A Pattern-based Data Structure for Manipulating Meshes with Regular Regions

Le-Jeng Shiue Jörg Peters

Computer and Information Science and Engineering
University of Florida

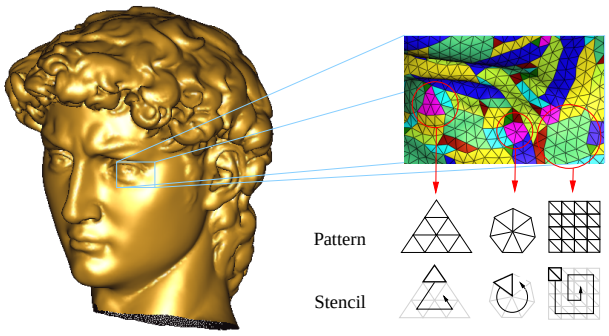


Figure 1: Quasi-regularity in a scanned and remeshed data set. The eye detail shows existing patterns that can be generated by enumeration of simple stencils in increasing orbits.

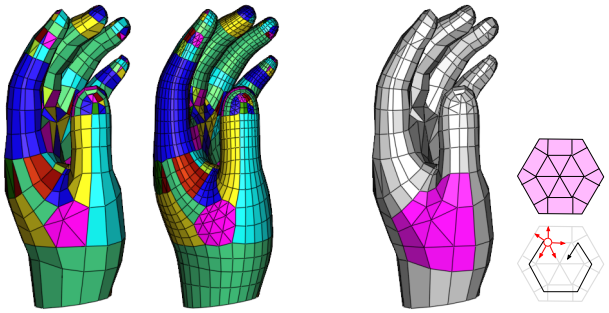


Figure 2: (a) Hand model from mesh remeshing [1] where clustering can be adjusted for subdivision (b), or to maximize cluster size (c).

Abstract

Automatically generated or laser-scanned surfaces typically exhibit large clusters with a uniform pattern. To take advantage of the regularity within clusters and still be able to edit without decompression, we developed a two-level data structure that uses an enumeration by orbits and an individually adjustable stencil to flexibly describe connectivity. The structure is concise for storing mesh connectivity; efficient for random access, interactive editing, and recursive refinement; and it is flexible by supporting a large assortment of connectity patterns and subdivision schemes.

Key words: mesh data structure, mesh refinement, subdivision surfaces, mesh compression.

1 Introduction

Any polyhedral mesh can be segmented into (worst case single element) clusters of elements (nodes or facets) so that within each cluster, every element has the same local neighborhood graph or *stencil* (see Figure 1, *right*). If the cluster elements can all be visited on a path following nested orbits, the mesh is called *quasi-regular*. Quasi-regular meshes are a strict superset of meshes with subdivision connectivity since they can have more complex stencils. Meshes with such clusters are obtained by region growing from scanned datasets (Figure 1), by remeshing (Figure 2), by refining subdivision surfaces (Figure 17), and in interactively-created models (Figure 19). Computing or defining an optimal decomposition into quasi-regular clusters is not the focus of this paper, because optimal decomposition is not a precondition for substantial savings when using the new data structure in place of a general polyhedral data structure. For example, for meshes arising from subdivision, the savings are proportional to the exponentially growing number of cluster elements.

Based on the quasi-regularity, This paper develops a *concise and efficient*, yet easily *editable and refinable* mesh data structure for modeling and rendering. The *q*uasi-*r*egular *e*numeration *g*raph structure (short Qreg) implements two layers of storing, access and manipulation: one stores quasi-regular clusters, and one maintains a general structure for visiting clusters. Qreg has three key components:

- enumeration by orbits,
- modifiable cluster stencils, and
- corner connectors to access clusters.

The *enumeration by orbits* supports node groupimg such as composite clusters and *n*-gon clusters. These

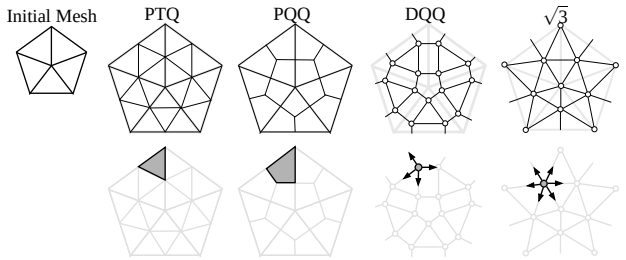


Figure 3: Refinement schemes applied to a triangle fan: primal triangle quadrisection (PTQ), primal quadrilateral quadrisection (PQQ), dual quadrilateral quadrisection (DQQ) and $\sqrt{3}$ refinement. The second row shows the stencils for the refined mesh.

