Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Proposta de Trabalho de Formatura Supervisionado
MAC0499

Renderização em Tempo Real Utilizando Ray Marching e sua Aplicação em Jogos Digitais

Pedro Tonini Rosenberg Schneider

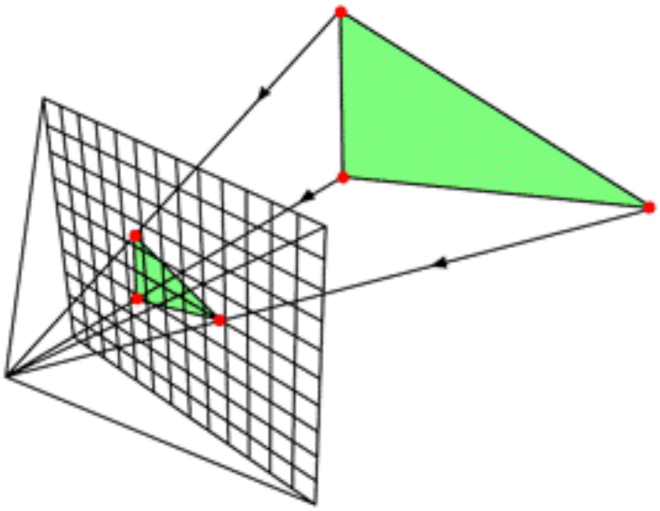
Orientador: Prof. Dr. Márcio Lobo Netto

São Paulo
Abril de 2024

1 Introdução

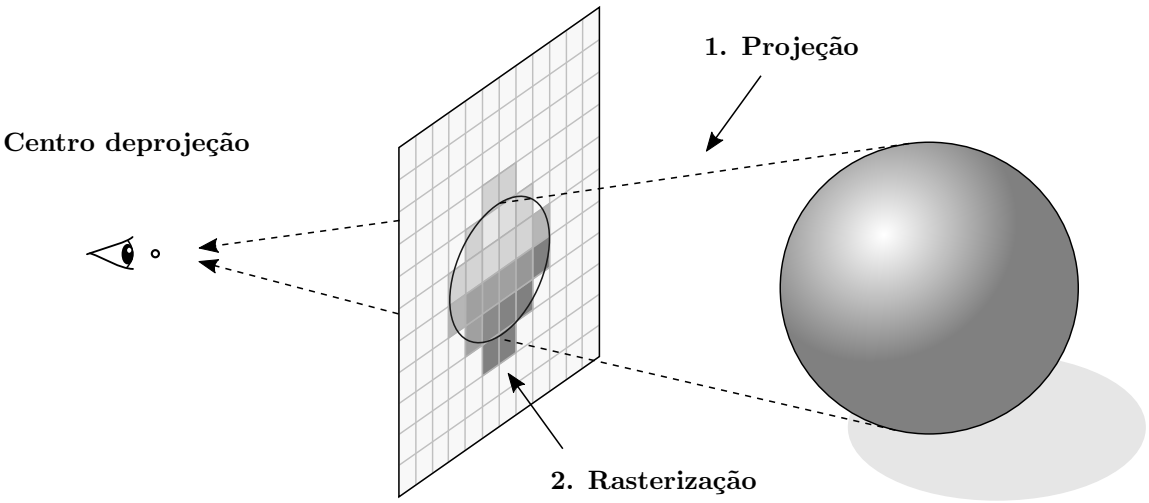
A técnica mais utilizada atualmente na renderização de aplicações de tempo real (incluindo jogos digitais) é a Rasterização. A Rasterização consiste, em suma, na projeção da geometria tridimensional em um plano bidimensional que formará a imagem final. Essa estratégia de renderização, embora extremamente eficiente como resultado de décadas de otimização, é, muitas vezes tomada como única, e acaba por impor restrições proibitivas no processo criativo em algumas aplicações. Alguns exemplos de efeitos computacionalmente complexos de serem implementados com a Rasterização seriam luzes, sombras, reflexões e refrações.

