13th International Conference on COMPUTERS, WSEAS, pp. 233-238, ISBN: 978-960-474-099-4, 2009

Hash Function for Triangular Mesh Reconstruction

Václav Skala, Jan Hrádek, Martin Kuchař
Department of Computer Science and Engineering, University of West Bohemia
Univerzitní 8, Box 314, CZ 306 14 Plzeň, Czech Republic
http://Graphics.zcu.cz

Abstract: - Some applications use data formats (for example STL file format), where a set of triangles is used to represent a surface of an object and it is necessary to reconstruct the regular triangular mesh from such a data format for many applications. It is a lengthy process for large data sets as the time complexity of this process is $O(N^2)$ or O(N lgN), where N is a number of triangles. Hash data structures are widely used all over the fields 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 problem. This paper describes a new hash function and presents properties obtained from large data sets.

Key-words: computer graphics, triangular mesh reconstruction, hash function

1. Introduction

There are several problems related to properties of the triangular mesh representation that describes a surface of the object. Sometimes, the surface is represented just as a set of triangles without other information and the STL file format which is used for data exchanges is a typical example of this case. The STL format is very simple and all objects are represented as a polyhedra. More precisely, polygonal facets represent the surface and in the vast majority the surface is represented as a set of triangles [1]. It contains information on the face's normal and co-ordinates of all vertices.

A drawback of the STL file format is that it does not contain any information on structure, topology, etc. This is needed if the triangular mesh is required, e.g. neighbours of a triangle or triangles, which share the given vertex, are needed, etc. Also vertex co-ordinates are stored several times.

One possibility is to reconstruct the triangular mesh from the given set of triangles. It enables us to compute normal vectors in for Gouraud vertices and Phong shading efficiently, etc. The main problem is to find all triangles that share the same vertex of the triangular mesh. This must be made for all the vertices of the triangular mesh. 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. To be able to reconstruct the triangular mesh, it is necessary to read all vertices, sort them according to the one co-ordinate, remove duplicities (the same vertex is stored several times if it is shared by more triangles) and create triangular mesh with information on neighbour triangles etc. In the triangular mesh a vertex is shared by triangles and co-ordinates are stored only once. This is a process of $\boldsymbol{O(N^2)}$ - for brute force - or $\boldsymbol{O(N \cdot lgN)}$, if sort is used, complexities generally (where N is a number of triangles) and it is highly time consuming process if the considered objects are represented by 10^6 - 10^7 triangles. Typical data sets to be processed are presented at Figure 1 and Figure 2.

There have been several attempts to subdivide a space into subspaces, but the obtained results heavily depended on data sets, especially on how the vertices were scattered in space. Some approaches how to overcome the complexity using the hash function have been published recently and resulted to an expected $\boldsymbol{O(N \cdot p)}$ algorithm complexity, in general [2,4] (p is an average cluster length). The hash function transforms a value (or a key value) into an index (or an address) that points to a table (or a file) where this value is stored. The fundamental requirements on the hash function are simplicity and collision elimination, when the index is the same for different values, as much as possible. It is impossible to avoid collisions in general and therefore different values with the equal index to the hash table are stored in clusters (or buckets). Number of collisions depends also on the *load factor* α. The load factor α is a ratio of the

number of values stored and the total length of the table. The basic idea of this approach is to obtain *O(1)* expected complexity for a query "find all triangles having the given vertex co-ordinates equal to ….". This type of the query is to be answered for all vertices of the given set of triangles. It can be seen that the efficient solution is very critical for the large triangular mesh sizes.

2. Hash function properties

2.1 Recent solution

It is known that the hash function has to have some properties and the most important are:

- to use all cells of the hash table,
- a maximal and an average cluster length should be as low as possible (cluster is usually implemented as a list of primitives if the hash function gives the same index for different values),
- the hash function must be as simple as possible in order to have very fast evaluation.

