Uma abordagem eficiente para o cálculo de viewshed em terrenos armazenados em memória externa

Chaulio R. Ferreira[1], **Marcus V. A. Andrade**[1],
Salles V. G. Magalhães[1], **André M. Pompermayer**[1]

[1]Departamento de Informática – Universidade Federal de Viçosa (UFV)
Campus da UFV – 36.570-000 – Viçosa – MG - Brazil

{chaulio.ferreira,marcus,salles,andre.pompermayer}@ufv.br

Abstract. *An important GIS application is computing the viewshed of a point on a DEM terrain, i. e. determining the visible region from this point. In some cases, it is not possible to process high resolution DEMs entirely in internal memory and, thus, it is important to develop algorithms to process such data in the external memory. This paper presents an efficient algorithm for handling huge terrains in external memory. As tests have shown, this new method is more efficient than other methods described in the literature.*

Resumo. *Uma importante aplicação de SIGs é o cálculo da região visível (viewshed) a partir de um determinado ponto em um terreno representado por um modelo digital de elevação (MDE). Muitas vezes, o processamento de MDEs de alta resolução não pode ser realizado em memória interna e, portanto, é importante o desenvolvimento de algoritmos para processar estes dados em memória secundária. Este trabalho apresenta um algoritmo para cálculo de viewshed que é capaz de lidar com grande volume de dados em memória externa de forma eficiente. Os testes realizados indicam que o método proposto é mais eficiente do que outros métodos descritos em literatura.*

1. Introdução

Os recentes avanços tecnológicos em sensoriamento remoto têm produzido uma grande quantidade de dados de alta resolução sobre a superfície terrestre, o que tem aumentado a necessidade de se desenvolver novas técnicas de Sistemas de Informação Geográfica (SIGs) para lidar com este enorme volume de dados [Laurini and Thompsom 1992].

Uma forma muito utilizada para se representar a superfície da Terra de forma aproximada é através de um modelo digital de elevação (MDE) que armazena as elevações de pontos amostrados sobre a superfície terrestre. Esses pontos podem ser amostrados de maneira irregular e ser armazenados como uma rede triangular irregular (*TIN - Triangulated Irregular Network*) ou de maneira regular sendo armazenados numa matriz [Felgueiras 2001]. Neste trabalho será adotada a segunda forma de representação do MDE. Muitas vezes estas matrizes necessitam de mais espaço de armazenamento do que tem-se disponível na memória interna da maioria dos computadores atuais. Por exemplo, um terreno de $100km \times 100km$ mapeado com resolução de $1m$ resulta em 10^{10} pontos. Supondo que sejam utilizados 2 *bytes* para armazenar a elevação de cada ponto, são necessários mais de 18 GB para representar este terreno.

Desta forma, é importante o desenvolvimento de algoritmos específicos para processamento de dados armazenados em memória externa. Vale ressaltar que normalmente os algoritmos tradicionais de análise e processamento de dados geográficos buscam otimizar o tempo de processamento em CPU, sem grandes preocupações com o tempo de acesso à memória. Mas, por outro lado, o projeto e análise de algoritmos para memória externa devem focar-se em minimizar os acessos a disco, uma vez que estes são da ordem de 10^6 vezes mais lentos do que os acessos à memória interna [Dementiev et al. 2005].

Mais especificamente, algoritmos que processam dados em memória externa devem ser projetados e analisados considerando um modelo computacional que os avaliem considerando as operações de transferência de dados em vez das operações de processamento interno. Um desses modelos, proposto por Aggarwal e Vitter [Aggarwal and Vitter 1988], determina a complexidade dos algoritmos com base no número de operações de E/S (entrada/saída) executadas. Este modelo será descrito com maiores detalhes na Seção 2.3.

Dentre as várias aplicações na área de SIG, há aquelas relacionadas a questões de visibilidade, como determinar o número mínimo de torres de celular necessárias para cobrir uma região [Ben-Moshe et al. 2002], otimizar o número e a posição de guardas para vigiar uma região [Bespamyatnikh et al. 2001], etc. Tais aplicações utilizam o conceito de observador e alvo: o observador tem o objetivo de visualizar (observar) outros objetos (os alvos) em um terreno, sendo que os observadores possuem um limite máximo para o alcance de sua visão chamado de raio de interesse. Por exemplo, uma torre de telefonia celular pode ser considerada um observador cujo raio de interesse corresponde ao alcance do sinal da torre e cujos alvos são os usuários do serviço de telefonia. A partir desse conceito, pode-se calcular o mapa de visibilidade (*viewshed*) de um ponto p do terreno, que indica a região do terreno que é visível por um observador posicionado em p. Há diversos métodos para cálculo de *viewshed* em memória interna, como os propostos por Franklin e Ray [Franklin and Ray 1994, Ray 1994] e Kreveld [van Kreveld 1996]. Além disso, há também alguns métodos eficientes para processar terrenos armazenados em memória externa, como o método proposto por Haverkort et al. [Haverkort et al. 2007], que utilizam uma adaptação do método de Kreveld para memória externa e o método *EMViewshed* recentemente proposto por Andrade et al. [Andrade et al. 2011], descrito na seção 2.2, que é cerca de 6 vezes mais rápido do que o método proposto por Haverkort.

