Slimming Brick Cache Strategies for Seismic Horizon Propagation Algorithms

Jonathan Gallon, Sébastien Guillon, Bruno Jobard, Hélène Barucq, Noomane Keskes

IEEE/EG International Symposium on Volume Graphics (2010)
R. Westermann and G. Kindlmann (Editors)

Slimming Brick Cache Strategies for Seismic Horizon Propagation Algorithms

J. Gallon[1,2,3], S. Guillon[1], B. Jobard[2,3], H. Barucq[3], and N. Keskes[1]

[1]TOTAL, France
[2]LIUPPA, UPPA, France
[3]INRIA-MAGIQUE3D, UPPA, France

Abstract

In this paper, we propose a new bricked cache system suitable for a particular surface propagation algorithm : seismic horizon reconstruction. The application domain of this algorithm is the interpretation of seismic volumes used, for instance, by petroleum companies for oil prospecting. To ensure the optimality of such surface extraction, the algorithm must access randomly into the data volume. This lack of data locality imposes that the volume resides entirely in the main memory to reach decent performances. In case of volumes larger than the memory, we show that using a classical brick cache strategy can also produce good performances until a certain size. As the size of these volumes increases very quickly, and can now reach more than 200GB, we demonstrate that the performances of the classical algorithm are dramatically reduced when processed on standard workstation with a limited size of memory (currently 8GB to 16GB). In order to handle such large volumes, we introduce a new slimming brick cache strategy where bricks size evolves according to processed data : at each step of the algorithm, processed data could be removed from the cache. This new brick format allows to have a larger number of brick loaded in memory. We further improve the releasing mechanism by filling in priority the "holes" that appear in the surface during the propagation process. With this new cache strategy, horizons can be extracted into volumes that are up to 75 times the size of the available cache memory.

We discuss the performances and results of this new approach applied on both synthetic and real data.

1. Introduction

Achieving a good efficiency level for *Surface Propagation Algorithms* on large volume datasets requires special attention to the design of the data access architecture and the memory management. Surface propagation algorithms constitute a particular class of surface extraction algorithms. It mainly differs from the marching algorithm class (like the "marching cube" algorithm [LC87]) by the order the voxels of the volume are accessed during the surface construction process. In the latter, all voxels are successively accessed in contiguous order and a polygonal piece of the surface is created every time a voxel intersects it. When the dataset becomes larger than memory size, a bricking strategy is used to successively load parts of the dataset into memory and free it when the whole brick has been processed. Thus, the volume can be processed one brick at a time before recomposing the whole polygonal surface [DJG*08].

With propagation algorithms, the order of voxel traversal is unpredictable. Starting from an initial seed location, a surface grows from it by testing its similarity with neighboring voxels. With a bricking strategy, a brick may be loaded and released several times since the surface can propagate outside the brick and later propagate back in from an adjacent brick. Therefore, a caching mechanism is required to decrease the penalty of the multiple brick transfers from the disk to memory.

The algorithms for propagating horizon (geological surface) in seismic volumes correspond to a class of these algorithms where voxel traversal order is unpredictable [Kes98, KZRM83]. They start from a seed that marks the chosen horizon and for each propagation step, a voxel is popped out from a priority queue that sorts all the candidate voxels by their degree of correlation with the initial seed. The traversal order is strictly imposed by this correlation criterion and for larger data volume, there is no way to restrict the prop-

agation in a controlled number of bricks. This characteristic should limit efficient horizon propagation to the case where the required bricks all fit into main memory. In this paper, we state an advanced caching mechanism suitable for horizon propagation in very large seismic volume datasets. We show how to manage more bricks than the memory can contain by releasing used parts of the brick data. We further accelerate the rate of the memory purge by noticing that propagating in horizon holes first gives the same quality as globally propagated horizons.

This work has been done to use standard seismic data formats as Inrimage, SEP or Seg-Y, commonly used in pretroleum industry.

In the next section, we present the horizon propagation algorithm. In section 3, we describe a classic bricked data access, which is efficient for horizon propagation until the maximum available memory size is reached. In section 4.1, we show how larger volumes can be handled with the use of a *slimming brick* cache system. To even better manage this cache system, we propose a hole horizon filling optimization in section 4.2. Finally some results are presented and discussed in section 5.

2. Horizon Propagation in Seismic Volumes

In this section, we first state the origin of seismic data and the usefulness of automatic horizon propagation algorithms for the petroleum industry. Some formal definitions are then introduced to better understand the propagation algorithm that we have improved.

