On IO-Efficient Viewshed Algorithms and Their Accuracy

Herman Haverkort[1] Laura Toma[2] Bob PoFang Wei[2]

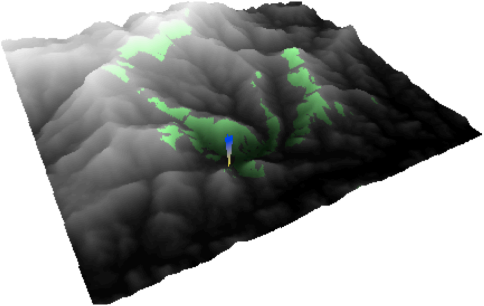
[1]Eindhoven University of Technology, the Netherlands
[2]Bowdoin College, USA

ACM SIGSPATIAL GIS 2013

November 2013, Orlando, FL

The problem

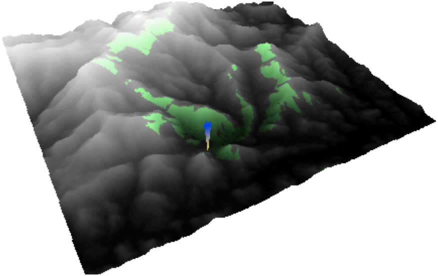
- Terrain T and viewpoint v
- Compute viewshed of v: set of points in T visible from v



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

Terrains

- Most commonly represented as grids of elevation values



20	23	25	26	32	46
21	20	24	28	41	46
24	21	23	31	36	36
23	22	24	27	33	34
32	22	29	30	35	34
29	30	33	34	36	37

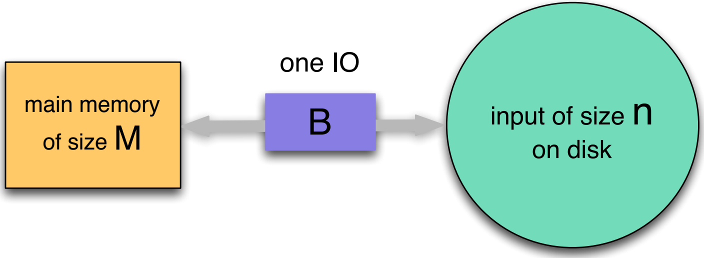
Big Data

- Large amounts of data have become available
  - NASA SRTM: 30m resolution data for entire globe ($\sim$10TB)
  - LIDAR data: sub-meter resolution
  - E.g.: Washington State, 1m grid: $\sim$689GB

- Traditional (internal memory) algorithms
  - Assume all data fits in memory

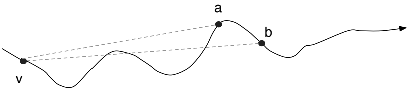
- Big data $\Longrightarrow$ IO-bottleneck
  - Main memory too small to hold all data
  - Data (partially) on disk
  - Hard disks are $\sim$ 1,000,000 slower than memory

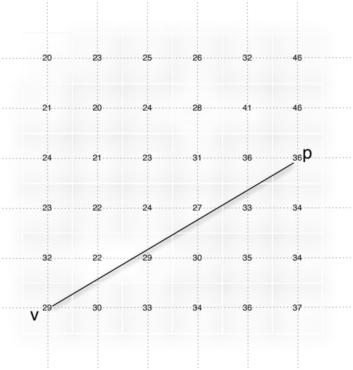
IO Model [AV'88]



- IO complexity: the number of IOs
- Goal: minimize (CPU- and) IO-complexity

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

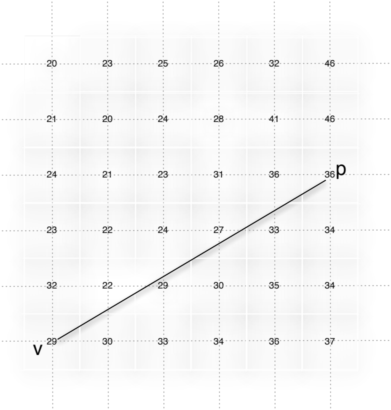




Need to interpolate elevation along the line-of-sight (LOS) *vp*

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

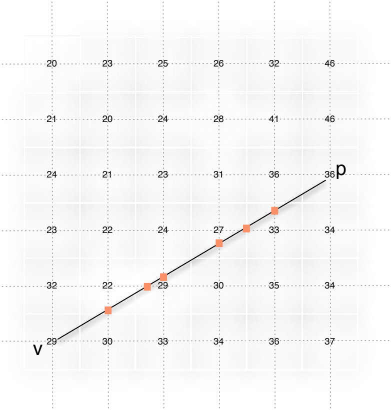
- For each p in grid
  - compute intersections between *vp* and grid lines
  - if all these points are below *vp* then *p* is visible



