UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Desarrollo de componentes HTML5 y WebGL para la ilustración de conocimientos en la docencia de gráficos

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Especialidad de Computación

Autor:
Miguel Ángel Malqui Cruz

Director:
Marta Fairen Gonzalez
Departamento de ciencias de la computación

Codirector:
Alvar Vinacua Pla
Departamento de ciencias de la computación

25 de abril del 2022

Resumen

En el campo de la computación gráfica se presentan varios conceptos que pueden resultar difíciles de asimilar por parte del alumnado. Algunos estudiantes no consiguen entender las ideas expuestas en las clases, y luego se encuentran perdidos en el momento de aplicar su conocimiento en la programación de gráficos.

Debido a que este campo es, de manera intrínseca, muy visual, la inclusión de programas interactivos podría facilitar la captación de conocimiento por parte de los estudiantes. Estos programas les permitirían a los alumnos representar diferentes problemas y experimentar con distintas configuraciones.

Este proyecto tiene como objetivo el ayudar a la docencia de las asignaturas de gráficos de la FIB. Por este motivo, se crean una serie de programas, que muestren a manera de ejemplos interactivos algunos de los conceptos que resultan ser más difíciles en la programación de gráficos por ordenador. Estos programas se desarrollan como componentes web independientes, que se puedan cargar en una página web y/o añadir a una que ya tenga otros elementos.

Resum

Al camp de la computació gràfica es presenten diversos conceptes que poden resultar difícils d'assimilar per part de l'alumnat. Alguns estudiants no aconsegueixen entendre les idees exposades a les classes, i després es troben perduts a l'hora d'aplicar el seu coneixement a la programació de gràfics.

Com que aquest camp és, de manera intrínseca, molt visual, la inclusió de programes interactius podria facilitar la captació de coneixement per part dels estudiants. Aquests programes permetrien als alumnes representar diferents problemes i experimentar amb diferents configuracions.

Aquest projecte té com a objectiu ajudar a la docència de les assignatures de gràfics de la FIB. Per aquest motiu, es creen una sèrie de programes, que mostrin a manera d'exemples interactius alguns dels conceptes que resulten més difícils en la programació de gràfics per ordinador. Aquests programes es desenvolupen com a components web independents, que es puguin carregar en una pàgina web i/o afegir-ne una que ja tingui altres elements.

Abstract

In the field of computer graphics there ae several concepts that can be difficult to assimilate for students. Some of them fail to understand the ideas presented in lectures. Then they are at a loss when it comes to applying their knowledge to graphics programming.

Since this field is intrinsically visual, the inclusion of interactive programs could make easier the acquisition of knowledge for students. These programs would allow students to represent different problems and experiment with different configurations.

This project aims to help the FIB in teaching of computer graphics. For this reason, a series of interactive programs are created to show some of the most difficult concepts in computer graphics programming. These programs are developed as independent web components that can be loaded into a webpage and/or added to another webpage that already has other elements.

Agradecimientos

Me gustaría dar las gracias a los directores del TFG, Marta Fairen y Alvar Vinacua, por asesorarme en el desarrollo del proyecto y en el diseño de la estructura y contenido de la memoria. La continua comunicación y comentarios recibidos me han facilitado la realización de este trabajo y me han permitido mejorar en la redacción de la documentación.

Índice

Lista de figuras

Lista de tablas

1 Contexto
1.1 Introducción

Este proyecto, *Desarrollo de componentes HTML5 y WebGL para la ilustración de conocimientos en la docencia de Gráficos*, es un trabajo de final de grado para la Facultad de Informática de Barcelona (FIB), concretamente para el Grado de Ingeniería Informática y la especialidad de Computación. Este trabajo se encuentra dentro de la modalidad de proyectos realizados dentro de la UPC.

El proyecto pretende desarrollar una serie de componentes que permitan una mejor comprensión de algunos de los conceptos de la computación gráfica que generan más dificultades para ser entendidos. Estos componentes tienen como objetivo ser utilizados por parte de los docentes de las asignaturas relacionadas con gráficos que se imparten dentro de la facultad.

1.2 Formulación del problema

En el grado de ingeniería informática de la FIB-UPC se enseña la programación de gráficos por ordenador en las asignaturas de Interacción y diseño de interfaces (obligatoria) [1] y Gráficos (obligatoria de la especialidad de computación) [2]. Estas materias son impartidas por parte de los profesores que pertenecen al departamento de ciencias de la computación.

En el campo de la computación gráfica se presentan varios conceptos que pueden resultar difíciles de asimilar por parte del alumnado. Algunos estudiantes no entienden las ideas expuestas durante las clases, y luego se encuentran perdidos en el momento de aplicar su conocimiento en la programación de gráficos. No entienden el significado de los parámetros de las funciones implicadas ni el efecto que tienen al cambiarlas de valor.

La falta de comprensión de estos conceptos puede deberse en parte al uso exclusivo de metodologías y herramientas tradicionales para la enseñanza de un campo que es, de manera intrínseca, muy visual. La inclusión de programas interactivos facilitaría la captación de conocimiento a los estudiantes pues les permitiría representar problemas y experimentar con diferentes configuraciones [3].

Por lo tanto, tomamos como objetivo del proyecto el crear una colección de programas que faciliten la docencia de las asignaturas de gráficos. De esta manera dotaríamos a los docentes de este departamento con una nueva herramienta, que les servirá como soporte digital para reforzar las explicaciones dadas en las clases de teoría. Además, haciendo uso de las tecnologías web podemos asegurar un mayor acceso a los componentes por parte de los estudiantes, al no ser necesaria la instalación de ningún *software* adicional para su ejecución y el poder acceder desde prácticamente cualquier lugar.

1.3 Actores implicados

Al ser este un proyecto que tiene como principal objetivo el ayudar a la docencia en las asignaturas de gráficos, encontramos dos grupos claramente diferenciados.

- **Profesorado de las asignaturas de gráficos.** Es el principal grupo de actores implicados que se beneficiará de la elaboración de este proyecto. Los docentes podrán utilizar los componentes desarrollados e incluirlos dentro de una web que les ayude con su trabajo de docencia. En este grupo se encuentran tanto la directora como el codirector del proyecto.
- **Alumnado de las asignaturas de gráficos.** Ellos se beneficiarán de los componentes creados puesto que son quienes más los utilizarán. Interactuar directamente con los programas les ayudará a comprender de una manera más intuitiva los conceptos, ahorrándoles el tiempo utilizado en tratar de comprender la funcionalidad de cada parámetro.

1.4 Definiciones

- **Gráficos por computador**: Es el campo de las ciencias de la computación que se encarga de estudiar métodos para la generación de imágenes digitales y la manipulación de contenido visual a través de ordenadores.
- **HTML5**: El término se refiere a un conjunto de tecnologías web modernas. Esto incluye el estándar actual de HTML, junto con las APIs de JavaScript para mejorar el acceso al almacenamiento, multimedia y hardware.
- **WebGL**: Es una API de JavaScript basada en OpenGL que se utiliza para procesar gráficos 2D y 3D interactivos de alto rendimiento dentro de cualquier navegador web compatible sin el uso de *plugins*.
- **API**: Es un conjunto de definiciones y protocolos que se usa para diseñar e integrar el *software* de las aplicaciones.
- ***Framework* (*software*)**: Es una plataforma que proporciona una base para desarrollar aplicaciones de *software*

1.5 Estado del arte

En este apartado se describen las diferentes soluciones propuestas para el problema del proyecto y se justifica la necesidad de realizarlo después de haber analizado las alternativas actuales.

1.5.1 Libro de gráficos de la facultad

El departamento de ciencias de la computación de la facultad ya intento ayudar al entendimiento de los conceptos de gráficos con la publicación de un libro para las asignaturas relacionadas [4]. Este no solo era un libro de texto con explicaciones tradicionales, sino que también incluía pequeñas aplicaciones que servían a modo de ejemplo interactivo. Estas

ayudaban a comprender mejor el efecto de los diferentes parámetros de la programación de gráficos por computadora.

Estas aplicaciones son los programas tutoriales creados por Nate Robins [5]. Entre estos se encuentran, por ejemplo, uno que facilitaba el entendimiento del funcionamiento de los diferentes tipos de cámara y de los parámetros que se utilizan en las diferentes funciones de OpenGL.

Pero por parte del profesorado, se desaconseja el uso de este libro digital ya que su instalación resulta excesivamente difícil. Esto se debe a que se utilizó para su elaboración una tecnología que ya se encuentra en desuso.

Esto nos lleva a descartar por completo esta opción, ya que, es inutilizable en la práctica. Lo que se podría extraer serían las ideas de los *applets* creados y los diseños de sus interfaces de usuario.

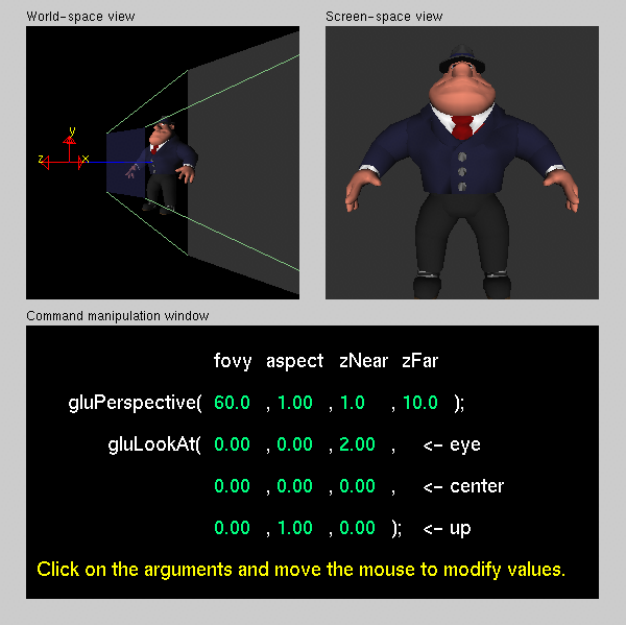

Figura 1: Programa tutorial de tipos de cámara de Nate Robins. Fuente: página web de Nate Robins [5]

1.5.2 Curso de computación gráfica de la Universidad de Tartu

Siguiendo con el mismo objetivo, la Universidad de Tartu cuenta con un conjunto de materiales online, entre los cuales, se encuentran pequeños ejemplos dinámicos que acompañan a las explicaciones que hay en su web [6]. Con este enfoque obtenemos la ventaja de que el recurso está disponible en cualquier lugar desde donde haya un dispositivo que se pueda conectar a Internet.

En el material públicamente accesible hay 37 ejemplos interactivos [7], destacando los que se encuentran dedicados a transformaciones geométricas o bien, proyecciones de cámara o también, material de objetos.

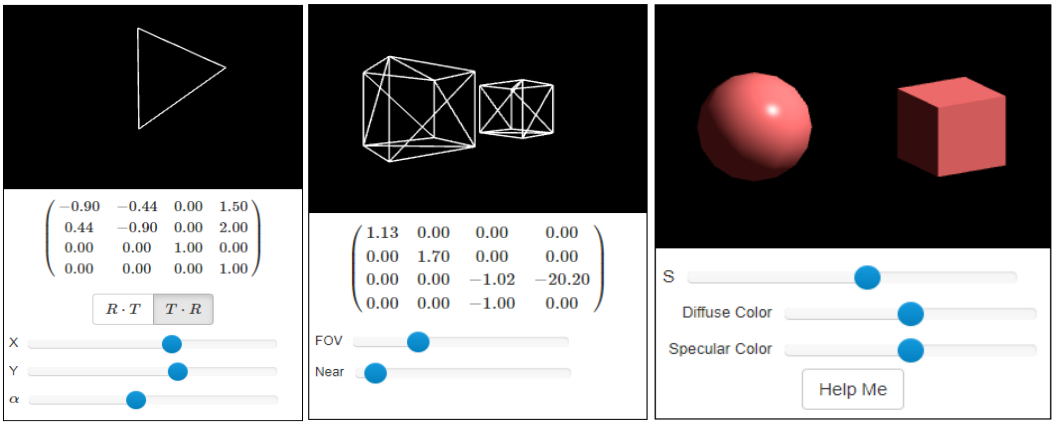

Figura 2: Ejemplos interactivos del curso de computación gráfica de la Universidad de Tartu. Fuente: elaboración propia

Pero estos ejemplos no se acaban de adaptar a nuestras necesidades. Entre otras cosas, los dedicados a transformaciones geométricas solo tratan los objetos que hay en un espacio 2D, cuando lo ideal sería tenerlo en 3D. También, nos encontramos limitados al momento de cuantas transformaciones se pueden aplicar a un objeto o en qué orden se pueden hacer. De igual manera, hay carencia en la modularidad y en los parámetros con los que se interactúa en el resto de los programas.

1.5.3 Web *applets* de la Universidad de Marburg y la Universidad de Toronto

Dentro del curso *Graphics Programming* [8] de la Universidad de Marburg también encontramos demostraciones interactivas creadas como apoyo a la docencia de la asignatura. De entre todos los programas, el que más nos interesa seria el dedicado a iluminación y material de un objeto [9], ya que es el único que interacciona con el contenido explicado en las asignaturas de gráficos de la facultad [1], [2]. Este programa incluye un panel que nos dota de la posibilidad de editar directamente el código utilizado para simular la iluminación de un objeto.

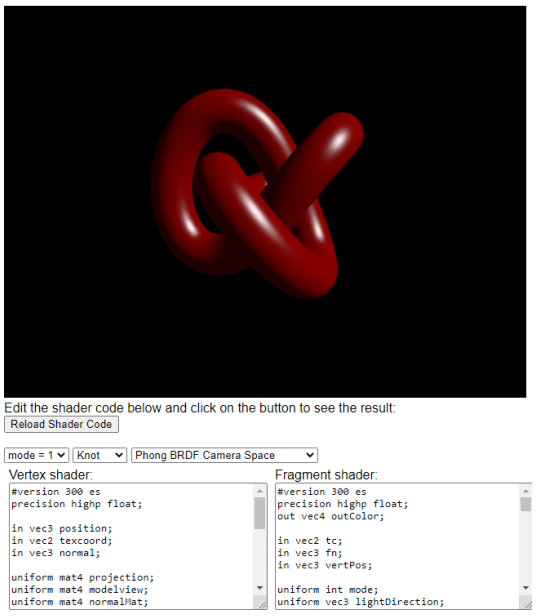

Figura 3: Web applet de la Universidad de Marburg dedicado a la iluminación y material. Fuente: WebGL example: Phong de la Universidad de Marburg [9]

Derivado del programa de la Universidad de Marburg, tenemos el ejemplo interactivo [10] de la Universidad de Toronto que forma parte de un conjunto de web *applets* creados por Johannes Kehrer. Este último, tiene una mayor riqueza para el proyecto porque incluye además un panel que nos permite modificar los valores del material y de la luz sin tener que editar código, a diferencia de como ocurría en el programa original. Gracias a que se utiliza una interfaz más simple e intuitiva, se consigue que al alumnado le sea más fácil interactuar con el programa, comprendiendo así, los conceptos de mayor dificultad.

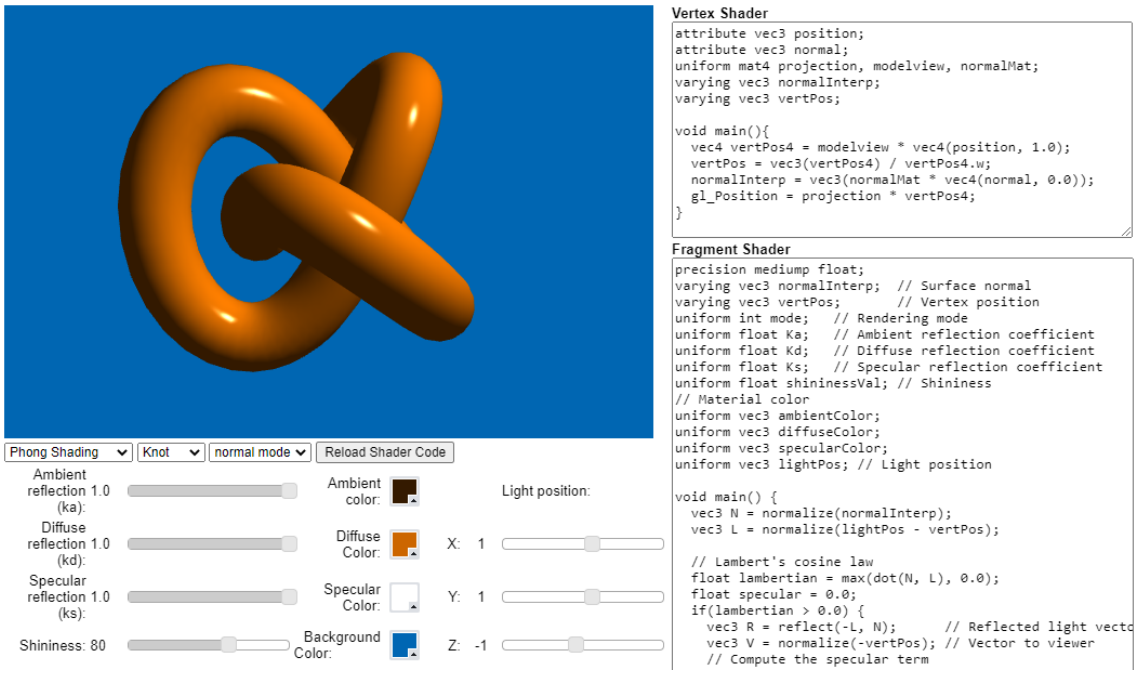

Figura 4: Web applet de la Universidad de Toronto dedicado a la iluminación y material. Fuente: WebGL Phong Shading de la Universidad de Toronto [10]

1.5.4 Justificación

Sabiendo que el diseño de todos estos programas ya se encuentra consolidado, se puede usar como ejemplo para uno de nuestros componentes que se encargue de mostrar estos conceptos. Además, se mejorarán los *applets*, añadiendo más funcionalidades como la de seleccionar el sistema de coordenadas para el foco en el componente de material y luz, o bien, ofrecer la posibilidad de añadir y reordenar transformaciones geométricas en el programa dedicado a este tema. Por ello, estos recursos pueden servir como inspiración y como punto de partida para la elaboración de este trabajo.

