

Faculdade de Ciências da Universidade de Lisboa

Departamento de Informática

Computação Gráfica – 2025/2026

Relatório do Projeto Final: Cena 3D Interativa

Leonardo Pardal 61836
Afonso Henriques 61826
Pedro Carvalho 61800

7 de dezembro de 2025

Conteúdo

1 Identificação e Esforço	2
2 Descrição da Aplicação e Mecanismos de Interação	2
2.1 Descrição introdutória do projeto	2
2.2 Estrutura de Ficheiros e Responsabilidades	2
2.3 Controlos:	4
3 Requisitos	4
3.1 Requisitos Principais	5
3.2 Requisitos Secundários	6
4 Descrição do Grafo de Cena	7
4.1 Grafo de Cena	7
4.2 Processamento e Renderização	7
4.2.1 Ciclo de Atualização Lógica (Update)	7
4.2.2 Pipeline de Renderização (Render)	8
4.3 Detalhe da Implementação dos Nós e Transformações	8
4.4 Descrição Detalhada das Animações	9
4.4.1 Cinemática e Hierarquia do Veículo	9
4.4.2 Mecanismo da Garagem	11
4.4.3 Follow Camera	11
4.4.4 Ciclo Solar (Dia/Noite)	11
4.4.5 Cinemática do Avião:	12
4.5 Relação Nós e Materiais	12
5 Screenshots Comentados	13
5.1 Nodes Principais da Cena	13
5.2 Direção das Rodas	15
5.3 Animação de Portas do Carro (K/L)	15
5.4 Animação da Porta da Garagem (F)	16
5.5 Alternância de Modos de Câmera (SPACE)	17
5.6 Objetos com Animação Automática	17
5.7 Skybox e Variações de Ambiente	18
5.8 Influência do Node sol como fonte de luz	18
5.9 Cena Completa	19
6 Link para o Código Fonte e para o vídeo do Youtube	19

1 Identificação e Esforço

- **Aluno:** [Leonardo Pardal] ([61836])
- **Horas dispendidas:** [20-30] horas
- **Aluno:** [Afonso Henriques] ([61826])
- **Horas dispendidas:** [20-30] horas
- **Aluno:** [Pedro Carvalho] ([61800])
- **Horas dispendidas:** [20-30] horas

2 Descrição da Aplicação e Mecanismos de Interação

2.1 Descrição introdutória do projeto

O presente projeto consiste no desenvolvimento de uma cena 3D interativa que implementa todos os requisitos propostos no enunciado (ver enumeração e implementação na Secção 3). Para a concretização do projeto, utilizou-se a linguagem **Python** em conjunto com a API gráfica **OpenGL 3.3**, recorrendo a bibliotecas auxiliares como **PyOpenGL**, **GLFW**, **pyrr** **numpy**.

A arquitetura da cena baseia-se num Grafo de Cena (*Scene Graph*) hierárquico (detalhado na Secção 4.1), integrando uma *Skybox* com texturas alternáveis e diversos objetos com materiais e texturas distintas. O sistema de iluminação implementa o modelo de reflexão de **Blinn-Phong**, suportando múltiplas fontes de luz.

Além da animação principal do veículo, a cena inclui animações em dois objetos adicionais: um avião e o sol (que orbita a cena). A interação com o utilizador permite o controlo total do veículo (movimentação e direção) e oferece três modos de câmara: livre, perseguição (*follow*) e primeira pessoa (visão do condutor). Adicionalmente, foram implementadas interações com partes móveis, como a abertura das portas do carro e do portão da garagem, bem como a funcionalidade de alternar para o modo de ecrã inteiro. (*fullscreen*).

2.2 Estrutura de Ficheiros e Responsabilidades

O código fonte foi modularizado para garantir a separação de responsabilidades e facilitar a manutenção. A estrutura principal do projeto (**src/**) organiza-se da seguinte forma:

- **main.py:** Ponto de entrada da aplicação e manipulador do sistema.
 - **Inicialização:** Configura a janela GLFW, o contexto OpenGL e os *callbacks* de input (teclado e rato).
 - **Construção da Cena:** Instancia todos os nós, *mesh* e materiais iniciais através da função `build_scene` que recebe as estruturas a usar.
 - **Ciclo Principal :** Gere o cálculo do tempo (`deltaTime`), invoca as atualizações lógicas e ordena a renderização a cada *frame*.
- **node.py:** Define a classe `Node`, a unidade fundamental do Grafo de Cena.
 - **Hierarquia:** Gere a lista de nós filhos e a referência para o nó pai.
 - **Transformações:** Armazena a matriz de transformação local e permite a sua modificação, permitindo ter animações e materiais para os objetos.
 - **Propagação:** Implementa o método `update()` recursivo para propagar lógica pela árvore.

