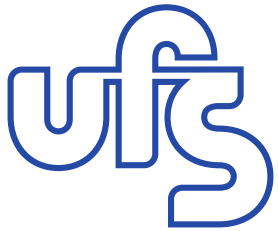


Compilador de Funções de Distribuição de Refletância Bidirecional descritas em LaTeX para Linguagem de Shading

Everton Santos de Andrade Júnior
Orientador(a): Dra. Beatriz Trinchão Andrade

Universidade Federal de Sergipe

Dez / 2024

Sumário

Contexto

- Na **computação gráfica**, a representação realista de cenas tridimensionais depende da modelagem da interação entre a luz e os materiais que compõem os objetos.
- Profissionais e pesquisadores modelam essa interação por meio das **funções de distribuição de refletância bidirecional** (BRDFs) [1].
- BRDFs são implementadas em programas especializados que rodam na GPU.

[1] Do inglês, *Bidirectional Reflectance Distribution Functions*. Referência: [PJH16]

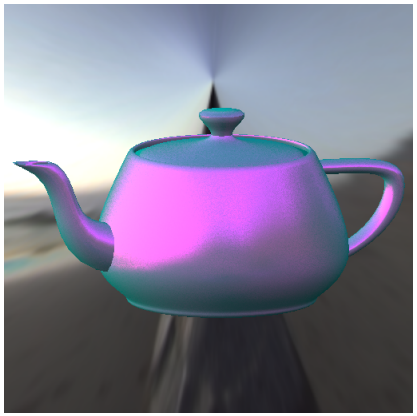
Contexto - BRDF Blinn-Phong

$$\rho_d = 0, \vec{1}, 1 \quad (1)$$

$$\rho_s = 1, \vec{0}, 1 \quad (2)$$

$$n = +2^8 \quad (3)$$

$$f = \frac{\rho_d}{\pi} + \rho_s * \frac{n+2}{2*\pi} * \cos\theta_h{}^n \quad (4)$$



Contexto - BRDF Cook-Torrance

$$m = 0.3 \tag{1}$$

$$f_0 = 0.4 \tag{2}$$

$$Beckmann(m,t) = \exp((t*t-1)/(m*m*t*t))/(m*m*t*t*t*t) \tag{3}$$

$$Fresnel(f_0,u) = f_0 + (1-f_0)*((1-u)^5) \tag{4}$$

$$H = \vec{h} \tag{5}$$

$$V = \vec{\omega_o} \tag{6}$$

$$L = \vec{\omega_i} \tag{7}$$

$$N = \vec{n} \tag{8}$$

$$D = Beckmann(m,(N\cdot H)) \tag{9}$$

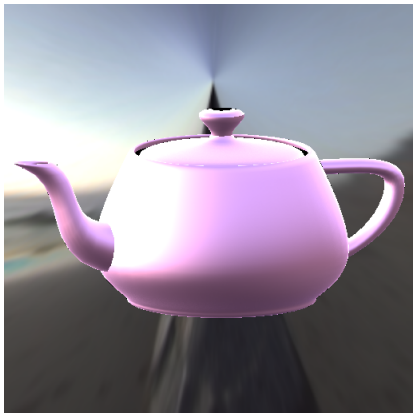
$$F = Fresnel(f_0,(V\cdot H)) \tag{10}$$

$$G = 1/(N\cdot V) \tag{11}$$

$$val = \max(D*G,0.0)\cdot F \tag{12}$$

$$\text{color} = 1,0.\vec{5},1 \tag{13}$$

$$f = \text{color}*val/(N\cdot L) \tag{14}$$



Contexto - *shaders*

- *Shaders* concedem a capacidade de cada objeto renderizado ter sua **aparência configurada por meio de um código** que implementa uma BRDF.

Motivação

Na renderização, as BRDFs são implementadas por meio de programas chamados de ***shaders***. Mas, as BRDFs são comumente descritas por equações em LaTeX[2]. Dois problemas surgem com essa modelagem:

1. Barreira técnica[3]: visualizar essa modelagem de BRDFs requer conhecimento especializado em programação em linguagem de *shading*.
2. Baixo nível de iteração: toda mudança nas equações exige baixar o nível e reescrever o *shader* novamente.

[2] LaTeX é um sistema de preparação de documentos para alta qualidade tipográfica. Assim como visto no início.

[3] Físicos ou matemático podem trabalhar com BRDFs e não, necessárimente saber programação baixo nível

Motivação

Surge a necessidade de **simplificar** a **criação de shaders para BRDFs**.

