Progressive Encoding and Compression of Surfaces Generated from Point Cloud Data

J. Smith, G. Petrova, S. Schaefer

Texas A&M University, USA

Abstract

We present a new algorithm for compressing surfaces created from oriented points, sampled using a laser range scanner or created from polygonal surfaces. We first use the input data to build an octree whose nodes contain planes that are constructed as the least square fit of the data within that node. Then, given an error threshold, we prune this octree to remove redundant data while avoiding topological changes created by merging disjoint linear pieces. From this octree representation, we provide a progressive encoding technique that encodes the octree structure as well as the plane equations. We encode the planes using distances to three points and a single bit. To decode these planes, we solve a constrained optimization problem that has closed-form solution. We then reconstruct the surface from this representation by implicitizing the discontinuous linear pieces at the leaves of the octree and take a level set of this implicit representation. Our tests show that the proposed method compresses surfaces with higher accuracy and smaller file sizes than other methods.

1. Introduction

Many recent advances in 3D scanner technology bring new opportunities and possibilities for creating digital models of real world objects. These digital models have many applications ranging from visualization and preservation of artistic works, to navigation and reconstruction of large environments. To acquire these models, one typically uses a laser range scanner that generates 3D point samples on the surface of the object. These point samples need to be processed further since almost all applications require polygonal models with explicit connectivity. Most methods for creating polygonal models are implicit, that is, they build from a set of (possibly oriented) point samples an implicit function whose level set is the reconstructed surface. Applying a polygonalization method such as Marching Cubes [1] to that function creates the desired polygonal model.

One of the main difficulties that has hampered the performance of surface reconstruction methods in recent years is the sheer quantity of data. For example, researchers in the Digital Michelangelo project [2] scanned statues for the purpose of digital preservation of these works of art to millimeter or sub-millimeter accuracy, which resulted in data sets for each statue in the range of hundreds of millions to billions of point samples. Light Detection And Ranging (LIDAR) data collected from an aircraft could be of the order of billions to tens of billions of samples. While several out-of-core reconstruction methods have been built [3, 4] to handle this massive amount of data, simply storing and/or transmitting the data is a challenge. Therefore, new innovative approaches for addressing storage and transmission issues via data compression are needed.

There are many opportunities for compression in the surface reconstruction process. One solution is to simply compress the input points. Another is to generate and compress an intermediate data structure that represents these points. One could also compress either the implicit function generated from the point cloud or the polygonal model that is the result of the surface reconstruction process (see Figure 1).

Contributions

We propose an algorithm that performs compression on an intermediate data structure that we build from the oriented input points (see the second image from Figure 1). This data structure is an adaptive octree with planes at every node containing input data points. From this representation, using [5], we can easily compute an implicit function and extract its level set to produce the final, reconstructed surface. More precisely, we

- provide a method for creating an adaptive octree that removes redundant geometric information and

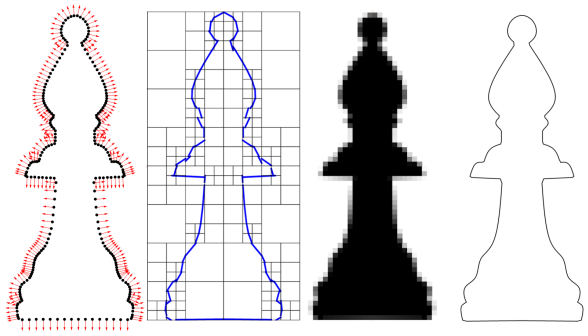

Figure 1: Reconstruction pipeline for the Bishop model from left to right: oriented point cloud generated from surface samples, tree data structure whose nodes contain planes, implicit function representing the reconstructed surface, and the final polygonal model.

at the same time avoids extrapolation and merging of different regions of the surface,

- compress the octree structure together with the planes at each of its nodes. To encode a plane at a node, we use the distances from three points associated with that node to the plane and a single bit.

- decode the planes by providing a closed-form solution for a constrained optimization problem.

2. Related Work

Data compression is an extensively studied area. To narrow our discussion of previous works, we will only examine progressive compression algorithms for surfaces. Unlike single rate encoders, progressive encoders embed a low resolution representation of the surface inside of a higher resolution representation. While generally having less potential for compression, progressive encoders are ideal for streaming data in low bandwidth environments or when transmissions may be truncated prematurely.

Based on the reconstruction pipeline in Figure 1, the first opportunity for compression is to compress the input points. Progressive point cloud compression methods rely on inserting points into a deep octree and encoding the connectivity of the resulting structure [6, 7, 8]. More precisely, a point is simply represented by the center of its leaf node (at high depths, this encoding is almost lossless). Note that if the point set is oriented, normals will also have to be compressed [9].

Many surface reconstruction algorithms [10, 11, 4] use the input points to create an implicit function represented as values over a uniform or adaptive grid, as shown in the third image in Figure 1. Another compression strategy is to compress this function instead of the input points. For example, Laney et al. [12] use wavelet compressed signed distance volumes and encode the wavelet coefficients using a zero tree encoder, while Chandrasekaran et al. [13] use a multiscale surflet representation to compress the implicit function.

The final opportunity for compression is to compress the polygonal model that represents the reconstructed surface (see Figure 1, right). While it is possible to treat the surface as an unstructured mesh and apply compression algorithms developed for such meshes [14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25], Lee et al. [26] showed that significant gain is possible if we specialize the compression to take advantage of how the surface was generated, namely as a level set extracted from a uniform/adaptive grid. The authors achieve significant compression results by using a uniform octree and an entropy encoder to encode inside/outside information as well as detail vectors at the leaves of that tree. Lewiner et al. [27] encode surfaces extracted from hierarchical tetrahedral grids. Saupe and Kuska [28] compress the extracted iso-surface by exploiting a grid based extraction method to predict intersection values.

