On IO-Efficient Viewshed Algorithms and Their Accuracy

Herman Haverkort\textsuperscript{1} Laura Toma\textsuperscript{2} Bob PoFang Wei\textsuperscript{2}

\textsuperscript{1}Eindhoven University of Technology, the Netherlands
\textsuperscript{2}Bowdoin College, USA

ACM SIGSPATIAL GIS 2013
November 2013, Orlando, FL
The problem

- Terrain $T$ and viewpoint $v$
- Compute viewshed of $v$: set of points in $T$ visible from $v$

Applications:
- path planning, navigation, placement of radar towers, etc
Terrains

- Most commonly represented as grids of elevation values
Large amounts of data have become available

- NASA SRTM: 30m resolution data for entire globe (∼10TB)
- LIDAR data: sub-meter resolution
- E.g.: Washington State, 1m grid: ∼689GB

Traditional (internal memory) algorithms

- Assume all data fits in memory

Big data ➔ IO-bottleneck

- Main memory too small to hold all data
- Data (partially) on disk
- Hard disks are ∼1,000,000 slower than memory
IO complexity: the number of IOs
Goal: minimize (CPU- and) IO-complexity

Basic building blocks and bounds:
- \(\text{scan}(n) = \Theta\left(\frac{n}{B}\right)\) IOs
- \(\text{sort}(n) = \Theta\left(\frac{N}{B} \log_{M/B}\right)\frac{n}{B}\) IOs
Visibility on Grids

Need to interpolate elevation along the line-of-sight (LOS) $vp$
Basic viewshed algorithm

Input: elevation grid
Output: visibility grid, each point marked visible/invisible

For each p in grid
- compute intersections between vp and grid lines
- if all these points are below vp then p is visible
Basic viewshed algorithm

Input: elevation grid
Output: visibility grid, each point marked visible/invisible

- For each $p$ in grid
  - compute intersections between $vp$ and grid lines
  - if all these points are below $vp$ then $p$ is visible
Basic viewshed algorithm

Input: elevation grid
Output: visibility grid, each point marked visible/invisible

For each \( p \) in grid
- compute intersections between \( vp \) and grid lines
- if all these points are below \( vp \) then \( p \) is visible

Assume grid of \( n \) points \((\sqrt{n} \times \sqrt{n})\)
Running time: \( O(n\sqrt{n}) \)
Related work

In memory:
- R3 algorithm: \(O(n\sqrt{n})\) time [Franklin & Ray ’94]
  - produces “exact” viewshed
  - slow
- XDraw, R2: \(O(n)\) time [Franklin & Ray ’94]:
  - approximations to R3
- Radial sweep: \(O(n \lg n)\) time [Van Kreveld ’96]
  - nearest neighbor interpolation

IO-efficient:
- Ferreira et al 2012: \(O(\text{sort}(n))\) IOs based on R2
- Fishman et al 2009: \(O(\text{sort}(n))\) IOs based on Van Kreveld
Accuracy!!

with ioradial from Fishman et al 2009

with GRASS
Our results

- An improved and IO-efficient version of the “exact” algorithm
  - gridlines vs. layers model
  - iterative vs. divide-and-conquer

- Horizons on grids have worst-case complexity $O(n)$
  - improves on $O(n^{\alpha}(n))$

- Running time and accuracy analysis
  - accuracy metric
  - compare with Van Kreveld’s model, R2, r.los in GRASS
Gridlines vs Layers Model

Grid model

Layers model:
- consider a subset of the obstacles in the grid model
- larger viewshed
Iterative viewshed (Layers model)

Traverse the grid in layers
Maintain the horizon of the region traversed so far
Algorithm \texttt{Vis-ITER}:

create grid $V$ and initialize as all invisible

$H \leftarrow \emptyset$

for each layer $l$ in the grid do

\hspace{1cm} //traverse layer $l$ in ccw order

\hspace{1cm} for $r \leftarrow 0$ to $-l$ //first octant

\hspace{2cm} get elevation $Z_{rl}$ of $p(r, l)$

\hspace{2cm} determine if $Z_{rl}$ is above $H$

\hspace{2cm} if visible, set value $V_{rl}$ in $V$ as visible

\hspace{2cm} $h \leftarrow$ projection of $p(r - 1, l)p(r, l)$

\hspace{2cm} merge $h$ into horizon $H$
Iterative viewshed (Layers model)

