UFMG - 2024/1 - DCC024

Geometria Computacional: Relatório do Trabalho Prático 1 de Algoritmos 2 (DCC207)

Professor: Renato Vimieiro

Nomes	**Matrículas**
Luiz Fernando Gonçalves Rocha	2022043582
Thales Augusto Rocha Fernandes	2022043825
Júlio Assis Souza Amorim	2022043590

1. Recepção da entrada

1.1. Leitura e carregamento

A execução do código se inicia com o comando ‘python main.py inputs/input**x**.txt’, sendo ‘main.py’ o arquivo Python responsável pela gerência total do programa, e ‘inputs/input**x**.txt’ o argumento de entrada (‘**x**’ representando um número que corresponde a uma das entradas de exemplo contidas no arquivo ‘inputs’), intercambiável com qualquer outro arquivo de texto que siga os formatos de entrada reconhecidos pelo algoritmo.

O código então acessa os argumentos para obter o nome do arquivo e o insere em uma função ‘parse_entrada’. Esta o abre para a leitura, atribui uma variável de tamanho ao conteúdo da primeira linha e segue lendo as demais, convertendo frações em coordenadas de ponto flutuante e adicionando as duplas contidas em cada linha como vértices em um polígono. O objeto do polígono é mantido por uma classe ‘Poligono’, que lista seus vértices e permite operações de adição e remoção destes.

1.2. Armazenamento dos estados

Buscando armazenar os dados necessários para apresentar a execução passo-a-passo, criamos duas classes: ‘Estado’ e ‘SequenciaEstados’. A primeira guarda um polígono, dois grafos, um correspondendo ao primal e outro ao dual (como será explorado mais à frente), um dicionário mapeando as cores no primal e uma lista de câmeras. Como não mais que um objeto correspondendo a cada categoria precisa ser criado no código todo, os objetos usados para inicializar a classe são todos declarados globalmente.

Nesse sentido, o objeto polígono notado na seção anterior (1.1.) é o mesmo usado para inicializar a classe ‘Estado’. Esta, porém, não é inicializada diretamente pela função ‘parse_entrada’, mas sim indiretamente, por meio de uma inicialização da classe ‘SequenciaEstados’, que guarda uma lista de estados. Tal lista será usada para guardar cada passo relevante dentro da execução do algoritmo, servindo então como o objeto da animação ao final do programa.

2. Triangulação

2.1. Descoberta das orelhas

Após o armazenamento do estado inicial do nosso algoritmo dar fim à função ‘parse_entrada’, o código segue para a função ‘triangular’, adicionando cada vértice e aresta do polígono ao grafo primal globalmente declarado e atualizando o estado do algoritmo na sequência. Em seguida, para cada vértice do polígono, a função checa ele e os dois seguintes, chamando a função ‘eh_orelha’ para averiguar a presença de orelhas na figura.

Uma orelha de um grafo não-direcionado é um caminho onde seus dois extremos podem coincidir, mas em que, alternativamente, nenhuma repetição de arestas ou vértices é permitida, então cada vértice interno do caminho tem grau dois. A função ‘eh_orelha’ usa duas outras funções, denominadas ‘orientacao’ e ‘nenhum_ponto_dentro’ (esta última se utilizando de 'ponto_no_triangulo'), para, a partir de um vértice de entrada, averiguar se ele e os dois vértices seguintes no polígono formam uma tal orelha.

2.2. Atualização dos vizinhos

Para cada orelha identificada no polígono, uma aresta é adicionada entre seu vértice inicial e seu vértice final (saltando o vértice entre eles) no grafo primal pela função ‘triangular’. O ponto saltado é então removido do polígono global e, a cada remoção, o estado atual é adicionado na sequência, para que o passo seja exibido na animação. Ao final das iterações da função, o polígono é apagado, posta a sua atual obsolescência: já temos no grafo primal a triangulação desejada.

3. Finalização

3.1. Faces e grafo dual

Encerrada a execução de ‘triangular’, o código segue para a função ‘get_dual’, que por sua vez chama a função ‘get_faces’, que por sua vez usa a biblioteca subprocess para acessar e executar o arquivo ‘main.cpp’ em C++. A função então obtém as faces do grafo primal e, por meio delas, cria um grafo dual, com um vértice correspondendo a cada face e uma aresta entre vértices correspondendo a cada aresta compartilhada entre as respectivas faces. A sequência só é atualizada uma vez em virtude do papel auxiliar do grafo, mostrando sua sobreposição já pronta na figura.

3.2. Coloração

Com o grafo dual montado, o código segue chamando a função ‘cor’, que preenche o dicionário global de mapeamento de cores no grafo primal usando o grafo dual em uma Busca em Profundidade, instanciada pela função interna ‘dfs’. A Busca em Profundidade resolve o problema da coloração ao explorar os vértices e as arestas do grafo sistematicamente, detectando componentes conexos e ciclos. Essa exploração encontra uma ordem para aplicar cores minimizando conflitos.

Como o grafo é planar e triangulado, apenas três cores são necessárias para que vértices vizinhos não tenham a mesma cor. O estado da execução é adicionado à sequência a cada vértice individual que é colorido. Ao final do processo, o grafo dual é apagado em virtude de sua obsolescência e assim a representação gráfica fica mais limpa.

3.3. Escolha das câmeras

Por fim, ainda na função ‘cor’, o programa identifica a cor com menor ocorrência dentre os vértices e atribui a ela o posicionamento das câmeras, que é guardado na lista global. A sequência é então atualizada, chegando ao seu estado final. A escolha de qualquer cor seria suficiente para a resolução do problema em si, posto que o número de triângulos é o mínimo e uma câmera em qualquer vértice do triângulo já o cobriria, mas a opção pela cor mais rara minimiza o número de câmeras necessárias para cobrir todos os triângulos da figura.

3.4. Animação

Finalizando de vez a execução, nosso programa chama a função ‘cria_animacao’, que se apoia na classe ‘Player’, implementada no arquivo animate.py,

tomando por entrada a sequência de estados gerada até então. A função itera pelos estados registrados e os plota um a um em uma animação, permitindo iteração manual. No princípio da animação, o polígono é representado como uma espécie de envoltória que cobre o grafo primal. Conforme o grafo é atualizado com as arestas correspondentes à triangulação, os vértices ignorados são removidos do polígono um a um, até que este deixe de existir. Daí, como já indicado anteriormente, o mapeamento das faces no grafo dual é sobreposto à imagem e então observamos o processo de coloração dos vértices um a um, cada face em presente exploração sendo visualmente demarcada por um preenchimento. Ao final deste, o dual é retirado da figura e os vértices correspondendo às câmeras são marcados com um X, todos eles coincidindo em cor.