar el resultado del cálculo de estadísticas acerca de la detección, funcionalidad la cual será descrita más en detalle posteriormente. Por último se tiene una variable para el método seleccionado, un conjunto de ellas para la información del píxel borde (posición, salto de intensidad, radio, pendiente y desplazamiento). Los últimos dos atributos están relacionados con la ayuda desarrollada para el uso del método.

A continuación se pasan a describir brevemente cada uno de los métodos miembro. Simplemente decir que todos son procedimientos, lo cual se denota en el lenguaje de *scripts* de AMILab con la palabra reservada `proc`:

- `load_input()`: este método es el encargado de cargar la imagen de entrada. Si es invocado y existe ya la variable que contiene dicha imagen, se borra ésta y se vuelve a cargar. Actúa de la misma forma si detecta la existencia de la imagen suavizada o de la restaurada.
- `Statistics()`: se ocupa de calcular estadísticas acerca de la detección (mínimo, máximo, media y varianza). Los parámetros sobre los que calcula las estadísticas son: el salto de intensidad, la curvatura del radio y la pendiente. El cálculo de mínimos y máximos es inmediato a través de las funciones `min(expr_image)` y `max(expr_image)` donde `expr_image` es cualquier expresión que devuelva una imagen o una imagen en sí. De la misma forma, se calcula la media usando la operación `mean(expr_image)`. Para el cálculo de la varianza se computan los cuadrados de las diferencias de las imágenes con sus medias calculadas previamente. Finalmente, el cálculo de la desviación estándar no es más que la raíz cuadrada de la varianza.
- `getBorderPixelInfo()`: este método muestra información acerca de un píxel seleccionado en la interfaz del visor. El visor de AMILab permite capturar la posición sobre la que se ha hecho clic y, de esta manera, mostrar cualquier información que pueda asociarse a dicha localización. En primer lugar comprueba que existan tanto la variable del visor como la del resultado de aplicar el método. En caso afirmativo, se capturan la posición x e y del visor a través de los métodos `GetXPos()` y `GetYPos()` respectivamente. En ese momento se va a buscar la posición en la imagen. Para ello se hace uso del objeto de tipo `AMIObject` que se ha devuelto de la aplicación del método (para mayor detalle ver figuras 4.8, 4.13 y 4.18). Se inicializa una variable a un valor mayor que el tamaño de las imágenes unidimensionales que se han creado para los parámetros de los píxeles borde. Entonces se comienzan a recorrer las imágenes que representan las posiciones en x e y y se intenta ver si esta coincide con los valores capturados de la posición del cursor en el visor. En caso afirmativo, se captura la susodicha posición dentro de las imágenes y se abandona el bucle a través de una operación de `break`. La forma de saber si se ha localizado algo dentro de dichas imágenes es mirar si la variable a tomado otro valor diferente al que se le asignó antes del bucle previo. En caso de haber hallado la posición, se muestran en la interfaz la posición de píxel, el tipo de borde (horizontal o vertical), el salto de intensidad, el radio, la pendiente y el desplazamiento. Si no se ha capturado posición alguna, es decir, si el píxel que se ha seleccionado en el visor no es un píxel borde, se muestra solamente la posición del mismo y se indica que no es un píxel de ese tipo a través del texto *no edge*.

Además de lo anterior, si no existiesen la variable del visor y el resultado se muestra una

ventana de diálogo que indica que se debe aplicar previamente el método.

- `drawBorder()`: anteriormente se habló sobre el dibujado de la detección sub-píxel, comentando que se había llegado a una solución de compromiso entre el lenguaje de *scripts* y C++, ya que la primera solución desarrollada en el *script* se volvía lenta al tratarse de imágenes de una resolución mayor. Es por ello que ahora se describe aquí tanto lo correspondiente al *script*, así como lo concerniente a la parte de C++.

Se va a comenzar por describir la parte que se encuentra desarrollada en el lenguaje de *scripts*. En primer lugar es preciso tener en cuenta que este método siempre debe ser llamado después de haber invocado a la operación `Display()`, ya que no tiene sentido dibujar los bordes si no se está mostrando la imagen. Inicialmente se tiene en cuenta que existan tanto el resultado de haber aplicado el método como el objeto de la clase `SubPixel2D`. Es preciso comprobar si se ha decidido dibujar en alguna de las otras imágenes, ya que con el método para imágenes afectadas por bajo nivel de ruido se deja dibujar sobre la imagen suavizada; así como en el método de restauración, sobre la restaurada. Con ello se tendrá cuál es la variable del visor que se va a usar para poder hacer el dibujado.

Partiendo de dicha referencia al visor, se obtiene la región de visualización, a través de las operaciones `GetXmin()`, `GetXmax()`, `GetYmin()` y `GetYmax()`. Esto es así porque no siempre la región de visualización corresponde a toda la imagen, sino que se puede haber hecho un *zoom*, haciendo que la región sea más pequeña (relativa a la imagen completa original).

Después de obtener los límites de la visualización, se crean cuatro imágenes nuevas para los desplazamientos en el interior del píxel. La dimensión de las mismas se obtiene a partir de una de las imágenes obtenidas como resultado. Serán, por tanto, imágenes unidimensionales. Considerándose que el origen de coordenadas se encuentra situado en la zona central del interior de cada píxel y que la dimensión en cada sentido es 1 en valor absoluto (1 y -1 en x e igual en y), se dibujará desde la posición $-0,4$ a la $0,4$, de forma que el contorno quede inscrito dentro del píxel. No obstante, debido a que el desplazamiento desde la zona central puede ser mayor, así como el tipo de orientación; en ocasiones podría dar la sensación de que el dibujo del contorno sale fuera del píxel, aunque su posición es totalmente correcta. Según lo dicho, por tanto, dos de las imágenes creadas tendrán los valores $-0,4$ y $0,4$ respectivamente. Las otras dos tendrán el resultado de calcular la ecuación de la parábola $y = a + bx + cx^2$, a partir de la variable que contiene el resultado de la detección. Gracias a que `AMILab` tiene operadores a nivel de imagen, estos cálculos se pueden hacer en un sola línea, aunque realmente se estará haciendo a bajo nivel en cada posición de la misma. Esto simplifica enormemente el código, de ahí que se haya decidido hacer algunas partes desde el lenguaje de *scripts* y otras no.

Lo siguiente a obtener son los puntos a partir de los cuáles pintar. Es decir, nuestro contorno local al interior del píxel irá desde un punto (x_0, y_0) a otro (x_1, y_1) . De la misma forma que se ha actuado antes, esto se hace a nivel de imágenes. Pero, es preciso tener en cuenta que, dependiendo de la orientación del contorno, el desplazamiento ha de ser usado sobre el eje x o sobre el eje y . Para ello se usa una sencilla operación lógica que permite que en cada punto de la imagen se tenga la posición correcta sobre el eje correcto. Esto se consigue multiplicando el desplazamiento por el tipo de borde que se tiene en cada punto, sumándole la otra opción. Es decir, se comprueba si el borde es de tipo vertical, mediante una operación lógica que devolverá un valor cero o uno en cada punto, multiplicando dicho

resultado por el valor del desplazamiento anteriormente calculado. De la misma forma se actúa comprobando si el contorno es horizontal y sumándolo al resultado anterior. De esta manera en el resultado se tendrá en cada punto el desplazamiento correcto, ya que en cada punto donde no se ha cumplido la operación lógica, el valor se habrá anulado gracias al cero que ha devuelto dicha operación; quedando solamente el que interesaba.

Con ello se tienen los valores de dónde a dónde pintar dentro de cada píxel borde, pero como se ha dicho antes, muchas veces no se tiene que dibujar sobre toda la imagen, sino que será preciso solamente sobre la región que se esté visualizando en ese momento. Es por ello que se crea una imagen que delimita la zona de dibujado a la región que se ha obtenido antes a partir de los métodos `GetXmin()`, `GetXmax()`, `GetYmin()` y `GetYmax()`. Será una imagen binaria, cuyos valores son dependientes del producto de todas las posiciones (x, y) del resultado de la aplicación del método comparadas con los límites.

Con lo obtenido, se crea una imagen de vector unidimensional con cuatro componentes. Éstas serán los puntos (x_0, y_0) y (x_1, y_1) , para poder dibujar.

A continuación se comprueba si se ha decidido dibujar las normales. Para ello se mira si se encuentra activo el *checkbox* que lo indica. El cálculo será realizado en función de los puntos que se han calculado antes, ya que no es necesario calcularlo para todos los de la imagen a no ser que sea necesario. Por tanto, sólo se calcularán aquellos de la zona que actualmente se esté visualizando. En primer lugar se obtiene el punto central del borde que se está calculando, que no será otra que la suma de las componentes en x divididas por dos, al igual que las de las componentes y (nótese que se está hablando punto a punto, no en global).

La obtención del punto extremo de la normal se logra a través de las ecuaciones descritas en la sección 3.1.2. Al igual que antes, se hace uso de operadores lógicos para obtener la orientación correcta de la normal en función de la orientación del contorno, para posteriormente crear una imagen de vector unidimensional que contiene de qué punto a qué punto pintarlas.

Por defecto, se establece que el color estándar para el borde será rojo, con un grosor de 1 y un estilo continuo. De igual forma sucede con las normales. Lo que faltaría por hacer es llamar con los parámetros adecuados a la función *wrapped* que se encarga del dibujado. Es necesario tener en cuenta si se ha decidido dibujar o no las normales, así como si se han cambiado los parámetros por defecto. De forma general, a dicha operación se le invocará como `SubPixel2D_object->drawBorder(&viewer, inside, border_pts, border_color, border_thickness, border_style, normal_checkbox)` en el caso de pintar sólo los contornos, y como `SubPixel2D_object->drawBorder(&viewer, inside, border_pts, border_color, border_thickness, border_style, normal_checkbox, normal_pts, normal_color, normal_thickness, normal_style)`. Independientemente, cada parámetro es:

- `&viewer`: es la referencia a la variable que representa al visor.
- `inside`: imagen que indica qué conjunto de los puntos resultado de la detección están dentro de la región que se está visualizando.
- `border_pts`: imagen de vector unidimensional de cuatro componentes, que son los puntos (x_0, y_0) y (x_1, y_1) entre los que pintar.
- `border_color`: color del borde.

- `border_thickness`: grosor del borde.
- `border_style`: estilo del borde.
- `normal_checkbox`: *checkbox* que indica si se van a dibujar o no las normales.
- `normal_pts`: imagen de vector unidimensional de cuatro componentes, que son los puntos (nx_0, ny_0) y (nx_1, ny_1) entre los que se va a pintar.
- `normal_color`: color de la normal.
- `normal_thickness`: grosor de la normal.
- `normal_style`: estilo de la normal.

Tras ésto se pueden plantear dos cuestiones. La primera de ellas es cómo podemos llamar a ese método con un número variable de argumentos. Y la última, pero no por ello menos importante, por qué se ha decidido hacer esa operación en C++ y no en el lenguaje de *scripts*.

En respuesta a la primera de ellas, no se hace uso de ninguna librería para poder llamar a una función con un conjunto variable de parámetros (como por ejemplo la librería `varargs.h` de C). Simplemente lo que se hace es que en el *wrapping* se saca sólo de la lista de parámetros en función del valor del *checkbox* de dibujar las normales. Si éste se encuentra activo, se sacan el resto de parámetros concernientes a las normales. Si no lo está, no se sacan de la lista de parámetros y se invoca la misma función con valores *Null* o cero. Dentro del código C++ también se tiene en cuenta el valor de dicha variable a la hora de hacer uso de los parámetros que le son pasados.

En cuanto a la segunda, viene motivada porque la existencia de bucles de muchas iteraciones en el lenguaje de *script* hace que la ejecución sea muy lenta. Es por eso que la parte de ir recorriendo los puntos y dibujar en ellos haciendo uso de las imágenes calculas en el *script* se encuentre desarrollada en código C++.

Entrando en detalle en esta parte de la operación, dentro del código C++ se reciben los parámetros antes comentados en la llamada a la función *wrapped*. En primer lugar se asignan los valores de color, grosor y estilo al visor para poder usarlos al dibujar (en principio para los bordes, aunque de la misma manera se actúa en el caso de las normales). Para dibujar el borde, simplemente se recorren las imágenes de puntos que se han calculado desde el *script*, comprobando que estos se encuentren en la zona que se está visualizando (recordar que en ese caso, en el que un punto se esté visualizando, la imagen binaria tendrá el valor uno). En el caso que se haya decidido dibujar las normales, nuevamente se asignan los valores de color, grosor y estilo; puesto que se ha podido decidir personalizar la manera en que se dibujan las normales al contorno. Igualmente, sólo se dibujan aquellas normales que se encuentren dentro de la zona de visualización. Además, se hace el dibujado de la punta de la flecha de la normal. Finalmente, se refresca el visor mediante la operación `DrawingAreaDisplay()` para que se refleje en pantalla lo que se ha dibujado.

- `Run()`: esta operación se encarga de invocar al método que se ha seleccionado en la interfaz para detectar los bordes. En primer lugar llama a la función `load_input()` para obtener la imagen de entrada. Si la imagen de entrada tiene más de un canal (por ejemplo una imagen RGB) se procede a normalizarla. Si existe algún resultado previo de haber aplicado el método o un objeto de la clase `SubPixel2D`, se borran.

Se crea un objeto de la clase `SubPixel2D` invocando al constructor, al que se le pasan: la imagen de entrada, el valor del umbral y la variable que indica si se trata de detectar un contorno de primer orden o no. En función del método que se haya seleccionado en la interfaz, se invoca a uno u otro método. Además de ello, se le asigna a una variable el nombre del método usado. La razón de hacer esto es para mostrarlo en la pestaña de estadísticas y así tener una referencia de cuál ha sido el método sobre el que se han calculado las mismas. Una vez se ha aplicado el método en cuestión, se habilitan las pestañas de información del píxel y de estadísticas, que hasta ese instante estaban deshabilitadas (evitando así que se pueda hacer alguna de las operaciones sin tener disponible previamente un resultado sobre el que aplicarlas).

Finalmente se invocan las operaciones de `Display()` y `drawBorder()` para mostrar la imagen y dibujar los bordes. En función de la selección que se haya hecho en los parámetros de visualización (dibujar en la imagen suavizada, en la restaurada o haber dejado la opción por defecto) se habilitará apropiadamente el `PaintCallback()` a la variable del visor asociada. Esto permitirá que ante cualquier operación de *zoom* o de desplazamiento, se invoque al método `drawBorder()` para redibujar los contornos que sean necesarios.

- `Display()`: este método se ocupa de mostrar por pantalla las imágenes a través del comando `show` que `AMILab` tiene asociado a las variables de tipo imagen. Para ello se comprueba que exista la imagen de entrada, ya que por defecto se dibuja el resultado sobre ella. Puede que el usuario haya decidido mostrar la imagen suavizada o restaurada si ha elegido alguno de los métodos en los que éstas son resultado. Por ello se chequea si se ha seleccionado que se desea mostrar alguna de ellas. En ese caso, se toma la imagen adecuada que se encuentra dentro de la estructura de tipo `AMIObject` y se muestra en comparación con la imagen de entrada. Así mismo se verifica que se haya decidido que se desee dibujar los contornos en la imagen suavizada o restaurada.
- `AutoRun()`: esta operación simplemente se ocupa de volver a invocar al método `Run()` cuando sea preciso.
- `Save()`: esta función permite salvar el resultado. Actualmente no es usada, pero ésta se ha incluido con vistas a cuando se use el algoritmo como fase de preproceso para otra funcionalidad.
- `Enable()`: se le permite al usuario cambiar algunas características de los bordes y las normales, como el color, grosor y tipo. Si se selecciona la opción de cambiar los parámetros por defecto, este método se ocupa de habilitar los paneles para los parámetros antes mencionados. Para ello está en *callback* del *checkbox* que permite seleccionar si se desea cambiarlos o no.
- `UpdateBoxPanel()`: esta operación es invocada cuando se cambia el método al que se quiere invocar y se encarga de resetear los parámetros por defecto, así como los *boxpanel* no asociados al mismo. Por ejemplo, si se selecciona el método básico, además de resetear los parámetros por defecto, se deshabilitarán los *boxpanel* de los otros dos métodos, ya que no se hará uso de ellos.
- `CreateGui()`: este procedimiento es el encargado de crear la interfaz de usuario, la cual será descrita en el siguiente apartado.

Interfaz

En la interfaz se crea un *book* en el que se irán añadiendo las diferentes pestañas que incluirán las funcionalidades necesarias. En la primera de ellas, la pestaña principal, denominada *Global parameters*, se encuentran las principales características del método (ver figura 4.23). Entre ellas se encuentra un menú desplegable para seleccionar la imagen de entrada, el nombre para la salida (actualmente deshabilitado porque no se está usando), un *slider* para seleccionar el valor de umbral y un *checkbox* para indicar si se desea detectar contornos de primer orden o no. Posteriormente, en otro menú desplegable, se puede seleccionar el método que se quiere aplicar. Actualmente se tienen a disposición del usuario tres métodos: el básico (ventana de 5×3) denominado *Basic detector*, el método para imágenes afectadas por bajo nivel de ruido con detección de bordes cercanos y muy cercanos (ventana de límites variables, dinámicos o ventana flotante) llamado *Averaged detector*, y el método de restauración *Subpixel denoising*.

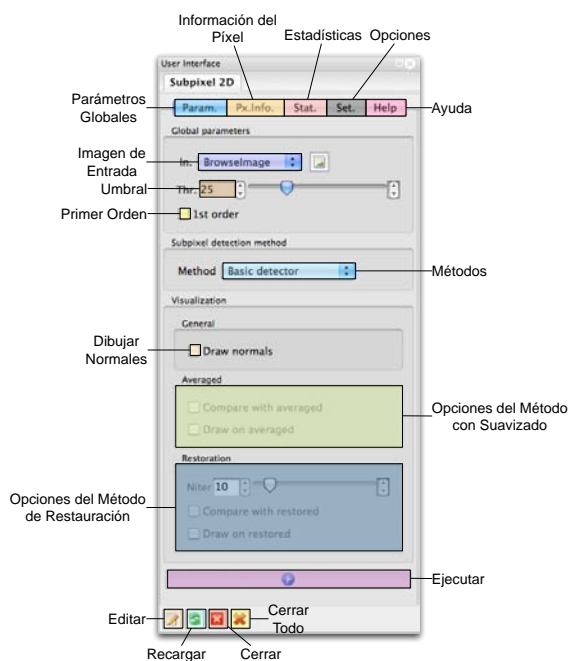


Figura 4.23: Pestaña principal de la interfaz para la detección con precisión sub-píxel

Posteriormente se encuentran parámetros de visualización. El primero de los *boxpanel* incluye un *checkbox* para seleccionar si desean dibujar las normales al contorno o no. Posteriormente se encuentran características relativas al método para imágenes afectadas por bajo nivel de ruido. Se tratan de dos *checkboxes* que permiten al usuario seleccionar si quiere mostrar la imagen suavizada en comparación con la imagen de entrada y/o si quiere además dibujar los contornos sobre la imagen suavizada en lugar de sobre la imagen de entrada (opción por defecto). Por último están las opciones relativas al método de restauración. En primer lugar se tiene un *slider* en el que se puede seleccionar el número de iteraciones que se quieren hacer. Luego se tienen, al igual que en el método anterior, dos *checkboxes* para que el usuario pueda elegir si quiere mostrar en comparación la imagen restaurada con la de entrada y/o dibujar los contornos sobre la restaurada. En la parte inferior de dicha pestaña se encuentra el botón *Run* que permite ejecutar el método con todas las opciones que se han seleccionado.

Además en la figura 4.23 se pueden ver en la parte inferior los botones de editar, recargar, cerrar y cerrar todo; que han sido descritos con anterioridad y que pertenecen a la herencia de `ScriptGui`. Además, en la parte superior puede observarse que se tienen tres pestañas más, que van a ser descritas a continuación.

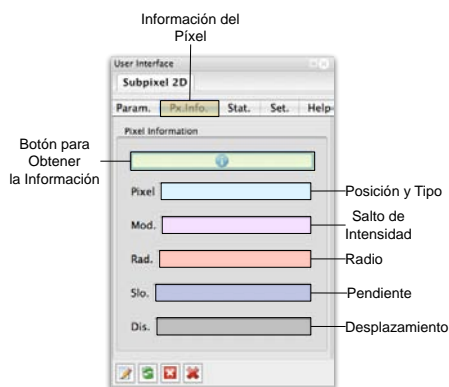


Figura 4.24: Pestaña de información del píxel

La siguiente pestaña es la dedicada a la información del píxel (figura 4.24). En la parte superior se encuentra un botón con el *bitmap* “i” al que hay que darle para obtener la información del píxel sobre el que previamente se ha hecho clic en el visor.

Dentro de la información con la que se cuenta está:

- La posición (x, y) y tipo de borde (orientación).
- El salto de intensidad entre ambos lados del borde.
- El radio.
- La pendiente.
- El desplazamiento.

En caso de que el píxel seleccionado no sea un píxel borde en el campo “*Pixel*” se indicará la posición (x, y) del mismo y se dirá que no se trata de un píxel borde a través del texto “*No edge*”.

Otra de las funcionalidades incluidas es el cálculo de estadísticas (fig. 4.25). En la parte superior de esta pestaña se encuentra un campo con la etiqueta *Method*. En él lo que aparecerá es el nombre del método que ha sido aplicado para la detección de los contornos, de forma que se sepa a cuál pertenecerían las estadísticas que se van a calcular. Para poder realizar al cómputo es preciso darle al botón que se encuentra bajo dicho campo y que posee el *bitmap* de un diagrama circular. Bajo él hay una serie de *boxpanels* que agrupan los diferentes cálculos estadísticos que se harán: mínimo, máximo, media, varianza y desviación estándar. En cada uno de ellos se encuentran las tres variables sobre las que se calculan los mismos, que no son otras que el salto de intensidad, el radio y la pendiente (detalle en la zona izquierda de la figura 4.25).

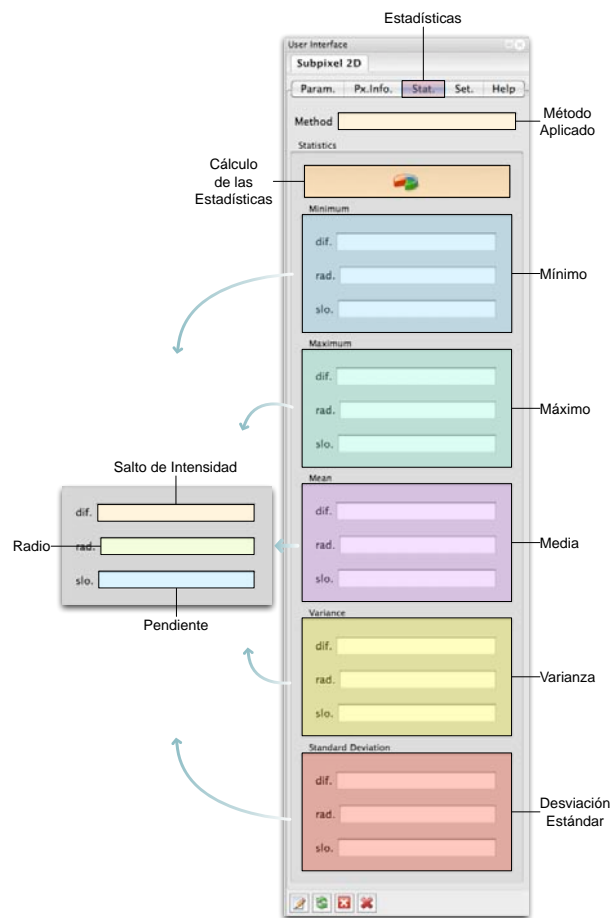


Figura 4.25: Pestaña del cálculo de estadísticas

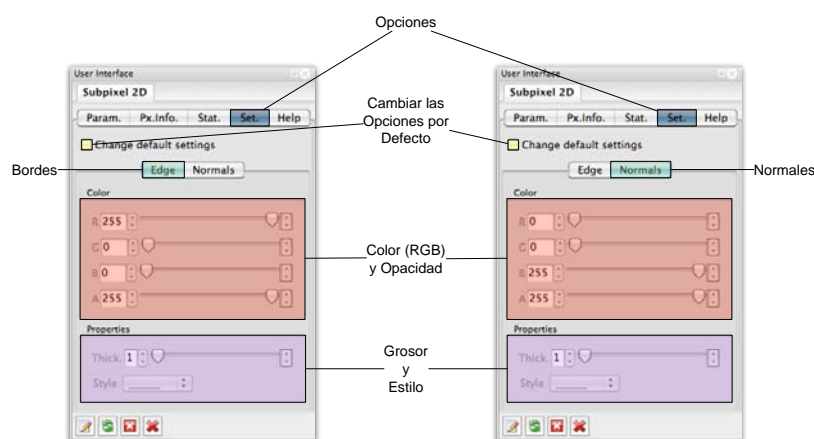


Figura 4.26: Pestaña de opciones

Además de todo lo anterior, se pueden cambiar las opciones que por defecto tiene el dibujado de los contornos. Por ello es que existe la pestaña de opciones (*Settings*) (para mayor detalle véase la figura 4.26). De manera estándar, los contornos serán dibujados en rojo, sin transpa-

rencia y con un estilo continuo. Las normales por su parte se dibujan en azul, sin transparencia y también con estilo continuo. No obstante, dentro de esta pestaña se permiten cambiar estas opciones por defecto si se selecciona el *checkbox* “*Change default settings*”. En ese caso aparecerán activas tanto la pestaña de los contornos como la de normales. En ambas se permite cambiar el color (en formato RGB) así como su opacidad (*slider* etiquetado con *A*, que denota al canal alfa). Además de ello, se pueden asignar diferentes grosores así como estilos (continuo o punteado).



Figura 4.27: Pestaña de ayuda

La última de las pestañas (fig. 4.27) es la dedicada a la ayuda. Es un pequeño visor html a través del cual se puede navegar por la documentación. Originalmente, como se ha dicho con anterioridad, esta documentación ha sido generada a partir de un documento en \LaTeX y luego convertida a html con \LaTeX2html .

En la parte superior de la misma se encuentran una serie de botones a través de los cuáles se puede acceder directamente a la wiki de AMILab, a la página *home* de la ayuda, ir hacia atrás o hacia delante. En la captura de la figura 4.27 se muestra la página principal o *home*. En ella se encuentra un pequeño resumen del *script*, así como un menú en el que se encuentran las diferentes páginas por las que se puede navegar en la ayuda.

En la figura 4.28 se puede ver el resultado de aplicar el método de detección sub-píxel a imágenes sintéticas.

En las figuras 4.29, 4.30, 4.31 y 4.32; se muestran resultados de la aplicación del método sobre imágenes reales.

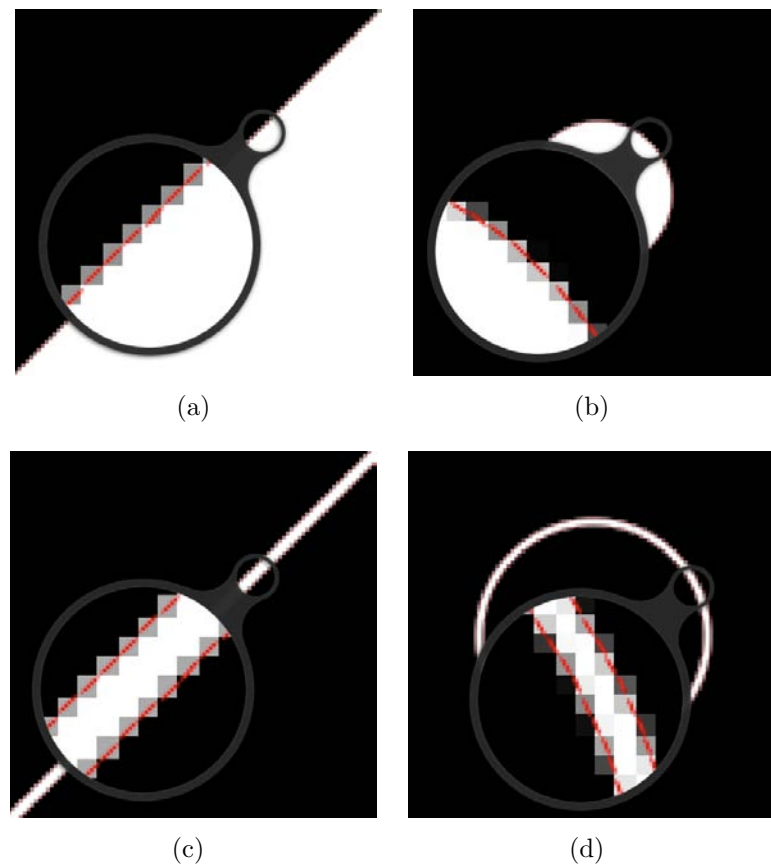


Figura 4.28: Resultado de la detección sub-píxel en imágenes sintéticas: (a)Rampa, (b)círculo, (c)vaso y (d)anillo

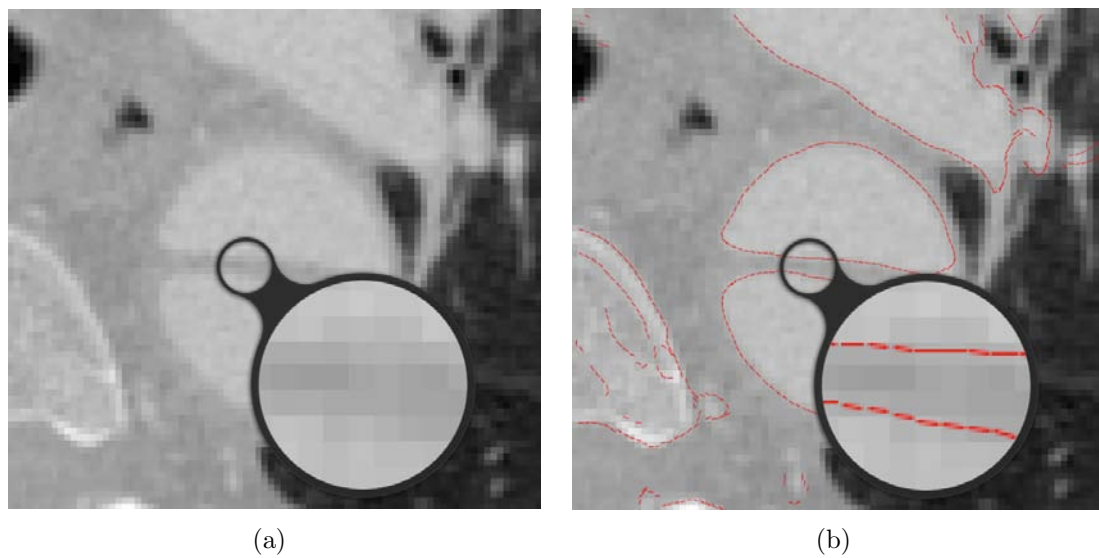


Figura 4.29: Resultado de la detección sub-píxel en un un corte axial de una tomografía computarizada de una disección aórtica: (a)Imagen original y (b)detección sub-píxel

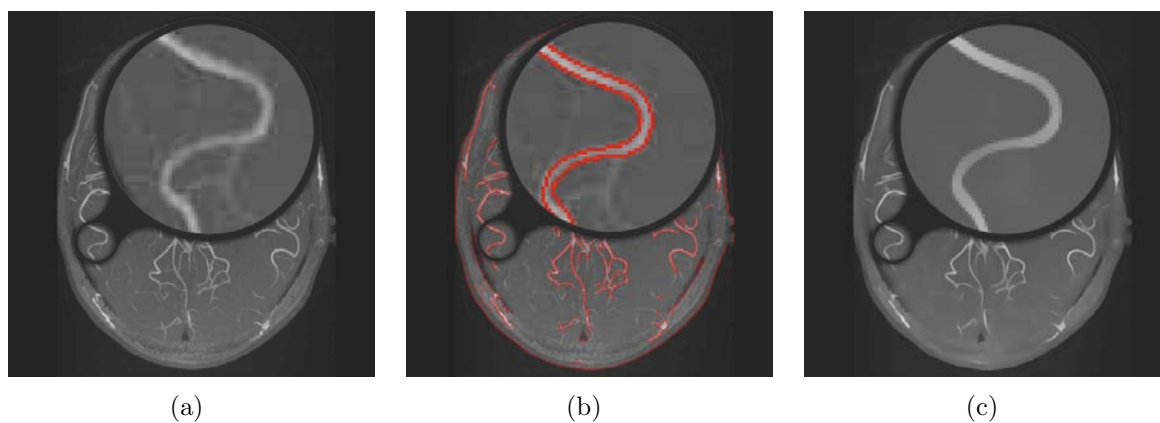


Figura 4.30: Resultado de la detección sub-píxel en una angiografía cerebral: (a)Imagen original, (b)detección sub-píxel y (c)imagen restaurada

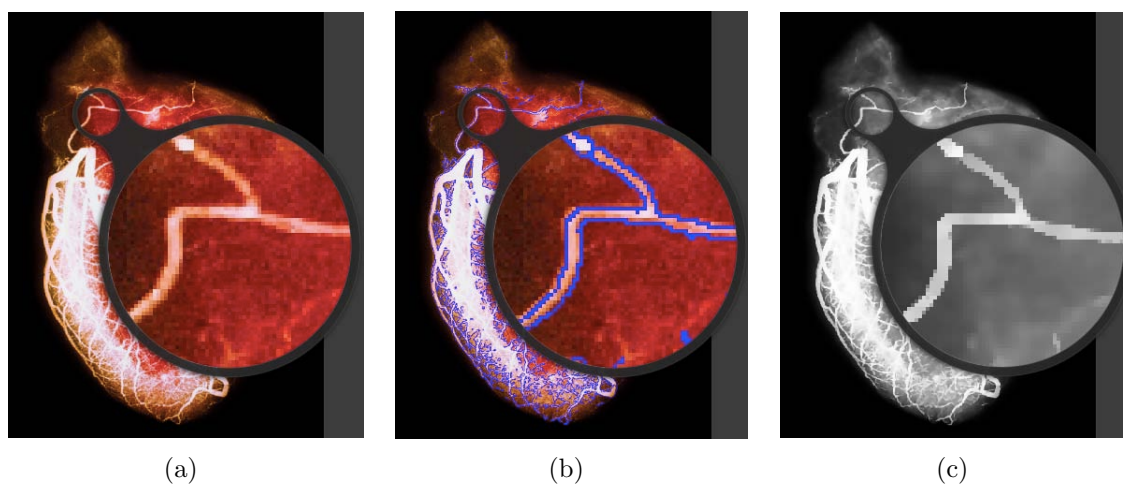


Figura 4.31: Resultado de la detección sub-píxel en una angiografía coronaria: (a)Imagen original, (b)detección sub-píxel y (c)imagen restaurada

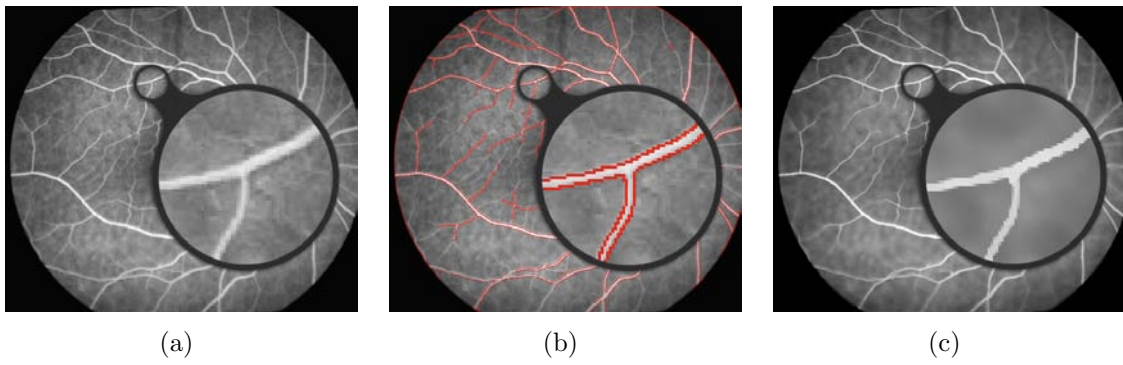


Figura 4.32: Resultado de la detección sub-píxel en una imagen de los vasos oculares: (a)Imagen original, (b)detección sub-píxel y (c)imagen restaurada

4.2. 3D

4.2.1. Métodos a Nivel Vóxel

Al igual que sucedía con las imágenes bidimensionales, se empieza por desarrollar métodos a nivel vóxel. Para ello se parte de lo descrito en la sección 3.2.1. No obstante, se sigue un camino que difiere al descrito en el desarrollo de los métodos a nivel píxel. En éste se realizaba una implementación en C++ de los detectores, para posteriormente hacer un *wrapping* y acceder a los mismos desde el lenguaje de *script*. Ahora, este desarrollo se hará directamente desde el lenguaje de *script*, haciendo uso del método `Convolve(var_image, var_image)` provisto por el lenguaje de AMILab.

