Adaptive Atlas of Connectivity Maps

Ali Mahdavi-Amiri[⋆] and Faramarz Samavati [⋆⋆]

University of Calgary

Abstract. The Atlas of Connectivity Maps (ACM) is a data structure designed for semiregular meshes. These meshes can be divided into regular, grid-like patches, with vertex positions stored in a 2D array associated with each patch. Although the patches start at the same resolution, modeling objects with a variable level of detail requires adapting the patches to different resolutions and levels of detail. In this paper, we describe how to extend the ACM to support this type of adaptive subdivision. The new proposed structure for the ACM accepts patches at different resolutions connected through one-to-many attachments at the boundaries. These one-to-many attachments are handled by a linear interpolation between the boundaries or by forming a transitional quadrangulation/triangulation, which we call a *zipper*. This new structure for the ACM enables us to make the ACM more efficient by dividing the initial mesh into larger patches.

Keywords: Atlas of Connectivity Maps, Adaptive Subdivision, Data Structure

1 Introduction

Semiregular models are very common in computer graphics, appearing in subdivision and multiresolution surfaces, Digital Earth representations and many parametrization techniques. Semiregular models are made of a set of regular patches that are attached to each other and the extraordinary vertices may only be located at the corner of patches [28, 27]. These models result from applying repetitive refinements on a model with arbitrary connectivity (see Figure 1). In a semiregular model, most vertices and faces are regular, allowing one to design an efficient data structure called the Atlas of Connectivity Maps (ACM) [1]. In the ACM, the connectivity queries within each patch are handled by simple algebraic operations and, to transit from one patch to another, simple pre-calculated transformations are used. In addition, there exists a hierarchy between the coarse and refined faces of each patch that factors into the design of the ACM. Consider Figure 1, which shows patches before and after refinement. The ACM also provides simple algebraic relations to support such a hierarchical correspondence between faces and vertices, which proves to be advantageous in applications such as mesh editing and multiresolution.

[⋆] amahdavi@ucalgary.ca

[⋆⋆] samavati@ucalgary.ca

Although ACM is efficient for supporting semiregular models in which all the patches have the same resolution (Figure 1 (b)), we wish to support adaptive refinement, where different regions of the mesh can have different resolutions satisfying a geometric or a user specified condition (e.g. smoothness). The original description of the ACM considers a connectivity map for each patch of a given semiregular model and it is useful for meshes that are uniformly refined. It is not designed for adaptive scenarios. In this paper, we extend the ACM to support adaptive refinement by modifying its structure to support patches at different resolutions. To adaptively refine a connectivity map, it is split into an optimized number of new connectivity maps. Necessary transformations for traversing between the connectivity maps are also calculated for each new connectivity map. In the original description of ACM, each connectivity map is connected to four other connectivity maps and transformations between them are pre-calculated and encoded into four integer numbers. However in the adaptive case, connectivity maps can be connected to an arbitrary number of other connectivity maps and transformations are not restricted to only four forms. We extend the ACM in such a way that it can support connections to multiple connectivity maps by maintaining a list of each connectivity map's neighbors. For each neighbor, we also store additional adjacency information, from which transformations are calculated using simple algebraic relationships. Gaps between different resolutions are resolved either by a set of transitioning triangles called zippers (see Figure 1 (c)) or by placing the vertices of the higher resolution patch on the edges of the lower resolution patch using a linear interpolation.

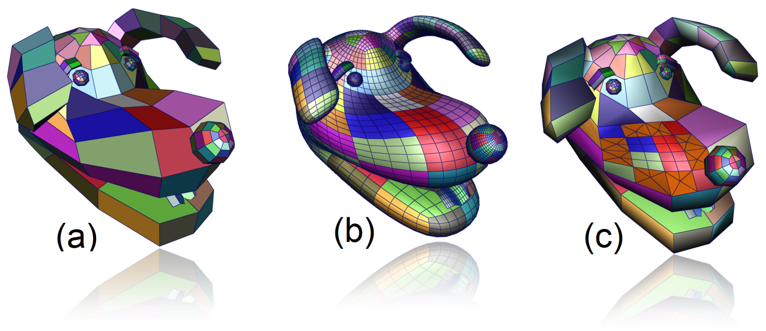


Fig. 1. (a) Coarse model. (b) A semiregular model obtained by uniformly subdividing the coarse model in (a). (c) Adaptively subdividing regions of interest. Patches at different resolutions are connected by zippers highlighted with orange faces.

With our proposed structure for the ACM, we can achieve a more compact representation of a mesh by forming larger initial connectivity maps. To do so, we provide an algorithm that can find regular quadrilateral patches. We also show that this algorithm significantly improves the performance of the ACM.

The paper is organized as follows: Related work is presented in Section 2. An overview of the ACM is described in Section 3. The extension of the ACM for adaptive subdivision is discussed in Section 4, followed by algorithms and discussions for improving the ACM by considering larger patches in Section 5. In Section 6, results and discussion are presented and we finally conclude in Section 7.

2 Related Work

Many data structures have been proposed to work with polygonal meshes [3]. Among the proposed data structures, edge-based data structures and their variations such as the half-edge structure are very common for applications that make liberal use of connectivity queries (e.g. subdivision) [11, 12]. As a result, the half-edge structure is professionally implemented in many libraries such as CGAL with the ability to support many types of subdivision schemes [4]. They are also used to support adaptive subdivision, as they can handle very local connectivity queries. In addition to subdivision, half-edges have also been extended in supporting multiresolution surfaces [13]. Although half-edges are very useful to support meshes with arbitrary connectivity, they do not benefit from the regularity of meshes and the hierarchy between faces that result after applying subdivision.