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

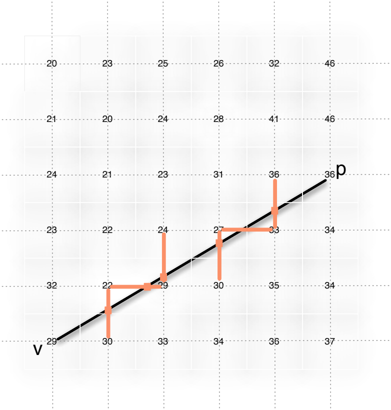


Basic viewshed algorithm

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 ($\sqrt{n}$ x $\sqrt{n}$)
Running time: $O(n\sqrt{n})$



Related work

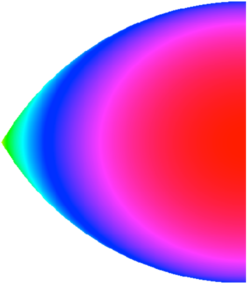
In memory:

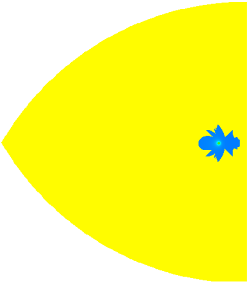
- R3 algorithm: $O(n\sqrt{n})$ time [Franklin & Ray '94]
  - produces "exact" viewshed
  - slow
- XDraw, R2: $O(n)$ time [Franklin & Ray '94]:
  - approximations to R3
- Radial sweep: $O(n \lg n)$ time [Van Kreveld '96]
  - nearest neighbor interpolation

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

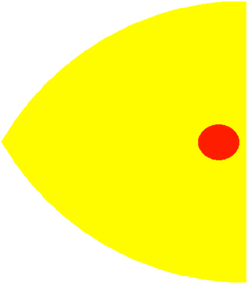
Accuracy!!





with ioradial from

Fishman et al 2009



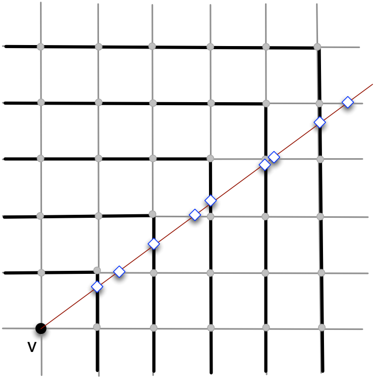
with GRASS

Our results

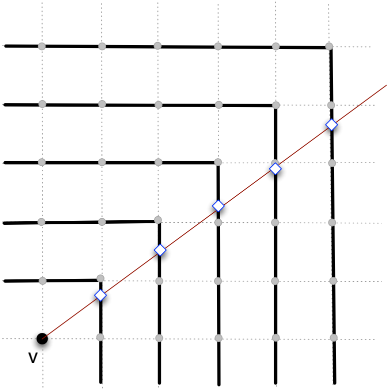
- An improved and IO-efficient version of the “exact” algorithm
  - gridlines vs. layers model
  - iterative vs. divide-and-conquer
- Horizons on grids have worst-case complexity $O(n)$
  - improves on $O(n\alpha(n))$
- Running time and accuracy analysis
  - accuracy metric
  - compare with Van Kreveld’s model, R2, r.los in GRASS

Gridlines vs Layers Model

Grid model



Layers model



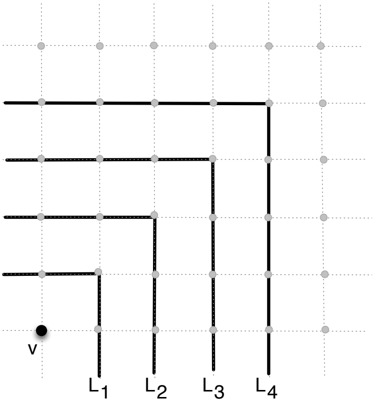
Layers model:

- 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

Layers model



Iterative viewshed (Layers model)

