Informe Técnico
DLSI 01/01/2004

Evaluación del Uso de Técnicas de Aceleración con Modelos Multirresolución

José Ribelles[i], Ángeles López[ii]

[i]Departamento de Lenguajes y Sistemas Informáticos
[ii]Departamento de Ingeniería y Ciencia de los Computadores

Correo electrónico: ribelles@lsi.uji.es

Departamento de Lenguajes y Sistemas Informáticos
Universitat Jaume I
Campus de Riu Sec
E-12071 Castellón, España

Evaluating the Use of Aceleration Techniques in Multiresolution Models

José Ribelles[i], Ángeles López[ii],

[i]Departamento de Lenguajes y Sistemas Informáticos
[ii]Departamento de Ingeniería y Ciencia de los Computadores
Universitat Jaume I
Campus de Riu Sec
E-12071 Castellón, España

E-mail:ribelles@lsi.uji.es

Abstract:

In the last years, the consumer graphics hardware has evolved such that nowadays any user can provide his computer a graphics processor capable of processing 338 millions of vertices per second. This fact facilitates that geometrically complex objects (thousands and thousands of triangles) are more and more used in applications which require real-time visualization. The aim of this report is (1) to combine efficient rendering methods, in order to reach the processing ratio stated by the manufacturer, with objects modeled using multiresolution techniques, which are specially useful in real-time visualization of complex objects, and (2) to evaluate the behaviour from the resulting number of frames per second. More concretely, the experiments have been performed using multiresolution objects defined with a general multiresolution scheme such as Multiresolution Ordered Meshes and using the efficient rendering methods currently available in the OpenGL graphic standard.

Keywords:

Multiresolution modeling, level of detail, interactive visualization, polygonal simplification.

Evaluación del Uso de Técnicas de Aceleración con Modelos Multirresolución

José Ribelles[i], Ángeles López[ii],

[i]Departamento de Lenguajes y Sistemas Informáticos
[ii]Departamento de Ingeniería y Ciencia de los Computadores
Universitat Jaume I
Campus de Riu Sec
E-12071 Castellón, España

E-mail:ribelles@lsi.uji.es

Resumen:

En estos últimos años, el hardware gráfico de consumo ha evolucionado de tal forma que hoy en día cualquier usuario puede disponer en su ordenador de un procesador gráfico capaz de procesar 338 millones de vértices por segundo. Esto anima a que modelos geométricamente complejos sean cada vez más utilizados en aplicaciones que requieren una visualización en tiempo real. El objetivo de este informe es utilizar métodos eficientes de visualización con objetos multirresolución y evaluar el comportamiento a partir del número de fotogramas por segundo obtenido. Este informe pretende medir las prestaciones de visualización que pueden llegar a obtenerse al utilizar objetos multirresolución definidos con un modelo multirresolución general como *Multiresolution Ordered Meshes* si se utilizan los métodos eficientes de visualización que actualmente se encuentran disponibles en el estandar gráfico *OpenGL*. El objetivo es utilizar dichos métodos en el proceso de visualización de un objeto multirresolución y evaluar el comportamiento a partir del número de fotogramas por segundo obtenido.

Palabras clave:

Modelado multirresolución, nivel de detalle, visualización interactiva, simplificación poligonal.

Evaluación del Uso de Técnicas de Aceleración con Modelos Multirresolución

J. Ribelles† A. López‡

†Departamento de Lenguajes y Sistemas Informáticos
‡Departamento de Ingeniería y Ciencia de los Computadores

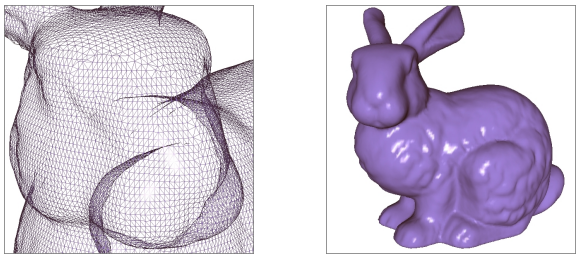
Universitat Jaume I, 12071 Castellón, Spain
e – mail: {ribelles,lopeza}@uji.es
Phone: +34-96-472-83-18, Fax: +34-96-472-84-35

