PERGAMON

Computers & Geosciences 29 (2003) 741–751

COMPUTERS & GEOSCIENCES

www.elsevier.com/locate/cageo

Hash functions and triangular mesh reconstruction ☆

Jan Hrádek*, Martin Kuchař, Václav Skala

Department of Computer Science and Engineering, University of West Bohemia in Plzeň, Univerzitní 8, Box 314, 306 14 Plzeň, Czech Republic

Received 24 May 2002; received in revised form 7 November 2002; accepted 16 December 2002

Abstract

Some applications use data formats (e.g. STL file format), where a set of triangles is used to represent the surface of a 3D object and it is necessary to reconstruct the triangular mesh with adjacency information. It is a lengthy process for large data sets as the time complexity of this process is $O(N \log N)$, where N is number of triangles. Triangular mesh reconstruction is a general problem and relevant algorithms can be used in GIS and DTM systems as well as in CAD/CAM systems. Many algorithms rely on space subdivision techniques while hash functions offer a more effective solution to the reconstruction problem. Hash data structures are widely used throughout the field of computer science. The hash table can be used to speed up the process of triangular mesh reconstruction but the speed strongly depends on hash function properties. Nevertheless the design or selection of the hash function for data sets with unknown properties is a serious problem. This paper describes a new hash function, presents the properties obtained for large data sets, and discusses validity of the reconstructed surface. Experimental results proved theoretical considerations and advantages of hash function use for mesh reconstruction.

Keywords: Algorithm complexity; 3D surface; Terrain modelling; GIS systems; Data structures

1. Introduction

There are several problems related to the properties of the triangular mesh representation that describes a surface of an object. Sometimes, the surface is represented just as a set of triangles without any other information and the STL file format, which is used for data exchanges, is a typical example of this situation.

The STL format is simple because all object surfaces are represented by polygons. More precisely, polygonal facets represent the surface and the surface is represented as a set of triangles as described in Foley et al. (1997). Each polygon (triangle) contains information on its normal and coordinates of all its vertices, see Fig. 1. The STL format is often used in CAD applications, e.g. in the rapid prototyping (RP) systems.

The main drawback of the STL format is that it does not contain any information on the structure and topology of the surface. Also the coordinates of each vertex shared by the given triangle are generally stored in the STL format several times. If the triangular mesh with the adjacency information is required, information on the neighbours of a triangle or triangles, shared by the given vertex, is needed.

One possibility is to reconstruct the triangular mesh from the given set of triangles. It enables us to compute normal vectors in vertices for Gouraud or Phong shading, triangular mesh reduction (see e.g. Franc, 2000) and compute slices for the efficient and rapid prototyping of systems. The main problem of triangular mesh reconstruction is to find the adjacent triangles for each vertex. The reconstruction of the triangular mesh from the given set of triangles is a critical operation because of its complexity, especially for large data sets.

☆ Supported by the Ministry of Education of the Czech Republic-project MSM 235200005.

*Corresponding author. Tel.: +420-37-7491-212; fax: +420-37-7491-212.

E-mail addresses: jhradek@kiv.zcu.cz (J. Hrádek), kuchy@kiv.zcu.cz (M. Kuchař), skala@kiv.zcu.cz (V. Skala).

0098-3004/03/$ - see front matter
doi:10.1016/S0098-3004(03)00037-2

```
solid
  facet normal 0.00 0.00 1.00
    outer loop
      vertex 1.00 2.00 0.00
      vertex -1.00 2.00 0.00
      vertex 0.00 -1.00 0.00
    endloop
  endfacet
  facet … end facet
endsolid
```

Fig. 1. An example of STL file.

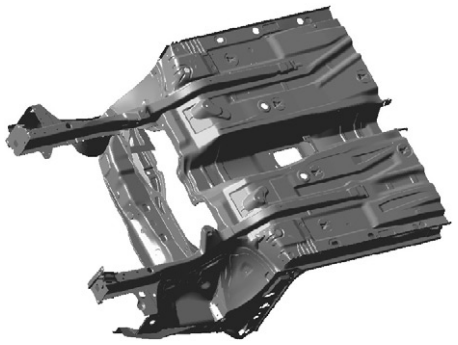

Fig. 2. Part of an auto chassis. Data set: A4_unterbau1.stl, Data courtesy of Skoda-Auto.

To be able to reconstruct the triangular mesh, it is necessary to read all the vertices, sort them according to the one coordinate, remove duplicates (the same vertex is stored several times if it is shared by more triangles) and create the regular triangular mesh with information on neighbouring triangles, etc. In the triangular mesh a vertex is shared by the triangles and the coordinates are stored only once. This is a process of $O(N \log N)$ complexity, where N is the number of triangles. This is a time consuming process if the considered objects are represented by 10^6–10^7 triangles. Typical data sets to be processed are presented in Fig. 2 with typical CAD data and in Fig. 3 with typical GIS data (triangular mesh).

In order to speedup the mesh reconstruction, space subdivision techniques were mostly used. Nevertheless, the results obtained depended heavily on the data sets used, especially, the way in which the vertices are scattered in the space.

Some approaches aimed at overcoming the complexity of using the hash function have been published recently and have resulted in an algorithm with expected $O(Np)$ complexity, in general (Formaggia, 2001; Glassner, 1994; Kuchar, 2000; Wolfson and Rigoutsos, 1997; Yao, 1985), where p is an average cluster length, see later.

