PROYECTO
ACUARIO VIRTUAL

PROFESOR: ING. ARTURO PEREZ DE LA CRUZ
316032863 FUENTES VIVEROS ADAN EMILIANO
416041343 SÁNCHEZ MANJARREZ ANDREW

Ciudad Universitaria, CDMX, México a 05 de enero de 2023

PROYECTO FINAL

REQUERIMIENTOS DE PROYECTO

1.1 PROPOSITO DEL PROYECTO

Agrupar elementos integrales de aprendizaje de la materia de CGEIHC (6590) en un proyecto compilado funcional entregable, que, a su vez, demuestre el uso eficiente de habilidades básicas, pragmáticas y teóricas.

1.2 OBJETIVOS DEL PROYECTO

Realizar la planeación y desarrollo de un proyecto que cumpla los requerimientos planteados.

- El alumno deberá seleccionar una fachada y un espacio que pueden ser reales o ficticios y presentar imágenes de referencia de dichos espacios para su recreación 3D en OpenGL. En la imagen de referencia se debe visualizar 7 objetos que el alumno va a recrear virtualmente y donde dichos objetos deben ser lo más parecido a su imagen de referencia, así como su ambientación.
- Entregar un manual de usuario en formato PDF, que deberá contener capturas de pantalla que ejemplifiquen cada elemento del manual.
- Entregar un manual técnico que incluya al cronograma de actividades realizadas para la conclusión del proyecto, con evidencias de herramientas colaborativas de desarrollo de software.

1.3 REQUERIMIENTOS DE ALTO NIVEL

Requerimientos Técnicos:

- Evidencias en plataformas colaborativas de software; con especificaciones de avance sobre los repositorios (Jira, Trello, Git, Github).
- Deberán utilizarse técnicas de modelado geométrico, modelado jerárquico, importación de modelos, y texturizado de escenarios.

1.4 SUPOCISIONES

La propuesta deberá ser entregada con anticipación para obtener su visto bueno antes de ser construida mediante un documento de propuesta de proyecto. De ser aprobada, se continuará su desarrollo para su entrega en diciembre de 2022

1.4.1 RESTRICCIONES

El proyecto se debe entregar de forma individual, con un manual de usuario donde se explique cada interacción dentro del ambiente virtual recreado y un manual técnico que contenga la documentación del proyecto que incluya objetivos, diagrama de Gantt, alcance del proyecto, limitantes y la documentación del código (no solo es copiar y pegar código y comentar algunas líneas de él).
Se debe compartir la liga de su proyecto en un repositorio en GitHub y se debe subir en la plataforma de classroom a más tardar a las 11:59 pm del martes para el grupo 06 y 07 y del día jueves para el grupo 11 y 12 para que el profesor pueda descargar el proyecto para su evaluación.

1.5 DESCRIPCIÓN DE ALTO NIVEL Y ALCANCES

Proyectos con evaluaciones menores a 5, se considerarán proyectos deficientes y por lo tanto serán sancionados con 1 o 2 puntos menos en su calificación final del curso.

- Los objetos recreados repetidamente se contarán como un objeto dentro de la evaluación.
- Se debe ocupar el código base visto durante el curso y otorgado por el profesor en caso contrario se anulará el proyecto y el alumno no acreditará el laboratorio.
- Queda prohibido recrear cualquier espacio perteneciente a la UNAM, con temática de los Simpson, Rick and Morty, la casa de Kamehouse de Dragon Ball y la casa de Coraje el perro cobarde.
- Las animaciones realizadas deben de tener contexto es decir no puede estar rotando un objeto nada más para cumplir con la rúbrica, tienen que ir acorde con el contexto del espacio recreado.
- Para que una animación sea compleja no tiene que ser lineal.
- No se pueden ocupar las animaciones complejas utilizadas dentro del curso ya que se valdrán como animaciones sencillas.

1.6 RESUMEN DEL CRONOGRAMA DE HITOS

ACTIVIDAD	DURACIÓN	S1	S2	S3	S4	S5	S6	S7
PLANEACIÓN	***2 hrs***							
Análisis de Requisitos	*1 hr.*							
Gestión Alcances, Elementos Utilizados	*1 hr.*							
DISEÑO	*4 hrs.*							
Descripción del Diseño	*1 hr.*							
Recopilación de elementos de Diseño	*2 hrs.*							
Acoplamiento de Componentes (plataformas)	*1 hr.*							
IMPLEMENTACIÓN	***40 hrs.***							
Programación de Requisitos	*6 hrs.*							
Búsqueda y buen uso de herramientas	*6 hrs.*							
Programación Modular	*14 hrs.*							
Revisión Preliminar	*7 hrs.*							
Documentación de Código	*4 hrs.*							
Filtrado y Análisis	*3 hrs.*							
PRUEBAS	***7 hrs.***							
Casos de uso para animaciones y requerimientos	*4 hrs.*							
Casos de uso Genéricos	*3 hrs.*							
DOCUMENTACIÓN	*23 hrs.*							
Documentación de Diseño	*5 hrs.*							
Documentación de Implementación	*5 hrs.*							
Documentación Entregable y Pruebas	*8 hrs.*							
Revisión y compilación de Entregables	*3 hrs.*							
Conclusiones de Proyecto	*2 hrs.*							

Encargado	Sánchez Manjarrez Andrew
Encargado	Fuentes Viveros Adán Emiliano

1.8 RESUMEN DE COSTOS Y APROXIMACIONES

CATEGORIA	*S1*	*S2*	*S3*	*S4*	*S5*	*S6*	*S7*	
Sueldos	*$1500.*^{00}	*$2500.*^{00}	*$4750.*^{00}	*$3750.*^{00}	*$3750.*^{00}	*$1500.*^{00}	*$500.*^{00}	
Plan de Propuestas	*$1500.*^{00}							
Gastos de Desarrollo		*$1000.*^{00}	*$1000.*^{00}	*$1000.*^{00}	*$1000.*^{00}			
Administración de Herramientas		*$1000.*^{00}	*$1000.*^{00}	*$1000.*^{00}	*$1000.*^{00}			
Material y Mantenimiento de Equipos							*$1000.*^{00}	**COSTO TOTAL**
Desarrollo de Entregables Plan de Proyecto	*$500.*^{00}		*$500.*^{00}		*$1000.*^{00}	*$1500.*^{00}	*$500.*^{00}	$32,750.^{00}
							PRECIO:	$40,000.^{00}

1.9 PROPUESTA PARA EL PROYECTO DE COMPUTACIÓN GRÁFICA

Cuarto a recrear

Se ubica en la parte central de nuestro complejo, contempla estructuras que como su nombre indica sirven propósitos de exhibición típicos de un Acuario, los elementos que podemos encontrar aquí son peceras, exhibidores, carteles, modelos de criaturas marinas en maqueta, entre otros elementos. Su extensión alberga la mayor parte central del recinto, desde el edificio central hasta rozar con los alrededores de las demás áreas.

Objetos:

Plantas Marinas, Decoración Marina: Son parte importante de la estética del acuario, dota de vitalidad a las escenas y la ambienta en un espacio subacuático.
Maquetas de Criaturas Marinas: Pertenecen al área de exposición sirven propósitos generales como elementos descriptivos, decorativos e informativos
Expositor de peceras, displays customizados: muestran la barrera de las diferentes criaturas marinas que si están animadas para interactuar en el complejo.
Ilustraciones variadas, carteles: Cumplen funciones indicativas, sugieren, dan instrucciones o guían a los consumidores de respectiva manera
Elementos especiales: Sillas de comedores, mesas temáticas, cadenas, escritorios de recepción

Estructura Principal (Contenedor de los Modelos Propuestos)

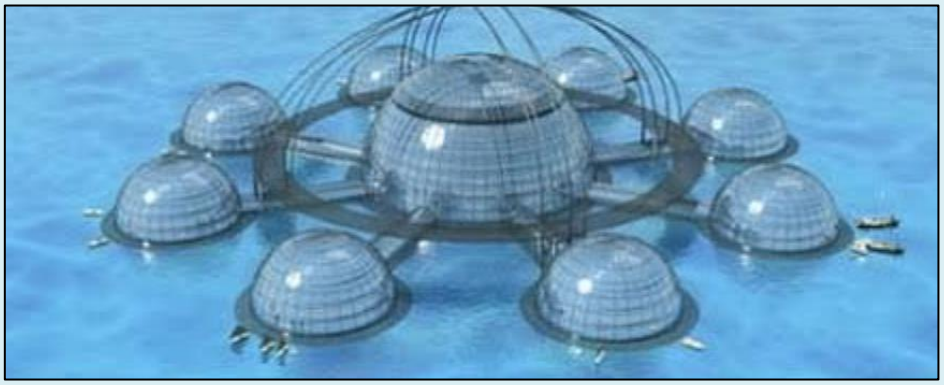


Fig. 1 Imagen conceptual del complejo de estructuras del Acuario submarino.

Nos basaremos en el siguiente modelo conceptual para recrear solo la semiesfera principal del nexo que albergara nuestra areas funcionales definidas en los Modelos Propuestos, se aprecia además la ambientación exterior a esta estructura principal, los cuales tambien buscaran ser recreados para los propositos de este proyecto.

Definición de Modelos Propuestos: Descripción de Elementos:

Area de Exposición:

Se ubica en la parte central de nuestro complejo, contempla estructuras que como su nombre indica sirven propositos de exhibición tipicos de un Acuario, los elementos que podemos encontrar aquí son peceras, exhibidores, carteles, modelos de criaturas marinas en maqueta, entre otros elementos. Su extensión alberga la mayor parte central del recinto, desde el edificio central hasta rozar con los alrededores de las demas areas.

Objetos

1. Plantas Marinas, Decoración Marina: Son parte importante de la estética del acuario, dota de vitalidad a las escenas y la ambienta en un espacio subacuático.
2. Maquetas de Criaturas Marinas: Pertenecen al área de exposición sirven propósitos generales como elementos descriptivos, decorativos e informativos
3. Expositor de peceras, displays customizados: muestran la barrera de las diferentes criaturas marinas para interactuar en el complejo.
4. Ilustraciones variadas, carteles: Cumplen funciones indicativas, sugieren, dan instrucciones o guían a los consumidores de respectiva manera
5. Elementos especiales: Aparadores, elementos de ambientación, fuentes, bancas, esculturas, etc.
6. Submarinos: Transportes característicos del complejo que arriban y parten de la zona de llegada.

Animaciones

1. Animación de movimiento de submarino, contiene una ruta especificada con el movimiento característico del transporte a los puntos predeterminados
2. Animación de movimiento de tiburones, mantienen con integridad su movimiento a través de su área.
3. Animación de movimiento de algas: movimientos propios dentro de su área definida que crean sensación de flujos y corrientes marinas
4. Animación de movimiento de peces, movimientos propios dentro de su área definida.
5. Antena, movimientos característicos de este objeto de comunicación.

1

Plataformas de Trabajo.

Para el desarrollo de nuestro ambiente del acuario fue necesario trabajar bajo el entorno de desarrollo integrado de Microsoft Visual Studio (IDE) el cual es utilizado para desarrollar programas informáticos, aplicaciones, y servicios, para nuestro caso, siguiendo de la mano las operaciones de OpenGL que se considera principalmente una API (una interfaz de programación de aplicaciones) el cual nos brinda un gran conjunto de funciones que podemos usar para manipular gráficos e imágenes. Sin embargo, OpenGL por sí mismo no es una API, sino simplemente una especificación, desarrollada y mantenida por el Grupo Khronos.

La especificación OpenGL define exactamente cuál debe ser el resultado/salida de cada función y cómo debe funcionar. Luego, depende de los desarrolladores como nosotros que implementemos estas especificaciones encontrar una solución de cómo debería operar esta función. Dado que la especificación de OpenGL no nos brinda detalles de implementación, las versiones desarrolladas reales de OpenGL pueden tener diferentes implementaciones, siempre que sus resultados cumplan con la especificación (y, por lo tanto, sean los mismos para los usuarios).

Es importante denotar que para el desarrollo de este proyecto jurásico se trabajó
con las bibliotecas OpenGL que están escritas en C y permiten muchas derivaciones en otros lenguajes, pero en esencia sigue siendo una biblioteca C.

Dado que muchas de las construcciones de lenguaje de C no se traducen tan bien a otros lenguajes de nivel superior, OpenGL se desarrolló con varias abstracciones en mente. Una de esas abstracciones son los objetos en OpenGL.

Se definieron múltiples objetos para este proyecto de recreación de acuario, siguiendo las especificaciones de requerimientos funcionales del apartado anterior. Un objeto en OpenGL es una colección de opciones que representa un subconjunto del estado de OpenGL. Por ejemplo, podríamos tener un objeto que represente la
configuración de la ventana de dibujo; luego podríamos establecer su tamaño, cuántos colores admite, etc. Uno podría visualizar un objeto como una estructura tipo C.

Lo bueno de usar estos objetos es que podemos definir más de un objeto en nuestra aplicación, configurar sus opciones y cada vez que comenzamos una operación que usa el estado de OpenGL, vinculamos el objeto con nuestra configuración preferida. Hay objetos, por ejemplo, que actúan como objetos contenedores para los datos del modelo 3D (una casa o un personaje) y cada vez que queremos dibujar uno de ellos, vinculamos el objeto que contiene los datos del modelo que queremos dibujar (primero creamos y configuramos opciones para
estos objetos). Tener varios objetos nos permite especificar muchos modelos y siempre que queramos dibujar un modelo específico, simplemente vinculamos el objeto correspondiente antes de dibujar sin volver a configurar todas sus opciones.

2. Parámetros Básicos del Entorno

2.1. Uso de la Ventana

Lo primero que debimos configurar antes de comenzar a crear el entorno gráfico submarino o de acuario fue crear un contexto OpenGL y una
ventana de aplicación para dibujar. Sin embargo, esas operaciones
son específicas por sistema operativo y OpenGL intenta abstraerse de estas operaciones a propósito. Esto significa que tenemos que crear una ventana, definir un contexto y manejar la entrada del usuario por nosotros mismos. Afortunadamente, existen bastantes bibliotecas que brindan la funcionalidad que buscábamos, algunas dirigidas específicamente a OpenGL. Utilizamos GLFW. GLFW es una biblioteca, escrita en C, dirigida específicamente a
OpenGL. GLFW nos brinda las necesidades básicas requeridas para mostrar cosas en la pantalla. Nos permite crear un contexto
OpenGL, definir parámetros de ventana y manejar la entrada del usuario, que es suficiente para nuestros propósitos.

2.2. Vinculación

Para que el proyecto use GLFW, necesitábamos vincular la biblioteca con nuestro proyecto. Esto se pudo lograr especificando que queremos usar glfw3.lib en la configuración del enlazador, pero
nuestro proyecto aún necesitaba saber dónde encontrar glfw3.lib ya que almacenamos nuestras bibliotecas de terceros en un directorio
diferente. Por lo tanto, primero debemos agregar este directorio al proyecto.
Podemos, además, decirle al IDE que tenga en cuenta este directorio cuando necesite buscar una biblioteca e incluir archivos.
Nuestra lista de vinculaciones contempla:
$(SolutionDir)/External Libraries/GLEW/lib/Release/Win32
$(SolutionDir)/External Libraries/GLFW/lib-vc2015
$(SolutionDir)/External Libraries/SOIL2/lib
$(SolutionDir)/External Libraries/assimp/lib

2.2.1 Bibliotecas Externas Incluidas.

Para que el proyecto pudiera saber en que parte buscar las herramientas necesarias, se inserto manualmente la cadena de ubicación adecuada, el IDE también buscará en esos directorios cuando busque bibliotecas y archivos de encabezado. Tan pronto como se incluyeron las carpetas como la de GLFW, se pudo

encontrar todos los archivos de encabezado para GLFW al incluir
<GLFW/..>., etc. Nuestra lista de Biblioteca incluye:
$(SolutionDir)/External Libraries/GLEW/include
$(SolutionDir)/External Libraries/GLFW/include
$(SolutionDir)/External Libraries/glm
$(SolutionDir)/External Libraries/assimp/include

2.2.2 Parámetros de Entrada del Vinculador.

Una vez que establecimos las cabeceras externas, nuestra IDE VisualStudio puede encontrar todos los archivos necesarios, y finalmente podemos vincular las bibliotecas al proyecto yendo a la pestaña Vinculador y Entrada para terminar nuestra configuración.
soil2-debug.lib;assimp-vc140-mt.lib;opengl32.lib;glew32.lib;glfw3.lib;

3 Shaders

Se utilizaron shaders para ayudar a construir el entorno de acuario de este proyecto, Los shaders son pequeños programas que descansan en la GPU.