The hash function was introduced for triangular mesh reconstruction [2] recently as

Index = ((int) ((α * ((int) (abs(X) * Q)) / Q
+ β * ((int) (abs(Y)*Q)) / Q
+ γ * ((int) (abs(Z) *Q)) / Q) * SIZE)) % SIZE

(int) is the conversion to integer - the fractional part of the float is removed (in current implementation DWORD is used),

α,β,γ are coefficients of hash function (originally used 3, 5 and 7),

% represents modulo operation,

Q defines sensitivity (numerical error elimination) - for 3 decimal digits set Q = 1000.0,

SIZE - hash table size - generally it is implemented as 2^k for a fast evaluation of the **modulo** operation,

The hash function shown above uses a very simple formula that can be recommended for small or medium data sets only (according to our experiments), see Table 1. The experiments proved that Q = 10^3 (used by Glassner [2]) is unsatisfactory because long clusters were generated. Better results were obtained for Q = 10^7, but these results strongly depended on the fractional part of vertices' co-ordinates, see Figures 3 - 7.

2.2 Proposed solution

Data analysis proved that

- it is not reasonable to remove the fractional part from co-ordinate value, it helps us to distinguish co-ordinates better,
- it is necessary to remove all coefficients or parameters that depend on the data set somehow, or should be given by a user
- to use all available memory as much as possible to get a longer hash table,
- the hash function should not be a static one - it should be dynamic according to the currently available memory, but generally the size of the hash table can be fixed by some rules.

When considering required properties of the hash function and data set processed, several functions have been derived after many experiments

Index = (DWORD) ((α * X' + β * Y' + γ * Z') * C + 0.5) & T

X', Y', Z' are vertex co-ordinates (see next),

α, β, γ are coefficients of the hash function,

C is a scaling coefficient set so that the full range of DWORD type (4 Bytes unsigned) is used, i.e. range of the interval <0, 2^{32}-1> is used,

T+1 is the table size (T = 2^k-1),

& represents a **modulo** that is realized as a logical operation **and** with the DWORD type

For simplicity we will assume that all co-ordinates *x* are from the < 0 , X_{max} > interval, similarly for others. Then we can compute maximal value ξ_{max} that can be obtained from the formula as

$$\xi_{max} = \alpha * X_{max} + \beta * Y_{max} + \gamma * Z_{max}$$

The *C* coefficient must be determined as:

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

where: $C_1 * \xi_{max} <= 2^{32} - 1 C_2 = 2^{32} - 2^k$

because the overflow operation must be avoided and we want to use the whole size of the table as well. The reason for that is that we want to use the whole interval <0, 2^{32}-1> before applying modulo operator. While testing the hash function we experimented with some adjustments of vertex co-ordinates X, Y, Z:

A. co-cordinates are not changed

$$X' = X, \quad Y' = Y, \quad Z' = Z$$

B. each co-ordinate is converted to range <0,1> using this formula

$$X' = \left(1 + \frac{X}{|X| + K}\right) * \frac{1}{2}$$, similarly for Y', Z',

C. each co-ordinate is converted to <0,1>

using this formula $X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$

,similarly for Y', Z'.

Although all the methods are almost the same we obtained the best results with method II, where max. cluster lengths were less than 10, see Sec. 3.

2.3 The table size determination

The length of the hash table must depend on the size of data that we want to process and it is necessary to point out that there are three times more vertices than triangles in the STL format.

It is well known that the length of the table and the estimated length of a cluster is in an empirical relation with the *load factor* α. If we consider the *load factor* $\alpha = 0,5$ we can expect the cluster length about 2,5.

The length T of the hash table can be expressed as $T \geq \frac{N_v}{q_{avg}} . \frac{1}{\alpha} = N_T$ as $N_v = 3N_T$

where: N_T is the number of triangles in STL file,
N_v is the number of vertices in STL file,
q_{avg} is an average number of triangles sharing the same vertex (approx. 6)
load factor - $\alpha = 0,5$ used*; the lower value used the better spread out.

In practice the value T is chosen as 2^k in order to be able to use the **logical and** operator instead of **modulo** as this solution is much faster.

