Eurographics Symposium on Point-Based Graphics (2004)
M. Alexa, S. Rusinkiewicz, (Editors)

Progressive Compression of Point-Sampled Models

M. Waschbüsch, M. Gross, F. Eberhard, E. Lamboray and S. Würmlin

Computer Graphics Laboratory, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland

Abstract

We present a framework for progressive compression of point-sampled models. It is based on a multiresolution decomposition of the point set and thus easily allows for progressive decoding. Our method is generic in the sense that it can handle arbitrary point attributes using attribute-specific coding operations. Furthermore, no resampling of the model is needed and thus we do not introduce additional smoothing artifacts. We provide coding operators for the point position, normal and color. Particularly, by transforming the point positions into a local reference frame, we exploit the fact that all point samples are living on a surface. Our framework enables for compressing both geometry and appearance of the model in a unified manner. We show the performance of our framework on a diversity of point-based models.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational geometry and object modeling—Curve, surface, solid, and object representations; E.4 [Data]: Coding and information theory—Data compaction and compression

1. Introduction

In recent years, 3D models extended the digital media spectrum in many application fields, such as e-commerce, entertainment, and education. While audio, image and video are still the dominating media type, 3D models become more and more important. 3D models can be either generated from scratch in a modeling tool or digitized from a physical object using a 3D scanner. In the latter case, acquisition devices such as range scanners often directly produce point-sampled representations from the object. Depending on the employed technique, each point sample is assigned multiple attributes defining its geometry and appearance. These include position, color, normal and radius. Generating a consistent triangle mesh for complex objects can be very time consuming and difficult. It is therefore beneficial to directly represent 3D models by the irregular collection of point samples which can be easily extended towards a multiresolution representation. One of the key advantages of point-sampled geometry over triangle meshes is that no explicit connectivity information needs to be stored. Instead, continuous surface reconstruction is done during rendering. Moreover, both appearance and geometry attributes of the point samples can be handled homogeneously. This is in contrast to triangle meshes where geometry is stored differently than appearance. The latter typically has to be stored in textures.

Recent research in point-sampled geometry coped with surface analysis, filtering, resampling, feature extraction, and high-quality rendering. See [ADG*03] for an overview. By using these techniques it is easy to come up with an efficient and non-expensive 3D content creation system [ZPKG02] for highly complex models from real-world objects.

However, with increasing complexity of 3D models, advanced compression of point-sampled models is needed. In this paper we propose an efficient and general multiresolution framework for compression of point-sampled models. Our scheme progressively compresses arbitrary attributes of point samples in a common way. We exploit similarities of each attribute by finding a specific order of the samples which minimizes entropy. To that end, we employ a matching algorithm which finds neighborhoods with high correlation based on distance measures for all attributes we want to compress. The data is then transformed into a multiresolution representation and decorrelated by coding the differences between subsequent resolutions. To adapt the transform to the specific characteristics of each attribute, we use a predictive encoding scheme. We define in particular both a

distance measure and prediction operators for compression of point positions. For the prediction we use least-squares planes as a local approximation of the surface. We complete our compression scheme by also providing coding operators for colors and normals of the points. The multiresolution nature of our method easily allows for progressive decompression.

2. Related Work

Many previous geometry compression efforts have focused on triangle mesh compression. Methods which separately encode connectivity and vertex coordinates require 10 to 20 bits per vertex [TG98, TGHL98, Ros99]. Devillers et al. [DG00] achieve about 9 bits per vertex by reconstructing the connectivity from the vertex coordinates during decoding. Khodakovsky et al. reduce the total bit rate for geometry information by mapping densely sampled meshes to a semi-regular mesh [KSS00, KG04]. This approach does not need explicit connectivity encoding and requires around 3 to 5 bits per vertex for fine visual quality. Comparable results are achieved by Gu et al. while remeshing arbitrary surfaces onto completely regular structures, called "geometry images", which are basically 2D arrays [GGH02]. They are compressed using conventional image coders. The latter approach enables the encoding of additional appearance attributes using the same implicit surface parameterization, whereas other approaches require explicit encoding schemes using e.g. texture coordinates. Geometry images require a certain mesh topology and a global parameterization which is very hard to achieve. Many other advanced methods demand at least for a local parameterization and local differential properties. Progressive encoding of mesh data structures can also be implemented by the recursive use of edge collapse operations [Hop96].

