2
Fundamentos teóricos

Neste Capítulo serão descritos os conceitos básicos que permitirão um melhor entendimento do método proposto. Primeiramente serão apresentados os conceitos de variedades implícitas. Posteriormente serão apresentados os conceitos da Aritmética Intervalar, da estrutura de dados *16–Tree*, do *ray casting*, do pipeline gráfico e do modelo de iluminação.

2.1
Variedades implícitas

Considere uma função diferençável $F : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$, onde 0 é um valor regular, isto é $\nabla F(p) \neq 0$ para todo p tal que $F(p) = 0$. Neste caso, o conjunto de pontos p é uma variedade diferenciável de dimensão $n - 1$, pelo Teorema da Função Implícita, que denotaremos por $F^{-1}(0)$ a qual simplesmente chamaremos de variedade implícita (Figueiredo et al., 1992; Spivak, 1965).

Este trabalho estuda a visualização de variedades implícitas de dimensão três no espaço $\mathbb{R}^4$, determinadas por uma única função implícita com quatro variávies ($n = 4$):

$$F(x, y, z, w) = 0. \tag{2-1}$$

A função implícita F divide o $\mathbb{R}^4$ em três regiões: a primeira corresponde ao conjunto de pontos em que F tem sinal negativo, a segunda corresponde ao conjunto de pontos em que F tem sinal positivo, e a terceira corresponde ao conjunto de pontos onde F é igual a zero. No primeiro caso, será dito que um ponto que pertence a essa região está dentro da variedade; no segunda caso, será dito que o ponto está fora da variedade; e, finalmente, no terceiro caso, o ponto será dito que o ponto está sobre a variedade.

Futuramente, para calcular a iluminação em um ponto, será preciso calcular a normal da variedade nesse ponto. A normal no ponto $\mathbf{p} = (x, y, z, w) \in F^{-1}(0) \subset \mathbb{R}^4$ é definida pela gradiente da função:

$$\mathbf{n} = \nabla F(\mathbf{p}) = (\frac{\partial F}{\partial x}(\mathbf{p}), \frac{\partial F}{\partial y}(\mathbf{p}), \frac{\partial F}{\partial z}(\mathbf{p}), \frac{\partial F}{\partial w}(\mathbf{p})) \tag{2-2}$$

2.2 Aritmética intervalar

O objetivo desta seção é definir as notações e os conceitos básicos de *Aritmética Intervalar* (AI). Mais detalhes sobre a teoria de AI podem ser encontrados em (Moore, 1966).

O conjunto de todos os intervalos reais compactos será denotado por $\mathbb{IR}$. Um elemento $[x]$ em $\mathbb{IR}$ corresponde a um intervalo $[\underline{x}, \overline{x}]$, onde $\underline{x}$ and $\overline{x}$ são dois números reais tal que $\underline{x} \leq \overline{x}$.

Para um intervalo $[x] = [\underline{x}, \overline{x}] \in \mathbb{IR}$, seus dois pontos extremos $\underline{x}$ e $\overline{x}$ são chamados, respectivamente, o *ínfimo* e o *supremo* de $[x]$ e eles são denotados por $\inf([x])$ e $\sup([x])$.

A operação aritmética binaria básica $\circ \in \{+, -, \times, \div\}$ para números reais pode ser estendida para dois intervalos $[x]$ e $[y] \in \mathbb{IR}$ na seguinte maneira: $[x] \circ [y] := \{x \circ y | x \in [x] \text{ and } y \in [y]\}$. O operador arimético $\div$ só pode ser aplicado quando $0 \notin [y]$. O resultado de qualquer operação aritmética intervalar é um intervalo, e usando suas propriedades de monotonicidade, eles podem ser expressos em termos dos pontos extremos dos operandos, por exemplo, $[\underline{x}, \overline{x}]+[\underline{y}, \overline{y}] = [\underline{x}+\underline{y}, \overline{x}+\overline{y}]$. Todas estas operações aritméticas binarias $\circ \in \{+, -, \times, \div\}$ têm *inclusão isotônica*, que significa: $[z] \subseteq [x]$ and $[w] \subseteq [y] \Rightarrow [z] \circ [w] \subseteq [x] \circ [y]$.

