(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 321 094**

(51) Int. Cl.:
G11B 27/034 (2006.01)
H04N 5/222 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Número de solicitud europea: **04022235 .8**
(96) Fecha de presentación : **17.09.2004**
(97) Número de publicación de la solicitud: **1638101**
(97) Fecha de publicación de la solicitud: **22.03.2006**

(54) Título: **Sistema de asistencia video para la grabación de video eficiente para la reproducción en varias pantallas.**

(45) Fecha de publicación de la mención BOPI:
02.06.2009

(45) Fecha de la publicación del folleto de la patente:
02.06.2009

(73) Titular/es: **Vantage Film GmbH**
Altstrasse 9
92637 Weiden, DE

(72) Inventor/es: **Prell, Bernhard y**
Märtin, Peter

(74) Agente: **Martín Santos, Victoria Sofía**

DESCRIPCIÓN

Sistema de asistencia video para la grabación de video eficiente para la reproducción en varias pantallas.

Campo técnico

La presente invención se refiere a un procedimiento para la creación de una estructura de datos para la grabación, el procesamiento y la reproducción asíncrona de imágenes. Además, la invención se refiere a un programa de ordenador para la creación de una estructura de datos de este tipo, que puede cargarse en una instalación de procesamiento de datos y que durante su ejecución realiza las etapas del procedimiento, así como a un sistema de procesamiento de datos, que comprende un dispositivo para la ejecución de las etapas del procedimiento.

Antecedentes y estado de la técnica

Durante los últimos años, la grabación, el procesamiento y la reproducción de películas ha cambiado progresivamente de medios analógicos a medios digitales. El motivo de este desarrollo estaba también en gran medida en las posibilidades multimedia que ofrece el medio digital, lo que se refiere, p.ej. al manejo interactivo de videos digitales, así como a las posibilidades más flexibles de guardar en memoria.

No obstante, los sistemas digitales deben cumplir requisitos estrictos ya durante la grabación, el procesamiento y la reproducción de un video individual, en lo que se refiere a la potencia de cálculo y los requisitos de control, sin mencionar los requisitos en el caso de varios videos, es decir, p.ej. en la reproducción simultánea de varios videos y/o imágenes en directo diferentes.

Debido a la gran potencia de cálculo y los requisitos de control necesarios para poder procesar simultáneamente varias funciones de pantalla diferentes de este tipo, en el software disponible en el comercio hasta ahora sólo fue posible distribuir una sola corriente de contenidos video de forma síncrona entre distintas pantallas. Por lo tanto, sólo se pudo ver en todos los equipos con pantalla conectados simultáneamente el mismo contenido video.

En el documento US 2003/0231259A1 está descrita la posibilidad de guardar varias fuentes de datos en una zona de memoria y visualizarlos a continuación en tiempo real en un equipo de salida. El contexto aquí es el campo de la televisión digital y los datos de imágenes son depositados por un dispositivo hardware en una memoria común y se emiten según la frecuencia de repetición de imagen (75 Hz) de forma conjunta en la pantalla. Típicamente las imágenes no son superpuestas sino que se incorporan en un layout de ventana con flexibilidad limitada del administrador de ventanas.

El documento US 6.570.581 B1 se mueve en el contexto del movimiento de actores reales en escenas 3D generadas por ordenador (“computer generated imagery” - CGI = “imágenes generadas por ordenador”) incl. efectos especiales. Aquí, los actores deben actuar de tal forma como si vieran realmente las CGI, aunque por lo general las CGI no se crean hasta después de la grabación en directo con los actores superponiéndose en la postproducción. Para evitar procesamientos posteriores que requieren mucho tiempo y costes, que pueden resultar porque posteriormente no coinciden las CGI y los movimientos de los actores, el documento US 6,570,581 B1 da a conocer un sistema con el que, por un lado, se crean las CGI y que, por otro lado, recibe la grabación de los actores reales. El sistema mezcla a continuación las dos señales en una pantalla.

En la clase QMovie (3qt) se dan a conocer métodos y estructuras de datos para la carga incremental de animaciones o imágenes. En este caso comienza la película en cuanto se haya generado un objeto Qmovie; en cuanto se haya reproducido el último fotograma de la película puede volverse nuevamente al punto inicial, siempre que ya se haya definido previamente un bucle de este tipo. No obstante, los objetos QMovie son “explicitly shared”, es decir, para crear películas independientes, los mismos deben construirse por separado.

Breve resumen de la invención

La presente invención tiene el objetivo de crear, a pesar de los problemas existentes en los sistemas digitales en relación con la potencia de cálculo y los requisitos del control, una arquitectura de sistema que permita una grabación, un procesamiento y una reproducción de varias corrientes video diferentes.

Esto se consigue mediante un procedimiento con las características de la reivindicación 1. Según la presente invención se crea una estructura de datos con la que se pueden visualizar y/o reproducir simultáneamente distintos contenidos multimedia. Por ejemplo, puede visualizarse en una pantalla táctil una película guardada en disco duro mientras que en una salida video puede verse permanentemente la imagen en directo de una de las cámaras que pueden conectarse. Al mismo tiempo, un sistema client conectado mediante una red (también de forma inalámbrica mediante WLAN) podría reproducir otra película ya guardada en memoria. De esta forma se crea un auténtico entorno multitasking para trabajos de dirección y cámara en el lugar de rodaje.

