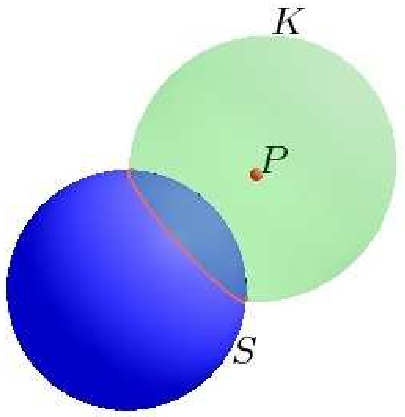
Foundations of Geometric Algebra Computing

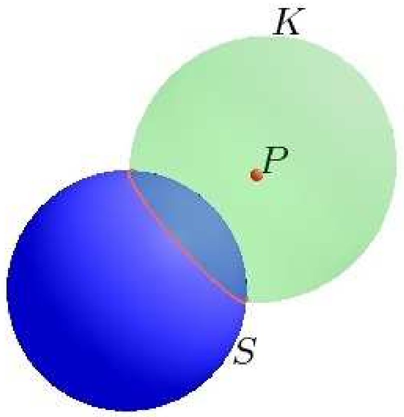
ICNAAM 2012, Kos, 21.09.2012

Dr.-Ing. Dietmar Hildenbrand
LOEWE Priority Program Cocoon
Technische Universität Darmstadt

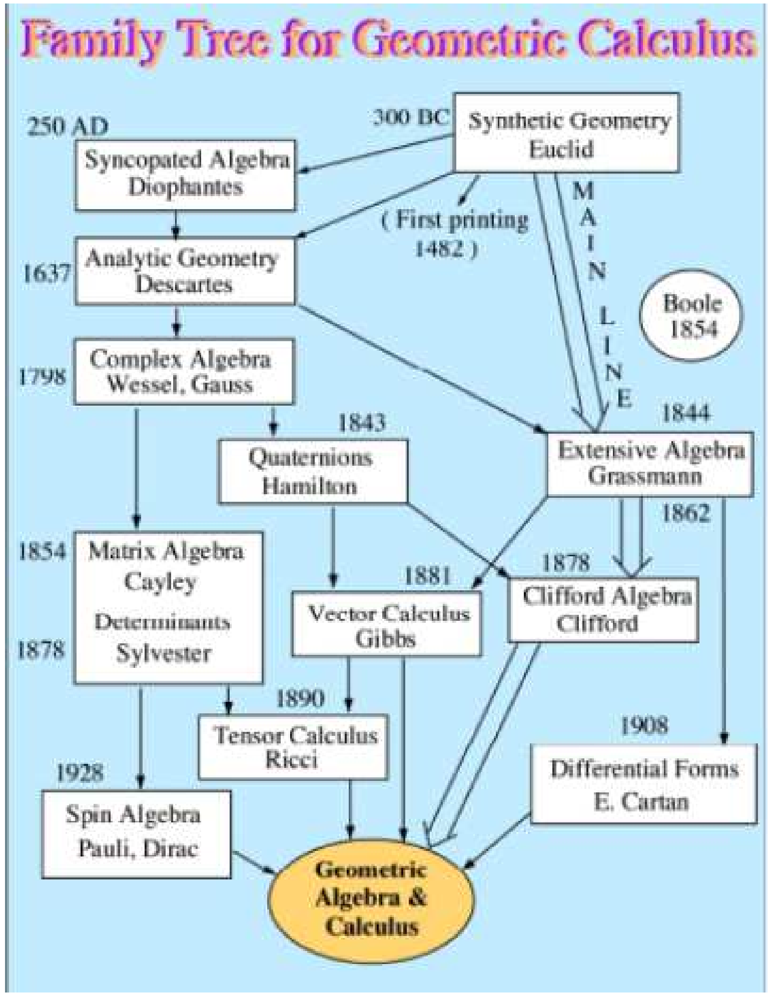


Overview

- Foundations of Geometric Algebra (GA)
- GA Applications
- One GA example
- GA Computing
  - precompiler for standard programming languages
- Molecular dynamics application

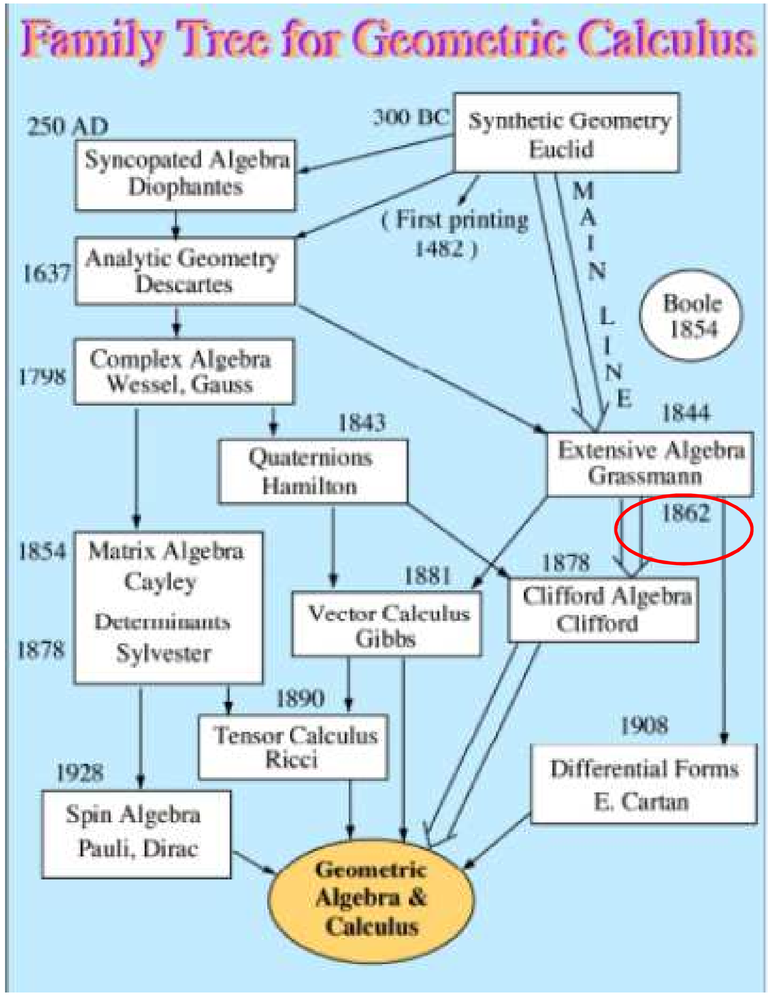


History of Geometric Algebra&Calculus



- [David Hestenes 2001]

History of Geometric Algebra&Calculus

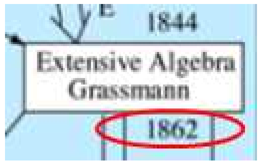


Family Tree for Geometric Calculus
300 BC
Synthetic Geometry Euclid
250 AD
Syncopated Algebra Diophantes
(First printing 1482)
MAIN LINE
1637
Analytic Geometry Descartes
Boole 1854
1798
Complex Algebra Wessel, Gauss
1843
Quaternions Hamilton
1844
Extensive Algebra Grassmann
1862
1854
Matrix Algebra Cayley
Determinants Sylvester
1878
1881
Vector Calculus Gibbs
1878
Clifford Algebra Clifford
1890
Tensor Calculus Ricci
1908
Differential Forms E. Cartan
1928
Spin Algebra Pauli, Dirac
Geometric Algebra & Calculus