More recently, the growing interest in point-based graphics led to the investigation of compression algorithms for point-based representations. In QSplat [RL00], which is a multiresolution rendering system based on a hierarchical bounding sphere data structure and splat rendering, each node of the data structure is quantized to 48 bits, including color and surface normal data. A similar performance is achieved by the intra-frame 3D video encoder of Wuermlin et al. [WLSG03]. Botsch et al. use an octree data structure for storing point-sampled geometry and show that the geometry of typical data sets can be encoded with less than 5 bits per point sample [BWK02]. Similar performance is achieved by Fleishman et al. who propose a progressive point set coder based on the projection of points onto local polynomial surface approximations [FCOAS03]. But their resampling method based on the moving least-squares (MLS) projection operator tends to smooth out sharp features. Furthermore, the decompression is very time consuming because the MLS projection also has to be applied during decoding.

3. Algorithm Overview

Our compression scheme comes as a general framework for coding a set of arbitrary point attributes. It is based on a multiresolution decomposition of the whole point set. Our predictive differential coding scheme has the ability of decorrelating the information contained in the model by exploiting local coherencies. The multiresolution approach splits the information into several frequency bands. This property easily allows for progressive decoding by successively adding more detail to the model.

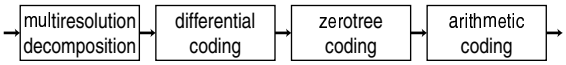


Figure 1: *Compression pipeline.*

An overview of the whole compression pipeline is given in Figure 1. In the first step, the multiresolution hierarchy is built up by recursively computing coarser approximations of the point set. This task requires local neighborhood relations between point samples that, unlike in triangle meshes, are not explicitly available in point based models. To that end, we employ a search algorithm that delivers us neighboring points. As a benefit, we can control the search in a way that gives us good correlations between the neighbors in all attributes and not only in the geometry. Both the neighborhood search and the multiresolution decomposition are described in section 4.

Next, we compute the hierarchy of detail coefficients that describes how to successively reconstruct the original model from the lowest-resolution point set. The respective computations are driven by a prediction which is customized to the specific characteristics of each point attribute separately. After completing one hierarchy layer we immediately perform the quantization of the coefficients and propagate the quantization error into the computation of the next layer. This method prevents a recursive accumulation of the quantization error over all multiresolution layers and thus minimizes the total error of our coding method. A detailed discussion follows in section 5.

We eventually end up with a set of quantized detail coefficients. They are not yet fully decorrelated but still contain some coherencies which can be eliminated very efficiently by a zerotree coder. Finally, the data is further compressed by arithmetic coding. Both coders allow for progressive decoding. In addition to the conventional zerotree coder, we present a modified algorithm with a progressive behavior that better collaborates with our multiresolution framework. All those methods are further explained in section 6.

The first two stages of the pipeline have to be customized for the specific characteristics of the compressed point attributes. Stage one needs a distance measure for the neighborhood search, stage two requires a prediction operator

and a coordinate transform. We provide specialized measures and operators for coding the point positions. We further show the extendibility of our framework towards other attributes by proposing coding methods for point colors and normals.

4. Multiresolution Decomposition

In the first stage, we compute a multiresolution hierarchy of the point cloud by recursively generating a sequence of subsampled versions $(\lambda^{-1},\ldots,\lambda^{-d})$ of the original point set λ^0. We use here a notation related to [Swe95] where λ^{-k} denotes the point set in the k-th level of the multiresolution hierarchy. A specific point in a subsampled model is identified by λ_i^{-k}. We build the hierarchy bottom up, i.e. from high to low resolution, by decomposing in each step the point set into disjunctive point pairs and contracting each pair to an average point by averaging its associated attributes. In this way, we obtain a forest of binary trees of depth d, of which each layer describes a certain resolution. As the point pairs describe neighborhood relations between two points in the attribute space, each subtree in the hierarchy with depth k describes a neighborhood of 2^k points, as illustrated in Figure 2. All those neighborhood relations are implicitly stored in the point ordering of the full resolution model and hence are still available after decompression.

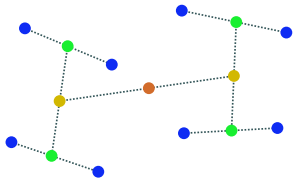


Figure 2: 8-*neighborhood after* 3 *passes of hierarchical point contraction.*

Note that such a hierarchy can only be built if each layer in the forest but the root layer contains an even number of points. Thus, the original model has to consist of a multiple of 2^d points. If this is not the case, the maximum 2^d-1 remaining points are stored separately and compressed by entropy coding only. For a moderate number of recursions, like $d=10$, this number is relatively small compared to the overall size of typical models that often consist of several $100k$ points.

