Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas

Desarrollo de algoritmos de teselación adaptativa en GPU

Primer informe de avance

Alumno: Fernando Nellmeldin
Director: Dr. Néstor Calvo

Santa Fe, Septiembre de 2013

Índice

1. Investigación sobre teselación adaptativa

1.1. Estado del arte

En la primer etapa del proyecto, se buscó tener una perspectiva general sobre el estado del arte en teselación. La búsqueda bibliográfica se concentró en trabajos publicados en los últimos 10 años (2003-2013), debido a que el vertiginoso avance en hardware y software causa que los desarrollos queden desactualizados rápidamente. Además, la industria del hardware en placas gráficas ha crecido mucho en los últimos 10 años y la *pipeline* gráfica (secuencia de pasos por los que pasan las primitivas en la placa aceleradora de video antes de renderizarse en la pantalla) ha sido modificada en múltiples ocasiones [1].

Si bien la tecnología que permite teselación adaptativa directamente en las placas aceleradoras de video (GPU) es muy reciente [2], antes de ese lanzamiento también se implementaban algoritmos del área, pero con algunas diferencias. Al no existir una etapa dedicada a teselación en la *pipeline*, los desarrolladores buscaron alternativas. Entre los distintos enfoques existentes, se pueden mencionar la implementación de algoritmos de teselación en la CPU, el uso de patrones de teselación almacenados en la GPU, o la utilización de "trucos" con el geometry shader [3] para generar geometría dinámicamente según demanda.

En el trabajo más antiguo consultado, Moule y McCool [4] propusieron la implementación de un dispositivo hardware que se encargaría de decidir si un triángulo debería ser subdividido. En el año de la publicación de este trabajo (2002), la capacidad de programar en la GPU era muy limitada, lo que llevó a los investigadores a proponer soluciones hardware que se anexarían a la placa gráfica y se encargarían de una única función: realizar teselación en triángulos. Este trabajo evidencia el deseo de los desarrolladores de tener una etapa propia en la GPU que realice teselación de primitivas (triángulos en este caso en particular).

Algunos años más tarde, los trabajos de Schwarz y otros [5], Dyken y otros [6] [7], y Boubekeur y otros [8] mostraban cuál sería la tendencia en teselación adaptativa en GPU. En todos estos trabajos, los autores proponen el almacenamiento en la GPU de patrones de subdivisión para las primitivas. De esta manera, a la GPU sólo se le ingresan los puntos de control de, por ejemplo, una superficie de Bézier, y en una etapa de la placa gráfica se realiza el muestreo de la superficie. Para realizar este muestreo, se toman distintas métricas según la aplicación deseada.

En otra línea de trabajo, Patney y otros [9], junto a Schwarz y otros [10] propusieron realizar teselación en paralelo utilizando CUDA [11]. Debido a la gran carga de procesamiento que requiere subdividir los *patches* (término genérico para referirse a superficies), el uso de un esquema de programación en paralelo es de gran interés para obtener resultados de teselación en tiempo real.

A partir de Direct3D 11 [12] y más tarde en OpenGL 4.0 [13], las placas aceleradoras de video tienen una etapa dedicada a teselación en la *pipeline*. En esta, se especifica qué nivel de subdivisión se quiere realizar en los *patches* y la GPU se encarga de subdividir el dominio automáticamente. De esta manera, se agiliza el diseño de algoritmos de teselación y permite a los programadores concentrarse en cómo elegir un nivel de subdivisión según la escena.

1.2. Métricas de teselación

A la manera de elegir un nivel de subdivisión se le puede denominar métrica de teselación. Se puede esbozar una clasificación de estas métricas según cómo se calculan:

- **Intrínsecas**: se puede computar para cada elemento sin tener en cuenta información externa (curvatura).
- **Extrínsecas**: la posición de la vista y la escena definen la métrica (proximidad, silueta).

Algunas métricas aplicadas en los trabajos consultados son:

- Forma de la silueta de la malla.
- Longitud de una arista.
- Diferencia de normales.
- Curvatura de la superficie.
- Distancia a la cámara.
- Posición de la superficie con respecto a la cámara.

