Adaptive Multi-chart and Multiresolution Mesh Representation

Andre Maximo[a], Luiz Velho[a], Marcelo Siqueira[b]

[a] *Institute of Pure and Applied Mathematics, Jardim Botanico, RJ 22460-320, Brazil, email: {andmax,lvelho}@impa.br*
[b] *Federal University of Rio Grande do Norte, Lagoa Nova, RN 59078-970, Brazil, email: mfsiqueira@dimap.ufrn.br*

Abstract

In this paper, we present an adaptive multi-chart and multiresolution mesh representation suitable for both the CPU and the GPU. We build our representation by simplifying a dense-polygon mesh to a base mesh and storing the original geometry in an atlas structure. For both simplification and resolution control, we extend a hierarchical method based on stellar operators to the GPU context. During simplification, we compute local parametrizations to generate charts and an atlas structure to be used later in multiresolution management. Unlike previous approaches, we employ the simplified mesh as our base domain in a novel atlas descriptor combined with a specialized halfedge data structure, achieving superior geometric accuracy while adding a low additional storage. Finally, we show that our mesh representation can be used to adaptively control the mesh resolution in the CPU and the GPU at the same time in a broad range of applications, from mesh editing to rendering.

Keywords:
Computational Manifolds, Geometric Representation, Triangle Meshes

1. Introduction

Polygonal meshes have become the de facto standard representation for surfaces in 3D graphics applications [1]. A surface representation based on triangle meshes constitutes the foundation of common rendering techniques, e.g. rasterization and ray-tracing, while quadrilateral (or *quad*) meshes are more appropriate for CAD modeling, texture-mapping and numerical simulations. In this work we are interested in both representations and how to combine them with an adaptive multiresolution atlas structure.

Dense-polygon meshes can be a burden to edit or render in complex scenes. To mitigate this problem, graphics applications often employ a multiresolution representation of the surface based on different strategies, as reviewed in section 2. One direct and efficient approach is to solve the problem in two steps. First, the original dense mesh is simplified in a hierarchical fashion, while either storing the simplification sequence, the method known as *Progressive Meshes* (PM) [2], or incrementally inducing local parametrizations of the surface on each simplification step, another well-known method called *Multiresolution Adaptive Parameterization of Surfaces* (MAPS) [3]. Second, the local parametrizations can be modified to fit parametric subdivision patches and to obtain the multiresolution representation, as done in MAPS using *Loop subdivision* [4]. The result is a smooth mesh with controllable adaptive resolution that approximates, as the resolution increases on each region, the original dense mesh.

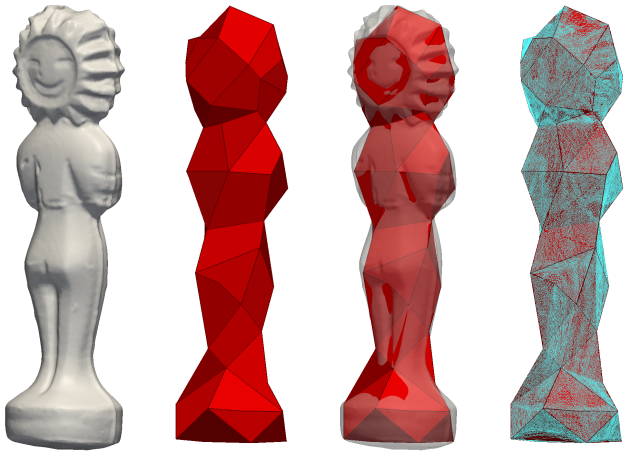
Although an effective solution, the traditional multiresolution representation has a few shortcomings. One is the memory consumption when storing the simplification sequence in order to restore the original mesh, as done by PM. Another is the disconnection between simplification and reconstruction, as done by MAPS; in particular, the operations carried out in the second step are not the direct inverse of the ones performed in the first step. These two correlated problems hinder the solution inadequate to act cooperatively in the CPU and the GPU computational domains.

The goal of this paper is to improve on the shortcomings of the traditional multiresolution representation, consolidating the operations involved in simplification and reconstruction without storing the simplification sequence. This allows our multiresolution representation to be effective on both the CPU and the GPU. One of the first steps towards this goal is the *Geometry Image* (GIM) introduced by Gu et al. [5]. In GIM, the geometry of the surface, as well as other attributes, is encoded in a simple matrix, transforming arbitrary surfaces into a completely regular representation. This regularity is particularly important for GPU-based applications since it enables the usage of textures and guarantees data coherence, diminishing the need of vertex-caching techniques in hardware. Additionally, a mip-map pyramid can be constructed from a given GIM texture, attuning the original surface to multiresolution representation.

The following work of Sander et al. [6] describes *Multi-Chart Geometry Images* (MCGIMs), taking a step further representing the surface with a regular atlas structure. In MCGIM, the surface is partitioned in a set of charts, where each chart is a regular GIM. This representation improves on the parametrization distortion of the first approach by mapping the surface piecewise to a regular domain instead of mapping the entire surface. However, both approaches undergo the same problem of gluing parts of the surface together. On the one hand, GIM uses a topological sideband to glue the boundary of the parametric domain when the surface is not a topological disk. On the other hand, MCGIM uses a zippering method gluing the boundaries

of each chart by snapping boundary samples on the regular domain. In this paper, we borrow the idea of regular charts sampling from GIM and MCGIM but combined with simplification in the same spirit of PM and MAPS to avoid the post-processing topological and zippering solutions.

To build our atlas structure, we first simplify the input dense mesh (see figure 1(a)) by removing vertices in a different way than PM. We make use of two stellar operators to remove vertices in the simplification process [7]: *face weld* and *edge weld*, explained in sections 3 and 4. The resulting coarse triangular mesh is converted to a base quad mesh, as depicted in figure 1(b), using a weighted graph matching algorithm [8]. During simplification, each removed vertex is projected onto the corresponding face (in face weld) or edge (in edge weld) to yield our local parametrizations and final atlas structure (see figure 1(d)).



