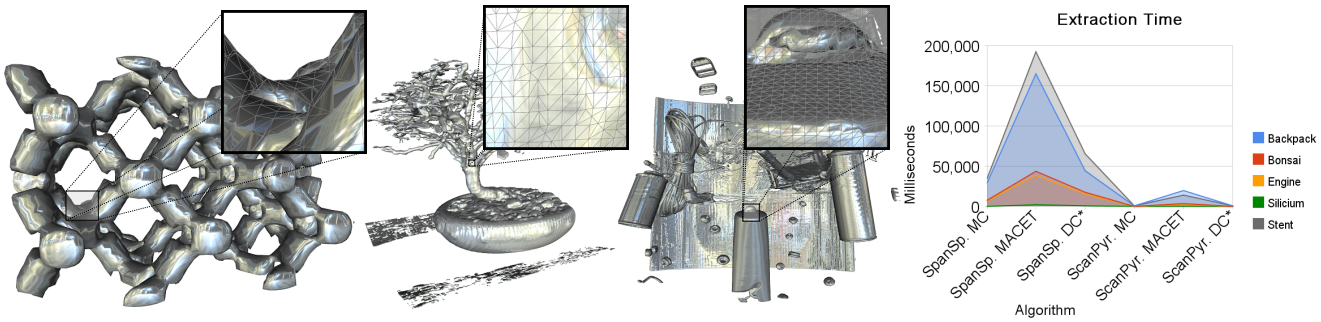
Interactive Isosurface Contouring and High Quality Meshes

Leonardo A. Schmitz
laschmitz@inf.ufrgs.br

Carlos A. Dietrich
cadietrich@inf.ufrgs.br

João L. D. Comba
comba@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul



Our contouring methods, MACET and MDC, achieve interactive framerates while extracting high quality meshes. Left to right: Silicium Dataset (64*64*128) polygonized with MC; Bonsai Dataset (256*256*256) polygonized with MACET; Backpack Dataset (512*512*256) polygonized with MDC; Extraction times from Span Space (CPU) and from Scan Pyramids (GPU). Datasets extracted from *volvis*[10].

Abstract

The interactive polygonization of isosurfaces is still a challenging problem. The improvement of 3D image scanners used in Medicine and the advances of mechanical simulation techniques in Engineering and Physics result in a large availability of huge 3D datasets, which are a real challenge to polygonization techniques available today. Although we are also experiencing significant advances in the CPU architecture, which is becoming more and more parallel, the processing power of the current CPUs seems to be insufficient to deal with this task. On the other hand, the advance of graphics hardware (Graphics Processing Unit or GPU) has been seen as an interesting alternative to implement computationally expensive algorithms. In this work, we demonstrate how the massive parallelism of current GPUs can be explored to improve efficiency of common subdivision-based polygonization methods. We show how we can achieve a speed up of 100 times in the implementation of Macet (a Marching Cubes extension with improved mesh quality) and Dual Contouring, enabling the interactive exploration of huge datasets in a low-cost hardware. Our implementation is based on a new GPU data structure which optimizes the search for cells which intersect the isosurface and reduces significantly the cost of the "marching" procedure.

1. Introduction

The recent advances in modern graphics hardware allow computation of not only rendering, but also general purpose algorithms (GPGPU[15]). Intrinsically parallel problems can benefit from this hardware architecture with divide-and-conquer strategies. Most of the polygonizers of isosurfaces do a grid subdivision, so they naturally fit. When scientists deal with huge amount of volume data, as 512^3, they have to wait from seconds to minutes to extract an isosurface. Consequently, the interactive navigation through isovalues is not possible. There are even worse situations: if these meshes are used in simulation procedures, they need high quality aspect ratio triangles.

The classical approach of isosurface extraction is Marching Cubes[14] (MC), which is fast and very straightforward to implement. However, it has some shortcomings such as ambiguity[21, 17], poor aspect ratio triangle meshes[20] and no octree adaptivity. These are well-known problems that have already been solved, but at the cost of poorer overall speed. Bloomethal[2] and Hall and Warren[6] are examples of that, but the use of GPUs to increase their speed is constrained by complex topology requirements. The grid adapts and the topology of the primitives inside relates directly to neighbor cells. This access in the grid neighborhood is difficult to map to the GPU. If all cells are com-

pletely independent, then the algorithm is easily adapted to the graphics hardware.

Therefore we chose to revise algorithms whose neighbouring cells have loosely coupled topology requirements. More important, algorithms that address some weaknesses of Marching Cubes. MACET[3] and Dual Contouring[11] have these charateristics, specially regarding poor aspect triangles. We use a novel approach to compute the latter, so it will be referred as Modified Dual Contouring (MDC).

To achieve maximum performance in the isosurface extraction, we used a GPU data structure.

The main contributions of this project are:

1. The extraction of isosurfaces at interactive frame rates, while retaining quality;
2. The creation of a Dual Contouring scheme for independence in topology.
3. The creation a simple Dual Contouring method.

This paper is organized as follows: we present a short description of previous work in Section 2. Section 3 and Section 4 outline MACET and MDC. Section 5 discusses the GPU data structure Scan Pyramid. Finally, Section 6 discusses results and Section 7 presents conclusions along with future research possibilities.

