A Comparison of Algorithms for Vertex Normal Computation

Shuangshuang Jin, Robert R. Lewis, David West

Washington State University

School of Electrical Engineering and Computer Science

Abstract　We investigate current vertex normal computation algorithms and evaluate their effectiveness at approximating analytically-computable (and thus comparable) normals for a variety of classes of model. We find that the most accurate algorithm depends on the class and that for some classes, none of the available algorithms are particularly good. We also compare the relative speeds of all algorithms.

1 Introduction

Vertex normals, which are normal vectors bound to the vertices of polygonal facets, are commonly used to render polygonal meshes as smoothly-shaded three-dimensional objects.

When the mesh has an underlying analytical implicit, explicit, or parametric representation, we can compute mesh vertex normals directly from that representation, but when such representations are not available, as in the case of depth scan or geographical data, an algorithmic approach based solely on the given mesh is necessary.

As we shall see, a number of researchers have proposed such algorithms. What we present here is a systematic survey of all of them on a variety of mesh sources. The sources we choose are all analytical in nature, so that we are able to compare the algorithmic with the analytical "exact" normals. By choosing a wide variety of data models, we believe that our results will extrapolate to non-analytical meshes. Also, by computing vertex normals for the same models with differing algorithms, we can evaluate their relative speed.

2 Vertex Normal Algorithms

Since the early 1970's, graphics researchers have produced several algorithms to compute vertex normals. These algorithms differ substantially from each other, but they all have in common the notion of weighting ad-

jacent face normals in some fashion. In this section, we will review each of them in chronological order.

2.1 The "Mean Weighted Equally" (MWE) Algorithm

The first vertex normal algorithm, which we will refer to as the "Mean Weighted Equally" (MWE) algorithm, was introduced by Henri Gouraud[1] in 1971:

$$\mathbf{N}_{\mathrm{MWE}} \parallel \sum_{i=1}^{n} \mathbf{N}_i \qquad (1)$$

where the summation is over all n faces incident to the vertex in question and $\mathbf{N}_i$ is the face normal (the normal to the plane containing the face) of the ith face. (We will use "is parallel to" ("$\parallel$") to make implicit the necessary yet trivial normalization step.) In this algorithm, the normal of each adjacent facet contributes equally to the vertex normal.

An extension of MWE which works with non-manifold surfaces but requires more time to do so is a three-pass algorithm introduced by Overveld and Wyvill[4] in 1997. As the surfaces we will deal with here are all manifolds or manifolds with edges, we will only note this in passing.

2.2 The "Mean Weighted by Angle" (MWA) Algorithm

Another algorithm for vertex normals, which we refer to as the "Mean Weighted by Angle" (MWA) algorithm, was proposed by Thürmer and Wüthrich[2] in 1998.

Unlike MWE, MWA also incorporates the geometric contribution of each facet, that is, considering the angle under which a facet is incident to the vertex. The MWA formula is:

$$\mathbf{N}_{\mathrm{MWA}} \parallel \sum_{i=1}^{n} \alpha_i \mathbf{N}_i \qquad (2)$$

where α_i is the angle between the two edge vectors $\mathbf{E}_i$ and $\mathbf{E}_{i+1}$ of the ith facet sharing the vertex.

For this algorithm and other algorithms presented here, $\sin \alpha_i$ can be quickly computed from the cross product of the edge vectors by the formula:

$$\sin \alpha = \frac{|\mathbf{E}_i \times \mathbf{E}_{i+1}|}{|\mathbf{E}_i| \, |\mathbf{E}_{i+1}|} \qquad (3)$$

with $\mathbf{E_{n+1}} \equiv \mathbf{E_1}$.

2.3 The "Mean Weighted by Sine and Edge Length Reciprocal" (MWSELR) Algorithm

MWA suggested assigning non-equal weights deriving from the geometry. In 1999, another non-equal weighted vertex normal algorithm, which we refer to as the "Mean Weighted by Sine and Edge Length Reciprocal" (MWSELR) was introduced by Max[3]. His formulation accounts for the differences in size of the facets surrounding the vertex by assigning larger weights for smaller facets, which he found helped to handle the cases when the facets surrounding a vertex differ greatly in length. The MWSELR formula is:

$$\mathbf{N}_{\mathrm{MWSELR}} \parallel \sum_{i=1}^{n} \frac{\mathbf{N}_i \sin \alpha_i}{|\mathbf{E}_i| \, |\mathbf{E}_{i+1}|} \qquad (4)$$

where $\mathbf{N}_i$, $\mathbf{E}_i$ and $\mathbf{E}_{i+1}$ are as in (2).

