ADAPTACIÓN, PARA LA LIBRERÍA DE CÓDIGO LIBRE INSIGHT TOOLKIT, DE UN MÉTODO AUTOMÁTICO DE SEGMETACIÓN DE VASOS CORONARIOS EN IMÁGENES DE TOMOGRAFÍA COMPUTARIZADA

Sara Arencibia García

Facultad de Informática.
Universidad de Las Palmas de G.C.

Facultad de Informática. Universidad de Las Palmas de G.C.

0.1. Proyecto fin de carrera

Título: Adaptación, para la librería de código libre Insight Toolkit, de un método automático de segmentación de vasos coronarios en imágenes de tomografía computarizada
Apellidos y nombre del alumno: Arencibia García, Sara
Fecha : Julio 2010

Tutores:
Krissian, Karl
Álvarez León, Luis
Esclarín Monreal, Julio

Facultad de Informática. Universidad de Las Palmas de G.C.

0.2. Agradecimientos

Aqui ponemos los agradecimientos

Facultad de Informática. Universidad de Las Palmas de G.C.

Índice general

Facultad de Informática. Universidad de Las Palmas de G.C.

Capítulo 1

Introducción

Capítulo 2

Estado actual del tema

Este proyecto se basa en la tomografía computarizada, sin la cual ninguno de los algoritmos implementados tendría sentido. Es por ello, que es importante abordar este tema y comprender la importancia que tiene hoy en día.

La tomografía computarizada es un método de formación de imágenes médicas utilizando la tomografía. Se utiliza el procesamiento de la geometría digital para generar imágenes en 3D del interior de un objeto partiendo de una gran serie de imágenes en 2D de rayos X tomadas en torno a un único eje de rotación. Combina el uso del ordenador con la rotación del dispositivo de rayos X para crear imágenes detalladas de la sección transversal o capas de los diferentes órganos y partes del cuerpo como los pulmones, hígado, riñones, páncreas, pelvis, extremidades, cerebro, columna vertebral y los vasos sanguíneos.

Entre las diversas técnicas de formación de imágenes como la resonancia magnética y los rallos X, la tomografía computarizada es la única que posee la habilidad de combinar tejidos blandos, huesos y vasos sanguíneos. Por ejemplo, la formación de la imagen de la cabeza mediante rayos X sólo puede mostrar la densidad de las estructuras del los huesos del cráneo. La angiografía por rayos X de la cabeza sólo es capaz de representar los vasos sanguíneos de la misma y del cuello, pero no los tejidos blandos del cerebro. La resonancia magnética hace un excelente trabajo mostrando los tejidos blandos y los vasos sanguíneos, pero no es capaz de dar en detalle las estructuras óseas. La tomografía computarizada de la cabeza permite a los médicos ver los tejidos blandos como los ventrículos del cerebro o la materia gris y blanca. Mediante esta técnica los médicos pueden elegir ver los tejidos blandos, las estructuras óseas o los vasos sanguíneos.

Por tanto la tomografía computarizada es:

- Una de las herramientas mejores y más rápidas para el estudio del pecho, abdomen y pelvis porque proporciona vistas detalladas y secciones transversales de todos los tipos de tejidos.

- A menudo es el método más adecuado para el diagnóstico de muchos tipos de cánceres, incluidos los de pulmón, hígado y cáncer de páncreas, ya que la imagen obtenida permite al médico confirmar la presencia del tumor, medir su tamaño, ubicación exacta y grado del mismo ayudándose para ello de los tejidos cercanos.

- Un examen que desempeña un papel importante en la detección, diagnóstico y tratamiento de enfermedades vasculares que pueden conducir a accidentes cerebro vasculares, insuficiencia renal o incluso la muerte. La tomografía computarizada se utiliza comúnmente para evaluar la embolia pulmonar así como para las aneurismas de la aorta abdominal.

- Inestimable en el diagnóstico y tratamiento de problemas de columna vertebral y lesiones en las manos, pies y otras estructuras esqueléticas porque incluso puede mostrar claramente huesos muy pequeños, así como los tejidos circundantes, tales como el músculo y los vasos sanguíneos.

Los médicos a menudo utilizan la tomografía computarizada para:

- Identificar rápidamente las lesiones de los pulmones, corazón, vasos sanguíneos, hígado, bazo, riñones y otros órganos internos en los casos de traumatismo.

- La guía de las biopsias y otros procedimientos, como drenajes de abscesos y tratamientos del tumor mínimamente invasivos.

- Planificar y evaluar los resultados de la cirugía.

- Planificar y administrar adecuadamente los tratamientos de radiación para tumores.

- Medir la densidad mineral ósea para la detección de la osteoporosis.

A día de hoy el tratamiento de imágenes de tomografía computarizada está muy extendido, siendo muchos los que se han decidido a abordar este tema. A continuación se citan aquellos que al igual que la librería elegida para el desarrollo de este proyecto se centran en el tratamiento de imágenes promoviendo el uso del software libre.

2.1. MITK

“ Medical Imaging Interaction Toolkit ” es un software libre para el desarrollo de software para el procesamiento interactivo de imágenes médicas. MITK combina ITK y VTK con las librerías basadas en PIC de DKFZ. MITK ofrece un conjunto de características relevantes para el desarrollo de software interactivo para imágenes médicas que ITK y VTK no poseen, como:

- Múltiples y coherentes vista de la misma información.
- Un concepto interactivo basado en los estados de las máquinas, que le ayuda a estructurar mecanismos de interacción complejos.
- Un concepto de deshacer /rehacer para las interacciones.
- Organiza toda la información de la aplicación en un repositorio jerárquico y central. La jerarquía permite representar relaciones lógicas (como que un ventrículo es una parte del corazón).
- Descripción de los datos por propiedades arbitrarias para la comunicación entre módulos del programa o para el control de renderización.

MITK reutilizada prácticamente todo de VTK e ITK. Por lo tanto, no es en absoluto un competidor para VTK o ITK, sino una extensión, que trata de facilitar la combinación de ambos y añadir las características antes mencionadas. Aunque es principalmente una guía y no una aplicación, MITK ofrece apoyo en los niveles de aplicación, por ejemplo, para la combinación estructurada de módulos, o para combinar y cambiar entre una funcionalidad para la segmentación y otra para la correspondencia.

2.2. ITK (librería elegida)

ITK es software libre de herramientas para la realización de correspondencias y segmentación en dos, tres o más dimensiones (análisis de imágenes).

La segmentación es un proceso de identificación y clasificación de la información encontrada en una representación digital de una muestra. Normalmente la representación de la muestra es una imagen adquirida mediante instrumentación médica como escáneres de tomografía computarizada (CT) o de resonancia magnética (MRI). La correspondencia es la tarea de alinear o desarrollar las correspondencias entre datos. Por ejemplo, en un entorno médico, un escáner CT puede ser alineado con un escáner MRI a fin de combinar la información contenida en ambos.

Debido a que ITK es un proyecto de código abierto, los desarrolladores de todo el mundo pueden utilizar, comprobar, mantener y extender el software. ITK utiliza un modelo de desarrollo de software llamado programación extrema. La programación extrema cambia la metodología habitual de desarrollo software por un proceso iterativo y simultaneo de diseño-implementación-prueba-versión. La clave de esta programación extrema es la comunicación y la prueba. La comunicación entre los miembros de la comunidad de ITK es la que ayuda a manejar la rápida evolución del software. La prueba es la que mantiene el software estable. En ITK, un extenso proceso de pruebas día a día son los que determinan la calidad.

2.3. VTK

VTK es un software libre para gráficos 3D, modelado, procesamiento de imágenes, renderización de volúmenes, visualización científica y visualización de la información.

VTK es usado en todo el mundo en aplicaciones comerciales, de investigación y de desarrollo y es la base de muchas aplicaciones de visualización avanzada como: ParaView, VisIt, VisTrails, Slicer, MayaVi, and OsiriX.

2.4. IGSTK

IGSTK es un componente de alto nivel basado en un framework que proporciona funcionalidades comunes para aplicaciones de cirugía guiada por imágenes. Este framework está basado en un conjunto de componentes de alto nivel integrados con otras librerías de bajo nivel y código abierto y con interfaces de programación de aplicaciones (API) de los vendedores de hardware.

Lo más importante de IGSTK es la robustez. IGSTK proporciona las siguientes funcionalidades de alto nivel: habilidad para leer y mostrar imágenes médicas incluyendo CT y MRI en formato DICOM; una interfaz común para el seguimiento de hardware; una interfaz gráfica de usuario y capacidad de visualización incluyendo vistas en los cuatro cuadrantes (axial, sagital, coronal y 3D) así como una vista axial de múltiples capas; correspondencia: el punto de correspondencia y un medio para seleccionar estos puntos; servicios para logging, manejo de excepciones y resolución de problemas.

2.5. VTKEdge

VTKEdge es una librería de visualización avanzada y de técnicas de procesamiento de datos que complementan a VTK, así como módulos propios que permiten el uso de estas técnicas con ParaView. VTKEdge no reemplaza a VTK, sino que se compila con VTK para proporcionar funcionalidades adicionales.

Las enfermedades de los vasos coronarios es una de las causas de muerte más frecuentes, es por ello que la detección de vasos a partir de imágenes de tomografía computarizada es un método de gran ayuda a nivel médico para la detección de dichas enfermedades. Partiendo de este hecho se ha realizado por parte de un equipo de investigadores un método de detección de vasos coronarios, intentando obtener la línea media de los mismos lo más realista posible. Esto permite la visualización, planificación de operaciones y la segmentación sobre los mismos.

A día de hoy el método escogido para la obtención de la línea media de los vasos coronarios compite con otros 13 métodos. A parte de lo comentado anteriormente, es de vital importancia tener un software de visualización de imágenes que ayude al diagnóstico de las posibles enfermedades existentes. Es por ello que se ha creado un software para el tratamiento de imágenes, denominado AMILab.

Capítulo 3

Objetivos

El proyecto que se plantea viene motivado por la necesidad de implementar un algoritmo de segmentación de vasos coronarios para código libre.

Para la realización de este objetivo se partirá de un método de segmentación de vasos coronarios (Minimally Interactive Knowledge-based Coronary Tracking in CTA using a Minimal Cost Path), de las librerías de código abierto Insight Toolkit y del software para tratamiento de imágenes AMILab.

Lo que se plantea es el paso de la totalidad o parte del método de segmentación de vasos coronarios seleccionado a las librerías de código abierto Insight Toolkit, permitiendo de esta forma la divulgación de este método a través de estas librerías de programación. Una vez realizada la implementación del método seleccionado se pretende incluir dicha herramienta en el software de tratamiento de imágenes AMILab.

Por tanto, como objetivos principales de este proyecto se pretende:

- Adaptación de la totalidad o parte del método de segmentación de vasos coronarios a las librerías de código abierto Insight Toolkit.
- Inclusión de la herramienta desarrollada en el software de tratamiento de imágenes AMILab.

Capítulo 4

Metodología

Capítulo 5

Recursos necesarios

En el desarrollo de este proyecto se pretende, en la medida de lo posible, la utilización de software libre. Por lo que, para la implementación del mismo se utilizará como S.O. Linux, en su distribución Ubuntu y Fedora. Por tanto, los programas utilizados serán todos de software libre.

Está elección viene motivada por el carácter que se pretende dar al desarrollo del proyecto, que en todo momento será orientado al software libre.

Otro detalle a tener en cuenta serán los recursos hardware necesarios para la fluidez del desarrollo del proyecto y la validación del mismo. Debemos recordar que los algoritmos implementados serán aplicados sobre imágenes de tomografía computarizada que ocupan un gran espacio en disco duro. Por lo que será necesaria la presencia de un dispositivo de almacenamiento de gran capacidad y un ordenador que sea capaz de procesar de manera eficiente y potente todo este material. Otro detalle a tener en cuenta es la proveniencia del material utilizado para la validación de los algoritmos implementados. Se utilizará una base de datos de 32 pacientes creada por el workshop "3D Segmentation in the Clinic: A Grand Challenge II", teniendo especial cuidado en cumplir todas las bases legales para su utilización.

Una vez comentados estos puntos pasaremos a numerar el software y el hardware usado para este proyecto:

- Software
  - Cmake
  - Librerías Insight Toolkit
  - Compilador de C++
  - AMILab
  - VTK

- Subversion

- Hardware
  - Disco Duro cómo mínimo de 200GB
  - Portátil Centrino Duo con 1GB de RAM

Capítulo 6

Plan de trabajo y temporización

A continuación se plantea la temporización del proyecto desglosada en etapas y en horas. Se debe tener en cuenta que este plan de trabajo es una estimación, por lo que podrá sufrir variaciones a lo largo del desarrollo del proyecto.

6.1. Análisis

En una primera etapa se hará una fase de análisis sobre las herramientas a utilizar, profundizando de manera exhaustiva en el manual de las librerías a utilizar (Insight Toolkit). Esto permitirá mayor soltura a la hora del diseño e implementación de las nuevas clases requeridas. Para la realización de esta etapa se desarrollarán varias tareas que ayudarán a una mejor asimilación de los conceptos, como son:

- Análisis de ejemplos simples.
- Análisis de ejemplos complejos.

A continuación, se podrá realizar un estudio del software de tratamiento de imágenes AMILab. Para conocerlo a fondo se seguirán los tutoriales disponibles y se ejecutarán los scripts, comprobando y entendiendo en todo caso los resultados obtenidos.

Como segunda etapa de esta fase se realizará un análisis de las necesidades de las clases a implementar. Para ello se deberá estudiar las relaciones que existirán entre clases, tanto entre las creadas como entre las nuevas y las creadas. Para toda esta fase de análisis se necesitarán las siguientes horas de trabajo:

- Familiarización con la librería Insight Toolkit $\rightarrow$ 63h

- Familiarización con los algoritmos existentes y AMILab → 60h
- Análisis de Ejemplos Complejos → 60h
- Generación de la documentación → 20h
- Primera fase de análisis → 60h
- Generación de la documentación de esta primera fase → 20h
- Segunda fase de análisis → 60h
- Generación de la documentación de esta segunda fase → 20h

6.2. Diseño

En esta fase se pretende determinar el diseño de las nuevas clases a implementar, teniendo en cuenta la fase de análisis previa y el diseño por defecto de las librerías Insight Toolkit.

Las horas de trabajo vendrán determinadas por:

- Primera fase de diseño → 60h
- Generación de documentación de esta primera fase → 20h
- Segunda fase de diseño → 60h
- Generación de documentación de esta segunda fase → 20h

6.3. Implementación

Durante la fase de implementación se desarrollarán las clases propiamente dichas. Esta fase vendrá regida por las fases de análisis y diseño estudiadas anteriormente.

Las horas de trabajo fijadas son:

- Primera fase de implementación → 60h
- Generación de documentación de esta primera fase → 20h
- Segunda fase de implementación → 60h
- Generación de documentación de esta segunda fase → 20h

6.4. Validación y publicidad del PFC

La última fase del proyecto es una de las más importantes ya que en ella se realizarán los test que validarán las clases implementadas. Además se creará una página web para la difusión del material desarrollado.

Dentro de los test de validación tendremos varias partes, siendo las siguientes:

- Definición de los test de validación.
- Aplicación de los test de validación.
- Análisis de los resultados obtenidos, los cuales serán cotejados con los ya existentes. Estos provienen del método del que se parte para la realización de este proyecto.

Por tanto, la planificación de esta fase queda de la siguiente forma:

- Test de validación → 90h
- Generación de la documentación → 20h
- Confección de manuales y desarrollo de la página web → 80h

Capítulo 7

Análisis

7.1. Herramientas

En esta primera fase del análisis se pretende dar una visión global de las herramientas a utilizar, sus principales características, ventajas e inconvenientes. Partiremos en un primer lugar de la herramienta fundamental en la que se basa todo el desarrollo de este proyecto, las librerías de código abierto Insight Toolkit.

