Eurographics Symposium on Geometry Processing (2006)
Konrad Polthier, Alla Sheffer (Editors)

Selectively refinable subdivision meshes

Enrico Puppo

Department of Computer and Information Sciences
University of Genova

Abstract

We introduce RGB triangulations, *an extension of red-green triangulations that can support selective refinement over subdivision meshes generated through quadrisection of triangles. Our purpose is to define a mechanism based on local operators that act on subdivision meshes while supporting operations similar to those available in Continuous Level Of Detail models. Our mechanism permits to take an adaptive mesh at intermediate level of subdivision and process it through both refinement and coarsening operations, by remaining consistent with an underlying Loop subdivision scheme. Our method does not require any hierarchical data structure, being based just on color codes and level numbers assigned to elements of a mesh, which can be encoded in a standard topological data structure with a small overhead.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Curve, surface, solid, and object representations

1. Introduction

Subdivision surfaces are used extensively in computer graphics and CAD. Subdivision schemes are based on the recursive refinement of the faces of a geometric mesh and converge to either C^1, or C^2 surfaces. Classical schemes are based on subdivision patterns that are applied to all faces of a mesh at each level. In practice, subdivision is often applied up to a certain level and the resulting mesh is used as an approximation of the limit surface [ZS00].

In some applications, it may be desirable to refine a mesh adaptively. This sort of mechanism is common in Continuous Level Of Detail (CLOD) models developed in the context of free-form mesh modeling and often applied in computer graphics [LRC*02]. In particular, *selective refinement* permits to vary the level of detail (LOD) smoothly across the mesh and dynamically through time. Transition between different LODs should be as smooth as possible and the resulting mesh should always be conforming (i.e., free of cracks). In order to support selective refinement efficiently, it is crucial that a mesh at intermediate LOD can be modified on-line in either way, by refining some parts of it while other parts may be coarsened. To this aim, refinement and coarsening operations must be based on local operators.

Transition between different levels is not easy in classical subdivision schemes, since non conforming situations arise. For instance, the popular Loop [Loo87] and butterfly [DLG90] schemes are based on recursive triangle quadrisection, which gives non-conforming meshes when applied adaptively at different levels of subdivision. Red-green triangulations [BSW83] have been widely used in the literature to obtain adaptive and conforming meshes for such subdivision schemes, but no efficient technique for selective refinement on such meshes has been designed so far.

In this paper, we present RGB triangulations, an extension of red-green triangulations that are built from iterative application of a local operator, namely *edge split*. Our method has the advantage of being progressive and to generate conforming meshes at all intermediate steps, which are consistent with the underlying subdivision scheme. This allows us to design a selective refinement algorithm for subdivision meshes based on triangle quadrisection, which exhibits the same features of the algorithms developed in the context of CLOD models.

We develop our method in the framework of Loop subdivision. We show that all levels of subdivision in the Loop scheme can be obtained and that control points of vertices are computed correctly in our scheme. This implies that our RGB meshes converge to the same surfaces obtained with the classical Loop method.

Differently from classical CLOD models, and from some other models supporting adaptive subdivision, our method does not require any hierarchical data structure. Selective refinement can be performed directly on a RGB mesh, which is maintained in a standard topological data structure, enriched with level numbers and color codes for triangles, edges and vertices, with a small overhead.

RGB triangulations may become a valid substitute or complement for standard subdivision of triangle meshes in solid modelers. Concerning applications in computer graphics, recent advances in reverse subdivision [Sab04] suggest that subdivision surfaces may become a valid alternative to CLOD models for free-form objects. In this view, RGB triangulations provide the tools to manage subdivision surfaces with the same flexibility of CLOD models also in this context.

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 introduce RGB triangulations. In Section 5 we describe the selective refinement algorithm. In Section 6 we describe how to set the position of vertices according to the Loop subdivision scheme. In Section 7 we describe the data structure used to implement selective refinement on RGB triangulations. Finally, in Section 8 we make some concluding remarks.

2. Related work

Subdivision surfaces. The literature on subdivision surfaces is quite extended. The interest reader can refer to [WW02] for a textbook, [ZS00] for a tutorial and [Sab04] for a survey. Here, we will review only those works related to adaptive subdivision of triangle meshes.

Red-green triangulations were introduced in the context of finite element methods [BSW83] as an empirical method to obtain conforming meshes from adaptive subdivision of triangle meshes. Red-green triangulations are usually built through a two-step procedure: first by applying triangle quadrisection adaptively, and then by subdividing some triangles further, through different patterns, to fix non conforming situations. Depending on the underlying subdivision scheme, the geometry of vertices (control points) that lie on the transition between different levels of subdivision may be different form that of the same vertices in a uniformly subdivided mesh. This fact, which is often overlooked, may prevent the correctness of further subdivision or coarsening of a red-green triangulation, unless the subdivision process is repeated from scratch.

Red-green triangulations were used in [ZSS97] to support multiresolution editing of meshes based on the Loop subdivision scheme. Adaptive meshes are computed by reversing subdivision, starting at the finest level and pruning over-refined triangles. Also in this case, a restricted nonconforming mesh is computed first, which is fixed next by further bisection of some triangles. Relocation of vertices is treated by using a hierarchical data structure that stores the positions of all control points in the uniform subdivision.