We achieve our compression results by creating, encoding and decoding an intermediate data structure, which is an adaptive octree with a plane inside each node (see the second image of Figure 1) that represents the original point cloud, and is used to produce the underlying surface. A similar approach has also been explored by Park and Lee [29], where the authors build an octree refined to a uniform depth wherever points intersect the tree. Each node (interior or leaf) stores a plane constructed using principal component analysis (PCA) on the points within that node. They encode these planes by storing the signed distance along the normal direction of the parent's plane from three points (in the parent plane) to the child's plane and perform a zero-tree encoding [30] of these coefficients. The authors essentially view every child's plane as a function over the domain of the parent's plane. However, this approach has two serious flaws. The first is that the signed distance is infinite when the child's plane is perpendicular to the parent's plane. In this case the encoding fails. If the distance to all three points is infinite, the child's plane will contain the normal of the parent and can be rotated in any way about this normal. Therefore, the child's plane will be impossible to reconstruct without additional information. Given the progressive nature of the encoding, all further data in the file will be corrupted. The second disadvantage of this approach is

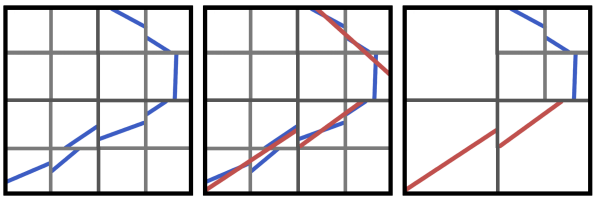

Figure 2: 2D explanation of pruning. From left to right: the plane in the child nodes (blue), the parent's plane (red), the pruned tree. The bottom two nodes have parents whose planes can represent their children within the collapse tolerance while the top right node cannot.

the fact that the distances to these three points cannot be bounded. In practice, the distances will be small if the parent's plane is a good prediction of the child's plane, but in general the distance can be unbounded (even infinite as just stated), which makes quantization impossible and leads to poor compression.

We also encode planes using signed distances to three points; however, we use minimum Euclidean distance instead of distance along the parent normal. The advantage of doing so is that we can bound these distances with respect to the depth of each node, which improves compression performance and eliminates the failure case from [29]. The disadvantage is that we cannot reconstruct the child plane by solving a linear system of equations. Instead, during reconstruction, we solve a constrained optimization problem that has a computationally efficient closed-form solution.

3. Octree Generation

Our input is a set of oriented points p_i, $i = 1, \ldots, N$, represented as column vectors with associated outward facing unit normals n_i. Without loss of generality, we assume that $p_i \in [0, 1)^3$. We form an octree, associated to this point cloud, by refining octree nodes that contain a point sample down to some maximum depth d, specified by the user. This maximum depth is typically chosen based on the number and sampling frequency of the input points. For each octree node, we build a plane restricted to the node that best fits the points contained within that node in the sense that it minimizes the error function

$$\sum_{k=1}^{m} \left| n \cdot (p_{i_k} - o) \right|^2, \tag{1}$$

where n is the unit normal of the plane, o is a fixed point on the plane, and m is the number of points contained within this node. We minimize this error function with respect to the variables n, o, with the constraint $n \cdot n = 1$. The solution is given by a simple PCA, where

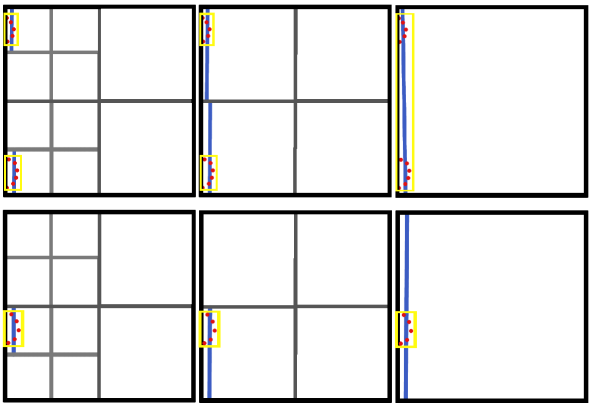

Figure 3: 2D depiction of common problems that arise in quadtree collapse. The bounding box formed by the points in the node is shown in yellow. The top row shows the merging of two disjoint pieces of surface. The bottom row shows extrapolation of a piece of the surface.

$$o = \frac{1}{m} \sum_{k=1}^{m} p_{i_k}, \tag{2}$$

and n is the unit eigenvector corresponding to the smallest eigenvalue of the matrix

$$\sum_{k=1}^{m} (p_{i_k} - o)(p_{i_k} - o)^T.$$

We choose the orientation of n such that n best aligns with the average normal $\frac{1}{m} \sum_{k=1}^{m} n_{i_k}$ of the points within that node. In the unlikely event that the average normal is 0, the point normals provide no useful information about orientation. In this case we choose the orientation of the eigenvector randomly. In the cases where the node contains fewer than three points, we use Equation 2 and the average of the point normals as the normal of the plane. Note that we can compute the best fitting plane by only storing 13 numbers ($\sum_k p_{i_k} p_{i_k}^T$, $\sum_k p_{i_k}$, $\sum_k n_{i_k}$, m) regardless of the number of points within the node. This constant space representation is clearly beneficial when processing very large point sets.

