ACM: Atlas of Connectivity Maps for Semiregular Models

Ali Mahdavi Amiri* Faramarz Samavati†

University of Calgary

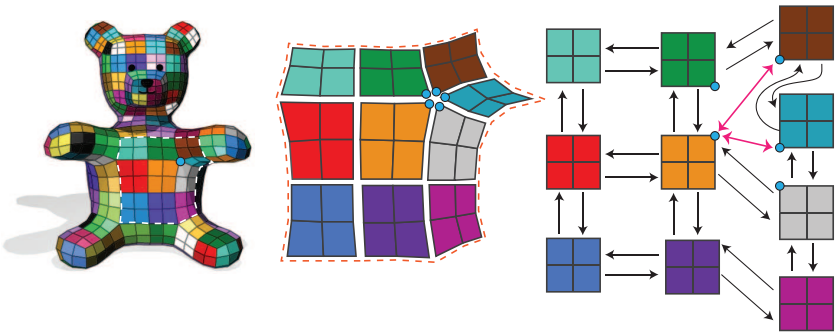

Figure 1: The atlas of connectivity maps for the Teddy's belly.

Abstract

Semiregular models are an important subset of models in computer graphics. They are typically obtained by applying repetitive regular refinements on an initial arbitrary model. As a result, their connectivity strongly resembles regularity due to these refinement operations. Although data structures exist for regular or irregular models, a data structure designed to take advantage of this semiregularity is desirable. In this paper, we introduce such a data structure called atlas of connectivity maps for semiregular models resulting from arbitrary refinements. This atlas maps the connectivity information of vertices and faces on separate 2D domains called connectivity maps. The connectivity information between adjacent connectivity maps is determined by a linear transformation between their 2D domains. We also demonstrate the effectiveness of our data structure on subdivision and multiresolution applications.

Index Terms: I.3.6 [COMPUTER GRAPHICS]: Methodology and Techniques—Graphics data structures and data types; I.3.5 [COMPUTER GRAPHICS]: Computational Geometry and Object Modeling—Hierarchy and geometric transformations

1 Introduction

Semiregular models are very important in computer graphics [4]. They are typically obtained by applying repetitive regular refinement on an initial arbitrary model. After applying regular refinements, most of the vertices are regular and the number of extraordinary vertices does not increase through the resolutions. In such models, the geometry of vertices might be modified by an averaging step, projection, or any type of vertex modification. However, there remains a strong notion of regular connectivity, featuring hierarchical relationships among the vertices at various resolutions.

Due to these important characteristics of semiregular models, they have appeared in several applications in computer graphics. Subdivision models are semiregular models created by a specific type of refinement (which changes the connectivity) and smoothing masks (which change the geometry) applied to an arbitrary initial model. Multiresolution models may also be classified as semiregular, if semiregularity (i.e., subdivision connectivity) is preserved when obtaining the low resolution model.

To take advantage of the characteristics of semiregular models (especially models with a large number of vertices and faces), a general and well designed data structure for this type of models is needed. Such a data structure must support adjacency and hierarchical queries in an efficient manner. Current data structures are mostly designed for either an irregular connectivity [25] or a completely regular model [1].

In this paper, we introduce a data structure that maps the connectivity of semiregular models onto a set of quadrilateral 2D domains that captures the connectivity of regular patches in semiregular models obtained from an arbitrary refinement. The connections between vertices and faces are captured by these 2D domains and their interconnections. We call this structure an *atlas of connectivity maps* (ACM) as illustrated in Fig 1. To map the connectivity of semiregular models, we use the connectivity of the initial mesh to form an atlas of connectivity maps (2D domains) in which their connections are captured. A coordinate system is assigned to each connectivity map such that each vertex has integer coordinates. These integer coordinates are used to index faces and vertices. Having such a coordinate system, the neighbors of a vertex can be determined using simple *neighborhood vectors* in constant time. To establish the hierarchical relationships of faces between resolutions, we use a transformation (rotation, translation, and scaling) between coordinate systems of two resolutions. We use a 2D array for recording the 3D locations of vertices associated with each connectivity map. The coordinate of each vertex in the connectivity map is used to obtain its index in this location array.

The proposed ACM provides efficient and general adjacency and hierarchical operations for semiregular models resulting from various types of refinements. We categorize regular refinements for quad meshes, and for each category, we propose methods to handle adjacency and hierarchical queries using our data structure. We then discuss how to support triangle meshes. The geometry of the semiregular models in our method can be provided through different sources. We discuss the possibility of supporting subdivision and multiresolution (reverse subdivision) as an application of re-

*e-mail:amahdavi@ucalgary.ca

†e-mail:samavati@ucalgary.ca

finement methods. We then compare the performance of ACM with two edge-based data structures supporting subdivision and multiresolution.

2 Related Work

Data structures for semiregular models can be found primarily in literature related to subdivision and multiresolution. We present work related to our proposed method as divided into two categories: subdivision and multiresolution.

Subdivision: Subdivision is a well-studied subject in computer graphics. There are many subdivision schemes, such as Loop, Catmull-Clark, Doo-Sabin, $\sqrt{2}$ and $\sqrt{3}$ subdivision [12, 6, 7, 11, 9]. Subdivision is typically a two-step process: one step of refinement followed by an averaging step. The relationship between lattices at different resolutions resulting from different types of refinement has been previously classified in [1, 8]. Our categorization of refinements is similar to their work. However, we have classified subdivision to assist in designing an efficient data structure to address connectivity queries on an arbitrary connectivity model.

The half-edge data structure and its variations are commonly used to model subdivision surfaces [25]. These data structures are designed for general topological objects' adjacency queries. However, the half-edge data structure cannot be directly used for hierarchical access. Furthermore, it does not benefit from the regularity of subdivision and therefore, for objects with a large number of vertices, it becomes inefficient.

An alternative data structure that supports hierarchical operations is the quadtree [20]. Quadtrees are commonly used for hierarchical meshes, particularly for hierarchical editing applications [26]. Although quadtrees are quite effective at supporting hierarchy between resolutions, they need to store many pointers to maintain their nodes' connectivities and hierarchical dependencies. To overcome this inefficiency, indexing methods exist assigning a unique index to every node to discard the tree structure [20]. However, these indexing methods are mostly designed to support hierarchy and ignore the adjacency relationships. Moreover, since quadtrees are designed to support 1-to-4 refinement, they cannot be directly used to support other refinements.

Patch-based refinement methods rely on data structures that are specifically designed for subdivision methods [5, 21, 22, 13]. In these methods, meshes are divided into some patches and subdivision is separately applied to each patch. Each patch is stored in an array and connectivity between the patches' boundaries is handled using repetitive points at the boundary edges or a first resolution edge based data structure. These methods are mostly designed for a specific type of refinement or shapes [5, 13]. Some of these data structures use spiral 1D indexing for vertices [21, 22]. Spiral indexing complicates neighborhood access, specially for non-immediate neighbors that are essential for applications like multiresolution. We instead use simple 2D domains to maintain connectivity information for extending patch-based methods to support all types of refinements.

Multiresolution: While subdivision generates high resolution objects, multiresolution provides a means to transition from high to low resolution and vice versa [23]. Among multiresolution frameworks, some maintain the semiregularity of objects. This can be achieved by reversing the process of subdivision (i.e. reverse subdivision process) [16, 17, 2] or by considering a property of the coarse vertices like smoothness via the Laplacian [26]. Since both the Laplacian and reverse subdivision use local operators to coarsely sample the fine model, our proposed method can handle these operations.

