Computer Graphics International 2001, Honk Kong, China, IEEE Proceedings, ISSN 1530-1052, pp.167-174, 2001.

The Hash Function and the Principle of Duality

Václav Skala, Martin Kuchař
Department of Computer Science and Engineering[1]
University of West Bohemia, Univerzitní 9, Box 314
306 14 Plzeň, Czech Republic
{Skala,Kuchy}@kiv.zcu.cz *http://herakles.zcu.cz/{~skala,~kuchy}*

Abstract

An algorithm complexity is a very crucial issue in the algorithm design, especially if large data sets are to be processed. Data search is very often used in many algorithms and hash function use gives us a possibility to speed up the process significantly. Nevertheless, it is very difficult do design a good hash function especially for geometric applications. This paper describes a new hash function, its behaviour and use for non-trivial problems. Some problems can be solved effectively using the principle of duality and the hash data structure. Also some problems that cannot be solved in Euclidean space can be solved if dual representation is used and some examples are presented, too.

1. Introduction

Hash functions and their use are often used in computer science. The hash function enables to convert the key value to an address where the value is stored. This scheme enables to reduce a complexity of the search operation to *O(1)* complexity in general, if no collisions (two different key values that are transformed to the same address) occur. Data bucketing solves the collision problems, see fig.1.

This technique enables to speed up the searching operation significantly especially for large data sets. One typical use of this method is a triangular mesh reconstruction from the given set of triangles. The fundamental problem is to find all triangles, which share the same vertex. The hash function use reduces the run-time complexity from *O(N lgN)* to *O(N)* expected complexity and offers a significant speed up.

It must be pointed out that the hash function use enables us to decrease complexity of the search operation. Let us imagine that we want to find whether the given point x is in the set $X = \{x_i\}_{i=1}^{N}$. The complexities for different algorithms are shown in tab.1.

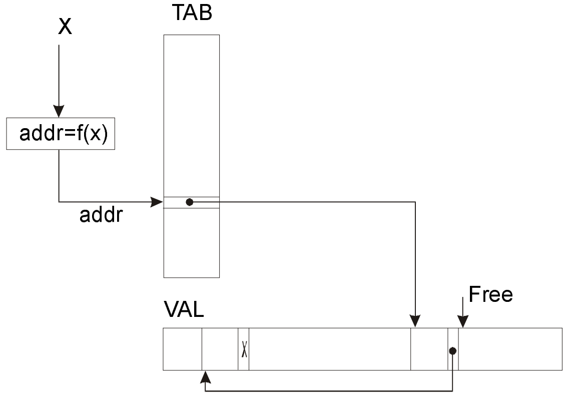


Figure 1. Hash data structure

	Pre-processing	Run-time
Linear search	none	*O(N)*
Binary search	*O(N lgN)**	*O(lgN)*
Hash function +	*O(N)*	*O(1)***

* x values in the set X must be sorted
** if ideal hash function is used - all buckets have the length equal to 1
+ requires additional memory for the table TAB

Table 1.

It means that the hash function enables us to decrease time needed to answer this query. It should be noted that the Binary search algorithm with the *O(lgN)* complexity cannot be used effectively if the set X is built incrementally as the elements of the set X are not ordered.

A hash function can be used effectively for triangular mesh reduction from the set of triangles, e.g. from the data in the STL format as well.

[1] This work has been supported by the Ministry of Education of the Czech Republic - project MSM 235200005

Proceedings of Computer Graphics International (CGI '01)
1530-1052/01 $10.00 © 2001 **IEEE**

The STL format, which is often used in CAD applications, is a data standard for polygonal mesh exchanges. The STL format offers just a set of polygons, mostly triangles, which represent a surface of the given object. There is no information which triangles share the given vertex etc.

For an effective manipulation with triangular surfaces the triangular mesh representation is needed, especially for:

- computation of normal vectors in vertices, e.g. for shading purposes,
- finding neighbours of triangles and sharp edges detection,
- effective triangular mesh reduction or decimation [Franc00a],
- memory reduction as the co-ordinates of all points are to be stored just ones.

There have been several approaches to triangular mesh reconstruction from the given set of triangles, one of those uses a hash function to speed up the reconstruction process significantly, see [Glass94a] for details.

2. The original hash function

The original hash function was introduced for triangular mesh reconstruction [Glass94a] as:

$$Index = \lfloor 3{*}x + 5{*}y + 7{*}z \rfloor \ \mathbf{mod}\ size$$

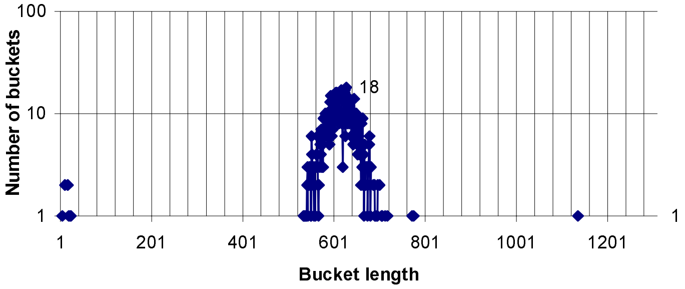
where: x, y, z are co-ordinates of a point,
q decimal digits to be distinguished,
size is the size of the hash table.

This hash function uses a very simple formula recommended for small or medium data sets. The fundamental requirement for any hash function is that maximal and average bucket's length should be as low as possible.

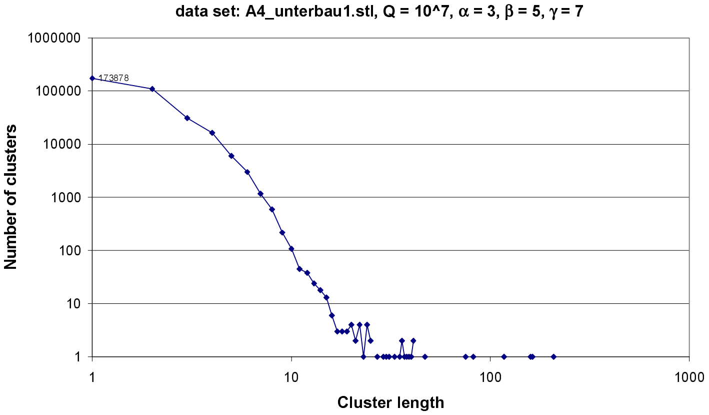
File name	No Triangles	Vertices No	Avg. bucket	Max. bucket
East EU	6 436 808	3 222 001	202.3	12 093
Karoserie1	2 264 544	1 213 783	2.2	12
Central EU	1 605 608	804 601	152.2	5 116
A4_unter	1 000 790	618 865	1.8	208

Typical characteristics of the original hash function for the triangular mesh reconstruction
Table 2.

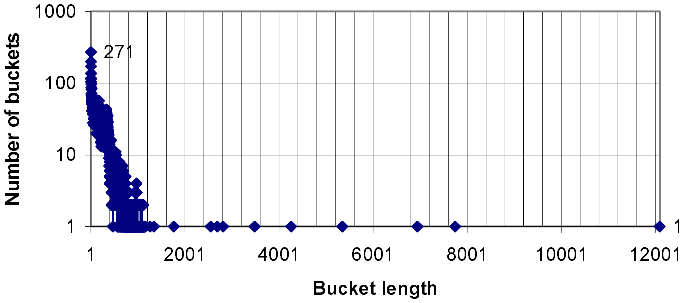
Nevertheless for the larger data sets this function generates long buckets, see tab.2. It is necessary to point out that the choice of *q* has a significant influence to the hash function behaviour. The obtained results strongly depend on the fractional part of co-ordinates vertices, see fig.3 - 5.



Data set: for A4_unterbau, *q = 3*
Figure 3. Bucket length distribution



Data set: for A4_unterbau, *q = 7*
Figure 4. Bucket length distribution



Data set: East EU, *q = 7*
Figure 5. Bucket length distribution

The experiments proved, that *q = 3* (used in [Glass94a]) is not satisfactory because long buckets were generated. Better results were obtained for *q = 7* (see fig.4), but there is no automatic option of that. Also the bucket length distribution is not acceptable, see fig.3 - 5.