2.4 The evaluation for comparison

The tests involved several types of hash functions so we needed to have a simple method for their comparison. Our criteria for the quality of the hash function were:

- maximal cluster length,
- average cluster length,
- the number of checked items in clusters while processing all the vertices in the data.

The hash function with better quality is that one which provides smaller number of checked items in clusters. The proposed hash function *B* (see Section 2.2) uses the advantage of large memory available today as well as properties of the proposed hash construction. The hash function has been tested on many data sets and proved similar properties for all data sets. The sizes of the tested files varied from 10^5 to 2.10^7 of vertices and the proposed hash function proved its stability. Table 1-2 present the differences between the recent and proposed hash functions.

Let us introduce the coefficient ν as

$$\nu = \frac{\text{Average cluster length}_{\text{recent}}}{\text{Average cluster length}_{\text{proposed}}}$$

It presents the expected speed-up (in average) for answering the query (it gives a pointer to the list of triangles actually): Which triangles share the given vertex ?

Let us define the coefficient η as

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

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

The third criterion is based on the following idea: If the cluster length is i then we access this cluster i-times. Because the number of these clusters (with length i) is N_i, the whole number of accesses to these clusters is $i \cdot N_i$. There are three cases of the number of checked items N_c while accessing a cluster of the length i:

a) the cluster is empty- nothing to check; $N_c = 0$
b) the cluster is not empty, but the vertex has not stored there yet
   - we check all the items in the cluster; $N_c = i$
c) the cluster is not empty and the vertex has already stored there
   - the number of checked items is from 1 up to i, but in average it is $i/2$, i.e. $N_c = i/2$

Therefore the quality R of the hash function can be evaluated by the formula

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

i_{max} is the maximal cluster length,
i is the cluster length,
N_i is the number of clusters with length i,
DV is the number of different vertices in data.

Value of this quality function is better for comparing hash functions on all the data sets, than just the average and maximal cluster length.

The comparison is made by dividing criterion and the results of the comparison of the recent and proposed function are at Figure 15.

3. Experimental results

The proposed hash function has been tested on about 150 different non-trivial data sets. The coefficient K was chosen $K = 10$ according to the

* $\alpha \in (0,25 ; 0,5>$ due to rounding to power of 2.

experiments. Table 2 presents the typical behaviour on some selected data sets. The relation of the cluster length and number of triangles is presented on Figures 8-9. It can be seen that the maximal cluster length is limited by the order of 10, which is a very good result if compared with the recent solution, see Table 1. Also the number of clusters decreases with the cluster length and this is a very good property of the proposed hash function as well, during all tests each (recent and proposed) hash function use the same length of the table for each data set.

Figure 1: Part of an auto chassis
Data set: A4_unter.stl, courtesy of Skoda-Auto

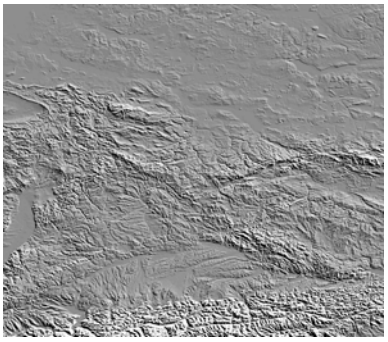


Figure 2: Earth's surface, Data set: Central Europe.stl

Figures 1 – 2 present some of the typical data sets used for testing and evaluation of the recent and the proposed method. The size of the data sets varied from $3 \cdot 10^6$ to $1{,}9 \cdot 10^7$ vertices.