The compression performance largely depends on the pair decomposition because it considers each average point as an approximation to each of its sons. Thus, we preferably want to contract "good" pairs of points with similar attribute values. Finding the best set of pairs basically is an optimization problem in the space spanned by all point attributes. This can be described as a minimum weight perfect matching problem [LP86] in an undirected, weighted graph $G=(V,E)$ with vertices V and weighted edges E. Its task is to find a set of edges $M\subset E$ of cardinality $|M|=|V|/2$ such that no two edges share a vertex in common and the sum of edge weights in M is minimal. In a complete graph with an even number of vertices, such a matching always exists. To speed up the matching process, we do not construct a complete graph but only an adjacency graph connecting each point with its $k=12$ nearest neighbors. In the very unlikely event when no matching is found we can still increase k and let the algorithm run again, but this never happened in our experiments. A classical solution of the matching problem is provided by Edmonds' blossom shrinking algorithm [Edm65]. In our implementation, we use the faster method described in [CR99].

The weight $w_{i,j}$ for each edge $\{i,j\}$ can be expressed as a sum of distance functions δ_A, describing for each attribute A the difference of its values at the points λ_i^{-k} and λ_j^{-k} adjacent to that edge:

$$w_{i,j}=\sum_A \delta_A(i,j).$$

In practice, we incorporate here the point positions $\mathbf{x}^{-k}$ and normals $\mathbf{n}^{-k}$. The distance between normals corresponds to the cosine of their enclosing angle:

$$\delta_{\mathbf{n}}(i,j)=\frac{1-\mathbf{n}_i^{-k}\cdot\mathbf{n}_j^{-k}}{2}.$$

If we consider as a position weight only the Euclidean distance, we may end up after two recursive subsampling passes in 4-neighborhoods consisting of linearly arranged points. This will lead to instabilities in the subsequent detail coefficient computation. So we additionally include the previous direction of the contraction of the points λ_{2i}^{-k+1} and λ_{2i+1}^{-k+1} towards λ_i^{-k} and similarly for the contraction to λ_j^{-k}:

$$\begin{aligned}\delta_{\mathbf{x}}(i,j)=&\left|\mathbf{x}_i^{-k}-\mathbf{x}_j^{-k}\right|\\&+\left|\frac{\mathbf{x}_{2i}^{-k+1}-\mathbf{x}_{2i+1}^{-k+1}}{|\mathbf{x}_{2i}^{-k+1}-\mathbf{x}_{2i+1}^{-k+1}|}\cdot\frac{\mathbf{x}_i^{-k}-\mathbf{x}_j^{-k}}{|\mathbf{x}_i^{-k}-\mathbf{x}_j^{-k}|}\right|\\&+\left|\frac{\mathbf{x}_{2j}^{-k+1}-\mathbf{x}_{2j+1}^{-k+1}}{|\mathbf{x}_{2j}^{-k+1}-\mathbf{x}_{2j+1}^{-k+1}|}\cdot\frac{\mathbf{x}_i^{-k}-\mathbf{x}_j^{-k}}{|\mathbf{x}_i^{-k}-\mathbf{x}_j^{-k}|}\right|.\end{aligned}$$

Figure 3 shows the resulting 64-neighborhoods after six recursive executions of the matching algorithm.

5. Predictive Differential Coding

Next, we successively compute a binary forest of detail coefficients $(\gamma^{-d},\ldots,\gamma^{-1})$ of which each layer γ^{-i} contains the information that is necessary to reconstruct λ^{-i+1} from λ^{-i}. The calculations are done with the help of two attribute-specific operators: a coordinate transform $\mathbf{C}$ and a prediction operator $\mathbf{P}$. With regard to later decompression, $\mathbf{C}$ has to be invertible but not $\mathbf{P}$. We finally end up in a set of coefficients $(\lambda^{-d},\gamma^{-d},\ldots,\gamma^{-1})$ which is sufficient for reconstruction of the original model. Because each detail layer γ^{-i} has the same size as its corresponding average layer λ^{-i}, the whole

Figure 3: 64-*neighborhoods after* 16 *matching passes.*

coefficient set is still of the same size as the original model. But the new data is much more decorrelated and therefore more suitable for compression.

Because high-resolution detail coefficients recursively depend on their lower-resolution counterparts, quantization errors that will be introduced due to compression would accumulate in each recursion. To prevent this, we compute the detail coefficients top down from γ^{-d} to γ^{1}, quantize them after each recursion and propagate the quantization error to the next higher resolution: After determining and quantizing the coefficients γ^{-k} we first update the next average layer λ^{-k+1} out of λ^{-k} and the quantized γ^{-k} and then recursively proceed computing the detail coefficients γ^{-k+1} of the next higher resolution layer. The decoder later successively reconstructs higher-resolution models λ^{-k+1} from the average points λ^{-k} and the detail γ^{-k}. There, all the data processing can be done in place by recursively substituting λ^{-k} and γ^{-k} by λ^{-k+1}, similarly to the lifting scheme [Swe95].