2 Alcance

2.1 Objetivos y subobjetivos

El objetivo principal de este trabajo es facilitar la enseñanza de las asignaturas relacionadas con gráficos de la facultad. Para poder facilitar las explicaciones, se desarrollarán diferentes componentes independientes, que se puedan cargar en una página web y/o añadirlos a una que ya tenga otros elementos. Estos componentes se mostrarán a manera de ejemplos de aquellos conceptos que sean más complicados en la programación de gráficos por computador.

Los conceptos que se tratarán son:

- **Codificación del color.** Para poder mostrar un color en el ordenador este debe estar codificado de forma numérica. Existen diferentes modelos empleados para este fin.
- **Transformaciones geométricas.** Mostrar el efecto de los diferentes parámetros y mostrar la no conmutatividad al aplicar las transformaciones.
- **Tipos de cámaras.** Diferencias entre cámara perspectiva y ortogonal, y la funcionalidad de cada parámetro.
- **Iluminación.** Ver el efecto de los parámetros de una fuente de luz en diferentes tipos de materiales.

Del objetivo descrito anteriormente mencionado se derivan los siguientes subobjetivos:

- Aprender desarrollo web. Para crear estos componentes es necesario saber utilizar las distintas herramientas de desarrollo web.
- Analizar las funcionalidades que tienen los diferentes componentes en común.
- Investigar los diferentes espacios de color. La codificación de un color depende del espacio en el que esté codificado.
- Cargar uno o más modelos en una escena.
- Control de la cámara de una escena pudiendo decidir desde qué parte se observa.
- Repasar los distintos algoritmos de iluminación básica enseñados en las asignaturas principales de gráficos por ordenador.
- Auto ajuste de los componentes. Los componentes deben mostrarse de manera correcta al cambiar el tamaño y relación de aspecto de la pantalla, pudiendo adaptarse a cualquier escenario.

2.2 Requisitos

Los requisitos por cumplir para considerar los objetivos cumplidos son:

- Los componentes deben ser capaces de funcionar en los principales navegadores de la actualidad.
- Los componentes se deben poder cargar de manera independiente.

- Los componentes deben ser livianos en la ejecución en cuanto a requerimientos hardware.
- Los componentes deben cargar rápidamente.
- La inserción de los componentes en cualquier página web debe ser simple y sencilla.
- Los componentes deben ser interactivos y no completamente estáticos para ayudar a la comprensión de los conceptos.

2.3 Riesgos y posibles soluciones

En cualquier desarrollo de un proyecto pueden aparecer imprevistos que ralentizan o impiden el deseado seguimiento de la planificación planteada. A continuación, se identifican las potenciales debilidades y se muestran posibles maneras de reducir el impacto provocado por su aparición.

2.3.1 Bugs

Programar y que todo funcione bien a la primera es imposible. Y más en un lenguaje que es completamente nuevo. Es natural, aunque indeseable, que haya errores en el código que impidan el correcto funcionamiento del programa. Seguramente será necesario invertir tiempo y tener mucha paciencia para encontrar estos errores y obviamente más tiempo y paciencia para hallar una solución a estos.

Este es un riesgo con alta probabilidad de aparición sobre todo si pensamos en que esta es una tecnología en la que no tengo ninguna experiencia. Esto puede llegar a retrasar de manera considerable el desarrollo, llegando a suponer hasta una semana de demora, con todos los gastos extra de recursos que implica. Por esta razón, he decidido dedicar un tiempo a la formación en los lenguajes de programación necesarios para desarrollar este proyecto. Esto permitirá reducir el riesgo de aparición de bugs o al menos mejorar el tiempo en el que estos se solucionen.

2.3.2 Gestión del tiempo

Este es un proyecto que se realiza como Trabajo Final de Grado y debe ser completado en el tiempo estipulado. Esto impone un reducido margen de maniobra a posibles desviaciones temporales como la mala planificación de las tareas a realizar durante la ejecución del proyecto.

Este es uno de los principales problemas que pueden surgir durante la realización de este proyecto. La planificación hecha puede no estar bien calculada debido a calcular mal el número de horas que se debe dedicar a cada tarea. Es por esto por lo que se ha decidido hacer reuniones periódicas con los tutores, para evitar cualquier desajuste del plan. De esta manera, se tienen distintos puntos de sincronización con el plan propuesto.

Pero, aun así, se ha decidido planear acabar el proyecto con 10 días de antelación respecto a la fecha límite, para poder adaptarnos a cualquier desvío del plan inicial.

3 Metodología y rigor

3.1 Metodología de trabajo

En la actualidad encontramos que existen diversas metodologías de trabajo. Escoger entre una u otra es una decisión de considerable importancia ya que esta es una herramienta muy importante al ayudar a optimizar los recursos y reducir riesgos en los proyectos.

Escogeré la metodología *agile* Kanban visto que me dio buenos resultados al aplicarla en la primera asignatura de proyectos que hice, PROP. Esta metodología se basa en el uso de cartas visuales. Esto nos permite ver en todo momento el estado de las tareas y poder seguir el flujo del trabajo. Estas tarjetas se reparten en columnas que indican el estado actual de cada una de las tareas. Las columnas utilizadas son:

- To Do: aquí están las tareas pendientes por hacer, que todavía no han sido empezadas.
- Developing: aquí están las tareas que se están realizando en el momento actual.
- Testing: aquí están las tareas que ya se han desarrollado, pero falta por comprobar su correcto funcionamiento.
- Done: aquí están las tareas que se han completado y testeado.

Para aplicar esta metodología utilizar Trello, una web que te permite crear el tablero y las cartas de tareas necesarias para este método.

3.2 Seguimiento y validación

Utilizaré Git como herramienta de control de versiones y GitHub para almacenar el código en un repositorio en la nube. Estas herramientas me facilitarán la recuperación de versiones anteriores en caso de que ocurra cualquier fallo y me permitirá tener disponibilidad del código en todo momento y lugar. En la rama principal solo deberá estar el código que haya pasado sus correspondientes pruebas. El repositorio estará compartido con los directores del TFG para que puedan seguir el progreso del proyecto en todo momento.

También se tiene planeado organizar reuniones esporádicas con los directores para supervisar el estado del trabajo y aprobar las tareas que se realizaran para el siguiente encuentro. Las reuniones se pensaron para ser virtuales, a través de Google Meet para así evitar el desplazamiento innecesario de las personas involucradas. En caso de no poder acordar una fecha o no considerar necesaria una reunión para tratar los temas, se puede sustituir por informes vía correo electrónico. Estos informes estarían pensados más para la parte que tiene que ver con la memoria, y así poder revisar su estado y señalar sus defectos para mejorar el documento.

4 Planificación temporal

4.1 Recursos

Para poder realizar este proyecto serán necesarios diferentes tipos de recursos. Estos los podemos clasificar en dos tipos.

4.1.1 Recursos humanos

En este proyecto participarán cuatro personas, que se pueden agrupar en forma de recursos de la siguiente manera:

- YO: Soy el principal recurso humano ya que soy quien se encarga del desarrollo del proyecto.
- DD - Directora y codirector: Otra parte fundamental son los directores del proyecto, ya que supervisan la elaboración del TFG.
- TG - Tutor de GEP: Se encarga de asistir en la gestión del proyecto durante el primer mes.

4.1.2 Recursos materiales

- PC: El principal dispositivo que se utilizará para realizar este proyecto será mi portátil. Cuenta con un Intel® Core™ i7-5500U como CPU y 8GB de RAM [11].
- CH - Chrome: Navegador que se utilizará para buscar la información necesaria para la realización del proyecto.
- GD - Google Docs: Procesador de texto que se utilizará para la redacción de los distintos documentos requeridos para el proyecto.
- ZT - Zotero: Gestor de referencias que se utilizará para la elaboración de la memoria del proyecto.
- GP - GanttProject: Herramienta para la creación de diagramas de Gantt.
- GM - Google Meet: Herramienta para la realización de videoconferencias.
- AT - Atenea: Web que se utilizará para el material y la comunicación durante el primer mes en GEP.
- TL - Trello: Herramienta que se utilizará para la aplicación de Kanban.
- VS - Visual Studio Code: Editor de texto que se utilizará para escribir el código.
- GH - Git y GitHub: Herramientas que se utilizarán para el control de versiones.
- RD - Render: Herramienta que me permite tener una página web estática gratuita.

4.2 Planificación inicial

4.2.1 Definición de tareas

Tareas relacionadas con la gestión del proyecto

- GP1 - Contextualización y alcance del proyecto: Redactar el documento que contextualiza el trabajo, define el problema, justifica el proyecto, acota su alcance y expone la metodología a utilizar.
- GP2 - Planificación temporal: Redactar el documento que explica la propuesta inicial de planificación temporal del proyecto.
- GP3 - Gestión económica y sostenibilidad: Redactar el documento donde se debate sobre las dimensiones económicas, sociales y ambientales de mi TFG.
- GP4 - Integración documento final de gestión: Unificar los tres documentos anteriores aplicando los cambios propuestos.
- GP5 - Reuniones: Cada dos semanas me reuniré con la directora y el codirector del TFG para analizar la evolución del proyecto.
- GP6 - Documentación de la fase de seguimiento: Redactar el informe que se presentará a los directores del TFG.
- GP7 - Documentación de la fase final del proyecto: Redactar lo que queda de la memoria del proyecto.
- GP8 - Preparación de la presentación del proyecto: Preparar la presentación para defender el proyecto delante del tribunal.

Tareas relacionadas con aprender a utilizar WebGL

- GL1 - Recordar OpenGL: Repasar los conceptos aprendidos para OpenGL ya que WebGL está basado en este.
- GL2 - Búsqueda y selección de un curso WebGL: Buscar que cursos hay disponibles en la web para aprender WebGL y seleccionar el que más se adecue a mis conocimientos y necesidades.
- GL3 - Curso WebGL: Realizar el curso sobre WebGL.
- GL4 - Implementación de un programa simple: Implementar un programa simple con una iluminación básica que cargue diferentes modelos a seleccionar para poner a prueba los conocimientos adquiridos.

Tareas relacionadas con el desarrollo del *framework*

- FW1 - Estudio sobre los requisitos del *framework*: Analizar los aspectos básicos y comunes a todos los componentes que serían requeridos para trabajar con gráficos y nos facilite el desarrollo del proyecto.
- FW2 - Implementación del *framework*: Implementar las funcionalidades definidas en el apartado anterior.

Tareas relacionadas con el desarrollo de componentes

- CP1 - Estudio y selección de componentes a desarrollar: Repasar los distintos conceptos y seleccionar un conjunto de los que se quieren explicar. Se utilizará el antiguo libro de gráficos de la facultad como referencia. Este conjunto debe cumplir con los requisitos mínimos de trabajo esperado para un TFG.
- CP2 - Diseño y mejora de las ideas: Diseñar los componentes y/o mejorar las ideas propuestas en el libro.
- CP3 - Implementación de componentes: Implementar los componentes acordados utilizando el *framework* desarrollado.

Tareas relacionadas con la presentación de componentes

- CC1 - Web de muestra: Implementación de una web que contenga los componentes a modo de presentación para la entrega final

4.2.2 Resumen de tareas

ID	Nombre	Tiempo (h)	Dependencias	Recursos
Gestión del proyecto				
GP1	Contextualización y alcance del proyecto	24		YO, PC, CH, GD, AT, ZT
GP2	Planificación temporal	20		YO, PC, CH, GD, AT, GP
GP3	Gestión económica y sostenibilidad	20		YO, PC, CH, GD, AT
GP4	Integración documento final de gestión	15	GP1, GP2, GP3	YO, PC, CH, GD, AT, ZT, GP
GP5	Reuniones e informes	10		YO, DD, GM
GP6	Informe de seguimiento	40	GP4	YO, PC, CH, GD, AT, ZT
GP7	Memoria del proyecto	40	GP6	YO, PC, CH, GD, AT, ZT
GP8	Preparación de la presentación del proyecto	45		YO, PC, CH, AT
Desarrollo del proyecto				
Aprender a utilizar WebGL				
GL1	Recordar OpenGL	20		YO, PC, CH, GD
GL2	Búsqueda y selección de un curso WebGL	2		YO, PC, CH
GL3	Curso WebGL	35	GL2	YO, PC, CH, VS, GH
GL4	Implementación de un programa simple	5	GL1, GL3	YO, PC, CH, VS, GH
Desarrollo del *framework*				
FW1	Estudio sobre los requisitos del *framework*	25		YO, PC, CH
FW2	Implementación del *framework*	40	FW1, GL4	YO, PC, CH, VS, GH

Desarrollo de componentes				
CP1	Estudio y selección de componentes a desarrollar	30		YO, PC, CH
CP2	Diseño y mejora de las ideas	55	CP1	YO, PC, CH
CP3	Implementación de componentes	100	CP1, CP2, FW2	YO, PC, CH, VS, GH
Presentación de componentes				
CC1	Web de muestra	8	CP3	YO, PC, CH, VS, GH, RD
Total		534		

Tabla 1: Resumen de las tareas de la planificación inicial. Fuente: elaboración propia

4.2.3 Gantt inicial

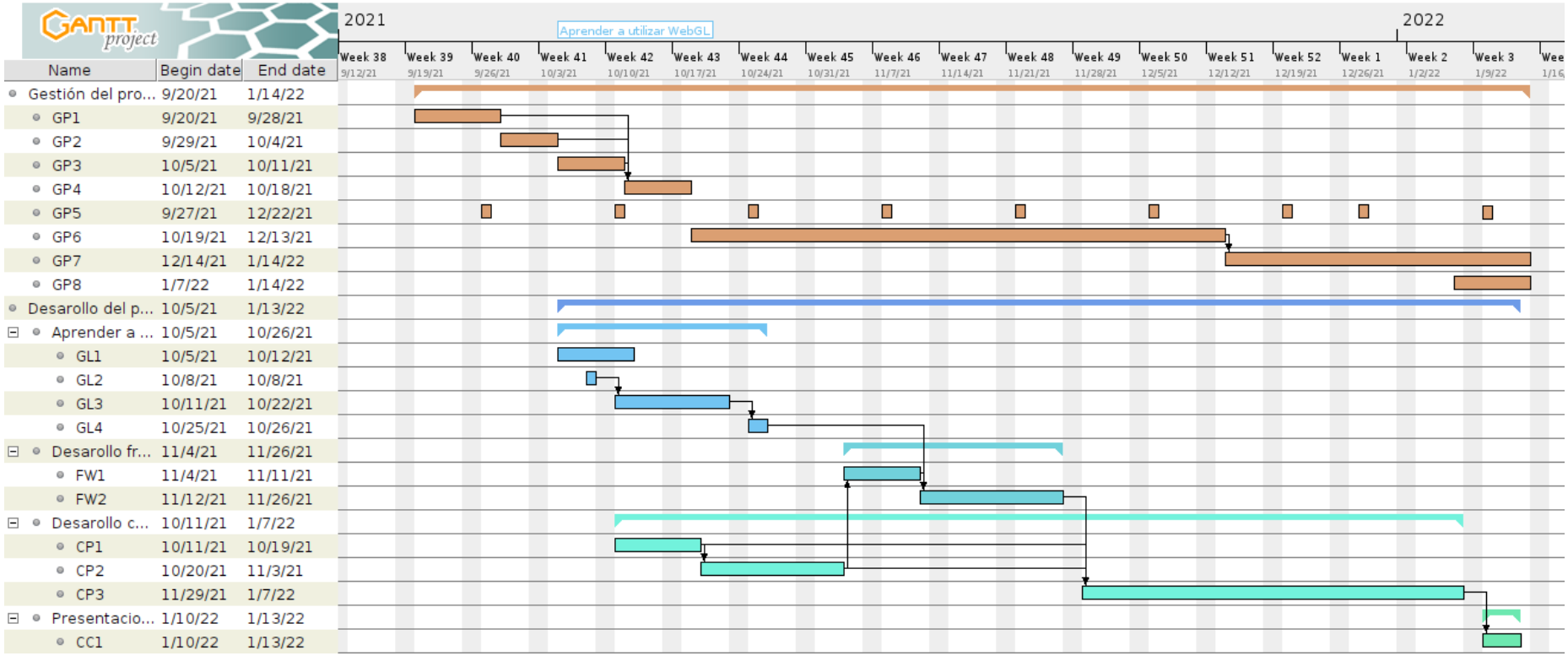

Figura 5: Diagrama de Gantt de la planificación inicial. Fuente: elaboración propia

4.3 Cambios respecto a la planificación inicial

La planificación inicial tenía previsto acabar el desarrollo del proyecto y entregarlo a mediados de febrero del 2022. El plan inicial se siguió correctamente hasta mediados de noviembre, donde ya se habían realizado con éxito todas las tareas correspondientes a este periodo. Incluso se había avanzado ligeramente respecto al plan inicial al haber tenido ya en funcionamiento los componentes referentes a la codificación del color, *Acierta el color* y *Comparación entre modelos de color*, ya que estos no utilizan el *framework* 3D.

Pero debido a una deficiente gestión temporal en el siguiente tramo por mi parte, y la infravaloración del coste temporal para desarrollar la parte principal encargada de gestionar las tareas gráficas (*framework* 3D), se tuvo que pedir una prórroga de un cuatrimestre adicional para poder finalizar el trabajo. Así pues, la nueva fecha de entrega, y definitiva, se mueve a finales de marzo.

A causa de haber alargado el tiempo dedicado en la elaboración del *framework*, el inicio del desarrollo de los componentes dependientes de este se ha tenido que retrasar. Para mostrar estos cambios en el nuevo diagrama de Gantt se divide la tarea *CP3 Implementación de componentes* en seis tareas más pequeñas vinculadas a la realización de cada uno de los web *applets* ya que la fase de diseño de finalizo y se tienen definidas los programas a realizar. Estas serán de las que van desde CP3 a CP8.

De la misma manera se atrasa la elaboración de la web de ejemplo, el informe de seguimiento, la memoria y la presentación. Estas se dejan a comenzar después de finalizar toda la parte dedicada al desarrollo.

Otro cambio importante es el de las reuniones previstas. Hasta la fecha han sido realizadas tres reuniones vía Google Meet y dos sincronizaciones por correo electrónico para mostrar el avance y/o acordar nuevos puntos. A partir de ahora se pasará a realizar informes que se enviaran por mail a los tutores para presentar los progresos realizados y que se valore y corrijan el trabajo hecho. Estos no requerirán un mayor esfuerzo ya que en principio serán las partes de la memoria que se van realizando.