Estos programas se ejecutan para cada sección específica de la canalización de gráficos. En un sentido básico, los shaders no son más que programas que transforman entradas en salidas. Los shaders también son programas muy aislados en el sentido de que no se les permite
comunicarse entre sí; la única comunicación que tienen es a través de sus entradas y salidas.
Los shaders utilizados están escritos en el lenguaje GLSL similar a C. GLSL está diseñado para su uso con gráficos y contiene características útiles específicamente dirigidas a la manipulación de vectores y matrices.
Los shaders siempre comienzan con una declaración de versión, seguida de una lista de variables de entrada y salida, uniformes y su función principal.
El punto de entrada de cada shader está en su función principal donde procesamos cualquier variable de entrada y mostramos los resultados en sus variables de salida.
Se utilizaron shaders para la carga de modelos, para la iluminación, para la ambientación (cubemaps), y para las animaciones del proyecto acuario.
Aparecen listados en una carpeta especifica llamada Shaders y su programación es autóctona de su función.

4. Carga de Modelos
4.1 ASSIMP
Para nuestro entorno acuario no podemos definir manualmente todos los vértices, las normales y las coordenadas de textura de formas complicadas.
En cambio, lo que queremos es importar estos modelos a la aplicación; modelos cuidadosamente diseñados por nosotros en herramientas graficas
como 3DS Max o Maya. Estas herramientas de modelado 3D nos permiten crear formas complicadas y aplicarles texturas a través de un proceso
llamado mapeo uv. Luego, las herramientas generan automáticamente todas las coordenadas de vértice, las normales de vértice y las coordenadas
de textura mientras las exportan a un formato de archivo de modelo que podemos usar. De esta forma, contamos con un extenso conjunto de herramientas para crear modelos de alta calidad sin tener que preocuparnos demasiado por los detalles técnicos. Todos los aspectos
técnicos están ocultos en el archivo del modelo exportado. Sin embargo, nosotros, como programadores de gráficos, tenemos que preocuparnos por estos detalles técnicos.
Una biblioteca de importación de modelos muy popular que fue implementada para este proyecto, es Assimp, que significa Biblioteca abierta de importación de activos. Assimp puede importar docenas de formatos de archivo de modelo diferentes (y exportar a algunos también) cargando todos los datos del modelo en las estructuras de datos generalizadas de Assimp. Tan pronto como Assimp haya cargado el modelo, podemos recuperar todos los datos que necesitamos de las estructuras de datos de Assimp. Debido a que la estructura de datos de Assimp permanece igual, independientemente del tipo de formato de archivo que importamos,
nos abstrae de todos los diferentes formatos de archivo que existen.
Al importar un modelo a través de Assimp, carga todo el modelo en un objeto de escena que contiene todos los datos del modelo/escena importados. Assimp luego tiene una colección de nodos donde cada nodo contiene índices de datos almacenados en el objeto de escena donde

cada nodo puede tener cualquier número de hijos.

4.2 Mesh

Con Assimp pudimos cargar muchos modelos diferentes en la aplicación, pero una vez cargados, todos se almacenan en las estructuras de datos de Assimp. Lo que eventualmente necesitamos es transformar esos datos a un formato que OpenGL entienda para que podamos representar los objetos. Una malla debería necesitar al menos un conjunto de vértices, donde cada vértice contiene un vector de posición, un vector normal y un vector de coordenadas de textura. Una malla también debe contener índices para el dibujo indexado y datos de materiales en forma de texturas (mapas difusos/especulares)

Gracias al constructor, obtuvimos grandes listas de datos de malla que podemos usar para renderizar. Necesitamos configurar los búferes apropiados y especificar el diseño del shader de vértices a través de punteros de atributo de vértice para que finalmente con la última función que necesitamos definir para que la clase Mesh esté completa es su función 'Draw'. Antes de renderizar la malla, primero queremos enlazar las texturas apropiadas antes de llamar a glDrawElements. Sin embargo, esto es algo difícil ya que no sabemos desde el principio cuántas texturas (si las hay) tiene la malla y qué tipo pueden tener.

4.3 Model

A partir de este punto en el proyecto, comenzamos a crear el código de traducción y carga real con Assimp. El objetivo es crear otra clase que represente un modelo en su totalidad, es decir, un modelo que contenga múltiples mallas, posiblemente con múltiples texturas. Un dinosaurio, por ejemplo, que mantiene garras, dientes, ojos y piel aún podría cargarse como un solo modelo. Cargaremos el modelo a través de Assimp y lo traduciremos a varios objetos de malla que hemos creado.

Teniendo en cuenta que asumimos que las rutas de los archivos de textura en los archivos del modelo son locales para el objeto del modelo real, Ejemplo. en el mismo directorio que la ubicación del propio modelo.

Entonces podemos simplemente concatenar la cadena de ubicación de textura y la cadena de directorio que recuperamos anteriormente (en la función loadModel) para obtener la ruta de textura completa (es por eso que la función GetTexture también necesita la cadena de directorio).

Algunos modelos que se obtuvieron en Internet para este proyecto usaron rutas absolutas para sus ubicaciones de textura, lo que no funcionó al momento de llegar a este punto. En ese caso, editamos manualmente el archivo para usar rutas locales para las texturas (si es posible). O se reemplazaron las texturas y se inició un proceso de adaptación al modelo que se describe a continuación.

2.1 MANUAL TÉCNICO:
PROCESO DE ELABORACIÓN Y ADAPTACIÓN

Con base en el cronograma de actividades y la culminación de la fase de Planeación y Diseño, se llegó a la fase contigua de Implementación donde colaborativamente se añadieron entradas, cambios, y eliminación de elementos para ajustarse a la entrega pactada en el análisis de requerimientos.

Entre la programación modular se establecieron métricas de trabajo para poder trabajar con el entorno jurásico.

Entre estos rubros se definieron procesos para añadir objetos;

1. Importación o creación del modelo.

   La mayoría de los modelos del proyecto son material intelectual propio, pero alrededor del 45% se obtuvieron de plataformas que ofertan diversos modelos gráficos con distintas licencias de uso.

   Los modelos obtenidos al ser de licencia gratuita no contenían texturas ni materiales. Para este punto se establecieron parámetros para su integración dentro del escenario.

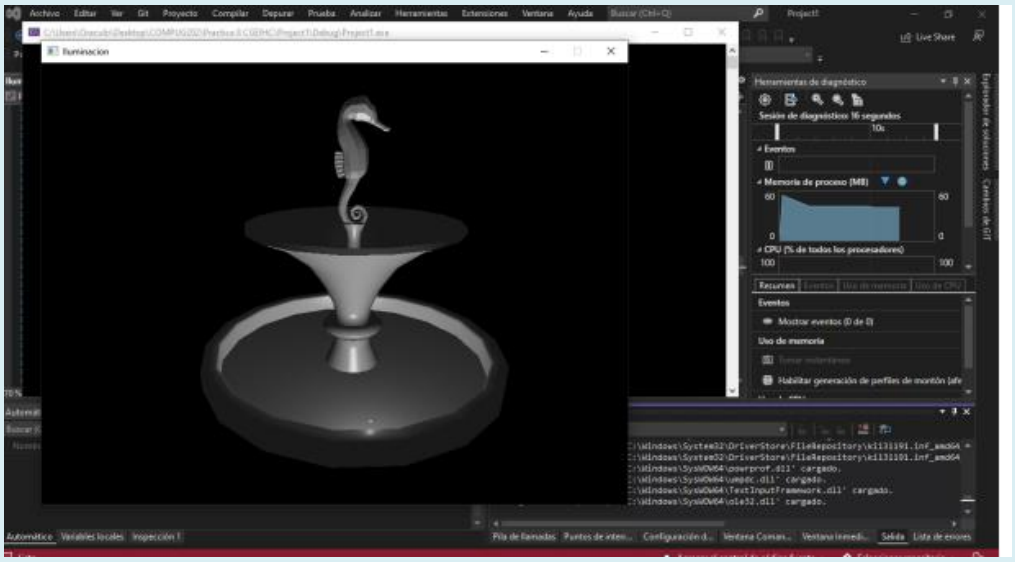


Imagen 1. Vista de interfaz de software grafico; se aprecia la importación del modelo sin texturas, y con transformaciones básicas para su acoplamiento al espacio de trabajo.

2. Selección de Texturas y adaptación de materiales.

Los modelos seleccionados y que fueron filtrados para la fase final de introducción al escenario, fueron revisitados para la toma de texturas. Se buscó la mejor calidad de materiales y se trabajaron las imágenes con el software GIMP, para poder utilizarse como textura y moldear el mapa de uv. Con la finalidad de crear una mejor ambientación al entorno acuatico.

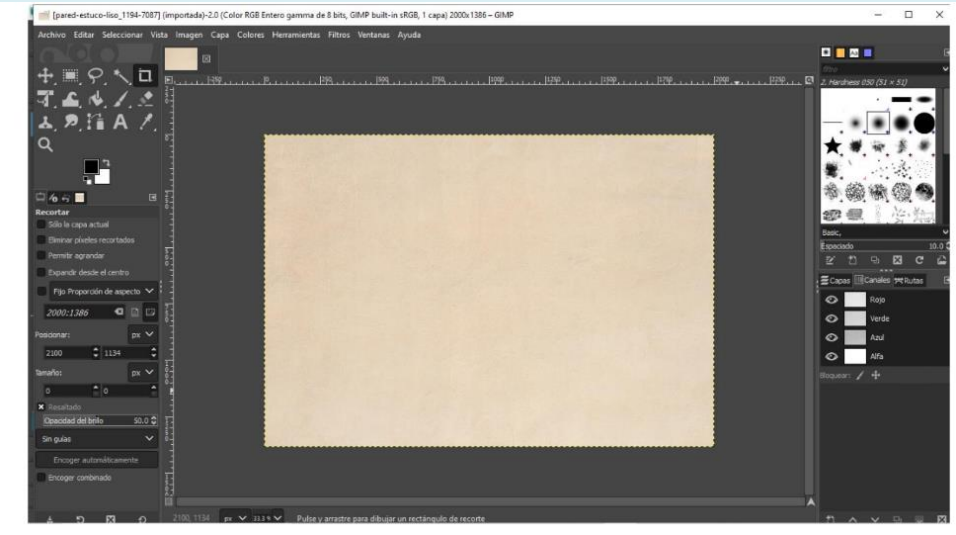


Imagen 2. Vista de interfaz de GIMP, se aprecia la importación de la imagen para la textura, se emplean herramientas propias de la plataforma para adecuar la textura, en tamaño y calidad.

3. Convergencia de materiales y finalización del modelado

Se acoplan los materiales al modelo, se vuelven a revisar transformaciones básicas del objeto para adecuar su posición dentro del escenario y finalmente, se importa para su posterior incorporación al entorno de desarrollo integrado que trabaja la integración completa del proyecto.

Imagen 3. Vista de interfaz de software grafico; se aprecia la forma final del modelo con texturas integradas, y con las transformaciones básicas de posicionamiento.

4. Integración del modelo al escenario final

   Finalmente, se intercambia el espacio de trabajo para continuar con la integración y acoplamiento al entregable final. Se asegura que el nuevo modelo no afecte la estabilidad del proyecto, que no existan posicionamientos no deseados, o que bien, la integración del componente sea la deseada.

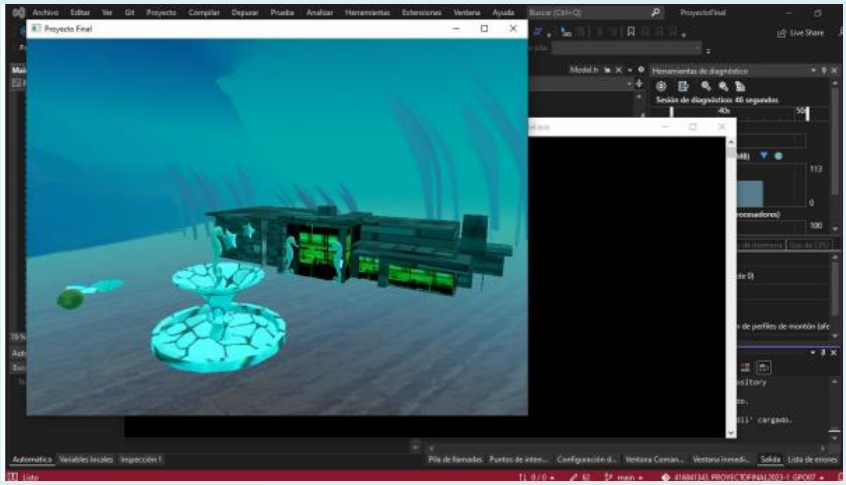


Imagen 4. Vista de interfaz de software de Visual Studio; se aprecia la forma final del modelo con los demás modelos también terminados.

5. Control de Versiones

   Se guardan los nuevos cambios y se actualizan en el repositorio central; se contemplan comentarios en la documentación como la fecha de edición, el cambio realizado y la funcionalidad.

Se utiliza la herramienta colaborativa de GitHub para establecer un parámetro de control de versiones, respaldo y trabajo contiguo.

2.2 MANUAL TÉCNICO: USO DE AMBIENTACIÓN: ILUMINACIÓN

Para este apartado colocamos todos las variables uniformes para los tipos de luces que tenemos; direccional, ambiental, difusa y especular. Tenemos que configurarlas manualmente e indexarlas.

Debemos adecuar la estructura PointLight propia en la matriz para establecer cada variable uniforme. Podemos incluso definir tipos de luz como clases y estableciendo sus valores allí, o usando un enfoque uniforme más eficiente

```
// == ========================
// Directional light
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), -0.8f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.4f, 0.4f, 0.4f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.5f, 0.5f, 0.5f);


// Point light 1
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositions[0].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), 0.05f, 0.05f, 0.05f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), LightP1.x, LightP1.y, LightP1.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), LightP1.x, LightP1.y, LightP1.z);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 0.032f);


// Point light 2
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].position"), pointLightPositions[1].x, pointLightPositions[1].y, pointLightPositions[1].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].ambient"), 0.05f, 0.05f, 0.05f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].diffuse"), 1.0f, 1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].specular"), 1.0f, 1.0f, 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].quadratic"), 0.032f);
```

70 % No se encontraron problemas. Línea: 545 Carácter: 117 Columna: 123 TABULACIONES CRLF

AL JUGAR, ADAPTAR Y MODIFICAR ESTOS PARÁMETROS PODEMOS CONSEGUIR DIFERENTES FENÓMENOS DE ENTORNO EN NUESTRO ESCENARIO FINAL, POR EJEMPLO;

PODEMOS CREAR UN AMBIENTE CON LUZ CLARA Y AGUA CRISTALINA, U OTRO CON MENOR CLARIDAD Y UN EFECTO AMBIENTAL DE MAYOR PROFUNDIDAD

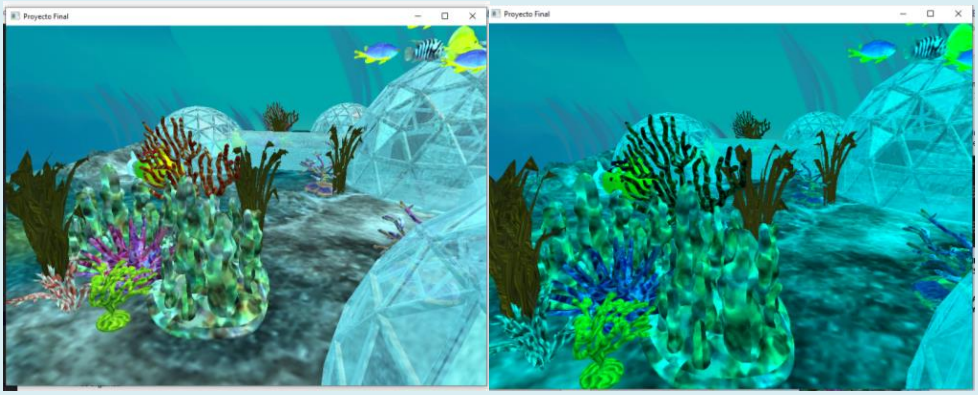


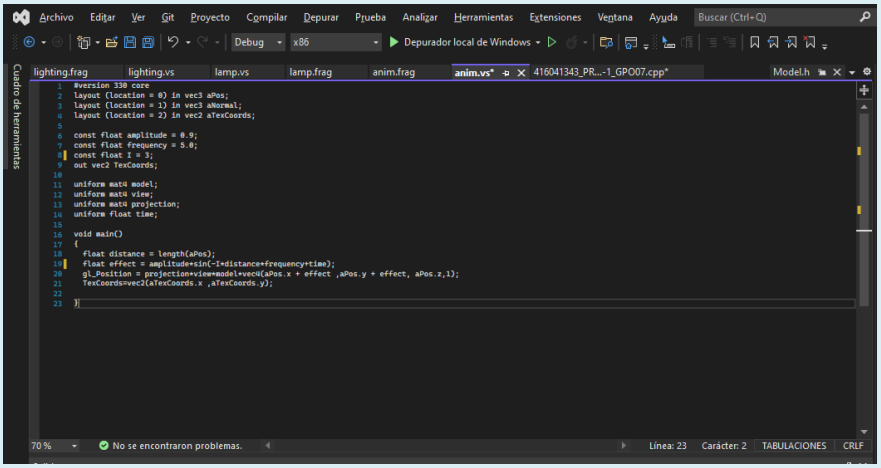
Imagen 3,4: (izq) Imagen con valores de luz ambiental más altos para la primer componente creando un aspecto más claro y definido de ciertas texturas, (der) imagen con valores ambientales más bajos, definiendo un ambiente más acuático.