The hash function transforms a value (or a key value) into an index (or an address) that points to the hash table (or a file) where the pointer to a vertex is stored. Fundamental requirements on the hash function are computational simplicity and collision elimination, i.e. indices should be different for different values.

It is impossible to avoid collisions in practice. Therefore, different values with equal indices to the hash table are stored in clusters. The expected number of collisions depends also on the *load factor f* (Korfhage and Gibbs, 1987). The load factor f is a ratio of the number of values stored and the total length of the hash table, $f \leqslant 0.5$ is the recommended value.

The basic idea of this approach is to obtain O(1) expected complexity for a query "find all triangles having the given vertex coordinates equal to …". This type of query is to be answered for all the vertices of the given set of triangles. It can be seen that an efficient solution is critical for large triangular meshes.

In the following sections a new hash function for triangular mesh reconstruction will be described, theoretical and experimental properties compared and non-trivial applications will be illustrated.

2. Hash function design and properties

2.1. Recent solution

Hash functions have several properties (Knuth, 1998) and the most important of these are:

- to be able to use the cells of the hash table as much as possible,
- maximal and average cluster lengths should be as low as possible (cluster is usually implemented as a list of primitives for which the hash function gives the same index for different values),
- the hash function should be simple in order to obtain fast evaluation.

The hash function for triangular mesh reconstruction was introduced recently by Glassner (1994) as

$$\begin{aligned}\text{Index} = ((\text{int})\ ((\alpha((\text{int})(\text{fabs}(X)Q))/Q \\ + \beta((\text{int})(\text{fabs}(Y)Q))/Q \\ + \gamma((\text{int})(\text{fabs}(Z)Q))/Q) \\ \times \text{SIZE}))\%\text{SIZE},\end{aligned} \quad (1)$$

where (int) is the conversion to integer—the fractional part of the float is removed (in current implementation 4–bytes unsigned integers are used), α, β, γ are coefficients of the hash function (originally 3, 5 and 7 were used), fabs is an absolute value, Q defines the insensitivity—number of valid decimal digits, e.g. for 3 decimal digits set $Q = 1000.0$, % represents modulo

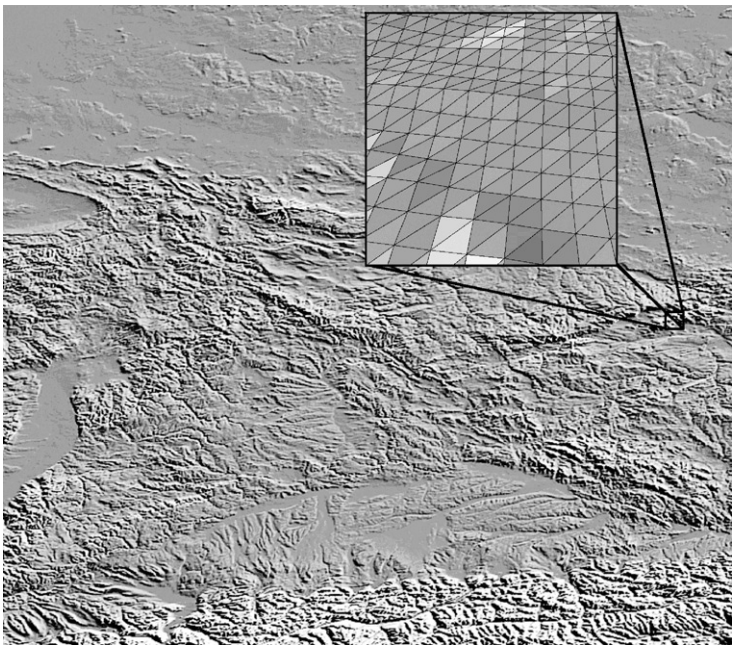

Fig. 3. Earth's surface reconstructed from satellite data. In rectangle is detail of mesh.

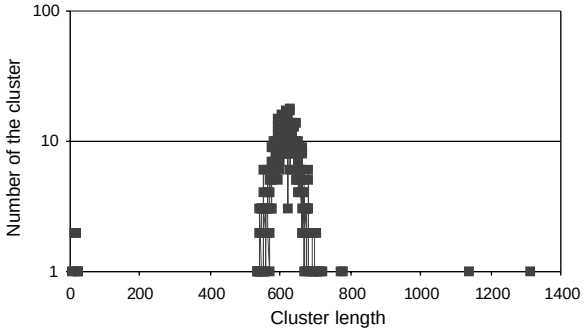

Fig. 4. Histogram of cluster lengths for recent hash function. Coefficients used: $Q = 10^3$, $\alpha = 3$, $\beta = 5$, $\gamma = 7$.

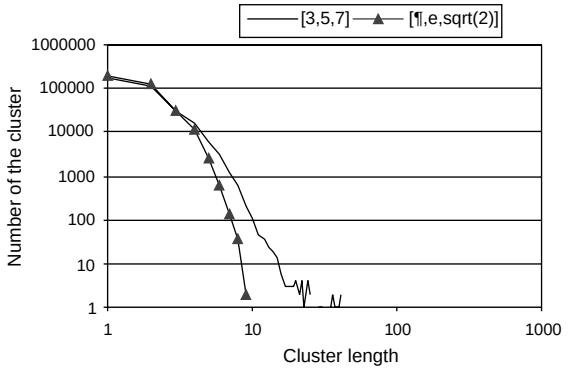

Fig. 5. Histogram of cluster lengths for recent hash function. Coefficients used: case 1: $Q = 10^7$, $\alpha = 3$, $\beta = 5$, $\gamma = 7$; case 2: $Q = 10^7$, $\alpha = \pi$, $\beta = e$, $\gamma = \sqrt{2}$.

operation, SIZE is the size of the hash table that is determined as described later, but generally it is implemented as 2^k for the fast evaluation of the modulo operation, and X, Y, Z are coordinates of a vertex.