An obvious question is why those buckets are so long and how the hash function for those geometric applications should be designed. The fig.6 shows a typical example of the surface represented by a triangular mesh, which was used for experiments, see [Skala00a] for details.

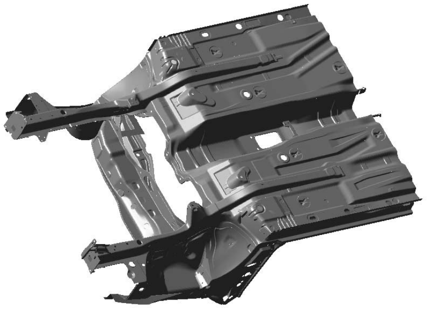


Figure 6. Part of the auto chassis data set: A4_unter

It can be seen that the data processing can be significantly speeded up if the length of all buckets is as small as possible. Tab.2 shows that there is a high potential for additional speed up if there is a better bucket length distribution.

This was the reason, why the exploration of the hash coding principle started.

3. The proposed hash function

The hash function is a function that transforms a value to an address where the value is stored, see fig.1. Data analysis proved that:

- it is not reasonable to remove fractional part from co-ordinate values as it helps us to distinguish co-ordinates better,
- it is necessary to remove all coefficients or parameters that somehow depend on the data set,
- to use available memory as much as possible to get longer hash table, i.e. shorter buckets in general,
- the hash function should not be static - it should be flexible according to the available memory and to the data set to be processed, but generally the size of the hash table can be fixed according to the number of vertices to be stored.

When considering required properties of the hash function, several functions have been derived. In general the hash function is constructed as (similarly as in [Glass94a]):

$$\xi = \alpha * x + \beta * y + \gamma * z$$

$$Index = \lfloor \xi * C \rfloor \ \mathbf{mod}\ size$$

where: α, β a γ are coefficients of the hash function; usually values 3, 5 a 7 are taken,

size is the hash table length ($size = 2^k - 1$),

mod operator is represented as logical operation **and** (A **mod** 8 ≡ A **and** '0111'),

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

In order to get maximal flexibility of the hash function we must use the whole address space interval (in our case $<0, 2^{32} - 1>$), influence of C_1 coefficient, and maximum of available memory, influence of C_2 coefficient.

For simplicity let us assume that $\boldsymbol{x} \in < \boldsymbol{0}, \boldsymbol{x}_{max} >$, similarly for others co-ordinates. Then the maximal value ξ_{max} can be computed as

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

and the C coefficient must be determined as:

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

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

as the overflow operation in the *Index (address)* ξ computation must be avoided and the whole size of the table should be used. The advantage of this approach is that the hash function depends on the size of the data set size, actual data range in the data set and available memory.

It is known that the length of the table and estimated length of a bucket (number of collisions per a bucket) is in an empirical relation expressed by the *load factor* α, see [Kofrh87a]. The *load factor* is generally the ratio of the number of values to be stored and the length of the table TAB. If we consider the *load factor* $\alpha = 0,5$ we can expect bucket length about 2,5 and the length of the hash table *size* can be expressed as

$$size \geq \frac{N}{\alpha} = 2N$$

where: N is the number of vertices,

load factor - $\alpha = 0,5$ used; the lower value used the better spread out, i.e. shorter buckets

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

$$k \geq \lceil \lg_2(2N) \rceil$$

4. Experimental Results

The proposed hash function has been tested on different non-trivial data sets. Tab.3 presents the typical behaviour on some selected data sets. The relation of the bucket length and number of buckets are presented on fig.7-8.

File name	Vertices No	Avg. bucket	Max. bucket
East EU	3 222 001	1.4608	22
Karoserie1	1 213 783	1.1524	7
Central EU	804 601	1.2336	7
A4_unter	618 865	1.3244	8

Typical behaviour of the proposed hash function
Table 3.