2.3 MANUAL TÉCNICO: USO DE ANIMACIONES

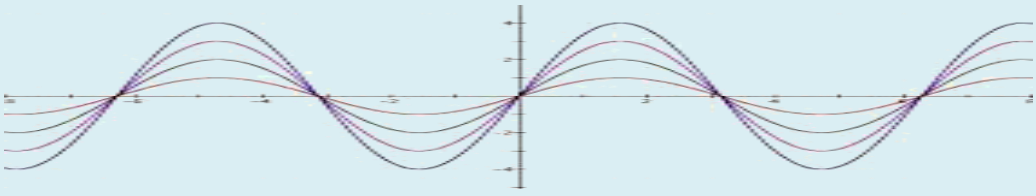
Para nuestra primera animación (alga) necesitamos dotar de movimiento a nuestro escenario creando la sensación de movimiento y flujo activo de corriente, esto puede ser logrado definiendo un movimiento típico asociado a los movimientos de marea.

```
Anim.Use();
modelLoc = glGetUniformLocation(Anim.Program, "model");
viewLoc = glGetUniformLocation(Anim.Program, "view");
projLoc = glGetUniformLocation(Anim.Program, "projection");
tiempo = glfwGetTime() * speed;
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
model = glm::mat4(1);
glUniform1f(glGetUniformLocation(Anim.Program, "time"), tiempo);

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));

Alga.Draw(Anim);
glBindVertexArray(0);
```

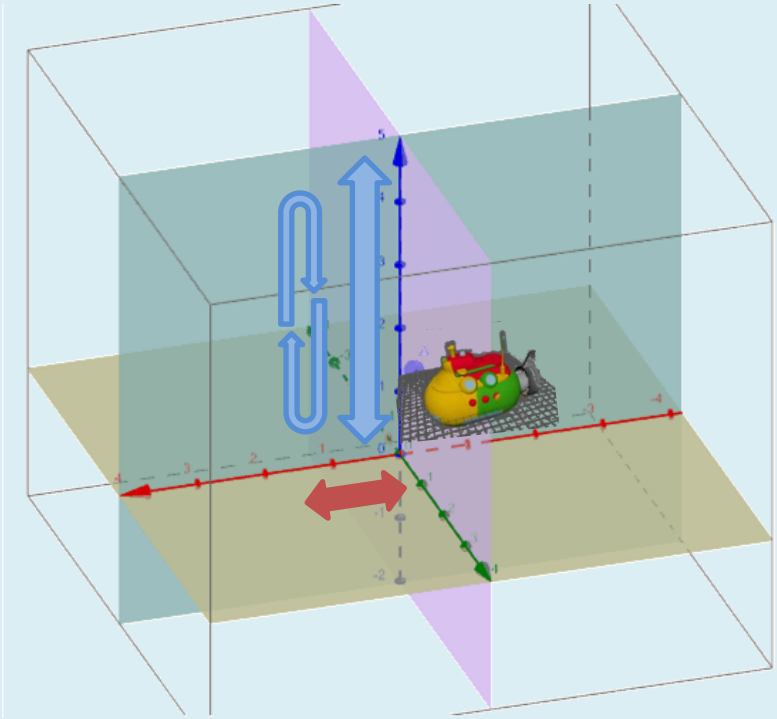
Definimos nuestro vertex shader con las constantes de apoyo dentro de la función, las variables uniformes de nuestro programa, para que nuestra alga que es un elemento (vertical) tenga el movimiento adecuado en los planos deseados necesitaremos operarla en los componentes x e y.



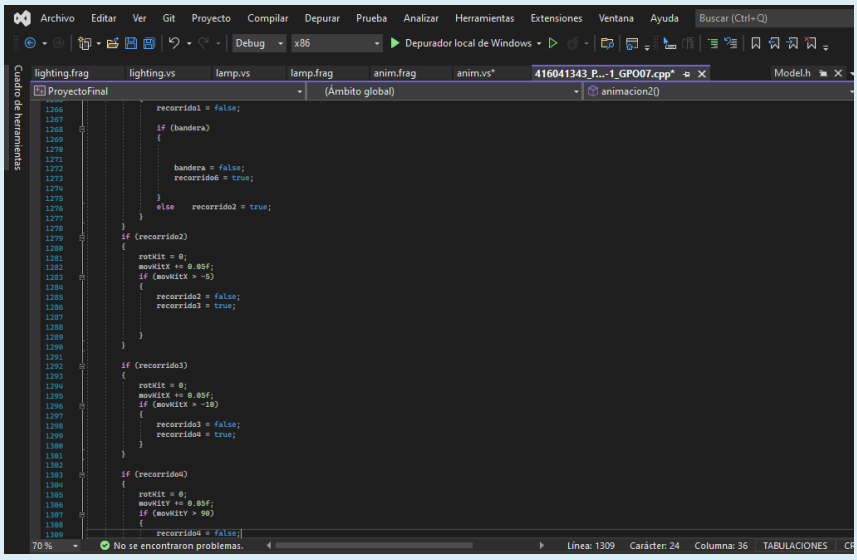
Se modifica una amplitud medianamente valuada para no deformar los elementos y seguir mostrando un movimiento adecuado, esto se consigue variando el parámetro y haciendo pruebas sobre el elemento que se encuentra siguiendo una trayectoria similar a la siguiente.



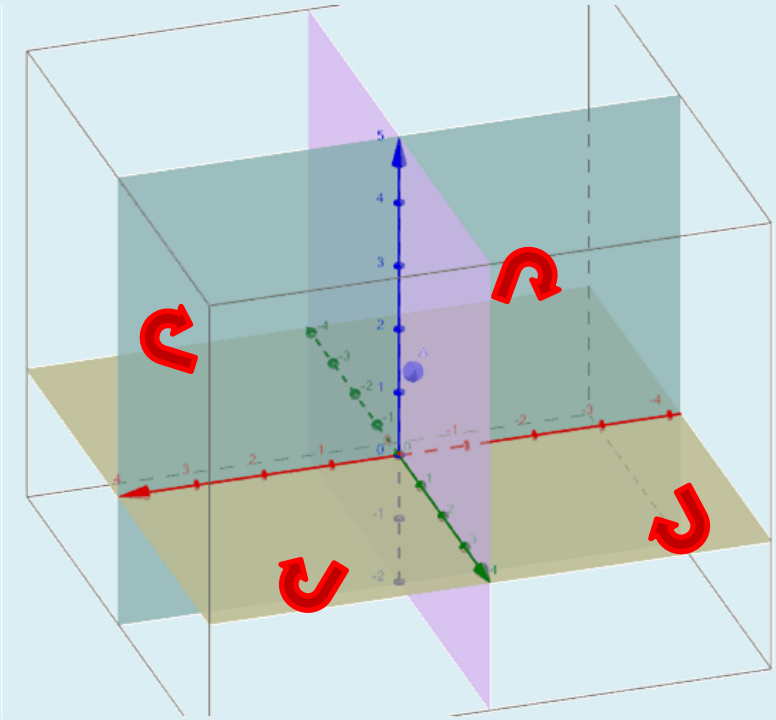
Para la animación de nuestro submarino, realizamos la secuencia de abordaje, ascenso y descenso de manera cíclica, una vez posicionado correctamente dentro del ambiente y activada la animación el submarino realiza las acciones determinadas en el siguiente plano

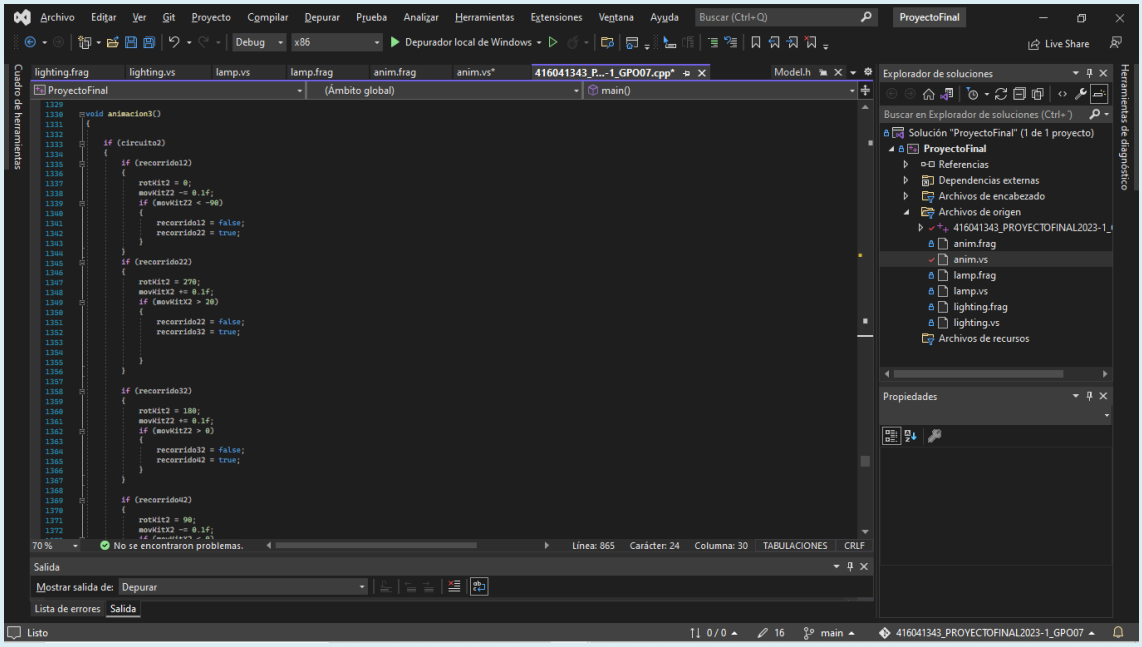


```
model = glm::mat4(1);
model = glm::translate(model, PosIni + glm::vec3(movKitX, movKitY, 0));
model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Submarino.Draw(lightingShader);
glBindVertexArray(0);
```

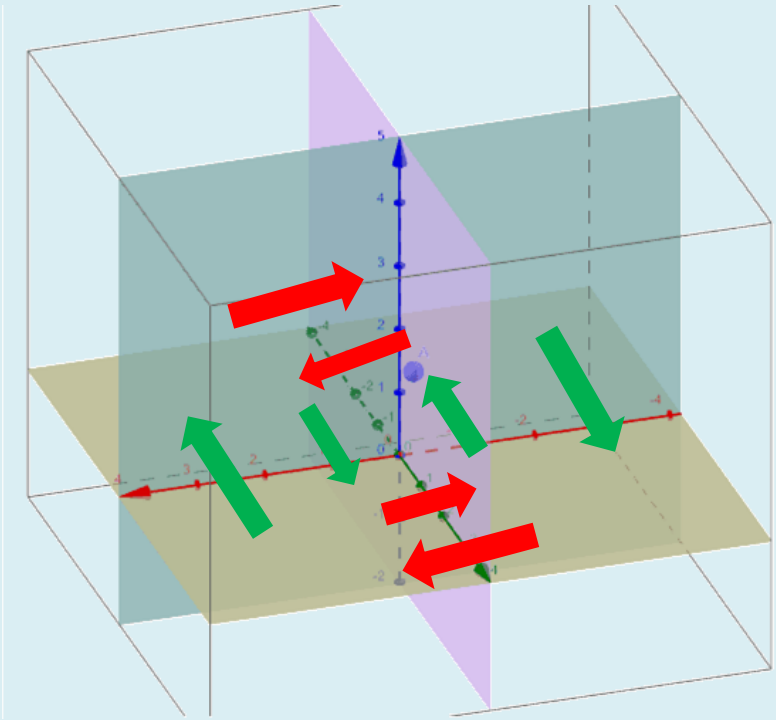


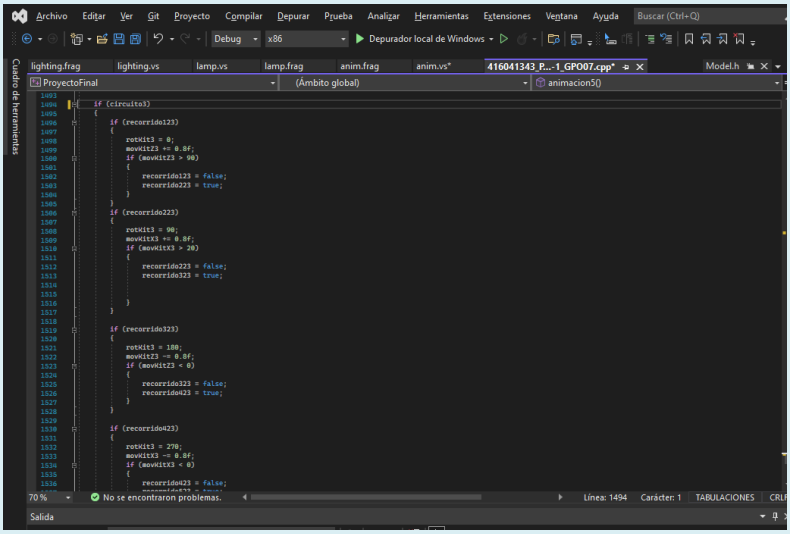
Para la animación de nuestro tiburón planteamos un circuito infinito alrededor de todo el escenario, esto incluye la rotación de su cuerpo para ir acorde a la dirección planteada, y a una velocidad definida adecuada. Los planos de su movimiento coinciden con las componentes definidas.