Um **compilador** capaz de traduzir BRDFs escritas em LaTeX para *shaders* permitiria uma maior acessibilidade e agilidade na criação de efeitos visuais complexos.

$$\rho_d = 0, \vec{1}, 1$$

$$\rho_s = 1, \vec{0}, 1$$

$$n = +2^8$$

$$f = \frac{\rho_d}{\pi} + \rho_s * \frac{n+2}{2*\pi} * \cos\theta_h{}^n$$

diretamente

Objetivo

Projetar e implementar um **compilador** capaz de:

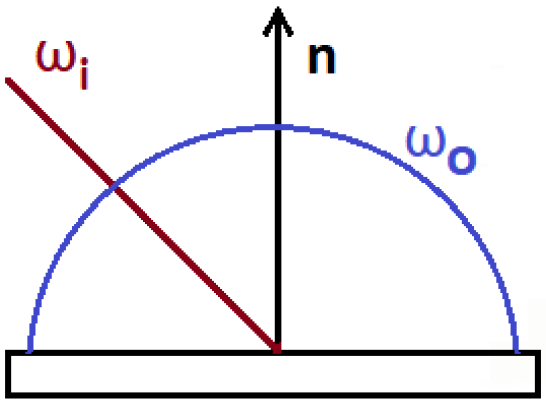
1. Processar **BRDFs** descritas em equações LaTeX.
2. Gerar **código de *shading*** na linguagem-alvo da API **OpenGL**.
3. Visualizar o efeito dessas BRDFs usando o **código GLSL** gerado.

Resultado esperado é um **shader** que reproduza as características de refletância da BRDF original.

Conceitos - BRDFs

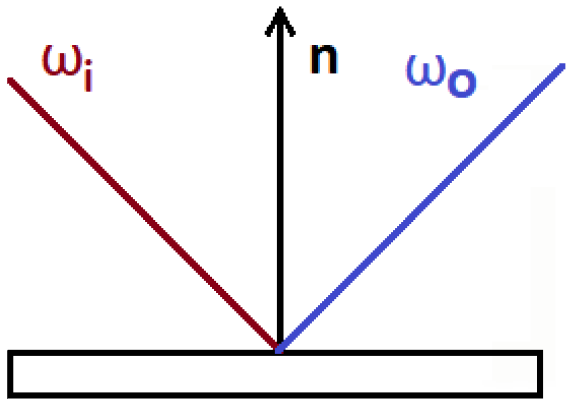
As BRDFs calculam a proporção entre a energia luminosa que atinge um ponto na superfície e como essa energia é:

- **Refletida**
- **Transmitida**
- **Absorvida**



$$f(\omega_i, \omega_o) = \frac{\rho_d}{\pi} \cdot \cos\theta$$

BRDF Difusa Ideal



$$f(\omega_i, \omega_o) = k_s \cdot \delta(\omega_i - \omega_o)$$

BRDF Pura Especular

Equação de Renderização

Um renderizador estima a "quantidade" luz L_o que sai de um ponto em uma direção ω_o usando a equação de renderização ([Kaj86]). Nela, a BRDF é encontrada:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{H^2} f(p, \omega_i, \omega_o) L_i(p, \omega_i) \cos(\theta_i) d\omega_i \tag{1}$$

L_o é radiância de saída (*outgoing*)

L_e é radiância emitida pela superfície (i.e. fonte de luz)

L_i é radiância incidente na superfície

ω_i é a direção do raio incidente

ω_o é a direção do raio refletido

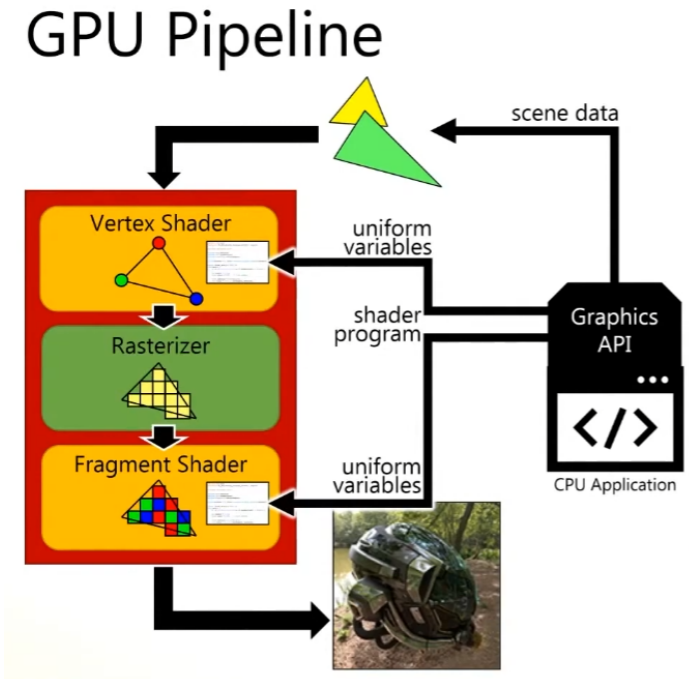
f função de refletância

Pipeline de GPU

Etapas Principais:

1. ***Shader* de Vértice**: Processa e transforma vértices.
2. **Rasterização**: Gera fragmentos a partir de primitivas.
3. ***Shader* de Fragmento**: Determina a cor final dos fragmentos.

Fluxo de dados são: CPU → GPU → *Shaders* → Imagem final.



Shader de Vértice

- Aplicação de transformações (ex.: rotação, projeção).
- Transmissão de dados (ex.: normais) ao *shader* de fragmento.

```
#version 330 core
layout(location = 0) in vec3 inPosition;
layout(location = 1) in vec3 inNormal;

uniform mat4 modelViewProjection;

out vec3 fragNormal;

void main() {
    vec3 manipulatedPosition = inPosition + (sin(gl_VertexID * 0.1) * 0.1);
    fragNormal = inNormal;
    gl_Position = modelViewProjection * vec4(manipulatedPosition, 1.0);
}

```

Shader de Fragmento

Nesse *shader* é onde a cor final é calculada. Roda paralelamente 1 vez para cada "pixel".

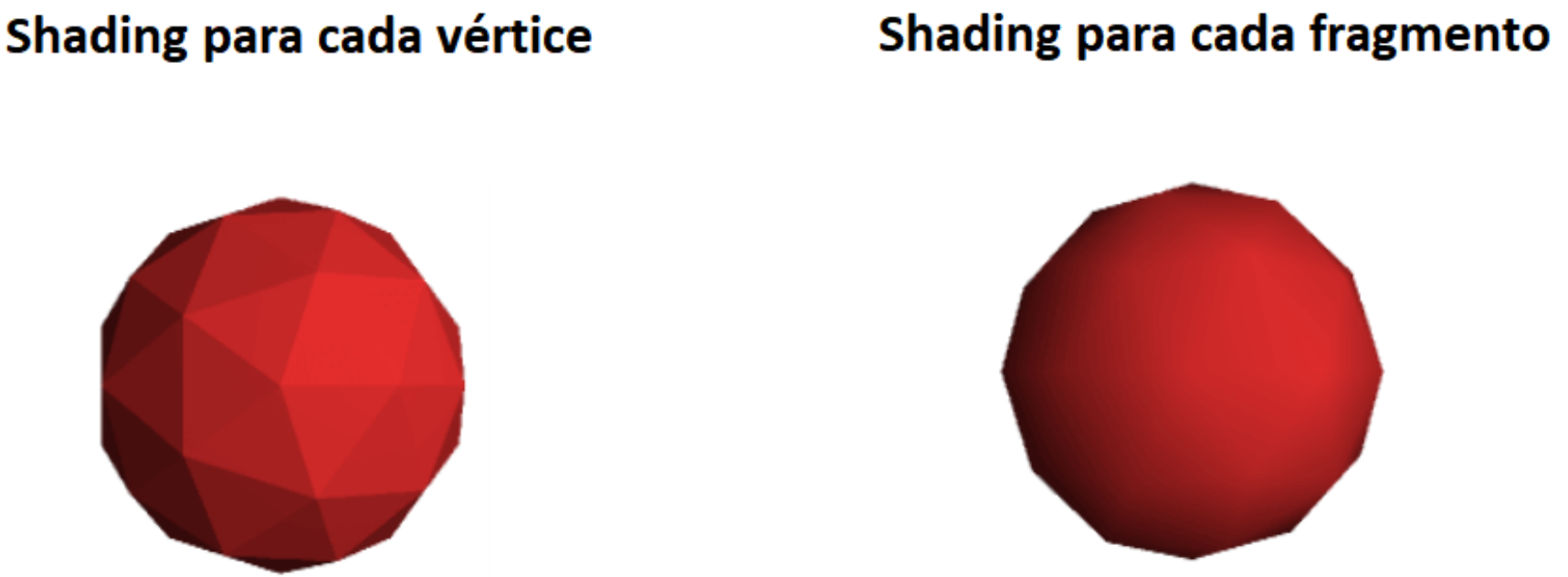


Figure: Diferença entre *shading* por vértice e por fragmento.

Compilação: Visão Geral

Comilador $C : L_1 \rightarrow L_2$ mapeia programa de L_1 para L_2 preservando semântica; mantém mesmo significado algorítmico.

1. **Análise Léxica (Lexing):**
   - Divide entrada em tokens (palavras-chave, identificadores)
   - Reconhecível por máquinas de estado
2. **Análise Sintática (Parsing):**
   - Valida tokens segundo gramática
   - Constrói árvore sintática hierárquica
3. **Verificação de Tipos:**
   - Garante consistência de tipos
   - Previne erros em tempo de execução
4. **Emissão de Código:**
   - Gera código na linguagem alvo

Desenvolvimento - Arquitetura do Compilador

- Organização modular para encapsulamento de responsabilidades:
  - `lexer`: Análise léxica e tokenização.
  - `parser`: Construção da AST com *Pratt Parsing*.
  - `walker`: Navegação e análise da AST.
  - `checker`: Inferência de tipos e validações semânticas.
  - `emitter`: Geração do código GLSL.

