Sistema integrado de generación de mapas de profundidad para videoconferencia 3D en tiempo real

John Congote[(1)(4)], Iñigo Barandiarán[(1)], Javier Barandiarán[(1)], Pere J. Mindan[(2)], Olga Mur[(2)], Genís Aguilar[(2)], Francesc Tarrés[(2)], Tomas Montserrat[(3)], Julien Quelen[(3)], Christian Ferran[(3)], Oscar Ruiz[(4)]

{jcongote, ibarandiaran, jbarandiaran}@vicomtech.org,
{perequim, olga.mur, genis.aguilar, tarres}@gps.tsc.upc.edu, {tmmora, julien, icfb}@tid.es

[(1)] VicomTech, Mikeletegi Pasealekua, 57, 20009, Donostia, España.

[(2)] Departamento de Teoría de la Señal y Comunicaciones, UPC
Campus Nord, D-5, Jordi Girona 1-3, 08034 Barcelona, España.
[(3)] Telefonica Research, *Barcelona*. *Via* Augusta, 177. 08021, *Barcelona*. España.
[(4)] CAD CAM CAE Laboratory, EAFIT University, Carrera 49 N° 7 Sur – 50, Medellín, Colombia

***Abstract*- This paper presents the implementation of a high quality real-time 3D video system intended for 3D videoconferencing. Basically, the system is able to extract depth information from a pair of images coming from a short-baseline camera setup. The system is based on the use of a variant of the adaptive support-weight algorithm to be applied on GPU-based architectures. The reason to do it is to get real-time results without compromising accuracy and also to reduce costs by using commodity hardware. The complete system runs over the GStreamer multimedia software platform to make it even more flexible. Moreover, an autoestereoscopic display has been used as the end-up terminal for 3D content visualization.**

I. INTRODUCCIÓN

El uso de la videoconferencia como método de comunicación a distancia ha sufrido un enorme cambio estos últimos años. Empresas y particulares recurren a ella con el objetivo de reducir costes y mejorar la comunicación entre individuos. Aun así, la sensación percibida por un usuario en una aplicación de videoconferencia 2D tradicional todavía dista mucho de la naturalidad que implica una conversación presencial. La videoconferencia 3D presenta una evolución lógica a su predecesora y ofrece una mayor y más natural sensación de presencia. El aumento actual de la producción y comercialización de monitores 3D, así como el aumento de las prestaciones de los ordenadores domésticos hacen de la tele-presencia una alternativa factible.

Este artículo presenta un método de videoconferencia basado en la implementación sobre GPU de un algoritmo de estéreo-visión. En el primer apartado se puede observar una descripción sobre los diferentes métodos implementados en la solución final, Seguidamente, en el tercer apartado, se describen los resultados obtenidos mediante el uso de los algoritmos implementados y su posterior integración en la plataforma común. Finalmente, en el último apartado podemos leer las conclusiones del trabajo realizado así como las futuras líneas de trabajo.

II. METODOLOGIA

En esta sección se describen en detalle los métodos para la extracción de mapas de profundidad en tiempo real que han sido implementados en nuestra arquitectura.

A. Rectificación de imágenes

La obtención de información de profundidad se basa en la distancia relativa entre las proyecciones de un mismo punto del mundo 3D sobre cada uno de los planos de proyección de las cámaras. La posición de las cámaras, así como la distorsión en la imagen causada por las lentes, hacen necesarias algunas correcciones previas, con vistas a una mejora de rendimiento y calidad en el cálculo de profundidades. Además de la corrección de la distorsión, es también necesario alinear horizontalmente las dos imágenes para restringir la búsqueda de correspondencias a un problema de una sola dimensión. Para este propósito, es necesario el previo cálculo de los parámetros de calibración binocular así como la caracterización de cada una de las cámaras.

