Universidad de las Ciencias Informáticas
Vertex, Interactive 3D Environments, Facultad 4

Módulo de post-procesamiento para Visualización Ilustrativa de Volúmenes usando GPU

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Karel Díaz Alfonso

Tutores: MSc. Luis Guillermo Silva Rojas

MSc. Rubén Alcolea Núñez

La Habana, 2018

Art challenges technology and technology inspires art.
John Lasseter

Dedicatoria

A mis padres, mi hermano y a mis familiares y amigos.

Agradecimientos

A mis tutores Luis Guillermo Silva Rojas y Rubén Alcolea Núñez.

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales sobre esta, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

Karel Díaz Alfonso
Autor

MSc. Luis Guillermo Silva Rojas
Tutor

MSc. Rubén Alcolea Núñez
Tutor

Resumen

En la medicina se emplean frecuentemente ilustraciones para representar los órganos y los sistemas internos del cuerpo. En la actualidad, con los avances en los gráficos por computadoras, es posible generar ilustraciones interactivas a partir de datos volumétricos obtenidos mediante Tomografías Computarizadas o Resonancias Magnéticas, un ejemplo de esto es el proyecto Vismedic, desarrollado en el centro Vertex de la Universidad de las Ciencias Informáticas. La versión actual del proyecto, nombrada Vismedic - Illustration, cuenta con técnicas ilustrativas de elevado costo computacional, puesto que los filtros se ejecutan para todo el volumen 3D. La ejecución de los filtros en tres dimensiones disminuye considerablemente el rendimiento de la visualización y no permite la combinación dinámica de diferentes tipos de filtros para obtener resultados más complejos. Para dar solución a este problema, el presente trabajo propone desarrollar un módulo de post-procesamiento en GPU de las imágenes obtenidas por el algoritmo Raycasting, que permita adicionar filtros ilustrativos sin afectar considerablemente el rendimiento. En el módulo propuesto se ofrecen interfaces para la creación e integración con Vismedic de *plugins* de post-procesamiento y para la implementación de filtros que utilicen la tarjeta gráfica. Adicionalmente los filtros pueden contener otros filtros en su interior, con lo que se consiguen efectos complejos, producto de la concatenación de los resultados del procesamiento. Se adicionaron un total de cinco filtros principales (Desenfoque Gaussiano, Sobel, Glow, Sharpen y Posterize) y cinco complementarios (Brillo, Contraste, Tono, Sepia e Invertir color). Durante las pruebas realizadas, para una única instancia de cualquiera de los filtros implementados no se observó impacto perceptible sobre el rendimiento de la visualización. Aun para condiciones extremas (50 filtros), el rendimiento solo disminuyó en 8 cuadros por segundo, lo que demuestra la interactividad del módulo. Los resultados visuales de la aplicación de los filtros implementados se mostraron similares a los obtenidos mediante el *software* Adobe Photoshop, lo que valida la calidad de la imagen resultante.

Palabras clave: efecto, filtro, ilustrativo, interactivo, medicina, rendimiento, tiempo real, volumen.

Índice general

Índice de figuras

Índice de tablas

Introducción

Desde la antigüedad surge la necesidad de representar gráficamente objetos complejos, con el objetivo de ilustrar su estructura interna o comunicar su funcionamiento. Usualmente se emplean abstracciones para simplificar el contenido de las imágenes, tal como se muestra en los bocetos anatómicos de Leonardo da Vinci, creados durante el Renacimiento, ver Figura 1. Las imágenes creadas con estos propósitos se conocen como ilustraciones.

Figura 1. Boceto dibujado por Leonardo da Vinci.

La medicina se destaca en el uso de ilustraciones para el estudio de órganos y sistemas internos. En la actualidad, con la evolución de las técnicas de adquisición de datos volumétricos, los especialistas son capaces de obtener información precisa de las estructuras anatómicas internas del organismo humano. Las representaciones gráficas de órganos, huesos, tejidos blandos y torrentes sanguíneos, se obtienen mediante el uso de diferentes técnicas de Visualización de Volúmenes.

La Visualización de Volúmenes consiste en la representación visual de todo el conjunto de datos volumétricos, obtenidos mediante diferentes modalidades de adquisición. Los datos volumétricos son colecciones de valores escalares que definen una propiedad física medible en una región del espacio [1]. Mediante una combinación de técnicas de muestreo, filtrado, clasificación y visualización, se genera una representación gráfica de los objetos contenidos en la región del espacio escaneada, basada en los valores escalares asociados.

La Visualización de Volúmenes se divide en dos grupos: Visualización Directa de Volumen (DVR, por sus siglas en inglés) y Visualización Indirecta de Volumen (IVR, por sus siglas en inglés). Los algoritmos pertenecientes a DVR generan imágenes de un volumen de datos sin extraer superficies geométricas. Ello

se debe a la evaluación de un modelo óptico que define como el volumen emite, refleja, dispersa y ocluye la luz [2]. Por otra parte, los algoritmos de IVR construyen una representación intermedia del volumen, generalmente una geometría que se muestra posteriormente en la pantalla [3].

En el Centro de Entornos Interactivos 3D (VERTEX), de la Universidad de las Ciencias Informáticas (UCI), se desarrolla el proyecto Vismedic, que tiene como objetivo la creación de ilustraciones a partir de datos volumétricos, usualmente escaneados mediante modalidades de adquisición como Tomografía Computarizada (CT, por sus siglas en inglés) e Imágenes por Resonancia Magnética (MRI, por sus siglas en inglés). Dada la complejidad de los algoritmos utilizados, los volúmenes se procesan en la Unidad de Procesamiento Gráfico (GPU, por sus siglas en inglés), con el objetivo de obtener tiempos interactivos. Vismedic cuenta con varios pasos para la generación de las imágenes, siendo estos: **adquisición** de los datos volúmetricos, **filtrado** del volumen, **segmentación** de las regiones de interés, **visualización** tridimensional y **etiquetado** [4]. Actualmente se desarrolla la versión Vismedic - Illustration, con el objetivo de visualizar los volúmenes de forma ilustrativa.

Los algoritmos de visualización ilustrativa de volúmenes implementados en el proyecto Vismedic, se evalúan durante la ejecución del algoritmo Raycasting [5] para todo el volumen. La complejidad del Raycasting depende de las dimensiones de la imagen que se pretende representar en pantalla y del tamaño del volumen. Como resultado, para volúmenes de alta resolución, la evaluación de los filtros en 3D disminuye considerablemente el rendimiento de la representación. A pesar de poseer varias técnicas ilustrativas, implementadas en algoritmos independientes, se dificulta la concatenación de varias de estas técnicas, debido a su elevado costo computacional. Para volúmenes de datos relativamente grandes, los filtros en 3D se consideran un impedimento que disminuye la flexibilidad de los estilos ilustrativos del *software*.

Para la generación de ilustraciones se emplean tradicionalmente filtros basados en Procesamiento de Imágenes Digitales. Estos filtros permiten, entre otras operaciones, resaltar los bordes, reducir el ruido, mejorar el contraste entre las estructuras, enfocar y suavizar la imagen. Los filtros se evalúan en dos dimensiones y se pueden concatenar para obtener resultados más complejos. El procesamiento de las imágenes en 2D solo depende de la resolución de la imagen y de las operaciones que realice el filtro. Sin embargo, en el caso de la visualización de volúmenes, la pérdida de una dimensión puede disminuir la variedad de operaciones que se pueden realizar.

Actualmente, el proyecto Vismedic Illustration no cuenta con técnicas de visualización ilustrativa sobre la imagen generada a partir del algoritmo *Raycasting*. Lo que ocasiona que los filtros 3D implementados impacten en el rendimiento de la visualización y desaprovecha las ventajas del uso de filtros tradicionales de procesamiento de imágenes.

Teniendo en cuenta la situación problémica antes descrita, se define como problema de investigación: **¿Cómo adicionar nuevos estilos ilustrativos al proyecto Vismedic - Illustration sin afectar considerablemente el rendimiento?**.

Para el desarrollo de la presente investigación se identifica como objeto de estudio el **Procesamiento de Imágenes Digitales**. Para dar solución al problema de investigación planteado, se propone como objetivo general: **Implementar un módulo de post-procesamiento de imágenes 2D basado en GPU, que permita adicionar filtros ilustrativos a la visualización del proyecto Vismedic - Illustration, sin afectar considerablemente el rendimiento**. Como campo de acción se define el **Procesamiento de Imágenes Digitales en GPU para Visualización Ilustrativa de Volúmenes**.

Este trabajo tiene como resultados esperados: la definición de interfaces de *plugins* para aplicar filtros de post-procesamiento de imágenes digitales a la visualización obtenida en Vismedic - Illustration, la integración con el proyecto y la creación de interfaces gráficas de usuario que permitan interactuar de forma sencilla con los filtros implementados.

Para dar cumplimiento al objetivo planteado, se definen las siguientes tareas de investigación:

- Elaboración del marco teórico de la investigación a través del estudio del estado del arte referente al procesamiento de imágenes en GPU.
- Selección de los filtros ilustrativos a implementar.
- Definición de las interfaces de plugins para la integración con Vismedic - Illustration.
- Implementación en GPU de los filtros identificados.
- Validación de los filtros implementados mediante pruebas de rendimiento y calidad de imagen.

En el proceso de investigación se tienen en cuanta los métodos de investigación científicas siguientes:

- **Analítico-Sintético:** método teórico utilizado para analizar las características de los filtros.
- **Consulta de fuentes de información:** método empírico utilizado para la consulta de las fuentes bibliográficas durante la investigación.
- **Observación:** método empírico utilizado para apreciar los cambios realizados por los filtros al visor de Vismedic - Illustration.

A continuación se muestra la estructura del presente trabajo, incluyendo una síntesis de los capítulos y secciones fundamentales:

- **Capítulo 1. Fundamentación teórica:** se aborda la fundamentación teórica de la investigación, donde se incluye el estudio del Procesamiento de Imágenes Digitales basado en GPU y la Visualización Ilustrativa de Volúmenes, los filtros identificados, sistemas homólogos. Además, se especifica la metodología a utilizar y se definen las tecnologías utilizadas en el entorno de desarrollo.

- **Capítulo 2. Propuesta de solución:** se muestra un mapa conceptual con los principales términos utilizados y su relación, se describe la propuesta de solución, se describe a Vismedic - Illustration y cómo se le integra el módulo de post-procesamiento. Se describe el módulo y sus clases fundamentales, se definen los requisitos definidos y mencionan los patrones de diseño empleados.
- **Capítulo 3. Evaluación de los resultados:** se incluye el diagrama de componentes, se muestran los resultados de las diferentes pruebas realizadas al módulo desarrollado, desde el punto de vista rendimiento y calidad visual. Todos las pruebas se realizan teniendo en cuenta el entorno de prueba y los juegos de datos establecidos.
- **Anexos:** en esta sección se incluyen las historias de usuario referentes a los requisitos funcionales definidos.

CAPÍTULO 1

Fundamentación teórica

El presente capítulo aborda las bases teóricas para la adición de nuevos estilos ilustrativos basados en el procesamiento de imágenes digitales usando GPU. También se introduce la Visualización Ilustrativa de Volúmenes como área de aplicación para los diferentes estilos ilustrativos que se desean adicionar. Además, se profundiza en las categorías de los filtros y su funcionamiento, así como la necesidad de incorporar al *software* Vismedic - Illustration una etapa de post-procesamiento con diferentes filtros 2D. Otra sección importante de este capítulo es el análisis de sistemas homólogos, para identificar funcionalidades que pueden incluirse en la propuesta de solución para dar cumplimiento al objetivo del presente trabajo. Finalmente, se define la metodología de desarrollo de *software* y el entorno de desarrollo que se utilizará para implementar la solución propuesta.

1.1. Programación en GPU

La GPU se ha convertido en una parte fundamental de los principales sistemas informáticos. Actualmente no es solo un poderoso motor gráfico, sino también un procesador programable masivamente paralelo, con aritmética y ancho de banda de memoria que supera de forma considerable a la Unidad de Procesamiento Central (CPU, por sus siglas en inglés). El rápido aumento tanto en la programabilidad como en la capacidad de la tarjeta de video, ha dotado a la comunidad científica de una herramienta de propósito general para resolver problemas complejos. Este empeño de utilizar la GPU como procesador de propósito general (Computación GPU), ofrece una alternativa a los microprocesadores tradicionales en computadoras de alto rendimiento en sistemas del futuro [6]. El *software* Vismedic - Illustration utiliza la GPU para realizar los complejos cálculos que demanda el algoritmo Raycasting para visualizar en tiempo real las ilustraciones generadas a partir de datos volumétricos. Para programar en la tarjeta de video, están disponibles los módulos *Vertex shader* y *Fragment shader*. En los siguientes epígrafes, se profundiza en el uso de los módulos programables *Vertex shader* y *Fragment shader*.

1.1.1. *Vertex shader*

Los *vertex*[1] *shaders* son el tipo de *shader* 3D más común y se ejecutan una vez por cada vértice representado en la escena. Una de las funciones del *vertex shader* es transformar la posición 3D de cada vértice en el espacio virtual en la coordenada 2D que aparece en la pantalla. Además, los *vertex shaders* posibilitan manipular propiedades tales como coordenadas de posición, color y textura. En consecuencia, este módulo brinda la posibilidad de crear efectos con un alto nivel de realismo que incluyen cambios de posición, movimiento, iluminación y color en una escena con modelos 3D [7].

1.1.2. *Fragment shader*

Los *fragment*[2] *shaders*, también conocidos como *pixel shaders*, calculan el color y otros atributos de cada fragmento. En este contexto, el término *fragment* se asocia a un único píxel. Los tipos más simples de *fragment shaders* generan un píxel de pantalla como valor de color. Por otra parte, es posible diseñar *fragments shaders* más complejos con múltiples entradas y salidas. Los *fragment shaders* inician desde un color base de textura al que es posible aplicarle un valor de iluminación, hacer mapas de relieve, sombras, reflejos especulares, translucidez, entre otros. En los gráficos 3D, usualmente un solo *fragment shader* por sí solo no produce efectos muy complejos, pues solo tiene acceso a la información del fragmento y no se tiene conocimiento sobre la geometría de la escena [8].

Sin embargo, los *fragment shaders* sí tienen conocimiento de las coordenadas de la pantalla en la cual se dibuja, y pueden muestrear los píxeles cercanos si el contenido de la pantalla completa se pasa como una textura al *shader*. Esta técnica permite una amplia variedad de efectos de post-procesamiento bidimensional, como la detección de bordes. También se pueden aplicar en etapas intermedias a cualquier imagen bidimensional, mientras que los *vertex shaders* siempre requieren una escena en 3D. Por ejemplo, un *fragment shader* es el único tipo de *shader* que puede actuar como un post-procesador o filtro para una transmisión de video después de haber sido rasterizado[3] [9].

1.1.3. *Kernel*

En la programación en la GPU, un *kernel* hace referencia a las posiciones de los píxeles que se van a procesar en la vecindad del píxel en cuestión [10]. El kernel se define como una matriz de NxN, donde N siempre es impar. Generalmente se utiliza una matriz de 3x3 como se muestra en la Figura 1.1. Cada valor del *kernel* representa un coeficiente de convolución que sirve para ponderar la contribución del píxel en la vecindad al valor final del píxel. De este modo, la relación entre el *kernel* y la máscara de convolución define

[1] Estructura de datos que describe ciertos atributos, como la posición de un punto en el espacio 2D o 3D, en múltiples puntos de una superficie.

[2] Un *fragment* son los datos necesarios para generar el valor de un *pixel* de una primitiva de dibujo en el *frame buffer*.

[3] Rasterización es la tarea de tomar una imagen descrita en formato de gráficos vectoriales y convertirla en una imagen para su salida en una pantalla de video o impresora, o para almacenarla en un formato de mapa de bits.

una medida de cuánto afectan los píxeles vecinos al píxel central. En la sección 1.2.1 se explican con más detalles los aspectos relacionados con la convolución.

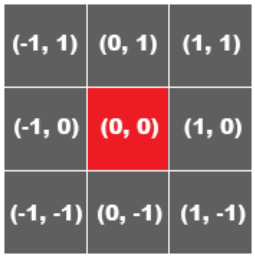


Figura 1.1. *Kernel* de 3x3.

1.2. Procesamiento de imágenes digitales

El procesamiento de imágenes digitales se refiere al procesamiento de imágenes digtales mediante el uso de un ordenador [11]. El procesamiento de imágenes digitales permite aplicar una amplia variedad de algoritmos a los datos de entrada para evitar problemas tales como la acumulación de ruido y la distorsión de la señal durante el procesamiento [12]. Los algoritmos de procesamiento de imágenes digitales pueden aplicarse con diferentes objetivos. Algunos de los más utilizados, debido a sus aplicaciones prácticas son la mejora de la calidad de la imagen, el filtrado, la segmentación, la restauración, la compresión, el procesamiento morfológico y el reconocimiento de patrones [11].

En ocasiones, los algoritmos de procesamiento de imágenes son complejos y demandan una potencia de cálculo que puede ser considerable. Por lo que se suele utilizar la GPU para realizar cálculos en paralelo y lograr como resultado tiempos interactivos. En este caso, cada instancia del algoritmo a nivel de *shader* se ejecuta por cada píxel, restringiendo el conocimiento disponible para el algoritmo a la posición y la textura del píxel.

1.2.1. Convolución

La mecánica de la convolución consiste en aplicar junto con el *kernel*, los coeficientes de convolución en forma de arreglo o matriz, que se conoce con el nombre de **máscara de convolución**. Para obtener el valor final del píxel, se multiplica cada valor del *kernel* por el valor de la máscara de convolución y se suman estos resultados. Este cálculo se realiza para cada píxel de la imagen, y se obtiene como resultado una nueva imagen de salida que contiene información nueva que puede ser utilizada para un propósito en específico [13].

Máscara de convolución

Las máscaras de convolución pueden tomar cualquier valor numérico, sin embargo, cuando se ejecuta el proceso de convolución, el valor final resultante debe hallarse entre 0 y 255 (para una imagen de salida de

8-bits). Para ello, generalmente suelen reemplazarse por 255 los valores mayores a 255 y por 0 los valores menores a 0. Si se considera un *kernel* de 3x3 como la figura 1.1 y una máscara de convolución cuyos nueve coeficientes se muestran en la Tabla 1.1.

a	b	c
d	e	f
g	h	i

Tabla 1.1. Máscara de convolución.