In [SHHG01], the quadrisection scheme is decomposed into atomic local operations, called *quarks*, based on the popular *vertex split* operation that is at the basis of Progressive Meshes [Hop96]. A red-green triangulation under the butterfly scheme [DLG90] is obtained through a sequence of quarks. Problems of topological consistency and relocation of vertices are treated by forcing some operations during refinement. The resulting mesh is over-refined with respect to a corresponding red-green triangulation computed with a traditional method. No explicit algorithm for selective refinement is proposed in [SHHG01].

The $\sqrt{3}$ subdivision [Kob00] and the 4-8 subdivision [VZ01] schemes are not based on the classical quadrisection operator. They are naturally adaptive, being both based on local conforming operators.

The $\sqrt{3}$ subdivision alternates triangle trisection (insertion of a new vertex at the center of each triangle) at one level, with edge swap at the next level. This scheme generates triangles that can be regarded as being of *green* and *blue* types in the terminology that we introduce in Section 4. The problem of correct relocation of vertices is addressed in [Kob00]. To this aim, some over-refinement of neighbors of even (green) triangles is imposed. A closed form solution of the subdivision rule permits to compute control points for a vertex at any level on the basis of just its initial position and its limit position. Adaptive refinement is supported, while adaptive coarsening is not investigated explicitly in [Kob00].

The 4-8 subdivision is based on edge split (as in our case) applied to a special case of triangle meshes, called *triquad meshes*. An initial tri-quad mesh can be obtained from any triangle mesh by doubling its number of triangles and changing its topology [VZ01]. The correct position of control points is addressed and resolved also in this case with a certain amount of over-refinement of the mesh. Only basic operations are investigated in [VZ01], while no selective refinement algorithm is proposed.

CLOD models. Also the literature on Continuous Level of Detail models is very wide. The interested reader may refer to [LRC*02] for a recent 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 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, developed in [Pup98], applies to almost all CLOD models proposed in the literature.

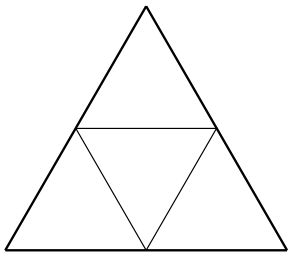

Figure 1: *The triangle quadrisection pattern.*

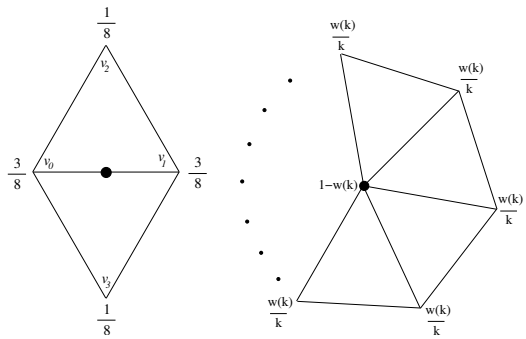

Figure 2: *The Loop subdivision scheme: the stencil used to compute the position of odd vertices (left); the stencil used to compute the position of even vertices (right). Numbers are weights assigned to vertices in the linear combination, k is the valence of the even vertex (*$k = 6$ *in the regular case) and* $w(k) = \frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\left[\frac{2\pi}{k}\right]\right)^2$.

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. The outer structure of the algorithm we propose for RGB triangulations is also based on this scheme.

There exist a few CLOD models based on recursive subdivision patterns. The model proposed in [DWS*97] is based on the recursive bisection of right triangles. This rule is also used by several other authors, and may be regarded as a subdivision. It can be applied just to meshes obtained from regular grids (typically representing terrains), while it is not easy to extend it to more general triangle meshes. One generalization is given by 4-k meshes [VG00], which have in fact a strong relation with 4-8 subdivision [VZ01].

3. Background

Triangle meshes. A *triangle mesh* is a triple $\Sigma = (V, E, T)$ where: V is a set of points in 3D space, called *vertices*; T is a set of triangles having their vertices in V and such that any two triangles of T either are disjoint, or share exactly either one vertex or one edge (thus, the mesh is inherently *conforming*); E is the set of edges of the triangles in T, where each edge is taken just once. Standard topological incidence and adjacency relations are defined over the entities of Σ.

We will assume to deal always with *manifold* meshes without boundary, i.e.: each edge of E is incident at exactly two triangles of T; and the *star* of a vertex (i.e., the set of entities incident at it) is homeomorphic to an open disc.

A triangle mesh is said to be *regular* if all its vertices have valence six. In a mesh that is not regular, vertices with a valence different from six are called *extraordinary*.

A *non-conforming mesh* is a structure similar to a mesh, in which triangles may violate the rule of edge sharing: there may exist adjacent triangles t and t' such that one entire edge of t overlaps just a portion of the corresponding edge of t'.

Loop subdivision. The Loop subdivision scheme was introduced in [Loo87] and it is the first and most famous subdivision scheme on triangle meshes. It is an approximating scheme, meaning that the position of control points is changed throughout levels of subdivision, and it converges to a C^2 surface if applied to a regular mesh.