2.4 The "Mean Weighted by Areas of Adjacent Triangles" (MWAAT) Algorithm

Max[3] also presented several other vertex normal algorithms besides MWSELR, with the intent of comparison. We refer to the first of these as as "Mean Weighted by Areas of Adjacent Triangles" (MWAAT). This algorithm incorporates the area of the triangle formed by the two edges of each facet (whether the facet is triangular or not) incident on the vertex:

$$\mathbf{N}_{\text{MWAAT}} \parallel \sum_{i=1}^{n} \mathbf{N}_i \, |\mathbf{E}_i| \, |\mathbf{E}_{i+1}| \sin \alpha_i \qquad (5)$$
$$= \sum_{i=1}^{n} \mathbf{N}_i \, |\mathbf{E}_i \times \mathbf{E}_{i+1}|$$

where $\mathbf{N}_i$, $\mathbf{E}_i$, $\mathbf{E}_{i+1}$, and α_i are as in (2).

2.5 The "Mean Weighted by Edge Length Reciprocals" (MWELR) Algorithm

We refer to another vertex normal algorithm appearing in Max[3] as the "Mean Weighted by Edge Length Reciprocals" (MWELR) algorithm. This modifies the MWSELR weights by omitting the $\sin \alpha_i$ factors:

$$\mathbf{N}_{\text{MWELR}} \parallel \sum_{i=1}^{n} \frac{\mathbf{N}_i}{|\mathbf{E}_i| |\mathbf{E}_{i+1}|} \qquad (6)$$

where $\mathbf{N}_i$, $\mathbf{E}_i$, and $\mathbf{E}_{i+1}$ are as in (2).

2.6 The "Mean Weighted by Square Root of Edge Length Reciprocals" (MWRELR) Algorithm

The last vertex normal algorithm in Max[3] is what we refer to as the "Mean Weighted by Square Root of Edge

Length Reciprocals" (MWRELR) algorithm. It is similar to the MWELR, with the addition of a square root:

$$\mathbf{N}_{\text{MWRELR}} \parallel \sum_{i=1}^{n} \frac{\mathbf{N}_i}{\sqrt{|\mathbf{E}_i| |\mathbf{E}_{i+1}|}} \qquad (7)$$

where $\mathbf{N}_i$, $\mathbf{E}_i$, and $\mathbf{E}_{i+1}$ are as in (2).

2.7 Some Comments on Max's Algorithms

It is worth noting that Max[3] proposed the MWSELR, MWAAT, MWELR, and WMRELR algorithms with the aim of empirically comparing their effectiveness. To do so, he first constructed 1,000,000 surfaces of the form

$$z = Ax^2 + Bxy + Cy^2 + Dx^3 + Ex^2y + Fxy^2 + Gy^3 \quad (8)$$

with A, B, C, D, E, F, and G all uniformly distributed pseudorandom numbers in the interval $[-0.1, 0.1]$. These he took to be representative of the third order behavior of smooth surfaces. He translated the surface so that the candidate vertex Q lay at the origin and then rotated them so that the analytical normal pointed in the z direction. He then generated a set of vertices $\mathbf{V}_0$, $\mathbf{V}_1$, $\ldots$, $\mathbf{V}_{n-1}$ with uniformly distributed pseudorandom values in polar coordinates r and θ and converted them to a Cartesian (x, y, z) with z given by (8). The arccosine of the resulting z component of the algorithmic normal was the angular discrepancy.

For each sequence $\{\mathbf{V}_i\}$, he computed the normal according to each algorithm and measured the angular discrepancy between the two. Under these circumstances, comparing RMS values of the angular discrepancy over

his test ensemble, he found that the MWSELR algorithm was the most accurate, with RMS discrepancies between 1.5° and 3.0°, depending on the valence of Q. The other algorithms produced RMS discrepancies between 2.8° and 10.8°.

3 Comparing Vertex Normals

When analytical vertex normals are available, algorithmic vertex normals are not needed. Nevertheless, if we choose a selection of models with known analytical normals, we can use them to test the accuracy of our algorithms.

This is similar to Max[3]'s approach, except that we will choose models that more closely resemble specific application domains than the synthetic one he used (described in Section 2.7).

3.1 Choice of Models

In order to do our comparison between each algorithm for vertex normal computation, we selected several classes of model with analytically-computable vertex normals from three categories: trigonometrically-parameterized surfaces, height fields, and marching tetrahedra-tesselated isosurfaces. All of them have varying amounts of geometric and topological regularity.

For each model in each class, we chose low, medium, and high resolution (triangle count) instances.