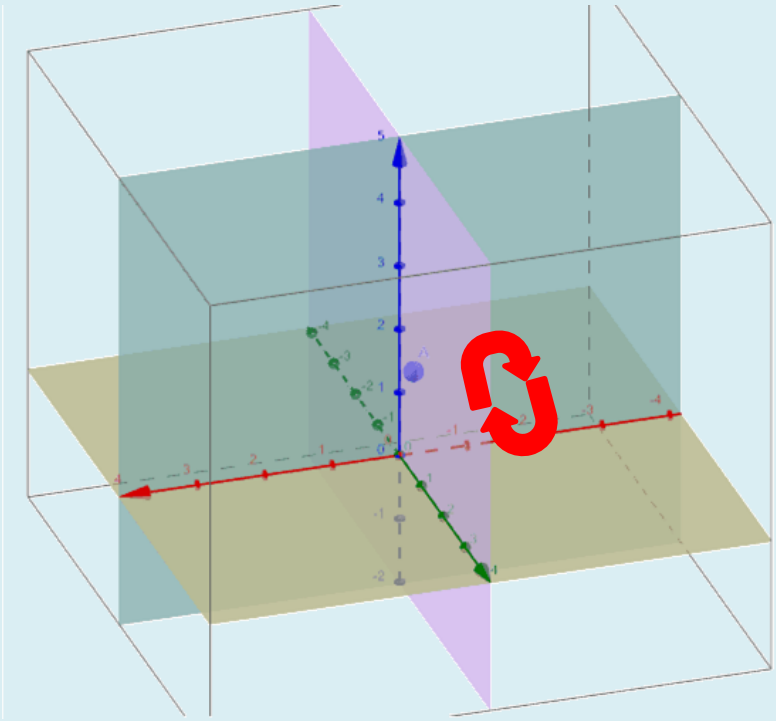


Para la animación de los peces existen dos circuitos, el primero en sentido horario como nuestra anterior animación y el siguiente en sentido antihorario que sería el propio de esta animación, las rotaciones de los cuerpos de los peces son las adecuadas para ambas direcciones y están en una secuencia infinita a través del escenario una vez iniciadas, además esta secuencia puede detenerse y reanudarse en cualquier momento

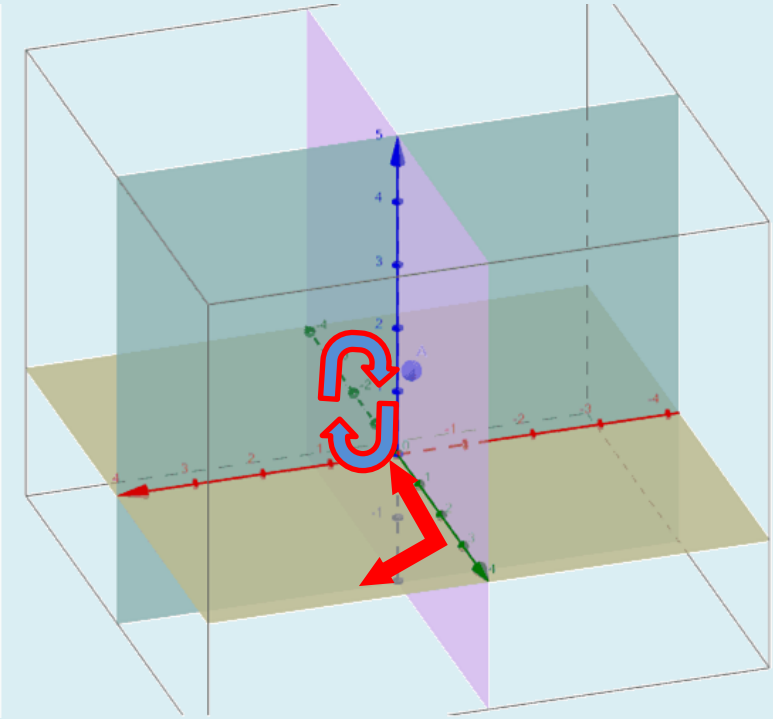




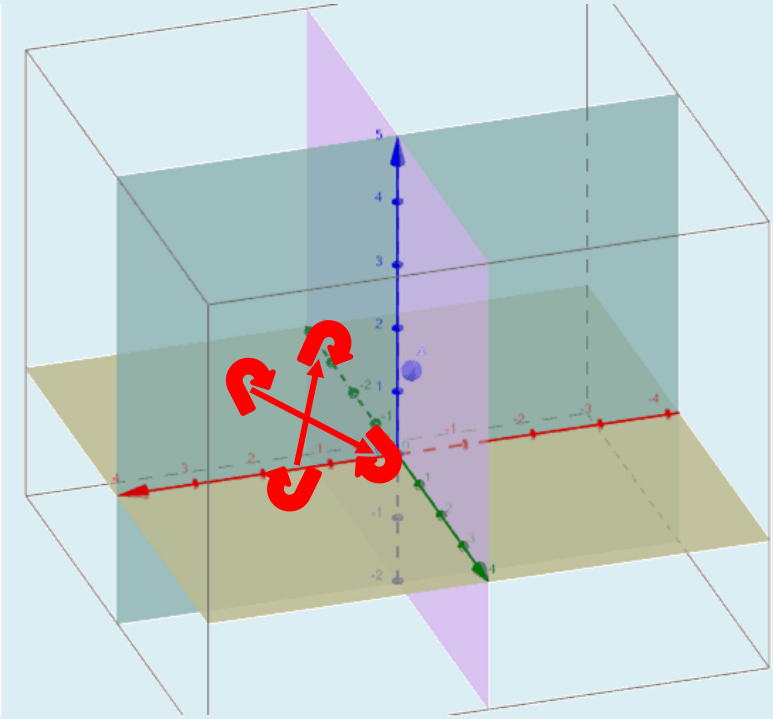
Para la animación de la antena se dotó de un movimiento característico a la misma con una rotación simple dentro de uno de los espacios destinados para su ambientación, este movimiento es cíclico y puede detenerse en cualquier momento y volver a reanudar



Para la animación de la bóveda se dotó de un movimiento característico a la misma con una rotación de la manivela dentro de uno de los espacios destinados con un giros de 360 grados, que al finalizar tiene una apertura característica de una puerte para su ambientación, este movimiento es cíclico y puede detenerse en cualquier momento y volver a reanudar



Para la animación del área del pingüino se realizo un pequeño recorrido o movimiento característico a la misma con un triangulaciones alrededor de uno de los espacios destinados para su ambientación, este movimiento es cíclico y puede detenerse en cualquier momento y volver a reanudar



2.4 MANUAL TÉCNICO:
AMBIENTACIÓN Y ADECUACIÓN

Para esta sección se colocarán algunas muestras con los modelos adecuados y las texturas definidas para la ambientación correcta del escenario propuesto. Se utilizaron diferentes técnicas de computación grafica y sumarización de conceptos navegados a través del temario.

Imagen 5. Se pueden observar más de 6 modelos diferentes cada uno con textura adecuada tan solo para recrear los distintos espacios dentro del escenario, los corales cumplen funcion decorativa, y las algas tienen una mayor relevancia al dar movimiento al escenario.

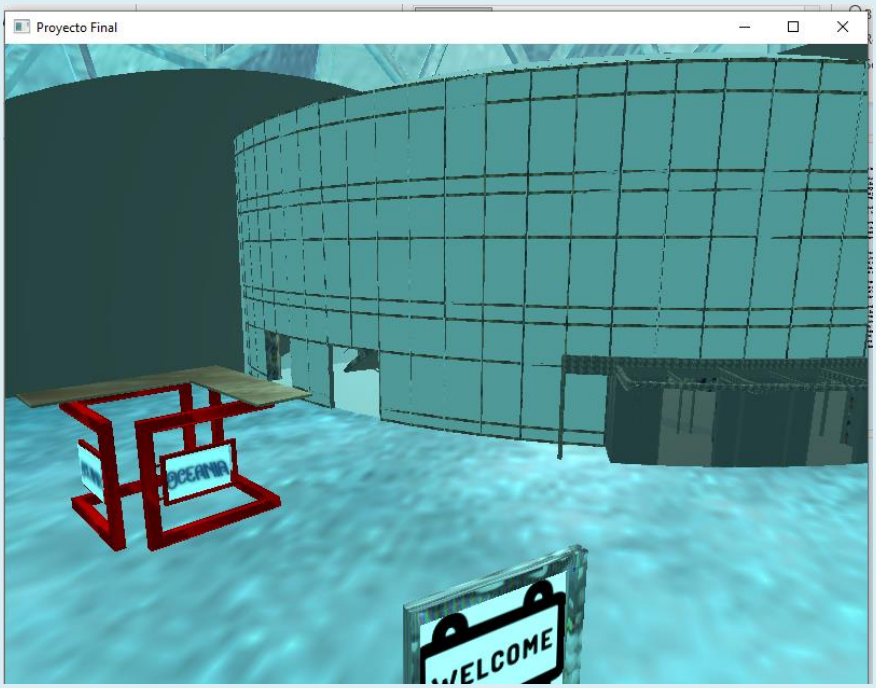


Imagen 6. El escenario es una homologación de distintos componentes fusionados, se eligieron distintas texturas para modelarlo y se colocaron elementos en la entrada.

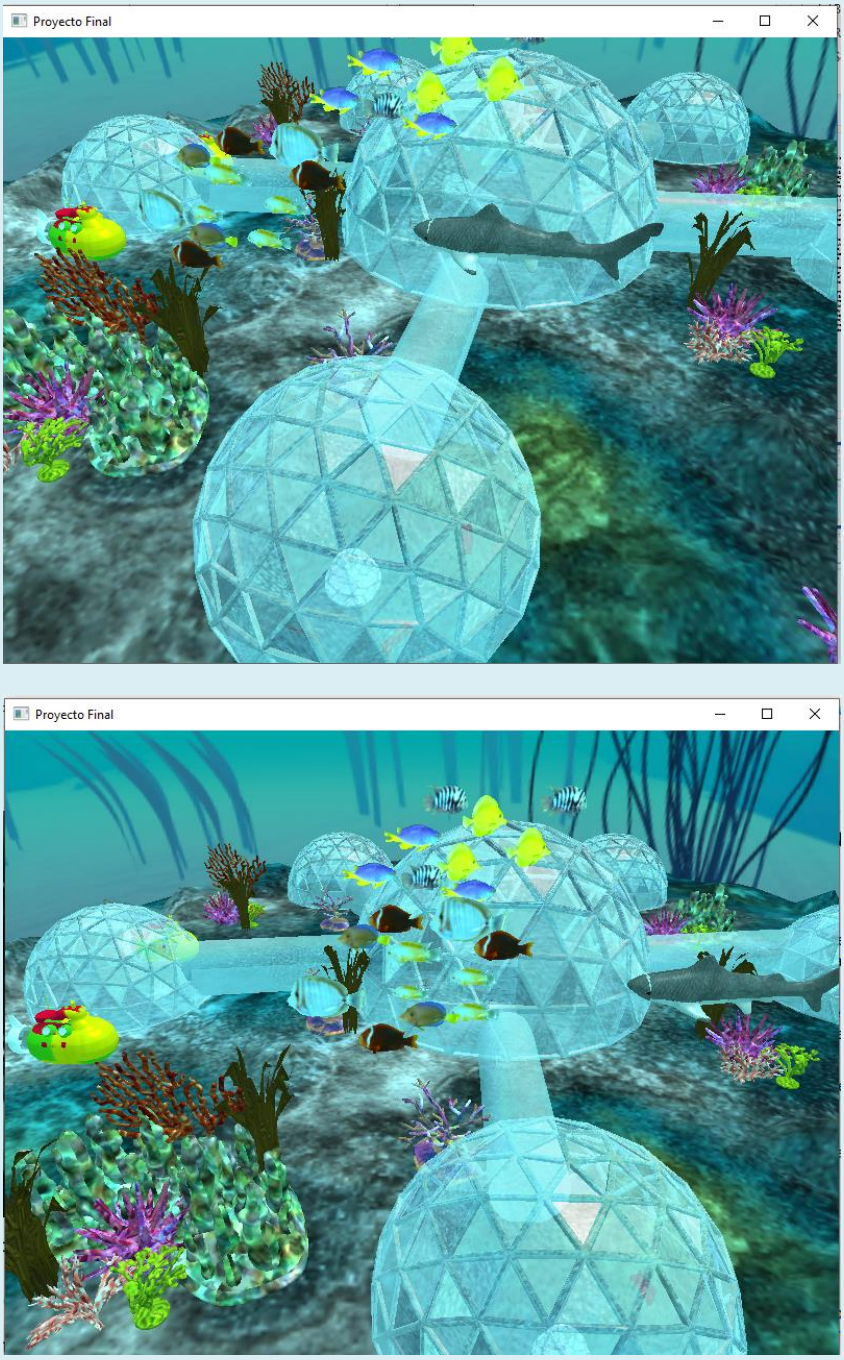


Imagen 7, 8. En esta escena se pueden denotar distintos elementos, el suelo fue texturizado adecuadamente y se alteró su morfología para tener distintas elevaciones para crear un terreno irregular, se dotó de transparencia a los domos y los caminos de conexión para naturalizar el fenómeno de un vidrio. Existe cierto reflejo de luz que se denota por los vidrios y los elementos. Se logra en buen efecto de brillo e integración

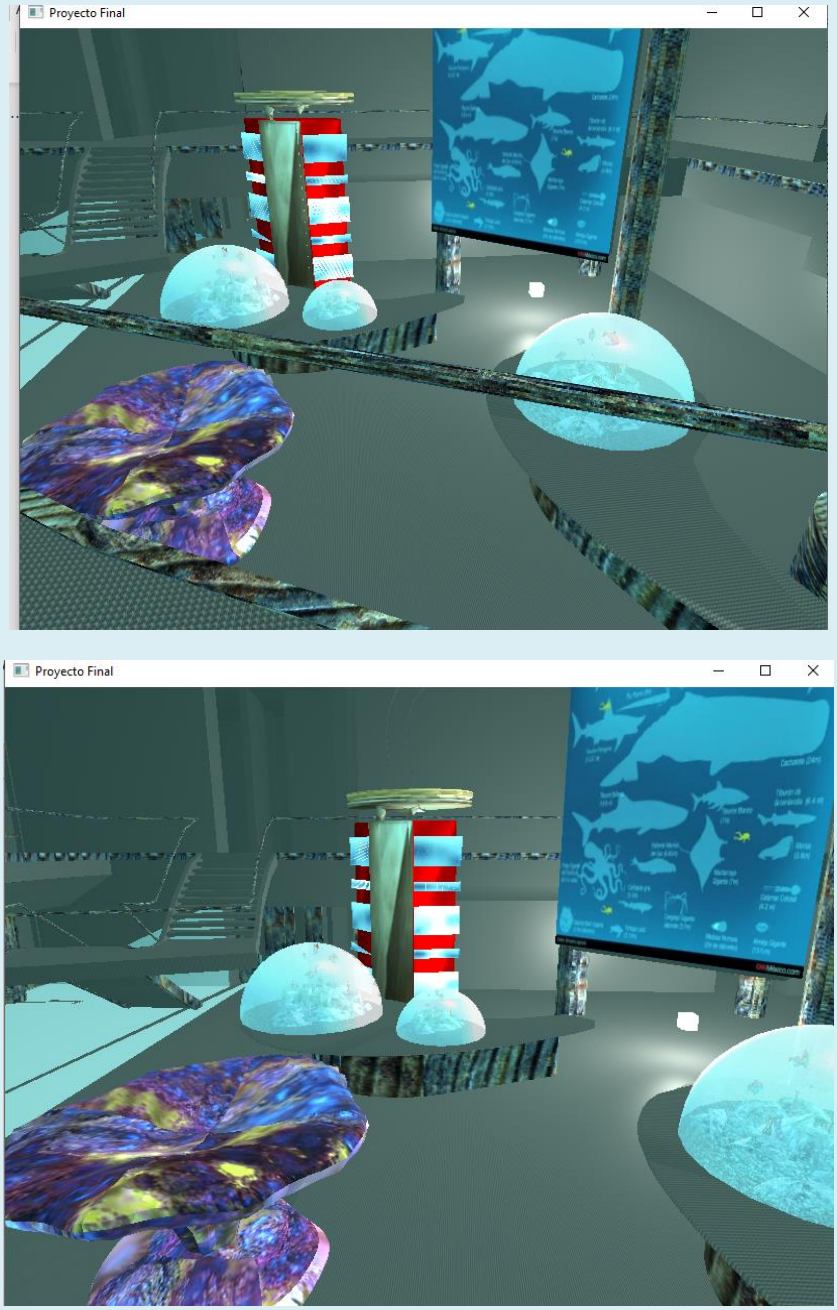


Imagen 9. Se recrearon semiesferas de transparencia que exhiben peceras, pantallas con anuncios, y elementos informativos y característicos de un acuario. Las escaleras, mesas, displays, etc se encuentran correctamente posicionados e integrados al escenario.