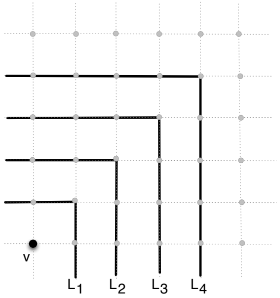
Algorithm VIS-ITER:

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



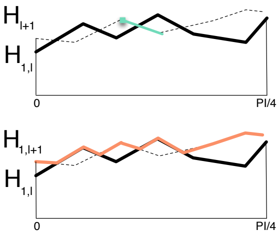
Algorithm VIS-ITER:

create grid V and initialize as all invisible

$H \leftarrow \emptyset$

for each layer l in the grid **do**

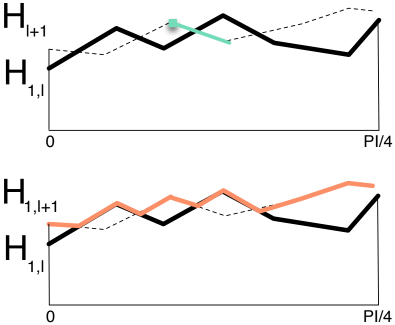
- //traverse layer l in ccw order
- **for** $r \leftarrow 0$ to $-l$ //first octant
  - get elevation Z_{rl} of $p(r,l)$
  - determine if Z_{rl} is above H
  - if visible, set value V_{rl} in V as visible
  - $h \leftarrow$ projection of $p(r-1,l)p(r,l)$
  - merge h into horizon H



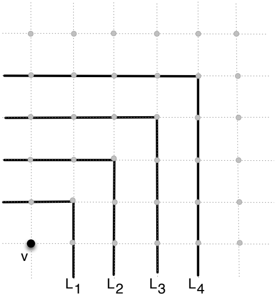
Denote $H_{1,i}$: horizon of points in layers $L_1 \cup ... \cup L_i$

After finishing L_i, H is $H_{1,i}$:

Iterative viewshed (Layers model)



Layers model

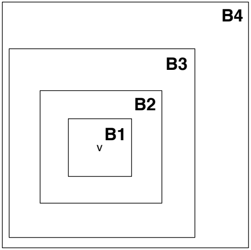


VIS-ITER runs in

$O(n + |H_{1,1}| + |H_{1,2}| + |H_{1,3}| + ...) = O(n + \sum_{i=1} |H_{1,i}|)$ time

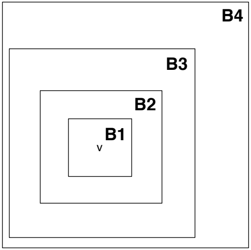
IO-efficient approach

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

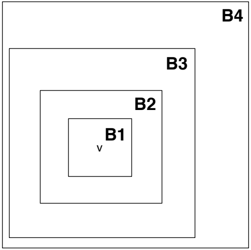


E_k and V_k are stored in row-major order $\Rightarrow$ Step 1 writes E_k sequentially and Step 3 reads V_k sequentially.

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

Size of band $\Theta(M)$.

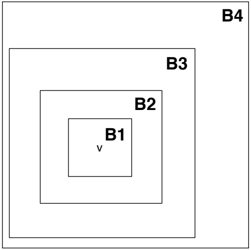


If $n = O(M^2/B)$: Step 1 and Step 3 take one sequential pass.

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

Size of band $\Theta(M)$.



Step 2 takes scan(n) + scan$(|H_{1,1}| + |H_{1,2}| + \ldots)$ IOs.

IO-efficient approach

Notation:

- $H_{1,l}$: horizon of points in the first l layers
- $H_{tot} = |H_{1,1}| + |H_{1,2}| + \dots$

In general, we have:

- $O(n \lg n + H_{tot})$ 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)

Viewshed with Divide-and-Conquer (Layers model)

Idea: Instead of merging the layers one at a time, use divide-and-conquer.

Algorithm DAC-BAND(E_k, V_k, i, j):

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

Viewshed with Divide-and-Conquer (Layers model)

Notation:

- $H^B_{1,i}$: horizon of points in the first i bands.
- $H^B_{tot} = |H^B_{1,1}| + H^B_{1,2}| + \dots$.

In general,

- $O(n \lg n + H^B_{tot})$ time and $O(\text{sort}(n) + \text{scan}(H^B_{tot}))$ IOs

provided that $n < cM^2$ for a sufficiently small c.