`Convolve(var_image, var_image)` permite realizar la operación de convolución sobre la primera imagen usando la imagen segunda. Es decir, nuestra imagen de entrada (o señal dicho de forma genérica) será el primer parámetro, mientras que la máscara será el segundo. Ya que las imágenes que se le pueden pasar a la función pueden ser tridimensionales, esto ahorra el tener que desarrollar un método de convolución directamente en C++, el cual sería bastante más complejo que el implementado para el caso bidimensional.

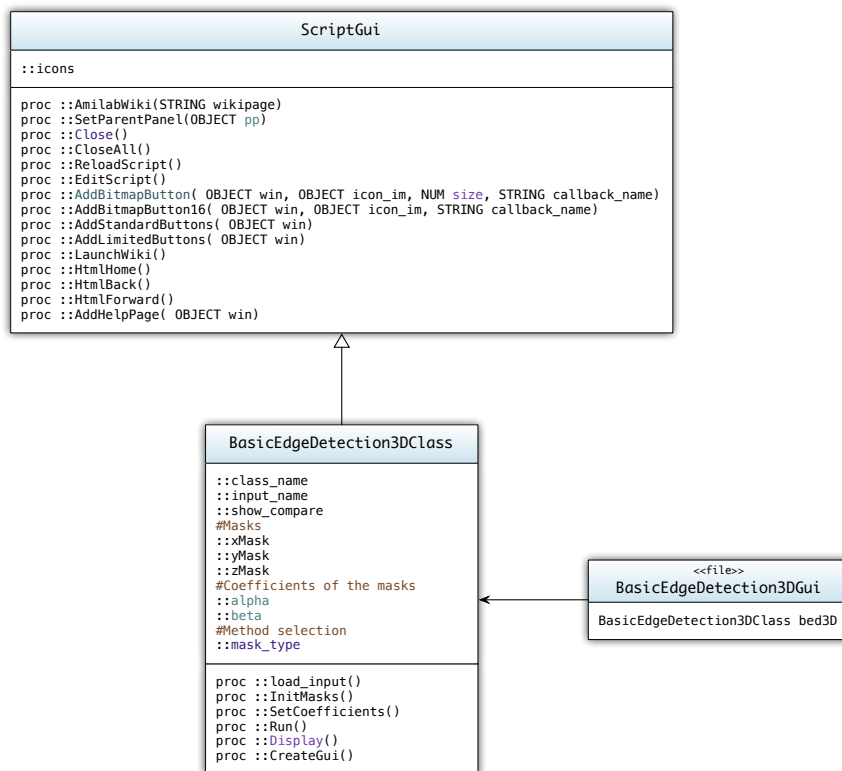


Figura 4.33: Estructura de clases del *script* para la detección a nivel vóxel

En la figura 4.33 se puede observar la estructura de clases que se tiene. De la misma forma que en la sección anterior se puede ver la existencia de una herencia a partir de la clase `ScriptGui`

que proporciona el entorno básico de interfaz, así como la inclusión de *bitmap buttons*; y que hay una separación entre lo que es la clase en sí (`BasicEdgeDetection3DClass`) y el fichero que se encarga de cargar la interfaz (`BasicEdgeDetection3DGui`).

La clase `BasicEdgeDetection3DClass` posee los siguientes atributos: el nombre de la clase, el nombre de la imagen de entrada que va a ser procesada, una variable para determinar si se quiere mostrar o no el resultado en comparación, es decir, poner en comparación la imagen de entrada con la detección realizada. Además se tienen las tres imágenes que van a representar las máscaras a partir de las cuáles se realizará el proceso de convolución. Asimismo, son necesarios los coeficientes α y β , que varían dependiendo del método seleccionado. Por último se tiene un atributo que contendrá cuál ha sido el método seleccionado para aplicar la detección a nivel vóxel.

En el apartado de métodos se tienen los siguientes:

- `load_input()`: este método se encarga de cargar la imagen de entrada, para ser procesada posteriormente.
- `InitMasks()`: esta función es la que se ocupa de inicializar las máscaras tridimensionales a partir de las ecuaciones descritas en 3.2.1, usando los valores adecuados para α y β en función del método de detección que se haya seleccionado.
- `SetCoefficients()`: considerando cuál ha sido el método seleccionado en la interfaz, esta operación es la encargada de asignar los valores adecuados a α y β .
- `Run()`: este procedimiento realiza la ejecución del método elegido. Para ello, lo primero que hace es invocar a `load_input()` (para tener la imagen de entrada), a `SetCoefficients()` (para tener los valores adecuados de α y β) y a `InitMasks()` (para tener las máscaras inicializadas). Con todo ello, se puede proceder a la aplicación de la operación de convolución a través de la función `Convolve(var_image, var_image)` que provee `AMILab`. Finalmente, el resultado será el que se obtiene de calcular el módulo del gradiente como la raíz cuadrada de la suma de cada componente al cuadrado. Con ésto se llama a la operación `Display()`, que mostrará lo que se ha obtenido.
- `Display()`: este método se ocupa de mostrar por pantalla el resultado. Se tiene en cuenta si el usuario ha decidido ver en comparación la imagen de entrada con la detección a nivel vóxel que se ha realizado.
- `CreateGui()`: es la operación encargada de crear la interfaz, que será descrita a continuación.

4.2.1.1. Interfaz para los Detectores de Bordos Básicos en 3D

Los detalles de la interfaz pueden verse en la figura 4.34.

Dicha interfaz cuenta sólo con una pestaña principal *Param.* en la que se pueden encontrar todas las opciones. En primer lugar se encuentra la lista para seleccionar cuál será la imagen de entrada para aplicar el método. También en una lista se puede seleccionar cuál es el que se quiere aplicar. Se encuentran a disposición del usuario tres métodos: Prewitt, Sbel y lvarez.

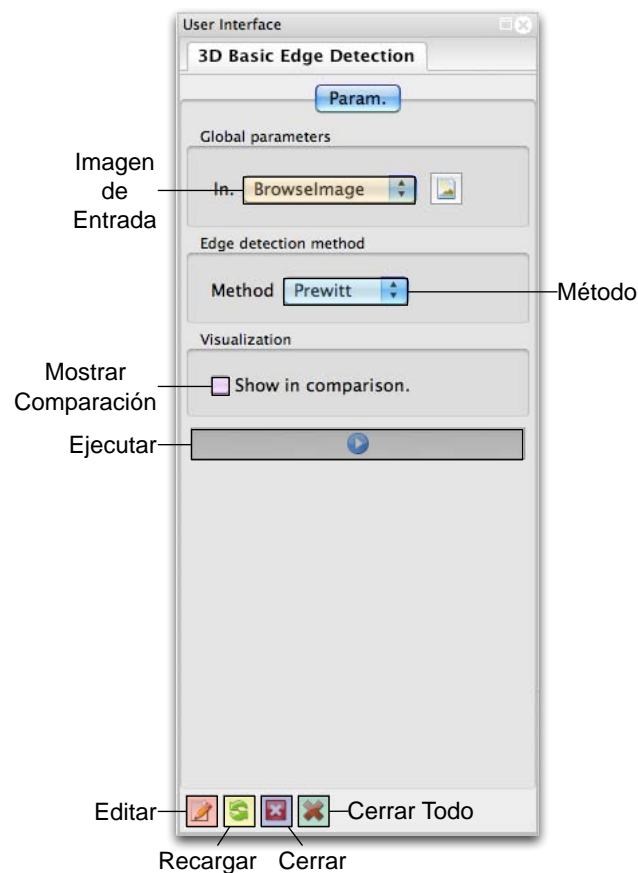


Figura 4.34: Interfaz para los detectores de bordes a nivel vóxel



Figura 4.35: Arteria carótida. Imagen cortesía de The Visual MD

Además, a través de un *checkbox*, es posible que se seleccione la opción de visualizar en comparación, de forma que aparecerá la imagen de entrada comparada con el resultado de la detección. Por último, en la parte inferior de la pestaña principal, se tiene un *bitmap button* para ejecutar.

Al contrario que sucedía con el *script* básico para la detección a nivel píxel en imágenes bidimensionales, en este caso, como se cuenta con la herencia de la clase `ScriptGui`, se pueden ver en la parte inferior de la interfaz los botones que permiten editar, recargar, cerrar y cerrar todo.

Para probar los métodos a nivel vóxel se ha usado una imagen real, correspondiente a la arteria carótida (figura 4.35). La carótida es la arteria principal de la cabeza y el cuello. Se divide en ramas izquierda y derecha, y éstas a su vez en ramas externa e interna [39].

En la figura 4.36 se muestran los resultados de aplicar una detección a nivel vóxel. En ella se muestran además de la original, los resultados de aplicar la detección usando las máscaras de Prewitt, Sobel y de L. Álvarez, que fueron descritas en la sección 3.2.1.

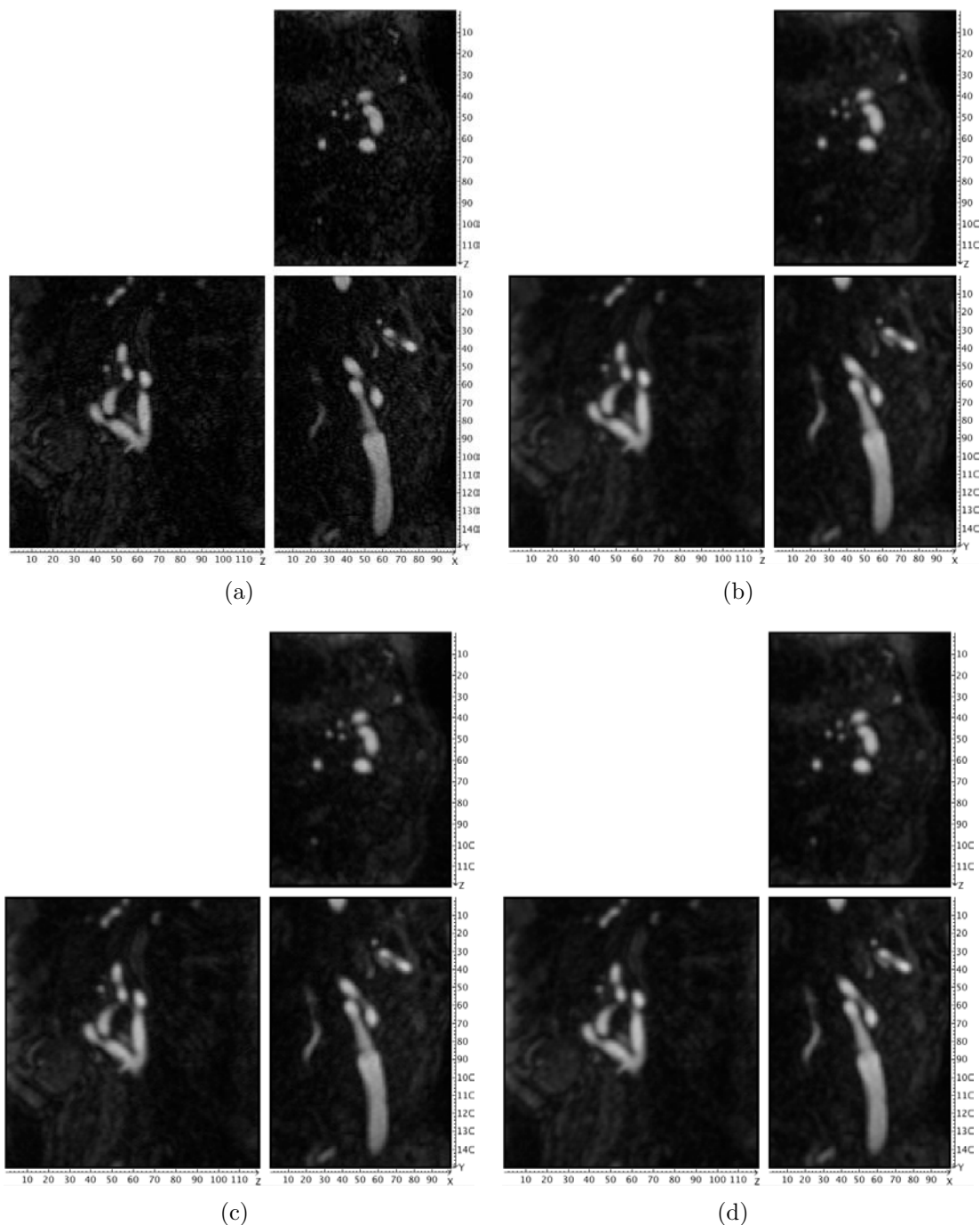


Figura 4.36: Detección a nivel vóxel de una imagen de TC de la carótida: (a) Imagen original, (b) resultado de aplicar las máscaras de Prewitt, (c) las de Sobel y (d) las de L. Álvarez

Sin embargo para mayor claridad, en la figura 4.37 se incluyen las reconstrucciones a través de renderizado volumétrico tanto de la imagen de entrada como de las detecciones realizadas

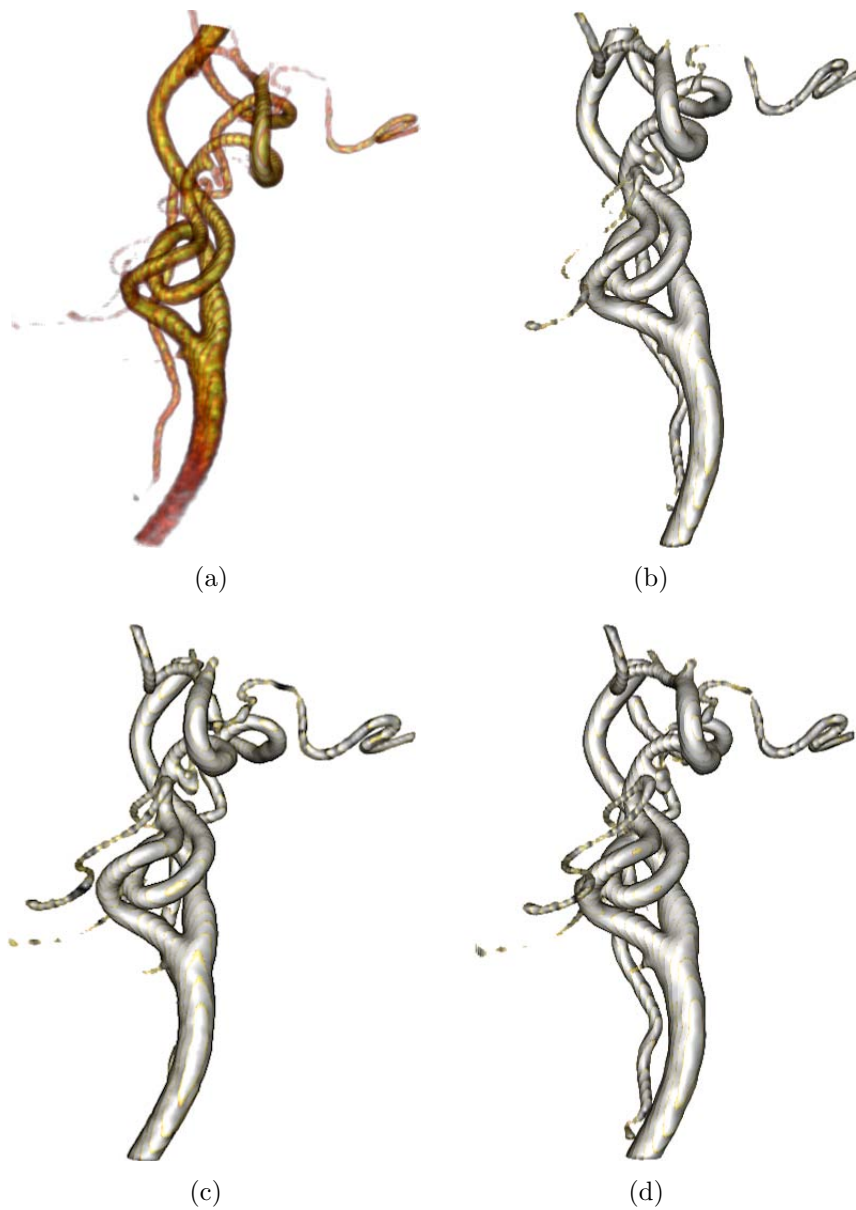


Figura 4.37: Reconstrucción mediante renderizado volumétrico de la detección a nivel vóxel de una imagen 3D de la carótida: (a) Imagen original, (b) resultado de aplicar las máscaras de Prewitt, (c) las de Sobel y (d) las de L. Álvarez

a nivel vóxel. Se ha intentado que la orientación de la reconstrucción sea aproximadamente la misma para mostrar los mismos detalles en todas las imágenes. Nótese que en la imagen sólo se ha tratado de mostrar la región de la propia arteria carótida, sin tener en cuenta las estructuras adyacentes de la región. Esto se ha logrado configurando la región sobre la que actúa la función de transferencia, dando sólo importancia a aquellos niveles de intensidad que forman parte de la arteria.

4.2.2. Método Sub-Vóxel de Segundo Orden Básico

Al igual que en el caso bidimensional, se cuenta con la implementación previa desarrollada para el entorno `XMegaWave`. No obstante, de la misma manera que antes, es preciso realizar una adaptación a la estructura de datos para las imágenes, `InrImage`. Sin embargo, en esta ocasión la tercera dimensión no será despreciada, ya que será necesaria. Por tanto, en esta ocasión, en el interior del código del método se pasará de tener un doble a un triple bucle anidado para recorrer todas las posiciones de la imagen de entrada.

```

borderVoxel
  ▼ Properties
  A:double
  B:double
  a:double
  b:double
  border:unsigned char
  c:double
  curvature:double
  d:double
  f:double
  g:double
  nx:double
  ny:double
  nz:double
  px:int
  py:int
  pz:int
  ▼ Operations
  borderVoxel ()
  getAIntensity ()
  getBIntensity ()
  getBorder ()
  getCoefficient_a ()
  getCoefficient_b ()
  getCoefficient_c ()
  getCoefficient_d ()
  getCoefficient_f ()
  getCoefficient_g ()
  getCurvature ()
  getNx ()
  getNy ()
  getNz ()
  getPosX ()
  getPosY ()
  getPosZ ()
  printBorderVoxel (lin...
  setBorderVoxelValue...
  ~borderVoxel ()

```

Figura 4.38: Clase `borderVoxel`

En relación a la detección de los contornos, nuevamente se hace una gestión eficiente de la memoria. Ya se ha comentado con anterioridad que este hecho es más relevante en imágenes tridimensionales, ya que el consumo de espacio es superior. En el caso 3D se hace uso de la clase `borderVoxel` para guardar la información concerniente a un vóxel borde (figura 4.38). Los miembros de esta clase son los valores de intensidad a ambos lados del borde, los coeficientes del paraboloides que mejor se ajusta al contorno, el tipo de borde, la curvatura, la normal y el punto en el espacio en que está dicho vóxel (x, y, z) .

En cuanto a los métodos, son prácticamente los mismos que en el caso 2D. Se tienen las operaciones `get` y `set` de cada uno de los miembros, así como el constructor y destructor, además del procedimiento `setBorderVoxelValue`.

Con esta clase se consigue almacenar la información relativa a un vóxel, pero ha de ser incluida en algún tipo de estructura para poder almacenar la información de todos los vóxeles

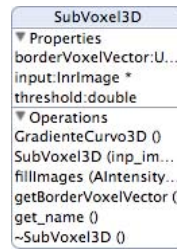


Figura 4.39: Clase SubVoxel3D

pertenecientes a un contorno. Igual que en el otro caso, se opta por una estructura vectorial de la STL de C++, cuyo tipo será la clase antes mencionada (`borderVoxel`). Con ello, cada vez que se haya detectado un vóxel borde y se realicen todos los cálculos relacionados con él, se introducirá la información en la instancia de un objeto del tipo `borderVoxel` que será añadido al vector mediante el método `push_back`. Este vector puede verse en la figura 4.39, en el atributo `borderVoxelVector`.

La clase `SubVoxel3D` contiene, además del vector de vóxeles borde, dos miembros más. El primero de ellos es un puntero a un objeto de la clase `InrImage` denominado `input`, que representará la imagen de entrada a la cual se le van a calcular los contornos. El segundo y último es, al igual que en 2D, el umbral necesario para decidir cuándo estamos ante un vóxel que es candidato a borde o no.

En el apartado de métodos, esta clase cuenta con los típicos constructor y destructor. La operación que se ocupa de la detección de contornos es la denominada `GradienteCurvo3D`. Además se cuentan con dos funciones más, una para obtener el vector miembro de la clase que contiene los contornos y la que se usa para rellenar las imágenes en el *wrapping*.

Atendiendo ahora a la función de detección de contornos, como ya se ha dicho, en esta ocasión la estructura principal se encuentra inscrita en el interior de tres bucles anidados, los cuáles permiten recorrer todos los vóxeles de la imagen de entrada. En la figura 4.40 se puede ver el diagrama de flujo que sigue el comportamiento del algoritmo.

A partir de la imagen de entrada y, vóxel a vóxel, se van a ir calculando las derivadas parciales en x , y y z . En función de cuál sea el mayor valor de las tres, se orientará de una manera u otra la ventana de detección (para mayor detalle ver la citada figura 4.40). No obstante, independientemente de cuál sea la orientación, el resto de pasos son comunes salvando el detalle que se usarán unos u otros vóxeles para los cálculos en función de cómo esté la ventana.

Una vez se sabe cuál es la posición de la ventana de detección, se chequea si la parcial es mayor que el umbral dado como parámetro de entrada. En caso de no ser así, se pasará a tratar el siguiente vóxel. Sin embargo, si es mayor, se continúa con el siguiente paso, que es el cálculo de las intensidades A y B (debajo y encima del contorno, respectivamente). Una vez se tienen los valores de ambas intensidades, se comprueba si su diferencia en valor absoluto es menor que el umbral. En caso afirmativo habría que pasar al siguiente vóxel candidato a borde. De lo contrario, sólo faltaría computar las sumas de las columnas, para con ello, calcular los coeficientes del paraboloides que mejor se ajusta al contorno, así como de las componentes de la normal a dicho

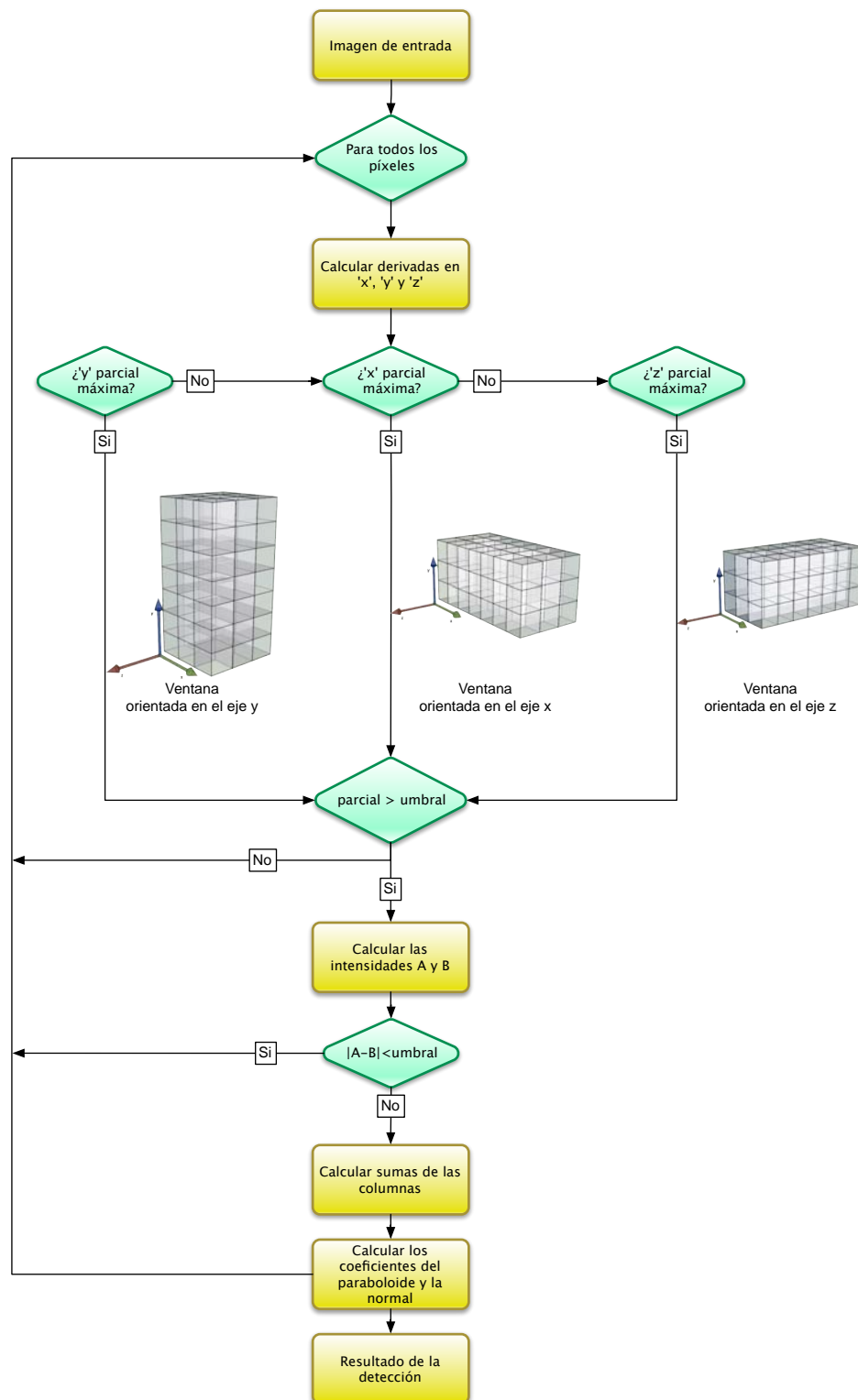


Figura 4.40: Diagrama de flujo para la detección de contornos de segundo orden en imágenes tridimensionales

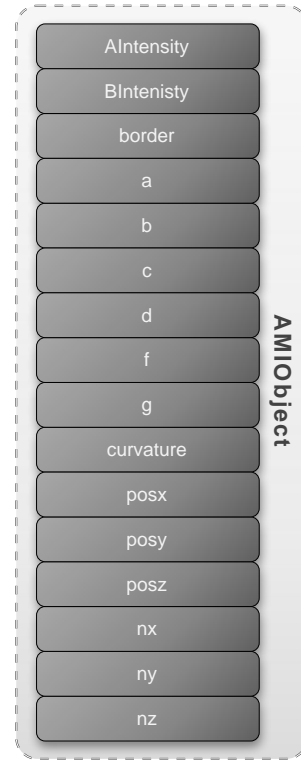
borde. Con todos esos valores se rellena una instancia de la clase `borderVoxel`, que pasará a

formar parte del vector de vóxeles borde que se vayan detectando (`borderVoxelVector`).

Al final de todas las iteraciones se tendrán todos los resultados de la detección en la estructura vectorial de la STL de C++ anteriormente descrita. De esta manera se procede con imágenes tridimensionales para realizar una detección con precisión sub-vóxel de contornos de segundo orden.

Atendiendo al apartado de *wrapping*, el sistema es bastante similar a los anteriores. El constructor de la clase cuenta como parámetros con la imagen de entrada, el valor del umbral y la variable que indicará si se intenta detectar un contorno de primer o de segundo orden.

En el *wrapping* de la función que se ocupa de la detección de los contornos, se hace la llamada al método de la clase que la realiza. A la hora de devolver el resultado se procede encapsulando los valores en imágenes unidimensionales dentro un objeto de tipo `AMIObject` (figura 4.41). Dentro del mismo se encontrarán: las intensidades *A* y *B*, el tipo de borde, los coeficientes del paraboloide, la curvatura, la posición del vóxel en el interior de la imagen y las componentes de la normal al contorno.



4.2.2.1. Script en AMILab

El *script* en `AMILab` es bastante similar al visto con anterioridad para el caso bidimensional, salvo que en esta ocasión sólo cuenta con el método que ha sido implementado hasta ahora.