Olsen et al. [16, 17] use the concept of even/odd vertices to distinguish between details and coarse vertices and provide a compact multiresolution framework. To provide a data structure for this idea, they use an edge based data structure to handle connectivity queries and a hashing method to map vertices to details or coarse vertices [15]. To show that our ACM can efficiently support multiresolution frameworks, we describe how to support the compact multiresolution proposed in [16, 17] and compare the speed of our data structure with [15].

To adapt the half-edge to multiresolution frameworks, Kraemer et al. [10] modify this data structure by defining sequences of half-edges. Using this *multiresolution half-edge* structure, it is possible to support primal and dual schemes. However, this data structure needs a large amount of memory for high resolution models due to the storing of all edges. It also needs extensive amount of time to maintain its structure after the refinement. By comparison, in our method, which benefits from the regularity of subdivision models, a significant amount of memory and time is saved.

3 Atlas of Connectivity Maps and Its Refinement

Semiregular models are obtained by refining an arbitrary coarse mesh. As a result, irregularity in semiregular models is limited to that introduced by the coarse model. To capture the connectivity information of the entire mesh, we position the initial connectivity in such a way that all connectivity queries can be executed using simple algebraic operations. This initialization is performed in such a way that the geometry of vertices is also easily accessible. Afterwards, the effect of all types of refinement on the initial mesh is analyzed. This leads to a refinement categorization that we describe in Section 4. In this section, we introduce the notations that we use in the entire paper. We also describe how to set up the atlas of connectivity maps and maintain it after applying refinements.

3.1 Connectivity Maps

Connectivity maps are 2D domains that locally capture the connectivity of a general mesh. We first describe how to establish connectivity maps for a quad mesh and then generalize it for triangles. Consider a quad mesh with M quads q_i, $i = 0,...,M-1$ (Fig 2(a)). Each quad q_i has four vertices that can be labeled by an arbitrary cyclic ordering p^i_j, $j = 0,...,3$ (Fig 2(b)). In this ordering, p^i_j and p^i_{j+1} are neighbors (p^i_3 is adjacent to p^i_0). We map q_i to a unit 2D square Q_i by mapping p^i_0 to $(0,0)$, p^i_1 to $(1,0)$, p^i_2 to $(1,1)$, and p^i_3 to $(0,1)$. Having these 2D coordinates for each Q_i, we establish a coordinate system for Q_i as illustrated in Fig 2(c). We refer to this as connectivity map (CM) coordinate system. Q_i is called the connectivity map of q_i and is denoted by $Q_i = CM(q_i)$. Any point in Q_i has a coordinate based on the CM coordinate system of Q_i. We call these *CM coordinates*. Q_i and its CM coordinates are used to handle connectivity queries of vertices throughout the refinement process.

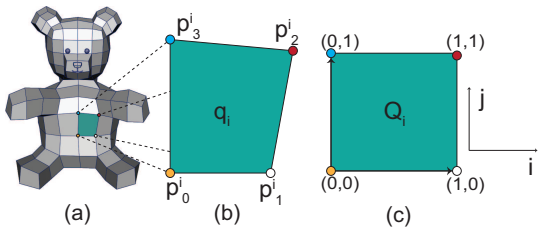

Figure 2: (a) A quad mesh with M quads. (b) q_i is a quad in 3D. (c) q_i is mapped to its connectivity map Q_i and a coordinate system is defined for Q_i.

To access to the connectivity information of the entire mesh, information on the connections between connectivity maps is needed. Since meshes are 2-manifold, q_i and its neighbors can be unfolded on a 2D neighborhood pattern (see Fig 3). The neighbors of q_i are denoted by $q_{i\alpha}$, $\alpha = 0,...,3$. In this neighborhood pattern, q_i can

be mapped to its connectivity map Q_i and $q_{i\alpha}$ corresponds to a unit square adjacent to Q_i. In fact, $q_{i\alpha}$ is mapped to a unit square $S_{i\alpha}$ adjacent to the edge of Q_i connecting p^i_α to $p^i_{\alpha+1}$. Note that $S_{i\alpha}$ is not exactly the connectivity map of $q_{i\alpha}$ ($Q_{i\alpha}$) since it is in the CM coordinate system of Q_i. However, a simple coordinate transformation $T_{i\alpha}$ can be used to map CM coordinates in $S_{i\alpha}$ to CM coordinates in $Q_{i\alpha}$.

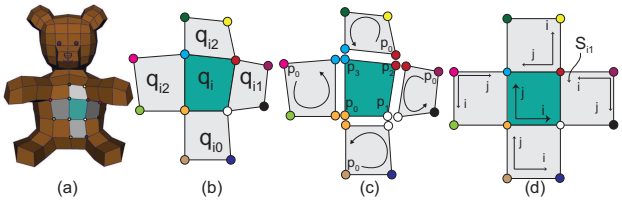

Figure 3: (a) q_i and its neighbors ($q_{i\alpha}$) on a 3D mesh are highlighted. (b) Zoomed q_i and $q_{i\alpha}$. (c) Ordering and orientation of points in q_i and $q_{i\alpha}$. (d) $S_{i\alpha}$ and CM coordinates systems of Q_i and $Q_{i\alpha}$ in the neighborhood pattern.

The CM coordinate systems of $Q_{i\alpha}$ can be assigned to this neighborhood pattern. We can take $p^{i\alpha}_0$ as the origin and the edges connecting $p^{i\alpha}_0$ to $p^{i\alpha}_1$ and $p^{i\alpha}_0$ to $p^{i\alpha}_3$ to be the i and j axes of the coordinate system. To determine the coordinate system of $Q_{i\alpha}$ in this pattern, we find the vertex in $Q_{i\alpha}$ equivalent to p^i_α. Knowing the position of one vertex of $Q_{i\alpha}$ is enough to find its coordinate system since the orientation of faces is the same. For example, if p^i_0 is equivalent to p^{i0}_3 (Fig 3(c)), the coordinate system shown in the bottom quad in Fig 3(d) is defined for Q_{i0}. $T_{i\alpha}$ is the transformation mapping CM coordinate system of Q_i to the CM coordinate system of $Q_{i\alpha}$ in the neighborhood pattern. $T_{i\alpha}$ are stored as a part of our data structure.

In later discussions, the neighborhood pattern and transformations are used to handle connectivity queries that need the information of two adjacent connectivity maps. An example of such a query is finding the neighbors of vertices located at the boundary edges of Q_i.

3.2 Refining Connectivity Maps

We explained how to set up the initial connectivity information of a general mesh. This initial connectivity information may be used to handle connectivity queries of the mesh after applying refinements. When a refinement is applied to q_i with a set of vertices V, set V_1 is generated. The CM coordinates of V_1 in Q_i are found by applying refinement on the CM coordinates of V. For instance, Fig 4(b) illustrates the CM coordinates of V and V_1 after applying 1-to-4 refinement. In order to distinguish the CM coordinates before and after refinement, we use a subscript indicating the level of refinement or *resolution*. We also use Q_i as a superscript for $(a,b)_r$ indicating the connectivity map in which $(a,b)_r$ is located. As a result, $(a,b)^{Q_i}_r$ refers to vertex (a,b) in the CM coordinate system of Q_i at resolution r. We may drop Q_i if it is clear which connectivity map $(a,b)_r$ belongs to.

