Journal of Korea Multimedia Society Vol. 13, No. 12, December 2010(pp. 1748-1759)

S-Octree: An Extension to Spherical Coordinates

Taejung Park[†], Sung-Ho Lee[††], Chang-Hun Kim[†††]

ABSTRACT

We extend the octree subdivision process from Cartesian coordinates to spherical coordinates to develop more efficient space-partitioning structure for surface models. As an application of the proposed structure, we apply the octree subdivision in spherical coordinates ("S-Octree") to geometry compression in progressive mesh coding. Most previous researches on geometry-driven progressive mesh compression are devoted to improve predictability of geometry information. Unlike this, we focus on the efficient information storage for the space-partitioning structure. By eliminating void space at initial stage and aligning the R axis for the important components in geometry information, the S-Octree improves the efficiency in geometry information coding. Several meshes are tested in the progressive mesh coding based on the S-Octree and the results for performance parameters are presented.

Key words: adaptive grid, spherical coordinate, progressive geometry compression, octree

1. INTRODUCTION

In this study, we view the octree algorithm as an iterative procedure to reduce the uncertainty of geometry information. During the subdivision process, the octree subdivision keeps probing the boundaries (eg. surface) of the model using binary sectioning in each of the three axes. Based on this observation, we investigate the possible ways to optimize the conventional octree structure which is typically constructed in Cartesian coordinates for surface models. One major issues we focus on is that the problem domain the conventional octree handles is not always optimized for all the different models; at higher levels it should always probe the inside - near the center areas, in particular - of the surface model, where in most cases, do not include any useful information. A case that would require probing those areas is for tetrahedral meshes which also contain useful information near the center of them.

In this regard, we conclude that there exist some opportunities to optimize further in the conventional octree. To address this issue, we introduce "S-Octree", i.e. Spherical coordinate-based Octree. This is an extension from Cartesian coordinates to the spherical based on our idea to see the octree subdivision as an iterative procedure to reduce the uncertainty.

We estimate the efficiency of the S-Octree by applying it to mesh compression. To show the opportunities for optimization in storing geometry information, we apply the S-Octree to a progressive mesh compression, which is based on the conventional octree scheme [1]. We choose this progressive coder because it achieves one of the best results so far for progressive mesh compression.

The results show that the compression rates and distortions are improved for some of the tested models whose shapes are close to sphere surface

※ Corresponding Author : Chang-Hun Kim, Address : (136-713) Room 203, A-San Science Complex, Korea Univ. Anam Campus, Anam-dong 5-ga, Seongbuk-gu, Seoul, Korea, TEL : +82-2-3290-3574, FAX : +82-2-929-1917, E-mail : chkim@korea.ac.kr
Receipt date : Aug. 18, 2010, Revision date : Sep. 17, 2010
Approval date : Oct. 11, 2010
[†] CG Lab., Dept. of Computer Scinece, Korea Univ. (E-mail: taejung.park@gmail.com)
[††] CG Lab., Dept. of Computer Scinece, Korea Univ. (E-mail: pocorall@gmail.com)
[†††] CG Lab., Dept. of Computer Scinece, Korea Univ.
※ This research is supported by the Second Brain Korea 21 Project..

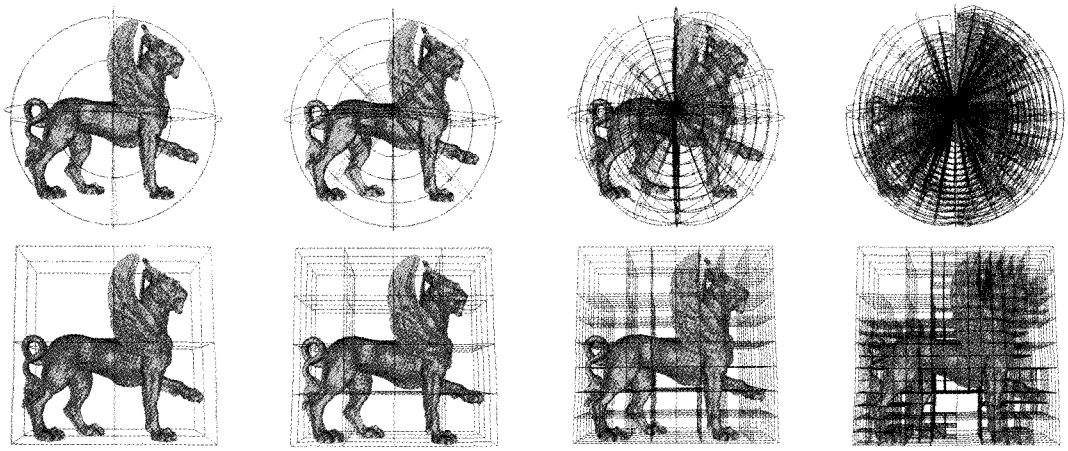


Fig. 1. Subdivision in the S-Octree (upper) and the C-Octree (lower).

with the S-Octree. For the other tested models, the compression rates and distortions are not highly degenerated and similar to those based on the conventional octree in Cartesian coordinates ("C-Octree"), though the applied progressive mesh coder is not fully optimized for the S-Octree. In later sections, detailed results are presented for several tested meshes.