En la figura 4.42 se puede ver la estructura del directorio donde se encuentra el *script* para la detección de contornos con precisión sub-vóxel. La estructura es bastante similar a la vista en el caso 2D. En ella puede observarse que dentro del directorio se encuentra una carpeta denominada “Doc”. Dentro de ella se encuentra el fichero `LaTeX` con el nombre `subvoxel3D.tex`, así como el directorio del mismo nombre. Dicha carpeta se ha obtenido al aplicar `LaTeX2html` al archivo `LaTeX`. Además, están los *scripts* de la clase y que cargan la interfaz (`subvoxel3DClass.amil` y `subvoxel3DGui.amil` respectivamente), el *script* que añade el método al menú y a la barra de herramientas (`Add2Menu.amil`), así como los iconos 16×16 y 32×32 que aparecerán en el menú y la barra de herramientas respectivamente.

En el diseño del icono (fig. 4.43) se ha partido de la imagen de un cubo presente en la colección libre que se encuentra dentro del directorio de *scripts*. Se ha intentado reflejar un plano inscrito en el interior del cubo, que representa un vóxel. Además, perpendicular a dicho plano se encuentra la normal. De esta forma se intenta representar la detección con precisión

Figura 4.41: Estructura de tipo `AMIObject` que encapsula el resultado de la detección de bordes en imágenes 3D

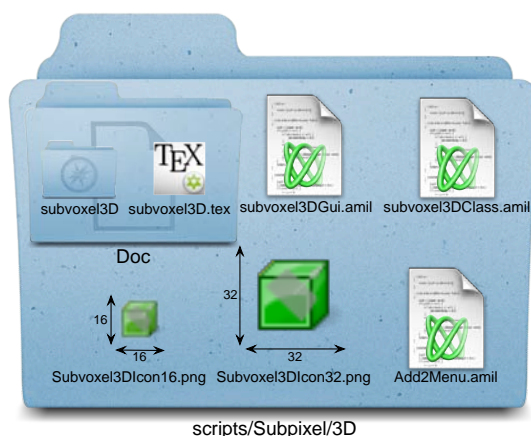


Figura 4.42: Estructura del directorio 3D del *script* para la detección con precisión sub-vóxel



Figura 4.43: Diseño del icono que aparece en la barra de menú y la barra de herramientas de AMILab para el algoritmo de detección con precisión sub-vóxel

sub-vóxel.

En la figura 4.44 se puede ver la estructura de clases de los *scripts* para imágenes 3D.

La separación y herencia es la típica vista con anterioridad. Existe un estereotipo fichero `subvoxel3DGui`, que es el encargado de crear un objeto de la clase `Subvoxel3DClass` y llamar a la función encargada de crear la interfaz.

Por su parte la clase `Subvoxel3DClass` hereda de `ScriptGui` las características básicas de interfaz y la capacidad de usar los *bitmap buttons*. Dentro de los atributos que se pueden encontrar dentro de la clase para la detección con precisión sub-vóxel se encuentran: el nombre de la clase, el nombre de la imagen de entrada, el umbral, el tipo de algoritmo que se va a llamar (pensado de cara al futuro para cuando haya más de uno) y el enumerado que los contiene. Además de ellos existen algunos destinados a la visualización o no del volumen, los planos y las normales.

El resto atributos tienen que ver con características propias de los elementos de la escena: la opacidad y color de los planos, opacidad y color de las normales, opacidad del volumen y color del fondo del visor. Por último, existen dos variables con referencias a la ayuda, tanto on-line

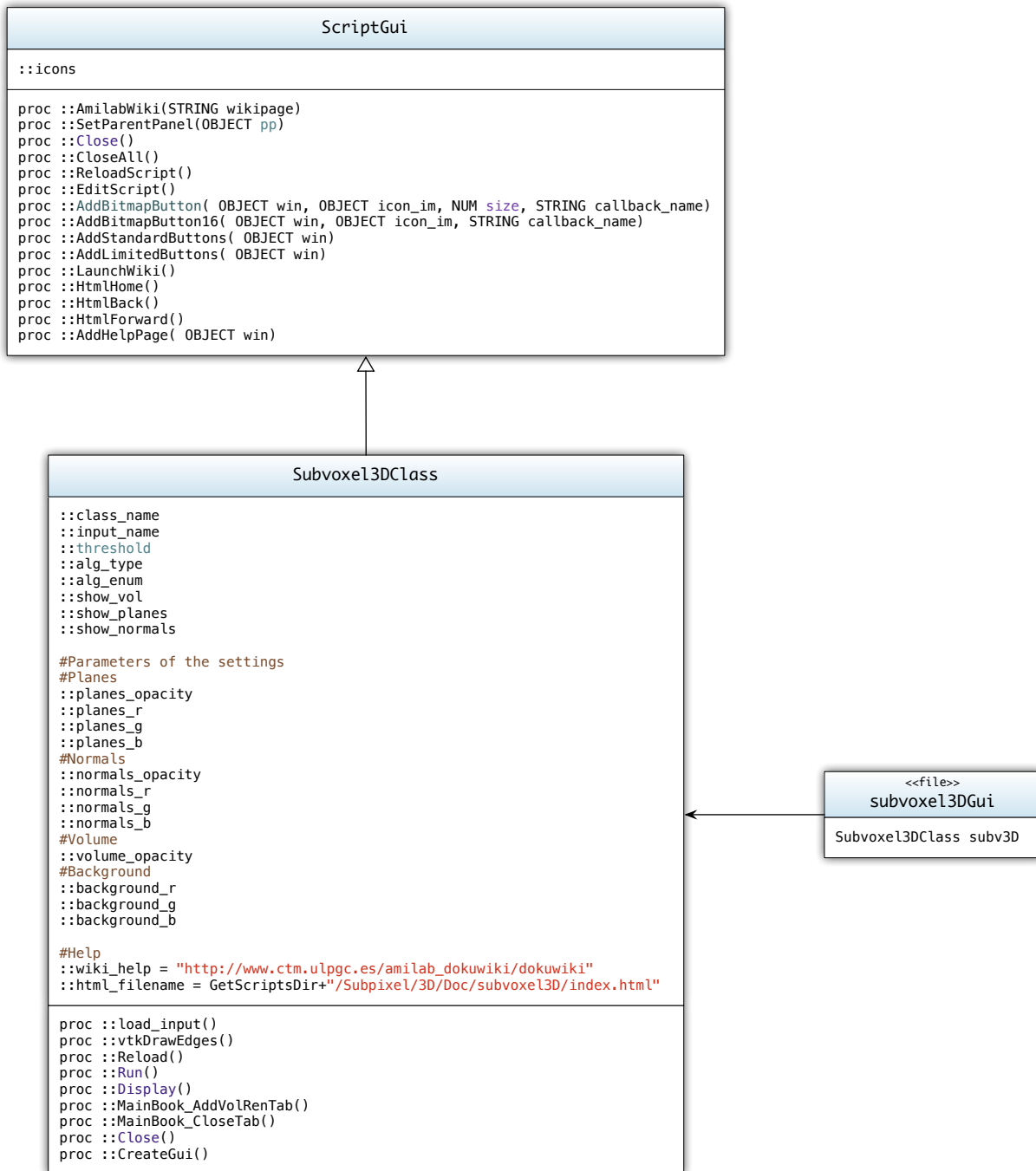


Figura 4.44: Diagrama de clases del *script* para la detección de contornos de segundo orden en imágenes 3D

como la que se visualizará en la pestaña *help*.

Los métodos con los que cuenta la clase son los siguientes:

- `load_input()`: esta función es la encargada de cargar la imagen de entrada a la cual se

le van a calcular los contornos.

- `vtkDrawEdges()`: este procedimiento se encarga de dibujar los contornos detectados. Para ello utiliza un proceso conocido como *Glyphing*. El *Glyphing* es una técnica de visualización de datos mediante el uso de símbolos o *glyphs* (figura 4.45).

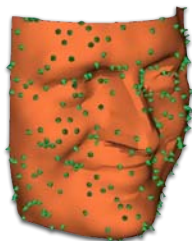


Figura 4.45: Ejemplo de *glyphing* en una imagen tridimensional

Los símbolos pueden ser simples o complejos, yendo desde conos orientados para mostrar vectores de datos, hasta otros *glyphs* bastante más complejos. En VTK, la clase `vtkGlyph3D` permite crear *glyphs* que pueden ser escalados, coloreados y orientados a lo largo de una dirección. Los *glyphs* son copiados en cada punto del conjunto de datos de entrada [25].

Por tanto, los *glyphs* se ven afectados por sus datos de entrada. Este objeto puede ser geométrico, un conjunto de datos o una imagen gráfica. Como se ha dicho, pueden ser orientados, escalados, trasladados, deformados o alterar de algún modo la apariencia del objeto en respuesta a los datos.

Existen representaciones más elaboradas como la técnica de Chernoff, donde se vinculan valores de datos a una representación icónica de la cara humana, las cejas, nariz, boca, y otras características son modificadas de acuerdo a los valores de los datos. Esta interesante técnica está construida sobre la capacidad humana de reconocer expresiones faciales.

Delmarcelle y Hesselink clasifican los *glyphs* de acuerdo a una de estas tres categorías:

- **Iconos elementales** que representan sus datos a lo largo de la extensión de su dominio espacial. Por ejemplo, un flecha orientada puede ser usada para representar la normal a una superficie.
- **Iconos locales** que representan información elemental más una distribución local de los valores alrededor del dominio espacial. Un vector normal a una superficie coloreada por la curvatura local es un ejemplo de icono local, desde que los datos locales detrás de la información elemental están codificados.
- **Iconos globales** que muestran la estructura del conjunto de datos completo. Una isosuperficie es un ejemplo de icono global [35].

Volviendo al método que dibuja los contornos, inicialmente se comprueba la existencia de la variable que contiene el resultado, ya que sin ella no sería posible realizar el dibujado de los bordes. En caso afirmativo, se crea una variable del tipo `vtkPoints` que servirá para almacenar los puntos en los que se han detectado los bordes. Para rellenar

esta variable se utiliza un bucle que recorre las imágenes que contienen dichas posiciones y que se encuentran dentro de la variable resultado. Dicha inserción se hace con el método `InsertNextPoint(x,y,z)` de la clase `vtkPoints`, donde (x,y,z) son las posiciones que se encuentran dentro de las imágenes unidimensionales que contienen la situación de cada vóxel borde dentro de la imagen.

A continuación se crea un objeto de la clase `vtkPolyData` para los bordes, a la que se le asigna los puntos que se han rellenado anteriormente a través del método `SetPoints`.

Posteriormente se crea un objeto *array* (`vtkDoubleArray`) y se indica que tendrá tres componentes (x,y,z) haciendo uso del método `SetNumberOfComponents`. El número total de tuplas será la dimensión de cualquiera de las imágenes unidimensionales del resultado. Ese es el valor que se le pasa a la función `SetNumberOfTuples` de la clase `vtkDoubleArray`. Después, se procede a rellenar el array. Para ello se utiliza un bucle en el que se irán recorriendo las imágenes que contienen las tres componentes de las normales (n_x, n_y, n_z - ver figura 4.41). Como para ello es preciso usar un puntero a un vector, se hace uso de un `vector_double` de `AMILab` (que no es más que el *wrapping* de un vector de `double` de la STL de C++) en el que se insertan las componentes (x,y,z) de cada normal usando `push_back`. Como lo que el método `SetTupleValue` de la clase `vtkDoubleArray` precisa es un puntero a un vector, se pasa la referencia del primer elemento del mismo.

Esas normales se le asignan al `vtkPolyData` creado anteriormente, invocando al método `SetNormals`. Así pues, en el *polydata* se tienen los puntos en los que se encuentran los contornos y, además, las normales a cada uno de los mismos.

Ahora se entra en un conjunto de comprobaciones acerca de lo que se quiere mostrar en la escena. En primer lugar se comprueba si se ha decidido mostrar el volumen del que se están calculando los contornos. Si es así, se genera una isosuperficie usando la operación `isosurf` de `AMILab` a la que se le pasa la imagen de entrada y un umbral sobre el que calcular dicha isosuperficie. Ya que dicho umbral no es conocido a priori, se calcula como la suma del máximo y el mínimo (de los valores de la imagen) dividida entre dos. Tras ello, se crea un *polydata* con la conversión de la isosuperficie a un `vtkPolyData` a través de la función `ToVtkPolyData`. Para las normales se coge como entrada el *polydata* que se acaba de computar, y se crea un `vtkPolyDataMapper`, el cuál se conecta con la salida de las normales (`mapper.SetInputConnection(&normals.GetOutputPort())`). Ahora, se crea un actor y se le asigna el *mapper*. A dicho actor se le asignará la opacidad que se haya seleccionado en la interfaz.

La siguiente comprobación tiene que ver con el dibujado de las normales. El mecanismo para dibujarlas es el mismo que se usará para los contornos, el *glyphing*, descrito anteriormente. Se crea un objeto de la clase `vtkArrowSource`, que será las flechas tridimensionales que representarán a las normales. A continuación se crea un `vtkGlyph3D` para dichas normales y se indica que estará orientado. Se asigna como fuente para dicho *glyph* la salida del `vtkArrowSource` (`glyph.SetSource(&arrowSource.GetOutput())`). Además, se indica que el modo a usar para el dibujado es el que usa las normales. Como entrada se le da el *polydata* creado en pasos previos, asignándose además un factor de escala de 0.5 a través del método `SetScaleFactor` de la clase `vtkGlyph3D`. Este factor de escala determina cuál es el tamaño de las flechas que representan a las normales. Una vez realizada esta configuración, se crea un *mapper* para la visualización, dándole como entrada el *glyph*. Sólo queda crear un actor que será agregado a la escena final. A este

actor se le asigna el valor de opacidad seleccionado y el color, mediante las operaciones `SetOpacity` y `SetColor` respectivamente. A dicho actor se le asignará el *mapper* que se ha creado.

La última de los chequeos está relacionado con mostrar los planos que representan los bordes. El esquema seguido es prácticamente el mismo que para las normales, salvo por el hecho de que en esta ocasión se toma como fuente para el *glyph* un `vtkPlaneSource` en lugar de un `vtkArrowSource`. De la misma forma, se termina creando un actor con valores propios de opacidad y color, para ser posteriormente agregado a la escena.

Cabe decir que los parámetros tanto de opacidad como color tienen valores por defecto, pero que pueden ser cambiados usando la pestaña *settings*, la cuál será descrita en el apartado de interfaz de la presente sección.

El último detalle para mostrar el resultado es crear la escena. Para ello se obtienen la ventana de renderizado a través de la operación `GetRenderWindow` realizada sobre la variable global a la clase y que representa a la misma. A esta se le añade un *renderer* previamente creado como una instancia de la clase `vtkRenderer`. Tras ello se procede a añadir los actores, previo chequeo de que se haya decidido mostrarlos (no tendría sentido añadir un actor que no ha sido creado, es más, daría error como es obvio). Esto se consigue mediante el uso de la operación `AddActor`. Finalmente se le asigna el valor del fondo al *renderer*, se resetea la cámara y se invoca a `Render()` para que muestre el resultado en pantalla.

Decir que, en caso de no existir el objeto que contiene el resultado, se muestra un mensaje por pantalla indicando que es necesario aplicar el método de detección previamente a mostrar el resultado por pantalla a través del método `vtkDrawEdges()`.

- `Reload()`: este método permite recargar la visualización a partir de nuevos parámetros que se hayan seleccionado. Eso sí, previamente es necesario que se haya ejecutado el método. Para comprobarlo se mira la existencia de la variable `::res`. Consecutivamente se va mirando si se ha decidido mostrar el volumen, los planos y las normales. En cada caso, de ser así, se le asignan las propiedades de opacidad y color a través de los métodos `SetOpacity` y `SetColor` de VTK. Además de ello se asigna el color de fondo al *render*, se resetea la cámara y se ejecuta la función `Render` del *interactor*. En caso de que se intente ejecutar esta operación y no exista la variable que contiene el resultado, se muestra un diálogo con un mensaje, indicando que previamente ha de aplicarse el método.
- `Run()`: este procedimiento se encarga de llamar a la función *wrapped*. Para ello, en primer lugar si la imagen tiene más de un canal, se normaliza. Además, si existe tanto el objeto resultado como la instancia de la clase sub-vóxel, se destruyen. Se crea un objeto de dicha clase, pasándole a dicho constructor la imagen de entrada y el umbral. Por último, se invoca a la operación correspondiente en función de la selección del método. En este caso, como actualmente sólo existe uno, se le invoca directamente. Para terminar, se llama al método `vtkDrawEdges()` de la clase que será el encargado de realizar el dibujo de los planos en la posición de los contornos.
- `Display()`: esta función se usa para mostrar la imagen directamente en el visor de AMILab.
- `MainBook_AddVolRenTab()`: este método se encarga de añadir la pestaña en la zona principal de la interfaz de AMILab. Además permite colocarle un icono a la pestaña, usando en este caso el que se ha diseñado para el algoritmo 3D.

- `MainBook_CloseTab()`: función para el cerrado de la pestaña creada con el anterior método.
- `Close()`: redefinición de la función `Close()` heredada de la clase `ScriptGui`. En esta versión se invoca además a la función anterior, para que se cierre la pestaña que contiene el visor de VTK.
- `CreateGui()`: es la operación encargada de crear la interfaz. Ésta se describe en mayor detalle a continuación.

Interfaz

La interfaz es bastante similar a la que fuera realizada para el caso de los algoritmos de detección con precisión sub-píxel (imágenes 2D). A través del método `CreateGui()` de la clase, se crea un *book* que contendrá tres pestañas, a saber: parámetros, preferencias y ayuda. Pasemos a ver cada una de ellas en mayor detalle.

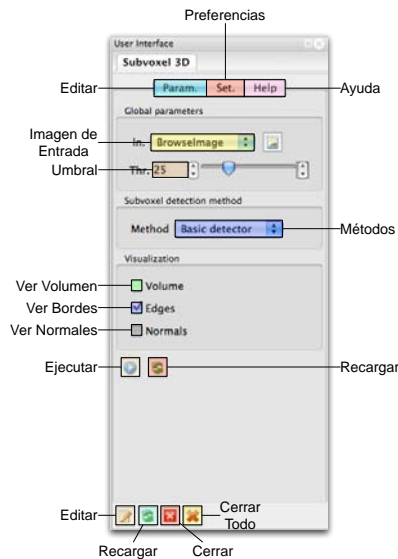


Figura 4.46: Pestaña principal

En la figura 4.46 se puede ver la pestaña de parámetros, identificada por *Param.* En ella se encontrarán las características centrales de la interfaz. En la parte superior existe un *BoxPanel* en el que están los parámetros globales: una lista desplegable donde seleccionar la imagen de entrada y un *slider* para el valor del umbral. A continuación puede seleccionarse cual es el método que se quiere usar para la detección con precisión sub-vóxel.

Además, a través de una serie de *checkboxes*, se permite al usuario decidir que es lo que quiere mostrar por pantalla: el volumen, los bordes y/o las normales. Decir que por ahora el volumen se muestra a través del cálculo de una isosuperficie de la imagen de entrada (como se dijo en la descripción del método `vtkDrawEdges()`). Lo ideal en un futuro sería integrar el dibujado de los contornos con el renderizado de la imagen de entrada, dando un mejor y más preciso resultado visual que lo actual.

Finalmente se cuenta con los botones de ejecutar y recargar. Al igual que en anteriores *scripts*, se cuentan con los botones estándar para editar, recargar, cerrar y cerrar todo; fruto de la herencia de la clase `ScriptGui` y el uso de la función `AddStandardButtons()`.

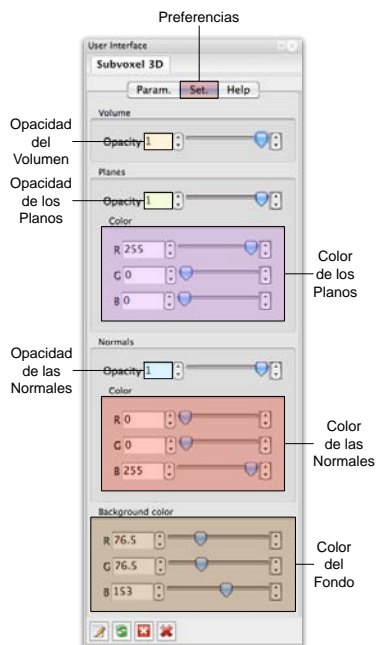


Figura 4.47: Pestaña de preferencias

El detalle de las preferencias que se pueden seleccionar relacionadas con el *script* se pueden encontrar en la figura 4.47. En ella se ve una serie de *BoxPanels* que permiten seleccionar algunas propiedades relacionadas con lo que se va a mostrar en la escena del visor de VTK. Primeramente se puede ver que es posible seleccionar la opacidad que tendrá el volumen, a través de un *slider* al uso.

Los dos grupos siguientes son muy parecidos. El primero tiene que ver con los planos y el segundo con las normales. En ambos se puede decidir tanto la opacidad como el color de cada uno de ellos respectivamente.

Por último, se puede elegir un color diferente para el fondo del visor en el *BoxPanel* denominado *Background color*.

La ayuda (figura 4.48) es parecida a la que se generó para el caso bidimensional. Al igual que entonces el texto se realiza en un fichero `LATEX`, que posteriormente será pasado a `html` mediante el uso de `LATEX2html`. Se puede ver la misma estructura, contando en la parte superior con los botones de acceso directo a la wiki de `AMILab`, la página principal de la ayuda, así como retroceder o avanzar. El menú de la ayuda se encuentra dividido en un inicio rápido y una descripción de todas y cada una de las pestañas de la interfaz.

En la figura 4.49 se puede observar el resultado obtenido al aplicar la detección sub-vóxel a imágenes sintéticas.

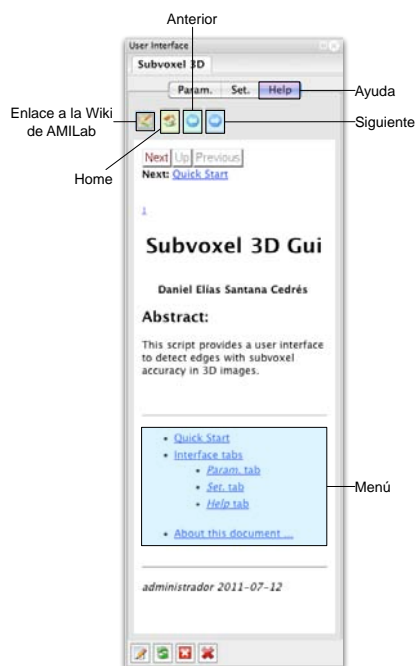


Figura 4.48: Pestaña de ayuda

En la figuras 4.50, 4.51, 4.52 y 4.54; se puede ver el resultado de la aplicación en imágenes reales.

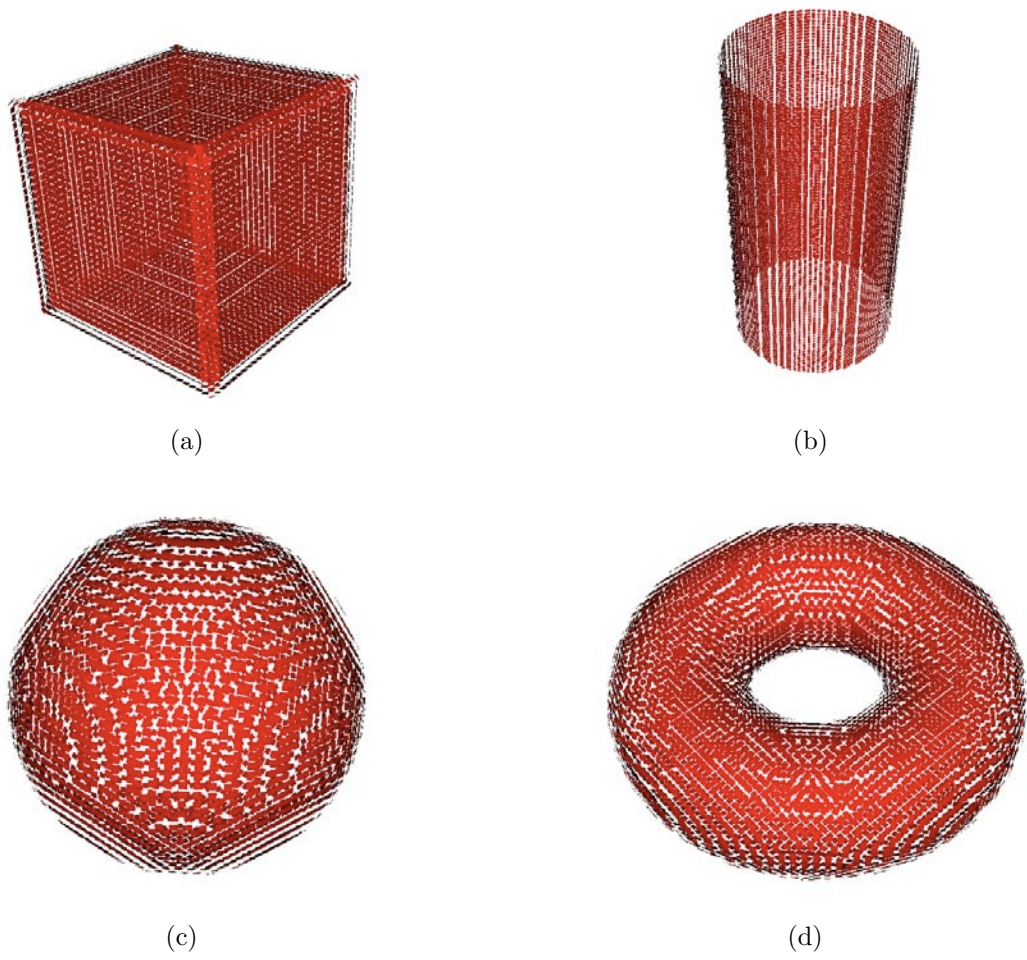


Figura 4.49: Resultado de la detección sub-vóxel en imágenes sintéticas: (a)Cubo, (b)cilindro, (c)esfera y (d)toro

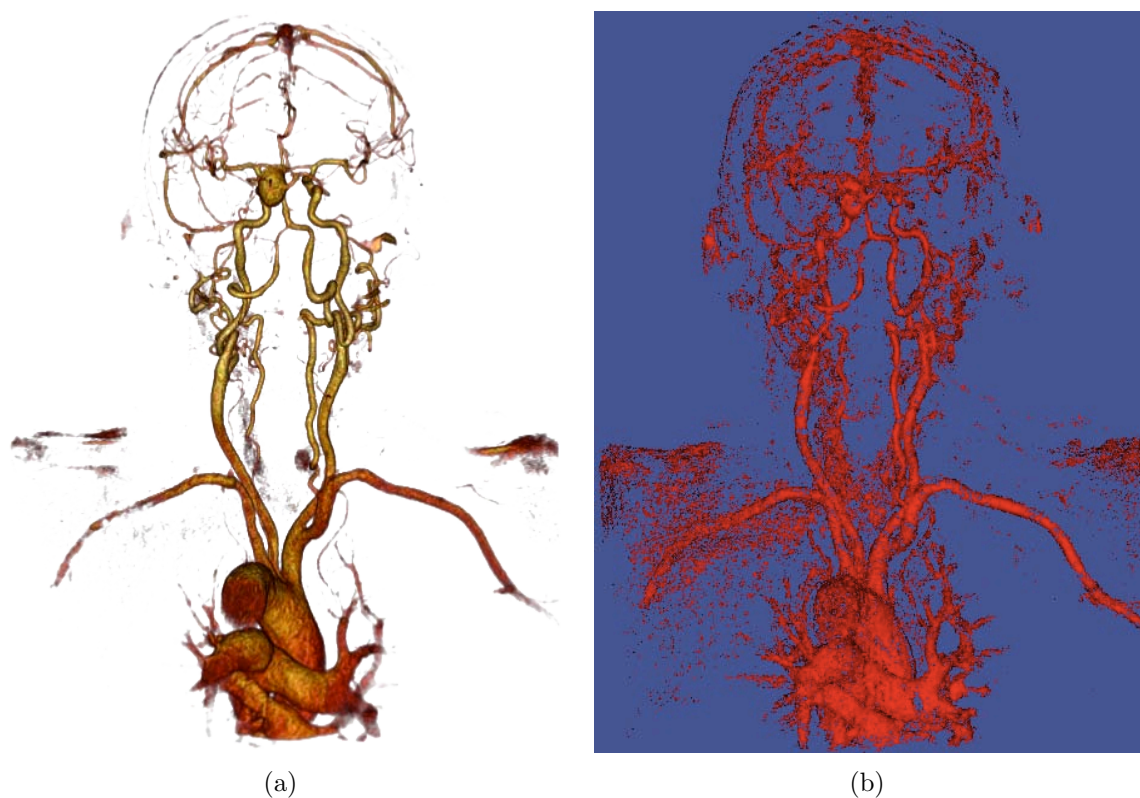


Figura 4.50: Resultado de la detección sub-vóxel en una tomografía computarizada de la parte superior del cuerpo (cabeza, cuello y hombros, hasta aproximadamente el llamado aórtico): (a)Renderizado volumétrico y (b)detección sub-vóxel

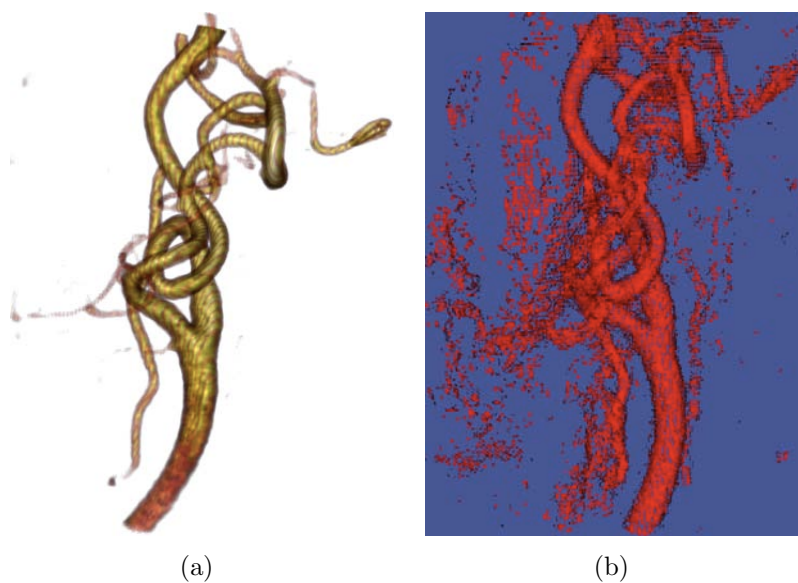


Figura 4.51: Resultado de la detección sub-vóxel en una tomografía computarizada de la arteria carótida: (a)Renderizado volumétrico y (b)detección sub-vóxel

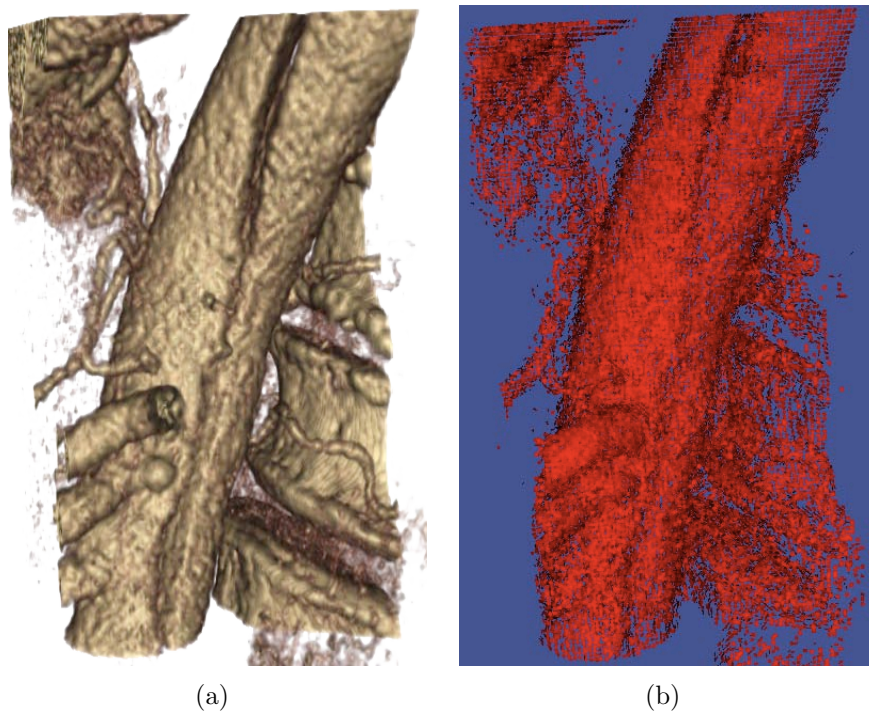


Figura 4.52: Resultado de la detección sub-vóxel en una tomografía computarizada de una disección aórtica: (a) Renderizado volumétrico y (b) detección sub-vóxel

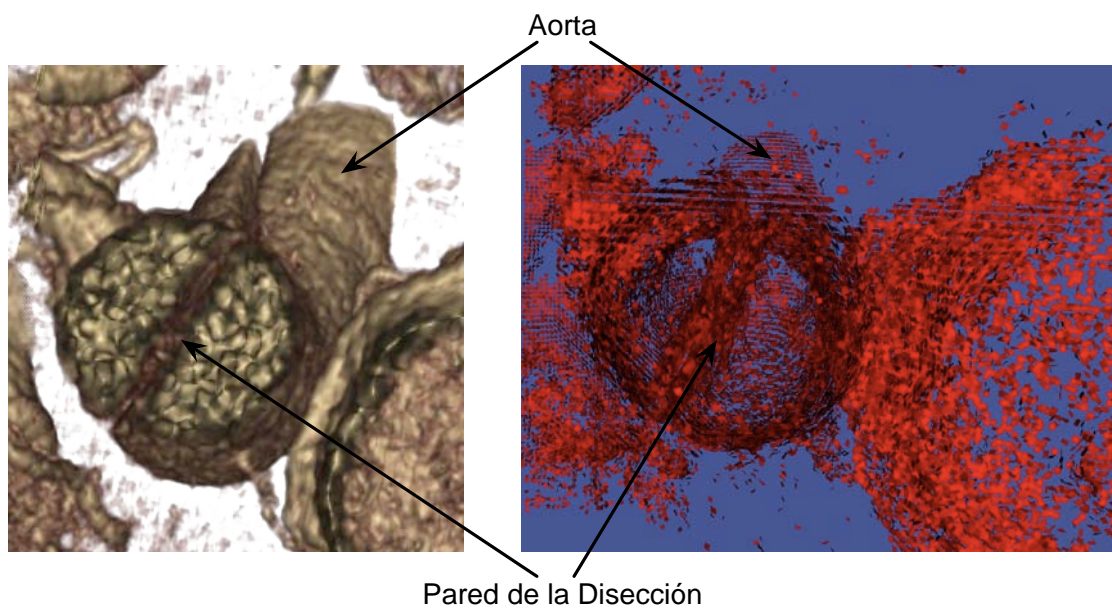


Figura 4.53: Resultado de la detección sub-vóxel en una tomografía computarizada de una disección aórtica. Vista axial.

