Rendering Acelerado de Volúmenes en GPU mediante *Splatting*

Gustavo Ramoscelli y Claudio Delrieux

Departamento de Ing. Eléctrica y Computadoras - Universidad Nacional del Sur
Alem 1253 (8000) Bahía Blanca ramoscel@criba.edu.ar

1. Objetivos del Proyecto

Se llama *rendering* de volúmenes al proceso mediante el cual se crea una imagen 2D a partir de un conjunto de valores en un espacio 3D. El conjunto de datos del espacio 3D se lo llama generalmente *dataset* mientras que a cada partición elemental del espacio 3D donde se toma la muestra se la llama *voxel*. Existen varias estrategias para resolver el problema del rendering de volúmenes, entre las que podemos destacar las técnicas de extracción de superficies y el *rendering directo de volúmenes* [1]. La extracción de superficies no es exactamente una técnica 3D sino que más bien preprocesa el dataset con el objeto de encontrar superficies significativas para su visualización que puedan renderizarse utilizando las técnicas usuales de la computación gráfica 3D, mientras que el rendering directo busca una representación genuinamente tridimensional, por lo que debe entre otras cosas resolver el problema del modelo de iluminación en 3D. Es mucho más complejo computacionalmente, pero permite obtener resultados de mayor calidad.

El rendering de volúmenes se aplica a menudo para visualizar imagenes médicas. En este campo los expertos opinan que las imágenes obtenidas con alguna de las técnicas de rendering directo de volúmens son las que presentan mayor calidad de información. Para el rending directo de volúmenes generalmente se utiliza el método de *ray-casting*. Este método consiste en calcular la proyección trazando rayos hacia el espacio del volumen. Cada rayo atravieza un pixel de la imagen y luego de recoger la información de color se pinta el pixel. Otros métodos consisten en proyectar sucesivamente la información del volumen en el plano de la imagen, por ejemplo por medio de la proyección de celdas (*cell by cell*), la superposición de capas semitransparetes de volumen (*slicing*) y la acumulación de la evaluación de la contribución ponderada de cada *voxel* en varios pixels (*splatting*). Se considera que el método más exacto en cuanto a la calidad de la imagen producida es el *ray casting*, pero también el más costoso computacionalmente.

Recientemente, bajo el influjo de la aparición del hardware gráfico acelerado (GPU), el rendering de volúmenes fue objeto de especial interés, donde el objetivo fue lograr sistemas (o adaptar los existentes) para tomar ventaja de la aceleración por hardware. El objetivo de este proyecto está encaminado en esa línea, especialmente en lograr un rendering de volúmenes en tiempo real, es decir, que permita el *interactive steering* o manipulación interactiva de los varios parámetros de renderizado (función transferencia, viewing, iluminación, etc.). En este trabajo mostramos el grado de avance de este proyecto, en el cual evaluamos el renderizado por *ray-casting* implementado en una biblioteca de dominio público, con una implementación desarrollada en nuestro laboratorio de renderizado por *cell splatting*. Se compara la calidad obtenida y el tiempo de renderizado de un mismo dataset, y se discuten algunos detalles de la implementación.

2. Ray Casting

El *ray casting* evalúa cada pixel de la imagen final, mediante la proyección de rayos que parten desde el punto del observador y atraviezan los pixeles del plano de la imagen [1]. Cada rayo se compone de los puntos $\mathbf{x}(\lambda)$ donde $\mathbf{x}$ son puntos del espacio y λ es el parámetro utilizado para obtener estos

puntos. Si el rayo atravieza el *dataset*, entonces se calcula la intensidad de la luz que el volumen aporta al pixel (fragmento) mediante la ecuación del rendering directo de volúmenes aplicado a objetos semitransparentes.

Para la evaluación del rayo, primero se traza un rayo desde el observador hacia el espacio ocupado por volumen de tal forma que atraviece un pixel de la imagen. Si este rayo itersecta uno o más *voxels* que pertenecen al *dataset*, se calcula el valor de varias muestras en distintos puntos del rayo y se guardan estos valores en una lista ordenada. Si el muestro del rayo es uniforme, estos valores serán interpolados. Si en cambio el muestro consiste en dicretizar el rayo en 3D coincidiendo con los *voxels* del *dataset*, los valores serán los muestreados en cada *voxel*. En este caso los valores no necesitan interpolarse. Luego con la información de la lista se calcula tambien el valor de la atenuación en cada punto y finalmente se recorre en forma inversa el rayo (desde el volumen hacia el observador) para realizar el cálculo del color del pixel.