2.1. Seismic Volume Data and Interpretation

3D seismic data interpretation is a key step for the petroleum prospecting in understanding local deposit systems and oil reservoir configurations. The seismic data is obtained by recording the multiple reflections of acoustic waves sent towards underground structures (equivalent to a "ground echography"). The studied geographic area is horizontally discretized and a vertical *seismic trace* is then computed for each ground location. Along each trace, high amplitude values characterize interface depths between rock layers of different nature. These traces are used to fill 3D grids with amplitude values, which are called seismic volume data (see figure 1 left).

Extracting the main rock interfaces (also called horizons) is necessary in order to build a geological 3D model of the explored area and to understand reservoir and fluid characteristics. For a long time, this important surface extraction operation was manually achieved by picking on seismic sections sample points belonging to the currently extracted horizon. The horizon was finally obtained by triangulating these relatively sparse sample points (see figure 1 right).

Today, many automatic horizon extraction algorithms

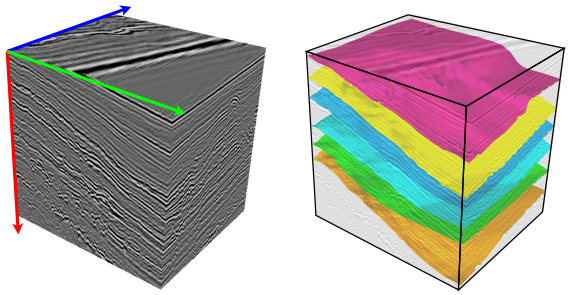

Figure 1: *Example of seismic volume data (left) and six extracted horizons (right)*

have been developed [HH91, FP04, PBVG10]. Most of them require an initial seed location from where a surface grows through "similar" neighboring voxels. The propagation process of these algorithms differs with the kind of similarity criterion they adopt ("pick", "trough", "form", "dip", etc).

The propagation algorithm considered in this paper has been proposed by Keskes et *al.* [KZRM83]. It's still one of the most robust algorithm thanks to its strategy that always propagate the most similar candidate to the initial seed.

This propagation strategy ensures that any point on the final horizon has been found following a *tracking path* that is optimal in regard to the maximum correlation conservation. Unfortunately, the unpredictable paths in the volume implies a total absence of locality in data accesses in the seismic volume.

The two next sections describe the algorithm and show that even if it performs well as an in-core algorithm, it is no longer able to handle the large seismic datasets (hundreds of gigabytes) that have become common in the petroleum industry with good performances.

We first introduce some more formal definitions that will ease understanding later considerations.

2.2. Definitions

A seismic volume V of size $N_x \times N_y \times N_z$ is a function

$$V : [1,N_x] \times [1,N_y] \times [1,N_z] \rightarrow \mathbb{R}$$

$$(x,y,z) \rightarrow V(x,y,z)$$

where $V(x,y,z)$ represents the seismic amplitude at position (x,y,z). For a given x,y, the set of amplitudes in z is called a *seismic trace*. And this volume is store in one file linearly in z,y,x-order, i.e. each trace values are contiguous in the file.

A horizon H is an elevation map defined by

$$H : [1,N_x] \times [1,N_y] \rightarrow \mathbb{R}$$

$$(x,y) \rightarrow H(x,y)$$

where $H(x,y)$ represents the z depth of the horizon at position (x,y).

Given a volume V and a point $p(x,y,z)$, we define a local trace $tr(p)$ around p as a vertical portion of the amplitude values stored in V :

$$tr(p(x,y,z)) = \{V(x,y,z+m), m \in [-M_1, +M_2]\} \quad (1)$$

where M_1 and M_2 define a margin size above and below p. This vertical interval is called the *correlation window*.

The correlation *corr* between two points $p_1(x,y,z)$ and $p_2(x,y,z)$ is defined as a correlation coefficient between the two traces $tr(p_1)$ and $tr(p_2)$ as:

$$corr(p_1,p_2) = \frac{tr^T(p_1).tr(p_2)}{||tr(p_1)||.||tr(p_2)||}$$

Then $corr(p_1,p_2) \in [-1;1]$ and $corr(p_1,p_2)$ equals to 1 when p_1 and p_2 are similar and decreases as p_1 and p_2 differs.

2.3. Horizon Propagation Algorithm

The propagation algorithm is based on a priority queue Q which allows to propagate always in the best voxel candidate. Each element of Q is sorted depending on the correlation factor $corr(s,p')$ between the candidate p' and the initial seed s.

At the beginning, the user places in the volume a seed $s(x_0,y_0,z_0)$ from where the horizon will be extracted. The seed is inserted in the priority queue with a weight $corr(s,s) = 1$ and a horizon H is initialized with $H(x_0,y_0) = z_0$ and undefined values elsewhere.