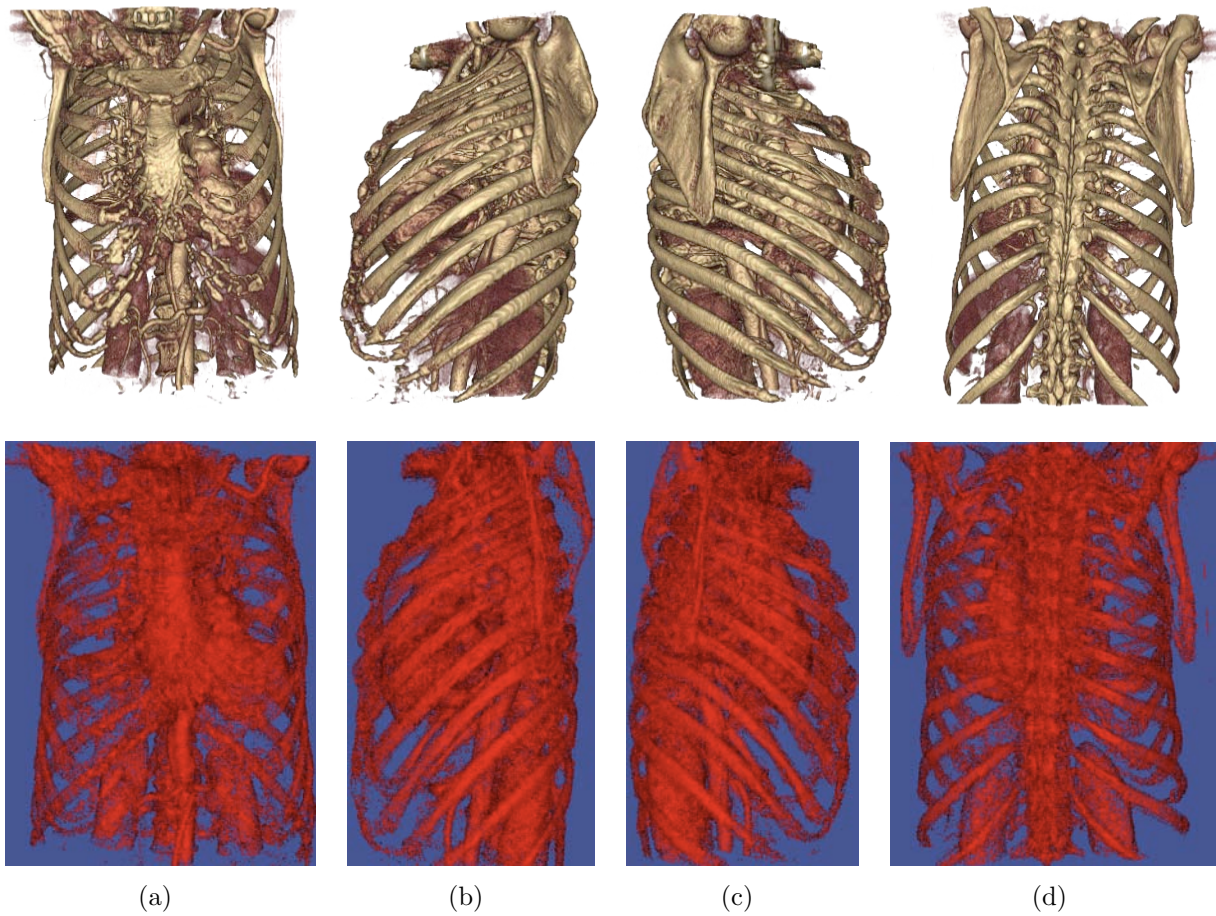


Figura 4.54: Resultado de la detección sub-vóxel en una tomografía computarizada del tronco: (a)Parte frontal, (b)perfil izquierdo, (c)perfil derecho y (d)espalda

4.3. Imágenes Sintéticas

El proceso de generación de imágenes sintéticas con vistas a poder realizar las pruebas, es un paso fundamental, puesto que dichas imágenes proporcionarán parámetros que podrán ser medidos, con el fin de estimar la precisión de los métodos. La idea principal de la generación de estas imágenes de prueba es poder generarlas a partir de la definición de una función analítica.

Este método de creación hace uso del código existente para la creación de imágenes mediante estimación por interpolación lineal existente ya en AMILab. Existen dos métodos uno iterativo y otro recursivo. Se plantea hacer uso del segundo, ya que sólo realiza la subdivisión en la zona en la que se encuentran conflictos (es decir, se encuentran valores positivos y negativos). Es zona conflictiva serán aquellos píxeles/vóxeles por los que pasa el borde. Para poder conseguir este objetivo es necesario redefinir la operación para que haga uso de, en lugar de la interpolación lineal, de la función analítica que describe la imagen que se quiere representar (en concreto, el cálculo del valor de un punto para esa función). Para mayor detalle véase la figura 4.55.

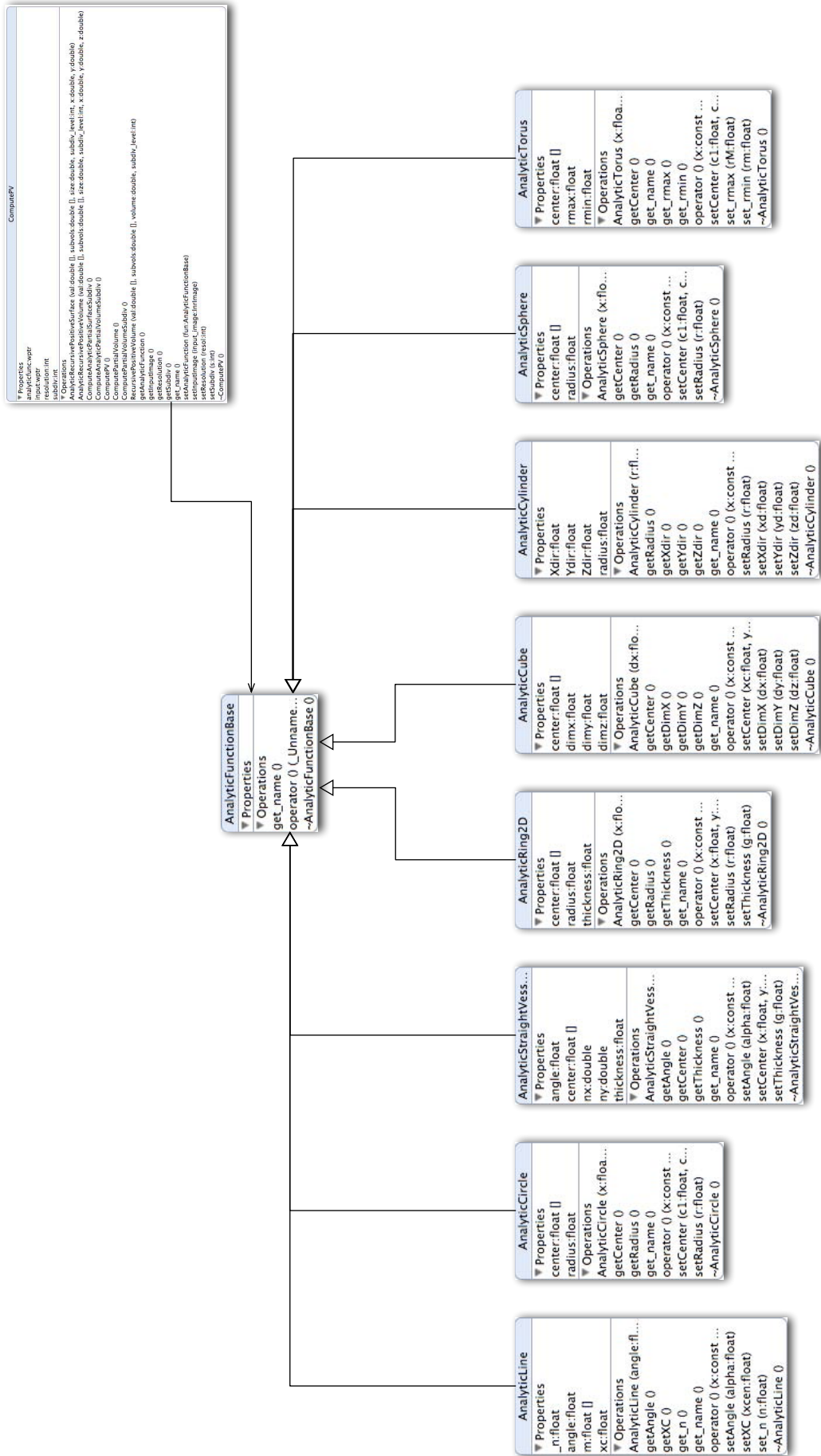


Figura 4.55: Diagrama de clases de la generación de imágenes sintéticas

Las funciones a representar se basan en una estructura de tipo jerárquico, en la que la clase base (denominada `AnalyticFunctionBase`) es sólo la definición virtual de una sobrecarga del operador paréntesis. Éste recibe tres parámetros de tipo `double` para el cómputo del valor del punto a través de la función que describe la imagen que se quiere representar. En esta superclase, el valor devuelto es cero, ya que no define ningún tipo de objeto genérico a representar, sino que sienta las bases del operador de la función que será necesario para el cómputo del efecto de volumen parcial de las diferentes imágenes. Serán pues, las clases hijas de ésta, las que definan las funciones para el cálculo del valor de un punto en la estructura que se desea representar. Aparte de ello, deberá incluir aquellos atributos que sean necesarios para la representación, así como los métodos `get` y `set` para obtener y asignar respectivamente valores a los mismos.

El diseño del cómputo del efecto parcial se aglomera en una nueva clase, denominada `ComputePV`. Ésta alberga, además de las antiguas funciones que hacen uso de la interpolación lineal, las nuevas que utilizan las funciones analíticas. Para representar esta nueva situación, se tiene como atributo de la clase un puntero a una función analítica básica (ver figura 4.55). Éste, por polimorfismo, sabrá exactamente a qué operador paréntesis debe llamar (en base a la clase heredera que representa la función). Así mismo, y para mayor simplicidad (desde el punto de vista del número de parámetros de las funciones) se tienen además como atributos a la imagen de entrada (del tipo `InrImage`), un entero que representa la subdivisión (número máximo de llamadas recursivas) y otro que indica la resolución (únicamente usado por la función iterativa que hace uso de interpolación lineal).

La adición de estos nuevos atributos no afecta a los métodos ya existentes en la clase. Es decir, los métodos de interpolación lineal no usarán para nada el puntero a la función analítica básica, puesto que no lo necesitan. Además, en el caso de funciones analíticas bidimensionales, éstas no harán uso del tercer parámetro de tipo `double` que se les pasa. No obstante, por convención y considerando la generalización (tridimensional con posiciones en el espacio dadas por (x, y, z)) que plantea la clase básica, éste tercer parámetro se mantiene, pese a que la especialización 2D no haga uso del mismo para el cálculo del valor de la función en un punto.

Así pues, una breve descripción de los métodos es la que se lista a continuación:

- `ComputeAnalyticPartialSurfaceSubdiv()`: se encarga del cálculo del efecto de volumen parcial a través de una función analítica que describe un objeto de dos dimensiones.
- `ComputeAnalyticPartialVolumeSubdiv()`: computa el efecto de volumen parcial partiendo de una función analítica que describe un objeto tridimensional.
- `ComputePartialVolume()`: calcula de manera iterativa el efecto de volumen parcial mediante el uso de interpolación lineal (sólo para 3D).
- `ComputePartialVolumeSubdiv()`: se ocupa de realizar el cálculo del efecto de volumen parcial en una imagen 3D recursivamente, a través del uso de interpolación lineal.

El procedimiento seguido para la generación de las imágenes es similar al que se puede ver en la figura 4.56. En esta imagen se intenta representar como funciona el método para el cálculo del efecto parcial. El método trata de ver qué píxeles/vóxeles están dentro o fuera, es decir, a un lado u otro de la función analítica que se va a representar. Para ello, en el caso de

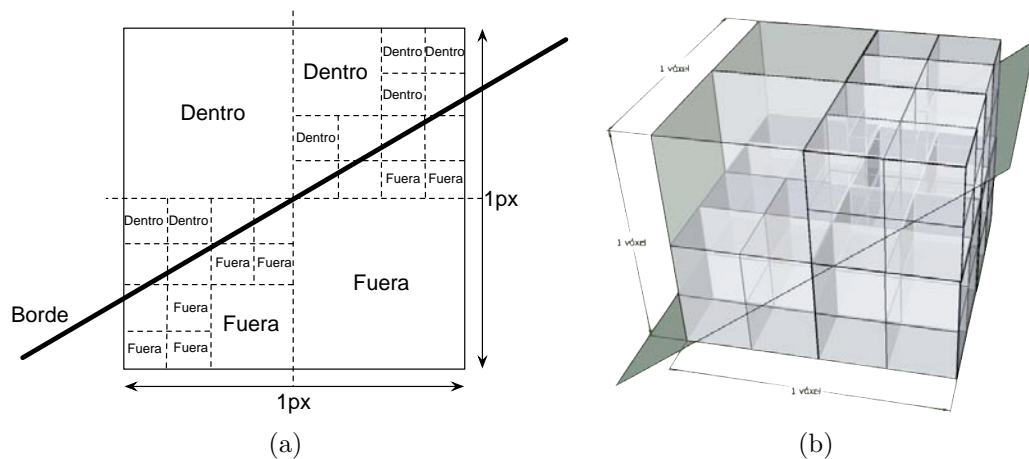


Figura 4.56: Ejemplo de subdivisión para la generación de imágenes sintéticas con efecto parcial: (a) Caso bidimensional y (b) tridimensional

que un píxel/vóxel no esté claro si pertenece a una clase u otra, se subdivide (en 4 en el caso 2D y en 8 en el caso 3D) y se vuelve a computar cada trozo por separado. La recursividad parará bien cuando se haya resuelto qué píxeles/vóxeles están a cada lado, o bien, cuando se haya alcanzado el nivel máximo de subdivisión. La intensidad del píxel/vóxel se obtendrá como $I = B + (A - B) \times p$, donde I es la intensidad que se calcula, A y B los niveles de intensidad a cada lado del borde y p es un valor entre 0 y 1, resultante del proceso de subdivisión. Nótese que si el valor de p es 0, el resultado es B , mientras que si es 1, será A . Valores intermedios de p darán también valores entre A y B .

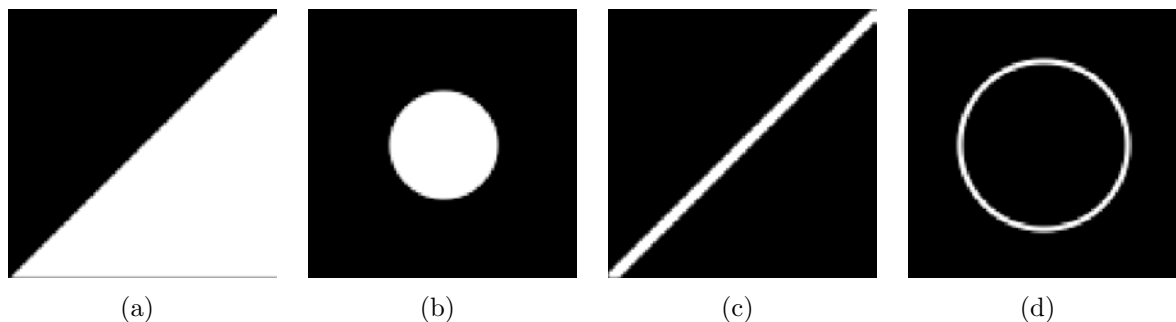


Figura 4.57: Imágenes sintéticas bidimensionales con efecto parcial: (a) Recta, (b) círculo, (c) vaso recto y (d) anillo

En la figura 4.57 se puede ver la colección de imágenes que se han desarrollado para el caso bidimensional. Entre ellas se encuentran la recta, el círculo, un vaso recto⁵ y un anillo; éstas últimas son sobre todo para probar los algoritmos capaces de detectar bordes cercanos y muy cercanos (aquellos que usan una ventana dinámica, flotante o de límites variables⁶; así como el

⁵Se denomina de esta manera porque simula un vaso sanguíneo recto en dos dimensiones

⁶Son diferentes denominaciones que se pueden encontrar a lo largo del documento. Bajo ningún concepto son métodos diferentes.

estudio de los valores de las parciales más allá de dichos límites).

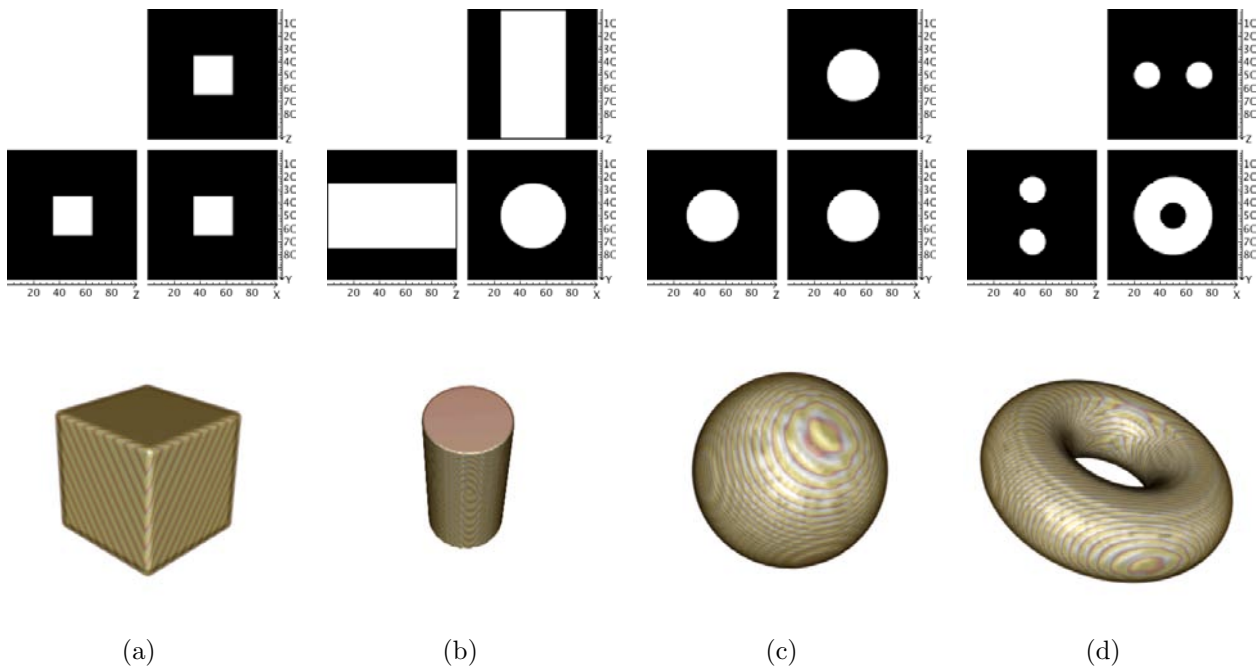


Figura 4.58: Imágenes sintéticas tridimensionales con efecto parcial: (a) Cubo, (b) cilindro, (c) esfera y (d) toro

En relación a las imágenes tridimensionales puede verse en la figura 4.58. Dentro de ellas se pueden encontrar el cubo, el cilindro, la esfera y el toro o toroide. Para mayor claridad, además de incluir la visualización del visor 2D, se incluye el renderizado volumétrico de cada una de ellas.

4.3.0.2. *Script* en AMILab

El *script* para imágenes sintéticas sigue la misma estructura que se ha visto en secciones anteriores. En la figura 4.59 se puede ver cómo está estructurado el directorio que contiene todo lo referente a la generación de imágenes con efecto parcial.

Dentro de dicha carpeta se puede encontrar el directorio “Doc”. Éste contiene el fichero \LaTeX con el nombre `SyntheticPVEImages.tex`, así como la carpeta del mismo nombre. Ésta se ha obtenido de aplicar \LaTeX 2html al fichero \LaTeX . Todo ello conforma la documentación que se puede encontrar en la pestaña de ayuda de la interfaz del *script* (descrita posteriormente).

Además del directorio de ayuda, están los *scripts* de la clase y que cargan la interfaz (`SyntheticPVEImagesClass.amil` y `SyntheticPVEImagesGui.amil` respectivamente), el *script* que añade la funcionalidad al menú y a la barra de herramientas (`Add2Menu.amil`), así como los iconos 16×16 y 32×32 que aparecerán en el menú y en la barra de herramientas respectivamente.

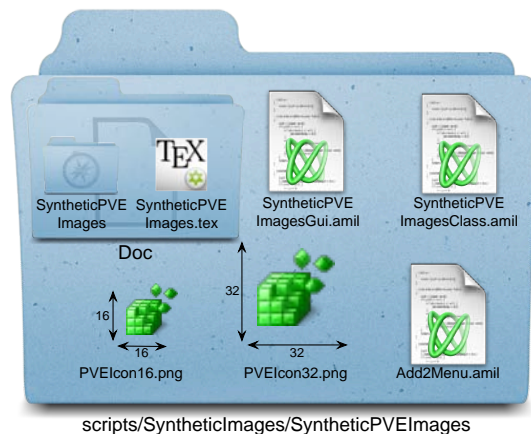


Figura 4.59: Estructura del directorio del *script* para la generación de imágenes sintéticas



Figura 4.60: Diseño del icono que aparece en la barra de menú y la barra de herramientas de AMILab para el algoritmo de generación de imágenes sintéticas

En el diseño del icono (fig. 4.60) se ha partido de la imagen de un cubo compuesto por pequeños cubitos, que se encuentra presente en la colección libre que está dentro del directorio de *scripts*. Se ha intentado reflejar que se trata de una imagen compuesta por pequeños vóxeles, que cambian su intensidad de izquierda a derecha en la imagen, dando la sensación de efecto parcial.

En la figura 4.61 se puede ver la estructura de clases del *script* para imágenes con efecto parcial. La estructura es la misma que la vista en casos anteriores, con un herencia de la clase `ScriptGui` por parte de la clase `SyntheticPVEImagesClass`, que provee las características mínimas de interfaz. Asimismo, aparece un objeto de estereotipo fichero que es el encargado de crear un objeto de clase y de invocar a la creación de la interfaz (`SyntheticPVEImagesGui`).

Dentro de los atributos de la clase `SyntheticPVEImagesClass` se encuentran: el tipo de función analítica 2D o 3D seleccionada, el método seleccionado para el cálculo del efecto parcial, el nivel de subdivisión y el nombre de la salida. Posteriormente los atributos se agrupan según cuál sea su uso. En primer lugar se encuentran un conjunto de variables *booleanas* que se van a usar para indicar si se ha dicho que se desea recargar el cómputo de la función analítica mientras se cambian sus parámetros. Como se tienen ocho funciones diferentes se tendrán, por tanto, ocho de estas variables. A continuación aparecen tres variables para el tamaño de cada dimensión

(x, y, z) de la imagen de entrada, que será aquella sobre la que se calcule el resultado. Después aparecen agrupadas las variables asociadas a cada una de las funciones (círculo, rampa, vaso recto, anillo, cubo, cilindro, esfera y toro), que básicamente son las características que pueden variarse; como por ejemplo el radio o el centro. Tras estos, aparecen las dos variables asociadas a la ayuda, tanto la que tiene el enlace a la wiki de **AMILab**, como la que apunta al directorio donde se encuentra la documentación **html**. Finalmente, se encuentran dos atributos que almacenan los valores de las intensidades de la imagen.

En relación a los métodos se tienen los siguientes:

- **NewType2D()**: este método genera una variable de tipo **VarVector** en la cual se introducen todas las pestañas del *book* de funciones analíticas 2D. Posteriormente utiliza dicha variable para dejar activa solamente la pestaña que ha sido seleccionada, es decir, la función analítica 2D que se ha decidido generar.
- **NewType3D()**: análoga a la función anterior, pero en este caso para las funciones analíticas asociadas a imágenes tridimensionales.
- **CreateInputImage()**: a partir de las dimensiones indicadas en la interfaz para cada una de las dimensiones de la imagen de entrada, este procedimiento genera la misma. Además, una vez que se ha creado, se habilita la pestaña de las funciones analíticas, para proceder a seleccionar una.
- **ResetInputImage()**: devuelve a los valores por defecto las dimensiones de la imagen de entrada ($x = 100$, $y = 100$, $z = 100$).
- **Compute()**: este método es el encargado de invocar a las funciones del cálculo del efecto parcial. Para ello, en primer lugar se crea una instancia de la clase **ComputePV**. En ese momento se comprueba si se ha seleccionado en la interfaz la función para calcular de manera recursiva el efecto parcial en una imagen 2D. Siendo así, se comprueba además si la imagen de entrada es bidimensional (en caso contrario se muestra un cuadro de diálogo que indica que esta función es solamente para aplicarla en imágenes bidimensionales). Pasados los chequeos, se la asigna la imagen de entrada a la variable de la clase **ComputePV**, así como el nivel de subdivisiones. En función del tipo de función analítica que se haya seleccionado, se asigna el objeto de la misma y se invoca al método **ComputeAnalyticPartialSurfaceSubdiv** con los valores de intensidad que también han sido seleccionados en la interfaz. En cada caso, se le da valor al atributo **output_name** apropiadamente (circle, si es un círculo, line si es la rampa, etc.). De esta forma, cuando se vaya a guardar la imagen resultante, ésta ya tendrá un nombre significativo, aunque siempre el usuario puede decidir cambiarlo.

Si por el contrario el método seleccionado es aquel para calcular de forma recursiva el efecto parcial en un volumen, se procede de manera similar, pero en este caso se controla que se haya creado alguna de las funciones analíticas tridimensionales disponibles. Al igual que antes, se asigna al objeto de la clase **ComputePV** la imagen de entrada y los niveles de subdivisión. Atendiendo al tipo de función seleccionada, se asignará una u otra a dicho objeto y se invocará al método **ComputeAnalyticPartialVolumeSubdiv** con los valores de intensidad indicados. Asimismo, nuevamente se le asigna un nombre significativo a través del atributo **output_name** de la clase.

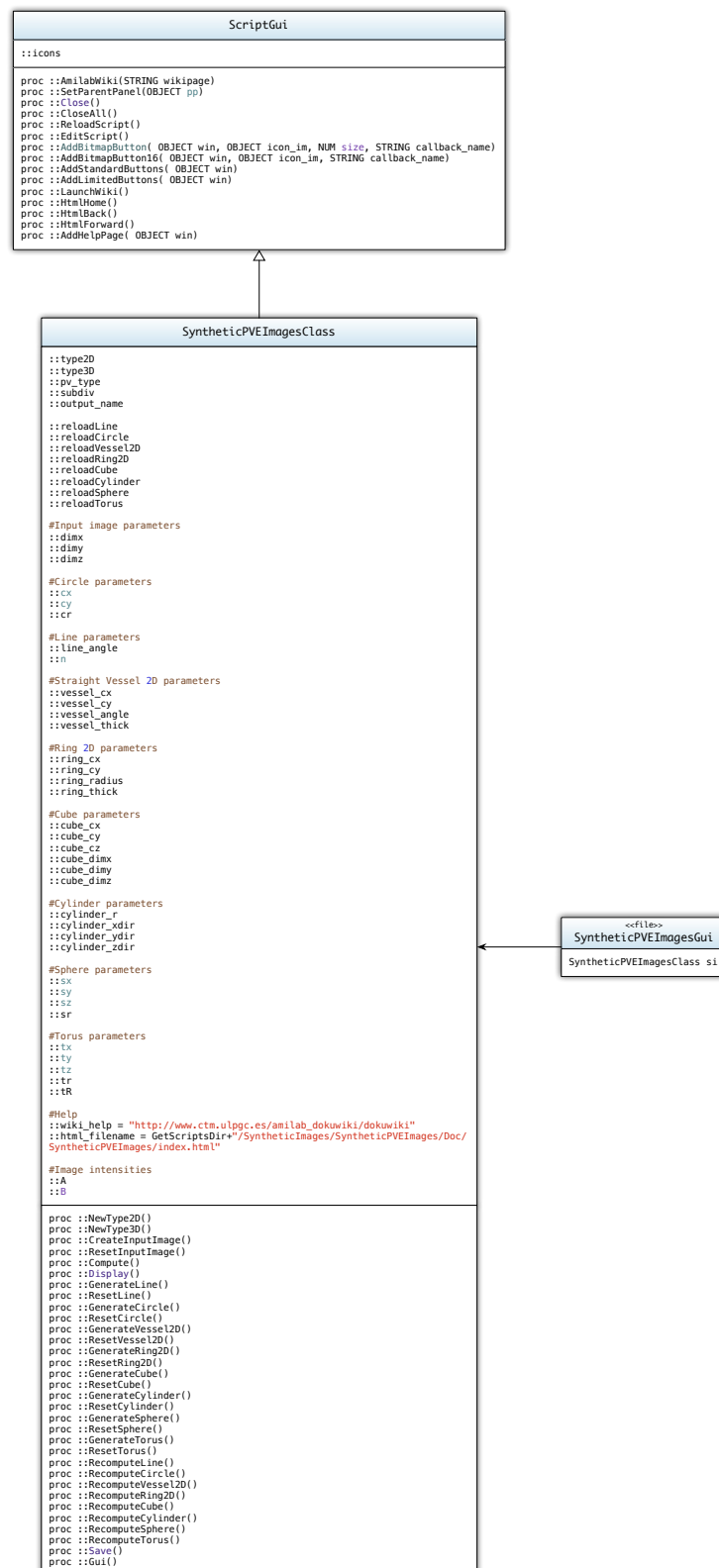


Figura 4.61: Diagrama de clases del *script* para la generación de imágenes sintéticas con efecto de volumen parcial

- `Display()`: esta función muestra el resultado del cómputo de la imagen con efecto de volumen parcial por pantalla, siempre y cuando la misma exista. En caso contrario, se muestra un cuadro diálogo con el mensaje que indica que la imagen resultado no existe.
- `GenerateLine()`: crea una instancia de la clase `AnalyticLine`, asignándole además el ángulo de inclinación y el punto de corte con el eje y .
- `ResetLine()`: devuelve los parámetros de la rampa a sus valores por defecto. Si se encuentra activo el *checkbox* de recargar, invoca al método `RecomputeLine()`.
- `GenerateCircle()`: crea una instancia de la clase `AnalyticCircle`, asignándole los valores para el centro y el radio de la circunferencia.
- `ResetCircle()`: resetea los valores de los parámetros del círculo. Si está activo el *checkbox* de recargar, invoca a la función `RecomputeCircle()`.
- `GenerateVessel2D()`: crea una instancia de la clase `AnalyticStraightVessel2D`, asignándole los valores del centro, el ángulo de inclinación y el grosor del vaso.
- `ResetVessel2D()`: pone los parámetros del vaso a sus valores por defecto. Si se encuentra activo el *checkbox* de recargar, invoca al procedimiento `RecomputeVessel2D()`.
- `GenerateRing2D()`: crea una instancia de la clase `AnalyticRing2D`, asignándole los valores para el centro, el radio y el grosor del anillo.
- `ResetRing2D()`: devuelve los parámetros del anillo a sus valores por defecto. Asimismo, si se encuentra activo el *checkbox* de recargar, invoca a `RecomputeRing2D()`.
- `GenerateCube()`: crea una instancia de la clase `AnalyticCube()` y le asigna los valores del centro y los tamaños para cada dimensión (x, y, z) .
- `ResetCube()`: resetea los valores de los parámetros del cubo a los de por defecto. Si se encuentra activo el *checkbox* de recargar, se invoca al método `RecomputeCube()`.
- `GenerateCylinder()`: crea una instancia de la clase `AnalyticCylinder()`, asignándole los valores del radio y las direcciones.
- `ResetCylinder()`: pone los parámetros del cilindro a sus valores por defecto. Si se encuentra activa la opción de recargar llama a `RecomputeCylinder()`.
- `GenerateSphere()`: crea una instancia de la clase `AnalyticSphere`, asignándole los valores para el centro y el radio de la esfera.
- `ResetSphere()`: devuelve a los parámetros de la esfera a sus valores por defecto. Si se encuentra activo el *checkbox* de recargar, invoca a la función `RecomputeSphere()`.
- `GenerateTorus()`: crea una instancia de la clase `AnalyticTorus`, dándole valores al centro y a los radios mínimo y máximo.
- `ResetTorus()`: resetea los valores de los parámetros del toro a los de por defecto. Si está activo el *checkbox* de recargar, invoca a `RecomputeTorus()`.
- `RecomputeLine()`: si está activa la opción de recargar la rampa y existe una instancia de dicha función analítica, se asignan los valores para el ángulo de inclinación y punto de corte con el eje y . Si existe el objeto del tipo `ComputePV`, asigna la función analítica cuyos parámetros se acaban de cambiar, invoca al método `Compute()` y muestra el resultado por pantalla con `Display()`.