The actual computation of the detail coefficients is performed as illustrated in Figure 4. For each contraction pair $(\lambda_{2i}^{-k+1}, \lambda_{2i+1}^{-k+1})$ and its corresponding average point λ_i^{-k}, we try to predict λ_{2i}^{-k+1} from λ_i^{-k} using a prediction operator **P** and store the prediction error in γ_i^{-k}. With this information, both higher resolution points can be reconstructed due to symmetry. In many cases, the prediction does not perform well if it is carried out in the global space of the model. So we apply beforehand a local coordinate transform **C** which depends on the points of λ^{-k} and better characterizes the attribute space locally. Figure 5 illustrates the operator framework both for encoding and decoding. Note that the decoder performs the same prediction **P** as the encoder, whereas the coordinate transform **C** during decoding is inverse in order to obtain again the global model coordinates from the coefficients encoded in the local reference frames.

The whole transformation is recursively applied up to a depth d. As the coherencies between neighboring points get worse for lower resolutions, a depth between $d = 6$ and $d = 8$ usually gives the best compression results. Because the coordinate transform **C** depends on neighborhood relations in the point set λ^{-k}, the multiresolution forest is built some recursions higher for getting neighborhood relations in λ^{-d}. We actually use there a depth of $d + 2$ because our coordinate transforms rely on 4-neighborhood relations.

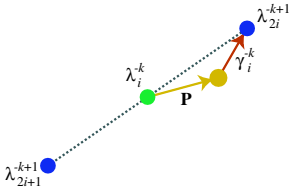
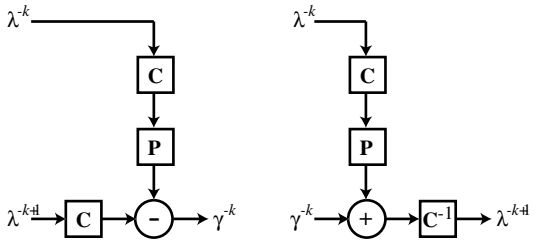


Figure 4: *Prediction and detail coefficient.*



(a) Encoding. (b) Decoding.

Figure 5: *Operator framework.*

In the following, we provide operators **C** and **P** for the compression of the point positions, colors and normals.

5.1. Positions

For compression of the point positions, we want to exploit the fact that all points are arranged on a surface and not for instance in a volume. So we locally approximate the geometry by a least-squares plane and store the prediction error relative to that plane.

For an average point λ_i^{-k}, the operator **C** constructs the least-squares plane from the point positions $(\mathbf{x}_{i-(i \bmod 4)}^{-k}, \ldots, \mathbf{x}_{i-(i \bmod 4)+3}^{-k})$ which are part of a neighborhood relation. The plane minimizes the squared distances to these four points and is therefore an approximation to the local geometry of the model. It is computed by a principal component analysis (PCA) over those points which delivers an orthonormal coordinate system $(\mathbf{n}, \mathbf{u}, \mathbf{v})$ describing the principal directions of the covariance ellipsoid of the point positions [PGK02]. This system is aligned to the least-squares plane of which **n** represents its normal vector. As depicted in Figure 6, we translate the origin of that system to $\mathbf{x}_i^{-k}$ and represent all points by cylindrical coordinates (r, θ, z).

The operator **P** then predicts the coordinates (r, θ, z) of

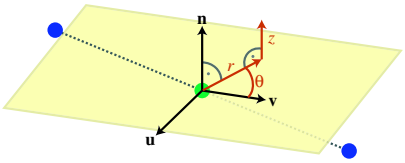
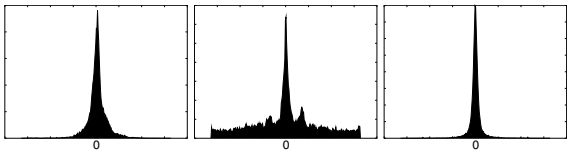


Figure 6: *The reference frame for the position detail coefficients.*

$\mathbf{x}_{2i}^{-k+1}$. Assuming that the least-squares plane is a close approximation of the geometry, we can predict the elevation z of the point over the surface as $z = 0$. Under the condition of an approximately regular sampling of the model, we can also make a prediction for r: As the average point density drops by a factor of $\sqrt{2}$ with each lower resolution step, r can be predicted from the density of the four average points divided by $2\sqrt{2}$. The density of the average points itself is estimated by their average distance. Finally, we also predict $\theta = 0$. The prediction error for θ then lies in the interval $[0, 2\pi]$. It can be further constrained to $[0, \pi]$ by swapping the points λ_{2i}^{-k+1} and λ_{2i+1}^{-k+1} along with their associated subtrees in the multiresolution hierarchy in the case of an error greater than π.



(a) Detail of r. (b) Detail of θ. (c) Detail of z.

