Acelerando algoritmos exatos de SIG com GPUs

Marcelo de M. Menezes[1], Salles V. G. de Magalhães[1], Rodrigo E. de O. B. Chichorro[1], Matheus A. de Oliveira[1], Marcus V. A. Andrade[1]

[1]Departamento de Informática, Universidade Federal de Viçosa (UFV)
Campus da UFV, Viçosa, MG, Brazil

{marcelo.menezes, salles, rodrigo.chichorro, matheus.a.aguilar, marcus}@ufv.br

Abstract. *This paper presents a technique for accelerating the evaluation of exact geometric predicates using CPU and GPU parallel computing. These predicates are important, for example, in GIS applications. The algorithm implemented as a case study in this paper presented a speedup of up to* 289 *times (if only the time spent evaluating the geometric predicates was considered) and up to* 40 *times considering the total running-time of the algorithm (when compared against the sequential implementation).*

Resumo. *Este artigo apresenta uma técnica que combina o uso de computação paralela em CPU e GPU para acelerar a avaliação de predicados geométricos de forma exata. Tais predicados são primitivas importantes, por exemplo, em algoritmos de SIG. No estudo de caso apresentado foi possível obter um ganho de até* 289 *vezes no tempo de avaliação de predicados e* 40 *vezes no tempo total do algoritmo (em relação à versão sequencial).*

1. Introdução

Sistemas de Informação Geográfica, ou SIGs, normalmente dependem de métodos que realizam tanto operações combinatoriais quanto geométricas. Um desafio clássico em algoritmos de geometria computacional é obter robustez: devido a erros de arredondamento associados à aritmética de ponto flutuante, tipicamente utilizada para manipular dados geométricos, SIGs frequentemente geram resultados incorretos, mapas com fronteiras inconsistentes (*slivers*) ou mesmo falham não gerando qualquer resultado [Goodchild and Gopal 1989].

Esse problema é ainda pior devido ao aumento da disponibilidade de dados espaciais de alta resolução visto que processá-los envolve realizar muitas operações geométricas. Existem técnicas que reduzem tais problemas como, por exemplo, o uso de tolerâncias na comparação de números de ponto-flutuante. Porém, elas não garantem o correto tratamento dos erros (apenas reduzem a chance de falha).

Uma técnica garantida (mas custosa) de realizar tais operações sem erros consiste em substituir números de ponto flutuante por números racionais de precisão arbitrária. Por exemplo, [Gruppi et al. 2016] utiliza números racionais para realizar a operação de simplificação de *polylines* evitando inconsistências topológicas.

Neste trabalho propomos o desenvolvimento de primitivas geométricas que são tanto exatas quanto eficientes. Como tais primitivas são utilizadas por diversos algoritmos de mais alto nível, uma vasta gama de aplicações podem se beneficiar

delas. Para obter eficiência, será utilizado o poder de processamento paralelo tanto de CPUs quanto de GPUs. A ideia consiste em utilizar números racionais de precisão arbitrária para representar as coordenadas geométricas. Além da representação exata, será utilizada a aritmética intervalar onde, para cada valor, também é armazenado um intervalo representado por números de ponto flutuante (que aproxima os valores exatos). As operações são inicialmente realizadas de forma eficiente com os valores aproximados. Se for detectado que os resultados calculados não são confiáveis, os cálculos são refeitos utilizando os números racionais exatos.

Como estudo de caso, foi desenvolvido um algoritmo eficiente e robusto para detecção de interseções entre pares de arestas 2D (uma primitiva geométrica que é utilizada como sub-rotina em diversas aplicações de SIG).

2. Erros de arredondamento

Valores não-inteiros normalmente são representados na memória do computador através de números de ponto flutuante. Contudo, muitos de tais valores não podem ser representados de maneira exata com números de ponto flutuante, sendo assim representados de maneira aproximada, causando erros de arredondamento, que se acumulam à medida em que operações aritméticas são realizadas.

Por exemplo, considere o predicado de orientação de pontos (que será utilizado no estudo de caso deste trabalho). Em 2D, dados três pontos p, q e r, esse predicado consiste em determinar se r está à esquerda, à direita, ou é colinear à reta formada por p e q. Para isto, verifica-se o sinal do determinante de uma matriz contendo as coordenadas desses 3 pontos: $\begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix}$

O determinante pode ter sinal positivo, negativo ou zero, o que implica, respectivamente, que r está à esquerda, à direita, ou é colinear à reta. Erros de arredonamento podem alterar esse sinal e tais problemas podem se propagar para operações de mais alto nível (por exemplo, para o predicado que testa se dois segmentos se interceptam). Assim, é necessário garantir a corretude do sinal do determinante.

