Submitted for publication, January 2001

Hybrid Meshes

Igor Guskov
Caltech

Andrei Khodakovsky
Caltech

Peter Schröder
Bell Labs

Wim Sweldens
Bell Labs

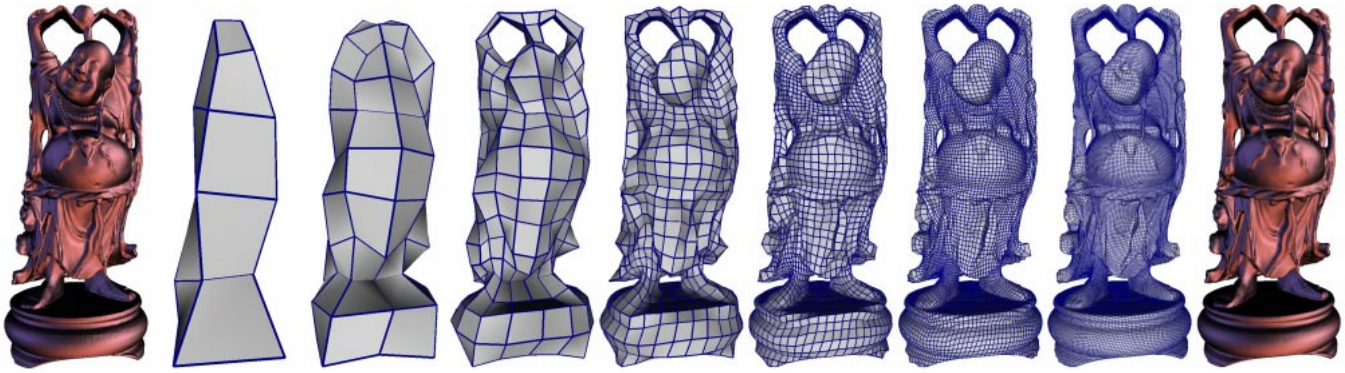

Figure 1: *Example of a hybrid mesh (left to right). Starting with the Stanford Buddha original dataset a base domain with genus 0 is constructed. After one regular refinement step, tunnels are created between arms and head followed by another regular refinement step and creation of a tunnel between the feet. After one more refinement step three small tunnels are created on the sides followed by further regular refinement. The model remains a valid 2-manifold throughout and the final remesh has genus six.*

Abstract

Hybrid meshes are a novel multiresolution mesh structure which combines the flexibility of irregular and the simplicity of (semi-)regular meshes. We show how to construct hybrid meshes using both regular refinement to build smooth patches and irregular operations to allow topology changes. We provide a user driven procedure for remeshing scanned geometry with hybrid meshes. Several examples and applications are included.

1 Introduction

Highly detailed and complex surfaces are most commonly described through meshes. Typical sources of such meshes range from CAD systems (e.g., STL files) to range sensing [19] and isosurface extraction [20]. Sizes can easily reach to millions, and even billions, of vertices. Processing such digital geometry requires efficient algorithms and powerful mathematical tools which go beyond standard signal processing approaches. Among other factors the connectivity of the meshes plays a major role in the construction of appropriate algorithms. Typically we distinguish between *irregular* and *(semi-)regular* meshes. Irregular meshes have no restriction on the valence of vertices, while regular meshes are formed by starting from a coarse irregular base domain and applying recursive regular refinement, resulting in large regular grid patches. Both types of meshes come with their own advantages and disadvantages.

Since there is no valence restriction, irregular meshes are very flexible; they can resolve complex geometric features as well as accommodate topology changes. The irregularity leads to costly linked data structures and complex algorithms for multiresolution, smoothing, compression, editing, etc. Regular (or semi-regular) meshes on the other hand have an almost everywhere regular structure which allows for efficient tree or array based data structures and signal processing algorithms supported by well developed mathematical theory such as Fourier, spline, subdivision, and wavelet methods. Also because of their regular and hence predictable parametric structure and connectivity, regular meshes are much more compressible than irregular ones [13].

The main drawback of regular meshes is their lack of flexibility in resolving complex or high genus shapes. A regular mesh always has a bijective mapping between the coarsest level (base domain) and the finest level, implying that the base domain has to have the same genus as the final model. This implies that high genus meshes cannot have simple base domains. Aside from that, the quality of this mapping, say its smoothness and membrane or stretching energy, determines the quality of the (semi-)regular mesh. Smooth maps can be computed using the right parameterization method. However, a stretched map leads to bad aspect ratio polygons, poor approximation, and numerical problems. To ensure a high quality mapping the complexity of the base domain must grow as the geometric complexity of the model grows. The most typical examples are spiky features. Consider bunny ears: unless the base domain has a few polygons outlining the ears, a semi-regular mesh will never be able to resolve the ears without very stretched polygons, see the left ear in Figure 2. In a sense there must be enough "skin" in the base domain to avoid later stretching. Note that the stretching problem is a fundamental problem of regular meshes due to the curvature of the model. It cannot be solved by computing some special parameterization or through the use of adaptivity. The former cannot work because curvature is inherent in the original model. The latter does not work since regular refinement of a polygon into congruent pieces does not improve the aspect ratio. An alternative solution is the use of non-regular adaptive refinements such as triangle bisection. However, this no longer creates a regular mesh and all associated advantages disappear.

Both the genus and the stretching problem restrict how coarse the base domain can be and limit scalability of semi-regular meshes. Even if the initial remesh is of high quality, aspect ratios can quickly deteriorate during editing operations as observed in [16]. Also a fully featured editing system should accommodate topology changes.

