



Descripción Proyecto

Museo de 3 Grandes Civilizaciones Antiguas de México

Nombres:

Barrionuevo Pérez Daniel Alejandro

No. de cuenta: 316303116

Grupo:03

Bustamante Colín Israel

No. de cuenta: 316193384

Grupo:04

Martínez Bautista Victor Eduardo

No. de cuenta: 316255637

Grupo:03

Profesor de Teoría: Ing. Arturo Pérez de la Cruz

Grupo Teoría: 01

Profesor de Laboratorio: Ing. Luis Sergio Valencia Castro

Fecha de entrega: 16 de noviembre de 2023

Propósito del proyecto

Agrupar todos los elementos de aprendizaje de la materia de Computación Gráfica e interacción Humano-computadora en un proyecto compilado funcional, que, a su vez, demuestre el uso de habilidades básicas, de modelado geométrico, jerárquico, texturizado, iluminación, y teóricas.

Objetivos

1. Que el alumno demuestre la aplicación de los conceptos adquiridos en clase y complementar su formación profesional, al planear, desarrollar, implementar y presentar un proyecto en equipo.
2. Entregar un manual de usuario en formato PDF, que deberá contener capturas de pantalla que ejemplifiquen cada elemento del manual.
3. 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.

Requerimientos de alto nivel

Requerimientos Técnicos:

- Evidencias en plataformas colaborativas de software; para el desarrollo del software pudiendo ser: (Jira, Trello, Git, Github).
- Deberán utilizarse técnicas de modelado geométrico, modelado jerárquico, importación de modelos, y texturizado de escenarios.

Restricciones

Al proyecto se agregarán al menos cinco (5) elementos con animaciones complejas diferentes, las cuales deberán estar aprobadas previamente en el documento de propuesta de proyecto. Al menos una de esas animaciones deberá ser por la técnica de KeyFrames.

Queda a criterio del alumno los objetos a animar, pero deben tener relación con el escenario que se está construyendo. PROHIBIDO ocupar las animaciones creadas durante las sesiones de laboratorio.

Resumen de cronograma

Duración estimada: 2 meses

Tarea	Plazo	S1	S2	S3	S4	S5	S6	S7	S8
Creación de modelos	3								
Encargado	Daniel Alejandro Barrionuevo								
Creación del museo	3								
Encargado	Israel Bustamante								
Elaboración del manual	2								
Encargado	Victor Eduardo Martinez								
Elaboración de conclusiones	1								
Encargados	Daniel, Israel, Victor								

Resumen de costos

Presupuesto estimado		
Licencias	Tiempo	Costo
3D Max	1 mes	\$2,594.00 MX
Textures	1 mes	\$130.00 MX
Visual Studio	Precio Único	\$12,334.00 MX
Programador Junior	1 mes	\$15,000.00 MX
	TOTAL:	\$ 30,100.00 MX

Propuesta para el proyecto

Recorrido virtual que mostrará las expresiones culturales de tres de las civilizaciones prehispánicas más importantes de México: los mayas, olmecas y teotihuacanos.

El museo presentará una colección de artesanías, esculturas, cerámicas y otros objetos representativos de cada cultura. El cual incluirá elementos que simulen cada uno de los espacios propuestos y una ambientación tridimensional para complementar lo mejor posible, mostrará réplicas que explicarán aspectos como la arquitectura.

- Objetos:

Ojo de Ollin, cabeza olmeca, disco de mictlantecuhtli, Son partes importantes de las civilizaciones prehispánicas, que ha día de hoy siguen existiendo y están exhibidas en los museos

Estructura Principal (Contenedor de los Modelos Propuestos)



Descripción de las salas:

Primer sala: Cultura Maya

En la primera sala del museo, los visitantes serán transportados a la antigua civilización Maya, conocida por su impresionante desarrollo en arquitectura, arte y ciencia. En esta sala se encontrará la pirámide de chichen itza o de kukulcán, fue creada en la legendaria ciudad Maya, así como un pilar tallado por ellos y el disco de la muerte, además se podrá ver a un guerrero de la época.