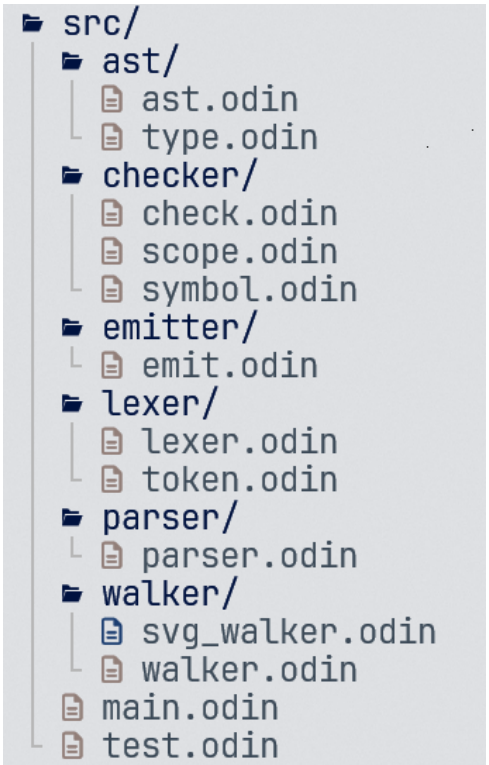


Figure: Estrutura de pacotes do compilador.

Arquitetura do Compilador

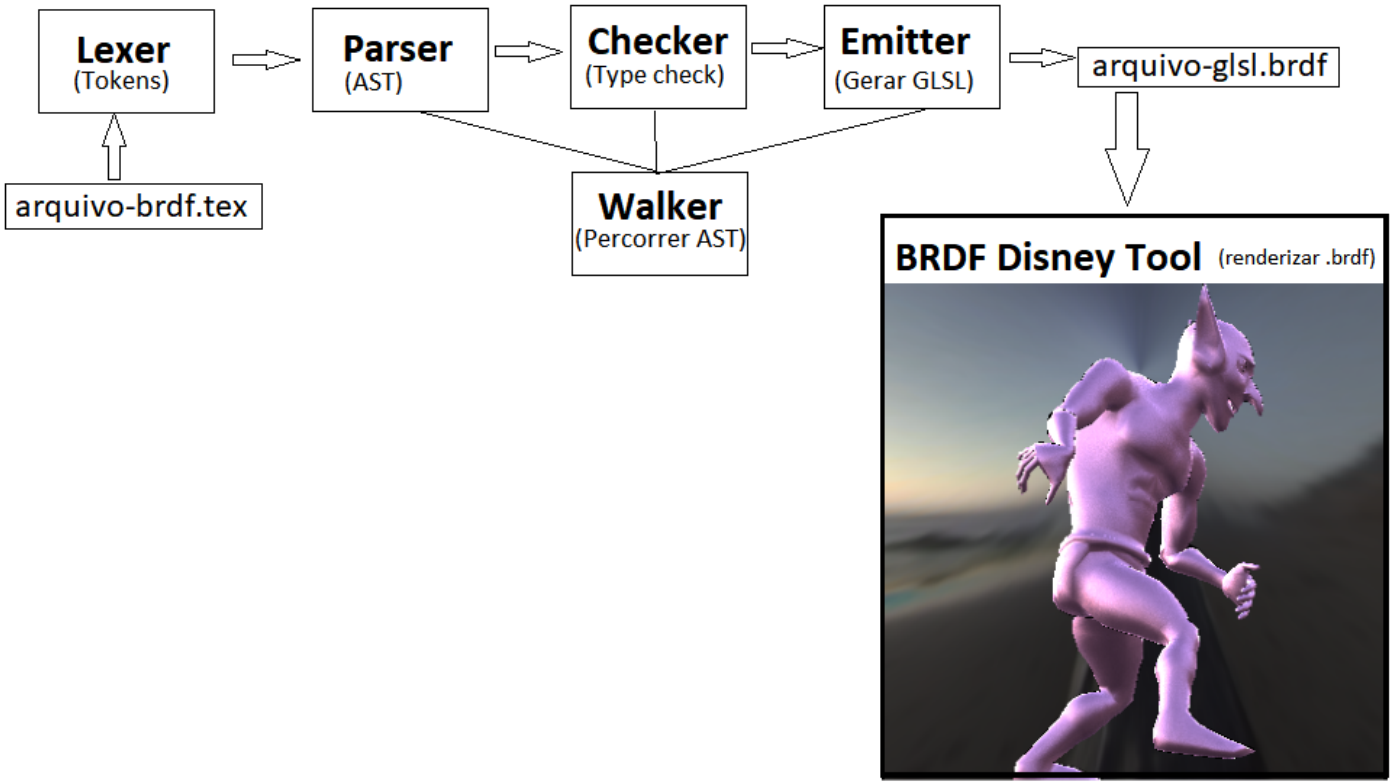


Figure: Estrutura geral da arquitetura do compilador.

Destaques do Desenvolvimento

- Análise sintática baseada em *Pratt Parsing*, garantindo precisão e hierarquia nas expressões.
- Funcionalidades do `walker`:
  - Navegação genérica e preparação para verificações.
  - Suporte uniforme à travessia de nós.
- Integração do `checker` com a tabela de símbolos para validações semânticas e consistência.
- Geração de *shaders* GLSL compatíveis com Disney BRDF Explorer.

