University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

Gradient Vector Estimation and Vertex Normal Computation

Research Report

Tomáš Jirka, Václav Skala

Technical Report No. DCSE/TR-2002-08
October, 2002

Distribution: public

Gradient Vector Estimation and Vertex Normal Computation

Tomáš Jirka, Václav Skala

Abstract

In this document a comparison of three methods of volumetric data gradient vector estimation (one of which was proposed by the authors) and five methods for triangle mesh vertex normal computation will be described and compared. All the methods can be used for regular as well as irregular meshes. The tests were focused primarily on the accuracy. However, a comparison of the temporal requirements of individual methods is included as well.

Copies of this report are available on
`http://www.kiv.zcu.cz/publications/`
or by surface mail on request sent to the following address:

> University of West Bohemia in Pilsen
> Department of Computer Science and Engineering
> Univerzitni 8
> 30614 Pilsen
> Czech Republic

Gradient Vector Estimation

1.   Introduction

In the following paragraphs, three methods for the scalar irregularly distributed volumetric data gradient vector (see Fig. 1) estimation will be presented. First, it is the 4D linear regression method using linear approximation function as proposed in [4]. Then its extension consisting in using quadratic regression function developed with the aim to reach higher estimation accuracy will be described as a second method. Third, it will be the approach based on generalization of the finite differences method presented in [3]. The performance of all the three methods will be examined from different points of view, compared and evaluated.
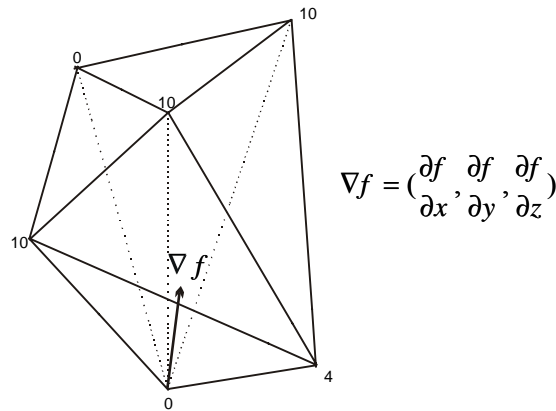
Figure 1: Illustration of a gradient vector in a vertex of an irregular scalar field, upon which a tetrahedral mesh has been constructed.

2.   Theoretical Background

In the following paragraphs, the principles of 4D linear regression method for gradient estimation will be described and the approach utilizing quadratic approximation function for the linear regression will be proposed. Then the 3D version of the approach generalizing the finite differences methods will be outlined.

2.1 4D Linear Regression using Linear Approximation Function

This method for gradient estimation from regular as well as irregular volumetric data proposed in [4] tries to find a 4D regression hyper plane $f(x,y,z) \approx A \cdot x + B \cdot y + C \cdot z + D$ with minimal error. The error function is represented as the summed squares of the difference between the original values in the interpolated vertices and the values that the solution of the hyper plane equation would give in these points. Mathematically:

$$E(A,B,C,D) = \sum_{k=0}^{n} w_k (A \cdot x_k + B \cdot y_k + C \cdot z_k + D - f_k)^2 \, , \qquad (1)$$

1

where x_k, y_k and z_k are the coordinates of the vertices involved in the approximation (the computed vertex being considered as the origin of the coordinate system) and f_k are the values in these points. A, B and C make up the vertex gradient that we search for and D is the filtered value in the computed vertex. The w_k symbol represents the weighting function, which should be spherically symmetric and monotonically decreasing as the distance from the origin (i.e. the computed vertex) grows.

To minimize the error function E from (1), its partial derivatives along A, B, C and D must be equal to zero:

$$\frac{\partial E}{\partial A} = 2 \cdot \sum_k w_k \cdot (A \cdot x_k + B \cdot y_k + C \cdot z_k + D - f_k) \cdot x_k = 0,$$

$$\frac{\partial E}{\partial B} = 2 \cdot \sum_k w_k \cdot (A \cdot x_k + B \cdot y_k + C \cdot z_k + D - f_k) \cdot y_k = 0,$$

$$\frac{\partial E}{\partial C} = 2 \cdot \sum_k w_k \cdot (A \cdot x_k + B \cdot y_k + C \cdot z_k + D - f_k) \cdot z_k = 0,$$

$$\frac{\partial E}{\partial D} = 2 \cdot \sum_k w_k \cdot (A \cdot x_k + B \cdot y_k + C \cdot z_k + D - f_k) = 0.$$

This system of simultaneous linear equations can be rewritten in a matrix notation in the following way:

$$\begin{bmatrix} \sum w_k x_k^2 & \sum w_k x_k y_k & \sum w_k x_k z_k & \sum w_k x_k \\ \sum w_k x_k y_k & \sum w_k y_k^2 & \sum w_k y_k z_k & \sum w_k y_k \\ \sum w_k x_k z_k & \sum w_k y_k z_k & \sum w_k z_k^2 & \sum w_k z_k \\ \sum w_k x_k & \sum w_k y_k & \sum w_k z_k & \sum w_k \end{bmatrix} \cdot \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} \sum w_k f_k x_k \\ \sum w_k f_k y_k \\ \sum w_k f_k z_k \\ \sum w_k f_k \end{bmatrix}. \tag{2}$$

Solving the system for A, B, C and D gives the hyper plane normal vector $\vec{n}(A,B,C)$, which is considered to be the estimation of the gradient $\nabla f = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z})$.