2. Related Work

The isosurface polygonization is one of the most challenging problems of computer graphics in the last decades. Since the pioneering works of Udupa [9], we have seen the proposal of several (and significantly different) approaches for extracting polygon meshes from implicit surfaces. These approaches can be categorized into three broad classes: (1) methods based on domain subdivision, (2) pseudo-physical methods and (3) surface tracking methods.

Methods based on domain subdivision follow the *divide-and-conquer* paradigm, and propose the subdivision of the domain of the function f ($f : R^n \rightarrow R$) into a set of *cells*, which are processed independently. The behavior of the isosurface in the interior of each cell is approximated with a set of triangles, in a way that, when visualized together, the local meshes form a watertight mesh (C^0). The efficiency and robustness of this approach attracted a lot of attention from academic and professional communities in the last years, since the proposal of the Marching Cubes algorithm (MC) [14], the most famous example of this category.

The MC algorithm itself can be subdivided into two steps: (a) the detection of *active cells* (the cells that intersect the isosurface) and (b) the generation of the triangle meshes inside each active cell. The latter step was subject of many works which attempt to improve the correctness of the algorithm [12] [13] [16]. The detection of active cells, in its turn, was the subject of works which attempt to improve the efficiency of the algorithm [19] [1] [5]. These works are commonly based on pre-processing stages, which organize the access to the domain of function f according to a predefined isovalue.

The Span Space structure [19] is one of the most interesting techniques that accelerate the access to active cells available today. It is basically a spatial hashing where the cells are organized in a 2D map, based on the minimum and maximum values of f in the cell vertices. The map is constructed in order to the active cells corresponding to any isovalue λ be constrained to a subset of the map, which is easily determined from λ.

The advance of GPUs (Graphics Processing Unit) in the last years also gave room to the design of new GPU-based acceleration data structures [5]. The Geometry Shader pipeline stage, for example, allows the creation and deletion of primitives inside the GPU. This powerful tool allowed the implementation of a GPU-based MC, where both steps of the algorithm are performed inside the Geometry Shader stage, with significant performance improvements when compared to CPU-based implementations.

The research for new GPU-based data structures to accelerate the detection of active cells becomes one of the most interesting research themes in this area. The application of the HistoPyramids data structure to accelerate the detection of active cells in MC [4], for example, results in the fastest implementation of a polygonizer so far. The algorithm solves elegantly two well-know problems of GPU-based polygonization methods, namely the high bandwidth demanded to trigger the GPU computations and the empty-space skipping, that is, the access to non-active cells.

We follow the approach of Dyken et al. [4] in this work, and propose the adaptation of HistoPyramids data structure to two subdivision-based polygonizers: (1) the Macet algorithm of Dietrich et al. [3] and the Dual Contouring [11] algorithm. The GPU-based Macet algorithm shows how the GPU implementation of MC is still receptive to the improvements proposed to the original MC, namely the warping of active cells proposed in Macet to improve the quality of the extracted mesh. The GPU-based Dual Contouring shows how we can adapt the HistoPyramids structure to support a different method to generate the triangulation inside each active cell, since Dual Contouring generates a mesh *dual* to MC. Both implementations stress the GPU capabilities, and result in the fastest (to our knowledge) polygonizers available today.

3. Accelerating the Detection of Active Cells

The bottleneck ...

The pyramid data structure is the reason why Dyken *et al* achieved interactive framerates in MC. We used it in MACET, with minor changes, and in MDC, with a novel approach. There are three steps needed to use the data structure: the base texture construction, the base reduction and the pyramid traversal. Each of these steps is mapped to a different shader in the implementation.

The MDC and MACET algorithm themselves are mixed with both base and traversal steps, while the reduction is data structure specific. Histogram Pyramids is a generic data structure, so it is also found in other techniques[22]. We chose a different name to Histogram Pyramids based on Harris and Dyken's thread discussion[8] in nVidia GPGPU forum.

3.1. Pyramid base construction

The pyramid base construction is the central part of the algorithm. It defines the primitives and their topology. They are stored on a 2D texture that can be mapped to a 3D texture[7]. Hence, this is the pass where the shader specifies the number of vertices needed (in this case, avoiding the use of Geometry Shader). In terms of memory, if RGBA is chosen as texture type, the information of four voxels is written in each pixel of the base.

MACET writes the sum of points needed per voxel case. The topology table of MC has the information, so we just have to count the edge labels. This means that the shader samples the 3D texture on the grid points and discovers what enumeration is that. In contrast, DC needs a bitwise table in order to maintain the topology information. Thus, we created one by following the idea that only three edges are unique per voxel (x, y and z axis).

The topology of MACET comes from MC, while MDC has Surface Nets one. As stated before, there are at most three unique active edges per voxel (four bits, as can be seen in Figure 3), so the table has the topology of three quads at maximum and sixteen bitwise combinations. Each color channel can be used to store both the bitwise info and the sum. This can be done in implementation by separating the fraction of the significative part from a *float* type.

3.2. Base reduction

After the stream creation, there is a bottom-up reduction (stream compaction) of the base texture (see Figure 2).

