Proyecto de Grado

Desarrollo de una aplicación para trabajar con la técnica de "Stop Motion"

Integrantes

Nicolás Sierra

Enrico Pizzorno

Juan Diego Ferré

Responsables

Eduardo Fernández

Pablo Rebufello

Instituto de Computación - Facultad de Ingeniería- Universidad de la República

Año 2003

Tabla de Contenido

Introducción

El objetivo inicial de este proyecto de grado fue desarrollar una aplicación que permita trabajar con la técnica de “Stop Motion”, utilizada para la realización de animaciones en cine, televisión, etc.

Dicha técnica consiste en animar un objeto a través de una secuencia de imágenes que son capturadas por una cámara, de manera que al reproducirlas luego en una sucesión temporal se obtenga la ilusión de un movimiento continuo. Esta técnica puede aplicarse también con dibujos (o imágenes generadas de alguna forma) para realizar caricaturas animadas.

La ventaja de utilizar una computadora para procesar una animación es que permite visualizar el resultado inmediatamente y realizar las correcciones necesarias. No hay que esperar a revelar una película, como sucedía anteriormente.

Este proyecto surge por interés de los responsables en promover un acercamiento entre la Facultad de Ingeniería y los miembros de la comunidad audiovisual uruguaya. A partir de esta idea, se realizó un primer contacto con Walter Tournier y su equipo (creadores de “Los Tatitos”), quienes se mostraron sumamente interesados en la colaboración y establecieron que uno de los problemas técnicos importantes que enfrentan se haya en el software utilizado para implementar la técnica de “Stop Motion”.

Una de las aplicaciones que utilizan, *Cartoon Television Program* (*CTP*) de Crater Software [1], les causa ciertas dificultades y carece, según su experiencia de uso, de algunas funcionalidades importantes que serían muy útiles en el desarrollo de la técnica.

La aplicación desarrollada cubre el subconjunto de funcionalidades de CTP que realmente utilizan e incorpora aquellas extra que requirieron específicamente.

Fue implementada de manera que sea independiente del hardware de captura de video específico con el que cuentan y pueda ser portada a diferentes sistemas operativos con relativa facilidad. Al momento de escribir este informe, es posible ejecutarla sobre Linux y las diferentes versiones de Windows.

Se dio a la aplicación el nombre de *QPencil Test*, o *QPT*.

Análisis de Requerimientos

Los requerimientos se obtuvieron principalmente del análisis de CTP [1]. Se realizaron reuniones de requerimientos con Tournier y su equipo al comienzo del proyecto para identificar las funcionalidades importantes de dicho programa, así como aquellas que consideraban importante incorporar.

Lo esencial de estas reuniones fue establecer claramente el alcance y los objetivos finales que debía alcanzar la aplicación. Vale aclarar que CTP fue pensado para ayudar al artista a lo largo de todo el proceso de creación de una animación, comenzando por la verificación y vista previa de los primeros dibujos, pasando por la coloración de imágenes o los movimientos de cámara y terminando con la exportación de la animación a un dispositivo capaz de grabarla en medios magnéticos como las cintas de video. Se consideró que implementar una aplicación que cumpla con todas estas funcionalidades supera las capacidad de desarrollo de un solo proyecto de grado.

La mayor necesidad de Tournier y su equipo era contar con una aplicación que los ayudara en la etapa inicial del proceso, que consiste en realizar los dibujos que formarán parte de la animación (a lápiz, en blanco y negro), capturarlos con la cámara de video como imágenes estáticas y generar con ellas un video para verificar que los movimientos de los personajes son continuos, evaluando además los tiempos de exposición de las imágenes para que se sincronicen con el audio (si corresponde). A esta etapa, CTP la denomina *Pencil Test*.

Se optó entonces por relevar como requerimientos las funcionalidades para la etapa de Pencil Test que brinda CTP y agregar a estos los que plantearon específicamente tanto Tournier y su equipo como los responsables del proyecto.

Descripción de la Realidad

El elemento central de la realidad a considerar es la *escena* (scene). Técnicamente, se define escena como un fragmento de animación pequeño donde, por lo general, intervienen los mismos personajes.

Los atributos básicos de una escena son nombre (que la identifica), dibujante, número de escena y número de secuencia. Cada escena cuenta además con un formato de salida que define:

- Alto y ancho de los cuadros (en píxels).
- Velocidad de reproducción (en cuadros por segundo).

Tanto el nombre como el formato de salida no cambian luego de establecidos.

Al trabajar con la técnica de “Stop Motion” se divide el tiempo en intervalos consecutivos de igual duración. Para cada intervalo se define un *cuadro*, que se compone del fragmento de audio asociado al intervalo y de una imagen estática, que define la expresión gráfica de la escena en ese período de tiempo. A partir de esta definición se puede establecer que una escena es, en esencia, una secuencia de cuadros.

La imagen estática de un cuadro se puede construir a partir de otras imágenes. En este caso, se ordenan dichas imágenes por *niveles* y se superponen una a una, comenzando por el nivel inferior, hasta llegar al superior. En general, se trabaja con imágenes de trazos o zonas negras sobre fondo blanco, tomando el blanco como “transparente” para el proceso de superposición. La Figura 1 explica esto con un pequeño ejemplo.

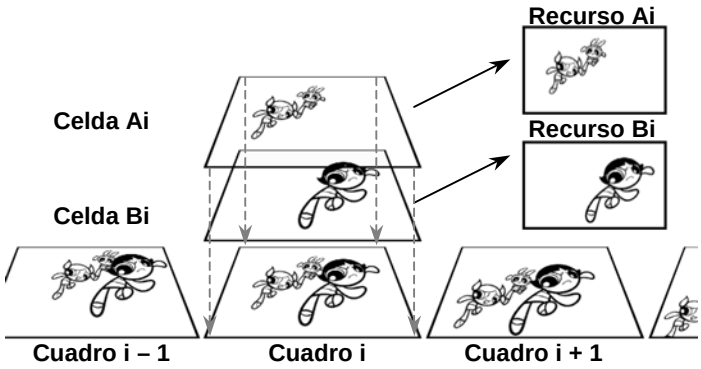

Figura 1 - Esquema de la Realidad. La imagen del cuadro *i* se obtiene superponiendo las imágenes de las celdas A*i* y B*i*. Se toma el fondo blanco como transparente.

Las imágenes se organizan con cierta coherencia en niveles. Por ejemplo, es común crear un nivel para el fondo de la escena, y otros para el movimiento de los personajes. A cada nivel se le asigna un nombre descriptivo.

El audio también se organiza en niveles. La pista de sonido de una animación se logra superponiendo los diferentes niveles de audio. En general no es común que se utilice más de un nivel de este tipo.

Se denominan *recursos multimedia* (resources), o simplemente recursos, a los archivos de imagen y audio que conforman los cuadros. Cada recurso tiene un nombre que lo identifica.

Los recursos de imagen pueden tener un cierto desplazamiento (offset) con respecto al cuadro de la escena. Dicho desplazamiento indica la posición de la esquina superior izquierda de la imagen con respecto a la esquina superior izquierda del cuadro.

Para facilitar el trabajo con los elementos antes mencionados se utiliza la *hoja de exposición* (exposure sheet), cuyo esquema se ve en la Figura 2. La hoja de exposición de una escena es una planilla o matriz que ordena los elementos hasta ahora mencionados. Cada fila de la planilla representa un cuadro de la escena y cada columna un nivel. El orden de arriba a abajo de las filas coincide con el orden cronológico de los cuadros, a la vez que el orden de izquierda a derecha de las columnas se corresponde con el orden de superior a inferior de los niveles. Un recurso se puede asignar a una o más *celdas* de la matriz, indicando que dicho recurso pertenece a ciertos niveles y se utiliza para construir ciertos cuadros.

	SND1	casa	pelota	fondo
1	S0001	**A0001**	**C0001**	**D0001**
2	S0001	A0001	C0001	D0001
3	S0001	A0001		D0001
4	S0001	A0001		D0001
5	S0001	**A0002**	**C0003**	D0001
6	S0001	A0002	C0003	D0001
7	S0001	A0002	C0003	D0001
8	S0001	A0002		D0001

Figura 2 - Fragmento de una hoja de exposición. Las filas representan los cuadros y las columnas los niveles de una escena. El nivel SND1 es de audio y los siguientes de imagen.

Los *bloques* (blocks) son conjuntos de celdas consecutivas de un mismo nivel, que tienen asignado el mismo recurso. Un bloque agrupa dichas celdas para poder trabajar con ellas en conjunto. Cada bloque se identifica por el nombre del nivel al que pertenece y el índice de la celda en el que comienza. Tiene además un cierto largo. En la Figura 2 se destaca la celda inicial de cada bloque con el identificador del recurso en “negrita”.

Modelo Conceptual

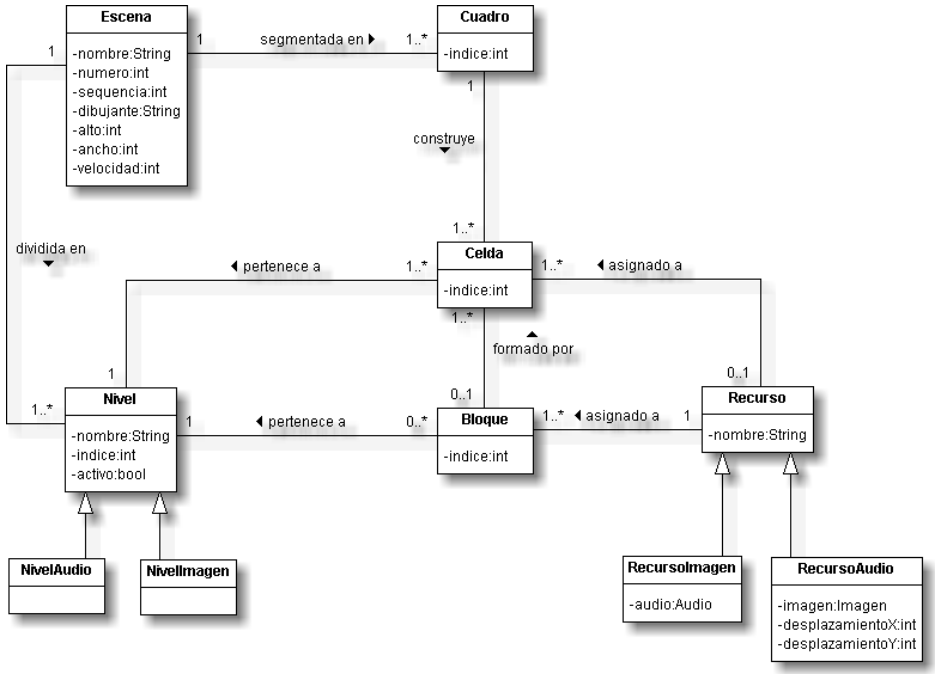

Figura 3 – Modelo Conceptual de la Realidad

Requerimientos Principales

Los requerimientos principales de la aplicación son:

- Crear escenas nuevas, aplicando ciertos valores por defecto.
- Guardar los cambios de una escena o guardarlos como una escena nueva.
- Abrir escenas guardadas anteriormente.

- Crear, eliminar, renombrar, activar y desactivar niveles.
- Aumentar o disminuir el largo de los bloques, brindando ciertas cantidades por defecto.
- Cortar, copiar, pegar, arrastrar y soltar celdas (y sus recursos asignados) entre escenas abiertas. En el caso de las celdas de sonido, realizar las operaciones en bloque.
- Importar imágenes desde una cámara o desde archivos y asignarlas como recursos de cualquier celda de imagen.
- Filtrar las imágenes importadas de manera de que queden los dibujos en negro y el fondo en blanco.
- Visualizar la imagen asignada a cualquier celda de imagen.
- Importar archivos de sonido y asignarlos como recursos a cualquier celda de sonido (en bloque).
- Construir y reproducir el video de una escena componiendo las celdas de sus niveles activos.
- Exportar el video como archivo.

Los requerimientos más importantes especificados por Tournier y su equipo son:

- Abrir varias escenas al mismo tiempo.
- Invertir el orden de un grupo seleccionado de celdas consecutivas.
- Reproducir el video de una escena en pantalla completa.
- Imprimir la hoja de exposición.
- Ejecutar en Windows (implícito).

Los requerimientos más importantes especificados por los responsables del proyecto fueron:

- Guardar toda la información y recursos de una escena agrupados en un único archivo.
- Deshacer y rehacer operaciones.
- Utilizar lo menos posible características propias de un sistema operativo y, en caso de ser necesario, identificarlas adecuadamente y limitarlas.
- Independencia del hardware de captura de video.
- Extensibilidad.
- Manual de usuario.

El Anexo A lista los todos requerimientos de la aplicación.

Arquitectura y Diseño

Descripción de la Arquitectura

La arquitectura se divide en dos secciones: la *aplicación principal* y los *plugins*, como muestra la Figura 4.

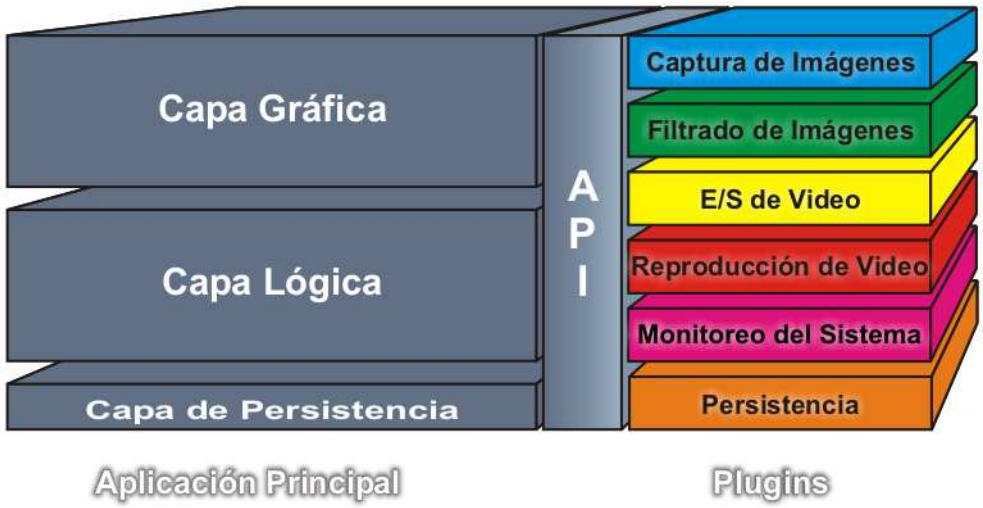

Figura 4 – Diagrama General de la Arquitectura

La aplicación principal se divide a su vez en dos grandes capas: la *lógica* y la *gráfica*, utilizando el patrón de diseño *Model–View–Controller* o Modelo–Vista–Controlador (MVC) [2] para asignar responsabilidades a cada una.

Dicho patrón indica que la capa lógica es la encargada del modelo de datos, encapsulando la información y el control sobre el ciclo de vida de escenas, recursos, etc. A su vez brinda servicios necesarios para modificar dichos datos en forma segura.

La capa gráfica reúne vista y controlador. Muestra la información en pantalla; captura y transfiere hacia el modelo las acciones que toma el usuario y refleja los cambios consecuentes. La Figura 5 muestra la ventana principal de la aplicación.

La comunicación de la capa lógica hacia la gráfica no es directa, sino a través de eventos. El modelo genera un evento por cada suceso o cambio relevante y lo informa a los componentes interesados, sin tener conocimiento directo sobre los mismos. Esto ayuda a la independencia entre capas.

Los errores que ocurren en las operaciones del modelo se propagan mediante el uso de excepciones, que son capturadas en la capa gráfica para poder informar de lo sucedido al usuario.

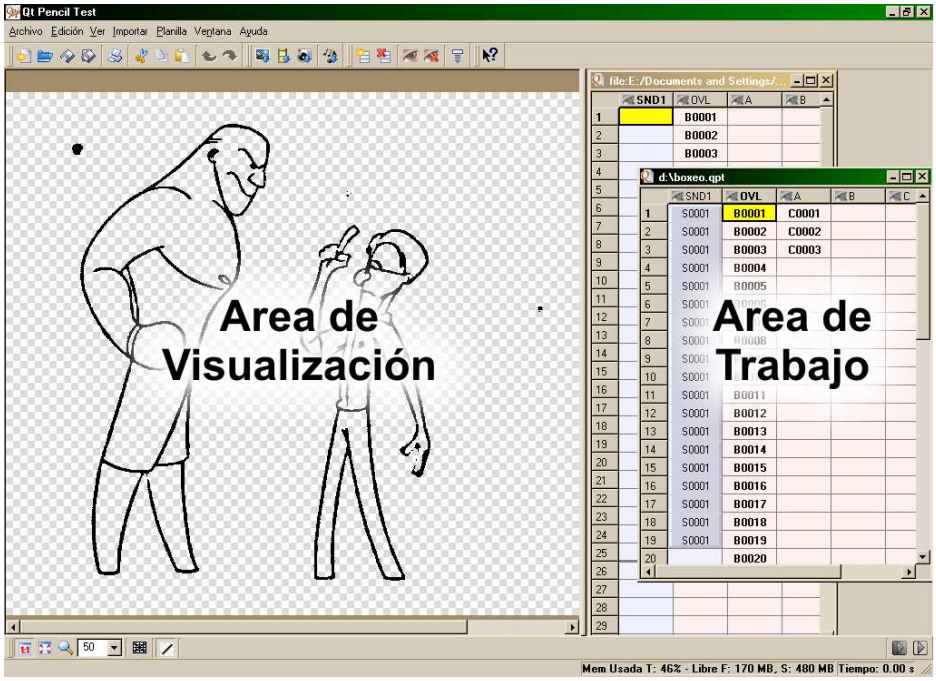

Figura 5 - Ventana principal de la aplicación.

La aplicación principal cuenta también con una delgada capa de persistencia que maneja únicamente los cambios en las opciones de configuración del programa. No cumple otra función, dado que la persistencia de las escenas está a cargo de los plugins de persistencia.

Los plugins son módulos o componentes responsables de ciertas funciones que es conveniente separar explícitamente de la aplicación principal, ya sea porque plantean una fuerte dependencia con el sistema operativo, o porque se consideran puntos clave de gran importancia para futuras extensiones.

Una *Application Programming Interface* o Interfaz de Programación de la Aplicación (API) define formalmente la interacción de la aplicación principal con los plugins, y viceversa. Consiste básicamente en una jerarquía de clases que los desarrolladores deben respetar para asegurarse la compatibilidad de ejecución.

Las funciones de los plugins son esencialmente lógicas. Sin embargo, la API provee también cierta capacidad de integración con la capa gráfica, para permitir al usuario modificar o consultar parámetros de configuración de los mismos en forma visual. A su vez, la capa de persistencia permite que los plugins guarden información junto con la configuración del programa.

Se puede consultar información más detallada sobre la API y la interacción con las clases que la forman en el Manual Técnico o a la Documentación de Clases.

La Figura 6 muestra la sección “Plugins” del diálogo de configuración, que lista todos los plugins disponibles en cierta ejecución de la aplicación, separados por categorías. A modo de ejemplo de la integración comentada antes se ven las

últimas dos secciones, “Reproductor externo” y “FFmpegAVEngine”, correspondientes a los plugins del mismo nombre.

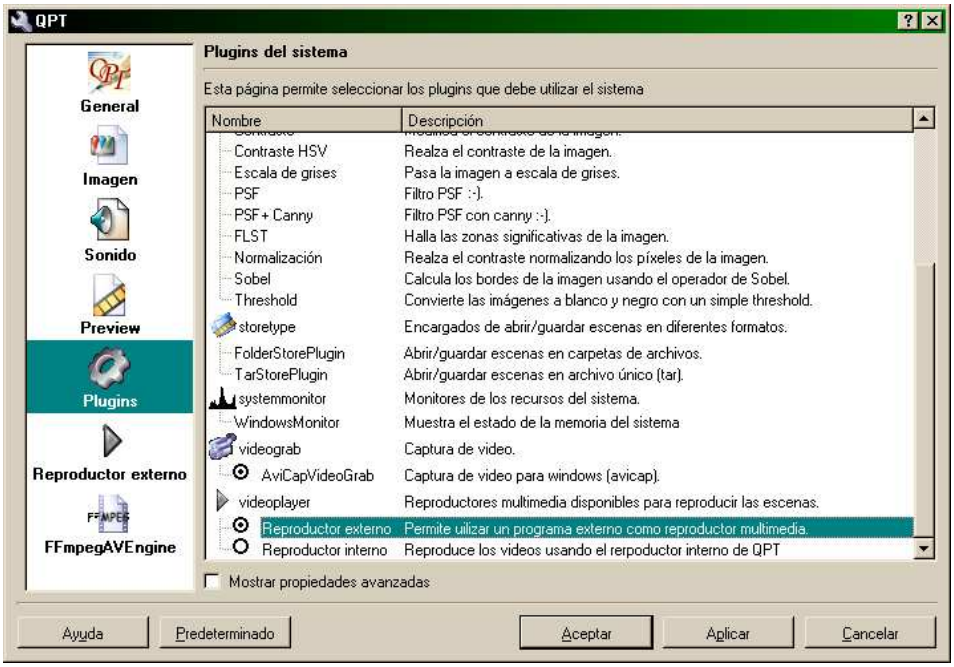

Figura 6 - Diálogo de configuración. Muestra la página “Plugins” que lista los plugins cargados por la aplicación principal al momento de iniciar.

Los plugins se dividen en seis categorías o tipos según las funciones que cumplen:

- **Captura de imágenes**: obtienen imágenes estáticas desde dispositivos de captura conectados a la computadora.
- **Filtrado de imágenes**: obtienen la imagen resultante de aplicar un cierto filtro a otra imagen.
- **Entrada/Salida de video**: manejan todo lo relacionado a la lectura y escritura de archivos de audio y video (formatos de archivo, codecs de audio y video, etc).
- **Reproducción de video**: exhiben al usuario un cierto archivo de video.
- **Monitoreo del sistema**: informan al usuario el estado del sistema (memoria libre, memoria usada, espacio disponible en el disco duro, etc.).
- **Persistencia**: manejan la persistencia de cada escena (cómo y dónde se guardan las escenas).

Puede haber varios plugins de cada tipo. Debido a la naturaleza de las operaciones, algunos tipos solo aceptan un único plugin activo a la vez. En ese caso, el diálogo agrega un botón de selección (round button) a cada entrada de la lista para que el usuario pueda seleccionar el que desea utilizar.

El Anexo B explica las razones principales por las cuales se introducen los plugins en este proyecto. Básicamente, surgen como una alternativa elegante a las directivas del precompilador para poder separar código destinado a una plataforma en particular, facilitando el proceso de compilación y favoreciendo el desarrollo en paralelo.