Experiments described later proved that setting $Q = 10^3$ (used by Glassner, 1994) was not satisfactory because very long clusters were generated. Better results were obtained for $Q = 10^7$. Nevertheless, the properties of this hash function strongly depend not only on the fractional part of the coordinates of the vertices, but also on the data sets used for experiments, see Figs. 4–6 with histograms of the cluster lengths for different types of data.

Due to very long clusters, see Fig. 4, the original hash function was analysed with the aim of finding a significant improvement.

2.2. Proposed solution

The analysis of the hash function proved that

- it is not reasonable to remove the fractional part from coordinate values, as it helps to distinguish the coordinates of the vertices better,
- it is necessary to remove all the hash function parameters that depend on the data set, or should be given by a user,

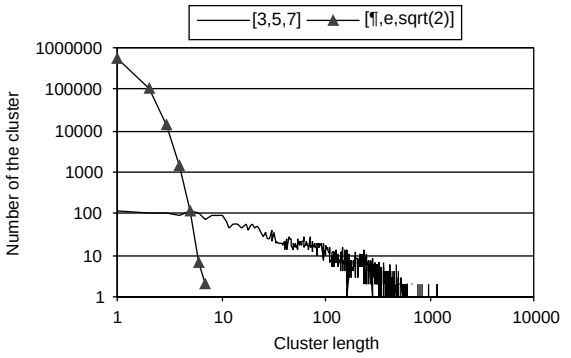

Fig. 6. Histogram of cluster lengths for recent hash function. Coefficients used: case 1: $Q = 10^7$, $\alpha = 3$, $\beta = 5$, $\gamma = 7$; case 2: $Q = 10^7$, $\alpha = \pi$, $\beta = e$, $\gamma = \sqrt{2}$.

- it is reasonable to use all available memory as much as possible in order to get a longer hash table, and therefore shorter clusters,
- the hash table should not be a static one—it should be dynamic according to the available memory, but generally the size of the hash table must be determined by some rules.

Considering the required properties, the hash function was defined as

$$\text{Index} = (\text{int})((\alpha X + \beta Y + \gamma Z)\ C + 0.5)\&T, \tag{2}$$

where X, Y, Z are vertex coordinates (see below), α, β, γ are coefficients of the hash function, C is a scaling coefficient set so that the full range of a 4–byte unsigned integer is used, i.e. range of the interval $\langle 0,\ 2^{32}-1 \rangle$ is fully used, $T+1$ is the table size ($T = 2^k - 1$), & represents a modulo that is implemented as a logical operation and with the DWORD type (for a fast computation), 0.5 is a constant that actually represents the rounding operation and according to our experiments gives a little better distribution of coordinates.

For simplicity we will assume that all coordinates x are from the $\langle 0,\ X_{\max} \rangle$ interval, similarly for coordinates y and z. Then we can compute maximal value $\xi_{\max}$ by the formula

$$\xi_{\max} = \alpha X_{\max} + \beta Y_{\max} + \gamma Z_{\max}. \tag{3}$$

Because we want to use the whole range of DWORD ($0-2^{32}-1$) we have to find an appropriate scaling coefficient C. The C coefficient is determined as:

$$C = \min\{C_1, C_2\}, \tag{4}$$

where

$$C_1 \xi_{\max} \leqslant 2^{32} - 1, \quad C_2 = 2^{32} - 2^k \tag{5}$$

because an overflow operation must be avoided and the whole size of the hash table should be used. It means that the whole interval $\langle 0,\ 2^{32}-1 \rangle$ is used before applying the operator modulo.

If the range of the vertex coordinates is known, then these coordinates can be converted to the interval $\langle 0,1 \rangle$ using the formula

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}. \tag{6}$$

This situation is rare because the range of vertex coordinates is generally not known. For an unknown range of coordinates the following formula can be used:

$$X' = \left(1 + \frac{X}{|X| + K}\right)\frac{1}{2}, \tag{7}$$

where K is a parameter which determines the non-linearity of this transformation, $K > 0$.

It transforms the interval $(-\infty, \infty)$ into the interval (0, 1) without the need to parse the whole data set in order to find the minimal and maximal values of the coordinates. This formula must be applied to Y and Z coordinates as well.

The proposed hash function has two parts. In the first part the coordinates are transformed to (0, 1) and in the second part the hash function is evaluated using those transformed values. The index to the hash table is therefore obtained.

2.3. Hash table length determination

The length of the hash table depends on the size of the data set we want to process. It is necessary to point out that there are approximately three times more vertices than triangles in the STL format.

The *load factor* f defined by Korfhage and Gibbs (1987) expresses an empirical relationship between the length of the hash table and the expected cluster length. If the *load factor* $f = 0.5$ is considered then the expected cluster length is about 2.5.

The required minimal hash table length T can be expressed as

$$T \geqslant \frac{N_v}{q_{\text{avg}}}\frac{1}{f} = N_T, \tag{8}$$

where N_T is the number of triangles in the STL file, $N_v = 3N_T$ is the number of vertices in the STL file, q_{avg} is the average number of triangles sharing the same vertex (approx. 6) *load factor–*$f = 0.5$ used[1] if the lower value f is used a better spread can be achieved.