El valor del píxel *O(x, y)* en cuestión se obtiene mediante la fórmula 1.2.4, donde I es la imagen de entrada.

$$Va = a * I(x-1, y+1), Vb = b * I(x, y+1), Vc = c * I(x+1, y+1) \tag{1.2.1}$$

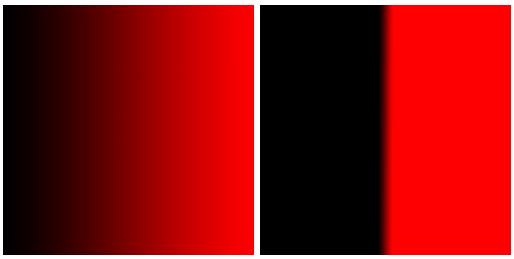
$$Vd = d * I(x-1, y), Ve = e * I(x, y), Vf = f * I(x+1, y) \tag{1.2.2}$$

$$Vg = g * I(x-1, y-1), Vh = h * I(x, y-1), Vi = i * I(x+1, y-1) \tag{1.2.3}$$

$$O(x, y) = Va + Vb + Vc + Vd + Ve + Vf + Vg + Vh + Vi \tag{1.2.4}$$

1.2.2. Gradiente

El gradiente es la base de la mayoría de los filtros de procesamiento de imágenes digitales. Este representa el cambio direccional en la intensidad o el color de una zona establecida mediante el *kernel*, tal y como se muestra en la Figura 1.2. Desde un punto de vista matemático, el gradiente es la primera derivada de la función $f(x, y)$ que representa la imagen. En programas informáticos para la edición de imágenes digitales, el término gradiente de color también se utiliza para obtener una combinación gradual de color que puede considerarse como una gradación uniforme de valores [14]. En el presente trabajo, el operador de gradiente se utilizará principalmente para la detección de bordes.



(a) Gradiente bajo. (b) Gradiente alto.

Figura 1.2. Ejemplos de gradiente.

1.2.3. Filtros de paso bajo

Los filtros de paso bajo tienen como objetivo suavizar los contrastes presentes en una imagen. Como resultado, estos no modifican los píxeles de la imagen con un valor bajo de gradiente. Por el contrario, los píxeles con un alto valor de gradiente se atenúan o reducen completamente en la imagen de salida [15]. Un filtro de paso bajo muy utilizado es aquel cuya máscara de convolución tiene dimensión 3x3 y sus nueve coeficientes son iguales a $1/9$, como se muestra a continuación en la Tabla 1.2.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Tabla 1.2. Ejemplo de una máscara de convolución para filtros de paso bajo.

Esta máscara produce un simple promedio de la intensidad de los píxeles y se conoce como filtro de la media. La suma de sus coeficientes es igual a 1 y todos ellos son positivos. Estas dos características son válidas para todas las máscaras de filtros de paso bajo. Si este filtro se aplica a una región de la imagen donde cada píxel del *kernel* tiene el mismo valor de color o un área de bajo gradiente, el resultado es igual a la imagen de entrada. Esto se corresponde con el hecho de que no existen cambios notables en los valores de intensidad de los píxeles en la región, lo que indica que el valor del gradiente tiene un valor cercano a cero. Por otra parte, al aplicar el operador de gradiente en una región donde el valor de los píxeles cambia rápidamente, es decir, un área de alto valor de gradiente, el resultado será un valor medio entre las intensidades de los píxeles. Esto produce una imagen de salida compuesta por valores medios que varían levemente. Las transiciones de alto valor de gradiente en la imagen de entrada son atenuadas a transiciones de gradiente bajo.

En las imágenes generadas por el proyecto Vismedic - Illustration, el filtro de paso bajo se puede utilizar para atenuar ruidos de la escena mediante un suavizado del volumen. De este modo, aunque se pierden detalles del volumen, es posible resaltar pequeños detalles que pueden ser relevantes para este tipo de *software*.

1.2.4. Filtros de paso alto

Los filtros de paso alto tienen como tarea principal aislar los píxeles que tienen un valor de gradiente alto. Este tipo de filtro tiene un efecto contrario al de paso bajo, debido a que acentúa los píxeles con un alto valor gradiente y conserva los píxeles con un valor de gradiente bajo [15].

Una máscara de paso alto muy común, de dimensión 3x3, es aquella que contiene un 9 en la posición del centro y -1 en las posiciones que lo rodean como se muestra a continuación en la Tabla 1.3.

-1	-1	-1
-1	9	-1
-1	-1	-1

Tabla 1.3. Ejemplo de una máscara de convolución para filtros de paso alto.

En esta máscara, la suma de los coeficientes es uno y los coeficientes de menor valor rodean el píxel central con el mayor valor positivo. Esta distribución indica que el píxel central del *kernel* tiene una alta influencia en el cálculo del valor final, mientras que los píxeles que lo rodean disminuyen el valor de intensidad. Cuando el píxel central posee un valor de intensidad muy diferente al de sus vecinos inmediatos, entonces el efecto de estos últimos es despreciable y el valor de salida es una versión acentuada del valor original del píxel. Esa gran diferencia indica una marcada transición en los niveles, lo que muestra la presencia de píxeles con alto valor de gradiente. Por consiguiente, en la imagen de salida se espera que la transición aparezca acentuada. Por el contrario, si los valores intensidad de los píxeles vecinos son suficientemente grandes para contrarrestar el peso del píxel central, entonces el resultado final se basa más en un promedio de los píxeles involucrados.

Si los valores de intensidad de todos los píxeles de un *kernel* de 3x3 son iguales, el resultado es simplemente el mismo valor. Es decir, este produce la misma respuesta que el filtro de paso bajo aplicado sobre regiones constantes. Esto significa que el filtro de paso alto no atenúa los píxeles de bajo gradiente. Más precisamente, este enfatiza los píxeles de alto gradiente mientras preserva aquellos con un valor bajo de gradiente.

Los filtros de paso alto permiten destacar los detalles que pertenecen a la frontera de los objetos presentes en una imagen, independientemente de su orientación. Los filtros de paso bajo y paso alto constituyen la base de la mayor parte de las operaciones de filtrado [14].

1.3. Visualización ilustrativa de volúmenes

La disciplina Visualización consiste en la representación visual de información procedente de varios tipos de datos, no necesariamente gráficos [2]. La Visualización de Volúmenes, en particular, la integran los métodos para la extracción, clasificación y visualización de la información contenida en conjuntos de datos con características volumétricas, o como también se le conoce en la bibliografía: volúmenes, *datasets*, modelos o simplemente datos (cuando no sea ambiguo). Los datos volumétricos son colecciones de valores escalares que definen una propiedad física medible en una región del espacio [1], por ejemplo la densidad. Empleando una combinación de técnicas de muestreo, filtrado, clasificación y visualización, se genera una representación gráfica de los objetos contenidos en dicha región del espacio, basada en los valores escalares

asociados.

La visualización de volúmenes emplea tradicionalmente una de dos aproximaciones fundamentales [16]. La primera consiste en una simulación físicamente precisa de fenómenos tales como la iluminación y atenuación de un rayo de luz que atraviesa un volumen gaseoso o la atenuación de los Rayos X a través del tejido. La primera variante produce los resultados más realistas y correctos comparados con el objeto original, al menos para datos que representan propiedades físicas. La segunda variante, relacionada débilmente con el comportamiento de la luz a través de un volumen, emplea en su lugar una función de transferencia que permite asociar el valor escalar de una muestra con propiedades ópticas como color y opacidad [17]. Esta aproximación posibilita a los usuarios crear un amplio rango de estilos para la visualización de un volumen específico, pero sacrifica la facilidad de interpretación de los modelos basados estrictamente en comportamientos físicos.

La visualización ilustrativa de volúmenes combina, de manera flexible, los beneficios de las dos variantes anteriores: la facilidad de interpretación de los modelos físicos con la flexibilidad de las funciones de transferencia. Por lo tanto, las representaciones creadas con visualización ilustrativa de volúmenes son más efectivas a la hora de mostrar las estructuras internas de los datos volumétricos [16]. Como su nombre lo indica, las técnicas de visualización ilustrativa de volúmenes son ampliamente usadas para generar ilustraciones para presentaciones, libros de texto, artículos científicos y videos educacionales, debido a su mayor capacidad comunicativa.

1.4. Filtros

Para lograr la visualización ilustrativa en el *software* Vismedic - Illustration, es necesario desarrollar diversos filtros 2D que permiten estilizar la imagen generada por el algoritmo *Raycasting*. La selección de los filtros se basa en tendencias actuales en el área del procesamiento de imágenes digitales. Un aspecto importante para seleccionar un filtro es su aplicación para el usuario final y las posibles combinaciones que se pueden realizar con dicho filtro para obtener resultados satisfactorios. En la Figura 1.3 se muestra un grupo de filtros 2D y sus respectivos ejemplos generados utilizando la herramienta de diseño digital Photoshop[4]. Estos filtros se explicarán con mayor profundidad en las siguientes secciones.

[4]Programa profesional de procesamiento de imágenes digitales desarrollada por la empresa Adobe Systems Incorporated.

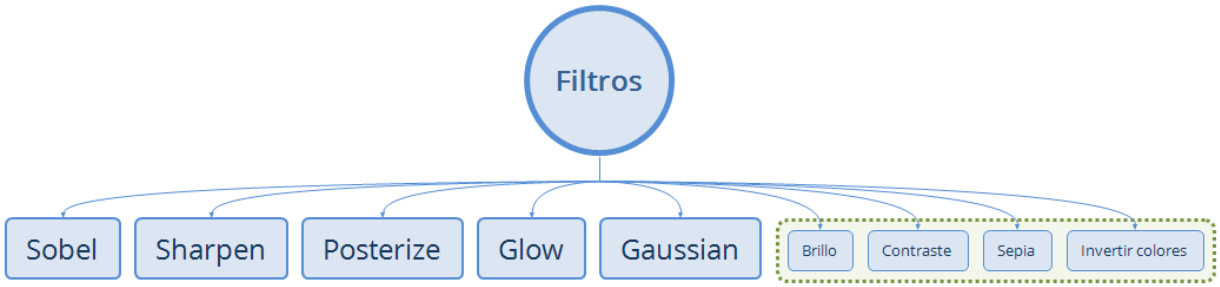


Figura 1.3. Ejemplos de filtros 2D.

1.4.1. Filtro Gaussiano

En procesamiento de imágenes, el desenfoque Gaussiano (Figura 1.4) es el resultado de desenfocar una imagen mediante una función gaussiana como la que se muestra en la ecuación 1.4.1. Este filtro es ampliamente utilizado por programas informáticos con el objetivo de reducir ruido y detalles de la imagen que se consideran irrelevantes. El efecto visual que produce esta técnica de desenfoque es una borrosidad suave que se asemeja a la de ver la imagen a través de una pantalla translúcida. El suavizado Gaussiano también se aplica como etapa de pre-procesamiento en algoritmos de visión por computadora, como una alternativa para mejorar la estructura de la imagen a diferentes escalas [18].

(a) Imagen original. (b) Desenfoque Gaussiano.

Figura 1.4. Desenfoque Gaussiano.

La base del filtro Gaussiano es la función que define el nivel o grado de cambio de los píxeles vecinos sobre el píxel en cuestión, como se puede observar en la Figura 1.5. Para ello se define una función gaussiana

como la de la ecuación 1.4.1, donde σ^2 es la varianza y x es la distancia del píxel respecto al píxel central del *kernel*.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \tag{1.4.1}$$

Figura 1.5. Resultado de la función aplicada a un píxel.

1.4.2. Filtro de Sobel

El operador Sobel (Figura 1.6), a veces llamado operador Sobel-Feldman o filtro Sobel, se utiliza particularmente como algoritmos de detección de bordes, donde se crea una imagen que los enfatiza. Este filtro debe su nombre a sus creadores, Irwin Sobel y Gary Feldman. Técnicamente, es un operador de diferenciación discreto que calcula una aproximación del gradiente de la función intensidad de la imagen. En cada píxel de la imagen, el resultado del operador Sobel es el vector gradiente correspondiente o la norma de este vector. Este operador se basa en la convolución (Ver sección 1.2.1) de la imagen con un filtro pequeño, separable y de valor entero en las direcciones horizontal y vertical. Por tanto, es relativamente económico en términos computacionales. Por otro lado, la aproximación del gradiente que produce es relativamente cruda, en particular para las variaciones de altas frecuencias en la imagen [19].

(a) Imagen original. (b) Imagen resultante de aplicar el filtro de Sobel.

Figura 1.6. Ejemplo del filtro de Sobel.

El *kernel* que utiliza el filtro de Sobel se aplica en las direcciones **x** e **y**. En la Tabla 1.4 se muestra el

kernel de Sobel en la dirección **x**.

-1	-2	-1
0	0	0
1	2	1

Tabla 1.4. *Kernel* de Sobel en la dirección **x**.

De forma análoga, existe un *kernel* de Sobel que se aplica en la dirección **y**. Este se muestra en la Tabla 1.5.

-1	0	1
-2	0	2
-1	0	1

Tabla 1.5. *Kernel* de Sobel en la dirección **y**.

El resultado final del filtro de Sobel se obtiene al sumar los valores de las imágenes obtenidas con los dos *kernels* anteriores.

El filtro de Sobel pondera en mayor medida los píxeles de la zona central con un valor de 2 en cada dirección. Es importante destacar que, en todos los casos, la suma de los coeficientes de las máscaras empleadas es cero. Esto es consistente porque el gradiente brinda una medida de la variación de intensidad en la vecindad del píxel. Por esta razón, en zonas donde no hay variaciones de intensidad el valor del gradiente es cero.

1.4.3. Filtro Sharpen

El filtro Sharpen mejora los detalles relativos a la nitidez de la imagen [20]. Este resalta detalles en la imagen y destaca los bordes. En la Figura 1.7 se muestra el resultado de aplicar este filtro a la imagen de Lena, una de las más utilizadas por la comunidad científica de procesamiento de imágenes.

(a) Imagen original. (b) Resultado de aplicar Sharpen a la imagen original.

Figura 1.7. Filtro Sharpen.

En la Tabla 1.6 se puede apreciar un posible *kernel* para el filtro Sharpen. Los valores de $sw = 0.0015625$ y $sh = 0.0027778$ definen el tamaño de la vecindad de píxeles en las direcciones **x** e **y** que se analiza respecto al píxel central.

t_1 (-sw , sh)	t_2 (0 , sh)	t_3 (sw , sh)
t_4 (-sw , 0)	t_5 (0 , 0)	t_6 (sw , 0)
t_7 (-sw , -sh)	t_8 (0 , -sh)	t_9 (sw , -sh)

Tabla 1.6. *Kernel* definido para el filtro Sharpen.

Además, se define una variable **k** para ponderar la intensidad del píxel central. El valor de esta variable lo define el usuario. Para obtener el valor final de cada píxel, se sustrae la suma de los valores de los píxeles adyacentes, del valor del píxel central, como se muestra en la ecuación 1.4.2, donde cada t_i representa el valor del píxel definido en el kernel.

$$f(x, y) = kt_5 - (t_1 + t_2 + t_3 + t_4 + t_6 + t_7 + t_8 + t_9) \tag{1.4.2}$$

1.4.4. Filtro Glow

El filtro Glow se aplica generalmente para resaltar los bordes, pero de una manera que conserva los colores originales en los bordes [21]. Como resultado de aplicar el filtro, este tiende a oscurecer la imagen,

debido a que en la imagen final solo son visibles los colores de los píxeles que tienen un valor alto de gradiente. El filtro Glow utiliza el mismo *kernel* que el filtro Sharpen (Tabla 1.6). La Figura 1.8 muestra el resultado de aplicar el filtro Sharpen a la imagen de Lena.

(a) Imagen original. (b) Resultado de aplicar Glow a la imagen original.

Figura 1.8. Filtro Glow.

El filtro de Glow utiliza las ecuaciones 1.4.3, 1.4.4, 1.4.5 y 1.4.6. El resultado final del píxel procesado se obtiene en la variable p_r.

$$p_x = t_1 + 2t_2 + t_3 - 2t_8 - t_7 - t_9 \tag{1.4.3}$$

$$p_y = t_1 + 2t_4 - t_3 - 2t_6 + t_7 - t_9 \tag{1.4.4}$$

$$p_d = \sqrt{x^2 + y^2} \tag{1.4.5}$$

$$p_r = 2p_d t_5 \tag{1.4.6}$$

1.4.5. Filtro Posterize

La posterización de una imagen implica la conversión de una gradación continua de tono a varias regiones de menor tono, con cambios notable de uno a otro, como se muestra en la Figura 1.9. Este filtro tiene sus orígenes en los procesos fotográficos, con el objetivo de crear carteles. En la actualidad, se utiliza en procesamiento de imágenes digitales [22].

(a) Imagen original. (b) Resultado de aplicar el filtro Posterize.

Figura 1.9. Filtro *Posterize*.

Para aplicar el filtro Posterize, se define el valor p como el color RGB del píxel en cuestión y un valor n que representa el número de colores a limitar. Luego, mediante las formulas 1.4.7, 1.4.8 y 1.4.9. El valor resultante del píxel procesado se obtiene en $p3$.

$$p1 = p * n \tag{1.4.7}$$

$$p2 = floor(p1) \tag{1.4.8}$$

$$p3 = \frac{p2}{n} \tag{1.4.9}$$

1.4.6. Complementarios

Los filtros complementarios dan valor agregado al módulo para lograr ilustraciones más artísticas y elaboradas. Las funciones de los filtros agregados son: controlar el brillo, contraste, tono, convertir la imagen en sepia e invertir los colores. En las Figura 1.10 y 1.11 se muestran los resultados de dos de estos filtros.

(a) Imagen original. (b) Resultado de aplicar el filtro Sepia.

Figura 1.10. Filtro de Sepia.

(a) Imagen original. (b) Resultado del filtro invertir colores.

Figura 1.11. Filtro de invertir colores.

