320018277
GRP: 05

FINAL PROJECT 2026-1

Montiel Juárez Oscar Iván

November 24, 2025

1 Project Scope

The main objective of this project is the creation of an interactive 3D scene in real-time using the OpenGL 4.0 graphics API and the C++ programming language. The scene is a thematic recreation of the house and room of the protagonists from the animated series "Phineas and Ferb".

The scope of the project covers the following key areas:

- **Modeling and Scenography:** Creation of the iconic house (exterior) and the protagonists' room, complemented by 7 characteristic objects from the series, modeled individually.
- **Asset Management:** Integration of a variety of assets, including:
  - Third-party 3D models (Fish, Phone Booth) obtained from *TurboSquid* under a *Free Non-Profit* license.
  - Textures for said models created manually by the author.
  - Environment textures (walls, floor) generated using Artificial Intelligence tools, with a *Free Non-Profit* license.
  - A *Skydome* background with an HDR (*High Dynamic Range*) texture obtained from *PolyHeaven* ("Citrus Orchard Road" by Jarod Guest and Dimitrios Sawa), under a *Free Non-Profit* license.
- **Kinematic Animation:** Implementation of 5 distinct animation systems, activated by the user, divided into:
  - **Complex Animations (3):**
    1. **Day/Night Cycle:** An integral system where a Sun model rotates, its point light intensifies and dims, the scene's ambient light changes, and an internal lamp turns on automatically during the "night".
    2. **Ceiling Clouds:** A shader-based animation that displaces the UV coordinates of the ceiling texture, simulating moving clouds.
    3. **Newton's Cradle:** A phased kinematic animation simulating the pendulum's movement, synchronized with a custom sound effect.
  - **Simple Animations (2):**
    1. **Fish Swimming:** A composite animation including the fish's translation, a 180° turn (flip) to change direction, and caudal fin oscillation.
    2. **Doors and Curtains:** A state-based animation system (open/closed) that applies translation and rotation transformations to multiple objects simultaneously.
- **Lighting and Rendering:**
  - Implementation of a *Skydome* with an HDR texture and a custom shader.
  - A dynamic lighting system (Blinn-Phong model) managing a directional light and four point lights, with dynamic states.
  - Rendering of opaque and translucent objects (glass, fish tank, sun) handling *blending* and the depth buffer.
- **Interactivity and Sound:**
  - Free camera control in first person (*fly-cam*) using keyboard (WASD) and mouse.

  - Keyboard controls to activate and deactivate the 5 animations.
  - Inclusion of a `.wav` sound effect (created by the author) for the pendulum, using the `windows.h` library.

2 State of the Art in Computer Graphics

The field of real-time computer graphics has experienced significant advances in recent years. This project is situated within the context of modern technologies while maintaining an accessible educational focus.

2.1 Cutting-edge Technologies in the Industry

Current professional industry standards include:

- **Real-Time Ray Tracing:** Technology implemented in RTX graphics cards that simulates the physical behavior of light, allowing for more realistic reflections, refractions, and shadows.
- **DLSS/FSR:** Intelligent upscaling techniques using artificial intelligence to improve performance while maintaining visual quality.
- **Physically Based Rendering (PBR):** Material models that simulate precisely how light interacts with different surfaces based on real physical principles.
- **Global Illumination:** Techniques calculating not only direct light but also indirect light bouncing between surfaces.
- **Real-Time Physics Simulations:** Advanced physics engines calculating collisions, fluids, cloth, and particles realistically.

2.2 Project Positioning

While this project does not implement all the mentioned cutting-edge technologies, it represents a significant step in learning and applying fundamental concepts:

- **Shader Programming:** The use of custom shaders for animations (clouds) and special effects (HDR skydome) demonstrates an understanding of fundamental modern graphics concepts.
- **Dynamic Lighting:** The implemented day/night cycle system, though basic compared to professional solutions, incorporates dynamic lighting principles essential in modern graphics engines.
- **Resource Management:** Integrating HDR textures and efficiently handling multiple 3D models reflects standard practices in graphics application development.
- **Programmatic Animations:** The implemented kinematic animations, while not full physics simulations, show an understanding of interpolations and transformations, which are the basis for more complex animation systems.

2.3 Technological Gap and Improvement Opportunities

The main gap between this project and professional solutions lies in:

- The absence of a full physics engine for dynamic simulations.
- The use of simplified lighting models instead of PBR.
- The lack of advanced optimization techniques such as Level of Detail (LOD) and frustum culling.
- The manual implementation of systems that are handled by established graphics engines in professional environments.

However, this project serves as a fundamental basis for understanding the principles underlying advanced technologies, providing a solid platform for future specialization in graphics engines like Unreal Engine 5 or Unity, which build upon these same fundamental concepts but with optimized implementations and additional features.

3 Limitations

Despite the achieved results, the project encounters the following technical and scope limitations:

- **Optimization and Performance:** The project is optimized for execution on a single machine (the development equipment). A stable *framerate* is not guaranteed on hardware with lower specifications.
- **Physics Simulation:** All animations are **kinematic** representations (following a predefined script) and not **dynamic**. No physics engine was implemented to calculate collisions, gravity, or real momentum transfer (e.g., in the pendulum).
- **Portability:** The source code uses specific Windows libraries (`windows.h` and `winmm.lib`) for sound playback. The project is not portable to macOS or Linux without refactoring this functionality.
- **Asset Licensing:** The use of models and textures under *Free Non-Profit* licenses restricts the use of this project strictly to academic and non-commercial purposes.
- **Advanced Rendering:** *Tone mapping* is only applied basically to the *Skydome*. The main scene does not use a full HDR *pipeline*, so the high intensity of the sky does not globally "illuminate" the scene, and intense brightness (like the sun or the lamp) saturates to white (1.0).