In practice the hash table length T is chosen as a power of 2 in order to use the logical 'and' operator instead of the modulo operator because it is faster.

2.4. Evaluation of the hash functions properties

Experimental tests involved several types of hash functions with different parameters. The following

[1] $f \in (0.25,\ 0.5)$ due to rounding to power of 2.

criteria for the hash function quality assessment were used:

1. Average cluster length.
2. Maximum cluster length.
3. The search complexity criterion—total number of checked items in clusters while processing all the vertices in the data set.

It can be seen that a hash function with good properties should minimize values of those three criteria.

A hash function with better properties is one that provides fewer items in the clusters.

In order to compare the properties of the hash function given by Glassner (1994) (referred to here as recent) and a new proposed hash function the following relative criteria are used:

Average cluster length evaluation: Let us introduce the coefficient v as

$$v = \frac{\text{Average cluster length}_{\text{recent}}}{\text{Average cluster length}_{\text{proposed}}}. \tag{9}$$

Coefficient v represents the expected relative speed-up of the proposed approach (on average) for answering the query (it gives a pointer to the list of triangles): "Which triangles share the given vertex?".

Maximum cluster length evaluation: Let us define the coefficient η as

$$\eta = \frac{\text{Max. cluster length}_{\text{recent}}}{\text{Max. cluster length}_{\text{proposed}}}. \tag{10}$$

It presents the ratio of the maximal cluster lengths of both hash functions.

Search complexity criterion: This criterion is based on the following idea. If the cluster length is equal to i then we access this cluster i-times, if all the vertices are checked. Because the number of clusters with the length i is N_i, the whole number of accesses to these clusters is equal to iN_i.

There are three possible cases in referring a vertex from a cluster (after the index to the hash table has been determined):

1. *The cluster is empty*: The vertex is not stored in the data structure thus the cost of this operation is considered as $Q_1 = 0$.
2. *The cluster is not empty and the vertex is not stored in this cluster*: The whole cluster of the length i must be searched thus the cost of this operation is considered as $Q_2 = i$.
3. *The cluster is not empty and the vertex is stored in this cluster*: To find this vertex only one-half of the cluster needs to be searched on average. The cost of this operation is considered as $Q_3 = i/2$.

Then the third criterion can be expressed as

$$\begin{aligned} R &= \frac{1}{DV}\sum_i (Q_1 + Q_2 + Q_3) iN_i \\ &= \frac{1}{DV}\sum_i (0 + i + i/2) iN_i \\ &= \frac{1}{DV}\sum_i \frac{3}{2} i^2 N_i. \end{aligned} \tag{11}$$

Therefore

$$R = \frac{3}{2DV}\sum_{i=1}^{i_{\max}} i^2 N_i, \tag{12}$$

where $i_{\max}$ is the maximal cluster length, i the cluster length, N_i the number of clusters with the length i, and DV the number of vertices in the data set. This criterion is better for the evaluation of the hash function properties than the first two recently mentioned (average and maximum cluster length evaluations).

To compare the recent and proposed hash functions the following comparison ratio is used:

$$\delta = \frac{R_{\text{recent}}}{R_{\text{proposed}}}, \tag{13}$$

where R_{recent} is the value of the criterion for the recent hash function, R_{proposed} is the value of the criterion for the proposed hash function.

Fig. 11 presents this ratio for different data sets.

3. Experimental results

The hash functions have been tested on many data sets containing up to 10^7 vertices. Figs. 2 and 3 present some of the typical data sets used for the testing and evaluation of both methods (recent and proposed). The size of the data sets varied from 3×10^6 to 1.9×10^7 vertices.[2] It is necessary to point out, that the Central Europe.stl file was generated from the Digital Terrain System (GTOPO30) with regular triangular mesh (Fornous, 2000), while the A4_unterbau1.stl file was "real life" CAD data from the automobile industry. Tables 1 and 2 present the differences between the recent and proposed hash functions.

Experiments proved that the hash function is insensitive to the choice of the parameter K for the interval $\langle \pi, 200 \rangle$. The coefficient $K = 10$ was used for experimental results presented here. Table 2 presents the typical behaviour on some selected data sets. Figs. 9 and 10 presents the relation of the cluster length and the number of triangles.

[2] Jenkins (2001). Hash Function FAQ. http://burtleburtle.net/bob/hash/hashfaq.html

It can be seen that the maximal cluster length is lower than 10. It is a good result if compared to the recent solution, see Tables 1–3 and Figs. 4–6. Also the number of clusters decreases with cluster length and this is another good property of the proposed hash function. During all tests both (recent and proposed) hash functions used the same length of hash table for each data set.

Cluster length distribution of the recent hash function: Table 1 and Figs. 4–6 present behaviour of the recent hash function for different parameters and typical data sets. It can be seen that the maximal and average cluster lengths are not acceptable for practical use, because they cause high overheads during the search operation. Figs. 4–6 present typical histograms of cluster length for different parameters and data sets.

Cluster length distribution of the proposed hash function: Table 2 presents typical behaviour of the proposed hash function for the same data sets. Figs. 7 and 8 show typical histograms of cluster length for different parameters and data sets.

The above-presented experimental results are similar for all the data sets tested and prove that the proposed hash function has better and more stable behaviour than the recent approach.

Results for representative data sets: Our experiments proved that the proposed approach is better for all data sets used and Fig. 9 presents results for the both hash functions with the best-selected parameters.