We represent the meshes of all of these objects using the ASCII version of the OFF ("Object File Format")[5] format, which is easy to both write and parse, and which permits the use of the *Geomview*[5] geometry viewer to view results and capture images.

3.1.1 Trigonometrically-Parameterized Surfaces Trigonometrically-parameterized (hereafter, "TP") surfaces are defined by a parametric mapping that involves trigonometric functions. We chose spheres and tori as representatives of this class of model. Their representations are trivial.

3.1.2 Height Fields Height fields (hereafter, "HF") are based on analytical functions of the form $z = f(x, y)$. Terrain models in computer graphics generally take the form of HFs. An HF is often sampled on a two-dimensional array of altitude values at regular intervals (or "post spacings", as referred to by geographers[6]). They are often defined on a regular grid and a rectangular domain. Representatives of this class are:

Perlin Noise Height Fields A standard way to produce "random"-appearing HFs is to use a Perlin noise function[6]. The standard "vector" noise function generates random normal vectors at grid points (where z is taken to be 0) which may be bicubically interpolated to compute z away from the grid points. Scaling and dilating Perlin noise functions allow the user to produce a wide variety of surfaces which are smooth yet "random".

Height fields are often termed "non-parametric", although perhaps a better term would be "trivially parametric", as they are a really a subset of polynomially-defined parametric surfaces. We believe that results for this class of object may be taken to be representative of those for similar parametric surfaces such as Bézier, Hermite, B-Spline, etc.

Fractal Noise Height Fields   Terrain often displays a certain degree of self similarity over a range of scales. Perlin noise functions are not self-similar, but multiple dyadic dilations of them can be combined to produce what Musgrave in [6] refers to as "hybrid multifractals".

An important parameter characterizing such HFs is the "fractal increment", usually denoted by H, which determines the relative contribution of each dilated level to the final result. If H is zero, all levels contribute equally and the resulting function approximates random noise. As H increases, the contribution of each successivly higher dilation (i.e. higher frequencies) is suppressed by a factor of e^{-H}. Typically, $H = 1$ produces a fairly smooth function. Controlling H allows us to control the "roughness" of our synthetic terrain. For this reason, we constructed fractal noise HF models of decreasing self-similarity with $H = 0.9$, $H = 0.5$, and $H = 0.1$.

3.1.3 Marching Tetrahedra   Implicit function surfaces being defined as of the form $f(x, y, z) = 0$, it is trivial to note that HFs are a subset of implicit functions having

the form $f(x, y, z) \equiv z - g(x, y)$ for some g. Nevertheless, general implicit functions are harder to tesselate than HFs.

Marching cubes is a popular algorithm for tesselating iso-surfaces from implicit functions. It also works with discrete three-dimensional data[8][9]. However, in its original form it does not guarantee the surface to be topologically consistent between cells. (This is actually an aliasing effect[7].) Marching tetrahedra (hereafter, "MT") is a variation of marching cubes, which both overcomes this topological problem[10] and is easier to implement.

We have chosen several representative implicit functions to study: spheres, Perlin 3D noise functions, and Perlin 3D turbulence functions.

Spheres Tesselated by Marching Tetrahedra   Although spheres are rarely rendered with MT, doing so in this case and contrasting the results with the sphere TP of Section 3.1.1 permits a comparision of vertex normal computation for two differing tesselations of the same implicit object. The isosurface value is, of course, the radius (or, for a slight increase in speed, the squared radius) of the sphere.

Cosine Sum Tesselated by Marching Tetrahedra   A more conventional implicit function to tesselate by MT is the sum of cosines given by

$$f(x, y, z) = \cos(2\pi\, d\, x) + \cos(2\pi\, d\, y) + \cos(2\pi\, d\, z) \quad (9)$$

where d is a user-adjustable dilation. The result in this case resembles a mechanical part as might be produced by a mechanical CAD system.

3D Perlin Noise Function Tesselated by Marching Tetrahedra   The Perlin noise function may be defined in a space of arbitrary dimensionality, so it is quite straightforward to extend it to an implicit function in three dimensions and extract an isosurface using marching cubes.

Perlin-style Turbulence Tesselated by Marching Tetrahedra   As noted in Section 3.1.2, the Perlin noise function is not fractal, but as noted by Perlin in [6], a weighted sum of absolute values of such functions over a dyadic range of dilations can exhibit fractal behavior. The simplest model for this is called "turbulence" (although not really based on any kind of solution to the flow phenomenon of that name), and it has been successfully used to represent clouds, marble, smoke, and explosions. We include it here to represent isosurfaces of these as well as MRI and similar medical scan data.

3.1.4 The Test Ensemble   We have implemented all the algorithms listed in Section 2 in C. Tables 1 and 2 show our ensemble of test models.