2. Investigación sobre representación de modelos

Luego del análisis del estado del arte en teselación adaptativa, se vio la necesidad de tomar conocimiento sobre la representación de modelos en computadora. Específicamente, y como la mayoría de los algoritmos de teselación operan sobre el dominio de superficies paramétricas, se consultó el libro "Curves and surfaces for computer graphics" de David Salomon [14]. Se hizo especial hincapié en los capítulos que exponen los temas de curvas y superficies de Bézier, B-Splines, NURBS, y superficies de subdivisión de Doo-Sabin, Catmull-Clark y Loop.

Para el desarrollo del proyecto se deberá concentrar el interés en diseñar técnicas de teselación que operen sobre las superficies antes mencionadas. Por lo tanto, el aprendizaje de las bases matemáticas y geométricas que utilizan estas superficies es de importancia primordial para la continuación del proyecto.

3. Búsqueda bibliográfica sobre bibliotecas y herramientas

Para llevar a cabo el proyecto, se tuvo que elegir entre un conjunto de herramientas y bibliotecas que facilitaran la programación de los algoritmos de teselación. En particular, se debería elegir una API gráfica para posibilitar la interacción entre la CPU y la GPU. Una API (*Application Programming Interface*) es un conjunto de funciones cuyo objetivo es el de brindar una interfaz entre aplicaciones software y el hardware de la computadora. Además de tomar conocimiento sobre las API gráficas, se vio la necesidad de estudiar la *pipeline* de las placas aceleradoras de video de la actualidad.

Por un lado, se buscó información sobre la *pipeline* gráfica de las GPU. Para ello se consultaron los capítulos referidos a este tema en el libro "Real-Time Rendering" de Tomas

Akenine-Möller y otros [15]. A partir del conocimiento de las etapas por las que pasa una primitiva en la GPU (triángulos, por ejemplo), se puede realizar un análisis más completo sobre las API gráficas que operan sobre la pipeline.

Por otro lado, se buscó información sobre el estado de desarrollo de las API gráficas. Se concluyó que las principales bibliotecas que implementan teselación directamente en GPU en la actualidad son dos: Direct3D en su versión 11 lanzada en 2009; y OpenGL 4.0 lanzada en 2010.

Para permitir la programación en GPU, se debe definir un programa específico llamado *shader* con instrucciones que se ejecutarán directamente en la GPU. Las APIs antes mencionadas definen su propio lenguaje de shading. En particular, Direct3D define el HLSL (High Level Shading Language), mientras que OpenGL define el GLSL (OpenGL Shading Language). Cada programa de shader se debe relacionar con una de las etapas programables del pipeline de la GPU. Por ejemplo, si se quiere escribir un shader que opere sólo sobre los vértices de las primitivas, se debe adjuntar un programa de vertex shader a la pipeline gráfica.

4. Análisis de bibliotecas y herramientas

4.1. Funcionalidad de las APIs

Direct3D está enfocado principalmente en videojuegos, de allí que los desarrolladores de esta API se preocupan por maximizar la performance de los videojuegos realizados con Direct3D. OpenGL, en cambio, tiene un uso más extendido. Además de ser utilizada en videojuegos (aunque con menor adopción [16] [17] [18]), es de utilidad para la visualización de aplicaciones donde el dibujado no es lo importante, si no que lo son los cálculos subyacentes realizados. Es por ello que los proyectos de investigación eligen utilizar OpenGL para la presentación visual de los resultados. Además, la característica multi-plataforma de OpenGL le brinda una ventaja adicional, ya que permite que se puedan presentar los resultados en cualquier software compatible.

4.2. Teselación en la API

En las placas aceleradoras de video de última generación, capaces de ejecutar las versiones Direct3D 11 y OpenGL 4.0, se ha incorporado una etapa adicional correspondiente a teselación (ver Fig. 1). La utilización del shader que opera en esta etapa es de principal importancia para la realización del proyecto, por lo que éste sólo puede llevarse a cabo utilizando las versiones específicas mencionadas (o superiores).

