Volume xx (200y), Number z, pp. 1–12

Implicit Hierarchical Quad-Dominant Meshes

Daniele Panozzo† and Enrico Puppo‡

DISI, University of Genoa, Genoa, Italy

Abstract

We present a method for producing quad-dominant subdivided meshes, which supports both adaptive refinement and adaptive coarsening. A hierarchical structure is stored implicitly in a standard half-edge data structure, while allowing us to efficiently navigate through the different level of subdivision. Subdivided meshes contain a majority of quad elements and a moderate amount of triangles and pentagons in the regions of transition across different levels of detail. Topological LOD editing is controlled with local conforming operators, which support both mesh refinement and mesh coarsening.We show two possible applications of this method: we define an adaptive subdivision surface scheme that is topologically and geometrically consistent with the Catmull-Clark subdivision; and we present a remeshing method that produces semi-regular adaptive meshes.

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

1. Introduction

Polygonal modeling is the main modeling paradigm for applications that require computational intensive tasks other then rendering, such as video games and finite element methods. In this context, quad-based meshes are often preferred to triangle-based ones, since they provide a more stable and better controllable framework for texturing, modeling and geometric computations. One notable property of quads is the possibility to be naturally aligned to anisotropic design features, as well as to line fields, or cross fields, such as those corresponding to principal curvatures [BZK09, DBG*06, KNP07, RLL*06].

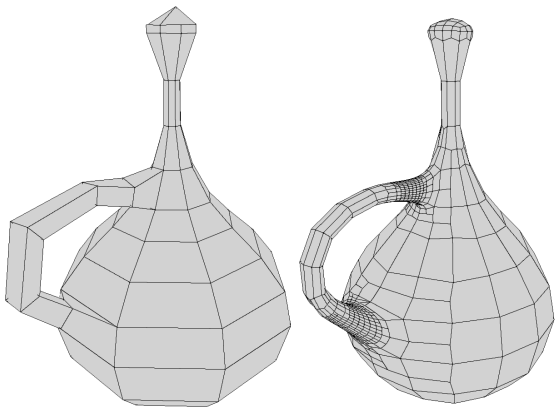
A standard approach to polygonal modeling consists of starting from a coarse base mesh, which is then interactively edited and refined to model the features of the desired shape. One main goal is to obtain a meshe having a controlled budget of polygons, while being close to an ideal smooth surface. Mesh subdivision is often used to this purpose [MS01]. Non-trivial shapes may require adaptive subdivision to model different parts: tiny but relevant features will require a much finer mesh than large uniform areas. But subdivision is generally meant as a global process, while adaptive refinement of quad meshes is non trivial: local refinement of quads produces non-quad faces and this process, if performed in an uncontrolled manner, can soon destroy the regular structure of a mesh.

On the other hand, several authors have remarked that quad-dominant meshes containing a small amount of non-quad elements can be more flexible and more effective than purely quad meshes in capturing surface features, and they may enrich the design space [MNP08, SL03]. For instance, triangular and pentagonal elements can be used to collapse, split and merge lineal features and line fields, as well as to model the surface in the proximity of singularities.

In this paper, we propose a method to adaptively refine a quad mesh through local operators for both mesh refinement and mesh coarsening (see Figures 10 and 1). Our method generates an implicit hierarchy of adaptively refined quad-dominant meshes, each containing a small amount of triangular and pentagonal transition elements. The adaptive subdivision patterns preserve the surface flows and lineal fea-

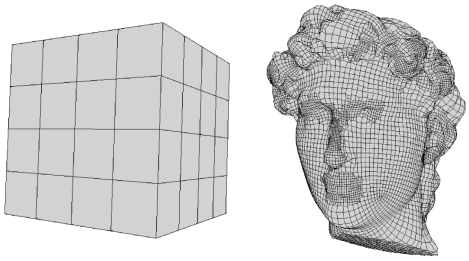
† e-mail: panozzo@disi.unige.it

‡ e-mail: puppo@disi.unige.it



(a) (b)

Figure 1: *Subdivision surface: (a) The control mesh of a smooth object, is adaptively subdivided (b) and its vertices are moved to their limit position.*



(a) (b)

Figure 2: *Remeshing: (a) A simple mesh used to impose the base topology; (b) an adaptively remeshed model is obtained by selective refinement and projection of vertices to the reference shape.*

tures, which are defined on the base mesh, at all meshes in the hierarchy.

Our hierarchy is *implicit,* meaning that only a mesh at any intermediate level of refinement is encoded at each time, while all other meshes of the hierarchy can be easily obtained from it, by means of local conforming operators, which support both refinement and coarsening and work on a plain mesh without the need of cumbersome hierarchical data structures. We also provide traversal operators that work on an adaptively refined mesh M, while supporting navigation of any mesh coarser than M in the implicit hierarchy.

We present two possible applications of our framework: an adaptive subdivision surface scheme that is consistent with the standard Catmull-Clark scheme; and a remeshing method that, starting from a sketched base mesh and a target shape, allows to produce a quad-dominant mesh with semi-regular topology and the geometry of the target shape.

2. Related work

Adaptive subdivision: Mesh refinement in polygonal modeling is most often based on subdivision patterns. Classical patterns, such as the face quadrisection, used in classical subdivision schemes, are meant to be applied to all faces of a mesh together, and generate non-conforming meshes when applied selectively. *Red-green triangulations* [BSW83] support adaptive refinement of triangle meshes by applying triangle quadrisection adaptively, and then subdividing some triangles further, to fix non conforming situations. Variants of red-green triangulations were developed in [PS07,ZSS97] to support multi-resolution editing of meshes and they adopt either the Loop or the butterfly subdivision schemes to set the geometry of the refined mesh. The RGB subdivision proposed in [PP09a, PP09b] extends red-green triangulations with the same subdivision schemes to a fully dynamic adaptive scheme supporting both local refinement and coarsening. The $\sqrt{3}$ subdivision [Kob00] and the 4-8 subdivision [VZ01] for triangle meshes are subdivision schemes based on different patterns that can be implemented through local conforming operators, making them naturally adaptive. In [KCB09], an extension of the half-edge data structure that allows the representation of multiresolution subdivision surfaces is presented. The multiresolution half-edges allow to adaptively refine a mesh, with either the Loop or the Catmull-Clark scheme. Adaptive methods similar in spirit to red-green triangulations were proposed for quad meshes in [MJ98] and [XK99], to extend the classical Catmull-Clark and Doo-Sabin subdivision schemes.

CLOD models: The interested reader may refer to [LRC*02] for a book on this subject. Generally speaking, a CLOD model consists of a base mesh at coarse resolution, plus a set of local modifications that can be applied to the base mesh to refine it. Such modifications are usually arranged in a hierarchical structure, which consists of a directed acyclic graph (DAG) in the most general case. Meshes at intermediate level of detail correspond to cuts in the DAG, and algorithms for selective refinement work by moving a front through the DAG and doing/undoing modifications that are traversed by this front. This general framework, as shown in [Pup98], encompasses almost all CLOD models proposed in the literature and it can be applied to the hierarchies generated by $\sqrt{3}$ subdivision and 4-8 subdivision as well. In [KL03], a CLOD model is introduced, which achieves better adaptivity by using local modifications more freely than in previous models. CLOD models can provide meshes at intermediate LOD, where detail can vary across the mesh and through time, at a virtually continuous scale and with fast procedures that work on-line even for huge meshes. The scheme proposed in [DWS*97] is very popular and most authors refer to it in order to implement their selective refinement algorithms. One generalization is given by 4-k meshes [VG00], which have in fact a strong relation with 4-8 subdivision [VZ01].

