GPU Fast and Robust Computation for Barycentric Coordinates and Intersection of Planes Using Projective Representation

Vaclav Skala
Department of Computer Science and Engineering
University of West Bohemia
Plzen, Czech Republic
http://www.VaclavSkala.eu

Abstract—**This paper describes algorithms for fast and robust GPU computation of barycentric coordinates and intersection of two planes. The presented algorithms are based on matrix-vector operations which make the algorithms convenient for GPU or SSE based architectures. Also a new formula for finding the closest point of two planes intersection to the given point is given.**

Keywords-GPU, barycentric coordinates, two planes intersection, Plücker coordinates, closest point, homogeneous coordinates, projective space

I. INTRODUCTION

Intersection of two planes is frequently used in geometric computations, e.g. in set operations with geometric objects given by triangular meshes. In spite of the simplicity of the problem, the standard formulas for computations are not robust in cases when given planes are almost parallel or parallel. Also barycentric coordinates computation, which leads to a solution of linear equations, is very often used to solve geometrical problems.

In this paper standard approaches will be presented together with a new more robust approach especially convenient for GPU or SSE application based on projective representation. In the case of two triangles intersection computation, the vertices of triangles can be given in homogeneous coordinates and intersection of triangles can be computed directly without need to convert vertices coordinates to the Euclidean coordinates. This saves 18 division operations per one pair of triangles intersection computation, which leads to significant speed-up.

Similarly, barycentric coordinates can be computed directly in the projective space if coordinates of vertices are in homogenous coordinates and the conversion to the Euclidean space is not needed, i.e. we save division operations as well.

In the following the notation will be used:

- $\boldsymbol{X} = (X, Y, Z) \in E^3$ for coordinates or vectors in E^3
- $\boldsymbol{x} = [x, y, z: 1]^T$ or $\boldsymbol{x} = [x, y, z: w]^T$ homogeneous coordinates in the projective space
- $\boldsymbol{a} \times \boldsymbol{b}$ is the outer (cross) vector product
- $\boldsymbol{a}^T\boldsymbol{b} = \boldsymbol{a} \cdot \boldsymbol{b}$ for dot (scalar) product

II. PROBLEM FORMULATION

Intersection of two planes is a seemingly simple problem, see Fig.1.

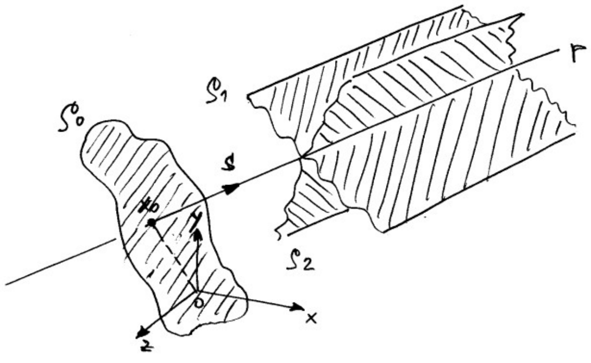

Figure 1: Intersection of two planes

Let us assume two planes ρ_1 and ρ_2, Fig.1, given by two vectors $\boldsymbol{\rho}_1 = [a_1, b_1, c_1: d_1,]^T$ and $\boldsymbol{\rho}_2 = [a_2, b_2, c_2: d_2,]^T$ as:

$$\begin{gathered}\rho_1: a_1X + b_1Y + c_1Z + d_1 = \\ \boldsymbol{n}_1^T\boldsymbol{X} + d_1 = 0\end{gathered} \tag{1}$$

and

$$\begin{gathered}\rho_2: a_2X + b_2Y + c_2Z + d_2 \\ = \boldsymbol{n}_2^T\boldsymbol{X} + d_2 = 0\end{gathered} \tag{2}$$

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

We need to determine a line $p \in E^3$ which is given as an intersection of those two planes if planes are "not parallel". However due to the numerical precision in the floating point representation, the condition must be weaken to "not close to parallel". There is the key problem, i.e. what does actually this condition mean and how it is defined from the algorithmic point of view.

