Projeto 5: Phong Shading com Transparência

Marcos Teixeira

4 de julho de 2018

1 Descrição

Um modelo de iluminação é um conjunto de equações que determinam quantitativamente a cor em um ponto da superfície de um objeto como uma função das propriedades da superfície da superfície e da luz incidente diretamente e indiretamente no ponto. Um modelo de iluminação local é aquele que leva em conta apenas a luz emitida pelas fontes virtuais de luz da cena, ou seja, iluminação direta. Um modelo global de iluminação também leva em conta a iluminação indireta, isto é, proveniente da inter-relação e refração da luz entre os objetos da cena.

Neste trabalho implementamos um refinamento do Trabalho 4, agora considerando opacidade para cada um dos objetos em cena e uma semi-transparência. Portanto, não vamos parar o raio de luz ao encontrar um ponto válido e visível do objeto, mas sim combinar com as cores dos próximos objetos ao longo do raio, até saturar uma opacidade acumulada.

2 Modelo de Iluminação de Phong com Opacidade

Um modelo local considera uma iluminação de um ponto de raios de luz provindos digitais como fontes de luz virtual. Um modelo local desconsidera a iluminação indireta resultante da inter-reflexo difusa e especular e da transmissao difusa e especular que ocorre entre superficies da cena e que afetam a iluminacao de um ponto. O modelo de Phong é o modelo local que considera apenas reflexao difusa e reflexao especular. A luz que chega aos olhos do observador combina a reflexão uniforme da luz ambiente com as reflexões difusa e especular da superfície visível do objeto.

Neste laboratório, iremos implementar o modelo de iluminação de Phong [1] um modelo simplificado no qual o observador e uma única fonte de luz se encontram na mesma posição, longe o suficiente do plano de visualização para obter uma projeção é ortogonal a cena. Como o objetivo é realizar um rendering pelo modelo de Phong, vamos considerar k superfícies visíveis de objetos semi-transparentes com opacidades $\alpha_j \in [0,1], j = 1,2,...,k$, sendo perfuradas por um raio P^j. Assim, as luzes refletidas $L(P^j)$ nas superfícies visíveis são combinadas usando as respectivas opacidades até a saturar da opacidade acumulada no raio [2].

Um exemplo do processo de iluminação pelo modelo de Phong é apresentado na Figura 1.

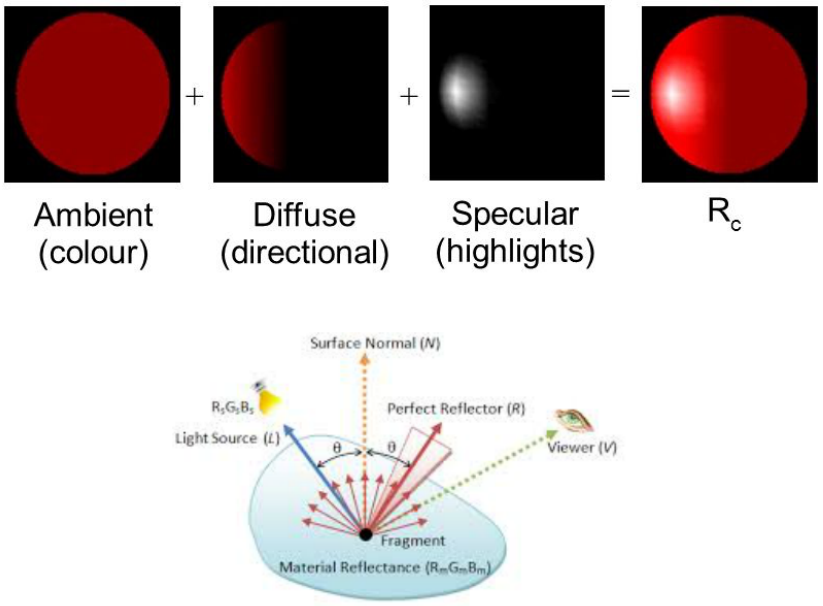

Figura 1: Exemplo do funcionamento do modelo de Phong e o resultado em um objeto 3D.

2.1 Luz ambiente

A iluminação e um ponto P derivada de da luz ambiente é

$$I = k_a L_a, \tag{1}$$

onde k_a, $0 \leq k_a \leq 1$ é o coeficiente de luz ambiente e L_a a cor de luz ambiente.