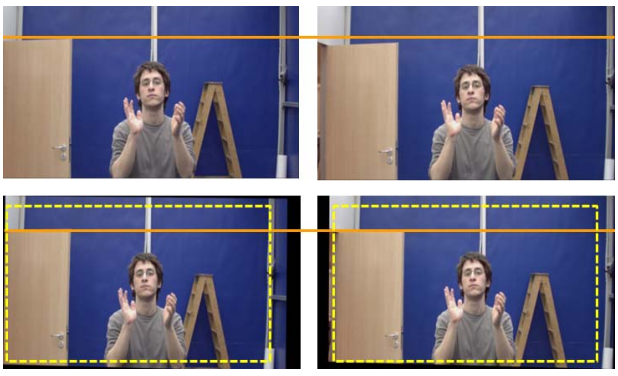

Fig. 1. Arriba: capturas izquierda y derecha. Abajo: corrección de la distorsión y rectificación de las imágenes. El cuadro amarillo indica el recorte del área de interés.

Mientras que la calibración de las cámaras puede realizarse previamente y sin restricción de tiempo de ejecución, la corrección y rectificación deben hacerse en tiempo real para cada par de imágenes. Para asegurar una frecuencia de captura aceptable, los algoritmos de rectificación han sido implementados en CUDA para permitir su ejecución sobre GPU. La ejecución en GPU de este tipo de procesos ofrece un alto aumento de rendimiento respecto a la ejecución sobre Procesadores de Propósito General (PPG) tradicionales a un coste razonable y asumible. Cada una de las imágenes rectificadas tiene una resolución de 960x540 puntos, tal como requiere el monitor 3D utilizado. Del proceso de rectificación pueden resultar píxeles fuera del área de interés del monitor, produciendo regiones vacías en la imagen. Para evitarlo, se capturan las imágenes a una resolución mayor (1200x600) y se recortan una vez rectificadas (como puede verse en la Figura 1).

B. Adaptive support-weight

Adaptive support-weight, AW, es un algoritmo local diseñado por Yoon[1]. Se basa en el uso de correspondencias a partir de ventanas de búsqueda y la asignación de pesos variables a los píxeles procesados. Fijada una ventana de búsqueda, *AW* otorga un peso para cada par estéreo de píxeles, representando el grado de similitud entre ambos. La asignación del peso viene determinada por el cálculo de una métrica de proximidad geométrica y de similitud en color. Finalmente, se establece correspondencias entre aquellos pares de píxeles que hayan obtenido un menor peso.

Sea *D* un mapa de disparidad del mismo tamaño que las dos imágenes originales *L* , *R* en que cada $D_{i,j}$ representa la disparidad entre las coordenadas de un píxel en *L* y en *R.* Es decir $L_{[i+D_{[i,j]},j]} = R_{[i.j]}$. Sean también *p, q* píxeles representados por las tuplas RGB siendo $[p_r, p_g, p_b]$, $[q_r, q_g, q_b]$ las componentes azul, verde y roja de *p* y *q*, en este orden. Entonces, se definen las métricas f_c y g_c tal como sigue:

$$f_c = |p_r - q_r| + |p_g - q_g| + |p_b - q_b| \quad (1)$$

$$g_c = \sqrt{(p_r - q_r)^2 + (p_g - q_g)^2 + (p_b - q_b)^2} \quad (2)$$

La ecuación 3 define la contribución de un píxel (k, j) de una imagen k a los pesos de una ventana centrada en (i, j). γc i γs son parámetros de ponderación cuyo valor se ha fijado empíricamente. En nuestra implementación, han tomado los valores 20 y 1 respectivamente. El factor γc depende de la proximidad de los valores de los píxeles en el espacio RGB, mientras que γs modifica la proximidad espacial.

$$f_w(m,i,j,k,l) = e^{(-(\frac{g_c(m_{i,j},m_{k,l})}{\gamma_c} + \frac{\sqrt{(i-k)^2+(j-l)^2}}{\gamma_c}))} \quad (3)$$