It can be seen that the line p in the parametric form can be then determined as:

$$\boldsymbol{s} = \boldsymbol{n}_1 \times \boldsymbol{n}_2 = [a_3, b_3, c_3]^T \quad \boldsymbol{X}(t) = \boldsymbol{X}_0 + \boldsymbol{s}t \tag{3}$$

As the computation of the vector $\boldsymbol{s}$ is simple and relatively precise, the "problem" is how to determine the point $\boldsymbol{X}_0$ complying (1) and (2), reliably from the numerical precision point of view. Unfortunately in some applications it can be found incorrect or non-robust solutions.

III. Standard Solution

Standard formula for a line given as an intersection of two planes in the Euclidean space is given as:

$$\boldsymbol{s} = \boldsymbol{n}_1 \times \boldsymbol{n}_2 \equiv [a_3, b_3, c_3]^T$$
$$\boldsymbol{X}(t) = \boldsymbol{X}_0 + \boldsymbol{s}t$$

Then the "starting" point $\boldsymbol{X}_0$ is given as

$$X_0 = \frac{d_2 \begin{vmatrix} b_1 & c_1 \\ b_3 & c_3 \end{vmatrix} - d_1 \begin{vmatrix} b_2 & c_2 \\ b_3 & c_3 \end{vmatrix}}{DET}$$
$$Y_0 = \frac{d_2 \begin{vmatrix} a_3 & c_3 \\ a_1 & c_1 \end{vmatrix} - d_1 \begin{vmatrix} a_3 & c_3 \\ a_2 & c_2 \end{vmatrix}}{DET} \qquad (4)$$
$$Z_0 = \frac{d_2 \begin{vmatrix} a_1 & b_1 \\ a_3 & b_3 \end{vmatrix} - d_1 \begin{vmatrix} a_2 & b_2 \\ a_3 & b_3 \end{vmatrix}}{DET}$$
$$DET = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}$$

This formula is quite "horrible" one and for users is too complex to remember and they do not see from the formula comes from.

It can be seen that coordinates are computed using division operation and division by DET is required. There is a legitimate question about specification of cases when division by DET causes floating point overflow or underflow. In many cases the programmer uses a sequence:

$$eps := 10^{-10};$$
$$\textbf{if } DET < eps \textbf{ then } \text{Error}$$

but it can happen that the expression is ${}^{eps}/_{eps}$, e.g. for the x_0 -coordinate, and the expression is actually "well conditioned". The formula for $\boldsymbol{X}_0$ computation is not convenient for GPU, nor SSE application, too.

IV. Projective Space Representation

The projective extension of the Euclidean space is often used in geometry, sometimes is "hidden" in the known formulas.

Let us consider a point $\boldsymbol{X} = (X, Y) \in E^2$ and its equivalent in the projective space $[x, y\colon w]^T \in P^2$. The mutual conversion is then given as [1],[3]:

$$X = x/w \qquad Y = y/w \qquad w \neq 0 \qquad (5)$$

It can be seen that it is actually one parametric set, i.e. a line passing the origin in the x, y, w coordinate system, but origin is excluded. The extension to the E^3 case is straightforward.

Homogeneous coordinates and the projective space representation is widely used in computer graphics to represent geometric transformations, projection operation etc.

Geometric transformations with points are described generally as:

$$\boldsymbol{x}' = \boldsymbol{Q}\boldsymbol{x} \qquad (6)$$

where $\boldsymbol{Q}$ is a matrix 3×3 in the E^2 case, or 4×4 in the E^3 case.

The advantage of the projective representation is that fundamental transformations like translation, rotation, scaling, shearing and projections $E^3 \rightarrow E^2$ are represented by a cumulative transformation matrix multiplied by a vector containing point's coordinates.

However, geometric transformations with lines in E^2 or planes in E^3 are given as:

$$\boldsymbol{p}' = (\boldsymbol{Q}^{-1})^T\boldsymbol{p} \qquad \boldsymbol{\rho}' = (\boldsymbol{Q}^{-1})^T\boldsymbol{\rho} \qquad (7)$$

as normal vectors are actually bivectors.
It can be seen that the transformation matrices for lines or planes are different from the transformation matrix of points defining the given goe.

