Mateus de Freitas Leite

Renderização de superfícies baseadas em partículas utilizando NVIDIA OptiX

Recife

Dezembro, 2017

Mateus de Freitas Leite

Renderização de superfícies baseadas em partículas utilizando NVIDIA OptiX

Trabalho apresentado ao Programa de Bacharelado em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Universidade Federal de Pernambuco - UFPE

Centro de Informática - CIn

Orientador: Silvio de Barros Melo

Recife

Dezembro, 2017

Agradecimentos

A lista de pessoas a agradecer por hoje eu ter chegado até aqui é bem grande, de tal maneira que, se eu for mencionar cada uma delas, eu escreveria um novo trabalho. Mas ainda assim, irei mencionar as que tiveram uma participação mais direta neste trabalho.

Aos meus pais, é claro, por nunca terem desistido de mim e me dado todo o apoio para continuar, até nos momentos em que eu mesmo pensei em largar tudo.

Aos meus amigos de universidade (Figueira, Castiel, França, Lucas, George, Belga, Peu... eu disse que a lista era grande, e ainda nem cheguei à metade), pelos bons momentos de conversa e pelas excelentes ideias que surgiam daí, o que me fez amadurecer bastante.

Ao meu orientador, Silvio Melo, por ter sido um exemplo para mim de que humildade e brilhantismo podem sim andar juntos, além de sempre ter me fornecido ótimos materiais para dar início aos meus trabalhos. Também gostaria de agradecer neste último aspecto à professora Verônica Teichrieb e ao Voxar Labs por tornar a concepção do tema deste trabalho possível.

Aos meus antigos colegas (e também amigos!) do Grupo de Realidade Virtual e Multimídia (GRVM): eu não teria me tornado o profissional que sou hoje sem vocês.

Por fim, gostaria também de agradecer ao Centro de Informática da UFPE por ter proporcionado uma experiência tão importante na minha vida. Pela primeira vez, tenho a sensação de que me tornei uma pessoa mais madura, além de começar a entender o verdadeiro significado de respeitar as diferenças, seja no âmbito de vivência pessoal, seja no lado profissional.

A intuição é uma das maiores facetas humanas. Você intui,
e mesmo que não seja do nada, não se sabe de onde...
mas que intui, intui.
—CLYLTON GALAMBA

Lista de ilustrações

Lista de tabelas

Sumário

Resumo

Superfícies baseadas em partículas são bastante utilizadas em aplicações de Engenharia que envolvem a discretização de um modelo físico contínuo, como em simulações de fluidos, por exemplo. Em particular, a necessidade de visualizar um sistema que envolve um grande número de partículas se tornou bastante comum. Tais ferramentas de visualização necessitam de uma qualidade visual fotorrealista, além de ser bastante desejável um alto desempenho em termos de taxa de quadros renderizados em um determinado intervalo de tempo. Este trabalho apresenta uma implementação de um método de renderização de superfícies com até 10 milhões de partículas utilizando NVIDIA OptiX, descrevendo as vantagens e as dificuldades na utilização da API.

Palavras-chave: reconstrução de superfícies, modelos baseados em partículas, NVIDIA OptiX, renderização de alto desempenho.

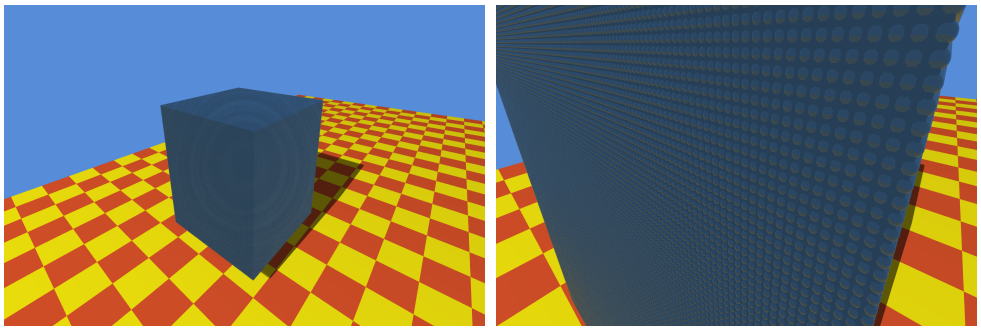
Abstract

Particle-based surfaces are often used in Engineering applications which involve discretizing a continuum physical model (e.g in fluid simulation systems). In particular, the need of previewing tools able to render a large number of particles became very common. Such tools need photorealistic visual quality on final rendering. It is also desirable to have a high performance in terms of the rate of frames rendered per a time interval. This work presents an implementation of a rendering method of surfaces with up to 10 million particles using NVIDIA OptiX while describing the advantages and the hardships of using this API.

Keywords: surface reconstruction, particle-based models, NVIDIA OptiX, high performance rendering.

1 Introdução

Métodos de *síntese de imagens* (ou *renderização*) são amplamente empregados em diversas áreas, em particular nas ferramentas de visualização científica e *design*, onde muitas vezes as aplicações necessitam de algoritmos eficientes para gerar a cena a partir de um modelo geométrico. Por exemplo, em ferramentas de simulações físicas, reproduzir as interações entre os corpos do sistema modelado a uma taxa de *frames* aceitável reduz o tempo de trabalho para remodelagem em caso de erro na construção do modelo físico. Tais ferramentas são bastante utilizadas em aplicações de Engenharia e nas indústrias de jogos eletrônicos e cinematográfica. Além disso, a qualidade visual da cena renderizada também é fundamental para certas aplicações de *Computer Aided Design*, como em projeções de ambientes internos na área de *design* de arquitetura.



(a) (b)

Figura 1 – Um cubo com cerca de 10 milhões de partículas renderizado pela ferramenta desenvolvida é mostrado na Figura 1a. Uma visão mais detalhada da representação de cada partícula pode ser visualizada na Figura 1b.

No escopo deste trabalho, o objetivo geral consiste em fornecer uma implementação de um método de síntese de imagens aplicado para reconstrução de superfícies baseadas em um sistema discretizado em um grande conjunto de partículas. Tais sistemas são bastante utilizados em simulações de efeitos violentos de fluidos, como fragmentações e deformações em larga escala (REICHL et al., 2014). A aplicação apresentada neste trabalho utilizou o NVIDIA OptiX (PARKER et al., 2010) como *framework* para implementar algoritmos baseados em *ray tracing* (WHITTED, 1979) com um alto desempenho. Aqui, o método de renderização implementado utiliza o *ray tracing* tanto para reconstruir a superfície, quanto para o cálculo da iluminação global da cena. O foco da aplicação contemplado neste trabalho consiste em avaliar o OptiX em termos de desempenho e sintetizar os detalhes de implementação que permitiram um uso eficiente do *framework*. Os resultados mostram que as superfícies reconstruídas neste trabalho possuem até 10 milhões de partículas (Figura 1), e atingiram uma taxa de *frames* por segundo significativa, o que indica o potencial do OptiX para o tipo de aplicação abordado neste trabalho.

