Shape Compression using Spherical Geometry Images

Hugues Hoppe[1] and Emil Praun[2]

[1] Microsoft Research, http://research.microsoft.com/~hoppe/
[2] University of Utah, http://www.cs.utah.edu/~emilp/

Abstract

We recently introduced an algorithm for spherical parametrization and remeshing, which allows resampling of a genus-zero surface onto a regular 2D grid, a spherical geometry image. These geometry images offer several advantages for shape compression. First, simple extension rules extend the square image domain to cover the infinite plane, thereby providing a globally smooth surface parametrization. The 2D grid structure permits use of ordinary image wavelets, including higher-order wavelets with polynomial precision. The coarsest wavelets span the entire surface and thus encode the lowest frequencies of the shape. Finally, the compression and decompression algorithms operate on ordinary 2D arrays, and are thus ideally suited for hardware acceleration. In this paper, we detail two wavelet-based approaches for shape compression using spherical geometry images, and provide comparisons with previous compression schemes.

1 Introduction

In previous work [20], we introduce a robust algorithm for spherical parametrization, which smoothly maps a genus-zero surface to a sphere domain. This sphere domain can in turn be unfolded onto a square, to allow remeshing of surface geometry onto a regular 2D grid — a geometry image. One important use for such a representation is shape compression, the concise encoding of surface geometry. In this paper, we explore the application of shape compression in more detail, describing two wavelet-based approaches.

As we will show, spherical geometry images are a powerful representation for the concise description of shape.

2 Related work on shape compression

The compression of geometric shape has recently been a very active area of research. Since we will not be able to cover every paper here, we refer the reader to recent comprehensive surveys [3, 10, 22]. The many compression techniques can be categorized into two broad approaches: *irregular mesh* compression and *remeshing* compression, depending on whether or not they preserve the original mesh connectivity.

Irregular mesh compression

Preserving the connectivity of the original mesh is important for accurately modeling sharp features such as creases and corners, particularly in manufactured parts. Also, meshes designed within graphical modeling systems may have face connectivities that encode material boundaries, shading discontinuities, or desired behavior under deformation.

The compression of irregular meshes involves two parts: connectivity and geometry. The mesh connectivity is a combinatorial graph; it can be encoded using approximately 2 bits per vertex [2]. The mesh geometry is given by continuous (x,y,z) vertex positions; these are typically quantized to 10, 12, or 14 bits per coordinate prior to entropy coding.

The compression of irregular meshes was pioneered by Deering [8], who describes a scheme for streaming decompression in the graphics system. Gumhold and Strasser [12] advance a front through a mesh using a state machine, and compress the necessary state changes. Touma and Gotsman [28] use a similar technique based on vertex-valence encoding. Many other papers have refined this approach, including the Edge Breaker scheme of Rossignac [21] and several methods for non-triangular meshes.

Several schemes support progressive representations, whereby coarser approximations can be displayed as the data stream is incrementally received. These include progressive meshes [14], progressive forest split compression [27], and the valence-driven simplification approach of Alliez and Desbrun [2].

Compressing the geometry of irregular meshes is difficult because the irregular sampling does not admit traditional multiresolution wavelet hierarchies. Most compression schemes predict the position of each vertex from its partially reconstructed neighborhood. A good example is the "parallelogram rule" of Touma and Gotsman [28]. The drawback of basing the prediction model on a local neighborhood is that it cannot capture the low-frequency features of the model. In other words, the local prediction rules cannot give rise to the broad basis functions that one would obtain in the coarsest levels of a traditional multiresolution hierarchy. One exception is the scheme of Karni and Gotsman [15], which constructs smooth basis functions using spectral analysis of the mesh adjacency matrix. However, this spectral analysis is costly and unstable, and therefore becomes practical only when performed piecewise on a partitioned model.

Remeshing compression

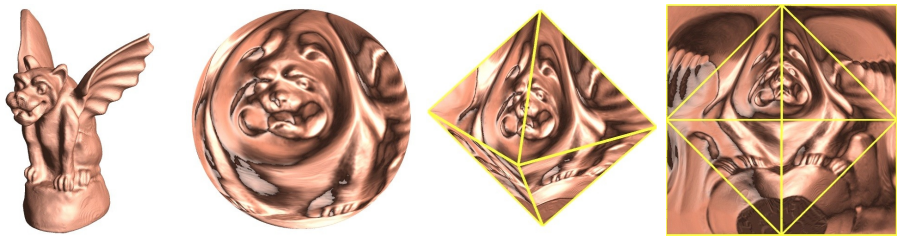
For many applications, preserving the connectivity of the given mesh is unnecessary. In particular, many models are obtained through 3D scanning technologies (e.g. laser range scanners, computed tomography, magnetic resonance imaging), and the precise connectivities in these dense meshes is somewhat arbitrary. Since shape compression is generally lossy, resampling the geometry onto a new mesh (with different connectivity) is quite reasonable.

In the remeshing approach of Attene et al. [5], an irregular-mesh compression algorithm resamples geometry as it traverses the mesh, by incrementally wrapping the mesh with isosceles triangles.

A number of methods use a *semi-regular* remeshing structure. Such a remesh is obtained by repeated quaternary subdivision of a coarse triangle mesh (i.e. each triangle face is regularly subdivided into 4 sub-faces). Lounsbery et al. [19] develop a wavelet-like framework over these semi-regular structures. Eck et al. [9] present a scheme for semi-regular remeshing of arbitrary triangle meshes, and achieve shape compression by removing small wavelet coefficients. Khodakovsky et al. [16] obtain better compression results using zero-trees; also, they express wavelet coefficients with respect to local surface coordinate frames, and assign fewer bits to the tangential components of the wavelet coefficients. The globally smooth parametrization of Khodakovsky et al. [18] reduces the entropy of these tangential components by constructing a remesh that is parametrically smooth across patch boundaries. The "normal mesh" representation of Khodakovsky et al. [17] attempts to remove tangential information altogether; each subdivision of the remesh is obtained by displacing most of the newly introduced vertices along the surface normal; only a small fraction of vertices require full 3D vector displacements.

