

CONNECTIVITY MAPS FOR SUBDIVISION SURFACES

Ali Mahdavi Amiri and Faramarz Samavati

University of Calgary, 2500 University Drive NW, Calgary, Canada

Keywords: Hierarchical Data Structure, Half-edge, Subdivision, Patch-based Refinement.

Abstract: In this paper, we introduce a hierarchical indexing for adjacency queries specially for applying subdivision schemes on some simple spherical and toroidal polyhedrons as the base model for the content creation process. The indexing method is created from integer coordinates of regular 2D domains (*connectivity maps*) resulting from unfolding polyhedrons. All connectivities are found using algebraic relationships of the connectivity map's indices; therefore, no additional data structure is required and operations are extremely fast and efficient. Although connectivity relationships of the polyhedrons are as simple as the first resolution, the models created by our proposed method is not restricted to the subdivided polyhedrons. Using our connectivity based method, complex objects can be created by adding sharp features and holes and applying deformation and remeshing techniques. We demonstrate capacities and the efficiency of the method with several example results and compare its speed with that of the half-edge data structure.

1 INTRODUCTION

To generate a complex graphical shape, it is common to start with a simple shape and progressively make it more complex using basic geometric operations. Most graphical software packages (e.g. Maya (Maya, 2011)) provide spheres, cubes, cylinders or toroidal polyhedrons as primitive objects. Basic operations for creating complex objects include repositioning their vertices, adding more vertices through subdivision and adding features (e.g. holes). In particular, using surface subdivision schemes not only provides more vertices for manipulating the shape but also creates a very useful hierarchy resulting from different levels of subdivision.

To support such a hierarchy and efficient operations (such as neighborhood-finding), a well-designed data structure is required. A common data structure for graphical objects is the half-edge data structure (Weiler, 1985; Kettner, 1998). However, the half-edge data structure is not naturally able to support subdivision hierarchies. Moreover, half-edge data structure is very slow at high levels of subdivision due to the extensive amount of connectivity information that the half-edge needs to store (see Section 9).

A frequently used alternative is quadtree representation (Samet, 2005). Quadtrees are able to support the hierarchy but are inefficient for neighborhood finding. Moreover, a quadtree needs to store connec-

tivity between nodes in consecutive resolutions making it inefficient in terms of space. To avoid this problem, there are some proposed indexing methods trying to assign an index to each vertex and remove the tree structure. Existing indexing methods for quadtrees are typically targeted at supporting hierarchies between parent and children nodes but they are unable to quickly find a specific node's neighbors. It is also possible to use patch-based methods for subdivision schemes (Bunnell, 2005; Shiue et al., 2005; Peters, 2000). However, using such methods connectivity of vertices specially for extra-ordinary and boundary vertices is not well defined. As a result, they need to maintain the connectivity using additional structures such as half-edge data structures.

Previously proposed data structures typically try to maintain the arbitrary connectivity using complicated pointer-based data structures. However, we establish our work based on simple polyhedrons with straightforward connectivities to ease the connectivity inquiries. To assign geometry on polyhedron's vertices, we use an indexing method as a data structure to support operations such as neighborhood finding, hierarchical traversal, and subdivision. To obtain this data structure, we first unfold a simple polyhedron, such as a cube, tetrahedron, octahedron, or toroidal polyhedron (see Section 3) as the shapes forming the base topology. The resulting 2D domain (*connectivity map*) is then used for assigning integer indices to

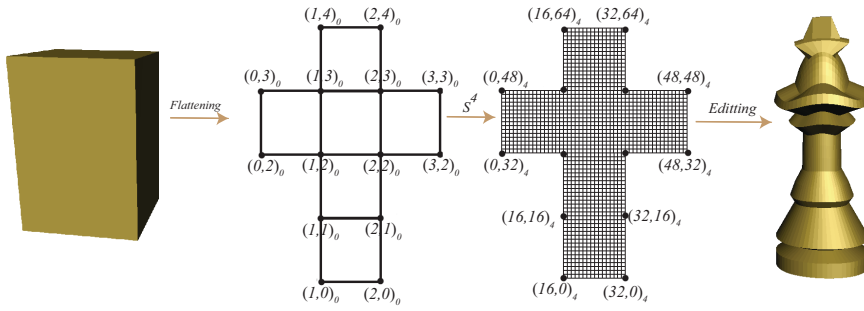


Figure 1: The initial cube’s connectivity map and its subdivided version. The cube’s vertices are manipulated and a chess piece is obtained. The connectivity map’s indices are shown beside them. $(a, b)_r$ indicates the vertex with index (a, b) at resolution r .

every vertex of the polyhedron based on its Cartesian coordinates. These integer indices refer to entries of a 2D array storing vertices’ 3D positions. Using such connectivity maps, simple operations in 2D are defined to find the neighbors of a face or vertex. Indeed, simplicity of the proposed connectivity maps result simple operations for connectivity inquiries. Using such connectivity maps, all the vertices, edges and face’s connectivities can be found using algebraic relationships. In summary, our approach is devoted to defining the connectivity and the topology of geometric models as a primary goal while the geometry and its modifications (including subdivision schemes) are added as a secondary goal.

Although we use simple polyhedrons, complex objects can be created using this data structure. For creating complex shapes, the initial simple polyhedron can be subdivided using various subdivision schemes such as linear, Catmull-Clark or Loop subdivision. The location of subdivided vertices can then be modified (manually or automatically) using deformation techniques or remeshing methods to create the desired shapes and holes and sharp features can also be added. Note that in this representation, it is only necessary to store vertex positions since the connectivity of edges and faces is implicit (See Fig 1).

Our main contribution is to introduce connectivity maps for polyhedrons. The coordinates of the connectivity maps’ vertices are considered as indices that can address adjacency queries and provide 3D locations. Initial connectivity maps are provided and complicated pre-processing steps are not required. After establishing the initial connectivity maps, subsequent resolutions are formed based on the initial connectivity maps. Using the proposed connectivity maps and the indexing method, adjacency and hierarchical queries are addressed using fast and straightforward algebraic relationships.

The rest of the paper is organized as follows: Related work is discussed in Section 2. We discuss the

indexing scheme in Section 3. Hierarchical traversal and neighborhood vectors are discussed in Section 4. We clarify how to apply Loop and Catmull-Clark subdivision using our method in Section 5. Hierarchical mesh editing is discussed in Section 6. We describe some possible methods to generate complex objects with simple connectivities in Section 7. Extracting connectivity information of geometry images is described in Section 8 and finally we compare the speed of the proposed indexing with that of the half-edge in Section 9.