4 Applied Software Methodology

For a project of this nature, with visual and functional requirements constantly evolving ("make the light brighter", "change the sun rotation"), an **Iterative and Incremental Development Methodology** was applied.

This approach allows building the software by functional modules, testing each component before integrating the next. The project workflow was as follows:

1. **Phase 1: Setup and Base Prototype**
    - Environment configuration (GLEW, GLFW, GLM, `stb_image`).
    - Creation of the window and helper classes `Shader`, `Camera`, and `Model`.
    - Implementation of the free camera and loading a test model.
2. **Phase 2: Static Scene Loading**
    - Loading all static models (house, room, furniture) in their positions.
    - Implementation of a static lighting system (one directional light) and transparency rendering (windows).
3. **Phase 3: Incremental Animation Development (Iterations)**
    - **Iteration 1 (Curtains and Doors):** The simplest animation logic is implemented. A boolean *flag* (`curtainClosed`) and a `closingFactor` (0.0 or 1.0) are defined. In the draw loop, a `glm::translate` or `glm::rotate` transformation multiplied by this factor is applied.
    - **Iteration 2 (Fish):** A composite animation is implemented. State variables are added (`fishSwimming`, `fishXOffset`, `fishFlipped`, `tailAngle`). Logic for translation, the state machine for the 180° flip, and tail oscillation based on `sin()` are developed.
    - **Iteration 3 (Pendulum):** A phased kinematic animation is implemented (`currentPhase`, `phaseTime`) using trigonometric functions. The `PlaySound` sound is added at the phase change.
    - **Iteration 4 (Clouds):** A new pair of shaders (`clouds.vs/frag`) is created. The animation is delegated to the GPU, passing `time` and `cloudsMoving` as *uniforms*.
    - **Iteration 5 (Day/Night Cycle and Skydome):** The most complex iteration.
        - The `skydomeShader` is created, and HDR rendering logic is implemented (`stb_loadf`, `GL_RGB16F`).
        - The Sun model and its rotation based on `sunAngle` are added.
        - The lighting block in `main.cpp` is refactored to make it dynamic, creating the `dayFactor` and switch logic (Sun vs. Lamp).
4. **Phase 4: Refinement and Documentation**
    - Fine-tuning of all parameters (speed, color, light intensity).
    - Debugging rendering order (translucents) and texture management (texture "bleeding" issues).
    - Writing technical documentation.

5 Project Planning (Gantt Chart)

To manage project development, a Gantt Chart was used to break down tasks over 8 weeks, from September 17th to November 11th, plus an additional week for delivery. This approach allowed visualizing the workload and ensuring both modeling and animation programming were completed on time.

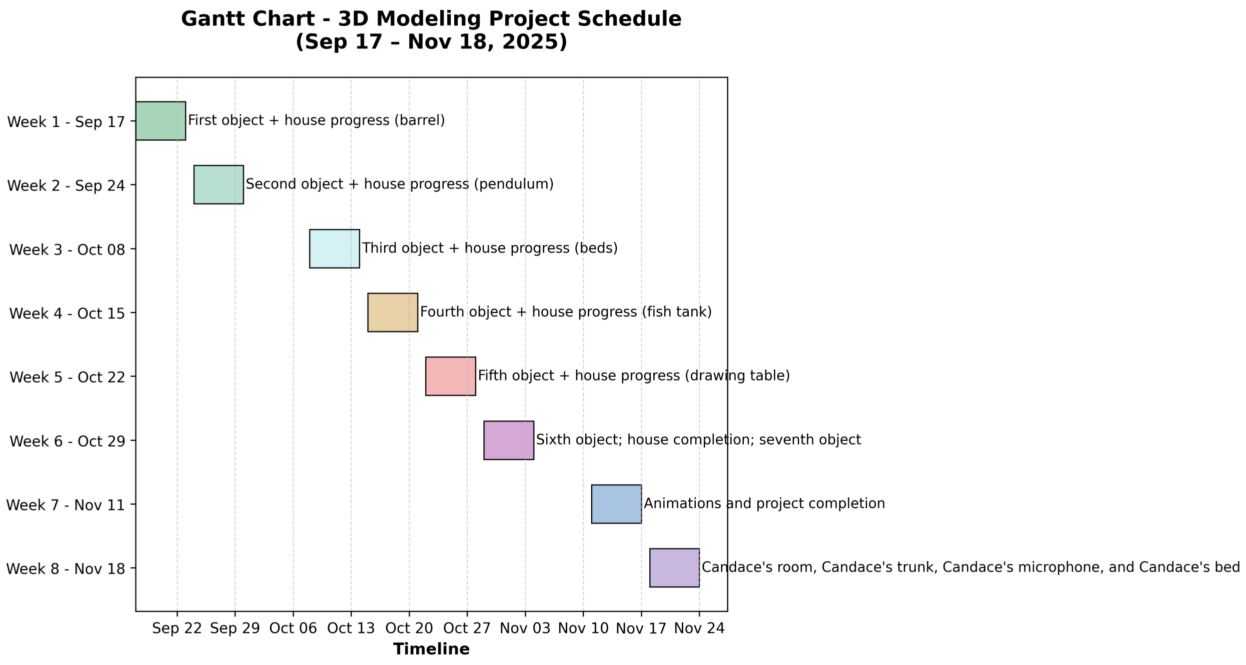

Figure 1: Project Gantt Chart, showing task distribution over 8 weeks.

