ISSN 0717-9103

APLICACIONES AVANZADAS DEL USO DE TEXTURAS PRECOMPUTADAS EN ENTORNOS DE SIMULACIÓN EN TIEMPO REAL

ADVANCED APPLICATIONS OF PRE-COMPUTED TEXTURES IN REAL-TIME SIMULATION ENVIRONMENTS

IÑAKI AYUCAR[1]
ALEJANDRO GARCÍA-ALONSO[2]
LUIS MATEY[3]

1 Lander Simulation & Training Solutions, S.A.,San Sebastián - España.
2 UPV – EHU, San Sebastián - España.
3 Universidad de Navarra, San Sebastián - España

RESUMEN

Tradicionalmente, las superficies de los objetos virtuales se han caracterizado por ciertos parámetros como su color, opacidad o reflectividad. Adicionalmente se han utilizado texturas como un medio para añadir matices a elementos de bajo detalle geométrico. Estos procedimientos son adecuados para entornos industriales, en los que la estética toma un papel secundario, pero no para otro tipo de escenarios, como la representación virtual de patrimonio histórico y del arte, que incluye la visualización de edificios u objetos cuyos materiales han de parecer realistas, teniendo en cuenta efectos de suciedad, envejecimiento o imperfecciones. El objetivo de este estudio es el análisis detallado y las posibles aplicaciones de una serie de técnicas de última generación que posibilitan la representación de objetos en tiempo real con dotes de realismo añadidos mediante el uso de multitexturas.

Los tres frentes desde los que se aborda el problema son:

- Representación de entornos virtuales. Iluminación.
- Objetos representados mediante materiales complejos.
- Variantes dinámicas de las texturas pre-computadas. Efectos especiales.

ABSTRACT

Virtual object´s surfaces have been traditionally set up by several parameters like colour, opacity or reflectivity. In addition to that, it has been common to use textures as a way of adding detail to low polygon count models. This kind of procedures fit into industrial environments, where visual appearance takes a second place, but not into other kind of scenarios, such virtual representations of cultural heritage and art. In those cases, we need to achieve a higher level of visual quality including the simulation of ancient materials, bumpiness or dirt, always in real-time.

The main objective of this paper is to analyse the applications of several multi-texturing techniques, which provide this kind of effects.

[1]Lander Simulation & Training Solutions, S.A..San Sebastián – España.
C/ Arroniz 2, 1c. Estella (Navarra). España.
e-mail: iayucar@yahoo.com
[2]UPV – EHU. San Sebastián – España.
e-mail:
[3]Departamento de Mecánica Aplicada, CEIT. Universidad de Navarra. San Sebastián – España.
e-mail: lmatey@ceit.es

We approach to this issue in three different ways:

- Lighting virtual environments.
- Complex materials.
- Dynamic uses of pre-computed textures.

PALABRAS CLAVE
Lightmaps: Texturas especiales que almacenan información referente a iluminación.
Multi-Texturas: Aplicación de más de una textura a una misma superficie con el objetivo de alcanzar un mayor realismo en la visualización.
Tiempo real: Procesamiento de información a una velocidad suficiente para proporcionar una respuesta inmediata al usuario, y conseguir así cierta interacción con dicha información.

1. INTRODUCCIÓN

Tradicionalmente, las superficies de los objetos virtuales se han caracterizado por ciertos parámetros como su color, opacidad o reflectividad. Adicionalmente se han utilizado texturas como un medio para añadir matices a elementos de bajo detalle geométrico. Estos procedimientos son adecuados para entornos industriales, en los que la estética toma un papel secundario, pero no para otro tipo de escenarios, como la representación virtual de patrimonio histórico y del arte, que incluye la visualización de edificios u objetos cuyos materiales han de parecer realistas, teniendo en cuenta efectos de suciedad, envejecimiento o imperfecciones. Además, la iluminación tradicionalmente utilizada en entornos tridimensionales no aporta el suficiente nivel de detalle para ofrecer una apariencia realista.

Los dispositivos de aceleración 3d actuales pueden utilizar varias imágenes de forma simultánea para un polígono. Además, las operaciones que se pueden aplicar a dichas texturas para producir el resultado final han aumentado tanto en calidad como en cantidad en los últimos años. Éste es el soporte tecnológico perfecto para simular efectos complejos mediante técnicas de texturizado, consiguiendo los citados materiales con imperfecciones, reflejos, suciedad o relieve, y efectos de iluminación avanzados como reflexión, refracción o radiosidad.

