

MANUAL DE USUARIO

Remodelación de
Laboratorio de
Computación Gráfica



Equipo 11

Introducción.....	3
Objetivo del manual.....	4
Instalación desde repositorio en GitHub.....	5
Prerrequisitos.....	5
Clonación del repositorio con líneas de comando.....	5
Clonación del repositorio con GitHub Desktop.....	6
Verificar la estructura de las carpetas.....	7
Contribuir y reportar issues.....	7
Requerimientos de hardware y software.....	9
Instalación y Configuración en Visual Studio 2022.....	11
Verificación de la estructura de carpetas.....	11
Primer arranque.....	12
Descripción de la Interfaz.....	14
Controles y Navegación.....	15
Navegación en el entorno 3D mediante la cámara.....	15
Control de Iluminación.....	16
Animaciones y Eventos.....	18
Funciones Principales.....	19
Skybox envolvente.....	20
Bucle principal.....	20
Animaciones.....	20
Animación por keyframes.....	20
Animación de monitores.....	21
Montaje de componentes de PC.....	21
Animación de logos.....	21
Animación del movimiento para profesor.....	22
Vista final del laboratorio.....	22
Consejos de uso.....	24
Resolución de Problemas Comunes.....	26
Fallos al enlazar bibliotecas (linker errors).....	26
Ventana negra o aplicación sin respuesta.....	26
Errores de compilación de shaders.....	27
Modelos o texturas faltantes.....	27
Animaciones inactivas o erráticas.....	28
Soporte y Contacto.....	28

Introducción.

Bienvenido al simulador interactivo del Laboratorio de Cómputo Gráfico de la Facultad de Ingeniería (ubicado en el edificio Q “Luis G. Valdés Vallejo”, conjunto sur). Este proyecto, desarrollado por el Equipo 1, aprovechando tecnologías de renderizado en tiempo real para ofrecerte una experiencia inmersiva y práctica al explorar una propuesta de mejoramiento del laboratorio.

En esta simulación, te sumergirás en un entorno 3D modelado íntegramente con OpenGL 3.3+, aprovechando bibliotecas como GLEW y GLFW para la creación y gestión de la ventana y el contexto gráfico, y GLM para las transformaciones matemáticas. Se han incorporado:

- Modelos 3D detallados: A partir de su archivo .obj de distintos componentes y material para el laboratorio, como computadoras, sillas, mesas, pizarrones y equipo diverso, cargados dinámicamente con las clases Model y Texture.
- Skybox envolvente, que sitúa al usuario en el espacio real del laboratorio, permitiendo una sensación de inmersión completa.
- Sistema de iluminación híbrido, con luz direccional, seis luces puntuales distribuidas estratégicamente en el techo y una linterna tipo spotlight sujeta a tu cámara y en el proyector del salón, para que puedas evaluar los efectos de distintos escenarios lumínicos.
- Animaciones configurables, desde el movimiento vertical de una pelota didáctica hasta la transición suave entre mobiliario antiguo y propuestas de equipamiento nuevo (explosión/implosión, rotaciones y keyframes), todas activables mediante teclas asignadas.
- Controles intuitivos, basados desde la entrada de teclado para una mejor respuesta y un sencillo manejo de desplazamiento en el entorno virtual, donde se pueden manipular varias cosas desde las animaciones, iluminación y cambio total del equipo.

El propósito de esta simulación es ofrecer a estudiantes, docentes y decisores un medio interactivo para visualizar y experimentar con distintas configuraciones del espacio de cómputo, facilitando la evaluación de propuestas de modernización y optimización de recursos. A lo largo de este manual encontrarás la guía paso a paso para instalar, configurar y aprovechar al máximo cada función, de manera que puedas centrarte en la observación crítica y la toma de decisiones fundamentadas.

Objetivo del manual.

El propósito de este Manual de Usuario es servir como una guía exhaustiva y accesible que acompañe al usuario desde la instalación inicial hasta el uso avanzado de la simulación del laboratorio virtual de computación gráfica. A través de instrucciones claras y detalladas, el manual permitirá a:

- **Estudiantes:** Comprender el flujo de trabajo completo, desde la configuración del entorno hasta la exploración de animaciones y efectos de iluminación, fomentando el aprendizaje práctico de conceptos de gráficos por computadora y entornos 3D interactivos.
- **Docentes:** Disponer de un recurso didáctico que facilite la demostración de técnicas de modelado, renderizado y animación en tiempo real, así como el análisis de propuestas de mejora en el equipamiento y el mobiliario del laboratorio.
- **Patrocinadores y autoridades académicas:** Evaluar de forma objetiva las ventajas y oportunidades de inversión en el laboratorio, observando de manera inmersiva el impacto que un entorno de trabajo modernizado puede tener en la calidad de la enseñanza y en la motivación de los usuarios.
- **Usuarios generales:** Explorar de manera intuitiva los distintos componentes del espacio de cómputo gráfico —modelos 3D, skybox, luces dinámicas y animaciones configurables— para proporcionar retroalimentación sobre la ergonomía, estética y funcionalidad del mobiliario y equipos.

Gracias a este manual, cada usuario podrá:

1. Instalar fácilmente el software desde el repositorio oficial, cumpliendo los requisitos de hardware y software necesarios.
2. Configurar y personalizar parámetros clave (resolución, calidad de texturas, niveles de iluminación) para adaptar la experiencia según las capacidades de su equipo.
3. Navegar y controlar el escenario 3D con atajos y comandos intuitivos que permitan centrar la atención en los elementos que requieren evaluación.
4. Activar y combinar animaciones y eventos interactivos, comparando en tiempo real las condiciones actuales del laboratorio con las propuestas de modernización.

5. Analizar y documentar observaciones, identificando áreas de mejora en mobiliario o equipamiento, y respaldar las decisiones con una visión inmersiva y práctica.

En conjunto, este manual busca potenciar el uso de herramientas digitales interactivas para facilitar la toma de decisiones fundamentadas en el contexto educativo, promoviendo un entorno de aprendizaje más dinámico, eficiente y atractivo.

Instalación desde repositorio en GitHub.

A continuación encontrarás una guía para obtener el código fuente del proyecto, tanto desde línea de comandos como usando GitHub Desktop, así como para contribuir con reportes de errores o mejoras en GitHub.

Prerrequisitos.

Antes de clonar el repositorio, asegúrate de contar con lo siguiente instalado en tu máquina:

- Contar con Git (versión 2.20 o superior) y C++ (compilador compatible con C++11, por ejemplo g++ en Linux/macOS o Visual Studio Community/Build Tools en Windows)
- Conexión a internet para descargar dependencias y sincronizar con GitHub.

Tip: En Windows puedes instalar Git y mediante el instalador de [Chocolatey](#).

Clonación del repositorio con líneas de comando.

1. Abrir una terminal, ya sea en Windows el PowerShell o Git Bash, si utiliza mac OS/Linux abrir la terminal nativa.
2. Navegar a la carpeta donde deseas ubicar el proyecto, **cd /ruta/carpeta/local**
3. Para clonar el repositorio oficial ejecuta el siguiente comando para descargar todos los archivos y el historial de versiones.