Este trabalho apresenta um novo método denominado *TiledViewshed*, que é cerca de 4 vezes mais eficiente do que o método *EMViewshed*. A idéia básica deste novo método consiste em adaptar o algoritmo proposto por Franklin e Ray [Franklin and Ray 1994] alterando a forma como os dados em memória externa são acessados. Para isto, foi utilizada uma estrutura de dados que gerencia as transferências de dados entre as memórias interna e externa, buscando diminuir o número de acessos a disco. Maiores detalhes sobre esta estrutura encontram-se na seção 3 e os resultados dos testes da comparação deste novo método com o *EMViewshed* são apresentados na seção 5.

2. Referencial teórico

2.1. Visibilidade em terrenos

Um *terreno* corresponde a uma região da superfície da Terra cujo relevo é representado por um modelo digital de elevação (MDE) que pode ser representado por uma

malha triangular irregular (*triangulated irregular network*, ou *TIN*), por linhas de contorno ou por uma matriz que contém elevações de pontos posicionados em intervalos regularmente espaçados. Devido à sua simplicidade e ao grande volume de dados disponíveis na forma matricial, esta é a representação utilizada nos algoritmos deste trabalho. É importante observar que há métodos eficientes para a conversão entre as diferentes formas de representação, e portanto, esta escolha não representa uma restrição relevante [Felgueiras 2001].

Um *observador* é um ponto no espaço a partir do qual se deseja visualizar ou comunicar com outros pontos no espaço, chamados de *alvos*. As notações usuais para um observador e um alvo são, respectivamente, O e T e estes pontos podem estar a certas alturas acima do terreno denotadas respectivamente por h_O e h_T. Os pontos do terreno verticalmente abaixo de de O e T são denominados pontos-base e são denotados por O_b e T_b, respectivamente.

O *raio de interesse*, R, de um observador O representa o alcance de sua visão, ou seja, o valor máximo da distância em que ele é capaz de enxergar ou se comunicar. Por conveniência, a distância entre um observador O e um alvo T é definida como a distância entre O_b e T_b.

Um alvo T é *visível* a partir um observador O se, e somente se, $|T_b - O_b| \leq R$ e não há nenhum ponto da superfície do terreno interceptando o segmento de reta $\overline{OT}$, que é chamado de *linha de visão*, ou *line of sight* (*LOS*). Um exemplo de cálculo de visibilidade é mostrado na Figura 1: o alvo T_1 é visível a partir de O, mas T_2 não é visível, pois $\overline{OT_2}$ é bloqueado por uma região do terreno.

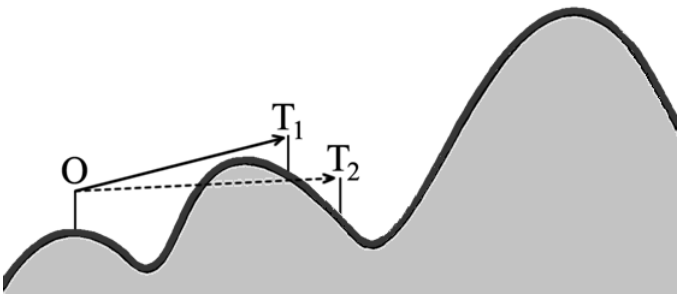

Figura 1. Cálculo da visibilidade em um corte vertical do terreno. O alvo T_1 é visível a partir de O e T_2 não é visível.

É importante ressaltar que a verificação de que a linha de visão intercepta ou não o terreno não é trivial. O problema é que a matriz de elevação contém informações somente sobre alguns pontos discretos do terreno, enquanto as linhas de visão são contínuas e geralmente passam entre pontos adjacentes, sem interceptá-los, o que muitas vezes requer um método de interpolação entre os pontos conhecidos [Magalhães et al. 2011].

O *viewshed* (ou mapa de visibilidade) de um observador O é o conjunto de pontos do terreno cujos alvos correspondentes são visíveis a partir de O.

2.2. Algoritmos de cálculo de *viewshed* em memória interna

Dados um terreno representado por uma matriz de elevação de dimensões $n \times n$, as coordenadas (x, y) do observador O, o seu raio de interesse R e as alturas h_O e h_T, o objetivo

de um algoritmo de cálculo do *viewshed* de O é determinar, para cada célula da matriz, se seu alvo correspondente é visível por O. Uma das maneiras de representar o *viewshed* calculado é utilizando uma matriz de bits com dimensão $n \times n$ onde um bit com valor 1 indica que o alvo T associado àquela célula do terreno é visível por O, enquanto um bit com valor 0 indica que T não é visível.

O algoritmo proposto neste trabalho para o cálculo do *viewshed* em grandes terrenos armazenados em memória externa é baseado no algoritmo para memória interna proposto por Franklin e Ray [Franklin and Ray 1994, Ray 1994], que será descrito de forma resumida a seguir.

