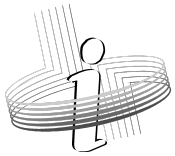


UNIVERSIDADE FEDERAL DE VIÇOSA
DEPARTAMENTO DE INFORMÁTICA
Inf 390 – Computação gráfica
Prof. Salles Magalhães

Trabalho 1 (parte 2)
O trabalho poderá ser feito em dupla

Regras gerais de trabalhos de INF390:
https://docs.google.com/document/d/1yvvfR6TrK3B9wBg7rHQNwHIJZ34l01rBtq0P7wFSrRA/edit (atenção: **leia TUDO**, mesmo se já tiver cursado alguma disciplina comigo)

Conforme visto em sala, uma importante operação em geometria computacional é o cálculo da orientação de um ponto em relação a outros dois (isso no caso em 2D). Com isso, pode-se ver se o terceiro ponto é colinear em relação aos dois primeiros, se ele faz uma volta à direita ou à esquerda. Esse tipo de predicado pode ser utilizado em inúmeras aplicações geométricas (por exemplo, para ver se dois segmentos se interceptam).

Neste trabalho, você deverá implementar um predicado de orientação 2D. A seguir, tal predicado deverá ser utilizado para verificar se dois segmentos de reta se interceptam. Você deverá, então, criar um programa que, dada uma lista de segmentos de reta, determina quantos pares desses segmentos se interceptam.

Isso tem aplicação em diversas áreas. Por exemplo, detecção de colisões em jogos, cálculo de interseção de mapas, em plantas de engenharia, etc.

Obs:
- Utilize sempre cálculos com números do tipo double.
- Considere que uma interseção existe apenas se ela ocorrer no interior dos dois segmentos em questão. Ou seja, casos especiais como interseções que ocorrem nas extremidades ou interseções de segmentos colineares são considerados como “não interseção”.

Acelerando os cálculos

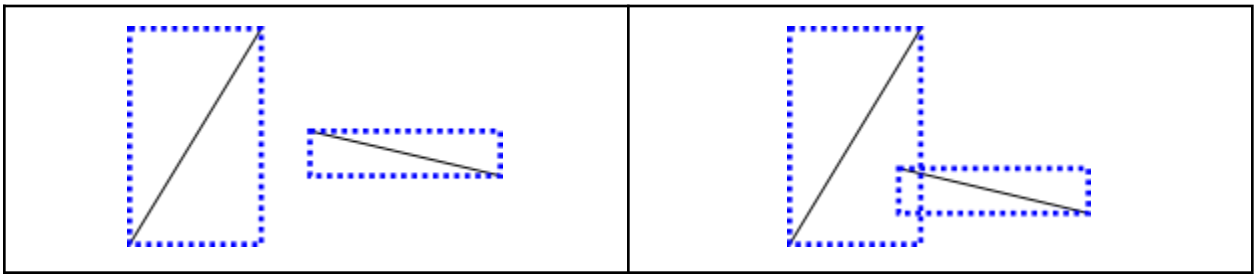
Como fazer os cálculos acima de forma trivial pode ser uma operação lenta (visto que o número de pares de segmentos a serem avaliados é quadrático em termos do tamanho da entrada), há técnicas que podem acelerar tais cálculos.

Seu programa deverá suportar o uso de duas técnicas: bounding-boxes e índice geométrico.

Bounding-boxes

Uma ideia simples para acelerar os cálculos consiste em se calcular o retângulo envolvente (bounding-box) a cada segmento (neste trabalho, você deverá criá-los durante a leitura da entrada e simplesmente utilizá-los no restante do programa). A bounding-box de um segmento s é o menor retângulo que contém s (os dois pontos de s são vértices de tal retângulo).

Ao testar dois segmentos para verificar se eles se interceptam, um “teste de bounding-box” pode ser aplicado antes de se utilizar os predicados de orientação: se as bounding-boxes desses dois segmentos não se interceptam (figura abaixo, da esquerda), então eles certamente não se interceptam (com isso, evita-se fazer os cálculos com os predicados de orientação, que muitas vezes são mais caros computacionalmente). Por outro lado, se as bounding-boxes se interceptam (figura abaixo, da direita) então os segmentos podem se interceptar (nesse caso, os testes de interseção devem ser realizados para confirmar isso).



Note que o uso dos testes de bounding-box não reduz a complexidade do algoritmo (ainda assim faremos um número quadrático de testes de bounding-box), mas pode evitar o uso de testes mais caros computacionalmente em muitas situações (já que verificar se dois retângulos se interceptam é muito mais fácil do que verificar dois segmentos).

