

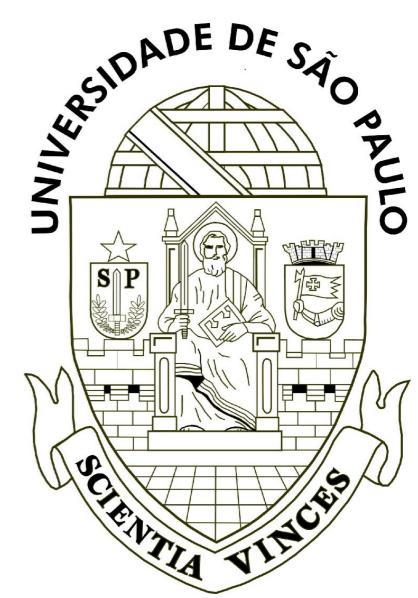


IME

# Renderização em Tempo Real utilizando Ray Marching e sua Aplicação em Jogos Digitais

Pedro Tonini Rosenberg Schneider

Departamento de Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo



## Introdução

O propósito principal deste trabalho é a avaliação da viabilidade da utilização da técnica de renderização de **Ray Marching**, particularmente utilizando o algoritmo de **Sphere Tracing**, para a renderização de cenas tridimensionais em aplicações interativas de tempo real, como jogos digitais. Além deste estudo, será desenvolvida uma prova de conceito de um renderizador utilizando a técnica de renderização supracitada como uma demonstração prática da viabilidade do algoritmo. Como o foco do trabalho são aplicações interativas, o renderizador terá funcionalidades para tal, sendo possível controlar aspectos das cenas utilizando entradas do usuário.

## Motivação

A **rasterização** é a técnica predominante na renderização de jogos 3D devido à sua eficiência e suporte por hardware gráfico dedicado. No entanto, seu uso impõe limitações no processo criativo devido a particularidades do funcionamento dessa técnica. Como exemplos, podemos citar:

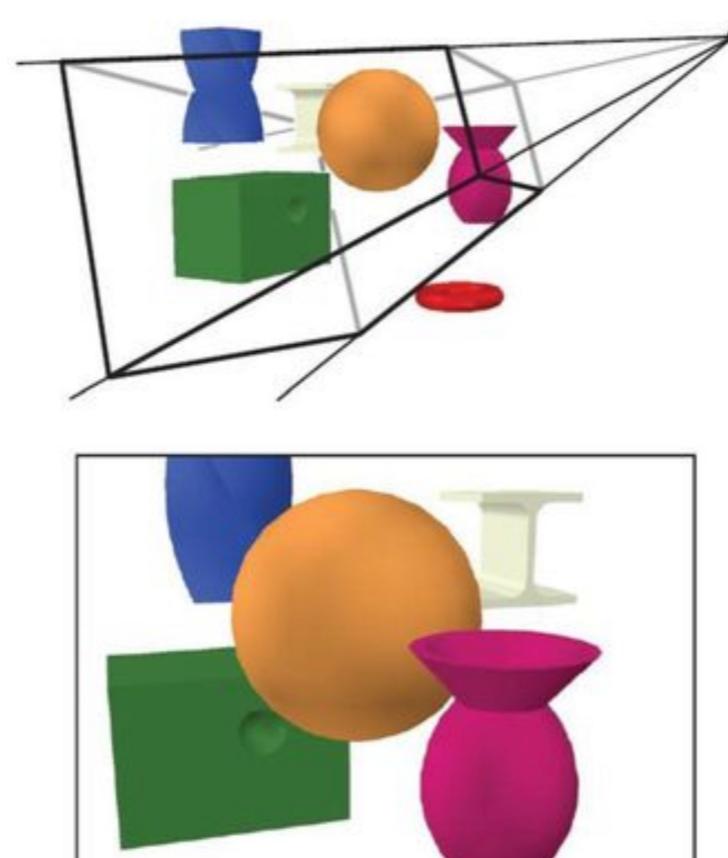
- **Efeitos complexos de iluminação** (reflexão, refração e Lei de Beer). Como o algoritmo de rasterização não simula o comportamento físico da luz no mundo real, reproduzir estes efeitos se torna computacionalmente custoso e pouco intuitivo;
- **Geometria procedural**. Como a grande maioria dos renderizadores trabalha com malhas de triângulos, qualquer sistema de geração procedural precisa criar essas malhas, dificultando, ou, até, inviabilizando, em alguns casos, sua execução em tempo real.