Apresentamos a estrutura deste trabalho a seguir. O Capítulo 2 provê um embasamento teórico acerca do problema da iluminação e das grandezas físicas diretamente relacionadas ao cálculo da quantidade de luz na cena. Uma descrição dos trabalhos diretamente relacionados ao método implementado é fornecida no Capítulo 3. O Capítulo 4, por sua vez, descreve o *pipeline* geral da aplicação, além de apresentar alguns detalhes de implementação fundamentais para que a ferramenta pudesse adquirir um bom desempenho. Apresentamos também a metodologia utilizada para avaliar a performance da aplicação no Capítulo 5, além de interpretar os resultados obtidos nos experimentos. Por fim, um resumo dos avanços alcançados com este trabalho e das dificuldades encontradas durante o processo são detalhadas no Capítulo 6, além de mencionar possíveis extensões da ferramenta a serem exploradas.

2 Fundamentação Teórica

Neste capítulo, será abordada a teoria em que os métodos de iluminação global se baseiam. A Seção 2.1 descreve os conceitos e princípios da Radiometria. A seção 2.2, por sua vez, define o que é uma BRDF e suas propriedades. Por fim, a Seção 2.3 descreve a equação de iluminação global. O caráter da abordagem teórica deste capítulo é apenas introdutório. O leitor conseguirá uma teoria mais detalhada em (KAJIYA, 1986), (SHIRLEY; MARSCHNER, 2009), (DUTRE et al., 2006), (GOMES; VELHO, 2008) e (JUDICE; GIRALDI; FILHO, 2011).

2.1 Radiometria

A Radiometria é uma área da Óptica que se dedica à medição do transporte de energia radiante. Em particular, ela provê ferramentas para descrever a propagação de luz. Utiliza-se a letra Q para representar a energia radiante, cuja unidade de medida no padrão SI é J (Joule). As principais grandezas envolvidas na medição da energia radiante são as seguintes: *potência radiante* (ou *fluxo de radiação*), *radiância*, *irradiação* e *radiosidade*. Cada uma dessas grandezas será definida a seguir.

2.1.1 Fótons

A fim de auxiliar no raciocínio nas próximas subseções, descreveremos as grandezas radiométricas em função de uma quantidade de *fótons*. Aqui definimos um fóton como um *quantum* de luz que possui uma posição, uma direção de propagação e um comprimento de onda λ. A medida que nos interessa para o cálculo de energia radiante Q é a *quantidade de energia* de um fóton, dada por

$$q = \frac{hc}{\lambda}, \tag{2.1}$$

onde $h \approx 6,63 \cdot 10^{-34}\text{J} \cdot \text{s}$ é a *constante de Planck*, e c é a velocidade da luz. Ainda podemos substituir o termo $\frac{c}{\lambda} = f$, onde f é a *frequência* do fóton. Assim, a equação 2.1 ficaria da seguinte maneira:

$$q = hf. \tag{2.2}$$

2.1.2 Fluxo de Radiação

O *fluxo de radiação* (Φ) mede a taxa de energia radiante que atravessa uma superfície por unidade de tempo:

$$\Phi = \frac{dQ}{dt}. \tag{2.3}$$

Sua unidade de medida no padrão SI é $\mathrm{W} = \mathrm{J} \cdot \mathrm{s}^{-1}$ (Watt). A emissão total de energia radiante proveniente de um conjunto de fontes luminosas geralmente é medida em termos do fluxo.

2.1.3 Radiância

Seja $n_{x\to\mathbf{u}}$ o número de fótons que fluem em uma direção $\mathbf{u}$ partindo de um ponto x. A quantidade total de energia deste conjunto de fótons é, portanto, $qn_{x\to\mathbf{u}} = hfn_{x\to\mathbf{u}}$. Suponha agora que esse feixe de fótons flua por um elemento de superfície dA confinado em um elemento de ângulo sólido $d\omega$. Então o feixe de fótons está confinado no elemento de volume $dA\cos\theta d\omega ds = cdA\cos\theta d\omega dt$, onde θ é o menor ângulo entre um vetor $\mathbf{n}$ normal ao elemento de superfície dA e $\mathbf{u}$. A Figura 2 ilustra a situação descrita.

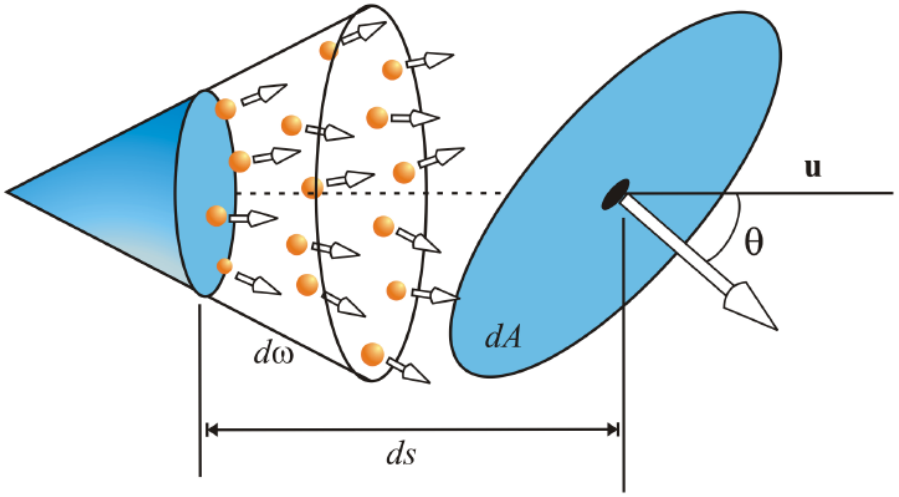


Figura 2 – Feixe de fótons num elemento de volume $dA\cos\theta d\omega ds$. Fonte: (GOMES; VELHO, 2008).

A quantidade de energia radiante dQ que flui na direção $\mathbf{u}$ a partir do ponto x é, portanto,

$$dQ = \left(\int_\Omega \int_A hfcn_{x\to\mathbf{u}} dA\cos\theta d\omega\right) dt. \tag{2.4}$$

Da equação 2.4, concluímos que o fluxo de radiação originado do ponto x na direção $\mathbf{u}$ é

$$\Phi_{x\to\mathbf{u}} = \int_\Omega \int_A (hfcn_{x\to\mathbf{u}}) dA\cos\theta d\omega. \tag{2.5}$$

A grandeza $hfcn_{x\to\mathbf{u}}$ é denominada *radiância* e é indicada pelo símbolo $L_{x\to\mathbf{u}}$. Assim,

$$L_{x\to\mathbf{u}} = \frac{d^2\Phi_{x\to\mathbf{u}}}{dA\cos\theta d\omega},\ [L_{x\to\mathbf{u}}] = \mathrm{W}\cdot\mathrm{m}^{-2}. \tag{2.6}$$

Da definição exposta na equação 2.6, podemos interpretar a radiância como a variação da densidade de fluxo de radiação em relação ao ângulo sólido numa determinada direção $\mathbf{u}$.