Descripción de Componentes

Diagrama

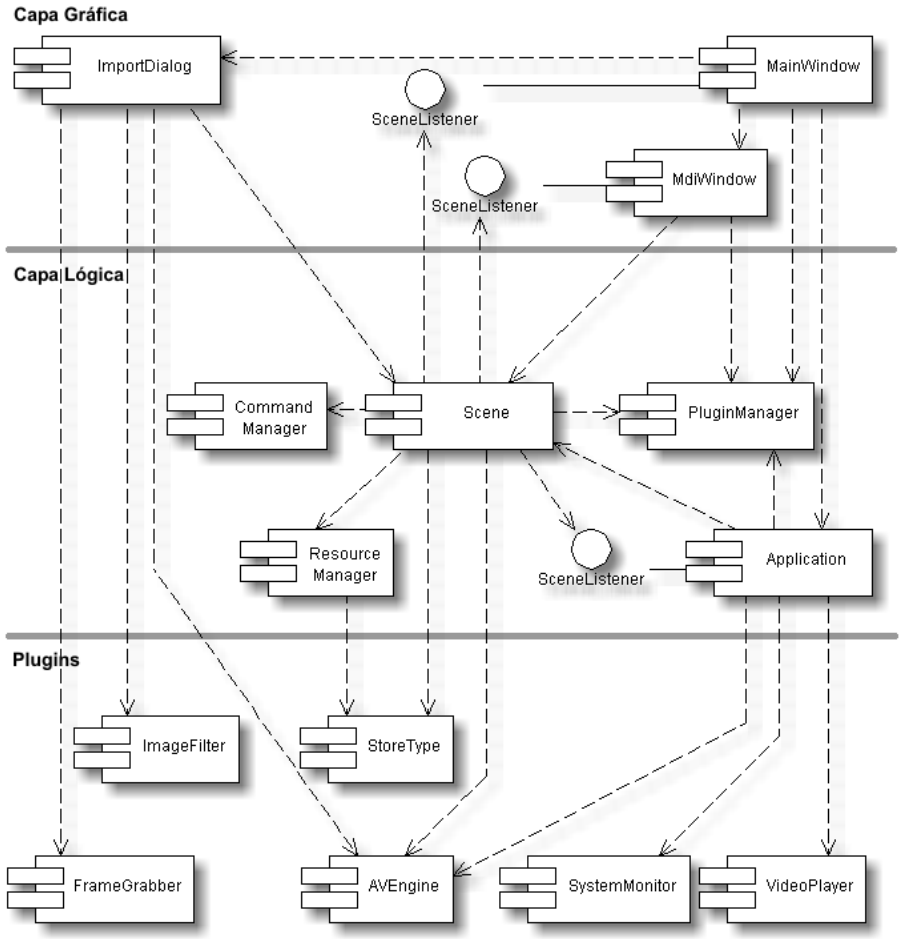

Figura 7 - Diagrama de Componentes.

Capa Lógica

Application

`Application` es el componente encargado de crear, eliminar y controlar el estado de las escenas. Cumple además tareas de inicialización y coordinación de funciones que se ejecutan sobre todas las escenas abiertas.

Scene

`Scene` reúne todas las clases que conforman el modelo de datos de una escena y brinda las operaciones necesarias para modificarlo. Los eventos que genera son comunicados a los componentes que soporten la interfaz `SceneListener`.

ResourceManager

`ResourceManager` administra los recursos de una escena y funciona básicamente como un diccionario. Cada `Scene` tiene su propio `ResourceManager` asociado.

CommandManager

`CommandManager` registra los comandos que se aplican sobre `Scene` en orden de ejecución y brinda operaciones para deshacer o rehacer dichos comandos. Cada `Scene` tiene su propio `CommandManager` asociado.

PluginManager

`PluginManager` maneja y controla el acceso a los plugins en tiempo de ejecución. Permite a los demás componentes realizar búsquedas de plugins por nombre o tipo.

Capa Gráfica

MainWindow

`MainWindow` sirve principalmente de marco para los componentes `MdiWindow`. Corresponde a la mayor parte de la interfaz gráfica. Captura las acciones del usuario y las redirige a la `MdiWindow` que se encuentre en primer plano.

MdiWindow

`MdiWindow` es vista y controlador para `Scene`. Muestra una grilla que representa la hoja de exposición y permite modificar niveles o celdas de una escena.

ImportDialog

`ImportDialog` es el componente que permite al usuario la importación de imágenes desde varias fuentes, como son archivos de imagen, video o dispositivos de captura. Es considerado como otra vista más específica de `Scene`.

Plugins

FrameGrabber

Los plugins responsables de la captura de imágenes se denominan `FrameGrabbers`. Obtienen para el diálogo de importación una lista de los dispositivos de captura de video disponibles y permiten capturar imágenes estáticas desde cualquiera de ellos.

También proveen funciones para modificar los parámetros de captura, como son el alto, ancho y resolución de las imágenes, y controlar que dichos parámetros sean soportados por la aplicación.

Solo puede haber un plugin activo de este tipo a la vez.

ImageFilter

Los plugins de este tipo cumplen con la importante función de aplicar un cierto filtro a una imagen. Esto es, el diálogo de importación le entrega una imagen al plugin, este le aplica una cierta transformación y devuelve el resultado como una nueva imagen.

Pueden ser utilizados de manera secuencial, es decir, el resultado de un filtro puede ser la entrada de otro. El diálogo de importación es quien controla este mecanismo.

AVEngines

Se denominan `AVEngines` (por “audio/video engines”) a los plugins que permiten a la aplicación principal leer y escribir archivos de audio y video.

Eso implica que encapsulan e implementan la lógica correspondiente a la interpretación de formatos de archivo, uso codecs de audio y video, conversión entre formatos de imagen, etc.

Este tipo de plugin es muy importante ya que la aplicación realiza todo el manejo de video a través de archivos y, por lo tanto, su rendimiento influye de manera significativa en la performance final y, en cierto grado, en la usabilidad.

Solo puede haber un plugin activo de este tipo a la vez.

VideoPlayers

La función principal de estos plugins es la de reproducir archivos de video que la aplicación construye mediante el uso de `AVEngines`.

Es necesario entonces que los plugins de ambos tipos estén “coordinados”, en el sentido que los `VideoPlayers` deben tener la capacidad de interpretar los formatos y codecs que utilicen los `AVEngines` para construir los videos.

Solo puede haber un plugin activo de este tipo a la vez.

SystemMonitors

Los plugins denominados `SystemMonitors` cumplen la función de informar el estado del sistema al usuario (memoria RAM ocupada y libre; espacio en disco disponible, etc). Cada cierto tiempo (constante) la aplicación principal interroga a los `SystemMonitors` para que actualicen la información que presentan.

StoreTypes

La principal responsabilidad de los plugins de este tipo es la de crear *almacenes* (stores).

Los almacenes son componentes que brindan a las escenas las funciones y los medios para guardar y recuperar información sobre la hoja de exposición o los archivos de recursos.

Si bien la implementación de estos almacenes no está restringida de ninguna manera, sí se define el mecanismo de intercambio entre escenas y almacenes: archivos en disco. Por ejemplo, la manera de agregar una nueva imagen a una escena es pedirle al almacén la ruta para guardarla como archivo. Para recuperar dicha imagen en otra oportunidad, se le pide al almacén la ruta de archivo desde dónde leerla.

Descripción de Clases

En esta sección se presentan algunos de los aspectos o decisiones del diseño más importantes y las clases que involucran directamente. Para ver en detalle la documentación de cada clase o método, referirse a la Documentación de Clases.

Escena

La Figura 8 muestra una vista de las clases que conforman el componente Scene y los métodos más importantes. Se omiten algunos métodos en favor de la claridad del diagrama.

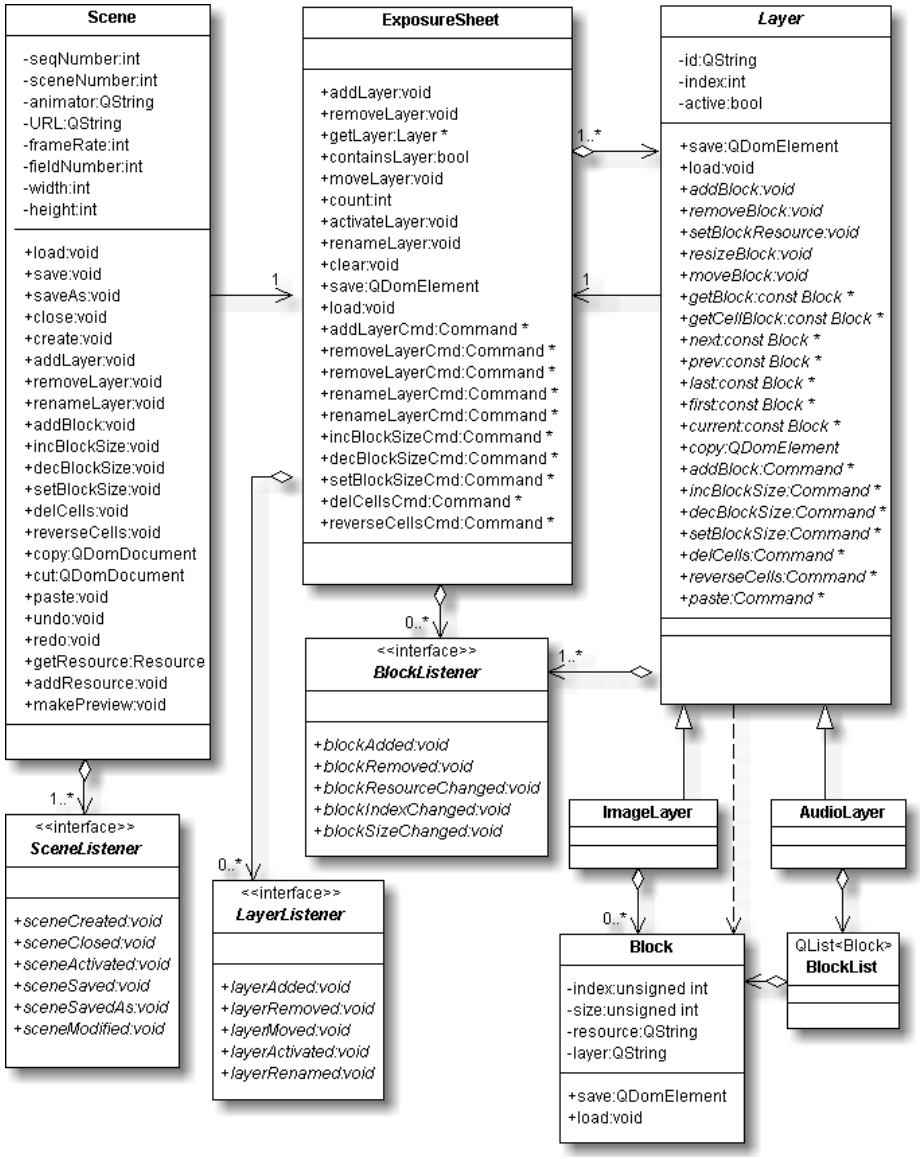

Figura 8 - Diagrama de clases del componente Scene (se omiten algunos métodos).

La clase Scene es la interfaz del componente. Expone hacia el exterior todas las operaciones implementadas y mantiene el estado (si fue creado, salvado, modificado, etc.). Coordina también ciertas tareas importantes, como lo es por ejemplo la generación de un video a partir de la escena.

Se modela la hoja de exposición como una matriz dispersa, donde los niveles son las columnas y cada nivel controla una colección de bloques. Se pierde en este punto del diseño el concepto de celda, ya que no es realmente necesario si se consideran como bloques de largo uno. Basta con agregar un atributo que indique el índice de la celda en la que comienza el bloque.

En los niveles de imagen se utiliza un vector de bloques, ya que es conveniente un rápido acceso a cada uno en la mayoría de las operaciones y se espera una alta densidad. En contraposición con los niveles de audio, cuya densidad de bloques va a ser baja en general y por lo tanto utiliza una lista ordenada.

Recursos

Para la creación de objetos `Resource` se aplica el patrón *Factory Method* [3]. Este patrón define una interfaz para crear un objeto, pero permite a subclases decidir qué clase instanciar efectivamente. Se utiliza con la intención de prever la futura extensión del tipo de recursos que maneja la aplicación. La Figura 9 muestra el diagrama de las clases relacionadas con el manejo de recursos.

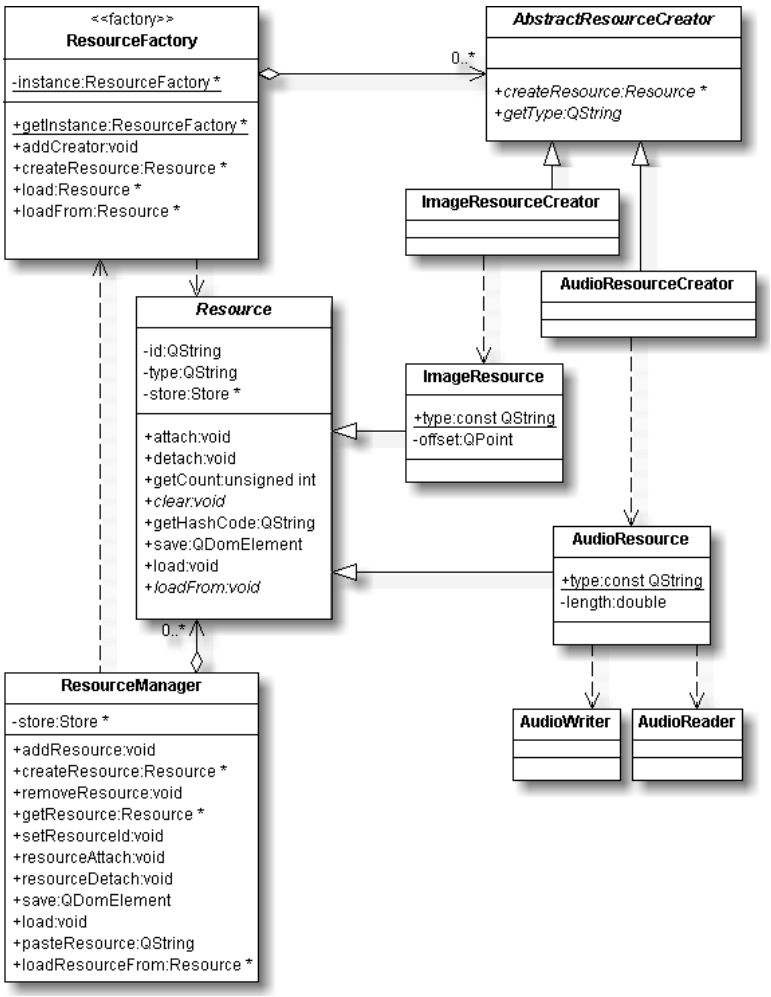

Figura 9 - Vista de clases relacionadas con el manejo de recursos (se omiten algunos métodos).

El patrón se varía levemente para que no sea `ResourceFactory` quien cree las instancias, sino que delegue esta tarea a clases creadoras que heredan de `AbstractResourceCreator`.

El diagrama de la Figura 10 muestra este proceso de creación, representado por las clases abstractas. El método estático `createResource()` recibe como parámetro un indicador del tipo de recurso que se desea crear. Los indicadores son strings asignados a la constante `type` de las clases creadoras específicas (`ImageResourceCreator` y `AudioResourceCreator`). Discriminando según el valor del indicador, `ResourceFactory` delega la creación misma del recurso a la clase creadora correspondiente.

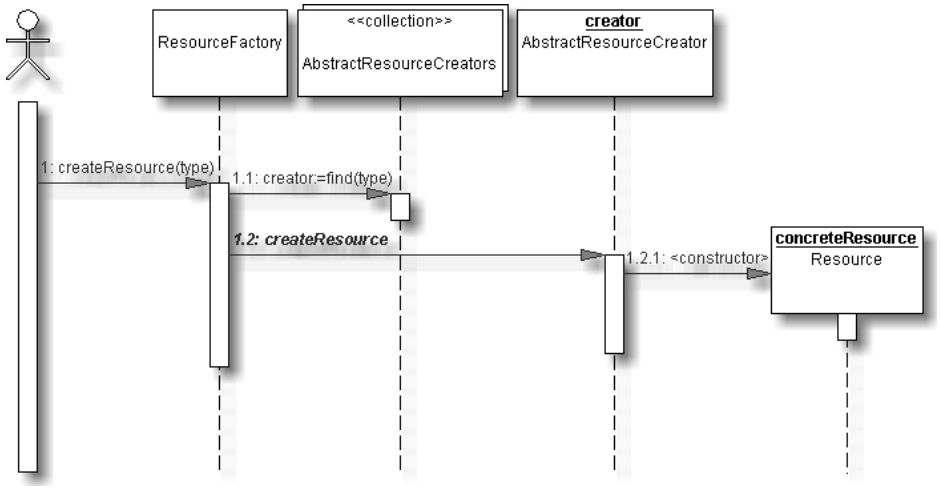

Figura 10 – Secuencia del proceso de creación de un objeto Resource.

Un recurso puede asignarse a varios bloques, pero se mantiene solamente una única instancia de cada recurso en tiempo de ejecución. Los objetos de clase `Resource` mantienen un contador de referencias `refcount` para controlar el número de bloques al que se encuentra asignado. En base al valor de este contador se podrán eliminar los recursos que dejen de utilizarse.

Tanto los `ImageResources` como los `AudioResource` tienen un archivo asociado. El contenido de estos archivos no se mantiene en memoria sino que se accede desde el disco cada vez que es necesario. Los tiempos de acceso no son grandes y la performance no se ve afectada, obteniéndose como ventaja un menor uso de la memoria.

Deshacer y Rehacer

Uno de los requerimientos que afectó de manera importante el diseño de la aplicación fue el de deshacer y rehacer las operaciones del usuario. Para resolverlo se aplica el patrón *Command* [3], que plantea encapsular operaciones como objetos, de manera de poder parametrizarlos y apilarlos, para brindar luego la posibilidad de deshacer dichas operaciones.

Por cada operación básica que se aplica al manejo de la hoja de exposición (agregar nivel, agregar bloque, etc.) se crea entonces una clase “comando” que hereda de la clase base `Command` y que implementa en sus métodos `execute`, `undo` y `redo` la lógica para hacer, deshacer y rehacer respectivamente las acciones que implica dicha operación.

Al deshacer una operación es necesario a veces restaurar el estado de un cierto objeto. Se combina entonces el patrón anterior con el patrón *Memento* [3], haciendo que cada instancia de comando recuerde y almacene los datos necesarios

y suficientes para poder restaurar los objetos que modifica a un estado anterior inmediato al de ejecución.

La Figura 11 muestra el diagrama de clases relacionadas con los comandos. Cada objeto de la clase `Scene` tiene un `CommandManager` propio, para poder deshacer y rehacer operaciones en cada una de las escenas abiertas (recordar que puede haber múltiples escenas abiertas en un momento dado). `Scene` obtiene los comandos de `ExposureSheet`, los ejecuta y luego los agrega a su `CommandManager`, donde serán apilados con cierta capacidad acotada y configurable. A medida que se ejecutan los comandos, se van descartando los más viejos.

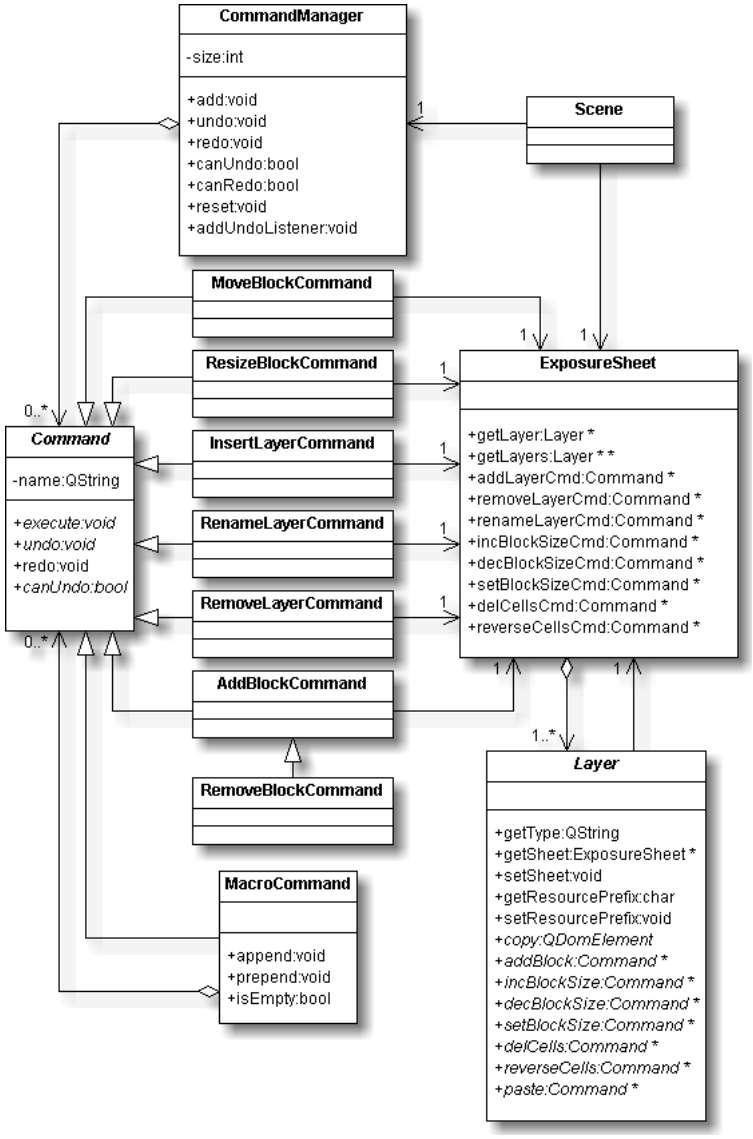

Figura 11 - Clases relacionadas con los comandos (se omiten algunos métodos)

Para ejecutar algunas de las funciones de la aplicación, como cortar o pegar, es necesario que se ejecuten varios comandos seguidos. Lo mismo sucede con comandos que se aplican sobre un conjunto de niveles. El patrón *Composite* [3] plantea cómo tratar objetos individuales y composiciones de los mismos indistintamente. Se define la clase `MacroCommand` como composición de un conjunto de comandos simples (eventualmente otro `MacroCommand`) que deben

ejecutarse en serie y ser considerados por el `CommandManager` como un único comando.

La Figura 12 muestra la creación de una instancia de `MacroCommand`, agrupando comandos que se ejecutan sobre varios niveles seleccionados. El método `setBlockSize()` redimensiona un conjunto de bloques. Recibe el nivel izquierdo, el cuadro superior, el nivel derecho y el cuadro inferior que delimitan el sector seleccionado por el usuario. `ExposureSheet` itera desde el nivel izquierdo al derecho, obteniendo un comando simple de cada uno y agrupándolos en un solo `MacroCommand` que retorna a `Scene`. Luego, `Scene` ejecuta dicho comando mediante el método `execute()` y lo agrega a la lista de `CommandManager`. Este es en general el proceso para cada comando de `Scene`.

