Eurographics 2013

Projective Geometry, Duality and Precision of Computation in Computer Graphics, Visualization and Games

Tutorial

Vaclav Skala
University of West Bohemia, Plzen, Czech Republic
VSB-Technical University, Ostrava, Czech Republic
http://www.VaclavSkala.eu
Plzen is an old city [first records of Plzen castle 976] city of culture, industry, and brewery.

City, where today’s beer fermentation process was invented that is why today’s beers are called Pilsner - world wide
Ostrava is

- an industrial city of coal mining & iron melting
- 3\textsuperscript{rd} largest city

![Industrial city](image1.jpg)

![Tunnel](image2.jpg)

![Map of Ostrava](image3.jpg)
University of West Bohemia 17530 students + 987 PhD students

Computer Science and Engineering  Mathematics (+ Geomatics)  
Physics  Cybernetics  Mechanics (Computational)

• Over 50% of income from research and application projects  
• New research center EU project - investment 64 mil. EUR  
• 2nd in the ranking of Czech technical / informatics faculties 2009, 2012
Eurographics 2013

“Real science” in the XXI century

Courtesy of Czech Film, Barrandov
An overview

- Precision and robustness
- Euclidean space and projective extension
- Principle of duality and its applications
- Geometric computation in the projective space
- Intersection of two planes in E3 with additional constrains
- Barycentric coordinates and intersections
- Interpolation and intersection algorithms
- Implementation aspects and GPU
- Conclusion and summary
Numerical systems

- Binary system is used nearly exclusively
- Octal & hexadecimal representation is used
- If we would be direct descendants of tetrapods – we would have a great advantage – “simple counting in hexadecimal system”

<table>
<thead>
<tr>
<th>Name</th>
<th>Base</th>
<th>Digits</th>
<th>E min</th>
<th>E max</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BINARY</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B 16</td>
<td>Half</td>
<td>2</td>
<td>10+1</td>
<td>−14</td>
</tr>
<tr>
<td>B 32</td>
<td>Single</td>
<td>2</td>
<td>23+1</td>
<td>−126</td>
</tr>
<tr>
<td>B 64</td>
<td>Double</td>
<td>2</td>
<td>52+1</td>
<td>−1022</td>
</tr>
<tr>
<td>B 128</td>
<td>Quad</td>
<td>2</td>
<td>112+1</td>
<td>−16382</td>
</tr>
<tr>
<td><strong>DECIMAL</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D 32</td>
<td></td>
<td>10</td>
<td>7</td>
<td>−95</td>
</tr>
<tr>
<td>D 64</td>
<td></td>
<td>10</td>
<td>16</td>
<td>−383</td>
</tr>
<tr>
<td>D 128</td>
<td></td>
<td>10</td>
<td>34</td>
<td>−6143</td>
</tr>
</tbody>
</table>

IEEE 758-2008 standard
Mathematically perfect algorithms fail due to instability

Main issues

- stability, robustness of algorithms
- acceptable speed
- linear speedup – results depends on HW, CPU .... parameters!

Numerical stability

- limited precision of float / double
- tests $A \neq B$ with floats

  if $A = B$ then ..... else ..... ;  if $A = 0$ then ..... else .... should be forbidden in programming languages

- division operation should be removed or postponed to the last moment if possible - “blue screens”, system resets
Typical examples of instability

- intersection of 2 lines in E3
- point lies on a line in E2 or a plane in E3
  \[ Ax + By + C = 0 \quad \text{or} \quad Ax + By + Cz + D = 0 \]
- detection if a line intersects a polygon, touches a vertex or passes through

Typical problem

```csharp
double x = -1; double p = ....;
while (x < +1)
{
    if (x == p) Console.Out.WriteLine("*** ");
    x += p;
}
/*  if p = 0.1 then no output,  if p = 0.25 then expected output */
```
Delaunay triangulation & Voronoi diagram

Point inside of a circle given by three points – problems with meshing points in regular rectangular grid.

It can be seen that the DT & is very sensitive to a point position change

?? ROBUSTNESS ??
Vectors and Points in Geometry

- **Vectors** – movable, no fixed position
- **Points** – no size, position fixed in the GIVEN coordinate system

Coordinate systems:

- Cartesian – left / right handed
- Polar
- Spherical, Cylindrical
- and many others, e.g. Confocal Ellipsoidal Coordinates
  http://mathworld.wolfram.com/ConfocalEllipsoidalCoordinates.html
Eurographics 2013

Floating point

- Not all numbers are represented correctly
- Logarithmic arithmetic
- Continuous fractions
- Interval arithmetic

Numerically NOT valid identities due to limited precision

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

\[
\pi = \frac{4}{1 + \frac{1^2}{3 + \frac{2^2}{5 + \frac{3^2}{\ldots}}}}
\]

\[
\pi = [3; 7,15,1,292,1,1,1,2,1,3,1 \ldots]
\]

\[
\begin{align*}
x + y &= [a + c, b + d] & x &= [a, b] \\
x - y &= [a - d, b - c] & y &= [c, d] \\
x \times y &= [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)] \\
x \div y &= [\min(a/c, a/d, b/c, b/d), \max(a/c, a/d, b/c, b/d)] \text{ if } y \neq 0
\end{align*}
\]
Statements like
if <float> = <float> then .... or if <float> ≠ <float> then ....
should not be allowed in programming languages

Quadratic equation - more reliable results

\[ at^2 + bt + c = 0 \]
usually solved as
\[ t_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]

If \( b^2 \gg 4ac \) then
\[ q = -(b + \text{sign}(b)\sqrt{b^2 - 4ac})/2 \]
\[ t_1 = \frac{q}{a} \quad t_2 = \frac{c}{a} \]

The discriminant should be computed with a twice precision

Vieta’s formula
\[ t_1 + t_2 = -\frac{b}{a} \quad t_1t_2 = \frac{c}{a} \]
Function value computation at $x = 77617, \ y = 33096$

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

$f = 6.33835 \times 10^{29}$ single precision

$f = 1.1726039400532 \times 10^{30}$ double precision

$f = 1.1726039400531786318588349045201838 \times 10^{30}$ extended precision

The correct result is “somewhere” in the interval of

$[-0.82739605994682136814116509547981629, 2005, -0.82739605994682136814116509547981629, 1986]$}

Exact solution

$$f(x, y) = -2 + \frac{x}{2y} = \frac{54767}{66192}$$
Summation is one of often used computations.

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

or

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

The result should be only one.

The correctness in summation is very important in power series computations.

!!!!ORDER of summation

\[
\sum_{n=1}^{10^6} \frac{1}{n} = 14.357357
\]

\[
\sum_{n=10^6}^{1} \frac{1}{n} = 14.392651
\]
Recursion

Towers of Hanoi

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

Ackermann function

\[
A(m, n) = \begin{cases} 
  n + 1 & \text{if } m = 0 \\
  A(m - 1, 1) & \text{if } M > 0 \text{ and } n = 0 \\
  A(m - 1, A(m, n - 1)) & \text{if } m > 0 \text{ and } N > 0 
\end{cases}
\]

The value of the function grows very fast as

\[
A(4,4) = 2^{2^{2^{2^{65536}}}} = 2^{2^{10^{197296}}}
\]
Numerical computations

\[ Ax = b \quad x = A^{-1}b \]

Hilbert’s Matrix Inversion

\[ H_{ij} = \frac{1}{i+j-1} \]

\[ H_{ij}^{-1} = (-1)^{i+j}(i+j-1)\binom{n+i-1}{n-j} \binom{n+j-1}{n-i} (i+j-2)^2 \]

![Graph showing the error in the Hilbert matrix inversion against the order of the matrix. The x-axis represents the order of the Hilbert matrix, ranging from 3 to 20, and the y-axis represents the error, ranging from 1E-15 to 1,0E+19. The graph includes error markers for different orders, indicating the accuracy of the inversion.]
Mathematical “forms”

There are several “forms”:

**Implicit**  
\[ F(x, y, z) = 0 \]  
or  
\[ F(x) = 0 \]  
or  
\[ F(x) = 0 \] (system of equations)

There is no orientation, e.g.

