Proceedings of ALGORITMY 2000
Conference on Scientific Computing, pp. 379 - 387

HASH FUNCTION FOR GEOMETRY RECONSTRUCTION IN RAPID PROTOTYPING

VÁCLAV SKALA[1], MARTIN KUCHAŘ[2]

Abstract. There are many applications where data structures use a hash function. The hash data structures are widely used across all fields of computer science. Nevertheless the design or selection of hash function for data sets with unknown properties is a problem. The Rapid Prototyping uses STL format, where a set of triangles is used to represent a surface of the object. It is necessary to construct the regular triangular mesh from the STL 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.

The hash table can be used to speed up the process but the speed strongly depends on hash function properties. This paper describes a new hash function and presents properties obtained on large data sets.

Keywords: data visualization, hash function, algorithm complexity, computer graphics, triangular mesh, STL format.

1. Introduction. There are several problems that are related to properties of the triangular mesh representation that describes a surface of the object. In some applications the surface is represented as a set of triangles and the Rapid Prototyping (RP) is one of those. The Rapid Prototyping is a very progressive technology that enables fast production of a prototype as a true mechanical body. It is possible to make quite complex shapes of inner parts and with holes, etc. There are several physical principles used for production of a final product (laminated paper, stereo-lithography etc.) but all of them are based on the STL format, that is standard format for data exchanges. The STL format is very simple as all objects are represented as polyhedra. More precisely, the surface is represented by polygonal facets, in vast majority of cases by triangular meshes. It contains information on the face normal and co-ordinates of all vertices, fig.1.

The major problem with the STL file format is that it does not contain any information on geometry, topology etc. The files created from solid models have about 10% of anomalies and those created from surface models have about 90% of problems. Error rates in this range make it clear that automated error checking is important for all RP operations.

```
solid
        facet   normal 0.1 -0.5 1.0
                outer loop
                        vertex 15.0 150.0 69.5
```

[1] Supported by the Academy of Sciences of the Czech Republic - project A2030801
[2] Supported by the Ministry of Education of the Czech Republic - project VS 97155

Department of Computer Science, University of West Bohemia, Univerzitní 8, Box 314, 306 14 Plzeň, Czech Republic, (e-mail: **skala@kiv.zcu.cz and kuchy@students.zcu.cz**)
http://iason.zcu.cz/~{skala | kuchy }

```
                        vertex 19.0 148.5 22.15
                        vertex 142.22 511.7 655.99
                endloop
        endfacet
        facet   normal ………….
                outer loop
                        vertex 15.0 150.0 69.5
                        vertex …………..
                        vertex …………..
                endloop
        endfacet
endsolid
```

FIG. 1 Example of the STL format - one vertex shared

For a reliable RP production some checks are needed but they are not generally reliable. Another possibility is the geometry reconstruction from the given set of triangles.

The main problem is to find all triangles that share the same given vertex of the triangular mesh. This must be made for all vertices of the given mesh. The reconstruction of the triangular mesh from the given set of triangles is a critical operation as the co-ordinates of one vertex occur in the STL format several times, see fig.1 for STL format description.

To be able to reconstruct the triangular mesh it is necessary to read all vertices, sort them according to one co-ordinate, remove duplicities (the same vertex is stored several times if the triangle vertex is shared) and create regular triangular mesh with information on neighbors triangles etc. In triangular mesh a vertex is shared with triangles properly and stored once, only. This process is of ***$O(N^2)$*** or ***O(N lgN)*** complexities and it is highly time consuming process if objects considered are represented by 10^6 - 10^7 triangles.

There have been some attempts to subdivide a space into subspaces, but the obtained results heavily depended on data sets, especially how the vertices were scattered in space. Some approaches how to overcome the complexity using the hash function has been published recently and resulted to hopefully expected ***O(N)*** complexity, in general [1].

The basic idea is to obtain ***O(1)*** complexity for a query "find all triangles having a vertex co-ordinates equal to …." as this type of query is to be answered for all vertices of the given set of triangles. It can be seen that for triangular mesh sizes under consideration the efficient solution is a very critical point.

2. Original solution. It is known that the hash function has to have some properties, the most important are:

- to use all cells of the hash table as much as possible,

- maximal and average cluster length should be as low as possible (cluster is usually implemented as a list of primitives for cases when the hash function gives the same value),
- the hash function must be as simple as possible in order to have very fast evaluation.