1.1 Octree-Revisit

In a digital computer, it is impossible to represent a real number with infinite accuracy. Instead, we are only able to handle a real number as a range (or "interval") using given quantity of information (i.e. "bits"). In a way, we represent a discrete real number with n bits by bi-sectioning a continuous real number n times in a row, within the interval given. Thus, we can interpret the successive bi-sectioning procedures of given interval as a process to translate a continuous (i.e., with unlimited accuracy) value into a discrete (i.e. limited accuracy) one. In this process, it is obvious that the number of bits determines the accuracy within the given interval. The narrower the given interval is, the more accurate the value is, if the same number of bits are available. Or, when the interval is fixed, we can achieve higher accuracy with more bits. For example, in Figure 2, we need 4 bits (i.e. 1011) to describe the real number (the red triangle) with the accuracy shown in Step 4. However, if the initial interval in Step 1 were 1/2 of that illustrated, we would need only 3 bits (i.e. 011) to describe the real number with the same accuracy, saving one bit. Also, if we had more bits, we could describe the real number more accurately with longer bit sequence.

When we evolve this idea from 1D to 3D, the octree subdivisions can be considered as iteratively bi-sectioning the three intervals along the X, Y, and Z axes with given number of bits. In general,

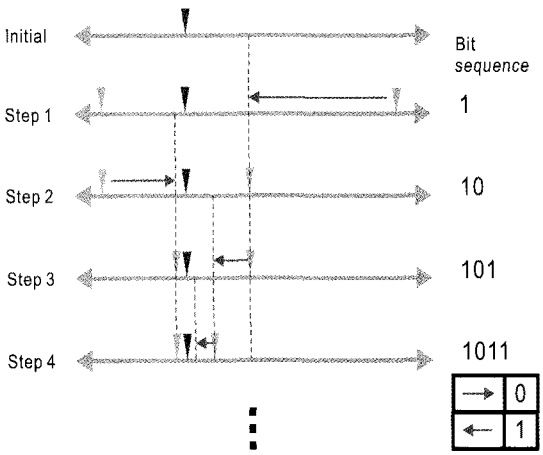


Fig. 2. Representing a 1D real number with 4 bits. The red triangle represents a real number with infinite accuracy (eg. 3.141526563···). The blue triangles show the finite intervals.

the octree subdivisions begin with a confined bounding box, which means we have three intervals for the three orthogonal axes, $[x_{min},x_{max}]$, $[y_{min},y_{max}]$, and $[z_{min},z_{max}]$. In this sense, we can see the octree subdivision process as a way to represent infinitely accurate geometry information in R^3using finite information (i.e. bits) in 3D space.

1.2 Spherical Coordinate

In computer graphics, we often handle surfaces, rather than solids. This fact casts doubts on us whether the conventional octree in Cartesian coordinates is truly optimized for surface models based on the idea discussed in the subsection 1.1. Because the conventional octree always probes the inner area of the surface, the accuracy of geometry information will have chances for improvement with a fixed amount of bits if we apply an adaptive space grid structure which effectively skips probing the unnecessary inner area.

Based on this idea, we extend the octree from Cartesian coordinates to spherical coordinates. In spherical coordinates, the bounding area is defined and the subdividing processes are performed in (R, Θ, ϕ), not (X, Y, Z), as shown in Figure 3. In particular, as represented in later sections, we note that the sphere surface models are described in this spherical octree extension ("S-Octree") with much less information (or, much more accurate or much less distorted if the given bits are the same). Such extreme efficiency for the spherical surfaces is achieved mainly because the range along R axis ($[R_{min},R_{max}]$) becomes extremely narrow. Note that the equivalent benefit is not quite expectable in the Cartesian octree ("C-Octree"). For example, you cannot represent a cuboid with very high accuracy in the C-Octree with such small number of bits, unless the cuboid is quite flat and thin and exactly aligned in one of the three surface normal directions of the XY, YZ, and ZX coordinate planes. Though the S-Octree is suitable for sphere surfaces or the surfaces which are topologically closer to the sphere surface, we have found that it also provides almost the same efficiency for progressive mesh compression in the C-Octree for other surfaces which are topologically different from the sphere surfaces. The reason is that, even if $R_{min}$is zero for such surfaces, the 3D intervals of the S-Octree become still close to those of the C-Octree.

