Revista Venezolana de Computación
ISSN: 2244-7040
http://www.svc.net.ve/revecom
Vol. 2, No. 1, pp. 70-82, Diciembre 2015
Selección de los Mejores Artículos de CoNCISa 2015

Shader Tool: Un Ambiente Web de Desarrollo Integrado para GLSL

Luiyit Hernández[1], Esmitt Ramírez[1]
luiyit.hernandez@gmail.com, esmitt.ramirez@ciens.ucv.ve

[1] Escuela de Computación, Universidad Central de Venezuela, Caracas, Venezuela

Resumen: Un shader se define como una unidad de programa escrita en un lenguaje de sombreado para la GPU, con la finalidad de producir imágenes (*render*). Actualmente, existen herramientas y tecnologías Web que realizan el despliegue de este contenido en navegadores basados en HTML5. Por ello, se ha impulsado el desarrollo de soluciones que apoyan la implementación de programas de shaders para el lenguaje GLSL, parte del estándar OpenGL. Sin embargo, de acuerdo a nuestra investigación, no existen aplicaciones Web disponibles con un criterio unificado y con todos los elementos necesarios involucrados en el proceso de desarrollo. En este trabajo se presenta una solución llamada Shader Tool que contiene un conjunto de herramientas que incorporan y enlazan todos los elementos requeridos para el desarrollo de shaders. La solución emplea plantillas, gráficos 3D, interactividad y tecnología de sincronización para la implementación de código GLSL. De acuerdo a los resultados obtenidos en las pruebas realizadas, se determinó que la solución optimiza adecuadamente los procesos y cuenta con un nivel de rendimiento significativamente alto. Así, se muestra el potencial de la aplicación como un ambiente integrado completo para el desarrollo de shaders en la Web.

Palabras Clave: Shader; Programación en la GPU; GLSL; MVC; WebGL; GPU.

Abstract: A shader is defined as a unit of program written in a shading language to the GPU to produce images (render). Nowadays, there are tools and Web technologies which used the display of this content in HTML5-based browsers. Thus, it has guided the development of solution to support the implementation of shader programs to the GLSL language, as part of OpenGL standard. However, being our research, there are not available Web applications with unified standard and which contains all involved elements in the development process. In this paper, a solution called Shader Tool is presented, it contains a set of tools which incorporated and linked all features to the shader development. This is a tool which uses templates, 3D graphics, interactivity and synchronization technologies to the implementation of GLSL code. According to the results of the tests, it was determined that the solution adequately optimizes processes and has a significantly higher level of performance. In this way, we shown the potential of the application as a complete integrated environment to the development of shaders in the Web.

Keywords: Shader; GPU Programming; GLSL; MVC; WebGL; GPU.

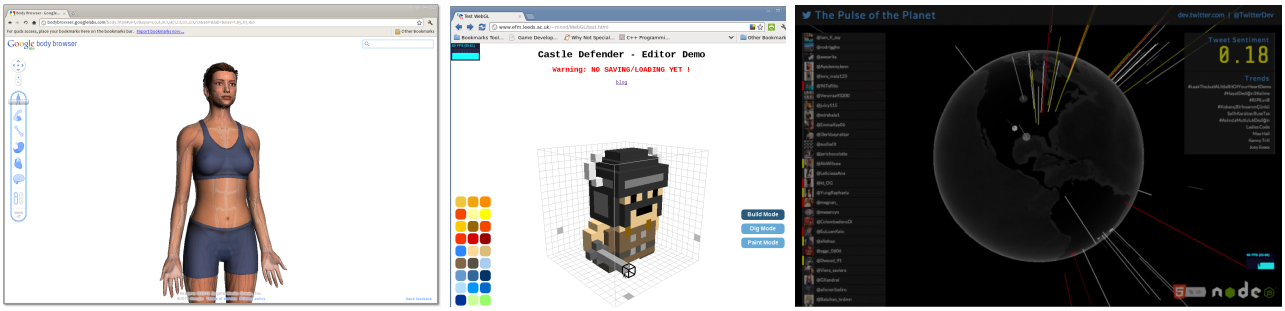
I. Introducción

Un software de procesamiento gráfico 3D permite la creación y manipulación de modelos 3D por computadora. Estas aplicaciones son empleadas tanto para la construcción de imágenes como para la animación por computadora. En la actualidad existe un número considerable de aplicaciones con propósitos muy diversos y de ámbito *standalone* y *online*.

Estas aplicaciones han permitido el crecimiento en la manera de percibir y crear contenido gráfico, generando cambios sustanciales en diversas áreas. Desde el año 2013, los navegadores han aumentado su rendimiento y son capaces de ofrecer una plataforma para albergar aplicaciones de alto procesamiento gráficos. La mayoría de los navegadores han adoptado la tecnología WebGL [1], que permite crear aplicaciones con un rendimiento óptimo de gráficos 2D y 3D en el navegador, empleando la unidad de procesamiento gráfico (GPU).

Actualmente, existen herramientas en la Web basadas en WebGL y bibliotecas especializadas en el área de los gráficos 2D/3D, que recrean un ambiente de desarrollo de aplicaciones gráficas. Del mismo modo, muchas de éstas incluyen programas especiales dirigidos a ser ejecutados en los procesadores gráficos, llamados shaders. Dichos programas son ejecutados de manera independiente por la GPU de un computador para un cierto programa de despliegue gráfico.

La forma tradicional de construcción de shaders es empleando un computador con alguna aplicación standalone. Dado que dichos programas no están enfocados para la Web, es ideal contar con las mismas capacidades de desarrollo en el entorno



(a) Google Body Browser (b) Editor 3D Castle Defender (c) The Pulse of the Planet - Twitter

Figura 1: Ejemplos del Uso de WebGL en Diversas Aplicaciones en la Web

Web. Sin embargo, en soluciones actuales disponibles no existe un criterio unificado de implementación. De hecho, en algunas aplicaciones no se percibe la incorporación de todos los elementos requeridos para realizar la programación y ejecución de shaders en GLSL.

En este trabajo se presenta el desarrollo de un sistema integrado para el desarrollo de shaders para el lenguaje de OpenGL, haciendo uso de tecnología Web de vanguardia, y de arquitecturas pensadas para ofrecer un óptimo rendimiento y opciones de escalabilidad. De esta manera se busca proveer un conjunto de herramientas Web que incorporen y enlacen todos los elementos involucrados en el desarrollo de programas GLSL, además de proporcionar accesos a la información relacionada con éstos. Shader Tool, nombre de la solución propuesta, permite un libre acceso a una biblioteca de recursos dentro de una plataforma de usuarios que está disponible desde cualquier dispositivo con soporte a un browser con HTML5. De forma puntual, nuestras principales contribuciones son:

- Una solución computacional para la creación de shaders en GLSL bajo un entorno de desarrollo Web.
- Un conjunto de librerías de recursos para la programación de shaders.
- Un sistema Web que almacena los trabajos realizados por cada usuario dentro de Shader Tool.

Este trabajo se organiza como sigue: en la Sección II es presentada una breve introducción a las tecnologías para el despliegue en la Web. La Sección III muestra una recopilación de los trabajos previos relevantes para nuestra investigación, así como un cuadro comparativo de las necesidades en un sistema integrado de desarrollo para la construcción de shaders. Luego, la Sección IV explica en detalle todos los aspectos asociados a Shader Tool como parte de nuestra investigación. La Sección V presenta los detalles propios de la interfaz de la aplicación Web. Una serie de experimentos para corroborar nuestro trabajo se muestra en la Sección VI. Por último, la Sección VII muestra una serie de conclusiones y recomendaciones a ser aplicadas.

II. Tecnologías para el Despliegue en la Web

La Web siempre ha sido un medio visual, sin embargo hasta hace poco, fue de uso exclusivo de un grupo de aplicaciones específicas. La mayoría portales Web basados en CSS y JavaScript, gracias a la incorporación de tecnologías como el elemento canvas de HTML5 [2], la especificación WebGL [1] y las imágenes SVG [3], el desarrollo Web ahora es más dinámico y adaptable. El uso combinado de toda esta tecnología puede dar como resultados aplicaciones, juegos o herramientas de alto impacto. La Web ofrece posibilidades de uso diversos y los editores de shaders es uno de ellos.

El despliegue de gráficos 3D ha tomado un gran impulso en los últimos años, especialmente en el ámbito de los videojuegos. Por ello se han desarrollado APIs especializadas para facilitar los procesos en todas las etapas de generación de gráficos por computador. Éstas han demostrado ser primordiales para los fabricantes de componentes, proporcionando un medio de acceso al hardware de una manera abstracta.