V. Principle of Duality

The principle of duality in P^2 states that any theorem remains true when we interchange the words "point" and "line", "lie on" and "pass through", "join" and "intersection", "collinear" and "concurrent" and so on. In the case of P^3 dual identities are "point" and "plane" etc. Once the theorem has been established, the dual theorem is obtained as described above [2], [5].

In other words, the principle of duality says that in all theorems in E^2 it is possible to substitute the term "point" by the term "line" and the term "line" by the term "point" and the given theorem stays valid. In the case of E^3 dual terms are "point" and "plane". This helps a lot to solve some geometrical problems.

A nice example of projective space representation and principle of duality application in the E^2 case is computation of a line p given by two given points $\boldsymbol{x}_i$, Eq.(8)

$$\boldsymbol{p} = \boldsymbol{x}_1 \times \boldsymbol{x}_2 = [x_1, y_1\colon w_1]^T \times [x_2, y_2\colon w_2]^T = [a, b\colon c]^T \qquad (8)$$

and an intersection point $\boldsymbol{x}$ of two lines p_i, Eq.(9)

$$\boldsymbol{x} = \boldsymbol{p}_1 \times \boldsymbol{p}_2 = [a_1, b_1\colon c_1]^T \times [a_1, b_1\colon c_1]^T = [x, y\colon w]^T \qquad (9)$$

It can be shown that computation of an intersection of two lines in E^2 is dual to computation of a line given by two points (it is actually a join operation). It means that there should be the same programming sequence for solving both dual cases [7], i.e.

$$\boldsymbol{x} = \boldsymbol{p}_1 \times \boldsymbol{p}_2 = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & k \\ a_1 & b_1 & d_1 \\ a_2 & b_2 & d_2 \end{vmatrix}$$
$$det \begin{bmatrix} x & y & w \\ a_1 & b_1 & d_1 \\ a_2 & b_2 & d_2 \end{bmatrix} = 0 \qquad (10)$$

and

$$\boldsymbol{p} = \boldsymbol{x}_1 \times \boldsymbol{x}_2 = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{vmatrix} \qquad det \begin{bmatrix} a & b & d \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix} = 0 \tag{11}$$

In the case of E^3 a plane given as a join of three points and dual problem, i.e. an intersection of three planes, can be computed as [8]:

$$\boldsymbol{x} = \boldsymbol{\varrho}_1 \times \boldsymbol{\varrho}_2 \times \boldsymbol{\varrho}_3 = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{vmatrix} \qquad det \begin{bmatrix} x & y & z & w \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{bmatrix} = 0 \tag{12}$$

and

$$\boldsymbol{\varrho} = \boldsymbol{x}_1 \times \boldsymbol{x}_2 \times \boldsymbol{x}_3 = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ x_1 & y_1 & z_1 & w_1 \\ x_2 & y_2 & z_2 & w_2 \\ x_3 & y_3 & z_3 & w_3 \end{vmatrix} \qquad det \begin{bmatrix} a & b & c & d \\ x_1 & y_1 & z_1 & w_1 \\ x_2 & y_2 & z_2 & w_2 \\ x_3 & y_3 & z_3 & w_3 \end{bmatrix} = 0 \tag{13}$$

There are significant advantages of this approach:

- Natural support for GPU computation leading to significant speed up.
- There is no division operation used that is needed if an intersection is computed in the Euclidean space, like $^{det_x}/_{det}$, which leads to instability in principle.
- There are no special cases which a programmer has to take care of, i.e. detection of collinearity etc.
- If points are given in homogeneous coordinates, transformation to the Euclidean coordinates is not required. It means that 6, resp. 9 division operations are not needed in the case of E^2, resp. E^3, and higher precision can be expected as well.

As a direct consequence it can be proved [9] that a solution of a linear system of equations is equivalent to an extended cross-product [9]. This is a significant result as instead of solving a linear system of equations $\boldsymbol{Ax} = \boldsymbol{b}$ we can use extended cross-product. It should be noted that replacing a solution of linear system of equations $\boldsymbol{Ax} = \boldsymbol{b}$ by the extended cross-product $\boldsymbol{x} = \boldsymbol{\xi}_1 \times \boldsymbol{\xi}_2 \times \dots \times \boldsymbol{\xi}_n$ enables also further formal manipulation using linear algebra.