Another approach, and the one pursued in this paper, is to form a *completely regular* remesh. As shown by Gu et al. [11], an arbitrary mesh can be resampled onto a regular 2D grid, a *geometry image*. The given mesh is cut along a network of cut paths to form a topological disk, and this disk is then parametrized over a square. The geometry image is obtained by creating a regular grid over the square and sampling the surface using the parametrization. Due to their simple regular structure, geometry images can be compressed using ordinary 2D image wavelets. However, one difficulty is that lossy decompression leads to "gaps" along the surface cut paths. Gu et al. [11] overcome these gaps by re-fusing the boundary using a topological sideband, and diffusing the resulting step function into the image interior.

In this work, we construct geometry images for genus-zero surface using a spherical remeshing approach, as described in the next section. By defining spherical extension rules beyond the geometry image boundaries, we avoid boundary reconstruction problems altogether.



Map of original mesh onto sphere, octahedron domain, and image

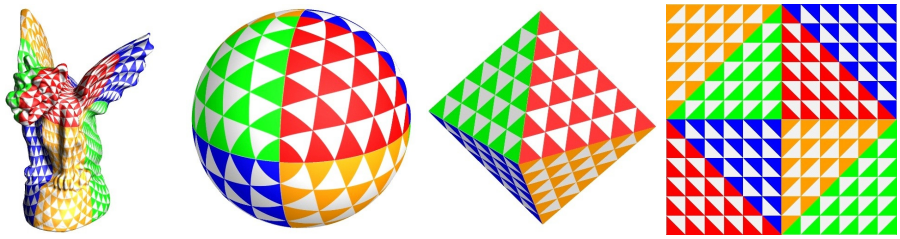
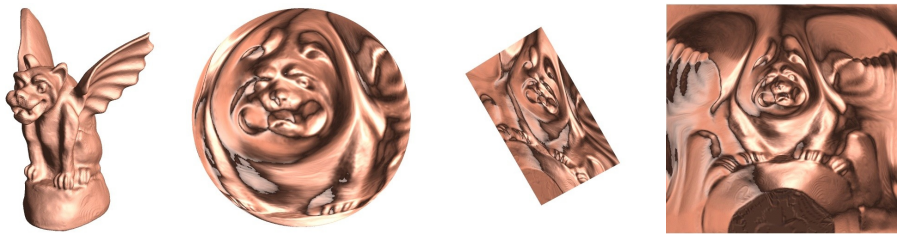


Illustration of the same map using image grid samples



Map of original mesh onto sphere, *flat* octahedron domain, and image

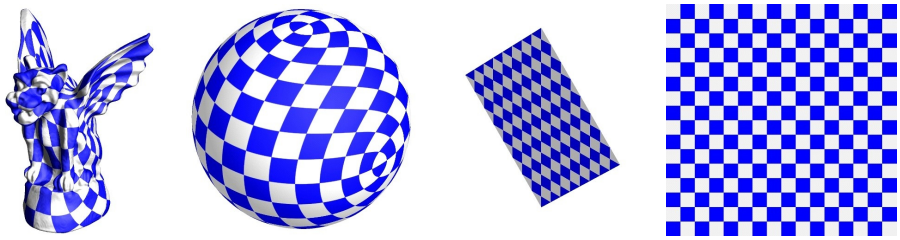


Illustration of the same map using image grid samples

Fig. 1. Illustration of remeshing onto octahedron and flat octahedron domains

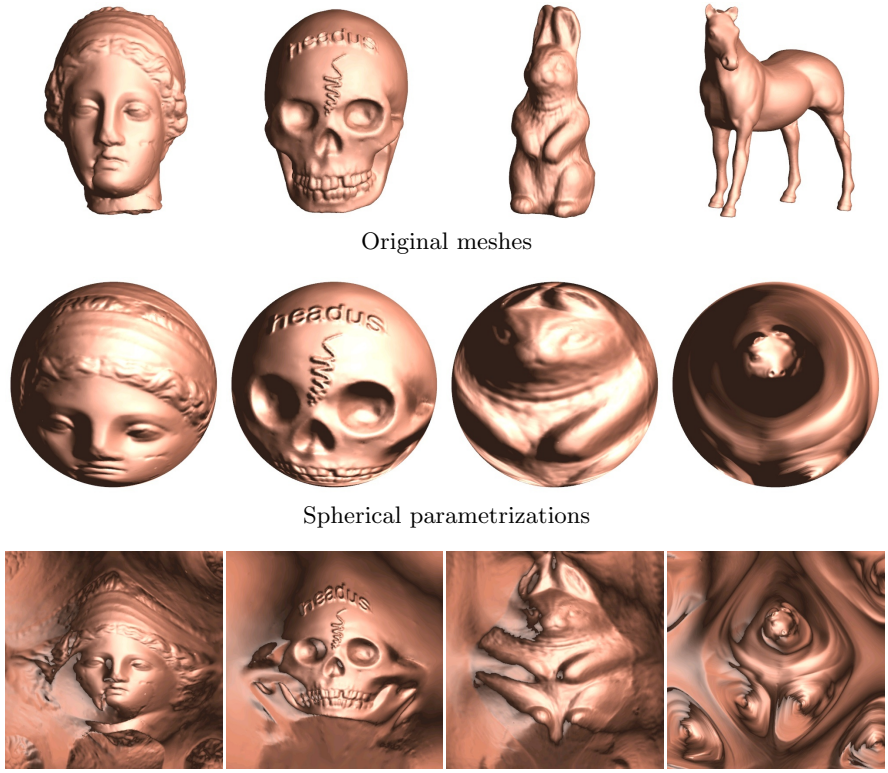


Fig. 2. Spherical parametrization and remeshing applied to the 4 test models. The geometry images are shown shaded to better illustrate the parametrization.

3 Review of spherical parametrization and remeshing

In previous work [20] we have presented a method for parametrizing a genus-zero model onto the sphere and remeshing it onto a geometry image, as illustrated in Figs. 1 and 2. Since the geometric signal is too smooth to be visualized by directly mapping (x,y,z) to the (R,G,B) channels of an image, we chose to visualize these geometry images in a different way. We compute for each pixel an approximated normal by taking neighbor differences, and shade the geometry image based on these normals (using two antipodal lights).

Spherical parametrization

The first step maps the original surface onto a sphere domain. For genus-zero models, the sphere is the most natural domain, since it does not require breaking the surface using any *a priori* cuts, which would otherwise artificially constrain the parametrization.