La estructura de datos según la invención prevé, además, que p.ej. puede superponerse la imagen en directo actual a una película reproducida de un disco, que pueden superponerse distintas imágenes en directo y que pueden visualizarse Mix-Takes (tomas mixtas) especiales (superposición de varias películas guardadas en la memoria del disco), también con la posibilidad de superponer adicionalmente la imagen en directo.

Por lo tanto, con la estructura de datos nuevamente desarrollada se ofrecen por vez primera en el entorno de las películas dos campos de aplicación valiosos nuevos sin costes adicionales por el hardware. Por un lado, pueden distribuirse varios contenidos digitales distintos, como secuencias video guardadas y/o señales video grabadas en directo, en tiempo real entre varios terminales con pantalla conectados directamente o mediante (W)LAN, pudiendo estar provistas estas señales video previamente de forma adicional de efectos de filtro calculados (el llamado modo asíncrono); por otro lado, pueden superponerse y mezclarse en tiempo real varios contenidos digitales diferentes, como secuencias video guardadas y/o señales video grabadas en directo (el llamado Mix & Overlay-Modus = modo de mezcla y superposición), pudiendo procesarse la corriente video que se genera nuevamente como contenido digital. Hasta ahora no existe un sistema de asistencia video digital y controlado por software comparable, en el que sea posible un multitasking de este tipo ya en el lugar de rodaje, mientras que el cámara controla, p.ej. la grabación de la cámara A y/o de la cámara B en los monitores TV-Out conectados con el sistema según la invención, el asistente de dirección puede encargarse al mismo tiempo en la pantalla principal (pantalla táctil) de seleccionar una serie de tomas ya existentes y unirlas para el montaje. Hasta ahora, estas medidas sólo podían realizarse alternativamente, o grabación o procesamiento posterior. Gracias a la posibilidad de la superposición de varios canales, según la presente invención también es posible por primera vez conseguir con un equipo compacto directamente en el lugar de rodaje efectos de grabación muy distintos, que hasta ahora en todo caso podían conseguirse mediante el procesamiento posterior en el estudio. Estas posibilidades no pueden conseguirse en el estado actual de la técnica de microprocesadores y software sin el procedimiento según la invención para la creación de una estructura de datos.

Conforme a la presente invención está previsto un procedimiento según la reivindicación 1.

De las reivindicaciones subordinadas resultan configuraciones especialmente ventajosas.

Breve descripción del dibujo

La fig. 1, muestra un Modelo de Colaboración UML (UML Collaboration Model) de la arquitectura de sistema asíncrono.

Descripción detallada de las formas de realización preferibles

En una capa de entrada, el sistema de procesamiento de datos según la invención lee las imágenes que entran desde una cámara de video mediante tarjetas capturadoras de video (Video Grabber Cards) o discos duro. Puede estar prevista una tarjeta capturadora propia para cada cámara cinematográfica que puede conectarse; además, los discos duros pueden estar configurados como RAID-0-array. En la capa de entrada del programa de ordenador según la invención está previsto para este fin para cada tarjeta capturadora un proceso de poco peso propio, un llamado thread, aquí denominado GrabThread y que solicita y recibe los datos de imágenes del controlador de tarjetas capturadoras. Además, cada GrabThread controla un writeThread que tiene asignado, que escribe las imágenes durante una grabación, p.ej. de forma comprimidas en JPEG, en el disco duro. El motivo de esta división del trabajo de lectura y escritura está en que el GrabThread no debería perder tiempo por la escritura en el disco duro mientras lee los datos de imágenes de la tarjeta capturadora. Para garantizar que los threads trabajan realmente en paralelo, pueden usarse varias CPU Intel Xeon, que trabajan adicionalmente en el modo hiperthreading (denominación Intel para multithreading simultáneo). Para poder reproducir una película así grabada desde el disco duro, se necesita un objeto que se denomina aquí DiscPlayer (reproductor de discos) y que trabaja con sólo lectura en una lista central de las tomas grabadas, teniendo cada toma nuevamente una lista de todas las imágenes individuales, los llamados frames (fotogramas). El reproductor de discos tiene un iterador que indica la toma activa en este momento y un iterador que indica el fotograma activo en este momento. Estos iteradores son la forma encapsulada de un puntero orientado al objeto. Puede ir hacia adelante o hacia atrás, es decir, incrementar o decrementar e indica correspondientemente el siguiente objeto o el anterior objeto de la lista a la que puede accederse con ayuda del iterador.

Para poder reproducir al mismo tiempo distintas películas de discos duros (o también de otros medios de almacenamiento) independientemente unos de otros, p.ej. en el sistema principal y en un terminal client, se necesitan varios objetos reproductores de disco que trabajan todos en una lista de tomas central. Los cambios en la lista de tomas (borrar, insertar, etc.) y en distintas tomas (cambios de montaje, cambio de la velocidad de fotogramas simulada, etc.) se realizan respectivamente desde un MovieManager que existe sólo una vez, que informa a su vez a todos los reproductores de disco de los cambios de este tipo. Además de la reproducción de las imágenes de película guardadas de esta forma, también debe ser posible la reproducción directamente mediante las tarjetas capturadoras de las imágenes en directo entrantes.