Este algoritmo assume que todas as células são inicialmente não visíveis e realiza um processamento iterativo para determinar quais células são visíveis a partir de O. Primeiramente é definida uma região quadrangular S que envolva o círculo de raio R centrado em O (por exemplo, veja a figura 2(a) onde é mostrada esta região para $R = 4$). Daí, são traçados $8R$ segmentos de reta ligando O a cada uma das células na borda de S. Cada um destes segmentos define um corte vertical no terreno - a figura 2 exibe um exemplo desses segmentos e de um desses cortes.

O passo seguinte consiste em determinar quais células fazem parte de cada corte vertical. Para alguns casos, como os dos segmentos $\overline{OA}$ e $\overline{OE}$ mostrados na figura 2(a), é fácil determinar estas células. No entanto, para a maioria dos segmentos essa determinação não é tão simples, como é o caso dos segmentos $\overline{OB}$, $\overline{OC}$ e $\overline{OD}$, onde é necessário determinar quais células são mais relevantes para cada segmento. Esse problema equivale à rasterização de segmentos e pode ser solucionado utilizando o algoritmo de Bresenham [Bresenham 1965], de modo que seja selecionada apenas uma célula para cada coordenada X ou Y (dependendo da inclinação do segmento).

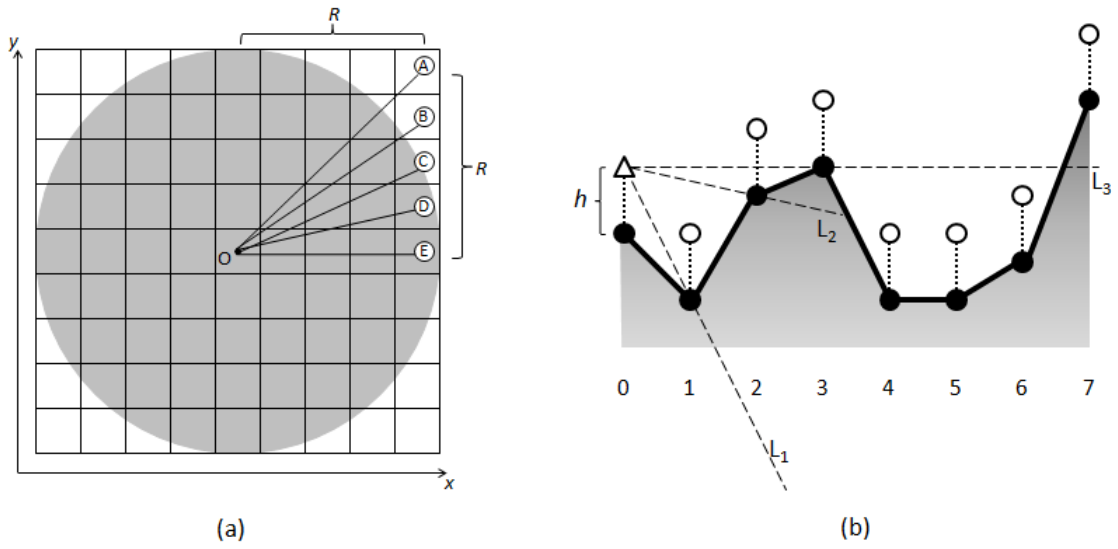

Figura 2. Exemplos de segmentos em um terreno (a) e de um corte vertical definido por um segmento (b)

Após a determinação das células de cada corte vertical, o próximo passo é percorrer estas células verificando quais são visíveis a partir de O. Seja um segmento composto pelas células $c_0, c_1, \cdots, c_k$, sendo que o observador está posicionado na célula c_0 e c_k é a

última célula na região S dentro do raio de visão do observador. Então o processo consiste em inicialmente definir as células c_0 e c_1 como visíveis e inicializar μ, que armazena a maior inclinação de uma linha de visão já processada, com a inclinação da reta que passa pelos pontos c_0 e c_1. A partir daí, cada célula c_i é processada em ordem crescente de i, analisando-se a inclinação da reta que liga o observador ao alvo que está posicionado acima de c_i; se esta inclinação é maior ou igual a μ então a célula c_i é marcada como visível e μ é atualizado com o valor da inclinação da reta que passa por O e c_i.

Aplicando este algoritmo ao exemplo dado na figura 2(b) temos que os alvos posicionados acima dos pontos $0, 1, 2, 3$ e 7 são visíveis a partir do observador posicionado acima do ponto 0 (representado por um triângulo) e os alvos posicionados acima dos pontos $4, 5$ e 6 não são visíveis.

2.3. Algoritmos eficientes para E/S

Durante o processamento de grande volume de dados, a transferência de dados entre a memória interna (mais rápida) e o armazenamento externo (mais lento) frequentemente torna-se o gargalo do processamento. Portanto, o projeto e análise de algoritmos usados para processar esses dados precisam ser feitos sob um modelo computacional que avalia as operações de entrada e saída (E/S). Um modelo frequentemente utilizado foi proposto por Aggarwal e Vitter [Aggarwal and Vitter 1988]. Nesse modelo, cada operação de E/S corresponde à transferência de um bloco de tamanho B entre a memória externa e a memória interna. O desempenho do algoritmo é determinado considerando-se o número de operações de E/S executadas.