3.2 Comparison Technique

Accuracy and speed are two important properties for vertex normal algorithms.

3.2.1 Accuracy   In our testing, given the algorithmic and analytical normals, we measure the angular discrepancy as

$$\theta = cos^{-1}\left(\mathbf{N}_{\text{analytical}} \cdot \mathbf{N}_{\text{algorithmic}}\right) \qquad (10)$$

which we express in degrees.

Max[3] chose to compare RMS values angular discrepancy for the various algorithms, but we believe that these do not provide the best indication possible of how the results differ. Instead, to evaluate the accuracy of each vertex normal algorithm, we have generated cumulative histograms, displaying the angular discrepancy (in degrees) between the computed vertex normal and the actual one.

For each model, we start with a set of analytical ("true") normals. Then, for each algorithm, we generate a set of algorithmic normals. We compute the angular discrepancy for each vertex in the model and sort them by increasing value. When combined with a fraction of the vertices having that discrepancy or less (a linear scaling of the sorted array index), we have a cumulative histogram of the results.

The resulting graph is easy to interpret visually, showing as it does for a given discrepancy the fraction of vertex normals that are off by that value or less. The value of the discrepancy at which the fraction is 0.5 is the median. The fraction for the largest discrepancy will be the 1.0. A comparison of the analytical data with itself would be a constant 1.0.
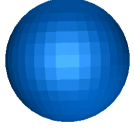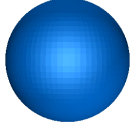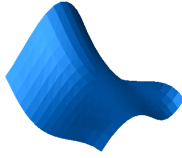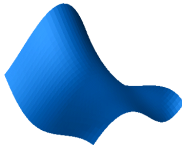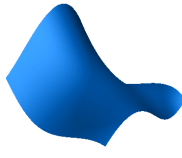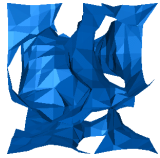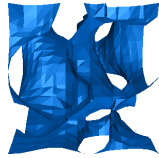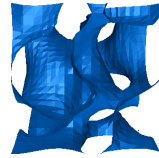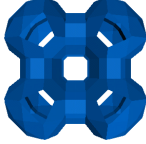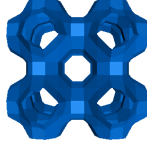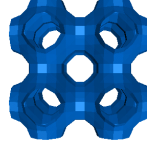
Model	low resolution	medium resolution	high resolution
sphere TP	320 triangles	1440 triangles	6080 triangles
torus TP	400 triangles	1600 triangles	4800 triangles
Perlin HF	450 triangles	1922 triangles	7938 triangles
sphere MT	960 triangles	2136 triangles	4176 triangles
Perlin MT	1501 triangles	3758 triangles	7136 triangles
cosine sum MT	2016 triangles	5088 triangles	8267 triangles

Table 1 Test Ensemble (non-fractal members).
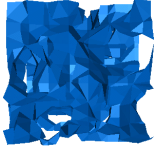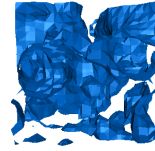
model	H	low resolution	medium resolution	high resolution
fractal HF	0.1	450 triangles	1922 triangles	7838 triangles
	0.5	450 triangles	1922 triangles	7838 triangles
	0.9	450 triangles	1922 triangles	7838 triangles
turbulence MT	–	172 triangles	2088 triangles	10438 triangles

Table 2 Test Ensemble (fractal members). H is the fractal increment[6].

Unlike RMS values, the cumulative histogram shows outliers, and it is possible to distinguish techniques that produce vertex normals that differ wildly in accuracy from those that are consistent and, presumably, more robust.

3.2.2 Speed   Speed is another criterion by which we may evaluate each vertex normal algorithm. It was straightforward to instrument our code to compute the time required per vertex solely to compute normals for each algorithm. Owing to the relatively short time needed and the 1 μsec resolution of our system CPU clock, it was necessary to perform multiple iterations to get high-resolution per-call time values.

4 Results

After testing with all our models, we obtained the data for the comparison of accuracy and speed between each algorithm for vertex normal computation.