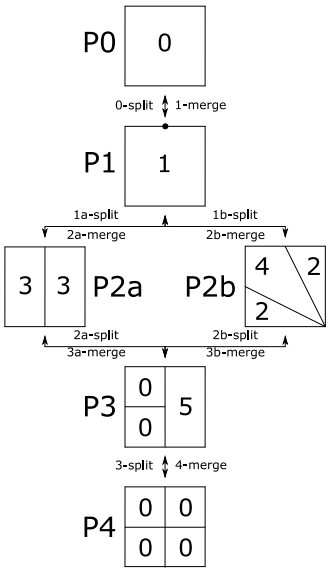


Figure 3: *The diagram of patterns for adaptive subdivision of a quad. Types of faces and patterns are denoted by labels placed inside and beside them, respectively. Transitions between adjacent patterns are labeled with the corresponding refinement and coarsening operators.*

3. Adaptive quad subdivision

We deal with manifold polygonal meshes containing triangles, quads and pentagons, adopting standard terminology. A *quad mesh* is a mesh where all faces are quads; a *quad-dominant* mesh is a mesh where most faces are quads. A vertex v in a quad mesh (or in a quad-dominant mesh containing just quads in the star of v) is *regular* if it has valence four; otherwise it is said to be *extraordinary*. The *edge neighbors* of a vertex v in a quad[-dominant] mesh are those vertices connected to v through edges; the *face neighbors* of v are those vertices opposite to v on its incident quads.

We edit the mesh by local refinement operations that split one edge by inserting a vertex, and local coarsening operations that merge a pair of edges by removing a vertex. A local operation eliminates faces in the neighborhood of the vertex to be inserted/removed, and re-tessellates the hole with new faces. This will be the subject of Section 3.1. The iterative application of local operators define an implicit hierarchy, described in Section 3.2. In Section 3.3 the concept of topological angles and lenghts are introduced and used to define navigation algorithms that allows us to navigate through the subdivided mesh and across the different levels of the implicit hierarchy.

3.1. Topological operators

The most common pattern for uniform subdivision of quad meshes is *quadrisection*: each face is subdivided into four new faces by splitting each edge with a new vertex, and connecting each such vertex with another new vertex at the center of the face. In the following, a vertex that splits an edge will be said to be of type E, while a vertex inserted in the middle of a face will be said to be of type F.

Since quadrisection splits all edges of a face, it cannot be applied selectively while maintaining the mesh conforming. In order to support transitions between different levels of subdivision, we devise alternative patterns that split one edge at a time. Figure 3 illustrates such a set of patterns, and related operators. These patterns produce quad-dominant meshes containing some triangular and pentagonal faces, which preserve the flow of lines of the base mesh (see Section 3.4 for a discussion). Other patterns, e.g. containing just quads and triangles could be also used with straightforward modifications of the method described below.

A key idea is that local operators subdivide a mesh by splitting one edge at a time, they always produce conforming triangulations, and they can be controlled just on the basis of attributes of local entities, i.e., types and levels of vertices, edges and faces. All rules that control operators are purely topological. Just for the sake of clarity, in the figures we will use fixed shapes to depict the different types of faces that may appear in our adaptive meshes: squares (type 0); rectangles(type 3); diamonds (type 4); right triangles (type 2); and pentagons with three collinear vertices, having the shape of either a square or a rectangle (type 1 and 5, respectively).

Consider a quad mesh Γ_0, called the *base mesh.* We assign level zero to all its vertices and edges, and type *standard* to all its edges. A selectively refined mesh will also contain vertices and edges labeled according to their level of subdivision; edges will be labeled with either type *standard* or type *extra.* The only extra edges are those internal to patterns P2a and P2b, which have the same level l of their parent face (i.e., face 0 in pattern P0); the remaining edges are standard and they have level either l or $l+1$, as in the standard subdivision. The level of a face in a mesh Γ is defined to be the lowest among the levels of its edges. Note that the type of a face is uniquely defined by the types and levels of its edges, and the type of a pattern is also uniquely defined by levels of edges in its boundary.

3.1.1. Refinement operators

According to definitions above, all faces in the base mesh are standard at level zero. Local subdivision operators can be applied iteratively to Γ_0 to generate a conforming mesh Γ composed of faces of the six types illustrated in Figure 3.

We say that an edge e at level $l \geq 0$ is *refinable* (i.e., it can split) if and only if it is standard and its two adjacent faces f_0 and f_1 are both at level l. In case of a boundary edge, only one such face exists. We split an edge e at level l, by inserting at its midpoint a new vertex v at level $l+1$. The edges generated by the two halves of e are standard at level

$l+1$. Note that levels of vertices and standard edges comply with the standard subdivision.

Splitting an edge e at level l may affect an area as large as that of the standard faces incident at e at level l. Tessellations on the two sides of e can be treated independently and each of them depends on the type of the face f incident at e and on its configuration. It is readily seen from Figure 3 that in all cases the type of face f incident at splitting edge e and the levels and labels of edges of f are sufficient to characterize the type of operator to be applied. This fact allows us to precompute and store in a lookup table the local tessellations to be deleted from, and to be plugged into a mesh. Note that all split operators affect the whole area covered by a pattern, except for 3-split, which affects just the area covered by face of type 5 in pattern P3. In fact, the other two (standard) faces at level $l+1$ may be actually refined independently at higher levels before this operator is applied.

By simple combinatorial analysis, it is easy to verify that the set of refinement operators is closed with respect to the meshes obtained, i.e.: if we start at a mesh Γ_0 containing all standard faces at level 0 and we proceed by applying any legal sequence composed of the operators above, the resulting mesh will be composed of faces of the types defined above, and all its refinable edges can be split through the same set of operators. In particular, all vertices of a standard subdivision up to a given level l can be added without adding any vertex of a level higher than l and the same uniform mesh generated from the standard subdivision scheme will be obtained.

If an edge e at level l is of standard type, but it is not refinable, we trigger recursive refinement of each face f incident at e and having a level $< l$. Recursive refinement is performed by recognizing the type of f and forcing refinement of either two (for pattern P1) or one (for all other patterns) of its edges at level $< l$.

3.1.2. Coarsening operators

Local merge operators invert edge split operators defined above, by removing one vertex v at level $l+1$ that splits an edge e at level l. As before, at most the area spanned by the faces incident at e at level l may be affected, and the tessellations of such two areas are treated independently.

