Relatório 3 de Introdução a Computação Gráfica

Hialo M. Carvalho[1]

[1]Faculdade UnB Gama (FGA) – Universidade de Brasília (UnB)
Caixa Postal 8114 – 72.405-610 – Gama – DF – Brasil

hialomuniz@gmail.com

1. Descrição da solução proposta

O problema proposto consiste na apresentação de uma imagem formada por uma malha triangular. Esta malha triangular consiste em um arquivo do tipo .OFF (*Object File Format*) com os pontos e faces que formam a figura. A imagem deve ser apresentada em uma janela SDL, de tamanho 800x600, e a renderização deveria ser feita por meio de funções da biblioteca OpenGL.

A solução desenvolvida consiste em um sistema de Computação Gráfica escrito na linguagem C++, que recebe como parâmetros o arquivo .OFF que contém os pontos e faces que formam a malha triangular, e um parâmetro que define se a construção da figura será feita por meio de linhas (GL_LINES) ou triângulos (GL_TRIANGLES). Para a renderização da malha, foram criadas algumas estruturadas de dados para armazenas os dados necessários: uma estrutura de dados *vertex*, que contém os pontos x, y, z, uma estrutura de dados Face, que contém os índices para os vértices da face, e uma estrutura de dados *mesh*, para a renderização da malha. Tais estruturas serão detalhadas mais a frente.

Para realizar a renderização da figura, primeiramente o sistema aloca uma estrutura de dados do tipo *mesh*, e realiza a leitura do arquivo passado como parâmetro, armazenando em suas estruturas os vértices, valores RGB e faces necessárias para a renderização da malha. Devido à especificação do problema, a função de leitura funciona apenas para arquivos .OFF, devido a sua estrutura de leitura. Para a leitura de outros arquivos, como o tipo .3DS, é necessária a modificação da função de leitura.

Após a leitura e armazenamento dos vértices e faces, duas funções são usadas para modificação dos vértices: uma função para redimensionamento da malha e outra para translação da malha. Devido às suas características matemáticas, tais funções são genéricas, ou seja, não importa o tamanho da malha, ela sempre será redimensionada e transladada para o centro da janela.

A função de redimensionamento é executada primeiro. Esta função consiste dos seguintes passos:

a) Encontrar o máximo e mínimo de cada um dos pontos que formam um vértice. Ou seja, é necessário encontrar os seguintes parâmetros: *Xmax, Xmin, Ymax, Ymin, Zmax, Zmin*. Estes valores fazem parte da estrutura de dados *mesh*.

b) Calcular o delta de cada ponto que compõem o vértice. Este delta consiste na diferença entre o máximo e o mínimo de x, y e z. Então, *deltaX, deltaY e deltaZ* são obtidos.

Estes dois primeiros passos são executados pela função *gettingDelta().*

c) Cada um dos pontos armazenados na estrutura de dados são divididos pelo seu delta equivalente. Ou seja, cada ponto x é dividido por *deltaX* e assim sucessivamente.

Após realizado o redimensionamento, a função de translação é executada. Esta função é semelhante à de redimensionamento, consistindo de três passos:

a) O mínimo e o máximo de cada ponto do vértice são recalculados.

b) O delta de cada ponto também é recalculado.

Estes dois primeiros passos são executados pela função *gettingDelta().*

c) Cada um dos pontos armazenados na estrutura de dados são subtraídos pelo seu delta equivalente dividido por 2. Ou seja, cada ponto x é subtraído por *deltaX/2*, e assim sucessivamente.

Estas duas funções garantem que a figura sempre será renderizada no centro da janela. É uma forma mais interessante de realizar estas duas tarefas do que usar funções como *glScale(),* ou *glTranslate().*

Então, a janela SDL é criada, por meio da função *createWindow().* Esta função cria a janela por meio de funções nativas do SDL, como *SDL_CreateWindowAndRenderer(), SDL_GL_CreateContext(),* além de setar o *viewport* por meio da função *setViewport().*

Hora de renderizar a malha. A renderização é feita pela função *displayRender()* e *renderMesh().* A primeira limpa os buffers usados pelo OpenGL e renderiza a malha na janela SDL. A segunda é a responsável pela renderização da malha, recebendo como parâmetros a malha a ser renderizada e o parâmetro que indica se a malha será renderizada usando linhas ou triângulos/quadrados (1 ou 2). Apesar do problema solicitar a renderização apenas por triângulos, como grande parte dos arquivos .OFF também possuem quadrados a serem renderizados, é interessante que a solução contenha esta implementação.

