3
Método proposto

3.1
Introdução

Neste Capítulo na primeira seção será explicado uma descrição geral do método, e na seguintes seções serão explicados os passos da primeira e segunda fase do método, que correspondem ao pré-processamento feito em CPU e ao processamento feito em GPU, respectivamente.

3.2
Descrição geral do método

Considere $F : \Omega \subseteq \mathbb{R}^4 \to \mathbb{R}$ uma função continuamente diferenciável, onde 0 é um valor regular de F. O método proposto faz a visualização em tempo real para a variedade implícita $\mathcal{M} = F^{-1}(0)$ de dimensão três.

Considere um domínio $\Omega = [\mathbf{h}] \subseteq \mathbb{R}^4$ que é uma hipercaixa dada por $[x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [z_{min}, z_{max}] \times [w_{min}, w_{max}]$. A primeira fase do método consiste em um algoritmo que subdivide $[h]$ usando uma estrutura de dados *16-Tree*. Esse algoritmo faz as subdivisões do domínio de acordo com critérios baseados na Aritmética Intervalar, para descartar as caixas que não interceptam a variedade. Para cada hipercaixa dessa subdivisão que a variedade intercepta são calculados os seus pontos centrais e o raio da hiperesfera que a envolvente. Essa estrutura de dados é enviada para a GPU juntamente com a matriz de rotação para realizar a segunda fase do método.

Na GPU são usadas as informações das hiperesferas associadas às hipercaixas da subdivisão em conjunto com a matriz de rotação para executar a técnica de *Ray Casting* afim de visualizar a variedade implícita dada pela função F. Para isso, são calculadas as cores de cada pixel considerando a fonte de luz, dando destaque aos pontos que pertencem à silhueta ou ao bordo do domínio.

A Figura 3.1 apresenta um diagrama contendo as etapas do método proposto para a visualização da variedade. Na próxima seção será descrito os detalhes de cada etapa do pré-processamento em CPU, e na seção (3.4) é descrito o processamento em GPU.

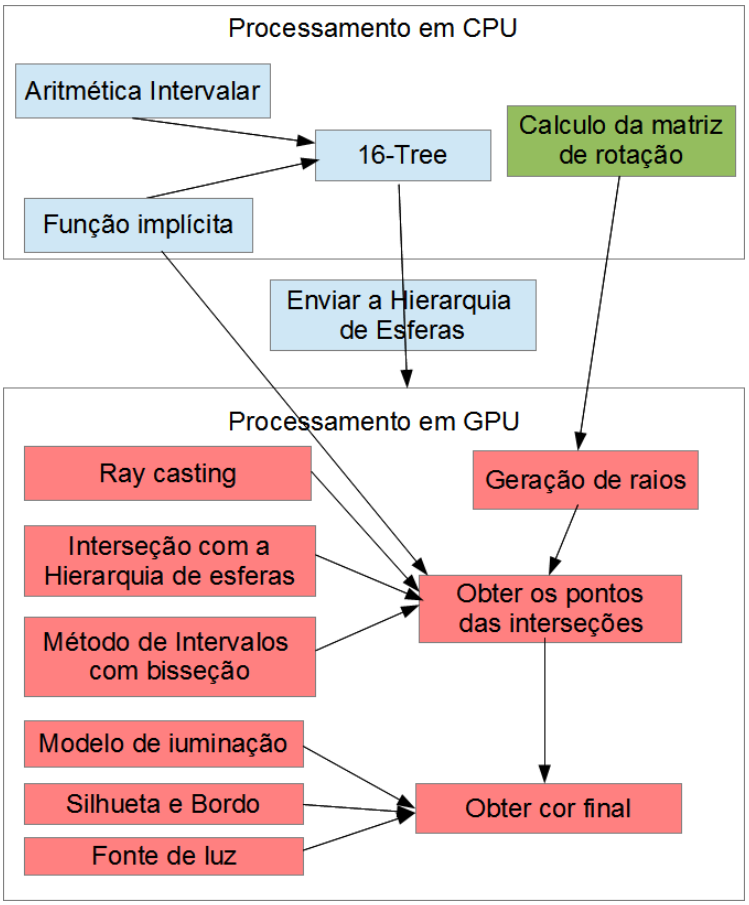

Figura 3.1: Diagrama da descrição geral do método proposto, as caixas azuis são processadas em CPU uma só vez, a caixa verde é processada em CPU em cada frame, e as caixas vermelhas são processadas em GPU em cada frame.

3.3 Pré-processamento em CPU

Nesta seção será explicado a primeira fase do método proposto, que corresponde ao pré-processamento em CPU.