- if \( F(x) = 0 \) is a iso-curve there is no hint how to find another point of this curve, resp. a line segment approximating the curve => tracing algorithms
- if \( F(x) = 0 \) is a iso-surface there is no hint how to find another point of this surface => iso-surface extraction algorithms

**Parametrical**  
\[ x = x(u) \quad x = x(u, v) \]

Points of a curve are “ORDERED” according to a parameter \( u \), resp. \( u, v \)

**Explicit**  
\[ z = f(x) \]
\[ z = f(x, y) \] [actually 2 ½ D]

For the given value \( x \), resp. \( x, y \) we get function value \( z \)
Implicit form

- Is used for separation - for detection if a point is inside or outside, e.g. a half-plane or a circle etc.
- There is always a question how to compute $x$ of $F(x) = 0$ as there are several solutions in general
- Complexity of computations $\times$ precision of computation

Compiler optimization is **DANGEROUS** in general - numerical precision

$$x^2 - y^2 = (x + y)(x - y)$$

$$\begin{vmatrix}
A_x & A_y & A_x^2 + A_y^2 & 1 \\
B_x & B_y & B_x^2 + B_y^2 & 1 \\
C_x & C_y & C_x^2 + C_y^2 & 1 \\
D_x & D_y & D_x^2 + D_y^2 & 1
\end{vmatrix} = \begin{vmatrix}
A_x - D_x & A_y - D_y & (A_x^2 - D_x^2) + (A_y^2 - D_y^2) \\
B_x - D_x & B_y - D_y & (B_x^2 - D_x^2) + (B_y^2 - D_x^2) \\
C_x - D_x & C_y - D_y & (C_x^2 - D_x^2) + (C_y^2 - D_x^2) \\
D_x - D_x & D_y - D_y & (D_x^2 - D_x^2) + (D_y^2 - D_x^2)
\end{vmatrix} > 0$$
Another example

$$f(x) = \frac{1 - \cos x}{x^2}$$

Computed values are wrong in an interval close to zero. In the interval $(-\epsilon, \epsilon)$ the function values are ZERO instead of 0.5!!!
Examples – what happened?

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.
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. From the CNN article:

“The internal SRI [Inertial Reference System] software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer.”

The conversion from the floating point to the integer representation is very dangerous as it is not reported by an exception and stored value represents an existing number.
Patriot Missile Failure

The system was originally designed in mid-1960 for a short and flexible operation. The system was actually running for more than 100 hours) and for intercepting cruise missiles running at MACH 2 speed and 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 and speed computation in floats. The clock setting to 1/10 was a critical issue and not acceptable even for application in sport activities at that time. Unfortunately $1/10 = 1/2^4 + 1/2^5 + 1/2^8 + 1/2^9 + 1/2^{12} + \ldots$. 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” area.

As a result of the fault assumptions, incorrect software design and irresponsible attitude of the army officials, 28 Americans were killed and over 100 other people injured in the Iraq’s Scud missile attack in Dhahran, Saudi Arabia on February 25, 1991 according to the GAO report.
Sleipner offshore platform sinking

Another well known example is the Sleipner offshore platform sinking. 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 finite element system and the shear stresses were underestimated nearly by 50%. It led to serious cracks in the structure and leakage that the pumps were unable to cope with. The sinking of the platform estimated cost is about $700 million.
We have presented some basic facts on numerical precision and examples of some disasters. Many engineering problems are somehow connected with geometry and geometrical computations with respecting physical phenomena etc.

The majority of computations are made in the Euclidean space representation and with the Cartesian coordinate system.

In the following we will show how the non-Euclidean representation, actually its projective extension, and the principle of duality can be used to solve some problems in a simple, robust and elegant ways.
Projective Space

\[ X = [X, Y]^T \quad X \in E^2 \]
\[ x = [x, y: w]^T \quad x \in P^2 \]

Conversion:

\[ X = [x/w, y/w]^T \quad w \neq 0 \]

If \( w = 0 \) then \( x \) represents “an ideal point” - a point in infinity, i.e. it is a directional vector.

The Euclidean space \( E^2 \) is represented as a plane \( w = 1 \).

Equivalent “mathematical” notation often used:

\[ x = [w:x,y]^T \]

generally for \( E^n \)

\[ x = [x_0:x_1,\ldots,x_n]^T \]

i.e. homogeneous coordinate is the first.
Points and vectors

- Vectors are “freely movable” – not having a fixed position
  \[ a_1 = [x_1, y_1: 0]^T \]

- Points are not “freely movable” – they are fixed to an origin of the current coordinate system
  \[ x_1 = [x_1, y_1: w_1]^T \quad \text{and} \quad x_2 = [x_2, y_2: w_2]^T \]
  usually in textbooks \( w_1 = w_2 = 1 \)

A vector \( A = X_2 - X_1 \) in the Euclidean coordinate system – **CORRECT**

\[ A = [A_x, A_y]^T = [X_2, Y_2]^T - [X_1, Y_1]^T \]
Horrible “construction”! DO NOT USE IT – IT IS TOTALLY WRONG

\[ \mathbf{a} = x_2 - x_1 = [x_2 - x_1, y_2 - y_1 : w_2 - w_1]^T \]

as \( w_1 = w_2 = 1 \)

\[ \mathbf{a} = [x_2 - x_1, y_2 - y_1 : 1 - 1]^T = [x_2 - x_1, y_2 - y_1 : 0]^T \]

What happen if \( w_1 \neq w_2 \) due to numerical representation?

\[ \mathbf{a} = x_2 - x_1 = [x_2 - x_1, y_2 - y_1 : w_2 - w_1]^T = [a_x, a_y : \varepsilon]^T \]

and \( \varepsilon \neq 0 \)

This is considered as a point !!!

This was presented as “How a vector” is constructed in the projective space \( P^k \) in a textbook!! \textbf{WRONG, WRONG, WRONG}

This construction has been found in SW!!
An Euclidean vector $\mathbf{a}$ given by two points expressed in

- the Euclidean coordinates

$$
\mathbf{a} = \left[ \frac{w_1 x_2 - w_2 x_1}{w_1 w_2}, \frac{w_1 y_2 - w_2 y_1}{w_1 w_2} \right]^T
$$

- the homogeneous coordinates

$$
\mathbf{a} = x_2 - x_1 = [w_1 x_2 - w_2 x_1, w_1 y_2 - w_2 y_1 : w_1 w_2]^T
$$

We use homogeneous coordinates to represent a denominator of a fraction. This is the correct solution, but what is the interpretation?

A “difference” of coordinates of two points is a vector in the mathematical meaning and $w_1 w_2$ is a “scaling” factor actually.