CM coordinates are used as references to the 3D locations of the vertices. In regular refinements (i.e., linear subdivision), the 3D location of vertices can be found by a simple mapping, such as a barycentric mapping from vertices of Q_i to q_i. However, in applications that 3D location of vertices are changed by an additional operation, such as the smoothing masks of subdivision schemes, the 3D locations of each vertex is stored. A natural choice for storing the geometry is to use a 2D location array containing 3D points (x,y,z). To obtain indices referring to the location array from the CM coordinates, it is best if vertices have integer coordinates in Q_i. Using a transformation, a coordinate system aligned with Q_i is introduced for V_1 in which all vertices receive integer coordinates (see Fig 4(c)). The transformation that maps non-integer CM coordinates to integer coordinates is called T_{int}. We call the resulting integer coordinates *CM index*.

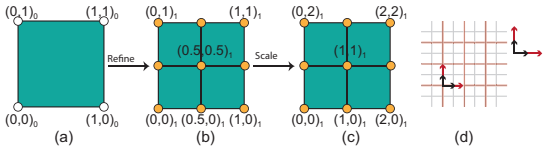

Figure 4: ○ and ● form the set V and V_1 in Q_i. (a) Initial connectivity map Q_i. (b) Applying refinement on the CM coordinates of Q_i. (c) T_i is used to avoid non-integer coordinates. (d) Red and black lattices are the connectivity lattices before and after 1-to-4 refinement respectively.

To identify the properties of refinements, the concept of lattices can be used. When a refinement is applied to a lattice representing the connectivity of vertices (connectivity lattices), a transformation T_{ref} is imposed to this connectivity lattice [1, 8]. We can also use connectivity lattices for each connectivity map resulting from refinements, since after refinement, each connectivity map can be considered as a bounded lattice. T_{ref} maps connectivity lattices between resolutions. For example, in 1-to-4 refinement, T_{ref} is a scaling by $\frac{1}{2}$. As illustrated in Fig 4(d), scaling the red lattice by $\frac{1}{2}$ aligns the red lattice and black lattices representing the connectivity of vertices at two successive resolutions. Refinements introduce different T_{ref}. Therefore, we categorize refinements based on T_{ref} in Section 4 and suggest transformation T_{int} for each category to show how T_{ref} and T_{int} are used to determine the connectivity of vertices.

It is possible to establish a hierarchical relationship among faces and vertices. This relationship can be used in applications such as multiresolution and mesh editing since we can find the index of a face or vertex after applying refinements. To this end, we need to find the index of a face or vertex at the next resolutions and establish an algebraic hierarchical relationship. For example, in 1-to-4 refinement, a vertex with index $(a,b)_r$ has index $(2^n a, 2^n b)_{r+n}$ after n levels of refinement. We can therefore deduce that a face with index $[a,b]_r$ covers faces with index $[c,d]_{r+n}$ at resolution n in which $2^n a \le c < 2^n(a+1)$ and $2^n b \le d < 2^n(b+1)$. In Fig 1, faces covered with the same coarse face are in the same color. We use the same manner of coloring for patches resulting from the same coarse face in the entire paper.

3.3 Neighborhood Vectors

To access the neighbors of a vertex in Q_i, we define a set of *neighborhood vectors*. Neighborhood vectors are added to the CM index of a vertex, from which the CM indices of its neighbors are obtained. The neighborhood vectors for a regular point in Q_i are simply $(1,0)$, $(-1,0)$, $(0,1)$ and $(0,-1)$.

To handle connectivity queries after refinement, it is necessary to consider the effect of refinement on the neighborhood vectors. Since refinements change the connectivity of vertices, neighborhood vectors are also affected by the imposed transformation T_{ref}. We also manipulate coordinate systems with T_{int} to get integer coordinates. As a result, neighborhood vectors are transformed by the combination of two transformations T_{ref} and T_{int} ($T_{ref} o T_{int}$). For example, in 1-to-4 refinement, $T_{ref} = scale(\frac{1}{2})$ and $T_{int} = scale(2)$. As a result, $T_{ref} o T_{int} = I$, so neighborhood vectors are identical after 1-to-4 refinement (Fig 5).

If vertex $(a,b)^{Q_i}_r$ is an internal vertex in Q_i (i.e., the vertex and its neighbors are in Q_i), they are in the same coordinate system and adding neighborhood vectors to $(a,b)_r$ results CM index of its

neighbors. However, if the neighbor of vertex $(a,b)_r^{Q_i}$ falls out of Q_i in $S_{i\alpha}$, its CM index in $Q_{i\alpha}$ must be determined. Since the coordinate systems of Q_i and $Q_{i\alpha}$ are different, we use $(T_{i\alpha})$ to find the CM index of the neighbor of $(a,b)_r^{Q_i}$ which falls in $Q_{i\alpha}$ in the neighborhood pattern. To this end, we add the neighborhood vector to $(a,b)_r$ and get $(c,d)_r$ and then apply $T_{i\alpha}$ on $(c,d)_r$ to obtain the CM index of this vertex in $Q_{i\alpha}$. Fig 5 illustrates this situation .

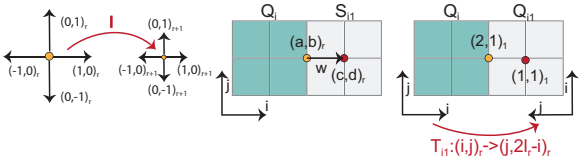

Figure 5: Left: Identity transformation of neighborhood vectors at two successive resolutions. Middle: Transformation T_{i1} maps coordinate system of Q_i to Q_{i1}'s. l_r is the number of vertices along one i or j axis at resolution r. Vector $w=(1,0)$ is added to $(a,b)_r=(2,1)_1$. $(c,d)_r=(3,1)_1$ is in S_{i1}. Right: $(c,d)_r$ is transformed by T_{i1} and the result is $(1,1)_1$ in Q_{i1}.

3.4 Forming Faces

In our method, the connectivity of faces is implicit and determined using vertex connectivities. Face $[a,b]_r^{Q_i}$ is associated with vertex $(a,b)_r^{Q_i}$. Having $(a,b)_r^{Q_i}$, all vertices making up $[a,b]_r^{Q_i}$ are obtained by adding some vectors. For example, in 1-to-4 refinement, $(a,b)_r^{Q_i}$ is the left-bottom corner of $[a,b]_r^{Q_i}$ in CM coordinate system of Q_i. As a result, other vertices forming $[a,b]_r^{Q_i}$ are found by adding vectors $(1,0)_r$, $(1,1)_r$, and $(0,1)_r$ to $(a,b)_r^{Q_i}$. For each category of refinement, we discuss the vertex associated with face $[a,b]_r^{Q_i}$ and the vectors that are necessary to form faces.

3.5 Data Structure Elements

So far, we have discussed the required elements to handle connectivity queries. Here, we explicitly discuss how to store these elements for our proposed data structure. For storing the connectivity information of a mesh, we use an array of connectivity maps called *Q_List*. Each entry of *Q_List* (*Q_List*$[i]$) corresponds to a Q_i. A global integer called *res* is also stored for the entire mesh that refers to the resolution or level of refinement of the mesh. The resolution of the mesh helps to determine the range of vertex indices in Q_i. For example, in 1-to-4 refinement, the CM indices $(a,b)_r$ are in the range $0 \le a,b \le 2^r$.