7.1.1. Insight Toolkit

Introducción

Las librerías Insight Toolkit son de código abierto, multiplataforma, orientadas a objetos y se utilizan para procesar, segmentar y hacer correspondencia de imágenes. Insight Toolkit está diseñado para ser usado con facilidad una vez que se aprende diseño orientado a objetos y su metodología de implementación. A su vez, Insight Toolkit proporciona métodos novedosos y a la orden del día para segmentación y correspondencia de imágenes en cualquier dimensión posible (2D, 3D,.., ND).

Las principales metas de Insight Toolkit son:

- Soportar el proyecto "Visible Human" (NLM).
- Crear una fundación para la investigación futura.
- Crear una fundación para la investigación futura.
- Crear un repositorio de algoritmos esenciales.
- Desarrollar una plataforma para productos de desarrollo avanzados.
- Apoyo comercial a la aplicación de la tecnología.
- Crear convenciones para trabajos futuros.
- Crecer como una comunidad que se alimenta del software de usuarios y desarrolladores.

Instalación

En este apartado se dará una pequeña visión de lo que se necesita para la correcta instalación de Insight Toolkit en nuestra máquina.

Recordemos que Insight Toolkit es multiplataforma por lo que para poder utilizarla en cualquier plataforma deberemos usar CMake (software libre). Este software controla el proceso de compilación usando una plataforma y un compilador de archivos de configuración independiente. CMake genera los "makefiles" y los espacios de trabajo que podrán ser usados en el entorno de compilación que se haya elegido. Es bastante sofisticado ya que da soporte a entornos complejos que requieren una configuración del sistema, testeo de características del compilador y generación de código.

CMake genera "makefiles" bajo Unix y sistemas Cygwin. Bajo Windows generará espacios de trabajo para Visual Studio. La información usada por CMake es proporcionada por los archivos "CMakeLists.txt" que están presentes en todos los directorios del código fuente de Insight Toolkit. Estos archivos contienen información que el usuario proporciona a CMake en tiempo de configuración. La información más común es la inclusión de rutas a las utilidades en el sistema y la selección de opciones del software especificadas por el usuario.

Una vez tengamos instalado CMake pasaremos a configurar Insight Toolkit mediante el uso del mismo. Cuando Insight Toolkit esté perfectamente instalado en nuestra máquina podremos empezar a utilizar las librerías. Para ello, la forma más fácil de crear un nuevo proyecto con Insight Toolkit es crear un directorio nuevo en el que tendremos dos nuevos archivos. Uno de ellos será el "CMakeLists.txt" que será usado por CMake para generar el "makefile" bajo Unix o un espacio de trabajo para Visual Studio si trabajamos desde Windows. El otro archivo a usar será el algoritmo en C++ que hará uso de las clases disponibles en Insight Toolkit.

Para una mejor comprensión y entendimiento de cómo instalar Insight Toolkit se recomienda estudiar el manual propio de ITK [1].

Organización

Insight Toolkit está formado por varios subsistemas:

- Conceptos esenciales del sistema → como cualquier sistema software, Insight Toolkit está basado en torno a algunos conceptos básicos de diseño. Algunos de los conceptos más importantes son la programación genérica, punteros inteligentes para el manejo de memoria, fábricas de objetos, manejo de eventos usando el paradigma de diseño vista controlador y soporte multihilo.

- Numéricos → Insight Toolkit utiliza la librería numérica VXL´s VNL.

- Representación de datos y acceso → existen dos clases principales para representar datos: itk::Image y itk::Mesh. Además, varios tipos de iteradores y contenedores son usados para mantener y recorrer los datos. Otra clase importante es aquella para representar histogramas.

- Procesamiento de datos del pipeline → las clases de representación de datos (conocidas como objetos de datos) son operadas por filtros que sucesivamente pueden ser organizadas en flujos de datos del pipeline. Estos pipelines mantienen estados y por lo tanto se ejecutan sólo cuando es necesario. También soportan multihilos y streaming (se puede operar sobre parte de los datos para minimizar el uso de memoria).

- IO Framework → asociado con el procesamiento de datos del pipeline tenemos:

  - "Sources": filtros que inicializan el pipeline.
  - "Mappers": filtros que terminan el pipeline.

    Los típicos ejemplos de "sources" y "mappers" son lectores y escritores respectivamente. Los lectores introducen datos (típicamente desde un archivo), y los escritores extraen datos desde el pipeline.

- Objetos espaciales → las formas geométricas están representadas en Insight Toolkit usando una jerarquía de objetos espaciales. Estas clases están destinadas a soportar el modelado de las estructuras anatómicas. Utilizando una interface básica común, los objetos espaciales son capaces de representar regiones del espacio en una variedad de formas diferentes. Por ejemplo: estructuras de mallas, máscaras y ecuaciones implícitas pueden ser usadas como esquemas de representación subyacentes. Los objetos espaciales son una estructura de datos para comunicar los resultados de los métodos de segmentación y para introducir priores anatómicos tanto en los métodos de segmentación como en los de correspondencia de imágenes.

- Framework de correspondencia → un framework flexible para la correspondencia soporta cuatro tipos de correspondencia: correspondencia de imágenes, correspondencia con multi-resolución, correspondencia basada en PDE y correspondencia FEM (método de elementos finitos).
- Framework FEM → Insight Toolkit incluye un subsistema para resolver problemas FEM, en particular la correspondencia no rígida. El paquete FEM incluye definición de mallas, cargas y condiciones límite.
- Level Set Framework → el framework de colección de nivel es una colección de clases para crear filtros para resolver ecuaciones diferenciales parciales sobre imágenes utilizando un plan de actualización de diferencias finito e iterativo. El framework de colección de nivel consiste en resolver diferencias finitas incluyendo un solucionador bayesiano, una colección de nivel genérica de filtros de segmentación y varias subclases específicas incluyendo umbralizado, Canny y métodos basados en la laplaciana.
- Wrapping (encapsulamiento) → Insight Toolkit utiliza un sistema único y potente para producir interfaces (por ejemplo, wrappers) para interpretar lenguajes como Tcl y Python. La herramienta GCC XML es utilizada para producir una descripción XML de código complejo en C++; CSWIG es entonces usado para transformar la descripción XML en wrappers usando el paquete SWIG.
- Utilidades auxiliares → varios subsistemas auxiliares están disponibles para complementar otras clases en el sistema. Por ejemplo, las calculadoras son clases que desempeñan operaciones especializadas en apoyo de los filtros (por ejemplo, MeanCalculator calcula la media de una muestra). Otras utilidades incluyen un analizador parcial DICOM, soporte para archivos MetaIO, visualización de imágenes para los formatos png, zlib y FLTK/Qt e interfaces para el sistema VTK.

Conceptos esenciales del sistema

A continuación se detallarán algunos de los principales conceptos y características de implementación que podemos encontrar en Insight Toolkit:

Programación genérica

La idea de esta forma de programar pretende generalizar las funciones utilizadas para que puedan usarse en más de una ocasión. Es un método de organización de librerías basadas en componentes genéricos o reusables.

Las principales ideas de la programación genérica son los contenedores que se utilizan para almacenar datos, los iteradores para acceder a estos datos y los algoritmos genéricos que usan

contenedores e iteradores para crear de manera eficiente algoritmos fundamentales tales como la clasificación.

La programación genérica está implementada en C++ con template (template programming) y con el uso de la librería STL (Standard Template Library).

Insight Toolkit usa programación genérica en su implementación. La ventaja de este enfoque es que una variedad casi ilimitada de tipos de datos están soportados simplemente por la definición de los tipos template apropiados. Por ejemplo, en Insight Toolkit es posible crear imágenes en la que sus pixeles sean casi cualquier tipo. Además, la resolución del tipo es realizada en tiempo de compilación, por lo que el compilador puede optimizar el código para obtener el máximo rendimiento.

La desventaja de la programación genérica es que muchos compiladores aún no soportan este enfoque y no pueden compilar Insight Toolkit.

Programación template

Los templates son una característica del lenguaje de programación C++ que permite a funciones y clases operar con tipos genéricos. Esto permite a una función o clase trabajar con diferentes tipos de datos sin tener que reescribir cada tipo para cada clase o función. Los templates son una gran utilidad para los programadores de C++, sobre todo cuando se combina con la herencia múltiple y la sobrecarga de operadores.

La técnica de programación template permite a los programadores escribir código en términos de uno o más tipos desconocidos T. Para crear código ejecutable, el programador debe especificar todos los tipos T y así procesará el código con el compilador. T puede ser un tipo nativo como un float o un int, o puede ser un tipo definido por el programador (por ejemplo, una clase). En tiempo de compilación, el compilador se debe asegurar que los tipos template son compatibles con el código instanciado y que estos están soportados por los métodos y operadores necesarios.

Por último, veamos ahora una serie de ventajas y desventajas de la programación mediante el uso de template.

Los template son considerados un tipo seguro, esto es porque requieren un chequeo de tipos en tiempo de compilación. De ahí, que el compilador pueda determinar en tiempo de compilación si el tipo asociado a la definición del template puede realizar todas las funciones requeridas por dicha definición.

Por diseño, los template pueden ser utilizados en problemas muy complejos, mientras que las macros están más limitadas.

Hay algunos principales inconvenientes a la hora de usar templates:

- Muchos compiladores tienen un soporte muy pobre de los templates. Por lo que el uso de los mismos decrementa la portabilidad del código.

- Muchos compiladores carecen de instrucciones de limpieza cuando detectan un error de definición de un template. Esto ha aumentado el esfuerzo en el desarrollo de templates, y ha impulsado la inclusión de nuevos conceptos en el siguiente estándar de C++.

- Puesto que el compilador genera código adicional para cada tipo template, el uso indiscriminado de los mismos puede llevar a sobrecargar el código, generando grandes ejecutables.

- El uso de los símbolos < y > como delimitadores es un problema para las herramientas que analizan el código fuente sintácticamente. Esto dificulta, o hace casi imposible para estas herramientas determinar si el uso de estos símbolos es como operadores de comparación o como delimitadores template.

Inclusión de archivos y clases

Las clases en Insight Toolkit están definidas como máximo por dos archivos: el fichero cabecera ".h" y el fichero de implementación, que podrá ser ".cxx" sino es una clase template y ".txx" si es una clase template. El fichero cabecera contiene las declaraciones de las clases y comentarios que serán usados por el sistema de documentación Doxygen para automáticamente producir manuales en HTML.

Fábricas de objetos

La mayoría de las clases en Insight Toolkit están instanciadas a través de un mecanismo de fábrica de objetos. Esto es, en vez de usar el constructor y destructor estándar de C++, las instancias de las clases en Insight Toolkit son creadas con un método estático llamado New().

La fábrica de objetos permite al programador controlar en tiempo de ejecución la instanciación de las clases registrando una o más fábricas con itk::ObjectFactoryBase. Estas fábricas registradas se apoyan en el método "CreateInstance(classname)" el cual toma como entrada el nombre de la clase a crear. La fábrica puede elegir crear la clase basándose en un número de factores, incluyendo la configuración del sistema y las variables de entorno.

Por ejemplo, en una aplicación particular, un usuario de Insight Toolkit puede querer utilizar su propia clase implementada usando hardware especializado en procesamiento de imágenes. Utilizando el mecanismo de fábrica de objetos, es posible en tiempo de ejecución reemplazar la creación de un filtro en particular de Insight Toolkit por una clase personalizada. Para hacer esto, el usuario compila su clase e inserta el código objeto en una librería compartida o en un DLL. La librería será entonces puesta en un directorio referenciado por la variable de entorno

ITK_AUTOLOAD_PATH. En la instanciación, la fábrica de objetos localizará la librería, determinará que puede crear la clase de un nombre particular con la fábrica y usará la fábrica para crear la instancia.

En la práctica, las fábricas de objetos serán usadas principalmente por clases de entrada/salida de Insight Toolkit. Para la gran mayoría de los usuarios, el mayor impacto es usar el método New() para crear una clase.

Punteros elegantes y manejo de memoria

Por naturaleza, los sistemas orientados a objetos representan y operan con datos a través de una gran variedad de tipos objetos o clases. Cuando una clase en particular es instanciada para producir una instancia de esa clase, se produce una reserva de memoria, de manera que la instancia pueda guardar los valores de los atributos y los punteros de los métodos. Este objeto puede ser referenciado por otras clases o por estructuras de datos durante operaciones normales del programa. Normalmente, mientras el programa está en ejecución todas las referencias a la instancia pueden desaparecer, es en este punto cuando la instancia debe ser borrada para recuperar recursos. Saber cuándo se debe borrar una instancia no es fácil. Borrar la instancia muy pronto provoca errores en el programa; borrarla muy tarde provoca consumo de memoria innecesario. Este proceso de reservar y liberación de memoria se conoce como manejo de la memoria.

En Insight Toolkit, el manejo de memoria es implementando a través de un contador de referencias. Esto se puede comparar con otros enfoques (garbage collection) usados por muchos sistemas en los que se incluye Java. En el contador de referencias, una cuenta del número de referencias a cada instancia es guardada. Cuando una referencia llega a cero, el objeto se libera. En el recolector de basura (garbage collection), un proceso en segundo plano barre el sistema identificando instancias que ya no están siendo referenciadas y las elimina. El problema con el recolector de basura es que el momento en el cual se libera la memoria es variable. Esto no se puede permitir cuando el tamaño de un objeto es muy grande. El contador de referencias, en cambio, libera la memoria inmediatamente, una vez que las referencias al objeto hayan desaparecido.

El contador de referencias es implementado a través de las funciones miembro “Register()” y “Delete()”. Todas las instancias de un objeto de Insight Toolkit tiene un método “Register()” que serán invocados por otro objeto que hace referencia a ellos. El método “Register()” incrementa el contador de referencias a instancias. Cuando la referencia a una instancia desaparece, el método “Delete()” es invocado en la instancia, el cual decrementará el contador de referencias. Cuando el contador de referencias devuelva cero, la instancia se destruirá.

Este protocolo se simplifica enormemente usando la clase auxiliar itk::SmartPointer. El puntero inteligente actúa como un puntero normal, pero automáticamente ejecuta “Register()” cuando hay una referencia a una instancia, y ejecuta “Unregister()” cuando no apunte más a la instancia. A diferencia de otras instancias de Insight Toolkit, los punteros inteligentes pueden ser alojados en

la pila del programa, y son automáticamente borrados cuando el ámbito para el que se creó es cerrado.

Tratamientos de errores y excepciones

En general, Insight Toolkit usa el tratamiento de excepciones para manejar errores durante la ejecución del programa. El tratamiento de excepciones es un parte básica del lenguaje C++.

Se puede elegir capturar un tipo de excepción en concreto, por lo tanto un error específico de Insight Toolkit. Pero también se pueden capturar cualquier excepción de Insight Toolkit mediante ExceptionObject.

Manejo de eventos

El manejo de eventos en Insight Toolkit es implementado usando el patrón de diseño vista/controlador. En este enfoque, los objetos indican que están esperando por un evento en particular (invocado por una instancia) registrando con la instancia que están esperando. Por ejemplo, los filtros en Insight Toolkit periódicamente invocan a itk::ProgressEvent. Los objetos que han registrado su interés en ese evento serán notificados cuando el evento ocurra. La notificación ocurre a través de la invocación de un comando (por ejemplo, un retorno de función, una llamada a un método, etc.) que es especificado durante el registro del proceso.

Multihilos

Los multihilos son tratados en Insight Toolkit a través de un alto nivel de abstracción. Este enfoque proporciona multihilos portables y esconde la complejidad de las diferentes implementaciones de hilos en los sistemas soportados por Insight Toolkit. Los multihilos son normalmente empleados por un algoritmo durante su fase de ejecución. Los multihilos pueden ser usados para ejecutar un único método en múltiples hilos o para especificar un método por hilo. Cuando un único método se ejecuta en múltiples hilos, los hilos tendrán cuidado de dividir la imagen (por ejemplo) en diferentes regiones de manera que no se solapen para las operaciones de escritura.

La filosofía de Insight Toolkit en lo que concierne a la seguridad de los hilos es que el acceso a diferentes instancias de una clase (y sus métodos) es una operación de hilos segura. Invocar métodos en la misma instancia en diferentes hilos está prohibido.

Visión general del sistema

Numérico

Insight Toolkit usa la librería numérica VNL para proporcionar recursos para la programación numérica combinando la facilidad de uso de paquetes como Mathematica y Matlab con la rapidez de C y la elegancia de C++.