4.3.1 Resumen de tareas

ID	Nombre	Tiempo (h)	Dependencias
Gestión del proyecto			
GP1	Contextualización y alcance del proyecto	24	
GP2	Planificación temporal	20	
GP3	Gestión económica y sostenibilidad	20	
GP4	Integración documento final de gestión	15	GP1, GP2, GP3
GP5	Reuniones e informes	10	
GP6	Informe de seguimiento	8	
GP7	Memoria del proyecto	65	
GP8	Preparación de la presentación del proyecto	45	
Desarrollo del proyecto			
Aprender a utilizar WebGL			
GL1	Recordar OpenGL	20	
GL2	Búsqueda y selección de un curso WebGL	2	

GL3	Curso WebGL	35	GL2
GL4	Implementación de un programa simple	5	GL1, GL3
Desarrollo del *framework*			
FW1	Estudio sobre los requisitos del *framework*	25	
FW2	Implementación del *framework*	65	FW1, GL4
Desarrollo de componentes			
CP1	Estudio y selección de componentes a desarrollar	30	
CP2	Diseño y mejora de las ideas	55	CP1
CP3	Implementación de componente: Adivina el color	20	CP1, CP2
CP4	Implementación de componente: Cambios entre modelos de color	5	CP1, CP2
CP5	Implementación de componente: Transformaciones geométricas	15	CP1, CP2, FW2
CP6	Implementación de componente: Tipos de cámaras	30	CP1, CP2, FW2
CP7	Implementación de componente: Efectos de focos RGB sobre una esfera	15	CP1, CP2, FW2
CP8	Implementación de componente: Modelo de Phong	20	CP1, CP2, FW2
Presentación de componentes			
CC1	Web de muestra	8	CP3, …, CP8
Total		557	

Tabla 2: Resumen de las tareas de la planificación final. Fuente: elaboración propia

4.3.2 Gantt final

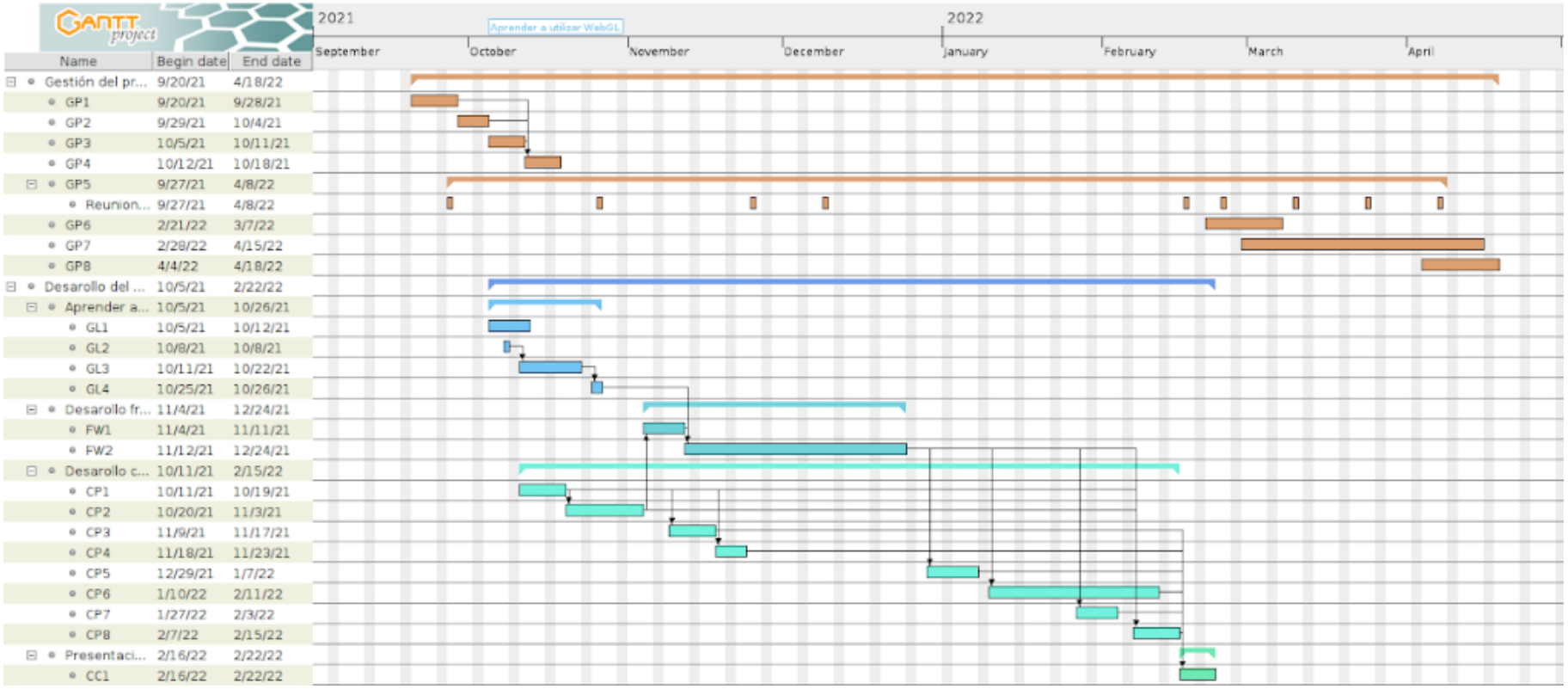

Figura 6: Diagrama de Gantt de la planificación final. Fuente: elaboración propia

5 Gestión económica

5.1 Costes del personal

Con la finalidad de estimar correctamente el coste derivado del personal primero tenemos que definir cuáles son los roles necesarios para la realización del proyecto y el salario por hora para estos perfiles. En este proyecto serán necesarios 5 roles.

- **Project Manager**. Responsable de la planificación y de la correcta ejecución del proyecto.
- **Programador**. Encargado de programar el código para desarrollar el proyecto.
- **Tester**. Verifica el correcto funcionamiento del código.
- **Escritor técnico**. Documenta todo el proceso de desarrollo y sus resultados.
- **Analista**. Es el encargado de la selección y diseño de los componentes.

Todos estos roles serán interpretados por mí ya que yo soy el principal actor de este proyecto. Pero el rol de Project Manager también involucra al tutor del GEP y los directores del TFG.

Rol	Coste (€) / h
Project Manager	28.97
Programador	19.46
Tester	15.29
Escritor técnico	19.50
Analista	18.75

Tabla 3: Coste por hora de los diferentes roles. Incluye costes de Seguridad Social. Fuente: información obtenida de los trabajos de fin de grado Comparative study of missing data treatment methods in radial basis function neural networks: is it necessary to impute? *[12, Tab. 7.1] y* Mòdul de correcció distribuït per a Jutge.org *[13, Tab. 2.2]*

Para poder calcular el coste total de este apartado debemos cuantificar el número de horas que desempeñará cada rol.

Tarea	Horas	Project Manager	Programador	Tester	Escritor técnico	Analista
Contextualización y alcance del proyecto	24	24	0	0	0	0
Planificación temporal	20	20	0	0	0	0
Gestión económica y sostenibilidad	20	20	0	0	0	0
Integración documento final de gestión	15	15	0	0	0	0
Reuniones e informes	10	10	0	0	0	0
Informe de seguimiento	40	0	0	0	40	0
Memoria del proyecto	40	0	0	0	40	0
Preparación de la presentación del proyecto	45	45	0	0	0	0
Recordar OpenGL	20	0	20	0	0	0
Búsqueda y selección de un curso WebGL	2	0	2	0	0	2
Curso WebGL	35	0	35	0	0	0

Implementación de un programa simple	5	0	5	0	0	0
Estudio sobre los requisitos del *framework*	25	0	0	0	0	25
Implementación del *framework*	40	0	25	15	0	0
Estudio y selección de componentes a desarrollar	30	0	0	0	0	30
Diseño y mejora de las ideas	55	0	0	0	0	55
Implementación de componentes	100	0	70	30	0	0
Web de muestra	8	0	6	2	0	0
Total	534	134	163	47	80	112

Tabla 4: Tiempo estimado por tarea. Fuente: elaboración propia

Finalmente estimamos el coste total de cada rol multiplicando el número de horas por el precio por hora.

Rol	Coste (€)
Project Manager	3881.98
Programador	3171.98
Tester	718.63
Escritor técnico	1560
Analista	2100
Total	11432.59

Tabla 5: Coste total del personal. Fuente: elaboración propia

5.2 Costes genéricos

5.2.1 Amortización de los recursos

Para calcular correctamente el coste del proyecto se debe tener en cuenta las partidas presupuestarias destinadas a los recursos materiales. En el caso de un proyecto, se tiene que incluir el *software* y el hardware necesarios. El coste de amortización para cada recurso se calcula de la siguiente manera.

$$\frac{Precio\ recurso}{Años\ de\ amortizacion * Dias\ laborables\ al\ año * Horas\ de\ uso\ diario} * Horas\ projecto$$

En el caso de este proyecto, el coste de amortización del *software* es nulo, ya que todos los programas que se utilizan son gratuitos. Pero no ocurre lo mismo con el coste de amortización del hardware, donde se tiene que calcular el coste del portátil HP.

Hardware	Precio (€)	Tiempo utilizado (h)	Amortización (€)
Portátil HP	1099	534	166.72

Tabla 6: Coste de amortización. Fuente: elaboración propia

5.2.2 Costes indirectos

Coste eléctrico

El precio del kWh varia de día a día, pero lo podemos aproximar a 0.27217€/kWh utilizando la media de hoy 06/10/2021 de las horas en las que se planea utilizar el portátil (9-17) [14].

Hardware	Potencia (W)	Tiempo utilizado (h)	Consumo (kWh)	Coste (€)
Portátil HP	56 [11]	534	29.904	8.14

Tabla 7: Coste eléctrico. Fuente: elaboración propia

Coste internet

La tarifa contratada de internet cuesta 30€ al mes. Como el proyecto se planea para durar 4 meses y se calculan unas 4.5 horas diarias de media, el coste será:

$$4\ meses * (30€/mes) * (24.5h/24h) = 22.5€$$

Coste transporte

Dado a que todo el proyecto se realizará desde mi casa y ninguna reunión requerirá desplazamiento, el coste de transporte es 0€.

Coste espacio de trabajo

Como se ha comentado anteriormente, el proyecto será realizado en mi casa. El coste del alquiler es de 600€ al mes. Debido a que no se utiliza todo el espacio para la realización del proyecto, he visto conveniente dividir el precio del alquiler entre 4. El coste será:

$$4\ meses * \big((600/4)\ €/mes\big) * (4.5h/24h) = 112.5€$$

5.2.3 Resumen de costes genéricos

Concepto	Coste (€)
Amortización	166.72
Coste eléctrico	8.14
Coste internet	22.5
Coste transporte	0
Coste espacio de trabajo	112.5
Total	309.86

Tabla 8: Coste genérico. Fuente: elaboración propia

5.3 Otros costes

5.3.1 Contingencias

Como en todo proyecto, durante la realización de este pueden surgir cualquier imprevisto que requiera de más dinero. Teniendo esto en cuenta, propongo añadir un 15% al coste total (CPA+CG) para tener un margen de contingencia. Esto supone un coste adicional de 1761.37€.

5.3.2 Costes imprevistos

Planteamos los siguientes costes debido a incidentes multiplicando la probabilidad de que estos sucedan y el coste estimado para cada uno de los imprevistos.

Incidente	Coste estimado (€)	Riesgo (%)	Coste (€)
Bugs (20h)	550	30	165
Gestión del tiempo (35h)	720	60	432
Total	1270	-	597

Tabla 9: Costes imprevistos. Fuente: elaboración propia

5.4 Resumen de coste total

Actividad	Coste (€)
CPA	11432.59
CG	309.86
Coste contingencia	1761.37
Costes imprevistos	597
Total	14100.82

Tabla 10: Coste total del proyecto. Fuente: elaboración propia

6 Informe de sostenibilidad

6.1 Dimensión económica

El impacto del proyecto en el ámbito económico se puede ver en detalle en el apartado de gestión económica. En resumen, el coste total, contando recursos humanos y materiales, es de 14100.82€.

Debido al aplazamiento de la fecha de entrega respecto a la que se tenía prevista al principio, los costes del personal y los costes genéricos aumentaron. El incremento en cuanto a las horas de trabajo previstas es de un 4.3%. Pero esta desviación del presupuesto queda dentro de los márgenes destinados a contingencias, que se propuso de un 15%. De modo que el coste final se mantiene dentro del inicialmente planteado.

Los costes derivados de la vida útil del proyecto me resultan difíciles de calcular, ya que estos dependen de la integración de los componentes que hagan los docentes de las asignaturas de gráficos. Ellos tenían planeado crear una página web con explicaciones, en las que aparecerían los programas desarrollados a modo de ejemplos interactivos. Por tanto, el coste seria el del mantenimiento del servidor donde se alojará la web. Seguramente harán uso del que ya utilizan para las webs dedicadas a los recursos actuales de gráficos.

En cuanto a la viabilidad económica, no se esperan tener ningún tipo de beneficio financiero de la realización de este trabajo, ya que es un proyecto dedicado a la docencia dentro de la facultad. De la misma forma, no se contemplan casos donde el gasto pueda augmentar de manera significativa.

6.2 Dimensión ambiental

Al ser este un proyecto de desarrollo de *software*, el impacto ambiental se ve reducido al consumo energético producido por los equipos utilizados durante la elaboración de los programas. Así pues, podemos cuantificar el consumo final a partir de las horas totales dedicadas y la potencia del portátil, único dispositivo utilizado.

$$557\ horas * 56\ W = 31.192 kWh$$

Durante la puesta en producción no se han tomado medidas destacables para reducir el impacto ambiental. Sin embargo, se ha intentado trabajar siempre con los mínimos recursos necesarios. Así que, si se volviera a realizar el proyecto, difícilmente habría lugar de donde poder disminuir el consumo.

De igual manera que con el coste económico, el coste ambiental durante la vida útil del proyecto será el del servidor que alberge la web donde se muestren los componentes desarrollados. Cabe destacar que esto no supondría un gran aumento, ya que probablemente harán uso de los que ya disponen en la actualidad. Aun así, en conjunto, se podría llegar a una mejora en la huella ecológica. Esto se debe principalmente al potencial de rebajar el tiempo que dedica el alumnado al estudio, significando esto un menor consumo.

Con relación a los riesgos que pueden surgir, no se consideran escenarios que pudieran hacer aumentar la huella ecológica. No se espera que haya un crecimiento considerable en el número

de usuarios de los componentes, que se reduce a los alumnos de la facultad. Consiguientemente, el consumo se mantendría constante a lo largo del ciclo de vida del proyecto.

6.3 Dimensión social

Durante la realización de este trabajo, he reforzado mis habilidades en la gestión de proyectos y desarrollo de *software*, más específicamente en lo que respecta a tecnologías de desarrollo web. Al mismo tiempo, he profundizado en el entendimiento de la programación de gráficos por computador.

El proyecto tendría un impacto positivo sobre los docentes de las asignaturas de gráficos y sus alumnos. Por parte del profesorado, ellos cuentan con una nueva herramienta que pueden incorporar a sus clases. Por la parte del alumnado, ellos se beneficiarían de los componentes al facilitarles la comprensión de los conceptos más difíciles. En ningún caso se contempla que haya un colectivo ni segmento que pueda verse perjudicado por el proyecto, ya que este solo afecta a los grupos mencionados anteriormente.

La solución propuesta no soluciona de manera total el problema planteado, que es la dificultad en el entendimiento de algunos conceptos de la programación de gráficos por computador. Los programas no cubren todos los conceptos, pero si un pequeño subconjunto de los que se consideraban más importantes. Es por ello por lo que estos nuevos instrumentos se tienen como ayuda extra, pero no se llegan a considerar como una respuesta definitiva.

En lo que atañe a los riesgos, no se contempla ningún riesgo que pueda perjudicar a ningún segmento de la población ni provocar algún tipo de dependencia que pudiera dejar a los usuarios en posición de debilidad.

7 *Framework* 3D

Este trabajo va dirigido a desarrollar una serie de componentes web interactivos para ayudar a la enseñanza de la programación en gráficos. La mayoría de estos componentes tienen como requisito el crear y mostrar escenas 3D animadas en el navegador. Por este motivo decidimos elaborar un *framework* simple y ligero que será el encargado de tratar con la parte gráfica de los componentes. Además, de este modo nos evitamos tener que repetir código por cada componente, ya que extraemos las llamadas comunes a la API de WebGL.

7.1 *Pipeline* gráfico WebGL

Para poder procesar y visualizar gráficos 3D se tienen que seguir una serie de pasos. Estos pasos se les conoce como el *pipeline* gráfico. En nuestro caso, vamos a ver el *pipeline* gráfico de WebGL, ya que utilizaremos el su API.

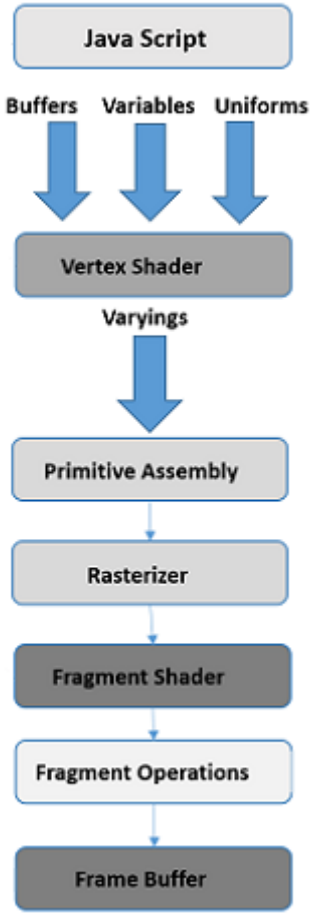

Figura 7: Pipeline gráfico WebGL. Fuente: TutorialsPoint [15]

De este *pipeline* las únicas partes programables son el *vertex shader* y el *fragment shader*. Los *shaders* son programas que se ejecutan en la GPU y se encargan de alguna etapa del procesado de gráficos. No obstante, también tenemos algo de libertad en algunas partes configurables de las *fragment operations*.