To address the above problems the present paper introduces *hybrid meshes*, which combine the best of the regular and irregular worlds, see Table 1. They are flexible because they allow the growth of extra "skin" and support topological changes; their implementation uses tree based data structures, inheriting their efficiency and simplicity; and they support signal processing type al-

	irregular	regular	hybrid
flexibility	high	limited	high
data structure	complex (links)	tree (easy)	forest (easy)
compressibility	limited	high	high
multiresolution	complex	easy	easy
signal processing	complex	easy	easy
topology changes	possible	impossible	possible

Table 1: *Comparison between irregular, regular, and hybrid meshes: Hybrid meshes combine the best of the regular and irregular worlds.*

gorithms easily, exploiting the mathematical machinery built up for the (semi-)regular setting. The main contributions of this paper are the formalization of the notion of hybrid meshes and a user driven algorithm to construct hybrid meshes from given geometry.

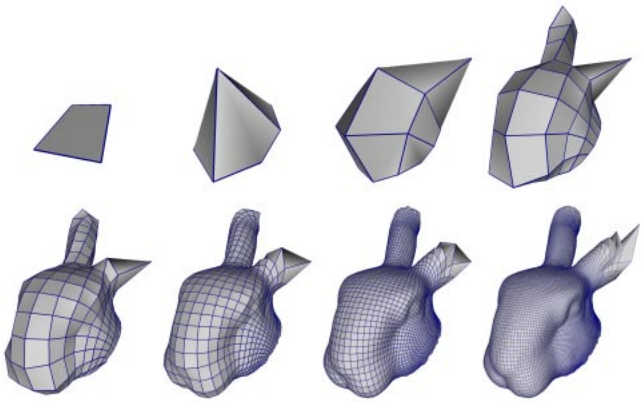

Figure 2: *A remesh of the bunny head: the left ear uses a standard semi-regular mesh, while the right ear uses a hybrid mesh. Starting from a single quad extra "skin" is provided for the right ear after three regular refinement steps through the addition of three cubes (see also Figure 12). The resulting quads continue to have good aspect ratio. The left ear on the other hand suffers from severe stretching and even after seven refinement steps large parts of the left ear parameter space have not yet been sampled.*

Related Work Most remeshing work has focused on the semi-regular setting, i.e., how to construct a semi-regular resampling of some irregular geometry. Hoppe and co-workers [12] used a global optimization framework to recover a Loop subdivision surface from a given point cloud. Their examples demonstrate how geometric detail increases the combinatorial complexity of the control mesh. In order to use surface based wavelet methods [21], which effectively add details to the subdivision setting, Eck and co-workers [3] gave the first automatic remeshing procedure, but with little control over the complexity of the base domain. In many settings a fully automatic method is not appropriate and patch layout under user control is essential. Krishnamurthy and Levoy [17] introduced such a system for the construction of bicubic patch layouts on scanned meshes, while Lee and co-workers [18] described a system with a flexible tradeoff between user control and automatic remeshing. These and other semi-regular remeshing algorithms all suffer from the same distortion issues when insisting on a coarse base domain and none can accommodate topology changes during remeshing.

Changes in topology have been considered in the context of irregular hierarchy constructions through mesh simplification. For example, Garland and Heckbert [6] allowed for vertex pair collapses during progressive mesh [11] construction, sacrificing the manifold property to better deal with many connected components. A fully general treatment was given by Popović and Hoppe [22]. Topology reduction as an explicit goal was pursued by El-Sana and Varshney [4]. None of these algorithms attempt to maintain the 2-manifold property as we do.

Changing topology while maintaining the manifold property in the context of a hierarchical modeler was first introduced by Gonzalez-Ochoa and Peters [8]. In a similar setting Akleman and Chen [1, 2] focus on the problem of maintaining the 2-manifold property for subdivision control meshes. While these two papers consider the ab initio modeling setting, we focus on the problem of constructing topology changing parameterizations for *given* geometry.

2 Hybrid Meshes

We begin with the definition of hybrid meshes. To be concrete we will assume quadrilaterals as the basic face primitive and quadrisection as the associated regular refinement operation, though our definitions and algorithms do not depend on it in any essential way. For example, one may equally well use triangles or hexagons as basic face types and employ regular refinement strategies other than quadrisection (e.g., regular trisection [15] or regular bisection [24]).

One way to think of a hybrid mesh is as a generalization of a semi-regular mesh in which irregularity is not constrained to the roots of trees (Figure 3, left), but instead is allowed to occur distributed throughout the refinement hierarchy (Figure 3, right). Formally, a hybrid mesh is a hierarchical mesh representation that

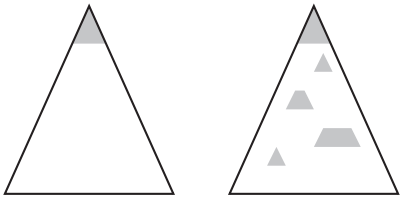

Figure 3: *Semi-regular (left) vs. hybrid meshes (right): In a semi-regular mesh, only the top is irregular (shaded) and everything on finer levels is regular. In a hybrid mesh, irregular pieces can be distributed throughout the tree*

starts with a coarsest base domain and builds finer levels using two types of operations:

- A *regular* operation in which a single quad (face) is split into four quads (regularly refined).
- An *irregular* operation in which a set of quads (faces) is replaced with another set of quads (faces) which agree on the boundary.

