A Sub-Atomic Subdivision Approach

S. Seeger[1] K. Hormann[2] G. Häusler[1] G. Greiner[2]

[1] University of Erlangen, Chair for Optics
Staudtstr. 7/B2, 91058 Erlangen, Germany
Email: {sseeger,ghaeusler}@optik.uni-erlangen.de

[2] University of Erlangen, Computer Graphics Group
Am Weichselgarten 9, 91058 Erlangen, Germany
Email: {hormann,greiner}@informatik.uni-erlangen.de

Abstract

One of the main objectives of scientific work is the analysis of complex phenomena in order to reveal the underlying structures and to explain them by means of elementary rules which are easily understood. In this paper we study how the well-known process of triangle mesh subdivision can be expressed in terms of the simplest mesh modification, namely the vertex split. Although this basic operation is capable of reproducing all common subdivision schemes if applied in the correct manner, we focus on Butterfly subdivision only for the purpose of perspicuity. Our observations lead to an obvious representation of subdivision meshes as selectively refined progressive meshes, making them most applicable to view-dependent level-of-detail rendering.

1 Related Work

Subdivision of triangle meshes is usually based on a *dyadic* edge split where a new vertex is inserted for every edge of the given triangle mesh and then these new vertices are connected. As a consequence each face is split into four triangles, motivating the term *1-to-4 split* (cf. Fig. 1). The new vertices are also known as *odd* vertices, whereas the old vertices of the given mesh are often referred to as *even* vertices.

Another variant of refining triangle meshes has been proposed recently in [6, 7]. It inserts new vertices at the center of each triangle, connects them with the old vertices, and replaces each original edge by the one that connects the two new vertices of the two adjacent triangles. Performing this scheme twice results in a triadic edge and a 1-to-9 face split, and thus this method is called $\sqrt{3}$ subdivision.

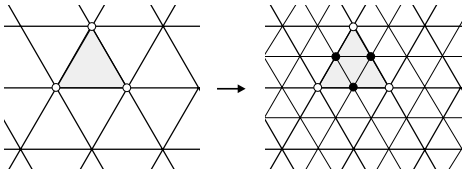

Figure 1: Subdivision of a triangle mesh by 1-to-4 splits.

These splitting operations per se concern the topology of the refined mesh only and additional rules are required for determining the geometry. This is usually done by specifying how to describe the vertices of the subdivided mesh as affine combinations of the neighboring even vertices. One of the most popular of such averaging rules for the 1-to-4 split was given in [2] and is known as *Butterfly subdivision*.

In contrast to *approximating* schemes like *Loop subdivision* [8] which give rules for placing both the old and the new vertices, Butterfly subdivision is an *interpolating* method and does not modify the positions of the even vertices. Repeatedly quadrisecting an initial triangle mesh and setting the odd vertices according to the Butterfly masks in Fig. 2 gives a surface in the limit that is C^1-continuous almost everywhere except at extraordinary vertices with valence $k = 3$ and $k > 7$. A modified Butterfly scheme that ensures overall C^1-continuity was proposed in [12].

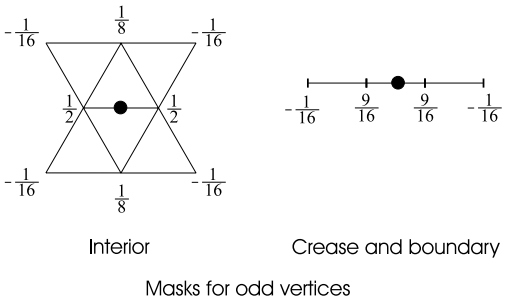

Figure 2: Butterfly subdivision.

Besides *global* or *uniform* subdivision, where the same splitting and averaging rules are used for all triangles and vertices of a given mesh, *adaptive subdivision* permits to restrict the refinement to regions of interest [13]. In order to avoid T-vertices, which would result in a discontinuous surface, additional edges are introduced according to the rules of the so-called *red-green triangulation* [1, 11]. The usual 1-to-4 split is called a *green* split. Unrefined triangles neighboring one and only one quadrisected triangle are split by a *red* 1-to-2 split in order to fix the T-vertex. If two neighbors of an unrefined triangle were subdivided, a green split will be applied to this triangle itself, causing a further red or green split of the third neighboring triangle. In this way the difference between the subdivision levels of neighboring triangles is never greater than 1 (cf. Fig. 3).

