CEIG'08, Barcelona, Sept. 3–5 (2008)
L. Matey and J. C. Torres (Editors)

Proyección esférica usando la programación en la GPU

A. Amundarain[1], C. Buchart[1],A. Valero[1],I. Aguinaga[1] y L. Matey[1,2]

[1]CEIT, San Sebastian
[2]Tecnun (Universidad de Navarra), San Sebastian

Abstract

Existen una gran cantidad de dispositivos de visualización que se emplean en los sistemas de realidad virtual; desde dispositivos de pantalla plana hasta sistemas que ofrecen una sensación de inmersión completa. Entre estos sistemas se puede encontrar los sistemas de proyección esférica, que presentan un buen grado de inmersión.
Sin embargo, el tipo de proyección necesario para este tipo de sistemas no está soportado directamente por el hardware gráfico actual. Las mejoras en la programabilidad de este tipo de hardware permiten implementar los métodos necesarios para usar sistemas de proyección esférica de forma efectiva.
En este artículo se describe un método para implementar las proyecciones necesarias en este tipo de sistemas usando el hardware gráfico. Así mismo, se presentan mejoras y métodos que permiten mejorar la eficiencia del método y la calidad de las imágenes obtenidas.

Categorías y descriptores de tema (según ACM CCS): I.3.3 [Gráficos por ordenador]: Algorítmos de Visualización I.3.7 [Gráficos por ordenador]: Realidad Virtual

1. Introducción

Existe una amplia gama de sistemas de visualización que se emplean habitualmente en los sistemas de realidad virtual. Se puede considerar que los sistemas de visualización más simples son aquellos basados en proyecciones planas como la pantalla de un ordenador. En este tipo de sistemas la imagen se proyecta en una única pantalla plana. Su ventaja reside en su simplicidad y bajo coste, pero como contrapunto, estos sistemas ofrecen un grado de inmersión en el entorno virtual bajo. Además, el campo de visión del usuario es limitado. En estos sistemas se puede incrementar el grado de inmersión empleando sistemas de visualización estereoscópica, ya que de esta forma se mejora la sensación de profundidad. Sin embargo, el uso durante mucho tiempo de este tipo de sistemas es complejo debido a la carga que sufre el usuario a causa de pequeños desajustes entre las proyecciones que genera el sistema virtual y las que se ajustan realmente a la anatomía del usuario.

En el otro extremo se encuentran los sistemas que permiten una inmersión completa en el entorno virtual. Los periféricos visuales están formados por *displays* muy cercanos a los ojos del usuario como los cascos de Realidad Virtual (Head Mounted Displays o HMD), o binoculares orientables (BOOM), o por sistemas tipo CAVE que se basan en la generación de entornos envolventes gracias a una configuración especial de múltiples pantallas. Aunque estos dispositivos mejoran sensiblemente el grado de inmersión obtenido, su alto coste hace que su aplicación se reduzca a entornos en los que el coste total sea importante.

Un dispositivo de visualización que ofrece un grado de inmersión intermedio es el VisionStation (Figura 1). Este dispositivo consiste en una pantalla semiesférica. Sobre ella, un proyector muestra las imágenes generadas del mundo virtual que se adaptan a la pantalla mediante una lente de ojo de pez. Estos dispositivos ofrecen dos ventajas sobre las pantallas planas. Al pasar de pantallas 2D a pantallas 3D, se logra incrementar la sensación de profundidad, mejorando la sensación de inmersión. Por otro lado, la pantalla y la proyección esférica permiten un ángulo de campo de visión de entre 140^o-180^o. Según el modelo con el que se este trabajando; permitiendo también disponer de visión periférica.

La generación de las imágenes en estos dispositivos difiere, no obstante, de los métodos tradicionales empleados en entornos virtuales [RvBWR04]. En este caso es necesario el uso de una proyección esférica para generar una imagen que al ser deformada por la lente de ojo de pez y a contin-

Figura 1: *Dispositivo de Visualizacion VisionStation.*

uación ser proyectada en la pantalla semiesférica produzca el resultado adecuado. Desgraciadamente, este tipo de proyección no se encuentra integrado dentro del *pipeline* de dibujado tradicional. Sin embargo, las tarjetas gráficas de última generación permiten su programación, lo que permite implementar los algoritmos necesarios para generar la imagen esférica. Esta imagen se generará a partir de imágenes creadas empleando proyecciones en perspectiva.