Segunda sala: Cultura Olmeca

La segunda sala del museo transportará a los visitantes al corazón de la enigmática civilización Olmeca, conocida como la "cultura madre" de Mesoamérica. En ella podrás encontrar a la gran cabeza olmeca, la cual retrata a alguno de sus gobernadores ancestrales, alrededor de la cabeza se puede ver a dos guerreros haciéndole reverencia y a uno saludando a los que llegan

Tercer sala: Cultura Teotihuacana

En la tercera sala, los visitantes estarán inmersos en la grandiosa metrópolis de Teotihuacán, conocida por sus monumentales pirámides y complejos urbanos. En donde se exhibe el calendario azteca, un cráneo con plumas, cofres hechos en la ciudad, piedra tallada por ellos. Sobre la pirámide se observa a un guerrero dando saltos

Animaciones:

```
524 ModelAnim guerrero_a("resources/objects/Guerrero_A/guerrero_a.dae");|
525 guerrero_a.initShaders(animShader.ID);
526
527 ModelAnim guerrero_o("resources/objects/Guerrero_O/guerrero_o.dae");
528 guerrero_o.initShaders(animShader.ID);
529
530 ModelAnim guerrero_m("resources/objects/Guerrero_M/guerrero_m.dae");
531 guerrero_m.initShaders(animShader.ID);
532
533 ModelAnim guerrero_odos("resources/objects/Guerrero_O2/guerrero_o2.dae");
534 guerrero_odos.initShaders(animShader.ID);
535
```

```

679
680 //Guerrero_Azteca
681 model = glm::translate(glm::mat4(1.0f), glm::vec3(230.0f, 0.0f, -180.0f));
682 model = glm::scale(model, glm::vec3(45.0f));
683 model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
684 animShader.setMat4("model", model);
685 guerrero_a.Draw(animShader);
686
687 //Guerrero_Olmeca
688 model = glm::translate(glm::mat4(1.0f), glm::vec3(-290.0f, 0.0f, -100.0f));
689 model = glm::scale(model, glm::vec3(0.5f));
690 model = glm::rotate(model, glm::radians(135.0f), glm::vec3(0.0f, 1.0f, 0.0f));
691 animShader.setMat4("model", model);
692 guerrero_o.Draw(animShader);
693
694 //Guerrero_Maya
695 model = glm::translate(glm::mat4(1.0f), glm::vec3(15.0f, 46.0f, -350.0f));
696 model = glm::scale(model, glm::vec3(0.5f));
697 animShader.setMat4("model", model);
698 guerrero_m.Draw(animShader);
699
700 //Guerrero_Olmeca2
701 model = glm::translate(glm::mat4(1.0f), glm::vec3(-290.0f, 0.0f, -200.0f));
702 model = glm::scale(model, glm::vec3(0.5f));
703 model = glm::rotate(model, glm::radians(-45.0f), glm::vec3(0.0f, 1.0f, 0.0f));
704 animShader.setMat4("model", model);
705 guerrero_odos.Draw(animShader);
706
707 //Guerrero_Olmeca3
708 model = glm::translate(glm::mat4(1.0f), glm::vec3(-235.0f, 0.0f, -150.0f));
709 model = glm::scale(model, glm::vec3(0.5f));
710 model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
711 animShader.setMat4("model", model);
712 guerrero_otres.Draw(animShader);
713

```

```

524 ModelAnim guerrero_a("resources/objects/Guerrero_A/guerrero_a.dae");
525 guerrero_a.initShaders(animShader.ID);
526
527 ModelAnim guerrero_o("resources/objects/Guerrero_O/guerrero_o.dae");
528 guerrero_o.initShaders(animShader.ID);
529
530 ModelAnim guerrero_m("resources/objects/Guerrero_M/guerrero_m.dae");
531 guerrero_m.initShaders(animShader.ID);
532
533 ModelAnim guerrero_odos("resources/objects/Guerrero_O2/guerrero_o2.dae");
534 guerrero_odos.initShaders(animShader.ID);
535
536 ModelAnim guerrero_otres("resources/objects/Guerrero_O3/guerrero_o3.dae");
537 guerrero_otres.initShaders(animShader.ID);

```

1. Plataformas de trabajo

Para el desarrollo de un museo virtual se utilizó el entorno de desarrollo integrado Microsoft Visual Studio. Se siguió la interfaz OpenGL, que provee funciones para manipular gráficos e imágenes. OpenGL en realidad es una especificación y no una API, fue desarrollada por Khronos Group y define el resultado de cada función, dejando la implementación a cargo de los desarrolladores. Las bibliotecas de OpenGL usadas en este proyecto están

escritas en C, permitiendo su uso desde otros lenguajes pero manteniendo su naturaleza como librerías C.

