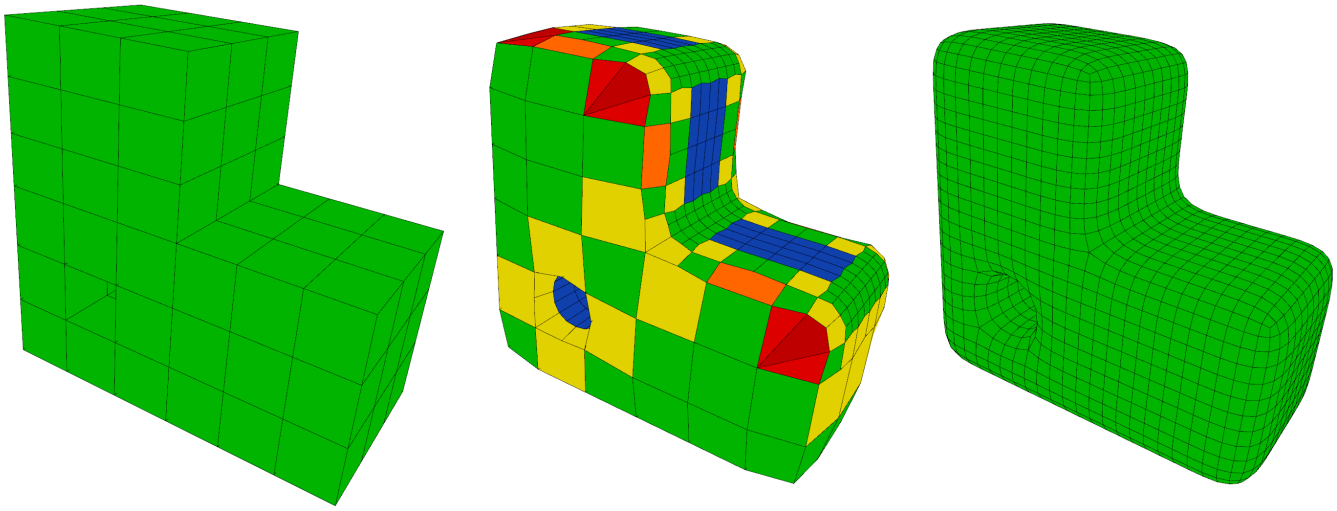
# Adaptive LOD Editing of Quad Meshes

Daniele Panozzo*
DISI, University of Genoa, Genoa, Italy

Enrico Puppo†
DISI, University of Genoa, Genoa, Italy



(a) (b) (c)

**Figure 1:** *A base mesh composed of three cubes (a); the mesh is adaptively refined to smooth three straight edges (b); the same mesh uniformly refined at level 2 by using the standard Catmull Clark subdivision scheme (c).*

## Abstract

We present a method for editing the LOD of quad meshes, which supports both adaptive refinement and adaptive coarsening. Starting at a base mesh, we generate a quad-dominant mesh which is consistent with the Catmull-Clark subdivision. Consistency is both topological and geometrical: an adaptively subdivided mesh coincides with the uniformly subdivided mesh wherever the level of subdivision is uniform, and the limit surface is the same. 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 and work on a plain mesh without the need of cumbersome hierarchical data structures.

**CR Categories:** 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

**Keywords:** level of detail, adaptive subdivision surfaces, quad-dominant meshes

*e-mail: panozzo@disi.unige.it
†e-mail: puppo@disi.unige.it

## 1 Introduction

Although advances in graphics hardware and API's provide support for real time rendering of smooth surfaces [Gee 2008; Castaño 2008; Eisenacher et al. 2009; Patney and Owens 2008], polygonal modeling still remains a primary approach for applications that require computational intensive tasks other then rendering, such as video games and finite element methods. A polygonal mesh may be also used for intermediate computations, even if a smooth surface is to be rendered. It is therefore useful to support modeling and manipulation of polygonal meshes that, on one hand, have a controlled budget of polygons and, on the other hand, are close to ideal smooth surfaces.

In the context of polygonal modeling, quad-based surface representations 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, such as those corresponding to principal curvatures [Alliez et al. 2003; Bommes et al. 2009; Marinov and Kobbelt 2004].