- `renderer.py`: Motor de renderização que abstrai a API OpenGL.
 - **Travessia**: Percorre o grafo para desenhar os objetos visíveis.
 - **Gestão de Luzes**: Identifica e recolhe todas as fontes de luz ativas na cena antes do desenho.
 - **Configuração de Shaders**: Envia as matrizes globais (Model, View, Projection) e uniformes de iluminação para a GPU.
- `anim.py`: Módulo de lógica e comportamentos dos *nodes*.
 - **Fábrica de Animadores**: Contém funções para comportamentos específicos(ex: `make_sun_animator`, `make_door_anim`), permitindo definir o comportamento de forma separada da essência do *node*.
- `carro.py`: Simulação física do veículo.
 - **Estado Físico**: Mantém variáveis como posição, velocidade, aceleração e ângulo de direção.
- `camera.py`: Sistema de câmaras virtuais.
 - **Modos**: Implementa a lógica para câmara livre (`free`), câmara de perseguição atrás do carro (`follow`) e câmara dentro do carro (`inside`)
 - **Matrizes**: Calcula a matriz *View* (LookAt) baseada na posição e alvo da câmara.
- `material.py`: Define a classe material.
 - **Propriedades**: Encapsula cor difusa, especular, brilho e texturas, recebendo um shader.
 - **Binding**: Método responsável por carregar estas propriedades para o *shader* ativo.
 - **Escala**: Permite definir um shader básico para cada objeto mas que terá valores diferentes atribuídos consoante o material definido.
- `geo.py`: Geração procedural de geometria.
 - **Primitivas**: Cria arrays de vértices, normais e UVs para formas básicas como planos, cubos e esferas.
 - **Escala**: Permite criar objetos e formas simples que são usadas para alguns *nodes*.
- `obj.py`: Importação de ativos 3D.
 - **Parser**: Lê ficheiros no formato *Wavefront* (.obj) e converte-os para o formato interno de *mesh* do sistema.
- `glib.py`: Camada de abstração de baixo nível (Wrapper).
 - **Recursos OpenGL**: Classes para gerir a criação e destruição de *Shaders*, *Textures*, *Vertex Array Objects* (VAO) e *Uniform Buffers* (UBO).

2.3 Controlos:

Tabela 1: Mapeamento dos Controlos da Aplicação

Input	Ação
W / A / S / D	Movimentação da Câmara Livre (Frente, Esq, Trás, Dir)
Q / E	Ajustar Altura da Câmara (Q=Descer, E=Subir)
Setas Direcionais	Condução do Carro (Acelerar, Travar, Virar)
Rato	Orientação da Câmara (Olhar em volta)
Espaço	Alternar Modo de Câmara (Livre → Seguir → Condutor)
K / L	Abrir/Fehar Portas do Carro (K=Esq, L=Dir)
F	Abrir/Fehar Portão da Garagem
Z	Alternar Ecrã Cheio (Fullscreen)
ESC	Sair da Aplicação

3 Requisitos

Requisito Principal	Estado
Veículo com rodas traseiras maiores	✓
Portas do veículo abrem com interação	✓
Volante roda	✓
Rodas giram com o deslocamento	✓
Garagem com porta interativa	✓
Mínimo de 2 fontes de luz (Sol e Poste)	✓
Mínimo de 5 materiais diferentes	✓
Câmara controlada pelo utilizador	✓
Chão texturado por repetição	✓

Requisito Secundário	Estado
Elementos extra (Avião, Café, Árvores)	✓
Rotação diferencial das rodas (traseiras mais lentas)	✓
Aplicação de texturas (Estrada, Edifícios)	✓
Carro com direção (virar)	✓
Volante sincronizado com a viragem	✓
Câmara de seguimento (Follow Cam)	✓
Câmara interior (Inside Cam)	✓
Efeitos extra (Ciclo Dia/Noite e Skybox)	✓

Tabela 2: Requisitos

Nesta secção, detalhamos a abordagem técnica adotada para cada um dos requisitos propostos, fornecendo uma breve explicação da implementação e referenciando as secções deste relatório onde os algoritmos são aprofundados.

3.1 Requisitos Principais

Veículo com rodas traseiras maiores que as dianteiras

O veículo foi modelado respeitando uma estrutura hierárquica de nós (ver Secção 4.1). A diferença de tamanho das rodas é visível graficamente (ver Secção 5) e foi definida na estrutura lógica em `carro.py`, onde as rodas traseiras possuem um raio e largura superior.

Tabela 3: Configuração das dimensões das rodas

Roda	Raio	Largura	Eixo
Frente (FL/FR)	0.13	0.13	Dianteiro
Trás (RL/RR)	0.16	0.16	Traseiro

Portas do veículo interativas

As portas esquerda e direita são nós independentes no grafo de cena, filhos do carro. A abertura é controlada pelas teclas K e L, utilizando interpolação para suavizar o movimento entre 0° e 75° . (Ver detalhes da animação na Secção 4.4.1).

Volante rotativo

O volante faz parte do carro, e a sua rotação no eixo Z local está ligada diretamente à variável de estado `steer` do carro, garantindo que roda visualmente em sincronia com a direção das rodas dianteiras (ver detalhes da animação na secção 4.4.1).

Giro das rodas com o deslocamento

A rotação das rodas é calculada fisicamente com base na velocidade linear do carro (v) e no raio da roda (r), acumulando o ângulo de rotação a cada frame: $\theta_{nova} = \theta_{antiga} + (v/r) \cdot \Delta t$. Como as rodas traseiras têm raio maior ($r = 0.16$) que as dianteiras ($r = 0.13$), rodam mais lentamente para a mesma velocidade linear, cumprindo o requisito físico (ver detalhes em 4.4.1).

Garagem interativa

A garagem possui uma porta animada que responde à tecla F. A animação utiliza a mesma lógica de interpolação das portas do carro, mas realizando uma translação da porta no eixo Y (movimento vertical) em vez de rodar. O carro pode entrar fisicamente na garagem pois as dimensões foram ajustadas para tal (ver detalhes da animação em 4.4.1).

Fontes de iluminação

A cena possui duas fontes de luz distintas, os *nodes* possuem um atributo `is_light` que permite saber se o seu comportamento é de emissor, assim, a sua luz é coletada pelo `render.py` para que os *shaders* dos outros nodes calculem a sua influencia na cor, usando o modelo *Blinn-Phong* (ver detalhes sobre a luz em 4.2.2).