1.3 Related Work

Lin et al. [2] have introduced the inverse polar octree which is constructed in the inverse spherical coordinate, i.e. (R^{-1},Θ,ϕ) for VR vision systems. In their application, the inverse polar octree converts the objects at a far distance (i.e. $R\rightarrow\infty$) into a finite and tangible range. As far as we know, this is the only case that the octree subdivision is applied practically to spherical coordinates. However, the

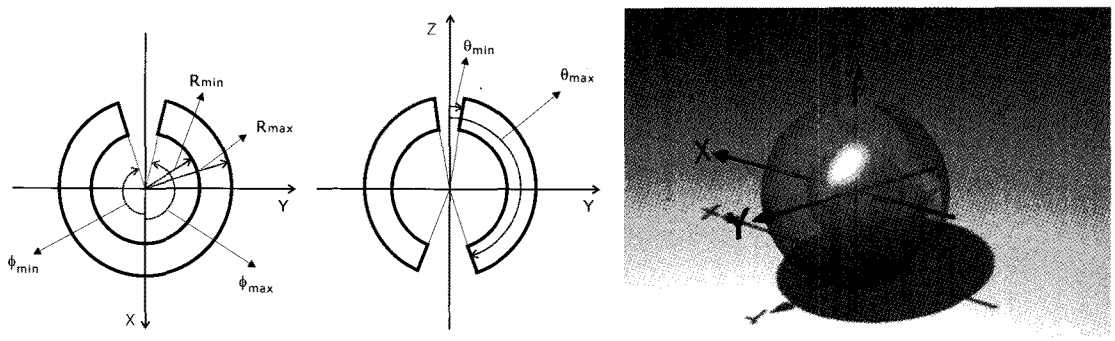


Fig. 3. Illustration of the root cell of S-Octree. All geometry elements are included in the illustrated space (the inside of blue solid lines in the upper two illustrations).

purpose and approach of [2] are quite different than those of our method. For example, the proposed S-Octree does not assume the inverse R axis. Also, the inverse polar octree takes full ranges for Θ and ϕ ($[0, \pi]$ and $[-\pi, \pi]$, respectively) but those ranges for the S-Octree are $[\varepsilon_0, \pi - \varepsilon_1]$ and $[-\pi + \varepsilon_2, \pi - \varepsilon_3]$. For meshes, small positive real numbers ε_i are determined based on the distribution of mesh vertices in spherical coordinates. Also, the inverse polar octree handles the whole environmental scene but the S-Octree takes individual surface models.

Gandoin et al. [3] have applied the kd-tree method to progressive mesh coding. This method is one of the first progressive mesh coders based on adaptive space-partitioning schemes. In general, those progressive mesh coders based on adaptive space-partitioning structures are called "geometry-driven coders". As the name implies, the geometry-driven coders code geometry information based on the partitioning structure and then they compress the connectivity information based on the coded geometry information. Unlike this, other mesh coders encode the connectivity information first, and geometry information is predicted based on the connectivity information (i.e. "connectivity-driven"). See [4]. Lee et al. [5] have utilized the octree subdivision to code isosurfaces using 15 neighbor sign (i.e. inside/outside) information as a context. In [1], the octree subdivision process is coded as sorted indices based on a sophisticated pseudo probability model. Park et al. [6] have presented a mesh coder based on [5] to improve geometry prediction. Huang et al. [7] use the octree algorithm to progressive point cloud coding. Those all studies use the octree defined in Cartesian coordinates, except [3] which uses the kd-tree in Cartesian coordinates. For those progressive coders, the main interest is given to the prediction methods based on the adaptive space-partitioning structure (i.e. the kd-tree or octree). Compared with this, the attention to the efficiency of the adaptive space-partitioning structure for information storage is relatively paid less. In this study, we analyze the ability to store information for the C-Octree and estimate the S-Octree as an alternative approach by applying the S-Octree to the algorithm presented in [1]. Usually, the visual quality (i.e. distortion) is measured by the sum of L2 norm of the distance between two models. The METRO tool [8] provides a convenient way to measure the metric for various meshes.

2. Transition to spherical coordinates

2.1 Optimization Opportunities in C-Octree

Though the conventional octree algorithm has been quite successful for years, we see that there are several opportunities for optimization. We summarize them as follows:

2.1.1 Inefficiency for Inner Area of Surface Models

The C-Octree subdivision consumes some bits to probe the inner area of surfaces because there is no mechanism to identify whether there is any geometrical information in the inner area. Practically, lots of surface models do not have meaningful information there. As illustrated in Fig 3, there are possible ways to save a few bits in this point, by reducing the initial interval for the C-Octree algorithm. In this regard, we see that the C-Octree wastes some information during subdividing at the cells at higher levels (i.e. closer to the root cell).

2.1.2 Capturing Geo Info in Surface Normal Directions

As discussed, we need to reduce the initial interval defined by the bounding box (or bounding area unless it is not defined in Cartesian coordinates) to save extra bits. However, simply reducing the initial interval does not always guarantee to improve the rate/distortion performance; from experiences, we have found that the rate/distortion metric is improved more by reducing the intervals

in the direction of surface normal vectors on the model surface. Before discussing the reason, we need to review the metric to measure the distortions. To measure the distortions between models, the normalized L^2 norm of the distance between them is measured, which is defined as follows:

$$d_{normalized}(X,Y) = \left(\frac{1}{S(X)} \int_{x \in X} d(x,Y)dx\right)^{1/2} / L_d \quad (1)$$

where $S(X)$ is surface area of model X and L_d is the diagonal length of the bounding box of the original model.