Each connectivity map Q_i has structures to store the 3D locations of vertices and its neighborhood pattern (see Fig 6(a)). Q_i has a 2D location array of 3D points (x,y,z) called *vertices* storing locations of its vertices. To access neighbors, each Q_i has also an integer array called *neighbors* that keeps the indices of the $Q_{i\alpha}$ in *Q_List*. For each $Q_{i\alpha}$, a $T_{i\alpha}$ is also stored in an array called $transformation$. To have an easier representation for transformations, we encode possible $T_{i\alpha}$ by integers(see Fig 6(b)).

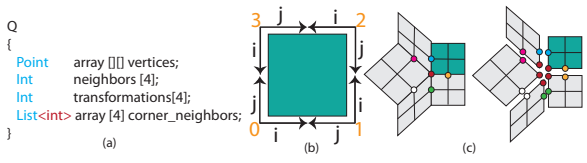

Figure 6: (a) Elements that are used for each connectivity map. (b) Encoding of transformations. (c) Neighbors of p_0^i. Duplicate vertices are illustrated in the same color.

The connectivity of corners must also be determined. Irregular vertices in semiregular meshes are only found at the corners of the connectivity maps. These vertices are shared by multiple connectivity maps. Therefore, to access the neighbors of a corner, we find its neighbors in each connectivity map and discard duplicates that are shared at the boundary edges of (Fig 6(b)).

Formally, to access the neighbors of corners p_α^i with valence ω, we store connectivity maps Q_{sv} ($v=1,...,\omega-1$) having a vertex p_n^{sv}, $n=0,...,3$, equivalent to p_α^i. The neighbors of each p_n^{sv} (for any v) are combined with the neighbors of p_α^i and duplicates are discarded. Figure 6(c) illustrates this scenario for p_0^i. The indices of Q_{sv} in *Q_List* are stored in *corner_neighbors*, which is a 4-element array of lists with length of $\omega-1$.

Storing *corner_neighbors* is primarily an implementation decision since the neighbors of a corner can be found using $Q_{i\alpha}$ and their adjacent connectivity maps. However, having direct access to the neighbors of the corners speeds up the process. Furthermore, since this information is constant throughout the resolutions, it is not expensive to store additional information for the initial mesh to simplify operations. Note that our suggested data structure set up is not unique and other possible initial set ups might work efficiently. As a result, one may change the elements of the data structure based on their own application.

4 Refinement Categorization

As discussed in Section 3.2, refinements impose transformations (T_{ref}) on the coordinate systems of the connectivity maps. T_{ref} may include a rotation, scaling, and translation. Based on the type of T_{ref}, we categorize refinements and analyze their effects on neighborhood vectors, *CM* indices, boundaries of the connectivity maps, and the form of faces.

4.1 Scaling, No rotation, No Translation

Refinements that do not induce any rotation or translation in the next resolution are very common and appear in popular subdivision schemes such as Catmull-Clark [6]. For instance, 1-to-4 refinement creates aligned lattices at two successive resolutions scaled by $\frac{1}{2}$ (see Fig 4 for the refinement and Fig 7(b) for subdivision). Therefore, as discussed earlier, $T_{ref}=scale(\frac{1}{2})$. To avoid non-integer indices for vertices, the CM coordinates are transformed by $T_{int}=scale(2)$. $T_{ref}oT_{int}=I$, therefore neighborhood vectors are not changed through the resolutions (see Fig 5). Based on T_{int}, the length of the 2D location array (*vertices*) is determined. In a 1-to-4 refinement, the location array is of size $(2^r+1)\times(2^r+1)$ at resolution r.

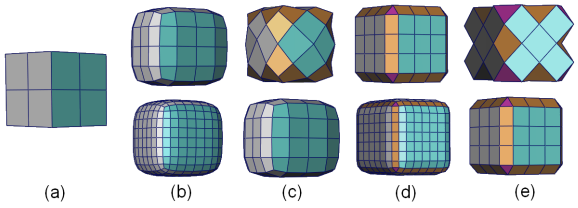

Figure 7: (a) Initial mesh. The result of different subdivision schemes are illustrated in (b) Catmull-Clark (c) $\sqrt{2}$ (d) Doo-Sabin (e) Simplest. Orange/pink faces are shared faces at boundaries/corners.

Transformation T_{int} is generalized by $T_{int}^r=scale(2^r)$ for resolution r. These relations are also generalized to any 1-to-m^2 refinement via scaling by m^r. Another instance of 1-to-m^2 refinement is ternary subdivision in which $m=3$ [14]. Note that since the coordinate systems of two resolutions are aligned, T_{ref} is always a scaling by $\frac{1}{m}$ and the location array is $(m^r+1)\times(m^r+1)$ at resolution r.

In 1-to-m^2 refinements, vertex $(a,b)_r$ has index $(m^n a, m^n b)_{r+n}$ after n level of refinements. A face with index $[a,b]_r$ also covers faces with index $[c,d]_{r+n}$ at resolution n in which $m^n a \leq c < m^n(a+1)$ and $m^n b \leq d < m^n(b+1)$.

4.2 Scaling, Rotation, No Translation

Some refinements impose a rotation in connectivity lattices at two successive resolutions. 1-to-2 refinement used by $\sqrt{2}$ subdivision, is one instance of such a refinement (Fig 7(c)) [11]. In 1-to-2 refinement, a vertex is inserted at the midpoint of each quad. Afterwards, the edges of the quads are removed and the midpoints are connected to the vertices of the quads (Fig 8).

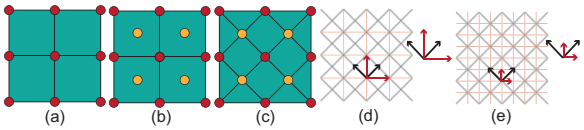

Figure 8: 1-to-2 refinement: (a) Four adjacent quads (b) Inserting Face vertices. (c) Changing the connectivity (d),(e) Lattices at two successive resolutions and their coordinate systems. The red and black lattices are, respectively, the connectivity lattice at even and odd resolutions.

As apparent in Fig 8(d), a rotation by $\frac{\pi}{4}$ and scaling by $\frac{1}{\sqrt{2}}$ aligns lattices at two successive resolutions, therefore we can deduce that $T_{ref} = scale(\frac{1}{\sqrt{2}})rot(\frac{\pi}{4})$. The red lattice in Fig 8(d) is the connectivity lattice of even resolutions. T_{ref} maps connectivity lattices of even resolutions to odd ones. However, two applications of 1-to-2 refinement cancels out the rotation and aligns the connectivity maps with the even resolutions scaled by $\frac{1}{2}$ (Fig 8(e)). Therefore, we can deduce that $\acute{T}_{ref} = scale(\frac{1}{\sqrt{2}})rot(\frac{-\pi}{4})$ maps the connectivity lattices of odd resolutions to even ones.

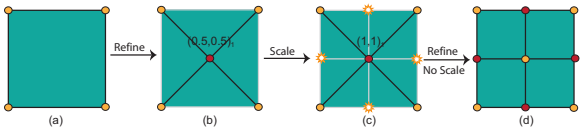

Figure 9: (a) Initial connectivity map. (b) 1-to-2 refinement makes non-integer coordinates. (c) Scaling to avoid non-integer coordinates. (d) Connectivity map at resolution two. No transformation is needed. ● and ●: vertex and face vertices, ✲: empty CM indices.