Este nivel de realismo en entornos de tiempo real supone un claro aliciente ya que hace tan sólo unos meses, este tipo de prestaciones sólo eran planteables en entornos no interactivos.

Por lo tanto, el objetivo de este estudio es, por un lado, aportar los métodos y procedimientos necesarios para el almacenamiento y la manipulación de iluminación avanzada en tiempo real mediante lightmaps, así como analizar otras técnicas adicionales existentes que incrementan el realismo de los materiales.

2. MATERIAL Y MÉTODOS

La técnica lightmapping

El lightmapping ha sido una técnica muy popular durante los últimos años ya que permite incluir, en una escena dibujada en tiempo real, iluminación, con un detalle muy superior a los tradicionales sistemas que efectúan los cálculos a nivel de vértice. El funcionamiento básico de esta técnica queda reflejado en la figura 1, en la que se aprecia una textura base a la que se multiplica el lightmap correspondiente a un polígono.

Figura 1: Lightmapping en detalle

Como puede observarse, se pre-calcula la iluminación para las superficies mediante cualquier tipo de algoritmo y se almacena en texturas.

La capacidad de los dispositivos 3d actuales para manejar de forma simultánea más de una textura sin penalización en el rendimiento, ha hecho de los lightmaps una técnica muy apropiada para este tipo de tareas. Los entornos ideales para su uso son aquellos con pocos polígonos, cuyos vértices están muy separados (escenas típicas de los entornos en tiempo real), ya que en estas condiciones la iluminación por vértice es insuficiente.

Trabajo relacionado

Actualmente, no existe mucha información acerca del lightmapping. Todo lo que puede encontrarse, son tutoriales introductorios que no profundizan lo suficiente como para implementar un sistema completo de generación de lightmaps. Éstos fueron introducidos en los gráficos en tiempo real por Carmack, en el desarrollo del videojuego Quake. Un pequeño texto que describe esta técnica puede encontrarse en [3].

Varias mejoras a esta metodología han ido apareciendo, incrementando la cantidad y resolución de los lightmaps. Dado que el Lightmapping es una técnica estática, otros estudios como el expuesto en [5], se han enfocado a dotar de propiedades dinámicas a los lightmaps.

Diseño e implementación

La orientación del trabajo que aquí se presenta no se centra en el algoritmo de iluminación utilizado en la escena, sino en el soporte apropiado para almacenar la información que éste genera y su representación posterior en el software 3d, ya que son las dos cuestiones que realmente permanecen indocumentadas.

Convivencia entre el Lightmapping y entornos optimizados

Aplicar lightmapping en entornos que utilizan transformación e iluminación por hardware presenta ciertos problemas:

Restricciones desde el punto de vista del dispositivo 3d

Hoy en día, los dispositivos de aceleración 3d requieren que los polígonos sean enviados al proceso de dibujado en grandes grupos. De este modo, se activan una serie de optimizaciones internas (como la utilización de memoria cache) que aceleran de forma notable el rendimiento de la aplicación. Dentro de este proceso de dibujado, no se permite cierto tipo de cambios, como los referentes a texturas, materiales, etc.

Debido a ello, es muy común reordenar la geometría de un objeto agrupando los polígonos que compartan materiales o texturas, tratando de maximizar el tamaño de estos conjuntos de datos.

Restricciones desde el punto de vista de los lightmaps

El modo en que la luz incide en un polígono es completamente diferente de la forma en que lo hace sobre el polígono vecino. Debido a esto, se generan lightmaps distintos para ambos. Dado que estos resultados son almacenados en texturas para utilizarlas después al dibujar, se hace necesario establecer una textura diferente para cada polígono.

Como se ha comentado antes, este tipo de cambios no pueden ser realizados sin echar por tierra las optimizaciones comentadas. Es decir, por una parte, necesitamos minimizar el número de cambios de textura para los objetos y por otra, establecer una textura diferente para cada polígono. Esto obviamente es una contradicción.

Solución

Este problema puede ser resuelto mediante el uso de cierto tipo de geometría muy sencilla, o eligiendo de forma muy detallada qué objetos van a utilizar lightmaps y cuáles no. Como es lógico, este tipo de proceder no supone una solución real al problema y restringe mucho la utilidad de la aplicación que se desarrolle, por lo que queda excluido de los objetivos de este estudio.

