Eurographics Symposium on Parallel Graphics and Visualization (2004)
Dirk Bartz, Bruno Raffin and Han-Wei Shen (Editors)

A hierarchical and view dependent visualization algorithm for tree based AMR data in 2D or 3D

Stéphane Del Pino

CEA DAM/DIF

Abstract

In this paper, a solution to the visualization of huge amount of data provided by solvers using tree based AMR *method is proposed. This approach strongly relies on the hierarchical structure of data and view dependent arguments: only the visible cells will be drawn, reducing consequently the amount of rendered data, selecting only the cells that intersect the screen and whose size is bigger than one pixel.*
After a brief statement of the problem, we recall the main principles of AMR methods. We then proceed to the data analysis which shows notable differences related to the dimension (2 or 3). A natural view dependent decimation algorithm is derived in the 2D case (only visible cells are plotted), while in 3D the treatment is not straightforward. The proposed solution relies then on the use of perspective in order to keep the same guidelines that were used in 2D. We then give a few hints about implementation and perform numerical experiments which confirm the efficiency of the proposed algorithms. We finally discuss this approach and give the sketch for future improvements.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Line and Curve Generation

Keywords: tree based AMR, hierarchy, view dependent, huge quantity of data

Acknowledgements: I would like to address special thanks to J. FAVRE and J.-P. NOMINÉ concerning their comments and advice.

1. Introduction

AMR (Adaptive Mesh Refinement) is a scientific computing technique that allows CPU time or memory saving using automatic hierarchical mesh adaptation methods. Typically, the grid will be "finer" where some quantity shows big variations and "coarser" elsewhere.

Being very efficient this method allows better resolution of problems for the same CPU time and memory cost. So, the quantity of data to post-process remains very important. We have now to handle results of hydrodynamic computations, produced by the HERA package [Jou04], using more than *100 millions* of cells and *10 billions* will be foreseen very soon!

Processing those data is such a formidable task that one can immediately consider that "classical ways" are not enough. Some experiments were done using an SGI-Onyx Infinite reality system with 10 Giga Bytes of memory running 2 pipes and `OpenGL` based packages on a 3200×2400 screen. Generating a single image of the total solution took up to 10 minutes for 10 millions of cells while 5 millions still runs interactively!

If the AMR visualization literature usually focuses on avoiding visualization artifacts (isosurfaces cracks may appear at mesh level variations for example), the treatment of huge amount of data is also investigated. Parallelism [Ma] and dedicated volume rendering [KWWB*] are often the considered techniques. In what follows, we will not deal with that methods but introduce a view dependent algorithm strongly relying on the hierarchical structure of the considered data.

2. What is AMR?

AMR methods are *mesh adaptation* techniques that use hierarchical refinement or unrefinement procedures. They were introduced by M.J. Berger in her PhD Thesis in 1982 [Ber82], but the reference paper was written with J. Oliger in 1984 [BO84].

These methods lead to simple empirical ways of adapting mesh to approximate the solution of problems. Mesh adaptation is a complex task, especially from the unrefinement point of view, so the use of hierarchy makes it much simpler.

Various AMR approaches and ways of applying them have been developed. The two main approaches and their common applications will be presented here. Both of them lead to manipulation of non conform grids (*i.e.* some vertices may live inside some edges). These two approaches are *patch based AMR* and *tree based AMR*. We will go deeper into the second one which is our center of interest.

2.1. Patch based AMR

This method consists in superimposing hierarchical grids on a mesh set in order to get a better approximation of the solution in particular areas. Added grids will be denoted as *patches*. This iterative process can be repeated for more accuracy.

Patches are usually Cartesian rectangular meshes, many of them may live in the same mother grid (see figure 1).

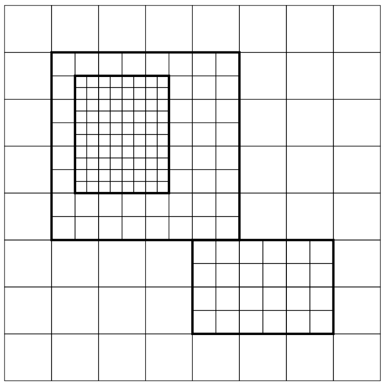

Figure 1: *Example of hierarchical grids. Note that the second level is composed of two grids.*

It may be noted that using nested meshes can lead to refined cells that may not need to be, but it allows the use of simple variations of multi-grid algorithms (see [Bri87]). Figure 2 shows the relationship between patch based AMR and multi-grid decomposition.