En este artículo se presenta un método para generar imágenes con la proyección adecuada. Además, se presentan métodos que permiten la reducción del tiempo necesario para generar la imagen corregida, y mejorar la calidad de la misma. Se presenta una técnica de *anti-aliasing* que se adecua a las características de las proyecciones esféricas . El renderizado de las imágenes se basa en la librería gráfica OpenGL [MB05] mientras que los métodos que se han programado sobre la tarjeta se han realizado empleando el lenguaje Cg [RK03].

2. Proyección esférica

2.1. Dispositivo VisionStation

El dispositivo de visualización VisionStation consiste en una superficie de proyección grande y curvada, un proyector de alta resolución y una lente de proyección de gran ángulo.

Como resultado, se obtiene un sistema de visualización que ofrece un amplio ángulo de visión de hasta 180^o. Esto permite que el dispositivo abarque todo el ángulo de visión del usuario, incluyendo su visión periférica. La capacidad de visualizar la periferia es muy útil en muchas aplicaciones de realidad de virtual.

Este amplio campo de visión se logra mediante la tecnología conocida como TruTheta. Esta tecnología combina avanzados algoritmos de software 3D junto a lentes f-theta [JSY06]. Esto permite tener una distribución uniforme de píxeles en la pantalla. La lente está diseñada para lograr que el ángulo de proyección de cada píxel sea linealmente proporcional a su distancia relativa al centro de proyección. Como resultado, se obtiene una imagen enfocada y de gran calidad.

2.2. Proyección de ojo de pez

Entre las infinitas formas en que se puede mapear una vista con una apertura angular elevada en una imagen, en los gráficos por ordenador se suelen emplear habitualmente dos: ojo de pez angular y ojo de pez hemisférica [Bou04]. Ambas formas se han implementado en este trabajo y la incorporación de otros tipos de proyecciones no sería costosa.

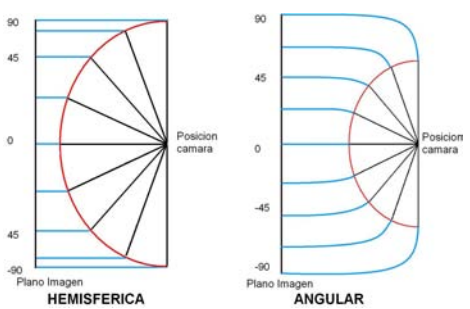

Figura 2: *Tipos de proyección ojo de pez*

La proyección hemisférica consiste en una proyección en paralelo del hemisferio sobre un plano (Figura 2), teniendo como resultado una imagen circular. El máximo de campo de visión que se puede lograr es de 180^o, teniendo una distorsión grande cercano a +/- 90^o.

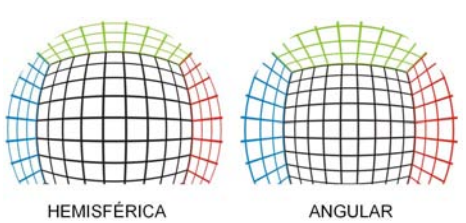

Figura 3: *Proyección hemisférica y angular*

La proyección angular se define de forma que la distancia al centro de la imagen es proporcional al ángulo que se forma

con la dirección de vista de la cámara (Figura 2). Este tipo de proyección se pueden emplear para ángulos de hasta 360º. En esta proyección la resolución es similar en toda la imagen y de esta forma se evitan las grandes distorsiones.

En la Figura 3 se puede observar la imagen que se obtiene del interior de un cubo texturado empleando los dos tipos de proyección, empleando un ángulo de apertura de 140º.

3. Descripción del método

El primer paso consiste en generar imágenes que abarquen todo el campo de visión que se va a visualizar en la proyección esférica. En nuestro dispositivo visual este campo de visión es de 140º, pero en el caso de que el campo visual fuese de hasta 180º, no habría ninguna modificación en el método. Estas imágenes se generan mediante proyección en perspectiva. Lo ideal, sería poder conseguir una única imagen que abarcase todo el campo de visión. Sin embargo, el empleo de una pirámide de visión con un ángulo de apertura de 140ºno es recomendable. El volumen de visualización que se obtiene de esta forma es muy grande, y al proyectarlo en una imagen con una resolución común (p.e. 2048x2048) la precisión que se logra es muy baja y la calidad de la imagen no es adecuada. Esto implica la necesidad de emplear más de un pase de dibujado. En cada pase se dibuja una parte del volumen de visualización, de forma que al final se obtiene todo el volumen de visualización representado en diferentes imágenes.

