UCL DEPARTMENT OF
COMPUTER SCIENCE

Research Note
RN/06/15

PNORMS: Platonic solid derived Normals for error bound compression

18th May 2006

João Fradinho Oliveira

Bernard Buxton

Abstract

3D models of millions of triangles invariably repeatedly use the same 12-byte unit normals. Several bit-wise compression algorithms exist for efficient storage and progressive transmission and visualization of normal vectors. However such methods often incur a reconstruction time penalty, which in the absence of dedicated hardware acceleration, make real-time rendering with such compression/reconstruction methods prohibitive. We present a new method for the compression of normals that does not require reconstruction, hence allowing for real-time rendering; normals are simply looked up from a table of representative normals at run-time. Our method follows an analogy with GIF image compression rather than with JPEG. We create Hierarchical databases of normals derived from 5 subdivided Platonic solids for a target application error and for a set normal index byte length. We study their respective distributions of normals, and associated error bounds. We present a fast method for encoding a normal as computer from the vertices of a triangle in the model to a representative normal in the database, which can be used at run-time for example to dynamically encode new normals of large sets of modified triangles in the context a modelling task, and a method that is slower but more accurate. Different levels of a database allow for different cartoon like shading effects. The advantages of these databases are that they can be re-used for any object with bounds on the maximum errors independent of the object. In particular we show that subdividing the icosahedron gives a smaller maximum and mean error than its counterparts Platonic solids. We present results using 2-byte indices for a target max error of 1.3° degrees and 4-byte for a max error of <0.1°.

PNORMS: Platonic solid derived Normals for error bound compression

João Fradinho Oliveira* Bernard Buxton*

* - Department of Computer Science, University College London

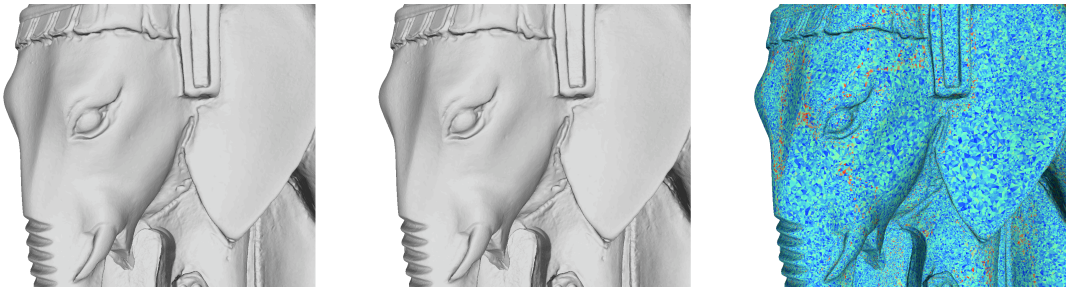

Figure 1: *left: Flat-shaded rendering of the original 12-byte 10 million triangle normals; centre:with a max error of 2.5 degrees, using 27300 12-byte normals from a 2-byte index icosahedron database (encoding in 95secs) right: colour coded error distribution, max error in red.*

Abstract

3D models of millions of triangles invariably repeatedly use the same 12-byte unit normals. Several bit-wise compression algorithms exist for efficient storage and progressive transmission and visualization of normal vectors. However such methods often incur a reconstruction time penalty, which in the absence of dedicated hardware acceleration, make real-time rendering with such compression/reconstruction methods prohibitive. We present a new method for the compression of normals that does not require reconstruction, hence allowing for real-time rendering; normals are simply looked up from a table of representative normals at run-time. Our method follows an analogy with GIF image compression rather than with JPEG. We create Hierarchical databases of normals derived from 5 subdivided Platonic solids for a target application error and for a set normal index byte length. We study their respective distributions of normals, and associated error bounds. We present a fast method for encoding a normal as computer from the vertices of a triangle in the model to a representative normal in the database, which can be used at run-time for example to dynamically encode new normals of large sets of modified triangles in the context a modelling task, and a method that is slower but more accurate. Different levels of a database allow for different cartoon like shading effects. The advantages of these databases are that they can be re-used for any object with bounds on the maximum errors independent of the object. In particular we show that subdividing the icosahedron gives a smaller maximum and mean error than its counterparts Platonic solids. We present results using 2-byte indices for a target max error of 1.3˚ degrees and 4-byte for a max error of <0.1˚.