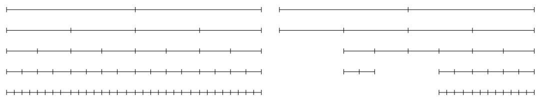

Figure 2: *1D grids. On the left: nested multi-grids. On the right: patched based AMR.*

Patch based AMR is for instance taken into account by `Chombo` and its visualization module `Chombo-Vis` [LVSS*03]. This paper does not focus on high performance visualization but describes `Chombo-Vis` AMR special facilities. It deals with many functionalities: from different level grid combination to seamless isosurface construction.

2.2. Tree based AMR

This variation of the same method is similar to wavelets. The trick is to subdivide chosen cells of the mesh using some criterion values.

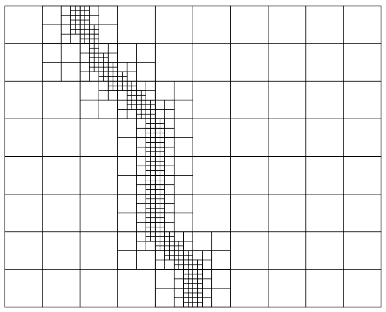

Figure 3: *tree based AMR mesh example over 4 levels. The grid is refined along an interest line, usually a discontinuity.*

A representation of this kind of adapted mesh is given in figure 3. A common constraint that is imposed to improve accuracy is to keep a security zone before going to a new level. This means that if one traverses the mesh in a given direction (x, y or z) a prescribed number n (typically from 3 to 5) of cells of the same level should be forward before a change of level. Note also that no more than one level jump is allowed along a given edge.

This approach is better suited for discontinuities approximations (figure 3 enlightens it) and finite-volume-like numerical methods (see [Lev02]) since only fluxes are exchanged between neighbor cells. Note that the method is unchanged from the Cartesian case when cells belong to the same level and is simply adapted (see figure 4) at level transitions.

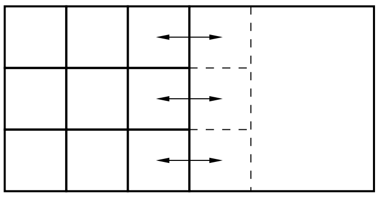

Figure 4: *Numerical flows are computed along smaller edges. This can be interpreted as using the standard scheme considering virtual cells — dashed on the figure.*

Let us remark that both patch and tree based techniques can equivalently be used on the same mesh. The main difference is due to the goal that drives them, and then changes to the hierarchical structure of the mesh.

3. Data analysis

While we just stressed that the two AMR approaches could be interpreted as one, we will now focus on the *tree based* point of view. This means that the data we will deal with will be refined hierarchically and cell by cell along discontinuity lines and surfaces (respectively in 2D and 3D). The following analysis is related to this particular case.

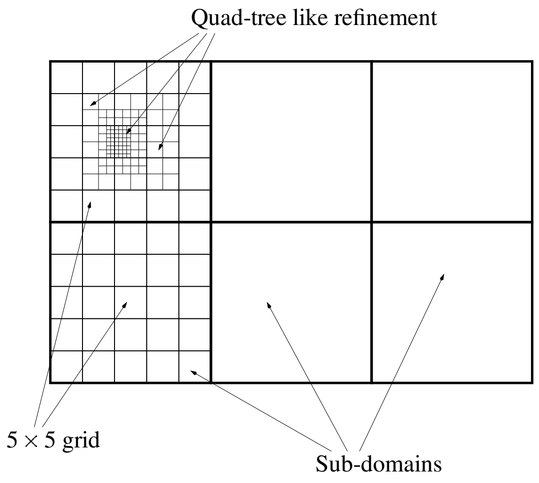

Figure 5: *An example in two dimension of space: the computational domain is composed of six non-overlapping subdomains. Each of them is meshed using a Cartesian grid using the same space steps.*

Since we want to post-process data provided by scientific computing, the type of numerical method may provide us useful hints (see figure 5).

- Data come from a finite volume approximation, they are described by *piecewise constant functions* (constant in each cell).
- The amount of data is such that it can only be obtained using parallel computing. The computational domain is divided in non overlapping sets. A partial solution is associated to each of them and stored in an individual file — a global reconstruction is usually not possible and may be meaningless.
- Each sub-domain is meshed using a Cartesian grid in such a way that before refining them, their union forms a Cartesian uniform and conforming grid.
- Refined cells are obtained using the same pattern for a given computation. The subdivision is the same all along each Cartesian direction (2, 3 or 4). Figure 6 shows patterns that are commonly used in our case.