Após a renderização na tela, a memória alocada para a estrutura de dados da malha é liberada e o programa encerra a sua execução.

A solução consiste em um sistema contendo três arquivos principais:

- mesh.h: Header contendo a implementação das principais estruturas de dados usadas para a modelagem da malha triangular: *vertex, color, face e mesh.*

- mesh.cpp: Arquivo contendo a implementação das funções que compõem o sistema, desde a criação da janela SDL e da criação do *viewport* OpenGL, até as funções de leitura do arquivo, alocação da estrutura de dados da malha, redimensionamento, translação e modelagem da malha.

- main.cpp - Arquivo principal, contendo a função principal do sistema.

Juntamente com os arquivos que compõem o sistema, estão presentes diversos arquivos .OFF, para testes.

Para a compilação do sistema, foi usada o seguinte comando:

g++ main.cpp mesh.cpp -o exec -lglut -lGLU -lGL -lSDL2 -W -Wall -pedantic -ansi

Como pode ser visto, para compilar o código é necessário ter as bibliotecas GLU, GLUT, GL e SDL2.0 presentes na máquina. O código foi compilado em uma máquina com Linux Ubuntu 12.04.

Após a compilação, para executar código basta usar o seguinte comando:

./exec <arquivo_texto><modo_de_modelagem>

Aonde o modo de modelagem consiste em:

Modo "1" - A malha será modelada usando apenas linhas.

Modo "2" - A malha será modelada usando triângulos e quadriláteros.

O resultado da execução pode ser visto abaixo:

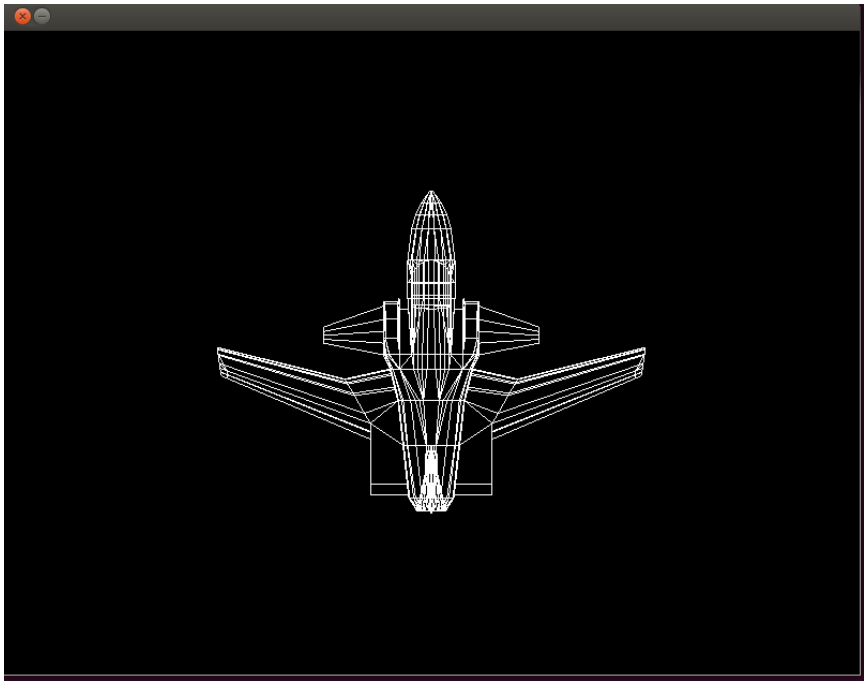

Figura 1. Renderização da malha usando linhas (modo 1).