CR Categories: I.3.0 [Computer Graphics]: General

1 Introduction

Many applications such as global models of fluid flow in meteorology and oceanography employ spherical geodesic grids that are based on a subdivided icosahedron, which leads to a quasi-uniform distribution of points without singularities at the poles [RRH*02]. Each triangle of a icosahedron, octahedron or tetrahedron (Figure 2) can be subdivided to create 4 new sub-triangles, by inserting 3 new vertices at the midpoints of a triangle's edges [WDS99]. The vectors representing these midpoints can then be normalized to unit length, and hence projected onto a sphere (Figure 3). The normals of these sub-triangles constitute a finite set of normals with different distribution characteristics for each solid. Normals within these sets can be re-used by several triangles or vertices of an object for both flat, and Gouraud shading. We study these distributions in the context of using lookup databases for surface normal compression. The presented method allows for 83.4% memory savings of normal attributes, with a bounded imperceptible max error of 2.5 degrees, without any reconstruction time. Several compression strategies exist for efficient compression of mesh geometry, connectivity/topology, and attributes. These methods are very useful for saving hard disk space when storing large meshes, and for progressive transmission and visualisation of compressed meshes over a limited network. However these methods require reconstruction time in order to retrieve and decompress the original data which might be run-length encoded along with other, different compressed attributes such as geometry, connectivity and colour, making the decompression process in a real-time setting prohibitive in the absence of dedicated hardware support.

Figure 2: *base Platonic solids; a) icosahedron b) octahedron c) tetrahedron d) cube e) dodecahedron*

This paper follows an analogy with GIF image compression rather than JPEG. The GIF image format sets the colour index to one single byte, to access 256 colours in a table, rather than utilising a variable bit length, for example, to exploit patterns which could be run-length compressed. The latter approaches tend to lead to complicated coding and decoding procedures which can prohibit real-time reconstruction. Non web-browser viewers of animated GIFs are able to render and switch images at a very fast rate, by just looking up colour indices in a table for the each pixel colour. In contrast, JPEG images are able to provide a fast overview of the full image, but require more time to switch between two full resolution images. Similarly we build lookup databases of a finite set of normals, following the observation that meshes of high resolution scanned data repeatedly use the same unit normals amongst their triangles. Unlike GIF, our tables contain coarse normals of a base solid, along with finer normals of deeper subdivisions in the same table so as to allow fast multi-resolution querying of normals in our encoding phase. Once the original true normals computed from the vertices of the triangles comprising a model have had a normal index assigned in the encoding phase, then 12 byte normals are simply looked-up at run-time for rendering. We have created several hierarchical databases of normals by subdivision of triangular nets based on the 5 Platonic solids. We created databases for 2 byte normal indices (2^{16} = 65536 possible look up normals in the table) and 4 byte (2^{32} = 4.294.967.295 normals). Different Platonic solids produce different numbers of triangles at each level when we repeatedly quadruple the number of their initial triangles. In Table 1, the row labelled zero on the left, shows the number of initial triangles of each base Platonic solids, whilst subsequent rows show the number obtained on subdivision. This paper shows which of the 5 solids yields the best *bang per buck*, in terms of which one achieves the smallest representation error, given a restriction on the number of normals each solid can generate so as to not overflow the number of possible normals accessible with 2 bytes or with 4 bytes.

We briefly review related work in section 2. In section 3 we study various subdivisions. In section 4 we present a fast and a slower but more accurate method of encoding a normal into a database with a bounded max error of 2.5 degrees, using a 5 times subdivided icosahedron, with 2 byte normal ids rather than the conventional 12 bytes. This error we found suitable for visualization of 3D models. The fast method is quick enough to allow dynamic re-encoding of large sets of modified triangles for example, in a modelling task. The slower but more accurate method is for applications that require sub-degree precision. We note that the encodings produced by both methods can be accessed at the same speed at run-time. In section 5 we present results, in section 6 we discuss a fair comparison between the distribution of normals derived from the different Platonic solids, and conclude in section 7.

2 Related Work

Compression of mesh geometry [Cho97], connectivity [TR98], and attributes [BPZ99] is a mature field. In this section we review methods that specifically compress normal attributes.

Storing a database of normals for lookup in a GPU is not new, Deering [Der95] uses 18 bits for each triangle normal index, we present bounded normal errors using 2 bytes/16 bits. Taubin et. al [THL*98] and [Der95] parameterize a sphere, which essentially equates to using an octahedron. They subdivide each base triangle of the octahedron into 4, and compress the normal id associated with the last sub triangles. We note that such compression imposes a reconstruction time; which can be prohibitive for real-time rendering of large meshes without dedicated hardware. Our method stores the normals of all subdivision levels, so as to allow for hierarchical querying of normals in the encoding phase, once normals are assigned to a representative normal, no reconstruction is necessary.

Deok-So et. al, [DYH04] note that, in previous work, normal ids are created independently of the concentration of normals in a model. They also note that a triangle normal is bound to be refered in neighboring triangles, they use normal clustering to find representative normals, furthermore they use a relative indexing scheme to further compress the sequence of normals. It would be desirable to encode on the fly large sets of new triangle normals, for example when one modifies a mesh, it is not clear that such high compression methods, can be used in such run-time context. Deok-So also notes that whilst several excellent compression methods developed in the past decade achieve excellent compression rates, not much attention has been given to the errors incurred in such normals, we refer the reader to a more extensive review in [DYH04].