We will now study representative cases. For each of them,

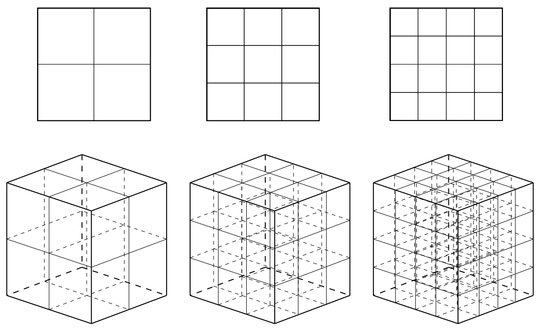

Figure 6: *Patterns: Different refinement patterns stored by dimension and subdivision type.* ***Up:*** *2D cases.* ***Down:*** *3D cases.* ***From left to right:*** 2^d, 3^d *and* 4^d *patterns.*

we sampled discontinuous functions of the form $\mathbf{1}_{\mathcal{O}_1} + \mathbf{1}_{\mathcal{O}_2}$, where $\mathcal{O}_i$ are connected open sets and $\mathbf{1}_{\mathcal{O}_i}$ are given by

$$\mathbf{1}_{\mathcal{O}_i} = \begin{cases} 1 \text{ if } x \in \mathcal{O}_i, \\ 0 \text{ else.} \end{cases}$$

In order to describe this case, we chose the domains this way:

in dimension 2: two disks centered at the origin, for which one border is perturbed using sinus functions to get details at smaller scales (see figure 7);

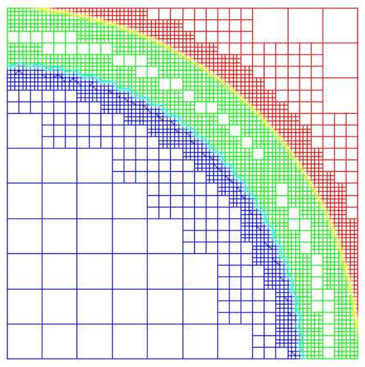

Figure 7: 10×10 *grid refined by a* 3×3 *pattern. Three levels are shown. The mesh is colored using function variations.*

in dimension 3: a sphere and a cylinder also centered at origin. This time, the cylinder's surface is modified (see figure 8).

Those data sets represent typical configurations of hydrodynamic instabilities.

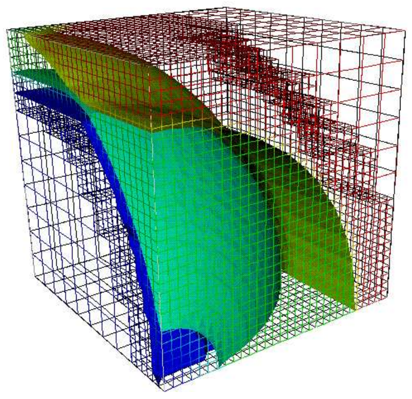

Figure 8: $10 \times 10 \times 10$ *mesh refined using a* $3 \times 3 \times 3$ *pattern. External mesh skin and isosurfaces are shown.*

3.1. 2D data

We first focus on the 2D case, considering the most commonly used refinement pattern: 3×3. We refine the mesh using up to 11 levels to approximate the function up to a number of 30 millions of cells. It leads to the generation of a case that requires the use of high performance visualization. This experiment is detailed by table 1. To create these data we assumed a $1\,000 \times 1\,000$ pixels screen — chosen for the sake of simplicity but representative since it is close to common screen resolutions.

3.1.1. Drawing all the data

Cells			Pixels	
lev.	number	prop. (%)	by cell	total (%)
0	37	4.25E-6	10 000	37
1	253	2.03E-5	1 111	28,1
2	1 838	2.11E-3	123	22,7
3	5 727	6.57E-3	13.7	7,86
4	18 686	2.14E-2	1.52	2,85
5	58 531	6.72E-2	1.69E-1	9.91E-1
6	177 629	2.04E-1	1.88E-2	3.34E-1
7	535 222	6.14E-1	2.09E-3	1.12E-1
8	1 610 669	1.85	2.32E-4	3.74E-2
9	4 838 126	5.55	2.58E-5	1.25E-2
10	14 522 040	16.7	2.87E-6	4.16E-3
11	65 369 430	75	3.19E-7	2.08E-3

Table 1: *This table describes the cell repartition by level of refinement in the particular case of the data represented by figure 7. The grid uses 87,138,188 a total of cells.*

This table is divided in two columns. The first deals with cells, their level, number and proportion in the total set. The second column deals with screen pixels. Pixel usage by cell and by percentage are shown for each level.