De fato,

$$L_{x\to\mathbf{u}} = \frac{d}{d\omega}\left(\frac{d\Phi_{x\to\mathbf{u}}}{dA\cos\theta}\right) = \frac{dE}{d\omega}. \tag{2.7}$$

2.1.4 Irradiação e Radiosidade

A radiância consegue medir a quantidade de energia radiante proveniente de uma determinada direção, o que permite modelar o efeito da luz recebida por um observador numa determinada posição, por exemplo. No entanto, ela não diz respeito à quantidade de energia radiante *total* emitida ou recebida por um objeto, independentemente da direção. As grandezas definidas a seguir buscam modelar essa situação.

Definimos a *irradiação* como a densidade por área de fluxo de radiação **incidente** em um ponto x de uma superfície numa direção $\mathbf{u}$. Assim:

$$E_x = \frac{d\Phi_{x\leftarrow\mathbf{u}}}{dA} = \int_\Omega L_{x\leftarrow\mathbf{u}}\cos\theta d\omega. \tag{2.8}$$

De maneira análoga, a *radiosidade* é definida como a densidade por área de fluxo de radiação **originada** em um ponto x de uma superfície numa direção $\mathbf{u}$:

$$B_x = \frac{d\Phi_{x\to\mathbf{u}}}{dA} = \int_\Omega L_{x\to\mathbf{u}}\cos\theta d\omega. \tag{2.9}$$

2.2 Função de Distribuição de Reflectância Bidirecional (BRDF)

Dependendo do ponto de vista do observador, a aparência de um objeto iluminado por um certo conjunto de fontes de luz pode variar. Para modelar este comportamento do ponto de vista da Óptica Geométrica, assumimos três principais elementos influentes na radiância emitida pelo objeto: iluminação, propriedades de reflectância da superfície e posição do observador. Assumimos também que o meio de propagação é *não participativo*, isto é, a trajetória da luz não é influenciada pelo meio em que propaga, e não há absorção de energia.

A *função de distribuição de reflectância bidirecional* (BRDF, do inglês *Bidirectional Reflectance Distribution Function*) busca modelar a reflectância de um material. Considere que a luz incida em um ponto x do objeto vindo de uma direção $\mathbf{v_i}$. Queremos saber a radiância recebida por um observador originada de x com uma direção $\mathbf{v_o}$. Temos que um elemento de irradiância $dE_{x\leftarrow\mathbf{v_i}}$ é dado por

$$dE_{x\leftarrow\mathbf{v_i}} = L_{x\leftarrow\mathbf{v_i}}\cos\theta d\omega_{\mathbf{v_i}}. \tag{2.10}$$

Pela hipótese da linearidade da Óptica Geométrica (PHAAR; HUMPHREYS, 2004), também temos que o elemento de radiância refletida é diretamente proporcional ao elemento de irradiância incidente:

$$dL_{x\to\mathbf{v_o}} \propto dE_{x\leftarrow\mathbf{v_i}}, \tag{2.11}$$

ou ainda,

$$dL_{x\to\mathbf{v_o}} = f_r(x,\mathbf{v_i},\mathbf{v_o})dE_{x\leftarrow\mathbf{v_i}}. \tag{2.12}$$

A função f_r é definida, portanto, como a nossa função de distribuição de reflectância bidirecional.

Uma BRDF deve possui algumas propriedades. Por exemplo, uma BRDF deve ser sempre não-negativa, ou seja, $f_r(x,\mathbf{v_i},\mathbf{v_o}) \geq 0$ para todo $x,\mathbf{v_i},\mathbf{v_o}$. disso, os valores da BRDF devem ser os mesmos no caso de o raio incidente e o refletido serem trocados, isto é, $f_r(x,\mathbf{v_i},\mathbf{v_o}) = f_r(x,\mathbf{v_o},\mathbf{v_i})$. O princípio por trás desta propriedade é conhecido como *reciprocidade de Helmholtz*. Por fim, da *lei de conservação de energia*, podemos afirmar que a radiância refletida deve ser menor ou igual à incidente. Segue desta afirmação que

$$\int_\Omega f_r(x,\mathbf{v_i},\mathbf{v_o})d\omega_{\mathbf{v_i}} \leq 1. \tag{2.13}$$

2.3 A Equação de Iluminação

A equação de iluminação determina a radiância originada de um ponto x e cuja direção é $\mathbf{v_o}$. Em outras palavras, ela busca determinar a quantidade de luz que é transportada até o observador. Determina-se $L_{x\to\mathbf{v_o}}$ em função da radiância emitida pelo objeto, $L^{(e)}_{x\to\mathbf{v_o}}$, e a refletida, $L^{(r)}_{x\to\mathbf{v_o}}$. Temos, portanto, que

$$L_{x\to\mathbf{v_o}} = L^{(e)}_{x\to\mathbf{v_o}} + L^{(r)}_{x\to\mathbf{v_o}}. \tag{2.14}$$

Das equações 2.10 e 2.12, temos o seguinte resultado:

$$L^{(r)}_{x\to\mathbf{v_o}} = \int_\Omega f_r(x,\mathbf{v_i},\mathbf{v_o})dE_{x\leftarrow\mathbf{v_i}} = \int_\Omega f_r(x,\mathbf{v_i},\mathbf{v_o})L_{x\leftarrow\mathbf{v_i}}\cos\theta d\omega_{\mathbf{v_i}}. \tag{2.15}$$

Substituindo por fim 2.15 em 2.14, obtemos então a nossa *equação de iluminação*:

$$L_{x\to\mathbf{v_o}} = L^{(e)}_{x\to\mathbf{v_o}} + \int_\Omega f_r(x,\mathbf{v_i},\mathbf{v_o})L_{x\leftarrow\mathbf{v_i}}\cos\theta d\omega_{\mathbf{v_i}}. \tag{2.16}$$

3 Trabalhos Relacionados

Neste capítulo, apresentamos os trabalhos diretamente empregados no método de renderização implementado neste trabalho. A Seção 3.1 descreve brevemente o algoritmo de *ray tracing*, um método computacional proposto por (WHITTED, 1979) como uma solução aproximada da equação de iluminação 2.16. Por último, na Seção 3.2, introduzimos e descrevemos algumas das características da principal ferramenta estudada neste trabalho: o NVIDIA OptiX.