1.5. Sistemas homólogos

Se han identificado varios sistemas de visualización de volúmenes homólogos al proyecto Vismedic. El más relevante y usado a nivel mundial es Osirix DICOM Viewer disponible solamente para el sistema

operativo Mac OS. Osirix es un programa para la visualización de volúmenes con mas de 10 años de desarrollo y certificado para uso profesional [23]. Es un sistema privativo con precio de licencia por encima de los 600.00 USD. A pesar de ser un programa de élite en el campo de la medicina no cuenta con métodos de visualización ilustrativa de volúmenes, posicionando a la versión Vismedic - Illustration y el módulo de post-procesamiento como una opción viable para el desarrollo del país en el campo de la medicina.

1.6. Metodologías de desarrollo de software

La metodología establece las etapas específicas de un proyecto y la forma en que se efectúan. Además, es el conjunto de métodos que se utilizan en una determinada actividad con el fin de formalizarla y optimizarla [24]. Estas se clasifican en dos grandes grupos: las metodologías tradicionales y las ágiles.

1.6.1. AUP-UCI

Dentro de las metodologías ágiles se encuentra el Proceso Unificado Ágil (AUP, por sus siglas en inglés) de Scott Ambler. Esta es una versión simplificada del Proceso Racional Unificado (RUP, por sus siglas en inglés) que describe de una manera simple y fácil la forma de desarrollar aplicaciones de *software* de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP con sus técnicas tradicionales. Al no existir una metodología de *software* universal, las metodologías se adaptan a las características de cada proyecto (equipo de desarrollo, recursos, etc), exigiéndose así que el proceso sea configurable. En la la UCI se utiliza la metodología AUP-UCI. Esta es una variación de la metodología AUP, de forma tal que se adapta al ciclo de vida definido para la actividad productiva de la universidad. Con esta adaptación de AUP, se logra estandarizar el proceso de desarrollo de *software*, dando cumplimiento además a las buenas prácticas que define el estándar internacional de calidad Capability Maturity Model Integration for Development (CMMI-DEV). En esta versión se definen tres fases de desarrollo: Inicio, Ejecución y Cierre. Además, se proponen 11 roles en lugar de los 9 definidos por AUP. Teniendo en cuenta las características antes mencionadas, la necesidad de una metodología que responda con facilidad a los cambios continuos y siguiendo las políticas de desarrollo de *software* de la institución, se define como metodología a emplear AUP-UCI en el escenario 4.

1.7. Entorno de desarrollo

El entorno de desarrollo se define como el conjunto de sistemas, programas y herramientas necesarias para el desarrollo del *software*. De manera general, es la interrelación o interacción entre varios sistemas que dan soporte al desarrollo y facilita el trabajo en equipo, brindando mecanismos estandarizados [25].

1.7.1. Lenguaje de modelado

Para elaborar el marco contextual del módulo se utiliza el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés). Este posibilita especificar, visualizar y generar artefactos de sistemas de *software*. Como lenguaje de UML, es un método de notación destinado a los sistemas de modelado que utilizan conceptos orientados a objetos [26]. La modelación de UML se realiza empleando herramientas de Ingeniería de Software Asistida por Computadoras (CASE, por sus siglas en inglés).

Las herramientas CASE son aplicaciones informáticas destinados a aumentar la productividad en el desarrollo de *software*, reduciendo el tiempo y costo. Estas herramientas brindan al ingeniero la posibilidad de automatizar actividades manuales y mejorar su visión general de la ingeniería. Al igual que las herramientas de ingeniería y de diseño asistidos por computadora que utilizan los ingenieros de otras disciplinas, las herramientas CASE ayudan a garantizar la calidad antes de desarrollar el producto [27].

Visual Paradigm

Entre la diversidad de herramientas CASE que se utilizan en la UCI para automatizar el desarrollo de un *software*, se encuentra Visual Paradigm for UML (VP-UML). Esta herramienta CASE ofrece un conjunto de ayudas para el desarrollo de programas informáticos, que van desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. Además, es capaz de soportar el modelado mediante UML y proporciona asistencia a los analistas, ingenieros de *software* y desarrolladores, durante todos los pasos del ciclo de vida del *software* [28].

VP-UML fue seleccionada para este trabajo debido a su capacidad para satisfacer las necesidades de la implementación y la experiencia acumulada por el equipo de desarrollo en el trabajo con la misma. Se selecciona como lenguaje de modelado UML, ya que permite definir los componentes que se utilizarán en todas las fases de desarrollo del *software*.

1.7.2. *Frameworks* y bibliotecas

El proyecto Vismedic - Illustration está basado en el *framework* Qt. Los *frameworks* sirven como base para la programación avanzada de aplicaciones, aportando una serie de funciones o códigos para realizar tareas habituales. El uso de *frameworks* simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Además, proporcionan estructuras, que obligan al desarrollador a crear códigos más legibles y estandarizados [29].

Framework Qt

Qt es un *framework* basado en el lenguaje de programación **C++** para el desarrollo de aplicaciones multiplataforma para escritorio, sistemas embebidos y dispositivos móviles [30]. Qt es compatible con Linux, OS X, Windows, VxWorks, QNX, Android, iOS, Blackberry, Sailfish OS y otros. También contiene módulos para el desarrollo de aplicaciones que utilizan redes, bases de datos, OpenGL, tecnologías web, sensores y protocolos de comunicación. Entre las principales características de Qt, se pueden mencionar las siguientes:

- Constituye una biblioteca para la creación de interfaces gráficas.
- Se distribuye bajo la Licencia Pública General (GPL, por sus siglas en inglés), además de la LGPL, que permite su utilización gratuita con fines comerciales.
- Compatibilidad multiplataforma con un solo código fuente.
- Presenta funcionalidades de internacionalización.
- Soporta la arquitectura de *plugins*.

1.7.3. Lenguajes de desarrollo

El desarrollo del módulo utiliza diferentes lenguajes de alto nivel. Entre ellos se encuentra **C++** para tareas ejecutadas en CPU y GLSL para tareas ejecutadas en GPU.

C++

C++ es un lenguaje de programación de alto nivel diseñado a mediados de los años 1980 por Bjarne Stroustrup, como extensión del lenguaje de programación C. Abarca tres paradigmas de la programación: estructurada, genérica y orientada a objetos, por lo que se considera un lenguaje híbrido. En la actualidad C++ es usado por millones de programadores y soporta miles de sistemas. Se considera un lenguaje de programación maduro y de gran velocidad de compilación [31].

GLSL

Existen tres lenguajes principales para programar directamente en la GPU: GLSL (OpenGL), HLSL (DirectX), y Cg (Nvidia). Sin importar el tipo de *shader* que se desea crear, se debe usar alguno de estos tres lenguajes. GLSL es el acrónimo para OpenGL Shading Language. También conocido como GLslang, es un lenguaje de código abierto basado en C/C++, que permite modificar directamente el *pipeline* del procesamiento gráfico (por medio de *shaders*).

Algunos de los beneficios de GLSL son:

- Soporte multiplataforma, y compatibilidad en múltiples sistemas operativos, incluyendo Mac OS, Windows, y GNU/Linux.

- Los *shaders* escritos en este lenguaje se pueden utilizar en cualquier tarjeta de video, independientemente del fabricante.
- Es posible generar código optimizado para la arquitectura de una tarjeta gráfica en particular, ya que cada fabricante provee un *driver* que incluye el compilador de GLSL.
- Proporciona algunas funciones predefinidas, a fin de agilizar operaciones comunes con vectores y matrices. Por ejemplo: *dot* (producto punto), *cross* (producto cruz), *normalize* (normaliza un vector).

GLSL es un lenguaje que esta soportado por casi todos los proveedores de tarjetas gráficas, los cuales incluyen en sus *drivers* un compilador para dicho lenguaje, logrando optimizaciones para *hardware* particulares [13].

1.7.4. Entorno de desarrollo integrado

Qt incluye un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) para su utilización llamado **Qt Creator**. Es un IDE multiplataforma programado en C++, JavaScript y QML. Como principales componentes incluye un editor avanzado de código y un depurador visual. Tiene auto-completado inteligente de código y un compilador de C++. Debido a que se utilizará el *framework* Qt, se propone el uso de Qt Creator como IDE de desarrollo.

1.7.5. Sistema de control de versiones

Para la gestión de las versiones del módulo de post-procesamiento se utiliza un sistema de control de versiones. Esta combinación de tecnologías y prácticas permite controlar los cambios realizados en los ficheros del proyecto, en particular en el código fuente y documentación.

Git

Git es un sistema de control de versiones distribuido, diseñado por Linus Torvalds en el año 2005. Surge como alternativa a BitKeeper, un control de versiones privativo que se usaba en ese periodo para el núcleo Linux. Es liberado bajo una licencia GNU GPLv2 y su última versión estable fue publicada a inicios de Abril de 2017. Por sus disímiles ventajas, se ha convertido en uno de los más usados alrededor del mundo, puesto que no depende de acceso a la red o un repositorio central; además, está enfocado en la velocidad, uso práctico y manejo de proyectos grandes [32]. Por esta razón, se decide utilizar Git, debido a las potencialidades que tiene y a las facilidades de uso.

1.8. Consideraciones del capítulo

En este capítulo se definieron los aspectos esenciales relacionados con la fundamentación teórica, como el procesamiento de imágenes digitales y la clasificación de los tipos de filtros. Además se describieron los filtros **Gaussiano**, ***Sobel***, ***Sharpen***, ***Glow***, ***Posterize*** y otros complementarios, y se presentaron los principios

fundamentales de la programación en la GPU para su implementación. Hasta este punto de la investigación se considera que:

- Se implementarán un conjunto de clases que posibilitarán la fácil incorporación de nuevos filtros.
- Se desarrollarán los filtros **Gaussiano**, ***Sobel***, ***Sharpen***, ***Glow*** y ***Posterize***, y de valor agregado los filtros **Brillo**, **Contraste**, **Tono**, **Invertir color** y **Sepia**.
- Todos los filtros serán implementados usando la GPU como unidad de procesamiento para lograr tiempos interactivos.
- Además, el módulo se integrará con el proyecto Vismedic - Illustration, dotando al sistema de una etapa de post-procesamiento con la que actualmente no se cuenta.

CAPÍTULO 2

Propuesta de solución

Partiendo del marco teórico establecido en el capítulo anterior, en este capítulo se describe detalladamente la propuesta de solución, materializada en el Módulo de Post-procesamiento implementado. Se explica la arquitectura del software Vismedic - Illustration, con el objetivo de comprender los detalles para la integración de la solución. Se realiza además el análisis y diseño del módulo y se definen los requisitos funcionales identificados. Adicionalmente, se describen los patrones de diseño utilizados en la implementación.

2.1. Mapa conceptual

Para comprender de manera general los conceptos utilizados y sus relaciones, se elaboró un mapa conceptual que los abarca, como se muestra en la Figura 2.1. Un mapa conceptual es un esquema de ideas que sirve de herramienta para organizar, de manera gráfica y simplificada, conceptos y enunciados a fin de reforzar un conocimiento [33].

2.2. Descripción de la propuesta de solución

Para dar solución al problema planteado se propone la implementación de un módulo de post-procesamiento de imágenes en GPU. De forma resumida, el módulo consta de una interfaz de *plugins* de post-procesamiento y varias implementaciones concretas de algoritmos de filtrado que emplean la GPU. El control de los efectos radica en la clase **EffectManager**, que contiene la lista de efectos y garantiza la ejecución combinada de los efectos de la lista. En la aplicación, el usuario puede agregar filtros dinámicamente a la lista. Los filtros se ejecutan en orden descendente. Tras ejecutarse el efecto n, continúa con el $n + 1$, siendo la entrada del $n + 1$ la salida del n. Al finalizar la ejecución de los filtros de la lista, el resultado del último filtro se muestra en el visor. La ejecución de los efectos se realiza en tiempos de respuesta interactivos, brindando como resultado la imagen procesada, producto de la combinación de todos los filtros activos.

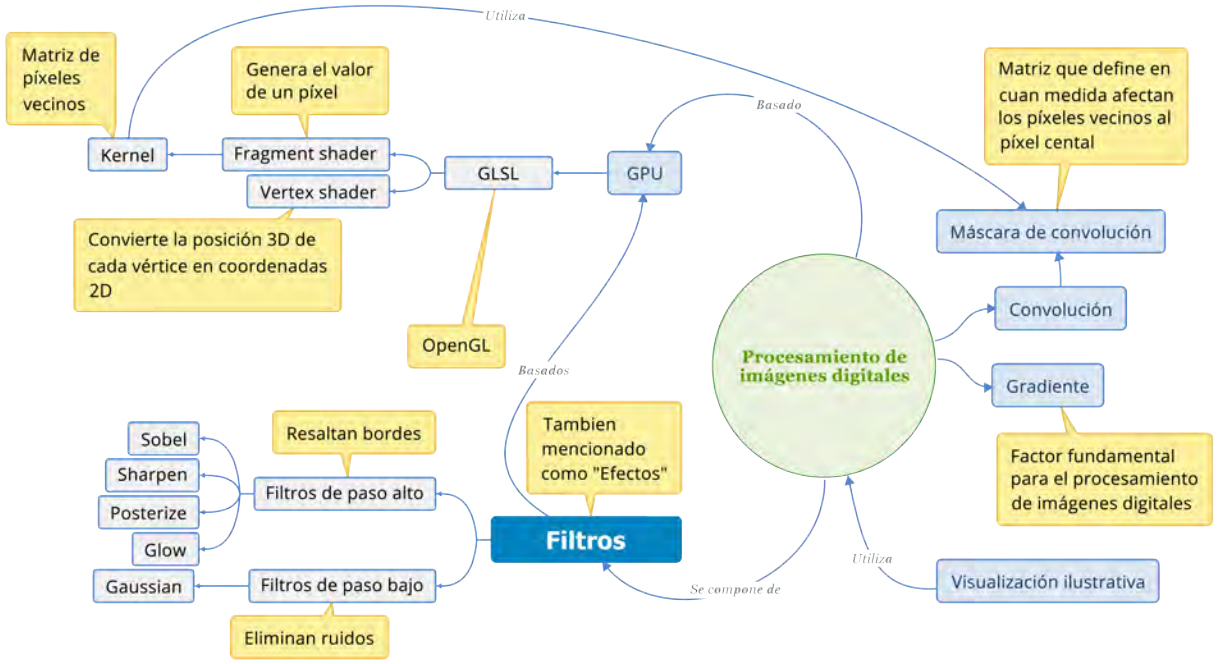


Figura 2.1. Conceptos generales abarcados en la fundamentación teórica

En las siguientes secciones se describen, con mayor nivel de detalle, las características del módulo resumidas en el párrafo anterior, así como los detalles de integración con Vismedic - Illustration.

2.3. Vismedic - Illustration

El módulo se desarrolla para el proyecto Vismedic - Illustration. Como parte de la familia de software de visualización de imágenes médicas registrados Vismedic (Visualizador 2D, Visualizador 3D e Illustration), la versión Illustration se encarga específicamente de facilitar la creación de ilustraciones volumétricas interactivas, a partir de los volúmenes de datos escaneados mediante modalidades como CT y MRI. La Figura 2.2 muestra la interfaz gráfica de Vismedic - Illustration, luego de aplicar un filtro 3D para resaltar las superficies del interior de los volúmenes.

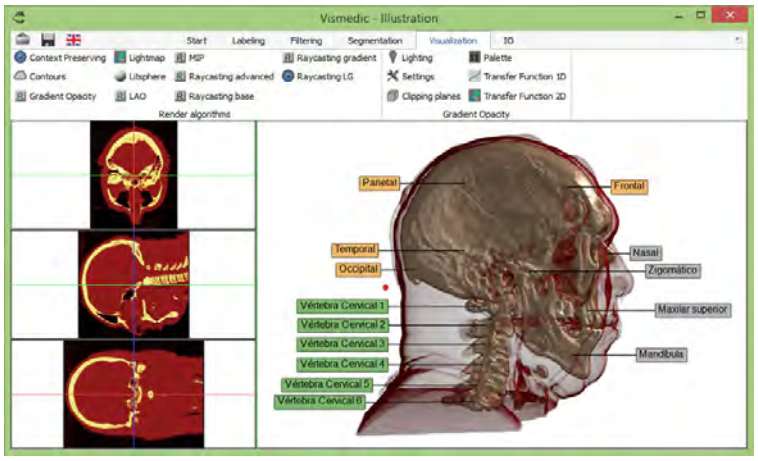


Figura 2.2. Interfaz gráfica de usuario de Vismedic - Illustration.

2.3.1. Arquitectura

La arquitectura general de Vismedic - Illustration se diseñó para soportar de manera natural y ordenada la ejecución de los pasos de Filtrado, Segmentación, Visualización y Etiquetado de ilustraciones, por lo que se crearon módulos independientes que se especializan en realizar cada trabajo específico. En los módulos donde se consideró necesario, se definieron interfaces de *plugins* para extender sus funcionalidades sin modificar el comportamiento general de la aplicación [34]. En la Figura 2.3 se observa un esquema que muestra los componentes fundamentales de la arquitectura.