A complexidade de um algoritmo é definida com base na complexidade de problemas fundamentais como varredura ($scan$) e ordenação ($sort$) de N elementos contíguos armazenados em memória externa. Se M é o tamanho da memória interna disponível, estas complexidades são definidas como:

$$scan\,(N) = \Theta\left(\frac{N}{B}\right) \quad \text{e} \quad sort\,(N) = \Theta\left(\frac{N}{B} log_{(\frac{M}{B})}\left(\frac{N}{B}\right)\right)$$

É importante salientar que normalmente $scan(N) < sort(N) << N$ e, então, um algoritmo que realiza $O(sort(N))$ operações de E/S é significantemente mais eficiente do que um que realiza $O(N)$ operações. Assim, muitos algoritmos tentam reorganizar os dados na memória externa com o objetivo de diminuir o número de operações de E/S feitas.

2.4. O método *EMViewshed*

A estratégia proposta por Andrade et al. [Andrade et al. 2011] consiste em gerar e armazenar em memória externa uma lista com informações de todas as células do terreno e ordená-la de acordo com a ordem em que estas células serão processadas pelo algoritmo. Assim, o algoritmo pode percorrer esta lista sequencialmente, evitando acessos aleatórios à memória externa.

Mais especificamente, o algoritmo cria uma lista Q de pares (c, i), onde c é uma célula e i é um índice que indica "quando"c deveria ser processada. Isto é, se uma célula c está associada a um índice k, então c seria a k-ésima célula a ser processada.

Para determinar os índices é utilizado um processo similar ao descrito na seção 2.2, onde são traçadas diversas linhas de visão em sentido anti-horário, e as células recebem índices numerados de forma crescente em cada linha de visão. Uma mesma célula pode receber vários índices (e possuir várias cópias em Q) já que ela pode ser interceptada por múltiplas linhas de visão.

Depois de criada, a lista Q é ordenada usando os índices como chave de comparação, e então as células são processadas na ordem da lista ordenada por um algoritmo similar ao de memória interna que, neste caso, lê os dados de elevação diretamente de Q. Além disso, este algoritmo utiliza uma outra lista Q' onde as células visíveis são inseridas. Após o processamento de todas as células, Q' é ordenada lexicograficamente pelas coordenadas x e y e as células visíveis são armazenadas em um arquivo de saída, onde as posições visíveis são indicadas por 1 e as não visíveis por 0.

Um ganho de eficiência no processo é alcançado mantendo parte das matrizes em memória interna. As células que estão em memória interna não são inseridas em Q e Q' e, quando uma célula precisa ser processada, o algoritmo verifica se esta célula está na memória interna. Se estiver, ela é processada normalmente; caso contrário, ela é lida de Q.

Para analisar a complexidade deste algoritmo, é necessário utilizar um modelo como o descrito na Seção 2.3. Seja T o terreno representado por uma matriz de elevação de dimensões $n \times n$, ou seja, que contenha n^2 células. No primeiro passo do algoritmo, para ler as células do terreno e criar a lista Q, são realizadas $O(scan(R^2))$ operações de E/S. Neste passo, como mostrado na seção 2.2, o algoritmo traça $8R$ linhas de visão, cada uma contendo R células. Assim, a lista Q contém $O(R^2)$ elementos.

Em seguida esta lista é ordenada e então percorrida sequencialmente para se calcular a visibilidade das células, operações que apresentam complexidades $O(sort(R^2))$ e $O(scan(R^2))$, respectivamente.

Finalmente, a lista de pontos visíveis que contém, no máximo, $O(R^2)$ células é ordenada e o *viewshed* do terreno é gravado em disco, operações que apresentam complexidades $O(sort(R^2))$ e $O(scan(n^2))$, respectivamente. Como no pior caso $R = O(n)$, temos que a complexidade do algoritmo é:

$$O\left(sort\left(R^2\right)\right) = O\left(sort\left(n^2\right)\right) = O\left(\frac{n^2}{B} log_{\left(\frac{M}{B}\right)}\left(\frac{n^2}{B}\right)\right)$$

3. O método *TiledViewshed*

Ao analisar o algoritmo descrito na seção 3, pode-se perceber que o acesso às posições das matrizes apresenta um padrão radial, ou seja, as células são acessadas a partir daquela que contém o observador até cada célula que está no limite de seu raio de interesse, traçando diversas linhas de forma circular. Este padrão de acessos apresenta a propriedade de localidade de referência espacial: as células acessadas em um curto espaço de tempo estão, na maioria das vezes, próximas umas das outras na matriz. O problema é que, em geral, uma matriz bidimensional é armazenada de forma linear na memória e, por isso, muitas vezes células que são vizinhas na matriz ficam armazenadas em posições distantes umas das outras na memória. Como normalmente a localidade de referência

espacial utilizada pela hierarquia de memória do computador é baseada no acesso sequencial [Patterson and Hennessy 2008], esta forma de representação não tira proveito da localidade de referência espacial em termos bidimensionais, tornando o acesso ineficiente.