El número de pases necesarios depende del ángulo de visión requerido en la proyección esférica. En el caso de que el ángulo sea de 90º, dos pases son suficientes. Con un sistema con un ángulo de visión de 140ºes recomendable emplear un pase más y en el caso de que el ángulo sea de 180º, lo recomendable es emplear cuatro pases para lograr una calidad adecuada.

Una vez que se han obtenido las imágenes, estas deben ser pasadas como argumento al programa encargado de realizar la proyección esférica [OD07]. Para ello es necesario almacenar estas imágenes como texturas. La forma más efectiva de lograr esto es empleando una técnica de renderizado en textura. Este tipo de técnicas permiten renderizar la imagen de una escena directamente en una textura, ahorrando costosas operaciones de transferencia de datos entre la CPU y la GPU. Con este fin se han empleado los *Frambuffer Objects* de OpenGL (FBO) a los que se asignan tantos canales de color como pases se vayan a realizar. A cada canal de color se le asigna una textura, y en cada pase de dibujado se activa un canal de color diferente. Al realizar los pases necesarios se almacena en cada una de las texturas la imagen correspondiente a una de las partes en que se ha dividido el volumen de visualización de la proyección esférica (Figura 4).

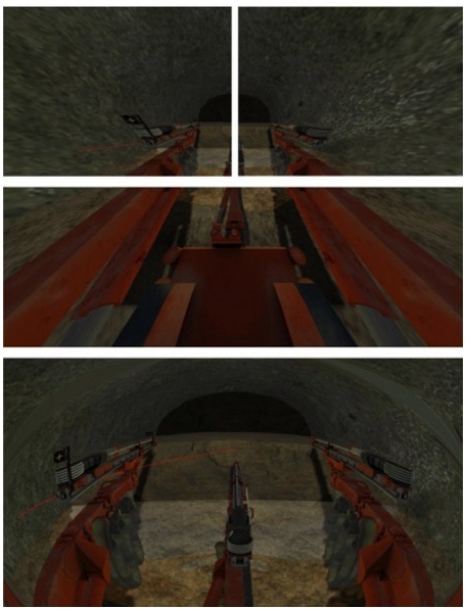

Figura 4: *Imágenes generadas mediante proyección perspectiva y la imagen final que se genera mediante proyección esférica*

3.1. Configuración de las pirámides de visión

Todo el volumen de visión que se vaya a proyectar de forma esférica tiene que estar ocupado por las pirámides de proyección empleadas en los diferentes pases. Estas pirámides se pueden organizar empleando diferentes configuraciones (Figura 5).

Por un lado se pueden emplear pirámides de proyección simétricas, a las que se aplica una rotación para orientarlas de forma que ocupen todo el volumen de visión. Por otro lado, se pueden emplear pirámides asimétricas, en cuyo caso, se adecua la forma de las pirámides para rellenar el volumen de visión deseado, sin necesidad de realizar ninguna rotación de las mismas.

La primera opción puede ser la más intuitiva, pero presenta ciertas limitaciones, por ejemplo, a la hora de iluminar el escenario. La iluminación especular depende de la dirección de visión y al rotar las pirámides se modifica está dirección. Como resultado, se tiene una iluminación diferente en cada pase. Este efecto se puede solucionar programando un *shader* que se encargue de la iluminación. Sin embargo, este inconveniente no aparece en el caso de emplear pirámides de proyección asimétricas, pues al no rotar no se modifica la dirección de visión y, por lo tanto, no es necesario recalcular la iluminación.

Esta razón nos ha impulsado a emplear la segunda configuración. Este método también sirve para la primera configuración, una vez solucionado el problema de la iluminación.

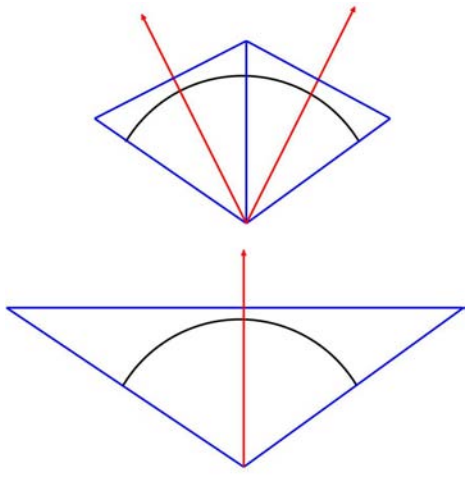

Figura 5: *Pirámides simétricas rotadas vs. Pirámides asimétricas*

3.2. Metodología de la proyección esférica

La técnica empleada para generar la imagen esférica es similar a la de *ray tracing* [PBMH02]. En este caso, sin embargo, no se emplea la escena virtual a la hora de identificar las intersecciones de los rayos, sino que el escenario que se emplea está compuesto por las pirámides de proyección obtenidas en las proyecciones previas.