These two types are fairly general and allow for a multitude of possible operations whose range cannot be easily delimited. Here are some examples of useful irregular operations:

- **Diagonal collapse:** This is the equivalent of an edge collapse in triangle meshes. Take two opposing vertices of a quad and identify them. This way one quad, one vertex, and two edges disappear (Figure 4). The opposite operation is a vertex split.
- **Edge flip:** This operation entails two neighboring quads and replaces the edge between them by an edge connecting a non shared vertex from each quad (Figure 5).
- **Cut a hole:** This operation selects a connected set of quads and simply removes them, thereby creating a boundary curve (Figure 6). The inverse operation takes a boundary curve and patches it with quads.
- **Add a cube:** This operations addresses the skin stretching problem and replaces a single quad with 5 quads arranged as a cube with open bottom (Figure 7).
- **Connect:** Select two connected sets of quads, remove them, thereby creating two boundary curves and build a topological cylinder between them (Figure 8). This operation can be used to merge components, build handles (outside cylinder), or build tunnels (inside cylinder). The inverse operation identifies a

simple closed curve on the manifold, cuts along the curve and patches up the resulting two boundary loops.

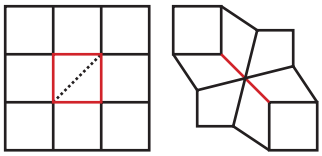

Figure 4: *Diagonal collapse.*

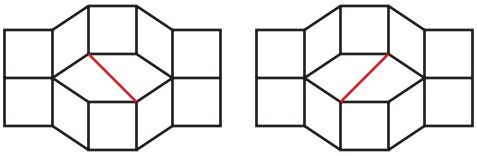

Figure 5: *Edge flip.*

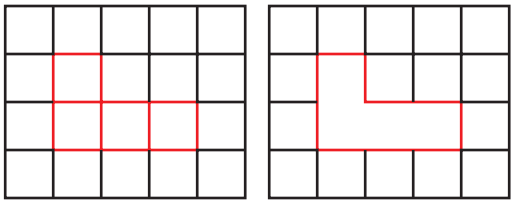

Figure 6: *Cut hole.*

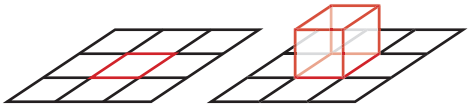

Figure 7: *Add cube.*

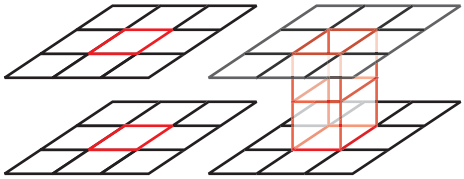

Figure 8: *Connect.*

Given the ability to grow skin and add and remove tunnels as well as boundaries the coarsest level of any given surface can be as simple as a few cubes. This demonstrates the scalability of hybrid meshes: no matter how complicated the initial surface, it can always be reduced to a few or even a single cube, say if it gets rendered as a single pixel. This is in stark contrast to semi-regular meshes for which the base domain complexity grows with the complexity of the input surface.

2.1 Hybrid Mesh Structure

With the above definition in mind we can describe a hybrid mesh as a sequence of meshes constructed through alternating regular and irregular operations. Recall the structure of semi-regular meshes. They start with an irregular coarse base domain $\mathcal{M}^0$ and successively build finer levels $\mathcal{M}^j$ through regular refinement. For quads the number of elements grows by a factor of four from level $j-1$ to level j. Of course, in practice there is no need to instantiate refinements uniformly, allowing for adaptively refined meshes.

To provide some structure we add the restriction that an irregular operation can only involve quads from the same level. This means that all quads removed have to be from the same level and all the newly inserted quads belong to the same level. This allows us to separate the regular and irregular operations by level. Hybrid meshes then consist of two types of meshes $\mathcal{M}^j$ and $\mathcal{N}^j$. A mesh $\mathcal{M}^j$ is built by regular refinement of mesh $\mathcal{N}^{j-1}$, while a mesh $\mathcal{N}^j$ follows from performing one or more irregular operations on mesh $\mathcal{M}^j$. The base domain is $\mathcal{M}^0 = \mathcal{N}^0$. We thus have the sequence $\mathcal{N}^0 \rightarrow \mathcal{M}^1 \Rightarrow \mathcal{N}^1 \rightarrow \mathcal{M}^2 \Rightarrow \mathcal{N}^2 \ldots$, where $\rightarrow$ stands for regular refinement and $\Rightarrow$ for irregular operations within a level. A semi-regular mesh is now the special case where no irregular operations happen and all $\mathcal{N}^j = \mathcal{M}^j$. The other extreme involves no regular refinement so all $\mathcal{M}^j = \mathcal{N}^{j-1}$ and we have a fully irregular pyramid.

2.2 Data Structures

Our hybrid mesh data structure is simply a forest, i.e., a collection of quad trees. We start out with one tree per quad of the base domain; every irregular operation terminates the subtrees formed by the quads that are removed and creates as many new trees as there are new quads. The only tricky part are all the neighborhood relationships to tie the trees together.