El cálculo del mapa de disparidad *D* para cada píxel $D_{i,j}$ se realiza mediante el método *Winner Takes All* mostrado en la ecuación 6, en la que *d* representa el rango de valores en que se comprobará la correlación entre píxeles. El valor de *d* será fijado en relación a parámetros de caracterización de las cámaras y al rango de profundidades del área de trabajo. Por lo que respecta al parámetro *w* de la ecuación 5, indica el tamaño de la ventana de búsqueda. Finalmente, el factor *SW* es un factor usado también en la ec. 5 para influir en las diferencias de posición y color de los píxeles de la ventana.

$$SW = f_w(L,i,j,s,t) f_w(R,i+d,j,s+d,t) \quad (4)$$

$$f_{aw}(L,R,i,j,d) = \frac{\sum_{s=i-w}^{i+w} \sum_{t=j-w}^{j+w} SW \cdot f_c(L_{s,t}, R_{s+d,t})}{\sum_{s=i-w}^{i+w} \sum_{t=j-w}^{j+w} SW} \quad (5)$$

$$f_d(L,R,i,j) = argmin f_{aw}(L,R,i,j,d), d \in \mathbb{Z} \quad (6)$$

Por más que *AW* sea un algoritmo de búsqueda local, es difícil conseguir una *framerate* aceptable en implementaciones para CPU. En cambio, es precisamente su naturaleza local lo que lo hace apropiado para implementaciones para arquitecturas paralelas tales como GPUs. Para nuestra implementación en concreto, se ha utilizado una ventana de 1x15, un nivel de disparidad máximo de 30 píxeles y un tamaño de imagen de 480x270 píxeles. El tiempo correspondiente para estas especificaciones es de 8ms. Como se puede observar, el tamaño de las imágenes utilizadas corresponde a un ¼ del tamaño real de las mismas. Como consecuencia, el siguiente paso consiste en re-escalar la imagen hasta el tamaño inicial. Dicho proceso, junto con las variantes que se han comentado anteriormente sobre el método de adaptive support-weight, introducen algunos defectos en el mapa de profundidad. Dichos defectos o errores se corrigen mediante la aplicación de métodos de pos-procesado. Dichos métodos se comentan en el apartado siguiente.

C. Post-procesado

El objetivo principal de la etapa de post-procesado es la de asignar valores de disparidad correctos a todos los píxeles que han sido detectados o marcados como poco fiables en etapas anteriores. Como se acaba de comentar, la simplificación del algoritmo de adaptive support-weight introduce algunas imperfecciones en el mapa de profundidad. Para intentar minimizar este efecto se ha optado por dos soluciones que aplican por separado. Por un lado tenemos el algoritmo de *cross-checking*. La función del método de *cross-checking* es la de comprobar la consistencia de los píxeles en las imágenes de profundidad izquierda y derecha.

Pero esta solución no es definitiva, pues además de detectar los píxeles erróneos hay que corregirlos y además las regiones donde hay poca textura introducirán errores difíciles de corregir. Para ambos problemas se han implementado dos soluciones. En el primer caso, donde hay que corregir los píxeles erróneos, se ha optado por usar un método basado en difusión anisotrópica para aproximar los valores a una solución mejor. Mientas que en el caso de las regiones homogéneas se ha asumido que estas formaban parte del fondo, el cual se extrae del mapa de profundidad. El método implementado para eliminar el fondo corresponde al descrito en [2] y ha sido implementado sobre GPU para un mayor rendimiento del

sistema. Este proceso de segmentación ha mejorado la calidad del mapa de profundidad evitando los típicos y molestos parpadeos del mapa de profundidad. Para optimizar el uso de ambas soluciones se ha optado por el uso de una máscara.