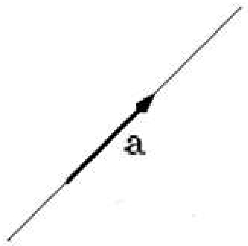
150 years

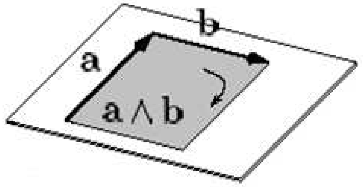
- [David Hestenes 2001]

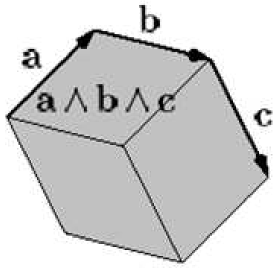
Hermann G. Grassmann



- Outer Product

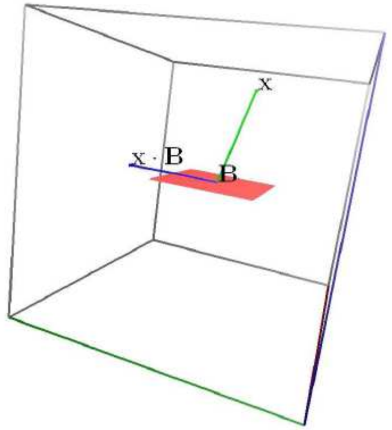




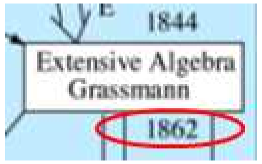


vector bivector trivector …

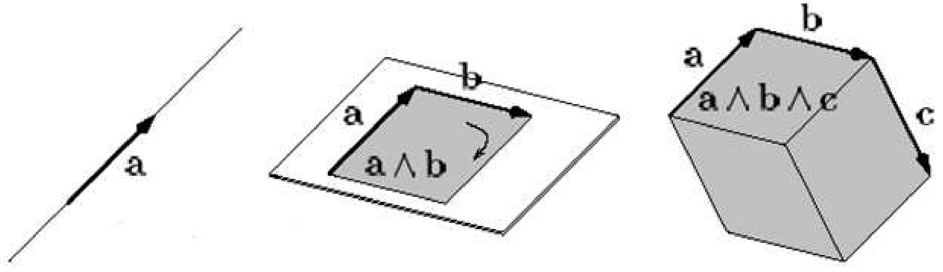
- Inner Product



Hermann G. Grassmann

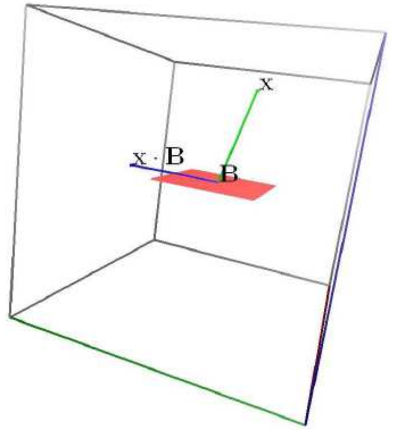


- Outer Product



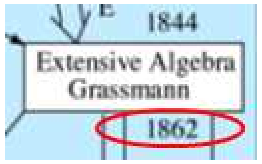
vector bivector trivector …

- Inner Product



cross product and scalar product are special cases of these general products

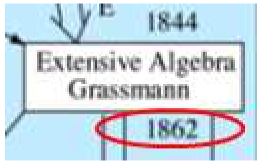
Hermann G. Grassmann



- Preface of „*Ausdehnungslehre*" (Extensive Algebra) of 1862:

"I remain completely confident that the labor I have expended on the science presented here and which has demanded a signicant part of my life, as well as the most strenuous application of my powers, will not be lost. It is true that I am aware that the form which I have given the science is imperfect and must be imperfect. But I know and feel obliged to state (though I run the risk of seeming arrogant) that even if this work should again remain unused for another seventeen years or even longer, without entering into the actual development of science, still that time will come when it will be brought forth from the dust of oblivion and when ideas now dormant will bring forth fruit."
and he went on to say:
"there will come a time when these ideas, perhaps in a new form, will arise anew and will enter into a living communication with contemporary developments."

Hermann G. Grassmann

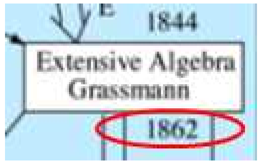


TECHNISCHE UNIVERSITÄT DARMSTADT

- Preface of „*Ausdehnungslehre*" (Extensive Algebra) of 1862:

"I remain completely confident that the labor I have expended on the science presented here and which has demanded a signicant part of my life, as well as the most strenuous application of my powers, will not be lost. It is true that I am aware that the form which I have given the science is imperfect and must be imperfect. But I know and feel obliged to state (though I run the risk of seeming arrogant) that even if this work should again remain unused for another seventeen years or even longer, without entering into the actual development of science, still that time will come when it will be brought forth from the dust of oblivion and when ideas now dormant will bring forth fruit."
and he went on to say:
"there will come a time when these ideas, perhaps in a new form, will arise anew and will enter into a living communication with contemporary developments."

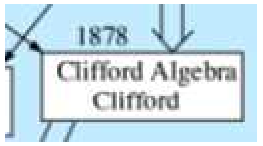
Hermann G. Grassmann



- Preface of *„Ausdehnungslehre“* (Extensive Algebra) of 1862:

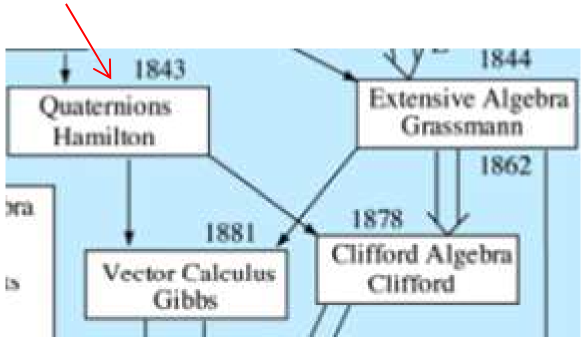
"I remain completely confident that the labor I have expended on the science presented here and which has demanded a signicant part of my life, as well as the most strenuous application of my powers, will not be lost. It is true that I am aware that the form which I have given the science is imperfect and must be imperfect. But I know and feel obliged to state (though I run the risk of seeming arrogant) that even if this work should again remain unused for another seventeen years or even longer, without entering into the actual development of science, still that time will come when it will be brought forth from the dust of oblivion and when ideas now dormant will bring forth fruit."
and he went on to say:
"there will come a time when these ideas, perhaps in a new form, will arise anew and will enter into a living communication with contemporary developments."

William K. Clifford



- Geometric Product
- For vectors $$uv = u \wedge v + u \cdot v.$$

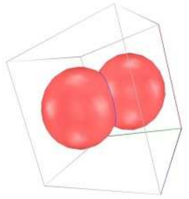
- Quaternions of Hamilton



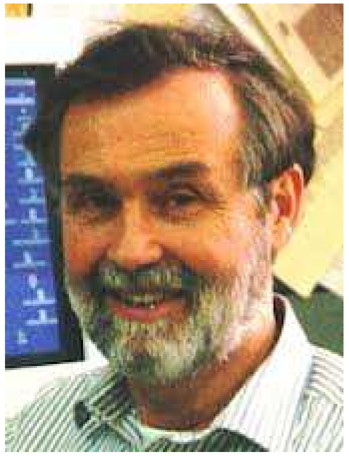
- Normally called Clifford algebra in honor of Clifford
- He called it geometric algebra

David Hestenes

- realized geometric algebra as a general language for physics („New Foundations of Classical Mechanics“, …)
- developed calculus ("Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics")
- developed the Conformal Geometric Algebra