In the implementation we use two types of quads: root and non-root quads. Each root quad has four pointers to its neighbors and if refined has four pointers to its children. A non root quad has a parent pointer, no neighbor pointers, and if refined four pointers to its children. All regular refinement is performed uniformly, however, it may be *instantiated* lazily to provide the benefits of adaptive refinement. The latter is done with a standard restriction criterion. To limit the number of possible arrangements we added two additional restrictions to the type of operations which can occur:

- root quads cannot be removed in an irregular operation;
- quads that are involved in *different* irregular operations on the *same* level cannot be neighbors, i.e., there has to be at least one quad between them.

Regular operations simply populate a standard tree. Irregular operations involve removing certain non-root quads as well as inserting new root quads. Putting in the neighbor pointers for the new root quads is straightforward. The only difference with respect to base domain root quads is that the neighbors need not be root quads themselves. When a non-root quad gets removed, it is replaced by a virtual root quad. This virtual root quad does not correspond to a quad in the mesh, but only stores the neighbor pointers of the removed quad to the newly created root quads. For example, for the add cube operation (Figure 7) the virtual root quad will point to the four sides of the cube. In case the removed quad has no new root quad as neighbors, there is no need for a virtual root quad. This happens for example when a single quad is removed to create a boundary, or in case a removed quad does not touch the boundary of the removed quad region.

There are standard algorithms for resolving neighborhood relations in quadtrees (e.g., [23]). Neighbor questions ascend up the tree, reach the nearest common ancestor, and descend back down to find the answer. For semi-regular meshes, one modification is needed: When the question reaches a root quad, its explicit neighborhood relationships combined with a stored orientation permutation are used before descending down the tree again. For hybrid meshes, another modification is needed. When descending the tree, one can reach a removed quad. Then the explicit neighbor relations of the virtual root quad, again with a suitable permutation, are used to continue the algorithm.

The actual implementation of the data structure is more efficient than the one described above. For example, given that four children are always created as a group, they are stored in the *same* structure. Hence only one, not four child pointers are needed. This also allows the sharing of pointers to vertices and results in a cost of only 12 bytes per quad (not counting storage for vertices). Alternatively one may use levelwise arrays which are even more compact and may be preferable on some architectures.

3 Building Hybrid Meshes

In this section we describe our algorithm to build hybrid meshes. The input is typically a fine irregular triangle mesh, though the

faces could be more general if desired. The system provides a mixture of automatic operations and interactive user control. Certain tasks such as tracing curves and smoothing patch boundaries are best done by the system while others such as deciding topological cuts or aligning patch boundaries with features are best done by the user.

In this section we go in detail through the steps for building a hybrid mesh for two examples: a Max Planck head model, which does not require topological changes, and a close-up of the side of Buddha, which does change genus. Details on all the algorithmic components such as parameterization, relaxation, and building cubes are given in Section 4.

Max Plank The hybrid mesh is built in 9 stages, see Figure 9.

1. First we need a base domain $\mathcal{M}^0$. This base domain can be built automatically or can be specified by the user. In the case of the Planck head, the user specifies four points on the boundary (bottom of the neck). Subsequently, the system automatically adds three cubes, places their vertices on the model, traces and smoothes the patch boundaries, and computes a parameterization for all patches as shown in the top left of Figure 9.

2. Next the parameterization is used to compute three levels of regular refinement; this gives $\mathcal{M}^3$. For each quad q of $\mathcal{M}^3$, the system also computes a stretch parameter σ_q defined as

$$\sigma_q = \frac{A_q}{P_q}$$

   where A_q is the area on the input model covered by this quad patch and P_q is the area of the quad in parameter space. Assuming root quads to have unit parameter area, the parameter area P_q for any quad of level j is 2^{-2j}. The top middle shows a colored texture map indicating the value of σ_q for $\mathcal{M}^3$. In this way the user can quickly identify that the region around the nose needs extra skin.

3. The user selects a region with 2×3 patches around the nose for an irregular operation (top right). The system automatically adds one layer of patches (giving 4×5 patches) as a buffer zone to ensure a parametrically smooth transition.

4. The system cuts the selected 4×5 region from the original mesh, removes the patch layout in the 2×3 interior, and presents it to the user (middle left).

5. The user draws a new patch layout in the empty region by specifying control points and their connectivity. The system responds by drawing geodesics connecting them, and ensures that the new patches connect correctly to the existing ones (center).

6. After the layout is specified, the system smoothes the curves and computes a parameterization for the new patches (middle right).

7. The system attaches the newly created patches at level 3 of the remesh replacing the quads selected earlier, thereby going from $\mathcal{M}^3$ to $\mathcal{N}^3$. The patches outside the selected area get refined automatically. The stretch parameters are recomputed for the quads of $\mathcal{N}^3$ (bottom left). Compare with the top middle figure to see how stretching around the nose has been reduced.

8. Remeshing now continues to build $\mathcal{M}^4$ (bottom middle). Around the nose, the new parameterization is used, everywhere else we still use the original one.

9. More adaptive refinement steps are added until we reach the final adaptive remesh $\mathcal{M}^8$. The remesh has $26K$ vertices compared to $25K$ for the original. The relative L_2 error is 0.01%.