Guskove et. al [GVSS00] parameterize a subdivision surface, where multiple level of details can be obtained from a base coarse mesh and a corresponding offset value. Each vertex can be stored with just one single float. This representation requires the mesh to be semi-regular, which in some cases requires the mesh to be remeshed, and is not suitable for non-manifold meshes, or CAD objects. This problem is often shared also by approaches that look for sequences of triangle strips [BPZ99]. Some methods such as [Cho97] store a compressed representation in main memory, and use a fast decoder to render in real-time. It is not clear whether this approach allows for encoding at run-time large triangle sets.

3 Platonic normals

Our compression method consists on 3 different phases, the first phase; which is only performed once, consists on the subdivision of Platonic solids and construction of hierarchical databases that once created and saved can be used for any object, with bounds on the max error. The second phase is an encoding phase, which consists on finding a representative normal in the database for each

true normal computed from the objects geometry. Once a representative normal is found, an id denoting the position of the representative normal in the database is stored. These normals ids, along with the database can be readily saved to file. The third and final phase, just looks up the normal ids at run-time for retrieving the full 12 byte normals for rendering. We show that there is no penalty in this lookup in section 5.2.

All 5 Platonic solids described by the Greek philosopher Plato (icosahedron, octahedron, tetrahedron, cube, and dodecahedron Figure 2), share the following interesting properties: The vertices of each solid all lie on a sphere; each solid has the same dihedral angle between its adjacent polygons (138.190°, 109.471°, 70.529°, 90°, 116.565° respectively); their polygons are regular; all their vertices are surrounded by the same number of faces; and their solid angle is equivalent, e.g. the area on a sphere of the projected solids surface is equivalent [Wei06]. We use some of these properties in different parts of PNORMS. For example, the property that all vertices lie on a sphere, allows one to divide or parameterize a sphere into different manageable parts; this in turn helps the subdivision (Figure 3) and construction phase of the hierarchical databases of representative normals described in section 3.1. The property of a Platonic solid having the same dihedral angle amongst all adjacent polygons helps the encoding phase of the accurate approach presented in section 4.2. We mention polygons, rather than triangles, because the dodecahedron is made of pentagons, and the cube is made of quads. Nevertheless in section 3, we have triangulated quads and pentagons into different triangle arrangements or polygon nets so as to be able to subdivide and study more distributions of normals.

3.1 Subdivision: Hierarchical database construction of representative normals

In this section we describe how to create hierarchical databases of representative normals. We note that contrary to other methods, we store the normals of every subdivision level. This strategy allows one to make fast hierarchical queries of the database at run-time for encoding new true normals on the fly. If we insert and normalize new vertices at edge midpoints for each base triangle as illustrated for triangle ABC in Figure 3, we obtain 4 sub triangles which can be used to compute 4 finer grain representative normals for the sphere projected area covered by the base triangle.

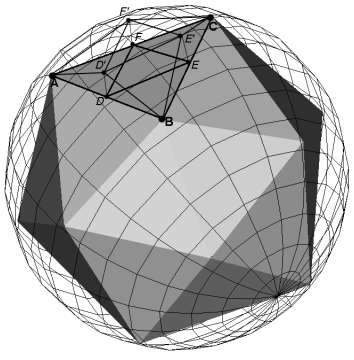

Figure 3: *subdivision of the base triangle ABC of the icosahedron, into triangles ADF, DBE, ECF, and FDE; inserted vertices at the midpoint of edges AB, BC, CA are normalized/projected to a unit sphere to form triangles AD'F', D'BE', E'CF', and F'D'E'.*

Each Platonic solid, is formed by a different number of base polygons; 20, 8, 4, 6, 12 for respectively the isosahedron, octahedron, tetrahedron, cube, and dodecahedron. Before we can subdivide the cube and dodecahedron, we need to create triangle tessellations of their quads and pentagons respectively. These triangle tessellations or polygon nets are arrangements of triangles that can affect the resulting normal distributions; the nets we studied are shown in Figure 6. After tessellation, the cube consists of 12 base triangles, and the dodecahedron of 36 base triangles.

Subdividing a Platonic solid at run-time would require connectivity information, geometric updates and memory management, hence run-time overheads. We opt for creating these subdivided models once, offline, and store only the generated representative normals in an orderly fashion in an array, this is similar to a GIF file's colour palette/array that gets saved in the header file of a gif image. Unlike GIF, we store the hierarchical databases separately from an object, in order to be able to re-use the same database for multiple objects. The first 20 base normals of an icosahedron occupy the first 20 positions of the array, the four subdivided normals of the first base triangle are added to the array next, and occupy positions 20, 21, 22, 23. The four subdivided normals of the second base triangle, are stored next in 24, 25, 26, 27, further levels of subdivision are added in the same manner to the array.

After subdividing each solid 5 times (Figure 4), one can see how uniform a distribution is, by observing how uniform the resulting triangles are in size, and how regular the tessellation is. The number of generated representative triangle normals may or might not contribute to lower errors as we will show in section 5. We note that the 5 times subdivided icosahedron in Figure 4, generates 20480 representative normals (first column of Table 1, row 6, the first row has the number of base triangles, and is not a subdivision), whilst the five times subdivided octahedron generates significantly less normals (8192 representative normals, second column of Table 1, row 6). However we can see from Figure 5, that the 4 times subdivided icosahedron with 5120 representative normals, is shown to be already more regular than the 5 times octahedron. Figure 4 and 5 also show that the different polygonal nets (shown in Figure 6) for the cube and dodecahedron did not vary significantly the overall characteristics of their distributions. The tetrahedron appears to generate the least regular distribution.