Por lo tanto, el enfoque correcto es encontrar una forma de aplicar diferentes bitmaps a los polígonos de un objeto, respetando por un lado la generalidad y el automatismo propuestos y por el otro, las optimizaciones que la tarjeta gráfica realizará después.

La solución que se propone consiste en unir en una única y gran textura todos los lightmaps de un grupo de polígonos que vayan a dibujarse de una sola vez. Como podrá verse más adelante, esta tarea no es sencilla en absoluto y requiere la resolución de multitud de problemas geométricos y de manipulación de texturas.

Siguiendo esta idea, se puede esquematizar el funcionamiento del sistema de la siguiente manera:

1. Preproceso: Crear los lightmaps individuales, incluirlos en el lightmap general (G.L) y establecer un método de recuperación que permita su extracción posterior (vía coordenadas de texturas)
2. Runtime: Para cada grupo de polígonos, cargar su G.L y dibujar.

Problemas geométricos y de texturas derivados del almacenamiento de los lightmaps

El almacenamiento de un conjunto de lightmaps dentro de un solo bitmap y su utilización en entornos de visualización 3d conlleva una serie de restricciones que han de solventarse.

El problema de los "vértices compartidos"

Normalmente, la geometría se construye en forma de vértices indexados. Más concretamente, se dispone de una lista de vértices y otra distinta de triángulos. En esta última se incluye para cada triángulo los índices de los vértices que lo componen. De esta forma, los vértices que son compartidos por más de un polígono tienen una única instancia en memoria.

Dado que los lightmaps son añadidos al G.L en orden de aparición y no de vecindad, un vértice compartido en dos facetas diferentes puede tener coordenadas de texturas diferentes en las dos (ver figura 2).

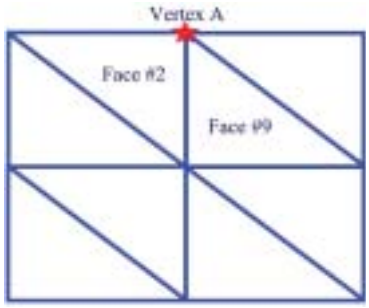

Figura 2: Vértice compartido por varias facetas

Las facetas son vecinas geométricamente, pero no dentro del G.L ya que el orden de aparición de las mismas es arbitrario (ver figura 3). Esta cuestión presenta un problema ya que se desea poder procesar la geometría de forma automática. Aunque las librerías más comunes permiten establecer diferentes "sets" de coordenadas de texturas para un vértice, el número de facetas que puede compartir un vértice es absolutamente variable, por lo que esta solución no es recomendable. La mejor solución es procesar la geometría en tiempo de carga clonando los vértices compartidos para que sólo tengan un "set" de coordenadas de texturas. Obviamente, siguiendo esta metodología se introduce cierta penalización en el rendimiento, pero se consigue alcanzar la generalidad propuesta.

Figura 3: Vértices compartidos en el G.L

Restricciones de precisión

Un efecto curioso fue detectado en el desarrollo del proyecto. Se descubrió que las coordenadas de texturas aplicadas a los vértices sufren redondeos automáticos efectuados por la librería gráfica. Estos redondeos producen "efectos" no deseados en nuestro caso, ya que un polígono accederá parcialmente a un lightmap que no le pertenece. La forma de establecer coordenadas que permanezcan intactas al dibujar es seguir la siguiente regla:

"En el momento de decidir el número de filas y columnas que compondrán el G.L, la cifra elegida debe ser potencia de dos."

De esta forma, si han de introducirse n lightmaps dentro del G.L, se escogerá el siguiente entero superior a la raíz cuadrada de n que respete esta regla.

El problema del filtrado de texturas

El color final que se dibuja para un píxel se toma del resultado de diversas operaciones como la rasterización y el filtrado de texturas. El filtrado de texturas se lleva a cabo para incrementar la calidad visual de una escena, disminuyendo el efecto de pixelado que se produce al acercarse a un objeto (ver figura 4).

Esta operación implica realizar una media aritmética del color del texel correspondiente al píxel a dibujar y sus vecinos (ver figura 5).