La administración central de toda la reproducción se realiza mediante el llamado ResourceManager. En primer lugar hay que mencionar que una imagen video representada, que se visualiza en una pantalla de proyección determinada, denominada aquí Canvas, puede estar formada por varias capas. Por esta razón, un Canvas tiene un número variable de capas, denominadas aquí layers. Cada capa puede tener activados atributos propios modificadores de la imagen, como reflexión alrededor del eje X e Y, blanco y negro, inversión, zoom, máscara y otros. Varios canvas se reúnen en un contenedor, denominado aquí CanvasContainer. El motivo de prever esta posibilidad es que se necesitan varios Canvas, p.ej. cuando deben visualizarse al mismo tiempo imágenes en directo de varias cámaras en un solo medio de salida (p.ej. la pantalla táctil). El CanvasContainer se encarga de la gestión de geometría de los distintos Canvas, es decir, del posicionamiento y del tamaño de los mismos en la pantalla destino.

Dicho de un modo general existe, por lo tanto, por un lado un conjunto de imágenes que son enviadas por X reproductores de discos, así como un conjunto de imágenes que proceden directamente de Y tarjetas capturadoras. Por otro lado, existen Z subconjuntos "interesantes" de éstos, que deben visualizarse en las pantallas. Por lo tanto hay que procurar que para cada pantalla se "compile un paquete" individual, que suministra la visualización deseada para esta pantalla. Como interfaz más importante hacia fuera sirve por lo tanto la función ShowPackage, que suministra un conjunto de imágenes de capas especificadas para los terminales con pantalla, estando asignado un ShowPackage individual a cada pantalla. Gracias a la selección selectiva de distintos subconjuntos para las imágenes suministradas por ShowPackage a la pantalla correspondiente pueden componerse antes de la salida propiamente dicha distintas variantes de salida para distintos terminales con pantalla, p.ej. al mismo tiempo varias imágenes independientes unas de otras, que se representan una al lado de la otra y/o distintas superposiciones o efectos para las distintas pantallas conectadas.

El ResourceManager ya anteriormente mencionado compila con ayuda de una lista de objetos de estado, los llamados objetos StateController, los ShowPackages para las pantallas. Cada StateController está asignado exactamente a un Canvas, que se direcciona mediante displayID, canvasContainerNr y canvasNr. En el caso de la reproducción de imágenes de películas guardadas en disco duro, el ResourceManager controla también los GrabThreads ya mencionados y recibe las imágenes de éstos y de los reproductores de disco. Los reproductores de disco se direccionan aquí mediante el discPlayerID. Aquí hay una restricción, según la cual cada Canvas sólo puede recibir al mismo tiempo imágenes de un solo reproductor de disco. Los Mix&Overlay-Takes no se realizan, por lo tanto, mediante varios reproductores de disco, sino que un reproductor de disco envía en este caso ya un ShowPackage con varias imágenes. En el caso de la reproducción de imágenes en directo no existe esta restricción. La llamada ChannelList representa en este caso una especie de lista de deseos, en la que figuran los canales capturadores de los que el Canvas debe recibir y representar imágenes al mismo tiempo. Una variable de estado, aquí denominada state, describe finalmente si el Canvas sólo debe recibir las imágenes del reproductor de disco, sólo imágenes en directo o imágenes del disco duro e imágenes en directo superpuestas. El StateController necesita, por consiguiente, los siguientes parámetros:

StateController

displayID

canvasContainerNr

canvasNr

discPlayerID

channellList

state

Puesto que puede haber varias pantallas, p.ej. una para el equipo principal y una para cada pantalla client, también se necesita una estructura de administración para ello. El llamado DisplayController sirve aquí como representante de una pantalla en el ResourceManager. Contiene varias listas que son necesarias para compilar los ShowPackages, así como un puntero que indica la pantalla propiamente dicha; ésta recibe a continuación justamente el ShowPackage que está previsto sólo para esta pantalla.

La estructura de la "capa de transporte", en la que se realiza la transferencia de las imágenes a las pantallas, está determinada por el requisito que una imagen para cada pantalla se copia también sólo una vez en la memoria de gráficos, puesto que justamente esta operación es la operación costosa (en tiempo) y en dinero. En la presente invención, este requisito ha tenido sus consecuencias entre otras cosas en la separación de imagen y destino de visualización. Con las explicaciones anteriormente expuestas debería haberse dejado claro que una imagen que se visualiza en la pantalla ya no es sólo una imagen sencilla, sino que está formada por varias imágenes, que se han ampliado con informaciones de direcciones. El caso realizado en el Harddisk-Recorder (grabador de disco duro) "PSU" ® comercializado hasta ahora por Vantage Film GmbH, según el cual una imagen se visualiza en un Canvas determinado, se convierte por lo tanto en el caso especial de la estructura de visualización ampliada según la presente invención.