As seen in the diagram (Figure 1), the work methodology was incremental:

- **Weeks 1 to 5:** Dedicated to the progressive creation of the 7 required objects, assigning approximately one week to each main object (Barrel, Pendulum, Beds, Fish Tank, Drafting Table). In parallel, general progress on the house (structure, walls, interiors, curtains) was made.
- **Week 6:** This was a crucial integration week, where the final objects (Desk and Lamp) were finished, and house modeling was completed 100%.
- **Week 7:** Reserved entirely for the implementation of the 5 animations (Pendulum, Fish, Sun, Clouds/Curtains), final testing, and delivery preparation.
- **Week 8:** Modeling of Candace's room was performed, including the trunk and microphone. Additionally, two objects (the booth and Phineas's desk) were moved to Candace's room to enrich the scene.

This planning aligns with the iterative methodology, allowing the scene to be built by functional parts (models) before integrating the final layer of complexity (animations).

6 Cost Analysis

This section presents a breakdown of costs associated with the project development, as well as a proposed sale price. It is important to highlight that the project was carried out by a single developer (the author) and that tools and assets under free licenses for educational purposes were used.

6.1 Development Costs

- **Labor:** The project took 8 weeks of active development, plus approximately 4 months of learning and theory (duration of a semester). Considering this is my first professional project and I seek to establish my portfolio in the market, my time has been valued at an entry rate of \$25 MXN/hour. This rate reflects my current level as a student looking to gain experience and build a solid portfolio. The hour calculation is:
  - Development: 8 weeks $\times$ 20 hours/week $=$ 160 hours.
  - Learning: 16 weeks $\times$ 10 hours/week $=$ 160 hours.
  - Total hours: $160 + 160 = 320$ hours.

  Labor cost: 320 hours $\times$ 25 MXN/hour $= 8,000$ MXN.
- **Equipment:** A computer with a 3050ti graphics card, new generation Ryzen 7, and liquid cooling was used. Considering a computer cost of \$24,000 MXN and a useful life of 3 years (156 weeks), the weekly depreciation cost is $24,000/156 \approx 153.85$ MXN/week. The project took 8 weeks of development, so the depreciation cost is $8 \times 153.85 \approx 1,230.80$ MXN.
- **Electricity:** Considering the computer consumes approximately 500W under load and was used 20 hours a week for 8 weeks, total energy consumption is $500W \times 20$ hours/week $\times$ 8 weeks $=$ 80 kWh. Assuming a cost of \$3.00 MXN/kWh, electricity cost is $80 \times 3.00 =$ 240 MXN.
- **Software:**
  - **Autodesk Maya:** The free educational version was used. Cost: \$0 MXN.
  - **Blender:** Although not used, it is a free alternative. Cost: \$0 MXN.
  - **Visual Studio:** Free development environment. Cost: \$0 MXN.
  - **OpenGL:** Free graphics API. Cost: \$0 MXN.
- **Assets:**
  - **3D Models:** Fish and phone booth models were obtained from TurboSquid under Free Non-Profit license. Cost: \$0 MXN.
  - **Textures:** Textures for models and environment were created manually or generated with AI under Free Non-Profit license. Cost: \$0 MXN.
  - **Skydome HDR:** Obtained from PolyHeaven under Free Non-Profit license. Cost: \$0 MXN.

6.2 Total Costs

- Labor: \$8,000 MXN
- Equipment (depreciation): \$1,230.80 MXN
- Electricity: \$240 MXN
- Software: \$0 MXN
- Assets: \$0 MXN
- **Total:** \$9,470.80 MXN

6.3 Sale Price

Considering this is my first professional project and I seek to establish my presence in the market, an entry-level pricing strategy has been chosen. This strategy recognizes that, although the real value of the work is higher, it is important to offer a competitive price to gain experience and build a solid portfolio.

- Total Cost: $9,470.80 MXN
- Profit Margin (20%): $1,894.16 MXN
- **Suggested Sale Price:** $11,364.96 MXN (approximately $635 USD)

However, understanding the nature of an initial project and my goal to enter the market, a promotional price of $9,500 MXN (approximately $530 USD) could be considered for clients interested in this type of development.

6.4 Justification of Entry-Level Pricing

The reduced price is justified for the following reasons:

- **First Professional Project:** As a developer entering the market, I recognize that my main value at this moment is the demonstration of technical skills rather than extensive experience.
- **Portfolio Investment:** This project represents an investment in my personal portfolio, so I am willing to accept a lower financial return in exchange for gaining exposure and experience.
- **Competitiveness:** In the junior developer market, it is essential to offer competitive prices reflecting both work quality and the current professional stage.
- **Value for Money:** Although the price is entry-level, the project demonstrates advanced computer graphics skills, including complex animations, custom shaders, and resource management, offering excellent value for the price.
- **Accessibility:** By maintaining a low price, I seek to make the project accessible to clients or studios that might be interested in hiring a junior developer with demonstrated skills.

6.5 Justification for Using Autodesk Maya

Autodesk Maya was chosen for 3D modeling for the following reasons:

- **Free Educational License:** As a student, I have access to a free license of Maya, allowing the use of an industrial tool at no cost.
- **Interface and Workflow:** Maya is widely used in the film and video game industry, and learning its workflow is valuable for my professional training.
- **Modeling Tools:** Maya offers a robust set of modeling, texturing, and animation tools that are industry standard.
- **Pipeline Integration:** Although a complex pipeline was not used in this project, Maya allows integration with game engines and advanced rendering software.