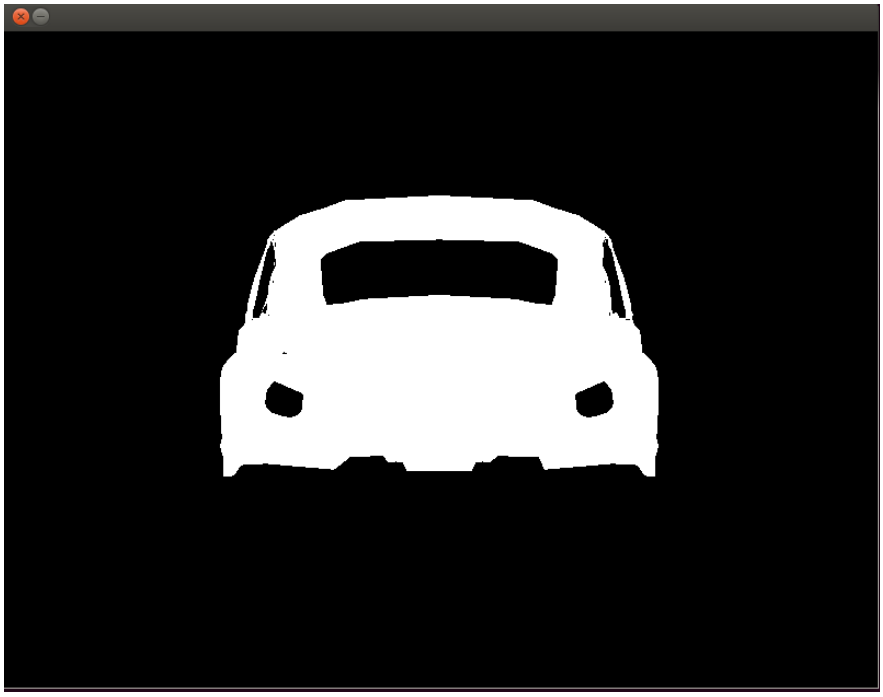

Figura 3. Renderização da malha usando triângulos/quadrados (modo 2).

2. Características e parâmetros da solução proposta

Para a construção deste sistema, foram usadas quatro estruturas de dados. A implementação de tais estruturas está presente no arquivo *mesh.h*:

a) *struct vertex (float x, y, z):* Uma estrutura de dados do tipo *vertex*. Como o nome sugere, esta estrutura armazena cada ponto que forma um vértice (x, y e z).

b) *struct color (int R, G, B, alpha):* Estrutura de dados que armazena as cores lidas no arquivo.

c) *struct face (int v1, v2, v3, v4, vertex_quantity):* Estrutura de dados que armazena as faces que compõem a malha triangular. Os parâmetros v1, v2, v3 e v4 são índices para o vetor de vértices.

Quando a face é um triângulo (ou seja, a quantidade de vértices, último parâmetro, é igual a 3), o parâmetro v4 é nulo.

d) *struct mesh (vertex *list_vertex, face *list_faces, color *list_color, int nvertex, int nface, int ncolor, float xmin, xmax, ymin, ymax, zmin, zmax)*: Estrutura principal do sistema. Os parâmetros *list_vertex, list_faces e list_color* são ponteiros para o vetor de índices e faces usados para a renderização da malha. Os parâmetros *nvertex, nface e ncolor* são usados como tamanho para a alocação dinâmica destes vetores, e os valores *float* são os máximos e mínimos de cada ponto do vértice, usados para calcular o redimensionamento e translação da malha.

3. Tipos de arquivos de malha 3D

Neste problema, a implementação do sistema foi feita com base em um arquivo do tipo .OFF. Porém, existem diversos tipos de arquivos de malha 3D além deste. Como exemplos, pode-se citar os arquivos do tipo .3DS (), .OBJ, e muitos outros.

Arquivo .3DS: Um dos tipos de arquivos mais populares, apesar da sua idade. É o formato de arquivo padrão do antigo *Autodesk 3D Studio*, que hoje se chama *Autodesk 3DS Max*. Comparado ao tipo de arquivo padrão do 3DS Max (arquivo. MAX), o arquivo 3DS guarda apenas informações essenciais, como coordenadas dos vértices, posição dos vértices na malha e etc. Já o arquivo .MAX é bem mais completo, com informações como texturas, animações, luz e etc.

