# Um Algoritmo Eficiente para Cálculo de Viewshed em Memória Externa

**Salles V. G. Magalhães**[1], **Marcus V. A. Andrade**[1], **Mirella A. Magalhães**[1]

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

{smagalhaes,marcus,mirella}@dpi.ufv.br

***Abstract.*** *As more detailed terrain data has become available, many terrain applications have processed large geographic areas at higher resolutions. The massive data processing involved in such applications has presented significant challenges to GIS systems and demands algorithms that are optimized for both data movement and computation. One of these applications is the viewshed computation, that is, to determine all visible points from a given point $p$. In this paper, we present an efficient algorithm to compute the viewshed on terrains stored in external memory. As shown in the results, our algorithm is more efficient than other algorithms described in the literature.*

***Resumo.*** *Com a maior disponibilidade de dados detalhados de terrenos, muitas aplicações precisam processar grandes áreas geográficas em alta resolução. O processamento massivo de dados envolvido em tais aplicações criou grandes desafios para sistemas de SIG e demanda algoritmos otimizados tanto para processamento interno quanto para movimento de dados. Uma dessas aplicações é o cálculo do viewshed, que consiste em obter o conjunto de pontos visíveis a partir de um ponto $p$. Nesse artigo, apresentaremos um algoritmo eficiente para calcular o viewshed de terrenos armazenados em memória externa. Como mostram os resultados, o algoritmo proposto é mais eficiente do que os algoritmos conhecidos descritos em literatura.*

## 1. Introdução

A modelagem de terrenos é uma área importante em aplicações de SIG e um problema interessante nessa área é a determinação de todos os pontos que podem ser vistos a partir de um dado ponto (o observador); a região formada pelos pontos visíveis é chamada de *viewshed* [Floriani and Magillo 2003, Franklin and Ray 1994, Schwartz and Pedrini 2001]. Esse problema é largamente estudado em muitas aplicações, como determinar o número mínimo de torres de celular necessárias para cobrir uma região [Ben-Moshe et al. 2007, Bespamyatnikh et al. 2001], otimizar o número e a posição de guardas para cobrir uma região [Franklin and Vogt 2006], etc.

Os recentes avanços tecnológicos em obtenção de dados (como o LIDAR e IFSAR) têm produzido um enorme volume informação sobre a superfície terrestre. Por exemplo, um terreno de $100km \times 100km$ amostrado com resolução de $1m$ resulta em $10^{10}$ pontos. A maioria dos computadores não pode armazenar e processar esse enorme volume de dados internamente e, portanto, os algoritmos precisam processar esses dados em memória externa, normalmente discos. Como o tempo necessário para acessar

e transferir dados em memória externa é muito maior do que o processamento interno, esses algoritmos devem tentar minimizar o acesso à memória externa.

Mais especificamente, algoritmos que processam dados em memória externa devem ser projetados (e analisados) considerando um modelo computacional que avalia a complexidade dos algoritmos baseada em operações de transferência de dados em vez de operações de processamento interno. Um desses modelos, proposto por Aggarwal e Vitter [Aggarwal and Vitter 1988], mede a complexidade dos algoritmos com base no número de operações de E/S ( entrada/saída) executadas.

Nesse trabalho, apresentaremos um algoritmo eficiente para calcular o viewshed de um ponto em terrenos armazenados em memória externa, esse algoritmo permite a manipulação de terrenos grandes (com 6GB ou mais). Ao comparar nosso algoritmo com o proposto por Haverkort et al. [Haverkort et al. 2007], podemos dizer que o nosso é muito mais fácil de implementar e é aproximadamente 6 vezes mais rápido. Esse trabalho foi desenvolvido pelo aluno de graduação Salles V. G. de Magalhães, pela aluna de mestrado Mirella A. de Magalhães sob a orientação do Prof. Marcus V. A. Andrade sendo que o aluno de gradução foi o responsável pelo projeto de boa parte do algoritmo, por sua implementação e testes.