- **Luz Direcional (Sol):** Simula a luz solar, afetando todos os objetos com a mesma direção. A direção muda dinamicamente com a animação do ciclo dia/noite (ver secção 4.4.4).
- **Luz Pontual (Poste):** Localizada no poste de iluminação, emite luz que decai com a distância, iluminando a área circundante do parque.

Materiais diferentes

Foram criados e aplicados diversos materiais com propriedades especulares e difusas distintas para enriquecer a cena, todos os materiais podem ser vistos na tabela 4.

Controlo da câmara

O utilizador pode controlar livremente a câmara em modo "Free Cam" usando o rato para

olhar e o teclado para mover, permitindo explorar toda a cena sem restrições (ver secção 4.4.3).

Chão texturado por repetição

O chão (relva) é um plano grande onde as coordenadas de textura (UVs) foram multiplicadas por um fator de escala (ex: 50x), fazendo com que a textura pequena da relva se repita em mosaico em vez de ser esticada.

3.2 Requisitos Secundários

Elementos extra na cena

Além do carro e garagem, a cena foi enriquecida com:

- Um **avião de papel** animado que voa em círculos sobre a cena.
- Um **café** com esplanada (bancos e mesas).
- **Árvores** geradas proceduralmente em posições aleatórias.
- Um **poste de iluminação** funcional.

Todos os objetos são implementados respeitando a estrutura de *node*, com uma *mesh* e um *material*, a secção 4 detalha todas as transformações e características.

Rodas refletem o movimento (rotação diferencial)

Como explicado no requisito principal das rodas, o sistema físico implementado em `carro.py` calcula a velocidade angular $\omega = v/r$. Isto garante automaticamente que as rodas maiores rodam mais devagar que as pequenas.

Aplicação de texturas

Foram aplicadas texturas *diffuse* em vários elementos: relva, estrada, chão de terra e nas paredes do edifício (café). O sistema suporta carregamento de imagens JPG/PNG via biblioteca PIL, as texturas são passadas pelo objeto *material* de cada *node*, que verifica se existe textura e faz o *bind* em `material.py`.

Carro vira e volante controla viragem

O carro não se move apenas em linha reta. Foi implementado um modelo de condução onde o input de viragem altera o ângulo das rodas da frente e, consequentemente, a orientação (*yaw*) do carro ao longo do tempo. O volante visual acompanha este ângulo em tempo real. Isto é possível devido à classe `carro.py` que usa o método `update_car`, que calcula os atributos físicos do carro e da roda consoante a direção, em comunhão com o `anim.py` que usa dessa informação pra desenhar o carro (detalhes na secção 4.4.1)

Câmara de seguimento e interior

Implementámos dois modos de câmara adicionais (alternáveis com a tecla **Espaço**):

- **Follow Cam:** Segue o carro de trás com suavização (*lag*), criando uma sensação de movimento fluido.
- **Inside Cam:** Coloca a câmara na posição da cabeça do condutor, seguindo o movimento do carro.

Ambos os estilos de câmara são feitos usando a função `make_follow_cam` (ver detalhes de implementação na secção 4.4.3), porém, diferem no *offset* e no *lag*, dado que o comportamento é igual em ambas.

4 Descrição do Grafo de Cena

4.1 Grafo de Cena

A estrutura da cena foi organizada num Grafo de Cena hierárquico, onde as transformações geométricas (translação, rotação, escala) são propagadas dos nós pais para os nós filhos. A raiz da cena (*Root*) contém todos os objetos estáticos e dinâmicos.

- **Root** (Nó Raiz)
 - **Floor** (Chão com relva)
 - **RoadX, RoadZ** (Estradas cruzadas)
 - **Ground1, Ground2** (Bases de betão)
 - **Car** (Nó principal do carro - controla a posição global)
 - * **CarBody** (Corpo - controla a rotação/escala do modelo)
 - **Door_L, Door_R** (Portas com animação de abertura)
 - **SteeringWheel** (Volante com rotação sincronizada)
 - **Wheel_FL, Wheel_FR** (Rodas dianteiras (menores) - direção e rotação)
 - **Wheel_RL, Wheel_RR** (Rodas traseiras (maiores) - rotação)
 - **Pole** (Poste de iluminação)
 - * **LightPole** (Fonte de luz pontual)
 - **Aviao** (Objeto animado em rota circular)
 - **Sun** (Objeto Sol com luz pontual e animação orbital)
 - **GO** (Garagem - Estrutura)
 - * **GD** (Portão da garagem com animação vertical)
 - **Cafe** (Edifício estilo "café")
 - **ParkElements** (Grupo de bancos e pedras)
 - **Tree** (Árvores geradas aleatoriamente)

Esta hierarquia permite, por exemplo, que ao mover o nó *Car*, todas as suas componentes (rodas, portas, volante) se movam solidariamente, enquanto as animações locais (como abrir a porta) ocorrem no espaço local do nó filho. As relações entre nós simplificam as transformações e a percepção da cena como um todo, ilustrando tanto dependências como relações todo->parte.

4.2 Processamento e Renderização

O programa corre num ciclo gerido no ficheiro `main.py`. Este ciclo é responsável por manter a sincronização entre a lógica da aplicação e a apresentação visual, permitindo que todos os objetos sejam desenhados de acordo com as transformações sofridas ao longo da cena.

4.2.1 Ciclo de Atualização Lógica (Update)

A primeira fase de cada *frame* é a atualização lógica. O motor calcula o tempo decorrido desde o último `frame` (`deltaTime`).

O método `root.update(dt)` é invocado na raiz do grafo de cena e propaga-se recursivamente a todos os nós filhos. Em cada nó, o sistema verifica a existência de um componente `animator`.