3.1. Surface Reconstruction

Given adaptive octree with nodes containing a plane, we can easily compute an implicit function to represent the reconstructed surface. First, we extract the portion of the plane that intersects the corresponding node by applying Marching Cubes [1]. Then, using the resulting polygons, we compute the wavelet coefficients of the

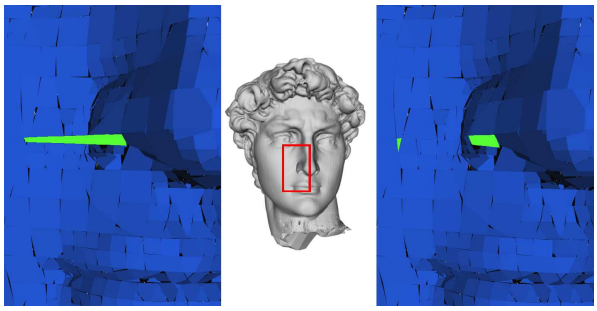

Figure 4: Left: example of merging two disjoint regions of the surface. Right: the same region after enforcing our bounding box condition.

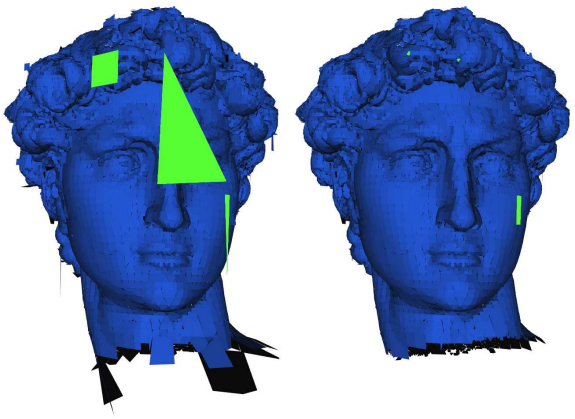

Figure 5: Left: example of extrapolation. Right: the same region after our two-phase pruning pass.

indicator function of the body enclosed by the reconstructed surface with the 3D wavelet rasterization algorithm of Manson et al. [5]. Finally, we extract a polygonal mesh using a variation of Marching Cubes that operates on octrees [31]. We apply this reconstruction algorithm to the adaptive tree structure that we create in Section 4.1.

Note that it may be possible to use other reconstruction methods that can reconstruct surfaces from disconnected polygons for this step. In particular, techniques such as Poisson reconstruction [11] adapted to polygons instead of oriented points may be better for noisy or missing data. Our choice was motivated by choosing a fast rasterization algorithm for volumes bounded by polygons.

4. Compression

We compress our octree data structure in two separate phases: pruning and encoding. During the pruning phase, we perform a collapse of the octree to remove geometrically redundant nodes of the tree. After constructing our adaptive octree, we encode both its structure and the planes in its nodes.

4.1. Pruning

Our pruning phase removes redundant geometric information from the octree and thus reduces the amount of data to be encoded. In flat or even low curvature regions of the surface, refinement of the octree is unnecessary since we store planes inside our nodes and a single plane can represent the reconstructed surface well over this region. Given an error tolerance ϵ, specified by the user, we start at the leaves and prune all children of a node if the parent's error, as defined by Equation 1, is below ϵ. Our error function is monotonically increasing as we prune nodes, so the pruning process will terminate when all of the parent nodes of the current leaves have error greater than or equal to ϵ. Figure 2 shows a 2D example of the pruning phase of the algorithm.

While this simple pruning phase works well, problems can occur even for point sets without noise. We identify two common issues that may happen during pruning, which are presented in Figure 3. The top of the figure shows an example, where pruning produces a single plane merging two disjoint regions of the surface. The bottom row also illustrates a problem where there are no points around an isolated region of the surface and pruning produces a plane that extrapolates the data in an undesired fashion. In all of these cases, the pruning was possible because the error defined by Equation 1 was below the tolerance ϵ specified by the user. Nevertheless, these are artifacts we wish to avoid. Next, we present two simple strategies that overcome these problems in common situations.

Let us assume there is some minimum sampling density δ, associated with the point set. Then the Hausdorff distance between the final linear fits and the point set should be less than or equal to δ, and the pruned data structure should maintain this Hausdorff constraint. Since computing Hausdorff distance between surfaces and very large point sets is expensive, we approximate this distance using bounding boxes. Our solution uses constant space, independent of the number of points inside each node, and requires that we only store the bounding box of the points within each node.

To prevent merging of disjoint regions of the surface, we compute the minimal distance between all pairs of bounding boxes of the children nodes and prevent collapse to the parent node if this distance is greater than 2δ. If the bounding boxes have distance greater than 2δ, then the Hausdorff distance between the plane in the

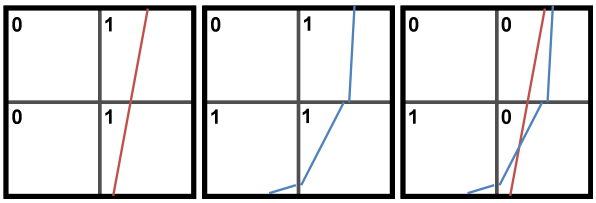

Figure 6: A 2D example demonstrating the octree connectivity prediction. Left: prediction by the parent plane. Center: actual connectivity. Right: overlay of both plane sets and the output bits for the connectivity.