2.2 4D Linear Regression using Quadratic Approximation Function

In order to reach higher accuracy of estimated gradient vectors, it is necessary to apply a nonlinear approximation function. In our approach we use a general quadratic function of the following form

$$g(x,y,z) = [x,y,z,1] \cdot \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ 0 & A_{22} & A_{23} & A_{24} \\ 0 & 0 & A_{33} & A_{34} \\ 0 & 0 & 0 & A_{44} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

instead of the original linear function $f(x,y,z) \approx A \cdot x + B \cdot y + C \cdot z + D$. For the further description, the non-matrix notation will be more illustrative:

2

$$g(x, y, z) = A_{11}x^2 + A_{12}xy + A_{13}xz + A_{14}x + A_{22}y^2 + A_{23}yz + A_{24}y + A_{33}z^2 + A_{34}z + A_{44}.$$

Now we need to express the error function:

$$E(A_{11},...,A_{44}) = \sum_k w_k (A_{11}x^2 + A_{12}xy + A_{13}xz + A_{14}x + A_{22}y^2 + A_{23}yz + A_{24}y + A_{33}z^2 + A_{34}z + A_{44} - f_k)^2$$

and find the partial derivatives according to all the ten unknown parameters A_{11} through A_{44}:

$$\frac{\partial E}{\partial A_{11}} = 2 \cdot \sum_k w_k (A_{11}x_k^2 + A_{12}x_k y_k + A_{13}x_k z_k + A_{14}x_k + A_{22}y_k^2 + A_{23}y_k z_k + A_{24}y_k + A_{33}z_k^2 + A_{34}z_k + A_{44} - f_k) \cdot x_k^2,$$

$$\frac{\partial E}{\partial A_{12}} = 2 \cdot \sum_k w_k (A_{11}x_k^2 + A_{12}x_k y_k + A_{13}x_k z_k + A_{14}x_k + A_{22}y_k^2 + A_{23}y_k z_k + A_{24}y_k + A_{33}z_k^2 + A_{34}z_k + A_{44} - f_k) \cdot x_k y_k,$$

$$...$$

$$\frac{\partial E}{\partial A_{44}} = 2 \cdot \sum_k w_k (A_{11}x_k^2 + A_{12}x_k y_k + A_{13}x_k z_k + A_{14}x_k + A_{22}y_k^2 + A_{23}y_k z_k + A_{24}y_k + A_{33}z_k^2 + A_{34}z_k + A_{44} - f_k) \cdot 1.$$

These partial derivatives must be equal to zero, thus we get a 10 x 10 matrix describing the set of simultaneous equations, which are linear in respect to the A_{11} through A_{44} parameters. The gradient of the function can be described by the following formula:

$$\nabla g(x, y, z) = (\frac{\partial g}{\partial x}, \frac{\partial g}{\partial y}, \frac{\partial g}{\partial z}) = (2 \cdot A_{11}x + A_{12}y + A_{13}z + A_{14}; 2 \cdot A_{22}y + A_{12}x + A_{23}z + A_{24}; 2 \cdot A_{33}z + A_{13}x + A_{23}y + A_{34}).$$

As the active vertex is always translated to the origin of the coordinate system, the x, y and z coordinates equal to zero. Thus having computed the A_{11} through A_{44} parameters, the gradient vector can be obtained from the simple formula:

$$\nabla g(0,0,0) = (A_{14}, A_{24}, A_{34}).$$

2.3 Finite Difference Method

In [3] a generalization of the finite differences method is proposed. According to the authors, the finite differences methods can only be used for rectangular (although not necessarily regular) grids. These methods are based on the Taylor's series:

$$F(a + \Delta x) = F(a) + \Delta x \frac{dF(a)}{dx} + \frac{\Delta x^2}{2!} \frac{d^2 F(a)}{dx^2} + ... + \frac{\Delta x^{n-1}}{(n-1)!} \frac{d^{(n-1)} F(a)}{dx^{(n-1)}} + r_n(\xi), \qquad (3)$$

where $r_n(\xi)$ is a remainder term. Solving the equation obtained as $F(a + \Delta x) - F(a - \Delta x)$ with $n = 3$ for $dF(a)/dx$ then gives the central difference equation:

$$\frac{dF(a)}{dx} = \frac{F(a + \Delta x) - F(a - \Delta x)}{2\Delta x} + \frac{r_n(\xi) + r_n(\zeta)}{2\Delta x} \approx \frac{F(a + \Delta x) - F(a - \Delta x)}{2\Delta x}.$$

The equations above describe a 1D case, where we search for the gradient of function $F(x)$, which is supposed to be smooth. For computing gradients of a smooth function $S(x, y)$, the Taylor's series from (3) can be generalized to R^2:

$$S(a + \Delta x, b + \Delta y) = \sum_{i=0}^{n-1} \frac{1}{i!} (\Delta x \frac{\partial}{\partial x} + \Delta y \frac{\partial}{\partial y})^i S(a, b) + r_n(\xi, \zeta).$$ (4)