O **Ray Tracing** pode ser utilizado juntamente com a rasterização para simular efeitos de iluminação, mas é uma técnica computacionalmente cara, se tornando inviável para renderização completa de cenas complexas.

Nesse contexto, o **Ray Marching** surge como uma alternativa promissora, oferecendo maior flexibilidade criativa, especialmente para modelagem de superfícies complexas, como fractais. Por simular o comportamento da luz de forma semelhante ao **Ray Tracing**, é possível criar os mesmos efeitos de iluminação com **Ray Marching**. Adicionalmente, por operar sobre superfícies implicitamente definidas por funções de distância, a geração de conteúdo procedural também pode ser implementada diretamente.

## Estratégias de Renderização

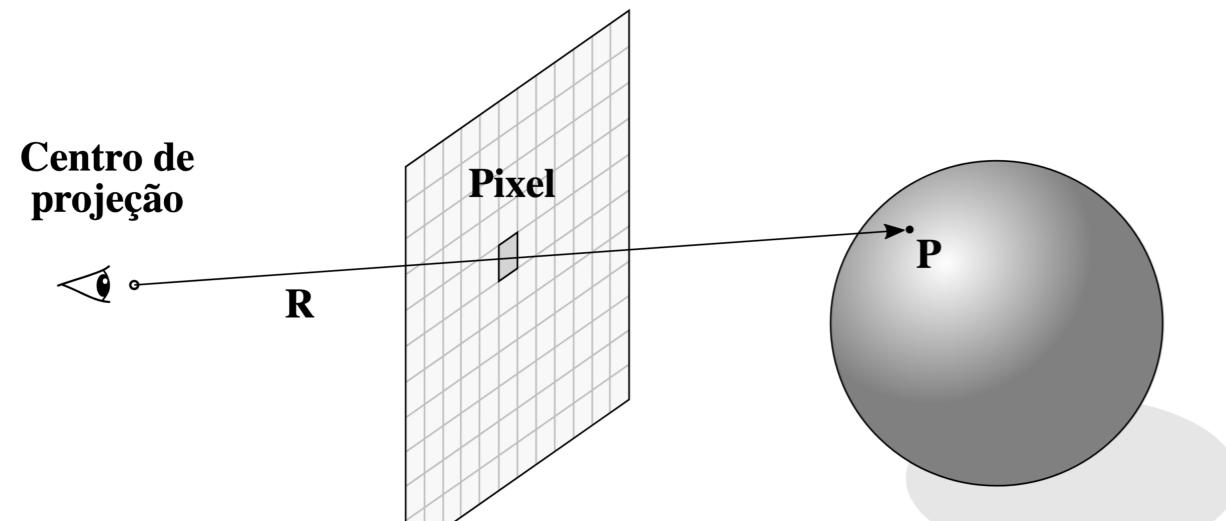
A função principal de qualquer **renderizador** é resolver o problema da **visibilidade de objetos**. Esse problema consiste na determinação de quais regiões de uma superfície estão visíveis para um determinado observador. Um algoritmo de renderização pode ser entendido como uma caixa preta que, para cada *pixel* da tela, dada uma determinada cena 3D, calcula a cor que deve ser exibida naquele *pixel* baseado nos objetos que são visíveis de um determinado referencial. Os algoritmos de renderização diferem, principalmente, na forma como resolvem o problema da visibilidade. A Figura ao lado mostra um exemplo de problema de visibilidade em uma cena contendo vários objetos.