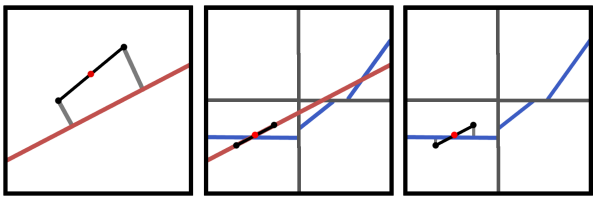

Figure 7: 2D explanation of plane encoding. From left to right: the parent's plane (red) encoded as distances from two points, the children's planes (blue) and the transformed points we measure distance from (black) for the bottom left child, the encoded distances for the bottom left child.

parent node and the points within the children could be greater than δ. The check prevents disjoint regions of the surface from merging even if the geometric error is low. Figure 4 shows the effect both before (left) and after (right) enforcing this condition in 3D.

To combat the extrapolation problem, we perform a two phase pruning process. In the first phase, we prune nodes as before but do not actually remove them. Instead, we mark these nodes for later deletion. During the second phase, we start at the nodes that will become leaves of the pruned octree. For each such node, we compute the Hausdorff distance between the polygon, generated by restricting the node's plane to the node, and the polygon, generated by restricting the same plane to the bounding box of the points within this node. Since the polygons are convex, coplanar and one is contained within the other, this computation is not difficult. If this distance is greater than δ, we then unmark this node and perform the same procedure to each of its children. When all of the nodes have been processed, we delete all marked nodes. Figure 5 shows the extrapolation problems before (left) and after (right) we apply this procedure to a 3D surface.

Note that these heuristics do not enforce any strict bound on the Hausdorff distance between the reconstructed surface and the point cloud, and there will be cases where the topology of the surface could change during pruning. However, we have found that the proposed techniques work well in practice and resolve many common cases when the error in Equation 1 is low but further pruning is undesirable.

4.2. Encoding

The second phase of our compression algorithm is a breadth first traversal of the octree to progressively encode both its structure and the planes inside its nodes. For this compression, we utilize an arithmetic encoder [32] (a form of variable-length entropy encoding), which can generate fractional bit compression using the probability distribution of the output symbols. The more skewed the distribution, the higher the compression. Hence, we use a predictive encoding of both the octree structure and the planes. Furthermore, we typically achieve higher compression rates by building conditional probability distributions using a context. For all of the different types of symbols we output, we build different distributions for each type and use the current depth of the octree as the context.

4.2.1. Encoding the octree structure

The result of the pruning algorithm from Section 4.1 is an adaptive octree whose nodes are one of the following three types: empty nodes, leaves with data, or interior nodes with data, with only two symbols possible at the maximum depth.

Given a parent node and its encoded plane, we predict if a child contains data by intersecting the parent's plane with the child's node. We predict that nodes intersecting this plane contain data and that there is no data in the remaining children nodes. We then encode the difference between our prediction and the actual structure by using a simple XOR operation on the bits.

The prediction is based on the observation that, if the geometry of the parent is a good approximation to the geometry of the children, we will only encode a value of zero. This operation will create an output with a skewed distribution that will compress well with an arithmetic encoder. Figure 6 shows a 2D example of this prediction. The left image shows the parent's plane in red and the predicted connectivity. The center image shows the actual children and the desired connectivity. The right image shows the two datasets overlaid and the symbols for the octree connectivity that are output during encoding. For children nodes that contain data, we also output a single bit specifying whether or not the node is a leaf.

4.2.2. Encoding the planes

To encode the planes of each node, we measure the signed distance from three points on an equilateral tri-

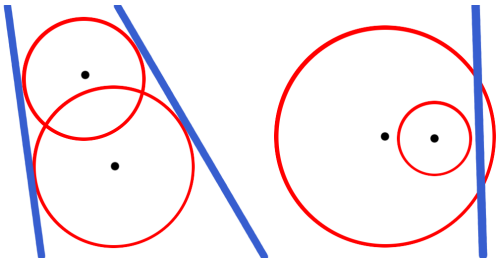

Figure 8: Reconstructing lines from two distance values in 2D yields multiple solutions (left) or no exact solutions (right).

Figure 9: From left to right: original surface reconstructed without pruning, the surface reconstructed using an adaptive octree with 80 percent of nodes pruned, the surface reconstructed using an adaptive octree with 98 percent of nodes pruned. The final compressed sizes of each model are 1179.18KB, 282.47KB, and 46.18KB respectively.

angle with center of mass c, vertices q_ℓ, $\ell = 0, 1, 2$, and sides of length $2^{-(j+1)}$ with j being the depth of the node. We first position these points by computing a minimal rotation that aligns the normal of the triangle used to compute the parent node with the encoded normal of the parent's plane and rotate the triangle about c. Next, we scale the triangle by a factor of 2^{-1} about its center (thus the length of its sides becomes $2^{-(j+1)}$ where j is the level of the child node). Now, to encode the plane of a child, we intersect the plane of the parent node with the child node and translate the triangle such that c coincides with the center of that intersection. If there is no intersection, then we translate the triangle so that c coincides with the closest corner of the child node to the parent's plane. The 2D version of this process is depicted in Figure 7. The left image shows the parent node's plane in red, encoded as a distances from two points. The center image shows the children's planes in blue and the translated, rotated and scaled line segment (corresponding to our equilateral triangle in 3D) in black for the bottom left child. The right image shows the plane of the bottom left child encoded as distances to our estimated points.