Similarly an R^3 extension can be used for estimating gradients of a scalar field defined by a smooth function $V(x, y, z)$. The Taylor's series, however, relies on partial derivatives that are aligned to curves parallel to the coordinate axis thus restricting the finite differences methods to be applicable for rectangular grids only. However, Δx may not be uniform as shown on Fig 2.
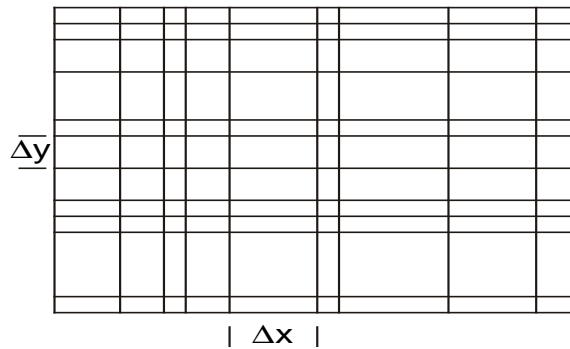

Figure 2: Irregular rectangular grid

To make the finite differences method suitable for irregularly distributed data as well, the authors generalize it using directional derivatives instead of partial derivatives. They first state the problem the following way:

- Let $\Re$ be an open, bounded, simply connected subset of the two-dimensional, Euclidean real space R^2.
- Let $S(x, y)$ be a smooth, real-valued function over $\Re$.
- Let $p \in \Re, p = (p_x, p_y)$ be a sample point of interest.
- Let $s = \{p_1, ..., p_j, ..., p_m \mid p_j \in \Re, p_j \neq p\}$ be a set of distinct sample points forming a neighborhood around p (see Fig 3). At least one point in s must not be collinear with the others.
- Find an approximation of the gradient of S at p.

Obviously, the authors describe the R^2 case. For scalar field gradient estimation the R^3 version would have to be used. We will, however, describe the R^2 case as described in [3] because it is easier to follow and the modification of the algorithm for use in R^3 is straightforward.
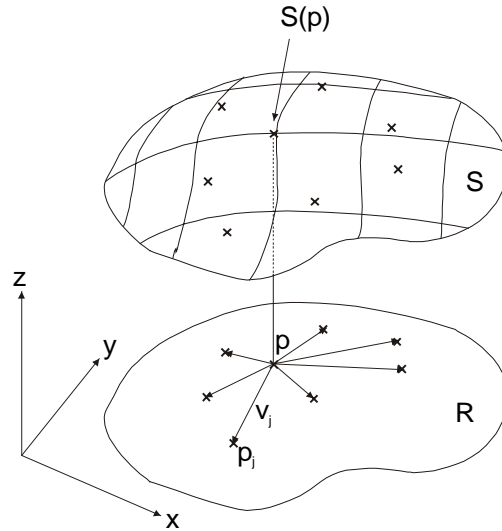
Figure 3: The neighborhood of the point of interest

The authors start to derive their method by expressing the difference in the value of S between p and p_j using equation (4) as

$$\Delta S = \Delta x S_x + \Delta y S_y + \ldots + r_n(\xi, \zeta),$$

with S_x and S_y denoting $\partial S(p_x, p_y)/\partial x$ and $\partial S(p_x, p_y)/\partial y$ respectively. They describe the directional vector $\vec{v}_j$ from p to some other point p_j, its magnitude v_j and the unit vector in the same direction $\hat{v}_j$:

$$\vec{v}_j = \hat{i}\Delta x + \hat{j}\Delta y = p_j - p, \qquad v_j = \left|\vec{v}_j\right|, \qquad \hat{v}_j = \hat{i}\frac{\Delta x}{v_j} + \hat{j}\frac{\Delta y}{v_j} = \frac{\vec{v}_j}{v_j},$$

$\hat{i}, \hat{j}, \vec{k}$ being unit vectors in the directions of the x, y and z Cartesian axis, respectively. The ratio between the value change ΔS between two points p and p_j and their distance v_j can be expressed as

$$\frac{\Delta S}{v_j} = \frac{\Delta x}{v_j}S_x + \frac{\Delta y}{v_j}S_y + \ldots + \frac{r_n(\xi,\zeta)}{v_j} = \hat{v}_j \cdot \nabla S + \ldots + \frac{r_n(\xi,\zeta)}{v_j} \qquad (5)$$

and for $v_j \to 0$ and $n \to \infty$, equation (5) leads to the directional derivative of S in the direction $\vec{v}_j$:

$$\lim_{v_j \to 0, n \to \infty} \frac{\Delta S}{v_j} \equiv \frac{dS}{d\vec{v}_j} = \hat{v}_j \cdot \nabla S,$$

which shows the desired relationship between the directional derivative and the gradient of S. Knowing this relation, the approximation can then be done for $n = 2$:

$$\frac{\Delta S}{v_j} = \hat{v}_j \cdot \nabla S + \frac{r_2(\xi,\zeta)}{v_j} \approx \hat{v}_j \cdot \nabla S \qquad (6)$$

5

where

$$\frac{r_2(\xi,\zeta)}{v_j} = \frac{1}{2!v_j}(\Delta x^2 S_{xx} + 2\Delta x \Delta y S_{xy} + \Delta y^2 S_{yy}) \tag{7}$$

and S_{xx} stands for $\partial^2 S(\xi,\zeta)/\partial x^2$, S_{xy} for $\partial^2 S(\xi,\zeta)/\partial x \partial y$ and S_{yy} for $\partial^2 S(\xi,\zeta)/\partial y^2$, where $(\xi,\zeta) \in R$. Since $v_j \approx \Delta x \approx \Delta y$, the truncation error represented by equation (7) is $O(\Delta x^2)$ as well as in case of the central difference method.