Applying 1-to-2 refinement on Q_i results in vertices with non-integer CM coordinates at the midpoints of quads in Q_i at odd resolutions (Fig 9). To obtain integer CM indices, we scale the CM coordinates by two ($T_{int} = scale(2)$). This results in some empty CM indices with no associated vertex at odd resolutions (Fig 9(c)). These integer CM indices are used for the new vertices at the next resolutions, therefore no transformation is needed for obtaining the CM indices from even to odd resolutions; $\acute{T}_{int} = I$ (see Fig 9(d)).

Transformations that map the neighborhood vectors at resolution r are again found by composing T_{ref} and T_{int}. The transformation mapping neighborhood vectors of even resolutions to odd resolutions is called $T_{e\to o}$ and is found by composing T_{ref} and T_{int}. As a result, $T_{e\to o} = T_{ref} o T_{int} = scale(\sqrt{2})rot(\frac{\pi}{4})$. Similarly, $T_{o\to e} = \acute{T}_{ref} o \acute{T}_{int} = scale(\frac{1}{\sqrt{2}})rot(\frac{-\pi}{4})$. Therefore, we can deduce that $T_{e\to o} : (i,j) \to (2i-j, i+j)$, $T_{o\to e} : (i,j) \to = \frac{1}{2}(i+j, j-i)$, and $T_{e\to o} o T_{e\to o} = I$. Fig 10(a) illustrates the neighborhood vectors at even and odd resolutions and transformations $T_{e\to o}$ and $T_{o\to e}$.

Faces at even resolutions are associated with a vertex at their bottom-left corner and formed by vectors $(1,0)$, $(1,1)$, and $(0,1)$

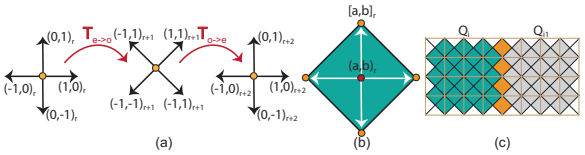

Figure 10: (a) Transformations of neighborhood vectors. (b) Essential vectors for forming a face. (c) Orange shared faces between two neighborhood vectors.

(see Section 4.1). However, faces at odd resolutions are associated with a vertex located at their midpoint and formed by vectors illustrated in Fig 10(b). Note that some faces are shared by two adjacent connectivity maps Q_i and $Q_{i\alpha}$ (Fig 10(c)). To avoid redundancy, these faces are considered to belong to Q_i if $i < i\alpha$. Shared faces are highlighted in orange in Figures 10(c) and 7(c). Since two steps of this refinement is the same as a 1-to-4 refinement, hierarchical relationships are similar to ones discussed in Section 4.2.

4.3 Scaling, No rotation, Translation

A translation might be imposed by a refinement to the connectivity lattices. One example is the 1-to-4 refinement used by the Doo-Sabin subdivision scheme [7] (Fig 7 (d)). For this scheme, $T_{ref} : (i,j) \to \frac{1}{2}(i+\frac{1}{2}, j+\frac{1}{2})$ is the transformation imposed by the refinement (Fig 11 (d)). To get integer CM coordinates, we can use $T_{int} : (i,j) \to 2(i-\frac{1}{2}, j-\frac{1}{2})$. In general, at resolution r, $T^r_{ref} : (i,j) \to \frac{1}{2^r}(i+\frac{1}{2}, j+\frac{1}{2})$ and $T^r_{int} : (i,j) \to 2^r(i-\frac{1}{2}, j-\frac{1}{2})$ and $T_{ref} o T_{int} = I$. Therefore, neighborhood vectors are the same throughout the resolutions. The neighborhood vectors are the same as those for 1-to-4 refinement without translations (Fig 5) and the location array has length $2^r \times 2^r$ $(r > 0)$. Since old vertices are removed and new vertices are inserted in this refinement, hierarchical relationships are established based on faces instead of vertices. A face with index $[a,b]_r$ $(r > 0)$ covers faces with index $[c,d]_{r+n}$ in which $0 \leq c,d \leq 2^n$.

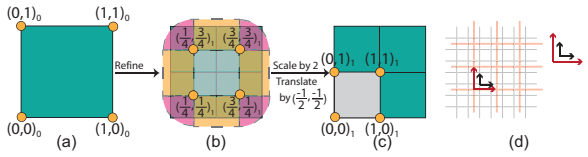

Figure 11: (a) $CM(Q)$ (b) ●: vertices obtained by refinement and lying in $CM(Q)$. Orange faces are shared by two connectivity maps. Pink faces are corner faces. (c) Applied scaling and translation to get integer CM indices. (d) Lattices of two successive resolutions.

For the internal vertices $(a,b)_r$, $0 < a,b < 2^r - 1$, adding neighborhood vectors to the CM indices results in the CM index of the neighbors (Fig 12(a)). However, for vertices whose neighbors fall in an adjacent connectivity map $Q_{i\alpha}$, we need to use the neighborhood pattern (Section 3.1). The connectivity of vertices is manipulated by a translation via T_{int} in Q_i. To access external neighbors, the CM indices are translated back by $(\frac{1}{2}, \frac{1}{2})$, neighborhood vectors are added and then they are translated by $(\frac{-1}{2}, \frac{-1}{2})$ in CM coordinate system of $Q_{i\alpha}$. Fig 12(b) illustrates such a scenario for vertex $(3,2)^{Q_i}_2$ whose neighbors fall in Q_{i1}.

The face indices are the same as those discussed in Section 3.4. Some shared faces again exist between connectivity maps Q_i and $Q_{i\alpha}$ that belong to Q_i if $i < i\alpha$. Corner faces require special treatments. Faces at corner P^i_α with valence ω have ω number of vertices. Each of these vertices is located in Q_{sv} adjacent to P^i_α as discussed in Section 3.5. If P^i_α is equivalent to P^{sv}_0, P^{sv}_1, P^{sv}_2, or P^{sv}_3,

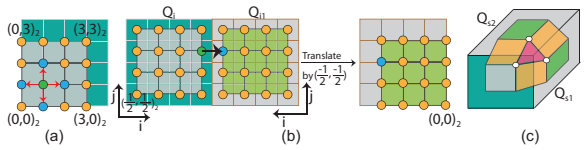

Figure 12: ● is the neighbor of ●. (a) Neighborhood vectors for internal neighbors added to *CM* indices. (b) *CM* indices are translated by $(\frac{1}{2},\frac{1}{2})_r$ in Q_i, neighborhood vectors are added and they are translated by $(\frac{1}{2},\frac{1}{2})_r$ in Q_{i1}. (c) Orange faces are shared by two connectivity maps. The pink face is a face at the corner.

the corner face has a vertex in Q_{sv} with index $(0,0)_r$, $(2^r-1,0)_r$, $(2^r-1,2^r-1)_r$, or $(0,2^r-1)_r$ respectively. Fig 12(c) illustrates an example of shared faces and a corner face.

4.4 Scaling, Rotation, Translation

If a refinement imposes both rotation and translation, we can combine the discussed methods to handle the connectivity queries. One instance of such a refinement is the 1-to-2 refinement used by the simplest subdivision (Fig 7(e))[18].

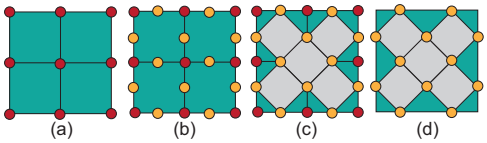

Figure 13: Simplest's refinement: (a) Four adjacent quads. (b) Inserting vertices in the midpoint of edges. (c) Connecting the midpoints. (d) Removing previous vertices and edges.