Fonte: Allan Ryan, 2018.

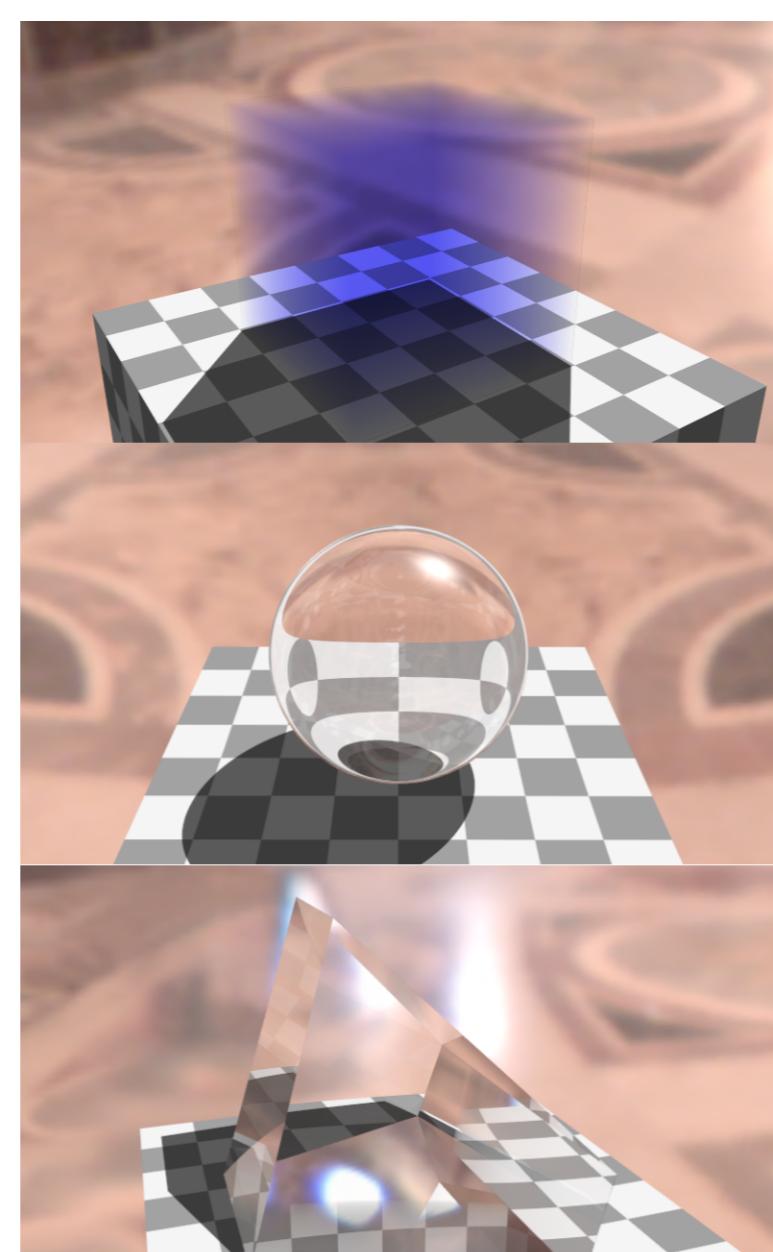
## Ray Tracing

**Ray Tracing** faz parte do grupo de técnicas baseadas na simulação do caminho dos raios de luz. Estas técnicas, de forma geral, procuram resolver o problema da visibilidade simulando como a luz reflete nas superfícies do mundo real, e como os olhos humanos percebem e processam esses raios de luz que chegam até eles. **Ray Tracing**, especificamente, utiliza fórmulas de intersecção e propriedades fotogramétricas dos objetos para calcular reflexões dos raios de luz na cena, até que eles cheguem à câmera. A Figura à esquerda mostra a renderização de uma esfera utilizando **Ray Tracing**. Para cada pixel da imagem final, é emitido um ou mais raios da perspectiva do observador ao pixel para calcular sua cor.