Desde hace dos décadas esta tendencia ha ido aumentando y se unió a la tecnología Web existente. De esta forma, surgen nuevas formas de producir gráficos 2D/3D y demandas en el ámbito de la Informática Gráfica. Una gran tendencia de desarrollo para su incorporación fue el uso de lenguajes de script para la Web como JavaScript (JS). Basados en este lenguaje, varios grupos de desarrollos coinciden que dicho lenguaje cuenta con la plataforma más sólida para ser considerado al momento de crear estándares y APIs que permitan el desarrollo de aplicaciones gráficas en los navegadores modernos [4].

WebGL es una API escrita totalmente en el lenguaje JavaScript para el despliegue interactivo de gráficos 2D/3D sobre cualquier navegador compatible, y sin el uso de plugins. WebGL permite el uso de aceleración a través de la GPU para cálculos físicos, procesamiento de imagen y efectos gráficos.

Entonces, WebGL es la solución para el despliegue de gráficos por computadora más aceptada en la actualidad por estar fundamentada en OpenGL, y por contar con la capacidad de estar presente en los navegadores de forma nativa [5]. Basados en WebGL, han surgido diversas bibliotecas de desarrolladores independientes que ofrecen una capa de abstracción para el despliegue de gráficos 2D/3D. La Figura 1 muestra ejemplos de despliegues: (a) una enciclopedia visual del cuerpo humano, (b) un editor de bloques 3D para personajes, y (c) un indicador visual de tendencias en Twitter para un hashtag.

En WebGL se emplean programas de shaders, los cuales

son pequeñas unidades de programas que son ejecutadas por el procesador gráfico para lograr acelerar el despliegue de gráficos 2D/3D. En WebGL, existen dos tipos de shaders: vertex shader y fragment shader, los cuales invocan a partes programables del pipeline de OpenGL. El vertex shader proporciona las posiciones para cada vértice que se dibujará, y el fragment shader proporciona el color de cada píxel a ser dibujado.

Estos shaders están escritos en GLSL (OpenGL Shading Language) [6]. Un shader puede ser incluido en una página Web con WebGL como texto codificado en un archivo fuente de JavaScript, como archivos independientes incluidos con el uso de la etiqueta $< script >$, ó se recupera desde el servidor como texto sin formato. El código JavaScript se ejecuta en la página y luego se envía para su compilación utilizando la API de WebGL y ejecutados en la GPU del dispositivo.

De esta forma básica, se ejecuta WebGL en los browser modernos, a continuación se presenta una serie de trabajos previos relacionados a nuestra investigación.

III. Trabajos Previos

Hasta hace poco un editor de texto era un programa que permitía exclusivamente crear y modificar archivos digitales compuestos por texto sin formato, conocidos comúnmente como archivos de texto plano. La forma de evaluar los editores de texto ha cambiado, y actualmente es posible, en pocos segundos, contar con un editor en la Web para su uso inmediato sin instalaciones o configuraciones. Según Parisi [7], partiendo de esta base, existen diversos editores que junto con WebGL permiten diseñar páginas Web con alto nivel de interactividad, construir modelos 3D, escribir código fuente en algún lenguaje de programación, e inclusive desarrollar shaders propios para WebGL. Siendo este último aspecto el aspecto principal de nuestra investigación.

En la actualidad, los editores de shaders existentes se esfuerzan por ofrecer la combinación más adecuada entre una aplicación Web y un procesador de gráficos 3D. El nivel de innovación en cada implementación varía, sin embargo, características sociales y la creación de comunidades son las más usuales.

Un ejemplo de estos editores es Shdr [8], el cual es un visor/editor y validador de shaders en línea desarrollado por Despoulain. Está soportado por WebGL, Three.js [9], Ace.js [10], RawDeflate.js [11] y jQuery [12]. Shdr combina parte de la tecnología en línea comúnmente aplicada solo a páginas o sistemas Web y la vertiente actual en herramientas gráficas en la nube. La Figura 2 muestra un ejemplo de la interfaz presentada en Shdr para el modelo de Dragón de la Universidad de Stanford [13].

Es interesante destacar que Shdr cuenta con indicadores visuales (*code highlight*), sombreado de líneas seleccionadas, *snippets* (parte reusable de código fuente), cambio de contexto entre shaders y opciones comunes de manejo de archivo como descargar, abrir y guardar el trabajo actual. Por otra parte, el despliegue es aplicado en modelos 3D descargados al equipo local desde un servidor.

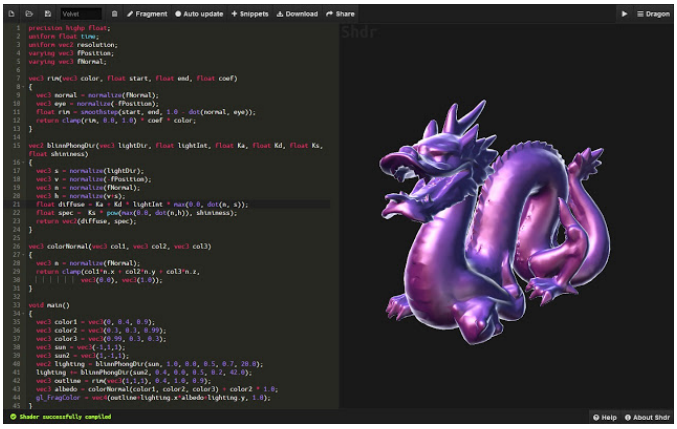


Figura 2: Interfaz de Shdr Mostrando un Modelo de Dragón [13]

Por su parte, GLSL Sandbox [14] es una aplicación creada por una comunidad de programadores para compartir y distribuir shaders. Cuenta con herramientas de edición y despliegue de efectos que lo categorizan como un editor, y posee una amplia galería de efectos y un visor a pantalla completa. Shader Lab (SL) [15] es otro editor que nace de las limitaciones propias del desarrollo nativo y la característica multiplataforma deseable entre equipos Unix y Windows. SL está significativamente limitado en cuanto a características de usabilidad. Las opciones de edición de código son prácticamente ninguna, ofreciendo 3 áreas de texto simple, sin indicadores visuales o alguna característica que potencie la experiencia de los usuarios.

Otro ejemplo notable es Kick.js [16], la cual es una biblioteca para navegadores Web modernos. Kick.js provee una API que permite crear una abstracción de WebGL y construir aplicaciones o juegos en los navegadores de manera sencilla. Basados en esta biblioteca, se desarrolló un editor de efectos 3D muy completo basado en las características fundamentales que puedan ser necesarias. Entre estas destaca una interfaz de usuarios simples sin mayor innovación en cuanto a su aspecto visual. No obstante, en cuanto a funcionalidades es el editor más completo que en la actualidad se encuentra disponible en la Web de acuerdo a nuestra investigación. Sus módulos novedosos son el uso de texturas múltiples, ajuste de variables (tipo uniform), panel de reporte de estado de efectos y diversas configuraciones del contexto 3D. Adicionalmente permite cargar y almacenar efectos, crearlos desde su inicio o usar alguno de los ejemplos incluidos en la aplicación.

Shader Toy (ST) [17] en su plataforma incluye compilación en tiempo real del código del shader, una amplia comunidad de desarrolladores, discusiones abiertas por medio de comentarios, insumos para los shader como videos, webcam, audio, texturas, hora del día y la posición del cursor, visualización en pantalla completa, editor con resaltado de sintaxis, guardar, compartir y publicar los shaders. El alcance de la herramienta es amplio, permite desarrollar efectos simples como animaciones de *sprites*, o composiciones 3D complejas tales como terrenos procedurales. El conocimiento técnico para utilizar ST es alto.

Son varias las etapas y requerimientos de software y hardware

que se necesitan para desarrollar un programa GLSL. Aunado a ello, para este proceso con tecnología Web se requiere considerar aspectos como seguridad en la red, velocidad de transferencia, estilo gráfico, manejo de sesión, redes sociales, etc. Los trabajos estudiados cuentan con un sub grupo de estas características.

Por ello, en la Tabla I se presenta una breve comparativa tomando en cuenta los aspectos más relevantes al tratarse de un editor de shader en línea. El objetivo es evaluar de manera más explícita sus características y las posibles fallas o carencias que presentan.

Tabla I: Comparación de Funcionalidades entre los Editores Estudiados. La letra **S** Representa "Sí", la letra **N** Representa "No"

Característica	SHDR	SandBox	SL	Kick	ST
Ayuda	S	N	N	S	S
Auto completado	N	N	N	N	N
Nro. de línea	S	S	N	S	S
Descargar código	S	N	N	N	N
Indicador gráfico	S	S	N	S	S
Colapso de secciones	S	N	N	S	S
Comodidad edición	5/6	3/6	1/6	5/6	3/6
Editor parámetros	N	N	N	S	N
Uso de texturas	N	N	N	S	S
Shaders (V. ó F.)	V-F	F	V-F	V-F	F
Biblioteca recursos	N	N	N	S	S
Galería de efectos	N	S	N	N	S
Manejo de sesiones	N	N	N	S	S
Compartir	N	N	N	S	S
Layout flexible	S	S	N	N	S

En la Tabla I se observan un conjunto de características comunes presentes y ausentes en los editores Shdr, GLSL Sandbox, Shader Lab, Kick.js y Shader Toy. De esta forma, en este trabajo se propone un editor, llamado Shader Tool, que contenga la mayor cantidad de características deseables, basados en el estudio comparativo mostrado.