- `RecomputeCircle()`: si está activa la opción de recargar el círculo y existe una instancia de dicha función analítica, se asignan los valores para el centro y el radio. Si existe el objeto del tipo `ComputePV`, asigna la función analítica cuyos parámetros se acaban de cambiar, invoca al método `Compute()` y muestra el resultado por pantalla con `Display()`.
- `RecomputeVessel2D()`: si está activa la opción de recargar el vaso y existe una instancia de dicha función analítica, se asignan los valores para el centro, el ángulo de inclinación y el grosor del vaso. Si existe el objeto del tipo `ComputePV`, asigna la función analítica cuyos parámetros se acaban de cambiar, invoca al método `Compute()` y muestra el resultado por pantalla con `Display()`.
- `RecomputeRing2D()`: si está activa la opción de recargar el anillo y existe una instancia de dicha función analítica, se asignan los valores para el centro, el radio y el grosor del anillo. Si existe el objeto del tipo `ComputePV`, asigna la función analítica cuyos parámetros se acaban de cambiar, invoca al método `Compute()` y muestra el resultado por pantalla con `Display()`.
- `RecomputeCube()`: si está activa la opción de recargar el cubo y existe una instancia de dicha función analítica, se asignan los valores para el centro y los tamaños de cada dimensión (x, y, z). Si existe el objeto del tipo `ComputePV`, asigna la función analítica cuyos parámetros se acaban de cambiar, invoca al método `Compute()` y muestra el resultado por pantalla con `Display()`.
- `RecomputeCylinder()`: si está activa la opción de recargar el cilindro y existe una instancia de dicha función analítica, se asignan los valores para el radio y las direcciones. Si existe el objeto del tipo `ComputePV`, asigna la función analítica cuyos parámetros se acaban de cambiar, invoca al método `Compute()` y muestra el resultado por pantalla con `Display()`.
- `RecomputeSphere()`: si está activa la opción de recargar la esfera y existe una instancia de dicha función analítica, se asignan los valores para el centro y radio. Si existe el objeto del tipo `ComputePV`, asigna la función analítica cuyos parámetros se acaban de cambiar, invoca al método `Compute()` y muestra el resultado por pantalla con `Display()`.
- `RecomputeTorus()`: si está activa la opción de recargar el toro y existe una instancia de dicha función analítica, se asignan los valores para el centro y los radios mínimo y máximo. Si existe el objeto del tipo `ComputePV`, asigna la función analítica cuyos parámetros se acaban de cambiar, invoca al método `Compute()` y muestra el resultado por pantalla con `Display()`.
- `Save()`: guarda la imagen resultado como variable global para un posterior uso (por ejemplo, para aplicar la detección con precisión sub-píxel o sub-vóxel).
- `Gui()`: este método se encarga de crear la interfaz, la cual será descrita a continuación.

Interfaz

Como se ha dicho con anterioridad, la interfaz es generada a través del método `Gui` de la clase `SyntheticPVEImagesClass`. Al igual que en las otras interfaces, se cuenta con un *book* principal sobre el que irán las diferentes pestañas. Así pues, se mostrarán las capturas de todas las partes de la interfaz para facilitar su explicación.

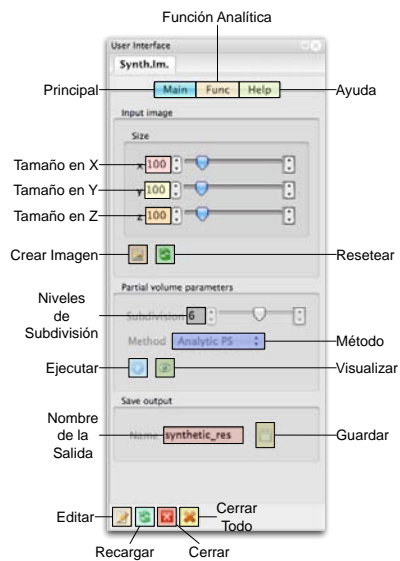


Figura 4.62: Pestaña principal de la interfaz para la generación de imágenes sintéticas

En la figura 4.62 se puede ver la interfaz de la pestaña principal del *script*. En la parte superior se encuentra un *BoxPanel* donde aparecen las dimensiones de la imagen de entrada. Debajo del mismo está el botón para crear la imagen así como para resetear las dimensiones a los valores por defecto.

Posteriormente, aparecen los parámetros para el cómputo del efecto de volumen parcial. Éstos son los niveles de subdivisión (que puede ser cambiado a través de un *slider*) y el método que se desea usar (dependiente de si es una imagen 2D o 3D). Bajo estos parámetros se encuentran los botones para ejecutar el cómputo y para visualizar el resultado.

Como se comentó durante la descripción de los métodos de la clase, es posible guardar el resultado obtenido. Para ello se cuenta en la parte inferior con un pequeño diálogo para ello, donde se puede seleccionar un nombre y darle al botón para guardar. Nótese que, como se ha descrito, este método de guardado almacena la imagen como variable global de *AMILab*, no en disco.

Finalmente, en la zona inferior, al igual que en otros *scripts* aparecen los botones estándar para editar, recargar, cerrar o cerrar todo; obtenidos a partir de la herencia de la clase *ScriptGui* y del uso del método *AddStandardButtons*.

La siguiente pestaña está dedicada a las funciones analíticas, divididas en bidimensionales y tridimensionales. Para una mayor sencillez se agruparán las imágenes que las muestran, en esas mismas categorías.

En la figura 4.63 se pueden observar las interfaces para las diferentes funciones analíticas con las que se cuenta en el *script* para imágenes bidimensionales: rampa, círculo, vaso y anillo.

De forma semejante sucede con las interfaces de las funciones analíticas tridimensionales que se encuentran en la figura 4.64. En ella se encuentran las funciones para generar las siguientes

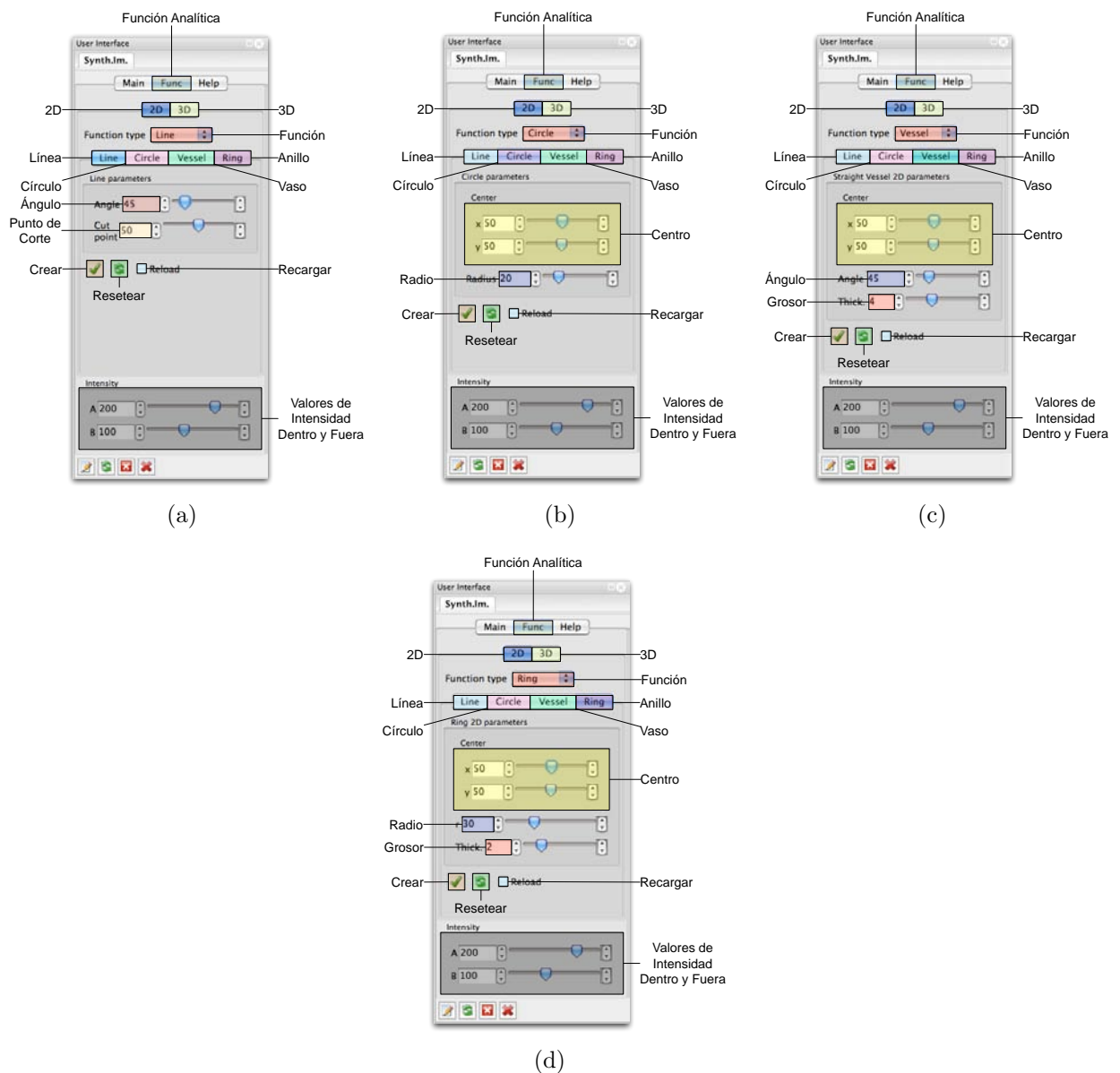


Figura 4.63: Interfaces para las funciones analíticas de imágenes bidimensionales: (a) Rampa, (b) círculo, (c) vaso y (d) anillo

figuras: cubo, cilindro, esfera y toro.

Tanto en un caso como el otro, cada una de las interfaces permite cambiar los principales parámetros para generar las diferentes funciones. Para generar finalmente el objeto asociado, es preciso darle al botón inferior izquierdo con el *bitmap* de aplicar (similar a una letra “v”). El botón a su derecha, de resetear, permite devolver todos los parámetros a sus valores por defecto. En el caso en que, en cualquiera de las funciones, se encuentre activo el *checkbox* de recargar, cada vez que se varíe alguno de los parámetros se recomputará el efecto parcial y se mostrará el resultado por pantalla.

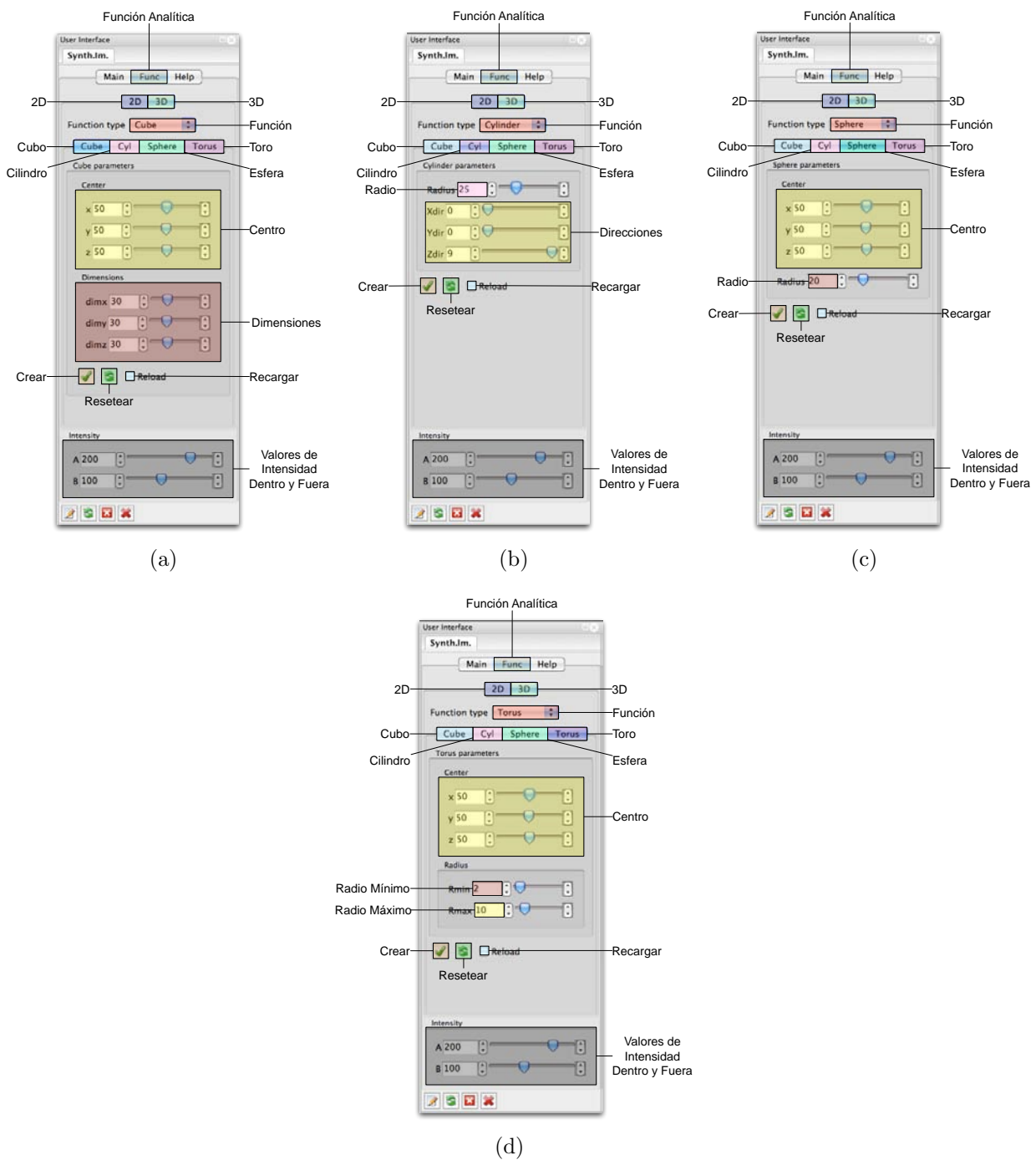


Figura 4.64: Interfaces para las funciones analíticas de imágenes tridimensionales: (a) Cubo, (b) cilindro, (c) esfera y (d) toro

Por último, en la figura 4.65, se puede ver la pestaña de ayuda del *script* para la generación de imágenes sintéticas con efecto de volumen parcial. En la misma se puede ver el contenido que se generó a partir del documento \LaTeX usando \LaTeX2html . En la parte superior se encuentran los botones para acceder a la wiki de AMILab, a la página principal de la ayuda y para retroceder y avanzar dentro de ella. Dentro del menú están los enlaces a la descripción de las funcionalidades

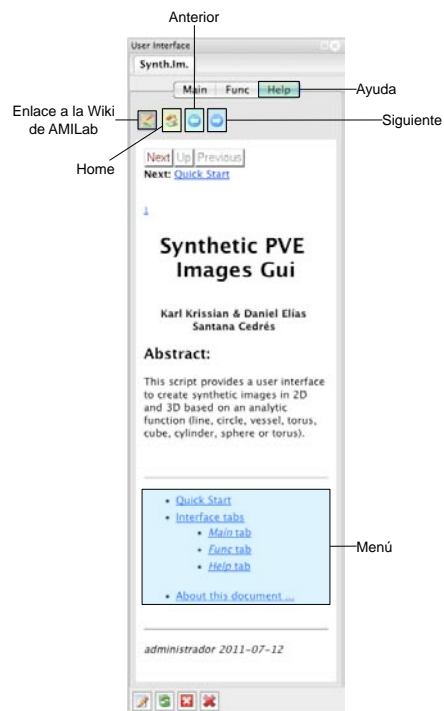


Figura 4.65: Pestaña de ayuda del *script* para la generación de imágenes sintéticas

de cada una de las pestañas.

Capítulo 5

Resultados

lega el momento de evaluar el trabajo realizado y descrito en los capítulos anteriores. Para ello, en este capítulo se tratarán tres temas principales. En primer lugar se comienza por estudiar cuál es la influencia del nivel de subdivisión en la generación de imágenes sintéticas, tanto bidimensionales como tridimensionales. Para ello se toman principalmente dos parámetros de referencia: el tiempo de ejecución y el valor de intensidad obtenido. Con ello se plantea una solución a medio camino entre un tiempo razonable y un valor de intensidad cercano al ideal.

La sección siguiente trata de evaluar la efectividad de los métodos de alta precisión. Para ello se estudian en particular dos características: el valor del salto de intensidad entre ambos lados del borde y la orientación. Con ello se compara los métodos tradicionales con los nuevos que han sido incorporados.

Por último existe una sección dedicada a una descripción breve de un caso práctico, fruto de la participación en la conferencia EUROCAST.

5.1. Estudio de la Generación de Imágenes Sintéticas

Antes de realizar las diferentes comparativas de los métodos de detección, es importante estudiar cuál sería el nivel de subdivisión óptimo para la generación de imágenes sintéticas. En esta sección se realizará dicho análisis, considerando los tiempos de ejecución y valor de intensidad obtenido para cada uno de los niveles de subdivisión. Con ello se tratará de tomar una solución de compromiso entre el tiempo de ejecución necesario para la creación de la imagen y la exactitud del valor de intensidad obtenido respecto al estimado.

5.1.1. 2D

Para analizar el caso de imágenes bidimensionales se va a considerar el siguiente caso: se va a generar una imagen sintética 2D con un borde recto de lado a lado, que separa dos regiones de intensidad 100 y 200 respectivamente, sobre una imagen de entrada de dimensiones 100×100 píxeles. Teniendo en cuenta esto, el valor estimado para el píxel borde con efecto parcial que separa ambas zonas debería ser teóricamente de 150.

En la tabla 5.1, así como en la figura 5.1 se puede observar el resultado de aplicar diversos niveles de subdivisión para la generación de la imagen sintética.

| Subdivisiones | Tiempo (seg) | Intensidad |
|---------------|--------------|------------|
| 1 | 0,00 | 175 |
| 2 | 0,00 | 162,5 |
| 3 | 0,00 | 156,25 |
| 4 | 0,00 | 153,125 |
| 5 | 0,01 | 150,781 |
| 6 | 0,01 | 150,781 |
| 7 | 0,01 | 150,391 |
| 8 | 0,01 | 150,195 |
| 9 | 0,02 | 150,098 |
| 10 | 0,04 | 150,049 |
| 11 | 0,08 | 150,024 |
| 12 | 0,14 | 150,012 |
| 13 | 0,27 | 150,006 |
| 14 | 0,52 | 150,003 |
| 15 | 1,09 | 150,002 |
| 16 | 2,09 | 150,001 |
| 17 | 4,14 | 150 |

Cuadro 5.1: Resultados para la generación de una imagen sintética 2D de una recta

Se parte de la hipótesis de que se desea alcanzar un nivel de subdivisión tal que el valor de intensidad obtenido para el píxel borde es el ideal. Se puede ver como el número total de experimentos realizados es de 17, hasta alcanzar dicha condición. No obstante, esto requiere un coste temporal de 4,14 segundos.

Atendiendo a los resultados obtenidos se puede concluir que un buen valor para el nivel de subdivisión en imágenes bidimensionales, tal que el valor de intensidad sea aproximado al ideal y cuyo tiempo no sea excesivo, podría ser de 9. Con 9 niveles de subdivisión se necesita invertir un tiempo de dos centésimas de segundo, obteniendo un valor de intensidad que se aproxima al ideal hasta las décimas (150,098).

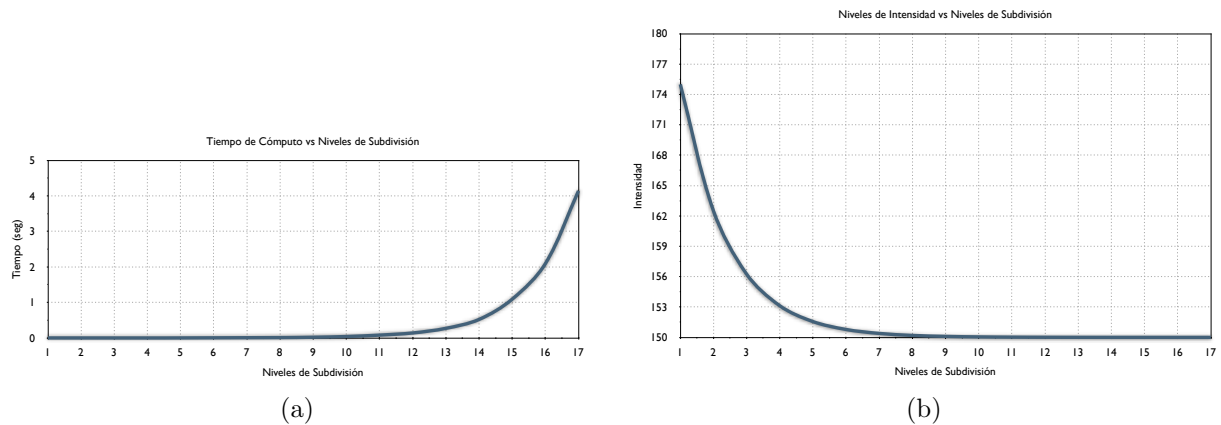


Figura 5.1: Gráficas de la generación de la imagen de una rampa 2D de 100×100 píxeles: (a)Tiempo frente a niveles de subdivisión y (b)intensidad frente a niveles de subdivisión

5.1.2. 3D

En relación a la generación de imágenes tridimensionales, se ha considerado estudiar la generación de una imagen de dimensiones $100 \times 100 \times 100$ vóxeles, dentro de la cual se va a crear un cubo centrado en el punto $(50, 50, 50)$ y de tamaño 30 en todas las dimensiones (x, y, z) . El interior del cubo tendrá un valor de intensidad 200, mientras que el exterior será 100. Al igual que en el caso anterior, se estima que el valor ideal para un vóxel borde que separa ambas regiones ha de ser de 150.

En la tabla 5.2, así como en la representación gráfica de la figura 5.2, se pueden ver los resultados obtenidos tanto en tiempo como en valores de intensidad para diferentes niveles de subdivisión.

| Subdivisiones | Tiempo (seg) | Intensidad |
|---------------|--------------|------------|
| 1 | 0,31 | 175 |
| 2 | 0,34 | 162,5 |
| 3 | 0,42 | 156,25 |
| 4 | 0,75 | 153,125 |
| 5 | 2,10 | 151,562 |
| 6 | 7,43 | 150,781 |
| 7 | 28,86 | 150,391 |
| 8 | 115 | 150,195 |
| 9 | 458 | 150,098 |
| 10 | 1837 | 150,049 |

Cuadro 5.2: Resultados para la generación de una imagen sintética 3D de un cubo

Se parte de la misma hipótesis inicial descrita en la sección anterior, es decir, llegar al número de subdivisiones necesarias hasta alcanzar el valor ideal de intensidad estimado para

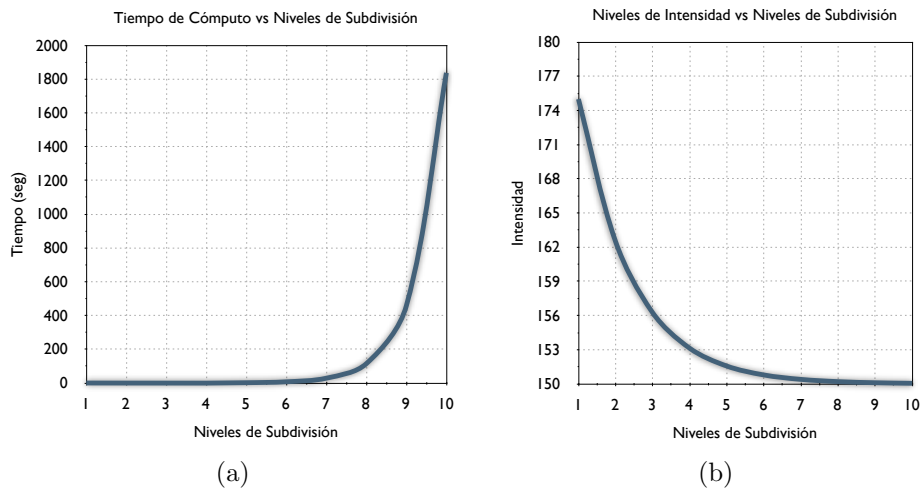


Figura 5.2: Gráficas de la generación de la imagen de un cubo 3D de $100 \times 100 \times 100$ vóxeles: (a)Tiempo frente a niveles de subdivisión y (b)intensidad frente a niveles de subdivisión

un vóxel borde. No obstante, debido a los tiempos de ejecución obtenidos y, al contrario que en el experimento anterior, el número de muestras se detuvo en 10; puesto que era necesario mucho tiempo de cómputo.

Teniendo en cuenta los resultados obtenidos y viendo que yendo más allá de los 7 niveles de subdivisión el tiempo de cálculo es bastante alto (con 10 niveles son precisos más de 30 minutos para obtener el resultado), una buena elección para este parámetro que conlleve un valor aceptable de la intensidad podría ser de 6, ya que mantiene un tiempo más razonable que el resto y el valor de intensidad es correcto al menos en su parte entera.

5.2. Estudio Comparativo

En la tabla 5.3 se muestran los errores cometidos en el módulo (unidades de intensidad) y dirección (grados) usando el método tradicional y el método sub-píxel, al aplicar la detección en una imagen sintética de una circunferencia de radio 20, con intensidad 255 en el interior y 0 en el exterior.

| Método | Error del módulo | | | | Error de la dirección | | | |
|----------------------|------------------|-------|------|-------|-----------------------|-------|-----|------|
| | Media | Desv. | Mín. | Máx | Media | Desv. | Mín | Máx. |
| Tradic. $\alpha = 0$ | 45,7 | 34,4 | 0,0 | 101,8 | 7,1 | 3,7 | 0,0 | 14,7 |
| Tradic. $\alpha = 0$ | 6,3 | 5,3 | 0,0 | 17,2 | 0,8 | 0,6 | 0,0 | 1,7 |
| Prop. $\alpha = 0$ | 0,0 | 0,0 | 0,0 | 0,0 | 0,05 | 0,04 | 0,0 | 0,17 |
| Prop. $\alpha = 0,5$ | 0,0 | 0,0 | 0,0 | 0,0 | 0,05 | 0,04 | 0,0 | 0,17 |

Cuadro 5.3: Errores cometidos por cada método al obtener el contorno de una circunferencia con radio 20

El método sub-píxel consigue una precisión total en cuanto al módulo, y para la dirección también se obtiene valores bastante precisos, siendo el error medio de 0,05 grados y el error máximo inferior a 0,2 grados. Por otro lado, el método tradicional con $\alpha = 0$ tiene un error medio mayor de 45 unidades (error máximo mayor que 101) y de más de 7 grados para la dirección (casi 15 grados el error máximo). Con $\alpha = 0,5$ los valores tienden a mejorar, siendo el error máximo del módulo mayor de 17 unidades y el de dirección casi de 2 grados.

5.3. Estudio de un Caso Práctico

El caso práctico que se va a presentar tiene que ver con el estudio de la detección a nivel sub-píxel aplicado a imágenes de tomografía computarizada. Durante la realización de este trabajo, el autor ha tenido la oportunidad de participar como ponente en la conferencia EUROCAST 2011 (Thirteen International Conference On Computer Aided Systems Theory), con el artículo “A Subpixel Edge Detector Applied to Aortic Dissection Detection”¹. Éste plantea combinar el método de detección sub-píxel junto con un esquema de difusión anisotrópico (NRAD) para la detección de la disección aórtica en cortes axiales de imágenes de tomografía computarizada. Se describirá brevemente la enfermedad, el método usado y los experimentos realizados.

La aorta es el tronco principal del sistema circulatorio arterial, que se divide en cuatro partes: aorta ascendente, cayado aórtico, aorta descendente torácica y aorta descendente abdominal. Se origina en el orificio aórtico del ventrículo izquierdo, donde su diámetro es aproximadamente de 3cm, tiene un corto recorrido ascendente hacia el cuello, se dirige hacia la izquierda y hacia atrás por encima del bronquio principal izquierdo, desciende a lo largo del tórax situada a la izquierda de la columna vertebral y cruza el hiato aórtico del diafragma para entrar en la cavidad abdominal [39].

Las arterias están compuestas por tres capas (fig. 5.3), que desde el interior al exterior son denominadas: túnica íntima, túnica media y túnica adventitia. En ocasiones también se les denomina capa interna, media y externa.

¹Financiado por el proyecto SIMVA (Sistema de procesamiento de imágenes para el diagnóstico, la planificación, la simulación y el seguimiento de operaciones de cirugía vascular) TIN2009-10770 del Ministerio de Ciencia e Innovación del Gobierno de España

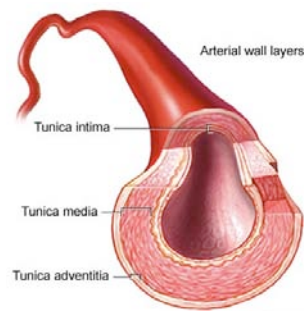


Figura 5.3: Estructura de la pared de una arteria. Imagen cortesía de A.D.A.M. Inc.

En condiciones normales, la sangre fluye por el interior del vaso. Pero, cuando ocurre una disección de la aorta, la capa más interna se rompe y la sangre entra en la capa media, existiendo un flujo de sangre dentro del vaso y otro entre las paredes del mismo (fig. 5.4).

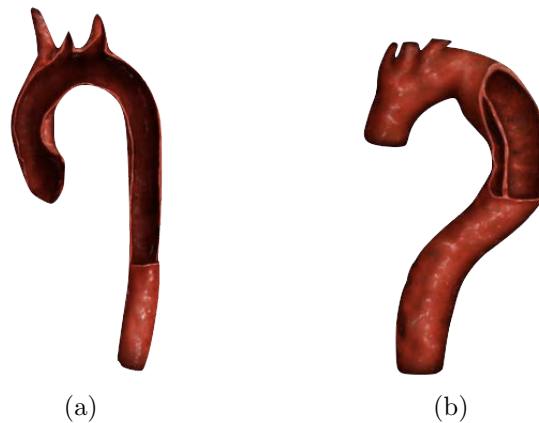


Figura 5.4: Imagen de una sección longitudinal del interior de la aorta: (a) Aorta en condiciones normales y (b) aorta con una disección. Imagen cortesía de The Visual MD

La disección puede tener una separación, longitud y grosor variables (desde unos pocos milímetros a varios centímetros). Cuando no se encuentran esas rupturas (llamadas puntos de entrada) se encuentra un hematoma intramural (la sangre ha coagulado dentro de la pared). Esto puede progresar en una disección completa, a través de una ruptura secundaria de la túnica media.