As discussed in [6], we observe that this metric does not vary much when the errors occur in the tangential directions of the surface but it is very sensitive to the errors occurred in the directions of the surface normal vectors. This makes sense considering that the errors in the direction of surface normal vectors generally lead to local volume change on the model which could easily be recognized. Unlike this, if the errors are in the tangential direction of the surface, little visual change would occur.

Thus, we need to assign more bits to the surface normal directions and less to the tangential to improve distortion with the same data resources. Or, we need to reduce the initial bounding area in the direction of surface normal directions with fixed number of bits. Also, we have to be able to get approximate normal vectors which are close to the actual surface normals. In [6] and [5], which are all based on the C-Octree, approximate surface normal vectors are calculated by applying the least square fitting method to the each leaf cell that has only one mesh vertex. After that, those two methods assigned more bits by applying finer quantization numbers to the estimated surface normal directions. Observed in those two studies, the C-Octree requires additional information, such as inside/outside information for each corner of leaf cells to estimate the surface normal vectors. S-Octree appears as shown in Figure 3.

2.2 Overview of S-Octree

As discussed so far, the conventional octree in the 3D Cartesian coordinate (C-Octree) begins by setting a bounding box. We can arrange the bounding box to align with the three axes of the 3D Cartesian coordinate (i.e. the axis aligned bounding box (AABB)). This bounding box makes the root cell of the octree. Also, this means we have three intervals for the three axes, $[x_{min}, x_{max}]$, $[y_{min}, y_{max}]$, and $[z_{min}, z_{max}]$, where $[a_{min}, a_{max}]$ indicates $a_{min} \le a \le a_{max}$. After that, each interval is subdivided into two sub-intervals making eight child cells. This subdivision process continues until the leaf cells have reached desired level or any specific requirement is met according to applications (for example, until every leaf cell has only one vertex).

Just like the C-Octree, the S-Octree starts construction by finding the bounding intervals along each axis, i.e. $[R_{min}, R_{max}]$, $[\Theta_{min}, \Theta_{max}]$, and $[\Phi_{min}, \Phi_{max}]$. In particular, R_{max} should be the radius of the smallest bounding ball around the geometry elements and R_{min} the largest radius with no geometry elements inside. After setting the tightest ranges, the initial bounding space defined with $[R_{min},R_{max}]$, $[\Theta_{min},\Theta_{max}]$, and $[\Phi_{min}, \Phi_{max}]$ is set to be the root of S-Octree. After then, the subdivision processes follow in the same way with those of the C-Octree.

When S-Octree is applied to meshes, due to the discrete nature of vertex positions, following relationships hold in general:

$$[\Theta_{min}, \Theta_{max}] = [\varepsilon_0, \pi - \varepsilon_1] \quad (2)$$

$$[\Phi_{min}, \Phi_{max}] = [-\pi + \varepsilon_2, \pi - \varepsilon_3] \quad (3)$$

where ε_i are small positive real numbers ($i = 0, 1, 2, 3$). Thus, unlike the cuboid bounding space defined in the C-Octree, the bounding space of S-Octree appears as shown in Figure 3.

2.3 Analysis of S-Octree

We discuss the properties of the S-Octree ac-

cording to the issues mentioned in subsection 2.1.

2.3.1 Inefficiency for Inner Area of Surface Models

The S-Octree avoids probing the inner area of surface by setting $[R_{min},R_{max}]$ as shown in Fig 2. We believe this attribute helps improve the efficiency for storing geometry information.

In the later sections, we discuss the efficiency achieved by applying this structural advantage to a progressive mesh coder proposed in [1]. Though this progressive mesh coder is not originally developed for the proposed S-Octree structure and thus not fully optimized for it, we find some improvements are possible in the S-Octree.

2.3.1 Capturing Geometric Infomation in Surface Normal Directions

As discussed in subsection 2.1, the geometry information in the directions of the surface normal vectors is important to improve the rate/distortion metrics. Except for the specific coder applications which explicitly handle normal vector compression, most of the progressive coders based on the C-Octree only handle connection and geometry information so that they hardly predict the surface normals. We find that some parts of surface normal vectors on a surface model are very close to those on a sphere surface. As a special case, for a spherical surface model, those normal vectors match exactly. Though it is not generally possible for all the surface normal vectors on an arbitrary surface model to be close to those on a spherical surface, we find the S-Octree assigns data resources more efficiently in the direction of R axis for many cases.

For the C-Octree, there exist some face normal vectors which lie in the directions of ±X, ±Y, and ±Z axes but the surface normal vectors only can be expressed as linear combinations of three unit vectors along X, Y, and Z axes. This makes it hard for the space-partitioning grids (i.e. octree, kd-tree, or even simple regular grids) to predict about the surface normal vectors without additional information.