- **Animators:** São funções em Python (definidas em `anim.py`) que encapsulam a lógica de comportamento. Por exemplo, o *animator* do Sol calcula a sua nova posição orbital baseada no tempo acumulado, atualizando a matriz de transformação local (`self.local`) do nó.

- **Hierarquia:** Como a atualização ocorre antes da renderização, garantimos que quando o desenho começar, todas as matrizes locais refletem o estado mais atual da simulação física e das interações do utilizador.

4.2.2 Pipeline de Renderização (Render)

A renderização é gerida pela classe `Renderer` (`renderer.py`), que abstrai as chamadas de baixo nível do OpenGL e simplifica imenso o `main.py`, assim respeitando um sistema de atribuição de responsabilidades que permite com que o código seja facilmente manipulável. O processo de desenho de um *frame* envolve os seguintes passos:

1. **Recolha de Luzes:** Antes de desenhar a geometria, o *Renderer* percorre o grafo para identificar nós marcados como fontes de luz (`is_light = True`). As suas posições globais são calculadas e armazenadas numa lista de luzes pontuais (*Point Lights*) que será enviada para os *shaders*.
2. **Travessia do Grafo:** O método `render()` percorre o grafo, e, para cada nó, o sistema mantém uma pilha de matrizes de transformação. A matriz global do nó (M_{global}) é calculada multiplicando a matriz global do pai (M_{parent}) pela matriz local do nó (M_{local}):

$$M_{global} = M_{parent} \times M_{local}$$

Esta operação é fundamental para garantir que, por exemplo, as rodas do carro acompanham o movimento do corpo.

3. **Gestão de Estado e Shaders:** O *Renderer* verifica se o *shader* necessário para o material atual já está ativo; se não estiver, ativa-o e envia as variáveis uniformes globais (*Uniforms*) comuns, como as matrizes de Projeção (P) e Vista (V), a direção da luz solar e a lista de luzes pontuais recolhida anteriormente.
4. **Desenho da Mesh:** Cada *node* tem uma *mesh* e um material, e, para cada, o método `material.bind()` é invocado. Este método carrega no GPU as propriedades específicas da superfície (cor difusa, textura, coeficiente especular) e a matriz *Model* (M_{global}) do objeto. Finalmente, a geometria é desenhada através de uma chamada `glDrawElements`, utilizando os índices dos vértices.

4.3 Detalhe da Implementação dos Nós e Transformações

- **Ambiente e Estradas (Estático)**

- **Floor:** Representa o chão de relva. Foi aplicada uma escala de $(500, 0.1, 500)$ para cobrir todo o horizonte visível.
- **RoadX e RoadZ:** Representam as estradas. O nó `RoadZ` é uma cópia do `RoadX` mas rodado 90° sobre o eixo Y, criando o cruzamento central. Ambos estão ligeiramente elevados ($y = 0.02$) para evitar *z-fighting* com o chão, ambos os *nodes* com escala de $(500, 1, 12)$.

- **Veículo (Hierarquia Complexa)**

- **Car (Nó Pai):** É um nó "invisível" (sem *mesh*) que controla a posição global do carro no mundo (x, z). Todas as físicas de movimento são aplicadas a este nó.
- **CarBody:** Filho do nó `Car`. Contém a *mesh* do corpo.

* **Transformação:** Foi necessário aplicar uma escala de 10.0 e uma rotação de 90° para alinhar o modelo OBJ com o eixo de movimento da cena.

- * **Filhos:** Este nó serve de pai para as rodas, portas e volante, garantindo que todos acompanham o corpo.
- **Door_L e Door_R:** As portas são posicionadas nas laterais do carro. A sua origem foi ajustada para funcionar como dobradiça, permitindo que a rotação no eixo Y simule a abertura da porta.
- **Wheels:** As quatro rodas são instanciadas como filhas do corpo. As rodas dianteiras possuem lógica adicional para rodar no eixo Y (direção), enquanto todas rodam no eixo X (movimento).

- **Garagem (Estrutura Interativa)**

- **Garagem (GO):** Uma estrutura posicionada em (30, 2.3, 30) com uma escala de 15.0. Foi rodada 270° para ficar virada para a estrada.
- **Portão (GD):** É filho da Garagem. Isto simplifica a animação, pois o movimento de abrir/fechar é apenas uma translação vertical local (y), sem necessidade de saber onde a garagem está no mundo.

- **Elementos Dinâmicos (Animação Autónoma)**

- **Sol:** O node sol usa uma *sphere mesh* e uma animação orbital que simula o ciclo dia/noite. A sua posição define também a direção da luz global.
- **Avião:** Um nó controlado por um *animator* específico que descreve uma trajetória circular de raio 20.0 a uma altura de 20.0, com uma escala de 5.0.

- **Elementos Decorativos e Estáticos**

- **Café:** Um edifício decorativo posicionado no quadrante oposto à garagem, com escala 1.5.
- **Poste de Luz:** Composto pelo poste (base) e um nó filho **LightPole** com uma *sphere mesh* que representa a lâmpada. O nó filho é marcado como emissor de luz (`is_light=True`).
- **ParkElements:** Um nó de agrupamento posicionado em (30, 0, -30). Serve de container para os bancos (compostos por cubos) e pedras (esferas deformadas) gerados proceduralmente na função `add_park`.

4.4 Descrição Detalhada das Animações

As animações na cena não são pré-gravadas (como em formatos FBX), mas sim calculadas proceduralmente em tempo real a cada *frame*. O sistema utiliza o tempo decorrido (`deltaTime`) para atualizar as matrizes de transformação local dos nós, garantindo que a velocidade das animações é independente da taxa de quadros (*framerate*) do computador.