Es importante recordar que OpenGL se desarrolló con abstracciones debido a que las construcciones del lenguaje C no se traducen bien a otros lenguajes superiores. Una de estas abstracciones son los objetos en OpenGL, los cuales representan un subconjunto del estado y permiten configurar opciones, como la ventana de dibujo. Definir objetos permite especificar varios modelos 3D desacoplando la configuración de la renderización, vinculando el objeto correspondiente antes de dibujar cada modelo sin configurar sus opciones. Esto facilita el manejo de múltiples modelos en una aplicación OpenGL.

1.2. Parámetros básicos del entorno

2.1 uso de ventana

Antes de crear el entorno gráfico del museo, fue necesario crear un contexto OpenGL y una ventana para dibujar. Sin embargo, estas operaciones son específicas del sistema operativo, por lo que OpenGL se abstrae de ellas. Esto implica que se debe crear la ventana, definir el contexto y manejar la entrada del usuario de forma independiente. Para ello se utilizó la biblioteca GLFW, escrita en C y dirigida a OpenGL, la cual provee las funcionalidades básicas requeridas como crear el contexto OpenGL, definir parámetros de ventana y manejar entrada de usuario, lo cual fue suficiente para los propósitos del proyecto.

1.2.2 Vinculación

Para usar la biblioteca GLFW en el proyecto, fue necesario vincularla especificando `glfw3.lib` en la configuración del enlazador. Sin embargo, el proyecto aún necesitaba saber dónde encontrar este archivo, ya que las bibliotecas de terceros se almacenaban en un directorio diferente. Por lo tanto, primero se debió agregar ese directorio al proyecto para que pueda encontrar la biblioteca. Adicionalmente, se le indicó al IDE que considere ese directorio al buscar bibliotecas e incluir archivos. Esto permitió completar la vinculación requerida de GLFW.

1.2.2.1 Bibliotecas Externas Incluidas.

Para que el proyecto pudiera saber en qué parte buscar las herramientas necesarias, se insertó 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

1.2.2.2 Parámetros de Entrada del vinculador.

Una vez que establecimos las cabeceras externas, nuestra IDE Visual Studio 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;

1.2.2.3 Biblioteca de audio

IrrKlang es una potente API de alto nivel para reproducir sonido en aplicaciones 3D y 2D como juegos, visualizaciones y aplicaciones multimedia. Es gratuito para uso no comercial. También existe una versión profesional llamada irrKlang Pro que se puede usar en productos comerciales pagando una licencia.


Usa sus propios decodificadores de audio por lo que es independiente del sistema operativo. Esto asegura que las aplicaciones que lo usen funcionen bien en diferentes sistemas operativos y configuraciones, sin depender de decodificadores integrados como el de MP3 de Windows.

irrKlang.lib;

1.3. Shaders

Se utilizaron shaders, que son pequeños programas que se ejecutan en la GPU, para construir el entorno del museo en este proyecto. Los shaders se encargan de secciones específicas de la canalización gráfica y transforman entradas en salidas de forma aislada, comunicándose sólo a través de estas.

Los shaders se escribieron en GLSL, un lenguaje similar a C diseñado para gráficos que incluye características para vectores y matrices. Cada shader comienza con una declaración de versión, variables de entrada/salida y uniformes, y procesa las variables de entrada en su función principal para mostrar los resultados.



Los shaders se usaron para la carga de modelos, iluminación, ambientación con animaciones del proyecto. Aparecen listados en una carpeta específica y su programación depende de su función.

1.4. Carga de modelos

1.4.1 ASSIMP

Para crear los modelos 3D del entorno no era viable definir manualmente todos los vértices, normales y coordenadas de textura debido a la complejidad. En su lugar, se importaron modelos creados en herramientas 3D como 3DS Max, las cuales permiten diseñar formas complejas y aplicar texturas a través de mapeo UV, generando automáticamente los datos del modelo.

Se implementó la biblioteca Assimp, que permite importar docenas de formatos de archivos de modelo cargando los datos en sus estructuras de datos genéricas, abstrayendo los diferentes formatos. Al importar un modelo, Assimp carga todos los datos en un objeto de escena, donde cada nodo contiene índices de datos almacenados y puede tener hijos, construyendo así la jerarquía del modelo. De esta forma, se obtuvieron los datos necesarios para renderizar los modelos complejos sin preocuparse por su generación manual.