The subdivision pattern is *triangle quadrisection*, as depicted in Figure 1, and it is applied to all triangles of the mesh at each level of subdivision. The position of each new vertex introduced from subdivision (called an *odd* vertex) is computed as weighted sum of vertices from the previous level (called the *even vertices*), as depicted in the stencil on the left of Figure 2. After inserting odd vertices at a given level, all even vertices are relocated according to the stencil on the right of Figure 2. For the sake of brevity we omit here and in the following the scheme for boundary vertices. Our method can be extended easily to treat meshes with boundary too.

Therefore, for a 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, $p^\infty(v)$ being its position on the limit surface.

For a vertex v of the base mesh position $p^0(v)$ is defined; while for a vertex v introduced at level l position $p^l(v)$ depends on positions p^{l-1} of vertices in its mask. The successive positions in the sequence are computed according to the mask for even vertices, such that for an even vertex v, $p^j(v)$ depends on $p^{j-1}(v)$ as well as on the positions p^j of all its neighbors (which are all odd vertices at level j).

Red-green triangulation. Consider a base mesh and assume the quadrisection scheme is applied adaptively to it. The resulting structure is a non-conforming mesh, as depicted in Figure 3a. This mesh is said to be *restricted* if two

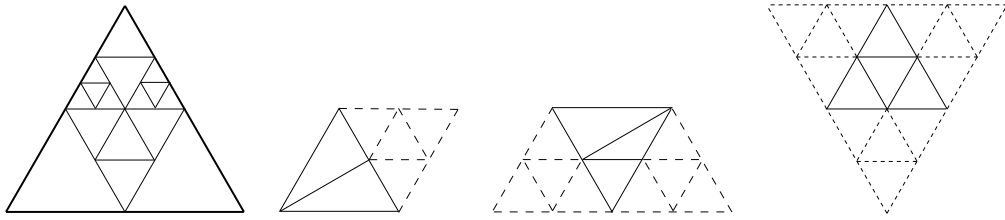

Figure 3: *Red-green triangulation: a non-conforming restricted mesh obtained from adaptive quadrisection (a); bisection is used to fix triangles that have one neighbor at the next level (b); trisection is used to fix triangles that have two neighbors at the next level (c). Triangles that have three neighbors at the next level are subdivided further by quadrisection (d).*

adjacent triangles may differ for no more than one level of subdivision. If a mesh is restricted, it may be made conforming by subdividing some triangles further, by either bisection, trisection, or quadrisection, as depicted in Figures 3b and c. Some authors avoid trisection by forcing further subdivision of triangles in the situation of Figure 3c. This fact, however, may propagate subdivision to adjacent triangles, possibly affecting a large area that will result over-refined.

In case both bisection and trisection are used, a given triangle may be subdivided by ten different patterns: three obtained by rotational simmetry from the pattern depicted in Figure 3b; six obtained by rotational simmetry and mirroring from the pattern depicted in Figure 3c; and one corresponding to the pattern depicted in Figure 3d. Note that, by construction, a triangle t that is subdivided with one of the first nine patterns must necessarily have neighbor(s), at all edges of t that are split, which were subdivided at the next level of subdivision by the thenth pattern.

All triangles that would appear in a standard subdivision are said to be green, while the other triangles that are introduced to make the mesh conforming are said to be red. In the following, we will introduce finer color codes for triangles and edges, in order to develop the details of our method. Green triangles and red triangles generated through bisection, as well as the red triangle generated through trisection, which is depicted as a square triangle in Figure 3c, will maintain the same color codes (green and red, respectively). On the contrary, in our framework, the triangle that is depicted as a skinny isosceles triangle in Figure 3c will be said to be blue. Note that, in both cases, the small equilateral triangle in Figure 3c, as well as all triangles in Figure 3d are green triangles at the next level of subdivision.

4. RGB triangulations

RGB triangulations are defined as all those triangulations that can be built through iterative application of given operators for local subdivision, starting at a base mesh Σ_0. Such operators always produce conforming meshes and can generate all and only those triangles that may appear in a red-green triangulation built from Σ_0, but the possible combinations of such triangles to form a mesh are more numerous that in red-green triangulations. In other words, RGB triangulations form a superset of red-green triangulations, with a higher expressive power. Reverse local operators to coarsen a mesh are also defined that, in combination with the subdivision operators, allows us to support selective refinement.

4.1. Local subdivision operators

Consider a base mesh Σ_0. We assign level zero to all vertices, edges and triangles of Σ_0, and color green to all edges and triangles of Σ_0. In the following, we define local subdivision operators that, when applied iteratively to Σ_0, will generate a conforming mesh where triangles will be colored of green, red and blue; edges will be colored of green and red; and vertices, edges, and triangles will have different levels. Color codes of red-green triangulations are extended here with further codes (blue triangles and colors for edges) in order to control the application of subdivision operators on a local basis.

For the sake of clarity, in our examples we will use regular meshes where all green triangles are equilateral, red triangles are square with one angle of sixty degrees and the other angle of thirty degrees, and blue triangles are isosceles with two angles of thirty degrees. However, our method is not restricted to such constraints, since color codes propagate according to recursive rules that do not depend on either geometry or valence of vertices.

We first define a sub-atomic rule, namely triangle bisection, which can be applied under certain conditions, and produces a certain configuration depending on the color of the triangle to be bisected. Next we define a first atomic rule, namely edge split, as the combination of triangle bisections applied to a pair of adjacent triangles. Finally we add a second atomic rule, namely edge swap, to be applied automatically only in a specific situation.