IV. Shader Tool

Shader Tool es una solución que contempla el desarrollo de un ambiente integrado, con una serie de características deseables en un ambiente Web para la construcción de shaders GLSL. Desde el punto de vista técnico, la solución emplea el uso de la arquitectura de dos niveles cliente-servidor, REST y un sistema centralizado de datos.

Cada nivel de la solución se encuentra dividido en módulos: el lado del servidor atiende aspectos como el acceso a los datos, la representación de las entidades y la presentación y respuesta de los recursos solicitados; el lado del cliente incluye funciones para la optimización de la transferencia de datos, diseño de componentes Web, enrutador, entre otros.

La descripción de las arquitecturas de desarrollo de software, sus módulos funcionales y el modelo de comunicación implementado para el intercambio de información entre ambos niveles (cliente-servidor) se muestran a continuación.

A. Arquitectura Cliente-Servidor

Como lo describe Purewal [18], las aplicaciones Web usan la infraestructura de la Web (protocolos, lenguajes, entre otras) para su funcionamiento. Sus componentes principales son los navegadores y servidores Web, permitiéndole tener acceso a la interfaz de usuario de manera global. Entre las ventajas que ofrece esta plataforma se encuentran su alto nivel de accesibilidad, soporte multiplataforma y facilidad en el mantenimiento.

Debido a la naturaleza de la solución planteada y el interés de aplicar métodos y herramientas actuales, la taxonomía Web utilizada en el desarrollo fue la orientada a servicio en un modelo cliente-servidor. En este tipo aplicación Web ofrece servicios especializados, requiriendo una lógica de negocio acorde a cada uno de ellos.

Desde el punto de vista funcional, se puede definir el modelo cliente-servidor como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información en forma transparente aún en entornos multiplataforma. En este modelo, el cliente envía un mensaje solicitando un determinado servicio a un servidor (realiza una petición), y este envía uno o varios mensajes con la respuesta (ver Figura 3).

Figura 3: Modelo Básico Cliente-Servidor Empleado en Shader Tool

La lógica de negocio se realiza en el servidor y el cliente se ocupa de la interacción con el usuario. Esta arquitectura permite distribuir físicamente los procesos y los datos en forma más eficiente, lo que en computación distribuida afecta directamente el tráfico de la red, reduciéndolo significativamente.

B. Lado Cliente

El cliente es el proceso que permite al usuario formular los requerimientos. Este maneja todas las funciones relacionadas con la manipulación y despliegue de datos, por lo que están desarrollados sobre plataformas que permiten construir interfaces gráficas de usuario (GUI), además de acceder a los servicios distribuidos en cualquier parte de una red. Las funciones que lleva a cabo el cliente se resumen en los siguientes puntos:

- Administrar la interfaz de usuario.
- Interactuar con el usuario.
- Procesar la lógica de la solución y hacer validaciones locales.
- Generar requerimientos de bases de datos.

- Recibir resultados del servidor.
- Dar formato a resultados.

El proceso cliente fue implementado siguiendo estándares de desarrollo, y patrones de arquitecturas adaptados a una aplicación Web. El patrón de diseño MVC (Modelo-Vista-Controlador) es la base de la arquitectura en este nivel de la solución, el cual contribuyó a la separación de los componentes relacionados con los datos y componentes de la interfaz de usuario. La separación de las capas permitió tener a nivel de desarrollo un código legible, flexible y reusable.

La estructura modelada establece una comunicación intrínseca entre las tres capas dentro del patrón MVC. Así, los archivos HTML de la solución representan la vista y debe ser separada del controlador y el modelo. Igualmente, el controlador prepara los datos a utilizar, captura los eventos de la vista y realiza las acciones pertinentes sobre el modelo para modificar los datos. El controlador es importado en el archivo de vista correspondiente para que este tenga efecto. En la Figura 4 se muestra el patrón MVC empleado en Shader Tool, así como los diferentes componentes que interactúan y componen cada capa.

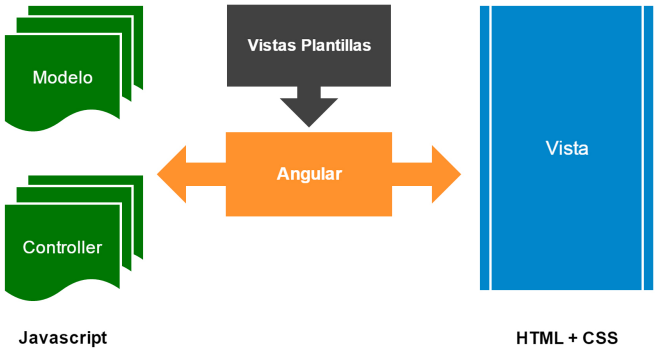


Figura 4: Patrón MVC Planteado en Shader Tool Empleando AngularJS

En nuestra propuesta, se empleó el framework AngularJS [19] para controlar las funcionalidades aplicadas en el lado cliente. Cada capa del diseño constituye una serie de componentes que proveen ciertos servicios o funcionalidades que permiten controlar aspectos como la solicitud de recursos al servidor, estado de la solución, componentes Web, entre otros.

En términos más precisos, Shader Tool fue diseñada con el esquema *single-page* en AngularJS. Este define una arquitectura formada por 8 módulos principales: aplication root, module, config, route, view, controller, directive y factories/services. La interacción entre ellos se muestra en la Figura 5.

El módulo Application representa el elemento raíz de la solución del lado cliente. Angular dispone del atributo ngApp para indicar dentro del documento HTML cual elemento se utiliza para iniciar la aplicación. El módulo Module permite incluir en el sistema componentes externos; el módulo Config establece la configuración de diversos parámetros de inicio y ejecución de la Shader Tool; y el módulo Route enlaza las URLs a los controladores y vistas.

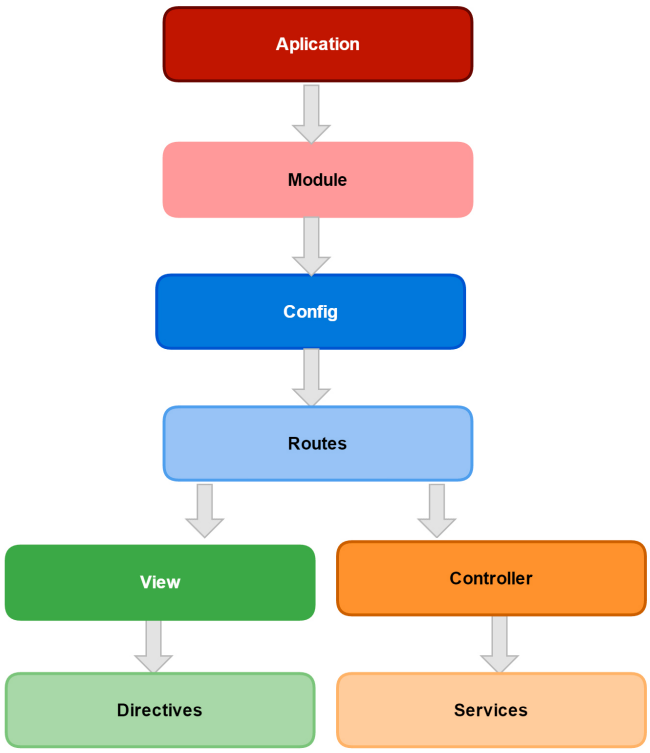


Figura 5: Módulos de la Arquitectura Provista por AngularJS

El módulo Controller emplea funciones de Javascript para aumentar el alcance del framework. El módulo View está formado por todos los archivos HTML que componen la solución. Por último, el módulo Directives permite declarar componentes Web, y el módulo Service describe una representación de un proceso u objeto.

Nuestra propuesta está compuesta por 10 vistas, y durante el desarrollo se implementaron varios componentes Web, factorías y servicios relevantes para el funcionamiento de la solución. Diversas bibliotecas y herramientas fueron empleadas con el fin de complementar nuestra solución, tales como jQuery [12], Bootstrap [20], UI-Layout [21], ngCropper [22] y uiAce [10].