1.4.2 Mesh

Con Assimp se pudieron cargar modelos en formatos diversos, aunque estos se almacenaban en las estructuras de datos de esa biblioteca. Por lo tanto, fue necesario transformar dichos datos a un formato entendible para OpenGL.

Las mallas necesitan como mínimo vértices con posición, normal y coordenadas de textura, así como índices para el dibujo y datos de materiales (maps difusos/especulares). El constructor de la clase Mesh devuelve grandes listas de datos de mallas, las cuales requieren configurar buffers y atributos de vértices en shaders.

La función Draw de la clase renderiza las mallas. Antes se enlazan las texturas correspondientes, aunque esto es difícil ya que no se conoce de antemano la cantidad y tipos de texturas por malla. Por lo tanto, fue necesario un proceso de

transformación de datos entre Assimp y OpenGL para la renderización de los modelos.

1.4.3 Model

Se creó una clase Modelo para representar un objeto que contenga múltiples mallas y posiblemente texturas, como la pirámide con diferentes partes.

Para cargar un modelo, se utilizó Assimp para traducirlo a objetos de malla creados anteriormente. Se asumió que las rutas de texturas en los archivos de modelo son locales al directorio del propio modelo.

La función GetTexture obtiene la ruta completa de textura concatenando la ruta local del archivo y el directorio donde está el modelo, necesitando ambos parámetros.

Algunos modelos de Internet usaban rutas absolutas de textura, lo que no funcionó. Entonces se editaron manualmente para utilizar rutas locales o se reemplazaron las texturas, iniciando un proceso de adaptación al modelo. El objetivo fue representar modelos complejos mediante varias mallas y texturas.

2. Manual técnico: Proceso de elaboración

Tomando como punto de partida el cronograma y la planeación, se llegó a la fase de creación del escenario en donde colaborativamente se hicieron cambios, eliminación de elementos para ajustarse a la entrega.

La mayoría de los modelos del proyecto fueron modelados en 3d Max, solo alguno de ellos se obtuvo 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, por lo que se le tuvieron que colocar.

2.1 Selección 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.

2.1.1 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.

2.1.2 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.

2.2 Manual técnico: Iluminación

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

```
//Setup Advanced Lights
staticShader.setVec3("viewPos", camera.Position);
staticShader.setVec3("dirLight.direction", lightDirection); //Direccion de los
staticShader.setVec3("dirLight.ambient", glm::vec3(0.5f, 0.5f, 0.5f));
staticShader.setVec3("dirLight.diffuse", glm::vec3(0.9f));
staticShader.setVec3("dirLight.specular", glm::vec3(0.0f, 0.0f, 0.0f));

staticShader.setVec3("pointLight[0].position", lightPosition);
staticShader.setVec3("pointLight[0].ambient", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setVec3("pointLight[0].diffuse", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setVec3("pointLight[0].specular", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setFloat("pointLight[0].constant", 0.08f);
staticShader.setFloat("pointLight[0].linear", 0.009f);
staticShader.setFloat("pointLight[0].quadratic", 0.000032f);

staticShader.setVec3("pointLight[1].position", glm::vec3(-80.0, 0.0f, 0.0f));
staticShader.setVec3("pointLight[1].ambient", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setVec3("pointLight[1].diffuse", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setVec3("pointLight[1].specular", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setFloat("pointLight[1].constant", 0.01f);
staticShader.setFloat("pointLight[1].linear", 0.0009f);
staticShader.setFloat("pointLight[1].quadratic", 0.0032f);
```

```
672      animShader.setVec3("material.specular", glm::vec3(0.5f));
673      animShader.setFloat("material.shininess", 32.0f);
674      animShader.setVec3("light.ambient", ambientColor);
675      animShader.setVec3("light.diffuse", diffuseColor);
676      animShader.setVec3("light.specular", 1.0f, 1.0f, 1.0f);
677      animShader.setVec3("light.direction", lightDirection);
678      animShader.setVec3("viewPos", camera.Position);
```

Iluminación en los modelos de los guerreros

AL MODIFICAR ESTOS PARÁMETROS PODEMOS CONSEGUIR DIFERENTES FENÓMENOS DE ENTORNO EN NUESTRO ESCENARIO FINAL, POR EJEMPLO; PODEMOS CREAR UN AMBIENTE CON LUZ CLARA, U OTRO CON MENOR CLARIDAD Y UN EFECTO AMBIENTAL DE MAYOR PROFUNDIDAD



2.3 Manual técnico: Animaciones

El formato del cual cada archivo se debe cargar, es complicado, pues podría no realizar los movimientos o acciones que deseamos, como se vio en el laboratorio, se estuvo utilizando la extensión .dae de la cual es más eficiente.