Actually the division operation is postponed and not performed immediately. A vector in projective notation

$$
\mathbf{a} = x_2 - x_1 = [w_1 x_2 - w_2 x_1, w_1 y_2 - w_2 y_1 : w_1 w_2]^T \triangleq \left[ \frac{w_1 x_2 - w_2 x_1}{w_1 w_2}, \frac{w_1 y_2 - w_2 y_1}{w_1 w_2} : 0 \right]^T
$$

where: $\triangleq$ means projectively equivalent.
A Euclidean vector in the projective representation
(if the vector length matters)

\[ \mathbf{a} = \mathbf{x}_2 - \mathbf{x}_1 = [w_1 x_2 - w_2 x_1, w_1 y_2 - w_2 y_1 : w_1 w_2]^T \triangleq [\frac{w_1 x_2 - w_2 x_1}{w_1 w_2}, \frac{w_1 y_2 - w_2 y_1}{w_1 w_2} : 0]^T \]

A vector in the projective space is given by coordinates \( x, y, w \) as

\[ \mathbf{a} = \mathbf{x}_2 - \mathbf{x}_1 = [x_2 - x_1, y_2 - y_1 : w_2 - w_1]^T \]

[\Rightarrow \text{Linear interpolation with a non-linear monotonic interpolation}]

We have to strictly distinguish meaning of one dimensional array [vector], i.e. if we are working with:

- **points**, i.e. a data structure represent point coordinates, or
- **vectors**, i.e. a data structure represent a vector in the mathematical meaning

**VECTORS x POINTS**
Duality

For simplicity, let us consider a line \( p \) defined as:

\[
aX + bY + c = 0
\]

We can multiply it by \( w \neq 0 \) and we get:

\[
awX + bwY + cw = 0
\]

\( w \neq 0 \)

As \( x = wX \) and \( y = wY \)

\[
ax + by + cw = 0 \quad \text{i.e.} \quad p^T x = 0
\]

\[
p = [a, b: c]^T \quad x = [x, y: w]^T = [wX, wY: w]^T
\]

A line \( p \in \mathbb{E}^2 \) is actually a plane \( \rho \) in the projective space \( \mathbb{P}^2 \) (point \( x = [0,0: 0]^T \) excluded)
Duality

From the mathematical notation \( p^T x = 0 \) we cannot distinguish whether \( p \) is a line and \( x \) is a point or vice versa in the case of \( P^2 \). It means that

- a point and a line are dual in the case of \( P^2 \), and
- a point and a plane are dual in the case of \( P^3 \).

The principle of duality in \( P^2 \) states that:

Any theorem in \( E^2 \) remains true when we interchange the words “point” and “line”, “lie on” and “pass through”, “join” and “intersection”, “collinear” and “concurrent” and so on.

Similarly for the \( E^3 \) case.

**Once the theorem has been established, the dual theorem is obtained as described above.**

This helps a lot to solve some geometrical problems.
Examples of dual objects and operators

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Dual primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P^2$</td>
<td></td>
</tr>
<tr>
<td>Point</td>
<td>Line</td>
</tr>
<tr>
<td>Line</td>
<td>Point</td>
</tr>
<tr>
<td>$P^3$</td>
<td></td>
</tr>
<tr>
<td>Point</td>
<td>Plane</td>
</tr>
<tr>
<td>Plane</td>
<td>Point</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operator</th>
<th>Dual operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Join</td>
<td>Intersect</td>
</tr>
<tr>
<td>Intersect</td>
<td>Join</td>
</tr>
</tbody>
</table>

**Computational sequence for a problem is the same as for a dual problem.**
**Definition**

The **cross product** of the two vectors

\[ x_1 = [x_1, y_1: w_1]^T \text{ and } x_2 = [x_2, y_2: w_2]^T \]

is defined as:

\[ x_1 \times x_2 = \text{det} \begin{bmatrix} i & j & k \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix} \]

where: \( i = [1,0:0]^T \quad j = [0,1:0]^T \quad k = [0,0:1]^T \)

or as

\[ x_1 \times x_2 = \begin{bmatrix} 0 & -w_1 & y_1 \\ w_1 & 0 & -x_1 \\ -y_1 & x_1 & 0 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ w_2 \end{bmatrix} = Tx_2 \]

Please, note that homogeneous coordinates are used.
Intersection of two lines

Let two lines $p_1$ and $p_2$ are given by

$$p_1 = [a_1, b_1: c_1]^T \quad \text{and} \quad p_2 = [a_2, b_2: c_2]^T$$

We have to solve a system of linear equations $Ax = b$

\[
\begin{align*}
    a_1 x + b_1 y + c_1 &= 0 \\
    a_2 x + b_2 y + c_2 &= 0
\end{align*}
\]

i.e.

\[
\begin{bmatrix}
    a_1 & b_1 \\
    a_2 & b_2 
\end{bmatrix}
\begin{bmatrix}
    x \\
    y 
\end{bmatrix} =
\begin{bmatrix}
    q_1 \\
    q_2 
\end{bmatrix}
\quad \text{and} \quad
\begin{bmatrix}
    q_1 \\
    q_2 
\end{bmatrix} =
\begin{bmatrix}
    -c_1 \\
    -c_2 
\end{bmatrix}^*
\]

Then well known formula is used

\[
x = \frac{\text{det}_x}{\text{det}} = \frac{\det}
\begin{bmatrix}
    a_1 & b_1 \\
    a_2 & b_2 
\end{bmatrix}
\begin{bmatrix}
    q_1 \\
    q_2 
\end{bmatrix}
\quad \text{and} \quad
y = \frac{\text{det}_y}{\text{det}} = \frac{\det}
\begin{bmatrix}
    a_1 & q_1 \\
    a_2 & q_2 
\end{bmatrix}
\begin{bmatrix}
    a_1 & b_1 \\
    a_2 & b_2 
\end{bmatrix}
\]

But what if $\text{det}$ is small? What is $\text{eps}$?

Usually a sequence like if $\text{abs(det(\.))} \leq \text{eps}$ then is used.
Theorem
Let two lines $p_1$ and $p_2$ be given. Then the coordinates of an intersection point $x$, which is defined by those two lines, are determined as the cross product of homogeneous coefficients of those lines as

$$x = p_1 \times p_2$$

$$x = [x, y: w]^T$$

Proof
We are actually looking for a solution to the following equations:

$$x^T p_1 = 0$$
$$x^T p_2 = 0$$

where: $x = [x, y: w]^T$

Note * usually a line is in its implicit form as $ax + by = q$ instead of $ax + by + c = 0$, or in the explicit form as $= kx + q$. 

A Line given by two points

Given two points $x_1$ and $x_2$ and we want to compute a line given by those two points, i.e. we need to compute 3 values $a, b, c$ from two values $x_1, x_2$.

⇒ One parametric set of solutions

$$ax_1 + by_1 + c = 0$$

$$ax_2 + by_2 + c = 0$$

In a matrix form

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

How to solve it?

Select $a = 1$? What happen if a line passing the origin?

or $b = 1$? or $c = 1$ or similarly? NO, NO, NO!

BUT HOW?
Another approach

We know that a line is dual to a point in $E^2$ and vice versa.