- **Market Readiness:** Mastery of Maya is a valued skill in professional studios, so its use in this project contributes to my preparation for future job opportunities.

Although Blender is a very capable free and open-source alternative, the decision to use Maya was based on the availability of the educational license and the desire to gain experience in a tool widely used in professional studios.

7 Software Flowchart

The program's execution flow centers on a main loop (Game Loop) executed in each frame. This loop is responsible for updating the state, processing user input, and rendering the scene.

Note: The complete visual flowchart, generated from the source code, is annexed at the end of this document.

The logical flow of the program, as detailed in the diagram, follows these main steps:

1. **Start and Setup:**
   - The program starts and initializes fundamental libraries: `srand` for randomness, `glfwInit` for the window, and `glewInit` for OpenGL functions.
   - The window is created, and keyboard and mouse *callbacks* are configured.
   - All resources are loaded into memory: Shaders (lighting, skydome, clouds), 3D Models (.obj), and Textures (including HDR for the sky and black/white prop textures).
2. **Main Loop (`while` loop):**
   - The program enters the main loop as long as the window should not close.
   - **Logic Update:** `deltaTime` is calculated. The status of all active animations (Pendulum, Clouds, Fish, Sun) is checked, and their state variables (times, angles, positions) are updated.
   - **Input Processing:** Keyboard events (WASD, keys 1-5) are processed, and the camera is updated.
3. **Rendering Pipeline (Per Frame):**
   - **Step 1: Clear Screen.** Color and depth buffers are cleared (`glClear`).
   - **Step 2: Draw Skydome.** The `skydomeShader` is activated, and the sky sphere is drawn first, using a depth test of `GL_LEQUAL` to ensure it always stays in the background.
   - **Step 3: Configure Lights.** The `lightingShader` is activated. The `if (sunAnimationActive)` logic executes to determine the intensity of the directional light and point lights (Sun and Lamp).
   - **Step 4: Draw Opaque Objects.** With `glDepthMask(TRUE)` and `glDisable(BLEND)`, all solid objects (house, furniture, grass, etc.) are drawn.
   - **Step 5: Draw Translucent Objects.** `glEnable(BLEND)` is activated, and depth writing is disabled (`glDepthMask(FALSE)`). Glass, the fish tank, and the sun model are drawn in order.
   - **Step 6: Swap Buffers.** The rendered image is shown on the screen (`glfwSwapBuffers`).
4. **End of Program:**
   - Upon exiting the loop (user closes window), all resources (textures) are freed, and GLFW is terminated.

8 Code Documentation

Technical documentation consists not of explaining every line of code, but of describing the **architecture** of key systems and the **logic** behind their implementation.

8.1 Asset Management and Licensing

A fundamental component of the project is the correct management of 3D assets.

- **Internet Models:** Two models from the *TurboSquid* platform were used under *Free Non-Profit* license without direct author attribution: the fish and the phone booth. **Textures for these models were created manually by the author**.
- **AI Textures:** Scene textures (walls, floors) were generated using artificial intelligence tools and used under a *Free Non-Profit* license.
- **Skydome HDR:** The sky background is a high-quality asset from *PolyHeaven*. The model used is **"Citrus Orchard Road (pure sky)"** by artists **Jarod Guest and Dimitrios Sawa**, used under the platform's *Free Non-Profit* license.
- **Sound:** The pendulum sound effect is an original recording by the author, created by hitting two magnets together, processed in the `golpe_esferas.wav` file.

8.2 Software Structure

The project centers on a `main.cpp` file acting as an orchestrator, supported by standard OpenGL helper classes:

- `main.cpp`: Contains the *Game Loop*, all animation state logic, GLFW *callbacks* management (keyboard/mouse), and rendering order.
- `Shader.h`: Helper class to load, compile, and activate GLSL shader programs.
- `Camera.h`: Class managing the free camera (position, orientation) and calculating the *View* matrix.
- `Model.h`: Class (based on Assimp) loading `.obj` files, managing their meshes and textures.
- `stb_image.h`: Single-header library (included in `main.cpp`) vital for loading textures, especially the `skybox/cielo.hdr` file thanks to its `stbi_loadf` function.

8.3 The Main Loop (Game Loop)

The order of operations within the `while` loop is critical for correct functioning:

1. **Logic Update:** `deltaTime` is calculated, and state values of all active animations (e.g., `sunAngle`, `phaseTime`, `fishXOffset`) are updated.
2. **Input and Matrices:** Events are processed (`glfwPollEvents`), and master `view` and `projection` matrices are calculated.
3. **Drawing the Skydome:** Drawn **first** of all, using its own shader (`skydomeShader`) and depth tricks (`glDepthFunc(GL_LEQUAL)`) and view matrix tricks (`glm::mat4(glm::mat3(view))`) to make it appear infinite.

4. **Activating Scene Shaders:** The `lightingShader` and `cloudsShader` are activated, passing `view` and `projection` matrices.
5. **Lighting Configuration:** The `if (sunAnimationActive)` logic executes, determining the intensity of all scene lights based on the `dayFactor`.
6. **Drawing Opaque Objects:** All solid models are drawn with `glDepthMask(GL_TRUE)` (house, furniture).
7. **Drawing Translucent Objects:** `glEnable(GL_BLEND)` and `glDepthMask(GL_FALSE)` are activated. Glass, the fish tank, and the sun model are drawn.
8. **Swap Buffers:** Buffers are swapped.