3.3.1 Construção da 16-Tree

O algoritmo de subdivisão é iniciado considerando o domínio [**h**] que representa a hipercaixa da raiz da *16-Tree*. Verifica-se se a caixa atual deve ser subdividida, usando critérios baseados na Aritmética Intervalar. A subdivisão de um nó da *16-Tree* gera 16 novas hipercaixas, e o processo é repetido recursivamente para cada uma delas. Cada hipercaixa da subdivisão é marcada se contem ou não a variedade, e essa marcação serve para processar futuramente apenas as caixas que contem a variedade (subseção 3.3.4).

Uso da Aritmética Intervalar

Considere uma função $F : \Omega \subseteq \mathbb{R}^4 \to \mathbb{R}$ e uma caixa retangular $[\mathbf{h}] \subseteq \Omega$, onde suas dimensões são representadas pelos intervalos $[x]$, $[y]$, $[z]$ e $[w]$, usando os conceitos de Aritmética Intervalar definidos na seção 2.2. Seja a função $F_{[]} : \mathbb{IR}^4 \to \mathbb{IR}$ a extensão intervalar da função F, então é possível dizer que:

$$F_{[]}([\mathbf{h}]) = F_{[]}([x], [y], [z], [w]) \subseteq \mathbb{IR}$$

Assim, $F_{[]}([\mathbf{h}])$ são intervalos que estimam o conjunto de valores resultantes da avaliação intervalar da função F na caixa $[\mathbf{h}]$. Isso permite avaliar se a caixa $[\mathbf{h}]$ não intercepta pontos da variedade implícita $\mathcal{M} = F^{-1}(0)$, que significa:

$$0 \notin F_{[]}([\mathbf{h}]) \Rightarrow 0 \notin F([\mathbf{h}]).$$

Esse fato é usado no processo de subdivisão da árvore *16-Tree*, como será descrito na próxima subseção.

Critérios de Subdivisão

Para que uma caixa seja subdividida ela precisará satisfazer três critérios: critério da componente conexa, critério topológico e critério de nível máximo.

Critério da Componente Conexa

Este critério avalia se uma caixa pode conter pontos da variedade $\mathcal{M} = F^{-1}(0)$. Para isso, utiliza-se a avaliação intervalar $F_{[]}([\mathbf{h}_n])$ da função F na hipercaixa $[\mathbf{h}_n]$.

Se o valor $0 \notin F_{[]}([\mathbf{h}])$, então não há pontos da variedade na hipercaixa $[\mathbf{h}_n]$. Caso contrário, se $0 \in F_{[]}([\mathbf{h}])$, possivelmente pode haver pontos interceptando a variedade dentro dessa hipercaixa, então deve-se subdividir essa hipercaixa $[\mathbf{h}_n]$ para explorar melhor essa possibilidade.

Critério Topológico

O critério da componente conexa não garante a preservação topológica de uma variedade que contenha túneis ou buracos. Para evitar isso avalia-se se o vetor $(0, 0, 0, 0)$ fica dentro da imagem do gradiente da função F na hipercaixa $[\mathbf{h}_n]$. Se a avaliação intervalar do gradiente de F contém a origem, então deve-se subdividir a caixa $[\mathbf{h}_n]$.

Critério de Nível Máximo

Os critérios anteriores podem fazer que ocorram um grande número de subdivisões, até mesmo infinitas. Para evitar isso é estabelecido um nível máximo de subdivisões.

3.3.2
Rotação com ângulos de Euler no $\mathbb{R}^4$

Para definir a direção de visualização define-se um novo sistema de coordenadas no $\mathbb{R}^4$. A orientação de um sistema de coordenadas fixo, dado pelos eixos X, Y, Z e W, com respeito a um sistema de coordenadas de referência pode ser expressa por uma matriz ortogonal A_{4x4} da seguinte forma:

$$\mathbf{q} = A\mathbf{p},$$

onde,
$\mathbf{q} = (\mathbf{q}_1, \ldots, \mathbf{q}_n)^T$ é um vetor expresso nas coordenadas de referência, e
$\mathbf{p} = (\mathbf{p}_1, \ldots, \mathbf{p}_n)^T$ é o mesmo vetor expresso nas coordenadas fixo.

Neste trabalho foi escolhido utilizar as matrizes de rotação básicas dos ângulos de Euler para realizar a orientação desse novo sistema de coordenadas, a matriz A é o produto de 6 matrizes de rotações básicas. Uma rotação básica é uma rotação de um ponto dentro de um plano gerado por dois vetores bases do sistema de coordenadas.

Os ângulos de Euler no $\mathbb{R}^4$ podem ser divididos em três fases de rotação, como segue:

1. A primeira fase orienta o eixo W por três rotações básicas nos planos XY, YZ e ZW, através das matrizes dos ângulos de Euler: $R^4_{xy}(\theta_1)$, $R^4_{yz}(\theta_2)$ e $R^4_{zw}(\theta_3)$, respectivamente.