Due to the **duality principle** in $E^2$:

\[
x = p_1 \times p_2
\]

\[
Ax = b
\]  \[<= \text{duality} \Rightarrow \]

\[
p = x_1 \times x_2
\]  \[<= \text{why different?} \Rightarrow \]

\[
Ax = 0
\]

Additional condition  \( a + b = 1? \)
Theorem
Let two points $x_1$ and $x_2$ be given in the projective space. Then the coefficients of the $p$ line, which is defined by those two points, are determined as the cross product of their homogeneous coordinates

$$p = x_1 \times x_2 = [a, b: c]^T$$

Proof
Let the $p$ line be defined in homogeneous coordinates as

$$ax + by + cw = 0$$

We are actually looking for a solution to the following equations:

$$p^T x_1 = 0 \quad p^T x_2 = 0$$

where: $p = [a, b: c]^T$

Note that $c$ represents a “distance” from the origin of the coordinate system.
It means that any point $x$ that lies on the $p$ line must satisfy both the equation above and the equation $p^T x = 0$ in other words the $p$ vector is defined as

$$p = x_1 \times x_2 = \det \begin{bmatrix} i & j & k \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix}$$

We can write

$$\begin{bmatrix} x_1 \times x_2 \end{bmatrix}^T x = 0$$

$$\det \begin{bmatrix} x & y & w \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix} = 0$$

Note that the cross product and the dot product are the instructions in Cg/HLSL on GPU.
Evaluating the determinant \( \det \begin{bmatrix} a & b & c \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix} = 0 \)

we get the line coefficients of the line \( p \) as:

\[
\begin{align*}
    a &= \det \begin{bmatrix} y_1 & w_1 \\ y_2 & w_2 \end{bmatrix} \\
    b &= -\det \begin{bmatrix} x_1 & w_1 \\ x_2 & w_2 \end{bmatrix} \\
    c &= \det \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}
\end{align*}
\]

**Note:**

1. A line \( ax + by + c = 0 \) is a one parametric set of coefficients

\[
p = [a, b; c]^T
\]

From two values \( x_1 \) and \( x_2 \) we have to compute 3 values, coefficients \( a \), \( b \) and \( c \)

2. For \( w = 1 \) we get the standard cross product formula and the cross product defines the \( p \) line, i.e. \( p = x_1 \times x_2 \) where:

\[
p = [a, b; c]^T
\]
DUALITY APPLICATION

In the projective space $P^2$ points and lines are dual. Due to duality we can directly intersection of two lines as

$$x = p_1 \times p_2 = \det \begin{bmatrix} i & j & k \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix} = [x, y: w]^T$$

If the lines are parallel or close to parallel, the homogeneous coordinate $w \rightarrow 0$ and users have to take a decision – so there is no sequence in the code like $if \ abs(\det(\ldots)) \leq \text{eps} \ then \ ...$ in the procedure.

Generally computation can continue even if $w \rightarrow 0$ if projective space is used.
Computation in Projective Space

- Extended cross product definition

- A plane $\rho$ is determined as a cross product of three given points

Due to the duality

- An intersection point $x$ of three planes is determined as a cross product of three given planes.

- Computation of generalized cross product is equivalent to a solution of a linear system of equations $\Rightarrow$ no division operation!

- Using the cross product we can continue with symbolic operations which could not be made if solution of $Ax = b$ is used.

\[
\rho = x_1 \times x_2 \times x_3 = \begin{vmatrix} i & j & k & l \\ x_1 & y_1 & z_1 & w_1 \\ x_2 & y_2 & z_2 & w_2 \\ x_3 & y_3 & z_3 & w_3 \end{vmatrix}
\]

\[
x = \rho_1 \times \rho_2 \times \rho_3 = \begin{vmatrix} i & j & k & l \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{vmatrix}
\]
We have seen that computation of

- an intersection of two lines is given as $Ax = b$
- a line given by two points is given as $Ax = 0$

Different scheme BUT

Those problems are DUAL.

Why algorithms should be different??

Cross product is equivalent to a solution of both linear systems of equations, i.e.

$Ax = b$ and $x = 0$ !

No division operations!
DISTANCE

Geometry is strongly connected with distances and their measurement. Geometry education deals strictly with the Euclidean geometry, where the distance is measured as

\[ d = \sqrt{(\Delta x)^2 + (\Delta y)^2}, \text{ resp. } \quad d = \sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2}. \]

This concept is convenient for a solution of basic geometric problems, but in many cases it results into quite complicated formula.

There are severe questions of stability and robustness in many cases.

*The main objection against the projective representation is that there is no metric.*
The distance of two points can be easily computed as

$$\text{dist} = \sqrt{\xi^2 + \eta^2}/(w_1w_2)$$

where: \( \xi = w_1x_2 - w_2x_1 \) \hspace{1cm} \( \eta = w_1y_2 - w_2y_1 \)

Also a distance of a point \( x_0 \) from a line in \( \mathbb{E}^2 \) can be computed as

$$\text{dist} = \frac{a^T x_0}{w_0\sqrt{a^2 + b^2}}$$

where: \( x_0 = [x_0, y_0 : w_0]^T \) \hspace{1cm} \( a = [a, b : c]^T \)

The extension to \( \mathbb{E}^3/\mathbb{P}^3 \) is simple and the distance of a point \( x_0 \) from a plane in \( \mathbb{E}^3 \) can be computed as

$$\text{dist} = \frac{a^T x_0}{w_0\sqrt{a^2 + b^2 + c^2}}$$

where: \( x_0 = [x_0, y_0, z_0 : w_0]^T \) \hspace{1cm} \( a = [a, b, c : d]^T \).
In many cases we do not need actually a distance, e.g. for a decision which 
object is closer, and \( \text{distance}^2 \) can be used instead, i.e. for the \( \mathbb{E}^2 \) case

\[
\text{dist}^2 = \frac{(a^T x_0)^2}{w_0^2(a^2 + b^2)} = \frac{(a^T x_0)^2}{w_0^2 n^T n}
\]

where: \( a = [a, b: c]^T = [n: c]^T \) and the normal vector \( n \) is not normalized.

If we are comparing distances of points \( x_0 \) from the given line \( p \) we can use "pseudo-distance" for comparisons

\[
(pseudo\_dist)^2 = \frac{(a^T x_0)^2}{w_0^2}
\]

Similarly for a plane \( \rho \) in the case of \( \mathbb{E}^3 \)

\[
\text{dist}^2 = \frac{(a^T x_0)^2}{w_0^2(a^2 + b^2 + c^2)} = \frac{(a^T x_0)^2}{w_0^2 n^T n} \quad \text{and} \quad (pseudo\_dist)^2 = \frac{(a^T x_0)^2}{w_0^2}
\]

where: \( a = [a, b, c: d]^T = [n: d]^T \)
Geometric transformations with points
(note $X = x/w, Y = y/w, w \neq 0$):

**Translation** by a vector $(A,B) \triangleq [a,b:c]^T$, i.e. $A = a/c$, $B = b/c$, $c \neq 0$:

In the Euclidean space:

$$x' = Tx$$

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & A \\ 0 & 1 & B \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} x + Aw \\ y + Bw \\ w \end{bmatrix} \triangleq \begin{bmatrix} x/w + A \\ y/w + B \\ 1 \end{bmatrix} = \begin{bmatrix} X + A \\ Y + B \\ 1 \end{bmatrix}$$

In the projective space:

$$x' = T'x$$

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} c & 0 & a \\ 0 & c & b \\ 0 & 0 & c \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} cx + aw \\ cy + bw \\ cw \end{bmatrix} \triangleq \begin{bmatrix} (cx + aw)/(cw) \\ (cy + bw)/(cw) \\ 1 \end{bmatrix} = \begin{bmatrix} x/w + a/c \\ y/w + b/c \\ 1 \end{bmatrix} = \begin{bmatrix} X + A \\ Y + B \\ 1 \end{bmatrix}$$

and $\det(T') = c^3$. For $c = 1$ we get a standard formula

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & A \\ 0 & 1 & B \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
**Rotation** by an angle \((\cos \varphi, \sin \varphi) = \left(\frac{a}{c}, \frac{b}{c}\right) \triangleq [a, b: c]^T\)