## 2. Trabalhos relacionados

Geralmente, um terreno pode ser representado por uma malha triangular irregular (*triangulated irregular network (TIN)*) ou por um modelo digital de elevação raster (*DEM - Raster Digital Elevation Model*) [Floriani et al. 1999] . Uma TIN é uma representação vetorial de uma superfície feita a partir de vértices distribuídos irregularmente com três coordenadas dimensionais que estão conectados e posicionados em uma rede de triângulos não sobrepostos. Um DEM é um arquivo digital ou uma matriz que contém elevações de pontos posicionados em intervalos regularmente espaçados. Devido à sua simplicidade, nesse trabalho, consideraremos terrenos representados por DEM.

A visibilidade em terrenos representados por DEM tem sido largamente estudada em muitas áreas distintas. Por exemplo,o trabalho de Stewart [Stewart 1998] utiliza visibilidade em problemas de posicionamento de torres de transmissão de rádio. Kreveld [van Kreveld 1996] propõe uma abordagem do tipo sweep-line para calcular o viewshed com complexidade de tempo $O(nlogn)$ em uma matriz de tamanho $\sqrt{n} \times \sqrt{n}$. Em [Franklin 2002, Franklin and Ray 1994], Franklin e Ray descrevem estudos experimentais para implementações eficientes de cálculo de visibilidade. Para uma referência sobre algoritmos de visibilidade, veja [Floriani and Magillo 2003].

Recentemente, Haverkort et al. [Haverkort et al. 2007] apresentaram uma adaptação do método de Kreveld para calcular o viewshed em terrenos armazenados em memória externa. A complexidade (E/S) desse algoritmo é $O(sort(n))$, onde $n$ é o número de pontos no terreno e $sort(n)$ é a complexidade de se ordenar $n$ elementos em memória externa.

O algoritmo descrito nesse artigo, no pior caso, também tem complexidade $O(sort(n))$, mas ele é executado mais rapidamente já que usa uma estratégia mais eficiente.

## 3. Viewshed

A maioria dos problemas de SIG relacionados à visibilidade envolve o cálculo do viewshed e, em geral, são problemas de otimização como o posicionamento ótimo de recursos, a minimização do posicionamento de guardas, planejamento de caminhos, etc.

Os problemas de visibilidade podem ser classificados em duas categorias principais: determinação de visibilidade e cálculo de estruturas de visibilidade. A determinação de visibilidade consiste em checar se um dado ponto é visível ou não a partir de um outro ponto situado no terreno. Isso pode ser determinado assumindo que um ponto $q$ é visível a partir de outro ponto $p$ se, e somente se, o segmento de reta $\overline{pq}$, denominado linha de visão, está estritamente acima do terreno (exceto nos pontos $p$ e $q$). Veja a figura 1

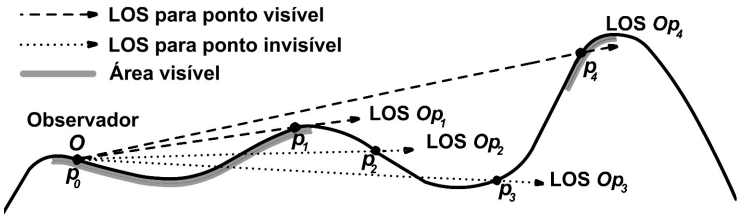



**Figura 1. Visibilidade de pontos: $p_1$ e $p_4$ são visíveis a partir de $p_0$; $p_2$ e $p_3$ não são visíveis a partir de $p_0$.**

O cálculo de estruturas de visibilidade consiste em determinar propriedades dos terrenos como o horizonte, o viewshed, etc. Formalmente, o viewshed de um ponto $p$ com raio de interesse $r$ em um terreno $T$ pode ser definido como:

$$viewshed(p, r) = \{q \in T | \text{ distancia}(p, q) \leq r \text{ e } q \text{ é visível a partir de } p\}$$

O raio de interesse é usado para restringir o viewshed para uma vizinhança do observador. Assim, os possíveis pontos visíveis estão restritos a um círculo centrado em $p$ e com raio $r$. A menos que explicitamente afirmado o contrário, quando o raio de interesse for definido, usaremos $viewshed(p)$ para se referir a $viewshed(p, r)$.