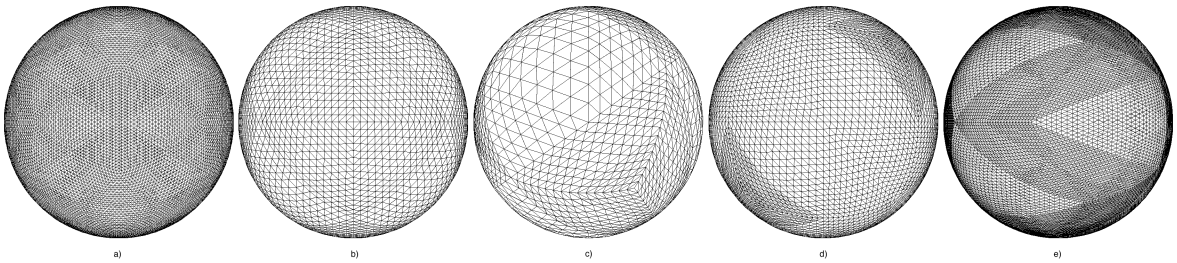

Figure 4: *platonic solids subdivided 5 times; a) icosahedron b) octahedron c) tetrahedron d) cube (using polygonal net 6c) e) dodecahedron (using polygonal net 6a)*

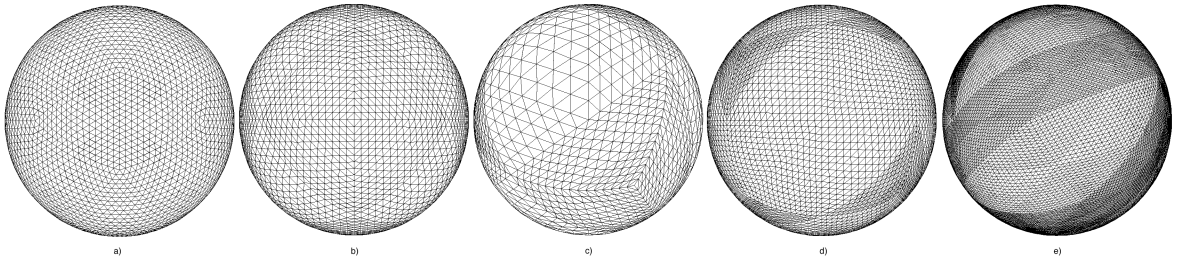

Figure 5: *a) icosahedron subdivided 4 times b) octahedron subdivided 5 times c) to e) solids subdivided 5 times c) tetrahedron d) cube (using polygonal net 6d) e) dodecahedron (using polygonal net 6b)*

accurate encoding - cascaded tolerance formula 9 subdivided PNORMS									
	icosaheron			*octahedron*			*tetrahedron*		
d	M	me	m2	M	me	m2	M	me	m2
0	**37.1**	*18.1*	32	**54.6**	*31.4*	54.2	**70.1**	*37.5*	69
1	**19.2**	*9.6*	17.7	**30.3**	*16*	29.4	**45.9**	*22,9*	44.3
2	**10**	*4.7*	8.68	**16.1**	*8*	14.6	**27.4**	*10.8*	21.1
3	**5.33**	*2.31*	4.29	**9.27**	*4*	7.35	**29.9**	*8.88*	18.6
4	**2.69**	*1.15*	2.14	**4.87**	*2.01*	3.7	**31.1**	*7.31*	17.7
5	**1.32**	*0.58*	1.08	**3.13**	*1.03*	1.93	**32.1**	*6.82*	18.3
6	**0.66**	*0.29*	0.54	**2.49**	*0.53*	1.07	**32.7**	*6.68*	18.9
7	**0.36**	*0.14*	0.27	**2.27**	*0.28*	0.69	**33.0**	*6.6*	19.2
8	**0.18**	*0.07*	0.13	**2.19**	*0.15*	0.54	**33.2**	*6.64*	19.4
9	**0.09**	*0.03*	0.07	**2.15**	*0.09*	0.48	**33.3**	*6.64*	19.4

Table 2: *maximum error (M), mean errors(me), and the error of 2 standard deviations from the mean(m2) for the Stanford bunny using accurate encoding; icos.(left), octa.(middle), tetra.(right)*

A developer can use 2 byte or 4 byte index databases of representative normals, depending on his application requirements. If we use 2 bytes for each triangle normal index, instead of the 12 bytes of the original normal, the statue of Lucy with 28 million triangles takes 56 Mbytes instead of 336 Mbytes. Storing the 5 levels of an icosahedon database is not expensive (20+80+320+1280+20480=27300x12bytes=327 Kbytes), but has to be taken into account in the total number of accessible normals for 2 bytes shown in brackets for each solid in Table 1. In all 83.4% memory can be saved. We address error in section 5.