Para aproveitar a propriedade de localidade espacial e tentar diminuir o número de acessos a disco, este trabalho propõe um método denominado *TiledViewshed*, cuja estratégia consiste em adaptar o método descrito na seção 2.2 de forma que todos acessos realizados às matrizes sejam gerenciados por uma estrutura de dados denominada *TiledMatrix* [Magalhães et al. 2012], que é capaz de armazenar e gerenciar grandes matrizes em memória externa. Mais especificamente, um objeto do tipo *TiledMatrix* representa uma matriz que é dividida em blocos menores de dimensões fixas que são armazenados de forma sequencial em um arquivo na memória externa. Então, esta estrutura de dados gerencia a transferência de blocos entre as memórias interna e externa sempre que necessário. Em outras palavras, alguns blocos do terreno ficam armazenados em memória interna enquanto estiverem sendo processados, e podem voltar à memória externa quando não forem mais necessários, dando lugar a outros blocos. Desta forma, a estrutura de dados funciona como uma memória *cache* gerenciada pela aplicação, que busca predizer quais serão os próximos blocos do terreno a terem posições acessadas no processamento, mantendo-os na memória interna.

Uma questão importante a se considerar na implementação desta estrutura refere-se à política utilizada para determinar qual bloco será escolhido para ceder espaço a novos blocos. Neste trabalho utilizou-se a estratégia de retirar da memória interna aquele que está a mais tempo sem ser acessado pela aplicação. Em outras palavras, sempre que uma célula de um bloco é acessada, este bloco é marcado com um *timestamp*. Quando for necessário retirar um bloco da memória interna para carregar outro, será escolhido aquele que tive o menor *timestamp*. Esta estratégia foi adotada baseado no fato de que há uma certa localidade no processamento das células pelo algoritmo. Isto é, se há um bloco que está a algum tempo sem ser acessado (nenhuma de suas células é processada) então há uma grande chance de que todas as células daquele bloco já tenham sido processadas e o bloco não precisará mais acessado.

Para ilustrar este processo, considere uma matriz que foi dividida em 5×5 blocos e uma memória interna capaz de armazenar no máximo 5 desses blocos. Suponha que em determinado momento do processamento os blocos de números 8, 9, 12, 13 e 14 estejam na memória interna, como mostrado na Figura 3(a). Se for requisitado acesso a alguma célula que está contida em algum destes blocos, não será necessário buscá-la na memória externa, e o acesso será feito de forma mais eficiente. Por outro lado, se for requisitado acesso a alguma célula de outro bloco, diz-se que ocorreu uma *cache miss*, e esta célula deverá ser buscada na memória externa. Porém, por acreditar que logo em seguida serão requisitados acessos a outras células deste mesmo bloco, a estrutura de dados transfere e carrega para a memória interna o bloco inteiro que contém a célula requisitada, substituindo um dos blocos que já estavam carregados. Considere, por exemplo, uma requisição de acesso a alguma célula do bloco 4, e que o bloco 14 é o que está a mais tempo sem ser acessado dentre aqueles carregados em memória interna e, portanto, é o que deverá ser substituído. Antes de realizar a substituição, é necessário verificar se alguma instrução de escrita foi executada no bloco 14 enquanto ele esteve em memória interna. Se sim, seus dados devem ser atualizados na memória externa. Caso contrário não é necessário

realizar esta atualização. Depois dessa verificação, o bloco 4 é copiado para a memória interna, resultando no estado representado na Figura 3(b). Assim, para *cada cache miss* ocorrida, acontece exatamente um acesso de leitura e no máximo um acesso de escrita a um bloco na memória externa.

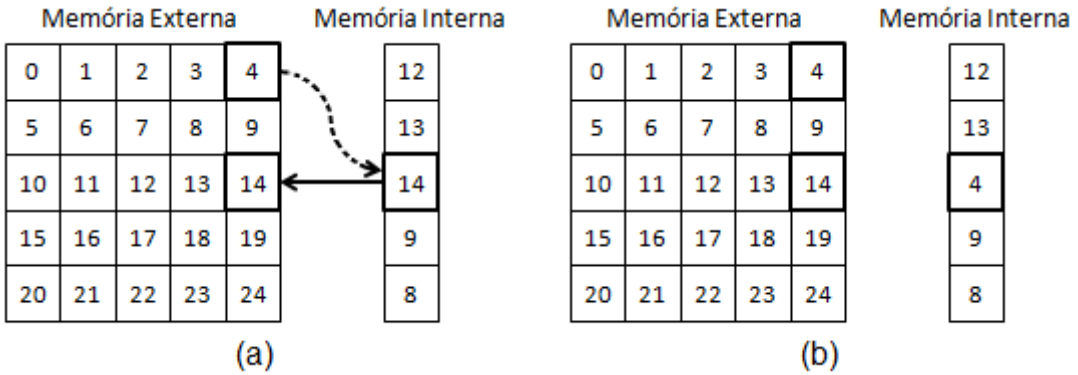

Figura 3. Transferências entre as memórias interna e externa gerenciadas pela classe *TiledMatrix*. Em (a), uma célula do bloco 4 é acessada e, como esse bloco não está na memória principal, ele é carregado, substituindo o bloco 14, que foi utilizado menos recentemente. A Figura (b) apresenta o estado da memória interna após a troca do bloco 14 pelo bloco 4.