Uma solução utilizada consiste em representar os valores através de números racionais de precisão arbitrária, mas o *overhead* [Pion and Fabri 2011] associado a esses números frequentemente inviabiliza o uso deles em aplicações que envolvem grandes massas de dados.

3. Aritmética Intervalar

A fim de realizar cálculos exatos eficientemente, neste trabalho são utilizadas as técnicas de aritmética intervalar e filtragem de aritmética [Pion and Fabri 2011, de Figueiredo and Stolfi 2004], aliadas ao fato de que apenas o sinal das expressões são suficientes para avaliar os predicados.

Cada número utilizado no algoritmo será representado pelo seu valor exato e por um intervalo de números de ponto flutuante que o aproxima. Será garantido que o intervalo sempre conterá o valor exato.

As operações aritméticas básicas são definidas em termos dos limites dos intervalos. Um número racional R é aproximado por um par de números de ponto

flutuante A e B, $A \leq R \leq B$. Cada operação aritmética é realizada inicialmente nos intervalos, que são ajustados para garantir que conterão os resultados exatos. Ao avaliar um predicado, os cálculos inicialmente são realizados com os intervalos. Se o resultado final for confiável, ele é retornado. Caso contrário (situação conhecida como *falha do intervalo*), os cálculos são refeitos utilizando aritmética exata. Essa técnica de se utilizar aproximações e filtrar os resultados não confiáveis (os reavaliando) é conhecida como filtragem de aritmética [Pion and Fabri 2011], e é utilizada, por exemplo, na biblioteca geométrica CGAL [The CGAL Project 2019].

Por exemplo, suponha que se deseja avaliar a expressão $a + b > c$. Para isso, basta verificar se $a + b - c$ possui sinal positivo. Se com o uso de aritmética intervalar for determinado que o resultado exato da expressão está no intervalo $[2.1, 2.3]$, é garantido que se os cálculos tivessem sido realizados de forma exata o resultado seria positivo. Por outro lado, se o intervalo resultante fosse $[-0.1, 0.2]$, o sinal da expressão não poderia ser avaliado de forma confiável com o uso dos intervalos, visto que o valor exato poderia ser negativo, nulo ou positivo.

Para garantir a corretude dos cálculos com intervalos, se faz necessário ajustar os limites após cada operação aritmética. Para a operação de adição, por exemplo, o arredondamento dos números de ponto flutuante que representam os limites inferior e superior do intervalo resultante deve ocorrer nos sentidos $-\infty$ e $+\infty$, respectivamente. Esse requisito chave para uma implementação correta da aritmética intervalar está disponível graças ao padrão IEEE-754 para números de ponto flutuante. O padrão garante que três modos de arredondamento (arredondamento para o valor de ponto flutuante representável mais próximo, sempre no sentido $-\infty$ ou sempre no sentido $+\infty$) são acessíveis e podem ser alternados em tempo de execução.

Vale mencionar que algoritmos geométricos frequentemente utilizam predicados e operações de construção, sendo que o uso da filtragem de aritmética proposta neste trabalho se restringe aos predicados. Por exemplo, um algoritmo que calcula a sobreposição (*overlay*) de mapas vetoriais utiliza predicados para verificar se pares de arestas (dos dois mapas de entrada) se interceptam. Nessa etapa a filtragem de aritmética pode reduzir a necessidade de uso de aritmética exata. Porém, para gerar o mapa resultante eventualmente as arestas que se interceptam precisam ser divididas nos pontos de interseção e, portanto, tais pontos precisam ser calculados (processo de construção). Dessa forma, se for desejado obter coordenadas exatas para tais pontos na saída do algoritmo, então a aritmética exata será necessária.

4. Acelerando os cálculos com programação paralela

[Magalhães et al. 2017] propôs um algoritmo de interseção de objetos 3D utilizando números racionais para se obter precisão. Uma boa eficiência foi obtida com o uso de filtragem de aritmética e o código foi paralelizado para CPUs multi-core.

Uma forma de se acelerar algoritmos é utilizar GPUs. Porém, algoritmos geométricos exatos normalmente demandam muita alocação dinâmica de memória (para realocar números de precisão arbitrária à medida em que crescem) e cálculos com valores inteiros. Satisfazer ambas demandas em GPUs é um desafio. Para resolver este problema, propomos uma estratégia híbrida, onde a GPU é empregada para avaliar predicados utilizando aritmética intervalar. Assim como na CPU, a

GPU também oferece métodos para alterar o modo de arredondamento dos números de ponto flutuante, o que é fundamental para realizar cálculos com intervalos. Uma vantagem adicional é que essas mudancas no modo de arredondamento podem ser feitas de forma mais eficiente nas GPUs do que em CPUs [Collange et al. 2012].