To be suitable for subsequent remeshing, the spherical map must satisfy two important properties: (1) it must be one-to-one, and (2) it must allocate enough domain area to all features of the mesh. Our spherical parametrization achieves these goals by employing a robust coarse-to-fine construction, and by minimizing a stretch distortion measure to prevent later undersampling.

Coarse-to-fine construction. The mesh is converted to a progressive mesh format [14] by repeatedly applying half-edge-collapse operations. For triangulated genus-zero models, one can always reach a tetrahedron as the base domain [25]. This base tetrahedron is mapped to a regular tetrahedron inscribed in the unit sphere. We then visit the progressive mesh sequence in coarse-to-fine order, adding vertices back into the mesh and mapping them onto the sphere. To guarantee that the map is one-to-one, a vertex must lie inside the kernel of the spherical polygon formed by its neighbors. Fortunately, if the map is one-to-one prior to inserting a new vertex, it can be shown that the new vertex's neighborhood is always non-empty, and thus new vertices can always be inserted into the parametrization while maintaining a bijection.

Stretch metric. To adequately sample all the features of a model, we employ a parametrization distortion metric based on stretch minimization. We minimize this nonlinear metric each time a new vertex is introduced, by locally optimizing its location and those of its immediate neighbors. Each time the number of vertices grows by factor of 1.5, we also perform a global pass, optimizing all vertices introduced so far. When optimizing any vertex, we constrain it to the kernel of its neighborhood, to prevent flips. Degenerate triangles are prevented naturally by the stretch metric, which becomes infinite in that case.

Spherical remeshing

Once we have a spherical parametrization for the mesh, we seek to resample the sphere onto a geometry image. For simplicity, this image should be square and should have simple boundary conditions. We have explored two schemes for unfolding the sphere onto the square, one based on a *regular octahedron* domain and the other based on a *flattened octahedron* domain (see Fig. 1). In either case, we regularly subdivide the octahedron, and map it to the sphere using the spherical parametrization procedure described earlier. (The one difference is that we measure stretch in the opposite direction, from the domain to the sphere.)

The samples of the octahedron are then associated with the grid locations of a square geometry image by cutting four edges of the octahedron meeting at a vertex, and unfolding the four faces incident to the vertex like flaps. In Fig. 1, rows 2 and 4 illustrate the sampling pattern imposed by the regular grid. For the octahedron, we use the linear 3-tap triangular reconstruction filter, and the filter footprint varies across the four quadrants of the geometry image, according to the faces of the base octahedron (shown in different colors for

easy identification). For the flat octahedron, we use the traditional 4-tap bilinear reconstruction filter, and this filter is uniform across the whole geometry image.

The geometry of the regular octahedron corresponds nicely with the use of spherical wavelets (Section 4.1), since it offers derivative continuity across edges under the equilateral triangle sampling pattern. Similarly, the geometry of the flattened octahedron corresponds with the use of image wavelets (Section 4.2), since the flattened octahedron unfolds isometrically (i.e. without distortion) onto the square image.

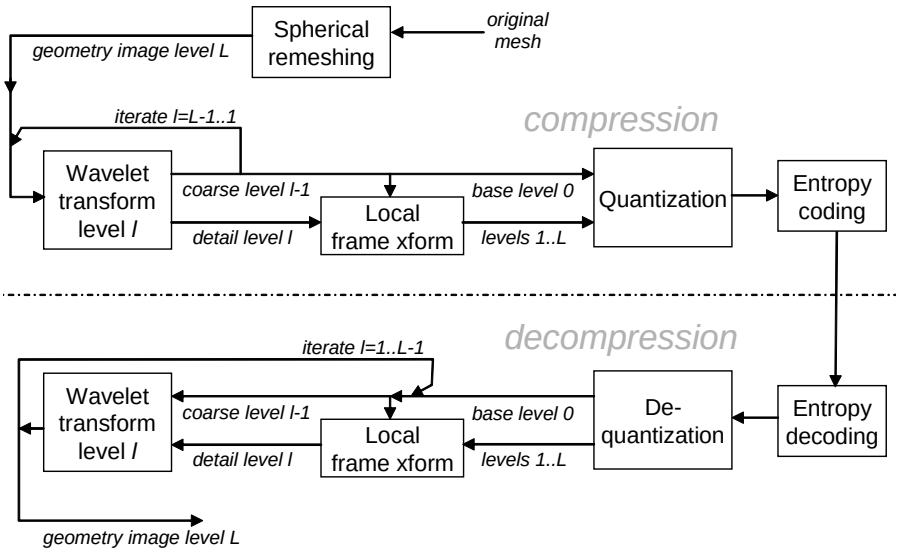


Fig. 3. Overview of compression process

4 Compression using spherical geometry images

To compress a model, we first apply a wavelet transform to the geometric signal, using either spherical wavelets or image wavelets. The bands resulting from the wavelet transform are then run through a general-purpose quantizer and entropy coder [7]. The process is summarized in Fig. 3, and in the pseudocode of Fig. 4.

Previous research [13] has shown that the geometric information associated with displacement of samples from their predicted location along the surface *normal* is more important than the "parametric" information associated with the *tangential* components of the displacement. Accordingly, we express the fine-scale detail (the high-pass bands or wavelet coefficients from each step of the wavelet transform) in local coordinate frames predicted using the low-pass band. During the quantization and entropy coding, we assign greater

Compression:

```
Read finest level L
For all levels from fine to coarse: l = L .. 1:
  - identify "even" grid locations (i.e. those in coarser level l-1)
  - apply low-pass analysis filter centered on "even" samples
  - apply high-pass analysis filter to other ("odd") samples
     (if low-pass or high-pass filter kernels reach outside
      the geometry image, use boundary extension rules)
  - gather all "even" samples into coarse level l-1
  - gather all "odd" samples into detail plane(s) for level l
  - using level l-1, compute local tangential frames for samples on
     detail plane(s) of level l
  - express detail in local tangential frames
Run coarsest level 0 and all detail planes for levels 1..L
  through quantizer and entropy encoder to achieve target bit budget
```

Decompression:

```
Run entropy decoder and dequantizer to produce coarsest level 0
  and detail planes for levels 1..L
For all levels from coarse to fine l = 0 .. L-1:
 - using level l, compute local tangential frames for samples in
    detail plane for level l+1
 - transform detail from local frames to absolute coordinates
 - apply synthesis filter to level l to predict level l+1
 - apply synthesis filter to detail plane(s) for level l+1
    and combine with prediction to produce level l+1
    (using boundary extension rules as necessary)
Output finest level L
```

Fig. 4. Pseudo-code for the compression and decompression algorithms

perceptual importance to transformed components normal to the surface than to the tangential components (we found a factor of 3 to give the best results for our models).

