Robust and Adaptive Surface Reconstruction using Partition of Unity Implicits

João Paulo Gois[1] Valdecir Polizelli-Junior[1] Tiago Etiene[1] Eduardo Tejada[2]
Antonio Castelo[1] Thomas Ertl[2]
Luis Gustavo Nonato[1]
[1]Universidade de São Paulo [2]Universität Stuttgart
{jpgois,valdecir,etiene,castelo,gnonato}@icmc.usp.br {tejada,ertl}@vis.uni-stuttgart.de

Abstract

Implicit surface reconstruction from unorganized point sets has been recently approached with methods based on multi-level partition of unity. We improve this approach by addressing local approximation robustness and iso-surface extraction issues. Our method relies on the J_1^A triangulation to perform both the spatial subdivision and the iso-surface extraction. We also make use of orthogonal polynomials to provide adaptive local approximations in which the degree of the polynomial can be adjusted to accurately reconstruct the surface locally. Finally, we compare our results with previous work to demonstrate the robustness of our method.

1 Introduction

Lately, the research on surface reconstruction from unorganized point sets has focused on methods based on implicit functions. Among them, methods based on partition of unity have been recently used due to their nice properties concerning processing time, reconstruction quality and capacity to deal with massive data sets [23, 21]. Basically, these methods consist in determining a domain subdivision for which local functions are computed and combined to define a continuous global implicit approximation for the point set.

In this work, we tackle important issues from previous methods based on partition of unity namely, the lack of robustness of the local approximations and the presence of spurious sheets and surface artifacts in the reconstructed model. In addition, we achieve adaptiveness not only by subdividing the domain, but also by employing local polynomial approximations whose degree can be recursively increased thanks to the use of orthogonal polynomials in two variables. Our method also attempts to profit from information gathered during the function approximation, regarding the features of the object, in order to condition an adaptive polygonization scheme by using an appropriate data struc-

Figure 1. Stanford Lucy (16M Points) reconstructed with the proposed method.

ture suited for both phases. An example of a surface reconstructed with our method is depicted in Figure 1.

1.1 Related work

Implicit surface reconstruction. Hoppe et al. [13] proposed a method to reconstruct surfaces from point clouds using local approximations, which motivated many other works. The method is based upon planar approximations and it does not ensure continuity of the function.

Multi-level partition of unity implicits were proposed by Ohtake et al. [23]. To define the supports of the partition of unity, the domain is decomposed using an octree. The reconstructed surface mesh is then obtained from a regular grid (resampled from the octree) using Bloomenthal's polygonizer [5]. Mederos et al. [21] also present an approach based on partition of unity that uses the *gradient one fitting* method to avoid discontinuities and to reduce the sensitivity of the approach to small perturbations. However, this method must solve systems of equations with matrices for which maximal rank cannot be ensured or are ill-conditioned. Thus, expensive techniques, specifically *ridge regression*, are used to improve stability. A similar function estimation approach was presented by Lage et al. [17] to approximate vector fields.

Approaches based on radial basis functions have been proposed [6, 24] and are based on solving one large, dense and ill-conditioned linear system (if ideal radial basis functions are required). Contrarily, methods based on moving least-squares [1, 2, 16] produce local approximations by solving a large amount of small systems of equations to fully approximate the surface. The main drawback of these methods is the narrowness of the surface domain [2]. More recently, Kazhdan et al. [14] proposed an interesting approach in which the surface approximation is formulated as a Poisson problem. This method presents several advantages over other formulations, but its processing time is higher than the one needed by partition of unity and moving least-squares formulations. Also, such method does not achieve topological control.

Iso-surface extraction. Since the introduction of the classic marching cubes algorithm [20], several works have addressed issues concerning quality and topological properties of the meshes generated by means of regular cuberille grids. It is a fact that the classic marching cubes suffers from ambiguities in its lookup table and, hence, may generate topologically incorrect meshes. This issue has been approached by the extension of marching cubes cases [22, 9, 18], by the use of tetrahedral elements as basic domain subdivision elements [26, 8, 28] or by a combination of cubic and tetrahedral cells [5].

Concerning the quality of the generated triangles, the work by Figueiredo et al. [10] employs a physically based approach to obtain high quality triangles. Also, an advancing front algorithm was proposed recently for creating iso-surfaces from regular and irregular volumetric datasets [27].

The marching cubes algorithm is also not able to represent sharp features (edges or vertices) which are not aligned with the regular grid, therefore, Kobbelt et al. [15] proposed a modified marching cubes that calculates points inside the cube in order to better adjust the surface to sharp features.