Table 3 presents typical behaviour for a different choice of parameters α, β, γ. It can be seen that the proposed hash function is not very sensitive to the choice of those parameters. It shows the ratio for maximum and average cluster sizes for the recent and the proposed hash functions, and for two typical data sets.

Fig. 10 presents experimental results for both hash functions but with the best-known coefficients for different data sets. It can be seen that the proposed hash function is better in nearly all cases.

Fig. 11 presents the ratio of the criterion for the recent and the proposed hash functions for data sets used in

Table 1
Typical characteristics of recent hash function for selected STL data

File	Number of triangles	Number of vertices in STL	Number of unique vertices	α	β	γ	Q	Average cluster length	Maximal cluster length	Figure
				3	5	7	10^3	202.3	12 093	4
A4_unterbau1.stl	1 000 790	3 002 370	618 865	3	5	7	10^7	1.8	208	
				π	e	$\sqrt{2}$	10^7	1.6	9	5
Central Europe.stl 1	605 608	4 816 824	804 601	3	5	7	10^7	152.2	5 116	
				π	e	$\sqrt{2}$	10^7	1.2	7	6

Table 2
Typical characteristics of proposed hash function for selected STL data

File	Number of triangles	Number of vertices in STL	Number of unique vertices	α	β	γ	K	Average cluster length	Maximal cluster length	Figure
				3	5	7	10	1.3	7	
A4_unterbau1.stl	1 000 790	3 002 370	618 865	π	e	$\sqrt{2}$	10	1.3	7	7
				3	5	7	10	1.2	6	
Central Europe.stl	1 605 608	4 816 824	804 601	π	e	$\sqrt{2}$	10	1.2	7	8

Table 3
Comparison of average and maximal cluster lengths of recent and proposed hash functions

Coefficient case	File	Average cluster length		ν	Maximal cluster length		η
		Recent	Proposed		Recent	Proposed	
	A4_unterbau1.stl	1.8	1.3	1.4	208	7	29.7
1	Central Europe.stl	152.3	1.2	126.5	5116	6	852.7
	A4_unterbau1.stl	1.6	1.3	1.2	10	7	1.4
2	Central Europe.stl	1.2	1.2	1.0	6	7	0.9

Coefficients used: case 1: $\alpha = 3$, $\beta = 5$, $\gamma = 7$, $Q = 10^7$, $K = 10$; Case 2: $\alpha = \pi$, $\beta = e$, $\gamma = \sqrt{2}$, $Q = 10^7$, $K = 10$.

our experiments. It can be seen that the proposed hash function is better than the recent one, and also its stability was experimentally demonstrated. The average value of this ratio is 1.127.

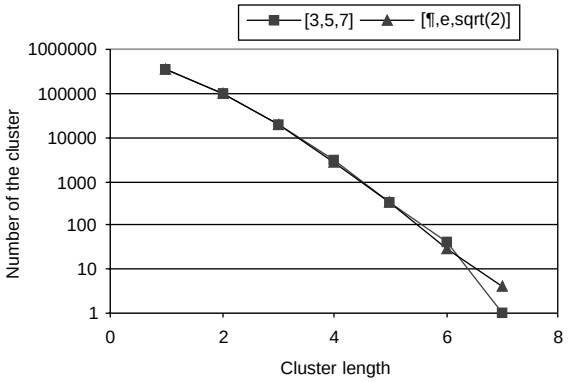

Fig. 7. Histogram of cluster lengths for proposed hash function. Coefficients used: case 1: $K = 10$, $\alpha = 3$, $\beta = 5$, $\gamma = 7$; case 2: $K = 10$, $\alpha = \pi$, $\beta = e$, $\gamma = \sqrt{2}$.

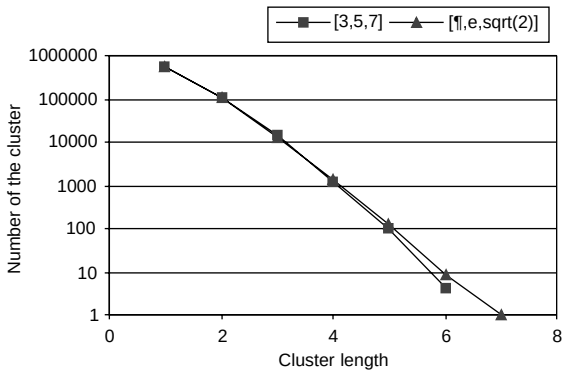

Fig. 8. Histogram of cluster lengths for proposed hash function. Data set: Coefficients used: case 1: $K = 10$, $\alpha = 3$, $\beta = 5$, $\gamma = 7$; case 2: $K = 10$, $\alpha = \pi$, $\beta = e$, $\gamma = \sqrt{2}$.

4. Hash function use for reconstruction

Triangular mesh reconstruction creates a triangular mesh from a set of individual triangles, i.e. it computes all neighbours for every triangle in the mesh and triangles that share the given vertex. As a vertex is stored several times in the original data set it is necessary to eliminate those duplicates, as each triangle in the original data set is processed. The elimination can be described by the algorithm in Fig. 12.

Fig. 13 is the structure of vertices and triangles along with the structure of hash table. Now it is necessary to take the final step in triangular mesh reconstruction and check the validity of the surface.

5. The validity of the reconstructed surface

While vertex duplicate, reduction is made "bucket-sorting" is used to create information about adjacency between triangles. The bucket sort takes advantage of the fact that each vertex coordinates are stored only once. For each vertex v in the triangular mesh a bucket is created, where the remaining two vertices u_i for each triangle that shares the vertex v are stored, see Fig. 14.