Note that the compression and decompression are lossy, and that the quantization errors combine non-linearly. For example, if a coarse level is reconstructed inexactly, the errors are not simply added to the final result; they additionally cause small rotations of the local coordinate frames transforming the detail for the finer levels.

For the wavelet compression/decompression stage we present two alternative schemes. The first is based on spherical wavelets introduced by Schröder and Sweldens [24]; the base (coarsest) sampling level is an octahedron, and progressively finer levels are obtained by applying standard subdivision rules such as Loop or Butterfly. The second is based on image wavelets. Both schemes are interesting to consider since they offer different advantages. The mesh-based spherical wavelet scheme seems more natural for coding geome-

try, and provides good reconstruction of sharp detail at very low bit budgets. The image based method is easier to implement (just by modifying one of the many existing image coders), and benefits from a large body of research into image wavelets that resulted in well optimized wavelet bases with large support [4]. After presenting the implementation of both methods in this section, we contrast their results in Section 5.

4.1 Spherical Wavelets

Rather than having a complicated pointer-based mesh data structure, we can apply all the mesh processing operations directly on the geometry image of the unfolded octahedron by manipulating grid location indices. The vertices of the base octahedron correspond to the samples at the corners, boundary midpoints, and center of the geometry image. The vertices of the octahedron subdivided k times will be at locations $(i*2^{L-k}, j*2^{L-k})$, for $0 \leq i, j \leq 2^{k+1}$, where L is the finest level. Two samples on a given level are neighbors in the subdivided octahedron if they are 4-adjacent in the grid restricted to the samples on that level, or linked by a diagonal. The orientation of this diagonal is determined by the image quadrant, i.e. "forward slash" for the upper left and lower right one, and "backward slash" for the other two. This distinction in the use of diagonals is needed because the sample connectivity in the grid is not arbitrary but is inherited from the subdivided octahedron. We check the quadrant at the midpoint between the two samples, to avoid ambiguities created when one of the samples is a vertex of the base octahedron. Using this simple set of rules, we efficiently gather the neighbors of a vertex on a given level, to form the stencils required for wavelet analysis and synthesis (e.g. the green stencil in Fig. 5).

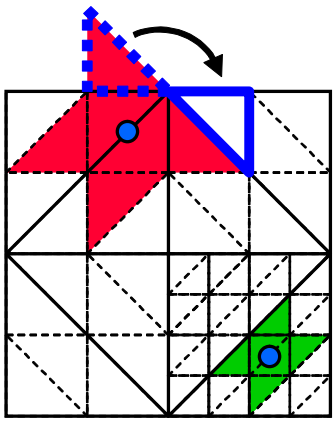


Fig. 5. Spherical wavelets on the unfolded octahedron geometry image. The green and red regions highlight two Butterfly stencils at different subdivision levels. Since the red stencil reaches outside the image, it uses boundary extension rules.

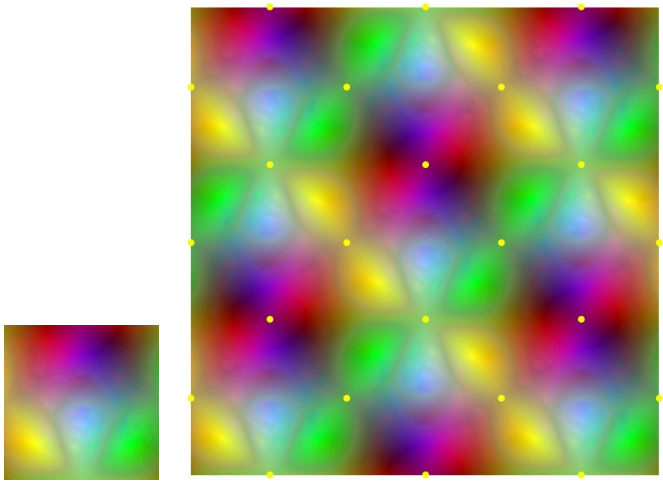


Fig. 6. A spherical geometry image and its infinite tiling in the plane. The parametrization is globally smooth except at the image boundary midpoints.

Boundary extension rules. To complete the wavelet transform stencils corresponding to samples near the boundaries we sometimes need to "hallucinate" values outside the square grid of values. Standard tricks used in signal processing to extend an image beyond its borders include replicating the boundary samples or reflecting the image across its boundaries. These methods provide a continuous signal, but introduce derivative discontinuities in the infinite lattice produced, and discontinuities are more expensive to code than smooth signals. Instead of using the standard tricks, we designed different boundary extension rules for our unfolded octahedron geometry image, which do produce an infinite lattice with derivative continuity. The general idea for these rules is that whenever we must "march" outside the image across a boundary to produce a sample, we flip the image (such that the boundary is mapped onto itself by a 180-degree rotation) and return the value located there (e.g. the red stencil in Fig. 5). Considering the image samples to be labeled in row-major order, starting from 0 (so samples in the left column are (0,j)), the rules are $I(-i,j) = I(i,n-1-j)$, $I(i,-j) = I(n-1-i,j)$, $I(n-1+i,j) = I(n-1-i,n-1-j)$ and $I(i,n-1+j) = I(n-1-i,n-1-j)$, where $n = 2^k + 1$ is the number of rows and columns. This is equivalent to filling the infinite plane of all sample locations by rotating the original image around the boundary midpoints (see Fig. 6). The infinite lattice produced is continuous everywhere and derivative continuous everywhere except at the repeated instances of the four boundary midpoints (dots in the figure).

Note that for $i,j = 0$ and $i,j = n-1$, the rules provide constraints on the boundaries rather than a way to extend the image outside its borders. Therefore, to avoid sample duplication we also consider the right half of the

top and bottom boundaries of the geometry image and the lower half of the left and right boundaries to be "outside" the image (so we map those locations to the surviving half of the same boundary, using the extension rules). These "outside" boundary grid locations are not processed by the quantizer and entropy coder.

