Duality and Robust Computation

VACLAV SKALA
Department of Computer Science and Engineering
Faculty of Applied Sciences, University of West Bohemia
Univerzitni 8, CZ 306 14 Plzen
Czech Republic
skala@kiv.zcu.cz http://www.VaclavSkala.eu

Abstract: - Robustness of computations in engineering is one of key issues as it is necessary to solve technical problems leading to ill conditioned solutions. Therefore the robustness and numerical stability is becoming a key issue more important that the computational time. In this paper we will show selected computational issues in numerical precision, well known cases of failures in computations. The Euclidean representation is used in today's computations, however the projective space (an extension of the Euclidean space) representation leads to more compact and robust formulations and to matrix-vector operations supported in hardware, e.g. by GPU.

Key-Words: - Euclidean space, projective space, homogeneous coordinates, duality, intersections, barycentric coordinates, planes intersection, Plücker coordinates, numerical precision.

1 Introduction

Data processing is one of the main fields in computer science. Data processing itself can be split to two main areas:

- processing of textual data
- processing of numerical data

Nowadays, computers use binary system for information and data representation.

The above mentioned main two areas are quite different, but have many common algorithms, e.g. hashing. In the case of textual data we have "unlimited" dimensionality ("unlimited" length of a string) but limited interval of values (usually given by a number of symbols in the given alphabet). On the contrary in the case of numerical or geometrical data we have a limited dimensionality (usually 2 or 3 in the case of E^2 or E^3) but "unlimited" interval of values (usually $(-\infty, \infty)$).

Numerical data processing and numerical computations bring quite significant difficulties due to the limited precision of a real number representation as the floating point representation offers only a limited length of a mantissa and exponent [49]. Today's programming standard languages do not offer constructions for computation with an "arbitrarily" long integers or "unlimited" mantissa length (Algol 68 had a construction **long** that could extend the basic data type, e.g. **long** .... **long real** etc.). Unfortunately it leads to numerical problems and possibly to disasters in engineering applications. There are also problems connected to uncontrolled overflow, infinities and NaN results.

It should be noted that the majority of computer science students and programmers are NOT AWARE of those aspects at all.

2 Numerical Precision and Robustness

Numerical data processing and numerical computation is the heart of nearly all engineering problems solution. On the other hand it seems to that in the engineering courses there is no attention given to the numerical precision in connection with the robustness of algorithms.

It can be seen from the floating point representation that the absolute precision depends on the actual exponent significantly, as the precision is given by the length of the mantissa **multiplied by the exponent**. As the mantissa is of the given length, not all numbers even rational numbers can be represented in a computer; of course irrational numbers cannot be stored in any case. It means that a value x is somehow modified in order to fit into the actual floating point representation. It means that a stored value x represents actually an interval $[a, b]$, i.e. any value from this interval is represented in a memory as one value x.

As values are used in numerical operations it is necessary to ask, at least in the case of basic arithmetic operations, what is the ***influence to the precision***?

Let as assume that we have two numbers x and y x = [a,b], y = [c,d].

The following interpretation of the basic arithmetic operations demonstrate how actual precision is defined.

- $x + y$ = [a + c, b + d] $x - y$ = [a - d, b - c]
- $x \times y$ = [min(ac, ad, bc, bd), max(ac, ad, bc, bd)]
- x / y = [min(a/c, a/d, b/c, b/d), max(a/c, a/d, b/c, b/d)] if $y \neq 0$

There are well known identities like

$$cos^2\alpha + cos^2\beta = 1 \qquad x^2 - y^2 = (x-y)(x+y)$$

However **these identities are not valid** if the floating point representation is used. For a computation of $x^2 - y^2$ it is better to use $(x-y)(x+y)$ due to better precision in evaluation, as if $|x| > |y|$ then $x^2 \gg y^2$ and therefore some last digits of the y^2 mantissa might be lost in the final subtraction. Also constructions like

if <float>=<float> **then**... **if** <float>≠<float> **then**…

should not be allowed in programming languages or at least a warning message should be generated. Usually this problem is "solved" by constructions

if abs (x - y) < *epsilon* **then** … or

if abs (x) < *epsilon* **then** q:= y / x **else** ERROR ,

but nobody knows what is the proper value of *epsilon*.

Let us explore a little bit the numerical problems on very simple examples, now.