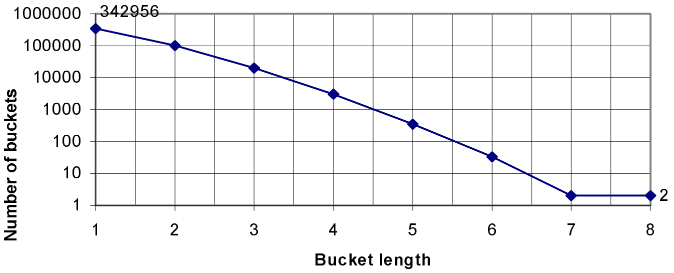


Figure 7. Bucket length distribution of the new hash function - data set: A4_unter

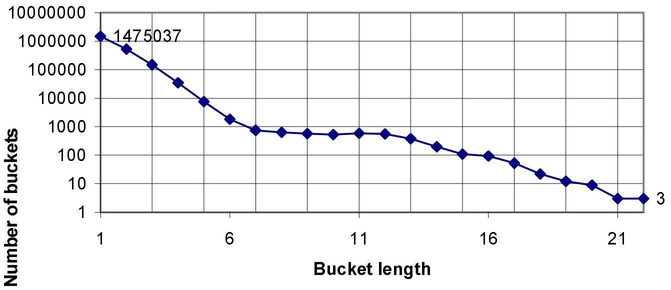


Figure 8. Bucket length distribution of the new hash function - data set: East EU

It can be seen that the maximal bucket length is limited to the length in the order of 10, that is a very good result. Also the number of buckets decreases with the bucket length and that is a very good property of the proposed hash function.

Over 150 industrial files were examined and similar results were obtained for all non-trivial data sets. The proposed function was used to speed up triangular mesh reconstruction from the given set of triangles, actually from the STL format and experiments proved the expected speed-up [Skala00a].

Nevertheless in some cases the co-ordinate range is not known. In this case the hash function can be easily modified so that co-ordinates are transformed by the function [Pasko95a]:

$$x' = \left(\frac{x}{|x| + k} + 1 \right) * 0,5$$

where: k is a parameter.
This function transforms the interval of $x \in (-\infty, \infty)$ to the interval $x' \in (0, 1)$. This transformation preserves the stability of the hash function behaviour and it is applied for the y and z co-ordinates as well. The hash function is then constructed similarly as if $\boldsymbol{x}_{max} = \boldsymbol{1}$, now.

5. Algorithm Analysis

The new hash function uses the advantage of large memory and the known interval of co-ordinates of the points processed. The function has a different sensitivity to each co-ordinate.

Let us introduce the coefficient ν as

$$\nu = \frac{\text{Average cluster length}_{\text{original}}}{\text{Average cluster length}_{\text{new}}}$$

that represents the expected speed-up (in average) for answering a query, see tab.4.

File name	Orig. function	New function	ν
East EU	202.30	1.46	202.30
Karoserie1	2.20	1.15	2.20
Central EU	152.20	1.23	152.20
A4_unter	1.80	1.32	1.80

Comparison of average bucket lengths of the original and the proposed hash functions[2]
Table 4

Let us define the coefficient η as

$$\eta = \frac{\text{Max. cluster length}_{\text{original}}}{\text{Max. cluster length}_{\text{new}}}$$

It represents the ratio of the maximal bucket lengths of the original and new hash functions., see tab.5.

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. Tab.4 - 5 show

[2] The East EU and Central EU files were generated from the Digital Terrain System (GTOPO30), the Karoserie1 and A4_unter files are “real life” data courtesy of Skoda-Auto Comp., Czech Republic

the differences between original and proposed hash functions.

It is obvious that we can obtain very high speed-up for a simple query whether the given point x is in the set $X = \{x_i\}_{i=1}^{N}$, see coefficients η in tab.5.