(a) (b) (c) (d)

Figure 1: Illustrative example of our atlas construction: (a) A scanned input model (Kikito) is simplified using stellar operations and then converted to a quad mesh (b). (c) The superposition of both dense and base meshes. (d) The parametrization done individually on each base-mesh face and edge yields our charts and final atlas structure.

1.1. Contributions

In this paper, we make three contributions in atlas-based mesh representation. We first introduce an alternative method to incrementally compute local parametrizations throughout the simplification process. In contrast with the original PM method, we use a hierarchical simplification algorithm based on stellar operators, which changes the mesh resolution by at most 1-ring neighborhood instead of 2-rings as done by the PM basic operation — *edge collapse*. Our choice of operators is similar to MAPS simplification via *vertex removal* and retriangulation, but improving on the mesh generation quality and allowing a quad, or *tile*, coverage of the mesh and the usage of dual operators for reconstruction.

Next, we adapt a well-known data structure, named *half-edge* [9], to handle multiresolution and the connectivity of the charts generated by each local parametrization. While the charts are stored regularly in a texture to respect the GPU requirement of data coherence, in the CPU the charts maintain the connectivity of the simplified mesh using the halfedge data structure. The resulting atlas stores the multiple charts, adding a low additional cost to the overall data structure while providing an adaptive, multiresolution hierarchy by controlling how many vertices to use from each individual chart.

Finally, we present a method to combine our connected structure of charts in the CPU with a boundary-aware structure of *chart interiors* in the GPU. This method enables a subdivision scheme to control the mesh resolution at lower levels in the CPU and, at the same time, allows the GPU to further increase the resolution at higher levels.

2. Related Work

Multiresolution representation of a surface, i.e. a polygonal mesh, can be obtained essentially from two different strategies. One is explicit, through an atlas parametrization of the surface capturing levels of detail from a base coarse mesh to the dense original mesh. And the other is parametric, by smooth surface representation, that is a surface composed of parametric patches glued together. Both strategies have a vast literature and several works intersect these two concepts, for a comprehensive overview please refer to Botsch et al. [1], Floriani and Magillo [10] and Grimm and Zorin [11].

The partition of a polygonal mesh into charts is the first target of study in many works devoted to construct a multiresolution representation. The work of Krishnamurthy and Levoy [12] presents an interactive system where the user manually partitions the polygonal mesh into charts, to then fit B-Spline patches to each chart and generate a smooth surface. This fitting is also done by Losasso et al. [13] but for genus-0 surfaces, using only one chart, and on an octahedral domain. MAPS [3] makes the partitioning automatic by combining the fitting with simplification from fine to coarse resolution. Carr et al. [14] improve on the chart-packing strategy of MAPS by using rectangular domains, packing is discussed on the context of our contribution on section 6. When considering the base coarse mesh as the base domain of a subdivision surface, several subdivision schemes can be used to obtain a smooth surface [4, 15, 16]. The choice of one over the other depends on the initial base domain and on the desired multi-chart and multiresolution properties.

The work of Sander et al. [17] partitions a mesh using greedy face clustering (similar to MCGIM [6]) to parametrize PM, minimizing texture stretch. The concurrent work of Garland et al. [18] also presents a face-clustering algorithm, but minimizing a more general error metric. The subsequent work of Shlafman et al. [19] develops a clustering-based partitioning method in the interesting context of surface metamorphosis. The more recent work of Tarini et al. [20] considers the intersection of the mesh with poly-cubic faces to build seamless texture mapping, called PolyCube-Map, stored and accessed by the GPU. Depending on the goal, the mesh partitioning method can account for stretch and texture deviation, chart planarity, or even a common morphing domain, among others.

After mesh partitioning, the multiresolution representation can be constructed explicitly by geometric atlas parametrization. The work of Ji et al. [21] builds and maintains an atlas structure using the GPU via PolyCube-Maps [20]. However, the PolyCube-Map is not adequate to detailed geometric features, as it is designed for texture mapping. The concurrent work of Hernández and Rudomin [22] prioritizes rendering simplifying the charts transition and restricting the multiresolution representation to a view-dependent application. In this paper, we address these limitations defining a generic multiresolution representation and using different strategies for the atlas construction and handling.

The multiresolution representation can also be constructed implicitly by approximation to a smooth surface [3, 12]. The work of Lee et al. [23] augments this idea with a per-patch displacement field for a better approximation of the original mesh. Later, the work of Boier Martin et al. [24] combines a face-clustering method to generate a quadrilateral base mesh with parametrization to obtain a Catmull-Clark [15] subdivision surface. The more recent work of Bokeloh and Wand [25] synthesizes geometry on each region using a subdivision procedure on the GPU. Although fitting smooth surfaces is an interesting alternative, we will focus on an explicit representation of the surface via multiple regular charts in an atlas structure, in the same line of work of the GIM [5] and MCGIM [6] approaches.

One direct way to construct an atlas parametrization of the surface is to cut it open to be topologically equivalent to the plane, and use a global parametrization. The work of Kharevych et al. [26] proceeds in this direction improving on the angle-based flattening algorithm [27] by introducing cone singularities and using circle patterns. Later, the circle packing metric is generalized to Ricci flow algorithms in the work of Jin et al. [28]. Although finding a global parametrization is a way to build an atlas structure, here we focus on local parametrizations done incrementally in the spirit of PM [2] and MAPS [3].

In this work we consider the problem of finding an adaptive, multi-chart and multiresolution mesh representation for surfaces, which is effective for both the CPU and the GPU. Obtaining an adaptive multiresolution representation in these two contexts is a complex problem, even considering only the GPU computational model [21, 25, 29, 30].

3. Background

We first define the problem and outline our solution to then describe a few concepts used in our mesh representation.