O *domínio* de uma função real f é denotado por $Dom(f)$. A *imagem* de uma função continua real f sobre todos os pontos x no intervalo real compacto $[x] \subseteq Dom(f)$ é definido como sendo o intervalo $Im(f, [x])$ tal que $Im(f, [x]) := [\min\{f(x) | x \in [x]\}, \max\{f(x) | x \in [x]\}]$.

A *extensão intervalar de uma função real elemental* $\phi \in \{\exp, \ln, \cos, \sin, \tan, \cos^{-1}, \sin^{-1}, \tan^{-1}, \cosh, \sinh, \tanh, \cosh^{-1}, \sinh^{-1}, \tanh^{-1}\}$ sobre um intervalo dado $[x] \subseteq Dom(\phi)$ é definida como sendo a imagem de ϕ sobre $[x]$, por exemplo, $\phi([x]) := Im(\phi, [x])$. As funções potência e racional são definidas de maneira semelhante.

Uma função intervalar γ é dita que tem *inclusão isotônica* quando para todos os pares de intervalos $[x]$ e $[y]$ contido em seu domínio, a proposição $[x] \subseteq [y] \Rightarrow \gamma([x]) \subseteq \gamma([y])$ é sempre verdadeira. Por definição, a extensão intervalar de todas as funções elementares citadas acima têm essa propriedade importante. Já que cada função real elemental é continua e sua extensão intervalar tem inclusão isotônica, o intervalo $\phi([x])$ pode também ser expresso em termos de pontos de limite de $[x]$, por exemplo: $\exp([x]) := [\exp(\inf([x])), \exp(\sup([x]))]$.

Seja f uma função escalar de uma variável real x definida por uma expressão que contém apenas operações aritméticas e funções elementares. A

extensão intervalar de f, denotada por $f_{[]}$, sobre um intervalo real compacto $[x] \subseteq Dom(f)$ é definida como sendo a função intervalar construída a partir da substituição de cada ocorrência da variável real x na expressão de f pela sua correspondente variável intervalar $[x]$. O intervalo $f_{[]}([x])$ é então obtido pelo uso das correspondentes operações aritméticas intervalares e funções intervalares elementares sobre a avaliação da sua expressão. Quando o intervalo $[x]$ é um único ponto ($[x] = [a, a]$), a extensão intervalar $f_{[]}$ satisfaz $f_{[]}([a, a]) = f(a)$. Já que todas as operações de intervalo e funções elementares têm inclusão isotônica, a extensão intervalar de uma função real também tem inclusão isotônica. O próximo teorema é a principal ferramenta utilizada neste trabalho. Ela liga a AI à imagem verdadeira da função de estudo.

Teorema. Seja f uma função continua real de uma variável e $f_{[]}$ sua extensão intervalar. Se $[x] \subseteq Dom(f)$, então $Im(f, [x]) \subseteq f_{[]}([x])$.

Quando um ponto $y \notin f_{[]}([x])$, este teorema garante que o ponto y não fica no conjunto $Im(f, [x])$. Desafortunadamente, o intervalo $f_{[]}([x])$ é em geral, uma sobrestimação da imagem de f sobre o intervalo $[x]$. Esse Teorema vale também para funções reais contínuas em n variáveis, que são os principais objetos de estudo desse trabalho.

2.3 Ray casting implícito

Ray Casting é uma técnica de computação gráfica que cria imagens lançando raios a partir de uma câmera e salvando na tela a interseção da trajetória dos raios com os objetos. Foi primeiramente apresentado por (Appel, 1968). Outra técnica semelhante é a de *Ray Tracing*, primeiro proposto por (Whitted, 1980). A principal diferença entre *Ray Tracing* e *Ray Casting* é que a primeira permite traçar raios secundários para calcular sombras, reflexões e refrações.