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. Mesh subdivision is often used to this purpose [Maillot and Stam 2001]. Non-trivial shapes generally require the base mesh to be subdivided adaptively 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, and adaptive subdivision 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,

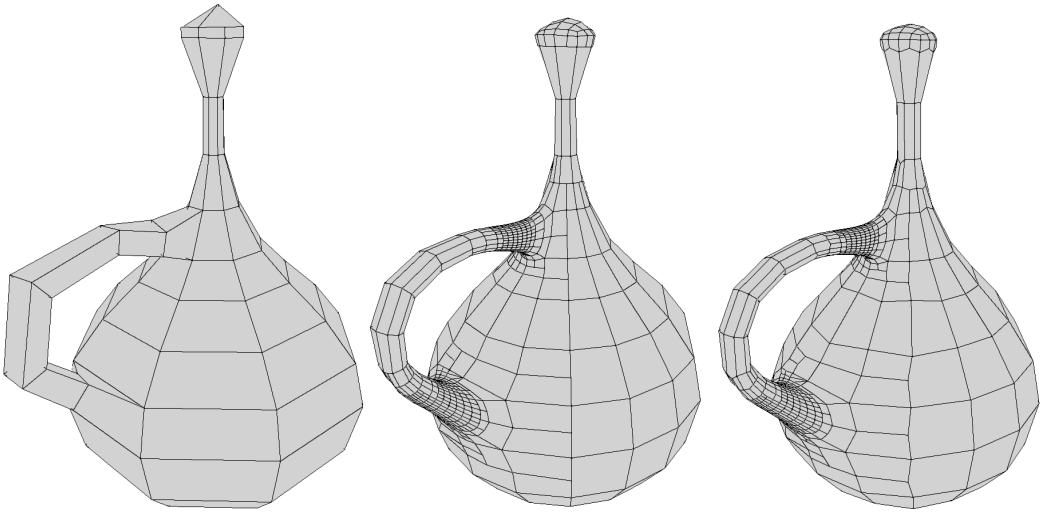



**Figure 2:** *A base mesh represents either the coarse version of a shape or the control mesh of a smooth shape (a); the mesh is adaptively refined to obtain a better polygonal model (b); vertices of the refined mesh are moved to their limit position to approximate the smooth subdivision surface (c).*

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 [Myles et al. 2008; Stam and Loop 2003]. 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 edit the LOD of a quad mesh through local operators for both mesh refinement and mesh coarsening (see Figure 2). Our method generates quad-dominant meshes containing a small amount of triangular and pentagonal transition elements, which are consistent with a Catmull-Clark subdivision of the base mesh. Consistency is both topological and geometrical, i.e., an adaptively subdivided mesh coincides with a uniformly subdivided mesh wherever the level of subdivision is uniform, and the limit surface is the same. Our approach naturally preserves the surface flows and lineal features defined on the base mesh. LOD editing is controlled with 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 present an interactive tool built on top of our method that supports editing a mesh through LOD brushes.

The rest of the paper is organized as follows. In Section 2, we briefly discuss related work. In Section 3, we introduce the necessary background. In Section 4, we describe our adaptive subdivision scheme. In Section 5 we discuss our implementation and we present experimental results. In Section 6 we make some concluding remarks.

## 2 Related work

Classical subdivision patterns, such as the face quadrisection, are meant to be applied to all faces of a mesh together, and generate non-conforming meshes when applied selectively. *Red-green triangulations* [Bank et al. 1983] 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 [Pakdel and Samavati 2007; Zorin et al. 1997] 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 [Panozzo and Puppo 2009; Puppo and Panozzo 2009] 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 [Kobbelt 2000] and the 4-8 subdivision [Velho and Zorin 2001] for triangle meshes are subdivision schemes based on different patterns that can be implemented through local conforming operators, making them naturally adaptive. In [Kraemer et al. 2009], 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 [Müller and Jaeschke 1998] and [Xu and Kondo 1999], to extend the classical Catmull-Clark and Doo-Sabin subdivision schemes. These methods are essentially based on introducing extra patterns to eliminate T-vertices, and they generate quad-dominant meshes containing triangular elements at transitions across levels of subdivision. The adaptive scheme proposed in [Kobbelt 1996] generates purely quad meshes by using transition elements subdivided with a Y pattern. The requirement of maintaining a purely quad mesh is paid in terms of a certain over-refinement, because elements are refined by introducing two boundary vertices at a time. Moreover, Y patterns may destroy the natural alignment of elements (see Figure 7(c)). All these methods are based on a two-step approach, i.e., a non-conforming mesh is built first, which is fixed next with special subdivision patterns. Such an approach makes interactive LOD editing unwieldy, especially if also coarsening tasks should be supported.

## 3 Background

We deal with manifold polygonal meshes containing triangles, quads and pentagons, adopting standard terminology. A *quad mesh*

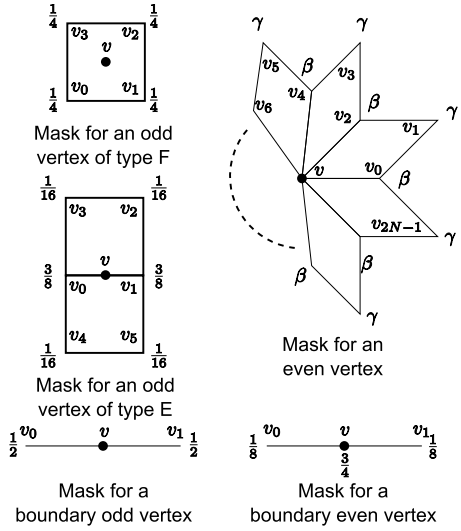



**Figure 3:** *The stencils used in the Catmull-Clark subdivision scheme. Numbers are weights assigned to vertices in the linear combination: $n$ is the valence of the even vertex; $\beta = \frac{3}{2N}$ and $\gamma = \frac{1}{4N}$.*

is a polygonal mesh where all faces are quads, while 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.