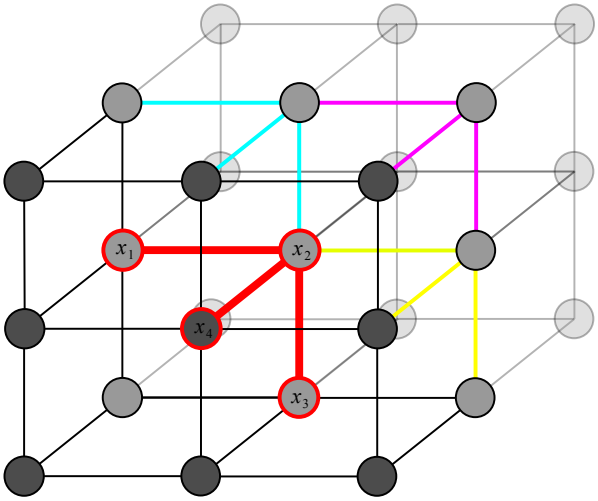


Figure 3. Criteria used to estabilish Modified Dual Contouring topology table. The colored edges are the ones evaluated per voxel. The highlighted red edges present a four bit combination x_1, x_2, x_3, x_4 used to dictate the cell topology. For example, 1101 means the y axis is active (the lowest edge point is outside the isosurface).

It follows the proportions of MipMapping, in which four pixels map to one in the upper texture and so forth, until it reaches the last and single pixel at the top. The data mapped is a sum and not and average as usually happens in MipMaps.

When the base is mapped to the base-1 level, some special care is necessary: only the number of primitives in the base is passed up to R. The topology specified at the fraction of *float* must be ignored. For example, R = 9.546875 means there are 9 points and the enumeration is 10001100, but only 9 is forwarded. The number of bits is not a problem regarding overflow (in this case, excessive points), because nowadays GPU's support *float32*, which allow numbers as big as 17.179.869.180 per channel.

3.3. Traversal

At this point, the pyramid data structure has been built. In the single pixel top, the sum of all vertices (n) is located. It is read back by the application and used to emit the number of primitives required to create the isosurface. The application sends indices of the primitives (k), from 1 to n, to the traversal shader.

The index k is used to reach the correct pixel in the base texture in a top-down traversal. When it reaches the base, it

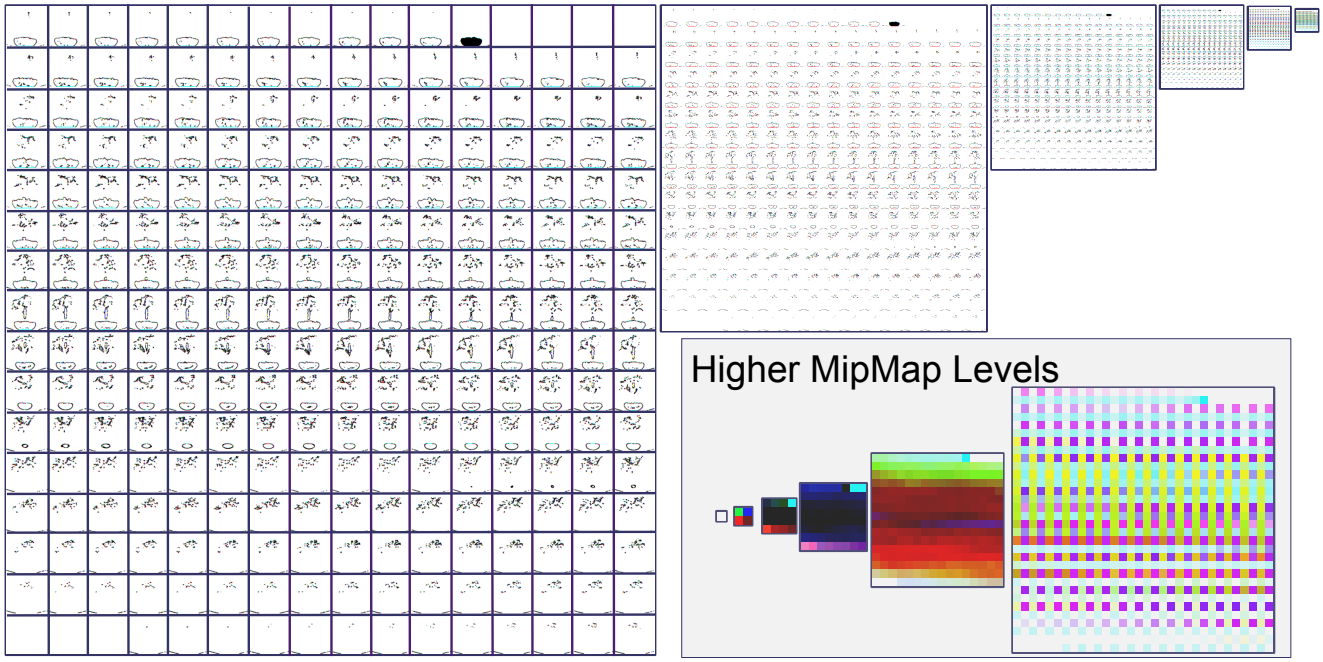


Figure 2. Illustrative Pyramid MipMap texture. The base shows the bonsai dataset in flat 3D with inverted colors. Each color channel represents one dataset voxel. At higher levels the information gets cluttered, so the images get more colorful. However, notice that these images do not represent the full pyramid, because they do not show Alpha channel (transparency). In addition, big values also do not appear, because colors are clamped at one (maximum).

will be pointing at the exact position inside the enumeration table. At each mipmap level, it is necessary to find out what pixel position $P_{x,y}$ that k' (k updated per level) leads to.