2. A segunda fase orienta o eixo Z em sua posição final no $\mathbb{R}^3$ por duas rotações básicas nos planos XY e YZ, através das matrizes: $R^3_{xy}(\theta_4)$ e $R^3_{yz}(\theta_5)$, respectivamente.

3. Na última fase uma única rotação orienta os eixos X e Y para suas posições no $\mathbb{R}^2$ fazendo uso da matriz: $R^2_{xy}(\theta_6)$

Dessa forma, a orientação de objetos por ângulos de Euler no $\mathbb{R}^4$ pode ser expressa como:

$$\mathbf{q} = A\mathbf{p} \Rightarrow \mathbf{q} = R^4_{xy}(\theta_1)\ R^4_{yz}(\theta_2)\ R^4_{zw}(\theta_3)\ R^3_{xy}(\theta_4)\ R^3_{yz}(\theta_5)\ R^2_{xy}(\theta_6)p,$$

onde as matrizes dos ângulos de Euler são expressas por:

$$R^4_{xy}(\theta_1) = \begin{bmatrix} cos(\theta_1) & -sin(\theta_1) & 0 & 0 \\ sin(\theta_1) & cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};\ R^4_{yz}(\theta_2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta_2) & -sin(\theta_2) & 0 \\ 0 & sin(\theta_2) & cos(\theta_2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

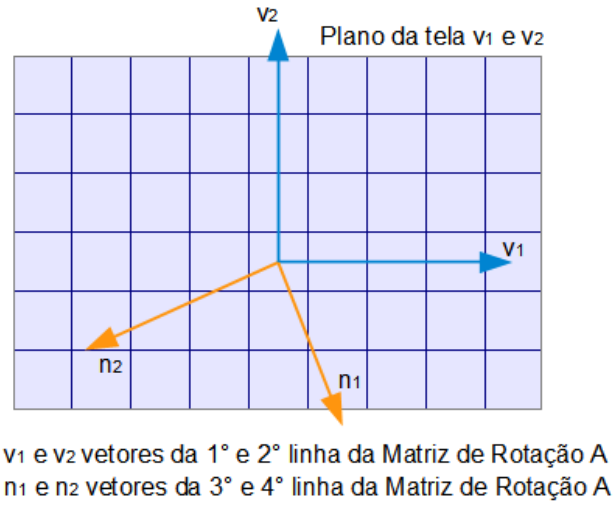

Figura 3.2: Vetores de projeção e observadores da Matriz A.

$$R^4_{zw}(\theta_3) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & cos(\theta_3) & -sin(\theta_3) \\ 0 & 0 & sin(\theta_3) & cos(\theta_3) \end{bmatrix}; \; R^3_{xy}(\theta_4) = \begin{bmatrix} cos(\theta_4) & -sin(\theta_4) & 0 & 0 \\ sin(\theta_4) & cos(\theta_4) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$R^3_{yz}(\theta_5) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta_5) & -sin(\theta_5) & 0 \\ 0 & sin(\theta_5) & cos(\theta_5) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \; R^2_{xy}(\theta_6) = \begin{bmatrix} cos(\theta_6) & -sin(\theta_6) & 0 & 0 \\ sin(\theta_6) & cos(\theta_6) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Os ângulos θ_1, θ_2, θ_3, θ_4, θ_5 e θ_6 são parâmetros definidos pelo usuário. A sistema de coordenadas inicial é dado por $\theta_1 = \theta_2 = \theta_3 = \theta_4 = \theta_5 = \theta_6 = 0$.

3.3.3 Projeção e observadores

Este trabalho usa a projeção ortogonal sobre o plano XY. No novo sistema de coordenadas, o plano de projeção XY é dado pela combinação linear dos vetores que correspondem às duas primeiras linhas da matriz de rotação A. Por outro lado, os observadores estão no plano perpendicular às direções X e Y, portanto esse plano perpendicular é obtido pela combinação linear das duas ultimas linhas da matriz de rotação A, observar a Figura 3.2.

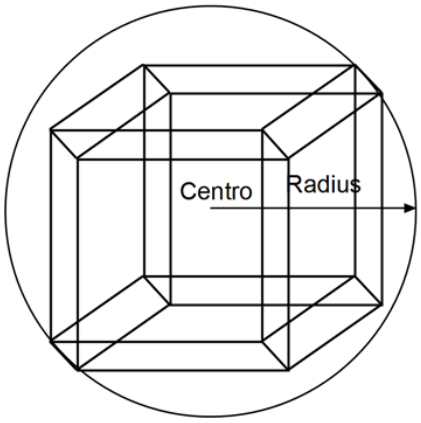

Figura 3.3: Hipercaixa e sua hiperesfera envolvente.