The Catmull-Clark subdivision [Catmull and Clark 1978] is an approximating scheme for subdivision surfaces that can be applied to any polygonal mesh and converges to a $C^2$ surface if applied to a regular quad mesh. The subdivision pattern for a quad face is *quadrisection*, i.e., a 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. 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 wighted average of control points of vertices surrounding it that belong to level $l$, according to the stencils and weights depicted in Figure 3. An odd vertex that splits an edge is said to be of type $E$, while an odd vertex inserted in the middle of a quad is said to be of type $F$.

Vertices already present at level $l$, called the *even vertices* are relocated at level $l+1$, through a weighted sum of their position and the positions of their edge and face neighbors at level $l$, according to the stencil and weights depicted in Figure 3. Therefore, for each vertex $v$ introduced at level $l$, there exist an infinite 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, where $p^\infty(v)$ is the position of $v$ on the limit surface.

Stam [Stam 1998b] derived a method for evaluating the position on the limit surface of any point of a subdivided mesh. More recently, Loop and Schaefer [Loop and Schaefer 2008] have proposed a simpler and more efficient method that approximates Catmull-Clark subdivision surfaces with bicubic patches. However, when subdivision is used for polygonal modeling, one is often not interested in knowing the limit positions of vertices, but rather to locate them correctly at the desired level of subdivision. To this purpose, any control point $p^k(v)$ for a vertex $v$ introduced at level $l$, with $0 \le l < k$ 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 Adaptive quad subdivision

Our aim is to provide a mechanism to edit the LOD of a mesh through local operations that modify the mesh locally, while maintaining it compatible with the Catmull-Clark scheme (henceforth called *the standard subdivision*). Compatibility is defined as follows. Given a base mesh $\Gamma_0$, then:

1. An adaptive mesh $\Gamma$ built starting at $\Gamma_0$ may contain all and only those vertices that appear in the standard subdivision of $\Gamma_0$;
2. If all vertices of a given face $f$ appearing in a the standard subdivision of $\Gamma_0$ belong to $\Gamma$, then either $f$, or a subdivision of it also belongs to $\Gamma$;
3. If $\Gamma$ 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 $\Gamma$ and in the standard subdivision. If $\Gamma$ contains also all vertices in the standard (even) stencil of $v$ at level $l$, then for any $k \ge l$ the control point $p^k(v)$ will be the same both in $\Gamma$ and in the standard subdivision.

Note that, if a uniform subdivision of $\Gamma_0$ at a given level $l$ is computed incrementally by adding all its vertices through our scheme, then it will be both topologically and geometrically coincident with the standard subdivision at level $l$, hence also the limit surface will be the same.

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, fulfilling requirements 1 and 2 above. This will be the subject of Section 4.1. Moreover, control points of vertices involved in the operation are computed and updated, according to requirement 3 above, by fetching vertices in their stencils. This will be the subject of Section 4.2.

## 4.1 Topological operators

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 4 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 5.1 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); square triangles (type 2); and

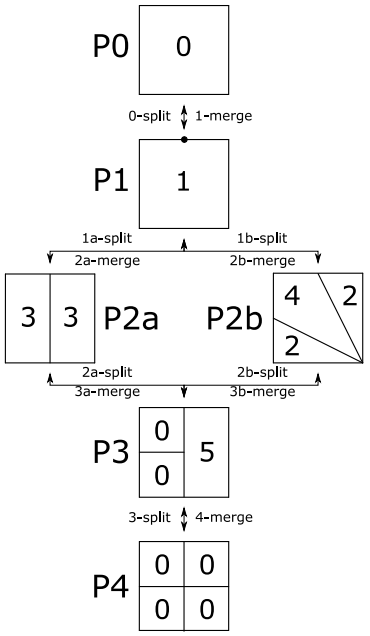



**Figure 4:** *The diagram of patterns for adaptive subdivision of a quad. Types of faces and patterns are denoted by labels placed inside and beside them, respectively. Transitions between adjacent patterns are labeled with the corresponding refinement and coarsening operators.*

pentagons with three collinear vertices, having the shape of either a square or a rectangle (type 1 and 5, respectively).

Consider a base mesh $\Gamma_0$. 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 $\Gamma$ 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.

#### 4.1.1 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 $\Gamma_0$ to generate a conforming mesh $\Gamma$ composed of faces of the six types illustrated in Figure 4.

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 4 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 pre-compute 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 $\Gamma_0$ 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. Finally note that the meshes we generate obey requirements 1 and 2 defined at the beginning of Section 4. Thus our mechanism provides a suitable topological basis to implement any adaptive scheme for the quadrisection pattern.