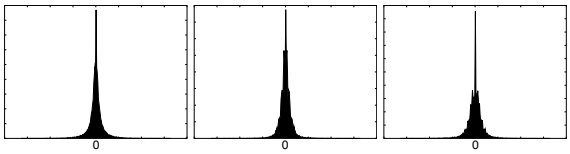
Figure 7: *Histograms of the position detail coefficients.*

Figure 7 shows histograms of the detail coefficients from the Chameleon model over all resolution levels. The peaks at zero prove the suitability of our predictions. It also shows an accumulation of the angular component around zero which is due to the alignment of our reference frame to the covariance ellipsoid.

5.2. Colors

All colors are first transformed from RGB into the global YUV space by the operator **C**. This representation is more closely related to the human perception as it splits the color information into a luminance part Y and a chromaticity U and V. Because humans are more sensitive for the luminance than the chromaticity, we spend more bits for Y than for U respectively V during quantization.

Assuming that neighboring points of the model have similar colors, the prediction operator **P** is zero. Hence, we store in the detail coefficients for each contraction pair the difference between the colors of one higher-resolution point and the average point. Histograms of those coefficients are depicted in Figure 8.

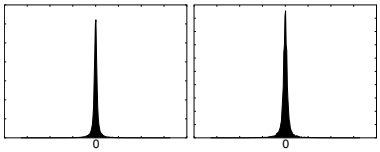


(a) Detail of Y. (b) Detail of U. (c) Detail of V.

Figure 8: *Histograms of the color detail coefficients.*

5.3. Normals

For each contraction pair we store the angle between the average normal and one of the higher-resolution normals in spherical coordinates. Hence, the prediction **P** is zero for both θ and ϕ. The coordinate transform **C** needs a local reference frame to align the spherical coordinate system. We use the average normal as the polar reference direction. To obtain an azimuthal reference vector, we project the vector with the biggest eigenvalue delivered by the PCA from the positions coder onto the plane perpendicular to the average normal. Figure 9 shows histograms of the detail coefficients.



(a) Detail of θ. (b) Detail of ϕ.

Figure 9: *Histograms of the normal detail coefficients.*

6. Encoding the Detail Coefficients

The detail coefficients are eventually compressed by a zerotree coder [Sha93] which is specialized on the coding of multiresolution coefficients and is able to encode them very efficiently. It exploits similarities between coefficients in subtrees of the multiresolution hierarchy. The output of the zerotree coder is a stream of symbols which is further compressed by arithmetic coding. We currently use the algorithm provided in [MNW98]

Both zerotree and arithmetic coder behave progressively. A partial arithmetic decoding of the compressed data stream

delivers a prefix of the zerotree stream after arithmetic decoding. The zerotree decoder then produces a set of inexact coefficients which gets refined as more data is delivered to the decoder.

The conventional zerotree algorithm however successively refines the coefficients of all resolutions simultaneously during progressive decoding. This behavior does not blend well with our quantization error propagation because our coder assumes an ascertained accuracy of lower-resolution coefficients. Introducing a greater error after compression leads to instabilities in the least-squares planes and thus produces an inferior quality of decompressed point positions. Therefore we use the conventional zerotree coder only for fixed rate compression.

If real progressive decompression is desired, we suggest a modification of the zerotree encoder which consists in reordering its output stream in such a way that, during later decompression, the detail coefficients get refined successively from low to high resolution. The conventional zerotree coder does the encoding in several passes. In each pass, an additional bit for each significant coefficient is coded, successively gaining more accuracy. Within each pass, the data is processed from the low resolution coefficients up to the high resolution. Our modification consists in swapping those two orderings. Thus, we first arrange the data from low to high resolution coefficients and then do the ordering according to the bit significance. During progressive decoding of that stream, the high resolution coefficients all start with a value of zero and they will not be refined until all lower resolution coefficients are fully decoded. This manifests itself in increasing the resolution of the decoded model. In the following section, we provide experimental results for both approaches.

7. Results

We evaluate our coders with various point models of different sizes. The compression performance is quantified for each attribute by the average number of bits per point (bpp) in relation to the loss of quality, measured by the peak signal to noise ratio (PSNR).

In the field of mesh compression, the geometric error is usually calculated from the distance between the compressed and uncompressed mesh surfaces. A similar approach for point set surfaces is the comparison of the corresponding MLS surfaces [PGK02] which is also used by Fleishman et al. [FCOAS03]. However, this metric is unable to measure the sampling of the model, which is a crucial criterion for high-quality point based rendering. In contrary to triangle meshes, a bad sampling would lead to cracks in the surface due to the lack of connectivity information. Furthermore, the MLS surface tends to smooth the quantization errors such that the measured error is usually lower than disturbance in the visual quality.