Applying (6) to each point $p_j \in s$ results in an over-determined system of simultaneous equations, whose solution gives an estimation of $\partial S/\partial x$ and $\partial S/\partial y$. Marking the matrix containing the sample vectors' x and y components $v_{x,j}$ and $v_{y,j}$ as V, the vector of the partial derivative approximations as t and the vector of the slope estimates $\Delta S_j / v_j$ as σ, the system of equations can be rewritten as

$$\sigma = V \cdot t \tag{8}$$

$$\begin{bmatrix} \Delta S_1/v_1 \\ \Delta S_2/v_2 \\ \vdots \\ \Delta S_m/v_m \end{bmatrix} = \begin{bmatrix} \widehat{v}_{x,1} & \widehat{v}_{y,1} \\ \widehat{v}_{x,2} & \widehat{v}_{y,2} \\ \vdots & \vdots \\ \widehat{v}_{x,m} & \widehat{v}_{y,m} \end{bmatrix} \cdot \begin{bmatrix} \partial S/\partial x \\ \partial S/\partial y \end{bmatrix}.$$

Then, t can be found from (8) in the following way:

$$Vt = \sigma$$
$$V^t V t = V^t \sigma$$
$$(V^t V)^{-1} V^t V t = (V^t V)^{-1} V^t \sigma$$
$$t = (V^t V)^{-1} V^t \sigma$$

$(V^t V)^{-1}$ always exists because of the assumption of non-collinearity. The authors also note that "$(V^t V)$ is a 2 x 2 matrix and therefore finding its inverse is not computationally demanding."

3.   Implementation & Testing

All the above-described approaches were implemented within one application for the purpose of comparison. All the implementations were done in the Borland Delphi environment and the tests were run on a system with the Intel Pentium III @ 448MHz CPU and 1024MB RAM.

3.1 Testing Data

The application requires the volumetric data to be structured to constitute a tetrahedra mesh whether regular or not. The tetrahedra structure only serves to determine each vertex's surrounding, which should be involved in the computation, and is not necessary for the approach itself.

Our tests have been performed on meshes constructed upon the sets of 5000, 10000, 15000 and 20000 vertices using Delaunay approach, maximal number of tetrahedra and minimal number of tetrahedra. To show, how the mesh structure influences the results, estimations from meshes constructed upon clusters of vertices have also been tested.

To be able to make comparisons and evaluations we need the exact gradient vectors. Thus it is necessary to use some known function to generate the scalar field values. However, the estimation methods are meant to search for gradients of general data with no common characteristic known in advance (e.g. empirically measured data). Therefore there was no definite criterion for choosing the testing function. The strategy was chosen to test the methods on some simple function (i.e. f_1 – see below), then on some simple function (i.e. f_2) with higher order then the order of the approximation functions used in the estimation method and eventually on a relatively complex function (i.e. f_3), which would be rather distant to those approximation functions thus at least partially substituting the empirically obtained data:

- $f_1(x, y, z) = x^2 + y^2 + z^2$,
- $f_2(x, y, z) = x^3 + y^3 + z^3$,
- $f_3(x, y, z) = 3 \cdot x^4 \cdot y^2 \cdot e^z + 8 \cdot y + 5 \cdot x \cdot e^y + 16 \cdot z^5$.

These functions will be referenced as f_1, f_2 and f_3.

3.2 Error Measurement

For the testing purposes, the boundary vertices of the mesh were filtered out from the statistics. The main measure of accuracy was the average error angle computed the following way. For each vertex, the angle in degrees between the exact gradient and the estimated one was found. Their arithmetic average then determined the average error:

$$E_\alpha = (\sum_{i=0}^{N-1} \alpha_i) / N .$$

(9)

The secondary measure was the error of the vector length. In this case, the error computation consisted of the following steps. First, the difference in the length of both the vectors was enumerated for each mesh vertex. The ratio of this distance and the length of the exact gradient vector in that vertex was then expressed. Finally, the arithmetic average of such ratios was computed:

$$E_l = \sum_{i=0}^{N-1} \frac{\left\| |\vec{u}_i| - |\vec{v}_i| \right\|}{|\vec{v}_i|} / N$$

(10)

where E is the average error, u_i and v_i are the estimated and the exact gradient vectors respectively and N is the number of evaluated vertices.

Instead of (9) and (10), it would also be possible to measure the error as the length of the error vector (11), which would be the distance between the end points of the exact and the computed vector. Symbolically written:

$$E = \frac{\sum_{i=0}^{N-1} \frac{|\vec{u}_i - \vec{v}_i|}{|\vec{v}_i|}}{N} ,$$

(11)

7

where E is again the average error, u_i and v_i are the estimated and the exact gradient vectors respectively and N is the number of evaluated vertices. However, some applications are only interested in the error angle and do not require the gradient length to be correct. For this reason, we have used the first evaluation procedure applying equations (9) and (10).