A vertex v at level $l+1$ is *potentially removable* if the levels of its incident faces are: $l+1$ for standard faces (type 0), and l otherwise. A potentially removable vertex is *removable* if it is of type E (i.e., it splits an edge of the previous level) and faces in its neighborhood can be arranged to form two patterns of the diagram, sharing a pair of edges at level $l+1$ incident at v. Vertices of type F (i.e., splitting a face of the previous level) do not trigger any merge operator, because they are removed together with vertices of type E from operators 3a/b-merge. Such vertices are discarded easily because a potentially removable vertex v at level $l+1$ is of type F if and only if either it has exactly four adjacent vertices at level $l+1$, or its star is formed by two standard faces and one face of type 5.

We divide the neighborhood of (internal) vertex v of type E in two halves as follows: there are at most four (standard) edges at level $l+1$ incident at v; among them, only two edges e' and e'' have the other end vertex at level $\leq l$; thus the pair e', e'' cover the edge e that was split by v and they divide the neighborhood. For each half neighborhood, we recognize the pattern adjacent to the pair e', e'' and we apply the corresponding merge operator, as depicted in Figure 3. Operators 1-merge and 2a/b-merge can be applied by checking just the types of faces f' and f'' incident at e' and e''. In order to discriminate between operators 3a-merge and 4-merge it is also necessary to check the type of face(s) adjacent to f' and f'' inside the pattern. Finally, operator 3b-merge needs checking also the other face of type 0 adjacent to the face of type 5 within pattern P3: in fact, v is not removable if such a face has been refined further.

It is easy to verify that the merge operators are consistent with the split operators and they have similar properties: all merge operators affect the whole area covered by a pattern, except for 4-merge, which affects just half a pattern; local tessellations to implement operators are precomputed and stored in a lookup table (in fact, the same lookup table that is used for the split operators).

The set of refinement operators is also closed with respect to the meshes obtained. If we start at a mesh Γ obtained from Γ_0 through refinement, we can apply merge operators in any legal order to go back to Γ_0; moreover, any intermediate mesh could be refined through split operators. So we can mix split and merge operators in any order while preserving consistency.

3.2. Transition space and implicit hierarchy

Following the approach of [PP09b], it is possible to define a transition space for our adaptive subdivision scheme. A transition space is a graph where each node corresponds to an adaptive mesh, and each arc corresponds to the application of an atomic local operation, as defined in Section 3.1. In the graph, a path from node a to node b corresponds to a set of atomic operations that transform the mesh associated with a into the mesh associated with b. Since all operators are invertible, it is always possible to execute also the inverse sequence of operations. Note that multiple paths may exist between a pair of nodes.

Consequently, the application of the local operators to a base mesh defines a (virtually unlimited) subdivision hierarchy. We do not need to store the hierarchy explicitly. Starting from a single node n, we are able to determine what operations are valid and subsequently what arcs of the graph are incident with n. In other words, we always know the subdivision hierarchy locally, around our current mesh, and we are able to navigate the graph by applying local operators.

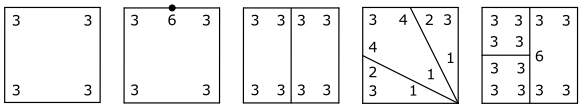


Figure 4: *Topological angles: a width is assigned to each vertex in each face.*

3.3. Topological angles and lengths

In order to support navigation efficiently, we assign a *topological width* to every angle defined by a pair Face-Vertex (F,V) in a mesh. The values are assigned to faces of the various types as indicated in Figure 4. An angle with topological width of 6 is said to be *flat*. Such values are not related to geometrical values, we call them "angles" since they satisfy some properties of geometrical angles, which we will show in the following. We do not need to store angle widths, since they can be found efficiently from types of faces and edges, and levels of vertices.

We give next some invariants on angles that will be useful for mesh navigation.

Lemma 1 If an edge e is split into two edges e_0 and e_1 by adding a vertex v, both angles formed by e_0 and e_1 are flat.

Proof. Figure 4 shows the only possible ways to split an edge. It is readily seen that in all cases the sum of angles on each side of a pair e_0e_1 is 6. □

Lemma 2 The width of a topological angle between a pair of edges is invariant upon editing operations on the mesh.

Proof. Consider a pair of edges e and e' incident at v and one of the two angles they form at v. It is sufficient to analyze editing operations that affect faces spanned by such an angle. For each such face f, there are three possible cases, which are readily verified by comparing transitions depicted in Figure 3 with angles depicted in Figure 4: If the editing operation neither splits f with an edge incident at v, nor merges t with an adjacent face around v, then the angle of f at v is unchanged; If the angle of f at v is split into two angles, then the sum of widths of such angles is equal to the width of the angle of f at v before split; If f is merged with another face f' adjacent to it around v, by deleting their common edge, then either e and e' are merged into a single edge, or the width of angle at v of the new face is equal to the sum of widths of angles of f and f' at v. □

Lemma 3 No matter how an edge e is subdivided into a chain of edges $e_0,\ldots,e_k$, angles between two consecutive edges e_{i-1} and e_i, $i=1,\ldots,k$ are flat.

Proof. The proof follows from the above two lemmas by noting that every split produces flat angles and such angles are invariant upon subsequent editing operations. □

Topological lengths are assigned to standard edges inductively: an edge at level 0 has unit length; an edge at level $l+1$ has half the length of an edge at level l.

The previous definitions and lemmas allow us to define a set of operators for mesh navigation, which help us extracting from an adaptively refined mesh a view of the same mesh at a lower level of subdivision. We define switch operators similar to those proposed in [Bri93], plus two new operators, called **rotate** and **move**, that are specific for our meshes. All operators use a unique identifier of position in a mesh, called a *pos*, which contains a vertex v, an edge e incident at v, and a face f bounded by e. Given a *pos* **p**, we will denote by **p.v**, **p.e** and **p.f** its related vertex, edge and face, respectively.

1. **p.switchVertex(), p.switchEdge()** and **p.switchFace()** move to the adjacent *pos* which differs from **p** just for the vertex, the edge and the face, respectively.
2. **p.rotate(i)**: executes an alternate sequence of **p.switchEdge()** and **p.switchFace()** operators until a topological angle of width **i** has been spanned.
3. **p.move(l)**: executes an alternate sequence of **p.switchVertex()**, possibly followed by **p.rotate(6)** and **p.switchFace()** operators until the length of an edge at level $\leq l$ has been traversed. If the first edge traversed has a level $< l$ (i.e., its length is larger than required) the operation has no effect.

The effect of operators is exemplified in Figure 9: single arrows correspond to switch and rotate operators; the expanded view shows the decomposition of a rotate(6) operator in terms of switches; sequences of vertical arrows correspond to move operators for the length of one edge at the coarsest level.

Lemma 4 The rotate and move primitives are invariant during editing, every result that we obtain on a level of the subdivision is invariant in any deeper level. For invariant we mean that if we consider a uniformly refined mesh at level l and we apply one of the previous operation on it, we obtain exactly the same result that we would get on another mesh at any further subdivision level.