As a typical example of this approach is computation of barycentric coordinates which can be used also for the case, when points $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$ are given in homogeneous coordinates.

VI. Plücker coordinates

The Plücker coordinates are often used in robotics and geometrical problems solutions, e.g. in a ray-triangle intersection detection and intersection computation [4].

Let us consider two points in the homogeneous coordinates:

$$\boldsymbol{x}_1 = [x_1, y_1, z_1 : w_1]^T \qquad \boldsymbol{x}_2 = [x_2, y_2, z_2 : w_2]^T \tag{14}$$

definining a line $p \epsilon E^3$. The Plücker coordinates l_{ij} of the anti-symmetric matrix $\boldsymbol{L}$ are defined as:

$$\begin{gathered} \boldsymbol{L} = \boldsymbol{x}_1\, \boldsymbol{x}_2^T - \boldsymbol{x}_2\, \boldsymbol{x}_1^T \\ l_{41} = w_1 x_2 - w_2 x_1 \qquad l_{23} = y_1 z_2 - y_2 z_1 \\ l_{42} = w_1 y_2 - w_2 y_1 \qquad l_{31} = z_1 x_2 - z_2 x_1 \\ l_{43} = w_1 z_2 - w_2 z_1 \qquad l_{12} = x_1 y_2 - x_2 y_1 \\ l_{ii} = 0 \qquad l_{ij} = -l_{ji} \end{gathered} \tag{15}$$

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

$$\boldsymbol{\omega} = [l_{41}, l_{42}, l_{43}]^T \qquad \boldsymbol{\nu} = [l_{23}, l_{31}, l_{12},]^T \tag{16}$$

It means that $\boldsymbol{\omega}$ represents the “directional vector”, while $\boldsymbol{\nu}$ represents the “positional vector”. It can be seen that for the Euclidean space ($w = 1$) we get:

$$\boldsymbol{X}_2 - \boldsymbol{X}_1 = \boldsymbol{\omega} \qquad \boldsymbol{X}_1 \times \boldsymbol{X}_2 = \boldsymbol{\nu} \tag{17}$$

where: $\boldsymbol{X}_i = \left({}^{x_i}/_{w_i}, {}^{y_i}/_{w_i}, {}^{z_i}/_{w_i}\right)$ are coordinates of points in the Euclidean space.

The line p given by two points in the homogeneous coordinates is then given as:

$$\boldsymbol{x}(t) = \frac{\boldsymbol{\nu} \times \boldsymbol{\omega}}{\|\boldsymbol{\omega}\|^2} + \boldsymbol{\omega} t \tag{18}$$

where $\boldsymbol{x} = (X, Y, Z)$ are coordinates of points on the line p.

Due to the principle of duality in E^3 we can exchange “point” and “plane” in the Eq.(14). As the panes are given as:

$$\begin{aligned} \boldsymbol{\rho}_1 &= [a_1, b_1, c_1 : d_1]^T = [\boldsymbol{n}_1^T : d_1]^T \\ \boldsymbol{\rho}_2 &= [a_2, b_2, c_2 : d_2]^T = [\boldsymbol{n}_2^T : d_2]^T \end{aligned} \tag{19}$$

Now, the matrix $\boldsymbol{L}$ is defined as

$$\boldsymbol{L} = \boldsymbol{\rho}_1\, \boldsymbol{\rho}_2^T - \boldsymbol{\rho}_2\, \boldsymbol{\rho}_1^T \tag{20}$$

and the rest is the same.

However, an intersection of two planes is the case also often solved in computer graphics and vision. Unfortunately in many cases available solutions are not robust or formulas used are neither simple, like above, nor convenient for GPU use. This approach also requires normalization of the $\boldsymbol{\omega}$ vector and division operations.