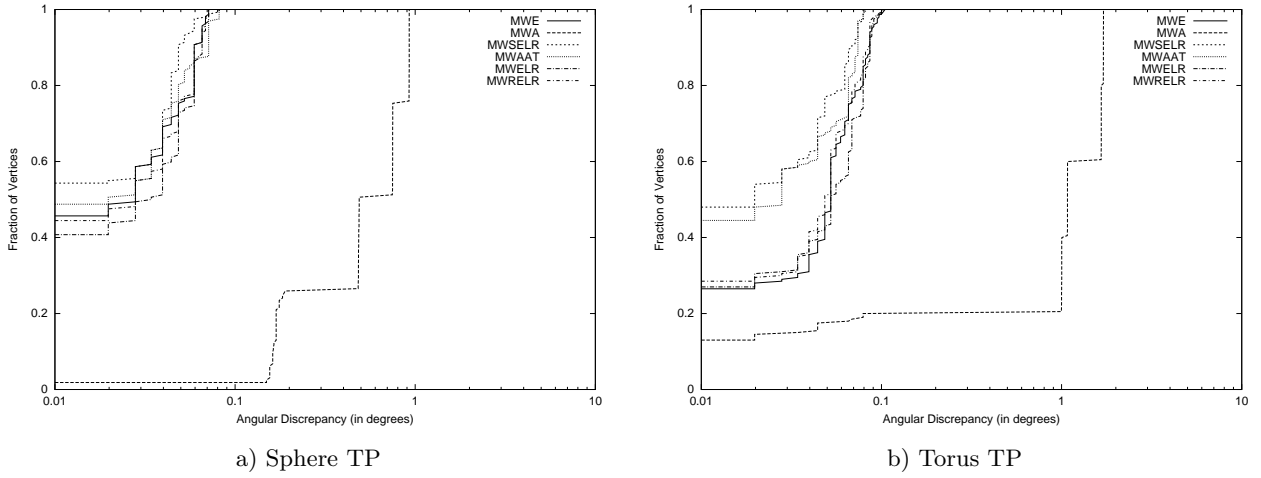**Fig. 1** Cumulative Histograms of Angular Discrepancies for Models Tesselated by Trigonometric Parameterization. All of these are at low resolution.

4.1 Accuracy

The most accurate algorithm depended on the nature of the model, which we will discuss class-by-class here. Table 3 summarizes these results.

4.1.1 Trigonometric Parameterization Models   Figure 1 shows the cumulative histogram results for TP models. For space reasons, we show here only the low-resolution results. Increasing resolution tended to improve the overall accuracy, as we should expect, but not the ranking of algorithms.

In the both histograms, we can see that MWSELR has the highest accuracy, although most of the others are quite close. The only exception is MWA, but even then the discrepancies there are no more than $1°$-$2°$.

4.1.2 Height Field Models   Figure 2 shows the cumulative histogram results for HF models. We show only the low-resolution data for the same reason given in Section 4.1.1.

The histograms indicate that MWAAT works best for the Perlin noise HF, but that MWA and MWE are almost as good. MWELR and MWSELR are noticeably worse.

MWAAT also works best for fractal HFs, as shown in Figures 2b, 2c, and 2d, followed by MWA. MWELR and MWSELR are, again, the worst. Variation of the fractal increment H makes little difference in either absolute or relative accuracy. Note the necessary change of scale of the fractal models: Approximately 8% of the vertices in these models had discrepancies of $10°$ or more.

4.1.3 Marching Tetrahedra Models   Unlike TP and HF models, the high resolution results are shown for the MT models in Figure 3, as lower resolutions exhibited very high discrepancies ($\sim 40°$ or more) that were obviously the results of spatial aliasing.

a) Perlin Noise HF

b) Fractal Noise HF (H = 0.1)

c) Fractal Noise HF (H = 0.5)

