Unidad de Procesamiento Gráfico

En los últimos años, el hardware gráfico ha experimentado una notoria evolución tecnológica, manteniendo un costo relativamente bajo y así facilitando su acceso a gran parte de la población.

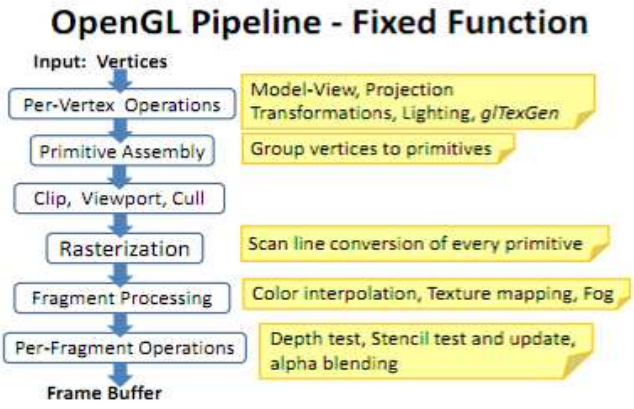
La GPU (Graphics Processing Unit o Unidad de Procesamiento Gráfico) es un coprocesador dedicado al procesamiento de gráficos. Su finalidad es alivianar la carga de trabajo de la CPU (Procesador Central) principalmente en aplicaciones 3D interactivas, produciendo una mejora general en el rendimiento del sistema. Posee su propio espacio de memoria y esto se traduce en una potencia de cálculo mucho mayor gracias a su arquitectura en paralelo, lo que le permite ejecutar paralelamente un gran número de *threads*. Ha pasado, de poseer la sola capacidad de ejecutar una parte fundamental del pipeline gráfico con algoritmos rígidamente implementados, a poder implementar una estructura de renderizado programable, con la capacidad de vincularse dinámicamente con la aplicación gráfica. La programación sobre la GPU permite incrementar sus prestaciones.

Actualmente la GPU está preparada para la aceleración de ciertas funciones gráficas, junto con la posibilidad de realizar cálculos en paralelo sobre puntos, vectores y matrices, haciendo que la programación sobre la GPU sea más eficiente que la programación sobre la CPU para tareas relacionadas con operaciones gráficas.

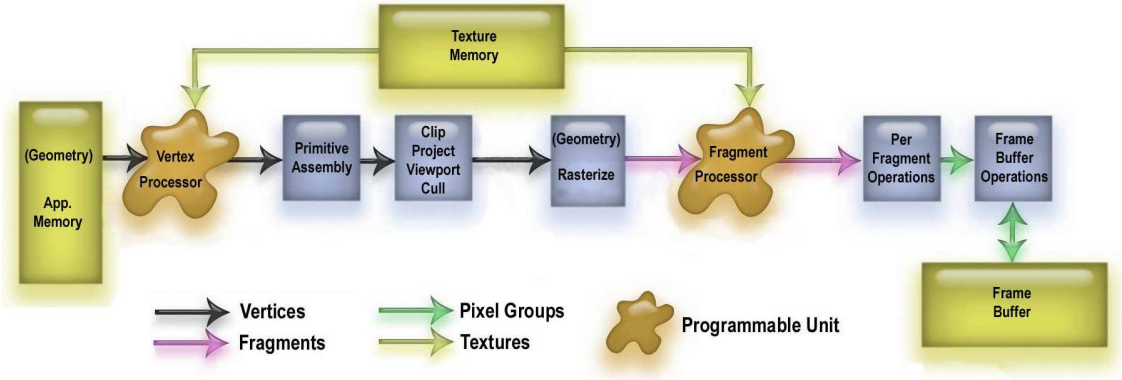
OpenGL y GLSL

Los gráficos son generados mediante el cómputo de una serie de etapas, que habitualmente es el llamado *rendering pipeline* o *tubería de renderizado*. Consiste en una serie de pasos más o menos rígidos que se realizan en el hardware gráfico y que transforman la escena en una imagen. La mayoría de las funciones de OpenGL emiten primitivas al pipeline gráfico o bien configuran la manera en la cual el *rendering pipeline* procesa dichas primitivas.