Esta estructura de datos de visualización se denomina al igual que la función ya anteriormente mencionada ShowPackage y está formada por una lista de llamados ShowItems. La lista ShowItem es generada por la aplicación y depende de la configuración de pantalla ajustada por el usuario, que puede ser cambiada dinámicamente. Cada ShowItem contiene exactamente una imagen y una lista de posibles destinos de visualización, denominados aquí ShowTargets. El ShowTarget indica a su vez detalladamente el destino de visualización. La estructura y la relación de ShowPackage, ShowItem y ShowTarget se presentan, por lo tanto, de la siguiente manera:

ShowPackage:	ShowItem:	ShowTarget:
* Lista de ShowItems	* Imagen	containerNr.;
	* Lista de ShowTargets	canvasNr.;
		layerNr.;

Cada imagen con el mismo contenido figura sólo una vez en la lista de ShowItems del ShowPackage. Si debe aparecer en distintos layers, la lista de ShowTargets, de la que hay una para cada imagen como se ve en la lista arriba indicada, tiene un número de entradas correspondientemente alto. Esto es así porque cada copia de una imagen, p.ej. desde la memoria principal a la memoria de gráficos de la tarjeta gráfica requiere bastarte tiempo. Gracias a este estructura se tiene por lo tanto en cuenta el requisito anteriormente indicado reduciéndose el tiempo para copiar al tiempo mínimo necesario.

La visualización en la pantalla, que termina el "recorrido" de las imágenes, se realiza con el lenguaje gráfico OpenGL, habiéndose realizado adaptaciones de eficiencia adicionales para el procesador gráfico Nvidia usado en la presente invención. Una ventaja sustancial del uso de OpenGL está en que la mayor parte de los efectos de imagen, denominados aquí PictureSettings, son realizados por el hardware del procesador gráfico, por lo que requieren claramente menos tiempo de cálculo de la CPU en los procesadores principales xeon. Las CPUs principales sólo deben hacer que todas las imágenes que han de ser representadas en paralelo, han de ser superpuestas o modificadas por efectos, se encuentren en los pipelines de procesamiento del procesador gráfico. Al principio de la ventana de cálculo de 40ms (PAL) o 33 ms (NTSC) se transfieren por lo tanto las imágenes necesarias para la aplicación de las CPUs principales de la memoria principal a la memoria de gráficos de la tarjeta gráfica. Con el hardware actual se necesita por imagen un tiempo de transferencia de 2 ms; cuando deben superponerse, por ejemplo, cinco imágenes, esto tarda 10 ms. En los equipos de salida PAL quedan por lo tanto aún 30 ms disponibles para el cálculo de los efectos y superposiciones y la salida en los terminales. Las imágenes como p.ej. máscaras fijamente ajustadas o superposiciones permanecen en la memoria de gráficos y no deben volver a transferirse nuevamente en cada ventana de tiempo al procesador gráfico. Con el procedimiento dado a conocer en la solicitud también pendiente "Procedimiento para crear una estructura caché para el acceso rápido a objetos guardados en memoria de forma separada" puede conseguirse, además, que siempre estén disponibles suficientes imágenes de video en tomas posiblemente distintas en la memoria principal.

Para el grabador de disco duro "PSU" ® comercializado hasta ahora por Vantage Film GmbH existen como efectos ya la reflexión X/Y, el zoom y la máscara, así como la representación en blanco y negro, que según la presente invención son calculados ahora todos por el procesador gráfico. La variante de superposición ya realizada en este grabador de disco duro con una posición fija se desplaza según la presente invención ahora también a la CPU gráfica, al igual que el gráfico tipo rampa superpuesta a una toma reproducida actualmente. Un gráfico tipo rampa representa el desarrollo de la velocidad de la cámara durante la toma e indica dinámicamente donde se encuentra la imagen actualmente representada en la rampa. El gráfico tipo rampa ahora puede estar incluso activo con un efecto zoom conectado simultáneamente, lo cual no era posible en el grabador de disco duro disponible hasta la fecha.

Técnicamente es posible activar en cada una de las capas distintos efectos de imagen o ningún efecto y superponer varias capas por terminal con pantalla. Gracias al uso de la función ShowPackage, por ejemplo es posible hacer que el gráfico de rampas superpuesto a una imagen de color aparezca sólo en el monitor principal táctil, mientras que en las pantallas adicionales conectadas mediante salidas de video se emite una imagen de blanco y negro de la escena sin gráfico tipo rampa. Para los efectos de pantalla azul/verde necesarios en el campo de los efectos especiales de estudio, se usa al menos en parte el hardware de gráficos. Se encarga de la mezcla de partes correspondientes de la imagen después de haberlas marcado como transparentes o opacas.

A continuación, se seguirá con ayuda de ejemplos de códigos comentados a título de ejemplo el recorrido de una imagen desde iniciar la acción "Live-Modus" (Modo en directo) desde la capa de entrada pasando por la administración central y la capa de transporte a la capa de visualización. En los ejemplos de códigos se encuentran informaciones descriptivas importantes para el proceso o las distintas funciones respectivamente en los comentarios marcados con "//".