Por último, la solución cliente está formada por 3 paquetes principales que hacen uso de los diversos módulos mencionados para ordenar y estructurar las funcionalidades. El paquete *Graphics* ordena los diferentes objetos involucrados en el proceso de carga, procesamiento y despliegue de un programa de shader. El paquete *Directives* contiene todos los componentes Web desarrollados. Y el paquete *Filters* define pequeños procesos que permiten transformar objetos o cadenas de caracteres para ser mostradas en la vista con un patrón en particular.

C. Lado Servidor

El lado servidor se encarga de atender a múltiples clientes que hacen peticiones de despliegue y manejo de los shaders y usuarios. Por una parte, al proceso cliente se le denomina como front-end; y al proceso servidor se le conoce con el término

back-end. El servidor maneja todas las reglas del negocio y los recursos de datos.

Adicionalmente, en la estructura del servidor, se complementa la arquitectura con la inclusión de un servidor de aplicación y base de datos. En Shader Tool se incorporó un servidor de aplicación que permite generar contenido de manera dinámica, y permite la toma de decisiones de acuerdo a parámetros de entrada incluidos en las peticiones por parte de cada cliente.

El servidor de base de datos está presente para apoyar de forma directa la persistencia de datos. Esta es requerida para almacenar y recuperar el estado de las entidades e información que pueda generarse a través de la solución. Todos los componentes descritos se resumen en la Figura 6, donde en la parte superior (color azul) se observan los componentes del servidor, y en la parte inferior un cliente conectado desde una portátil (lado izquierdo - color verde) y desde un dispositivo móvil (lado derecho - color rojo).

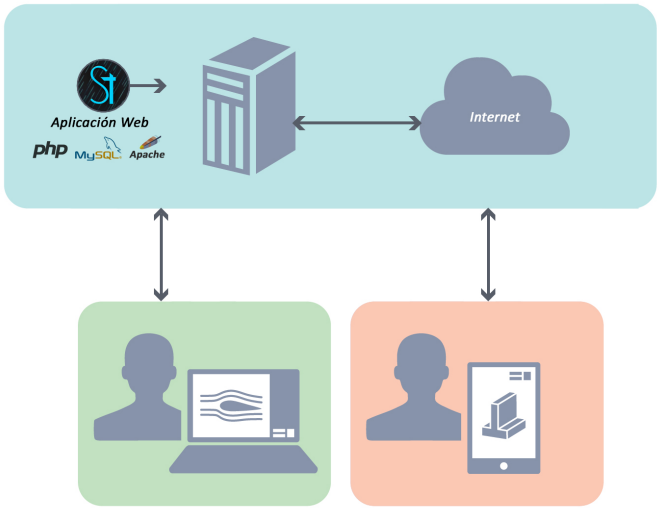


Figura 6: Arquitectura de Comunicación y Software del Lado Servidor

Para concebir la solución de forma abstracta y adaptable, el servidor de aplicación es un sistema distribuido dividido en capas (i.e. patrón de arquitectura multicapa) para separar los componentes de acuerdo a su función. Luego de un proceso de análisis se establecieron las siguientes capas funcionales: persistencia de datos, acceso a datos, entidades, y lógica de negocio.

A nivel técnico el trabajo presenta otros aspectos relevantes que forman parte del núcleo principal de la aplicación, sin embargo, estos son detalles a nivel de implementación del lado del cliente y servidor que son de mayor interés en el área de la programación. Aun así, es importante destacar las características asociadas al resultado global obtenido enmarcado en un ambiente Web de desarrollo integrado para GLSL.

V. Interfaz de Shader Tool

La solución interactiva creada, permite al usuario manipular el sistema de forma gráfica haciendo uso de la entrada estándar de teclado y ratón. Esta puede ser definida como un sistema integrado de desarrollo que permite implementar programas de shaders para el lenguaje GLSL, organizada en secciones y componentes.

La interfaz gráfica está dividida en cinco grandes secciones. Cada una de estas secciones está agrupada en sub-elementos según sus características, funcionalidades y similitudes. Por ejemplo, todos los comandos para la programación de los programas de shaders se encuentran agrupados en la parte izquierda de la barra de menú. Como puede verse en la Figura 7, las cinco secciones principales son las siguientes:

1) Barra de menús: Corresponde a la fuente de comandos más común. En la barra de menús se encuentran todos los métodos y funcionalidades de la solución. Está dividida en 3 partes bien marcadas: opciones de editor, opciones de despliegue y perfil de usuario.
2) Panel de Editor: Define el área en la cual se escribe el código de los programas de shaders. Cada programa (de vértice y fragmento) disponen de este espacio, sin embargo, solo uno podrá estar activo en un momento dado.
3) Panel de vista: Permite percibir el contenido de la escena, en la cual se aplica el resultado tras la compilación de ambos programas de shaders. Junto al panel de editor, puede ser ajustado su tamaño a gusto del usuario.
4) Área de Resultado: Muestra el resultado obtenido tras la compilación del shader. Está ubicada en la parte inferior del panel de vista, y agrupa en dos pestañas los mensajes para cada programa.
5) Área de Notificación: Despliega las notificaciones generadas por procesos iniciados por el usuario. Está ubicada en la parte superior derecha del panel de vista.

A. *Panel de Editor*

El panel del editor permite a los usuarios escribir el código que componen su shader. El editor dispone de tres elementos de interfaz mostrados en la Figura 8, que ayudan a la interacción: Número de línea, colapso de reglones de código y menú contextual de edición.

Es posible colapsar/abrir una región de código presionando en un indicador en forma de flecha (ver Figura 8). Posee un menú contextual que permite opciones como deshacer y rehacer el estado del editor, seleccionar todo el código y los comandos estándar copiar, pegar y cortar.

B. *Panel de Vista*

Es la ventana donde se despliega la escena vista desde cierta perspectiva. Esta ventana posee numerosas opciones que le permiten al usuario diferentes configuraciones al momento de visualizar la escena. La primera opción de configuración es proyectar la escena con una vista perspectiva u ortogonal.

La Figura 9 muestra un ejemplo de proyección perspectiva dentro del panel vista de un modelo de auto, junto con el menú contextual. Otro elemento de configuración del panel son los atributos de despliegue del objeto en la escena. Entre los parámetros que pueden ser adaptados están: despliegue

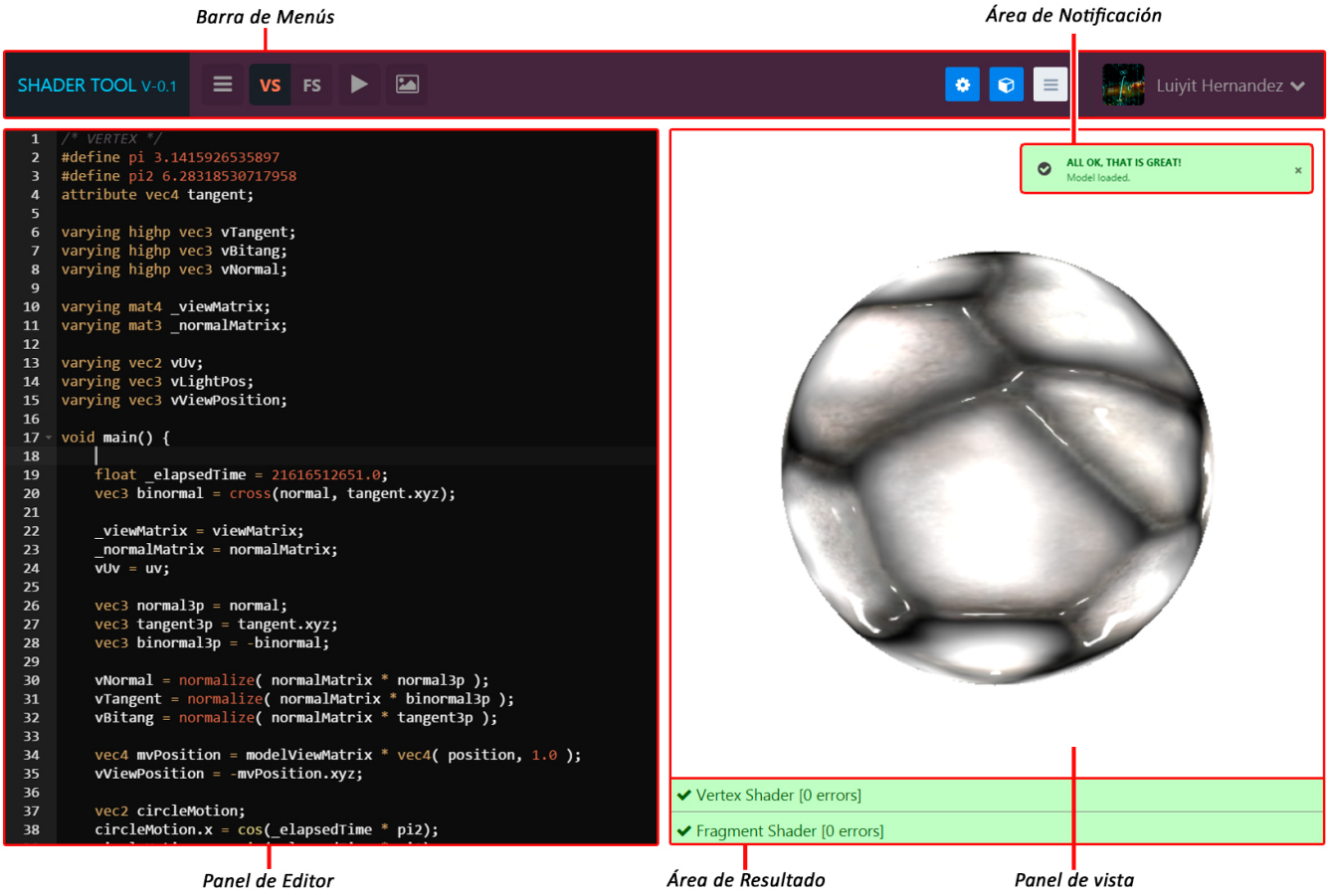


Figura 7: Diferentes Secciones de la Página Editor de Shader Tool (Lado Cliente)