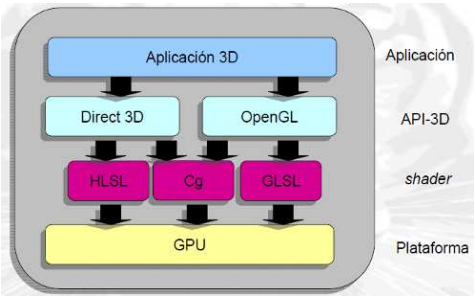
Desde la primera especificación de OpenGL que fue publicada en 1992 hasta la versión 1.5 del 2003, en cada etapa del pipeline se ejecutaba una funcionalidad que estaba prefijada (*fixed function*), resultando poco configurable, es decir, el usuario solamente podía especificar algunos parámetros de configuración, pero el funcionamiento y el proceso de shading era siempre el mismo.



A partir de la versión 2.0 publicada en el 2004 algunas etapas del *rendering pipeline* pasaron a ser programables. Esta programabilidad permite al usuario la posibilidad de escribir programas (*shaders* o *programs*) con un lenguaje específico y que serán ejecutados por la GPU, en alguno de sus procesadores (*shaders processors*) destinados a ese fin.



Existen varios lenguajes para programar la GPU, entre ellos GLSL (OpenGL Shading Language), HLSL y Cg.



GLSL es un lenguaje de programación de alto nivel basado en C que permite crear programas que serán ejecutados en la GPU. Posee muchos aspectos similares a C, como las palabras reservadas y la sintaxis, además la ejecución de un programa comienza en la función main() y termina una vez que el programa sale de esta función.

En este curso nos enfocaremos en GLSL debido a su continuo crecimiento y a que es parte de OpenGL.

La principal ventaja de la programabilidad es la gran mejora en el rendimiento de los cálculos de iluminación y transformación de geometría, aplicándose sobre un gran conjunto de elementos al mismo tiempo. También permite incrementar las prestaciones, pudiéndose generar una amplia gama de efectos de renderizado, por ejemplo materiales no fotorealísticos o el uso de un modelo de iluminación propio en lugar de usar el modelo de Blinn-Phong adoptado por OpenGL.

Con la publicación del 2004 empezaron a ser programables la etapa de procesamiento de vértices (*vertex processor)* a través del *vertex program* y la etapa de procesamiento de fragmentos (*fragment processor*) por medio del *fragment program*. Luego, en el 2009 se incorporó el *geometry processor* para procesar las primitivas y otras operaciones y en el 2010 se agregó el *tessellation processor* para subdividir las primitivas.

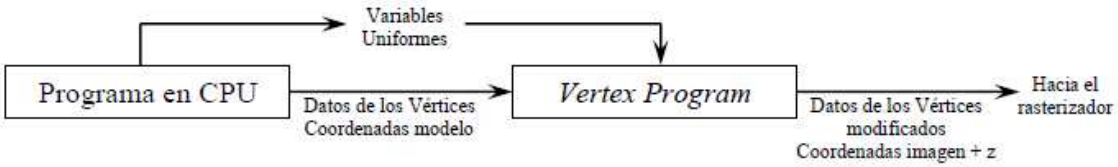
Versión GLSL	Versión OpenGL	Año	Nota
---	1.5	2003	Funcionalidad fija del rendering pipeline
1.10.59	2.0	2004	Soporte para GLSL
1.20.8	2.1	2006	
1.30.10	3.0	2008	Muchas características de OpenGL son deprecadas
1.50.11	3.2	2009	Se incorpora el *geometry processor/program*
4.00.9	4.0	2010	Se incorpora el *tessellation processor/program*
4.30.8	4.3	2012	Se incorpora *compute program*

En este curso veremos solamente la programación del *vertex processor* y del *fragment processor*, es decir, nos acotaremos al uso de GLSL 1.20.

Vertex processor – Vertex program

El *vertex processor* o procesador de vértices es una unidad programable que opera en los vértices y sus datos asociados. El procesador de vértices suele hacer operaciones gráficas tradicionales como: transformación de los vértices y las normales desde el espacio del modelo al visual, manteniendo la coordenada z para oclusión; también se encarga de la transformación de las coordenadas de texturas y los cálculos de iluminación.

El script gráfico que se ejecutará en el *vertex processor* se llama *vertex program*, siendo invocado por cada uno de los vértices enviados desde la aplicación o programa en CPU.

