An Adaptive Multiresolution Mesh Representation for CPU-GPU Coupled Computation

A. Maximo[1] and L. Velho[1] and M. Siqueira[2]

[1]Institute of Pure and Applied Mathematics, Brazil[†]
[2]Federal University of Rio Grande do Norte, Brazil[‡]

Abstract

In this paper, we present an adaptive multiresolution mesh representation exploring the computational differences of the CPU and the GPU. We build our representation considering a dense-polygon mesh simplified to a base mesh which stores the original geometry by means of an atlas structure. For both simplification and refinement processes, we present a hierarchical method based on stellar operators. During simplification, we compute local parametrizations to generate charts and an atlas structure to be used later in refinement. Unlike previous approaches, we employ the simplified mesh as our base domain in a novel atlas descriptor using a specialized halfedge data structure combined with our charts. Finally, we show that our mesh representation can be used to adaptively control the mesh resolution in CPU-GPU coupled applications, including mesh editing and visualization.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations;
I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types.

1. Introduction

Polygonal meshes have become the de facto standard representation form for surfaces in 3D graphics applications. This is mainly due to their generality, to recent advances and popularization of laser scanning technologies, and to the existence of efficient and numerically robust algorithms for displaying, editing, smoothing, simplifying, remeshing, parametrizing, and compressing them [BPK*07]. Furthermore, dense and complex polygonal models can be rapidly displayed and processed by newly designed graphics processing units (GPUs) [SJP05, SS09, FCS*10]. However, depending on the goal, the preferred representation could be parametric [BFK84], implicit [BBB*97], based on subdivision [Zor00] or even volumetric [BK03].

A fundamental problem in geometry processing is the one of converting a surface representation form into another. This problem occurs whenever the surface data is generated or acquired in a representation that cannot be handled by the intended application, or the representation is not the more appropriate one [Ede05]. Here, we present a novel surface representation which is suitable for CPU–GPU coupled computation. The key features of our proposed representation are two-fold. First, it allows us to adaptively and dynamically represent, as well as effectively manipulate, large and complex surface models. Second, it offers a "minimum" interface that should be general enough for supporting conversion from most representation forms. Both features were made possible by the combination of two well-established notions in geometry processing: an atlas and a mesh subdivision scheme.

An *atlas*, $\mathcal{A} = \{(U_i, \varphi_i)\}_{i \in I}$, on a surface $S \subset \mathbb{R}^3$ is a collection of *charts*, (U_i, φ_i), where U_i is a subset of S, called the *chart domain*, and $\varphi_i : U_i \to \varphi_i(U_i) \subseteq \mathbb{R}^2$ is a bijective map, called the *chart map*, that maps U_i onto a subset, $\varphi_i(U_i)$, of $\mathbb{R}^2$. The chart domains "cover" S, i.e., $S = \bigcup_{i \in I} U_i$, and two or more of them may overlap at the same point in S. A surface equipped with an atlas is called a 2-*manifold* [Tu07].

[†] A. Maximo, L. Velho: {andmax,lvelho}@impa.br
[‡] M. Siqueira: mfsiqueira@dimap.ufrn.br

The concept of atlas generalizes the one of parametrization, and it has been used in the context of texture mapping [MYV93], remeshing [RLL*06], and geometry encoding [SWG*03]. In all cases, the surface S was assumed to be represented by a polygonal mesh. Here, we employ the notion of atlas without assuming any particular representation form for S. Instead, we assume that the atlas is described by a network of curves on S (i.e., the chart domain boundaries), and that we can subdivide and compute curves on S.

From a given atlas $\mathcal{A}$ on S, we build a polygonal mesh representation of S. Our construction is based on an adaptive subdivision scheme [Vel03] (controlled by the CPU) and a dynamic tessellation (done on the GPU), aiming at representing any mesh property with multiresolution. The resulting representation, which we call Manifolds-for-GPU (M4G), is a dynamic adaptive surface representation appropriate for GPU processing.

The adaptive multiresolution feature of our representation is particularly suited for progressive visualization [Hop96]. The ability to separate the mesh into parts that must be sampled densely and parts that can be represented by coarse geometry is also useful for reducing the bandwidth required for transmitting a surface between different computational domains (e.g. hard-disk, main memory and GPU memory).

The remaining of this paper is organized as follows: Section 4 gives an overview of the M4G representation; Section 6 illustrates our representation in the context of visualization, highlighting the light-weight multiresolution data structure of M4G; Section 8 discusses our representation in different contexts and concludes our work giving future research directions.

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. We use a hierarchical simplification algorithm based on stellar operators, which changes the mesh resolution by at most 1-ring neighborhood. Our choice of operators allows a quad, or *tile*, coverage of the mesh and the usage of dual operators for reconstruction.

Next, we extend a well-known data structure, named *halfedge* [Män88], 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 provides 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

Our work falls into two categories: surface representation and GPU processing. But, since they are very broad areas with vast literature, we will focus only on the aspects directly related to our research, namely: surface representation forms and parametrization, multiresolution structures, and adaptive tessellation techniques for GPU.

The most general representation for surfaces is a triangle mesh, which is often used for constructing approximations of arbitrary two-dimensional shapes [Whi40, BPK*07]. However, many important graphics applications, including texture mapping, spline-based surface modeling, and character animation, greatly benefit from a quadrangular structure. So, many algorithms for generating quadrilateral surface meshes out of triangle ones have been proposed in recent years [KNP07, HZM*08, BVK10, TPC*10, DILS*11].

Smooth surfaces representations are the main alternative to polygonal meshes. They can represent a surface by stitching parametric patches together [BFK84], as the zero-level set of an implicit function [BBB*97], by a differential atlas [GH95, YZ04, SXG*09], or by successively subdividing a control mesh [CC78, Loo87, Kob00, VZ01]. Smooth surface representations offer higher-order degree of differentiability, which can improve the convergence and accuracy of numerical computations [YZ04]. They are also much more compact than polygonal meshes [KL96], and often represent geometry exactly, which is crucial when meeting functional and aesthetic requirements [CKJ02]. On the other hand, algorithms for operating on smooth surface representations are more complex, expensive and prone to numerical problems than their counterparts for polygonal meshes.