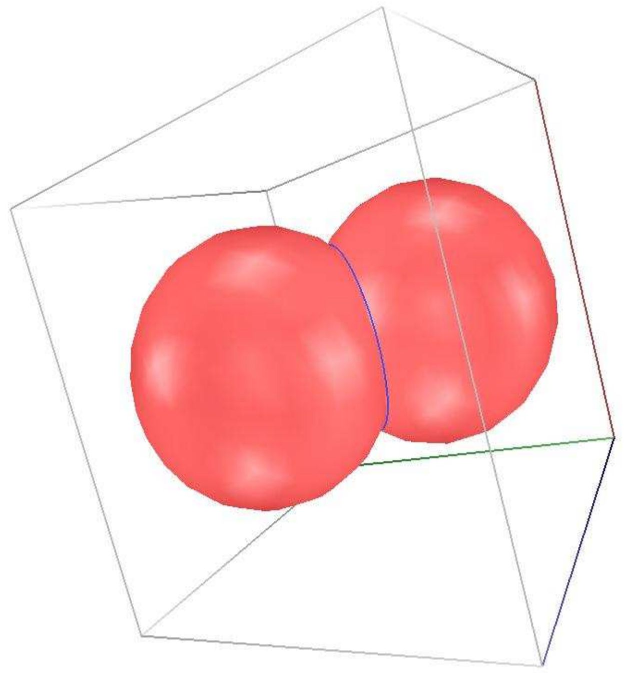


- Geometric Algebra <-> Clifford Algebra?



Goal of geometric algebra

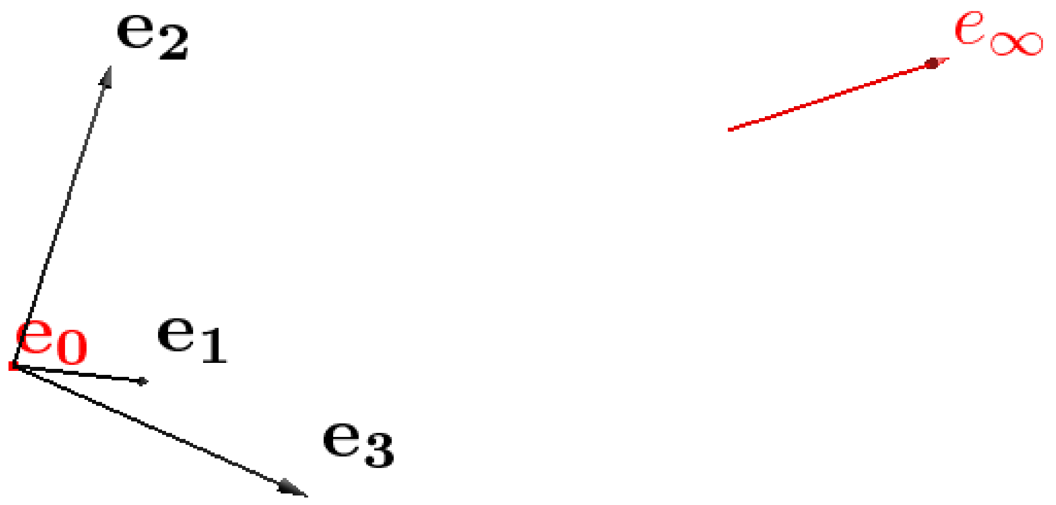
- Mathematical language close to the geometric intuition



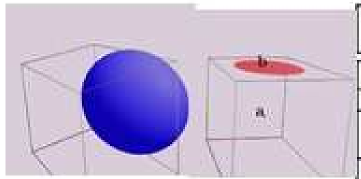
Conformal Geometric Algebra (CGA)

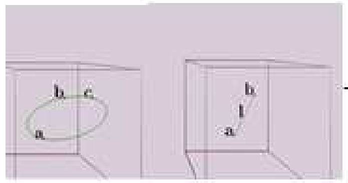
- 5 basis vectors:

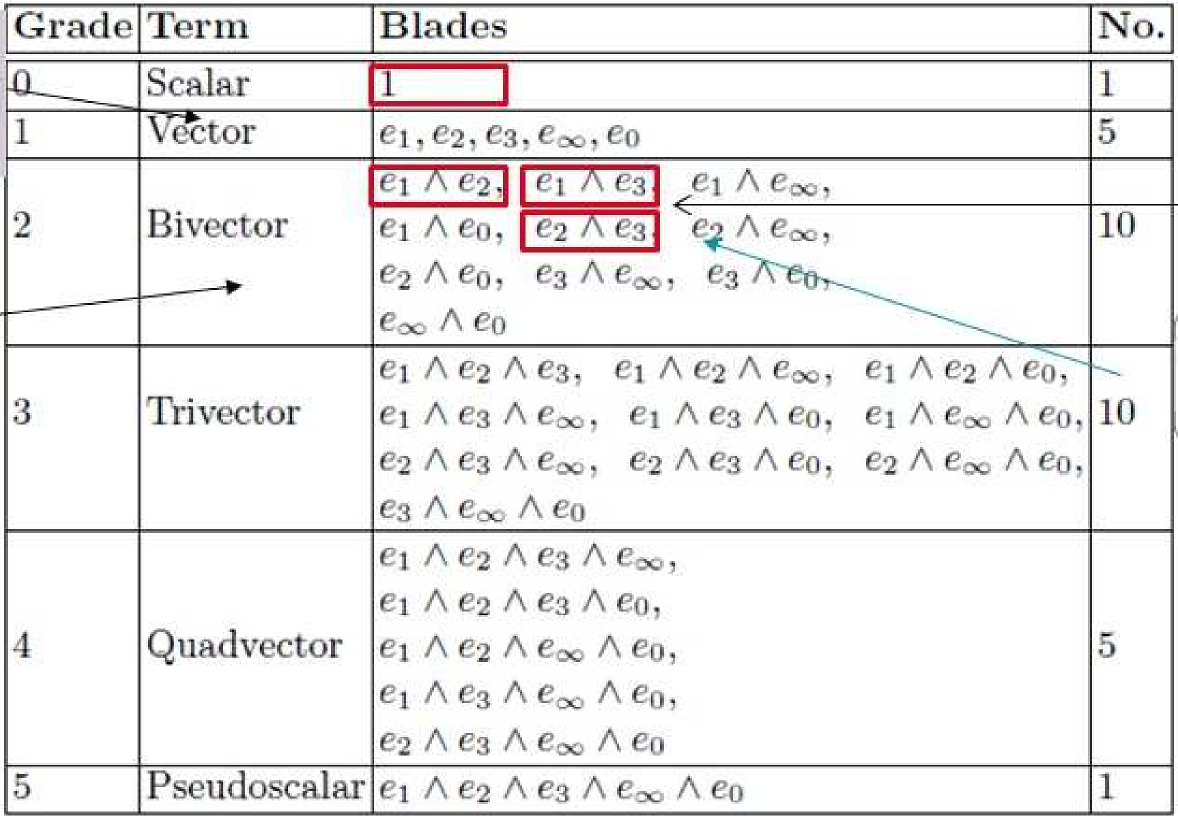
  - e_1, e_2, e_3
  - e_0 : origin
  - e_∞ : point at infinity