	icosa-hedron	*octa-hedron*	*tetra-hedron*	*cube*	*dodeca-hedron*
0	20	8	4	12	36
1	80	32	16	48	144
2	320	128	64	192	576
3	1280	512	256	768	2304
4	5120	2048	1024	3072	9216
5	**20480** (27300)	8192	4096	12288	**36864** (49140)
6	81920	**32768** (43688)	**16384** (21844)	**49152** (65532)	147456
7	327680	131072	65536	196608	589824
8	1310720	524288	262144	786432	2359296
9	5242880	2097152	1048576	3145728	9437184

Table 1: *Maximum indexable normals in curved brackets including all normals of each subdivision level for each Platonic solid, for 2 byte indices (2^{16}=65536). For 4 byte indices (2^{32} =4.294.967.295) are indexable. The number of triangle normals of the last subdivision level for 2 bytes is highlighted in bold.*

An icosahedron can only be subdivided 5 times before overflowing the number of indexable normals with 2 bytes: 20 (base normals)+80+320+1280+5120+20480 = 27300) < 65536 normals representable with 2-bytes. A sixth subdivion, first column row 7, would create 81920 which would exceed the 65536 budget. We were excited to see that the cube achieves a number of triangle normals very close to the theoretical limit of accessable normals for 2 bytes, in other words, very little address space is wasted: 12+48+192+768+3072+12288+49152=65532 < 65536, unfortunately more normals does not translate directly into less error, as we shall see in section 5.

In an attempt to maximize the address space of 2 byte indices, we created a Master object of normals, that included two rotated hiearchical databases of a 5 times subdivided icosahedron (54600<65532). Each of the two databases of normals of the Master object of normals had an offset to their starting position in the array. The Master Object would query both databases to search for the best representative normal for a given true normal, and would return the corresponding normal id.
We tried different rotations, and noted that whilst the mean error did decrease with more representable normals available, the maximum error did not. This is due to the fact that large portions of the second sphere successively reduced the error for several normals affecting the mean error, but ultimately the placing of the second database will align with the first database at places, and hence not contribute in reducing the maximum error in those places.

The polygonal nets that we built for converting the quads and pentagons of the cube and dodecahedron of figure 4 and 5, were connected with two criterias in mind, the encouragement of local uniformity around the most number of vertices possible (Figure 6; 6a and 6c) and a global uniform criteria (Figure 6; 6b and 6d). As can be seen in Figure 4, and 5 (4th and 5th subfigures from left)), these tessellations unfortunately did not improve the overall distribution characteristics of the subdivided cube and dodecaherdron.

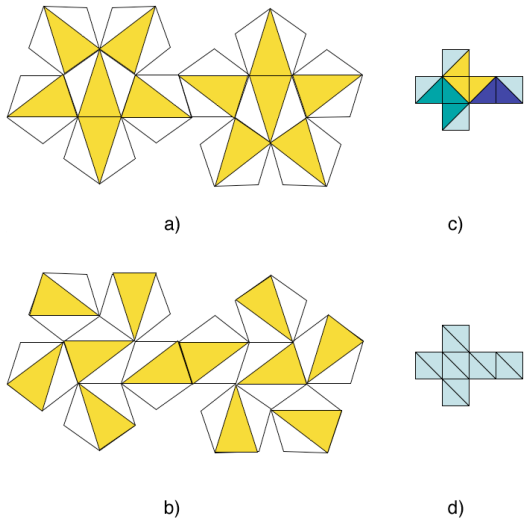

Figure 6: *polygonal nets 6a) dodecahedron 6b) dodecahedron 6c) cube 6d) cube*

4 Encoding

Now that we have created the hierarchical databases, we present two methods for encoding or finding a representative normal in the database for a given computed true normal. We present a fast method suitable for encoding large sets of triangle normals at run-time, and a slower but more accurate method for applications that require more precision.

4.1 Fast encoding

For each triangle of a model, typically during reading the model from a file we readily compute its true normal with a cross product of its vertex pairs. We then dot product this 12 byte true normal with all the normals of the base unsubdivided platonic solid. Note that the normals in the database are full 12 byte normals, and that the base normals occupy the first positions in the array database. The inverse cosine of this dot product will indicate which of the base normals has the smallest angle error with the true normal, and hence approximates it better. If subdivided normals exist in the database we further compare the true normal with the 4 finer grain normals of the base triangle normal that had smallest

error. In order to assist the querying process, we precompute the start and end offsets for the beginning and end of each level in the database. For example the start offset for the 5th level of subdivision of the icosahedron is the sum of the base triangle normals plus all normals generated from the previous levels. The advantage of storing these start offsets, is that we can use a simple relative indexing to access the 4 normals that we are interested in at each level simply by knowing the level, the start offset, and the number of the triangle from the previous level.
normalid = levelstartid + ((previous level triangle no x 4) -1) + subtriangle id(ranging from 0-3)).