In this refinement, one vertex is inserted at the midpoint of each edge, midpoints are connected to each other and the previous sets of vertices and edges are removed (see Fig 13). Odd resolutions for this refinement are handled similarly to 1-to-2 refinement (Section 4.2), except a different set of empty CM indices. Even resolutions, by contrast, are dealt with in the exact same way as with Doo-Sabin (Section 4.3). Hierarchical relationships and forming faces are also similar to the case discussed in Section 4.3.

Note that once we know the effect of each refinement on the connectivity map, we can combine different refinements on the connectivity maps. Fig 14 illustrates the effect of various subdivisions on a model. This is an advantage of our work that does not exist in other patch-based methods.

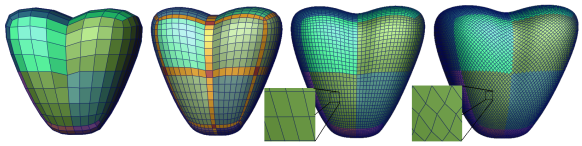

Figure 14: From left to right: initial mesh, Doo-Sabin, Catmull-Clark, and $\sqrt{2}$ subdivision.

5 Connectivity Maps for Triangles

So far, we have discussed how to set up the atlas of connectivity maps for handling connectivity queries of quadrilateral semiregular models. In the following section, we describe how to modify the method to support triangular meshes.

5.1 Triangles

As with quad meshes, for triangle meshes it is desirable to have a quadrilateral domain for the connectivity map, since the 3D locations of vertices are stored in a 2D array. In order to create such a domain, we can pair adjacent triangles to form a quad, creating a single connectivity map. Suppose that we have a set of faces $F=\{f_1,f_2,...,f_M\}$, we can pair f_i with f_j if f_i and f_j are adjacent. Afterwards, both f_i and f_j are removed from F and the process repeats until no adjacent faces exist in F. A complete pairing of triangles is possible and is computable in $O(Mlog^4M)$ where M is the number of triangles [19, 3]. Triangles in F that remain unassigned to a pair (isolated triangles) may each be assigned to a half-empty connectivity map. As a result, having isolated triangles is not fatal (Fig 17). However, for efficiency it is better to reduce the number of isolated triangles by using methods that can make a pure quad mesh from a given triangular one [3, 24].

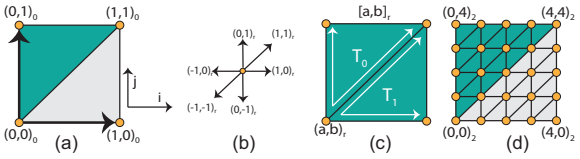

Figure 15: (a) Coordinate system for triangular connectivity maps. (b) Neighborhood vectors for triangles. (c) Form of triangular faces. (d) Applying two steps of 1-to-4 refinement on a triangular connectivity map.

For consistency, the coordinate system of the connectivity maps is restricted so that the diagonal connects $(0,0)_0$ to $(1,1)_0$. To handle connectivity queries, a new set of neighborhood vectors is used (Fig 15.(b)). Since two triangles are assigned to one connectivity map, each $[a,b]_r^{Q_i}$ refers to two faces T_0 and T_1 which can be constructed using vectors illustrated in Fig 15(c).

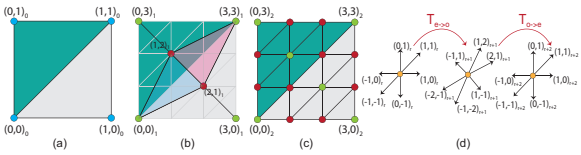

Figure 16: (a),(b) and (c) Connectivity maps at resolutions zero to 2. (● and ●: face and vertex vertices.) (d) Neighborhood vectors at even and odd resolutions and their transformations.

Refinements on triangular connectivity maps are also very similar to the quadrilateral cases. For instance, applying 1-to-4 refinement on a triangular connectivity map leads to the scaling of the connectivity map by two (Fig 15(d)). Another example of refinement for triangular meshes is 1-to-3 refinement used by $\sqrt{3}$ subdivision. In this refinement, the midpoint of each triangle (face-vertex) is inserted and the connectivity is changed, as illustrated in Fig 16. Therefore, connectivity lattices are transformed by a $\frac{\pi}{6}$ rotation and $\sqrt{3}$ scaling. Similar to Section 4.2, to get integer *CM* indices, we scale the CM coordinates by three ($T_{int}=scale(3)$) from even to odd resolutions and no scaling is applied from odd to even resolutions ($\acute{T}_{int}=I$).

Neighborhood vectors for 1-to-3 refinement, are illustrated in Fig 16(d) for even and odd resolutions. The transformations mapping neighborhood vectors are $T_{e\to o}:(i,j)\to(2i-j,i+j)$ and $T_{o\to e}:(i,j)\to\frac{1}{3}(i+j,2j-i)$. Figure 17 illustrates applying $\sqrt{3}$ subdivision on a pawn. To form faces, similar to Section 4.2, faces at odd resolutions are associated with the index of their midpoint and shared faces between Q_i and $Q_{i\alpha}$ are created in Q_i if $i<i\alpha$

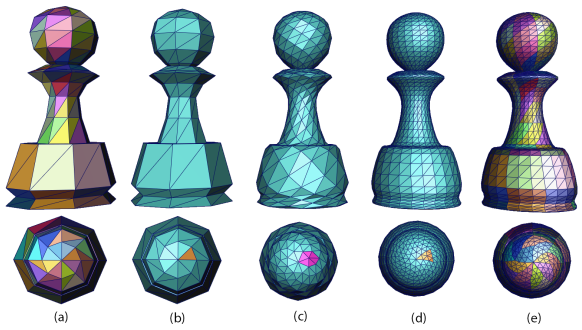

Figure 17: Applying $\sqrt{3}$ subdivision on the pawn model. Bottom: zoomed screen shots of the head of the pawn. The isolated triangle is shown in orange. The pink portion indicates the region formed by subdividing the isolated triangle. Triangle pairs are shown with different colors in (a) and (e).

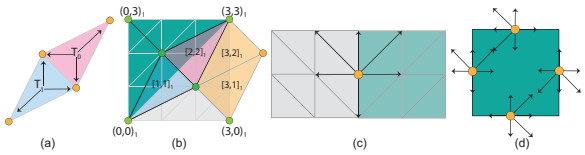

Figure 18: (a) Forming faces at an odd resolution for T_0 and T_1. (b) Orange faces at the boundaries that are shared with a neighbor. (c) Two adjacent connectivity maps with different connectivity patterns. (d) Neighborhood vectors for boundary edges at two connectivity maps with different connectivity patterns.

(see Fig 18(a) and (b)). Note that it is possible that Q_i and $Q_{i\alpha}$ have different connectivity patterns (Fig 18(c)). To handle adjacency queries at the boundaries, different neighborhood vectors illustrated in Fig 18 (d) are used in these cases.

6 Applications and Comparisons

As a first application, we perform fast subdivision on surfaces. We compare the run time needed to reach resolution $i+1$ from resolution i using the half-edge structure with our approach and show that ACM is much faster. Furthermore, we show how connectivity maps are useful for handling adjacency and hierarchical queries in multiresolution frameworks.