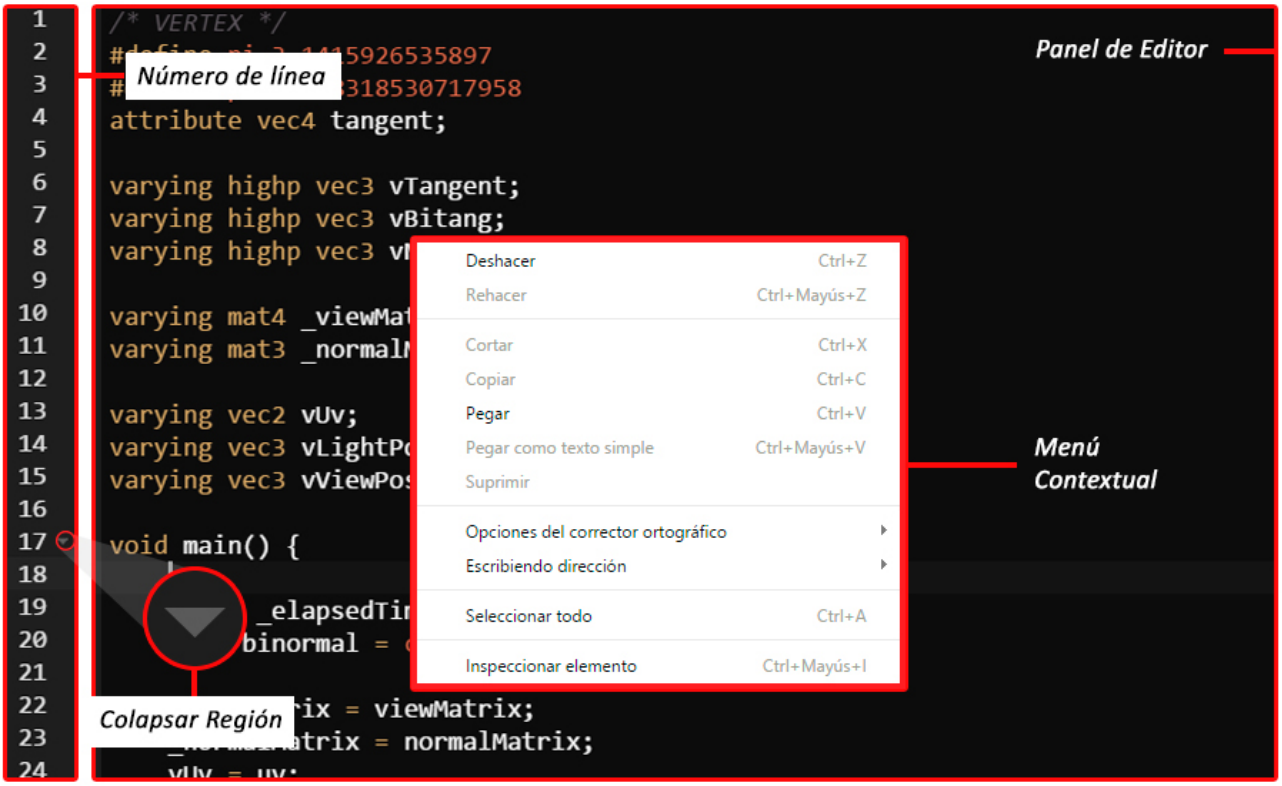


Figura 8: Elementos que Conforman el Panel de Editor

de las caras delanteras/traseras, activar/desactivar el zBuffer, deplegar el objeto como malla/puntos/relleno.

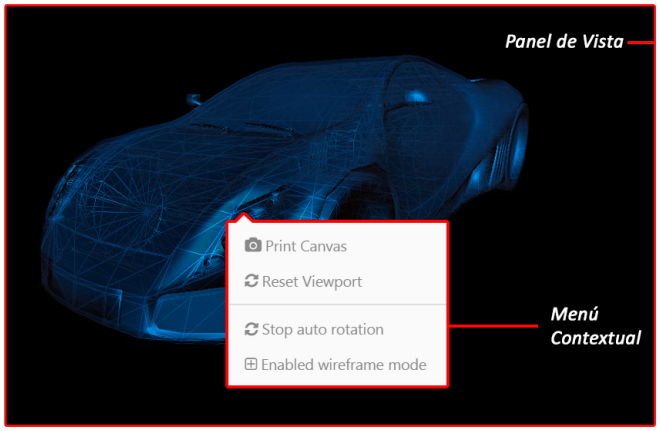


Figura 9: Elementos que Conforman el Panel de Vista

Igualmente, la cámara también puede ser adaptada y ajustar valores del frustum. Es importante destacar que estas opciones de configuración son accesibles desde la barra de menú principal, y a través del menú contextual que se muestra con el click derecho en cualquier área del panel.

C. Modales Funcionales

La solución Shader Tool cuenta con un gran número de opciones y herramientas de configuración asociados al proceso de programación de los shaders. Teniendo en cuenta que el espacio más relevante de la ventana del browser fue apropiadamente dispuesto para los dos paneles principales, se empleó un sistema de modales para agrupar funcionalidades particulares. De esta manera las diferentes funciones de la solución son accedidas a demanda.

Cada modal se superpone a los paneles principales. Esto permite ordenar la información y controles en un espacio independiente sin alterar la disposición de la interfaz del usuario. Las funciones más relevantes que usan el sistema de modal son: crear Shader, cargar Shader, biblioteca de texturas, biblioteca de modelos y perfil de usuario.

Por ejemplo para cargar un shader a Shader Tool, se dispone de un modal que permite seleccionar cuál shader se cargará. Para ello, se presenta una lista de shaders la cual es dividida en páginas formando grupos de cinco, y el shader actual es mostrado con un icono en forma de *v* en la columna de opciones de la tabla. Para seleccionar un shader, se debe hacer clic sobre el botón de acción en color azul al final de cada fila. En la Figura 10 se muestran todos los elementos involucrados en dicho modal.

Dada la dinámica de la solución, existen diversos modales que resultan apropiados en sistemas con resoluciones de menor tamaño o donde la selección de opciones requiera una mejor precisión (i.e. tablets y teléfonos inteligentes).

Un ejemplo de ellos es el modal para añadir un nuevo modelo a la biblioteca de modelos de un usuario. Para ello, se cuenta con un área donde se puede seleccionar el modelo arrastrándolo y soltándolo en una región dispuesta para ello, o seleccionándolo desde el selector nativo del browser. En ambos casos, el archivo se validará y procesará.
Una vez seleccionado el modelo se creará de forma automática una escena que desplegará el modelo. Para garantizar el correcto despliegue, el modelo es normalizado y centrado antes de ser mostrado. Mientras el área de pre-visualización está activa, se activan dos opciones adicionales: descartar modelo, y detalles del modelo. La escena y tabla de detalles del modelo es mostrado en la Figura 11 donde se observa los diálogos modales para la carga de un modelo, y su descripción.

VI. Pruebas y Resultados

En esta sección se presentan una serie de pruebas para validar nuestra propuesta. Así, se muestran pruebas cuantitativas y cualitativas obtenidas por el uso de la solución. Dado que es una solución Web, resulta interesante medir la velocidad de descarga de los diversos recursos, la eficiencia del proceso de compilación y ejecución de los shaders y la experiencia de los usuarios en términos de utilidad, facilidad de uso y cumplimiento de los objetivos de la investigación.