3.1 Ray Tracing

O *ray tracing* é um algoritmo que simula a trajetória inversa da luz que chega ao observador, traçando raios originados de uma câmera posicionada atrás de uma cena em 3D. Quando um raio atinge uma superfície, a cor do ponto é computada levando em consideração as contribuições luminosas *diretas*, ou seja, as que foram originadas de outras fontes de luz, e as *indiretas*, que são resultantes de raios refletidos por outros objetos. As componentes são calculadas levando em conta efeitos como reflexão e refração, além de considerar a iluminação ambiente da cena.

Para calcular o ponto de interseção do raio em um objeto da cena, a abordagem mais simples (e menos eficiente) seria realizar o teste de interseção em todos os objetos. No entanto, tal método é bastante ineficiente, o que torna necessária a utilização de estruturas de dados eficientes para representar a disposição espacial dos objetos. Algumas das mais utilizadas são a *Bounding Volume Hierarchy* (BVH), proposta em (KAY; KAJIYA, 1986), e a *k-d tree* (BENTLEY, 1975). Variações destas estruturas foram criadas ao longo do tempo, buscando otimizar desempenho ou memória. O OptiX implementa várias dessas estruturas, algumas das quais são mencionadas na seção a seguir.

3.2 NVIDIA OptiX

O NVIDIA OptiX (PARKER et al., 2010) é um *framework* para aplicações de renderização que provê ferramentas para implementar algoritmos baseados em *ray tracing* com um alto desempenho. Ele fornece um *pipeline* recursivo e, portanto, simples de compreender, além de possibilitar uma certa flexibilidade quanto ao uso de estruturas de aceleração para algoritmos geométricos utilizados em aplicações de *ray tracing*, como BVHs e variações, dentre as quais podemos mencionar as SBVHs (*Split Bounding Volume Hierarchies*) (STICH; FRIEDRICH; DIETRICH, 2009) e as LBVHs (*Linear Bounding Volume Hierarchies*) (LAUTERBACH et al., 2009), (GARANZHA; PANTALEONI; MCALLISTER, 2011), por exemplo. O programa 3.1

mostra uma possível implementação para calcular a radiância em um determinado ponto da cena utilizando o OptiX.

Programa 3.1 – Procedimento para cálculo da radiância após a interseção com o raio e o objeto mais próximo (*closest-hit*). O código abaixo apenas efetua o cálculo da componente direta utilizando a BRDF de Blinn-Phong (BLINN, 1977). Adaptado de (NVIDIA CORPORATION, 2014–2017).

```
RT_PROGRAM void closestHitRadiance() {
  float3 world_geo_normal = normalize(
      rtTransformNormal(RT_OBJECT_TO_WORLD, geometric_normal));
  float3 world_shade_normal = normalize(
      rtTransformNormal(RT_OBJECT_TO_WORLD, shading_normal));
  float3 ffnormal = faceforward(
      world_shade_normal,
      -ray.direction,
      world_geo_normal);
  float3 color = Ka * ambient_light_color;

  float3 hit_point = ray.origin + t_hit * ray.direction;

  for (int i = 0; i < lights.size(); ++i) {
    BasicLight light = lights[i];
    float3 L = normalize(light.pos - hit_point);
    float nDl = dot( ffnormal, L);

    if (nDl > 0.0f) {
      // Cast shadow ray.
      PerRayData_shadow shadow_prd;
      shadow_prd.attenuation = 1.0f;
      float Ldist = length(light.pos - hit_point);
      optix::Ray shadow_ray(
          hit_point,
          L,
          shadow_ray_type,
          scene_epsilon,
          Ldist);
      rtTrace(top_shadower, shadow_ray, shadow_prd);
      float light_attenuation = shadow_prd.attenuation;

      if (light_attenuation > 0.0f ) {
        float3 Lc = light.color * light_attenuation;
        color += Kd * nDl * Lc;
```

```
        float3 H = normalize(L - ray.direction);
        float nDh = dot( ffnormal, H );
        if(nDh > 0)
          color += Ks * Lc * pow(nDh, phong_exp);
      }
    }
  }
  prd_radiance.result = color;
}
```

Aplicações comerciais de diversas áreas como as indústrias cinematográficas e de jogos eletrônicos, design e visualização científica utilizam o OptiX. Algumas delas incluem *renderers* comerciais como o NVIDIA Iray (NVIDIA CORPORATION, 2017) e o FurryBall da AAAStudio (AAA STUDIO, 2015). Na esfera acadêmica, podemos mencionar o VMD (Visual Molecular Dynamics) (NIH CENTER FOR MACROMOLECULAR MODELING AND BIOINFORMATICS (UNIVERSITY OF ILLINOIS), 2006–2017), um programa de visualização que permite analisar e animar sistemas biomoleculares em cenas 3D.

4 O Método de Renderização

Descreveremos aqui o método baseado em *ray tracing* implementado neste trabalho. A seção 4.1 fornece uma visão geral do *pipeline* da aplicação. A Seção 4.2, por sua vez, detalha a solução numérica aproximada para o problema da iluminação que foi implementada. Por fim, os detalhes relacionados à otimização de desempenho e à representação da cena utilizando o OptiX são expostos na Seção 4.3.

4.1 Visão Geral

A Figura 3 mostra o diagrama do *pipeline* de renderização do método. A aplicação inicialmente cria um *contexto*, que é a maneira com a qual o OptiX fornece uma representação da cena. No contexto são definidas as variáveis utilizadas pelas instâncias de programas do OptiX (e.g. *closest-hit*, *any-hit* e interseção). Após a criação do contexto, ele é então *validado* a fim de verificar a existência de programas necessários para uma determinada instância. Por exemplo, um programa de interseção é sempre obrigatório para uma instância geométrica da cena. Além disso, a validação checa se há estados inválidos no contexto, como variáveis que são utilizadas, mas que não são definidas, ou se há compartilhamento indevido dessas variáveis entre programas. Por fim, o contexto é *compilado* para gerar um *kernel*, que será executado a cada vez que o contexto for lançado.

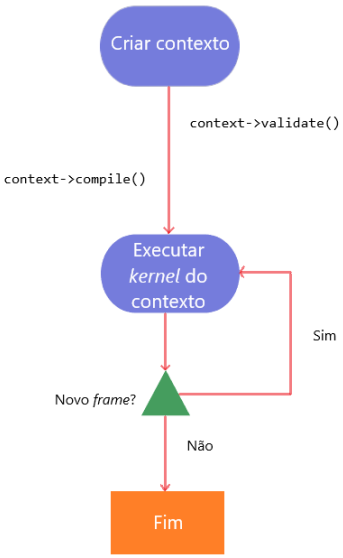


Figura 3 – O *pipeline* da aplicação.