We define a metric to check the closeness of surface normals between a model and the a_R unit vectors in the direction of R axis in spherical coordinates as follows:

$$\xi = (1-\cos(2\rho))/2 = \sin^2(\rho) \tag{3}$$

where ρ is the angle between the surface normal on the model and a_R. Figure 4 shows the graph for equation (4); this equation returns larger values (close to one) when the angle between the two normal vectors is near π/2. In this case, the area will be darker as shown in Figure 6. Figure 6 visualizes the results in a few mesh models based on the brightness. As will be obvious in Section 3, the brighter a mesh model is in Figure 6, the better the rate/distortion tends to be for the progressive coder [1] implemented on the S-Octree. The differences in brightness in Figure 6 can be explained by the face normal vectors that will be formed in S-Octree as shown in Figure 5; many face normal

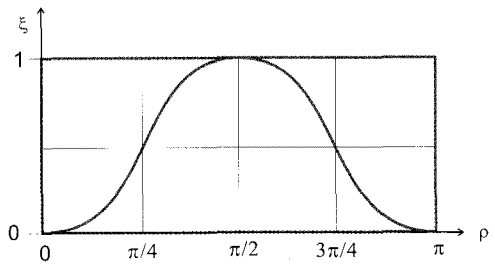


Fig. 4. Graph of equation (4).

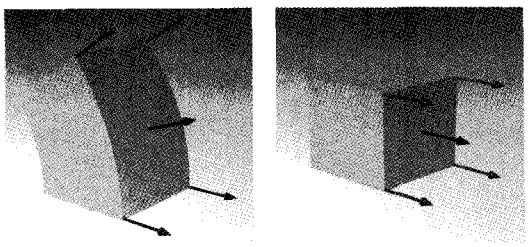


Fig. 5. Individual octree cells for the S-Octree (left) and the C-Octree (right). The red arrows indicate the normal vectors directly related to the grid structure and the blue ones indicate examples of simple prediction for the surface normals on the virtual surfaces across the cells.

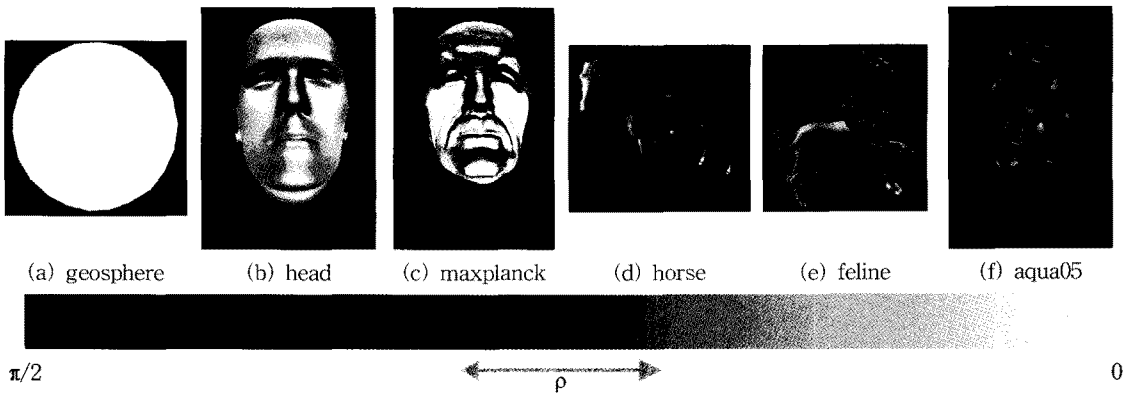


Fig. 6. Visualization of the closeness between surface normal vectors. The bright white areas indicate the normal vectors on those areas are very close to the surface normal vectors on the spherical surface and the dark ones are vice-versa. The geosphere model (a) shows that all surface normal vectors coincide with the unit vector a_R. (b) and (c), which are human faces show that lots of face normal vectors are very close to the direction of R axis as shown in Figure 3. Meanwhile, (d), (e), and (f) show darker areas.

vectors in S-Octree are parallel with those surface normal vectors on a sphere (i.e. a_R).

3. Applications to Progressive Mesh Coding

In mesh compression, geometry information (i.e. the positions of vertices or surface elements) is considerably larger than topology (connection) information. Thus, there are more opportunities for optimization in geometry compression and we focus on octree-based geometry compression in this study.

Among many compression techniques, the progressive compression methods based on the space-partitioning structures 1) utilize the inherent features of the partitioning structures (eg. octree, kd-tree, etc) for grouping geometry information, and 2) predict the next geometry information to code and then compress the error between the predicted value and the actual one.

To predict the unknown geometry information of the meshes, lots of prediction techniques apply the parallelogram prediction technique [4] or a modified version of it. This method works better if the triangular faces are regular. For isosurface compression, similarities in neighbor geometry elements are used to predict the unknown geometry information in [9]. As described in [10], basically those techniques are based on the assumption that the differences between any neighboring elements are small. However, the octree subdivision always traces the high frequency components (i.e. surface), this assumption does not always work for the octree coding. For this issue, [1] takes a different approach with a sophisticated prediction technique adopting the pseudo-probabilities reducing information entropy by sorting and indexing information as a group (i.e. "tuple").

However, most of the previous geometry compression studies focus on the prediction technique and pay less attention to the adaptive partitioning structure. As we discussed in the previous sections, the octree itself also has the ability to store geometry information efficiently. In this regard, the proposed S-Octree handles the geometry data more efficiently thanks to those properties discussed.