Para poder lograr lo que escribimos en la propuesta del proyecto, se utilizaron dos software de diseño gráfico, Blender y 3D Max

- Uno de los guerreros olmecas se arrodilla al costado derecho de la cabeza olmeca.
- Segundo guerrero olmeca, se arrodilla al costado izquierdo
- Tercer guerrero olmeca, hace una reverencia para recibir a los visitantes
- La animación del guerrero que se encuentra sobre la piramide realiza un salto. En el modelo, se puede notar la movilidad de brazos , piernas, cabeza y ligeramente de la cabeza al momento de caer, haciendo énfasis

en la pierna derecha que hace la simulación y efecto de que sobre esa pierna está concentrando la fuerza para poder dar el salto.

- El guerrero que se encuentra a un costado del cráneo, tiene la animación de caminata en línea recta y podemos notar claramente el movimiento que realiza en pies, piernas, brazos y muñecas y su desplazamiento que realiza.
- Los modelos como la cabeza olmeca, el calendario, el ojo de Ollin, la pirámide, tienen animaciones por keyframes

Technical Guide

English

1. Work platforms


For the development of a virtual museum, the Microsoft Visual Studio integrated development environment was used. The OpenGL interface was followed, which provides functions to manipulate graphics and images. OpenGL is actually a specification and not an API, it was developed by Khronos Group and defines the result of each function, leaving the implementation to developers. The OpenGL libraries used in this project are written in C, allowing their use from other languages but maintaining their nature as C libraries.

It is important to remember that OpenGL was developed with abstractions because C language constructs do not translate well to higher-level languages. One of these abstractions are objects in OpenGL, which represent a subset of the state and allow configuring options such as the drawing window. Defining objects allows specifying multiple 3D models by decoupling rendering configuration, linking the corresponding object before drawing each model without configuring its options. This facilitates the handling of multiple models in an OpenGL application.

1.2. Basic parameters of the environment

2.1 window use

Before creating the graphics environment of the museum, it was necessary to create an OpenGL context and a window to draw. However, these operations are



specific to the operating system, so OpenGL abstracts them. This implies that the window must be created, the context defined and user input handled independently. For this, the GLFW library was used, written in C and directed to OpenGL, which provides the basic functionalities required such as creating the OpenGL context, defining window parameters and handling user input, which was sufficient for the project purposes.

1.2.2 Linkage

To use the GLFW library in the project, it had to be linked by specifying `glfw3.lib` in the linker configuration. However, the project still needed to know where to find this file, since third-party libraries were stored in a different directory. Therefore, that directory had to be added to the project first so that it could find the library. Additionally, the IDE was instructed to consider that directory when looking for libraries and include files. This allowed completing the required GLFW linkage.

1.2.2.1 External Included Libraries.

So that the project could know where to look for the necessary tools, the appropriate location string was manually inserted, the IDE will also search in those directories when looking for libraries and header files. As soon as the folders like GLFW were included, all the GLFW header files could be found

1.2.2.2 Linker input parameters.

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

`soil2-debug.lib;assimp-vc140-mt.lib;opengl32.lib;glew32.lib;glfw3.lib;`

1.2.2.3 Audio library.

irrKlang is a powerful high-level API for sound playback in 3D and 2D applications such as games, visualizations and multimedia applications.

It is free for non-commercial use. There is also a professional version called irrKlang Pro which can be used in commercial products by paying a license.

It uses its own audio decoders so it is independent of the operating system. This ensures that applications using it work well on different operating systems and configurations, without depending on built-in decoders like Windows' MP3 one.

In summary, irrKlang is a free and cross-platform API for playing sound in applications, which guarantees portability and functioning independently of changes in underlying operating systems. It has a professional version for commercial use.

irrklang.lib;

1.3. Shaders

Shaders, which are small programs that run on the GPU, were used to build the museum environment in this project. Shaders handle specific sections of the graphics pipeline and transform inputs into outputs in an isolated manner, communicating only through them.

The shaders were written in GLSL, a C-like language designed for graphics that includes features for vectors and matrices. Each shader begins with a version declaration, input/output variables and uniforms, and processes the input variables in its main function to show the results.