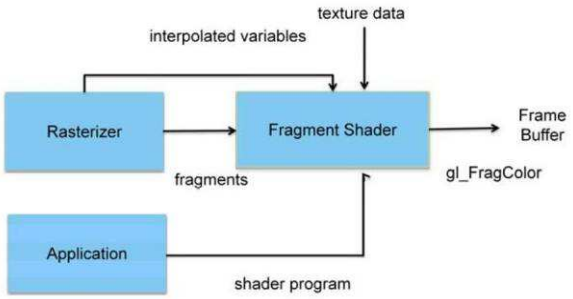


El *vertex program* no genera nuevos vértices, ni elimina existentes, simplemente los procesa. Espera como mínimo la posición del vértice (*gl_Vertex*) y genera como mínimo la posición transformada (*gl_Position*) a coordenadas de la imagen, con z. Para cada vértice, puede mantener información asociada, como ser: color, alpha, normal y coordenadas textura (variables predefinidas o *built-in*) y otras variables del vértice (por ejemplo temperatura) que el usuario puede agregar.

Con el *vertex program* se pueden mover los vértices para lograr determinados efectos, como los que tienen que ver con la deformación en tiempo real de un objeto.

Fragment processor – fragment program

El *fragment processor* o procesador de fragmentos es una unidad programable que trabaja sobre cada fragmento generado durante el proceso de rasterización y sobre sus datos asociados. El *fragment processor* puede ser programado mediante un *fragment program*. Aquí normalmente se calcula la iluminación individual de cada fragmento (*Phong shading* o cualquier otro procedimiento de *per-fragment shading*) y se mezcla con el color de la textura y el de la neblina (*fog*) correspondiente a ese fragmento; siendo posible también la modificación en la profundidad del fragmento (se puede cambiar *z* pero no las coordenadas *x* e *y*, ya ligadas al píxel).



Un *fragment program*, como mínimo, tiene que asignar un color al fragmento (*gl_FragColor*). El acceso a los fragmentos vecinos no está permitido, lo cual es esperable ya que el cálculo de los fragmentos se realiza en paralelo.

El procesador de fragmentos finalmente devuelve, por cada fragmento procesado, el color para un píxel dado. También existe la posibilidad de descartar el fragmento (*discard*) y salir sin retornar nada, indicando que el fragmento no genera ninguna actualización de color para su pixel asociado.

El color del fragmento es la mezcla del resultado de la iluminación, mezclado con el color de neblina o *fog* y el color de la imagen de textura según el modo en que estas operaciones se encuentren definidas.

Este tratamiento individual de los fragmentos permite realizar iluminación por pixel, utilizar esquemas complejos de iluminación y generar efectos muy interesantes.

Comunicación de los shaders con la aplicación

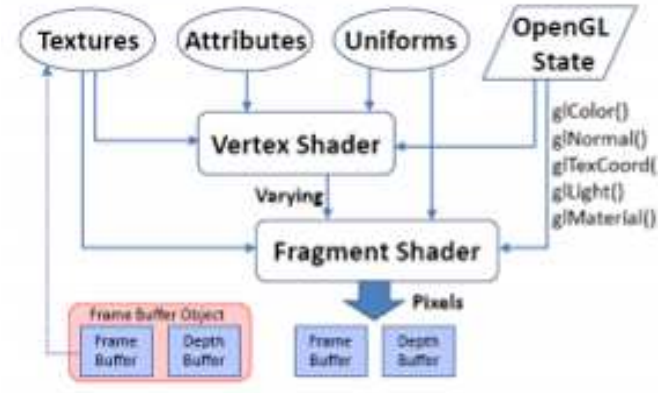
Existen diferentes tipos de variables para comunicar los programas en la GPU con el programa en CPU, y entre ellos mismos. En OpenGL, en el programa en GPU, son todas variables de estado, pero hay variables que se aplican a toda una primitiva y otras que pueden cambiar entre los vértices de una primitiva, es decir que pueden incluirse dentro del grupo glBegin() - glEnd(), donde se definen los vértices de cada primitiva. En principio, las variables que se pueden cambiar por vértice son posición, normal, color y coordenadas de textura. Las variables que reciben se dividen en tres categorías:

Uniform: Son los datos que no cambian frecuentemente; a lo sumo una vez por primitiva, es decir que son las que no pueden ir en el grupo glBegin() – glEnd(). Los *shaders* pueden utilizarlas pero no pueden modificarlas (*read-only*).