Another desirable feature in a polygonizer is adaptiveness, in that surfaces can present both high and low-curvature regions which require different resolutions for a good approximation. One of the earliest works that deal with this issue uses recursive subdivision of tetrahedra [12]. Recently, Paiva et al. [25] employed octree subdivision in an adaptive triangulation algorithm whereas Castelo et al [7] defined an adaptive triangulation, named J_1^A , for the same purpose. One of the main features presented by J_1^A is its capacity to be extended to any dimensions due to its well-defined algebraically description.

1.2 Main contributions

We propose a surface reconstruction method by effectively combining the J_1^A triangulation with an adaptive construction of local polynomial approximations by means of multivariate orthogonal polynomials [4, 3]. This allows us to increase the degree of the polynomial at locations where higher-degree polynomials are needed to obtain a good approximation to the surface. Following, we present some contributions of our work:

Adaptive implicit surface reconstruction with topological guarantees: contrary to previous methods, we extract the iso-surface directly from the data structure used to subdivide the space; namely, the J_1^A . This enables the adaptive surface extraction to take advantage of refinement information obtained during function approximation. Furthermore, as J_1^A is composed of tetrahedra, the surface extraction algorithm guarantees topologically coherent surfaces.

Numerical stability: in general, least-squares formulations solved by normal equations using canonical polynomials lead to ill-conditioned systems of equations. By using a basis of orthogonal polynomials we avoid solving systems of equations and improve stability without requiring expensive computations in opposition to other methods such as QR decomposition with Householder factorization, singular value decomposition or pre-conditioned conjugate gradient.

Adaptive local function approximation: the use of orthogonal polynomials allows us to efficiently increase the degree of the local polynomial approximation due to the recursive nature of their construction. For that reason, our method is also adaptive with respect to local fittings.

Avoiding spurious surface sheets: contrary to previous work on multi-level partition of unity implicits, our method is able to avoid generating spurious surface sheets and surface artifacts. This is achieved by using some tests that dis-

card approximations considered non-robust, i.e., approximations which oscillate within the local support.

2 Background

In this section we describe the multi-level partition of unity implicits in order to set the basis to present our method. We also describe the J_1^A data structure to better explain its properties used to define our adaptive partition of unity implicits.

2.1 Multi-level partition of unity implicits

Partition of unity implicits are defined on a finite domain Ω as a global approximation $\mathcal{F}$ obtained with a linear sum of local approximations. As with other implicit surface approximation methods, the surface is defined as the zero set of $\mathcal{F}$. For this purpose, a set of non-negative compactly supported weight functions $\Phi = \{\phi_1, \ldots, \phi_n\}$, where $\sum_{i=0}^{n} \phi_i(\mathbf{x}) \equiv 1, \quad \mathbf{x} \in \Omega$, and a set $\mathbb{F} = \{f_1, \ldots, f_n\}$ of local signed distance functions f_i must be defined on Ω. Given the set $\mathbb{F}$ and Φ, the function $\mathcal{F} : \mathbb{R}^3 \rightarrow \mathbb{R}$ is defined as:

$$\mathcal{F}(\mathbf{x}) \equiv \sum_{i=0}^{n} f_i(\mathbf{x})\phi_i(\mathbf{x}), \quad \mathbf{x} \in \Omega. \tag{1}$$

A set of nonnegative compact support functions can produce the partition of unity functions as

$$\phi_i(\mathbf{x}) = \frac{\theta_i(\mathbf{x})}{\sum_{k=1}^{n} \theta_k(\mathbf{x})},$$

where θ_i is a compactly supported weight function. Figure 2 depicts a two dimensional example: the domain Ω is covered by a set of circles – supports– and, for each one, a function f_i and a weight function ϕ_i are defined. Otahke et al. subdivide the domain using an octree and define a spherical support for each cube. The functions $f_i : \mathbb{R}^3 \rightarrow \mathbb{R}$ at each local support are computed using the set of points by initially defining a local coordinate system (ξ, η, ν) at the center of the support, where (ξ, η) define the local plane (domain), and ν coincides with the orthogonal direction (image). Hence, f_i is defined as $f_i(\mathbf{x}) = w - g_i(u, v)$, where (u, v, w) is $\mathbf{x}$ in the (ξ, η, ν) basis. The function g_i is obtained by the two-dimensional least-squares method. Note that this method requires points equipped consistently with oriented normal vectors.

2.2 The J_1^A triangulation

The J_1^A triangulation [7] is an algebraically defined structure that can be constructed in any dimension and is able to handle refinements to accommodate local features. Among J_1^A 's advantages, we can highlight the mechanism for representing simplices and the existence of algebraically rules for traversing the triangulation which prevents storing connectivity and, therefore, enables efficient storage.