groupings are not available in existing patch-based data structures; existing patch-based data structures can only enumerate nodes on regular rectangle or triangle grids. The enumeration by orbits also encodes the orientation and simplifies access across cluster boundaries. *Cluster stencils* moved along orbits implicitly define the quasi-regular connectivity within each cluster. They efficiently support various refinement patterns, including primal, dual and $\sqrt{3}$ subdivisions (see Figures 3). Uniform multiresolution is supported by an LOD hierarchy per cluster. A kd-tree-like adaptive refinement can be built with the joining or splitting of clusters. The *corner connectors* are a variant of edge-based data structure. They represent a facet corner's neighborhood via four adjacency links, independent of valence of the vertex associated with the corner, and support general structured meshes and connectivity changing operations such as Euler operations and genus-change. The corner connectors are chosen in lieu of other general structures, since they nicely complement the enumeration by orbits for accessing nodes of adjacent clusters; and since they support T-corners between clusters of unmatched patterns or clusters of different resolution.

The resulting Qreg combines direct access with compact storage and efficient recursive refinement.

1.1 Related Work and Background

We focus on meshes representing orientable 2-manifolds: every interior point has a neighborhood homomorphic to a 2D disk (to a half-disk on the mesh boundary) and a global orientation can be assigned.

Edge-based structures able to represent general polyhedra include the winged-edge structure [2], the halfedge structure [22] and the quad-edge structure [7]. Edge-based data structures are particularly efficient for local reconfiguration of the mesh since adding or deleting edges amount to changing adjacency pointers. For regular refinement, indirect pointer-based access is unnecessarily general and information about the initial mesh is lost after refinement, unless extra data is associated with the (half- or quad-) edges. Attributes attached to the edges end up dispersed in memory. A variant of the edge-based data structure, called corner-table, is used in [17] to encode the compressed triangle mesh. Each corner in corner-table is a scaled-down halfedge, which only registers the next and the previous corner within the triangle. Corner-based structures are illustrated in [10]. The scalability of the edge-based data structure is reported in [3], which shows the basic trade-off between access speed and the complexity of the connectivity structures.

Patch-based structures [16, 15, 19] rely on a static domain topology and employ regular arrays, akin to polynomial patches. The arrays *implicitly* represent the uniform, regular connectivity, e.g. the tensor-product arrangement of Catmull-Clark subdivision [4]. They are particularly efficient for recursive refinement of the mesh since the refinement amounts to scaling the arrays. Attributes are stored compactly in the same order as array nodes. While the implicit connectivity is time and space efficient, it is restrictive in a user-interactive environment, and certain refinement patterns do not fit this structure well (e.g. Doo-Sabin subdivision [5]). Patch-based structures support uniform level-of-detail (LOD). Data structures for LOD meshes are studied in [6].

Subdivision algorithms (see e.g. [21]), recursively refine coarse meshes as in Figure 17 and generate ever closer approximations to a smooth surface. Subdivision algorithms, in particular, generate quasi-regular regions. As illustrated in Figure 3, primal quadrilateral quadrisection [4] and primal triangle quadrisection [14] schemes refine by replicating a facet stencil, while dual quadrilateral quadrisection [5] and $\sqrt{3}$-triangulation [12] schemes refine by repeating a vertex stencil. Some subdivisions combine several stencils [20, 13].

Mesh compression algorithms usually encode the mesh in a non-editable format to obtain the maximum compression [18, 9] or the progressive representation [8]. The connectivity regularity (in the form of the repeated operations or regular traversal) is the key of the compressed meshes.

2 The Qreg Data Structure

To take advantage of local regularity, Qreg contains a layer that *implicitly* enumerates and manipulates the elements within a cluster, and a general structure that accesses and connects clusters (Figure 4). In analogy to topological concepts, we refer to the two levels as charts and atlas.

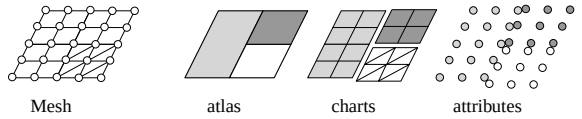


Figure 4: Qreg represents a mesh via atlas, charts and attributes.

2.1 Chart: enumeration by orbits

Each chart represents a cluster of enumerated elements. The challenge is to efficiently and flexibly enumerate the elements of the cluster without introducing additional structures and to efficiently collect the local neighborhood of an element. We address both issues by requiring that any cluster satisfy the **spiral constraint**: all elements lie on a (Hamiltonian) path that starts at a corner and spirals inward in counterclockwise order. As illustrated in Figure 3, elements can be mesh nodes or mesh facets. Figure 5 gives examples satisfying and failing the constraint.