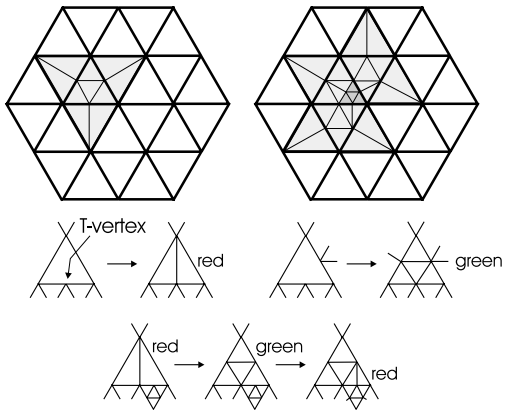

Figure 3: Two levels of usual adaptive subdivision: In order to avoid T-vertices additional edges are introduced according to the rules of red-green triangulation.

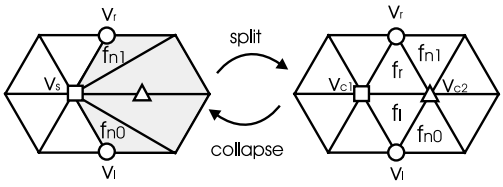

Figure 4: Vertex split and edge collapse.

While subdivision introduces additional vertices to a triangulation, *mesh decimation* algorithms remove vertices. The approach taken in [3] reduces the number of vertices by iteratively applying *edge collapses* as fundamental decimation steps. This operation removes an edge $\overline{\mathrm{v}_{c1}\mathrm{v}_{c2}}$ and the adjacent triangles by combining the vertices v_{c1} and v_{c2} to a single vertex v_s (cf. Fig. 4). Note that v_{c1} and v_s can be identified topologically although their geometric positions may differ.

The inverse operation of an edge collapse is called a *vertex split*. It replaces the vertex v_s with v_{c1} and v_{c2} and inserts the edge $\overline{\mathrm{v}_{c1}\mathrm{v}_{c2}}$ as well as the two faces $f_l = \triangle(\mathrm{v}_{c1}, \mathrm{v}_l, \mathrm{v}_{c2})$ and $f_r = \triangle(\mathrm{v}_{c1}, \mathrm{v}_{c2}, \mathrm{v}_r)$. We refer to the collection of faces between f_{n0} and f_{n1} as the *support* of a vertex split operation.

In Fig. 4 and throughout this paper we use the following symbols to briefly characterize a vertex split operation. The split vertex (v_s, v_{c1}) is depicted by $\square$, the vertices defining the support of the split operation (v_l, v_r) are represented by $\bigcirc$, and $\triangle$ is used for the newly introduced vertex (v_{c2}).

By applying a sequence of edge collapses to a given triangle mesh $\hat{M}$,

$$\hat{M} = M^n \xrightarrow{\mathrm{ecol}_{n-1}} M^{n-1} \xrightarrow{\mathrm{ecol}_{n-2}} \dots \xrightarrow{\mathrm{ecol}_0} M^0,$$

the complexity of $\hat{M}$ is successively reduced up to the simple base mesh M_0. The sequence of edge collapses is usually chosen such that the intermediate meshes M^i are optimal approximations of $\hat{M}$ with respect to some appearance metric.

The original mesh $\hat{M}$ can later be reconstructed by applying the vertex splits that correspond to the edge collapses to the base mesh in reverse order,

$$M^0 \xrightarrow{\mathrm{vsplit}_0} M^1 \xrightarrow{\mathrm{vsplit}_1} \dots \xrightarrow{\mathrm{vsplit}_{n-1}} M^n = \hat{M}.$$

The base mesh M^0 and the vertex split operations $\mathrm{vsplit}_0, \dots, \mathrm{vsplit}_{n-1}$ define a hierarchy of triangle meshes that is called a *progressive mesh* (PM) [3].

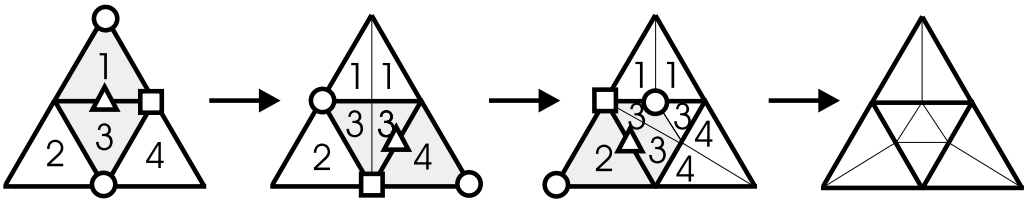

Figure 6: Generating an adaptive 1-to-4 subdivision split with three successive vertex split operations. The support of all quarks contains exactly two different triangle indices (B-quarks).