Then while the queue is not empty, the best point $p(x,y,z)$ is extracted from Q, and its neighbours are explored: for each (4-connected or 8-connected) neighbour $p'(x',y',z')$ in x and y direction where the horizon $H(x',y')$ is undefined, the best $z' \in [z-T_1, z+T_2]$ is searched such that the correlation $C_{max} = corr(s,p'(x',y',z'))$ is maximized. z' is the z_i such that

$$C_{max} = \underset{z_i \in [z-T_1, z+T_2]}{\arg\max} (corr(s,p'(x',y',z_i))) \quad (2)$$

where s is the user's initial seed. If the maximum correlation C_{max} is greater than a given user's threshold, the candidate $p'(x',y',z')$ is added to Q with the weight C_{max} and the horizon is updated accordingly, $H(x',y') = z'$.

To illustrate this algorithm, Figure 2 shows the evolution of surface H during its propagation.The orange area represents the set of already propagated points from the seed s, and the grey area the set of points to be propagated. The border points between orange and grey regions are either validated points (inserted into Q) or rejected points (points with a correlation value lower than the user's threshold). The three points P_1, P_2 and P_3 represent the three next points in the priority queue Q to be processed and the red lines show their propagation path from the seed point s.

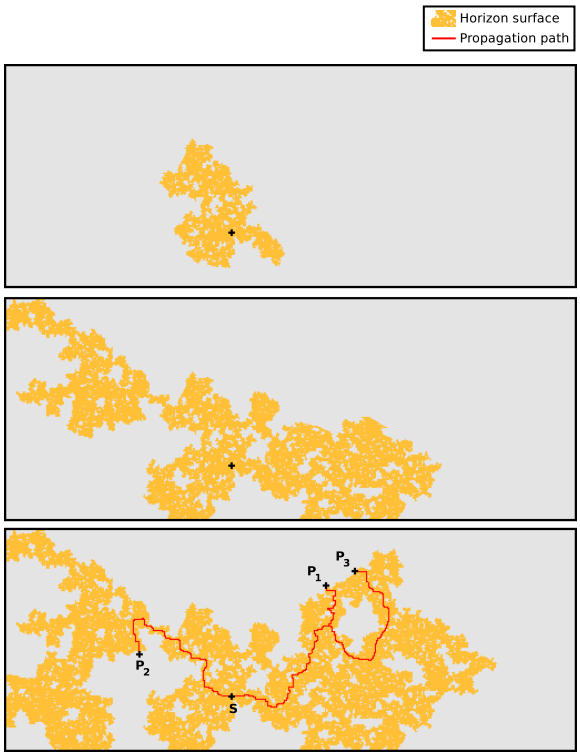

Figure 2: *Example of horizon during propagation which shows all the validated points. P_1, P_2 and P_3 are the three next points in the priority queue Q and s is the seed.*

2.4. Low Data Access Efficiency

The previous section demonstrates that processing successive best candidates from Q, such as P_1, P_2 and P_3 in Figure 2, induces random accesses through the volume to find the future candidates.

With a small enough seismic volume that totally fits into main memory, access penalty due to the lack of data locality is still bearable. But current seismic volumes are much larger than the main memory capacity and this implies direct reads from disk at very different places of the files, which is around 10000 to 100000 times slower than memory accesses. A comparative study on access time of different memory levels is done in [KBKG07].

The next section shows how a classic brick strategy allows to work on a subvolume V' that fits again into main memory. When the subvolume V' itself becomes bigger than memory capacity, we present an innovative slimming brick strategy that pushes further away the size limit of efficiently processable seismic volumes (section 4).

3. Classic Static Brick Strategy

When large datasets have to be processed, they are commonly subdivided into bricks so that one or several of them can entirely be loaded into memory. For example, in [KBKG07] Kohlmann et *al.* such a process is explained for the optimisation of volume rendering in a medical application, and in [GBKG04] Grimm et *al.* describe the advantage of both bricked access and hierarchical caching for accelerate volume raytracing.

In the case of horizon propagation, we know that the extracted surface is an elevation map. Therefore only a limited vertical portion of the data centered around each point of the horizon should be sufficient to compute it. *A posteriori*, this optimal subvolume V' is defined by the M_1 and M_2 margins necessary for computing the correlation (eq. 1) and the additional search margins T_1 and T_2 (eq. 2) to find the best z' value.