Triangle bisection can be applied only in the following configurations (see Figure 4):

- **G-bisection:** let t be a green triangle at level l and e be an edge of t (e can be any edge of t and it is always green

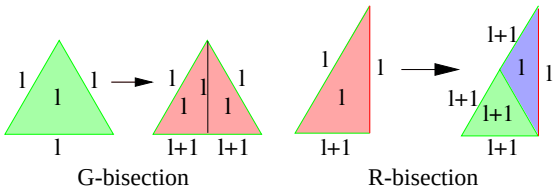

Figure 4: *Triangle bisection: bisection of a green triangle (G-bisection); bisection of a red triangle (R-bisection). Labels denote the level of edges and triangles.*

and at level l; vertices of t can be at any level $\leq l$). The bisection of t at the midpoint of e generates two red triangles at level l. Each such triangle will have: one green edge at level l (the one common with old triangle t), one green edge at level $l+1$ (one half of e) and one red edge at level l (the new edge inserted to split t). The new vertex inserted to perform bisection will have level $l+1$.

- **R-bisection:** let t' be a red triangle at level l and e' its green edge at level l (there is only one such edge). The bisection of t' at the midpoint of e' generates one blue triangle at level l and one green triangle at level $l+1$. The green triangle is incident at the green edge at level $l+1$ of old triangle t' and also its other two edges are at level $l+1$ (the edge inserted to subdivide t', and one half of e'). The blue triangle is incident at the red edge of old triangle t' and has also two green edges at level $l+1$ (the edge inserted to subdivide t', and the other half of e'). The new vertex inserted to perform bisection will have level $l+1$.

Edge split consists of the simultaneous bisection of two adjacent triangles t_0 and t_1 by splitting their common edge e, and can occur only if the bisections of both t_0 and t_1 along e are legal according to the rule defined above. It is readily seen that edge split is legal only in the following three cases (see Figure 5):

- **GG-split:** t_0 and t_1 are both green and at the same level;
- **RG-split:** t_0 is green and t_1 is red and they are both at the same level;
- **RR-split:** t_0 and t_1 are both red at level l and e is a green edge at level l. This case may come in two variants (RR_1-split and RR_2-split). Each variant can be recognized by the cycle of colors of edges on the boundary of the diamond formed by t_0 and t_1: this may be either red-green-red-green for RR_1-split, or red-red-green-green for RR_2-split.

BB-swap is applied automatically whenever two blue triangles at level l become adjacent along their red edge at level l (see Figure 6). In this case, such edge is eliminated and the other diagonal of the quadrilateral formed by such two triangles is inserted. The result is a pair of green triangles at level $l+1$ (all their edges are also green and at level $l+1$). See Figure 6. In practice, BB-swap will occur immediately after triangle bisection is applied to a red triangle that is already

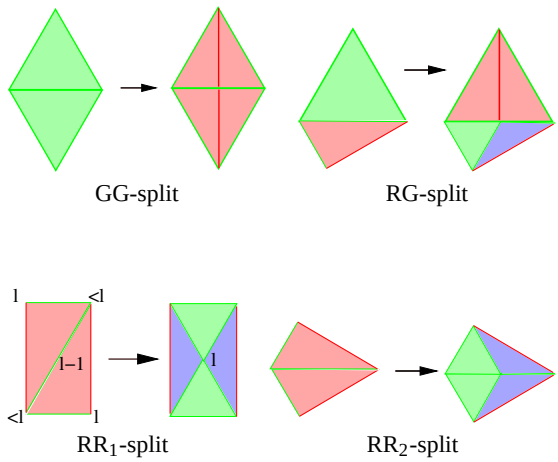

Figure 5: *Edge split: two green triangles (GG-split); one red and one green triangle (RG-split); two red triangles (RR_1- and RR_2-split). Labels denote the level of vertices and edges.*

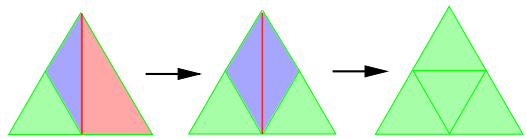

Figure 6: *BB-swap: when a red triangle is bisected, which was already adjacent to a blue triangle, two blue triangles become adjacent along a red edge. Such edge is swapped, thus producing two green triangles at the next level.*

adjacent to a blue triangle along its red edge. Note that, by construction, one of the two new green triangles will have all three vertices at level $l+1$.

Edge split is the main operator used to perform mesh refinement. It can be applied to legal pairs of adjacent triangles, selected according to rules that drive refinement, while swap is forced whenever two blue triangles become adjacent along a red edge. Note that just green edges can be split, while red edges are only swapped.

4.2. Consistency of RGB triangulations

The family of RGB triangulations from base mesh Σ_0 contains Σ_0 as well as all other meshes can be generated starting at Σ_0 and applying the refinement rules above. We claim that RGB triangulations form a superset of red-green triangulations generated starting at Σ_0 (hence, also of all uniform subdivisions of Σ_0).