There are four candidate siblings $P_{-1,1}$ (upper-left), $P_{1,1}$ (upper-right), $P_{-1,-1}$ (lower-left) and $P_{1,-1}$ (lower-right) per level. They are chosen based on the intervals of their values ($V_{x,y}$). $k' = k$ at the first mipmap level, but then the first interval endpoint is subtracted from k'.

Let

$$\begin{aligned} x &= V_{-1,1}, \\ y &= V_{-1,1} + V_{1,1}, \\ z &= V_{-1,1} + V_{1,1} + V_{-1,-1}, \\ w &= V_{-1,1} + V_{1,1} + V_{-1,-1} + V_{1,-1} \end{aligned}$$

So,

$$\begin{aligned} P_{-1,1} &= [0; x), && (1) \\ P_{1,1} &= [x; y), && (2) \\ P_{-1,-1} &= [y; z), && (3) \\ P_{1,-1} &= [z; w]. && (4) \end{aligned}$$

For example, if $k = 10$ (first level index), $V_{-1,1} = 7$, $V_{1,1} = 3$, $V_{-1,-1} = 5$ and $V_{1,-1} = 0$, then k maps to the pixel $P_{-1,-1} = [y = 7 + 3; z = 7 + 3 + 5)$ (lower-left pixel). After that, $k' = k - y$, where $y = 7 + 3$.

When the level of mipmap is the base, the remainder of k' is used to get to the position inside the topology table. For instance, if $k' = 7$ and the topology case is 1001 (x and z active) in MDC, then it will be the third point of the second quad (around z).

4. GPU-based Marching Cubes

...

4.1. Macet

The algorithm is an extension of MC. It changes the position of the grid edges so that situations in which bad triangles happen are avoided. The authors have not found an ideal transform, but found two (see Figure 4) that improve edge quality. They are combined in postprocess, and can be described as follows:

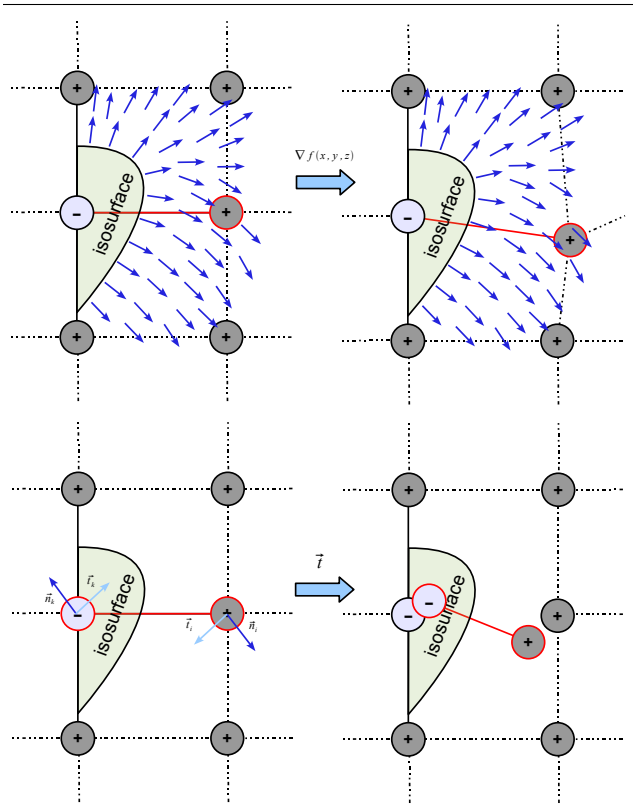


Figure 4. MACET 2D gradient and tangent edge transform examples. The gradient field outside the isosurface (upper-left); One edge point marked with red contour moving along the gradient (upper-right); Both edge points tangent trails (lower-left); Edge after tangent transform (lower-right).

- **Gradient**: The edge points ($\overline{V_iV_j}$) move along the gradient

$$\nabla f(x,y,z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right) \tag{5}$$

in which the partial derivatives of f are approximated by fixed sampled differences (through the use of the grid lattices L) per axis:

$$\begin{aligned} g_x &= f(x+L_x,y,z) - f(x-L_x,y,z), \\ g_y &= f(x,y+L_y,z) - f(x,y-L_y,z), \\ g_z &= f(x,y,z+L_z) - f(x,y,z-L_z), \end{aligned}$$

$$\nabla f(x,y,z) \approx \|(g_x,g_y,g_z)\| \tag{6}$$

Thus, the points are moved

$$V_i = V_i + \alpha * \nabla f(V_i) \tag{7}$$
$$V_j = V_j + \alpha * \nabla f(V_j) \tag{8}$$

The constraint required to maintain differentiable meshes is to test whether one edge point crossed the isosurface. Furthermore, it uses α to limit the step.