Por cada píxel de la imagen se lanza un rayo. Primero se identifica en que pirámide de proyección entre las empleadas en los pases previos se encuentra este rayo. A continuación, se calcula la intersección del rayo con el plano de fondo de la pirámide. Tal como se ha comentado previamente, la imagen generada por cada pirámide de proyección se almacena en una textura. A cada rayo de intersección calculado le corresponde un téxel de la textura (o una combinación de varios dependiendo del filtrado de la textura). El color del téxel es el color que se representa en el píxel desde el que se ha lanzado el rayo.

Todos los cálculos de intersecciones se realizan en un *shader* y por lo tanto son computados por la tarjeta gráfica. A cada *shader* se le pasa toda la información necesaria para realizar sus cálculos. Entre estos datos se encuentra la información que describe las diferentes pirámides de proyección y también las imágenes generadas en los pases previos. Se han diseñado diferentes *shaders*, dependiendo del número de pases previos necesarios. El motor gráfico se encarga de indicarle a la GPU que *shader* corresponde en cada ocasión.

Una vez que se ha generado la imagen final, para mejorar su calidad, se aplica una técnica de *antialiasing*. Se han diseñado diferentes técnicas que tienen en cuenta las propiedades geométricas de la proyección esférica

3.2.1. Lanzamiento de rayos

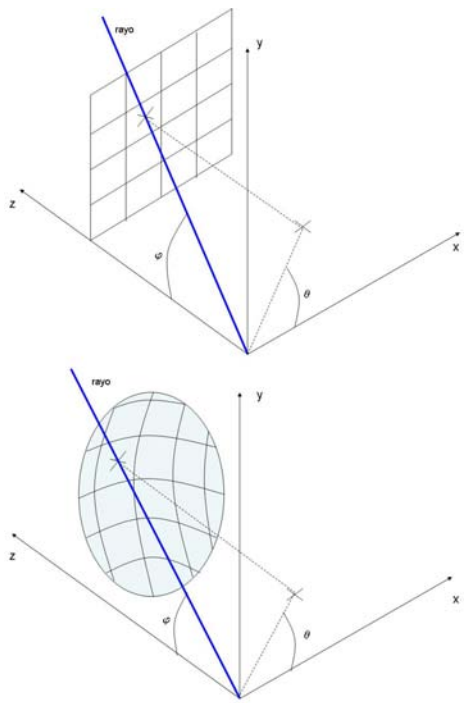

Figura 6: *Ray trancing perspectiva vs. Ray tracing esférica*

Para generar la imagen se lanza un rayo por cada píxel de la imagen final. Para lograr esto se renderiza un cuadrado texturado cuya proyección ocupa todo el área de la imagen. De esta forma, todo el área de la imagen se rasteriza y se tiene la opción de ejecutar un programa por cada píxel de la imagen (*pixel shader*). Este programa es el encargado de calcular la dirección del rayo, identificar la intersección del rayo con las pirámides de proyección y calcular el color que le corresponde al píxel (Figura 6).

Las dos proyecciones esféricas habituales son la proyección ojo de pez esférica y la proyección ojo de pez angular. En el método desarrollado, dos tipos de proyecciones únicamente difieren a la hora de calcular la dirección del rayo que corresponde a cada píxel.

3.2.1.1. Dirección del rayo en la proyección ojo de pez hemisférica. La dirección del rayo es determinado por el punto de vista (0,0,0) y la posición espacial que representa cada píxel (x,y,z).

En la proyección hemisférica se realiza una proyección paralela de la semiesfera sobre el plano. Los valores de x e y vienen determinados por la posición del píxel analizado. El valor de z se obtiene empleando la ecuación de la esfera:

$$x^2 + y^2 + z^2 = R^2$$

La dirección del rayo se define mediante los ángulos φ y

θ. Estos ángulos se calculan empleando las siguientes ecuaciones.

$$\varphi = atan2\left(\sqrt{x^2+y^2}, z\right)$$

$$\theta = atan2\left(y, x\right)$$

3.2.1.2. Dirección del rayo en la proyección ojo de pez angular. En este caso el ángulo φ viene determinado por la distancia del píxel al centro de la imagen. De este modo las ecuaciones empleadas para calcular los ángulos que definen el rayo son:

$$\varphi = (apertura/2) * sqrt\left(x^2+y^2\right)$$

$$\theta = atan2\left(y, x\right)$$

Donde *apertura* es el ángulo de apertura, en el caso analizado 140^o.