Resultados

- Experimentos com diferentes BRDFs para validar o compilador
- Metodologia padronizada:
  - Código fonte em é o ambiente `equation` do LaTeX
  - Tradução para GLSL pelo compilador
  - Visualização no Disney BRDF Explorer
- Condições controladas:
  - Ângulos de luz fixos ($\theta_i = 33.89$, $\phi_i = 145.83$)
  - Gamma = 2.112, Exposição = -1.248

BRDFs Implementadas

11 Experimentos Realizados:

- Modelos Clássicos: Blinn-Phong, Cook-Torrance, Ward
- Modelos Avançados: Ashikhmin-Shirley, Oren-Nayar
- Variações: Cook-Torrance$_2$, Ashikhmin-Shirley$_2$
- Especializados: Dür, Edwards-2006, Kajiya-Kay-1989, Minnaert

Cada experimento mostra:

- Gráficos 3D e 2D de distribuição
- Renderização de objetos 3D

Equações da BRDF do experimento Blinn-Phong em documento LaTeX

$$\rho_d = 0, \vec{1}, 1 \tag{1}$$

$$\rho_s = 1, \vec{0}, 1 \tag{2}$$

$$n = +2^8 \tag{3}$$

$$f = \frac{\rho_d}{\pi} + \rho_s * \frac{n+2}{2*\pi} * \cos\theta_h{}^n \tag{4}$$

Código fonte da BRDF do experimento Blinn-Phong

```
\begin{equation}
    \rho_{d} = \vec{0,1,1}
\end{equation}

\begin{equation}
    \rho_{s} = \vec{1,0,1}
\end{equation}

\begin{equation}
    n = +2^8
\end{equation}

\begin{equation}
f = \frac{\rho_{d}}{\pi} + \rho_{s} * \frac{n+2}{2*\pi} *
\cos{\theta_{h}}^{n}
\end{equation}
```

Código GLSL da BRDF do experimento Blinn-Phong (1 de 2).

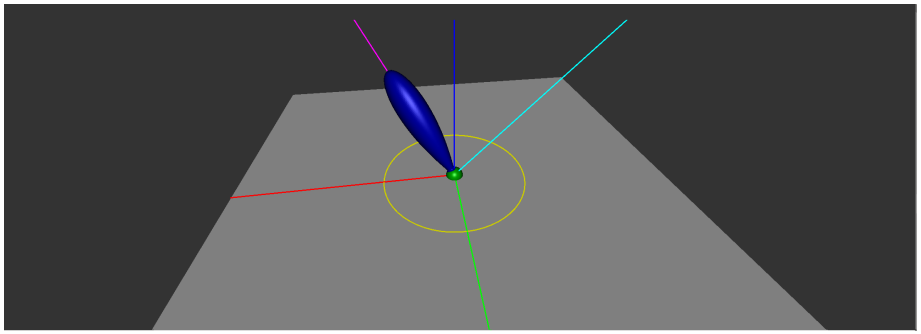
```
analytic ::begin parameters
#[type][name][min val][max val][default val]
::end parameters
::begin shader
////////////// START OF BUILTINS DECLARTION //////////////
vec3 var_0_vec_h;
vec3 var_3_vec_n;
float var_10_theta_h;
float var_11_theta_d;
float var_1_pi;
float var_2_epsilon;
vec3 var_4_vec_omega_i;
float var_5_theta_i;
float var_6_phi_i;
vec3 var_7_vec_omega_o;
float var_8_theta_o;
float var_9_phi_o;
////////////// END OF BUILTINS DECLARTION //////////////
////////////// START OF USER DECLARED //////////////
vec3 var_12_rho_s;
float var_13_n;
vec3 var_14_rho_d;
vec3 var_15_f;
////////////// END OF USER DECLARED //////////////
```