2 RELATED WORK

Generating an object using subdivision schemes such as Loop and Catmull-Clark is a standard method for geometric modeling (Loop, 1987; Catmull and Clark, 1998). The half-edge and quadtree are two commonly used data structures to support subdivision schemes (Weiler, 1985; Kettner, 1998; Samet, 2005; Zorin et al., 1997). However, when the information stored by a data structure becomes very large, it is inefficient to use pointer based data structures for storing connectivity information (Samet, 1990; Samet, 1985). In order to avoid pointers, some indexing methods have been proposed to index mesh vertices and faces.

Proposed indexing methods for quadtrees (Samet, 1985; Gargantini, 1982) can be used for representing meshes. However, as a very important inquiry for mesh processing, neighborhood finding is not efficient enough for interactive applications. In our work, we derive some simple algebraic operations for finding neighbors of a given face or vertex. The benefit of algebraic operations is not only that neighbors of a face or vertex can be found in constant time, but further faces or vertices such as neighbors of neighbors (more than 1-ring neighborhood) are also accessible in constant time.

Patch-based methods generally divide the mesh

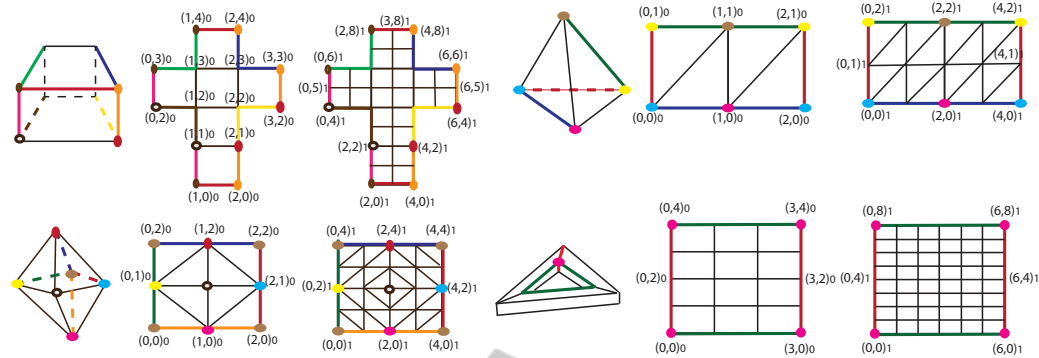


Figure 2: A cube, tetrahedron, octahedron and a toroidal polyhedron and their 2D domain at the first and the second resolutions. $(a, b)_r$ indicates the index of a vertex at resolution r . Vertices or edges with the same colors on 2D domain are copies of one vertex or edge in 3D. Boundary edges are highlighted by thicker lines.

into collection of separated faces and consider arrays for each single patch (Bunnell, 2005; Shiue et al., 2005; Peters, 2000). These methods are specifically designed for handling subdivision schemes specially Catmull-Clark subdivision. Using the method presented by (Bunnell, 2005), the surface can be divided into some patches connected by an edge based data structure such as the half-edge and fine resolution vertices are stored in separate 2D arrays. This method is designed for general quad meshes resulting from Catmull-Clark subdivision but it cannot support triangular meshes and Loop subdivision. In comparison with (Bunnell, 2005), we propose an indexing for important simple shapes. We can support Loop and Catmull-Clark subdivision schemes and we establish algebraic relationships for neighbors of boundary vertices that enable us to avoid half-edge data structure at the first resolution. Although we use simple polyhedrons for establishing connectivities, our method is not restricted to simple objects. Using remeshing and sketch-based deformation techniques and supporting sharp features and holes, we can create complex objects that benefit from polyhedron's simplicity.

Shiue et al. use an alternative way to apply subdivision methods (Shiue et al., 2005). They initially subdivide the mesh using an edge based data structure in a pre-processing step. They then divide the mesh into some patches with repetitive boundary vertices. Each patch's vertices is indexed using a spiral indexing and located in a 1D array. Using a spiral indexing methods, close vertices may get unrelated indices. Therefore, this method does not establish a straightforward relationship for neighborhood finding and access to neighbors of a vertex needs to traverse the spiral. Consequently, previous patch based refinement methods are specifically designed to support subdivision methods and they do not consider the shape's connectivity relationship which is an important aspect of the geometric modeling.

Since our method uses a flattened regular shape for meshes, it is also somehow related to the parametrization problem (Hormann et al., 2007). Parameterization is used for different applications such as texture mapping, mesh editing and Geometry Images. For creating Geometry Images, Gu et al. (Gu et al., 2002) uses cuts in order to flatten a general topology object. They then resample the flattened object to create a regular surface called a "geometry image". They generate an image to store the vertex locations of objects in such a way that red, green and blue components of an image represent x, y and z locations of a vertex respectively. Boundary extension rules are not straightforward in this method; therefore, connectivity relationships are hard to find for boundaries.

Praun and Hoppe (Praun and Hoppe, 2003) propose a spherical parameterization to generate regular meshes of genus 0 objects using the octahedron, cube or tetrahedron. They parameterize an object to a sphere then use one of the mentioned polyhedra for resampling and create a regular mesh. Meshes obtained from (Praun and Hoppe, 2003) are dependent on the geometry of initial shapes that have to be remeshed. Therefore, small changes in geometry images need modification of the initial shape, requiring an expensive reparameterization of the entire mesh. However, our method begins with a 2D regular domain (connectivity map) for an efficient access to the connectivity information, and also maintains this regularity during geometry changes (editing, deformation, and subdivision operations). Additionally, changing the mesh at any resolution can be propagated to further resolutions without using expensive operations. Because of this, our method can be a good complementary work for Geometry Images. In summary, to have a comparable title with (Gu et al., 2002), our work can be conceptually considered as *Connectivity Images*.

3 THE INDEXING SCHEME

Since the indexing method works based on 2D vectors of Cartesian coordinates, the initial polyhedron must be flattened. There are many ways to unfold these initial objects but we use the one in which connectivity maps have easier boundary rules (see Figures 1 and 2). We then locate every vertex of the polyhedron on integer Cartesian coordinates. These integer coordinates are used as the vertices' index. Working with this simple shape is not enough for many applications and it only creates a base topology of shapes, therefore it is necessary to support subdivision. To keep track of the levels of subdivision, we also use an extra index, the subscript index, as the *resolution* or level of subdivision. Therefore, a vertex with index $(a, b)_r$ is a and b steps far from the origin in X and Y directions at resolution r .

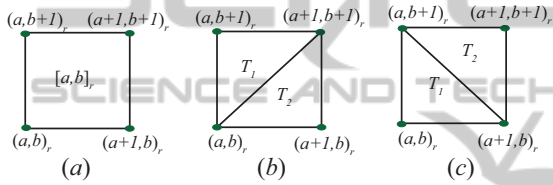


Figure 3: (a) A quadrilateral face, and the index of its faces and vertices. (b,c) Two forms of triangular faces of a quadrilateral face.