Após o uso da aritmética intervalar, os resultados não confiáveis são filtrados e re-avaliados de forma exata e paralela utilizando números racionais na CPU.

Um desafio nessa implementação é que GPUs são dispositivos *SIMT* (*single instruction, multiple thread*) e, dessa forma, para se obter um bom desempenho é importante que o algoritmo aplique uma mesma operação a múltiplos conjuntos de dados. Além disso, outro desafio é o processo de comunicação entre a CPU e a GPU. Se a cada cálculo todos os dados precisarem ser transferidos entre os dois dispositivos, o *overhead* dessa comunicação pode reduzir o ganho de desempenho obtido com os cálculos. Devido a esses motivos, os algoritmos desenvolvidos utilizando a estratégia proposta precisam ser cuidadosamente projetados considerando essas restrições.

5. Estudo de caso

Como estudo de caso, considere o seguinte problema: dado um conjunto de pares de segmentos de retas em 2D, detectar quais pares se interceptam. Esse problema clássico de geometria possui várias aplicações na área de SIG como, por exemplo, calcular a sobreposição (*overlay*) de mapas vetoriais, detectar se polígonos se interceptam, etc. De fato, essa é uma operação básica em bancos de dados geográficos e sua aceleração pode trazer ganhos em diversas operações de mais alto nível.

Dado um par de segmentos de retas (r_1, r_2), o predicado que verifica se eles se interceptam pode ser implementado avaliando 4 orientações em 2D para verificar se os dois extremos de r_1 estão em lados opostos (ou seja, se possuem orientações opostas) em relação a r_2 e vice-versa.

Os pares são alocados em um *array* e enviados em *batch* para a GPU, de forma que cada *thread* fica responsável por um par. As *threads* executam as 4 orientações para avaliar o predicado e retornam se houve ou não interseção, ou indicam uma falha de intervalo. Os pares que não passaram na filtragem são então reavaliados pela CPU utilizando aritmética exata. Como mostrado por [Brönnimann et al. 2001], o número esperado de falhas de intervalo é pequeno. Os experimentos realizados nesse trabalho (descritos na seção 5.1) reforçam essa afirmação.

5.1. Experimentos

Para analisar o impacto das ideias propostas nesse trabalho, o algoritmo para detecção de interseções de segmentos de reta descrito na seção anterior foi implementado e testado com dois pares de mapas vetoriais: UsCounty e UsAquifers (com, respectivamente, 4 milhões e 350 mil segmentos) e UsCounty e UsCountyRotacionado (ambos com 4 milhões de segmentos, sendo UsCountyRotacionado uma versão de UsCounty rotacionada em 0.1° no sentido anti-horário).

A implementação foi feita em C++. Foi utilizado GMP para aritmética de precisão arbitrária e OpenMP e CUDA para se obter o paralelismo na CPU e GPU. Os testes foram realizados em um computador com GPU NVIDIA GeForce GTX 1070 Ti e processador AMD Ryzen 5 1600, com 6 núcleos e 3.2GHz.

Implementação	GMP*	Intervalar*	CGAL*	GMP	Intervalar	GPU
Mapas	UsCounty e UsAquifers					
Pre-processamento	7,884	0,812	2,628	1,610	0,392	0,164
Interseção	42,816	4,059	0,023	11,198	0,612	0,096
Tempo total	50,700	4,871	2,651	12,808	1,004	0,260
# Testes ($x10^3$)	12.756	12.756	159	12.756	12.756	12.756
Mapas	UsCounty e UsCountyRotacionado					
Pré-processamento	14,532	1,422	7,482	2,798	0,454	0,251
Interseção	675,616	63,677	1,027	194,918	9,422	1,367
Tempo total	690,148	65,099	8,509	197,716	9,876	1,618
# Testes ($x10^3$)	216.543	216.543	11.254	216.543	216.543	216.543

Tabela 1. Tempos (em segundos) de execução das diferentes implementações nos pares de mapas avaliados. As versões sem * são sequenciais. A linha # Testes indica o número de testes de interseção realizados pelas diferentes versões.

Os resultados foram comparados com a CGAL, biblioteca de geometria computacional utilizada como *backend* exato pelo banco de dados espacial PostGIS. Os pares de segmentos a serem testados para interseção foram gerados aplicando um índice espacial nos mapas. Por performance, foi implementado um índice paralelo por meio de uma grade regular similar ao utilizado por [Magalhães et al. 2017] (porém, a versão utilizada neste trabalho aplica as técnicas mencionadas acima). Vale mencionar que o índice empregado pelo CGAL é sequencial (baseado na técnica de *sweep-line*, difícil de ser paralelizada) e, portanto, tanto o número de testes realizados por tal método quanto o tempo de execução são diferentes.