Shaders were used for model loading, lighting, environment mapping and animations of the project. They appear listed in a specific folder and their programming depends on their function.

1.4. Model loading

1.4.1 ASSIMP

To create the 3D models of the environment, it was not feasible to manually define all the vertices, normals and texture coordinates due to complexity. Instead, models created in 3D tools like 3DS Max were imported, which allow designing complex shapes and applying textures through UV mapping, automatically generating the model data.

The Assimp library was implemented, which allows importing dozens of model file formats by loading data into its generic data structures, abstracting from



different formats. When importing a model, Assimp loads all the data into a scene object, where each node contains stored data indexes and can have children, thus building the model hierarchy. In this way, the data needed to render the complex models without worrying about manual generation was obtained.

1.4.2 Mesh

With Assimp, models in diverse formats could be loaded, although they were stored in Assimp's data structures. Therefore, it was necessary to transform said data to a format understandable to OpenGL.

Mesheres need at least vertices with position, normal and texture coordinates, as well as drawing indexes and material data (diffuse/specular maps). The Mesh class constructor returns large mesh data lists, which require configuring buffers and vertex attributes in shaders.

The Draw function of the class renders the meshes. Textures are linked before, although it is difficult since the amount and types of textures per mesh are unknown beforehand. Therefore, a data transformation process between Assimp and OpenGL was needed for model rendering.

1.4.3 Model


A Model class was created to represent an object containing multiple meshes and possibly textures, like the pyramid with different parts.

To load a model, Assimp was used to translate it to mesh objects previously created. It was assumed that texture paths in model files are local to the model's own directory.

The GetTexture function obtains the full texture path by concatenating the local file path and the directory where the model is located, needing both parameters.

Some online models used absolute texture paths, which did not work. Then they were manually edited to use local paths or textures were replaced, starting a model adaptation process. The objective was to represent complex models through multiple meshes and textures.

2. Technical manual: Elaboration process



Taking the schedule and planning as a starting point, the phase of creating the scenario was reached, where changes and element elimination were collaboratively made to adjust to the delivery.

Most of the project's models were modeled in 3d Max, only some were obtained from platforms that offer various graphic models with different license terms.

The obtained models being of free license did not contain textures or materials, so they had to be added.

2.1 Texture selection

The best quality materials were searched for and the images were worked on with GIMP software, in order to be used as texture and mold the UV map. In order to create a better setting for the environment.

2.1.1 Convergence of materials and completion of modeling

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

2.1.2 Integration of the model into the final scenario

Finally, the workspace is swapped to continue with the integration and coupling to the final deliverable. It is ensured 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.

2.2 Technical manual: Illumination

We place all the uniform variables for the types of lights we have; directional, ambient, diffuse and specular. We have to manually configure them and index them. We must adapt the PointLight structure itself in the matrix to establish each uniform variable. We can even define light types as classes and establishing their values there, or using a more efficient uniform approach