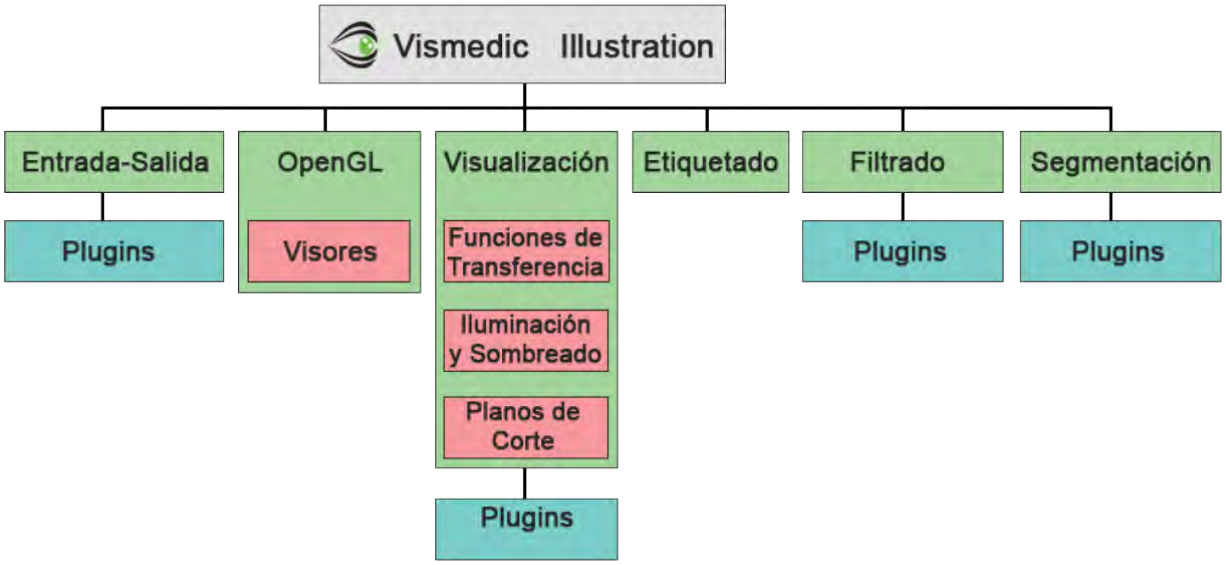


Figura 2.3. Arquitectura de Vismedic - Illustration.

Los objetos globales de la aplicación se comparten en la clase **CoreSettings**, diseñada usando el patrón

Singleton para facilitar su acceso. Vismedic Illustration trabaja con dos volúmenes fundamentales, un volumen Contexto y un volumen Foco. Los algoritmos de visualización que utilicen un solo volumen, siempre usan el Contexto. El volumen Foco se utiliza para diferenciar elementos luego de la segmentación o el etiquetado, puede considerarse también como una selección en determinados escenarios.

Vismedic - Illustration ofrece además un conjunto de funcionalidades útiles para comparar algoritmos de visualización, entre estas se encuentran: calcular los Fotogramas Por Segundos (FPS), guardar la imagen de la visualización actual e intercambiar los algoritmos de visualización, manteniendo el volumen de datos actual y la proyección. La última característica facilita la comparación de dos o más algoritmos de visualización, teniendo en cuenta la calidad de la imagen que generan para los mismos datos de entrada.

2.3.2. Sistema de *plugins*

Como se menciona en la sección anterior, los módulos de Vismedic - Illustration presentan interfaces de *plugins* que permiten extender las funcionalidades de la aplicación. La arquitectura se encarga, en tiempo de ejecución, de crear las interfaces gráficas de los *plugins* concretos y de ubicarlos ordenadamente en la aplicación. Para crear un nuevo *plugin*, solo es necesario un proyecto de Qt de tipo ***QtLib*** y crear una clase que implemente las interfaces de *plugins* definidas en la biblioteca ***core***. Al inicio de la presente investigación, Vismedic - Illustration contaba con las siguientes interfaces de plugins:

- **PluginInterface**: Es la clase interfaz que heredan los demás *plugins*.
- **FilterPluginInterface**: Encargado del filtrado de las imágenes en pre-proceso.
- **IOPluginInterface**: Entrada y salida.
- **RenderAlgorithmPluginInterface**: Encargado del renderizado.
- **SegmentationPluginInterface**: Encargado de la segmentación.
- **PluginManager**: Es el controlador de los demás *plugins*.

Para cumplir con el objetivo de la investigación, se adicionó la interfaz de *plugins* **PostProcessPluginInterface**, para ofrecer un punto de extensión de nuevos algoritmos de post-proceso, que se integren de la misma forma que lo hace el resto de las funcionalidades en Vismedic - Illustration.

2.4. Integración con Vismedic - Illustration

Teniendo en cuenta la arquitectura de Vismedic - Illustration, se crea la clase **EffectManager**, para el control y la gestión del módulo de post-procesamiento. **EffectManager** implementa el patrón *Singleton* para asegurar una única instancia de la clase en la aplicación. Dicha clase, además, contiene la lista de efectos que procesan el volumen del visor de Vismedic - Illustration. Cada efecto hereda de la interfaz **PostProcessAlgorithm**, creada para esta investigación, para estandarizar y generalizar las propiedades y

funciones necesarias en todos los efectos. **PostProcessAlgorithm** cuenta con una lista de la propia clase, es decir, contiene varios algoritmos de post-procesado en su interior. Esto permite flexibilizar el desarrollo de los efectos y a su vez predeterminar conjuntos de algoritmos, para facilitar el uso del *software* al usuario final.

En la Figura 2.4 se muestra el estado de la arquitectura de Vismedic - Illustration luego de la adición del módulo propuesto. Como se puede observar, se adiciona un nuevo paso de post-procesamiento para la creación de ilustraciones.

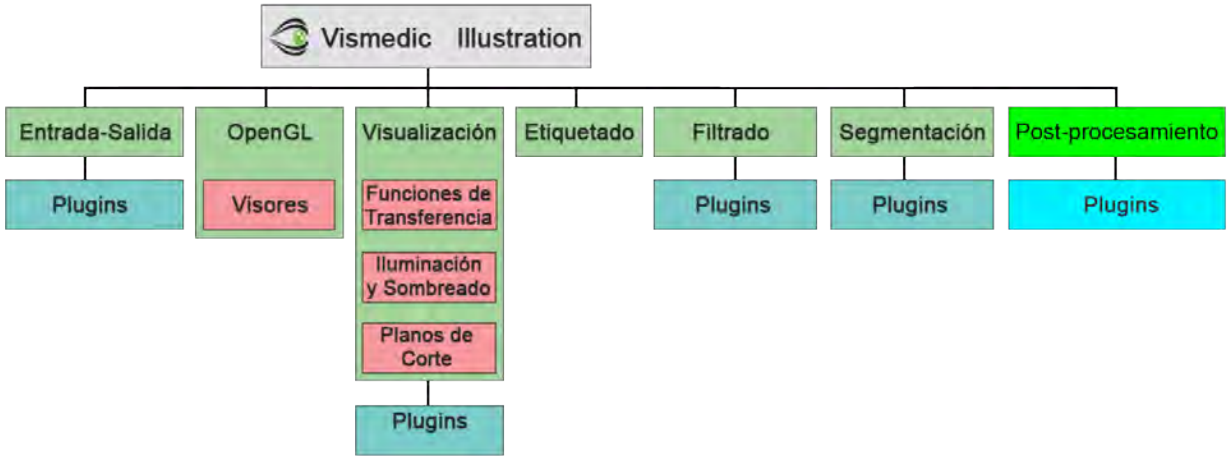


Figura 2.4. Post-procesamiento en la arquitectura de Vismedic Illustration

2.5. Descripción de las clases fundamentales

Las clases del módulo propuesto se pueden dividir lógicamente en cuatro grupos fundamentales. El primer grupo contiene la interfaz de *plugins* **PostProcessPluginInterface**. El segundo grupo, a todas las clases que heredan de **PostProcessPluginInterface**, por lo que representan las implementaciones de los *plugins* concretos. El tercer grupo es contenedor de las clases que heredan de **PostProcessAlgorithm**. Como cuarto y último grupo, se encuentran los algoritmos concretos de post-procesado, es decir, las clases que heredan de **PostProcessAlgorithm** y en este grupo de puede adicionar lógicamente la clase **EffectManager**, que se encarga de manejar los filtros concretos, como se muestra en la figura 2.5.

A continuación se describen detalladamente las clases fundamentales de la solución propuesta. Los elementos marcados entre corchetes ([Name]), significan que son implementaciones concretas a partir de las interfaces de *plugins* y de algoritmos de post-procesado, en este caso, el nombre de la clase solo cambia en función del nombre concreto del *plugin* o filtro, para garantizar la uniformidad del código del módulo y la coherencia con el resto de las clases de Vismedic - Illustration:

- **PostProcessPluginInterface:** clase abstracta que hereda de **PluginInterface** y permite la adición de

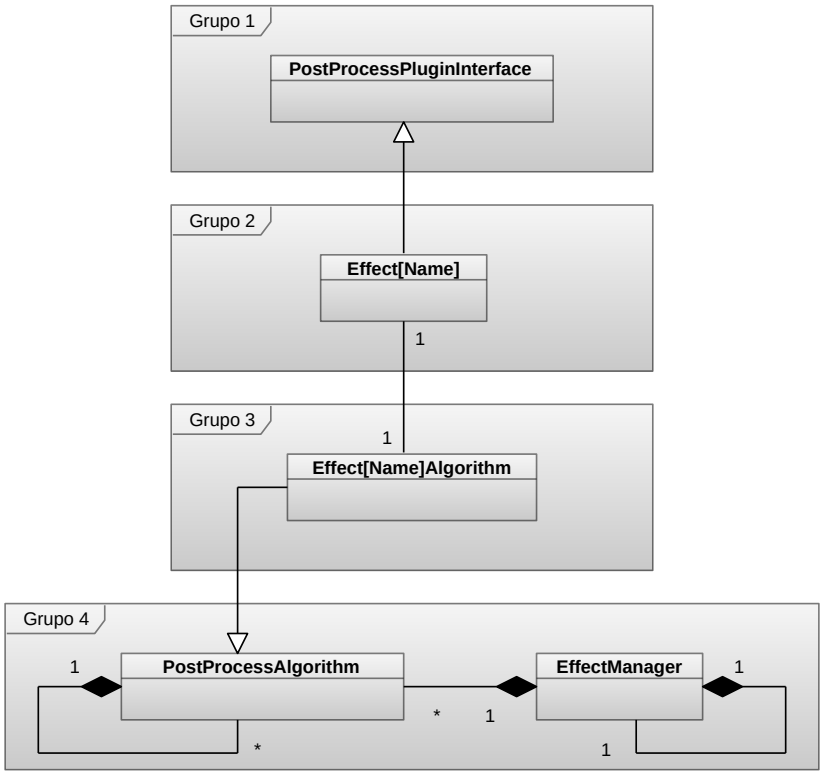


Figura 2.5. Diagrama de clases

plugins de post-procesado, con su correspondiente integración en la *ribbon* de Vismedic - Illustration.

- **Effect[Name]:** clase que hereda de **PostProcessPluginInterface**. Contiene una instancia del algoritmo de post-procesado concreto (**Effect[Name]Algorithm**), referente al efecto en cuestión. Es la clase encargada de establecer el acceso visual al usuario en la **QtRibbon**, mediante la definición de propiedades como *page*, referente a la pestaña; *group*, a un grupo específico dentro de la pestaña y *type*, para el tamaño del icono. Para todos los *plugins* del módulo propuesto se utiliza el texto *Effect* para la propiedad *page*.
- **PostProcessAlgorithm:** clase abstracta que proporciona la base para la implementación de los algoritmos concretos para los efectos. Es padre de las clases **Effect[Name]Algorithm**. Contiene la referencia del *vertex shader* y el *fragment shader* que aplican el efecto en GPU. Además, contiene una lista de **PostProcessAlgorithm** que flexibiliza el desarrollo de los efectos y permite predeterminar varios algoritmos independientes en una única instancia del algoritmo.
- **Effect[Name]Algorithm:** Clase que hereda de **PostProcessAlgorithm** y se encarga de la implementación del efecto concreto.

2.6. Especificación de requisitos del software

La especificación de requisitos de software es el producto del trabajo final que genera la ingeniería de requisitos, donde se describe la función y el desempeño de un sistema basado en computadora, así como las restricciones que regirán su desarrollo [35]. Los requisitos se clasifican en dos grupos: requisitos funcionales y no funcionales. En las siguientes secciones se describen los requisitos funcionales y no funcionales definidos para la solución propuesta.

2.6.1. Requisitos funcionales

Los Requisitos Funcionales (RF) son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que este debe reaccionar a entradas particulares y de cómo se debe comportar ante situaciones específicas. En algunos casos, pueden declarar también explícitamente lo que el sistema no debe hacer [36]. Para la propuesta de solución se utilizaron los siguientes requisitos funcionales:

- **R1 Agregar efecto:** agrega el efecto seleccionado a la lista de efectos que procesa la imagen. Se posiciona como el último filtro de la lista. La acción de agregar a la lista se efectúa con un clic en el botón del *plugin* del efecto en la *QtRibbon*.
- **R2 Eliminar efecto:** elimina el efecto seleccionado de la lista de efectos.
- **R3 Mover efecto:** en la lista es posible mover los efectos hacia arriba o abajo, cambiando el orden de ejecución de los filtros y por tanto el resultado final.
- **R4 Configurar parámetros del efecto:** cada efecto tiene un conjunto de parámetros que el usuario puede variar. La cantidad y los tipos de parámetros dependen del filtro concreto, por lo que las interfaces gráficas de usuario son específicas para cada filtro.
- **R5 Habilitar efecto:** los efectos se puede habilitar o deshabilitar, mediante un clic sobre la celda correspondiente en el listado de filtros. Esto posibilita al usuario ver el resultado de la aplicación o no del efecto y comprobar su influencia en la imagen final.

Aunque los requisitos descritos anteriormente son suficientes para describir el funcionamiento general del módulo, se consideró conveniente adicionar los Requisitos Funcionales relativos a los filtros principales de la propuesta de solución, estos son:

- **R6 Aplicar filtro Gaussian:** aplica un Desenfoque Gaussiano usando la GPU que se encarga de desenfocar la imagen.
- **R7 Aplicar filtro Glow:** detectar los bordes y mantener sus colores, mientras que las áreas de bajo gradiente se oscurecen.
- **R8 Aplicar filtro Sharpen:** enfocar la imagen. Adicionalmente resalta las áreas de alto gradiente.
- **R9 Aplicar filtro Sobel:** aplicar el operador de Sobel para calcular el gradiente. Con el resultado de la magnitud del gradiente se resaltan los bordes de la imagen.

- **R10 Aplicar filtro Posterize:** restringir la cantidad de colores de la imagen. Obtiene como resultado una imagen simplificada con apariencia de póster.

2.6.2. Requisitos no funcionales

Los requisitos no funcionales son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y normas o estándares. Además, no se aplican regularmente a rasgos o servicios individuales, sino al sistema como un todo [36].

- **Hardware:** terminal con disponibilidad de tarjeta gráfica avanzada.
- **Software:** terminal con *drivers* que soporten OpenGL 2.3 o superior.
- **Mantenibilidad:**
    - **Modularidad:** capacidad de un sistema o programa de ordenador, compuesto de componentes discretos, que permite que un cambio en un componente tenga un impacto mínimo en los demás [37]. En la solución propuesta es un requisito indispensable, debido a que los filtros son completamente independientes unos de otros, lo que permite desplegar aplicaciones personalizadas con un número arbitrario de efectos de post-procesado.

2.6.3. Historias de usuario

La historia de usuario es una técnica utilizada para especificar los requisitos del *software*. Se trata de tarjetas de papel donde el cliente describe brevemente las características que el sistema debe poseer. El tratamiento de las Historia de Usuario (HU) se considera dinámico y flexible. Cada una es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas [38]. Todas las HU generadas están disponibles en el Apéndice A de este trabajo.

2.7. Patrones de diseño

Un patrón de diseño describe una estructura que resuelve un problema en particular dentro de un contexto específico y en medio de fuerzas que pueden tener un impacto en la manera en que se aplica y utiliza el patrón [35]. Los patrones de diseño proveen soluciones reutilizables a problemas comunes de diseño e implementación, por lo que su uso puede contribuir a la calidad del código generado. A continuación se describe el uso de los patrones diseño empleados en la propuesta de solución.

2.7.1. Patrones GRASP

Describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones [26]. En el módulo desarrollado se emplearon los siguientes patrones GRASP:

Experto

La responsabilidad de la implementación de un método debe recaer sobre la clase que conoce toda la información necesaria para hacerlo [39], este patrón se evidencia en las clases **PostEffectAlgorithm** y en sus hijas, pues serán las encargadas de conocer cómo se crea cada objeto.

Creador

Ayuda a identificar quién debe ser el responsable de la creación de nuevos objetos [39]. Se evidencia en la clase **EffectManager** pues tiene la información necesaria para la creación de objetos.

Polimorfismo

Se usa cuando varía el tipo de alternativas o comportamientos relacionados; permite asignar la responsabilidad del comportamiento a los tipos en que varía el mismo [39], se evidencia en la clase polimórfica **PostProcessAlgorithm** y en sus hijas, implementando el método **process**, que posee diferente implementación para cada uno de los filtros concretos.

2.7.2. Patrones GOF

Describen soluciones simples y elegantes a problemas específicos en el diseño de software orientado a objetos [40]. El módulo desarrollado emplea los siguientes patrones GOF:

Singleton

Patrón encargado de restringir la creación de instancias de una clase a un solo objeto, de tal manera que la única instancia de la clase solo está contenida en la propia clase. El constructor de la clase es privado y mediante el método **instance()** se accede a la única instancia, como se muestra en la Figura 2.6. La clase **EffectManager** utiliza este patrón, con el objetivo de poder adicionar filtros desde cualquier *plugin*.

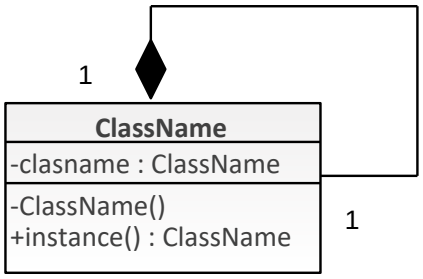


Figura 2.6. Patrón *Singleton*.

Observador

Construye una dependencia entre un sujeto y sus observadores, de modo que cada modificación del sujeto se notifique a los observadores, para que puedan actualizar su estado [40]. Este patrón está presente entre las clases **EffectManager** y **PostProcessAlgorithm**.

2.8. Consideraciones del capítulo

Dentro de los resultados fundamentales de este capítulo se encuentra la definición de las interfaces de *plugins* de post-procesado y de los algoritmos de post-procesado, así como su integración durante la ejecución del proyecto Vismedic - Illustration. Hasta este punto de la investigación se puede concluir que:

- El módulo se integra a Vismedic Illustration como un nuevo paso para la generación de ilustraciones. Los pasos, luego de la integración del módulo, quedan de la siguiente forma: Adquisición, Filtrado, Segmentación, Visualización, **Post-procesamiento**) y Etiquetado.
- Se desarrolló una colección de filtros principales de post-procesamiento (Gaussiano, Glow, Sharpen, Sobel y Posterize) y otra de filtros complementarios (Brillo, Contraste, Invertir y Sepia), donde todos los filtros emplean las interfaces definidas en el módulo.