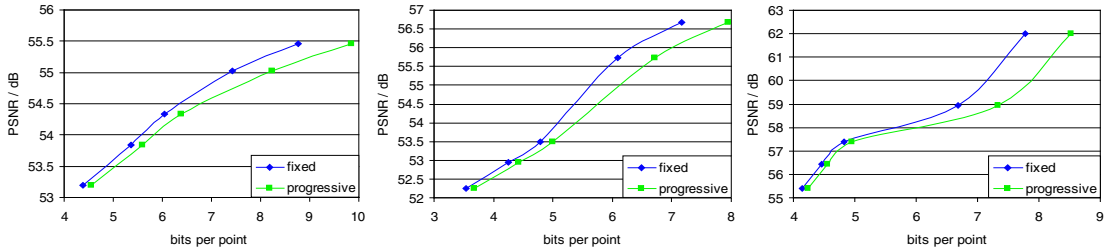
For these reasons we use the MLS metric for comparison with [FCOAS03] only. Whereas, in the main part of the following evaluation, we concentrate on measuring the quantization error directly between discrete pairs of corresponding samples from the original and compressed model. The PSNR for the position attribute is evaluated using the Euclidean distance between the points. The peak signal is given by the length of the diagonal of the bounding box. The error between the normals is calculated from the enclosed angle with a peak angle of 180 degrees. For the colors, we compute the PSNR on the scalar values of each channel Y, U and V separately.

We first compare our fixed rate compression with our progressive scheme, using various coarse quantizations for the different point attributes. Figure 10 shows rate-distortion curves from the position coder. The progressive coding performs between 2 and 10 percent worse than the fixed rate coding. This is due to the fact that the compression gain of the arithmetic coding is worse for the progressively reordered zerotree stream than for the conventional stream. The behavior of the other pipeline stages is identical in both cases. Compared to state of the art mesh compression, the PSNRs given here are lower since they also measure the quality of the sampling. As shown by the images of Figure 13, the compression artifacts differ from those of mesh compression due to the lack of connectivity information. For low bit rates, the sampling gets more and more irregular. We compensate for these irregularities by recomputing the splat radii during decompression which can be done very efficiently using the neighborhood relations that are inherently stored in the point ordering.

We further give some preliminary results for our color and normal coders. The color compression is evaluated in Table 1. As can be seen in Figure 14, it shows some blurry artifacts which are typical for high compression of images. Table 2 shows rate-distortion values of our normal coder; sample images are given in Figure 15. In contrast to mesh compression, we are able to code the appearance of the model in the same way than the geometry, using the same data structures. Hence there is no need to store additional texture images and to associate and compress texture coordinates with each vertex.

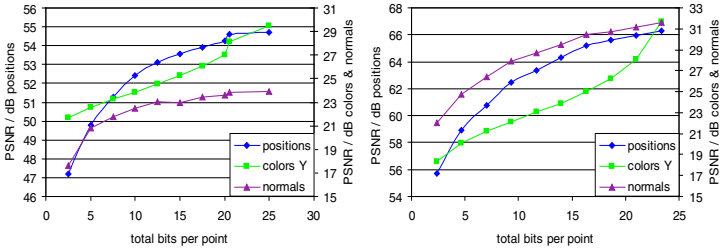
Its multiresolution character makes our compression approach very suitable for streaming and progressive decompression. Figure 11 shows rate-distortion curves for progressive decoding of different models. The amount of data is described in total bits per point. One fifth of the bits is used for color compression. The rest is spread evenly over the position and normal coding. As more and more data is decoded, the resolution of the model successively increases. Visual results of this process are provided in Figure 16.

Figure 12 shows a comparison of our progressive position coder with the one presented by Fleishman et al. [FCOAS03]. We use the error metric from their paper which



(a) Chameleon (102k points). (b) Venus (134k points). (c) Dragon (436k points).

Figure 10: *Compression performance of position coder.*



(a) Chameleon (102k points). (b) Octopus (466kpoints).

Figure 11: *Rate-distortion curves for progressive decompression.*

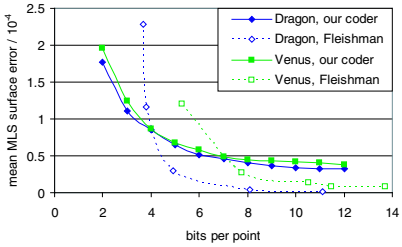


Figure 12: *Comparison of our progressive position coder with the method by Fleishman et al.*

	PSNR / dB			bits per point	
	Y	U	V	fixed	progr.
Chameleon (102k points)	29.5 24.4	39.5 38.1	38.8 37.3	2.5 0.9	2.7 1.0
Octopus (466k points)	26.9 21.9	33.7 32.2	31.0 28.7	2.0 0.8	2.1 0.9
Face (41k points)	35.5 31.9	47.4 45.1	42.3 39.7	1.7 0.8	1.8 0.8

Table 1: *Compression performance of color coder.*

	PSNR / dB	bits per point	
		fixed	progr.
Chameleon (102k points)	25.8 24.8	10.5 5.8	13.1 6.9
Venus (134k points)	40.0 36.7	13.5 7.9	17.0 9.4
Dragon (436k points)	31.1 27.8	10.5 5.6	12.7 6.4

Table 2: *Compression performance of normal coder.*

measures the mean distance between the MLS surfaces of the original and compressed models. While Fleishman's coder introduces less error for high bit rates we are able to achieve superior results for lower rates of about 4 bits per point.