Local tangential frame. We use the lifted Butterfly scheme, as described in [24]. We compute a normal for each "odd" sample by averaging the normals of the faces from the Butterfly stencil (with weights 1,4,1,1,4,1). Note that the vertices of these faces are all "even" vertices. The Y coordinate of the frame is obtained as the cross product between this normal and the row direction in the grid of samples (obtained from differences of neighboring samples, similarly to the image wavelet case). We take a cross product again to obtain the X axis of the frame.

4.2 Image Wavelets

The other scheme we consider is based on image wavelets, using the flat octahedron parametrization. We use a general-purpose wavelet-based image compression package [7], modified to make use of boundary extension rules and local tangential frame coding.

Boundary extension rules. These rules come into play during both compression and decompression phases when we are applying a signal processing kernel to a sample that is closer to the image boundary than the kernel width (see Fig. 7). In these cases, we have to "hallucinate" samples for the grid locations outside the original image. Similarly to the spherical wavelets case, we fill these grid locations using boundary extension rules.

For image wavelets there are two types of rules, depending on whether the image boundary is located "on" the samples or "between" samples. To simplify the discussion, let's assume we need a sample located outside the image across the left boundary. The first case corresponds to the original unfolded octahedron, and all the coarser levels. These levels have 2^k+1 columns, and the left column is constrained: sample $I(0, j)$ must be equal to sample $I(0, n_r - 1 - j)$, where n_r is the number of rows. In this case, the reflection rule is the same as in the spherical wavelets case $I(-i, j) := I(i, n_r - 1 - j)$.

The second type of extension rules apply to the detail planes obtained from the image wavelet transform. Some of these images have a number of columns equal to 2^k rather than $2^k + 1$ and lack the left boundary with constrained samples. Specifically, the image wavelet transform uses separable filters, so to apply a 2D transform, it first applies a 1D transform to all the rows, producing images L and H (see Fig. 8) and then a 1D transform to all the columns, producing images LL, LH, HL, and HH. Planes with an even number of columns (such as H, HL, and HH) lack the leftmost boundary with constrained samples, and use a reflection boundary located "between" grid locations: $I(-i, j) := I(i - 1, n_r - 1 - j)$. Intuitively, in the geometry image

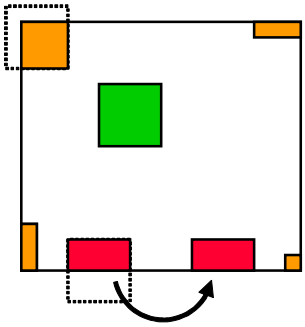


Fig. 7. Effect of domain extension rules on wavelet basis extents using the flat octahedron image wavelets.

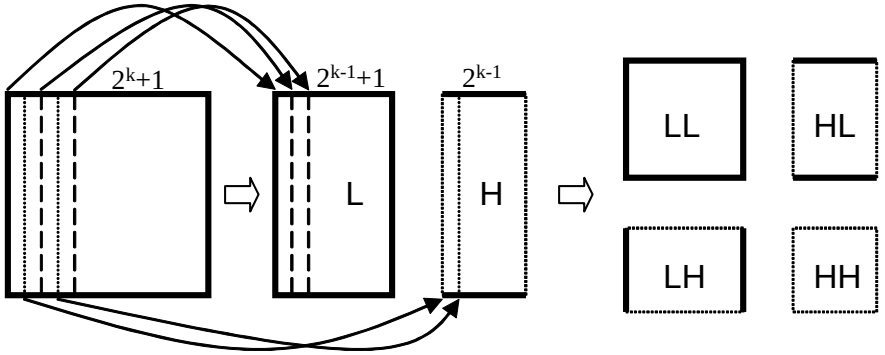


Fig. 8. Wavelet transform of an image level. Applying a 1D transform to each row results in a low-pass plane L and a high-pass (detail) one, H. After a 1D transform on each of the columns, we get the coarser level LL, and three detail planes. The thick boundary edges have flip-symmetry constraints.

case, when we go outside the image across a "fat" edge in Fig. 8, we come back inside the image across the same boundary, skipping the sample on the boundary itself, while in the case of a dotted boundary edge of a detail plane in Fig. 8 we don't skip the sample on the boundary.

Similarly to the spherical wavelets case, the first type of boundary rules applied to the samples on the boundary provide constraints, rather than ways to extend the image. During the compression phase, those constraints are satisfied since the original geometry image was constructed using them. However, since the whole compression/decompression process is lossy, the constraints may be unsatisfied during decompression. To ensure that the resulting model is crack-free, we enforce the constraints at all the resolution levels during decompression. After we recover a level, before using it to produce the local frame and the coarse approximation for the next level, we first average together the samples on the boundary that should be equal (for example, samples $(i, 0)$ and

$(n-1-i, 0)$ on the top row of the image are averaged together and set to the resulting value). Pairs of corresponding locations on the image boundaries are averaged together, and the four corners are also averaged and kept consistent. It is advantageous to enforce consistency at all the resolution levels, rather than just once at the end, in order to avoid "step functions" from appearing in the result.

The boundary rules benefit the compression/decompression algorithm in two ways. First, they maintain consistency between samples on the boundaries, preventing cracks in the 3D model corresponding to the geometry image. Second, they help the prediction for the samples near the boundaries, since the image signal appears smooth not only in the interior of the image but also near its borders (see Fig. 6).

An important thing to note is the fact that applying a symmetric filter kernel to a lattice satisfying the boundary extension rules will result in a new lattice with the same extension rules. We make use of this fact since we rely on the extension rules at all the resolution levels, not just the finest resolution geometry image. For arbitrary kernels, one would need to store two instances of the processed image, one with the kernel itself and one with the reflected kernel, in order to be able to represent the whole new infinite lattice. This would be a significant drawback for compression, since the bit budget would double. We therefore use wavelets based on symmetric kernels.

Local tangential frame. To compute a local frame for each "odd" sample, we first obtain vectors corresponding to the row and column directions of the grid. If the sample is on an even row (and necessarily an odd column), we get the row direction from the difference of the two neighboring (even) samples in the same row. If the sample is from an odd row, we average the directions computed using the two adjacent even rows. We compute the column direction in a similar fashion. Taking the cross product of the two vectors we obtain the normal direction, which we cross with the column direction to get the X axis. The Y axis is obtained by cross product between the normal and X. Finally we normalize the three vectors composing the frame.