Problem Definition. The problem is to obtain an adaptive multi-chart and multiresolution representation of the same surface in two different computational domains. The first is the CPU, capable of handling irregular connectivity which is inherited directly from the static input mesh; the second is the GPU, designed to deal with regular implicit connectivity in the form of texture images. The solution we present takes as input a triangulated two-manifold mesh and returns two data structures suitable for both computational domains. This solution allows us to control the mesh resolution in two cascading level-of-detail processes, each of which attuned to the differences between the CPU and the GPU.

Triquad and 4-k Meshes. Our adaptive multi-chart and multiresolution representation builds on the variable resolution 4-k meshes [7] framework. The multiresolution in this framework is obtained by minimal local modifications combined with a special type of mesh — a triangulated quadrangulation (or *triquad*), where every triangle is uniquely paired up with another triangle to form a quad which is not necessarily planar.

Stellar Operators. In a triangulated manifold, the set of operators that changes the mesh in a minimum local neighborhood are called stellar operators [16]. These operators are used in our surface representation to simplify, refine and change the connectivity of the mesh. Figure 2 summarizes the action of each stellar operator, namely: *face weld* and its dual *face split*; *edge weld* and its dual *edge split*; and *edge flip* whose dual is itself.

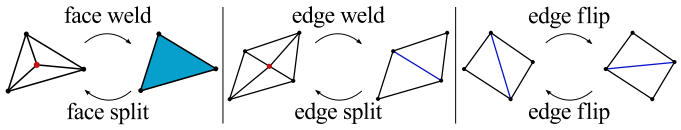


Figure 2: The stellar operators used in our representation.

4. Adaptive Multi-chart and Multiresolution Mesh

We follow the idea of minimum modifications to construct our surface representation in two complementary methods. We start by simplifying an input dense-polygon mesh storing the induced parametrization in a simple way, as detailed in section 4.1. Then the triangles of the coarse mesh are paired to form *tiles*, producing a hierarchical 4-k mesh, explained in section 4.2. The result is a triquad mesh equipped with information of the input mesh (see an example in fig 1(d)). This information produces two data structures, explained in section 4.3, used to reconstruct the surface in section 4.4.

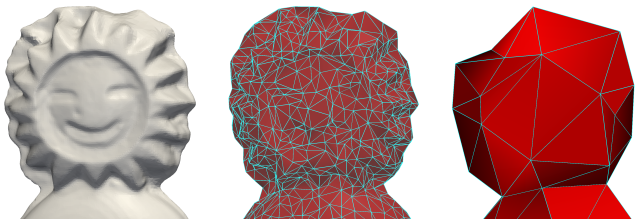
4.1. Simplifying the Mesh

The simplification method we use follows the construction method described by Velho and Gomes [7]. The idea is to apply only stellar weld operators to the mesh, altering its resolution in a minimal way. More specifically, only the 1-ring neighborhood of faces of the vertex being removed (in red in figure 2) will change. As a consequence, the boundary vertices and edges of this 1-ring region, or its link, remain intact.

In our scenario we use a combination of edge flips and face or edge welds to perform the simplification process. The edge flip operator can change the surface drastically and it is only applied to better approximate the original surface; or to match the vertex degree requirement of 3 or 4 for simplification. In the first case, the flip is used to choose an interior edge when performing an edge weld operator. While in the second, the flip is used to reduce the degree of a vertex selected to be removed enabling a weld operator. In both cases we estimate the

error of performing the operator using the quadric error metric, introduced by Garland and Heckbert [31]. Our simplification method uses only flip and weld operators, later we explain how to use the split operator for adaptive resolution control.

On each step of the simplification process, the removed vertex is parametrized on the simplified face (in case of a face weld) or edge (in case of an edge weld) using an exponential mapping. This induces a hierarchical parametrization of the surface that is maintained throughout simplification, i.e. vertices that have been mapped to a face in previous simplification steps are re-mapped to the current face using barycentric coordinates. The resulting multiresolution parametrization is similar to MAPS [3], but it is simpler to implement and achieves better results due to the locality of the stellar simplification operators (see figure 3 for an example of a simplified mesh). Triangles and remeshing quality is further discussed in section 6.



(a) (b) (c)

Figure 3: The head of the Kikito scanned model (a) is simplified using stellar operators to 2% of the original mesh (b) and 0.08% in the final coarse mesh (c).

After completing the simplification process (we decimate to approximately 0.1% of the original mesh), the resulting mesh is our base domain with the dense mesh parametrized locally on top of it. The simplification can reduce the original mesh size to an arbitrary size as long as the simplified mesh is a triangulated two-manifold mesh.

The companion parametrization has two important properties. First, the pre-image of every vertex of the dense mesh is a point whose coordinates are barycentric coordinates with respect to the vertices of a base-mesh face or edge. Second, the base-mesh vertices are, in fact, vertices of the original mesh. We use these properties later to construct our atlas descriptor.

4.2. Building the Tiles

The result of simplifying the input mesh is an equivalent triangulated two-manifold mesh. In order to use the stellar refinement operators to attain multiresolution, we need to cover the entire simplified mesh with tiles, that is, pairs of triangles forming quads. To that end we use a graph matching idea similar to the one described in Daniels II et al. [8].

The idea is to pair triangles by finding a maximum weight perfect matching on the dual graph of the triangulated mesh. This matching is guaranteed to exist in meshes without boundaries, i.e. with an even number of triangles. However, in practice, we overcome this limitation by performing edge splits on unpaired border triangles, in case of non-degree-3 dual vertices. The matching is then equivalent to pairing all mesh triangles in such a way that every triangle belongs to only one pair (see an example in figure 4). To compute the matching, we define a function that assigns a weight to each edge of the dual graph, or mesh edge. These weights are used to find a maximum weight perfect matching on the (weighted) dual graph, which is a perfect matching whose sum of edge weights of paired triangles is maximum over all perfect matchings.