Após a etapa inicial de configuração, o *kernel* gerado é executado para cada quadro da cena. Para cada raio lançado a cada chamada da função *rtTrace* no *kernel*, um grafo que associa a hierarquia dos programas e ações do contexto da cena é processado (Figura 4). O ponto de

entrada para a inicialização deste fluxo é o *programa de geração de raios* (Programa 4.1), que determina quais os raios amostrados dentre os visualizados pelo observador da cena. O fluxo de execução a cada chamada de *rtTrace* segue então os seguintes passos:

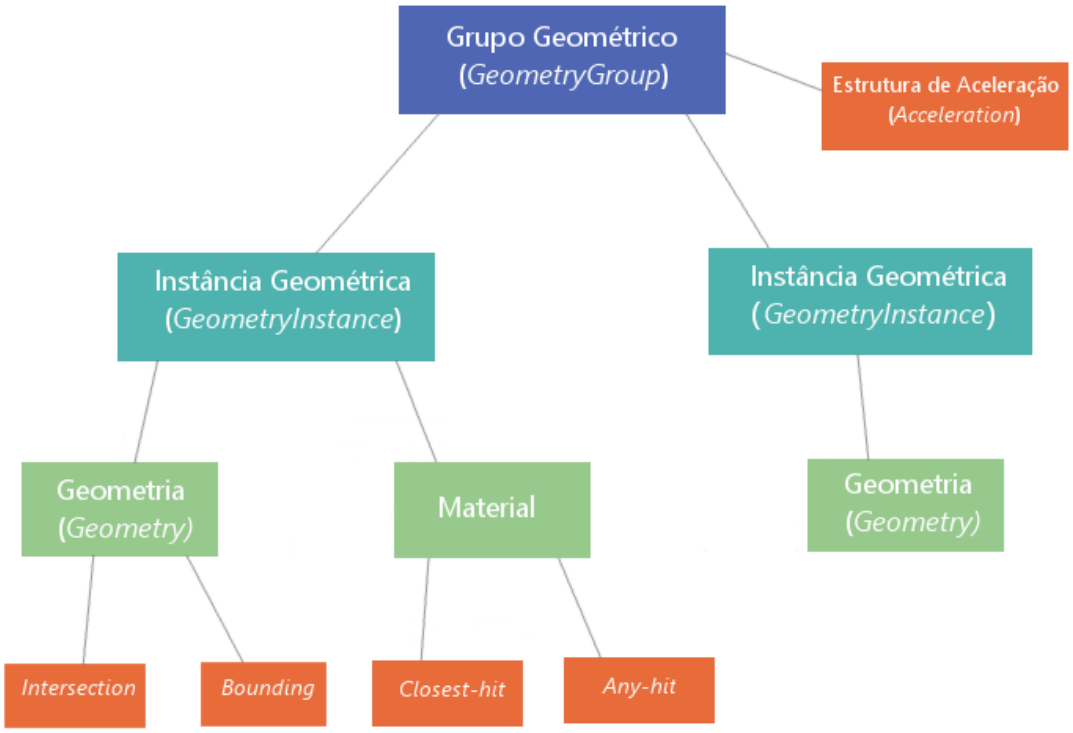


Figura 4 – Um exemplo de grafo que representa uma hierarquia dos elementos de um contexto. É nele que é determinado qual programa de interseção a ser chamado, ou como o objeto deve ser iluminado dependendo do seu material.

- A estrutura de aceleração que representa a cena é percorrida pra determinar qual o objeto mais próximo da origem do raio que é intersectado. Para isso, uma função de interseção associada a *geometria* do objeto é chamada para cada primitiva da cena.
- Caso nenhum objeto seja intersectado pelo raio, o *miss program* do contexto é executado, retornando assim a cor do fundo da cena para o pixel em que o raio foi lançado. Caso contrário, executa-se o *closest-hit program* do *material* do objeto intersectado para determinar sua cor naquele pixel.

Programa 4.1 – Exemplo de programa de geração de raios (*ray generation program*).

```
using optix::float2;
using optix::float3;
using optix::float4;
using optix::Ray;
using optix::size_t2;
using optix::uint2;
```

```
struct PerRayDataRT {
  float3 result;
  int depth;
};

rtDeclareVariable(float3, eye, , );
rtDeclareVariable(float3, u, , );
rtDeclareVariable(float3, v, , );
rtDeclareVariable(float3, w, , );

rtDeclareVariable(uint2, launch_index, rtLaunchIndex, );
rtBuffer<float4, 2> output_buffer;

rtDeclareVariable(rtObject, top_obj, , );
rtDeclareVariable(float, scene_eps, , );

RT_PROGRAM void pinholeCamera() {
  size_t2 screen = output_buffer.size();
  float2 d = 2 * (make_float2(launch_index)) /
             make_float2(screen) - 1;
  float3 ray_direction(d.x * u + d.y * v + w);

  Ray ray(eye, ray_direction, /*radiance ray type*/ 0, scene_eps);
  PerRayDataRT radiance_prd;
  radiance_prd.depth = 0;

  rtTrace(top_obj, ray, radiance_prd);

  output_buffer[launch_index] = make_float4(
      radiance_prd.result,
      1.0f);
}
```

4.2 Ray Tracing

O algoritmo de *ray-tracing* implementado computa uma solução numérica para a equação de iluminação semelhante a que é proposta em (WHITTED, 1979). A tabela 1 lista o que cada um dos termos das equações a seguir significam. A intensidade de luz I computada é

$$I = I_{amb} + \sum_{j=1}^{N} f_r^{(j)}(x, \mathbf{v_i}, \mathbf{v_o}) + r \cdot S + (1 - r) \cdot T, \tag{4.1}$$

onde $f_r^{(j)}(x, \mathbf{v_i}, \mathbf{v_o})$ é a BRDF de Blinn-Phong (BLINN, 1977) definida para uma determinada fonte de luz L_j:

$$f_r^{(j)}(x, \mathbf{v_i}, \mathbf{v_o}) = k_d \cdot \langle \mathbf{n}_x, \mathbf{l}_{x,j} \rangle + k_s \cdot \langle \mathbf{n}_x, \mathbf{h} \rangle, \quad \mathbf{h} = \frac{\mathbf{v_i} + \mathbf{v_o}}{||\mathbf{v_i} + \mathbf{v_o}||} \tag{4.2}$$

O cálculo da reflectância r é dado através da aproximação de Schlick (SCHLICK, 1994) para as *equações de Fresnel*:

$$r = R_0 + (1 - R_0)(1 - \langle \mathbf{n}_x, \mathbf{v_o} \rangle)^5, \quad R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2} \right)^2. \tag{4.3}$$

Tabela 1 – Lista de termos das equações 4.1, 4.2 e 4.3.