3.2.2. Intersección rayo - pirámides de proyección

Una vez que se ha definido el rayo, se ha de calcular la intersección entre el rayo y las pirámides de proyección. Para ello, se pasa la información de cada pirámide de proyección al programa que se ejecuta en la tarjeta gráfica.

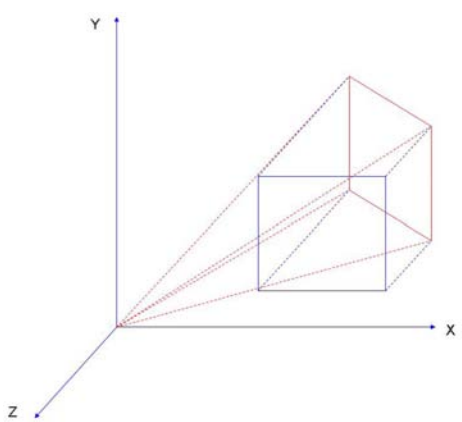

Figura 7: *Proyección del plano de profundidad de la pirámide en el plano XY del sistema de coordenadas de la cámara*

A cada pirámide se le asigna una profundidad igual a 1. Este valor es indiferente, pues lo esencial es identificar la intersección en relación a los cuatro vértices que definen el plano base de la pirámide; esta posición relativa no depende de la profundidad del plano. Una vez establecida la profundidad, los cuatro puntos que definen las pirámides se encuentran definidos. A continuación, se proyectan estos puntos sobre el plano xy del sistema de coordenadas de la cámara. Estos cuatro puntos definen un paralelepípedo. La información que define este paralelepípedo es la que se pasa al programa (*shader*). También se le pasan las direcciones de visión de las pirámides. En el caso de emplear pirámides asimétricas esta información se simplifica, pues la dirección de visión será (0,0,1) y el paralelepípedo será un rectángulo (Figura 7).

A continuación, se detallan lo pasos que se realizan en el *shader* que se ejecuta en la tarjeta gráfica. El primer paso consiste en identificar la intersección entre el rayo y el plano de profundidad de la pirámide. La dirección de este rayo se define mediante los ángulos φ y θ y el plano de profundidad es perpendicular a la dirección de visión de la pirámide (n) y se encuentra situado a una distancia 1. El valor calculado es la distancia que ha de recorrer el rayo hasta la intersección. Este valor se calcula mediante la siguiente ecuación:

$$d = \frac{1}{sin(\varphi)*cos(\theta)*n.x+sin(\varphi)*sin(\theta)*n.y+cos(\varphi)*n.z}$$

Una vez que se ha definido el punto de intersección, se proyecta sobre el plano xy del sistema de coordenadas de la cámara:

$$x = d * sin\left(\varphi\right) * cos\left(\theta\right)$$

$$y = d * sin\left(\varphi\right) * sin\left(\theta\right)$$

Por último, se busca la posición relativa del punto proyectado en el paralelepípedo proyectado que definía a la pirámide proyección. Esta posición relativa sirve como coordenada de textura y al acceder a la textura con esta información se obtiene el color que corresponde al píxel que se está procesando y por lo tanto el color que aparecerá en la representación final en esa posición.

3.3. Proyección fuera de eje

Un aspecto a considerar al realizar la proyección, es la posición desde la que el usuario observa la pantalla hemisférica. Al no coincidir el punto de vista del usuario con la posición de la cámara, se han de realizar algunas modificaciones en la proyección para que el usuario pueda observar una imagen esférica correcta.

En un apartado previo se ha comentado que los rayos se generan desde el punto de vista de la cámara. Para corregir la proyección generada debido a la no coincidencia del punto de vista con el punto de cámara, los rayos se lanzan desde el punto de vista del usuario. En definitiva, la generación de los rayos empieza desde el punto de vista del usuario en vez de la posición del proyector.

Al variar el punto de vista, las dimensiones de los volúmenes de visión también sufren variaciones, pues el volumen de visión desde estos dos puntos no es exactamente igual.