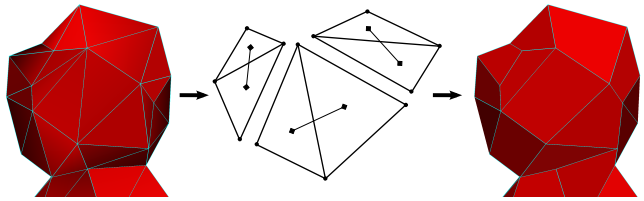


Figure 4: The head of the Kikito coarse mesh is covered with tiles using triangle pairing. The result is our base quad mesh.

In our scenario we use the weights to obtain a tile coverage suitable to be used as an atlas structure, considering one chart for each tile. Instead of a function that values planarity of the paired triangles, as in the original work of Daniels II et al. [8], we use a function that penalizes overlapping of neighboring tiles. Our weight function first ranks possible tiles by orthogonality; and then by "manifoldness", i.e. the weight is set to zero if the corresponding edge has one vertex of degree 3. This penalty feature enables the reconstruction of the original surface on top of each tile, avoiding the case of pairing two triangles incident to a degree-3 vertex and generating geometry overlaps.

The result of the matching algorithm is a triangulated mesh with all triangles paired, or a triquad mesh. The paired triangles form quad tiles that are used in our representation to change the surface resolution adaptively. Moreover, the tiles are also used as charts in a multi-chart regular structure.

4.3. Multiresolution Data Structures

With the techniques described so far, we have simplified an input dense mesh, storing local parametrizations, and paired the simplified triangles, grouping parametrizations pairwise. The output is a triquad base mesh equipped with information from the input mesh, which suffices to build our multiresolution representation of the mesh through two data structures.

The first is a halfedge-based data structure in the CPU specialized to represent the collection of all parametrizations, i.e. our multiple charts, and to support stellar operators, i.e. our adaptive mesh. We start by constructing the regular halfedge data structure from the triquad base mesh, but classifying halfedges in two types: *boundary* and *interior*. Each dual-graph edge not chosen when pairing base-mesh triangles (explained in section 4.2) corresponds to a base edge in the boundary of a chart, which generates two boundary halfedges. Each dual-graph edge chosen by the pairing corresponds to a base edge in the interior of a chart, generating two interior halfedges. Figure 5 (center) shows an example of boundary halfedges (in

brown) and interior halfedges (in blue). We classify the halfedges by making each triangular face point to its interior halfedge, which avoids the need for additional data structure space.

After classification, we store on each halfedge an *index* to the chart that contains it, and two positions on the parameter domain: *start* and *end*. The parameter domain of a chart is a unit-square region, illustrated in figure 5 (right), following the texture-mapping design of the GPU. The start and end positions are stored as two (u, v) coordinates. Both index and position coordinates ensure a one-to-one correspondence between mesh and atlas. Finally, we store on each vertex a *resolution level* (starting at 0 for base-mesh vertices) to control the mesh adaptivity. This is done similarly to the 4-8 subdivision method [16] but, instead of a 4-8 mesh with smooth subdivision control, we have a 4-k mesh with a parameter domain per tile. Remarkably, the subdivision scheme works exactly in the same way.

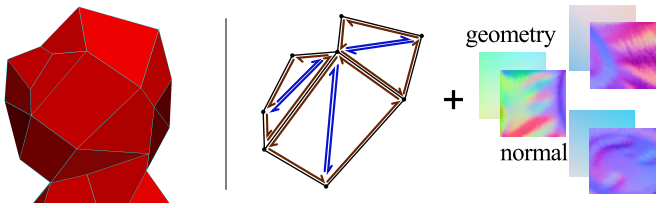


Figure 5: The base mesh (left) is converted to a specialized halfedge data structure (center) and a collection of regular charts (right), one for each base-mesh quad face.

The second data structure is a set of charts, stored on both the CPU and the GPU, computed using the local parametrizations and the triquad base mesh. The boundary edges of a quad face are mapped to the unit square, and the parametrization on each edge is transferred from its linear domain (explained in section 4.1) to each side of the square. This produces duplicates of edge parametrizations (each boundary edge is shared by two charts) but it amounts for a small additional storage on the total data structure (discussed in section 6). The replication of chart boundaries is important to guarantee that the resulting adaptive tessellation in the GPU is crack-free, while maintaining texel-fetching coherence.

The interior of a chart is obtained by triangle rasterization using ordinary scan conversion, similar to MCGIM [6]. We rasterize vertex attributes, such as geometry and normals (see examples in figure 5), of the dense 3D triangles inside a chart (one vertex inside is enough to consider a triangle inside) into a 2D image. This image is cropped to the unit square and yields our chart domain (we discretize it in a 33×33 image). Finally, the affine interpolation along the edge lying between two neighboring charts (a 1D image also discretized with 33 pixels) is used to average the values around the boundary of the two charts. Both chart boundaries are updated to the average value ensuring a correct overlap of attributes. It is interesting to note that this average is not necessary for chart corners, since the base-mesh vertices are the original dense vertices and the average would consider k equal vertices of degree k.

The two data structures comprise our multiresolution representation. The original surface is approximated using the specialized halfedge and set of charts, allowing not only the CPU but also the GPU to control the resolution.

4.4. Reconstructing the Surface

The reconstruction of the surface uses the halfedge data structure to determine the edges that can be split, or the vertices that can be weld, by the stellar operators. Initially, all vertices are set to be at level 0 and unable to be removed and only interior edges can be used to refine the mesh. The edge split operator inserts a new vertex at the middle of the split edge, illustrated in figure 6, with attributes given by the sample at that position on the corresponding chart image. The sample position is determined by the split halfedge data, i.e. chart index, and start and end (u, v) positions. The complementary edge weld operator removes a vertex from the interior of a tile.

The edge split operator changes all four boundary halfedges to be interior halfedges, consequently changing the new and interior halfedges to be boundary. The edge weld operator undoes these changes. Altering halfedges to be either interior or boundary is sufficient to determine split edges and weld vertices. An edge can be split if both its halfedges are interior, and a vertex can be weld if its 1-ring halfedges are boundary. Figure 6 shows an example of a sequence of split operators applied to refine the mesh adaptively on a certain region.