Since a vertex split is a local operation many of the split operations are independent of each other and can therefore be performed in a different order which allows to selectively refine regions of interest. The only restriction on the order in which the split operations can be carried out is that a vertex split can only be performed if the split vertex v_s and the triangles f_{n0} and f_{n1} are already present in the current mesh. Moreover, in the context of view dependent meshes intermediate edge collapses need to be performed in order to adapt the mesh to changing viewing parameters [4]. This requires the selectively refined mesh to be unique, regardless of the sequence of vertex splits and edge collapses that led to it. To satisfy this requirement, both vertices v_{c1} and v_{c2} are stored as children of the split vertex v_s, whereas in a usual PM the vertex v_s is simply replaced by v_{c1}. The duplication of the split vertex leads to a vertex hierarchy called the *vertex forest* and each selectively refined mesh corresponds to an *active vertex front* through this hierarchy (cf. Fig. 5).

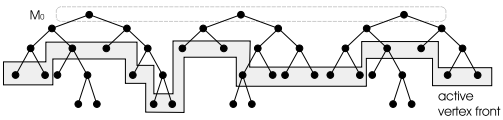

Figure 5: The vertex forest with an active vertex front.

2 Topology of Subdivision Quarks

As we have seen in the previous section there are several ways of refining the topology of a triangle mesh which differ by the sort and the extent of modification they apply to the mesh. While the vertex split adds one vertex, two triangles, and three edges only, a global 1-to-4 subdivision step quadruples these quantities. Regarding the complexity of such topological operators, the natural question arises whether the more complex ones can be expressed in terms of the simpler ones. In the same way that physicists use elementary particles for describing atoms, we will explain in this section how to express complex topological refinement operators such as 1-to-4 and $\sqrt{3}$ subdivision in terms of the most elementary one, namely the vertex split. In order to stick to the analogy of physics [5, 9] we call these elementary operations *subdivision quarks* and will find them to come in different types (e.g. M-, B-, and T-quarks).

Let us consider the 1-to-4 split of a single triangle first. Such a split requires to insert three new vertices, one for each edge, and we therefore expect that such an operation can be represented by three subdivision quarks. Fig. 6 confirms that we obtain indeed the regular quadrisection of a triangle after applying three vertex split operations in the correct manner. Note that the problem of T-vertices is automatically avoided since vertex splits always preserve the validity of a triangle mesh.

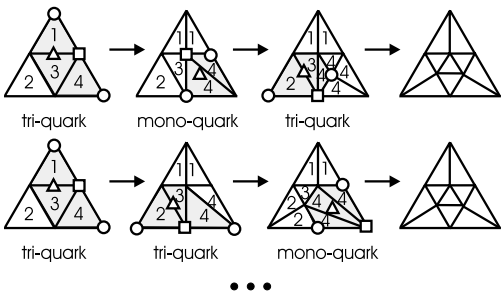

Figure 7: It is also possible to reproduce a 1-to-4 split via quarks whose support contains only one or even three triangle indices (M-quarks and T-quarks).

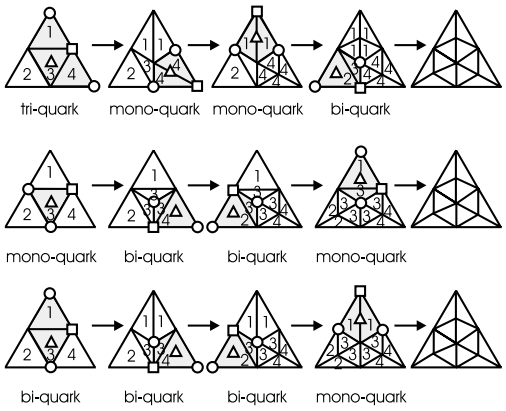

Figure 8: In order to reproduce $\sqrt{3}$ subdivision we need M-quarks and/or T-quarks.

A deeper analysis shows that there are many ways of representing a 1-to-4 split by three subdivision quarks (cf. Fig. 7). In order to be able to manage this diversity we categorize subdivision quarks in the following way. Suppose we provide each triangle in the given mesh with a unique index and let the new faces f_l and f_r inherit the index from f_{n0} and f_{n1} respectively whenever a vertex split is performed (cf. Fig. 4). Then each subdivision quark can be classified by the number d of different triangle indices of all triangles in the support of this operation:

d	quark type
1	mono-quark (M-quark)
2	bi-quark (B-quark)
3	tri-quark (T-quark)
$\vdots$	$\vdots$

In this work we restrict ourselves to the analysis of B-quarks since they are sufficient for representing 1-to-4 subdivision. Note that $\sqrt{3}$ subdivision cannot be described by B-quarks solely (cf. Fig. 8).