Two nice features of this encoding scheme, is that it only requires 36 dot products for finding a representative normal amongst 27300. (20 base triangle dot products+ 4levels*4 dot products). Since we are only interested in the relative smallest error normal at each level, we do not need to find the actual corresponding angle of the dot produtct, encoding Lucy's 28 million triangle normals took 303 seconds (Table 3, 5th column) on a PowerPC 500MGhz with 1 GByte of RAM, timings for several models are also available in the top row of Table 3 in section 5. Note that the same database produces a maximum error of not more than 2.5 degrees with all models, and that the time varies linearly with the input model. 1 million triangle normals took 9.6s (Table 3, 2nd column), this indicates that if we were to encode at run-time a set half as large it would take approximately 4-5 seconds.

Another nice feature is that no tolerances are required to produce a max error of 2.5 degrees with a 5 level subdivided icosahedron, Once a representative normal is chosen, the triangle gets the id of that representative normal's position in the array. In a profile mode, all the errors incurred are added, and divided by the number of triangles in the model to compute the mean average error, and two standard deviations from the mean. We also keep track of the largest error occurred as the max error. Figure 1, right shows the colour coding and distribution of errors in normals, with the maximum error in red, and blues with small errors.

4.2 Accurate encoding

In the previous section we presented a method based on offsets for hierarchically finding a representative normal for a true normal in a database. One problem with the fast approach is that a subdivided triangle will have a degree of symmetry with its 4 sub triangles, this symmetry cannot be captured with the previous approach. The problem lies when a true normal is centered or almost co-linear with a normal in the database. The 4 subdivided representative normals will have arbitrary small errors with the true normal. To cater for symmetry, we extend the previous offsetting approach to include a tolerance test, if a normal is inside the tolerance it is added to a list of triangle normals, all the subdivided normals from this list will be considered too.
From Figure 3, we observe that any horizontal line crossing the subdivided triangle, can cross at most 3 triangles. The dihedral angle between adjacent polygons of each Platonic solid is: 138.190°, 109.471°, 70.529°, 90°, and 116.565° for the icosahedron, octahedron, tetrahedron, cube and dodecahedron respectively. We use the supplementary angles in our calculations: 41.8°, 70.5°, 109.4°, 90°,and 63.4° respectively.
The tolerances are set as an inverted cascaded pyramid of decreasing values. The initial dihedral angle is divided by 3 at each subdivision level. The final angle tolerance for a given level uses the value that is divided by three at each level and multiplies it by 1.5 to cater for just below two full triangle normal angle deviations. Since we are dealing with angle tolerances, we need to find the inverse cosine of the dots products. The same tolerance formula allowed us to study the numerical error of all Platonic solids.

We note that an arbitrary large tolerance can significantly slow down encoding, as more triangle normals are considered in the lists at each level. We did exhaustive searching of angle tolerances to optimize for speed the accurate approach for different solids, the timings of the accurate approach in the bottom row of Table 3 used set radian tolerances for level 0-5 of: 0.05; 0.05; 0.02; 0.02; 0.01; 0.01, instead of using the cascaded tolerance formula. We note that the maximum error was not affected with this optimization, the Stanford Bunny model has a bounded maximum error of 1.3 degrees in Table 3, first column, bottom row, and in Table 2, first column, sixth row, it has a maximum error of 1.32 using the cascaded tolerance formula. In the next section we present results of both methods.

5 Results

5.1 Angle error analysis

Table 2 shows errors for the subdivided icosahedron, octahedron, and the tetrahedron, using accurate encoding with the same cascaded tolerance, starting with dehidral angles (41.8°, 70.5°, 109.4° respectively). Specifically it shows for each subdivision depth (d), the maximum error (M), the mean error (me), and the error for two standard deviations from the mean ~98% (m2). Figures 4 and 5, show that the cube and dodecahedron were too irregular, for subdivision, and produced errors too large to be considered. Similarly the tetrahedron shows very large and small triangles in the same mesh, this is consistent with all of its subdivision levels, we include it's results in Table 2.
The next table compares the time and maximum error of the fast encoding versus the fine tuned accurate encoding, for several models, using a x5 subdivided icosahedron database. It can be seen that the time varies linearly in both cases with the size of the input model.

fast versus accurate encoding - tuned tolerance 5 subdivided icosahedron, 2 byte index database				
Bunny 69k triΔ	Happy $1x10^6\Delta$	Thai $10x10^6\Delta$	PowerP. $13x10^6\Delta$	Lucy $28x10^6\Delta$
2.5°0.6s	2.5° 9.6s	2.5° 95s	2.4° 110s	2.5° 303s
1.3°3.6s	1.3° 53s	1.3° 535s	1.3° 948s	1.3° 1501s

Table 3: *max error & time for fast(top row) and accurate (bottom row.) encoding, with 2 byte normal indexing of x5 subdivided icosahedron*

5.2 Run-time performance

We used a PowerBook G4 500 MGhz, with 1 GB RAM for all the experiments in this paper. Figure 7 shows the frame rate of direct conventional rendering of normals versus PNORMS lookup when rendering the camera sequence of Figure 8. In particular it shows that there is no significant difference in frame rate from conventional rendering of true 12 byte normals, and run-time looked up PNORMS whilst using the graphics card with a window resolution 600x600, and rendering in software with a window size of 1142x718.