Além de tirar proveito de dados com localidade de referência espacial, outra grande vantagem de utilizar esta estrutura é que ela proporciona uma maior facilidade para a adaptação de algoritmos para memória interna já existentes, de modo que seus desempenhos considerando dados armazenados em memória externa sejam melhorados. Basicamente, é necessário somente substituir a utilização de matrizes tradicionais pela utilização da classe *TiledMatrix*. Desta forma, esta estratégia pode ser utilizada em diversas aplicações, não somente na área de SIG, mas sempre que forem utilizadas matrizes maiores do que o espaço disponível em memória interna e que o acesso às posições das matrizes apresente um padrão de localidade de referência espacial.

4. Complexidade do algoritmo

As dimensões dos blocos utilizados e o número máximo de blocos que a memória interna utilizada pode armazenar têm influência direta no desempenho do algoritmo. Considere, por exemplo, a inicialização da matriz de elevação, em que uma parte do terreno é carregada na estrutura *TiledMatrix*. Mais especificamente, nesta etapa são copiadas para a estrutura as células que estão contidas em um quadrado de dimensões $(2R+1) \times (2R+1)$ com centro no observador O, onde R é o raio de interesse de O.

Por questões de eficiência, é importante que a memória interna disponível seja suficiente para armazenar no mínimo $\frac{2R+1}{t}$ blocos, onde t é a dimensão de cada lado dos blocos. Esta condição é necessária para que a inicialização da estrutura *TiledMatrix* possa ocorrer sem que seja necessário carregar um mesmo bloco mais de uma vez, ou seja, para que não ocorram *cache misses*. Para ilustrar a necessidade dessa condição, suponha que uma matriz de elevação de dimensões 50×50 seja armazenada em uma *TiledMatrix* M que a divide em blocos de dimensões 10×10 e que a memória interna comporte apenas 4 desses blocos. Note que, devido ao padrão que remove o bloco menos recentemente utilizado, quando os 5 últimos elementos da primeira linha da matriz forem copiados para M

o bloco que contém os 5 primeiros elementos dessa linha deverá ser removido da memória interna. Porém, ao copiar os 5 primeiros elementos da próxima linha da matriz para M, esse bloco deverá voltar novamente para a memória interna. Esse padrão de acesso ocorrerá em todas as linhas de M e, assim, esse processo seria ineficiente. Portanto, para realizar a análise de complexidade do algoritmo, será suposto que o tamanho da memória interna atenda a esta restrição e, assim, cada bloco será carregado apenas uma vez durante a inicialização dessa estrutura de dados.

Como o algoritmo precisa carregar da matriz de elevação $(2R + 1) \times (2R + 1)$ células, a inicialização do algoritmo realiza $O(scan(R^2))$ operações de E/S.

Durante a etapa do algoritmo onde são traçadas diversas linhas de visão em sentido anti-horário para calcular o *viewshed* de O, também será necessário carregar cada bloco do terreno uma vez, com exceção dos blocos que contém simultaneamente células das primeiras e últimas linhas de visão a serem processadas, que precisarão ser carregados duas vezes. Como o número de células processadas é $O(R^2)$ e cada célula é transferida da memória externa no máximo duas vezes, esta etapa também é realizada em $O(scan(R^2))$ operações de E/S.

Depois de calculado, o *viewshed* é armazenado em um arquivo de saída com um *bit* para cada célula do terreno. Se o terreno tiver dimensões $n \times n$, serão efetuadas $O(scan(n^2))$ operações de E/S. Portanto, a complexidade do algoritmo *TiledViewshed* é:

$$O\left(scan\left(n^2\right)\right) = O\left(\frac{n^2}{B} log_{\left(\frac{M}{B}\right)}\left(\frac{n^2}{B}\right)\right)$$

5. Resultados

O método *TiledViewshed* foi implementado em C++ e compilado com o g++ 4.5.2. Optou-se por realizar os testes em um computador com pouca memória RAM para que fosse necessário realizar operações de processamento em memória externa mesmo para terrenos não tão grandes (como os com 10000^2 e 20000^2 células). Por isso, os testes foram executados em um PC Pentium 4 com 3.6GHz, 1GB de RAM, HD Sata de 160GB e 7200RPM. O sistema operacional utilizado foi o o Linux, distribuição Ubuntu 11.04 de 64bits. Dos 1024MB disponíveis em memória interna, foram utilizados 800MB para armazenar os dados, e os demais foram reservados para uso do sistema.

Os terrenos utilizados nos testes foram obtidos na página do *The Shuttle Radar Topography Mission* (SRTM) [Rabus et al. 2003]. Os dados correspondem a duas regiões distintas dos EUA amostradas em diferentes resoluções, gerando assim MDEs de diferentes tamanhos (10000^2, 20000^2, 30000^2, 40000^2 e 50000^2). Para cada tamanho de terreno, foi calculada a média do tempo necessário para processamento das duas regiões.