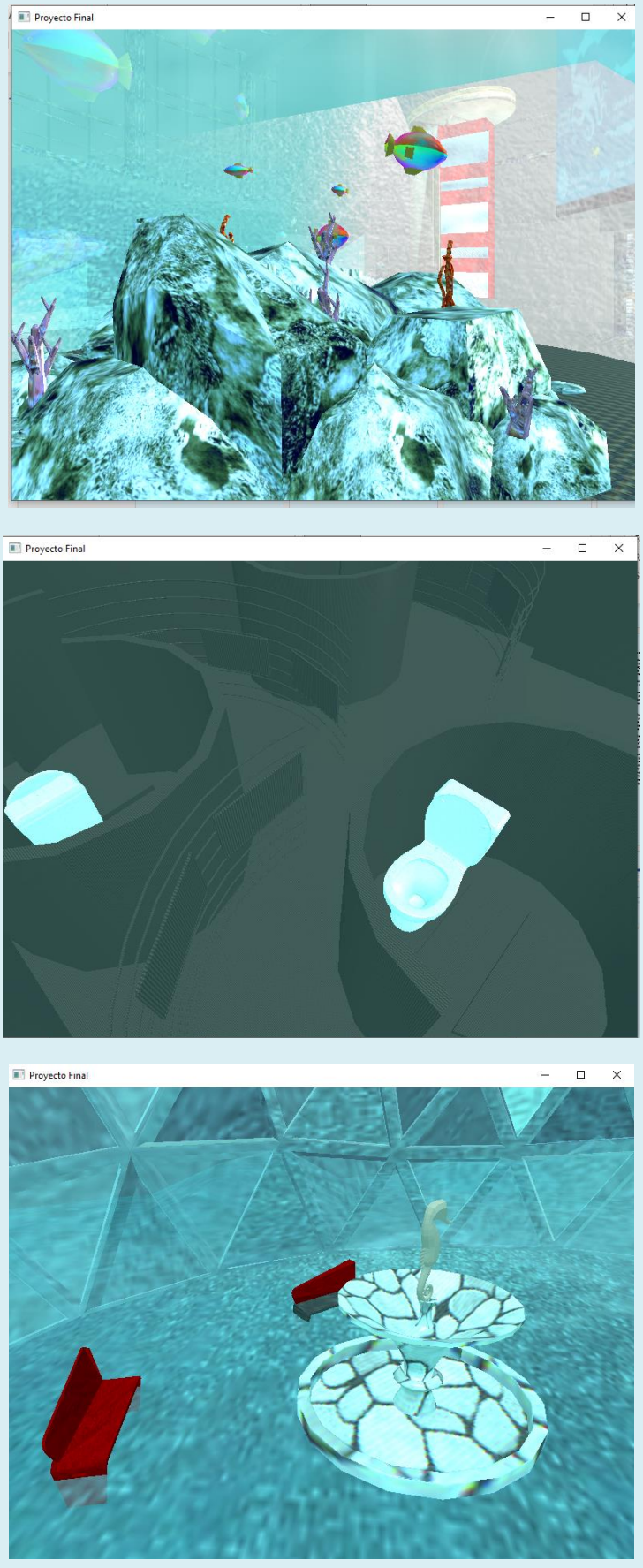


Imagen 10. Se cuidaron los elementos restantes de ambientación como bancas, fuentes, inodoros, peceras, etc.

Work Platforms.

For the development of our aquarium environment, it was necessary to work under the Microsoft Visual Studio (IDE) integrated development environment, which is used to develop computer programs, applications, and services, in our case, following OpenGL operations. which is mainly considered an API (an application programming interface) which gives us a great set of functions that we can use to manipulate graphics and images. However, OpenGL itself is not an API, but simply a specification, developed and maintained by the Khronos Group.

The OpenGL specification defines exactly what the result/output of each function should be and how it should work. Then it's up to developers like us who implement these specs to find a solution for how this feature should work. Since the OpenGL specification does not give us implementation details, actual developed versions of OpenGL may have different implementations, as long as their results conform to the specification (and thus are the same for users).

It is important to note that for the development of this Jurassic project, work was done

with the OpenGL libraries which are written in C and allow many derivations in other languages, but in essence it is still a C library.

Since many of C's language constructs don't translate as well to other higher-level languages, OpenGL was developed

with several abstractions in mind. One of those abstractions is objects in OpenGL.

Multiple objects were defined for this aquarium recreation project, following the functional requirements specifications of the previous section. An object in OpenGL is a collection of options that represents a subset of the OpenGL state. For example, we could have an object that represents the

drawing window settings; then we could set its size, how many colors it supports, etc. One could visualize an object as a C-like structure.

The nice thing about using these objects is that we can define more than one object in our application, set its options, and each time we start an operation that uses the OpenGL state, we bind the object with our preferred configuration. There are objects, for example, that act as container objects for the 3D model data (a house or a character) and every time we want to draw one of them, we link the object that contains the model data we want to draw (we first create and we configure options for

these objects). Having multiple objects allows us to specify many models and whenever we want to draw a specific model, we simply bind the corresponding object before drawing without reconfiguring all of its options.

2. Basic Environment Parameters

2.1. Window Use

The first thing we had to configure before starting to create the underwater or aquarium graphical environment was to create an OpenGL context and a

application window to draw. However, those operations

they are specific to the operating system, and OpenGL tries to abstract from these operations on purpose. This means that we have to create a window, define a context, and handle user input ourselves. Fortunately, there are quite a few libraries out there that provide the functionality we were looking for, some specifically targeting OpenGL. We use GLFW. GLFW is a library, written in C, specifically targeted at

OpenGL. GLFW gives us the basic necessities required to display things on the screen. It allows us to create a context

OpenGL, defining window parameters, and handling user input, which is sufficient for our purposes.

2.2. linkage

In order for the project to use GLFW, we needed to link the library with our project. This could be achieved by specifying that we want to use glfw3.lib in the linker configuration, but

our project still needed to know where to find glfw3.lib since we store our third-party libraries in a directory

different. Therefore, we must first add this directory to the project.

We can also tell the IDE to take this directory into account when it needs to search for a library and include files.

Our list of links includes:

$(SolutionDir)/External Libraries/GLEW/lib/Release/Win32

$(SolutionDir)/External Libraries/GLFW/lib-vc2015

$(SolutionDir)/External Libraries/SOIL2/lib

$(SolutionDir)/External Libraries/assimp/lib

2.2.2 Linker Input Parameters.

Once we set the external headers, our VisualStudio IDE can find all the necessary files, and we can finally link the libraries to the project by going to the Linker and Input tab to finish our configuration.

soil2-debug.lib;assimp-vc140-

mt.lib;opengl32.lib;glew32.lib;glfw3.lib;

3 Shaders

Shaders were used to help build the aquarium environment for this project. Shaders are small programs that sit on the GPU.

These programs are run for each specific section of the graphics pipeline. In a basic sense, shaders are nothing more than programs that transform inputs into outputs. Shaders are also very isolated programs in the sense that they are not allowed to

communicate with each other; the only communication they have is through their

entrances and exits.

The shaders used are written in the C-like GLSL language. GLSL is designed for use with graphics and contains useful features specifically targeted at vector and matrix manipulation.

Shaders always start with a version declaration, followed by a list of input and output variables, uniforms, and their main function.

The entry point of each shader is in its main function where we process any input variables and display the results in its output variables.

Shaders were used to load models, for lighting, for setting (cubemaps), and for the animations of the aquarium project.

They are listed in a specific folder called Shaders and their programming is native to their function.

4. Loading Models

4.1 ASSIMP

For our aquarium environment we cannot manually define all the vertices, normals and texture coordinates in complicated ways.

Instead, what we want is to import these models into the application; models carefully designed by us in graphic tools

like 3DS Max or Maya. These 3D modeling tools allow us to create complicated shapes and apply textures to them through a process

called uv mapping. The tools then automatically generate all vertex coordinates, vertex normals, and coordinates

texture while exporting them to a model file format that we can use. In this way, we have an extensive set of tools to create high-quality models without having to worry too much about the technical details. All aspects

technical data are hidden in the exported model file. However, we as graphics programmers have to worry about these technical details.

A very popular model import library that was implemented for this project is Assimp, which stands for Open Asset Import Library. Assimp can import dozens of different model file formats (and export to some as well) loading all model data into Assimp's generalized data structures. As soon as Assimp has loaded the model, we can retrieve all the data we need from the Assimp data structures. Because Assimp's data structure stays the same regardless of the type of file format we import,

it abstracts us from all the different file formats out there.

When you import a model via Assimp, you load the entire model into a scene object that contains all the imported model/scene data. Assimp then has a collection of nodes where each node contains indices of data stored in the scene object where each node can have any number of children.

4.2 Mesh

With Assimp we were able to load many different models into the application, but once loaded they are all stored in Assimp's data structures. What we eventually need is to transform that data to a

format that OpenGL understands so that we can render objects. A mesh should need at least one set of vertices, where each vertex contains a position vector, a normal vector, and a vector of

texture coordinates. A mesh must also contain indexes for the indexed drawing and material data in the form of textures (maps

diffuse/specular)

Thanks to the constructor, we got huge lists of mesh data that we can use for rendering. We need to configure the buffers

and specify the vertex shader layout via vertex attribute pointers so that finally with the last function

What we need to define for the Mesh class to be complete is its 'Draw' function. Before rendering the mesh, we first want to bind the appropriate textures before calling glDrawElements. However, this is somewhat difficult as we don't know up front how many textures (if any) the mesh has and what type they can be.

4.3 Patterns

From this point in the project, we started creating the actual uploading and translation code with Assimp. The goal is to create another class that represents a model in its entirety, that is, a model that contains multiple meshes, possibly with multiple textures. A dinosaur, for example, that keeps claws, teeth, eyes, and skin could still be loaded as a single model. We'll load the model via Assimp and translate it into various mesh objects we've created.

Given that we assume that the texture file paths in the model files are local to the actual model object, Example. in the same directory as the location of the model itself.

So we can simply concatenate the texture location string and the directory string we retrieved earlier (in the

loadModel function) to get the full texture path (that's why the GetTexture function needs the directory string as well).

Some models sourced from the internet for this project used absolute paths for their texture locations, which didn't work when

time to get to this point. In that case, we manually edit the file to use local paths for the textures (if possible). I know

they replaced the textures and started a process of adaptation to the model that is described below.

2.1 TECHNICAL GUIDE:
ELABORATION AND ADAPTATION PROCESS

Based on the schedule of activities and the completion of the Planning and Design phase, the next phase of Implementation was reached, where inputs, changes, and elimination of elements were collaboratively added to adjust to the agreed delivery in the requirements analysis.
Among the modular programming, work metrics were established to be able to work with the Jurassic environment.

Among these items, processes were defined to add objects;

1. Import or creation of the model.

Most of the models in the project are their own intellectual material, but around 45% were obtained from platforms that offer various graphic models with different licenses for use.

The models obtained by being a free license did not contain textures or materials. For this point, parameters were established for its integration into the scenario.

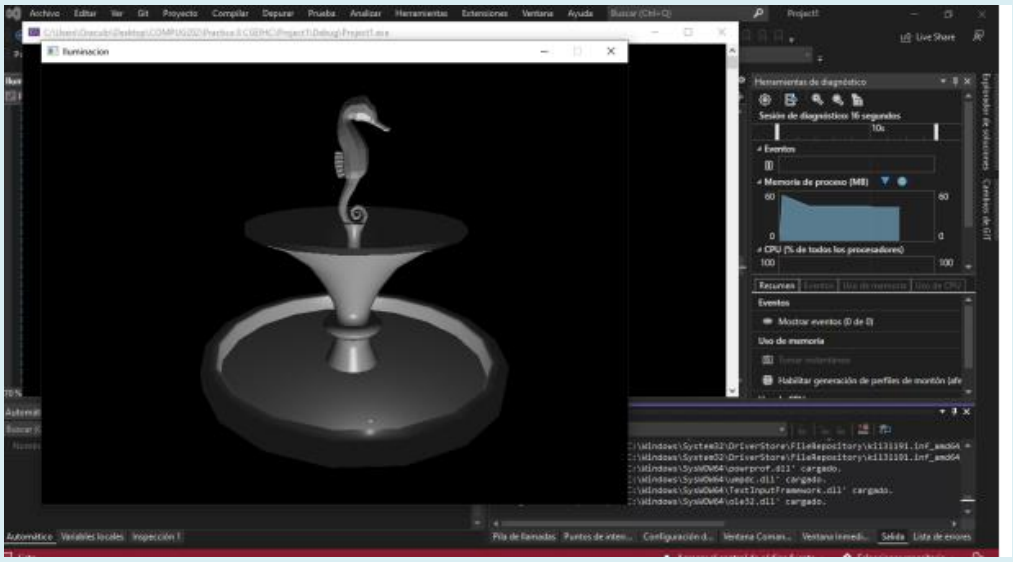


Image 1. Graphical software interface view; the importation of the model without textures is appreciated, and with basic transformations for its coupling to the workspace.

2. Selection of Textures and adaptation of materials.

The models selected and that were filtered for the final phase of introduction to the scenario, were revisited for the taking of textures. The best quality of materials was sought and the images were worked with the GIMP software, to be able to be used as a texture and to shape the uv map. In order to create a better setting for the aquatic environment.

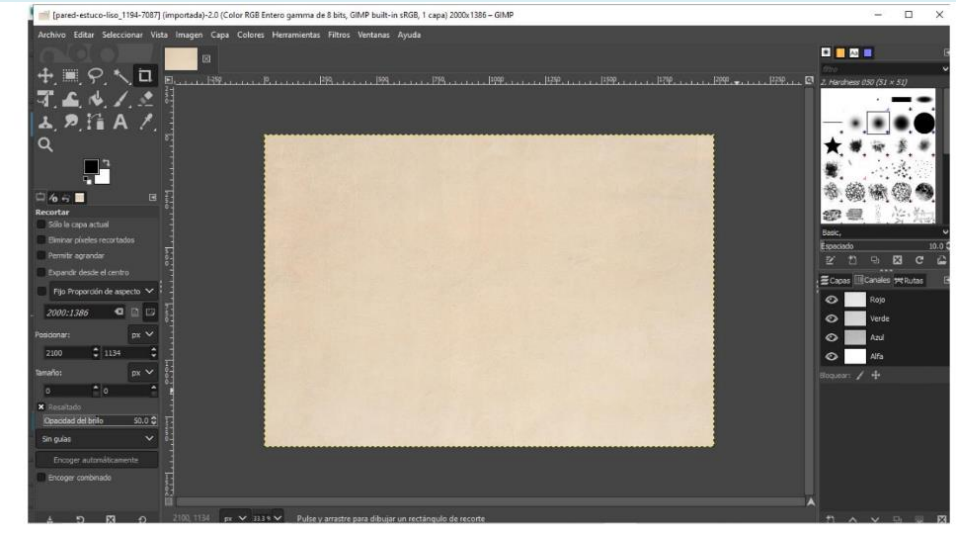


Image 2. GIMP interface view, you can see the import of the image for the texture, the platform's own tools are used to adapt the texture, in size and quality.

6. Convergence of materials and completion of modeling

The materials are attached to the model, basic transformations of the object are reviewed again to adapt its position within the scenario and finally, it is imported for its later incorporation into the integrated development environment that works on the complete integration of the project.

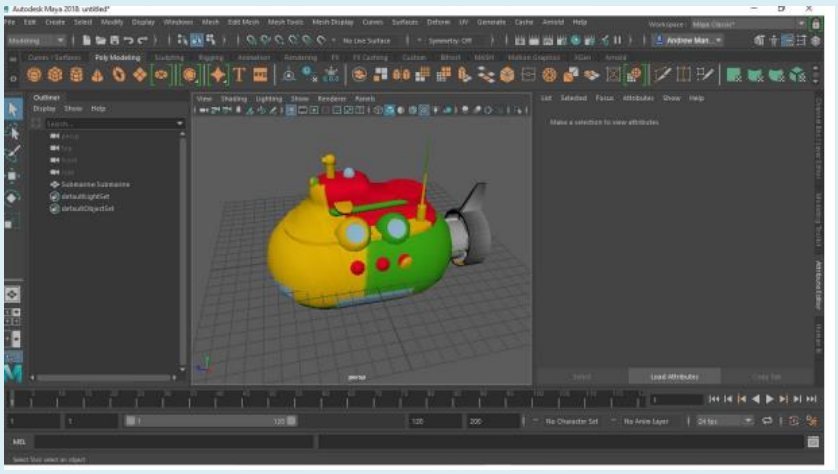


Image 3. Graphical software interface view; The final shape of the model with integrated textures and with the basic positioning transformations can be appreciated.