Proof. The rotate primitive is invariant since the angle it spans is invariant by Lemma 2. The move primitive is invariant since if we refine a mesh we can only add vertices at a higher level, and any angle we add along the line traversed by the Move operation has a width of 6, which is skipped by the Move operation; moreover, the topological length of any chain of edges splitting an edge e is also invariant by definition. □

The invariance lemmas shown in the previous section guarantee that, starting at a splitting edge **p.e** at level l, we can navigate the mesh by moving to adjacent faces of the stencil at level l (through a **p.rotate(3)** operation) and we can follow chains of edges until we reach the other end of an edge at level l (through a **p.move(l)** operation).

3.4. Alignment with surface flows

In many cases, the alignment of edges of a quad mesh are relevant to the modeled shape. For instance, both in the

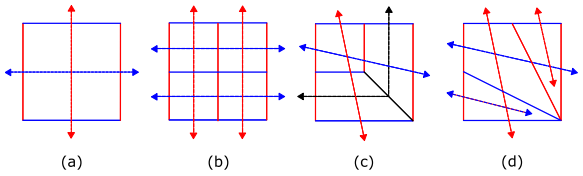


Figure 5: *Preserving flows in a quad mesh: (a) flows traverse a quad element in orthogonal directions; (b) flows are preserved by uniform subdivision; (c) some adaptive patterns break flows and do not allow for consistent labeling of edges; while others maintain flows but introduce non-quad elements (d).*

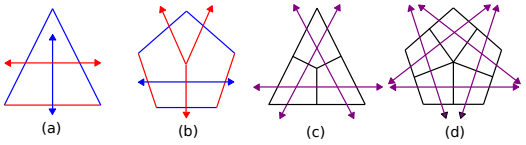


Figure 6: *Triangular and (a) pentagonal (b) transition elements collapse and split/merge the flow in one direction, respectively. Catmull-Clark subdivision of triangles (c) and pentagons (d) does not preserve the flows, though.*

finite element methods and in shape approximation, good anisotropic meshes for a given budget of elements can be obtained if elements are aligned to principal directions of curvature [She02]. If quads are properly aligned with a line field defined on the surface, then edges can be colored with two colors, say red and blue, depicting two orthogonal flows on the surface (see Figure 7). These flows are obviously preserved through quadrisection of quads - see Figures 5(a) and (b) - but they can be deviated by some adaptive patterns - see Figure 5(c).

Triangular and pentagonal faces in a quad-dominant mesh collapse and split/merge flows, respectively, as depicted in Figures 6(a) and (b). Note that the direct application of Catmull-Clark patterns to non-quad meshes would *not* preserve flows. See Figures 6(c) and (d). We have used the pattern depicted in Figure 5(d) instead of the more popular Y pattern of Figure 5(c) for configuration P2b of our adaptive scheme, since it allows us preserving the same flows of its parent quad. It is straightforward to see that also the other patterns of our scheme preserve flows.

Figures 7 shows a base mesh representing a torus, with edges aligned with principal directions of curvature, and an adaptively subdivided mesh obtained from it. Flows are shown on edges and faces by means of textures.

4. Adaptive Catmull-Clark subdivision

In this section we will use our method to edit the LOD of a mesh through operations that modify the mesh locally, while

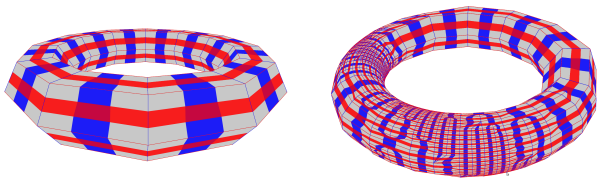


Figure 7: *A torus with edges aligned to principal curvature directions and an adaptive subdivision of it.*

maintaining it compatible with the Catmull-Clark scheme (henceforth called *the standard subdivision*). Compatibility is defined as follows. Given a base mesh Γ_0, then:

1. An adaptive mesh Γ built starting at Γ_0 may contain all and only those vertices that appear in the standard subdivision of Γ_0;
2. If all vertices of a given face f appearing in a the standard subdivision of Γ_0 belong to Γ, then either f, or a subdivision of it also belongs to Γ;
3. If Γ contains a vertex v introduced at level l in the standard subdivision, then the control point $p^l(v)$ will be the same both in Γ and in the standard subdivision. If Γ contains also all vertices in the standard (even) stencil of v at level l, then for any $k \geq l$ the control point $p^k(v)$ will be the same both in Γ and in the standard subdivision.

Our mechanism provides a suitable topological basis to implement an adaptive scheme for the quadrisection pattern. In paticular, a uniform subdivision at a given level l computed incrementally by adding all its vertices through our scheme will be both topologically and geometrically coincident with the standard subdivision at level l, hence also the limit surface will be the same.

Note that the meshes refined selectively as described in the previous section naturally fulfill requirements 1 and 2. Thus in the following we will concentrate on requirement 3.

4.1. Catmull-Clark subdivision

The Catmull-Clark subdivision [CC78] is an approximating scheme for subdivision surfaces that can be applied to any polygonal mesh and converges to a C^2 surface. The subdivision pattern is quadrisection. A new vertex v introduced at level $l+1$ of subdivision is called an *odd* vertex, and the position of its control point $p^l(v)$ at level $l+1$ is computed as a weighted average of control points of vertices surrounding it that belong to level l, according to the stencils and weights depicted in Figure 8.

Vertices already present at level l, called the *even vertices* are relocated at level $l+1$, with a weighted sum of their position and the positions of their edge and face neighbors at level l, according to the stencils depicted in Figure 8. Therefore, for each vertex v introduced at level l, there exist an in-

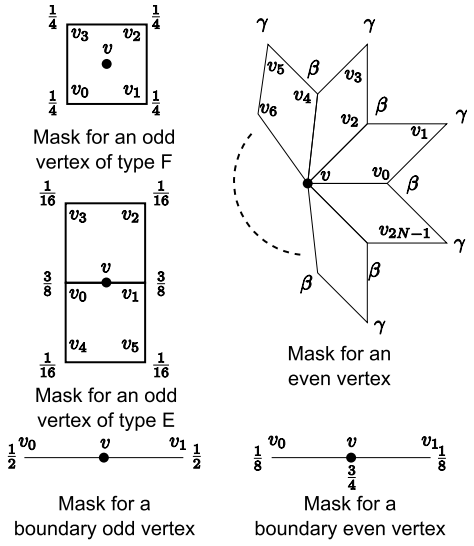


Figure 8: *The stencils of the Catmull-Clark subdivision. Numbers are weights of vertices in the linear combination: n is the valence of the even vertex;* $\beta = \frac{3}{2N}$ *and* $\gamma = \frac{1}{4N}$.

finite sequence of control points $p^l(v), p^{l+1}(v), \ldots, p^\infty(v)$, that define the positions of v at level l and all successive levels, $p^\infty(v)$ being the position of v on the limit surface.

In principle, it is possible to adopt either exact methods [Sta98], or fast approximated methods [LS08] for the direct evaluation of the limit position of any point of a subdivided mesh. However, evaluation requires that: the parametric coordinates of each point, with respect to the face of the base mesh containing it, are known; and the geometry of all control points of the base mesh in the neighborhood of such a face are retrieved. Since we only encode the adaptively refined mesh, retrieving such information may be unpractical, involving traversal of a large area.