3.3.4 Envio de dados ao GPU

Terminada a construção da árvore *16-Tree*, o passo seguinte é o envio dos dados dessa árvore e da matriz de rotação para o GPU. Extrai-se da árvore uma hierarquia de hiperesferas envolventes, que é dada por:

- Os pontos centrais de cada hipercaixa que contém a variedade, e o valor da diagonal, que representa o raio da hiperesfera envolvente dessa hipercaixa. Equações 3-1 e 3-2.
- Referências às hipercaixas filhas que contêm a variedade.

A Figura 3.3 representa essa extração.

$$\mathbf{c} = ((\underline{x}+\overline{x})/2, (\underline{y}+\overline{y})/2, (\underline{z}+\overline{z})/2, (\underline{w}+\overline{w})/2) \quad (3\text{-}1)$$

$$R = \sqrt{(\overline{x}-\underline{x})^2 + (\overline{y}-\underline{y})^2 + (\overline{z}-\underline{z})^2 + (\overline{w}-\underline{w})^2} \quad (3\text{-}2)$$

A extração dessa informação é feita apenas uma vez, e assim que ela estiver disponível é enviada à GPU.

O cálculo da matriz de rotação e feita em cada *frame*, e enviada à GPU em cada processamento do *frame*.

O processo continua no próximo capítulo, onde serão apresentados os cálculos para a visualização que são efetuados na GPU.

3.4 Processamento em GPU

Nesta seção será explicado a segunda fase do método proposto, que corresponde ao processamento em GPU do cálculo de *Ray Casting*. Também são descritas algumas otimizações para diminuir os cálculos em GPU.

3.4.1 Descrição geral do processamento em GPU

O processamento começa com o envio de dois triângulos à GPU. No estágio do *vertex shader*, os vértices permanecem inalterados, e são enviados para o próximo estágio. Os triângulos são rasterizados e para cada pixel se ativa um *fragment shader*. Neste estágio é calculado a posição do pixel no $\mathbb{R}^4$, usando as duas primeiras filas da matriz de rotação, e são gerados n raios no plano formado pelos dois observadores. Para cada raio procura-se a interseção com a variedade, usando os centros e o raio de cada hiperesfera da hierarquia dada pela árvore *16-Tree*. Utiliza-se o método de bisseção para computar as possíveis interseções. Essas interseções, caso existentes, são ordenadas segundo a distância à origem do raio. Para cada ponto de interseção é calculada a iluminação e é feito o *blending*. Também é analisado se o ponto pertence à silhueta ou ao bordo do domínio.

Nas próximas seções serão descritos os detalhes de cada etapa do processamento em GPU.

3.4.2 Uso de dois triângulos

O *Ray Casting* é feito no estágio do *fragment shader*, mas para que se ativem os fragmentos (pixeis) onde serão processados os raios, primeiro deve-se ativar o *vertex shader*. Como não existe nenhuma primitiva geométrica, então deve-se criá-la de tal forma que cubra todo o domínio projetado. Uma solução para o caso 3D é utilizar os lados de uma caixa envolvente que cubra o modelo. Assim, pode-se usar os lados da hipercaixa raiz da estrutura *16-Tree*. Mas essa solução não seria eficiente para o caso $4D$, pois na projeção dessa hipercaixa vários planos poderiam se superpor, o que causaria uma análise repetida de muitos pixeis.

Uma solução mais simples foi usar dois triângulos de um retângulo que cubra todo o domínio, e assim cada pixel apenas é analisado uma vez, ver Figura 3.4. Os vértices desses dois triângulos seriam, então enviados à GPU. No estágio do *vertex shader* do pipeline gráfico, os vértices são passados sem modificações para o *fragment shader* (ver código listado em 3.1). Dessa maneira, são ativados todos os pixeis dentro dos dois triângulos, pixeis esses que serão processados no *fragment shader* como será descrito na próximo seção.

Listing 3.1: Vertex Shader, passa para o próximo estágio os vértices sem

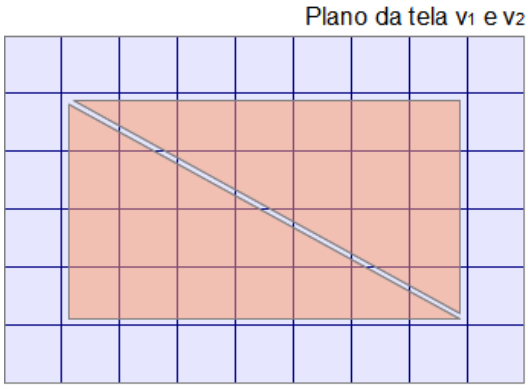

Figura 3.4: Dois triângulos usados para ativar os pixeis que serão processados.

modificações

```
in vec3 position;
void main() {
  gl_Position = vec4(position, 1.0);
}
```

3.4.3 Dentro do fragment shader

Uma vez dentro do *fragment shader*, temos que determinar qual pixel vai ser processado, tendo em consideração que o pixel $(0, 0)$ deve ficar no centro da tela, para isso se utiliza o código listado em 3.2.