La librería numérica VNL incluye clases para:

- Matrices y vectores.
- Matrices especiales y clases de vectores.
- Descomposición de matrices.
- Polinomios reales.
- Optimización.
- Funciones estándar y constantes.

Insight Toolkit también proporciona funcionalidades numéricas adicionales, como funciones estadísticas.

Representación de datos

Hay dos tipos principales para representar datos en Insight Toolkit: imágenes y mallas. Estas funcionalidades se implementan en las clases Image y Mesh, siendo ambas subclases de itk::DataObject.

Itk::Image representa un muestro de datos n-dimensional y regular. La dirección del muestro es paralela a cada eje de coordenadas. El origen del muestreo, el espacio entre píxeles y el número de muestras en cada dirección (por ejemplo, la dimensión) puede ser especificado. El tipo del muestreo o del pixel en Insight Toolkit puede ser aleatorio (un parámetro template TPixel especifica el tipo una vez instanciado el template). La dimensión de la imagen deberá ser especificada cuando la clase imagen es instanciada. La clave es que el tipo del pixel debe soportar ciertas operaciones (por ejemplo, suma o resta) si el código es compilado en todos los casos (por ejemplo, para ser procesado por un filtro en particular que usa estas operaciones). En la práctica los usuarios de Insight Toolkit usan tipos simples de C++ (por ejemplo, int o float) o tipos de pixel predefinidos y raramente crearán un nuevo tipo de pixel.

Uno de los conceptos más importantes en Insight Toolkit en lo que a las imágenes se refiere es que los trozos rectangulares y continuos de una imagen se conocen como regiones. Las regiones son usadas para especificar que parte de la imagen se va a procesar, por ejemplo en multihilos, o que parte se guarda en memoria. En Insight Toolkit hay tres tipos básicos de regiones:

1. LargestPossibleRegion: la imagen en su totalidad.
2. BufferedRegion: la parte de la imagen guardada en memoria.
3. RequestedRegion: la parte de la región solicitada por un filtro o por otra clase cuando se opera en la imagen.

Procesamiento de datos en el pipeline

Mientras los objetos de datos (por ejemplo, imágenes y mallas) son usados para representar datos, los objetos de procesos son clases que operan en los objetos de datos y pueden producir nuevos objetos de datos. Los objetos de procesos son clases como sources, objetos de filtros o mappers. Sources (tal que los lectores) producen datos, los objetos de filtros toman datos y los procesan para producir nuevos datos y los mappers aceptan datos desde el exterior o de un archivo o de otro sistema. A veces el término filtro es usado en líneas generales para referirse a los tres tipos.

El procesamiento de datos en el pipeline vincula los objetos de datos con los objetos de procesos. El pipeline suporta un mecanismo automático de actualización que causa que se ejecute un filtro si y sólo si su entrada o su estado interno ha cambiado. Además, el pipeline de datos soporta streaming, la habilidad de automáticamente dividir los datos en partes más pequeñas, procesar cada parte y juntar los datos procesados en un resultado final.

Normalmente los objetos de datos y los objetos de procesos están conectados entre sí usando los métodos “SetInput()” y “GetOutput()”.

Cuando el método “Update()” es invocado en el escritor, el procesamiento de datos del pipeline provoca en cada uno de los filtros la finalización, escribiendo los datos finales en un archivo en disco.

Clases principales a utilizar

En este apartado se comentarán aquellas clases más relevantes a la hora de desarrollar este proyecto. A medida que se avanza en el mismo, podrán ir surgiendo nuevas clases de interés a utilizar.

Clase Image

La clase itk::Image sigue el espíritu de la programación genérica, donde los tipos están separados del comportamiento algorítmico de la clase. Insight Toolkit soporta imágenes formadas por cualquier tipo de pixel y de cualquier dimensión espacial.

A continuación se detalla cómo realizar una serie de operaciones sobre imágenes que nos pueden ser de ayuda:

Crear una imagen

Lo primero que debemos hacer para crear una clase imagen es incluir el fichero cabecera "itkImage.h". Después decidiremos de qué tipo serán los píxeles de nuestra imagen y qué dimensión tendrá. Con estos dos parámetros ya podemos instanciar la clase imagen. La imagen puede entonces ser creada invocando al operador "New()" del tipo de imagen correspondiente y asignando el resultado a un puntero inteligente.

En Insight Toolkit, las imágenes existen por la combinación de una o más regiones. Una región es un subconjunto de la imagen e indica la parte de la imagen que puede ser procesada por otras clases en el sistema.

En Insight Toolkit, la creación manual de una imagen requiere que la imagen sea instanciada como hemos explicado anteriormente y que las regiones que describen la imagen sean entonces asociadas con ella.

Una región se define por dos clases: itk::Index e itk::Size. El origen de una región dentro de la imagen con la que está asociado se define mediante "Index". La extensión, o el tamaño, de la región viene dado por "Size", el cual vendrá representado por un array dónde los componentes del mismo son enteros que indican la extensión en pixeles de la imagen a lo largo de todas las dimensiones. "Index" se representa por un vector n-dimensional donde cada componente es un entero (en coordenadas de la image) que indica el pixel inicial de la imagen. Cuando una imagen es creada manualmente, el usuario es responsable de la definición del tamaño de la misma y del índice en el cual la imagen comienza. Estos dos parámetros hacen posible el proceso de selección de regiones.

Teniendo definido el comienzo y el tamaño de la imagen, estos dos parámetros serán usados para crear un objeto de tipo "ImageRegion" el cual encapsula estos dos conceptos. La región será inicializada con el índice y el tamaño.

Por último, la región será pasada al objeto imagen con el fin de definir su extensión y origen, esto se realiza mediante el método "SetRegions()". Por último se llamará al método "Allocate()" para tomar memoria dónde almacenar los datos de los pixeles de la imagen.

En la práctica es raro reservar memoria e inicializar una imagen directamente. Las imágenes se leen normalmente de un recurso, como un archivo o de sistema de adquisición.

Leer una imagen de un archivo

El primer requisito para leer una imagen desde un archivo es incluir el fichero cabecera "itkImageFileReader.h". Después, el tipo de la imagen debe ser definido especificando el tipo utilizado para representar los pixeles. También se deberá indicar la dimensión de la imagen.

Usando el tipo de la imagen, es posible instanciar la clase encargada de leer la imagen. El tipo de la imagen es usado como parámetro template para definir como los datos serán representados una vez sean cargados en memoria. Este tipo no tiene porque corresponder con el tipo guardado en el archivo. De todos modos, se utiliza una conversión como la utilizada por C en la conversión entre tipos. Los lectores no aplican ninguna transformación a los datos de los pixeles a excepción de la conversión realizada entre el tipo del pixel del archivo y el tipo del pixel del lector. Una vez tengamos instanciado el tipo del lector podremos usarlo para crear un objeto lector (ImageFileReader). Un puntero inteligente será usado para recibir la referencia al nuevo lector creado. El método "New()" será invocado para crear una instancia del lector.

La información mínima que necesita un lector es la ruta del archivo que va a ser cargado en memoria. Esto se obtiene a través del método "SetFileName()".

Los objetos lectores son referidos como los objetos recursos del pipeline; ellos responden a las peticiones de actualización del pipeline e inicializan el flujo de datos del pipeline. El mecanismo de actualización del pipeline se asegura de que el lector sólo se ejecuta cuando una petición de datos es hecha al lector y este no esté leyendo ninguna información. El método "Update()" puede ser invocado explícita o implícitamente. Si se realiza explícitamente puede ser porque la salida del lector no está conectada al resto de los filtros. Normalmente la salida del lector está conectada a la entrada de un filtro y la actualización en el filtro produce una actualización en el lector.

Para acceder a la imagen leída basta con llamar al método "GetOutput()" del lector. Este método se puede ejecutar incluso antes de que se realice la actualización del lector, lo que pasará será que la referencia a la imagen será válida aunque la imagen estará vacía hasta que el lector actual se ejecute.

Cualquier intento de acceder a los datos de la imagen antes de que el lector se ejecute producirá una imagen sin datos en los pixeles.

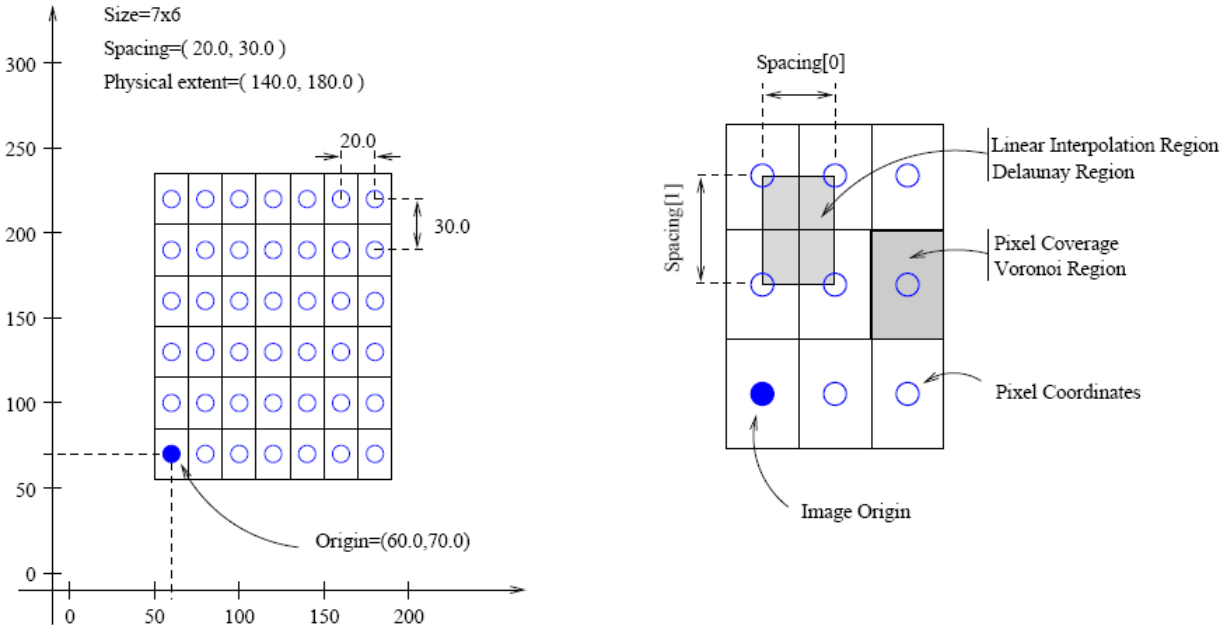

Figura 7.1: Origen y espacio [1]

Definiendo el origen y el espacio

Aunque Insight Toolkit se puede usar para trabajar con tareas de procesamiento de imágenes generales, el objetivo principal del sistema es el procesado de imágenes médicas. En este sentido, cierta información adicional acerca de las imágenes es necesaria. En particular la información asociada con el espacio físico entre los píxeles y la posición de la imagen en el espacio respecto a un sistema de coordenadas son muy importantes.

El origen y el espacio de una imagen son fundamentales para muchas aplicaciones. Las imágenes médicas sin información espacial no podrán ser usadas ni para diagnóstico médico, análisis de imágenes, extracción de características, terapias de radiación asistida o cirugía guiada por imágenes. En otras palabras, las imágenes médicas carentes de información espacial no sólo son inútiles sino que además son peligrosas.

Imágenes RGB

El espacio RGB ha sido construido como una representación de las respuestas fisiológicas a la luz por los tres tipos de conos del ojo humano. RGB no es un espacio de vectores. Por ejemplo, los números negativos no son apropiados en un espacio de colores porque serían el equivalente a una estimulación negativa en el ojo humano. En el contexto de la colorimetría, los valores de colores

negativos se usan como una construcción artificial para la comparación de colores

$$ColorA = ColorB - ColorC \tag{7.1}$$

Es sólo una forma de decir que podemos producir el Color B mediante la combinación del Color A y el Color C. Sin embargo, debemos ser conscientes de que (por lo menos en la emisión de luz) no es posible restar luz.

Cuando tratamos con color impreso y con pinturas, en contraposición con la luz emitida en las pantallas de los ordenadores, el comportamiento físico del color permite la resta. Esto es porque estamos hablando estrictamente de objetos que lo que vemos como rojo es aquello que ha absorbido todas las frecuencias de luces excepto aquellas que en su espectro posean el rojo.

El concepto de suma y resta en los colores debe ser interpretado cuidadosamente. De hecho, RGB tiene una definición diferente en lo que respecta a si estamos hablamos acerca de los canales asociados a los tres colores de los sensores del ojo humano, o a los tres fósforos encontrados en la mayoría de los monitos o a las tintas de color utilizadas en las impresoras. El espacio de colores no suele ser linear y a veces ni siquiera es de un grupo. Por ejemplo, no todos los colores que vemos pueden ser representados en el espacio RGB.

Insight Toolkit introduce el tipo itk::RBGPixel como soporte para la representación de los valores de un espacio de color RGB. Como tal, la clase "RGBPixel" expresa un concepto diferente de la de itk::Vector en el espacio. Por esta razón, "RGBPixel" carece de muchos de los operadores que cabrían esperar de él. En particular, no están definidas las operaciones de resta y suma. Cuando pensamos en la realización de la media en un tipo RGB estamos presuponiendo que en el espacio de colores proporcionado, la acción de encontrar un color en el centro de dos colores, puede ser calculada utilizando una operación linear entre sus representaciones numéricas. Este no es, por desgracia, el caso de los colores en el espacio debido a que están basados en las respuestas fisiológicas del ser humano.

Si se decide interpretar las imágenes RGB simplemente como tres canales independiente se tendrá que usar bastante el tipo itk::Vector como tipo para los píxeles. En este sentido, se tendrá acceso a todo el conjunto de operaciones definidas para los espacios de vectores. La implementación actual de "RGBPixel" en Insight Toolkit supone que las imágenes de color RGB están previstas para ser usadas en aplicaciones donde se requiera una interpretación formal del color, de ahí que sólo las operaciones que son válidas en los espacios de colores estarán disponibles para la clase "RBGPixel".

Gracias a la flexibilidad ofrecida por el estilo de programación genérica en el que Insight Toolkit está basado, es posible instanciar imágenes en las que sus píxeles sean de cualquier tipo.

Existe una clase en Insight Toolkit prevista para soportar píxeles del tipo RGB, por lo que si queremos tener una imagen con colores RGB podremos hacerlo. Para usar esta clase (itk::RGBPixel) debemos incluir el fichero cabecera "itkRGBPixel.h". El acceso a los componentes

de color de los píxeles puede ser realizado con los métodos proporcionados por la clase "RGBPixel".

Clase Iterator

La programación genérica parte de los contenedores y de los algoritmos. Los contenedores almacenan datos y los algoritmos operan con estos datos. Para acceder a los datos en los contenedores, los algoritmos usan tres tipos de objetos llamados iteradores. Un iterador es una es una abstracción de un puntero de memoria. Todos los tipos contenedor definen sus propios tipos iterador, pero todos los iteradores están escritos para proporcionar una interfaz común de tal manera que el algoritmo puede referenciar los datos de una forma genérica y mantener cierta independencia funcional con los contenedores.

El iterador se llama así porque se es usado para el acceso secuencial e iterativo a los valores de un contenedor. Los iteradores aparecen en las estructuras for y while. Un puntero en C es, por ejemplo, un tipo de iterador. Puede ser incrementado o decrementado a través de la memoria para referenciar secuencialmente los elementos de un vector. Muchas implementaciones de los iteradores tienen una interfaz similar a la de un puntero en C.

En Insight Toolkit se usan los iteradores para escribir código de procesamiento de imágenes genéricas. Con ello se permite instanciar imágenes formadas por combinaciones diferentes de tipos de píxeles, por tipos contenedor de píxeles y por dimensiones variadas. Debido a que los iteradores de imágenes están especialmente diseñados para trabajar con contenedores de imágenes, su interfaz e implementación están optimizadas para tareas de procesamiento de imágenes. El uso de un iterador de Insight Toolkit en vez del acceso directo a los datos a través de la interfaz itk::Image proporciona muchas ventajas. El código es mucho más compacto, los algoritmos son mucho más rápidos y los iteradores simplifican tareas como multihilos y procesamiento de imágenes basado en vecindades.