**Algorithm** Vis-ITER:

create grid $V$ and initialize as all invisible  
$H \leftarrow \emptyset$

for each layer $l$ in the grid do

  //traverse layer $l$ in ccw order

  for $r \leftarrow 0$ to $-l$ //first octant

    get elevation $Z_{rl}$ of $p(r, l)$

    determine if $Z_{rl}$ is above $H$

    if visible, set value $V_{rl}$ in $V$ as visible

    $h \leftarrow$ projection of $p(r-1, l)p(r, l)$

    merge $h$ into horizon $H$

Denote $H_{1,i}$: horizon of points in layers $L_1 \cup ... \cup L_i$

After finishing $L_i$, $H$ is $H_{1,i}$:
Iterative viewshed (Layers model)

Vis-Iter runs in
\[ O(n + |H_{1,1}| + |H_{1,2}| + |H_{1,3}| + \ldots) = O(n + \sum_{i=1}^{\infty} |H_{1,i}|) \text{ time} \]
Split the elevation grid into bands around $v$ and compute visibility one band at a time.
IO-efficient approach

1. **Build elevation bands** $E_k$
   for each $(i, j)$ in grid:
   - $k \leftarrow$ band containing $(i, j)$
   - append $Z_{ij}$ to $E_k$

2. **Compute visibility in each band**
   for $k = 1$ to $N_{bands}$:
   - load $E_k$ into memory
   - traverse it one layer at a time, writing visibility values to $V_k$

3. **Collect visibility bands** $V_k$
   for each $(i, j)$ in grid:
   - $k \leftarrow$ band containing $(i, j)$
   - read $V_{ij}$ from $V_k$ and write it to $V$

$E_k$ and $V_k$ are stored in row-major order $\Rightarrow$ Step 1 writes $E_k$
sequentially and Step 3 reads $V_k$ sequentially.
IO-efficient approach

1. **Build elevation bands** $E_k$ for each $(i, j)$ in grid:
   - $k \leftarrow$ band containing $(i, j)$
   - append $Z_{ij}$ to $E_k$

2. **Compute visibility in each band** for $k = 1$ to $N_{bands}$:
   - load $E_k$ into memory
   - traverse it one layer at a time, writing visibility values to $V_k$

3. **Collect visibility bands** $V_k$ for each $(i, j)$ in grid:
   - $k \leftarrow$ band containing $(i, j)$
   - read $V_{ij}$ from $V_k$ and write it to $V$

If $n = O(M^2 / B)$: Step 1 and Step 3 take one sequential pass.
IO-efficient approach

1. **Build elevation bands** $E_k$ for each $(i,j)$ in grid:
   - $k \leftarrow$ band containing $(i,j)$
   - append $Z_{ij}$ to $E_k$

2. **Compute visibility in each band** for $k = 1$ to $N_{bands}$:
   - load $E_k$ into memory
   - traverse it one layer at a time, writing visibility values to $V_k$

3. **Collect visibility bands** $V_k$ for each $(i,j)$ in grid:
   - $k \leftarrow$ band containing $(i,j)$
   - read $V_{ij}$ from $V_k$ and write it to $V$

Step 2 takes $\text{scan}(n) + \text{scan}(|H_{1,1}| + |H_{1,2}| + \ldots))$ IOs.
Notation:
- $H_{1,l}$: horizon of points in the first $l$ layers
- $H_{tot} = |H_{1,1}| + |H_{1,2}| + ...$

In general, we have:
- $O(n \log n + H_{tot})$ time and $O(\text{sort}(n) + \text{scan}(H_{tot}))$ IOs provided that $n < cM^2$ for a sufficiently small $c$.

In practice, $H_{1,l}$ fit in memory and $n = O(M^2/B)$:
- $O(\text{scan}(n))$ IOs (3 passes over the grid)
Idea: Instead of merging the layers one at a time, use divide-and-conquer.

Algorithm $\text{DAC-BAND}(E_k, V_k, i, j)$:

- **if** $i == j$
  - $h \leftarrow \text{compute-layer-horizon}(i)$
  - return $h$
- **else**
  - $m \leftarrow \text{middle layer between } i \text{ and } j$
  - $h_1 \leftarrow \text{DAC-BAND}(E_k, V_k, i, m)$
  - $h_2 \leftarrow \text{DAC-BAND}(E_k, V_k, m+1, j)$
  - mark invisible all points in $L_{m+1,j}$ that fall below $h_1$
  - $h \leftarrow \text{merge}(h_1, h_2)$
  - return $h$
Notation:

- $H_{1,i}^B$: horizon of points in the first $i$ bands.
- $H_{tot}^B = |H_{1,1}^B| + H_{1,2}^B + \ldots$.