Many remarks can be made here. First the number of cells grows exponentially even if the refinement occurs only near an interest line. 75% of the information is related to the last level cells, but this just fills 0.002% of the screen, completely negligible without zooming. Note that the table is divided in two parts after level 4. This has been done to show after which limit cells occupy less than one pixel and their sum less than 1% of the screen. Looking carefully at the data, one will see that 0.03% of the data fills 98.5% of the screen, so that 99.97% of information generates 1.5% of the image.

Remark 1 It is important to note that most of the information is concentrated in data whose contribution to the final image is very small. The effort made to associate an image to the numerical solution is concentrated on "invisible" cells drawing.

This analysis lets us think that getting a good representation of the data may not need to use all of them. We just considered the initial situation: drawing the whole computational domain. The user's interest is to go into his data to see details and to understand them better.

3.1.2. Zooming and windowing

We will now discuss the extreme case which consists of assuming that the smallest cells occupy at least a pixel on the screen.

Let us then suppose that the zoom factor is such that any last level cell can occupy a pixel. Since subdivision pattern is 3×3, each cell of the previous level will use nine pixels. The surface on the screen will be then multiplied by nine for each lower level.

Following the example, a cell surface will be given by $s = 9^{11-l}$ at level l. So a single level 0 cell would require *31 billions* of pixels to be represented!

Let us now consider the worst case: every visible cell at the screen belongs to level 11. Only 1 million of them can simultaneously be plotted, so even in this case, this corresponds to 87 times less cells than the total.

Table 2 tries to show the behavior of a standard zooming along a discontinuity line.

The zoom considered here is the extreme case where the smallest cells (level 11) are visible. In this case, the refined cells draw a line that crosses the screen — the discontinuity line. One can immediately see that the number of needed cells is negligible compared to the total number.

3.2. 3D data

We now investigate the 3D case. As we will see the addition of a third dimension will dramatically change the analysis.

For this study a $1\,000 \times 1\,000$ screen will still be used and we will only consider parallel projection mode. We will also suppose that each cell has two faces parallel to the screen, the others being horizontal or vertical.

Visible cells		
level	number	surface
11	9000	9000
10	1998	17982
9	666	53946
8	222	161838
7	74	485514
6	25	1476225
Total	11985	2204505

Table 2: *Zoom simulation in a refined area around a discontinuity line — surface is given in pixels.*

Remark 2 Those assumptions are not critical but simplify the presentation. This way cells' projection to the screen will be rectangles.

3.2.1. Criterion choice

In the previous paragraph, direct visibility was our discussion criterion. In 3D this is more subtle since many cells may be hidden but very important to the image generation — think to isosurfaces.

One could have considered the volume of the cells, and this would have lead to the same results as in 2D; but in 3D visualization, projection step cannot be put away and changes the rules. This criterion would not give us relevant information.

So one criterion will be a *potential visibility*. This means the fact that a cell is large enough to be visible on the screen if nothing hides it.

3.2.2. Analysis

For this study, we will consider a two part table as well. First the cells, their level, number and level part. Then, pixels that would be used for the projection of one cell. Since this projection is a square we will compute an edge length which will help us to better understand the situation. The results are presented in table 3.

Cells			Pixels	
level	number	proportion (%)	face	edge
0	293	1.16E-4	10000	100
1	9781	3.88E-3	1111	33.3
2	155791	6.18E-2	123	11.1
3	1648095	6.54E-1	13.7	3.70
4	16480523	6.53	1.52	1.23
5	233783199	92.7	0.169	0.41

Table 3: *Cells presence by refinement level. The grid counts 252,077,682 cells (see figure 8).*

The first fact to note is the number of levels. Nearly three times more cells than for the 2D example is achieved in less than 6 levels whereas 12 were necessary in 2D. The cell number grows definitively faster in 3D. From this it can immediately be deduced that the smallest 3D cells will be much larger than the smallest in 2D.

The second remark is in some way contradictory: as previously the great majority of the information is composed of non visible cells — nearly 93% — but it is only carried by one level. Moreover, if those cells are individually not visible, every cell would be potentially visible with a slightly larger screen ($\times 2.5$).

Another bad news is the number of cells whose projection could cover more than one pixel on screen: more than *18 millions*. This is a lot!

Finally all cells contained in the last level could be located on a face of the domain and be in front of the camera: their contribution to the image would then be 100%.