Cluster length distribution for the recent hash function

File	No.triangles	No. vertices	No.vertices	α	β	γ	Q	Avg. length	Max. length
A4_unter	1 000 790	3 002 370	618 865	3	5	7	3	202.3	12 093
				3	5	7	7	1.8	208
				π	e	$\sqrt{2}$	7	1.6	9
Central Europe	1 605 608	4 816 824	804 601	3	5	7	7	152.2	5 116
				π	e	$\sqrt{2}$	7	1.2	7

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

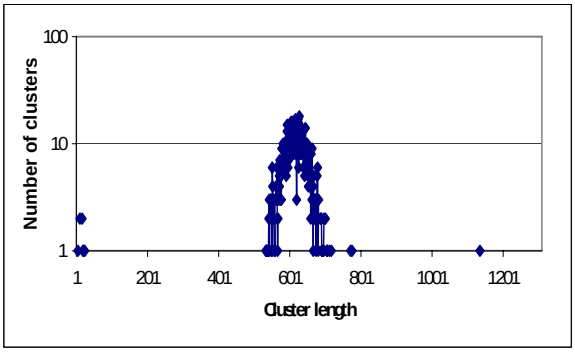
File	No.triangles	No. vertices	No. vertices	α	β	γ	Q	Avg. length	Max. length
A4_unter	1 000 790	3 002 370	618 865	3	5	7	10	1.3	7
				π	e	$\sqrt{2}$	10	1.3	7
Central Europe	1 605 608	4 816 824	804 601	3	5	7	10	1.2	6
				π	e	$\sqrt{2}$	10	1.2	7

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

	File	Average cluster length		ν	Maximal cluster length		η
		Recent	Proposed		Recent	Proposed	
1	A4_unter	1.8	1.3	1.4	208	7	29.7
	Central Europe	152.3	1.2	26.5	5 116	6	852.7
2	A4_unter	1.6	1.3	1.2	10	7	1.4
	Central Europe	1.2	1.2	1.0	6	7	0.9

Table 3: Comparison of cluster lengths of the recent and the proposed hash functions[+]
Coefficients used: (1) $\alpha = 3$, $\beta = 5$, $\gamma = 7$, $Q = 10^7$, $K = 10$, (2) $\alpha = \pi$, $\beta = e$, $\gamma = \sqrt{2}$, $Q = 10^7$, $K = 10$

[+] It is necessary to point out, that the Central Europe.stl file was generated from the Digital Terrain System (GTOPO30), while the A4_unterbau1.stl file was "real life" data from the automobile industry.



Figures 3 - 5 demonstrate the recent hash function behavior according to different parameters and for typical data sets. From the Table one it can be seen that the maximal and average cluster lengths are not acceptable for practical use of this function, because it causes high overhead during search operation.

Figure 3: data set: A4_unter.stl, $Q = 10^3$, $\alpha = 3$, $\beta = 5$, $\gamma = 7$

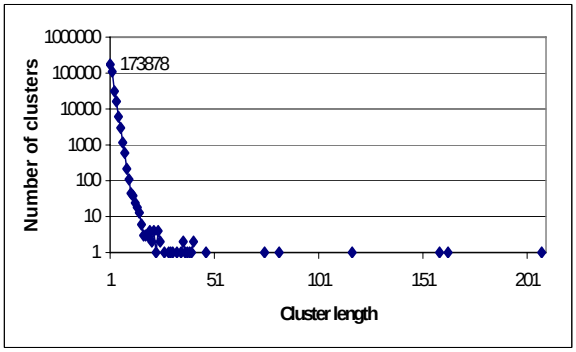


Figure 4: data set: A4_unter.stl, $Q = 10^7$, $\alpha = 3$, $\beta = 5$, $\gamma = 7$

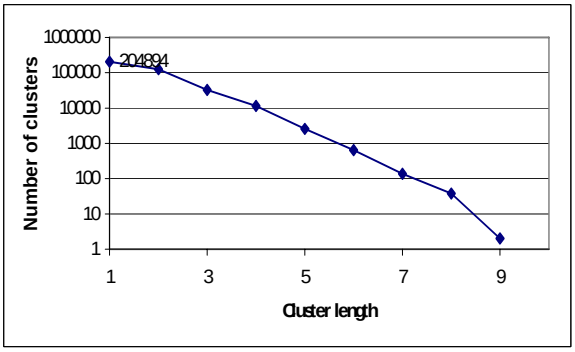


Figure 5: data set: A4_unter.stl,$Q = 10^7$, $\alpha = \pi$, $\beta = e$, $\gamma = \sqrt{2}$

Cluster length distribution for the proposed hash function

Table 2 and Figures 6-7 present behaviour of the proposed hash function for the same data sets.