Al hacer clic en el botón "Live-A" (GUI) se llama en primer lugar una función slotLiveA en la clase MainGUI:

```
slotLiveA()
(
   ...
   // El objeto de aplicación tiene dos StateController, que corresponden
   // respectivamente al canal A o B
```

```
    // Desconectar Controller B
    _ctrlB->disable(true):
    // cambiar Controller A a LiveModus
    _ctrolA->liveMode (lineA);
...
)
```

Aquí, se desconectó el StateController para el canal B y se cambió el del canal A modo en directo. Como ya se ha mencionado anteriormente, cada objeto StateController tiene asignado exactamente un Canvas y dispone entre otras cosas de atributos y funciones que indican el destino de visualización exacto, así como los canales “interesantes”, los canales en directo que eventualmente han de ser superpuestos y el modo en cuestión (en directo o grabación):

```
Class StateController
(
      int _displayID:     // ¿Adónde deben enviarse imágenes para este
      int _discPlayerID;  // StateController?
      int _canvasContainerNr.:
      int _canvas Nr.;
      StatusType _state;  // Entre otros, pueden ser: State_DISC, STATE_LIVE,
                          // STATE_REC

      ChannelList_channelList:  // Lista de todos los canales en directo para
                                // los que se interesa este StateController;
                                // denominada posteriormente lista de
                                // intereses

      bool _liveOverlayActive;
      ChannelList _overlayChannels;    // Lista de todos los canales en
                                       // directo que deben superponerse
                                       // a la imagen del disco (ayuda
                                       // informativa)

      void disable (bool val);  // Conectar/desconectar controller
      void discMode ();
      void livemode (SourceType channel); // cambiar a modo directo, etc.
      void recMode (SourceType channel);
...
)
```

En el ejemplo aquí descrito se ha seleccionado el modo en directo, por lo que ahora se describirá más detalladamente la función LiveMode ():

```
StateController:: liveMode(SourceType channel)
(
   ...
   state=STATE_LIVE; // cambiar a estado directo
   // Si el canal aún no está en la lista de intereses -> añadir
   if ((channelListContains(channel))
       /
       _channelList.clear();
       _channelList.append(channel);
       )
   // Mensaje a ResourceManager "Cambia dado el caso GrabThread a LiveModus"
emit startLive(channel);
   // Si Overlay activo, añadir todos los canales que figuran en la
   // lista overlayChannels a la _channelList
   if (_liveOverlayActive)
       (
       addLiveoverlay(_overlayChannels);
       )
)
```

Durante la ejecución de esta función se cambia al estado en directo, el canal seleccionado se añade a la lista de intereses (si no figura aún allí) y al ResourceManager se envía el mensaje startLive (canal), que es recibido por la siguiente función:

```
ResourceManager::slotStartLive(SourceType source)
(
GrabThreadList::Iterator it;
...
   // Recibir GrabThread para el canal transferido
   it=_grabthr4ead.at(source);
   // cambiar GrabThread a LiveModus

   (*it)->startShowing();
...
...
)
```

El proceso hasta ahora descrito ha servido para la descripción de lo que pasa directamente en los ajustes predeterminados en el sistema cuando se aprieta el botón “Live-A”. A continuación, se explicará como las imágenes llegan del GrabThread mediante la administración central y la capa de transporte a la pantalla.

La llamada de la función GrabThread::startShowing () despierta el thread, dado el caso. Comienza en un bucle sin fin a leer imágenes desde una tarjeta capturada de fotogramas. A partir de cada imagen leída se genera un mensaje que termina finalmente en el ResourceManager. Descrito de una forma fuertemente simplificada, el ResourceManager tiene el siguiente aspecto, siendo especialmente importante la función slotDispatch (), puesto que en ella se compilan los paquetes individuales, es decir, los Show-Packages para la visualización (véanse al respecto las explicaciones más adelante).

```
Class ResourceManager
(
private:
MovieManager* _mm;                // Administrador de todas las tomas
DisplayControllerList _displays; // Lista de todos los Displaycontroller
StateControllerList _stateControls; // Lista de todos los StateController
GrabThreadList _grabThread;       // Lista de todos los GrabThreads
ShowItemList _lastGrabImage;      // Lista de todas las imágenes recibidas
                                  // en último lugar de los GrabThreads
                                  // (una imagen por Thread)
DiscPlayerList _discPlayers;      // Lista de todos los DiscPlayer
ShowPackageList _lastShowPacks;   // Lista de todos los ShowPackages
                                  // recibidos en último lugar de los
                                  // DiskPlayer
                                  // (un package por DiscPlayer)
bool _newInput;                   // Bandera: Ha llegado una nueva imagen
                                  // desde el último envío
void handleShowImage (ShowImageEvent* ie); // Recibe imágenes de GrabThreads
public slots;
void slotShowPackage(int. ShowPackage): // Recibe packages de DiscPlayers
void slotDispatch(): // compilación de los paquetes para la visualización
void slotStartLive (SourceType source);
void slotStopLive (SourceType source);
void slotStartRecord (SourceType source);
void slotStopfRecord (SourceType source);
);
```

Cada imagen de los GrabThreads se alimenta a la siguiente función, en la que se convierte en un ShowItem, que ahora contiene además de la imagen también la lista de los destinos de visualización, es decir, de los ShowTargets:

```
void ResourceManager: :handleShowImage (ShowImageEvent* ie)
(
   ShowItemList;;iterator it;
   SourceType src;
  // Extraer canal del paquete
   src=ie->_srcNr;
   // Hay una lista de todas las imágenes recibidas en último lugar de los
   // GrabThread o aquí ya una lista de los ShowItems
   it = _lastGrabImages.at(src);
       (
       // Copiar imagen en la lista, ¡convertir al mismo tiempo en ShowItem!
       (*it)=ie->_image;
       _newInput=true;
       )
)
```