d) Fractal Noise HF (H = 0.9)
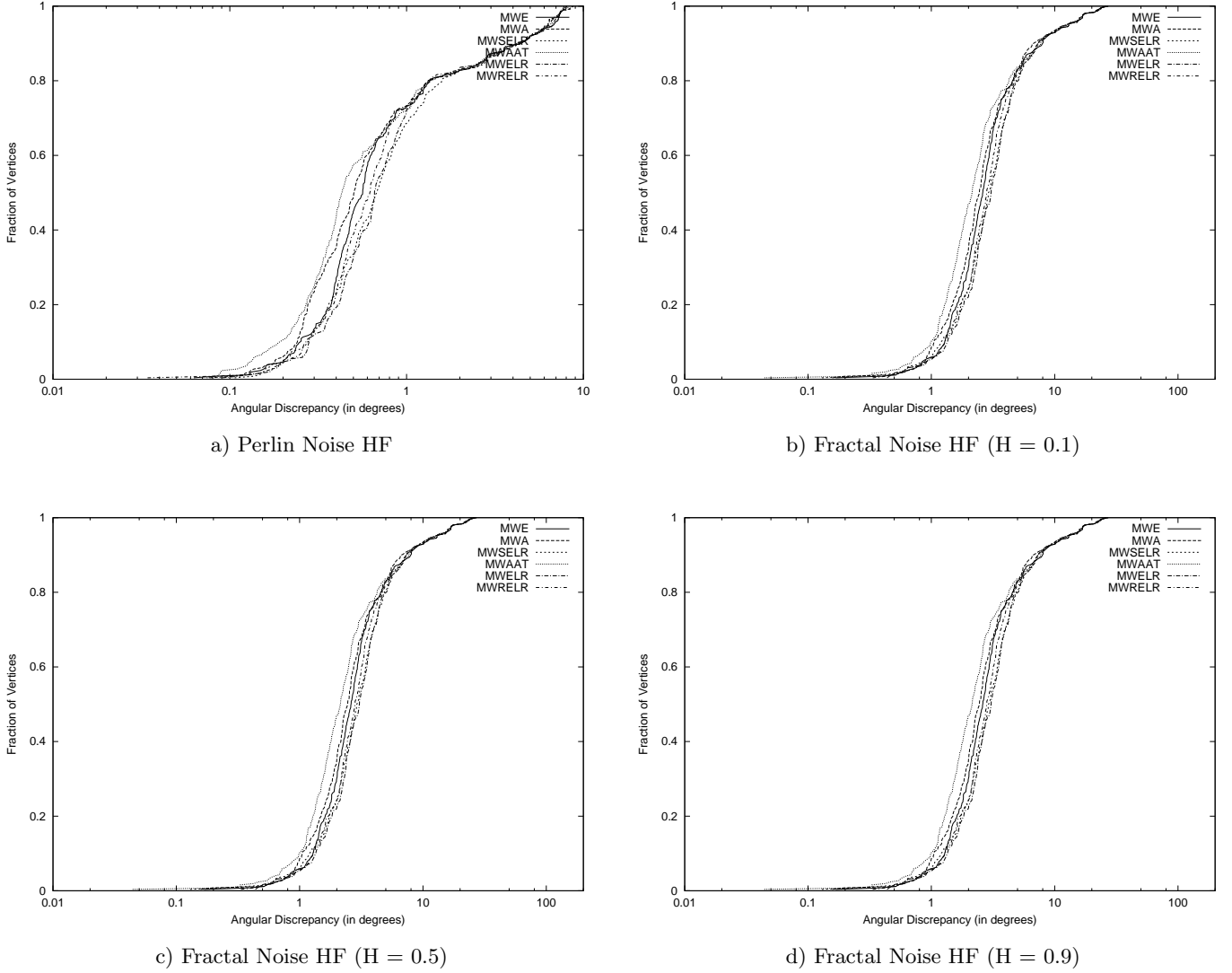
Fig. 2 Cumulative Histograms of Angular Discrepancies for Models Tesselated by Height Fields. All of these are at low resolution.

This aliasing, which takes place between the grid sampling rate and the intrinsic spatial frequency spectrum of the function being sampled produces adverse results for all MT models, but the most pronounced effects are visible in the histograms for the turbulence model. We can see this in Figure 4. Going from low resolution to high resolution produces a notable shift in the discrepancies.

For the sphere tesselated by MT, Figure 3a shows that the best algorithms are MWA, MWELR, and MWSELR, although MWA has a wider dispersion in accuracy. MWAAT is clearly worse than the rest.

For the cosine sum tesselated by MT, Figure 3b shows slight advantages for MWA, followed by MWE, while MWSELR and MWELR are the worst.