URL: <https://github.com/Miguel07FI/Computacion-Grafica-Equipo-11>

Clonación:

```
git clone
```

```
https://github.com/Miguel07FI/Computacion-Grafica-Equipo-11.git  
cd "ubicación local"/Computacion-Grafica-Equipo-11
```

4. Esto creará una carpeta llamada **Computacion-Grafica-Equipo-11** con todo el contenido del proyecto.

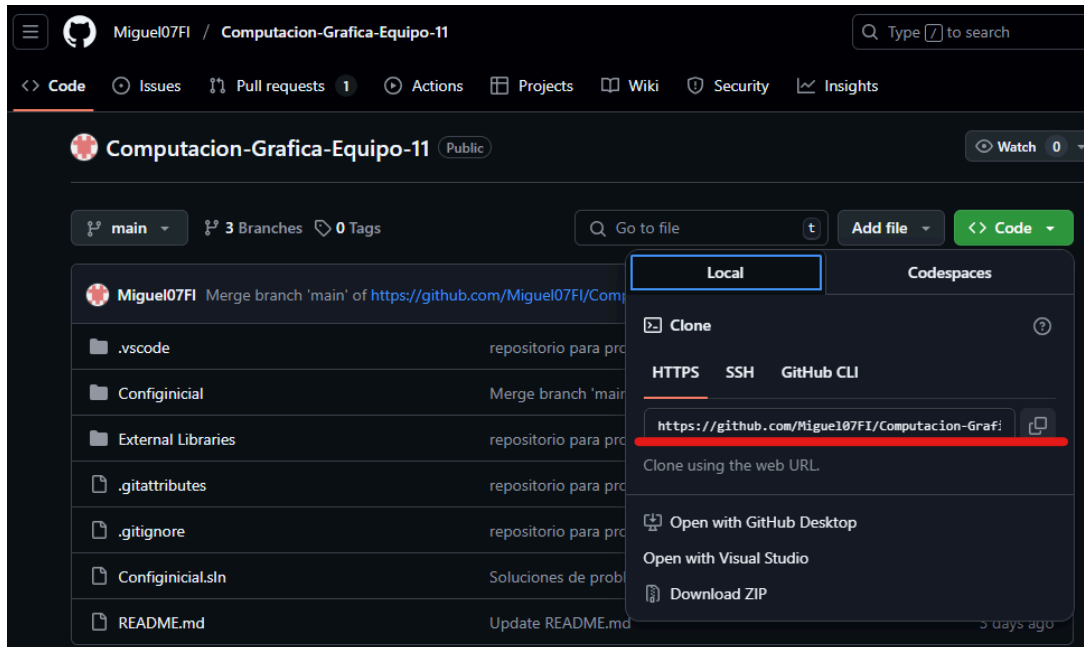


Figura 1. Enlace HTTPS para poder realizar la clonación del repositorio

Clonación del repositorio con GitHub Desktop.

Si se desea utilizar una aplicación de escritorio con interfaz para poder crear y bajar repositorios, se recomienda lo siguiente:

1. Descarga e instala GitHub Desktop desde <https://desktop.github.com/>
2. Inicia GitHub Desktop y haz login con tu cuenta de GitHub.
3. En la esquina superior izquierda, haz clic en File → Clone repository...
4. En la pestaña URL, pega la dirección del repositorio: <https://github.com/Miguel07FI/Computacion-Grafica-Equipo-11>
5. Elige la **ubicación local** donde desees clonar el proyecto (por ejemplo, tu carpeta de "Proyectos").
6. Haz clic en **Clone**. GitHub Desktop descargará el repositorio y lo mostrará en la lista de tus proyectos.

7. Para acceder a los archivos, abre la carpeta resultante (Computacion-Grafica-Equipo-11) con tu explorador de archivos o directamente con Visual Studio 2022.

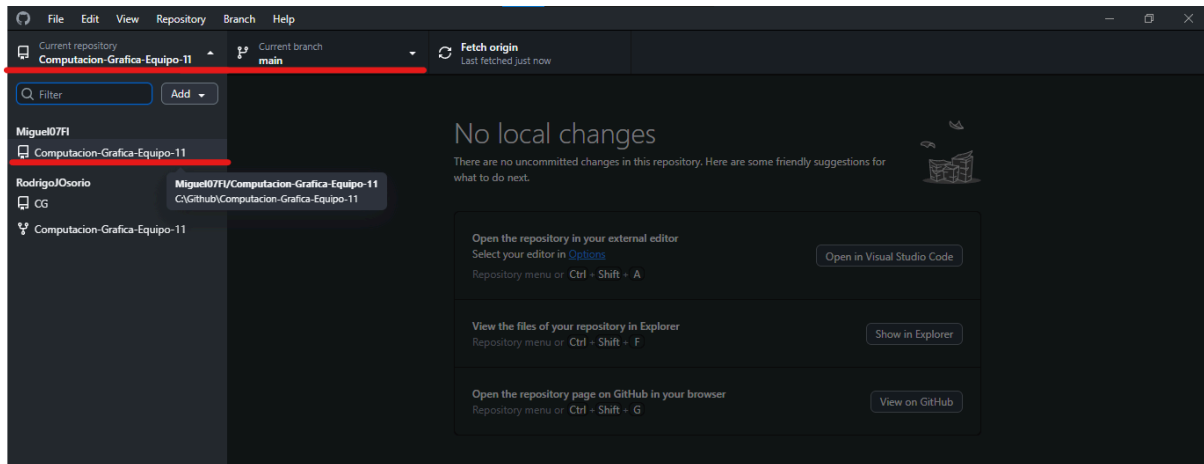


Figura 2. Detalles a verificar para poder trabajar de forma adecuada

Verificar la estructura de las carpetas.

Dentro de Computacion-Grafica-Equipo-11 deberías ver al menos:

- **.git/**
Carpeta oculta con el historial de versiones y la configuración de Git.
- **Configinicial/, External Libraries/**
Contiene la configuración base, librerías DLL (Windows), modelos y texturas.
- **Models/, Shader/, Debug/, etc.**
Subdirectorios que organizan tus recursos 3D, archivos de sombreado y compilaciones de prueba.
- **Archivos fuente (.cpp, .h)**
El núcleo de la simulación, incluyendo PROYECTO EQUIPO 11.cpp y clases como Model.h, Shader.h, Camera.h.

Si alguno de estos elementos no aparece, comprueba que el comando git clone se haya completado sin errores.

Contribuir y reportar issues.

Para abrir un issues, sigue las indicaciones:

1. Desde tu navegador, ve a la sección **Issues** del repositorio.

2. Haz clic en New issue, describe el problema, indicando sistema operativo, versión de dependencias y pasos para reproducirlo.