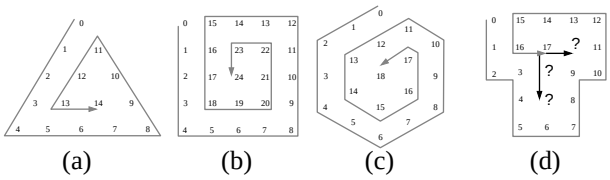


Figure 5: (a-c) regular n-gons satisfying the spiral constraint. The $\mathbb{T}$ cluster in (d) does not satisfy the constraint.

Regular *n*-gons, composite rectangles or fans, and certain annuli meet the constraint. The enumeration by orbits has two advantages over row-major or column-major indexing: (1) it can enumerate *n*-sided structures and (2) it encodes orientation so that it is easy to collect neighborhoods straddling the boundary between two charts.

2.2 Chart: neighborhood stencils

A stencil that travels along the orbital path establishes the local connectivity within a chart. The stencil represents the neighborhood of a cluster element. Figure 3 shows the stencils for major refinement schemes. With the same stencil, different patterns can be generated by reorienting the stencil at every corner (Figure 6). The $\sqrt{3}$ stencil alternates between a node stencil for the odd steps and a facet stencil for the even steps (Figure 7). Note that porous stencils, such as the odd stencil for $\sqrt{3}$, can be compressed by simply skipping the unused elements while traveling along the orbital path.

2.3 Atlas: unstructured connectivity

The atlas represents the cluster connectivity with the help of *corner connectors* similar to the structures in [10, 17]. While formally equivalent to edge-based access,

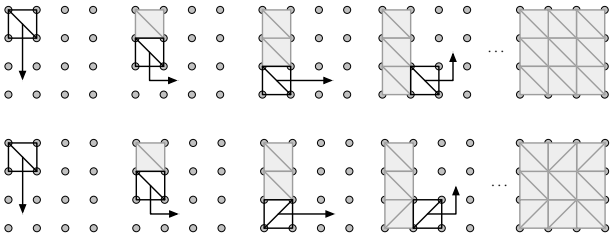


Figure 6: Two different patterns obtained by reorientation of the stencil at cluster corners.

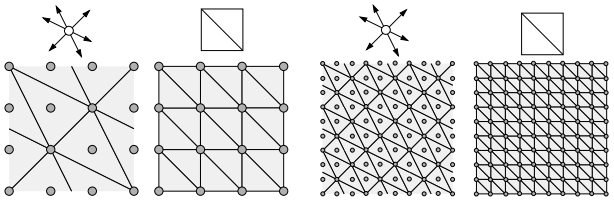


Figure 7: $\sqrt{3}$ subdivision alternates between two stencils.

this corner-based structure is highly compatible with the orbital chart access as each corner connector indices a corner element of the cluster. Each corner connector corresponds to a vertex-cluster pair, and is represented as an oriented 4-tuple $(\tau_{00}, \tau_{01}, \tau_{10}, \tau_{11})$ where (see Figure 8, (a))

— τ_{0x} are the links along the incident edges,
— τ_{1x} are the links across the incident edges.

To circulate within a cluster, one follows the inner links τ_{0x}. The τ_{1x} are called outer links. A shortest loop of inner links circulates a facet, a shortest loop of outer links circulates a vertex and a shortest loop of alternating links circulates an edge. Global boundaries are represented with boundary corner connectors whose inner links are self loops (Figure 8, (b)). Boundary vertices are circulated by the shortest loop of outer links involving a boundary corner connector.

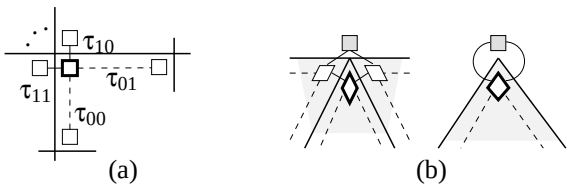


Figure 8: (a) A corner connector has two inner links (dashed) and two outer (solid) links. (b) Global boundaries have gray corner connectors whose inner links are self loops.