- **Tangent**: The edge points ($\overline{V_iV_j}$) move along a tangent trail in order to leave the edge as close as possible to becoming perpendicular to the isosurface. In the case the edge is active the points are moved using the local coordinates ($\vec{b}$ stands for binormal):

$$\begin{aligned} \vec{b_i} &= \nabla f(V_i) \times (V_j - V_i), \\ V_i &= V_i + \alpha * (\vec{b_i} \times \nabla f(V_i)), \end{aligned} \tag{9}$$

$$\begin{aligned} \vec{b_j} &= \nabla f(V_j) \times (V_i - V_j), \\ V_j &= V_j + \alpha * (\vec{b_j} \times \nabla f(V_j)) \end{aligned} \tag{10}$$

The algorithm has the same constraints as the gradient. Equation 2 is used as the points local coordinate normals.

After extracting both meshes, the combination is done by establishing a quality criteria evaluation per vertex. The one used by Dietrich *et al* was radii ratio average [18] of the triangles that share the vertex. Moreover, the best vertices are chosen between tangent and gradient transform. Notice that the neighboring quality changes after a vertex is picked, so the method needs more than one pass to converge. However, the combination of both has complex topology requirements (accessing triangles in neighbour cells), which are too costly for a GPU implementation. This is not a problem since it can be done in the CPU, in milliseconds, using saved GPU meshes. Besides, the bottleneck of MACET is the mesh extraction and not their combination.

5. GPU-based Dual Contouring

The algorithm presented here is different from the one presented by Ju. The minimizer of the quadratic error function (QEF) is found by our particle based method. This is why we refer as the Modified Dual Contouring method instead of Dual Contouring. It iterativelly moves the quad points toward the isosurface. These points will be treated and referred hereafter as particles.

The particles start at a centroid $\bar{C}$ of the cell, calculated by the arithmetic mean of all active edge intersection points (P_i). This preprocess reduces the time for the particle to converge.

$$\bar{C} = \sum_{i=1}^{n} \frac{P_i}{n} \tag{11}$$

The next step is to find the force $\vec{F}$ that moves the particle at $\bar{C}$ toward the minimizer of the quadratic error function (Figure 5). It is is generated using forces $\vec{F_0}$ to $\vec{F_8}$ located at the grid points $\vec{V_0}$ to $\vec{V_8}$. These forces are calculated as

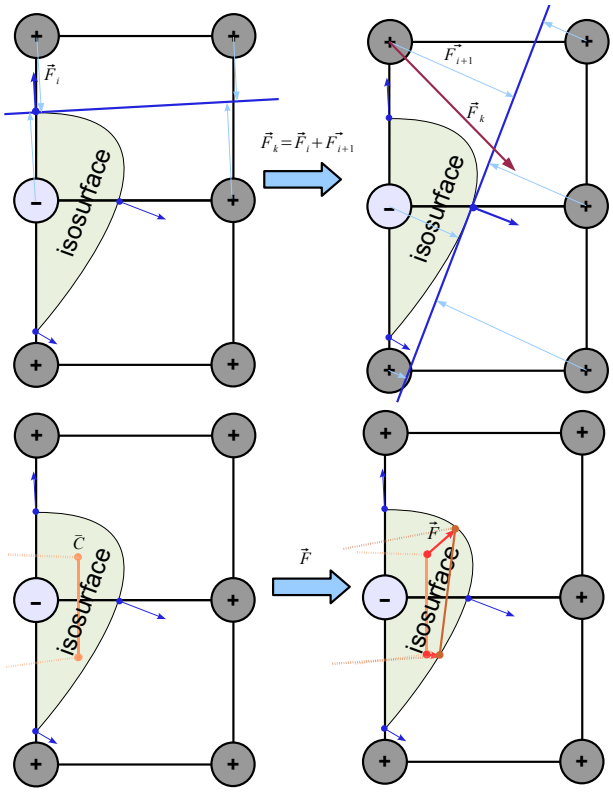


Figure 5. Dual Contouring 2D example. First active edge iteration to calculate forces $\vec{F_0}$ to $\vec{F_4}$ on the voxel grid points (upper-left). The dark-blue arrows represent the plane normals at the intersection points; Second active edge iteration and example of one resulting force $\vec{F_k}$ (upper-right); Surface Nets initial line positioning (represented in orange) using the centroid $\bar{C}$ (lower-left); After integration in n steps, the points converge to the minimizer of the quadratic error function (lower-right).

the sum of the vector distances to the planes defined by all pairs $\vec{n_i}$ (normal approximated by the gradient $\nabla f(P_i)$) and P_i (intersection points from the voxel):

$$\vec{F_k} = \sum_{i=1}^{n} (\vec{n_i}, (P_i \cdot \vec{n_i})) \cdot S * \vec{n_i} \qquad (12)$$

in which S represents a plane. $S \equiv (\widehat{L} \cdot x = -p)$ in Hessian coordinates, with the lattices L of the active edge used as normal, $w = 1$ and the point p at the coordinate system origin. With the eight forces at hand, the force $\vec{F}$ is found by the trilinear interpolation:

$$\begin{aligned}
\vec{F_{l_1}} &= \left(1 - \left(\frac{\bar{C}_x}{L_x}\right)\right) * \vec{F_0} + \left(\frac{\bar{C}_x}{L_x}\right) * \vec{F_3}, \\
\vec{F_{l_2}} &= \left(1 - \left(\frac{\bar{C}_x}{L_x}\right)\right) * \vec{F_4} + \left(\frac{\bar{C}_x}{L_x}\right) * \vec{F_7}, \\
\vec{F_{l_3}} &= \left(1 - \left(\frac{\bar{C}_x}{L_x}\right)\right) * \vec{F_1} + \left(\frac{\bar{C}_x}{L_x}\right) * \vec{F_2}, \\
\vec{F_{l_4}} &= \left(1 - \left(\frac{\bar{C}_x}{L_x}\right)\right) * \vec{F_5} + \left(\frac{\bar{C}_x}{L_x}\right) * \vec{F_6}, \\
\vec{F_{b_1}} &= \left(1 - \left(\frac{\bar{C}_y}{L_y}\right)\right) * \vec{F_{l_1}} + \left(\frac{\bar{C}_y}{L_y}\right) * \vec{F_{l_2}}, \\
\vec{F_{b_2}} &= \left(1 - \left(\frac{\bar{C}_y}{L_y}\right)\right) * \vec{F_{l_3}} + \left(\frac{\bar{C}_y}{L_y}\right) * \vec{F_{l_4}}, \\
\vec{F} &= \left(1 - \left(\frac{\bar{C}_z}{L_z}\right)\right) * \vec{F_{b_1}} + \left(\frac{\bar{C}_y}{L_z}\right) * \vec{F_{b_2}} \qquad (13)
\end{aligned}$$

After that, the particle moves along the force $\vec{F}$ with a diminished magnitude driven by a constant. The constant we used was 5 %, which produced good results. This new position is used in the trilinear interpolation to find a new $\vec{F}$. This cycle is done n (threshold driven) steps until particle converges to the isosurface.

6. Discussion and Results

First of all, the shader language used is GLSL (OpenGL Shader Language) and the equipment used was a GeForce[1] 8800 GTX 768MB with an AMD Athlon[2] 64 Processor 2.2 GHz. There is one extension explored which need the GLSL v1.20, the *NV_transform_feedback*. Transform Feedback is used to save the geometry passed to a Vertex Buffer Object in order to polygonize the isosurface only once. The datasets used were Backpack (512*512*256), Stent (512*512*173), Bonsai (256*256*256), Engine (256*256*128) and Silicium (128*128*64)[10]. Some of their dimensions and border values are not the original ones.

Time (milliseconds)					
	Backpack	Bonsai	Engine	Silicium	Stent
CPU MC	29503.7	7899.57	6908.83	323.711	35126.6
CPU MACET	164852	43704.6	38642.6	1967	191969
CPU MDC	44453.7	17382.2	15182.8	558.186	65366.2
GPU MC	593.878	114.808	103.032	4.20861	461.932
GPU MACET	1984.63	3404.21	3047.22	60.0819	14112.7
GPU MDC	408.56	116.462	100.956	6.63994	455.331

Table 1. Extraction times.

1 Copyright ©2007 NVIDIA Corporation
2 Copyright ©2007 Advanced Micro Devices, Inc.

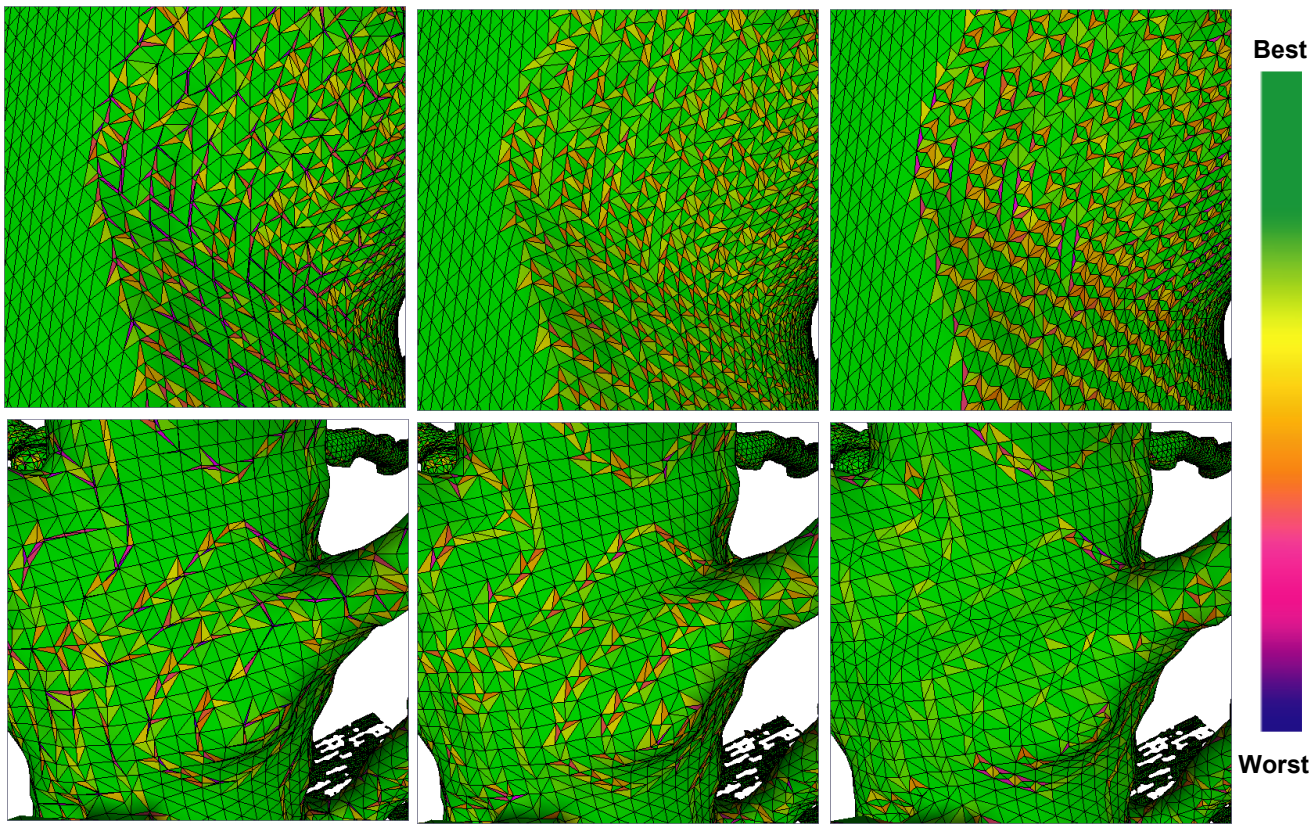


Figure 6. Engine (top-row) and Bonsai (last-row) datasets; From left to right: MC, MACET and MDC. The color scheme illustrates the radii ratio quality mapped into colors. As one can see, MC suffers from very poor shaped triangle meshes, while MDC and MACET do not.

The span space (CPU) reaches interactive frame rates with datasets no larger than 128^3 (see table 1). With larger ones, the extraction time looses interactivity. This happens because there is no massive parallelism on the CPU yet. In contrast, the bottleneck of scan pyramid is the number of triangles and not the volume size. This leads to a great advantage regarding datasets which have sparse isosurfaces.

The GPU technique does not use any strategy to go directly to probable voxel as the CPU does. So keep in mind that if span space is not used, the GPU extraction times leave the CPU's even more behind. Moreover, the data of span space preprocessing time is not included in the table 1, which takes about the same as sampling the entire dataset.

The radii ratio average and minimum in MC is inferior to both MDC and MACET (see Figure 7). The best triangle mesh results can be seen on MACET, which has an average of 80 % and its worst case is at least 200 times better than MC. However, its computational costs are the most expensive. On the other hand, MDC may be an alternative when the time requirements are stricter than the triangle mesh quality.

The quality regarding the distance of the isosurface to the polygons is difficult to measure. We adopted a simple isolevel metric error with a MC comparison along. By resampling the intersection point and calculating the difference to the isovalue, we have a gross and dataset specific error. Nevertheless, it suffices our need to investigate MACET and MDC. We did the comparison by viewing the differences to MC (which is considered good) using the same dataset. All techniques had similar results to MC, showing that they were also near the isosurface.

7. Conclusions and Future Work

This paper presented two robust polygonizers of isosurfaces that achieved interactive framerates. We showed how to explore the parallel nature of graphics processors, while avoiding its constraints. Topology is still an obstacle to the

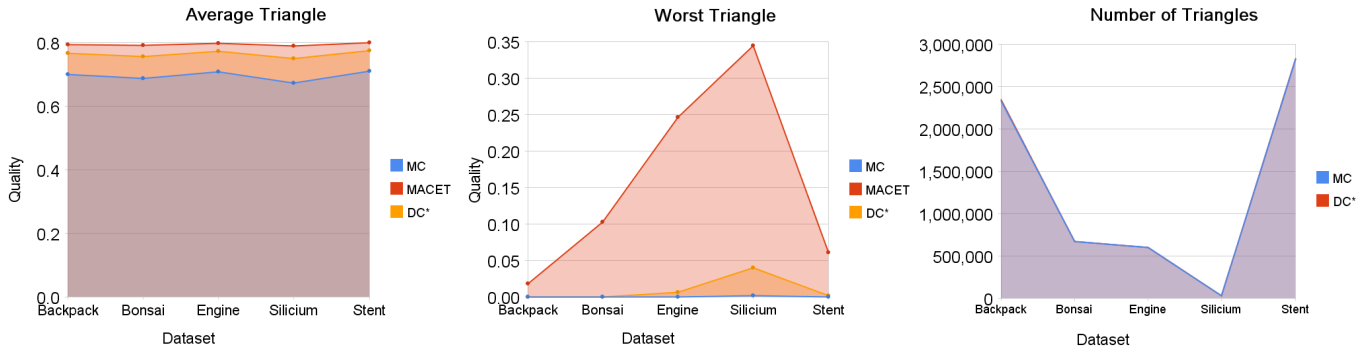


Figure 7. Quality comparison and number of triangles. From left to right: The average quality follows MC $<$ MDC $<$ MACET; The worst triangle case (important for Finite Element Analisys) is the best using MACET; The number of triangles (MDC counts the triangles as quads divided into two).