Parametric and implicit representations have been extensively studied, their advantages are complementary, and their properties are well-understood [BFK84, BBB*97]. Manifold-based representations were motivated by the ease with which a C^k-surface, for any positive integer k or $k = \infty$, can be represented by an atlas consisting of overlapping open sets [GH95, YZ04, SXG*09]. This representation form is particularly suitable, and superior to the parametric one, for representing C^k-surfaces, for $k \geq 2$, fitted to polygonal mesh vertices [YZ04]. However, in contrast to parametric patches, chart maps constructed by the schemes available in the literature are either not polynomial or yield surfaces with at least one singular point [SXG*09]. So, algorithms for handling manifold-based surfaces are even more complex and expensive than the ones for parametric surfaces.

Subdivision surfaces offer a good compromise between the generality of meshes and the smoothness degree of manifold-based surfaces, with the advantage of being suitable for processing [CC78, Loo87, Kob00, VZ01]. The main drawback of subdivision schemes is that they are designed to describe smooth surfaces and fall short to represent shape features and details. Multiresolution surfaces and wavelet constructions exploit a hierarchy of nested scale spaces to separate different levels of detail [ZSS97]. The only restriction of this type of representation is the lack of fractional translation invariance over the surface due to the structure of basis functions.

In our work, we combine some of the best characteristics of the above representations for manifold surfaces using an atlas.

Perhaps the most powerful concept behind the manifold representation is that it enable us to work locally on a surface similarly to the two-dimensional Euclidean space. This is achieved through a parametrization, that establishes a mapping of the surface embedded in 3D to a region of the plane. The parametrization also defines a coordinate system on the surface. But since this mapping is essentially flattening a curved surface, inevitably it will cause geometric distortions. Parametrization methods try to reduce these distortions in order to preserve certain properties, such as angles, distances and areas [FHR02]. Also, depending on how this mapping is computed, it can be designed to conform to a canonical region, such as a regular polygon (therefore constraining the boundary) or to leave the boundary free as an additional degree of freedom for distortion minimization [DMA02]. Moreover, because the topology of the surface is, in general, different from the plane it is not possible to map the entire surface without cutting it open. In that respect, the parametrization methods can be classified into local and global, that respectively compute the mapping for small surface patches or the entire surface [RLL*06].

The atlas structure in our representation relies on a network of curves on the surface. Based on the assumption that we can compute curves across chart domains, as well as subdivide their boundary curves, we can define an intrinsic, curve-based parametrization of each chart. This kind of parametrization is particularly useful because it naturally yields a chart map, and allows for a good control of both the map and the boundary of the region. In addition, it has been successfully used before for computing geodesic distances in the context of mesh parametrization [LTD05] and remeshing [SSG03]. Finally, the collection of individual charts, i.e. the domains and their associated maps, is a piecewise parametrization for the whole surface. Nonetheless, we can build our atlas using different parametrization strategies as discussed in Section 8.

As mentioned before, it is desirable to work with an atlas structure in which the charts are quadrilateral regions, at least in terms of the connectivity [BVK10]. Furthermore, the importance of the decomposition of a surface into a base domain equipped with a good quadrilateral structure was recently recognized, motivating intense research. For that purpose, quadrilateral base meshing techniques have resorted to analysis based on spectral methods [MTAD08, HZM*08], and alternatively to simplification approaches [TPC*10] or other strategies [FP08, dGGDV11]. Our representation employs a base domain with quadrilateral structure generated using simplification and triangle pairing [DILS*11] as explained in Section 4.

Independently of the surface representation, very often for compatibility reasons, it becomes necessary to convert to a polygonal approximation in order to perform certain tasks, such as visualization. In these situations, it is desirable to have a mechanism to produce an adaptive mesh. Examples of such strategies are the progressive meshes [Hop96] and the stellar 4K meshes [VG00]. Here we adopt the 4–8 adaptation scheme [Vel04], which is a variant of the stellar 4K mesh.

Visualization and processing of surfaces has benefited enormously from the continuous development of graphics hardware. More specifically, the recent advances in programmable GPUs have allowed a degree of customization never seen before for real-time visualization. The recent results regarding tessellation of surfaces on the GPU can be divided into techniques that exploit subdivision schemes [ZHX*07, SJP05, PEO09, AB08] and general techniques that are targeted for triangle meshes [FFB*09, HSH10]. All the above cited methods work with programmable shaders of the graphics engine, which impose some restrictions. Other methods work directly on the parallel processors of the GPU using CUDA [SS09].

In this paper, we will exploit the tessellator stage recently introduced in OpenGL 4, which gives us simplicity of implementation and other practical advantages. However, the GPU tessellator does not allows control on how patches are triangulated. This implies in restrictions to the connectivity of the resulting mesh. Thus, as part of our contributions, we develop a strategy to overcome such limitations of the GPU tessellation.

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 multiresolution representation of the same surface in two different computational units. 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 without boundaries and returns two coupled data structures suitable to work on both units. 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 computational units.

Triquad and 4-k Meshes Our adaptive multiresolution representation builds on the variable resolution 4-k meshes [VG00] 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 [Vel03]. These operators are used in our surface representation to simplify, refine and change the connectivity of the mesh. Figure 1 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.

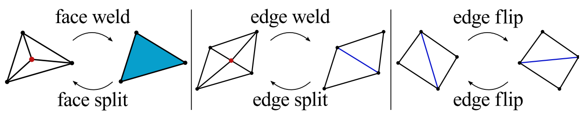

Figure 1: The stellar operators used in our representation.