The following picture should make the geometrical meaning of individual error expressions clear:
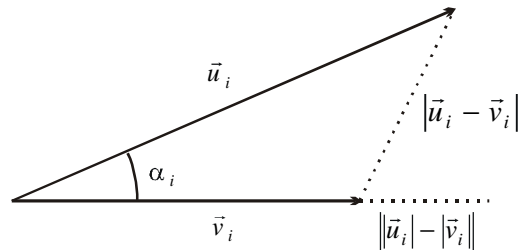

Figure 4: Error measurements illustration

4. Results

In the following paragraphs, where the implemented three approaches will be examined from several points of view and their results compared, *LR-Lin* denotes the method that uses linear regression based on linear approximation function, *LR-Nonlin* stands for linear regression based on quadratic approximation function and *FDM* represents the method based on generalization of the finite differences method described in [3].

## 4.1 Accuracy Statistics – Varying Sampling Functions

A good basic notion of how the methods perform can be acquired just by testing them on a mesh of only 1000 vertices using the Delaunay criteria. Table 1 shows that, from the accuracy point of view, LR-Nonlin performs best (marked by darker shading). The exception was using it for estimating linear function (plane) gradients, where the FDM reached the best results. The reason is that LR-Nonlin attempts to approximate sample values in the vertices by a quadratic function. Therefore, when applied on a simple plane, it performs worse than the linearly oriented methods. In all the other cases, however, LR-Nonlin reached the most accurate results while FDM the worst, LR-Lin being in the middle. The linear sample function (plane) will not be included in further testing as the results balance on the edge of computational numerical precision and are not of high importance, for in practice, linear sampling function can hardly be expected.

Error Angle in Degrees	LR-Lin	LR-Nonlin	FDM
x (plane)	1.31E-15	1.29E-12	9.30E-16
$x^2 + y^2 + z^2$ (sphere)	2.55	0.45	3.89
$x^3 + y^3 + z^3$	4.66	0.92	6.32
$3 x^4 y^2 e^z + 8 y + 5 x e^y + 16 z^5$	3.36	1.54	3.86

Tab. 1: Tests on Delaunay tetrahedra mesh with 1000 vertices.

4.2 Accuracy Statistics – Varying Data Density

The following three graphs (one graph for each of the three sample functions) show how the estimation results improve when supplying more information by using a denser mesh. Although the graphs look quite similar, it is necessary to note that the scale on the *y* axis differs to keep the graphs legible.
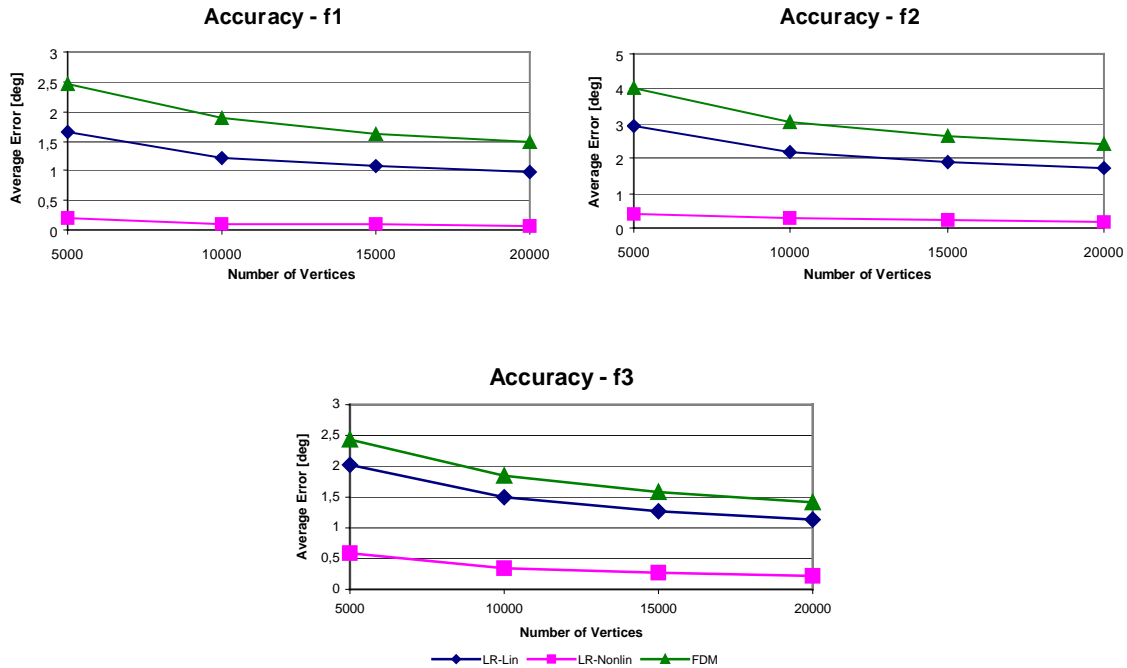

Graph 1: Estimation accuracy for f_1, f_2 and f_3

Each value in the graphs has been obtained as an average of three measurements, each using a different tetrahedra mesh (i.e. Delaunay mesh and meshes with maximal and minimal number of tetrahedra). It is obvious from the graphs above that the denser sampling is available the more accurate gradient estimation can be expected. As well, the graphs confirm that (except for the linear sample functions) the LR-Nonlin method returns best results. On the other hand, comparing these three graphs to each other reveals an interesting fact that more complex sampling function does not imply lower estimation accuracy. Regardless of the estimation method used, the results of gradient computation are more precise for f_3 than for f_2. In fact, in case of FDM, the results for f_3 are even slightly better than those for f_1. These, maybe a little surprising, results are promising for practice, where the samples will probably not approximate neat simple functions.