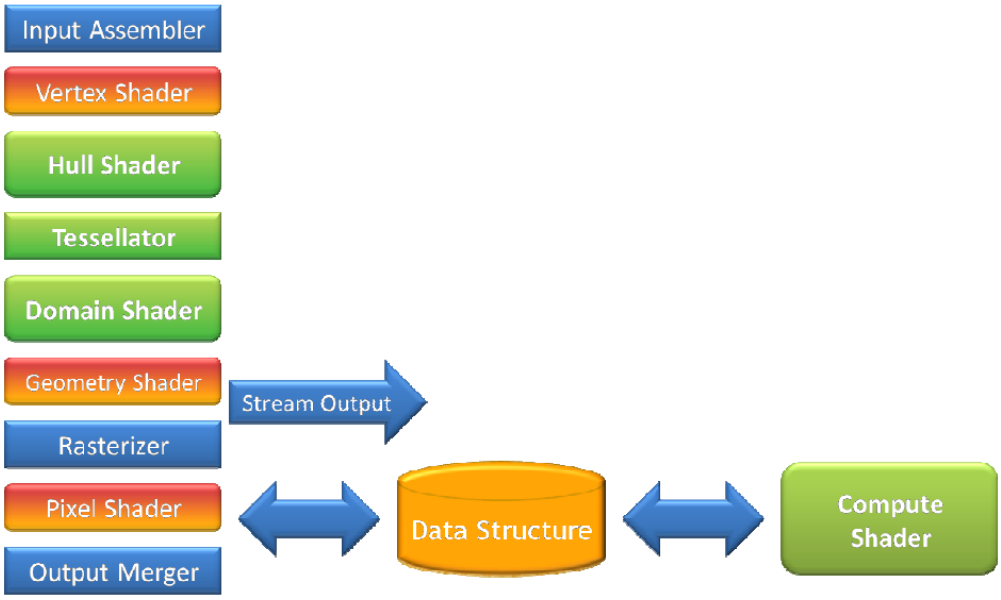

Figura 1: Pipeline de Direct3D 11. Las etapas en verde son las nuevas incorporaciones en la versión 11.

5. Selección de bibliotecas y herramientas

5.1. API gráfica

Luego del análisis de la pipeline gráfica y las API para interactuar con la misma, se decidió continuar el proyecto utilizando **OpenGL 4.0**. Las razones de la elección se resumen a las siguientes:

1. *Multiplataforma*: al ser una biblioteca multiplataforma, OpenGL se encuentra disponible para ser utilizada en todos los sistemas operativos convencionales. De esta manera, el proyecto que se desarrolle no estará destinado a un sistema operativo en particular, y podrá ser ejecutado en cualquier sistema sin realizar cambios sustanciales. En contraste, Direct3D 11 tiene la limitación de que se necesita un sistema operativo Microsoft Windows 7 o superior para ser utilizado.
2. *Material existente*: durante la investigación sobre teselación, se consultaron diversos trabajos que implementaban técnicas de teselación en GPU utilizando las últimas bibliotecas disponibles. Sin embargo, gran porcentaje de los trabajos utilizaban Direct3D 11 como API para interactuar con el hardware. Esto puede ser debido a que la versión de Direct3D capaz de ejecutar el shader de teselación, fue anunciada dos años antes que la correspondiente a OpenGL. Entonces, en el presente proyecto, se elige la utilización de OpenGL para realizar un aporte diferente a los disponibles, y presentar la alternativa multiplataforma para realizar teselación en GPU.
3. *Utilización en la universidad*: en los cursos universitarios de Computación Gráfica, tanto locales como nacionales, y posiblemente internacionales, se opta por dictarlos utilizando la biblioteca OpenGL. Por lo tanto, y como se pretende que el proyecto sea puntapié inicial para estimular el desarrollo local de nuevas tecnologías, el uso de una biblioteca que el alumno interesado ya conoce, facilita su inserción en la temática.

5.2. Otras herramientas

Además de la elección de la API, para el desarrollo del software se necesitó elegir algunas herramientas adicionales para facilitar ciertos aspectos de la programación del mismo. A continuación se detallan las herramientas elegidas:

- Carga de imágenes en memoria: se eligió a la biblioteca **DevIL** [19] debido a su facilidad para cargar imágenes y texturas a ser utilizadas en el desarrollo. Además, al ser una biblioteca multiplataforma, no existirán problemas compatibilidad entre sistemas operativos.
- Carga de modelos en memoria: se eligió la biblioteca **Assimp** [20] dado que es portable y permite la importación de modelos tridimensionales de manera uniforme y rápida.
- Estructuras de datos: se eligió utilizar **GLM** [21] ya que es una biblioteca diseñada especialmente para interactuar con el lenguaje de *shading* de OpenGL. Esta biblioteca facilita la definición de vectores y matrices que luego serán transmitidos a la GPU para realizar cálculos. Debido a que expone una sintaxis similar al GLSL, no se tendrá dificultad en el aprendizaje de su uso.