Thus, for a propagated horizon H, as illustrated in Figure 3 left, the optimal subvolume V' needed is defined by :

$$V' = \{(x,y,z) \in V \\ |z \in [H(x,y) - M_1 - T_1, H(x,y) + M_2 + T_2]\}$$

with a size $card(V') = N_x \times N_y \times (M_1 + T_1 + M_2 + T_2)$.

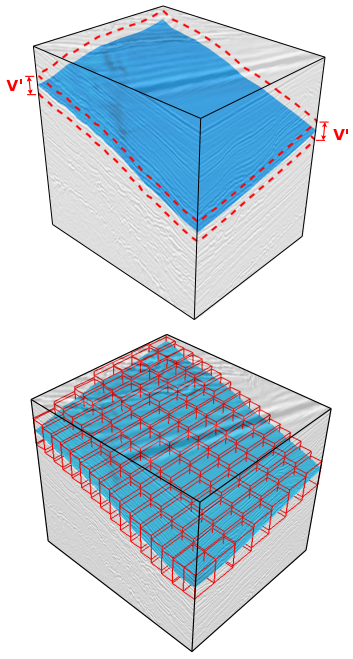

Figure 3: *Volume V' of required data for the propagation. (Top) Optimal volume such as defined in eq. 3, (Bottom) Approximated volume with a set of adjacent bricks.*

In practice, the correlation is computed with vertical margin size of 20 voxels and the search area is of 5 voxels in up and down directions. So the equation becomes: $card(V') \simeq N_x \times N_y \times 30$.

Therefore, a propagation on 100 GB volume for instance ($5000 \times 5000 \times 2000$, stored in 2 bytes) will need to access to only 1.4 GB which could easily be loaded into memory.

Unfortunately, this subvolume V' cannot be determined *a priori* due to the unpredictable aspect of the horizon. However, it can be determined by pieces as the surface propagates: each time the propagation leaves existing bricks, adjacent ones are created, vertically centered around the outgoing voxels (see figure 3 right).

Splitting the horizontal domain $[1,N_x] \times [1,N_y]$ in a set of subrectangles of size K_x, K_y, we define a brick $B_{ij}(z)$ as a part of the volume defined by :

$$B_{ij}(z) = [iK_x,(i+1)K_x[\times[jK_y,(j+1)K_y[\\ \times[z-\frac{K_z}{2},z+\frac{K_z}{2}]$$

with z a vertical position and K_z the vertical size.

The brick extraction costs less than an equivalent number of random readings in the volume. But since it is still a costly operation, the $B_{ij}(z)$ bricks are stored in a two level cache in main memory and on disk. Several brick management policies are possible when dealing with a cache [AZMM04] but we chose to keep the bricks which contain the best points p of the priority queue Q. Thus, we don't remove bricks that have to serve in next iterations.

Figure 4 explains how the cache works.

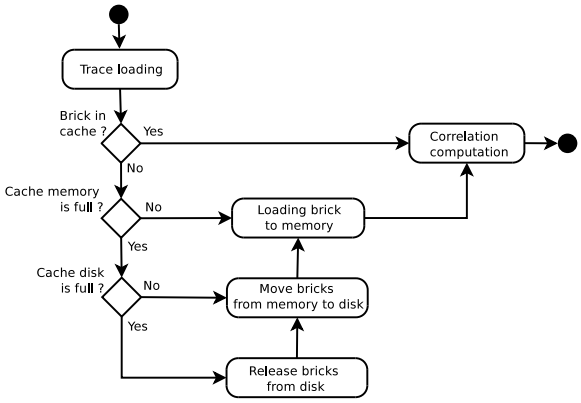

Figure 4: *Classic cache system mechanism. Moved or released bricks are chosen funtion of a management policy.*

Using this brick strategy, the data access time is drastically reduced. As illustrated in Figure 5, displaying the cache status (blue bricks stored on cache disk and green bricks stored on memory cache), processing P_1, P_2 and P_3 now requires local cache accesses instead of a random data access in the global volume. Furthermore, while the cache is big

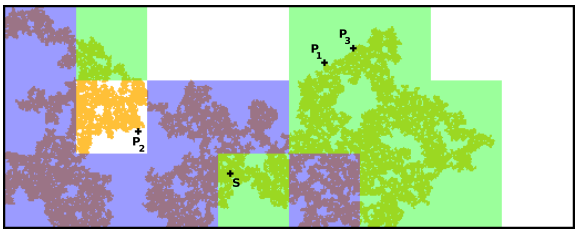

Figure 5: *Example of propagated surface with current loaded bricks (blue bricks are on disk and green ones are in main memory). The brick containing P_2 have been released because the cache was full.*

enough for storing all the $B_{ij}(z)$ bricks, the performance of the algorithm remains very stable.