The vertices inside the bucket are sorted to obtain the vertices u_i which are in the same order as they appear around the vertex v. Then those two triangles whose vertices are beside each other in the bucket are determined as adjacent. The complexity of this part is $O(Nk \log k)$, where N is the number of vertices in the final triangular mesh and k is the average number of vertices in the bucket.

5.1. Checking the validity

It is possible that there are some inconsistencies in the final triangular mesh after the vertex duplicate elimination, e.g. coordinates can differ a little due to the numerical precision. In this case, long and thin disruptions in

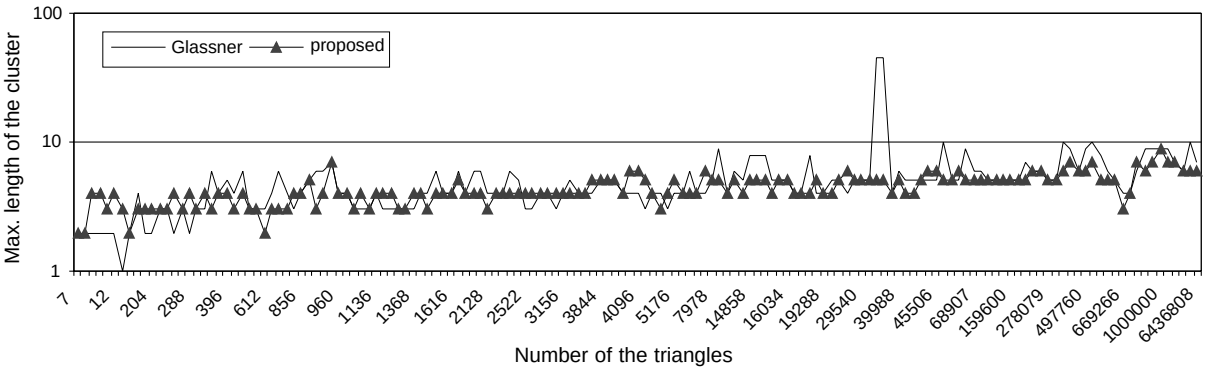

Fig. 9. Maximum cluster lengths for recent and proposed hash function. Coefficients used: $Q = 10^7$, $K = 10$, $\alpha = \pi$, $\beta = e$, $\gamma = \sqrt{2}$.

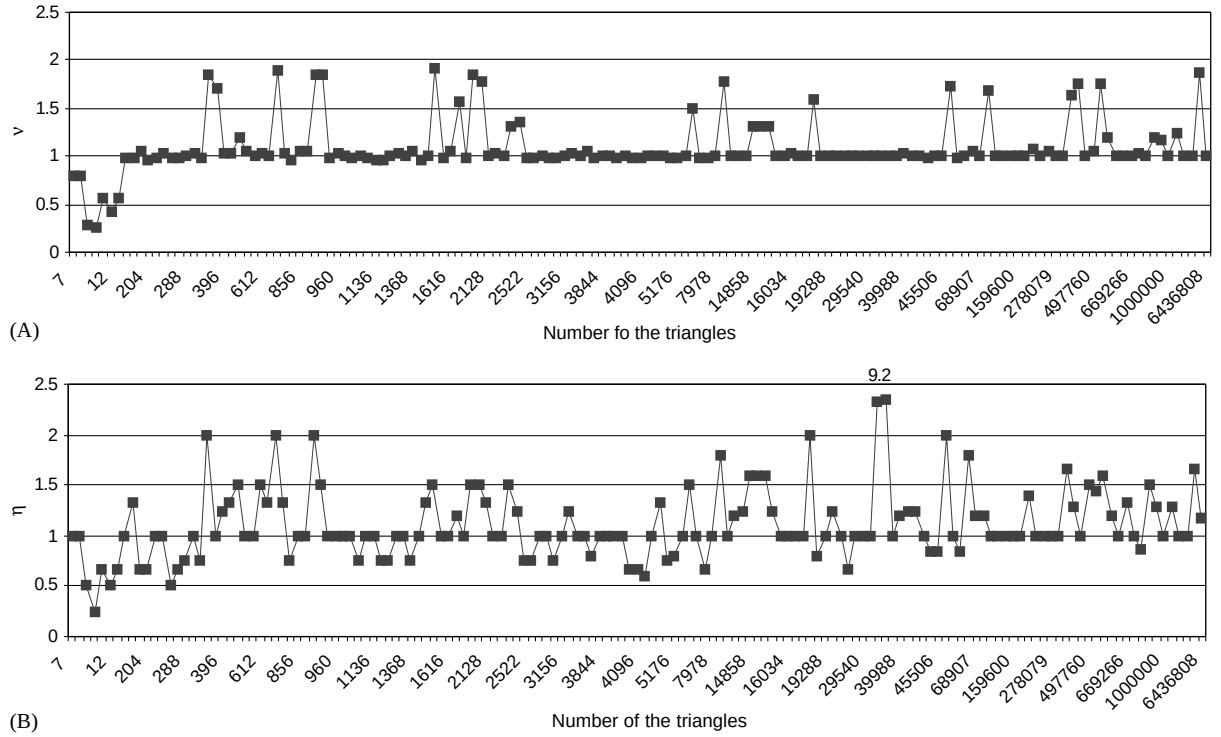

Fig. 10. (A) Comparison of average cluster lengths of recent and proposed hash functions. Coefficients used: $\alpha = \pi$, $\beta = e$, $\gamma = \sqrt{2}$., $Q = 10^7$, $K = 10$. (B) Comparison of maximal cluster lengths of recent and proposed hash functions. Coefficients used: $\alpha = \pi$, $\beta = e$, $\gamma = \sqrt{2}$. $Q = 10^7$, $K = 10$.