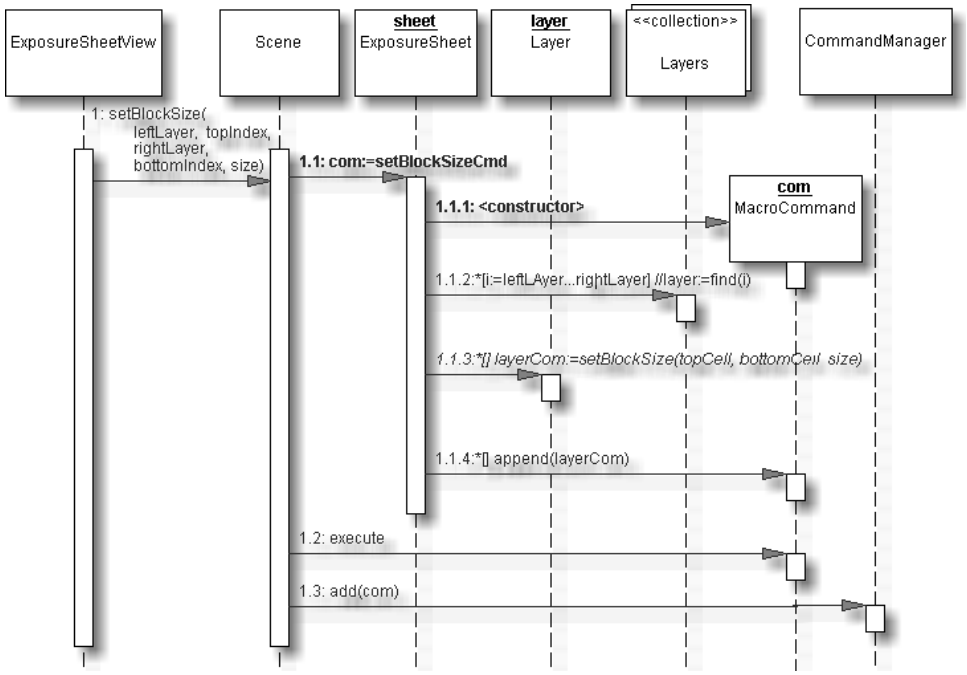

Figura 12 - Construcción y ejecución de un MacroCommand a partir de la función setBlockSize

Interfaz Gráfica

La interfaz gráfica de la aplicación cuenta con una ventana principal que corresponde a la clase `MainWindow`. Esta ventana se divide en dos sectores bien diferenciados, como muestra la Figura 13. A la izquierda el área de visualización de imágenes, que es un objeto de la clase `DisplayArea`. A la derecha, se encuentra el espacio de trabajo, con ventanas del tipo *Multi Document Interface* o Interfaz de Multiples Documentos (MDI) correspondientes a las escenas abiertas. Cada una de estas ventanas pertenecen a la clase `MDIWindow`. La Figura 14 muestra la relación entre las clases de la capa gráfica y la lógica.

Es necesario que `MainWindow` implemente la interfaz `SceneListener` para poder crear nuevas `MDIWindow` y asociarlas a las escenas que se crean o abren en la aplicación. También para ocultarlas y destruirlas cuando se cierran dichas escenas. La clase cuenta además con un conjunto de acciones y menús con los que interactúa el usuario. Gran parte de dichas acciones son transformadas en llamadas a métodos de la `MDIWindow` que se encuentre en primer plano (aquella con la que el usuario esté trabajando).

El principal objetivo de una `MDIWindow` es contener la vista de la hoja de exposición (`ExposureSheetView`) de la escena a la que fue asociada. Implementa por supuesto la interfaz `SceneListener` para ser notificada de los cambios que le ocurran a dicha escena.

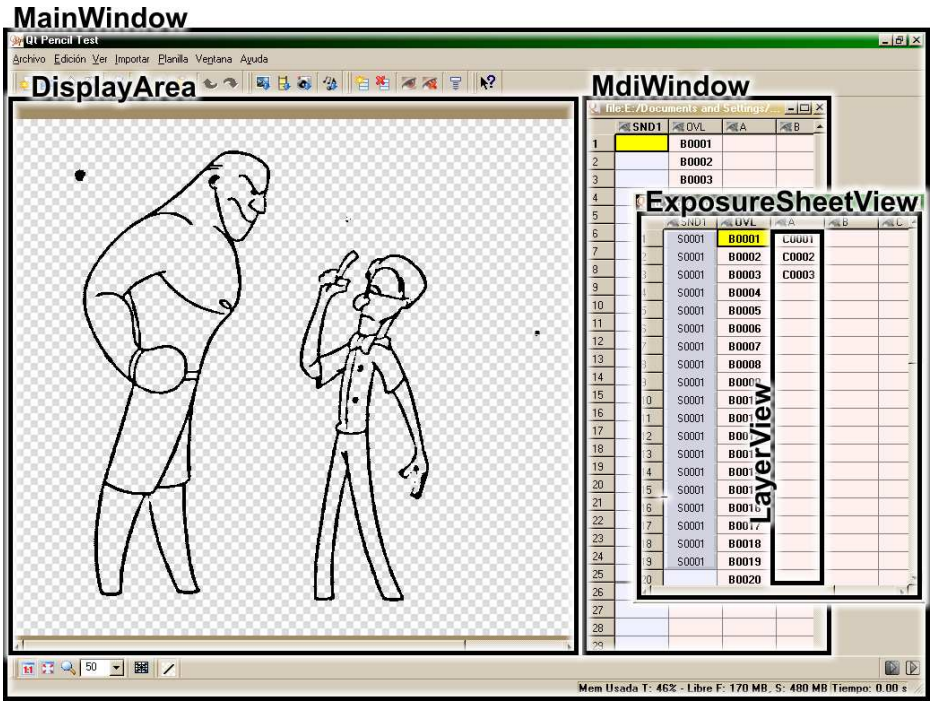

Figura 13 – Vista de la interfaz gráfica dividida en las clases que la implementan.

Una `ExposureSheetView` representa en forma de tabla o grilla la hoja de exposición de una escena. Cada columna de la tabla corresponde a un nivel y cada fila al período de tiempo de un cuadro de la animación. Esta clase implementa las interfaces `LayerListener` y `BlockListener` para recibir los eventos relacionados con los niveles y los bloques.

La tarea de pintar las celdas es delegada por la `ExposureSheetView` a objetos que heredan de la clase abstracta `LayerView`. Existe una subclase de `LayerView` por cada tipo de nivel existente (`ImageLayerView y AudioLayerView`) y una instancia de dichas subclases por cada nivel de la escena. Son estas instancias las que se encargan de pintar las celdas de la columna correspondiente en la tabla.

A modo de ejemplo, el diagrama de la Figura 15 muestra la interacción entre `ExposureSheetView` y `LayerView` ante la notificación de `Layer` de haber agregado un nuevo bloque. Se delega en `LayerView` el trabajo de repintar cada una de las celdas que componen el bloque. Esto mismo se hace para todos los eventos de `BlockListener`.

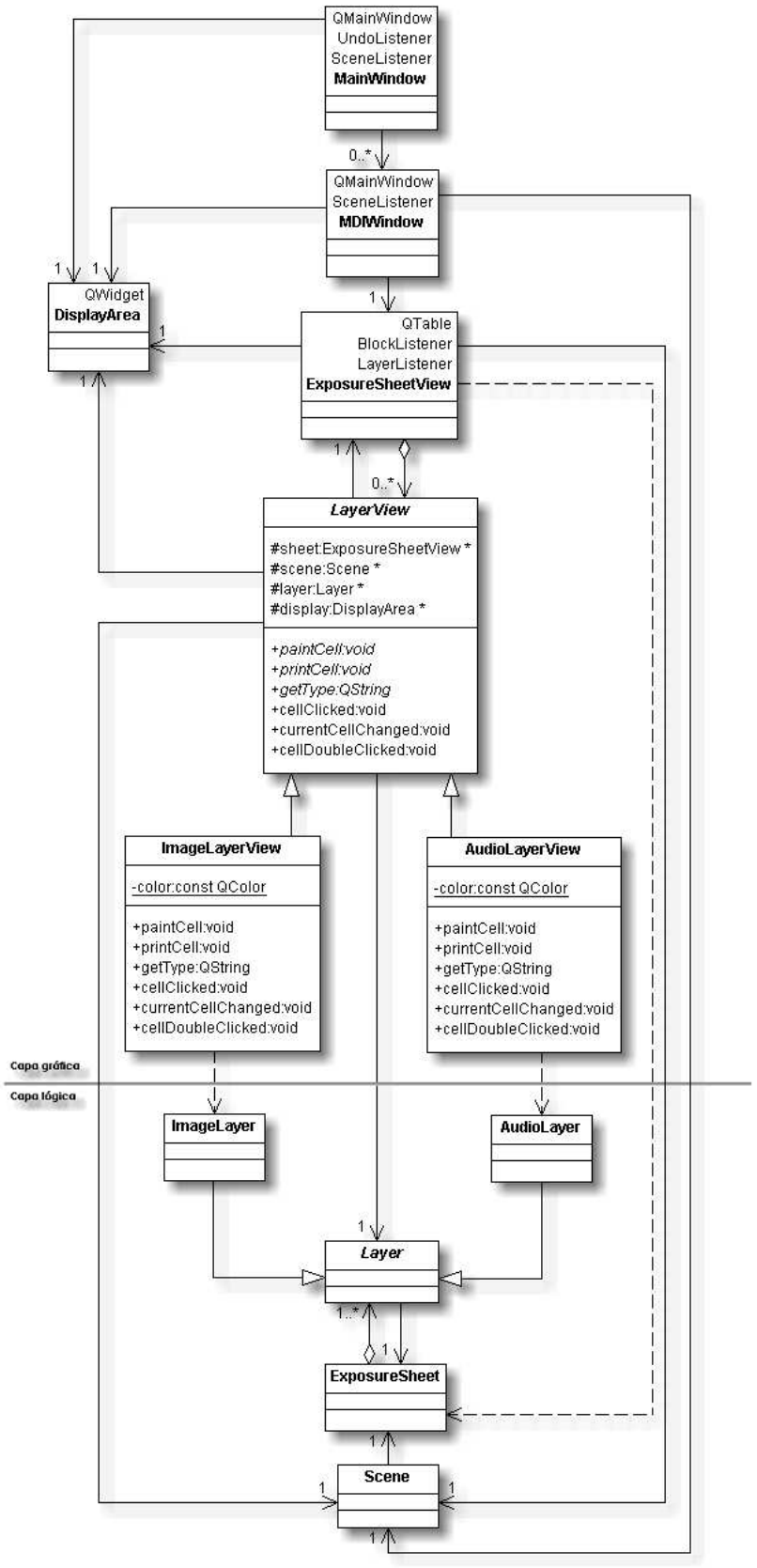

Figura 14 - Diagrama de clases de la capa gráfica y su interacción con la lógica (se omiten algunos métodos)

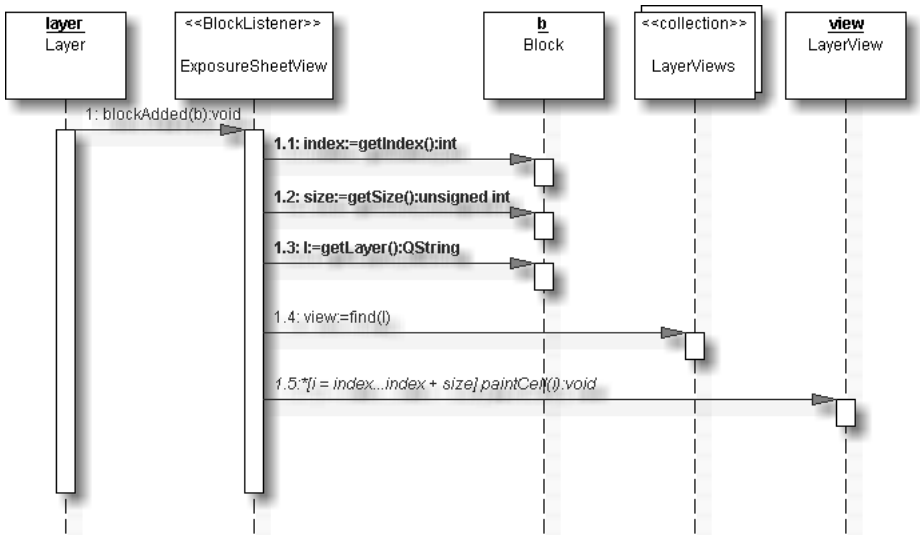

Figura 15 – Secuencia de ejecución para pintar las celdas de un nuevo bloque.

La `ExposureSheetView` notifica otros eventos a la `LayerView`, como ser el cambio de la celda activa, o que el usuario haga clic o doble clic sobre una celda. Cada especialización de `LayerView` responderá a esos eventos de forma adecuada, modificando si corresponde la `DisplayArea`. Por ejemplo al hacer clic sobre una celda no vacía de un nivel de imagen, la `ImageLayerView` obtendrá el recurso asignado a dicha celda y pintará la imagen en la `DisplayArea`. En cambio al hacer clic sobre una celda de una `AudioLayerView` se limpiará el contenido de la `DisplayArea`.

La delegación en `LayerView` de tareas dependientes del tipo de `Layer` aumenta la extensibilidad del programa, permitiendo agregar con cierta facilidad nuevos tipos de niveles y recursos e integrarlos a la interfaz gráfica.

Generación de video

La previsualización de una escena se realiza reproduciendo un archivo de video generado a partir de la hoja de exposición. La generación de este archivo de video es responsabilidad de la clase `Scene`.

Para construir el video, se generan en forma secuencial los cuadros del mismo a partir de los niveles activos de la escena.

Un cuadro completo de video se representa con la clase `SceneFrame.` Dicha clase contiene la imagen del cuadro (cuyo tamaño es el de los cuadros de la escena) y un fragmento de audio, representado por la clase `AudioFrame`. Un `AudioFrame` encapsula el sonido correspondiente solamente al tiempo de exposición del cuadro de video, esto es, el sonido que se reproduce mientras se ve la imagen. No fue necesario considerar una clase ImageFrame porque, como se verá más adelante, el marco de trabajo que se utiliza brinda una clase para el manejo de imágenes.

Cada cuadro de la escena se genera superponiendo los recursos de las celdas correspondientes de los niveles activos, comenzando con el nivel que se encuentra más a la derecha en la planilla y continuando hacia la izquierda hasta llegar al primer nivel de la selección de la hoja de exposición que se quiere previsualizar. La tarea de mezclar los recursos es delegada por `Scene` en objetos que implementan la

interfaz declarada en la clase abstracta `LayerVideoProducer`. Cada `Layer` debe proveer un `LayerVideoProducer` capaz de agregar la información de sus recursos al cuadro en construcción.

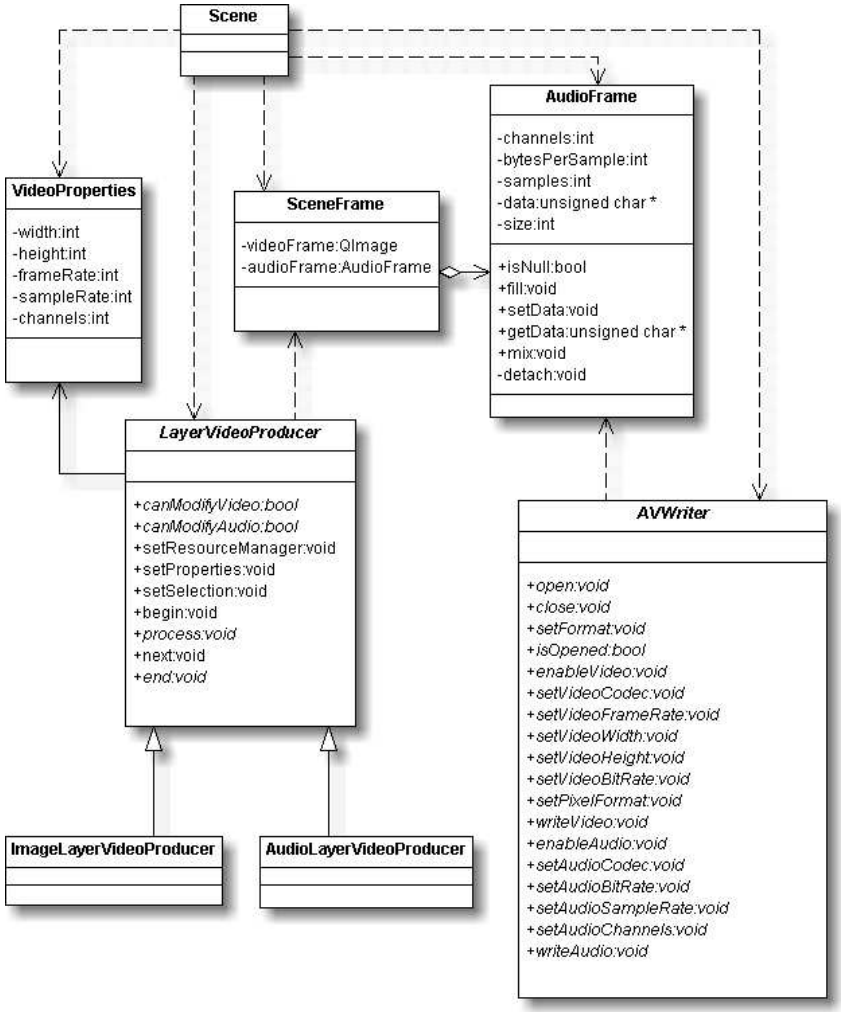

Figura 16 – Clases relacionadas con la generación de video

En cada paso de la generación del video `Scene` llama al método `process` de cada `LayerVideoProducer`, pasándole como parámetro el `SceneFrame` generado hasta el momento, es decir procesado por los `LayerVideoProducer` de los niveles de mayor índice. El método `process` es donde el `LayerVideoProducer` agrega la información relativa a su `Layer`, usando para eso el cuadro de imagen, o el fragmento de sonido del cuadro de escena. Por ejemplo el `ImageLayerVideoProducer` (`LayerVideoProducer` para la `ImageLayer`) procesa el `SceneFrame` superponiendo la imagen del recurso del bloque correspondiente a la celda a procesar, sobre el cuadro de imagen. En cambio el `AudioLayerVideoProducer` (`LayerVideoProducer` para la `AudioLayer`) mezcla la porción del recurso de audio correspondiente a la celda con el contenido del `AudioFrame`.

El diagrama de secuencia de la Figura 17 ilustra el proceso de generación de video de una escena. Las primeras operaciones son de inicialización, donde `Scene` obtiene los `LayerVideoProducer` de las escenas activas y establece las propiedades del video a generar (encapsuladas en la clase `VideoProperties`).

Posteriormente se ejecuta el ciclo en donde se genera cada cuadro y se agrega al video usando el `AVWriter`.

La clase abstracta `AVWriter` define la interfaz de los objetos que pueden ser utilizados por la aplicación para escribir archivos multimedia. Dicha interfaz contiene los métodos necesarios para abrir y cerrar archivos multimedia, así como para insertar información de audio y video en los mismos. Los `AVWriter` son provistos a la aplicación por los plugins del tipo AVEngine.

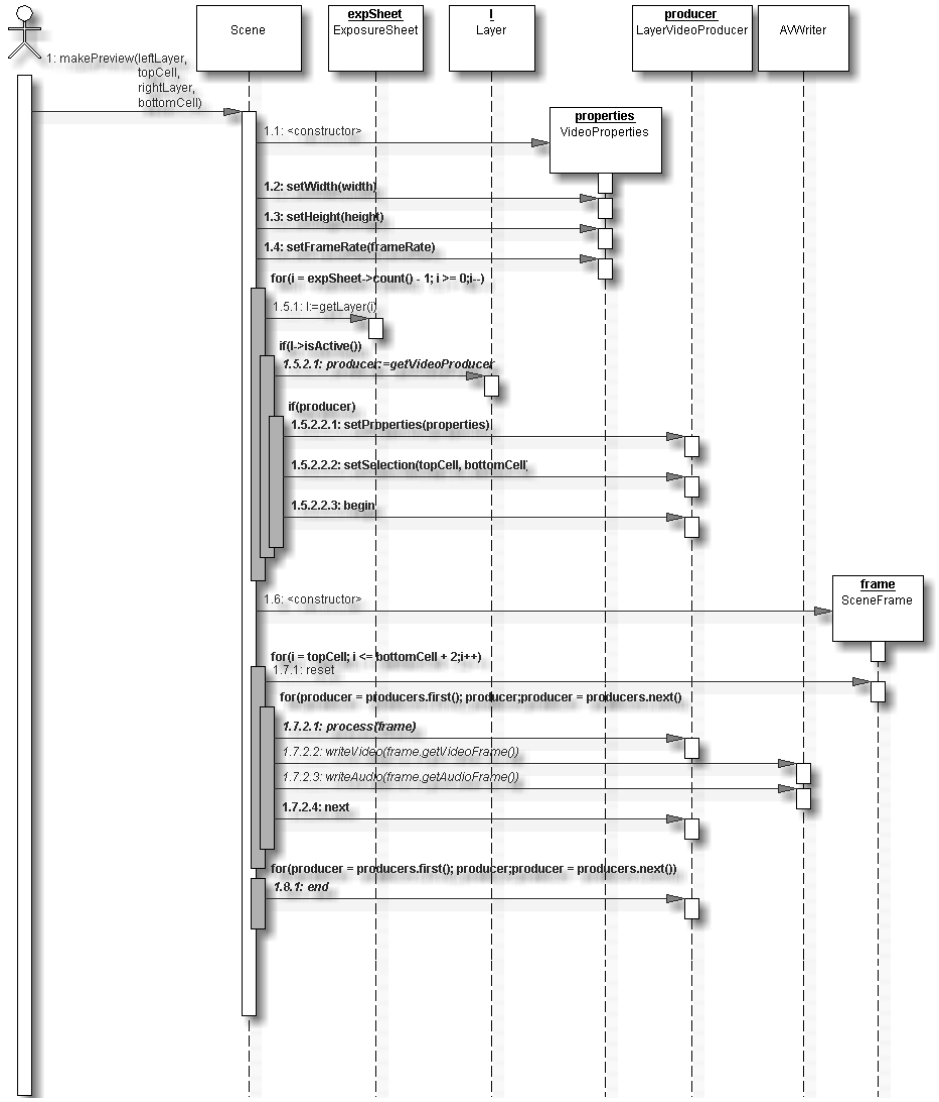

Figura 17 – Diagrama de secuencia para la generación de video

Implementación

Aplicación Principal

Herramientas de Desarrollo

La implementación de la aplicación principal (y los plugins) fue realizada en lenguaje C++. Se evaluó la opción de utilizar otros lenguajes que brindaran implícitamente una capa de abstracción sobre el sistema operativo y la plataforma, como Java o C#. Fue descartada principalmente por razones de rendimiento, dado que las computadoras objetivo de esta aplicación (las de Tournier y su equipo) no son de gran potencia.

Otra razón importante es que la naturaleza de la aplicación obliga a interactuar directamente con ciertas APIs de sistemas operativos o drivers de dispositivos, que están implementados en C. La comunicación en ese sentido es mucho mejor con C++ , dado que con cualquier otro lenguaje orientado a objetos se requieren bibliotecas intermedias que interpreten o traduzcan las estructuras de datos, llamadas a funciones, etc.

En Linux, se utilizaron las herramientas de desarrollo *GNU* [4] (g++, gmake, etc.) en combinación con el IDE *Kdevelop* [4], que facilitó la compilación y generación de Makefiles.

En Windows, la gran mayoría del desarrollo se pudo llevar adelante utilizando las herramientas y el IDE de *Visual C++*, que es parte del paquete *Microsoft Visual Studio 6.0*. Sin embargo, algunas bibliotecas incluidas en el proyecto no han sido pensadas para su uso en Windows y son muy dependientes del compilador GNU, en cuanto a que utilizan notaciones particulares del mismo que el compilador de Visual C++ no puede interpretar (por ejemplo, la forma de embeber código assembler).

En esos casos, la compilación fue posible mediante el uso de *MinGW* [6], una colección de archivos header (.h) específicos de Windows y de bibliotecas de libre distribución, combinados con las herramientas de desarrollo GNU, que permiten producir binarios nativos de Windows que no dependan de entornos de ejecución provistos por terceras partes.

Lamentablemente, no fue posible utilizar MinGW para desarrollar todo el proyecto porque también se utilizan bibliotecas dependientes de Visual C++.

Marco de Trabajo (Framework)

Luego de seleccionado C++ como lenguaje de programación, el objetivo inicial del grupo fue generar código o utilizar bibliotecas de software que pudieran compilarse satisfactoriamente en distintas plataformas. En ese sentido, se evaluaron varios marcos de trabajo que facilitan esta tarea al desarrollador, brindando implícitamente dicha capacidad multiplataforma. Los más importantes, considerando estabilidad, robustez y diversidad de aplicaciones implementadas, fueron *QT* [7], *GTK*[1] [8] y *wxWindows* [9]. Para el proyecto fue seleccionado QT, en base a ciertos criterios que se discuten más adelante.