In the Euclidean space: \[ x' = Rx \]

\[
\begin{bmatrix}
x' \\
y' \\
w'
\end{bmatrix}
= \begin{bmatrix}
\cos \varphi & -\sin \varphi & 0 \\
\sin \varphi & \cos \varphi & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
w
\end{bmatrix}
= \begin{bmatrix}
\cos \varphi & -\sin \varphi & 0 \\
\sin \varphi & \cos \varphi & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
X \\
Y \\
1
\end{bmatrix}
\]

In the projective space: \[ x' = R'x \]

\[
\begin{bmatrix}
x' \\
y' \\
w'
\end{bmatrix}
= \begin{bmatrix}
a & -b & 0 \\
b & a & 0 \\
0 & 0 & c
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
w
\end{bmatrix}
= \begin{bmatrix}
a x - b y \\
b x + a y \\
c w
\end{bmatrix}
\]

\[
\begin{bmatrix}
(ax - by)/(cw) \\
(bx + ay)/(cw) \\
1
\end{bmatrix}
= \begin{bmatrix}
x a & y b \\
\frac{x}{w} & \frac{y}{w} \\
\frac{x}{w} & \frac{y}{w} \\
\frac{x}{w} & \frac{y}{w} \\
1
\end{bmatrix}
= \begin{bmatrix}
X \cos \varphi - Y \sin \varphi \\
X \sin \varphi + Y \cos \varphi \\
1
\end{bmatrix}
\]

as \(c^2 = (a^2 + b^2)\) by definition, \(\det(R') = (a^2 + b^2)c = c^3\)

As we are working in the projective space, it does not matter
Scaling by a factor \((S_x, S_y) = \left(\frac{s_x}{w_s}, \frac{s_y}{w_s}\right) \triangleq [s_x, s_y; w_s]^T\)

\[
x' = Sx \quad \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}
\]

\[
x' = S'x \quad \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & w_s \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}
\]

\[\text{det}(S') = s_x s_y w_s\]

It is necessary to note that the determinant of a transformation matrix \(Q\), i.e. matrices \(T', R', S'\), is \(\text{det}(Q) \neq 1\) in general, but as the formulation is in the projective space, there is no need to “normalize” transformations to \(\text{det}(Q) = 1\) even for rotation.

It can be seen that if the parameters of a geometric transformation are given in the homogeneous coordinates, no division operation is needed at all.
Transformation of lines and planes

\[ E^2 \]
\[ p = x_1 \times x_2 \]
\[ x = p_1 \times p_2 \]

\[ E^3 \]
\[ \rho = x_1 \times x_2 \times x_3 \]
\[ x = \rho_1 \times \rho_2 \times \rho_3 \]

Dual problem

In graphical applications position of points are changed by an interaction, i.e.

\[ x' = Tx \]

The question is how coefficients of a line, resp. a plane are changed if the points are transformed without a need to be recomputed from the definition.

It can be proved that

\[ p' = (Tx_1) \times (Tx_2) = \det(T)(T^{-1})^T p \triangleq (T^{-1})^T p \]

or

\[ \rho' = (Tx_1) \times (Tx_2) \times (Tx_3) = \det(T)(T^{-1})^T \rho \triangleq (T^{-1})^T \rho \]
Transformation of lines and planes

As the computation is made in the projective space we can write

$$ p' = (T^{-1})^T p = [a', b', c']^T $$
for lines in $E^2$

or

$$ \rho' = (T^{-1})^T \rho = [a', b', c': d']^T $$
for planes in $E^3$

**This simplifies computations**

Transformation matrices for lines, resp. for planes are **DIFFERENT** from transformations for points!

Note that a normal vector of a line is actually a co-vector, i.e. an oriented “surface”.
Transformation of lines and planes

Transformation about a general axis in $E^3 / P^3$

$$x(t) = x_A + st$$

Usually used transformation ($T$ is translation):

$$Q = T^{-1} R_{zx}^{-1} R_{yz}^{-1} R(\varphi) R_{zx} R_{xy} T$$

$$R_{yz} = \begin{bmatrix} 1 & 0 & c & 0 & 0 \\ 0 & 0 & \frac{b}{\sqrt{b^2 + c^2}} & \frac{-b}{\sqrt{b^2 + c^2}} & 0 \\ 0 & \frac{b}{\sqrt{b^2 + c^2}} & 0 & 0 & 0 \\ 0 & 0 & \frac{c}{\sqrt{b^2 + c^2}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$s = [a, b, c]^T$ is an axis directional vector. This is unstable if $\sqrt{b^2 + c^2} \to 0$ and not precise if $b^2 \gg c^2$ or vice versa.

That is generally computationally complex and unstable as a user has to select which axis is to be used for a rotation.
Transformation of lines and planes

Transformation about an axis $n$
in the Euclidean space $E^3$

$$X = X \cos \varphi + (1 - \cos \varphi)(n^T X).n + (n \times X) \sin \varphi$$

$$Q = I \cos \varphi + (1 - \cos \varphi)(n \otimes n) + W \sin \varphi$$

where: $n \otimes n = n \cdot n^T$ is a matrix.
In the Euclidean space $E^3$ the vector $n$ has to be normalized

The matrix $W$ is defined as: $Wv = w \times v$

$$W = \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix}$$
in our case
Projection

Standard projection

\[
\begin{bmatrix}
  x_I \\
y_I \\
w_I
\end{bmatrix} = T_{\text{projection}} \begin{bmatrix}
  x_I \\
y_I \\
1
\end{bmatrix}
\]

How to determine \( z_I \) coordinate?

\[
x_I = (1 - \lambda)x_P + \lambda x_Q
\]

\[
y_I = (1 - \lambda)y_P + \lambda y_Q
\]

\[
\frac{1}{z_I} = (1 - \lambda) \frac{1}{z_P} + \lambda \frac{1}{z_Q}
\]

⇒ "Reverse" depth computation, e.g. for correct intensity computation if perspective projection is used.
Computation in Projective Space

Linear interpolation

Linear parameterization:

\[ X(t) = X_0 + (X_1 - X_0) \ t \quad t \in (-\infty, \infty) \]

Non-linear monotonous parameterization:

\[ x(t) = x_0 + (x_1 - x_0) \ t \quad t \in (-\infty, \infty) \]

\[ y(t) = y_0 + (y_1 - y_0) \ t \]

\[ z(t) = z_0 + (z_1 - z_0) \ t \]

\[ w(t) = w_0 + (w_1 - w_0) \ t \]

- we can interpolate using homogeneous coordinates without “normalization” to \( \mathbb{E}^k \) !!
- homogeneous coordinate \( w \geq 0 \)

In many algorithms, we need “monotonous” parameterization, only
Computation in Projective Space

Spherical interpolation

\[ \text{slerp}(X_0, X_1, t) = \frac{\sin[(1 - t)\Omega]}{\sin \Omega} X_0 + \frac{\sin[t\Omega]}{\sin \Omega} X_1 \]

Instability occurs if \( \Omega \to k\pi \).