Figura 4: Efecto del filtrado de texturas

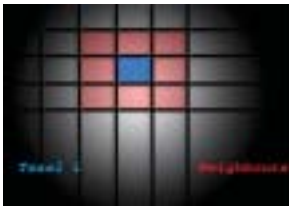

Figura 5: Texels para el filtrado de texturas

De vuelta en el G.L, cuando se dibujen pixels correspondientes a los bordes de los lightmaps, se aplicará esta misma regla, escogiendo texels para la media aritmética de un lightmap diferente. La

iluminación en este segundo lightmap puede ser drásticamente diferente y afectará erróneamente al color del texel actual (ver figura 6).

Figura 6: Texel erróneamente promediado

Este efecto es inversamente proporcional a la resolución de los lightmaps, de modo que existen dos formas de solucionarlo:

1. Incrementar la resolución de los lightmaps hasta encontrar unos resultados aceptables.
2. Desactivar el filtrado de texturas así como el MipMapping para la fase del proceso de dibujado que se encargue de los lighmaps. Dado que el filtrado permanece activo para la textura base, no se producirán efectos negativos notables en la calidad final de la imagen.

Generación automática del G.L.

Dado un conjunto de geometría y sus lightmaps, ha de encontrarse una forma de ordenarlos dentro del G.L. El concepto se expone gráficamente en la figura 7.

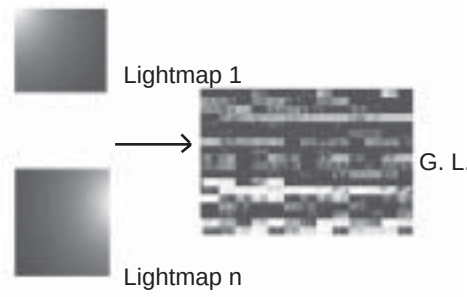

Figura 7: Composición del G.L

El orden establecido debería ser fácilmente recuperable en tiempo de ejecución. Un sistema perfecto sería añadir los lightmaps en el orden en el que las facetas aparecen en el objeto. La correspondencia entre los lightmaps y los polígonos se expone en la figura 8.

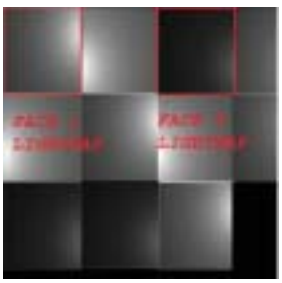

Figura 8: Composición del G.L. (II)

Recuperación automática desde el G.L.

El modo en que se recuperan los lightmaps individuales se basa en las coordenadas de texturas, asignando a cada vértice un segundo set de coordenadas que acceden exactamente a la parte del G.L que les corresponde. Este segundo set de coordenadas se genera de forma automática basándose en la siguiente información:

1. Número de filas y columnas dentro del G.L, siguiendo el mismo algoritmo expuesto en la sección 3.3.3.
2. Tamaño de los lightmaps individuales, fácilmente calculable en base a las dimensiones del G.L. y el numero de filas y columnas.

De esta forma, se puede recorrer el G.L asignando coordenadas de forma incremental, según el número de lightmap en el que nos encontremos y estos dos datos calculados previamente.

Generación de los lightmaps individuales

Esta parte del problema es la más documentada y a su vez la más sencilla de implementar. Por esto, nos centraremos en las partes más desconocidas del algoritmo.

Para generar el lightmap correspondiente a un polígono, ha de recorrerse mediante interpolación la parte interior de éste, calculando un punto 3d para cada paso y trazando desde él un rayo hasta la luz. Una vez que dicho rayo está construido, cualquier algoritmo de iluminación puede ser aplicado: Ecuación de Lambert, proyección de sombras (vía detección de colisiones con la geometría), etc. También se puede aplicar a algoritmos de radiosidad, aunque con distinta implementación.

Generando el rayo de luz

Dado un triángulo y una luz, han de recorrerse todos los puntos interiores del triángulo creando un rayo desde ellos hasta la luz.

Dado que el objetivo final es almacenar los cálculos en un bitmap, la mejor forma de llevar a cabo esta tarea es justo al revés de como se ha expuesto hasta ahora:

1. Ciclar para cada punto dentro del bitmap a construir.
2. Para ese punto, calcular las coordenadas del punto (interior al triángulo) al que correspon-derá, una vez aplicada esa textura al polígono.