QT consiste esencialmente en un conjunto de clases que abstraen las funciones del sistema operativo y encapsulan las implementaciones dependientes de la plataforma. Dichas clases son compiladas en una biblioteca dinámica, que las aplicaciones acceden en tiempo de ejecución.

Es una práctica común que el nombre de las aplicaciones que se implementan con QT comience con la letra Q, de ahí el nombre de la aplicación desarrollada: *QPencil Test*.

La selección se basó en:

- plataformas soportadas;
- sencillez de la API;
- riqueza de la API;
- estado de la documentación;
- mecanismo para el procesamiento de eventos;
- existencia de un componente gráfico “grilla” (grid);
- disponibilidad.

Cabe destacar también que *KDE* [10], uno de los entornos de escritorio más famosos de Linux es implementado usando QT. Esto provocó el interés, de los integrantes del grupo de proyecto, de aprender a programar en ese marco de trabajo.

Plataformas soportadas

Los marcos de trabajo debían contar (al menos) con portes estables para Linux y Windows. Tanto QT como wxWindows tienen versiones estables corriendo bajo Linux, varios “sabores” de Unix, Windows 9x/2000/XP y Apple Mac OS X. GTK, por el contrario, fue escrita para Linux y su porte para Windows [11] no solo es bastante reciente, sino que además no goza de buena publicidad. El propio autor advierte en su sitio web que los programas pueden colgarse inesperadamente o comportarse de forma extraña. En el mismo sitio se explica como resolver ciertos problemas y diferencias del porte con respecto a la versión para Linux.

[1] Técnicamente, GTK es una biblioteca para la creación de interfaces gráficas de usuario.

Sencillez de la API

QT y wxWindows brindan una API de desarrollo muy sencilla, intuitiva, orientada a objetos y escrita para C++ (característica fundamental). La API de GTK es mucho más compleja que las anteriores, orientada a objetos pero escrita para lenguaje C (existe una interfaz C++ para GTK llamada gtkmm que no fue evaluada).

Riqueza de la API

En este aspecto QT y wxWindows también son superiores a GTK. Brindan soporte para el manejo de archivos, lectura/escritura de distintos formatos de imagen, interacción con el sistema de impresión, manejo de procesos, etc. e implementan estructuras de datos básicas pero muy útiles como listas, vectores y hashes. QT cuenta además con funciones para el manejo de expresiones regulares y parsers de XML.

En el caso de GTK, es necesario recurrir a varias bibliotecas asociadas para alcanzar las funcionalidades mencionadas antes. Esto significa enfrentar peor o mejor documentación, paradigmas diferentes, inconsistencias entre sus APIs (por ejemplo: la forma en que se representan los strings, el manejo de memoria, etc.).

Estado de la documentación

En los tres casos la documentación es muy buena: existen tutoriales, ejemplos y por supuesto, la descripción completa de las APIs.

Mecanismo para el procesamiento de eventos

En la programación de interfaces gráficas de usuario se consideran dos grandes aspectos: dibujar la interfaz y responder a eventos que sucedan en la interfaz (cierre de una ventana, selección de una opción de menú, clic sobre un botón, etc.). La manera de indicar el código que debe ejecutarse cuando ocurre determinado evento varía según el marco de trabajo.

GTK implementa el manejo de eventos a través de callbacks. Eso implica trabajar con punteros a funciones y la omisión de chequeos de tipos, tanto en tiempo de compilación como de ejecución.

wxWindows por otro lado, utiliza un esquema interno análogo a MFC, llamado tabla de eventos. Se declaran correspondencias entre eventos identificados por constantes y métodos de clases usando una serie de macros bien conocidas. Esta alternativa es más amigable que la de GTK pero todavía guarda cierto grado de complejidad.

QT incorpora una extensión al lenguaje C++ que brinda una sintaxis clara y natural para la programación orientada a eventos. De la misma manera que se definen métodos públicos y privados en las clases, se pueden definir señales (eventos que emite esa clase) y ranuras (código que se ejecuta cuando ocurre algún evento). Solo hace falta declarar qué señales se deben conectar con qué ranuras (y de cuáles objetos). En este caso tampoco hay chequeo de tipos en tiempo de compilación pero sí en ejecución (se muestra un mensaje de advertencia cuando no se puede conectar una señal con una ranura).

El uso de esta extensión al lenguaje hace necesaria una etapa previa de pre-compilación para generar código C++ estándar que pueda ser interpretado correctamente por cualquier compilador. Las herramientas de automatización de la compilación que provee QT hacen transparente esta etapa para el programador (por ejemplo, en Windows se integran con el IDE de Visual C++).

Existencia de un componente gráfico “grilla”

Al momento de realizar la evaluación de marcos de trabajo, wxWindows no contaba con un componente gráfico tabla o grilla. Recién estaba comenzando un proyecto asociado para crear una (llamada wxGrid). La versión actual de wxWindows ya incluye este componente. QT ya contaba con este componente (el cual era muy flexible, permitiendo al programador extenderlo de varias maneras). GTK aún no tiene este tipo de componente.

Aunque este punto pueda resultar muy específico, era esencial contar con una grilla para la interfaz gráfica de la aplicación.

Disponibilidad

Tanto GTK como wxWindows son software libre, liberados bajo licencia GNU LGPL [12] y wxWindows [13] (licencia similar a LGPL) respectivamente. Las versiones más recientes de QT para Windows no son gratuitas, mientras que los portes para Linux y Mac son liberados bajo licencia GNU GPL [14]. La última versión gratuita de QT para Windows es la 2.3.1 (solo para desarrollo no comercial) y funciona exclusivamente con Visual C++.

Formato de Imágenes

Para el manejo de imágenes en memoria se hizo uso del soporte y las clases que brinda QT. La estructura de almacenamiento es sencilla y comúnmente utilizada: una tabla de colores (paleta) y una matriz de puntos o píxels. Para una profundidad de color menor o igual a 8 bits (256 colores), los valores de la matriz corresponden a una entrada en la tabla de colores. Para profundidades mayores, los valores mismos son los colores, codificados como rojo-azul-verde, y la tabla de colores es nula.

Para almacenar las imágenes en disco, el formato elegido fue *Portable Network Graphics* (*PNG*) [15]. Las características que se destacan de este formato en comparación con otros son:

- Es un formato estándar, libre de patentes, ampliamente utilizado y difundido (sobre todo en ambientes Linux y web).
- Se puede obtener en forma gratuita la documentación y la biblioteca *libpng* para el manejo de este formato.
- QT lo soporta como formato interno, brindando operaciones de lectura y escritura que solucionan el problema de diferente orden de bytes para diferentes plataformas (little endian – big endian).
- Utiliza compresión sin perder información en el proceso (como sucede con otros formatos como JPEG).

En este formato también se encuentran las imágenes de los íconos y botones de la aplicación, logrando alcanzar una calidad mucho mayor de la que brinda el formato de íconos estándar de Windows (que solo soporta 256 colores).

Formato de Archivos

Se utiliza XML como formato para los archivos de escenas y de configuración de la aplicación. QT incluye un módulo de XML con una implementación de DOM nivel 2 [16], una interfaz independiente para el acceso y modificación de datos almacenados en dicho formato.

Se utiliza entonces este módulo de QT para interpretar los archivos de escena y cargar los datos en memoria. La Figura 18 muestra una representación gráfica de la estructura válida de estos archivos. La especificación formal del XSD o schema se encuentra en el Manual Técnico.

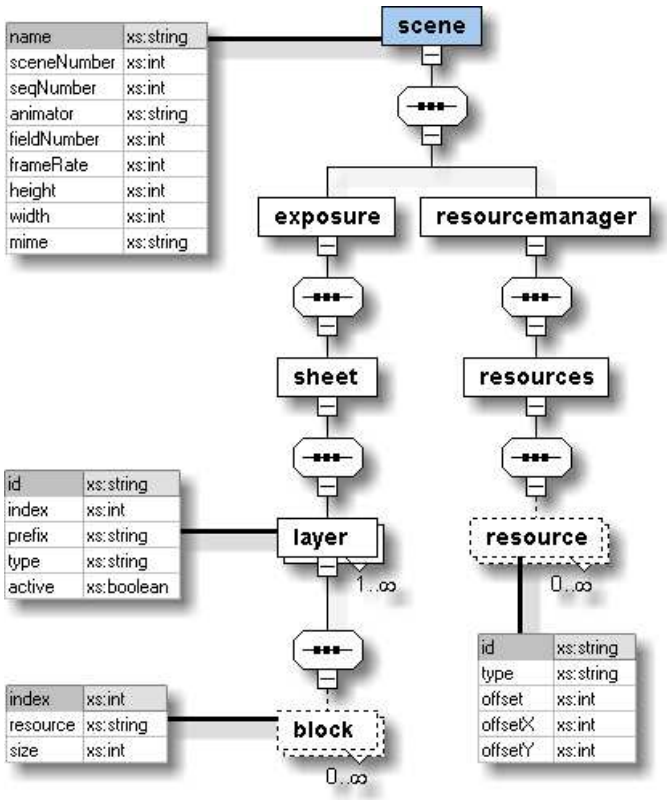

Figura 18 - Estructura de los archivos de escena

El archivo de configuración de la aplicación también está en XML. Básicamente, se dividen las preferencias en grupos y, para cada grupo, se definen parejas clave-valor. La Figura 19 muestra una representación gráfica de la estructura que sigue el archivo. La especificación formal del XSD o schema se encuentra en el Manual Técnico.

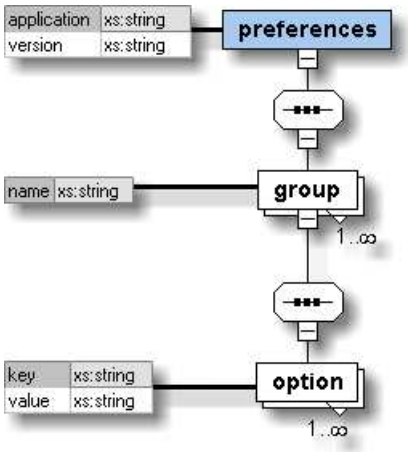

Figura 19 - Estructura del archivo de configuración de la aplicación

Binarios

Se dividió a la aplicación principal en dos binarios: una biblioteca dinámica, denominada *libqpt*, y un ejecutable.

La razón de esta separación fue facilitar el desarrollo de plugins, aislando todas las funciones necesarias en una biblioteca separada del ejecutable principal y favoreciendo el desarrollo y la compilación en paralelo.

Esta división es muy común en el ambiente Linux donde la mayoría de las aplicaciones cuenta con un paquete *devel* para el desarrollo de plugins o add-ons. En Windows no es tan común y solo algunas de las aplicaciones más importantes tienen su *Software Development Kit* (SDK).

El Anexo F explica como compilar la aplicación. Se generaron proyectos para Visual C++ y Makefiles para MinGW y Linux que compilan automáticamente la aplicación, independientemente de la ruta en donde se encuentre ubicada la estructura de directorios del código fuente.

También se generó un instalador de QPT para Windows, utilizando la herramienta *Inno Setup Compiler* [17], de distribución gratuita. La ventaja principal de este instalador con respecto a copiar simplemente los binarios es que asocia las extensiones qpt y qptx para que los archivos de estos tipos se abran automáticamente con “doble clic”.

Ayuda en Línea

El manual de usuario está disponible para poder consultarlo desde dentro de la aplicación y explica uno a uno todos los comandos y diálogos. Se puede ver el índice general mediante un ítem del menú “Ayuda” o abrirlo desde cualquiera de los diálogos importantes de la aplicación, accediendo directamente a la ayuda particular del diálogo.

El manual está escrito en HTML, aprovechando la gran difusión de este formato en cualquier plataforma y la posibilidad de consultarlo mediante un simple navegador web. La aplicación utiliza además un módulo de QT que implementa un modesto visor de páginas HTML y brinda la posibilidad de consultarlo en línea. La Figura 20 muestra este visor y el índice del manual.

Figura 20 - Manual de usuario

Se prestó mucha atención además a escribir *tooltips* y ayudas contextuales eficaces para todos los comandos y barras de herramientas, así como mensajes extra de ayuda en la barra de estado, que se ubica en el borde inferior de la ventana principal.

Detección y Carga de Plugins

Los plugins para QPT son bibliotecas dinámicas que la aplicación detecta y carga al iniciar la ejecución. El mecanismo de detección es sencillo, facilitando así la instalación y desinstalación de plugins sin depender de configuración alguna. Consiste en buscar dentro del directorio de nombre *plugins*, bajo el directorio de instalación de la aplicación, todas las bibliotecas dinámicas que definan las funciones `getPlugin` o `getPlugins`. La última es para darle la posibilidad al desarrollador de incluir varios plugins en la misma biblioteca, devolviendo una lista en lugar de un único plugin.

La carga de bibliotecas dinámicas en tiempo de ejecución requiere distintas implementaciones para distintos sistemas operativos. La versión utilizada de QT no brinda ningún tipo de abstracción al respecto, por lo que fue necesario en este caso recurrir a directivas del precompilador para discernir entre el código a compilar para un sistema operativo u otro. Notar que uno de los principales argumentos para trabajar con plugins es evitar este tipo de técnica pero para usarlos, primero es necesario cargarlos.

Cabe destacar igualmente que este es el único punto en todo el código de la aplicación principal donde se utiliza dicho mecanismo y está encapsulado dentro del módulo de carga de plugins. Sería fácilmente reemplazable en caso de encontrarse una mejor alternativa en otro momento (como la utilización de una nueva versión de QT, por ejemplo).

Persistencia

Se implementaron plugins para guardar escenas mediante dos tipos de almacenamiento distinto: en directorio y en archivo único.

Para el *almacenamiento en directorio*, se considera una escena como un directorio completo del sistema de archivos, que contiene un archivo principal de extensión *qpt* con la información de la hoja de exposición (en formato XML) y todos los archivos de imagen y sonido que correspondan a los recursos.

La ventaja principal de este tipo de almacenamiento es el acceso sin operaciones intermedias a todos los archivos que la aplicación requiera para el manejo de la escena. Se obtiene por consiguiente la mejor performance. Además, la implementación del plugin utiliza solo funciones de QT, por lo que se puede utilizar sin problemas en distintos sistemas operativos.

Por otra parte, los archivos quedan expuestos y es posible que un usuario descuidado corrompa una escena fácilmente.

Otra desventaja desde el punto de vista del usuario es que el manejo de los directorios es poco práctico o intuitivo, sobre todo para mover o copiar escenas. Para solucionar esto surgió el tipo de *almacenamiento de archivo único*, que no es más que el contenido del directorio del tipo anterior, pero agrupado con el formato *tar* y utilizando la extensión *qptx*.

Al abrir una escena de este tipo, la aplicación desagrupa los archivos en un directorio temporal y trabaja con ellos como si se tratara de una escena del tipo de directorio. Al guardar los cambios o cerrar la escena, el contenido de dicho directorio temporal se vuelve a agrupar en un archivo único. Quiere decir entonces que las operaciones que ven afectada su performance son solamente las de abrir, guardar o cerrar escenas. El resto ejecutan exactamente de la misma manera.

Para el manejo de archivos tar se utilizó la biblioteca *libtar* [18]. Para poder compilarla y utilizarla en Windows fue necesario modificar su código y adaptarlo a particularidades propias de dicho sistema operativo como el carácter de separación de rutas de archivos ‘\’ o el número de parámetros que reciben ciertas funciones como `mkdir`. Modificaciones mediante, la biblioteca quedó apta para compilar tanto en Linux como en Windows, por lo que el plugin es utilizable en ambos sistemas operativos.

Se evaluó la posibilidad de aplicar además algún tipo de compresión a los archivos tar. Fue descartada considerando que el formato utilizado para las imágenes (PNG) ya tiene compresión (muy eficiente), por lo que no aportaría una reducción significativa en el tamaño de los archivos y solo serviría para volver más lentas las operaciones. Igualmente, agregar la compresión en otro momento no sería complicado, ya que toda la implementación se encuentra encapsulada en el plugin.

Para hacer la aplicación más amigable al usuario, se desarrolló un diálogo de apertura de archivos que detecta las escenas almacenadas del tipo directorio y muestra directamente un icono de escena en lugar de la clásica “carpeta”.

Más específicamente, el diálogo permite a los plugins de tipo StoreType modificar el listado de archivos o los íconos que muestra. Es decir, que un nuevo plugin que

implemente la persistencia de forma distinta podría personalizar en cierta medida el diálogo y adaptarlo a sus características.

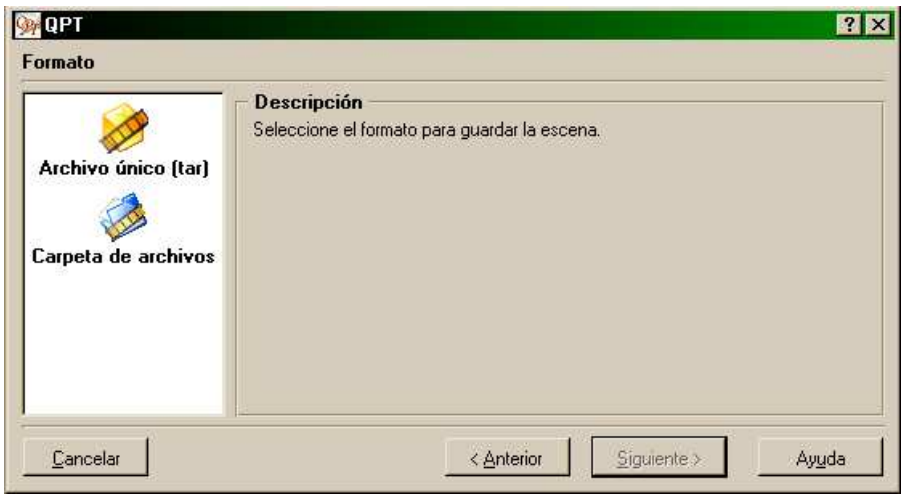

Figura 21 – Página del asistente de creación de escenas donde el usuario selecciona entre los formatos disponibles para guardar la escena.

También se implementó un asistente para la creación de escenas que cuenta con posibilidades de extensión, brindando al usuario una página donde puede elegir entre los formatos disponibles (Figura 21) y luego la página siguiente que el plugin puede implementar de manera personalizada para obtener datos específicos requeridos para dicho formato.

Captura de Imágenes

Video for Windows

Las APIs que ofrecen los distintos sistemas operativos para interactuar con dispositivos de captura son muy diferentes entre sí. Inclusive en un mismo sistema operativo pueden existir diversas opciones.

Tal es el caso de Windows, para el cual se implementó un plugin de captura de imágenes basado en la biblioteca *Avicap* [19], que integra la API de *Video for Windows* (VfW) [20]. Avicap es obsoleta y al momento *DirectX* [21], un conjunto de APIs de bajo nivel para la creación de juegos y aplicaciones multimedia, cubre y extiende todas sus funcionalidades. Sin embargo, los fabricantes continúan liberando al mercado nuevos dispositivos que incluyen controladores compatibles con VfW, y se sigue instalando como API nativa del sistema operativo hasta la versión XP al menos.

En el caso de este proyecto, era casi un requerimiento implícito utilizar Avicap, porque las tarjetas de captura con las que cuenta el equipo de Tournier solo tienen controladores compatibles con esta biblioteca y no con las de DirectX. El modelo de las tarjetas es *Pinnacle DC10*.

La API es poco amigable. El único mecanismo que brinda para interactuar con el proceso de captura es el de *callback*, que consiste en registrar con el sistema operativo una función de cierta firma definida, que será invocada por el sistema mismo cuando suceda un cierto evento.

La función de callback que se implementó es invocada cada vez que el dispositivo envía una imagen capturada al sistema operativo, y recibe como parámetro de entrada un puntero a una cierta área en memoria interna donde quedó almacenada dicha imagen.

Un problema que presenta el hardware antes mencionado es que, al momento de ser invocada la función de callback, la imagen almacenada se encuentra codificada en formato *Motion JPEG* (*MJPG*). Asumimos que esto es para acelerar el proceso de envío de imágenes y favorecer entonces a los programas que intentan mostrar video en tiempo real. Lamentablemente, esta característica no es útil para la aplicación, ya que en general el uso que se le da es para capturar imágenes estáticas.

Además, el uso del formato MJPG es una característica propia de este hardware y no se aplica necesariamente a otro.

Es necesario entonces decodificar la imagen para poder trabajar con ella. Para esto se utilizan funciones propias del Windows, logrando así independizarse de la forma en que se realiza esta tarea.

Si bien se pierde cierto nivel de rendimiento al tener que realizar el proceso de decodificación cada vez que una imagen es capturada pero, se logra una implementación genérica e independiente del hardware sobre el que se aplica (requerimiento importante para el proyecto).

Una ventaja no menor de VfW es que los propios controladores de los dispositivos proveen diálogos de configuración para ajustar todas las propiedades de captura. La API que brinda Windows prevé la existencia de estos diálogos y la posibilidad de invocarlos por programación.

Video for Linux

Para Linux se implementó un plugin basado en *Video for Linux* (V4L) [22], que es la API que provee este sistema para interactuar con dispositivos receptores de radio, televisión, captura de video, etc. Su implementación viene incluida directamente en el *kernel* [23].

La interacción con V4L fue mucho menos complicada que con su contrapartida en Windows (VfW) gracias al uso de *FFMPEG* [24], una biblioteca multimedia muy completa que también se utiliza para el manejo de archivos de video (tema que se trata con mayor profundidad en la sección Lectura/Escritura de Video). Esta biblioteca abstrae la captura de video al punto que para el programador es igual a leer un video desde un archivo (con la salvedad de indicar un dispositivo de captura en lugar de un archivo como origen).

A pesar de esto, la API de FFMPEG no es lo suficientemente amigable como para usarse directamente (preparación de estructuras para abrir archivos o dispositivos, varias invocaciones a funciones, etc.), de manera que se implementó una biblioteca de más alto nivel, denominada *libffmpeg*, que se dispone sobre FFMPEG y se ajusta perfectamente a las necesidades de la aplicación.

Para la configuración de los dispositivos de captura es necesario interactuar directamente con V4L debido a que escapa a la abstracción propuesta por FFMPEG. Cabe destacar que en Linux dichos dispositivos tienen una serie de propiedades comunes como tinte, brillantez, nivel de blancos, tamaño de la

ventana, etc., que V4L permite consultar y modificar. Para acceder a otras propiedades intrínsecas a un modelo en particular de dispositivo se requiere programación a más bajo nivel.

Fue necesario implementar un diálogo de configuración genérico y propio de la aplicación (que se muestra en la Figura 22). Este diálogo permite ajustar las propiedades comunes de los dispositivos de captura (mencionadas anteriormente) y omite propiedades de receptores de televisión (canal, norma, etc.).

Figura 22 - Diálogo de configuración de los dispositivos de captura en Linux

Durante la incursión en la captura de video en Linux se hizo evidente un problema que todavía no ha sido solucionado: la falta de controladores para los distintos dispositivos. Se han hecho grandes avances, hasta el punto que el propio sistema operativo reconoce los dispositivos y carga los controladores necesarios para poder usarlos. Pero de poco sirve esto cuando los fabricantes de dispositivos se limitan a generar y distribuir solamente los controladores para Windows. Luego de varios intentos (fallidos) de conseguir un modelo de cámara web que funcionara en Linux llegamos a una cámara web mexicana (de marca Labtec), compatible con un antiguo modelo de una cámara que sí era soportada.