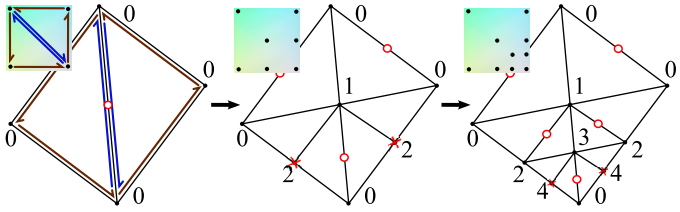


Figure 6: Illustrative example of the adaptive resolution control of a tile and its chart, showing the corresponding vertex levels. At each resolution, only a set of edges can be split (with a circle) and a set of vertices can be welded (with a cross).

The successive applications of stellar operators induces a variable-resolution lattice, where the coarse (dense) mesh is the source (sink) node. Any cut in this graph is an adaptive mesh and, since the base coarse mesh is initially a 4-k mesh, it is not necessary to store the whole graph, only the current mesh is required to have the entire adaptive multiresolution. Moreover, the resolution level at each vertex is used to enforce an invariance of one-level maximum difference between adjacent faces. This feature produces a smooth resolution transition when adapting the mesh.

The refinement and simplification rules are managed by the CPU, where the halfedge data structure resides, and guarantee that the surface can also be adaptively reconstructed in the GPU. The CPU can stipulate any resolution to the mesh, from coarse to fine (up to the chart image resolution), and the GPU starts from this predefined resolution. The GPU can further control the resolution of the mesh, using built-in tessellation

shaders, following different subdivision rules. These rules derive from the fact that the GPU tessellates the surface in parallel with the information given by the CPU and the set of charts stored as an atlas image in texture memory.

The CPU information sent to the GPU captures the current halfedge data structure via a list of patches with four attributes per patch: (u, v), s and c. The first two attributes (u, v) are the lower-left coordinates in the atlas image, s is the side of the patch in pixels, and c defines the initial subdivision of the patch. The patch can be seen as a square piece of the atlas of up to the size of the chart. In the case the patch is the entire tile, as in figure 6 (center), the (u, v) is the chart origin position in the atlas, s is the chart size, and c defines the inner and outer GPU tessellation levels to match the edges inside the tile. When more than one odd-level vertex appears in the interior of a tile, the patch is subdivided into four patches and their attributes are updated accordingly. This additional CPU subdivision is similar to the restricted quadtree of the Catmull-Clark [15] subdivision scheme, but allowing adaptivity. Figure 7 shows an example of a tile with level-1 and level-3 vertices subdivided into four patches by the CPU to be sent to the GPU.

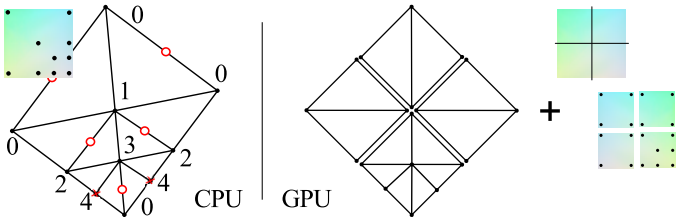


Figure 7: The tiles and charts are converted by the CPU into four patches sent to the GPU.

After the patches are sent, the GPU subdivides the patches in parallel, using the atlas image to evaluate vertex attributes generated by the tessellator. The patch subdivision follows two rules to avoid cracks in the surface. The first is to have the minimum subdivision level given by the patch attribute c. The second is to specify border subdivision consistently across patches, that is, using only the information from pixels at the patch boundary. With these two rules, the parallel specification of outer subdivision levels is guaranteed to generate a valid triangulated two-manifold mesh (see examples in figures 8(c) and (d)).

The combined data structures with patch definitions and subdivision rules constitute our adaptive multi-chart and multiresolution representation. Figure 8 illustrates a model reconstructed from its base triquad mesh (coarsest resolution) to arbitrary mesh resolutions. Note that different parts of the mesh are at different resolutions defined by either the CPU or the GPU.

5. Applications

Multiresolution meshes have a wide variety of applications, from real-time collision detection to high-definition texture mapping. Multiresolution with adaptivity is even better positioned than fixed resolution control, since it can be applied to regions of interest in the mesh rather than the entire mesh. Here, we describe two illustrative applications of our mesh representation.

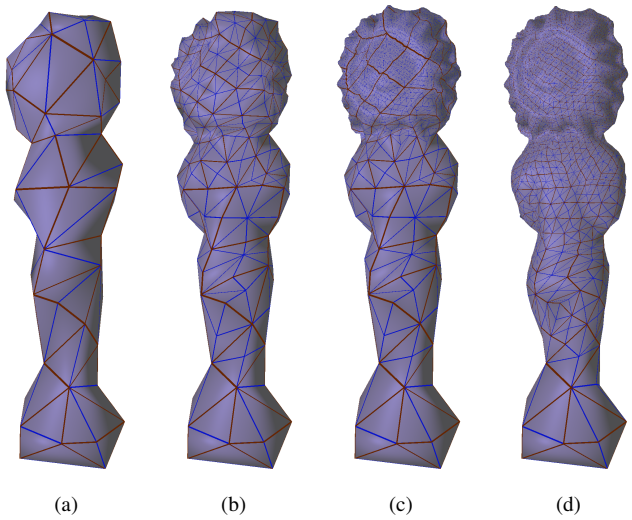


Figure 8: Final reconstruction of the Kikito statue: its base triquad mesh (a) is adaptively refined from bottom to top in the CPU (b); the head of the model is further refined in the GPU (c); and in (d) a more detailed refinement is imposed by the CPU. Boundary edges (in brown) and interior edges (in blue) denote the patches resolution level on each stage.

Cascading Level-of-Detail. The first application aims to render hundreds of meshes using level-of-detail (LOD) at the same time in the CPU, with adaptive refinement as in figure 8(b), and in the GPU, with view-dependent tessellation refinement. Figure 9 illustrates this cascading LOD applied to a checkerboard of meshes rendered in real-time.