4.3 Accuracy Statistics – Varying Vertex Distribution

In this section the influence of the structure of the input mesh on the accuracy of the estimation will be demonstrated. For this purpose, a pair of Delaunay tetrahedra meshes was generated upon 5000 and 10000 vertices distributed in clusters. Graph 2 shows the average estimation error for all three methods on both the uniform as well as the clustered vertices, meshed by the Delaunay method. Although the graph was meant primarily to illustrate the influence of the mesh structure on the results, we can also notice that the LR-Nonlin method

performed best again with significant advance to LR-Lin, let alone FDM. Since the graphs for different sample functions f1, f2 and f3 resembled each other, only one of them will be presented here. For easier orientation, the marks at the ends of the lines are rectangular for the estimation from the mesh with uniformly distributed vertices and triangular for the mesh on clusters.

Accuracy (f1) - Uniform vs. Cluster Vertex Distribution
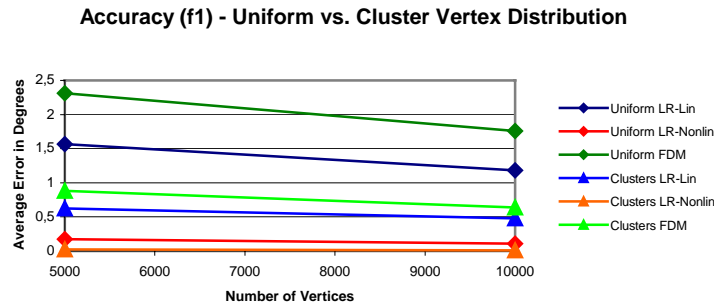
Graph 2: The average error of the estimation on meshes with clustered and uniform vertex distribution.

At the first glance it might seem rather surprising, but the graph shows that the gradient estimation applied on the tetrahedra mesh generated upon the clusters of vertices gives better results than the mesh with the same number of vertices distributed uniformly. Closer analysis shows that such results in fact correspond to what was described in the previous section. When the vertices are grouped in clusters, some of them are positioned at locations, where the clusters are connected to each other. In these locations, big errors can be expected as the surrounding of these vertices consists of small tetrahedra in the direction of the cluster, on the boundaries of which the vertex resides, and large tetrahedra in the other direction, where the cluster is connected to the other clusters. This unbalanced distribution of information around these vertices causes the failure of all the gradient estimation methods. The estimated gradient vectors are strongly inaccurate in such locations. Yet, this situation applies to only a small percent of vertices. The majority is located inside the clusters, where their density is higher than in case of uniform distribution, which leads to better estimations. The bigger errors are compensated and the overall average error is lower for the clustered data than for the uniformly distributed vertices, where some error peaks appear as well, especially in the locations close to the surface.

The distribution of the error within the data is illustrated by graph 3. To keep the graphs understandable, files with only 1000 vertices have been used. The vertices, where the error has its peaks, are easily recognizable and the situation in some of these "problematic" vertices has been analyzed visually. This analysis was the ground to the explanations described above.

10
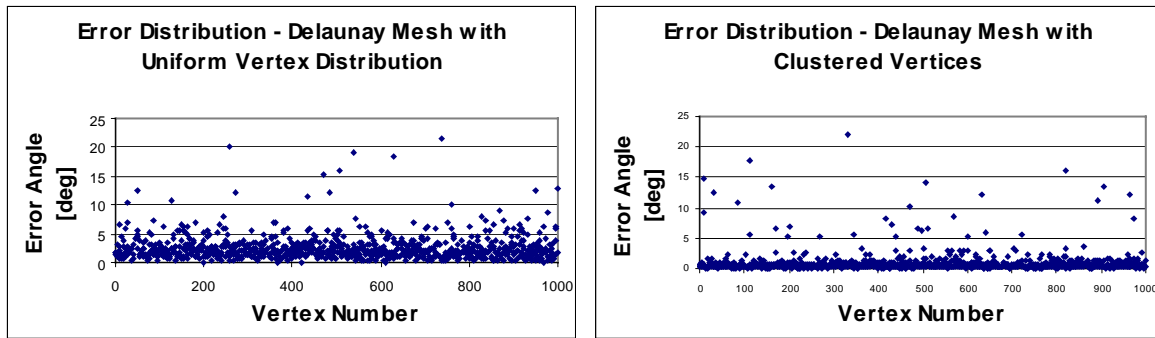
Graph 3: The distribution of the error for uniformly distributed and clustered vertices.