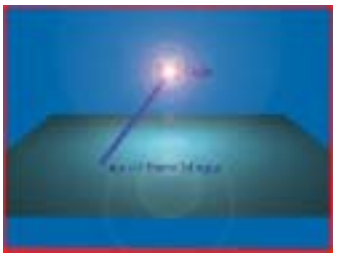

Figura 9: Generación de un lightmap individual

3. Generar un rayo desde la luz hasta ese punto (ver figura 9).
4. Aplicar los cálculos de iluminación escogidos.
5. Salvar el resultado en el punto del bitmap del que se ha partido en el punto 1.

Una vez que se haya ciclado para todos los puntos del bitmap, sólo restará salvar el lightmap completo para su posterior adición al G.L.

Otras técnicas adicionales para la simulación de materiales complejos

Las texturas precomputadas pueden ser utilizadas para otro tipo de tareas, aparte de almacenar información concerniente a la iluminación de una escena.

Como veremos en este apartado, mediante técnicas de texturización se pueden ahora simular materiales complejos que anteriormente no se podían emplear en sistemas en tiempo real.

Los distintos ejemplos que se van a exponer aquí utilizarán siempre una misma base: la aplicación de varias capas de texturas a un objeto.

En las librerías gráficas más comunes, estas capas se reparten entre una serie de fases en el interior del "pipeline" gráfico (cadena de procesamiento gráfica). Los datos geométricos entran por un extremo del proceso y van pasando por todas estas fases o "stages" activas que encuentren. Cada una utilizará los resultados de la anterior y su propia textura para aplicar una determinada operación y producir un resultado que se propagará hacia adelante o hacia el exterior del "pipeline" (ver figura 10).

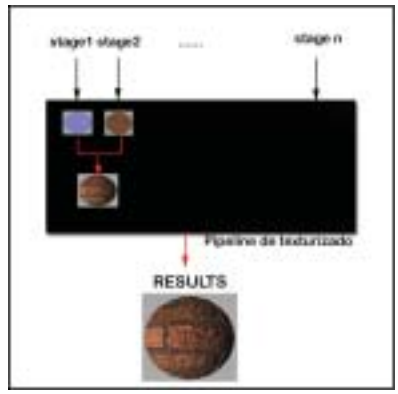

Figura 10: Detalle del pipeline de texturizado

Simulación de imperfecciones y suciedad

Tradicionalmente se utiliza una textura base repetitiva para dotar a los objetos de un aspecto determinado. Los resultados de esta técnica son demasiado homogéneos, ya que en la realidad, una superficie está salpicada de imperfecciones, manchas, matices, etc.

Para conseguir este efecto más natural se utilizan texturas adicionales que van aportando los matices mencionados. Como puede verse en el ejemplo siguiente (figura 11) se ha utilizado una textura base que proporciona la apariencia de una pared de piedra. Adicionalmente, se ha incluido un efecto de hielo encima de la primera imagen:

Figura 11: Aplicación de un efecto de hielo

La operación utilizada para mezclar ambas texturas es una simple sustitución, guiada por un canal alpha incluido en la segunda textura, que indica qué partes de la misma son transparentes (parte gris de la imagen).

En un segundo ejemplo (ver figura 12) se efectúa una operación de envejecimiento de un material. Partiendo de una textura base, se van añadiendo diferentes capas para añadir detalle en las zonas deseadas. Una primera capa se encarga de dotar de manchas irregulares toda la superficie. La última refleja un efecto de óxido en la parte superior de la misma.

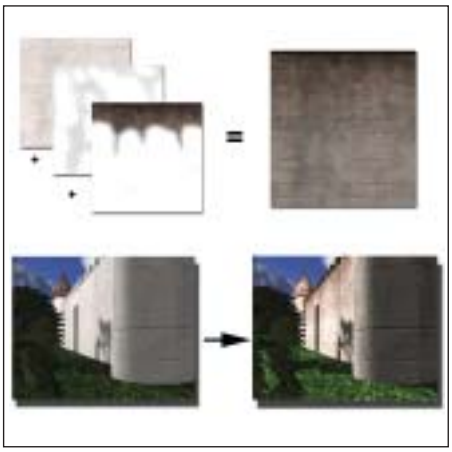

Figura 12: Materiales envejecidos

Simulación de relieve

Determinado tipo de materiales, cuya superficie es muy rugosa, reflejan la luz de manera diferente, produciendo multitud de pequeñas sombras en los huecos de su superficie. La técnica que simula este efecto fue introducida por Blinn en 1978 y se denomina "BumpMapping". Una variante de la misma que se adapta de forma perfecta a las necesidades de rendimiento actuales es la denominada "Dot Product Bump Mapping" o "Dot3 Bump Mapping", comentada en detalle en [9]. Esta técnica también utiliza texturas precomputadas y dado que es una operación implementada por el chip gráfico de la mayoría de dispositivos recientes, puede ser utilizada en tiempo real.