JavaScript

JavaScript es el lenguaje utilizado para realizar el control del flujo del programa. Las acciones para las que se utiliza son:

- Inicializar el contexto WebGL.
- Crear, compilar y enlazar los *shaders.*
- Generar la geometría de los modelos 3D.
- Crear loa *buffer objects* para guardar la información de los vértices.
- Enviar información que cambia a cada cuadro a través de los *uniforms.*

Vertex shader

Todo el proceso comienza cuando se envía a dibujar una geometría. Para esto se utiliza las funciones de *drawElements* o *drawArray*. El *vertex shader* es ejecutado para cada uno de los vértices. En este momento se calcula la posición de cada vértice. También se puede calcular otros atributos como el color, el vector normal o las coordenadas de textura.

***Primitive assembly* y rasterización**

Algunas primitivas se dividen en una secuencia de primitivas bases individuales. Un ejemplo de esto sería pasar de una línea definida como una lista de 12 vértices a primitivas bases de 11 líneas. Después se interpolan los pixeles a partir de los vértices.

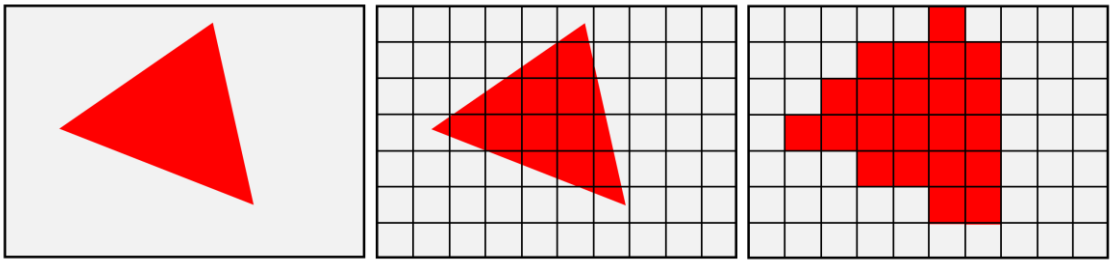

Figura 8: Proceso de rasterización. Fuente: TechSpot [16]

También se realiza la eliminación de triángulos que no se ven por su orientación (*face culling*) y el descarte de elementos que quedan fuera de la pirámide de visión (*clipping*).

Fragment shader

Procesa cada fragmento para calcular su color tomando en cuenta los datos generados por los pasos anteriores.

Fragment operations

Realiza diferentes test para determinar el color final de cada pixel. Entre estas pruebas se encuentra el *depth test*, que según el valor de cada píxel en el *depth buffer* (o *z-buffer*), decide si el fragmento se descarta o sus valores pasan al *frame buffer*.

Frame buffer

Es una parte de la memoria que contine los datos de la escena que se va a pintar.

7.2 Clases creadas

En esta sección, se describen las principales clases creadas para cumplir con los objetivos planteados anteriormente.

Object3D

Esta es la clase encargada de representar un objeto 3D. Este objeto está compuesto por una geometría y una matriz de transformación modelo. La matriz de modelo es la responsable de modificar la posición del objeto en el mundo. Esta posición se ve afectada por la posición, el escalado y la rotación del objeto.

Geometry

Es la clase base para las distintas geometrías. Se encarga de guardar las propiedades de los vértices, como posición y vector normal, y calcular la caja mínima contenedora del modelo. La caja mínima contenedora es el ortoedro más pequeño, alineado con los ejes, que encierra todos los vértices de la geometría. Cada geometría cuenta con parámetros que nos permite generar una mayor cantidad de figuras. Las geometrías creadas son:

- **PlaneGeometry**. Clase encargada de generar la geometría de un plano. Los parámetros son la altura y la profundidad.
- **CircleGeometry**. Clase encargada de generar la geometría de una circunferencia. Los parámetros son el radio y el número de lados.
- **BoxGeometry**. Clase encargada de generar la geometría de un ortoedro. Los parámetros son la altura, el ancho y la profundidad.
- **UVSphereGeometry**. Clase encargada de generar la geometría de una esfera formada por caras de cuatro vértices. Los parámetros son el radio y el número de segmentos y anillos, que son como los meridianos y paralelos de la Tierra.

Camera

Es la clase base para las cámaras. Las clases que derivan de esta son:

- **PerspectiveCamera**. Este tipo de cámara está diseñada para simular la manera en el que el ser humano ve.
- **OrthographicCamera**. Con este tipo de cámara el tamaño del objeto permanece constante sin importar la distancia a la cámara a la que este esté.

Scene

Esta clase representa una escena en tres dimensiones. Se encarga de contener a los diferentes objetos que luego se mostrarán.

SceneRenderer

Es la clase responsable de tratar con las llamadas a WebGL y se encarga de procesar y representar una escena utilizando una cámara. Cuenta con distintas opciones de configuración, como la de indicar los distintos *shaders* que se utilizaran o, por ejemplo, la de pintar los ejes del mundo o de cada uno de los objetos de la escena.

7.3 Implementación

A lo largo del desarrollo de las clases mencionadas en el apartado anterior ha habido aspectos que destacan por su complejidad. Es por esto por lo que a continuación se explican de manera más detallada como se ha llegado a implementar las distintas soluciones para estos aspectos.

7.3.1 Visualización de escena

Para animar una escena 3D utilizando WebGL es necesario cambiar los que quieres animar a lo largo del tiempo y dibujarlo una y otra vez. En el caso de JavaScript, cada vez que se quiere pintar un nuevo cuadro, se tiene que llamar a la función *requestAnimationFrame* y pasarle la función que se empleará para actualizar la animación. En nuestro caso, la función para animar es la que se encarga de pintar la escena.

Este modelo que se usa para animar nos favorece al permitirnos añadir la ejecución de código antes y después de pintar la escena de manera sencilla. Un ejemplo de esto sería el envío de datos necesarios para la ejecución de un *shader* específico o aplicar un filtro al resultado del primer procesamiento de la escena. De esta manera nos podemos centrar en solo encargarnos de lo básico, que sería activar el *shader* correspondiente y enviar a pintar las distintas geometrías. El paso del *shader* es importante, ya que nada nos garantiza que antes no se haya activado otro. Por un motivo similar, tenemos que ofrecer la opción de limpiar de manera automática o no los *buffers* antes de volver a empezar con todo el procesamiento. Esto es debido a que puede que previamente se haya pintado algo relevante que nos interese conservar.

Ahora bien, para conseguir que el proceso de pintado se ejecute de manera rápida en cada fotograma, es necesario guardar la información de la escena en objetos propios de WebGL. Utilizamos los *vertex buffer objects* (VBO) para guardar los datos de la geometría de los objetos de la escena en la memoria de la GPU. Además, para conseguir un código más eficiente, agrupamos todos los VBO de un mismo objeto en un *vertex array object* (VAO) para reducir el número de llamadas a WebGL en cada iteración.

7.3.2 Dibujado de los ejes

Dibujar los ejes de coordenadas del mundo es en principio una tarea sencilla. Basta con tener un modelo simple que contenga la información de la geometría y el color de los ejes. Pero en nuestro caso queremos tener algo más de modularidad a la hora de pintar los ejes. Queremos poder indicar dónde dibujar estos ejes, y así reutilizar el mismo modelo para pintar tanto los ejes del mundo como los ejes de cada uno de los objetos de la escena. También queremos decidir el estilo con el que dibujar los ejes, decidir entre utilizar una línea continua, discontinua o punteada.

7.3.2.1 Reutilización de un mismo modelo de ejes para diferentes objetos

Para el problema de la ubicación de los ejes la solución sería bastante sencilla. Bastaría simplemente con multiplicar el modelo de los ejes por una matriz que nos permitiera modificar su posición y tamaño. Esta matriz de transformación se podría obtener a partir de la transformación de modelo de cada objeto y de su caja mínima contenedora.

```
const box = object.geometry.boundingBox;
const center = box.center;

// Obtener trasformación para encajar el modelo de los ejes con
// la posicion y orinetacion actual del objeto
let t = object.transform
    .translate(center.x, center.y, center.z)
    .scale(box.width / 2, box.height / 2, box.depth / 2);
```

Figura 9: Código para adecuar ejes a un objeto. Fuente: elaboración propia

Pero como se ve en la Figura 10, esto resulta no ser suficiente. Al aplicar la transformación anterior encontramos que los ejes no se ven en todos los casos. Si por el ejemplo el objeto es un cubo opaco, la caja mínima contenedora coincide con las dimensiones de este, y los ejes no se verían al quedar dentro de los límites.

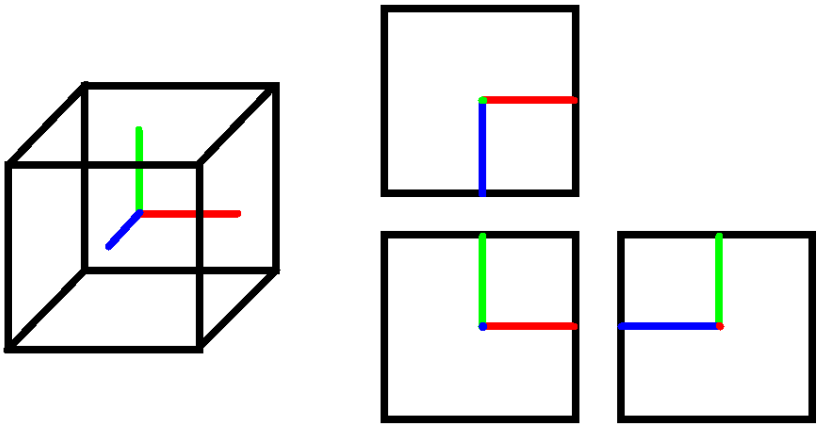

Figura 10: Esquema ejes con transformación inicial. Fuente: elaboración propia

Con el fin de poder ver correctamente los ejes, haría falta escalarlos para que sobresalieran del objeto. En nuestro caso queremos que los ejes sean 0.5 unidades más grandes en cada dirección, y que sobresalgan de manera uniforme en todos los lados.

```
// Hacer los ejes 0.5 mas grandes que la ccaja contenedor del objeto
const ll = 0.5;
let sx = new Vector3(t.data[0], t.data[1], t.data[2]).length();
let sy = new Vector3(t.data[4], t.data[5], t.data[6]).length();
let sz = new Vector3(t.data[8], t.data[9], t.data[10]).length();
sx = (sx + ll) / sx;
sy = (sy + ll) / sy;
sz = (sz + ll) / sz;
t = t.scale(sx, sy, sz);
```

Figura 11: Código para hacer sobresalir a los ejes. Fuente: elaboración propia

7.3.2.2 Estilo de los ejes

Con el objetivo de poder estilizar los ejes, se decide crear un *shader* específico para pintarlos. A través de parámetro, podremos decidir si los ejes son líneas continuas o discontinuas. En el caso de ser discontinuas también escoger el tamaño y distancia entre líneas.

En la Figura 12 se muestra el código del *fragment shader* específico para los ejes. La variable *dashed* indica como va a ser pintada la línea. Si es cierta, se pintará de manera discontinua. Si es falsa, la línea será continua. La variable *invDashLeght* es la inversa del tamaño de las líneas discontinuas. A partir de estas variables y la distancia del fragmento al origen del eje de coordenadas (*vOrigin*), se decide si el fragmento se pinta o se descarta.

```
#version 300 es
precision mediump float;

in vec3 vColor;
in vec3 vPosition;
in vec3 vOrigin;

out vec4 color;

uniform int dashed;
uniform float invDashLength;

void main() {
    if (dashed == 1) {
        float d = distance(vPosition, vOrigin);
        float n = mod(d * invDashLength, 2.0f);
        if (n < 1.0f ) {
            discard;
        } else {
            color = vec4(vColor.xyz, 1.0);
        }
    } else {
        color = vec4(vColor.xyz, 1);
    }
}
```

Figura 12: Fragment shader para pintar los ejes. Fuente: elaboración propia

7.3.3 Geometría esfera

Una de las geometrías básicas que tenemos es la de una esfera. Existen varias maneras de aproximar la geometría de una esfera. Nosotros utilizaremos el método de la esfera UV, debido a que es más fácil mapear una textura rectangular. Esto se ha hecho pensando en un posible futuro donde se utilicen texturas. Este tipo de estera es similar a un globo terráqueo, dividido por meridianos y paralelos, que para nosotros serán los segmentos y anillos.

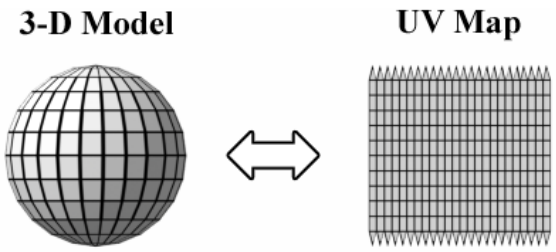

Figura 13: Aplicación de una textura a una esfera UV. Fuente: Wikipedia [17]

Cada geometría debe tener las posiciones de los vértices y sus vectores normales. Para generar las posiciones utilizaremos cuatro puntos, que representa uno de los cuadrados de la esfera. A partir de dos bucles crearemos la geometría. En el bucle más interno generaremos los anillos. Rotaremos sobre el eje Y tantas veces como segmentos haya. En el bucle más externo giraremos sobre el eje X tantas veces como el número de anillos más uno.

```
function positionsUVSphere(radius, segments, rings) {
    const positions = [];
    const a = 2 * Math.PI / segments;
    const b = Math.PI / rings;
    const ry = Matrix3x3.yRotation(a);
    const rx = Matrix3x3.xRotation(b);

    /**
     *  v0_____v1      [][]...          [][][][][][][][][]
     *  | \     |                       [][][][][][][][][]
     *  |   \   |   =>              =>          :
     *  |     \ |
     *  v2_____v3
     */

    // inicializar los cuatro puntos
    let v0 = new Vector3(0, radius, 0);
    let v1 = ry.mulVector3(v0);
    let v2 = rx.mulVector3(v0);
    let v3 = ry.mulVector3(v2);

    for (let i = 0; i < rings + 1; i++) {
        for (let j = 0; j < segments; j++) {
            // añadir los posiciones a la lista
            positions.push(v1.x, v1.y, v1.z, v0.x, v0.y, v0.z, v2.x, v2.y, v2.z);
            positions.push(v1.x, v1.y, v1.z, v2.x, v2.y, v2.z, v3.x, v3.y, v3.z);
            // rotar los puntos sobre el eje Y
            v0 = v1;
            v1 = ry.mulVector3(v0);
            v2 = v3;
            v3 = ry.mulVector3(v2);
        }
        // rotar los puntos sobre el eje X
        v0 = v2;
        v1 = ry.mulVector3(v0);
        v2 = rx.mulVector3(v0);
        v3 = ry.mulVector3(v2);
    }

    return new Float32Array(positions);
}
```

Figura 14: Código para genera una esfera UV. Fuente: elaboración propia

Aprovechando que calculamos todos los vértices con la esfera centrada en el origen, cada punto representa también el vector normal aparte de la posición. Entonces, con el mismo algoritmo, podemos generar los datos de las posiciones y los de las normales.

7.4 Ejemplos de uso

En la Figura 15 se muestra un ejemplo muy básico de cómo utilizar el *framework* 3D para generar una escena, añadir objetos y pintarla. Como se ve, necesitaremos también de *HTMLCanvas*, que será donde dibujaremos la escena.

```
const canvas = document.querySelector("canvas");
// crear objeto y moverlo
const cube = new Object3D(new BoxGeometry());
cube.transform = Matrix4x4.translation(0, 0, -2);
// crear renderizador de escenas y añadir objetos
const renderer = new SceneRenderer(canvas, {autoClear: true});
renderer.addObject(cube);
// crear cámara
const camera = new PerspectiveCamera();

// renderizar escena
const animate = () => {
    requestAnimationFrame(animate);
    renderer.render(camera);
}
animate();
```

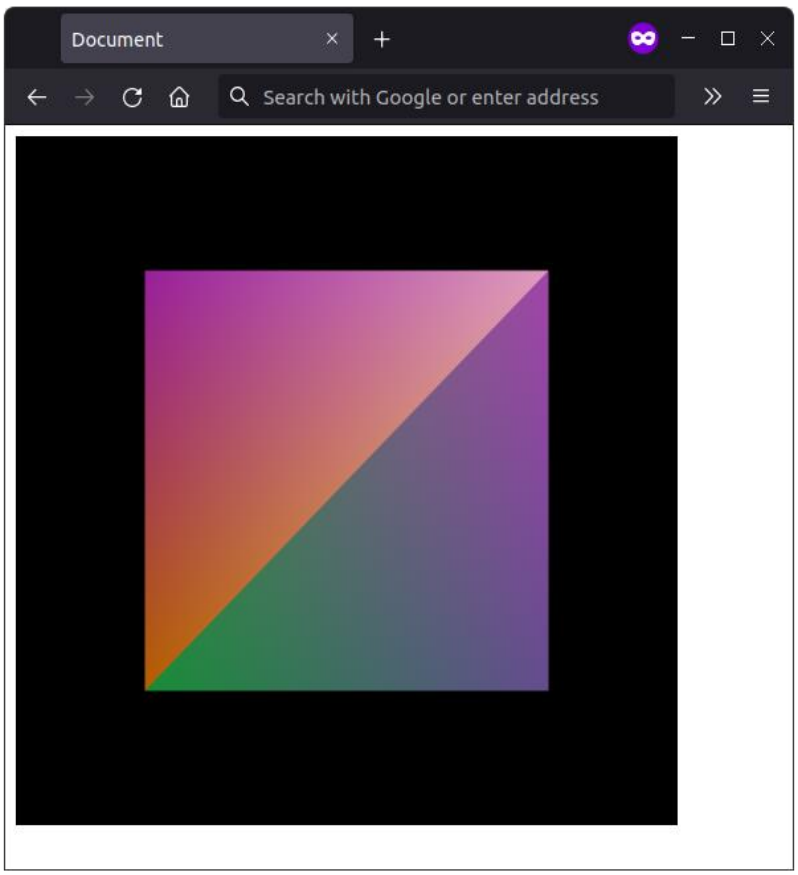

Figura 15: Ejemplo de uso del framework 3D. Código (arriba) y resultado (abajo). Fuente: elaboración propia

8 Componentes

En este apartado, explicaremos el desarrollo y funcionalidad de los componentes desarrollados. Nos centramos en exponer el objetivo y las funcionalidades de cada uno de los web *applets*, así como las características más relevantes de su implementación.