Implementation details. We used the Antonini [4] image wavelet bases, which are symmetric separable kernels with 7 entries for the 1D high-pass and 9 entries for the low-pass. Since the wavelet kernels have large support, we do not apply the wavelet transform all the way down to the 3x3 image, but instead use a fixed number of stages (specifically, 5), starting from a fine 513x513 geometry image.

5 Results and discussion

We have run compression experiments using the 4 test models shown in Fig. 2. The spherical parametrization process took 1–3 minutes on the original meshes with 28–134K faces. (This significant improvement in processing times over

those reported in [20] are simply due to code optimization.) The given models were remeshed into geometry images of size 513x513 and then compressed.

The results for both spherical wavelets and image wavelets are shown in Figs. 9–12. The rightmost images show each shape compressed to approximately 12,000 bytes. At this compression rate, the geometric fidelity is excellent, and these images should be considered as references for the more aggressive compressions to their left. At approximately 3,000 bytes (middle images), compression effects become evident in the blurring of sharp features. At 1,500 bytes (left images), effects are even more pronounced. It is interesting however that 1,500 bytes is generally sufficient to make the object recognizable.

The graphs in Figs. 9–12 show Peak Signal-to-Noise Ratio (PSNR) graphs for each model. We compare our rate-distortion curve with those of progressive geometry compression (PGC) [16], globally smooth parametrization (GSP) [18], and normal mesh compression (NMC) [17]. Also included for comparison is the irregular mesh compression scheme of Touma and Gotsman [28] (which preserves mesh connectivity). As can be seen from the graphs, our rate-distortion curves are generally better than PGC, but just below GSP and NMC.

The spherical wavelets generally offer better visual reconstruction, as is most evident on the skull model (Fig. 10). The reason is that the spherical wavelet kernels have more localized support than the particular image wavelets that we used, and therefore adapt more quickly to the fine detail. However, the PSNR graphs indicate that the error as measured using L^2 Hausdorff distance is generally lower when using the image wavelets. Thus, it can be argued that L^2 error is not an accurate visual norm [28], and that one should attempt to recover high-frequency detail first [26]. A more comprehensive comparison using other image wavelets (with more local support) would be useful.

The horse model (Fig. 12) shows a limitation of our spherical parametrization approach. For shapes containing many extremities, the parametrization onto the sphere suffers from distortion, and these distortions give rise to rippling effects under lossy reconstruction. In such cases, our compression is much less effective than semi-regular remeshing.

6 Summary and future work

We have described a spherical parametrization approach to remeshing genus-zero surfaces for shape compression. The surface is remeshed into a regular 2D grid of samples, which is then compressed using wavelets. The compression and decompression algorithm have great potential for hardware acceleration, since they do not involve any pointer-based data structures. We have presented a wavelet scheme based on ordinary 2D image wavelets, and applied it to the spherical domain using boundary extension rules, effectively creating spherical topology over a square domain. Like prior semi-regular remeshing

schemes, our geometry image remeshing approach naturally supports progressive compression.

Experiments show that spherical geometry images are an effective representation for compressing shapes that parametrize well onto the sphere. Although the scheme is robust on arbitrary models, shapes with long extremities suffer from rippling artifacts during lossy decompression. One area of future work is to attempt to reduce these rippling effects by modifying the parametrization process.

Also, our approach should be extended to support surfaces with boundaries. One possibility would be to encode a separate bit-plane indicating which subset of samples lie in the "holes" of the remeshed model.

The accuracy of remesh representations can be improved by refitting to the original model as an optimization (e.g. [23]). Such optimization would likely help rate-distortion behavior, particularly using an appropriate visual error norm. However, local geometry optimization does increase the entropy of the "tangential" signal within the surface remesh, so it would be important to introduce a smoothing term to minimize such entropy away from significant geometric features.

In this work, we have used the sphere as an intermediate domain for parametrizing a surface onto an octahedron or flattened octahedron. One could also consider parametrizing the surface directly onto the octahedron, thus bypassing the sphere. This task would be more challenging, since the octahedron is not everywhere smooth like the sphere. However, it may allow the construction of improved (more stretch-efficient) parametrizations.

Briceño et al. [6] explore the compression of animated meshes using the geometry images of [11]. It would be interesting to apply a similar framework using spherical geometry images.

Acknowledgments

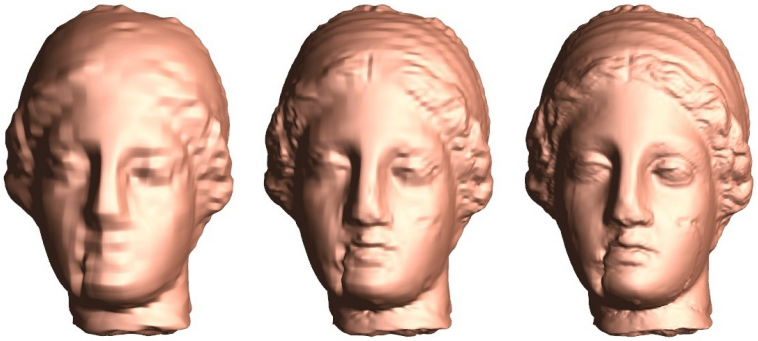
We gratefully thank Andrei Khodakovsky for providing results for our comparisons. We thank Cyberware for the Venus, rabbit and horse models, and Headus for the skull model.

References

1. Alliez, P., and Desbrun, M.: Progressive encoding for lossless transmission of 3D meshes. SIGGRAPH 2001.
2. Alliez, P., and Desbrun, M.: Valence-driven connectivity encoding for 3D meshes. EUROGRAPHICS 2001.
3. Alliez, P., and Gotsman, C.: Recent advances in compression of 3D meshes. Symposium on Multiresolution in Geometric Modeling. Cambridge, September 2003.