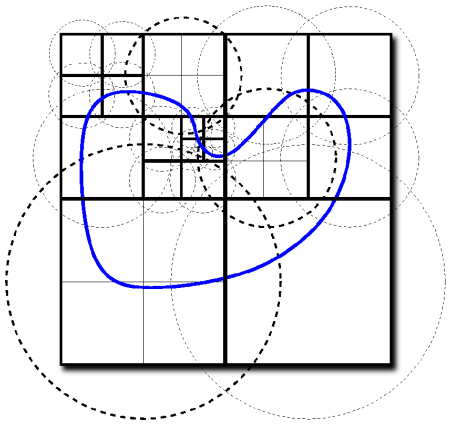

Figure 2. The behavior of the J_1^A concerning the refinements induced by the error criterion and by J_1^A restrictions.

Basically, the J_1^A triangulation consists of a grid in which the basic unit in $\mathbb{R}^n$ is a n-dimensional hypercube, which we will refer to as block. These units are divided into $2^n n!$ n-simplices that can be described algebraically using six values $S = (g, r, \pi, s, t, h)$. The first two elements of S define in which block the simplex is contained. The n-dimensional vector g locates a particular block in a particular refinement level r. Figure 3 illustrates, on the left, a two-dimensional J_1^A grid and, on the right, a highlighted block of refinement level $r = 0$ (0-block) and $g = (3, 1)$. Also in Figure 3, one can notice that dark gray blocks present refinement level $r = 1$ – thus are called 1-blocks – and, for that reason, are part of a higher resolution grid.

Once a block can be located in the grid, there must be a way to specify which simplex we are referring to. However, before doing so, it is necessary to observe that J_1^A handles refinements – in $\mathbb{R}^n$ – by splitting one block into 2^n blocks and applying local changes on the grid in order to accommodate newly created blocks. This accommodation process originates a different kind of block: the transition block. The J_1^A triangulation prohibits situations in which neighboring blocks present a bigger than one difference in their refinement levels by propagating refinements. It also imposes that, whenever there are neighbor blocks whose refinement levels differ by one, the one possessing smaller r is transformed into a transition block.

A transition block is a block that possesses only some of its k-dimensional faces $(0 < k < n)$ refined. This is better illustrated by Figure 3 in which basic 0-blocks are colored white, basic 1-blocks are colored dark-grey and transition blocks are colored light-grey. In particular, the highlighted transition block has only its upper edge refined. To maintain notation coherence, non-transition blocks will be referred to as basic blocks from now on.

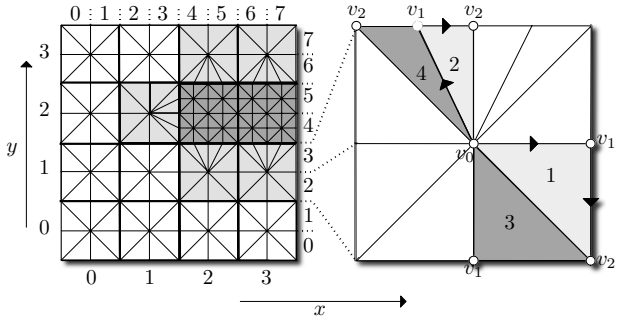

Figure 3. An example of a 2D J_1^A triangulation (left) and the detail of block $g = (3, 1)$, $r = 0$ and two paths for tracing simplices (right).

The representation of all simplices inside a block is based upon the fact that all of them share at least one vertex: v_0. Starting in v_0, which is the center of a n-dimensional hypercube, the next step is taken in the positive or negative direction of one chosen coordinate axis. This will produce v_1 as the center of a $(n-1)$-dimension face and, as the process goes on, the vertices $v_2 \dots v_n$ will be defined as the center of $(n-2), \dots, 0$ dimensional faces respectively. In other words, simplices can be represented by the path traversed from v_0 to v_n which is coded by π and s. The π vector stores a permutation of n integers from 1 to n representing coordinate axes, while s represents the direction – positive or negative – that must be followed in each axis. For instance, in Figure 3, simplex 1 is represented by $\pi = (1, 2)$ and $s = (1, -1)$, which means that first we go through axis $\pi_1 = 1$ (x, in the figure) in direction $s_{\pi_1} = 1$ and then through axis $\pi_2 = 2$ (y, in the figure) in direction $s_{\pi_2} = -1$.