A Figura 2.1, apresenta o processo de *ray tracing* em $3D$ com uma câmera, uma janela com pixeis, uma fonte de luz e dois raios intersectando uma esfera e um toro.

Muitos trabalhos em visualização de variedades implícitas são relacionados às técnicas de *Ray Tracing*, nas quais são utilizados raios secundários para obter uma iluminação realista e modelos de sombreado tal como iluminação global (Kajiya, 1986). Mais recentemente, a comunidade vem concentrando esforços em tomar o *Ray Tracing* mais interativo, através de paralelização e otimizações (Singh, 2013).

Este trabalho só se concentrará na técnica de *Ray Casting*, para não sobrecarregar os cálculos, já que será feito uso da programação em GPU. Nas seguintes subseções serão descritas os conceitos de Raio (subseção 2.3.1),

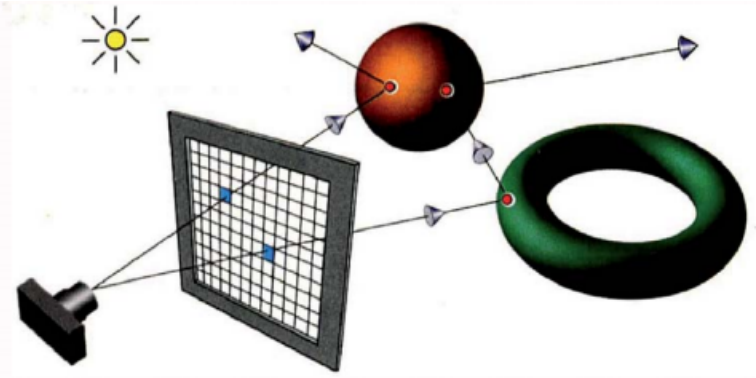

Figura 2.1: O processo do *Ray Tracing*. Fonte: (Suffern, 2007).

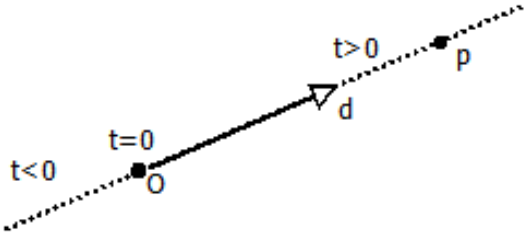

Figura 2.2: Definição do raio.

interseção do raio com uma hiperesfera (subseção 2.3.2), e a interseção do raio com uma função implícita (subseção 2.3.3) no $\mathbb{R}^4$.

2.3.1 Raio

Um raio é uma linha infinita que está definida por um ponto $\mathbf{o}$, chamado de origem, e um vetor $\mathbf{d}$, chamado de direção. Um raio $\mathbf{r} : \mathbb{R} \rightarrow \mathbb{R}^4$ é uma função parametrizada por um número real t. Quando $\mathbf{r}(0)$ corresponde ao ponto de origem do raio. Um ponto arbitrário $\mathbf{r}(0)$ no raio é expressado por (Suffern, 2007):

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}. \tag{2-3}$$

A Figura 2.2 ilustra a definição de um raio.

2.3.2 Interseção de raio com uma hiperesfera

Uma hiperesfera é definida pelo conjunto de pontos $\mathbf{p} = (x, y, z, w) \in \mathbb{R}^4$ que distam R de um ponto $\mathbf{c}$, onde R é o raio e $\mathbf{c} = (c_x, c_y, c_z, c_w)$ é o centro da hiperesfera. Esse objeto pode ser representado através de uma equação implícita que é dada por:

$$(\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) - R^2 = 0 \tag{2-4}$$

onde o operador $\cdot$ corresponde ao produto interno usual no $\mathbb{R}^4$. Essa equação pode ser re-escrita na seguinte forma:

$$(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 + (w - c_w)^2 - R^2 = 0 \tag{2-5}$$

Para interceptar um raio com uma hiperesfera, devemos substituir a Equação 2-3 em 2-4 para obter:

$$(\mathbf{o} + t\mathbf{d} - \mathbf{c}) \cdot (\mathbf{o} + t\mathbf{d} - \mathbf{c}) - R^2 = 0 \tag{2-6}$$