6.1 Subdivision

ACM provides simple adjacency operations using neighborhood vectors running in $O(1)$. Due to their simple structure, they are also more efficient at supporting applications (such as subdivision) than other common edge-based data structures. For example, in 1-to-4 refinement the half-edge structure needs to manipulate at least 16 pointers per face. However, in ACM no pointer manipulation is needed and scaling the length of 2D location array is enough to increase the resolution. To quantify the difference between the speed of ACM and half-edge, we compare the run-time (CPU) of the ACM with the half-edge data structure for subdividing the Pawn using $\sqrt{3}$ subdivision in Table 1. It is apparent that the connectivity maps are much faster than edge-based data structures. ACM is very fast for even very large meshes. To subdivide a pawn at resolution nine with 1994544 faces, ACM needs less than a second (0.987). Our implementation was coded in the C#.Net framework on an Intel i7 quad core processor under Windows 7. We also compare the space complexity of the ACM with a simple data structure with low space usage (face-list and vertex-list) in Appendix A.

Table 1: Subdivision time (in seconds) for the Pawn mesh (Fig 17) to reach resolution $i+1$ from resolution i. ACM: Atlas of connectivity maps. HE: Half-edge.

Resolution	#of Faces	ACM	HE
2	912	0.003	0.161
3	2736	0.007	0.231
4	8208	0.015	1.837
5	24624	0.016	9.375

6.2 Multiresolution

While subdivision methods are used to create high resolution objects, reverse subdivision can be used to make low resolution objects. When combining the two into a multiresolution framework, it is possible to define a compact representation in which the storage requirements for the details and coarse vertices together equal the storage requirement of the fine vertices. As noted earlier, Olsen et al. [16] provides a compact multiresolution in which vertices are categorized into even and odd vertices. After reverse subdivision, even vertices are replaced by coarse vertices and details are located at odd vertices. The details of even vertices, therefore, are found using a linear combination of odd details in its neighborhood.

Using our proposed method, it is possible to build efficient data structures for this type of multiresolution based on a variety of subdivision methods. In this section, we describe our proposed data structure for multiresolution frameworks based on Catmull-Clark, Loop, and $\sqrt{3}$ reverse subdivision.

Catmull-Clark: The reverse subdivision filters for Catmull-Clark and Loop subdivision are respectively provided in [17] and [16]. Here, we discuss how to access the neighbors of a coarse vertex and its corresponding details, which are essential operations in the reconstruction process. In fact, the even/odd representation of vertices can be extended to several levels of reverse subdivision by using mentioned hierarchical relationships. As a result, vertices with indices $(a,b)_r$ after n levels of reverse subdivision are a coarse vertex if $\left\lfloor \frac{a}{2^n} \right\rfloor = \frac{a}{2^n}$ and their corresponding details are located at $(c,d)_r$ if $\left\lfloor \frac{c}{2^{n-1}} \right\rfloor = \frac{d}{2^{n-1}}$. Fig 19 illustrates the application of Catmull-Clark reverse subdivision to a connectivity map. To access the neighbors of a coarse vertex and its corresponding details, scaled neighborhood vectors are used. The structure of the neighborhood vectors remains the same as in Fig 5 but they are scaled by 2^n (after n levels of reverse subdivision) to access the neighbors of a coarse vertex and 2^{n-1} to access the corresponding details (Fig 19).

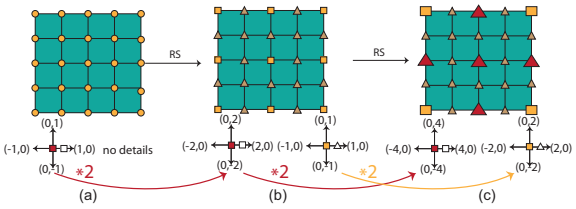

Figure 19: A subdivided connectivity map after two levels of reverse subdivision. □: coarse vertices. △ and ▲: details after one and two levels of RS. ↕ and ↔△: access to coarse vertices and details.

Loop: Since Loop subdivision uses a 1-to-4 refinement similar to Catmull-Clark subdivision, the only difference between these schemes is the different neighborhood vectors that must be used. Fig 20 illustrates a semiregular Venus and its reverse subdivision at three resolutions.

$\sqrt{3}$ subdivision: The filters of $\sqrt{3}$ reverse subdivision in a compact multiresolution framework have not yet been derived in the

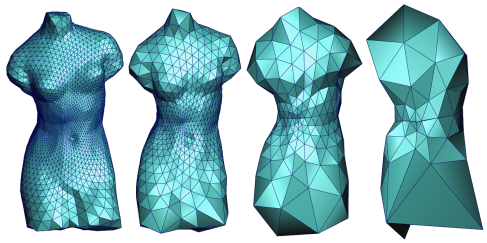

Figure 20: A semiregular venus after three applications of Loop reverse subdivision.

literature. Using the method proposed in [16], we have derived the reverse filters of $\sqrt{3}$ subdivision (see Appendix B for this derivation). To handle adjacency queries, we can use neighborhood vectors with a scaling factor of three. To access details and coarse vertices, transformations $T_{e\to o}$ and $T_{o\to e}$ discussed in Section 5.1 are used to change the neighborhood vectors as illustrated in Fig 21.

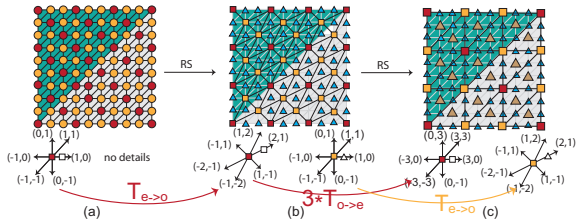

Figure 21: (a) ○ and ● are face and vertex vertices that are replaced by details and coarse vertices. (b) □ and △ : coarse vertices and details. (c) △:details after two levels of RS.  and  : access to coarse vertices and details. Transformations of essential neighborhood vectors are shown.

Connectivity operations for multiresolution methods are handled in $O(1)$ time since we only need to add scaled neighborhood vectors to the CM indices of vertices. Furthermore, similar to the discussion for subdivision methods, the ACM does not manipulate any pointers to change the connectivity when reverse subdivision is applied. To quantify this comparison, we compare the speed of ACM with the data structure of [15] to reverse subdivide a subdivided tetrahedron. Specification of the system is as mentioned in Section 6.1.

Table 2: Reverse subdivision time (in seconds) required by ACM and the data structure provided in [15] for a tetrahedron to reach resolution $i-1$ from resolution i.

Resolution	#of Faces	ACM	[15]
7	16384	0.027	2.851
6	4096	0.006	0.312
5	1024	0.001	0.032
4	256	0.0009	0.028

7 Conclusion

In this paper, we have proposed a mapping to capture the connectivity of semiregular meshes obtained from repetitive refinements. This mapping is based on the initial connectivity of the mesh by defining an atlas of 2D domains called connectivity maps. After setting up the connectivity, a local coordinate system (and transformations) is assigned to each connectivity map which is used to handle connectivity queries and refer to a 2D array containing the 3D locations of vertices. Having such an initial set up, it is possible to support regular refinements. Based on the effect of the refinements on the local coordinate system of each connectivity map, we have categorized the refinements and proposed how to handle them. Our method is applicable to triangular and quadrilateral faces. If the faces of the initial mesh are totally arbitrary, the mesh is first triangulated and then connectivity maps are assigned to each triangle pair. We have also examined the efficiency of our data structure for subdivision and multiresolution surfaces, and shown that our method is efficient in comparison with two other alternatives.

Acknowledgments

We thank Troy Alderson for proofreading and editorial comments. This research was supported in part by National Science and Engineering Research Council of Canada and GRAND Network of Centre of Excellence of Canada.