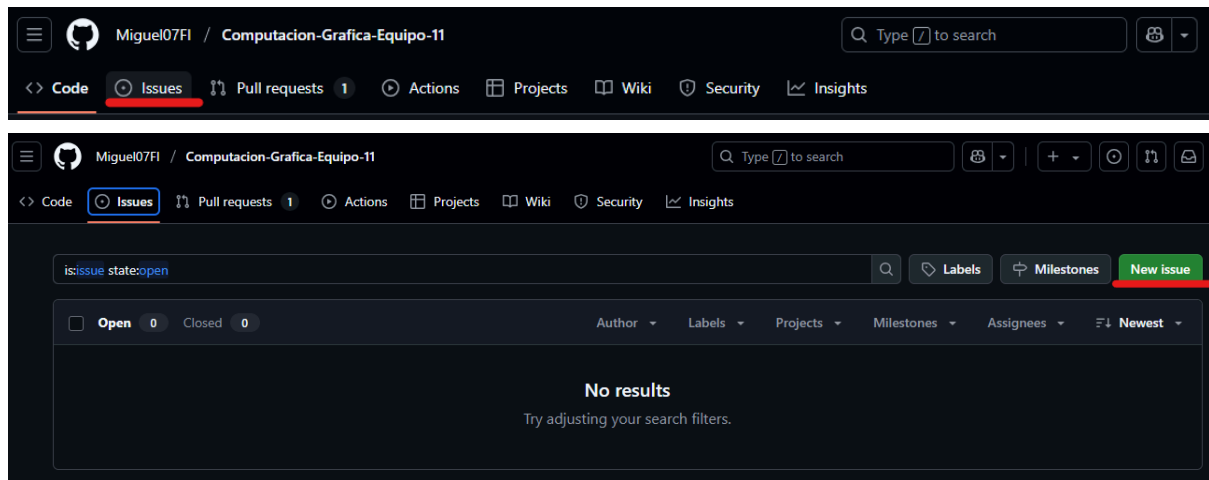


Figura 3. Pestaña de issues para contribuir a la mejora del desarrollo

Para enviar un **Pull request**:

1. En GitHub Desktop, selecciona Branch → New Branch, nómbrala descriptivamente (por ejemplo, mejora-modelos).
2. Realiza tus cambios y confirma con Commit to <branch>.
3. Haz clic en Push origin para subir la rama a GitHub.
4. En la interfaz web de GitHub, abre un Pull request desde tu rama hacia main, describe tus modificaciones y enviarlo para revisión.

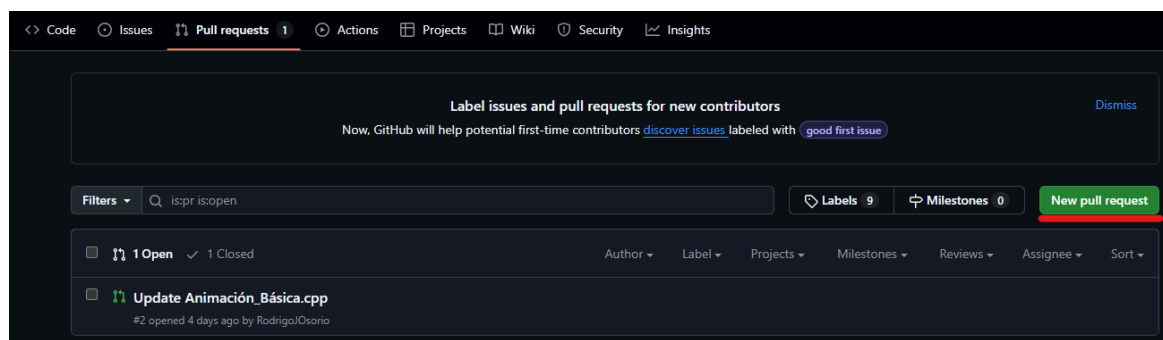
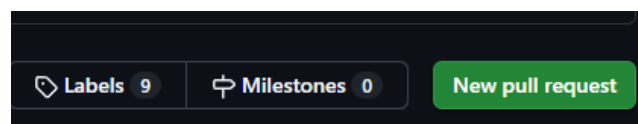


Figura 4. Pestaña Pull Request

Con estos métodos podrás clonar, explorar y colaborar fácilmente en el proyecto, ya sea usando comandos de Git o una interfaz gráfica amigable como GitHub Desktop.

Requerimientos de hardware y software.

Para asegurar un funcionamiento óptimo de la simulación del laboratorio virtual de computación gráfica, a continuación se detallan y explican en profundidad los recursos mínimos y recomendados tanto de hardware como de software.

Sistema operativo: La aplicación es multiplataforma, por lo que puedes ejecutarla en cualquiera de los siguientes sistemas:

- Windows 10/11, en versiones de 64 bits. Se recomienda mantener el sistema actualizado para contar con los últimos controladores de GPU.
- Linux (Ubuntu 20.04+), en distribuciones basadas en Debian/Ubuntu, puedes instalar las dependencias mediante el gestor apt.
- MacOS 12+. Asegúrate de habilitar el soporte de desarrollo de línea de comandos (Xcode Command Line Tools) para compilar desde terminal.

Componente.	Especificación mínima.	Por qué es importante.
GPU	Soporte para OpenGL 3.3 o superior.	OpenGL 3.3 introduce shaders, buffers y transformaciones necesarias para el renderizado.
Memoria RAM	8 GB	Permite cargar modelos 3D, texturas y realizar animaciones sin ralentizaciones.
CPU	Dual-core a 2.5 GHz (o equivalente moderno).	Responsable de la lógica del programa, gestión de animaciones y llamadas a OpenGL.
Almacenamiento	10 GB de espacio libre en disco (SSD preferido)	Para compilar, guardar recursos (modelos, texturas) y mantener un rendimiento ágil.

Tabla 1. Requerimientos mínimos de Hardware y Software

Hardware mínimo: *Nota:* Para escenas muy complejas, con múltiples luces y modelos de alta resolución, se recomienda una GPU con al menos 4 GB de VRAM y 16 GB de RAM para mantener 60FPS estables.

Software:

- Compilador C + + , con soporte de C++11 o superior.
 1. g++ (GNU Compiler Collection) en Linux/macOS.
 2. Clang en macOS o Linux modernos.
 3. Visual Studio Community / Build Tools en Windows.

Por qué: Las características de C++11 (como auto, Lambda functions y bibliotecas estándar mejoradas) se utilizan ampliamente en las clases de la aplicación.

Se debe considerar las siguientes bibliotecas gráficas y de utilidades:

Biblioteca.	Funcionalidad principal.
GLEW	Gestión de extensiones de OpenGL, necesario para acceder a funciones modernas de la API.
GLFW	Creación de ventanas, gestión de contexto OpenGL y eventos de entrada (teclado, ratón).
GLM	Colección de funciones y tipos matemáticos (vectores, matrices) para transformaciones 3D.
Assimp	Carga de formatos de modelos 3D (.obj, .fbx, etc.) de manera unificada.
SOIL2 o stb_image	Carga de imágenes y texturas en formatos comunes (.png, .jpg, etc.).

Tabla 2. Bibliotecas utilizadas dentro del código.

Con estos elementos correctamente instalados y configurados, tu entorno estará listo para compilar y ejecutar la simulación, garantizando una experiencia fluida y sin sorpresas durante la ejecución.

Instalación y Configuración en Visual Studio 2022.

A continuación se describe paso a paso cómo preparar y ejecutar el proyecto en un entorno Visual Studio 2022, partiendo de la estructura básica del repositorio.

Verificación de la estructura de carpetas.

