Computação Gráfica - Trabalho Prático 2

Anna Luiza Pereira Rosa, João Vitor de Faria, Wesley da Silva Ribeiro

Instituto de Matemática e Computação – Universidade Federal de Itajubá (UNIFEI)
Caixa Postal 50 – 37500 903 – Itajubá – MG – Brasil

{anna.luiza, joaovitorfaria,
wesleysilvaribeiro}@unifei.edu.br

1. Introdução

Grafos são estruturas de dados formadas por um conjunto de vértices e um de arestas; é uma estrutura utilizada basicamente para representar um modelo em que existem relações entre objetos de um grupo.

Dijkstra é um algoritmo que calcula o caminho mínimo entre os vértices de um grafo com arestas de peso não negativas. Um exemplo prático de problema que pode ser resolvido por esse algoritmo é: uma pessoa precisa se deslocar de uma cidade para outra e para isso, ela pode escolher entre vários caminhos e deseja fazer o percurso na menor distância possível.

O objetivo deste trabalho é ilustrar como essa decisão seria tomada, baseada na utilização do algoritmo de Dijkstra, através de uma animação em 2D. A implementação foi feita com a linguagem de programação C, utilizando a API OpenGL.

2. Detalhes da implementação

2.1 Algoritmo de Dijkstra

Teorema: Para qualquer grafo direcionado com arcos de comprimento não negativos, o algoritmo calcula corretamente as distâncias dos caminhos mais curtos a partir de um vértice fonte.

No início, o conjunto S contém somente o nó origem. A cada passo, selecionamos no conjunto de nós sobrando, o que está mais perto da origem. Depois atualizamos, para cada nó que está sobrando, a sua distância em relação à origem. Se passando pelo novo nó acrescentado, a distância ficar menor, é essa nova distância que será memorizada.

Escolhido um nó como origem da busca, este algoritmo calcula, então, o custo mínimo deste nó para todos os demais nós do grafo. O procedimento é iterativo, determinando, na iteração 1, o nó mais próximo do nó O, na segunda iteração, o segundo nó mais próximo do nó O, e assim sucessivamente, até que em alguma iteração todos os n sós sejam atingidos.

Este algoritmo parte de uma estimativa inicial para o custo mínimo e vai sucessivamente ajustando essa estimativa. Ele considera que um vértice estará fechado quando já tiver sido obtido um caminho de custo mínimo do vértice tomado como raiz da busca até ele; caso contrário ele é dito aberto.

Funcionamento: Seja G(V,A) um grafo orientado e s um vértice de G

- Atribui valor zero à estimativa do custo mínimo do vértice s (a raiz da busca) e infinito às demais estimativas
- Atribui um valor qualquer aos precedentes (o precedente de um vértice t é o vértice que precede t no caminho de custo mínimo de s para t)
- Enquanto houver vértices aberto:
    - Seja k um vértice ainda aberto cuja estimativa seja a menor dentre todos os vértices abertos
    - Feche o vértice k
    - Para todo vértice j ainda aberto que seja sucessor de k:

- Soma a estimativa do vértice k com o custo do arco que une k a j
- Caso esta soma seja melhor que a estimativa anterior para o vértice j, substitui e marca k como precedente de j

2.2 Modelo 2D

Foram utilizados quadrados, triângulos e retas para construir a animação, de forma a construir a imagem das casas de forma clara.

A animação utiliza de casas para representar as cidades (vértices) e retas para representar os caminhos (arestas). Dessa forma, as casas e os caminhos são adicionadas com um clique, assim como os nós de origem e destino.

Durante a execução, será possível ver o caminho que o algoritmo percorre a fim de retornar o menor caminho possível entre os nós através da alteração de cores.

3. Funcionalidades

3.1 Função desenhaVertice

Recebe como parâmetros as coordenadas X e Y para inserção de um vértice na janela. Em seguida, utiliza-se esses valores para fazer a projeção de uma casa com uso da função glVertex2D, combinado com as funções para desenho de quadrados (GL_QUADS), linhas (GL_LINE_SMOOTH) e triângulos (GL_TRIANGLES). Cada parte da ilustração é feita individualmente e com base em valores pré-definidos.

3.2 Função desenhaAresta

Esta função é responsável por desenhar o caminho entre dois vértices do grafo, representado pelas casas, definindo a cor de cada caminho, a depender o estado durante a execução do algoritmo: para um caminho não visitado, a cor é cinza escuro; para um caminho visitado, a cor é cinza claro e para o caminho mais curto (saída do algoritmo) a cor é verde.

3.3 Função mouse

O objetivo desta função é capturar o evento de clique do mouse, uma vez que o botão esquerdo é responsável pela definição dos vértices e as arestas e o botão direito pela definição dos vértices de origem e destino.