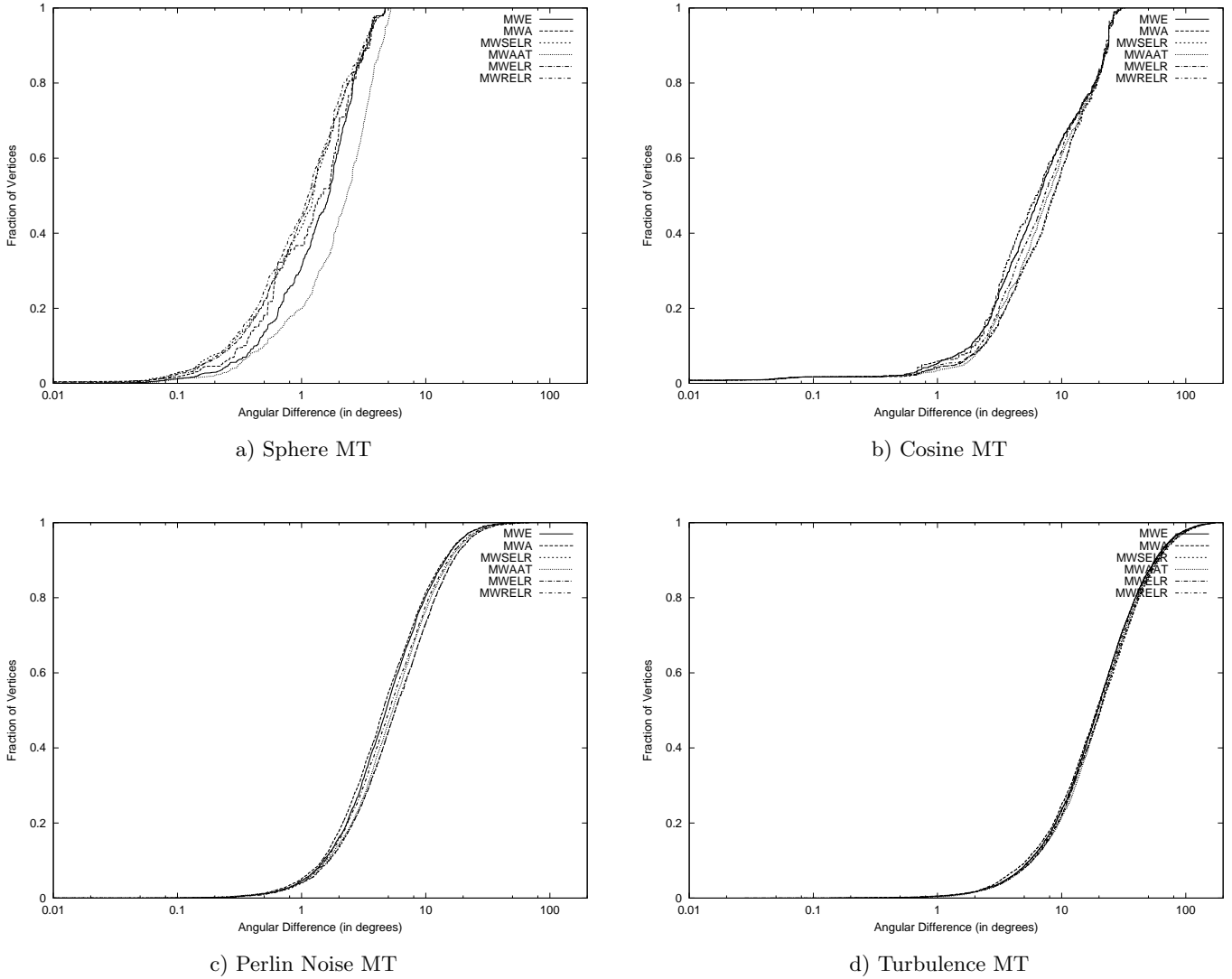a) Sphere MT

b) Cosine MT

c) Perlin Noise MT

d) Turbulence MT

Fig. 3 Cumulative Histograms of Angular Discrepancies for Models Tesselated by Marching Tetrahedra. All of these are at high resolution.

Differences are less pronounced for the Perlin noise function tesselated by MT, as shown in Figure 3c, but MWA still maintains a slight advantage over the rest, while MWAAT has a slight disadvantage.

Finally, there is very little difference in accuracy for the turbulence function tesselated by MT, as shown in Figure 3d, except that MWAAT is slightly worse than the rest.

4.2 Speed

As we mentioned in Section 3, we instrumented our code to compute the speed of each algorithm.

Figure 5 shows the mean CPU time per call for each algorithm, as measured on a Dell 1.0GHz Intel Pentium III machine with 256MB of memory. We can see that the MWE algorithm always cost the shortest time to compute the vertex normals, MWA is the second, and all
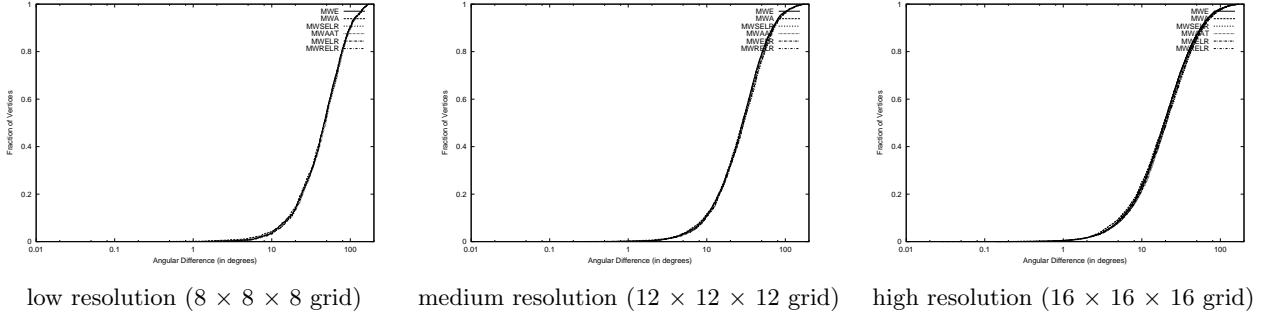
low resolution ($8 \times 8 \times 8$ grid)   medium resolution ($12 \times 12 \times 12$ grid)   high resolution ($16 \times 16 \times 16$ grid)

Fig. 4 Effects of Aliasing on the Accuracy of the Turbulence Model Tesselated by Marching Tetrahedra.