Fonte: Harlen Batagelo, Bruno Marques, 2021.

A Figura ao lado mostra alguns efeitos de iluminação que são possíveis utilizando **Ray Tracing**. De cima para baixo temos: Lei de Beer, refração e reflexão interna total. Esses efeitos são possíveis pois a técnica de **Ray Tracing** simula o comportamento real dos raios de luz e da ótica.

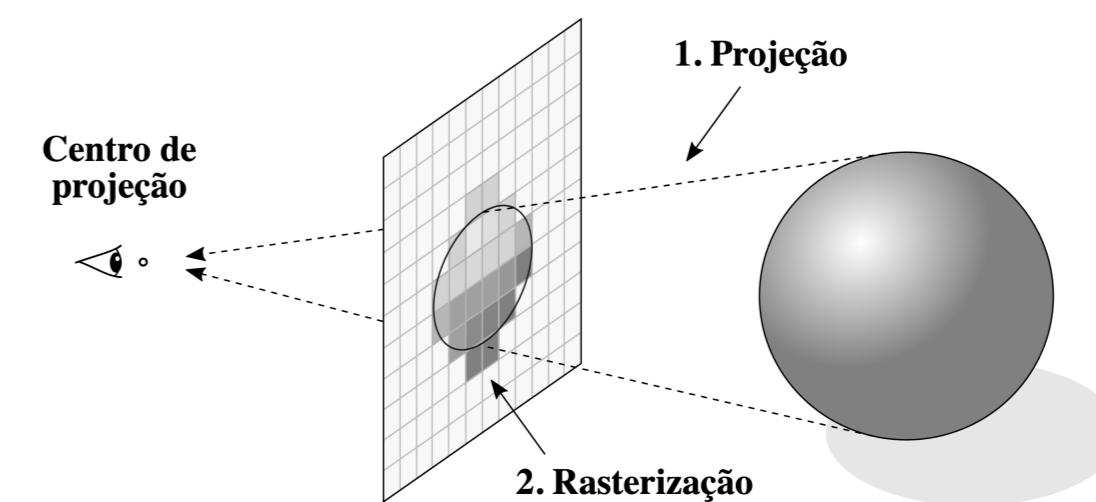


Fonte: Alan Wolfe, 2019.

Orientador: Prof. Dr. Márcio Lobo Netto

## Rasterização

Enquanto o **Ray Tracing** opera em um laço que itera por cada *pixel* na imagem, a **rasterização** foca primariamente nas formas primitivas que compõem a cena. Para renderizar uma cena tridimensional, um renderizador por rasterização passa por três etapas. Primeiro, é feita a projeção de todas as primitivas que compõem a cena sobre a imagem final. Embora seja possível calcular a projeção de diversas formas geométricas sobre um plano, o triângulo é a forma primitiva mais usualmente utilizada pelos renderizadores. Depois, é realizada a etapa de rasterização, que dá o nome para a técnica. Nesta etapa, é realizada a discretização da projeção das primitivas nos *pixels* da imagem; ou seja: o objetivo desta etapa é determinar quais *pixels* da imagem final fazem parte de cada objeto que foi projetado na etapa passada. Por fim, há a etapa de colorização. A Figura abaixo mostra o processo de rasterização de uma esfera.