3.3.1. Optimización de los cálculos

El *shader* que realiza la proyección se ejecuta por cada píxel y eso supone que en una imagen de resolución 1024x768 se procese alrededor de un millón de veces por cada imagen. Este aspecto hace que sea muy recomendable la reducción

de los cálculos que se realizan en el *shader* para mejorar el tiempo de refresco [Ngu07].

Entre las operaciones que se han descrito hasta ahora, varias de ella se repiten al generar cada imagen. Una de estas operaciones es el cálculo de los ángulos φ y θ para cada rayo. La posición de los píxeles siempre es la misma, nunca va a variar, luego los rayos que pasan por cada píxel también son invariantes. En lugar de calcular estos ángulos para el rayo por cada nueva imagen es más efectivo calcularlos en una etapa de preproceso inicial. En esta etapa se calculan los ángulos que se corresponden con cada rayo. Esta información se almacena en una textura de números de coma flotante. Esta textura se pasa al *shader* como información de entrada lo que permite tener acceso a esa información con un único acceso a memoria sin tener que realizar ninguna operación.

Algo similar se puede realizar a la hora de realizar operaciones como cosenos y senos de estos ángulos. Estos datos son necesarios a la hora de identificar la intersección entre el rayo y la pirámide y a la hora de realizar la proyección del punto de intersección al plano XY del sistema de coordenadas de la cámara. Sin embargo, estos valores son siempre constantes para cada uno de los píxeles de la proyección. De igual manera, es recomendable calcular estor valores en una etapa de preproceso, almacenar los datos calculados en una textura y pasar esta textura como dato de entrada al programa que se ejecuta en el hardware gráfico.

3.3.2. Anti Aliasing

El *aliasing* se refiere a las líneas dentadas y otros efectos de renderizado presentes en las imágenes generadas por ordenador. Estos efectos son debidos a las simplificaciones que se incorporan en varios algoritmos presentes en el proceso de dibujado, las cuales generan inexactitudes en la imagen final. Estas simplificaciones suelen ser necesarias para generar una imagen con bajo coste y que permita obtener frecuencias de refresco adecuadas. Este efecto suele aparecer cuando la resolución de la imagen no es lo suficientemente alta para representar los detalles de alta frecuencia presentes en la señal de entrada.

Una forma de subsanar este problema es incrementando la resolución del *buffer* de color y luego aplicar un postfiltrado para obtener el color del píxel final. De esta forma el color de cada píxel no viene determinado por una única muestra que suele ser el del centro del píxel. En este caso se calculan los valores de otras ubicaciones en el píxel y el valor final se calcula a partir de la media ponderada de todos los valores calculados.

La calidad del filtrado también depende de la forma de obtener las muestras, es decir del patrón de muestreo empleado a la hora de calcular las muestras. Se puede emplear un patrón regular para cada píxel, obteniendo valores siempre en las mismas ubicaciones dentro del píxel o emplear un patrón aleatorio para cada píxel. Este último es más recomendable, pues este método incluye ruido en el filtrado que permite mitigar algunas de las razones que causan los efectos de *aliasing*.

La librería gráfica OpenGL ofrece una opción de reducir los efectos del *aliasing* en un único pase mediante una técnica conocida como *MultiSampling*.

Se han implementado tres métodos de *anti-aliasing* [BES05]. El primer método es muy similar al tradicional. Una vez que se ha identificado la intersección entre el rayo y la pirámide de proyección se accede a la textura para obtener el color del píxel. En este caso, en vez de acceder a un único valor de la textura también se obtienen los valores que rodean al valor encontrado y se realiza una media ponderada según la distancia. Este método es rápido puesto que no se debe realizar ningún cálculo extra, sin embargo, no tiene en cuenta las características propias de la proyección esférica. La resolución de los píxeles es diferente, provocando que la eficacia de la técnica no sea uniforme en toda la imagen.

En el segundo método, la técnica de *aliasing* se ejecuta un paso antes. Por cada píxel de la imagen final en vez de un único rayo se lanzan más rayos, formando un cono desde el punto de vista, dependiendo del número de muestras que se quieran tomar. Cada rayo tendrá un punto de intersección diferente y accederá a un valor de la textura diferente. El valor final empleado también es una media ponderada según la distancia. Este método se ajusta mejor a la geometría de la proyección esférica, pero los cálculos a realizar en cada píxel se multiplican por el número de muestras que se toman.