Let T_1 and T_2 be the sets of those triangles in the support of a B-quark that have the same triangle index. We can then divide any B-quark into two *preons* [10], one operating on T_1, the other on T_2. Since T_1 as well as T_2 naturally correspond to one of the coarse triangles in the initial triangle mesh we can further view each preon as to successively refining a coarse triangle until it has been quadrisected. Fig. 9 shows all possible configurations of a preon.

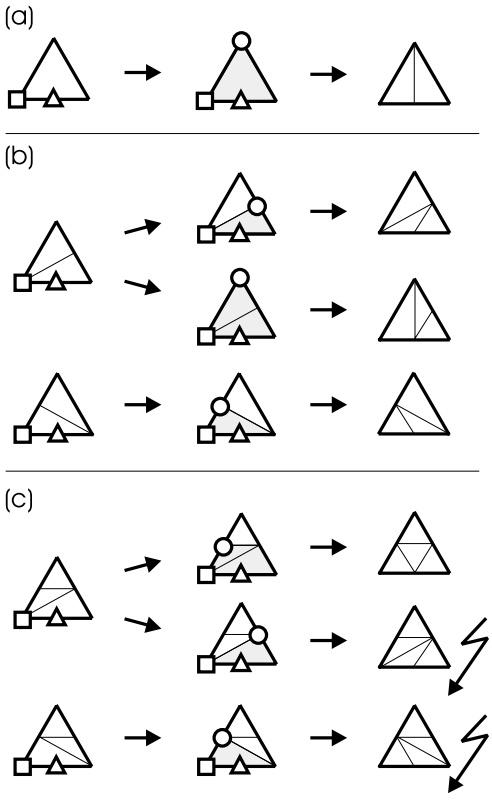

Figure 9: All possible configurations (except for symmetry) of a preon (one half of a B-quark). Any B-quark is a combination of one of the above preons with a mirrored version of another preon such that the split vertex □ and the new vertex △ coincide.

If a triangle is refined by a preon for the first time the configuration will always be the one shown in Fig. 9 a. In case a preon is about to refine a triangle that has already been split once, the three configurations in Fig. 9 b can occur. Whenever a triangle is split for the third time, this last vertex split will not necessarily result in a regular quadrisection of the triangle. Only the topmost preon in Fig. 9 c leads to the desired configuration, the other two preons are therefore *invalid*. We have to exclude all B-quarks that contain an invalid preon if we want to reproduce dyadic subdivision connectivity.

During the quark subdivision refinement process we may encounter the situation displayed in Fig. 10, where the remaining unsplit edge can only be re-

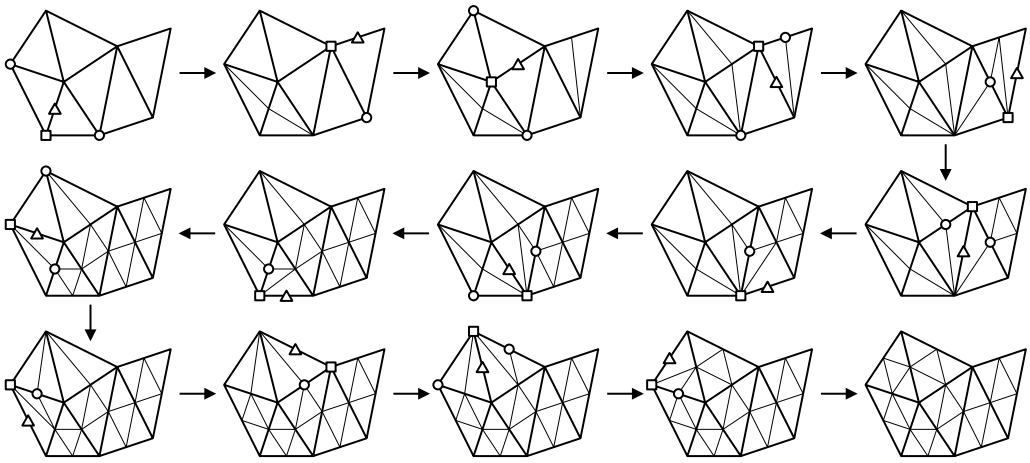

Figure 11: Global 1-to-4 subdivision of a triangle mesh by successive B-quark subdivision steps. Note that steps 5, 7, 8, 9, 11, 13, and 14 were forced by the regularity rule.