8.1 Características generales de la implementación

Existen muchas maneras de crear pequeños programas gráficos que se puedan cargar en una página web. Nosotros hemos escogido la opción de desarrollarlos como elementos HTML personalizados. Gracias a esto, contamos con unas ventajas que nos facilitan la integración en páginas web.

Fácil inserción

Para añadir un componente a una web, tan solo se debe importar el script y escribir la etiqueta correspondiente en la parte del documento HTML en la que se desea que aparezca. También se puede introducir a partir de código JavaScript, pero esta opción queda relegada a un segundo plano.

Estilo aislado

Este método nos permite tener un estilo propio de los componentes. Un estilo que no pueda ser modificado desde la hoja de estilo que se aplique a la web que contenga el componente. Esto es especialmente útil al evitar que el código CSS general nos pueda afectar a la disposición interna de los elementos de nuestros componentes, causada por un involuntario conflicto entre los nombres de variables.

Reusabilidad

Los componentes web han sido creados con la idea de ser reutilizados en distintas partes sin provocar problema de colisiones de código. Esto nos favorece al poder reutilizar código en partes de la interfaz que se repiten. Se crean estas partes comunes como bloques básicos de un elemento mayor.

8.2 Modelos de color

Existen varios modelos matemáticos de representar un color. Estas maneras se les conoce como modelos de color. Generalmente, se presentan como una tupla de números, donde cada componente describe una de las características del color dentro de un modelo.

En las asignaturas de gráficos de la facultad se explican tres modelos de color, el RGB, el HSB y el CMYK. Estos modelos son los que se utilizan para definir colores en la programación en las asignaturas de gráficos.

A continuación, se explican los componentes creados para cubrir algunos de los conceptos que pueden resultar más difíciles sobre los modelos de color.

8.2.1 Parte común: panel de selección de color

8.2.1.1 Objetivo

Este panel tiene la finalidad de permitirnos seleccionar un color. Cuenta con controles para especificar un color en cualquiera de los tres modelos de color que nos enseñan en la facultad.

8.2.1.2 Funcionalidades

Figura 16: Esquema panel de selección de color. Fuente: elaboración propia

1. Pestaña del panel RGB. Activa el panel para seleccionar un color utilizando el modelo de color RGB.

4. Campo para el componente Red (rojo) del modelo RGB, en el rango de 0 a 255.

5. Campo para el componente Green (verde) del modelo RGB, en el rango de 0 a 255.

6. Campo para el componente Blue (azul) del modelo RGB, en el rango de 0 a 255.

2. Pestaña del panel HSB. Activa el panel para seleccionar un color utilizando el modelo de color HSB.

7. Campo para el componente Hue (tono) del modelo HSB, en el rango de 0 a 359.

8. Campo para el componente Saturation (saturación) del modelo HSB, en el rango de 0 a 100.

9. Campo para el componente Brightness (brillo) del modelo HSB, en el rango de 0 a 100.

3. Pestaña del color CMYK. Activa el panel para seleccionar un color utilizando el modelo de color CMYK.

10. Campo para el componente Cyan (cian) del modelo CMYK, en el rango de 0 a 100.

11. Campo para el componente Magenta (magenta) del modelo CMYK, en el rango de 0 a 100.

12. Campo para el componente Yellow (amarillo) del modelo CMYK, en el rango de 0 a 100.

13. Campo para el componente blacK (negro) del modelo CMYK, en el rango de 0 a 100.

4-13. Cada campo de componente de los modelos está compuesto por una etiqueta, una entrada de texto y un control deslizante. La entrada de texto y el control deslizante se sincronizan entre sí, esto significa que uno se actualiza si el otro cambia.

1-3. Al cambiar el color de una de las pestañas de un modelo de color se actualizan también las otras pestañas. Las pestañas de los modelos HSB y CMYK solo se actualizarán si el color es uno diferente del que ya está, esto es porque existe más de una manera de representar un mismo color con estos modelos y se quiere representar este hecho.

8.2.2 Acierta el color

8.2.2.1 Objetivo

El objetivo de este componente es ayudar a comprender el significado de cada uno de los parámetros de los modelos de color que se enseñan en la facultad. De aquí surge la idea de crear un pequeño programa en el que el alumno pueda "jugar" a acertar un color generado de manera aleatoria. De esta manera, para poder hallar el color correcto le será necesario al estudiante entender cada uno de los parámetros y los efectos que tiene el cambiarlos.

8.2.2.2 Funcionalidades

Figura 17: Esquema del componente de acierta el color. Fuente: elaboración propia

1. Panel del color aleatorio. Aquí se muestra el color aleatorio, que es el objetivo al que nos tenemos que aproximar. Este panel tiene una etiqueta explicativa que indica su utilidad. El color del texto de la etiqueta cambia de color, entre negro y blanco, dependiendo del color del fondo para que haya el contraste suficiente para poder leerlo con facilidad.

2. Panel del color seleccionado. Aquí se muestra el color seleccionado con los controles que se encuentran en la parte inferior del componente. De la misma manera que el panel del color aleatorio, este tiene una etiqueta explicativa cuyo texto cambia de color dependiendo del mostrado en del fondo.

3. Panel de selección de color. Aquí se encuentran los controles para seleccionar el color que se presenta en el panel de color seleccionado. Hay tres pestañas que sirven para elegir entre los modelos de color RGB, HSB y CMYK.

4. Botón de nuevo color aleatorio. Genera un nuevo color aleatorio para el panel del color aleatorio.

5. Botón para comparar colores. Muestra una alerta formada por la dirección de la web, el color aleatorio en codificación RGB y la similitud entre el color aleatorio y el color seleccionado. La similitud se calcula como la distancia entre los colores en el espacio definido por el modelo RGB, normalizado para que el valor este entre el 0 y el 100, siendo 0 cuando son totalmente distintos y 100 cuando son completamente iguales.

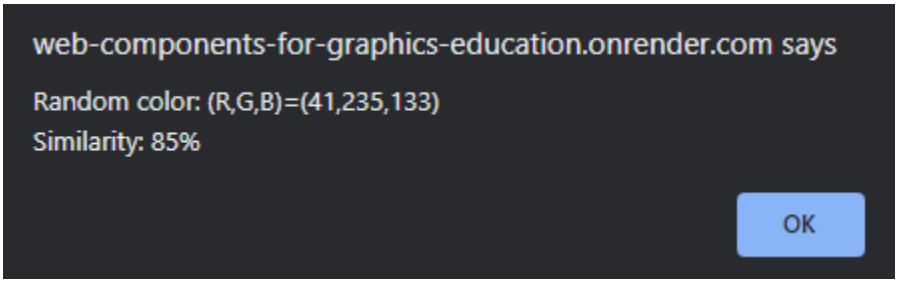

Figura 18: Alerta mostrada al clicar el botón de comparar colores. Fuente: elaboración propia

8.2.3 Comparación de modelos de color

8.2.3.1 Objetivo

Este componente también tiene la intención de facilitar el entendimiento de los modelos de color, pero desde otro enfoque más dirigido a observar cómo varía los valores de un modelo cuando cambiamos los parámetros de otro. Así que se muestran dos modelos, a seleccionar por el usuario, sincronizados para comparar los valores.

8.2.3.2 Funcionalidades

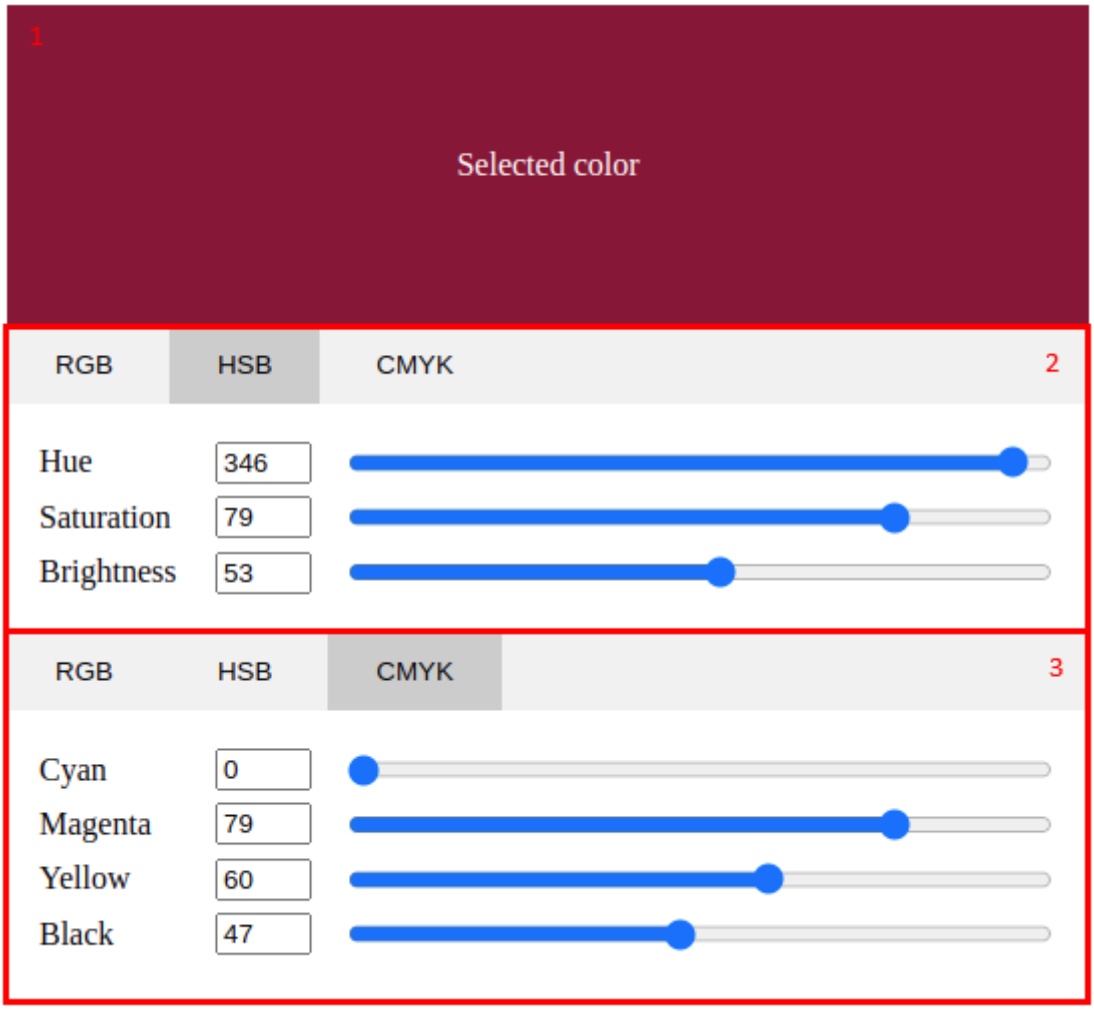

Figura 19: Esquema del componente de comparación de modelos de color. Fuente: elaboración propia

1. Panel del color seleccionado. Muestra el color ingresado por los paneles de selección de abajo. El texto de este también cambia para mejorar la legibilidad.

2-3. Paneles de selección de color. Permiten seleccionar el color a mostrar en el panel superior. Estos están sincronizados, así que si cambias uno se modifica el otro. De la misma manera que dentro de un mismo panel de selección de color, estos solo se afectan entre ellos si el color representado es distinto. En la Figura 20 se muestra como hay dos maneras de representar un mismo color, el negro, dentro de un modelo.

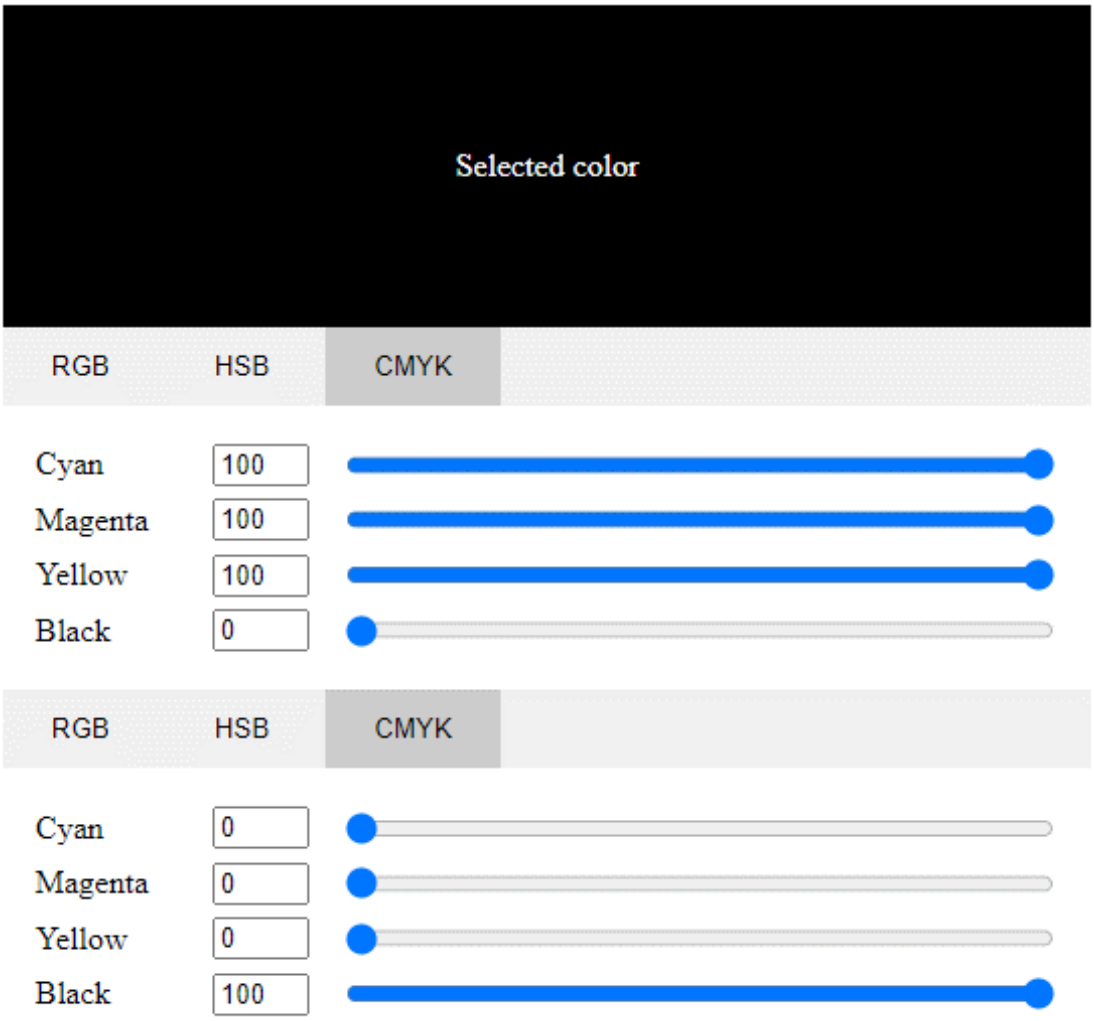

Figura 20: Ejemplo de dos configuraciones de negro utilizando el modelo CMYK. Fuente: elaboración propia

8.3 Transformaciones geométricas

Las transformaciones geométricas nos permiten mover y cambiar de forma la geometría de un modelo. En computación gráfica se utilizan matrices para realizar una transformación. Para ejecutar una transformación, estas matrices se multiplican por la posición de los vértices.

La utilidad de las transformaciones geométricas recae en que nos permite usar un mismo modelo para dibujar un objeto en diferentes estados. De esta manera nos ahorramos memoria al solo necesitar un modelo y no incontables para representar los diferentes estados posibles.

Existen varias transformaciones geométricas, pero las que utilizamos en las asignaturas de gráficos, para los modelos, utilizamos las de traslación, rotación y escalado.

- La transformación de traslación nos permite mover un punto en el espacio una cantidad determinada.
- La transformación de rotación nos permite girar respecto a un vector. En un espacio 3D, este vector típicamente es uno de los que forman la base del sistema de coordenadas, pero puede ser cualquier otro.
- La transformación de escalado nos permite aumentar o disminuir el tamaño de un modelo.

A continuación, se describe el componente creado para facilitar el entendimiento de este concepto.

8.3.1 Transformaciones geométricas sobre un cubo

8.3.1.1 Objetivo

Este componente tiene la finalidad de contribuir al entendimiento de las distintas transformaciones geométricas y el resultado de su aplicación a un modelo 3D. Nos queremos centrar principalmente en dos hechos. El primero es mostrar que el orden en el que se decide aplicar las distintas transformaciones es importante, pues la multiplicación de matrices no es conmutativa. El segundo es entender los distintos parámetros de cada transformación, con especial énfasis en la de rotación y su giro respecto a un vector cualquiera. Para esto se da una total libertad en cuanto a las transformaciones que se aplican al objeto, pudiendo añadir, reordenar y eliminar las transformaciones, además de modificar los parámetros de cada una de estas.