Una vez realizados estos pasos, todavía podemos intentar mejorar el resultado un poco más. Para ello utilizamos la información sobre oclusión y homogeneidad. Los problemas los vamos a encontrar principalmente en regiones con poca textura o con oclusiones. Un método consensuado para rellenar los huecos es que las oclusiones no deben interpolarse des del objeto que oculta, sino des del objeto ocultado. Para realizar esto, es necesaria la extrapolación del fondo hacia regiones de oclusión. Por contrapartida, los huecos dejados por malos emparejamientos (no por oclusiones) los podemos corregir sencillamente mediante interpolación de valores con los píxeles vecinos. Dicha interpolación puede aplicarse asumiendo que las discontinuidades en las imágenes de disparidad y color están alineadas.

Para poder propagar el valor correcto de disparidad se ha utilizado una variación del método propuesto por Perona-Malik[3]. Este método utiliza un filtrado anisotrópico y como consecuencia de la no-linealidad del filtro, se produce una imagen Gaussiana suavizada. Dicha imagen es el resultado de la ecuación de calor, con un término variable de conductividad que limita el suavizado en los bordes. Para el caso concreto implementado se ha optado por asignar el valor de conductividad como la función correspondiente a la magnitud del gradiente de cada píxel y pada cada canal de color. Además de esta modificación, otro punto donde se ha diferenciado el método respecto al de Perona-Malik ha sido en las iteraciones iniciales. En nuestro caso, se ha optado por modificar solamente aquellos píxeles que eran poco fiables, evitando que influyeran sobre los que se habían marcado como más fiables. Mediante estas iteraciones iniciales se ha pretendido rellenar los huecos de los mapas de disparidad. Posteriormente, una vez los píxeles descartados han convergido, se realiza un número pequeño de iteraciones que modifiquen todos los píxeles (ver figura 2). La utilización del paso de difusión en esta segunda ocasión intenta eliminar el efecto de cuantificación del mapa de disparidad, además de resultar eficiente y sencilla de implementar en arquitecturas de tipo GPU.

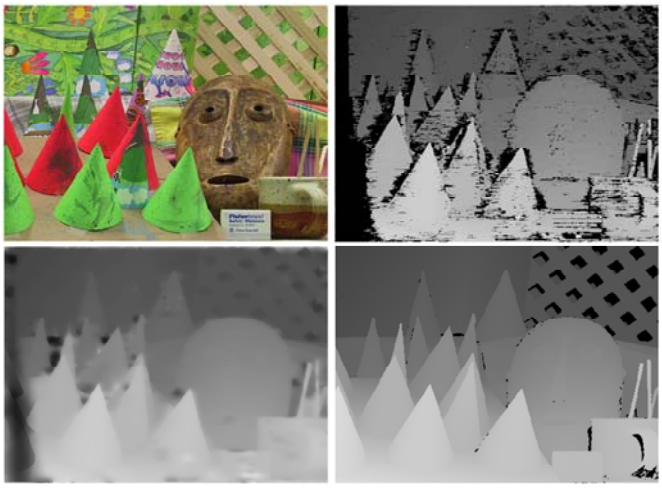

Fig. 2. Post-procesado (De izquierda a derecha y de arriba a abajo): Imagen de referencia. Mapa Inicial de disparidad (los píxeles negros son los incorrectos). Mapa final de disparidad. *Ground Truth*.

III. Resultados

La percepción de profundidad que se tiene cuando se visualiza un contenido a través un monitor 3D refuerza de manera significativa la sensación de volumen y espacio de los objetos visualizados. Esto a su vez, enfatiza la sensación de tele-presencia. En el prototipo utilizado para la realización de nuestros ensayos es un monitor Wow de 42" diseñado por Philips. Este dispositivo es capaz de generar múltiples vistas de una misma escena a partir del formato textura-mas-profundidad y con una resolución de 960x450. Este formato de entrada está estandarizado por el MPEG y está implementado actualmente en muchos monitores 3D de ámbito comercial. Este formato se basa en la utilización de imágenes el doble de grandes de la resolución nativa, donde en el lado izquierdo se sitúa la información de color y en el lado derecho el correspondiente mapa de disparidad o profundidad. Un ejemplo de dicha disposición se puede observar en la Figura 3.