Para achar as duas raízes, expandimos a Equação 2-6 de tal modo a obter:

$$(\mathbf{d} \cdot \mathbf{d})t^2 + [2(\mathbf{o} - \mathbf{c}) \cdot \mathbf{d}]t + (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - R^2 = 0 \tag{2-7}$$

A Equação 2-7 é uma equação quadrática em t, que pode ser escrita como:

$$at^2 + bt + c = 0, \tag{2-8}$$

onde

$$\begin{aligned} a &= \mathbf{d} \cdot \mathbf{d}, \\ b &= 2(\mathbf{o} - \mathbf{c}) \cdot d, \\ c &= (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - r^2 \end{aligned} \tag{2-9}$$

A solução da Equação 2-8 é dada pela expressão:

$$t = \frac{-b \pm \sqrt{(b^2 - 4ac)}}{2a}$$

As equações quadráticas podem ter nenhuma, uma ou duas raízes reais, dependendo do valor do discriminante $b^2 - 4ac$.

Neste trabalho consideramos os casos que possuem duas raízes reais, e serão descartadas as outras duas situações.

2.3.3 Interseção de raio com função implícita

A interseção entre uma função implícita $F(x, y, z, w) = 0$ e um raio é definida substituindo os parâmetros de F pela Equação do raio (2-3), com o qual obtemos:

$$g(t) = F(o_x + td_x, o_y + td_y, o_z + td_z, o_w + td_w) \tag{2-10}$$

Calcular a interseção com um raio significa encontrar as raízes da função g dada pela Equação 2-10.

Existem muitas técnicas para encontrar essas raízes (Díaz, 2008). O enfoque usado neste trabalho é do amostragem de pontos com bisseção, um exemplo está na Figura 2.3.

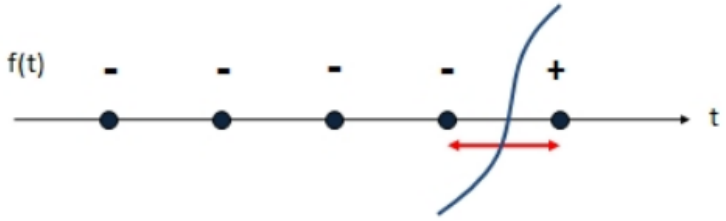

Figura 2.3: Encontrando a raiz com amostragem de pontos com bisseção.

Considerando um domínio t_{min} e t_{max} para os valores do parâmetro t do raio, devemos amostrar esse intervalo, numa quantidade finita de pontos, e para cada ponto avaliar o sinal da função g. Se houver dois pontos com sinais diferentes, então isso significa que nesse intervalo há pelo menos uma interseção, pelo Teorema do Valor Intermediário, já que g assim como F são funções contínuas (ver Figura 2.3).

Dados esses dois pontos extremos do intervalo em t, é executado um algoritmo da bisseção para determinar uma raiz definida (ver Algoritmo 2.1), ou seja um ponto sobre a variedade $F^{-1}(0)$.

2.4 Programação em GPU

O objetivo desta seção é apresentar os conceitos básicos da programação em GPU usando o pipeline programável. Uma explicação mais detalhada pode ser encontrada no livro (Shreiner et al., 2013), (Engel et al., 2006) e em (Akenine-Möller et al., 2008).

A maioria de computadores modernos estão equipados com um processador para gráficos 3D acelerados por *hardware*. As unidades de processamento de gráficos (GPU, pelas suas siglas em inglês *graphics processing units*), são altamente otimizadas para processamento de dados em paralelo. As GPUs têm substituído o tradicional pipeline de função fixa por um pipeline programável, o qual permite ao programador escrever pequenos programas para serem executados muito rápidos e eficientemente.

Agora será descrito o pipeline e seus estágios programáveis.