Listing 3.2: Coordenadas do pixel

```
void main() {
  vec2 pixel = uTamPixel*(gl_FragCoord.xy-0.5*(uRes.xy-1.0));
  ...
}
```

Nesse código: *uTamPixel* representa o tamanho do pixel; *gl_FragCoord* são as coordenadas do pixel mantidas pelo *fragment shader*, cujos valores vão de 0 até a largura e altura da janela; e *uRes* representa o valor de largura e da altura da janela de visualização.

Logo, é preciso determinar qual é o origem e direção de cada raio no espaço $\mathbb{R}^4$. Já que está sendo considerada a projeção ortogonal, então o origem de cada raio está no mesmo pixel orientado segundo a janela de projeção. Esse plano é formado pelos vetores $\mathbf{v}_1$ e $\mathbf{v}_2$, os quais são as dois primeiras filas da matriz de rotação. Pode-se observar na esquerda da Figura 3.5.

O seguinte código faz o cálculo do origem dos raios.

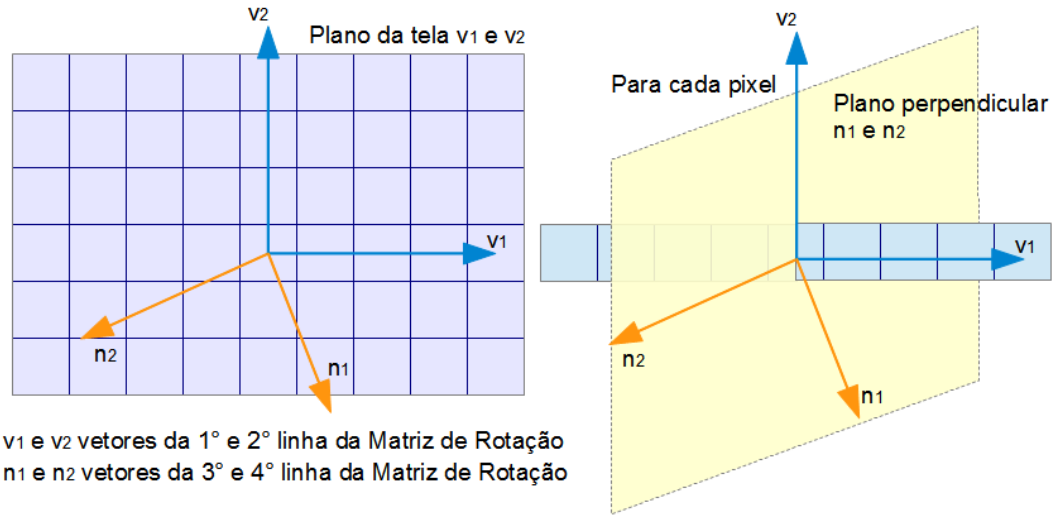

Figura 3.5: Plano perpendicular que passa por cada pixel, formado pelos vetores $\mathbf{n}_1$ e $\mathbf{n}_2$.

Listing 3.3: Cálculo do origem do raio

```
void main() {
  ...
  ray.ori = pixel.x * v1 + pixel.y * v2;
  ...
}
```

A próxima seção descreverá sobre a geração de raios.

3.4.4 Geração de raios

Por cada pixel passa um plano perpendicular, formado pelos vetores $\mathbf{n}_1$ e $\mathbf{n}_2$, que correspondem às duas últimas linhas da matriz de rotação A. Pode-se observar na direita da Figura 3.5.

Para visualizar a variedade deve-se gerar os raios no plano perpendicular a cada pixel. Para isso são executados os seguintes passos para gerar a direção de cada raio:

1. Criar dois valores aleatórios r_1 e r_2, sendo valores aleatórios uniformemente distribuídos em $[0, 1]$.

2. Calcular:
   $\alpha_1 = -1 + 2 * r_1$
   $\alpha_2 = -1 + 2 * r_2$

3. Calcular $\mathbf{d} = \alpha_1\mathbf{n}_1 + \alpha_2\mathbf{n}_2$

4. Normalizar $\mathbf{d} = \frac{\mathbf{d}}{\|\mathbf{d}\|}$

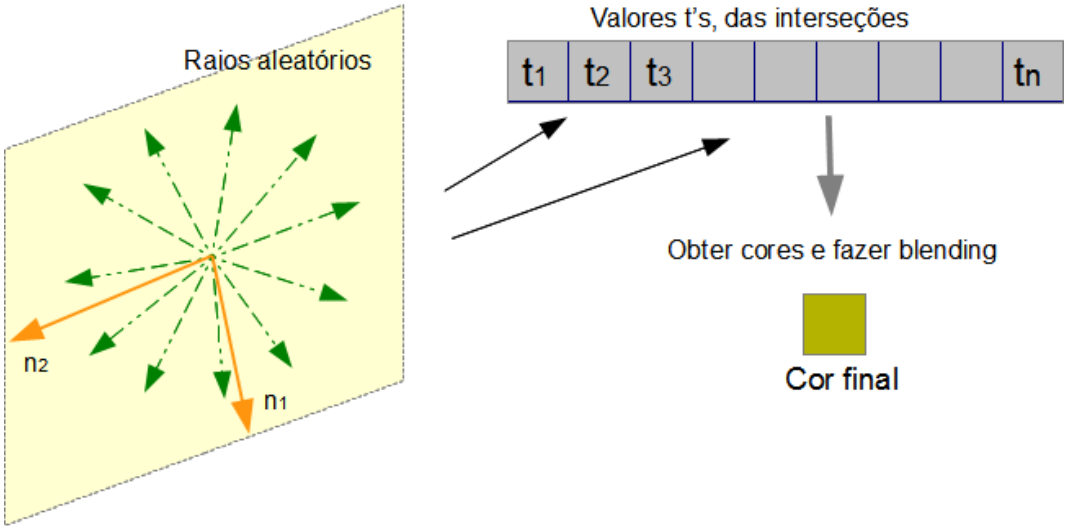

Figura 3.6: Raios aleatórios, e o cálculo da cor final.

Na parte esquerda da Figura 3.6 é ilustrada uma imagem dos raios gerados ao redor do plano $\mathbf{n}_1$ e $\mathbf{n}_2$. A próxima seção descreve a busca pelas raízes da função F que intercepta o raio.

3.4.5 Encontrando as raízes

Para cada raio lançado se deve encontrar as interseções com a variedade. Para isso usamos os conceitos descritos na seção 2.3, onde é descrito o método da bisseção para determinação das raízes de uma função. Vale observar que na linguagem do *shader* não é permitido escrever funções recursivas, e, por isso, o método da bisseção teve que ser implementado em sua versão iterativa, ver Algoritmo 3.1.

O algoritmo percorre a hierarquia das hiperesferas representadas pela árvore 16-Tree. Para cada hiperesfera, primeiramente determina-se quais são os valores de t_{min} e t_{max} através do cálculo de interseção analítico entre um raio e uma esfera, ver subseção 2.3.2. Esse cálculo é feito muito rapidamente, e com isso consegue-se procurar as raízes apenas nas hiperesferas que esse raio intercepta. Pode-se observar na Figura 3.7 a interseção entre um raio e uma esfera para determinar os valores de t_{min} e t_{max}. Se a hiperesfera tem filhos, ou seja não é folha, procurar por t_{min} e t_{max} nas hiperesferas filhas. Se a hiperesfera é uma folha, então com os valores de t_{min} e t_{max} pode-se encontrar a interseção do raio com a variedade implícita usando o método descrito na subseção 2.3.3, onde é utilizado o método de intervalos composto com o algoritmo da bisseção para encontrar as raízes, observar a Figura 3.8.

Se existem interseções, então são salvos os valores de t e os respectivos

Algorithm 3.1 Algoritmo da bisseção iterativo, implementado na GPU.

Require: $s_1 \neq s_2$, $numIter > 0$, $Approximation > 0$
Ensure: p_m fica muito perto da variedade F.

```
1: function BisectionIterative(p1, p2, s2)    ▷ s: Sinal do valor de F(p).
2:     for i ← 0 to numIter do
3:         pm ← (p1 + p2)/2
4:         fm ← F(pm)
5:         if abs(fm) < Approximation then
6:             break                          ▷ Sair porque se encontrou uma raiz.
7:         end if
8:         sm ← Sinal(fm)
9:         if sm ≠ s2 then
10:            p1 ← pm
11:        else
12:            s2 ← sm
13:            p2 ← pm
14:        end if
15:    end for
16:    return pm                              ▷ pm é uma raiz.
17: end function
```

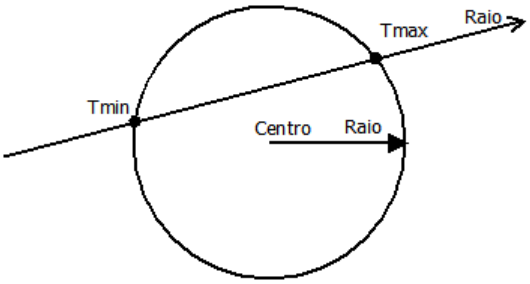

Figura 3.7: Calculando os valores t_{min} e t_{max}.

pontos de interseção. Pode acontecer que tenham muitas interseções, e por isso, apenas serão salvos os n menores valores de t com seus respectivos pontos.