Para estimar la profundidad de una escena y a su vez obtener el mapa de disparidad correspondiente dos o más imágenes de esa misma escena son necesarias. En nuestro laboratorio hemos usado dos cámaras digitales Pixelink para la adquisición de las imágenes correspondientes. Estas cámaras ofrecen la posibilidad de capturar imágenes a una resolución de 1200x600 píxeles con un *framerate* de 25fps. Se ha utilizado el formato RAWBayer que ofrecía la cámara para la adquisición de los datos. Posteriormente, se ha calculado el demosaico de Bayer así como el balance de blancos dentro de un módulo de Gstreamer. El motivo de dicho proceder se debe a la reducción del ancho de banda obtenido mediante el uso de imágenes en formato RAW, cosa que optimiza el uso de memoria y permite un mayor control sobre el demosaico de Bayer.

Nuestro sistema, el cual funciona sobre la plataforma GStreamer [4], está basado en la modularidad. Cada proceso del flujo de datos se puede encapsular en un módulo independiente o plug-in. Esta modularidad ofrece flexibilidad al sistema así como la posibilidad de múltiples configuraciones por parte del usuario. La arquitectura del sistema está dividida principalmente en dos partes. La primera hace las funciones de generador de contenidos o servidor. En nuestro caso, el material generado será del tipo que se va a usar en una videoconferencia, imágenes listas para ser vistas en 3D por ejemplo. La segunda parte representa la parte del cliente.

En la parte cliente es donde se reciben y se muestran las imágenes generadas. A continuación se puede observar con más detalle cada una de las partes.

A. *Lado servidor*

Como ya se ha comentado, el servidor se encara de generar el contenido. Gestiona las cámaras que capturan la escena y los

Fig.3 Imagen de textura-más-profundidad de nuestra solución

módulos de estimación de profundidad y codificación. Las cámaras están sincronizadas de manera externa a través de un hardware específico para obtener una sincronización lo más precisa posible. Esta sincronización mejora la precisión del mapa de disparidad. En la Figura 4 se puede observar un esquema del flujo de datos en la parte del servidor.

El módulo principal, o plug-in, es el que se encuentra etiquetado como Depthestplugin. Este módulo recibe los dos flujos de vídeo de las hilos sincronizados, ejecuta una rectificación de las imágenes y una estimación del mapa de profundidad. La salida del módulo se compone de una imagen de referencia y de su correspondiente mapa de disparidad. Estos flujos se encolan en *buffers* antes de ser codificados usando el compresor de vídeo h.263. Finalmente, los flujos de vídeo codificados se empaquetan en paquetes RTP i se envían a través de un *socket* UDP.

B. Lado cliente

En contrapartida al lado servidor, en el lado cliente no encontramos con el encargado de recibir el material generado. Representa el punto final del flujo de datos, donde el dispositivo de salida (un monitor auto-estereoscópico, por ejemplo) está activo. Tal y como se muestra en la Figura 4, la primera parte del procesado consiste en la decodificación de los flujos de datos que llegan a través de la red. Antes de poder ver el contenido en el monitor, el módulo SideBySide debe modificar la información proveniente de la red para poder ser visualizada correctamente en la pantalla. Básicamente, su rutina se reduce a juntar la imagen de referencia y el mapa de disparidad, ambos a 960x540 pixeles de resolución, en una sola imagen de 1080p que cumpla los requisitos especificados por Philips (ver Figura 3).