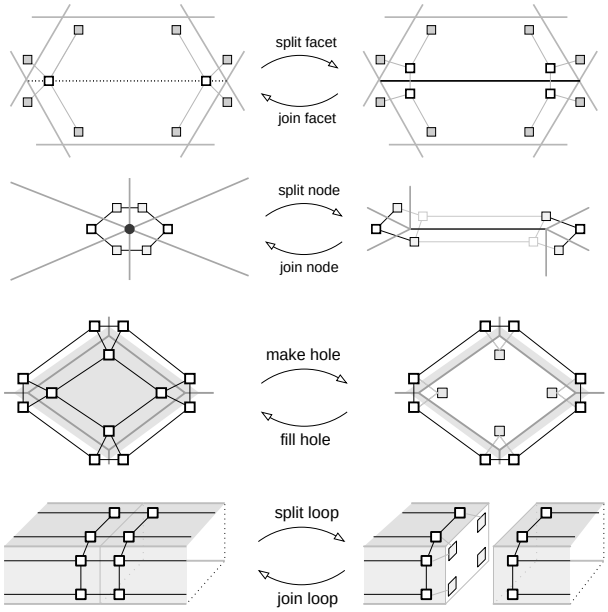


Figure 9: Connectivity (top two) and genus changing operations (bottom two).

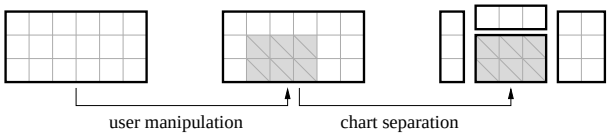


Figure 10: Separating the chart and replacing the stencil to support local changes.

2.4 Attributes

For highly regular mesh such as subdivision surfaces, Qreg stores attributes in the order of the enumeration by orbits and duplicates those on chart boundaries. Duplicates refer to the value of the attribute with least index to avoid numerical inconsistency. The memory locality within each chart assures the fast access and hence the rendering. Some efficient rendering functions (such as the vertex array in OpenGL) can also take advantage of the compactness of the attributes. For meshes with little regularity (hence small clusters), Qreg stores pointers to the attributes to avoid massive duplication.

3 Mesh Operations

The atlas supports Euler[1] and genus changing operations by reconfiguring the corner connectors as illustrated in Figure 9.

[1] Euler operations preserve the Euler-Poincaré equation: $V - E + F = 2 - 2G$ where $V/E/F$ is the number of the vertices/edges/facets and G is the genus of the underlying surface.

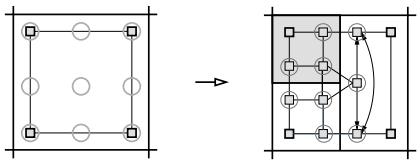


Figure 11: When separating out the upper left chart, a split into three charts enforces the spiral constraint. The newly duplicated elements are circled.

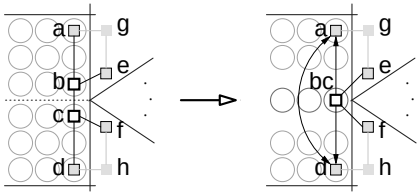


Figure 12: Removal of the dotted separating edge creates a T-corner. The long arrows between a and d indicate the asymmetric traversal at a T-corner. Asymmetric entries in the corner connectors allow skipping bc when traversing the vertical edge $a \rightarrow d$ on the aggregated side.

Connectivity can also be changed within charts. Replacing the stencil, yields a uniform edit on the entire chart. For a localized edit, the submesh becomes a separate chart with a new stencil as shown in Figure 10.

3.1 Separation and Aggregation

Chart separation splits a parent chart into several subcharts, so that the spiral constraint holds (Figure 11). Charts are allowed to join forming T-corners: the corresponding corner connector is skipped when traversing the unseparated chart (Figure 12). By definition, separation breaks the memory locality of the parent chart. The attributes are rearranged to localize the memory of each new chart. Although such memory shuffling is, in principle, costly, its effect is negligible during user-interactive connectivity editing and preferable to an indirection during rendering.

Chart aggregation reverses separation by reconfiguring

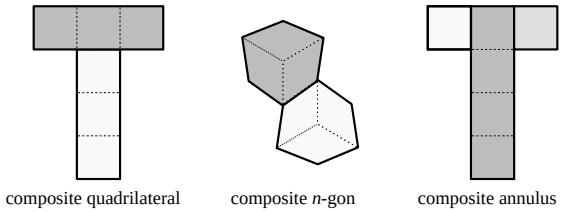


Figure 13: Chart aggregation. Decompositions of a cube that meet the spiral constraint.

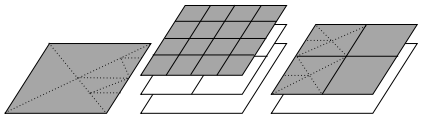


Figure 14: Uniform level-of-detail refinement. The dotted edges indicate the renderers stitching of different levels without cracks.