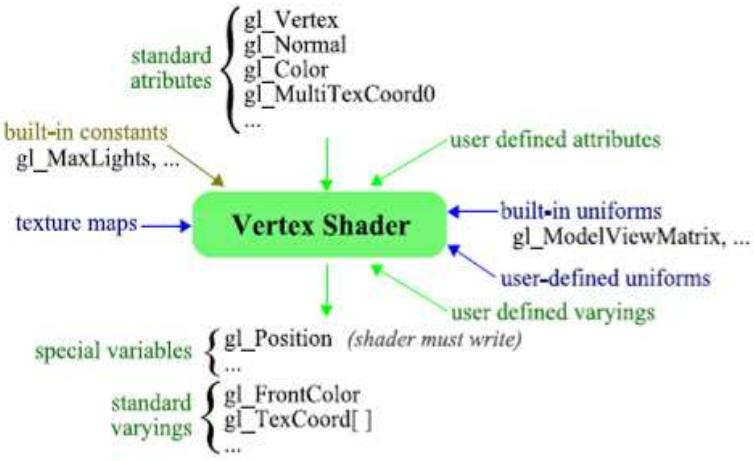
Attribute: Son los datos que se corresponden a los vértices y son usadas en los *vertex program*. El valor de esta variable es actualizado automáticamente con la información correspondiente al vértice y puede estar dentro del grupo glBegin() – glEnd(). Éste tipo de variable al igual que el anterior es de solo lectura. El programador puede incorporar variables attribute de propósito general (por ejemplo la temperatura).

Varying: Son utilizadas para pasar información entre el *vertex program* y el *fragment program*. Los valores de estas variables son generados en el *vertex program* para cada vértice, luego son interpolados, para cada fragmento, al rasterizar las primitivas y pasados al *fragment program*. Deben ser declaradas tanto en el *vertex program* como en el *fragment program* con el mismo tipo.

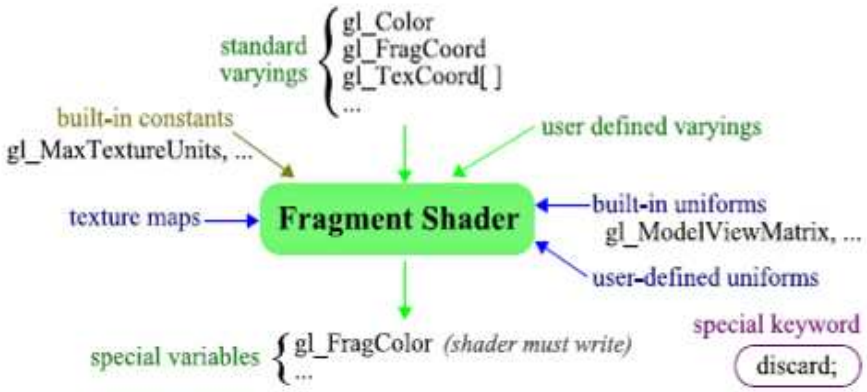
Los tres tipos de variables son globales en cada programa. Para los tres tipos hay algunas variables *built-in* (prefabricadas) que no hace falta definir, como por ejemplo, la posición final del vértice.



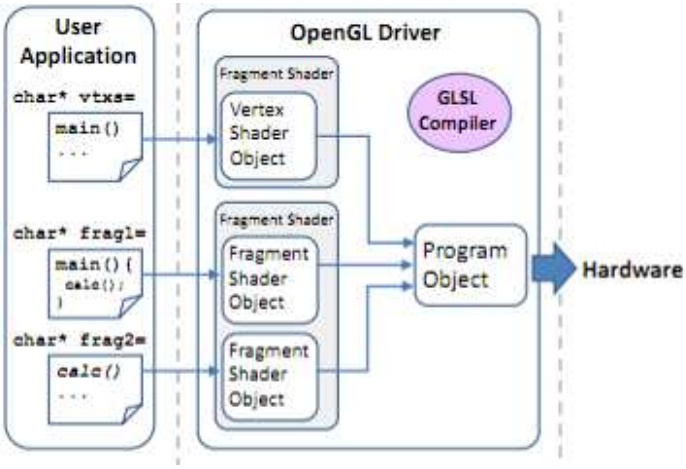
***Input/Ouput* en el *vertex program*:**