We therefore develop an alternative method that makes use just of close neighbors of any given vertex. Any control point $p^k(v)$ for a vertex v introduced at level l, with $0 \leq l < k$ can be computed directly just from the positions p^l of v and of all its even and odd neighbors at level l. In Section 4.2.1, we derive a multi-pass closed form for computing directly control points at an arbitrary level k, which provides a basis for the effective and efficient computation of correct control points in an adaptively subdivided mesh.

4.2. Computing control points

Since we work selectively, it is not trivial to find the right vertices to use for a stencil, and to compute the control points at their proper levels. In this section, first we introduce some tools for the computation of control points, and next we discuss how to use them to maintain geometry up-to-date. In Subsection 4.2.1 we present a multi-pass formula for the Catmull-Clark subdivision, which allows us to compute in closed form the control point of any vertex at any level of subdivision. On this basis, in Subsection 4.2.2 we introduce a mechanism for computing the control point of a given vertex incrementally, as its neighbors are inserted into the mesh. Updates to control points must be made for odd vertices during refinement, and for even vertices both during refinement and during coarsening. We discuss such operations in detail in Subsections 4.2.3, 4.2.4 and 4.2.5, respectively.

4.2.1. Multi-pass subdivision

Let us consider a vertex v inserted at level l. If $l = 0$ then v belongs to the base mesh and its geometry $p^0(v)$ is known, otherwise its control point $p^l(v)$ is computed on the basis of stencils for odd vertices (see Figure 8).

By applying the concept of multi-step subdivision rule [Kob00] to the analysis of the Catmull Clark scheme developed in [Sta98], we derive equations that compute the control point of v and any level k on the basis of its initial position and on the positions of its neighbors at level l.

Lemma 5 [PP10] The control point $p^k(v)$ of an internal vertex v, inserted at level l, for $k > l$ is given by

$$p^k(v) = s^k_{11} v + s^k_{12} \sum_{i=1}^{N} v_{2i} + s^k_{13} \sum_{i=1}^{N} v_{2i-1} \qquad (1)$$

where s^k_{11}, s^k_{12} and s^k_{13} are defined below, and vertices in the summations are the edge and face neighbors of v at level l, respectively.

$$s^k_{11} = \frac{\sqrt{\alpha_n}(11N-35)(\beta_{n,k}-\gamma_{n,k}) + 5\alpha_n(\beta_{n,k}+\gamma_{n,k})}{2 \cdot 8^k \lambda_n} + \frac{N}{N+5}$$

$$s^k_{12} = \frac{-2\sqrt{\alpha_n}(4N-16)(\beta_{n,k}-\gamma_{n,k}) - 4\alpha_n(\beta_{n,k}+\gamma_{n,k})}{2 \cdot 8^k \lambda_n N} + \frac{4}{N(N+5)}$$

$$s^k_{13} = \frac{-\sqrt{\alpha_n}(3N-3)(\beta_{n,k}-\gamma_{n,k}) - \alpha_n(\beta_{n,k}+\gamma_{n,k})}{2 \cdot 8^k \lambda_n N} + \frac{1}{N(N+5)}$$

with $a = \frac{3N-7}{N}$, $b = \frac{\sqrt{\alpha_n}}{N}$, $\beta_{n,k} = (a+b)^k$, $\gamma_{n,k} = (a-b)^k$, $\lambda_n = 5N^3 - 5N^2 - 101N + 245$

Similarly, if v is a boundary vertex we have

$$p^k(v) = sb^k_{11} v + sb^k_{12}(v_0 + v_1). \qquad (2)$$

Note that, by computing the limits for $k \to \infty$ of equations 1 and 2 we obtain the well known limit positions of internal and boundary vertices, respectively:

$$p^\infty(v) = \frac{N}{N+5} v + \frac{4}{N(N+5)} \sum_{i=1}^{N} v_{2i} + \frac{1}{N(N+5)} \sum_{i=1}^{N} v_{2i+1}$$

and

$$p^\infty(v) = \frac{2}{3} v + \frac{1}{6}(v_0 + v_1).$$

4.2.2. Incremental summations

In a standard subdivision, if a vertex v is inserted at a level $l > 0$, two of its edge neighbors already belong to the mesh, while the other two edge neighbors as well as its four face neighbors are inserted at the same time and level as v, and the control points at level l of all such vertices are computed. In an adaptive subdivision, some neighbors of v might be inserted at a later time. Therefore, it is not always possible to apply Equation 1 right after inserting v in the mesh.

We use an approximation of Equation 1 as long as not all neighbors of v are available. In the data structure encoding the mesh, for each vertex v inserted at level l, we store its control point $p^l(v)$ and we reserve two other fields to store the sums $\mathrm{SUM}_e(v)$ and $\mathrm{SUM}_f(v)$ of control points at level l of its edge and face neighbors, respectively. We also store two counters of contributions already stored in $\mathrm{SUM}_e(v)$ and $\mathrm{SUM}_f(v)$. At startup, we fill such fields for all vertices of the base mesh.

For a generic vertex v inserted at level $l > 0$, its control point $p^l(v)$ is computed and stored when creating v as an odd vertex (see Section 4.2.3), while $\mathrm{SUM}_f(v)$ and $\mathrm{SUM}_e(v)$ are computed incrementally, as the control points of neighbors of v become available. Initially, we set both $\mathrm{SUM}_e(v)$ and $\mathrm{SUM}_f(v)$ to $4 * p^l(v)$. Every time a control point $p^l(v_i)$ for a neighbor of v at level l becomes available, its value is accumulated by substituting it to an instance of $p^l(v)$ in either $\mathrm{SUM}_e(v)$ or $\mathrm{SUM}_f(v)$, depending on the position of v_i in the stencil of v.

A vertex v is represented in the mesh with a control point at level k, where k is the smallest level of its incident edges. As long as the correct value of the two sums is not known, v will be represented with an approximation of its position computed by Equation 1 with the values of sums currently stored at $\mathrm{SUM}_e(v)$ and $\mathrm{SUM}_f(v)$. Note that the quality of the approximation progressively increases as new control points are inserted, and the formula becomes exact as soon as all contributions from neighbors of v become available.

4.2.3. Control points for odd vertices during refinement

In the sequel, for the sake of brevity, we will address only internal vertices. Boundary vertices are handled similarly.

Let v be a vertex of type E introduced at level $l+1$ of subdivision by splitting a refinable edge e at level l. In order to compute control point $p^{l+1}(v)$, we need to fetch the vertices in the stencil of v at level l.Referring to Figure 8, vertices v_0 and v_1 of the stencil of v are the endpoints of edge e, so they are found immediately. The other vertices of the stencil are not trivial to fetch since the faces defined by v_0, v_1, v_2, v_3 and v_1, v_0, v_4, v_5 may have been refined further. This can be done with a combination of move and rotate operators. Since these operators are not influenced by editing operations, then the stencil will be fetched correctly also if the mesh has been refined (see Figure 9).