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 $< l$.

#### 4.1.2 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 4. 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 meshes we generate through merge operators obey requirements 1 and 2 defined at the beginning of Section 4.

The set of refinement operators is also closed with respect to the meshes obtained. If we start at a mesh $\Gamma$ obtained from $\Gamma_0$ through refinement, we can apply merge operators in any legal order to go back to $\Gamma_0$; moreover, any intermediate mesh could be refined through split operators. So we can mix split and merge operators in any order while preserving consistency.

## 4.2 Computing control points

Control points of all vertices are computed using the stencils in Figure 3. 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, we first introduce some tools for the computation of control points and the navigation of the mesh, and next we discuss how to use them to maintain geometry up-to-date. In Subsection 4.2.1 we derive 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. In Subsection 4.2.3 we introduce operators to navigate an adaptively subdivided mesh that are based on the concepts of *topological angle* and *length*, which allow us to retrieve stencils efficiently. 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 4.2.4, 4.2.5 and 4.2.6, respectively.

### 4.2.1 Multi-pass formulae for the Catmull-Clark 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 3).

By applying the concept of multi-step subdivision rule [Kobbelt 2000] to the analysis of the Catmull Clark scheme developed in [Stam 1998b], 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 1** *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 in the proof, and vertices in the summations are the edge and face neighbors of $v$ at level $l$, respectively.*

*Similarly, if $v$ is a boundary vertex we have*

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

**Proof.** Following Stam [Stam 1998b] the portion of subdivision matrix involving an internal vertex $v$ and its $2N$ neighbors $v_1, \ldots, v_{2N}$ has the following structure:

$$S = \begin{pmatrix}
a_n & b_n & c_n & b_n & c_n & b_n & \ldots & b_n & c_n & b_n & c_n \\
d & d & e & e & 0 & 0 & \ldots & 0 & 0 & e & e \\
f & f & f & f & 0 & 0 & \ldots & 0 & 0 & 0 & 0 \\
d & e & e & d & e & e & \ldots & 0 & 0 & 0 & 0 \\
f & 0 & 0 & f & f & f & \ldots & 0 & 0 & 0 & 0 \\
 & & \vdots & & & \ddots & & \vdots & & & \\
d & e & 0 & 0 & 0 & 0 & \ldots & e & e & d & e \\
f & f & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & f & f
\end{pmatrix}$$

where $a_n = 1 - \frac{7}{4N}$, $b_n = \frac{3}{2N^2}$, $c_n = \frac{1}{4N^2}$, $d = \frac{3}{8}$, $e = \frac{1}{16}$ and $f = \frac{1}{4}$.

If $\mathbf{v} = (v, v_1, \ldots, v_{2N})^T$, then the product $S\mathbf{v}$ gives the control point of $v$ and all its neighbors at the next level of subdivision. Thus, the control point of $v$ after $k$ levels of subdivision can be computed by using matrix $S^k$ and multiplying the first row of such a matrix by $\mathbf{v}$. We can obtain $S^k$ from the decomposition $S = U\Lambda U^{-1}$, where $U$ is the matrix of eigenvectors and $\Lambda$ is the diagonal matrix of eigenvalues. Then we will have $S^k = U\Lambda^k U^{-1}$. Actually, we are interested just in the first row of such a matrix.Moreover, by symmetry, we know that all coefficients $s^k_{1(2j)}$ for $j = 1, N$ in the first row of $S^k$ must be equal, and the same applies to coefficients $s^k_{1(2j-1)}$ for $j = 1, N$. Thus, it is sufficient to compute just coefficients $s^k_{11}$, $s^k_{12}$ and $s^k_{13}$. From Stam [Stam 1998b], we have that the first row of matrix U is:

$$U_{1,*} = (1, 16\mu_2^2 - 12\mu_2 + 1, 16\mu_3^2 - 12\mu_3 + 1, 0, \ldots, 0)$$

with $\alpha_n = 5N^2 - 30N + 49$, $\mu_2, \mu_3 = \frac{1}{8N}(-7 + 3N \mp \sqrt{\alpha_n})$, and we also we have that

$$\Lambda = \mathrm{diag}(1, \mu_2, \mu_3, \ldots)$$

For our purposes, we can disregard all the other eigenvalues. If we explicit the first three elements of the first row of $S^k$ we obtain

$$s^k_{1j} = u_{11} u^{-1}_{1j} + u_{12} \mu^k_2 u^{-1}_{2j} + u_{13} \mu^k_3 u^{-1}_{3j}. \qquad (3)$$

We now need to compute $u^{-1}_{ij}$ for $i, j = 1, 2, 3$. From $UU^{-1} = I$ we obtain:

$$u_{11} u^{-1}_{1j} + u_{12} u^{-1}_{2j} + u_{13} u^{-1}_{3j} = z \qquad (4)$$

with $z = 1$ if $j = 1$, $z = 0$ elsewhere. Since $S = U\Lambda U^{-1}$ we have:

$$u_{11} u^{-1}_{1j} + u_{12} \mu_2 u^{-1}_{2j} + u_{13} \mu_3 u^{-1}_{3j} = S_{1j} \qquad (5)$$