3.3. Synthesis

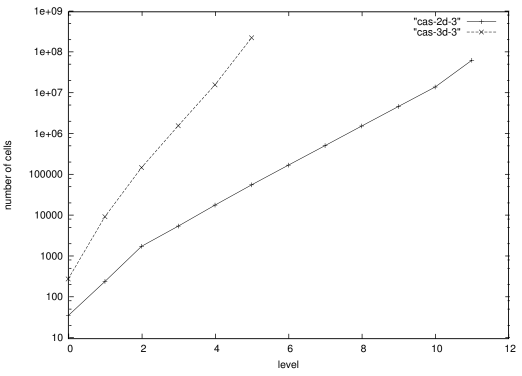

Figure 9: *Comparison of the exponential growing of the number of cells according to the level in 2D and 3D — logarithmic scale.*

Those two examples showed radically different behavior for the 2D and 3D cases. If 2D results could have been easily conjectured, the 3D case is not intuitive. The main reason for this difference is the data scale. It is almost sure that twelve 3D levels would lead to the same kind of repartition but at a different scale — the number of non visible cells would be of gigantic size (see figure 9).

A 2D solution is now easy to propose and since we would like to use the same kind of approach in the 3D case, we will have to introduce a new ingredient.

4. A strategy

The importance of the hierarchy has been stressed. We will first introduce the 2D algorithm and then propose an extension in the 3D case.

For the sake of simplicity we will give the following definition.

Definition 1 We will call *mother cell* a cell that has been subdivided, and *daughter cell* a cell result of subdivision.

Note that according to definition 1 a cell can be daughter as well as mother, which is actually the case in our data structure.

In the following, we will assume that the mesh contains all the hierarchical structure: mothers, daughters and non-refined level 0 cells. The hierarchy of data indicates that the implementation will have to manage s^d-trees, where s is the subdivision and d the dimension of space.

4.1. In two dimensions of space

The analysis made in section 3.1 is promising. We recall here that our goal is the visualization of huge amount of data. As we will see the proposed algorithm's efficiency will grow according to the complexity of the problem.

4.1.1. Level of details

When performance improvement in visualization is considered, three classical approaches are generally possible.

- The first is the easiest — from the user point of view. It consists in using better hardware, but as it was noticed in introduction, even the best video cards do not answer to the question. This situation may continue since while card performances will improve computation size will grow accordingly.
- The second way is a fashionable research subject. As in many other fields requiring huge CPU resources, use of parallel computing seems attractive. But even if some recent research work lead to promising results, performances are still lagging behind. `Chromium`-like tools [HHN*02] allow performance improvement but the scaling is not linear — more precisely, n processors will not lead to a n-time speed-up.
- A third idea has been extensively explored in terrain visualization[LKR*96]. It consists in decimating cleverly the data — depending of the view point — to allow navigation into it. This level of detail approach is the strategy we chose. But, if the basis ideas are close, our field is quite different and more straightforward through some aspects.

  1. The goal of terrain visualization is real-time display. We just seek reasonable interactivity.
  2. The quality criterion is not the same: we want the best representation every time (accuracy preservation).
  3. We want to be able to change the view point, the quality of the transition is not our main concern.
  4. Our data, has shown previously natural hierarchy: we do not need to find a way to decimate it. This makes a big difference with terrain visualization. We just need to exploit the data hierarchy.

Level of detail (LOD) approaches are not new by themselves. Almost all visualization packages provide deteriorate images for transitions, but this is done to improve user comfort. The originality of the work lies in there: we do not propose a new LOD management algorithm, but a way to generate our image using the minimum amount of data *without* any deterioration of the final image.

4.1.2. Visualization view point

In order to decimate information, we will use a criterion which is as intuitive as possible, and show that no better way can be achieved. Some ergonomic details can still be improved in a second time.

A cell will be visible if:

- its projection intersects the screen,
- its image is bigger than a pixel,
- if it is a mother cell, her daughters do not use more than one pixel each.

Let us now precise an important fact: if a visible cell does not correspond to a leaf of the tree, the data it will contain will be the mean of its children.

This is very important since it is consistent with the finite volume method which generated the processed data — finite volume data represent the mean of a quantity inside a cell.

Remark 3 The obtained image will be more correct than the one provided using the full data through the graphic pipeline.

The remark 3 is easy to understand. One of the bottlenecks of visualization is rasterization, so many tricks are used to optimize its speed. A consequence is that one cannot know *a priori* which cell's data will be used — any child of the mother cell can be chosen, depending on the rasterization algorithm.