8.4 Animation Architecture (by System)

8.4.1 System 1: Doors and Curtains (Simple)

- **Logic:** Based on a simple boolean state, `doorOpen`.
- **Activation:** The `KeyCallback` (key '5') inverts the `doorOpen` state and updates `closingFactor` to `1.0f` (closed) or `0.0f` (open).
- **Execution:** In the draw loop, each curtain's `model` matrix is multiplied by a translation. This animation is binary and not interpolated.

Listing 1: Door animation logic (rotation)

```
glm::vec3 pivotPuerta(-3.975f, 5.921f, 0.4826f);
float rotPuerta = puertaAbierta ? 90.0f : 0.0f; // 90 degrees or 0

glm::mat4 modelPuerta = glm::mat4(1.0f);
modelPuerta = glm::translate(modelPuerta, pivotPuerta);
modelPuerta = glm::rotate(modelPuerta, glm::radians(rotPuerta),
    glm::vec3(0.0f, 1.0f, 0.0f));
modelPuerta = glm::translate(modelPuerta, -pivotPuerta);
// ...
puerta_cuarto_phineas.Draw(lightingShader);
```

8.4.2 System 2: Fish (Simple)

- **Logic:** A simple state machine with two independent animation components.
- **Component 1 (Tail):** Constant oscillatory animation. The `tailAngle` variable updates every frame using `sin(tailTime)`.

Listing 2: Fish tail oscillation logic.

```
// tailTime increments with deltaTime
tailAngle = TAIL_AMPLITUDE * sin(tailTime * TAIL_FREQ);
// ...
// Apply rotation at the tail pivot
tailModel = glm::translate(tailModel, tailPivotLocal);
tailModel = glm::rotate(tailModel, glm::radians(currentTailAngle),
    glm::vec3(0.0f, 1.0f, 0.0f));
tailModel = glm::translate(tailModel, -tailPivotLocal);
```

- **Component 2 (Body):** State machine for translation and turning.
  - **Translation State:** The `fishXOffset` variable increments or decrements with `FISH_FORWARD_SPEED`. The direction depends on the `fishFlipped` boolean.
  - **Turn State (Flip):** A timer `fishSwimTime` increments. Upon reaching `FLIP_TIME`, the `isFlipping` boolean is activated.
  - During the flip, `flipRotation` interpolates from 0 to 180 (or 180 to 360) based on `flipProgress`.
  - Upon finishing the flip, `isFlipping` becomes false, and `fishFlipped` is inverted, changing translation direction.

8.4.3 System 3: Newton's Cradle (Complex)

- **Logic:** Phased kinematic animation, not a physics simulation.
- **State:** Controlled by `currentPhase` (int) and `phaseTime` (float).
- **Execution:** An `if/else if` checks the current phase:
  - **Phase 0 (Start):** Sphere 1 (`angle1`) moves from -30° to 0° using `cos(omega * phaseTime)`.
  - **Odd Phases (1, 3, ...):** Sphere 4 (`angle4`) oscillates from 0° to 30° and back to 0° using `sin(omega * phaseTime)`.
  - **Even Phases (2, 4, ...):** Sphere 1 (`angle1`) oscillates from 0° to -30° and back to 0° using `sin(omega * phaseTime)`.
- **Subtle Detail:** Middle spheres (2 and 3) move slightly (`angleMiddle`) to give an impact sensation.
- **Sound:** When a phase ends (`phaseTime >= halfPeriodTime`), `advancePhase` activates, `currentPhase` increments, and `PlaySound` is called to play the custom `.wav` (magnet clash).

8.4.4 System 4: Cloud Ceiling (Complex - Shader)

- **Logic:** This animation is delegated almost entirely to the GPU, making it very efficient.
- **CPU (`main.cpp`):** The C++ code simply activates the `cloudsShader` and passes two *uniforms*: `cloudsMoving` (boolean) and `cloudsTime` (float).
- **GPU (Shader `clouds.vs`):** The Vertex Shader performs the animation.

Listing 3: Vertex Shader (clouds.vs) - Animation Logic

```
// ...
uniform float time;
uniform bool cloudsMoving;
uniform float speed;
// ...
```

```
void main()
{
    // ...
    if (cloudsMoving) {
        // mod() creates a repeating value from 0.0 to 1.0
        float offset = mod(time * speed, 1.0);
        // Displace the UV coordinate on the X axis
        TexCoords = vec2(uv.x - offset, uv.y);
    } else {
        TexCoords = uv;
    }
    gl_Position = projection * view * model * vec4(aPos, 1.0);
}
```

- **GPU (Shader `clouds.frag`):** The Fragment Shader implements manual "alpha testing". The ceiling texture has transparent parts (low alpha). The `discard` line tells the GPU not to draw those pixels, creating the cloud "cutout" effect.

Listing 4: Fragment Shader (clouds.frag) - Alpha Discard

```
// ...
void main()
{
    vec4 texColor = texture(texture1, TexCoords);
    // If pixel is very transparent, do not draw it
    if (texColor.a < 0.1)
        discard;
    FragColor = texColor;
}
```

8.4.5 System 5: Day/Night Cycle and Skydome (Complex)

- **Logic:** The most complex system, combining three components: sky rendering, sun animation, and global lighting logic.
- **Component 1 (Skydome Rendering):**
  - **Loading (C++):** `stbi_loadf` is used to load `cielo.hdr` into a `float` buffer. It is uploaded to the GPU with internal format `GL_RGB16F`.
  - **Rendering (C++):** Drawn **first**. `glDepthFunc(GL_LEQUAL)` is activated so the dome is only drawn on background pixels (where Z=1.0).
  - **Shader (`skydome.vs`):** Implements two key tricks. First, removes camera translation (`view = glm::mat4(glm::mat3(view))`) so the user never approaches the sky. Second, forces the sphere's Z coordinate to 1.0 (`gl_Position = pos.xyww;`) so it is always behind all other objects.