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

VII. INTERSECTION OF TWO PLANES

Let us consider again two planes given in implicit form, i.e. $\boldsymbol{\rho}_i^T \boldsymbol{x} = 0$, $i = 1,2$ and given points $\boldsymbol{x}_i = [x_i, y_i, z_i : w_i]^T$ in homogeneous coordinates, see Fig.1.

$$\begin{aligned} \boldsymbol{\rho}_1 &= [a_1, b_1, c_1 : d_1]^T & \boldsymbol{\rho}_2 &= [a_2, b_2, c_2 : d_2]^T \\ &= [\boldsymbol{n}_1^T : d_1]^T & &= [\boldsymbol{n}_2^T : d_2]^T \end{aligned} \tag{21}$$

The directional vector $\boldsymbol{s}$ of the line p given as an intersection of those two planes is again given as:

$$\begin{gathered} \boldsymbol{s} = \boldsymbol{n}_1 \times \boldsymbol{n}_2 = [a_3, b_3, c_3]^T \\ \boldsymbol{X}(t) = \boldsymbol{X}_0 + \boldsymbol{s}t \end{gathered} \tag{22}$$

An angle φ between two planes ρ_1 and ρ_2 is given as:

$$\cos\varphi = \frac{\boldsymbol{n}_1^T \boldsymbol{n}_2}{\|\boldsymbol{n}_1\| \|\boldsymbol{n}_2\|} \tag{23}$$

If the "standard" formula is used, the $\cos\varphi$ value is to be used for detection of close to singular cases or:

$$\cos^2\varphi = \frac{(\boldsymbol{n}_1^T \boldsymbol{n}_2)^2}{(\boldsymbol{n}_1^T \boldsymbol{n}_1)(\boldsymbol{n}_2^T \boldsymbol{n}_2)} \tag{24}$$

as this eliminates $sqrt$ function computation in the vector normalization as well.

Now, let us assume a plane $\boldsymbol{\rho}_0$ passing the origin of the coordinate system having as a normal vector the vector $\boldsymbol{s}$, see Fig.1. Therefore the point $\boldsymbol{x}_0$ is an intersection of three planes $\boldsymbol{\rho}_0$, $\boldsymbol{\rho}_1$ and $\boldsymbol{\rho}_2$ can be computed as a direct application of the principle duality in the P^3 case as a straightforward application of the Eq.(8) and Eq.(9):

$$\boldsymbol{x}_0 = \boldsymbol{\rho}_0 \times \boldsymbol{\rho}_1 \times \boldsymbol{\rho}_2 = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ a_0 & b_0 & c_0 & 0 \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \end{vmatrix} \tag{25}$$

From the Eq.(25) we can write for the $\boldsymbol{x}_0$ point using homogeneous coordinates $\boldsymbol{x}_0 = [x_0, y_0, z_0 : w_0]^T$:

$$\begin{aligned} x_0 &= +d_2 \begin{vmatrix} b_0 & c_0 \\ b_1 & c_1 \end{vmatrix} - d_1 \begin{vmatrix} b_0 & c_0 \\ b_2 & c_2 \end{vmatrix} \\ y_0 &= -\left\{ +d_2 \begin{vmatrix} a_0 & c_0 \\ a_1 & c_1 \end{vmatrix} - d_1 \begin{vmatrix} a_0 & c_0 \\ a_2 & c_2 \end{vmatrix} \right\} \\ z_0 &= +d_2 \begin{vmatrix} a_0 & b_0 \\ a_1 & b_1 \end{vmatrix} - d_1 \begin{vmatrix} a_0 & b_0 \\ a_2 & b_2 \end{vmatrix} \\ w_0 &= -\begin{vmatrix} a_0 & b_0 & c_0 \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{vmatrix} \end{aligned} \tag{26}$$

The point $\boldsymbol{x}_0$ is the closest point on the line p to the origin of the coordinate system. As the computation is made in the projective space no division operation is needed.

The actual decision on "singularity" of computation is postponed. As a determinant is multilinear, it is clear that normalization of plane's equations to $\|\boldsymbol{n}_i\| = 1$ not needed. and it is not a way how to increase stability of computations.

Even more we are getting a formula, which can be used for symbolic manipulation and computation as well.

Derivation of this formula is simple, fast and it is well suited for the GPU or SSE application as the cross product is an GPU instruction, see **Appedix A**

VIII. BARYCENTRIC COORDINATES

Barycentric coordinates are often used in many applications, not only in geometry. Barycentric coordinates computation leads to a solution of a system of linear equations. However it can be shown, that a solution of a linear system equations is equivalent to a cross product [7], [8], [10]. Therefore it is possible to compute barycentric coordinates using cross product for GPU oriented applications.