3.Integration of the model to the final scenario

Finally, the workspace is exchanged to continue with the integration and coupling to the final deliverable. It ensures that the new model does not affect the stability of the project, that there are no unwanted positions, or that the integration of the component is as desired.

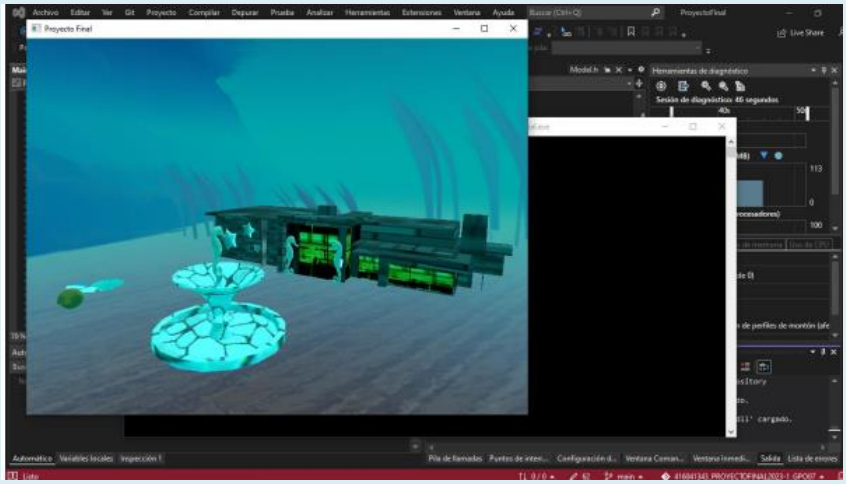


Figure 4. Visual Studio software interface view; the final shape of the model can be seen with the other models also finished.

6. Version Control

New changes are saved and updated in the central repository; Comments are included in the documentation, such as the date of publication, the change made, and the functionality.

The GitHub sharing tool is used to set a version control, backup, and adjoining parameter.

2.2 TECHNICAL GUIDE: USE OF AMBIANCE: LIGHTING

For this section we put all the uniform variables for the types of lights we have; directional, ambient, diffuse and specular. We have to manually configure and index them. We must fit our own PointLight structure in the matrix to set each uniform variable. We can even define light types as classes and setting their values there, or using a more efficient uniform approach

```
// == ============================
// Directional light
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), -0.8f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.4f, 0.4f, 0.4f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.5f, 0.5f, 0.5f);


// Point light 1
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositions[0].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), 0.05f, 0.05f, 0.05f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), LightP1.x, LightP1.y, LightP1.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), LightP1.x, LightP1.y, LightP1.z);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 0.032f);



// Point light 2
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].position"), pointLightPositions[1].x, pointLightPositions[1].y, pointLightPositions[1].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].ambient"), 0.05f, 0.05f, 0.05f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].diffuse"), 1.0f, 1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].specular"), 1.0f, 1.0f, 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].quadratic"), 0.032f);
```

70 % No se encontraron problemas. Línea: 545 Carácter: 117 Columna: 123 TABULACIONES CRLF

BY PLAYING, ADAPTING AND MODIFYING THESE PARAMETERS WE CAN ACHIEVE DIFFERENT ENVIRONMENT PHENOMENA IN OUR FINAL SCENARIO, FOR EXAMPLE;WE CAN CREATE AN ENVIRONMENT WITH CLEAR LIGHT AND CRYSTAL CLEAR WATER, OR ANOTHER WITH LESS CLARITY AND A GREATER DEPTH ENVIRONMENTAL EFFECT

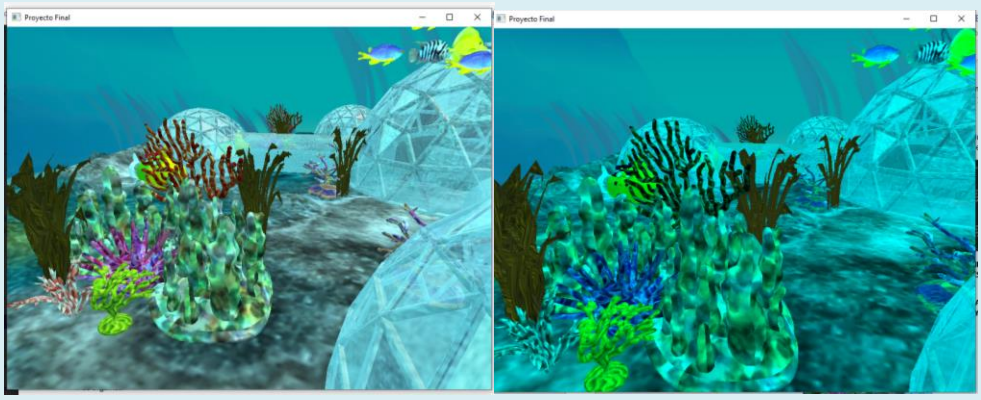


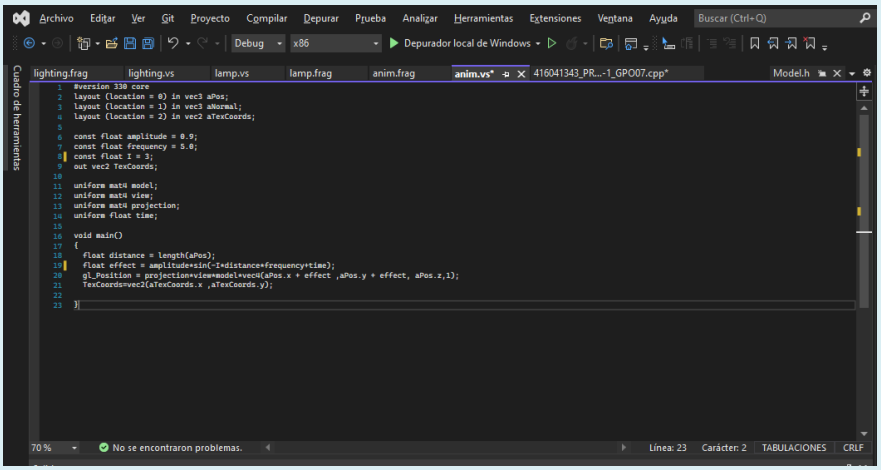
Image 3,4: (left) Image with higher ambient light values for the first component creating a clearer and more defined appearance of certain textures, (right) image with lower ambient values, defining a more aquatic environment.

2.3 TECHNICAL GUIDE: USE OF ANIMATIONS

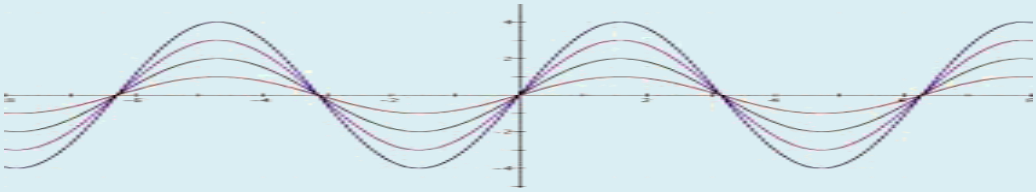
For our first animation (alga) we need to provide movement to our scene creating the sensation of movement and active flow of current, this can be achieved by defining a typical movement associated with tidal movements.

```
Anim.Use();
modelLoc = glGetUniformLocation(Anim.Program, "model");
viewLoc = glGetUniformLocation(Anim.Program, "view");
projLoc = glGetUniformLocation(Anim.Program, "projection");
tiempo = glfwGetTime() * speed;
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
model = glm::mat4(1);
glUniform1f(glGetUniformLocation(Anim.Program, "time"), tiempo);

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));

Alga.Draw(Anim);
glBindVertexArray(0);
```

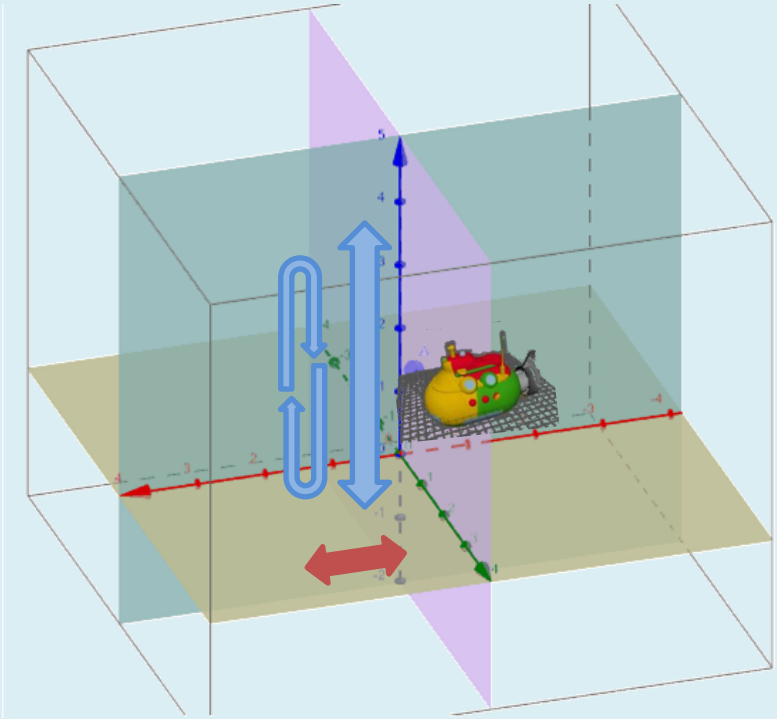
We define our vertex shader with the support constants inside the function, the uniform variables of our program, so that our alga that is an element (vertical) has the proper movement in the desired planes we will need to operate it in the x and y components.



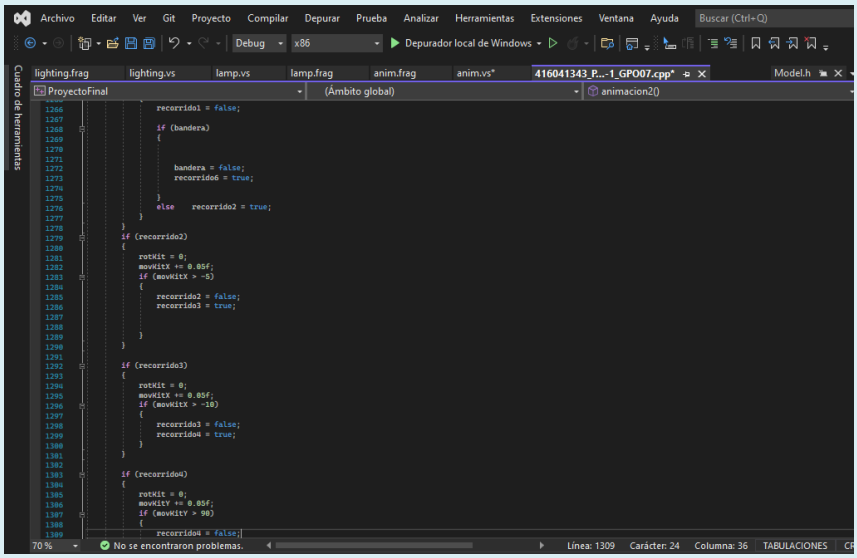
A moderately valued amplitude is modified so as not to deform the elements and continue showing an adequate movement. This is achieved by varying the parameter and testing the element that is following a trajectory similar to the following.



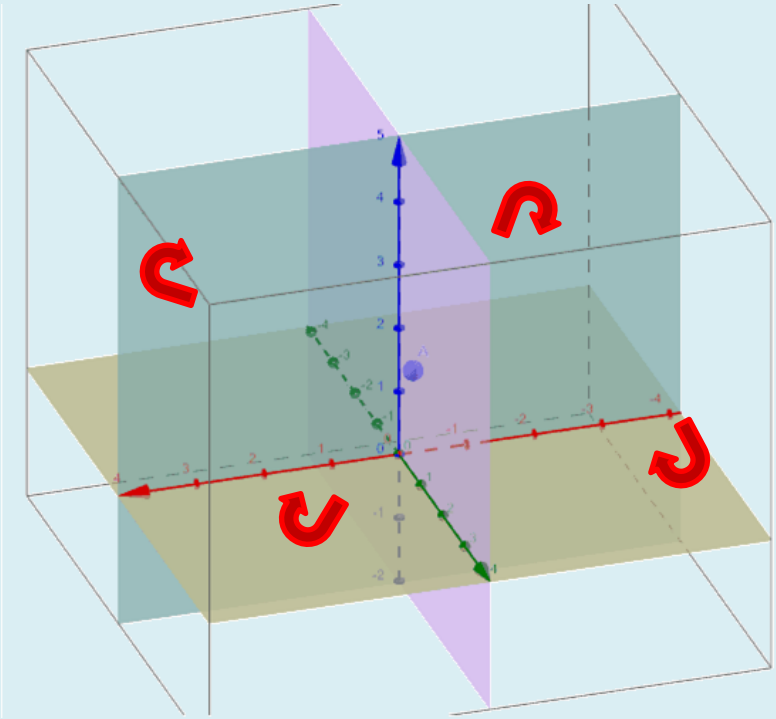
For the animation of our submarine, we carry out the boarding, ascent and descent sequence in a cyclical way, once it is correctly positioned within the environment and the animation is activated, the submarine performs the actions determined in the following plane

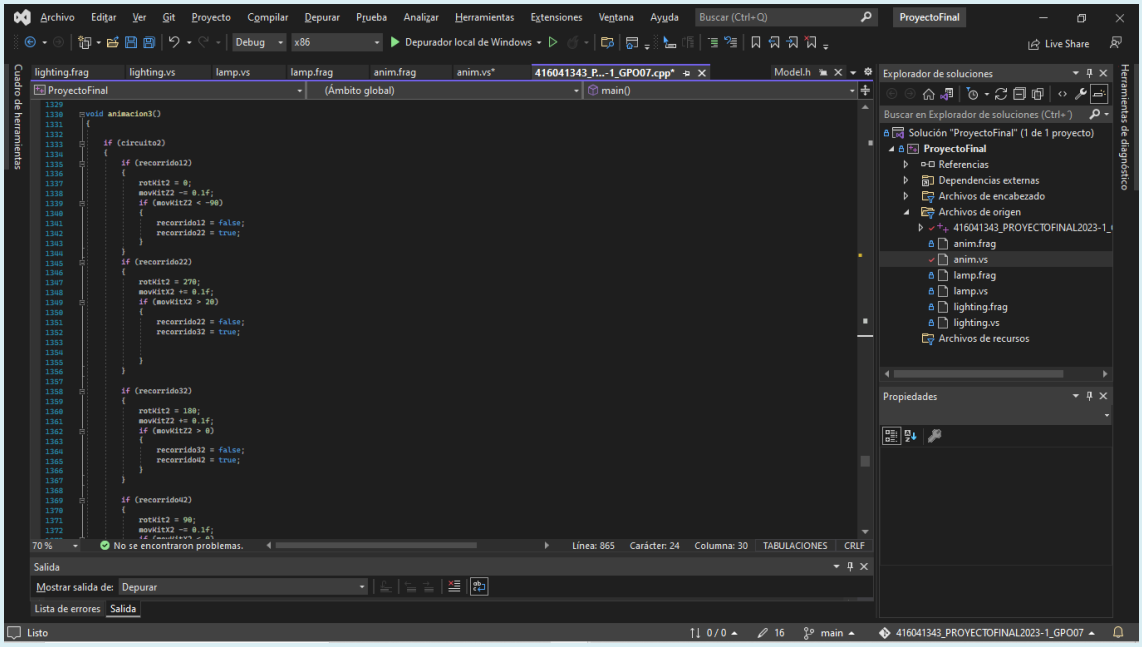


```
model = glm::mat4(1);
model = glm::translate(model, PosIni + glm::vec3(movKitX, movKitY, 0));
model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Submarino.Draw(lightingShader);
glBindVertexArray(0);
```