Stam decomposes matrix $S$ in a block-diagonal matrix $\hat{S}$ by defining a matrix $T$ such that $\hat{S} = TST^{-1}$. Using the same notation we have that $T^{-1}\hat{S}T = S$, thus $T^{-1}\hat{S}^{-1}T = S^{-1}$. Matrix $\hat{S}^{-1}$ can be computed easily in symbolic form by inverting each diagonal block of $\hat{S}$ independently. We use such a decomposition to compute matrix $S^{-1}$. We omit the detailed computation here for the sake of brevity. By inverting $S = U\Lambda U^{-1}$ we obtain $S^{-1} = U\Lambda^{-1}U^{-1}$, which leads to another set of equations of the form:

$$u_{11} u^{-1}_{1j} + u_{12} \mu_2^{-1} u^{-1}_{2j} + u_{13} \mu_3^{-1} u^{-1}_{3j} = S^{-1}_{1j} \qquad (6)$$

By resolving the linear system formed by Equations 4, 5 and 6 with a fixed $j = 1, 2, 3$ we obtain an explicit formula for $u^{-1}_{1j}$, $u^{-1}_{2j}$ and $u^{-1}_{3j}$. The algebraic manipulations required for the computation of the solution of the three systems are rather ugly, we report only the solutions already substituted in Equation 3:

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

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

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

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

Hence, the control point of vertex $v$ after $k$ levels of subdivisions (following the level at which $v$ was inserted as an odd vertex) is $p^k(v) = S_{1,*}^k \mathbf{v}$, i.e.:

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

For a boundary vertex, the corresponding portion of subdivision matrix is:

$$S = \frac{1}{8}\begin{pmatrix} 6 & 1 & 1 \\ 4 & 4 & 0 \\ 4 & 0 & 4 \end{pmatrix}.$$

In this case, we can compute the decomposition explicitly:

$$U = \begin{pmatrix} 1 & 1 & 0 \\ 1 & -2 & 1 \\ 1 & -2 & -1 \end{pmatrix}, \qquad U^{-1} = \begin{pmatrix} \frac{2}{3} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{3} & -\frac{1}{6} & -\frac{1}{6} \\ 0 & \frac{1}{2} & -\frac{1}{2} \end{pmatrix}$$

and $\Lambda = \mathrm{diag}(1, \frac{1}{4}, \frac{1}{2})$. Therefore we have:

$$sb_{11}^k = \frac{1}{3}\left(2 + \left(\frac{1}{4}\right)^k\right) \qquad sb_{12}^k = sb_{13}^k = \frac{1}{6}\left(1 - \left(\frac{1}{4}\right)^k\right).$$

Hence, the control point of a border vertex $v$ after $k$ levels of subdivisions (following the level at which $v$ was inserted as an odd vertex) is

$$p^k(v) = sb_{11}^k v + sb_{12}^k (v_0 + v_1).$$

□

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

We build the stencils of even vertices incrementally, while neighbors of a vertex are inserted in the adaptively refined mesh, and to speed-up computation of control points as the neighborhood of a vertex changes.

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 thus use an approximation of such equations 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 $\mathrm{SUM}_{edge}(v)$ and $\mathrm{SUM}_{face}(v)$ of control points at level $l$ of its edge and face neighbors, respectively. We also store two counters of contributions already stored in $\mathrm{SUM}_{edge}(v)$ and $\mathrm{SUM}_{face}(v)$. At startup, we fill such fields for all vertices of the base mesh.

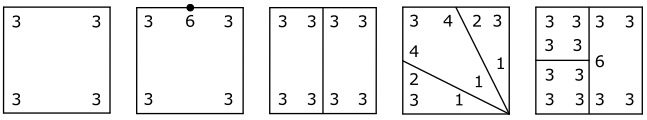



**Figure 5:** *Topological angles: a width is assigned to each vertex in each face.*

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 4.2.4), while $\mathrm{SUM}_{face}(v)$ and $\mathrm{SUM}_{edge}(v)$ are computed incrementally, as the control points of neighbors of $v$ become available. Initially, we set both $\mathrm{SUM}_{edge}(v)$ and $\mathrm{SUM}_{face}(v)$ 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 $\mathrm{SUM}_{edge}(v)$ or $\mathrm{SUM}_{face}(v)$, 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 $\mathrm{SUM}_{edge}(v)$ and $\mathrm{SUM}_{face}(v)$. 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 Topological Angles and Lengths

Fetching stencils in an adaptively refined mesh requires navigating the mesh. In order to support navigation efficiently, we assign a *topological width* to every pair Face-Vertex $(F, V)$ in a mesh. The values are assigned to faces of the various types as indicated in Figure 5. 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 now give some invariants on angles that will be useful for mesh navigation.