Filtrado de Imágenes

El objetivo principal del filtrado de imágenes es “limpiar” los dibujos capturados por la cámara. Dado que la iluminación y la lente inciden en la claridad o nitidez de las imágenes capturadas, es conveniente filtrarlas para poder destacar más los dibujos y apreciar mejor el resultado de la animación. Además, generalmente es necesario trabajar con imágenes de fondo totalmente blanco, dado que es el color que la aplicación considera como “transparente” al superponer las imágenes que forman los cuadros de video de la escena.

A modo de ejemplo, la Figura 23 muestra una imagen de prueba antes y después de aplicarle uno de los filtros de la aplicación. Este es en general el resultado deseado.

Notar que en la imagen de prueba (izquierda) se destacan de manera importante algunas manchas que en realidad no se encuentran en el dibujo original de la línea.

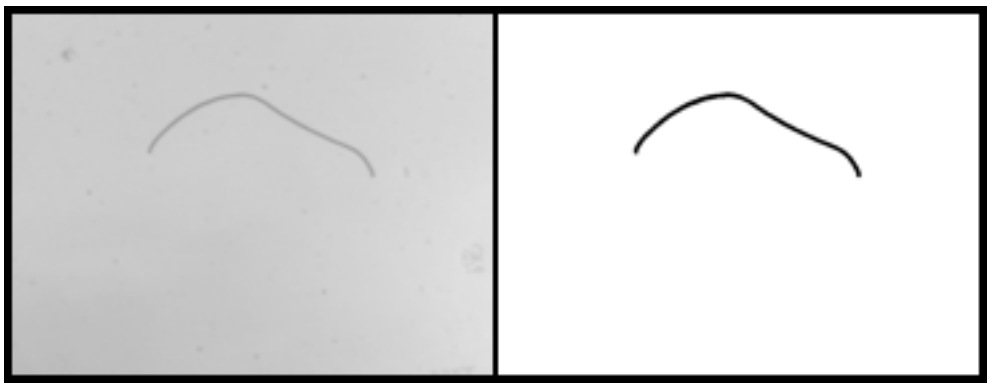

Figura 23 – A la izquierda, imagen de prueba. A la derecha, imagen filtrada.

Se desarrollaron varios filtros para incluir en la aplicación. Algunos de ellos fueron implementados totalmente y otros más complejos utilizan funciones que brindan bibliotecas de tratamiento de imágenes. El Anexo C los define formalmente y trata con mayor profundidad la lógica detrás de cada uno.

El filtro más simple implementado fue una *umbralización fija* (threshold). Parte de una versión en escala de grises de la imagen original y recorre la imagen analizando el nivel de gris de cada píxel. Según sea menor o mayor a un cierto valor umbral, le asigna al píxel el color blanco o negro respectivamente.

Los resultados que se obtienen de este filtro son aceptables, pero no ideales. La mayor desventaja para su uso es que en general los dibujos capturadas por la cámara no se destacan demasiado del fondo (o de ciertas manchas). Es necesario un valor de umbral muy alto para filtrar adecuadamente la imagen y en consecuencia, las líneas quedan bastante más delgadas que los trazos originales en el papel.

Pensando en lograr mejores resultados, se investigaron en profundidad las técnicas de detección de bordes y segmentación de imágenes. Se entiende por borde en una imagen a una zona de píxeles donde existe un cambio brusco de intensidad de luz o nivel de gris. Se implementaron varios filtros de detección de bordes en base a dichas técnicas, entre ellos *Sobel* y *Laplace*, según aparecen en [25] y [26]. Si bien estos filtros detectan bordes de manera muy precisa, los resultados que se obtienen no son los que verdaderamente se requieren. Muchas veces los dibujos cuentan con zonas negras “llenas” que interesa detectar tanto como los bordes.

Se hicieron pruebas con varias bibliotecas gráficas para el tratamiento de imágenes. Las más importantes fueron *ImageMagick* [27], *Gandalf* [28], *MegaWave* [29] y *OpenCV* [30]. Las dos primeras se descartaron rápidamente dado que las funciones que proveen no tienen relación con la segmentación de imágenes o detección de figuras.

MegaWave brinda una implementación del algoritmo *FLLT* [31] que permite la detección de formas en una imagen. Esta biblioteca está escrita en C y fue necesario modificar una pequeña parte de su código para poder utilizar sus funciones con C++.

Se implementó un plugin en base a dicho algoritmo, pero los resultados no fueron suficientemente buenos en la mayoría de los casos. En ocasiones ocurren errores en

la ejecución que no quedan claros y es difícil profundizar en el código de la biblioteca.

Una desventaja importante es que el algoritmo requiere un tiempo de ejecución total bastante alto. Luego de realizar varias pruebas se pudo comprobar que las dimensiones de las imágenes afectan de manera importante dicho tiempo.

Esto motivó a agregar una nueva funcionalidad importante a la aplicación: la vista previa del resultado. Para configurar y ajustar los parámetros de ejecución de cada filtro se creó un diálogo que muestra una versión de la imagen a filtrar a una escala bastante menor. Con esto se logra tener una idea bastante precisa del resultado final del filtro en un tiempo bastante menor.

OpenCV (Open Source Computer Vision), es una biblioteca gráfica orientada principalmente a las áreas de visión por computadora, robótica, interfaz humano-computadora, etc. Las características principales de esta biblioteca son la vasta gama de funciones que brinda y la gran performance con la que ejecutan.

Basándose en varias de sus funciones se desarrollaron los plugins de *umbralización adaptativa*, *canny*, *erosión* y *dilatación*. El primero es una variación de la umbralización simple que separa la imagen en zonas pequeñas y busca un umbral local a cada zona. Es un filtro que brinda en general muy buenos resultados.

Las funciones de erosión y dilatación permiten adelgazar y engrosar las áreas negras de una imagen en blanco y negro, según un cierto patrón o elemento estructural.

Por último, canny es básicamente un filtro detector de bordes que incluye como paso final una umbralización. Su nombre viene de su autor, J. Canny.

Sin embargo, ninguna de estas funciones, o las mencionadas anteriormente, logra proveer en si misma el filtro adecuado para realizar la tarea de detectar los dibujos y limpiar el fondo como es deseado. Por esta razón se implementó una solución que permitiera combinar varios filtros en forma secuencial y así “construir” el filtro adecuado aprovechando las ventajas que cada filtro aporta por separado. Por ejemplo, la combinación de umbralización simple con dilatación produce en general buenos resultados.

Se implementó además un filtro, denominado PSF (Pizzorno – Sierra – Ferré), que reúne varias ideas de los filtros anteriores y se basa fuertemente en el tipo de imágenes que captura la aplicación. Consiste en realizar una umbralización utilizando dos valores de umbral, mínimo y máximo. Los píxeles que superan el máximo son rechazados y se asumen blancos. Los que estén por debajo del mínimo son aceptados y se asumen negros. Aquellos que se encuentren entre ambos límites son aceptados solamente si están conectados a algún punto aceptado anteriormente.

Este filtro asume que el nivel de gris en el centro de una línea dibujada es bastante más fuerte que en los bordes, dado que por ahí pasa el lápiz, y que será detectado como parte del dibujo seguramente con el umbral mínimo. Luego, se consideran también como parte de la línea todos aquellos píxeles que se encuentren “cerca” y cuyo nivel de gris no sea muy distinto que el del centro.

Fue con este filtro que se obtuvo la imagen de la derecha en la Figura 23.

Lectura/Escritura de Video

Al igual que sucede con la captura de imágenes, la lectura y escritura de distintos formatos de video (y audio) se puede implementar de distintas maneras en las diferentes plataformas.

Las alternativas en Windows son VfW y DirectX, mencionadas anteriormente. Tienen como ventajas importantes su fuerte integración con el sistema operativo y la capacidad que brindan a las aplicaciones de instalar nuevas bibliotecas propietarias para extender el conjunto de formatos que soporta el sistema.

Mientras tanto Linux cuenta una plétora de bibliotecas como FFMPEG [24], Avifile [32], Gstreamer [33], etc., ninguna de ellas ligada directamente con el sistema operativo. Tanto así que ciertas aplicaciones de reproducción de video hacen uso de varias de ellas a la vez.

Se planteó entonces la decisión de utilizar bibliotecas (o APIs) diferentes para cada sistema operativo ó una única biblioteca que funcione en ambos. La primer opción implicaba a su vez decidir qué API utilizar en Windows, implementar un plugin de la aplicación que usara esa API, investigar bibliotecas para Linux e implementar otro plugin en base a la biblioteca seleccionada. La segunda opción significaba buscar una única biblioteca que tuviera una buena calificación con respecto a otras (buena performance, buena documentación, soporte para muchos formatos, etc.) e implementar un único plugin en base a la misma. Nos inclinamos por la segunda opción buscando escribir la menor cantidad de código dependiente del sistema operativo (como en general a lo largo de todo el proyecto).

FFMPEG es una biblioteca multiplataforma que resuelve la lectura y escritura de archivos de audio y video, la captura de video desde dispositivos compatibles con V4L y la emisión de video por Internet (estas dos últimas características solo están implementadas para Linux). Está escrita en lenguaje C y ensamblador, con extensiones GNU, por lo que es requerimiento para su compilación contar con las herramientas de desarrollo GNU, en particular con el compilador gcc. Por lo tanto, en Windows debe compilarse usando MinGW [6] (no se exploró la opción de “cross-compiling” desde Linux).

Las razones por las cuales se eligió FFMPEG son:

- Funciona en Linux y Windows. La compilación y posterior uso en Windows requería un mínimo esfuerzo (modificar algunas pocas líneas de código y el Makefile) pero este problema se ha ido solucionado en sucesivas versiones de la biblioteca.
- Cuenta con soporte para una gran cantidad de formatos de archivo y codecs de audio y video. A medida que se liberan nuevas versiones, va mejorando la estabilidad de los formatos y codecs existentes y se van agregando nuevos.
- Los formatos de archivo así como los codecs soportados son parte de la biblioteca, asegurándose la disponibilidad de los mismos en cualquier sistema e independizándose de software de terceras partes. El lado negativo de esta característica es que no hay otra manera de extender las capacidades de la biblioteca que recompilándola, al contrario de lo que sucede en Windows, donde se pueden instalar nuevos formatos con relativa facilidad. Una posible solución a esta debilidad está en implementar un puente entre FFMPEG y alguna de las APIs de Windows.

- Se encuentra optimizada para varias arquitecturas (i386, alpha, ppc), aprovechando las extensiones que ofrecen los procesadores (como MMX y SSE) y el rendimiento que provee el uso de lenguaje ensamblador.
- Como se explicó anteriormente, posee una capa de abstracción para la captura de video en Linux, que facilita el uso de buena parte de la API de V4L.

El proyecto FFMPEG se puede separar en cuatro partes:

- Biblioteca de formatos (libavformat). En esta se incluyen todos los formatos de archivos soportados, la capacidad de emitir video por Internet, la captura de video mediante V4L, lectura y escritura de archivos.
- Biblioteca de codecs (libavcodec). En esta se incluyen todos los codecs soportados, las funciones para convertir entre distintos formatos de imagen (de YUV a RGB, de RGB a BGR, etc.), funciones de recorte y escalamiento de imágenes y funciones de manipulación de audio.
- ffmpeg. Interfaz de línea de comando que permite convertir audio y video. Permite leer de varios orígenes diferentes (dispositivo de captura de video, receptor de radio, archivo de audio, archivo de video, etc.) y generar nuevos archivos de audio y video. En las versiones actuales de FFMPEG existe otra aplicación de línea de comando, ffplay, un reproductor de audio y video.
- ffserver. Servidor de emisión de audio y video a través de Internet.

Una desventaja del proyecto FFMPEG en general es la poca documentación que provee. Recién en las últimas versiones se incluye código de ejemplo con suficientes comentarios como para entenderlo. En un principio no hubo otra opción que aprender leyendo el código de la interfaz de línea de comando que se incluye para entender el funcionamiento de la API. Esto significó un esfuerzo importante.

A partir del estudio del código de la interfaz de línea de comando (ffmpeg) se diseñó e implementó una nueva API de más alto nivel que hace uso de libavformat y libavcodec. Esta API, denominada *libffmpeg*, se presenta en forma de biblioteca dinámica y puede usarse independiente de la aplicación.

La intención principal de implementar libffmpeg fue simplificar la interacción entre la aplicación y FFMPEG, encapsulando chequeos, preparación de estructuras e invocaciones a varias funciones dentro de funciones de mayor nivel como: “abrir archivo de entrada”, “leer”, “cerrar archivo de entrada,” “abrir archivo de salida”, “escribir”, “cerrar archivo de salida”, etc.

Lamentablemente, la falta de documentación hizo muy difícil el trabajo. Además, era de interés mantenerse actualizado con respecto a las nuevas versiones de FFMPEG (debido a las mejoras que incluyeran), lo cual hacía necesario el control de cualquier cambio en las APIs de libavformat y libavcodec.

Reproducción de Video

En un principio, se pensó en implementar totalmente la reproducción de video, es decir, se crearía un reproductor nuevo que interactuaría directamente con el sistema operativo o con los controladores de los dispositivos, sin depender de ninguna biblioteca extra. Surgió entonces un problema muy importante y difícil de resolver: la *sincronización* entre audio y video. Los dispositivos de audio no reproducen el

sonido instantáneamente cuando un programa así lo requiere, sino que introducen un cierto retraso (delay) debido básicamente a cuestiones de hardware. Por lo tanto, una aplicación que reproduzca video debe calcular y controlar este retraso (que varía por dispositivo) para lograr que la imagen y el sonido se sincronicen adecuadamente.

Se descartó entonces la idea inicial estimando que el esfuerzo sería demasiado grande como para cumplir con el alcance del proyecto y se pasó a investigar bibliotecas multimedia o aplicaciones multiplataforma que brindaran las funciones requeridas. Entre ellas SDL [34], Xine [35] y VideoLAN [36]. Sin embargo, ninguna de las opciones evaluadas demostró la estabilidad y madurez necesarias como para poder ser incluidas de forma segura. Sobre todo, se comprobó que la sincronización no era bien manejada en la mayoría de los casos.

Como solución intermedia se implementó un plugin que permitiera ejecutar programas externos a la aplicación y pasarles como parámetro un archivo de video. De esta manera al menos, se podían ver los videos generados en los reproductores multimedia nativos de cada sistema y continuar la verificación de la aplicación. Esta fue la solución final para Linux.

En Windows se llegó a implementar otro plugin. Un reproductor de video interno a la aplicación, utilizando como base las APIs de DirectX, más específicamente, DirectShow. La ventaja más interesante que ofrecía esta tecnología era la posibilidad de reproducir video en pantalla completa.

DirectShow está basado en el Component Object Model (COM) y presenta una arquitectura estructurada en *filtros*. Los filtros son los bloques de construcción básicos que dividen el procesamiento de información multimedia en pasos discretos.

Un conjunto de filtros que trabajan juntos es conocido como un grafo de filtros (filter graph). Los grafos de filtros son manejados por un componente COM llamado Filter Graph Manager, quién entre otras cosas, controla los cambios de estado en el grafo, sincroniza los filtros y permite la comunicación entre el grafo y la aplicación. Filter Graph Manager permite construir fácilmente un grafo capaz de reproducir un archivo de video.

Luego de construido el grafo, la reproducción se maneja accediendo a un conjunto de interfaces COM que también implementa Filter Graph Manager. Las más importantes son `ImediaControl` que controla el flujo de video (iniciar, pausar, detener, etc.) e `IvideoWindow` que establece las propiedades de la ventana de video, como el modo de pantalla completa por ejemplo.

Cabe mencionar que uno de los problemas de utilizar FFMPEG para generar los videos es que los reproductores no siempre cuentan con el codec adecuado para poder reproducirlos. Cuando se utiliza el reproductor interno, los codecs disponibles son los que se encuentran en el sistema y, por lo tanto, basta con instalar el codec requerido. Para el caso de los reproductores externos, existe la posibilidad tanto en Windows como en Linux de instalar aplicaciones o bibliotecas que permitan utilizar directamente los codecs de FFMPEG, solucionando así el problema.

El Manual de Usuario de la aplicación sugiere ciertas combinaciones de formatos de archivo y codecs de uso común.

Monitoreo del Sistema

Las función de monitoreo del sistema, en particular informar la cantidad de memoria disponible en el sistema surge a partir de analizar la interfaz gráfica de CTP.

En un principio resulta útil el hecho de saber cuanta memoria se encuentra disponible, dado que de eso depende muchas veces la performance con que ejecute la aplicación o la calidad con la que se reproduzcan los videos (con poca memoria, los reproductores suelen omitir cuadros).

Un uso lateral de esta función, que fue extremadamente ventajoso en la etapa de desarrollo, es la de verificar que toda la memoria que reserva la aplicación sea liberada en algún momento y no haya pérdidas que traigan como consecuencia el uso ilimitado de la memoria del sistema hasta provocar algunas veces su inestabilidad.

Por supuesto que los datos sobre el estado de la memoria no se consultan de igual forma en los distintos sistemas operativos. Se implementó un plugin para Windows que hace uso de funciones nativas del sistema, las mismas que utiliza por ejemplo el “Administrador de Tareas”. Estas funciones están disponibles en cualquier versión de Windows a la fecha.

En Linux se desarrolló un plugin que hace exactamente lo mismo, pero en base a la función `sysinfo`. Esta función es característica de Linux y no de cualquier sistema Unix disponible.

Conclusiones

La aplicación desarrollada cumple con los requerimientos propuestos al comienzo del proyecto. Es de utilidad real para los usuarios y logra resolver sus problemas técnicos en lo que respecta a la etapa de Pencil Test.

En particular, se destaca que la mayor parte de la implementación es independiente del sistema operativo, gracias al uso de QT y otras bibliotecas multiplataforma.

La separación de plugins plantea además la posibilidad de extender la aplicación, agregando funcionalidades nuevas o mejorando las ya existentes, permitiendo una evolución hacia futuras tecnologías.

La interfaz de programación de plugins brinda al desarrollador gran flexibilidad al momento de diseñar e implementar una extensión.

Se implementó la captura de video de forma que sea independiente del hardware disponible, en base a funciones y APIs nativas de cada sistema operativo.

La interfaz gráfica de la aplicación es muy similar a la de CTP, permitiendo a los usuarios acostumbrarse rápidamente a la misma. A esto se suma además el sistema de ayuda contextual y manual de usuario.

Es notoria la diferencia en la “filosofía” y las posibilidades que brindan al desarrollador los sistemas operativos Windows y Linux. El primero incluye gran cantidad de APIs que resuelven de manera integrada (en general) las necesidades del programador. El segundo provee un núcleo pequeño de operaciones básicas y delega funcionalidades de mayor nivel en bibliotecas que implementan terceras partes, a veces de manera muy diferente entre sí. No es intención tomar posición sobre uno u otro estilo, pero si destacar la gran dificultad que implica en el desarrollo de software multiplataforma que no ejecute sobre una máquina virtual. También es clara la diferencia en cuanto a la disponibilidad de controladores para dispositivos, en particular, para dispositivos de video.

A lo largo del proyecto se aprendieron técnicas para el desarrollo de aplicaciones multiplataforma (identificación y abstracción de diferencias entre sistemas operativos, evaluación de los lenguajes de programación, elección de bibliotecas, alternativas para la arquitectura y diseño de la aplicación, etc.). Cabe destacar la importancia que tuvo el estudio de software de código abierto (open source) como fuente de ideas, tanto para el diseño como para la implementación.

Los integrantes del proyecto profundizaron sus conocimientos en el uso de QT como marco de trabajo. Se obtuvieron resultados más que satisfactorios y se comprobó que los tiempos de desarrollo en C++ se acortan significativamente con las herramientas adecuadas (diseñador de diálogos de interfaz gráfica, generadores de Makefiles, debbuger, etc.) y una buena selección de bibliotecas.

Trabajos Futuros

Soporte para Escáner

Una de las extensiones a la aplicación que se considera importante es la de soporte para escáner, es decir, poder importar imágenes capturadas mediante un escáner como recursos de una escena.

CTP brinda esta funcionalidad, pero quedó fuera del alcance de este proyecto debido a que no es realmente útil para la etapa de Pencil Test. Resulta mucho más práctico y rápido importar imágenes mediante una cámara.

Sin embargo, el soporte para escáner podría ser un comienzo para que la aplicación pase a utilizarse también en las siguientes etapas del proceso de crear una animación.

Es factible agregar esta funcionalidad a la aplicación implementando simplemente un nuevo plugin, simulando de alguna manera la “captura” de imágenes.

Filtros de Imagen

La solución de combinar varios filtros resulta efectiva para “limpiar” las imágenes en la gran mayoría de los casos. Sin embargo, sería más práctico contar con un único filtro que resuelva el problema (aunque esto limita las posibilidades del usuario).

Sería necesario definir de manera formal la función del filtro y profundizar más en las bibliotecas gráficas investigadas, sobre todo MegaWave.

Agregar nuevos filtros de imagen es muy simple, basta con encapsularlos dentro de un plugin.

Movimiento de Cámaras

Otra funcionalidad que quedó fuera del proyecto y resulta muy interesante es la del movimiento de cámaras.

Consiste básicamente en poder definir transformaciones como traslación, rotación y escalamiento a las imágenes de la escena y aplicarlas antes de generar el video.

También podrían aplicarse a las imágenes otras transformaciones que no impliquen necesariamente movimiento, como deformaciones, por ejemplo.

AVEngine para Windows

Si bien FFMPEG cumple con las funcionalidades requeridas para la lectura y escritura de video, y se comporta bien en Windows, sería bueno contar con otro plugin del tipo AVEngine que sea implementado en base a APIs nativas del sistema operativo. En particular, VfW sería una buena opción para aprovechar las ventajas particulares de Windows (capacidad de instalar nuevos formatos de archivos y codecs, y el manejo de archivos de video en memoria).

Se menciona anteriormente otra solución aún mejor que sería implementar algún tipo de “puente” entre FFMPEG y VfW que permitiera sacar ventaja de las características de ambas bibliotecas. Sería interesante acceder a los codecs instalados en Windows para utilizar con VfW o DirectX a través de la API de FFMPEG.

Salvar Automáticamente

Una funcionalidad interesante que surge casi al finalizar el proyecto es la de salvar automáticamente los cambios en una escena cada cierto tiempo configurable, para que el usuario no los pierda en caso que ocurra algún error en la aplicación y se cierre sin darle la posibilidad de guardarlos.

Otra funcionalidad muy ligada a la anterior es la recuperación automática. Es decir, en caso que la aplicación se cierre por error, dar al usuario la siguiente vez que la ejecute la posibilidad de recuperar los cambios realizados a las escenas abiertas hasta la última oportunidad que fueron salvadas automáticamente.

Agregar estas funcionalidades no es trivial y requiere un cambio profundo en la manera que la aplicación maneja las escenas abiertas.

Caché de Recursos

Los recursos, y en particular los recursos de imagen, son almacenados en el disco y accedidos desde allí cada vez que son necesarios.

Una alternativa a este comportamiento sería mantener en memoria un caché de imágenes, que maneje cierta política de control de carga y descarga de las mismas.

El uso de un caché fue considerado dentro del desarrollo del proyecto e incluso se llegaron a realizar pruebas al respecto. Sin embargo, fue descartado debido a problemas de implementación que no valían el gasto de tiempo dado que los tiempos de acceso de las imágenes no eran malos.

Porte para Mac

La plataforma Mac es muy utilizada en el ambiente del desarrollo y diseño gráfico por computadora. Sería interesante investigar y evaluar la factibilidad de poder portar la aplicación a dicha plataforma.