We use the resulting 2D integer coordinates as indices to a 2D array containing the locations of vertices. We also use the same index for addressing faces. The index of the left-bottom vertex of a face is considered as its index, based on a face's index, indices of its vertices are computable and vice versa. Indeed, if the left bottom vertex of a face is $(a, b)_r$, vertices making this face have indices $(a, b)_r$, $(a + 1, b)_r$, $(a + 1, b + 1)_r$ and $(a, b + 1)_r$. This relationship is valid for quadrilateral faces. Because of the regularity of triangular faces in our initial polyhedron, it is possible to assign faces' indices to quads formed by triangles. There are two possible triangular faces for each quad (see Fig 3). All triangular faces of a tetrahedron has the form of Fig 3.b. However, based on the region in which a quad falls (see Fig 2), triangles of an octahedron can have forms of Fig 3.b or Fig 3.c.

4 HIERARCHICAL TRAVERSAL AND NEIGHBORHOOD VECTORS

Hierarchical Traversal. In our method, when a face is subdivided using linear, Catmull-Clark or Loop subdivision methods, it is split into four faces after each level. Fine faces generated by splitting a coarse face are called *children* and the coarse face is called *parent*. One of the important hierarchical operations is to find indices of all the children of a given face $[a, b]_r$ (we use $[a, b]_r$ for faces' indices to distinguish them from vertices' indices represented by $(a, b)_r$) at resolution $r + k_1$ or finding the index of its parent at resolution $r - k_2$ ($k_1, k_2 > 0$).

To maintain efficient hierarchical operations and regularity, after each level of subdivision, we use the same 2D coordinates for the vertices from previous levels. Since every face is split into four faces, the length of unit vectors is divided by two. Therefore, unit vectors $(0, 1)_r$ and $(1, 0)_r$ are equal to $2 \times (0, 1)_{r+1}$ and $2 \times (1, 0)_{r+1}$ respectively. Resulting from these relationships, we can derive the hierarchical relationship of a vertex at different resolutions. Equations 1 and 2 imply correspondent indices of vertex $(a, b)_r$ at resolutions $r + k$ and $r - k$ respectively. Note Equation 2 must only applied to vertex-vertices.

$$(a, b)_r = (2^k a, 2^k b)_{r+k} \quad (1)$$

$$(a, b)_r = \left(\frac{a}{2^k}, \frac{b}{2^k}\right)_{r-k} \quad (2)$$

Similarly, a face with index $[a, b]_r$ has children with indices $[2a, 2b]_{r+1}$, $[2a + 1, 2b]_{r+1}$, $[2a, 2b + 1]_{r+1}$ and $[2a + 1, 2b + 1]_{r+1}$ (Fig 5). In general after n ($n > 0$) levels of subdivision, a face with index $[a, b]_r$ would have children with indices $[c, d]_{r+n}$ in which $2^n a \leq c < 2^{n+1} a$ and $2^n b \leq d < 2^{n+1} b$. Therefore, a face with index $[a, b]_r$ has a parent with index $\left[\left\lfloor \frac{a}{2^n} \right\rfloor, \left\lfloor \frac{b}{2^n} \right\rfloor\right]_{r-n}$.

Neighborhood Vectors. Since each vertex of the connectivity map is located on an integer 2D Cartesian coordinate, there are vectors that connect vertex v to all its neighbors. These vectors are called *neighborhood vectors* which differ based on the underlying polyhedra's connectivity maps. Figure 4 illustrates them for the initial polyhedra.

Neighborhood vectors illustrated in Figure 4 are used for internal vertices. Unfolded polyhedrons have boundary edges and vertices which are shown in Figure 2. Since boundary vertices have multiple copies in connectivity maps, their neighbors are split into multiple locations surrounding each copy. As a result, to determine neighbors of boundary vertices, their copies must also be found (Fig 2).

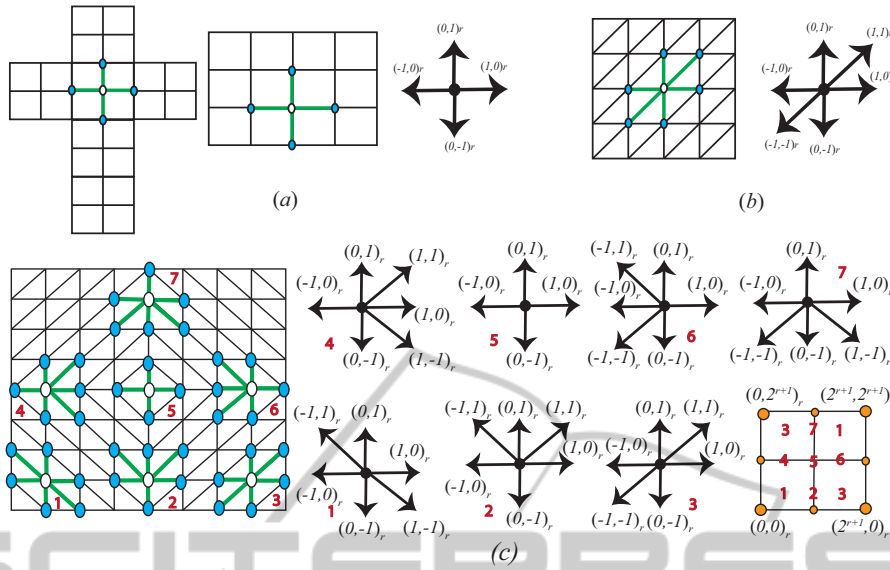


Figure 4: (a) Neighborhood vectors for a cube or a toroidal polyhedron . (b) Neighborhood vectors for a tetrahedron. (c) Seven possible neighborhood vectors for an octahedron. Based on the region in which the vertex falls, neighborhood vectors can be determined. For example, if a vertex falls in the region $0 < a, b < 2^r$, its neighborhood vector is of the condition 1.

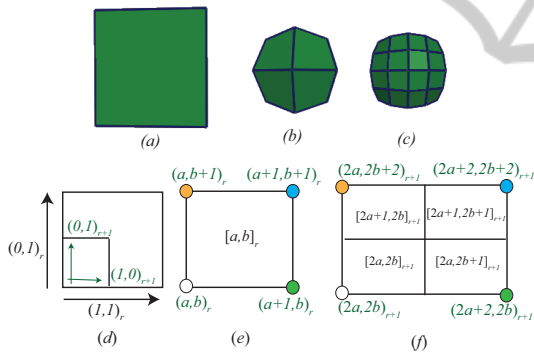


Figure 5: (a) A face at resolution 0. (b) The patch obtained from (a) at resolution 1. (c) The patch obtained from (a) at resolution 2. (d) Unit vectors at resolutions r and $r + 1$. (e) A face's index and its vertices's indices at resolution r . The patch obtained from (e) and it's faces' indices. Green indices are vertex indices and black ones are face's indices.