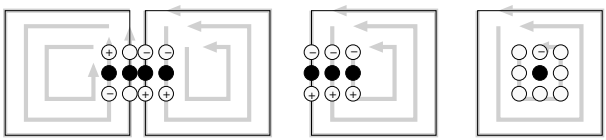


Figure 15: The moving window of a 1-ring neighborhood (a) across a cluster boundary, (b) inside the chart, and (c) at the center. The touching circles in (a) indicate duplicated boundary data.

the atlas and rearranging attributes. Aggregation can also generate T-corners and must enforce the spiral constraint for the composite chart. In addition to the composite triangle, quadrilateral and *n*-gon, the composite can be an annulus (as a cylinder without top and bottom) with equal length along both edges or have one side shorter. Figure 13 shows three alternative representations of a cube as composite charts.

3.2 Uniform and Adaptive Refinement

Mesh refinement of a Qreg amounts to the scaling of each chart. Uniform level-of-detail refinement is implemented by storing all levels of the refined chart. Each chart keeps count of the current level for rendering and the maximal level of refinement. Stitching different levels without cracks (Figure 14) is delegated to the rendering routine. The alternative, fine-grain adaptive refinement, applies the separation of the chart (Figures 18). Repeated separation is registered in a kd-tree multiresolution hierarchy.

4 Implementation and Analysis

4.1 Implementation

The atlas is realized as an array of corner connectors. The corner connectors of each chart are stored consecutively in counter-clockwise order and clusters of the same cardinality *n* are grouped together. Since inner links can be replaced by modulo *n* computation, uni-directional traversal requires only one outer link to be stored per corner connector.

There are two ways to implement chart connectivity emphasizing either performance or flexibility. In a static setting, such as a specific subdivision algorithm, the enumeration by orbits and the stencil are a *fixed-function procedure* or a *static lookup table* of the neighborhood. To support a large variety of dynamic patterns, the connectivity is computed *on-the-fly* by moving a window to neighbor orbits along. To this end, each chart records (1) an index base, (2) the last element of the outermost orbit and (3) the change δ in the number of elements per orbit when switching to the next, more inward orbit. For the clusters in Figure 5, $\delta := -1/-2/-3$ for hexagon/quad/triangle respectively. Each corner stores the index offset from the start element so that a consecutive pair defines a range for stepping linearly from element to element. These informations facilitate the enumeration of the chart.

The moving window encloses potential neighbors of the current element. For example, for a PQQ mesh and a 1-ring neighborhood, the window maintain indices from three orbits. Most of the time, (Figure 15 (b)) the window is updated by incrementing the index. When moving along the chart boundary (Figure 15 (a)), one index, across the boundary, has to be decremented. At a corner, the indices are updated using δ and the index offset of the corner. The center of a cluster has special connectivity. Access is fast for the main case of structured traversal but also fast enough when a user interactively picks a random node.

Clustering Tool and Heuristic. To cluster small meshes, such as the the hand in Figure 2, the user specifies a stencil and picks a start element among the unclustered elements. Then as many orbits are added as possible. For large meshes as in Figure 1, such user input is not practical. Fortunately, optimal placement of the start element is also not as crucial. Our routine searches for the prescribed pattern and grows the region matching the pattern while maintaining an approximate center. The orbits are added around the center as long as the spiral constraint holds. Once the large regular areas are exhausted, the remaining triangles are clustered as *n*-gon fans and, once these are exhausted, as pairs where possible.

4.2 Experimental Analysis

Connectivity compression. In general data sets, the savings depend on the heuristics employed and the specific suite of sample objects. For a highly regular mesh, such as the Pawn in Figure 17 *upper-left*, the number of connectivity primitives can be compressed by 90%. Here a connectivity primitive is either a halfedge or the size-equivalent corner connector. For hand crafted meshes, such as the Duck in Figure 19 or the Gripper in Figure 16, the savings are about 50% (without the user-intervention during clustering). For highly irregular meshes, such as the Bonehand and the Dragon in Figure 16, the savings

Test Model	Halfedge	Qreg	Reduction
Pawn	624	32	94.87%
Duck	1048	506	51.72%
Gripper	2904	1520	47.66%
Bonehand	6390	2660	58.37%
Dragon	8190	3496	57.31%

Table 1: Number of connectivity primitives for highly regular models (Pawn, hand-clustered), hand-crafted models and meshed models ((Duck, Gripper, Bonehand, Dragon, clustered by our heuristic).