4. M4G Surface Representation

Our Manifolds-for-GPU (M4G) surface representation consists of two complementary components, which are aligned with the CPU–GPU coupled computation concept. The first is a set of independent charts forming an atlas and describing properties of the surface, such as geometry and normals. Each chart shares at its boundaries a one-dimensional curve over the manifold, the minimum surface information to glue the regions together accordingly to the surface topology. This curve interface is used to join the charts assuring that the two-manifold represented by the M4G atlas does not contain cracks. Inside each chart, the corresponding surface region is parametrized separately, assembling a piece-wise parametrization of the entire surface. This first component of the M4G representation, explained in Section 4.1, is static and can be stored as a texture in the GPU.

The second component is dynamic and comprises an adaptive subdivision scheme based on tilings, more specifically 4–8 tilings [VZ01], covering the surface. Each tile is associated with a chart of the atlas and its structure is a pair of triangles forming a topological-square block divided along one of its diagonals. The surface tiling can be refined adding more tiles per chart or simplified returning to one tile per chart at the base multiresolution level. The refinement and simplification procedures create an adaptive mesh representation based on two stellar operators: edge split and weld. With each edge-split or edge-weld operation, the manifold resolution increases or decreases, according to regions of interest. The M4G tiling, further detailed in Section 5.2, is controlled by the CPU and can be stored as a halfedge data structure.

The two components of the M4G surface representation define a multiresolution pyramid with interesting properties. At the base level, the CPU has global information about the surface and controls how the tiles, in the form of quadrangular patches, will be sent to the GPU. At the top level, the GPU has local information about the surface and controls how each patch will be tessellated. This hierarchical representation fits the different computational granularities of both units. On the one hand, the CPU plays the role of a controller stipulating the minimum subdivision level of the surface. On the other hand, the GPU works at a finer granularity pushing forward the subdivision level differently for each surface region as needed. To illustrate the different aspects of our representation, we present a visualization application in Section 6 combining both computational paradigms to adaptively render geometric models.

4.1. Building the M4G Atlas

Although we think of the M4G atlas as a representation-independent, abstract entity, we need to make it "concrete" in order to compute the chart domain boundary curves and chart maps. This computation is obviously representation-dependent. Therefore, for the purpose of this paper, we work with a surface representation that is a dense triangle mesh M. The first step to build the M4G atlas is to partition the domain. This reference mesh, called M_0, corresponds to a coarse mesh computed by decimation of M. There are various ways to reduce the number of elements of M using local operators, such as vertex removal or edge collapse [Hop96]. For simplicity, we use the 4-K method [VG00] (decimating M to approximately 0.5%) which has the property of conserving the vertices of the original mesh. For the rest of the paper, we will refer to M as the dense mesh and M_0 as the associated base mesh.

5. Adaptive Multiresolution Mesh Representation

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 5.1. Then the triangles of the coarse mesh are paired to form *tiles*, producing a hierarchical 4-k mesh, explained in section 5.2. The result is a triquad mesh equipped with information of the input mesh. This information produces two coupled data structures, explained in section 5.3, used to reconstruct the surface in section 5.4.

5.1. Simplifying the Mesh

The simplification method we use follows the four-face cluster algorithm described in [Vel01]. 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 1) 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 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 [GH97]. Our simplification method uses only flip and weld operators, later we explain how to use the split operator for adaptive-resolution refinement.

On each step of the simplification process, the removed vertex is parameterized 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 [LSS*98], but it is simpler to implement and achieves better results due to the locality of the stellar simplification operators (see figure 2 for an example of a simplified mesh).

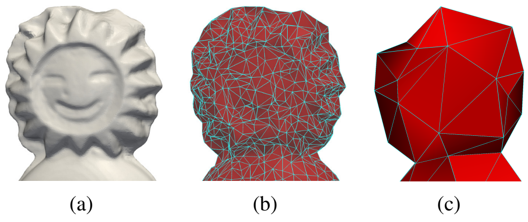

Figure 2: The head of the Kikito scanned model (a) is simplified using stellar operators to 2% of the original mesh (b) and 0.08% in the final coarse mesh (c).

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. There are two important properties in this parametrization. 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.

5.2. 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 [DILS*11].

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 and finding it is equivalent to pairing all mesh triangles in such a way that every triangle belongs to only one pair (see an example in figure 3). 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, i.e. a perfect matching whose sum of edge weights of paired triangles is maximum over all perfect matchings.

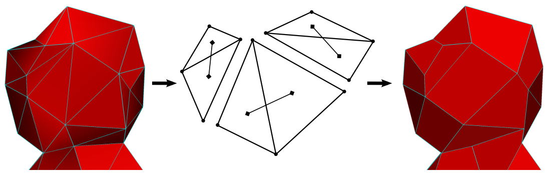

Figure 3: The head of the Kikito coarse mesh is covered with tiles using triangle pairing. The result is our base quad mesh.

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, 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 is concerned with 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 possible 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 coupled regular structure.

5.3. 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 atlas, 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 (see section 5.2) 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 4 (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 4 (right), respecting the texture-mapping design of the GPU. The start and end positions are stored as two (u,v) coordinates. This index and position coordinates are important to ensure a one-to-one correspondence between mesh and atlas. Finally, we store on each vertex a *resolution level* (set to 0 for base-mesh vertices) to control the mesh adaptivity. This is done similarly to the 4-8 subdivision method [VZ01] 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.

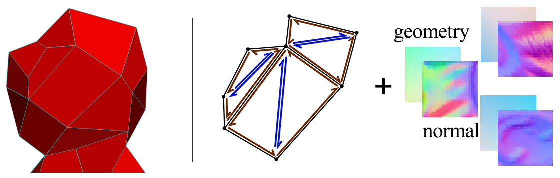

Figure 4: The base mesh (left) is converted to a specialized halfedge data structure (center) and a collection of regular charts (right), one for each base-mesh quad face.