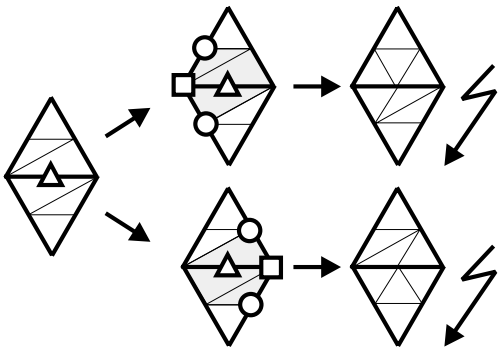

Figure 10: Regardless of which B-quark we choose to create the new vertex in the center, the result is not a valid subdivision structure. Therefore we have to ensure this configuration never to occur.

fined by a forbidden B-quark and there is no way of obtaining regular subdivision connectivity through quark subdivision. In order to avoid these cases, we always force the third split of a triangle to be performed immediately after the second one and take a valid B-quark for this purpose. This *regularity rule* is similar to the rules of red-green triangulations in adaptive subdivision [1, 11].

Respecting the regularity rule we can apply successive B-quark subdivision steps to a given triangle mesh in *any* order and will always end up with a regular 1-to-4 subdivision structure (cf. Fig. 11). As a criterion for the order in which the B-quarks are to be carried out we could take the area of the two triangles adjacent to the edge that is going to be split by the quark subdivision step and perform those vertex splits first for which this value is largest. However, other criteria, view dependent ones, for example, are possible.

So far we have only considered quark subdivision that leads to one level of 1-to-4 subdivision. If we want to adaptively refine parts of the triangle mesh up to further subdivision levels we may have to force some quark subdivision steps in the neighborhood of the split vertex similarly to the rules of adaptive red-green triangulations. In principle, the criterion is the same, namely that the difference between neighboring subdivision levels should never be greater than 1. However, these rules will be discussed in detail in the next section.

3 Geometry of Subdivision Quarks

In the previous section we have described how each subdivision quark adds one vertex to the triangle mesh, concerning the *topology* of the operation. Now we want to discuss how to determine the *geometry*, in other words the position of this new vertex. In addition, in the context of a selectively refinable mesh, a new vertex is introduced for the split vertex, too. We refer to this new vertex as the *even* child of the split vertex, whereas the other newly

Figure 12: After a split in the Butterfly mask of an odd vertex $\triangle$ has been performed this mask is no longer given by the three neighboring triangles above and the three ones below the split edge (cf. Fig. 2).

introduced vertex is called the *odd* one. Of course we also have to assign a new position to the even child in this setting. But for the sake of simplicity we will focus on reproducing Butterfly subdivision [2], an interpolating subdivision scheme where the split vertex does not change its position. In this way the positions of even vertices are always the same as the positions of their parents. But note that in general we are also able to reproduce an approximating subdivision scheme like the one by Loop [8].

Using Butterfly subdivision, the position of the odd child is determined by the mask in Fig. 2. However, there is a problem in accessing the vertices of the subdivision mask due to our central requirement of being able to apply subdivision splits in an arbitrary order and in an incremental way. The region that is to be adaptively subdivided should not be known a priori. This means that it should be allowed to begin subdivision in one region and proceed somewhere else. Also when two subdivided regions join the result should be consistent with the usual Butterfly subdivision. In this way our subdivision quarks are as independent of each other as possible. The problem of accessing the vertices of the subdivision mask under these circumstances is illustrated in Fig. 12.

After a vertex split has been performed that affected a triangle inside the Butterfly mask, this mask is no longer given by the three neighboring triangles above and the three ones below the split edge (cf. Fig. 2). If we want to access the vertices of the original Butterfly mask we have to analyze the current subdivision configuration. In order to explain how to perform such an analysis we take a look at the C++ data structure of our selectively

```
struct Vertex {
    ListNode<Vertex>  active;        // list stringing active vertices
    Point             point;
    Vertex*           parent;        // 0 if vertex is in M0
    Vertex*           child[2];      // child[0] = 'even' vertex; child[1] = 'odd' vertex
    Face*             fl;            // fr = fl->faces.next
    Face*             fr[2];         // define split operation
    int               splitOpIndex;  // =subdivision level (<0) for subdivision vertices
    };


struct Face {
    ListNode<Face>  faces;           // list stringing all faces
    ListNode<Face>  active;          // list stinging all active faces
    Vertex*         vertices[3];     // ordered counter-clockwise
    Face*           neighbors[3];    // neighbors[i] across from vertices[i]
   };
template<struct T> struct ListNode {
    T* next;
    T* prev;
    };
struct SRPM {
    ListNode<Vertex> base_vertices;  // head of list
    ListNode<Face>   faces;          // head of list
    ListNode<Vertex> active_vertices; // head of list
    ListNode<Face>   active_faces;   // head of list
    };
```