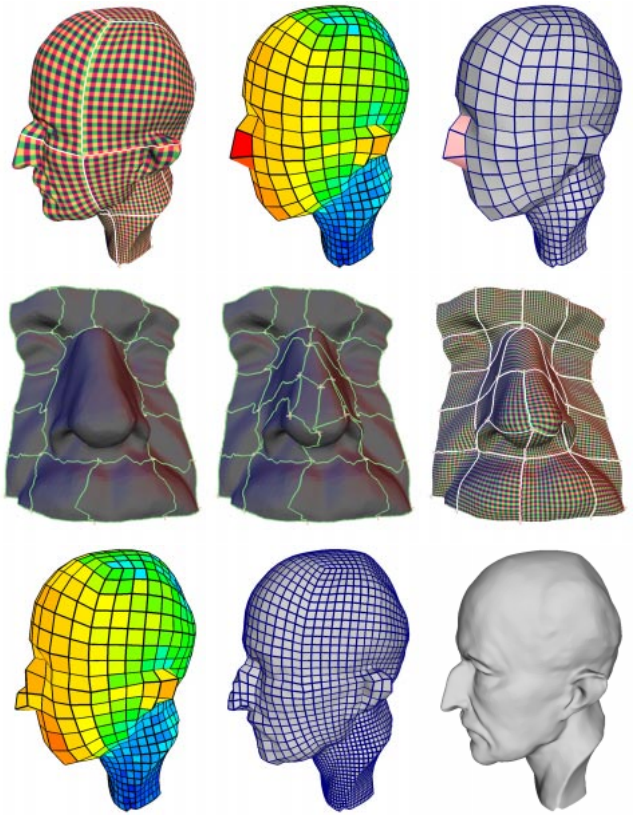

Figure 9: *Sequence of screen shots for building the Max Planck hybrid mesh.*

Close-up of Buddha If the original mesh has genus higher than zero, we first simplify its topology to obtain a genus 0 model. For each tunnel, the user needs to specify a non-separating cut on the input mesh. (A non-separating cut is a simple closed curve which, if cut, does not create a new component.) The system then cuts the mesh, thereby creating two new boundary curves, and fills the two holes. As an example we show how a tunnel on the side of the Buddha was handled in eight steps, see Figure 10.

1. The user draws a curve along the tunnel and the system closes the tunnel with two filled caps (Figure 10a).

2. Once all tunnels are closed or handles are cut, a parameterization onto a genus 0 base domain can be computed and remeshing can begin. All filled caps get parameterized as well. Note that the two back to back filled caps that were put in for each cut get assigned to different regions in the parametric domain. At the appropriate level, the user can ask the system to reintroduce a genus change. The system responds by selecting the appropriate region on the remesh (Figure 10b) which covers both filled caps. Note that in general there could be two separate regions, one for each of the caps.

3. The system cuts out the submesh of the original surface which is associated with this region (Figure 10c).

4. The system removes the caps (Figure 10d).

Figure 10: *Sequence of screen shots for building the side of the Buddha.*

5. An entirely new patch layout needs to be generated for this submesh, with the only requirement that it matches the boundary of the cut-out region (Figure 10e). The new layout is either drawn by the user from scratch, or can be copied from a layout template (see "Transferring Layouts" in Section 4).

6. The specified layout is automatically relaxed (Figure 10f).

7. The system computes a parameterization for the submesh (Figure 10g) and the highlighted region of Figure 10b is replaced by the set of newly created root quads.

8. The remeshing process continues (Figure 10h). The remesher uses the original base domain parameterization outside of the pink region, and the new parameterization for the submesh.

4 Algorithmic Components

Here we give more details of the algorithmic components used for building hybrid meshes.

Path tracing Paths on the original mesh are used to delineate boundaries between parametric patches. In order to build them we employ a simplified version of the standard shortest path algorithm on triangulated domains [14]. It uses a topological distance in the dual graph of the triangulation. Note that these paths are relaxed later.

Parameterization Any remeshing method first needs to compute a parameterization, i.e., build a bijection between a region of the original input surface and a region of the parameter plane. Here we present the basic algorithm for parameter computations. Consider a region $\mathcal{R}$ of the mesh homeomorphic to a disc that we want to parameterize onto a convex planar region $\mathcal{B}$, i.e., find a bijective map $u : \mathcal{R} \rightarrow \mathcal{B}$. The map u is fixed by a boundary condition $\partial\mathcal{R} \rightarrow \partial\mathcal{B}$ and minimizes a certain energy functional. We here consider the functional provided by Floater [5]. The function u needs to satisfy the following equation in the interior:

$$u_i = \sum_{k \in \mathcal{V}(i)} \alpha_{ik} u_k, \qquad (1)$$

where $\mathcal{V}(i)$ is the 1-ring neighborhood of the vertex i in the original mesh and the weights α_{ik} are as in [5]. The main advantage of the Floater weights is that they are always positive, which, combined with the convexity of the parametric region, guarantees that the mapping is a bijection so no quad flipping can occur. The system (1) is solved with the biconjugate gradient method [7]. This method is used to compute a parameterization of each patch onto the unit square $[0, 1] \times [0, 1]$ in the (u, v) plane.

Patch boundary and global vertex relaxation Computing parameterizations per patch does not give us a globally smooth mesh. We also need to relax the patch boundaries and global vertex positions. This is done in a manner similar to [9]. Specifically, to relax a patch boundary, we parameterize the two neighboring patches onto a rectangular region $[0, 2] \times [0, 1]$ in the (u, v) plane. The path endpoints are mapped to $(1, 0)$ and $(1, 1)$. The smoothed path is then the mapping of the straight line between $(1, 0)$ and $(1, 1)$ (in fact only the u parameter needs to be computed). To relax a global vertex position, the vertex and its neighboring patches are mapped to a star shaped region in the parameter plane (Figure 11). Then the global vertex is reassigned to a vertex on the original mesh region which is closest to the parametric center of the star. Given that we select a vertex from the interior of the region we do not care that the region is non-convex. Subsequently, the incident paths are relaxed with the procedure described above.