Normalmente, o viewshed é representado por uma matriz de lado $2r$. Cada célula armazena $1$ ou $0$ para indicar se essa célula é visível ou não, respectivamente.

## 4. Algoritmos eficientes para E/S

Durante o processamento de grande volume de dados, a transferência de dados entre a memória interna rápida e o armazenamento externo lento (como os discos) freqüentemente se torna o gargalo do processamento. Portanto, o projeto (e análise) de algoritmos usados para processar esses dados precisa ser feito sob um modelo computacional que avalia as operações de entrada e saída. Um modelo freqüentemente usado 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 número de operações de E/S executadas determina o desempenho do algoritmo.

A complexidade de um algoritmo é definida com base na complexidade de problemas fundamentais como varredura e ordenação de $N$ elementos contíguos armazenados em memória externa. Tais complexidades relacionadas a operações de E/S são:

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

onde $M$ é o tamanho da memória interna.

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

## 5. Cálculo de Viewshed em Memória Externa (EMVS)

Nosso algoritmo, chamado *External Memory Viewshed* (EMVS), é baseado no método proposto por Franklin e Ray [Franklin and Ray 1994] que calcula o viewshed de um ponto em um terreno representado por uma matriz em memória interna. Uma breve descrição desse método é dada a seguir.

### 5.1. O método de Franklin e Ray

Dado um terreno representado por uma matriz de elevação $T$ de dimensões $n \times n$ e dado um ponto $p$ em $T$, o algoritmo calcula o viewshed de $p$ restrito a um círculo de raio $r$ (o raio de interesse) centrado em $p$. O algoritmo realiza uma varredura radial do círculo usando um raio, ou linha de visão (*LOS – line of sight*), iniciando em $p$ e caminha por cada *LOS* para determinar se as posições (as células) na *LOS* são visíveis a partir de $p$ ou não. Uma posição $q$ do terreno é visível a partir de $p$ se o segmento $\overline{pq}$ não intercepta nenhuma posição cuja altura seja maior do que a de $q$.

Para simplificar a varredura do círculo, o algoritmo utiliza um quadrado (chamado de quadrado de varredura) que circunscreve o círculo, tem lado $2r$ e é centrado em $p$. As linhas de visão são definidas conectando $p$ a cada ponto da borda desse quadrado. Inicialmente, todas as células que estão dentro do quadrado são consideradas não visíveis e, para cada linha de visão $l$, o algoritmo começa em $p$ definindo a inclinação de $l$ como $-\infty$ (isto é, um número negativo grande). Então, essa inclinação é atualizada cada vez que uma célula que gera uma *LOS* de inclinação maior é alcançada. Isso é, supondo que a altura atual de $l$ em um ponto $c$ é $h$ e a altura de $c$ é $h'$, se $h < h'$ então a célula $c$ é marcada como visível e a inclinação de $l$ é atualizada para que a altura de $l$ seja $h'$. Por outro lado, se $h \geq h'$, o status da célula e a inclinação de $l$ são preservados.

Como as células são acessadas em uma ordem definida pela varredura radial, o uso desse algoritmo para grandes terrenos exigiria um acesso aleatório ao arquivo e a execução seria ineficiente. A adaptação descrita abaixo evita esse tipo de acesso.

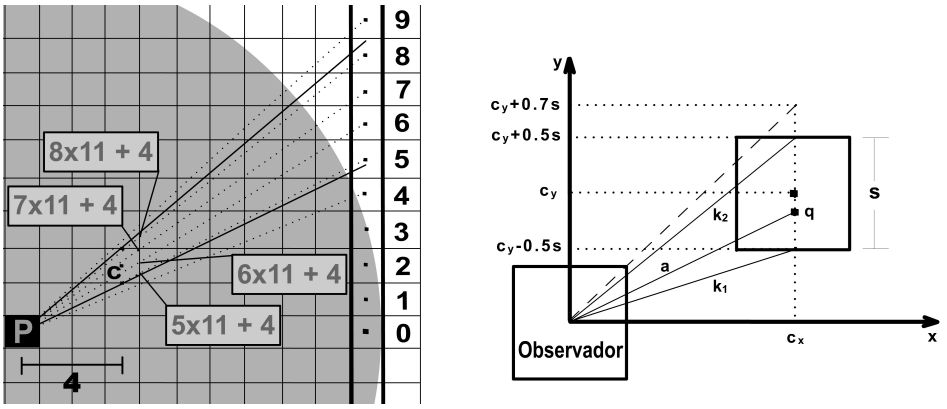
### 5.2. O algoritmo EMVS

A idéia básica consiste em gerar uma lista com os pontos do terreno ordenados pela ordem de processamento. Para calcular o viewshed, o algoritmo percorre a lista evitando