In the case of simplices inside basic blocks and simplices inside transition blocks that do not reach any refined face, the information provided by π and s is enough. However, in the remaining cases, further information must be taken into account, because when a refined k-dimensional face is reached, there is not only one center, but 2^k centers. For this reason, the scalar h is defined to inform how many steps are taken before a refined face is reached, while vector t defines extra signs for axis $\pi_{h+1} \dots \pi_n$ that are used for selecting one center from all possibilities. For instance, in Figure 3, simplex 2 is represented by $\pi = (1, 0)$, $s = (1, 1)$, $h = 1$ and $t = (-1, 0)$ because only one step is taken before reaching a refined edge and the chosen center for placing v_1 is in the negative direction of π_{h+1}. The following expressions formally describe how to compose a simplex inside a hypercube centered at the origin $(0, \dots, 0)$ with edges of length 2:

$$\begin{cases} v_0 = (0, \dots, 0) \\ v_i = v_{i-1} + e_{\pi_i} s_{\pi_i}, \text{for } 1 \le i < h \\ v_h = v_{h-1} + e_{\pi_h} s_{\pi_h} + \frac{1}{2} \sum_{k=h+1}^{n} e_{\pi_k} t_{\pi_k} \\ v_i = v_{i-1} + \frac{1}{2} e_{\pi_i} s_{\pi_i}, \text{for } h < i \le n \end{cases}, \quad (2)$$

where e_i is a vector with value 1 in position i and 0 in the remaining positions.

Besides describing simplices algebraically, the J_1^A triangulation also defines pivoting rules for traversing the triangulation without using any other topological data structure. Figure 3 illustrates two pivoting operations in which simplices 1 and 2 are pivoted in relation to vertices v_1 and v_2 respectively, generating simplices 3 and 4. All pivoting rules can be found in the work by Castelo et al. [7].

3 Robust adaptive partition of unity implicits

Traditionally, an adaptive partition of unity implicit is built using an octree to partition the space and calculate local approximations that are subsequently combined using weights. The more details the object possesses, the more refined the octree must be. Thus, the octree can be used to identify complicated or soft features on the surface. Hence, our goal was to use this information acquired during function approximation to obtain an adaptive polygonization. Therefore, as J_1^A has an underlying restricted octree as its backbone, we adaptated the triangulation to serve both approximation and polygonization purposes. The achieved adaptiveness allows capturing fine details without using refined grids. Another important feature of our method is the increased quality of the local approximations compared to previous methods, which prevents spurious sheets and surface artifacts.

3.1 Our method at a glance

The method works in two phases: the first generates the implicit function and the second the polygonization.

To generate the implicit function, we start by subdividing the domain using the J_1^A triangulation. Given a block in the J_1^A, a local approximation is generated to fit the data inside a support radius surrounding this block. If the error criterion is not met, the block is refined by subdividing it into eight new blocks and, consequently, new approximations are calculated for each one of them. This process is repeated until all blocks satisfy some error and robustness conditions or the maximal pre-defined depth is reached. In the latter case the approximation is used anyway.

After the approximation process is completed, we have the J_1^A subdivision of the domain, and therefore the supports and the local approximations for each support. Hence, the implicit function can be evaluated and the iso-surface extraction is performed by simply visiting all transversal simplices by means of pivoting rules and generating the triangular mesh.

3.2 Local approximations

We compute a local approximation at each support by means of a polynomial fitting using least-squares. However, instead of using a canonical basis $\{u^i v^j : i, j \in \mathbb{N}\}$, we opt for a basis of orthogonal polynomials with respect to the inner product induced by the normal equation. This way, we do not need to solve any system of equations, avoiding ill-conditioned matrices. To find this basis of orthogonal polynomials we use the method by Bartels and Jezioranski [4, 3]. The authors argue that the computation is faster and more stable than using numerical methods to solve the systems of equations.

Given a set of orthogonal polynomials $\Psi = \{\psi_1, \ldots, \psi_l\}$, we can compute the polynomial in local coordinates as

$$g_i(u,v) = \sum_{\psi_j \in \Psi} \psi_l(u,v) \frac{\sum_{i=1}^{m} w_i \psi_j(u_i, v_i)}{\sum_{i=1}^{m} \psi_j(u_i, v_i)\psi_j(u_i, v_i)}, \quad (3)$$

where g_i is the function g that minimizes

$$\min_{g} \sum_{(u_i, v_i)} (g(u_i, v_i) - w_i))^2. \quad (4)$$

An important advantage of using orthogonal polynomials is the ability to generate higher-degree approximations from previously computed low-degree approximations with low additional computational effort. This motivated us to use orthogonal polynomials to define a method that is adaptive both in the spatial subdivision and in the local approximation.

