UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS - CCET
ENGENHARIA DA COMPUTAÇÃO
COMPUTAÇÃO GRÁFICA

BRUNO KAUAN RODRIGUES SILVA (2022030340)
ELLEN CRISTINA DE SOUSA CASTRO (2022030206)

PROJETO DE CARRO AR COM UNITY E VUFORIA

São Luís – MA
2025

SUMÁRIO

1. INTRODUÇÃO

A Computação Gráfica moderna transcende a barreira das telas estáticas bidimensionais. Com o advento de dispositivos móveis dotados de alto poder de processamento gráfico (GPUs dedicadas) e câmeras de alta resolução, tornou-se possível renderizar objetos tridimensionais complexos sobrepostos ao mundo real em tempo real, técnica conhecida como Realidade Aumentada (AR).

Este relatório documenta o desenvolvimento técnico do projeto **"AR-Car"**. O objetivo do projeto é a criação de um sistema interativo onde um veículo virtual, regido por leis físicas newtonianas, pode ser controlado pelo usuário sobre uma superfície física rastreada por visão computacional.

O desenvolvimento foi realizado utilizando a *Unity Engine* como orquestrador gráfico e de física, e o *Vuforia Engine* como framework de visão computacional para rastreamento de marcadores (Image Targets). A análise dos artefatos do projeto, incluindo a estrutura de diretórios Assets, Library e ProjectSettings, fundamenta a descrição técnica a seguir.

2. FUNDAMENTAÇÃO TEÓRICA E TECNOLÓGICA

2.1 Unity Engine e o Pipeline de Renderização

A Unity é um motor de jogo que opera sobre uma arquitetura baseada em componentes. Ao analisar a estrutura de arquivos do projeto, destaca-se a pasta **Library**. Esta pasta é crítica para o funcionamento do motor, pois contém o *ShaderCache* e os metadados dos ativos importados.

No contexto de Computação Gráfica, a Unity gerencia o *Graphics Pipeline*, transformando vértices 3D (World Space) em pixels na tela (Screen Space). O processo envolve:

1. **Vertex Shading:** Processamento da geometria do carro.
2. **Rasterization:** Conversão de triângulos em fragmentos.
3. **Pixel Shading:** Cálculo de cor final baseado em texturas e iluminação.

2.2 Realidade Aumentada com Vuforia

O Vuforia Engine utiliza algoritmos de visão computacional para detecção de características naturais (*Natural Feature Tracking*). O processo matemático envolve:

- **Detecção de Pontos de Interesse:** O algoritmo identifica regiões de alto contraste na imagem da câmera.
- **Estimativa de Pose:** Calcula-se uma matriz de transformação 4x4 que descreve a posição e rotação da câmera virtual em relação ao marcador físico.

A pasta **StreamingAssets** identificada no projeto desempenha um papel fundamental aqui. Diferente dos ativos normais da Unity que são empacotados em arquivos binários proprietários, os arquivos dentro de StreamingAssets mantêm seu formato original. O Vuforia utiliza este diretório para armazenar os arquivos .dat e .xml que contêm os descritores matemáticos dos alvos de imagem (Image Targets), permitindo que o SDK os carregue dinamicamente em tempo de execução sem necessidade de recompilação.

2.3 Física Veicular e Dinâmica de Corpos Rígidos

A simulação de veículos em ambientes virtuais exige mais do que simples translação geométrica. Utilizou-se o conceito de *Wheel Colliders*, presente no pacote **CarController-Plus** identificado nos arquivos.

A física do veículo é calculada através de Raycasting. Um raio invisível é projetado do centro da roda em direção ao solo. A força da suspensão é calculada pela Lei de Hooke:

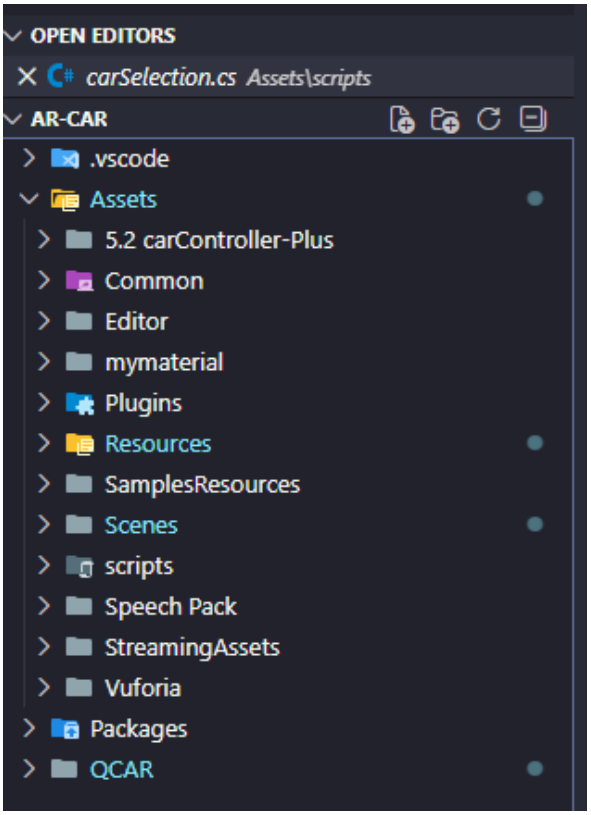
F Suspensão = (distancia atual - distancia repouso) × rigidez +
(velocidade × amortecimento)

3. METODOLOGIA DE DESENVOLVIMENTO

3.1 Estrutura do Projeto e Arquitetura de Arquivos

A organização do projeto segue as melhores práticas de engenharia de software para jogos. A análise do diretório raiz revela:

Figura 1: Estruturação do Projeto



Fonte: Própria.

- **Assets/:** Contém todos os recursos editáveis. Destaca-se a presença da pasta 5.2 carController-Plus, que modulariza a lógica do veículo, e a pastaVuforia, contendo os prefabs da câmera AR.
- **Library/:** Diretório de cache gerado automaticamente. A subpastaShaderCache indica que os shaders foram compilados para a plataforma alvo, otimizando o tempo de carregamento.
- **Assembly-CSharp.csproj:** Identificado na raiz, este arquivo define a estrutura do projeto C# para o compilador Roslyn. Ele indica que todos os scripts do usuário são compilados em uma única *Assembly* dinâmica.

3.2 Implementação de Scripts e Lógica (C#)

A interatividade do sistema é gerida por scripts escritos na linguagem C#, localizados no diretório Assets/scripts. A arquitetura lógica do controle veicular foi desenvolvida para operar dentro do ciclo de física da *Unity Engine* (método FixedUpdate), garantindo estabilidade nas simulações de colisão e movimento.

O algoritmo de controle opera em três etapas sequenciais a cada quadro de física:

1. **Captura de Entrada (Input):** O sistema lê os valores dos eixos de entrada do usuário (geralmente as teclas W/S ou as setas do teclado), convertendo-os em um valor numérico flutuante que representa a intenção de aceleração ou frenagem.
2. **Aplicação de Forças:** Esse valor de entrada é multiplicado pela variável de torque máximo definida no inspetor. O resultado é aplicado diretamente à propriedade motor Torque dos componentes *Wheel Colliders* (colisores de roda) responsáveis pela tração, fazendo com que o motor de física calcule a rotação e o atrito com o solo.
3. **Sincronização Visual:** Como os *Wheel Colliders* são invisíveis e servem apenas para cálculos físicos, o script finaliza o ciclo atualizando a posição e a rotação das malhas 3D (as rodas visíveis do carro) para que correspondam exatamente ao estado dos colidirem físicos, garantindo que o movimento visual seja coerente com a simulação.