El método de ray casting es sin duda el favorito cuando se busca alta calidad de imagen. Sin embargo, utilizando métodos de cálculo convencionales (CPU) el tiempo requerido para generar la imagen es muy alto. Por esto, cuando se busca interacción con el usuario, se implementa alguna aproximación más rápida pero que generalmente crea imagenes de calidad inferior. Algunos autores han mostrado como acelerar el cálculo del *ray casting* mediante una implementación parcial del algoritmo compilada en procesadores gráficos (GPU) [2][4]. De esta forma parte del cálculo se realiza en CPU y parte en GPU.

3. Proyección de Celdas, Slicing y Splatting

Los algoritmos de proyección del volumen sobre la imagen más utilizados son: proyección de celdas, *slicing* y *splatting*. Estos métodos se los clasifica como *back-to-front* ya que el cálculo se efectúa se comienza por la parte más lejana del volumen hasta la parte más cercana (desde el punto de vista del observador). En la proyección de celdas, cada celda es conformada por ocho muestras de puntos conexos pertenecientes al *voxel* a representar. La celda es luego proyectada hacia la pantalla y el color de cada pixel se compone de la contribución calculada de esta celda más la contribución de las celdas proyectadas anteriormente.

La técnica de proyección de capas o *slicing* se basa en utilizar planos que cortan el volúmen que se va a representar. Los planos se pintan entonces con una textura que representa los valores de muestra del volumen y luego se los proyecta en forma ordenada de atras hacia adelante, sobre el plano de la imagen, calculando la acumulación y atenuación de energia mediante de unos sobre otros. En sistemas con aceleración gráfica, los valores de muestro del volumen se guardan en una textura 3D y a partir de esta textura se calcula en forma inmediata la textura 2D que pinta cada plano mediante algún método de interpolación. La acumulación de los planos se realiza mediante funciones de *blending*.

El método de *splatting* aparece cuando se considera que cada punto del muestreo del volumen es una fuente de energía [6]. El algoritmo consiste entonces en ir acumulando de atras hacia adelante las proyecciones de energía. Se considera además que la muestra atenúa la proyección de las fuentes de energía que estan detras. De esta forma se calcula el valor del color del fragmento como la contribución de energía de la nueva fuente componiendola con la energía calculada hasta ese momento multiplicada por el factor de atenuación de la nueva fuente.

Para el cálculo de la contribución de una muestra se toma como patron una imagen 2D formada por la emisión de energía que teóricamente produce una fuente puntual de intensidad unitaria y distribución de energía Gaussiana. Esta imagen se la conoce como *footprint* [7]. El cálculo de la contribución se obtiene al ponderar el valor de color de cada pixel del *footprint* con el valor de energía de la muestra. El resultado es una imagen 2D donde cada pixel corresponde a la emisión de energía de la muestra. Esta imagen se la conoce como *splat*. El *splat* se coloca entonces en el punto del espacio de la muestra. Si la proyección es paralela, se la orienta paralela a la imagen y se proyecta sobre la imagen. Si la proyección es perspectiva, el *splat* se orienta de frente a la cámara u observador y se proyecta sobre la imagen.

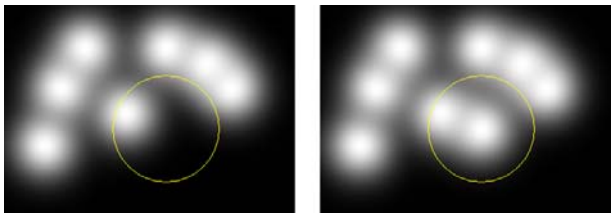

Figura 1: Ejemplo de acumular un nuevo *splat*.