	MWE	MWA	MWSELR	MWAAT	MWELR	MWRELR
sphere TP		×	√			
torus TP		×	√			
Perlin HF		√	×	√	×	
fractal HF		√	×	√	×	
sphere MT		√	√	×	√	
cosine MT	√	√	×		×	
Perlin MT		√		×		
turbulence MT				√		

Table 3 Summary of Accuracy Results. A "√" indicates an algorithm that performed well. A "×" indicates an algorithm that performed badly.
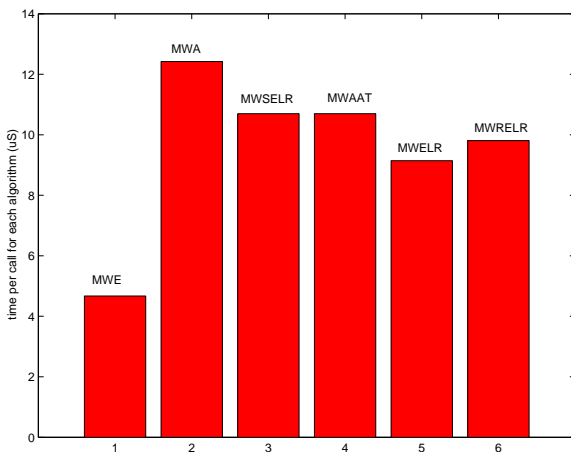

Fig. 5 Summary of Speed Results.

the other algorithms left are roughly equal. This result is reasonable, all the vertex normals are calculated based on weighted sums of face normals, and MWE does the fewest operations to compute the weights.

Overall, the speed results are all close enough that most users can probably choose the most accurate algorithm without regard to speed.

We note in passing that different implementations of these algorithms may change the times-per-call somewhat, but the relative speed of each algorithm should not be affected.

5 Conclusions and Future Work

We have evaluated the results of applying six vertex normal algorithms to ten different models at three different resolutions. Relatively speaking, except for trigonometrically-parameterized surfaces, MWA is always either the best or among the best. If speed is of a concern, however, MWE holds up remarkably well. As resolutions increase, the differences between these algorithms diminish.

In an absolute sense, however, none of these algorithms does particularly well for marching tetrahedra, with median discrepancies in the 5° to 20° range, even at the highest resolution. Even then, they are still quite large. For now, all that we can recommend is to increase the spatial sampling frequency.

In the future, however, this suggests a need to develop a new vertex normal algorithm more suitable for marching tetrahedra. We are particularly interested in looking at subdivision surface algorithms to provide this.

References

1. Gouraud, H., *Continuous Shading of Curved Surfaces,* IEEE Trans. on Computers, **C-20(6)**, (June, 1971) pps. 623–629.

2. Thurmer, G., Wuthrich, C., *Computing Vertex Normals from Polygonal Facets,* Journal of Graphics Tools, **3(1)**, (1998) pps. 43–46.

3. Max, N., *Weights for Computing Vertex Normals from Facet Normals,* Journal of Graphics Tools, **4(2)** (1999), pps. 1–6.

4. Overveld, C., Wyvill, B., *Phong Normal Interpolation Revisited,* ACM Trans. on Graphics, **16(4)** (October 1997) pps. 379–419.

5. Phillips, M., *Geomview Manual* http://www.geomview.org (The Geometry Center, November 2000).

6. Ebert, D., et al. *Texturing and Modeling A Procedural Approach,* 2nd. ed., (Academic Press, 1998).

7. Fournier, A., *private communication.*

8. Lorenson, W., Cline, H. *Marching Cubes: A High Resolution 3D Surface Construction Algorithm,* Computer Graphics, **21(4)**, (1987) pps. 163-169.

9. Wyvill, G., McPheeters, C., Wyvill, B. *Data Struct For Soft Objects,* The Visual Computer, **2(4)**, (1988) pps. 227-234.

10. Treece, G.M., Prager, R.W., Gee, A.H., *Regularised Marching Tetrahedra: Improved Iso-Surface Extraction*, Cambridge University (September 1998).