8. Conclusion and Future Work

In this paper we provide a framework for compression of point-sampled models. In contrary to mesh compression we are able to code not only the geometry but arbitrary appearance attributes associated with the point samples in a unified way. The lack of connectivity information further helps to save storage space.

Our method is based on multiresolution predictive coding which has the capability of exploiting similarities between

neighboring points. By employing a graph matching algorithm we are able to find neighborhood relations that give us a maximum correlation in all attributes. The multiresolution approach naturally leads to a compression scheme that allows for progressive decoding.

In particular we present a coder for the model geometry. By transforming the point positions into a local reference frame, we exploit the fact that all the points are living on a surface. We have shown by a set of various examples that this approach works quite efficiently. We complete our framework by also suggesting methods for compression of colors and normals.

Nonetheless, there are still some issues of future work: First we aim to further improve the compression performance, especially of the color and normal coders. A main task here is a thorough examination of the effects that different coding approaches have on the visual quality of the model. The visual quality of the decompressed model can be further improved by supporting elliptical splats in object space which achieve a better coverage of an irregular sampled surface than circular disks. Another issue is speeding up the matching process that currently has a relatively high time complexity of about $O(n^2 \log n)$ for a search over n points. This could be greatly accelerated by using an approximative matching method.

References

[ADG*03] ALEXA M., DACHSBACHER C., GROSS M., PAULY M., VAN BAAR J., ZWICKER M.: Point-based computer graphics. *Eurographics Tutorial T1* (2003). 1

[BWK02] BOTSCH M., WIRATANAYA A., KOBBELT L.: Efficient high quality rendering of point sampled geometry. In *Proc. Eurographics workshop on Rendering* (2002), pp. 53–64. 2

[CR99] COOK W., ROHE A.: Computing minimum-weight perfect matchings. *INFORMS J. Comput. 11*, 2 (1999), 138–148. 3

[DG00] DEVILLERS O., GANDOIN P.-M.: Geometric compression for interactive transmission. In *Proc. VIS '00* (2000), pp. 319–326. 2

[Edm65] EDMONDS J.: Paths, trees and flowers. *Canad. J. Math. 17*, 3 (1965), 449–467. 3

[FCOAS03] FLEISHMAN S., COHEN-OR D., ALEXA M., SILVA C. T.: Progressive point set surfaces. *ACM TOG 22*, 4 (Oct. 2003), 997–1011. 2, 6

[GGH02] GU X., GORTLER S. J., HOPPE H.: Geometry images. In *Proc. SIGGRAPH '02* (2002), pp. 355–361. 2

[Hop96] HOPPE H.: Progressive meshes. In *Proc. SIGGRAPH '96* (1996), pp. 99–108. 2

[KG04] KHODAKOVSKY A., GUSKOV I.: Compression of normal meshes. In *Geometric Modeling for Scientific Visualization*. Springer Verlag, 2004, pp. 189–206. 2

[KSS00] KHODAKOVSKY A., SCHRÖDER P., SWELDENS W.: Progressive geometry compression. In *Proc. SIGGRAPH '00* (2000), pp. 271–278. 2

[LP86] LOVASZ L., PLUMMER M. D.: *Matching Theory*. Elsevier Science Ltd, 1986. 3

[MNW98] MOFFAT A., NEAL R. M., WITTEN I. H.: Arithmetic coding revisited. *ACM Trans. Inf. Sys. 16*, 3 (Mar. 1998), 202–211. (Proc. Data Compression Conf.). 5

[PGK02] PAULY M., GROSS M., KOBBELT L.: Efficient simplification of point-sampled surfaces. In *Proc. VIS '02* (2002), pp. 163–170. 4, 6

[RL00] RUSINKIEWICZ S., LEVOY M.: Qsplat: a multiresolution point rendering system for large meshes. In *Proc. SIGGRAPH '00* (2000), pp. 343–352. 2

[Ros99] ROSSIGNAC J.: Edgebreaker: Connectivity compression for triangle meshes. *IEEE Trans. Vis. and Comput. Graphics 5*, 1 (Jan. 1999), 47–61. 2