To benefit from the regularity and hierarchy of patches obtained from subdivision, some hierarchical data structures have been proposed. Quadtrees are one of the most common data structures used to support subdivision, particularly when the factor of refinement is four (e.g Loop and Catmull-Clark subdivision) [9, 10, 24, 25]. Variations on quadtrees, such as balanced quadtrees, have been proposed for adaptive subdivision in order to establish smooth transitions between patches [29]. A quadtree is balanced if any two neighboring nodes differ at most one in depth. However, since quadtrees generally require many hierarchical connections to maintain the hierarchy of faces, other data structures have been proposed that are specifically designed for subdivision. Some of these data structures are based on indexing methods specific to the type of subdivision. Shiue et al. use a 1D spiral indexing for subdividing quadrilateral and triangular meshes in [6] and use the same indexing method to employ the GPU in subdividing meshes with arbitrary topology [5]. Some patch-based methods have been also proposed for subdivision schemes in which each patch is separately stored in a 2D array [7, 8]. These data structures are very efficient when employing subdivision. However, they often restrict the initial connectivity of the mesh or specific type of subdivision.

The ACM is a data structure that supports connectivity queries on semiregular models. Similar to a patch-based data structure, the ACM stores the geometric information of each patch in a 2D array. A unique aspect of the ACM is its ability to handle global and inter-patch connectivity queries using inter-patch transformations. In addition, the ACM comprehensively handles all existing refinements used in subdivision surfaces [27, 1, 2]. However, adaptive subdivision is not supported by the ACM since the primary assumption in its design is that all patches are at the same resolution.

Several adaptive subdivision methods have been proposed. In these methods, the geometry and the connectivity of the original subdivision scheme are modified to maintain the adaptivity. Geometric modifications to vertices in a locally subdivided region are handled differently [15, 23, 16, 14]. Moreover, since cracks may be created in the transition from a coarse region to a smooth region, the connectivity has to be modified. While directly connecting the newly inserted vertices to existing vertices is a solution [15], more sophisticated approaches have been taken, such as red-green and incremental adaptive subdivision. In red-green algorithm, faces with one crack are bisected (green triangulation) while faces with more than one crack per edge are split into four (red triangulation) [21]. Under the incremental adaptive subdivision, a one ring neighborhood is introduced to transition from a smooth region to a coarse region, and in the process avoids high valence vertices and skinny triangles [18–20]. In addition, Pannozo and Puppo designed a method for adaptive Catmull-Clark subdivision by limiting the transitional polygons to pentagons and triangles [22]. The main challenge to creating an efficient data structure for adaptive subdivision is how to handle the change in connectivity. We provide two solutions to handle connectivity modifications by providing transitional domains (zippers) between coarse and fine connectivity maps, or linearly interpolating the boundary edge between a high and low resolution connectivity map.

3 Overview of the ACM

In this section, we provide an overview of the ACM [1]. As mentioned earlier, the ACM is designed for semiregular meshes, which are made of connected regular patches resulting from a regular refinement (Figure 2). Each patch i is assigned to a 2D domain for which a 2D coordinate system is considered. This 2D domain along with its connectivity information is called a *connectivity map* ($CM(i)$). In an ACM, an array of connectivity maps CM is stored for a semiregular model with M patches in which $CM(i)$ $(0 \leq i < M)$ refers to the ith patch of the model. The coordinate system of $CM(i)$ is used to index vertices within a 2D location array that records the 3D positions of the vertices through the resolutions. Connectivity queries for internal vertices of $CM(i)$ are handled by neighborhood vectors that are added to the index of a vertex (Figure 2 (c)). As refinements are applied, the connectivity information of the model continues to be maintained by the connectivity maps. Based on the type of refinement, hierarchical relationships are defined for vertices and faces that are useful in applications such as mesh editing and multiresolution.

The connectivity information between $CM(i)$ and its neighboring connectivity maps, denoted by $CM(N_j(i))$, should be recorded as well in order to support connectivity queries outside of each patch. As a result, the of neighbors of $CM(i)$ are stored in an array, say *neighbors*. Inter-patch queries between $CM(i)$ and $CM(N_j(i))$ are handled by a set of transformations that map the coordinate

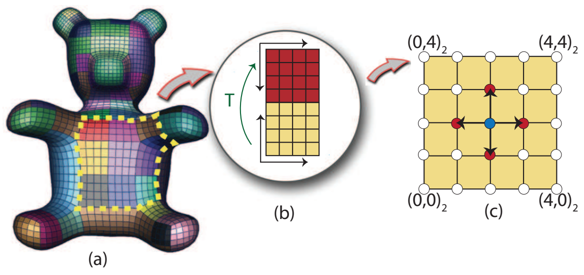
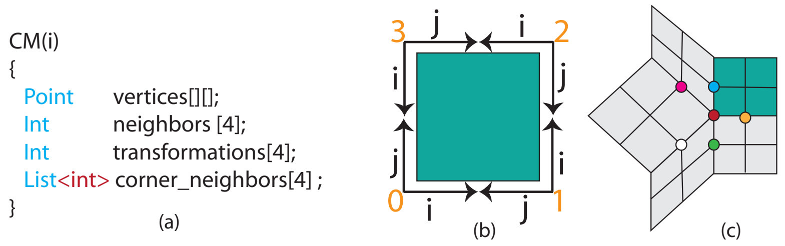


Fig. 2. (a) A semiregular mesh. (b) For each patch, a 2D domain (connectivity map) with a coordinate system is assigned. To move from one patch to another, a transformation between the coordinate systems of two adjacent connectivity maps can be used. (c) To index vertices, the coordinate system defined for each patch is used. The subscript of the index refers to the resolution of the refinement. Neighborhood vectors are used to obtain neighbors of internal vertices.

system of $CM(i)$ to the coordinate system of $CM(N_j(i))$. To simplify these transformations, they are encoded as integer numbers (Figure 3 (b)). Consequently, $CM(i)$ has a 2D array of 3D points storing the locations of vertices, a 1D array recording $CM(N_j(i))$ and an array storing the inter-patch transformation codes (see Figure 3 (a)). Since the corner vertices of $CM(i)$ may be irregular/extraordinary, a separate structure is used to store corner neighbors of $CM(i)$. This structure - a list for each corner - stores the connectivity maps that are attached to each corner of $CM(i)$.



```
CM(i)
{
  Point       vertices[][];
  Int         neighbors [4];
  Int         transformations[4];
  List<int> corner_neighbors[4] ;
}
```

Fig. 3. (a) The elements that are stored for each connectivity map. (b) Inter-patch transformations are encoded as integer numbers. (c) A corner (red vertex) can be extraordinary. Connectivity maps attached to corners (in this case, two of them) are saved in corner_neighbors.