Quadratic Equation Solution

Let us consider two formulations as follows [6]:

$$at^2 + bt + c = 0$$

The solution usually used is $t_{1,2} = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$

or if substituted $t = {}^1/_\tau \qquad \tau_{1,2} = \frac{2c}{-b \pm \sqrt{b^2-4ac}}$

However in some cases the "standard" formula can lead to incorrect results due to a limited number precision. If $b^2 \gg 4ac$ then it is recommended to use the following formula:

$$q = -(b + sign(b)\sqrt{b^2 - 4ac}\,)/2$$

$$t_1 = {}^q/_a \qquad t_2 = {}^c/_a$$

to get more reliable results.

It can be seen that even such a simple case might be quite sensitive to the numerical precision.

Function Value Computation

Computation of a function value is one of the basic common operations in engineering problems. However many programmers are not aware of the danger in the coding process. There seems to be two the most dangerous cases:

- division by a value close to zero, e.g. in an intersection computation of two nearly parallel lines
- addition or subtraction of two values with significantly different absolute value, e.g. recently mentioned $x^2 \pm y^2$.

As the result of this, the summation (repeated addition) result depends on the order of summation in general.

Let us explore one very interesting case [4] and some other interesting comments [1], [7], [28].

$$f(x,y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

The question is, what is the value of the function if it is evaluated at $x = 77617,\ y = 33096$ if different floating point precision is used.

$f = 6.33835\ 10^{29}$ in single precision
$f = 1{,}1726039400532$ in double precision
$f = 1{,}17260394005317863185883490452018\,38$ in extended precision

However even the result in the extended precision is incorrect and even the sign itself is incorrect. The correct! The correct result is "somewhere" in the interval of

$[-0{,}827396059946821368141165095479816\,29\mathbf{2005},$
$-0{,}827396059946821368141165095479816\,29\mathbf{1986}]$

if approx. 40 digits were used [4]. Of course this function is constructed in a special way, but it demonstrate that

- simple increase of precision does not guarantee the correctness of the result
- roundoff error has significant influence to for a limited floating point computation.

Detailed analysis of this function can be found in [1] and the correct result is

$$f(x,y) = -2 + \frac{x}{2y} = \frac{54767}{66192}$$

Unfortunately precision of the numerical results are significantly influenced by compilers properties and options used, as the optimization of the code is not considering the numerical stability issues.

Addition and Computational Order

So far we have dealt with "complicated cases", usually seen as "not practical". Power series summation is one of the very practical and often used computations. Let us imagine simple examples of summation if single precision is used [25]:

$$\sum_{i=1}^{10^3} 10^{-3} = 0.999990701675415$$

or

$$\sum_{i=1}^{10^4} 10^{-4} = 1.000053524971008$$

It can be seen that in the both cases the result should be one. The correctness in summation is very important in power series computations, e.g.

$$\sum_{n=1}^{10^6} \frac{1}{n} = 14.357357 \quad \text{or} \quad \sum_{n=10^6}^{1} \frac{1}{n} = 14.392651$$

It means that even for a small number of elements we do not obtain correct results.

Recursion

Recursion is very useful tool for finding a nice description of a problem solution, e.g. well known Tower of Hanoi, however if implemented directly it might causes some problems, like the stack overflow etc. The algorithm itself can be described as follows:

```
MOVE (A, C, n);      # MOVE (from, to, number) #
{MOVE(A,B,n-1); MOVE(A,C,1); MOVE(B,C,n-1)
}
```

This recursive elegant solution is simple to implement and only stack overflow can be expected; Iterative solution is known as well. The recursive definition usually leads to two main searching strategies in implementation, i.e. depth first search or breath first search. Let us explore recursive definition of the well known Ackermann defined as function [20]:

$$A(m,n) = \begin{cases} n+1 & if\ m = 0 \\ A(m-1,1) & if\ m > 0\ and\ n = 0 \\ A\big(m-1, A(m,n-1)\big) & if\ m > 0\ and\ n > 0 \end{cases}$$

This function is simple, but the problem its computation as the value of the function grows very fast as

$$A(4,4) = 2^{2^{2^{65536}}} = 2^{2^{10^{197296}}}$$

As the computation is made in integers, no overflow is detected at all.

However engineering applications are more oriented to computation with numbers in floating point representation.