As animações foram categorizadas em três tipos de implementação:

4.4.1 Cinemática e Hierarquia do Veículo

A animação do veículo é construída através de uma cadeia de transformações matriciais hierárquicas. A implementação divide-se em três componentes principais:

1. Matriz Global do carro (M_{carro}) Controla a posição e orientação do veículo no mundo.

$$M_{carro} = T(P_x, 0, P_z) \cdot R_y(\psi) \quad (1)$$

- P_x, P_z : Posição global atualizada pela integração da velocidade.
- ψ : Ângulo de orientação (*Yaw*) do veículo.

2. Matriz Local das Rodas (M_{roda}) Cada roda é transformada relativamente ao carro. A ordem de multiplicação é crítica para combinar rotação, direção e posicionamento:

$$M_{roda} = T_{offset} \cdot R_{steer}(\delta) \cdot R_{roll}(\phi) \cdot S_{dim} \quad (2)$$

Componentes da Transformação:

- **Escala (S_{dim}):** Ajusta o cilindro unitário às dimensões reais.

$$S_{dim} = S(R_{roda}, R_{roda}, W_{roda})$$

- **Rolamento (R_{roll}):** Rotação no eixo X baseada na distância percorrida ($\Delta d = v \cdot \Delta t$).

$$\phi_{nova} = \phi_{antiga} + \frac{v \cdot \Delta t}{R_{roda}}$$

- **Direção (R_{steer}):** Rotação no eixo Y (apenas rodas dianteiras).

$$\delta = \begin{cases} \text{ângulo de viragem,} & \text{se dianteira} \\ 0, & \text{se traseira} \end{cases}$$

- **Posição (T_{offset}):** Translação fixa para a posição da roda no carro (ex: frente-esquerda).

3. Acoplamento do Volante O volante roda no seu eixo local Z, amplificando o movimento das rodas para simular a caixa de direção.

$$\theta_{volante} = -k \cdot \delta \quad (3)$$

- $k = 6.0$: Constante que define relação da rotação da volante/roda.
- O sinal negativo corrige a orientação visual do objeto na cena.

4. Animação das Portas As portas utilizam uma lógica de interpolação dependente da entrada contínua do utilizador (teclas 'k' e 'l'), em vez de um estado persistente.

- **Lógica de Controlo:** O ângulo alvo θ_{target} é definido diretamente pelo estado da tecla:

$$\theta_{target} = \begin{cases} 70^\circ & \text{enquanto a tecla estiver pressionada} \\ 0^\circ & \text{caso contrário} \end{cases} \quad (4)$$

- **Interpolação:** O ângulo atual θ_{curr} é atualizado frame a frame usando suavização para garantir movimento fluido:

$$\theta_{curr} = \theta_{curr} + (\theta_{target} - \theta_{curr}) \cdot (1 - e^{-\text{speed} \cdot \Delta t}) \quad (5)$$

- **Transformação:** A matriz local da porta é reconstruída preservando a posição T_{base} e escala S_{base} , aplicando apenas a rotação no eixo Y:

$$M_{porta} = T_{base} \cdot R_y(\theta_{curr}) \cdot S_{base} \quad (6)$$

4.4.2 Mecanismo da Garagem

A animação do portão da garagem funciona como um *toggle* com interpolação, realizando um movimento vertical.

- **Estado Lógico:** Uma variável booleana *is_open* alterna o seu valor cada vez que a tecla de interação ('F') é pressionada (detetada na borda de subida).
- **Definição do Alvo:** Com base no estado lógico, define-se a posição vertical alvo *ytarget*:

$$y_{target} = \begin{cases} y_{base} + offset_{open} & \text{se } is_open = \text{True} \\ y_{base} & \text{se } is_open = \text{False} \end{cases} \quad (7)$$

- **Interpolação:** A posição atual *ycurr* aproxima-se do alvo, desacelerando à medida que chega ao fim, assim como as portas do carro.

$$y_{curr} \leftarrow y_{curr} + (y_{target} - y_{curr}) \cdot (1 - e^{-\text{speed} \cdot \Delta t}) \quad (8)$$

- **Reconstrução:** Por fim, a matriz é reconstruída com o novo valor de *ycurr*.

4.4.3 Follow Camera

A câmara opera no espaço global, calculando a sua posição frame a frame baseada na matriz do carro.

- **Definição do Alvo Ideal:** Primeiro, calcula-se onde a câmara *deveria* estar se estivesse rigidamente presa ao carro. Dada a matriz do carro M_{carro} , extraímos a sua posição P_{carro} e a sub-matriz de rotação R_{carro} . Aplicamos um deslocamento local δ_{offset} (ex: atrás e acima):

$$P_{ideal} = P_{carro} + R_{carro} \cdot \delta_{offset} \quad (9)$$

Desta forma é possível escolher entre estar atrás do carro ou dentro do carro.

- **Ponto de Foco (Look-At):** Para garantir que o *user* vê o que está à frente, o ponto para onde a câmara olha não é o centro do carro, mas sim um ponto projetado à frente do veículo. Usando o vetor "frente" do carro (\vec{F}):

$$P_{foco} = P_{carro} + \vec{F} \cdot d_{lookahead} \quad (10)$$

- **Interpolação:** A posição real da câmara (P_{eye}) não salta instantaneamente para P_{ideal} . Utiliza-se uma interpolação dependente do tempo (Δt) e de um fator de atraso (λ):

$$\alpha = 1.0 - e^{-\Delta t / \lambda} \quad (11)$$

$$P_{eye}(t) = P_{eye}(t-1) + (P_{ideal} - P_{eye}(t-1)) \cdot \alpha \quad (12)$$

4.4.4 Ciclo Solar (Dia/Noite)

A animação do Sol simula o ciclo dia/noite através de uma órbita circular em torno do eixo Z. Isto faz com que o Sol nasça e se ponha, alterando a sua altura (Y) e posição horizontal (X).