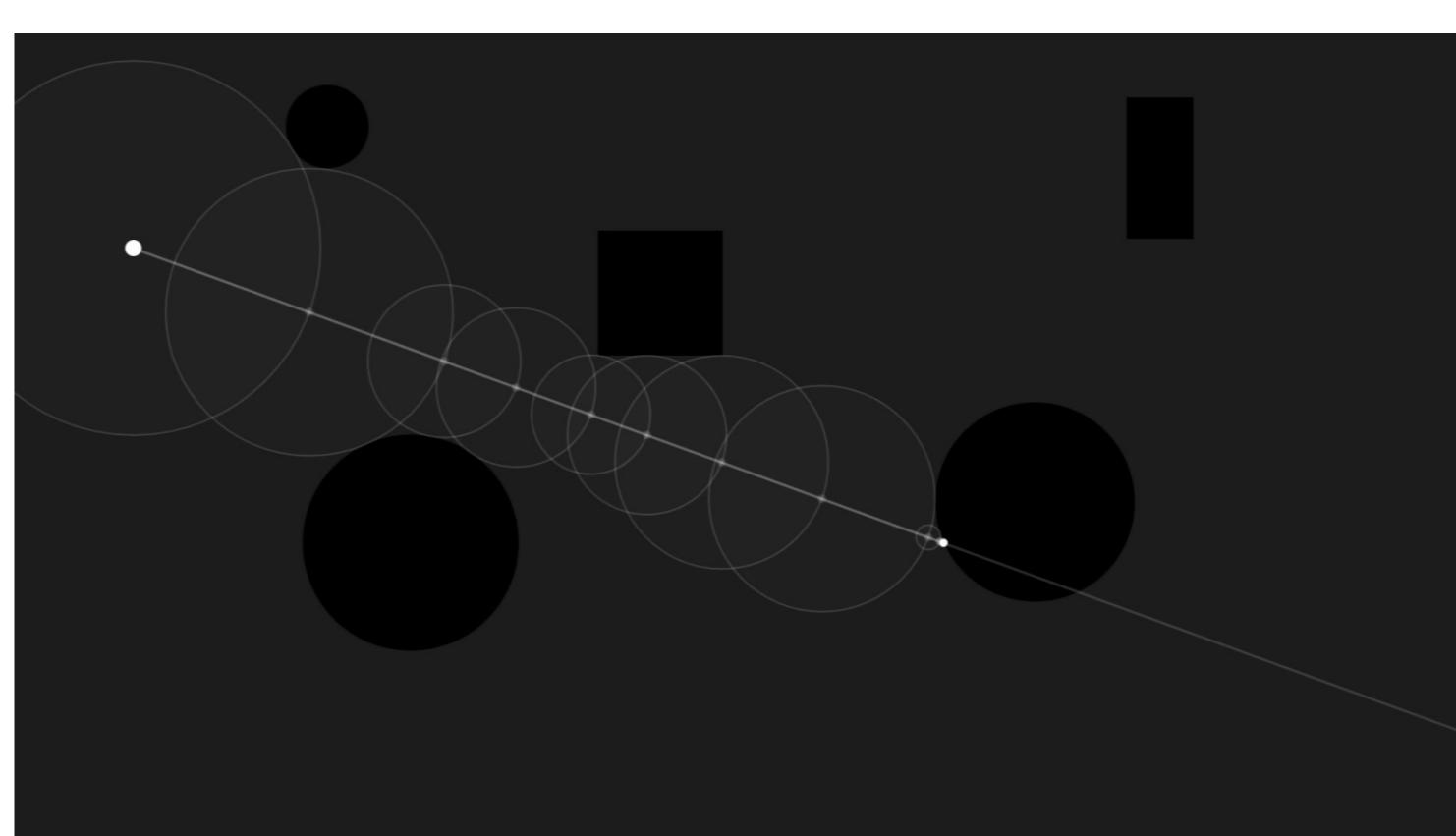


Fonte: Harlen Batagelo, Bruno Marques, 2021.

