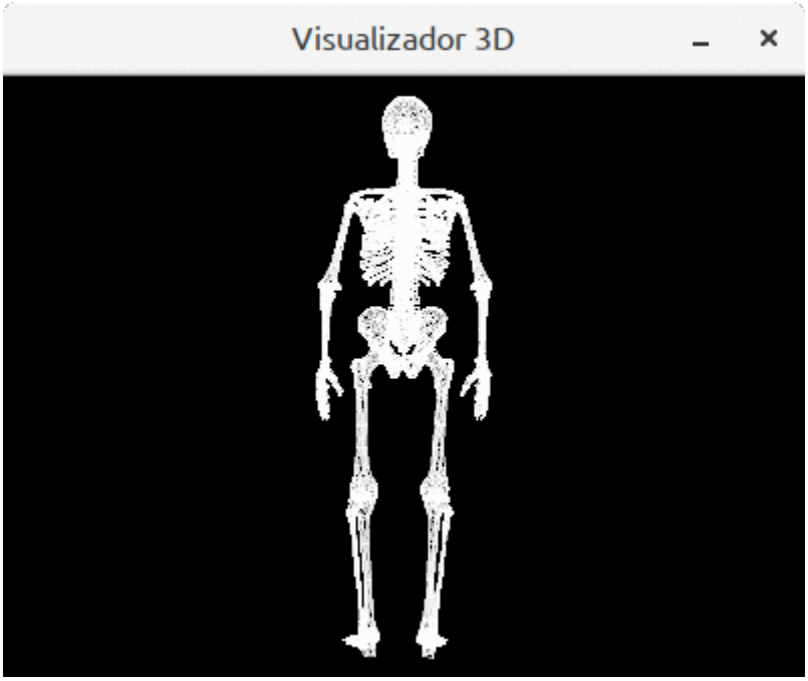
Visualização de Modelos 3D



Este projeto consiste em construir um programa para ler uma descrição de objeto em 3 dimensões de um arquivo em disco e apresentar na tela a visualização desse objeto, considerando uma projeção em perspectiva.

Os requisitos do projeto são:

- Ler a descrição de objeto 3D de um arquivo em disco. O objeto está descrito no formato *Wavefront OBJ*, usado em ferramentas de modelagem/visualização 3D como *Blender*, *3Dstudio* e *Rhino*.
- Calcular a projeção em perspectiva dos vértices e arestas do objeto.
- Mostrar em uma janela gráfica a projeção obtida com representação *wireframe*, usando uma biblioteca gráfica.
- Usar as teclas de setas *left/right* e *up/down* para mudar a posição do observador nos eixos X e Y, respectivamente, recalculando a projeção a cada mudança de posição.
- Usar a tecla “ESC” ou “q” para sair do programa.

A seguir serão apresentados mais detalhes sobre cada um desses requisitos.

Formato Wavefront OBJ

O formato de dados Wavefront OBJ é considerado um “formato universal” para a representação de objetos em 3 dimensões, sendo reconhecido pela maioria dos softwares de modelagem/visualização em 3D.

Nesse tipo de arquivo, os dados do objeto são representados em formato ASCII. Os dados com representação mais frequente são os *vértices* e as *faces*:

- um **vértice** é um ponto no espaço, com coordenadas (x, y, z) em ponto flutuante.
- uma **face** é uma sequência de 3 ou mais vértices que define uma superfície.

Um exemplo simples de objeto 3D no formato OBJ:

cubo.obj

```
# OBJ - Wavefront object file
# Esta linha contém um comentário

# definição dos vértices (coordenadas x y z)
# números podem ser inteiros ou reais
v 16 32 16        # definição do vértice v1
v 16 32 -16       # definição do vértice v2
v 16 0 16         # ...
v 16 0 -16
v -16 32 16
v -16 32 -16
v -16 0 16
v -16 0 -16

# definição das faces
f 1 3 4 2         # face f1: v1 -> v3 -> v4 -> v2 -> v1
f 6 8 7 5
f 2 6 5 1
f 3 7 8 4
f 1 5 7 3
f 4 8 6 2
```

Além dos vértices e faces, o formato OBJ também permite definir linhas, pontos, normais, texturas e outros elementos importantes em um modelo 3D.

Neste projeto, somente precisam ser tratadas as declarações de vértices e faces. As demais declarações (vn, vt, l, etc) podem ser ignoradas.