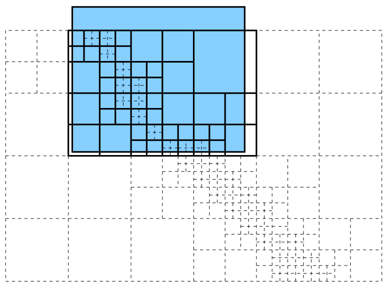
Figure 10: *Only relevant data is drawn. The colored region represents the screen. Plain lines indicate useful cells, dashed ones are for unused cells.*

The method is enlightened by figure 10. This will obviously lead, for implementation, to the management of trees as shown in figure 11.

Before detailing the method that is effectively used, let us

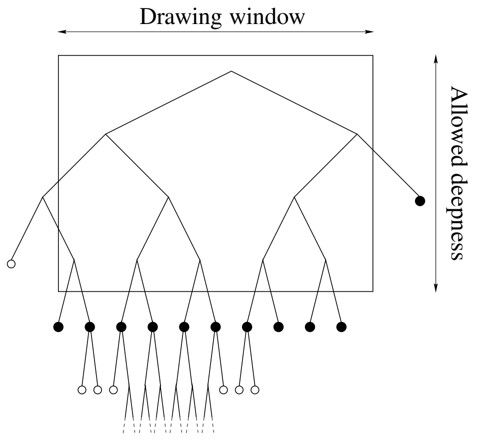

Figure 11: *The implementation consists in browsing a tree by satisfying criteria. Black nodes show drawn cells. In three dimensions of space, the same implementation will be used. Only the criteria will be changed.*

remark that before browsing the tree goes deeper from the root to the leaves, a maximum depth will have to be pre-computed — this is possible since resolution, point of view and zoom factor are known. For instance, it can consist in computing the size of the window in the physical space, it is then easy to compute the size of the cells in pixels. The maximum depth will correspond to the cells whose size is the smallest but still greater than 1 pixel. The algorithms 1 and 2 implement the method.

Algorithm 1 Tree browsing

```
Compute maximum depth
for all grid's cell do
  if the cell intersects drawing window then
    call algorithm 2 using the cell as its argument
  end if
end for
```

Algorithm 2 Going deeper in the tree

```
Require: A cell as argument
  if maximum depth is reached or cell is a leaf of the tree
  then
    Plot the cell
  else
    for all daughter cell do
      if daughter cell intersects view window then
        call algorithm 2 with daughter cell as argument
      end if
    end for
  end if
```

4.2. In three dimensions of space

The algorithm proposed in 4.1 is efficient. One can figure it before doing any numerical tests: *the number of plotted cells is bounded by the screen number of pixels*. One could obviously think that same argument applies also in 3D. In fact, it does not!

The first reason is that we *do not want* to re-implement all visualization algorithms. To find relevant cells this should be done to respect the 2D case rule — one could consider the example of isosurfaces.

The second reason is even more critical: modern visualization packages make extensive use of *transparencies*. This is an essential feature that cannot be ignored. With transparency, some hidden cells may have contribution to the image. It is now easy to figure out that in dimension three:

Remark 4 The number of useful cells *is **not** bounded* by the number of screen pixels. In fact, no absolute bound can be found similarly to the 2D case.

4.2.1. Perspective

Nevertheless, we still want to use the same kind of algorithm that was used in 2D. Taking care of remark 4, a solution consists in finding a way to naturally remove more cells. Unlike the 2D case, we will not be able to guarantee an upper bound to the number of used cells.

Since adding the third dimension is more challenging, we should use a third dimension argument to decimate data. This argument is *perspective*.

We will not be able to give here the same kind of demonstration as in two dimensions, but arguments that will ensure method improvement.

The use of perspective has two consequences that both lead to reduction of visible cells:

1. since perspective projection uses a focal point only cells that intersect an influence cone will be visible.
2. Perspective has the effect that a given object projection surface decays as the object goes away from the eye position.

We will take even more advantage of these two points thanks to the *a priory* classification of the cells. For each size of cell, we can actually define a cone and a distance of visibility (see figure 12).

Figure 12 shows nested sectors. Each of them is associated to a cell size (or level) and pre-computed. Note that sectors get smaller while cell size decays. Considering this 4 levels example, one can check that only a few cells will be used — none of the third level of refinement cell intersects the smallest sector. This seems to give a good answer to our problem.