To explain the approximation algorithm we can take an arbitrary block to be processed, for which the support is defined as its circumsphere enlarged by a factor larger than one (in our implementation we use 1.5). The approximation process starts by finding all points inside the support of the block. At this point, we face three different situations: (i) the number of points in the support is enough to calculate the approximation properly; (ii) the number of points is not enough ; (iii) there are too many points in the support.

In case (i), we approximate the point set by a least-squares plane defined in a local coordinate system. If the error criterion is not satisfied, but there are enough points to perform a higher degree least-squares, we recursively compute a polynomial of one degree greater than the current and recompute the error. This process is repeated until the error criterion is met, until there are not enough points to increase the degree of the polynomial, until a covering criterion (explained below) is not met or until a maximal degree is reached (degree four in our implementation). In the first case, the approximation is stored in the block and the process stops, otherwise subdivision is performed unless the block possesses a critical amount of points (in our case, we test blocks with fewer than 100). In this situation, we test whether possible new approximation blocks would increase the error instead of decreasing it due to the fact that high degree approximations could not be achieved for them. If they do increase, the subdivision of the block is aborted.

The error criterion is trivial and consists in checking if the relative average least-squares error is larger than a user-specified threshold. The covering test is more complex and is based upon the fact that when dealing with high degree polynomials, a special care must be taken to avoid blocks with functions that, even though presenting small least-square error, are actually bad approximations inside the support radius. For instance, in Figure 4, although point p lies inside the support, the function does not offer a good approximation on it. This problem occurs because high degree polynomials are able to approximate point data nicely in-between points, however, they can also significantly oscillate at locations with no points to restrict the approximation, as illustrated on left side of Figure 4. This fact means that, depending on how points are distributed inside the support, the associated approximations may lead to artifacts and spurious sheets; therefore, the problem may be reduced to determining when a particular point distribution does not guarantee robustness. To analyze this question, we must take into account the fact that our approximations are calculated and evaluated in a local coordinate system and, hence, our domain is actually on a plane. Thus, we have to determine for which area of this plane the approximation *must* be robust, named support domain (A_r), and for which area the approximation *can* be robust, named coverage domain (A_c). The former is obtained by projecting the spherical support on the plane whereas the latter is obtained by projecting the local points' bounding box on the plane. Our criterion simply consists in calculating the covering ratio $k = A_c/A_r$ (k lies between 0 and $4/\pi$) and checking if k is enough for a given polynomial degree; i.e., if it surpasses a specified threshold. In our implementation we use 0.00, 0.85, 0.90 and 0.95 for polynomials with degree 1, 2, 3 and 4 respectively due to the fact that high degree polynomials tend to have a more oscillating behavior outside the coverage domain and thus require a tighter ratio. Figure 4, on the right, represents a two-dimensional case in which the coverage domain is not enough to guarantee a good approximation.

Approximation case (ii) is handled in a different way than previous methods in order to achieve greater robustness. Instead of iteratively growing the coverage volume of the block until the minimal number of points is reached [23], which may cause a local approximation to influence a large part of the domain, or using the approximation of the parent of the block [21], which could be a poor approximation, we address this case by observing that approximations in blocks that do not contain points must be only good enough to avoid or reduce undesirable arti-

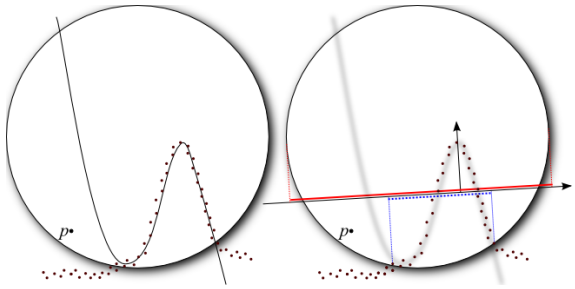

Figure 4. The coverage domain (shown in dashed blue line) is too small compared to the support domain (shown in solid red line).

facts. The solution consists in finding the nearest sample point $\mathbf{r}$ to the center of the block, querying a small number of neighbors of $\mathbf{r}$ (in our implementation we used 23 points) and approximating a least-square plane. An issue about least-squares planes is that, for some point distributions, the plane can be orthogonal in relation to what we were expecting [2]. Therefore, we detect this situation, by comparing the calculated plane normal to the mean of the normals of the neighbors of $\mathbf{r}$. If the normals differ by an angle greater than $\pi/6$, we discard the least-squares approximation and use the plane with normal equal to the mean of the normal vectors of the neighbors and origin equal to the mean of the positions of the neighbors.