Mathematically formula is correct; in practice the **code is generally incorrect**! [ 0 0 ]

\[ \text{slerp}(X_0, X_1, t) = \text{slerp}_p(X_0, X_1, t) = \frac{\sin[(1 - t)\Omega]}{\sin \Omega} X_0 + \frac{\sin[t\Omega]}{\sin \Omega} X_1 \]

≡ \[ \frac{\sin[(1 - t)\Omega]}{\sin \Omega} X_0 + \frac{\sin[t\Omega]}{\sin \Omega} X_1 : \sin \Omega \]^T  \text{projective scalar used}  

Homogeneous coordinates

⇒ better numerical stability & division operation can be postponed

Homogeneous coordinate
Computation in Projective Space

\[ x(t) = \text{slerp}(x_0, x_1, t) = \frac{\sin[(1 - t)\Omega]}{\sin \Omega} x_0 + \frac{\sin[t\Omega]}{\sin \Omega} x_1 \]

\[ \triangleq [\sin[(1 - t)\Omega] x_0 + \sin[t\Omega] x_1 : \sin \Omega]^T \]

What is a result in the Euclidean space of a spherical interpolation with non-linear parameterization, i.e. \( w_i \neq 1 \)?

\[ x(t) = \frac{\sin[(1 - t)\Omega]}{\sin \Omega} x_0 + \frac{\sin[t\Omega]}{\sin \Omega} x_1 \quad \quad w(t) = \frac{\sin[(1 - t)\Omega]}{\sin \Omega} w_0 + \frac{\sin[t\Omega]}{\sin \Omega} w_1 \]

\[ X(t) = \frac{\sin[(1 - t)\Omega]}{\sin \Omega} x_0 + \frac{\sin[t\Omega]}{\sin \Omega} x_1 = \frac{\sin[(1 - t)\Omega] x_0 + \sin[t\Omega] x_1}{\sin[(1 - t)\Omega] w_0 + \sin[t\Omega] w_1} \]

If represented as “projective scalar” value

\[ x(t) = [\sin[(1 - t)\Omega] x_0 + \sin[t\Omega] x_1 : \sin[(1 - t)\Omega] w_0 + \sin[t\Omega] w_1]^T \]

=> Better numerical stability
Influence of homogeneous values \( w_i \neq 1 \) & \( w_i \neq 0 \) (Fun1: \( w_i = 1 \))

Can be used for:
- relative position comparison
- interpolation if a “linear” parameterization is required \( w_i = 1 \)
Computation in Projective Space

Intersection line – plane (Cyrus-Beck clipping algorithm)

\[ x(t) = x_0 + (x_1 - x_0) \ t = x_0 + s \ t \]

Computation in projective space!

\[ s = x_1 - x_0 = [x_1 - x_0, y_1 - y_0, z_1 - z_0: w_1 - w_0]^T \quad t \in (-\infty, \infty) \]

\[ a^T x = 0 \quad a = [a, b, c: d]^T \quad x_i = [x_i, y_i, z_i: w_i]^T \]

Euclidean solution:

\[ t = -\frac{a^T x_0}{a^T s} \]

Projective solution:

\[ \tau = -a^T x_0 \quad \text{and} \quad \tau_w = a^T s \]

\[ \tau = [\tau: \tau_w]^T \]

if \( \tau_w < 0 \) then \( \tau := -\tau \)

projective scalar:

Test: if \( t < t_{\text{min}} \) then ...modification: \( \tau_w \geq 0 \)

if \( \tau \star \tau_{\text{min}} > \tau_w \star \tau_{\text{min}} \) then .....  

- An intersection of a plane with a line in \( \mathbb{E}^2 / \mathbb{E}^3 \) can be computed efficiently, no division operation, but comparison operations must be modified!
- Cyrus-Beck line clipping algorithm 10-25% faster
Line Clipping – convex polygon

procedure CLIP_Line ( x_A , x_B );
/* x_A=[x_A,y_A:w_A]^T x_B=[x_B,y_B:w_B]^T */
begin /* p=[a,b:c]^T given - NO STEP 1 */
{1} p := x_A x x_B;    /* p: ax+by+c = 0 */
{2} for k:=0 to N-1 do /* x_k=[x_k,y_k,w_k]^T */
{3}     if p^T x_k ≥ 0 then c_k:=1 else c_k:=0;
{4} if c=[0...0]^T or c=[1...1]^T then EXIT;
{5} i:= TAB1[c];    j:= TAB2[c];
{6} x_A := p x e_i ;  x_B := p x e_j ;
{7} DRAW (x_A; x_B )    /* e_i – i-th edge */
end /* CLIP_Line */
/* c identifies an edge intersected */

TOO COMPLEX?

NO SIMPLE, ROBUST and FAST

- Cohen-Sutherland
- Liang-Barsky
- Hodgman
- Skala – modification of Clip_L for line segments

NO SIMPLE, ROBUST and FAST

• Skala,V.: A new approach to line and line segment clipping in homogeneous coordinates, The Visual Computer, SpringerVol.21, No.11, pp.905-914, 2005
For a rectangular normalized window

\[ x_A := p \times e_i \]

For the edge \( y = -1 \), i.e. \( y + 1 = 0 \)

\[ x_A = [x_A, y_A; w_A]^T = \begin{bmatrix} i & j & k \\ a & b & c \\ 0 & 1 & 1 \end{bmatrix} = [b - c, -a; a] \]

\[ \Delta \begin{bmatrix} b - c, -a \\ a \end{bmatrix}, \frac{-1}{a} : 1]^T = \begin{bmatrix} b - c \\ -1 \end{bmatrix} \]

Actually expression for \( y_A \), resp. for \( x_A \) is given by the window edge.

**No multiplication or division operations**

A simple modification if a line is given parametrically (in the Euclidean or projective space) as \( x(t) = x_A + st \)

Simple modification for non-convex polygon but it requires intersections sorting \( => O(M \ lg M) \)
function CODE (x); /* Cohen Sutherland window coding */
begin  c:= [0000];  /* land / lor – bitwise operations and / or */
  if x<x_min then c:=[1000] else if x>x_max then c:= [0100];
  if y<y_min then c:=c lor [1001] else if y>y_max then c:=c lor [0010];
CODE := c
end /*CODE */;
procedure Clip_LineSegment (x_A , x_B); /* short x long line segments */
begin  c_A := CODE (x_A);  c_B := CODE (x_B);
  if (c_A lor c_B) = 0 then { output (x_A; x_B); EXIT }  /* inside */
  if (c_A land c_B) ≠ 0 then EXIT;  /* outside */
  p := x_A x x_B;      /* ax+by+c = 0; p = [a,b,c]^T */
  for k:=0 to 3 do /* x_k=[x_k,y_k: w_k]^T  c=[ c_3, c_2, c_1, c_0 ]^T */
    if p^T*x_k ≥ 0 then c_k:=1 else c_k:=0;
  if c = [0000]^T or c = [1111]^T then EXIT;
  i:= TAB1[c];  j:= TAB2[c]; /* original code x_A := p x e_i ; x_B := p x e_j ; DRAW (x_A; x_B) */
/* additional code for Line Segment clipping */
if \( c_A \neq 0 \) and \( c_B \neq 0 \)
then \{ \( x_A := p \times e_i \); \( x_B := p \times e_j \) \} /* two intersections */
else /* only one intersection point */
if \( c_A = 0 \) then /* \( x_B \) is outside */
\{ \( c_B \) land MASK[\( c \)] \neq 0 \)
then \( x_B := p \times e_i \)
else \( x_B := p \times e_j \)
\}
else if \( c_B = 0 \) then /* \( x_A \) is outside */
\{ \( c_A \) land MASK[\( c \)] \neq 0 \)
then \( x_A := p \times e_i \)
else \( x_A := p \times e_j \)
\};
output \( (x_A, x_B) \)
end /* Clip_LineSegment */

Algorithm can be extended to a convex polygon clipping & modified for parametric lines/rays as well

For a convex \( n \)-sided convex polygon the table can be generated synthetically.
Modification for non-convex polygon possible.
Computation in Projective Space - Barycentric coordinates