3.1 Tested Progressive Octree-based Coder

We apply our S-Octree to the progressive mesh coder proposed in [1]. In this progressive coder, every octree subdivision is encoded by arithmetic coding 1) the number of non-empty child cells

(designated as T) and 2) the sorted index representing the tuple of child cells. Because the octree cells tend to have less child cells at lower levels and only one (i.e. T=1) below certain levels, each octree cell level is given as a context for the arithmetic coder. Also, based on the observation that the number of connective neighbor octree cells, or valence, is also related to T, the valence is used as a context for arithmetic coding T.

After coding T, some arithmetic calculation is performed to quantify the possibility of being non-empty cell (i.e. *priority value*) for each child cell. For all the ${}_8C_T$ tuple combinations, pseudo-probabilities are estimated by summing up the priority values for any non-empty child cells. Those pseudo-probabilities are sorted out and the index which represents the current tuple in the octree cell to encode is arithmetic coded, with T as a context. For more information, see [1].

3.2 Compression Rates and Distortions

For comparison purpose, we have implemented the geometry coder in [1] both for the C-Otree and the S-Octree. Those coders have been implemented in Windows OS using MFC (Microsoft Foundation Class library) and OpenGL. Also, we have used METRO tool [8] for measuring distortion rates based on L2 norm. Figure 1 shows the intermediate subdivision processes for the S-Octree and the C-Octree from level 2 to level 5. In the C-Octree implementation, the compression rates for geometry information show 1.3% of discrepancy on average between the compressed results with our implementation and those reported in the original paper for the tested models. Because our main interest is geometry coding, the connectivity coder is not considered in this paper and uncompressed connectivity information is given to the geometry coder.

In Table 1, the meshes tested are listed with model codes and number of vertices and faces. The test meshes are also shown in Figure 7. The meshes are organized into three classes; class A includes those meshes which are topologically closer to the sphere surface, class C has non-manifold meshes and flat ones, and those meshes in class B are manifold meshes but not that close to the sphere surface. The classification is done by calculating the percent of surface normal vectors using the equation (4). The column (a) and (c) in Table 2 show the number of the octree cell subdivisions for the C-Octree and the S-Octree, respectively. All of the cases, both the C-Octree and S-Octree, all of the meshes are subdivided till level 12 for each vertex.

Table 1. Meshes details

model code	mesh name	# of verts	# of faces
A-1	geosphere random	1086	2168
A-2	venus	8268	16532
A-3	head	11703	23402
A-4	maxplanck	25445	50801
A-5	tiger high	2738	5472
B-1	horse	19851	39698
B-2	feline	49864	99732
B-3	bunny	35948	69451
B-4	armadillo	172974	345944
B-5	dragon	100250	202520
C-1	aqua05	16784	50370
C-2	nefertiti partial	299	562
C-3	nefertiti remesh 9k	4492	8737
C-4	nefertiti holes	4291	8288
C-5	chandelier	56795	113326

In [1], each cell subdivision is coded as an index of the tuple for the distribution of child cells. So, the numbers appear in column (a) and (c) in Table 2 indicate the amount of information to code the same geometry information, i.e. the higher this number is, the less efficient the octree structure in describing the geometry data. In the sixth column ("(a)/(c)") of Table 2, we can see that the S-Octree requires less subdivision except the model C-4 and C-5. In the third and the fifth columns ((b) and (d)) in Table 2, the number of cases when T is

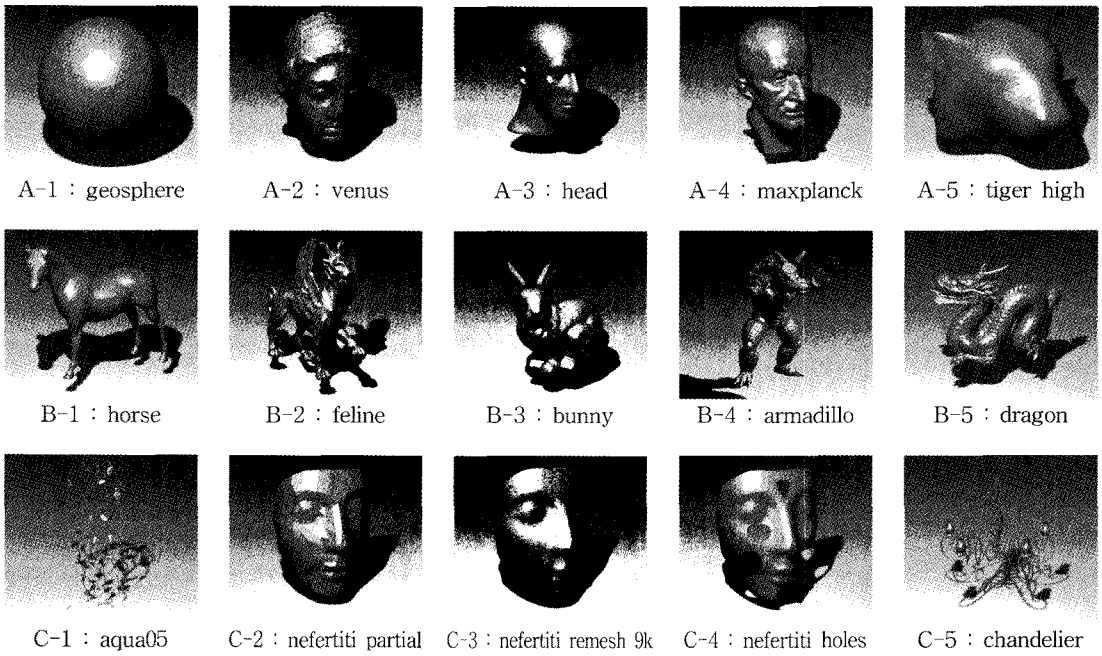


Fig. 7. Test meshes.