2.2 Reflexão difusa

Raios de luz que incidem em um ponto P de uma r=superfície refletem, após chocar-se com P, em uma direção arbitrária. Este fenômeno é proveniente da rugosidade do material da superfície, provocando a difusão dos raios de luz em torno do ponto P.

A iluminação de um ponto devido â iluminação difusa é definida pela lei dos cossenos de Lambert. No modelo de Phong, a iluminação da reflexão reflexiva é expressa pela seguinte equação:

$$I = k_d D(p) \cos\theta, \tag{2}$$

$$\cos\theta = (-N\dot{L}_l), \tag{3}$$

onde k_d, $0 \leq k_d \leq 1$ é o coeficiente de reflexão difusa, $D(p)$ é a tonalização baseada em profundidade (depth shading) e θ é o ângulo entre o vetor $\vec{v}$ e o vetor -$\vec{N}(p)$ normal a superfície em p. A tonalização pelo modelo de Phong só é executada para $0 \leq \theta \leq \pi/2$.

2.3 Reflexão especular

A reflexão especular ocorre em superfícies polidas e seu efeito depende da posição do observador. O componente especular só é calculado se $0 \leq \theta \leq \pi/4$. Esta característica é modelada no Phong através da seguinte equação:

$$I = k_s D(p) \cos 2\theta^{ns}, \tag{4}$$

$$\cos\theta = (-N\dot{L}_l), \tag{5}$$

2.4 Considerando opacidade e cor

A combinação das Equações 1, 2 e 6, para a iluminação direta de um ponto P, culminam no modelo geral de Phong descrito através de :

$$L(p) = k_a L_a + D(p)(k_d \cos\theta + k_s \cos 2\theta^{ns}), \tag{6}$$

A equação abaixo representa o resumo geral do modelo de iluminação de Phong considerando transparência e opacidade dos objetos em cena.

$$J(q) = \alpha_1 L(P^1) + \sum_{j=2}^{k} \alpha_j L(P^j) \prod_{i=1}^{j-1}(1-\alpha_i) \tag{7}$$

Note que o termo $\prod_{i=1}^{j-1}(1-\alpha_i)$ corresponde ao percentual de luz transmitida até a superfície j, em função das reflexões em superfícies anteriores. Além disso, $\prod_{i=1}^{j-1}(1-\alpha_i) < \epsilon$ representa a saturação admitida. Finalmente, o valor obtido do modelo de Phong após fazer os devidos cálculos de opacidade (considerando a transparência) é aplicado para os canais de cores do ponto P^j, gerando uma projeção colorida.

```
1 opac = gc->object[gc->label->val[idx]].opacity;
2 phong_val = PhongShading(gc, idx, N, dist);
3 phong_val = opac * phong_val * acum_opacity;
4
5 *red   += phong_val * gc->object[gc->label->val[idx]].red ;
6 *green += phong_val * gc->object[gc->label->val[idx]].green;
7 *blue  += phong_val * gc->object[gc->label->val[idx]].blue;
8
9 acum_opacity = acum_opacity * (1.0 -  opac);
```

Listing 1: Trecho do cálculo do Phong com opacidade e cor

Esta equação é implementada como mostramos no trecho de código 1. Todo este trecho está sendo executado no DDA somente para pontos válidos, visíveis e com opacidade

positiva. Primeiramente calculamos o Phong para o ponto observado. Em seguida, multiplicamos este valor pela opacidade do objeto e da opacidade acumulada. O resultado atualizado após este cálculo é multiplicado pelo R, G e B do objeto e somado ao red, green e blue que serão atribuídos a imagem final. Por fim, a opacidade acumulada é atualizada. É importante ressaltar que esse trecho de código só é executado caso o label do objeto obserado naquela iteração ainda não tiver sido processado, ou seja, visitamos no DDA pontos de um determinado objeto apenas uma vez.