Continuous Fractions

There is one very interesting approach based on continuous fractions. It enables to represent even irrational numbers in some cases. The basic definition can be described as:

$$x = b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{b_4 + \cfrac{a_3}{\dots}}}}$$

For generalized continuous fractions $a_i \neq 1$ and we can express as $\pi = [3; 7,15,1,292,1,1,1,2,1,3,1 \dots]$ if $a_i = 1$

As $\pi = 4\ arctan\ (1)$ then π can be expressed as [23]

$$\pi = \cfrac{4}{1 + \cfrac{1^2}{3 + \cfrac{2^2}{5 + \cfrac{3^2}{\dots}}}}$$

Such a number representation is quite different and detailed description can be found in [5].

We have presented some selected fundamental issues in numerical computations that have direct influence to results of numerical computation.

There is a significant question how today's computations are reliable and robust as we are using a continuous mathematical models, but using discrete systems for physical phenomena representation; number of digits for a number representation is limited. Only very careful coding with regard to numerical errors can prevent disaster situations and possible losses on humans.

Matrix Inversion

Matrix inversion is very often used in solution of engineering problems. However in many cases the matrix is ill conditioned and the results are not checked to the correctness of the solution. Many libraries available just return a matrix, which might be far from the matrix inverted we would expect without any message or warning message.

Let us assume a matrix inversion as

$$\boldsymbol{Ax} = \boldsymbol{b} \qquad \boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$$

and the Hilbert's Matrix

$$\boldsymbol{H}_{ij} = \frac{1}{i+j-1}$$

then the inversion of the matrix is known in the analytical form and can be expressed as

$$\boldsymbol{H}_{ij}^{-1} = (-1)^{i+j}(i+j-1) \binom{n+i-1}{n-j}\binom{n+j-1}{n-i}\binom{i+j-2}{i-1}^2$$

The inversion of the Hilbert's matrix can be used to evaluate algorithms or available numerical library for the stability and correctness of results delivered. Matrix inversion and a linear system of equations can be solved effectively without division operation if projective geometry is used [8], [9].

3 Numerical Disasters

There are famous examples of numerical disasters. When reading the original reports and followed comments and details one must be really surprised how simple errors occur and should be worried what could happen in complex problems solution. Let us shortly explore some "traditional" cases.

The following is a modified excerpt from public resources [21], [22].

Explosion of Ariane 5

An Ariane 5 rocket was launched by the European Space Agency (ESA) on June 4, 1996. The development cost over \$7 billion. The rocket exploded after lift-off in about 40 sec. Destroyed rocket and cargo were valued at \$500 million. The cause of a failure was a software error in inertial reference system.

Patriot Missile Failure

The system was originally designed in mid-1960 for a short and flexible operation and for intercepting cruise missiles running at MACH 2 speeds. It was used to intercept the Scud missile running at MACH 5. The computation of intercepting and hitting was based on time counting with 24 bits integers with the clock of 1/10[s] and speed computation in floats. The clock setting to 1/10[s] was a critical issue and not acceptable even for application in sport activities at that time. Unfortunately $1/10 = 1/2^4+1/2^5+1/2^8+1/2^9+1/2^{12}+....$ and therefore the error on 24 bits is about 0.000000095 and in 100 hours the error is 0.34. As the Scud flies at MACH 5, the error was actually 687[m] and the missile was out of the "range gate" (a space window) area.

Offshore Platform Sinking

Another well known example is the Sleipner offshore platform sinking. It should be noted that the top deck is about 57 000 tons, drilling and support equipments weight about 40 000 tons and the deck provides an accommodation for about 200 people.

The Sleipner platform structure was "optimized" using a finite element system and the shear stresses were underestimated nearly by 50%. It led to serious cracks in the structure and water leakage that the pumps were unable to cope with. The sinking of the platform estimated cost is about \$700 million.

These examples show how danger the numerical errors can be.

4 Intersection of two planes

The Euclidean space representation is used in today's computations with the floating point representation. Unfortunately the imprecision of the floating point computations is given by a number of mantissa digits that is limited. However the robustness of algorithms is more connected with the mathematical formulation and the actual implementation as well.

In many cases the Euclidean representation leads to unnecessary computations that even decrease the computational precision. The division operation is heavily used in engineering computations and it decreases the precision of computation significantly. There is a question whether the division operation can be eliminated or at least postponed within the computational pipeline. In geometry, the projective representation is a way how things could be made simple, robust and easy to implement.

