A Parallel Sweep Line Algorithm for Visibility Computation

Chaulio R. Ferreira
Marcus V. A. Andrade
Salles V. G. Magalhães
W. Randolph Franklin
Guilherme C. Pena
Summary

• Introduction
  – Visibility computation
  – Van Kreveld’s algorithm

• Our algorithm

• Results

• Conclusions and Future Work
Introduction
Visibility Computation

• Determining whether some targets are visible from a given observer.
Visibility Computation

• Viewshed: set of all terrain points that are visible from a given observer.
Visibility Computation

• Some applications:
  – Computing the coverage of an observation tower.
Visibility Computation

- Some applications:
  - Computing the coverage of an observation tower.
  - Siting multiple observers to cover a given terrain.
Visibility Computation

- Digital Elevation Matrix

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Van Kreveld’s Algorithm

• Most viewshed algorithms rotate a line-of-sight around the observer and, at each iteration, compute which cells are intercepted by the line.
Van Kreveld’s Algorithm

• Van Kreveld’s algorithm maintains a data structure, named *active*, that stores which cells are currently being intercepted by the line.
Van Kreveld’s Algorithm

- active is actually implemented as a balanced binary tree, keyed by the cells’ distance from the observer.
Van Kreveld’s Algorithm

• Thus, all query and update operations in active are performed in $\Theta(\log n)$. 
Van Kreveld’s Algorithm

• To rotate the line-of-sight and keep active updated, Van Kreveld defines three types of events for every cell:

  • ENTER EVENT: when the cell starts being intercepted by the line;
Van Kreveld’s Algorithm

• To rotate the line-of-sight and keep active updated, Kreveld defines three types of events for every cell:

  • ENTER EVENT: when the cell starts being intercepted by the line;
  • CENTER EVENT: when the line passes over the cell’s center;
Van Kreveld’s Algorithm

• To rotate the line-of-sight and keep active updated, Kreveld defines three types of events for every cell:

  • ENTER EVENT: when the cell starts being intercepted by the line;
  • CENTER EVENT: when the line passes over the cell’s center;
  • EXIT EVENT: when the cell stops being intercepted by the line.
Van Kreveld’s Algorithm

• Each event is identified by its type and its azimuth angle.

• All events for all cells are stored in one single list, named Events.

• Events is then sorted according to the events azimuth angles.

• Thus, the events are stored in Events in the same order they would happen while rotating the line-of-sight.
Van Kreveld’s Algorithm

• The sorted list $Events$ is swept and, for each event, an action is taken:

  • If it is an ENTER event, the cell is inserted into $active$;

  • If it is a CENTER event, $active$ is searched to check whether the cell is visible;

  • If it is an EXIT event, the cell is removed from $active$. 
Van Kreveld’s Algorithm

• Summarizing: Van Kreveld’s algorithm consists of three major parts:

  • 1) Compute *Events*;

  • 2) Sort *Events*;

  • 3) Sweep *Events* and keep *active* updated while computing the viewshed.
Our algorithm
Our algorithm

• Recently, some viewshed algorithms have been adapted for parallel computing.

• For instance, [Zhao et al., 2013] have implemented the R3 algorithm using GPU computing.

• However, we have not found any previous parallel implementation of Van Kreveld’s algorithm.
Our algorithm

• In fact, [Zhao et al., 2013] stated that:

  “A high degree of sequential dependencies in Van Kreveld’s algorithm makes it less suitable to exploit parallelism”.

• That is mainly because, for processing an event, we should have already processed all earlier events.
Our algorithm
Our algorithm

• To design a parallel algorithm, we subdivided the terrain into sectors and created one Events list for each one of them:
Our algorithm

• This sector, for instance, will have its own *active* structure and its own *Events* list, which contains the events for theses cells:
Our algorithm

- The hardest task here is to efficiently find out which cells are inside this sector.
Our algorithm

• To do that, we traced rays from $O$ to all cells in the sector perimeter:
Our algorithm

- By rasterizing these rays, we can find out the cells and create the *Events* list.
Our algorithm

• Now, each sector has its own *Events* list. We just need to sort these lists and sweep them processing the events.

• Since all *Events* lists are independent, each sector may be processed independently.

• Thus, we can assign one sector for each core, and compute the viewshed in parallel.
Results
Results

• We implemented our algorithm in C++ using OpenMP.

• Our experimental platform:
  • Dual Intel Xeon E5-2687 3.1GHz 8 core
  • Ubuntu 12.04 LTS, Linux 3.5 Kernel.
Results

- We ran it on six terrains with different dimensions and limiting the number of parallel threads (2, 4, 8 and 16).

- We compared its performance with a serial implementation of Van Kreveld’s algorithm.
Results

- 2 parallel threads: average speedup of 1.9 times
Results

- 4 parallel threads: average speedup of 3.6 times
Results

- 8 parallel threads: average speedup of 6.5 times
Results

- 16 paral. threads: average speedup of 10.7 times
Conclusions and Future Work
Conclusions and Future Work

• We proposed a parallel implementation of Van Kreveld’s algorithm for viewshed computation.

• It uses the shared memory model (OpenMP), which is available in most current computers.

• Using only 4 parallel cores (a very common architecture nowadays), we achieved speedups of up 3.94 times.
Conclusions and Future Work

• Thus, our algorithm might be useful not only for scientific computers, but also for users with regular computers.

• As future work, we propose to implement Van Kreveld’s algorithm using GPU computing.

• Since GPU architectures are much more complex, this will not be a straightforward adaption.
References


Questions? Suggestions?

Acknowledgements: