Efficient and High Quality Contouring of Isosurfaces on Uniform Grids

Leonardo A. Schmitz
laschmitz@gmail.com

Carlos A. Dietrich
cadietrich@gmail.com
Instituto de Informática - UFRGS, Brazil

João L.D. Comba
comba@inf.ufrgs.br

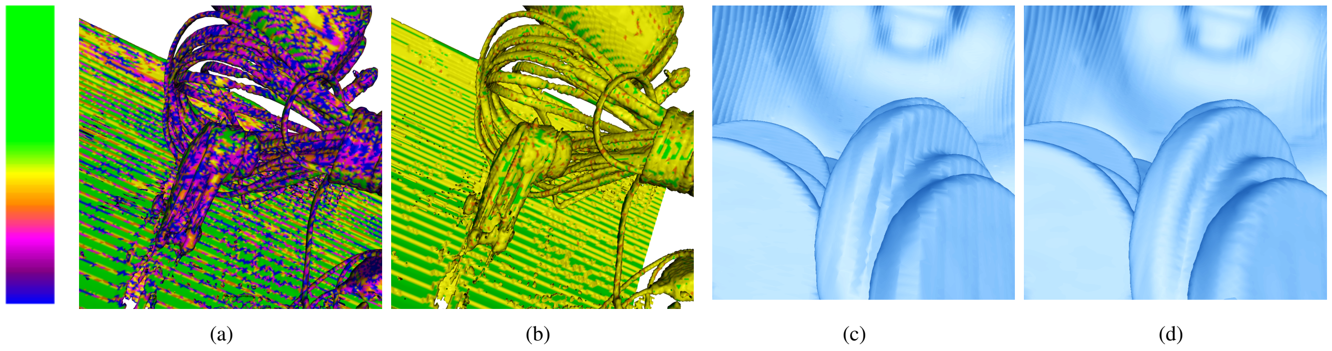


Figure 1. GPU Marching Cubes (a) and (c) and higher quality GPU Macet (b) and Dual Contouring (d). Backpack dataset is used in (a) and (b), while Pig dataset is used in (c) and (d). Triangle quality is measured as radii-ratio in (a) and (b), normalized between 0 and 1, and color-code using the scale on the left (green = good, blue = bad). Fewer terracing artifacts and a better approximation of the polygonization are presented in (d) in comparison to (c).

***Abstract*—The interactive polygonization of isosurfaces has become possible with the mapping of the Marching Cubes (MC) and Marching Tetrahedra (MT) algorithms to GPUs. Such mapping is not as straightforward in cases that the algorithm generate meshes closer to the isosurface or result in better polygon shapes, since they often require complex computations for the vertex positioning of the polygons or even do not have table-driven implementations. In this paper, we revisit Dual Contouring (DC) and Macet algorithms and propose, respectively: (i) a novel parallel efficient version on uniform grids and (ii) novel GPU modules which extend the original MC. Our DC algorithm is table-driven and positions the vertices in a particle-based fashion, which is then used to map into a GPU implementation. In addition, we enumerate the current ways to implement efficient contouring algorithms on the GPUs as orthogonal features, and present the tradeoff of each approach. We validate the efficiency of our algorithms with its comparison to interactive versions of MC algorithms.**

***Keywords*-Isosurface extraction; Volumetric visualization; Contouring;**

I. Introduction

Several tasks in scientific processing have well-established algorithms and techniques. However, the growing need for faster methods and high accuracy results that we are experiencing now may require the constant improvement of these algorithms. For instance, isosurface extraction using polygonization algorithms is still a challenging task, since improvements in image scanners or advances in simulation techniques result in very large volumetric images. Having the ability to interactively change the isovalue while visualizing the data would offer the scientist a better insight on the data being processed, but this is not always possible for large data.

The intrinsic parallel nature of the most common isosurfacing algorithms such as Marching Cubes (MC) [1] makes it amenable for parallel (and thus faster) implementations. Even though there is an increasing parallel power on using multi-core CPUs, the most promising approach today for this task is to use the parallel power of GPUs. There are different alternatives on how such algorithms can be mapped to GPUs and they impact how efficient the polygonization will be. The crucial step is related to the geometry specification, and two proposals were described for MC. The first relies on implementing the entire MC algorithm inside the geometry shader (GS) feature of current GPUs. Such feature represents a programmable stage added into the graphics pipeline to allow limited control on the creation and deletion of vertices (and consequently triangles). Results show a clear performance improvement over the CPU implementation, but several performance issues regarding the use of the GS suggest that results could be even better. Acceleration structures such as Span Space (SS) [2] can also be combined with the GS for improved performance. A different approach to the problem is to use a solution of multiple passes in the fragment shader, called HistoPyramids (HP) [3]. Results using this method are faster than using the single pass GS, but it also has shortcomings, since it requires large GPU memory allocation and may limit the size of input volumes.

Other contouring algorithms that address problems with MC can also benefit from the computing power of GPUs. For improvement of triangle quality, the Macet algorithm [4], [5], [6] is a modification of MC and its implementation on the GPU requires the same extra modules of the CPU. For the detection of sharp features, an interesting choice is the Dual Contouring (DC) algorithm [7], which generates quads on the dual grid. However, mapping this algorithm to the GPU is not straightforward. First, its adaptive nature resulting from using an octree instead of a regular grid leads to a more elaborate neighboring analysis while generating the polygonal approximation. Also, the vertex positioning on the dual grid is computed by solving a QR or SV decomposition in order to minimize a Quadric Error Function (QEF), which can be memory expensive since it requires 10 floats per QEF and is not easy to implement as a linear interpolation of MC.

In this work, we describe a mapping of a modified version of DC to the GPU. The first change in the algorithm is the use of an uniform grid instead of an indexed octree. We believe this is an acceptable trade-off, since the adaptive version on the CPU has to compute a full octree (uniform grid) before simplification while our version is much faster processed on the GPU and can be simplified on the CPU in a post process. The use of a regular grid allows rewriting the algorithm into a table-driven version, which generates quads for each cell from a fixed number of possibilities encoded in a table, in a similar fashion to MC. The second change is the replacement of the QEF approach to place intersection points over the isosurface by a particle-based minimizer function that is simpler to compute.

In order to evaluate the results, we revisit the MC implementations on the GPU, and compare against an implementation of Macet and the table-driven DC on the GPU. For each method, we evaluate different implementations on the GPU using the GS, the GS with SS, and using HP. We obtain results from isosurface extraction with speed-ups to 1500 times in the implementation of both Macet and Dual Contouring, thus enabling the interactive exploration of isosurfaces.

In summary, the main contributions in this paper are:

1) The acceleration of high-quality polygonization of isosurfaces.
    a) The proposal of a table-driven approach for Dual Contouring;
    b) The acceleration of Macet [6] with the mapping to the GPU;
2) The validation of the proposed approaches.

II. Related Work

Isosurface polygonization is a well-studied problem, but still a focus of interest nowadays. Since the pioneering work of Udupa [8], several (and significantly different) approaches for extracting polygon meshes from implicit surfaces were proposed. Our work is focused on methods based on domain subdivision, since its underlying approach usually leads to high performance polygonizers. Such methods follow the *divide-and-conquer* paradigm, that subdivides the domain of the function f ($f : R^n \rightarrow R$) into a set of *cells*, which are processed independently. The isosurface inside each cell is approximated by a set of triangles, in a way that when combined for all cells, form a watertight mesh (C^0). This approach, the well-known Marching Cubes [1], is efficient and robust. MC operates on two fundamental steps: (a) detection of *active cells* (cells crossed by the isosurface) and (b) generation of triangles inside each active cell. The first step was the subject of works which attempt to improve the efficiency of the algorithm [2], [9], [3], while many work focus on improving the quality and correctness of the triangles generated in the second step [10], [11], [12].

Triangular mesh quality was first improved with the help of pseudo-physical smoothing algorithms like SurfaceNets [13]. The SurfaceNets algorithm is based on the *Cuberille* sampling technique, which places vertices at the center of each active cell and connect them to vertices in adjacent active cells. The resulting mesh, as a dual of MC base mesh (in the absence of self-intersections), does not have as many badly-shaped triangles as MC. The quality of the mesh is further improved with a post-processing smoothing step, which applies a Laplacian smoothing while constraining each vertex to the active cells in that it was generated. The Dual Contouring (DC) algorithm [7] combines the sampling technique of SurfaceNets with the feature sensitive approach of Extended Marching Cubes [14], which results in an even more accurate polygonizer. Each vertex, instead of being positioned in the center of the active cell, is pushed to the corner of the sharp features (if they exist) in the interior of the cell. This approach improves the accuracy of the sampling technique, while standing in the efficiency of the SurfaceNets. The Macet [6] algorithm was based on the notion of edge transformations and edge groups. By allowing the position of MC edges to be placed in more convenient locations and creating a convenient table, their proposal achieves much better triangle quality and requires minor extensions in the original MC code.

Improving active cell detection is commonly done in preprocessing stages, which organize the access to the domain of function f according to a predefined isovalue. The Span Space structure [2] is a spatial hashing with cells organized in a 2D map, based on the minimum and maximum values of f at cell vertices. The map is constructed in such way that active cells corresponding to any isovalue are constrained to a subset of the map, which are easily determined from the isovalue used. GPU-based polygonizers were also proposed to minimize detection costs by computing each cell in a separate processor.

The early programmable modules allowed custom shading on pixels and vertices, but lacked geometry generation inside the GPU. The Marching Tetrahedra (MT) polygonizer described by Pascucci [15] performed a significant part of the extraction inside the graphics processor. A quadrilateral was used as input for every tetrahedra, because it represents the configuration case with the maximum number of output primitives. If a configuration of a tetrahedra resulted in a single triangle, then two points of the quad were collapsed into one. If there was an empty cell, the quad was discarded by collapsing its points into a single point. In the case of a uniform grid, twenty or twenty four points were sent per voxel, which became too costly for the CPU. Reck *et al* [16] used Span Space to reduce the amount of geometry sent to the GPU. Klein [17] used vertex arrays and shifted computations to the fast pixel processors. Kipfer [18] further improved this implementation by letting tetrahedra share vertices. However, all the above algorithms perform tetrahedral subdivision, which is still undesirable for accuracy purposes [19] and for their excessive increase in number of triangles in comparison with MC.

The MC polygonizers described by Goetz [20] and Johansson [21] discovered the number of primitives generated by each cell on the CPU and culled empty cells with Span Space. Thus, excessive computation was left for the CPU, resulting in a bottleneck. On-the-fly MC only appeared recently with the new shader pipeline.

Current GPUs have the geometry shader (GS) [22]. It allows the creation and deletion of primitives, making obsolete previous workarounds to create geometry and taking out excessive computations from CPU. MC and other subdivision based polygonizers were fully implemented inside the GPU [23]. However, an alternative data structure for compaction/expansion of primitives, called Histogram Pyramids (HP or HistoPyramids) [3], minimized CPU to GPU communication. It uses multiple rendering passes to create the geometry from a set of canonical indices (allocated as triangles on the GPU memory).

In this work, we evaluate how the GS and the HP impact the GPU implementations of Macet and DC. The GPU-based Macet algorithm shows edge transformations that improve the quality of the extracted mesh. The GPU-based DC shows improvement in the approximation of the isosurface. Both implementations explore GPU parallelism in detail and result in the fastest (to our knowledge) high-quality polygonizers available today.

III. Dual Contouring

Dual Contouring is a method which refines the accuracy of Marching Cubes and works for adaptive grid resolutions. It is a hybrid method between Extended Marching Cubes [14] and SurfaceNets. The first has the characteristic of finding sharp edges based on the point normals of the active edges. The latter has a dual space nature. This combination leverages the good characteristics from both methods.

The algorithm consists on finding the active edge and creating a quadrilateral around it, as in Surface Nets. After that, it positions the vertex in the isosurface feature using information of position p_i of the cut points (as MC does) and also their normals n_i. This is similar to the way that Extended Marching Cubes (EMC) sample features. Vertex positioning is done by minimizing the QEF described in equation 1 with QR or SV decompositions.

$$E[x] = \sum_i (n_i \cdot (x - p_i))^2 \qquad (1)$$

The adaptivity of DC is an advantage over most polygonizers. Since most isosurface extractors have a large memory demand, DC is an interesting alternative. In addition, there is an implicit feature identification. Both characteristics are probably the most important features of DC. However, its implementation is not as trivial as MC. Both SVD and QRD often require complex computations for the vertex positioning of polygons and are useful for adaptive resolutions, therefore we created a novel iterative particle and table-based approach.

A. Particle-based minimizer function

This method replaces the QR factorization with a particle-based approach, which iteratively moves the vertices of the quad (referred and treated hereafter as particles) towards the isosurface. The iteration is responsible for a good approximation of the isosurface, as well as finding sharp features. We ensure the stability of the method by constraining the particle inside the voxel.

The particles start at the mass point $\bar{C}$ of the cell, calculated from the arithmetic mean of the active edge intersection points (P_i). This process reduces the number of iterations of the particle and is a good hint of the isosurface location [24]. The next step is to find the force $\vec{F}$ that starts moving the particle from $\bar{C}$ towards the isosurface (Figure 2). Since the data used is Hermite, the normals of cut points are necessary. We use the gradient of the volume to approximate the normals ($\vec{n_i} = \nabla f(P_i)$).

The force $\vec{F}$ is generated by trilinear interpolation of the forces $\vec{F_0}$ to $\vec{F_7}$ located at the grid points $\vec{V_0}$ to $\vec{V_7}$ driven by the centroid $\bar{C}$. These forces are calculated as the sum of the vector distances to the planes defined by all pairs $\vec{n_i}$ and P_i (intersection points from the voxel):

$$\vec{F_k} = \sum_{i=1}^{n} (\vec{n_i}, (P_i \cdot \vec{n_i})) \cdot S * \vec{n_i} \qquad (2)$$

in which $S \equiv (\widehat{L} \cdot x = -p)$ represents a plane in the Hessian normal form, the grid lattices L of the active edge used as normal, $w = 1$ and the point p at origin of the coordinate system.

Particle moves along the force $\vec{F}$ with a diminished magnitude driven by a constant ($c * \vec{F}$). Best results were obtained using 5% of $\vec{F}$ in all our volumes. This new position ($\bar{C}' = \bar{C} + (c * \vec{F})$) is used in the second iteration as input to the trilinear interpolation (recalculate $\vec{F}$). This procedure is repeated for a fixed number of steps or until the particle converges (threshold driven) to the isosurface.

B. *Table-Driven Geometry Specification*

There are three unique edges per voxel (x, y and z axis) and the table is constructed based on them. We chose three edges that share the same origin, resulting in only four vertices to be analysed per voxel. We classify vertices as inside or outside the isosurface (similarly to Marching Cubes), mapping the combination into bits, as can be seen on Figure 4. The resulting table is even simpler than the one of Marching Tetrahedra, which uses a combination of five vertices. The complete and literal table used for Dual Contouring is described in Figure 3.

0000: none
0001: V_4', V_0', V_1', V_5'
0010: V_4', V_7', V_3', V_0'
0011: V_4', V_0', V_1', V_5' & V_4', V_7', V_3', V_0'
0100: V_4', V_0', V_3', V_2'
0101: V_4', V_0', V_1', V_5' & V_0', V_3', V_2', V_1'
0110: V_4', V_7', V_3', V_0' & V_0', V_3', V_2', V_1'
0111: V_4', V_0', V_1', V_5' & V_4', V_7', V_3', V_0' & V_0', V_3', V_2', V_1'
Complement from cases 0 to 7:
1000: V_4', V_0', V_1', V_5' & V_4', V_7', V_3', V_0' & V_0', V_3', V_2', V_1'
1001: V_4', V_7', V_3', V_0' & V_0', V_3', V_2', V_1'
1010: V_4', V_0', V_1', V_5' & V_0', V_3', V_2', V_1'
1011: V_4', V_0', V_3', V_2'
1100: V_4', V_0', V_1', V_5' & V_4', V_7', V_3', V_0'
1101: V_4', V_7', V_3', V_0'
1110: V_4', V_0', V_1', V_5'
1111: none

Figure 3. Indices for generating DC quads. The index is used to identify in which voxel the dual vertex V_X' is positioned. The dual vertices ordering is described in figure 4.

The elements of the table are the neighboring voxel labels for the vertices of that quad, instead of the edge labels in MC. For instance, if the index $v_4v_3x_2x_1$ is 0001, the first vertex is located in the voxel along its z direction (see Figure 4), the second is the current voxel, the third is in the upper voxel y and the last one is in the in z plus y direction.

IV. GPU Polygonization of Isosurfaces

A. *Geometry Shader*

The Geometry Shader allows the deletion or creation of a limited number of primitives after a vertex program. Therefore, if points can be deleted or transformed into triangles, subdivision-based polygonization can be implemented. Each voxel that contains the isosurface is triangulated by its corresponding point. The voxels in which the isosurface does not cross are discarded. These empty voxels can be avoided by Span Space on the CPU. This accelerates even more the algorithm, but implies in no longer transparent polygonization to the CPU.

B. *HistoPyramids*

HistoPyramids does not skip empty space, but leaves the scalar function accesses to the very fast fragment processors. It also creates vertices and discard them, not directly inside the GPU, but with a multipass strategy. It compacts data similarly to scan [25] and saves the compaction steps in a pyramid that the CPU expands indirectly. The complete implementation is more complex than the one of Geometry Shader, thus more detail is necessary.

The HistoPyramids data structure is used for creation/deletion of primitives inside an uniform grid. Each voxel has its information mapped into one texel in the pyramid base. The deletion is achieved through data filtering [26], [25], while the expansion is achieved by the indirect communication from CPU to the GPU. There are three steps needed to use this data structure: the base construction, the base reduction and the pyramid traversal. Each of these steps is mapped into one or more shader passes.

The pyramid base contains all the topology and vertex information. Its construction is done in the first shader pass and it stores all data in texture memory. A flat 3D method [27] is used to map one texel to one 3D voxel, which is basically tiling z 2D textures into a squared texture. Hence, in this step the shader specifies the number of vertices, which can be triangles or any other pre-specified type of primitives, and their topology inside each voxel.

The base is then reduced with multiple shader passes into only one pixel, which contains the sum of all primitives. This step is analog to the deletion of primitives on the Geometry Shader. The reduction is bottom-up and the textures follow the proportions of MipMaps. Each four pixels in the lower texture is mapped into one in the upper. The mapping function corresponds to the sum of the primitives contained in the base. Thus, at the top there is the number of primitives required, which can be read back by the CPU to be expanded into actual primitives.

At this point, with the pyramid data structure built and the number of primitives available on the CPU, the traversal per vertex happens. The single texel in the top, which contains the sum of all vertices (n), is analog to a root of a tree that points at four siblings with a hashing-like funtion. Each texel (voxel) in the base texture is reached through unique identifiers (k), from 0 to n (sum), which enumerate each vertex. Hence, no direct communication from the CPU is needed by the GPU besides emitting the sum of vertices with predefined indices.

This index k trespasses all texture levels in a top-down traversal. At each mipmap level, it is necessary to find out what texel position $T_{x,y}$ that k' (k updated per level) points

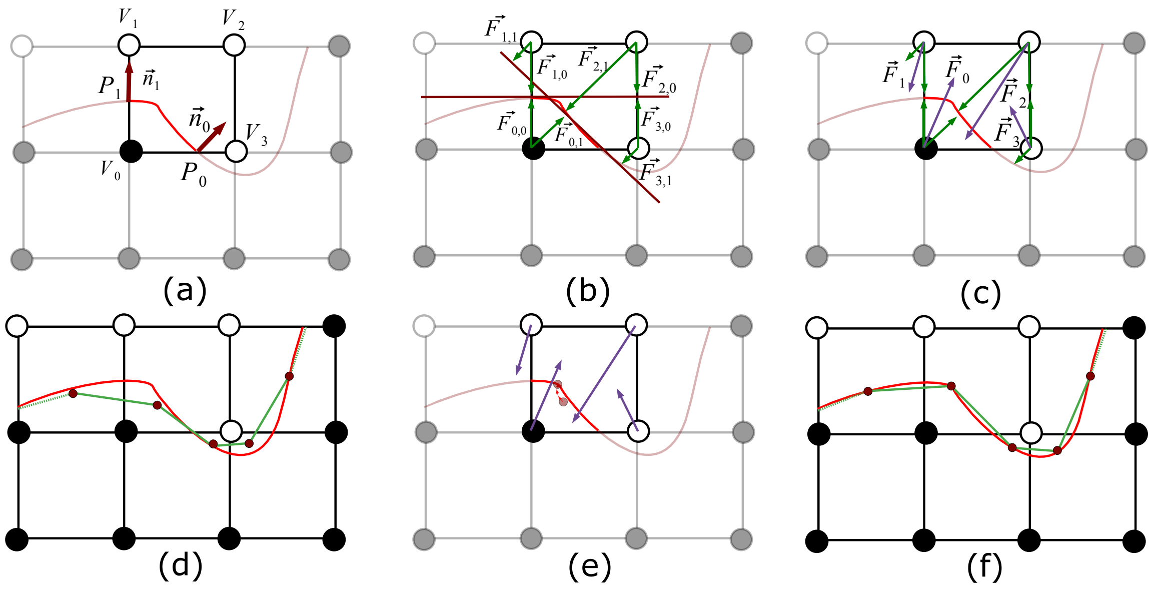


Figure 2. Particle-based approximation of features in Dual Contouring (2D example). (a): hermite data. (b): vectors (green arrows) from the vertices of the grid to the planes defined by the hermite data; (c): forces $\vec{F_k} = \vec{F_i} + \vec{F_{i+1}} + ...$ in the vertices of the grid; (d): initial line positioning using the mass point $\bar{C}$; (e): moving the particle towards the feature. The red arrows show the path the particle took. (f) final polygonization.

to. There are four candidate siblings $T_{-1,1}$ (upper-left), $T_{1,1}$ (upper-right), $T_{-1,-1}$ (lower-left) and $T_{1,-1}$ (lower-right) per level. They are chosen based on the intervals of their values ($V_{x,y}$). 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,

$$T_{-1,1} = [0; x), \tag{3}$$
$$T_{1,1} = [x; y), \tag{4}$$
$$T_{-1,-1} = [y; z), \tag{5}$$
$$T_{1,-1} = [z; w]. \tag{6}$$

At the first level $k' = k$. After that, k' is updated by subtracting the first interval endpoint. At the lower level the same process is done, and it goes on until the base texture is reached. 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 texel $T_{-1,-1} = [y = 7 + 3; z = 7 + 3 + 5)$ (lower-left pixel). After that, $k' = k - y$, where $y = 7 + 3$. When the pyramid base is reached, the stream expansion is finished. The remainder of k' represents the identifier from 0 to the sum of the voxel vertices. So k' is used with pre-specified topology, which in this case is a table (patterns of Marching Cubes) that identifies where the vertex is located.

V. Discussion and Results

We implemented our method with GLSL (OpenGL® Shading Language) running in a GeForce[1] 8800 GTX with 768 MB of memory and an AMD Athlon 64 Processor of 2.2 GHz and 2 GB of RAM (DDR1). The datasets used for our tests were *Backpack*, *Stent*, *Bonsai* and *Engine* [28].

Figure 5 shows that even with help of Span Space acceleration, all CPU-based polygonizers do not reach interactive framerates in any dataset. The CPU hardware is not adapted (yet) to take advantage of the independence between active cells, and the inherent parallelism of domain subdivision methods. Results regarding GPU approaches, however, expose the graphic hardware high capabilities, once the HistoPyramids-based implementation showed to be up to 1500 times faster than CPU ones, even when applied on relatively large datasets.

Such performance gain is reached in exchange for higher memory consumption. Maintaining large datasets as the backpack in GPU memory along with the data structure and other GPU structures (vertex buffer objects to save the polygons and to traverse the pyramid, for instance) is a challenging task. It becomes even harder due to the squared texture requirement, which increases quadratically and most of the times needs padding. For example, if a texture has dimensions 256^3, it needs a 4096^2 base texture, which fits perfectly. On the other hand, if one volume slice is included $256^3 + 256^2$, the base increases to 8192^2, wasting lots of memory with padding.

[1]Copyright ©2007 nVIDIA Corporation

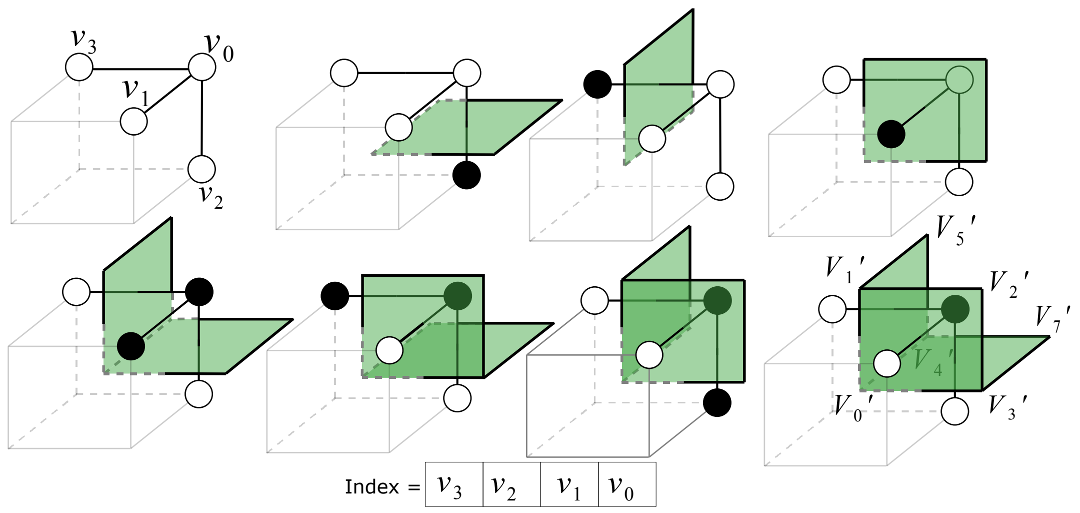
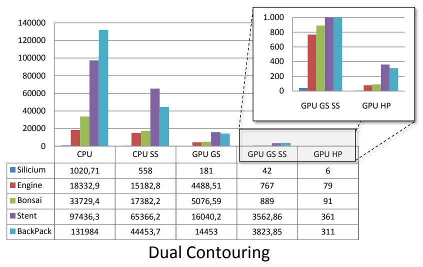


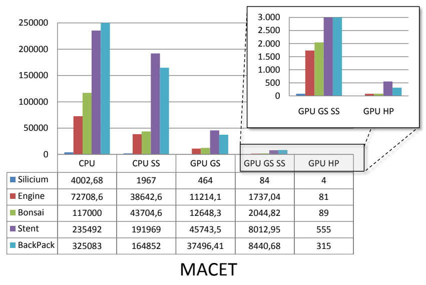
Figure 4. Table cases of Dual Contouring on uniform grids. The cases are based on scalar information contained in four vertices of the cell, which compose the highlighted edges of the figure. The scalar information indentifies the vertices as being inside (black) or outside (white) the isosurface, which defines active edges and, consequently, geometry. The vertex condition is coded into a bitwise combination $i = v_3 v_2 v_1 v_0$ (four vertices map into four bits). This index is used to identify the topology of the quad by one of the table cases, similarly to MC, but extrapolating geometry into neighbor cells.



	CPU	CPU SS	GPU GS	GPU GS SS	GPU HP
🟦 Silicium	1020,71	558	181	42	6
🟥 Engine	18332,9	15182,8	4488,51	767	79
🟩 Bonsai	33729,4	17382,2	5076,59	889	91
🟪 Stent	97436,3	65366,2	16040,2	3562,86	361
🟦 BackPack	131984	44453,7	14453	3823,85	311

Dual Contouring

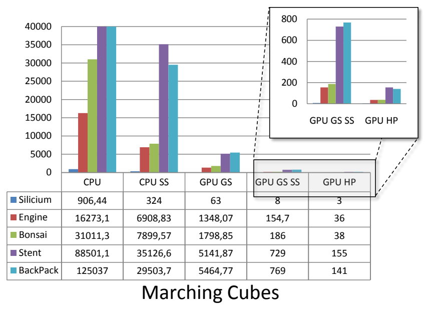
(a)



	CPU	CPU SS	GPU GS	GPU GS SS	GPU HP
🟦 Silicium	4002,68	1967	464	84	4
🟥 Engine	72708,6	38642,6	11214,1	1737,04	81
🟩 Bonsai	117000	43704,6	12648,3	2044,82	89
🟪 Stent	235492	191969	45743,5	8012,95	555
🟦 BackPack	325083	164852	37496,41	8440,68	315

MACET

(b)



	CPU	CPU SS	GPU GS	GPU GS SS	GPU HP
🟦 Silicium	906,44	324	63	8	3
🟥 Engine	16273,1	6908,83	1348,07	154,7	36
🟩 Bonsai	31011,3	7899,57	1798,85	186	38
🟪 Stent	88501,1	35126,6	5141,87	729	155
🟦 BackPack	125037	29503,7	5464,77	769	141

Marching Cubes

(c)

Figure 5. Extraction times (in miliseconds) from our GPU-based Dual Contouring and Macet compared to Marching Cubes.

It is interesting to observe in the Figure 5 and Table I that the bottleneck of HistoPyramids is related to the number of triangles, due to pyramids traversal, and not to the volume size, similarly to Span Space. This does not imply, however, that our implementation suffers with a large number of triangles, as shown in the Table I. Sparse isosurfaces, conversely, result in very fast polygonizations even with large datasets.

VI. Conclusions and Future Work

This paper discussed the contouring of high-quality isosurfaces at interactive framerates using GPU implementations of DC and Macet. Both approaches have different objectives, so each one has its own advantages and weaknesses. If an user of this polygonizers needs featured polygonization, the choice is Dual Contouring. If he needs to process the mesh afterwards, he might choose Macet. We discussed how to explore the parallel nature of graphics processors, while avoiding some of its constraints.

	Datasets			
	Backpack	Bonsai	Engine	Stent
X x Y x Z	512^2x256	256^3	256^3	512^2x174
λ (Isovalue)	1000.5	49.5	49.5	500.5
# Tri. (MC)	2338896	671296	599024	2823756
# Tri. (DC)	2342608	673048	598920	2826208
Av.Qlty(MC)	0.700012	0.687802	0.70733	0.71089
Av.Qlty(DC)	0.766	0.75725	0.77274	0.77473
Av.Qlty(Macet)	0.81369	0.82694	0.81674	0.80432
Min.Qlty(MC)	1.525e-5	1.373e-3	4.232e-4	1.709e-5
Min.Qlty(DC)	1.785e-5	1.138e-4	6.55e-3	1.502e-3
Min.Qlty(Macet)	0.44475	0.47395	0.51064	0.45069

Table I
Volume and isosurface extraction information. The isovalues and number of triangles are related to the extraction times in Figure 5. The average and minimum triangle quality is given by the radii ratio.

One of the goals of the original Dual Contouring, the mesh adaptivity by means of octree subdivision, was not explored in our work. It is used for simplification on the mesh regions with low curvature, which is an important tool to store large meshes. However, this simplification can still be done in the CPU and then transferred to GPU, by means of stored GPU meshes (Vertex Buffer Objects). In addition, there are limitations in the output size of Geometry Shader program not addressed in this paper. Our GS single rendering pass could be extended into multiple passes for efficiency improvement, such as in the implementation of Geiss [29].

VII. Acknowledgements

The work of Leonardo A. Schmitz and João L. D. Comba is sponsored by CNPq-Brazil grants (136569/2008-0, 485853/2007-8 and 569239/2008-7) and CAPES.

References

[1] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *ACM SIGGRAPH '87*, 1987, pp. 163–169.

[2] H.-W. Shen, C. D. Hansen, Y. Livnat, and C. R. Johnson, "Isosurfacing in span space with utmost efficiency (issue)," in *VIS '96*, 1996, pp. 287–ff.

[3] C. Dyken, G. Ziegler, C. Theobalt, and H. P. Seidel, "High-speed marching cubes using histopyramids," in *Computer Graphics Forum*, vol. 27, no. 8, Sep. 2008, pp. 2028–2039.

[4] C. A. Dietrich, L. P. Nedel, C. Scheidegger, J. Schreiner, C. T. Silva, and J. L. D. Comba, "Edge transformations for improving mesh quality of marching cubes," *IEEE Trans. on Vis. and Comp. Graph.*, 2009.

[5] C. Dietrich, C. Scheidegger, J. L. D. Comba, L. P. Nedel, and C. T. Silva, "Edge groups: An approach to understanding the mesh quality of marching methods," *14(6)*, vol. IEEE Trans. on Vis. and Comp. Graph., pp. 1651–1658, 2008.

[6] C. A. Dietrich, C. E. Scheidegger, J. L. Comba, L. P. Nedel, and C. T. Silva, "Marching cubes without skinny triangles," in *Comp. in Sci. & Eng.*, vol. 11, no. 2, 2009, pp. 82–87.

[7] T. Ju, F. Losasso, S. Schaefer, and J. Warren, "Dual contouring of hermite data," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 339–346, 2002.

[8] G. T. Herman and J. K. Udupa, "Display of 3d digital images: Computational foundations and medical applications," in *Medcomp '82*, 1982, pp. 308–314.

[9] C. L. Bajaj, V. Pascucci, and D. R. Schikore, "Fast isocontouring for improved interactivity," in *VVS '96: Proc. of the 1996 Symposium on Vol. Vis.*, 1996, pp. 39–ff.

[10] T.Lewiner, H.Lopes, A.W.Vieira, and G.Tavares, "Efficient implementation of marching cubes' cases with topological guarantees," *Journal of Graph. Tools*, vol. 8, no. 2, pp. 1–15, 2003.

[11] A. Lopes and K. Brodlie, "Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing," *IEEE Trans. on Vis. and Comp. Graph.*, vol. 9, no. 1, pp. 16–29, 2003.

[12] G. M. Nielson, "On marching cubes," *IEEE Trans. on Vis. and Comp. Graph.*, vol. 9, no. 3, pp. 283–297, 2003.

[13] S. F. F. Gibson, "Constrained elastic surface nets: Generating smooth surfaces from binary segmented data," in *MICCAI'98*, vol. 1496, 1998, p. 888.

[14] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel, "Feature sensitive surface extraction from volume data," in *ACM SIGGRAPH '01*, 2001, pp. 57–66.

[15] V. Pascucci, "Isosurface computation made simple: Hardware acceleration, adaptive refinement and tetrahedral stripping," in *IEEE TVCG VisSym*, 2004, pp. 293–300.

[16] F. Reck, C. Dachsbacher, R. Grosso, G. Greiner, and M. Stamminger, "Realtime isosurface extraction with graphics hardware," Eurographics Short Presentations, pp. 33–36, 2004.

[17] T. Klein, S. Stegmaier, and T. Ertl, "Hardware-accelerated reconstruction of polygonal isosurface representations on unstructured grids," *Comp. Graph. and Applications, 2004. PG 2004*, pp. 186–195, Oct. 2004.

[18] P. Kipfer and R. Westermann, "Gpu construction and transparent rendering of iso-surfaces," in *Proc. Vision, Modeling and Visualization*, 2005, pp. 241–248.

[19] J. Snoeyink, "Artifacts caused by simplicial subdivision," *IEEE Trans. on Vis. and Comp. Graph.*, vol. 12, no. 2, pp. 231–242, 2006.

[20] F. Goetz, T. Junklewitz, and G. Domik, "Real-time marching cubes on the vertex shader," *Eurographics Short Presentations*, August 2005.

[21] G. Johansson and H. Carr, "Accelerating marching cubes with graphics hardware," in *ACM CASCON '06*, 2006, p. 39.

[22] D. Blythe, "The direct3d 10 system," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 724–734, 2006.