acessar o arquivo aleatoriamente. Essa lista é armazenada em memória externa e é gerenciada por uma biblioteca especial chamada STXXL (*Standard Template Library for Extra Large Data Sets – Biblioteca Padrão de Gabaritos para Grandes Conjuntos de Dados*) [Dementiev et al. 2005].

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

Para determinar os índices, as linhas de visão (que iniciam no observador em $p$) são numeradas em sentido anti-horário. Assim, as células são numeradas de forma crescente em cada linha de visão; quando o fim de uma linha de visão é alcançado, a numeração continua a partir do observador seguindo a próxima linha de visão. Uma mesma célula pode receber vários índices (e possuir várias cópias em $L$) já que ela pode ser interceptada por múltiplas linhas de visão.




(a) (b)

**Figura 2. (a) Projeção de LOS e cálculo dos índices de uma célula e (b) linhas de visão interceptando uma célula**

Para criar a lista $L$, o algoritmo lê as células do terreno seqüencialmente a partir do arquivo externo e, para cada célula $c$, determina o número de todas as linhas de visão que interceptam aquela célula.

Como as células não são "unidimensionais", podemos determinar as células interceptadas pela linha de visão usando um processo similar à rasterização de linhas [Bresenham 1965]. Isto é, seja $s$ o lado de cada célula (quadrada) e suponha que a referência da célula é o seu centro. Seja $a$ a linha de visão cuja inclinação é $\alpha \ : \ 0 < \alpha \leq 45°$ [1]. Então, dada uma célula $c = (c_x, c_y)$, veja a figura 2(b), a linha de visão $a$ "intercepta"a célula $c$ entre os pontos $(c_x, c_y - 0.5s)$ e $(c_x, c_y + 0.5s)$; mais precisamente, dado $(q_x; q_y) = a \cap c_x$, $a$ intercepta $c$ se, e somente se, $c_y - 0.5s \leq q_y < c_y + 0.5s$ . Nesse caso, a linha pontilhada na figura 2(b) interceptará a célula que está acima de $c$.

Assim, todas as linhas de visão que interceptam $c$ são as que estão entre as duas linhas que conectam o observador aos pontos $(c_x, c_y - 0.5s)$ e $(c_x, c_y + 0.5s)$ - Figura 2(b). Sejam $k_1$ e $k_2$ os números dessas duas linhas, respectivamente. Os números de todas as

[1] Para $45° < \alpha \leq 90°$, use uma idéia similar trocando $x$ e $y$.

linhas que interceptam $c$ são dados pelas células cujo centro estejam entre $k_1$ e $k_2$ - veja a figura 2(a). Nessa figura, por exemplo, a célula $c$ é interceptada pelas linhas 5, 6, 7 e 8.

Dada uma célula $c$, seja $r$ o número de uma linha de visão que intercepta $c$. O índice $i$ de $c$ associada a $r$ é dado pela formula $i = r * n + d$, onde $n$ é o número de células em cada raio e $d$ é a distância entre os pontos $c$ e $p$ - veja a figura 2(a).

Após isso, a lista $L$ é ordenada usando os índices dos elementos como chave de comparação e, então, as células são processadas na ordem dada pela lista ordenada. Esse processamento é feito por um algoritmo similar ao de memória interna que, nesse caso, lê os dados de elevação diretamente de $L$. Além disso, esse algoritmo utiliza uma outra lista $L'$ onde as células visíveis são inseridas. Após o processamento de todas as células, $L'$ é ordenada lexicograficamente por $x$ e $y$ e as células visíveis são armazenadas em um arquivo onde as posições visíveis são indicadas por $1$ e as não visíveis por $0$.