1. INTRODUCCIÓN

En estos últimos años, el hardware gráfico de consumo ha alcanzado, y en muchas áreas sobrepasado, las capacidades del hardware gráfico extremadamente caro de justo ayer. Hoy en día, por ejemplo, por un precio de unos 40 euros es posible disponer de una tarjeta gráfica para PC con un procesador gráfico (GPU) capaz de procesar 20 millones de triángulos iluminados y texturados por segundo. Pero por un poco menos de 500 euros, la tarjeta gráfica dispone de una GPU capaz de procesar 338 millones de vértices por segundo. Además, los fabricantes hacen que su hardware esté bien soportado, ofreciendo buenos *drivers* para distintos sistemas (Windows y Linux) y distintas plataformas (PC y Mac), y cumpliendo con las más modernas especificaciones de los estándares gráficos OpenGL y DirectX.

Por otra parte, las prestaciones del sistema gráfico también se han visto incrementadas sustancialmente al conseguir mover información gráfica (geometría o texturas, por ejemplo) de la memoria del sistema central al sistema gráfico mediante el uso de buses dedicados de alta velocidad, en concreto, el bus AGP. De esta forma, la tecnología AGP posibilita que la GPU pueda utilizar la memoria central del ordenador para almacenar la información que vaya a utilizar sin tener que depender de la limitada, aunque cada vez menos, memoria local ubicada en la tarjeta gráfica.

Todo esto ha animado a que modelos geométricamente complejos (ver figura 1) sean cada vez más utilizados en aplicaciones que requieren una visualización en tiempo real. Sin embargo, para que una aplicación consiga que el sistema gráfico alcance las tasas de procesamiento indicadas por el fabricante es necesario hacer el mayor uso posible de ciertos recursos. En concreto, dos son las características a tener en cuenta. Estas son:



(a) Alambre (b) Sombreado

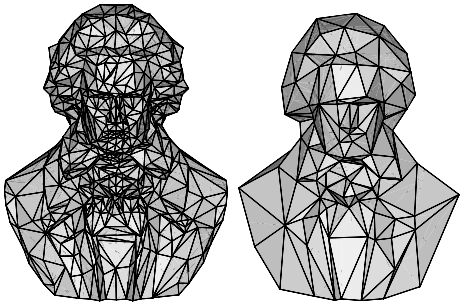
Figura 1: *Modelo Bunny formado por 34.835 vértices y 69.451 triángulos.*

Memoria AGP. La memoria AGP es rápida cuando la CPU escribe en ella y también cuando la GPU la lee. Tanto OpenGL como DirectX permiten al programador de aplicaciones hacer uso de la memoria AGP mediante *buffers* o vectores de datos y listas de visualización.

Memorias caché. Por un lado está la memoria caché típica de una CPU en la que el aumento de la localidad de los datos produce una mayor tasa de acierto de la caché y, por tanto, un incremento en las prestaciones del sistema gráfico. Por otro lado está la caché de vértices. Cada vez que un vértice entra en la GPU sufre un largo proceso en dicha memoria caché de manera que si el vértice es requerido de nuevo, y aún se encuentra en la caché, entonces no es necesario volver a procesarlo.

Para aprovechar al máximo estos recursos es necesario preprocesar los datos que van a ser visualizados (geometría, co-

lor, texturas y normales) de manera que puedan ser enviados al sistema gráfico tal y como el hardware requiere. Esta tarea no resulta difícil cuando los objetos que forman la escena son modelos estáticos, es decir, modelos que no sufren cambio alguno ni en su forma ni en su aspecto mientras permanecen en la escena. Sin embargo, cuando una aplicación requiere de elementos cuya geometría o aspecto cambien con el tiempo, es decir, modelos dinámicos, habrán de procesarse en tiempo de ejecución antes de ser enviados al sistema gráfico si se desea obtener el máximo rendimiento del hardware gráfico.