We then simply compute the signed distance v_ℓ from the vertices q_ℓ of the resulting triangle to the child's plane by evaluating the child's plane equation at q_ℓ. The translation/rotation of the triangle is designed to perform an educated guess of where the child's plane will be. If our guess is a good one, then the magnitude of these distances will be close to zero on average and will compress well.

Since our construction ensures that the plane in each node passes through that node and the size of the nodes decreases by a factor of 2 at each level, the number of bits necessary to represent v_ℓ decreases linearly with the depth of the octree. Given a node at level j (with side lengths 2^{-j}), the distance from c to any plane intersecting the node is less than or equal to $2^{-j}\sqrt{3}$ because c is contained within the node by construction. Note that the q_ℓ's can actually lie outside the node. Our equilateral triangle has sides of length $2^{-(j+1)}$. Therefore, the distance from c to q_ℓ is $2^{-(j+1)}/\sqrt{3}$. By the triangle inequality, the distance v_ℓ from q_ℓ to a plane within the node is bounded by

$$v_\ell \leq 2^{-j}\frac{7}{2\sqrt{3}},$$

which implies the magnitude of v_ℓ decreases by a factor of 2 each level of the octree. Using this property, we quantize v_ℓ by multiplying by $(2^{s-j}-1)\frac{2^{j+1}\sqrt{3}}{7}$, rounding to the nearest integer, and storing the resulting $s-j$ bits.

Note that the signed distances from three points to a plane are not sufficient to uniquely determine that plane. There are at most two planes that have the same signed distances to three non-collinear points and we need an additional bit to select the correct one. The 2D illustration of this problem is depicted on the left image in Figure 8. If we consider the two circles with centers q_ℓ and radii $|v_\ell|$, there are two lines that are at signed distance v_ℓ from the points q_ℓ, and these are the common tangent lines to the two circles. Moreover, since we quantize our values (and therefore do not use exact values) sometimes it is possible that no such line exists as shown in the right image in Figure 8.

Next, we describe how we reconstruct the planes in each node from the q_ℓ and the quantized data v_ℓ. Given the vertices q_ℓ and values v_ℓ, the plane with normal n and offset b, compatible with these values, is given by

$$\begin{pmatrix} q_0^T & -1 \\ q_1^T & -1 \\ q_2^T & -1 \end{pmatrix}\begin{pmatrix} n \\ b \end{pmatrix} = \begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix},$$

where $n \cdot n = 1$. Note that we can directly find b as a function of n

$$b = n \cdot \frac{q_0+q_1+q_2}{3} - \frac{v_0+v_1+v_2}{3} = n \cdot c - \bar{v}, \quad (3)$$

Data Set	Num Points	Park and Lee	Ours
African Statue	220,318	47.93	38.61
Bunny	362,272	154.37	53.14
David Head	4,520,803	405.33	282.47
Atlas	8,195,996	1269.63	309.99
Awakening	7,621,482	722.65	235.86
Barbuto	6,577,132	1030.25	331.03

Figure 10: From let to right: model, number of points in the input, the size in kilobytes of Park and Lee's results and our results.

where $\overline{v}$ is the average of the v_ℓ's. Substituting this value into the equation yields

$$\begin{pmatrix} (q_0 - c)^T \\ (q_1 - c)^T \\ (q_2 - c)^T \end{pmatrix} n = \begin{pmatrix} v_0 - \overline{v} \\ v_1 - \overline{v} \\ v_2 - \overline{v} \end{pmatrix}.$$

The matrix on the left of this equation does not have full rank and, in fact, has a nullspace spanned by the unit normal n_q to the triangle with vertices q_ℓ. We can exactly write down the line that represents the solution space of the system by finding a point $\hat{n}$,

$$\hat{n} = \begin{pmatrix} (q_0 - c)^T \\ (q_1 - c)^T \\ (q_2 - c)^T \end{pmatrix}^+ \begin{pmatrix} v_0 - \overline{v} \\ v_1 - \overline{v} \\ v_2 - \overline{v} \end{pmatrix},$$

from the solution space using the pseudo-inverse of the matrix. The entire space of solutions is then given by $n = \hat{n} + n_q t$, and b is found by substituting this value into Equation 3. The nonlinear constraint $n \cdot n = 1$ yields a quadratic equation in t with roots

$$t_{1,2} = -\hat{n} \cdot n_q \pm \sqrt{(\hat{n} \cdot n_q)^2 - \hat{n} \cdot \hat{n} + 1}. \qquad (4)$$

Therefore, we also output a single bit that determines which of the two roots t_1 or t_2 of the quadratic represents our plane equation. Moreover, we can even avoid writing this extra bit in some cases, since we know that the plane of the child must intersect the child node. If one of the two planes corresponding to the roots of the quadratic equation does not intersect the node, we simply omit this extra bit.