3 Pipeline

Um modelo de Phong Shading é usado para simular os efeitos da luz interagindo em uma superfície, agora considerando transparência dos objetos e suas respectivas cores. O pipeline de execução do modelo de iluminação de Phong com opacidade pode ser expresso da seguinte maneira:

i Criar a imagem de saída R(D, D)

ii Criar a matriz de transformação M_{-1}

iii Definição das faces da cena

iv Criação da tabela de normais

v Criação do vetor de Depth Shading

vi Aplicar M_{-1} na matriz normal

vii Para cada ponto p_i do plano de visualização, faça

(a) Aplicar M_{-1} em p_i

(b) Se houver intersecção

i. Calcular os λ válidos que dam origem a $p1$ e pn

ii. Aplicar o DDA para percorrer os pontos $p_i \in p1, ..., pn$ em direção a cena

iii. Se opacidade_acumulada $> \epsilon$

A. Pare;

iv. Ao encontrar um ponto p_i válido, visível e que seja de um objeto ainda não analisado:

A. $L(i) = k_a L_a + D(p)(k_d \cos\theta + k_s \cos 2\theta^{ns})$

B. $J(i) = opacidade\_objeto * L(i) * opacidade\_acumulada$

C. Multiplicar $J(i)$ com cada canal RGB

D. Atualizar a opacidade acumulada

(c) Caso contrário, $R = 0$

viii Retornar R

, onde a matriz M_{-1} = T(N_x/2,N_y/2,N_z/2) * $R_x(theta_x)$ * $R_y(theta_y)$ * T(-D/2,-D/2,-D/2), com $D = \sqrt{nx^2 + ny^2 + nz^2}$. Como são várias estruturas que precisam ser manipuladas durante a execução do algoritmo, nós optamos por criar uma estrutura chamada GraphicalContext para facilitar a composição das funções. A estrutura do contexto gráfico é apresentado no pedaço de código 2.

```
typedef struct gc {
  ObjectAttributes *object;
  int               numberOfObjects;
  PhongModel       *phong;
  iftMatrix        *Tinv;
  iftVector         vDir;
  iftImage         *tde;
  iftImage         *scene;
  iftImage         *label;
  iftVolumeFaces   *faces;
  iftImage         *normal;
} GraphicalContext;
```

Listing 2: Estrutura do Contexto Gráfico

A variável *object* é do tipo **ObjectAttributes**, onde guardamos informações pertinentes de cada objeto em cena, tais como visibilidade, opacidade, cor, etc. Em *phong* ficam as informações referentes ao modelo de Phong, como as constantes (k_a, k_s, k_d, n_s) e o *Depth Buffer*, que é utilizado para transformar distância em cor. Os atributos *scene* e *label* representam a imagem de entrada e o mapa de rótulos, respectivamente. Para guardar as normais da imagem, seja ela calculada pela cena ou pelo **TDE**, alocamos a variável *normal*. As faces que encobrem a imagem ficam guardadas na variável *faces*, que corresponde a uma estrutura que armazena um vetor de pontos que possuem a normal e centro.

Para cada uma das 6 faces, realizamos os cálculos apresentados na equação abaixo. Para cada um dos lambdas obtidos, nós pegamos o menor e o maior válidos para conseguir obter $p1$ e pn. O valor de menor lambda será atribuído a equação da reta para encontrar $p1$ e o maior lamba para encontrar pn. Note que é necessário garantir que $(\vec{n'} \cdot \vec{n_j})$ não sejam 0 para que os testes das equações acima funcionem.

$$\lambda_j = ((c_j - p_0) \cdot \vec{n_j})/(\vec{n'} \cdot \vec{n_j}),$$
$$p_{'} = p_0 + \lambda \vec{n}$$

Posteriormente, o algoritmo DDA será utilizado para encontrar todos os pontos da linha entre $p1$ e pn. Além de encontrar o primeiro ponto p_i válido de um objeto da cena, calculamos o valor de $L(p)$. Para tal, vamos calcular as informações necessárias para o cálculo do Phong. Primeiramente, calculamos a distancia d de p_i ao observado (ponto P0), dado pela distância euclidiana. Obtemos o valor da normal pré calculada no ponto

p_i observado. A distância d transformada em brilho através de uma transformação linear $T(d)$ no qual, pixels mais distantes tem valores baixos e os mais pertos valores mais altos.