Depth	Halfedge	Per-Facet	Group	Qreg/Halfedge
Cube				
1	0.0002	0.0002	0.0002	00.00%
2	0.0005	0.0004	0.0004	80.00%
3	0.0021	0.0014	0.0014	66.67%
4	0.0091	0.0051	0.0051	56.04%
5	0.0435	0.0200	0.0199	45.98%
Gripper				
1	0.0193	0.0168	0.0143	74.09%
2	0.0855	0.0565	0.0477	55.79%
3	0.3573	0.1748	0.1625	45.48%
4	1.4107	0.6380	0.6181	43.82%
5	5.6965	2.4766	2.4292	42.64%

Table 2: Catmull-Clark subdivision (time in seconds) for the CGAL-based halfedge implementation, vs the Per-Facet Qreg and the Group Qreg implementations.

are almost 60%. Details are recorded in Table 1. Note that refinement of clustered meshes improves the savings exponentially. Two steps of PQQ/PTQ refinement on the Duck (see Figure 19) require 16672 halfedges for halfedge-based structures but a constant 506 corner connectors for Qreg. The number of connectivity primitives is compressed by 96.96%.

Refinement efficiency. Qreg uses space and time comparable to the less flexible patch-based structures and avoids the exponential cost of computing and storing connectivity of refined halfedge based structures. We analyzed Catmull-Clark subdivision when one chart corresponds to one input facet (Per-Facet Qreg), and when our heuristic groups facets into larger clusters (Group Qreg). Access is computed on-the-fly (see Section 4). After a few steps of subdivision, both are faster than the CGAL Polyhedron [11] which is considered the most efficient halfedge-based implementation. Group Qreg is slightly faster than Per-Facet Qreg. Table 2 gives detailed numbers for a cube and the Gripper mesh.

The analytical comparison between Qreg and edge-based implementations of primal quadrisection is given in Appendix.

5 Conclusion

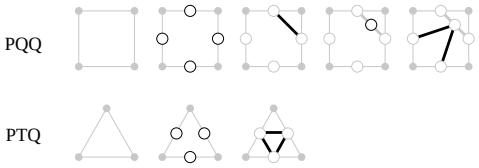
The Qreg data structure is tailor-made and very versatile for manipulating large meshes with regular regions. For subdivision meshes, it is shown to be about as efficient as (less flexible) patch-based structures and more efficient than halfedge-based structures. In particular, Qreg does not scatter attributes in memory. We are currently using Qreg in a hierarchical modeling environment with generically implemented subdivision algorithms (Figure 17). While the purpose of Qreg is mesh modification, it can serve as a good starting point for existing compression and transmission algorithms.

Appendix: Analytical Comparison

We consider an input subdivision control mesh with n_e edges, n_f quadrilateral facets and n_v vertices. The assumption that facets are at least paired or connected into a fan eliminates at least $\frac{n_f}{2}$ edges.

We first compare the space requirements. The number of halfedges equals the number of corner connectors in a mesh. If we now perform k steps of a PQQ-type subdivision, then any patch-based structure and Qreg retain their fixed number of respectively $2n_e$ and $2n_e - n_f$ connectivity primitives (i.e. corner connectors) while a general halfedge structure needs to allocate $\sim 4^k n_e$ connectivity primitives (i.e. halfedges). Since nodes on patch boundaries are duplicated, patch-based structures allocate $(2^k+1)^2 n_f$ space for node attributes, Qreg allocates $\sim (2^{k+1}+1)(2^k+1)\frac{n_f}{2}$, and the halfedge structure allocates $\sim 4^k n_v$.

To quantify the computational savings for PQQ and PTQ refinement. We measure the Euler operations broken into insert-vertex-on-edge (IVE) and insert-edge-between-vertices (IEV) instructions used by a halfedge structure as,



Each IVE changes 8 adjacency pointers and each IEV 10. Both IVE and IEV allocate memory for two halfedges. There is 1 IVE per edge. PQQ uses 1 IVE and $(d-1)$ IEVs per facet, where d is the degree of the facet. PTQ uses 3 IEV per facet as summarized below with the pointer manipulation count in parentheses.

Refinement	IVE per edge	IVE : IEV per facet	
PQQ	1 (8)	$1 : d-1$	$(10d-2)$
PTQ	1 (8)	$0 : 3$	(30)

For the Duck model, Figure 19, the number of IVEs, IEVs and pointer manipulations for refining from level 3 to level 4 are 40544, 59328, and 917632 respectively. None of these operations are necessary for patch-based structures and Qreg.

Acknowledgements

We thank Pierre Alliez and David Gu for the remeshed dataset in Figure 1 and Figure 2. This work was supported in part by NSF grants DMI-0400214 and CCF-0430891.