6. Adquisición de experiencia en el desarrollo

Antes de poder iniciar cualquier desarrollo de algoritmos de teselación, surgió la necesidad de tomar práctica con las herramientas seleccionadas. Si bien se tenía experiencia previa en el desarrollo con OpenGL, se optó por hacer un repaso rápido de los conocimientos ya adquiridos, así como también tomar contacto con otros nuevos. En particular, se tenía experiencia en el desarrollo con OpenGL 2.0, pero para continuar con el proyecto se necesitan características sólo disponibles a partir de la versión OpenGL 4.0.

Para realizar dicha renovación de conocimientos, se siguieron los ejemplos expuestos en el libro "OpenGL 4.0 shading language cookbook" de David Wolff [22]. En particular, se exploraron temas tales como el uso de programas de shaders, modelado de iluminación y sombreado, uso de texturas, y renderizado de modelos 3D. Para terminar con este apartado, se hizo una exploración inicial sobre la utilización del shader de teselación para generar un modelo tridimensional simple cuyo nivel de detalle se pueda modificar dinámicamente según lo desee el usuario (ver Fig. 2).

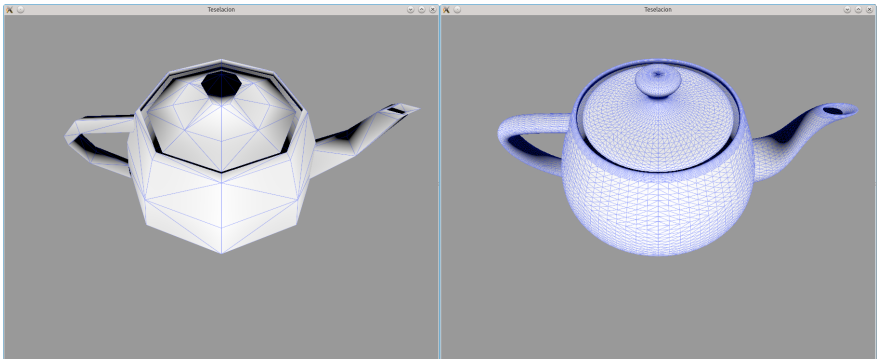

Figura 2: Teselación de un modelo tridimensional.

Bibliografía

[1] Tatarchuk, N., Barczak, J., y Bilodeau, B., *"Programming for Real-Time Tessellation on GPU."* AMD whitepaper 5 (2010).

[2] Gee, K., *"DirectX 11 Tessellation"*. Microsoft GameFest, 2008.

[3] Rost, R., *"OpenGL Shading Language"*, Second Edition, 2006.

[4] Moule, K. y McCool, M. D., *"Efficient Bounded Adaptive Tessellation of Displacement Maps."* Graphics Interface, 2002.

[5] Schwarz, M., Staginski, M. y Stamminger, M., *"GPU-Based Rendering of PN Triangle Meshes with Adaptive Tessellation."* Vision, Modeling, and Visualization 2006, pg. 161-168, 2006.

[6] Dyken, C., Reimers, M. y Seland, J., *"Real-Time GPU Silhouette Refinement using Adaptively Blended Bézier Patches."* Computer Graphics Forum. Vol. 27. No. 1. Blackwell Publishing Ltd, 2008.

[7] Dyken, C., Reimers, M. y Seland, J., *"Semi-Uniform Adaptive Patch Tessellation"* Computer Graphics Forum. Vol. 28. No. 8. Blackwell Publishing Ltd, 2009.

[8] Boubekeur, T., y Schlick C., *"A Flexible Kernel for Adaptive Mesh Refinement on GPU."* Computer Graphics Forum. Vol. 27. No. 1. Blackwell Publishing Ltd, 2008.

[9] Patney, A., y Owens, J., *"Real-time Reyes-style Adaptive Surface Subdivision."* ACM Transactions on Graphics (TOG). Vol. 27. No. 5. ACM, 2008.

[10] Schwarz, M., y Stamminger, M., *"Fast GPU-based Adaptive Tessellation with CUDA"* Computer Graphics Forum. Vol. 28. No. 2. Blackwell Publishing Ltd, 2009.

[11] NVIDIA Corporation, *"NVIDIA CUDA Programming Guide 2.0"*, 2008.