The original hash function was defined [1] as

```
Index  =  ((int) ((3 * ((int) (fabs(X) * Q)) / Q + 5 * ((int) (fabs(Y) * Q)) / Q
                  + 7 * ((int) (fabs(Z) * Q)) / Q) * SIZE)) % SIZE
```

where: (int) is the conversion to integer - the fraction part of the float is removed,

fabs is a Absolute Value function,

% represents modulo operation,

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

SIZE is the size of the hash table that is determined as described later, but generally as 2^k for fast evaluation of the **modulo** and **division** operations

X, Y, Z are co-ordinates of a vertex.

File	Number of triangles	Original number of vertices	Final number of vertices	Maximal cluster length
CTHead.stl	555 411	1 666 233	278 856	356
Gener.stl	500 000	1 500 000	50 002	577
Teapot.stl	159 600	478 800	80 202	110

TABLE 1 *Typical characteristic of the original hash function for STL data*
Original number of vertices - number of vertices in the STL data
Final number of vertices - number of vertices in the final triangular mesh

The hash function shown about uses very simple formula that is recommended in all publications usually for small or medium data sets. Nevertheless when the property of the hash function was experimentally verified it has not proved good properties for large data sets, see fig. 2 - 4. The experiments proved that the function has relatively stable properties nearly without influence of the coefficient Q.

One of disadvantage of the hash function is that the coefficient Q depends on the data and can lead to mixing some vertices together. Due this fact the small triangles might be taken as zero-sized triangles and the triangular mesh is not correct at the end.