4. Antonini, M., Barlaud, M., Mathieu, P., and Daubechies, I.: Image coding using wavelet transform. IEEE Transactions on Image Processing, 205–220, 1992.
5. Attene, M., Falcidieno, B., Spagnuolo, M., and Rossignac, J.: SwingWrapper: Retiling triangle meshes for better EdgeBreaker compression. ACM Transactions on Graphics, to appear.
6. Briceño, H., Sander, P., McMillan, L., Gortler, S., and Hoppe, H.: Geometry videos. Symposium on Computer Animation 2003.
7. Davis, G.: Wavelet image compression construction kit. http://www.geoffdavis.-net/dartmouth/wavelet/wavelet.html (1996).
8. Deering, M.: Geometry compression. SIGGRAPH 1995, 13–20.
9. Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M., and Stuetzle, W.: Multiresolution analysis of arbitrary meshes. SIGGRAPH 1995, 173–182.
10. Gotsman, C., Gumhold, S., and Kobbelt, L.: Simplification and compression of 3D-meshes. Tutorials on multiresolution in geometric modeling, A. Iske, E. Quak, M. Floater (eds.), Springer (2002).
11. Gu, X., Gortler, S., and Hoppe, H.: Geometry images. SIGGRAPH 2002, 355–361.
12. Gumhold, S., and Strasser, W.: Real time compression of triangle mesh connectivity. SIGGRAPH 1998, 133–140.
13. Guskov, I., Vidimce, K., Sweldens, W., and Schröder, P.: Normal meshes. SIGGRAPH 2000, 95–102.
14. Hoppe, H.: Progressive meshes. SIGGRAPH 1996, 99–108.
15. Karni, Z., and Gotsman, C.: Spectral compression of mesh geometry. SIGGRAPH 2000, 279–286.
16. Khodakovsky, A., Schröder, P., and Sweldens, W.: Progressive geometry compression. SIGGRAPH 2000.
17. Khodakovsky, A., and Guskov, I.: Normal mesh compression. Geometric Modeling for Scientific Visualization, Springer-Verlag, Heidelberg, Germany (2002).
18. Khodakovsky, A., Litke, N., and Schröder, P.: Globally smooth parameterizations with low distortion. SIGGRAPH 2003.
19. Lounsbery, M., DeRose, T., and Warren, J.: Multiresolution analysis for surfaces of arbitrary topological type. ACM Transactions on Graphics, 16(1), 34–73 (1997).
20. Praun, E., and Hoppe, H.: Spherical parametrization and remeshing. SIGGRAPH 2003, 340–349.
21. Rossignac, J.: EdgeBreaker: Connectivity compression for triangle meshes. IEEE Trans. on Visualization and Computer Graphics, 5(1), 47–61 (1999).
22. Rossignac, J.: 3D mesh compression. Chapter in The Visualization Handbook, C. Johnson and C. Hanson, eds., Academic Press, to appear (2003).
23. Sander, P., Wood, Z., Gortler, S., Snyder J., and Hoppe, H.: Multi-chart geometry images. Symposium on Geometry Processing 2003, 157–166.
24. Schröder, P., and Sweldens, W.: Spherical wavelets: Efficiently representing functions on the sphere. SIGGRAPH 1995, 161–172.
25. Shapiro, A., and Tal, A.: Polygon realization for shape transformation. The Visual Computer, 14 (8-9), 429–444 (1998).
26. Sorkine, O., Cohen-Or, D., and Toledo, S.: High-pass quantization for mesh encoding. Symposium on Geometry Processing, 2003.
27. Taubin, G., Gueziec, A., Horn, W., and Lazarus, F.: Progressive forest split compression. SIGGRAPH 1998.
28. Touma, C., and Gotsman, C.: Triangle mesh compression. Graphics Interface 1998.

1,445 bytes (61.6 dB) 2,949 bytes (67.0 dB) 11,958 bytes (75.7 dB)
Compression using spherical wavelets



1,357 bytes (60.8 dB) 2,879 bytes (66.5 dB) 11,908 bytes (77.6 dB)
Compression using image wavelets

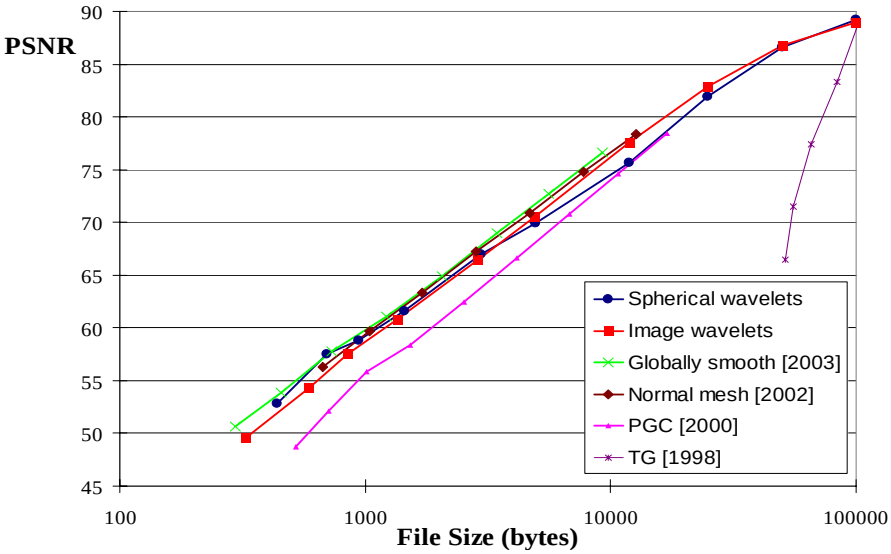


Fig. 9. Compression results on Venus model

1,444 bytes (59.6 dB) 2,951 bytes (65.5 dB) 11,959 bytes (76.1 dB)
Compression using spherical wavelets

1,367 bytes (60.5 dB) 2,889 bytes (66.2 dB) 11,915 bytes (77.5 dB)
Compression using image wavelets

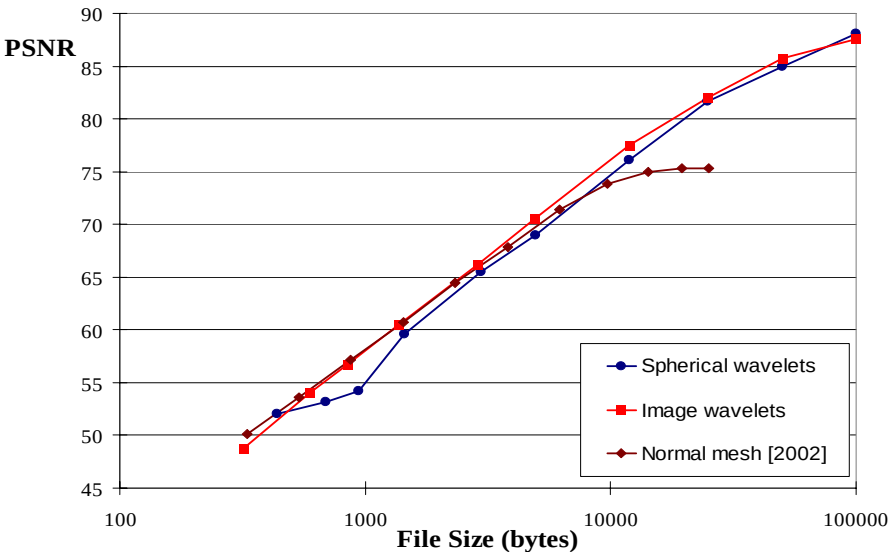


Fig. 10. Compression results on skull model

1,447 bytes (64.0 dB) 2,950 bytes (69.5 dB) 11,958 bytes (79.8 dB)
Compression using spherical wavelets