This procedure is also useful when building layouts. If, while building a layout, one has a patch with an even number of vertices the above procedure can be used to add a new center vertex and divide the patch into quadrilateral patches.

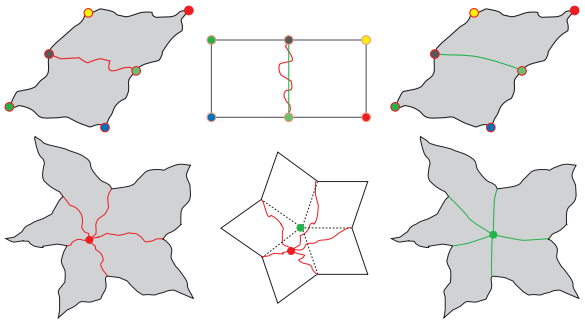

Figure 11: *Patch boundary and global vertex relaxation.*

Transferring layouts Because the input mesh can be large, it is not efficient to run global relaxations on the original input mesh. Instead we run relaxations on a coarser version built through mesh simplification. To transfer a patch boundary back to the original model, we project all vertices onto the model, connect them with geodesics, and rerun the patch boundary relaxation. Global vertex relaxation is much more expensive and does not need to be rerun on the fine model. We typically build the hybrid mesh layout interactively on the coarse version and later transfer the entire layout off-line to the original model. If needed, the user can still make changes to the finest level layout. The final version of Buddha in Figure 1 was generated using this method. The user worked with a mesh which was 10 times smaller than the original.

Cube building If a patch has severe stretching, the system provides a semi-automated way for adding skin by building extra cubes. Consider a patch with a high stretch parameter and fix its boundary on the original mesh. The system will find the original mesh vertex in the patch which is furthest from the boundary in geodesic distance (Figure 12). It then traces four geodesic curves from the patch corners on the boundary to this extremal point. The right number of cubes to insert can be determined automatically by rounding the ratio of the average length of the traced curves to the average length of the four boundary curves. The vertices of the new cubes are then found by equally dividing the four curves traced to the extremal point.

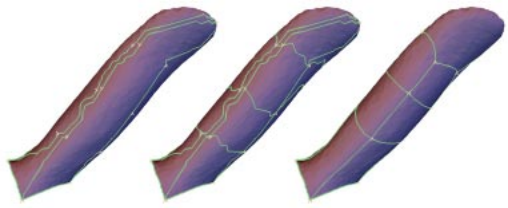

Figure 12: *Building skin with cubes for the bunny ear. Left: geodesics to the furthest point, center: geodesics are split and connected, right: relaxed version.*

Connect with cylinder This procedure is used to connect two components of the *original model*. The user draws a simple closed curve on each component. The inside caps are removed on each component and the system inserts a triangulated cylinder between the two curves on the original mesh. Both curves are parameterized and sampled uniformly to place the vertices (see Figure 18 and the paragraph "Foot bones" in the next section for more details).

5 Results

Feline The results for the Feline model are shown in Figure 13. We use the cube building algorithm to construct a 4 cube base domain, insert legs and tail on level 1, and horns on level 2. The original tail has two tunnels which get recreated on levels 2 and 3, see Figure 14.

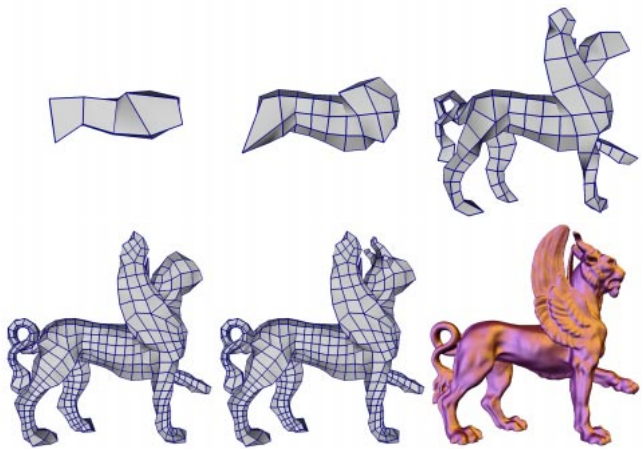

Figure 13: *The Feline remesh begins with 4 cubes adding long chains of cubes automatically at level 1 to accommodate legs, tail, head and wings. At level 2 horns are added and the tail fuses (see Figure 14). The bottom right shows the adaptively refined $\mathcal{M}^8$ with a relative L_2 error of 0.01%.*

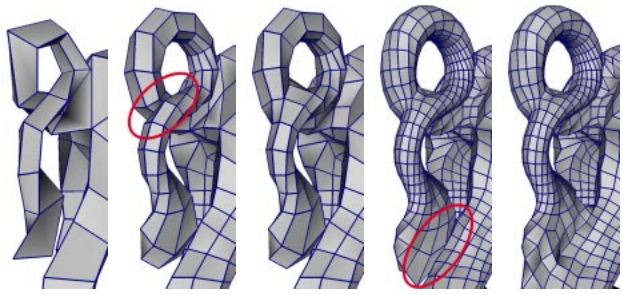

Figure 14: *Detail of the Feline tail section. On level 1 a long cube chain is grown to accommodate the tail. At level 2 the top loop of the tail fuses, while at level 3 the end of the tail fuses with the leg.*

David's head The David head starts with a 2 cube base domain (Figure 17, left). Irregular operations are used to add skin for the nose and align patch boundaries with the eye lids, see Figure 15. Figure 16 shows a close-up around the eye while Figure 17 shows the uniform $\mathcal{M}^6$ as well as the final adaptive $\mathcal{M}^{11}$.