CAPÍTULO 3

Evaluación de los resultados

En el presente capítulo se evalúan los principales resultados obtenidos en la etapa de pruebas del módulo de post-procesamiento propuesto. Inicialmente se discuten los resultados de aplicar los filtros en el proyecto Vismedic - Illustration. Posteriormente se analizan los parámetros de calidad de imagen y rendimiento de la visualización. Desde el punto de vista de rendimiento, se realizan pruebas extremas, donde se incorporan numerosos filtros, para valorar tiempo de respuesta y la cantidad de FPS. Se describe además el entorno para la realización de las pruebas. Por último, se comparan los resultados obtenidos en Vismedic - Illustration con filtros homólogos generados en la herramienta Adobe Photoshop.

3.1. Clasificación de los filtros implementados

Los filtros implementados, según las funciones que realizan, se dividen en 4 grupos: los encargados del trabajo con bordes, el desenfoque o suavizado, el tratamiento de los colores y para tareas complementarias. Cada grupo aborda filtros específicos que aportan, de forma general, al tratamiento de la imagen final. Para lograr efectos complejos se requiere la concatenación de varios filtros de diferentes grupos. A continuación se relacionan los filtros concretos implementados para cada uno de los grupos lógicos descritos:

Bordes:

- ***Glow:*** realza los bordes y oscurece las zonas de bajo gradiente.
- ***Sharpen:*** aumenta la nitidez de la imagen, en especial de los bordes.
- ***Sobel:*** resalta los bordes en función de la magnitud del gradiente.

Desenfoque o suavizado:

- ***Gaussiano:*** desenfoca de modo homogéneo toda la imagen.

Colores:

- ***Posterize:*** limita la cantidad de colores de la imagen.

Complementarios:

- ***Brillo:*** controlar el brillo.
- ***Contraste:*** controlar el contraste.
- ***Tono:*** controlar el tono.
- ***Sepia:*** homogeneizar la imagen en color sepia.
- ***Invertir:*** invertir los colores de la imagen.

3.2. Diagrama de componentes

El diagrama de componentes muestra las dependencias entre los componentes de un sistema. Los componentes son unidades físicas de implementación, con interfaces bien definidas y pensadas para ser utilizadas como parte reemplazable de un sistema. Los diagramas de componentes pueden mostrar un sistema configurado, con la selección de componentes usados para construirlo o un conjunto de componentes disponibles con sus dependencias [41]. En la Figura 3.1 se muestra el diagrama de componentes del módulo propuesto.

3.3. Evaluación de algoritmos de visualización ilustrativa de volúmenes

La evaluación cuantitativa de la calidad de la imagen generada por algoritmos de DVR tradicionales se considera una tarea complicada, debido a su dependencia del Sistema Visor Humano (SVH) [42]. Frecuentemente, los investigadores ubican los resultados de los filtros uno al lado del otro (*side by side*), utilizando los mismos valores de proyección y dejando al SVH la función de jurado. Otro método popular para comparar algoritmos a nivel de imagen es la diferencia de imágenes [43], donde simplemente se sustrae una imagen de la otra y se muestran solo las diferencias sobre una imagen resultante.

Es bien conocido que el SVH es menos sensible a algunos errores (ruido estocástico) y más sensible a otros (patrones regulares). Sorprendentemente, en ocasiones, el SVH otorga evaluaciones más bajas a algunas imágenes con mayor error numérico que a otras donde el error es menor [42]. Por tanto, las comparaciones visuales se consideran más apropiadas que las numéricas, teniendo en cuenta, además, que las imágenes se producen para observadores humanos. Un modelo que involucre el SVH sería más apropiado que uno puramente numérico, pero, para establecer las comparaciones, se necesitaría la representación real del volumen, lo que limita el uso de funciones de transferencia y las libertades artísticas de los métodos ilustrativos.

Teniendo en cuenta los elementos anteriores, la evaluación de los algoritmos de visualización ilustrativa de volúmenes se hace más compleja. En este caso, se adicionan abstracciones artísticas para simplificar

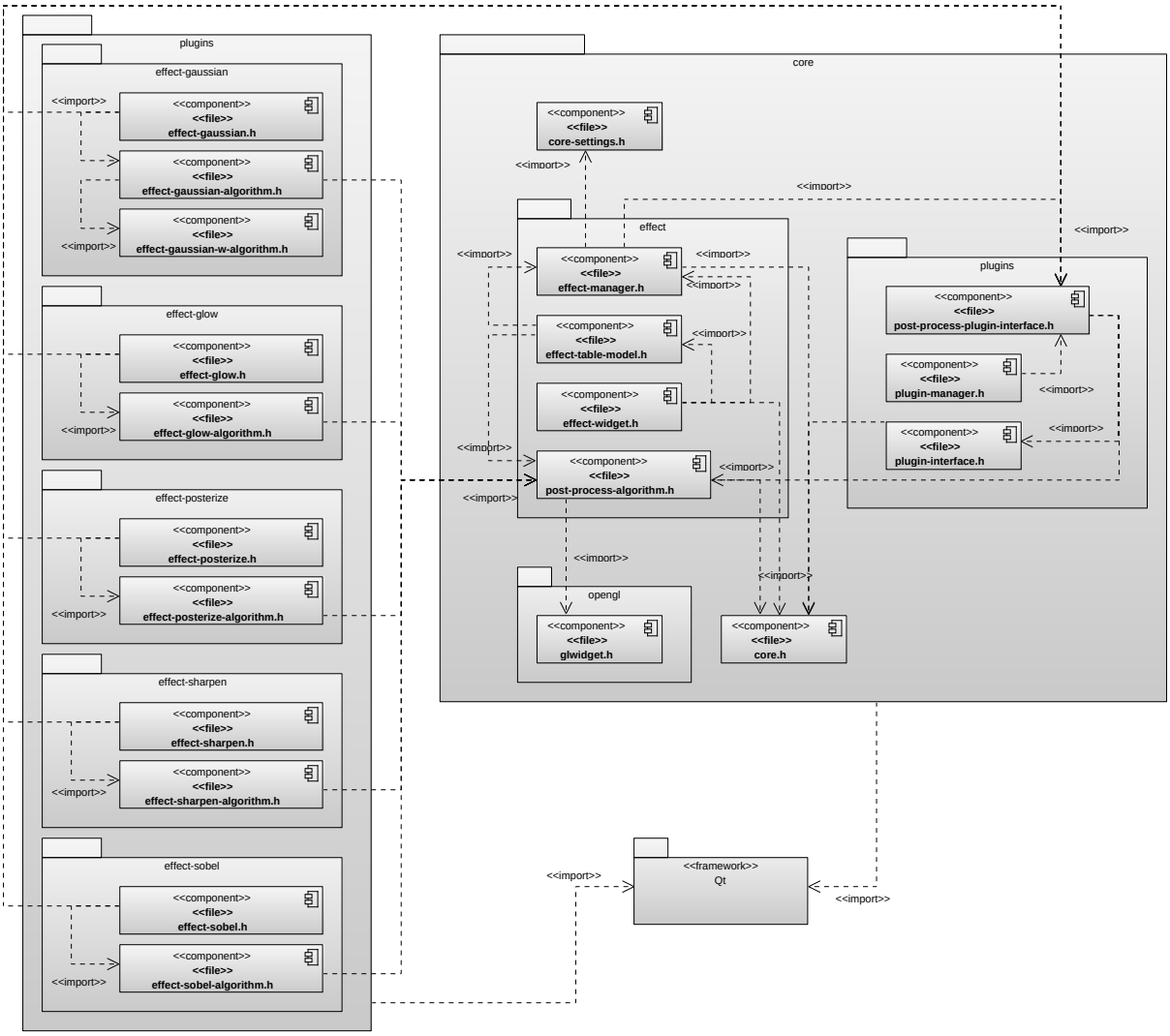


Figura 3.1. Diagrama de componentes

la representación, mejorar la visualización de estructuras relevantes y resaltar las zonas de interés. El estilo ilustrativo depende, además del público objetivo, de la ilustración, la habilidad y conocimiento del ilustrador. Por tanto, desde el punto de vista de calidad de imagen, se decide evaluar los algoritmos de forma visual.

3.4. Entorno de pruebas y juegos de datos

Las pruebas se realizaron en una computadora personal con un procesador Core i5-3317U a 1.70 GHz, 4GB de RAM DDR3 a 1600 Mhz y una tarjeta gráfica Intel HD Graphics 4000 como GPU. En todos los casos se empleó una resolución de 800x600 píxeles para la vista 3D. Es necesario destacar que para todas las pruebas se representaron también las tres vistas 2D con sus respectivos *shaders* y funciones de

transferencia, lo que influye además en el rendimiento. Los juegos de datos son extraídos de repositorios libres para propósitos académicos entre los que se acuentran:

- The volume library: http://lgdv.cs.fau.de/External/vollib/
- Osirix: http://www.osirix-viewer.com/resources/dicom-image-library/
- Vienna University of Technology: https://www.cg.tuwien.ac.at/research/vis/datasets/
- The visible human project: https://www.nlm.nih.gov/research/visible/getting-data.html

3.5. Resultados visuales

Uno de los principales resultados del presente trabajo lo constituye el desarrollo de la colección de filtros para el post-procesamiento de Vismedic - Illustration. El uso de combinaciones de filtros y sus parámetros, para la obtención de resultados complejos, se deja a disposición del usuario. En las siguientes secciones se realiza la discusión, desde el punto de vista de calidad visual, de los principales filtros implementados.

3.5.1. Desenfoque Gaussiano

El desenfoque Gaussiano (***Gaussian*** en inglés) logra homogeneizar el volumen. Ello permite eliminar ruido de la imagen, detalles poco relevantes no necesarios o irrelevantes a la ilustración. A pesar de perder la definición de los bordes, dependiendo del valor del parámetro de intensidad aplicado, se suavizan y redondean. La imagen (c) de la Figura 3.2 se obtuvo con el máximo valor disponible en el filtro, sin embargo, para lograr resultados concretos se utiliza un valor entre 20/100 y 40/100.

(a) Imagen original (b) Desenfoque gaussiano 30/100 (c) Desenfoque gaussiano 100/100

Figura 3.2. Desenfoque Gaussiano aplicado en Vismedic - Illustration

3.5.2. *Glow*

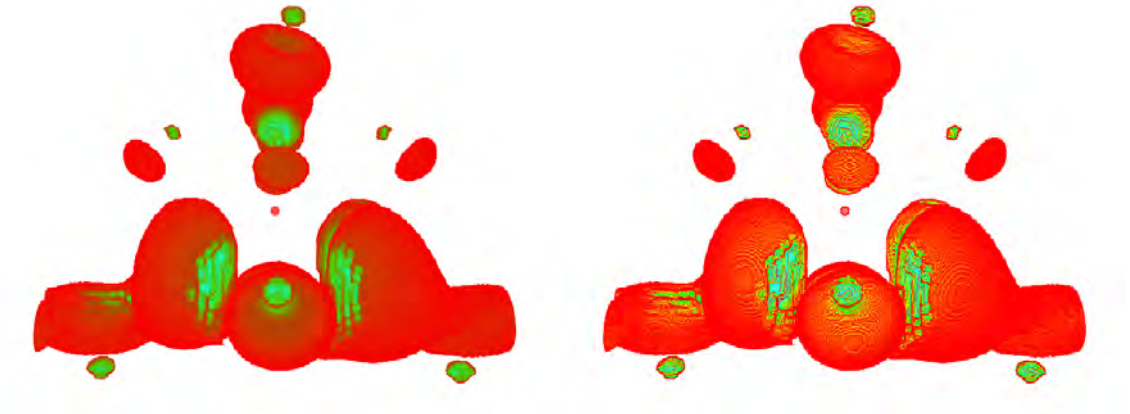
Dentro de los filtros implementados se encuentran los especializados en la detección de bordes. Uno de ellos es el filtros ***Glow*** (ver Figura 3.3) que contiene un parámetro variable de -100 a 100. Los valores negativos significan baja intensidad del efecto. Como se puede observar, el resultados de la aplicación del filtro, hace evidente la presencia de los bordes.

(a) Imagen original (b) *Glow* con valor de 50/100

Figura 3.3. Filtro *Glow* aplicado en Vismedic - Illustration

3.5.3. *Sharpen*

Como resultado evidente en el filtro ***Sharpen***, se destaca la nitidez de los bordes. Contiene un parámetro variable de 10 hasta 50 que representa la intensidad del filtro. Como se puede apreciar en la Figura 3.4, se obtiene un resultado satisfactorio.



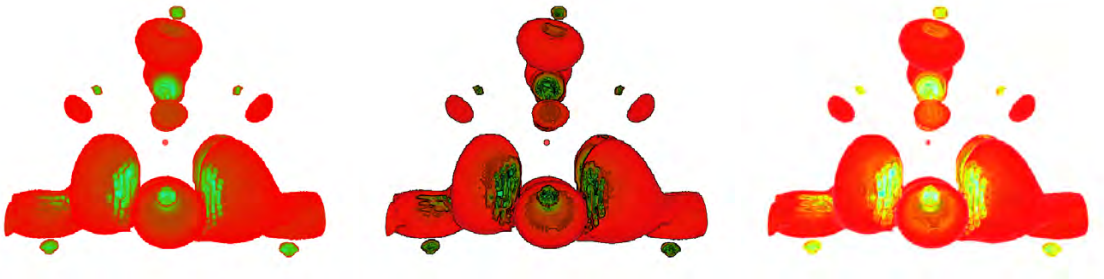
(a) Imagen original (b) *Sharpen* con valor de 20/50

Figura 3.4. Filtro *Sharpen* aplicado en Vismedic - Illustration

3.5.4. *Sobel*

Este filtro utiliza el operador ***Sobel*** para calcular el gradiente de los píxeles de la imagen. En función de la magnitud del gradiente, se resaltan los bordes, lo que proporciona una estética de dibujo a la imagen. El único parámetro del filtro *Sobel* varía desde -200 hasta 100. Donde el valor 0 representa la ausencia de

cambio. Por otra parte, en valores negativos, el borde se aclara y en valores positivos, el borde se oscurece. Los resultados de este efecto se aprecian en la Figura 3.5.

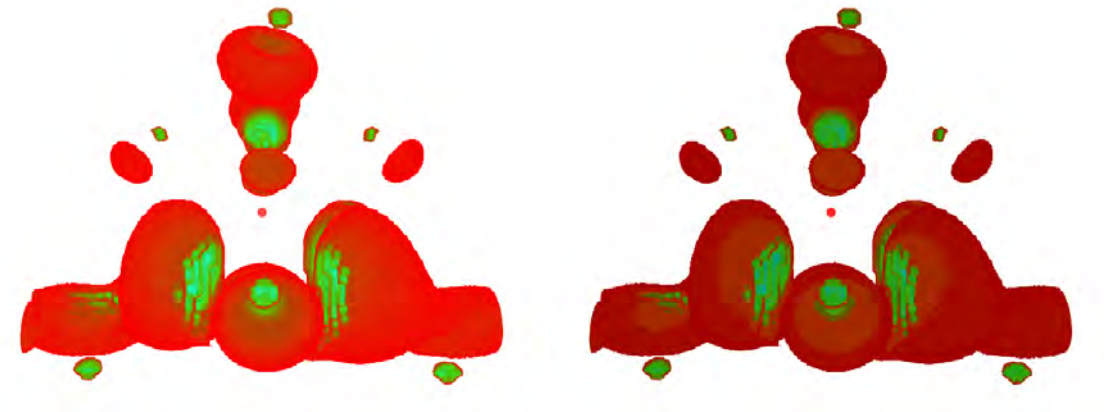


(a) Imagen original (b) *Sobel* con valor de 30/100 (c) *Sobel* con valor de -100/-200

Figura 3.5. Filtro *Sobel* aplicado en Vismedic - Illustration

3.5.5. *Posterize*

Posterizar (***Posterize*** en inglés) se refiere a la acción de limitar el número de colores de la imagen. El filtro desarrollado cuenta con dos parámetros. El primer parámetro controla el número de colores y el segundo define el *gamma*, que controla la difuminación de los colores. El valor de *gamma* puede variar de 50 hasta 100 y el número de colores de 2 hasta 255. La Figura 3.6 muestra los resultados de la aplicación de este filtro.



(a) Imagen original (b) *Posterize* con gamma 50/100 y 5/255 colores

Figura 3.6. Filtro *Posterize* aplicado en Vismedic - Illustration

3.6. Análisis de rendimiento

Teniendo en cuenta el entorno de pruebas definido en la Sección 3.4, se realizaron diferentes pruebas de rendimiento, considerando el número de instancias de cada filtro y la cantidad de FPS resultante. Se usaron

dos variantes fundamentales para las pruebas, una basada en múltiples instancias de un único filtro, Tabla 3.1, y otra empleando combinaciones de diferentes filtros, Tabla 3.2.

Filtro	No. de instancias del filtro	FPS sin filtros	FPS con filtros
Sobel	1	28	28
Sobel	25	28	22
Sobel	50	28	20
Sharpen	1	28	28
Sharpen	25	28	26
Sharpen	50	28	22
Posterize	1	28	28
Posterize	25	28	26
Posterize	50	28	24
Glow	1	28	28
Glow	25	28	24
Glow	50	28	22
Gaussian	1	28	28
Gaussian	5	28	24
Gaussian	10	28	20

Tabla 3.1. Rendimiento de los filtros con instancias de sí mismo

Como se observa en la Tabla 3.1, el filtro más costoso computacionalmente es el desenfoque Gaussiano. Normalmente, un mismo filtro se emplea una sola vez, solo en contadas ocasiones se utilizan varias instancias para obtener resultados más complejos, aunque normalmente no superan las 3 instancias. Para una instancia, el impacto sobre el rendimiento es imperceptible. Para propósitos específicos de pruebas, se usó un número de instancias elevado (25 y 50), con el objetivo de cuantificar los resultados en situaciones extremas. Aun en estas situaciones, el rendimiento se comportó bajo los parámetros esperados.