Em todos os testes foi considerado o pior caso, ou seja, o valor para o raio de interesse utilizado foi grande o suficiente para cobrir todo o terreno. Para avaliar a influência do número de pontos visíveis no tempo de execução, o observador foi posicionado em diferentes alturas acima do terreno (50 e 100 metros). Além disso, em todos os testes foi considerado que $h_O = h_T$, ou seja, os possíveis alvos estão posicionados à mesma altura que o observador. Os tempos de execução do método *TiledViewshed* foram comparados com os tempos do método *EMViewshed*, uma vez que este mostrou-se mais eficiente que os demais métodos encontrados em literatura [Andrade et al. 2011].

As Tabelas 1 e 2 mostram os tempos médios de execução, em segundos, necessários para processar cada um dos terrenos, utilizando o método *EMViewshed* (EMVS) e o método *TiledViewshed* considerando diferentes tamanhos de blocos (100^2, 250^2, 500^2, 1000^2, 2500^2 e 5000^2), sendo que a Tabela 1 refere-se aos testes com $h_O = h_T = 50$, enquanto a Tabela 2 refere-se aos testes com $h_O = h_T = 100$. A linha *Dim. Bl.* indica o tamanhos dos blocos utilizados, e a linha *# Bl.* indica o número máximo de blocos de cada tamanho que podem ser armazenados nos 800MB disponíveis na memória interna, ou seja, o tamanho da *cache* utilizada pela *TiledMatrix*.

Tamanho do terreno # células)	TiledViewshed						EMVS
	Dim. Bl.	100^2	250^2	500^2	1000^2	2500^2	5000^2	
	# Bl.	27962	4473	1118	279	44	11	
10000^2		58	57	57	58	63	60	**37**
20000^2		443	305	304	298	300	307	**220**
30000^2		1395	828	752	**729**	735	752	1465
40000^2		1772	1229	**1179**	1184	1185	1223	4709
50000^2		7717	2833	2417	2334	**2309**	2338	8988

Tabela 1. Tempos médios de execução, em segundos, para os métodos *EMViewshed* (EMVS) e *TiledViewshed*, considerando diferentes tamanhos de blocos e terrenos e altura de 50 metros. A linha *Dim. Bl.* indica as dimensões dos blocos utilizados, e a linha *# Bl.* indica o número máximo de blocos que podem ser armazenados na memória interna.

Tamanho do terreno # células)	TiledViewshed						EMVS
	Dim. Bl.	100^2	250^2	500^2	1000^2	2500^2	5000^2	
	# Bl.	27962	4473	1118	279	44	11	
10000^2		64	64	63	65	65	65	**37**
20000^2		470	333	319	311	312	318	**234**
30000^2		1753	891	831	803	**784**	806	1502
40000^2		2148	1407	1303	1293	**1256**	1293	5530
50000^2		7570	2868	2437	2406	**2346**	2364	9839

Tabela 2. Tempos médios de execução, em segundos, para os métodos *EMViewshed* (EMVS) e *TiledViewshed*, considerando diferentes tamanhos de blocos e terrenos e altura de 100 metros. A linha *Dim. Bl.* indica as dimensões dos blocos utilizados, e a linha *# Bl.* indica o número máximo de blocos que podem ser armazenados na memória interna.

Para cada tamanho de terreno, o tempo médio destacado em negrito indica o método que apresentou o melhor desempenho. Os resultados mostram que, para os terrenos de dimensões 10000^2 e 20000^2, o método *EMViewshed* apresentou desempenho melhor do que o *TiledViewshed*, independentemente do tamanho dos blocos. Já para os terrenos de tamanhos 30000^2, 40000^2 e 50000^2, o método *TiledViewshed* mostrou-se mais rápido do que o *EMViewshed* para praticamente todos os testes realizados, com exceção do que utilizou blocos de tamanho 100^2 para processar o terreno de dimensões 30000^2 com altura de 100 metros.

É possível perceber que o tamanho dos blocos afeta diretamente o desempenho da estrutura *TiledMatrix*. Enquanto os testes com blocos de tamanho 100^2 apresentaram os piores tempos, os testes com blocos de tamanho 500^2, 1000^2, e 2500^2 apresentaram os melhores tempos entre os tamanhos testados. Os gráficos da Figura 4 mostram os tempos de execução do *EMViewshed* e do *TiledViewshed* utilizando estes 3 tamanhos de blocos.

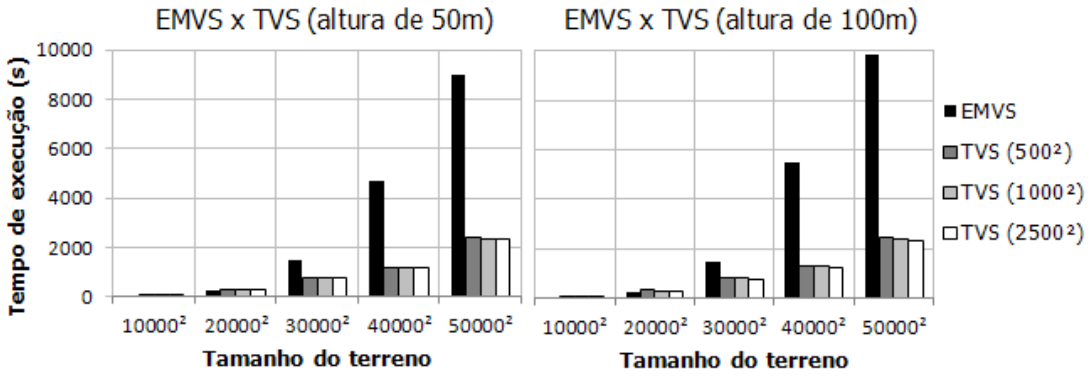

Figura 4. Tempos de execução do *EMViewshed* (EMVS) e do *TiledViewshed* (TVS) utilizando os 3 tamanhos de blocos com melhores desempenhos e alturas de 50 e 100 metros.