We first prove that any red-green triangulation can be built starting at Σ_0 through the local operators defined before. Proof is by induction on the maximum level of subdivision m of a red-green triangulation Σ that subdivides Σ_0. By definition, if $m=0$ then $\Sigma \equiv \Sigma_0$ and it is obviously a RGB triangu-

lation. Let us suppose now that our claim is true up to level of subdivision $m-1$. Let Σ' be the mesh obtained from Σ as follows: all green triangles at level m and all red triangles at level $m-1$ (i.e., red triangles that subdivide green triangles at level $m-1$ to make the mesh conforming with triangles at level m) are removed, and the holes left are naturally filled with green triangles at level $m-1$. It is readily seen that Σ' is conforming and its maximum level of subdivision is $m-1$. Therefore, Σ' can be built through our local operators by inductive hypotesis. Now, in order to obtain back Σ, we must refine all those triangles at level $m-1$ that were used to fill the holes, each in its proper configuration. Let us call $S = S_r \cup S_g$ this set of triangles, where S_r [S_g] is the subset of S of triangles that will refine into red [green] triangles in Σ. Note that S_r is void if Σ is a uniform subdivision. Consistently with color codes of red-green triangulations, they are all green and at level $m-1$. We also assign green color to their edges, consistently with our color codes. All remaining triangles of Σ' already belong to Σ.

We refine first all triangles of S_r. Let $t \in S_r$, then t must split according to either bisection or trisection (see Figure 3b and c). Note that, since t is the first triangle that we subdivide, all its neighbors are green at level $m-1$, and they will eventually be refined into green triangles at level m (by construction rules of red-green triangulations). We may obtain bisection of t by applying GG-split; and trisection of t by applying GG-split followed by RG-split. Note that after this refinement, each triangle adjacent to t along a splitting edge is now subdivided into a pair of red triangles. Let $t' \in S_r$ be another triangle we take for subdivision, and let us consider its neighbors along edges that must be subdivided. If such triangles are green, then t' can be subdivided in the same manner as t. Otherwise, t' might have some red neighbors that have been obtained by subdividing triangles of S_g because of edge splits applied to subdivide some other triangle of S_r processeded before t'. By exhaustive analysis of the possible configurations, it is easy to see that bisection of t' can be obtained by applying either a GG-split or a RG-split, depending on whether the neighbor is green or red; and trisection of t' is obtained by the same operation, followed by either a GR-split or RR2-split, depending on whether the neighbor is green or red. All remaining triangles of S_r can be subdivided as t'.

At this point, we have obtained the correct refinement of all triangles of S_r and partial refinement of all those triangles of S_g that were adjacent to triangles of S_r. Let us consider now $t \in S_g$ and see how it has been refined so far. There are four possibilities:

- All three edges of t have been split. In this case, the triangles that subdivide t are now in the configuration depicted in the center of Figure 6. A BB-swap is sufficient to complete subdivision of t to level m, without affecting its neighbors;
- Two edges of t have been split. In this case, the triangles that subdivide t are now in the configuration depicted on the left of Figure 6. Note that the neighbor t' of t on its right side (referring to the figure) must necessarily belong to S_g or be a red triangle subdividing a triangle in S_g. Therefore, we first apply either a RG-split or a RR*-split (where * may be 1 or 2 depending on configuration) depending on t' being either green or red. Next we apply edge swap as in the previous case.
- One edge of t has been split. In this case, t has been bisected into two red triangles as in the case of G-bisection. This case is similar to the previous, but the first split must be performed on both sides of t (i.e., on both red triangles refining it).
- No edge of t has been split yet. In this case, all three neighbors of t must either belong to S_g or subdivide triangles of S_g. We first apply either a GG-split or a RG-split, depending on whether the neighbor of t along the splitting edge is green or red. Then we proceed as in the previous case.

By exhaustive analysis of possible configurations, it is easy to see that the neighbors of t affected by the split operations we perform will get to one of the first three configurations. Therefore, the repeated application of these operations will eventually refine all triangles of S_g to level m. At that point, we have obtained mesh Σ.

In order to complete the proof of our claim, we must show that there exist RGB triangulations that are not red-green triangulations. One example is the mesh obtained from a pair of adjacent green triangles by applying a GG-split. The mesh depicted in Figure 9 is a more elaborated example. It is interesting to notice how RGB triangulation may manage fast transitions of LOD better than red-green triangulations. Consider the RGB triangulation on the left of Figure 8. The LOD transition from the base to the apex of the big triangle may be carried out to an arbitrary number of levels by using the same pattern. The equivalent transition using red-green triangulations is depicted on the right side of the same figure and requires about twice the number of triangles.

4.3. Reverse subdivision operators

Since a selective refinement algorithm also needs to reverse subdivision (i.e., to coarsen a mesh), we define also local operators that invert edge split and edge swap.

Triangle merge is the reverse operation of triangle bisection and it is defined as follows:

- **RR-merge:** a pair of red triangles t_0, t_1 of level l that are adjacent at a red edge may merge into one green triangle at level l; the two green edges at level $l+1$ of t_0 and t_1 are merged to form a new edge e at level l of the new triangle; the red edge and its endpoint at level $l+1$ disappear.
- **GB-merge:** A pair of adjacent triangles t_0, t_1 such that t_0 is green at level $l+1$ and t_1 is blue at level l may merge into one red triangle at level l; the common edge of t_0 and t_1 and the endpoint v of such edge that is incident at both

green edges of t_1 disappear; the other two green edges that were incident at v are merged to form one green edge at level l.