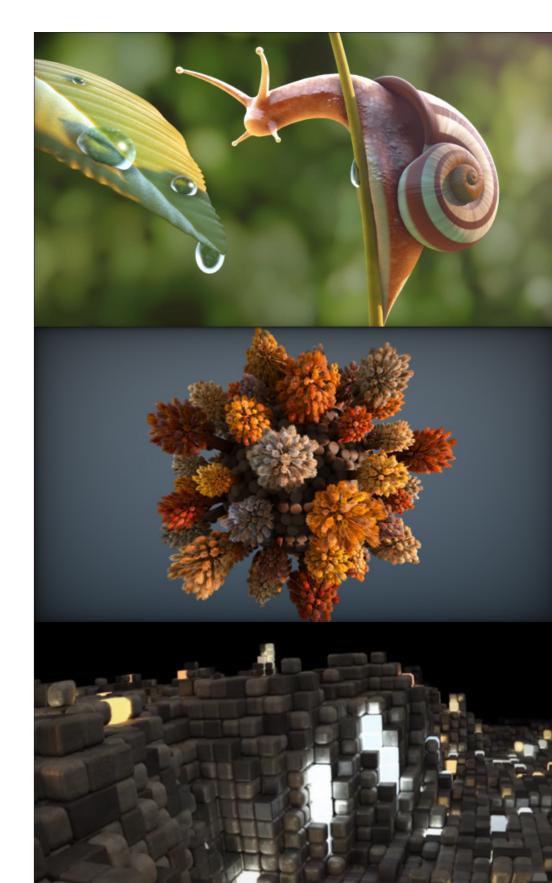
## Ray Marching e Sphere Tracing

A técnica de **Ray Marching** parte do mesmo princípio do **Ray Tracing**, visto que também utiliza raios propagados pela cena para resolver o problema da visibilidade. A diferença entre os dois está na forma como os raios são propagados. Enquanto no **Ray Tracing** a colisão dos raios com a geometria da tela é diretamente calculada por equações de intersecção, **Sphere Tracing** utiliza um processo iterativo para propagar o raio. Isso significa que, para cada raio que sai da câmera, é iniciado um laço, e, para cada iteração desse laço, o raio avança uma certa distância em sua direção.

**Sphere Tracing** se trata de uma forma específica de **Ray Marching** em que, para decidir o quanto um raio deve avançar em uma determinada iteração, **Ray Marching** utiliza funções de distância para descobrir o quanto distante a posição atual do raio está da geometria da cena. Essa distância pode ser visualizada ao pensar em uma esfera com centro na posição atual do raio. O diâmetro da esfera expande continuamente até que ela encoste em alguma superfície da geometria da cena. O raio pode, então, avançar a medida do raio da esfera sem colidir com nenhum objeto da cena. Esse processo é iterativamente repetido para cada raio, e, quando a distância retornada pela função de distância for zero, ou muito próxima de zero, sabemos que o raio colidiu com algum objeto da cena. Dessa forma, quando o raio está muito distante de qualquer geometria da cena ele poderá avançar mais, e, quando estiver perto de algum objeto, irá avançar mais devagar para garantir que não atravessará nenhuma parte da geometria. A Figura à esquerda mostra a utilização de **Sphere Tracing** para propagar um raio em uma cena. Cada um dos círculos na figura representa uma iteração do algoritmo de **Ray Marching**. A Figura à direita mostra três exemplos de cenas renderizadas utilizando **Ray Marchinig** e **Sphere Traching**.



Fonte: Fonte: Sebastian Lague, 2019.



Fonte: Inigo Quilez, 2020.

## Informações

Para mais informações, visite a página do projeto ou o repositório com o código desenvolvido:

- <https://www.linux.ime.usp.br/~ptrschneider/mac0499/>
- <https://github.com/pedrotrschnieder/blob-engine>

Escaneie o **QR Code** ao lado para ver mais recursos interessantes sobre **Ray Marching**.



## Referências

- Tomas Akenine-Möller and Eric Haines and Naty Hoffman, "Real-Time Rendering 3rd Edition", 2008.
- Michael Abrash, "Michael Abrash's Graphics Programming Black Book", 1997.
- John C. Hart, "Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces", in The Visual Computer, 12(10):527-545, 1996.
- Hart, J. C. and Sandin, D. J. and Kauffman, L. H., "Ray tracing deterministic 3-D fractals", in ACM SIGGRAPH Computer Graphics, 23(3):289-296, 1989.