Com relação à variação nas alturas dos observadores e alvos, pode-se perceber que o número de observadores visíveis afeta o tempo de execução de ambos os métodos, uma vez que os tempos médios para os testes com altura de 100m são maiores do que os com altura de 50m. No entanto, este aumento foi consideravelmente maior para o método *EMViewshed* (11,1% em média) do que para o método *TiledViewshed* (4,7% em média).

6. Conclusões e trabalhos futuros

Neste trabalho foi apresentado o algoritmo *TiledViewshed* para cálculo de *viewshed* em terrenos armazenados em memória externa que mostrou-se até 4,4 vezes mais rápido do que o *EMViewshed*, proposto por Andrade et al. [Andrade et al. 2011], que era até então o método mais eficiente encontrado em literatura.

Além disso, a estratégia utilizada no método apresentado pode ser adaptada para outras aplicações que utilizem matrizes grandes armazenadas em memória externa. Como trabalhos futuros, propõe-se a utilização da classe *TiledMatrix* em outras aplicações com este perfil, assim como um estudo para avaliar a influência do tamanho dos blocos no desempenho do algoritmo e desenvolver uma estratégia para determinar automaticamente o tamanho mais adequado.

7. Agradecimento

Este trabalho foi parcialmente financiado pela CAPES, pela FAPEMIG e pelo CNPq.

Referências

Aggarwal, A. and Vitter, J. S. (1988). The input/output complexity of sorting and related problems. *Communications of the ACM*, 9:1116–1127.

Andrade, M. V. A., Magalhães, S. V. G., Magalhães, M. A., Franklin, W. R., and Cutler, B. M. (2011). Efficient viewshed computation on terrain in external memory. *GeoInformatica*, pages 381 – 397.

Ben-Moshe, B., Mitchell, J. S. B., Katz, M. J., and Nir, Y. (2002). Visibility preserving terrain simplification - an experimental study. In *Proceedings of ACM Symposium of Computational Geometry*, pages 303–311, Barcelona, Spain.

Bespamyatnikh, S., Chen, Z., Wang, K., and Zhu, B. (2001). On the planar two-watchtower problem. In *In 7th International Computing and Combinatorics Conference*, pages 121–130.

Bresenham, J. (1965). An incremental algorithm for digital plotting. *IBM Systems Journal*.

Dementiev, R., Kettner, L., and Sanders, P. (2005). Stxxl : Standard template library for xxl data sets. Technical report, Fakultat fur Informatik, Universitat Karlsruhe. http://stxxl.sourceforge.net/ (acessado em 05/03/2012).

Felgueiras, C. A. (2001). Modelagem numérica de terreno. In In G. Câmara, C. Davis, A. M. V. M., editor, *Introdução à Ciência da Geoinformação*, volume 1. INPE.

Franklin, W. R. and Ray, C. (1994). Higher isn't necessarily better - visibility algorithms and experiments. In *6th Symposium on Spatial Data Handling*, Edinburgh, Scotland.

Haverkort, H.,], L., and Zhuang, Y. (2007). Computing visibility on terrains in external memory. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments / Workshop on Analytic Algorithms and Combinatorics (ALENEX/ANALCO)*.

Laurini, R. and Thompsom, D. (1992). *Fundamentals of Spatial Information Systems*. Academic Press.

Magalhães, S. V. G., Andrade, M. V. A., Ferreira, C. R., Pena, G. C., Luange, T. G., and Pompermayer, A. M. (2012). Uma biblioteca para o gerenciamento de grandes matrizes em memória externa. Technical report, Departamento de Informática, Universidade Federal de Viçosa.

Magalhães, S. V. G., Andrade, M. V. A., and Franklin, W. R. (2011). Multiple observer siting in huge terrains stored in external memory. *International Journal of Computer Information Systems and Industrial Management Applications*, pages 143–149.

Patterson, D. A. and Hennessy, J. L. (2008). *Computer Organization and Design, Fourth Edition, Fourth Edition: The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition.

Rabus, B., Eineder, M., Roth, A., and Bamler, R. (2003). *The Shuttle Radar Topography Mission (SRTM)*. http://www2.jpl.nasa.gov/srtm/. Acessado em março de 2012.

Ray, C. K. (1994). *Representing Visibility for Siting Problems*. Phd thesis, Rensselaer Polytechnic Institute.

van Kreveld, M. (1996). Variations on sweep algorithms: efficient computation of extended viewsheds and class intervals. In *Symposium on Spatial Data Handling*, pages 15–27.