The ACM can support a variety of refinements for applications such as subdivision surfaces, Digital Earth frameworks and multiresolution [1, 2].

4 Adaptive ACM

Although the ACM is an efficient data structure for semiregular models, adaptive subdivision cannot be supported within the original formulation for the ACM. One simple approach to make the ACM usable for adaptive subdivision is to consider the possibility of allowing patches to exist at different resolutions (see Figure 4 (a)). To do so, we only need to add an integer value recording the resolution of each patch. However, this does not support the adaptive refinement of faces within a patch (see Figure 4 (b)).

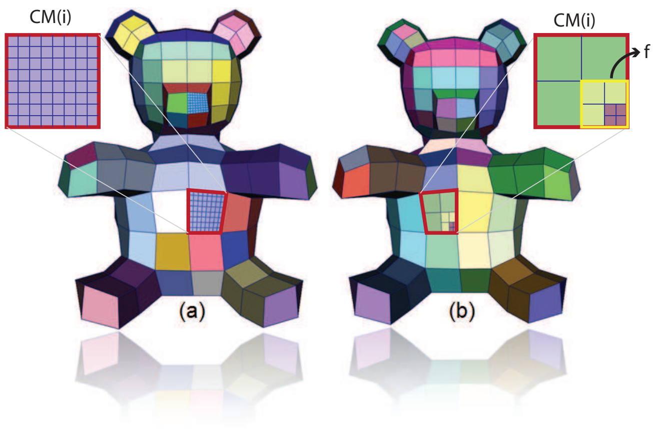


Fig. 4. (a) Adaptive subdivision with patches at different resolutions. (b) Adaptive subdivision with faces in a patch at different resolutions.

To support such a case, we need to enable the possibility of breaking a given connectivity map $CM(i)$ into a set of smaller connectivity maps. Let $f \in CM(i)$ be the face selected for adaptive subdivision (see Figure 4 (b)). A new connectivity map with a new coordinate systems should be assigned to f (see Figure 5). The question is what should happen to the other faces of $CM(i)$. As illustrated in Figure 5 (b), one simple solution is to generate individual connectivity maps for every face in $CM(i)$. However, it is more efficient to divide these faces between larger connectivity map blocks (see Figure 5 (c)). To benefit from the regularity of $CM(i)$, we can categorize all possible cases of this division based on the position of $f \in CM(i)$. As shown in Figure 6, three possible cases exist when $CM(i)$ is divided to blocks. When f is located at the corner of, the boundary of, or internal to $CM(i)$, we split $CM(i)$ into three, four, or five connectivity maps respectively. Notice that while the dividing patterns are not unique, the number of blocks in each case is minimal. Coordinate systems aligned with that of $CM(i)$ are assigned to each new connectivity map (Figure 6).

Dividing a connectivity map into these blocks requires that we support one-to-many attachments between connectivity maps (see Figure 7 (a)). These one-

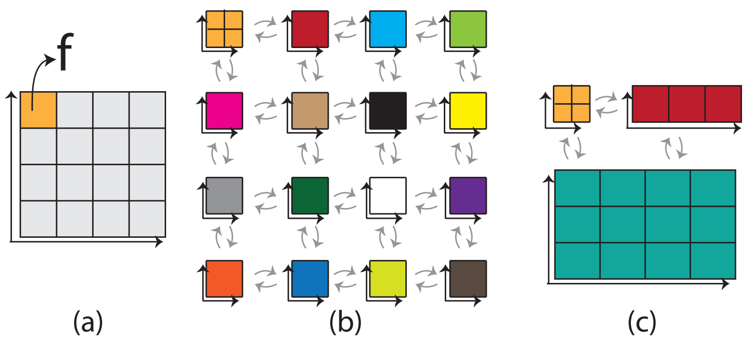


Fig. 5. (a) Face $f \in CM(i)$ is supposed to be subdivided, therefore $CM(i)$ has to be split. (b) Simple and inefficient solution for splitting $CM(i)$ by assigning a connectivity map to each face. (c) A more efficient split of $CM(i)$ that benefits from the regularity of $CM(i)$.

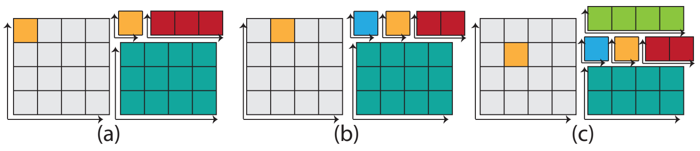
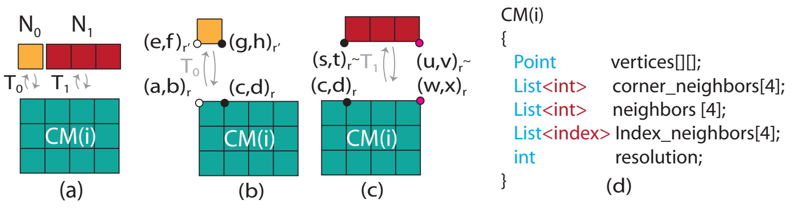


Fig. 6. $CM(i)$ is split into three, four, or five connectivity maps when f is located at (a) a corner of, (b) a boundary of, or (c) internal to $CM(i)$.

to-many attachments are not supported in the original ACM, as connectivity maps are only connected to one neighbor along each boundary edge. Here, we show how to extend the ACM to support such one-to-many attachments at boundary edges.