4.4 Accuracy Statistics – Vector Length

So far, we have only been concerned in measuring the error angle between the exact and the estimated gradient vectors. It is however necessary to realize that, unlike for example surface normal vector, gradient is determined by its length as well. Therefore the methods were also tested from this point of view and the results have been summarized in Graph 4.
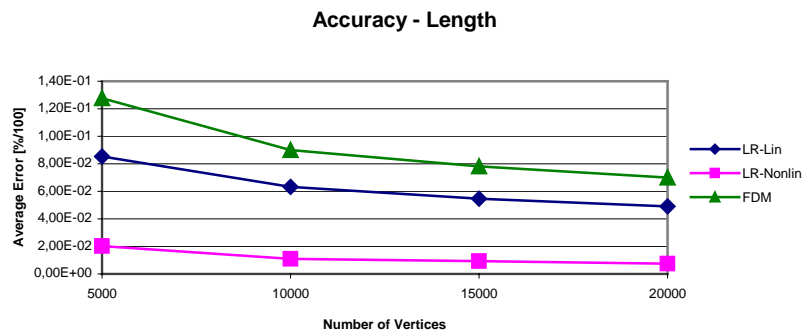

Graph 4: The average error of the estimated gradients' length in percents of the exact vector length.

Each value in this graph was obtained as the arithmetic average of nine measurements combining the usage of three sampling functions on the three types of meshes described above. We can see that the LR-nonlin method gives the most precise results being far ahead of the other two. The LR-lin estimations were approximately four times less accurate and those of FDM more than six times. Using different mesh types did not lead to significant differences here. Concerning the sampling function, results for f_1 were a little better than results for f_2, f_3.

4.5 Speed Statistics

Although we have adopted the accuracy of individual methods as the main criterion according to which the methods should be judged, the time requirements of the computations are usually much too important to bee ignored. To be consistent, the temporal needs of the three approaches were measured on the same data files as in section 4.2 and the results are summarized in graph 5. Each value in the graph has thus been obtained as an average of three

11

measurements on different tetrahedra meshes. When compared to each other, the results of these three measurements differed just slightly i.e. at most around 10 percent in one direction or the other.
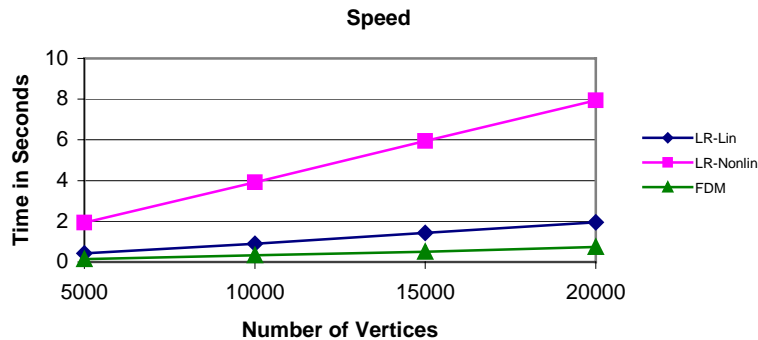
Graph 5: Comparison of the methods according to their time requirements

Apparently, the more accurate the method is, the more time it needs to do the computation. While producing the best results, the linear regression method utilizing quadratic regression function required about four times more time than the method using linear approximation function and up to ten times more than the finite difference method. Yet, all three methods have linear time complexity $O(N)$, where N is the number of vertices.

5. Conclusion & Recommendations

In the paragraphs above, the behavior of the gradient estimation methods has been examined in terms of accuracy while varying several aspects as the mesh internal structure, density and the sample function defining the scalar field's values. From this point of view, the linear regression method with quadratic approximation function turned out to be the best, providing significantly better results than the other two. On the other hand, it was the most time demanding one. Although the time complexity is linear for all these methods, the growth of the time requirements is steeper for the method using quadratic function. Thus, an imaginary ratio accuracy/speed is roughly the same for all the methods. Therefore there are no straightforward recommendations in this case. For each application a decision must first be made, what the primary criterion will be, whether accuracy, speed or some kind of trade-off between both. As stated at the beginning, accuracy was the main criterion for this study. From this point of view, the method based on linear regression with nonlinear approximation function presented in previous sections performed best.

Vertex Normal Computation

1.   Introduction

In the following paragraphs, five methods of vertex normal computation will be described, tested and compared. The application of the first three approaches is restricted to triangle meshes. The other two do not require any mesh and thus can be applied on more general data.

2.   Theoretical Background

Generally, there are two main approaches for vertex normal computation. The first is based on combining the normal vectors of the facets that share the vertex being computed. This is the case of the first two methods described below in *2.1* and *2.2*. The fourth and fifth methods in *2.4* and *2.5*, on the other hand, try to approximate the surrounding of the vertex in question by a surface of certain order. The normal of such surface is than considered to be the approximation of the vertex normal of the original surface. Method three, described in *2.3*, is a combination of both these approaches. It uses a normal of an approximation surface, which is, however, computed as a weighted sum of surrounding facets' normals. Since this method's theoretical background resembles methods *2.4* and *2.5*, the procedure of the computation is similar to *2.1* and *2.2*. Thus, it is half way from the first approach to the other one and therefore it will be described in section *2.3*.