3.4 Função keyboard

O objetivo desta função é capturar o evento de clique da tecla enter, uma vez que essa é a tecla responsável por disparar a execução do algoritmo após definidos os vértices, as arestas e os vértices de origem e destino.

3.5 Função getVerticesMaisProximo

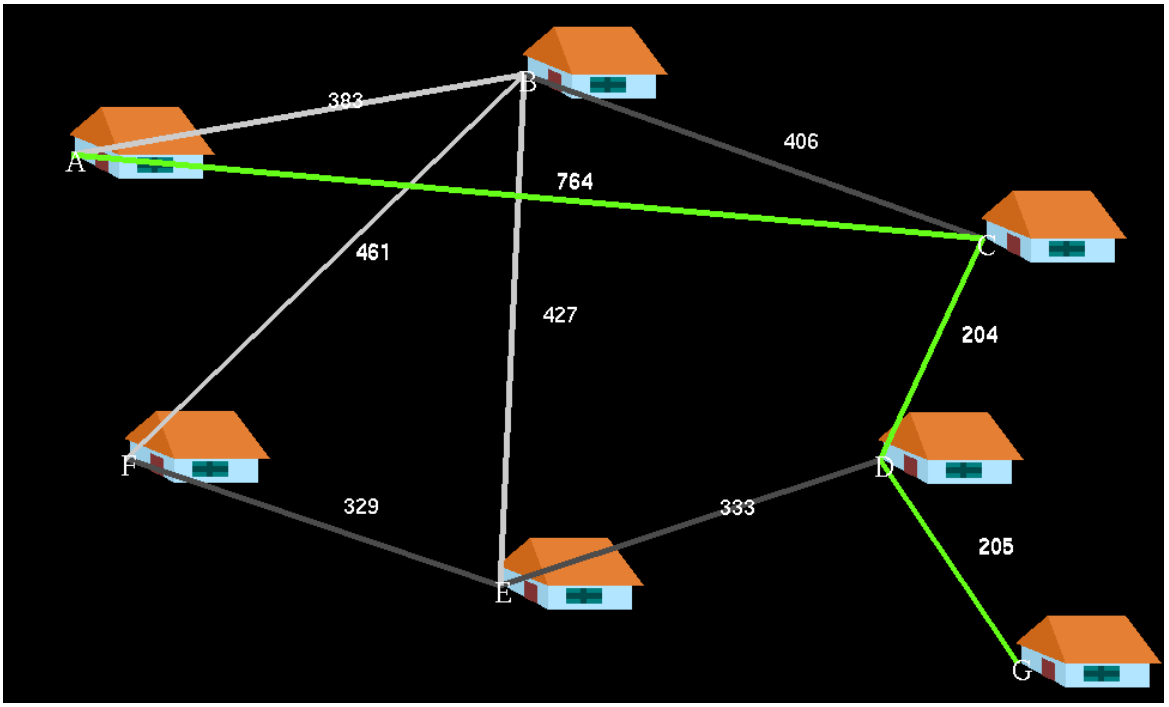
O objetivo desta função é viabilizar parte da execução do Algoritmo de Dijkstra. Através dela é possível percorrer todos os vértices não visitados e descobrir a distância mínima entre um dado vértice e outro vértice vizinho.

3.6 Função dijkstra

Função que implementa a execução do algoritmo e que chama as outras funções e, em conjunto, são responsáveis pelo desenvolvimento da animação.

4. Resultados

Exemplo da animação depois de executada, em que o vértice A é a origem e o vértice G é o destino.



5. Conclusão

A maior dificuldade deste trabalho foi combinar a implementação de elementos 3D com o funcionamento do Dijkstra, para atingir o objetivo de ilustrar como o algoritmo funciona. Transformar animações 2D em 3D não é uma tarefa trivial, e não existem muitos auxílios quanto à descrição e utilização de alguns métodos da API.

6. Referências

Coordinate Systems. Learn OpenGL. Disponível em: <https://learnopengl.com/Getting-started/Coordinate-Systems>. Acesso em: 09/10/2020.

OpenGL Documentation. Khronos. Disponível em: <https://www.khronos.org/registry/OpenGLRefpages/gl2.1/xhtml/>. Acesso em: 07/12/2020

OpenGL 3D. Disponível em: <https://www.inf.pucrs.br/~pinho/CG/Aulas/OpenGL/OpenGL3D.html>. Acesso em 08/12/2020.

MOTA, Kleber. OpenGL Desenhando Polígonos. Disponível em: <https://klebermota.eti.br/2015/05/08/opengl-desenhando-poligonos/>. Acesso em: 10/12/2020.

WILHELM, Volmir Eugênio. Problema do Caminho Mínimo. Disponível em: <https://docs.ufpr.br/~volmir/PO_II_10_caminho_minimo.pdf/>. Acesso em: 11/12/2020.

7. Anexo