The second data structure is a set of charts, mentioned above, in 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 (see section 5.1) 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. 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 [SWG*03]. We rasterize vertex attributes, such as geometry and normals (see examples in figure 4), 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 coupled produce 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 reconstruction.

5.4. Reconstructing the Surface

The adaptive 4–8 structure in the M4G tiling is responsible to determine the edges (vertices) of the current mesh that can be split (welded). Initially, the *internal edges*, i.e. each diagonal edge paired inside a tile, are the only edges that can be used to refine the current mesh. None of the initial vertices in M_0 can be welded and are set to be at level zero. The split operation inserts a new vertex at the next proper level i inside the corresponding tile, generating the next mesh M_i and changing the configuration of that tile (see an example in Figure 5). Analogously, the weld operation removes a vertex from inside a tile. Stellar subdivision refinement and simplification procedures induce a variable-resolution lattice, where the coarse (dense) mesh is the source (sink) node. Any cut in this graph is an adapted mesh and, since the M4G pyramid contains initially a semi-regular formation, it is not necessary to store the whole graph, only the current mesh is required to walk anywhere in the multiresolution pyramid [Vel04]. Moreover, the adaptive stellar subdivision operators enforce an invariance of one level maximum difference between adjacent faces. This feature produces a smooth resolution transition when adapting the mesh.

One of the main advantages of the stellar subdivision is that the adaptive mechanism can be completely general. The adaptation function is defined on complementary coverings of the mesh by quadrilateral regions, one for simplification associated with the star of weld vertices and another for refinement associated with the star of split edges. The function ranks each region accordingly to an arbitrary criterion evaluated over the interior of the region. The adaptation is done by establishing a threshold on the scalar range of rank values. The CPU adaptation is implemented through two heaps, one for simplification and another for refinement.

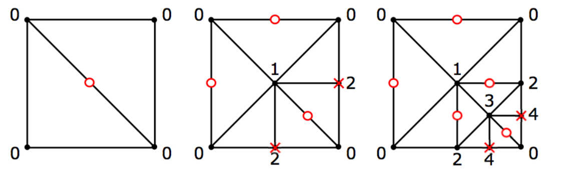

Figure 5: Illustrative example of the 4–8 structure with the corresponding vertex levels. The M4G tiling is refined (from left to right) creating intermediary adaptive meshes. At each resolution level, only a set of edges can be split (with a circle) and a set of vertices can be welded (with a cross).

The M4G tiling resulting from the CPU adaptation constitutes the minimum subdivision level imposed to the GPU. Thanks to the subdivision operator one-level invariance, it is possible to create a correspondence between the current adapted mesh and a restricted quadtree that is mapped to the GPU. From the current resolution, the GPU is free to refine the mesh further with a different adaptation criterion, as long as this criterion is consistent at the edges. This guarantees that the resulting adaptive tessellation will be crack-free.

Note the complementary nature of our CPU–GPU adaptation mechanism: the CPU adaptation has access to global surface information and is evaluated over the two-dimensional region on the interior of patches, while the GPU adaptation has only access to local patch information and is evaluated over their boundaries.

Figure 6 shows three examples of minimum subdivisions stipulated by the CPU (without any further subdivisions done by the GPU). The first (a) is the initial tiling coverage of the triquad mesh M_0. The second (b) and third (c) show subsequent refinements increasing the minimum subdivision level propagated across the mesh.

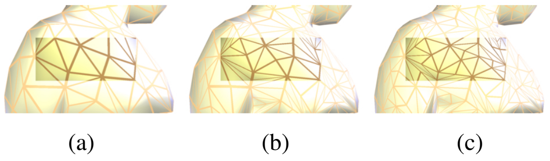

Figure 6: Three basic tiles (a) are refined progressively from right to left (b), showing that the 4–8 subdivision transition is smooth (with maximum one level difference). Refining the neighborhood also affects the three tiles (c).

To specify the patches, the CPU uses a subset of vertices and edges from M_i as representatives of a GPU patch (the 7 possible configurations are shown in Figure 7). Initially, the interior edge (or the vertex generated by splitting it) represents one patch. After a split operation at any border edge, the tile is considered to be 4 patches (as if all border edges were split). This configuration changes the representation to the four new interior edges. These interior edges are treated as new tiles and the process restarts.

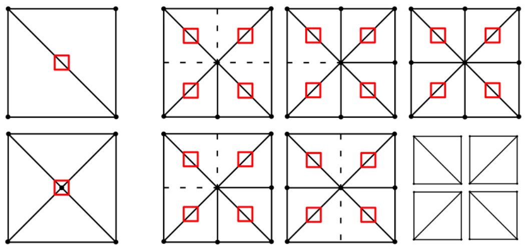

Figure 7: Adaptive M4G structure. While the two basic tile configurations (left) in the CPU are sent to the GPU as one patch, the next five (right) are sent as 4 patches. The elements (vertex or edge) in the CPU data structure representing a patch in the GPU data buffer are denoted with a square.

The required information to bind the CPU–GPU computation is the patch data of M_i. This data consists of the parametric coordinates (u, v) in the M4G atlas, the size of the patch (s) in the parametric domain and a patch codification (c) describing the minimum and maximum subdivision levels for each edge of the patch and its interior. And optionally, a chart index i depending on how the atlas is stored on the GPU. Since the 4–8 subdivision structure ensures a level transition of at most 1, the minimum subdivision level can be only 0 or 1 and results in a crack-free reconstruction. Additionally, the maximum level is used to control the resolution transition across the mesh. Because the M4G tiling is only topological, each patch can be encoded as one vertex with four attributes: (u, v), s and c. The surface properties at each patch, e.g. geometry and normals, are read from the M4G atlas as needed.

The combined data structures with patch definitions and subdivision rules constitute our adaptive multiresolution representation. Figure 9 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.