Edge merge is the reverse operation of edge split and can be applied to triangles incident at vertices of valence four, such that triangle merge can be applied to pairs of such triangles. The same cases depicted in Figure 5 occur (modifications apply right-to-left in this case):

- **R4-merge** inverts GGsplit;
- **R2GB-merge** invertes GR-split;
- **GBGB-merge** inverts RR_1-split;
- **G2B2-merge** inverts RR_2-split.

A little care must be taken in applying GBGB-merge in order to avoid inconsistencies. Referring to Figure 5, note that the quadrilateral must have two vertices at the same level l and two other vertices at a level lower than l. GBGB-merge must be performed in such a way that the diagonal incident at vertices of level lower than l is maintained.

Reverse edge swap cannot be applied to any pair of adjacent green triangles, otherwise the structure of subdivision would be destroyed. **GG-swap,** which inverts BB-swap, can be applied to a pair of adjacent green triangles t_0 and t_1 at level l if one of them, say t_0, has all three vertices at level $l > 0$. This condition is necessary and sufficient to guarantee that t_0 and t_1 have the same parent triangle t in the subdivision and t_0 is the central triangle obtained by subdividing t. In order to be consistent with refinement rules, reverse edge swap should be applied only if one of the blue triangles generated is removed immediately after through an edge merge operation. We therefore constrain reverse edge swap to occur only in the configurations depicted in Figure 7. These configurations are easy to verify by exploring the stars of vertices common to t_0 and t_1, and checking the colors and levels of their incident triangles.

With these local operators at hand, we can modify any RGB triangulation by either refining or coarsening it locally. This means that we can traverse the whole family of RGB triangulations originated from a given base mesh, reaching any adaptively subdivided mesh that may be needed from an application. This task will be performed through the selective refinement algorithm described in the following section.

5. Selective refinement

Selective refinement is applied to a RGB triangulation Σ and consists of the iterated application of local operators defined in the previous until some user-defined halt condition is verified.

Following [CDM*04, DWS*97], selective refinement is driven by two priority queues:

- A queue Q_r of refinement operations to be performed to meet LOD requirements. Refinement operations are related to edges to split. Queue Q_r is initialized by considering all green edges of Σ and inserting into Q_r all and only those edges that need refinement. Depending on user needs, priority of an edge may depend, e.g., from its length, or from distance between its midpoint and the position of the vertex to split it at next level of subdivision.
- A queue Q_c of coarsening operations to be performed whenever possible to reduce the size of the mesh without violating LOD requirements. Coarsening operations are related to vertices to be removed through edge merge operations. Queue Q_c is initialized by considering all vertices of Σ that may be removed from an edge merge operation, possibly preceded from suitable edge swap operation(s), and inserting them into Q_c. Priority of a vertex in the queue will be set consistently with the previous case.

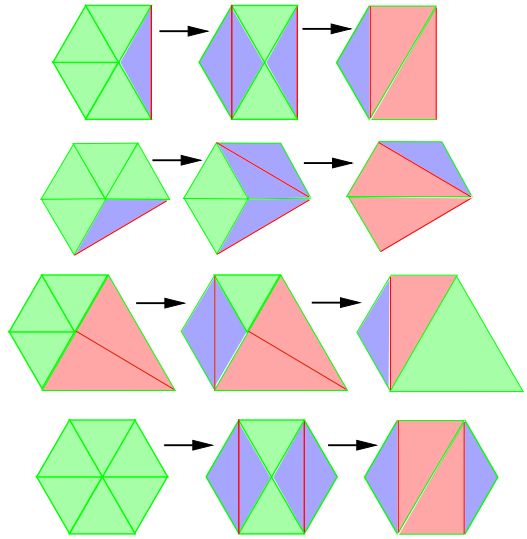

Figure 7: *Legal configurations for applying GG-swap.*

After initialization, elements are popped from either Q_r or Q_c, according to higher priority, and related operations are applied to modify Σ, until a halt condition is verified. This may typically depend either on the full satisfaction of LOD requirements, or on the size of the extracted mesh. See [CDM*04] for more details about managing the queues during selective refinement.

The most crucial aspects of the algorithm are related to refinement and coarsening operations to be performed on a RGB triangulation.

Coarsening. Assume a vertex v having level l has been popped from Q_c. If the star of v in Σ has one of the configurations depicted in Figure 7, then the corresponding reverse swap is applied first. At this point, the star of v will have one of the four configurations depicted in Figure 5 (right side). The following operations are performed:

1. Edge merge is applied and Σ is updated accordingly;
2. The four vertices that were adjacent to v are analyzed and those vertices that can be now legally removed are inserted into Q_c; the control point of each vertex is updated (see next section).

Refinement. Refinement is more complex, since edges in Q_r may require further refinement operations on other edges prior to be refined. Assume a green edge e having level l has been popped from Q_r. If the pair of triangles incident at e do not form one of the configurations depicted in Figure 5 (left side) then such triangles are analyzed to trigger recursive edge split operations. Let t be one such triangle:

- if t is a red triangle at level $l-1$ then its green edge at level $l-1$ is recursively split;
- if t is a blue triangle at level $l-1$ then t must be adjacent to a red triangle t' at level $l-1$ along its red edge; the green edge of t' at level $l-1$ is split recursively;
- otherwise no action is required.