Tras clonar o descargar el repositorio, comprueba que tu carpeta **ProyectoGrafica/** contenga al menos lo siguiente:

```
ProyectoGrafica/
├── .git
├── .vs
├── .vscode    ← Configuración local de Visual Studio
├── Configinicial
│   ├── Debug    ← DLLs y librerías compiladas
│   ├── Models
│   │   ├── Modelos necesarios para laboratorio actual y nuevo.
│   │   └── cada carpeta contiene `.obj`, `.mtl` y texturas.
│   ├── Shader
│   ├── Shader_
│   ├── SOIL2
│   ├── assimp-vc140-mt.dll
│   ├── Camera.h
│   ├── Mesh.h
│   ├── Model.h
│   ├── Shader.h
│   ├── stb_image.h
│   ├── Configinicial.vcxproj
│   ├── Configinicial.vcxproj.filters
│   ├── Configinicial.vcxproj.user
│   ├── glew32.dll
│   └── PROYECTO EQUIPO 11.cpp
├── Debug      ← Carpeta de salida tras compilar
├── External Libraries
├── .gitattributes
├── .gitignore
└── Configinicial.sln ← Solución de Visual Studio 2022
```

Si faltara alguno de estos elementos, revisa que la clonación (*git clone ...*) se completará sin errores.

Primer arranque.

Para poner en marcha la simulación del laboratorio virtual por primera vez, sigue estos pasos centrados en el uso de Visual Studio 2022, sin repetir las configuraciones previas:

1. Descarga e instalación de Visual Studio 2022.
 - Asegúrate de instalar el Workload “Desarrollo de escritorio con C++”, que incluye el compilador y las herramientas necesarias para proyectos OpenGL.
2. Abrir la solución del proyecto.
 - Inicia Visual Studio 2022 y selecciona Archivo → Abrir → Proyecto/Solución.
 - Navega hasta la carpeta raíz del repositorio y abre Configinicial.sln.
 - Verifica en el Explorador de soluciones (panel derecho) que aparezcan los proyectos “Configinicial” y “External Libraries”.

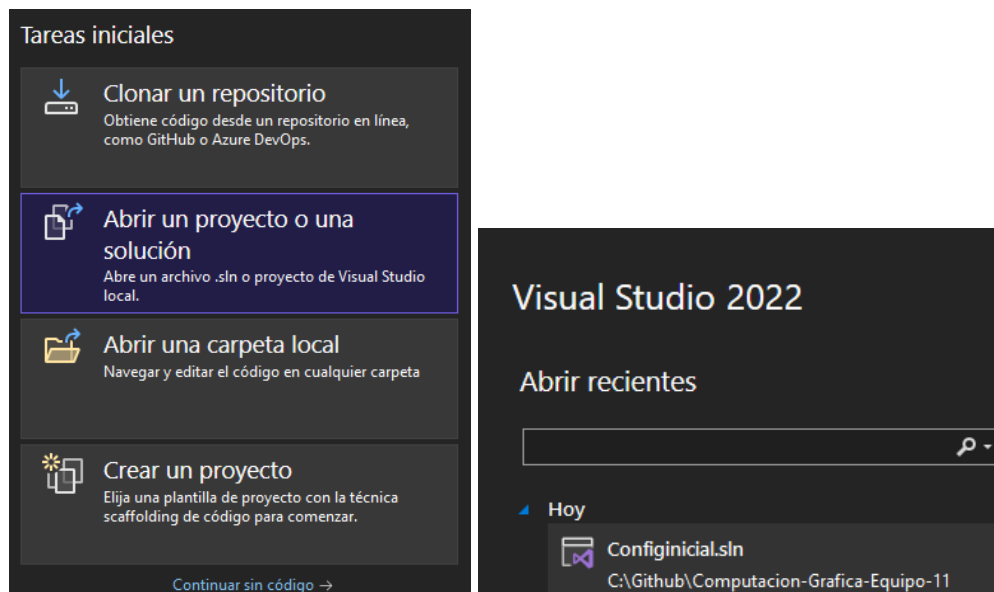


Figura 5. Panel inicial de Visual Studio, en donde se seleccionará el explorador de soluciones.

Figura 6. Ventana “Abrir un proyecto...” en donde seleccionaremos la solución requerida.

3. Seleccionar proyecto de inicio.

En el explorador de soluciones, del lado derecho de la ventana, debemos observar los documentos, carpetas y códigos del proyecto. Se debe dar click sobre “Archivos de origen” y después el código fuente del proyecto “PROYECTO EQUIPO 11.cpp”.

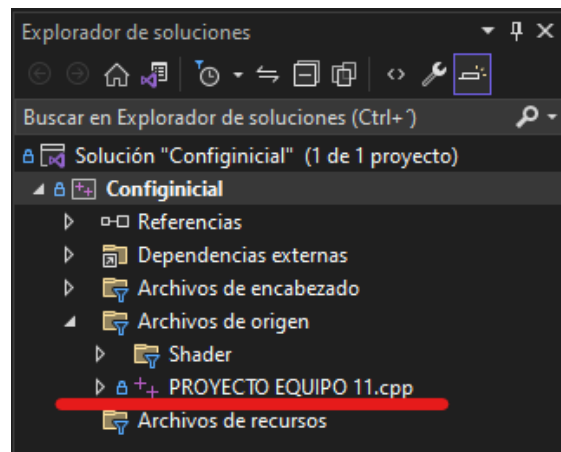


Figura 7. Código del proyecto

4. Elegir configuración y plataforma.

En la barra superior se debe colocar en la configuración para compilar y ejecutar el “Debug” y con “x86” (coincide con las DLLs provistas), para posteriormente depurar el proyecto con la flecha verde o con la tecla F5.

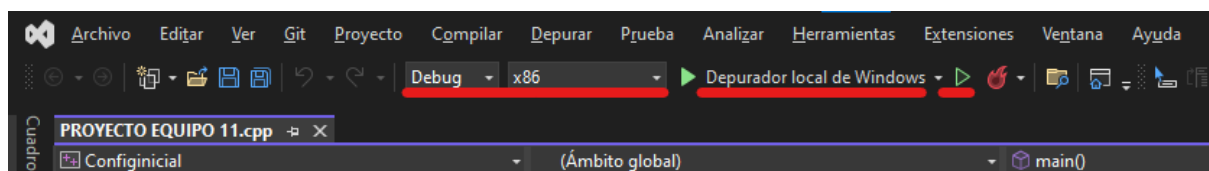


Figura 8. Detalles importantes a tomar en cuenta para el correcto funcionamiento del código.

5. Verificar directorio de trabajo.

Tienes que dirigirte a **Proyecto** → **Propiedades** → **Depuración**. Para asegurarte de que **Directorio de trabajo** apunte a la carpeta donde están las DLLs y recursos. De esta forma el programa encuentra modelos, texturas y bibliotecas en tiempo de ejecución.

6. Navegar al archivo fuente principal.

Desde el Explorador de soluciones, expanda **Archivos de origen**, para después hacer doble clic en **PROYECTO EQUIPO 11.cpp** para abrir el editor.

Aquí puedes colocar breakpoints en `main()` o en funciones de inicialización y bucle principal para depurar comportamientos específicos.