A continuación se detallarán algunas operaciones fundamentales para el uso de los iteradores:

Crear un iterador

Todos los iteradores de imágenes tienen al menos un parámetro template que es el tipo de la imagen sobre el cual van a iterar. No hay ninguna restricción sobre la dimensión de la imagen o sobre el tipo de píxeles que contiene la misma.

Un constructor de iteradores necesita al menos dos argumentos, un puntero inteligente a la imagen que tiene que recorrer y una región de una imagen. La región de la imagen, llamada región de iteración, es un área rectangular en la cual se producirá el recorrido. La región de iteración debe estar totalmente contenida dentro de la imagen. Más específicamente, una región de iteración válida es cualquier subregión de una imagen (BufferedRegion).

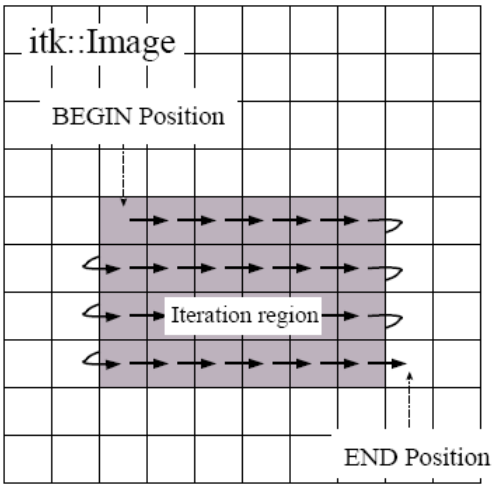

Figura 7.2: Recorrido de un iterador [1]

Hay versiones constantes y no constantes para la gran mayoría de los iteradores de imágenes en Insight Toolkit. Un iterador no constante no puede ser instanciado apuntando a una imagen que sea constante. La versión constante de un iterador puede leer pero no puede escribir los valores de los píxeles.

Mover un iterador

Un iterador se va moviendo por su región de iteración. En cada momento, el iterador apuntará, punto a punto, a la ubicación de un pixel en una imagen N-dimensional. La iteración hacia adelantes va desde el comienzo de la región de iteración hasta el final de la misma. La iteración hacia atrás va desde pasado el último píxel de la región hasta el comienzo ce la misma. Por tanto, hay dos puntos de comienzo para los iteradores, el principio y el final de una imagen. Un iterador puede ser movido directamente a una de estas dos posiciones utilizando los siguientes métodos:

- "GoToBegin()" → Fija el iterador en el primer elemento válido de la región.
- "GoToEnd()" → Fija el iterador en la siguiente posición al último elemento de la región.

Se debe recordar que la posición a la que apunta GoToEnd() no está dentro de la región de iteración. Este hecho se debe recordar ya que si se intenta liberar el iterador en este punto obtendremos resultados indefinidos.

Los iteradores de Insight Toolkit son movidos usando los operadores de incremento y decremento:

- "Operator++()" → incrementa el iterador una posición en dirección positiva.
- "Operator++()" → decrementa el iterador una posición en dirección negativa.

En la figura 7.2 se puede ver cómo actúa un iterador en una región de una imagen. Un iterador primero se mueve a través de las columnas, después por filas, después capa a capa y así sucesivamente. Además de la iteración secuencial a través de la imagen, algunos iteradores pueden definir operadores de acceso aleatorio. A diferencia de los operadores incrementales, los operadores de acceso aleatorio pueden no estar optimizados para ser rápidos y requieren algunos conocimientos de la dimensión de la imagen y de la extensión de la región de iteración para poder ser usados correctamente.

- "Operator+=(OffsetType)" → mueve el iterador a la ubicación del pixel que surja como resultado de sumar el índice actual al desplazamiento dado.
- "Operator-=(OffsetType)" → mueve el iterador a la ubicación del pixel que surja como resultado de restar el índice actual al desplazamiento dado.

El método "SetPosition()" puede ser extremadamente lento para los iteradores más complicados. En general, sólo se utiliza para fijar la posición de comienzo, tal y como hacen los métodos "GoToBegin()" y "GoToEnd()". Muchos iteradores no siguen la ruta que se espera que sigan a través de sus regiones de iteración y no tienen establecido la posición de comienzo o de fin. Los iteradores aleatorios, por ejemplo, incrementan y decrementan de manera aleatoria la posición del iterador. De esta forma pueden visitar una posición más de una vez. A un iterador se le puedo preguntar si está al comienzo o al final de una región de iteración:

- "bool IsAtEnd()" → devuelve verdadero si el iterador apunta a la siguiente posición del último pixel de la región de iteración.
- "bool IsAtbegin()" → devuelve verdadero si el iterador apunta a la primera posición de la región de iteración. Este método se utiliza para comprobar si estamos en el final de una iteración al revés.

Un iterador también puede informar de su posición actual en la imagen:

- "IndexType GetIndex()" → devuelve el índice de la posición actual del iterador.

Por eficiencia, la mayoría de los iteradores de imágenes en Insight Toolkit no realizan comprobación de límites. Por lo que es posible mover un iterador fuera de su región válida de iteración.

Clase Image Iterator

A continuación se describirán aquellos iteradores que recorren regiones de imágenes rectilíneas y apuntan a una posición en concreto en cada momento. itk::ImageRegionIterator es el iterador de imágenes más común en Insight Toolkit y suele ser la primera elección para la gran mayoría de las aplicaciones. El resto de los iteradores que se explicarán son especializaciones de "ImageRegionIterator". Estos se han diseñado para realizar tareas básicas de procesamiento de imágenes de manera más eficiente o para facilitar la implementación.

ImageRegionIterator

itk::ImageRegionIterator está optimizado para iterar de forma rápida y es la primera elección para operaciones iterativas en lo que respecta a píxeles cuando la posición en la imagen no es importante. "ImageRegionIterator" es el iterador menos especializado de las clases iterador de imágenes de Insight Toolkit.

ImageRegionIteratorWithIndex

La familia de iteradores "WithIndex" están diseñados para algoritmos que usan tanto el valor como la posición del pixel para sus cálculos. A diferencia de itk::ImageRegionIterator, el cual calcula el índice sólo si se requiere, itk::ImageRegionIteratorWithIndex guarda la posición del índice como variable miembro la cual será actualizada mientras se incrementa o decrementa la iteración. La rapidez con la que se realiza la iteración es más lenta pero las peticiones de la posición del índice son más eficientes.

ImageLinearIteratorWithIndex

itk::ImageLinearIteratorWithIndex está diseñado para ir procesando línea por línea una imagen. Este iterador recorre de forma linear la imagen en dirección paralela a uno de los ejes de coordenadas. Conceptualmente, este iterador divide la imagen en una serie de líneas paralelas que se extienden a lo largo del tamaño de la imagen.

Como todos los iteradores de imágenes, el movimiento de "ImageLinearIteratorWithIndex" está limitado dentro de la región de la imagen R. La línea "l" a través de la cual se mueve el iterador está definida seleccionando una dirección y un origen. La línea l se extiende desde el origen hasta el límite superior de R. El origen puede ser movido a cualquier posición a lo largo de los límites inferiores de R.

Existen varios métodos adicionales para este iterador que controlan el movimiento del mismo

a lo largo de la línea l y el desplazamiento del origen de l:

- "NextLine()" → mueve el iterador a la primera posición de la siguiente línea de la imagen.
- "PreviousLine()" → mueve el iterador a la última posición válida de la línea anterior.
- "GoToBeginOfLine()" → mueve el iterador a la primera posición de la línea actual.
- "GoToEndOfLine()" → mueve el iterador una posición más allá de la última posición válida de la línea actual.
- "IsAtReverseEndOfLine()" → devuelve verdadero si el iterador apunta a una posición anterior a la primera posición de la línea actual.
- "IsAtEndOfLine()" → devuelve verdadero si el iterador apunta a una posición más allá de la última posición válida de la línea actual.

ImageSliceIteratorWithIndex

La clase itk::ImageSliceIteratorWithIndex es una extensión de itk::ImageLinearIteratorWithIndex, pasando de iterar a lo de largo de las líneas a iterar tanto a lo largo de las líneas como de los planos de la imagen. Una capa es un plano 2D abarcado por dos vectores apuntando a lo largo del eje de coordenadas ortogonal. La orientación de la capa está definida por la especificación de sus dos ejes de expansión.

- "SetFirstDirection()" → especifica la primera dirección del eje de coordenadas de la capa.
- "SetSecondDirection()" → especifica la segunda dirección del eje de coordenadas de la capa.

Varios métodos controlan el movimiento de una capa a otra:

- "NextSlice()" → mueve el iterador a la primera posición de la siguiente capa en la imagen.
- "PreviousSlice()" → mueve el iterador a la última posición válida en la capa anterior.
- "IsAtReverseEndOfSlice()" → devuelve verdadero si el iterador apunta a una posición anterior a la primera posición de la capa actual.
- "IsAtEndOfSlice()" → devuelve verdadero si el iterador apunta a una posición más allá de la última posición válida de la capa actual.

El iterador de capas se mueve línea a línea utilizando NextLine() y PreviousLine(). La dirección de la línea es paralela a la dirección del segundo eje de coordenadas del plano de la capa.

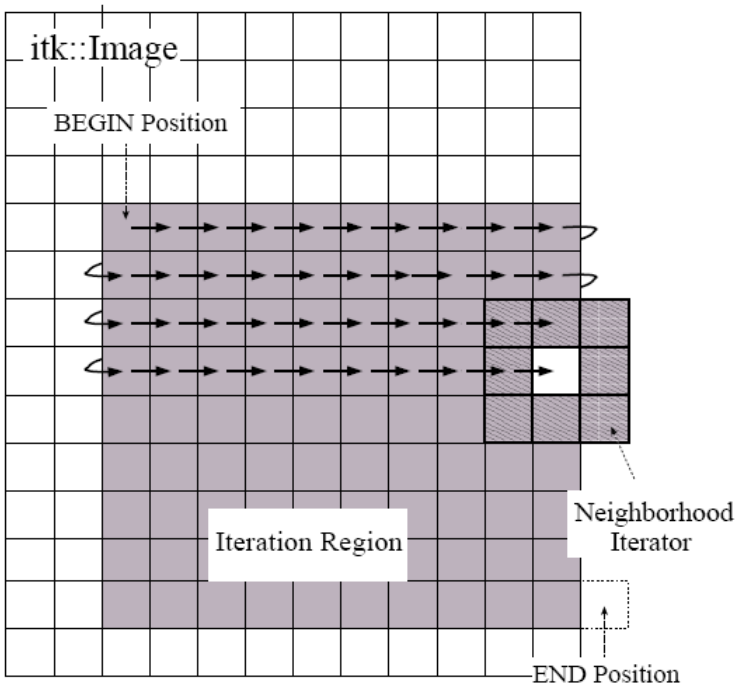

Figura 7.3: Neighborhood Iterator [1]

ImageRandomConstIteratorWithIndex

itk::ImageRandomConstIteratorWithIndex fue desarrollado para elegir aleatoriamente valores de una muestra. Cuando incrementamos o decrementamos el iterador este salta a una posición aleatoria en la región de la imagen.

El usuario debe especificar el tamaño de la muestra cuando se crea el iterador. "IsAtEnd()" devuelve verdadero cuando el número actual de la muestra es igual al tamaño de la muestra. "IsAtBegin()" devuelve verdadero cuando el número actual de la muestra es igual a cero. Una diferencia importante es que a diferencia de los otros iteradores, este iterador puede visitar la posición más de una vez.

Este iterador puede ser usado para calcular datos estadísticos de una imagen, como por ejemplo la media.

Clase Neighborhood Iterators

En Insight Toolkit la vecindad de un pixel está definida como un conjunto pequeño de píxeles localmente adyacentes unos a otros en una imagen. El tamaño y la forma de la vecindad, también la conectividad entre los pixeles de una vecindad, puede variar con la aplicación.

Muchos algoritmos de procesamiento de imágenes están basados en vecindades, es decir, el resultado de un pixel i se calcula a partir de los valores de los pixeles en la vecindad ND de i. Un ejemplo común en el que se usan vecindades son las operaciones que incluyen convoluciones.

Por tanto, en este apartado se describirán aquellos iteradores que están diseñados para trabajar con las vecindades de los pixeles. En Insight Toolkit los iteradores de vecindades recorren la región de la imagen como un iterador normal, pero en vez de referenciar sólo a un único pixel en cada paso, apunta simultáneamente a toda la vecindad ND. Como ampliación de los iteradores normales podemos tener acceso de lectura y escritura a todos los pixeles vecinos e información sobre el tamaño, extensión y posición de la vecindad.

En la figura 7.3 podemos ver un iterador de vecindades moviéndose a través de una región de iteración. Este iterador define una vecindad de 3x3 alrededor de cada pixel que visita. El centro de la vecindad está siempre posicionado en el índice actual y el resto de los índices de los pixeles serán referenciados mediante el desplazamiento desde el centro.

Además de un puntero a la imagen y a la región de iteración, el constructor del iterador de vecindades necesita un argumento que especifique la extensión de la vecindad a cubrir. La extensión de la vecindad es simétrica y viene dada por un vector de N distancias que se denomina radius.

Neighborhood extent is symmetric across its center in each axis and is given as an array of N distances that are collectively called the radius. Each element d of the radius, where $0 < d < N$ and N is the dimensionality of the neighborhood, gives the extent of the neighborhood in pixels for dimension N. The length of each face of the resulting ND hypercube is 2d +1 pixels, a distance of d on either side of the single pixel at the neighbor center.

El radio de un iterador de vecindades puede ser consultado. Otros métodos proporcionan información útil acerca del iterador y de su imagen:

- "SizeType GetRadius()" → devuelve el radio ND de una vecindad como un itk::Size.
- "Const ImageType *GetImagePointer()" → devuelve un puntero a la imagen referenciada por el iterador.
- "Unsigned long Size()" → devuelve el tamaño en número de pixeles de la vecindad.
-

El iterador de vecindades extiende el concepto de obtener y fijar los valores de un pixel con respecto al iterador normal. La vecindad de un pixel se puede ver como un vector linear donde cada pixel tiene un único índice entero. El pixel central está siempre en la posición n/2, donde n es el tamaño del vector.

- "PixelType GetPixel(const unsigned int i)" → devuelve el valor del pixel que se encuentra en la posición i de la vecindad.
- "Void SetPixel(const unsigned int i, PixelType p)" → cambia el valor del pixel en la posición i por el valor p.

Otra forma de pensar en la posición de un pixel en una vecindad es como desplazamientos ND desde el centro de la vecindad. La esquina superior izquierda de una vecindad 3x3x3, por ejemplo, puede ser descrita mediante el desplazamiento (-1,-1,-1). La esquina inferior derecha de la misma vecindad tendrá el desplazamiento (1,1,1).

- "PixelType GetPixel(const OffsetType &o)" → toma el valor del pixel que se encuentra en la posición dada por el desplazamiento o medido desde el centro de la vecindad.
- "void SetPixel(const OffsetType &o, PixelType p)" → cambia el valor que se encuentra en la posición dada por el desplazamiento o medido desde el centro de la vecindad por el valor p.

Los iteradores de vecindades también proporcionan formas de obtener y fijar el valor del pixel central de una vecindad.

- "PixelType GetCenterPixel()" → toma el valor del pixel central de la vecindad.
- "void SetCenterPixel(PixelType p)" → cambia el valor del pixel central de la vecindad por el valor p.

Hay otras formas de obtener y fijar valores de pixeles que están a cierta distancia del centro de la vecindad.