8.3.1.2 Funcionalidad

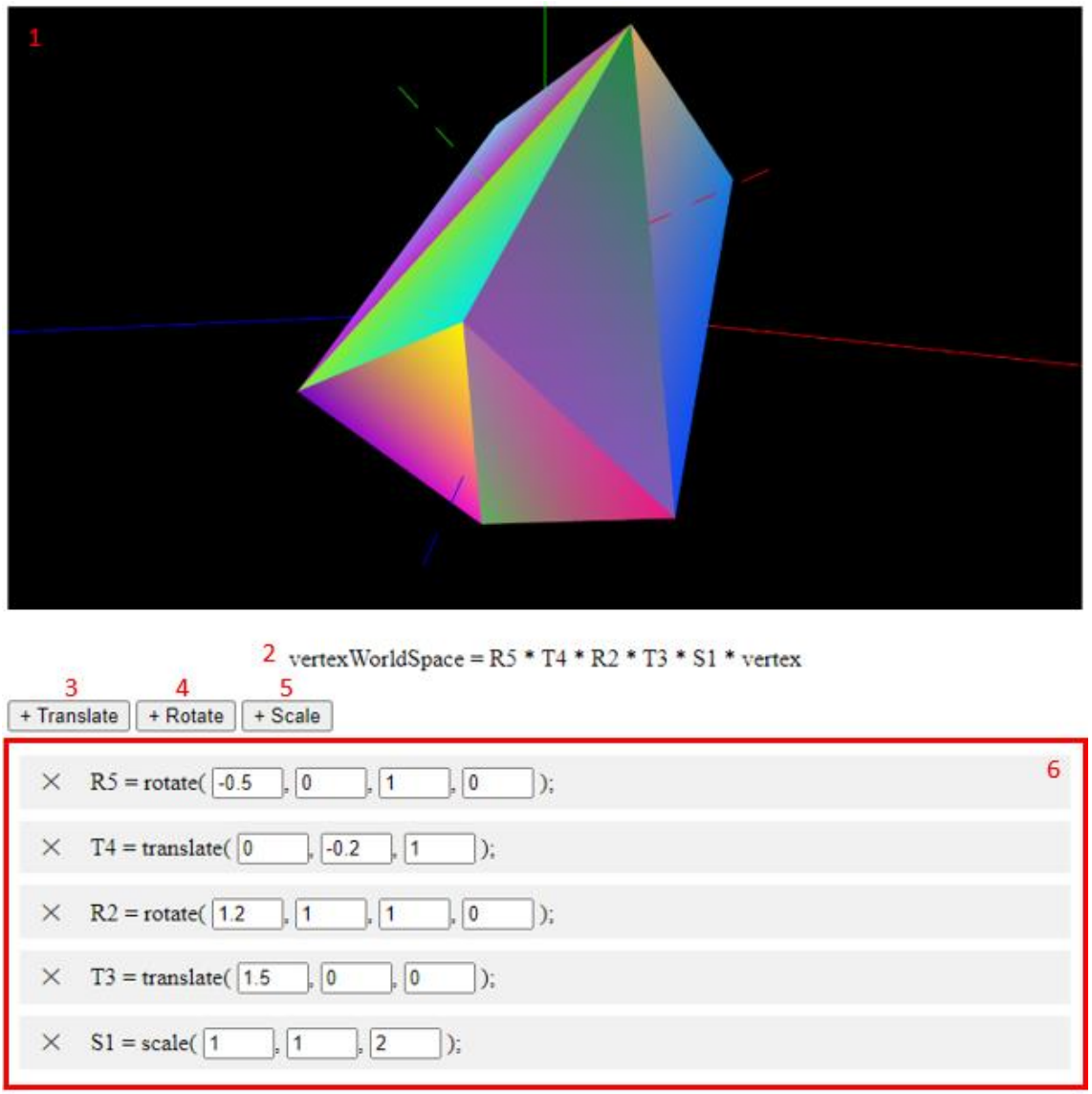

Figura 21: Esquema del componente de transformaciones geométricas. Fuente: elaboración propia

1. Vista escena. Aquí se muestra la escena formada por simplemente un cubo, que es el modelo al que se le aplicaran las transformaciones geométricas. Para tener un punto de referencia y poder ubicarnos mejor, se dibujan los ejes de coordenadas del mundo. Además, se pintan los ejes del objeto para tener una mayor claridad en el seguimiento de las transformaciones aplicadas. Se puede navegar en la escena utilizando el ratón o los gestos táctiles.

2. Fórmula. Indica el orden en el que se aplican las transformaciones para calcular la posición de un vértice del objeto en el mundo. Cada transformación está representada por su identificador.

3. Botón traslación. Añade una nueva transformación de traslación al inicio del panel de transformaciones geométricas.

4. Botón rotación. Añade una nueva transformación de rotación al inicio del panel de transformaciones geométricas.

5. Botón escalado. Añade una nueva transformación de escalado al inicio del panel de transformaciones geométricas.

6. Panel de transformaciones geométricas. Aquí se encuentran las transformaciones que se aplican al modelo. El orden en el que aparecen indica el orden en el que se aplican, siendo la de más abajo la primera en aplicarse. Estas se pueden cambiar de orden arrastrando con el ratón o con el dedo en paneles táctiles.

Figura 22: Esquema de una transformación de traslación. Fuente: elaboración propia

Figura 23: Esquema de una transformación de rotación. Fuente: elaboración propia

Figura 24: Esquema de una transformación de escalado. Fuente: elaboración propia

7. Botón eliminar transformación. Elimina la transformación del panel de transformaciones.

8. Identificador de transformación. Código único que identifica una transformación.

9. Transformación de traslación. Contiene los campos que representan los parámetros de una de traslación.

10. Cantidad a desplazar en el eje x.

11. Cantidad a desplazar en el eje y.

12. Cantidad a desplazar en el eje z.

13. Transformación de rotación. Contiene los campos que representan los parámetros de una rotación.

14. Angulo de rotación en radianes.

15. Componente x del vector respecto al cual se hace la rotación.

16. Componente y del vector respecto al cual se hace la rotación.

17. Componente z del vector respecto al cual se hace la rotación.

18. Transformación de escalado. Contiene los campos que representan los parámetros de un escalado.

19. Cantidad a escalar en el eje x.

20. Cantidad a escalar en el eje y.

21. Cantidad a escalar en el eje z.

8.4 Cámara

Para representar una cámara utilizamos dos matrices, la *view matrix* y la *proyection matrix*. La *view matrix* nos permite decidir desde donde y en qué dirección observamos la escena. La *proyection matrix* define la manera en la que se ve la escena y los planos de recortado. Estos planos determinan cuanto de la escena se ve.

Las *projection matrix* más comúnmente utilizadas son las de proyección perspectiva y las de proyección ortogonal. La proyección perspectiva es la que simula la manera en la que el ojo humano ve, y la proyección ortogonal es aquella en la que todos los objetos mantienen el tamaño sin importar la distancia a la que se encuentren.

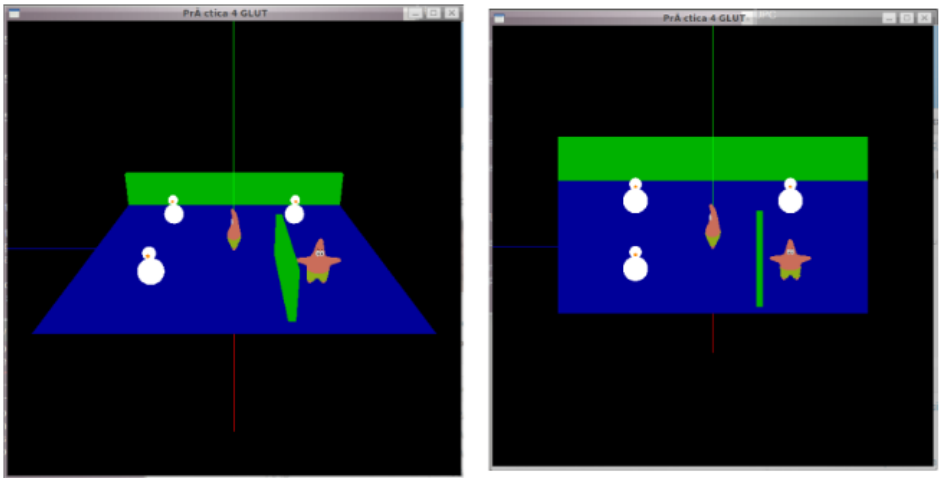

Figura 25: Ejemplo tipos de proyección. Perspectiva (izquierda) y ortogonal (derecha). Fuente: diapositivas cámara IDI [1]

Existen distintos métodos para generar la *view matrix*. Los que se enseñan en las asignaturas de gráficos de la facultad son el de *LookAt* y el de ángulos de Euler. El método de *LookAt* consiste un ubicar la cámara a partir de la posición esta (OBS), el objetivo al que apunta (VRP), y el vector que indica la dirección de arriba para la cámara. El método de los ángulos de Euler utiliza el objetivo (VRP), la distancia (d) a la que se encuentra la cámara (OBS), y los ángulos de rotación (φ, θ y ψ).

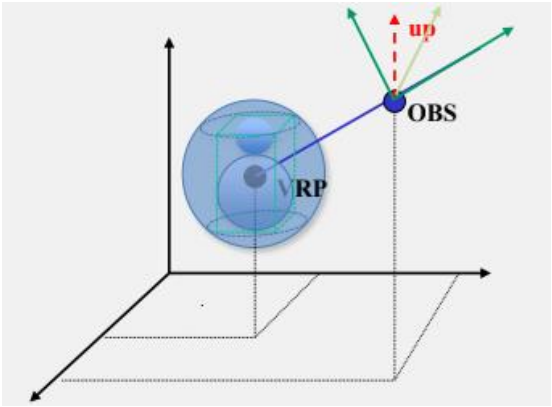

Figura 26: Esquema método LookAt. Fuente: diapositivas cámara IDI [1]

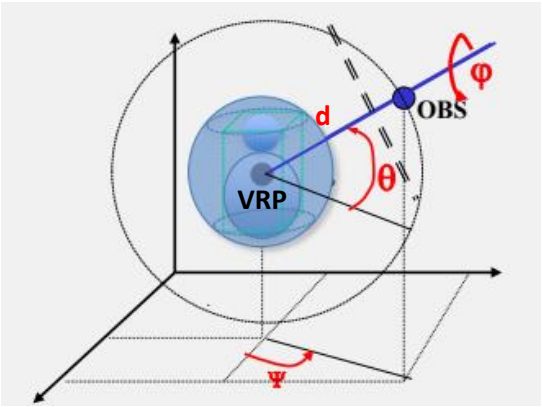

Figura 27: Esquema método ángulos Euler. Fuente: diapositivas cámara IDI [1]

A continuación, se explican el componente desarrollado para cubrir el tema de la cámara.

8.4.1 Tipos de cámara

8.4.1.1 Objetivo

El propósito de este componente es ayudar a comprender como se define una cámara para gráficos por ordenador. Entender los diferentes tipos de proyección y los diferentes métodos para posicionarla y direccionarla que se nos enseñan en la facultad. Para esto podremos modificar los parámetros de las funciones que comúnmente se utilizan para determinar una cámara y ver el resultado en su proyección y en un modelo que representa a la cámara.

8.4.1.2 Funcionalidades

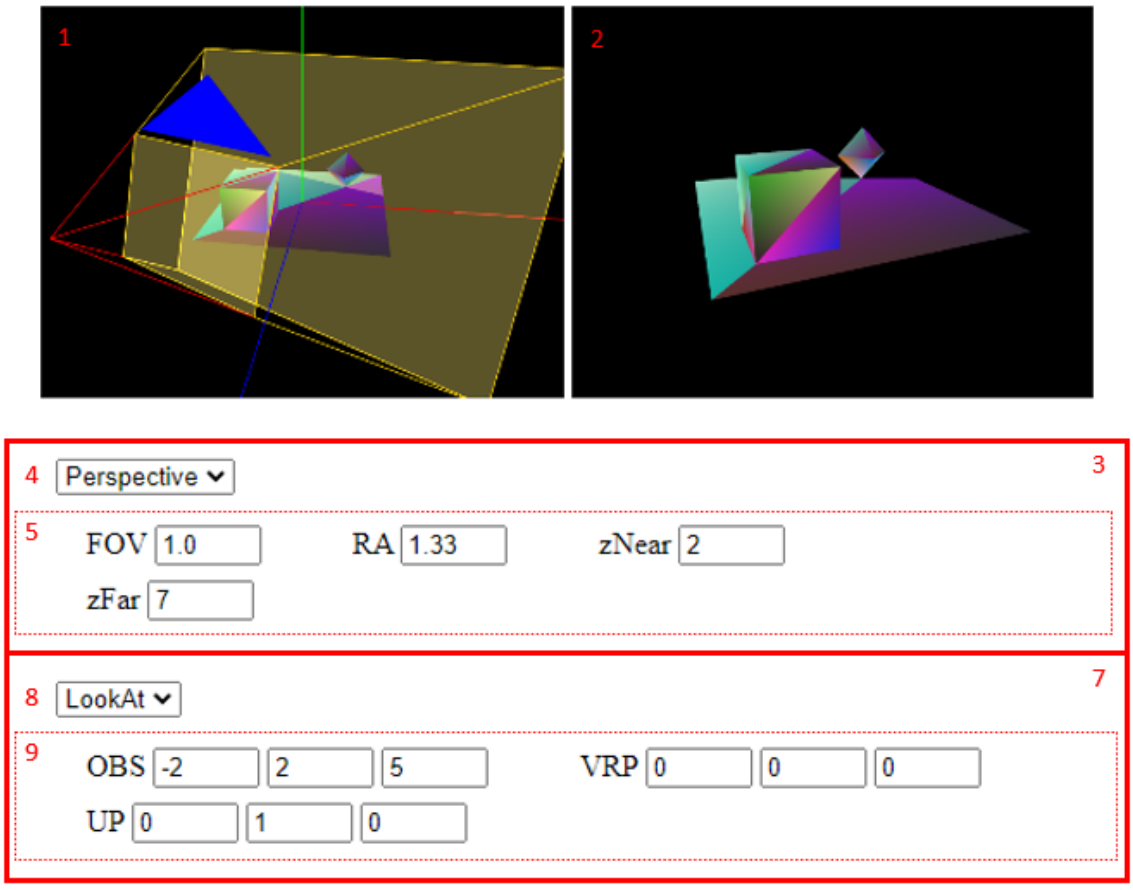

Figura 28: Esquema del componente de tipos de cámara. Fuente: elaboración propia

1. Vista escena. Muestra la escena formada por un cubo y un octaedro sobre un plano, y la cámara desde que se observa la escena. Esta cámara está formada por tres polígonos, una pirámide roja, un triángulo azul y un *frustum naranja*. La pirámide tiene la punta ubicada en la posición de la cámara. El triángulo representa el vector *UP* de la cámara. El *frustum* encierra la región que se dibuja en la vista cámara. Además, también se dibujan los ejes de coordenadas del mundo. Esta escena es navegable con el ratón y los gestos táctiles.

2. Vista cámara. Muestra la escena del plano, cubo y octaedro desde el punto de vista de la cámara.

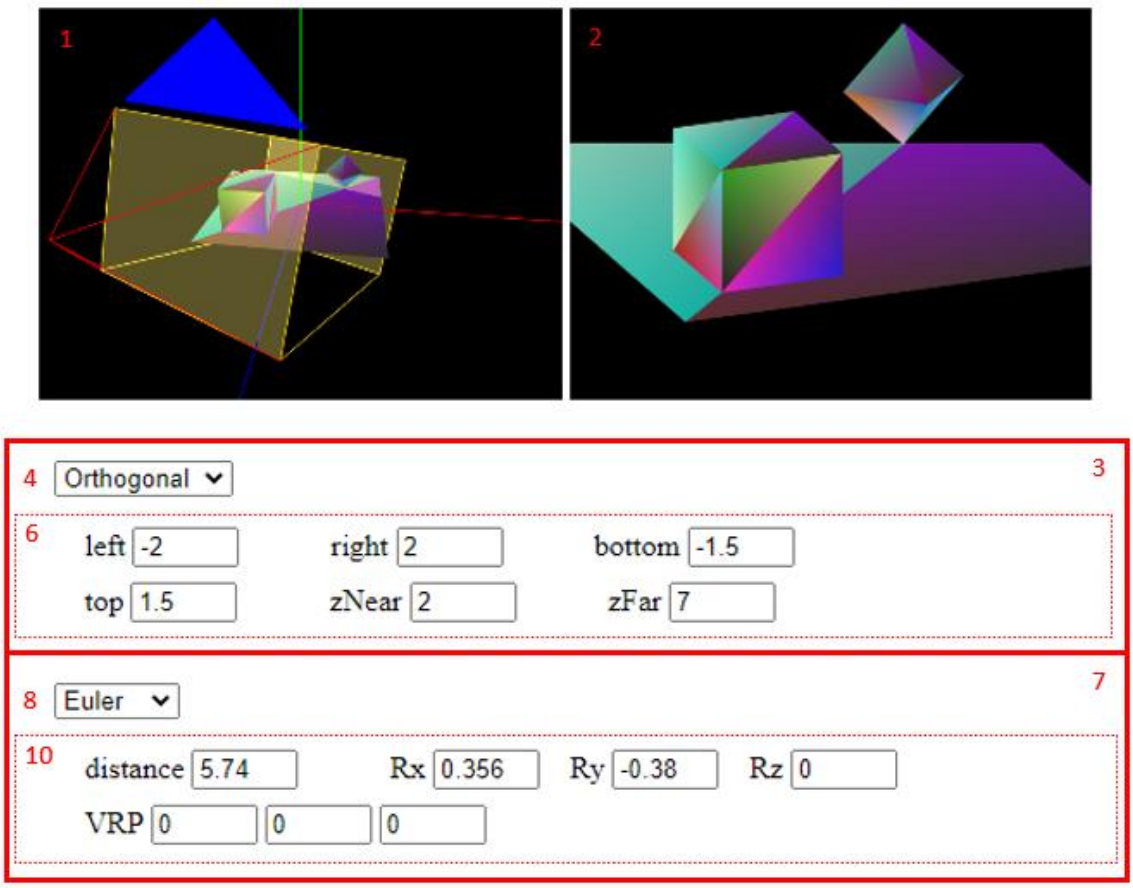

Figura 29: Esquema del componente de tipos de cámara. Fuente: elaboración propia

3. Panel *projection matrix*. Permite definir la matriz de proyección de la cámara.

4. Selector tipo *projection matrix*. Cambia el formulario mostrado dejándote escoger entre proyección perspectiva y ortogonal.

5. Formulario proyección perspectiva. Especifica los parámetros de una proyección perspectiva.

FOV: campo de visión (*field of view*) de la proyección perspectiva.

RA: relación de aspecto (*aspecto ratio*) de la proyección perspectiva.

zNear: distancia a la que se encuentra el plano más cercano de recortado.

zFar: distancia a la que se encuentra el plano más lejano de recortado.

6. Formulario proyección ortogonal. Especifica los parámetros de una proyección ortogonal.

Left: define el plano de recortado de la izquierda.

Right: define el plano de recortado de la derecha.

Bottom: define el plano de recortado de abajo.

Top: define el plano de recortado de arriba.

zNear: distancia a la que se encuentra el plano más cercano de recortado.

zFar: distancia a la que se encuentra el plano más lejano de recortado.

7. Panel view matrix. Sirve para especificar la view matrix de la cámara, que define la posición, dirección y sentido de esta.

8. Selector tipo view matrix. Cambia el formulario mostrado dejándote escoger entre el método de *look at* o el de ángulos de Euler.

9. Formulario LookAt. Permite definir la view matrix de la cámara con la matriz de *look at*.

OBS: punto donde se encuentra la cámara.