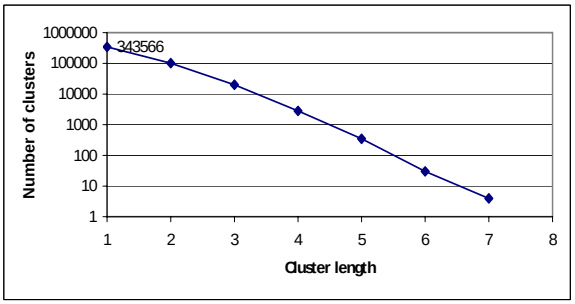


Figure 6: data set: A4_unterbau1.stl, $K = 10$, $\alpha = \pi$, $\beta = e$, $\gamma = \sqrt{2}$

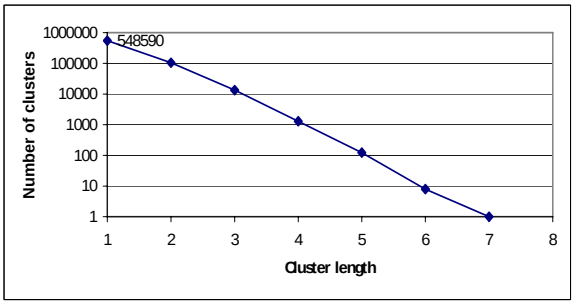


Figure 7: data set: Central Europe.stl, $K = 10$, $\alpha = \pi$, $\beta = e$, $\gamma = \sqrt{2}$

Results for representative data sets

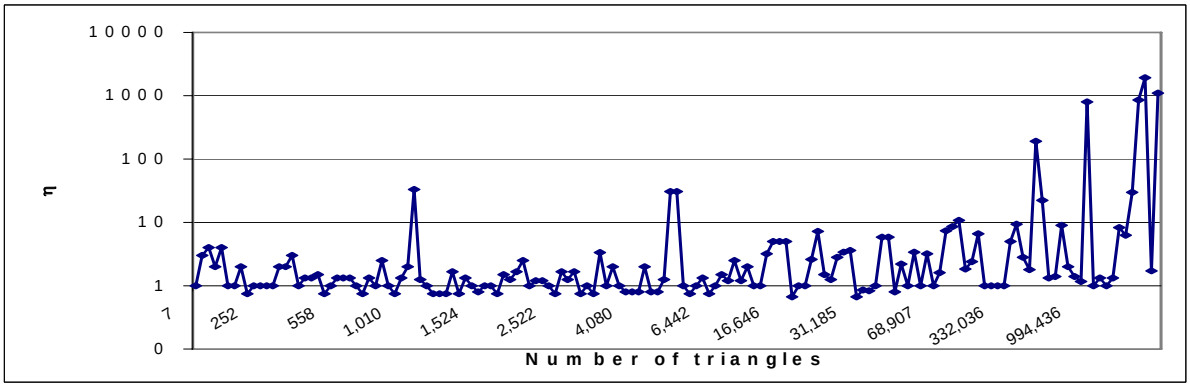


Figure 8: The proposed hash functions – Max.cluster length: $\alpha = 3$, $\beta = 5$, $\gamma = 7$, $Q = 10^7$, $K = 10$