- "PixelType GetNext(unsigned int d)" → toma el valor adyacente al pixel central de la vecindad en la dirección positiva del eje d.
- "void SetNext(unsigned int d, PixelType p)" → cambia el valor adyacente al pixel central de la vecindad en la dirección positiva del eje d por el valor p.
- "PixelType GetPrevios(unsigned int d)" → toma el valor adyacente al pixel central de la vecindad en la dirección negativa del eje d.
- "void SetPrevious(unsigned int d, PixelType p)" → cambia el valor adyacente la pixel central de la vecindad en la dirección negativa del eje d por el valor p.
- "PixelType GetNext(unsigned int d, unsigned int s)" → toma el valor del pixel que se encuentra a s pixeles del pixel central de la vecindad en la dirección positiva del eje d.

- "void SetNext(unsigned int d, unsigned s, PixelType p)" → cambia el valor del pixel que se encuentra a s pixeles del pixel central de la vecindad en la dirección positiva del eje d por el valor p.
- "PixelType GetPrevious(unsigned int d, unsigned int s)" → toma el valor del pixel que se encuentra a s pixeles del pixel central de la vecindad en la dirección negativa del eje d.
- "void SetPrevious(unsigned int d,unsigned int s, PixelType p)" → cambia el valor del pixel que se encuentra a s pixeles del pixel central de la vecindad en la dirección negativa del eje d por el valor p.

Es posible extraer un conjunto de valores de una vecindad en una sola vez. Esto puede ser útil en algoritmos que realizan un gran número de operaciones en una vecindad y que requieren un gran número de referencias a los mismos pixeles.

- "NeighborhoodType GetNeighborhood()" → devuelve un itk::Neighbordhood del mismo tamaño y forma que el iterador de vecindades y contiene todos los valores del iterador en esa posición.
- "void SetNeighborhood(NeighborhoodType &N)" → cambia todos los valores de una vecindad a los valores contenidos en la vecindad N, la cual debe ser del mismo tamaño y forma.

Varios métodos están definidos para proporcionar información acerca de la vecindad.

- "IndexType GetIndex()" → devuelve el índice del pixel central de la vecindad.
- "IndexType GetIndex(OffsetType o)" → devuelve el índice del pixel que se encuentra en la posición indicada por el desplazamiento o medida desde el centro de la vecindad.
- "IndexType GetIndex(unsigned int i)" → devuelve el índice del pixel que se encuentra en la posición i del vector.
- "OffsetType GetOffset(unsigned int i)" → devuelve el desplazamiento desde el centro de la vecindad que se encuentra en la posición i del vector.
- "unsigned long GetNeighborhoodIndex(OffsetType o)" → devuelve la posición del vector con desplazamiento o desde el centro de la vecindad.
- "std::slice GetSlice(unsigned int n)" → devuelve un std::slice a través del iterador de vecindades a lo largo del eje n.

Un cálculo basado en vecindades en una vecindad cercana a los límites de la imagen puede requerir que los datos estén por fuera de los límites. Cuando la extensión de una vecindad cae

fuera de los límites de una imagen, los valores para los vecinos perdidos son facilitados según una regla, normalmente elegidos para satisfacer las necesidades numéricas del algoritmo. Una regla para facilitar los valores fuera de los límites de llama condición límite.

Los iteradores de vecindades en Insight Toolkit detectan las referencias fuera de los límites devolviendo valores acordes a las condiciones límites. El tipo de la condición límite viene especificada el segundo parámetro del iterador (opcional). Por defecto, los iteradores de vecindades usan la condición de Neumann donde la primera derivada a través del límite es cero. La regla de Neumann devuelve los valores de los pixeles más cercanos a aquellos que se han solicitado y que están fuera de los límites. Existen varios métodos de condiciones límite en Insight Toolkit. Uno de estos es la condición periódica que devuelve el valor del pixel que se encuentra en el lado contrario, o la condición de valores constantes que devuelve un conjunto de valores para todos los pixeles que están fuera de los límites. Esto es como si rellenásemos la imagen con los valores.

El chequeo de límites es una operación con un alto coste computacional porque se realiza cada vez que el iterador es incrementado. Para aumentar la eficiencia, los iteradores de vecindades automáticamente deshabilitan esta opción cuando detectan que no es necesario. El usuario debería también especificar si desea tener el chequeo de límites habilitado o no.

- "void NeedToUseBoundaryConditionOn()" → se activa el chequeo de límites.
- "void NeedToUseBoundaryConditionOff()" → se desactiva el chequeo de límites.
- "void NeedToUseBoundaryConditionOff() → se desactiva el chequeo de límites.
- "void OverrideBoundaryCondition(BoundaryConditionType *b)" → cambia la condición límite por la condición límite b. Este método se puede utilizar para cambiar en tiempo de ejecución el comportamiento del iterador.
- "void ResetBoundaryCondition()" → se deja de utilizar cualquier condición de límite especificada (en tiempo de ejecución) y se vuelve a utilizar la condición de base.
- "void SetPixel(unsigned int i, PixelType p, bool status)" → se cambia el valor de una posición i del vector de vecindades por el valor p. Si la posición i está fuera de los límites, status contendrá false, en caso contrario contendrá true.

DART

Una de las características únicas de Insight Toolkit es el uso de DART (sistemas de pruebas de regresión). En pocas palabras, lo que hace DART es proporcionar información cuantificable a los desarrolladores que comprueban código nuevo y hacen cambios.

La información se basa en los resultados de una variedad de tests y los resultados son publicados en una página web, la cual se conoce como "dashboard". Dado que todos los usuarios y

desarrolladores de Insight Toolkit pueden ver la página web, DART sirve como de vehículo para la comunicación de desarrolladores, especialmente cuando se hacen aportaciones con fallos. Es aconsejable consultar el "dashboard.ªntes de realizar cualquier actualización del software en el repositorio.

Otra característica es que DART es independiente de Insight Toolkit y puede ser utilizado para tener un control de calidad sobre cualquier tipo de proyecto software.

DART soporta una gran variedad de tipos de test. Entre ellos están los siguientes:

- Compilación: todo el código fuente y de prueba es compilado y enlazado. Cualquier error o warning será publicado en el "dashboard".
- Regresión: algunos test en Insight Toolkit producen imágenes de salida. Cada test necesita comparar la imagen de salida con una imagen de referencia válida. Si las imágenes coinciden entonces el test se da por válido. La comparación debe ser realizada con cuidado dado que muchos sistemas que usan gráficos 3D producen resultados ligeramente diferentes en distintas plataformas.
- Memoria: los problemas relacionados con la memoria como fallos, accesos a memoria no inicializada, lecturas/escrituras más allá del espacio asignado pueden provocar resultados inesperados y hacer que el programa falle. Insight Toolkit chequea en tiempo de ejecución los accesos y el manejo de memoria utilizando "Purify".
- "Printself": se espera que todas las clases en Insight Toolkit impriman por pantalla todas sus variables (aquellas asociadas con los métodos "Get y Set") correctamente. Este test comprueba que realmente esto se realiza.
- Módulo: cada clase en Insight Toolkit debería tener su correspondiente módulo de testeo donde las funcionalidades son probadas y comparadas cuantitativamente con los resultados esperados. Estos test normalmente son escritos por el desarrollador de la clase y este debe esforzarse para cubrir todas las líneas de código incluyendo los métodos "Get/Set" y el manejo de errores.
- Cubrir: aquel código que no se ejecuta durante los test es probable que sea erróneo. Los test de cobertura identifican aquellas líneas que no se están ejecutando en los test de Insight Toolkit, informando del porcentaje total cubierto al final del mismo. Si bien es casi imposible cubrir el 100 %, del código debido al código de manejo de errores y construcciones similares que raramente se encuentran en la práctica, se busca cubrir al menos el 75 % o más. Aquel código que no está bien cubierto necesitará pasar pruebas adicionales.

Cuando un usuario o un desarrollador decide actualizar el código de Insight Toolkit desde el repositorio es importante verificar primero que el "dashboard" está correcto. Esto se puede comprobar rápidamente por la coloración general del "dashboard". Cuando está es verde significa

que el software está compilado correctamente y que es un buen momento para empezar a utilizar Insight Toolkit o para obtener una actualización. Cuando está en rojo indica inestabilidad en el sistema y, por tanto, los usuarios deben abstenerse de revisar o actualizar el código fuente.

Otra característica de DART es que mantiene un historial de los cambios realizados en el código fuente y resumen los cambios como parte del "dashboard". Esto es útil para llevar un seguimiento de los problemas y mantenerse al día de las actualizaciones de Insight Toolkit.

Insight Toolkit procesa funciones a través de tres ciclos (el ciclo continuo, el ciclo diario y el ciclo de versión).

El ciclo continuo gira en torno a las acciones de los desarrolladores como comprobación de código en el repositorio. Cuando se cambia o se comprueba código nuevo en el repositorio, el proceso de testeo continuo de DART se inicia. Un pequeño número de test son realizados, y si algo falla, se envía un correo electrónico a todos los desarrolladores que han comprobado el código durante el ciclo continuo. Se espera que los desarrolladores arreglen el problema inmediatamente.

El ciclo diario se produce en un periodo de 24 horas. Los cambios que se realizan en el código fuente durante el día son ampliamente probados por la noche por la secuencia de pruebas de regresión de DART. Estas pruebas ocurren en diferentes máquinas combinadas y en diferentes sistemas operativos alrededor del mundo, y los resultados son publicados diariamente en el "dashboard" de DART. Los desarrolladores que comprueban el código deben visitar el "dashboard" y asegurarse que sus cambios son aceptables (no introducen errores de compilación o warnings, o fallos en otros test incluyendo la regresión, memoria, "printself" y los métodos "Set y Get").

El ciclo de versión ocurre un pequeño número de veces al año. Esto requiere etiquetar el repositorio, actualizar la documentación y producir nuevos paquetes para la versión. Además, se realizan pruebas adicionales para asegurar la consistencia del paquete, manteniendo el repositorio libre de errores minimizando de esta forma el trabajo requerido para quitar una versión. Los usuarios de Insight Toolkit trabajan normalmente con las publicaciones, dado que son más estables. Los desarrolladores trabajan con el repositorio y a veces, con las publicaciones recientes a fin de ver las ventajas de las nuevas características añadidas. Es muy importante que los desarrolladores presten especial atención al "dashboard" y actualicen su software únicamente cuando este esté en verde. Si no se hace así puede causar importantes perturbaciones si un día en concreto la versión del software es inestable.

La eficacia de este proceso es bastante buena, al proporcionar información inmediata a los desarrolladores a través del correo electrónico y páginas web, la calidad de Insight Toolkit es excepcionalmente alta, sobre todo considerando la complejidad de los algoritmos y del sistema. Los errores, cuando se introducen accidentalmente, son capturados rápidamente, en comparación a cuando son capturados en el momento de la publicación de una versión. Esperar hasta este punto es esperar demasiado, dado que la relación causal entre el código cambiado o añadido y el error se ha perdido.

7.1.2. CMake

CMake es un sistema multiplataforma para la automatización de la compilación. Su nombre es una abreviatura de "cross platform make", a pesar del uso de "make" en su nombre, CMake, es una aplicación separada y de más alto nivel que normalmente se utiliza para el desarrollo en Unix. CMake es una familia de herramientas diseñadas para compilar, probar y empaquetar software. CMake se utiliza para controlar el proceso de compilación del software utilizando una plataforma simple y un compilador de ficheros de configuración independientes. CMake genera "makefiles" nativos y espacios de trabajo que pueden ser usados en el entorno de compilación que se elija. Se puede comparar al sistema de compilación de UNIX GNU en que el proceso de compilación está controlado por los archivos de configuración, en el caso de CMake llamados ficheros "CMakeLists.txt". Como las herramientas del sistema de compilación de GNU, CMake no genera directamente el software final, en cambio, compila ficheros estándar (por ejemplo, "makefiles" en UNIX y proyectos en Windows Visucal C++ o Eclipse) que serán utilizados de manera estándar, permitiendo un enfoque más fácil para los desarrolladores familiarizados con un entorno de desarrollo particular. Sin embargo, a diferencia del sistema de compilación de GNU, el cuál está restringido sólo a plataformas UNIX, CMake soporta la generación de ficheros de compilación para muchos sistemas de compilación en muchos sistemas operativos. CMake, por tanto, soporta desarrollo multiplataforma eliminando la necesidad de mantener sistemas de compilación separados para cada plataforma.

CMake es un sistema extensible, de código abierto, que maneja el proceso de compilación en un sistema operativo y de manera independiente por el compilador. A diferencia de muchos sistemas multiplataformas, CMake está diseñado para ser utilizado en conjunto con entornos nativos de compilación. Los ficheros de configuración (CMakeLists.txt) que están colocados en cada directorio fuente son utilizados para generar archivos de compilación estándares que serán utilizados de forma habitual. CMake es capaz de compilar código fuente, crear librerías, generar contenedores y compilar ejecutables en combinaciones arbitrarias. CMake soporta compilaciones dentro y fuera de sitio (in-place y out-of-place builds)y por lo tanto puede soportar múltiples compilaciones a partir de un único árbol fuente. CMake también soporta la compilación de librerias estáticas y dinámicas. Una característica distintiva de CMake es que genera un fichero cache que está diseñado para ser utilizado con un editor gráfico. Por ejemplo, cuando se ejecuta CMake sitúa ficheros include, librerías y ejecutables pudiendo encontrar directivas opcionales de compilación. Esta información se almacena en la cache, la cuál podrá ser cambiada por el usuario antes de la generación de los ficheros nativos de compilación.

CMake está diseñado para soportar complejas jerarquías de directorios y aplicaciones que dependen de muchas librerías. Por ejemplo, CMake soporta proyectos basados en múltiples conjuntos de herramientas (por ejemplo, librerías), donde cada conjunto de herramientas puede contener varios directorios, y la aplicación depende de los conjuntos de herramientas más codigo adicional. CMake también puede manejar situaciones en las que los ejecutables deben ser compilados para generar código que después será compilado y enlazado en una aplicación final.

El proceso de compilación está controlado por la creación de uno o más ficheros "CMake-

Lists.txt" en cada directorio (incluyendo subdirectorios) que construyen el proyecto. Cada "CMakeLists.txt" está formado por uno o más comandos. Cada comando tiene la forma **COMMAND(args...)** dónde **COMMAND** es el nombre del comando, y **args** es una lista de argumentos separados por espacios en blanco. CMake proporciona comandos predefinidos y comandos definidos por el usuario.

7.1.3. AMILab

AMILab es un lenguaje interpretado para el procesamiento y tratamiento de imágenes. Para su diseño se ha utilizado:

- C/C++
- Flex/Bison
- wxWidgets
- OpenGL
- VTK
- ITK

AMILab tiene muchas características, y ha sido especialmente utilizado para la investigación académica en el procesamiento de imágenes médicas. A día de hoy la herramienta se encuentra en fase de desarrollo y sufre cambios continuamente.

A continuación se detallan algunas de las herramientas utilizadas por AMILab.

Lex

Lex es un programa para generar analizadores léxicos (en inglés scanners o lexers). Lex se utiliza comúnmente con el programa yacc que se utiliza para generar análisis sintáctico. Lex, escrito originalmente por Eric Schmidt y Mike Lesk, es el analizador léxico estándar en los sistemas Unix, y se incluye en el estándar de POSIX. Lex toma como entrada una especificación de analizador léxico y devuelve como salida el código fuente implementando el analizador léxico en C. Aunque tradicionalmente se trata de software propietario, existen versiones libres de lex basadas en el código original de AT&T en sistemas como OpenSolaris y Plan 9 de los laboratorios Bell. Otra versión popular de software libre de lex es Flex.

Bison

Yacc es un programa para generar analizadores sintácticos. Las siglas del nombre significan Yet Another Compiler-Compiler, es decir, "Otro generador de compiladores más". Genera un analizador sintáctico (la parte de un compilador que comprueba que la estructura del código fuente se ajusta a la especificación sintáctica del lenguaje) basado en una gramática analítica escrita en una notación similar a la BNF. Yacc genera el código para el analizador sintáctico en el Lenguaje de programación C.

Fue desarrollado por Stephen C. Johnson en AT&T para el sistema operativo Unix. Después se escribieron programas compatibles, por ejemplo Berkeley Yacc, GNU bison, MKS yacc y Abraxas yacc (una versión actualizada de la versión original de AT&T que también es software libre como parte del proyecto de OpenSolaris de Sun). Cada una ofrece mejoras leves y características adicionales sobre el Yacc original, pero el concepto ha seguido siendo igual. Yacc también se ha reescrito para otros lenguajes, incluyendo Ratfor, EFL, ML, Ada, Java, y Limbo.