Existen varias clasificaciones. La más antigua, la clasificación DeBakey, dividía la disección en tres tipos. Pero la usada actualmente es la clasificación Stanford, basada en los diferentes pronósticos y el tratamiento terapéutico. Divide las disecciones en dos tipos: tipo A (empieza en la aorta ascendente y puede extenderse al callado aórtico y la aorta torácica descendente) y tipo B (todas las demás).

Independientemente de porcentajes de clasificación, el número más relevante es que un 70 % de los casos terminan con una rotura de la pared de la arteria, con sangrado en estructuras

adyacentes, como el sangrado pericárdico (más frecuente en el tipo A) [15].



Figura 5.5: Imagen de una tomografía computarizada de una disección aórtica. Cortes axial, coronal y sagital

En la figura 5.5 se puede observar una imagen diagnóstica de tomografía computarizada de una disección aórtica. En ella, la flecha negra indica la situación de la disección.

El método propuesto para la detección de los contornos de la disección en cortes axiales de la TC, hace uso del método de detección sub-píxel en imágenes con ruido (incorporando la detección de contornos cercanos y muy cercanos) más un método de filtrado anisotrópico, para la reducción del ruido. Este esquema, denominado Noise Reducing Anisotropic Diffusion (NRAD), preserva los bordes de una manera similar al método de Perona y Malik:

$$\begin{cases} u(0) = u_0 \\ \frac{\partial u}{\partial t} = \text{div}(c\nabla u) \end{cases} \quad (5.1)$$

Esta ecuación (5.1) usa un coeficiente de difusión, que es una función decreciente de la norma del gradiente. Esta función de difusión depende de un parámetro relacionado con el gradiente de la imagen y debe ser seleccionado por el usuario. Sin embargo, el filtro NRAD (ec. 5.2) usa una función de difusión $1 - k$, que depende de estadísticas locales de la imagen y del modelo del ruido.

$$\begin{cases} u(0) = u_0 \\ \frac{\partial u}{\partial t} = \text{div}((1 - k)\nabla u) = \text{div}\left(\frac{\sigma_n^2}{v_g}\nabla u\right) \end{cases} \quad (5.2)$$

En los experimentos se usó un modelo aditivo del ruido, $g = f + n$, donde la imagen observada g es la suma de la imagen ideal f más un ruido gaussiano aditivo n .

La ecuación 5.3 muestra que la imagen corregida \hat{f} se calcula usando un estimador, y es la suma de la media local \bar{g} y el producto entre el coeficiente k y la diferencia entre la imagen g y su media local \bar{g}

El coeficiente k (ec. 5.4) es el resultado de la división entre la varianza local de f , v_f , y la varianza local de g , v_g . El parámetro σ_n^2 es la varianza global del ruido, calculada en cada iteración usando una región de interés. La ecuación diferencial 5.2 está basada en este estimador [26].

$$\hat{f} = \bar{g} + k(g - \bar{g}) \quad (5.3)$$

$$k = \frac{v_f}{v_g} = \frac{v_g - \sigma_n^2}{v_g} \quad (5.4)$$

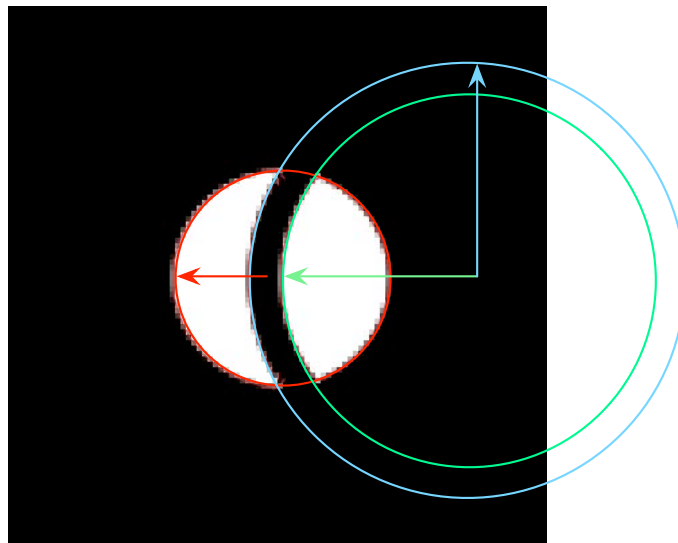


Figura 5.6: Disección sintética

Para los experimentos se usó una versión sintética de la disección (fig. 5.6) construida a partir de la intersección de un círculo con un anillo, realizándose tres tipos de pruebas:

- El primer tipo de experimentos prueba el método con diferentes valores para el grosor de la región central, la disección sintética.
- En el segundo tipo, se selecciona un grosor con un bajo nivel de error y se añade ruido a la imagen, desde 0 a 20 (desviación estándar).
- Finalmente, se combinan todos los grosores con todos los niveles de ruido.

El estudio concluye diciendo que la combinación del detector sub-píxel y el método NRAD proporciona un proceso de alta precisión para la detección de la disección aórtica. En un píxel que cumpla la hipótesis de efecto parcial, a pesar de un alto ruido, se consigue una detección de bordes con error medio máximo del 16 por ciento de un píxel [41].

Capítulo 6

Conclusiones y Trabajo Futuro

A lo largo de este trabajo se ha descrito cómo se ha realizado la integración de los algoritmos de detección de bordes con alta precisión dentro del software de procesamiento y visualización de imágenes AMILab. Durante dicho proceso se han incorporado mejoras en relación al uso de memoria, a través de mecanismos de almacenamiento que parten del hecho de que sólo es importante almacenar la información de los píxeles borde. A través del proceso de *wrapping* o encapsulado de código, se ha logrado poner estos métodos a disposición del lenguaje de *scripts* de AMILab. Mediante el uso de este lenguaje, se ha creado la interfaz para poder probar los métodos de forma sencilla, tanto en imágenes reales como sintéticas. En relación a éstas últimas, se ha conseguido un sistema de generación de imágenes artificiales bidimensionales y tridimensionales con efecto parcial.

Los resultados expuestos demuestran que se ha logrado una adecuada generación de estas imágenes, obteniéndose valores aceptables de intensidad para los píxeles borde, haciendo uso de un tiempo razonable de cómputo. Además, se observa que los métodos propuestos proporcionan mejores resultados que los tradicionales en relación a la estimación del salto de intensidad entre ambos lados del borde, así como la orientación de éste. También se refleja la aplicación práctica de los métodos, haciendo uso de los mismos en la resolución de un problema real, la localización de los contornos en imágenes diagnósticas de TC de disección aórtica.

No obstante, todo trabajo es susceptible de mejora y ampliación, y el descrito en el presente documento no será una excepción. Dentro de las mejoras posibles se encuentran las siguientes:

- Incorporación de la detección de contornos para altas curvaturas en imágenes 2D: aunque los métodos desarrollados para imágenes bidimensionales son capaces de detectar contornos muy cercanos en imágenes afectadas por bastante ruido, cuando se trata de altas curvaturas se pueden observar algunos errores. No obstante, en [42] ya se plantea una solución a este problema, sólo faltaría incorporarla al desarrollo hecho en AMILab.
- Añadir el método de detección de bordes en imágenes 3D con ruido: para imágenes 3D sólo se ha desarrollado el método más sencillo, pero que no es tolerante a la existencia de ruido y de bordes muy cercanos en la imagen.

- Agregar el método de restauración de imágenes tridimensionales: al igual que en el caso anterior, sería deseable incorporar la detección y consiguiente restauración en imágenes 3D.
- Incorporar en los métodos de detección 3D los mecanismos para la localización de bordes cercanos: mediante el uso de ventanas tridimensionales de límites variables y el estudio de los valores de las parciales más allá de esos límites.
- Mejorar la velocidad de dibujado del *script* 3D analizando la parte crítica del mismo, para acelerarla implementándola directamente en C++, al igual que se hiciera con el caso 2D: actualmente el código de dibujado es un poco lento para imágenes de grandes dimensiones. De la misma forma que sucedió en el caso bidimensional, la mejora pasa por ver las secciones críticas del código de dibujado en el *script*, con el objetivo de acelerarlas a través de una implementación directa en C++. No obstante, este paso no debería ser demasiado complejo, puesto que el uso que se hace en el lenguaje de *scripts* de las clases de VTK, es muy similar al que se hace directamente en C++.
- Adaptar los *scripts* para que se ajusten al sistema de *wrapping* automático: cuando se comenzó el presente trabajo aún no existía el mecanismo automático de *wrapping* de código, incorporándose al software AMILab más tarde. Se ha proseguido usando durante el desarrollo un *wrapping* manual. Una posible mejora pasaría por adaptar el código a que fuera compatible con la generación automática, pasando por incorporar a las propias funciones C++ la generación de los objetos `AMIObject` que encapsulan los resultados en variables de tipo `InrImage`.
- Si en un futuro se pasa a un visor de VTK, se podría adaptar el dibujado de contornos usando el *glyphing*, pero en este caso para imágenes 2D: la estructura sería similar a la desarrollada para el caso tridimensional. La única diferencia principalmente sería hacer uso de objetos `vtkGlyph2D`, así como de *sources* bidimensionales (líneas en lugar de planos, flechas 2D en lugar de flechas tridimensionales).
- Uso de la información sub-píxel para la generación de contornos continuos: actualmente el método aporta información local al píxel, de forma que cuando se realiza el dibujado del mismo éste es relativo únicamente a su región interna. Sería interesante, de cara a un proceso de segmentación por ejemplo, que se pudieran generar bordes continuos a partir de la información sub-píxel obtenida.
- Uso de la información sub-vóxel para la generación de mallados: similar al caso anterior pero para imágenes tridimensionales.

Apéndice A

Manuales de Usuario

A.1. Compilación de AMILab en Mac OS X

Para la instalación del software AMILab se precisan una serie de librerías:

- ITK
- VTK
- wxWidgets
- Pthreads
- Zlib
- Lib2zip
- Boost

Además de un conjunto de herramientas:

- CMake
- MacPorts
- Porticus (opcional, ya que es una interfaz gráfica para MacPorts)
- CodeBlocks (opcional, puesto que las compilaciones e instalaciones se harán en este tutorial mediante el terminal del sistema, aunque podría usarse sin ningún problema este IDE ya que CMake genera proyectos para él, aunque son makefiles igualmente)

Primeros pasos

Antes de nada es necesario descargar lo siguiente:

- Descargar e instalar [CMake](#).

- Descargar el código fuente de [ITK](#).
- Descargar el código fuente de [VTK](#).

A.1.1. ITK

Configurar ITK con CMake de la siguiente manera (al configurar con CMake puede ser útil utilizar “Grouped View”):

1. Al darle a “Configure”, seleccionar “CodeBlocks - Unix Makefiles” si se desea usar `Code::Blocks` o directamente “Unix Makefiles” para hacerlo desde el terminal.
2. Desmarcar todas las opciones de BUILD (Doxygen, Examples, Shared_Libs y Testing).
3. Dentro de las opciones de CMAKE:
 - CMAKE_BUILD_TYPE → Release
 - CMAKE_CXX_COMPILER → /usr/bin/c++
 - CMAKE_C_COMPILER → /usr/bin/gcc
 - CMAKE_OSX_ARCHITECTURES → i386
4. Darle a “Configure” y luego a “Generate”.
5. Desde un terminal, acceder a la carpeta en la que se ha volcado la salida de la configuración del CMake. Compilar mediante el comando `make` (puede hacerse `make -j n`, donde `n` es el número de tareas que queremos que se hagan en paralelo).
6. Una vez terminada la compilación, instalar mediante el comando `sudo make install`. En este momento las librerías de ITK se encontrarán en la ruta “/usr/local/lib/Insight-Toolkit”.

A.1.2. VTK

Hacer la misma configuración con CMake que se hizo con ITK en el apartado anterior (opciones de BUILD y de CMAKE). Una vez hecho este paso, realizar la compilación e instalación de la misma forma desde el terminal del sistema. Las librerías quedan en “/usr/local/lib/vtk”.

A.1.3. wxWidgets

1. Descargar los fuentes de [wxWidgets](#). La versión actual (2.8.10) presenta problemas al intentar [compilar en Snow Leopard](#). Es probable que la versión posterior (2.9.0) lo resuelva, pero actualmente se encuentra en desarrollo.
2. Descomprimir el fichero.
3. Acceder mediante un terminal a la carpeta donde se ha descomprimido. Antes de realizar la compilación e instalación, es preciso hacer lo siguiente:

```
arch\_flags=''-arch i386''  
./configure CFLAGS=''$arch_flags'' CXXFLAGS=''$arch_flags''  
CPPFLAGS=''$arch_flags'' LDFLAGS=''$arch_flags''  
OBJCFLAGS=''$arch_flags'' OBJCXXFLAGS=''$arch_flags''  
--with-opengl --enable-threads --enable-xpm
```

4. Ahora se compila mediante el comando `make` y se instala con `sudo make install`.
5. Las librerías se encontrarán en la misma ubicación que las anteriores.

A.1.4. Pthreads

Pthreads ya forma parte del propio sistema operativo. Si quiere comprobarlo sólo ha de ejecutar desde el terminal “`locate pthread`” y verá que se encuentra dentro de “`/Developer/SDKs`”.

A.1.5. zlib y bzip2

Para realizar esta instalación se usará la herramienta [MacPorts](#) comentada anteriormente. Opcionalmente, si quiere mayor comodidad, puede usar la interfaz gráfica para MacPorts llamada [Porticus](#). Para instalar ambas librerías sólo ha de usarse esta herramienta. MacPorts las descargará, compilará e instalará. Puede hacerlo a través de la interfaz gráfica, usando la búsqueda y seleccionando los “Ports” correspondiente y dándole a “Install”. Otra manera es desde el terminal, mediante los comandos `port search` (para buscar) y `port install` (para instalar).¹

A.1.6. Boost

Boost puede ser instalado de la misma forma que las anteriores librerías (mediante MacPorts), pero es un proceso más lento. Puede seguirse el siguiente procedimiento:

1. Descargarse los fuentes de [Boost](#) (descargar la versión que incluye CMake).
2. Configurar con CMake de la misma forma que se hizo anteriormente con ITK y VTK (Opciones de CMAKE).
3. Compilar e instalar de la misma forma que anteriormente (`make, sudo make install`).
4. Boost quedará instalado en “`/usr/local/lib`”

¹NOTA: para poder usar Porticus es necesario que anteriormente haya sido instalado MacPorts. Antes de poder instalar nada ha de actualizarse la base de datos de MacPorts. En Porticus, desde el menú Ports, la opción `Self update`. Desde el terminal, ejecutar el comando “`port selfupdate`”.

A.1.7. AMILab

1. Descargar el código fuente de [AMILab](#).
2. Configurar libAMIFluid con CMake. Tener en cuenta, como en casos anteriores, las opciones de CMAKE (Release, compiladores y arquitectura).
3. Compilar e instalar libAMIFluid desde el terminal con `make` y `sudo make install`. Quedará instalado en “/usr/local/lib”.
4. Copiar libAMIOpticalFlow en el directorio de instalación:

```
cp -R include/OpticalFlow /usr/local/include/
```

5. Configurar AMILab con CMake. Considerar las mismas opciones que anteriormente.
6. Ahora, desde el terminal del sistema, compilar AMILab con `make`.
7. En este momento ya se puede ejecutar AMILab como `./amilab_X.X.X_release`

A.1.8. Problemas Conocidos

Debido a que la versión 10.6 del sistema operativo de Mac es nativamente de 64bits, las compilaciones por defecto las hace para dicha arquitectura. Es preciso tener esto en cuenta pues, aunque alguna de las instalaciones se podrían hacer a través de la herramienta MacPorts, además de ser muy lentas pueden generar compilaciones que no sean compatibles con el resto. Un ejemplo claro es Boost. Si se descarga esta librería desde MacPorts, su compilación se hará para 64bits, mostrándose posteriormente problemas a la hora de compilar AMILab. Es por ello recomendable seguir los pasos indicados en el manual de usuario. Además, la actual versión de xwWidgets no puede ser compilada para 64bits.

La versión actual de AMILab, al ser compilada con la última versión del compilador GCC, consume bastantes recursos. Es posible intentar compilar AMILab con versiones anteriores de GCC, pero tanto ITK, como VTK y Boost; precisan de la última versión del compilador para poder llevar a cabo la instalación. Si se intenta combinar distintas versiones (por ejemplo C++ para ITK, VTK y Boost, y gcc-4.0 para AMILab) la compilación del software será más rápida, pero no será satisfactoria.

Puede suceder que usted no pueda compilar. Es probable que ocurra porque no se han instalado las herramientas de desarrollo, que se encuentran en el DVD de instalación del sistema operativo.

A.2. Ayuda del *script* para la Generación de Imágenes Sintéticas

Este *script* provee una interfaz de usuario para la creación de imágenes sintéticas 2D y 3D basadas en una función analítica (línea, círculo, vaso, anillo, cubo, cilindro, esfera y toro).

A.2.1. Inicio Rápido

Para crear una imagen sintética siga estos pasos:

1. Cree una nueva imagen de entrada (2D o 3D) en la pestaña **Main**.
2. Cree una nueva función analítica desde la pestaña **Func**.
3. Aplique el método en la pestaña **Main**.
4. Usted puede mostrar o guardar el resultado final desde la misma pestaña.

A.2.2. Interfaz

La interfaz está dividida en varias pestañas.

A.2.2.1. Pestaña **Main**

En la pestaña **Main** puede crear una imagen de entrada, indicando las dimensiones en x, y, z .

Después de crear la función analítica, usted puede aplicar el método. Puede elegir el nivel de subdivisión recursiva, así como aplicar dos métodos diferentes:

- **Analytic Partial Surface**: calcula el efecto parcial en una imagen 2D usando la función analítica de una línea, un círculo, un vaso o un anillo.
- **Analtic Partial Volume**: calcula el efecto parcial en una imagen 3D usando la función analítica de un cubo, un cilindro, un esfera o un toro.

Además, puede guardar la imagen resultante como una variable global en **AMILab**.

A.2.2.2. Pestaña **Func**

En la pestaña **Func** usted puede crear el objeto de una función analítica. Existen ocho funciones con algunos parámetros:

- Línea o rampa: ángulo de inclinación y corte con el eje y .
- Círculo: el centro y el radio.
- Vaso: el centro, el ángulo y el grosor.
- Anillo: el centro, el radio y el grosor.
- Cubo: el centro y las dimensiones en x, y, z .

- Cilindro: el radio y las direcciones en x, y, z .
- Esfera: el centro y el radio.
- Toro: el centro y los radios mínimo y máximo.

Todas las funciones analíticas tienen la opción de recarga. Si es seleccionado dicho *checkbox*, podrá ver los cambios en la imagen cuando se modifica algún parámetro.

A.2.2.3. Pestaña Help

Esta ayuda en inglés.

A.3. Ayuda del *script* para la Detección de Bordes con Precisión Sub-Píxel

Este *script* proporciona una interfaz de usuario para la detección de bordes con precisión sub-píxel en imágenes 2D.

A.3.1. Inicio Rápido

Para detectar los bordes en una imagen 2D siga los siguientes pasos:

1. Seleccione una imagen de entrada 2D en la pestaña **Param**.
2. Seleccione el valor del umbral en la misma pestaña (además puede elegir si usted está detectando un borde de primer orden).
3. Seleccione el método a aplicar y si usted quiere dibujar las normales de los bordes. Dependiendo del método, usted puede seleccionar algunos parámetros.
4. Dele al botón run.

A.3.2. Interfaz

La interfaz se encuentra dividida en varias pestañas.

A.3.2.1. Pestaña Param

En la pestaña **Param** usted puede seleccionar una imagen de entrada para la detección sub-píxel de los bordes. Es necesario indicar un valor para el umbral para la detección (25 por

defecto). Además, es posible indicar al algoritmo que se está intentando detectar bordes de primer orden.

Tras seleccionar estos parámetros, usted puede elegir el método a aplicar, decidir si pintar las normales a los bordes o seleccionar algunos parámetros dependiendo del método que haya seleccionado.

Los métodos que puede seleccionar para la detección sub-píxel son:

- **Basic detector:** es una detección básica sub-píxel. No puede detectar bordes muy cercanos o en imágenes con ruido.
- **Averaged detector:** detecta bordes en imágenes con un bajo nivel de ruido. Detecta también bordes cercanos.
- **Subpixel denoising:** es el método más completo. Puede detectar bordes en imágenes con un ruido mayor y bordes muy cercanos. Además, durante la detección, crea una restauración de la imagen de entrada.

En general, los parámetros basados en el método que usted puede seleccionar están relacionados con:

- La comparación entre la detección sub-píxel de la imagen de entrada y la imagen suavizada/restaurada.
- La posibilidad de dibujar la detección sub-píxel en la imagen suavizada/restaurada.

En el caso del método **Subpixel denoising**, se puede seleccionar también el número de iteraciones a aplicar.

En la parte inferior de esta pestaña se encuentra el botón **run**.

A.3.2.2. Pestaña **Px Info**

En la pestaña **Px Info** puede ver información acerca de un píxel. Esto es:

- **Pixel:** la posición y el tipo de borde (horizontal o vertical) del píxel seleccionado.
- **Mod:** el salto de intensidad a ambos lados del píxel borde.
- **Rad:** el radio.
- **Slo:** la pendiente.
- **Dis:** el desplazamiento respecto al centro del píxel.

Si quiere ver información sobre un píxel, solamente debe hacer clic con el botón central del ratón en la imagen y entonces presionar el botón de información situado en la parte superior de esta pestaña. En el caso de que el píxel seleccionado no sea un píxel borde, la posición y el texto **NO EDGE** son mostrados.

A.3.2.3. Pestaña **Stat**

En la pestaña **Stat** usted puede calcular las estadísticas de la detección sub-píxel. En el *text box method* aparece el nombre del método que ha sido aplicado a la imagen de entrada.

Las estadísticas que pueden obtenerse son:

- Mínimo.
- Máximo.
- Media.
- Varianza.
- Desviación estándar.

Estos cálculos se hacen para el salto de intensidad, el radio y la pendiente.

A.3.2.4. Pestaña **Set**

En la pestaña **Set** usted puede cambiar las preferencias de los bordes y las normales. Primero, usted debe seleccionar que desea cambiar las preferencias por defecto.

En ambos casos, bordes y normales, usted puede seleccionar:

- **Color:** puede seleccionar el color para los bordes y las normales en escala RGB. Además, puede indicar el valor del canal alfa para la transparencia (0 para transparente y 255 para opaco).
- **Properties:** aquí usted puede seleccionar el grosor y el estilo de la línea.

Cuando aplique el método verá los cambios establecidos.

A.3.2.5. Pestaña **Help**

Esta ayuda en inglés.

A.4. Ayuda del *script* para la Detección de Bordes con Precisión Sub-Vóxel

Este *script* proporciona una interfaz de usuario para la detección de bordes con precisión sub-vóxel en imágenes 3D.

A.4.1. Inicio Rápido

Para detectar los bordes en una imagen 3D siga los siguientes pasos:

1. Seleccione una imagen 3D de entrada en la pestaña **Param**.
2. Seleccione el valor del umbral en la misma pestaña.
3. Seleccione el método a aplicar y si desea dibujar el volumen, los bordes y/o las normales.
4. Presione el botón **run**.

A.4.2. Interfaz

La interfaz está dividida en varias pestañas.

A.4.2.1. Pestaña **Param**

En la pestaña **Param** usted puede seleccionar una imagen de entrada para la detección sub-vóxel de los bordes. Es necesario indicar un valor para el umbral para la detección (25 por defecto).

Después de seleccionar estos parámetros, usted puede elegir el método a aplicar, y seleccionar si quiere dibujar el volumen, los bordes o las normales de los bordes.

En el *box panel* de visualización usted puede seleccionar dibujar:

- El volumen.
- Los bordes.
- Las normales.

En la parte inferior de esta pestaña está el botón **run**. Además, usted puede encontrar a la derecha de dicho botón, el botón de recarga. Puede usarlo si cambia cualquier parámetro de la pestaña **Set**.

A.4.2.2. Pestaña **Set**

En la pestaña **Set** usted puede cambiar algunos parámetros de visualización. Las preferencias están divididas en cuatro partes:

- **Volume:** puede seleccionar la opacidad del volumen. Por ahora, el volumen cargado es una isosuperficie de la imagen 3D de entrada. Funciona con imágenes sintéticas.

- **Planes:** los planos representan los bordes detectados en la imagen. En cada vóxel el algoritmo dibuja un pequeño plano en la posición sub-vóxel obtenida por la detección. Puede seleccionar la opacidad y el color (en RGB). Por defecto los planos son opacos y el color es rojo (255,0,0).
- **Normals:** al igual que en el punto anterior, aquí puede elegir la opacidad y el color de las normales de los bordes. Por defecto, las normales son opacas y el color es azul (0,0,255).
- **Background:** el color del fondo del visor se puede personalizar. Puede seleccionar un color alternativo para el mismo desde esta pestaña.

A.4.2.3. Pestaña Help

Esta ayuda en inglés.

Apéndice B

Algunas mejoras añadidas en **AMILab**

B.1. Mejoras en el Editor de *Scripts* de **AMILab**

B.1.1. Buscar y Reemplazar (Find and Replace)

Para la implementación de la nueva funcionalidad de buscar y reemplazar se ha hecho la inclusión de una nueva clase en la estructura del fichero `wxEditor.h`. La clase `wxEditor` es una herencia del objeto `wxStyledTextCtrl` de `wxWidgets`. La nueva clase incluida tiene las operaciones necesarias tanto para la construcción y destrucción de un objeto de este tipo, como aquellas para el control de eventos. En ese sentido la captura de los mismos se puede diferenciar en dos clases distintas en función del tipo:

- Eventos de ratón: se refiere a todos aquellos eventos de clic sobre los botones de la interfaz de la ventana. Entre dichos botones se encuentran buscar la siguiente aparición (`next`), la anterior (`prev`), reemplazar (`replace`), reemplazar todos (`replace all`) y abandonar la ventana (`done`).
- Eventos de teclado: permite poder ejecutar las mismas funcionalidades provistas desde la interfaz a través del uso de atajos de teclado o “shortcuts”, como el uso de teclas de función o la captura del evento “enter” en la caja de texto donde se escribe lo que se desea encontrar.

Dentro del conjunto de miembros de la clase se incluyen las cajas de texto para el texto a buscar y el usado para reemplazar (para posteriores evaluaciones de su contenido desde distintos métodos), los *checkbox* que indican si se quiere hacer la búsqueda de la palabra completa o teniendo en cuenta mayúsculas y minúsculas; así como un puntero al editor, para poder aplicar las operaciones sobre el mismo. Es preciso destacar que en la declaración de la clase ha de indicarse que ésta tendrá manejo de eventos a través de la directiva `DECLARE_EVENT_TABLE()`. Ésta es una macro que puede encontrarse en cualquier parte de la declaración de la clase, aunque habitualmente se suele poner al final de la misma, ya que cambia implícitamente el acceso a la parte `protected` y esto puede ser bastante inesperado si hay algo que le siga debajo.

Debido a que en el ejemplo del editor *Scintilla* que incorpora *wxWidgets* se contempla la posibilidad de una futura ampliación mediante el desarrollo de la funcionalidad de poder buscar y reemplazar un texto, sólo será preciso crear el nuevo objeto de esta clase en la función que maneja el evento, bien haya sido producido por la selección de la operación en el menú **Edit** o bien mediante el atajo de teclado **ctrl+f**; indicando que su padre es el editor actual (si se cerrase el editor se cerraría esta nueva ventana creada).

A continuación se describirán los pasos seguidos para el desarrollo de las diferentes operaciones de esta clase.

B.1.1.1. Constructor

Es el encargado de crear e inicializar un objeto del tipo *FindAndReplace*. Para ello hace uso de una llamada al constructor de su clase padre *wxFrame*. Una vez hecho esto es preciso generar los elementos que configurarán la interfaz.

Para la estructuración de la interfaz se hace uso de los *sizer*. Éstos permiten organizar estructuralmente elementos de la interfaz de usuario, a través de disposiciones en forma horizontal, vertical o de rejilla (esta estructura es denominada en *wxWidgets* como *GridSizer*). En la figura B.1 se puede observar cuál es la configuración que queda para la interfaz que permite buscar y reemplazar en el editor de *scripts* de *AMILab*.

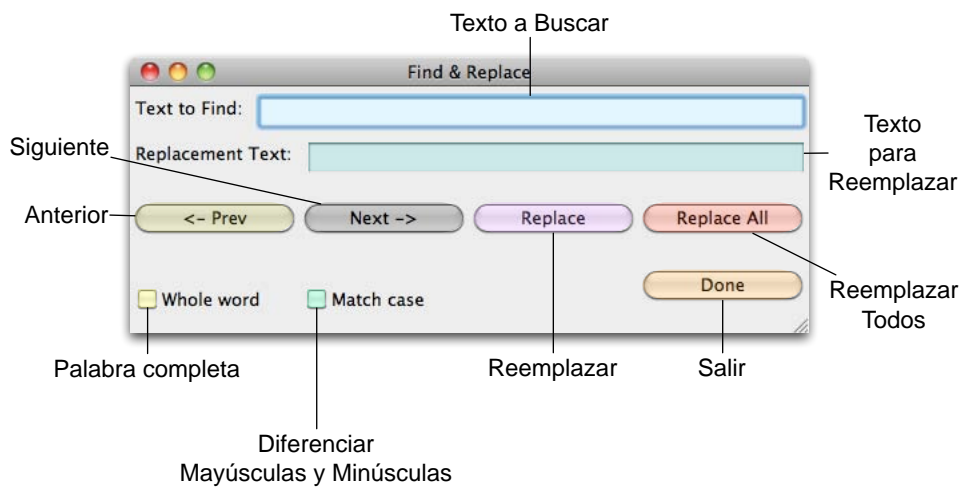


Figura B.1: Detalle de la interfaz para buscar y reemplazar

Destacar que al hacerse la llamada al constructor de la superclase, se indica en el estilo de la ventana que se mantenga encima de la del editor. Esto es bastante deseable, sobre todo cuando se está realizando alguna búsqueda y de forma involuntaria se hace clic en la ventana posterior.

Aparte de las diferentes funcionalidades con las que cuenta la interfaz y que serán descritas en lo siguiente (refiriéndose a aquellas realizadas por los botones) se encuentra además a disposición del usuario la posibilidad de seleccionar que la búsqueda sea hecha teniendo en cuenta la palabra completa o siendo sensible a mayúsculas y minúsculas (recuadros en amarillo y verde

respectivamente que se pueden localizar en la figura B.1). Asimismo, aparecen sendos campos para introducir tanto el texto a buscar como aquel a usar para realizar un reemplazo (marcados en azul y verde azulado en B.1).

A continuación se describirá la implementación de las principales operaciones que pueden ser realizadas mediante la nueva funcionalidad de buscar y reemplazar del editor de *scripts*. No obstante, es necesario explicar algo antes. A pesar de que el ejemplo de editor incluido en el apartado `contrib` de `wxWidgets` no cuenta con la implementación de estas operaciones (ni de la ir a una línea) el wrapping realizado en estas librerías de `Scintilla`, y denominado `wxStyledTextCtrl`, provee de las funciones necesarias para dicho desarrollo, haciendo que tener que implementarlas no sea de gran dificultad. Es por ello que contando con la documentación existente, tanto de `wxWidgets` como de `wxStyledTextCtrl`, resulta más sencilla esta tarea [43].

B.1.1.2. Buscar siguiente (Next)

Esta es la operación que se ocupa de buscar la siguiente aparición de la cadena introducida. Se puede localizar en la interfaz de usuario mediante un botón etiquetado con `Next→` (véase botón marcado en gris en la figura B.1).

El método de la clase que realiza esta funcionalidad es aquel denominado como `OnNextButtonClick`. Esta especificación es debida a que trata de identificarse a través de qué evento es llamado, que en primera instancia puede ser al realizar un clic en el botón `Next→` (aunque como se explicará en la parte de tratamiento de eventos se puede realizar esta operación de otras maneras). Es preciso destacar que para poder realizar esta operación es necesario introducir un texto a buscar. Por tanto, se tiene en cuenta esta eventualidad (informando al usuario de este suceso a través de un `wxMessageDialog` de `wxWidgets`). Otra situación que puede darse es que el texto que se haya introducido no se encuentre en todo el código del *script*. Para comprobarlo se hace uso de una función interna denominada `isFound` que devuelve verdadero si encuentra al menos una vez el texto o falso en caso contrario. En ese caso, se hace uso de otra operación interna, `notFound`, que muestra un `wxMessageDialog` indicando que la palabra que se intenta buscar no se encuentra dentro del texto (se incluye en dicho mensaje el texto introducido además de avisos sobre si se tiene activada alguna de las opciones de búsqueda de la palabra completa o sensibilidad a mayúsculas y minúsculas).