Figura 1: Ilustração da renderização de um triângulo por Rasterização



Fonte: Alexandre Ziebert, NVIDIA, 2020. Disponível:
`https://www.nvidia.com/pt-br/drivers/prbr-05282018/`

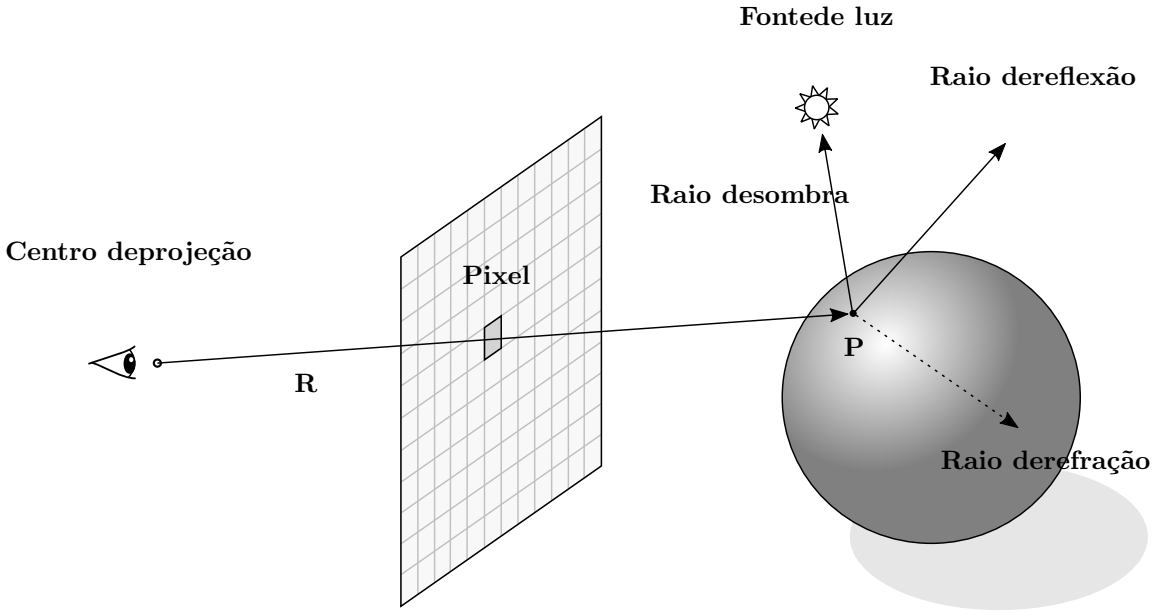
Figura 2: Ilustração da renderização de uma esfera por Rasterização



Fonte: Harlen Batagelo, Bruno Marques, 2021. Disponível:
`https://www.brunodorta.com.br/cg/ray-casting-x-rasteriza%C3%A7%C3%A3o.html`

Um método de renderização alternativo é o Traçamento de Raios (Ray Tracing), onde um ou mais raios são atirados para cada pixel da tela, e, após colidirem com os objetos da cena, a cor final do pixel é calculada. Essa estratégia simula de forma mais acurada o comportamento físico da luz, o que permite aproveitar o entendimento do funcionamento da luz para a implementação dos efeitos supracitados. Embora Ray Tracing esteja sendo cada vez mais utilizado em jogos digitais modernos, ainda se trata de um algoritmo computacionalmente caro, e, portanto, pouco utilizado em aplicações de tempo real.

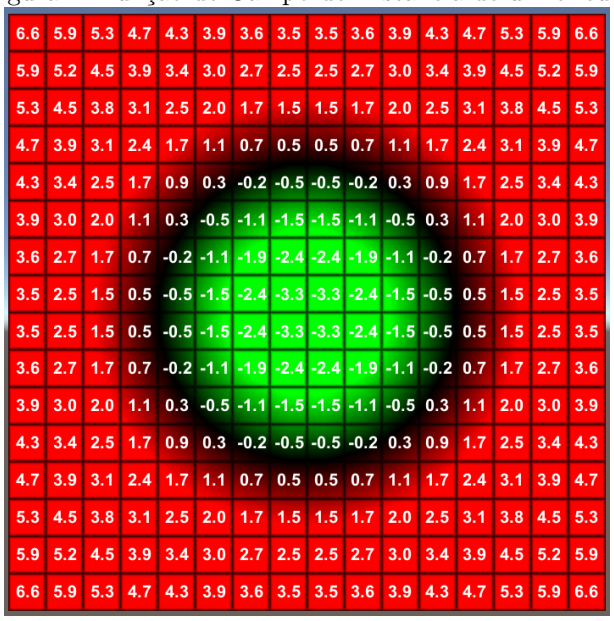
Figura 3: Ilustração da renderização de uma esfera por Ray Trancing



Fonte: Harlen Batagelo, Bruno Marques, 2021. Disponível:
`https://www.brunodorta.com.br/cg/ray-casting-x-rasteriza%C3%A7%C3%A3o.html`