VRP: punto hacia donde mira la cámara (view reference point)

UP: vector que indica la parte de arriba de la cámara.

10. Formulario Euler. Permite definir la view matrix de la cámara con los ángulos de Euler.

Distance: distancia a la que se encuentra la cámara del punto de referencia (VRP).

Rx: ángulo de rotación en el eje x, en radianes.

Ry: ángulo de rotación en el eje y, en radianes.

Rz: ángulo de rotación en el eje z, en radianes.

VRP: punto hacia donde mira la cámara (view reference point).

8.4.1.3 Dibujado cámara

Para este componente es necesario poder representar lo que ve una cámara en la escena. De esta manera podemos ver con mayor facilidad en que afectan los cambios que se hacen a sus distintos parámetros.

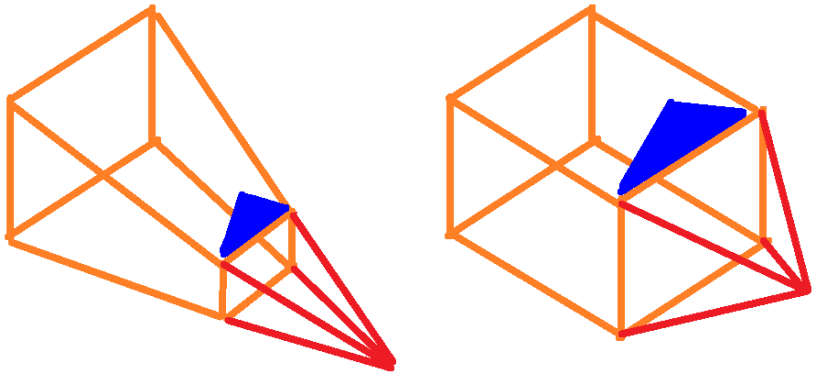

Figura 30: Esquema representación cámara. Proyección perspectiva (izquierda) y proyección ortogonal (derecha). Fuente: elaboración propia

Para nosotros, una cámara estará formado por las siguientes tres geometrías.

- *Frustum* u ortoedro. Dependiendo de si es una proyección perspectiva (*frustum*) u ortogonal (ortoedro). Encierra la región que será dibujada. Esta pintada de color naranja en la Figura 30.
- Pirámide. Tiene la punta ubicada en la posición *view matrix* de la cámara. La base coincide con el plano más cercano de recortado. Pintado de rojo en la Figura 30.

- Triángulo. Representa el vector up de la cámara, que indica lo que está arriba para su sistema de coordenadas local. Pintado de azul en la Figura 30.

Además de dibujar la geometría básica mostrada en la Figura 30, también se quiere pintar de manera semitransparente los planos de recortado más lejano y cercano. Esto es para poder ver con mayor claridad donde intersecan estos planos con los objetos y entender el motivo por el cual hay partes de la escena que no se dibujan.

Con esta finalidad, se ha desarrollado un módulo encargado de pintar una cámara. Este está adaptado para con una misma función poder dibujar tanto una cámara perspectiva como una cámara ortogonal. El algoritmo utilizado se divide en cuatro pasos.

1. Dibujar el *frustum* u ortoedro

Este paso es tan sencillo como simplemente dibujar un cubo de 2x2 centrado en el origen y utilizar la inversa de la *view matrix* y *projection matrix* como matriz de transformación de modelo. Nos aprovechamos de que estas matrices representan la proyección y la posición de la cámara.

2. Dibujar el vector up

De igual manera que en el paso anterior, utilizamos un modelo de un triángulo ubicado de manera inteligente para que al multiplicarlo por la inversa de la *view matrix* y *projection matrix* quede en la posición deseada. Esta ubicación inteligente de la que hablamos sería encima del plano delantero del cubo del aparatado anterior.

3. Dibujar la pirámide

Este paso resulta más complicado que los anteriores. Esta vez utilizamos la inversa de la *porjection matrix* para obtener los cuatro puntos del plano de recortado más cercano a la cámara. A partir de estos puntos, actualizaremos las posiciones de los vértices de la base de la pirámide. Con la geometría ya actualizada, únicamente queda utilizar la inversa de la *view matrix* como matriz de transformación para colocar el modelo en la posición y dirección correcta.

Como se muestra en la Figura 31, con este método podemos dibujar cámaras asimétricas, es decir, que no tienen la punta de la pirámide alineada con el centro de plano más cercano.

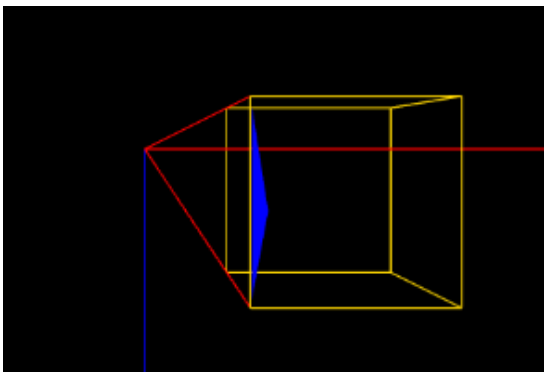

Figura 31: Cámara ortogonal asimétrica. Fuente: elaboración propia

4. Dibujar planos de recortado

Para dibujar los planos utilizamos el mismo truco que en los pasos 1 y 2. Sin embargo, este paso representa un desafío algo mayor. Como los planos tienen que dibujarse con un color

semitransparente y hacer uso del canal alfa, se tienen que pintar en un cierto orden. Para garantizar un resultado correcto, primero se dibuja el plano más lejano a la cámara, y luego el más cercano.

La razón de tener que pintar en un orden determinado es el riesgo de no pintar correctamente zonas de intersección entre dos objetos semitransparentes. Si se pinta primero el objeto cercano, y luego el objeto lejano, y resulta que tienen fragmentos que caen en el mismo píxel, debido a la distancia mayor del segundo, su fragmento se descartará por el *depth test*. Este es lo que ocurre en el ejemplo de la Figura 32, donde no se pinta el plano más cercano en la imagen de la izquierda.

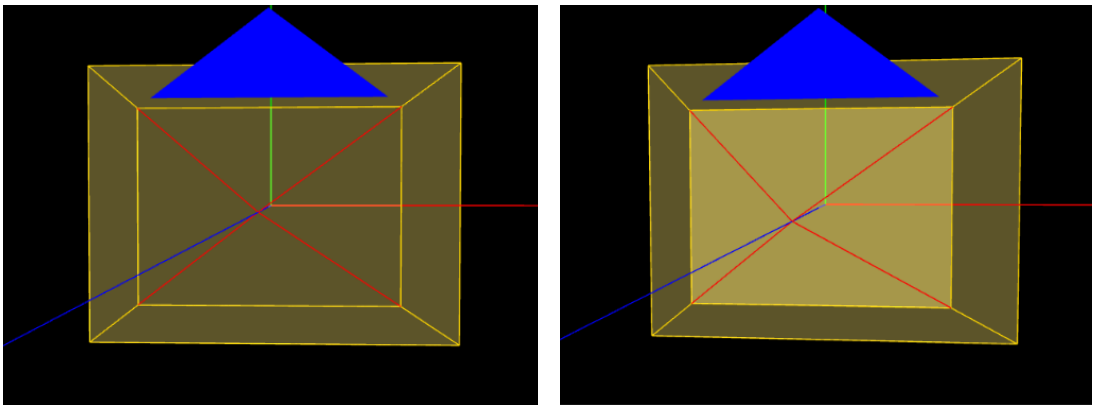

Figura 32: Error al pintar desordenados fragmentos semitransparentes. Resultado con el error a la izquierda y resultado correcto a la derecha. Fuente: elaboración propia

Del caso anterior se nos podría venir la idea de pintar las zonas semitransparentes desactivando el z-buffer. Pero esto tampoco sería una solución. Al desactivar el *depth test*, se producen inconsistencias en el color con el resto de los objetos. Como se ve en Figura 33, se produce el error de pintar el plano semitransparente más lejano sobre el cubo o el triángulo azul, pese a que se encuentra más atrás.

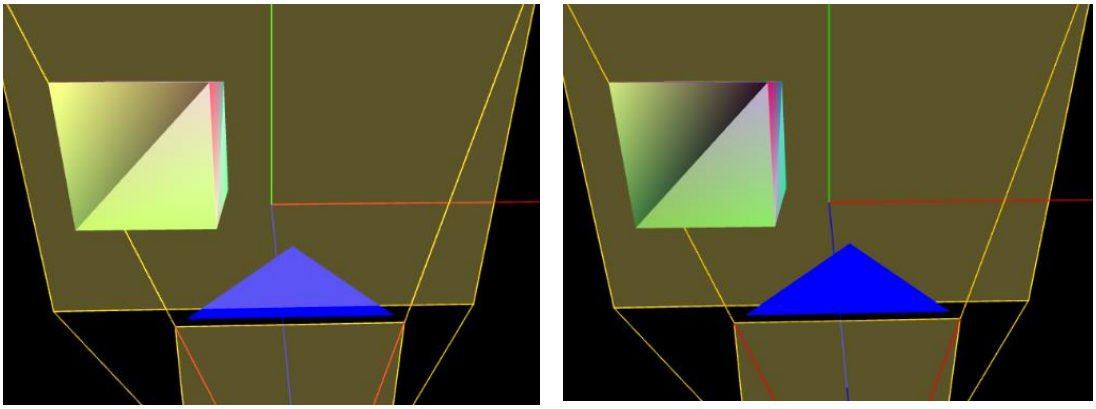

Figura 33: Error al pintar fragmentos semitransparentes desactivando el z-buffer. Resultado con el error a la izquierda y resultado correcto a la derecha. Fuente: elaboración propia

Por tanto, la manera correcta de pintar estos planos semitransparentes es dibujarlos en orden y sin desactivar el *depth test*. Primero ir a por el plano que está más lejos del observador, y luego por el que está más cerca.

8.5 Iluminación

La iluminación es un apartado importante en la simulación de escenas, pues contribuye al realismo de estas. A continuación, se describen los componentes creados para tratar la cuestión de la iluminación en gráficos por ordenador.

8.5.1 Efectos de focos RGB sobre una esfera

8.5.1.1 Objetivo

El objetivo de este componente es ver como se afectan tres focos (rojo, verde y azul) sobre un objeto mate y el resultado de las mezclas de las luces. Para esto, se nos permitirá cambiar la intensidad de manera independiente de cada foco para probar diferentes configuraciones.

8.5.1.2 Funcionalidades

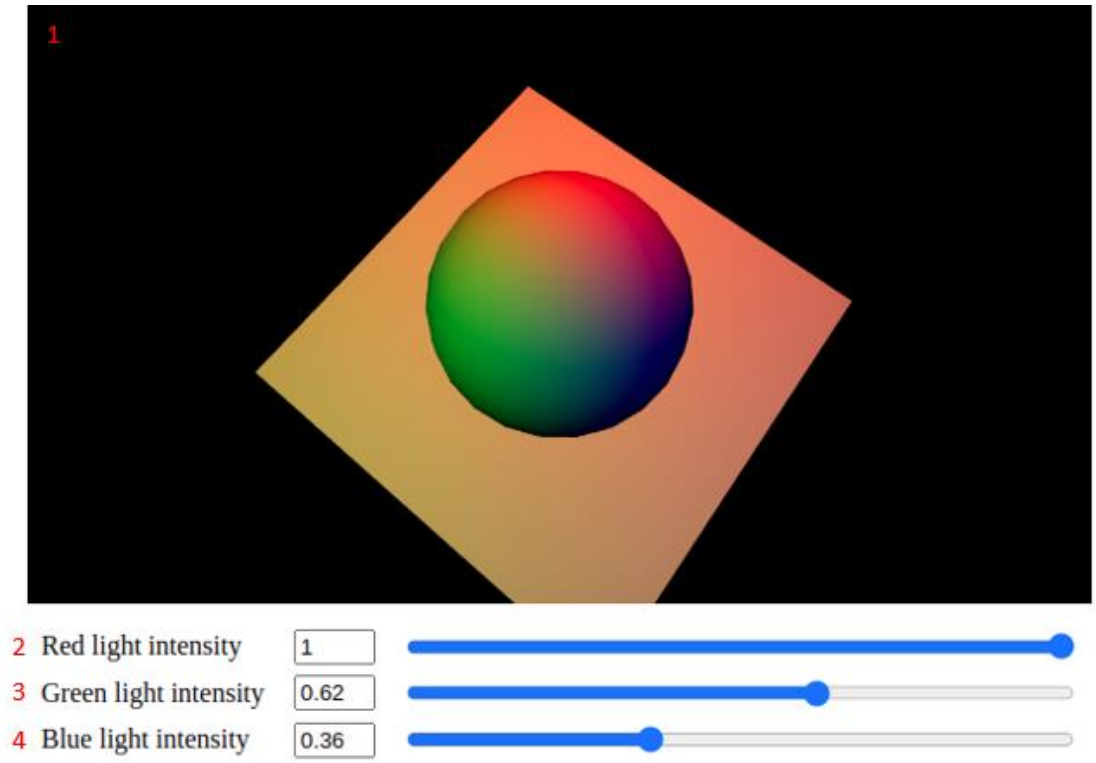

Figura 34: Esquema del componente de efectos de focos RGB sobre una esfera. Fuente: elaboración propia

1. Vista escena. Se muestra la escena formada por una esfera sobre un plano, los dos objetos de color mate, y tres focos de luz situados alrededor de la esfera. Los focos son de color rojo, verde y azul, correspondientes a cada uno de los componentes del modelo RGB. Esta escena también incluye navegación con ratón y gestos táctiles.

2. Campo intensidad luz roja. Indica la intensidad del foco de color rojo, en el rango de 0 a 1.

3. Campo intensidad luz verde. Indica la intensidad del foco de color verde, en el rango de 0 a 1.

4. Campo intensidad luz azul. Indica la intensidad del foco de color azul, en el rango de 0 a 1.

2-4. La entrada de texto y el control deslizante de cada uno de estos campos están sincronizados y muestran los cambios efectuados uno del otro.

8.5.1.3 *Shader* para el modelo de iluminación de los tres focos

Para este componente es necesario simular la iluminación producida por tres focos puntuales sobre una esfera. Como la esfera está hecha de un material mate, podemos imaginar que cualquier punto de dicha esfera irradia la misma cantidad de luz en todas las direcciones. Por tanto, su color no depende de la dirección de visión.

Tras haber analizado los requisitos del sistema de iluminación, vemos que con utilizar el modelo de iluminación de Lambert es suficiente para la tarea. Este modelo solo tiene en cuenta los componentes ambiente y difuso.

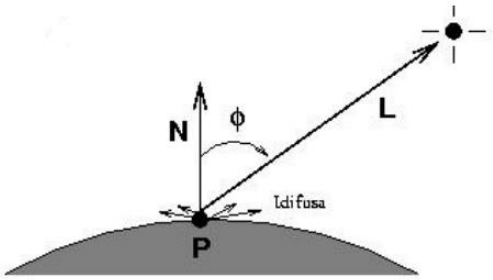

Figura 35: Esquema modelo de iluminación de Lambert. Fuente: Diapositivas realismo IDI [1]

La fórmula para calcular el color de un punto con el uso de este modelo es la siguiente.

$$color\ ambiente = luz\ ambiente * coeficiente\ reflexión\ ambiente$$

$$color\ difuso = luz\ difusa * coeficiente\ reflexión\ difusa\ material * \cos(\phi)\,,\quad si\ |\phi| < 90°$$

$$color\ Lambert = color\ ambiente + \sum_{\forall\ foco} color\ difuso$$

Ahora bien, en nuestro caso particular, no queremos que la parte del color ambiente contribuya al color final, para así darle una total importancia a la iluminación provocada por los focos. Como en nuestro caso tenemos tres focos, habrá que calcular el color difuso para los tres focos.

```
23  vec3 Lambert (vec4 posFocusSCO, vec3 colFocus) {
24      vec3 NormSCO = normalize(normalSCO);
25      vec3 L = normalize(posFocusSCO.xyz - vertexSCO.xyz);
26
27      vec3 colRes = vec3(0.0);
28
29      if (dot (L, NormSCO) > 0.0) {
30          colRes = colRes + colFocus * matdiffFS * dot (L, NormSCO);
31      }
32      return (colRes);
33  }
```

Figura 36: Función para el modelo de iluminación de Lambert. Fuente: elaboración propia

El fragmento de código de la Figura 36 muestra una función que calcula el color en un punto a partir de la posición de un foco (posFocusSCO) y su color (colFocus). Este código está extraído de los ejercicios realizados por mí para la asignatura de IDI [1], y, por tanto, su correcto funcionamiento ya ha sido comprobado.

8.5.2 Modelo de iluminación de Phong

8.5.2.1 Objetivo

Este componente está diseñado para explicar los parámetros de la luz y el material de un objeto utilizando el modelo de iluminación de Phong. Así que, disponemos un objeto y un foco de los cuales podemos modificar cualquier parámetro, y así ver el resultado que provoca.