Basis elements of CGA







Grade	Term	Blades	No.
0	Scalar	$\boxed{1}$	1
1	Vector	$e_1, e_2, e_3, e_\infty, e_0$	5
2	Bivector	$\boxed{e_1 \wedge e_2,}$ $\boxed{e_1 \wedge e_3,}$ $e_1 \wedge e_\infty,$ $e_1 \wedge e_0,$ $\boxed{e_2 \wedge e_3,}$ $e_2 \wedge e_\infty,$ $e_2 \wedge e_0,$ $e_3 \wedge e_\infty,$ $e_3 \wedge e_0,$ $e_\infty \wedge e_0$	10
3	Trivector	$e_1 \wedge e_2 \wedge e_3,$ $e_1 \wedge e_2 \wedge e_\infty,$ $e_1 \wedge e_2 \wedge e_0,$ $e_1 \wedge e_3 \wedge e_\infty,$ $e_1 \wedge e_3 \wedge e_0,$ $e_1 \wedge e_\infty \wedge e_0,$ $e_2 \wedge e_3 \wedge e_\infty,$ $e_2 \wedge e_3 \wedge e_0,$ $e_2 \wedge e_\infty \wedge e_0,$ $e_3 \wedge e_\infty \wedge e_0$	10
4	Quadvector	$e_1 \wedge e_2 \wedge e_3 \wedge e_\infty,$ $e_1 \wedge e_2 \wedge e_3 \wedge e_0,$ $e_1 \wedge e_2 \wedge e_\infty \wedge e_0,$ $e_1 \wedge e_3 \wedge e_\infty \wedge e_0,$ $e_2 \wedge e_3 \wedge e_\infty \wedge e_0$	5
5	Pseudoscalar	$e_1 \wedge e_2 \wedge e_3 \wedge e_\infty \wedge e_0$	1

i, j, k

$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

. . .

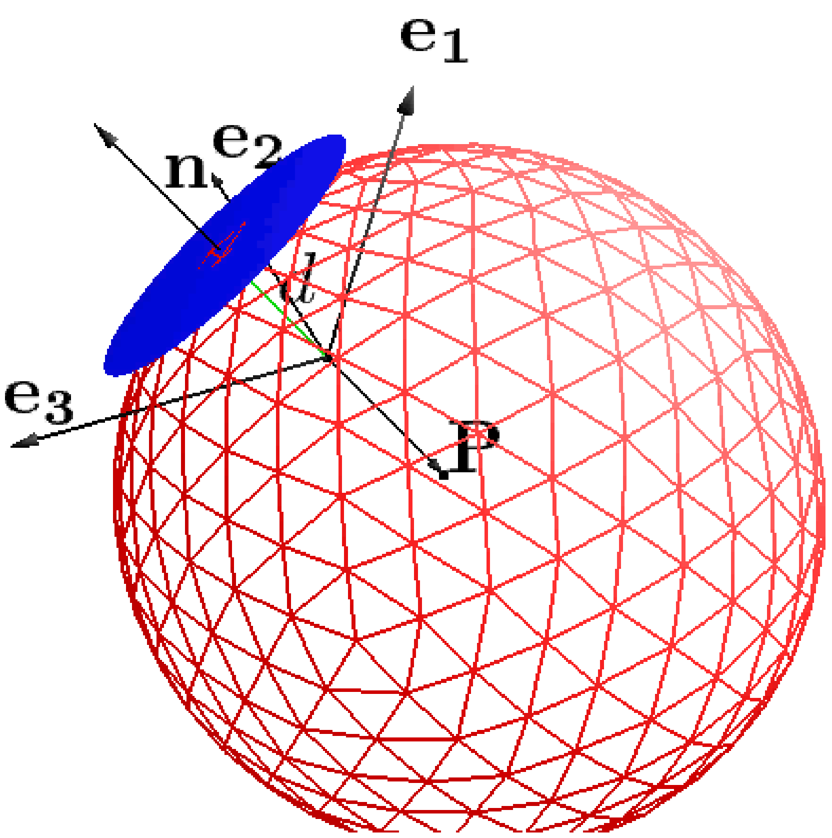
Plane and sphere as vector expression

- **sphere**

$$S = P - \frac{1}{2}r^2 e_\infty$$

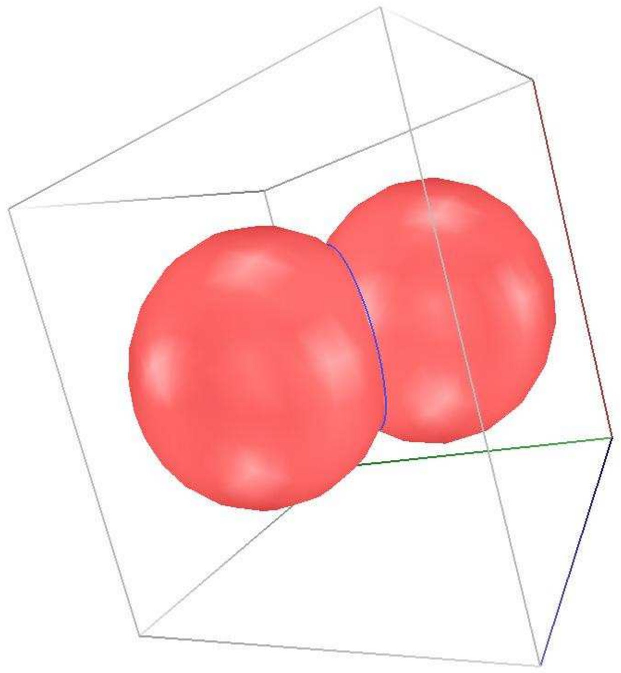
- **plane**

$$\pi = \mathbf{n} + d e_\infty$$



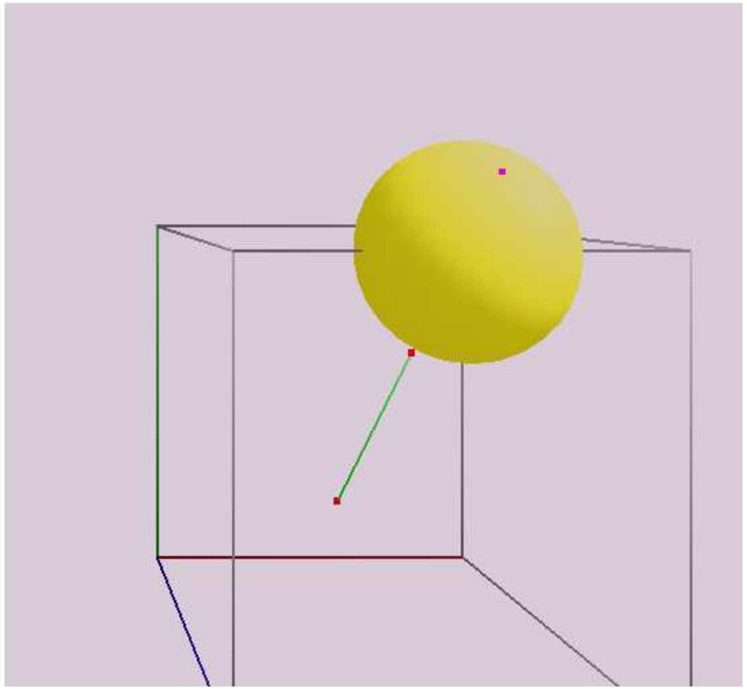
Intersection of geometric objects

- a,b are sphere
- $a \wedge b$
  - describes the intersection of the spheres
  - represents a circle