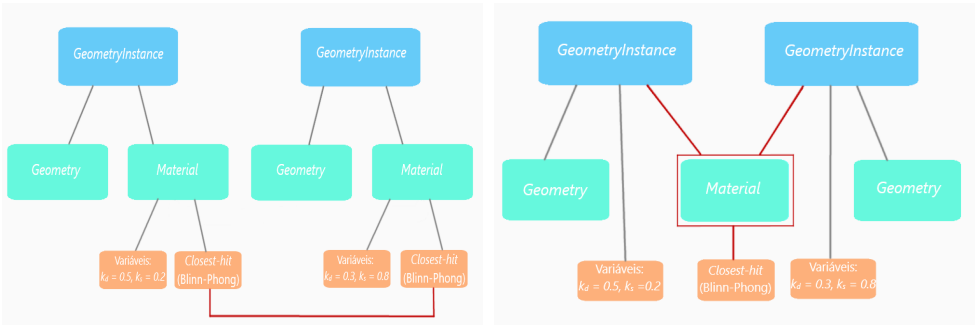
Termos	Descrição
I	Intensidade total da luz
I_{amb}	Intensidade da luz ambiente
N	Número de fontes de luz
$\mathbf{v_i}$	Direção do raio incidente
$\mathbf{v_o}$	Direção da trajetória do raio até o observador
r	Reflectância
S	Componente indireta da luz (reflexão)
T	Componente indireta da luz (refração)
$\mathbf{n}_x$	Vetor unitário normal à superfície no ponto x
$\mathbf{l}_{x,j}$	Direção do raio originado no ponto x até a fonte de luz L_j
$\mathbf{h}$	Vetor unitário paralelo ao vetor bissetor de $\mathbf{v_i}$ e $\mathbf{v_o}$
R_0	Coeficiente de reflexão com incidência de luz na direção normal da superfície
n_1, n_2	Coeficientes de refração dos meios externo e interno, respectivamente

4.3 Implementação

Um dos principais problemas enfrentados no início do processo de implementação incluía a escalabilidade sofrível no caso de um leve aumento no número de partículas na cena. De fato, a quantidade de memória consumida ultrapassava a ordem de 1 GB para cerca de $1 \cdot 10^4$ partículas. Era necessário, portanto, um maior cuidado ao usar o *framework* da aplicação. Listamos abaixo algumas boas práticas adotadas para aumentar o desempenho do método implementado.

Como mencionado na Seção 4.1, uma das etapas da criação do contexto envolve sua compilação. Em particular, é necessário compilar cada um dos programas do OptiX criados no contexto. Eventualmente, um programa pode ser utilizado para vários objetos na cena. Definir uma instância deste programa para cada objeto que o utiliza gera uma perda de desempenho desnecessária, pois cada uma dessas instâncias será compilada separadamente, embora o resultado final seja idêntico para todas elas, além do consumo excessivo de memória alocada para

cada instância. Em outras palavras, é fundamental que uma instância de um determinado programa seja *compartilhada* entre todos os objetos que a utilizem. A Figura 5 ilustra o cenário que acabamos de descrever.



(a) (b)

Figura 5 – Na Figura 5a, as duas instâncias geométricas utilizam o mesmo programa *closest-hit*, mas que possui dois objetos distintos alocados na memória, o que é desnecessário. As instâncias geométricas, portanto, devem compartilhar a mesma instância do material, removendo a redundância tal qual mostra a Figura 5b.

Um exemplo para esta situação foi encontrado na fase inicial de implementação, e é ilustrada no programa 4.2. Uma maneira de modelar cada partícula é interpretá-la como uma instância geométrica singular. Cada instância geométrica possui, em termos gerais, uma *geometria* e um *material* associados. No entanto, uma partícula que compõe a representação discretizada de uma superfície compartilha as mesmas propriedades em termos de geometria e de material com todas as outras partículas da superfície. Uma partícula é, portanto, *primitiva* da superfície e deve ser encapsulada dentro de uma instância geométrica comum a todas as partículas da superfície. Por simplicidade, o modelo geométrico adotado para a partícula é de uma esfera.

Programa 4.2 – Duas modelagens distintas de uma partícula: a primeira, ineficiente, assume que cada partícula é uma instância geométrica; a segunda, por sua vez, encapsula, de fato, a partícula como uma primitiva da instância geométrica (a superfície).

```
// --------------
// #    ERRADO    #
// --------------
std::vector<GeometryInstance> particles;

// (...) Para cada particula a ser adicionada, criar uma instancia.

Geometry particle = context->createGeometry();
particle->setBoundingBoxProgram(bounding_program);
particle->setIntersectionProgram(intersection_program);
particle["center"]->setFloat(make_float3(...));
```

```
GeometryInstance instance = context->createGeometryInstance();
instance->setGeometry(particle);
instance->setMaterial(material);

particles.push_back(instance);

// --------------
// #  CORRETO   #
// --------------
std::vector<float3> positions;

// (...) Preenche o vetor com as posicoes das particulas.

Geometry particles = context->createGeometry();
particles->setPrimitiveCount(
    static_cast<unsigned int>(positions.size()));
particles->setBoundingBoxProgram(bounding_program);
particles->setIntersectionProgram(intersection_program);

Buffer buffer = context->createBuffer(
    RT_BUFFER_INPUT,
    RT_FORMAT_FLOAT3);
buffer->setSize(static_cast<RTsize>(positions.size()));
memcpy(buffer->map(), reinterpret_cast<const void*>(&positions[0]),
       positions.size() * sizeof(float3));
buffer->unmap();

GeometryInstance surface = context->createGeometryInstance();
surface["centers"]->setBuffer(buffer);
surface->setGeometry(particles);
surface->addMaterial(material);
```

Outro ponto a ser mencionado está relacionado ao caráter recursivo do *pipeline* do OptiX. Para simular as chamadas recursivas do algoritmo de *ray tracing*, o OptiX usa uma pequena pilha de memória para cada *thread* de execução, cujo tamanho é salvo no contexto da aplicação. Naturalmente, para aplicações de *ray tracing*, o tamanho da pilha não pode ser muito pequeno, visto que isso tornaria a aplicação suscetível a estouro da pilha (*stack overflow*). Problemas de estouro são tratados por *programas de exceção* (*exception programs*) (NVIDIA CORPORATION, 2012). Deve-se, portanto, escolher um tamanho de pilha com extremo cuidado para que exista um equilíbrio entre desempenho e estabilidade da aplicação. Neste trabalho, o tamanho escolhido da pilha de recursão é de 1500 bytes.

5 Metodologia

Neste capítulo iremos descrever o processo de análise de desempenho da aplicação. O passo-a-passo do plano de experimentação é descrito na Seção 5.1, enquanto a Seção 5.2 apresenta e interpreta os resultados obtidos. Os testes foram executados numa máquina com as seguintes configurações: processador Intel Core i7-7700HQ de 2,8GHz, 16GB de memória RAM, placa de vídeo NVIDIA Geforce GTX 1050Ti com 4GB de memória dedicada e sistema operacional Windows 10.