Plücker Coordinates

Let us consider two points in the homogeneous coordinates:

$$\boldsymbol{x}_1 = [x_1, y_1, z: w_1]^T \qquad \boldsymbol{x}_2 = [x_2, y_2, z_2: w_2]^T$$

The Plücker coordinates l_{ij} are defined as follows:

$$\begin{array}{ll} l_{41} = w_1x_2 - w_2x_1 & l_{23} = y_1z_2 - y_2z_1 \\ l_{42} = w_1y_2 - w_2y_1 & l_{31} = z_1x_2 - z_2x_1 \\ l_{43} = w_1z_2 - w_2z_1 & l_{12} = x_1y_2 - x_2y_1 \end{array}$$

It is possible to express the Plücker coordinates as

$$\mathbf{L} = \mathbf{x}_1\mathbf{x}_2^T - \mathbf{x}_2\mathbf{x}_1^T$$

where: $l_{ij} = - l_{ji}$ and $l_{ii} = 0$.

Let us define two vectors $\boldsymbol{\omega}$ and $\mathbf{v}$ as:

$$\boldsymbol{\omega} = [l_{41}, l_{42}, l_{43}]^T \qquad \boldsymbol{v} = [l_{23}, l_{31}, l_{12}]^T$$

The vector $\boldsymbol{\omega}$ represents the "directional vector", while $\mathbf{v}$ represents the "positional vector". The line p is then defined as

$$\mathbf{q}(t) = \frac{\mathbf{v} \times \boldsymbol{\omega}}{\|\boldsymbol{\omega}\|^2} + \boldsymbol{\omega}t$$

Let us imagine that we have to solve a dual problem, i.e. a line defined as an intersection of two given planes $\boldsymbol{\rho}_1$ and $\boldsymbol{\rho}_2$ in the Euclidean space:

$$\boldsymbol{\rho}_1 = [a_1, b_1, c_1: d_1]^T \qquad \boldsymbol{\rho}_2 = [a_2, b_2, c_2: d_2]^T$$

It is well known that the directional vector $\boldsymbol{s}$ of the line is given by those two planes as a ratio:

$$s_x : s_y : s_z = \begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix} : \begin{vmatrix} c_1 & a_1 \\ c_2 & a_2 \end{vmatrix} : \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}$$

that is actually the ratio $l_{23} : l_{31} : l_{12}$ if the principle of duality is used, i.e. vector of $[a_i, b_i, c_i: d_i]^T$ instead of $[x_i, y_i, z_i: w_i]^T$ is used, and it defines the vector $\boldsymbol{v}$ instead of $\boldsymbol{\omega}$. Applying the principle of duality as we can interchange the terms “point” and “plane” and exchange $\boldsymbol{v}$ and $\boldsymbol{\omega}$ in the equation for $\boldsymbol{q(t)}$ and we get:

$$\mathbf{q}(t) = \frac{\boldsymbol{\omega} \times \mathbf{v}}{\|\mathbf{v}\|^2} + \mathbf{v}t$$

If $\|\mathbf{v}\| = 0$ *then the given planes are* parallel.

It means that we have obtained the known formula for an intersection of two planes $\boldsymbol{\rho_1}, \boldsymbol{\rho_2}$ in the Euclidean coordinates:

$$\mathbf{q}(t) = \mathbf{q}_0 + \mathbf{n}_3\, t$$

where: $\mathbf{n}_3 = \mathbf{n}_1 \times \mathbf{n}_2$, $\boldsymbol{q}_0 = [X_0, Y_0, Z_0]^T$ and planes

$$\boldsymbol{\rho}_1 : \mathbf{n}_1^T \mathbf{x} + d_1 = 0 \qquad \boldsymbol{\rho}_2 : \mathbf{n}_2^T \mathbf{x} + d_2 = 0$$

The intersection point $\boldsymbol{X}_0$ of three planes in the Euclidean coordinates is defined as:

$$X_0 = \frac{d_2\begin{vmatrix} b_1 & c_1 \\ b_3 & c_3 \end{vmatrix} - d_1\begin{vmatrix} b_2 & c_2 \\ b_3 & c_3 \end{vmatrix}}{DET} \quad Y_0 = \frac{d_2\begin{vmatrix} a_3 & c_3 \\ a_1 & c_1 \end{vmatrix} - d_1\begin{vmatrix} a_3 & c_3 \\ a_2 & c_2 \end{vmatrix}}{DET}$$

$$Z_0 = \frac{d_2\begin{vmatrix} a_1 & b_1 \\ a_3 & b_3 \end{vmatrix} - d_1\begin{vmatrix} a_2 & b_2 \\ a_3 & b_3 \end{vmatrix}}{DET} \qquad DET = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}$$

An intersection of two planes is the case very often solved in computer graphics and computer vision. However the formula above is neither robust nor convenient for GPU use.