Uma vez tendo estas informações, chamamos a função *PhongShading*, que vai calcular o iluminação no ponto p_i encontrado. Calculamos inicialmente o valor o produto interno de V' e N, que resulta em $\cos\theta$. Avaliamos se $\theta \leq pi/2$ para ver se é possível realizar o shading e se $\theta \leq pi/4$ para verificar a necessidade do componente especular. Ao final teremos o valor para R, G e B para a imagem de saída que são multiplicados por 255., pois os resultados do DDA são normalizados em $[0, 1]$. Antes de realizar a normalização, também convertemos $B > 1$ para 1, $B \in R, G, B$. Por fim, fazemos uma conversão de RGB para YCbCr e atribuímos o resultado na imagem de saída.

Diferentemente do modelo de Phong para superfícies opácas, onde paramos o raio após encontrar uma região opaca com visibilidade *True*, aqui nós combinamos as luzes usando as respectivas opacidades até saturar a opacidade acumulada no raio. Estabelecemos com 1.0 como sendo a opacidade acumulada.

4 Implementação

Nesta seção são elencados alguns detalhes de implementação do projeto. Primeiramente, as constantes de Phong escolhidas foram as seguintes: $k_a = 0.1, k_d = 0.7, k_s = 0.2$ e $n_s = 5.$. O vetor de depth shading $T(d)$ foi construído a partir da distância máxima de um ponto P para a origem $p0$. Com isso, valores mais próximos da origem do vetor $T(d)$ tem maiores brilhos do que os mais próximos do fim de vetor. Também calculamos uma tabela de objetos, com atributos como sua cor, visibilidade, e opacidade.

As normais foram pré-calculadas e alocadas em uma tabela, para serem obtidas quando necessário. Uma função foi implementada para mapear o índice de um voxel com a tabela de normais. Experimentos com a estratégia de cálculo on-the-fly das normais também foram conduzidos, mas como demoraram demais (+10 min) para realizar o rendering, optamos por descartar esta ideia. O cálculo das normais foi realizado de duas maneiras: (i) baseado na cena e (ii) baseado na transformada de distância euclidiana e mapa de rótulos. Em ambos os casos foi considerado um raior $r = 3$ para definir a vizinhança em 3D na etapa de busca de pontos da borda. Na estratégia baseada na cena, um voxel p pertence a borda de um objeto da cena quando:

$$\exists q \in A_1(p), L(q) \neq L(p) e L(p) > 0 \tag{8}$$

Se o contraste entre os objetos (incluindo fundo) for bom, o vetor normal $\vec{N}_p$ pode ser aproximado pelo gradiente G_p. O gradiente em 3D é computado em todas as 6 direções.

$$\vec{G}_p = \sum_{\forall q \in A(p)} [I[q] - I[p]] \tag{9}$$

Nesta estratégia é preciso sempre verificar se a normal $\vec{N}_p$ está apontando para dentro ou para fora o objeto.

Na estimativa baseada na transformada de distância D_E^p, associamos a cada voxel $p \in D_i$ a uma distância ao voxel mais próximo na borda do objeto cujo rótulo é $L(p) \neq 0$.

$$\hat{L} = (D_i, L), D_i \implies Z \tag{10}$$

$$\hat{D}_E = (D_i, D_p) \tag{11}$$

onde $\hat{D}_E$ armazena iterativamente a distancia quadrada para os valores da borda. Após obter a transformada de distâncias, realizamos um cálculo semelhante ao da Equação 9, mas agora com a diferença entre $\hat{D}_E(p)$ e I[p]. Nessa forma de estimativa da normal, sempre negamos o seu valor no final pois sabemos que sua direção estará sempre apontada pra dentro do objeto.

5 Instruções

Para execução do programa **Phong.c**, siga a seguinte formatação :

```
1.  $ ./Phong input.csn
              input-label.scn
              tilt
              spin
              config.txt
              output-image.png
```

onde *config.txt* é o arquivo que contém as informações de cada objeto (cores, visibilidade e opacidade) e *output-image.png* é o arquivo de saída, que será gerado ao fim do programa na pasta **data**. Cada linha do arquivo de configuração representa as características de um objeto em cena, seguindo rigorosamente esta formatação:

1. vermelho, verde, azul, opacidade, visibilidade

no qual vermelho, verde, azul e opacidade são escalares [0.,1.] e visibilidade é um inteiro binário 0 ou 1. Note que a quantidade de linhas neste arquivo deve ser a mesma da quantidade de objetos em cena.