**Lemma 2** *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 5 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 $e_0 e_1$ is 6. □

**Lemma 3** *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 4 with angles depicted in Figure 5: 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 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 4** *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 [Brisson 1993], 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.

**Lemma 5** *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 3. 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. □

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 triangles 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).

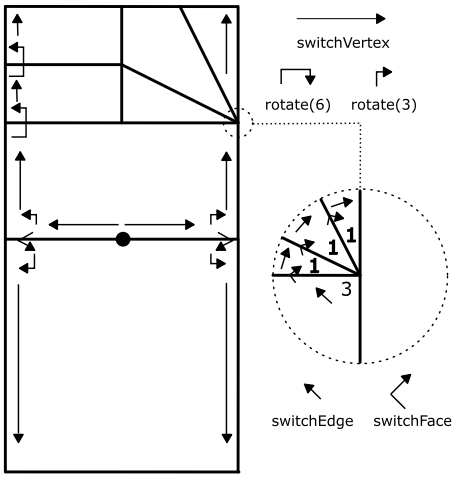



**Figure 6:** *An example of mesh traversal to fetch the stencil of an odd vertex v. A spanning tree of the neighborhood of v is traversed through move and rotate operations. Move operations are decomposed into sequences of switch.Vertex, rotate and switch.Face; rotate operations for a given angle are decomposed into sequences of switch.Edge and switch.Face (see magnified region).*

#### 4.2.4 Control points for odd vertices during refinement

In the sequel, for the sake of brevity, we will address only internal vertices. Boundary vertices are treated in the same way by using the suitable stencils and corresponding multi-pass formulae.

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 eight vertices in the stencil of $v$ at level $l$, and their control points $p^l$. Referring to Figure 3, 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. Fetching can be performed through a combination of move and rotate operators. Since these operators are not influenced by editing operation on the mesh, we are guaranteed that the stencil will be fetched correctly also if the mesh has been refined (see Figure 6).

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 counter-clockwise 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 [Kobbelt 2000; Seeger et al. 2001; Velho and Zorin 2001]. 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 the possible vertices that give contribution to compute summations $\text{SUM}_{edge}(v)$ and $\text{SUM}_{face}(v)$. 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 $\text{SUM}_{edge}(v)$ or $\text{SUM}_{face}(v)$, 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 $\text{SUM}_{edge}(v)$ or $\text{SUM}_{face}(v)$, 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, its correct control point can be computed at every level $\geq l$. We call such a vertex a **complete** vertex. In this case we immediately provide its contribution to all vertices of level $\geq l$ that are present in its stencil.

#### 4.2.5 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 greatly reduces the popping effect during selective refinement and it converges to the correct position as more points in the stencil of $v$ are inserted into the mesh.

#### 4.2.6 Control points during coarsening

The removal of a vertex $v$ must undo the updates to the contibution counter and to the contribution accumulators $\text{SUM}_{edge}$ and $\text{SUM}_{face}$ 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.

## 5 Implementation and Results

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, as well as summations and counters to compute its control points. 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 and six floats per vertex.

In order to analyze space occupancy, we note that our scheme implicitly encodes the subdivision hierarchy corresponding to a quadtree representation, by representing just the 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. In comparison with the multi-resolution half-edge data structure presented in [Kraemer et al. 2009], our scheme uses about 33% less space. In fact, a space occupancy of about 3% more than that of a full quadtree is reported in [Kraemer et al. 2009].

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.

We have developed an application, which supports editing the level of detail of a mesh by either refining or coarsening it through adaptive subdivision. The user is allowed either to perform single refinement/coarsening operations, or to use two brush tools, one for refining and the other for coarsening the area of mesh spanned by the brush, by setting the desired level of subdivision. A prototype of our application is released to the public domain at our web site `http://ggg.disi.unige.it`.

Figure 2(a) shows a rough polygonal model designed in Blender by merging a cylindrical handle to a lathe object. In Figure 2(b) the mesh has been refined by a few strokes of brush in our tool in order to refine the joints between the handle and the bottle as well as to improve its overall shape. The coarsening brush as well as single local refinement operations have been used during editing 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. In Figure 2(c) all vertices have been moved to their limit positions.