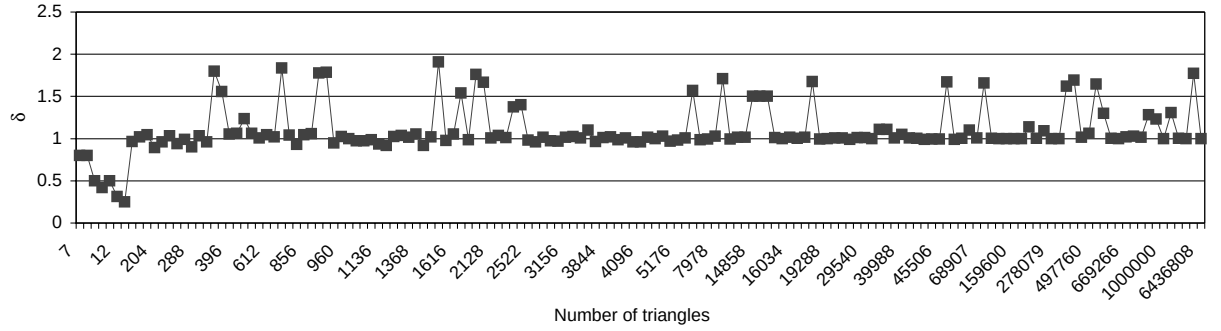

Fig. 11. Ratio of quality criteria R between recent and proposed function. Coefficients used: $Q = 10^7$, $K = 10$, $\alpha = \pi$, $\beta = e$, $\gamma = \sqrt{2}$.

the final triangular mesh can occur and have to be corrected.

There are several methods for checking errors in the triangular mesh:

- Vertex-to-vertex rule.
- Checking the number of edges and triangles in the mesh.
- Euler's rules.

Vertex-to-vertex rule: This rule reflects a fact that all inner edges must be shared by two triangles and each triangle must have two common vertices with its neighbours see Fig. 15.

Checking the number of edges and triangles: From the validity of the vertex-to-vertex rule can be seen that the number of edges of a solid closed surface is equal to $\frac{3}{2}$ of the number of triangles. Therefore the following three rules for checking surfaces are:

- Number of triangles must be even;
- Number of edges must be a multiple of 3;
- $2 \times$ number of edges must be equal to $3 \times$ number of triangles.

If at least one rule fails it means that there is a disruption in the surface. This validity check can be derived from Euler's rule, for the case when each face is triangular, and the surface is closed.

For each triangle:

a) For each vertex in the triangle perform the following steps:
   1. Evaluate an *index* to the hash table for the given vertex coordinates.
   2. Check the appropriate cluster for the presence of this vertex (the given vertex coordinates are checked with the vertex coordinates stored in the vertex array for all the cluster elements)
      - If the given vertex is already stored in the vertex array, its position stored in the cluster is returned.
      - Otherwise the vertex is not stored and therefore the vertex is inserted to the vertex array and the cluster is changed appropriately. The vertex position is returned.
   3. Insert returned position in the vertex array for the given vertex to the triangle array.

b) When three vertex positions are inserted, the given triangle is stored in the data structure and all duplicates are eliminated.

Fig. 12. Algorithm of triangular mesh reconstruction.

Euler's rule: Euler's rule (also described by Fornous, 2000) for *simple closed surfaces*, i.e. a simple polyhedron, is defined as

$$T + V - E = 2, \tag{14}$$

where T is the total number of triangles, V the total number of vertices, and E the total number of edges. A generalization of Euler's formula that applies to 2-*manifolds which have faces with holes* is defined as

$$T + V - E - H = 2(C - G), \tag{15}$$

where H is the number of holes in the faces, G the number of holes that pass through the object (also known as *genus*), and C the number of separate components.

For the *opened surfaces* Euler's formula is stated as

$$T + V - E = 1. \tag{16}$$

In each case of the surface, Euler's formula must be valid. If there is any inconsistency within the triangular mesh then the long thin holes are "sewed" or have to be corrected interactively by the user.

The proposed hash function was used for data duplicate elimination for large data sets in other modular visualization environments (MVE) (MVE Final Report, 2001; MVE User's Manual, 2001; MVE Programmer's Manual, 2001) modules and for a triangular mesh decimation module (Franc, 2000). The mesh reduction algorithm complexity was reduced from $O(N \log N)$ to $O(N)$.

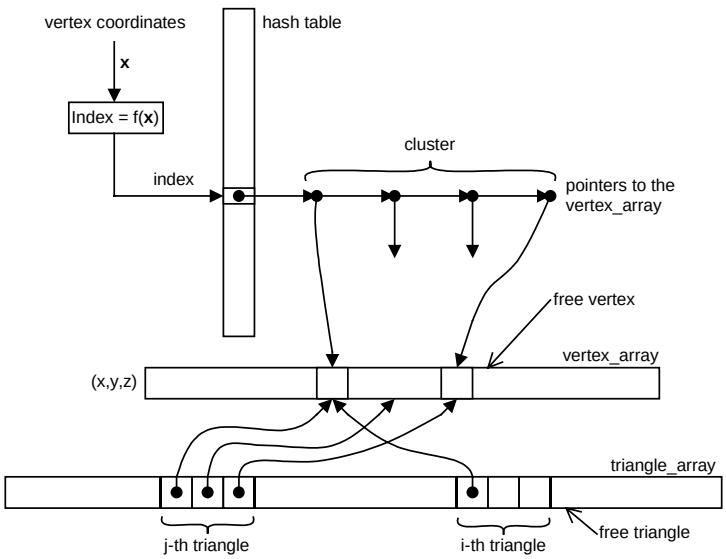

Fig. 13. Data structure of vertices and triangles.