O arquivo .3DS é um arquivo do tipo binário. O arquivo é dividido em blocos, aonde cada bloco de dados contém um identificador e o tamanho do bloco, a fim de facilitar a leitura do arquivo. Uma lista com os blocos de dados mais comuns que um arquivo .3DS contém pode ser vista abaixo:

```
0x4D4D // Main Chunk
├─ 0x0002 // M3D Version
├─ 0x3D3D // 3D Editor Chunk
│  ├─ 0x4000 // Object Block
│  │  ├─ 0x4100 // Triangular Mesh
│  │  │  ├─ 0x4110 // Vertices List
│  │  │  ├─ 0x4120 // Faces Description
│  │  │  │  ├─ 0x4130 // Faces Material
│  │  │  │  └─ 0x4150 // Smoothing Group List
│  │  │  ├─ 0x4140 // Mapping Coordinates List
│  │  │  └─ 0x4160 // Local Coordinates System
│  │  ├─ 0x4600 // Light
│  │  │  └─ 0x4610 // Spotlight
│  │  └─ 0x4700 // Camera
│  └─ 0xAFFF // Material Block
│     ├─ 0xA000 // Material Name
│     ├─ 0xA010 // Ambient Color
│     ├─ 0xA020 // Diffuse Color
│     ├─ 0xA030 // Specular Color
│     ├─ 0xA200 // Texture Map 1
│     ├─ 0xA230 // Bump Map
│     └─ 0xA220 // Reflection Map
│        │  /* Sub Chunks For Each Map */
│        ├─ 0xA300 // Mapping Filename
│        └─ 0xA351 // Mapping Parameters
└─ 0xB000 // Keyframer Chunk
   ├─ 0xB002 // Mesh Information Block
   ├─ 0xB007 // Spot Light Information Block
   └─ 0xB008 // Frames (Start and End)
      ├─ 0xB010 // Object Name
      ├─ 0xB013 // Object Pivot Point
      ├─ 0xB020 // Position Track
      ├─ 0xB021 // Rotation Track
      ├─ 0xB022 // Scale Track
      └─ 0xB030 // Hierarchy Position
```

Figura 3. Exemplo de arquivo .3DS.

Arquivo .OBJ: É um formato de arquivo desenvolvido pela Wavefront Technologies. Hoje em dia, pode ser considerado o formato de arquivo padrão para modelagem 3D, devido a sua simplicidade.

O formato do arquivo .OBJ é muito parecido com o formato .OFF. A principal diferença é que o arquivo .OBJ contém um identificador como primeiro caractere de cada linha, identificando se é um vértice, face ou normal, facilitando a leitura, pois basta ler o caractere de cada linha e armazenar os dados de acordo com o seu tipo. Um exemplo simples de um arquivo .OBJ pode ser visto abaixo:

```
# List of Vertices, with (x,y,z[,w]) coordinates, w is optional and defaults to 1.0.
v 0.123 0.234 0.345 1.0
v ...
...
# Texture coordinates, in (u ,v [,w]) coordinates, these will vary between 0 and 1, w is optional and default to 0.
vt 0.500 1 [0]
vt ...
...
# Normals in (x,y,z) form; normals might not be unit.

vn 0.707 0.000 0.707
vn ...
...
# Parameter space vertices in ( u [,v] [,w] ) form; free form geometry statement ( see below )
vp 0.310000 3.210000 2.100000
vp ...
...
# Face Definitions (see below)
f 1 2 3
f 3/1 4/2 5/3
f 6/4/1 3/5/3 7/6/5
f ...
...
```

Figura 4. Exemplo de arquivo .OBJ.

4. Lições aprendidas no processo

As principais lições aprendidas na leitura e execução desta lista foram:

- A utilização da biblioteca SDL 2.0 para a criação da janela necessária para que o usuário possa ver o resultado final da modelagem.

- A utilização de estruturas de dados para auxiliar na renderização da malha, deixando o código mais modularizado e fácil de ser entendido.

- A criação da função de leitura do arquivo .OFF, já que foi necessário o entendimento do formato do arquivo para que o mesmo pudesse ser lido de forma otimizada. Como trabalho futuro, seria interessante uma generalização da função para leitura de outros arquivos de malha, como os arquivos .OBJ e .3DS.

- Principalmente, a criação das funções de redimensionamento e translação. Apesar do OpenGL oferecer funções prontas para estas tarefas (*glScale() e glTranslate()),* o aprendizado obtido pela implementação destas funções foi o ponto mais importante desta lista.

5. Dificuldades encontradas

Esta lista não trouxe muitas dificuldades comparado à lista anterior, pois desta vez o ambiente já estava totalmente instalado, ou seja, não houve problemas com as bibliotecas do OpenGL e SDL. Para a implementação de leitura, graças a simplicidade do formato .OFF, também não houve problemas. A maior dificuldade encontrada foi para a criação das funções de redimensionamento e translação, já que não havia um background matemático para a criação destas funções. Ao se obter este background, a implementação se tornou relativamente simples.

Referências:

Shreiner, D., Sellers, G., Kessenich, J., Licea-Kane, B. (2013) "OpenGL Programming Guide, Eighth Edition"

http://filext.com/file-extension/3DS

http://groups.csail.mit.edu/graphics/classes/6.837/F03/models/teapot.obj

http://www.martinreddy.net/gfx/3d/OBJ.spec