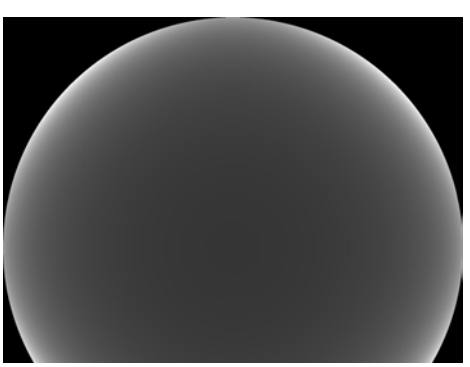

Figura 8: *Ángulo de visión soportado por cada píxel. El color negro representa el menor ángulo soportado y el blanco el mayor*

En la proyección de ojo de pez hemisférica, el campo de visión soportado por cada píxel no es constante. Los píxeles centrales suelen soportar un campo de visión menor que los píxeles exteriores. En la Figura 8 se puede observar como varía el ángulo soportado por cada píxel. Esto provoca que al tener un volumen mayor los píxeles exteriores, estos tengan una mayor probabilidad de que dentro de su volumen se encuentren detalles de alta frecuencia que generen efectos de *aliasing*. En la proyección de ojo de pez angular la resolución soportada por cada píxel es similar.

Por otro lado, tal como se ha descrito previamente, el volumen de visión de la proyección esférica se divide en pirámides de proyección para realizar la proyección perspectiva en ellas. Las dimensiones de estas pirámides de visión no tienen que ser iguales. Esto provoca que el volumen de visión soportado por cada píxel en las diferentes pirámides de proyección sea también diferente.

El último método de *aliasing* es una extensión del segundo método. Sin embargo tiene en cuenta el volumen de visión soportado por cada píxel de la imagen final. El número de rayos que se lanzan por cada píxel depende del volumen de visión soportado por este. En la etapa de preproceso se calcula el volumen soportado por cada píxel y esta información se almacena en una textura de números de coma flotante que se pasa como dato de entrada al programa que se ejecuta en la tarjeta gráfica. Dependiendo del valor almacenado por cada píxel, se lanza un número de rayos determinado. De este modo, se logra ajustar la calidad del *aliasing* a las necesidades de la proyección esférica. Los valores blancos son las zonas donde el píxel soporta un mayor ángulo de visión. Esta es la información que se transfiere al programa. En la Figura 9 se puede observar parte de una imagen antes y después de aplicar las técnicas de *anti-aliasing*.

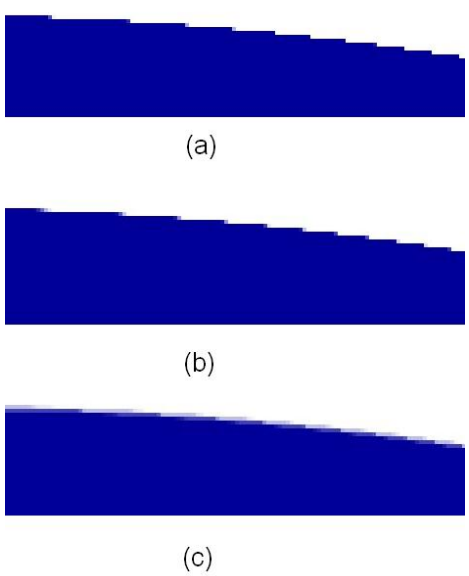

Figura 9: *(a) Sin anti-aling (b) Tras aplicar el método 1 (c) Tras aplicar el método 3*

4. Resultados

La tarjeta gráfica sobre la que se han realizado las mediciones es una nVidia geForce 8800 GTX.

Primero se ha querido cuantificar cual es la pérdida en frecuencia de refresco respecto a visualizar el mismo escenario en perspectiva y en un único pase. Para dibujar la misma cantidad de polígonos, la pirámide de visión de la proyección en perspectiva tiene un ángulo de apertura de 140^o. La imagen generada no es de una gran calidad pero sirve a modo de evaluación.

El escenario empleado para la prueba es un modelo con poca carga poligonal. La técnica descrita se ejecuta en la etapa de pixelado, de forma que se considera que la carga poligonal no debe tener una gran influencia a la hora de medir la efectividad del programa desarrollado. En el dibujado del escenario, si se han incluido diferentes técnicas que incorporan una gran carga en la etapa de rasterización: generación de sombras, simulación de fluidos, iluminación por píxel...