File name	Orig. function	New function	η
East EU	12 093	22	549.69
Karoserie1	12	7	1.71
Central EU	5 116	7	730.86
A4_unter	208	8	26.00

Table 5. Comparison of maximal bucket length of the original and the proposed hash functions

The proposed hash function has the following advantages:

- stable behaviour,
- short maximal bucket length,
- number of buckets decreases with the bucket length nearly monotonically,
- does not significantly depend on:
  - parameters defined by user (parameter q),
  - actual co-ordinate values,
- is flexible according to the number of vertices processed and co-ordinate values,
- gives us faster solution for triangular mesh reconstruction, see coefficient ν in tab.4

The behaviour of the new hash function leads to the question where the hash function and hash data structures can be used in order to speed up solution of problems. One very simple and very often used is the test whether a given point is in the given set of points.

```
{ $N$ - number of points required }
$X := \emptyset$;  { empty set }
$k := 0$;  { actual number of points in the set $X$ }
while $k < N$ do
begin   $x :=$ random; {random generator }
        if $x \notin X$ then
        begin k := k + 1; $X = x \cup X$ end
end
```

Algorithm 1

6. Test Point-in-a Set

There are many applications where the point-in-a set test is needed. As a simple example can be a random number generator, where all generated numbers must be different. Algorithms usually used are based on a sequential search on the set already generated and if the value is not in the set the generated value is added to this set, see alg.1.

This test is not necessarily restricted to the one-dimensional space. There is a possibility to use also the space subdivision technique using regular space subdivision etc. but the hash function can be used as well.

It can be seen that the **sequential** algorithm is of $\boldsymbol{O(N^2)}$ **complexity** and it is a quite time consuming process for large data sets generation. The test $x \in X$ is of $O(N)$ complexity itself as the sequential search can be used only because the set X is not ordered and cannot be kept ordered effectively.

If the hash coding technique is used, the test $x \in X$ is of $O(1)$ expected complexity only, see tab.5. Also the $X = x \cup X$ operation is of $O(1)$ complexity, if the hash function is an ideal hash function with the bucket length equal to 1. This is not true in general and therefore this $O(1)$ complexity is the **expected complexity**, only. Nevertheless the experiments with the real and non-trivial data, see chapter 2 and 3, proved that this behaviour can be expected with high probability. It means that the whole algorithm is of $\boldsymbol{O(N)}$ **expected complexity** if the hash data structure used with an ideal hash function.

It is necessary to know expected properties of the given data set for the hash function design. If the *random* generator produces values with the uniform distribution from the interval < 0 , 1 > the hash function can be design as

$$Index = \lfloor x * C \rfloor \,\mathbf{mod}\, size$$

where: C is a scaling factor determined similarly as in the chapter 3.

It can be seen that this approach is not restricted to the E^1 case, but also can be used for E^n case generally, if handled properly, see chapter 3.

As a direct consequence of this it is possible to answer queries like:

- Is the point $\boldsymbol{x} \in E^n$ in the given set of points $\{ \boldsymbol{x}_i \}$?
- Is the given circle $k \in E^2$ in the given set of circles $\{ k_i \}$?
- Is the given sphere $s \in E^3$ in the given set of spheres $\{ s_i \}$?

Those queries can be answered with $\boldsymbol{O(1)}$ **expected** complexity!

Last two queries can be answered as the query is based on an equation:

$$(\mathbf{x} - \mathbf{x_s})^T (\mathbf{x} - \mathbf{x}_s) - R^2 = 0$$

where: $\boldsymbol{x}_s$ is the centre of a circle or a sphere
R is the radius of it.

It means that the hash function can give us a good tool for object localisation in the set of scattered objects in general.

If the *random* generator produces values with Gauss distribution the situation is a little bit more complicated as we have to handle an unlimited interval. In this case the following transform function can be used (see chapter 4):

$$x' = \left(\frac{x}{|x| + k} + 1 \right) * 0{,}5$$

and $\qquad Index = \lfloor x' * C \rfloor \ \mathbf{mod}\ size$

where: k is a parameter.

The general Gauss distribution function is defined as:

$$\varphi(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where: μ is the mean value,
σ is the dispersion.

The data sets can be recomputed so that the mean value $\mu = 0$, easily.

For the Gauss distribution function with the dispersion $\sigma = 1$ is the optimal value $k_{opt} = 1{,}1$ if the ratio of maximal bucket length/average bucket length is used as the optimality criterion. If the dispersion $\sigma \neq 1$, the value k should be taken as $k = k_{opt} * \sigma$. Those results have been obtained by numerical optimisation process.