Mais exemplos de objetos 3D estão disponíveis a seguir:

- Pirâmide (5 vértices, 5 faces)
- Cubo (8 vértices, 6 faces)
- Dodecaedro (20 vértices, 36 faces)
- Bule (822 vértices, 1600 faces)
- Esqueleto (8338 vértices, 16034 faces)
- Coelho (34834 vértices, 69451 faces)

Esses arquivos podem ser facilmente visualizados no Linux através do programa `g3dviewer`, ou neste visualizador online (ou neste).

Projeção em perspectiva

A técnica de projeção 3D consiste em transformar uma representação 3D em 2D, ou seja, com todos os seus pontos em um mesmo plano, possibilitando então sua visualização na tela do computador.

Existem vários tipos de projeção, como a projeção ortográfica e a projeção em perspectiva. Na projeção em perspectiva, a posição do observador é considerada nos cálculos da transformação 3D-2D. Uma boa explicação da projeção em perspectiva pode ser encontrada neste site.

Perspectiva fraca

Os cálculos necessários à projeção podem ser complexos e demorados, mas algumas simplificações podem ser realizadas. Este projeto usa uma simplificação chamada *perspectiva fraca* (*weak perspective*), descrita a seguir:

Considerando que as coordenadas da câmera são $[x_c\ y_c\ z_c]$ e que a câmera está olhando para a origem [0 0 0], a conversão das coordenadas 3D de cada vértice $v = [x_v\ y_v\ z_v]$ em sua projeção 2D $p = [x_p\ y_p]$ no plano z = 0 pode ser calculada desta forma:

- $x_p = x_c + z_c \times ((x_v - x_c) \div (z_v + z_c))$
- $y_p = y_c + z_c \times ((y_v - y_c) \div (z_v + z_c))$

Assim é obtida uma coleção de pontos $[x_p\ y_p]$ que representa os vértices do objeto projetados no plano z=0.

Conversão para coordenadas de tela

Para que os pontos obtidos na etapa anterior possam ser plotados na tela, eles devem primeiro ser ajustados para o intervalo [(0,0) ... (*width*, *height*)], onde *width* e *height* são as dimensões da janela de visualização, em pixels.

Esta etapa transforma o o conjunto de pontos projetados $[x_p\ y_p]$ em um conjunto de pontos de desenho $[x_d\ y_d]$, que podem ser usados para plotar as arestas dos objetos na tela.

Passo 1: calcular mínimos, máximos, centros e diferenças das coordenadas em X e Y

- $x_{min} = \min (x_p)$
- $x_{max} = \max (x_p)$
- $x_{cen} = (x_{max} + x_{min}) / 2$
- $x_{dif} = x_{max} - x_{min}$

- $y_{min} = \min (y_p)$
- $y_{max} = \max (y_p)$
- $y_{cen} = (y_{max} + y_{min}) / 2$
- $y_{dif} = y_{max} - y_{min}$

Passo 2: calcular fator de escala para o desenho na tela

- W: largura da janela de desenho (em pixels)
- H: altura da janela de desenho (em pixels)
- $esc_x = W / x_{dif}$
- $esc_y = H / y_{dif}$
- escala = $\min (esc_x, esc_y)$

Passo 3: centrar pontos da projeção em [0 0]

- ∀ (x y)
  - $x'_p = x_p - x_{cen}$
  - $y'_p = y_p - y_{cen}$

Passo 4: ajustar escala dos pontos da projeção para a tela

- ∀ (x y)
  - $x“_p = x'_p$ * escala
  - $y”_p = y'_p$ * escala

Passo 5: ajustar pontos do desenho em relação ao centro da tela

- ∀ (x y)
  - $x_d = x“_p + W / 2$
  - $y_d = y”_p + H / 2$

Os passos 3 a 5 podem ser condensados em um único passo:

- ∀ (x y)
  - $x_d = ((x_p - x_{cen}) * escala) + W / 2$
  - $y_d = ((y_p - y_{cen}) * escala) + H / 2$

Com isso é obtido um conjunto de pontos no intervalo [(0, 0) .. (*width*, *height*)] que pode ser usado para plotar na janela gráfica as arestas que definem o objeto 3D.

Atividade

Para apresentar a visualização de um objeto 3D, o programa deve:

1. Ler o conjunto de vértices e faces do arquivo de entrada
   1. Modelos grandes podem conter milhões de vértices, portanto o uso de alocação dinâmica de memória é obrigatório.
   2. Para tratar os dados do arquivo OBJ, sugere-se usar a função `strtok`.
2. Para uma dada posição da câmera, calcular as projeções 2D dos vértices 3D.
3. Converter os pontos projetados para coordenadas de tela; sugere-se usar uma janela de 800×600 pixels.
4. Desenhar as faces na tela, produzindo uma representação *wireframe* do objeto. Para desenhar uma face, desenhe separadamente as arestas que a compõem. Por exemplo, para desenhar a face (1 2 3 4), desenhe as arestas (1 2), (2 3), (3 4) e (4 1).
5. Ler as teclas de setas (← → ↑ ↓), ajustar as coordenadas da câmera, recalcular a projeção e desenhá-la novamente.
6. Tecla ESC (ou fechar a janela) para sair do programa.