Figure 13: The principal C++ data structure of our selectively refinable progressive mesh with infinite refinement.

refinable progressive mesh with infinite refinement (cf. Fig. 13).

The data structure of a vertex includes an attribute, the *splitOpIndex* member, which we will utilize for our purposes. For non-subdivision vertices it contains the index of the split operation that splits this vertex. If there is no split operation defined for a certain vertex we indicate this fact by setting its *splitOpIndex* member to -3. Also for vertices created by subdivision the *splitOpIndex* member is negative and codes how often a vertex was split during the current 1-to-4 subdivision step.

In Fig. 14 we illustrate how the *splitOpIndex* member is set during subdivision. The even child (child[0]) inherits its parent's *splitOpIndex*, decremented by one, whereas the *splitOpIndex* of the odd child (child[1]) is initialized by -1, marking it as a vertex that is forbidden to be further split for the time being. We also mark even vertices whose neighborhood has been entirely subdivided in a special way by setting their *splitOpIndex* member to -2. If we want to split such vertices we have to proceed to the next 1-to-4 subdivision level. This successive refinement is discussed later in this section. Note that since in the next 1-to-4 subdivision level also the odd children belong per definition to the original mesh, being an *odd* vertex in the usual subdivision sense is not determined by being child[1] of its parent but by having a *splitOpIndex* member of -1 instead.

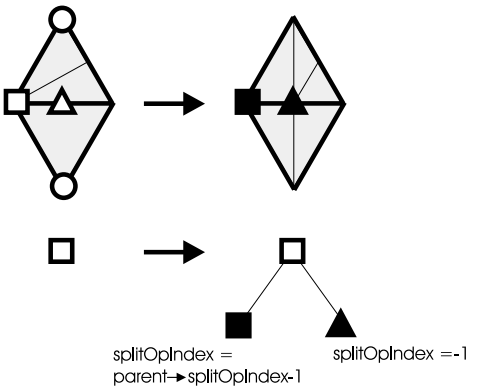

Figure 14: When a split operation is performed two new vertices, ■ and ▲, are introduced in the vertex forest as the children of the split vertex □. The even child (child[0]) inherits its parent's *splitOpIndex*, decremented by one, whereas the *splitOpIndex* of the odd child (child[1]) is initialized by -1. As long as the *splitOpIndex* member of a vertex is -1 the vertex is not allowed to be split.

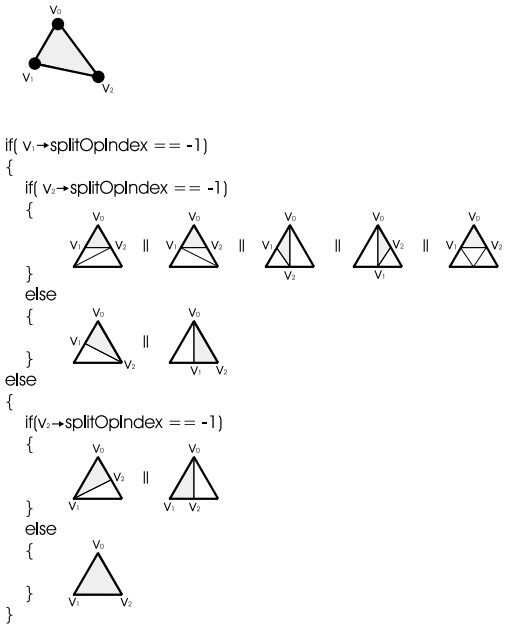

Figure 15: Given an arbitrary triangle in which v_0 is to be split we can restrict the possible subdivision configurations with the help of the *splitOpIndex* member of the vertices v_1 and v_2.

Analysis of the local subdivision configuration In Fig. 15 we demonstrate how the *splitOpIndex* member of a vertex helps analyzing the possible subdivision configurations when an arbitrary face f with a vertex v_0 is given. Note that v_0 is either a vertex of the original mesh or an even child. Dependent on whether the other two vertices in the face are odd children or not, i.e. whether their *splitOpIndex* member is -1 or not, we know whether the coarse triangle has been split twice, once, or never. We can further determine the exact subdivision configuration if we consider the face parameters of the parents of the odd child(ren). Fig. 16 demonstrates the basic principle.

