Symmetry-Aware Mesh Processing

Aleksey Golovinskiy, Joshua Podolak, and Thomas Funkhouser

Department of Computer Science, Princeton University
35 Olden Street, Princeton, NJ, 08540
{agolovin,jpodolak,funk}@cs.princeton.edu

Abstract. Perfect, partial, and approximate symmetries are pervasive in 3D surface meshes of real-world objects. However, current digital geometry processing algorithms generally ignore them, instead focusing on local shape features and differential surface properties. This paper investigates how detection of large-scale symmetries can be used to guide processing of 3D meshes. It investigates a framework for mesh processing that includes steps for symmetrization (applying a warp to make a surface more symmetric) and symmetric remeshing (approximating a surface with a mesh having symmetric topology). These steps can be used to enhance the symmetries of a mesh, to decompose a mesh into its symmetric parts and asymmetric residuals, and to establish correspondences between symmetric mesh features. Applications are demonstrated for modeling, beautification, and simplification of nearly symmetric surfaces.

Key words: symmetry analysis, mesh processing

1 Introduction

Symmetry is ubiquitous in our world. Almost all man-made objects are composed exclusively of symmetric parts, and many organic structures are nearly symmetric (e.g., bodies of animals, leaves of trees, etc.). It is almost impossible to find a real-world object that does not have at least one nearly perfect symmetry and/or is not composed of symmetric parts. Moreover, symmetry is an important cue for shape recognition [1], as humans readily notice departures from perfect symmetry.

For decades, however, mesh reconstruction and processing algorithms in computer graphics have largely ignored symmetries. Most algorithms operate as sequences of mesh processing operations based on local shape features and/or differential surface properties. As a result, they have difficulty reproducing and preserving global shape properties, such as symmetry.

Consider simplification, for example – when presented with an input mesh for a nearly symmetric object (e.g., a face), a simplification algorithm should produce a nearly symmetric mesh. However, to our knowledge, there is no current algorithm that satisfies this basic requirement. Certainly, if the input is perfectly symmetric, then the problem is trivial – simply process half of the mesh and then

copy the result. However, if the underlying surface is symmetric but the mesh topology is not, or if the underlying surface is only approximately symmetric, then standard simplification algorithms fail to preserve symmetries present in the underlying object (Figure 10c). The result is potential artifacts in physical simulations, manufacturing processes, animations, and rendered images (e.g., asymmetric specular highlights).

Recently, researchers have introduced several methods for detecting and characterizing the symmetries in 3D data. For example, Zabrodsky et al. [2] provided a measure of approximate symmetry with respect to any transformation, and Mitra et al. [3] and Podolak et al. [4] have described algorithms for extracting the most significant approximate and partial symmetries of a 3D mesh. While symmetry analysis methods like these have been used to guide high-level geometric processing operations, such as registration, matching, segmentation, reconstruction, reverse engineering, editing, and completion, they have not yet been incorporated into low-level mesh processing algorithms.

The main goal of this paper is to investigate ways in which symmetry analysis can guide the representation and processing of 3D surface meshes. To support this goal, we make the following contributions. First, we describe an algorithm for geometric symmetrization – i.e., deforming a surface to respect a given set of symmetries while retaining its shape as best as possible. Second, we describe an algorithm for topological symmetrization – i.e., remeshing a surface so that symmetric regions have consistent mesh topology. Third, we propose a "symmetry-aware" mesh processing framework in which geometric and/or topological symmetrization algorithms provide high-level shape information (symmetric correspondences and asymmetric residuals) that can guide mesh processing applications to produce more symmetric results for approximately symmetric inputs. Finally, we demonstrate applications of this framework for surface beautification, symmetry enhancement, attribute transfer, and simplification.

2 Background and Previous Work

Understanding the symmetries of shapes is a well studied problem with applications in many disciplines. Perfect symmetries are common in CAD models and used to guide compression, editing, and instancing [5]. However, only considering perfect symmetries is of limited use in geometric processing, in general. First, the presence of noise, numerical round-off error, or small differences in tessellation can cause models of objects that are in fact symmetric to lack perfect symmetry. Second, many asymmetric objects are composed of connected parts with different symmetries. Finally, most organic objects exhibit near, but imperfect, symmetries (leaves of trees, human bodies, etc.), and understanding those types of symmetry is important, too. Thus, it is useful to have methods to detect and utilize partial and approximate symmetries.

Towards this end, Zabrodsky et al. [2] defined the *symmetry distance* of a shape with respect to a transformation as the distance from the given shape to the closest shape that is perfectly symmetric with respect to that transformation.

They provide an algorithm to find the symmetry distance for a set of connected points for any given reflective or rotational transformation. Mitra et al. [3] and Podolak et al. [4] find a set of prominent symmetries by having points on a mesh vote for symmetries in a process similar to a Hough transform.

Measures for partial and approximate symmetry of this type have been used in a variety of computer vision applications. Perhaps the earliest example is by [6], who used deformable models with symmetry-seeking forces to reconstruct 3D surfaces from 2D images. Zabrodsky et al. used a continuous measure of symmetry for completing the outline of partially-occluded 2D contours [7], for locating faces in an image, determining the orientation of a 3D shape [2], for reconstructing 3D models from images, and for symmetrizing 3D surfaces [8].

More recently, symmetry analysis has received attention in computer graphics. Kazhdan et al. [9] constructed a symmetry descriptor and used it for registration and matching. Podolak et al. [4] used a symmetry transform for surface registration, shape matching, mesh segmentation, and viewpoint selection. Mitra et al. [3] described a method to extract a discrete set of significant symmetries and used them for segmentation and editing. Thrun et al. [10] used local symmetries and used them for completion. Gal et al. [11] developed local shape descriptors to look for approximate symmetries in 3D surfaces and used them for visualization and matching. Mills et al. [12] utilized approximate symmetries to guide reverse engineering of CAD structures from range scans. Simari et al. [13] decomposed meshes into a hierarchical tree of symmetric parts to be used for compression and segmentation. Finally, Martinet et al. [5] has detected perfect symmetries in parts of scenes and used them to build instancing hierarchies.

Perhaps closest to our work is the work of Mitra et al. [14], where a method of symmetrizing the geometry of meshes is presented. Their technique of symmetrization is similar to the method we present in Section 4. However, whereas their work focuses on symmetrizing geometry, we aim to develop a pipeline for making mesh-processing algorithms symmetry-aware, of which symmetrizing geometry is one step.

3 Overview

The goal of our work is to provide tools for symmetry-aware processing of 3D surface meshes. We propose a multi-step processing framework, in which approximate and partial symmetries are detected, preserved, exploited, and sometimes even enhanced as meshes are processed. To support this framework, we provide the following tools, which typically will be used in the sequence of steps shown in Figure 1:

1. **Symmetry analysis:** the mesh is analyzed to detect perfect, approximate, and partial symmetries. The output of this step is a set of transformations (e.g., planes of reflection), each with a list of vertices indicating the subset of the surface mapped approximately onto itself by the transformation. For this step, which is not a focus of this paper, we use methods previously described in [4] and [3].

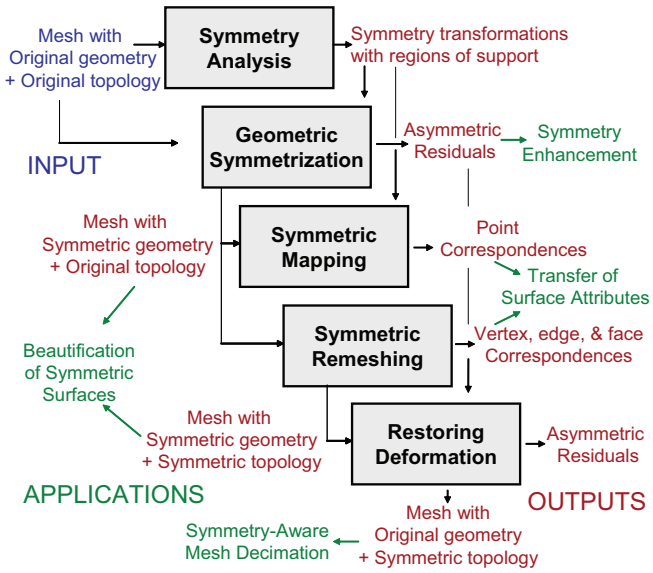


Fig. 1. Symmetry-aware mesh processing framework.

2. **Geometric symmetrization:** the surface is warped to make it symmetric with respect to a given set of transformations. The primary output of this step is a "symmetrized" mesh having the same topology as the original, but with geometry that is perfectly symmetric up to the resolution of its tessellation. A secondary output is a set of asymmetric residuals storing the vector difference between the original and symmetrized position of every vertex, which can be used to compute an inverse to the symmetrizing warp.
3. **Symmetric mapping:** correspondences are established between vertices and their images across every symmetry transformation. The output of this step is a dense set of point pairs, where one point is associated with a vertex and the other is associated with a face and its barycentric coordinates. These point pairs provide a mapping between symmetric surface patches.
4. **Symmetric remeshing:** the surface is remeshed so that every vertex, edge, and face has a one-to-one correspondence with another across every symmetry. The output of this step is a new mesh with perfectly symmetric topology, along with a list of topological correspondences.
5. **Restoring deformation:** the inverse of the symmetrizing warp is applied to the symmetrically remeshed surface to restore the original geometry. The output of this step is a mesh that is topologically symmetric, but geometrically approximates the input mesh.

The motivating idea behind this framework to provide tools that can help mesh processing algorithms to preserve large-scale symmetries present in 3D objects. Our general strategy is to factor a 3D surface into a symmetric mesh and its asymmetric residual and then to perform analysis on the symmetric

mesh to gain insight into its symmetric structure. We transfer knowledge about symmetric structure back onto the original geometry so that it can be preserved and exploited as the surface is processed.

In the following sections, we investigate algorithms to support this symmetry-aware mesh processing framework, focusing on the most challenging steps: geometric symmetrization (Section 4) and symmetric remeshing (Section 5). Thereafter, in Section 6, we describe potential applications and present prototype results. Finally, we conclude with a discussion of limitations and topics for future work.

4 Geometric Symmetrization

Our first objective is to provide an algorithm that can take a given surface mesh and output a new mesh with similar shape that is symmetric with respect to a given set of transformations. More formally, given a mesh M and a set of symmetry transformations, each having a possibly local region of support on the mesh, our goal is to find the most shape-preserving warp W that produces a new mesh M' with the same topology as M, but where every vertex of M' is mapped onto corresponding points on the surface of M' by all of its symmetry transformations.

This objective is similar to classical problems in non-rigid alignment for morphing of 3D surfaces, medical imaging, surface reconstruction, and several other fields. The challenge is finding both symmetric point correspondences and the warp that aligns them simultaneously. Following traditional iterative approaches [15], we greedily minimize alignment error while allowing increasingly non-rigid deformation [16–18]. At each iteration, we first propose correspondences from every vertex in the mesh M to its closest compatible point on the transformed surface for every symmetry. Then, given these correspondences, we solve for new vertex positions that minimize a symmetrizing error function (Figure 2). These two steps are iterated until the mesh is fully symmetric (i.e., every vertex transformed by all its symmetries produces a point directly on a face of the mesh).

Our symmetrizing error function balances the primary goal of making the surface more symmetric with the secondary goals of retaining its original shape and position with three error terms:

$$E(M) = \alpha E_{sym}(M) + (1 - \alpha)(E_{shape}(M) + \beta E_{disp}(M))$$

The first two error terms are the important ones, as they balance the trade-off between deviations from perfect symmetry and deformations of the surface. The first term, E_{sym}, measures the sum of squared distances between vertices of the mesh and the closest point on the transformed surface. For the second term, E_{shape}, we use the shape preservation function proposed by [18], which minimizes a measure of warp distortion. Any deformation error function would work, but this choice has the advantage of being quadratic in vertex positions. This deformation error is not rotationally invariant, but the symmetrizations

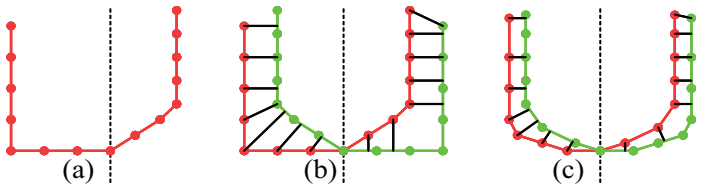


Fig. 2. Schematic of one iteration in our symmetrization process (in 2D). (a) Given a curve (red) and a symmetry transformation (reflection across the dotted line), (b) we find correspondences between vertices and the closest to point on the reflected curve (green), and (c) solve for new vertex positions that minimize an error function based on those correspondences.

performed in our experiments do not have large rotational components. The last term, E_{disp}, measures overall displacement using the sum of squared distances between current and original vertex positions. It is required to penalize global translations because E_{shape} is translationally invariant and because the surface is being warped onto itself (in contrast to traditional alignment problems where either the source or target is fixed in space). We set the weight of this error term very low relative to the others ($\beta = 1/100$), and so it has very little influence on the output surface's shape. Since all three terms of the error function are quadratic in the positions of the vertices, we can solve for the minimal error at each iteration with a least-squares solution to a linear system.

Our implementation contains several simple features that help provide stability and speed as the optimization proceeds. First, it uses multiresolution surface approximations to accelerate convergence and avoid local minima. Prior to the optimization, the input mesh is decimated with *Qslim* [19] to several nested levels of detail. Then, coarser levels are fully symmetrized and used to seed the initial vertex placements for finer levels (new vertices added at each finer level are positioned relative to the current ones using thin-plate splines). Within each level, further stability is gained by slowly shifting emphasis of the error function from shape preservation ($\alpha = 0$) to full symmetrization ($\alpha = 1$). Finally, for each vertex, we use a k-d tree to help find the closest point on the transformed surface, and only retain correspondences to a closest point whose transformed normal does not point in the opposite direction. Overall, compute times on a 3Ghz processor range from 4 seconds for the 1,166 vertex model of the dragon in Figure 8 to 10 minutes for a 240,153 vertex face scan.

For instance, consider Figure 3, which shows the result of symmetrizing a bust of Max Plank with respect to reflection across a single vertical left-right plane. Note that the original input mesh (Figure 3a) is quite asymmetric, as seen by the misalignment of the surface (red) and its reflection (green) in the bottom images of Figure 3a – i.e., significant shape features (eyes and ears) do not map onto their symmetric counterparts when reflected across the plane. However, we are able to find a non-rigid warp that aligns those features while producing

a symmetric mesh with a small amount of shape distortion. The symmetrized mesh, shown Figure 3b, is perfectly symmetric up to the resolution of the mesh, as indicated by the high-frequency interleaving of the original surface (red) with its reflection (green) in the bottom images of Figure 3b.

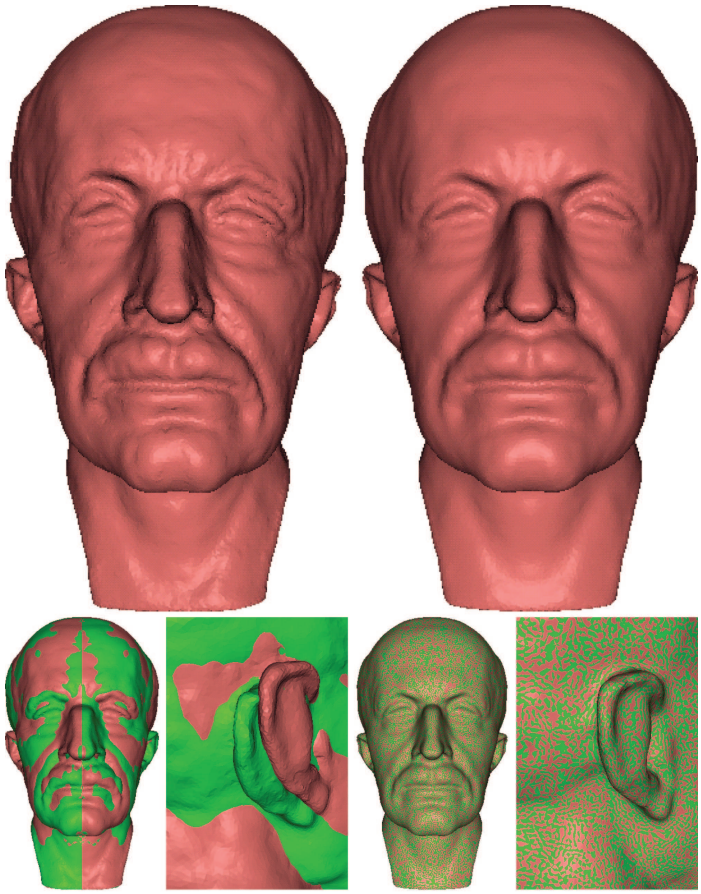


Fig. 3. Symmetrizing Max Planck. The model of a bust of Max Planck (a) asymetric, as can be seen by overlaying the mesh (red) with its reflection (green) in the bottom images. Our method symmetrizes its geometry (b), while retaining and aligning sharp features like eyes, mouth, and ears.)

.

A more complicated example demonstrates symmetrization across partial, approximate planes of symmetry in the Stanford Bunny (Figure 4). Using the method of [4], the bunny was automatically segmented into two symmetric parts

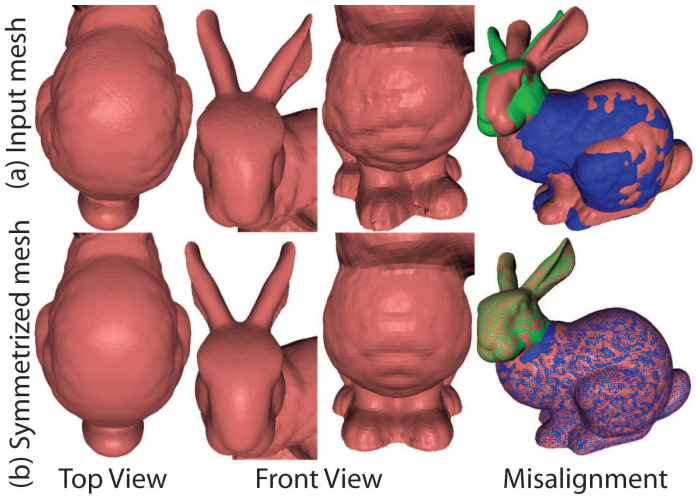


Fig. 4. Symmetrizing the bunny. The top row shows the original bunny, while the bottom row shows the symmetrized result. The bunny contains partial symmetries with respect to planes through the body and through the head. Our method symmetrizes both parts of the bunny while performing a shape-preserving blend in between, preserving features such as the eyes and feet. The right pair of images show the asymmetry of the original model and the accurate alignment of symmetric parts in our result.

(the head and the body), each supporting a plane of partial symmetry. Of course, both of these parts are highly asymmetric, as can be seen from top-right image in Figure 4, where each part of the bunny (red) is shown along with its reflection (green and blue) over its symmetry plane. Note how poorly the ears and feet align with their reflections. However, our geometric symmetrization algorithm is able to warp both parts into alignment with their reflections simultaneously while retaining significant shape features (e.g., ears, feet) and blending symmetries across the intermediate region of the neck. Note that the crease between the feet is preserved although the symmetry plane does not run through it on the original model, as the surface was warped to align with the plane.

The output of the process is not only a symmetrized mesh, but also a *symmetric map*, a set of point correspondences where every vertex is associated with a point on the surface across every symmetry transformation. We store the corresponding points in barycentric coordinates with respect to triangles of the mesh so that they deform with the surface (e.g., when we apply the inverse of the symmetrizing deformation to restore the original geometry). This is a key point, since it allows us to transfer the symmetric map learned from the perfectly symmetric surface back to the asymmetric one.

5 Symmetric Remeshing

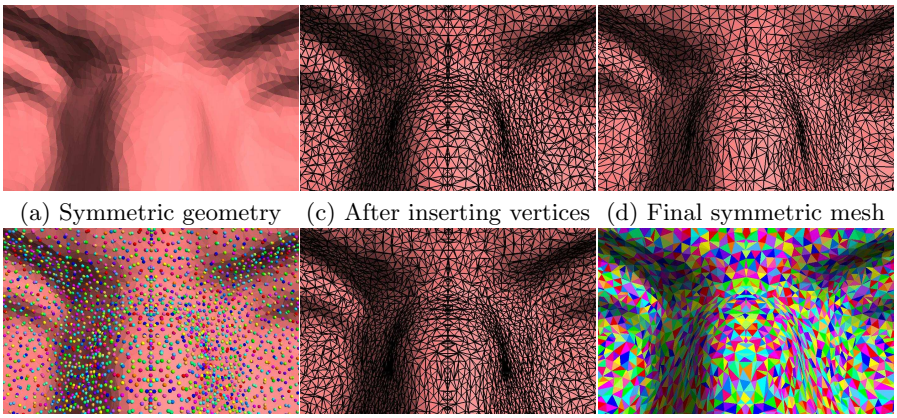
Our second objective is to develop an algorithm that can take a geometrically symmetrized mesh with arbitrary topology and remesh it so that every vertex, edge, and face has a direct correspondence with another with respect to every symmetry transformation. Our motivation is to provide a topology that not only reflects the symmetric structures of the object, but also can provides efficiency in representation and manipulation due to topological redundancies (e.g., compression), cues for preservation of symmetries during topological modifications (e.g., simplification), and symmetric sampling to avoid assymetric artifacts in photo-realistic renderings and physical simulations (e.g., boundary element methods).

This problem is a special case of compatible remeshing [20, 21]. Given the symmetric map from every vertex to a point on the surface for every symmetry transformation provided by the geometric symmetrization algorithm, we aim to find the mesh with perfectly symmetric topology that has the least geometric error and/or fewest extra vertices.

A strawman approach that may be appropriate for highly symmetric and/or oversampled meshes is to partition the mesh into "asymmetric units" and then copy the topology from one instance of others in correspondence and then stitch at the boundaries. For a single planar reflection, this would entail cutting the mesh along the plane, throwing away the mesh connectivity on one side (M_t), and then copying the connectivity over from the other side (M_s). While this simple method would provide symmetric topology with the same number of vertices as the original mesh, it would produce an asymmetry in the quality of the geometric approximation (M_s would have the quality of the original surface, but M_t would have blurring where edges oriented appropriately for the geometry of M_s are not appropriate for M_t), and it would produce artifacts where the topology of M_s provides a poor approximation for M_t.

There are many methods in the literature to overcome this problem, most of which introduce a large number of extra vertices to capture the geometric variations of both M_s and M_t. For example, one way is to create an overlay meta-mesh that contains the original vertices of both M_s and M_t along with new vertices at all edge-edge intersections [22, 23]. Another way is to map M_s and M_t to a common base domain (e.g., a sphere [22], or a simplified triangle mesh [23, 24]) and then remesh with semi-regular connectivity until all geometric features are resolved. Alternatively, it is possible to create a meta-mesh M_{st} by inserting all the vertices of M_s into M_t, and vice-versa, and then iteratively swapping edges until a compatible mesh topology is achieved [20]. These methods all produce compatible mesh topology and so could be used for symmetric remeshing. However, the resulting mesh would usually be significantly over-sampled.

We provide a simple method to address the problem: *compatibility-preserving mesh decimation* (or, in our case, *symmetry-preserving mesh decimation*). Our general approach is to use any of the above methods to produce compatible mesh topology with vertices from both M_s and M_t, and then to decimate the resulting meta-mesh with a series of edge collapse operations that operate on corresponding edges in lock-step. Specifically, we build clusters of edges whose



(a) Symmetric geometry (c) After inserting vertices (d) Final symmetric mesh

(b) Point correspondences (d) After edge flips (e) Final symmetric mesh

Fig. 5. Symmetric remeshing. The input is shown in the left column (the symmetrized bust of Max Planck, zoomed to the bridge of the nose); intermediate steps are shown in the middle column, and the output is shown in the right column. The final mesh has perfectly symmetric topology (as shown by colored face correspondences in (e)) and still approximates the surface well with the original number of faces.

vertices are in symmetric correspondence and then follow the same basic approach as the original *Qslim* algorithm [19], however working on clusters rather than individual edges. We load the clusters into a priority queue sorted by the Quadric Error Measure (QEM) of the edge with highest error in each cluster, and then we iteratively collapse all edges in the cluster with minimal error until a desired number of triangles or a maximum error has been reached. Since all edges in the same cluster are processed atomically, the method is guaranteed to maintain topological symmetries as it decimates the mesh. Yet, it still provides a good approximation of the original surface, as QEMs approximate deviation from the original surface.

This method is similar in goal to the method of [21], which copies the mesh topology of M_s onto M_t and then optimizes the positions and number of vertices to match the geometry of both M_s and M_t with a combination of smoothing and refinement operations. The difference is that we first produce an over-sampled mesh with vertices from both M_s and M_t, and then "optimize" it to minimize the QEM by decimation. Since our process is seeded directly with (a conservatively large set of) compatible vertices and edges from both M_s and M_t, the optimization starts from an initial configuration that encodes features from the entire mesh. So, our challenge is mainly to decide which vertices and edges can be removed, rather than discovering suitable places for new vertices from scratch. As a result, it is easy to produce compatible mesh topologies for any number of surface regions with any number of vertices.

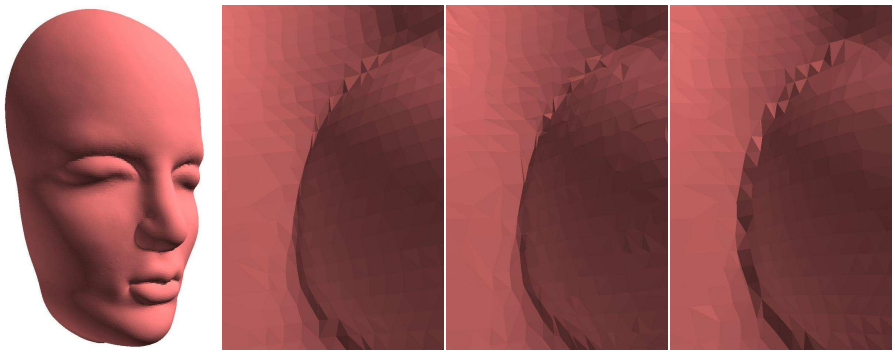
We have experimented with this approach using an algorithm based on the Connectivity Transformation technique of [20] to form an over-sampled

mesh with symmetric topology prior to decimation. Given a geometrically symmetrized mesh and a set of vertex-point correspondences (Figure 5a-b), we first produce a meta-mesh M_{st} with symmetric vertex correspondences by inserting all the vertices of M_s into M_t, and vice-versa, splitting faces into three when an inserted vertex maps to the interior of an existing face (Figure 5c). We then swap edges in order of an error function that measures the differences in the QEM for edge midpoints before and after the swap, plus a quadratically growing penalty for swaps of an edge multiple times, plus an infinite penalty for any swap that would generate a topological fin in the mesh or break a greater number of symmetric correspondences than it creates. The process terminates when all edges are found to be in symmetric correspondence, or when the minimal error of any cluster exceeds some preset threshold. We have not implemented the edge-crossing constraint and termination criterion of [20], as it only guarantees convergence for meshes on a plane [25]. However, we find that our method finds symmetric correspondences for all but few edges in practice (99.9% in all of our examples). For the remaining edges, we simply copy those edges from one asymmetric unit to the other(s). The net result is a mesh with fully symmetric topology containing approximately twice as many vertices as the original (Figure 5d). We give that mesh as input to the symmetry-preserving version of Qslim to produce the final result – a topologically symmetric mesh with a user-specified number of faces or geometric error. Figure 5e-f shows the result for decimation to the number of faces in the original mesh (98K). Compute times for the entire symmetric remeshing process range between tens seconds for the dragon and two hours for the armadillo on a 3GHz processor. Of course, this process must be done only once per model.

It is difficult to make comparisons of our symmetric meshing method to others, since our problem is somewhat different from previous ones. However, to validate that our approach provides benefits over the simple strawman approach described earlier in the section (copy the topology of the right side over to the left), we provide a comparison of symmetrically remeshed surfaces of a mask along the left crease of the nose (Figure 6). Note how our method (middle) produces a mesh that retains sharp features of the original (left), whereas the simpler approach (right) suffers from blurring due to poorly oriented edges. Besides these differences in surface quality, our method has the additional advantage that it works without modification for partial and multiple symmetries of any type of transformation, and produces symmetric mesh topology at any user-selected face count.

6 Applications

The main theme of this paper is that awareness of symmetries can and should be incorporated into mesh processing algorithms. Since objects with perfect and/or approximate symmetries are prevalent in our world, and since symmetries are often critical to an object's function and/or a human's perception of it, we believe that algorithms processing 3D models should understand their symmetries and



(a) Original Mesh (b) Symmetric Remeshing (c) Copy topology from right to left

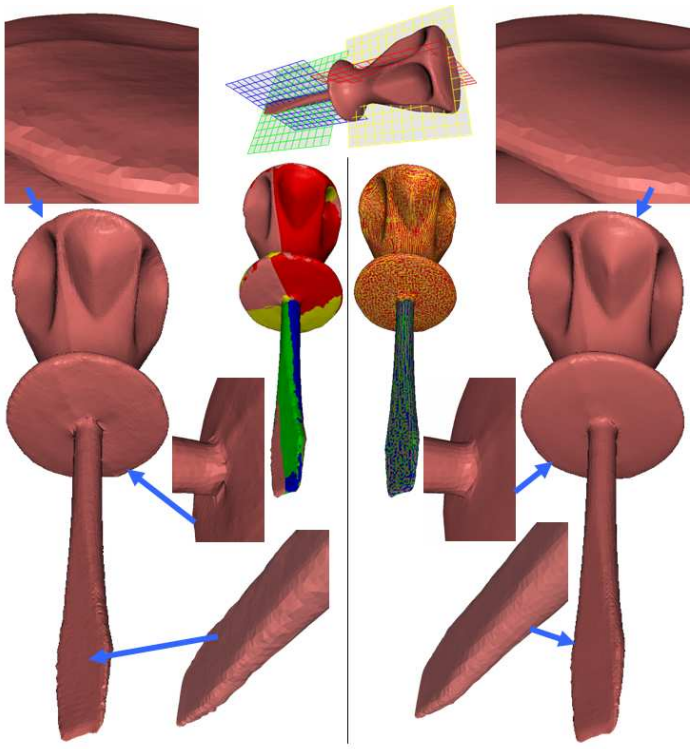
Fig. 6. A zoomed-in comparison the crease along left side of the nose on the surface of the mask shown on the left. Note how our approach (b) does not produce blurring when compared to the original (a), while the simple alternative (c) of copying topology from one side to the other does.

preserve them. In this section, we investigate how this can be done for several classes of applications.

Roughly speaking, applications can be divided into classes according to what type of mesh data they process, and almost equivalently, what type of symmetry information they can exploit: (1) Some applications are concerned mainly with creating new geometry (e.g., surface scanning, interactive modeling, etc.). For this class, geometric symmetrization provides a useful tool for coercing the geometry of approximate input (e.g., scanned points, sketched surfaces, etc.) to become more, less, or perfectly symmetric to match the intended structure of the object being modeled. (2) Other applications are concerned with manipulating attributes associated with local regions of a surface (e.g., texture mapping, signal processing, etc.). For them, symmetric mapping provides a way to blend and transfer attributes between symmetric regions. (3) Still other applications are concerned with the manipulating the topology of a mesh (e.g., remeshing). For those applications, symmetric remeshing provides an automatic way to coerce the mesh topology to respect the symmetric structure of an object and provides correspondence information that can be used to preserve topological symmetries as the mesh is processed further. Finally, of course, there are applications that can exploit all three types of symmetry information simultaneously (e.g., beautification, compression, etc.). In the following subsections, we show at least one example from each of these classes.

6.1 Beautification of Meshes for Symmetric Objects

There are many application domains in which scans are acquired for symmetric real-world objects. For example, in rapid prototyping applications, physical mockups are often constructed for a proposed design (e.g., with clay) and then



(a) Original mesh (b) Symmetrized mesh

Fig. 7. Symmetrizing a scanned screwdriver model. The input mesh is shown on the left side, and the output mesh is on the right. Note that the output mesh is perfectly symmetric, has less noise (e.g., on the tip and at the junction with the handle), and retains sharp features (e.g., the ridge on the top of the handle).

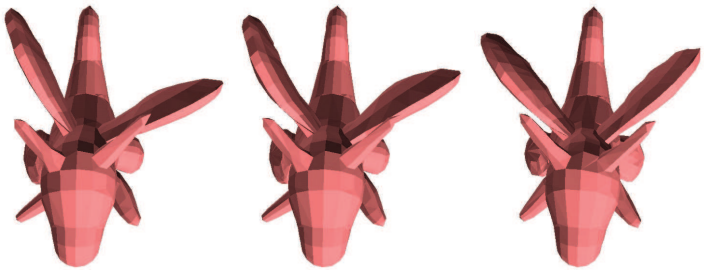
scanned for computer simulation and processing. Likewise, in reverse engineering, objects are scanned when the original design is not available. However, rarely are the scanned models perfectly symmetric, due in part to scanner bias and noise, and due in part to processing tools that introduce asymmetries as a surface mesh is reconstructed. Since so many scanned objects are in fact symmetric, it seems useful to have a tool that takes a scanned mesh as input and produces the most similar symmetric mesh as output.

As an example, consider the scanned screwdriver downloaded from the Cyberware repository of Desktop 3D Scanner Samples (left side of Figure 7). In this case, the physical object has two parts (handle and tip), each of which is approximately symmetric with respect to two plane reflections (top-middle of Figure 7). Yet, the scanned mesh contains significant asymmetries with respect to all of these planes (e.g., artifacts at the junction of the tip and the handle).

Motivated by the idea of "beautifying" this mesh, we extracted planes of symmetry automatically with the Iterative Symmetric Points algorithm of [4], augmented to ensure that pairs of planes for the same part were perpendicular, and that all four planes aligned on a single axis (note that the planes for the handle are rotated by 20 degrees with respect to those of the tip). Then, we ran our geometric symmetrization algorithm on the entire mesh, with all four planes of symmetry guiding the surface deformation.

The result is shown in the images on the right side of Figure 7. Looking closely at the image in the middle right, it can be verified that the surface is symmetric up to the resolution of the mesh (note the high-frequency interleaved pattern of yellow, red, green, and blue overlaid surfaces). It can also be seen that significant shape features are retained during symmetrization (e.g., the ridge in the top of the handle), while noise is reduced (e.g., the tip shown in close-up on the bottom right). In general, shape features that align across multiple symmetries are retained, while those that do not are diminished. Overall, the mesh on the right of Figure 7 has the principal symmetries of the physical object and lower levels of noise, and thus is probably preferable for most simulation and visualization applications.

6.2 Symmetry Enhancement



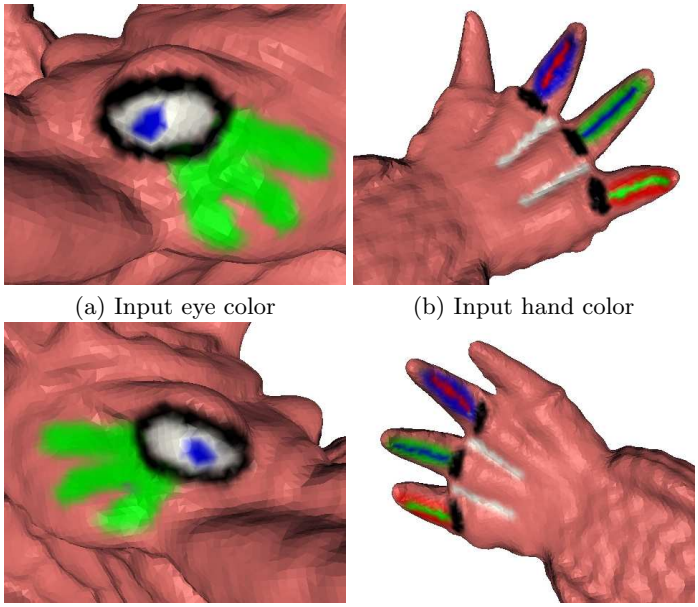
(a) Original (b) Symmetrized Part-way (c) Symmetrized Completely

Fig. 8. Enhancing the symmetries of a sketched model under interactive control.

In some applications, it may not be desirable to symmetrize a surface completely, but rather to enhance or to diminish symmetries instead. As a concrete example, imagine that a person has drawn the dragon shown in Figure 8a using a sketching tool like Teddy [26], but wants to make it more symmetric (note that the wings are quite misaligned with respect to the left-right symmetry plane). While this type of operation is possible with a series of deformations and local surface edits, it would be tedious with current modeling tools.

Instead, we propose an interactive tool that allows a user to control the degree of symmetrization applied to a surface. We provide a slider that the user can manipulate to make a surface more or less symmetric with respect to a selected transformation while the model is updated with real-time visual feedback. As an example, Figure 8b-c shows screenshots after the user has interactively symmetrized the dragon part-way (middle) and completely (right). In this simple case, the symmetrizing deformation could be computed in real-time. For more complex models, symmetry enhancement can be performed in real-time following symmetrization as a pre-process (see the video for examples). We believe that such a tool would be a useful addition to the suite of commands for interactive surface design.

6.3 Attribute Transfer



(a) Input eye color (b) Input hand color

(c) Transferred eye color (d) Transferred hand color

Fig. 9. Transferring surface attributes between symmetric parts.

There are many applications that require blending or transferring attributes between semantically related surface regions – for example, texture transfer, denoising, and morphing. The challenge is usually to establish correspondences between semantically related parts. In the case of objects with approximate symmetries, symmetric mapping provides a useful way to solve this problem.

For example, consider the Armadillo model. Although the surface is "semantically symmetric" (e.g., the arm on the left has a functional correspondence with the one on the right), the surface is not symmetric geometrically (e.g., the arms are in significantly different poses). In cases like this, our symmetrization framework provides a natural way to establish correspondences between approximately symmetric parts via symmetric mapping.

This mapping can be used to transfer and blend surface attributes. For example, Figure 9 shows a demonstration of transferring per-vertex colors between symmetric regions of the Armadillo model. In the top row, the user has drawn colors on the eyes and hands on one side of the surface with an interactive painting interface. The system then automatically transfers the colors to the other side (bottom row) via an automatically generated symmetric map.

In this example, the main benefit is to save the user the effort of painting details twice. However, in other examples, perhaps it is important that the surface attributes are applied to both sides in exactly the same way, or that surface details are blended very precisely, which would be difficult without guidance from a symmetric map.

6.4 Simplification

Simplification algorithms take a mesh and produce an approximation with fewer polygons, usually to increase rendering speed, decrease storage, and/or provide a base domain for parameterization. Generally, however, they do not preserve large-scale symmetries (or other global shape features), in favor of minimizing local geometric errors.

In this section, we investigate whether the symmetry-preserving mesh decimation algorithm described in Section 5 can be used effectively for extreme simplification of approximately symmetric surfaces. Following the general approach outlined in Section 3, we establish symmetric topology for an asymmetric surface by first symmetrizing it, remeshing with symmetric topology, and then warping the new topology and correspondences back to the original geometry. We then perform symmetry-preserving mesh decimation on the symmetric topology over the asymmetric mesh.

Figure 10 shows the results of this method (first two columns) in comparison to the original version of *Qslim* (last column). Note that the topology of the mesh output by our algorithm is perfectly symmetric, even though the geometry of the surface is not. Note also that the geometric approximation achieved with symmetry-aware simplification is similar to the original (according to Metro [27], it has a Hausdorff distance approximately 6% larger). Since the symmetric mesh better reflects the semantic structure of the surface, we believe it may be preferable as a base domain for parameterization, animation, simulation, and other applications.

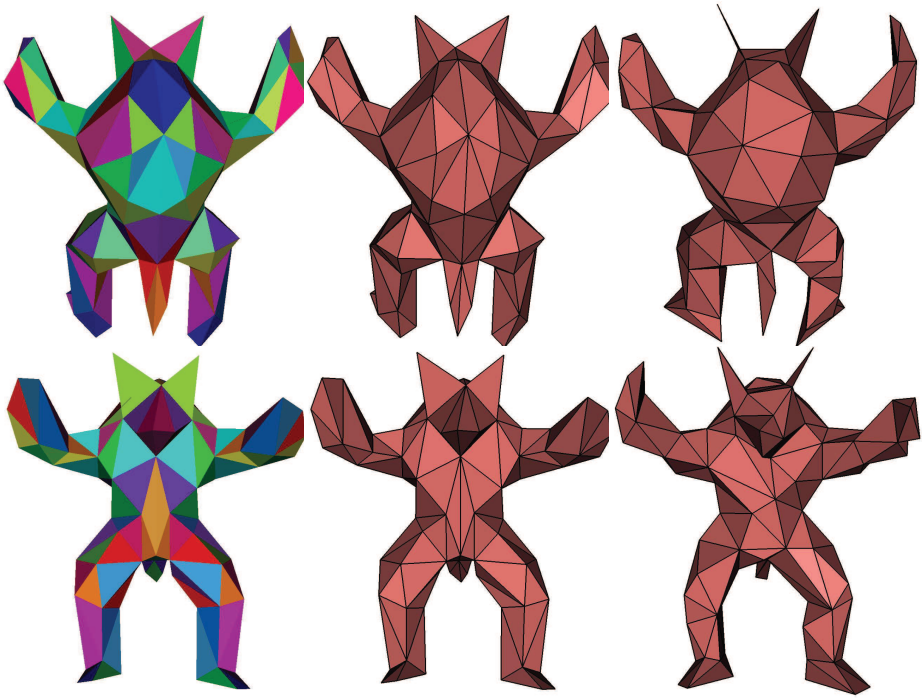


Fig. 10. Symmetry-preserving QSlim (a-b) produces a surface approximation comparable to the original algorithm (c) of [19], but guaranteed to have symmetric topology, even for an asymmetric surface (the first column (a) shows symmetric face correspondences preserved during the decimation).

7 Conclusion

In summary, this paper has investigated methods for and applications of symmetrizing 3D surface meshes. The main idea is that symmetry-aware algorithms can be used to preserve, exploit, and enhance structural symmetries of a surface, even if the underlying geometry is only approximately symmetric. This idea is important because the vast majority of objects in the world have some sort of structural symmetries, and current mesh processing algorithms generally do not preserve them.

The main contribution of this paper is the symmetry-aware mesh processing framework, which includes algorithms for geometric symmetrization and symmetric remeshing. We provide demonstration of the framework for mesh beautification, symmetry enhancement, attribute transfer, and simplification.

The initial results seem promising, but our implementation has limitations, which suggest immediate topics for future work. We have demonstrated our

algorithms only for symmetries across planar reflections. Although our code can handle symmetries for arbitrary affine transformations, we have not investigated examples of this type in our study.

Considering steps forward, the most obvious next step is to investigate other applications enabled by symmetry-aware processing. First candidates include compression and denoising. In the former case, it is possible that factoring a mesh into its symmetric part and its asymmetric residual could provide increased compression ratios, since at least half of the symmetric part can be discarded [13]. For denoising, the symmetric map could provide a way to blend noise across symmetric surfaces, as in Smoothing by Example [28]. These are just two examples – considering other applications that exploit symmetries will be a fruitful topic for future work.

The main long-term direction suggested by this work is that digital geometry processing algorithms can and should consider large-scale structural features as well as local surface properties when processing a mesh. So, future work should consider better ways to detect and encode large-scale shape features (such as symmetry) and to preserve and exploit them during surface processing.

Acknowledgments The authors would like to thank Forrester Cole and the Princeton Graphics group for useful discussions about this project. We are also grateful to several funding agencies that provided partial funding for this work: National Science Foundation Grants IIS-0612231, CCR-0093343, CNS-0406415, and 11S-0121446; Air Force Research Laboratory Grant FA8650-04-1-1718; and the Google Research Grants program.

References

1. Ferguson, R.W.: Modeling orientation effects in symmetry detection: The role of visual structure. In: Proc. Conf. Cognitive Science Society. (2000)
2. Zabrodsky, H., Peleg, S., Avnir, D.: Symmetry as a continuous feature. Trans. PAMI **17**(12) (1995) 1154–1166
3. Mitra, N.J., Guibas, L., Pauly, M.: Partial and approximate symmetry detection for 3D geometry. **25**(3) (2006) 560–568
4. Podolak, J., Shilane, P., Golovinskiy, A., Rusinkiewicz, S., Funkhouser, T.: A planar-reflective symmetry transform for 3D shapes. ACM Transactions on Graphics (Proc. Siggraph) **25**(3) (July 2006)
5. Martinet, A., Soler, C., Holzschuch, N., Sillion, F.: Accurately detecting symmetries of 3D shapes. Technical Report RR-5692, INRIA (September 2005)
6. Terzopoulos, D., Witkin, A., Kass, M.: Symmetry-seeking models and 3D object reconstruction. **3**(1) (1987) 221–221
7. Zabrodsky, H., Peleg, S., Avnir, D.: Completion of occluded shapes using symmetry. In: Proc. CVPR. (1993) 678–679
8. Zabrodsky, H., Weinshall, D.: Using bilateral symmetry to improve 3D reconstruction from image sequences. Comput. Vis. Image Underst. **67**(1) (1997) 48–57
9. Kazhdan, M., Chazelle, B., Dobkin, D., Funkhouser, T., Rusinkiewicz, S.: A reflective symmetry descriptor for 3D models. Algorithmica **38**(1) (October 2003)

10. Thrun, S., Wegbreit, B.: Shape from symmetry. In: Proceedings of the International Conference on Computer Vision (ICCV), Bejing, China, IEEE (2005)
11. Gal, R., Cohen-Or, D.: Salient geometric features for partial shape matching and similarity. In: ACM Transaction on Graphics. (2005)
12. Mills, B.I., Langbein, F.C., Marshall, A.D., Martin, R.R.: Approximate symmetry detection for reverse engineering. In: SMA '01: Proceedings of the sixth ACM symposium on Solid modeling and applications, New York, NY, USA, ACM Press (2001) 241–248
13. Simari, P., Kalogerakis, E., Singh, K.: Folding meshes: Hierarchical mesh segmentation based on planar symmetry. In: Proceedings of the Symposium on Geometry Processing (SGP '06). (June 2006) 111–119
14. Mitra, N.J., Guibas, L., Pauly, M.: Symmetrization. In: ACM Transactions on Graphics. Volume 26. (2007) #63, 1–8
15. Besl, P.J., McKay, N.D.: A method for registration of 3-D shapes. IEEE Trans. PAMI **14**(2) (1992) 239–256
16. Allen, B., Curless, B., Popović, Z.: The space of human body shapes: reconstruction and parameterization from range scans. In: SIGGRAPH '03: ACM SIGGRAPH 2003 Papers, New York, NY, USA, ACM Press (2003) 587–594
17. Sumner, R.W., Popović, J.: Deformation transfer for triangle meshes. In: SIGGRAPH '04: ACM SIGGRAPH 2004 Papers, New York, NY, USA, ACM Press (2004) 399–405
18. Pauly, M., Mitra, N.J., Giesen, J., Gross, M., Guibas, L.: Example-based 3D scan completion. In: Symposium on Geometry Processing. (2005) 23–32
19. Garland, M., Heckbert, P.S.: Surface simplification using quadric error metrics. In: Computer Graphics (Siggraph 1997). (1997) 209–216
20. Ahn, M., Lee, S., Seidel, H.: Connectivity transformation for mesh metamorphosis. In: Eurographics/ACM SIGGRAPH symposium on Geometry processing. (2004) 75–82
21. Kraevoy, V., Sheffer, A.: Cross-parameterization and compatible remeshing of 3D models. ACM Transactions on Graphics (Proc. SIGGRAPH 2004) **23**(3) (2004) 861–869
22. Alexa, M.: Merging polyhedral shapes with scattered features. The Visual Computer **16**(1) (2000) 26–37
23. Lee, A., Dobkin, D., Sweldens, W., Schroeder, P.: Multiresolution mesh morphing. ACM Transactions on Graphics (Proc. SIGGRAPH 2001) (1999) 343–350
24. Praun, E., Sweldens, W., Schroeder, P.: Consistent mesh parameterizations. ACM Transactions on Graphics (Proc. SIGGRAPH 2001) (August 2001) 179–184
25. Hanke, S., Ottmann, T., Schuierer, S.: The edge-flipping distance of triangulations. Journal of Universal Computer Science **2**(8) (1996) 570–579
26. Igarashi, T., Matsuoka, S., Tanaka, H.: Teddy: A sketching interface for 3D freeform design. In: Computer Graphics (Siggraph 1999), Addison Wesley Longman (1999) 409–416
27. P. Cignoni, C.R., Scopigno, R.: Metro: measuring error on simplified surfaces. Computer Graphics Forum **17**(2) (June 1998) 167–174
28. Yoshizawa, S., Belyaev, A., Seidel, H.: Smoothing by example: Mesh denoising by averaging with similarity-based weights. In: Shape Modeling International. (June 2006) 38–44