7. Principle of Duality

It is well known that several problems can be solved easily if a dual representation is used [Johns96a]. The principle of duality is a general principle very often used in practice and its usage can bring quite new solutions [Nielse95a], [Stolfi89a], [Kolin94a]. Let us assume that the following dualities can be defined according to the tab.6.

	Euclidean space	Dual space
E^2	point	line
	line	point
E^3	point	plane
	plane	point

Table 6. Possible dualities

It means that the queries such as:

- Is the given line $p \in E^2$ in the given set of lines $\{ p_i \}$?
- Is the given plane $\rho \in E^3$ in the given set of planes $\{\rho_i\}$?

can be answered with ***O(1)* expected** complexity!

It is necessary to note that some additional conditions must be applied, e.g. coefficients of lines or planes must be "normalised", as they form one parametric set actually.

Let us consider a line $p \in E^2$ defined as

$$a * x + b * y + c = 0$$

It can be seen that if this equation is multiplied by any constant $d \neq 0$, we get different equations that represent the same line p. Now we can handle with lines similarly as points with co-ordinates $[a , b]$.

It is necessary to note that some queries based on principle of duality cannot be answered directly in the Euclidean space.

Nevertheless all those tests presented above can be considered as "very academic" or "pure theoretical" ones because the arithmetic precision is limited and test for exact equality is not directly applicable.

8. Point-In-a Set as a Range Test

The range test or ε - test is very often used to find an element close to the given value. This test is possible if there is a *monotonic dependency* between the value and the element position in the hash table. The hash function can be also used if the hash function is designed so that the operation **mod** is not used and the C constant is to be properly chosen.

As a direct consequence of this is that it is possible to answer queries like:

- Is there any point x_i in the set X that is in $x \pm \varepsilon \in \{ x_i \} = X$?
- What is the minimal circle k_i from a set of circles $\{ k_i \}$ in which the given point x lies?

If we use the principle of duality, the following queries might be answered (if we define properly what ε means) as well:

- Is there any line p_i in the set of lines $\{ p_i \}$ that is in $p \pm \varepsilon \in \{ p_i \}$?
- Is there any plane ρ_i in the set of planes $\{\rho_i\}$ that is in $\rho \pm \varepsilon \in \{ \rho_i \}$?

The principle is very simple indeed because the hash function is monotonic. It is necessary to compute:

$$Addr_1 = F(x - \varepsilon) = F(x_1)$$
$$Addr_2 = F(x + \varepsilon) = F(x_2)$$

and then the hash table must be searched sequentially in the interval $< Addr_1 , Addr_2 >$. It means that the higher ε is the longer sequential search must be made if no element of TAB is used, see fig.9, i.e. there is no pointer to the VAL data structure.

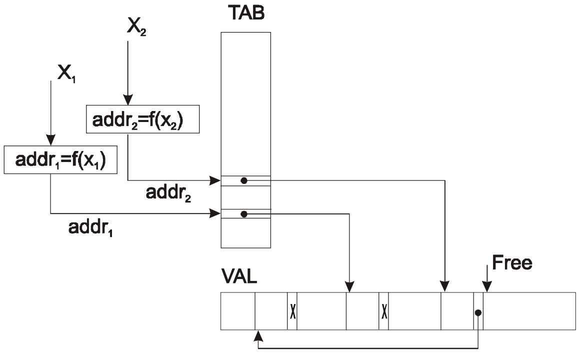


Figure 9. The ε-test with the hash function

This approach can be used directly for answering the query:

- Which point forms the smallest circle circumscribed for the given edge, see fig.10?

This can be useful for Delaunay triangulation construction under some circumstances.