Por otra parte, la Tabla 3.2, muestra el impacto sobre el rendimiento de la combinación de diferentes filtros, lo que constituye el escenario más probable de uso de los filtros. De igual forma, usualmente se emplean pocos filtros del mismo tipo (normalmente menos de 3). Las cantidades superiores de filtros se emplearon para llevar al límite la aplicación.

No. *Sobel*	No. *Sharpen*	No. *Posterize*	No. *Glow*	No. *Gaussian*	FPS sin filtros	FPS con filtros
1	1	1	1	1	28	26
5	5	5	5	5	28	22
10	10	10	10	10	28	20
25	25	25	25	25	28	14

Tabla 3.2. Rendimiento de los filtros con instancias de varios filtros

Como se observa en la Tabla 3.2, la combinación de filtros en condiciones normales, tiene poco impacto sobre el rendimiento, disminuyendo solo 2 FPS para 5 filtros activos, en una GPU de bajas prestaciones.

Antes de la adición del módulo propuesto, Vismedic - Illustration utilizaba filtros en 3D para obtener estilos ilustrativos. Para establecer una comparación entre las dos clases de filtros, se parte de los valores obtenidos en la Tabla 3.1 y los valores obtenidos con un filtro en 3D. Se seleccionó el filtro *Sobel* entre los 2D y *Opacity* como filtro 3D. En la Tabla 3.3 se muestra la cantidad de FPS obtenidos en la prueba. Aunque es un resultado esperado, se comprueba la diferencia notable de rendimiento a favor del fitro en dos dimensiones.

Filtro	FPS sin filtro	FPS con filtro
2D *Sobel*	28	28
3D *Opacity*	28	11

Tabla 3.3. Rendimiento de los filtros 2D *Sobel* y 3D *Opacity*

3.7. Calidad de la imagen

Como se plantea en la Sección 3.3, la calidad de la imagen de los algoritmos de visualización de volúmenes se evalúa usualmente colocando las representaciones una al lado de la otra (*side by side*), dejando al SVH la función de jurado. La medicina constituye el caso extremo en cuanto a calidad de imagen para los algoritmos de visualización. Es decir, los especialistas médicos toman decisiones, basados en las representaciones del volumen, que influyen en el tratamiento posterior del paciente. Por estas causas, los visores de imágenes médicas se enfocan, generalmente, en obtener representaciones con alta precisión y realistas del volumen. En el caso de Vismedic - Illustration se espera obtener un resultado no realista, puesto que el objetivo es generar ilustraciones que permitan el posterior estudio de órganos y sistemas internos.

Para la comparación, en cuanto a calidad de imagen, se tomó como referencia la herramienta Adobe Photoshop. Algunos filtros no existen explícitamente en Photoshop, por lo que se combinaron varios fil-

tros para obtener resultados similares. A continuación comparan visualmente los resultados de los filtros propuestos.

Gaussian

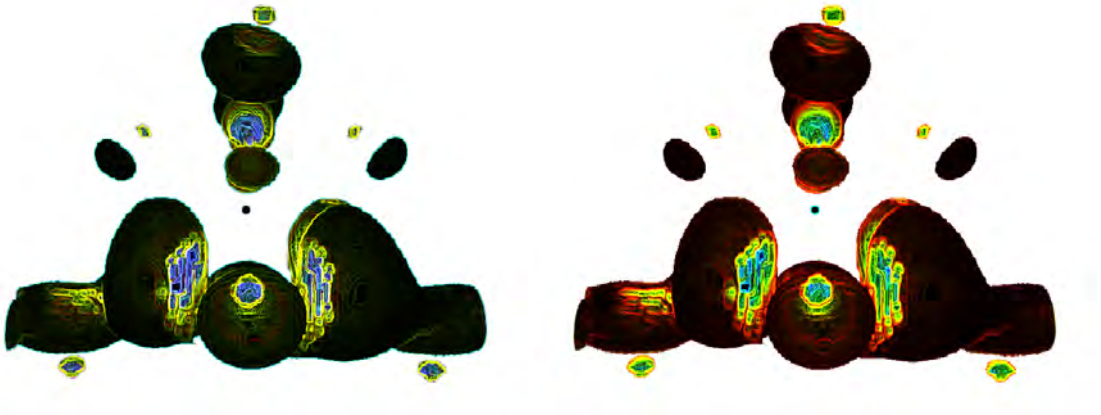
Como se puede observar en la Figura 3.7, la comparación visual del resultado obtenido en Photoshop evidencia la calidad del filtro implementado.

(a) Photoshop (b) Vismedic Illustration

Figura 3.7. Comparación del filtro *Gaussian* respecto a un resultado homólogo obtenido con Photoshop.

Glow

El filtro *Glow* implementado en Vismedic - Illustration no tiene un homólogo en Photoshop. Para lograr tal efecto se realizaron varias operaciones en Photoshop. Los resultados del filtro *Glow* 3.8 de Vismedic - Illustration se valoran como satisfactorios, puesto que logra resaltar los bordes de la imagen.

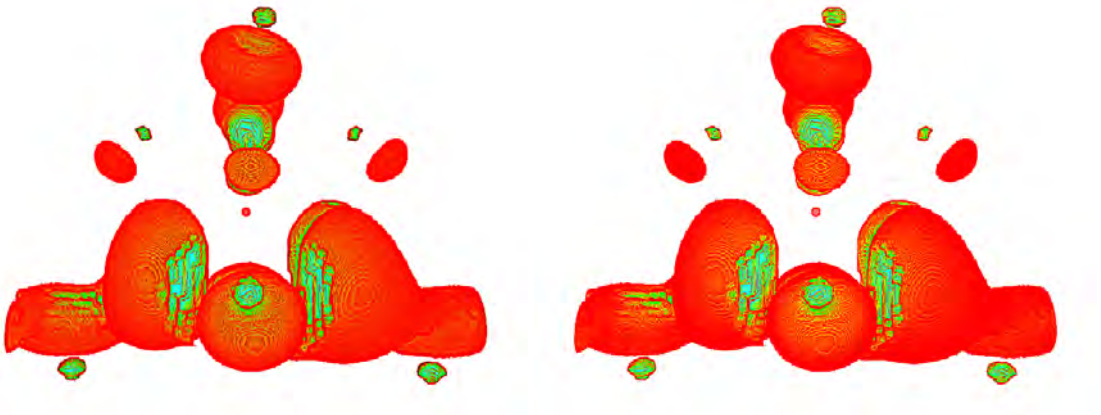


(a) Photoshop (b) Vismedic Illustration

Figura 3.8. Comparación del filtro *Glow* respecto a un resultado homólogo obtenido con Photoshop.

Sharpen

Los resultados obtenidos con el filtro implementado son muy similares los obtenidos con Photoshop, como se observa en la Figura 3.9. El filtro logra enfocar y resaltar los detalles de la imagen. Este filtro es útil para identificar bordes y detalles no visibles en la imagen original.

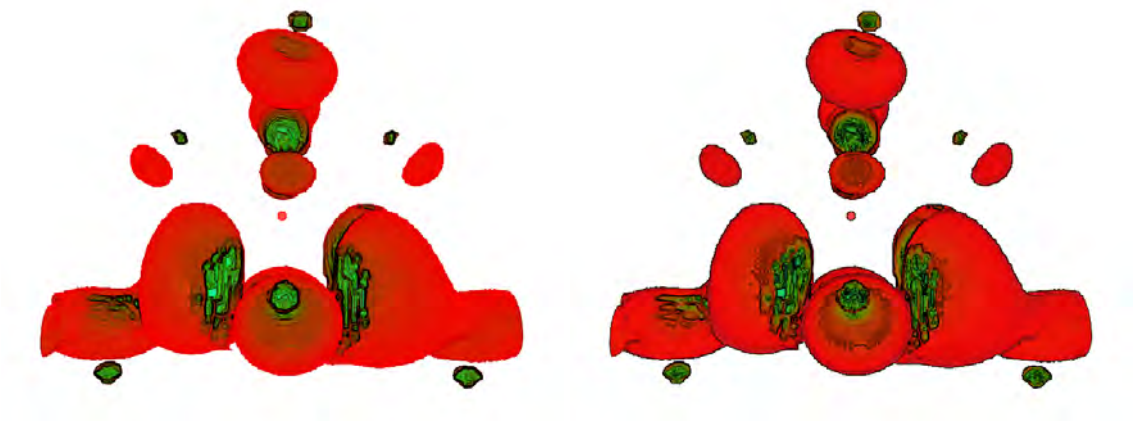


(a) Photoshop (b) Vismedic Illustration

Figura 3.9. Comparación del filtro *Sharpen* respecto a un resultado homólogo obtenido con Photoshop.

Sobel

De los filtros implementados para la detección y realce de bordes, se consideró como el más importante, puesto que proporciona el parecido a un dibujo. Con respecto a Photoshop. el filtro implementado resalta mejor los bordes externos del volumen 3.10. Photoshop tiende a ser más leve y menos marcado.



(a) Photoshop (b) Vismedic Illustration

Figura 3.10. Comparación del filtro *Sobel* respecto a un resultado homólogo obtenido con Photoshop.

Posterize

El filtro *Posterize* implementado cumple con la función de limitar la cantidad de colores de la imagen. A pesar de variar levemente la tonalidad del color al compararse con Photoshop, sí limita la cantidad de colores satisfactoriamente, como se muestra en la Figura 3.11.

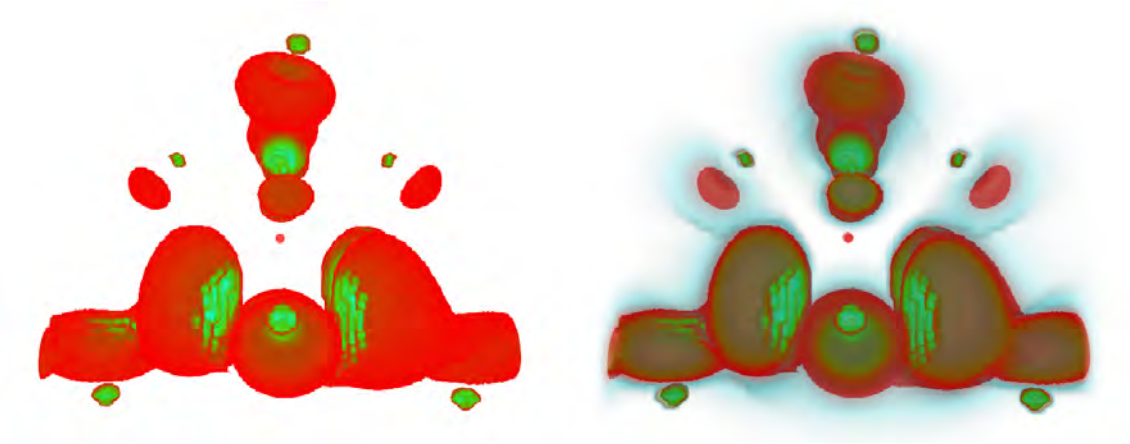
(a) Photoshop (b) Vismedic Illustration

Figura 3.11. Comparación del filtro *Posterize* respecto a un resultado homólogo obtenido con Photoshop.

3.7.1. Filtros 3D

En la Figura 3.12 se observan los resultados de la aplicación del filtro 3D *Opacity*. La aplicación de filtros 3D, al contar con una dimensión más, permiten obtener resultados complejos, pero, como se explicó anteriormente, no permite la concatenación de filtros y disminuye considerablemente el rendimiento. Con

los filtros 2D, como se observa en las imágenes anteriores, se pueden obtener resultados con similar calidad de imagen, sin afectar prácticamente el rendimiento. Los filtros 2D se concatenan de forma fácil, lo que permite obtener resultados prácticos complejos para resaltar bordes, eliminar ruido y realizar tratamientos de colores.



(a) Original (b) Filtros 3D *Opacity*

Figura 3.12. Filtro 3D *Opacity*

3.8. Pruebas de aceptación

Con el objetivo de comprobar si el software se comporta según los requisitos planteados, se realizaron pruebas de aceptación. Las pruebas de aceptación se especifican por el cliente y se centran en las características y funcionalidades generales del sistema que son visibles y se derivan de las historias de usuario. Una prueba de aceptación es similar a una prueba de caja negra. Cada una representa una salida esperada del sistema. Es responsabilidad del cliente verificar la corrección y toma de decisiones acerca de estas pruebas. De las Tablas 3.4 a la 3.9, se muestran las pruebas de aceptación realizadas.

Código: HU1-P1	**Historia de usuario:** 1
Nombre: Agregar un efecto.	
Descripción: Prueba para la funcionalidad de Agregar efecto.	
Condiciones de ejecución: • Debe tener cargado un volumen.	
Pasos de ejecución: • Seleccionar el *tab* **Effect** de la *ribbon*. • Accionar el botón del efecto que desee en la *ribbon*.	
Resultados esperados: • El efecto se agrega correctamente como último en ejecutarse.	

Tabla 3.4. Prueba de aceptación 1

Código: HU2-P1	**Historia de usuario:** 2
Nombre: Eliminar un efecto.	
Descripción: Prueba para la funcionalidad de Eliminar efecto.	
Condiciones de ejecución: • Debe tener al menos un efecto en agregado.	
Pasos de ejecución: • Accionar el botón referente a la acción eliminar de un efecto que está en ejecución.	
Resultados esperados: • El efecto se elimina correctamente.	

Tabla 3.5. Prueba de aceptación 2

Código: HU3-P1	**Historia de usuario:** 3
Nombre: Mover un efecto.	
Descripción: Prueba para la funcionalidad de Mover efecto.	
Condiciones de ejecución: • Debe tener al menos dos efectos en ejecución.	
Pasos de ejecución: • Accionar el botón referente a la acción mover del efecto.	
Resultados esperados: • El efecto se mueve arriba o abajo. • Cambia de posición con un efecto adyacente.	

Tabla 3.6. Prueba de aceptación 3

Código: HU4-P1	**Historia de usuario:** 4
Nombre: Parámetros del efecto.	
Descripción: Prueba para la funcionalidad de Parámetros del efecto.	
Condiciones de ejecución: • El efecto que se desee modificar los parámetros debe tener disponible el botón para modificar el o los parámetros.	
Pasos de ejecución: • Accionar el botón referente a la acción de modificar parámetros del efecto.	
Resultados esperados: • El efecto debe sufrir cambios visibles.	

Tabla 3.7. Prueba de aceptación 4

Código: HU5-P1	**Historia de usuario:** 5
Nombre: Habilitar efecto.	
Descripción: Prueba para la funcionalidad de Habilitar efecto.	
Condiciones de ejecución: • Debe tener al menos un efecto en ejecución.	
Pasos de ejecución: • Accionar el botón referente a la acción de habilitar o inhabilitar.	
Resultados esperados: • El efecto se ejecuta si está habilitado. • El efecto se no ejecuta si está inhabilitado.	

Tabla 3.8. Prueba de aceptación 5

Código: HU6-10-P1	**Historias de usuario:** 6 - 10
Nombre: Efectos.	
Descripción: Prueba para la funcionalidad de los efectos.	
Condiciones de ejecución: • Debe tener cargado un volumen.	
Pasos de ejecución: • La ejecución de los efectos depende del orden en que se fueron agregando.	
Resultados esperados: • Se ejecutan los efectos y se obtiene un volumen ilustrativo interactivo.	

Tabla 3.9. Prueba de aceptación 6

3.9. Consideraciones del capítulo

En el presente capítulo se discutieron los principales resultados obtenidos con el módulo de post-procesamiento implementado. Se analizó el impacto de la concatenación extrema de filtros sobre el rendimiento de la aplicación, demostrando una mejora considerable con respecto a los filtros 3D. Adicionalmente, se comparó la calidad de la imagen final con respecto a filtros similares generados en Photoshop, donde, en todos los casos se comportó de manera aceptable y específicamente para el filtro Sobel, los resultados del módulo propuesto se consideraron superiores. Las pruebas de aceptación demostraron que el módulo integrado a Vismedic - Illustration cumple con los requisitos definidos.

Conclusiones

Con el desarrollo de esta investigación se adicionó un módulo de post-procesamiento en GPU de imágenes digitales al proyecto Vismedic - Illustration, que permite agregar y concatenar filtros para procesar la imagen resultante del algoritmo *Raycasting*. El módulo implementado permite obtener estilos ilustrativos sin afectar considerablemente el rendimiento del *software*, dando cumplimiento al objetivo propuesto al inicio de la investigación. Adicionalmente, se concluye que:

- La definición de interfaces para los *plugins* y para los algoritmos de filtrado, facilitó la posterior adición de efectos ilustrativos. Se adicionaron un total de cinco filtros principales y cinco complementarios, que se pueden combinar para obtener resultados más complejos.
- La aplicación de una única instancia de cualquiera de los filtros implementados no tuvo impacto perceptible sobre el rendimiento de la visualización. Aun para condiciones extremas (50 filtros), el rendimiento solo disminuyó en 8 FPS, lo que demuestra la interactividad del módulo propuesto.
- Los resultados visuales de la aplicación de los filtros implementados se mostraron similares a los obtenidos mediante el *software* Adobe Photoshop, lo que valida la calidad de la imagen generada.

Recomendaciones

Aunque el módulo de post-procesamiento integrado al software Vismedic - Illustration permite obtener estilos ilustrativos complejos, a partir de la combinación de filtros, se pueden añadir una serie de mejoras que contribuyan a ampliar sus posibilidades y a facilitar la interacción con el usuario. Dentro de las principales mejoras a considerar para trabajos futuros se encuentran las siguientes:

- Ofrecer la posibilidad al usuario de predeterminar un conjunto de filtros que se comporten como un único filtro.
- Aumentar la colección de filtros, mediante la adición de nuevos *plugins* de post-procesamiento.
- Agregar el filtro Desenfoque Inteligente, para suavizar regiones homogéneas de la imagen sin afectar los bordes.

APÉNDICE A

Historias de usuarios

Número: 1	**Nombre del requisito:** Agregar efecto
Programador: Karel Díaz Alfonso	**Iteración Asignada:** 1era
Prioridad: Alta	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A

Descripción:

1. **Objetivo :** Permite agregar un efecto a la lista de efecto que se ejecuta el post-procesamiento.
2. **Acciones para lograr el objetivo (precondiciones y datos) :**
   - Debe existir un volumen de datos importado previamente.
3. **Flujo de la acción a realizar :**
   - Para agregar el efecto a la lista de post-procesamiento se le acciona un clic al efecto.
   - El efecto se ubicará como el último efecto a ejecutarse.