Toda la aplicación principal no debería sufrir inconveniente gracias a QT. Adaptar los plugins hechos para Linux tampoco debería resultar muy difícil. El punto de mayor incertidumbre es la captura de video, debido principalmente a la inexperiencia de los integrantes del proyecto sobre dicha plataforma.

Glosario

- A -

- Add-on. Ver plugin.
- Alpha. (Wikipedia) “El DEC Alpha (también conocido como Alpha AXP) es un microprocesador RISC de 64 bits desarrollado y fabricado por Digital Equipment Corp. (DEC), quien lo usaba en su propia línea de estaciones de trabajo y servidores.”
- Animación. Resultado de construir los cuadros de una escena y exhibirlos durante un cierto intervalo de tiempo, uno a continuación del otro.
- API (Application Programming Interface). Interfaz de programación de la aplicación. Es un conjunto de especificaciones de comunicación (firmas de funciones, estructuras, pre y pos condiciones, protocolos) entre componentes de software.
- Arquitectura i386. Colección de procesadores compatibles con el Intel 80386.
- Arquitectura PPC (PowerPC). (Wikipedia) “PowerPC es una arquitectura de microprocesador RISC creado por la alianza de 1991 Apple-IBM-Motorola, conocida como AIM.”

- B -

- BGR. Ver espacio de colores RGB.
- Bloque. Conjunto de celdas consecutivas que tienen el mismo recurso asignado. Se utiliza para aplicar operaciones sobre todo el conjunto a la vez.
- Borde. Se define borde o frontera de una imagen como una zona de píxeles donde existe un cambio brusco de intensidad de luz o nivel de gris.

- C -

- Caché. Una memoria rápida, fácil de acceder, usada para almacenar un subconjunto de datos que pueden ser caros o lentos de leer o computar. Accesos subsecuentes a los mismos datos pueden referenciar el caché en lugar de obtenerlos nuevamente, disminuyendo así el costo o tiempo promedio de acceso.
- Callback. Esquema usado en programas orientados a eventos donde el programa registra una función (llamada función de callback) para manejar cierto evento. Cuando ocurre el evento la función registrada es invocada.
- Canny. Ver Filtro Canny.

- Celda. Elemento en el que se divide a un nivel. Se le puede asignar una imagen o fragmento de sonido que será utilizado luego para construir un cuadro.
- Codec. (Wikipedia) “Abreviación de codificador/decodificador (coder/encoder), un programa o dispositivo capaz de realizar transformaciones en una corriente de datos o señal. Los codecs pueden poner la corriente de datos o la señal en una forma codificada (para transmisión, almacenamiento o encriptado) y recuperar o decodificar esa forma para visualización o manipulación en un formato más apropiado para esas operaciones.”
- Codec de audio. Programa que comprime y descomprime una corriente de datos de audio digital. Ejemplos de codecs de audio son: MP1, MP2, MP3 (MPEG audio layer 1, 2 y 3), Vorbis, Windows Media Audio 9 Series.
- Codec de video. Programa que comprime y descomprime una corriente de datos de video digital. Ejemplos de codecs de video son: H.263, MPEG-1 Video, MPEG-2 Video, Sorenson, Cinepak, Indeo, DivX.
- COM (Component Object Model). (Wikipedia) “Es la tecnología de Microsoft para componentes de software. Se usa para habilitar la comunicación entre programas (cross-software communication). Su precursor fue OLE (object linking and embedding) y será reemplazado por el Microsoft .NET framework.”
- Cross-compiling. Compilar en una plataforma generando código para ejecutar en otra (ejemplo: usando un compilador que ejecuta en Linux generar código binario que ejecuta en Windows).
- CTP (Cartoon Television Program). Aplicación de la cual se tomaron la mayoría de los requerimientos del proyecto, utilizada por Tournier y su equipo para la etapa de Pencil Test.
- Cuadro. Imagen estática, más un fragmento de sonido, que se exhiben en la animación durante un cierto intervalo de tiempo.

- D -

- Detección de bordes. Proceso de buscar los bordes de una imagen.
- Detector de bordes. Algoritmo que produce un conjunto de bordes a partir de una imagen.
- DOM (Document Object Model). (Wikipedia) “Es una forma de representación de documentos estructurados como un modelo orientado a objetos. DOM es el estándar oficial de W3C (World Wide Web Consortium) para representar documentos estructurados, en una manera independiente del lenguaje y la plataforma.”

- E -

- Escena. Fragmento de animación pequeño donde, por lo general, intervienen los mismos personajes. En el caso de QPT, las escenas son los “documentos” que la aplicación maneja.
- Espacio de colores. (Wikipedia) “Un espacio de colores es una manera específica de representar colores como tuplas de números, típicamente como 3 o 4 valores o componentes de color.”
- Espacio de colores RGB. Es un espacio de colores en el cual el rojo, verde y azul son combinados para obtener otros colores. La abreviación RGB

proviene de Red, Green, Blue (rojo, verde y azul en inglés). Cada color se representa con una tupla (r,g,b) con valores en el intervalo de 0 a 1, siendo r el aporte de rojo, g el aporte de verde y b el aporte de azul para obtener ese color. En general, las imágenes de color verdadero (TrueColor) se almacenan en la memoria RAM como una matriz de puntos, tal que cada punto se define con 3 bytes, uno para el rojo, uno para el verde y otro para el azul (puede haber un byte extra para indicar nivel de transparencia). El orden de los valores (de esos 3 bytes) puede ser rojo-verde-azul (RGB) o azul-verde-rojo (BGR), dependiendo de la biblioteca gráfica utilizada, necesitándose entonces funciones que conviertan un formato en otro para trabajar con otras bibliotecas.

- Espacio de colores YUV. Es un espacio de colores en el cual la Y es el componente de luminancia (brillantez) y U y V son componentes de cromaticidad (color). La ventaja primaria de YUV es que permanece compatible con los televisores blanco y negro. La señal Y es esencialmente la misma señal que se emitiría de una cámara blanco y negro y las señales U y V pueden ser ignoradas. Otra ventaja es que la señal YUV puede ser fácilmente manipulada para descartar algo de información para reducir ancho de banda.
- Exposure Sheet. Ver hoja de exposición

- F -

- Filtro. Cierta función o transformación que se aplica sobre una imagen, dando como resultado otra imagen (eventualmente, idéntica a la original).
- Filtro Canny. Filtro detector de bordes basado en los trabajos de John Canny.
- Filtro de dilatación. Filtro que permite engrosar las áreas negras de una imagen en blanco y negro, según un cierto patrón o elemento estructural.
- Filtro de erosión. Filtro que permite adelgazar las áreas negras de una imagen en blanco y negro, según un cierto patrón o elemento estructural.
- Filtro de umbralización adaptativa. Filtro que transforma una imagen a color o en escala de gris a una imagen binaria (en blanco y negro) aplicando un umbral que varía como una función de las características locales de la imagen.
- Filtro de umbralización fija. Filtro que transforma una imagen a color o en escala de gris a una imagen binaria (en blanco y negro) aplicando un umbral fijo sobre los valores de intensidad de luz. Todos los puntos de intensidad menor al umbral son considerados negros, mientras que el resto son considerados blancos.
- Filtro FLLT. Filtro que utiliza el algoritmo FLLT para identificar formas en una imagen basándose en el nivel de intensidad de luz de sus píxeles.
- Filtro gaussiano. Filtro que permite suavizar una imagen aplicando una función gaussiana a la misma. El suavizado provoca la disminución en las diferencias de intensidad de luz de píxeles contiguos, causando que la imagen resultante parezca más “borrosa” o difusa que la original.
- Filtro Laplace. Detector de bordes que se basa en la búsqueda de cruces por cero en el laplaciano de la función de intensidad de la imagen.
- Filtro Sobel. Detector de borde basado en los operadores de Sobel.
- FLLT (Fast Level Line Transform). Algoritmo que construye un un árbol de formas que obtiene de la descomposición de una imagen según el nivel de gris de sus píxeles.

- Formato de archivo de audio. Formato de archivo para almacenar datos de audio. Ejemplos de formatos de archivo de audio son: Ogg, MP3, WAV.
- Formato de archivo de video. Formato de archivo para almacenar datos de audio, video y otra información auxiliar (información para sincronizar el audio y el video, subtítulos, etc.). Ejemplos de formatos de archivo de video son: AVI, Ogg, ASF, WMV, MP4, QuickTime (MOV), RealMedia (RM).
- Formato de archivo TAR (Tape ARchive format). Es un formato de archivo usado para acumular grandes cantidades de archivos en un solo archivo preservando usuario y grupo, permisos, fecha y estructura de directorios.
- Frame. Ver Cuadro.

- G -

- GCC (GNU Compiler Collection). Originalmente GCC correspondía a GNU C Compiler pero ahora maneja muchos otros lenguajes de programación además de C.
- GNU. Acrónimo recursivo para “GNU's Not Unix”. Proyecto lanzado por Richard Stallman con el objetivo de crear un sistema operativo completamente gratuito.
- GNU GPL (GNU General Public License). Es una licencia de software libre que permite la modificación, mejora y redistribución de los trabajos realizados (bajo esta licencia) y todos sus derivados siempre y cuando estas copias carguen con la misma licencia y se hagan disponibles de una manera que facilite su modificación (se debe ofrecer el código fuente a todo aquel que haya recibido una copia del trabajo o un derivado). El texto de la licencia está disponible en Internet [14].
- GNU LGPL (GNU Library General Public License o Lesser General Public License). Licencia con menos restricciones que la GNU GPL. Mientras que una biblioteca bajo licencia LGPL puede ser usada en programas propietarios, las bibliotecas bajo GPL solo están disponibles para programas libres. El texto de la licencia está disponible en Internet [14].
- GPL. Ver GNU GPL.
- Gradiente. Equivalente bidimensional de la primera derivada.
- GTK (GIMP toolkit). Inicialmente creado para el programa GIMP (General Image Manipulation Program), se trata de una biblioteca para crear interfaces gráficas de usuario. Junto a QT es una de las dos bibliotecas más populares en Linux.
- GUI (Graphic User Interface). Interfaz gráfica de usuario.

- H -

- Histéresis. Último paso del detector de bordes de Canny. Consiste en una umbralización donde se fijan un umbral superior y otro inferior. Los puntos cuyo valor sea mayor que el umbral superior son aceptados. Los puntos cuyo valor sea menor que el umbral inferior son rechazados. Los puntos entre los dos valores son aceptados siempre y cuando estén conectados a un punto previamente aceptado.
- Hoja de Exposición. Planilla que contiene y organiza toda la información necesaria para generar una animación o video a partir de una escena. Cada fila de la planilla representa un cuadro y cada columna un nivel.

- I -

- i386. Ver arquitectura i386.
- IDE (Integrated Development Environment). (Wikipedia) “Un ambiente integrado de desarrollo (Integrated Development Environment o IDE) es un programa de computadora que consiste de un editor de texto, un compilador, un intérprete, herramientas de automatización y un depurador. Aunque existen varios ambientes de desarrollo multi-lenguaje, por lo general son dedicados a un lenguaje de programación específico. En ocasiones son incluídos también un sistema de control de versión y herramientas para la construcción de interfaces gráficas.”
- Interacción humano-computadora. Es el estudio de las interacciones entre las personas y las computadoras. Es un campo interdisciplinario que relaciona ciencia de la computación, psicología, sociología, ciencia de la información entre otros. Uno de sus objetivos es hacer que las computadoras sean más amigables para el usuario.
- Interfaz humano-computadora. Ver interacción humano-computadora.

- J -

- JPEG (Joint Photographic Experts Group). Es un método estándar para comprimir imágenes fotográficas. El formato de archivo que emplea esta compresión también es llamado JPEG (archivos de extensiones .JPG, .JPE).

- K -

- KDE (K Desktop Environment). KDE es un entorno de escritorio gráfico e infraestructura de desarrollo construido con el QT toolkit para sistemas Linux y Unix. Uno de sus objetivos principales es llenar la necesidad de un escritorio fácil de usar para las estaciones de trabajo Unix, similar a los entornos de escritorio de MacOs y Windows.
- Kernel. (Wikipedia) “El kernel (núcleo) es aquella parte de un sistema operativo que interactúa de forma directa con el hardware de una máquina. Entre las funciones principales del kernel se encuentran: la gestión de memoria; administración del sistema de archivos; administración de servicios y dispositivos de entrada/salida y asignación de recursos entre los usuarios. El software puede comunicarse con el kernel por medio de llamadas al sistema (system calls), las cuales le indican al kernel que realice tareas como abrir y escribir un archivo, ejecutar un programa, finalizar un proceso u obtener la fecha y hora del sistema.”

- L -

- Laplace. Ver filtro Laplace.
- Laplaciano. Equivalente bidimensional de la segunda derivada.
- Layer. Ver Nivel.
- LGPL. Ver GNU LGPL.

- M -

- MDI (Multiple Document Interface). Las aplicaciones gráficas con interfaz MDI son aquellas cuyas ventanas residen bajo una única ventana “padre” (a excepción de las ventanas modales).
- MFC (Microsoft Foundation Classes). Biblioteca de clases C++ que encapsula la API de Windows.
- MJPEG (Motion JPEG). Es un codec de video que comprime separadamente cada cuadro en una imagen JPEG.
- MMX (Multimedia Extensions). Es un conjunto de instrucciones SIMD diseñado por Intel e introducido en sus microprocesadores Pentium MMX.
- MVC. Patrón Model-View-Controller.

- N -

- Nivel. Capa que contiene información multimedia parcial de la escena. Parcial en el sentido que tiene imágenes correspondientes al movimiento de un único personaje, o solamente el fondo, y es necesario superponer varios niveles para lograr la animación completa de la escena.

- P -

- Pencil Test. Nombre con el cual CTP denomina a la etapa de verificación de la continuidad de los movimientos de los personajes y la evaluación de los tiempos de exposición de las imágenes para sincronizarlas con el audio.
- Plugin. (Wikipedia) “Un plugin es un programa de computadora que interactúa con otro programa para proveer cierta funcionalidad. Ejemplos típicos son plugins para reproducir archivos multimedia, para encriptar/desencriptar correo electrónico, o para filtrar imágenes en programas gráficos. El programa principal establece un estándar para el intercambio de datos con el plugin, le pasa los datos al plugin y actúa sobre los resultados del procesamiento del plugin.”
- PNG (Portable Network Graphics). Es un formato de almacenamiento de imagen basado en un algoritmo de compresión sin pérdida (de información) diseñado originalmente para sustituir al formato GIF. A diferencia de GIF que permite hasta 256 colores y un grado de transparencia el formato PNG permite imágenes en color verdadero (TrueColor), escala de grises, paleta de colores de 8 bits y 256 grados de transparencia.
- PPC. Ver arquitectura PPC.

- Q -

- QPT (QPencil Test). Nombre de la aplicación desarrollada en el marco del proyecto.
- QT. QT es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario, producida por la compañía noruega Trolltech. Como características no gráficas se encuentra soporte para internacionalización, acceso a bases de datos SQL, lectura/escritura de XML y una API independiente de la plataforma para manejo de archivos.

- R -

- Recurso. Ver Recurso Multimedia.
- Recurso Multimedia. Archivo de imagen o audio que conforma las escenas.
- RGB. Ver espacio de colores RGB.
- RISC (Reduced Instruction Set Computing). Es una filosofía de diseño de CPUs que favorece un conjunto simple y pequeño de instrucciones que toman el mismo tiempo en ejecutar.

- S -

- SDK (Software Development Kit). (Wikipedia) “Es un conjunto de herramientas de desarrollo que permiten crear aplicaciones para cierto paquete de software, marco de trabajo, plataforma de hardware, sistema de computadora, sistema operativo o similar.”
- SGML (Standard Generalized Markup Language). SGML es un metalenguaje con el cual se pueden definir lenguajes de marcación ("markup languages") para documentos. Provee de una variedad de sintaxis de marcación que pueden ser usadas por muchas aplicaciones. Ejemplo de la sintaxis de SGML es:

  <QUOTE TYPE="sample">un <ITALICS>ejemplo</ITALICS></QUOTE>

- SIMD (Single Instruction, Multiple Data). Refiere a un conjunto de operaciones para manejo eficiente de grandes cantidades de datos en paralelo. Un ejemplo de aplicación que puede tomar ventaja de SIMD es cuando se quiere sumar el mismo número a varios valores en memoria (en general, operaciones en procesamiento de imágenes). Con un procesador SIMD hay dos mejoras, primero, se entiende que los datos están en bloques pudiendo cargar varios valores al mismo tiempo y segundo, se aplica la operación sobre todos los valores cargados al mismo tiempo.
- Segmentación de imágenes. Proceso que consiste en dividir una imagen digital en regiones homogéneas con respecto a una o más características (ejemplo: por brillo o color) con el fin de facilitar su posterior análisis y reconocimiento.
- Sobel. Ver filtro Sobel.
- SSE (Streaming SIMD Extensions). Es un conjunto de instrucciones SIMD diseñado por Intel e introducido en su serie de procesadores Pentium III como respuesta al 3DNow! de AMD. SSE solucionaba una serie de problemas que tenía el conjunto de instrucciones MMX. SSE2 es una extensión al conjunto de instrucciones de SSE introducida en los procesadores Pentium 4.
- Stop Motion. Técnica que consiste en animar un objeto a través de una secuencia de imágenes que son capturadas por una cámara, de manera que al reproducirlas luego en una sucesión temporal se obtenga la ilusión de un movimiento continuo. Esta técnica puede aplicarse también con dibujos (o imágenes generadas de alguna forma) para realizar caricaturas animadas.

- T -

- TAR. Ver formato de archivo TAR.

- Threshold. Ver filtro de umbralización fija y filtro de umbralización adaptativa.
- Tooltip. Mecanismo de ayuda de una aplicación que consiste en una descripción breve de un comando o acción del programa. Dicha descripción aparece cuando el usuario deja el puntero del ratón un breve período de tiempo sobre el elemento gráfico asociado a dicho comando o acción (ejemplo: un botón, un icono de una barra de herramientas, etc.).

- U -

- Umbralización adaptativa. Ver filtro de umbralización adaptativa.
- Umbralización fija. Ver filtro de umbralización fija.

- V -

- Velocidad de reproducción. Propiedad de una escena que indica la cantidad de cuadros que deben exhibirse al observador por segundo.
- V4L (Video For Linux). API provista por el sistema operativo Linux para interactuar con dispositivos receptores de radio, televisión, captura de video, etc. Su implementación viene incluida directamente en el kernel.
- VfW (Video for Windows). API provista por el sistema operativo Windows para interactuar con dispositivos receptores de televisión, captura de video. Además, lectura/escritura de archivos de audio y video y su reproducción.
- Visión por computadora (Computer vision). (Wikipedia) “Es un área dentro del campo Inteligencia Artificial. Su propósito es programar una computadora que entienda una escena o las figuras de una imagen. Objetivos típicos de la visión por computadora son: detección, ubicación y reconocimiento de ciertos objetos en las imágenes (ejemplo: rostros de humanos); registro de diferentes vistas de la misma escena u objeto; seguimiento de un objeto durante una secuencia de imágenes; definición de un modelo tridimensional de una escena.”

- X -

- XML (eXtensible Markup Language). El lenguaje extensible de marcas (XML) es un subconjunto simplificado de SGML (ver SGML), capaz de describir muchas clases diferentes de datos. Su principal propósito es facilitar el intercambio de texto e información estructurada a través de Internet. Algunas de sus ventajas para transferencia de datos son: compatibilidad con protocolos de Internet; formato legible para humanos y computadoras; habilidad para representar listas, registros y árboles; formato auto-documentado (se describe a sí mismo).
- XSD (XML Schema). Describe la estructura de un documento XML, definiendo el formato de los bloques de construcción legales (elementos y atributos aceptados, qué elementos son hijos de otros elementos, orden y cardinalidad de los elementos hijos, etc).

- Y -

- YUV. Ver espacio de colores YUV.

Referencias[2]

[1] Cartoon Television Program
<http://www.cratersoftware.com/ctp_pro.html>

[2] Buschmann, Meunier, Rohnert, Sommerland, Stal; *Pattern-Oriented Software Architecture (A System of Patterns)*; John Wiley & Sons; 1996

[3] Gamma, Helm, Johnson, Vlissides; *Design Patterns*; Addison Wesley; 1998

[4] GCC – GNU Compiler Collection
<http://gcc.gnu.org/>

[5] The KDevelop Project
<http://www.kdevelop.org>

[6] MinGW – Minimalist GNU for Windows
<http://www.mingw.org>

[7] Trolltech – QT
<http://www.trolltech.com/products/qt/index.html>

[8] GTK – The Gimp ToolKit
<http://www.gtk.org>

[9] wxWindows – Cross-Platform GUI Library
<http://www.wxwindows.org>

[10] KDE – K Desktop Environment
<http://www.kde.org>

[11] Tor Lillqvist--GTK+ and GIMP for Windows
<http://www.gimp.org/~tml/gimp/win32/>

[12] Licencia GNU LGPL
<http://www.gnu.org/copyleft/lesser.html>

[13] Open Source Initiative OSI – The wxWindows Library License
<http://www.opensource.org/licenses/wxwindows.php>

[14] Licencia GNU GPL
<http://www.fsf.org/licenses/gpl.html>

[2] Todas las direcciones web se encuentran disponibles al 28 de febrero del 2004.

[15] PNG – Portable Network Graphics
<http://www.libpng.org/pub/png/>

[16] DOM – Document Object Model
<http://www.w3.org/DOM/>

[17] Inno Setup Compiler
<http://www.innosetup.com>

[18] LibTar – Tar File Manipulation API
<http://www-dev.cites.uiuc.edu/libtar/>

[19] Avicap
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/multimed/htm/_win32_video_capture.asp>

[20] Video for Windows
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/multimed/htm/_win32_video_for_windows.asp>

[21] DirectX
<http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000410>

[22] Video for Linux
<http://kernelbook.sourceforge.net/videobook.html>

[23] The Linux Kernel
<http://www.kernel.org>

[24] FFMPEG – Multimedia System
<http://ffmpeg.sourceforge.net/>

[25] Gonzalez, Woods; *Digital Image Processing*; Addison Wesley; 1992

[26] Russ; *The Image Processing Handbook*; IEEE Press; 2ª Edición; 1994

[27] ImageMagick
<http://www.imagemagick.org>

[28] Gandalf – Computer Vision and Numerical Algorithm Library
<http://gandalf-library.sourceforge.net/>

[29] MegaWave 2
<http://www.cmla.ens-cachan.fr/Cmla/Megawave/>

[30] OpenCV – Open Source Computer Vision Library
<http://www.intel.com/research/mrl/research/opencv/>

[31] Fast Computation of a Contrast Invariant Image Representation
P. Monasse and F. Guichard, *IEEE Transactions on Image Processing*, Vol. **9**, No. 5, pp. 860-872, May 2000.

[32] Avifile – Linux AVI file library
<http://avifile.sourceforge.net>

[33] GStreamer – Open Source Multimedia Framework
<http://freedesktop.org/~gstreamer/>

[34] SDL – Simple DirectMedia Layer
<http://www.libsdl.org>

[35] Xine – Free Multimedia Player
<http://www.xinehq.de>

[36] VideoLAN
<http://www.videolan.org>

Anexo A – Requerimientos

Requerimientos Funcionales

Manejo de Escenas

R 1. Crear escenas con las siguientes opciones para formatos de salida:

- PAL – 768 x 576 píxeles, a 25 cuadros por segundo.
- NTSC – 720 x 540 píxeles, a 30 cuadros por segundo.
- VGA – 640 x 480 píxeles, a 25 cuadros por segundo.
- Personalizable – el usuario especifica los valores de alto, ancho y velocidad.