4. Hardware Gráfico Actual

Las placas aceleradoras gráficas 3D para computadoras personales comenzaron a volverse más sofisticadas. En un principio, las placas gráficas comenzaron por incluir potencia de cálculo para realizar muchas de las aproximaciones existentes para la representación de una escena 3D. Luego, las placas gráficas posteriores almacenaban múltiples texturas en memoria de video las que se componían eligiendo alguna de las diversas funciones de blending disponibles para lograr efectos complejos en la escena mostrada. Las siguientes generaciones de placas gráficas permitieron ejecutar programas en los procesadores de vértices y fragmentos. A la unidad o chip que contiene ambos tipos de procesadores se la llama GPU.

El rendering directo de volúmenes tambien puede implementarse parcialmente en hardware. Para esto se efectúa el cálculo de la ecuación de iluminación en diversas etapas usando información almacenada en recursos disponibles como por ejemplo las texturas 3D. También se pueden implementar parcial o totalmente otras técnicas menos exactas que el rendering directo, pero que alcanzan aproximaciones aceptables y buena calidad en la imagen generada.

5. Implementación del Método de Splatting

Para implementar el programa de rendering de volúmenes mediante la técnica de *splatting* se recurrió al uso de técnicas ya documantadas en trabajos anteriores y se agregaron técnicas desarrolladas *ad hoc* para optimizar el tiempo de ejecución y el espacio de memoria requerido. El objetivo de esta aproximación al rendering de volúmenes es utilizar una figura llamada *splat* que representa la contribución de energía de una muestra del volumen a representar. Esta muestra nueva se proyecta en la imagen que se está generarando y se acumula con las proyecciones anteriores. Al mismo tiempo se calcula también la atenuación causada a la energía calculada anteriormente por la presencia de esta nueva muestra. La ventaja de proyectar de atras hacia adelante es el cálculo de la atenuación ya que al proyectar un nuevo *splat* ya se conoce la información del volumen que esta detras. Entonces, al ir pintando pixel a pixel el *splat* nuevo se toma la información de color del fragmento (que representa de la energía acumulada), se calcula su valor atenuado y se le agrega la energía nueva aportada. El resultado se guarda en el fragmento hasta que sea tocado por otro *splat*. De esta forma el *frame buffer* trabaja como memoria de la energía parcialmente acumulada. En las arquitecturas de GPU modernas, se puede utilizar el procesador de fragmentos mediante un programa *CG* para realizar este cálculo. En la Fig. 1 se muestra un ejemplo de rendering realizado hasta cierto punto del algoritmo de *splatting*.

La información del volúmen muestreado se guarda en un VBO, junto con la información de índices. Luego de varios ensayos, se encontró que no hay diferencias en los tiempos de ejecución si se divide la información en múltiples VBO o si se utiliza un solo VBO. Por esta razón se optó por utilizar un solo VBO para almacenar toda la información del volumen.

El orden de las proyecciones de un muestreo regular para el caso general una ubicación arbitraria del observador se puede reducir a solo uno de ocho direcciones principales. Cada dirección indica el orden de la proyección de los *splats*. En la figura de abajo se muestra un cubo que representa el volumen muestreado y siete de estas direcciones representadas con las flechas en azul que salen del cubo. Estas direcciones corresponden a un orden distinto de los *voxels* del espacio según los ocho octantes que se

forman tomando como centro del espacio el centro del cubo y con planos paralelos a las caras del cubo. Luego según el octante en el que se encuentra el observador, se proyectarán los *splats* en la dirección que corresponda.

Para ahorrar espacio en memoria, solo se deben almacenar los índices ordenados de los ocho casos posibles. Luego, al momento de realizar el rendering se detemina cuáles índices deben usarse para realizar la dirección de las proyecciones en forma correcta. En general se utilizan capas (*slices*) del volúmen que se ordenan segun un eje en coordenadas de muestreo. Estas capas contienen se dividen a su vez en rectángulos que sirven para generar cada *splat*.

Finalmente para la correcta representación de cada rectángulo, deben rotarse manteniendo el centro hasta que el normal de cada uno apunte al observador. Este proceso se denomina alineación y generalmente alcanza con alinear los *splats* en forma paralela al plano de la imagen [5]. Para este cálculo se utiliza el procesador de vértices. Al momento de procesar un vértice, un programa CG toma el vértice y lo modifica en el espacio para generar un rectángulo resultante que tenga la dirección normal deseada.