7. Compilar y ejecutar.

- Pulsa F5 o haz clic en el botón verde Iniciar depuración.
- Visual Studio compila el proyecto (mostrando errores o advertencias en la ventana Salida) y luego lanzará la ventana de la simulación con el skybox y los controles activos.

8. Uso de las herramientas de depuración.

Durante la ejecución, la ventana de **Locals** y la ventana de **Watch** te permitirán inspeccionar variables como `explosionFactor` o matrices de transformación.

También se utiliza **Step Over** (F10) y **Step Into** (F11) para recorrer el código línea a línea y entender el flujo de inicialización de OpenGL, la carga de modelos y la gestión de animaciones.

Con estos pasos tendrás el proyecto compilado y en ejecución, listo para probar las teclas de control, experimentar con iluminación y evaluar cada animación interactiva desde el entorno de Visual Studio 2022.

Descripción de la Interfaz.

Al iniciar la simulación del laboratorio virtual, se abre automáticamente una ventana de aplicación en modo de pantalla ajustable, cuyo título refleja el nombre del proyecto y la modalidad de ejecución (por ejemplo, "PROYECTO EQUIPO 11 – Debug"). En la primer vista del programa se encontrará lo siguiente:

1. Skybox envolvente: Se trata de un cubo o esfera invertida mapeada con seis texturas que representan el entorno exterior del laboratorio (fachada, cielos y edificaciones adyacentes). Este se renderiza primero en cada cuadro, garantizando que cualquier objeto 3D posterior se muestre dentro de ese "fondo infinito" sin interrupciones ni bordes visibles.
2. Viewport y cámara: La ventana de render se divide conceptualmente en un viewport único que ocupa todo el espacio cliente. Por otro lado, la cámara adopta una perspectiva de 60° por defecto, con controles de proyección perspectiva (no ortográfica), calculados vía la librería GLM.

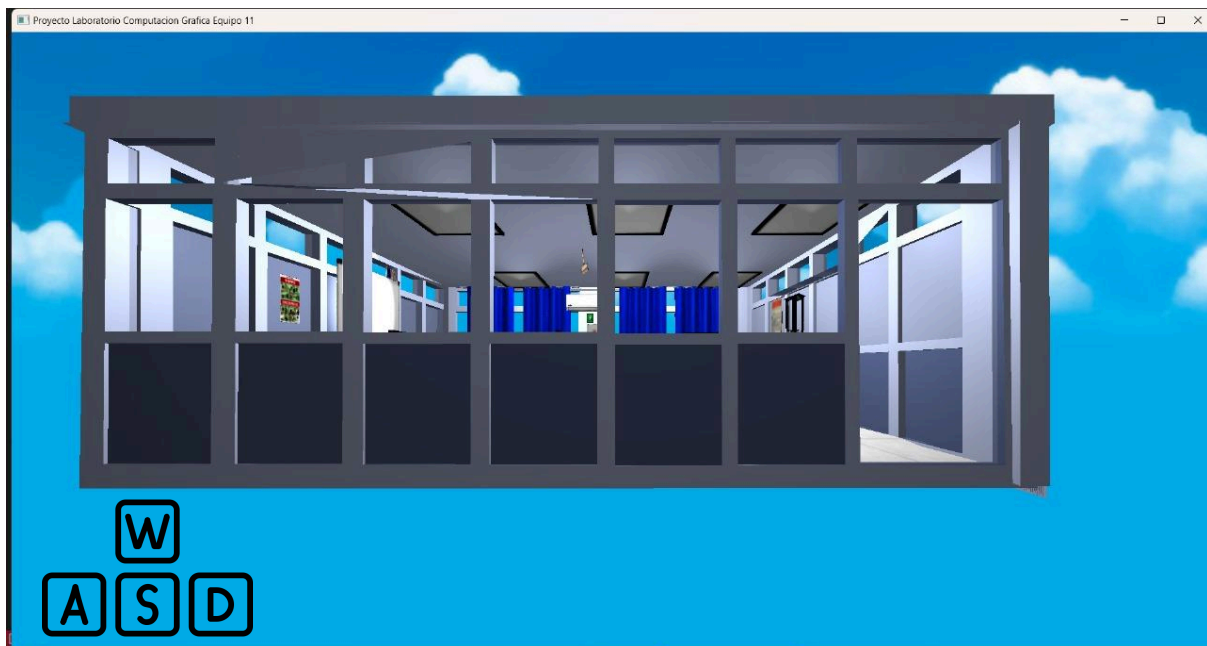


Figura 9. Primera interacción al correr el código del proyecto.

En conjunto, esta interfaz está diseñada para ser lo más limpia y minimalista posible, dejando al usuario explorar libremente el espacio 3D sin distracciones, pero ofreciendo en todo momento la información esencial para controlar y depurar la simulación.

Controles y Navegación.

Para sacarle el máximo provecho al laboratorio virtual, es fundamental dominar tanto el desplazamiento en el espacio 3D como la activación de luces y animaciones. A continuación se describen en detalle cada uno de estos controles.

Navegación en el entorno 3D mediante la cámara.

La simulación utiliza una cámara “tipo FPS” que responde a teclas y al ratón para desplazarte y mirar alrededor, dándonos una sensación de vista en primera persona:

Tecla	Descripción
W / ↑	Avanzar
S / ↓	Retroceder
A / ←	Mover izquierda

D / →	Mover derecha
Mouse	Girar cámara (click y mover)
Esc	Cierra la ventana y sale del programa.

Tabla 3. Teclas de navegación del entorno 3D

Control de Iluminación.

Para la iluminación se crea por hileras dónde se encuentran las mesas del laboratorio, teniendo 3 filas de luces que se manejan independientemente, además de contar con la luz reflectora que sale desde el proyector.

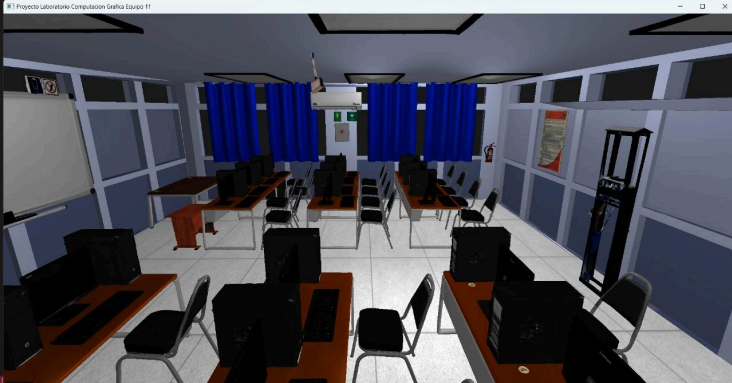

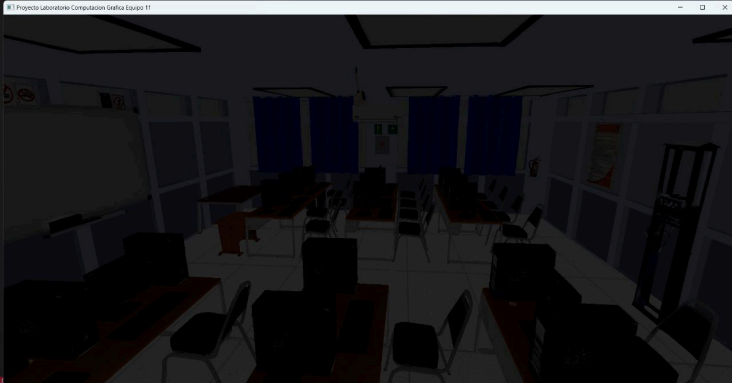
Se pueden manipular independientemente para apagar y prender mediante los siguientes controles de teclado:

Tecla	Descripción
U	Se apaga o prende la última fila.
I	Se apaga o prende la fila de enmedio.
O	Se apaga o prende la primera fila.
L	Luz reflector del proyector.