Figure 1 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 4): 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 1 (b)

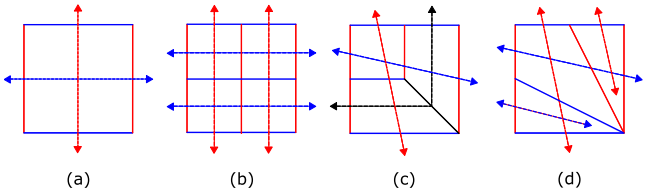



**Figure 7:** *Preserving flows in a quad mesh: (a) two flows of integral lines traverse a quad element in orthogonal directions; (b) flow is preserved by uniform subdivision; (c) some adaptive patterns destroy the flow and do not allow for consistent labeling of edges; while others maintain the flow but introduce non-quad elements (d). Arrows denote flows traversing elements.*

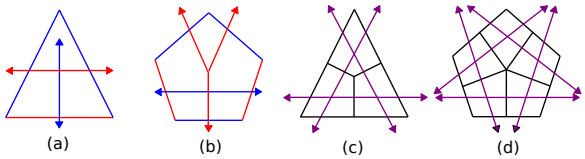



**Figure 8:** *Triangular and (a) pentagonal (b) transition elements collapse and split/merge the flow in one direction, respectively. Catmull-Clark subdivision of triangles (c) and pentagons (d) does not preserve the flows, though.*

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. In Figures 1 (b) and (c) all vertices are taken to their limit position.

### 5.1 Alignment with surface flows

In many cases, the alignment of edges of a quad mesh are relevant to the modeled shape. One example is when edges are aligned with shape features, another is when they are aligned with a line field defined on the surface [Bommes et al. 2009]. For instance, it is well known in the finite element methods that good anisotropic meshes for a given budget of elements can be obtained if elements are aligned to principal directions of curvature [Shewchuk 2002], and this is also true for shape approximation.

If the edges of quads are properly aligned with curvature, or with any other line field defined on the surface, the edges of each quad can be colored with two colors, say red and blue, depicting two orthogonal flows on the surface (see Figure 9). These flows are obviously preserved through quadrisection of quads (see Figures 7(a) and (b)), but they can be deviated by some adaptive patterns (see Figure 7(c)).

As already mentioned, triangular and pentagonal faces in a quad-dominant mesh can be used to collapse and split/merge flows, respectively, as depicted in Figures 8(a) and (b). Note that the direct Catmull-Clark subdivision of non-quad meshes would not preserve flows. See Figures 8(c) and (d). We have used the pattern depicted in Figure 7(d) instead of the Y pattern of Figure 7(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, therefore the flows defined by edges of elements in an adaptively refined mesh will be consistent with those of the base mesh.

Figures 9 shows a base mesh representing a torus, with edges

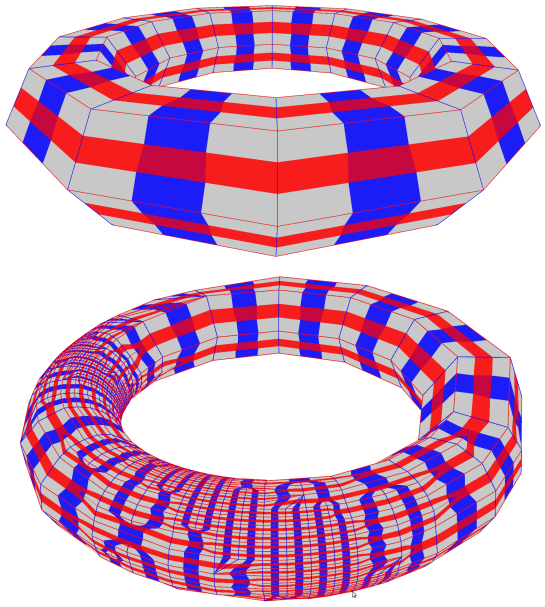


**Figure 9:** *A torus with edges aligned to principal curvature directions and an adaptive subdivision of it.*

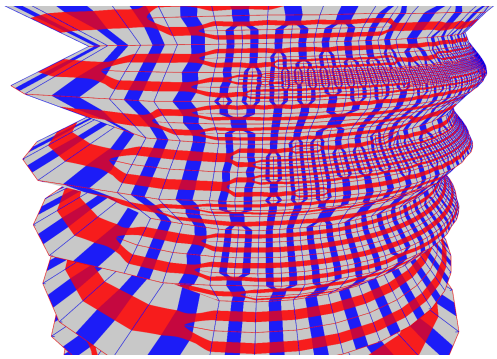


**Figure 10:** *An adaptive subdivision of the CAD model of a screw.*

aligned with principal directions of curvature, and an adaptively subdivided mesh obtained from it. Blue and red flows are shown both on edges and on faces by means of suitable textures. An example of adaptive refinement of a CAD model of a screw with red edges aligned to feature lines is shown in Figure 10.

## 6 Conclusion

The adaptive subdivision scheme we have presented has several advantages: it is fully compliant with the Catmull-Clark scheme; 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 surface.

The same approach can be extended in several directions. 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 adap-

tively refine and coarsen triangles [Puppo and Panozzo 2009], and composite computation of control points through averaging and smoothing [Stam and Loop 2003]. Another easy extension is to support management of creases and corners: it is sufficient to let the user flag creases in the base meshes, and to use stencils for creases (which are the same as stencils for boundary edges) during refinement.