(a) 25 % (b) 5 %

Figura 2: *Dos aproximaciones distintas del modelo* Ludwig. *El porcentaje representa el número de caras utilizadas en comparación con el original.*

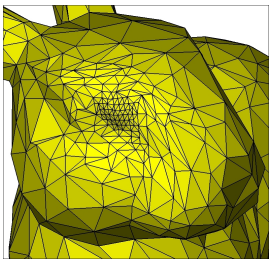


Figura 3: *Aproximación con nivel de detalle variable del modelo Bunny.*

Los objetos multirresolución[17] son modelos dinámicos. Así, la aplicación solicita una aproximación y esta se crea en tiempo de ejecución. Algunos esquemas multirresolución tratan de proporcionar aproximaciones que puedan ser visualizadas eficientemente, por lo que el coste en el que se incurre por utilizar representaciones multirresolución es sólo el coste de generar la aproximación. Sin embargo, hay esquemas que proporcionan aproximaciones que deben, además, ser procesadas para visualizarse de forma eficiente, incurriendo en un coste mucho mayor. En cualquiera de los dos casos, las aproximaciones de nivel de detalle constante (ver figura 2) normalmente son utilizadas durante una secuencia de fotogramas por lo que los costes que conlleve su uso pueden, probablemente, ser amortizados. Sin embargo, las aproximaciones con nivel de detalle variable (ver figura 3) normalmente son válidas para un sólo fotograma por lo que resultan de difícil uso en aplicaciones de visualización en tiempo real.

El objetivo de este informe es utilizar métodos eficientes de visualización con objetos multirresolución y evaluar el comportamiento a partir del número de fotogramas por segundo obtenido. Para llevarlo a cabo, se van a utilizar objetos multirresolución definidos con el modelo multirresolución general *Multiresolution Ordered Meshes*[15] y se utilizarán los métodos eficientes de visualización que actualmente se encuentran disponibles en el estandar gráfico *OpenGL* (explicados en la sección 3). El estudio incluye la realización de experimentos en distintas plataformas, es decir, diferentes sistemas operativos y diferente hardware gráfico, con el fin de observar si existen diferencias importantes entre plataformas distintas.

Este informe se estructura de la siguiente manera. En la sección 2 se presenta el trabajo relacionado. En la sección 3 se describen las técnicas de aceleración que proporciona OpenGL y que se han utilizado en la realización de este informe. En la sección 4 se describe el experimento realizado y los resultados obtenidos. Finalmente, se presentan las conclusiones en la sección 5.

2. TRABAJO RELACIONADO

Muy ligado al modelado multirresolución está el problema de la simplificación del objeto, es decir, cómo generar la secuencia de aproximaciones que a la postre servirán para crear la representación multirresolución. La necesidad de utilizar métodos de simplificación también es común a otras áreas por lo que existen una gran diversidad de métodos. Varios autores han realizado trabajos muy completos con el fin de clasificar y analizar los distintos métodos de simplificación [1, 9]. También se han presentado trabajos que muestran comparaciones de las prestaciones entre diferentes métodos [4, 13], y se han creado herramientas que miden la calidad de las mallas obtenidas como resultado de la simplificación [5].

Una vez realizado el proceso de simplificación se construye la representación multirresolución. Los esquemas multirresolución para superficies arbitrarias se dividen en dos grupos: *representaciones incrementales*, que codifican el conjunto de aproximaciones utilizando la de menor detalle y una secuencia lineal de actualizaciones que permiten refinar la superficie; y *representaciones jerárquicas*, que se basan en estructuras de tipo árbol donde las hojas representan la aproximación de mayor detalle, los nodos las simplificaciones y las raíces la aproximación de menor detalle. Algunas revisiones del estado del arte de ambos tipos de esquemas se pueden encontrar en [17, 8, 14].