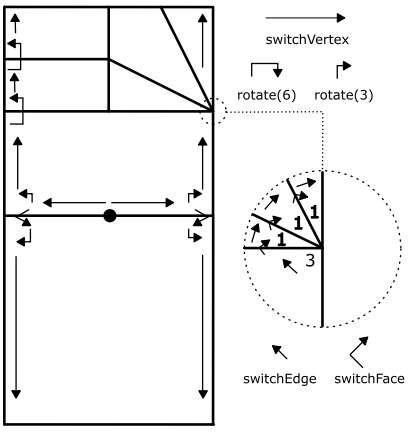


Figure 9: *An example of mesh traversal to fetch the stencil of an odd vertex v. A spanning tree of the neighborhood of v is traversed through move and rotate operations. Move operations are decomposed into sequences of switchVertex, rotate and switchFace; rotate operations are decomposed into sequences of switchEdge and switchFace.*

Any odd vertex v of type F is inserted by operator 2a/b-split together with a vertex v' of type E, and the stencil of v is in fact a subset of the stencil of v', so we do not need to fetch it separately.

If the control point at level l is not available for some vertex v_i in the stencil of v, then recursive refinement must be triggered to insert in the mesh all neighbors of v_i at its insertion level l_i. To this aim, it is necessary to recursively split the edges in the neighborhood of v_i until all faces incident in it are of level greater or equal to l_i. This can be done with a recursive split of all standard edges incident at v_i and having a level lower than l_i, followed by an analysis of the incident faces. If an incident face f is of type 4, then a recursive split of the standard edge with lower level of one of the two triangles that share an extra edge with f is necessary.

This can be done simply by traversing in counterclockwise order the edges incident at v_i. Every time an edge at a level lower than l_i is found, we recursively split it. The algorithm halts when a complete scan of the edges is performed without performing any split. The additional splits required for faces of type 4 can be performed by traversing the faces and splitting a single edge for every face of type 4 found. Note that a regular vertex may have standard incident edges that differ for at most two levels, thus the number of new vertices to be computed during this operation is usually quite small.

Computation of control points may trigger some over-refinement of the mesh, which might be not necessary to fulfill LOD requirements. This is similar to what happens in other adaptive subdivision schemes [Kob00, SHHG01, VZ01]. Since we wish to avoid over-refinement of the result,

edge splits performed during computation of control points, which are unnecessary according to LOD requirements, are marked as temporary and inserted in a queue. A temporary vertex becomes permanent in case one of its incident edges undergoes a standard edge split. At the end of selective refinement, this queue is scanned, and all vertices that are still temporary are removed by performing corresponding edge merge operators.

After v has been inserted at level l, we must find all possible vertices that give contribution to compute summations $\mathrm{SUM}_e(v)$ and $\mathrm{SUM}_f(v)$. This is done again with a navigation algorithm based on topological angles. The stencil that we want to identify can be incomplete, i.e. some vertices may not be present in the mesh, and we have no a priori information on the level of refinement the portions of stencil currently available. We traverse the stencil with a breath-first strategy, starting at v and navigating on subsets of all possible paths that we can use to reach a vertex v_i of the stencil. The algorithm starts by identifying the standard edges at level l_i incident at v. For each edge e_i, connecting v with another vertex v_i, the algorithm navigates the stencil using rotate and move operators until either v_{i-1} and v_{i+1} are found, or it is detected that they are not present in the mesh.

For each candidate v_c found, we must check what kind of contribution is needed:

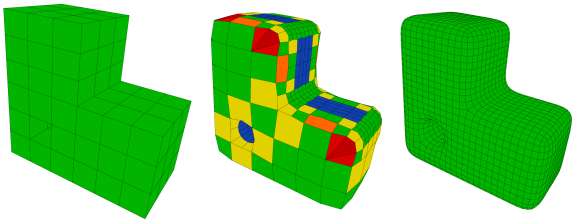
1. If the level of v_c is l, we simply accumulate $p^l(v_c)$ on $\mathrm{SUM}_e(v)$ or $\mathrm{SUM}_f(v)$, depending if v_c being an edge or a face neighbor of v, respectively;
2. If the level of v_c is $< l$ and all the neighbors in the even stencil of v_c are present, we compute the correct control point, and we accumulate it on $\mathrm{SUM}_e(v)$ or $\mathrm{SUM}_f(v)$, exactly as before.

In both cases, we keep track of the fact that v_c has given its contribution to v by keeping a counter for each vertex. This is necessary both to understand when all neighbors have given their contribution and to avoid accumulating a contribution twice, since a vertex can be deleted from the mesh and introduced again at a later time. This operation is performed also in the opposite direction, since every candidate can need the contribution of v to compute its own summations.

As soon as a vertex v at level l receives the contribution from all the vertices in its stencil, we cal it **complete** and its correct control point can be computed at every level $\geq l$. In this case, we immediately provide its contribution to all vertices of level $\geq l$ that are present in its stencil.

4.2.4. Control points for even vertices during refinement

The case of even vertices is simpler. When a new vertex v is inserted to split an edge e, this may affect the control points of the candidate vertices in its neighborhood. For each such vertex v_i, if the minimum level of edges incident at v_i has increased to level k, then the current position of v_i is updated to $p^k(v_i)$. Otherwise no action is required. Note that value $p^k(v_i)$ will be approximated as long as v_i is not complete. This approximation reduces the popping effect during selective refinement and it converges to the correct position as more points in the stencil of v are inserted.



(a) (b) (c)

Figure 10: *A base mesh (a) is adaptively refined to smooth three sharp edges and round the hole (b); the same mesh uniformly refined by using the standard Catmull Clark subdivision scheme (c).*

4.2.5. Control points during coarsening

The removal of a vertex v must undo the updates to the contibution counter and to the contribution accumulators SUM_e and SUM_f for every vertex in the stencil of v. This is done by straightforward adaptations of the algorithms described above. We do not remove contributions from complete vertices, which already have sufficient information to compute their correct control points at all levels.

4.3. Results

We have developed an interactive application that allows us editing the LOD of a subdivided mesh by means of a brush tool. Figure 1(a) shows a rough polygonal model designed in Blender by merging a cylindrical handle to a lathe object. In Figure 1(b) the mesh has been edited by a few strokes of brush in order to refine the joints between the handle and the bottle, as well as to improve its overall shape. A coarsening brush as well as single local refinement operations have been used for adjusting over-refined parts and fine-tuning. Note that the pot-bellied part has been refined anisotropically in order to better approximate shape in the direction of higher curvature. Transitions across different LODs involve pentagonal faces.

Figure 10 shows a simple L-shaped block with a hole. We have refined the hole anisotropically, we have smoothened two convex and one concave edge with two levels of subdivision, and we have refined anisotropically a strip traversing the top faces of the object with one level of subdivision. The different colors represent the different types of faces (see Figure 3): green, blue and dark red faces are quads; yellow and orange faces are pentagons; and light red faces are triangles. As it can be seen by comparing Figures 10 (b) and (c) the adaptively refined mesh contains a much smaller number

of faces than the uniform Catmull Clark subdivision at level two, while it approximates well the shape in the regions of interest. In fact, it can be seen that in the adaptively refined mesh, the most refined parts are identical to the corresponding parts in the uniformly refined mesh.