1,364 bytes (63.2 dB) 2,881 bytes (70.2 dB) 11,906 bytes (81.9 dB)
Compression using image wavelets

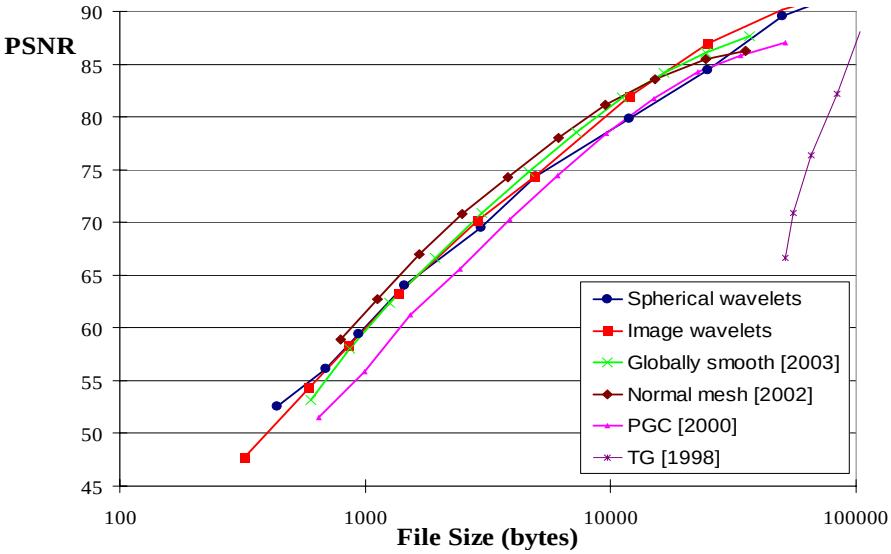
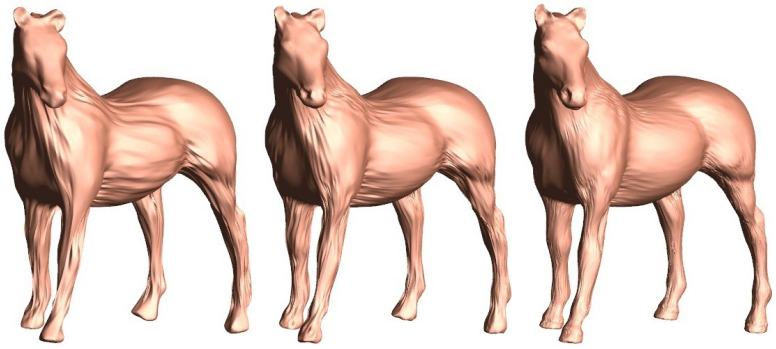
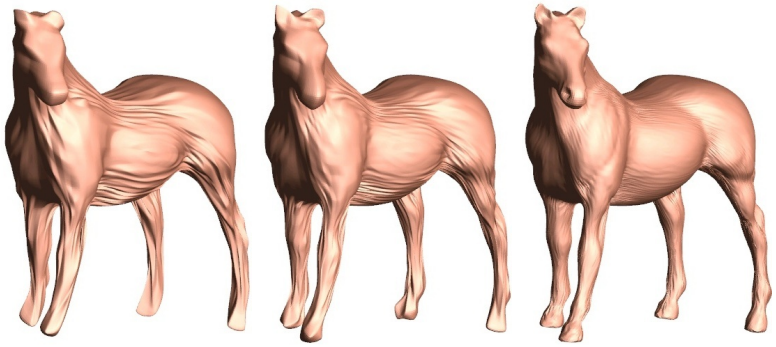


Fig. 11. Compression results on rabbit model



1,456 bytes (55.7 dB) 2,961 bytes (57.6 dB) 11,968 bytes (69.7 dB)
Compression using spherical wavelets



1,376 bytes (54.2 dB) 2,900 bytes (60.6 dB) 11,932 bytes (73.1 dB)
Compression using image wavelets

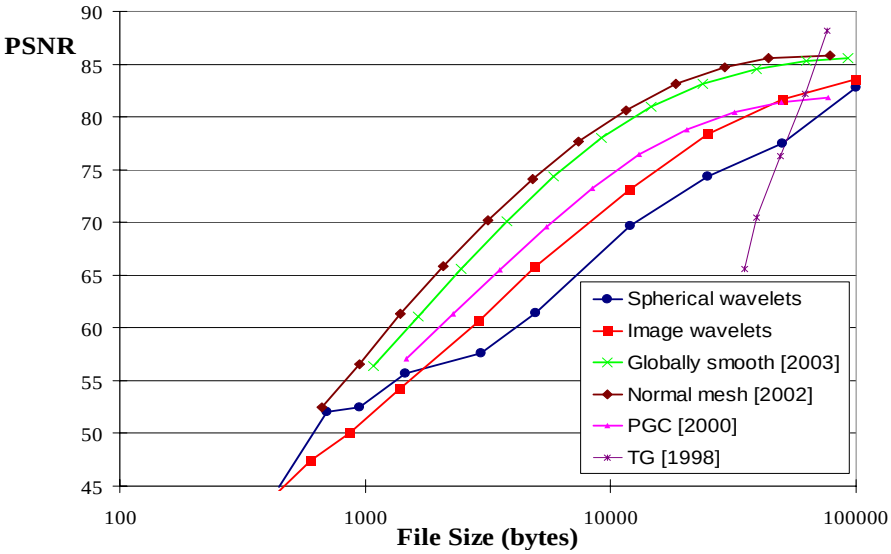


Fig. 12. Compression results on horse model