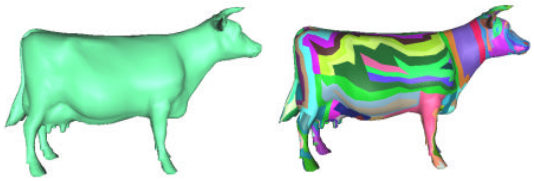


Figura 4: *A la izquierda el modelo Cow dibujado como un conjunto de triángulos independientes, a la derecha el mismo modelo dibujado como un conjunto de tiras de triángulos.*

Los esquemas multirresolución presentados en la literatura se han precupado en contadas ocasiones de generar aproximaciones que puedan ser visualizadas de forma eficiente. Cuando así se ha hecho, la investigación se ha dirigido a que los modelos multirresolución soporten primitivas gráficas eficientes como son la tira y el abanico de triángulos. Hoppe [12] propone que, una vez generada la aproximación que se ha de visualizar, se obtengan tiras de triángulos en ese preciso instante. El-Sana et al.[6] presentan *Skip-Strips*, que mantiene las tiras de triángulos aunque cambie el nivel de detalle del objeto. El esquema presentado por Ribelles et al.[16] utiliza el abanico como primitiva de representación en el propio esquema multirresolución. Belmonte et al. [2] presentan un esquema en el que la primitiva básica de representación es la tira de triángulos multirresolución (ver figura 4). En general, todos estos métodos incurren en un elevado coste para obtener la aproximación. Además, los abanicos o tiras de triángulos obtenidas suelen ser cortas. En consecuencia, las prestaciones de la aplicación apenas se ven mejoradas.

Hasta la fecha, sólo Bogomjakov y Gotsman[3] han presentado un método que permite optimizar el uso de la caché de vértices. Sin embargo, los autores sólo consiguen aplicarlo al modelo multirresolución *Progressive Meshes*[11] y además de manera poco eficiente.

3. TÉCNICAS DE ACELERACIÓN EN OPENGL

En esta sección se resumen las diversas técnicas de aceleración presentes hasta la fecha en la librería *OpenGL 1.5* que se van a utilizar para la evaluación objeto de este informe.

3.1. Vertex Arrays

Esta técnica permite especificar los datos (vértices, normales, color, textura, etc.) que forman el objeto mediante vectores con el fin de reducir el número de llamadas a funciones (*glBegin, glEnd, glVertex, glNormal*, etc.). Existe la posibilidad de utilizar para cada grupo de datos un vector diferente, es decir, un vector para vértices, otro para normales, etc., o utilizar un único vector con toda la información dispuesta de forma alterna.

Una vez almacenada la información en los correspondientes vectores, existen diversos mecanismos que permiten visualizarla.

Elemento a elemento

Mediante la orden *glArrayElement* se especifica qué elemento se envía al sistema gráfico. Es necesario hacer una llamada a la función por vértice independientemente de si la geometría se acompaña de color, normal u otros atributos.

Vector de elementos

Se utiliza la orden *glDrawElements* o *glDrawRangeElements*. En primer lugar se ha de crear un vector de índices de los vértices a pintar. Con las órdenes anteriores se especifica el primero del vector de índices y cuántos se envían al sistema gráfico.

Vector de datos

Esta opción consiste en disponer en un nuevo vector (uno para geometría, otro para normales, etc.) no de los índices sino de los datos del propio vértice. La orden para dibujar es *glDrawArrays* y se especifica el primero y el número de elementos a pintar.

3.2. Display Lists

Una *display list* no es más que una lista de funciones previamente almacenada para su ejecución posterior. Su principal ventaja es que el objeto a visualizar se define una sola vez y puede ser codificado y almacenado de forma eficiente para su visualización.

3.3. Vertex Buffer Objects

Los *Vertex Buffer Objects (VBO)* son una extensión que permite almacenar la información en *buffers* residentes en memoria de altas prestaciones. Esta técnica realmente permite decidir entre utilizar la memoria de vídeo, la memoria AGP o la memoria del sistema. Los mecanismos para envíar los datos al sistema gráfico son los mismos que en los *Vertex Arrays*.