5. Semi-regular remeshing via adaptive subdivision

In this section we present a remeshing method, which can be either user-assisted, or fully automatic, and it is able to produce a semi-regular adaptively refined mesh representing a given shape. The method starts with a sketched base mesh, which is refined and fitted to the input shape. Selective refinement can be either user-assisted or error-driven.

5.1. Generation and fitting of a base mesh

The target shape is given as a mesh T at high resolution. We start form a coarse quad mesh M, providing a drastically simplified, yet consistent, version of the shape. Mesh M can be generated either manually, or automatically by means of a simplification method, such as the one in [TPC*10]. The overall assumption is that M is roughly *projectable* to T along surface normals, i.e., the projections of points of M roughly subdivide the surface of T into patches with the same connectivity of M. There is no need that M achieves perfect projectability, since this will improve during subsequent refinement. For a shape of genus zero without large protrusions or cavities, such as the head in Figure 2, a base mesh as simple as a meshed cube can be used to the purpose; for more complex shapes with non-zero genus and/or having important protrusions and cavities, such as the horse of Figure 11, or the gargoyle of Figure 12a, a more elaborated base mesh may be necessary. The connectivity of M affects final meshing, since the directions of edges will be preserved during refinement.

Given M, we fit it to the target mesh T through spatial projection: for every vertex v of M, we compute its normal $\mathbf{n}_v$; we shoot a ray in direction of $\mathbf{n}_v$ and another ray in the opposite direction; and we displace v to the closest point hit from a ray. Ray shooting is supported by means of a spatial index that contains all faces of mesh T, which is created once and for all at the beginning of computation. We use a simple regular grid, but more complex and efficient data structures may be adopted for huge datasets [Sam05].

This fitting procedure works well in most cases, supporting interactive editing speed. We have noticed stability problems only if the initial fitting of mesh M to T is very poor, i.e., M lacks entire features of the shape, or the shape contains very thin features. Better and more stable results can be obtained if a parametrization of T is available, which is defined on M. In this case, ray shooting is not necessary, and both surface mapping and smoothing (see also next subsection) can be done via parametrization.

5.2. Editing operations and tangent space smoothing

Editing operations described in Section 3 are applied to selectively refine mesh M. For every refining operation, a vertex of type E is created, which is initially placed at the midpoint of the splitting edge; an additional vertex of type F is also created in pattern P3, which is initially placed at the barycenter of the subdivided face. The normal of each new vertex is estimated, and the vertex is displaced to its projection to the target mesh, as before. Interactive refinement is supported through brush tools, similarly to the case of adaptive subdivision. In this case, either refinement or coarsening operations are applied, up to a prescribed level of subdivision, in the area spanned by the brush. Automatic error-driven refinement is explained in the next section.

To increase the quality of meshing, smoothing is performed in tangent space, by displacing the position of vertices tangentially on the surface of T. The aim of tangential smoothing is to obtain quad faces with a better (i.e., more rectangular) shape. After each local operation, we consider all vertices in the 2-ring of faces affected by the operation. For each such vertex v, we execute a step of Laplacian smoothing - i.e., v is displaced to the barycenter of its neighbors - followed by a re-projection to the target mesh. In case a conformal parametrization of T has been defined on M, computation can be carried out more easily and accurately in parametric space.

5.3. Error-driven remeshing

While a user-assisted approach may be useful in many contexts, some times a fully automatic algorithm is to be preferred. We define the approximation error associated to every face f of the current mesh M as the RMS difference between f and the patch spanned by its projection on T:

$$\frac{1}{Area(f)}\sqrt{\int_f (p-\phi_T(p))^2 dp},$$

where function ϕ_T provides the normal projection of a given point of M to the target mesh T. Computation is discretized on a set of samples selected uniformly on each face f:

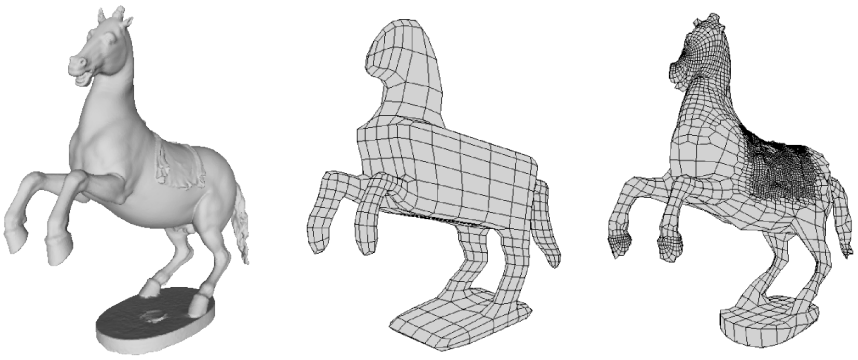
$$\frac{1}{k}\sum_{i=0}^{k} |s_i - \phi_T(s_i)|$$

where k is the number of samples, and s_i is a sample point inside f. To obtain a uniform sampling, the number of samples per face f depends on its area:

$$k = \frac{Area(f)}{Area(M)} * v_t$$

with $Area(f)$ the area of f, $Area(M)$ the total area of mesh M, and v_t the number of vertices of the target mesh T.

To sample triangles, we pick points on the unique plane that contains the triangle. For quads, we pick samples in the



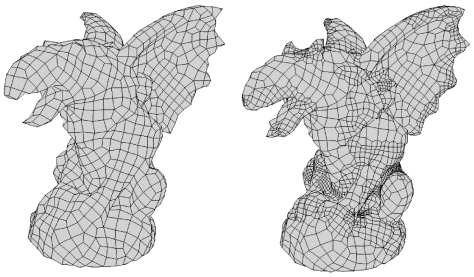
(a) (b) (c)

Figure 11: *Remeshing the Rampart dataset: (a) target model; (b) base mesh manually sketched; (c) remeshing of the dataset that highly refines the head and the saddle.*

bilinear patch that interpolates the four vertices. For pentagons, we first triangulate and then sample each triangle separately.

Faces of M are maintained in a priority queue and refinement is perfomed iteratively. The face with the largest error is extracted from the queue at each iteration, and it is refined with a local operation. For a face f of type 0, 1, 3, or 5, at level l, its edge e at level l yielding the largest error is selected for split. The error associated to an edge is computed similarly by sampling along it. For a face of type 2 at level l, its standard edge at level l is split. For a face f of type 4, its adjacent face of type 2 yielding the largest error is refined, and a vertex of type F is inserted consequently inside f.

After each iteration, error is computed for the new faces resulting from refinement, and they are inserted in the priority queue accordingly. Iteration may be carried out until either a certain budget of faces has been reached, or when error gets below a given threshold, depending on user's needs. A result of this automatic remeshing procedure is shown in Figure 12. The original target mesh has been simplified with [TPC*10] to obtain the base mesh M, and then it has been remeshed with our algorithm.



(a) (b)

Figure 12: *Fully automatic remeshing of the Gargoyle dataset. (a) The base mesh generated using [TPC*10]. (b) Remeshed model.*

6. Implementation

We have implemented our scheme under the OpenMesh library [Ope] by using its standard data structures. The data structure to represent a mesh has been extended just with attributes to keep the level and type of each edge, and the level of each vertex. Assuming a maximum of 16 levels of subdivision, which is more than sufficient for practical purposes, such attributes can be maintained with one byte per edge, and one byte per vertex. For adaptive subdivision only, also summations and counters to compute control points are maintained, requiring additional six floats per vertex.