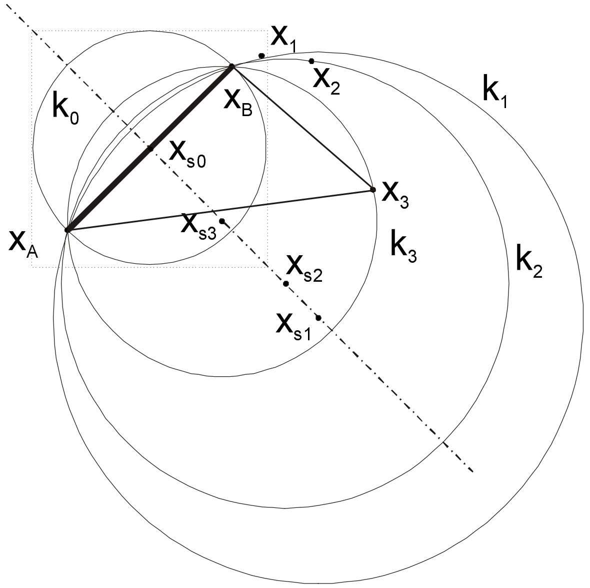


Figure 10.

The problem can be solved as follows:

1. Find the middle point $\boldsymbol{x}_{s0}$ for the given edge $\boldsymbol{x}_A,\boldsymbol{x}_B,$
2. Compute location *addr* = *f*($\boldsymbol{x}_{s0}$) and search sequentially the table TAB in both direction symmetrically for the first point $\boldsymbol{x}_1$ (we expect the similar sensitivity of the hash function in *x* and *y* axes),
3. The point $\boldsymbol{x}_1$ defines the circle $\boldsymbol{k}_1$; compute the center of this circle $\boldsymbol{x}_{s1}$
4. The table TAB must be searched sequentially for all addresses *addr* = *f*(***x***), where: $\boldsymbol{x} \in <x_{s0}, x_{s1}+R_1> x <y_{s0}, y_{s1}-R_1>$ for our case as it depends on the geometric positions of the points $\boldsymbol{x}_A$ and $\boldsymbol{x}_B,$
5. A point with the minimal circle will be taken from the points found during the sequential search.

This algorithm is a little bit more complicated, but there is a hope that this approach can decrease the algorithm complexity significantly. It is necessary to note that the step 2 is more complicated in practical use as the coefficients α and β used in the hash function are different.

There is a believe that there are other tests that can be made using the hash function and duality principle, too.

9. Conclusion

The new approaches for hash function construction and the hash data structure use have been developed. The proposed hash function has been tested on many non-trivial data sets. It proved very good properties and stability for large data sets with quite different geometric characteristics. Experiments also proved that the hash function is convenient for the triangular mesh reconstruction and for other purposes, too.

The main advantage of the hash data structure use for geometric purposes is that it gives us a possibility to answer non-trivial queries with *O(1)* **expected** complexity and answers for the **ε-tests** as well.

The authors believe that the proposed hash function use gives higher flexibility than others techniques used, such as space subdivision, decision trees etc.

In the nearest future it is expected that:

- those approaches will be examined with the existing geometric algorithms using Euclidean space as well different dual space representations,
- the influence of the hash function coefficients α, β, γ to the bucket length will be studied,
- a comparison of the hash function and different space subdivision properties will be made.

Acknowledgement

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 function and its use. This paper benefits from several discussions with them a lot. Special thanks belong to Dr.I.Kolingerová for critical comments. Thanks belong also to anonymous reviewers of this paper as they shared some valuable insights on this problem solution. Theirs invaluable comments and suggestions improved the manuscript significantly.

References

[Franc00a] Franc,M., Skala,V.: Triangular Mesh Decimation in Parallel Environment, EUROGRAPHICS Workshop on Parallel Graphics and Visualization, Girona, Spain, pp.39-52, ISBN 84-8458-025-3, 2000.

[Glass94a] Glassner,A.: Building Vertex Normals from an Unstructured Polygon List, Graphics Gems IV, pp.60-73, Academic Press,Inc., 1994.

[Johns96a] Johnson,M.: Proof by Duality: or the Discovery of "New" Theorems, Mathematics Today, pp.171-174, November/December, 1996.

[Kofrh87a] Kofrhage,R.R., Gibbs,N.E.: Principles of Data Structures and Algorithms with Pascal, Wm.C.Brown Publ., 1987.

[Kolin94a] Kolingerova,I.(supervisor V.Skala): Dual Representation and its Use in Computer Graphics (in Czech), PhD thesis, Univ.of West Bohemia, Plzen, Czech Republic, 1994.