4. RESULTADOS

4.1. Bancos de datos

Los bancos de datos utilizados son mallas generales (ver tabla 1). Sin embargo, se han realizado diversas consideraciones a la hora de definir las estructuras de datos debido al especial interés en acelerar el proceso de visualización. Estas son:

- Coordenadas de vértices y normales. Se han definido como enteros cortos. Esto supone dos bytes por coordenada de vértice o normal en lugar de cuatro que es lo necesario para un real de simple precisión. El ahorro de memoria es muy considerable.

- Componentes del color. Se han definido como carácter sin signo. Esto supone 3 bytes por color en lugar de 12 que ocuparía en caso de utilizar reales de simple precisión.

	Original		MOM	
	Vértices	Triángulos	LODs	Triángulos
Cow	2.905	5.804	2.803	14.982
Bunny	34.835	69.451	33.990	182.192

Tabla 1: *Características de los objetos.*

4.2. Test de visualización

El test que se ha implementado simula el acercamiento de un objeto a la cámara. Este acercamiento dura un total de cuatro segundos. En el instante inicial, el objeto se encuentra alejado de la cámara y se utiliza la representación de menos nivel de detalle para el objeto. En el instante final, el objeto se encuentra muy cerca del observador y se utiliza el nivel de mayor detalle. El objeto se acerca a la cámara con velocidad constante por lo que el tamaño de su proyección aumenta en cada fotograma. Sin embargo, el nivel de detalle seleccionado para representar el objeto se actualiza sólo cinco veces por segundo, es decir, se requieren un total de veinte aproximaciones diferentes en la simulación.

4.3. Configuraciones

Los experimentos se han realizado en ordenadores con diferente configuración. Como el informe trata de mostrar las diferencias entre las técnicas de aceleración, sólo se citan la GPU, memoria y sistema operativo. Estas son:

1. GPU NVIDIA Quadro 2 Pro, 64MB, LINUX
2. GPU NVIDIA GeForce FX 5900, 128MB, LINUX
3. GPU NVIDIA GeForce FX 5900, 128MB, WINDOWS

4.4. Experimentación

Las tablas 2, 3 y 4 recogen los resultados obtenidos. En cada tabla se muestra la evolución del número de fotogramas por segundo (fps) durante los cuatro segundos de visualización, para cada objeto y para las distintas técnicas de aceleración. La última columna indica el valor medio de fps.

De los resultados se pueden extraer diversas conclusiones:

- El uso de algunas técnicas de aceleración es totalmente compatible con el uso de esquemas multirresolución, al menos con el esquema utilizado en la experimentación, llegando a aumentar hasta en cinco veces el rendimiento a pesar de utilizar datos dinámicos.
- Se puede observar como las prestaciones obtenidas no siguen una norma fija, sino que en cada configuración existen diversas particularidades. Dicho de otro modo, lo mejor para una configuración dada no tiene por qué ser lo mejor para todas.
- En general, las técnicas de aceleración producen mejores resultados en el objeto *Bunny* y también es en este donde se pueden apreciar diferencias entre las técnicas de aceleración.
- El uso de los VBO, última extensión añadida a OpenGL, es la que mejor prestaciones produce aunque con poca diferencia respecto a las técnicas que ya existían.
- La técnica DRAW_ELEMENTS, si se tiene en cuenta el coste de almacenamiento necesario para implementar cada una de las distintas técnicas, ofrece con diferencia la mejor relación coste/prestaciones.
- La técnica de aceleración que peor resultado ofrece, ARRAY_ELEMENT, resulta peor incluso que el método tradicional (VERTEX).
- Los resultados obtenidos utilizando listas de visualización, que no han sido incluidos en las tablas, han duplicado, y en algunos casos casi triplicado, los fps que se muestran en las tablas. Sin embargo, el coste que supone generar una lista de visualización en tiempo de ejecución es muy alto e impide obtener un movimiento libre de saltos. Es por esto que esta técnica sólo es útil para modelos estáticos.

5. CONCLUSIONES

A raíz de los resultados obtenidos se puede concluir que el uso de las técnicas de aceleración con modelos multirresolución, exceptuando ARRAY_ELEMENT y las listas de visualización, producen mejoras importantes en las prestaciones del sistema gráfico. Además, estas técnicas han podido ser aplicadas sin tener que rediseñar el esquema multirresolución existente.