References

[1] Pierre Alliez, David Cohen-Steiner, Olivier Devillers, Bruno Levy, and Mathieu Desbrun. Anisotropic polygonal remeshing. In *SIGGRAPH '03 Conference Proceedings*, pages 485–493, 2003.

[2] Bruce G. Baumgart. A polyhedron representation for computer vision. In *Proceedings of the National Computer Conference*, pages 589–596, 1975.

[3] Swen Campagna, Leif Kobbelt, and Hans-Peter Seidel. Directed edges — A scalable representation for triangle meshes. *Journal of Graphics Tools*, 3(4):1–12, 1998.

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

[5] D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10:356–360, September 1978.

[6] L.De Floriani, L.Kobbelt, and E. Puppo. A survey on data structures for level-of-detail models. In N.Dodgson, M.Floater, and M.Sabin, editors, *Advances in Multiresolution for Geometric Modelling, Series in Mathematics and Visualization*, pages 49–74. Springer Verlag, 2004.

[7] Leo J. Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. In *Proceedings of the fifteenth annual ACM Symposium on Theory of Computing*, pages 221–234, 1983.

[8] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108, 1996.

[9] Martin Isenburg and Jack Snoeyink. Face fixer: Compressing polygon meshes with properties. In *SIGGRAPH '00 Conference Proceedings*, pages 263–270, 2000.

[10] Kenneth I. Joy, Justin Legakis, and Ron MacCracken. Data structures for multiresolution representation of unstructured meshes. In Gerald Farin, Hans Hagen, and Bernd Hamann, editors, *Hierarchical Approximation and Geometric Methods for Scientific Visualization*. Springer-Verlag, 2002.

[11] Lutz Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *Computational Geometry*, 13(1):65–90, May 1999.

[12] Leif Kobbelt. $\sqrt{3}$ subdivision. In *SIGGRAPH '00 Conference Proceedings*, pages 103–112, 2000.

[13] A. Levin and D. Levin. Analysis of quasi uniform subdivision. *Applied and Computational Harmonic Analysis*, 15(1):18–32, 2003.

[14] Charles T. Loop. Smooth subdivision surfaces based on triangles, 1987. Master's Thesis, Department of Mathematics, University of Utah.

[15] Jörg Peters. Patching Catmull-Clark meshes. In *SIGGRAPH '00 Conference Proceedings*, pages 255–258, 2000.

[16] Kari Pulli and Mark Segal. Fast rendering of subdivision surfaces. In *Proceedings of the eurographics workshop on Rendering techniques '96*, pages 61–70, 1996.

[17] J. Rossignac, A. Safonova, and A. Szymczak. 3d compression made simple: Edge-breaker on a corner table. In *Proceedings of the Shape Modeling International 2001*, 2001.

[18] Jarek Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, 1999.

[19] Le-Jeng Shiue, Vineet Goel, and Jorg Peters. Mesh mutation in programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 15–24, 2003.

[20] Jos Stam and Charles Loop. Quad/triangle subdivision. *Computer Graphics Forum*, 22:79–85, 2002.

[21] Joe Warren and Henrik Weimer. *Subdivision Methods for Geometric Design*. Morgan Kaufmann Publishers, 2002.

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

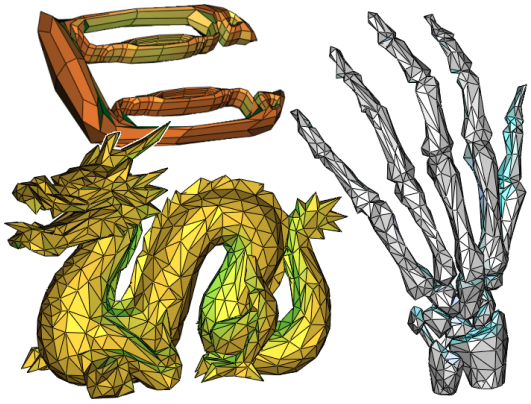


Figure 16: Test models: Gripper, Dragon, Bonehand.