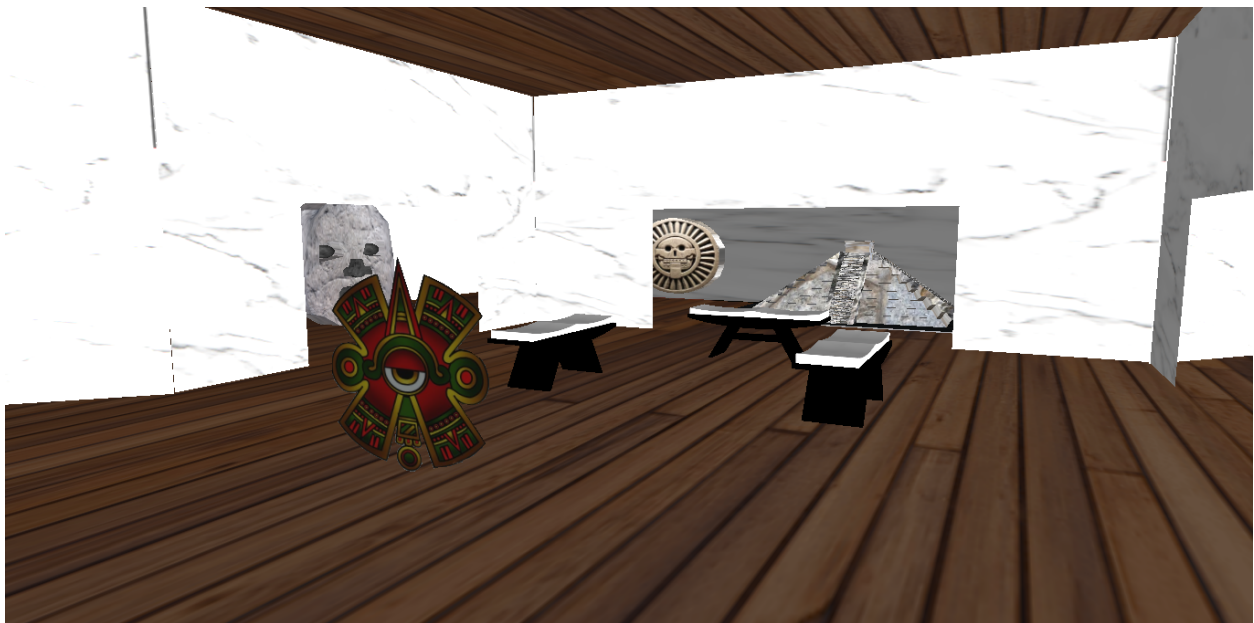
```
//Setup Advanced Lights
staticShader.setVec3("viewPos", camera.Position);
staticShader.setVec3("dirLight.direction", lightDirection); //Direccion de los
staticShader.setVec3("dirLight.ambient", glm::vec3(0.5f, 0.5f, 0.5f));
staticShader.setVec3("dirLight.diffuse", glm::vec3(0.9f));
staticShader.setVec3("dirLight.specular", glm::vec3(0.0f, 0.0f, 0.0f));

staticShader.setVec3("pointLight[0].position", lightPosition);
staticShader.setVec3("pointLight[0].ambient", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setVec3("pointLight[0].diffuse", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setVec3("pointLight[0].specular", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setFloat("pointLight[0].constant", 0.08f);
staticShader.setFloat("pointLight[0].linear", 0.009f);
staticShader.setFloat("pointLight[0].quadratic", 0.00032f);

staticShader.setVec3("pointLight[1].position", glm::vec3(-80.0, 0.0f, 0.0f));
staticShader.setVec3("pointLight[1].ambient", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setVec3("pointLight[1].diffuse", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setVec3("pointLight[1].specular", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setFloat("pointLight[1].constant", 0.01f);
staticShader.setFloat("pointLight[1].linear", 0.0009f);
staticShader.setFloat("pointLight[1].quadratic", 0.0032f);
```

BY CHANGING THESE PARAMETERS WE CAN ACHIEVE DIFFERENT ENVIRONMENTAL PHENOMENA IN OUR FINAL SCENARIO, FOR EXAMPLE;

WE CAN CREATE AN ENVIRONMENT WITH BRIGHT LIGHT, OR ANOTHER WITH LOWER CLARITY AND A GREATER AMBIENT EFFECT



2.3 Technical manual: Animations

The format that each file must be uploaded is complicated, because it could not perform the movements or actions we want, as seen in the laboratory, the .dae extension was being used which is more efficient.

To achieve what we wrote in the project proposal, two graphic design software, Blender and 3D Max, were used

One of the Olmec warriors kneels on the right side of the Olmec head.

Second Olmec warrior kneels on the left side

Third Olmec warrior bows to receive visitors

The animation of the warrior that is on top of the pyramid does a jump. In the model, you can notice the mobility of arms, legs, head and slightly of the head at the moment of falling, emphasizing the right leg that simulates and has the effect that on that leg it is concentrating the force to be able to give the jump.

The warrior that is on one side of the skull has the walking animation in a straight line and we can clearly see the movement it makes in feet, legs, arms and wrists and its displacement it makes.

Models like the Olmec head, the calendar, the Ollin eye, the pyramid, have animations by keyframes.

Conclusiones:

Barrionuevo Perez Daniel Alejandro:

- In this project, the main activities I carried out were the creation of the scenario, the construction of models using the 3D Max program and, although to a lesser extent, I also supported the development of the user and technical manuals respectively, since my main contribution was in the development of the models.
- Regarding model construction, I can mention that although they are quite simple designs, it took me a considerable amount of time to achieve them as I had to familiarize myself with the 3D Max software by watching various video tutorials in order to be able to understand the basic operation of it and be able to perform the models required with the knowledge acquired.

