Point set compression through BSP quantization

Alex Bordignon, Thomas Lewiner, Hélio Lopes, Geovan Tavares and Rener Castro
PUC–Rio — Departamento de Matemática — Matmídia Project — Rio de Janeiro — Brazil
{alexlaier,tomlew,lopes,tavares,rener}@mat.puc-rio.br

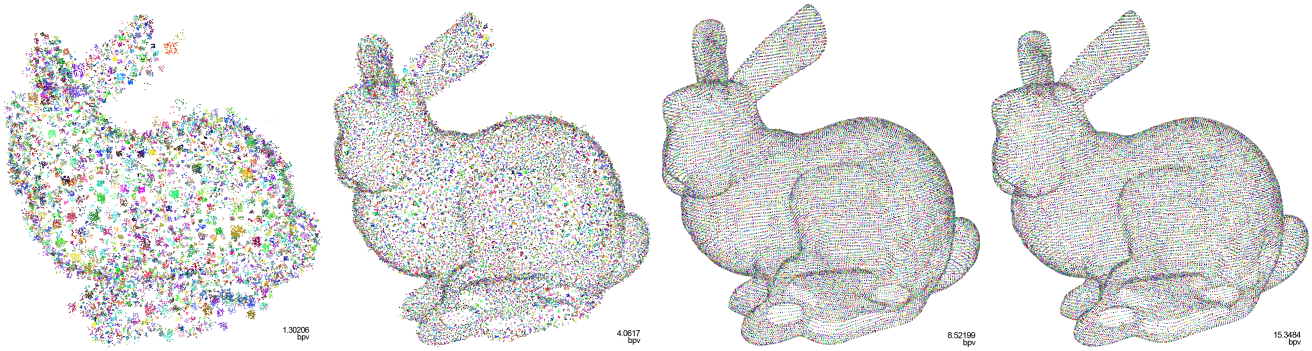


Figure 1. Compressing the original bunny point set (34834 points), using 1.3, 4.1, 8.5 and 15.3 bits per vertex: points in the same BSP cell have the same color.

Abstract

This work introduces a new compression scheme for point sets. This scheme relies on an adaptive binary space partition (BSP) which takes into account the geometric structure of the point set. This choice introduces geometrical rather than combinatorial information in the compression scheme. In order to effectively improve the final compression ratio, this partition is encoded in a progressive manner, decreasing the number of bits used for the quantisation at each subdivision. This strategy distributes the extra cost of the geometry encoding onto the maximal number of points, compressing in average 15% more than previous techniques.

Keywords: *Point Sets, Compression, Binary Space Partition, Geometry-Driven Compression, Geometry Processing.*

1 Introduction

Geometry processing usually relies on pointwise representation of the geometry, either through the vertices of a mesh or directly on meshless models. In the context of collaborative or published work, these representations need to be compacted before being transmitted. On one side, meshless models, such as raw scans, statistical simulation or particle interaction models, must be compressed directly as a point set. On the other side, meshes, built from mathematical tools or reconstructed from these point sets, can be compressed by various means: connectivity–driven compression schemes deduce the geometry representation from previously transmitted elements, whereas the more recent area of geometry–driven compression start with compressing the geometry. Direct compression of the point set can thus improve both mesh and meshless models compression.

This work proposes a new compression scheme for point sets. As opposed to previous point set compression schemes, which rely on regular tree decomposition, it uses a carefully designed BSP decomposition to adapt better to the coherency of the input data. For example, if the point set has been measured on a real surface by a scanning technique, these points contains sequences of points on parallel planes. If these planes are not parallel to the regular tree decomposition, previous methods do not extract this redundancy, and thus result in poorer compression ratios.

2 Related works

Among the point sets compression strategies, we can distinguish the methods inspired by mesh compression, mesh