8.5.2.2 Funcionalidades

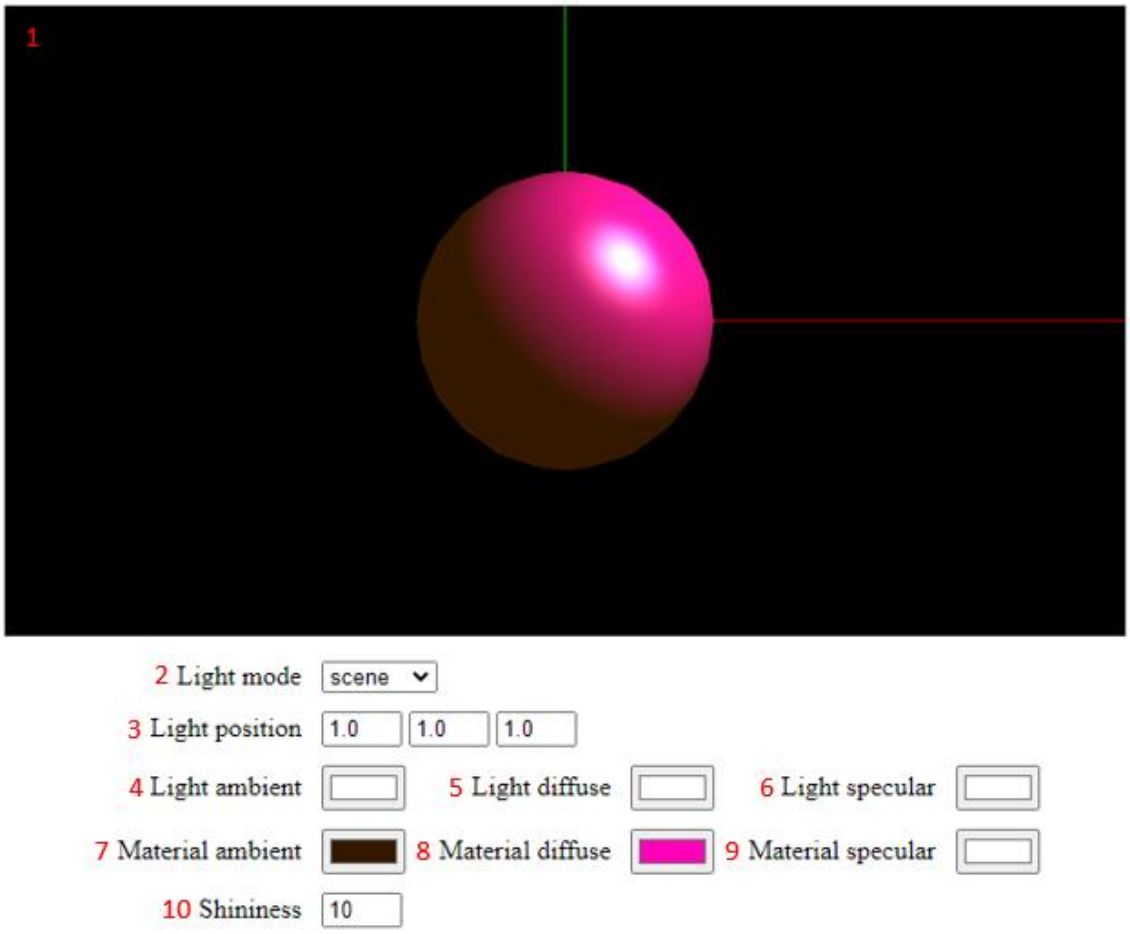

Figura 37: Esquema del componente de Phong shading. Fuente: elaboración propia

1. Vista escena. Aquí se dibuja la escena, formada por solo una esfera y un foco. Además, se pintan los ejes de coordenadas del mundo para tener un punto de referencia con la posición del foco. Al igual que la mayoría de las vistas, esta incluye navegación con ratón y gestos táctiles.

2. Modo de luz. Permite seleccionar entre una luz de cámara o escena. Esto sirve para saber que utilizar como punto origen para la posición de la luz, si el origen de coordenadas del mundo o la posición de la cámara.

3. Posición de la luz. Posición del foco respecto al sistema de coordenadas seleccionado en el modo de luz.

4. Color ambiente de la luz.

5. Color difuso de la luz.

6. Color especular de la luz.

7. Color ambiente del material.

8. Color difuso del material.

9. color especular del material.

10. Factor de brillo del material.

4-9. Para seleccionar el color de cada uno de estos parámetros se utiliza un *HTML input* del tipo color, implementado por cada navegador web.

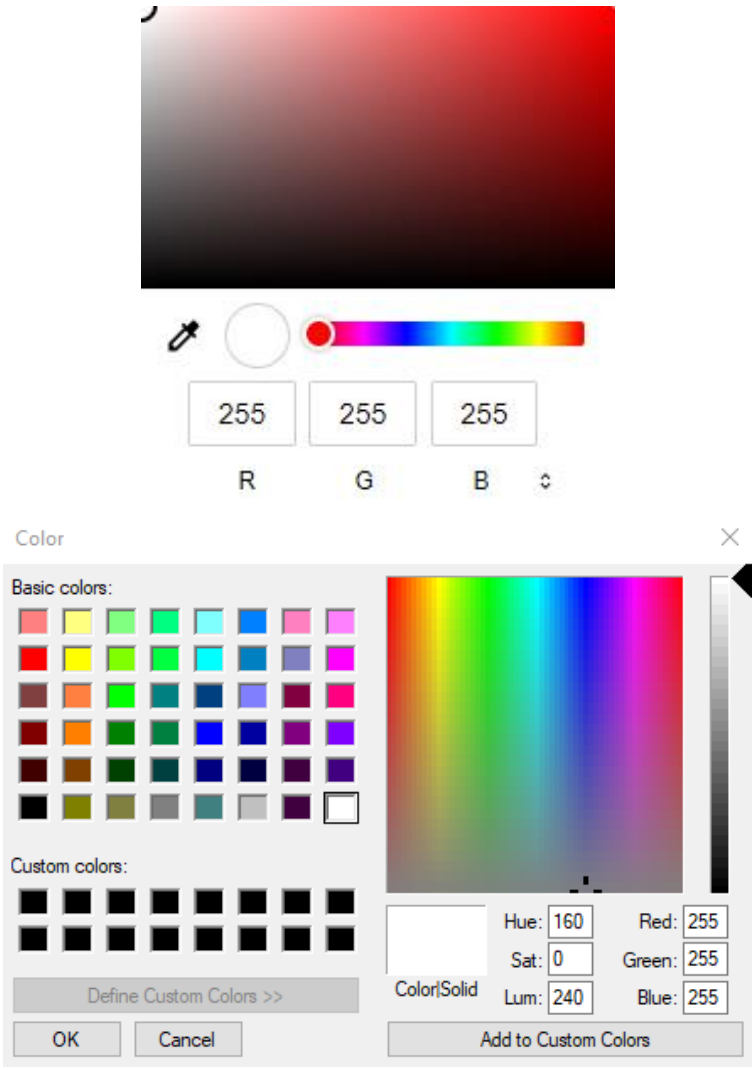

Figura 38: HTML input tipo color. Elemento en Chrome (arriba) y Firefox (abajo). Fuente: elaboración propia

8.5.2.3 *Shader* para el modelo de iluminación de Phong

Como el propio nombre del componente indica, en este programa tenemos que simular la luz utilizando el modelo de iluminación de Phong. Este modelo considera focos de luz puntuales y objetos con reflexión especular. Para este modelo es importante la posición del espectador, ya que dependiendo de donde este, podrá ver o no la reflexión especular. La dirección de esta reflexión es simétrica al vector del punto al foco respecto al vector normal al punto.

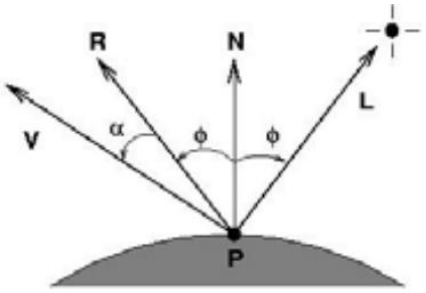

Figura 39: Esquema modelo de iluminación de Lambert. Fuente: Diapositivas realismo IDI [1]

Para calcular el color de un punto con el modelo de Phong se utiliza la siguiente fórmula.

$$color\ especular = luz\ especular * coeficiente\ reflexión\ especular\ material * \cos^{exponente\ reflexion\ especular}(\alpha), si\ |\phi| < 90°$$

$$color\ Phong = color\ ambiente + \sum_{\forall\ foco} color\ difuso + \sum_{\forall\ foco} color\ especular$$

La función que se muestra en la Figura 40 calcula el color con la fórmula de Phong a partir del vector normal (N), el vector foco observador (L), y la posición del punto (V). El método ha sido extraído de un ejercicio realizado por mí mismo para la asignatura de Gráficos [2].

```
19  vec3 phong(vec3 N, vec3 L, vec3 V)
20  {
21      vec3 R = 2.0 * max(0.0, dot(N, L)) * N - L;
22      vec3 phongColor = matAmbient * lightAmbient + matDiffuse * lightDiffuse * max(0.0, dot(N, L));
23      if (dot(N, L) >= 0.0)
24          phongColor += matSpecular * lightSpecular * pow(max(0.0, dot(R, V)), matShininess);
25      return phongColor;
26  }
```

Figura 40: Función para el modelo de iluminación de Phong. Fuente: elaboración propia

9 Diseño adaptable

Hoy en día las páginas web se ven en multitud de dispositivos como móviles, tabletas, portátiles u ordenadores de sobremesa. Además, dentro de cada tipo, los distintos dispositivos presentan características concretas, tales como el tamaño de pantalla y la resolución. Es por esto por lo que se necesitan diseños webs adaptativos, que puedan adaptar su apariencia al dispositivo que se esté utilizando para visitarlas.

Los componentes desarrollados han estado pensados con este propósito. El de poder ser accesibles a la mayor cantidad de la apariencia de dispositivos posibles. Para esto, se ha hecho uso de sentencias que permitan definir reglas para ajustar de manera automática la interfaz de los programas.

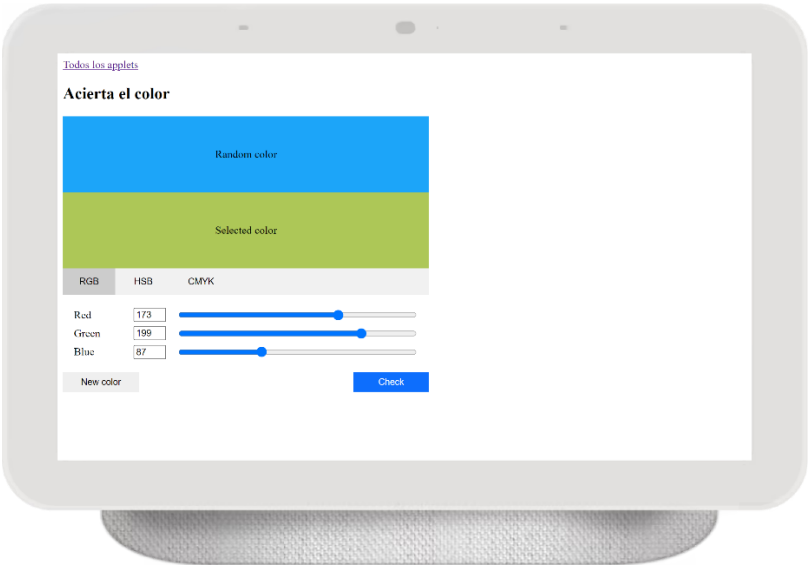

Figura 41: Componente de acierta el color emulado en Nexus Hub. Fuente: elaboración propia

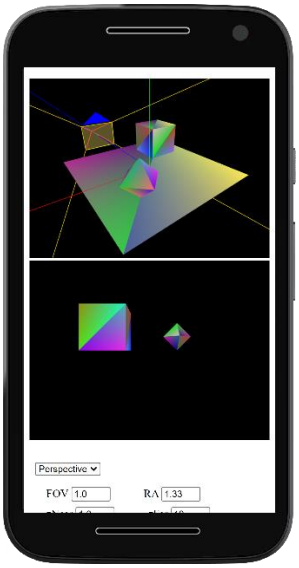

Figura 42: Componente de tipos de cámara emulado en un Moto G4. Fuente: elaboración propia

10 Página web de muestra

Los componentes desarrollados tienen pensado mostrarse en una página web, como un recurso más del profesorado de gráficos. Pero como a fecha de entregar esta memoria, dicha web aún no se ha creado. Es por este motivo que se ha creado una web simple con el único objetivo de mostrar el trabajo ralizado. En esta web de muestra [18] podemos encontranr enlaces a todos los programas creados. A partir de ahora no será necesario ir al repositorio del proyecto [19] y descargárselo en local para poder probar los web *applets*.

Para desplegar este sitio web se ha utilizado Render [20], ya que permite tener una página direcatmente enlazando un repositorio de GitHub. Además, cualquier actualización que se haga a la rama principal se sube de manera automática. Gracias a ser una web estática, el servicio es gratuito, siempre y cuando no supere los 100 GB de ancho de banda al mes.

11 Conclusiones

Este proyecto ha sido una experiencia positiva para mí. Desde siempre me han interesado los temas relacionados con la computación gráfica, y en este trabajo he visto la oportunidad de adentrarme más en este mundo. Además, me motiva el pensar que con este proyecto puedo ayudar a más alumnos como yo a facilitar la comprensión de conceptos que resultan difíciles de asimilar.

El proyecto me ha servido para ampliar mis conocimientos en tecnologías de desarrollo web, tales como HTML, CSS, JavaScript, y descubrir el potencial de la API de WebGL. También he aprendido sobre la importancia de una buena planificación y gestión de proyectos.

Al finalizar este trabajo podemos afirmar que se han cumplido con todos los objetivos y subobjetivos planteados al inicio. Se ha conseguido desarrollar componentes para todos los conceptos propuestos e incluso ampliar con uno más para la iluminación, el del modelo de Phong.

Aun así, somos conscientes de que el problema inicialmente planteado no se ha solucionado completamente, sino que se ha creado una herramienta que puede ayudar a aliviar la dificultad, suponiendo esto una mejora en cuanto al estado del arte. Esta herramienta cuenta con un gran potencial y sus resultados se verán en los siguientes años. Cabe resaltar que aún quedarían varios conceptos en los que podríamos aplicar la misma técnica y mejorar así el impacto positivo.

11.1 Futuras mejoras

Las siguientes características son mejoras que se podrían incluir en una segunda iteración del proyecto.

- Nuevos componentes. Como se mencionó antes, la computación gráfica es un campo muy extenso y aún quedan muchos conceptos de los cuales podríamos crear programas interactivos.
- Escenas editables. Añadir, eliminar y ubicar diferentes objetos. Sería útil tener la posibilidad de editar las escenas, sobre todo para componentes similares al de tipo de cámaras, que dependen del paisaje para mostrar los conceptos.
- Importar objetos desde archivos. Tener la posibilidad de añadir objetos desde archivos con los formatos más comunes, como Wavefront OBJ [21].
- Adaptar los componentes a más lenguajes. Incluir la opción de mostrar los programas en más idiomas, mínimamente el catalán y castellano, que son los que se utilizan en la universidad.

12 Bibliografía

[1] «Interacción y Diseño de Interfaces | Facultad de Informática de Barcelona». https://www.fib.upc.edu/es/estudios/grados/grado-en-ingenieria-informatica/plan-de-estudios/asignaturas/IDI (accedido 3 de marzo de 2022).

[2] «Gráficos | Facultad de Informática de Barcelona». https://www.fib.upc.edu/es/estudios/grados/grado-en-ingenieria-informatica/plan-de-estudios/asignaturas/G (accedido 3 de marzo de 2022).

[3] R. K. Frank y F. Hanisch, «Web based Teaching of Computer Graphics: Concepts and Realization of an Interactive Online Course.», 1998.

[4] Carlos Andújar Gran, Pere Brunet Crosa, Marta Fairén González, Isabel Navazo Álvaro, Àlvar Vinacua Pla, *Informatica Grafica*. Accedido: 23 de septiembre de 2021. [En línea]. Disponible en: https://www.cs.upc.edu/~virtual/G/index.php?dir=3.%20Material%20addicional/

[5] «Nate Robins - OpenGL - Tutors». https://user.xmission.com/~nate/tutors.html (accedido 15 de octubre de 2021).

[6] «Computer Graphics Learning - Computer Graphics». https://cglearn.codelight.eu/pub/computer-graphics (accedido 15 de octubre de 2021).

[7] «CglearnPoster.pdf». Accedido: 8 de marzo de 2022. [En línea]. Disponible en: https://cglearn.codelight.eu/doc/CglearnPoster.pdf

[8] «Graphics Programming», *Philipps-Universität Marburg*. https://www.uni-marburg.de/en/fb12/research-groups/grafikmultimedia/lectures/graphics (accedido 9 de marzo de 2022).

[9] «WebGL Example: Phong / Blinn Phong Shading». https://www.mathematik.uni-marburg.de/~thormae/lectures/graphics1/code/WebGLShaderLightMat/ShaderLightMat.html (accedido 9 de marzo de 2022).

[10] «WebGL - Phong Shading». http://www.cs.toronto.edu/~jacobson/phong-demo/ (accedido 9 de marzo de 2022).

[11] «Especificaciones HP Spectre 13-4003ns». https://icecat.biz/p/hp/m0r02ea/ordenadores-port-tiles-13-4003ns-26922193.html (accedido 6 de octubre de 2021).

[12] X. Martín Ballesteros, «Comparative study of missing data treatment methods in radial basis function neural networks: is it necessary to impute?», Universitat Politècnica de Catalunya, 2020, p. 36. Accedido: 6 de octubre de 2021. [En línea]. Disponible en: https://upcommons.upc.edu/handle/2117/340454

[13] A. Adán Navarro, «Mòdul de correcció distribuït per a Jutge.org», Universitat Politècnica de Catalunya, 2020, p. 94. Accedido: 15 de octubre de 2021. [En línea]. Disponible en: https://upcommons.upc.edu/handle/2117/334939

[14] «¿Cuánto cuesta el kilovatio hora de luz (kWh) en España?», *tarifaluzhora.es*. https://tarifaluzhora.es/info/precio-kwh (accedido 6 de octubre de 2021).

[15] «WebGL - Graphics Pipeline». https://www.tutorialspoint.com/webgl/webgl_graphics_pipeline.htm (accedido 14 de abril de 2022).

[16] «How 3D Game Rendering Works, A Deeper Dive: Rasterization and Ray Tracing», *TechSpot*. https://www.techspot.com/article/1888-how-to-3d-rendering-rasterization-ray-tracing/ (accedido 18 de abril de 2022).

[17] «UV mapping», *Wikipedia*. 8 de marzo de 2022. Accedido: 18 de abril de 2022. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=UV_mapping&oldid=1075997598

[18] «Webapplets para la comprensión de computación gráficos». https://web-components-for-graphics-education.onrender.com/demo/index.html (accedido 12 de abril de 2022).

[19] M. Malqui, *web-components-for-graphics-education*. 2022. Accedido: 12 de abril de 2022. [En línea]. Disponible en: https://github.com/MiguelMalqui/web-components-for-graphics-education
[20] «Cloud Application Hosting for Developers | Render», *Cloud Application Hosting for Developers | Render*. https://render.com/ (accedido 12 de abril de 2022).
[21] «Wavefront OBJ File Format», 23 de enero de 2020. https://www.loc.gov/preservation/digital/formats/fdd/fdd000507.shtml (accedido 6 de abril de 2022).