La investigación se dirige ahora a estudiar otras posibilidades que podrían mejorar las prestaciones sin tener que definir un nuevo esquema multirresolución. En concreto:

- Utilizar métodos que exploten la propiedad de la coherencia.
- Paralelizar el proceso de visualización con el de generación de la aproximación.
- Procesar convenientemente la aproximación en tiempo de ejecución antes de ser visualizada (crear tiras, aumentar la localidad de los vértices, etc).

Agradecimientos

Este trabajo ha sido financiado por el proyecto TIC2001-2416-C03-02 (MCYT, Ministerio de Ciencia y Tecnología).

Referencias

1. C. Andujar, “Simplificación de Modelos de Poliédricos”, *Report LSI-98-1-T, DLSI, UPC*, 1998.
2. O. Belmonte, I. Remolar, J. Ribelles, M. Chover, M. Fernández, “Efficient Use Connectivity Information between Triangles in a Mesh for Real-Ti me Rendering”, *Future Generation Computer Systems, Special issue on Computer Graphics and Geometric Modeling* (2003) (en prensa).

3. A. Bogomjakov, C. Gotsman, "Universal Rendering Sequences for Transparent Vertex Caching of Progressive Meshes", *Computer Graphics Forum*, (21:2) (2002) 137–148.

4. P. Cignoni, C. Montani, and R. Scopigno, A comparison of mesh simplification algorithms, *Computers & Graphics* **22(1)** (1998) 37–54.

5. P. Cignoni, C. Rocchini, and R. Scopigno, Metro: measuring error on simplified surfaces, *Computer Graphics Forum* **17(2)** (1998) 167–174.

6. J. El-Sana, E. Azanli and A. Varshney, Skip Strips: Maintaining triangle strips for view-dependent rendering, *Proc. of Visualization '99* (1999) 131–137.

7. C. Erikson, Polygonal Simplification: An Overview, *Technical Report TR96-016* (Department of Computer Science, UNC-Chapel Hill, 1996).

8. L. De Floriani, P. Magillo, "Multiresolution mesh representation: Models and data structures", *Multiresolution in Geometric Modelling*, (2002) 363–418.

9. M. Garland, Multiresolution Modeling: Survey & Future Opportunities, *State of the Art Reports of EUROGRAPHICS '99* (1999) 111–131.

10. P.S. Heckbert and M. Garland, Survey of polygonal surface simplification algorithms, *Multiresolution Surface Modeling Course Notes of SIGGRAPH'97* (1997).

11. H. Hoppe, Progressive Meshes, *Proc. of SIGGRAPH '96* (1996) 99–108.

12. H. Hoppe, View-Dependent Refinement of Progresive Meshes, *Proc. of SIGGRAPH '97* (1997) 189–198.

13. P. Lindstrom and G. Turk, Evaluation of Memoryless Simplification, *IEEE Transactions on Visualization and Computer Graphics* **5(2)** (1999).

14. D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, R. Huebner, "Level of Detail for 3D Graphics" *Morgan Kaufmann Publishers*, (2003).

15. J. Ribelles, "Multiresolution Ordered Meshes", *PhD Thesis*, Universitat Jaume I, (2000)

16. J. Ribelles, A. López, I. Remolar, O. Belmonte, M. Chover, "Multiresolution Modelling of Polygonal Surface Meshes Using Triangle Fans", *Lecture Notes in Computer Science. Discrete Geometry for Computer Imagery (DGCI 2000)*, (2000) 431–442.

17. J. Ribelles, A. López, O. Belmonte, I. Remolar, M. Chover, "Multiresolution modeling of arbitrary polygonal surfaces: a characterization", *Computers & Graphics*, (26:3) (2002) 449–462.