Since each possible split operation requires the given face f to be a certain face of the parent's face members that encode its split operation, it suffices to compare f with these face parameters to exactly determine the subdivision configuration. Knowing the subdivision configuration it is then straightforward to get the vertices of the Butterfly mask, since we exactly know in which faces they lie and which faces are opposite to these vertices.

The analysis of the subdivision configuration described so far does not necessarily access the vertices in the vertex forest that belong to the original subdivision mask but only their even children in the active vertex front. However, since we use an interpolating subdivision scheme, the positions of the even children coincide with the positions of the mask vertices. In an approximating subdivision scheme we would have to access the original mask vertices. This can easily be done by using the information stored in the *splitOpIndex* member.

For the analysis of the subdivision configurations so far it was important that the odd children were not split again. As can be seen in Fig. 15 we rely on identifying odd children by a *splitOpIndex* member of -1 which restricts the number of possible subdivision configurations. This has to be taken into account for successive refinement when proceeding to the next 1-to-4 subdivision level.

Successive refinement Let us consider the situation in Fig. 17 where the odd vertex □ in (a) shall be split. At first the region around this vertex is subdivided by forced splits of even vertices in the neighborhood (b). This step is quite similar to usual red-green triangulation and already guarantees the dif-

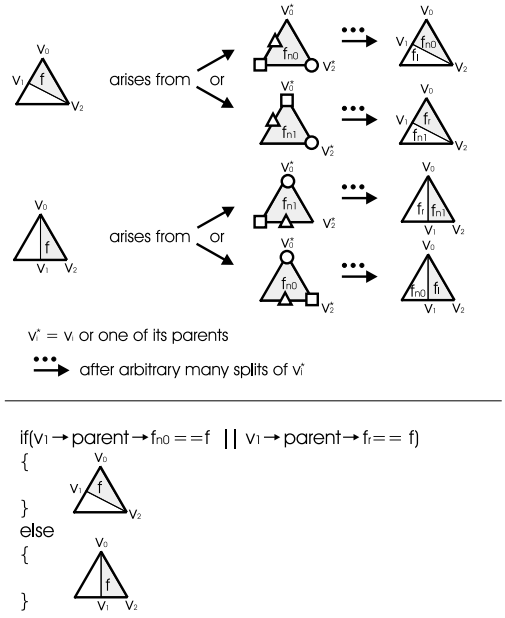

Figure 16: With the help of the parent's face parameters it is straightforward to analyze the exact subdivision configuration.

ference between neighboring subdivision levels to be less or equal to 1. In addition we have to ensure that the subdivision mask of any split operation that might be carried out later on the coarser level does not intersect with the dark grey area in (b) which will be subdivided when proceeding to the next subdivision level. Otherwise the method of analyzing the subdivision configuration that we presented in the last paragraph cannot be applied. Therefore we need to split all even vertices that are neighbors of the black dots in (b), giving the result shown in (c). After all these splits have been forced, the actual split of $\square$ can be performed without any problems. This is indicated by its *splitOpIndex* member which we decrement from -1 ("forbidden") to -3 ("ready to split"). Note that the modified area is much larger than in the case of usual adaptive subdivision (cf. Fig. 3). However, in the context of a selectively refinable progressive mesh this configuration corresponds to an active vertex front that runs through the leaves of the vertex forest, but the active vertex front can also be moved such that it represents the less complex triangulation shown in (d).

Similar rules have to be observed if we want to split an even vertex once its entire neighborhood has been quadrisected (cf. Fig. 18). We have to ensure that no future split operation needs access to the vertices in the dark grey area in (b). Again, this can be guaranteed by forcing splits of even vertices that introduce the vertices represented by black dots in (b). We further indicate that the desired vertex is now allowed to be split by decrementing its *splitOpIndex* member from -2 ("forbidden") to -3. As illustrated in (c) the modified region is even larger than in the case of an odd vertex split. However, after moving the vertex front (d) the same result as in Fig. 17 (d) is obtained.

4 Conclusion

In this paper we have discussed how to express triangle mesh subdivision in terms of the simplest mesh modification, namely the vertex split. This enables us to combine the concept of subdivision with the framework of selectively refined progressive meshes, thus making subdivided meshes applicable to view-dependent level-of-detail rendering. We have explained in detail how the topology as well as the geometry of the adaptive Butterfly subdivision scheme can be reproduced by a sequence of adequate vertex splits. In future work we are going to extend our investigations to the reproduction of other subdivision schemes like Loop and $\sqrt{3}$ subdivision.