Seleccionar en la creación el formato de almacenamiento de la escena (*ver R 3*).

R 2. Crear una hoja de exposición por defecto para las escenas nuevas, que incluya los niveles “OVR”, “A”, “B”, “C” y “BGR” del tipo imagen y “SND1” del tipo sonido, todos inactivos. El número inicial de cuadros es 50 y se incrementa automáticamente a medida que es necesario.

R 3. Guardar en disco escenas creadas o modificadas. Brindar al menos dos formatos de almacenamiento:

- Directorio: toda la información de la escena y los recursos que la forman se almacenan dentro de un único directorio del sistema de archivos.
- Archivo único: toda la información de la escena y los recursos que la forman se almacenan dentro de un único archivo.

La elección del formato de almacenamiento de una escena se realiza al momento de creación. Al guardar se utiliza el mismo formato.

R 4. Guardar las modificaciones a una escena como otra escena, con otro nombre y formato de almacenamiento (“Guardar como”).

R 5. Guardar los cambios de todas las escenas abiertas al mismo tiempo.

R 6. Abrir escenas guardadas previamente para realizar modificaciones. Es posible tener varias escenas abiertas en un momento dado, pero solo una activa.

R 7. Mantener una lista de las últimas escenas abiertas para acceder a ellas rápidamente.

R 8. Cerrar cualquier escena abierta o cerrarlas todas a la vez. Consultar al usuario, de ser necesario, si quiere guardar las modificaciones en las escenas antes de cerrarlas.

R 9. Agregar a una escena nuevos niveles, en cualquier posición. Sugerir por defecto la última posición y verificar que el nombre sea único para la escena.

R 10. Mover un nivel, cambiando su posición.

R 11. Eliminar cualquier nivel de la escena, verificando que quede al menos una.

R 12. Activar y desactivar cualquier nivel.

R 13. Detectar la formación de bloques y marcar claramente en la interfaz el comienzo y final de cada uno.

R 14. Aumentar o disminuir la cantidad de cuadros que forman un bloque. Brindar funciones de rápido acceso para diferencias de 1, 5, 10 y 25 bloques.

Manejo de Recursos

R 15. Adquirir imágenes desde dispositivos de video conectados a la computadora.

Permitir seleccionar el dispositivo entre todos los que se encuentren instalados en el sistema.

R 16. Adquirir imágenes almacenadas como archivos de imagen en el disco.

Permitir adquirir series de imágenes. Se consideran parte de una serie a todas aquellas imágenes que tengan igual nombre a excepción de un número.

R 17. Permitir filtrar las imágenes al momento de adquirirlas para obtener una imagen con los trazos dibujados en negro y el fondo en blanco.

R 18. Adquirir la pista de audio almacenada en archivos de audio o video.

R 19. Permitir referenciar los recursos desde un cuadro en el mismo momento de adquirirlos. El usuario selecciona una capa donde ingresar el recurso y la manera de ingresarlo:

- Directamente al final, referenciándolo desde el último cuadro disponible.
- Insertándolo en un cuadro específico, identificado por su capa e índice.

R 20. Eliminar todos los recursos de una escena que no se encuentren referenciados por algún cuadro al cerrar la escena o la aplicación.

Funciones Misceláneas

R 21. Copiar, cortar y pegar una celda o un conjunto de celdas contiguas entre escenas abiertas.

Copiar una celda y pegarla en otra significa asignar a la segunda el mismo recurso que la primera. En caso de que sean escenas distintas, se agrega además el recurso. Cortar una celda equivale a eliminar su asignación a un recurso.

En el caso de las celdas de imagen, copiar o cortar afectan únicamente a la celda, mientras que para las de sonido, afecta a todo el bloque.

Evitar que se peguen celdas sobre niveles que sean de tipo distinto.

R 22. Permitir arrastrar y soltar cuadros entre escenas abiertas, correspondientes a las funciones cortar y pegar respectivamente.

R 23. Deshacer y rehacer las acciones del usuario sobre la hoja de exposición de las escenas. Las acciones se registran de forma independiente para cada escena. La cantidad de acciones que se pueden deshacer es configurable.

Información en Pantalla

R 24. En todo momento existe un cuadro seleccionado por el usuario, denominado “cuadro actual”. Mostrar en pantalla, si corresponde, la imagen que referencia el cuadro actual.

R 25. Mostrar en pantalla el tiempo en segundos desde el comienzo de la escena hasta el cuadro actual.

R 26. Al seleccionar dos o más cuadros de igual índice pero de distintas capas, mostrar en pantalla la superposición de todos los cuadros seleccionados.

Generación de Video

R 27. Generar un video en base a toda la información de las capas activas de una escena y reproducirla según lo indiquen las características del formato de salida de la escena.

R 28. Generar un video solo en base a la información de ciertos cuadros seleccionados de una escena.

R 29. Permitir que el video generado puedan verse en pantalla completa.

R 30. Permitir la reducción de las imágenes de los cuadros antes de generar los cuadros para que resulten videos de menor tamaño.

R 31. Exportar la animación generada por la escena como un archivo de video.

Impresión

R 32. Imprimir la hoja de exposición. Brindar la opción de imprimir solo las capas activas.

R 33. Imprimir la imagen asignada a una celda.

Requerimientos no Funcionales

Interfaz Gráfica

R 34. La interfaz gráfica debe ser semejante a la de CTP para favorecer el rápido aprendizaje de la aplicación.

Independencia del Hardware

R 35. La aplicación debe ser independiente del hardware de captura de video (no está limitada a funcionar con un solo tipo de cámara).

Sistema Operativo

R 36. Ejecutar al menos en el sistema operativo Windows.

Debe ser fácil portar, compilar y ejecutar en otros sistemas operativos.

Manual de Usuario

R 37. La aplicación debe contar con manual de usuario.

Anexo B – Utilización de Plugins

Las decisión de utilizar este mecanismo estuvo fuertemente basada en el requerimiento de ejecución en varios sistemas operativos. Al comienzo del proyecto se intentó satisfacer este requerimiento utilizando funciones y bibliotecas que también lo cumplieran.

Ya avanzada la implementación, se podían identificar en el código varias secciones como la siguiente:

```
#ifdef WIN32
   #include win32depclass.h
   ...
#else
   #include linuxdepclass.h
   ...
#endif
...
#ifdef LINUX
   ...
#else
   ...
#endif
```

Lo anterior resulta inconveniente por varias razones. Primero, obliga a escribir código "dummy" o funciones de cuerpos vacíos que permitan la compilación de la aplicación en un sistema operativo donde aún no se han implementado totalmente ciertas funcionalidades específicas.

Segundo, si bien es posible acostumbrarse a este estilo de programación, en ciertos casos resulta complicado de leer o implementar y hace más difícil la detección de errores en el código.

Por último, ata la implementación a los sistemas operativos considerados. Un porte a otra plataforma obliga a la tediosa tarea de modificar gran cantidad de código fuente, con el riesgo además de introducir nuevos errores en el proceso.

El mecanismo de plugins logra solucionar los problemas mencionados en la mayoría de los casos, haciendo más legible el código y facilitando mucho la compilación del programa. Favorece además el desarrollo en paralelo de plugins para distintos sistemas operativos, mejorando en gran medida la productividad.

Por otro lado, permite implementar un mismo plugin de distintas maneras o con diferentes tecnologías, que pueden intercambiarse y compararse a fin de sacar conclusiones, como por ejemplo cual brinda mayor performance.

Igualmente, la gran ventaja de los plugins es la extensibilidad.

Anexo C – Filtros de Imagen

Umbralización

El más sencillo de los filtros implementados es conocido como *umbralización fija* (threshold). En su forma más simple, la imagen binaria resultante $B(i, j)$ se define a partir de la versión en escala de grises de la imagen digital original $I(i,j)$, en función de un valor U que corresponde al umbral de separación seleccionado.

$$B(i, j) = \begin{cases} 0, & \text{si } I(i, j) \geq U \\ 1, & \text{si } I(i, j) < U \end{cases}$$

En la fórmula anterior 0 indica el color blanco y 1 el color negro.

El filtro de umbralización fija desarrollado para QPT agrega un segundo valor de umbral, de manera que se consideran blancos aquellos píxeles cuyo nivel de gris se encuentra dentro del rango definido por los dos valores de umbral U_1 y U_2.

$$B(i, j) = \begin{cases} 0, & \text{si } U_1 \leq I(i, j) \leq U_2 \\ 1, & \text{en otro caso} \end{cases}$$

Umbralización Adaptativa

En general, la utilización de un único valor de umbral fijo no es aplicable en imágenes complejas. Esto lleva a considerar otros tipos de umbralización, que resultan de generalizar la idea de umbral explicada anteriormente.

La *umbralización adaptativa* (adaptive threshold) permite resolver el problema de la segmentación en imágenes complejas haciendo que el valor del umbral varíe en función de las características locales de la imagen. En otras palabras, dividir la imagen en regiones pequeñas y calcular un umbral local para cada una.

Se implementó un plugin que utiliza la función de umbralización adaptativa `cvAdaptiveThreshold` de la biblioteca OpenCV. Dicha función calcula

$$B(i, j) = \begin{cases} 0, & \text{si } I(i, j) \geq U(i, j) \\ 1, & \text{si } I(i, j) < U(i, j) \end{cases}$$

donde $U(i, j)$ es la diferencia entre la media de nivel de gris de los píxeles en un entorno de (i, j) de tamaño $p \times p$ y un cierto valor v. Tanto p como v son parámetros de la función.

Sobel

Se puede definir borde o frontera en una imagen como una zona de píxeles donde existe un cambio brusco de intensidad de luz o nivel de gris. Si se toma una imagen como una función continua *f*(*x*,*y*) de intensidad de luz, se verá que su derivada tiene un máximo local en la dirección del borde. Es por esto que las técnicas más usadas de detección se basan justamente en la medida del gradiente de *f* a lo largo de *r* en una dirección θ:

$$\frac{\partial f}{\partial r} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial r} = f_x \cos\theta + f_y \sin\theta$$

El valor máximo de $\frac{\partial f}{\partial r}$ se obtiene cuando $\frac{\partial}{\partial \theta}\frac{\partial f}{\partial r} = 0$

$-f_x \sin\theta + f_y \cos\theta = 0 \Rightarrow \theta = \tan^{-1}\frac{f_y}{f_x}$ y por lo tanto $\left(\frac{\partial f}{\partial r}\right)_{max} = \sqrt{f_x^2 + f_y^2}$

En el tratamiento de imágenes se trabaja con píxeles y en un ambiente discreto. En base a los conceptos vistos anteriormente, se han ideado varios operadores detectores de bordes basados en máscaras, que representan aproximaciones en diferencias finitas de los gradientes ortogonales f_x y f_y .

Se define máscara *H* como una matriz de $p \times p$, y su producto interno con una imagen *U*, en un píxel o posición (*m*,*n*) como:

$$\langle U, H \rangle_{m,n} = \sum_{i=1}^{p}\sum_{j=1}^{p} h(i,j) \cdot u(i+m, j+n)$$

La forma habitual de establecer el gradiente de una imagen en un punto dado es mediante el producto de la imagen por dos máscaras *H1* y *H2* que representen la magnitud del gradiente en dos direcciones perpendiculares:

$$\left.\begin{aligned} g_1(i,j) &= \langle U, H_1 \rangle_{i,j} \\ g_2(i,j) &= \langle U, H_2 \rangle_{i,j} \end{aligned}\right\} \Rightarrow g(i,j) = \sqrt{g_1(i,j)^2 + g_2(i,j)^2}$$

Existe una serie de máscaras de uso muy frecuente en análisis de imágenes, entre ellas las de ***Sobel***, que se muestran a continuación:

$$H_1 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad H_2 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Son las utilizadas para implementar el plugin homónimo de QPT.

Considerando una región de la imagen como muestra la siguiente:

$$\begin{bmatrix} z_1 & z_2 & z_3 \\ z_4 & z_5 & z_6 \\ z_7 & z_8 & z_9 \end{bmatrix}$$

donde las *z* representan el nivel de gris del píxel donde se superpone la máscara y z_5 en particular al de la posición (*i*, *j*), las operaciones para aproximar las derivadas a partir máscaras de Sobel son:

$$g_1(i, j) = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

$$g_2(i, j) = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

El píxel de la posición z_5 es declarado como perteneciente a un borde cuando $g(i, j)$ excede un determinado valor umbral, que se toma como parámetro de la operación, permitiendo variar la sensibilidad de la detección.

Laplace

Los detectores de bordes como *Sobel* que calculan el gradiente dan como resultado por lo general demasiados puntos. Una mejor aproximación es considerar como puntos de borde solo aquellos en donde existe un máximo local en el gradiente, es decir, aquellos en los que existe un máximo local en la primera derivada y por lo tanto un cruce por cero en la segunda. Este principio es utilizado por el detector de bordes de *Laplace* o *Laplaciano.*

El laplaciano de una función bidimensional *f*(*x*, *y*) es una derivada de segundo orden definida por:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

La segunda derivada en la dirección *x* se puede aproximar utilizando la ecuación en diferencias (para $h = 1$):

$$\frac{\partial^2 f}{\partial x^2} = \frac{G_x}{\partial x^2} \approx \frac{\partial(f(i+1, j) - f(i, j))}{\partial x} \approx f(i + 2, j) - 2f(i + 1, j) + f(i, j)$$

Análogamente para *y* se obtiene:

$$\frac{\partial^2 f}{\partial y^2} = f(i, j+2) - 2f(i, j+1) + f(i, j)$$

El problema con estas dos aproximaciones es que no están centradas en la posición (*i*, *j*). Otra aproximación que si lo está es:

$$\frac{\partial^2 f}{\partial x^2} = f(i+1,j) - 2f(i, j) + f(i-1, j)$$

$$\frac{\partial^2 f}{\partial y^2} = f(i, j+1) - 2f(i, j) + f(i, j-1)$$

Combinando las dos últimas ecuaciones se obtiene la siguiente máscara para aproximar el laplaciano que muestra a continuación:

$$\nabla^2 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

El detector de bordes laplaciano marca el píxel central de la región donde se aplica la máscara como perteneciente a un borde cuando el resultado del producto interno calculado en dicho píxel realiza una transición por cero, es decir, pasa de un valor positivo a uno negativo, o viceversa.

Otros operadores que se utilizan para aproximar el laplaciano son los siguientes [25]:

$$\nabla^2 = \begin{bmatrix} -1 & -4 & -1 \\ -4 & 20 & -4 \\ -1 & -4 & -1 \end{bmatrix}$$

$$\nabla^2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\nabla^2 = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & -1 \end{bmatrix}$$

Canny

Si se considera un operador que sea la combinación de un filtro gaussiano (que suaviza las imágenes) con uno de aproximación de gradiente, se tiene un operador que es sensible al borde en la dirección de máximo cambio. Dentro de este modelo se encuadra el detector propuesto por *J. Canny*, considerado como uno de los más eficientes.

Se implementó un plugin que utiliza la función de OpenCV `cvCanny`. El algoritmo de dicha función se divide en cuatro pasos principales:

Paso 1. Suavizado de la imagen

La imagen es suavizada con una función gaussiana. El suavizado provoca la disminución en las diferencias de intensidad de píxeles contiguos, causando que la imagen resultante parezca más “borrosa” o difusa que la original.

Paso 2. Derivación

Se deriva la imagen obtenida en el paso 1 respecto a las direcciones x e y. A partir de estos valores se calculan en cada píxel la magnitud y ángulo del gradiente utilizando las funciones de hipotenusa y arcotangente.

La implementación de OpenCV utiliza un operador de Sobel.

Paso 3. Supresión de no-máximos

Una vez calculada la magnitud del gradiente y su dirección, los bordes pueden ser localizados en los píxeles donde el gradiente presenta un máximo local. Se eliminan de este conjunto los píxeles que no son máximos en la dirección perpendicular al borde.

Para esto se examina el gradiente en uno de los cuatro sectores que muestra la Figura 24 dentro de un entorno de 3×3 píxeles. El elemento central del entorno es comparado con sus dos vecinos en la dirección del sector correspondiente y si no es el máximo, es decir no es mayor que sus dos vecinos, es suprimido.

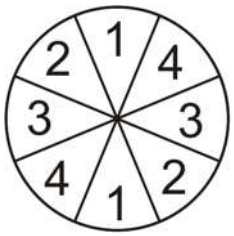

Figura 24 – Sectores para la supresión de no-máximos

Paso 4. Umbralización de los bordes

Se aplica una umbralización sobre los valores del gradiente al resultado del paso anterior para detectar bordes “falsos”, es decir, píxeles considerados como bordes pero que en realidad no lo son.

El operador de Canny aplica un tipo de umbralización llamado *histéresis*, que utiliza dos valores de umbral, superior e inferior. Considerando un segmento de línea, si el valor del gradiente de un píxel es mayor que el umbral superior, es inmediatamente aceptado. Si dicho valor es menor que el umbral inferior, es inmediatamente rechazado. Puntos que se encuentran entre ambos límites son aceptados si están conectados a algún punto aceptado anteriormente.

Notar que este proceso es aplicado sobre los valores del gradiente (calculados en los pasos 2 y 3) y no sobre los valores de intensidad de luz. Es por esto que los píxeles que se aceptan son los que superan el mayor umbral (y por lo tanto un valor de gradiente elevado por un cambio brusco de intensidad de luz) y no a la inversa como sucede cuando se umbraliza la intensidad de luz.

En general, cuando los valores del gradiente de un borde fluctúan sobre y debajo del umbral, el borde resulta cortado, lo que es conocido como *streaking*. Con la histéresis, la probabilidad de que ocurra el fenómeno de *streaking* es muy pequeña, ya que implica que el valor del gradiente de un borde fluctua sobre el umbral superior y debajo del umbral.

Erosión

El filtro de *erosión* o *erode* provoca en imágenes blanco y negro el afinado de las regiones de píxeles negros. Esto es de utilidad por ejemplo para adelgazar los trazos de un dibujo. Se implementó un plugin que utiliza la función de OpenCV `cvErode`.

Dicha función toma como parámetro la imagen a erosionar *I* y un elemento estructural *B*, que determina la forma de un entorno de píxeles, del cuál el mínimo es elegido y colocado en el lugar del píxel procesado:

$$C(x,y) = \min\{I(x+s, y+t) \mid (s,t) \in B\}$$

En caso de no especificarse el elemento estructural (como en el caso del plugin implementado), la función utiliza una matriz por defecto de 3×3.

El proceso de erosión se puede aplicar consecutivamente una cierta cantidad de veces, que se especifica como parámetro de la función.

Dilatación

El filtro de ***dilatación*** o ***dilate*** provoca en imágenes blanco y negro el crecimiento de las regiones de píxeles negros. Esto es de utilidad por ejemplo para engrosar los trazos de un dibujo. Se implementó un plugin que utiliza la función de OpenCV `cvDilate`.

Dicha función toma como parámetro la imagen a dilatar *I* y un elemento estructural *B*, que determina la forma de un entorno de píxeles, del cuál el máximo es elegido y colocado en el lugar del píxel procesado:

$$C(x,y) = \max\{I(x+s, y+t) \mid (s,t) \in B\}$$

En caso de no especificarse el elemento estructural (como en el caso del plugin implementado), la función utiliza una matriz por defecto de 3×3.

El proceso de dilatación se puede aplicar consecutivamente una cierta cantidad de veces, que se especifica como parámetro de la función.

PSF

Muchas veces los dispositivos de captura introducen manchas a las imágenes que son casi tan oscuras como los trazos de lápiz del dibujo, por lo que no es suficiente una umbralización fija para limpiarlas, ya que disminuir el umbral para eliminar las manchas produce que los trazos de lápiz se adelgacen demasiado, o incluso desaparezcan.

La idea detrás del filtro PSF, es utilizar una estrategia un poco más inteligente que una simple umbralización para eliminar las manchas y el ruido de la cámara. La estrategia utilizada es la misma que la del proceso de histéresis del detector de borde de Canny. Se establecen dos umbrales U_1 y U_2 de forma que todos los puntos con intensidad de luz menor a U_1 son automáticamente considerados parte del dibujo y pintados de negro, y todos los puntos con intensidad mayor a U_2 son automáticamente considerados fondo y pintados de blanco. Los píxeles que se encuentran entre los dos umbrales son pintados de negro siempre y cuando estén conectados a otro píxel previamente considerado como parte del dibujo.

La idea detrás de este procesamiento es obtener los puntos más oscuros de la imagen (aquellos más oscuros que U_1), que casi seguramente pertenecerán a algún trazo del dibujo y luego hacer crecer los trazos que los contienen con los píxeles

conectados que son lo suficientemente oscuros (aquellos más oscuros que U_2) como para considerarlos parte del dibujo.

Se puede suponer que los puntos más oscuros de las imágenes corresponden al centro de los trazos, ya que es donde el lápiz apoya mejor y, por lo tanto, al procesar los vecinos, los trazos crecen desde el centro hacia fuera.

Una optimización agregada al filtro consiste en no procesar los vecinos de un punto que se considera perteneciente a un borde. Para esto, cada vez que se acepta un punto, se verifica que el mismo no pertenezca a un borde utilizando los operadores de Sobel y un valor de umbral definido por el usuario. El objetivo de este control es evitar el procesamiento de píxeles fuera de las regiones definidas por los trazos del dibujo.

El filtro PSF “barre” cada línea de la imagen original buscando puntos aceptables por U_1 y utilizando una cola para procesar los vecinos de los píxeles aceptados, en lo que se puede considerar una búsqueda en amplitud en la imagen.

Ejemplos

A continuación se incluye una imagen capturada del dibujo de una línea, particularmente difícil de detectar dados los niveles de gris del trazo y del fondo, y resultados de pruebas con algunos filtros.

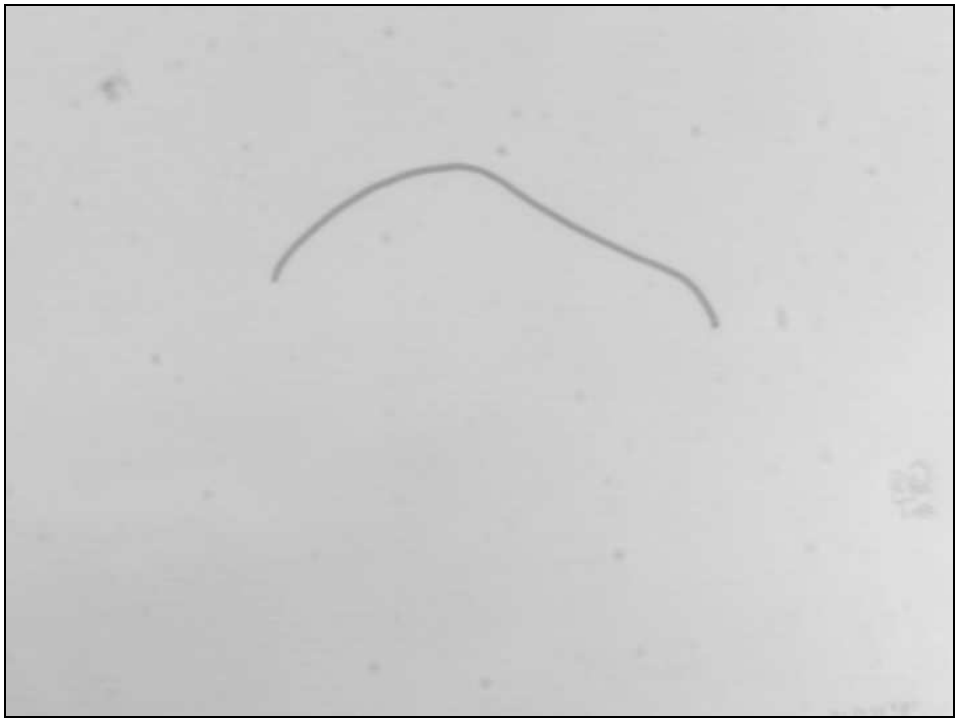

Figura 25 - Imagen original con la que se realizaron las pruebas.