Prototipo de interfaz:

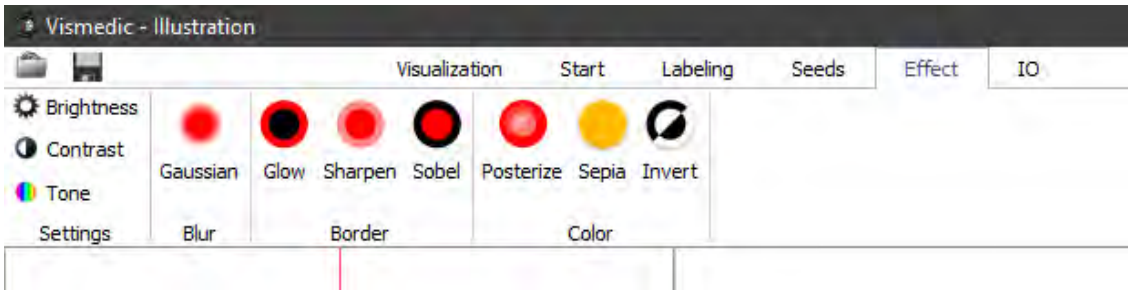


Tabla A.1. Historia de usuario R1

Número: 2	**Nombre del requisito:** Eliminar efecto
Programador: Karel Díaz Alfonso	**Iteración Asignada:** 1era
Prioridad: Alta	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A

Descripción:

1. **Objetivo :** Permitir eliminar un efecto de los que se encuentran en ejecución.
2. **Acciones para lograr el objetivo (precondiciones y datos) :**
   - Debe existir en el sistema al menos un efecto.
3. **Flujo de la acción a realizar :**
   - Presionar el botón correspondiente a la acción de eliminar.

Prototipo de interfaz:

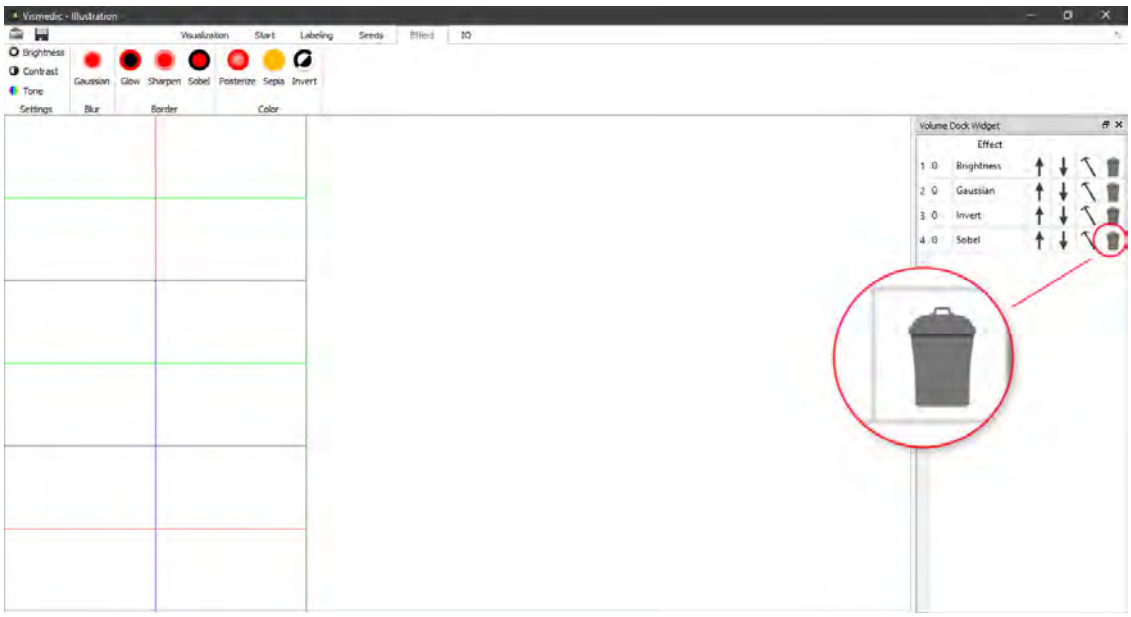


Tabla A.2. Historia de usuario R2

Número: 3	**Nombre del requisito:** Mover efecto
Programador: Karel Díaz Alfonso	**Iteración Asignada:** 1era
Prioridad: Medio	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A

Descripción:

1. **Objetivo :** Permitir mover el efecto arriba o abajo convenientemente.
2. **Acciones para lograr el objetivo (precondiciones y datos) :**
   - Debe existir en el sistema al menos dos efectos en ejecución.
3. **Flujo de la acción a realizar :**
   - Presionar el botón correspondiente a la acción de mover arriba o abajo.

Prototipo de interfaz:

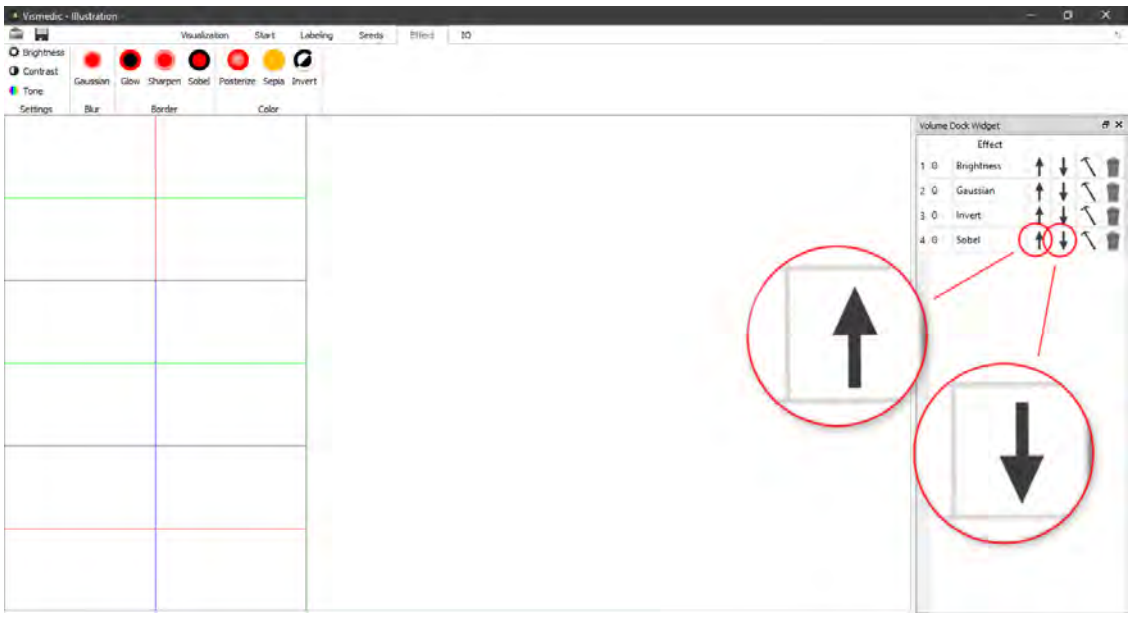


Tabla A.3. Historia de usuario R3

Número: 4	**Nombre del requisito:** Parámetros del efecto
Programador: Karel Díaz Alfonso	**Iteración Asignada:** 1era
Prioridad: Medio	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A

Descripción:

1. **Objetivo :** Permitir modificar los parámetros de un efecto.
2. **Acciones para lograr el objetivo (precondiciones y datos) :**
   - Debe existir en el sistema al menos un efecto en ejecución.
   - El efecto debe contener parámetros modificables.
3. **Flujo de la acción a realizar :**
   - Presionar el botón correspondiente a la acción de modificar parámetros.

Prototipo de interfaz:

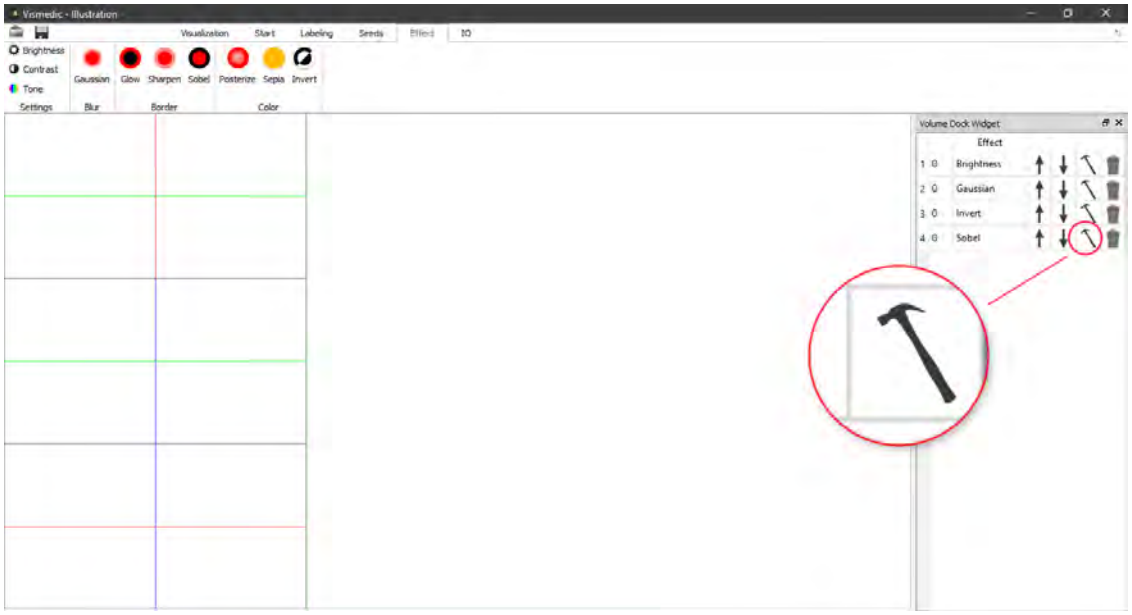


Tabla A.4. Historia de usuario R4

Número: 5	**Nombre del requisito:** Habilitar efecto
Programador: Karel Díaz Alfonso	**Iteración Asignada:** 1era
Prioridad: Medio	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A

Descripción:

1. **Objetivo :** Permitir habilitar o inhabilitar un efecto.
2. **Acciones para lograr el objetivo (precondiciones y datos) :**
   - Debe existir en el sistema al menos un efecto en ejecución.
3. **Flujo de la acción a realizar :**
   - Presionar el botón correspondiente a la acción de habilitar o inhabilitar el efecto.

Prototipo de interfaz:

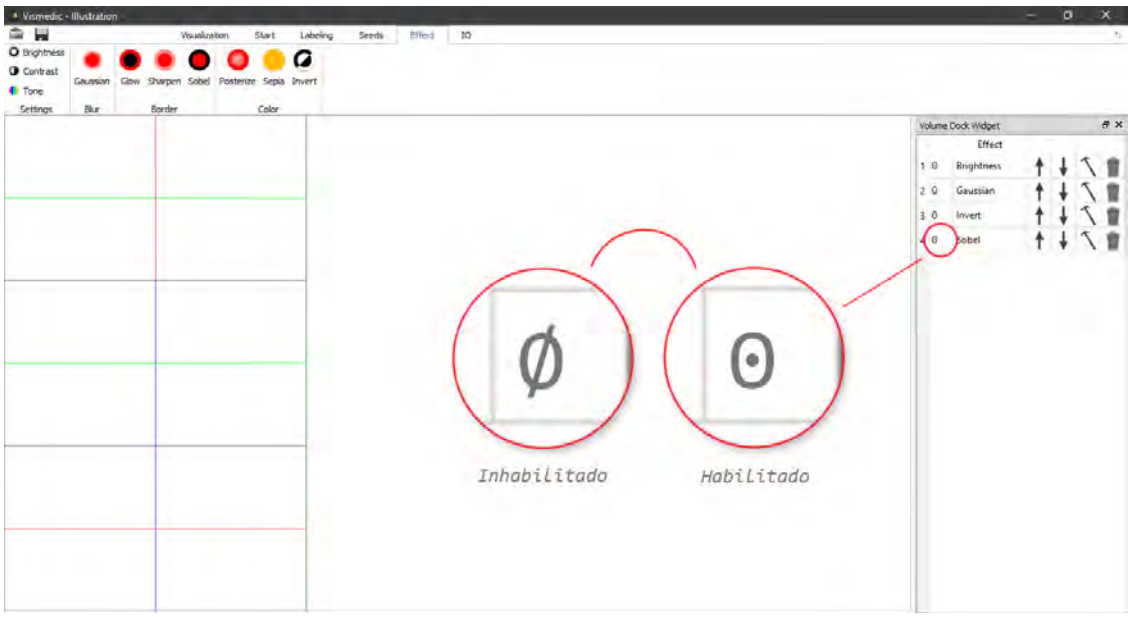


Tabla A.5. Historia de usuario R5

Número: 6	**Nombre del requisito:** *Gaussian*
Programador: Karel Díaz Alfonso	**Iteración Asignada:** 1era
Prioridad: Alta	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A

Descripción:

1. **Objetivo :** Permitir utilizar el efecto *Gaussian*.
2. **Acciones para lograr el objetivo (precondiciones y datos) :**
   - Debe existir un volumen cargado.
3. **Flujo de la acción a realizar :**
   - Presionar el botón correspondiente al efecto *Gaussian*.

Prototipo de interfaz:

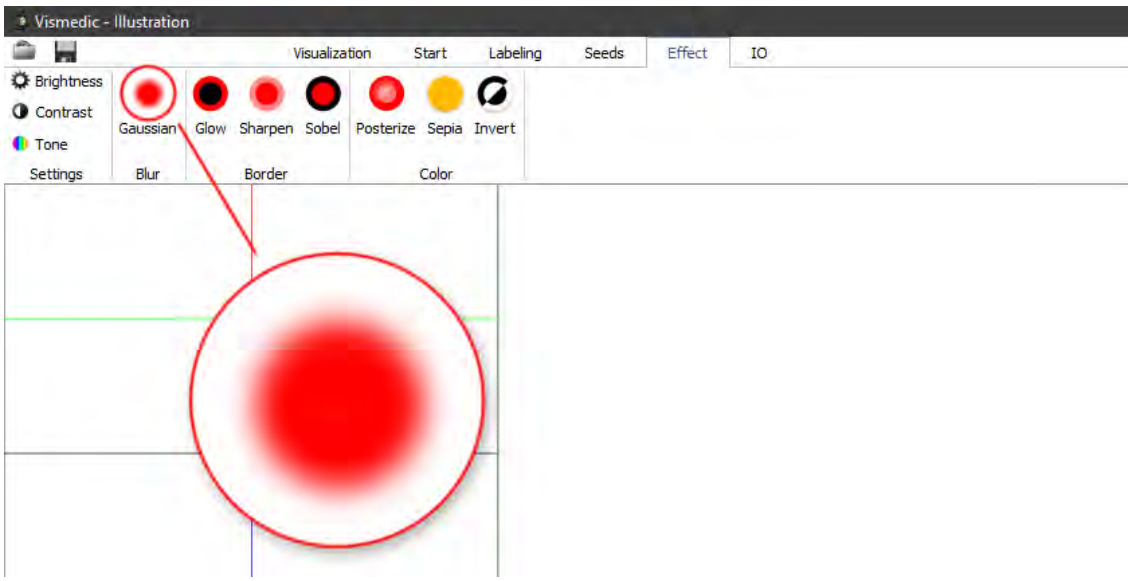


Tabla A.6. Historia de usuario R6

Número: 7	**Nombre del requisito:** *Glow*
Programador: Karel Díaz Alfonso	**Iteración Asignada:** 1era
Prioridad: Alta	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A

Descripción:

1. **Objetivo :** Permitir utilizar el efecto *Glow*.
2. **Acciones para lograr el objetivo (precondiciones y datos) :**
   - Debe existir un volumen cargado.
3. **Flujo de la acción a realizar :**
   - Presionar el botón correspondiente al efecto *Glow*.

Prototipo de interfaz:

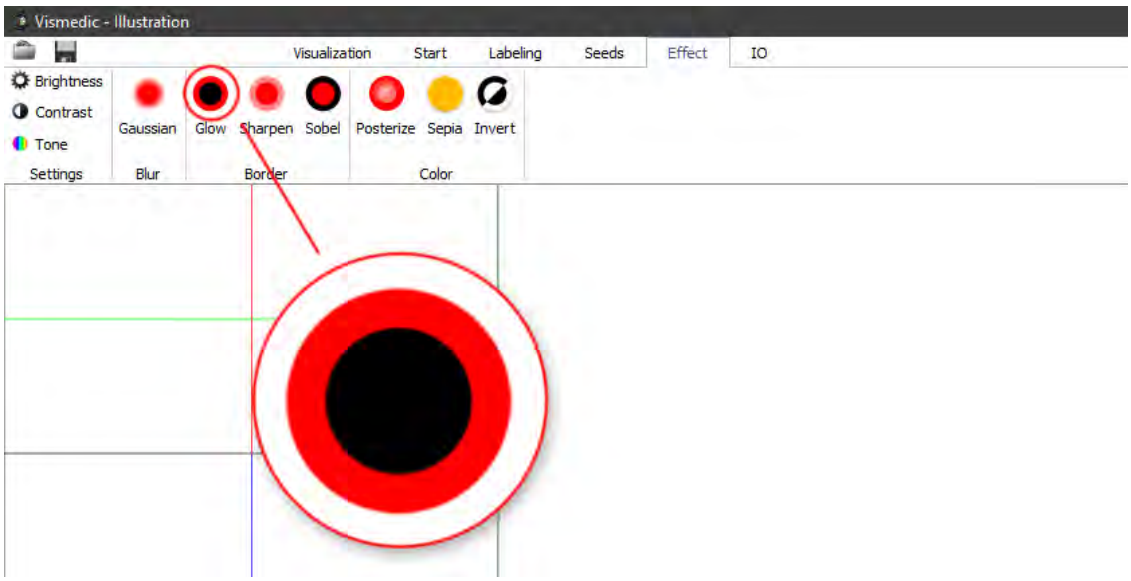


Tabla A.7. Historia de usuario R7

Número: 8	**Nombre del requisito:** *Sharpen*
Programador: Karel Díaz Alfonso	**Iteración Asignada:** 1era
Prioridad: Alta	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A

Descripción:

1. **Objetivo :** Permitir utilizar el efecto *Sharpen*.
2. **Acciones para lograr el objetivo (precondiciones y datos) :**
   - Debe existir un volumen cargado.
3. **Flujo de la acción a realizar :**
   - Presionar el botón correspondiente al efecto *Sharpen*.