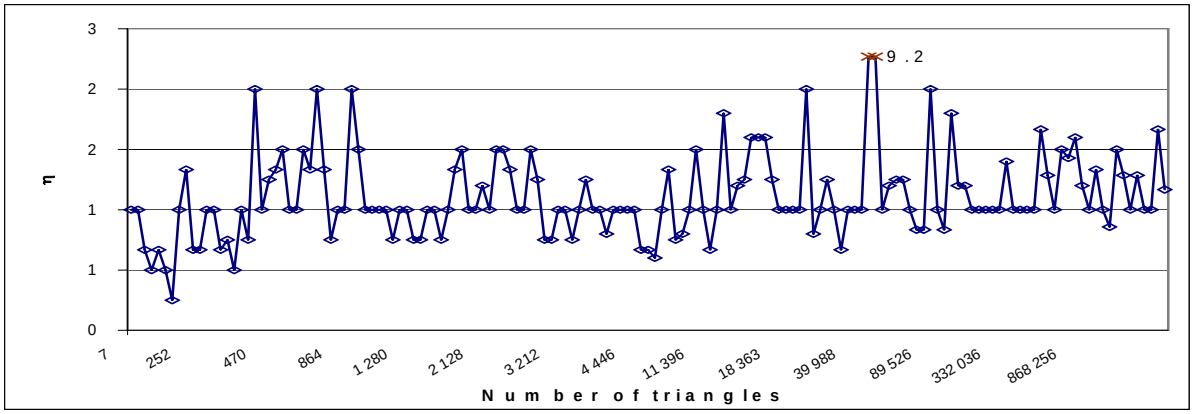


Figure 9: The proposed hash functions – Max.cluster length: $\alpha = \pi$, $\beta = e$, $\gamma = \sqrt{2}$, $Q = 10^7$, $K = 10$

Table 3 shows the ratio for maximal and average cluster sizes for recent and proposed hash function for two typical data sets. Figure 8 presents the experimental results for maximal and average cluster sizes of the recent and proposed hash functions for representative data sets. Figure 9 presents results for the same functions but with different parameters. The proposed hash function is better nearly in all cases. When the proper data structure representing a triangular mesh is created it is necessary to make final steps in triangular mesh reconstruction and check the validity of the model surface. The validity checking and the proposed hash functions have been used in a special module for triangular mesh reconstruction in the MVE (Modular Visualization Environment) [4,6] system.

4. Conclusion

The proposed hash function has the following advantages:

- a stable behaviour,
- a short maximal cluster length,
- the number of clusters decreases with the cluster length monotonically, it does not significantly depend on parameters defined by user (Q), on actual co-ordinate values,
- it is flexible according to the number of vertices processed and co-ordinate values,
- we can obtain very high speed-up for a simple query - coefficient η in Table 3,
- it gives us faster solution for triangular mesh reconstruction - coefficient ν in Table 3.

The shown experiments proved that there is a high potential for a hash function use and improvement. Experiments have proved that the proposed measurement of hash function quality gives exactly the same results as measurement proposed in [5] if empty cells in the hash table are not considered.

Acknowledgement

The authors would like to thank to all who contributed to this work, especially to colleagues, MSc. and PhD. students at the University of West Bohemia in Plzen. We would like to thanks also to Auto-Skoda VWgroup for provided data sets, Cyberware.com model gallery and Georgia Institute of Technology data repository. project supported by MSMR CR No.2C06002.

References

[1] Foley,J.D., van Dam,A., Feiner,S.K., Huges,J.F.: *Computer Graphics*, *Principle and Practice*. Addison-Wesley, 1997.

[2] Glassner, A.: Building Vertex Normals from an Unstructured Polygon List, *Graphic Gems IV*, 60 – 73. Academic Press, Inc., 1994.

[3] Kuchar,M., (supervisor Skala,V.): *Construction of the triangular meshes from STL Data Format and Stereoscopic visualization*, MSc. thesis. UWB, Plzen, Czech Republic, 2000.

[4] Franc,M., Skala,V.: Triangular Mesh Decimation in Parallel Environment, *EUROGRAPHICS Workshop on Parallel Graphics and Visualization*, Girona, Spain, pp.39-52, 2000.

[5] Jenkins,B.: *Hash Function FAQ,*, http://burtleburtle.net/bob/hash/hashfaq.html

[6] *MVE - Final Report*. UWB, Plzen, Czech Republic, 2001, http://Graphics.zcu.cz