To store the 3D locations of vertices, we still use a 2D array. A list recording the connectivity information at the corners is also still sufficient. Hence, we only need to change the neighbors array, which we accomplish by changing it to an array of lists of neighbors (Figure 7). In the original ACM, all possible transformations are encoded in four integer numbers. However, due to the existence of one-to-many attachments, it may be the case that more complex transformations map a connectivity map to one of its side neighbors. As a result, a new method of storing transformations has to be used that is flexible enough to include any type of transformation. To do so, the range of vertices connected to $CM(i)$ along a boundary edge is stored. To capture this, we store the indices of each neighbor's first and last vertices along the shared boundary edge. For example, in Figure 7, $CM(i)$ is connected to its neighbor (N_0) at indices $(e, f)_{\acute{r}}$ and $(g, h)_{\acute{r}}$ (Figure 7 (b)). These two indices are both stored for $CM(i)$. Using these two indices and the relative positions of these two connectivity maps, transformation T_0 that maps the coordinate system of $CM(i)$ to N_0 is calculated. Similarly, $(s, t)_{\tilde{r}}$ and $(u, v)_{\tilde{r}}$ are used to calculate T_1 (Figure 7 (c)). Consequently, we do not need to explicitly store transformations as they can be found from the indices. Therefore, our adaptive ACM has five components, as listed in Figure 7 (d).



```
CM(i)
{
  Point          vertices[][];
  List<int>      corner_neighbors[4];
  List<int>      neighbors [4];
  List<index>    Index_neighbors[4];
  int            resolution;
}
```

Fig. 7. (a) $CM(i)$ is connected to two connectivity maps N_0 and N_1. Transformations T_0 and T_1 are necessary to traverse these neighbors. (b), (c) The transformations that map a vertex in $CM(i)$ to its neighbors are determined by storing necessary indices from the neighboring connectivity maps. (d) The new structure of the ACM.

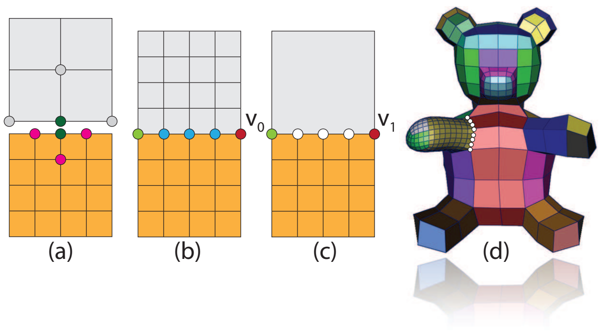


Fig. 8. (a) There exists ambiguity in determining the neighbors of a vertex when it is shared between a coarse and fine connectivity map. (b) The geometry of the blue vertices is obtained using regular refinement filters when two patches have the same resolution. (c) Two connectivity maps at two different resolutions. The white vertices at the boundary are linearly interpolated. (d) An example of refinement where vertices along the boundary between two connectivity maps at different resolutions are linearly interpolated.

When two neighboring patches have different resolutions, this creates an ambiguity in the definition of a vertex's neighborhood. For instance, the green vertex in Figure 8 (a) has two different sets of neighbors depending on whether we consider it from the high or low resolution patch. Hence, this ambiguity in the neighborhood definition should be somehow addressed based on the needs of the application. Using the adaptive ACM, connectivity information between patches is accessible even when they are at two different resolutions. Therefore, any adjacency queries or geometric modifications on the vertices needed to per-form adaptive subdivision remain possible. For example, a possible method for performing adaptive subdivision is to smoothly subdivide patches to a desired resolution (Figure 8 (b)) and linearly interpolate the vertices on boundary edges shared between high and low resolution patches to avoid cracks (Figure 8 (c)). In this case, the geometry of the ith boundary vertex on the high resolution patch is calculated by $\frac{i}{n}(v_1 - v_0) + v_0$ where n is the number of vertices on the high resolution patch and v_0 and v_1 are vertices on the low resolution patch. Figure 8 (d) illustrates an example where a mesh is subdivided using Catmull-Clark subdivision and the edges between two patches at two different resolutions are linearly interpolated.

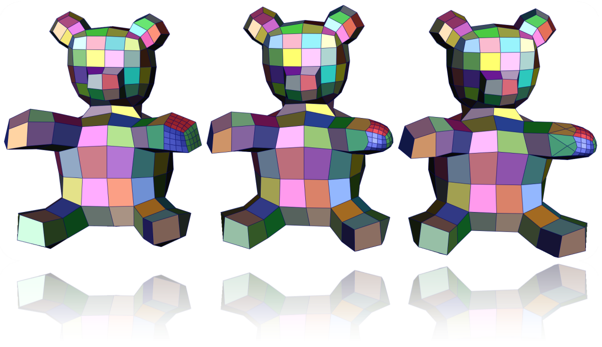


Fig. 9. (a) Linear subdivision along the boundaries of the faces does not smooth the hand of the Teddy. (b) Faces can be smoothly subdivided. Faces at the coarse resolution can be rendered as polygons. (c) Zippers can be used to connect smooth faces to coarse faces.

