Game Programming GEMS 6(Ed.Dickheiser,M.), pp.137-147,ISBN 1-58450-450-1, Charles River Media, 2006

GPU Computation in Projective Space using Homogeneous Coordinates

Vaclav Skala – University of West Bohemia, Czech Republic
skala@kiv.zcu.cz

There are many examples in computer graphics where linear algebra is used to solve given problems, e.g. computing a line from two points, an intersection of two lines, or line clipping. In some cases, difficulties may occur if the given mathematical formula is applied directly. In particular, tests similar to $a = b$ are highly suspicious from the computational stability point of view. Even a very simple collinearity test between two lines is very unreliable due to a limited precision of computation.

There is another factor to be considered. In computer graphics applications, points are mostly represented in homogeneous coordinates, i.e., in the real projective space [Ferguson01], [Hill01], [Shirley02]. Users and programmers mostly convert points in homogeneous coordinates into Euclidean coordinates using a division operation. This division can take a long time relative to other floating point operations and might cause severe numerical instability or a fatal division by zero exception.

In this article, we will show how common intersection computations can be performed directly using homogeneous coordinates. This approach results in higher numerical stability and in some cases a significant speed up. The presented approach is especially convenient if vector and matrix operations are supported on the main processor or for applications on a graphics processing unit, or GPU.

Mathematical background

Let us begin by reviewing some mathematical facts, which are necessary for understanding the remainder of the article.

Homogeneous coordinates

Suppose we have a point in E^2, or 2-dimensional space, represented in Euclidean coordinates as $\mathbf{X} = (X, Y)$. It is well known that we can represent the same point in 3-dimensional *homogeneous coordinates* as $\mathbf{x} = [x, y, w]^T$, where

$$x = wX \qquad y = wY, \qquad w \neq 0 \tag{1}$$

A common value for *w* is 1, so the point **X** is usually represented as $\mathbf{x} = [X, Y, 1]^T$. Note that since there are an infinite number of possibilities for *w*, **X** can be represented by an

infinite number of points in homogeneous coordinates. Also, if $w = 0$, then the entity represented is not a point but a vector, i.e. a "point at infinity."

Given a point $\mathbf{x} = [wX, wY, w]^T$ in homogeneous space, the corresponding Euclidean coordinates (X, Y) are computed as

$$X = x / w \quad Y = y / w, \quad w \neq 0 \tag{2}$$

The homogeneous representation for points in E^3, or 3-dimensional space, behaves similarly.

Operations with determinants

The determinant of a matrix **A** can be defined recursively as

$$\det(\mathbf{A}) = |\mathbf{A}| = \sum_{j=1}^{n} a_{ij} (-1)^{(i+j)} \det(\tilde{\mathbf{A}}_{ij}) \tag{3}$$

where $\tilde{\mathbf{A}}_{ij}$ is the submatrix formed by removing the *i*th row and *j*th column from **A**. For a 3×3 matrix this can be computed as

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a(ei - fh) - b(di - fg) + c(dh - eg) \tag{4}$$

We will be using two additional properties of determinants. The first is that scaling a single row by a factor *q* scales the determinant by the same factor *q*, for $q \neq 0$. This can be represented as

$$\begin{vmatrix} a_{11} & & \cdots & a_{1n} \\ \vdots & & & \vdots \\ qa_{k1} & qa_{k2} & \cdots & qa_{kn} \\ \vdots & & & \vdots \\ a_{n1} & & \cdots & a_{nn} \end{vmatrix} = q \cdot \begin{vmatrix} a_{11} & & \cdots & a_{1n} \\ \vdots & & & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kn} \\ \vdots & & & \vdots \\ a_{n1} & & \cdots & a_{nn} \end{vmatrix} \tag{5}$$

The second is that if the determinant is 0, then the rows or columns of the determinant matrix are linearly dependent; that is, you can create one row or column by scaling and adding the remaining rows or columns. Such a matrix has no inverse. Similarly, if the matrix is representing a system of linear equations, such a system has no solution.

Operations with vectors

A dot product of two vectors $\mathbf{a} = [a_1, a_2, a_3]^T$ and $\mathbf{b} = [b_1, b_2, b_3]^T$ is defined as:

$$\mathbf{a}^T\mathbf{b} = \mathbf{b}^T\mathbf{a} = \mathbf{a} \bullet \mathbf{b} = \|\mathbf{a}\|\|\mathbf{b}\|\cos\varphi \tag{6}$$

where φ is the angle between vectors **a** and **b**.