Performing a single edge split involves the following operations:

1. Compute the control point at level $l+1$ for the new vertex v (see next section);
2. Update the control point at level $l+1$ at the four vertices of triangles to be split;
3. Update mesh Σ performing edge split;
4. Test each new green edge generated from split and add it to Q_r if it does not fulfill LOD requirements.

As we will see in the next section, computation of the control point for the new vertex may require some extra refinement of the mesh, necessary to obtain the correct values for the Loop stencil of odd vertices. 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 are tested for LOD requirements. In case one such split is not necessary according to LOD requirements, the new vertex generated from split is 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 operations.

6. Geometry of control points

So far we have been concerned only with topological changes in a RGB triangulation. We now study the geometry of vertices. As we work in the framework of Loop subdivisions, control points of all vertices are computed at all levels through the proper stencils. However, since RGB triangulations work selectively, it may be not trivial to determine the right vertices to use for a stencil, and their proper control points.

Updates to control points must be done for odd vertices during refinement, and for even vertices both during refinement and during coarsening. We address the three relevant cases in the following.

Control point for an even vertex updated after refinement. When an edge e at level l is split by introducing a new (odd) vertex v, its control point $p^{l+1}(v)$ is computed (see next paragraph) and the control points p^{l+1} of end vertices v_0 and v_1 of e need to be updated with a contribution from $p^{l+1}(v)$. Without loss of generality, let us consider the case of v_0. We adopt a lazy strategy, which generates $p^{l+1}(v_0)$ with a partial value as soon as the first neighbor of v_0 at level $l+1$ is introduced, and mark it as *restricted*. Then the value of $p^{l+1}(v_0)$ is updated every time another neighbor of v_0 at level $l+1$ is introduced, by adding its contribution according to the formula used for even vertices in the Loop scheme. After the last neighbor has been introduced, control point $p^{l+1}(v_0)$ gets its correct value by combining the current summation with value $p^i(v_0)$ (that must be already available, by recursion) and is marked *available*.

Control point for an odd vertex introduced via refinement. Let vertex v be introduced at level $l+1$ of subdivision by splitting an edge e at level l. In order to compute control point $p^{l+1}(v)$, we need to fetch the four vertices in the stencil of v at level l, and their control points p^l (see Figure 2).

In a RGB triangulation Σ, vertices v_0 and v_1 of the stencil of v are the endpoints of edge e to split, so they are found immediately. On the contrary, v_2 and v_3 are not necessarily vertices of the triangles incident at e. In fact, such triangles might have been refined already to higher levels of subdivision. Without loss of generality, let us consider just the case of vertex v_2. Let t be the triangle incident at e on the side of vertex v_2 and let us refer to Figure 8 (left side). If t is green, then vertex v_2 is its vertex opposite to e. Otherwise, t must be red and its neighbor t' on the other side of its red edge e' must be either red or blue. If t' is red, then vertex v_2 is the vertex of t' opposite to t, otherwise the neighbor t'' of t' on the side of 2 is retrieved (this can be decided by looking whether the red edge is incident at v_0 or at v_1). If t'' is green, then vertex v_2 is its vertex opposed to t', otherwise we set $t \leftarrow t''$ and restart the search. This search can be implemented through a simple while cycle and may take $O(m-l)$ time, where m is the maximum level of subdivision, in the worst case (that can occur only in the proximity of an abrupt change of LOD). If Σ were refined uniformly, we will have $m \in O(\log N)$, where N is the number of vertices of Σ. In theory, we may have $m \in O(N)$ in the worst case (e.g., if high level of subdivision is used only in the proximity of a given point and degrades abruptly to level zero in the other parts of the mesh). In practice, since a subdivision mesh is usually built on a small number of levels, and abrupt changes of

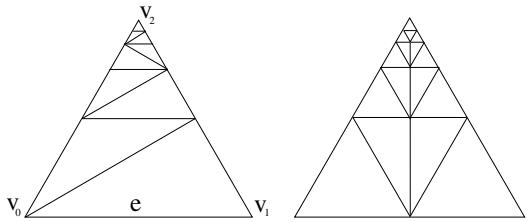

Figure 8: *Left side: finding vertex v_2. The triangle incident at e at level l might have been split through all successive levels of subdivision. Right side: in order to achieve the same transition of LOD from base to apex of the big triangle, almost twice the number of triangles is necessary in red-green triangulations.*

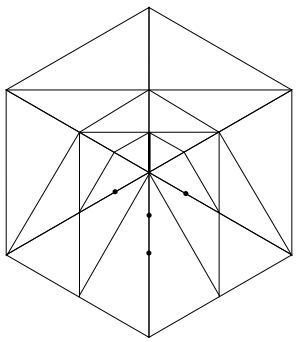

Figure 9: *In order to bisect the edge in bold, vertices marked by bullets must be computed by recursive edge split.*

LOD seldom occur, we may consider this procedure to run in constant time.

Once a vertex v_i, $i = 0,1,2,3$ is fetched, we check whether its control point $p^l(v_i)$ is available or not. If not, we need to compute it recursively by splitting edges in the star of v_i, through the procedure already described in Section 5. All green edges incident at v_i are considered. For each such edge e_j, if it is at level $l' < l$, we split it to generate a new vertex v'_j and its control point $p^{l'}(v'_j)$. Split is repeated until the edge reaches level l. Note that a regular vertex may have green incident edges that differ for at most three levels, thus the number of new vertices to be computed during this operation is usually quite small (see Figure 9).