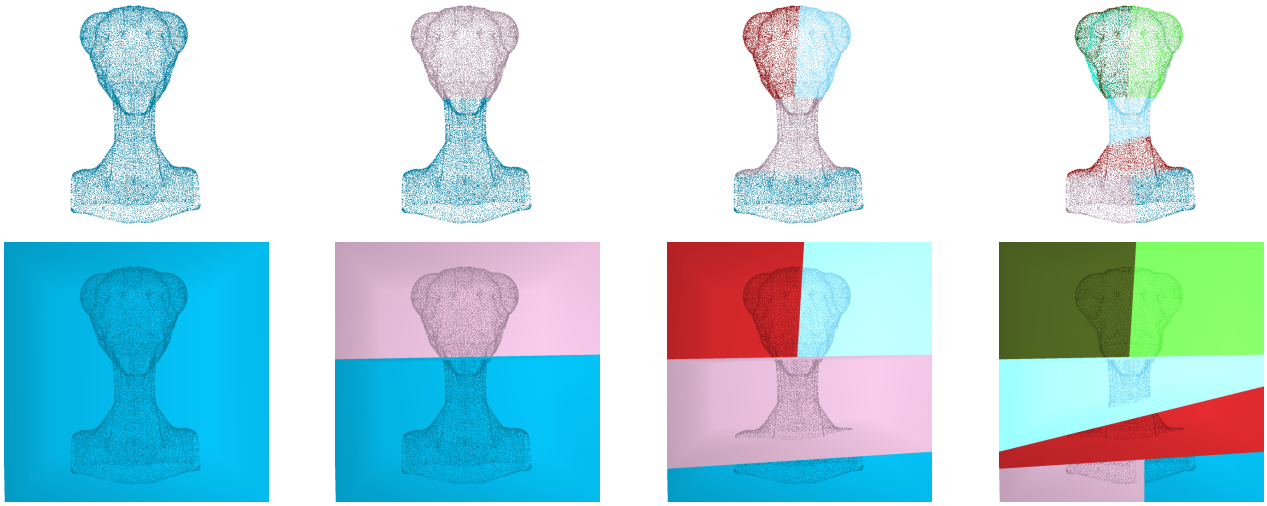


Figure 2. Our (quantised) adaptive subdivision captures the structure of the point set.

simplification and spatial subdivision. Although algorithms of the last category were introduced first, they still provide, until now, the best compression results for progressive and lossless compression.

Local mesh encoding. Since the early works of Deering [3] and Taubin and Rossignac [4], mesh compression reached impressive results. In particular, compressing the local adjacency of the mesh triangles became inexpensive, and it helps in compressing the position of the point through simple local predictors. This idea grounds the works of Gumhod *et al.* [9] and Merry *et al.* [5], who compute local spanning trees of the points optimizing the prediction error. The extra cost of the spanning tree is partly compensated by the efficiency of these predictors.

Local simplifications. As opposed to the previous methods which compress exactly the point set, several techniques reduce the complexity of the point set by recursively removing points from the original point set. These methods retrieve point sets either at a single reduced resolution, such as the scheme of Waschbüsch *et al.* [2], or in multiresolution, such as the one of Ochotta and Saupe [14]. Further work of Krüger *et al.* [16] optimises the simplification process to allow compact region and hardware implementation of this technique.

Lossy space subdivision. The above techniques use the local structure of the point set to predict or remove the next vertex to encode. Other strategies compress the whole point set globally, by subdividing it recursively. The work of Chen *et al.* [1], and the various variations of Peng and Kuo [7, 8, 6] use a lossy octree subdivision scheme which orders the subdivision by their importance. This scheme provides quick and inexpensive decompressions of medium quality, but waste many bits in recovering the original quality of usual point sets.

Progressive space subdivision. The lossy subdivision techniques actually intended to optimize the rate/distortion ratio of progressive octree encodings. These schemes subdivide an octree until separating each node, and then encode the non–empty cells of following the octree hierarchy. The number of points contained in each cell of the octree is encoded explicitly in the work of Gandoin and Deviller [10], while only the emptiness of each cell is transmitted in the work of Botsch *et al.* [15]. A synthesis of those two methods has been proposed in Lewiner *et al.* [11] work, as the first part of a geometry–driven mesh compression.

3 Contributions and overview

This work introduces an efficient compression scheme for point sets, which relies on a binary space partition (BSP) adapted to the geometric structure of the point set:

Geometry–aware BSP. Although the space subdivision techniques provide the best compression ratio, they do not use the geometric structure of the point set. We propose here to set the cut planes of each cell of the BSP orthogonal to the principal direction of the point contained in that cell. This strategy reflects correctly the structure of the point set, as shown on Figure 2. Moreover, this choice will introduce geometrical rather than combinatorial information in the point set compression.

Distributed cost. During the compression, this direction must be encoded in order to transmit the space subdivision, generating extra bits. In order to distribute this extra cost onto the maximal number of points, we use a strategy similar to Gandoin and Deviller's work [10], recalled at Section 4. They compress explicitly the number of points inside each cell of the octree, which wastes more bits close to the root of the octree (it contains all the points) and only one bit close to the leaves (it contains at most one point). The main observation is that, compared to Botsch *et al.* [15], this strategy performs much better, although [15] compresses only a binary information (emptyness of a cell). We therefore oriented our scheme towards Gandoin and Deviller's work.