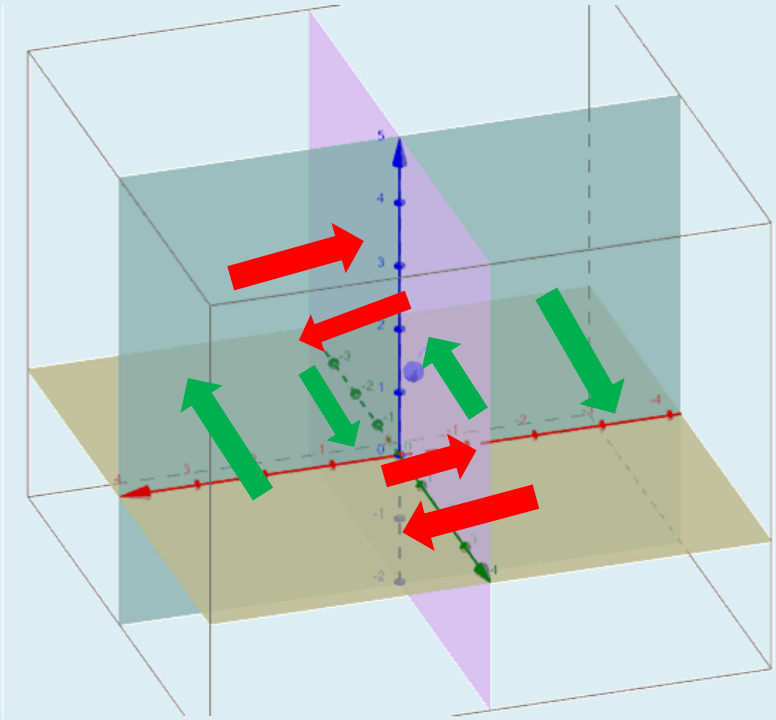


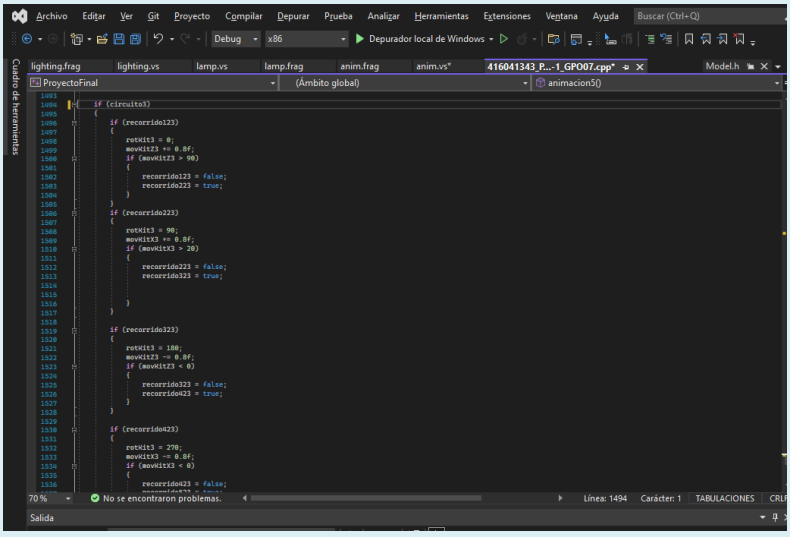
For the animation of our shark we propose an infinite circuit around the entire stage, this includes the rotation of its body to go according to the proposed direction, and at a suitable defined speed. The planes of its movement coincide with the defined components.



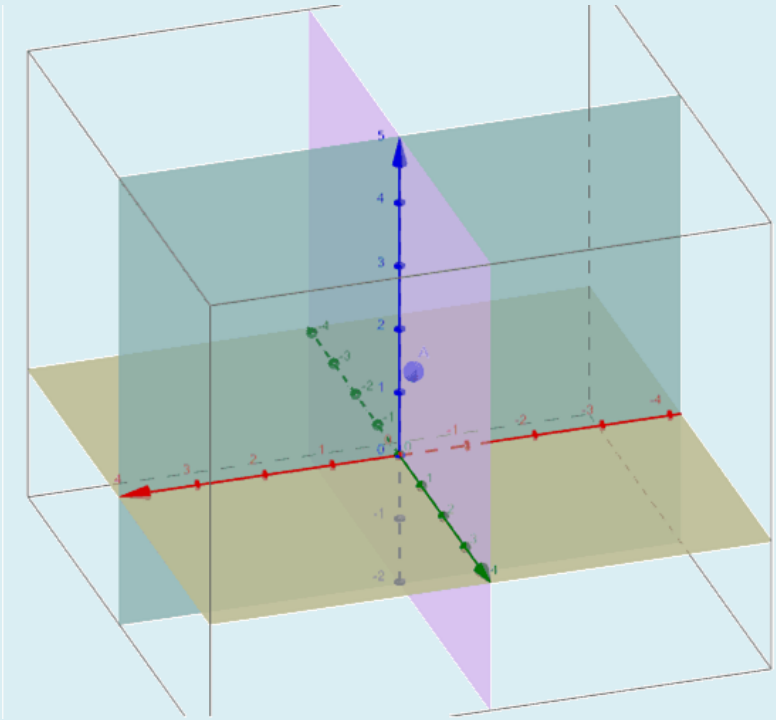


For the animation of the fish there are two circuits, the first in a clockwise direction like our previous animation and the next one in an anticlockwise direction that would be the same for this animation, the rotations of the bodies of the fish are appropriate for both directions and are in an infinite sequence through the stage once started, plus this sequence can be stopped and resumed at any time

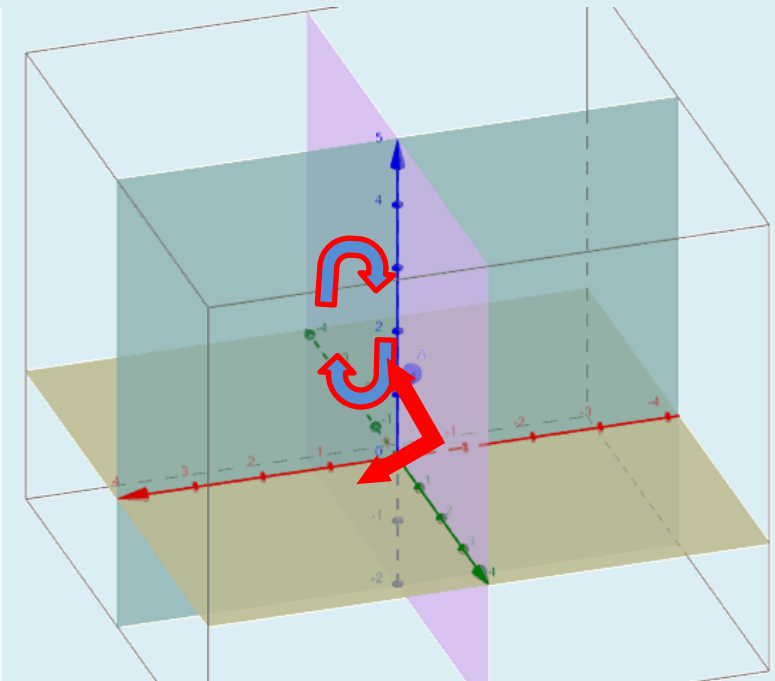




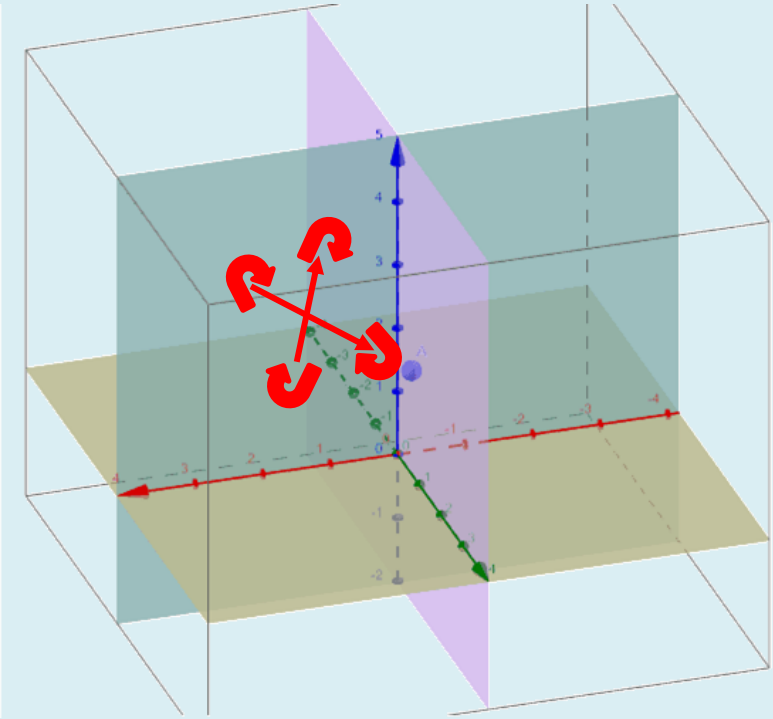
For the last animation of the antenna, a characteristic movement was given to it with a simple rotation within one of the spaces destined for its setting, this movement is cyclical and can stop at any time and resume again



For the animation of the vault, a characteristic movement was provided to it with a rotation of the crank within one of the spaces destined with a 360 degree turn, which at the end has a characteristic opening of a door for its setting, this movement is cyclical and can be stopped at any time and resumed again



For the animation of the penguin area, a small route or characteristic movement was carried out with a triangulation around one of the spaces destined for its setting, this movement is cyclical and can be stopped at any time and resumed again.



2.4 TECHNICAL GUIDE:

AMBIANCE AND ADEQUACY

For this section, some samples will be placed with the appropriate models and the textures defined for the correct setting of the proposed scenario. Different computer graphics techniques and summarization of concepts navigated through the agenda were used.

Image 5. More than 6 different models can be observed, each one with an adequate texture just to recreate the different spaces within the stage, the corals fulfill a decorative function, and the algae have a greater relevance by giving movement to the stage.

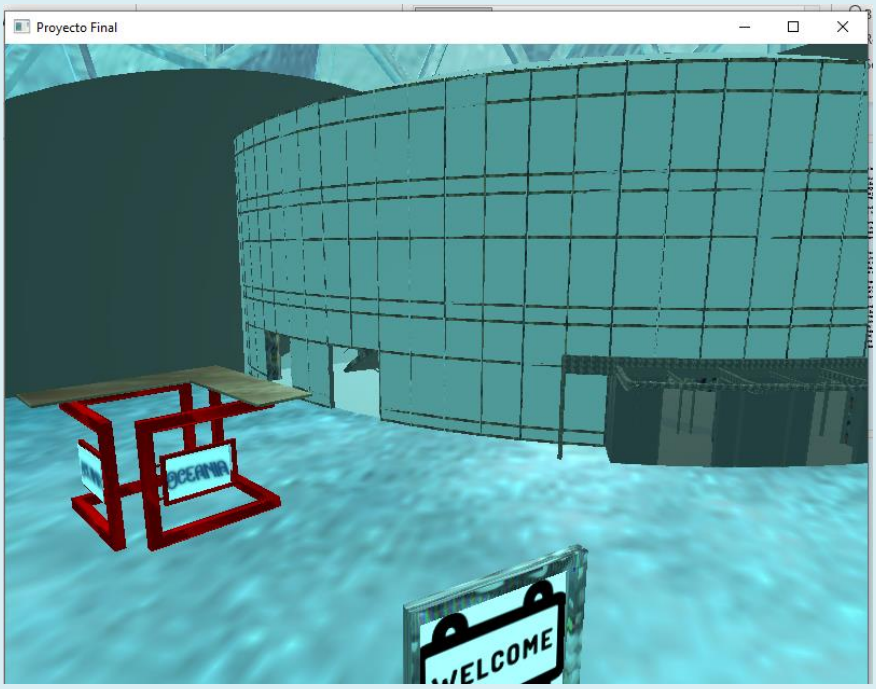


Image 6. The stage is a homologation of different fused components, different textures were chosen to model it and elements were placed at the entrance.

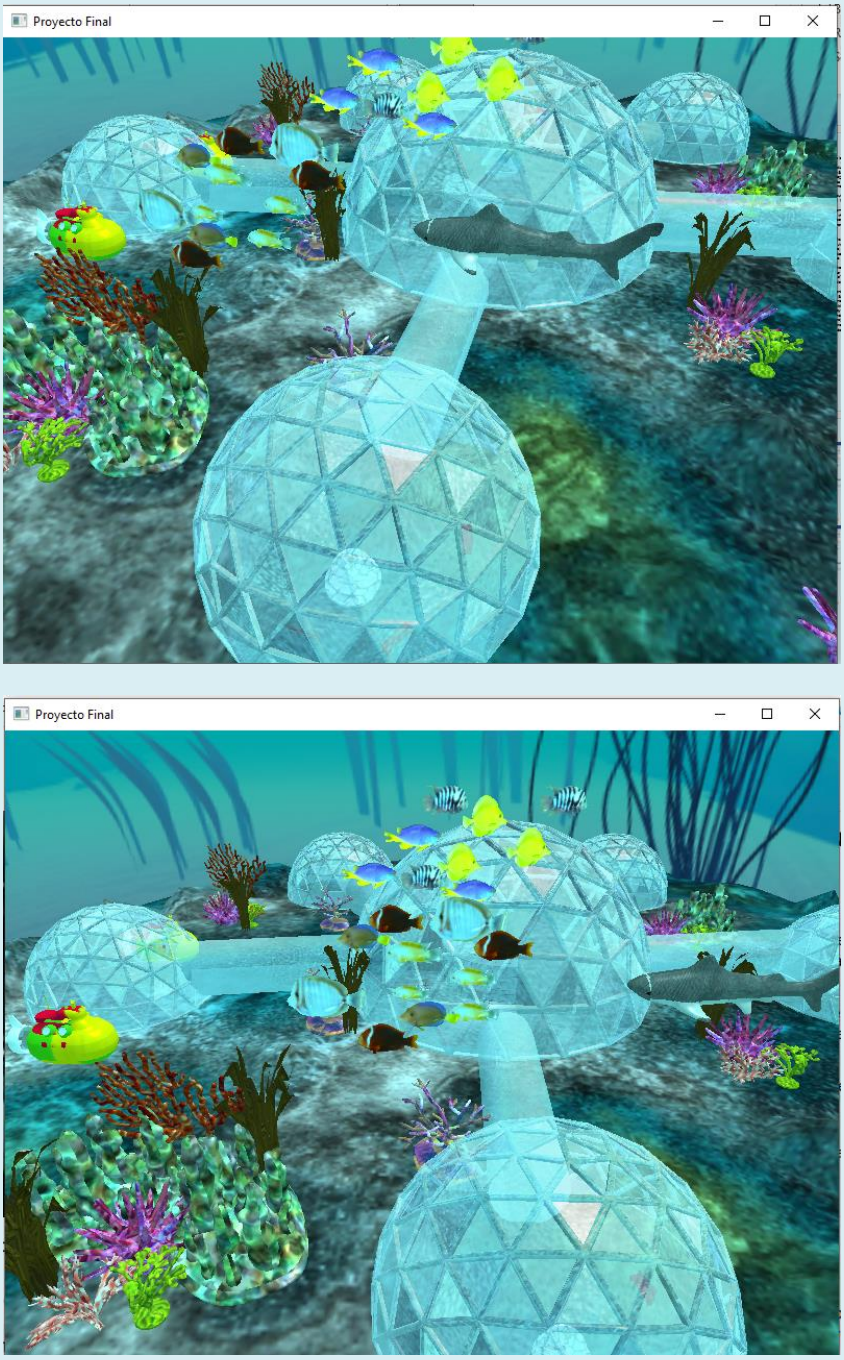


Image 7, 8. In this scene, different elements can be denoted, the ground was adequately textured and its morphology was altered to have different elevations to create an irregular terrain, transparency was given to the domes and the connecting paths to naturalize the phenomenon of a glass There is a certain reflection of light that is denoted by the glass and the elements. It is achieved in good gloss and integration effect