Listing 5: Vertex Shader (skydome.vs) - Depth Trick

```
// ...
void main()
```

```
{
    TexCoords = aTexCoords;
    vec4 pos = projection * view * vec4(aPos, 1.0);
    // pos.xyww makes z = w. After perspective division,
    // z/w becomes w/w = 1.0 (the background)
    gl_Position = pos.xyww;
}
```

- **Component 2 (Sun Animation):**
  - **Position (C++):** `sunAngle` updates with `deltaTime`. A new `sunPos` is calculated using `glm::rotate` on the X axis.
  - **Drawing (C++):** The `sol` model is drawn in the translucent pass, applying the *same* rotation matrix used to calculate `sunPos`.
- **Component 3 (Dynamic Lighting):**
  - **The "Day Factor" (C++):** This is the key variable. Calculated using `dayFactor = sin(glm::radians(sunAngle))` for the 0° to 180° range. This gives a smooth interpolation from 0 (night) to 1 (noon) and back to 0 (night). From 181° to 359°, `dayFactor` remains at 0.
  - **Interpolation (C++):** Ambient light (`dirLight`) and sun intensity (`pointLights[0]`) are linearly interpolated (using `glm::mix`) between their night values (almost 0) and their day values (maximum).
  - **Switch (C++):** The system acts as a switch. If `dayFactor > 0.0` (it is day), the lamp (`pointLights[1]`) turns off. If `dayFactor == 0.0` (it is night), the lamp turns on.

9 Project Gallery (Results)

Below is a series of screenshots illustrating the final project results, showing the scene, models, and implemented animations.

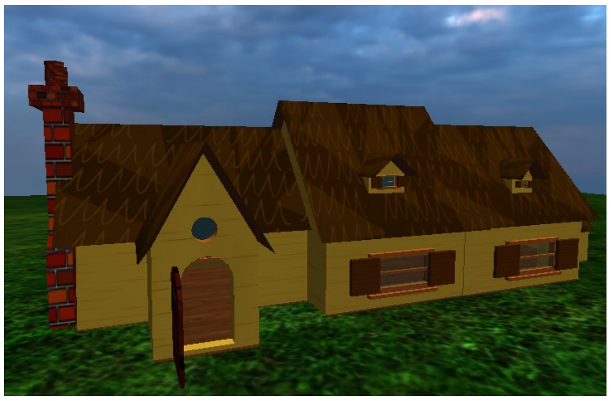
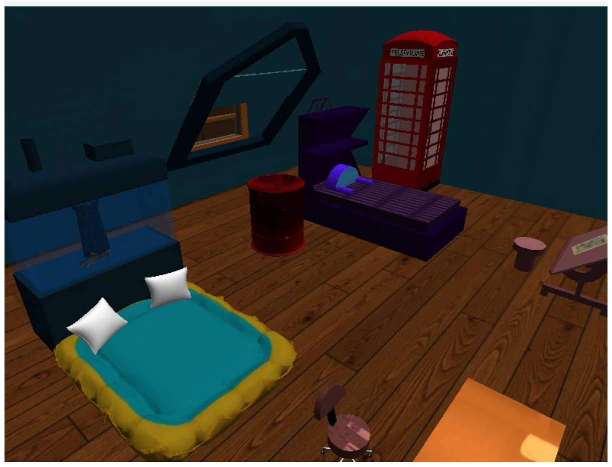

Figure 2: General views of the scene: (Left) House exterior. (Right) Interior of Phineas and Ferb's room.

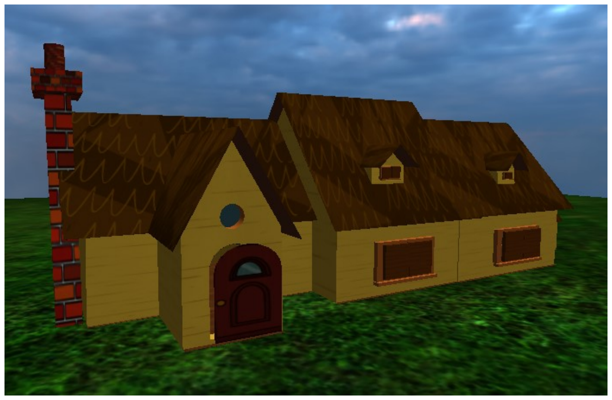
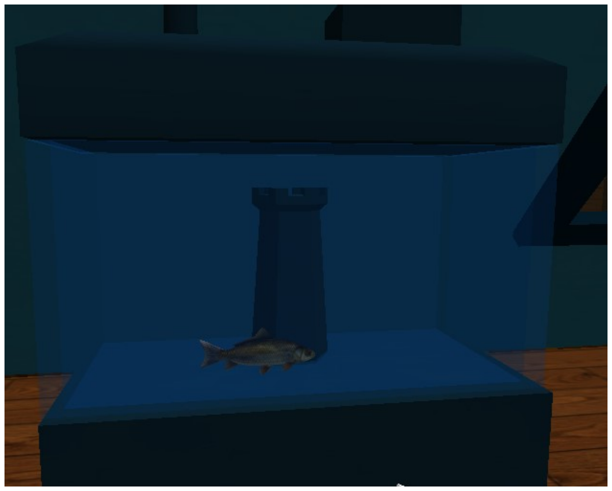

Figure 3: Simple animations: (Left) Door and curtain opening/closing logic (closed state). (Right) Fish swimming animation in the tank.