Puesto que el analizador sintáctico generado por Yacc requiere un analizador léxico, se utiliza a menudo conjuntamente con un generador de analizador léxico, en la mayoría de los casos lex o Flex, alternativa del software libre. El estándar de IEEE POSIX P1003.2 define la funcionalidad y los requisitos a Lex y Yacc.

La versión Yacc de AT&T se convirtió en software libre; el código fuente está disponible con las distribuciones estándares del Plan 9 y de OpenSolaris.

VTK

VTK es un software de código abierto y orientado a objetos para el tratamiento de imágenes y su procesado. Aunque VTK es amplio y complejo, está diseñado para su fácil utilización siempre y cuando se haga un estudio previo de los conceptos básicos de diseño orientado a objetos y la metodología de implementación.

Debido a que VTK es un software de código abierto, son muchos los usuarios que han participado en su desarrollo.

ITK

Como ya se vio anteriormente, las librerías Insight Toolkit son de código abierto, multiplataforma, orientadas a objetos y se utilizan para procesar, segmentar y hacer correspondencia de imágenes. Insight Toolkit está diseñado para ser usado con facilidad una vez que se aprende diseño orientado a objetos y su metodología de implementación. A su vez, Insight Toolkit propor-

ciona métodos novedosos y a la orden del día para segmentación y correspondencia de imágenes en cualquier dimensión posible (2D, 3D,.., ND).

wxWidgets

wxWidgets es un conjunto de herramientas para implementar aplicaciones con interfaz gráfica (GUI). Es un framework en el sentido que hace muchas tareas de limpieza y proporciona un comportamiento para la aplicación por defecto. La librería de wxWidgets contiene un gran número de clases y métodos que facilitan su uso y personalización.

Las aplicaciones por norma general muestran ventanas que contienen controles básicos, son capaces de trazar imágenes y gráficos especializados y pueden responder a eventos de ratón, teclado u otras fuentes. También se puede comunicar con otros procesos y manejar otros programas. En otras palabras, wxWidgets hace que la implementación de una aplicación sea relativamente sencilla proporcionando todas las funcionalidades propias de las aplicaciones de hoy en día.

Puede que wxWidgets esté considera como un conjunto de herramientas para el desarrollo de GUI, pero de hecho es mucho más que eso ya que posee algunas características que la hacen muy útil en muchos aspectos del desarrollo de aplicaciones. Esto es así debido a que todas las aplicaciones implementadas con wxWidgets debe ser portables a diferentes plataformas, no sólo la parte gráfica.

wxWidgets proporciona clases para trabajar con ficheros y flujos de datos, ejecución multihilo, configuración de las aplicaciones, comunicación entre procesos, ayuda online, acceso a bases de datos y mucho más.

Boost

Las librerías de Boost C++, son un conjunto de librerías C++ portables peer-reviewed que extienden las funcionalidades de C++.

Las librerias de Boost están registradas bajo la licencia de software de Boost, diseñanadas para que Boost pueda ser utilizado tanto en proyectos de software libre como en proyectos privados.

Las librerías están destinadas a una amplia gama de usuarios de C++ y a un amplio dominio de aplicaciones. Van desde las librerías de propósito general como la libería smart_ptr (punteros inteligentes), abstracciones del funcionamiento del sistema como el sistema de ficheros de Boost, a las librerías principalmente dirigidas a usuarios avanzados de C++ y a desarrolladores de otras librerías, como el template de meta programación (MPL) y el de creación DSL (Proto).

Con el fin de garantizar la eficiencia y la flexibilidad, Boost hace un amplio uso de los templates.

Boost ha sido una fuente de intenso trabajo e investigación en la programación genérica y la meta programación en C++.

La actual versión de Boost contiene cerca de 80 liberías individuales, incluyendo librerías para algebra lineal, pseudo generación numérica, multihilos, procesamiento de imágenes, expresiones regulares, unidad de prueba, y muchas más.

La mayoría de las librerías de Boost están basadas en ficheros cabeceras, que consisten en funciones inline y templates, y como tal no tienen que ser contruidas antes de su uso.

El principal uso de estas librerías en AMILab es el de dar formato a las salidas por pantalla (printf en C++). A día de hoy y gracias a su utilidad se ha incluido el uso de los punteros inteligentes, permitiendo gestionar de manera más eficientes los recursos disponibles.

PThreads

Desde siempre, los diseñadores de hardware han implementado sus propias versiones de hilos. Estas implementaciones eran totalmente diferentes unas de otras haciendo que el desarrollo de aplicaciones multihilos portables fuese muy complicado.

Con el fin de aprovechar al máximo las capacidades proporcionadas por la ejecución multihilo, se hizo necesario una interfaz de programación estandarizada.

Para los sistemas Unix, esta interfaz viene especificada por el estándar IEEE POSIX 1003.1c (1995). Las implementaciones que utilizan este estándar son conocidas como hilos POSIX o Pthreads. Muchos diseñadores hardware ofrecen Pthreads además de su propia API.

El estándar POSIX ha seguido evolucionando y ha sido revisado en múltiples ocasiones, incluyendo la especificación de Pthreads. La última versión conocida es IEEE Std 1003.1, 2004.

Pthreads se define como un conjunto de tipos de lenguaje de programación C y llamadas a procedimientos, implementado con el fichero de cabecera *pthread.h* y la librería multihilos, aunque esta librería puede ser parte de otra librería, como *libc*, en algunas implementaciones.

Los principales motivos para utilizar multihilos son:

- Realizar mejoras sustanciales en el rendimiento del programa.
- Cuando se compara el coste de crear y manejar un proceso, el sistema multihilos es capaz de crearlo con una sobrecarga mucho menor del sistema operativo. El manejo de multihilos utiliza menos recursos del sistema que el manejo de procesos.
- Todos los hilos dentro de un proceso comparten el mismo espacio de direcciones. La comunicación entre hilos es mucho más eficiente y en muchos casos muchos más fácil de utilizar

que la comunicación entre procesos.

- Las aplicaciones multihilos ofrecen ganancias potenciales de rendimiento frente a las aplicaciones que no son multihilos en muchos sentidos:
  - Sobrecarga del procesador con entrada/salida: por ejemplo, un programa puede tener secciones en las que se está realizando una operación larga de entrada/salida. Mientras un hilo está esperando a que la llamada al sistema de entrada/salida se complete, el trabajo del procesador puede ser realizado por otros hilos.
  - Cola de prioridad: aquellas tareas que sean más importantes serán ejecutadas antes que aquellas que tienen menor prioridad, o incluso pueden interrumpirlas para ejecutarse ellas.
  - Manejo de eventos asíncronos: tareas que sus eventos de servicio son de frecuencia y duración indeterminadas pueden ser intercaladas. Por ejemplo, un servidor web puede, tanto transferir datos de las peticiones previas, como manejar la entrada de nuevas peticiones.
- La principal motivación para considerar el uso de Pthreads en un sistema multihilos es la de alcanzar un rendimiento óptimo.

Dado que AMILab es multiplataforma y que este proyecto ha sido desarrollado bajo Windows, se ha tenido que utilizar *Pthreads Win32*. Un gran número de sistemas operativos modernos incluyen una librería para el tratamiento de hilos propia: Solaris, Win32 threads, DCE threads, etc. La tendencia es que la mayoría de estos sistemas poco a poco adopten la API estándar de hilos (POSIX 1003.1-2001), pero este no es el caso de Win32, que es bastante improbable que lo haga, por lo que se ha de utilizar Pthreads Win32.

¿Cómo implementar un script en AMILab?

Una parte fundamental en AMILab es la existencia de scripts, esto permite al usuario la creación de una interfaz que facilitará la ejecución de los filtros o tareas que desee realizar.

Gracias a los recursos que posee AMILab, se puede crear una interfaz completa, con todas aquellas necesidades que requiera el usuario. Esto es posible debido a wxWidgets, ya que AMILab hace uso de sus funcionalidades y día a día éstas se amplían para proporcionar más y más recursos.

Un script en AMILab parte de un fichero ".amil" en el cuál se incluyen ciertas directivas que harán que se pueda mostrar la interfaz. A continuación se verá una pequeña descripción en la que se nombrarán diferentes aspectos a la hora de escribir una interfaz.

En primer lugar, lo más importante a tener en cuenta, es que AMILab es un lenguaje interpretado orientado a objetos por lo que cuando se crea un script se parte de la base que un script

será una clase, es decir, el script en sí será una clase que podrá ser instanciada tantas veces se quiera e incluso incluida y utilizada por otro script (clase). Partiendo de esto, como toda clase tendrá sus atributos y sus métodos.

Para poder inicializar nuestra clase, tenemos un método que se encarga de fijar todos los valores de los atributos a un valor por defecto, digamos que este método seria el constructor de nuestra clase. Dentro de este constructor podemos declarar atributos de los siguientes tipos:

- init
- boolean
- enum
- float
- string
- filename

Una vez sabemos los tipos que se pueden utilizar, es importante tener en cuenta los siguientes puntos:

- Al declarar una variable como int o enum se debe inicializar como INT("*valor*").
- Cuando declaras un tipo float no hace falta especificar al inicializarlo que es tipo float, porque por defecto ya sabe que lo es.
- Una variable de tipo boolean debe ser inicializada con el tipo UCHAR("*valor*").
- Al inicializar una variable de tipo string o filename basta con asignarle el valor que queramos que tenga.

Otro método importante en un script es el que se encarga de crear la interfaz propiamente dicha. Esta puede estar formada por:

- AddInt $\rightarrow$ f

¿Cómo añadir un filtro en AMILab?

Ejecución de scripts

7.1.4. Profiling

7.2. Actividad 2 Análisis

7.2.1. ¿Cómo implementar un filtro?

Los filtros se definen en base al tipo de datos de entrada y en base al tipo de datos de salida. La clave para escribir un filtro en Insight Toolkit es identificar el número y tipo de las entradas y salidas. Una vez hecho esto, hay superclases que simplifican la tarea a través de la derivación. Por ejemplo, la mayoría de los filtros en Insight Toolkit toman una imagen como entrada y producen una única imagen como salida. La superclase itk::ImageToImageFilter es la clase más adecuada para este filtro.

- "ImageToImageFilter" → es el filtro más común utilizado para algoritmos de segmentación. Toma una imagen y produce una nueva imagen, por defecto con el mismo tamaño.

Consideraciones a tener en cuenta

Multihilos

Los filtros que pueden procesar datos por partes pueden normalmente trabajar utilizando los datos en paralelo. Para crear un filtro multihilo, simplemente se define y se implementa el método "ThreadedGenerateData()". Por ejemplo, itk::ImageToImageFilter crearía el método: *void ThreadedGenerateData(const OutputImageRegionType &outputRegionForThread, int threadId)*

La clave del procesamiento multihilo es generar una salida para la región de salida dada (como primer parámetro de la lista de argumentos). En Insight Toolkit esto es fácil de hacer porque el iterador de salida puede ser creado utilizando la región proporcionada. Por tanto, la salida puede ser iterada, accediendo a los pixeles correspondientes de la entrada según sea necesario para calcular los valores de los pixeles de salida.

La ejecución multihilos necesita tener cuidado cuando trabaja con I/O (cerr o cout) o cuando invoca eventos. Un método seguro es sólo permitir que un único hilo con identificador cero realice I/O o genere eventos. (El identificador id es pasado como argumento en "ThreadedGenerateData").

Si más de un hilo intenta escribir en el mismo sitio al mismo tiempo, el programa puede comportarse de manera errónea y posiblemente se produzca un fallo.

Convenios a la hora de implementar un filtro

Se pretende que los filtros en Insight Toolkit sigan una serie de convenios. A continuación se explican los requisitos mínimos para la integración de un filtro en Insight Toolkit.

La declaración de la clase para un filtro debe incluir la macro "ITK_EXPORT", de manera que en determinadas plataformas una declaración de exportación pueda ser incluida.

Un filtro debe definir tipos públicos para la clase propia (Self), para su superclase (Superclass) y para los punteros inteligentes constantes y no constantes:

typedef ExampleImageFilter Self;

typedef ImageToImageFilter¡TImage,TImage¿Superclass;

typedef SmartPointer¡Self¿Pointer;

typedef SmartPointer¡const Self¿ConstPointer;

El tipo Pointer es bastante útil, ya que es un puntero inteligente que será utilizado por todos los usuarios del código para mantener un contador de referencias al filtro (instancias al filtro).

Una vez tenemos definidos estos tipos, se pueden utilizar las siguientes macros, las cuales permiten al filtro implementado participar en el mecanismo de fábrica de objetos y ser creado utilizando el método New().

- Método para crear a través de la fábrica de objetos → "itkNewMacro(Self)"
- Escritura en tiempo de ejecución de la información y de los métodos relacionados → "itkTypeMacro(ExampleImageFilter, ImageToImageFilter)"

El constructor por defecto debe ser "protected". La copia del constructor y el operador de asignación deben ser declarados privados y no se deben implementar para impedir instanciaciones del filtro sin los métodos de la fábrica.

Por último, la implementación del código template debe ser incluida y catalogada por un test para una instanciación manual:

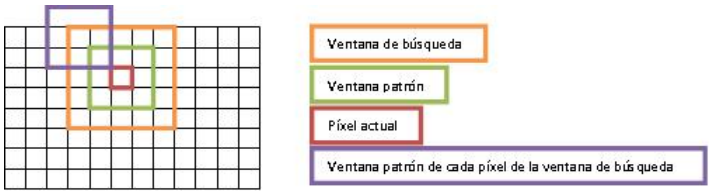

Figura 7.4: Non Local Means

#ifndef ITK_MANUAL_INSTANTIATION

#include "itkExampleFilter.txx"

#endif

7.2.2. Filtro NLMeans Básico

Dada una imagen con ruido el valor estimado para un punto se calcula como una media ponderada de todos los píxeles de la imagen, donde los pesos dependerán de la similitud entre dos píxeles dados.

El algoritmo NLMeans básico se centra en que cada para cada píxel de la imagen de entada se calculan dos vecindades alrededor del píxel, una será la ventana de búsqueda (t) y otra la ventana patrón (f). Para cada ventana de búsqueda y cada píxel actual se calcula su media ponderada en relación al resto de píxeles de la ventada de búsqueda y a sus ventanas patrón. Esto se hace de la siguiente forma:

1. Se delimita la ventana de búsqueda centrada en el píxel actual.
2. Se delimita la ventana patrón centrada en el píxel actual.
3. Para todos los píxeles de la ventana de búsqueda se calculará la distancia entre la ventana patrón del píxel actual y cada ventana patrón de cada píxel de la ventana de búsqueda. A continuación se hace la promedia y se continúa el proceso con el siguiente píxel de la ventana de búsqueda. Así hasta completarla.
4. Una vez se termina con la ventana de búsqueda se hace la media en base a las promedias obtenidas y se fija un valor para el píxel del salida.

En base a lo expuesto con anterioridad, se deben seleccionar aquellas clases que se pueden reutilizar para el desarrollo del algoritmo.

Gracias al estudio realizado en una primera fase de las librerías ITK se ha tomado la decisión de utilizar las siguientes clases para el desarrollo del filtro en cuestión:

- La clase relativa a las imágenes en ITK, "itkImage.h". Se utiliza esta clase ya que permite tener imágenes de cualquier dimensión (n-D), siendo el tipo de los píxeles cualquiera.

- La clase utilizada en el tratamiento de filtros, aquella que a partir de una imagen de entrada, la transforma y genera otra imagen de salida, "itkImageToImageFilter.h". Esta clase además proporciona soporte para el procesamiento de imágenes mediante multihilos.

- Las clases que permite iterar sobre toda la imagen, "itkImageRegionIterator.h" e "itkImageRegionConstIteratorWithIndex.h". La primera permite iterar sobre una región de una imagen de forma rápida. La segunda realiza lo mismo que la primera pero con la diferencia de que en cada píxel de la imagen tenemos el índice del mismo, es decir, sabemos la posición del iterador en cada momento. Este hecho produce que el iterador sea más lento pero no tanto como si para el primer iterador le pidiésemos el índice del iterador en cada momento.