Because we quantize the values v_ℓ, it is possible that the roots in Equation 4 may not be real. Therefore, rather than using the above steps for finding t, we minimize the function $(n \cdot n - 1)^2$. This quartic function has three critical points, two of which are identical to the roots of the quadratic equation. If these roots exist, then an exact solution exists, and we use this solution. However, in the case where these roots do not exist, we use

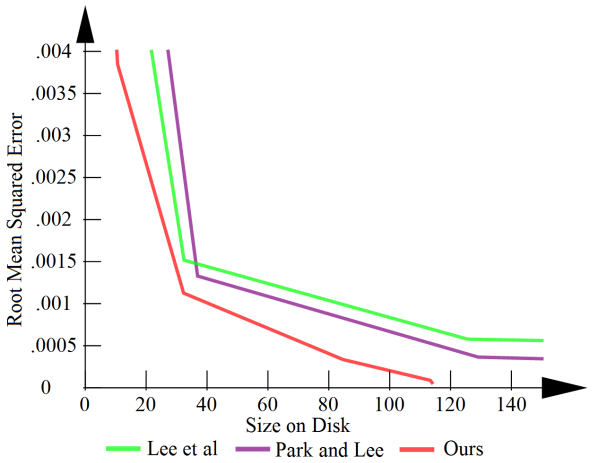

Figure 11: The RMS error from compressing a polygonal model of the Stanford Bunny shown in Fig.14 using various compression methods.

the third critical point

$$t_3 = -\hat{n} \cdot n_q,$$

to generate the plane. Note that if we use this point, we do not have to output an extra bit to distinguish between the solutions because only one real solution exists.

5. Results

Our compression technique has two main components: building the adaptive octree structure that represents the surface and encoding this structure and the planes in its nodes. One important step of the first component is the pruning phase, described in Section 4.1. We have found that pruning around 80% of the original octree nodes results in a lossy compression without any noticeable changes in visual quality. This effect is mainly due to the fact that we can prune regions with low curvature without much loss to the visual appearance of the shape since we use planes to represent the geometry. Figure 9 shows an example where using a very small value for the tolerance ϵ removes about 80% of the nodes without much, if any, loss in visual fidelity. Since the pruning phase acts like smoothing in the presence of noise, pruning can even improve the quality of the reconstructed surface for noisy input data. Figure 9 also illustrates the dependence of the quality of the surface on the values of the tolerance ϵ. If we continue to increase ϵ, the piecewise linear nature of the reconstructed surface becomes apparent and the surface quality degrades. However, even if we keep only 2% of the nodes from the original octree, the shape is still recognizable (Figure 9, right).

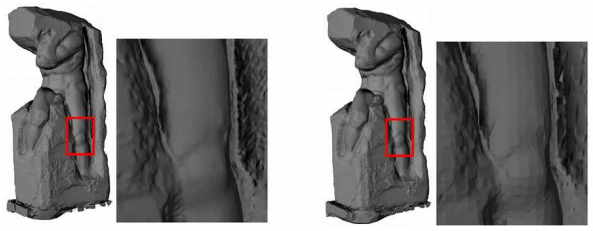

Figure 12: A compressed model of Atlas generated using our method at 291KB (left) and Park and Lee at 332KB (right).

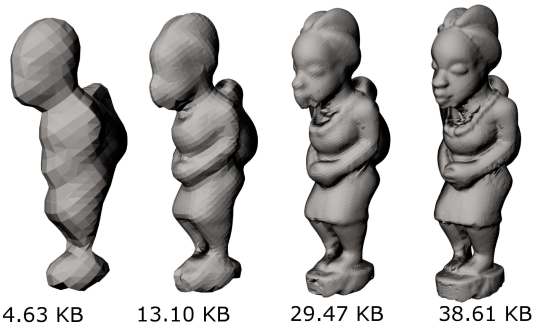

Figure 13: African Statue progressively decoded. 133,694 polygons in final mesh.

Next, we compare the results of our algorithm with other known compression techniques. Note, that this is not an easy task since different compression methods use various types of input data or produce different output. For example, Park and Lee [29] use point cloud as input and output an octree whose leaves contain planes (as we do), but only use this structure for visualization and do not create a surface. Lee et al. [26] take a grid of values from an implicit function as input and produce a compressed structure that can be used to generate a polygonal model of the surface. Note that in order to perform a fair comparison between the different compression approaches, we also need a notion of a ground truth to compute the error of the compressed results.

First, we compare our method to the compression algorithm described in Park and Lee [29]. Since their output is a compressed tree structure, we need to post process this tree (using wavelet rasterization and polygonal extraction, as described in Section 3.1) and generate a polygonal model of the surface. Using Metro [33], we then compute the RMS error between the surfaces generated from the uncompressed and compressed octrees, respectively, obtained from both our method and Park and Lee [29]. In each case, we choose settings for both algorithms such that the RMS error reported by Metro is approximately the same for both methods. For all examples in the paper we use a maximum octree depth $d = 10$ for all examples in the paper except the African statue in Figure 13 and the bunny in Figure 14, where $d = 8$. The maximum octree depth should be proportional to the minimum sampling density, and both the African statue and bunny have far fewer samples than the other models. All of our examples were created from real scanner data (laser range scanners for Figures 9, 12, 14, 15, 16 and a structured light scanner for Figure 13) and contain noise.

Figure 10 shows that our method out-performs Park and Lee in all examples and, on average, provides about a factor of three better compression rates. Depending on the model, the improvement is between 20% and 400%. For example, the African statue does not compress as well because of the high amount of noise from the structured light scanner, but other models, such as Awakening, compress well because the surface has large, nearly planar faces. Figure 12 shows an example reconstruction of Atlas with both techniques at approximately the same file size. Since Park and Lee's method is not as efficient, there is significant loss of details in the model.

Second, we compare our method to compression schemes at different stages of the reconstruction pipeline (see Figure 1). Note that we cannot use real scanner data to perform a quality comparison versus a ground-truth model, since the models the data was obtained from are not available. Therefore, we use a polygonal model as input and modify our octree construction in Section 2 (and that of Park and Lee) to build centroids and covariance matrices by integrating over the polygons clipped to the current nodes. We then compress this octree with each of the respective techniques, perform wavelet rasterization [5] to create an implicit function and extract polygons. We also perform wavelet rasterization on the uncompressed octree to create an implicit function and provide this function to Lee et al [26] for their implicit surface encoder. For each of these methods, we use the compressed representation to construct polygons and compare the output surface to the original polygon model using the RMS error reported by Metro [33].