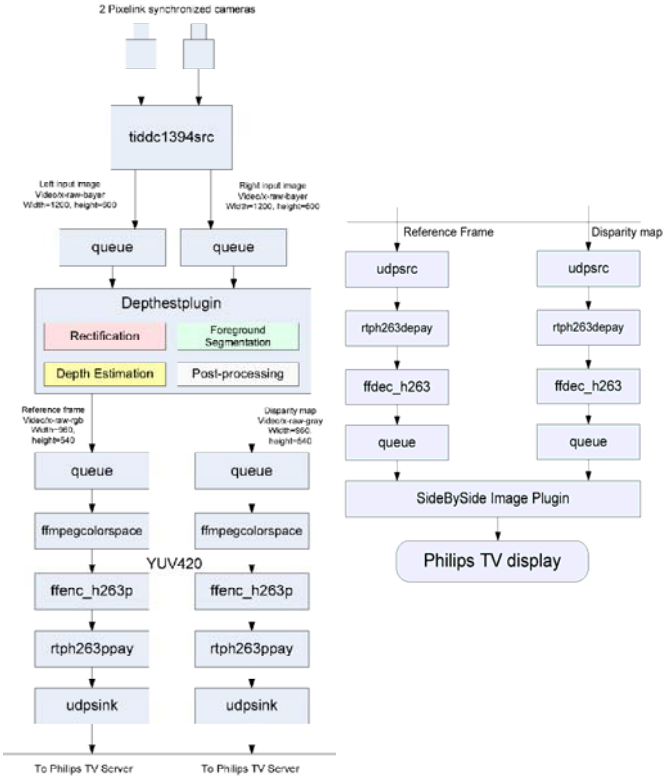

Fig. 4 Esquema de módulos para el lado servidor (esquema izquierdo) y el lado cliente (esquema derecho)

IV. Conclusiones

En el artículo se ha propuesto una solución capaz de funcionar en una aplicación de videoconferencia 3D en tiempo real con el uso de monitores auto-estereoscópicos. Se han implementado y optimizado varios algoritmos sobre GPU tratando de alcanzar el mejor compromiso entre calidad de visualización y coste computacional. La solución desarrollada es capaz de funcionar a 25fps con una resolución estándar (SD) usando un PC con *hardware* convencional (procesador Intel Dual Core con una GPU Nvidia GTX 295) y la plataforma multimedia GStreamer. Este resultado se ha podido obtener gracias a la implementación y optimización del algoritmo Adaptive support-weight. Esta optimización es capaz de calcular mapas de disparidad de manera precisa en un tiempo corto y con un coste computacional bajo. Esta eficiencia nos ha permitido añadir otras mejoras y optimizaciones como la substracción de fondo, la comprobación de consistencia mediante *cross-checking* o algunos mecanismos de corrección de errores, como por ejemplo el rellenado de huecos mediante métodos basados en Anisotropic Difussion. Todos estos algoritmos se han podido desarrollar bajo el entorno CUDA, permitiendo su uso en GPU con el consiguiente aumento de paralelización de los algoritmos, así como el aumento del rendimiento de los mismos.

Actualmente se está trabajando en la incorporación de una cámara de tipo Time of Flight (ToF) en nuestra arquitectura. En el futuro se prevé el uso de la información proporcionada por esta cámara junto con la información que ya nos ofrece el par estéreo. La solución híbrida saliente del uso de ambas tecnologías debería permitirnos mejorar los resultados obtenidos por ambas tecnología por separado. Esto se debe a que con el uso de los dos métodos de obtención de profundidad, se podrían solventar los problemas de cada cámara utilizando la información adicional de la que se dispondría.

Agradecimientos

Este trabajo ha sido parcialmente subvencionado por el CDTI dentro del proyecto CENIT-VISION 2007-1007.

Referencias

[1] Kuk-Jin Yoon and In So Kweon, “Adaptive support-weight approach for correspondence search,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 28, no. 4, pp. 650, 2006

[2] Thanarat Horprasert, David Harwood, and Larry S. Davis, “A statistical approach for real-time robust background subtraction and shadow detection,” in ICCV Frame-Rate WS, 1999.

[3] Pietro Perona and Jitendra Malik, “Scale-space and edge detection using anisotropic diffusion,” Tech. Rep. UCB/CSD- 88-483, EECS Department, University of California, Berkeley, december 1988.

[4] http://www.gstreamer.net, “Gstreamer,”