Although linearly subdividing patches is useful to insert more vertices and create low scale features (see Figure 15), in some cases it may produce unwanted artifacts, as illustrated in Figure 9 (a). An example of such a scenario is in Digital Earth frameworks, in which patches might have different resolutions and vertices are projected to the sphere. As a result, it would be desirable to be able to smoothly subdivide or modify the geometry of all vertices in a patch. To avoid artifacts, we can can render the low resolution faces as polygons rather than quads, with the edges of the polygons aligned with the vertices of the high resolution quads (see Figure 9 (b)). To obtain a mesh with higher quality, the high resolution patches can be connected to low resolution patches using transitional quadrangulations or triangulations called *zippers* (see Figure 9 (c)) [26].

These zippers are applicable for triangles and quads. Consider two connectivity maps $CM(i)$ and $CM(j)$ with m and n faces ($m > n$), respectively, along the common boundary. To connect $CM(i)$ to $CM(j)$, one simple solution is to insert a vertex in the adjacent triangle (or quad) strip of the lower resolution connectivity map or in the gap between these two connectivity maps. We can then simply connect all vertices of $CM(i)$ to $CM(j)$ (Figure 10 (a)). However, this is not a good solution due to the existence of high valence vertices and skinny triangles. A better solution is to insert $\frac{m}{n}$ vertices that connect $\frac{m}{n}$ number of faces in $CM(i)$ to a face in $CM(j)$ (Figure 10 (b)). We can achieve an even better transition by inserting extra vertices and requadrangulating or retriangulating the zipper as shown in Figure 9 (c) and Figure 10 (c). Figure 11 illustrates an example of zippers on the surface of the Earth. Note that these zippers can be formed on the fly using a simple algorithm, therefore an additional data structure is not needed.

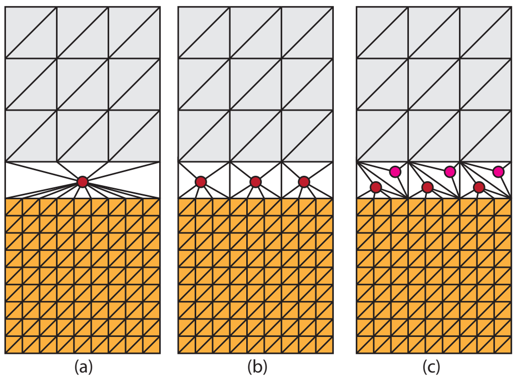


Fig. 10. (a) A zipper with one high valence vertex produces poor triangles (b) Adding more vertices to the zippers produces better triangulations. (c) It is possible to add vertices and retriangulate the zipper in (b) to achieve a better transition from a high resolution connectivity map to a low resolution one.

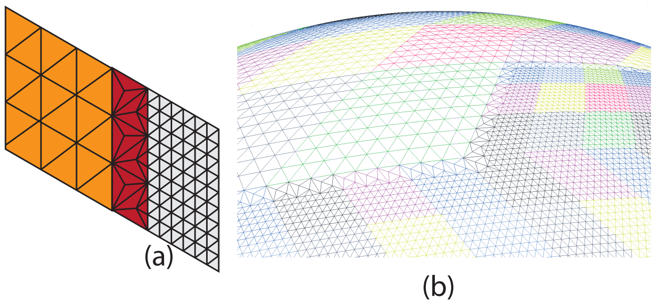


Fig. 11. (a) Zipper between two triangular connectivity maps. (b) Connectivity maps and zippers on a portion of the Earth.

5 Compact ACM

In the original ACM, a connectivity map is assigned to each quad at the coarsest resolution (i.e. control mesh) where each connectivity map has four neighbors (See Figure 12 (a)). The boundary vertices of each connectivity map are duplicated at each neighbor to obtain a separate, simple quadrilateral domain for each connectivity map. By merging two connectivity maps into one, such repetitions can be discarded and a more efficient ACM is obtained (See Figure 12 (c)). Using the adaptive ACM, connectivity maps of any size ($m \times n$) are acceptable, and they can have multiple neighbors (See Figure 12 (b)). Therefore, in an adaptive ACM, it is possible to combine several initial connectivity maps into one connectivity map as long as they form a quadrilateral domain.

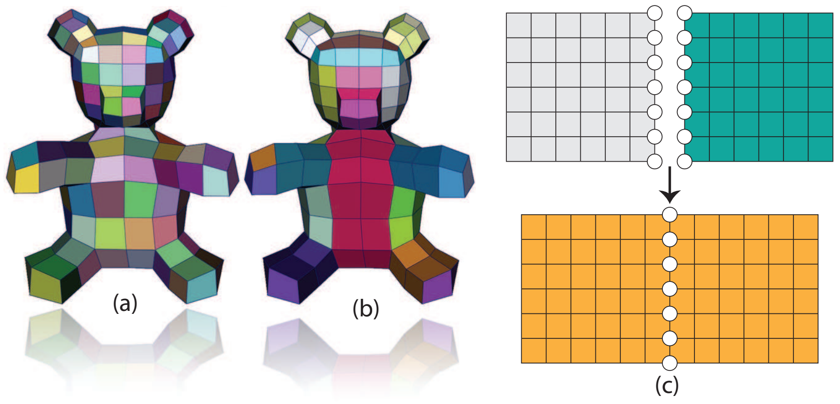


Fig. 12. (a) The original ACM, where a connectivity map is assigned to each individual quad at the first resolution. (b) Adaptive ACM supports larger initial connectivity maps. (c) When two connectivity maps are combined, duplicated vertices are discarded.