- **Cálculo da Órbita:** O ângulo orbital θ é derivado do tempo acumulado t e do período da órbita T :

$$\theta = \frac{t}{T} \cdot 2\pi$$

- **Posição:** A posição do Sol é definida pela aplicação de uma matriz de rotação R_z a um vetor base, seguida de translação. Sendo r o raio da órbita:

$$P_{sol} = R_z(\theta) \cdot \begin{bmatrix} r \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad (13)$$

Esta transformação resulta num movimento circular no plano XY vertical. A matriz final do nó combina esta posição com uma escala fixa para a esfera visual do Sol.

4.4.5 Cinemática do Avião:

O movimento do avião é mais complexo, pois exige que o objeto não só descreva uma trajetória circular, mas que também se oriente fisicamente na direção do movimento (o "nariz" deve apontar para a frente) e realize uma rotação sobre o seu próprio eixo (spin).

- **Cálculo da Posição (P):** A posição num instante t é calculada rodando um ponto inicial $(r, 0, 0)$ em torno de um eixo arbitrário \vec{A} (definido na cena):

$$P(t) = \text{centro} + R_{\vec{A}}(\theta) \cdot P_{base} + \text{altura}$$

- **Vetor Tangente (\vec{T}):** Para orientar o avião, calculamos a derivada numérica da posição (vetor velocidade):

$$\vec{T} = -\text{normalize}(P(t + \epsilon) - P(t)) \quad (14)$$

Nota: O vetor é invertido para corresponder à orientação do modelo 3D (frente do avião).

- **Construção do Referencial (Frame):** Para construir a matriz de orientação, usamos os vetores *Right* (\vec{R}) e *Up* (\vec{U}) usando produtos vetoriais, garantindo ortogonalidade:

$$\vec{R} = \text{normalize}(\vec{A} \times \vec{T})$$

$$\vec{U}_{corrigido} = \vec{T} \times \vec{R}$$

- **Matriz de Orientação:** Estes três vetores formam as colunas da matriz de rotação que alinha o avião com a trajetória:

$$M_{orient} = \begin{bmatrix} T_x & U_x & R_x & 0 \\ T_y & U_y & R_y & 0 \\ T_z & U_z & R_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **Transformação Final:** A matriz final combina todas as transformações, incluindo um *spin* local (rotação no eixo X do avião) para simular uma acrobacia.

$$M_{aviao} = T_{pos} \cdot M_{orient} \cdot R_{local_x}(\phi_{spin}) \cdot S \quad (15)$$

4.5 Relação Nós e Materiais

A tabela seguinte detalha as propriedades dos materiais atribuídos aos nós, especificando a componente difusa (Cor ou Textura), a componente especular (K_s) e o coeficiente de brilho (*Shininess*).

Tabela 4: Propriedades Físicas dos Materiais (Modelo Blinn-Phong)

Nó	Material	Difusa (Cor/Textura)	Especular (K_s)	Brilho (s)
Floor	Relva (floor)	Textura: Relva	(0.0, 0.0, 0.0)	-
RoadX, RoadZ	Asfalto (road)	Textura: Asfalto	(0.0, 0.0, 0.0)	-
CarBody	Metal (car)	RGB (0.8, 0.1, 0.1)	(0.8, 0.8, 0.8)	100.0
Wheels	Borracha (wheel)	RGB (0.1, 0.1, 0.1)	(0.3, 0.3, 0.3)	10.0
Sun	Luz (sun)	RGB (1.0, 0.95, 0.8)	N/A (Emissive)	-
LightPole	Luz (light_pole)	RGB (1.0, 1.0, 0.9)	N/A (Emissive)	-
Cafe	Estuque (cafe_textura)	RGB (0.7, 0.5, 0.3)	(0.1, 0.1, 0.1)	10.0
Tree (Tronco)	Tronco (bark)	RGB (0.4, 0.25, 0.1)	(0.0, 0.0, 0.0)	5.0
Tree (Folhas)	Folha (bf_wood)	RGB (0.1, 0.4, 0.1)	(0.0, 0.0, 0.0)	1.0
Garagem (GO)	Betão (go)	RGB (0.2, 0.2, 0.2)	(0.0, 0.0, 0.0)	3.0
Portão (GD)	Metal (gd)	RGB (0.5, 0.5, 0.6)	(0.2, 0.2, 0.2)	32.0
Avião	Papel (aviao)	RGB (0.9, 0.9, 0.9)	(0.5, 0.5, 0.5)	50.0

5 Screenshots Comentados

Nesta secção, apresentamos *screenshots* dos principais *nodes* da cena e demonstrações das funcionalidades interativas implementadas. Na pasta onde consta o trabalho (link em 6), foi deixado um vídeo completo, que demonstra todas as animações, objetos, texturas, influência da luz e interações do utilizador. Recomenda-se assistir o vídeo dado que *screenshots* não demonstram totalmente a natureza da cena.

5.1 Nodes Principais da Cena

Os seguintes nodes representam os elementos visuais fundamentais do projeto:



Figura 1: Node do carro: Representação do veículo na cena com hierarquia completa (corpo, portas, rodas, volante).

