INF1608 – ANÁLISE NUMÉRICA

Visualização Volumétrica

Integrantes: Marcelo Costalonga Cardoso - 1421229
Pedro Sousa Meireles – 1510962

Introdução:

O objetivo do trabalho foi gerar uma imagem 3D de uma tomografia computadorizada de um crânio, no formato PGM em diferentes tons de cinza, a partir do resultado obtido de uma visualização volumétrica por traçado de raio do dado de CT scan. Neste modelo de visualização monocromática, a intensidade de luz , de cada pixel, foi dada pela Integral de Renderização Volumétrica:

$$I = \int_0^L \tau(d(s)) e^{-\int_0^s \tau(d(t))dt} ds$$

Sendo L o comprimento do raio traçado para cada pixel, $d(.)$ a densidade do volume e $\tau(.)$ a opacidade em função da densidade ao longo do raio (função de transferência). Dessa forma, quanto maior for a densidade, mais opaco será o pixel.

A função de transferência utilizada é dada por:

$$\tau(d(t)) = 0.0, \text{ se } d(t) < 0.3$$

ou

$$\tau(d(t)) = 0.05(d(t) - 0.3), \text{ caso contrário}$$

Fazendo com que os pixels pouco opacos fiquem mais escuros que o normal, ressaltando o contraste entre os tons da imagem.

Desenvolvimento:

Utilizamos um vetor unsigned char para armazenar os dados do CT scan, de tamanho $256 \times 256 \times 99$, equivalente à $(2 \times Nx) \times Ny \times Nz$, onde Ny e Nx representam as dimensões de cada imagem e Nz o número de "imagens" (fatias). Dessa forma, os índices (i, j, k) do vetor data foram acessados através de $(kNy(2Nx) + jNx + i)$.

Inicialmente definimos os valores de $Nx = 128$, $Ny = 256$ e $Nz = 99$ como variáveis globais e implementamos a função de transferência:

```
double transfer_function (int i, double j, int k, unsigned char *data);
```

A primeira versão dessa função considerava apenas um valor inteiro de h, o passo de integração. Depois implementamos ela de forma que aceitasse h como um double, fazendo uma interpolação linear, ligando os pontos do vetor CTscan, relativos aos índices (i, j, k) e $(i, j+1, k)$, por uma reta, podendo assim acessar uma aproximação do índice (i, jf, k), sendo jf um índice double referente ao eixo Y. Por fim, implementamos a função de transferência utilizando interpolação pelo método de diferenças finitas de Newton, apresentando melhor precisão entre todos os métodos.

Depois, implementamos a parte da função de intensidade, que representa ao cálculo do expoente elevado à integral $-\int_0^s \tau d(t))dt$, presente na equação mostrada no começo:

double intensity_function (int i, double j, int k, unsigned char *data);

Para resolver esta integral que vai de 0 até S, utilizamos um passo correspondente ao valor de h. A cada passo era chamado o Método de Simpson (na primeira versão utilizamos o Método de Simpson simples e na segunda versão o Método Adaptive de Simpson), passando como parâmetro a função de transferência, com limite inicial do intervalo ("prev", que inicialmente começava com valor igual a zero), e limite final do intervalo (que começava com o próprio valor de h). Assim a cada iteração, era incrementado o valor de h a cada um dos limites, até que chegasse ao valor de S.

Posteriormente, implementamos o resto da função intensidade, correspondendo ao cálculo do da outra integral:

double intensity (int i, int k, unsigned char *data);

Essa função calcula a integral de 0 até L (sendo L nesse caso o tamanho do comprimento do eixo Y), que, assim como a função acima, a cada passo de h era chamado o Método de Simpson (na segunda versão usamos o Método de Simpson Adaptive), e depois era incrementado os limites do intervalo de h. Porém, nesse caso, havia duas chamadas deste método a cada iteração, uma avaliando o índice $(2i, 0, k)$ e outra avaliando $(2i+1, 0, k)$, para que no final, a intensidade de um pixel fosse dada pela média dos dois pontos. Isso foi feito para que a imagem final tivesse dimensões 128x99. Caso fosse feita uma integral para cada ponto $(i, 0, k)$, utilizando $n_x = 256$, a imagem obtida estaria esticada no eixo x. As duas imagens serão comparadas posteriormente neste relatório.

A função retorna a intensidade de cada ponto (i, k) da imagem resultante, viabilizando a produção do arquivo PGM final.