En este caso, se dispondrá de dos tipos de textura diferentes. Primero se utilizan tantas texturas como sea necesario para producir el resultado adecuado (ver ejemplos anteriores). Después, en la última etapa del proceso se incluye una textura especial denominada "normal-map". Esta textura se genera mediante software específico (normalmente en forma de "plug-ins" para paquetes de retoque fotográfico) y contiene la normal para cada píxel en forma de vector x,y,z.

Como operación para componer el resultado final se establece el mencionado "Dot Product Bump Mapping", que consiste en interpolar a lo largo del polígono un vector construido hasta la fuente de luz, para después realizar el producto escalar con las normales almacenadas en el "normal-map". Los resultados pueden observarse en la figura 13:

Figura 13: Resultado de Dot3 BumpMap

Reflexiones simuladas

Otro tipo de materiales, reflejan su entorno debido a determinadas propiedades de su superficie. Una aproximación a este efecto muy utilizada en tiempo real es la utilización de mapas de entorno o "enviroment-maps". Existen diferentes variantes de esta técnica. La que aquí se expone es la técnica denominada "CubeMapping", basada en proyecciones ortogonales en todas las direcciones que rodean al objeto. Además, los "environment-maps" pueden ser precomputados o no. En este apartado se estudia el primer caso, mientras que en la sección 4.4 se citará su variante dinámica.

Aunque precalcular estos mapas no es una representación fiel a la realidad, es muy eficiente y para determinados casos proporciona resultados satisfactorios. Al inicio del programa se proyecta el entorno que rodea al objeto a dibujar en 6 texturas correspondientes a cada una de las direcciones.

Estas 6 texturas compondrán el llamado “cube-map” relacionado con el objeto. En la figura 14 puede verse el diagrama de creación de esta textura incluido en [2]. Una vez en la etapa de dibujado, han de generarse coordenadas de texturas de forma dinámica para acceder a la parte apropiada del “cube-map”. Esta operación también es soportada por el hardware gráfico actual, consiguiendo el rendimiento apropiado para la visualización en tiempo real.

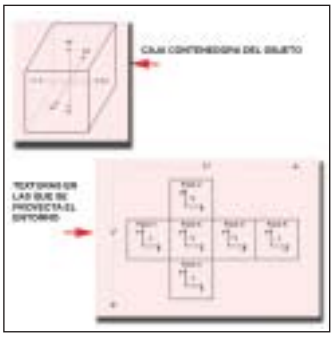

Figura 14: Construcción del “cube-map”

Variante dinámica de las técnicas de texturizado precomputadas

La principal diferencia de este tipo de procedimientos con los expuestos en el capítulo anterior es que los mapas precomputados van a sufrir algún tipo de modificación en tiempo real, bien sea para producir resultados más realistas o para reflejar algún tipo de animación.

Billboards animados

Se denomina “BillBoarding” a una técnica de dibujado mediante la cual, se muestran objetos que geométricamente deberían ser muy complejos mediante un simple plano y una textura. Dicho plano se ajusta en tiempo real para orientarlo hacia el observador, de modo que no note la falta de tridimensionalidad. Su variante animada puede ser utilizada para simular efectos imposibles de representar en tres dimensiones con los recursos actuales, como explosiones o humo. La animación se simula pre-generando una serie de texturas que, presentadas una tras otra en forma de ciclo, producen el resultado deseado (ver figura 15).

Figura 15: Texturas utilizadas en BillBoarding

CubeMapping Dinámico

La variante dinámica de esta técnica sólo varía en el cálculo del “cube-map”, que anteriormente se generaba en tiempo de carga de la escena. En determinadas situaciones será conveniente recalcular este mapa en cada frame, por ejemplo, si el objeto reflectante está en movimiento o si es su entorno el que se mueve.