To recognize if $A = (a, b)_r$ is a boundary vertex, we check whether it falls along one of the boundary edges of the initial connectivity map. Consider boundary edge e connecting two first resolution vertices $P = (x_p, y_p)_0$ and $Q = (x_q, y_q)_0$. A is located on boundary edge e if its index $(a, b)_r$ is in between of the indices of P and Q at resolution r . Using Equation 1, P and Q respectively have indices $(2^r x_p, 2^r y_p)_r$ and $(2^r x_q, 2^r y_q)_r$ at resolution r . Therefore, $A = (a, b)_r$ is located on e if $\min\{2^r x_p, 2^r x_q\} \leq a \leq \max\{2^r x_p, 2^r x_q\}$ and $\min\{2^r y_p, 2^r y_q\} \leq b \leq \max\{2^r y_p, 2^r y_q\}$. This relationship is general for all

boundary edges. However, boundary edges of the proposed polyhedrons typically have simpler conditions. For instance, the toroidal polyhedron has four boundary edges and $A = (a, b)_r$ is located along one of the boundary edges when $a = 0, b = 0, a = 3 \times 2^r$ or $b = 4 \times 2^r$ (Fig 2).

After recognizing that a vertex is located on the connectivity map's boundary, its neighbors have to be determined. As we discussed earlier, to find neighbors of a boundary vertex, its copies are required. Copies of vertices at the initial resolution are known based on the connectivity map. To determine copies of a boundary vertex at resolution r , we propose a general method that can be applied to all the proposed polyhedrons. This general method's basic idea is to first find edge e on which boundary vertex A is located (Fig 6). Afterwards, having the e 's copy which is obtainable from the connectivity map, the index of A 's copy is determined. We describe the mathematical representation of this issue in the following.

Let $A = (a, b)_r$ be a boundary vertex whose copy, $C = (c, d)_r$, is required (see Fig 6). A is located on a first resolution edge e connecting $P = (x_p, y_p)_0$ and $Q = (x_q, y_q)_0$. P and Q respectively have copies $Z = (x_z, y_z)_0$ and $W = (x_w, y_w)_0$ at the first resolution. A can be written as a linear combination of P and Q . $A = (1 - \alpha)P + \alpha Q$. Using copies of P and Q , the copy of A can be determined as $C = (1 - \alpha)Z + \alpha W$.

For instance, consider the vertex with index $A = (a, b)_r = (0, 21)_3$ on the cube's 2D domain (Fig 6). Since $a = 0$, this vertex is located on the edge con-

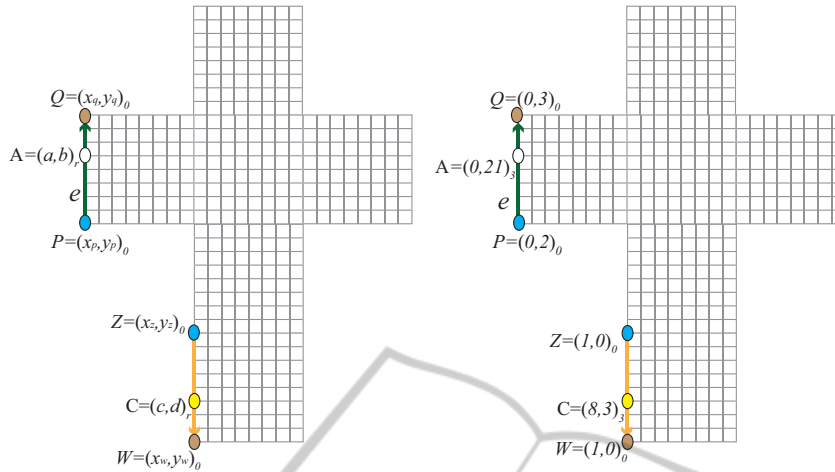


Figure 6: Left: copy of $A = (a, b)_r$ is shown by a white vertex. Copy of $A, C = (c, d)_r$, is found using the described general method. e and its copies are respectively highlighted by green and orange vectors. Right: an example of the general method to find the copy of $(0, 21)_3$.

necting vertices with indices $P = (0, 2)_0 = (0, 16)_3$ and $Q = (0, 3)_0 = (0, 24)_3$. Therefore, $(0, 21)_3 = (1 - \alpha)(0, 16)_3 + \alpha(0, 24)_3 \Rightarrow \alpha = \frac{5}{8}$. Since copies of P and Q have respectively indices $Z = (1, 1)_0 = (8, 8)_3$ and $W = (1, 0)_0 = (8, 0)_3$. $C = \frac{3}{8}(8, 8) + \frac{5}{8}(8, 0)_3 = (8, 3)_3$.

After finding copies of a boundary vertex (A), neighborhood vectors are applied on A and its copies. Resulting neighbor vertices are stored in set N . Afterwards, Repetitive Vertices of N and those vertices that fall out of the 2D domain are excluded. To discard repetitive vertices, we keep neighbors that are directly obtained by applying neighborhood vectors on A and then eliminate their copies. It is not necessary to do the same process for every single resolution. It is enough to find neighbors of boundary edges at one resolution and then extend it to next resolutions using Equation 1.

5 SUBDIVISION METHODS

To generate complex objects, it is possible to subdivide the initial object, manipulate resulting vertices in a small number of iterations until the desired object is formed. After an application of linear, Loop or Catmull-Clark subdivision, each face is split into four faces, and some new vertices are generated. To support the possibility of creating more advanced shapes and also to demonstrate the usefulness of the indexing method proposed for the connectivity maps, in this section, we describe how to perform smooth subdivision schemes using the indexing method.

5.1 Catmull-Clark Subdivision

Each iteration of Catmull-Clark subdivision requires a face split operation and also a relocation of resulting vertices through subdivision masks. We have already described hierarchical relationships resulting from the face split in Section 4. Here we discuss how to apply subdivision masks on the quad faces of the cube and toroidal polyhedron. Masks of Catmull-Clark subdivision are as follows (Halstead et al., 1993):

$$f_j^{i+1} = \frac{1}{4} \sum_j v_j^i \quad (3)$$

$$e_j^{i+1} = \frac{v^i + e_j^i + f_{j-1}^{i+1} + f_j^{i+1}}{4} \quad (4)$$

$$v^{i+1} = \frac{n-2}{n} v^i + \frac{1}{n^2} \sum_j e_j^i + \frac{1}{n^2} \sum_j f_j^{i+1} \quad (5)$$

In the neighborhood of vertex v^i , f_1^{i+1} , f_2^{i+1} , ..., f_n^{i+1} and e_1^{i+1} , e_2^{i+1} , ..., e_n^{i+1} are respectively face-vertices and edge-vertices and v^{i+1} is the vertex-vertex at the resolution $i+1$.