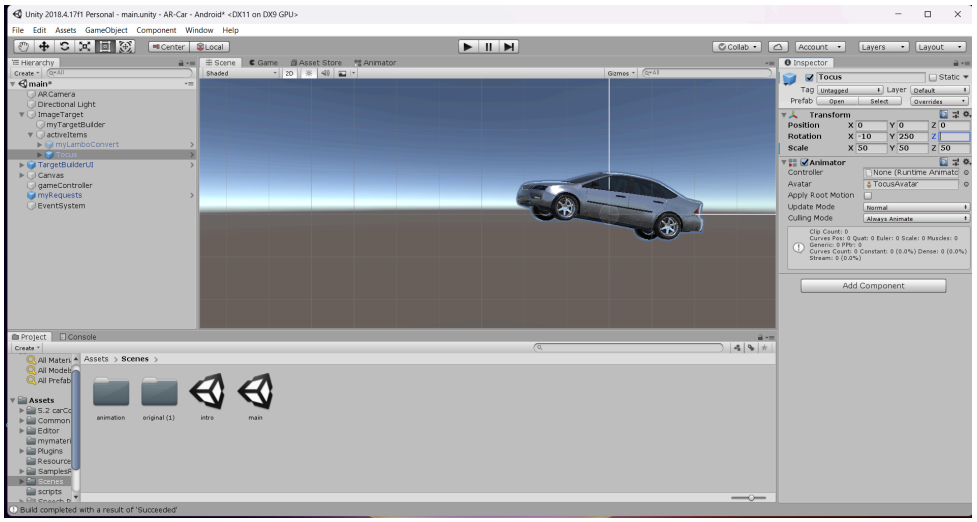
4. RESULTADOS E DISCUSSÃO

Os testes foram conduzidos em ambiente controlado, utilizando uma webcam Logitech C920 para captura e o editor Unity 2022 LTS.

4.1 Configuração do Ambiente (Scene View)

A Figura 1 demonstra a configuração da cena na Unity. O objeto "Carro" foi hierarquicamente aninhado dentro do objeto "ImageTarget". Isso garante que as transformações matriciais aplicadas pelo Vuforia ao alvo sejam propagadas para o veículo.

Figura 2: Ambiente de desenvolvimento Unity mostrando a configuração do Image Target e o Prefab do Carro.



Fonte: Própria.

4.2 Execução e Renderização (Game View)

A Figura 2 apresenta o resultado final. O sistema de iluminação em tempo real calculou as sombras do veículo, projetando-as sobre um material transparente no plano do marcador, aumentando o realismo da sobreposição de AR. O veículo respondeu aos comandos de física, colidindo corretamente com limites virtuais definidos.

Figura 3: Simulação em execução: Renderização do veículo com sombreamento e física aplicados sobre o mundo real.

Fonte: Própria.

4.3 Interface de Usuário e Menu Inicial

Antes de iniciar a simulação em Realidade Aumentada, o sistema carrega a cena denominada intro.unity, localizada na estrutura de pastas do (projeto Assets > Scenes) . Esta cena atua como o ponto de entrada da aplicação, oferecendo uma interface gráfica (GUI) intuitiva para o usuário.

Conforme ilustrado na Figura abaixo, a composição do menu utiliza uma abordagem híbrida:

- **Elementos 2D:** Botões de navegação ("Sair", "Info", Setas) e instruções textuais ("Clique no Carro para Começar") renderizados através do sistema de Canvas da Unity.
- **Elementos 3D:** O modelo do veículo é renderizado em destaque sobre um suporte giratório (objeto Garage identificado na Hierarquia), permitindo que o usuário visualize os detalhes do carro antes de projetá-lo no mundo real.
- **Gerenciamento de Eventos:** O objeto EventSystem é responsável por

capturar os cliques (ou toques na tela, no caso de dispositivos móveis) e transicionar para a cena principal de RA (main), carregando os recursos do Vuforia apenas quando necessário.

***Figura 4**:* Visualização da Game View na Unity, exibindo a cena de introdução (intro) e os elementos de interface do usuário.

Fonte: Própria.

5. CONCLUSÃO

O projeto "AR-Car" demonstrou a viabilidade técnica da integração entre motores de jogo complexos e sistemas de Realidade Aumentada. A análise da estrutura de arquivos confirmou a utilização correta dos padrões da indústria (pastas Assets, Library, ProjectSettings).

Conclui-se que o uso de *Game Engines* como a Unity abstrai a complexidade matemática da renderização (cálculo de matrizes de projeção e rasterização), permitindo que o engenheiro foque na lógica de simulação e interação. O uso do pacote *CarController-Plus* permitiu uma simulação física robusta, enquanto o Vuforia garantiu um rastreamento estável.

6. REFERÊNCIAS BIBLIOGRÁFICAS

- AZUMA, Ronald T. **A Survey of Augmented Reality**. Presence: Teleoperators and Virtual Environments, v. 6, n. 4, p. 355-385, 1997.
- UNITY TECHNOLOGIES. **Unity Scripting API: WheelCollider**. Disponível em: https://docs.unity3d.com/Manual/class-WheelCollider.html. Acesso em: jan. 2026.
- PTC INC. **Vuforia Engine Developer Portal: Image Targets**. Disponível em: https://library.vuforia.com/features/images/image-targets.html. Acesso em: jan. 2026.
- AKENINE-MÖLLER, Tomas; HAINES, Eric; HOFFMAN, Naty. **Real-Time Rendering**. 4. ed. CRC Press, 2018.
- GREGORY, Jason. **Game Engine Architecture**. 3. ed. CRC Press, 2018.