Tabla 4. Teclas para control de iluminación

Internamente, cada hilera está representada por un arreglo de luces puntuales (**PointLights**), cuya propiedad `enabled` se invierte al pulsar la tecla.

Observamos primero la iluminación que se tiene en el salón actual, contemplando las luces puntuales y la luz reflector que sale del proyector.

Tipo de luz	Imagen
<p>Todo encendido, con luces puntuales encendidas.</p>	 <p>Figura 10. Luces ambiental.</p>
<p>Solo una luz puntual.</p>	 <p>Figura 11. Luz puntual.</p>
<p>Salón apagado.</p>	 <p>Figura 12. Luces desactivadas.</p>

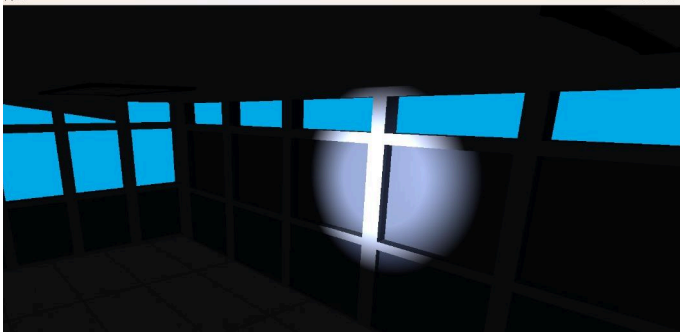
Luz reflectora del proyector.	 <p>Figura 13. Luz reflectora.</p>
-------------------------------	--

Tabla 5. Muestra de comportamiento de fuentes de luz.

Animaciones y Eventos.

Para comparar el laboratorio actual con propuestas de modernización, se han implementado varias animaciones controladas por tecla:

Tecla	Descripción
N	Explosión de monitores para cambiar a nuevos (inicia explosionFactor).
X	Secuencia de submodelos para armar el nuevo gabinete de la PC.
P	Animación de sillas (salida si y entrada sn)
Y	Se realiza el cambio de salón de viejo a nuevo.
T	Se hacen cambios de mobiliario secundario.
F	Realiza animación de logos en pantalla
H	Comienza animación para profesor.
Space	Hace un reinicio del salón para tenerlo en el modelo actual.

Tabla 6. Teclas de control de animaciones.


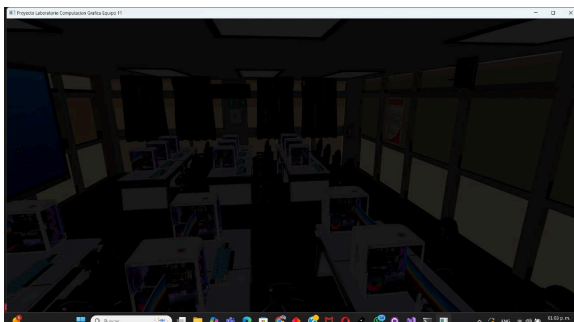
Descripción	Imagen
Salón nuevo con animaciones implementadas para la innovación del mobiliario y equipos de cómputo, además de tener solo una luz puntual activa.	 <p>Figura 14. Salón con remodelación implementada.</p>
Salón nuevo con iluminación apagada, para mostrar el cambio de luces, ya cuando se hayan realizado las animaciones correspondientes.	 <p>Figura 15. Salón previo al cambio de luces.</p>

Tabla 7. Muestra de eventos implementados.

Con estos controles tendrás total libertad para explorar el espacio, manipular la iluminación y activar las animaciones que comparan el estado actual del laboratorio con las mejoras propuestas, todo en tiempo real y de forma interactiva.

Funciones Principales.

El núcleo de la simulación reside en la función **Animation()** y en el bucle principal, que juntos coordinan la actualización de estados, la interpolación de animaciones y el renderizado de la escena con iluminación dinámica. A continuación se describen sus responsabilidades y fases principales:

Skybox envolvente.

Al iniciar el programa, antes de dibujar cualquier objeto 3D, se configura y renderiza un skybox mediante un cubo invertido con texturas aplicadas en sus seis caras. Este entorno envolvente evita "bordes" visibles y sitúa al usuario en un espacio realista alrededor del laboratorio.

Bucle principal.

Para cada frame, el programa realiza los siguientes pasos en orden:

1. Cálculo de `deltaTime`: diferencia entre el tiempo actual y el del frame anterior, para normalizar velocidades.
2. Procesado de eventos (`glfwPollEvents`) y movimiento de cámara (`DoMovement()`).
3. Actualización de animaciones con llamada a `Animation()`.
4. Limpieza de buffers (`glClear`) y activación de prueba de profundidad.
5. Configuración de shaders y envío de uniformes de iluminación direccional y de los seis point lights, así como del spotlight del proyector, cuyo estado (encendido/apagado) depende de flags que cambian con las teclas **U/I/O/L**.
6. Dibujo de cada modelo (`Model.Draw()`), aplicando traslaciones, rotaciones, escalados y, en el caso de monitores y sillas, factores de explosión o interpolación calculados.

Animaciones.

Animación por keyframes.

Cada silla está representada por una instancia de **SillaAnimada**, con un estado interno (fase) que avanza secuencialmente:

1. Escalado inicial (**EscalandoAntesDeMover**): la silla reduce su escala del valor original hasta 1.0 mediante `glm::mix`.
2. Elevación (**SubirAntesDeMover**): una vez escalada, la silla sube 4 unidades en Y para destacarse visualmente.
3. Desplazamiento hacia la "salida" (**MoverASalida**): usa **SillaKeyframeAnimation**, que interpola posición y escala entre keyframes definidos, invocando `animacion.update(deltaTime)`.

4. Explosión y desaparición (**DesaparecerSi**): incrementa **siExplosionFactor** y mezcla la escala a 0, ocultando el modelo original al completar la fase.
5. Aparición de la silla nueva (**AparecerSn**): la versión renovada crece de escala 0 a 1.0, primero de forma suave con *glm::mix*, luego regresando a su posición original mediante un segundo *keyframe*.
6. Finalización: al llegar a la fase **Completa**, la silla vuelve a su estado inicial hasta la siguiente activación.

Este enfoque basado en máquinas de estado garantiza transiciones limpias y permite interrumpir o revertir la animación si el usuario lo desea.

Animación de monitores.

Al pulsar **N**, se activa **explosionActive** y **explosionFactor** crece gradualmente hasta 1.7. Durante este proceso, cada monitor aplicará una transformación que reduce su escala en proporción inversa al factor de explosión (**1.0 – explosionFactor/1.7**) y lo elevará ligeramente para simular partículas que flotan.