After one level of subdivision, a face with index $[a, b]_r$ is split into four faces $[2a, 2b]_{r+1}$, $[2a+1, b]_{r+1}$, $[2a+1, 2b+1]_{r+1}$ and $[2a, 2b+1]_{r+1}$ as illustrated in Fig 7. These faces' vertices must be computed using Catmull-Clark subdivision masks. Applying Equation 3 on face $[a, b]_r$, the face-vertex $Q_{(1,1)} = (2a+1, 2b+1)_{r+1}$ is computed using the following equation:

$$Q_{(1,1)} = \frac{P_{(0,0)} + P_{(1,0)} + P_{(1,1)} + P_{(0,1)}}{4}, \quad (6)$$

where $P_{(0,0)}$, $P_{(1,0)}$, $P_{(1,1)}$, and $P_{(0,1)}$ respectively have indices $(a, b)_r$, $(a+1, b)_r$, $(a+1, b+1)_r$, and $(a, b+1)_r$.

After finding face vertices, edge-vertices $(2a+1, 2b)_{r+1}$, $(2a+2, 2b+1)_{r+1}$, $(2a+1, 2b+2)_{r+1}$ and $(2a, 2b+1)_{r+1}$ are computed using (4). For example, as demonstrated in Fig 7 (d), the edge-vertex $Q_{(1,2)} = (2a+1, 2b+2)_{r+1}$ is determined by:

$$Q_{(1,2)} = \frac{Q_{(1,1)} + Q_{(1,3)} + P_{(0,1)} + P_{(1,1)}}{4}, \quad (7)$$

where $Q_{(1,1)}$, $Q_{(1,3)}$, $P_{(0,1)}$, and $P_{(1,1)}$ respectively have indices $(2a+1, 2b+1)_{r+1}$, $(2a+1, 2b+3)_{r+1}$, $(a, b+1)_r$, and $(a+1, b+1)_r$.

After finding all edge-vertices, it is possible to compute the vertex-vertices. Vertex-vertices are in fact existing vertices at the coarser resolution translated to new positions. When $(a, b)_r$ is a regular vertex, the involving neighboring vertices in the vertex-vertex mask can be easily found. As demonstrated in Fig 7 (e), we need face vertices with indices $(2a+1, 2b+1)_{r+1}$, $(2a-1, 2b+1)_{r+1}$, $(2a-1, 2b-1)_{r+1}$ and $(2a+1, 2b-1)_{r+1}$, and edge vertices with indices $(a-1, b)_r$, $(a+1, b)_r$, $(a, b+1)_r$ and $(a, b-1)_r$. Using these indices and Equation 5 the position of $(2a, 2b)_{r+1}$ can be computed.

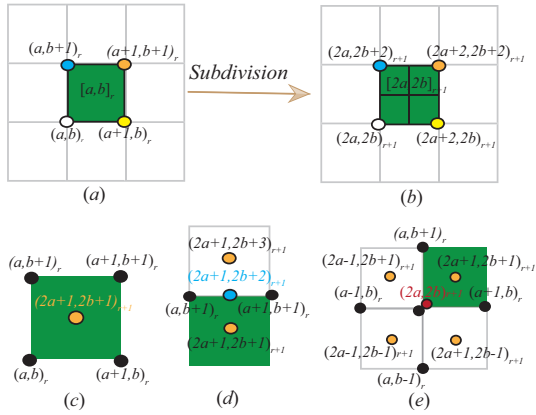


Figure 7: (a) Face $[a, b]_r$ and its vertices. (b) Children of face $[a, b]_r$ at resolution $r+1$ and their vertices. ((c), (d), (e)) Black vertices are vertices at resolution r . Orange, blue and red vertices represent face, edge and vertex-vertices respectively.

It is also necessary to handle extraordinary vertices ($n \neq 4$). Since each step of Catmull-Clark subdivision does not introduce any new extraordinary vertex the set of extraordinary vertices resulting from this subdivision correspond to the extraordinary vertices of the initial connectivity map. Therefore, the index of this kind of vertices is predictable during

subdivision. In fact, using Equation 1, given the index of the extraordinary vertices at the first resolution, the index of extraordinary vertices at any resolution r can be found. For example, extraordinary vertices of the cube are: $(2^r, 0)_r$, $(2^{r+1}, 0)_r$, $(2^{r+1}, 2^{r+1})_r$, $(2^{r+1}, 2^r)_r$, $(0, 2^{r+1})_r$, $(2^r, 2^{r+1})_r$, $(3 \times 2^r, 2^{r+1})_r$, $(0, 3 \times 2^r)_r$, $(2^r, 3 \times 2^r)_r$, $(2^{r+1}, 3 \times 2^r)_r$, $(3 \times 2^r, 3 \times 2^r)_r$, $(2^r, 2^{r+2})_r$ and $(2^{r+1}, 2^{r+2})_r$. So by keeping track of extraordinary vertices and applying appropriate masks to them, we can subdivide the initial polyhedra. We applied Catmull-Clark subdivision to the cube and toroidal polyhedron. Figure 8 shows a toroidal polyhedron at three different resolutions.

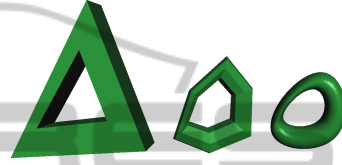


Figure 8: The toroidal polyhedron at three different Catmull-Clark subdivision levels.

5.2 Loop Subdivision

Another common subdivision method is Loop's scheme (Loop, 1987). In this triangular-based scheme, each triangular face is split to four triangles. There are two types of masks for Loop subdivision: edge-vertex and vertex-vertex masks. Using the notation demonstrated in Fig. 9, these masks are defined:

$$e^{i+1} = \frac{3}{8}v_1^i + \frac{3}{8}v_2^i + \frac{1}{8}v_3^i + \frac{1}{8}v_4^i \quad (8)$$

$$v^{i+1} = (1 - n\alpha)v^i + \alpha \sum v_j^i \quad (9)$$

$$\alpha = \frac{1}{n} \left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4} \cos \left(\frac{2\pi}{n} \right) \right)^2 \right) \quad (10)$$

We discuss the case of neighborhood vectors for a tetrahedron. The octahedron case is similar (note that it is also possible to apply Loop subdivision on the toroidal polyhedron and cube by splitting every quad to two triangles). To apply the mask of edge-vertices presented in Equation 8, vectors $(1, 0)_r$, $(1, 1)_r$ and $(0, -1)_r$ are needed. For vertex-vertices however, all the neighborhood vectors presented in Section 4 (also Fig 4) for the tetrahedron are required to find necessary neighbors. Extraordinary vertices are handled similar to Catmull-Clark. Figure 10 illustrates the result of Loop subdivision on the initial tetrahedron.