BSP quantisation. In order to factor out the extra cost of the geometry, we need to send more information at for the BSP cells close to the root, and less for the one closer to the leaves. To do so, we propose an adaptive quantisation of the cell subdivision planes. With no bit transmitted, this quantisation reduces to a regular octree subdivision. Each additional bit transmitted then encodes information on the local statistical deviation of the point set.

BSP encoding. We also use the number of points inside the cell to encode its emptiness, as in [10], as detailed in Section 5. Moreover, we shift the subdivision plane in order to divide in almost equally parts the set of points in the BSP cell. This strategy further reduces the cost of the emptiness codification, since it will be accurately predicted. All together, this technique improves compression ratios in average by 15%, as described in Section 6.

4 Space Partition Encoding

Among point set compression techniques, the most efficient ones rely on an encoding of a space partition. Aside from the efficiency advantage, these techniques naturally work for points in any dimension since space partition are formulated for generic dimensions. Moreover, the hierarchical nature of these partitions allows a progressive encoding: these procedures can traverse the space partition in a breadth–first manner, and each step traversal increase the resolution of the previous one.

Recursive compression. The general principle of these techniques relies on creating a space partition such that each leaf cell has its size below the numerical precision of the point set, typically 12 bits per coordinate, and contains only one point of the set. The compression of the point set then reduces to the encoding of the space partition, which is essentially a fixed valence tree structure with the geometry of each cell stored at each node of the tree. The encoding of this tree is performed through a recursive traversal, which emits a *subdivision symbol* at each step. The traversal stops only when it reached a cell of size below the numerical precision of the point set.

Subdivision symbols. A subdivision symbol corresponds to the subdivision of the current cell in subcells. Those symbols encode at the same time the geometry of the subdivision and the repartition of the points inside the subcells. For some space subdivision schemes such as octrees and 2^d trees, the geometry of the subdivision can be automatically deduced without specific symbol. In that case, the only information to encode is the repartition of the points during the subdivision, and on that point distinguishes the three closest previous works [15, 10, 11].

Examples The strategy of [15] encodes for each of the eight subcells whether they are empty or not. This generates an 8-bits symbol which can be nicely predicted. However, it wastes 8 bits even when there is only one point in the cell.

The technique of [10] works with binary trees, and encodes the number of points contained in one of the two subcells, deducing the number contained in the other one by difference. This technique has the advantage of encoding only 1 bit per coordinate when there is only one point left in the cell, but has a higher cost for the first nodes.

The method of [11] encodes for each subcell whether it is empty, has a single point or contains more points. This method improves on [15, 10] for small and medium size point sets, since it wastes few bits for the first nodes and maintain 1 bit per coordinate when there is only one point in the cell. However, our proposal extends [10] rather than [11].

	Partition	Subdivision symbol	
		Geometry	Combinatory
[15]	octree	none	0/+ nodes ($\times 8$)
[10]	2^d tree	none	number of nodes
[11]	2^d tree	none	0,1 or + nodes
Ours	BSP	cut plane	number of nodes

Table 1. Principle of our proposal compared to the closest previous works.

5 BSP Compression

In computer graphics, point sets usually represent the *geometry* of an object. However, the above methods rely on encoding the *combinatory* of the point set in their subdivision symbols, which lack of geometric information. We

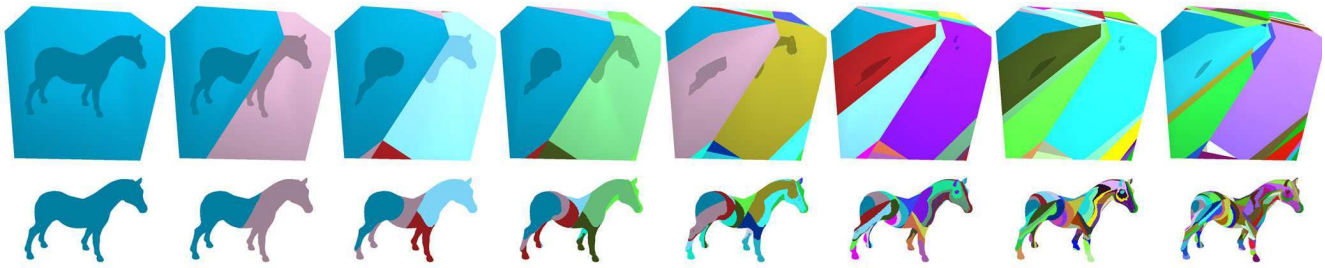


Figure 3. Our BSP is constructed in a top–down manner, subdividing according to the local structure of the point set.

propose to encode information specific to the geometry of the point set within the subdivision symbols (see Table 1). The BSP framework provides the basic structure for this goal, since cuts of the BSP can contain geometric information such as principal directions of the point set. Moreover, since octrees and 2^d trees are specific BSP, this framework extends the above methods.