3. RESULTADOS

Se incluyen en este apartado algunos de los resultados obtenidos al aplicar la técnica Lightmapping para entornos virtuales de apariencia fotorrealista, así como del resto de técnicas comentadas en el estudio.

En la figura 16 se observa un sistema de proyección de sombras con varias luces. Por otro lado, la figura 17 muestra una proyección de sombras más compleja, utilizando árboles simulados mediante texturas.

Figura 16: Escena iluminada mediante lightmaps

Figura 17: Proyección avanzada de sombras

Una comparación entre una escena dibujada en tiempo real y la misma escena representada por un paquete de diseño 3d comercial puede encontrarse en la figura 18. El lightmapping en este caso utiliza como base iluminación difusa sencilla (mediante el algoritmo de Lambert) y proyección de sombras.

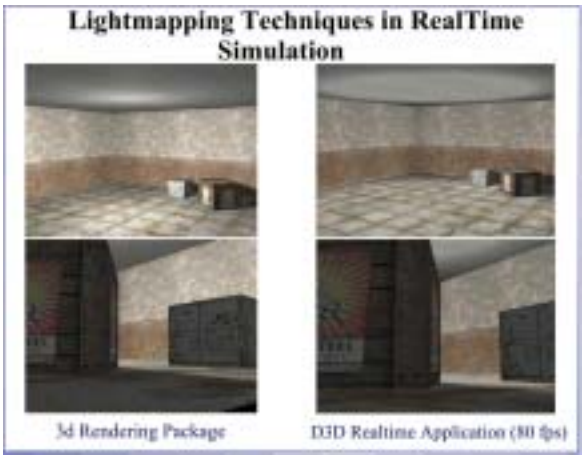

Figura 18: Comparación Modelado–RealTime

Figura 19: Entorno exterior nocturno

En la figura 20 se muestran los resultados de aplicar reflexiones mediante “cube-maps”, pre-calculados en tiempo de carga de la escena para el agua y el vehículo.

Figura 20: Cube Mapping estático

4. CONCLUSIONES.

Una gran parte de los efectos visuales que han impulsado el incremento en el realismo de las escenas virtuales se basan en técnicas de texturizado. Como se ha visto, es muy útil que este tipo de procedimientos sean optimizados mediante el cómputo anticipado de determinada información y su almacenamiento en texturas especiales. A su vez, los dispositivos gráficos van adaptándose para dar un mejor soporte a la gestión de este tipo de información, haciéndola más transparente para el usuario y más eficiente.

Recopilando toda esta información, hemos desarrollado un sistema que se apoya en texturas precomputadas para almacenar la iluminación concerniente a una escena, de modo que la complejidad de ésta no afecte a su representación en tiempo real.

De este modo, añadiendo nuestra metodología a la tendencia establecida de utilizar entornos virtuales tradicionales, cada vez más detallados y complejos y a la simulación de materiales realistas, se consiguen unas cotas de realismo a nivel global de gran calidad en tiempo real.

5. REFERENCES

[1] Tomas Akenine-Möller, Eric Haines. RealTime Rendering (Second Edition), *A K Peters* (2002).
[2] DirectX9 SDK Documentation, Microsoft Corp. (http://msdn.microsoft.com)
[3] M. Abrash. Michael Abrash´s Graphics Programming Black Book. Special Edition. *Coriolis Group Books* (1997), 1245 – 1271.
[4] J. Arvo. Article by N. M. Thalmann, D. Thalmann & H. T. Minh. Graphics Gems II. *Academic Press, Inc.* (1991), 232 – 241.
[5] M. A. DeLoura. Article by Sim Dietrich. Game Programming Gems. *Charles River Media, Inc.* (2000), 543 – 548.
[6] D. H. Eberly. 3d Game Engine Design. *Morgan Kaufmann Publishers* (1999), 427 – 434.
[7] J. Foley, A. van Dam, S. Feiner & J. Hughes. Computer Graphics, Principles and Practice. *Addison Wesley Publishing Company.* (1990), 721 – 813.
[8] P. S. Heckbert. Article by E. Heines. Graphics Gems IV. *Academic Press, Inc*. (1994), 24 – 27.
[9] T. Möller & E. Hines. Real-Time Rendering. *A K Peters, ltd.* (1999), 124 – 125.