Finalmente, um ganho de eficiência é alcançado armazenando-se uma parte da matriz do terreno em memória interna. As células que estão em memória interna não são inseridas em $L$ e $L'$ e, quando uma célula precisa ser processada, o algoritmo verifica se essa célula está em memória interna. Se estiver, ela é processada normalmente; caso contrário, ela é lida de $L$.

## 6. Complexidade do Algoritmo

Seja $T$ o terreno representado por uma matriz de elevação de dimensões $n \times n$. Então, $T$ possui $n^2$ células. Além disso, seja $p$ a posição do observador e seja $r$ o raio de interesse. Como descrito na seção 5.2, o algoritmo considera as células que estão dentro do quadrado de dimensões $2r \times 2r$ centrado em $p$. Assumindo que o lado de cada célula é $s$, haverá, no máximo, $\frac{2r}{s}$ células em cada lado do quadrado o que implica que há $\frac{8r}{s}$ células no perímetro do quadrado. Seja $K = \frac{r}{s}$. Então, o algoritmo traça $8K$ linhas de visão e, já que cada linha de visão tem $K$ células, a lista $L$ possui, no pior caso, $O(K^2)$ elementos.

No primeiro passo, o algoritmo realiza $\frac{n^2}{B}$ operações de E/S para ler as células do terreno e criar a lista $L$. Depois, a lista com $O(K^2)$ elementos é ordenada e então é varrida para se calcular a visibilidade das células. Assim, o número total de operações de E/S é:

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

Normalmente, o raio de interesse $r$ é (muito) menor do que $n$ (o lado da matriz do terreno), assim $K$ é menor do que $n$ e então, geralmente, o número de operações de E/S é dada por $O(\frac{n^2}{B}) = O(scan(n^2))$. Mas no pior (não usual) caso, se o raio de interesse é grande a ponto de cobrir quase todo o terreno, o número de operações de E/S é $O\left(\frac{K^2}{B}\log_{\left(\frac{M}{B}\right)}\left(\frac{K^2}{B}\right)\right)$ = $O(sort(K^2))$.

O algoritmo também usa uma lista externa adicional $L'$ para manter as células visíveis e essa lista precisa ser ordenada. Como o tamanho dessa lista é (muito) menor do que o tamanho de $L$, o número de operações de E/S executadas nesse passo não muda a complexidade do algoritmo.

## 7. Resultados

O algoritmo EMVS foi implementado em C++, usando g++ 4.1.1, e os testes foram executados em um PC Pentium com 2.8 GHZ, 1 GB de RAM, hd serial ATA de 80 GB e 7200 RPM e sistema operacional Mandriva Linux. Para uma avaliação melhor das operações de E/S, consideramos duas configurações: uma usando toda a memória RAM (1 GB) e permitindo o uso de 800 MB para armazenar os dados internamente e outra usando 256MB e permitindo o uso de 200 MB para dados.

Os terrenos usados nos testes foram adquiridos na página do SRTM [The Shuttle Radar Topography Mission (SRTM) 2007], possuem menos de 1.5 % de dados inválidos, e pertencem a três regiões distintas dos EUA. Nos testes, usamos várias porções desses terrenos com tamanhos diferentes[2]. Para cada tamanho de terreno, foi determinada a média do tempo necessário para processar dados das três regiões. Resultados mais detalhados podem ser obtidos em http://www.dpi.ufv.br/projetos/EMViewshed.

A tabela 1 mostra o tempo médio de execução do EMVS nesses terrenos. Em todos os testes, consideramos o pior caso, isso é, usamos um raio de interesse grande a ponto de cobrir todo terreno. Para avaliar a influência do número de pontos visíveis (a coluna *# Pts. Vis.*) no tempo de execução, o observador foi posicionado em diferentes elevações acima do terreno (1,50,100,1000 e 10000 metros). Embora 1000 e 10000 metros não sejam elevações muito comuns em aplicações práticas, usamos esses valores para confirmar a escalabilidade do algoritmo.