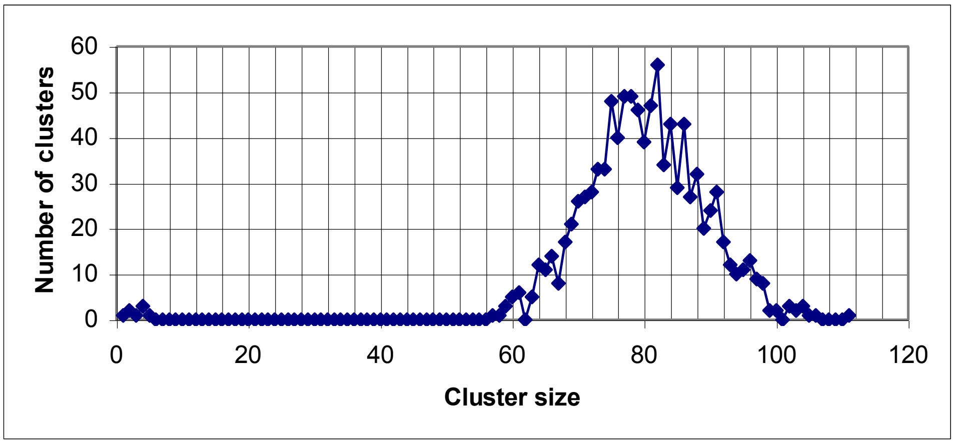

FIG. 2 *Original hash function property for precision 7 decimal points*

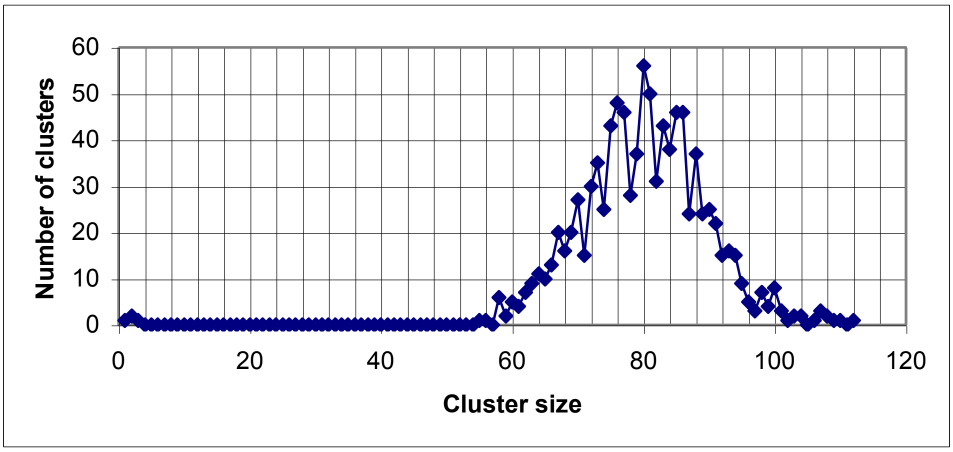

FIG. 3 *Original hash function property for precision 8 decimal points*

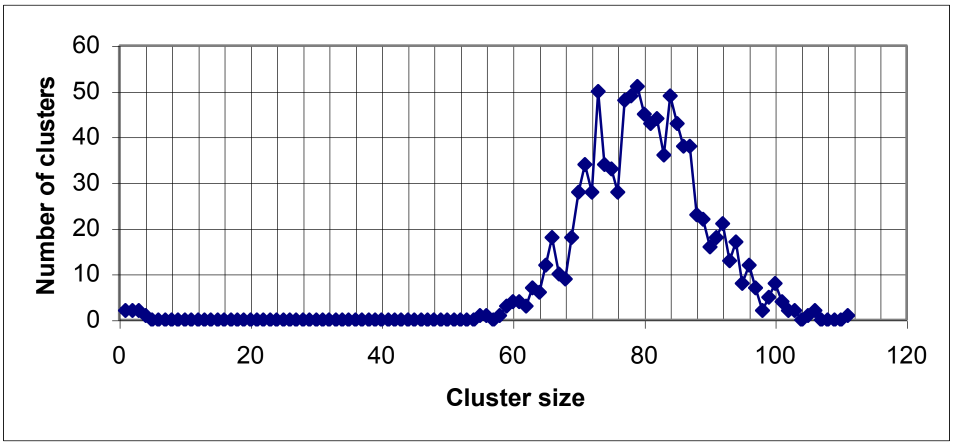

FIG. 4 *Original hash function property for precision 9 decimal points*

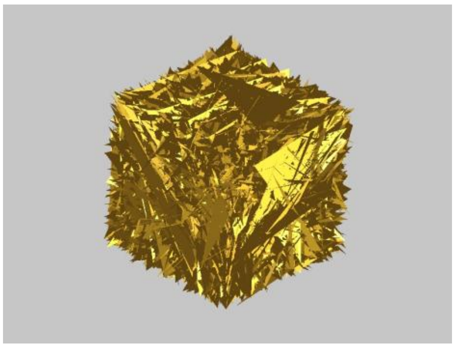

FIG. 5 *Randomly generated triangles*

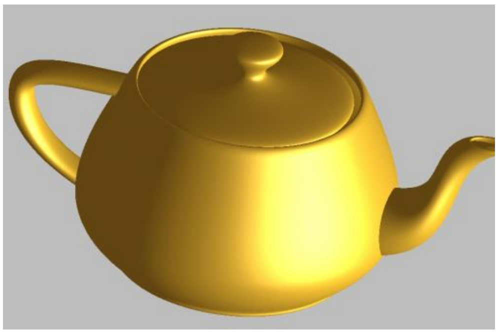

FIG. 6 *Teapot data set*

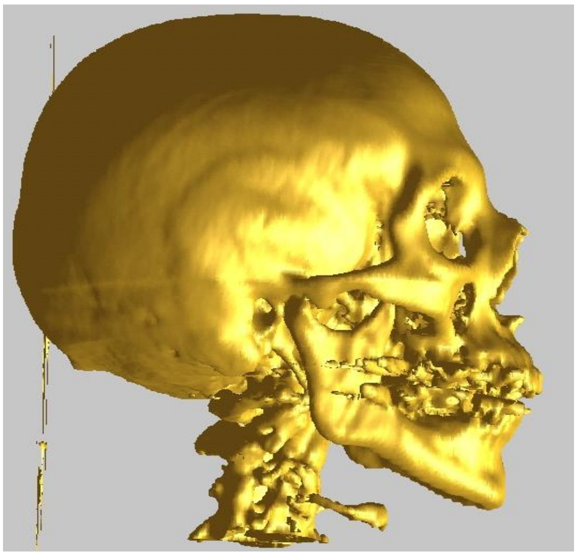

FIG. 7 *CT Head data set*

Fig.5 - 7 present some of the typical data sets used for testing and evaluation of the proposed method. The size of the data sets vary from 478 800 to 1 666 233 vertices.

3. Proposed solution. Data analysis proved that

- it is not reasonable to remove fraction part from co-ordinate value as it helps us to distinguish co-ordinates better,
- it is necessary to remove all coefficients that depends on data set somehow,
- to use the available memory as much as possible to get larger hash table,
- the hash function should not be static one - it should be dynamic according to currently available memory, but generally the size of the hash table can be fixed.

Taking into account required properties of the hash function, several functions have been derived.

$$\text{Index} = (\text{DWORD})\,((\alpha * X + \beta * Y + \gamma * Z) * C + 0.5f)\ \&\ T$$

where:

α, β and γ are coefficients of the hash function 3, 5 and 7,

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

T + 1 is the table size and the operator & represents **modulo** that is realized as logical operation **and** with the DWORD type - for fast computation,

0.5f is a constant that represents actually the rounding operation and gives better spread out of co-ordinates.