Por lo demás, aquí no ocurre mucho en cuanto al procesamiento. Como ya se ha mencionado varias veces, la compilación del ShowPackage definitivo no debería iniciarse cada vez cuando llega una nueva imagen (del grabber) o un nuevo paquete (del reproductor de disco), porque sino sería excesiva la carga de cálculo para el procesador gráfico (tarjeta gráfica). Cuando hay, por ejemplo, dos canales en directo A y B y un reproductor de disco, llega como promedio cada 40 ms/3=13 ms una nueva imagen. El promedio de este tiempo es demasiado corto para volver a crear nuevamente la visualización. En lugar de ello se “compila cada 40 ms un paquete global”. Esto se inicia mediante uno de los GrabThread, que tras cada imagen recibida transmite además de la imagen propiamente dicha también un “impulso”, que inicia la función central slotDispatch() ya mencionada en la que se compilan los ShowPackages individuales para la visualización. Esta función slotDispatch es el “núcleo” propiamente dicho del concepto según la invención. Por ello, se explicará el funcionamiento de esta función central aquí en primer lugar a modo de una especie de seudocódigo:

¿Hay nuevas imágenes que deben ser visualizadas? No -> salir de la función, bucle que afecta a todos los State-Controller.

¿Debe integrarse el último paquete que se ha recibido del reproductor de disco correspondiente?

Sí: Añade a todos los Items del ShowPackage asignado el destino (target) del StateController actual (contenedor, Canvas).

¿El StateController actual se interesa para alguna imagen en directo (_channelList tiene entradas)?

Sí: Bucle que afecta a todas las entradas de la lista. Añade imagen o ShowItem del canal correspondiente al paquete definitivo. (Este se encuentra en el DisplayController). La operación de insertar garantiza que esta imagen no está ya incluida. Añade el target a este ShowItem.

Bucle que afecta a todos los DisplayController

Añade todos los DiscPlayer-Packages al paquete global del DisplayController que debe visualizarse en esta pantalla.

Ahora, el código real correspondiente de la función slotDispatch:

```
   void ResourceManager: :slotDispatch()
   (
   int usedLayers;
  int displayID=1;
   int container, canvas;
   StatusType state=STATE_DISC;
   DisplayControllerList::iterator displayIt;-
   StateControllerList::Iterator stateIt;
   ChannelList::Iterator channelIt;
   ShowPackageList::Iterator lastPackIt;
   ShowItemList::Iterator grabImageIt;
   DisplayController::DiscPlayerIDMap::Iterator playerID_it;
If (!-newInput) ( return; ) // No ha llegado nada nuevo
...
// Bucle que afecta a todos los StateController
for (stateIt=_stateControls.begin(); stateIt/=_stateControls.end(); ++stateIt)
   (
   usedLayers=0;
   if ((*stateIt)->isDisabled()) continue;
   // ¿A qué Display(-Controller) pertenece el StateController?
   displayID=(*stateIt)->displayID();
   state=(*stateIt)->state();
   displayIT=_displays.at(displayID);
   // Conseguir contenedor y Canvas del Statecontroller actual
    (*stateIt)->getCanvas(container, canvas);
   // ¿El contenedor está interesado en el último
   // Disc-ShowPackage de su DiscPlayer?
   If (state==STATE_DISC)
       (
       // Añade a todos los items del lastShowPackage el target
       //a.) Consigue lastShowPackage
       lastPackIt=getLastShowPackageIt((*stateIt)->disPlayerID());
       // Se añade a cada Item un target con
       // índice números layer del ShowItem
       (*lastPackIt).addTarget(container, canvas);
       // entrada en lista informativa en DisplayController, que el
       // Package de este DiscPlayer debe copiarse más tarde (al final
       // del Dispatch) en el DisplayController
       (*displayIt)->noteDiscPlayerID((*stateIt)->discPlayerID());
       usedLayers=(*lastPackIt).itemCount();
     )
// ¿El Controller está interesado en las imágenes en directo de su
// ChannelList? Bucle que afecta a todas las entradas del ChannelList del
Controller for (channelIt=(*stateIt)-> channelList.begin();
```

```
channelIt(=(*stateIt)->_channelList.end(); ++channelIt)
   (
      // Generar nuevo ShowItem y añadir al ShowPackage del
      // Display-Controller (!)
      // En primer lugar conseguir imagen en directo
      grabImageIt=_lastGrabImages.at((*channelIt));
      (*displayIt)->addItem((*grabImageIt), container, canvas, usedlayers);
      // usedLayers++;
     )
)  // Final del bucle que afecta a todos los StateController
// Copiar finalmente los DiscPlayer-ShowPackages que están anotados
  // previamente en cada DisplayController al DisplayController
  // (véase arriba noteDiscPlayerID()).
  // A continuación, emitir a Diaplay
  for (displayIt=_displays.begin(); displayIt!=_displays.end();(++displayIt.
   (
     for (playerID_It=(*displayIt).->_noteDiscPlayers.begin();
          playerID_It!=(*displayIt)->_noteDiscPlayers.end();++playerID_it)
     (
       lastPackIt=getLastShowPackageIt(playerID It.data ());
       (*displayIt)->addDiscPackage((*lastPackIt));
     )
   (*displayIt)->emitPackage();
   )
 _newInput=false;
)
```