A figura 3 resume os tempos médios de processamento interno e externo. Como esperado, o tempo de processamento externo é muito maior em terrenos maiores do que a memória interna; veja os gráficos (c), (d) e, principalmente, (b) onde o tempo de processamento externo é maior (menor) do que o de processamento interno quando se usa 256 MB (1 GB). Essa diferença em (b) ocorre porque o terreno de 1122 MB pode ser armazenado (quase) completamente nos 1 GB de memória, necessitando, assim, de menos operações de E/S.

Além disso, o desempenho do EMVS foi comparado com o desempenho do algoritmo de Haverkort et al. (IO_VS). A figura 4 resume essa comparação considerando os resultados disponíveis em [Haverkort et al. 2007] e assumindo que o observador esteja 1 metro acima do terreno. Os resultados do EMVS foram obtidos a partir dos tempos dados na tabela 1. Note que o EMVS é mais de 6 vezes mais rápido do que o IO_VS e, vale a pena mencionar que os testes do IO_VS foram executados em um computador Power Macintosh G5 dual 2.5 GHz, 1 GB de RAM e hd de 80 GB com 7200 RPM que é significantemente mais rápido do que a máquina utilizada em nosso testes. Assim, podemos supor que nosso algoritmo é ainda mais rápido do que aquele. Finalmente, como vantagem adicional, o EMVS é mais simples de implementar do que o IO_VS.

Na nossa opnião, as razões principais para o melhor desempenho do EMVS são: ele usa uma estrutura de dados simples (uma lista ordenada) e, após a ordenação (externa) da lista, não são feitas atualizações (inserção e/ou remoção) na lista. Por outro lado, o algoritmo IO_VS utiliza estruturas de dados mais elaboradas que são manipuladas por

[2]Para comparar a eficiência de nosso algoritmo com o de Haverkort et al., usamos terrenos com tamanhos similares àqueles usados por eles.

**Tabela 1. Tempo médio de execução do EMVS utilizando 1 GB e 256 MB de RAM e altura do observador acima do terreno variável (gerando viewshed com número diferente de pontos visíveis - mostrado na coluna # Pts Vis.).**

| Tamanho | Altura | # Pts | Tempo (s) 1 GB | | Tempo (s) 256 MB | |
|---|---|---|---|---|---|---|
| Terreno | Obs. | Vis. | Externo | Total | Externo | Total |
| $9029^2$ 311 MB | 1 | $4.9 \times 10^5$ | 19 | 46 | 21 | 49 |
| | 50 | $3.5 \times 10^7$ | 17 | 50 | 22 | 56 |
| | $10^2$ | $4.6 \times 10^7$ | 18 | 50 | 21 | 58 |
| | $10^3$ | $6.1 \times 10^7$ | 18 | 57 | 22 | 60 |
| | $10^4$ | $6.3 \times 10^7$ | 19 | 58 | 21 | 61 |
| $17150^2$ 1122 MB | 1 | $2.2 \times 10^5$ | 69 | 209 | 523 | 709 |
| | 50 | $1.0 \times 10^8$ | 67 | 223 | 527 | 737 |
| | $10^2$ | $1.0 \times 10^8$ | 67 | 233 | 528 | 744 |
| | $10^3$ | $2.0 \times 10^8$ | 67 | 240 | 525 | 752 |
| | $10^4$ | $2.2 \times 10^8$ | 68 | 247 | 520 | 746 |
| $33433^2$ 4264 MB | 1 | $4.0 \times 10^5$ | 1746 | 2627 | 4143 | 5099 |
| | 50 | $7.9 \times 10^7$ | 1759 | 2662 | 4647 | 5636 |
| | $10^2$ | $2.9 \times 10^8$ | 1774 | 2725 | 5067 | 6124 |
| | $10^3$ | $7.8 \times 10^8$ | 1752 | 2796 | 5315 | 6513 |
| | $10^4$ | $8.6 \times 10^8$ | 1758 | 2816 | 5401 | 6626 |
| $40000^2$ 6103 MB | 1 | $2.1 \times 10^6$ | 3402 | 4860 | 6752 | 8143 |
| | 50 | $8.3 \times 10^7$ | 3786 | 5330 | 6978 | 8418 |
| | $10^2$ | $2.8 \times 10^8$ | 4415 | 6161 | 7489 | 8996 |
| | $10^3$ | $9.1 \times 10^8$ | 4462 | 6312 | 7937 | 9672 |
| | $10^4$ | $1.1 \times 10^9$ | 4820 | 6734 | 8242 | 10066 |

processos recursivos que envolvem operações de busca e atualização.