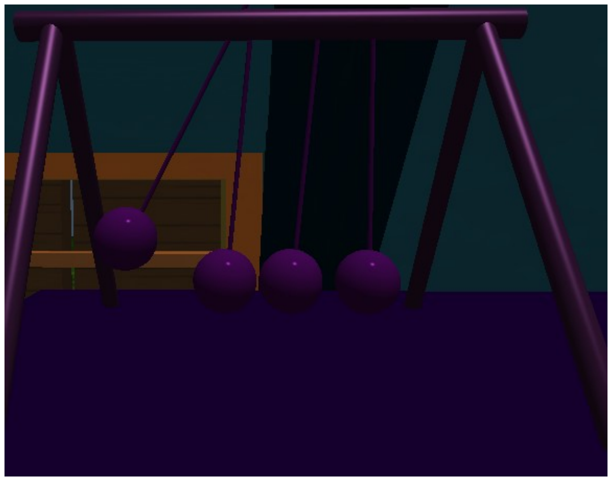
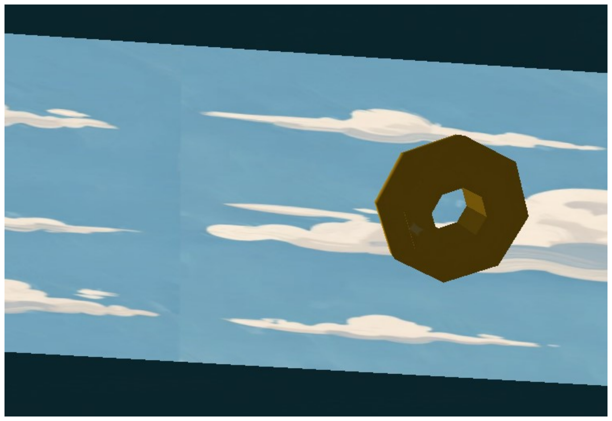

Figure 4: Complex animations: (Left) Phased kinematic animation of Newton's cradle. (Right) Shader-controlled cloud ceiling animation.

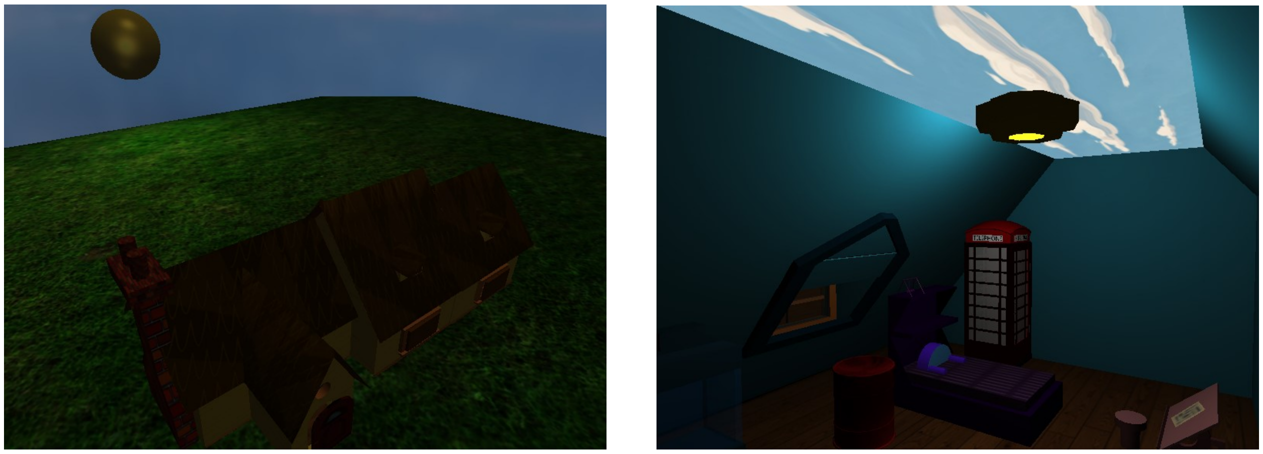

Figure 5: Day/Night cycle system: (Left) The sun visible during its day rotation. (Right) Night scene, where the sun has set and the desk lamp turns on automatically.

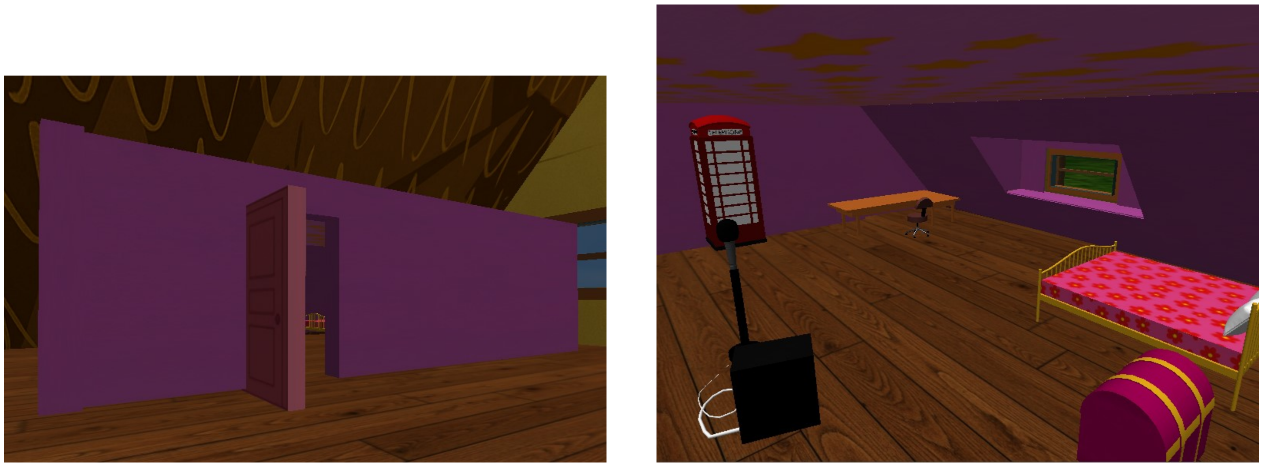

Figure 6: Project expansion - Candace's Room: (Left) Exterior view of Candace's room. (Right) Interior view showing space and furniture arrangement.