Intersection of sphere and line

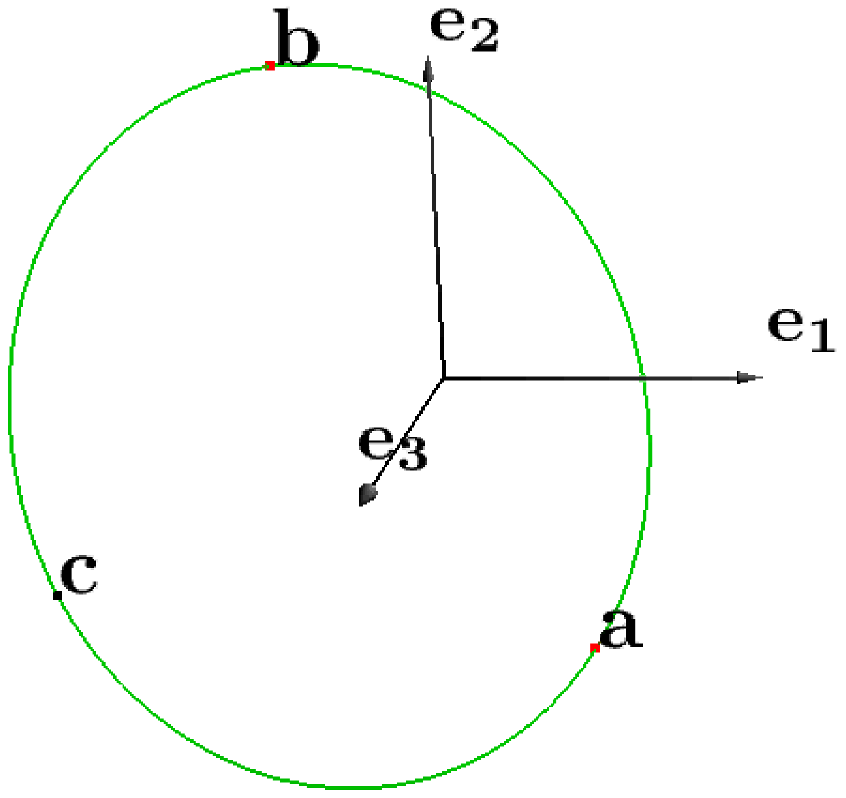
- $Pp = s_1 \wedge l$



Circle and line

- Circle (a^b^c)
- Circle -> Line

($c \longrightarrow e_\infty$)



Geometric Operations

- Rotation about L (through the origin)
- **Rotor**

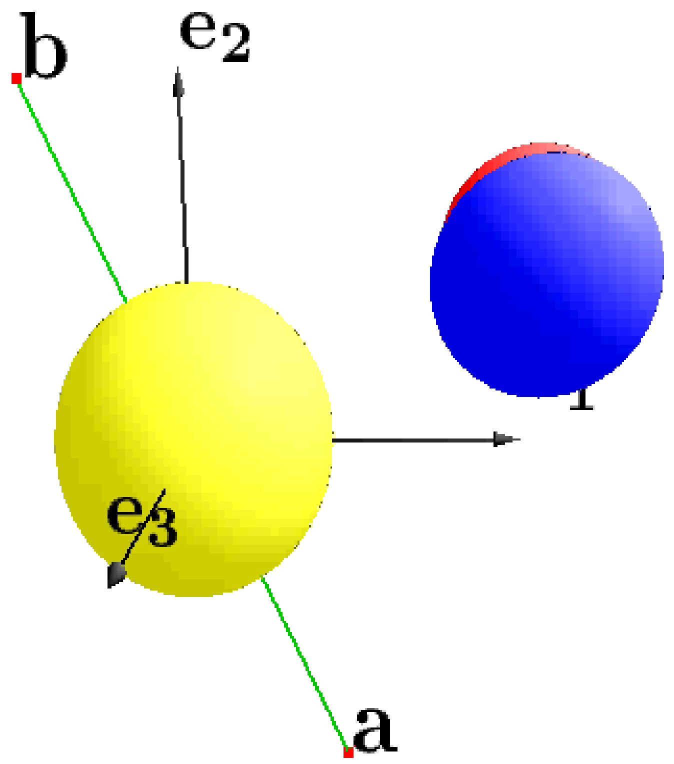
$$e^{-\frac{\phi}{2}\mathbf{l}}$$

= Quaternion

$$Q = \cos(\frac{\phi}{2}) + L\sin(\frac{\phi}{2})$$

- Note.: the line can be arbitrary

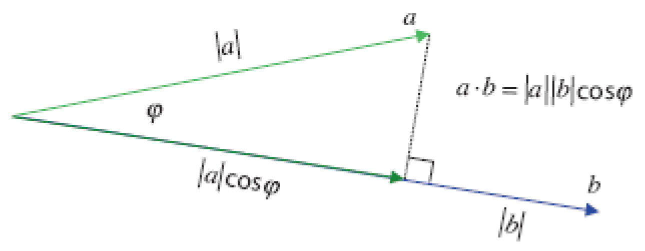
-> Rotation about an arbitrary axis(dual quaternion)



The inner product

$a \cdot b$

- a,b 3D vectors
  -> scalar product

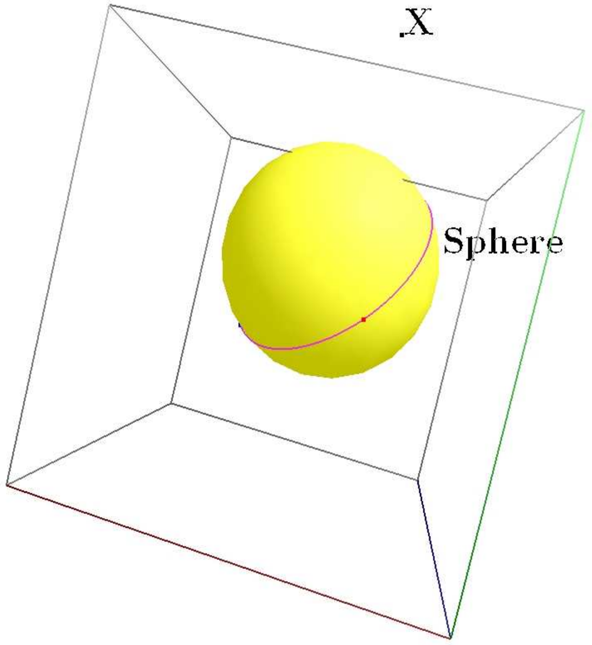


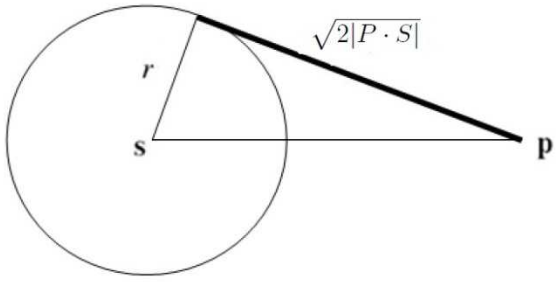
Distances

$a \cdot b$

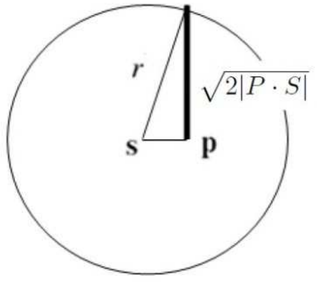
One formula for

- distance of two points
- distance between point and plane
- is a point inside or outside of a sphere? $a^2 = -2S \cdot P$





a)



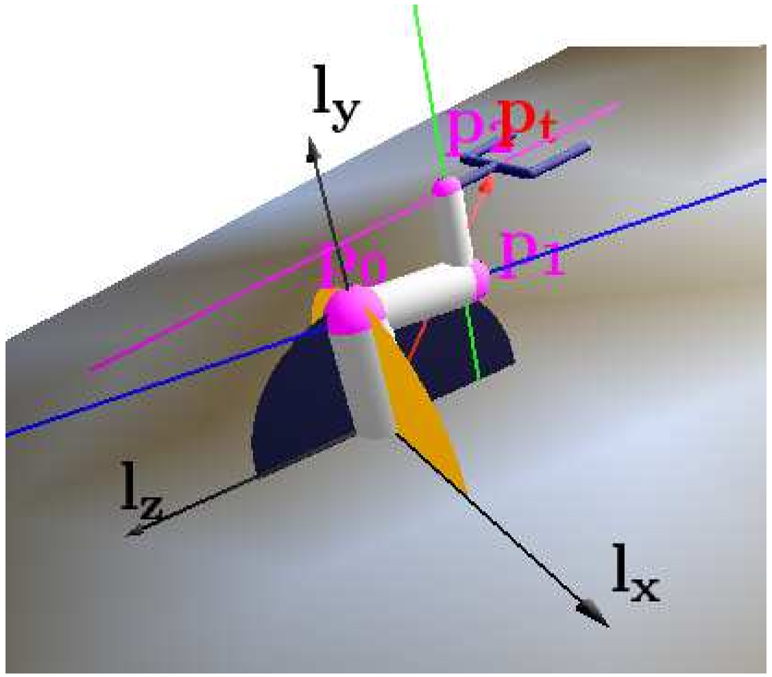
b)