No pior caso (escreva no seu README em qual situação isso poderia acontecer), as bounding-boxes poderiam não ajudar em nada! (mas normalmente ajudam muito).

Na verdade, mesmo no melhor caso pode ser que elas não ajudem muito, já que o cálculo de interseção 2D é relativamente simples (em cálculos mais complexos, como interseção de triângulos em 3D elas são mais úteis).

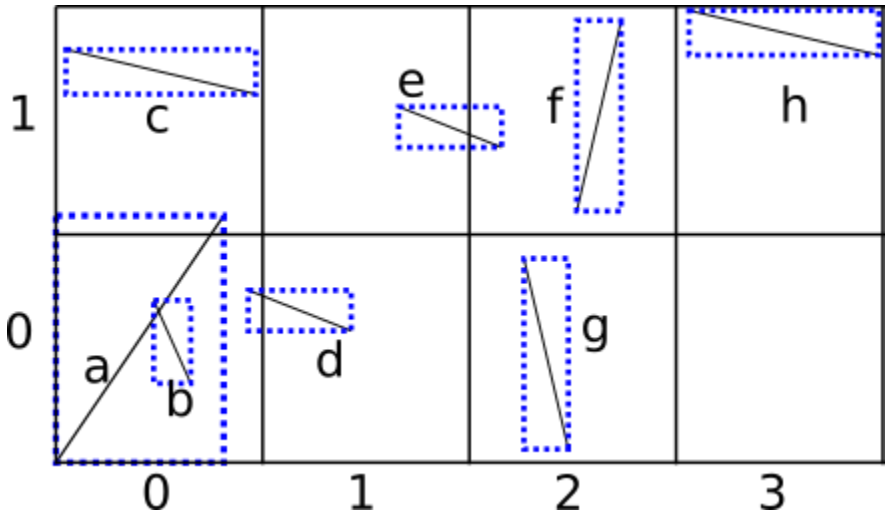
Índice geométrico

Há alguns algoritmos (mais complexos de serem implementados) capazes de detectar as interseções de forma eficiente. Um exemplo de tal algoritmo é o de “sweep-line”. Porém, uma técnica simples (e normalmente muito eficiente -- na prática talvez até mais eficiente do que algoritmos complexos) consiste em se utilizar um índice geométrico para acelerar os cálculos.

Há vários tipos de índices: R-trees, quadtrees, octrees (3D), *uniform grids*, etc. Neste trabalho utilizaremos o mais simples deles: a *uniform grid* (ou grade regular). Apesar da simplicidade,

esse tipo de índice normalmente é muito eficiente e possui várias vantagens sobre outros (por exemplo, facilidade de se paralelizar os cálculos, de ser criada muito eficiente, etc).

A ideia consiste em se criar uma grade de retângulos e sobrepô-la aos segmentos de entrada. Os dois extremos da grade são posicionados de modo que ela englobe todos os segmentos (ou seja, a grade é a bounding-box de todos os segmentos da entrada). A seguir, os segmentos são inseridos nas células nas quais suas bounding-boxes se interceptam.



Considere o exemplo de uniform grid com 2 linhas e 4 colunas acima. A bounding-box do segmento *a* intercepta as células (0,0) e (1,0) e, portanto, *a* será inserida nas duas células (estamos usando a ordem (y,x) para definir coordenadas). A bounding-box de *g*, por outro lado, só intercepta a célula (0,2) e, assim, *g* estará apenas nessa célula.

No exemplo acima há 8 segmentos e, portanto, o número de pares que deveriam ser testados por interseção seria 28. Porém, com o índice esse número pode ser bastante reduzido: note que se dois segmentos se interceptam tal interseção ocorrerá dentro de uma célula. Assim, a interseção de f e g não precisa ser testada (como estão em células diferentes, não tem como eles se interceptarem).

Com isso, as interseções podem ser obtidas de forma mais simples: basta percorrer as células da grade e, em cada célula C, testar os pares de segmentos que estão em C. No exemplo acima, na célula (0,0) testaríamos os pares (a,b); (a,d) e (b,d). Na célula (1,0) estaríamos (a,c) e na célula (1,2) testaríamos (e,f). Com isso, o número de testes seria reduzido de 28 para 5 (os 5 testes devem, então, ser feitos utilizando a aceleração por bounding-box e, por fim, os testes de orientação).