6 Resultados

Nesta seção apresentamos os resultados obtidos na imagem de teste **lungs.scn**, que fica na pasta *data*, onde será escrito o resultado final também. As imagens de saída são da forma (D, D), onde D é igual a $D = \sqrt{nx^2 + ny^2 + nz^2}$, representando a diagonal da imagem. Para padronizar as figuras nesse relatório, dado que esse D depende de cada imagem, nós limitamos a altura para 5cm e a largura para 7.5cm. Uma normalização de (0,255) é

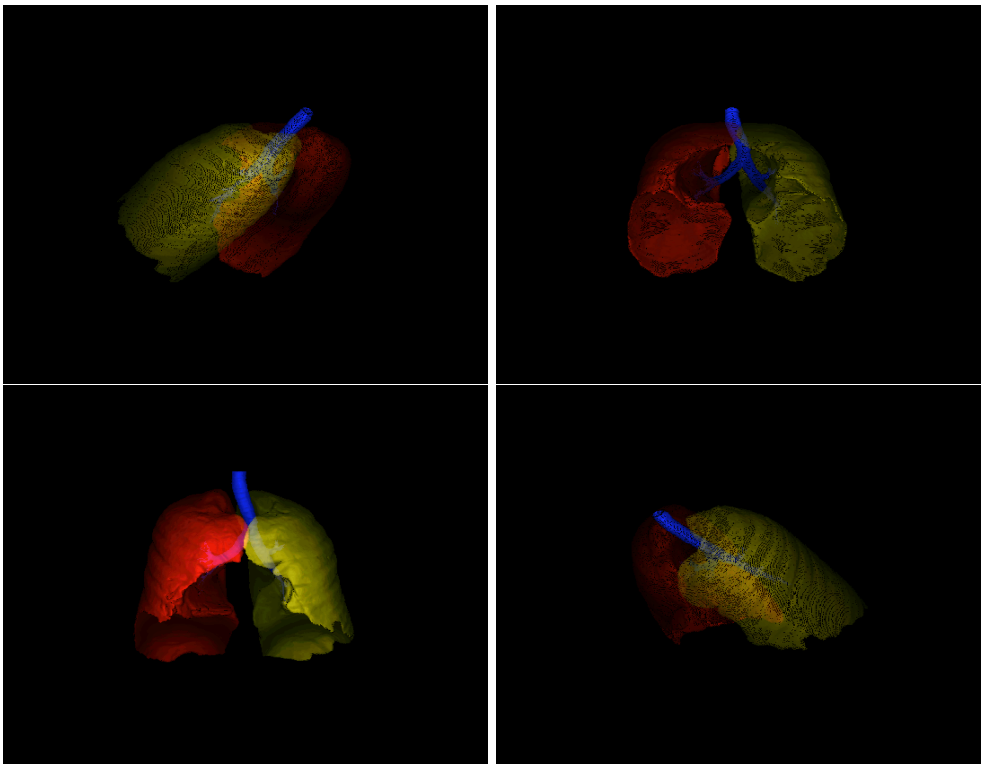

Figura 2: Resultados do Phong com opacidade e transparência utilizando estimativa da normal pela cena.

feita após a finalização do algoritmo de tonalização de Phong, para melhor visualização. Inicialmente, o plano de visualização é reposicionado em $z = D/2$, para afastar um pouco da visão do observador.

Na Figura 2 mostramos os resultados utilizando estimativa das normais pela cena, para rotações de $R_x = 45, R_y = 45$, $R_x = 45, R_y = 18$,$R_x = 90, R_y = 180$, $R_x = 135, R_y = 135$, respectivamente. É visível que existem "buracos"na projeção, evidenciados pelas regiões mais escuras. Esse fenômeno se dá pelo fato da estimativa baseado na cena ser tomada observando o vetor gradiente de intensidade da cena, o que pode ser muito impreciso em casos onde a superfície apresenta baixa homogeneidade dos pixel com relação aos seus vizinhos. Outro problema é que precisamos verificar ao criar a tabela de normais, para cada normal, se ela está apontando pra dentro ou para fora de um objeto, o que dependendo da maneira como é feito, pode gerar essas falhas apresentadas nas figuras.