Finally, case (iii) can be thought of as an heuristic employed to avoid useless and expensive calculations. In our case, we consider fitting polynomials to more than 1000 points an unnecessary effort; therefore, we force subdivision of the block. If the subdivision is not possible, due to maximal refinement level, the approximation is computed in the current block with the large number of points.

Before concluding this section, it is important to clarify the difference between block splitting caused by approximation conditions and those triggered by J_1^A restrictions (explained in Section 2.2). In the latter case, new approximations are not computed because the approximation for the block being refined could be already good. Figure 2 illustrates a case in which not all leaf nodes possess approximations associated to them. In the figure, light dashed circles represent supports associated to leaf nodes containing approximations whereas dark dashed circles correspond to supports associated to non-leaf nodes which were only subdivided due to J_1^A restrictions.

3.3 Function evaluation and polygonization

The function evaluation process consists in determining which blocks affect, with their supports, a particular point $\mathbf{x}$ and obtaining the value of $\mathcal{F}(\mathbf{x})$ as a combination of all local functions:

$$\mathcal{F}(\mathbf{x}) = \frac{\sum_{i=1}^{N} f_i(\mathbf{x})\theta_i(\|\mathbf{x}-\mathbf{c}_i\|/R_i)}{\sum_{i=1}^{N} \theta_i(\|\mathbf{x}-\mathbf{c}_i\|/R_i)},$$

where θ_i is a cubic spline [19], N is the number of blocks that affect $\mathbf{x}$, and $f_i(\mathbf{x})$, $\mathbf{c}_i$ and R_i are the local signed functions, center and radius of the block i respectively.

The problem of finding which blocks affect a particular point consists in an J_1^A octree-like traversal that prunes blocks whose influence volume does not encapsulate $\mathbf{x}$.

The surface polygonization from the implicit function is performed using the J_1^A defined in the approximation step. The algorithm starts by finding an initial simplex, which is straightforward because we can determine which simplex contains a particular input point, and then by traversing all transversal simplices using pivoting rules while generating the surface mesh.

4 Results and discussion

An interesting feature about our method, presented in Figure 5, is the double adaptiveness achieved by levels of refinement and degrees of polynomials. On the left, we can notice that greater refinement levels are found in more complicated regions of the model, for instance, the legs and the head. On the right, we have the distribution of polynomial degrees along the model and it is clear that high degree polynomials are employed to represent big areas with low refinement level, i.e., the refinement was avoided due to the approximation power presented by these functions. In the legs of the ant, we can verify the presence of many planes which was mostly due to the fact that higher degree polynomials could not meet the robustness criterion in that particular region, therefore, refinements were made and low degree approximations were used.

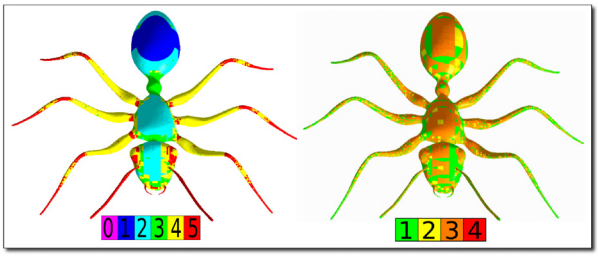

Figure 5. The EtiAnt Model [11]. On the left, the color shows the degree of refinement whereas, on the right, it shows the degree of the polynomial used.

One of the main issues we approached in this work was the robustness of the generated models. In Figure 6, we compared our method with Ohtake's by using similar configuration parameters and we can notice that our method

was able to generate more robust approximations. In this model [11], we also observed, in preliminary testing, that without our robustness criterion, even second degree polynomials could generate spurious sheets and artifacts. It is important to mention that, due to the fact that we do not implement any special treatment for representing sharp features, Ohtake's method presents better defined sharp edges for this model.

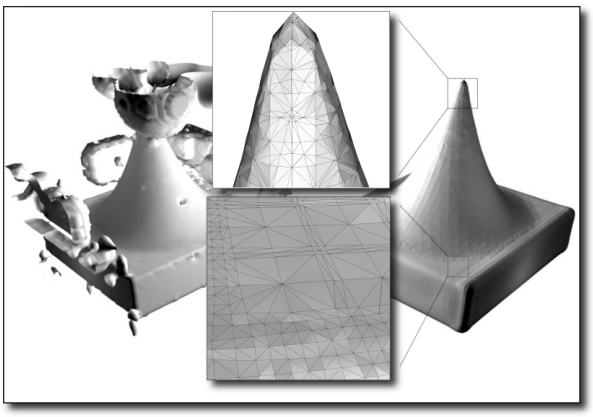

Figure 6. Comparison between Ohtake's method (on the left) and ours (on the right) using the Witchhat model (25K points). Details of the mesh can be seen in the zoom-in.