Si se sobrepasan estos controles, ha de tenerse en cuenta las características de la búsqueda, es decir, si se ha solicitado que se considere la palabra completa o que sea sensible a mayúsculas y minúsculas. Esto es posible realizarlo a través de una serie de macros del método `SearchNext` de `wxStyledTextCtrl`:

- `wxSTC_FIND_WHOLEWORD`: sólo habrá éxito en la búsqueda si los caracteres que se encuentran antes y después de la palabra que se está buscando no son caracteres que pueden formar parte de ella (p.ej. espacios en blanco).
- `wxSTC_FIND_MATCHCASE`: habrá acierto si la cadena a buscar y el texto coinciden completamente (mayúsculas y minúsculas).

En caso de que haya éxito en la búsqueda, el método `SearchNext` dejará seleccionado el texto encontrado. No obstante, si la línea en la que ha sido localizada la aparición no está a la vista, esta operación no hará que se muestre (a través de una operación de scroll del contenido de la ventana del editor). Es por ello que se hace necesario el uso del método `LineFromPosition` que devuelve el número de línea a partir de una posición dentro del fichero. El contenido es mostrado correctamente pasando dicho valor a la función `EnsureVisibleEnforcePolicy`, que fuerza que la línea que se le pasa por parámetro esté visible para el usuario.

Si se alcanza el final del fichero durante la búsqueda, se le solicita al usuario si desea volver al principio. Si la respuesta es afirmativa, se recoloca la posición al principio y se busca desde ahí la primera aparición del término.

B.1.1.3. Buscar anterior (Prev)

Este método se ocupa de buscar la aparición previa de un texto dentro del código del *script*. Puede identificarse en la figura B.1 como el botón que aparece a la izquierda marcado en marrón con la etiqueta `←Prev`.

El método que realiza esta operación es denominado `OnPrevButtonClick`, por la misma razón especificada anteriormente. La filosofía seguida es muy similar a la operación anterior, tanto en el control de que se ha haya introducido texto a buscar, como que éste no sea localizado a priori dentro del contenido del fichero.

La diferencia fundamental es que en este caso se hace uso del método `SearchPrev` de `wxStyledTextCtrl` que busca “hacia atrás” dentro del editor. En relación a tener en cuenta la palabra completa o que la búsqueda sea sensible a mayúsculas y minúsculas, se hace uso de las mismas *flags* que en el caso de buscar el siguiente. Si durante la búsqueda se alcanza el comienzo del fichero se realiza la misma consulta al usuario que cuando antes se llegaba al final del mismo, recolocándose en caso de respuesta afirmativa al final del *script* y buscando la primera aparición del texto a buscar ubicada desde ahí.

B.1.1.4. Reemplazar (Replace)

Esta función es la encargada de realizar el reemplazo de un texto. El texto a reemplazar es el indicado en `Text to Find` y el que será usado para la sustitución es el que se introduce en `Replacement Text` (ver figura B.1). Aparece en un botón etiquetado como `Replace` (en violeta en la figura citada anteriormente).

El método de la clase `FindAndReplace` que se ocupa de esta funcionalidad es `OnReplaceButtonClick`. Al igual que se hiciese en anteriores operaciones, se tiene en cuenta si existe contenido dentro del campo de texto a buscar, ya que no es posible reemplazar una cadena si no se indica qué es lo que se quiere cambiar. En el caso del texto que se usa para sustituir esto no es necesario, puesto que puede interesar simplemente sustituir por nada (borrar del *script* la aparición de la cadena introducida como texto a buscar). También se tiene en cuenta

que el texto a reemplazar esté en el fichero haciendo uso de `isFound`, operación que ya ha sido descrita.

Cuando se quiere hacer una sustitución dentro de un texto se pueden dar dos situaciones. En primer lugar, puede darse el caso que se estuviese realizando una búsqueda y en ese preciso instante se quiera reemplazar el texto encontrado. En esta situación, simplemente se realiza la sustitución a través del método `ReplaceSelection` de `wxStyledTextCtrl`. El otro caso que puede darse es que se conozca de antemano qué es lo que se quiere sustituir (p. ej. que haya sido parte del error por el que se abrió el editor). Ahí, el usuario puede tomar la decisión de introducir tanto el texto a buscar como el que se usará para la sustitución, sin realizarse una búsqueda previa. En esa situación, el método actúa de manera inteligente y si ve que no hay una selección producto de una búsqueda, invoca al método para buscar el siguiente y realiza la sustitución.

En cualquier caso, una vez se ha reemplazado el texto a buscar por el indicado, el editor busca la siguiente aparición del texto, por si el usuario quiere seguir reemplazando el término que introdujo.

B.1.1.5. Reemplazar todos (Replace all)

El funcionamiento de esta operación es similar a la anterior, salvo que en este caso el reemplazo es realizado a lo largo de todo el código del *script*. Puede observarse que en la interfaz de usuario (figura B.1) aparece en el botón con etiqueta **Replace all** marcado en rojo.

La denominación del método de la clase que se ocupa de esta operación es `OnReplaceButtonClick`. El control de eventualidades es similar a los descritos anteriormente (que no haya texto a buscar o que no se encuentre dentro del fichero). Hay que tener en cuenta que esta puede ser una operación peligrosa, ya que se va a realizar una sustitución a lo largo de todo el fichero. Es por esta razón que, en caso de haber superado los controles previos, se pregunta si realmente se desea hacer esta operación. En caso de respuesta afirmativa se tiene en cuenta, como siempre, que se haya elegido que el proceso se haga teniendo en cuenta palabras completas o sea sensible a mayúsculas y minúsculas. Para realizar el proceso de sustituciones a lo largo de todo el fichero se hace uso de los métodos `SearchNext` y `ReplaceSelection`, que son llamados sistemática y repetidamente hasta que la búsqueda del término a sustituir se infructuosa.

Para finalizar, esta operación da un informe al usuario, indicando cuál ha sido el número total de sustituciones que se han hecho en el texto.

B.1.1.6. Tratamiento de eventos

A lo largo de la descripción de las distintas operaciones se ha ido indicando el nombre a las mismas, que en un principio, parece estar asociado a un evento `click` en un botón determinado de la interfaz de usuario. No obstante, es posible hacer uso de las operaciones de buscar y

reemplazar sin tener que hacer uso del ratón. Es habitual encontrarse con usuarios que prefieren hacer uso de *shortcuts* o atajos de teclado.

Teniendo en cuenta esto, se ha hecho una función que se ocupa de capturar los eventos de teclado. En concreto, para las funcionalidades de buscar el anterior y el siguiente, se capturan las teclas F2 y F3 respectivamente (siguiendo la misma filosofía existente en otros editores como **KDevelop** p. ej.). A pesar de ello surge un pequeño problema asociado al carácter multiplataforma de **AMILab** y al comportamiento que tiene en distintos sistemas **wxWidgets**. En **wxMac**, por defecto, se realiza un propagado de los eventos al padre. Es decir, independientemente del elemento de interfaz que en ese momento tenga el foco, si se pulsa la tecla, ésta será atendida por quien corresponda, no quedándose el evento en el objeto que tiene el control en ese instante. En **Linux** no sucede así. Por ello, se añade una nueva clase en el apartado **wxParams** de **AMILab**, denominada **CEventPropagator**. Su función es recorrer el conjunto de hijos de una ventana que es indicada por parámetro al método **registerFor** haciendo que el tratamiento de los eventos se propague.

Además, se ha habilitado la captura de la pulsación de la tecla **Enter** cuando se está en cualquiera de los campos de texto de la interfaz, proporcionando así que se realice la operación asociada al pulsar dicha tecla (que también es un comportamiento bastante natural al hacer uso de este tipo de funcionalidades).

Posteriormente a lo descrito, se han hecho mejoras como p. ej. que si al hacer uso de la operación de buscar a través de **ctrl+f** y existe un texto seleccionado, éste pase automáticamente a estar en el cuadro de texto del a buscar. Pequeñas mejoras como ésta no han sido realizadas por el autor de este trabajo, si no por el equipo de desarrollo de **AMILab**.

B.1.2. Ir a la línea (Go to line)

Para seguir la misma filosofía que con la funcionalidad de buscar y reemplazar, se ha añadido también una nueva clase dentro del fichero **wxEditor.h**. Esta nueva clase cuenta con dos atributos: un puntero al editor (para poder acceder a la línea dentro del mismo) y el campo de texto donde se introduce la línea a la que se quiere acceder. Asimismo, se cuenta con los métodos que se ejecutan al capturar un evento.

Al igual que con la clase anterior, el constructor de la misma es el encargado de generar la interfaz de usuario. Para ello, hace igualmente uso de los **sizer** de **wxWidgets**. En este caso se trata de una estructura bastante más simple, que se puede observar en la figura B.2.

El método de la clase que se ocupa de realizar la operación de ir a la línea se denomina **onOKButtonClick**, siguiendo la convención descrita en apartados anteriores. Ésta se ocupa de obtener, por un lado el número total de líneas del fichero, y por otro de convertir a número lo indicado en el campo de texto de la interfaz (a través de la operación **wxAtoi**). El objetivo de tener en cuenta el número total de líneas del fichero es comprobar que no se intenta acceder a una línea inválida. Igualmente se comprueba que el número indicado no es menor que 1. En caso de darse alguna de estas dos situaciones, se informa al usuario a través de un **wxMessageDialog**, indicando además entre qué valores debería estar la línea a la que se quiere acceder.

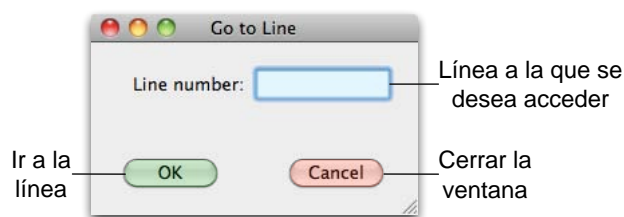


Figura B.2: Detalle de la interfaz para ir a una línea determinada

Si el procedimiento es correcto (la línea se encuentra dentro del rango indicado), se accede a la misma mediante el método `GotoLine` de `wxStyledTextCtrl`. A éste se le pasa el número indicado menos uno. Esto es así porque, a pesar de que las líneas se encuentran numeradas de $(1 \dots n)$ se acceden como $(0 \dots n - 1)$.

En relación a la captura de eventos, como ya se ha indicado, el nombre de la función viene dado porque se identifica como la ejecución ante un evento clic de ratón sobre el botón `OK`. De igual forma sucede cuando se pulsa sobre `Cancel`, pero en este caso simplemente se cierra la ventana. Además de los eventos de ratón, también se captura la pulsación de la tecla `Enter` en el campo de texto para indicar el número de línea. Éste es atendido a través del método `OnNumTextEnter`, que simplemente invoca a la misma operación que atiende al evento clic sobre `OK` anteriormente indicado.

B.1.3. Mejoras de la interfaz

Otra de las nuevas mejoras a incluir en el editor es la posibilidad de tener abiertos varios ficheros a la vez. Esto se hará gracias a la clase `wxAuiNotebook` de `wxWidgets`. Esta clase permite gestionar una ventana con pestañas, incluyendo además detalles como: poder desplazarlas, ubicarlas en diferentes partes de la ventana (para dividir la vista) o tener un botón para cerrar el fichero en la propia pestaña (todas ellas características bastante comunes en cualquier editor).

Para hacer estos cambios habrá que editar los ficheros `wxStcFrame.h` (cabecera) y `wxStcFrame.cpp` (implementación). En este fichero se encuentra la clase `wxStcFrame` que contiene al editor y que hereda de la clase `wxFrame` de `wxWidgets`. Los atributos añadidos a la clase para poder hacer las mejoras en la interfaz son los siguientes:

- `notebook`: es un puntero a un objeto de la clase `wxAuiNotebook` de `wxWidgets`. Será el encargado de manejar las pestañas que albergarán los distintos ficheros que se encuentren abiertos.
- `tabs`: es un objeto del tipo `tabsVector` que será descrito posteriormente.

Para poder hacer un tratamiento de las pestañas, es preciso tener una referencia de las que se encuentran abiertas, de cara a poder acceder a una en concreto o chequear si el fichero ya está abierto. Para ello se usa el atributo anteriormente citado `tabs`. `tabs` es un objeto del tipo

`tabsVector`. Ésta es una clase que engloba las operaciones típicas que pueden hacerse con las pestañas: adición, borrado y acceso. Para ello, ésta clase tiene como atributo un vector (de la STL de C++) cuyo contenido es del tipo `epf`.

`epf` es otra clase, y dichas siglas identifican a la terna “‘edit + page + filename’”, que representa a una pestaña concreta. Es así que cada pestaña tendrá un puntero al editor que contiene (e), un puntero a la ventana que representa la página del *notebook* (p) y una variable de tipo `wxString` que representa el nombre del fichero (f). Se puede ver un mayor detalle de este desarrollo en la figura B.3. En el mismo se puede observar una relación de agregación tanto entre la clase `wxStcFrame` y `tabsVector`, como entre este último y la clase `epf`. Esto es así puesto que la existencia de las clases agregadas depende de la existencia de aquellas que las agregan (de su tiempo de vida). En la primera de ellas se puede ver que la cardinalidad es de uno a uno, puesto que por cada ventana de editor habrá un sólo objeto de control de las pestañas. En la segunda, la relación es de uno a uno o muchos, ya que el vector que contiene las referencias a la identificación de cada una de las pestañas es de tamaño variable.

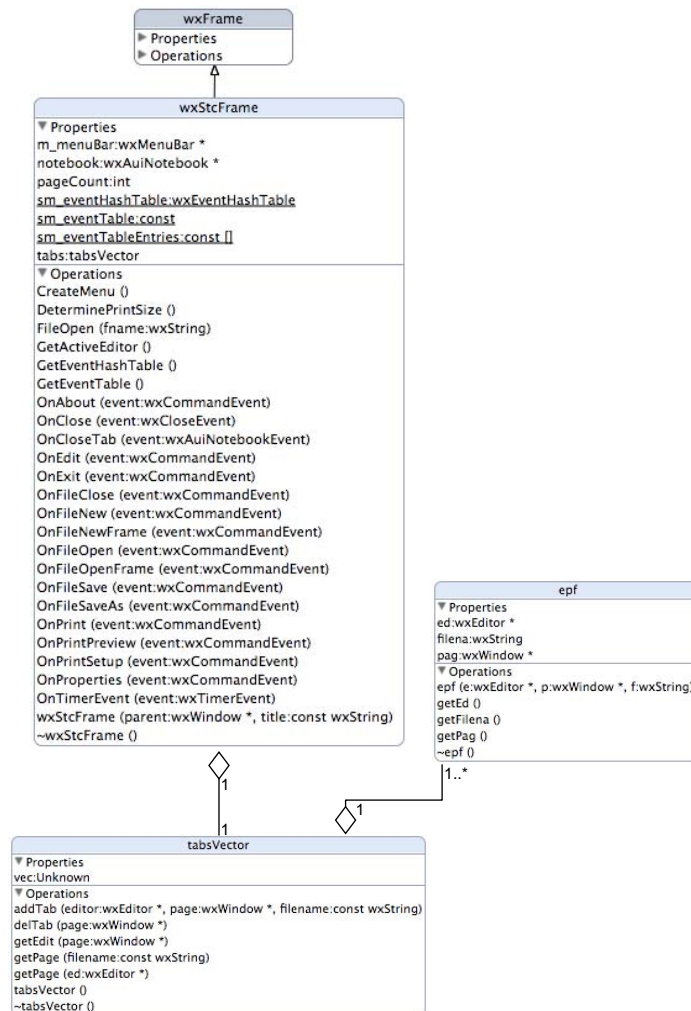
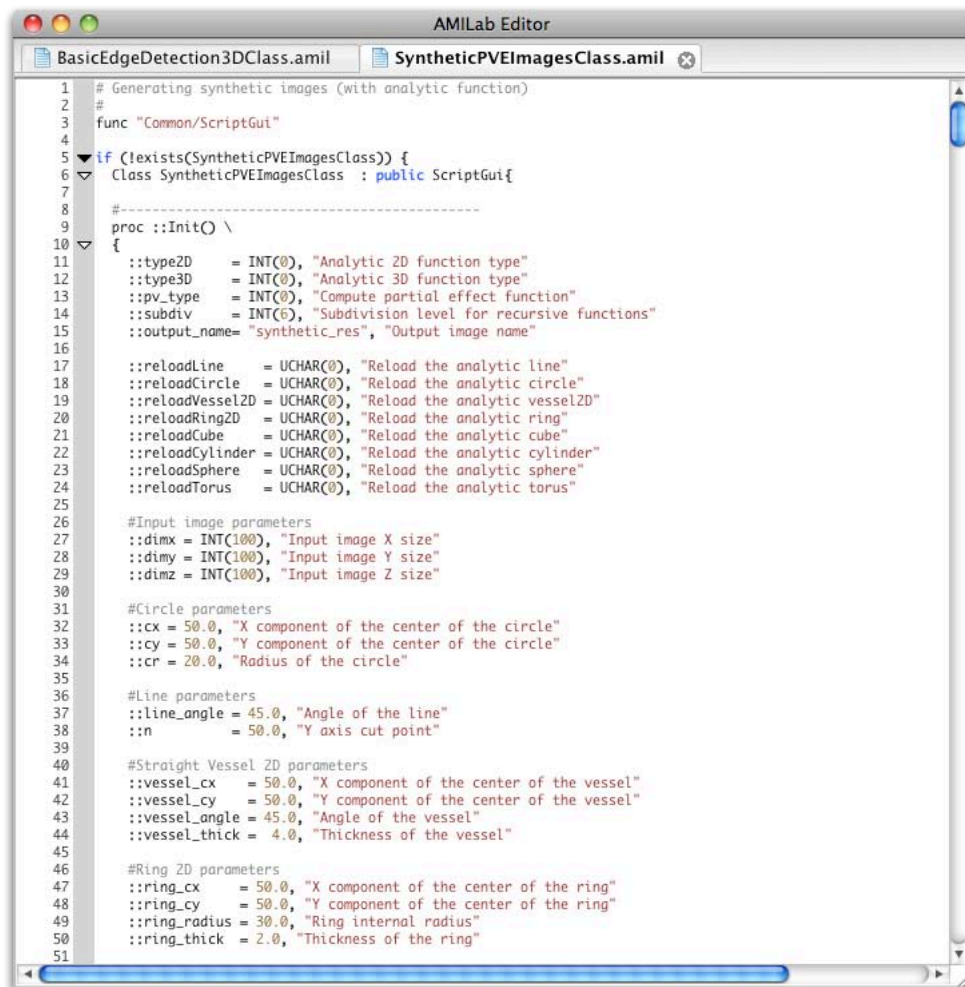


Figura B.3: Diagrama de clases para la gestión de pestañas

Es en el constructor de la clase principal (`wxStcFrame`) donde se asocia el *notebook* con

la ventana (que es el objeto `this`). De esta forma quedan ligados ventana y *notebook*. Ahora han de tenerse en cuenta las situaciones que se pueden dar, considerando las operaciones de apertura y cierre de ficheros. Ahora, cada vez que se abra un fichero, en primer lugar se hará un recorrido de la lista de pestañas activas, para ver si el mismo se encuentra ya abierto. En caso afirmativo, se preguntará al usuario si desea abrir nuevamente el fichero (dividiéndose entonces la vista en dos para que pueda ver a ambos lados del editor los dos ficheros) o si no quiere volver a abrirlo, dándose en este caso el foco a la pestaña que lo contiene. Si el fichero no está abierto, simplemente se crea una nueva pestaña en el *notebook* y se rellenan los campos asociados al objeto de control de pestañas (mediante el método `addTab`).



```

1 # Generating synthetic images (with analytic function)
2
3 func "Common/ScriptGui"
4
5 if (!exists(SyntheticPVEImagesClass)) {
6   Class SyntheticPVEImagesClass : public ScriptGui{
7
8     #-----
9     proc ::Init() \
10    {
11      ::type2D = INT(0), "Analytic 2D function type"
12      ::type3D = INT(0), "Analytic 3D function type"
13      ::pv_type = INT(0), "Compute partial effect function"
14      ::subdiv = INT(0), "Subdivision level for recursive functions"
15      ::output_name= "synthetic_res", "Output image name"
16
17      ::reloadLine = UCHAR(0), "Reload the analytic line"
18      ::reloadCircle = UCHAR(0), "Reload the analytic circle"
19      ::reloadVessel2D = UCHAR(0), "Reload the analytic vessel2D"
20      ::reloadRing2D = UCHAR(0), "Reload the analytic ring"
21      ::reloadCube = UCHAR(0), "Reload the analytic cube"
22      ::reloadCylinder = UCHAR(0), "Reload the analytic cylinder"
23      ::reloadSphere = UCHAR(0), "Reload the analytic sphere"
24      ::reloadTorus = UCHAR(0), "Reload the analytic torus"
25
26      #Input image parameters
27      ::dimx = INT(100), "Input image X size"
28      ::dimy = INT(100), "Input image Y size"
29      ::dimz = INT(100), "Input image Z size"
30
31      #Circle parameters
32      ::cx = 50.0, "X component of the center of the circle"
33      ::cy = 50.0, "Y component of the center of the circle"
34      ::cr = 20.0, "Radius of the circle"
35
36      #Line parameters
37      ::line_angle = 45.0, "Angle of the line"
38      ::n = 50.0, "Y axis cut point"
39
40      #Straight Vessel 2D parameters
41      ::vessel_cx = 50.0, "X component of the center of the vessel"
42      ::vessel_cy = 50.0, "Y component of the center of the vessel"
43      ::vessel_angle = 45.0, "Angle of the vessel"
44      ::vessel_thick = 4.0, "Thickness of the vessel"
45
46      #Ring 2D parameters
47      ::ring_cx = 50.0, "X component of the center of the ring"
48      ::ring_cy = 50.0, "Y component of the center of the ring"
49      ::ring_radius = 30.0, "Ring internal radius"
50      ::ring_thick = 2.0, "Thickness of the ring"
51

```

Figura B.4: Detalle de la interfaz del editor de *scripts* de AMILab

De la misma manera, han de hacerse ciertas consideraciones cuando se va a cerrar un fichero. Evidentemente, si se hace uso del botón de cerrar de la propia pestaña o del menú, sólo ha de mirarse si ese fichero ha cambiado, lanzando la misma pregunta que en la antigua interfaz sobre si se desean guardar los cambios. Sin embargo, si se decide cerrar el editor por completo, es necesario ir recorriendo el vector de control de pestañas y comprobando si cada uno de los ficheros asociados ha cambiado; en cuyo caso se solicitará si se desean guardar los cambios o no. En cualquier caso, al cerrarse un fichero en la interfaz, ha de hacerse lo propio con el vector

de pestañas, eliminando esa entrada del mismo a partir de la operación `delTab`. Se puede ver un ejemplo de cómo es la interfaz en la figura B.4.

En cuanto a las nuevas funcionalidades implementadas con anterioridad no cambian, ya que se encuentran enlazadas a cada edición en concreto. Es decir, al realizar la operación de buscar y reemplazar o la de ir a la línea, éstas serán relativas a la pestaña que se encuentre activa en ese instante. Por tanto seguirán funcionando de la misma forma que había sido descrita.

B.2. Mejoras en el Visor

Otra de las labores realizadas a lo largo de este trabajo ha sido la modificación de algunos aspectos del visor de imágenes del software `AMILab`. Estos cambios comprenden básicamente dos puntos:

- Refactorización del código fuente: dentro del código existente se aplican técnicas de refactorización con el objetivo de aumentar la cohesión y disminuir el acoplamiento del mismo, de forma que cada funcionalidad aparezca de manera independiente y que el mantenimiento futuro pueda ser mucho más sencillo para el equipo de desarrollo.
- Mejoras en la interfaz: cambios de algunos aspectos como la integración de las ventanas de parámetros dentro del propio visor e inclusión de barras de herramientas (*toolbars*).

A continuación se pasan a describir ambos.

B.2.1. Refactorización del Código

De forma lógica, es inevitable pensar que el visor de imágenes es una parte fundamental del núcleo de `AMILab` desde su génesis. Si bien el software ha ido migrando de tecnología de interfaces (inicialmente estaba hecho con `Motif`, luego pasó a `wxMotif` hasta llegar a la adaptación total al actual interfaz con `wxWidgets`).

El visor sigue la estructura típica de `C++` en la que se cuenta con un fichero cabecera y otro implementación. Denominado como `DessinImage` hereda de `DessinImageBase`, un visor con unas funcionalidades muy básicas. Éste a su vez hereda de `FenetreDessin`. Esto puede verse reflejado en la figura B.5. `DessinImage` es la que implementa luego nuevas funcionalidades. Este código se había hecho por completo como parte de una misma clase, lo que producía un fichero bastante extenso y cuyo mantenimiento en caso de error o de querer introducir mejoras se hacía complicado.

Surge por tanto la idea de realizar una refactorización de este código con el objetivo de distribuir en diferentes clases las funcionalidades y mejorar así el mantenimiento, aumentando la cohesión y disminuyendo el acoplamiento. De esa manera nacen una serie de nuevas

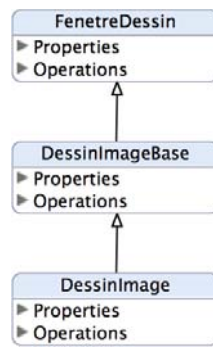


Figura B.5: Diagrama de clases de DessinImage

clases que pasarán a ser incluidas por `DessinImage`. Cada una de ellas será denominada teniendo como prefijo “`ImageDraw_`” seguido del nombre de la funcionalidad que contiene (p. ej. “`ImageDraw_PositionParam`”).

Así, en la fig. B.6, pueden verse las nuevas clases generadas. Éstas heredan de la clase `ParamPanel` ya que posteriormente se sustituirá su aparición en la interfaz en lugar de en una ventana independiente por un panel en la parte derecha del visor.

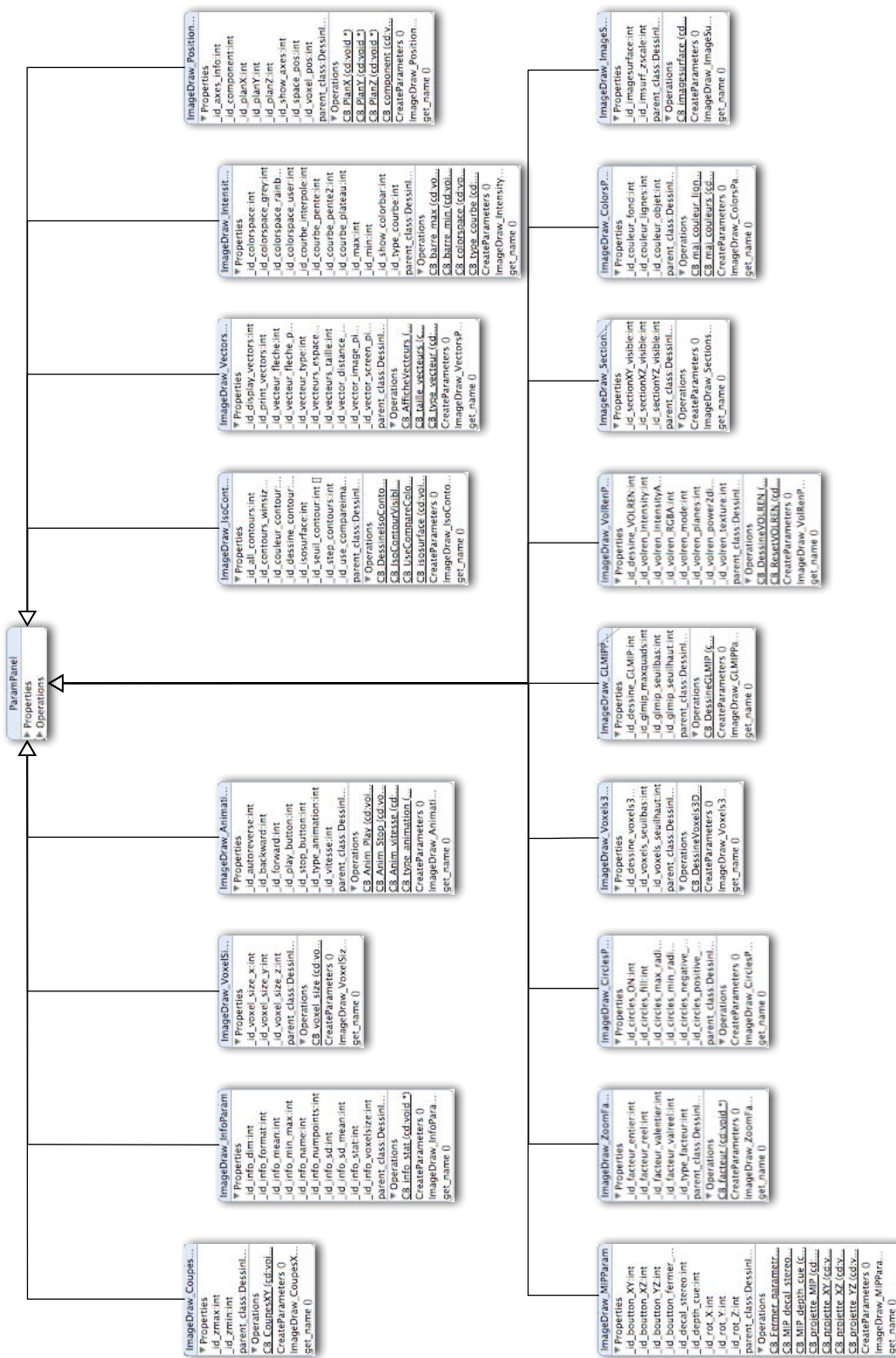


Figure B.6: Diagrama de las nuevas clases fruto de la refactorización de DessinImage

Para llegar a esta estructura se siguió un procedimiento sistemático, prácticamente igual para cada uno de los casos. La idea inicial era que la mayoría de opciones que se podían seleccionar desde los menús fueran operaciones independientes que estuvieran en un nuevo fichero. De esta manera se crearon un total de diecisiete nuevas clases, las cuales serían incluidas por `DessinImage`. Inicialmente era necesario localizar cuáles eran los atributos pertenecientes a cada una de las operaciones, así como sus funciones miembro. Aparte de ello, como en toda clase, es necesario añadir su constructor y destructor. Para poder permitir que `DessinImage` acceder a sus miembros, se hace que sea una clase *friend* de cada una de las clases `ImageDraw_*`.

Al igual que se ha descrito en otras secciones, además de crear estos ficheros, es preciso añadirlos al fichero `CMakeLists.txt` del directorio para que `CMake` pueda localizarlos. Recordar que, si se está trabajando con el entorno de desarrollo `XCode`, es preciso invocar a `CMake` para que reconfigure el fichero antes de volver a compilar.

B.2.2. Mejoras en la Interfaz

Previamente se ha dicho que el objetivo de esta mejora no era solamente hacer una refactorización del código, si no mejorar el aspecto actual del visor. En la figura B.7 se puede ver cual era el aspecto del visor en versiones anteriores del software. Puede observarse en la izquierda cómo es la visualización de la imagen y el resto de ventanas de parámetros que aparecen en pantalla se acceden a través de menús.