But as soon as the addressed volume increases and the size of the optimal subvolume V' becomes larger than the cache, the performance decreases very quickly. This behavior is illustrated in Figure 6. It displays the time of propagation for increasing size of required data (V') using a global cache of 8GB (3GB of memory and 5GB of disk). The red curve shows the performance of the bricked access and the black curve, the performance with a direct access to the seismic data.

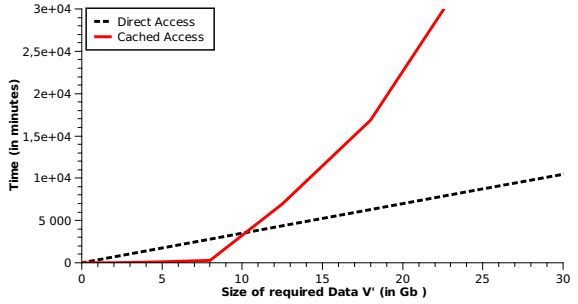

Figure 6: *Algorithm performance comparison between direct access and cache access. Cache access is much faster, but when it saturates, swapping bricks costs much more than many direct accesses to the data.*

Figure 6 shows that when the required data V' is bigger than the cache, a direct access becomes more efficient. Indeed, when the cache is full, any brick can be loaded and unloaded many times until every points of the horizon have been filled. This behaviour leads to a loss of performance. For example, processing P_2 in Figure 5 could require loading again the corresponding brick if it has been released from cache to free memory space for accommodating new bricks.

To postpone further the critical size of the processable datasets, we propose in the next section a strategy where the size of the bricks can decrease over time.

4. The New Slimming Brick Strategy

We present in this section a strategy that allowed us to efficiently propagate horizons in very large seismic volumes. The idea consists in keeping the brick occupancy as low as possible by releasing data that will not be used anymore in the remaining of the propagation. In the next section, we first explain the basic release mechanism and in section 4.2, we profit from this mechanism to release more data of bricks by giving priority to the filling of the "holes" in the propagated horizon.

4.1. Slimming Down by Releasing Used Traces

Looking at the propagation algorithm, one can observe that when a candidate $p(x,y,z)$ is processed, the trace at position (x,y) will not be accessed anymore. Based on this observation, the cache system could be easily optimized by releasing all the processed traces in the bricks. But releasing data is not possible with a classic brick format that store the data in an indexed array stacking all the traces (see Figure 7).

We managed to efficiently release traces in a brick by slightly modifying the storage format of a brick. As illustrated in Figure 7, the new brick format is now a $2D$ matrix of pointer toward independent traces. The suppression of a trace $tr(p)$ becomes easier since it just consists in deleting the $1D$ arrays corresponding to the processed traces.

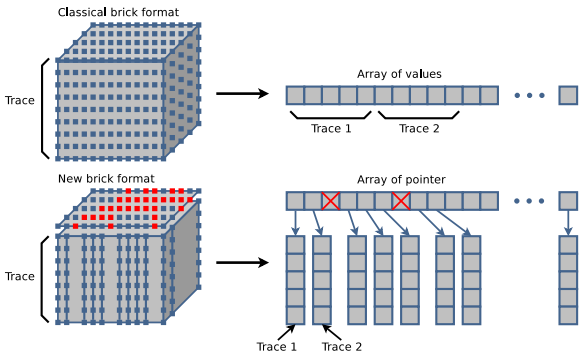

Figure 7: *Comparison between static and slimming brick format*

In order to evaluate the effectiveness of this new cache strategy, let's first compare the memory occupancy of the bricks for these two brick formats during the whole propagation (see Figure 8). The cache used for this evaluation can handle all the requested data V'.

First, we observe that the lifetime of a brick is the same for both cache strategies and is equal to the global time of the propagation. This point is explained by the fact that the distribution of candidate's weights (correlation with the seed) is closed to a uniform density. Then a brick contains an equivalent number of good candidates (processed quickly by the