Let us consider a triangle with vertices $X_1$, $X_2$, $X_3$

A position of any point $X \in \mathbb{E}^2$ can be expressed as

$$a_1X_1 + a_2X_2 + a_3X_3 = X \quad a_1Y_1 + a_2Y_2 + a_3Y_3 = Y$$

Additional condition:

$$a_1 + a_2 + a_3 = 1 \quad 0 \leq a_i \leq 1 \quad i = 1, \ldots, 3\quad a_i = \frac{p_i}{p}$$

$$\begin{bmatrix} X_1 & X_2 & X_3 \\ Y_1 & Y_2 & Y_3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$b_i = -a_i b_4 \quad i = 1, \ldots, 3$$

$$\begin{bmatrix} X_1 & X_2 & X_3 & X \\ Y_1 & Y_2 & Y_3 & Y \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Linear system has to be solved

If points $x_i$ are given as $x_i = [x_i, y_i, z_i : w_i]^T$ and $w_i \neq 1$ then $x_i$ have to be “normalized” to $w_i = 1$, i.e. $4 \times 3 = 12$ division operations are used.
Computation in Projective Space

\[ b_1X_1 + b_2X_2 + b_3X_3 + b_4X = 0 \]
\[ b_1Y_1 + b_2Y_2 + b_3Y_3 + b_4Y = 0 \]
\[ b_1 + b_2 + b_3 + b_4 = 0 \quad b_i = -a_ib_4 \quad b_4 \neq 0 \quad i = 1, \ldots, 3 \]

Rewriting

\[
\begin{bmatrix}
  X_1 & X_2 & X_3 & X \\
  Y_1 & Y_2 & Y_3 & Y \\
  1 & 1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
  b_1 \\
  b_2 \\
  b_3 \\
  b_4
\end{bmatrix}
= 0
\]

\[ b = [b_1, b_2, b_3, b_4]^T \]
\[ \xi = [X_1, X_2, X_3, X]^T \]
\[ \eta = [Y_1, Y_2, Y_3, Y]^T \]
\[ \omega = [1, 1, 1, 1]^T \]

A solution of the linear system of equations (LSE) is equivalent to generalized cross product:

\[ b = \xi \times \eta \times \omega \]
Computation in Projective Space

For $w_i \neq 0$ each column can be multiplied by the relevant $w_i$

$$
\begin{bmatrix}
w_1 X_1 & w_2 X_2 & w_3 X_3 & w X \\
w_1 Y_1 & w_2 Y_2 & w_3 Y_3 & w Y \\
w_1 & w_2 & w_3 & w
\end{bmatrix}
\begin{bmatrix}
b_1 \\
b_2 \\
b_3 \\
b_4
\end{bmatrix} = 0
$$

where again

$$
b = [b_1, b_2, b_3, b_4]^T \quad \xi = [x_1, x_2, x_3, x]^T \quad \eta = [y_1, y_2, y_3, y]^T \quad \omega = [w_1, w_2, w_3, w]^T
$$

Barycentric coordinates (Euclidean):

$$
0 \leq (-b_1 : w_2, w_3, w) \leq 1 \quad 0 \leq (-b_2 : w_3, w_1, w) \leq 1 \quad 0 \leq (-b_3 : w_1, w_2, w) \leq 1
$$

$\Rightarrow$ Tests – point in triangle, point in tetrahedron -

**all in homogeneous coordinates**

$\Rightarrow$ **new entities**: projective scalar, projective vector

(Skala, V.: Barycentric coordinates computation in homogeneous coordinates, Computers&Graphics, 2008)
Computation in Projective Space

Area of a triangle

\[ P = \frac{1}{2} x_1^T (x_2 \times x_3) / (w_1 w_2 w_3) \]

Volume of a tetrahedron

\[ V = \frac{1}{6} x_1^T (x_2 \times x_3 \times x_4) / (w_1 w_2 w_3 w_4) \]

As the principle of duality is valid, one could ask: What is a “dual” value \( G \) to a computation of the area \( P \) if the triangle is given by three lines in the “normalized” form, e.g. \( a_1^T \cdot (a_2 \times a_3) \) instead of three points?

\[ G = a_1^T \cdot (a_2 \times a_3) \triangleq \begin{vmatrix} \cos \alpha_1 & \cos \alpha_2 & \cos \alpha_3 \\ \sin \alpha_1 & \sin \alpha_2 & \sin \alpha_3 \\ d_1 & d_2 & d_3 \end{vmatrix} = \begin{vmatrix} 1 & \cos \alpha_2 & \cos \alpha_3 \\ 0 & \sin \alpha_2 & \sin \alpha_3 \\ 0 & 0 & d_3 \end{vmatrix} = d_3 \sin \alpha_2 \]

\[ = d_3 \cdot a / (2R) = P / R \]

It can be seen that \( G = d_3 \sin \alpha_2 = P / R \), where: \( a \) is the length of the line segment on \( a_3 \) and \( R \) is a radius of the circumscribing circle.

\[ \Rightarrow \text{value } G \text{ can be used as criterion for a quality triangular meshes.} \]

Computation in Projective Space

Line in $\mathbb{E}^3$ as Two Plane Intersection

Standard formula in the Euclidean space

$$\rho_1 = [a_1, b_1, c_1 : d_1]^T = [n_1^T : d_1]^T \quad \rho_2 = [a_2, b_2, c_2 : d_2]^T = [n_2^T : d_2]^T$$

Line given as an intersection of two planes

$$s = n_1 \times n_2 \equiv [a_3, b_3, c_3]^T$$

$$x(t) = x_0 + st$$

$$x_0 = \frac{d_2 \begin{vmatrix} b_1 & c_1 \\ b_3 & c_3 \end{vmatrix} - d_1 \begin{vmatrix} b_2 & c_2 \\ b_3 & c_3 \end{vmatrix}}{\text{DET}}$$

$$y_0 = \frac{d_2 \begin{vmatrix} a_3 & c_3 \\ a_1 & c_1 \end{vmatrix} - d_1 \begin{vmatrix} a_3 & c_3 \\ a_2 & c_2 \end{vmatrix}}{\text{DET}}$$

$$z_0 = \frac{d_2 \begin{vmatrix} a_1 & b_1 \\ a_3 & b_3 \end{vmatrix} - d_1 \begin{vmatrix} a_2 & b_2 \\ a_3 & b_3 \end{vmatrix}}{\text{DET}}$$

$$\text{DET} = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}$$

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.
Eurographics 2013

Computation in Projective Space

Line in E3 as Two Plane Intersection

- Standard formula in the Euclidean space
- Plücker’s coordinates a line is given by two points. DUALITY – a point is dual to a plane and vice versa => an intersection of two planes can be computed as a dual problem
  But computationally expensive computation

- Projective formulation and simple computation
Computation in Projective Space

Line in E3 as Two Plane Intersection

\[ \rho_1 = [a_1, b_1, c_1 : d_1]^T \quad \rho_2 = [a_2, b_2, c_2 : d_2]^T \]

normal vectors are

\[ n_1 = [a_1, b_1, c_1]^T \quad n_2 = [a_2, b_2, c_2]^T \]

Directional vector of a line of two planes \( \rho_1 \) and \( \rho_1 \) is given as

\[ s = n_1 \times n_2 \]

“starting” point \( x_0 \)?