In general,

- $O(n \lg n + H_{tot}^B)$ time and $O(\text{sort}(n) + \text{scan}(H_{tot}^B))$ IOs provided that $n < cM^2$ for a sufficiently small $c$.

In practice, $H_{1,i}^B$ fit in memory and $n = O(M^2/B)$:

- $O(\text{scan}(n))$ IOs (3 passes over the grid)
Iterative vs. Divide-and-Conquer

Worst-case complexity of horizon: $O(n^\alpha(n))$

Theorem

Let $S$ be a set of line segments in the plane, such that the widths of the segments of $S$ do not differ in length by more than a factor $d$, then the upper envelope of $S$ has complexity $O(dn)$.

$\Rightarrow$ Worst-case complexity of horizon: $O(n)$

In the worst-case: $|H_{tot}| = O(n\sqrt{n})$, $|H_{tot}^B| = O(n^2/M)$

In the worst case, handling horizons dominate and DAC < ITER

If horizons are small: ITER may be faster
Gridlines model

\[ H(L_i) + H(X_i) \]
Experimental analysis

Platform:
- HP 220 blade servers, Intel 2.8GHz
- 512MB RAM
- 5400rpm SATA hard drive

Datasets:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>cols × rows</th>
<th>GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cumberlands</td>
<td>8 704 × 7 673</td>
<td>0.25</td>
</tr>
<tr>
<td>Washington</td>
<td>31 866 × 33 454</td>
<td>3.97</td>
</tr>
<tr>
<td>SRTM1-region03</td>
<td>50 401 × 43 201</td>
<td>8.11</td>
</tr>
<tr>
<td>SRTM1-region04</td>
<td>82 801 × 36 001</td>
<td>11.10</td>
</tr>
<tr>
<td>SRTM1-region06</td>
<td>68 401 × 111 601</td>
<td>28.44</td>
</tr>
</tbody>
</table>

On IO-Efficient Viewshed Algorithms and Their Accuracy
ITER is consistently 10-20% faster than DAC
Horizon Size

$H_i$: horizon of layer $i$, $|H_i| = O(i)$

$H_{1,i}$: horizon of first $i$ layers, $|H_{1,i}| = O(i^2)$

$H_{1,i}$ stays very small, way below its worst-case bound. All SRTM datasets have $|H_{1,\sqrt{n}}|$ between 132 and 32,689.
For a dataset and a viewpoint, denote $H_{1,O(\sqrt{n})}$ its final horizon.

Worst-case bound: $O(n)$

Stays below $O(\sqrt{n})$
Lots of variation (due to position of viewpoint, shape of grid)
Running time

- Build-Bands, Collect-Bands run in one pass over the data.
- 75% of running time spent in reading or writing bands, 25% in computing visibility.
- Compared to previous work:
  - approx. as fast as IO-CENTRIFUGAL in [Fishman et al 2009]
  - approx. 2x faster than IO-RADIAL in [Fishman et al 2009]
  - approx. 2.5x slower than TILEDVS in [Ferreira et al 2012]

BUT, IO-CENTRIFUGAL, IO-RADIAL and TILEDVS compute different viewshed approximations.
Ideally, need ground truth

Given viewshed algorithms A (reference) and B:
- Pick a sample of viewpoints $X$
- For each viewpoint $v \in X$
  - compute viewshed($v$) with A and B
  - compute $f_v$ (number of false visibles) and $f_i$ (number of false invisibles) of B wrt A, as percentage of viewshed size
- average over $X$

Select $X$ from the set of points with topological significance (ridges and channels)
Reference algorithm: r.los in GRASS

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>fv</th>
<th>fi</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITER-LAYERS</td>
<td>.2%</td>
<td>.4%</td>
</tr>
<tr>
<td>IO-RADIAL</td>
<td>53%</td>
<td>14%</td>
</tr>
<tr>
<td>IO-CENTRIFUGAL</td>
<td>8%</td>
<td>33%</td>
</tr>
<tr>
<td>TILEDVS</td>
<td>7%</td>
<td>7%</td>
</tr>
</tbody>
</table>

ITER-LAYERS vs ITER-GRIDLINES:

\[fv = 0, \, fi = .2%\]
Conclusions

- Scalable algorithms for computing the viewshed that fully exploit the resolution of the data
- Layers model is simpler, faster and computes practically the same viewshed as the gridlines model
- Horizons on grids are small, far below worst-case bound ⇒ horizon-based approaches promising
- Accuracy important when comparing viewshed algorithms

Thank you!