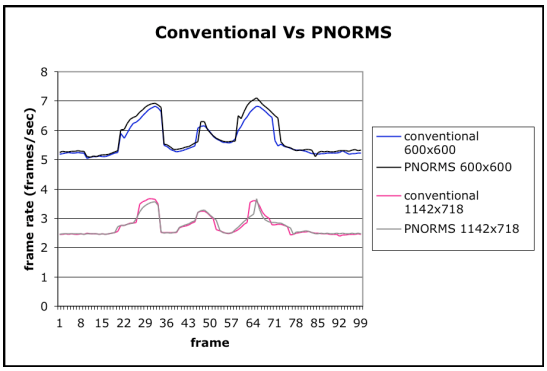

Figure 7: *frame rate of direct conventional rendering of normals versus PNORMS lookup and then rendering of camera sequence of Figure 7.*

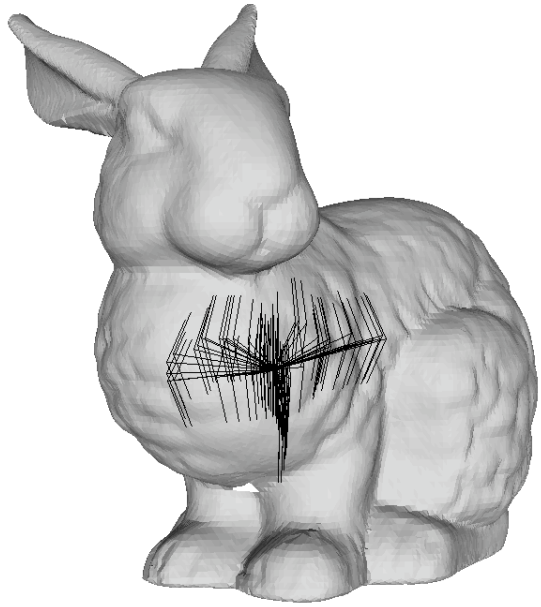

Figure 8: *camera view direction sequence for timings of Figure 7.*

5.3 Shading effects

Figures 9 and 10 show different shading effects by using representative normals of different subdivision levels of a 5 subdivided icosahedron, with the 8 million triangle normals of the of Michael Angelo's statue of David encoded with accurate encoding.

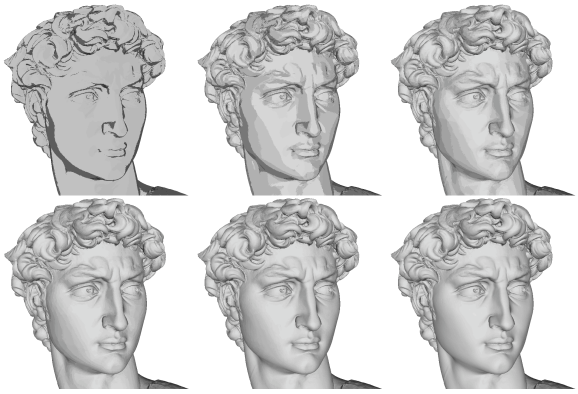

Figure 9: *Shading effects of the Statue of David, using 2 byte normal indices of different levels of a x5 subdivided icosahedron database. Normals encoded with accurate approach. top row: from left to right: subdivision level, and maximum error in curved brackets 0(37.3°); 1(19.3°); 2(10°); bottom row: from left to right 3(5.3°); 4(2.7°); original/true normals(0°).*

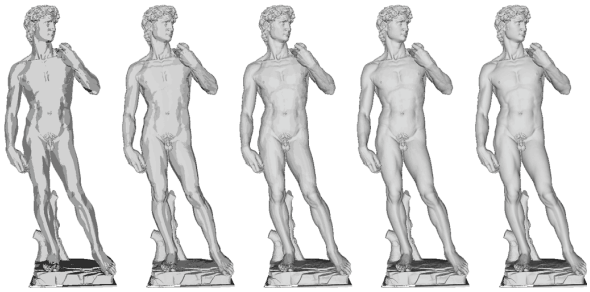

Figure 10: *Shading effects of the Statue of David, using 2 byte normal indices of different levels of a x5 subdivided icosahedron database. Normals encoded with accurate approach, from left to right: subdivision level, and maximum error in curved brackets 0(37.3°); 1(19.3°); 2(10°); 3(5.3°); 4(2.7°); original/true normals(0°).*

Table 2 shows that the icosahedron leads to the most accurate results, by cross referencing the number of accessible normals for 2 byte and 4 byte indices of differenent solids, with the errors in Table 2, it can be seen that the more regular icosahedron also generates the most *bang per buck*. Using 2 byte index normals and 1 byte for colour lookup (the colour coding of Figure 1 was achieved in this manner), the Lucy model takes 739Mbytes instead of 1.3Gbytes.