Después de haberse "compilado los paquetes" de esta forma con la función slotDispatch, es decir, después de haberse compilado los ShowPackages, la función DisplayController::emitPackage transmite ahora cada paquete acabado a "su" pantalla. Allí es recibido por la función slotShowPackage():

```
DisplayGL::slotShowPackage(ShowPackage package)
)
  ShowItemList::Iterator ItemIt;
  ShowTargetList::Iterator TargetIt;
...
// Bucle que afecta a todos los ShowItems en el paquete
 for (ItemIt=package._itemList.begin();
  ItemIt!=package._itemList.end(); ItemIt ++)
```

```
(
  // Bucle que afecta a todos los targets del ShowItem actual
  for (TargetIt=(*ItemIt)._targetList.begin();
       TargetIt!=(*ItemIt)._targetList.end(); TargetIt++)
    (
      // Conseguir el objeto Layer que se direcciona mediante el
      // target. Contiene las coordenadas donde debe ser dibujado
      // y diversas informaciones acerca del "Cómo" (efectos,
      // etc.) Es importante la "dirección" donde los datos de
      // imágenes están guardados en la memoria de gráficos que
      // deben aparecer en este Layer.
      LayerInfoItem& layerInfo=getLayerInfo((*targetIt)._containerNr,
            (*TargetIt)._canvasNr. (*TargetIt)._layerNr);
      // entrega de diversos valores, evaluación más tarde
      // al realizar el dibujo propiamente dicho
      layerInfo._alphaBlending=(*ItemIt)._alphaBlending;
      layerInfo._effectsPossible=(*ItemIt).effectsPossible;
      ...
      // Transferencia de la imagen como textura a la memoria de la
      // tarjeta gráfica, introducir identificador textura en el
      // objeto del layer
      layerInfo._currentTextureIt=texMan.insertTexture((*ItemIt)._image
        (*ItemIt)._id, layerInfo._currentTextureIt, ...);
          )
    )
  )
paintGl ();          // Aquí se realiza el dibujo propiamente dicho de las
                     // imágenes
swapBuffers();       // Cambia el tampón en el que se está dibujando en este
                     // momento por el tampón indicado.
)
```

Aquí, el objeto Layer (capa) que se direcciona mediante el target contiene las coordenadas donde debe ser dibujado, así como diversas informaciones acerca de la forma como debe ser dibujado (es decir, determinados efectos, etc.). También es importante la dirección donde los datos de imágenes se encuentran exactamente en la memoria de gráficos, que deben aparecer en esta capa. En la función paintGL () implementada en OpenGL se itera mediante una lista de objetos Layer. Según las entradas en estos objetos Layer (posición, tamaño, factor zoom, blanco y negro, etc.) se proyectan las distintas imágenes como textura en rectángulos y se dibujan una encima de otra de forma semitransparente, siempre que existan imágenes a superponer.

Con ello se ha reproducido completamente el recorrido representado a modo de ejemplo de una imagen desde iniciar la acción "Live Modus" desde la capa de entrada pasando por la administración central y la capa de transporte hasta la capa de visualización.

La interfaz del usuario para el control, desde la cual también se ha puesto en marcha la acción arriba indicada "Live Modus", contienen los botones y elementos de visualización ya conocidos por el grabador de disco duro "PSU" ®. En respuesta a acciones del usuario activa, p.ej. las variables de algunos statecontroller y envía entre otras señales al ResourceManager (startLive (canal), ...), al Moviemanager (delete Take (x)...), al DiscPlayer (play ()...) y al Display (hawk (), dual-View ()...).

Naturalmente, el sistema arriba descrito puede ampliarse añadiéndose el uso de clients de red. Para ello deben registrarse en el lado del servidor en el ResourceManager un número de statecontroller que corresponde al número de Canvas en los clients. Un objeto proxy recibe mensajes a través de la red y activa p.ej. las variables stateController en el servidor. Además, para cada client debe haber también un reproductor de disco en el servidor, en caso de desearse una reproducción desacoplada. El objeto proxy actúa al mismo tiempo como "pantalla virtual". En el ResourceManager hay un DisplayController propio para cada client, mediante el cual pueden enviarse los paquetes al client.

Además de la salida de imágenes, el ResourceManager controla también el sonido, es decir, la grabación de sonido (2 threads) y la reproducción. Para ello administra los equipos de sonido (dispositivo de sonido, mezclador). Cada reproductor de disco necesita también dos threads para la reproducción del sonido, leyendo un thread los datos de sonido de un fichero y transfiriendo el otro los datos de sonido a la tarjeta de sonido para la salida.