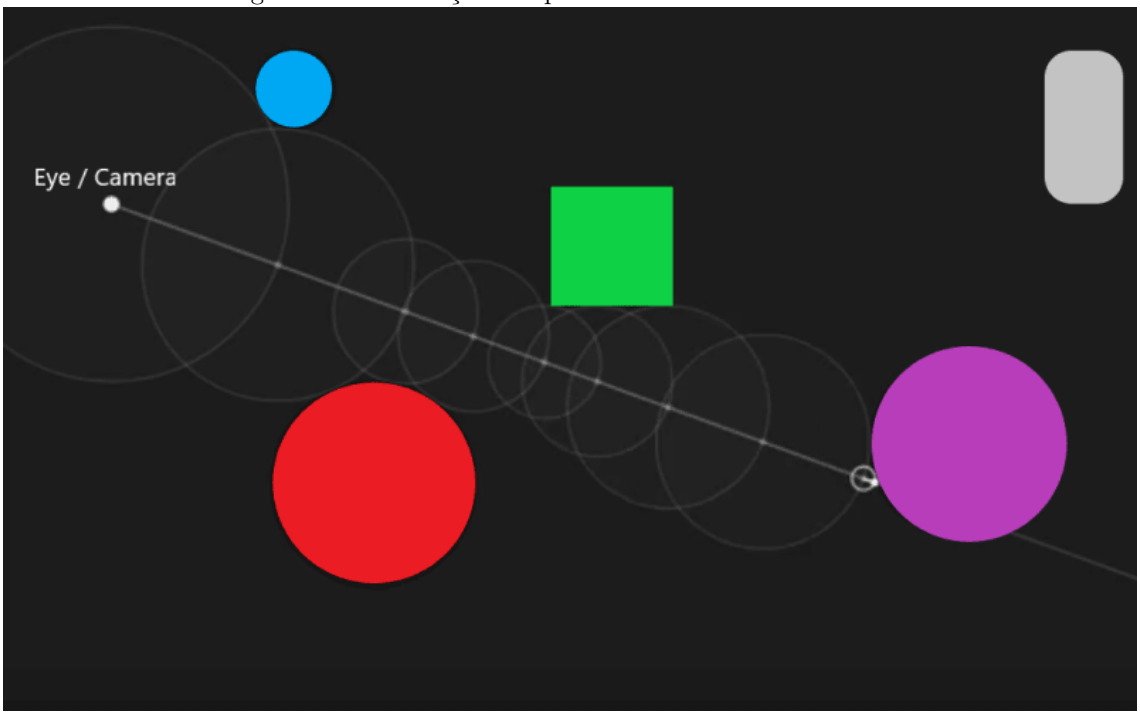
Como uma terceira alternativa, há o Ray Marching. Como o nome sugere, é um processo semelhante ao de Ray Tracing, onde um ou mais raios são atirados para cada pixel da tela. A diferença está no fato de que, ao invés de calcular a intersecção direta dos raios com a geometria, nesse algoritmo, o raio avança uma certa quantidade a cada passo. O quanto cada raio avança em um passo é dado pelo cálculo da distância da posição atual do raio para a geometria da cena. Esse processo é repetido até que a distância calculada seja igual (ou muito próxima de) zero, quando é considerado que ocorreu uma colisão com a geometria. Esse processo, no entanto, depende de termos um modelo muito específico de cena. Enquanto rasterizadores e traçadores de raios utilizam cenas com modelos 3D compostos de triângulos, a renderização por Ray Marching requer que as cenas sejam modeladas utilizando Funções de Campo de Distância. Essas funções são da forma $\mathbb{R}^n \to \mathbb{R}$, onde seu domínio corresponde aos pontos do espaço e sua imagem é a distância de cada ponto para a geometria da cena. Em inglês, elas são chamadas de "Signed Distance Fields"ou SDFs.

Figura 4: Função de Campo de Distância de um círculo



Fonte: Shader Fun, 2018. Disponível:
`https://shaderfun.com/2018/03/25/signed-distance-fields-part-2-solid-geometry/`

Figura 5: Visualização dos passos de um raio em uma cena



Fonte: Sebastian Lague, 2019. Disponível: `https://www.youtube.com/watch?v=Cp5WWtMoeKg&t=3s`

2 Objetivos

Este trabalho busca explorar a possibilidade de utilizar algoritmos de renderização baseados em Ray Marching em aplicações de tempo real, como jogos digitais, avaliando seus benefícios e problemas quando comparado com o estado da arte, além de estudar formas de otimizar tais algoritmos para esta finalidade.