Regarding the generated mesh, the adaptiveness of our mesh can be verified in the zoom-in window in Figure 6. Also, due to our underlying tetrahedral mesh, the reconstructed surface is topologically correct, even though a direct counterpart of tetrahedron-based polygonizers is the presence of poor-quality triangles.

We tested our method in a Athlon X2 2.2Ghz with 3 GB RAM and we were able to generate a good approximation for the 398K points Dragon model in $299.3s$. For the same model, it took $374.8s$ to evaluate 1,000,000 points, which is equivalent to the number of evaluations required for polygonizing completely a regular grid of resolution $100 \times 100 \times 100$. We acknowledge that our method is still computationally less efficient than Ohtake's, even though our least-squares using orthogonal polynomials is faster than the one based on a canonical basis. That can be explained by the fact that our method reconstructs the function for the whole domain, whereas Ohtake's builds approximations only around the surface because its polygonization is performed for one connected component and local approximations are calculated only when needed. This fact may be seen as a disadvantage, but actually our method provides a more robust approximation for the complete domain, which means we can perform implicit function modeling operations without generating unexpected results, such as spurious sheets or artifacts. In addition, we believe that we may be able to enhance the efficiency of our method by optimizing the spatial queries by combining it with the subdivision performed over the domain.

Finally, in Figure 7 we present some models reconstructed by our method.

5 Conclusion

In this work we presented an adaptive multi-level partition of unity implicit scheme which employs the J_1^A triangulation not only for partitioning the domain, but also for polygonizing the surface. In addition, we introduced the use of multi-variate orthogonal polynomials to define local surface fittings as well as some heuristics to achieve robust approximations. Although our processing time for evaluating the function is currently higher than previous work, the presented results show that we are able to achieve a good level of robustness and, therefore, promising reconstruction quality.

As future work, besides improving our implementation, we intend to thoroughly study the domain of partition of unity methods as it was already done by Kil and Amenta [2] for the domain of Point Set Surfaces. Finally, we can notice that the quality of the meshes produced by the J_1^A polygonizer is not very attractive, therefore it is also important to address this issue and provide a post-processing step or an iso-surface extraction method capable of producing higher quality meshes based on the J_1^A .

Acknowledgments

This work was partially supported by DAAD, with grant number A/04/08711, by FAPESP, with grant numbers 04/10947-6, 06/54477-9 and 05/57735-6 and by Capes/DAAD, with grant number 262/07.

References

[1] A. Adamson and M. Alexa. Approximating and intersecting surfaces from points. In *Proc. of Eurographics/ACM Symposium on Geometry Processing*, pages 230–239. Eurographics Assoc., 2003.

[2] N. Amenta and Y. J. Kil. The domain of a point set surfaces. *Eurographics Symposium on Point-based Graphics*, 1(1):139–147, June 2004.

[3] R. H. Bartels and J. J. Jezioranski. Algorithm 634: Constr and eval: routines for fitting multinomials in a least-squares sense. *ACM Trans. Math. Softw.*, 11(3):218–228, 1985.

[4] R. H. Bartels and J. J. Jezioranski. Least-squares fitting using orthogonal multinomials. *ACM Transactions on Mathematical Software*, 11(3):201–217, 1985.

[5] J. Bloomenthal. An implicit surface polygonizer. In P. Heckbert, editor, *Graphics Gems IV*, pages 324–349. Academic Press, Boston, 1994.

[6] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH '01*, pages 67–76, 2001.