Una vez alcanzado el límite, se oculta el modelo antiguo (**model1Visible = false**) y se muestra el nuevo (**model2Visible = true**), que gira sobre sí mismo y aumenta su escala hasta 1.0 antes de detenerse. La tecla M invierte la secuencia, reiniciando todos los flags y variables (**explosionFactor**, **implosionFactor**, visibilidades y rotaciones) a su estado inicial.

Montaje de componentes de PC.

Los gabinetes, tarjetas y periféricos se animan mediante el struct **InstanciaAnimacion**, que coordina fases de:

1. Explosión inicial (inflar y contraer).
2. Aparición de submodelos (*ganAppearing*).
3. Rotaciones y traslaciones sucesivas (*glnPhase1*, *glnPhase2*).
4. Escalados dinámicos según el tiempo transcurrido.

Cada submodelo (CPU, RAM, disipadores) implementa su propia mini-máquina de estado, asegurando que el montaje se vea progresivo y fluido.

Animación de logos.

Esta animación se realiza dentro del salón actualizado, ya que se trabaja sobre la pantalla que reemplaza el pizarrón, de tal modo que, se comienza la animación con un logo de la Facultad de Ingeniería rebotando en las esquinas y los límites de la

pantalla, para terminar un ciclo de rebote dando paso a la aparición de los 3 logos que se tienen en la pantalla principal del laboratorio de computación gráfica.

Para poder implementar esta animación en el recorrido del laboratorio virtual es necesario activarla con la tecla F.

Animación del movimiento para profesor.

Se vuelve a utilizar keyframes para definir los movimientos para el modelo del profesor, ya que tiene mediante variables se define una trayectoria que pasa por los pasillos del laboratorio sin tener el choque con algún objeto o mobiliario del salón.

Este entra por la puerta principal, para seguir caminando de manera sincronizada con sus piernas y brazos, por los pasillos del laboratorio, para que finalmente pueda llegar a un punto, y haga como si “vigilara” al mover la cabeza.

Vista final del laboratorio.

Al completar todas las animaciones y transiciones, el usuario ve el laboratorio renovado en todo su esplendor, con cada elemento actualizado y dispuesto para maximizar la funcionalidad y la experiencia de usuario. La vista final del laboratorio no es simplemente un conjunto de objetos animados, sino un espacio coherente y funcional que combina diseño, tecnología e interactividad para ofrecer una experiencia de usuario envolvente y práctica, alineada con las necesidades de enseñanza.



Figura 16. Modelado de salón actual.



Figura 17. Modelado de salón remodelado.



Figura 18. Vista final del laboratorio remodelado.

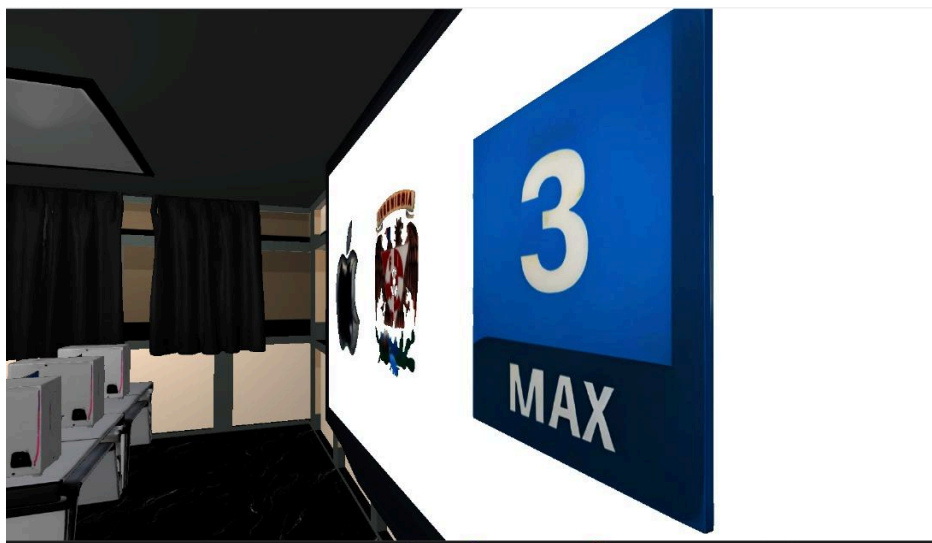


Figura 19. Vista de logos en pantalla.



Figura 20. Profesor en laboratorio de gráfica.

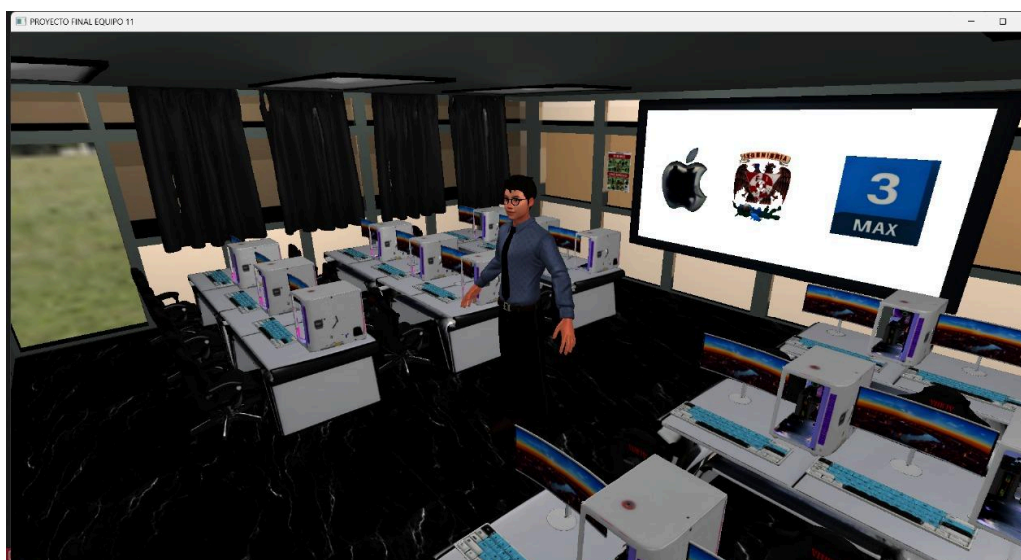


Figura 21. Innovación del laboratorio de gráfica.

Consejos de uso.