Table 2. Number of octree cells to code

Model code	C-Octree		S-Octree		(a)/(c)	(b)/(d)
	# of total subdiv. (a)	# of cases T=8 (b)	# of total subdiv. (c)	# of cases T=8 (d)		
A-1	7458	1	7106	6	1.05	0.17
A-2	44636	2	42992	22	1.04	0.09
A-3	59593	20	57017	40	1.05	0.50
A-4	113795	21	112488	38	1.01	0.55
A-5	16826	6	16329	12	1.03	0.50
B-1	89131	31	86158	60	1.03	0.52
B-2	200960	155	193063	217	1.04	0.71
B-3	155448	27	148202	75	1.05	0.36
B-4	540820	85	528822	158	1.02	0.54
B-5	383870	135	360559	233	1.06	0.58
C-1	22727	7	22030	4	1.03	1.75
C-2	2211	1	2189	2	1.01	0.50
C-3	25329	1	25187	7	1.01	0.14
C-4	23156	2	23438	8	0.99	0.25
C-5	142192	257	165614	317	0.86	0.81

8 is shown. As discussed in the previous sections, this number represents the number of non-empty child cells for each octree subdivision. Since there is only one possible combinatory tuple when T=8 (i.e. 8C8=1), we do not need to code the index for that tuple. In other words, if there are more cases

for T=8, we can conclude that the geometry information is handled more compactly. As we can see in Table 3, the amount of bits required to code the sorted tuple indices are about 6 or 7 times larger than those to code T values. In this regard, improvement in coding tuple indices will have higher impacts on the whole performance.

As shown in the last column ("(b)/(d)") of Table 2, the S-Octree has more cases where T=8 except Model C-1.

Table 3 lists the bit rates and distortions for each case. To analyze the difference between the C-Octree and the S-Octree, the bit rates for coding the T values and the tuple indices are separately presented. As shown in Table 3, the bit rates for the T values have the tendency to be less for the C-Octree than the S-Octree. However, the bit rates for the tuple indices are likely to be lower for the S-Octree than the C-Octree.

The above results can be explained as follows. First, the improved bit rates for the tuple index coding in the S-Octree can be explained with the smaller number of total subdivisions to code, as discussed for Table 2 (compare column (a) with (c)). Also, the fact that the cases where T=8 are more frequent for the S-Octree contributes the improvement in the bit rates for the tuple indices. Second, as for the higher T values in the S-Octree, we have found that the contexts (i.e. level and valence) for encoding T values are not optimized in the S-Octree. One of the reasons is that the geometry information in the S-Octree is sampled more compactly so that the distribution becomes less distinct and less partial according to different levels. In that sense, the coding for T values in the S-Octree has some more opportunities for improvement by developing proper contexts. As for the distortion rates, the *geosphere random* model (A-1) shows particularly low distortion rate in the S-Octree. This result is easily predictable because sphere surface models have extremely narrower interval for the R axis in spherical coordinates and

Table 3. Bit rates (for T and tuple) anddistortionatlevel12

model code	C-Octree				S-Octree			
	T* (e)	tuple* (f)	total† (e)+(f)	dist§	T* (g)	tuple* (h)	total† (g)+(h)	dist§
A-1	2.79	20.33	23.12	0.48	2.79	19.27	22.06	0.08
A-2	1.97	15.86	17.83	0.48	2.20	15.45	17.65	0.44
A-3	1.87	12.78	14.65	0.46	1.95	12.44	14.39	0.50
A-4	1.94	12.38	14.32	0.44	1.96	12.36	14.32	0.51
A-5	1.86	17.27	19.13	0.47	1.96	16.91	18.87	0.47
B-1	1.78	11.72	13.5	0.44	2.11	11.35	13.46	0.58
B-2	1.72	11.01	12.73	0.42	2.03	10.50	12.53	0.67
B-3	1.43	11.87	13.3	0.46	1.78	11.01	11.01	0.58
B-4	1.40	8.57	9.97	0.47	1.73	8.43	10.16	0.55
B-5	1.61	10.76	12.37	0.44	1.92	10.26	12.18	0.66
C-1	0.91	4.41	5.32	0.01	0.92	4.35	4.35	0.01
C-2	5.52	24.00	29.52	0.45	5.62	23.87	23.87	0.57
C-3	1.44	14.13	15.57	0.41	1.71	14.33	14.33	0.77
C-4	2.03	14.04	16.07	0.41	1.71	14.33	14.33	0.69
C-5	1.85	7.38	9.23	0.49	2.11	8.83	10.94	0.47