Em casos muito extremos a eficiência poderia ser tão ruim quanto a do algoritmo de força bruta, mas em geral essa técnica funciona muito bem.

Obs:
- Antes de implementar faça alguns exemplos em papel e pense bem em suas escolhas de implementação. Lembre-se que em geometria computacional sempre há alguns casos especiais que complicam os algoritmos um pouco. Neste problema tais casos são relativamente simples de tratar, mas existem….

Entrada

A entrada do seu programa começará com uma string, que poderá ser “bruta”, “box” ou “grid”. Se a opção for “bruta” seu algoritmo deverá utilizar o método de força bruta tradicional para testar os pares (ou seja, o método quadrático).

Se for “box”, um teste de bounding-box deverá ser feito para acelerar cada avaliação de interseção (conforme descrito acima).

Por fim, se for “grid”, você deverá criar uma uniform grid e utilizá-la para acelerar os cálculos. O número de linhas e colunas (sempre positivos) da grade serão fornecidos imediatamente após a string grid (tudo separado por um espaço em branco).

A seguir, haverá um número N indicando o número de segmentos fornecidos na entrada. Por fim, teremos N linhas cada uma descrevendo um segmento.

Cada segmento é descrito por 4 números reais y1 x1 y2 x2 indicando, respectivamente, que o segmento em questão liga os pontos (y1,x1) e (y2,x2).

Saída

Seu programa deverá gravar na saída padrão (stdout) o número de interseções detectadas.

Adicionalmente, imprima na saída stderr as seguintes estatísticas sobre os cálculos (na ordem descrita abaixo, sendo um número por linha):

- Números de pares de segmentos na entrada.
- Número de testes de bounding-box feitos (deverá ser 0 no caso do algoritmo de força-bruta).
- Número de testes de interseção feitos utilizando orientação.
- Número de interseções detectadas (ou seja, o mesmo número impresso na saída padrão).
- Tempo para ler os dados da entrada e criar os bounding-boxes (em segundos, 3 casas de precisão).
- Tempo para criar a uniform grid (em segundos, 3 casas de precisão). (esse tempo será 0.000 nos modos box e bruta)

- Tempo para contar as interseções (em segundos, 3 casas de precisão). No modo grid, esse tempo deverá incluir percorrer o índice para ver quais pares deverão ser processados.
- Tempo total (deverá ser a soma dos 2 tempos anteriores, ou seja, o tempo total **sem considerar a leitura dos dados**).

Exemplo de entrada e saída

ENTRADA	STDOUT	STDERR
box 7 1 1 5.2 5.2 1 3 3 1 15.2 35.1 19 36 1 5 1 10 0 6 2 6 0 8 1.5 7 0 9 1.5 9.2	4	21 21 5 4 Tempo (s) para ler dados: 0.000 Tempo (s) para criar grid: 0.000 Tempo (s) para contar intersecoes: 0.000 Tempo (s) total: 0.000

Nesse exemplo, há 21 pares de segmentos na entrada. Como um índice não foi utilizado (e o modo box foi selecionado), 21 testes de bounding-box serão realizados (se tivéssemos usado o modo grid provavelmente esse número seria menor). Dos 21 testes, 5 pares de bounding-box se interceptam. Das 5 bounding-box que se interceptam, 4 pares de segmentos realmente se interceptam.

Obs: a saída STDERR não será avaliada utilizando testes automáticos, pois pode ser que haja pequenas diferenças entre essa saída e a obtida pelo código do professor. (devido a escolhas no processo de implementação, erros de arredondamento, etc)

Experimentos

Você deverá realizar dois tipos de experimentos. Os resultados (informações sobre a entrada, o tempo, respostas para as perguntas, comentários, conclusões, etc) devem estar no seu relatório. No final deste documento há um pequeno programa (bem simples) que pode ser utilizado para gerar entradas.

Comparação dos algoritmos:

Avalie os 3 métodos de interseção implementados (no caso da grid, use sempre uma grade de 100x100 células). Nesses experimentos, recomendo que os pontos iniciais dos segmentos sejam sorteados entre 0 e 50000 e que o lado de suas bounding-boxes seja no máximo em torno de 500. (ou seja, teremos segmentos relativamente pequenos perto do tamanho da região onde são sorteados)

Teste vários tamanhos de entradas (em termos do número de segmentos) e avalie os 3 métodos. Qual o melhor? Por que? O tempo de criar a grid compensa a economia de tempo em detectar as interseções? Alguma outra observação.