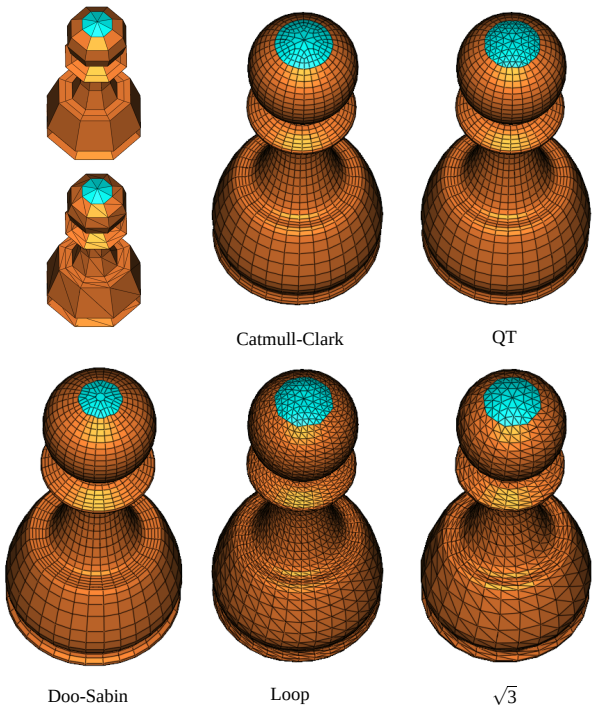


Figure 17: Quasi-regular meshes represented by Qreg include primal, dual and $\sqrt{3}$ subdivision. The 160-facet input mesh is grouped into 3 clusters – a fan on top and one on the bottom. The remainder forms one cylindrical quadrilateral or triangulated cluster.

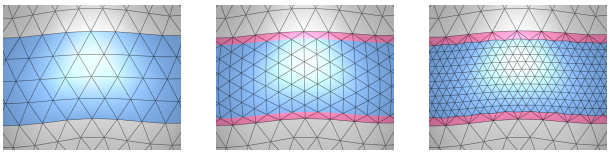


Figure 18: Adaptive $\sqrt{3}$ refinement uses chart separation. The adaptive cluster is blue.

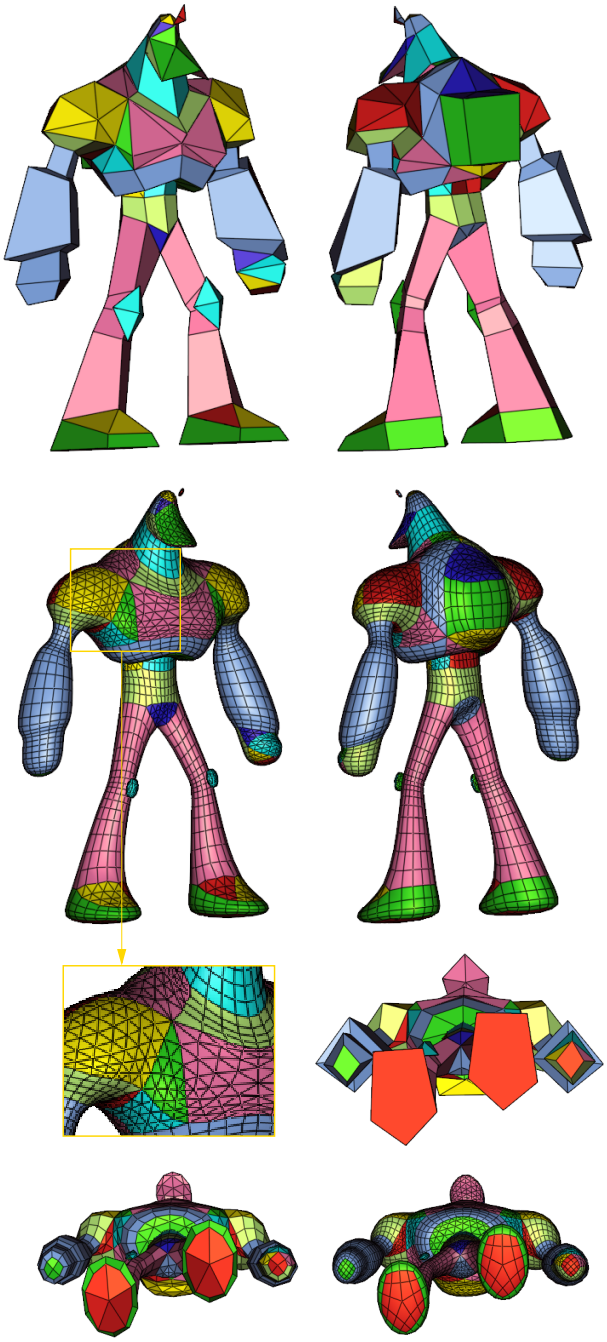


Figure 19: (row 1:) Clustering in a user-created polyhedron. (row 2:) Quad-Tri subdivision using the Qreg representation. (rows 3, left:) Different stencils are applied to represent PQQ and PTQ refinement structures. (row 3 and 4:) Note that the bottom of each foot (shown as a red pentagon) forms one *cluster and need not be split into quads as in patch-based subdivision.*