6. Comparación de los Métodos

Para hacer una comparación del método de *splatting* con el rendering directo de volúmenes mediante *ray-casting* se implementó por un lado en forma independiente un programa de *splating*. Por otro lado se utilizó el programa Volview[1] que utiliza las librerías vtk[2] que implementan el rendering de volúmenes mediate *ray-casting*[3].

El programa Volview logra una sensación de tiempo real mediante muestrar la imagen en distintas etapas de refinamiento mientras el usuario ajusta distintos parámetros del rendering (como por ejemplo las funciones de tranferencia y la posición del observador). De esta forma, al ajustar cualquier parámetro, el programa nos mostrará una imagen inicial con poco detalle. Si el parámetro no se vuelve a variar, pocos segundos más tarde se mostrará la imagen final el máximo nivel de detalle.

Para la implementación del método de *splatting* se implementó un programa en lenguaje *obejct Pascal* utilizando el entorno de desarrollo *Delphi*. Para la aceleración por GPU se optó por utilizar CG frente a otras opciones como HLSL.

El *dataset* utilizado es el ejemplo de VolView de una imagen CT. Para utilizar este *dataset* en el programa de *splatting* se lo exportó a formato binario utilizado un archivo por *slice*. El tamaño del *dataset* es de 128x128x92 muestras.

Los resultados obtenidos muestran que las conclusiones de distintos autores son exactas, al notar la diferencia de velocidad de rendering a costa de perder un poco de detalle en la imagen. Respecto de las imagenes mostradas en la figura 2 de abajo, debemos notar que Volview y el programa de *splatting* implementado ofrecen distintos tipos de ajuste. Por esta razón se procuró ajustar las funciones de transferencia de ambos programas para obtener imágenes similares, las cuales siempre tendrán alguna diferencia de tonalidad.

En cuanto a los tiempos de ejecución, el algoritmo de *splatting* alcanzó una velocidad de alrededor de 2 a 2,5 frames por segundo con una imágen de 512x512 pixels. En el caso de Volview la velocidad de imagen final es de 0,4 frames por segundo. Si bien se notan diferencias en el brilo, el contraste y el color, el nivel de detalle es bastante similar. Para realizar estos ensayos se utilizó una tarjeta GeForce 6800 montada en un slot PCI Express.

7. Conclusiones y Trabajos Futuros

Podemos destacar que la velocidad del algoritmo de *splatting* fué muy superior al del rendering directo de vtk en un factor entre 5 y 6. Sin embargo, como vtk utiliza una técnica de refinamiento de imagen y muestra parcialmente el resultado, al usuario le parece que esta trabajando en tiempo

[1] El programa Volview es propiedad de Kitware Inc.
[2] Las librerías VTK son propiedad de Kitware Inc.