A *cross product* of two vectors $\mathbf{a} = [a_1, a_2, a_3]^T$ and $\mathbf{b} = [b_1, b_2, b_3]^T$ is defined as

$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} \tag{7}$$

where

$$\mathbf{i} = [1,0,0]^T,\ \mathbf{j} = [0,1,0]^T,\ \mathbf{k} = [0,0,1]^T$$

By Equation 3 above, this result is equal to

$$\mathbf{a} \times \mathbf{b} = (a_2b_3 - a_3b_2)\mathbf{i} - (a_1b_3 - a_3b_1)\mathbf{j} + (a_1b_2 - a_2b_1)\mathbf{k} \tag{8}$$

This produces a vector perpendicular to **a** and **b**. Note that this is only defined for vectors in 3-dimensional space and that $\mathbf{a} \times \mathbf{b} = -\mathbf{b} \times \mathbf{a}$.

We can introduce a cross product for the 4-dimensional case as another determinant:

$$\mathbf{a} \times \mathbf{b} \times \mathbf{c} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{l} \\ a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \end{vmatrix} \tag{9}$$

where

$$\mathbf{i} = [1,0,0,0]^T,\ \mathbf{j} = [0,1,0,0]^T,\ \mathbf{k} = [0,0,1,0]^T,\ \mathbf{l} = [0,0,0,1]^T$$

Similar to the 3-dimensional case, this produces a vector perpendicular to **a**, **b**, and **c**. This definition will be used later on.

Principle of duality

The principle of duality states that any theorem remains valid when we interchange dual words in the theorem. For example, in the E^2 case the words “point” and “line” are dual, together with “pass through” and “lie on,” and “intersect” and “join”. Similarly, in the E^3 case a point is dual to a plane.

What does this mean in computer graphics? It is well known that a line p in E^2 can be defined as

$$aX + bY + c = 0 \tag{10}$$

Based on this equation, we can state that a vector $\mathbf{a} = [a, b, c]^T$ in E^3 defines the line p. Furthermore, Equation 10 can be multiplied by any $w \neq 0$ and the line is geometrically the same. So the line p is actually dual to a homogeneous point (as we'd expect) and both are represented in the dual space by a vector $\mathbf{a}$. Understanding this is very important, as the principle of duality will help us to derive some formulas later.

Computation in homogeneous coordinates

Now we will show how to use homogeneous coordinates to solve some simple linear systems. We will begin by concentrating on the E^2 case and later on we will extend this to E^3.

E^2 case

Let us consider two lines p_1 and p_2 in E^2:

$$p_1 : a_1X + b_1Y + c_1 = 0 \qquad \text{and} \qquad p_2 : a_2X + b_2Y + c_2 = 0 \tag{11}$$

The intersection point of those two lines is defined as a solution of a linear system of non-homogeneous equations

$$\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} -c_1 \\ -c_2 \end{bmatrix} \tag{12}$$

It is well known that the solution is given as

$$X = \frac{D_x}{D} \qquad Y = \frac{D_y}{D} \tag{13}$$

where: $D_x = \det \begin{vmatrix} -c_1 & b_1 \\ -c_2 & b_2 \end{vmatrix}$ $D_y = -\det \begin{vmatrix} a_1 & -c_1 \\ a_2 & -c_2 \end{vmatrix}$ $D = \det \begin{vmatrix} -a_1 & b_1 \\ -a_2 & b_2 \end{vmatrix}$

However, since we are dividing by D, the solution becomes numerically unstable as $D \to 0$. Instead, note that Equation 12 can be rewritten using homogeneous coordinates:

$$\begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{14}$$

So an alternative is to express the solution of Equation 11 using homogeneous coordinates, as well:

$$\mathbf{x} = [x, y, w]^T = [D_x, D_y, D]^T \tag{15}$$

and using Equation 2 we can eventually compute *X* and *Y* as

$$X = \frac{x}{w} = \frac{D_x}{D} \qquad Y = \frac{y}{w} = \frac{D_y}{D}, \qquad D \neq 0 \tag{16}$$

which matches our solution in Equation 13. If $D = 0$ then the given lines are parallel or identical.