Angles

$\cos(\theta) = a \cdot b$

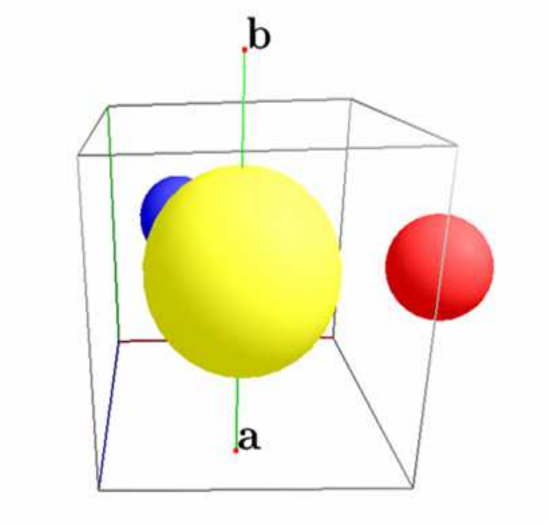
One formula for the angle between

- 2 lines
- 2 planes
- 2 circles
- …



Properties of Conformal Geometric Algebra

- Easy calculations with geometric objects and transformations
  - Geometric intuitiveness
  - Simplicity
  - Compactness
- Unification of mathematical systems
  - Complex numbers
  - Vector algebra
  - Quaternions
  - Projective geometry
  - Plücker coordinates
  - ….

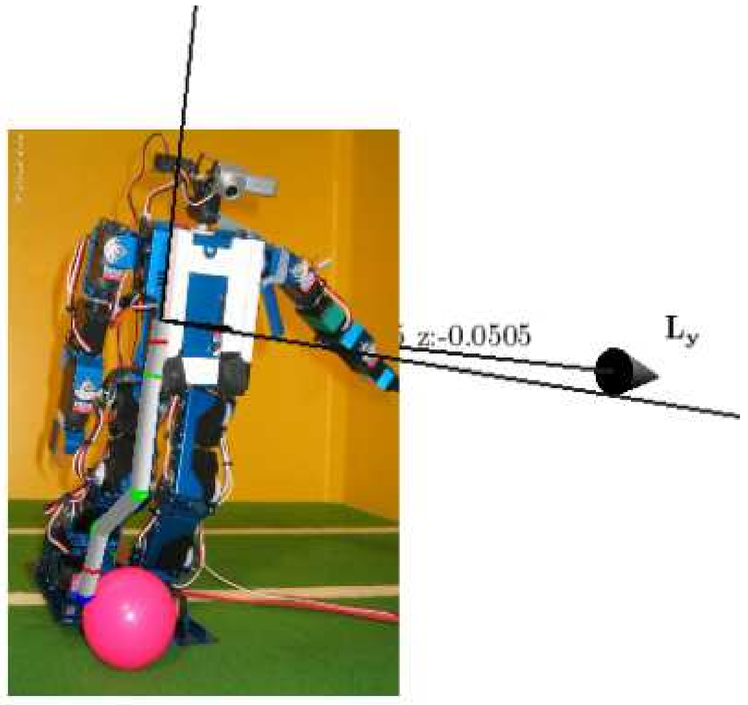


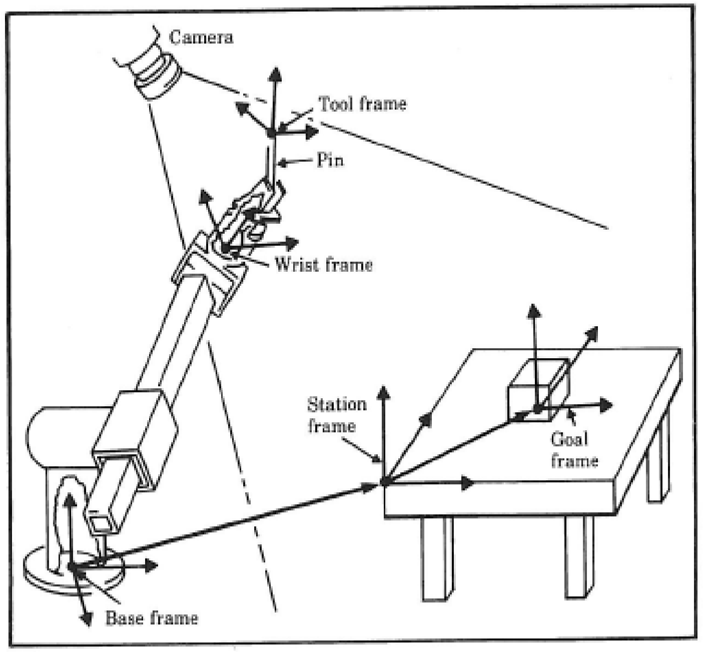
First commercial product

- since 2007:
  - Real time lighting engine Enlighten
  - Geomerics, Cambridge UK
  - Unreal Game Engine