O passo seguinte é obter as cores com a iluminação e fazer *blending* desses pontos, isso será descrito na próxima seção.

3.4.6 Iluminação

Para fazer a iluminação é utilizado o modelo de Phong, cujos conceitos básicos foram descritos na seção 2.5.

O vetor dos pontos de interseções dos valores t, é percorrido de trás para frente, onde para cada ponto é calculado a normal da variedade, e com o modelo de iluminação é obtido a cor do ponto, o qual é misturado como o seguinte ponto, segundo o equação de *blending*: $cor = cor_n(1.0 - \alpha) + cor_{n-1} * \alpha$ com $\alpha = 0.7$.

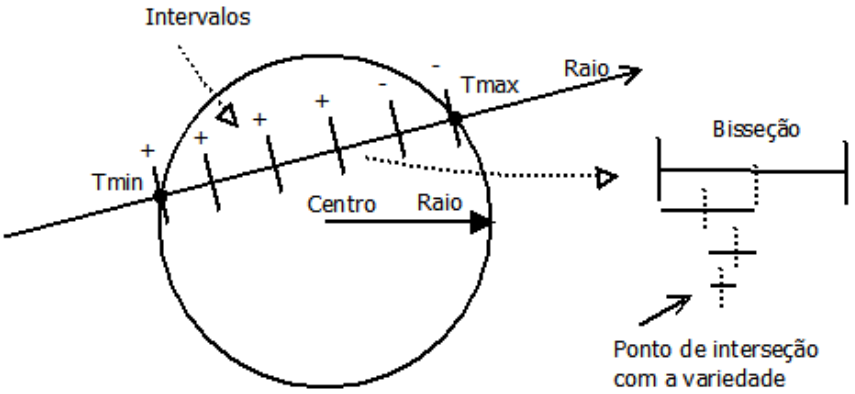

Figura 3.8: Método de intervalos composto com o algoritmo de bisseção.

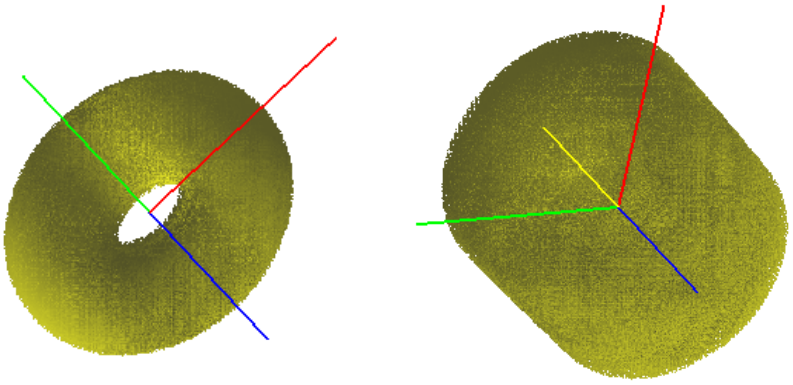

Figura 3.9: Iluminação da função $F(x, y, z, w) = w - (x^2 + y^2 + z^2 + R^2 - r^2)^2 + 4R^2(x^2 + y^2)$ com $R = 2$ e $r = 1$, nos ângulos: esquerda ($\theta_1 = 0.75, \theta_2 = 0.75, \theta_3 = 0.0, \theta_4 = 0.0, \theta_5 = 0.0, \theta_6 = 0.0$) e direita ($\theta_1 = 0.75, \theta_2 = 0.75, \theta_3 = 0.75, \theta_4 = 0.75, \theta_5 = 0.0, \theta_6 = 0.0$).

Na Figura 3.9, podem ser observados dois exemplos usando o modelo de iluminação com uma fonte de luz.

3.4.7 Silhueta e bordo do domínio

Com o propósito de melhorar a visualização são adicionados dois atributos: a silhueta e o bordo do domínio.

Silhueta

A silhueta é definida como sendo os pontos da variedade onde sua normal é perpendicular aos dois vetores observadores, isto é:

$$\nabla F(\mathbf{p}) \cdot \mathbf{n}_1 = 0 \ \& \ \nabla F(\mathbf{p}) \cdot \mathbf{n}_2 = 0 \tag{3-3}$$

A Figura 3.10 ilustra os pontos de silhueta.