2.4.1 O pipeline programável

Para visualizar uma cena virtual, primeiro a cena deve ser descomposta em polígonos planares, geralmente triângulos. Ao enviar esses polígono à GPU, ela gera imagens *raster* de forma muito rápida e eficientemente. Esse processo de conversão de um conjunto de primitivas poligonais em uma imagem *raster* é chamado de *display transversal*.

Algorithm 2.1 Algoritmo da bisseção recursivo.

Require: $f_1 \cdot f_2 < 0$.
Ensure: A raiz da função F entre t_1 e t_2.

```
1: function SOLVE(t1, t2)
2:     f1 ← F(t1)
3:     f2 ← F(t2)
4:     if f1 < 0 then
5:         return BISECT(t1, t2, f1, f2)
6:     else
7:         return BISECT(t2, t1, f2, f1)
8:     end if
9: end function
```

Require: $f_1 < 0$ and $f_2 > 0$.
Ensure: A raiz da função F entre t_1 e t_2.

```
10: function BISECT(t1, t2, f1, f2)
11:     tm = (t1 + t2)/2
12:     fm = F(tm)
13:     if fm = 0 then
14:         return tm                          ▷ tm é uma raiz
15:     end if
16:     if fm < 0 then
17:         return BISECT(tm, t2, fm, f2)
18:     else
19:         return BISECT(t1, tm, f1, fm)
20:     end if
21: end function
```

Todos os processadores gráficos 3D implementam o *display transversal* como um pipeline consistindo de uma seqüência fixa de estágios de processamento. A Figura 2.4 apresenta a ordem de operações de um processador gráfico moderno com o uso do OpenGL com versões a partir da 3.2. As caixas azuis são os estágios programáveis, as caixas brancas representam os processamentos internos da GPU, e os números em parênteses indicam qual versão de OpenGL é requerida. Nas seguintes subseções cada estágio será descrito separadamente.

2.4.2 Vertex Shader

O *vertex shader* opera nos vértices de forma individual, ou seja: um vértice cada vez. O *vertex shader* não têm conhecimento dos outros vértices que formam a primitiva geométrica. Ele calcula transformações lineares dos vértices de entrada, tais como rotação, translação e escalonamento no espaço tridimensional. Este passo compreende a transformação dos vértices desde as

coordenadas do modelo local dentro do espaço do mundo (matriz do modelo), logo dentro do espaço da câmera (matriz de visão), até o espaço da janela (matriz de projeção).

2.4.3 Tessellation

Tessellation é um estágio que recebe *patches* como entradas e gera novas primitivas geométricas que podem ser pontos, linhas ou triângulos. Um *patch* é dado por um array de vértices e seus atributos são calculados pelo *vertex shader*.

2.4.4 Geometry Shader

Este estágio é opcional, se está ativo o *geometry shader* ele recebe como entrada as primitivas geométricas construídas no estágio anterior. O *geometry shader* tem acesso a todos os vértices da primitiva na qual ele está trabalhando, e se foi especificado, ele pode também ter acesso a informação de adjacência. Os tipos de primitivas geométricas de entrada e de saída devem ser declarados, a saída pode ter zero ou mais primitivas geométricas.

2.4.5 Rasterization and Interpolation

Este estágio não é programável, é um componente fixo do pipeline gráfico, mas é importante descrever o que acontece aqui, para entender melhor o pipeline.

Rasterização

É o processo de determinar o conjunto de pixeis na imagem final, que são parte da primitiva. Para cada pixel que tem seu centro dentro dos limites do triângulo serão adicionados ao conjunto de pixeis para processamento futuro. Na Figura 2.5 na parte esquerda, os pontos com cores representam as posições dos vértices no espaço da tela.

Interpolação

O seguinte passo é calcular os atributos para cada pixel baseado nos atributos e na distância do pixel há cada vértice na posição da tela. Na Figura 2.5 na parte direita, pode-se observar os cores dos pixeis interpolados.