Let us consider the E^1 case, i.e. interpolation between two points, and vector:

$$\boldsymbol{x} = [x_1, x_2 : x]^T \qquad \boldsymbol{w} = [w_1, w_2 : w]^T \tag{27}$$

Then the projective barycentric coordinates are given as:

$$\boldsymbol{\xi} = \boldsymbol{x} \times \boldsymbol{w} = [\xi_1, \xi_2 : \xi_w]^T \tag{28}$$

The Euclidean barycentric coordinates are given as:

$$\lambda_1 = -\frac{\xi_1}{\xi_w} \qquad \lambda_2 = -\frac{\xi_2}{\xi_w} \tag{29}$$

Let us consider the E^2 case, i.e. interpolation between three points of the given triangle, and vectors:

$$\begin{gathered} \boldsymbol{x} = [x_1, x_2, x_3 : x]^T \\ \boldsymbol{y} = [y_1, y_2, y_3 : y]^T \\ \boldsymbol{w} = [w_1, w_2, w_3 : w]^T \end{gathered} \tag{30}$$

The projective barycentric coordinates are given as:

$$\boldsymbol{\xi} = \boldsymbol{x} \times \boldsymbol{y} \times \boldsymbol{w} = [\xi_1, \xi_2, \xi_3 : \xi_w]^T \tag{31}$$

The Euclidean barycentric coordinates are given as:

$$\lambda_1 = -\frac{\xi_1}{\xi_w} \qquad \lambda_2 = -\frac{\xi_2}{\xi_w} \qquad \lambda_3 = -\frac{\xi_3}{\xi_w} \tag{32}$$

Let us consider the E^3 case, i.e. interpolation between four points of the given tetrahedron, and vectors:

$$\begin{aligned} \boldsymbol{x} &= [x_1, x_2, x_3, x_4 : x]^T & \boldsymbol{y} &= [y_1, y_2, y_3, y_4 : y]^T \\ \boldsymbol{z} &= [z_1, z_2, z_3, z_4 : z]^T & \boldsymbol{w} &= [w_1, w_2, w_3, w_4 : w]^T \end{aligned} \tag{33}$$

Then projective barycentric coordinates are given as:

$$\boldsymbol{\xi} = \boldsymbol{x} \times \boldsymbol{y} \times \boldsymbol{z} \times \boldsymbol{w} = [\xi_1, \xi_2, \xi_3, \xi_4 : \xi_w]^T \tag{34}$$

The Euclidean barycentric coordinates are given as:

$$\begin{aligned} \lambda_1 &= -\frac{\xi_1}{\xi_w} & \lambda_2 &= -\frac{\xi_2}{\xi_w} \\ \lambda_3 &= -\frac{\xi_3}{\xi_w} & \lambda_4 &= -\frac{\xi_4}{\xi_w} \end{aligned} \tag{35}$$

How simple and elegant solution!

It can be seen that the presented computation of barycentric coordinates is simple, convenient for GPU or SSE application. Even more, as we have assumed from the very beginning, there is no need to convert coordinates of points from the homogeneous coordinates to the Euclidean coordinates. As a direct consequence of that is that we save lot of division operations and also increase robustness of the computation.

IX. The Closest point to an intersection of planes

Finding the closest point on the intersection of two planes to the given point $\boldsymbol{\xi}$ is not an easy problem. The usual solution is based on application of the Lagrange multipliers [6] leading to a solution of a system of linear equations with a matrix 5×5. This solution is computationally costly and not convenient for the GPU applications. Using projective representation we can easily solve the problem as follows.

Let us assume given two planes:

$$\boldsymbol{\rho}_1 = [a_1, b_1, c_1 : d_1]^T \qquad \boldsymbol{\rho}_2 = [a_2, b_2, c_2 : d_2]^T \tag{36}$$

with normal vectors:

$$\boldsymbol{n}_1 = [a_1, b_1, c_1]^T \qquad \boldsymbol{n}_2 = [a_2, b_2, c_2]^T \tag{37}$$

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

$$\boldsymbol{s} = \boldsymbol{n}_1 \times \boldsymbol{n}_2 \tag{38}$$

and the "starting" point $\boldsymbol{x}_0$ is determined as recently stated

$$\boldsymbol{x}_0 = \boldsymbol{\rho}_1 \times \boldsymbol{\rho}_2 \times \boldsymbol{\rho}_0 \tag{39}$$

and the plane $\boldsymbol{\rho}_0$ is passing the origin with a normal vector $\boldsymbol{s}$, i.e. the plane is defined by $\boldsymbol{\rho}_0 = [a_0, b_0, c_0 : 0]^T$.