Primero se ha medido la diferencia de frecuencia de refresco entre los dos tipos de proyecciones dibujando la escena con todas las técnicas de dibujado comentadas en los apartados anteriores. La frecuencia de refresco media que se ha obtenido con la proyección en perspectiva ha sido de 144 fps, mientras con la proyección esférica se ha logrado una frecuencia de 70 fps. A continuación, se ha eliminado la técnica para calcular sombras y la simulación de fluidos y la frecuencia en perspectiva sube hasta 212 fps mientras que en proyección esférica es de 84 fps. Una última prueba consiste en sustituir el programa desarrollado por otro programa que no realizaba ningún tipo de cálculo. En este caso la frecuencia de refresco no disminuye.

Estos datos indican que el factor principal que afecta al método de visualización esférica desarrollado es el empleo de tres pases para lograr la proyección de todo el volumen de visión y no el programa encargado de realizar la conversión en proyección esférica.

El fabricante del dispositivo VisionStation ofrece a sus usuarios un API para poder generar las imágenes esféricas, *Omnimap* [Elu08]. Se ha incorporado este API al motor gráfico donde se han realizado las pruebas. Se ha comparado la respuesta del programa que se ha desarrollado con las prestaciones que ofrece este nuevo API. La frecuencia que se alcanza con este API es de 50 fps, de forma que el programa generado logra incrementar en un 40% las prestaciones respecto de las del API comercial.

También se ha querido evaluar el rendimiento de los sistemas de *anti-aliasing* diseñados. Empleando el primer método de *anti-aliasing* el rendimiento no disminuye. Al introducir nuevos cálculos en el programa, su rendimiento no disminuye. Sin embargo, cuando se emplean el segundo método el rendimiento decrece en un 35%. Al lanzar más rayos por cada píxel, los cálculos se multiplican por el número de muestras empleados por cada píxel. En el último método, el número de muestras por píxel depende de la ubicación del píxel en la imagen. Como computo general se emplea un número total de muestras menor, reduciendo la respuesta del programa en un 25%.

Figura 10: *Simulador de tunelación*

El sistema de visualización esférica desarrollado ha sido se ha incorporado a un simulador de sistemas de tunelación (Figura 10). Este simulador exige un campo de visión amplio para que el alumno pueda visualizar en todo instante el frente del túnel. El dispositivo VisionStation se adecua a las exigencias y el programa desarrollado cumple de forma correcta los requisitos del simulador.

5. Conclusiones

En este artículo se ha presentado un programa que permite generar imágenes empleando la proyección esférica. La proyección esférica se ha implementado haciendo uso de las capacidades programáticas que ofrecen las tarjetas gráficas hoy en día. Esto ha permitido emplear un dispositivo de visualización que se adecua muy bien a algunas aplicaciones de realidad virtual.

Se han implementado tanto las proyecciones ojo de pez hemisférica como la angular.

Se han introducido técnicas de optimización que permiten incrementar las capacidades del programa tanto a la hora de aumentar su velocidad de ejecución como a la hora de mejorar la calidad de la imagen final.

La proyección esférica diseñada se ha introducido en un simulador de entrenamiento para sistemas de tunelación.

Referencias

[BES05] BEISTER M., ERNST M., STAMMINGER M.: A hybrid gpu-cpu renderer. *Vision, Modeling, and Visualization* (2005), 415–420.

[Bou04] BOURKE P.: Computer generated angular fisheye projections. http://local.wasp.uwa.edu.au/~pbourke/projection/fisheye/.

[Elu08] ELUMENATI T.: Omnimap api: Real-time geometry correction library. http://www.elumenati.com/products/omnimap.html.

[JSY06] JI Y., SHEN W., YU J.: Large working area f-theta lens. In *Proceedings of the SPIE* (2006), vol. 6034, pp. 418–424.

[MB05] MCREYNOLDS T., BLYTHE D.: *Advanced Graphics Programming Using OpenGL*. Morgan Kaufman, 2005.

[Ngu07] NGUYEN H.: *GPU Gems 3*. Addison-Wesley Professional, 2007.

[OD07] OTT D., DAVIS T. A.: Simulating a virtual fisheye lens for the production of full-dome animations. In *Proc. 45th annual southeast regional conference* (Mar. 2007), pp. 294 – 299.

[PBMH02] PURCELL T. J., BUCK I., MARK W. R., HANRAHAN P.: Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics 21*, 3 (2002), 703–712. (Proceedings of ACM SIGGRAPH 2002).

[RK03] RANDIMA F., KILGARD M. J.: *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley Professional, 2003.

[RvBWR04] RASKAR R., VAN BAAR J., WILLWACHER T., RAO S.: Quadric transfer for immersive curved screen displays. *Computer Graphics Forum 23*, 3 (Sept. 2004), 451–460.