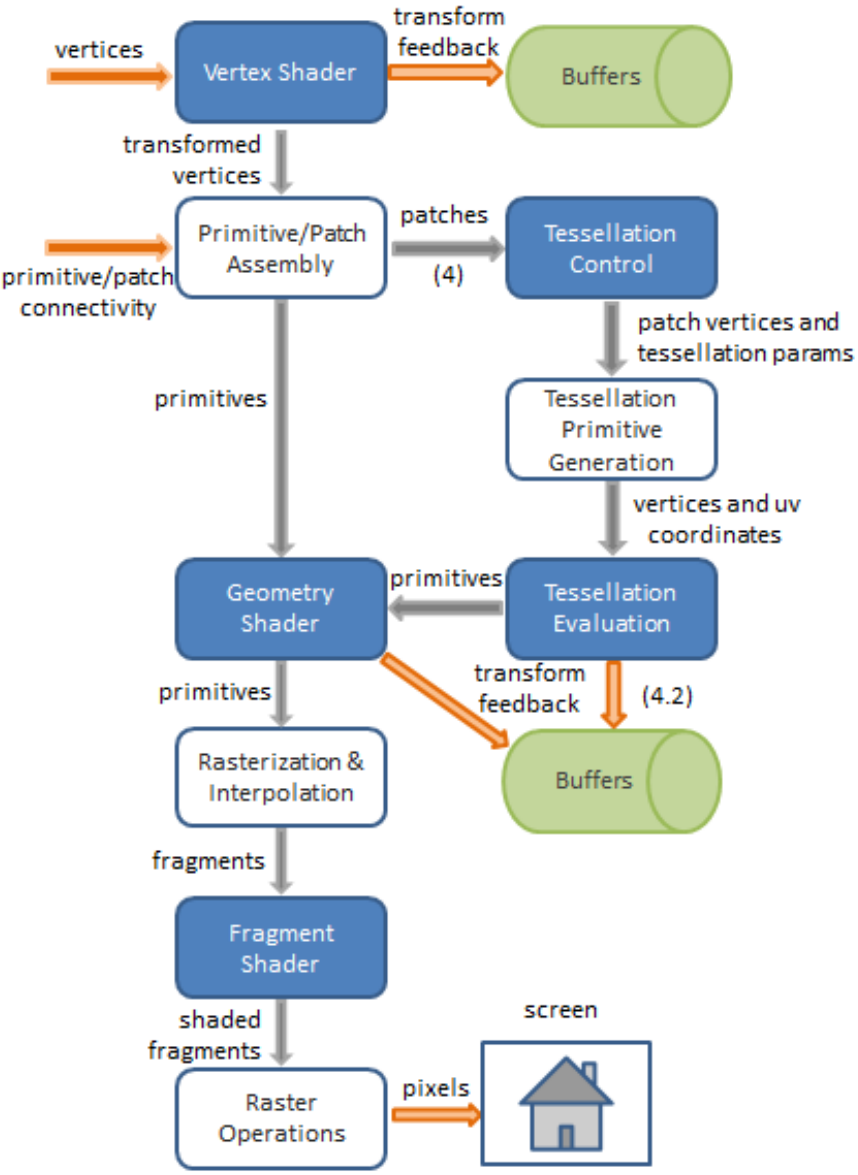

Figura 2.4: O pipeline gráfico programável. Fonte: (Lighthouse3d).

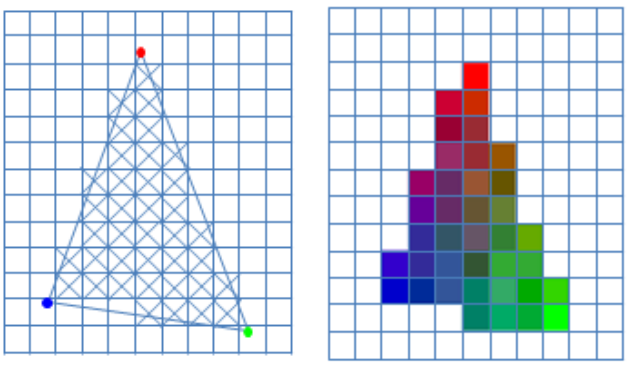

Figura 2.5: Um exemplo do estágio de rasterização (esquerda) e interpolação (direita) do pipeline gráfico. Fonte: (Lighthouse3d).