The main difficulty with this idea consists in maintaining a small number of bits for each subdivision symbol. The clue is also the interpretation of the efficiency of [10]: on one hand, although [10] wastes many bits for the first nodes since it encodes the whole number of nodes instead of a $0/1/+$ symbol. On the other hand, this information is shared by all the points of the set, and when reaching lower nodes, the overhead of encoding the number of nodes shrinks. The balance is globally positive, as proved in [10].

We will use here a similar strategy: the geometric symbols of the cut are encoded with precision for the first nodes, and as we subdivide the BSP the precision of these symbols reduces, until not being encoded at all. At that point, the geometry of the subdivision is automatic, and corresponds to 2^d tree encoding. We can then adopt any of the above mentioned strategies for the combinatorial subdivision symbols. However, we chose to extend [10] since our analysis came from the study of their algorithm. Moreover, we will design the BSP in order to distribute almost equally the points between the subcells. The number of nodes of the subcell tends to be half of the number of nodes of the cell, improving the prediction mechanism.

5.1 BSP Construction

Cell geometry. We define our adaptive space decomposition by a BSP with planar cuts (see Figure 3). We choose conventionally a unit cube for the root cell. The construction of the BSP subdivides this cube into two convex polyhedrons, assigned to the two sons of the root in the BSP. For lossless compression, these polyhedrons are further subdivided until their assigned cell fit in a box of size 2^{-b}, where b is the numerical precision, in bits, of each coordinate of the point cell. In computer graphics applications, the usual data precision is $b = 12$ bits. For lossy compression, the cell may not be subdivided if it contains less that a few points.

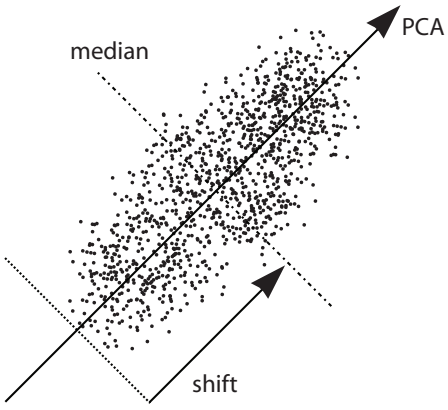


Figure 4. Cell planar cut from the principal component analysis and the median of the points

Principal Component Analysis. We define the cut plane direction using a common statistical technique, known as *principal component analysis* (PCA), similarly to [12]. The PCA is a least square minimization that extracts the main structure of a statistical data, here the coordinates of the points contained in a cell (see Figure 4): Given a cell and the set of points it contains, we compute the covariance matrix of their positions. The eigenvector corresponding to the greatest eigenvalue of this matrix corresponds to the closest line to the point, which is called the principal direction of the points. In order to best separate the points, we will chose the cut plane perpendicular to the principal direction.

Median shift. The PCA thus defines the direction of the cutting plane, and we can then adjust where to cut the cell

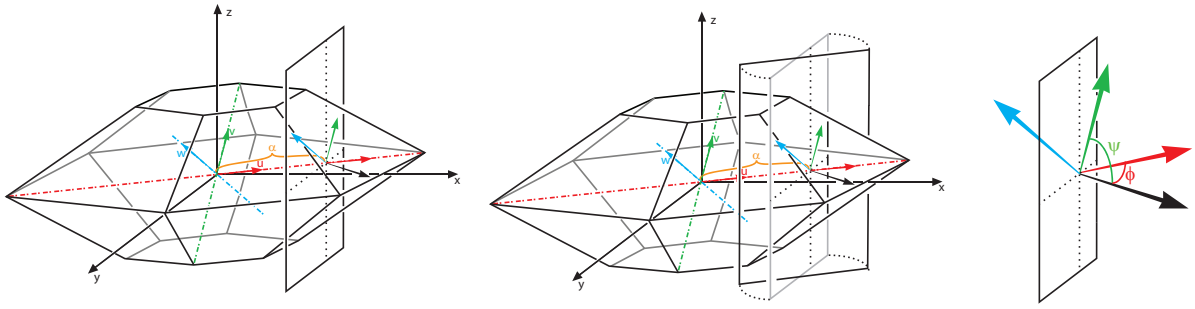


Figure 5. The quantisation parameters for the cut plane: the quantisation of the optimal plane (left) moves slightly the plane (middle) by clamping the parameters φ, ψ and α.