Our current framework provides an approximated estimate of limit points of the subdivision surface, because the summations of neighbors used to evaluate the limit position of a vertex are not exact if not all neighbors are present in an adaptively refined mesh. A correct evaluation of the limit position at all vertices can be obtained by incorporating the mechanism proposed by Stam [Stam 1998a], which involves computing a polynomial patch for each face of the base mesh and tracing the parametric coordinates of each vertex that refines a given patch. A more efficient, although still approximated, solution can also be obtained by incorporating in a similar way the evaluation method described in [Loop and Schaefer 2008].

Our prototype tool is proposed mainly for interactive use in polygonal modeling. However, it can be easily extended to batch tasks, in which refinement/coarsening operations to be performed are selected automatically on the basis of LOD requirements. For instance, it may be useful to drive selective refinement either according to edge length (to obtain a more uniform mesh), or according to curvature (to obtain a better, possibly anisotropic, approximation of curved surfaces). In this case, many independent split/merge operations could be performed together on a mesh during selective refinement. Implementation of selective refinement on a data parallel architecture is probably possible and could be investigated.

## References


ALLIEZ, P., COHEN-STEINER, D., DEVILLERS, O., LÉVY, B., AND DESBRUN, M. 2003. Anisotropic polygonal remeshing. *ACM Transactions on Graphics 22*, 3 (July), 485–493.

BANK, R., SHERMAN, A., AND WEISER, A. 1983. Some refinement algorithms and data structures for regular local mesh refinement. In *Scientific Computing*, R. Stepleman, Ed. IMACS/North Holland, 3–17.

BOMMES, D., ZIMMER, H., AND KOBBELT, L. 2009. Mixed-integer quadrangulation. *ACM Trans. Graph. 28*, 3, 1–10.

BRISSON, E. 1993. Representing geometric structures in d dimensions: Topology and order. *Discrete and Computational Geometry 9*, 387–426.

CASTAÑO, I., 2008. Tessellation of displaced subdivision surfaces in dx11. GameFest 2008.

CATMULL, E., AND CLARK, J. 1978. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer Aided Design 10*, 350–355.

EISENACHER, C., MEYER, Q., AND LOOP, C. 2009. Real-time view-dependent rendering of parametric surfaces. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 137–143.

GEE, K., 2008. Direct3d 11 tessellation. GameFest 2008.

KOBBELT, L. 1996. Interpolatory subdivision on open quadrilateral nets with arbitrary topology. *Computer Grphics Forum 15*, 409–420.

KOBBELT, L. 2000. $\sqrt{3}$ subdivision. In *Proceedings ACM SIGGRAPH 2000*, 103–112.

KRAEMER, P., CAZIER, D., AND BECHMANN, D. 2009. Extension of half-edges for the representation of multiresolution subdivision surfaces. *Vis. Comput. 25*, 2, 149–163.

LOOP, C., AND SCHAEFER, S. 2008. Approximating catmull-clark subdivision surfaces with bicubic patches. *ACM Trans. Graph. 27*, 1, 1–11.

MAILLOT, J., AND STAM, J. 2001. A unified subdivision scheme for polygonal modeling. *Computer Graphics Forum 20*, 3, 471–479.

MARINOV, M., AND KOBBELT, L. 2004. Direct anisotropic quad-dominant remeshing. In *12th Pacific Conference on Computer Graphics and Applications*, 207–216.

MÜLLER, H., AND JAESCHKE, R. 1998. Adaptive subdivision curves and surfaces. In *Proceedings of Computer Graphics International '98*, 48–58.

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

Openmesh. `http://www.openmesh.org/`.

PAKDEL, H., AND SAMAVATI, F. 2007. Incremental subdivision for triangle meshes. *International Journal of Computational Science and Engineering 3*, 1, 80–92.

PANOZZO, D., AND PUPPO, E. 2009. Interpolatory adaptive subdivision for mesh lod editing. In *Proceedings GRAPP 2009 - International Conference on Computer Graphics Theory and Applications*, 70–75.

PATNEY, A., AND OWENS, J. D. 2008. Real-time reyes-style adaptive surface subdivision. *ACM Trans. Graph. 27*, 5, 1–8.

PUPPO, E., AND PANOZZO, D. 2009. RGB subdivision. *IEEE Transactions on Visualization and Computer Graphics 15*, 2, 295–310.

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

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

STAM, J., AND LOOP, C. 2003. Quad/triangle subdivision. *Computer Graphics Forum 22*, 1, 79–85.

STAM, J. 1998. Evaluation of Loop subdivision surfaces. In *SIGGRAPH '98 CDROM Proceedings*.

STAM, J. 1998. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *Proceedings SIGGRAPH '98*, 395–404.

VELHO, L., AND ZORIN, D. 2001. 4-8 subdivision. *Computer-Aided Geometric Design 18*, 397–427.

XU, Z., AND KONDO, K. 1999. Adaptive refinements in subdivision surfaces. In *Eurographics '99, Short papers and demos*, 239–242.

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