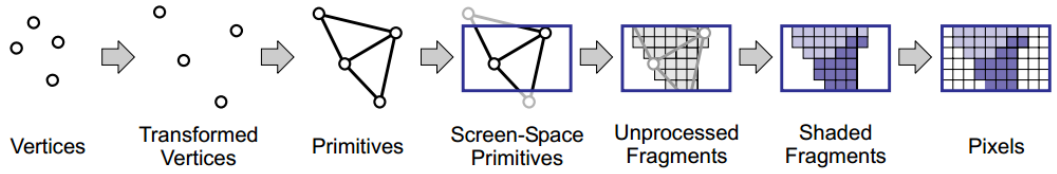

Figura 2.6: Uma descrição visual dos estágios do pipeline gráfico, desde os vértices até os pixeis. Fonte: (Engel et al., 2006).

2.4.6 Fragment Shader

Neste estágio cada fragmento tem seus próprios atributos interpolados no estágio anterior, não é possível acessar atributos de fragmentos vizinhos. O processador de fragmento é capaz de efetuar várias texturas e usar operações de filtragem para cada fragmento. O programa calcula a cor final do fragmento a partir dos atributos de vértice interpolados, as amostras de texturas filtrados e o código implementado.

Um *fragment shader* só pode ter uma variável de saída, a qual deve ser o cor do fragmento. Algumas variáveis próprias de cada fragmento são:

- gl_FragCoord: contêm as coordenadas do fragmento (xf, yf, zf, wf), onde (xf, yf) é a posição do pixel na tela, zf é a profundidade, e wf é $1/wc$, onde wc é o componente do fragmento no espaço de *clipping*.
- gl_FrontFacing: diz a orientação da primitiva que originou o pixel.
- gl_FragDepth: contêm a profundidade do pixel, se está ativo o buffer de profundidade.

Na Figura 2.6 pode-se observar uma descrição visual dos estágios anteriormente descritos, desde os vértices até os pixeis.

2.5 Modelo de iluminação de Phong

Nesta seção é explicado o modelo de iluminação que foi implementado. Foi escolhido o modelo de iluminação de Phong, porque é o modelo de iluminação mais usado em computação gráfica.

A iluminação é realizado no espaço $\mathbb{R}^4$, considerando as posições de uma ou varias fontes de luz. A Figura 2.7 apresenta os vetores implicados no processo de iluminação:
$\vec{e}$: vetor desde o ponto de vista, em neste trabalho este é o vetor direção do raio desde o pixel,
$\vec{l}$: vetor para a luz,

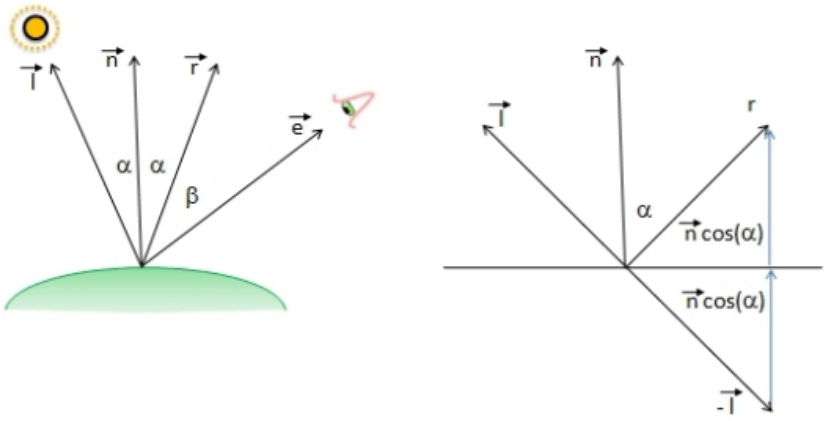

Figura 2.7: Geometria do modelo de iluminação de Phong, à esquerda os vetores que intervêm e a direita o cálculo do raio de reflexão da luz (Díaz, 2008).