Using this new cell selection criterion, we will also use the

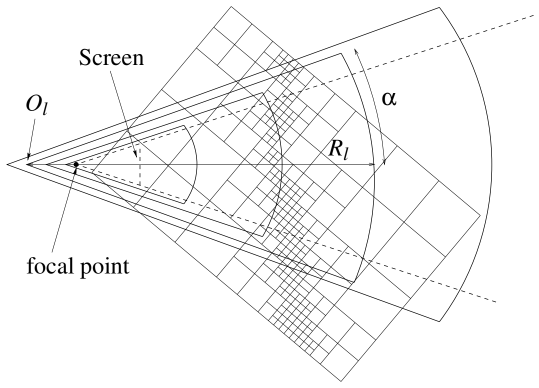

Figure 12: *Use of perspective allows natural decimation of the cells. Note different origins O_l defining each level cone. This is an implementation trick: each sector corresponds to the admissible positions of cells' mass centers.*

algorithms 1 and 2. Implementation will also take advantage of those similarities: the same code will be used for both 2D and 3D, the only variation will be the tree browsing criterion.

4.3. Additional remarks

We have proposed solutions for both 2D and 3D cases that allow lower cost image rendering without quality lost. Since our goal is also to provide interaction to the user, we have to ensure that data extraction — through tree browsing — will not be too expensive. In fact as numerical simulations have shown, this cost is marginal (see section 5).

It could be argued that since the tree browsing is linked to the drawing window size, the number of useful cells will grow exponentially with it; but a simple ergonomic improvement consists in changing minimum cell size in pixels: bigger values implies better performances removing details.

Finally, since the data set is reprocessed each time the view point changes, all visualization information is also to be rebuilt — this means recomputing isosurfaces for example. This could also be optimized by computing isosurfaces incrementally for instance.

5. Implementation and numerical results

We will now discuss some aspects of the implementation and give results of numerical experiments confirming the choices that were made.

To perform those numerical tests we wrote a tool using C++ [Str97] and VTK [SML03]. Performances are reached by extensively using template techniques. This allows for instance to generate predefined trees associated to each refinement pattern. The strategy here is to store all the information into memory in a first time. This is very expensive so those tests were performed on a 64Gb memory machine.

It is not easy to provide numerical tests as there is no reference solution to compare with — drawing of the complete data set may take “infinite” time. To enlighten the performances we will plot refreshing time that proves interactivity. Following this idea the results are presented by figures 13 and 14.

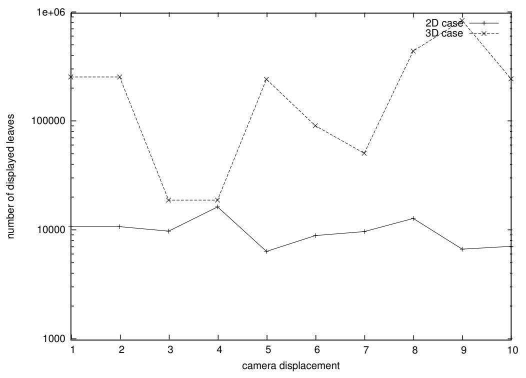

Figure 13: *Number of cells that were used to generate images for 10 view point modifications. 2D and 3D results are compared — logarithmic scale.*

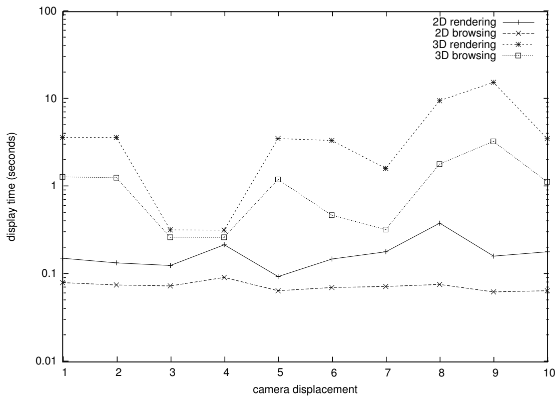

Figure 14: *Comparison of* soft rendering *and* tree browsing *for 10 view point modifications. 2D and 3D cases are presented using a logarithmic scale. View points are the same that for figure 13.*

Those results have been obtained by navigating 10 times into the data provided by tables 1 for the 2D test and 3 in three dimensions. Since the algorithm stores the whole information into memory, those tests have been performed using a 64Gb memory Itanium-II and exporting the display on a SUN blade workstation without 3D hardware in a 1024×728 window.

For both cases, data extraction from the tree runs significantly faster than soft rendering — a factor 3 is often observed.