Algorithm 1 compress($points, cell$): encode the BSP recursively.

// compute the principal deviation of the points in the cell
$plane \leftarrow \mathsf{PCA}^{\perp}(points)$
$(\varphi, \psi) \leftarrow$ quantise $(plane)$
// shift the plane to separate the points into equal parts
$shift \leftarrow$ median $(points, \varphi, \psi)$
$\alpha \leftarrow$ quantise $(shift)$
// subdivide the cell
$(cell_{front}, cell_{back}) \leftarrow$ subdivide $(cell, \varphi, \psi, \alpha)$
// encode the subdivision and the number of points of the front cell
encode $(\varphi, \psi, \alpha, \#cell_{front})$
// recurse
compress $(cell_{front})$; compress $(cell_{back})$

along that direction. In order to predict efficiently the number of nodes, we will place the cut at the median of the points (see Figure 4). We compute this median by projecting all the points onto the PCA line, deduce the median of the resulting 1D data and shift the plane along the PCA line until it passes through the median.

5.2 Quantization parameters

Plane parameters φ and ψ. The cut plane is represented in the following local frame of the cell: the first axis $\vec{u}$ is the biggest diagonal of the cell (convex) polyhedron P, the second axis $\vec{v}$ is the biggest diagonal of the projection of P along $\vec{u}$, and the third axis $\vec{w}$ is the vector product $\vec{w} = \vec{u} \wedge \vec{v}$ (see Figure 5). In this frame, we express the normal $\vec{n}$ of the cut plane through the angle φ and ψ: $\cos(\varphi) = \vec{n}.\vec{u}$, $\cos(\psi) = \vec{n}.\vec{v}$. The coordinate of $\vec{n}$ along $\vec{w}$ can be deduced from the unity of $\vec{n}$: $\vec{n}.\vec{w} = \pm\sqrt{1 - \cos^2(\phi) - \cos^2(\psi)}$. Since the plane is defined without the normal orientation, we take as a convention that $\vec{n}.\vec{w} \geq 0$. This orientation determines whether a subcell is the front or the back one.

Shift parameter α. The shift to the median determines the constant coordinate of the plane equation. Again, we use the biggest diagonal PQ of the cell polyhedron to represent that shift: denoting by O the intersection of the plane with the diagonal PQ, we define α by: $\overrightarrow{PO} = \alpha\ \overrightarrow{PQ}$.