6 Discussion

We would like to assess how successful our cascading tolerance formula used in Table 2 was in comparing and using the different Platonic solid databases.
Looking at the mean error pattern of the icosaheron and the octahedron we can see that it follows the pattern of steady halving of error in both solids (row 0 downwards of Table 2), this would not be possible if the formula was not catering enough symmetrical normals at each level. We note that there is no hole of 33 degrees at depth 9 of the tetrahedron as suggested by Table2. Figure 4 shows that at level 5 the tetrahedron has larger subdivided triangles and less regular than the other solids, but the solid angle of these triangles is smaller than 33 degrees.

This error is partially due to the distortion of this solid, if an error is created higher up, the symmetry tolerance can not make up for a different normal bucket. If one considers a true normal that happens to be almost coinciding with a base normal, then the next 4 subdivided normals will have similar error, the fast method falls for this as it considers only the smallest angle, but the tolerance method can consider all 4 normals. The problem arises when an error is made higher up, the tolerances at the bottom cannot make up for incorrect binning.

We did not expect better results for other solids due to the inherent large and irregular triangles. We tried different triangle nets patterns/tessellations for both the cube and dodecahedron but they created even more distorted subdivided triangles than the presented tetrahedron. We note that the results achieved with the fast encoding algorithm presented in Figure 1 did not require any tolerance measure. The top row of Figure 9, shows images that are not dissimilar to GIF's 256 quantization of grey levels.

7 Conclusions

We showed that the number of generated normals resulting from the subdivision of platonic solids, does not necessarily equate to smaller errors in all solids. We presented hierarchical databases that could be used to save memory from normal attributes of up to 83% without any reconstruction time penalty at run-time. The performance of the fast encoding approach allows one to dynamically encode new true normals of large sets of modified triangles.

The presented hierarchical databases have upper bounds on the maximum error and mean error, and can be used by multiple objects simultaneously. We presented results using scanned statues, and the UNC power plant CAD model. We presented a fast encoding scheme that does not require tolerances, and a more accurate encoding method that caters symmetry with cascading tolerances. The icosahedron generated the best maximum and mean errors of the 5 platonic solids. We plan to make the 2 byte and 4 byte databases of normals available online. The 2 byte (x5 subdivided icosahedron, can be used to *accurate encode* normals with a max error bound of 1.3˚, or *fast encode* normals with a max error bound of 2.5˚) and the 4 byte (x9 subdivided icosahedron, can be used to *accurate encode* normals with a max error bound of <0.1˚, or *fast encode* normals with a max error bound not better than the fast encoding using x5 subdivided of 2.5˚,this is due to symmetry errors made higher up).

Acknowledgements

We would like to thank Matt Hall. credits: Stanford University (scanned models) University of North Carolina and Chapel Hill (Power Plant model).

References

[BPZ99] BAJAJ, C. L., PASCUCCI, V., and ZHUANG, G., Single resolution compression of arbitrary triangular meshes with properties, Computational Geometry: Theory and Application, Vol. 14, 1999, pp. 167-186 .

[Cho97] CHOW, M. M. 97, Optimized Geometry Compression for Real-Time Rendering. In *Proceedings of IEEE Visualization*, pp. 347-354.

[Der95] DEERING,M.1995,Geometry Compression, In *Proc. SIGGRAPH'95 (1995), pp.* 13-20.

[DYH04] DEOK-SOO, K., YOUNGSONG, C., AND HYUN, K. 2004, Normal vector compression of 3D mesh model based on clustering and relative indexing. *Future Generation Computer Systems*, Vol .20 pp. 1241-50.

[Wei06] WEISSTEIN, E. W. 2006, Platonic Solid, From MathWorld-- A Wolfram Web Resource http://mathworld.wolfram.com/PlatonicSolid.html

[GVSS00] GUSKOV, I., VIDIMCE, K., SWELDENS, W., AND SCHRODER, P. 2000, Normal Meshes, In *Proc. SIGGRAPH00 (2000)* , pp. 95-102.

[RRH*02] RANDALL, D. A., RINGLER, T. D., HEIKES, R. P., JONES, P., AND BAUMGARDNER, J. 2002, Climate Modeling With SphericalGeodesicGrids, *Computing in Science&Engineering*.

[THL*98] TAUBIN, G., HORN, W., LAZARUS, F., AND ROSSIGNAC, J. 1998. Geometry Coding and VRML. In *Proc. of the IEEE, Special issue on Multimedia Signal Processing*, 86(6) 1228-43

[TR98] TAUBIN, G. AND ROSSIGNAC, J. 98, Geometric Compression Through Topological Surgery, ACM Transactions on Graphics, Vol. 17, No.2, pp.84-115.

[WDS99] WOO, M, DAVIS, AND SHERIDAN, M. B.1999, OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2, 3rd edition.