***Input/Ouput* en el *fragment program*:**



Puesta en marcha de GLSL



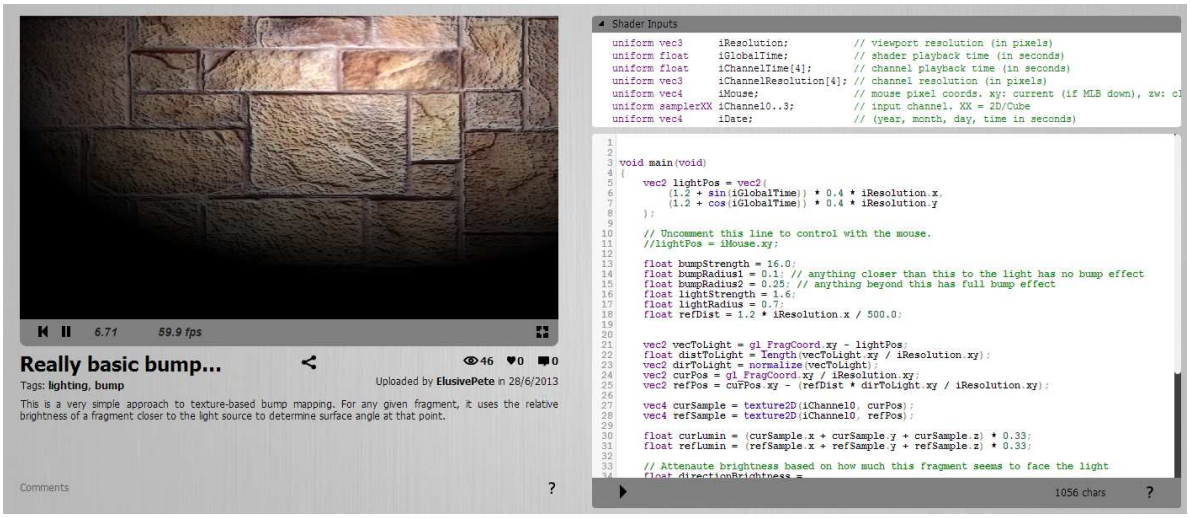
Un programa para GPU *o shader* es un archivo que contiene el código fuente que se ejecutará en el *vertex processor* o en el *fragment processor*.

Los pasos a seguir para utilizar un *shader* son:

- Leer el código fuente del *shader* a partir de un archivo.
- Crear los objetos *shaders* y el objeto programa.
- Cargar los objetos *shaders* con el código leído del archivo fuente.
- Compilar los objetos *shaders*.
- Adjuntar los *shaders* al objeto programa.
- Ligar el objeto programa con nuestra aplicación.
- Indicar a la aplicación que utilice nuestro objeto programa en lugar de las funciones de OpenGL.

Editor web de Shaders

Existen algunas páginas web que proporcionan la posibilidad de crear *shaders* y probarlos en el browser gracias a la utilización de WebGL, que una especificación que permite utilizar OpenGL en browsers. Una de estas páginas es ShaderToy (https://www.shadertoy.com/) en la cual se pueden crear *fragment programs* y visualizarlos en tiempo real. Para utilizar esta página es necesario utilizar un browser que soporte WebGL como Chrome o Firefox. A continuación se puede ver una captura de esta página:



En la sección izquierda de la página se puede ver una vista previa del resultado de aplicar el *shader*, mientras que en la sección derecha se encuentran las entradas del *shader* y el código de la función main del *fragment program*.

Una limitación que presenta la página ShaderToy es la imposibilidad de crear *vertex programs* por lo cual no se pueden aplicar efectos sobre mallas de vértices, sino que solo se pueden realizar efectos sobre imágenes con la utilización de solo un *fragment program*.

BIBLIOGRAFÍA

OpenGL Shading Language OrangeBook. (Second Edition) .2006

Lighthouse3d.com: Tutorial [en línea] http://www.lighthouse3d.com/tutorials/glsl-tutorial/