In practice, $H^B_{1,l}$ 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: $O(n\alpha(n))$

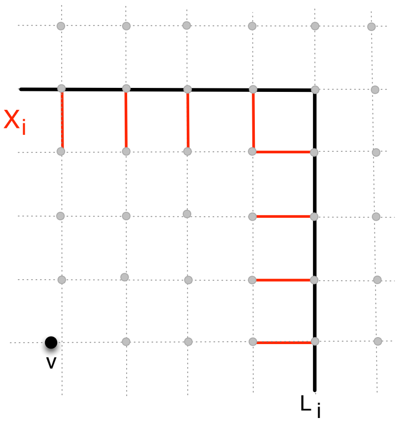
Theorem

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

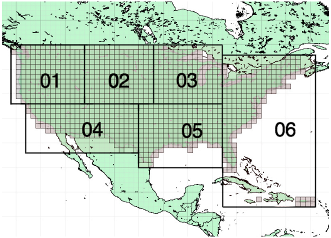


$$H(L_i) + H(X_i)$$

Experimental analysis

Platform:

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

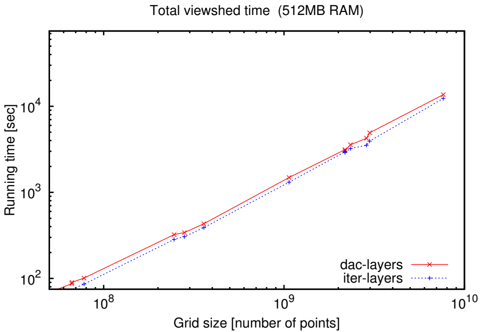


Datasets:

Dataset	Size		
	cols $\times$	rows	GB
Cumberlands	8 704 $\times$	7 673	0.25
Washington	31 866 $\times$	33 454	3.97
SRTM1-region03	50 401 $\times$	43 201	8.11
SRTM1-region04	82 801 $\times$	36 001	11.10
SRTM1-region06	68 401 $\times$	111 601	28.44

Running Time

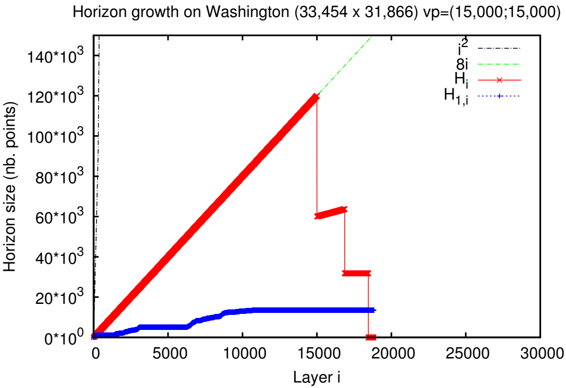
ITER is consistently 10-20% faster than DAC



Horizon Size

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

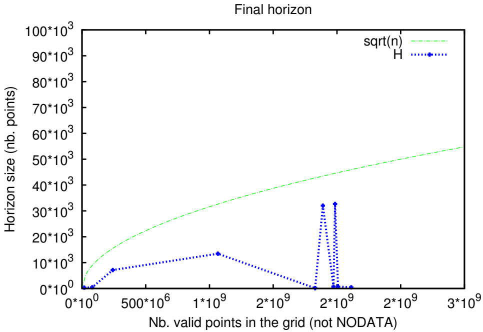
$H_{1,i}$: horizon of first i layers,
$|H_{1,i}| = O(i^2)$



$H_{1,i}$ stays very small, way below its worst-case bound
All SRTM datasets have $|H_{1,\sqrt{n}}|$ between 132 and 32,689

Horizon Size

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)

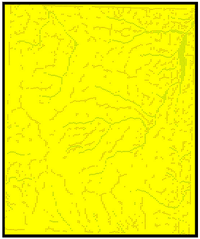
Running time

- 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

Accuracy

- 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 *fv* (number of false visibles) and *fi* (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)

Accuracy

Reference algorithm: r.los in GRASS

	fv	fi
ITER-LAYERS	.2%	.4%
IO-RADIAL	53%	14%
IO-CENTRIFUGAL	8%	33%
TILEDVS	7%	7%

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 $\Rightarrow$ horizon-based approaches promising
- Accuracy important when comparing viewshed algorithms

Thank you!