The basic adaptation mechanism is shown in Algorithm 1 following the above description of the M4G representation. After initializing the CPU and GPU components (lines 1 and 2), the algorithm enters in a loop with two inter-dependent components: the M4G tiling adaptation in the CPU (line 6) and the patch tessellation in the GPU (line 4). Because of the different granularities of the adaptation, the GPU patches are only updated (line 8) when the CPU adaptation changes (line 7). Observe that any external data (line 5), such as mouse or simulation events, can be used to influence the adaptation through the rank functions. Note also that complementing the tessellation, any processing can be performed by the GPU on the resulting elements (line 4), such as drawing or a numeric simulation.

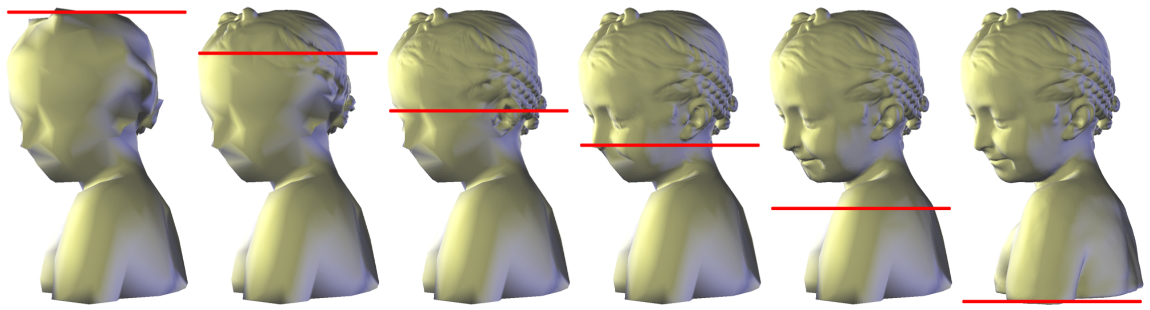

Figure 8: Example application of the Bimba dataset with variable mesh resolution. The surface features of the reconstructed mesh is revealed (from left to right) as the control plane (in red) moves from top to bottom.

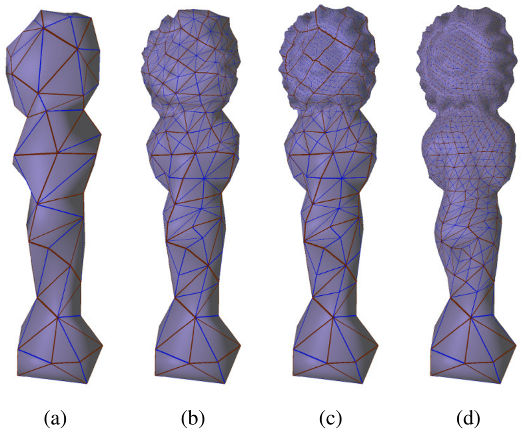

Figure 9: Final reconstruction of the Kikito statue: its base triquad mesh (a) is adaptively refined from bottom to top in the CPU (b); the head of the model is further refined in the GPU (c); and in (d) a more detailed refinement is imposed by the CPU. Boundary edges (in brown) and interior edges (in blue) denote the patches resolution level on each stage.

This basic algorithm can be extended to further exploit the granularities of the CPU–GPU adaptation. For instance, if the external data is changing slowly, it is not necessary to evaluate the CPU adaptation at every step and this can be estimated based on the rate of change of the data. Moreover, the CPU can do an interval calculation to predict the upper and lower bounds of the adaptation in the GPU and control it by changing the M4G tiling minimum/maximum levels accordingly.

6. Applications

Multiresolution meshes have a wide variety of applications, from real-time collision detection and rendering to high-definition multi-scale texture mapping. Multiresolution with

Algorithm 1 Basic CPU–GPU adaptation.

```
 1: Initialize CPU adaptation
 2: Setup GPU patches
 3: loop
 4:   GPU tessellation and processing
 5:   Get external data
 6:   CPU simplify/refine
 7:   if M4G Tiling changes then
 8:     Update GPU patches
 9:   end if
10: end loop
```

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 three illustrative applications of our mesh representation.

After the applications, we discuss our atlas and tiling data structures providing an assessment of the M4G surface representation.

CPU–GPU adaptation The first application uses the new GPU tessellator control and evaluation shaders (introduced in OpenGL 4) to visualize scanned models converted to our M4G representation. We use two complementary adaptation criteria for defining the CPU and GPU mesh resolution. For the CPU, we have used a Laplacian-based criterion to rank simplification vertices with small discrete curvature variation. And for the GPU, we have used a LOD-based criterion, which tessellates more edges close to a given plane and tessellates the interior of the patches using the average tessellation level of opposite edges. Although not challenging in terms of performance (all results are real-time independent of subdivision and tessellation levels), the surface reconstruction can be as close to the original surface as the parametrization of each chart is. The geodesics-based parametrization gives interesting results as a proof-of-concept for the tested models. On the flip side, the geodesics-based boundary computation, explained in Section 4.1, allows two important characteristics of the visualization application: a simple and efficient implementation of the M4G tiling approach in the CPU; and a coupled GPU implementation using tessellator shaders. It is clear, from the choices used for this example application, that the M4G surface representation is capable of multiresolution adaptive tessellation, both in the CPU and GPU, for generic scanned models.

One illustrative example of the GPU tessellation using the M4G atlas and controlled by the CPU is shown in Figure 8. In this figure, the Bimba dataset features are revealed interactively by moving the plane controlling the level-of-detail (in red) from top to bottom. Another illustrative example, this time with both CPU and GPU adaptation, appears in Figure 10. In this example, the Botijo model is shown without adaptation (left), with only CPU adaptation (middle) and with both CPU and GPU adaptation (right). A last example is shown in Figure 11, where the Christ dataset is increasingly adapted from far to close, showing a zoom-in view-dependent coupled LOD-based adaptation.