[Sha93] SHAPIRO J. M.: Embedded image coding using zerotrees of wavelet coefficients. *IEEE Trans. Image Proc. 31*, 12 (Dec. 1993), 3445–3462. 5

[Swe95] SWELDENS W.: The lifting scheme: A new philosophy in biorthogonal wavelet constructions. In *Wavelet Applications in Signal and Image Processing III* (1995), vol. 2569, pp. 68–79. (Proc. SPIE). 3, 4

[TG98] TOUMA C., GOTSMAN C.: Triangle mesh compression. In *Proc. Graphics Interface* (1998), pp. 26–34. 2

[TGHL98] TAUBIN G., GUÉZIEC A., HORN W., LAZARUS F.: Progressive forest split compression. In *Proc. SIGGRAPH '98* (1998), pp. 123–132. 2

[WLSG03] WÜRMLIN S., LAMBORAY E., STAADT O., GROSS M.: 3d video recorder: A system for recording and playing free-viewpoint video. *Computer Graphics Forum 22*, 2 (2003), 181–193. 2

[ZPKG02] ZWICKER M., PAULY M., KNOLL O., GROSS M.: Pointshop 3d: An interactive system for point-based surface editing. *ACM TOG 21*, 3 (July 2002), 322–329. (Proc. SIGGRAPH '02). 1

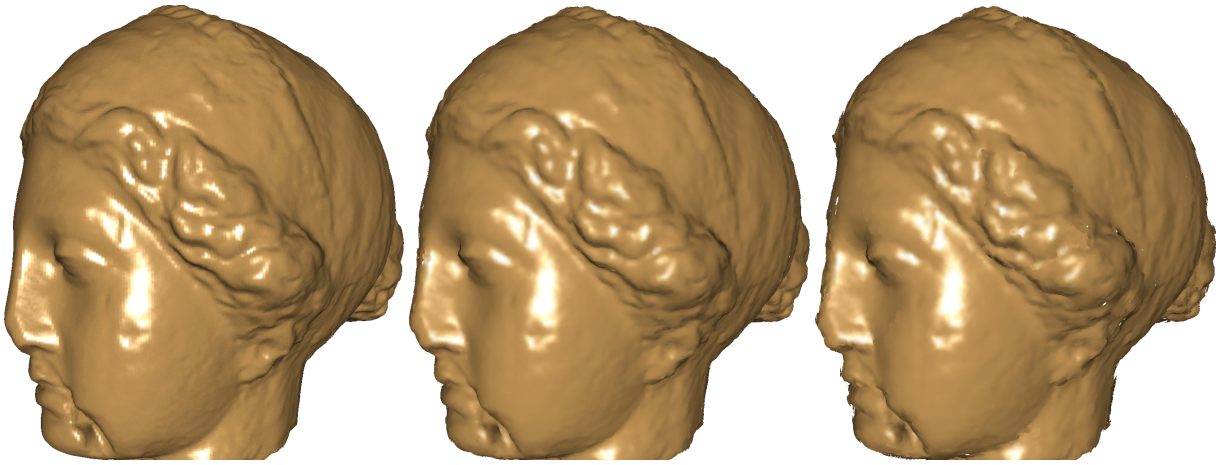


Figure 13: *Comparison of position compression (uncompressed, 8.8 bpp, 4.4 bpp).*

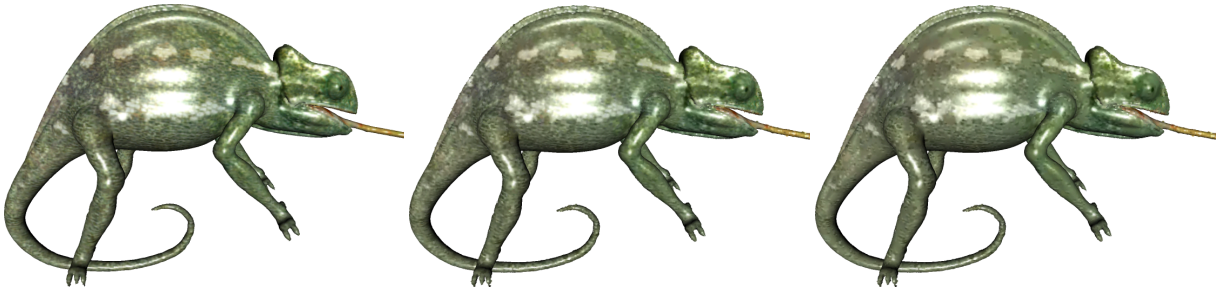


Figure 14: *Comparison of color compression (uncompressed, 2.5 bpp, 1.0 bpp).*

Figure 15: *Comparison of normal compression (uncompressed, 12.7 bpp, 6.4 bpp).*

Figure 16: *Progressive decompression with total bit rates of 7, 14 and 23 bits per point.*