5.3 Holes and Sharp Features

There are many objects that have sharp features and holes. A data structure should be able to support these

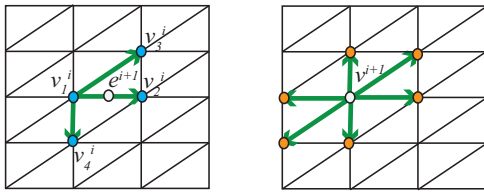


Figure 9: Left: necessary neighborhood vectors for computing edge vertices. Right: necessary neighborhood vectors for computing vertex vertices.



Figure 10: Two left figures are the tetrahedron at the second and third resolutions. Three right figures are the cube with sharp edges along the bottom over three successive levels of subdivision. The last subdivided cube is depicted in another view.

features to make realistic and useful objects. In this section, we describe how to handle holes and sharp features.

5.3.1 Holes

To support objects with holes, individual faces at any arbitrary resolution are tagged as *empty*. As discussed in Section 4, it is possible to find all the faces in the resolution $r+k$ that are children of a face at the resolution r . Using this property, all children of an empty face and its related vertices are marked as empty in further resolutions. Once we render the mesh, empty faces and their vertices (except the vertices of the boundary) are ignored. A cubic B-spline subdivision mask can be applied to the edges (vertices of the boundary) of these holes. Figure 11 shows a cube with three holes at three successive levels of subdivision.

5.3.2 Sharp Features

To support sharp features, we use the method described in (DeRose et al., 1998) and (Hoppe et al., 1994) for Catmull-Clark and Loop schemes respectively. In general, for objects that have sharp features, there are three kinds of vertices. Normal vertices, vertices with no sharp edges; creases, vertices with two sharp edges that a cubic B-Spline mask should be applied to them; and corners, vertices with more than two sharp edges whose positions are changed.

Consider a sharp edge between $(a, b)_r$ and $(a+1, b)_r$, after k levels of subdivision all the edges between a pair of vertices in the range $(2^k a, 2^k b)_{r+k}$ and $(2^k a + 2^k, 2^k b)_{r+k}$ are sharp. Therefore, vertices of

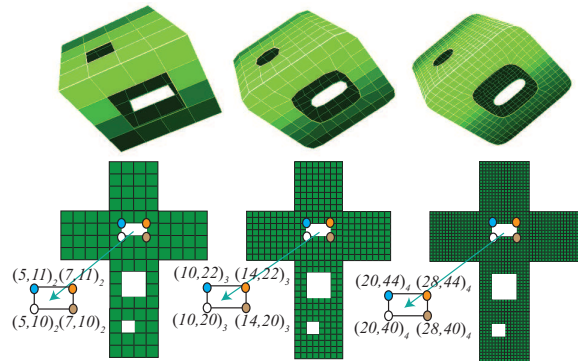


Figure 11: Top: A cube is subdivided using linear subdivision twice, three holes are made in the cube and it is subdivided twice using Catmull-Clark subdivision. Bottom: The connectivity maps of the top figures. It is possible to keep the track of empty faces' indices during the subdivision that are highlighted by white blocks.

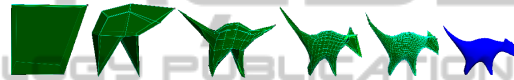


Figure 12: The evolution of a cat from a cube using direct manipulations of vertices.

these edges are known during resolutions and a sharp or corner mask is applied to them. Figure 10 shows a cube with sharp edges along the bottom over three successive levels of subdivision.

It is not necessary to tag all the sharp edges at further resolutions. To apply different masks to sharp features, we keep an array including all sharp features at the coarsest resolution $((a, b)_r)$. We first subdivide the mesh without considering sharp tags and then find all the sharp edges' vertices at current resolution (from $(2^k a, 2^k b)_{r+k}$ to $(2^k a + 2^k, 2^k b)_{r+k}$) and modify their positions by the appropriate mask.

6 HIERARCHICAL MESH EDITING

To generate a mesh, one can modify an initial polyhedron, subdivide and modify, and add features to it until the desired shape is achieved (Fig 12 and 13). During this process, one may want to go back to coarser resolutions and makes some modifications. To modify the model at a coarser resolution, we can use the connectivity map of the coarse resolution, modify some vertices and propagate the modification to the current resolution using some local subdivision masks.

Consider vertex v is modified by vector Δv . Δv has

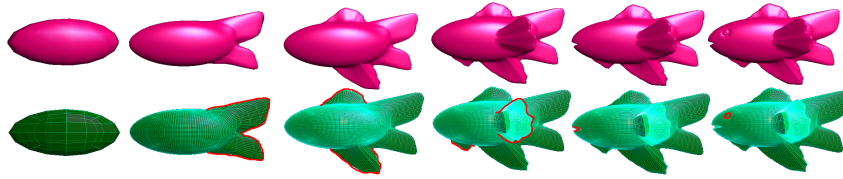


Figure 13: The evolution of a fish from a subdivided cube using sketches.

local contributions at subsequent resolutions. Instead of subdividing the entire mesh, we can determine the contributions and modify affected vertices. Since there are extra-ordinary vertices in the proposed polyhedrons, there are three types of local masks based on the location of the modified vertex. These three cases are: 1. a regular vertex without any direct connection to an extra-ordinary vertex; 2. a regular vertex adjacent to an extra-ordinary vertex; 3. an extra-ordinary vertex. Figure 14 illustrates these possible cases for the toroidal polyhedron. In Figure 14, the extra-ordinary vertex has valence 3 and its subdivided vertex is highlighted by a red vertex when it is located in a regular vertex's neighborhood. Therefore, setting n to 6 and 3 in Equation 10 respectively results α_1 and α_2 shown in Figure 14. These possible cases can also be determined using a similar method for other proposed polyhedrons and Catmull-Clark subdivision.

Note for case 2, the extra-ordinary vertex may be located at a different edge as shown in Fig 14. In this situation, the local mask is rotated in such a way that extra-ordinary vertex of the mask is matched with the extra-ordinary vertex of the regular vertex's neighborhood.