- La clase relativa al tratamiento de la ventana patrón, es decir, la clase iterador de vecindades, "itkConstNeighborhoodIterator.h". Permite tener para cada píxel de la imagen una vecindad del tamaño que se desee, normalmente suele ser un tamaño pequeño ya que si no se pierde eficiencia.

- Para poder hacer un uso eficiente de los iteradores de vecindades, se activará la opción de comprobación de los límites. Esto permite, que sólo aquellas zonas cercanas a los límites sean evaluadas como tal, mientras que el resto se recorrerán de forma normal obteniendo de esta forma un aumento significativo de la velocidad, para ello se hará uso del objeto "itk::ImageBoundaryFacesCalculator" que se encuentra en la clase "itkNeighborhoodAlgorithm.h". Dividir la imagen en las regiones necesarias es una tarea sencilla cuando se utiliza este objeto, se llama así porque devuelve una lista con las "caras" del conjunto de datos N-D. Las caras son aquellas regiones donde todos los píxeles se encuentran a una distancia "d" del límite, siendo "d" el radio del patrón de vecindades utilizado. En otras palabras, las "caras" son aquellas regiones donde el iterador de vecindades de radio "d" siempre se superpondrá a los límites de la imagen. Este objeto también devuelve el interior de la región, en la cual los valores fuera de los límites nunca serán utilizados y el chequeo de límites no es necesario.

- Es necesario leer la imagen de entrada y poder escribir la imagen resultante, para ello se utilizarán las siguientes clases, "itkImageFileReader.h" e "itkImageFileWriter.h". Utilizando el tipo de datos de la imagen se puede instanciar la clase encargada de leer la imagen. El tipo de la imagen es utilizado como un parámetro template para definir cómo los datos serán representados una vez sean cargados en memoria. Este tipo no tiene porque corresponder exactamente con el tipo almacenado en el fichero. Se puede realizar una conversión de un tipo a otro, siempre y cuando el tipo elegido para representar los datos en el disco admita dicha conversión.

7.2.3. Segmentación de la carótida

Clase BackTracking

7.3. Actividad 3 Análisis

Capítulo 8

Diseño

8.1. Actividad 1 Diseño

8.1.1. Multihilos

Insight Toolkit está diseñado para trabajar en entornos multiprocesador. Muchos de los filtros de Insight Toolkit usan multihilos. Cuando un filtro que usa multihilos se ejecuta, este automáticamente divide el trabajo entre los procesadores haciendo uso de la memoria compartida, esto se llama "Filter Level Multithreading".

Las aplicaciones compiladas con Insight Toolkit pueden también manejar la ejecución de sus propios hilos. Por ejemplo, una aplicación podría utilizar un hilo para procesar información y otro hilo para la interfaz de usuario, esto se llama "Application Level Multithreading".

Un filtro que utiliza multihilos proporciona una implementación del método "ThreadedGenerateData()". La superclase del filtro producirá varios hilos (normalmente se corresponde con el número de procesadores del sistema) y llamará a "ThreadedGenerateData()" por cada hilo especificando la parte de la salida que cada hilo debe generar. Por ejemplo, en un ordenador con dual core, un filtro que procese una imagen generará dos hilos, cada hilo producirá una de las mitades de la imagen de salida y a cada hilo sólo se le permite escribir en su porción de imagen asignada. Un detalle a tener en cuenta es que la imagen completa de entrada y la imagen completa de salida está disponibles en cada llamada de "ThreadedGenerateData()". Cada hilo es libre de leer cualquier posición de la imagen de entrada pero en el momento de la escritura sólo pueden hacerlo en su porción asignada.

La imagen de salida es un único bloque contiguo de memoria que es utilizado por todos los hilos. Cada hilo tiene asignado una serie de pixeles para los que tiene que generar los valores

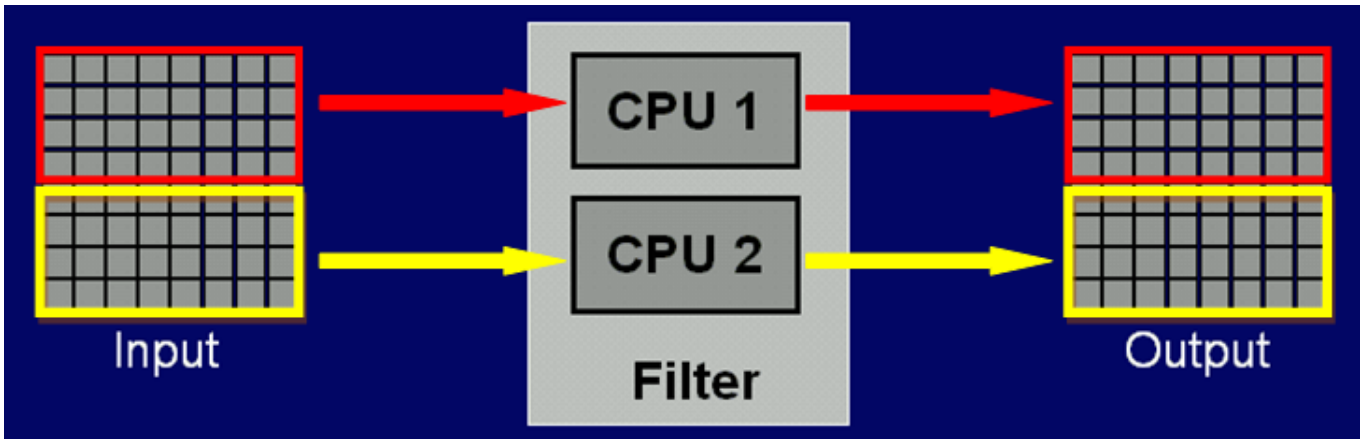

Figura 8.1: Multihilos [1]

correspondientes. Todos los hilos escriben en el mismo bloque de memoria pero a cada hilo se le permite escribir sólo un conjunto de píxeles.

8.1.2. Wrapping en AMILab

Se ha propuesto realizar una mejora en la distribución de los filtros ITK en AMILab. Lo que se ha buscado es que cada filtro que se añada a AMILab esté contenido en un fichero diferente para así facilitar el entendimiento del código.

En un principio todos los filtros ITK que se añadían a AMILab venían contenidos en un mismo fichero, lo que provocaba códigos demasiados largos y difíciles de seguir. Se partía de un fichero cabecera .h y un fichero dónde estaban las implementaciones .cpp. En el fichero cabecera estaban contenidas las cabeceras de los nuevos filtros y sobretodo la cabecera del procedimiento encargado de añadir una nueva variable a AMILab, la cual será la encargada de permitir la llamada y ejecución del nuevo filtro añadido. Este procedimiento es de vital utilidad ya que nos permite ir añadiendo, de manera sencilla, funcionalidades a AMILab.

A día de hoy, lo que se ha buscado es que cada nueva funcionalidad venga englobada en un fichero por separado, de manera que facilite el entendimiento de los mismos y permita que las posibles modificaciones que se necesiten realizar se hagan de manera más sencilla y rápida. Se ha mantenido un fichero .cpp que contiene la implementación del procedimiento que permite añadir funcionalidades a AMILab, de manera que cada vez que se quiera añadir un nuevo filtro se ha de crear su variable en este fichero. A su vez cada nueva funcionalidad tendrá un fichero .h y un .cpp dónde estarán la cabecera e implementación de cada una.

Por tanto, tendremos dos ficheros por cada nueva funcionalidad y dos ficheros que se irán incrementando encargados de añadir dichas funcionalidades a AMILab.

8.1.3. Filtro Non Local Means Básico

El diseño principal de nuestro filtro "Basic NL-Means" viene determinado por las consideraciones especificadas por Insight Toolkit a la hora de programar un filtro, por tanto, la realización del mismo se ha basado en estas restricciones.

A continuación recordaremos estas restricciones para tener presente cada paso que hemos dado en el diseño de dicho filtro. También cabe destacar que gracias a la gran cantidad de herramientas de que disponemos en Insight Toolkit hemos podido partir de otro filtro similar que nos ha servido de ayuda.

En primer lugar debemos tener en cuenta que Insight Toolkit tiene utiliza la filosofía multihilos, característica que nos será de gran ayuda a la hora de procesar imágenes de gran tamaño y dimensión.

Otra de las características de Insight Toolkit es que parte de una programación genérica, que como ya se ha visto se fundamenta en los contenedores que se utilizan para almacenar datos, los iteradores para acceder a estos datos y los algoritmos genéricos que usan contenedores e iteradores para crear de manera eficiente algoritmos fundamentales tales como la clasificación. También es importante resaltar que la programación genérica viene soportada por un mecanismo de programación que utiliza template, proporcionando de esta forma reusabilidad al código y a las funciones.

Otra de las ventajas a la hora de utilizar template es que Insight Toolkit busca poder tratar cualquier imagen de cualquier tipo de datos y sin tener en cuenta la dimensión de la misma. Este hecho queda cumplido gracias a los template.

Por último, Insight Toolkit tiene establecida una "clasificación" en la que ir englobando cada una de las funcionalidades que proporciona, sobre todo en lo que concierne a los filtros, por tanto:

- Filtros de imágenes
- Filtros de mallas
- Filtros de transformaciones geométricas
- Filtros de correspondencia entre imágenes
- Filtros de entrada/salida (I/O)

En el caso en el que estamos nuestro filtro irá englobado en filtros de imágenes, ya que el mecanismo de procesado es la adquisición de una imagen para su tratamiento, la aplicación del filtro en cuestión y la generación de una nueva imagen. Los filtros se definen respecto al tipo de datos de entrada (sea cual sea) y al tipo de datos de salida (sea cual sea). La clave para escribir un filtro en Insight Toolkit es la de identificar el número y tipo de las imágenes de entrada y de salida. Una vez

tengamos eso, normalmente existe una superclase que simplifica esta tarea a través de derivación de clases. Por ejemplo, casi todos los filtros en Insight Toolkit toman una imagen simple como entrada y producen una imagen simple como salida. La superclase itk::ImageToImageFilter es la clase adecuada que proporciona la suficiente funcionalidad para ese filtro.

Algunas clases base para incluir nuevos filtros son:

- ImageToImagFilter: el filtro base más común para los algoritmos de segmentación. Toma una imagen y produce una nueva imagen, por defecto con la misma dimensión.
- UnaryFunctorImageFilter: se utiliza cuando se define un filtro que aplica una función a una imagen.
- BinaryFunctorImageFilter: se utiliza cuando se define un filtro que aplica una operación a dos imágenes.
- ImageFunction: un funtor que puede ser aplicado a una imagen, evaluando f(x) en cada punto de la imagen.
- MeshToMeshFilter: un filtro que transforma mallas, como mosaicos, reducción de polígonos, etc...
- LightObject: base abstracta para los filtros que no se pueden incluir en ninguna otra clase de la jerarquía.

Una vez que se identifica la superclase a utilizar, el escritor del filtro implementa la clase definiendo los métodos requeridos por la gran mayoría de los objetos en Insight Toolkit: New(), PrintSelf(), un constructor protegido, una copia del constructor, un destructor y un operador =. Es importante no olvidar la definición de tipos estándar como Self, Superclass, Pointer y ConstPointer. A continuación, el escritor del filtro puede centrarse en las partes más importantes de la implementación: definición de la API, data members y otros detalles de la implementación del algoritmo. En particular, el escritor del filtro tendrá que implementar el método GenerateData() o ThreadedGenerateData().

Un detalle importante es que el método GenerateData() necesita reservar memoria para la salida mientras que el método ThreadedGenerateData() no. En las implementaciones por defecto GenerateData() reserva memoria y después invoca a ThreadedGenerateData().

Una de las decisiones más importantes que un desarrollador debe tomar es si el filtro separará la imagen en partes; esto quiere decir que sólo se procesará una porción de la entrada para generar una porción de la salida. A menudo, la superclase realiza bien su trabajo: si el filtro procesa la entrada utilizando acceso único a píxeles, entonces el comportamiento por defecto será el adecuado. Si no es así, el usuario tendrá que:

1. Obtener una clase más especializada para derivar a partir de ella.

2. Sobrescribir uno o más métodos que controlen cómo el filtro operará durante la ejecución del pipeline. Estos métodos se describirán a continuación.

Los datos asociados con las imágenes multidimensionales son grandes y pueden llegar a ser más grandes. Esta tendencia se debe a los avances de en la resolución de escaneo, así como los aumentos de capacidad de computación. Cualquier software de segmentación o de correspondencia de imágenes debe hacer frente a este hecho con el fin de ser útil en cualquier aplicación. Insight Toolkit aborda este problema a través de su facilidad de flujo de datos ("data streaming").

En Insight Toolkit, "streaming" es el proceso de dividir información en partes o en regiones, y a continuación procesar esta información a través del pipeline. Recordar que el pipeline se basa en un conjunto de objetos de procesos que generan objetos de datos conectados en la topología del pipeline. La entrada a un objeto de proceso es un objeto de dato (a menos que el proceso inicialice el pipeline y a continuación sea una fuente de objeto de proceso). Estos objetos de datos, a su vez, se consumen por otros objetos de procesos, y así sucesivamente, de manera que se construye un gráfico de flujo de datos. Finalmente el pipeline termina por uno o más "mappers", que podrán escribir información a almacenar, o por una interface gráfica u otro sistema.

El beneficio significativo que se consigue con esta arquitectura es que la relativa complejidad del proceso de manejo de ejecución del pipeline está diseñada en el sistema. Esto quiere decir que el mantenimiento del pipeline al día, la ejecución en el pipeline sólo de aquellas partes que hayan cambiado, la ejecución utilizando multihilos, el manejo de la reserva de memoria y el "streaming" son todos construidos en la arquitectura. Sin embargo, estas características introducen complejidad al sistema. A continuación se describirá el proceso de ejecución del pipeline en detalle, centrándonos en el flujo de datos ("data streaming").

El proceso de ejecución del pipeline realiza varias funciones importantes:

1. Determina qué filtro, en un pipeline de filtros necesita ser ejecutado. Esto evita la ejecución redundante y minimiza el tiempo de ejecución global.

2. Inicializa la salida de los objetos de datos, preparándose para nueva información. Además determina qué cantidad de memoria debe reservar cada filtro para su salida y la reserva.

3. El proceso de ejecución determina qué cantidad de información debe procesar un filtro con el fin de producir una salida lo suficientemente grande para los filtros; también tiene en cuenta los límites de la memoria o las necesidades especiales del filtro. Otro de los factores incluye el tamaño de la información que son capaces de procesar los núcleos, que afecta a la cantidad de datos que se necesita.

4. Subdivide la información en subpartes utilizadas para la ejecución multihilos. (Se debe tener en cuenta que la división de la información en subpartes es exactamente el mismo problema

de dividir la información en partes para el "streaming"; por tanto, la ejecución multihilos se incluye como parte de la arquitectura del "streaming").

5. Puede liberar la información de salida si los filtros no la necesitan más para calcular, y el usuario puede solicitar que la información sea liberada. (Se debe tener en cuenta que los objetos de salida de los filtros se consideran como una caché. Si a la caché se le permite permanecer entre la ejecución del pipeline y el filtro, o la entrada del filtro nunca cambia, entonces los objetos de procesos de los filtros sólo utilizarán la caché del filtro para volver a ejecutarse.)

Para realizar estas funciones, el proceso de ejecución negocia con los filtros que definen el pipeline. Sólo cada filtro puede saber cuánta información se necesita en la entrada para generar una cierta salida. Por ejemplo, un filtro de reducción con un factor de reducción de dos necesita una imagen dos veces mayor que la entrada para generar una salida en particular. Un filtro de convolución de imágenes necesita una entrada extra dependiendo del tamaño del núcleo de convolución. Algunos filtros necesitan toda la imagen de entrada para producir la salida (por ejemplo, un histograma), y tienen la opción de solicitar toda la imagen de entrada. (En este caso el "streaming" no se ejecutará a menos que el desarrollador cree un filtro que sea capaz de solicitar la información en partes, tomando el estado entre cada parte y juntándolo todo al final como salida.)