In order to analyze space occupancy, we note that our scheme implicitly encodes the subdivision hierarchy corresponding to a quad-tree representation, by representing just its leaves. In the case of a complete tree, which encodes a uniform subdivision, we encode about 67% of the total number of quad-tree nodes. Our scheme uses about 33% less space than the multi-resolution half-edge data structure presented in [KCB09], and about 3% more space than a full quad-tree, encoded as in [KCB09].

Our prototype running on a single core of a T9300 Intel Core Duo at 2.5 Ghz can insert/remove about 40K vertices per second. The framework can thus easily support interactive LOD editing even with large meshes.

7. Conclusion

The adaptive subdivision scheme we have presented has several advantages: it is fully dynamic, i.e., the mesh can be freely refined and coarsened through local operators; it does not require any hierarchical data structure; and it preserves the orientation of mesh elements according to flows defined on the base mesh.

A base mesh containing non-quad elements (e.g., in the proximity of singularities) is better suited than a purely quad mesh in order to maintain alignment of elements. In order to correctly manage such meshes, our scheme can be easily extended to process a quad/triangle base mesh, by integrating in the same framework local operators to adaptively refine and coarsen triangles [PP09b].

Implementation of selective refinement on a data parallel architecture is probably possible and could be investigated.

References

[Bri93] BRISSON E.: Representing geometric structures in d dimensions: Topology and order. *Discrete and Computational Geometry 9* (1993), 387–426. 5

[BSW83] BANK R., SHERMAN A., WEISER A.: Some refinement algorithms and data structures for regular local mesh refinement. In *Scientific Computing*, Stepleman R., (Ed.). IMACS/North Holland, 1983, pp. 3–17. 2

[BZK09] BOMMES D., ZIMMER H., KOBBELT L.: Mixed-integer quadrangulation. *ACM Trans. Graph. 28*, 3 (2009), 1–10. 1

[CC78] CATMULL E., CLARK J.: Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer Aided Design 10* (1978), 350–355. 6

[DBG*06] DONG S., BREMER P.-T., GARLAND M., PASCUCCI V., HART J.: Spectral surface quadrangulation. *ACM Trans. Graph. 25*, 3 (2006), 1057–1066. 1

[DWS*97] DUCHAINEAU M., WOLINSKY M., SIGETI D., MILLER M., ALDRICH C., MINEEV-WEINSTEIN M.: ROAMing terrain: Real-time optimally adapting meshes. In *Proceedings IEEE Visualization '97* (Oct. 1997), IEEE, pp. 81–88. 2

[KCB09] KRAEMER P., CAZIER D., BECHMANN D.: Extension of half-edges for the representation of multiresolution subdivision surfaces. *Vis. Comput. 25*, 2 (2009), 149–163. 2, 11

[KL03] KIM J., LEE S.: Transitive mesh space of a progressive mesh. *IEEE Transactions on Visualization and Computer Graphics 9*, 4 (2003), 463–480. 2

[KNP07] KÄLBERER F., NIESER M., POLTHIER K.: Quadcover – surface parameterization using branched coverings. 375–384. 1

[Kob00] KOBBELT L.: $\sqrt{3}$ subdivision. In *Proceedings ACM SIGGRAPH 2000* (2000), pp. 103–112. 2, 7, 8

[LRC*02] LÜBKE D., REDDY M., COHEN J., VARSHNEY A., WATSON B., HÜBNER R.: *Level Of Detail for 3D Graphics*. Morgan Kaufmann, 2002. 2

[LS08] LOOP C., SCHAEFER S.: Approximating catmull-clark subdivision surfaces with bicubic patches. *ACM Trans. Graph. 27*, 1 (2008), 1–11. 7

[MJ98] MÜLLER H., JAESCHKE R.: Adaptive subdivision curves and surfaces. In *Proceedings of Computer Graphics International '98* (1998), pp. 48–58. 2

[MNP08] MYLES A., NI T., PETERS J.: Fast parallel construction of smooth surfaces from meshes with tri/quad/pent facets. *Computer Graphics Forum 27*, 5 (July 2008), 1365–1372. 1

[MS01] MAILLOT J., STAM J.: A unified subdivision scheme for polygonal modeling. *Computer Graphics Forum 20*, 3 (2001), 471–479. 1

[Ope] Openmesh. http://www.openmesh.org/. 11

[PP09a] PANOZZO D., PUPPO E.: Interpolatory adaptive subdivision for mesh lod editing. In *Proceedings GRAPP 2009 - International Conference on Computer Graphics Theory and Applications* (Lisboa, Portugal, February 5–8 2009), pp. 70–75. 2

[PP09b] PUPPO E., PANOZZO D.: RGB subdivision. *IEEE Transactions on Visualization and Computer Graphics 15*, 2 (2009), 295–310. 2, 4, 12

[PP10] PANOZZO D., PUPPO E.: Adaptive lod editing of quad meshes. In *Afrigraph* (2010), pp. 7–16. 7

[PS07] PAKDEL H., SAMAVATI F.: Incremental subdivision for triangle meshes. *International Journal of Computational Science and Engineering 3*, 1 (2007), 80–92. 2

[Pup98] PUPPO E.: Variable resolution triangulations. *Computational Geometry 11*, 3-4 (1998), 219–238. 2

[RLL*06] RAY N., LI W.-C., LÉVY B., ALLIEZ P., SHEFFER A.: Periodic global parameterization. *ACM Trans. Graph.* (2006). 1

[Sam05] SAMET H.: *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. 10

[She02] SHEWCHUK J. R.: What is a good linear finite element? - interpolation, conditioning, anisotropy, and quality measures. In *Proc. of the 11th International Meshing Roundtable* (2002). Unpublished extended version available at http://www.cs.berkeley.edu/ jrs/papers/elemj.pdf. 6

[SHHG01] SEEGER S., HORMANN K., HÄUSLER G., GREINER G.: A sub-atomic subdivision approach. In *Proceedings of Vision, Modeling and Visualization 2001* (Berlin, 2001), Girod B., Niemann H., Seidel H.-P., (Eds.), Akademische Verlag, pp. 77–85. 8

[SL03] STAM J., LOOP C.: Quad/triangle subdivision. *Computer Graphics Forum 22*, 1 (2003), 79–85. 1

[Sta98] STAM J.: Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *Proceedings SIGGRAPH '98* (1998), pp. 395–404. 7

[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. 10, 11

[VG00] VELHO L., GOMES J.: Variable resolution 4-k meshes: Concepts and applications. *Computer Graphics Forum 19*, 4 (2000), 195–214. 2

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

[XK99] XU Z., KONDO K.: Adaptive refinements in subdivision surfaces. In *Eurographics '99, Short papers and demos* (1999), pp. 239–242. 2

[ZSS97] ZORIN D., SCHRÖDER P., SWELDENS W.: Interactive multiresolution mesh editing. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH 97), ACM Press* (1997). 259-268. 2