A Tabela 1 apresenta os resultados obtidos. Foram implementadas 5 versões do algoritmo para possibilitar a comparação entre as diferenças de performance advindas da aplicação de cada estratégia. A implementação com rótulo *GMP* utiliza apenas números racionais, enquanto a com rótulo *Intervalar* utiliza aritmética intervalar. A quinta variante (*GPU*) é a implemetação completa das ideias propostas nesse trabalho, a qual utiliza a GPU para filtragem de aritmética e a CPU em paralelo para cálculos exatos, quando necessários.

Considerando o segundo caso de teste, por exemplo, o tempo para se detectar as interseções foi 47 vezes menor utilizando a CPU (em comparação com o uso de aritmética intervalar em sequencial). Note que, embora o índice utilizado pelo CGAL seja mais eficiente para reduzir o número de pares de segmentos a serem testados por interseção, o tempo gasto com isso não é recuperado, fazendo com que o algoritmo proposto seja até 10 vezes mais rápido do que o CGAL.

Vale mencionar que se for considerado apenas o tempo gasto avaliando predicados geométricos (ou seja, desconsiderando alocação de memória, transferência de dados e outros passos do algoritmo), a GPU foi até 289 vezes mais rápida do que a implementação sequencial em CPU. Isso indica que algoritmos que fazem uso pesado de predicados geométricos podem se beneficiar ainda mais da técnica proposta.

Finalmente, o número de casos onde aritmética exata foi necessária, conforme

esperado, foi pequeno. No segundo caso de teste, por exemplo, 0.000002% dos predicados falharam (exigindo uma re-avaliação com aritmética exata).

6. Conclusões e trabalhos futuros

Nesse artigo foi proposto o uso de GPUs para acelerar a avaliação exata de predicados geométricos, primitiva essencial para algoritmos exatos de SIG. Apesar da implementação de aritmética exata ser um desafio para a arquitetura das GPUs, a filtragem de aritmética permite que os predicados sejam avaliados de forma robusta nesses dispositivos. As poucas instâncias cujos resultados não podem ser garantidos pela aritmética intervalar, podem ser recalculadas de forma exata na CPU, o que garante eficiência e robustez ao processo.

Para avaliar as ideias apresentadas nesse trabalho, foi implementado um algoritmo eficiente e exato para detectar interseções entre pares de segmentos de reta. A utilização da GPU para filtragem de aritmética, se comparada à versão sequencial na CPU, proporcionou um ganho de desempenho de até 289 vezes para a etapa de avaliação dos predicados, ou de até 40 vezes se for considerado o tempo total de execução (incluindo um pré-processamento para a criação do índice espacial).

Os próximos passos deste trabalho em andamento incluem a aplicação da ideia a outros algoritmos de SIG. Algoritmos exatos de interseção de mapas e localização de pontos, por exemplo, que já utilizam a estratégia de filtragem de aritmética podem ser facilmente adaptados para uso da GPU.

Referências

Brönnimann, H., Burnikel, C., and Pion, S. (2001). Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Applied Mathematics*, 109(1-2):25–47.

Collange, S., Daumas, M., and Defour, D. (2012). Chapter 9 - interval arithmetic in CUDA. In mei W. Hwu, W., editor, *GPU Computing Gems Jade Edition*, Applications of GPU Computing Series, pages 99 – 107. Morgan Kaufmann, Boston.

de Figueiredo, L. H. and Stolfi, J. (2004). Affine arithmetic: Concepts and applications. *Numerical Algorithms*, 37(1):147–158.

Goodchild, M. F. and Gopal, S. (1989). *The accuracy of spatial databases*. CRC Press.

Gruppi, M. G., de Magalhães, S. V., Andrade, M. V., Franklin, W. R., and Li, W. (2016). using rational numbers and parallel computing to efficiently avoid round-off errors on map simplification. *Revista Brasileira de Cartografia*, 68(6).

Magalhães, S. V. G., Andrade, M. V. A., and Franklin, W. R. (2017). Fast exact parallel 3d mesh intersection algorithm using only orientation predicates. In *Proc. 25st ACM SIGSPATIAL*, SIGSPATIAL'17, New York, NY, USA. ACM.

Pion, S. and Fabri, A. (2011). A generic lazy evaluation scheme for exact geometric computations. *Sci. Comput. Program.*, 76(4):307 – 323.

The CGAL Project (2019). *CGAL User and Reference Manual*. CGAL Editorial Board, 4.14.1 edition.