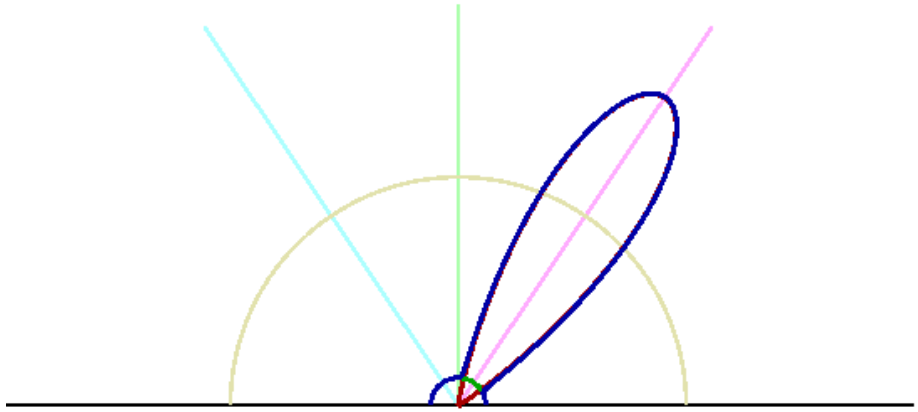
Código GLSL da BRDF do experimento Blinn-Phong (2 de 2).

```
vec3 BRDF(vec3 L, vec3 V, vec3 N, vec3 X, vec3 Y) {
  /////////////// START OF BUILTINS INITIALIZATION ///////////////
  var_0_vec_h = normalize(L + V);
  var_3_vec_n = normalize(N);
  var_1_pi = 3.141592653589793;
  var_2_epsilon = 1.192092896e-07;
  var_4_vec_omega_i = L;
  var_5_theta_i = atan(var_4_vec_omega_i.y, var_4_vec_omega_i.x);
  var_6_phi_i = atan(sqrt(var_4_vec_omega_i.y * var_4_vec_omega_i.y +
                          var_4_vec_omega_i.x * var_4_vec_omega_i.x),
                     var_4_vec_omega_i.z);
  var_7_vec_omega_o = V;
  var_8_theta_o = atan(var_7_vec_omega_o.y, var_7_vec_omega_o.x);
  var_9_phi_o = atan(sqrt(var_7_vec_omega_o.y * var_7_vec_omega_o.y +
                          var_7_vec_omega_o.x * var_7_vec_omega_o.x),
                     var_7_vec_omega_o.z);
  var_10_theta_h = acos(dot(var_0_vec_h, N));
  var_11_theta_d = acos(dot(var_0_vec_h, var_4_vec_omega_i));
  /////////////// END OF BUILTINS INITIALIZATION ///////////////
  var_12_rho_s = vec3(1.0, 0.0, 1.0);
  var_13_n = pow(2.0, 8.0);
  var_14_rho_d = vec3(0.0, 1.0, 1.0);
  var_15_f = ((var_14_rho_d / var_1_pi) +
              ((var_12_rho_s * ((var_13_n + 2.0) / (2.0 * var_1_pi))) *
               pow(cos(var_10_theta_h), var_13_n)));
  return vec3(var_15_f);
}
```

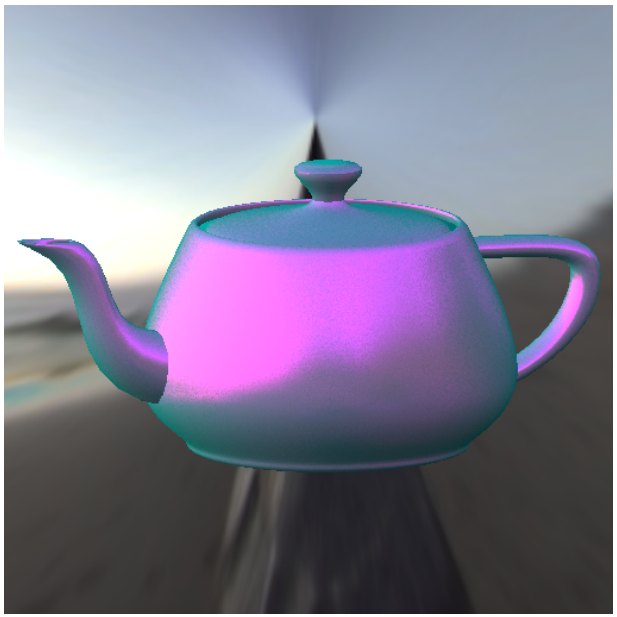
Plots de Distruição Blinn-Phong

Figure: *Plots* da distribuição de reflexão especular e difusa do experimento Blinn-Phong.





Objetos 3D renderizados pelo experimento Blinn-Phong.



Conclusão

- Objetivo alcançado: desenvolvimento de um compilador para traduzir BRDFs para código GLSL.
- Facilita a criação de *shaders* a partir de descrições matemáticas em LaTeX, democratizando o acesso a técnicas de computação gráfica avançada.
- Integração com ferramentas gratuitas como o visualizador da Disney.
- Projeto interdisciplinar envolvendo:
  - Gramáticas livres de contexto.
  - Traversia em Árvores.
  - Programação com *shaders*.
  - Fundamentos teóricos de refletância e radiometria.

Principais Contribuições

- Tradução de equações LaTeX em código GLSL para BRDFs:
- Visualização das árvores sintáticas geradas (SVG).
- Geração de mensagens de erro informativos para facilitar depuração.
- Suporte a operações fundamentais em computação gráfica:
  - Produto interno, vetorial.
  - Funções trigonométricas e exponenciação.

Resultados Obtidos

- Redução da barreira técnica para implementação de BRDFs.
- Sistema funcional e bem-sucedido em experimentos realizados.
- Possibilidade de focar na modelagem de refletância sem detalhes de baixo nível.

Perspectivas Futuras

- Ampliar suporte a novas construções matemáticas:
  - Somatórios (Σ) e produtos acumulados (Π).
  - Vetores de mais dimensões do que apenas 3.
- Adicionar derivadas ($\frac{d}{dx}$) e integrais ($\int$) para resolução numérica.
- Suporte a outras linguagens de *shading* como a usada por Unity[4] e Unreal.
- Desenvolver um editor LaTeX integrado com visualização simultânea.
- Melhorar o tratamento de erros para maior clareza e contextualização.