Control point for an even vertex updated after coarsening. When a vertex v at level $l+1$ is removed from an edge merge operation, the four vertices that were adjacent to v are checked and, for each such vertex v_i, if its current position in the mesh is at a control point of level higher than l, then it is changed to $p^l(v_i)$. However, the control point $p^l(v_i)$ in the list of control points of v_i is not changed and it is maintained *available* for subsequent processing.

7. Data structure

A RGB triangulation can be maintained in a standard topological data structure for triangle meshes. One possibility is using two dynamic arrays, one for vertices and the other for triangles, with a garbage collection mechanism to manage reuse of locations freed because of coarsening operations. For each triangle, links to its three vertices as well as to its three neighbors are maintained. For each vertex, just a link to one of its incident triangles is maintained (this is sufficient to compute the star of a vertex in optimal time).

This data structure is extended as follows. For each vertex v we store: its level, a link to a linked list of its control points, and a link to the control point in the list corresponding to the position of v in the current mesh. Each control point in a list contains its three coordinates, and a flag to indicate whether it is available or restricted. This flag is actually a counter that is initialized either at six, or at the valence of the vertex if it is an extraordinary vertex, and is decremented each time a contribution to computation of the control point is added. Value zero means that the control point is available.

For each triangle we store its color and its level. We use two different codes for red triangles, depending whether the short (higher level) green edge is followed by the other green edge, or it is followed by the red edge when traversing the triangle counterclockwise. Since two bits are sufficient for the color, and levels in subdivision are usually not many, one byte is sufficient to store both color and level. Edges are addressed as pairs triangle-index, where index can take values 0, 1, 2. Depending on the color of a triangle we store edges in entries 0, 1, 2 in a conventional order (e.g., for a blue triangle 0=red 1=green 2=green). Because we used two different codes for red triangles, this is sufficient to unambiguously determine color and level of each edge.

Since selective refinement is meant to be used dynamically, we set up a caching mechanism to save vertices and their related control points when they are removed from the mesh because of edge merge operations. A caching policy based on least recently used vertex is adopted to manage the cache. Vertices in cache can be restored together with all their control points once they are reinserted in the mesh because of an edge split operation.

8. Concluding remarks

We have introduced RGB triangulations, a mechanism for the subdivision of triangle meshes that can support fully dynamic selective refinement and is compatible with classical subdivision schemes based on the recursive quadrisection of triangles.

We have developed our selective refinement algorithm for the Loop subdivision scheme. However, an analogous algorithm can be developed similarly for the (modified) butterfly subdivision [DLG90, ZSS96]. In that case, subdivision is interpolating, thus each vertex has a fixed position and the data

structure becomes simpler. Even vertices do not need any update, while the stencil for odd vertices is larger than in Loop subdivision, and needs a slightly more complex procedure to compute vertex position. In the case of Loop subdivision, it should be also possible to develop a closed form solution for the control points of even vertices, similar to that proposed in [Kob00], which would also allow us to use a simpler data structure. We plan to investigate these issues in our future work.

RGB triangulations are currently under implementation. We expect that our method will outperform red-green triangulations in terms of over-refinement, by maintaining a comparable visual quality. We also plan to compare the performance of RGB triangulations with respect to $\sqrt{3}$-subdivision and 4-8-subdivision, in terms of speed, storage space, visual quality and over-refinement.

Acknowledgments

This work has been partially supported by the European Network of Excellence AIM@SHAPE under contract number 506766, and by Project FIRB-MIUR SHALOM (SHApe modeLing and reasOning: new Methods and tools) funded by the Italian Ministry of Education, University and Research under contract number RBIN04HWR8.

References

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

[CDM*04] CIGNONI P., DE FLORIANI L., MAGILLO P., PUPPO E., SCOPIGNO R.: Selective refinement queries for volume visualization of unstructured tetrahedral meshes. *IEEE Transactions on Visualization and Computer Graphics 10*, 1 (January/February 2004), 141–159.

[DLG90] DYN N., LEVIN D., GREGORY J.: A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics 9*, 2 (April 1990), 160–169.

[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.

[Hop96] HOPPE H.: Progressive meshes. In *SIGGRAPH 96 Conference Proceedings* (Aug. 1996), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 99–108.

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

[Loo87] LOOP C.: Smooth subdivision surfaces based on triangles. Master thesis, University of Utah, Dept. of Mathematics, 1987.

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

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

[Sab04] SABIN M.: Recent progress in subdivision: a survey. In *Advances in Multiresolution for Geometric Modelling*, Dogdson N., Floater M., Sabin M., (Eds.). Springer-Verlag, 2004, pp. 203–230.

[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.

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

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

[WW02] WARREN J., WEIMER H.: *Subdivision Methods for Geometric Design*. Morgan Kaufmann, 2002.

[ZS00] ZORIN D., SCHRÖDER P. (Eds.): *Subdivision for Modeling and Animation (SIGGRAPH 2000 Tutorial N.23 - Course notes)*. ACM Press, 2000.

[ZSS96] ZORIN D., SCHRÖDER P., SWELDENS W.: Interpolating subdivision for meshes with arbitrary topology. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH 96)* (1996), ACM Press, pp. 189–192.

[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.