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

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

Because the overflow operation must be avoided and also we must use the whole size of the table. Therefore the C coefficient must be determined as

$$C = \min \{ C_1 , C_2 \} \qquad \text{where: } C_1 * \xi <= 2^{32} - 1 \qquad C_2 = 2^{32} - 2^k$$

So far we have dealt with the hash function property regardless to the length of the hash table. It must depend on the size of data we are going to process.

It is well known that the length of the table and estimated length of a cluster is in relation with the *load factor* α, see [2] for details. There are three times more vertices than triangles in the STL format. If we consider the *load factor* $\alpha = 0,5$ we can expect cluster length about 2,5.

The length *T* of the hash table can be expressed as

$$T \geq \frac{N_v}{q_{avg}} . \frac{1}{\alpha} = N_T \qquad \& \qquad N_v = 3N_T$$

where:

N_T is a number of triangles,

N_v number of vertices in the STL format,

q_{avg} is average number of triangles that share 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.

File	Number of triangles	Original number of vertices	Final number of vertices	Maximal cluster length
Gener.stl	500 000	1 500 000	500 002	9
CT Head.stl	555 411	1 666 233	278 856	6
Teapot.stl	159 600	478 800	80 202	7

TABLE 2 *Typical characteristic of the proposed hash function for STL data*
Original number of vertices - number of vertices in the STL data
Final number of vertices - number of vertices in the final triangular mesh

4. Experimental results. The proposed hash function has been tested on different data sets and tab.2 presents behavior of those data sets. Fig.8 - 10 present the relation of the cluster length and number of clusters. It can be seen that maximal cluster length is limited to the length lower than 10, what is very good result. Also the number of clusters decreases with the cluster length that is a very good property of the hash function.

File	Number of triangles	Original number of vertices	Final number of vertices	Maximal cluster length
CT Head.stl	555 411	1 666 233	278 856	6

TABLE 3 *Proposed hash function behavior for 4-times longer hash table, i.e.* $\alpha = 0{,}125$

Fig.11 shows the influence of the length of the hash table - actually the value of α has been changed to $\alpha = 0{,}125$. It means that the setting α value lower than 0,5 does not mean dramatic changes in hash function behavior, that is well know feature of the well designed hash functions.