Prototipo de interfaz:

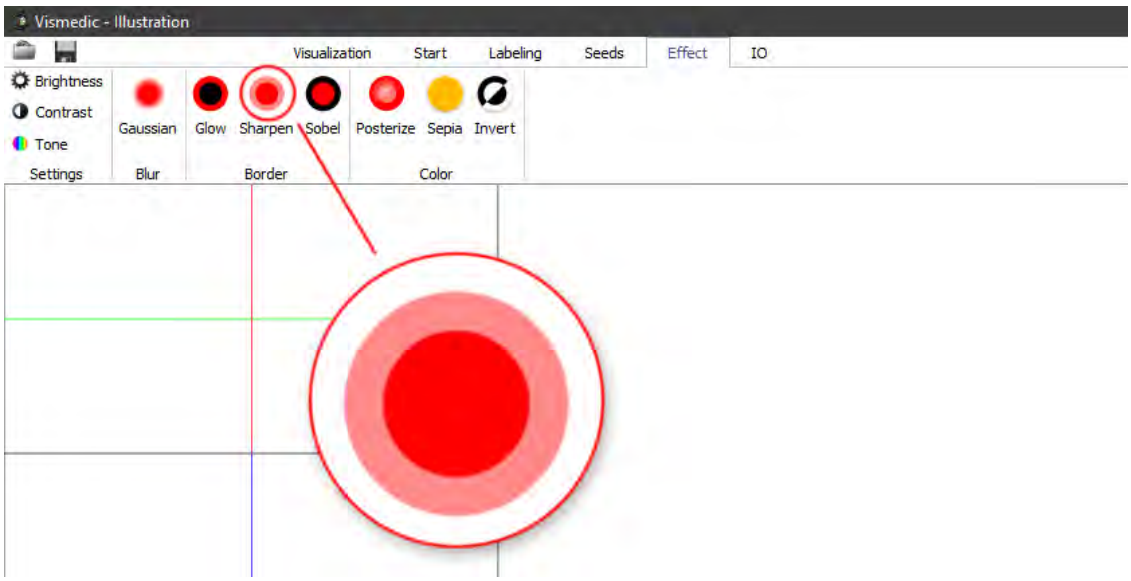


Tabla A.8. Historia de usuario R8

Número: 9	**Nombre del requisito:** *Sobel*
Programador: Karel Díaz Alfonso	**Iteración Asignada:** 1era
Prioridad: Alta	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A

Descripción:

1. **Objetivo :** Permitir utilizar el efecto *Sobel*.
2. **Acciones para lograr el objetivo (precondiciones y datos) :**
   - Debe existir un volumen cargado.
3. **Flujo de la acción a realizar :**
   - Presionar el botón correspondiente al efecto *Sobel*.

Prototipo de interfaz:

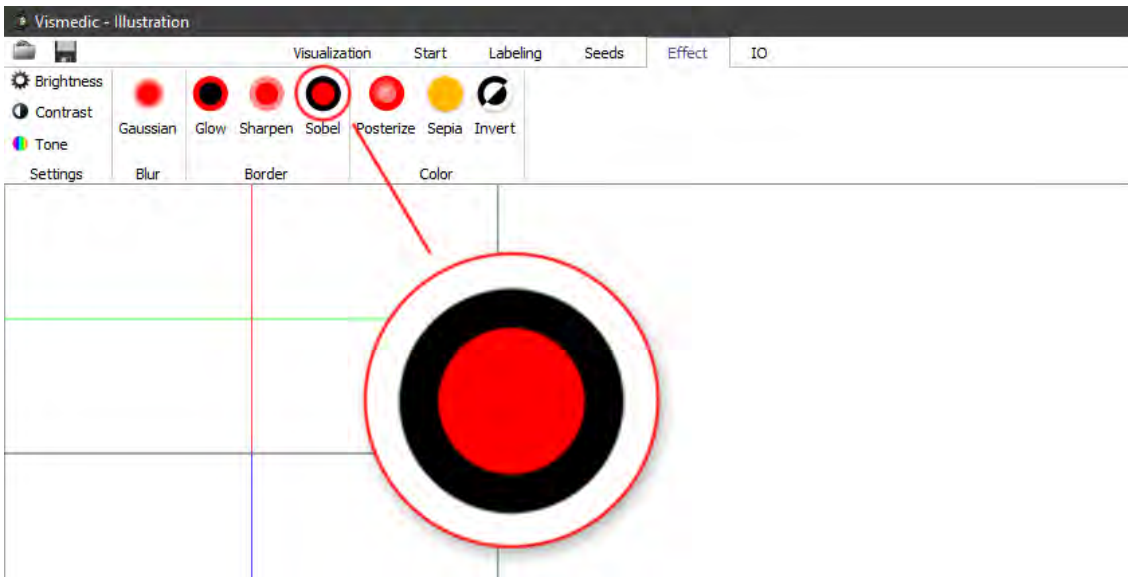


Tabla A.9. Historia de usuario R9

Número: 10	**Nombre del requisito:** *Posterize*
Programador: Karel Díaz Alfonso	**Iteración Asignada:** 1era
Prioridad: Alta	**Tiempo Estimado:** N/A
Riesgo en Desarrollo: N/A	**Tiempo Real:** N/A

Descripción:

1. **Objetivo :** Permitir utilizar el efecto *Posterize*.
2. **Acciones para lograr el objetivo (precondiciones y datos) :**
   - Debe existir un volumen cargado.
3. **Flujo de la acción a realizar :**
   - Presionar el botón correspondiente al efecto *Posterize*.

Prototipo de interfaz:

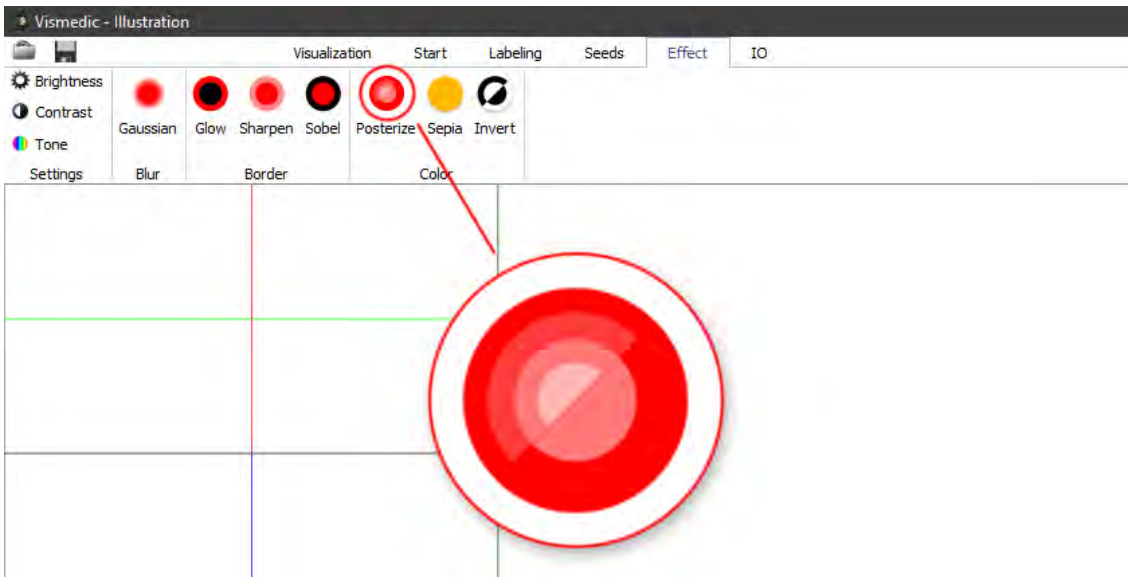


Tabla A.10. Historia de usuario R10

Acrónimos

AUP Proceso Unificado Ágil. 17

CASE Ingeniería de Software Asistida por Computadoras. 17, 18
CMMI-DEV Capability Maturity Model Integration for Development. 17
CPU Unidad de Procesamiento Central. 4, 19
CT Tomografía Computarizada. 2, 23

DVR Visualización Directa de Volumen. 1, 34

FPS Fotogramas Por Segundos. 3, 24, 32, 37, 39, 47

GPL Licencia Pública General. 19
GPU Unidad de Procesamiento Gráfico. 2–4, 6, 19, 21, 23, 48

HU Historia de Usuario. 29

IDE Entorno de Desarrollo Integrado. 20
IVR Visualización Indirecta de Volumen. 1

MRI Imágenes por Resonancia Magnética. 2, 23

RF Requisitos Funcionales. 27
RUP Proceso Racional Unificado. 17

SVH Sistema Visor Humano. 34, 39

UCI Universidad de las Ciencias Informáticas. 2, 17, 18
UML Lenguaje Unificado de Modelado. 17, 18

VP-UML Visual Paradigm for UML. 18

Referencias bibliográficas

[1] Marius Gavrilescu. «Visualization and Graphical Processing of Volume Data». Tesis doct. Gavrilescu, 2011 (vid. págs. 1, 10).

[2] Bernhard Preim y Charl P Botha. *Visual computing for medicine: theory, algorithms, and applications*. Newnes, 2013 (vid. págs. 2, 10).

[3] William E Lorensen y Harvey E Cline. «Marching cubes: A high resolution 3D surface construction algorithm». En: *ACM siggraph computer graphics*. Vol. 21. 4. ACM. 1987, págs. 163-169 (vid. pág. 2).

[4] Luis Guillermo Silva Rojas. «Visualización ilustrativa de datos volumétricos». Tesis de mtría. Universidad de las Ciencias Informáticas, 2017 (vid. pág. 2).

[5] Marc Levoy. «Display of surfaces from volume data». En: *IEEE Computer graphics and Applications* 8.3 (1988), págs. 29-37 (vid. pág. 2).

[6] John D Owens et al., «GPU computing». En: *Proceedings of the IEEE* 96.5 (2008), págs. 879-899 (vid. pág. 5).

[7] Randi J Rost et al., *OpenGL shading language*. Pearson Education, 2009 (vid. pág. 6).

[8] Mike Bailey. «Using gpu shaders for visualization». En: *IEEE Computer Graphics and Applications* 29.5 (2009) (vid. pág. 6).

[9] Tor Dokken, Trond R Hagen y Jon M Hjelmervik. «The GPU as a high performance computational resource». En: *Proceedings of the 21st spring conference on Computer graphics*. ACM. 2005, págs. 21-26 (vid. pág. 6).

[10] Kayvon Fatahalian y Mike Houston. «A closer look at GPUs». En: *Communications of the ACM* 51.10 (2008), págs. 50-57 (vid. pág. 6).

[11] Rafael C Gonzalez. *Digital image processing*. 2016 (vid. pág. 7).

[12] Anil K Jain. *Fundamentals of digital image processing*. Prentice-Hall, Inc., 1989 (vid. pág. 7).

[13] Ricardo Marroquim y André Maximo. «Introduction to GPU Programming with GLSL». En: *Computer Graphics and Image Processing (SIBGRAPI TUTORIALS), 2009 Tutorials of the XXII Brazilian Symposium on*. IEEE. 2009, págs. 3-16 (vid. págs. 7, 22).

[14] B Aldalur y M Santamara. «Realce de imágenes: filtrado espacial». En: *Revista de teledetección* 17 (2002), págs. 31-42 (vid. págs. 8, 10).

[15] Rafael C González y Richard E Woods. *Tratamiento digital de imágenes*. Vol. 3. Addison-Wesley New York, 1996 (vid. pág. 9).

[16] Penny Rheingans y David Ebert. «Volume illustration: Nonphotorealistic rendering of volume models». En: *IEEE Transactions on Visualization and Computer Graphics* 7.3 (2001), págs. 253-264 (vid. pág. 11).

[17] Marc Levoy. «Efficient ray tracing of volume data». En: *ACM Transactions on Graphics (TOG)* 9.3 (1990), págs. 245-261 (vid. pág. 11).

[18] Philip J Stephens y Nobuyuki Harada. «ECD cotton effect approximated by the Gaussian curve and other methods». En: *Chirality* 22.2 (2010), págs. 229-233 (vid. pág. 12).

[19] Nick Kanopoulos, Nagesh Vasanthavada y Robert L Baker. «Design of an image edge detection filter using the Sobel operator». En: *IEEE Journal of solid-state circuits* 23.2 (1988), págs. 358-367 (vid. pág. 13).

[20] Zhi-Feng Xie et al., «A gradient-domain-based edge-preserving sharpen filter». En: *The Visual Computer* 28.12 (2012), págs. 1195-1207 (vid. pág. 14).

[21] Yigal S Horowitz y D Yossian. «Computerised glow curve deconvolution: application to thermoluminescence dosimetry». En: *Radiation Protection Dosimetry* 60.1 (1995), págs. 3-3 (vid. pág. 15).

[22] Kian B Ng, Andrew P Bradley y Ross Cunnington. «Effect of posterized naturalistic stimuli on SSVEP-based BCI». En: *Engineering in Medicine and Biology Society (EMBC), 2013 35th Annual International Conference of the IEEE*. IEEE. 2013, págs. 3105-3108 (vid. pág. 16).

[23] Antoine Rosset, Luca Spadola y Osman Ratib. «OsiriX: an open-source software for navigating in multidimensional DICOM images». En: *Journal of digital imaging* 17.3 (2004), págs. 205-216 (vid. pág. 19).

[24] Maida, Esteban Gabriel Pacienzia y Julián. «METODOLOGÍAS DE DESARROLLO DE SOFTWARE». Espanol. Tesis Final de Licenciatura en Sistemas y Computación. Argentina: PONTIFICIA UNIVERSIDAD CATÓLICA ARGENTINA SANTA MARIA DE LOS BUENOS AIRES, 2015. URL: http://bibliotecadigital.uca.edu.ar/repositorio/tesis/metodologias-desarrollo-software.pdf (vid. pág. 19).

[25] Jonás Montilva, Nelson Arapé y J Colmenares. «Desarrollo de software basado en componentes». En: *Actas del IV. Congreso de Automatización y Control. Mérida, Venezuela*. 2003 (vid. pág. 19).

[26] Craig Larman. *UML y Patrones: Introducción al análisis y diseno orientado a objetos*. Espanol. 1ra. México: Prentice Hall Hispanoamericana, S.A, 1999 (vid. págs. 20, 31).

[27] Roger S Pressman. *Ingeniería de software: Un enfoque práctico*. Español. 5ta. Nueva York, EUA: McGraw-Hill, 2003. ISBN: 907-17-0261-1 (vid. pág. 20).

[28] Visual Paradigm. «Visual paradigm for uml». En: *Visual Paradigm for UML-UML tool for software application development* (2013), pág. 72 (vid. pág. 20).

[29] Dirk Riehle. «Framework design». Tesis doct. ETH Zurich, 2000 (vid. pág. 20).

[30] Ray Rischpater. *Application Development with Qt Creator*. Packt Publishing Ltd, 2014 (vid. pág. 21).

[31] Bjarne Stroustrup. *The C++ programming language*. Pearson Education India, 2000 (vid. pág. 21).

[32] Gabriel Saldana. *Introducción a Git: Un Sistema de control de versiones...bien hecho*. Español. URL: http://blog.nethazard.net (vid. pág. 22).

[33] Fermín González García y JD Novak. «El Mapa Conceptual y el Diagrama V. Recursos para la Enseñanza Superior en el siglo XXI». En: *Narcea, SA de Ediciones* (2008) (vid. pág. 24).

[34] Alina Dolores Rodrguez Pea y Luis Guillermo Silva Rojas. «Arquitectura de software para el sistema de visualización médica Vismedic». En: *Revista Cubana de Informática Médica* 8.1 (2016), págs. 75-86 (vid. pág. 26).

[35] Roger S Pressman. *Ingeniería de software: Un enfoque práctico*. Español. 6ta. Nueva York, EUA: McGraw-Hill, 2007 (vid. págs. 30, 31).

[36] Ian Sommerville. *Software Engineering*. Inglés. 8va. Pearson Education Limited, 2006. ISBN: 0-321-31379-8 (vid. págs. 30, 31).

[37] José P Miguel, David Mauricio y Glen Rodríguez. «A review of software quality models for the evaluation of software products». En: *arXiv preprint arXiv:1412.2977* (2014) (vid. pág. 31).

[38] José H Canós, Patricio Letelier y María Carmen Penadés. *Metodologías Ágiles en el Desarrollo de Software*. Espanol. Inf. téc. Valencia, Espana: Universidad Politécnica de Valencia, pág. 8 (vid. pág. 31).

[39] Craig Larman. *UML y Patrones*. Pearson Educación, 2003 (vid. pág. 32).

[40] Carlos A Guerrero, Johanna M Suárez y Luz E Gutiérrez. *Patrones de Diseno GOF (The Gang of Four) en el contexto de Procesos de Desarrollo de Aplicaciones Orientadas a la Web*. Espanol. Inf. téc. Santander, Colombia, 2009, pág. 12 (vid. págs. 32, 33).

[41] Grady Booch et al., *El lenguaje unificado de modelado*. Vol. 1. Addison Wesley Madrid, 1999 (vid. pág. 35).

[42] Michael Meißner et al., «A practical evaluation of popular volume rendering algorithms». En: *Proceedings of the 2000 IEEE symposium on Volume visualization*. ACM. 2000, págs. 81-90 (vid. pág. 35).

[43] Kwansik Kim y Alex Pang. «Ray-based data level comparisons of direct volume rendering algorithms». En: *Scientific Visualization Conference, 1997*. IEEE. 1997, págs. 137-137 (vid. pág. 35).

Generado con LaTeX: 20 de junio de 2018: 2:00pm