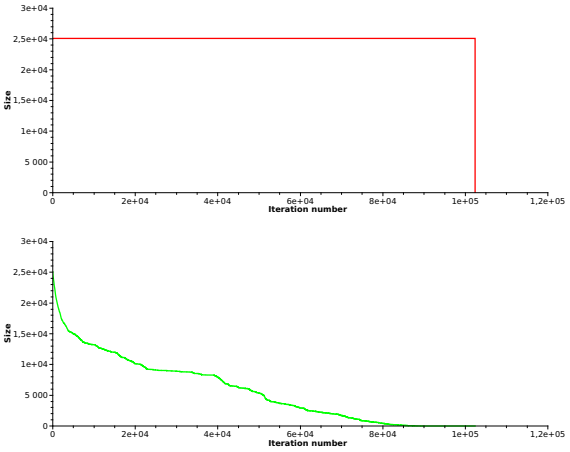

Figure 8: *Averaged brick memory occupancy profiles. (top) The size of a static brick remains maximum until the voxel candidates have been all visited. (bottom) The slimming brick approach releases allocated memory as soon as data becomes useless and therefore allows more bricks to be loaded into the cache.*

queue) and bad candidates (that are statically processed later at the end of the propagation).

The main difference is that with the new strategy, the bricks quickly get slimmer during the propagation. This property is very well highlighted in Figure 9 showing the evolution of the overall cache occupancy during the propagation. The red curve shows that with a classic brick cache system, all the bricks must be loaded to finalize the propagation and are released only at the end of the algorithm. On the contrary, with the slimming brick strategy (green curve), memory is regularly released during the propagation. This strategy allowed us to propagate horizons in volumes 3 times larger than with classic bricks.

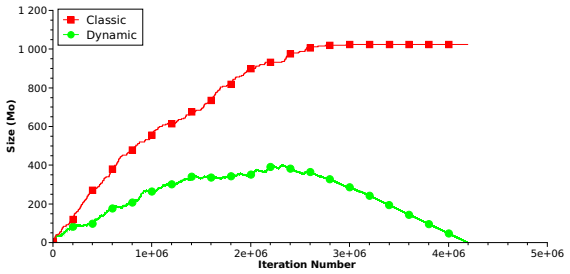

Figure 9: *Comparison of evolution of the overall cache occupancy with a classic brick strategy (red line) and our new slimming brick strategy (green line)*

4.2. Slimming Further Down by Filling Surface Holes

During the propagation, *holes* (closed areas) of various size may form in the propagated horizon (see Figure 2 for instance). These holes appear when the tracking paths find their way around candidates that have lower correlations than the currently propagated candidates. Normally these areas will be treated later when no better quality candidates can be propagated.

But, one can observe that processing those areas could be done independently of the global propagation, indeed candidates on the border of a hole could only influence surface in the limit of this hole. Therefore, with no loss on the global quality of the propagated horizon, we choose to fill the holes when we detect them.

A direct consequence is that it contributes in releasing sooner portions of the bricks data, making more room for other bricks to be accommodated into the cache.

Hole detection is done by a so-called *labeling algorithm* that consists in detecting closed areas inside an image. Many labeling techniques are available in the literature, and for the purpose of this paper, we focused our interest on the method presented in [CCL04]: given a binary image, where white pixels correspond to unpropagated area, the algorithm finds all the closed areas by a contour tracing approach.

The new workflow then consists in applying a hole detection every n iterations (for example, $n = 10000$) and filling these holes by launching a local propagation.

Every n treated candidates :

1. create a $2D$ image of horizon status (+1 for propagated pixels, 0 for loaded pixel but not yet propagated and -1 for not loaded pixels).
2. detect hole by the labeling approach.
3. for each hole :
   a. create a new priority queue Q' with candidates belonging to the hole border.
   b. process Q' with the propagation algorithm.
4. continue global propagation until next hole detection. When dequeuing Q, if a candidate is already propagated then just ignore it.

Figure 10 shows an example of such a process and how thanks to the hole filling operations, cached data can be an advantage to propagate the horizon and to quickly release as much memory as possible.

The comparison of the brick memory occupancies for the two slimming strategy (see Figure 11) shows a faster releasing rate when the hole filling is activated. More bricks could then be loaded into the cache.

Finally Figure 12 shows that our *slimming brick cache strategy with hole filling* (blue curve) requires almost 5 times less cache memory than the classic static brick approach (red curve).