Figure 11 displays the results obtained from these methods when applied to the model of the Stanford bunny. Similar to our previous tests, our method outperforms Park and Lee, especially at lower error values. Our compression results for the same amount of error are also better than the results, obtained using the algorithm from Lee et al.

Finally, we would like to point out that our method encodes a progressive representation of the surface that

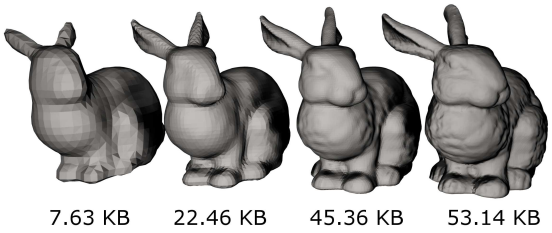

Figure 14: Bunny progressively decoded. 247,064 polygons in final mesh.

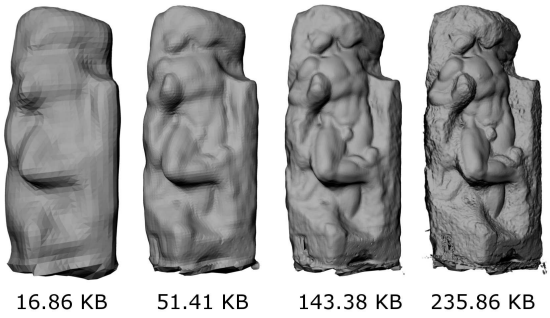

Figure 15: Awakening Statue progressively decoded. 2,283,540 polygons in final mesh.

can be decoded and displayed during transmission. Figures 13, 14, 15, and 16, depict examples of decoded surfaces with the size of the data transmitted up to that point as well as the final surface generated from the fully transmitted data stream (far right images).

A potential limitation of our algorithm is the fact that the pruning phase could perform undesirable collapses or prevent collapses in regions that should collapse. We heuristically estimate the Hausdorff distance during the pruning phase, and while our approach performs well in practice, we have no theoretical guarantees that the reconstructed surface will have the same geometry or topology as the real model. Another drawback comes from the progressive nature of our compression/decoding. While this type of encoding stores multiple resolutions that can provide a view of the reconstructed surface without receiving all of the encoded data, it does not allow for random access to arbitrary data in the model.

6. Future Work

Our main contribution is the development of a new method for surface compression that builds and encodes an adaptive octree representation of this surface. We decode the octree by solving a constrained optimization problem that has closed-form solution. To display the surface, we use wavelet rasterization [5] to compute an approximation to the indicator function of the solid body whose boundary is the surface and reconstruct the shape by extracting a level set of this function using Marching Cubes. The proposed algorithm is fast (small examples like the African statue take less than 0.7s for encoding and about 0.2s for decoding, while larger examples like Atlas take about 15s for encoding and 2s for decoding on an Intel Core i7 960 processor), resilient to noise, and outperforms the state of the art compression algorithms.

In the future, we would like to study the advantages and disadvantages of using functions other than linear polynomial fits at the octree nodes. Higher order polynomials should allow better approximation, further pruning of the octree and hopefully better compression rates. However, their use comes with more complicated decoding strategies and dangers of overfitting.

While we performed a comparison of the accuracy of the reconstructed surfaces according to RMS error, we mainly see surfaces through the variation of lighting (and hence normals) on the surface. Therefore, good normal reproduction is also important, especially in the presence of sharp features [34]. We have not investigated the quality of normal approximation using our method. Clearly the choice of reconstruction method in Section 3.1 will have a significant effect on normal quality. It would be interesting to compare the different techniques to determine if one compression method has better normal reproduction, but potentially worse positional error, than others.

Another avenue we would like to pursue is the development of out-of-core compression algorithm similar in style to the out-of-core surface reconstruction algorithms from [3, 4]. Currently, the input datasets we use from the Digital Michelangelo project are randomly sub-sampled by a factor of 50 from the full point distribution. While our method could handle the full point distribution, the extra detail would be lost at the reconstruction step since we currently limit the maximum octree depth to 10 (1024^3 grid). Our current method is limited by the size of the octree in memory and not the size of the input data, and a streaming, out-of-core compression algorithm would allow us to create deeper trees and use larger datasets.

Acknowledgements

This work was supported by DARPA grant HR0011-09-1-0042 and the NSF grant DMS 0915231.

Figure 16: Barbados Statue progressively decoded. 1,990,811 polygons in final mesh.

References

[1] W. Lorensen, H. Cline, Marching cubes: A high resolution 3d surface construction algorithm, in: Proceedings of SIGGRAPH, 1987, pp. 163–169.

[2] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, D. Fulk, The digital michelangelo project: 3d scanning of large statues, in: Proceedings of SIGGRAPH, 2000, pp. 131–144.

[3] M. Bolitho, M. Kazhdan, R. Burns, H. Hoppe, Multilevel streaming for out-of-core surface reconstruction, in: Symposium on Geometry Processing, 2007, pp. 69–78.

[4] J. Manson, G. Petrova, S. Schaefer, Streaming surface reconstruction using wavelets, Computer Graphics Forum 27 (5) (2008) 1411–1420.