Desta forma, o objetivo final deste trabalho consiste na criação de um renderizador baseado em Ray Marching para uso em tempo real.

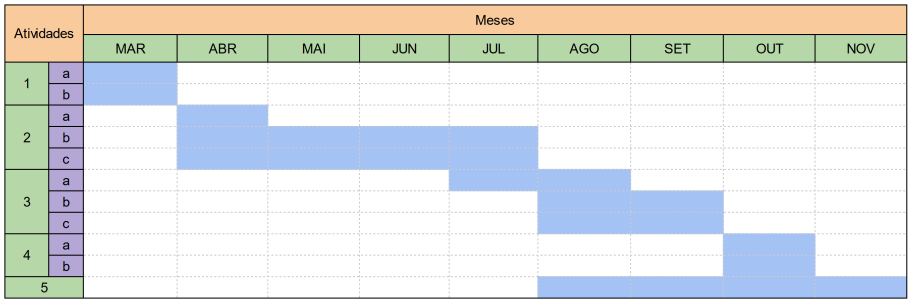
3 Metodologia

O trabalho será desenvolvido utilizando a linguagem de programação Rust, e o pacote WGPU. A escolha dessas ferramentas foi feita para que seja possível evitar a necessidade de lidar com sistemas como gerenciamento de janelas do sistema operacional e comunicação com os drivers de gráficos, tarefas que são feitas automaticamente pela framework WGPU, permitindo que o foco do projeto seja no desenvolvimento do renderizador. Todo o código fonte do projeto estará disponível por meio de um repositório no GitHub, sob a licença de software MIT. Além disso, o aluno estará em constante contato com o orientador ao longo do desenvolvimento do trabalho para garantir seu bom progresso.

4 Cronograma

O cronograma estipulado é o seguinte:

Figura 6: Tabela do cronograma estimado de atividades



Fonte: Arquivo do autor

Legenda das atividades:

1. Arquitetura do projeto
   (a) Definir ferramentas, linguagens de programação e frameworks
   (b) Arquitetura inicial do arcabouço de código
2. Renderização
   (a) Definição das etapas da pipeline de renderização
   (b) Implementação da pipeline de renderização
   (c) Implementação dos algoritmos de renderização
3. Usabilidade
   (a) Definição de formatos de descrição de arquivos
      i. Formato de descrição de modelos 3D
      ii. Formato de descrição de Cenas
      iii. Formato de descrição de shaders e materiais
   (b) Reestruturação do projeto em um formato de biblioteca que pode ser importado por outros projetos
   (c) Definição de uma API externa para uso do renderizador
4. Documentação
   (a) Criação de uma página web de documentação do projeto
   (b) Desenvolvimento de diversos exemplos de utilização do projeto
5. Escrita da monografia e preparação para a apresentação final

5 Referências

Referências

[1] Michael Abrash. Graphics programming black book. Disponível em: https://www.drdobbs.com/parallel/graphics-programming-black-book/184404919, 2001. Acesso em 18/04/2024.

[2] Harlen Batagelo e Bruno Marques. Mcta008-17 computação gráfica. Disponível em https://www.brunodorta.com.br/cg/. Acesso em 18/04/2024.

[3] Shader Fun. Signed distance fields part 1: Unsigned distance fields. Disponível em: https://shaderfun.com/2018/03/23/signed-distance-fields-part-1-unsigned-distance-fields/, Março 2018. Acesso em 18/04/2024.

[4] Steve Hollasch Peter Shirley, Trevor David Black. Ray tracing in one weekend. Disponível em: https://raytracing.github.io/books/RayTracingInOneWeekend.html, April 2024. Acesso em 18/04/2024.

[5] Scratchapixel. An overview of the rasterization algorithm. Disponível em: https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/overview-rasterization-algorithm.html. Acesso em 18/04/2024.

[6] Kent Sharkey David Coulter Drew Batchelor Michael Satran Steven White, Jason Martinez. Rasterization rules (direct3d 9). Disponível em: https://learn.microsoft.com/en-us/windows/win32/direct3d9/rasterization-rules, Abril 2021. Acesso em 18/04/2024.

[7] Alexandre Ziebert. Precisamos falar sobre ray tracing. Disponível em https://www.nvidia.com/pt-br/drivers/prbr-05282018/. Acesso em 18/04/2024.