Examining the equations for D_x, D_y and *D*, it can be seen that the homogeneous intersection point $\mathbf{x}$ can be computed as a cross product of two vectors $\mathbf{a}_1$ and $\mathbf{a}_2$, i.e.

$$\mathbf{x} = \mathbf{a}_1 \times \mathbf{a}_2 = \det \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{vmatrix} \tag{17}$$

where: $\mathbf{i} = [1,0,0]^T$, $\mathbf{j} = [0,1,0]^T$, $\mathbf{k} = [0,0,1]^T$

Now let's look at the dual problem. As we covered before, the principle of duality in E^2 states that a line is dual to a point and vice versa. So for our situation, the dual problem is determining a line given by two points. One way to solve this is via the following system of linear equations:

$$\begin{bmatrix} X_1 & Y_1 & 1 \\ X_2 & Y_2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{18}$$

where the two points are given as $\mathbf{x}_i = [X_i, Y_i, 1]^T$, and the line *p* represented by the vector $\mathbf{a} = [a, b, c]^T$. Alternatively, we can use:

$$\mathbf{a} = \mathbf{x}_1 \times \mathbf{x}_2 = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ X_1 & Y_1 & 1 \\ X_2 & Y_2 & 1 \end{vmatrix} \tag{19}$$

So, as with the intersection of two lines, this means that the line given by two points can be easily determined by the cross product.

However, suppose the points are given in homogeneous coordinates with non-unit *w* values, i.e. $\mathbf{x}_i = [x_i, y_i, w_i]^T$. Normally a division operation would be performed to convert the homogeneous coordinates to Euclidean coordinates, which we would then substitute into Equation 18. Instead, note that the effect of using general homogeneous coordinates is to multiply the 1st row of the determinant in Equation 19 by $w_1 \neq 0$ and multiply the 2nd row by $w_2 \neq 0$. Due to the identity defined by Equation 5 , we can then write:

$$\mathbf{x}_1 \times \mathbf{x}_2 = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{vmatrix} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ w_1 X_1 & w_1 Y_1 & w_1 \\ w_2 X_2 & w_2 Y_2 & w_2 \end{vmatrix} = w_1 w_2 \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ X_1 & Y_1 & 1 \\ X_2 & Y_2 & 1 \end{vmatrix} = w_1 w_2 \mathbf{a} \tag{20}$$

The end result is that we have scaled our original vector **a** by $w_1 w_2$. However, as we stated above, if $w_1 w_2 \neq 0$ the line represented by $w_1 w_2 \mathbf{a}$ is geometrically the same as the line represented by **a**. So by using Equation 20 we can determine the line *p* directly using the general homogeneous coordinates of the given points and without use of the division operation.

E^3 case

The extension to the E^3 case is very simple. In this situation a point is dual to a plane and vice versa. So we can write that:

- The intersection point **x** of three mutually non-coplanar planes can be computed as

$$\mathbf{x} = \mathbf{a}_1 \times \mathbf{a}_2 \times \mathbf{a}_3 = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{l} \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{vmatrix} \tag{21}$$

where: $\mathbf{a}_i = [a_i, b_i, c_i, d_i]^T$ represents a plane $\rho_i = a_i X + b_i Y + c_i Z + d_i \quad i = 1,...,3$, $\mathbf{x} = [x, y, z, w]^T$ is the intersection point in homogeneous coordinates, and $X = x/w$, $Y = y/w$, and $Z = z/w$ are the Euclidean coordinates.

As with the E^2 situation, prior to the homogeneous divide we can evaluate the coordinate w of the vector $\mathbf{x}$ and analyze singular or close to singular cases, which will represent coplanar or near-coplanar planes.

- The plane determined by three distinct (or non-coincident) points can be determined by

$$\mathbf{a} = \mathbf{x}_1 \times \mathbf{x}_2 \times \mathbf{x}_3 = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{l} \\ x_1 & y_1 & z_1 & w_1 \\ x_2 & y_2 & z_2 & w_2 \\ x_3 & y_3 & z_3 & w_3 \end{vmatrix} \tag{22}$$

where: $\mathbf{x}_i = [x_i, y_i, z_i, w_i]^T$ represents the given points $\mathbf{x}_i$, $i = 1,...,3$ and $\mathbf{a} = [a, b, c, d]^T$ represents the plane ρ determined by these three points, i.e., $\rho = aX + bY + cZ + d$

We can evaluate the homogeneous coordinate w of the vector $\mathbf{a}$ and analyze singular or close to singular cases, if $\mathbf{a} = [0,0,0,0]^T$ then the given points are not mutually distinct.

All four cases taken together mean that we have a very robust and simple method for determination of:

- a line, given by two points in the homogeneous coordinates in E^2
- the intersection point of two lines in E^2
- a plane, given by three points in the homogeneous coordinates in E^3
- the intersection point of three planes in E^3

The presented approach is especially very efficient if vector/matrix operations are supported in main hardware or if a GPU can be used. An implementation of a 4D cross product in Cg/HLSL is presented in Appendix A.