A. Detalles de Implementación

El ambiente Web desarrollado requirió de numerosas de bibliotecas y clases (RESTful, Scene, Chameleon, entre otras) en los lenguajes PHP v 5.4 y Javascript, las cuales fueron divididas en dos niveles (cliente-servidor); los cuales contienen los métodos y propiedades necesarios para el funcionamiento de Shader Tool.

La biblioteca creada del lado servidor se encarga de procesar las peticiones recibidas, y prepara el entorno para atenderla y dar respuesta, entre otras funciones. Del lado del cliente las clases desarrolladas se dividen en 3 grupos principales:

1) Gestión de las peticiones y respuestas obtenidas.
2) Gestión de las tareas involucradas en el proceso de despliegue de la escena 3D.
3) Gestión de componentes Web para encapsular funcionalidades y gestión de manejo de eventos en la interfaz de usuario.

B. Experimentación Cuantitativa

Estos experimentos tienen como objetivo medir el tiempo y recursos necesarios para la carga de los diversos recursos

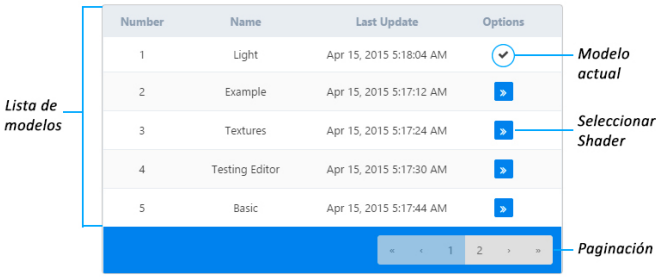


Figura 10: Lista de Shaders Asociados a un Usuario

Figura 11: Diálogo Modal para la Carga y Descripción de los Modelos

gestionados en la solución desarrollada en este trabajo. Entre las pruebas se incluyen: velocidad de carga de los modelos en varias configuraciones de red, velocidad de procesamiento de los modelos y tiempo de carga y compilación de los shaders usando 4 técnicas de sombreado.

1) Procesamiento de Modelos: Para realizar las pruebas de velocidad de carga y procesamiento se desarrolló un pequeño paquete *benchmark* que se integra a la solución y reproduce los procesos involucrados en las pruebas. Las clases implementadas registran el tiempo de respuesta del servidor y el tiempo requerido para procesar dicho recurso luego de tenerlo disponible para su uso. El proceso completo involucrado en las pruebas incluye las siguientes etapas:

- Solicitar lista de recursos disponibles (identificadores) del lado servidor para realizar las pruebas. Esta etapa es previa al registro del tiempo.
- Procesar cada identificador de manera individual. Se solicita el recurso completo al servidor.
- Al recibir el modelo, se valida la respuesta y se inicia el procesamiento de los datos.
- Se transforma la cadena de caracteres que contiene información de geometría a una estructura en formato manipulable y estructurado.
- Se procesa la estructura y se crea un objeto propio de la solución manipulable por el framework de despliegue.
- Con la información de geometría se crea un objeto modelo, el cual posee propiedades adicionales necesarias para el despliegue e interacción con el resto de los elementos de la solución.

El proceso fue aplicado a una muestra de 10 modelos con distinciones claras en cuanto al número de vértices y triángulos involucrados. Las redes simuladas en las pruebas fueron 2G (450 kbps), 3G (1 Mbps) y 4D (4 Mbps). En la Tabla II se muestran los resultados obtenidos del proceso completo (*TTFB* significa tiempo de espera del primer byte).

Los resultados obtenidos indican un aumento en el tiempo requerido para descargar y procesar los modelos a medida que la velocidad de la red utilizada es menor. Es claro que la velocidad de descarga recomendada se encuentra entre 1 y 4 MB. De igual forma se puede evidenciar que el tiempo de procesamiento de cada modelo no es un factor crítico en el tiempo total, teniendo en cuenta que el promedio obtenido en una red con una velocidad de transferencia alta es de apenas 0.53 segundos.

Tabla II: Tiempos de Carga de los Modelos y su Valor Promedio por cada Red (en Color Azul)

Modelo	Vértices	Triá ng	2G	3G	4G	TTFB
Tetrahedron	4	4	0.45 s	0.35 s	0.32 s	0.34 s
Cube	8	12	0.65 s	0.36 s	0.36 s	0.31 s
Cylinder	147	228	3.15 s	1.33 s	0.89 s	0.68 s
Plane	256	450	3.80 s	1.81 s	0.88 s	0.36 s
Ring	297	512	4.38 s	2.54 s	0.94 s	0.42 s
Torus	336	600	7.23 s	2.97 s	1.46 s	0.40 s
TorusKnot	768	1536	16.76 s	8.02 s	3.48 s	0.47 s
Sphere	1296	2380	27.31 s	10.21 s	4.85 s	0.63 s
Text	2292	4572	28.09 s	11.88 s	4.72 s	0.53 s
			9.18 s	**3.92 s**	**1.79 s**	**0.46 s**

Es importante destacar que las pruebas se realizaron bajo un entorno red estándar, y se promediaron los resultados obtenidos en 20 simulaciones para cada tipo de red estudiado. Adicionalmente hay que tener en cuenta que en los tiempos indicados no se están considerando los milisegundos correspondientes a la fase búsqueda de DNS (*DNS Lookup*) y tiempo de bloqueo (*Proxy negotiation* o *Establishing Connection*).

Para complementar el análisis de los resultados, se estudió el tiempo de respuesta que ofrece la solución una vez que el recurso ha sido descargado del servidor. En la Tabla III se muestra el tiempo de procesamiento de los datos recibidos para cada modelo analizado, así como la cantidad de memoria RAM que requiere cada objeto. Las fases que conforman el procesamiento del modelo son:

- Validar la información recibida (integridad del objeto).
- Crear la instancia del objeto de geometría a partir de las cadenas de caracteres recibidas.
- Crear objeto 3D a partir de la geometría resultante.

Al crear la instancia del objeto, queda disponible para su uso en cualquier otro proceso. Teniendo en cuenta que son varias las rutas que pudiera tener el modelo (despliegue

Tabla IV: Resultados Obtenidos en 3 Configuraciones de Hardware para 5 Shaders Distintos

Shader	# líneas	¿Textura?	GeForce GTX 750	AMD Radeon HD 8550G	ARM Mali-400
Basic	9	N	0.0043 s	0.0055 s	0.0177 s
Sample Texture	15	S	0.0047 s	0.0183 s	0.0195 s
Displacement	35	S	0.0085 s	0.0094 s	0.0257 s
Per Fragment Lighting	45	S	0.0077 s	0.0122 s	0.0317 s
Normal Mapping	153	S	0.0153 s	0.0278 s	0.0613 s
Frames por segundo mínimo / máximo			**60 fps / 60 fps**	**57 fps / 60 fps**	**25 fps / 60 fps**

Tabla III: Cantidad de Memoria y Tiempo de Procesamiento de los Modelos

Modelo	Memoria	Tiempo de Procesamiento
Tetrahedron	27 kB	0.42 s
Cube	78 kB	0.45 s
Dodecahedron	231 kB	0.49 s
Cylinder	1.4 MB	0.45 s
Plane	2.8 MB	0.46 s
Ring	3.3 MB	0.48 s
Torus	3.8 MB	0.74 s
TorusKnot	9.8 MB	0.55 s
Sphere	15.3 MB	0.61 s
Text	29.1 MB	0.65 s

principal, pre-visualización, asociación, listar datos básicos para su selección, entre otros), la prueba de procesamiento se evalúa hasta este punto, considerando única y exclusivamente el tiempo requerido para transformar los datos recibidos desde el servidor, en un objeto Javascript válido.

Las pruebas realizadas evidencian que la velocidad de la red utilizada para acceder a la solución tendrá un alto impacto en la velocidad de respuesta. EL mayor porcentaje de tiempo en obtener un recurso corresponde al proceso de descarga, teniendo en cuenta que en promedio el valor TTFB es de 0.5 segundos. Durante este tiempo de espera por parte del cliente, se llevan a cabo entre otros procesos, la recepción de la petición, su procesamiento y respuesta. Manteniendo un tráfico bajo, la solución del lado servidor es capaz de atender las peticiones en fracciones de segundo.

Por otra parte, el procesamiento de los modelos luego de recibir el 100% de los datos oscila entre 0.42 segundos y 0.74 segundos, lo cual indica que en conjunto (cliente y servidor) la solución requiere en promedio de 1 segundo para procesar cualquier modelo que estén ajustado a las muestras utilizadas.

2) Procesamiento de Shaders: El proceso encargado de gestionar y procesar el código de los shaders es uno de los principales dentro de Shader Tool. Las pruebas asociadas a los programas GLSL se dividen en dos partes. La primera parte registra el tiempo de procesamiento del shader, desde el momento en que se recibe toda la información desde el servidor, hasta crear las estructuras necesarias y compilar el código. Los resultados varían de acuerdo a las capacidades de procesamiento, tanto del procesador como de la tarjeta gráfica del equipo que ejecuta la solución del lado del cliente.