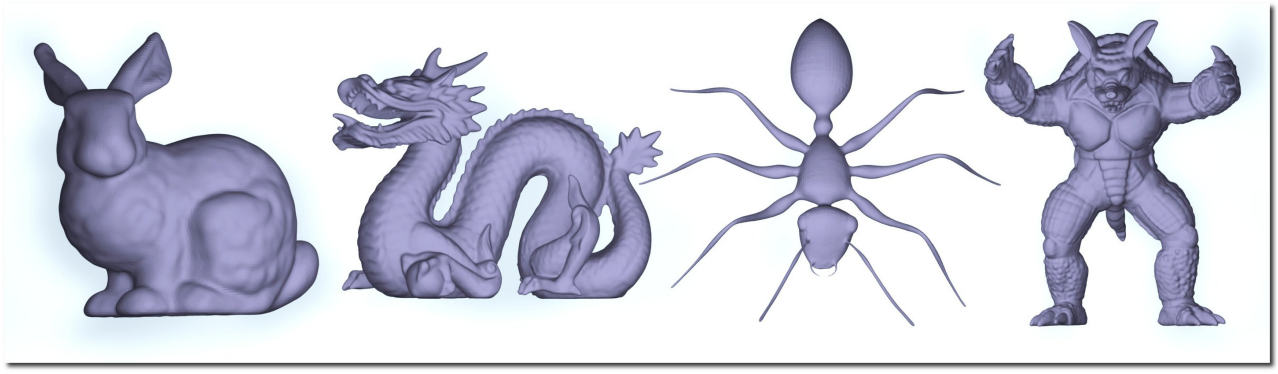

Figure 7. Reconstructed models. From left to right: Stanford Bunny (34K points), Stanford Dragon (398K points), EtiAnt (261K points), Stanford Armadillo Man (172K points).

[7] A. Castelo, L. G. Nonato, M. Siqueira, R. Minghim, and G. Tavares. The j1a triangulation: An adaptive triangulation in any dimension. *Computer & Graphics*, 30(5):737–753, 2006.

[8] S. L. Chan and E. O. Purisima. A new tetrahedral tesselation scheme for isosurface generation. *Computer & Graphics*, 22(1):83–90, 1998.

[9] E. V. Chernyaev. Marching cubes 33: construction of topologically correct isosurfaces. Technical report, CERN, 1995.

[10] L. H. de Figueiredo, J. M. Gomes, D. Terzopoulos, and L. Velho. Physically-based methods for polygonization of implicit surfaces. In *Proc. of the conference on Graphics interface '92*, pages 250–257, 1992.

[11] J. P. Gois. http://www.lcad.icmc.usp.br/∼gois/models. july/2007.

[12] M. Hall and J. Warren. Adaptive polygonalization of implicitly defined surfaces. *Computer Graphics and Applications, IEEE*, 10(6):33–42, Nov. 1990.

[13] H. Hoppe, T. DeRose, T. Duchampy, J. McDonaldz, and W. Stuetzlez. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2):71–78, 1992.

[14] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proceedings of Eurographics Symposium on Geometry Processing*, 2006.

[15] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. Feature sensitive surface extraction from volume data. In *SIGGRAPH '01*, pages 57–66, USA, 2001. ACM Press.

[16] R. Kolluri. Provably good moving least squares. In *SODA '05: Proc. of the 16th annual ACM-SIAM symposium on Discrete algorithms*, pages 1008–1017, 2005.

[17] M. Lage, F. Petronetto, A. Paiva, H. Lopes, T. Lewiner, and G. Tavares. Vector field reconstruction from sparse samples with applications. In *SIBGRAPI*, pages 297–306, Brazil, 2006. IEEE CS.

[18] T. Lewiner, H. Lopes, A. W. Viera, and G. Tavaresi. Efficient implement of marching cubes cases with topological guarantees. *Journal of Graphics Tools.*, 8(2):1–15, 2003.

[19] G. R. Liu and M. B. Liu. *Smoothed Particle Hydrodynamics*. World Scientific, 2003.

[20] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87*, pages 163–169. ACM Press, 1987.

[21] B. Mederos, S. Arouca, M. Lage, H. Lopes, and L. Velho. Improved partition of unity implicit surface reconstruction. Technical Report TR-0406, IMPA, Brazil, 2006.

[22] C. Montani, R. Scateni, and R. Scopigno. A modified look-up table for implicit disambiguation of marching cubes. *The Visual Computer*, 10(6):353–355, December 1994.

[23] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. *ACM Trans. Graph.*, 22(3):463–470, 2003.

[24] Y. Ohtake, A. Belyaev, and H.-P. Seidel. 3d scattered data interpolation and approximation with multilevel compactly supported RBFs. *Graphical Models*, 67(3):150–165, 2004.

[25] A. Paiva, H. Lopes, T. Lewiner, and L. H. de Figueiredo. Robust adaptive meshes for implicit surfaces. In *Proceedings of SIBGRAPI*, 2006.

[26] P. A. Payne and A. W. Toga. Surface mapping brain function on 3d models. *IEEE Computer Graphics and Applications*, 10(5):33–41, 1990.

[27] J. Schreiner, C. Scheidegger, and C. Silva. High-quality extraction of isosurfaces from regular and irregular grids. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1205–1212, 2006.

[28] G. M. Treece, R. W. Prager, and A. H. Gee. Regularised marching tetrahedra: improved iso-surface extraction. *Computer & Graphics*, 23(4):583–598, 1999.