Análise:

O programa gerou corretamente a imagem final. A princípio, utilizando-se $n_x = 256$, a imagem gerada ficou distorcida, resultando em uma figura esticada no eixo x. Após a redução de n_x para 128, a imagem obtida foi igual à esperada. Isso se deve ao fato de ter sido usada a média da intensidade de dois pixels consecutivos no eixo x.

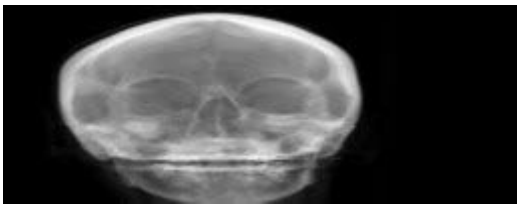
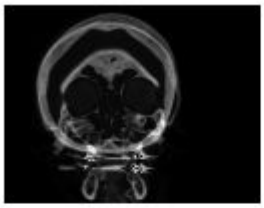


Figura 1: Imagem esticada ($n_x = 256$)

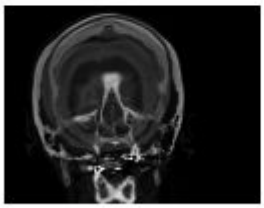
Figura 2: Imagem original ($n_x = 128$)

Um dos parâmetros que altera a qualidade da imagem gerada é o tamanho do passo de integração. Por padrão, utilizou-se passo $h = 4.5$, que gerou a imagem acima. Na figura abaixo pode-se ver diferentes resultados para diferentes passos. É possível perceber que, quanto maior o passo de integração, pior é a qualidade da imagem gerada.

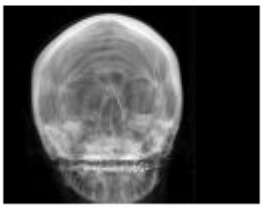


$h > L$

$h = 34.5$



$h = 200.5$



$h = 19.5$

$h = 120.5$

$h = 9.5$

$h = 60.5$

$h = 4.5$

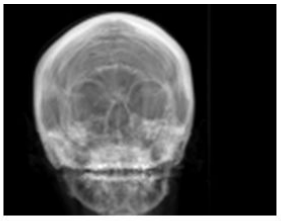
O programa foi executado de duas maneiras: utilizando Método de Simpson simples e Método de Simpson Adaptativo. Apesar de, na teoria, o adaptativo dar melhores resultados, na maioria dos casos, não foi possível verificar diferença significativa nas imagens geradas pelos dois métodos para um mesmo passo de integração. No exemplo abaixo foi utilizado $h = 4.5$.

Simpson Adaptativo

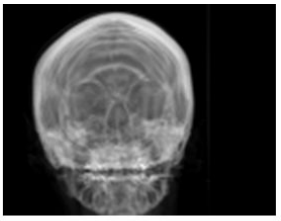
Simpson Simples

Uma vantagem do método adaptativo é que ele aceita uma tolerância. Porém, devido à grande quantidade de dados e de funções recursivas presentes no programa, utilizar uma tolerância muito baixa pode acarretar erro de excesso de pilha. Pois, para cada chamada do método de Simpson que excedesse a tolerância, ocorreria pelo menos mais duas chamadas, recursivas, do método, subdividindo o intervalo em duas partes, podendo assim causar este erro de stack overflow. A menor tolerância utilizada que não gerou erro foi de 0.5×10^{-2}, porém esse valor de tolerância não admitia passos muito altos, nesse caso o maior valor era de aproximadamente $h = 25.71$, acima disso ocorria erro.

Para esse valor de $h = 25.71$, foi possível visualizar uma leve diferença entre os métodos simples e adaptativo:



Simpson Adaptativo
(com tol = 0.005)



Simpson Simples

Além disso, para valores de tolerância muito baixos, foi possível perceber também, de forma clara, a diferença de custo computacional a entre os métodos. Visto que o processamento da imagem, utilizando o método de Simpson Simples, era mais rápido do que utilizando o método de Simpson Adaptativo, e quanto menores fossem os valores da tolerância e dos passos, mais lento era o processamento da imagem.

Todo o projeto foi realizado utilizando-se raios na direção (0, 1, 0). Porém, com algumas alterações no programa, seria possível gerar imagens para raios em qualquer direção.

Para isso, deveria-se modificar a função de intensidade, que precisa receber o vetor diretor do raio. A integral deve ser feita na direção do novo raio. Para isso, deve-se encontrar o plano perpendicular ao raio recebido e realizar uma mudança de eixos. Assim, a imagem seria gerada no novo eixo, imprimindo, no caso de um raio na direção (1, 0, 0), um crânio em perfil.