Formas de chamada do executável (**ambas** devem ser implementadas):

```
# usando argc/argv
wireframe arquivo.obj

# usando stdin
wireframe < arquivo.obj
# ou
cat arquivo.obj | wireframe
```

A implementação deve atender os seguintes requisitos:

- Usar as funções da biblioteca gráfica indicada pelo professor
- Usar `struct` para representar os elementos do modelo (vértices, faces, arestas)
- Alocar memória para os elementos do modelo de forma dinâmica (`malloc`/`free`)
- Funcionar para **todos** os exemplos disponíveis nesta página
- Estrutura **sugerida** de arquivos do código-fonte:
  - `wireframe.c`: arquivo principal
  - `datatypes.h`: tipos de dados usados no programa
  - `objread.c/h`: funções de leitura do arquivo OBJ
  - `perspect.c/h`: funções de cálculo de perspectiva

    - `graphics.c/h`: funções de apresentação gráfica
    - ... (se necessário)
- Usar `Makefile`
    - cláusulas `all` (default), `clean` e `purge`
    - usar flag de compilação `-Wall`
    - separar as fases de compilação e de ligação
    - **usar regras implícitas**

Bônus (+ 20/100 pontos cada):

- Usar o mouse para girar o objeto na tela
- Usar perspectiva completa ao invés da perspectiva fraca

O que deve ser entregue ao professor:

- arquivos `.c` e `.h`
- arquivo `Makefile`
- **não enviar** os arquivos OBJ de teste

Estruturas de dados sugeridas

As estruturas mais adequadas para este projeto são vetores de elementos (de pontos 3D, de pontos 2D, de retas, etc). Como o número de elementos não é fixo, esses vetores devem ser alocados dinamicamente. Entretanto, o número de elementos só é conhecido ao final da leitura do arquivo OBJ, por isso não é possível alocar cada vetor com seu tamanho final desde o início do programa.

A estratégia recomendada para esse problema é fazer uma alocação inicial e aumentá-la conforme a necessidade (usando a chamada `realloc`). Como a realocação de memória é uma operação demorada, sugere-se realocar os vetores em "blocos" de centenas ou milhares de elementos de cada vez.

A biblioteca SDL

Existe uma infinidade de bibliotecas gráficas; entre as mais populares está a biblioteca SDL (*Simple DirectMedia Layer*). Além de gráficos, essa biblioteca permite a produção de sons e o gerenciamento de dispositivos de entrada/saída (*mouse*, teclado, *joystick*) em várias plataformas.

Recomenda-se usar SDL 2, a versão mais recente da biblioteca. Tome cuidado, pois existem muitos exemplos e tutoriais na Internet que usam versões mais antigas da SDL, incompatíveis com a versão atual.

Documentação e tutoriais sobre SDL:

- https://www.libsdl.org
- https://wiki.libsdl.org/SDL_RenderDrawLine (exemplo de desenho de retas)
- http://lazyfoo.net/SDL_tutorials/index.php

Instalação da biblioteca SDL 2 em Linux (Ubuntu, Mint, Debian):

```
sudo apt-get install libsdl2-dev
```

Arquivos de cabeçalho necessários:

```
#include <SDL2/SDL.h>
```

Flags de compilação:

```
LDLIBS = -lSDL2
```

A biblioteca Allegro

A biblioteca gráfica Allegro permite a manipulação de gráficos simples e áudio, sendo bem adaptada para a construção de jogos 2D. É uma biblioteca mais simples e limitada que SDL, mas boa para projetos menores.

Algumas de suas características:

- multiplataforma: Linux, Windows, MacOS, Android, iOS
- pode ser usada em C e outras linguagens
- usa aceleração gráfica (através de OpenGL ou DirectX)
- manipulação de áudio e vídeo
- leitura de mouse, teclado e joystick

Instalação da biblioteca Allegro 5 em Linux (Ubuntu, Mint, Debian):

```
sudo apt-get install liballegro5-dev
```

Arquivos de cabeçalho necessários (mínimo, depende dos módulos usados):

```
#include <allegro5/allegro.h>
```

Flags de compilação (idem):

```
LDLIBS = -lallegro
```