Our Applications

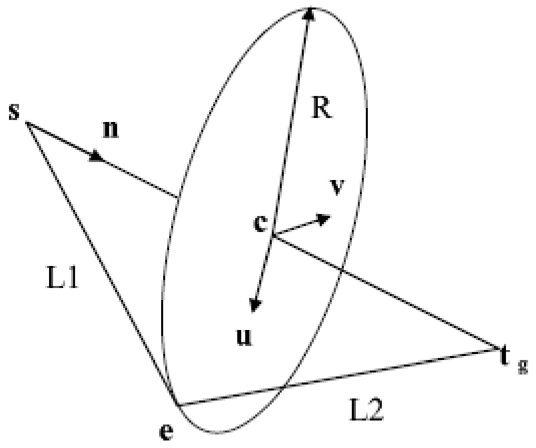
- Robot kinematics





Application

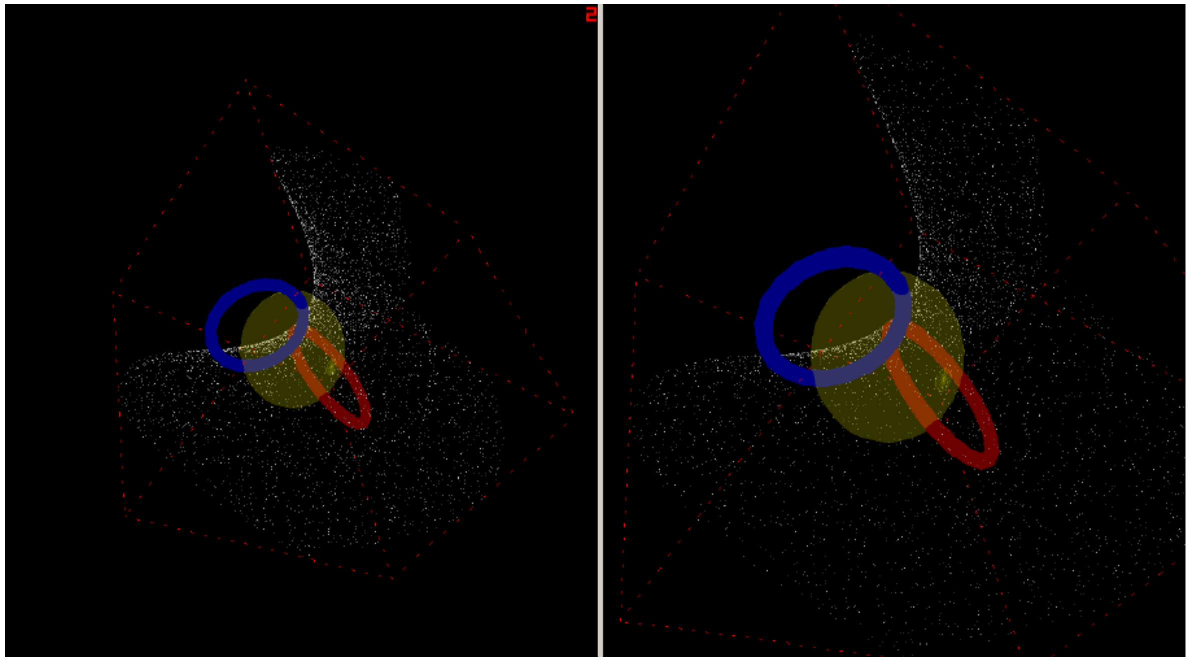
- Computer animation/
  Virtual reality



[Tolani et al. 2000]

Application

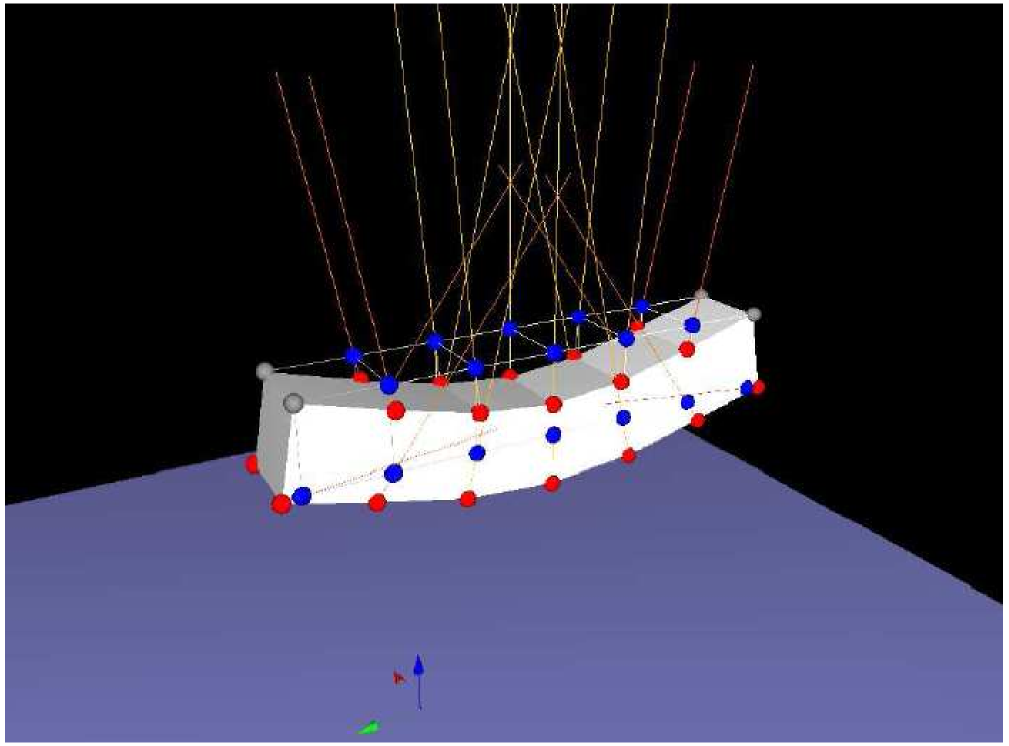
- Approximation of geometric objects to point clouds of laser scans



Application

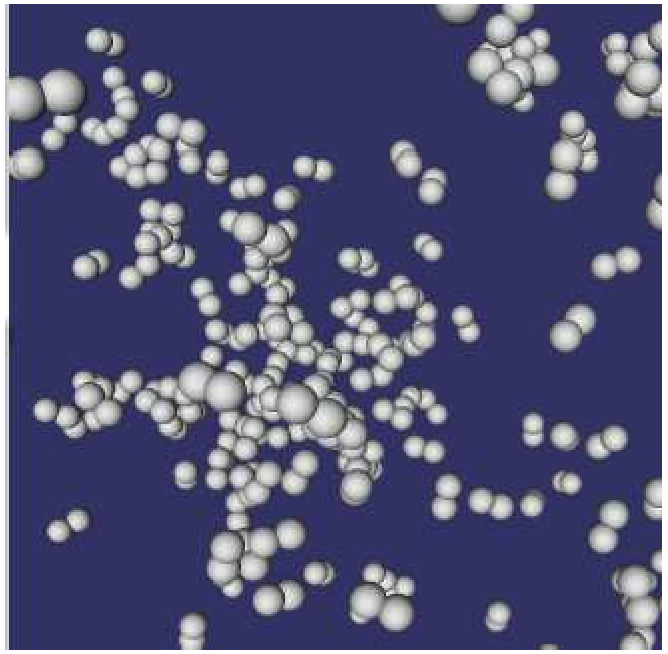
- Finite Element Solver
  - compact expressions for
    - velocities
    - forces
  - combining rotational and linear parts

  [Elmar Brendel et al]



Application

- Molecular dynamics simulation



Application within Cocoon

- EM simulation

Vector Algebra

$$\nabla \cdot \boldsymbol{E} = \rho$$
$$\nabla \times \boldsymbol{E} = -\partial_t \boldsymbol{B}$$

$$\nabla \cdot \boldsymbol{B} = 0$$
$$\nabla \times \boldsymbol{B} = \boldsymbol{J} + \partial_t \boldsymbol{E}$$

Geometric Algebra

$$\nabla \cdot \boldsymbol{E} = \rho$$
$$\nabla \wedge \boldsymbol{E} = -\partial_t (I\,\boldsymbol{B})$$

$$\nabla \cdot \boldsymbol{B} = 0$$
$$\nabla \wedge \boldsymbol{B} = I(\boldsymbol{J} + \partial_t \boldsymbol{E})$$

Replace:
scalar product with *inner product*
cross product with *outer product*

Geometric Algebra

$$\nabla\, \boldsymbol{E} = \rho - \partial_t (I\,\boldsymbol{B})$$

$$\nabla\, \boldsymbol{B} = 0 + I(\boldsymbol{J} + \partial_t \boldsymbol{E})$$
$$\Leftrightarrow$$
$$\nabla (I\,\boldsymbol{B}) = -(\boldsymbol{J} + \partial_t \boldsymbol{E})$$

Geometric Product:
Combine inner and outer product

$$\boldsymbol{a}\,\boldsymbol{b} = \boldsymbol{a} \cdot \boldsymbol{b} + \boldsymbol{a} \wedge \boldsymbol{b}$$

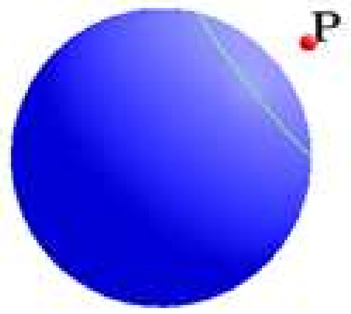
Finally combine them to a **single multivector**:

$$\boxed{\nabla (\boldsymbol{E} + I\,\boldsymbol{B}) + \partial_t (\boldsymbol{E} + I\,\boldsymbol{B}) = \rho - \boldsymbol{J}}$$