Dimension two shows very impressive results:

- no more than 20 thousands cells, of the 85 millions original cells, are used after each camera movement.
- The tree browsing requires less than 0.1s, this is close to real time.

3D results may appear weaker since the amount of used cells remains important: nearly one million in the worst observed case, but even in this case, the number of selected cells corresponds to a reduction by a factor 250 of the total number of cells. Interactivity is not really obtained: image requires here up to 19 seconds, but most of this time is rendering. Data extraction never needs more than 3 seconds. The use of 3D hardware should provide better results — this was not possible there since no workstation had enough memory.

6. Conclusion and future work

We have answered the question of visualization of huge quantity of tree based AMR data with an appropriate algorithm. We showed that this algorithm could be applied in both 2D and 3D — if adding the perspective in the later case. If 2D experiments showed some kind of optimality, 3D results may not be completely satisfying — interactivity is however very close. The main reason is that *no upper bound* can be provided to the number of useful cells to a given image. However, it is important to note that the number of used cells could be further reduced by changing some criteria (minimum number of pixels to draw a cell, focal point of the camera to change perspective...), and then allow to browse more interactively the data and only get better quality image when required by the user.

This approach can also be coupled to more standard techniques like parallelism to improve performances particularly in the 3D case.

Keeping all the information into memory is not realistic. We are now following the path of E. COLIN and G. HARREL [CH03], who experienced a similar view dependent algorithm but implemented in an out-of-core way based on the rewriting of the data in an HDF5 file that stores the hierarchical information. Ongoing work consists into merging the two approaches to improve performance as well as memory consumption. This should consists in managing distributed trees: a part in memory and the other living on the disk.

Future work will go further in this direction since many improvements can already be foreseen: parallel browsing of the trees, cache management policies, for instance.

References

[Ber82] BERGER M.: *Adaptive Mesh Refinement for Time-Dependent Partial Differential Equations*. Ph.d. dissertation, Stanford University, 1982. Computer Science Report No. STAN-CS-82-924.

[BO84] BERGER M., OLIGER J.: Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics 53* (Mar. 1984), 484–512.

[Bri87] BRIGGS W. L.: *Society for Industrial and Applied Mathematics, A Multigrid Tutorial.* Philadelphia, PA, 1987.

[CH03] COLIN E., HAREL G.: *Étude de techniques de visualisation haute performance: Applications à des grandeurs sur maillage 2D-3D AMR "tree-based" à l'aide de VTK et de HDF5.* Tech. rep., CEA/DIF (internal report), 2003.

[HHN*02] HUMPHREYS G., HOUSTON M., NG R., FRANK R., KLOSOWSKI J. T., AHERN S., KIRCHNER P. D.: Chromium: A stream-processing framework for interactive rendering on clusters. In *ACM SIGGRAPH* (2002).

[Jou04] JOURDREN H.: *HERA: an AMR hydrodynamic plateform for multiphysics simulation.* Lecture notes in computational engineering, Springer, 2004.

[KWWB*] KREYLOS O., WEBER GUNTHER O. H., WES BETHEL E., SHALF J. M., HAMANN B., JOY K. I.: Remote interactive direct volume rendering of amr data.

[Lev02] LEVEQUE R. J.: *Finite volume methods for hyperbolic problems.* Cambridge Texts in Applied Mathematics. Cambridge: Cambridge University Press. xix, 558 p., 2002.

[LKR*96] LINDSTROM P., KOLLER D., RIBARSKY W., HODGES L. F., FAUST N., TURNER G. A.: Real-time, continuous level of detail rendering of height fields. In *ACM SIGGRAPH 96* (August 1996), pp. 109–118.

[LVSS*03] LIGOCKI T. J., VAN STRAALEN B., SHALF J. M., WEBER G. H., HAMANN B.: A framework for visualizing hierarchical computations. In *Hierarchical and geometrical methods in scientific visualization*. Springer, 2003, pp. 197–204.

[Ma] MA K.-L.: *Parallel Rendering of 3D AMR Data on the SGI/Cray T3E.* Tech. rep., Institute of Computer Applications in Science and Engineering, Mail Stop 403, NASA Langley Research Center Hampton, Virginia.

[SML03] SCHROEDER W., MARTIN K., LORENSEN B.: *The Visualization Toolkit An Object-Oriented Approach To 3D Graphics, 3rd Edition.* Kitware, Inc. publishers, 2003.

[Str97] STROUSTRUP B.: *The C++ programming language*, 3nd ed. Addison-Wesley, 1997.