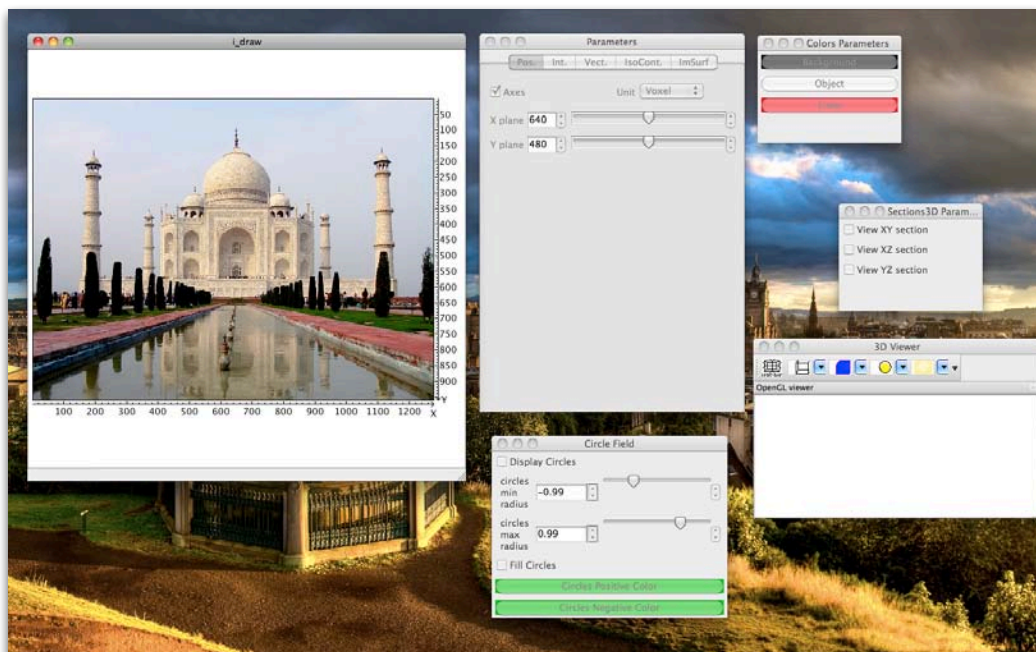


Figura B.7: Captura de pantalla del antiguo visor correspondiente a AMILab 2.0.4

Las mejoras que se quieren aplicar podrían resumirse en dos puntos:

- Adición de barras de herramientas en el visor (*toolbars*).
- Hacer que las ventanas de parámetros no aparezcan aparte si no como paneles a la derecha dentro del mismo visor.

Con ambas funcionalidades se pretende que las herramientas estén lo más a la vista posible del usuario así como evitar que existan una gran cantidad de ventanas desperdigadas por la pantalla, dando una mayor sensación de unicidad del visor, dando un aspecto más robusto y versátil.

Una vez realizados los cambios mencionados anteriormente y relacionados con la refactorización de código, las mejoras en cuanto a interfaz se llevan a cabo en el constructor de la clase `DessinImage`, donde se añadirán las barras de herramientas. Hay que tener en cuenta que para que esto sea posible, así como la aparición de los paneles a la derecha del visor con los parámetros, era necesario que la ventana del visor pasase a estar dentro de un entorno `wxAuiManager`, que es la clase central de `wxAUI`. Permite manejar los paneles asociados a él para un `wxFrame` particular. Además permite opciones como desplazar los paneles y colocarlos de una manera diferente si el usuario lo desea.

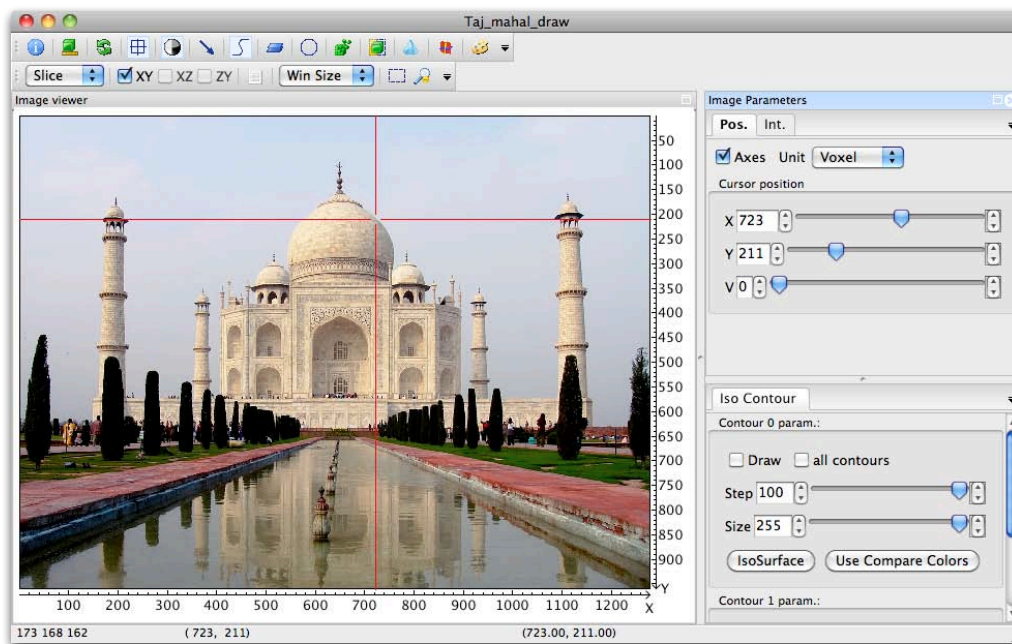


Figura B.8: Captura de pantalla del nuevo visor disponible desde la versión estable 3.0.0 de AMILab

Cada una de las nuevas clases creadas y que pueden verse en B.6, poseen una función miembro denominada `CreateParameters()`, que es la encargada de generar la interfaz asociada a cada una de las funcionalidades. Anteriormente, como es de suponer, este código estaba dentro de `DessinImage`, pero englobado dentro de funciones denominadas como `Cree` seguido del nombre de la funcionalidad en francés (recordemos que se trata de algo que forma parte de AMILab desde sus inicios). Esta operación ha de ser invocada adecuadamente desde el constructor de

cada clase para hacer que aparezcan las opciones que podrá modificar el usuario. Ya que cada una de esas clases pasó de crear un `ParamBox` (una ventana para los parámetros) a heredar de `ParamPanel`, ahora podrán ser integradas bajo el control de `wxAuiManager` y así tener todo en la misma vista integrado dentro de la misma ventana. En la figura B.8 se puede ver un ejemplo de cual es la configuración actual de la interfaz.

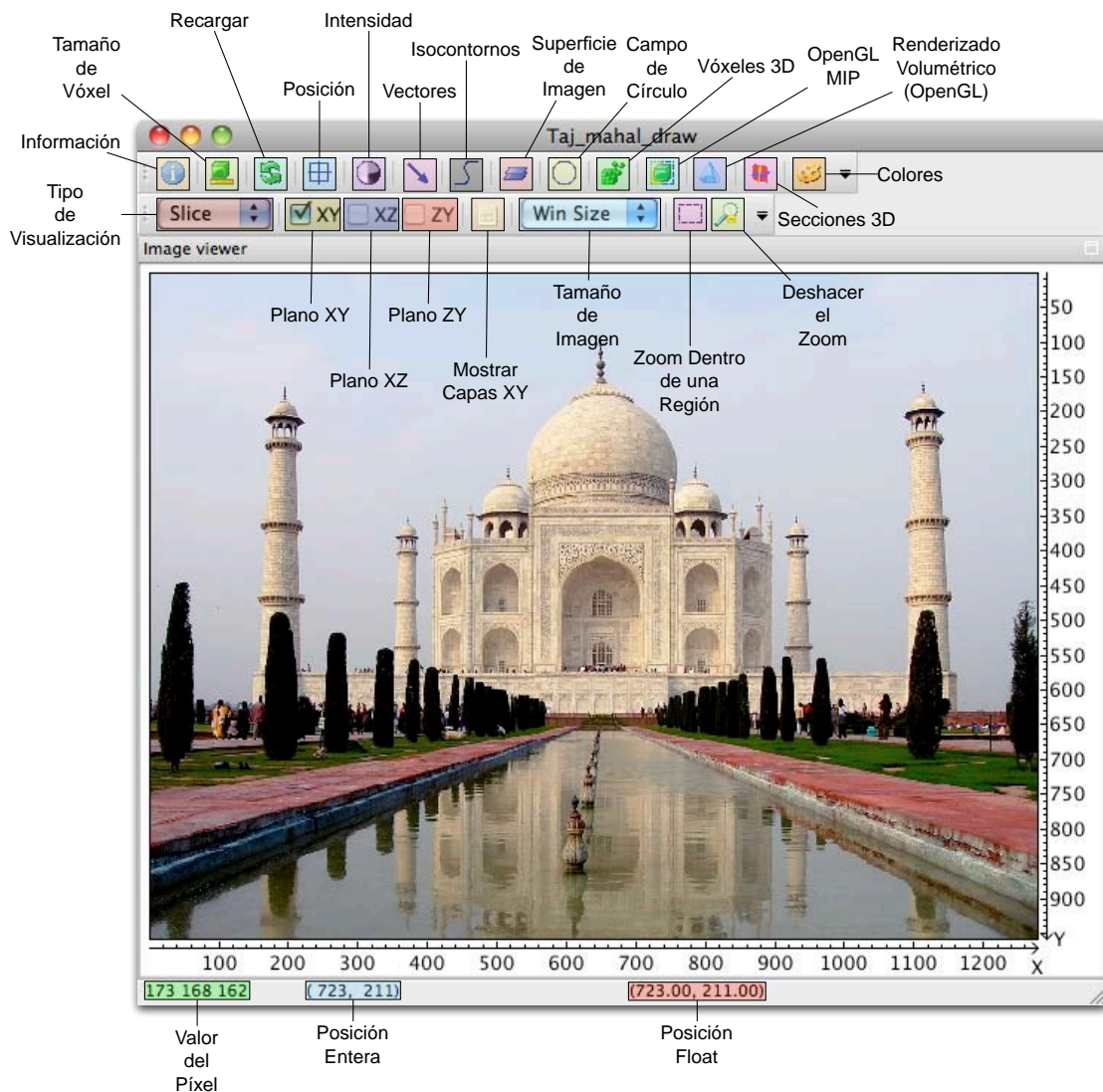


Figura B.9: Detalle de las características incorporadas por el visor de imágenes

En la figura B.9 pueden verse en detalle los cambios realizados a nivel de interfaz. En la barra de herramientas superior se encuentran la mayor parte de los parámetros que se pueden cambiar en una imagen como la posición en la que se encuentra el puntero o el rango de intensidad, así como poder ver información acerca de la misma (nombre, formato, dimensiones y cálculo de estadísticas). Además se incorporan funcionalidades como poder dibujar un campo de vectores sobre la imagen (el cual debe ser previamente proporcionado) o renderizado volumétrico de la misma (hecho sobre OpenGL no VTK).

La barra de herramientas inferior tiene que ver con la visualización. De izquierda a derecha,

se permite seleccionar en una lista entre ver las capas de la imagen (*Slice*), la máxima proyección de intensidad (*MIP*) o en animación (únicamente disponible para imágenes tridimensionales). Luego hay una serie de *checkboxes* para seleccionar qué planos se quieren tener visibles de la imagen, así como poder mostrar todo el conjunto de capas de una imagen 3D. Puede elegirse además que la imagen crezca proporcionalmente al tamaño de la ventana o en función de un valor entero que el usuario seleccione. Los dos últimos botones tienen que ver con el *zoom*. El primero de ellos es aquel que habilita el poder acercar la imagen a partir de una región rectangular dibujada con el ratón sobre la imagen. El último restaura la imagen a su *zoom* original.

En la parte inferior del visor, también conocida como barra de estado, se encuentran de izquierda a derecha el valor del píxel actual en el que se esté (este valor dependerá del tipo de imagen que se esté visualizando, en el caso del ejemplo que se muestra se pueden apreciar tres valores ya que es una imagen en escala RGB), la posición entera del cursor sobre la imagen y además la posición *float*.

Apéndice C

Detalles Implementación

En esta sección se darán algunos detalles acerca de la implementación del proyecto, con mayor profundidad que lo descrito en el cuerpo del documento. Principalmente se trata de relatar las cabeceras de las funciones principales, así como el significado de sus parámetros. La estructura que se seguirá es como si se estuviese en el interior de la clase, intercalando la definición de la misma con explicaciones. Nótese que, como se ha dicho, no aparecerán todas las operaciones, si no sólo las más importantes.

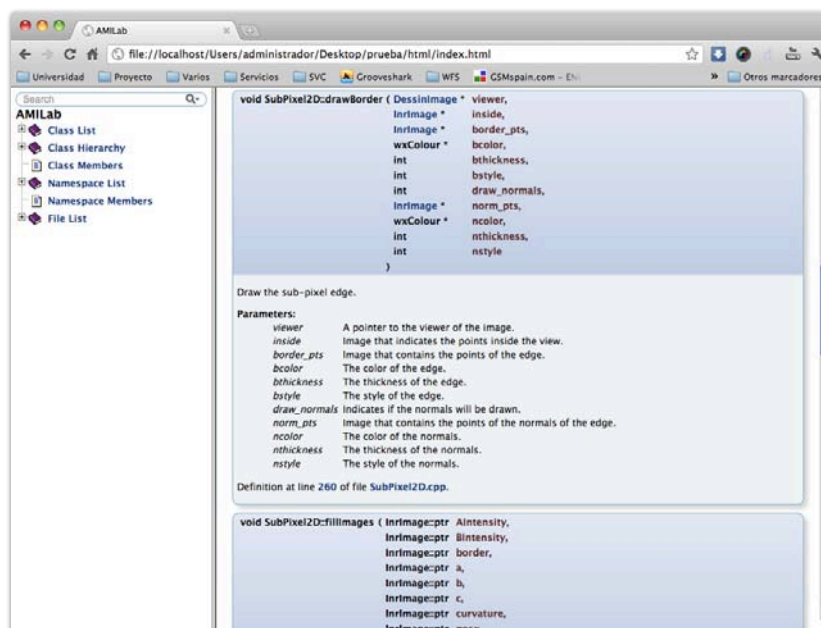


Figura C.1: Ejemplo de la documentación Doxygen generada

Antes de proceder a esta explicación, es preciso decir que todo el código desarrollado en C++ ha sido comentado haciendo uso de la herramienta **Doxygen**. Ésta, a través de unas directivas encerradas en comentarios parecidos a los del código fuente, permite dar detalles sobre el funcionamiento de los métodos, así como sus parámetros y el tipo de dato que devuelven. Esta

incorporación se hace en los ficheros cabecera `.h`. Una vez han sido introducidos los comentarios Doxygen, éste tiene un sistema que permite generar una documentación `html`, de fácil consulta. En la figura C.1 se puede ver un ejemplo de esta documentación, en la que aparece la descripción de la función `drawBorder` de la clase `SubPixel2D`. Para mayor detalle, puede acudir a esta documentación.

De la misma forma que se ha hecho en secciones anteriores, se dividirá la presente en el caso bidimensional y tridimensional.

C.1. 2D

C.1.1. Clase `borderPixel`

La clase `borderPixel` representa la información de un píxel que es parte de un borde. Es usada por la clase `SubPixel2D`

```
class borderPixel
{
```

Método encargado de asignar la información relativa a un píxel borde: valores de intensidad, tipo de borde, coeficientes de la parábola (a, b, c) , curvatura y posición (x, y) .

```
void setBorderPixelValues(double intA, double intB, unsigned char bord,
                        double coef_a, double coef_b, double coef_c,
                        double cu, int posx, int posy);
```

Atributos de la clase `borderPixel`. Se corresponden con los citados en el método anterior: intensidades, tipo de borde, coeficientes, curvatura y posición.

```
private:
double A;
double B;
unsigned char border;
double a;
double b;
double c;
double curvature;
int px;
int py;
};
```

C.1.2. Clase `SubPixel2D`

La clase `SubPixel2D` contiene los métodos para imágenes 2D. Además incluye el método que dibuja los bordes a nivel sub-píxel.

```
class SubPixel2D
{
```

Constructor de la clase `SubPixel2D`. Recibe por parámetro un puntero a la imagen de entrada (del tipo `InrImage` de `AMILab`), el umbral y la variable que indica si se intenta localizar un borde de primer orden.

```
SubPixel2D(InrImage* inp_image , float thres , int lc);
```

Método para dibujar los bordes a nivel sub-píxel. Recibe por parámetro un puntero al visor, una imagen que indica que puntos se están visualizando, una imagen que contiene las posiciones de los bordes, el color, grosor y estilo del borde. Además se indica si las normales serán dibujadas, apareciendo así la imagen que contiene las posiciones de las normales a los bordes, el color, grosor y estilo de las mismas.

```
void drawBorder(DessinImage* viewer , InrImage* inside ,
InrImage* border_pts , wxColour* bcolor ,
int bthickness , int bstyle , int draw_normals ,
InrImage* norm_pts , wxColour* ncolor ,
int nthickness , int nstyle);
```

Este procedimiento aplica el método básico para la detección de bordes con precisión sub-píxel en imágenes 2D.

```
void SuperGradienteCurvo ();
```

Este método aplica la detección sub-píxel en imágenes 2D afectadas por bajos niveles de ruido

```
void SuperGradienteGaussianoCurvo ();
```

Este procedimiento aplica la detección sub-píxel en imágenes 2D con ruido, usando una ventana dinámica o flotante, así como el estudio de los valores de las parciales tras dichos límites. Usa un esquema iterativo (restauración).

```
void SubpixelDenoising(int niter);
```

Basándose en el contenido del vector de los píxeles miembros de un borde, este procedimiento rellena algunas imágenes: intensidad A , intensidad B , el tipo de borde, coeficientes a, b, c de la parábola respectivamente, la curvatura y la posición (x, y) del píxel borde.

```
void fillImages(InrImage::ptr AIntensity , InrImage::ptr BIntensity ,
InrImage::ptr border , InrImage::ptr a , InrImage::ptr b ,
InrImage::ptr c , InrImage::ptr curvature ,
InrImage::ptr posx , InrImage::ptr posy);
```

Atributos de la clase `SubPixel2D`: imagen de entrada, imagen suavizada, vector de la STL que contiene la información de los píxeles borde haciendo uso de la clase `borderPixel`, el umbral y la indicación de caso lineal.

```
private:
    InrImage* input;
    InrImage* denoised;
    vector<borderPixel> borderPixelVector;
    double threshold;
    int linear_case;
};
```

C.2. 3D

C.2.1. Clase borderVoxel

La clase `borderVoxel` representa la información de un vóxel que es miembro del borde. Es usada por la clase `SubVoxel3D`.

```
class borderVoxel
{
```

Método encargado de asignar los valores referentes a la información de un vóxel borde: intensidad A y B , tipo de borde, coeficientes del paraboloide, curvatura, posición (x, y, z) dentro de la imagen y normal.

```
void setBorderVoxelValues(double intA, double intB, unsigned char bord,
                          double coef_a, double coef_b, double coef_c,
                          double coef_d, double coef_f, double coef_g,
                          double cu, int posx, int posy, int posz,
                          double gradx, double grady, double gradz);
```

Miembros de la clase `borderVoxel`, citados en el método anterior: intensidades, tipo de borde, coeficientes, curvatura, posición y normal al borde.

```
private:
    double A;
    double B;
    unsigned char border;
    double a;
    double b;
    double c;
    double d;
    double f;
    double g;
    double curvature;
    int px;
    int py;
    int pz;
    double nx;
```

```

    double ny;
    double nz;

};

```

C.2.2. Clase SubVoxel3D

La clase `SubVoxel3D` contiene los métodos para la detección con precisión sub-vóxel en imágenes 3D.

```

class SubVoxel3D
{

```

Constructor de la clase `SubVoxel3D`. Recibe por parámetro el puntero a una imagen de tipo `InrImage` (la imagen de entrada) y el valor del umbral.

```

    SubVoxel3D(InrImage* inp_image, float thres);

```

Este procedimiento aplica el método básico de detección con precisión sub-vóxel en imágenes 3D para contornos de segundo orden.

```

    void GradienteCurvo3D();

```

Basándose en el contenido del vector de los vóxeles miembros de un borde, este procedimiento rellena algunas imágenes: las intensidades A y B , las imágenes con los coeficientes del paraboloide, la curvatura, la posición (x, y, z) y las componentes de las normales a los bordes (nx, ny, nz) .

```

    void fillImages(InrImage::ptr AIntensity, InrImage::ptr BIntensity,
                  InrImage::ptr border, InrImage::ptr a, InrImage::ptr b,
                  InrImage::ptr c, InrImage::ptr d, InrImage::ptr f,
                  InrImage::ptr g, InrImage::ptr curvature, InrImage::ptr posx,
                  InrImage::ptr posy, InrImage::ptr posz,
                  InrImage::ptr nx, InrImage::ptr ny, InrImage::ptr nz);

```

Atributos de la clase `SubVoxel3D`: imagen de entrada, vector de la STL de tipo `borderVoxel` con la información de los vóxeles borde y valor del umbral.

```

private:
    InrImage* input;
    vector<borderVoxel> borderVoxelVector;
    double threshold;

};

```


Bibliografía

- [1] About mactex - <http://www.tug.org/mactex/2011/aboutmactex.html>.
- [2] Boost c++ libraries - http://en.wikipedia.org/wiki/Boost_C%2B%2B_Libraries.
- [3] Cmake - <http://en.wikipedia.org/wiki/CMake>.
- [4] Code::blocks - <http://en.wikipedia.org/wiki/Codeblocks>.
- [5] Gimp. programa de manipulación de imágenes de gnu. manual de usuario - <http://docs.gimp.org/2.6/es/introduction.html>.
- [6] Kdevelop4. manual. meet kdevelop - http://userbase.kde.org/Kdevelop4/Manual/Meet_KDevelop.
- [7] Mevislab - features - <http://www.mevislab.de/mevislab/features/>.
- [8] Online osirix documentation - about osirix - http://en.wikibooks.org/wiki/Online_OsiriX_Documentation/About_OsiriX.
- [9] Posix threads - <http://en.wikipedia.org/wiki/Pthreads>.
- [10] Slicer 3.6 manual - <http://www.slicer.org/slicerWiki/index.php/Documentation-3.6>.
- [11] Texmaker. summary - <http://www.xmlmath.net/texmaker/index.html>.
- [12] zlib - <http://en.wikipedia.org/wiki/Zlib>.
- [13] L. Álvarez, K. Krissian, and A. Trujillo. *Invariant Computation of Differential Characteristics in 3D Image*. Number 13. Cuadernos del Instituto Universitario de Ciencias y Tecnologías Cibernéticas, Universidad de Las Palmas de G.C., 2001.
- [14] J. Bucanek. *Beginning Xcode*. Programmer to Programmer Series. Wrox/Wiley Pub., 2006.
- [15] P. Capasso. *Terapéutica endovascular en la aorta torácica*. En: *Diagnóstico y terapéutica endoluminal. Radiología intervencionista*, pages 303–322. Masson, 2002.
- [16] C.W. Dawson and G.M. Quetglás. *El Proyecto Fin de Carrera en Ingeniería Informática. Una Guía para el Estudiante*. Prentice Hall, 2002.

- [17] Jose Luis Díaz and Javier Bezos. *The gloss Package*, Julio 2002.
- [18] Rafael C. González and Richard E. Woods. *Tratamiento Digital de Imágenes*. Addison - Wesley, 1996.
- [19] The Omni Group. *OmniGraffle 5 Professional*. The Omni Group, 2008.
- [20] Guillem Borrel i Nogueras. *Introducción a Matlab y Octave*, Marzo 2011.
- [21] Luis Ibáñez, Will Schroder, Lydia Ng, Josh Cates, and the Insight Software Consortium. *The ITK Software Guide*. Kitware, 2nd edition, 21 Noviembre, 2005.
- [22] Apple Inc. *A Tour of Xcode*. Apple Inc., 2009.
- [23] Apple Inc. *Xcode Debugging Guide*. Apple Inc., 2009.
- [24] Javier González Jiménez. *Visión por Computador*. Paraninfo, 2000.
- [25] Inc Kitware. *The VTK User's Guide. Install, Use and Extende The Visualization Toolkit*. Kitware, Inc., 11th edition, 2010.
- [26] Karl Krissian and Santiago Aja-Fernández. Noise-driven anisotropic diffusion filtering of mri. *IEE Transactions on Image Processing*, 18(10):2265–2274, Octubre 2009.
- [27] Leslie Lamport. *Latex: A Document Preparation System: User's Guide and Reference Manual*. Addison - Wesley, 2nd edition, 1996.
- [28] Tonya Limberg. *Osirix as a resource*. University of Illinois at Chicago, 2008.
- [29] F. Mittelbach, M. Goossens, J. Braams, and C. Rowley. *The Latex Companion*. Addison-Wesley series on tools and techniques for computer typesetting. Addison-Wesley, 2nd edition, 2004.
- [30] Ruben Olsen. *OmniGraffle 5 Diagramming Essentials. Create better diagrams with less effort using OmniGraffle*. Packt Publishing, Octubre 2010.
- [31] Juan Antonio Pérez Ortiz. *Diccionario Urgente de Estilo Científico del Español*, Marzo 1999.
- [32] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990.
- [33] I. Piper. *Learn Xcode Tools for Mac OS X and iPhone Development*. Learn Series. Apress, 2009.
- [34] B.C. Salinas. *El Libro de Latex*. Pearson Educación. Pearson Educación, 2003.
- [35] Will Schroder, Kenneth M. Martin, and William E. Lorenzen. *The Visualization Toolkit. An Object-Oriented Approach to 3D Graphics*. Kitware, Inc., 4th edition, 2006.
- [36] William J. Schroeder, Kenneth M. Martin, and William E. Lorenzen. The design and implementation of an object-oriented toolkit for 3d graphics and visualization. *Visualization Conference, IEEE*, 0:93, 1996.

- [37] Julian Smart, Kevin Hock, and Stefan Csomor. *Cross-Platform GUI Programming with wxWidgets*. Prentice Hall, 16 Julio 2005.
- [38] M.D. Spivak. *The Joy of T_EX. A Gourmet Guide to Typesetting with the AMS-T_EX macro package*. American Mathematical Society, 1990.
- [39] Mosby Publishing Staff. *Diccionario Mosby. Medicina, Enfermería y Ciencias de la Salud*. Harcourt, 2000.
- [40] SCIRun Development Team. *SCIRun User Guide*. Center for Integrative Biomedical Computing. Scientific Computing and Imaging Institute. University of Utah.
- [41] A. Trujillo-Pino, K. Krissian, D. Santana-Cedrés, J. Esclarín-Monreal, and J.M. Carreira-Villamor. A subpixel edge detector applied to aortic dissection detection. In R. Moreno-Díaz et al., editor, *EUROCAST 2011, Part II, LCNS 6928*, pages 217–224. Springer-Verlag Berlin Heidelberg, 2011.
- [42] Agustín Trujillo Pino. *Localización de contornos con precisión sub-píxel en imágenes bidimensionales y tridimensionales*. PhD thesis, Universidad de Las Palmas de Gran Canaria, 2004.
- [43] YellowBrain. Documentación de referenica del wrapping del editor scintilla en wxwidgets - <http://www.yellowbrain.com/stc/index.html>.

Glosario

A

anisotrópicos Se denominan filtros de difusión anisotrópica a aquellos que varían su comportamiento en función de la dirección. En procesamiento digital de imágenes, estos métodos usan un término de regularización que tiende a uniformizar la imagen en las zonas homogéneas, dejando intactos los bordes bien definidos (la difusión no es igual en todos los píxeles), pág. 59.

API Application Programming Interface (Interfaz de Programación de Aplicaciones). Es un conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizadas por otro software como una capa de abstracción, pág. 30.

C

convolución La convolución es una operación que modifica el nivel de gris de los píxeles de la imagen teniendo en cuenta los píxeles de su entorno de vecindad, pág. 48.

CT Viene de Computed Tomography (Tomografía Computerizada en español). También conocida como TAC o por escáner, es una técnica de diagnóstico utilizada en medicina. Es una tecnología de exploración de rayos X que produce imágenes detalladas de cortes axiales del cuerpo, pág. 29.

D

DEFLATE El algoritmo de deflación, en inglés denominado DEFLATE, es un algoritmo de compresión de datos sin pérdidas que usa una combinación del algoritmo LZ77 y la codificación Huffman, pág. 31.

F

función de transferencia Una función de transferencia es un modelo matemático que a través de un cociente relaciona la respuesta de un sistema (modelada) a una señal de entrada o excitación (también modelada). En el caso concreto de **AMILab** la función

de transferencia se usa en la representación de imágenes tridimensionales a través de renderizado volumétrico, como una curva que se desplaza sobre el histograma. Sobre el eje x se encuentran representados los valores de intensidad y en el eje y la opacidad de dichos valores, pág. 116.

G

GCC GNU Compiler Collection es un conjunto de compiladores creados por el proyecto GNU. GCC es software libre y lo distribuye la FSF (Free Software Foundation) bajo la licencia GPL, pág. 19.

GNOME Es un entorno de escritorio e infraestructura de desarrollo para sistemas operativos Unix y sus derivados como GNU/Linux, BSD o Solaris; compuesto enteramente de software libre, pág. 20.

GNU El proyecto GNU fue iniciado por Richard Stallman con el objetivo de crear un sistema operativo completamente libre: El sistema GNU. Es un acrónimo recursivo que significa No es Unix (GNU is Not Unix), pág. 20.

GPL General Public License (Licencia Pública General de GNU). Es una licencia creada por la Free Software Foundation en 1989 (la primera versión) y está orientada principalmente a proteger la libre distribución, modificación y uso de software, pág. 20.

GTD Getting Things Done, cuyas siglas son GTD, es un método de gestión de las actividades. GTD se basa en el principio de que una persona necesita borrar de su mente todas las tareas que tiene pendientes, guardándolas en un lugar específico. De este modo, se libera a la mente del trabajo de recordar todo lo que hay que hacer, y se puede concentrar en la efectiva realización de aquellas tareas, pág. 14.

GUI Graphical User Interface (Interfaz Gráfica de Usuario). Es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo, pág. 30.

I

IDE Integrated Development Environment (Entorno de Desarrollo Integrado). Es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien, poder utilizarse para varios, pág. 20.

isotrópicos Se denominan filtros de difusión isotrópica a aquellos que tienen siempre el mismo comportamiento independientemente de la dirección. En procesado digital de imágenes, en cada iteración, cada píxel se actualiza a un valor intermedio entre el valor inicial y el valor promedio de su vecindad, pág. 59.

K

KDE Es un proyecto de software libre para la creación de un entorno de escritorio e infraestructura de desarrollo para diversos sistemas operativos, pág. 20.

M

Mac OS X Es un sistema operativo desarrollado y comercializado por Apple Inc que ha sido incluido en su gama de ordenadores Macintosh desde 2002, pág. 19.

MRI Magnetic Resonance Image (IMR, siglas en español de Imagen por Resonancia Magnética). Es una técnica no invasiva que utiliza el fenómeno de resonancia magnética para obtener información sobre la estructura y composición del cuerpo a analizar. Esta técnica es principalmente utilizada en medicina para observar alteraciones en los tejidos y detectar cáncer y otras patologías, pág. 29.

MSVC Microsoft Visual C++ es un entorno de desarrollo integrado para lenguajes de programación C y C++. Está especialmente diseñado para el desarrollo y depuración de código escrito para las API's de Microsoft Windows, DirectX y la tecnología Microsoft .NET Framework, pág. 19.

muestreo El muestreo (en inglés, sampling) consiste en tomar muestras periódicas de la amplitud de onda. La velocidad con que se toma esta muestra, es decir, el número de muestras por segundo, es lo que se conoce como frecuencia de muestreo, pág. 52.

P

Píxel Es el acrónimo de “picture element” y representa la unidad mínima de una imagen en dos dimensiones. Cada píxel se codifica mediante un conjunto de bits de longitud determinada, a la que se denomina profundidad de color. Por ejemplo, en el modelo RGB se suelen usar 8 bits para cada una de las componentes (roja, verde y azul), pág. 2.

PVE Es el acrónimo de Partial Volume Effect, efecto de volumen parcial en castellano. Se describe como el efecto que se produce por la mezcla de las intensidades correspondientes a dos o más regiones coincidentes en un mismo vóxel o píxel, pág. 52.

S

STL La Standard Template Library es una biblioteca de componentes genéricos para C++ estandarizada. Su nombre viene por el uso de clases template. La biblioteca está basada en un diseño ortogonal para permitir combinar cada componente con cualquier otro, proporcionando una gran expresividad y flexibilidad. La biblioteca contiene los siguientes tipos de componentes principales: Contenedores, iteradores, algoritmos, objetos función y adaptadores, pág. 82.

V

Vóxel Viene de “volumetric pixel” y es la unidad cúbica mínima que compone una imagen tridimensional, equivalente al píxel en 2D. Para crear una imagen en tres dimensiones, los vóxeles tienen que sufrir una transformación de opacidad. Esta información da diferentes valores de opacidad a cada vóxel. Esto es importante cuando se han de mostrar detalles interiores de una imagen que quedarían tapados por la capa exterior más opaca de vóxeles. Las imágenes con vóxeles se usan generalmente en el campo de la medicina y se aplican, por ejemplo, en la tomografía axial computarizada o para las resonancias magnéticas; aunque ya se extienden a otras disciplinas como la ingeniería, el cine o los videojuegos, pág. 2.

W

WYSIWYG Es el acrónimo de What You See Is What You Get (lo que ves es lo que obtienes). Se aplica a los procesadores de texto y otros editores de texto con formato que permiten escribir un documento viendo directamente el resultado final, frecuentemente el resultado impreso, pág. 32.