In the following a new formulation of intersection of two planes is presented and if the projective space is used for formulation, the solution is quite simple.

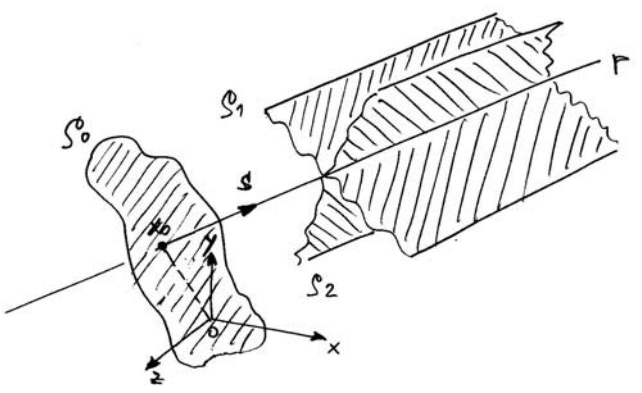

Figure 1: Intersection of two planes

Let us consider two planes $\boldsymbol{\rho}_1$ and $\boldsymbol{\rho}_1$ given as

$$\boldsymbol{\rho}_1 = [a_1, b_1, c_1 : d_1]^T \quad \boldsymbol{\rho}_2 = [a_2, b_2, c_2 : d_2]^T$$

and the normal vectors of those planes are

$$\boldsymbol{n}_1 = [a_1, b_1, c_1]^T \qquad \boldsymbol{n}_2 = [a_2, b_2, c_2]^T$$

It is obvious that a directional vector of a line is determined as an intersection of two planes $\boldsymbol{\rho}_1$ and $\boldsymbol{\rho}_1$ given as

$$\boldsymbol{s} = \boldsymbol{n}_1 \times \boldsymbol{n}_2$$

However, the “starting” point $\boldsymbol{x}_0$ of the line is determined in quite complicated ways, sometimes even not robustly enough and based on a user choice of some value, or on a solution of a system of linear equations leading to a standard formula given above.

The formula is quite “horrible” one and for students not acceptable as it is too complex and they do not see from the formula comes from.

However, there is a quite simple geometrical explanation and solution. So the question is how to find the “starting” point $\boldsymbol{x}_0$ of the line $\boldsymbol{p}$ given by two planes $\boldsymbol{\rho}_1$ and $\boldsymbol{\rho}_2$. If a robust solution is required a user should be prevented from a selection of some “parameters”.

Let us imagine that there exists a plane $\boldsymbol{\rho}_0$, whose normal vector is given as $\boldsymbol{s} = \boldsymbol{n}_1 \times \boldsymbol{n}_2$**.** It means that its position needs to be “fixed” in the space. As there is no other requirement on this plane, we can “fix” it so it passes through the origin of the Euclidean coordinate system, i.e. the plane $\boldsymbol{\rho}_0$ is given as

$$\boldsymbol{\rho}_0 = [a_0, b_0, c_0 : 0]^T$$

and the line $\boldsymbol{p}$ is orthogonal to the plane $\boldsymbol{\rho}_0$ This is resulting into a robust geometric position.

Now, the intersection point of those three planes is the “starting” point $\boldsymbol{x}_0$ we are looking for. Coordinates of the point $\boldsymbol{x}_0$ are determined by generalized cross-product as

$$\boldsymbol{x}_0 = \boldsymbol{\rho}_1 \times \boldsymbol{\rho}_2 \times \boldsymbol{\rho}_0$$

It is obvious that the point $\boldsymbol{x}_0$ is also the closest point on the line to the origin, too. The formula is very compact and it is suitable for GPU application.

From the formulation presented above, it can be seen that it is not only very simple, easy to understand and remember, but also easy to implement as well. As a result, the Plücker coordinates formulation of this problem solution is not needed when looking for such properties.

6 Conclusion

This paper briefly describes some problems in numerical computations, advantages of the projective space representation use and some well known disasters caused by impropriate use in numerical computations.

The projective space representation and reformulation of geometrical problems lead to more robust algorithms and simple formulations as shown above. The matrix-vector operations lead to more compact algorithms and due to the today’s hardware architecture also to additional computation acceleration, especially if GPU is used.