Para validar el comportamiento de la solución en varios escenarios, las pruebas se llevaron a cabo en tres configuraciones de hardware distintas: GeForce GTX 750, AMD Radeon HD 8550G y ARM Mali-400. En la Tabla IV se muestran los valores obtenidos, donde los shaders analizados son:

- Basic: shader con operaciones básicas. Se procesa de forma simple la posición de cada vértice y se asigna un color estático a cada fragmento.
- Sample Texture: incluye el manejo de un mapa de bits en el programa de fragmento. Para definir el color de cada fragmento se hace uso de las coordenadas de texturas y se extrae la muestra directamente de la textura.
- Displacement: modifica dinámicamente los vértices y colores. Se configuran variables uniformes y atributos a cada vértice para animar la geometría, implicando enviar en cada cuadro los datos actualizados a la tarjeta gráfica.
- Per Fragment Lighting: técnica de iluminación en la cual la iluminación se basa en cada fragmento de la imagen final. Se incluyen componentes de luz difuso y especular, así como también el uso de los vectores normales para cálculos de incidencia de luz sobre cada fragmento.
- Normal Mapping: ilumina y simula relieve en superficies planas empleando dos mapas de textura.

Los resultados obtenidos al procesar cada shader muestran que tanto en computadores personales como en dispositivos móviles el tiempo requerido por la solución está por debajo de $\frac{1}{10}$ s. Teniendo en cuenta la velocidad de carga y procesamiento de los modelos, esta cifra se encuentra en un rango aceptable, ya que en el peor de los casos el shader será procesado en el mismo tiempo que los modelos. Las pruebas realizadas muestran una relación entre el número de líneas y tiempo requerido para ser procesadas por número de operaciones aritméticas involucradas en el shader, lo cual indica que las optimizaciones realizadas al código de cada programa es un factor importante a considerar.

Otro valor importante asociado a los shaders es la velocidad con la que el despliegue se realiza. Las imágenes por segundos (*frame per seconds* - fps) es la medida de la frecuencia a la cual un reproductor de imágenes reproduce distintos fotogramas (*frames*). Estos fotogramas están constituidos por un número determinado de píxeles que se distribuyen a lo largo de una red de texturas. La frecuencia de los fotogramas es proporcional al número de píxeles que deben generarse, incidiendo en el rendimiento del ordenador que los reproduce. En la parte inferior de la Tabla IV se muestran los valores mínimos y máximos de despliegue para cada shaders analizado.

3) Velocidad de Carga: Inicialmente la ubicación del servidor de pruebas no fue un aspecto a considerar. Sin embargo, al momento de realizar las pruebas resultó beneficioso una distancia física considerable entre el servidor y el equipo cliente, con el fin de obtener resultados en la velocidad

Tabla V: Tiempos Obtenidos en Diversos Sistemas Operativos y Browsers para el Despliegue

Dispositivo - Sistema Operativo	Browser	1er despliegue	2do despliegue	TTFB
Desktop - Microsoft Windows 8	Chrome 40	32.451 s	2.505 s	1.079 s
Desktop - Microsoft Windows 8	Firefox	35.880 s	1.707 s	0.681 s
Desktop - Microsoft Windows 8	IE11	28.965 s	0.471 s	1.077 s
Motorola Moto G - Android 4.3	Chrome	34.385 s	4.703 s	1.452 s
Nexus 5 - Android 4.1	Chrome	33.962 s	4.197 s	1.477 s
Nexus 7 Landscape - Android 5.0	Chrome	33.865 s	3.849 s	1.201 s
iPhone 4.0 - iOS 5.1	Safari	23.392 s	4.835 s	0.001 s
	Tiempo promedio	**27.207 s**	**2.823 s**	**0.736 s**

más fiables. Así, se realizaron las conexiones de pruebas desde Caracas, Venezuela hacia un servidor en Virginia, USA. Actualmente, Shader Tool se encuentra alojado en la dirección http://ccg.ciens.ucv.ve/~esmitt/projects/shader_tool.

El ambiente integrado está formado por un conjunto de archivos en formato Javascript, CSS, HTML, entre otros. Por su naturaleza, todo el contenido y archivos necesarios para el funcionamiento de la solución deben ser descargados de forma íntegra. De igual manera, los diversos módulos deben ser iniciados y enlazados.

Para estimar el rendimiento de la solución en cuanto a tiempo de inicio, se realizaron simulaciones de descarga combinando diversas plataformas de sistemas operativos actuales y browser comerciales. En la Tabla V se muestra el tiempo requerido en el cual la aplicación se inicia y se encuentra disponible para su uso.

En la ejecución de las pruebas se empleó una conexión de rendimiento promedio (Móvil 3G - 780 kbps/330 kbps) y las peticiones se realizaron, como se mencionó anteriormente, desde un equipo ubicado en Dallas, VA, USA. El objetivo de realizar las pruebas desde un equipo que esté significativamente distante del servidor que aloja la solución, es generar los resultados bajo condiciones que representen una exigencia mayor en cuanto al nivel de optimización de la solución.

Por otra parte, al evaluar los resultados de inicio se observa una mejora en el despliegue de la aplicación en accesos posteriores a la carga inicial. En todos los casos de estudio se obtiene una mejora superior al 80 %. También se muestra que el tiempo necesario se mantiene estable en los diversos dispositivos y sistemas operativos.

Para ilustrar la composición final de la solución y la división de los recursos descargados en las pruebas se muestra en la Figura 12, donde se muestra el resumen porcentual de cada tipo de archivo descargado y su tamaño medido en kilobytes.

Se observa que la solución está conformada principalmente por archivos Javascript (js en la Figura 12). Aunque la solución contempla el uso de aproximadamente 15 documentos HTML, éstos son gestionados usando un servicio de optimización llamado *TemplateCache*. El funcionamiento del servicio se basa en almacenar en memoria una función Javascript que da acceso al contenido de cada vista. Cada archivo HTML ahora puede ser accedido mediante una palabra clave.

Basado en ese concepto, cada archivo HTML es representado

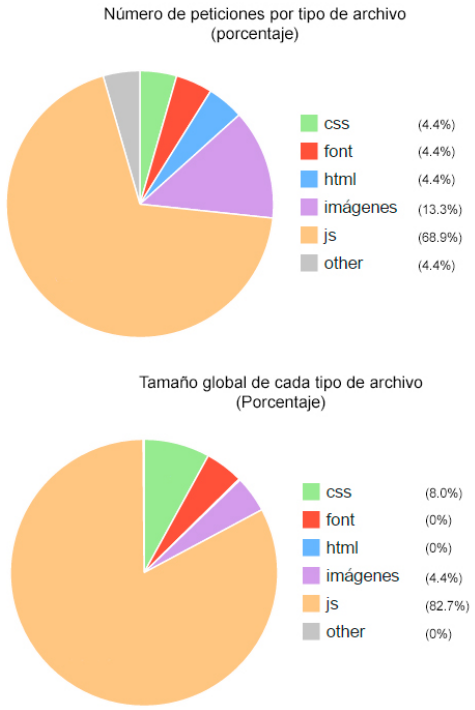


Figura 12: Tipos de Archivo en Número de Peticiones y Tamaño

por un archivo JS el cual es incluido como parte de la aplicación. Este procedimiento incrementa significativamente el tiempo de respuesta de la aplicación una vez que el usuario accede a los diferentes módulos, ya que todas las vistas fueron descargadas en la inicialización de la aplicación.

C. Experimentación Cualitativa

Las pruebas cualitativas se basaron en recopilar las distintas experiencias de los usuarios al utilizar la solución en términos de facilidad de uso, utilidad y cumplimiento de los objetivos de la investigación. El instrumento diagnóstico utilizado fue una encuesta en torno a 8 preguntas contenidas en 4 tópicos principales:

1) Utilidad de la interfaz gráfica de usuario.
2) Facilidad en la creación de shaders.
3) Evaluación cualitativa de los recursos disponibles.
4) Utilidad de la solución en términos de tiempo, esfuerzo y recursos utilizados.

La síntesis de los resultados obtenidos en la encuentra agrupados por tópicos se puede visualizar en la Figura 13. El rango de valoración fue establecido basado en una escala de Likert de 5 niveles; donde 5 representa un desempeño óptimo dentro de las características del tópico y 1 representa deficiencias en uno o varios aspectos asociadas.