Para aprovechar al máximo la simulación del laboratorio virtual y garantizar un rendimiento estable y visualmente atractivo, ten en cuenta las siguientes recomendaciones:

Motivo	Recomendación
El tiempo de ejecución y la fluidez de los fotogramas por segundo (FPS) dependen directamente de la forma	Antes de cada prueba importante, realiza una compilación "limpia" (Clean + Rebuild) desde Visual Studio 2022

<p>en que el código está compilado y optimizado.</p>	<p>para eliminar artefactos de compilaciones previas que puedan generar inconsistencias. También si dispones de un procesador multinúcleo, habilita la compilación en paralelo (en Visual Studio: Proyecto → Propiedades → C/C++ → Compilación paralelo), acelerando significativamente el tiempo de build y permitiéndote iterar con mayor rapidez.</p>
<p>Texturas de alta resolución y múltiples fuentes de luz pueden saturar la GPU y la memoria de video (VRAM), causando caídas de FPS o incluso inestabilidad.</p>	<p>Para reducir la resolución de las texturas emplea formatos comprimidos (por ejemplo, DXT1/5 o ASTC en hardware compatible) para mantener la calidad visual con menor consumo de VRAM. Puedes crear versiones de menor tamaño para objetos secundarios o detalles lejanos, reservando 2048×2048 o superiores sólo para elementos de primer plano. Si la escena no requiere énfasis en la iluminación puntual (por ejemplo, para pruebas de animación de modelos), considera desactivar temporalmente el spotlight del proyector con L.</p>
<p>La posición y orientación de la cámara determinan qué aspectos del laboratorio y sus componentes serán más visibles y apreciados durante la animación. Ajustarlos con antelación facilita capturar el ángulo ideal para presentaciones y análisis.</p>	<p>Antes de disparar cualquier secuencia de animación, sitúa la cámara en un punto elevado o lateral según el objeto de interés (por ejemplo, sobre la hilera trasera de mesas para la explosión de monitores). Utiliza la barra espaciadora para restablecer la vista inicial si necesitas reiniciar tu perspectiva y probar un ángulo diferente.</p>

Tabla 8. Consejos de uso.

Siguiendo estos consejos, maximizarás la eficiencia en la compilación y ejecución, controlarás el consumo de recursos en escenas complejas y obtendrás vistas óptimas.

Resolución de Problemas Comunes.

Aunque la simulación está diseñada para ejecutarse de forma fluida, pueden surgir inconvenientes al preparar el entorno o durante la ejecución. A continuación encontrarás una guía detallada para identificar, diagnosticar y solucionar los errores más frecuentes.

Fallos al enlazar bibliotecas (linker errors).

Si al compilar, Visual Studio muestra errores “unresolved external symbol” o “cannot open file” referidos a GLEW, GLFW, Assimp, SOIL2, etc.

Solución:

- Revisa las rutas de librería en **Propiedades → Linker → General → Directorios de bibliotecas adicionales**. Debe apuntar a la carpeta que contiene los .lib correspondientes (por ejemplo, ...\\Configinicial\\Debug\\).
- Verifica las dependencias en **Linker → Input → Entradas** adicionales de dependencias: asegúrate de listar correctamente glew32.lib, glfw3.lib, assimp-vc140-mt.lib y soil2.lib.
- Consulta el tutorial del Ing. Arturo Pérez de la Cruz sobre configuración de OpenGL en Visual Studio, para pasos visuales y ajustes de proyecto:
<https://youtu.be/cHNs1glyUJ4?si=tSQo6ceSe4PhtRvt>

Ventana negra o aplicación sin respuesta.

Si se abre la ventana, pero permanece en negro o no responde a los controles de cámara.

Siga los pasos de diagnóstico:

1. Revisa la consola de salida en Visual Studio (Output window) para mensajes de error: rutas de recursos no encontradas, fallos al crear el contexto OpenGL o excepciones en **glfwInit()**.
2. Actualiza los controladores de la GPU a la última versión desde la web del fabricante (NVIDIA, AMD o Intel). Los Controladores desactualizados pueden causar incompatibilidades con OpenGL 3.3+.
3. Comprueba la compatibilidad de OpenGL: ejecuta una pequeña prueba, por ejemplo, el código de ejemplo de GLEW/GLFW “triangle”, para confirmar que tu tarjeta gráfica soporte la versión requerida.

4. Verifica el directorio de trabajo (Project → Properties → Debugging → Working Directory) y asegúrate de que apunte a la carpeta con recursos (**Configinicial\Debug**), de modo que los shaders, modelos y texturas se carguen correctamente.

Errores de compilación de shaders.

En la ventana se muestra un mensaje de error o la simulación se cierra abruptamente al intentar compilar un shader.

Siga las acciones correctivas:

1. Activa el log de errores de shader imprimiendo **glGetShaderInfoLog()** tras cada compilación. Esto mostrará la línea exacta y la descripción del fallo.
2. Válida rutas y nombres: confirma que las carpetas **Shader/** y **Shader_/** incluyen los archivos **.vs**, **.fs** o **.glsl** referenciados en el código. Recuerda que en Linux los nombres son sensibles a mayúsculas y minúsculas.
3. Revisa la sintaxis GLSL: errores comunes incluyen declarar variables sin uso, olvidar **layout** en variables de entrada/salida o desalineaciones en los **uniforms**.
4. Prueba con shaders simplificados: crea un fragment shader mínimo (por ejemplo, que pinte un color plano) para descartar fallos de versión o incompatibilidades de la GPU.

Modelos o texturas faltantes.

Si parte de la escena aparece sin objetos o sin texturas (modelos invisibles o superficies totalmente blancas/grises).

Realice una verificación de los siguientes puntos:

1. Comprueba la ruta relativa usada para cargar modelos y texturas en la clase **Model**. Por defecto, el código busca en **Configinicial/Models/<carpeta>/**.
2. Nombres de archivo exactos: asegúrate de que los archivos **.obj**, **.mtl** y sus texturas **.png/.jpg** coincidan con la ruta y el nombre (respetando mayúsculas).
3. Permisos de lectura: en Linux/macOS, verifica con **ls -l** que tu usuario tenga acceso de lectura sobre los archivos.

4. Logs de Assimp y SOIL2: imprime los mensajes de error que devuelven ***Assimp::Importer::ReadFile()*** y ***stbi_failure_reason()***, para saber si el fallo está en el parseo o en la carga de datos.

Animaciones inactivas o erráticas.

Al presionar las teclas asignadas (N, X, P, Y, T), no sucede nada o la animación se comporta de forma irregular.

Siga los pasos para depurar el código fuente:

1. Revisa el **KeyCallback**: confirma que cada tecla está mapeada correctamente a su flag (***explosionActive***, ***assemblePC***, ***moveChairs***, etc.) y que esos flags llegan con valor ***true*** a ***Animation()***.
2. Verifica el flujo de fases: imprime en consola el valor actual de la fase (***fase*** o ***state***) de la animación de sillas o gabinetes para asegurarte de que cambia según lo esperado.
3. Jerarquía de llamadas: comprueba que dentro de ***Animation()*** no hayas modificado sin querer la lista de objetos o que no existan ***return*** prematuros antes de actualizar ***glm::mix*** o ***update()***.
4. Delta time estable: asegúrate de que el ***deltaTime*** no sea cero o excesivamente grande, lo cual puede detener o acelerar la animación de forma errática.

Siguiendo estas indicaciones paso a paso podrás resolver la mayoría de los inconvenientes habituales y garantizar que la simulación del laboratorio virtual funcione de manera estable y predecible.

Soporte y Contacto.

Para dudas o reportes de errores, contacta a:

Nombre.	Correo electrónico.	Teléfono.
Del Razo Sánchez Diego Adrián.	dadrs03@hotmail.com	5537299820
Hernández Ramírez Miguel Ángel.	miguelhernandez0532@gmail.com	5551436962
Osorio Ángeles Rodrigo Jafet.	rodri.osorio19@gmail.com	5618316866