Posteriormente, fizemos experimentos com a estimativa baseada na transformada de distâncias e mapa de labels. Nesta abordagem, estimamos as normais baseado no vetor gradiente TDE sinalizado ao redor da borda do objeto observado. Os resultados estão apresentados na Figura 3, para rotações de $R_x = 45, R_y = 45$, $R_x = 45, R_y = 18$,$R_x = 90, R_y = 180$, $R_x = 135, R_y = 135$. Para encontrar as bordas com TDE, utilizamos um raio

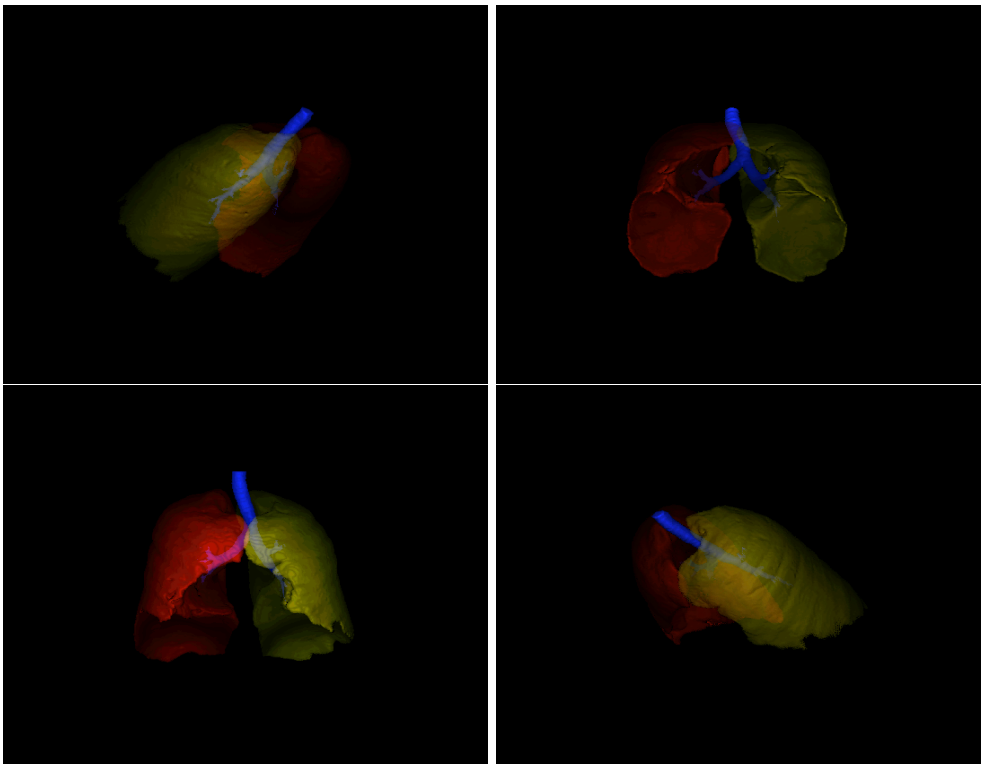

Figura 3: Resultados do Surface Rendering com iluminação de Phong utilizando estimativa da normal pela Transformada de Distância Euclidiana e Mapa de Objetos.

$r = 3.0$ na função *iftSpheric*. As configurações dos objetos em cena foram as seguintes:

1. red = 0.7, green = 0.7, blue = 0.1, opacidade = 0.3, visibilidade = 1.
2. red = 0.9, green = 0.1, blue = 0.1, opacidade = 0.3, visibilidade = 1.
3. red = 0.1, green = 0.1, blue = 0.8, opacidade = 1., visibilidade = 1.

Como esperado, os resultados ficaram bem melhores que a abordagem de estimativa baseada na cena. Esta técnica, por requerer o cálculo da transformada de distâncias euclidianas, demora um pouco mais a ser executada. No entanto, isso se reflete em um resultado bem melhor que a normal pela cena, eliminando os buracos vistos na Figura 2.