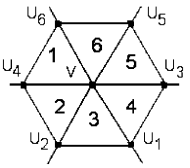

Fig. 14. Triangles sharing vertex v. Bucket in v contains vertices u_1-u_6.

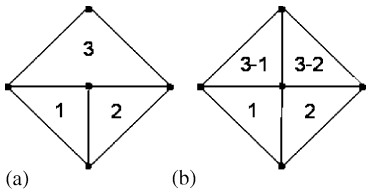

Fig. 15. Vertex-to-vertex rule. Triangle 3 has two neighbours on 1 edge, thus vertex-to-vertex rule has failed (a). Solution lies in dividing triangle 3 into two triangles (b).

6. Conclusion

A new hash function was designed and experimentally verified for geometric purposes. It was used for triangular mesh reconstruction and reduction purposes. It uses the advantage of the full-addressability available with AI32 processors.

The proposed hash function has the following advantages:

- There are no parameters related to the data set properties, like the parameter Q in the recent hash function.
- It has stable behaviour for all data sets used in experiments.
- There are no additional memory requirements in comparison with the recent hash function.
- It provides high speedup especially for cases where the parameter Q of the recent hash function is not known or is selected "randomly". It provides better performance even if the parameter Q is chosen carefully ("tuned for the given data set"), i.e. average cluster length ratio is 89.1%, maximum cluster length ratio is 87.1% and ratio of the search criterion is 88.7% of the recent hash function.
- The proposed hash function will be at least 12.7% faster for data sets with unknown properties. The recent hash function is sensitive to the choice of the parameters α, β, γ and also to the choice of the parameter Q, that is critical, see Fig. 4, when very long clusters are generated.
- The number of clusters decreases with the cluster length almost monotonically.

Experiments have demonstrated that the proposed measurement of hash function quality gives exactly the same results as the measurement proposed in Jenkins (2001) if empty cells in the hash table are not considered.

The proposed hash function was used in a module of triangular mesh reconstruction for the modular visualization environment (MVE) system developed at the University of West Bohemia, see http://herakles.zcu.cz for details.

Acknowledgements

The authors would like to thank all those who contributed to this work, especially to colleagues and Ph.D. students at the University of West Bohemia in Pilsen who have stimulated this work. This paper has benefited greatly from several discussions with them. Their invaluable critical comments and suggestions improved the manuscript significantly.

We would also like to thank the Auto-Skoda Volkswagen group, Cyberware.com and the Georgia Institute of Technology for providing the data sets used in our experiments.

References

Foley, J.D., van Dam, A., Feiner, S.K., Huges, J.F., 1997 Computer Graphics, Principles and Practice. Addison-Wesley, Reading, MA, 1200pp.

Franc, M., Skala, V., 2000. Triangular mesh decimation in parallel environment. In: EUROGRAPHICS Workshop on Parallel Graphics and Visualization, Girona, Spain, pp. 39–52.

Formaggia, L., 2001. Data structures for unstructured mesh generation. In: Thomson, J.F., Soni, B.K., Weatherill, N.P. (Eds.), Handbook of Grid Generation. CRC Press, Boca Raton, FL, pp. 14–1:14–22.

Fornous, J., 2000. MVE module for Earth surface model. B.Sc. Thesis, University of West Bohemia, Pilsen, Czech Republic, 34pp [in Czech].

Glassner, A., 1994. Building vertex normals from an unstructured polygon list. In: Graphic Gems IV. Academic Press, New York, pp. 60–73.

Jenkins, B., 2001. Hash Function FAQ. http://burtleburtle.net/bob/hash/hashfaq.html

Korfhage, R.R., Gibbs, N.E., 1987. Principles of Data Structures and Algorithms with Pascal. Wm. C. Brown Publishers, McGraw-Hill Science/Engineering/Math. Dubuque, Iowa. 480pp.

Kuchar, M., 2000. Construction of the triangular meshes from STL data format and stereoscopic visualization. M.Sc. Thesis. University of West Bohemia, Pilsen, Czech Republic, 70pp [in Czech].

Knuth, D., 1998. The Art of Computer Programming. Vol. 3, Sorting and Searching, 2nd Edition. Addison-Wesley, Reading, MA, 736pp.

MVE Final Report, 2001. MVE modular visualization environment-final report. University of West Bohemia, Pilsen, Czech Republic, 29pp.

MVE User's Manual, 2001. MVE modular visualization environment-user's manual. University of West Bohemia, Pilsen, Czech Republic, 45pp.

MVE Programmer's Manual, 2001. MVE modular visualization environment-programmer's manual. University of West Bohemia, Pilsen, Czech Republic, 21pp.

Wolfson, H.J., Rigoutsos, I., 1997. Geometric hashing: an overview. IEEE Computational Science and Engineering 4 (4), 10–21.

Yao, A.C., 1985. Uniform hashing is optimal. Journal of the Association for Computing Machinery 32 (3), 687–693.