5.1 Experimentos

Duas variáveis são avaliadas para determinar o desempenho da aplicação: a taxa de quadros (*frames*) por segundo, e a quantidade de memória consumida. A qualidade visual da cena também é levada em consideração. Os testes consistem em uma cena contendo um paralelepípedo delimitado pelos pontos $(x_{min}, y_{min}, z_{min}) = (3,5,-2)$ e $(x_{max}, y_{max}, z_{max}) = (7,8,2)$. A câmera que representa o observador da cena possui a seguinte configuração inicial: $eye = (0,0,0)$, $lookat = (5,5,0)$ e $up = (0,0,1)$. A resolução das imagens é em *full HD* (1920×1080). O nível (profundidade) máximo atribuído para o algoritmo de *ray tracing* nos testes foi $max\_depth = 3$. Os parâmetros variados são a quantidade de partículas na cena e as estruturas de aceleração. Os valores testados são descritos abaixo:

- *Quantidade de partículas.* $1 \cdot 10^5$, $1 \cdot 10^6$ e $1 \cdot 10^7$ partículas.
- *Estruturas de aceleração.* BVH, LBVH, SBVH e *k-d tree*.

São amostrados *frames* por um intervalo de 1 minuto. Durante a amostragem, a câmera é movimentada a fim de variar o número de partículas visíveis na cena. Foram testados três padrões de deslocamento da câmera: transformações de rotação, translação, e composições dos dois tipos anteriores. Calcula-se, por fim, a média e o desvio padrão da taxa de *frames* em Hz destas amostras para avaliar o desempenho da aplicação. A quantidade máxima de memória exigida pela aplicação também é computada.

5.2 Resultados

A ideia do primeiro experimento (apenas rotacionando a câmera) consiste em verificar a eficiência das estruturas de aceleração diante de condições de visibilidade semelhantes. A tabela 2 mostra que a LBVH se apresentou levemente mais eficiente que as demais, visto que todas elas se comportaram de maneira semelhante após a sequência de rotações aplicadas à

câmera: de fato, a variação entre os desvios padrões da taxa de FPS não é significativa em relação à ordem de grandeza da média. Também foi medido o tempo de pré-processamento das estruturas de aceleração: novamente, a LBVH obteve a melhor performance, juntamente com a *k-d tree* (aproximadamente 2,5s). Já a SBVH foi a estrutura que demandou um maior custo nessa etapa (mais de 1min20s).

Tabela 2 – FPS (média e desvio padrão) para uma cena com 10 milhões de partículas (apenas rotação).

	LBVH	BVH	SBVH	*k-d tree*
Média	**13,3905**	13,0577	13,2272	12,4552
Desvio padrão	2,78308	2,72863	2,91955	**2,62737**

Tabela 3 – FPS (média e desvio padrão) para uma cena com 10 milhões de partículas (apenas translação).

	LBVH	BVH	SBVH	*k-d tree*
Média	36,1986	37,5214	34,5066	**38,3809**
Desvio padrão	23,2129	**22,6495**	24,152	24,7774

Tabela 4 – FPS (média e desvio padrão) para uma cena com 10 milhões de partículas (rotação e translação).

	LBVH	BVH	SBVH	*k-d tree*
Média	17,8845	15,0589	**23,6186**	17,9082
Desvio padrão	9,27187	**8,36336**	18,5089	10,5496

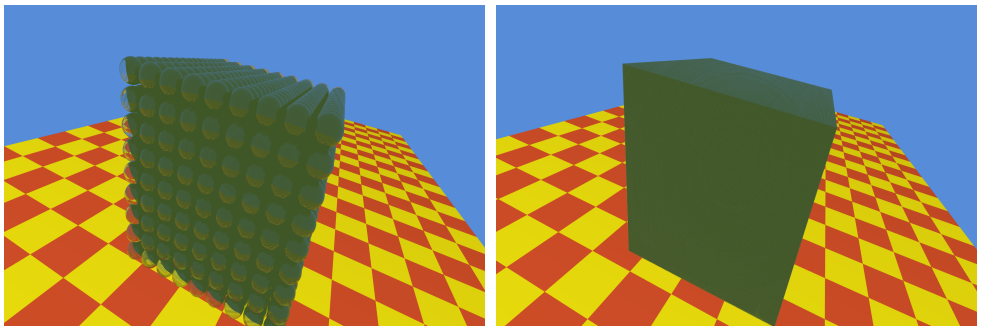
Tabela 5 – Quantidade máxima de memória consumida pela aplicação em MB.

Número de Partículas	LBVH	BVH	SBVH	*k-d tree*
$1 \cdot 10^5$ partículas	524,4	531,7	542,6	529,1
$1 \cdot 10^6$ partículas	808	799,7	900,3	759,1
$1 \cdot 10^7$ partículas	3481,6	3584	4403,2	3481,6

Já no caso da translação, o efeito esperado era de uma maior variação na taxa de FPS, pois o afastamento ou a aproximação da câmera da superfície renderizada resultaria na diminuição ou no aumento do número de regiões na cena em que há a presença ou não de uma partícula a ser processada. De fato, a ordem de grandeza dos desvios padrões é igual à da média de FPS, como mostra a tabela 3. O mesmo comportamento pode ser inferido da tabela 4, o que indica que, em um cenário com a participação do usuário da aplicação, a mudança de performance é perceptível.

Uma análise da escalabilidade e do custo de memória da aplicação também foi realizada. Mesmo utilizando estruturas de baixa complexidade de espaço, como a LBVH e a *k-d tree*, eram necessários mais de 500MB para uma simples cena com $1 \cdot 10^5$ partículas, chegando aos 3,4GB

com a superfície de $1 \cdot 10^7$ partículas com essas mesmas estruturas. A tabela 5 detalha os valores obtidos para as configurações testadas. É necessária, portanto, uma representação mais eficiente das partículas, visto que a variação da quantidade de memória exigida pela aplicação ao se alterar a estrutura de aceleração é desprezível em comparação à diferença de consumo devido ao aumento do número de partículas.



(a) (b)

Figura 6 – Utilizando a modelagem implementada neste trabalho, os raios de luz dificilmente atingem o interior de superfícies de material transparente representadas por uma nuvem de partículas densa. Na Figura 6a, percebe-se que a única parte visivelmente transparente do objeto é a sua fronteira. Quanto menor o tamanho da partícula, mais difícil perceber essa propriedade do material, como se pode perceber na Figura 6b.

Outro aspecto que está relacionado com a representação adotada para as partículas é a qualidade visual da cena. Superfícies modeladas como uma nuvem de partículas densa necessitam de uma estrutura que otimize a visualização das partículas delimitadas pela fronteira do objeto, pois é necessário um nível máximo de recursão bastante elevado para que o algoritmo de *ray tracing* alcance partículas mais internas, por exemplo. Podemos perceber na Figura 6 que apesar do material transparente da superfície, os raios de luz mal conseguem alcançar as partículas imediatamente adjacentes à fronteira do paralelepípedo renderizado.