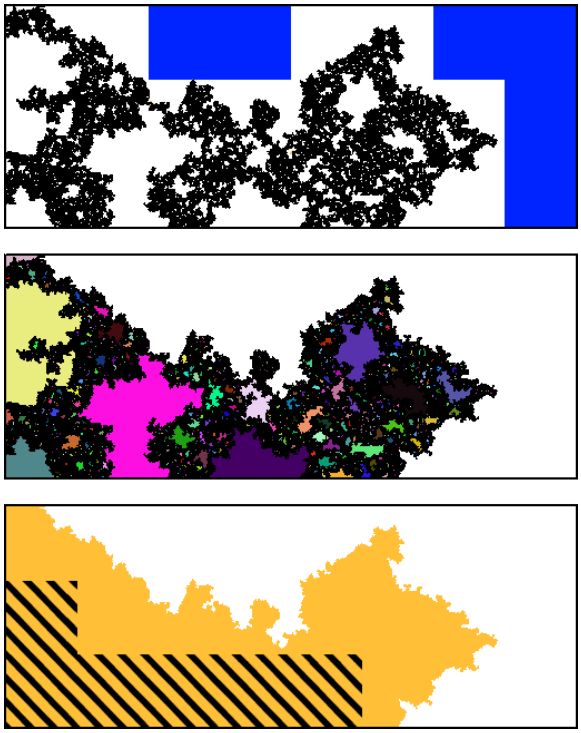

Figure 10: *Hole filling strategy in action. (top) Creation of a status image. Blue pixels represent unloaded bricks. (middle) Region labeling. Holes appear with distinct colors. (bottom) Hole filling by local propagations. Hatched bricks can entirely be released from the cache.*

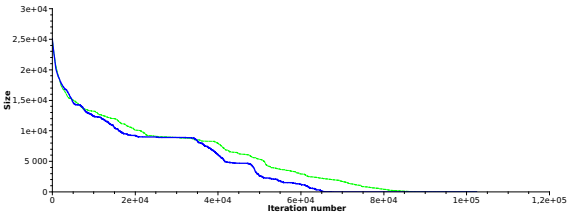

Figure 11: *The brick memory occupancy reduces more rapidly with the hole filling strategy (blue curve) than without (dotted green curve).*

4.3. Slimming cache strategy for very large volumes

We demonstrated in the previous sections how the slimming brick approach can reduce the required amount of cache memory by a ratio of 5. However, it does not solve completely the problem when addressing very large volumes for which we still do not have enough cache memory. We already explained (see Figure 6) the drawback of our strategy for the cache management: bricks could be loaded and unloaded many times and this becomes very costly.

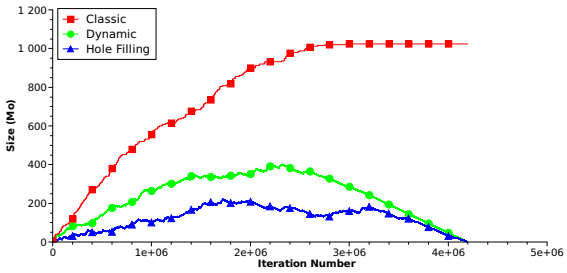

Figure 12: *Comparison of the evolution of cache occupancy with the static bricks (red curve), the slimming bricks (green curve) and the hole filled slimming bricks (blue curve).*

This artifact can be canceled by replacing the replacement strategy behavior with the following strategy :

1. if the cache is full, bricks are kept and outside data accesses are done directly on the global volume to extract the required trace.
2. if there is space left in the cache, a new slimming brick is filled with the traces that have not yet been explored (this is made possible with the new slimming brick format).

This final cache strategy is summarized in Figure 13.

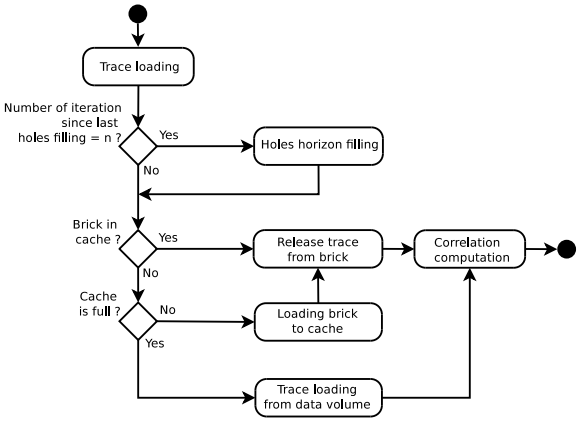

Figure 13: *Slimming cache system algorithm*

The performance of this new cache is displayed in Figure 14. It shows that for a cache size of 8 GB:

1. classic cache can only handle seismic volumes requiring that its V' part size is around the size of the cache system.
2. slimming cache can handle with very equal performances volumes requiring a V' part around 4 times the cache size. For larger volumes, performances still remain better than with pure direct accesses.

More practically, with our new cache strategy, a cache of 8 GB can handle the requested subvolume for propagating an horizon on a volume of 600GB.