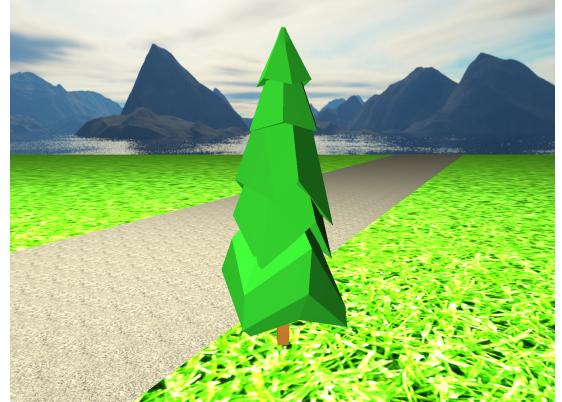


Figura 2: Node da árvore: Representação de uma árvore com sub-nodes para tronco (bark) e folhas (bf_wood).

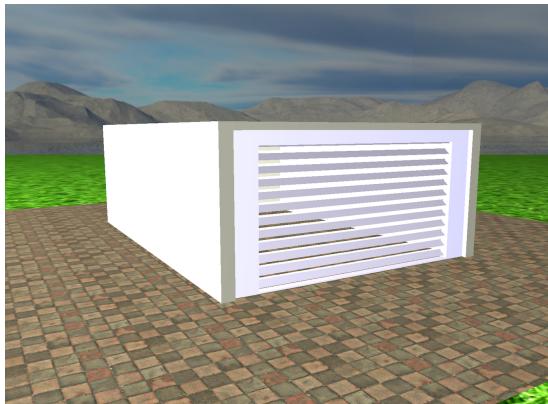


Figura 3: Node da garagem (GO): Estrutura da garagem com porta animável (GD).



Figura 4: Nodes da estrada (RoadX e RoadZ): Representação das estradas perpendiculares com textura repetida.



Figura 5: Node da relva (Floor): Representação do chão coberto de relva (500×500 unidades).



Figura 6: Node do café: Edifício carregado de ficheiro OBJ com escala ajustada.

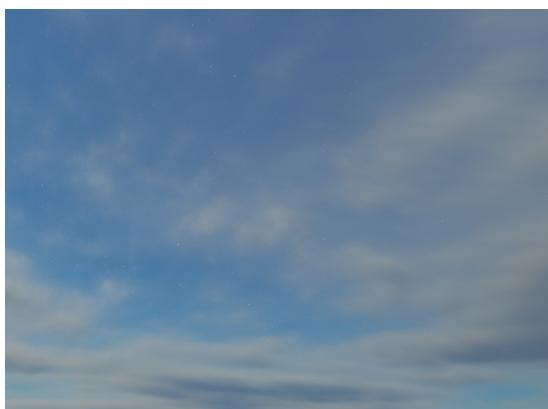


Figura 7: Skybox: Representação do céu no ambiente usando cubemap (fantasy_sunless).

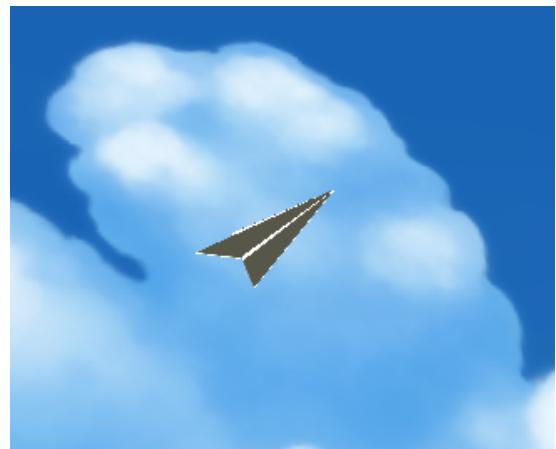


Figura 8: Node do avião: Objeto animado com voo orbital automático.

5.2 Direção das Rodas

As rodas dianteiras do carro viram para esquerda e direita durante a condução, simulando o sistema de direção do veículo. Esta rotação é visível e sincronizada com a orientação do carro.



Figura 9: Rodas em posição neutra (0° , carro em linha reta).



Figura 10: Rodas viradas para a esquerda (ângulo positivo, curva à esquerda).



Figura 11: Rodas viradas para a direita (ângulo negativo, curva à direita).

5.3 Animação de Portas do Carro (K/L)

As portas do carro abrem e fecham suavemente usando animadores (`make_door_anim`). Tecla K controla a porta esquerda (Door_L), tecla L controla a porta direita (Door_R).



Figura 12: Portas do carro fechadas (estado inicial).



Figura 13: Porta esquerda aberta (tecla K, rotação $+75^\circ$).



Figura 14: Porta direita aberta (tecla L, rotação -75°).



Figura 15: Ambas as portas abertas simultaneamente (K+L).

5.4 Animação da Porta da Garagem (F)

A porta da garagem (node GD) sobe e desce com a tecla F usando `make_garage_door_animator`.

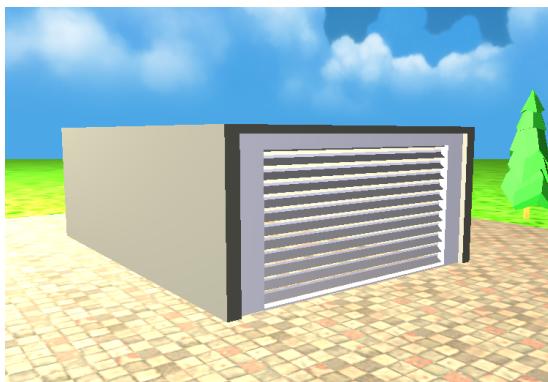


Figura 16: Porta da garagem fechada (estado inicial).