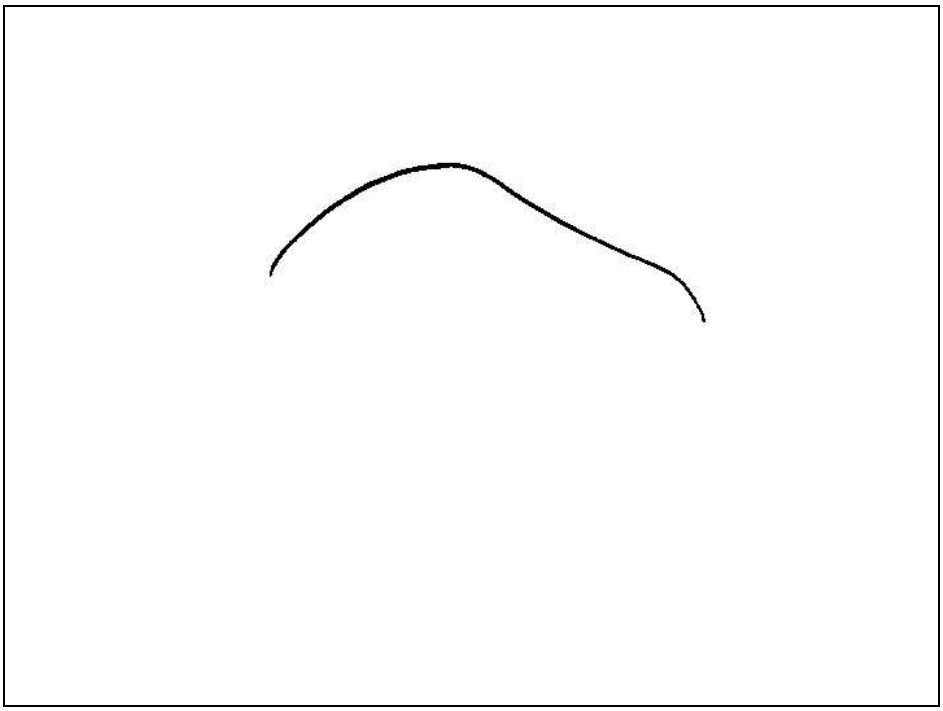

Figura 26 - Resultado de aplicar la umbralización fija a la Figura 25 con el mayor valor de umbral para el cual solo aparece la línea y todas las demás manchas son filtradas.

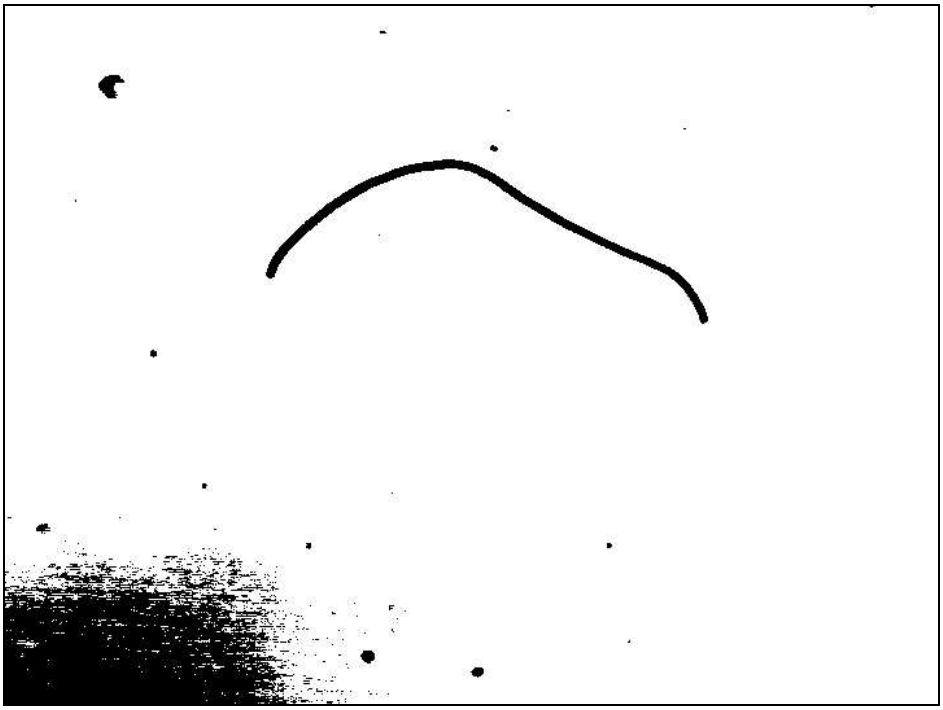

Figura 27 - Resultado de aplicar umbralización fija donde se fija el umbral en un valor para el cual la línea quede del grosor adecuado. Estos resultados muestran claramente la desventaja de este filtro.

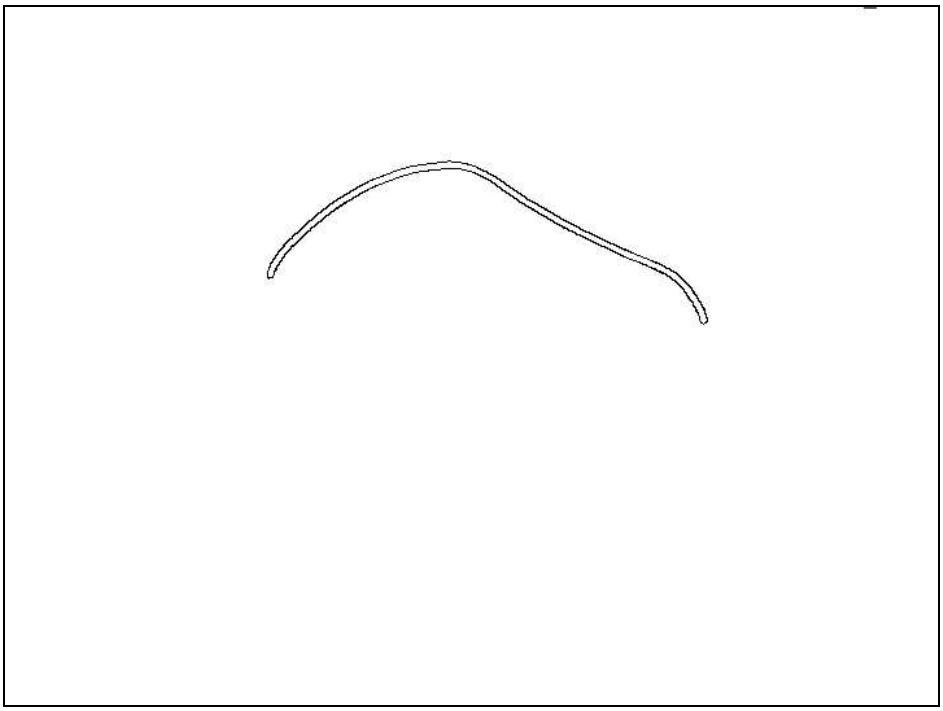

Figura 28 - Resultado de aplicar el filtro de Canny. Se puede comprobar que detecta correctamente los bordes de la línea y no la zona interior.

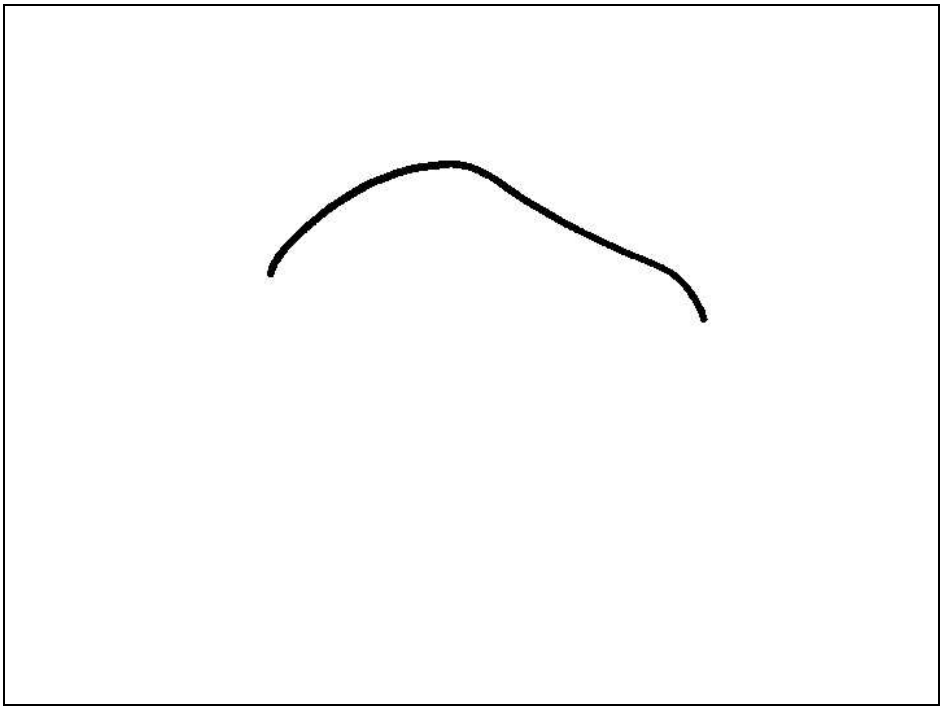

Figura 29 – Resultado de aplicar el filtro PSF. En general, el filtro logra los mejores resultados para imágenes de este tipo.

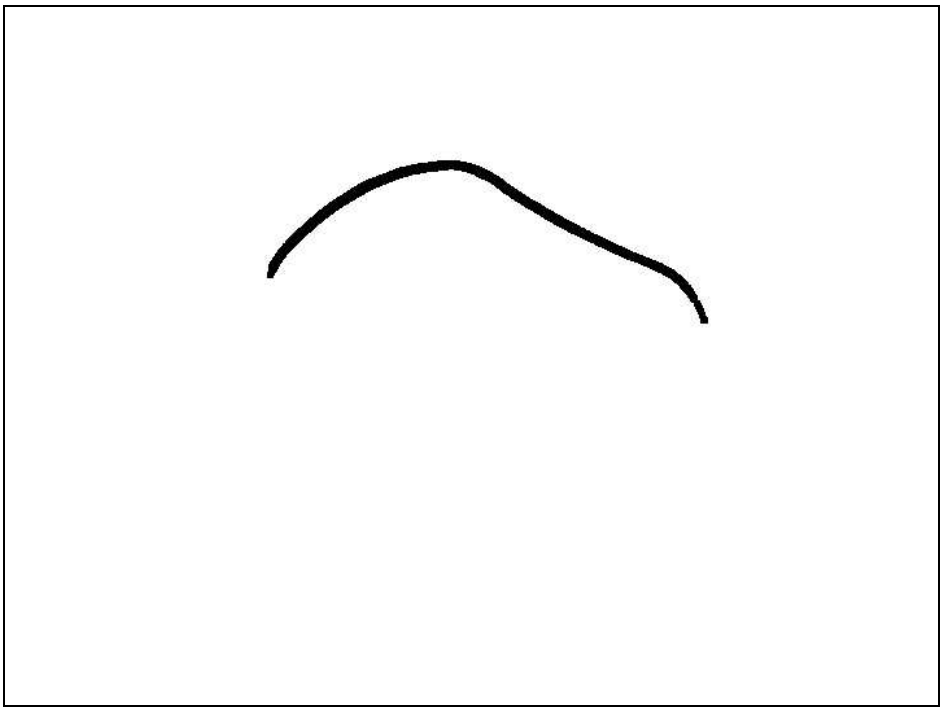

Figura 30 - Resultado de aplicar dos iteraciones de dilatación a la imagen de la Figura 26. Comprueba que la combinación de filtros puede arrojar resultados efectivos.

Anexo D – Proceso

Descripción

El proceso de desarrollo puede definirse como iterativo e incremental. Cada iteración se compuso por actividades de análisis de requerimientos, investigación, diseño e implementación, dando como resultado una nueva versión de la aplicación. Dichas actividades comenzaban lo antes posible en la iteración, buscando lograr un alto grado de paralelismo. En la Figura 31 se representa gráficamente esta distribución temporal.

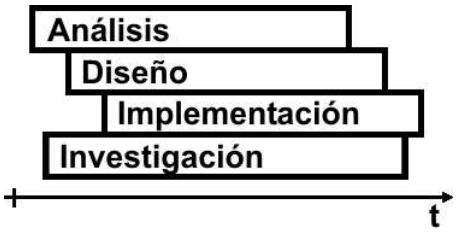

Figura 31 – Distribución temporal de las actividades en una iteración.

Cada nueva versión de la aplicación se basó en la anterior, realizando los ajustes necesarios en el diseño y el código para mejorar las funcionalidades existentes y agregar nuevas.

Organización

Al comienzo del proyecto, uno de los integrantes del grupo se dedicó a formalizar los requerimientos de la aplicación mientras que los dos restantes investigaban las herramientas y tecnologías a utilizar. Todo el grupo participó luego en la definición de la arquitectura inicial, así como en el diseño de las áreas más significativas de la aplicación (ejemplo: el modelo de datos).

Se identificaron entonces distintos temas en base a los requerimientos y al diseño inicial (captura de video, persistencia, manejo de recursos, interfaz de usuario, etc.). Los integrantes del grupo fueron asumiendo las tareas de análisis, investigación, diseño e implementación relacionadas con estos temas según su afinidad con los mismos. A partir de ese momento se trabajó individualmente pero en comunicación continua para coordinar aspectos de interacción de componentes y mantener la consistencia global del diseño de la aplicación. El uso de un sistema de control de versiones fue muy importante para llevar adelante esta forma de trabajo.

La verificación de cada componente implementado estuvo a cargo de su autor mientras que las pruebas del sistema fueron responsabilidad de todo el grupo. Dichas pruebas no fueron definidas rigurosamente quedando a criterio de cada integrante.

Cabe destacar que uno de los integrantes fue responsable del desarrollo para Linux de la aplicación, trabajando exclusivamente en esta plataforma durante el transcurso del proyecto.

Iteraciones

Primera iteración (6 semanas)

Se realizan varias reuniones con el cliente y se estudia profundamente CTP para relevar los requerimientos. Al mismo tiempo, ser realiza un estudio inicial y posterior evaluación de distintas tecnologías y bibliotecas de software.

La iteración termina con la definición el diseño inicial de la arquitectura y el desarrollo de un prototipo para verificar el funcionamiento de las bibliotecas seleccionadas.

Segunda iteración (13 semanas)

Cada integrante profundiza individualmente en el diseño a medida que es necesario y se implementa la primer versión de la aplicación.

Las funcionalidades más significativas son:

- Captura de video en Windows.
- Reproducción del video de la escena en Windows utilizando únicamente Windows Media Player (reproductor por defecto del sistema operativo).
- Filtrado de imágenes capturadas mediante umbralización fija.
- Interfaz de usuario casi definitiva.
- Deshacer/rehacer acciones del usuario.
- Copiar y pegar.

Se realiza una primera "demo" con el cliente y falla debido a imprevistos con la captura de video. En esa misma reunión se introducen dos nuevos requerimientos:

- Permitir guardar las escenas en un archivo único (en lugar de tener varios archivos dentro de un directorio).
- Permitir trabajar con varias escenas al mismo tiempo, pudiendo copiar y pegar recursos de unas en otras.

Tercera iteración (9 semanas)

Se profundiza el trabajo sobre la captura de video en Windows. A su vez, los requerimientos introducidos demandan una nueva instancia de investigación y evaluación de bibliotecas, así como de un ajuste importante en el diseño y código de la aplicación.

Se adopta en esta etapa el mecanismo de excepciones de C++ para el manejo de errores.

Como resultado de la iteración, se logra la primer versión estable y funcional de la aplicación, resolviendo completamente los nuevos requerimientos.

Las nuevas funcionalidades más significativas son:

- Guardar escenas en un archivo único.
- Trabajar con varias escenas al mismo tiempo.

Se comienza a utilizar entonces como software de base para un curso de animación que dictado en el taller de Tournier. La captura de video funciona correctamente, dejando al descubierto un nuevo problema: el filtro que incluye la aplicación no produce buenos resultados para el tipo de imágenes capturadas.

Cuarta iteración (14 semanas)

Se investigan nuevos filtros para lograr alcanzar mejores resultados. Se prueban opciones existentes y se desarrolla un nuevo filtro orientado específicamente al tipo de imágenes que manejan Tournier y su equipo.

A su vez, se diseña e implementa la interfaz de programación para extender la aplicación mediante plugins. Se identifican las secciones de la aplicación con fuerte dependencia de la plataforma y se “migran” a plugins.

Al realizar pruebas en el taller de Tournier se hace evidente el bajo rendimiento de los equipos con los que cuentan al reproducir videos de gran tamaño. Se incorpora entonces la posibilidad de escalar las imágenes de los cuadros para generar videos más pequeños.

Al finalizar esta iteración se logra la tercer versión de la aplicación, que cumple con todos los requerimientos del alcance establecido.

Las funcionalidades más significativas son:

- Nuevos filtros de imágenes.
- Soporte para pistas de audio en las escenas.
- Escalamiento de las imágenes de los cuadros al generar el video de la escena.
- Reproducción del video de la escena basado en DirectX.
- Reproducción del video de la escena utilizando cualquier aplicación externa.
- Captura de video en Linux.

Se realizan mediciones de líneas de código para esta última versión, sin tomar en cuenta el código generado automáticamente por las herramientas de QT. La aplicación cuenta con:

- 29156 loc sin líneas en blanco ni comentarios.
- 39218 loc con líneas en blanco.
- 49731 loc con líneas en blanco y comentarios.

Anexo E – Impresión de los Usuarios

TOURNIER
ANIMACION

Montevideo, 1 de marzo de 2004

El programa diseñado por los estudiantes Juan Diego Ferré, Enrico Pizzorno y Nicolás Sierra para el estudio Tournier Animaciones ha logrado los requerimientos planteados para mejorar el trabajo de captura y visionado de las animaciones.
Se logro independizar el programa con un visor de video propio y con una buena variedad de filtros que funcionan bien.
Herramientas útiles como ser la impresión de plantillas y edición de fotogramas trabajados por capas transparentes, herramientas de captura de imágenes cómodas y que agilitan el trabajo y el tamaño de la imagen logra un buen tamaño para visualizar.
Fue un desafío exitoso el trabajo con el filtro de imagen para capturar líneas de lápiz, en las condiciones de la estación de trabajo de nuestro estudio donde las capturas son blanco y negro y tiene variaciones de foco y diafragma.
Se hicieron mejoras a nivel de accesos sencillos y resolver necesidades que fueron surgiendo en el trabajo.
En definitiva quedamos muy conformes con el trabajo realizado.
Atte.

Walter Tournier
Director

Dirección: Anzani 2015 11600 / Montevideo, Uruguay
Telfax: (5982) 486 17 79
Email: tournier@adinet.com.uy

Anexo F – Compilación

Estructura de Directorios

Durante toda la etapa de desarrollo de la aplicación se utilizó la estructura de directorios que se describe a continuación (y coincide con la del material entregado en cd).

En el directorio `distribucion` se encuentran las distribuciones específicas para Windows y Linux. Para el primero, se genera un instalador y los binarios para copiar y ejecutar en forma directa. Para el segundo, un tarball (archivo .tar.gz) con el código para compilar (mecanismo útil para preservar los permisos de ejecución).

El directorio `documentacion` tiene toda la documentación generada para el proyecto. Esto es, este informe, el Manual Técnico, el Manual de Usuario y la Documentación de Clases (generada a partir del c.

En el directorio `herramientas` se encuentran las versiones utilizadas de todas las herramientas y programas necesarios para generar los binarios de la aplicación (sin considerar los compiladores). Cada herramienta tiene su propio subdirectorio donde se encuentra el instalador y, en algunos casos, un archivo `ReadMe.txt` con comentarios o aclaraciones adicionales.

Bajo el directorio `productos` existe un subdirectorio por cada biblioteca externa utilizada y el directorio `qpt` con la estructura necesaria para compilar la aplicación. Dicha estructura ubica el código fuente en `src`, las imágenes e íconos en `images`, las bibliotecas en `lib` y los binarios generados en `bin`.

En el directorio `msvc` se encuentran los proyectos Microsoft Visual C++ para compilar la aplicación en Windows y en `make`, el Makefile para hacerlo en Linux.

La distribución de binarios para Windows incluye los directorios `bin`, `images` y `usuario`, este último renombrado como `doc`.

Compilación en Windows

La compilación de la aplicación en Windows requiere (además de Visual C++) la instalación previa de:

- QT >= 2.3.0
- MinGW >= 3.1.0
- DirectX SDK >= 8.1

Para generar la Documentación de Clases es necesario instalar también Doxygen >= 1.3.5 y GraphViz >= 1.10.

libtar

Esta biblioteca debe compilarse con las herramientas de desarrollo que provee MinGW. Utilizando la terminal de este sistema, ubicarse en el directorio `productos/libtar`, e ingresar los comandos:

```
#./configure
# make qpt
```

El archivo `README-QPT.txt` explica los cambios realizados al código y Makefile por defecto de la biblioteca para poder compilarla bajo este sistema.

libffmpeg

Esta biblioteca también debe compilarse utilizando MinGW. Utilizando la terminal de este sistema, ubicarse en el directorio `productos/libffmpeg`, e ingresar los comandos:

```
#./configure --disable-debug --disable-zlib
# make qpt
```

El archivo `README-QPT.txt` explica los cambios realizados al código y Makefile por defecto de la biblioteca para poder compilarla bajo este sistema.

QPT

Para compilar el resto de las bibliotecas y la aplicación principal (junto con todos los plugins) se debe abrir el archivo de espacio de trabajo `productos/qpt/msvc/qpt.dsw` con Visual C++.

Acceder al ítem “Set Active Configuration” del menú “Build” y seleccionar “QPT-Win32 Release” como configuración activa. Luego, seleccionar el ítem “Rebuild All” del mismo menú.

Para ejecutar la aplicación desde Visual C++ (para realizar pruebas, por ejemplo) se deben seguir los siguientes pasos:

- Seleccionar el ítem “Settings” del menú “Project”.
- Seleccionar el proyecto “QPT” de la lista.
- Seleccionar la página “Debug" y asignar `..\bin\` al campo “Working directory”

Finalmente, copiar el directorio `productos/qpt/images` dentro de `productos/qpt/bin`.

Compilación en Linux

La compilación de la aplicación en Linux requiere que las siguientes bibliotecas se encuentren instaladas en el sistema:

- libpng >= 1.2.5
- zlib = 1.1.4
- qt >= 3.0

También es necesario que las herramientas de desarrollo de QT se encuentren disponibles en el PATH.

Para generar la Documentación de Clases es necesario instalar también Doxygen >= 1.3.5 y GraphViz >= 1.10.

Utilizando una terminal, ubicarse bajo `productos/qpt/make` e ingresar los siguientes comandos:

```
# make
# make install
```

El `make` compila las bibliotecas necesarias, la aplicación principal y todos los plugins. Se podría ejecutar directamente entonces desde el directorio de compilación. Luego, `make install` lo instala en el sistema, para lo cual se requiere ejecución como root.