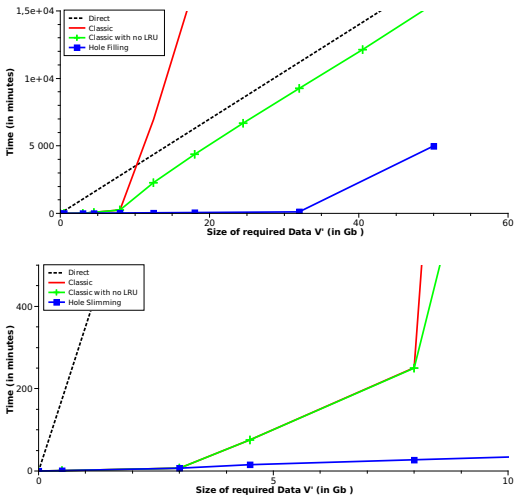

Figure 14: *Comparison of algorithm performances. (black curve) direct accesses with no cache, (red curve) static brick cache with swap problems, (green curve) static brick cache with direct accesses when saturated, (blue curve) slimming brick cache with direct accesses when momentarily saturated*

5. Experimental Results

Finally, to demonstrate our contribution on real data, we present in Figure 15, the result of an horizon propagated on a 200 GB volume ($7000 \times 7000 \times 2200$ on 2 bytes). This horizon has been propagated with an 8 GB cache (3 GB of main memory and 5 GB of disk space) with a global propagation time of 35 minutes. The size of the subvolume V' required for the global propagation is 12 GB ($7000 \times 7000 \times 128$ on 2 bytes). We use a vertical brick size of 128 voxels to take horizon inclination in account. In comparison, the propagation would have been 45 times slower by using of a static brick cache with a direct data access strategy when the cache is full. It would have been 140 times slower for a static brick cache with a management policy, and 100 times slower with a direct data access.

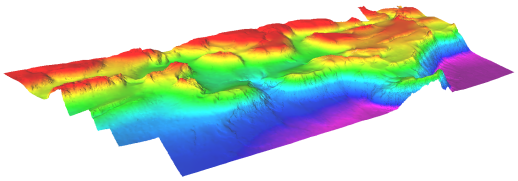

Figure 15: *Example of propagated horizon on a* 200 *GB seismic volume*

6. Conclusion

We have presented in this paper a new cache system suitable for seismic horizon propagation algorithms. It is based on bricks of data that can get slimmer during the propagation process as the accessed data becomes unused and can be released from memory. The release rate has been accelerated by filling in priority holes that appear in the propagated surface.

Our *slimming brick strategy* requires much less cache memory and performs better than other cache strategies on equivalent seismic volumes. It allows to considerably postpone further the size limit of the ever growing seismic volumes one can handle.

All these optimizations have been integrated in Sismage^{TM}, TOTAL in-house interpretation tool.

References

[AZMM04] AL-ZOUBI H., MILENKOVIC A., MILENKOVIC M.: Performance evaluation of cache replacement policies for the spec cpu2000 benchmark suite. In *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference* (2004), pp. 267–272.

[CCL04] CHANG F., CHEN C.-J., LU C.-J.: A linear-time component-labeling algorithm using contour tracing technique. *Comput. Vis. Image Underst.* (2004).

[DJG*08] DUPUY G., JOBARD B., GUILLON S., KESKES N., KOMATITSCH D.: Isosurface extraction and interpretation on very large datasets in geophysics. In *SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling* (2008), pp. 221–229.

[FP04] FARAKLIOTI M., PETROU M.: Horizon picking in 3d seismic data volumes. *Machine Vision and Applications* (2004), 216 – 219.

[GBKG04] GRIMM S., BRUCKNER S., KANITSAR A., GRÖLLER E.: A refined data addressing and processing scheme to accelerate volume raycasting. *Computers & Graphics* (2004).

[HH91] HILDEBRAND, HAROLD: Method for attribute tracking in seismic data. *Patent 5 056 066* (1991).

[KBKG07] KOHLMANN P., BRUCKNER S., KANITSAR A., GRÖLLER E.: Evaluation of a bricked volume layout for a medical workstation based on java, 2007.

[Kes98] KESKES N.: Procédé et programme de propagation d'un marqueur sismique dans un ensemble de traces sismiques. *Patent 2 869 693* (1998).

[KZRM83] KESKES N., ZACCAGNINO P., RETHER D., MERMEY P.: Automatic extraction of 3d seismic horizon. *SEG Las Vegas* (1983).

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (1987), pp. 163–169.

[PBVG10] PATEL D., BRUCKNER S., VIOLA I., GRÖLLER E.: Seismic volume visualization for horizon extraction. In *Proceedings of the IEEE Pacific Visualization Symposium 2010* (2010).