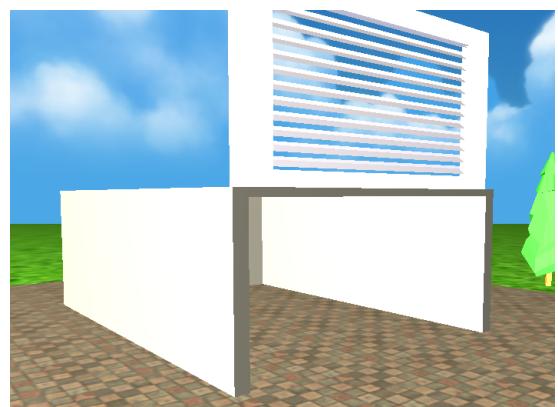


Figura 17: Porta da garagem aberta (tecla F, offset Y=-0.4).

5.5 Alternância de Modos de Câmera (SPACE)

A tecla **SPACE** alterna entre três modos: **Free** (livre com WASD+mouse), **Follow** (segue carro), **Inside** (vista interna/próxima).



Figura 18: Modo **Free Camera**: controlo manual WASD + rotação do mouse.



Figura 19: Modo **Follow Camera**: câmera segue carro com offset atrás (smoothing lag=0.18s).



Figura 20: Modo **Inside Camera**: vista próxima/interna do carro.

5.6 Objetos com Animação Automática

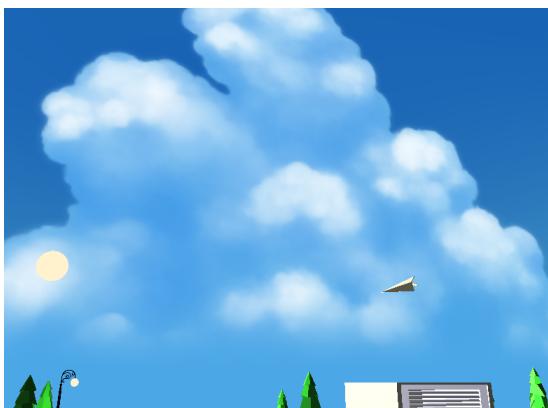


Figura 21: Avião em voo orbital: posição 1.

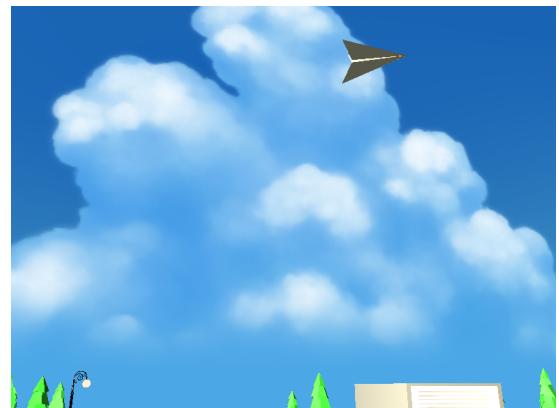


Figura 22: Avião em voo orbital: posição 2.



Figura 23: Sol em órbita: posição inicial.



Figura 24: Sol em órbita: posição avançada.

5.7 Skybox e Variações de Ambiente

O projeto suporta múltiplas skyboxes. A seleção é feita alterando a variável `SKYBOX_COLLECTION` em `main.py`.



(a) Montanhas



(b) Dia



(c) Noite

Figura 25: Comparação das diferentes Skyboxes implementadas na cena.

5.8 Influência do Node sol como fonte de luz

O Sol é a fonte de luz mais forte da cena, é possível ver a sua influência consoante a sua órbita e ver a influência do poste de luz na ausência do sol

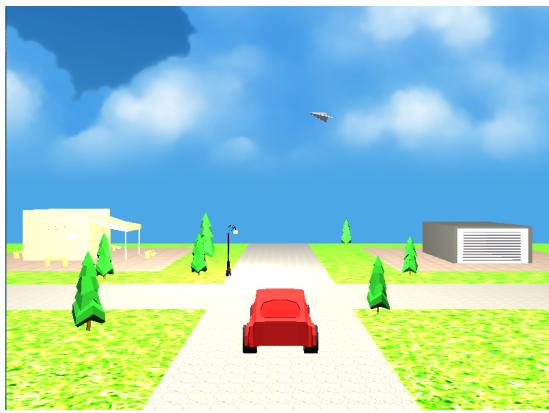


Figura 26: Sol Meio-dia: Sol ao meio dia.



Figura 27: Sol-Afternoon: Ambiente Tarde.



Figura 28: Sol-noite: Ambiente noturno

Nota: A intensidade do Sol foi aumentada para demonstrar a influência na cena, as imagens podem diferir do produto final.

5.9 Cena Completa

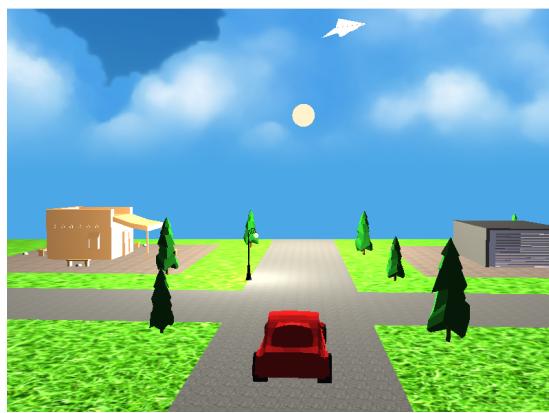


Figura 29: Visão Total da Cena.

6 Link para o Código Fonte e para o vídeo do Youtube

O código fonte completo pode ser consultado no seguinte repositório:

https://github.com/leoPardall/projetoCG_grupo02

Como o vídeo tem 1GB de espaço, aqui está o link do youtube:

<https://www.youtube.com/watch?v=cogYM4YylCk>