We do not consider the case of two extra-ordinary vertices in one neighborhood which occurs only at the first resolution. For low resolution objects, it is efficient to globally subdivide the 2D domain considering Δv for the modified vertex and 0 for other vertices. However, vertex modifications at high resolutions using global subdivision need huge amount of calculations therefore it is more efficient to apply the local subdivision for the modification and recursively propagate it to subsequent resolutions.

7 OTHER GEOMETRIC MANIPULATIONS

So far, we have shown that our method can efficiently support subdivision schemes. In general, any solitary geometric operation on the initial connectivity map is very simple and efficient. However, any major geometric change may require a new distribution of vertices in a multi scale manner. To show that the proposed method can be employed in various scenarios,

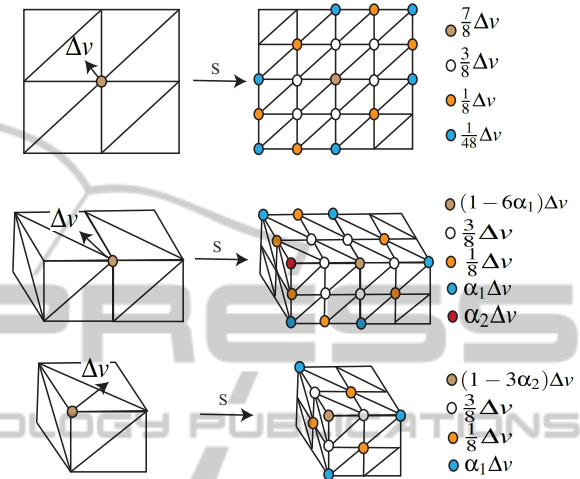


Figure 14: Local subdivision masks for described case. Top: Case 1. Middle: Case 2. Bottom: Case 3. Right figures represent the modified vertices' locations after one level of subdivision due to the modification of Δv . The modification values of the colorful ovals are presented at the right of each figure. Vertices highlighted by colorful ovals get the same modification values as appeared near the ovals.

we use sketch-based deformation and remeshing techniques to generate more complex objects.

7.1 Sketch-based Deformation

To deform a model, the positions of vertices are somehow modified. Such modifications can be done simply using direct manipulation of vertices at any resolution (see Fig 1 and 12) or using some deformation methods. To show the capacities of our method, we have employed the deformation technique proposed in (Pusch and Samavati, 2010) in a sketch-based modeling interface to deform an initial shape to make more complex shapes as the one demonstrated in Fig 13.

Another benefit of our method for the sketch-based modeling is to provide a very regular meshing of closed, sketched curves. There are several applications in the sketch-based modeling in which a user draws a closed curve and a 3D mesh is obtained from the curve (Olsen et al., 2009; Nealen et al., 2007). It is appropriate that the resulting mesh of these appli-

cations becomes regular with very low number of extraordinary vertices (Nealen et al., 2009; Nasri et al., 2009). To achieve this goal, we start with an octahedron and fit its base to the sketched curve by snapping vertices of the octahedron’s base to the sketched curve (Olsen et al., 2009). The base of an octahedron is a square at the first resolution. After each level of subdivision, the base of the octahedron becomes smoother and can approximate the curve in a better way. By repeating the subdividing and fitting process, we can increase the accuracy of the stroke’s approximation as demonstrated in Fig 15. For inflation, we can use methods described in (Olsen et al., 2009; Igarashi et al., 2007). The resulting mesh is very regular and only has six extraordinary vertices.

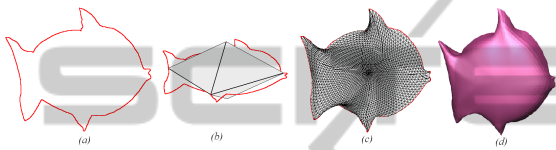


Figure 15: (a) A sketched curve. (b) The base of the octahedron is fit to the curve. (c) The mesh resulted from subdividing and fitting process described above. (d) Mesh is rendered using smooth shading.

7.2 Remeshing Techniques

Remeshing methods typically try to approximate a target mesh using a simple base mesh such as a subdivided polyhedron. One possible way to approximate a general mesh is to use a method similar to (Sharf et al., 2006). In (Sharf et al., 2006), the target mesh is approximated using a simple base mesh (sphere). The base mesh’s vertices are moved in the direction of their normals (evolving process) until the distance

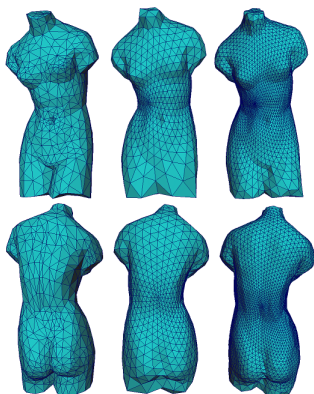


Figure 16: Left figures are the venus with a general mesh. Middle and right figures are the approximation of left figures using a method similar to (Sharf et al., 2006) at two successive levels of approximation.

between the vertices of the base mesh and the target mesh is less than a threshold. After the evolving process, the base mesh is locally subdivided to have a better approximation of the target. The base mesh again evolves and this process continues until the deformable model is close enough to the target mesh. We use a similar approach but we subdivide the whole base mesh to approximate the target mesh. Figure 16 illustrates this approach to approximate the Venus.

8 GEOMETRY IMAGES

Our connectivity map can also be used as a complementary tool for deforming, modifying and subdividing meshes resulting from geometry images (Losasso et al., 2003; Praun and Hoppe, 2003). To apply our proposed method on a geometry image resulting from the spherical parameterization method, we must first synchronize the resolutions. This means that number of vertices in the geometry image and the number of vertices of the connectivity map must be equal. Therefore, we subdivide the initial polyhedron (consistent with one that is used by the spherical parameterization) until we get the same resolution. Then we use the geometry image’s data to set the vertex positions of our indices. We now are able to edit, modify and subdivide the mesh. Figure 17 illustrates an object obtained by geometry image’s approach before and after applying Loop subdivision.

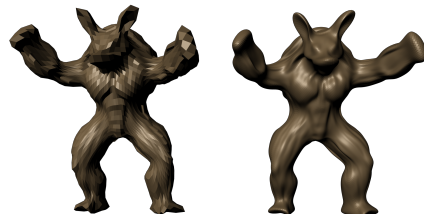


Figure 17: Left: an object obtained from geometry images. Right: left figure after one level of loop subdivision.

9 SPEED EFFICIENCY

One of the most common data structures for meshes is the half-edge (Weiler, 1985; Kettner, 1998). This data structure needs to save edges, pairs of edges, vertices (their location and their connection to half-edges) and faces. In our method, the only information that must be stored is the location of vertices and the initial connectivity map’s indices. Connectivity information is efficiently extracted from these indices.

Table 1: Number of faces in every step of Loop subdivision of the tetrahedron and run-time of our method (connectivity map) and the half-edge. NA shows an extremely large number.