7 Acknowledgments

The author would like to express thanks colleagues at the University of West Bohemia, Plzen and to anonymous reviewers for their critical and constructive comments that helped to improve the manuscript.

This work was supported by the Ministry of Education – projects No.ME10060 and LH12181.

References:

[1] Cuyt,A., Verdonk,B., Becuwe,S., Kuterna,P.: A remarkable Example of Catastrophics Cancellation Unraveled, Computing 66, pp.309-320, 2011

[2] Jimenez,J.J., Segura,R.J., Feito,F.R.: Efficient Collision Detection between 2D Polygons, Journal of WSCG, Vol.12, No.1-3, 2003

[3] Johnson,M.: Proof by Duality: or the Discovery of "New" Theorems, Mathematics Today, December 1996.

[4] Leclerc,A.P.: Efficient and Reliable Global Optimization, PhD Thesis, Ohio State University, 1992

[5] Lorentzen,L.: Continued Fractions, Atlanties Studies in Mathematics for Engineering and Science, World Scientific Publ., 2008

[6] Press,W.H., Teukolsky,S.A., Vetterling,W.T., Flannery,B.P.: Numerical recipes in C, Cambridge University Press, 1999

[7] Rump,S.M.: Realiability in Computing, The role of Interval Methods in Scientific Computing, Academic Press, 1988

[8] Skala,V., Kaiser,J., Ondracka,V.: Library for Computation in the Projective Space, 6th Int.Conf. Aplimat, Bratislava, pp. 125-130, 2007

[9] Skala,V., Ondracka,V.: A Precision of Computation in the Projective Space, Recent Researches in Computer Science, pp.35-40, 15th WSEAS Int.Conference on Computers, Corfu, Greece, 2011

[10] Skala,V.: A New Line Clipping Algorithm with Hardware Acceleration, CGI'2004 conference proceedings, IEEE, Greece, 2004

[11] Skala,V.: Computation in Projective Space, MAMECTIS conference, La Laguna, Spain, WSEAS, pp.152-157, 2009

[12] Skala,V.: Duality and Intersection Computation in Projective Space with GPU support, Applied Mathematics, Simulation and Modeling - ASM 2010 conference, NAUN, Corfu, Greece, pp.66-71, 2010

[13] Skala,V.: Geometric Computation, Duality and Projective Space, ICGG 2010 conference, pp.363-364, Kyoto, Japan, 2010

[14] Skala,V.: Intersection Computation in Projective Space using Homogeneous Coordinates, Int.Journal on Image and Graphics, Vol.8, No.4, pp.615-628, 2008

[15] Skala,V.: Length, Area and Volume Computation in Homogeneous Coordinates, International Journal of Image and Graphics, Vol.6., No.4, pp.625-639, 2006.

[16] Skala,V: A new Approach to Line and Line Segment Clipping in Homogeneous Coordinates, The Visual Computer, Vol.21, No.11, pp.905-914, 2005

[17] Skala,V: Duality and Intersection Computation in Projective Space with GPU Support, WSEAS Trans.on Mathematics, Vol.9., No.6., pp.407-416, 2010

[18] Oh,E., Walster,W.G.: Rump's Example Revisited, Reliable Computing, Kluwer Academic Publ., Vol.9., pp.245-248, 2002.

[19] Yamaguchi,F., Niizeki,M.: Some basic geometric test conditions in terms of Plücker coordinates and Plücker coefficients, The Visual Computer, Vol.13, pp.29-41, 1997

WEB resources

[20] Ackermann function, http://en.wikipedia.org/wiki/Ackermann_function, <retrieved 2012-02-23>

[21] Arnold,D.A.: The sinking of the Sleipner offshore Platform, http://www.ima.umn.edu/~arnold/disasters/sleipner.html <retrieved 2012-02-02>

[22] Arnold,D.A.: Two disasters caused by computer arithemtic error, http://www.ima.umn.edu/~arnold/455.f96/disasters.html <retrieved 2012-02-02>

[23] Continuous Fractions http://www.numericana.com/answer/fractions.htm <retrieved 2012-01-29>

[24] IEEE-754 Data Format, http://en.wikipedia.org/wiki/IEEE_754-2008 <retrieved 2012-01-29>

[25] Tucker,W.: Automatic Differentiation, http://www.sintef.no/project/eVITAmeeting/2010/vn2010.pdf <retrieved 2012-01-29>