GPU and was not solved here. We chose Dual Contouring and MACET because they are intrinsically parallel, with minor topological issues.

One of the strenghts of Dual Contouring, the octree adaptativity, is currently being explored and was not included in the paper. This type of refinement reduces significantly the extraction time, because it only computes cells in which there is high variation on the curvature of the isosurface. MACET could use the octree scheme proposed by Bloomenthal, but there are some big topological requirements that would need to be fulfilled on the GPU.

Span Space or other strategy to get directly to probable active cells will likely accelerate both GPU algorithms. The only disadvantage of this type of approach are the cache misses. This destroys the streaming of neighboring voxels, which implies in no reuse of samples in the grid.

The ambiguity problem of MC is solved with octree adaptivity in MDC, but its still unresolved in MACET. With extended enumeration tables or with one more sample at the center of the cell, the issue might be closed. However, more computation means more extraction time, which is already a problem in MACET. Thus, finding alternatives may be more interesting.

Acknowledgments. This research has been sposored by CAPES.

References

[1] C. L. Bajaj, V. Pascucci, and D. R. Schikore. Fast isocontouring for improved interactivity. In *VVS '96: Proceedings of the 1996 Symposium on Volume Visualization*, pages 39–ff., Piscataway, NJ, USA, 1996. IEEE Press.

[2] J. Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988.

[3] C. A. Dietrich, J. L. D. Comba, L. P. Nedel, C. Scheidegger, J. Schreiner, and C. T. Silva. Edge transformations for improving mesh quality of marching cubes. *IEEE Transactions on Visualization and Computer Graphics*, 2007.

[4] C. Dyken, G. Ziegler, C. Theobalt, and H.-P. Seidel. Gpu marching cubes on shader model 3.0 and 4.0. Research Report MPI-I-2007-4-006, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, August 2007.

[5] C. Dyken, G. Ziegler, C. Theobalt, and H.-P. Seidel. Histopyramids in iso-surface extraction. Technical Report MPI-I-2007-4-006, Max-Planck-Institut für Informatik, August 2007.

[6] M. Hall and J. Warren. Adaptive polygonalization of implicitly defined surfaces. *IEEE Comput. Graph. Appl.*, 10(6):33–42, 1990.

[7] M. J. Harris, W. V. Baxter, T. Scheuermann, and A. Lastra. Simulation of cloud dynamics on graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 92–101, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[8] M. J. Harris and G. Ziegler. Gpgpu forum thread - http://www.gpgpu.org/forums/viewtopic.php?t=3577.

[9] G. T. Herman and J. K. Udupa. Display of 3d digital images: Computational foundations and medical applications. In *Medcomp '82*, pages 308–314. IEEE Computer Society Press, 1982.

[10] http://www.volvis.org/. Datasets extracted from the site, 2008.

[11] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. *ACM Trans. Graph.*, 21(3):339–346, 2002.

[12] T. Lewiner, H. Lopes, A. W. Vieira, and G. Tavares. Efficient implementation of marching cubes' cases with topo-

logical guarantees. *Journal of Graphics Tools*, 8(2):1–15, 2003.

[13] A. Lopes and K. Brodlie. Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):16–29, 2003.

[14] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, New York, NY, USA, 1987. ACM Press.

[15] M. R. Macedonia. The gpu enters computing's mainstream. *IEEE Computer*, 36(10):106–108, 2003.

[16] G. M. Nielson. On marching cubes. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):283–297, 2003.

[17] G. M. Nielson and B. Hamann. The asymptotic decider: resolving the ambiguity in marching cubes. In *VIS '91: Proceedings of the 2nd conference on Visualization '91*, pages 83–91, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[18] P. P. Pébay and T. J. Baker. A comparison of triangle quality measures.

[19] H.-W. Shen, C. D. Hansen, Y. Livnat, and C. R. Johnson. Isosurfacing in span space with utmost efficiency (issue). In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 287–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[20] J. R. Shewchuk. What is a good linear finite element? - interpolation, conditioning, anisotropy, and quality measures.

[21] J. Wilhelms and A. V. Gelder. Topological considerations in isosurface generation extended abstract. In *VVS '90: Proceedings of the 1990 workshop on Volume visualization*, pages 79–86, New York, NY, USA, 1990. ACM Press.

[22] G. Ziegler, R. Dimitrov, C. Theobalt, and H.-P. Seidel. Real-time quadtree analysis using HistoPyramids. In *Real-Time Image Processing 2007. Edited by Kehtarnavaz, Nasser; Carlsohn, Matthias F.. Proceedings of the SPIE, Volume 6496, pp. 64960L (2007).*, volume 6496 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, Feb. 2007.