[4] HLSL

Encerramento

Fim

- O projeto abre caminho para a democratização de técnicas avançadas em computação gráfica.
- evertonse.junior@gmail.com.

Referências

[Kaj86] James T Kajiya. “The rendering equation”. In: *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. 1986, pp. 143–150.

[PJH16] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation (3rd ed.)* 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Nov. 2016, p. 1266. ISBN: 9780128006450.

Modelos de BRDFs

- BRDF (Bidirectional Reflectance Distribution Function): Função que descreve como a luz é refletida por uma superfície.
- Apresentaremos modelos clássicos de BRDFs:
  - BRDF Pura Especular.
  - BRDF Difusa Ideal.
  - BRDF Brilhante.
  - BRDF Retro-Refletora.

BRDF Pura Especular

- Superfície reflete luz apenas em uma direção.
- Segue a Lei da Reflexão:

$$f(\omega_i, \omega_o) = k_s \cdot \delta(\omega_i - \omega_o)$$

- Materiais típicos: metal polido, vidro.

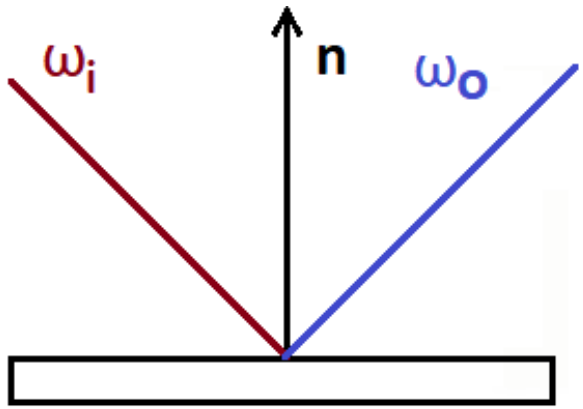


Figure: Reflexão especular. Raio incidente em vermelho e raio refletido em azul.

BRDF Difusa Ideal

- Reflexão uniforme em todas as direções.
- Função BRDF:

$$f(\omega_i, \omega_o) = \frac{\rho_d}{\pi} \cdot \cos\theta$$

- Exemplos: tinta fosca, papel.

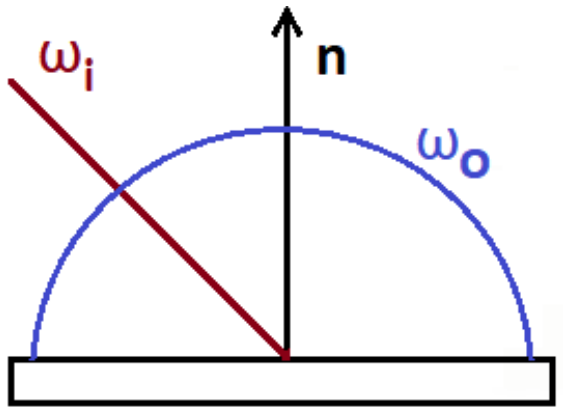


Figure: Reflexão difusa. Raios refletidos independem do ângulo de entrada.

BRDF Brilhante

- Combina reflexões especulares e difusas.
- Modelo típico: Blinn-Phong.

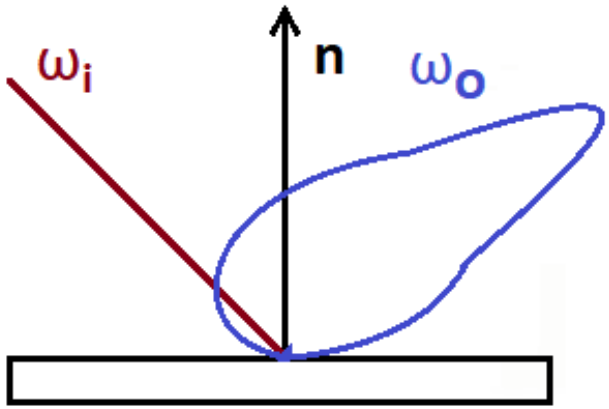


Figure: Reflexão *glossy*.

BRDF Retro-Refletora

- Redireciona a luz de volta à fonte.
- Utilizado em superfícies como placas de trânsito e sinais de segurança.

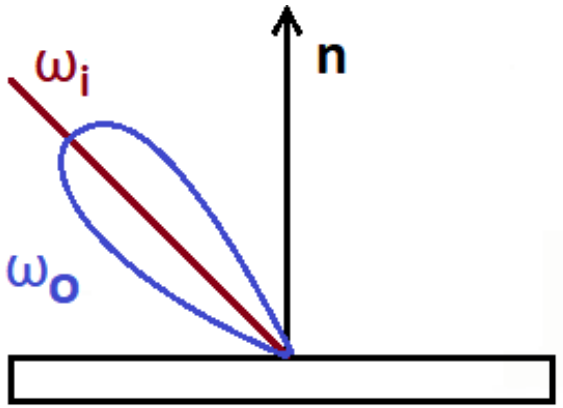


Figure: Reflexão retro-refletora.