We systematize our approach to this problem by proposing a simple algorithm that has three main functions: Pair, Union, and CleanUp (see Algorithm 1). Our algorithm takes a list of connectivity maps (CM) as input and outputs a list of combined connectivity maps denoted by l. In the Pair function, two neighboring connectivity maps $CM(i), CM(j) \in CM$ are combined and added to l and discarded from CM (see Figure 13 (a)). This process continues until no connectivity map with a neighbor in CM exists, at which time the isolated connectivity maps (connectivity maps that do not have any neighbor in Q) are added to l. The Union function combines neighboring connectivity maps in l that share the same number of vertices at their common boundaries (Figure 13 (b)). In CleanUp, connectivity maps with a small dimension (usually 2×2, $m \times 1$, or $1 \times m$) are divided into a number of connectivity maps that can be combined with their neighbors (Figure 13 (c)). Union and CleanUp will be called interchangeably until no more modifications occur. Note that since connectivity maps each need a coordinate system, after combining two connectivity maps, a coordinate system aligned with the coordinate system of one of the connectivity maps is chosen for the new connectivity map.

```
Data: Mesh $M$ given by a list of connectivity maps denoted by $CM$
Result: List $l$ of combined Connectivity Maps.
Pair();
while There are modifications do
    Union();
    CleanUp();
end
```

Algorithm 1: An algorithm to create the ACM by combining patches into connectivity map blocks.

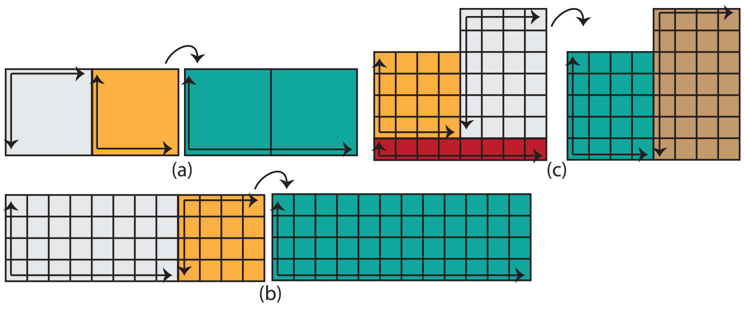


Fig. 13. (a) Pair, (b) Union, and (c) CleanUp.

Table 1 lists the number of connectivity maps for several models when the algorithm is and is not applied. As apparent, this algorithm significantly reduces the number of connectivity maps. As a result, there are fewer repetitive vertices and less redundancy in the resulting ACM. For instance, after three applications of Catmull-Clark subdivision on the Big Guy model, about 30000 redundant vertices are removed if the connectivity maps are combined using the proposed algorithm. Figure 14 illustrates the models that are used in Table 1. Consequently, we can conclude that, using the adaptive ACM, the performance of the ACM is significantly improved and adaptive subdivision is also supported.

Table 1. Number of connectivity maps in the original ACM and adaptive ACM. The reduction in the number of connectivity maps is apparent.

Models	ACM	AACM
Cube	6	3
Teddy	272	45
Big Guy	1450	191
Monster Frog	1292	187

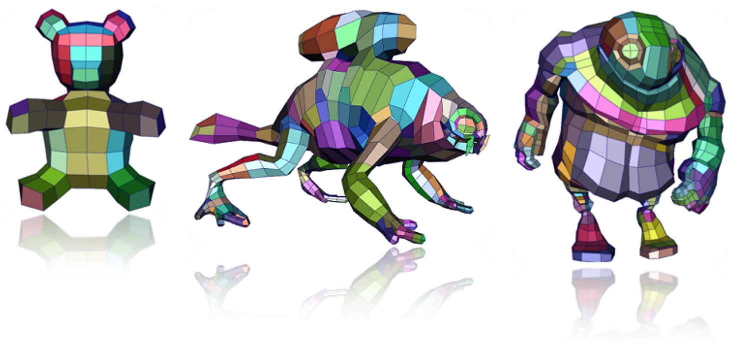


Fig. 14. (a) Teddy, (b) Monster Frog, (c) Big Guy stored in an adaptive ACM.

6 Comparisons and Results

Using the adaptive ACM, we no longer need to uniformly refine an entire model to add further details, and can efficiently model a wider variety of objects. For example, as illustrated in Figure 15, we can add local low scale details such as engravings. Using the algorithm for compacting the connectivity maps, we can also join together connectivity maps created from splitting the connectivity maps during adaptive refinement. For instance, as illustrated in Figure 15 (a), adaptively subdivided faces on the neck of the Big Guy from two different connectivity maps (blue and purple) are joined into a single connectivity map (red).

The ACM performs efficiently in comparison to other data structures employed for adaptive subdivision. Consider the case illustrated in Figure 5, in which face $f \in CM(i)$ is subdivided. $CM(i)$ is an $N \times N$ patch obtained from subdividing a coarse face C at the first resolution. We compare the memory consumption of the adaptive ACM with a half-edge structure and a quadtree; two standard data structures used to support adaptive subdivision. We also discuss the efficiency of zippers in comparison to red-green rules and incremental adaptive subdivision.

In the case illustrated in Figure 16, three connectivity maps are stored with dimensions $((N+1) \times N)$, $(2 \times N)$, and (3×3). The connections between these connectivity maps are also stored in an adaptive ACM through 12 additional indices. By contrast, a half-edge data structure needs to store $(\approx (5 \times N \times N) + 12)$ for half-edge objects as well as $((N+1) \times (N+1) + 5)$ vertices and $(N \times N + 3)$ faces. In adaptive ACM, $(N+2)$ more vertices are needed, though we avoid $(\approx (5 \times N \times N) + 12)$ pointers for edges and $(N \times N + 3)$ faces. Hence, by respecting the regularity of the patches in an adaptive ACM, a large amount of memory can be saved.