Por fim, também apresentamos os resultados obtidos com a imagem de crânio (**skull.csn**). Nesse experimento, utilizamos as seguintes configurações para o único objeto em cena: red = 0.6, green = 0.1, blue = 0.8, opacidade = 1., visibilidade = 1. Os resultados são apresentados na Figura 4.

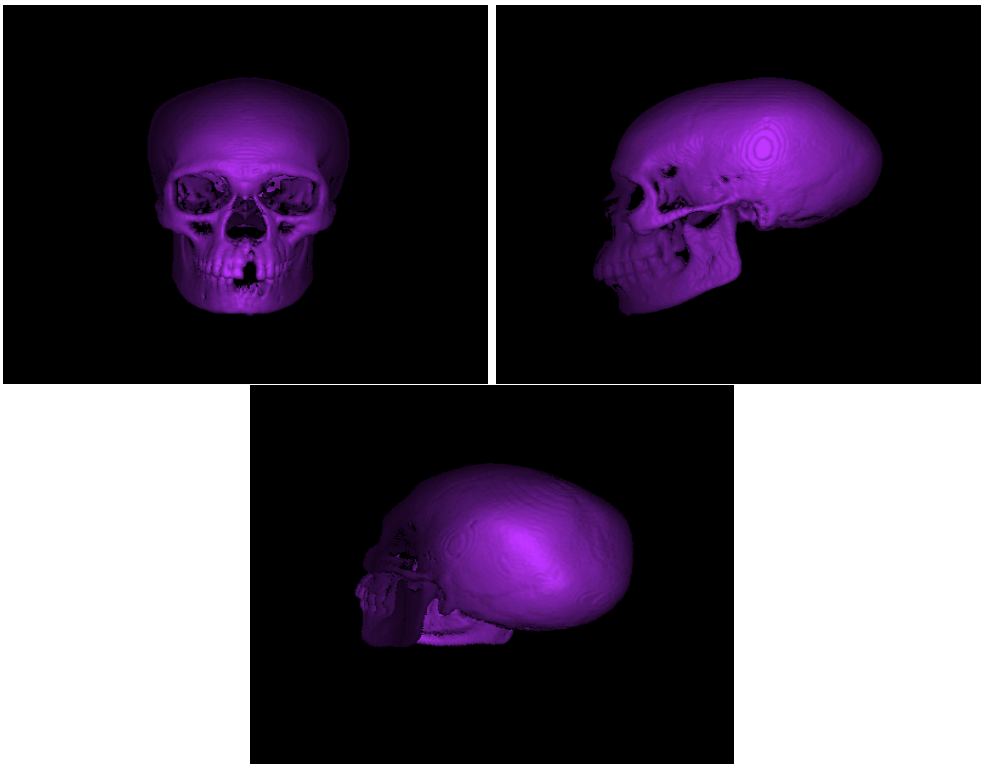

Figura 4: Resultados para imagem *skull.scn*.

7 Conclusão

Neste trabalho foi apresentado uma implementação para o algoritmo de surface rendering baseado no modelo de iluminação de Phong com opacidade e transparência entre os objetos. Através da teoria vista em sala de aula, fizemos uma modelagem que levasse em conta todos os passos do algoritmo de forma simples e objetiva. Os experimentos foram conduzidos levando em consideração duas formas de se realizar a estimativa das normais: pela cena e pela transformada de distâncias (mapa de objetos).

Este trabalho apresentou alguns desafios, mesmo sendo muito parecido com o Trabalho 4. O principal foi relacionado ao DDA, para considerar apenas um ponto de cada objeto ao longo do traçado. Isso foi resolvido com um buffer dos objetos já visitados. Através desta correção, os resultados foram normalizados. De qualquer forma, foi um aprendizado relevante para concretizar a teoria vista em sala.

Algumas observações técnicas para a implementação. Foi necessário utilizar a biblioteca compilada para Mac OS disponibilizada pelo monitor para o código. Além disso, mudamos também a versão do gcc no Makefile para o gcc-7, para poder funcionar.

Referências

[1] B. T. Phong, "Illumination for computer generated pictures," *Communications of the ACM*, vol. 18, no. 6, pp. 311–317, 1975.

[2] A. Falcão, "Notas de aula mo815 visualização de imagens volumétricas," *Unicamp*, 2018.