En última instancia, el proceso de negociación está controlado por la solicitud de información de un cierto tamaño (por ejemplo, una región). Puede ser que el usuario solicite que se procese una región de interés dentro de la totalidad de la imagen, o que debido a las limitaciones de memoria se procese la información en varias partes. Por ejemplo, una aplicación puede calcular la memoria que necesita el pipeline y después utilizar el "streaming" para dividir la información a procesar en varias partes. La información solicitada se propaga a través del pipeline en dirección ascendente y el proceso de negociación configura cada filtro para producir la información de salida de un cierto tamaño.

El secreto para crear un filtro de "streaming" es entender cómo trabaja el proceso de negociación, y cómo anula su comportamiento por defecto utilizando las funciones virtuales apropiadas.

Por regla general la ejecución del pipeline se inicializa cuando un objeto de proceso recibe la invocación del método "ProcessObject::Update()". Este método es simplemente delegado a la salida del filtro, invocando al método "DataObject::Update()". Este comportamiento es normal en la interacción entre "ProcessObject" y "DataObject": un método invocado en uno es normalmente delegado en el otro. En este sentido la información solicitada por el pipeline es propagada hacia arriba, inicializando el flujo de datos que retornará.

El método "DataObject::Update()" a su vez invoca a otros tres métodos:

- DataObject::UpdateOutputInformation().
- DataObject::PropagateRequestedRegion().

- DataObject::UpdateOutputData().

Una vez planteadas estas consideraciones, pasaremos a analizar más en profundidad el problema que nos concierne, el filtro "Basic NL-Means". En primer lugar tendremos en cuenta que al utilizar un mecanismo de programación que utiliza template se suele aplicar el convenio de utilizar dos archivos, uno ".h" que contendrá las cabeceras y puede que los constructores y destructores y un ".txx" dónde se implementarán las funciones template.

Nuestro filtro está diseñado de manera que se ha creado una clase formada por una zona pública, otra privada y otra protegida. En cada una de ellas irán los métodos y atributos correspondientes. A continuación de comentará lo que se ha establecido en cada una de estas "zonas":

- Zona pública contendrá los tipos públicos de la propia clase, de la superclase que utiliza y de los punteros inteligentes constantes o no. También tendrá todas aquellas macros utilizadas para poder agregar "parámetros" a nuestro filtro usando el mecanismo de "object factory" proporcionado por Insight Toolkit.

- Zona protegida el constructor por defecto irá aquí y proporcionará por defecto valores para todos los parámetros. También declararemos aquí el destructor y las funciones propias de nuestro filtro, las cuales harán uso de la opción multihilos. Por último, deberemos agregar la cabecera de la función de testeo de Insight Toolkit.

- Zona privada la copia del constructor y del operador de asignación deben ser declarados como privados y no deben ser implementados para prevenir la instanciación del filtro sin los métodos de "object factory". Dentro de la zona privada deben figurar todos aquellos parámetros necesarios para la aplicación de nuestro filtro. Los valores de los mismos serán establecidos mediantes las macros definidas en la zona protegida de la clase.

Por último, el nombre del fichero el cual contiene la implementación del código template (fichero ".txx") debe ser incluido. Por tanto, lo expuesto anteriormente concierne al fichero con extensión ".h".

En cuanto al fichero ".txx" deberemos incluir aquellas funciones necesarias para el tratamiento de los datos teniendo en cuenta que usaremos multihilos. Por tanto, necesitamos dos funciones principales, una encargada de procesar los datos antes de realizar el filtro en cuestión y otra que lo ejecutará.

Debemos tener en cuenta que como utilizamos multihilos la imagen se dividirá en tantos procesadores como tengamos, por lo que la ejecución del filtro se llevará a cabo en un tiempo menor.

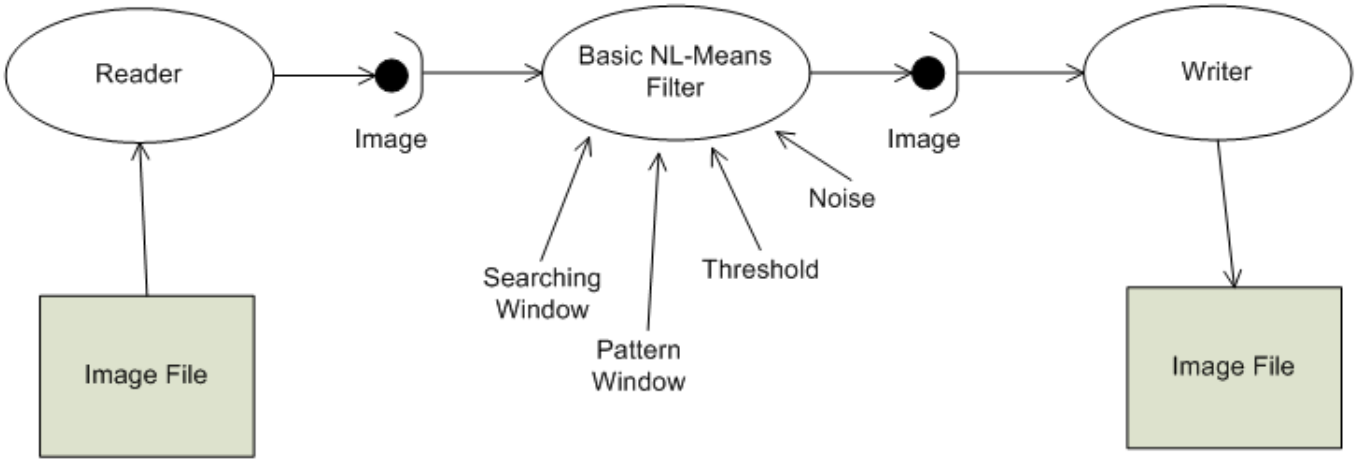

Figura 8.2: Pipeline

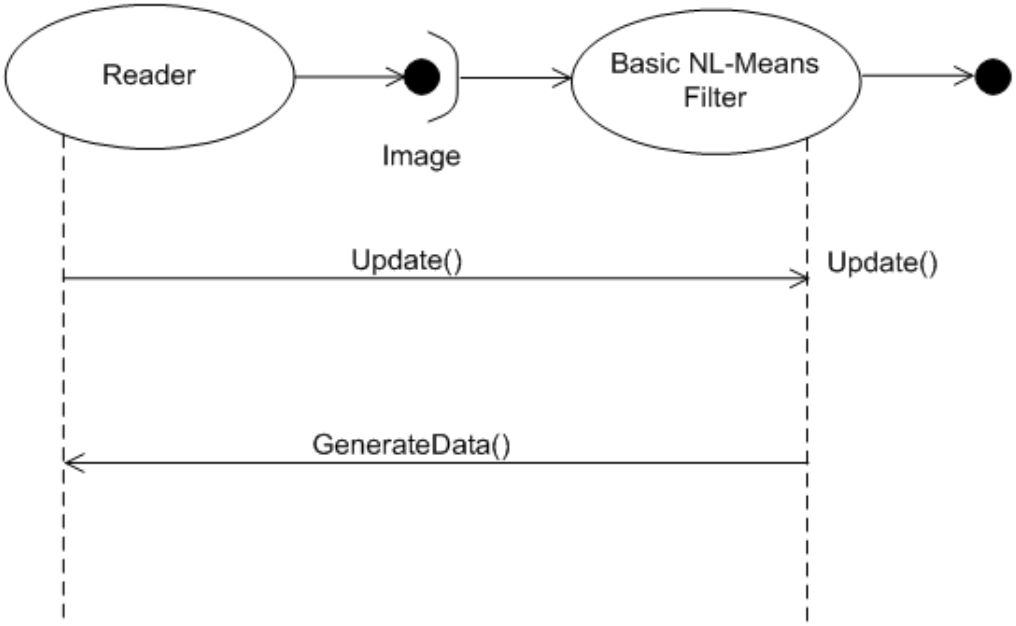

Figura 8.3: Pipeline

8.1.4. Segmentación de la carótida

Clase BackTracking

Diagrama de clases

Casos de uso

Capítulo 9

Implementación

9.1. Filtro Non Local Means Básico con iteradores de vecindades

El algoritmo Non Local Means, como ya se ha explicado anteriormente, es un filtro que facilita la eliminación de ruido de una imagen. Partiendo de la base de que dicho método se puede implementar de múltiples maneras, aquí hemos optado por la forma básica, la más lenta. En un futuro se pretende realizar las implementaciones más rápidas para comprobar si ITK favorece su rapidez o todo lo contrario.

Para ayudarnos a implementar el método Non Local Means nos hemos ayudado de un filtro propio de ITK que calcula la media local. Hemos seleccionado esta implementación debido a que hace uso de los iterados de vecindades, gran utilidad, que en principio nos será de gran ayuda.

Debido al diseño implícito de ITK, nuestro filtro constará con dos métodos que se ejecutarán en base a los procesadores y por consiguiente hilos que se lancen.

Tenemos un primer método que se ejecutará antes del método principal, es el encargado de crear un vector que contendrá una serie de pesos ponderados para después ser utilizados en la reducción de ruido de la imagen. Estos pesos serán creados partiendo básicamente del tamaño de la ventana patrón, f.

El segundo método a ejecutar es el encargado de aplicar el filtro en cuestión, se basa en el hecho del uso de múltiples hilos para su aplicación. En base al número de procesadores se lanzará dicho método por cada procesador, procesando la imagen en secciones y de manera más rápida. El método implementa el algoritmo de Non Local Means, explicado anteriormente, teniendo en cuenta que tendremos varias regiones de la imagen sobre las que iremos trabajando.

Gracias a los iteradores de vecindades podemos controlar que los mismos no se salgan de los límites de la imagen establecidos. ITK tiene la opción de activar la comprobación de bordes o no en sus iteradores de vecindades, de esta forma permite que se realice un trabajo más eficiente y evita que se hagan comprobaciones innecesarias. Para reducir estas comprobaciones hemos implementado el método de forma que la ventana de búsqueda, t, sea calculada aparte, por lo que para ella no haremos uso de los iteradores de vecindades.

Otro detalle a tener en cuenta es el uso de "caras", esto permite que una región se divida en varias pequeñas regiones, haciendo distinción entre las que tocan el borde y las que no. Esto permite que para aquellas zonas que no sea necesario comprobación de bordes no se realice.

Uno de los problemas que se plantea y que perjudica al tiempo de ejecución del algoritmo es el uso del método GetPixel, ya que la utilización del mismo, como bien se plantea en ITK, hace que se tarde más en acceder al valor de ese pixel en concreto. La filosofía de ITK busca la utilización de otras herramientas que pone a disposición de los usuarios, el problema en este caso es que al utilizar los iteradores de vecindades nos vemos obligados a utilizar este método.

Por último, cabe destacar que los dos métodos implementados son de tipo template, permitiendo que la reducción de ruido se aplique sobre cualquier tipo de imagen. La única restricción es que se haga sobre una imagen 2D o 3D.

9.2. Nuevas funcionalidades de AMILab

El primer detalle a tener en cuenta es que hemos cambiado la organización de las funciones implementadas utilizando ITK, hemos pasado de tener todo en un solo archivo a tenerlo en archivos separados. Ahora por cada función nueva se crea un nuevo archivo que formará parte del conjunto WrapITK en el cuál se engloban todas aquellas funciones que AMILAB utiliza y que se implementan usando librerías ITK.

Se parte del fichero "wrapITK.cpp" en el que se declara un procedimiento propio de AMILAB que sirve para añadir nuevas variables al programa. Gracias a este procedimiento podemos incluir, siempre que queramos, funcionalidades implementadas en ITK. Para ello debemos hacer uso de la adición de una variable en la que se especificará el tipo de la funcionalidad a incluir:

- Type_void
- Type_image
- Type_float
- Type_int
- Type_uchar

- Type_string
- Type_imagedraw
- Type_surface
- Type_surfdraw
- Type_file
- Type_c_procedure
- Type_c_image_function
- Type_c_function
- Type_ami_function
- Type_ami_class
- Type_ami_object
- Type_paramwin
- Type_parampanel
- Type_matrix
- Type_gltransform
- Type_array

A continuación se especifica el nombre de la funcionalidad a incluir, que será el que usemos a la hora de invocarla. Gracias a estos sencillos pasos hemos logrado incluir nuestra nueva función en AMILAB, digamos que tenemos el cuerpo de la misma, ahora bastará con añadir la implementación para poder utilizarla.

Cada nueva funcionalidad será implementada en un fichero a parte permitiendo de esta forma un mayor entendimiento del código, acceso al mismo (posibles modificaciones) y facilidad de uso.

Un paso importante que se ha dado es la creación de dos funciones de tipo template encargadas de transformar una imagen ITK en el tipo de imagen propio de AMILAB (InrImage) y viceversa. De esta forma cuando se invoca una funcionalidad desde AMILAB se le pasará una imagen del tipo InrImage que deberá ser transformada en el formato de ITK, para una vez aplicado el filtro o el tratamiento a esta imagen reconvertirla en el formato InrImage pudiendola utilizar por AMILAB.

Gracias a esta pequeña modificación en el código logramos que no exista código repetido, lo que provoca implementaciones poco legibles y pesadas de entender.

Las dos funciones de tipo template implementadas están englobadas en el archivo wrapConversion.tpp. El hecho de que sean de tipo template nos permite que sean utilizadas con cualquier tipo de imagen que se necesite.

9.2.1. wrapITKWrite

Esta nueva funcionalidad permite guardar imágenes, para posteriormente poder utilizarlas en otros filtros, por ejemplo. Los parámetros de entrada serán una imagen en 2D o 3D, de tipo:

- Unsigned_char
- Unsigned_short
- Signed_short
- Unsigned_int
- Signed_int
- Unsigned_long
- Float
- Double

y la ruta dónde queremos que se guarde la imagen.

El mecanismo mediante el cual se guarda la imagen viene dado por la creación de la clase itkWriteClass en la que se hace uso de la clase itkImageFileWriter proporcionada por ITK. Esta clase permite guardar una imagen, tan sólo se le debe pasar por parámetro la ruta dónde se va a guardar la imagen y la imagen a guardar.

Por tanto, la nueva funcionalidad implementada toma una imagen de formato InrImage, la transforma en formato legible por ITK y la guarda en la ruta indicada.

9.2.2. wrapITKBackTrackingMeshFilter

9.2.3. wrapITKBasicNLMeansFilter

9.2.4. wrapITKBinaryThresholdImageFilter

9.2.5. wrapITKDICOMRead

9.2.6. wrapITKFastMarchingImageFilter

9.2.7. wrapITKLevelSet

9.2.8. wrapITKMultiScaleVesselnessFilter

9.2.9. wrapITKRecursiveGaussianImageFilter

9.2.10. wrapITKSigmoidImageFilter

9.2.11. wrapITKWaterShedImageFilter

9.3. Script Proceso Completo

Capítulo 10

Validación

Capítulo 11

Resultados y conclusiones

Bibliografía

[1] L. Ibáñez, W. Schroeder, L. Ng, J. Cates. *"The ITK Software Guide"* 2005

[2] W. Schroeder *"The VTK User´s Guide"* Kitware,Inc. 2001

[3] J. Smart, K. Hock, S. Csomor *"Cross-Platform GUI Programming with wxWidgets"* Prentice Hall PTR 2005

[4] Boost C++ Libraries *http://en.wikipedia.org/wiki/Boost_C%2B%2B_Libraries#Overview*

[5] The Visualization Toolkit *http://www.vtk.org/*

[6] PThreads - POSIX Threads *https://computing.llnl.gov/tutorials/pthreads/*

[7] Pthreads Win32 *http://sourceware.org/pthreads-win32/*

[8] CMake *http://en.wikipedia.org/wiki/CMake*

[9] AMILab *http://www.ctm.ulpgc.es/amilab_dokuwiki/dokuwiki/doku.php*

[10] Lex *http://es.wikipedia.org/wiki/Herramienta_de_programaci%C3%B3n_lex*

[11] Yacc *http://es.wikipedia.org/wiki/Yacc*