* and † are measured in bits per vertex. § distortions are measured by equation (1).

all the surface normal vectors match 100 % with the unit normal vector along the R axis, a_R. As listed in the fifth and the last columns in Table 3, the differences in the distortion rates in Class A tend to be very small except A-1. In particular, for *head* and *maxplanck* (A-3 and A-4), there are considerable "dark" face normal vectors on the necks and under the chins as shown in Figure 6-(b) and (c). We believe those dark areas degenerate the distortions in the S-Octree. Just like the findings that the distortion rate tends to be less if the model is closer to the sphere surface in the S-Octree, we find that a mesh model tends to have less distortion in the C-Octree if it is as flat as a plane and orthogonal to any of three coordinate planes (i.e. XY, YZ, and ZY planes). In Table 3, we see this tendency for three versions of *nefertiti* mesh models (C-2, C-3, and C-4). In the mesh C-1, whose face normal vectors are not particularly biased for both the S- and C-Octree, the distortions for both are the same within given precision.

4. Conclusion and Future Work

An alternative form of the octree structure is presented in this paper. By applying the S-Octree to a progressive mesh coder, we find several performance parameters are improved including the compression rate and distortion (Table 3) in general. We view this result promising because the applied mesh coder from [1] is not particularly optimized for the S-Octree. Thus, we expect a better progressive mesh coder optimized for the S-Octree is feasible. Based on the discussions in the previous sections, we believe that there are various applications for the S-Octree. In the future, we will apply the S-Octree structure to other applications. Since there are lots of areas requiring the octree subdivision, we believe that there are multiple opportunities for the S-Octree to improve the applications' performance.

REFERENCES

[1] J. Peng and C-C-J. Kuo, "Geometry-guided progressive lossless 3D mesh coding with octree decomposition," *ACM Trans. Graphics*, Vol. 24, No. 3, pp. 609-616, 2005.

[2] C. Y. Lin et al., "Toward automatic reconstruction of 3D environment with an active binocular head," *In Proceedings of the 14th International Conference on Pattern Recognition*, Vol. 2, pp. 1708-1710, 1998.

[3] P-M. Gandoin and O. Devillers, "Progressive lossless compression of arbitrary simplicial complexes," *ACM Trans. Graphics*, Vol. 21, No. 3, pp. 372-379, 2002.

[4] C. Touma and C. Gotsman, "Triangle mesh compression," *In Proceedings of Graphics Interface*, pp. 26-34, 1998.

[5] H. Lee et al., "Progressive encoding of complex isosurfaces," *ACM Trans. Graphics*, Vol. 22, No. 3, pp. 471-476, 2003.

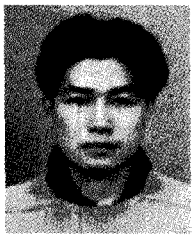
[6] T. J. Park et al., "Progressive compression of geometry information with smooth intermediate meshes," *In Proceedings of the 3rd Iberian Conference on Pattern Recognition and Image Analysis*, Part II, pp. 89-96, 2007.

[7] Y. Huang et al., "A Generic Scheme for Progressive Point Cloud Coding," *IEEE Trans. on Visualization and Computer Graphics*, Vol. 14, No. 2, pp. 440-453, 2008.

[8] P. Cignoni et al., "METRO: Mesauring error on simplified surfaces," *Computer Graphics Forum*, Vol. 17, No. 2, pp. 167-174, 1998.

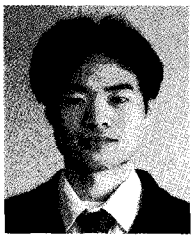
[9] G. Taubin, "BLIC: Bi-level isosurface compression," *In Proceedings of IEEE Visualization*, pp. 451-458, 2002.

[10] L. Ibarria, et al., "Spectral interpolation on 3x3 stencils for prediction and compression," *Journal of Computers*, Vol. 2, No. 8, pp. 53-63, 2007.



Taejung Park

1997. BS in Electrical Engineering, Seoul National University
1999. MS in Electrical Engineering, Seoul National University
2006. PhD in Electrical and Computer Engineering, Seoul National University
2006~Current. BK21Research Professor, Korea University
Research interests: geometry compression, TCAD, 3D modeling



Sung-Ho Lee

2005. BS in Dep. of Compueter Science, Korea University
2007. MS in Dep. of Compueter Science, Korea University
2007~Current. PhD Student in Dep. of Computer Science, Korea University
Research interests: 3D modeling, data structures for 3D shapes, image-based rendering

Chang-Hun Kim

1979. BS in Dept. of Economics, Korea University, Korea
1988. PhD in Dept. of Computer Science, University of Tsukuba, Japan
1995~Current. Professor, Dept. of Computer Science & Engineering, Korea University
2005~2006. Director, Korean Institute of Information Scientists and Engineers
2009~2010. Vice-Chairman, Korean Institute of Information Scientists and Engineers
Research interests: computer graphics, physics-based simulation, mesh processing