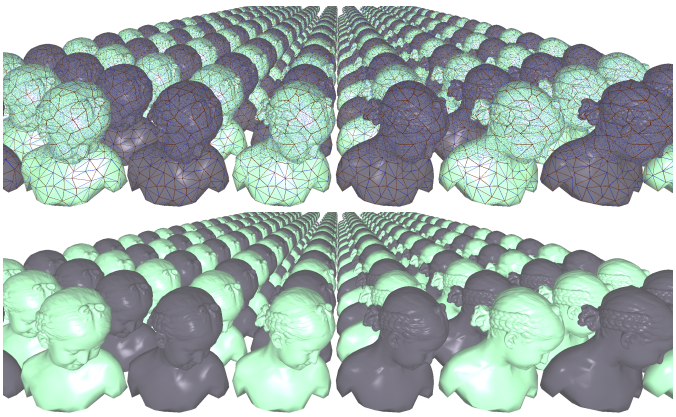


Figure 9: The Bimba model is adaptive refined in both the CPU and the GPU (top) to be rendered with LOD (bottom).

Fine-detail Editing. The second application uses the atlas structure to add details to a mesh by creating extra charts. These charts are combined with the regular charts to create a projection effect of the editing. Figure 10 shows this fine-detail editing of the word *3-torus* in a model, regions close to the word (inner ring bottom) are at higher resolution than far regions (outer rings top) defined by the CPU and the GPU.

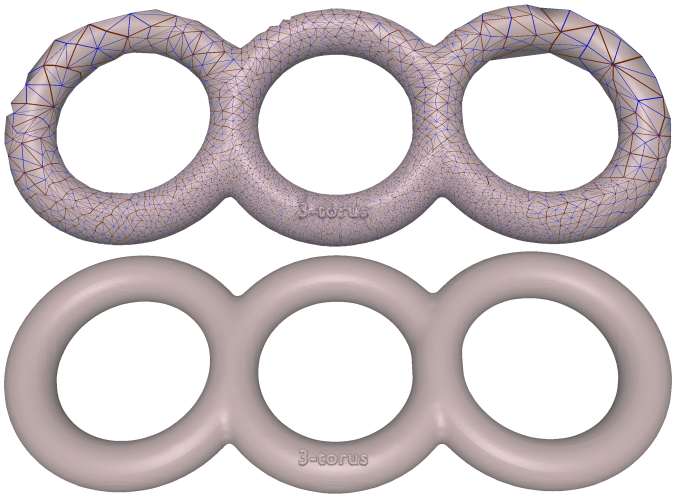


Figure 10: The 3-torus model is edited using adaptive refinement (top) to then be rendered in full detail (bottom).

6. Results and Discussion

Our test platform is an Intel Core i7 2600 CPU with 16GB of RAM and an nVidia GeForce GTX 560 Ti GPU with 1GB. We have constructed our adaptive multi-chart and multiresolution representation for a large number of models. The entire pre-computational pipeline is automatic and takes less than 15 minutes to complete per model, where simplification and matching are the most expensive steps.

Table 1 summarizes the conversion pipeline for several models. The simplification followed by matching transform the model's dense mesh in a triquad base mesh. After matching, the rasterization step creates each chart to be packed in an atlas structure. This structure comprehends all surface attributes, such as geometry, normals and height-map editing, stored as a 32-bit per-channel atlas texture per attribute in the GPU. *Atlas efficiency* measures the amount of significant information in the pixels of each texture. The packing step organizes the charts in a matrix form close to a square, which may leave several chart slots empty inside the final atlas (see figure 13), reducing atlas efficiency. The replication of boundaries inside charts also slightly reduces the atlas efficiency.

Our specialized halfedge data structure is used from the initial state of our multiresolution representation to the final full-resolution state in the CPU. The data structure is dynamic and expands as the mesh resolution increases. The additional storage is low compared with a regular halfedge data structure. It requires 6 bytes per halfedge (2B for the chart index and 4B for the start and end positions) and 1 byte per vertex for the resolution level. Considering the storage of only the base mesh and the atlas texture, our combined multiresolution data structure is smaller than the original static dense mesh using regular halfedge.

We express geometric accuracy in our multiresolution representation as Peak Signal-to-Noise Ratio (PSNR), following MCGIM [6]. The PSNR, $PSNR = 20\,log_{10}(peak/dist)$, is computed using the *peak* as the bounding box diagonal of the dense mesh and *dist* as the Hausdorff distance between the dense mesh and each multiresolution mesh from level 0 (base mesh) to 10 (full-resolution mesh). The decimation rate of 0.1% and chart size of 33×33 pixels ensure that the level-10 mesh has approximately the same number of vertices and triangles of the dense mesh. Figure 11 shows the measured PSNR for each tested model.

Table 1: Details of our multiresolution mesh representation. For each tested model, we show the number of vertices (#v) and triangles (#t) of the original dense mesh and the simplified base mesh, the number of charts (#c) in the atlas structure, the size of the atlas in pixels and its efficiency (eff.).

Model	Dense Mesh		Base Mesh		Atlas Structure		
	#v	#t	#v	#t	#c	size	eff.
Kili	490 K	977 K	559	1028	514	759×759	91.3%
Neptune	2 M	4 M	2336	4680	2340	1617×1584	91.4%
Bimba	192 K	384 K	338	672	336	627×594	86.7%
Gargoyle	97 K	194 K	182	360	180	462×429	85.2%
3-torus	16 K	34 K	71	150	75	297×297	86.9%
Kikito	75 K	150 K	66	128	64	264×264	93.9%

Our strategy of combining a stellar-based simplification with a manifold-aware triangle matching results on a good aspect ratio for triquads, as can be seen in figures 1(b) and 8(a). The base-mesh vertices are distributed across the base surface as a result of our pipeline. Additionally, reconstructing the surface using subdivision followed by tessellation with squared charts increases mesh quality while maintaining the aspect ratio of triangles, as shown in figure 8(d) and by the PSNR results. This is true even for complex models, such as the the Neptune statue and the Kilimanjaro terrain model, illustrated in figures 12 and 13, respectively. The Kilimanjaro model (Kili) achieves the highest geometric accuracy, with irregular connectivity, almost one million triangles and boundaries.