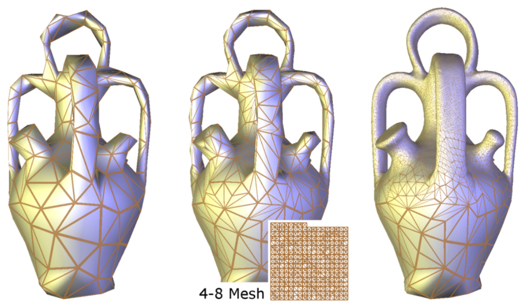

Figure 10: Example of the CPU–GPU coupled visualization of the Botijo dataset. The base mesh (left) is refined adaptively to a certain level (middle) in the CPU, generating a 4–8 mesh. Finally this mesh is further refined from middle to top (right) in the GPU.

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 9b, and in the GPU, with view-dependent tessellation refinement. Figure 12 illustrates this cascading LOD applied to a checkerboard of meshes rendered in real-time.

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 13 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.

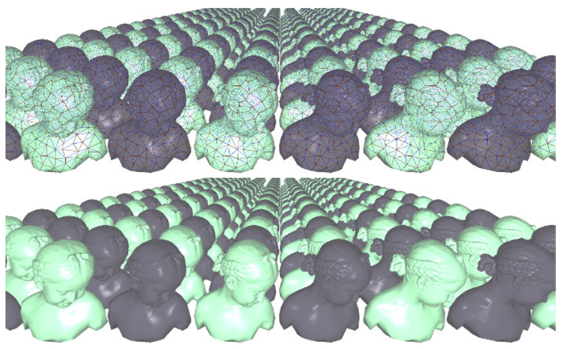

Figure 12: The Bimba model is adaptive refined in both the CPU and the GPU (top) to be visualized with LOD (bottom).

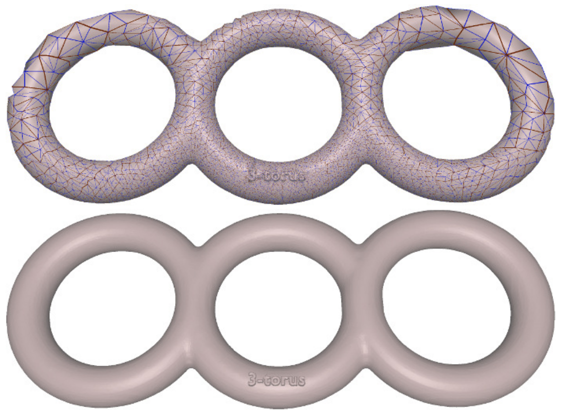

Figure 13: The 3-torus model is edited using adaptive refinement (top) to then be rendered in full detail (bottom).

7. 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 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, reducing atlas efficiency. The replication of boundaries inside charts also slightly reduces the atlas efficiency.

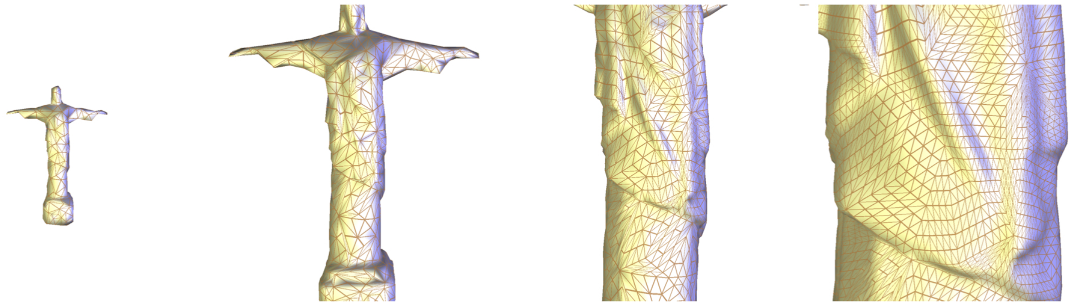

Figure 11: LOD-based application example of the Christ dataset simulating a zoom in the middle of the statue (from left to right). Note the increase in creases appearing in the mantle.

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.
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%

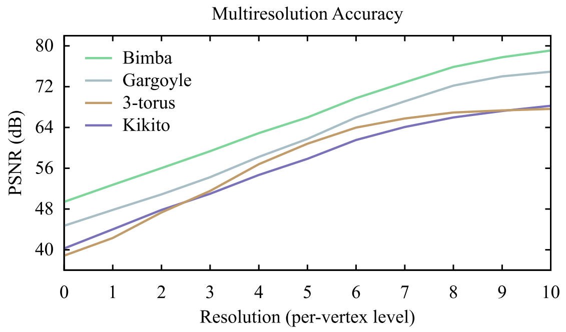

Figure 14: Precision of our multiresolution representation.

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 compared with a regular halfedge data structure is 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.

We express geometric accuracy in our multiresolution representation as Peak Signal-to-Noise Ratio (PSNR), following MCGIM [SWG*03]. 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 level-10 mesh has approximately the same number of vertices and triangles of the dense mesh. Figure 14 shows the measured PSNR for each tested model.

Analyzing the PSNR results, the highest resolution reconstruction of the Gargoyle model is 8.9dB lower than using MCGIM (see [SWG*03, figure 14]). The geometry accuracy is lower given the fact that our chart domains are square 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 a similar optimization procedure.

Compared against MAPS [LSS*98], our simplification step is based on more *atomic* operations, i.e. stellar operators, that can reproduce both PM [Hop96] and MAPS simplification. Our full-resolution reconstruction ranges from 0.5% maximum error (for the 3-torus model) to 0.3% (for the Bimba model), improving on the *remeshing tolerance* of MAPS. In contrast with MAPS, our reconstruction uses the inverse set of atomic operators, allowing the connection we present between the CPU and the GPU resolution control.