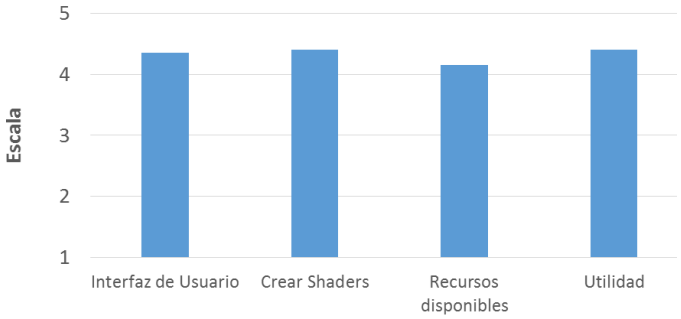


Figura 13: Valoración Promedio de cada Tópico Evaluado de la Encuesta

La encuesta fue realizada a 20 estudiantes de la Escuela de Computación de la Universidad Central de Venezuela. Los encuestados forman parte de las asignaturas impartidas en el área de Computación Gráfica debido a que requieren conocimientos técnicos involucrados en el desarrollo de shaders en el lenguaje GLSL. La mejor y peor valoración (en promedio de los 4 rubros) fue retirada de la encuesta.

Los resultados indican que todas las herramientas interactivas proporcionadas por la interfaz gráfica de usuario facilitan en gran medida el uso de la solución, ya que las mismas permiten ajustar el área del editor y viewport a voluntad, acceso a las diferentes opciones a través de múltiples interfaces y la posibilidad de ejecutar comandos directamente desde atajos de teclado. Igualmente, se pudo validar que el proceso de desarrollo de shaders es considerado práctico y rápido por los usuarios. En todo momento se conoce claramente sobre cual programa GLSL se está trabajando, la solución permite rápidamente cambiar entre los programas y los resultados del Shaders son mostrados rápidamente en el panel de despliegue.

Por otro lado, Shader Tool permite a los usuarios usar una biblioteca de modelos, texturas y shaders estándar da un valor agregado a la herramienta. De esta forma, los pueden usuarios crear shaders y concentrarse exclusivamente en la programación y no en los recursos necesarios. Por último, la velocidad de carga de la solución es el aspecto más cuestionado. Los usuarios no coinciden en una calificación uniforme en este rubro. Una parte de los encuestados indican que el desempeño es adecuado y otro grupo que es bajo o ineficiente. Sin embargo, hay que considerar que la calificación obtenida está asociada a la velocidad de conexión que posee el usuario.

VII. Conclusiones y Recomendaciones

En este trabajo se presenta un entorno Web integrado que permite desarrollar programas de sombreado para OpenGL, haciendo uso de frameworks avanzados de despliegue y herramientas de marcado y estilo. Durante nuestra investigación se analizaron todas las variables y condiciones que intervienen durante el desarrollo de programas GLSL en un ambiente de Web. Con dicha información se estudiaron las limitaciones de los editores actuales disponibles en la actualidad y se encontró que se debía desarrollar un sistema integrado que con tecnología Web de vanguardia, y con componentes adecuados de una interfaz gráfica de usuario multiplataforma.

El análisis de la interacción del usuario con la solución desarrollada y los diversos principios de diseños como la jerarquía visual, legibilidad e integridad, fueron aplicados para reducir la curva de aprendizaje de la solución y de todas sus funcionalidades. De acuerdo a los resultados obtenidos, se encontró que el conjunto de componentes e interfaces gráficas que se diseñaron e implementaron permiten al usuario interactuar con el sistema de manera sencilla, facilita el desarrollo de los shaders y su asociación con elementos como texturas, modelos y variables de control.

La solución permite cargar y procesar los recursos principales empleados en la fase de despliegue del shader. Los indicadores de carga, controles de páginas y carga fragmentada de recurso en los modales que despliegan la lista de modelos 3D y texturas, fueron introducidos como mejoras para ayudar a la velocidad de carga e interacción con los usuarios. Shader Tool ofrece a los usuarios una biblioteca pública que puede ser utilizada en sus shaders. Igualmente, cada programador puede crear y gestionar sus modelos de manera independiente.

Los resultados de la encuesta indican que existen factores que presentan la solución como una posible herramienta de entrenamiento y formación interactiva para la comunidad de estudiantes y programadores del área de computación gráfica.

A futuro, el sistema puede ser adaptado para permitir la incorporación de un mayor número de parámetros para el despliegue; como incluir la posibilidad de manipular parámetros de transparencia o blending en los materiales asociados a los modelos, uso de una fuente de luz configurable y posibilidad de adaptar las funciones aplicadas a la prueba de profundidad.

Nuestra solución permite incluir variables de tipo uniforme a sus shaders, sin embargo, no existe ninguna herramienta que permita registrar variables personalizadas de tipo atributos o uniformes. Una solución sería incorporar un nuevo modal a la interfaz de usuario que permita gestionar (crear, modificar y eliminar) las variables asociadas a un shader en particular. Es importante mencionar que los requerimientos necesarios para la implementación de esta funcionalidad del lado servidor fueron previstos, y actualmente el esquema de base de datos y la API REST están preparados para soportarla.

La investigación realizada podría ser la base para el desarrollo de un sistema más complejo que permita el implementar shaders en un contexto 2D o post procesamiento. Dicho sistema podría permitir la elección por parte del usuario del tipo de shader que se desea codificar y configurar el ambiente de desarrollo en base a la elección. Esto implicaría el desarrollo de módulos adicionales como el manejo de sprites y la manipulación de buffers de despliegues propios de OpenGL.

Teniendo en cuenta los resultados obtenidos en las pruebas de rendimiento aplicadas a los shaders, resulta viable realizar tareas de procesamiento gráfico más exigentes. Por lo cual, resultaría interesante realizar adaptaciones en la solución para despliegues de volúmenes 3D a través de técnicas como volume rendering.

REFERENCIAS

[1] Khronos Group, *OpenGL ES 2.0 for the Web*, https://www.khronos.org/webgl.

[2] World Wide Web Consortium (W3C), *HTML Canvas 2D Context*, http://www.w3.org/TR/2dcontext [W3C Candidate Recommendation del 2 de Julio 2015].

[3] World Wide Web Consortium (W3C), *Scalable Vector Graphics (SVG)*, http://www.w3.org/TR/SVG [W3C Recommendation del 16 de Agosto 2011].

[4] M. Myers, *A Smarter Way to Learn JavaScript: The New Approach That Uses Technology to Cut Your Effort in Half*, 1ra edición, CreateSpace Independent Publishing Platform, 2014.

[5] K. Matsuda y R. Lea, *WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL*, 1ra edición, Addison-Wesley Professional, 2013.

[6] R. Rost, B. Licea-Kane, D. Ginsburg, J. Kessenich, B. Lichtenbelt, H. Malan y M. and Weiblen, *OpenGL Shading Language*, 3ra edición, Addison-Wesley Professional, 2007.

[7] T. Parisi, *Programming 3D Applications with HTML5 and WebGL: 3D Animation and Visualization for Web Pages*, 1ra edición, O'Reilly Media, 2014.

[8] T. Despoulain, *Shdr*, http://shdr.bkcore.com.

[9] Community Project, *JavaScript 3D library - Three.js*, http://threejs.org [Código Fuente - https://github.com/mrdoob/three.js].

[10] Community Project, *Ace - The High Performance Code Editor for the Web*, http://ace.c9.io [Código Fuente - https://github.com/ajaxorg/ace].

[11] I. Yuta, *Zlib.js*, https://github.com/imaya/zlib.js.

[12] The jQuery Foundation, *jQuery*, https://jquery.com.

[13] Stanford University Computer Graphics Laboratory, *The Stanford 3D Scanning Repository*, http://graphics.stanford.edu/data/3Dscanrep [Modelo - Dragon].

[14] R. Cabello, *GLSL Sandbox*, http://mrdoob.com/ projects/glsl_sandbox [Código Fuente - https://github.com/mrdoob/glsl-sandbox].

[15] B. Bass, *WebGL Shader Lab*, http://goo.gl/iFsvb.

[16] Community Project, *Kick.js*, http://goo.gl/01sxRs [Código Fuente - https://github.com/mortennobel/KickJS].

[17] I. Quilez y P. Jeremias, *Shadertoy*, https://www.shadertoy.com.

[18] S. Purewal, *Learning Web App Development*, 1ra edición, O'Reilly Media.

[19] Google Inc., *AngularJS*, https://github.com/koorgoo/ngCropper.

[20] Community Project, *UI Layout*, http://layout.jquery-dev.com [Código Fuente - https://github.com/twbs/bootstrap].

[21] F. Balliano y K. Dalman, *Bootstrap*, http://getbootstrap.com [Código Fuente - https://github.com/twbs/bootstrap].

[22] Community Project, *AngularJS Wrapper of jQuery Image Cropping Plugin*, https://github.com/koorgoo/ngCropper.