## 8. Conclusão

Apresentamos um algoritmo muito eficiente para calcular o viewshed de um ponto em terrenos grandes armazenados em memória externa. Como os testes mostraram, nosso algoritmo é mais de 6 vezes mais rápido do que o desenvolvido por Haverkort et al [Haverkort et al. 2007] e, também, pode processar terrenos muito grandes (nós o testamos em terrenos de 6.1 GB). Além disso, o algoritmo é simples de entender e implementar. A implementação do algoritmo está disponível em http://www.dpi.ufv.br/ marcus/TerrainModeling/EMViewshed/EMVS.tgz como um código open source distribuído sob licença Creative Common GNU GPL [Commons 2007].

## 9. Agradecimento

Este trabalho foi parcialmente financiado pelo CNPq e pela FAPEMIG.

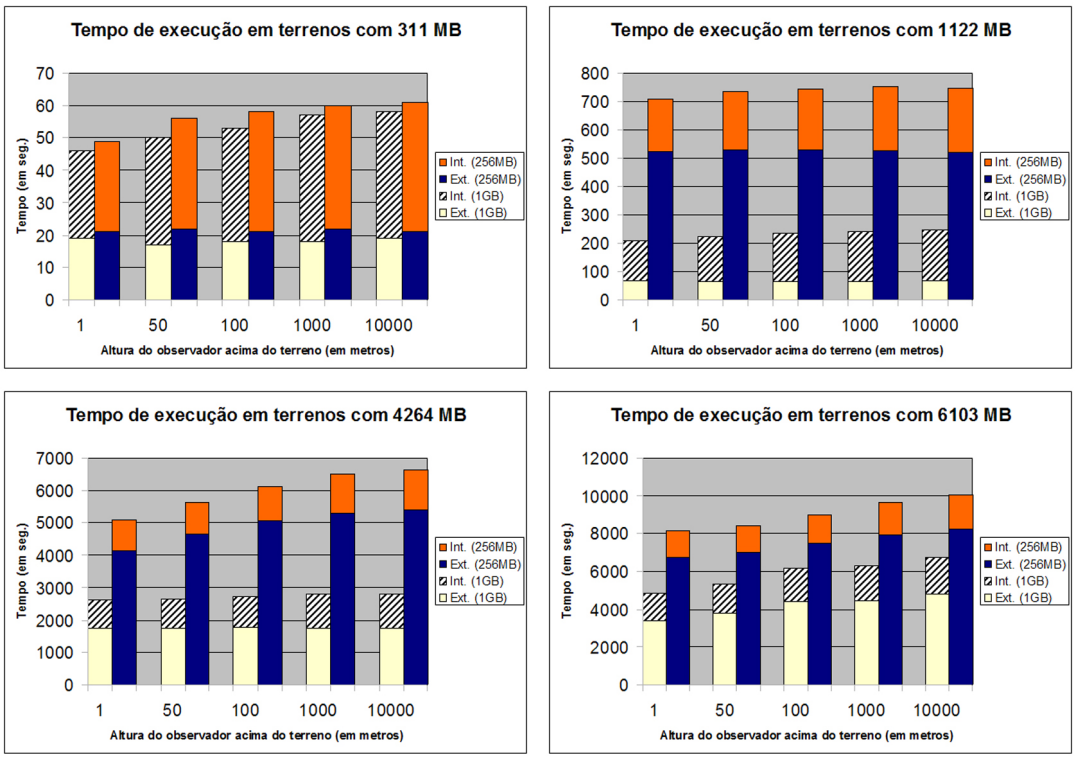
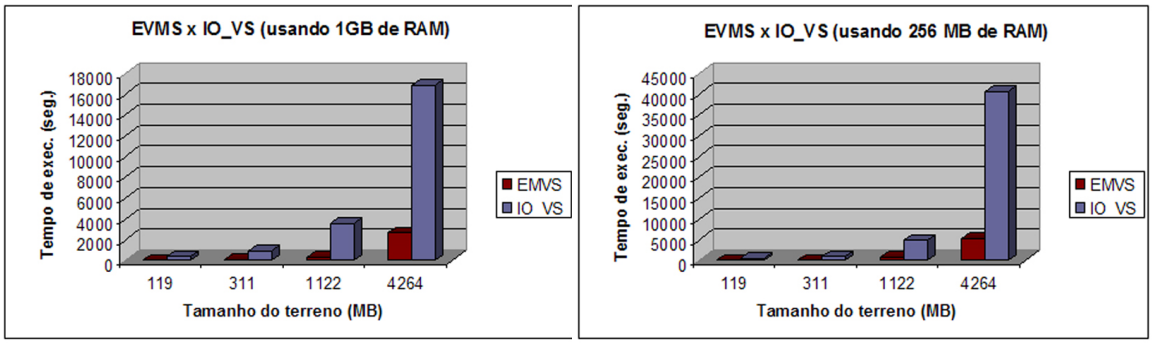