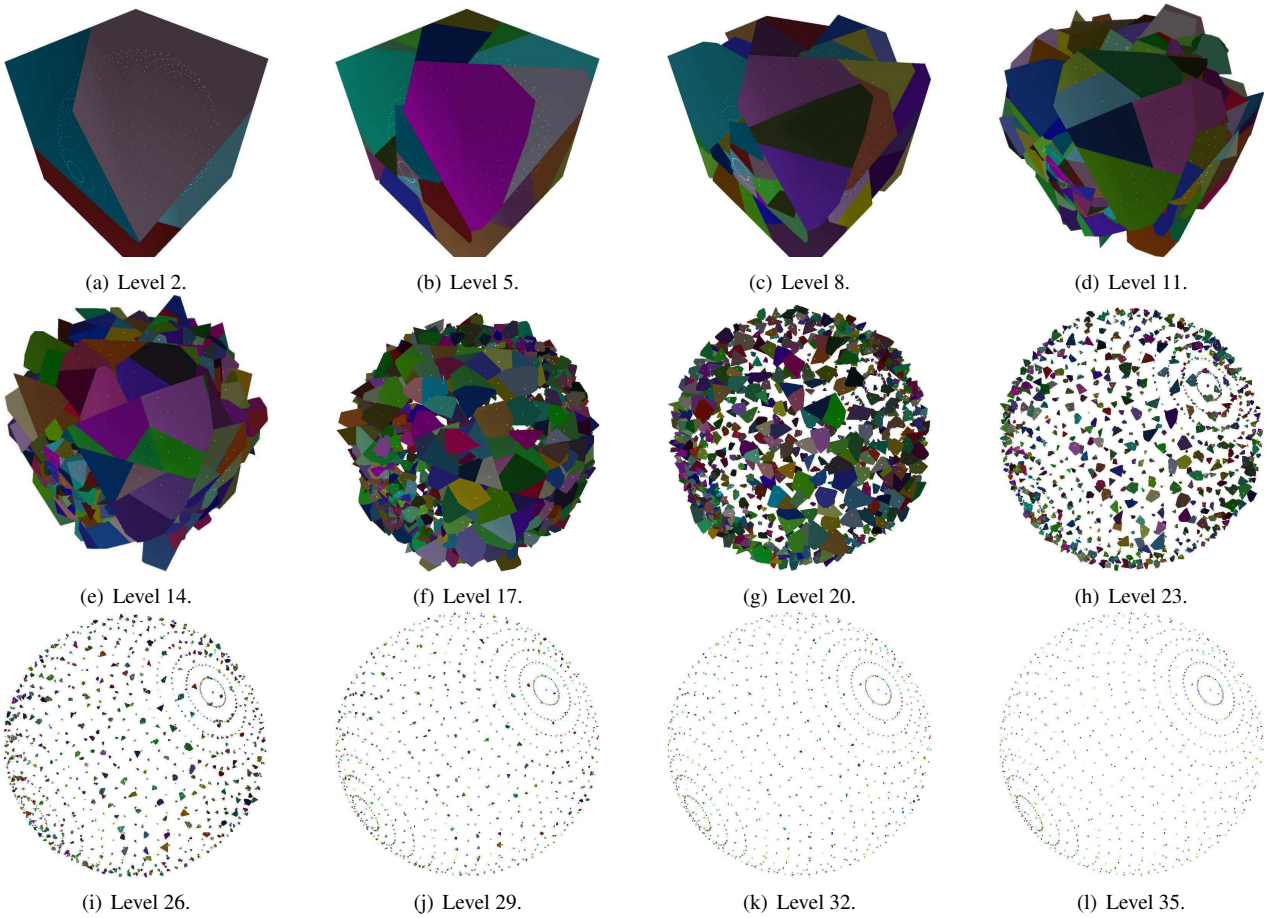
5.3 Quantization strategy

Bit allocation. Following the strategy deduced from [10], we will quantise the cut plane parameters with fewer bits at each subdivision. More precisely, the number of bits nb allocated for each parameter depends on the number $\#$ of points contained in the cell: $nb = \lfloor \log_2(\#) \rfloor - k$. We chose $k = 7$ for α, $k = 5$ for ψ, and $k = 3$ for φ. Moreover, in order to avoid degenerated cases, nb is set to zero if the cell has level below a given maximal value l_{max} (we chose $l_{max} = 10$ for our experiments).

Parameters quantisation. In order to quantise the cut plane, we first restrict the values of our parameters. α is generally a number between 0 and 1, although planes parallel to the diagonal may induce other values. Since this case has a low probability and may induce degenerated cell cut, we clamp α into the interval $[0.2, 0.8]$. We then map it to $[-1, 1]$ with an inverse quadratic function in order to have more quantised values close to $\alpha = \frac{1}{2}$.

The plane parameters are treated similarly: we encode $\sin(\varphi)$ by clamping it into $\left[-\frac{1}{2}, \frac{1}{2}\right]$. Then $\cos(\varphi)$ is clamped into $[-|\sin(\varphi)|, |\sin(\varphi)|]$ (since $\cos^2(\phi) + \cos^2(\psi) \leq \|\vec{n}\| = 1$). For both parameters we apply the same apply inverse quadratic function as α.

0 bit quantisation. When the number of bits allowed to encode the plane is zero, the above scheme cuts the cell perpendicularly to the diagonal, and at its middle point. This scheme is efficient in practice, but may not converge quickly, since the cut may not reduce significantly the volume of the cell. However, since we aim at encoding a con-



(a) Level 2. (b) Level 5. (c) Level 8. (d) Level 11.

(e) Level 14. (f) Level 17. (g) Level 20. (h) Level 23.

(i) Level 26. (j) Level 29. (k) Level 32. (l) Level 35.

Figure 6. The size of each cell of the BSP must converge quickly to zero.

stant precision for all the points of the set, we need to reduce quickly the size of the cells. We thus change the above scheme when a cell contains only one point of the set. In that case, we subdivide it as a regular octree, which is guaranteed to converge (see Figure 6).

Prediction. We finally encode these quantised parameters using a simple arithmetic coder [13], where the initial probability of the symbols is a Gaussian distribution. This suits particularly well since the plane parameters were designed to be close to 0 with high probability. Moreover, the median cut of the cells increase the probability for the number of points inside the subdivided cell to be half of the number of points inside the whole cell. We thus use the same Gaussian distribution for encoding the number of points in the front cell.

6 Experiments

We implemented the above method for points in $\mathbb{R}^3$ as described in the previous section, and resumed in Algorithm 1. We obtain compression ratio in average 15% lower than the previous methods (see Table 3). This validates the analysis presented here and puts the BSP compression for point set in front of octree based compression. Our method is relatively fast in execution (see Table 2) and very robust in geometric calculus.