$\vec{n}$: normal da superfície,
$\vec{r}$: raio de reflexão.

O modelo de iluminação de Phong calcula a luz refletida por um objeto como uma combinação de três diferentes termos: ambiente, difuso e especular (Engel et al., 2006), através da seguinte equação:

$$I_{Phong} = I_{ambiente} + I_{difuso} + I_{especular}$$

Os termos são escritos como valores RGB.

O termo ambiente é modelado como uma constante de luz global multiplicado pelo cor ambiente e o coeficiente ambiente do material:

$$I_{ambiente} = k_a \; C_a \; I_a,$$

onde:
I_a representa a cor e a intensidade da luz ambiente global,
C_a representa a cor ambiente do material,
k_a representa o coeficiente da luz ambiente, que é uma constante entre 0 e 1. Essa constante controla o quanto da luz ambiente que chega à superfície é refletida.

O termo difuso corresponde à reflexão *Lambertiana*, onde a luz é refletida igualmente em todas as direções:

$$I_{difuso} = k_d \; C_d \; I_d \; max(0, n \cdot l),$$

onde:
I_d representa a cor e a intensidade emitida pela fonte de luz,

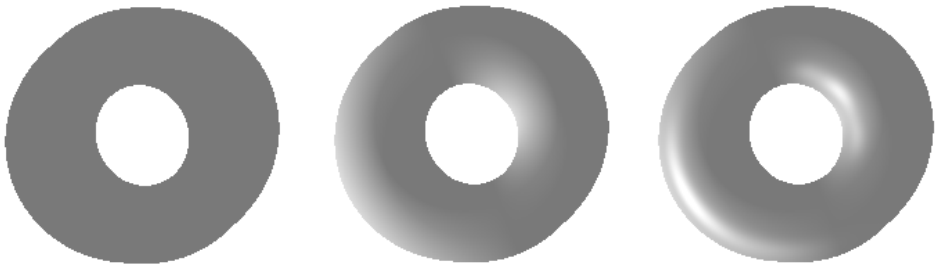

Figura 2.8: Exemplo da iluminação com o modelo de Phong. Esquerda, iluminando com o termo ambiente. Centro, adicionando o termo difuso. Direita, imagem final com o termo especular.

C_d representa a cor difuso do material,
k_d representa o coeficiente difuso, constante entre 0 e 1.
Segundo o termo difuso, a cor de um ponto na superfície é proporcional ao cosseno do ângulo (produto ponto) entre a direção da luz $\vec{l}$ e a normal $\vec{n}$ no ponto. A função max é usada para os casos no qual o produto ponto é negativo, nesses pontos a luz não chega e ficam na sombra.

O termo especular modela a reflexão de superfícies brilhantes e depende do vetor de visão $\vec{e}$:

$$I_{especular} = k_s \; C_s \; I_s \; max(0, r \cdot e)^n,$$

onde:
I_s representa a intensidade emitida pela fonte de luz,
C_s representa a cor especular do material,
k_s representa o coeficiente especular, que determina a quantidade de reflexão especular do material,
n representa o exponente especular chamado de brilhantes da superfície, e é usado para controlar o tamanho da iluminação resultante.

Na Figura 2.8, pode-se observar um exemplo da iluminação com o modelo de Phong.

2.5.1 Tipos de fontes de luz

Em computação gráfica tem muitas variedades de tipos de fontes de luz, cada tipo cria um efeito visual único, e adiciona uma quantidade de complexidade computacional à cena. Mais informações podem ser obtidas em (Engel et al., 2006).

Neste trabalho foram utilizadas as fontes de luz pontual e direcional, por serem de menor carga computacional.

- *Fontes de luz pontual* emitem luz a partir de um só ponto no espaço, igualmente para todas as direções.
- *Fontes de luz direcional* são fontes de luz pontual no infinito. São somente descritos pela direção da luz, e todos os raios emitidos são paralelos um ao outro.