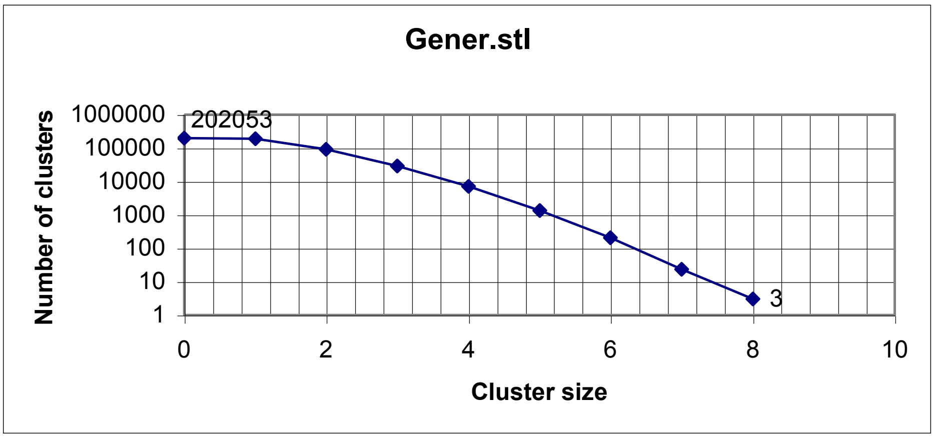

FIG. 8 *Proposed hash function property for Gener.stl data file*

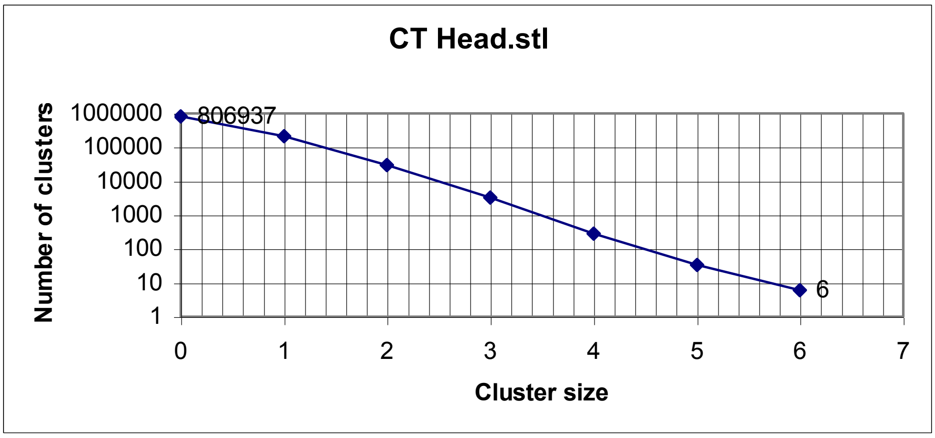

FIG. 9 *Proposed hash function property for CT Head.stl data file*

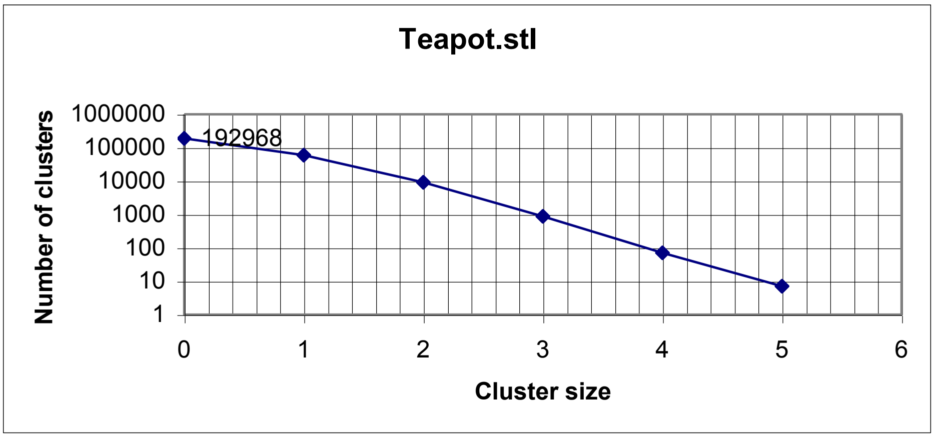

FIG. 10 *Proposed hash function property for Teapot.stl data file*

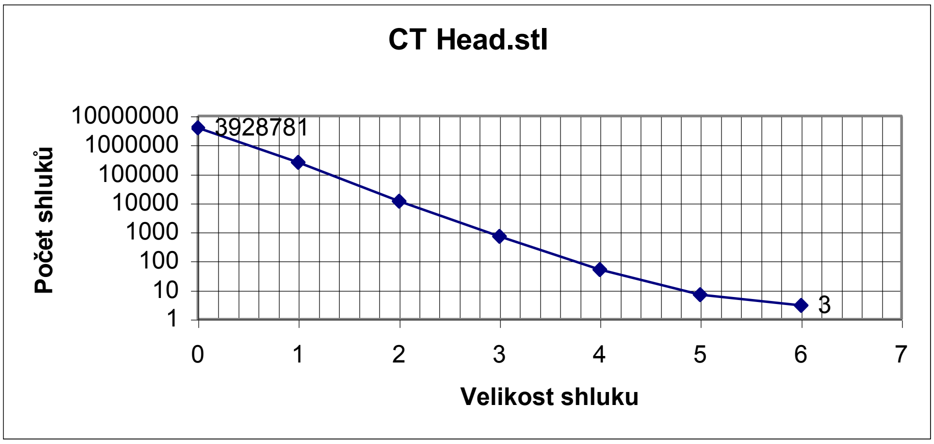

FIG. 11 *Proposed hash function property for if the table is longer 4-times*

5. Conclusion. This work has been part of the Modular Visualization Environment (MVE) project. The proposed hash function uses the advantage of large memory available today as well as the properties of the proposed hash construction. The hash function has been tested on several data sets and proved similar behavior. The size of the tested files varied from 10^5 to 2.10^6 of triangles and the proposed hash function proved behavior stability.

6. Acknowledgment. The author would like to thanks to all that contributed to this work, especially to MSc. and PhD. students at the University of West Bohemia in Plzen who have stimulated this thoughts and development of this new algorithm. This paper benefits from several discussions with them a lot. Special thanks belong to anonymous reviewers of this paper as they shared some valuable insights on this problem solution. Theirs invaluable critical comments and suggestions improved the manuscript significantly.

REFERENCES

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

[2] Kofrhage,R.R., Gibbs,N.E.: Principles of Data Structures and Algorithms with Pascal, Wm. C. Brown Publishers, 1987.