8. 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 three applications: CPU-GPU adaptation; rendering hundreds of meshes with LOD; and editing a small part of an adaptive mesh.

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

Our framework has great potential for practical applications and a wide perspective for continuing development, as our list of topics for future work indicates.

In that respect, we plan to extend this research in the following directions:

- Evaluate other ways to construct the M4G representation from dense triangle meshes. This includes both alternative methods to generate the base mesh, such as quadrilateral simplification, surface segmentation and manual patch layout, as well as, different types of local and global parametrization algorithms.
- Exploit the characteristics of particular surface types, such as subdivision surfaces, sketch-based and procedural models, that can be directly evaluated on the GPU.
- Investigate the implementation of multiresolution surfaces that can be used for compression and level-of-detail editing.
- Augment the multiresolution representation with a mesostructure layer defined by surface normal modulation.
- Develop applications that take advantage of joint parametrizations of the M4G atlas, such as surface morphing and animation.
- Study more complex and effective adaptation functions, targeted to different applications, such as view and texture dependent mechanisms.
- Integrate numerical simulation over the surface using GPGPU techniques.
- Work with very high resolution scanned models, such as the David model from Digital Michelangelo project and also with huge 3D datasets generated by structure-from-motion reconstruction, where the model can be an entire city with millions of points. Within this context our framework could pave the way of a comprehensive multiresolution representation with flexible adaptation mechanism suitable for giga-elements models.

Acknowledgments

We would like to acknowledge the grant of the first author provided by Brazilian agency CNPq (National Counsel of Technological and Scientific Development).

References

[AB08] Alexa M., Boubekeur T.: Subdivision shading. *ACM Trans. Graph. 27* (December 2008), 142:1–142:4. 3

[BBB*97] Bloomenthal J., Bajaj C., Blinn J., Cani-Gascuel M.-P., Rockwood A., Wyvill B., , Wyvill G. (Eds.): *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997. 1, 2

[BFK84] Böhm W., Farin G., Kahmann J.: A survey of curve and surface methods in CAGD. *Computer Aided Geometric Design 1* (July 1984), 1–60. 1, 2

[BK03] Botsch M., Kobbelt L.: Multiresolution surface representation based on displacement volumes. *Computer Graphics Forum 22*, 3 (2003), 483–491. 1

[BPK*07] Botsch M., Pauly M., Kobbelt L., Alliez P., Lévy B., Bischoff S., Rössl C.: Geometric modeling based on polygonal meshes. In *ACM SIGGRAPH 2007 courses* (New York, NY, USA, 2007), SIGGRAPH '07, ACM. 1, 2

[BVK10] Bommes D., Vossemer T., Kobbelt L.: Quadrangular parameterization for reverse engineering. In *Mathematical Methods for Curves and Surfaces*, DÃęhlen M., Floater M., Lyche T., Merrien J.-L., MÃÿrken K., Schumaker L., (Eds.), vol. 5862 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2010, pp. 55–69. 2, 3

[CC78] Catmull E., Clark J.: Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design 10*, 6 (1978), 350–355. 2, 3

[CKJ02] Choi B. K., Kim B. H., Jerard R. B.: Sculptured surface nc machining. In *Handbook of Computer Aided Geometric Design*, Farin G., Hoschek J., Kim M.-S., (Eds.). Elsevier Science, Amsterdam, The Netherlands, 2002. 2

[dGGDV11] de Goes F., Goldenstein S., Desbrun M., Velho L.: Exoskeleton: Curve network abstraction for 3d shapes. *Computers and Graphics 35*, 1 (2011), 112 – 121. Extended Papers from Non-Photorealistic Animation and Rendering (NPAR) 2010. 3

[DILS*11] Daniels II J., Lizier M., Siqueira M., Silva C. T., Nonato L. G.: Template-based quadrilateral meshing. In *Proceedings of the Shape Modeling International Conference (SMI'11)* (June 22-24 2011), IEEE Computer Society. (to appear). 2, 3, 5

[DMA02] Desbrun M., Meyer M., Alliez P.: Intrinsic parameterizations of surface meshes. *Computer Graphics Forum 21*, 3 (2002), 209–218. 3

[Ede05] Edelsbrunner H.: Surface tiling with differential topology. In *Proceedings of the third Eurographics symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2005), Eurographics Association. 1

[FCS*10] Fogal T., Childs H., Shankar S., Krüger J., Bergeron R. D., Hatcher P.: Large data visualization on distributed memory multi-gpu clusters. In *Proceedings of the Conference on High Performance Graphics* (Aire-la-Ville, Switzerland, Switzerland, 2010), HPG '10, Eurographics Association, pp. 57–66. 1

[FFB*09] Fisher M., Fatahalian K., Boulos S., Akeley K., Mark W. R., Hanrahan P.: Diagsplit: parallel, crack-free, adaptive tessellation for micropolygon rendering. *ACM Trans. Graph. 28* (December 2009), 150:1–150:10. 3

[FHR02] Floater M., Hormann K., Reimers M.: Parameterization of Manifold Triangulations. In *Approximation Theory X: Abstract and Classical Analysis* (2002), Citeseer, pp. 197–209. 3

[FP08] Fan J., Peters J.: On smooth bicubic surfaces from quad meshes. In *Proceedings of the 4th International Symposium on Advances in Visual Computing* (Berlin, Heidelberg, 2008), ISVC '08, Springer-Verlag, pp. 87–96. 3

[GH95] Grimm C. M., Hughes J. F.: Modeling surfaces of arbitrary topology using manifolds. In *Proceedings of the 22nd* *annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1995), SIGGRAPH '95, ACM, pp. 359–368. 2

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 209–216. 5

[Hop96] HOPPE H.: Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 99–108. 2, 3, 4, 10