REIVINDICACIONES

1. Procedimiento para la creación de una estructura de datos para la grabación, el procesamiento y la reproducción asíncrona de varias corrientes video diferentes en un ordenador, que entran desde diferentes fuentes,

- realizándose la reproducción en el interior de ventanas de reproducción Canvas en al menos una pantalla,

- y estando representada cada pantalla en una estructura de datos central de ResourceManager por un elemento de datos Displaycontroller,

  - asignándose a cada pantalla al menos un elemento de datos Canvas, así como exactamente un elemento de datos ShowPackage, que contiene una lista de elementos de datos ShowItem a visualizar en esta pantalla,

  - teniendo asignado cada elemento de datos ShowItem exactamente una imagen de una corriente video y conteniendo una lista de elementos de datos ShowTarget, que están asignados respectivamente a exactamente un elemento de datos Canvas, que designa la ventana de reproducción Canvas, en la que debe visualizarse esta imagen,

  - estando asignado a cada elemento de datos Canvas exactamente un elemento de datos StateController, que direcciona este elemento de datos Canvas mediante el elemento de datos ShowTarget e indica qué corrientes video deben visualizarse en la ventana de reproducción Canvas correspondiente,

  - comprendiendo la actualización de las ventanas de reproducción Canvas en las pantallas con ayuda de las imágenes entrantes las siguientes etapas:

    para cada elemento de datos StateController y para cada imagen entrante de una corriente video:

    - consulta del elemento de datos StateController actual para averiguar si la corriente video de la que procede la imagen entrante actual debe visualizarse en la ventana de reproducción Canvas que está asignada al StateController actual,

    - en caso de deber visualizarse la corriente video en la ventana de reproducción Canvas correspondiente y en caso de que la imagen entrante actual aún no tiene asignado un elemento de datos ShowItem, generación de un nuevo elemento de datos ShowItem, que se asigna a la imagen entrante y

    - añadidura del elemento de datos ShowItem asignado a la imagen entrante actual a la lista de elementos de datos ShowItem del elemento de datos ShowPackage correspondiente del elemento de datos StateController actual, así como

    - añadidura del elemento de datos ShowTarget del elemento de datos StateController actual a la lista de elementos de datos ShowTarget del nuevo elemento de datos ShowItem.

2. Procedimiento según la reivindicación 1, **caracterizado** porque se asigna al menos un elemento de datos Layer, en el que pueden activarse respectivamente atributos modificadores de la imagen, a cada elemento de datos Canvas y porque cada elemento de datos ShowTarget está asignado exactamente a un elemento de datos Canvas y exactamente a un elemento de datos Layer.

3. Procedimiento según una de las reivindicaciones anteriores, **caracterizado** porque además está previsto un objeto proxy, que recibe mensajes a través de la red y que fija las variables para los elementos de datos StateController de los elementos de datos Canvas que están asignados a las pantallas conectadas mediante clients.

4. Programa de ordenador para la creación de una estructura de datos para la grabación, el procesamiento y la reproducción asíncrona de varias corrientes video diferentes, que puede cargarse en una instalación de procesamiento de datos y que realiza en el momento de su ejecución las etapas del procedimiento según una de las reivindicaciones anteriores.

5. Sistema de procesamiento de datos que comprende dispositivos para la ejecución de las etapas del procedimiento según una de las reivindicaciones 1 a 3.

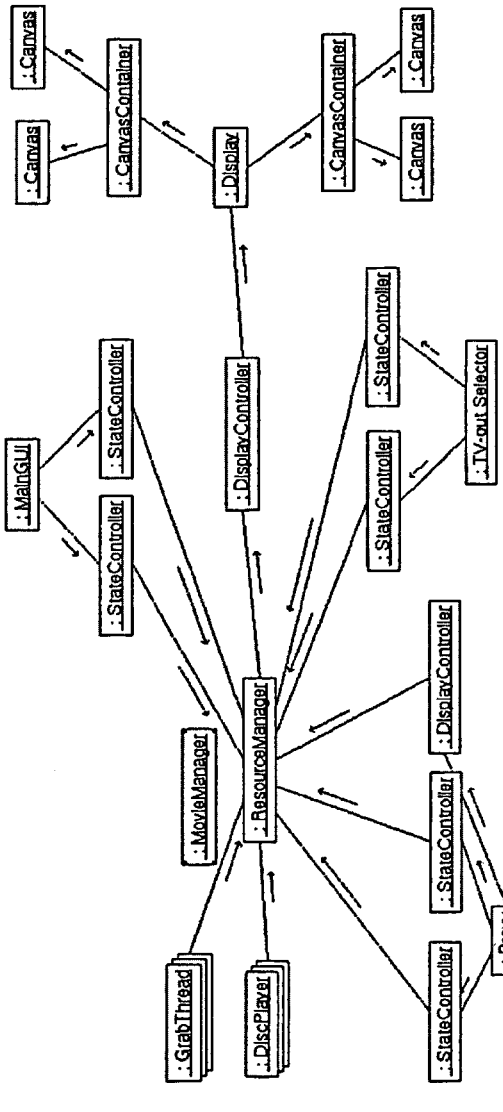
:Canvas
:Canvas
:CanvasContainer
:Display
:CanvasContainer
:Canvas
:Canvas
:MainGUI
:StateController
:StateController
:DisplayController
:StateController
:StateController
:TV-out Selector
:ResourceManager
:MovieManager
:DisplayController
:StateController
:StateController
:GrabThread
:DiscPlayer
:Proxy
RED
:Proxy
:Display
:CanvasContainer
:Canvas
:Canvas
:Client-GUI

Figura 1