2.1 Gouraud

This method has been described in [1] by Gouroud, who suggests computing normals in the vertices of a triangle mesh as the average of the normal vectors of the facets that share the vertex being computed. In his approach, all the facets, which contribute to the vertex normal computation, are weighted equally. Mathematically,

$$N \; = \; \frac{\sum\limits_{i=1}^{n} N_i}{\left| \sum\limits_{i=1}^{n} N_i \right|} \, ,$$

where N is the normal in the vertex and N_i are the normals of the triangles that share it.

2.2 Thurmer

Thurmer and Wuthrich [6] aim to improve the accuracy of the vertex normal computation method suggested by Gouraud. They claim that the results of the Gouraud's method strongly depend on the topology of the mesh around the vertex being processed. In other words, if we start with certain triangle mesh, choose one of its vertices and compute the normal vector there, then if we restructure the surrounding of this vertex and then we re-

compute the vertex normal, the result should ideally be the same. By reconstructing the vertex surrounding we mean using a different triangulation upon the same set of vertices. This can be reached for example by adding new vertices on the edges of some of the existing triangles thus dividing these triangles into two or more smaller ones while keeping the overall shape of the surface untouched – see Figure 1.
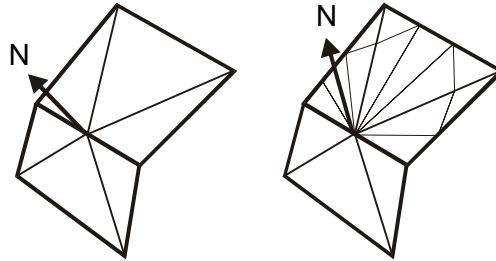

Figure 1: Illustration of the normal's dependency on the meshing

The authors of this article try to reach the independency on the mesh structure through weighting the contribution of each facet's normal by the size of the angle of the facet's edges incident to the computed vertex. This can be expressed as

$$N \;=\; \frac{\sum\limits_{i=1}^{n} \alpha_i N_i}{\left| \sum\limits_{i=1}^{n} \alpha_i N_i \right|} \;,$$

where α_i is the angle between the two edges of the facet that lead to the vertex. Thurmer and Wutrich state that although other measures (e.g. the areas of the triangles) might also be used, the incident angle is the only one to obtain correct results.

Remark:

In both the methods described above, there is another aspect to be pointed out. The purpose of using these methods is to make the surface shaded smoothly and to get a rid of those edges between adjacent polygons that do not occur on the original object and that are caused by the approximation. However, the rendered body may also contain some real edges. Smoothing out these edges would rather decrease than increase the realism of the rendered image. Unfortunately, these edges are usually not marked within the input data, which is why the rendering algorithm can hardly recognize them. The method that can sometimes help to overcome this drawback is based on defining certain "decision angle". If the angle between two adjacent polygons is less sharp than the decision angle, the edge is considered to be an unwanted one and is smoothed. Otherwise, the edge probably represents a real edge on the rendered object and should be displayed sharply.

2.3 Using Facet Normals Weighted by the Area

In this method, the normal $\vec{n}$ is computed as a normalized weighted sum of the unit length normal vectors $\vec{n}_i$ of the facets that belong to the cycle around the vertex being processed similarly as in Gouraud's [1] and Thurmer's [6] approach. This time, each facet's

14

area S_i is considered as the weighting function for the corresponding normal. Thus, the larger facets have higher influence than the smaller ones. Symbolically expressed:

$$\vec{N} = \frac{\sum_m \vec{n}_i S_i}{\sum_m S_i} \text{, where } \vec{n} = \frac{\vec{N}}{|\vec{N}|}.$$

Remark:

As mentioned above, this method is a combination of two main approaches in a way. On one hand, the actual computation resembles the *Gouraud's* or *Thurmer's* methods, because the vertex is found as a weighted average of the surrounding triangles' normals. On the other hand, the theoretical background of this approach rather resembles the idea underlying the following two methods, *Linear Regression* and the *Finite Differences Method,* where the normal of an approximating surface is viewed as the estimation of the vertex normal. In other words, the neighbors of the vertex where the normal is to be computed are found first. This set of vertices is then interleaved by an average surface, whose normal is then considered to be co-linear with the surface normal in the computed vertex.

According to [7], one way to express the normal of the approximating surface is to make an area weighted average of the surrounding triangles' normals, which is the case of the method above.

Nevertheless, there are more ways to find the average surface normal. The following two approaches estimate the gradient of the function $f(x, y)$ that approximates the values in the neighborhood of the computed vertex as the first step. The elements of this gradient vector $\nabla f = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$ represent partial derivatives of *f* along the *x* and *y* directions respectively. The vectors $\vec{u} = \vec{i} + \vec{k} \cdot \frac{\partial f}{\partial x}$ and $\vec{v} = \vec{j} + \vec{k} \cdot \frac{\partial f}{\partial y}$ are thus the tangent vectors of *f* for $y = const.$ and $x = const.$ Subsequently, the cross product $\vec{n} = \vec{u} \times \vec{v}$ represents the normal vector of the approximating function in the location of the vertex in question. This vector $\vec{n}$ is than viewed as the estimate of the original surface normal (see figure 2).