A plane \( \rho_0 \) passing the origin with a normal vector \( s, \rho_0 = [a_0, b_0, c_0 : 0]^T \)

The point \( x_0 \) is defined as

\[ x_0 = \rho_1 \times \rho_2 \times \rho_0 \]

Simple formula for matrix-vector architectures like GPU and parallel processing. Compare the standard and projective formulas
Computation in Projective Space – the nearest point

Find the nearest point on an intersection of two planes to the given point $\xi$

Simple solution:
- Translate planes $\rho_1$ and $\rho_2$ so the $\xi$ is in the origin
- Compute intersection of two planes i.s. $x_0$ and $s$
- Translate $x_0$ using $T^{-1}$

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

Solution DETAILS next

Krumm, J.: Intersection of Two Planes, Microsoft Research
The closest point to an intersection of two planes

In some applications we need to find a closest point on a line given as an intersection of two planes. We want to find a point $\xi_0'$, the closest point to the given point $\xi$, which lies on an intersection of two planes $\rho_1 \triangleq [n_1^T: d_1]^T$ and $\rho_2 \triangleq [n_2^T: d_2]^T$

This problem was recently solved by using Lagrange multipliers and an optimization approach leading to a solution of a system of linear equations with 5 equations.
Solution in the projective space

1. Translate the given point $\xi = [\xi_x, \xi_y, \xi_z: 1]^T$ to the origin – matrix $Q$

2. Compute parameters of the given planes $\rho_1$ and $\rho_2$ after the transformation as $\rho'_1 = Q^{-T} \rho_1$ and $\rho'_2 = Q^{-T} \rho_2$.

3. Compute the intersection of those two planes $\rho'_1$ and $\rho'_2$

4. Transform the point $\xi_0$ to the original coordinate system using transformation

$$n_0 = n_1 \times n_2 \quad \rho_0 \triangleq [n_0^T: 0]^T \quad \xi_0 = \rho_1 \times \rho_2 \times \rho_0 \quad \xi_0' = Q^{-1} \xi_0$$

$$Q = \begin{bmatrix} 1 & 0 & 0 & -\xi_x \\ 0 & 1 & 0 & -\xi_y \\ 0 & 0 & 1 & -\xi_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Q^{-T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ [\xi_x \ \xi_y \ \xi_z \ 1] \end{bmatrix} \quad Q^{-1} = \begin{bmatrix} 1 & 0 & 0 & \xi_x \\ 0 & 1 & 0 & \xi_y \\ 0 & 0 & 1 & \xi_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It is simple, easy to implement on GPU.
Curves and surfaces

Rational Bézier curve – Euclidean \( X = [X, Y, Z]^T \)

\[
X(t) = \frac{\sum_{i=0}^{n} B_i^n(t)w_iq_i}{\sum_{i=0}^{n} B_i^n(t)w_i} \quad 0 \leq t \leq 1 \quad B_i^n \left( \frac{n}{i} \right) (1 - t)^{n-1}t^i
\]

1st derivative

\[
\left[ \begin{array}{c}
  a' \\
  b
\end{array} \right] = \frac{a'b - ab'}{b^2} \quad \text{quite complicated}
\]

Projective \( x = [x, y, z : w]^T \)

\[
x(t) = \sum_{i=0}^{n} B_i^n(t)q_i \quad x'(t) = \sum_{i=0}^{n} (B_i^n(t))'q_i
\]

How simple!
Computation in Projective Space

Disadvantages

• Careful handling with formula in the projective space
• “Oriented” projective space is to be used, i.e. \( w \geq 0 \);
  HW could support it – simple solution
• Exponents of homogeneous vectors can overflow – special handling
  o exponents should be normalized; HW could support it
    however not supported by the current hardware
  o P_Lib – library for computation in the projective space - uses SW
    solution for normalization on GPU (C# and C++), GPU support
Computation in Projective Space

Advantages

- “Infinity” is well represented
- No division operation is needed, a division operation can be hidden to the homogeneous coordinate
- Many mathematical formula are simpler and elegant
- One code sequence solves primary and dual problems
- Supports matrix–vector operations in hardware – like GPU etc.
- Numerical computation can be faster
- Precision is nearly 2-times higher in mantissa and exponent - $\frac{a}{b}$
- More robust and stable solutions can be achieved
- System of linear equations can be solved directly without division operation, if exponent normalization is provided
Eurographics 2013

Implementation aspects and GPU

- GPU (Graphical Processing Unit) - optimized for matrix-vector, vector-vector operation – especially for \([x,y,z:w]^{\top}\)
- Native arithmetic operations with homogeneous coordinates – without exponent “normalization”
- Programmable HW – parallel processing
Implementation aspects and GPU

4D cross product can be implemented in Cg/HLSL on GPU (not optimal implementation) as:

```c
float4 cross_4D(float4 x1, float4 x2, float4 x3)
{
  float4 a;  // simple formula
  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;
}

// more compact formula available
```
Appendix

**Data processing** - main field in computer science

Data processing itself can be split to two main areas:

- **processing of textual data**
  
  limited interval of values, unlimited dimensionality
  
  [a char as one dimension - Methionylthreonylthreonylglutaminylarginyl...isoleucine 189,819 chars]
  
  No interpolation is defined

- **processing of numerical data**
  
  unlimited interval of values,
  
  limited dimensionality – usually 2 or 3
  
  Interpolation can be used

<table>
<thead>
<tr>
<th></th>
<th>Textual</th>
<th>Graphical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dim</td>
<td>∞</td>
<td>2, 3</td>
</tr>
<tr>
<td>Interval</td>
<td>0-255(ASCII)</td>
<td>(-∞, ∞)</td>
</tr>
</tbody>
</table>
Hash functions

- usually used for textual data processing
- prime numbers and modulo

Usual form $Addr = (3x + 5y + 7z) \mod \text{size}$

Multiplication int * float needed
If the hash function is constructed as

\[ \text{Addr} = \lfloor C(ax + \beta y + \gamma z) \rfloor \mod m \]

where \( a, \beta, \gamma \) are “irrational” numbers and \( m = 2^k \) better distribution is obtained

\[ => \text{much faster processing.} \]

\( C \) is a “magic” constant which maps the expression for \( x_{\text{min}}, x_{\text{max}} \) to \((0, \text{maxint})\)

\[ \text{Addr} = \lfloor C(ax + \beta y + \gamma z) \rfloor \text{and } (2^k - 1) \]

Textual processing

The has function is constructed as

\[ h(x) = \left( C \sum_{i=1}^{L} q^i x_i \right) \mod m \]

\[ q \text{ "irrational" } 0 < q < 1 \]

\[ m = 2^k - 1 \]

Both geometrical and textual hash function design have the same approach coefficients are “irrational” and no division operation is needed.

Some differences for Czech, Hebrew, English, German, Arabic, ... languages and “chemical” words.
Summary and conclusion

We have got within this course an understanding of:

- projective representation use for geometric transformations with points, lines and planes
- principle of duality and typical examples of dual problems, influence to computational complexity
- intersection computation of two planes in E3, dual Plücker coordinates and simple projective solution
- geometric problems solution with additional constrains
- intersection computations and interpolation algorithms directly in the projective space
- barycentric coordinates computation on GPU
- avoiding or postponing division operations in computations

Projective space representation supports matrix-vector architectures like GPU – faster, robust and easy to implement algorithms achieved
References


References related

Eurographics 2013

?? Questions ??

Contact: Vaclav Skala
c/o University of West Bohemia
CZ 306 14 Plzen, Czech Republic
http://www.VaclavSkala.eu
skala@kiv.zcu.cz  subj. EG 2013


INVIATION

21st WSCG conference on Computer Graphics, Visualization and Computer Vision 2013
Plzen [Pilsen] close to Prague, June 24-27, 2013

Supported by the Ministry of Education of the Czech Republic, projects No.LH12181, LG13047