The solution is now quite simple applying the following steps:

- Translate planes $\boldsymbol{\rho}_1$ and $\boldsymbol{\rho}_2$ so the given point $\boldsymbol{\xi}$ is in the origin, transformation matrix is $\boldsymbol{T}$
- Compute intersection of two planes, i.e. determine the directional vector $\boldsymbol{s}$ of the line and the point $\boldsymbol{x}_0$
- Translate the computed point $\boldsymbol{x}_0$ using $\boldsymbol{T}^{-1}$

The point $\boldsymbol{x}_0$ is the closest point of the line to the given point $\boldsymbol{\xi}$. Even more, the parameter value t to this point $\boldsymbol{x}_0$ is $t = 0$.

Again – an elegant solution, simple formula supporting matrix-vector architectures like GPU and parallel processing.

X. Conclusion

In this paper we have presented a new robust approach for a solution of selected geometrical problems using projective representation. This approach offers much simpler formulation, algorithms based on matrix-vector multiplications. Resulting formulas are convenient for GPU and SSE applications with achievable significant speedup as well.

XI. Acknowledgment

The author thanks to anonymous reviewers for their critical comments and hints that improved this paper significantly. Thanks also belong to students and colleagues at the University of West Bohemia, Plzen for their suggestions.

The project was supported by the Ministry of Education of the Czech Republic, projects No.LG13047 and LH12181.

XII. References

[1] Bloomenthal,J., Rokne,J.: Homogeneous Coordinates, The Visual Computer, Vol.11,No.1, pp.15-26, 1994.

[2] Coxeter,H.S.M.: Introduction to Geometry, John Wiley, 1969.

[3] Hartley,R, Zisserman,A.: MultiView Geometry in Computer Vision, Cambridge Univ. Press, 2000.

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

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

[6] Krumm,J.: Intersection of Two Planes, Microsoft Research, Research note, 2000

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

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

[9] Skala,V.: Barycentric Coordinates Computation in Homogeneous Coordinates, Computers & Graphics, Elsevier, ISSN 0097-8493, Vol. 32, No.1, pp.120-127, 2008

[10] Skala,V.: Projective Geometry, Duality and Precision of Computation in Computer Graphics, Visualization and Games, Tutorial Eurographics 2013, Girona, 2013

[11] Skala,V.: Projective Geometry and Duality for Graphics, Games and Visualization - Course SIGGRAPH Asia 2012, Singapore, ISBN 978-1-4503-1757-3, 2012

Appendix

The cross product in 4D is defined as

$$\boldsymbol{x}_1 \times \boldsymbol{x}_2 \times \boldsymbol{x}_3 = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} & \boldsymbol{l} \\ x_1 & y_1 & z_1 & w_1 \\ x_2 & y_2 & z_2 & w_2 \\ x_3 & y_3 & z_3 & w_3 \end{vmatrix}$$

and can be implemented in Cg/HLSL on a GPU as follows:

```
float4 cross_4D(float4 x1, float4 x2, float4 x3)
{           float4 a;
            a.x = dot(x1.yzw, cross(x2.yzw, x3.yzw));
            a.y = - dot(x1.xzw, cross(x2.xzw, x3.xzw));
            a.z = dot(x1.xyw, cross(x2.xyw, x3.xyw));
            a.w = - dot(x1.xyz, cross(x2.xyz, x3.xyz));
            return a;
}
```

or more compactly as

```
float4 cross_4D(float4 x1, float4 x2, float4 x3)
{
return (
             dot(x1.yzw, cross(x2.yzw, x3.yzw)),
           - dot(x1.xzw, cross(x2.xzw, x3.xzw)),
             dot(x1.xyw, cross(x2.xyw, x3.xyw)),
           - dot(x1.xyz, cross(x2.xyz, x3.xyz)) );
}
```