#verts	Comp. ratio (bits/vert)	Comp. time (sec)	Decomp. time (sec)
$0-1000$	21.1975	0.17	0.99
$1000-5000$	19.1819	0.19	3.23
$5000-10000$	17.6372	0.21	6.50
$10000-20000$	16.0554	0.24	11.77
$20000-50000$	14.6821	0.32	20.62
>50000	14.5613	0.42	35.32

Table 2. Compression ratios and timings on a hundred classical models of Computer Graphics, ordered by size.

	#verts	[15]	[10]	[11]	Ours
head_camel	777	26.17	24.04	24.29	20.47
dinosaur	927	27.22	25.10	25.05	21.19
horse	927	27.67	25.47	25.31	21.39
pig_low	927	26.63	24.65	24.54	20.70
kangaroo	929	26.13	23.93	23.78	20.33
dolphin	979	25.15	23.11	23.00	19.59
ant	1147	23.52	22.03	22.00	18.54
pig	1843	25.64	23.61	23.50	20.07
triceratops	2832	22.08	20.66	20.68	17.30
ellipsoid	3820	23.93	22.62	22.51	19.10
pig_high	4999	22.92	21.88	21.81	18.51
chair	5095	23.77	22.87	22.58	19.14

Table 3. Comparison on classical Computer Graphics models, of our method with previous ones based on space partition encoding (the values are in bits per vertex for 12 bits of precision).

7 Conclusions

This work introduced a new scheme for progressive compression of point sets. It introduces geometry information in the subdivision scheme, and proposes an efficient method to distribute the extra cost of coding this information. Furthermore, this geometry information contributes to the point set compression in such a way that it ends up improving the final compression ratio.

The improvement on previous lossless compression methods for point sets advocates for the use of geometry–driven techniques for mesh compression, since the geometry compression alone tends to be sufficiently efficient. Moreover, our design of the adapted binary space partition turned out to be particularly efficient, and may serve for other applications.

Acknowledgments The authors are members of the Matmidia laboratory, which is sponsored by FINEP, PETROBRAS, CNPq, and FAPERJ. H. Lopes is partially supported by a CNPq and FAPERJ Primeiro Projeto research grants.

References

[1] D. Chen, Y.-J. Chiang, and N. Memon. Lossless Compression of Point-Based 3D Models. *Pacific Graphics*, pages 124–126, 2005.

[2] M. Waschbüsch, M. Gross, F. Eberhard, E. Lamboray, and S. Würmlin. Progressive compression of point-sampled models. In *Eurographics Symposium on Point Based Graphics*, pages 95–102, 2004.

[3] M. F. Deering. Geometry compression. In *Siggraph*, pages 13–20. ACM, 1995.

[4] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *Transactions on Graphics*, 17(2):84–115, 1998.

[5] B. Merry, P. Marais, and J. Gain. Compression of dense and regular point clouds. In *Computer graphics, virtual reality, visualisation and interaction in Africa*, pages 15–20, 2006.

[6] J. L. Peng and C.-C. J. Kuo. Geometry-guided progressive lossless 3D mesh coding with octree (OT) decomposition. In *Siggraph*, pages 609–616, 2005.

[7] J. L. Peng and C. J. Kuo. Octree-based progressive geometry encoder. In *SPIE Human Vision and Electronic Imaging*, pages 301–311, 2003.

[8] J. L. Peng and C. J. Kuo. Progressive geometry encoder using octree-based space partitioning. In *Multimedia and Expo*, pages 1–4. IEEE, 2003.

[9] S. Gumhold, Z. Karni, M. Isenburg, and H.-P. Seidel. Predictive point-cloud compression, 2004. Siggraph Sketch.

[10] P.-M. Gandoin and O. Devillers. Progressive lossless compression of arbitrary simplicial complexes. In *Siggraph*, volume 21, pages 372–379. ACM, 2002.

[11] T. Lewiner, M. Craizer, H. Lopes, S. Pesco, L. Velho, and E. Medeiros. GEncode: geometry–driven compression in arbitrary dimension and co–dimension. In *Sibgrapi*, pages 249–256, Natal, Oct. 2005. IEEE.