6 Conclusão

Apresentamos neste trabalho uma aplicação para síntese de superfícies modeladas a partir de uma nuvem de partículas capaz de renderizar cenas consideravelmente complexas no que diz respeito ao tamanho do conjunto de partículas. O método implementado também foi capaz de atingir taxas de *frames* por segundo que permitem uma interação razoável do usuário com a ferramenta (mais de 10Hz em média). Isso mostra o potencial do OptiX como *framework* para aplicações desse tipo, pois a modelagem da cena priorizou simplicidade em vez de otimizações mais complexas.

A falta de documentação mais simples para aqueles que utilizam o OptiX pela primeira vez, entretanto, dificulta o aprendizado e a adoção de boas práticas que maximizam a performance da aplicação. Tendo em vista que o OptiX é um *framework* com alta relevância por se tratar do estado-da-arte em *ray tracing*, é necessário torná-lo mais acessível. Este trabalho também descreve algumas técnicas que permitem a otimização do desempenho de programas que utilizem esta API.

6.1 Trabalhos Futuros

As limitações da aplicação, como o consumo elevado de memória e a qualidade visual da cena, são problemas amplamente abordados na literatura. Por exemplo, (REICHL et al., 2014) utiliza uma representação binária de um voxel para otimizar a determinação da superfície visível e reduzir a complexidade de espaço do método de renderização, além de filtrar o mapa de profundidade da cena para suavizar a superfície resultante, em vez de representá-la como um conjunto de esferas. Ainda em relação à reconstrução da superfície, (XIAO; ZHANG; YANG, 2017) propõe uma estimativa das normais em cada ponto através da Análise de Componentes Principais (*Principal Component Analysis* - PCA).

Outra questão a ser abordada diz respeito à qualidade visual da cena, em particular utilizando métodos baseados em *ray tracing* que aproveitem o alto desempenho de ferramentas como o OptiX. Uma possível solução é aproximar a fronteira da superfície a um polinômio ou a modelos resultantes de métodos de subdivisão e tratar os componentes conexos do volume delimitado pela fronteira como uma única instância geométrica. Desta maneira, a propagação da luz na região interna da superfície não precisaria do cálculo extensivo de raios secundários. A complexidade do problema se resumiria então a determinar um método eficiente para computar a interseção do modelo aproximado com um raio.

Referências

AAA STUDIO. *FurryBall RT - Incredibly fast GPU render*. 2015. Disponível em: <http://furryball.aaa-studio.eu/>. Acesso em: 29 nov. 2017.

BENTLEY, J. L. Multidimensional binary search trees used for associative searching. *Commun. ACM*, ACM, Nova York, NY, EUA, v. 18, n. 9, p. 509–517, 1975. ISSN 0001-0782.

BLINN, J. F. Models of light reflection for computer synthesized pictures. *ACM SIGGRAPH Computer Graphics*, ACM, New York, NY, USA, v. 11, n. 2, p. 192–198, 1977.

DUTRE, P. et al. *Advanced Global Illumination*. [S.l.]: AK Peters Ltd, 2006. ISBN 1568813074.

GARANZHA, K.; PANTALEONI, J.; MCALLISTER, D. Simple and faster hlbvh with work queues. *ACM SIGGRAPH Symposium on High Performance Graphics*, p. 59–64, 2011.

GOMES, J.; VELHO, L. *Fundamentos de Computação Gráfica*. [S.l.]: IMPA, 2008.

JUDICE, S. F.; GIRALDI, G. A.; FILHO, J. K. *A Equação de Iluminação em Computação Gráfica*. Petrópolis, RJ, Brasil, 2011.

KAJIYA, J. T. The rendering equation. *ACM SIGGRAPH Computer Graphics*, ACM, Nova York, NY, EUA, n. 4, p. 143–150, 1986.

KAY, T. L.; KAJIYA, J. T. Ray tracing complex scenes. *ACM SIGGRAPH Computer Graphics*, ACM, Nova York, NY, EUA, n. 4, p. 269–278, 1986.

LAUTERBACH, C. et al. Fast bvh construction on gpus. *Eurographics*, n. 2, 2009.

NIH CENTER FOR MACROMOLECULAR MODELING AND BIOINFORMATICS (UNIVERSITY OF ILLINOIS). *VMD - Visual Molecular Dynamics*. 2006–2017. Disponível em: <http://www.ks.uiuc.edu/Research/vmd/>. Acesso em: 29 nov. 2017.

NVIDIA CORPORATION. *NVIDIA OptiX Ray Tracing Engine - Programming Guide (Version 3.0)*. 2012. Disponível em: <http://developer.download.nvidia.com/assets/tools/files/optix/3.0.0/NVIDIA-OptiX-SDK-3.0.0-OptiX_Programming_Guide_3.0.0.pdf>. Acesso em: 26 nov. 2017.

NVIDIA CORPORATION. *OptiX QuickStart*. 2014–2017. Disponível em: <https://docs.nvidia.com/gameworks/content/gameworkslibrary/optix/optix_quickstart.htm>. Acesso em: 17 nov. 2017.

NVIDIA CORPORATION. *NVIDIA IRAY SDK | NVIDIA Developer*. 2017. Disponível em: <https://developer.nvidia.com/iray-sdk>. Acesso em: 29 nov. 2017.

PARKER, S. G. et al. Optix: A general purpose ray tracing engine. *ACM Transactions on Graphics*, Agosto 2010.

PHAAR, M.; HUMPHREYS, G. *Physically Based Rendering*. 1. ed. [S.l.]: Morgan Kauffman Publishers Inc., 2004.

REICHL, F. et al. Interactive rendering of giga-particle fluid simulations. *Proceedings of High Performace Graphics*, p. 105–116, 2014.

SCHLICK, C. An inexpensive brdf model for physically-based rendering. *Eurographics*, v. 13, n. 3, p. 233–246, 1994.

SHIRLEY, P.; MARSCHNER, S. *Fundamentals of Computer Graphics*. 3. ed. Natick, MA, EUA: A. K. Peters, Ltd., 2009. ISBN 1568814690, 9781568814698.

STICH, M.; FRIEDRICH, H.; DIETRICH, A. Spatial splits in bounding volume hierarchies. *ACM SIGGRAPH Conference on High Performance Graphics*, ACM, Nova York, NY, EUA, p. 7–13, 2009.

WHITTED, T. An improved illumination model fo shaded display. *ACM SIGGRAPH Computer Graphics*, n. 6, p. 343–349, 1979.

XIAO, X.; ZHANG, S.; YANG, X. Real-time high-quality surface rendering for large scale particle-based fluids. *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, Nova York, NY, EUA, p. 12:1–12:8, 2017.