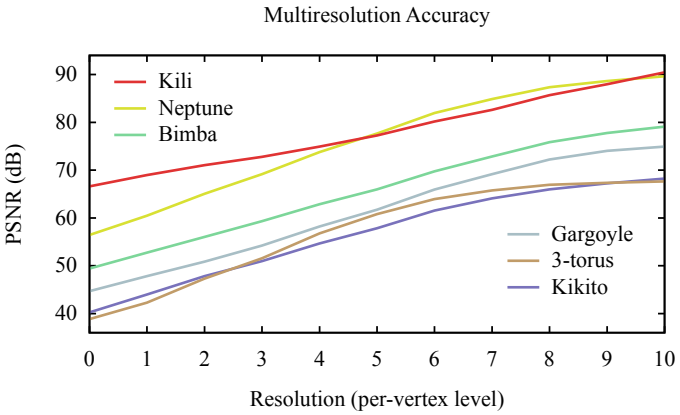


Figure 11: Precision of our multiresolution representation.

Analyzing further the PSNR results, the highest resolution of the Gargoyle model achieves a slightly lower accuracy (8.9dB

Figure 12: The Neptune model simplified and tiled to its base mesh (left), increasing the resolution to level-3 (second from the left), level-6 (third) and level-9 (fourth); and the original dense mesh (right). Each upper-left corner details the top of the model's head. Quads drawn are in fact triquads.

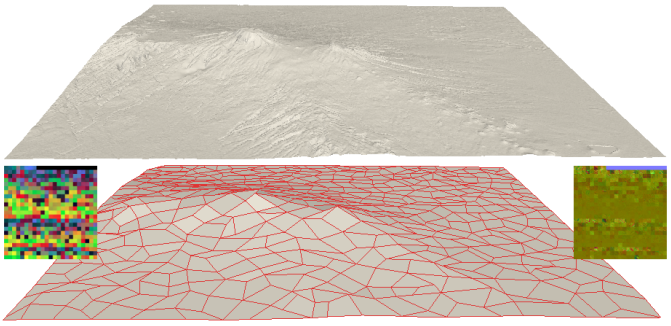


Figure 13: The Kili model (top) converted to our multiresolution representation (bottom): the base mesh (center); geometry (left) and normal (right) atlases.

difference) compared with MCGIM (see [6, figure 14]). The geometry accuracy is lower given the fact that our chart domains are squared regions and we do not have an optimization step. On the flip side, our packing step is more efficient and simpler; we have a final atlas image more suitable to GPU-based applications; and the accuracy of our charts can also be improved by an optimization procedure similar to MCGIM.

Compared against MAPS [3], our simplification process is based on *atomic* operations, i.e. stellar operators, that can reproduce both PM [2] and MAPS simplification. The maximum error of our full-resolution reconstruction ranges from 0.5% (for the 3-torus model) to 0.1% (for the Kili model), improving on the results for *remeshing tolerance* of MAPS. In contrast with MAPS, our reconstruction uses the inverse set of atomic operators, allowing resolution control and the connection we present between the CPU and the GPU data structures.

7. Conclusions

We describe a mesh representation that changes its resolution in a dynamic and adaptive way. The multiresolution representation builds on an explicit atlas-based parametrization of the static input surface using a novel simplification method. The duals of the simplification operators are used in the reconstruction method from coarse to fine resolution. The atlas combined with dual operators leads to a multiresolution mesh representation efficient to both the CPU and the GPU. We demonstrate the practical use of our representation in two applications: rendering hundreds of meshes with cascading LOD and editing a small part of an adaptive mesh.

Future Work. Although we have focused on both the CPU and the GPU computational models, our multiresolution representation should specialize to either model using the presented data structures. One example of CPU specialization is to maintain the irregular domain of each chart, instead of rasterizing it to a regular domain, improving reconstruction accuracy.

Another more profound specialization is to adapt the atlas structure to a multiresolution subdivision surface fitted to the dense mesh. Since our multiresolution representation is based on stellar operators it would be trivial to incorporate either a Catmull-Clark subdivision or $\sqrt{2}$-subdivision while still maintaining adaptivity control.

Acknowledgments

We would like to acknowledge the grants provided by CNPq (Brazilian National Counsel of Technological and Scientific Development), including numbers 305845/2012-8 and 486951/2012-0. We thank ParaView (www.paraview.org) for aiding the rendering of several images; MeshLab (meshlab.sourceforge.net) for the Hausdorff distance filter applied on all models; and AIM@SHAPE (www.aimatshape.net) for providing several models used in this paper.

References

[1] Botsch M, Kobbelt L, Pauly M, Alliez P, Lévy B. Polygon Mesh Processing. AK Peters; 2010. ISBN 978-1-56881-426-1.

[2] Hoppe H. Progressive Meshes. In: Proc. of ACM/SIGGRAPH Conf. ISBN 0-89791-746-4; 1996, p. 99–108. doi:10.1145/237170.237216.

[3] Lee A, Sweldens W, Schröder P, Cowsar L, Dobkin D. MAPS: Multiresolution Adaptive Parameterization of Surfaces. In: Proc. of ACM/SIGGRAPH Conf. ISBN 0-89791-999-8; 1998, p. 95–104. doi: 10.1145/280814.280828.

[4] Loop CT. Smooth subdivision surfaces based on triangles. Master's thesis; Department of Mathematics, University of Utah; Salt Lake City, Utah, USA; 1987.

[5] Gu X, Gortler SJ, Hoppe H. Geometry Images. In: Proc. of ACM/SIGGRAPH. ISBN 1-58113-521-1; 2002, p. 355–61. doi: 10.1145/566570.566589.