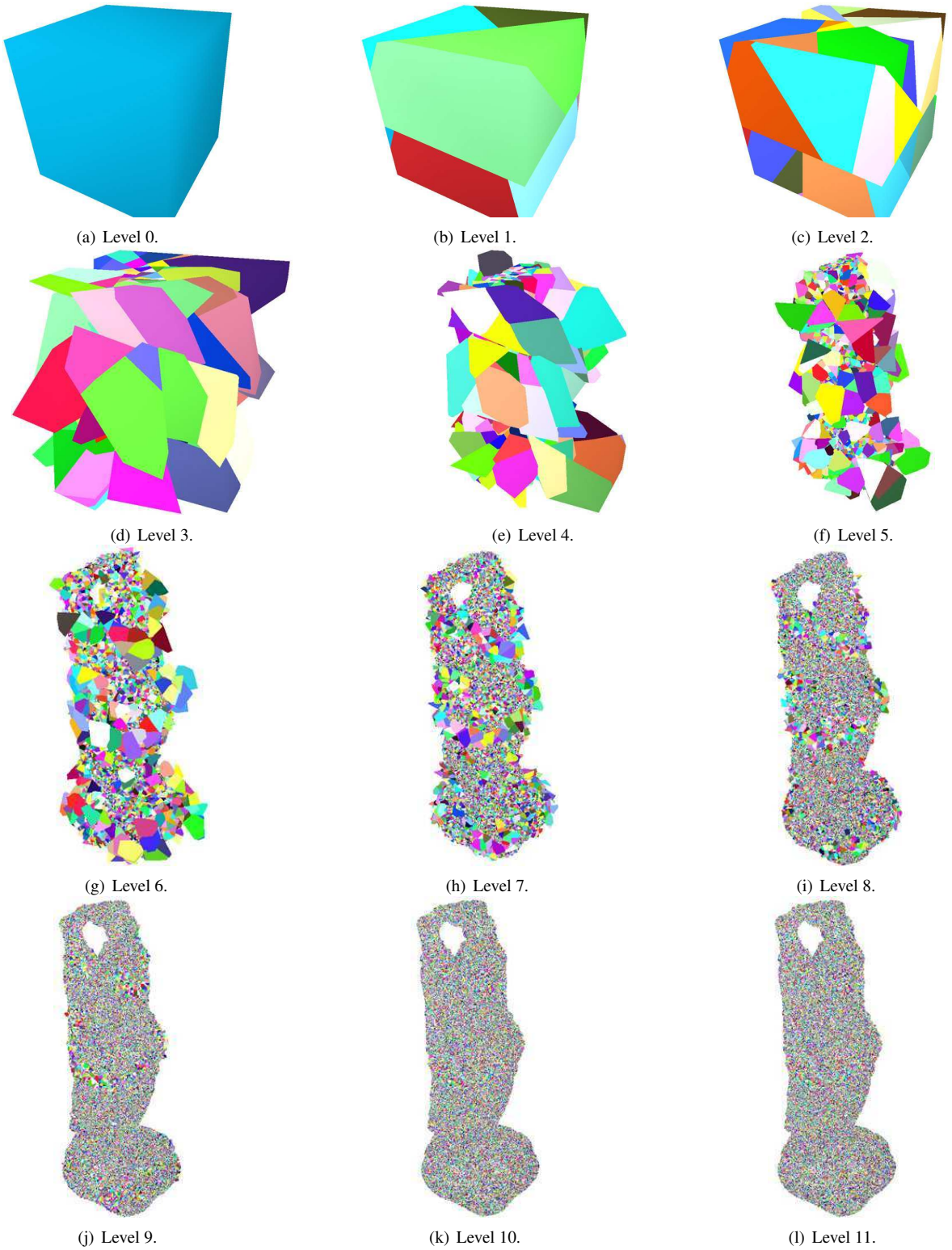
[12] E. Shaffer and M. Garland. Efficient adaptive simplification of massive meshes. In *Visualization*, pages 127–134. IEEE, 2001.

[13] J. Rissanen. Generalized Kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, 20:198–203, 1976.

[14] T. Ochotta and D. Saupe. Compression of point-based 3D models by shape-adaptive wavelet coding of multi-height fields. In *Eurographics Symposium on Point–Based Graphics*, pages 103–112, 2004.

[15] M. Botsch, A. Wiratanaya, and L. Kobbelt. Efficient high quality rendering of point sampled geometry. In *Eurographics Workshop on Rendering*, pages 53–64, 2002.

[16] J. Krüger, J. Schneider, and R. Westermann. DuoDecim - A Structure for Point Scan Compression and Rendering. In *Eurographics Symposium on Point-Based Graphics*, pages 99–107, 2005.



(a) Level 0. (b) Level 1. (c) Level 2.

(d) Level 3. (e) Level 4. (f) Level 5.

(g) Level 6. (h) Level 7. (i) Level 8.

(j) Level 9. (k) Level 10. (l) Level 11.

Figure 7. Adaptive compression of the point set of the happy Buddha model.