[Nielse95a] Nielsen,H.P.: Line Clipping Using Semi-Homogeneous Coordinates, Computer Graphics Forum, Vol.14, No.1, pp.3-16, 1995.

[Pasko95a] Pasko,A., Adzhiev,V., Sourin,A., Savchenko,V.: Function Represenatation in Geometric Modeling: Concepts, Implementation and Applications, The Visual Computer, Vol.11, pp.429-446, Springer Verlag, 1995.

[Skala96a] Skala,V.: Line Clipping in E^2 with O(1) Processing Complexity, Computers & Graphics, Vol.20, No.4, pp.523-530, 1996.

[Skala00a] Skala,V., Kuchar,M.: Hash Function for Geometry Reconstruction in Rapid Prototyping, Algoritmy2000 proceedings, Slovakia, pp.379-387, 2000

[Stolfi89a] Stolfi,J.: Primitives for Computational Geometry, SRC DEC System Research Center, Research Report 36 (PhD. Thesis), 1989

Appendix

The Fig.A.1 shows the bucket length behaviour for the proposed hash function for selected data files (numbers present number of vertices of the triangles after triangular mesh reconstruction). The length of the hash table was taken as parameter.

It can be seen that the length of buckets do not change significantly for the recommended or longer hash table (chosen as 2-times or 4-times longer), but when table length is shorten (to 1/2 or to 1/4 of the recommended length) the length of buckets starts to grow. The graph also proofs the stability of the proposed hash function properties for quite different triangular meshes.

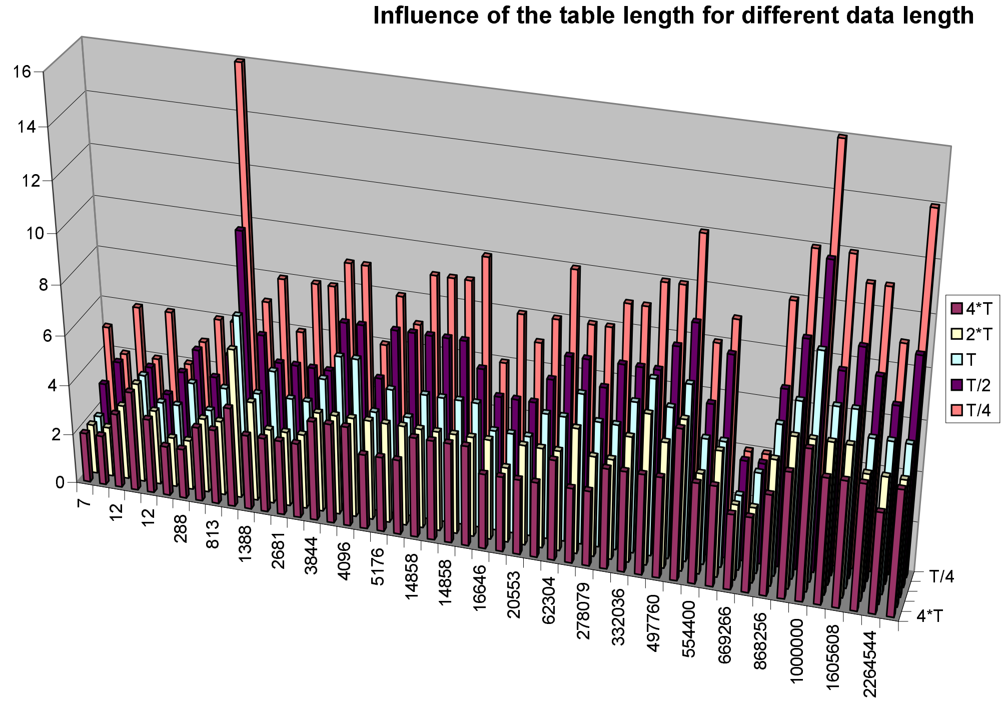


Figure A.1. Maximal bucket length dependence on the number of vertices and the hash table length

Parameters: $(\alpha, \beta, \gamma) = (\pi, e, \sqrt{2})$, T is the recommended table length