It is possible to use quadtrees when a 1-to-4 refinement is used. As a result, a quadtree is often used to store $N \times N$ patch when $N = 2^M$. In this case,

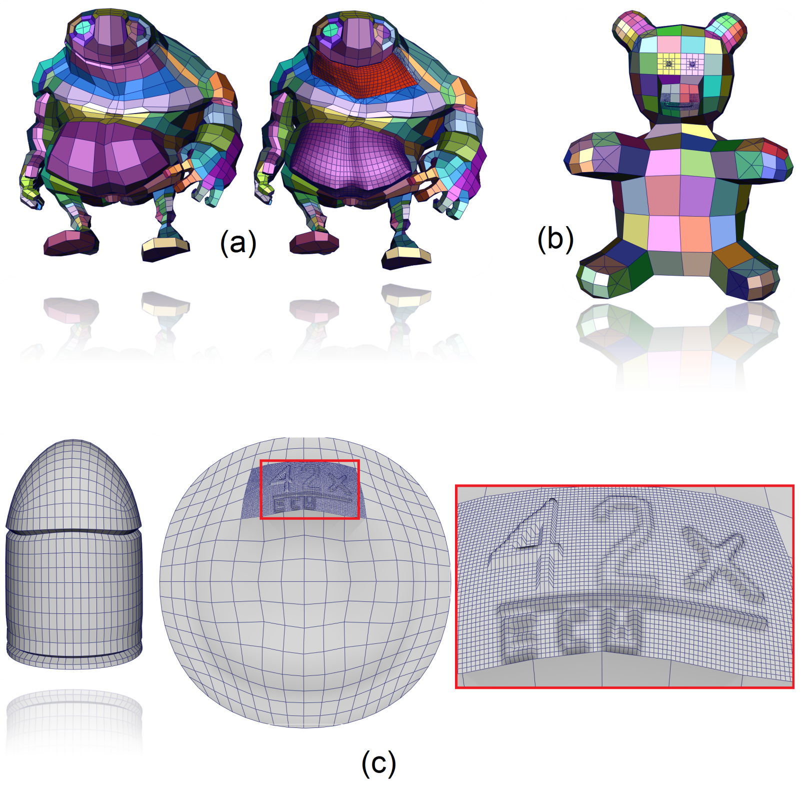


Fig. 15. (a) The Big Guy is adaptively subdivided on its belly and neck. Fine faces resulting from adaptively subdividing two different connectivity maps on its neck (purple and blue) are merged into a single connectivity map (red). (b) The hands and legs of Teddy are adaptively smoothed and some low scale details are added to the lips and the eyes of using adaptive refinement and local vertex manipulation. (c) The caliber of the Bullet is engraved on its back using adaptive refinement.

$4\left(\frac{1-4^M}{1-4}\right)$ pointers are needed to identify the resolutions of the patches. For instance, for the patch illustrated in Figure 16, 24 pointers are needed. However, such pointers are unnecessary in adaptive ACM and the resolution of each patch is directly accessible. In addition, a more compact adaptive ACM can be formed by grouping first resolution faces with regular connectivity into a single connectivity map, removing some redundant boundary vertices. Quadtrees, by contrast, require a unique node be assigned to each face of the first resolution.

To connect a high resolution patch to a lower resolution patch, we use zippers. Zippers are the quadrangulation or triangulation of the lower resolution patch (or the gap between them) to avoid cracks. Red-green rules and incremental adaptive subdivision are two other methods that may be used to connect high resolution patches to lower ones and establish a progressive resolution change among

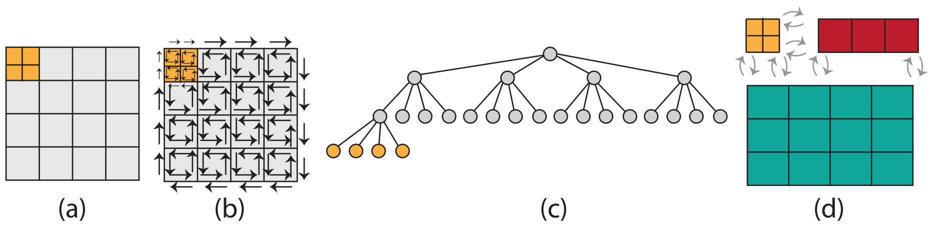


Fig. 16. (a) $f \in CM(i)$ is subdivided. (b) Half-edge pointers needed for edges of the patch in (a). (c) Pointers needed to store the patch in (a). (d) Adaptive ACM and its essential connectivity information.

faces. Although we can connect the connectivity maps using red-green rules or incremental adaptive subdivision, we use zippers to avoid producing irregularities. Consider a coarse quadrilateral face subdivided two times. Both red-green rules and incremental adaptive subdivision propagate the face splits through the neighboring faces by creating irregular vertices and non-quadrilateral (triangular) faces, which are undesirable in a quadrilateral mesh. Using zippers, only four extraordinary vertices and four triangles are formed, in comparison to the 40 extraordinary vertices and 36 triangles of red-green rules and incremental adaptive subdivision. As a result, the adaptive ACM preserves the regularity better than red-green rules and incremental adaptive subdivision. Figure 17 illustrates this comparison. The zipper used in this example is the same zipper shown in Figure 9 (c).

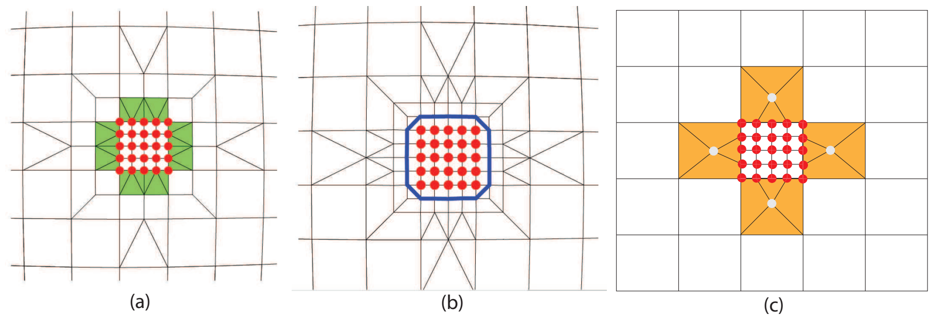


Fig. 17. (a) Red-green rule. (b) Incremental adaptive subdivision. (c) Zippers in the adaptive ACM. Images (a) and (b) are taken from [20, 19].