Steps of Subdivision	No. of faces	Our method (seconds)	Half-edge (seconds)
1	4	0.01562	0.031
2	16	0.01562	0.031
3	64	0.01562	0.0625
4	256	0.01562	0.1093
5	1024	0.01562	0.89
6	4096	0.04687	7.29
7	16384	0.078125	185.78
8	65536	0.3125	NA
9	262144	1.375	NA
10	1048576	5.82	NA

Our proposed method is not only efficient in terms of space, but it is also very fast in comparison with the half-edge. To quantify the speed benefit, we compare the run times needed by the half-edge and the indexing in 10 different levels of a Loop subdivision on a tetrahedron. The results of other polyhedrons and Catmull-Clark subdivision are fairly similar. Table 1 shows the run-time (CPU) of the connectivity map and the half-edge data structure. Note the time at step i indicates the required time to achieve resolution i from the initial tetrahedron. It is readily apparent that after few levels of subdivision, our method is significantly faster than the half-edge.

Although for low resolution objects, the half-edge may be efficient enough, there are many applications that need very large surfaces. Medical visualization (Taubin, 1995) (see Fig 18), Digital Earth representation (Goodchild, 2006; GeoWeb, 2011) and geometry images (Gu et al., 2002) are examples of such applications. Using our connectivity based method, we can significantly speed up the process of these applications.



Figure 18: A vertebra created using the proposed deformation method in (Pusch and Samavati, 2010) and the toroidal polyhedron as the base mesh.

10 CONCLUSIONS

Using connectivity maps of the proposed polyhedrons, we define an indexing method for quadrilateral and regular triangular meshes. This method is very useful for applications that need operations such as neighborhood finding or access to hierarchical structures resulted from subdivision schemes. The connectivity information of the proposed polyhedron is implicit in their connectivity maps and no extra information is required. In comparison with other common data structures such as the half-edge, our connectivity based method provides more straightforward operations for neighborhood finding and is extremely faster for applications that need neighborhood finding operations such as subdivision schemes. Our proposed method is not restricted to simple objects and it can be used for remeshing general meshes, sketch-based modeling, and editing or subdividing geometry images.

ACKNOWLEDGEMENTS

We thank David Williams-King for early assistance and discussions. This research was supported in part by the National Science and Engineering Research Council of Canada and GRAND Network of Centre of Excellence of Canada.

REFERENCES

- Bunnell, M. (2005). *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, chapter Adaptive Tessellation of Subdivision Surfaces with Displacement Mapping, pages 33–40. Addison Wesley.
- Catmull, E. and Clark, J. (1998). *Recursively generated B-spline surfaces on arbitrary topological meshes*, pages 183–188. ACM, New York, NY, USA.
- DeRose, T., Kass, M., and Truong, T. (1998). Subdivision surfaces in character animation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques, SIGGRAPH '98*, pages 85–94. ACM.
- Gargantini, I. (1982). An effective way to represent quadtrees. *Commun. ACM*, 25(12):905–910.
- GeoWeb (2011). Pyxis innovation.
- Goodchild, M. F. (2006). Discrete global grids for digital earth. In *Proceedings of 1st International Conference on Discrete Global Grids, March, 2000*.
- Gu, X., Gortler, S. J., and Hoppe, H. (2002). Geometry images. *ACM Trans. Graph.*, 21(3):355–361.

- Halstead, M., Kass, M., and DeRose, T. (1993). Efficient, fair interpolation using catmull-clark surfaces. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 35–44. ACM.
- Hoppe, H., DeRose, T., Duchamp, T., Halstead, M., Jin, H., McDonald, J., Schweitzer, J., and Stuetzle, W. (1994). Piecewise smooth surface reconstruction. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 295–302. ACM.
- Hormann, K., Lévy, B., and Sheffer, A. (2007). Mesh parameterization: theory and practice. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 1. ACM.
- Igarashi, T., Matsuoka, S., and Tanaka, H. (2007). Teddy: a sketching interface for 3d freeform design. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07. ACM.
- Kettner, L. (1998). Designing a data structure for polyhedral surfaces. In *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry*, pages 146–154. ACM.
- Loop, C. (1987). Smooth subdivision surfaces based on triangles. Department of mathematics, University of Utah.
- Losasso, F., Hoppe, H., Schaefer, S., and Warren, J. (2003). Smooth geometry images. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '03, pages 138–145. Eurographics Association.
- Maya (2011). Autodesk inc.
- Nasri, A., Karam, W. B., and Samavati, F. (2009). Sketch-based subdivision models. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling (SBIM'09)*, pages 53–60. ACM.
- Nealen, A., Igarashi, T., Sorkine, O., and Alexa, M. (2007). Fibermesh: designing freeform surfaces with 3d curves. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07. ACM.
- Nealen, A., Pett, J., Alexa, M., and Igarashi, T. (2009). Gridmesh: Fast and high quality 2d mesh generation for interactive 3d shape modeling. In *Shape Modeling and Applications, 2009. SMI 2009. IEEE International Conference on*, pages 155–162.
- Olsen, L., Samavati, F., Sousa, M., and Jorge, J. (2009). Sketch-based modeling: A survey. *Computers & Graphics*, 33:85–103.
- Peters, J. (2000). Patching catmull-clark meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 255–258.
- Praun, E. and Hoppe, H. (2003). Spherical parameterization and remeshing. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 340–349. ACM.
- Pusch, R. and Samavati, F. (2010). Local constraint-based general surface deformation. In *Proceedings of the International Conference on Shape Modeling and Applications (SMI 2010)*, pages 256–260. IEEE Computer Society.
- Samet, H. (1985). Data structures for quadtree approximation and compression. *Commun. ACM*, 28(9):973–993.
- Samet, H. (1990). *Applications of spatial data structures: computer graphics, image processing, and GIS*. Addison-Wesley.
- Samet, H. (2005). *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc.
- Sharf, A., Lewiner, T., Shamir, A., Kobbelt, L., and Cohen-Or, D. (2006). Competing fronts for coarse-to-fine surface reconstruction. In *Eurographics 2006 (Computer Graphics Forum)*, volume 25, pages 389–398, Vienna. Eurographics.
- Shiue, L.-J., Jones, I., and Peters, J. (2005). A realtime gpu subdivision kernel. *ACM Trans. Graph.*, 24:1010–1015.
- Taubin, G. (1995). A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 351–358. ACM.
- Weiler, K. (1985). Edge-based data structures for solid modeling in curved-surface environments. *Computer Graphics and Applications, IEEE*, 5(1):21–40.
- Zorin, D., Schröder, P., and Sweldens, W. (1997). 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.