[6] Sander PV, Wood ZJ, Gortler SJ, Snyder J, Hoppe H. Multi-Chart Geometry Images. In: Proc. of SGP. ISBN 1-58113-687-0; 2003, p. 146–55.

[7] Velho L, Gomes J. Variable resolution 4-k meshes: Concepts and applications. Computer Graphics Forum 2000;19(4):195–212. doi: 10.1111/1467-8659.00457.

[8] Daniels II J, Lizier M, Siqueira M, Silva CT, Nonato LG. Template-based Quadrilateral Meshing. Computers & Graphics 2011;35(3):471 –82. doi: 10.1016/j.cag.2011.03.024.

[9] Mäntylä M. An introduction to solid modeling. Computer Science Press; 1988. ISBN 9780881751086. URL `books.google.com.br/books?id=CJVRAAAAMAAJ`.

[10] Floriani LD, Magillo P. Multiresolution Mesh Representation: Models and Data Structures. In: Tutorials on Multiresolution in Geometric Modelling. Springer-Verlag; 2002, p. 363–418.

[11] Grimm CM, Zorin D. Surface Modeling and Parameterization with Manifolds. In: ACM SIGGRAPH Courses. ISBN 1-59593-364-6; 2006, p. 1–81. doi:10.1145/1185657.1185728.

[12] Krishnamurthy V, Levoy M. Fitting Smooth Surfaces to Dense Polygon Meshes. In: Proc. of ACM/SIGGRAPH Conf. ISBN 0-89791-746-4; 1996, p. 313–24. doi:10.1145/237170.237270.

[13] Losasso F, Hoppe H, Schaefer S, Warren J. Smooth Geometry Images. In: Proc. of SGP. ISBN 1-58113-687-0; 2003, p. 138–45. URL `http://dl.acm.org/citation.cfm?id=882370.882389`.

[14] Carr NA, Hoberock J, Crane K, Hart JC. Rectangular Multi-Chart Geometry Images. In: Proc of SGP. ISBN 3-905673-36-3; 2006, p. 181–90. URL `http://dl.acm.org/citation.cfm?id=1281957.1281981`.

[15] Catmull E, Clark J. Recursively generated B-spline surfaces on arbitrary topological meshes. Computer-Aided Design 1978;10(6):350–5. doi: 10.1016/0010-4485(78)90110-0.

[16] Velho L, Zorin D. 4-8 Subdivision. Computer-Aided Geometric Design 2001;18(5):397–427. doi:10.1016/S0167-8396(01)00039-5.

[17] Sander PV, Snyder J, Gortler SJ, Hoppe H. Texture Mapping Progressive Meshes. In: Proc. of ACM/SIGGRAPH. ISBN 1-58113-374-X; 2001, p. 409–16. doi:10.1145/383259.383307.

[18] Garland M, Willmott A, Heckbert PS. Hierarchical Face Clustering on Polygonal Surfaces. In: Proc. of I3D. ISBN 1-58113-292-1; 2001, p. 49–58. doi:10.1145/364338.364345.

[19] Shlafman S, Tal A, Katz S. Metamorphosis of Polyhedral Surfaces using Decomposition. Comput Graph Forum 2002;21(3):219–28. doi: 10.1111/1467-8659.00581.

[20] Tarini M, Hormann K, Cignoni P, Montani C. PolyCube-Maps. In: Proc. of ACM/SIGGRAPH Conf. 2004, p. 853–60. doi: 10.1145/1186562.1015810.

[21] Ji J, Wu E, Li S, Liu X. View-Dependent Refinement of Multiresolution Meshes using Programmable Graphics Hardware. Vis Comput 2006;22(6):424–33. doi:10.1007/s00371-006-0020-8.

[22] Hernández B, Rudomin I. Simple Dynamic LOD for Geometry Iimages. In: Proc of GRAPHITE. ISBN 1-59593-564-9; 2006, p. 157–63. URL `http://doi.acm.org/10.1145/1174429.1174454`.

[23] Lee A, Moreton H, Hoppe H. Displaced Subdivision Surfaces. In: Proc. of ACM/SIGGRAPH Conf. ISBN 1-58113-208-5; 2000, p. 85–94. doi: 10.1145/344779.344829.

[24] Boier Martin I, Rushmeier H, Jin J. Parameterization of Triangle Meshes over Quadrilateral Domains. In: Proc. of SGP. ISBN 3-905673-13-4; 2004, p. 193–203. doi:10.1145/1057432.1057459.

[25] Bokeloh M, Wand M. Hardware Accelerated Multi-Resolution Geometry Synthesis. In: Proc. of I3D. ISBN 1-59593-295-X; 2006, p. 191–8. doi: 10.1145/1111411.1111446.

[26] Kharevych L, Springborn B, Schröder P. Discrete Conformal Mappings via Circle Patterns. ACM Trans Graph 2006;25(2):412–38. doi: 10.1145/1138450.1138461.

[27] Sheffer A, Sturler Ed. Surface Parameterization for Meshing by Triangulation Flattening. In: Proc. of IMR. 2000, p. 161–72.

[28] Jin M, Kim J, Luo F, Gu X. Discrete Surface Ricci Flow. IEEE TVCG 2008;14(5):1030–43. doi:10.1109/TVCG.2008.57.

[29] Ritschel T, Botsch M, Müller S. Multiresolution GPU Mesh Painting. Eurographics Short Papers; 2006, p. 17–20.

[30] Hu L, Sander PV, Hoppe H. Parallel View-Dependent Refinement of Progressive Meshes. In: Proc. of I3D. ISBN 978-1-60558-429-4; 2009,doi: 10.1145/1507149.1507177. 169–176.

[31] Garland M, Heckbert PS. Surface Simplification using Quadric Error Metrics. In: Proc. of ACM/SIGGRAPH Conf. ISBN 0-89791-896-7; 1997, p. 209–16. doi:10.1145/258734.258849.