[23] N. Tatarchuk, J. Shopf, and C. DeCoro, "Real-time isosurface extraction using the gpu programmable geometry pipeline," in *ACM SIGGRAPH 2007 courses*, 2007, pp. 122–137.

[24] S. Schaefer and J. Warren, "Dual contouring: The secret sauce," Dep. of Comp. Science, Rice University, Tech. Rep. 02-408, 2002.

[25] M. Harris, S. Sengupta, and J. D. Owens, "Parallel prefix sum (scan) with cuda," in *GPU Gems 3*. Addison Wesley, 2007.

[26] D. Horn, "Stream reduction operations for gpgpu applications," in *GPU Gems 2*. Addison Wesley, 2005, pp. 573–589.

[27] M. J. Harris, W. V. Baxter, T. Scheuermann, and A. Lastra, "Simulation of cloud dynamics on graphics hardware," in *HWWS '03*. Eurographics Association, 2003, pp. 92–101.

[28] Volvis. [Online]. Available: www.volvis.org

[29] R. Geiss, "Generating complex procedural terrains using the gpu," in *GPU Gems 3*. Addison Wesley, 2007.

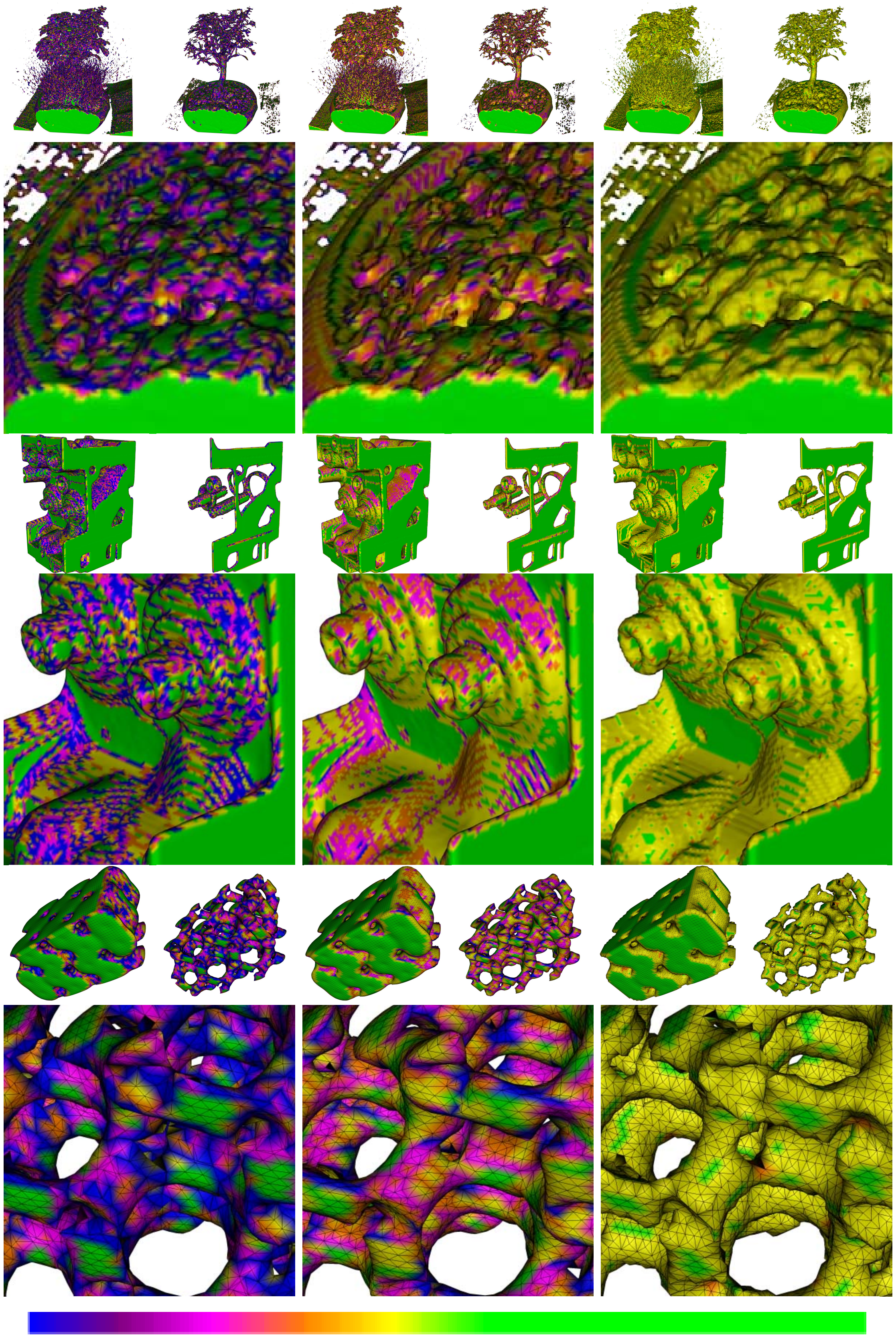


Figure 6. Interactive navigation through isosurfaces results and radii ratio coloring (green = good, blue = bad) triangle vertices. Bonsai (row 1-2), Engine (row 3-4) and Silicium (row 5-6) datasets. Marching Cubes (column 1), Dual Contouring (column 2) and Macet (column 3) polygonizations with two different isovalues.