[5] J. Manson, S. Schaefer, Wavelet rasterization, Computer Graphics Forum 30 (2) (2011) 395–404.

[6] R. Schnabel, R. Klein, Octree-based point-cloud compression, in: Proceedings of Eurographics Symposium on Point-Based Graphics, 2006, pp. 111–120.

[7] Y. Huang, J. Peng, C.-C. J. Kuo, M. Gopi, Octree-based progressive geometry coding of point clouds, in: Proceedings of the Symposium on Point-Based Graphics, 2006, pp. 103–110.

[8] Y. Huang, J. Peng, C.-C. J. Kuo, M. Gopi, A generic scheme for progressive point cloud coding, Visualization and Computer Graphics, IEEE Transactions on 14 (2) (2008) 440 –453.

[9] M. Deering, Geometry compression, in: Proceedings of, SIGGRAPH, ACM, 1995, pp. 13–20.

[10] M. Kazhdan, Reconstruction of solid models from oriented point sets, in: Symposium on Geometry Processing, 2005, pp. 73–82.

[11] M. Kazhdan, M. Bolitho, H. Hoppe, Poisson surface reconstruction, in: Symposium on Geometry Processing, 2006, pp. 61–70.

[12] D. Laney, M. Bertram, M. Duchaineau, N. Max, Multiresolution distance volumes for progressive surface compression, International Symposium on 3D Data Processing Visualization and Transmission (2002) 470–479.

[13] V. Chandrasekaran, M. Wakin, D. Baron, R. Baraniuk, Representation and compression of multidimensional piecewise functions using surflets, Information Theory, IEEE Transactions on 55 (1) (2009) 374 –400.

[14] J. Li, C.-C. J. Kuo, Progressive coding of 3-d graphic models, Proceedings of the IEEE 86 (6) (1998) 1052–1063.

[15] G. Taubin, A. Guéziec, W. Horn, F. Lazarus, Progressive forest split compression, in: Proceedings of SIGGRAPH, 1998, pp. 123–132.

[16] C. L. Bajaj, V. Pascucci, G. Zhuang, Progressive compression and transmission of arbitrary triangular meshes, in: Proceedings of IEEE VISUALIZATION, 1999, pp. 307–316.

[17] D. Cohen-Or, D. Levin, O. Remez, Progressive compression of arbitrary triangular meshes, in: Proceedings of IEEE Visualization, 1999, pp. 67–72.

[18] R. Pajarola, J. Rossignac, Compressed progressive meshes, Visualization and Computer Graphics, IEEE Transactions on 6 (1) (2000) 79 –93.

[19] O. Devillers, P.-M. Gandoin, Geometric compression for interactive transmission, in: Proceedings of the conference on Visualization, 2000, pp. 319–326.

[20] P. Alliez, M. Desbrun, Progressive compression for lossless transmission of triangle meshes, in: Proceedings of SIGGRAPH, 2001, pp. 195–202.

[21] P.-M. Gandoin, O. Devillers, Progressive lossless compression of arbitrary simplicial complexes, in: Proceedings of SIGGRAPH, 2002, pp. 372–379.

[22] J. Peng, C.-C. J. Kuo, Geometry-guided progressive lossless 3d mesh coding with octree (ot) decomposition, in: ACM SIGGRAPH, 2005, pp. 609–616.

[23] J. Peng, C.-S. Kim, C.-C. J. Kuo, Technologies for 3d mesh compression: A survey, Journal of Visual Communication and Image Representation 16 (2005) 688–733.

[24] T. Park, H. Lee, C.-H. Kim, Progressive compression of geometry information with smooth intermediate meshes, in: Proceedings of the Iberian conference on Pattern Recognition and Image Analysis, Part II, 2007, pp. 89–96.

[25] S. Valette, R. Chaine, R. Prost, Progressive lossless mesh compression via incremental parametric refinement, in: Proceedings of the Symposium on Geometry Processing, 2009, pp. 1301–1310.

[26] H. Lee, M. Desbrun, P. Schröder, Progressive encoding of complex isosurfaces, ACM Trans. Graph. 22 (3) (2003) 471–476.

[27] T. Lewiner, L. Velho, H. Lopes, V. Mello, Simplicial isosurface compression, in: Vision, Modeling, and Visualization, 2004, pp. 299–306.

[28] D. Saupe, J.-P. Kuska, Compression of isosurfaces for structured volumes with context modelling, in: International Symposium on 3D Data Processing Visualization and Transmission, 2002, pp. 384 – 390.

[29] S.-B. Park, S.-U. Lee, Multiscale representation and compression of 3-d point data, Trans. Multi. 11 (1) (2009) 177–182.

[30] J. Shapiro, Embedded image coding using zerotrees of wavelet coefficients, IEEE Transactions on Signal Processing 41 (12) (1993) 3445–3462.

[31] S. Schaefer, J. Warren, Dual marching cubes: Primal contouring of dual grids, in: Proceedings of Pacific Graphics, 2004, pp. 70–76.

[32] F. Wheeler, Adaptive arithmetic coding source code, http://www.cipr.rpi.edu/˜wheeler/ac (1996).

[33] P. Cignoni, C. Rocchini, R. Scopigno, Metro: Measuring error on simplified surfaces, Computer Graphics Forum 17 (2) (1998) 167–174.

[34] Y. Ohtake, A. Belyaev, A. Pasko, Dynamic mesh optimization for polygonized implicit surfaces with sharp features, The Visual Computer 19 (2–3) (2003) 115–126.