Line Intersection

There are many algorithms that are devoted to determining the intersection of a line or ray with an object. In some applications the line is in a parametric form and the object is an implicitly described primitive (e.g. a line or plane). A typical example of such a

situation is the Cyrus-Beck (CB) algorithm [Cyrus78] for line clipping by a convex polygon in E^2 or by a convex polyhedron in E^3. The E^2 case is simple, but the E^3 case is a little bit tricky. One possibility is to define the line in E^3 using Plücker coordinates, see [Blinn77], but the approach below is simpler.

Line–plane intersection

As the E^3 case is more complex we will cover it in detail. Modification of the equations for the E^2 case is not very difficult and is left to the reader.

We define a line p in E^3 in the parametric form as $\mathbf{X}(t) = \mathbf{X}_A + (\mathbf{X}_B - \mathbf{X}_A)t$ where $\mathbf{X}_A = [X_A, Y_A, Z_A]^T$, $\mathbf{X}_B = [X_B, Y_B, Z_B]^T$ are given points in Euclidian coordinates. As before, we define a plane ρ as $aX + bY + cZ + d = 0$, where a vector $\mathbf{a} = [a, b, c, d]^T$ is a representation of the given plane. We can represent this in a slightly different form as $\boldsymbol{\alpha}^T\mathbf{X} + d = 0$, where $\boldsymbol{\alpha} = [a, b, c]^T$ is a normal vector of the given plane.

It is well known that the parameter of intersection of such a line and plane can be calculated as:

$$t = -\frac{\boldsymbol{\alpha}^T\mathbf{X}_A + d}{\boldsymbol{\alpha}^T(\mathbf{X}_B - \mathbf{X}_A)} \tag{23}$$

As we did with points, we can use homogeneous coordinates to represent t as $[\tau, \tau_W]$, where

$$\begin{aligned} \tau &= -(\boldsymbol{\alpha}^T\mathbf{X}_A + d) \\ \tau_w &= \boldsymbol{\alpha}^T(\mathbf{X}_B - \mathbf{X}_A) \end{aligned} \tag{24}$$

By doing this, no division is needed immediately and can be postponed in the modified algorithm.

This representation leads directly to a question: What happen if the points defining the line are given in the homogeneous coordinates in general? Usually programmers convert the given points to Euclidean coordinates using the division operation. However, if we keep them in the homogeneous form, we can rewrite Equation 23 as

$$t = -\frac{\boldsymbol{\alpha}^T\mathbf{X}_A + d}{\boldsymbol{\alpha}^T(\mathbf{X}_B - \mathbf{X}_A)} = -\frac{\boldsymbol{\alpha}^T\frac{\boldsymbol{\xi}_A}{w_A} + d}{\boldsymbol{\alpha}^T\left(\frac{\boldsymbol{\xi}_B}{w_B} - \frac{\boldsymbol{\xi}_A}{w_A}\right)} \tag{25}$$

where $\mathbf{X}_A$, $\mathbf{X}_B$, $\boldsymbol{\alpha}^T$, and d are as before, and $\mathbf{x}_A = \left[x_A, y_A, z_A, w_A\right]^T$ and $\mathbf{x}_B = \left[x_B, y_B, z_B, w_B\right]^T$ are the given points in homogeneous coordinates, and $\xi_A = \left[x_A, y_A, z_A\right]^T$ and $\xi_B = \left[x_B, y_B, z_B\right]^T$.

Multiplying both sides of Equation 25 by $w_A \neq 0$ and $w_B \neq 0$ we get

$$t = -\frac{w_B\left(\boldsymbol{\alpha}^T\xi_A + w_A d\right)}{\boldsymbol{\alpha}^T\left(w_A\xi_B - w_B\xi_A\right)} \tag{26}$$

So for the case where our line points are in homogeneous coordinates, we can rewrite Equation 24 as

$$\begin{aligned} \tau &= -w_B\left(\boldsymbol{\alpha}^T\xi_A + w_A d\right) = -w_B\mathbf{a}^T\mathbf{x}_A \\ \tau_w &= \boldsymbol{\alpha}^T\left(w_A\xi_B - w_B\xi_A\right) \end{aligned} \tag{27}$$

If the proposed representation in homogeneous coordinates is used and basic arithmetic operations including comparison operations are properly optimized we obtain more robust algorithms. A speed up on the CPU can be expected, as well, due to the fact that the division operation is not needed.

Recently, the proposed approach was applied to the CB algorithm. The modification of the CB algorithm in E^2 is faster for $N \geq 6$, where N is the number of edges of the given convex polygon. For $N \geq 500$ it is about 20% faster than the optimized original CB algorithm. This means that for E^3 the performance will grow as we can expect more polyhedron facets to be examined.