BRDF: Função de Refletância Bidirecional

- BRDF (f) descreve como a luz reflete em diferentes direções:

$$f(\omega_i, \omega_o) = \frac{dL_o(\omega_o)}{L_i(\omega_i)\cos(\theta_i)d\omega_i}$$

- Propriedades:
  1. Positividade: $f \geq 0$.
  2. Conservação de energia: $\int_\Omega f \cos(\theta_i) d\omega_i \leq 1$.
  3. Reciprocidade de Helmholtz: $f(\omega_i, \omega_o) = f(\omega_o, \omega_i)$.

Radiometria: Introdução

- Estuda a interação da luz com superfícies na computação gráfica.
- Quantifica energia luminosa:
  - Brilho da fonte de luz.
  - Iluminação e refletância da superfície.
- Fundamenta a renderização realista de cenas tridimensionais.

Energia Radiante Q

- Quantifica a energia total dos fótons atingindo uma superfície.
- Fórmula:

$$Q = \frac{hc}{\lambda}$$

onde:

- h: Constante de Planck.
- c: Velocidade da luz.
- λ: Comprimento de onda.

Fluxo Radiante e Irradiância

- Fluxo Radiante: Energia por unidade de tempo (J/s):

$$\phi = \frac{dQ}{dt}$$

- Irradiância: Fluxo radiante por unidade de área:

$$E(p) = \frac{d\phi(p)}{dA}$$

- Quantifica os impactos de fótons em uma superfície.[5]

[5] Saber quantos fótons "tocam" uma superfície por segundo, é saber quão iluminado é aquela parte do objeto.

Radiância

Radiância (L): Fluxo radiante por unidade de área e ângulo sólido:

$$L = \frac{d^2\Phi}{dA\, d\omega \cos(\theta)}$$

Onde:

- $d\omega$: Ângulo sólido (em sr).
- θ: Ângulo entre direção de incidência e normal da superfície.

Visualização da Radiância

- Radiância considera direção específica no hemisfério sobre a superfície.

Figure: Visualização da radiância em uma direção específica do hemisfério.

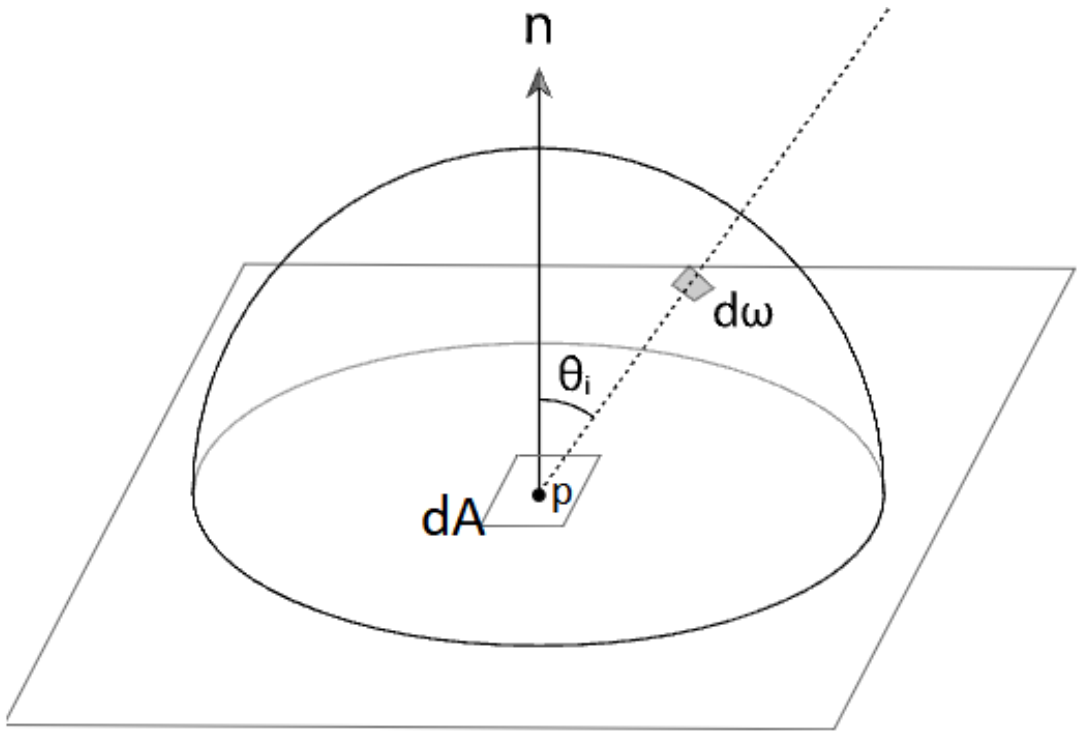


Figure: Ângulo sólido s do objeto B visto pelo ponto p.

Árvore SVG