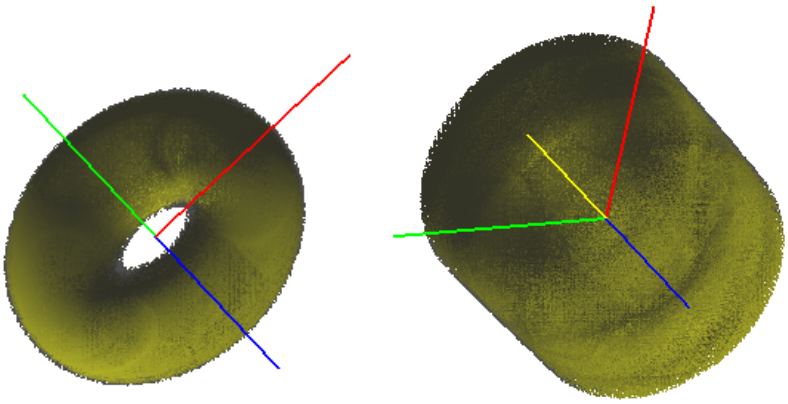

Figura 3.10: Silhueta da função $F(x, y, z, w) = w - (x^2 + y^2 + z^2 + R^2 - r^2)^2 + 4R^2(x^2 + y^2)$ com $R = 2$ e $r = 1$, nos ângulos: esquerda ($\theta_1 = 0.75, \theta_2 = 0.75, \theta_3 = 0.0, \theta_4 = 0.0, \theta_5 = 0.0, \theta_6 = 0.0$) e direita ($\theta_1 = 0.75, \theta_2 = 0.75, \theta_3 = 0.75, \theta_4 = 0.75, \theta_5 = 0.0, \theta_6 = 0.0$).

Bordo do domínio

As variedades podem estender-se ao infinito, mas elas são calculadas e analisadas em um domínio dado pelo usuário que corresponde a hipercaixa [**h**]. Dessa forma, a variedade implícita pode intersectar o borda da caixa e esses pontos de interseção serão realçados nas imagens geradas. Para isso, define-se uma distância mínima ao bordo. Os pontos que distam menos do que essa tolerância do bordo também serão realçados.

Na Figura 3.12 pode-se observar o processo de iluminação, considerando a silhueta, o bordo do domínio. Em todos os casos é utilizada somente uma fonte de luz.

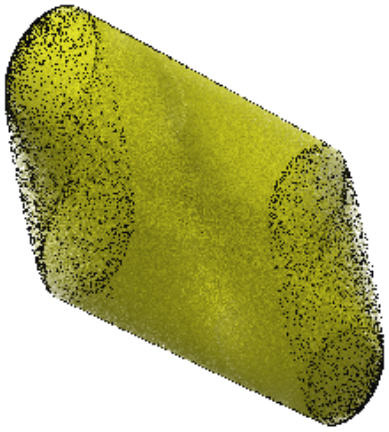

Figura 3.11: Bordo do domínio da função $F(x, y, z, w) = w - (x^2 + y^2 + z^2 + R^2 - r^2)^2 + 4R^2(x^2 + y^2)$ com $R = 2$ e $r = 1$, nos ângulos $(\theta_1 = 1.1, \theta_2 = 1.1, \theta_3 = 1.1, \theta_4 = 1.1, \theta_5 = 1.1, \theta_6 = 1.1)$, com 10 raios.

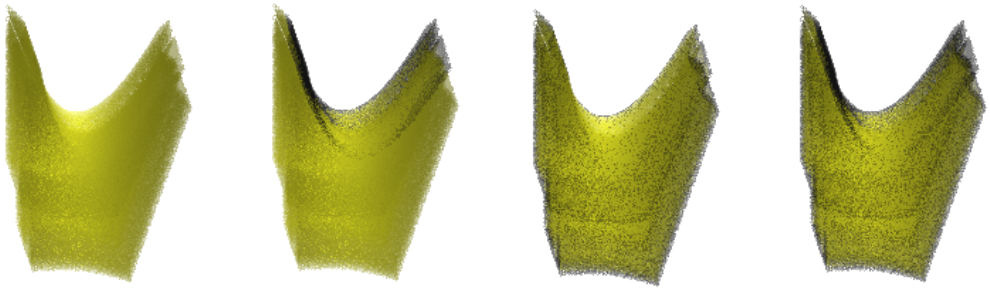

Figura 3.12: Processo de iluminação, começando da esquerda: iluminação só com fontes de luz, só desenhando a silhueta, só desenhando o bordo e a iluminação completa, variedade da função $F(x, y, z, w) = w - (x^2 + y^2 + z^2 + R^2 - r^2)^2 + 4R^2(x^2 + y^2)$ com $R = 2$ e $r = 1$, nos ângulos $(\theta_1 = 1.13, \theta_2 = 1.13, \theta_3 = 1.13, \theta_4 = 0.75, \theta_5 = 0.75, \theta_6 = 0.75)$, com 100 raios.