Acknowledgment

This work has been sponsored by the SFB 603 "Model Based Analysis and Visualization of Complex Scenes and Sensor Data" of the DFG (German Research Foundation).

References

[1] R. E. Bank, A. H. Sherman, A. Weiser. Refinement Algorithms and Data Structures for Regular Local Mesh Refinement. In *Scientific Computing*, R. Stepleman (ed.), pp. 3–17, IMACS/North Holland, Amsterdam, 1983.

[2] N. Dyn, D. Levin, J. A. Gregory. A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control. *ACM Transactions on Graphics*, vol. 9, 1990, pp. 160–169.

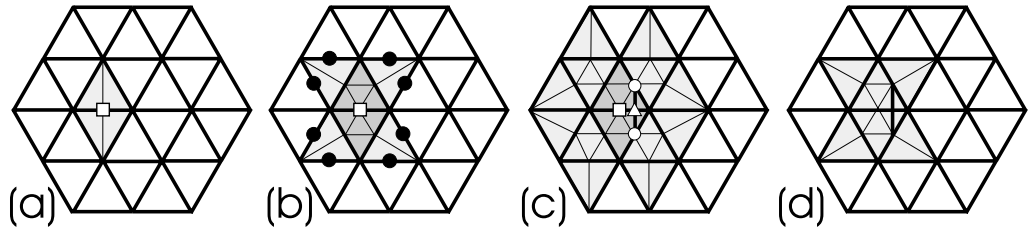

Figure 17: Successive refinement for odd vertices in the quark scheme.

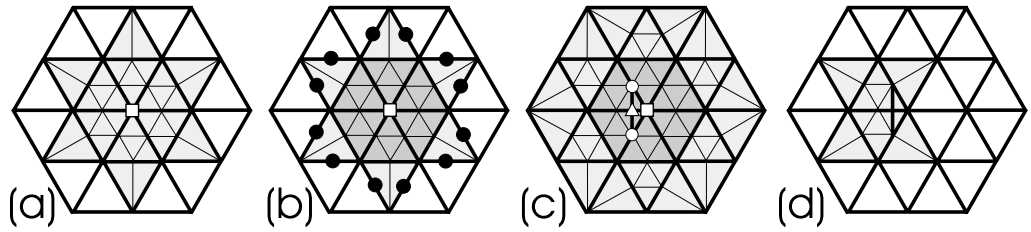

Figure 18: Successive refinement for even vertices in the quark scheme.

[3] H. Hoppe. Progressive Meshes. *SIGGRAPH '96 Proceedings*, pp. 99–108, 1996.

[4] H. Hoppe. View-Dependent Refinement of Progressive Meshes. *SIGGRAPH '97 Proceedings*, pp. 189–198, 1997.

[5] K. Huang. Quarks, Leptons & Gauge Fields. World Scientific, 1992.

[6] L. Kobbelt. $\sqrt{3}$ Subdivision. *SIGGRAPH '00 Proceedings*, pp. 103–112, 2000.

[7] U. Labsik, G. Greiner. Interpolatory $\sqrt{3}$ Subdivision. *EUROGRAPHICS 2000 Proceedings*, pp. 131–138, 2000.

[8] C. Loop. Smooth Subdivision Surfaces Based on Triangles. Master's Thesis, University of Utah, Department of Mathematics, 1987.

[9] T. Muta. Foundations of Quantum Chromodynamics. World Scientific, 1987.

[10] I. A. D'Souza, C. S. Kalman. Preons: Models of Leptons, Quarks and Gauge Bosons as Composite Objects. World Scientific, 1992.

[11] M. Vasilescu, D. Terzopoulus. Adaptive Meshes and Shells: Irregular Triangulation, Discontinuities, and Hierarchical Subdivision. In *Proceedings of Computer Vision and Pattern Recognition Conference*, pp. 829–832, 1992.

[12] D. Zorin, P. Schröder, W. Sweldens. Interpolating Subdivision for Meshes with Arbitrary Topology. *SIGGRAPH '96 Proceedings*, pp. 189-192, 1996.

[13] D. Zorin, P. Schröder, W. Sweldens. Interactive Multiresolution Mesh Editing. *SIGGRAPH '97 Proceedings*, pp. 259–268, 1997.