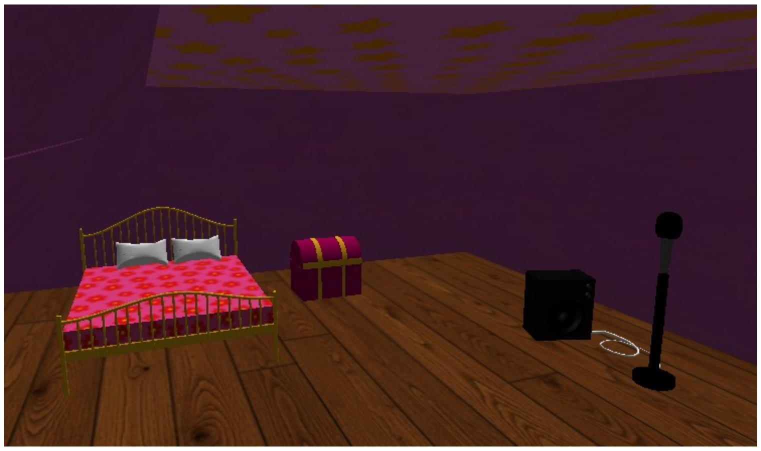

Figure 7: Candace's room details: Showing objects modeled specifically for this space - the bed, the characteristic trunk, and the microphone, iconic character elements.

10 Conclusions

The main objective of the project was achieved, developing a functional, interactive 3D scene aesthetically consistent with the "Phineas and Ferb" theme. The 7 required object models and 5 animation systems (3 complex and 2 simple) were successfully implemented.

The project development allowed for the application and deep learning of fundamental real-time computer graphics concepts, highlighting:

- **Rendering Architecture:** Handling drawing order, managing color and depth buffers (`glDepthMask`), and using `glDepthFunc` for the *Skydome*.
- **Dynamic Lighting:** Implementing a lighting system that evolves over time, combining directional and point lights, and altering the scene state (day/night).
- **Shader Programming (GLSL):** Using multiple shader programs for different purposes (`lighting`, `skydome`, `clouds`) and passing *uniforms* to control animations (`time`) and logic (`discard`).
- **Asset Management:** Successful integration of third-party models (Fish, Booth) with custom textures, and use of AI-generated textures, respecting non-commercial licenses.
- **HDR Rendering:** Loading and rendering high dynamic range textures (`.hdr`) with `stb_image` and configuring floating-point texture formats (`GL_RGB16F`).
- **Audio:** Integration of an original sound effect (magnet clash) in a kinematic animation, synchronized with program events.

As future work, the project could be expanded to include more advanced techniques, such as a global *tone mapping* system for the entire scene (not just the sky), implementation of shadows (*shadow mapping*) projected by the dynamic sun, and rendering optimization via *instancing*.

Diagrama de Flujo - main.cpp (Proyecto Final OpenGL)

Inicializar srand()
glfwInit()
Crear ventana
Ventana creada?
no
Error: glfwTerminate()
Configurar callbacks (teclado, ratón)
glewInit()
GLEW OK?
no
Error
Configurar OpenGL
Cargar shaders (lighting, skydome, etc.)
Cargar modelos 3D
Cargar textura HDR (cielo)
Crear texturas blanco/negro
Proyección = perspective()
Bucle Principal
no
¡glfwWindowShouldClose?
sí
Calcular deltaTime
Péndulo activo?
sí
Actualizar phaseTime
Calcular ángulos esferas
Fase completa?
sí
currentPhase++
phaseTime = 0
Reproducir sonido
Nubes activas?
sí
cloudsTime += deltaTime
Pez nadando?
sí
Actualizar timers
Tiempo para flip?
sí
Iniciar flip
Animar cola (sin)
Mover pez (izq/der)
Sol animado?
sí
sunAngle += velocidad * deltaTime
Calcular sunPos
Interpolar luces día/noche
Procesar entrada (WASD, teclas 1-5)
Actualizar cámara
glClear()
Usar skydomeShader
Dibujar skydome (HDR)
Usar lightingShader
Configurar dirLight, pointLights, spotLight
glDisable(BLEND); glDepthMask(TRUE)
Dibujar:
- casa, paredes, techo
- puertas (con rotación)
- cortinas (factorCierre)
- mesa, escritorio, barril
- camas, pasto, lámpara
- cabina, péndulo
- esferas (con pivotes)
Matriz cuerpo (flip + offset)
Dibujar cuerpo
Aplicar rotación cola
Dibujar cola
glEnable(BLEND); glDepthMask(FALSE)
Dibujar:
- ventanas (α=0.20)
- pecera vidrio (α=0.15)
- cabina vidrio (α=0.40)
- sol (α=0.75, rotación)
Restaurar estado opaco
glfwSwapBuffers()
Liberar texturas
glfwTerminate()