[12] DirectX: API para facilitar el manejo de recursos multimedia. 1995-2013. Disponible en `http://windows.microsoft.com/es-AR/windows7/products/features/directx-11`. [Consultado el 10 de Junio de 2013].

[13] OpenGL: API para el renderizado de gráficos de computadora en 3D. 1992-2013. Disponible en `http://www.opengl.org`. [Consultado el 10 de Junio de 2013].

[14] Salomon, D., *"Curves and Surfaces for Computer Graphics"*, Springer Editorial, 2006.

[15] Akenine-Möller, T., Haines, E., y Hoffman, N., *"Real-Time Rendering"*, Third Edition, 2008.

[16] Abi-Chahla, F., *"OpenGL 3 & DirectX 11: The War Is Over"*. Tom's Hardware. 2008. Disponible en `http://www.tomshardware.com/reviews/opengl-directx,2019.html`. [Consultado el 23 de Julio de 2013]

[17] Bolas, N., *"Why do game developers prefer Windows?"*. Programmers Stack Exchange, 2011. Disponible en `http://programmers.stackexchange.com/questions/60544/why-do-game-developers-prefer-windows`. [Consultado el 23 de Julio de 2013]

[18] Riccio, C., *"A look at OpenGL and Direct3D performance with Unigine"*. G-Truc Creation, 2013. Disponible en `http://www.g-truc.net/post-0547.html`. [Consultado el 23 de Julio de 2013]

[19] DevIL: biblioteca multiplataforma para carga de imágenes. 2000-2010. Disponible en `http://openil.sourceforge.net/`. [Consultado el 29 de Agosto de 2013].

[20] Assimp (Open Asset Import Library): biblioteca portable para la importación de modelos tridimensionales. 2008-2012. Disponible en `http://assimp.sourceforge.net/`. [Consultado el 29 de Agosto de 2013].

[21] GLM (OpenGL Mathematics): biblioteca multiplataforma para programación gráfica. 2005-2013. Disponible en `http://glm.g-truc.net/`. [Consultado el 29 de Agosto de 2013].

[22] Wolff, D., *"OpenGL 4.0 shading language cookbook"*, First Edition, Packt publishing, 2011.

REALIZADO POR	FECHA	FIRMA
Fernando Nellmeldin	29/08/2013	
REVISADO POR	**FECHA**	**FIRMA**
Dr. Néstor Calvo	02/09/2013	
APROBADO POR	**FECHA**	**FIRMA**

Nombre del Proyecto: Desarrollo de algoritmos de teselación adaptativa en GPU

Periodo del Informe: Julio-Agosto de 2013

Alcance: Etapas 1 y 2

Estado del proyecto:

Cronograma

Etapa 1: Lectura de bibliografía.

Actividad	Fecha realización	
	Estimada	Real
1.1 Investigación sobre el estado del arte en teselación.	40 horas	40 horas
1.2 Investigación sobre representación de modelos tridimensionales.	20 horas	16 horas

Resultados obtenidos
Se consultaron diversos autores referentes del área de computación gráfica y diseñadores de hardware de empresas fabricantes de placas aceleradoras gráficas (GPU).

Etapa 2: Análisis y selección de herramientas.

Actividad	Fecha realización	
	Estimada	Real
2.1 Búsqueda bibliográfica sobre bibliotecas y herramientas de desarrollo de software.	20 horas	15 horas
2.2 Análisis de bibliotecas y herramientas a utilizar.	20 horas	15 horas
2.3 Selección de bibliotecas y herramientas.	10 horas	8 horas
2.4 Adquisición de experiencia en el desarrollo utilizando las bibliotecas y herramientas seleccionadas.	40 horas	60 horas
2.5 Redacción del primer informe de avance.	15 horas	10 horas

Resultados obtenidos
Se analizaron las alternativas de software para el desarrollo del resto del proyecto. Se eligió OpenGL y se desarrollaron diversos ejemplos para adquirir práctica con la biblioteca.

Riesgos

Riesgo	Se efectivizó		Impacto	Mitigación
Pérdida de recursos hardware	~~Sí~~	No		
Pérdida del desarrollo o investigación realizados	~~Sí~~	No		
Cambio de tecnologías con las que se realiza el desarrollo	~~Sí~~	No		

Riesgos futuros	Probabilidad	Impacto

Notas