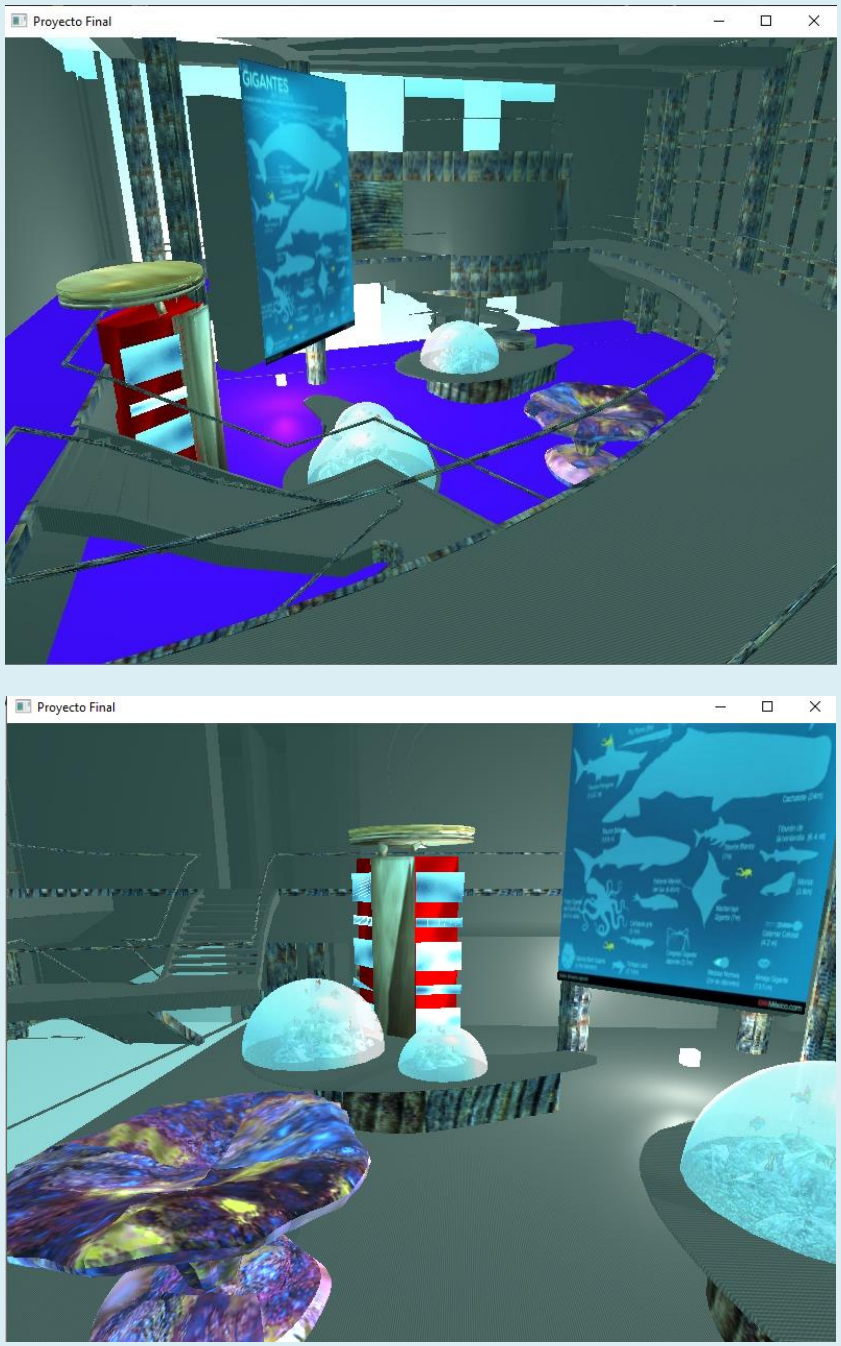


Image 9. Transparency semi-spheres that exhibit fish tanks, screens with advertisements, and informative and characteristic elements of an aquarium were recreated. The stairs, tables, displays, etc. are correctly positioned and integrated into the stage.

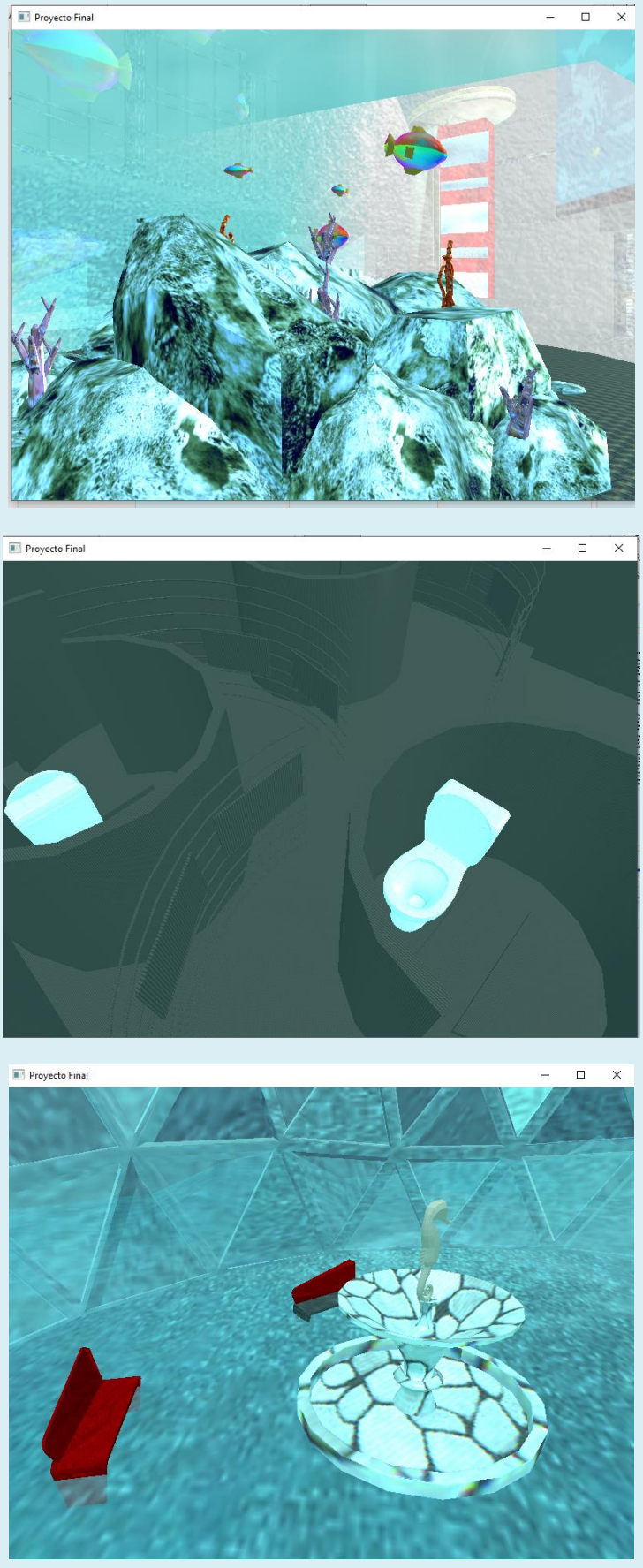


Image 10. The remaining ambience elements such as benches, fountains, toilets, fish tanks, etc. were taken care of.

CONCLUSIONES

Sánchez Manjarrez Andrew

We were able to meet the objectives of this project, the topics learned throughout the Computer Graphics course and human-computer interaction were comprehensively explored. There is great satisfaction when homogenizing the knowledge and experience of multiple topics related to computer graphics from the loading of models, through texturing, hierarchy of elementary operations, to the complexity of the setting to achieve the user's immersion naturally in a graphic environment.

Essential things such as lighting, animations or the correct choice of materials show us the level of detail that we often go unnoticed in our environment when interacting with the elements of our daily life, and that are full and express a great wealth of elements. . They mostly have a purpose, an intention to persuade us in a story or simply to provide any scene with context. As computer engineers it is important to include these tools under the complexity of engineering dissection, to denote each component that makes these recreations possible, to be able to shape these functions and fit them into any graphical design, development and implementation.

Fuentes Viveros Adan Emiliano

My experience in develoment of the project is not so much that I desire because could be better that the actual project if I pose the knowledge and the time but I decide to work little bit late and that makes a product that I don’t believe is enough of my part for develoment, I mean, I stay in the middle of the nigh doing some upgrates and add part of my project to laboratory and that makes not so difficult move the models from one project to the another, that really helps me.

DOCUMENTACIÓN

LearnOpenGL - OpenGL. (2017). Learn OpenGL. Retrieved november 18, 2022, from https://learnopengl.com/Getting-started/OpenGL

LearnOpenGL - Shaders. (2017). Learn OpenGL. Retrieved november 18, 2022, from https://learnopengl.com/Getting-started/Shaders

Using the TurboSquid Royalty Free License. (2021, April 13). TurboSquid Blog. november 18, 2022, from https://blog.turbosquid.com/royalty-free-license/

Microsoft. (2022, January 12). *Introducción a*. Visual Studio. Retrieved november 18, 2022, from https://visualstudio.microsoft.com/es/vs/getting-started/

LearnOpenGL - Shaders. (2017). Learn OpenGL. november 18, 2022, from https://learnopengl.com/Getting-started/Shaders

LearnOpenGL - Model. (2017). Learn OpenGL. november 18, 2022, from https://learnopengl.com/Model-Loading/Model

LearnOpenGL - Assimp. (2017). Learn OpenGl. Retrieved november 18, 2022, from https://learnopengl.com/Model-Loading/Assimp

Términos y condiciones para los modelos de CGTrader

Royalty Free LicenseEC
November 5, 2020
If the model is under Royalty Free license, you can use it as long as it is incorporated into the product and as long as the 3rd party cannot retrieve it on its own in both digital and physical form. You cannot resell the model you bought in its digital form and you cannot resell it in its printed form as a separate, single item.
Product may not be sold, given, or assigned to another person or entity in the form it is downloaded from the Site.
The Buyer's license to Product in this paragraph is strictly limited to Incorporated Product. Any use or republication, including sale or distribution of Product that is not Incorporated Product is strictly prohibited. For illustration, approved distribution or use of Product as Incorporated Product includes, but is not limited to:
as rendered still images or moving images; resold as part of a feature film, broadcast, or stock photography;
as purchased by a game's creators as part of a game if the Product is contained inside a proprietary format and displays inside the game during play, but not for users to re-package as goods distributed or sold inside a virtual world;
as Product published within a book, poster, t-shirt or other item;
as part of a physical object such as a toy, doll, or model.
If you use any Product in software products (such as video games, simulations, or VR-worlds) you must take all reasonable measures to prevent the end user from gaining access to the Product. Methods of safeguarding the Product include but are not limited to:
using a proprietary disc format such as Xbox 360, Playstation 3, etc.;
using a proprietary Product format;
using a proprietary and/or password protected database or resource file that stores the Product data;
encrypting the Product data.
Without prejudice the information above, the Seller grants to the Buyer who purchases license rights to Product and uses it solely as Incorporated Product a non-exclusive, worldwide, license in any medium now known or hereinafter invented to:
reproduce, post, promote, license, sell, publicly perform, publicly display, digitally perform, or transmit for promotional and commercial purposes
use any trademarks, service marks or trade names incorporated in the Product in connection with Seller material;
use the name and likeness of any individuals represented in the Product only in connection with Your material.
The resale or redistribution by the Buyer of any Product, obtained from the Site is expressly prohibited unless it is an Incorporated Product as licensed above.
We also always suggest buyers discuss the usage with the seller and gain their approval, this would make you a hundred percent sure that you can proceed with your decision of using the purchase.

Editorial License

Editorial license gives a permission to use the product only in an editorial manner, relating to events that are newsworthy, or of public interest, and may not be used for any commercial, promotional, advertising or merchandising use.
In a few instances, you may otherwise have the rights to IP in content that is under Editorial license. For instance, you may be the advertising agency for a brand/IP owner or you may be the brand/IP owner itself purchasing user generated content. Given you have the rights clearance through other means, you may use the content under Editorial License commercially. Every user failing to comply with Editorial Use restrictions takes the responsibility to prove the ownership of IP rights.
For users, who are not brand/IP owners or official affiliates, the restrictions of Editorial-licensed content usage include, but are not limited to, the following cases:
Products may not be used on any item/product created for resale such as, commercials, for-profit animations, video games, VR/AR applications, or physical products such as a merchandise or t-shirt.
Products may not be used as part of own product promotional materials, billboard, trade show or exhibit display.
The product may not be incorporated into a logo, trademark or service mark. As an example, you cannot use Editorial content to create a logo design.
Products may not be used in any insulting, abusive or otherwise unlawful manner.
The product may not be used for any commercial related purpose.

Términos y condiciones para los modelos de TurboSquid

Royalty Free License
This is a legally binding agreement between licensee (“you”), and TurboSquid regarding your rights to use Stock Media Products from the Site under this license. “You” refers to the purchasing entity, whether that is a natural person who must be at least 18 years of age, or a corporate entity. The rights granted in this agreement are granted to the purchasing entity, its parent company, and its majority owned affiliates on a “royalty free” basis, which means that after a Purchase, there are no future royalties or payments that are required. Collectively, these rights are considered “extended uses”, and are granted to you, subject to applicable Editorial Use Restrictions described below. The license granted is wholly transferable to other parties so long it is in force and not terminated, otherwise violated, or extinguished, as set forth herein. This agreement incorporates by reference the Terms of Use as well as the Site’s policies and procedures as such.

6. Creations of Imagery.
Permitted Uses of Creations of Imagery. Subject to the following restrictions, you may use Creations of Imagery within news, film, movies, television programs, video projects, multi-media projects, theatrical display, software user interfaces; architectural renderings, Computer Games, virtual worlds, simulation and training environments; corporate communications, marketing collateral, tradeshow promotional items, booth decorations and presentations; pre-visualizations, product prototyping and research; mobile, web, print, television, and billboard advertising; online and electronic publications of blogs, literature,

social media, and email campaigns; website designs and layouts, desktop and mobile wallpapers, screensavers, toolbar skins; books, magazines, posters, greeting cards; apparel items, brochures, framed or printed artwork, household items, office items, lenticular prints, product packaging and manufactured products.

Restrictions on Permitted Uses of Creations of Imagery.

a. Stock Media Clearinghouse. You may NOT publish or distribute Creations of Imagery through another stock media clearinghouse, for example as part of an online marketplace for photography, clip art, video, or design templates.

b. Promotional Images. Images displayed for the promotion of Stock Media Products, such as preview images on the Stock Media Product's Product Page ("Promotional Images"), may be used in Creations of Imagery, provided that the Stock Media Product itself has been Purchased and subject to the following restrictions:

i. You may NOT use a Promotional Image that has any added element which is not included as part of the Stock Media Product. An example of this type of restricted use is if the Stock Media Product contains a 3D model of an airplane, and there is a Promotional Image of that airplane rendered over a blue sky; however, the blue sky image is not included as part of the Stock Media Product. Other prohibited examples include use of Promotional Images from movies or advertisements that may have used Stock Media Product.

ii. You may NOT use any Promotional Image that has a logo, mark, watermark, attribution, copyright or other notice superimposed on the image without prior approval from TurboSquid Support.

c. Business Logos. You may NOT use Imagery in any Creation that is a trademark, servicemark, or business logo. This restriction is included because the owners of these types of Creations typically seek exclusivity on the use of the imagery in their Creation, which is incompatible with the non-exclusive license granted to you under this agreement.

7. Creations of Computer Games and Software

Permitted Uses in Creations of Computer Games and Software. Subject to the following restrictions, you may include Stock Media Products in Creations of Computer Games, virtual worlds, simulation and training environments; mobile, desktop and web applications; and interactive electronic publications of literature such as e-books and electronic textbooks.

Restrictions on Permitted Uses of Stock Media Products in Creations of Games and Software.

a. Interactivity. Your inclusion of Stock Media Products within any such Creation is limited to uses where Stock Media Product is contained in an interactive experience for the user and not made available outside of the interactive experience. Such a permitted example of this use would be to include a 3D model of human anatomy in a medical training application, in a way that the 3D model or its environment may be manipulated or interacted with.

b. Access to Stock Media Products. You must take all reasonable and industry standard measures to prevent other parties from gaining access to Stock Media Products. Stock Media Products must be contained in proprietary formats so that they cannot be opened or imported in a publicly available software application or framework, or extracted without reverse engineering. WebGL exports from Unity, Unreal, Lumberyard, and Stingray are permitted. Any other open format or format encrypted with decryptable open standards (such as an encrypted compression archive or other WebGL programs not listed here) are prohibited from using Stock Media Products. If your Creation uses WebGL and you are not sure if it qualifies, please contact use@turbosquid.com and describe your Creation in detail