BUNNY	0.2	0.6	1	1.2	1.6	2	2.2	2.6	3	3.2	3.6	4	Medio
VERTEX	976	324	179	146	95	70	60	49	37	35	27	25	154
ARRAY_ELEMENT	935	215	113	97	67	50	43	34	29	25	22	18	121
DRAW_ELEMENTS	970	323	181	162	117	87	82	70	56	56	50	42	170
DRAW_ARRAYS	1653	321	195	152	120	96	83	75	65	55	54	49	207
VBO_ARRAYS	1756	325	195	160	120	97	86	74	65	60	53	48	211
COW	0.2	0.6	1	1.2	1.6	2	2.2	2.6	3	3.2	3.6	4	Medio
VERTEX	1590	1239	1026	941	810	712	673	600	546	519	475	404	782
ARRAY_ELEMENT	1589	1241	948	942	781	588	581	492	400	400	359	280	717
DRAW_ELEMENTS	1589	1233	1016	938	807	710	670	600	545	520	475	404	781
DRAW_ARRAYS	1761	1229	1005	929	800	700	664	595	537	515	472	396	782
VBO_ARRAYS	1755	1230	1007	930	798	700	660	595	539	516	473	400	782

Tabla 2: *Resultados de los objetos Bunny y Cow en la configuración 1. Evolución del Fps a lo largo del tiempo.*

BUNNY	0.2	0.6	1	1.2	1.6	2	2.2	2.6	3	3.2	3.6	4	Medio
VERTEX	1613	310	169	140	93	64	60	56	46	45	38	32	190
ARRAY_ELEMENT	944	175	99	77	56	44	39	33	29	25	20	20	110
DRAW_ELEMENTS	2494	910	590	510	376	309	280	213	182	167	132	115	494
DRAW_ARRAYS	3972	985	591	496	366	290	263	219	182	178	152	133	570
VBO_ARRAYS	3914	968	610	507	370	304	273	230	209	191	170	159	583
COW	0.2	0.6	1	1.2	1.6	2	2.2	2.6	3	3.2	3.6	4	Medio
VERTEX	3809	2917	1777	1445	1132	911	798	705	603	543	469	327	1286
ARRAY_ELEMENT	3428	1472	874	736	546	432	395	334	235	220	190	151	722
DRAW_ELEMENTS	3934	3109	2732	2570	2188	2026	1918	1688	1530	1384	903	482	2038
DRAW_ARRAYS	3965	3116	2738	2566	2175	2010	1902	1677	1509	1365	894	479	2031
VBO_ARRAYS	3973	3149	2741	2566	2205	2006	1903	1682	1511	1362	892	479	2029

Tabla 3: *Resultados de los objetos Bunny y Cow en la configuración 2. Evolución del Fps a lo largo del tiempo.*

BUNNY	0.2	0.6	1	1.2	1.6	2	2.2	2.6	3	3.2	3.6	4	Medio
VERTEX	1543	394	220	180	132	103	93	78	66	63	54	47	222
ARRAY_ELEMENT	1375	288	160	130	94	73	67	55	45	43	37	33	173
DRAW_ELEMENTS	1586	790	515	446	344	281	266	227	199	188	162	144	415
DRAW_ARRAYS	2003	635	375	310	230	184	166	136	118	112	97	87	342
VBO_ARRAYS	1999	792	522	442	345	281	260	223	197	183	166	151	436
COW	0.2	0.6	1	1.2	1.6	2	2.2	2.6	3	3.2	3.6	4	Medio
VERTEX	1990	1753	1588	1519	1321	1095	1005	857	742	684	566	372	1125
ARRAY_ELEMENT	1994	1762	1507	1288	994	810	738	630	545	505	427	306	957
DRAW_ELEMENTS	1988	1784	1620	1555	1439	1337	1287	1209	1096	1019	746	426	1292
DRAW_ARRAYS	1998	1760	1593	1523	1391	1282	1227	1144	1038	965	717	419	1253
VBO_ARRAYS	2006	1780	1626	1567	1437	1350	1300	1219	1106	1025	750	429	1296

Tabla 4: *Resultados de los objetos Bunny y Cow en la configuración 3. Evolución del Fps a lo largo del tiempo.*