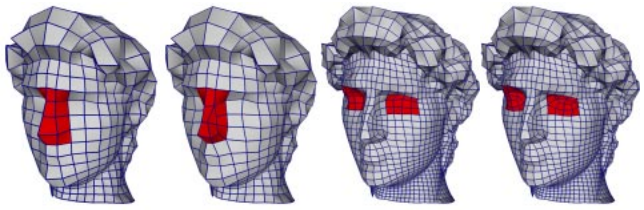

Figure 15: *Irregular operations for David's nose at regular refinement level 3 (left; before and after) and for his eyes at level 4 (right; before and after).*

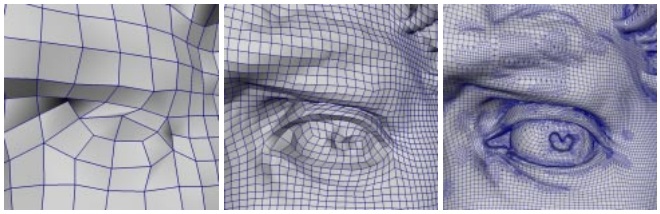

Figure 16: *Detail of David's left eye at remeshing levels 4 (after insertion of hybrid patches for lid alignment), 6, and 11 (adaptive).*

Figure 17: *Hybrid mesh of David's head. Coarsest level (two cubes with open bottom), level 6 after hybrid operations (see Figure 15, and the final remesh with 11 levels (see eye detail in Figure 16, right).*

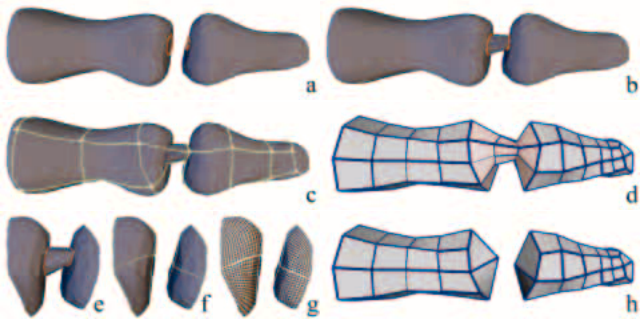

Figure 18: *Remeshing of multiple components.*

Foot bones The foot bones is an example with 19 connected components, see Figure 19. We first discuss the procedure in detail for two bones, see Figure 18. The user starts by outlining a simple closed curve on each original component (Figure 18a). The system creates a cylinder in between (Figure 18b). Then a patch layout is generated for the combined components (Figure 18c). After one level of refinement, the user decides to make a cut and highlights the region around the tunnel (Figure 18d). The system cuts out the corresponding region of the original mesh (Figure 18e) and removes the cylinder. The user provides new patch layouts for each

of the caps (Figure 18f) and the system computes parameterizations (Figure 18g). The cut can now be made and remeshing can continue (Figure 18h). For the entire model, see Figure 19, the user first adds cylinders to group the components into a single component. The base domain is laid out by hand (top left), except for the "fingers" which are done by cube building. During various stages of the remeshing, cuts get made to create new components as described above.

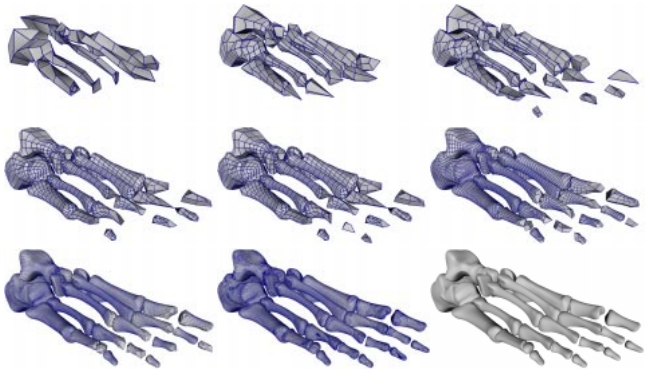

Figure 19: *Foot bones example: remeshing of a multiple component dataset starting from a single connected component base mesh.*

Statistics Table 2 shows various statistics for the models used in this paper. The user time is the time needed for the user to build all hybrid mesh operations on an intermediate level. For big meshes (Buddha and David's head) the system time is the time needed to transfer the patch layout to the original model together with the time needed to compute the fine remesh. The error is computed with the I.E.I-CNR Metro tool (the Metro tool was unable to compute the error for the finest level Buddha).

Note that the original David data is contaminated with "topological noise," i.e., there are numerous tiny tunnels, especially around the hair. These tunnels were automatically removed with a topological noise removal algorithm described in [10]. After this topological cleanup step there are still significant "stalactite" like features left which are mostly located *inside* the model. These stalactites were removed during hybrid remeshing.