[HSH10] HU L., SANDER P. V., HOPPE H.: Parallel view-dependent level-of-detail control. *IEEE Transactions on Visualization and Computer Graphics 16* (2010), 718–728. 3

[HZM*08] HUANG J., ZHANG M., MA J., LIU X., KOBBELT L., BAO H.: Spectral quadrangulation with orientation and alignment control. *ACM Trans. Graph. 27* (December 2008), 147:1–147:9. 2, 3

[KL96] KRISHNAMURTHY V., LEVOY M.: Fitting smooth surfaces to dense polygon meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 313–324. 2

[KNP07] KÄLBERER F., NIESER M., POLTHIER K.: Quadcover - surface parameterization using branched coverings. *Computer Graphics Forum 26*, 3 (2007), 375–384. 2

[Kob00] KOBBELT L.: $\sqrt{3}$-subdivision. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 103–112. 2, 3

[Loo87] LOOP C. T.: *Smooth subdivision surfaces based on triangles*. Master's thesis, Department of Mathematics, University of Utah, Salt Lake City, Utah, USA, 1987. 2, 3

[LSS*98] LEE A., SWELDENS W., SCHRÖDER P., COWSAR L., DOBKIN D.: MAPS: Multiresolution Adaptive Parameterization of Surfaces. In *Proc. ACM/SIGGRAPH Conf.* (1998), pp. 95–104. 5, 10

[LTD05] LEE H., TONG Y., DESBRUN M.: Geodesics-based one-to-one parameterization of 3d triangle meshes. *IEEE MultiMedia 12* (January 2005), 27–33. 3

[Män88] MÄNTYLÄ M.: *An introduction to solid modeling*. Computer Science Press, 1988. 2

[MTAD08] MULLEN P., TONG Y., ALLIEZ P., DESBRUN M.: Spectral conformal parameterization. In *Proceedings of the Symposium on Geometry Processing* (Aire-la-Ville, Switzerland, Switzerland, 2008), SGP '08, Eurographics Association, pp. 1487–1494. 3

[MYV93] MAILLOT J., YAHIA H., VERROUST A.: Interactive texture mapping. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1993), SIGGRAPH '93, ACM, pp. 27–34. 2

[PEO09] PATNEY A., EBEIDA M. S., OWENS J. D.: Parallel view-dependent tessellation of catmull-clark subdivision surfaces. In *Proceedings of the Conference on High Performance Graphics 2009* (New York, NY, USA, 2009), HPG '09, ACM, pp. 99–108. 3

[RLL*06] RAY N., LI W. C., LÉVY B., SHEFFER A., ALLIEZ P.: Periodic global parameterization. *ACM Trans. Graph. 25* (October 2006), 1460–1485. 2, 3

[SJP05] SHIUE L.-J., JONES I., PETERS J.: A realtime gpu subdivision kernel. *ACM Trans. Graph. 24* (July 2005), 1010–1015. 1, 3

[SS09] SCHWARZ M., STAMMINGER M.: Fast gpu-based adaptive tessellation with cuda. *Computer Graphics Forum 28*, 2 (2009), 365–374. 1, 3

[SSG03] SIFRI O., SHEFFER A., GOTSMAN C.: Geodesic-based surface remeshing. In *In Proc. 12th Intnl. Meshing Roundtable* (2003), pp. 189–199. 3

[SWG*03] SANDER P. V., WOOD Z. J., GORTLER S. J., SNYDER J., HOPPE H.: Multi-chart geometry images. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2003), SGP '03, Eurographics Association, pp. 146–155. 2, 6, 10

[SXG*09] SIQUEIRA M., XU D., GALLIER J., NONATO L. G., MORERA D. M., VELHO L.: Technical section: A new construction of smooth surfaces from triangle meshes using parametric pseudo-manifolds. *Comput. Graph. 33* (June 2009), 331–340. 2

[TPC*10] TARINI M., PIETRONI N., CIGNONI P., PANOZZO D., PUPPO E.: Practical quad mesh simplification. *Computer Graphics Forum (Special Issue of Eurographics 2010 Conference) 29*, 2 (2010), 407–418. 2, 3

[Tu07] TU L. W.: *An Introduction to Manifolds*. Springer, New York, NY, USA, 2007. 1

[Vel01] VELHO L.: Mesh Simplification using Four-Face Clusters. In *Proc. SMI* (May 2001). 4

[Vel03] VELHO L.: Stellar Subdivision Grammars. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2003), SGP '03, Eurographics Association, pp. 188–199. 2, 4

[Vel04] VELHO L.: A dynamic adaptive mesh library based on stellar operators. *Journal of Graphics Tools 9*, 2 (2004), 1–29. 3, 6

[VG00] VELHO L., GOMES J.: Variable resolution 4-k meshes. In *Computer Graphics and Image Processing, 2000. Proceedings XIII Brazilian Symposium on* (2000), pp. 123–130. 3, 4

[VZ01] VELHO L., ZORIN D.: 4-8 Subdivision. *Computer Aided Geometric Design 18*, 5 (2001), 397–427. 2, 3, 4, 6

[Whi40] WHITEHEAD J. H. C.: On c1-complexes. *The Annals of Mathematics - Second Series 41* (1940), 809–824. 2

[YZ04] YING L., ZORIN D.: A simple manifold-based construction of surfaces of arbitrary smoothness. *ACM Trans. Graph. 23* (August 2004), 271–275. 2

[ZHX*07] ZHOU K., HUANG X., XU W., GUO B., SHUM H.-Y.: Direct manipulation of subdivision surfaces on gpus. *ACM Trans. Graph. 26* (July 2007). 3

[Zor00] ZORIN D.: Subdivision for modeling and animation. *SIGGRAPH 2000 Course Notes ACM SIGGRAPH* (2000). 1

[ZSS97] ZORIN D., SCHRÖDER P., SWELDENS W.: Interactive multiresolution mesh editing. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 259–268. 3