**Figura 3. O tempo de processamento interno e externo usando 256 MB e 1 GB de RAM em terrenos com tamanhos diferentes.**




| Tamanho do Terreno | Tempo de exec. (s) (1GB de RAM) | |
|---|---|---|
| | EMVS | IO_VS |
| 119 MB | 18 | 353 |
| 311 MB | 46 | 865 |
| 1122 MB | 209 | 3546 |
| 4264 MB | 2627 | 16895 |

(a)

| Tamanho do Terreno | Tempo de exec. (s) (256MB de RAM | |
|---|---|---|
| | EMVS | IO_VS |
| 119 MB | 22 | 364 |
| 311 MB | 49 | 916 |
| 1122 MB | 709 | 4831 |
| 4264 MB | 5099 | 40734 |

(b)

**Figura 4. Comparação do tempo de execução do EMVS e do IO_VS usando: (a) 1GB e (b) 256 MB de RAM.**

## Referências

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

Ben-Moshe, B., Ben-Shimol, Y., and Y. Ben-Yehezkel, A. Dvir, M. S. (2007). Automated antenna positioning algorithms for wireless fixed-access networks. *Journal of Heuristics*, 13(3):243–263.

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*.

Commons, C. (2007). http://creativecommons.org/license/cc-gpl (acessado em Março de 2008).

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 Março de 2008).

Floriani, L. D. and Magillo, P. (2003). Algorithms for visibility computation on terrains: a survey. *Environment and Planning B - Planning and Design*, 30:709–728.

Floriani, L. D., Puppo, E., and Magillo, P. (1999). Applications of computational geometry to geographic information systems. In J. R. Sack, J. U., editor, *Handbook of Computational Geometry*, pages 303–311. Elsevier Science.

Franklin, W. R. (2002). Siting observers on terrain. In Springer-Verlag, editor, *In D. Richardson and P. van Oosterom editors, Advances in Spatial Data Handling: 10th International Symposium on Spatial Data Handling*, pages 109–120.

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.

Franklin, W. R. and Vogt, C. (2006). Tradeoffs when multiple observer siting on large terrain cells. In *12th International Symposium on Spatial Data Handling*.

Haverkort, H., Toma, 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)*.

The Shuttle Radar Topography Mission (SRTM) (2007). http://www2.jpl.nasa.gov/srtm/ (acessado em Março de 2008).

Schwartz, W. R. and Pedrini, H. (2001). Determinação de mapas de visibilidade em modelos digitais de terrenos. In *II Colóquio Brasileiro de Ciências Geodésicas (CBCG'2001)*, pages 44–45, Curitiba-PR, Brazil.

Stewart, A. J. (1998). Fast horizon computation at all points of a terrain with visibility and shading applications. In *IEEE Trans. Visualization Computer Graphics*, pages 82 – 93.

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.