Consequently, the adaptive ACM performs well when adaptively subdividing models in which the patches are mostly regular. However, if the subdivision is very local and the object exhibits irregular behaviors in the connectivity, the ACM does not provide a significant advantage over data structures that efficiently support arbitrary local modifications, such as the half-edge structure.

As a result, we do not claim that adaptive ACM is the best data structure for adaptive subdivision in all cases, but that it is advantageous for applications in which there exists a relatively strong notion of regularity in the connectivity of the object.

7 Conclusion and Future Work

By extending the ACM, adaptive subdivision is supported and a more efficient ACM is achieved. We provide two solutions to support adaptive subdivision: one that linearly interpolates the edges at the boundary and one that uses transitional patches (zippers) to connect connectivity maps at two different resolutions. An algorithm is also proposed to obtain a compact set of regular patches from a given mesh. This algorithm performs fairly well and can discard redundant vertices found on the boundaries of adjacent patches. Although the algorithm provided in this paper significantly improves the ACM, future work may look into discovering a more optimized algorithm as our proposed algorithm does not guarantee the best possible set of regular patches in a given mesh. The adaptive ACM can be extended and improved in several other possible directions. For example, one possible problem is to determine the optimal zippers for a specific type of face, refinement, or application.

References

1. Mahdavi-Amiri, A., Samavati, F.: Atlas of Connectivity Maps. J. Computers & Graphics. 1–11 (2014).
2. Mahdavi-Amiri, A., Harrison, E., Samavati, F.: Hexagonal connectivity maps for Digital Earth. International Journal of Digital Earth. 1–11 (2014).
3. De Floriani, L., Magillo, P.: Multiresolution Mesh Representation: Models and Data Structures. Tutorials on Multiresolution in Geometric Modelling. 363–418 (2002).
4. Kettner, L.: Halfedge Data Structures : CGAL User and Reference Manual. (2013).
5. Shiue, L., Jones, I., Peters, J.: A pattern-based data structure for manipulating meshes with regular regions. ACM Trans. Graph. 363–418 (2005).
6. Shiue, L., Peters, J.: A realtime GPU subdivision kernel. Graphics Interfaces 2005. 153–160 (2005).
7. Peters, J.: Patching Catmull-Clark meshes. SIGGRAPH 00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques. 255–258 (2000).
8. Bunnell, M.: Adaptive tessellation of subdivision surfaces with displacement mapping: GPU Gems 2.(2005).
9. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann Publishers Inc. (2005).
10. Zorin, D., Schröder, P., Sweldens, W.: Interactive multiresolution mesh editing. SIGGRAPH 97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques. 259–268 (1997).
11. Weiler, K.: Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments. Computer Graphics and Applications, IEEE. 21–40 (1985).
12. Kettner, L.: Designing a data structure for polyhedral surfaces. SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry. 146–154 (1998).

13. Kraemer, P., Cazier, D., Bechmann, D.: Extension of half-edges for the representation of multiresolution subdivision surfaces. The Visual Computer. 149–163 (2009).
14. Nießner, M., Loop, C., Meyer, M., Derose, T.:Feature-adaptive GPU Rendering of Catmull-Clark Subdivision Surfaces. ACM Trans. Graph. 6:1–6:11 (2012).
15. Müller, H., Jaeschke, R.: Adaptive Subdivision Curves and Surfaces. In Proceedings of the Computer Graphics International 1998 (CGI '98). 6:1–6:11 (1998).
16. Kobbelt, L.: $\sqrt{3}$ Subdivision. Proceedings of the 27th annual conference on Computer graphics and interactive techniques. 103–112 (2000).
17. Guiqing, L., Weiyin, M., Hujun, B.: $\sqrt{2}$ Subdivision for quadrilateral meshes. 180–198 (2004).
18. Pakdel, H., Samavati, F.: Incremental subdivision for triangle meshes: International Journal of Computational Science and Engineering. 80–92 (2007).
19. Pakdel, H., Samavati, F.: Incremental Catmull-Clark subdivision: Fifth International Conference on 3-D Digital Imaging and Modeling 2005 (3DIM 2005). 95–102 (2005).
20. Pakdel, H., Samavati, F.: Incremental Adaptive Loop Subdivision: Computational Science and Its Applications - ICCSA 2004. 237–246 (2004).
21. Andrew, R.B., Sherman, A.H, Weiser, A.: Some Refinement Algorithms And Data Structures For Regular Local Mesh Refinement: Scientific Computing. 3–17 (1983).
22. Panozzo, D., Puppo, E.: Implicit Hierarchical Quad-Dominant Meshes: Comput. Graph. Forum. 1617–1629 (2011).
23. Li, G., Ma, W., Bao, H.: $\sqrt{2}$ subdivision for quadrilateral meshes: Vis. Comput. 180–198 (2004).
24. Loop, C.: Smooth Subdivision Surfaces Based on Triangles: Masters Thesis, University of Utah, Department of Mathematics. (1987)
25. Catmull. E, Clark. J: Recursively generated B-spline surfaces on arbitrary topological meshes: Computer-Aided Design. 350–355 (1978)
26. Heredia, V. M, Urrutia. J: On Convex Quadrangulations of Point Sets on the Plane. Lecture Notes in Computer Science, Springer. 38–46 (2005)
27. Mahdavi-Amiri, Ali: ACM: Atlas of Connectivity Maps : PhD Thesis, University of Calgary, Department of Computer Science. (2015)
28. Bommes, D., Lévy, B., Pietroni, N., Puppo, E., Silva, C., Tarini, M., Zorin, D.: State of the Art in Quad Meshing: Eurographics STARS. (2012)
29. Har-Peled, S.: Geometric approximation algorithms (Vol 173): Providence: American mathematical society. (2011)