Conclusion

We have presented a new approach using homogeneous coordinates for some frequently used computations. There is a direct impact to algorithms that are convenient for GPU implementation. The main motivation of this work is to increase the robustness of algorithms, postpone the division operation to the very last moment, and enable direct computation in the homogeneous coordinates, if applicable. The presented approach has been already applied to line and line segment clipping algorithms and led to a simpler and faster solution with increased robustness [Skala04], [Skala05].

The main advantages of the presented approach are:

- graphical primitives are naturally represented in homogeneous coordinates
- elimination of division operations are possible if the result can remain in homogeneous coordinates

- significant speed up if vector/matrix operations are supported in hardware
- short and compact code for algorithms
- simple implementation on current GPUs, see Appendix A and Appendix B

There is a hope that the presented approach can lead to a new design of algorithms and computational models as well.

Acknowledgments

The author would like to express his thanks to students and colleagues at the University of West Bohemia for recommendations, constructive discussions, and suggestions that helped finish this work, especially to Ivo Hanak for hints in evaluation of hardware futures, Martin Janda for experimental verification within the Cyrus Beck algorithm, and Libor Vasa and Petr Lobaz for critical comments.

This work was supported by the Ministry of Education of the Czech Republic - project MSM 235200005, Microsoft Research Ltd.(U.K.) Project No.2004-360, ATI Technologies Inc.

References

[Blinn77] Blinn, J.F., "Homogeneous Formulation for Lines in 3 Space," Computer Graphics, Vol.11, no.2. (SIGGRAPH 77): pp.237-241.

[Cyrus78] Cyrus, M. and J. Beck, "Generalized Two- and Three Dimensional Clipping," *Computers & Graphics*, Vol. 2, No.1, pp.23-28, 1978.

[Ferguson01] Ferguson, Stuart,R., *Practical Algorithms for 3D Computer Graphics*, A.K. Peters, 2001.

[Hill01] Hill, Francis S., *Computer Graphics using OpenGL*, Prentice Hall, 2001.

[Shirley02] Shirley, Petr, *Fundamentals of Computer Graphics*, A.K. Peters, 2002.

[Skala04] Skala, Vaclav, "A New Line Clipping Algorithm with Hardware Acceleration," *cgi*, (Computer Graphics International 2004): pp. 270-273.

[Skala05] Skala, Vaclav, "A new approach to line and line segment clipping in homogeneous coordinates," *The Visual Computer*, Springer Verlag, accepted for publication, 2005.

Appendix A

The cross product in 4D defined as

$$\mathbf{x}_1 \times \mathbf{x}_2 \times \mathbf{x}_3 = \det \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{l} \\ x_1 & y_1 & z_1 & w_1 \\ x_2 & y_2 & z_2 & w_2 \\ x_3 & y_3 & z_3 & w_3 \end{vmatrix} \tag{A1}$$

can be implemented in Cg/HLSL on 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));
     // or a.y=dot(x1.xzw, cross(x3.xzw, x2.xzw));
     a.z=dot(x1.xyw, cross(x2.xyw, x3.xyw));
     a.w=-dot(x1.xyz, cross(x2.xyz, x3.xyz));
     // or a.w=dot(x1.xyz, cross(x3.xyz, x2.xyz));

     return a;
}
```

or more compactly

```
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)) );
}
```

The code is simple and uses vector operations available on current GPU. The cross-product in E^3 is supported directly by Cg/HLSL.

Appendix B

The line/plane intersection parameter defined as

$$t = -\frac{w_B\left(\boldsymbol{\alpha}^T \boldsymbol{\xi}_A + w_A d\right)}{\boldsymbol{\alpha}^T\left(w_A \boldsymbol{\xi}_B - w_B \boldsymbol{\xi}_A\right)} \tag{B1}$$

can be represented in a homogeneous form as

$$\tau = -w_B\left(\boldsymbol{\alpha}^T\boldsymbol{\xi}_A + w_A d\right) = -w_B\mathbf{a}^T\mathbf{x}_A \qquad \tau_w = \boldsymbol{\alpha}^T\left(w_A\boldsymbol{\xi}_B - w_B\boldsymbol{\xi}_A\right) \tag{B2}$$

Equation B2 can be easily implemented on GPU using Cg/HLSL as:

```
float4 intersection_3D(float4 a, float4 xa, float4 xb)
{
     float4 t;
     t.x=-xb.w*(dot(a.xyz,xa.xyz)+xa.w*a.w);
     t.w=dot(a.xyz,xa.w*xb.xyz-xb.w*xa.xyz);
     return t;
}
```

It can be seen that the presented approach significantly benefits from vector/vector operations.