Crie um caso de teste onde a grid não apresente um bom desempenho em relação aos outros métodos (a resolução da grid deverá ser 100x100 e o tamanho da entrada deverá ser similar a algum outro tamanho testado (onde a uniform grid tinha apresentado uma boa performance)). Explique o motivo.

Avaliação da uniform grid com diferentes resoluções:
No segundo conjunto de experimentos, avalie o desempenho da uniform grid com diferentes resoluções (por simplicidade, sempre use a mesma largura/altura da grade nesses testes e sempre casos de teste com os mesmos segmentos e mesmo tamanho). Tente resoluções bem pequenas e outras bem grandes.

O que você observa no tempo total de execução (sem considerar a leitura de dados)? O que você observa nos outros tempos?

Arquivo README

Seu trabalho deverá incluir um arquivo README.

Tal arquivo conterá (nessa ordem):

- Nome/matricula
- Informacoes sobre fontes de consulta utilizadas no trabalho
- Pequeno relatório.

No relatório, primeiro responda as seguintes perguntas (escreva as perguntas e respostas na mesma ordem apresentada aqui):

1) Em qual situação a avaliação de uma entrada não seria acelerada se a opção “box” fosse utilizada? Descreva como tal entrada seria. (exemplo: se fosse o algoritmo quicksort original, a resposta deveria ser aproximadamente “Como o pivô é sempre o primeiro elemento, o pior caso do quicksort ocorre quando a entrada já está ordenada (ou quase ordenada), tanto em ordem crescente quanto decrescente. Isso aconteceria porque as divisões seriam completamente desbalanceadas (uma das chamadas recursiva seria chamada com apenas um dos valores e a outra com o resto)”).
2) Mesma pergunta anterior, mas agora para o modo “grid”.

A seguir, inclua o relatório de seus experimentos.

Entrega

O trabalho deverá ser entregue pelo Submitty. O código deverá estar em um arquivo main.cpp (ele será compilado utilizando o comando “g++ main.cpp”)

Avaliacao

Seu programa deverá compilar/funcionar no Linux (sugere-se que você o desenvolva no Linux -- você deverá pelo menos testá-lo nesse Sistema Operacional antes de entregá-lo).

Duvidas

Dúvidas sobre este trabalho deverão ser postadas no sistema Piazza. Se esforce para implementá-lo e não hesite em postar suas dúvidas!

Avaliacao do codigo

O código do seu trabalho também será avaliado. Tente mantê-lo bem organizado, com comentários, etc.

Adicione comentários explicando os passos de cada algoritmo implementado.

Pequeno programa para gerar entradas

Segue o código de um programa simples para gerar entradas aleatórias para experimentos. (sempre usamos a mesma semente do rand() e, portanto, será sempre gerado o mesmo arquivo caso os parâmetros sejam os mesmos)

```
#include <vector>
#include <iostream>
#include <cassert>
#include <algorithm>
#include <cmath>
#include <array>
#include <cstdlib>
using namespace std;

/*
Ideia: para cada um dos N segmentos a serem sorteados:
-- sorteia o y1,x1 aproximadamente entre 0 e mxDim
-- sorteia y2 = um valor que estará na posicao y1 + numeroAleatorioEntre(-mxLen e mxLen)
-- o mesmo para x2
*/
```

```
int main(int argc, char **argv) {
	if(argc!=4) {
		cerr << "Erro: use ./gerar N maxDim maxLen" << endl;
		cerr << "Onde N é o número de segmentos" << endl;
		cerr << "maxDim é a largura máxima da bounding-box (quadrada) da região
onde os vértices iniciais dos segmentos serão sorteados" << endl;
		cerr << "mxLen é o lado máximo do bounding-box dos segmentos sorteados" <<
endl;
		exit(1);
	}
	int n = atoi(argv[1]);
	int mxDim = atoi(argv[2]);
	int mxLen = atoi(argv[3]);

	cout << "bruta\n";
	cout << n << endl;
	for(int i=0;i<n;i++) {
		double y1 = (rand()%(10*mxDim))/10.0;
		double x1 = (rand()%(10*mxDim))/10.0;

		//y2,x2 poderao ser sorteados tanto maiores quanto menores do que y1,x1...
		double y2 = y1 + (rand()%2==0?-0.1:0.1)*( (rand()%(10*mxLen)) );
		double x2 = x1 + (rand()%2==0?-0.1:0.1)*( (rand()%(10*mxLen)) );
		cout << y1 << " " << x1 << " " << y2 << " " << x2 << "\n";
	}
}
```