A Space Complexity Analysis

ACM is also efficient in terms of space. Consider a mesh with f faces, v vertices and e edges. ACM stores a constant amount of data for the initial set-up that does not grow through the resolutions. After r applications of 1-to-4 refinement, ACM stores $\approx 4^r \times v + (2^r+1) \times e$ vertices ($\approx (2^r+1) \times e$ are duplicated vertices at edges). Consider that an alternative data structure S stores only faces and vertices. S needs to store $4^r \times f$ faces and $4^r \times v$ vertices at resolution r. ACM stores $\approx (2^r+1) \times e$ more vertices than S while S stores $4^{r+1} \times f$ more pointers to refer to vertices. Since S needs data information with a larger exponential factor compared to ACM, after a few resolutions, ACM needs to store much less data. For triangular meshes, since ACM reduces number of faces to $\frac{f}{2}$ and edges to $e-\frac{f}{2}$ at the initial resolution by pairing triangles, it is even more efficient than the quad case.

The weakness of ACM is the need for empty indices at odd resolutions in refinements with rotations, such as the 1-to-2 refinement. In this case, ACM stores $\approx 4^r \times v + (2^r+1) \times e$ vertices while S stores $2^r \times f$ faces and $2^r \times v$ vertices. Note that S is a basic data structure and much simpler than edge-based data structures (such as the half-edge), yet ACM performs better than S. As a result, we can conclude that ACM is more efficient than other data structures, including half-edge, not benefiting from the regularity of refinements.

B $\sqrt{3}$ Reverse Subdivision

The filters of $\sqrt{3}$ reverse subdivision are provided in this section. Using the method proposed in [16], the following equations are obtained for $\sqrt{3}$ reverse subdivision. In this method, the details of vertex vertices (d_0) are determined by a linear combination of the details of face vertices (d_i) in their neighborhood (Fig 22). Equation 2 demonstrates this relationship. Equation 3 also indicates how the coarse vertex of a face vertex can be obtained using the vertices in its neighborhood. To avoid magnifying the results from reverse subdivision, coarse vertices are refined by vector δ_0 which is obtained by an optimization to reduce the magnitude of the details d_i [16]. Note that $\alpha = \frac{4-2cos(\frac{2\pi}{n})}{9}$ is the parameter to find the position of vertex-vertices in $\sqrt{3}$ subdivision [9].

$$d_0 = \frac{3}{2n}\alpha \sum_{i=1}^{n} d_i \quad (1)$$

$$c_0 = \frac{1}{1-\frac{3}{2}\alpha} f_0 - \frac{\alpha}{n(\frac{2}{3}-\alpha)} \sum_{i=1}^{n} f_i \quad (2)$$

$$\delta_0 = \frac{\frac{3}{2n}(1-\alpha)+\frac{1}{3}}{(1-\alpha)^2+\frac{n}{9}} \sum_{i=1}^{n} d_i \quad (3)$$

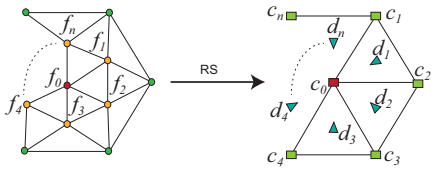

Figure 22: Vertex and face-vertices (●,● and ●) are replaced by coarse vertices and details (■ ,■ and ▲)

.

REFERENCES

[1] M. Alexa. Refinement operators for triangle meshes. *Comput. Aided Geom. Des.*, 19(3):169–172, 2002.

[2] M. Bertram. Biorthogonal loop-subdivision wavelets. *Computing*, 72(1-2):29–39, Apr. 2004.

[3] T. C. Biedl, P. Bose, E. D. Demaine, and A. Lubiw. Efficient algorithms for petersen's matching theorem. *Journal of Algorithms*, 38(1):110 – 134, 2001.

[4] D. Bommes, B. Levy, N. Pietroni, E. Puppo, C. S. a, M. Tarini, and D. Zorin. State of the art in quad meshing. In *Eurographics STARS*, 2012.

[5] M. Bunnell. *Adaptive tessellation of subdivision surfaces with displacement mapping, In: GPU Gems 2*, volume 2. Addison-Wesley, 2005.

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

[7] D. Doo and M. Sabin. Seminal graphics. chapter Behaviour of recursive division surfaces near extraordinary points, pages 177–181. ACM, 1998.

[8] I. P. Ivrissimtzis, N. A. Dodgson, and M. A. Sabin. A generative classification of mesh refinement rules with lattice transformations. *Comput. Aided Geom. Des.*, 21(1):99–109, 2004.

[9] L. Kobbelt. $\sqrt{3}$ subdivision. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH 2000, pages 103–112. ACM, 2000.

[10] P. Kraemer, D. Cazier, and D. Bechmann. Extension of half-edges for the representation of multiresolution subdivision surfaces. *The Visual Computer*, 25:149–163, 2009.

[11] G. Li, W. Ma, and H. Bao. $\sqrt{2}$ subdivision for quadrilateral meshes. *Vis. Comput.*, 20(2):180–198, 2004.

[12] C. Loop. Smooth subdivision surfaces based on triangles. Department of mathematics, Masters Thesis, University of Utah, 1987.

[13] A. Mahdavi-Amiri and F. Samavati. Connectivity maps for subdivision surfaces. In *GRAPP/IVAPP*, pages 26–37, 2012.

[14] T. Ni, A. H. Nasri, and J. Peters. Ternary subdivision for quadrilateral meshes. *Comput. Aided Geom. Des.*, 24(6):361–370, Aug. 2007.

[15] L. Olsen. *Constraining Wavelets for Multiresolution*. Canadian theses. University of Calgary (Canada), 2006.

[16] L. Olsen, F. Samavati, and R. Bartels. Multiresolution for curves and surfaces based on constraining wavelets. *Computers and Graphics*, 31(3):449 – 462, 2007.

[17] L. J. Olsen and F. F. Samavati. A discrete approach to multiresolution curves and surfaces. In *Proceedings of the 2008 International Conference on Computational Sciences and Its Applications*, pages 468–477. IEEE Computer Society, 2008.

[18] J. Peters and U. Reif. The simplest subdivision scheme for smoothing polyhedra. *ACM Trans. Graph.*, 16(4):420–431, Oct. 1997.

[19] J. Petersen. Die theorie der regulren graphs. *Acta Mathematica*, 15:193–220, 1891.

[20] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., 2005.

[21] L.-J. Shiue, I. Jones, and J. Peters. A realtime gpu subdivision kernel. *ACM Trans. Graph.*, 24:1010–1015, 2005.

[22] L.-J. Shiue and J. Peters. A pattern-based data structure for manipulating meshes with regular regions. In *Proceedings of Graphics Interface 2005*, GI '05, pages 153–160, 2005.

[23] E. J. Stollnitz, T. D. Derose, and D. H. Salesin. *Wavelets for computer graphics: theory and applications*. Morgan Kaufmann Publishers Inc., 1996.

[24] M. Tarini, N. Pietroni, P. Cignoni, and D. Panozzo. Practical quad mesh simplification. *CG Forum (Eurographics*, pages 407–418, 2010.

[25] K. Weiler. Edge-based data structures for solid modeling in curved-surface environments. *Computer Graphics and Applications, IEEE*, 5(1):21 –40, 1985.

[26] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 259–268. ACM Press/Addison-Wesley Publishing Co., 1997.