- The main problems I faced was that really at first I felt lost because I did not know the software at a level where I could model without problems, but all this was solved when I discovered the "box modeling" technique which was the one I mainly used to make my models.
- With respect to the manual, my contribution was not as notable in this part of the project, but I helped with some of the research necessary for it and to advance part of the manual.
- In conclusion, I can say that this project presented a true challenge personally since the amount of time it took me to carry it out was much more than I had thought but it is gratifying to see the results obtained.

Bustamante Colin Israel:

- In the task distribution, I was assigned the design and modeling of warriors belonging to the three main cultures of Mexico. To achieve this, a pre-defined human model was used, and various characters were created through textures. This process took a considerable amount of time since the textures were customized for each character, requiring the use of different photo editing programs. One of these programs allowed us to draw on the texture and subsequently modify the extensions, sizes, and color codes necessary to maintain the planned design (GIMP, PicsArt, and Sketchbook). Subsequently, the textures were applied to each model. However, complications arose when using the 3ds Max program, leading to a decision to switch to Blender, which facilitated a more effective texturing process.
- Once the static model was obtained, the Mixamo website was utilized to achieve the complex animations that were planned. This was accomplished through Blender, but exporting the model to an optimal extension resulted in errors. Consequently, the decision was made to use the .fbx extension, importing this file through 3ds Max, and then exporting it to .dae, which is the most suitable extension for working in OpenGL.

- The code implementation followed the example code presented in the laboratory classes. The complication arose in the .dae models, as they presented uninitialized or unclosed indices. Each .dae file had to be modified to avoid these errors and ensure that the models were loaded correctly.
- After addressing these issues, the five models were loaded correctly, allowing for translation, scaling, and rotation operations to customize the location, size, and orientation of each warrior displayed in the museum. Finally, the models were rendered with animShader, which initially encountered errors. However, with guidance, the error was identified and corrected, resulting in the expected outcomes.

Martínez Bautista Victor Eduardo:

- For this project I made models, as well as helped in the creation of the technical manual where I researched information about the different libraries available, on forums, official pages and videos to start understanding the reason for each of them.
- At various moments the project was difficult, when we removed parts of the code, when accidentally a parenthesis was missing or thinking about how to distribute everything in the virtual scenario. However, the idea of the museum appealed to the three team members, and even more by bringing together pre-Hispanic civilizations from Mexico.
- Where to faithfully portray each of the Mesoamerican cultures, we had to conduct research on the Olmec, Teotihuacan, Maya, Aztec and others. I spent some hours reading about their main architectural constructions and historical milestones to be able to recreate each of the rooms of the virtual museum. Another great challenge was to develop the sculptures. It required making multiple attempts and versions until achieving the required level of historical fidelity.

- We also tried to implement the sound library, however, it was unsuccessful and we believe that would allow visualizing at real scale being in those ancient places in the geographical area where they were found. In the end, despite the setbacks, we felt very proud of the result obtained, since we managed to convey in a didactic way the cultural richness of the ancient Mesoamerican civilizations so that more people can learn about our origins in an interactive way.

Repositorio:

- De Github: https://github.com/Israelbbc/ProyectoFinal_Grafica

El repositorio que está aquí es en el que estamos trabajando y también se va a utilizar para la teoría, por si llega a ver que hay cambios.

Documentación:

LearnOpenGL - OpenGL. (2017). "Learn OpenGL. Retrieved" Recuperado el 15 de noviembre de 2023, del sitio web: <https://learnopengl.com/Getting-started/OpenGL>

Microsoft. (2022). "Introducción a. Visual Studio". Recuperado el 15 de noviembre de 2023, del sitio web: <https://visualstudio.microsoft.com/es/vs/getting-started/>

LearnOpenGL - Shaders. (2017). "Learn OpenGL". Recuperado el 15 de noviembre de 2023, del sitio web: <https://learnopengl.com/Getting-started/Shaders>

LearnOpenGL - Model. (2017). "Learn OpenGL". Recuperado el 15 de noviembre de 2023, del sitio web: <https://learnopengl.com/Model-Loading/Model>

LearnOpenGL - Assimp. (2017). "Learn OpenGL". Recuperado el 15 de noviembre de 2023, del sitio web: <https://learnopengl.com/Model-Loading/Assimp>

Ambiera (s.f) "Características de irrKlang" Recuperado el 16 de noviembre de 2023, del sitio web: <https://www.ambiera.com/irrklang/features.html>