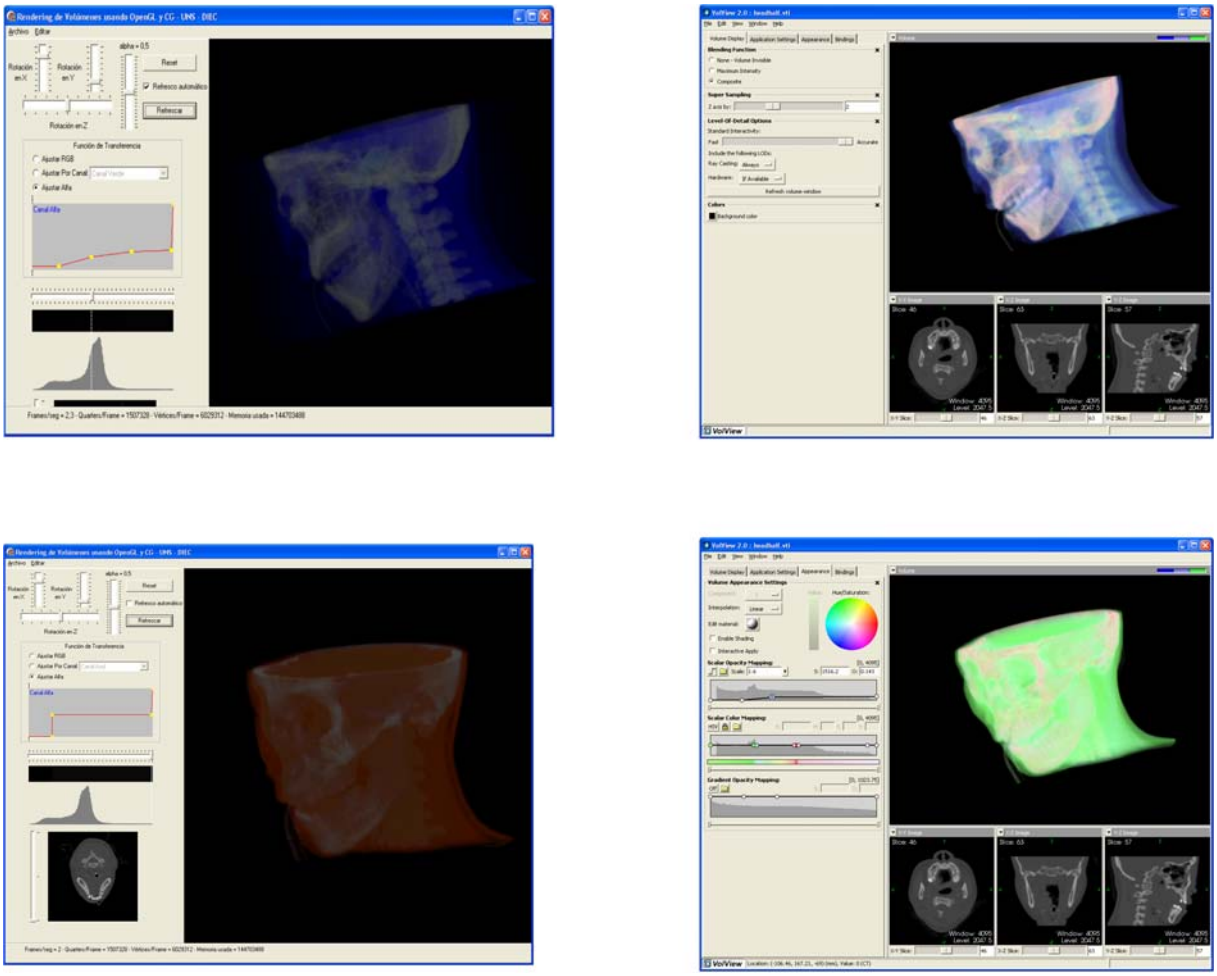

Figura 2: Imágenes generadas utilizando *splatting* a la izquierda y *ray-casting* a la derecha.

real cuando realiza los distintos ajustes. Por esta razón en una futura implementación se agregará la técnica de refinamiento de imagen en el programa de *splatting*.

Otros trabajos que planeamos realizar son la implementación de aceleración de GPU en los métodos de *slicing* y rendering directo. De esta forma, se podrán obtener resultados más precisos al comparar los distintos métodos. Finalmente se explorará una mejora en la interface para definir las funciones de trasferencia de color y de opacidad.

Referencias

[1] Mark Watt Alan Watt. *Advanced Animation and Rendering Techniques: Theory and Practice*. ACM Press, New York, 1992.

[2] Ertl T. Engel K., Kraus M. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In Inc. Addison-Wesley Publishing Company, editor, *Eurographics / SIGGRAPH Workshop on Graphics Hardware '01*, pages 9–16, 2001.

[3] Kitware Inc. *The Visualization Toolkit Documentation*. Kitware Inc., Enero 2008. http://www.vtk.org/doc/release/5.0/html/.

[4] R. Westermann J. Kruger. Acceleration techniques for gpu-based volume rendering. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 38, Washington, DC, USA, 2003.

[5] K. Mueller N.Ñeophytou. Gpu accelerated image aligned splatting. In I. Fujishito (Editors) E. Groller, editor, *Volume Graphics*, The Eurographics Associations 2005, 2005.

[6] Lee Westover. Interactive volume rendering. In *VVS '89: Proceedings of the 1989 Chapel Hill workshop on Volume visualization*, pages 9–16, New York, NY, USA, 1989. ACM.

[7] Lee Westover. Footprint evaluation for volume rendering. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 367–376, New York, NY, USA, 1990. ACM.