Model	# Vertices Original	# Vertices Remesh	Relative L_2 error	User Time	System Time
Buddha	544171	306644	N/A	2 hr	4 hr
Planck	25445	26067	1.0e-4	10 min	2 min
Feline	49864	70793	9.2e-5	1.5 hr	14 min
David	590337	380582	4.0e-4	2 hr	43 min
Bones	33684	75747	7.3e-5	1.5hr	12 min

Table 2: *Summary of hybrid mesh results for different models. The relative L_2 errors are computed with the I.E.I.-CNR Metro tool. Timings were performed on a 1Ghz Pentium III machine.*

6 Conclusions and Future Work

The past few years have seen a whole series of papers on (semi)-regular mesh applications. Almost all of them mention adding topological operations in their future work section. The present paper provides a solution to this challenge through the introduction of hybrid meshes. They represent a powerful new mesh description which combines the best of regular and irregular meshes. In turn hybrid meshes provide the foundation for a whole series of new algorithmic developments. Examples include, hybrid subdivision, hybrid wavelets, hybrid mesh processing, hybrid mesh editing, and hybrid mesh compression.

Acknowledgments

This work was supported in part by NSF (DMS-9875042, ACI-9721349, ACI-9982273, DMS-9872890), Lucent, a Packard Fellowship, and generous grants from Alias|Wavefront, Microsoft, Intel, and Pixar. We would like to thank Zoë Wood and Cici Koenig for production help. The Buddha dataset is courtesy Stanford Univesity, David's head is courtesy of the Digital Michelangelo Project, Stanford University. The dataset of Max Planck was provided by Leif Kobbelt and the foot bone dataset derived from an original due to Viewpoint Digital.

References

[1] AKLEMAN, E., AND CHEN, J. Guaranteeing 2-manifold property for meshes. In *Proceedings of the Shape Modeling International* (1999), pp. 18–25.

[2] AKLEMAN, E., CHEN, J., AND SRINIVASAN, V. A new paradigm for changing topology during subdivision modeling. In *Proceedings of Pacific Graphics* (2000).

[3] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution analysis of arbitrary meshes. *Proceedings of SIGGRAPH 95* (1995), 173–182.

[4] EL-SANA, J., AND VARSHNEY, A. Topology simplification for polygonal virtual environments. *IEEE Transactions on Visualization and Computer Graphics 4*, 2 (1998), 133–144.

[5] FLOATER, M. S. Parameterization and smooth approximation of surface triangulations. *Computer Aided Geometric Design 14* (1997), 231–250.

[6] GARLAND, M., AND HECKBERT, P. S. Surface simplification using quadric error metrics. *Proceedings of SIGGRAPH 97* (1997), 209–216.

[7] GOLUB, G. H., AND LOAN, C. F. V. *Matrix Computations*, third ed. The Johns Hopkins University Press, 1996.

[8] GONZALEZ-OCHOA, C., AND PETERS, J. Localized-hierarchy surface splines (less). *ACM Symposium on Interactive 3D Graphics* (1999), 7–16.

[9] GUSKOV, I., VIDIMČE, K., SWELDENS, W., AND SCHRÖDER, P. Normal meshes. *Proceedings of SIGGRAPH 00* (2000), 95–102.

[10] GUSKOV, I., AND WOOD, Z. Topological noise removal. submitted for publication, 2000.

[11] HOPPE, H. Progressive meshes. *Proceedings of SIGGRAPH 96* (1996), 99–108.

[12] HOPPE, H., DEROSE, T., DUCHAMP, T., HALSTEAD, M., JIN, H., MCDONALD, J., SCHWEITZER, J., AND STUETZLE, W. Piecewise smooth surface reconstruction. *Proceedings of SIGGRAPH 94* (1994), 295–302.

[13] KHODAKOVSKY, A., SCHRÖDER, P., AND SWELDENS, W. Progressive geometry compression. *Proceedings of SIGGRAPH 00* (2000), 271–278.

[14] KIMMEL, R., AND SETHIAN, J. Fast marching method on triangulated domains. In *Proceedings of the National Academy of Science* (1998), vol. 95, pp. 8341–8435.

[15] KOBBELT, L. $\sqrt{3}$ subdivision. *Proceedings of SIGGRAPH 2000* (2000), 103–112.

[16] KOBBELT, L. P., BAREUTHER, T., AND SEIDEL, H.-P. Multiresolution shape deformations for meshes with dynamic vertex connectivity. *Computer Graphics Forum 19*, 3 (2000), 249–260.

[17] KRISHNAMURTHY, V., AND LEVOY, M. Fitting smooth surfaces to dense polygon meshes. *Proceedings of SIGGRAPH 96* (1996), 313–324.

[18] LEE, A. W. F., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. Maps: Multiresolution adaptive parameterization of surfaces. *Proceedings of SIGGRAPH 98* (1998), 95–104.

[19] LEVOY, M., PULLI, K., CURLESS, B., RUSINKIEWICZ, S., KOLLER, D., PEREIRA, L., GINZTON, M., ANDERSON, S., DAVIS, J., GINSBERG, J., SHADE, J., AND FULK, D. The digital michelangelo project: 3d scanning of large statues. *Proceedings of SIGGRAPH 00* (2000), 131–144.

[20] LORENSEN, W. E., AND CLINE, H. E. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics (Proceedings of SIGGRAPH 87) 21*, 4 (1987), 163–169.

[21] LOUNSBERY, M., DEROSE, T. D., AND WARREN, J. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics 16*, 1 (1997), 34–73.

[22] POPOVIC, J., AND HOPPE, H. Progressive simplicial complexes. *Proceedings of SIGGRAPH 97* (1997), 217–224.

[23] SAMET, H. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[24] VELHO, L., AND ZORIN, D. 4-8 subdivision. *Computer Aided Geometric Design* (2001). to appear.