- One formula for Maxwell equations

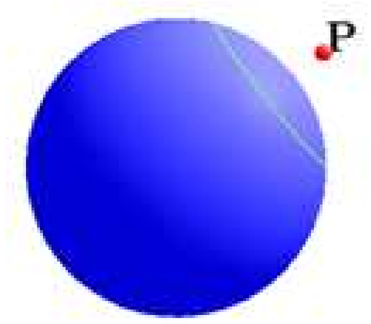
Horizon Example

- Compute the horizon circle with
  - observer *P*
  - Earth *S*



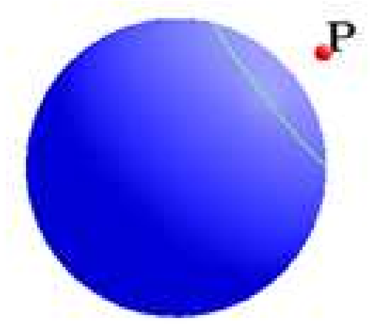
Warum Horizont-Aufgabe?

- Nachweis von
  - Nähe der algebraischen Beschreibung zur geometrischen Vorstellung
  - Einfachheit/Kompaktheit der Algorithmen



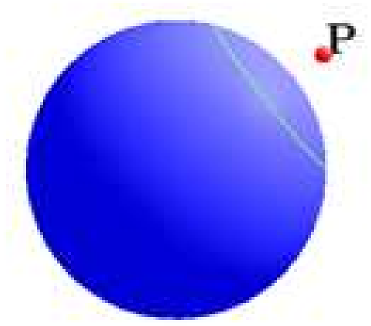
The Solution

- *P*, *S*, … *C*?



The Solution

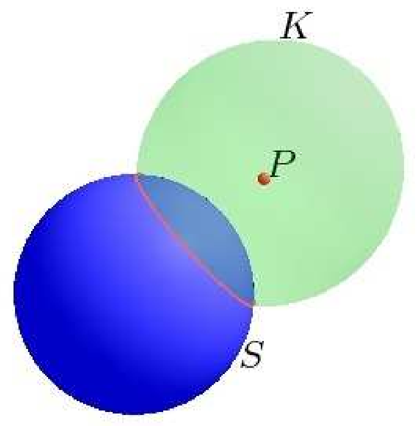
- Which sphere intersects with the sphere *S* at the horizon circle?



The Solution

- Which sphere intersects with the sphere *S* at the horizon circle?

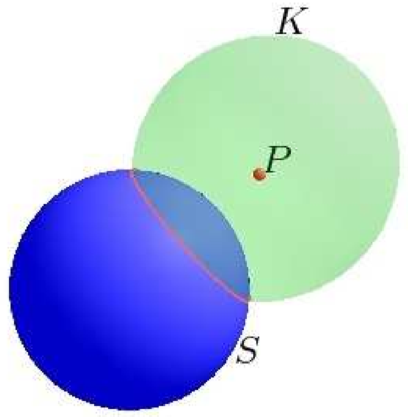
  Sphere *K* with center *P* and radius such that it touches *S*



The solution

- Which sphere intersects with the sphere *S* at the horizon circle?
  Sphere *K* with center *P* and radius such that it touches *S*

- The radius of *K*?



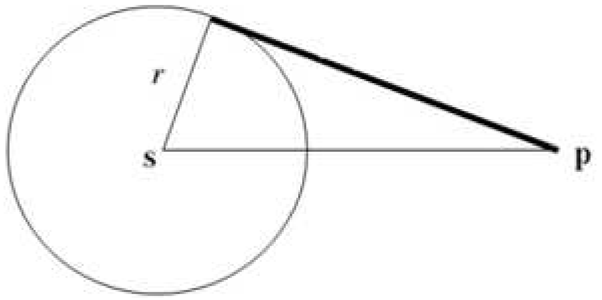
The solution

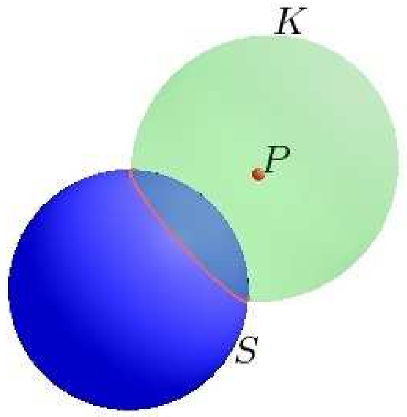
- Which sphere intersects with the sphere *S* at the horizon circle?

  Sphere *K* with center *P* and radius such that it touches *S*

- The radius a of *K*?

$$a^2 = -2S \cdot P$$





The solution

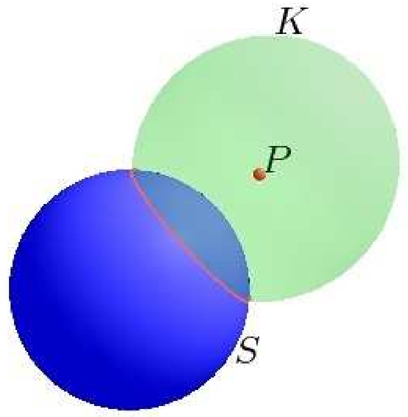
- Which sphere intersects with the sphere *S* at the horizon circle?

  Sphere *K* with center *P* and radius such that it touches *S*

- The radius a of *K*?

  $$a^2 = -2S \cdot P$$

- How to describe the sphere *K*?



The solution

- Which sphere intersects with the sphere *S* at the horizon circle?

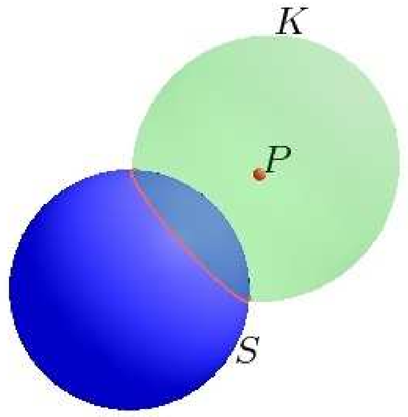
  Sphere *K* with center *P* and radius such that it touches *S*

- The radius a of *K*?

$$a^2 = -2S \cdot P$$

- How to describe the sphere *K*?

$$K = P - \frac{1}{2}a^2 e_\infty = P + (S \cdot P)e_\infty$$



The solution

- Which sphere intersects with the sphere *S* at the horizon circle?

  Sphere *K* with center *P* and radius such that it touches *S*

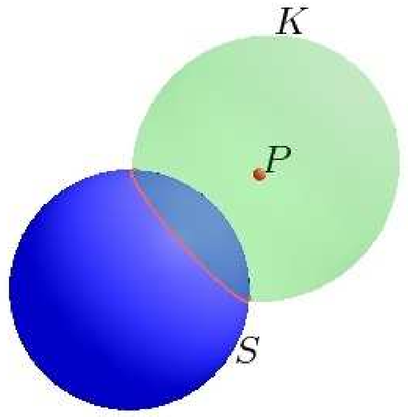
- The radius a of *K*?

$$a^2 = -2S \cdot P$$

- How to describe the sphere *K*?

$$K = P - \frac{1}{2}a^2 e_\infty = P + (S \cdot P)e_\infty$$

- The formula for the horizon circle *C*?



The solution

- Which sphere intersects with the sphere *S* at the horizon circle?

  Sphere *K* with center *P* and radius such that it touches *S*

- The radius a of *K*?

$$a^2 = -2S \cdot P$$

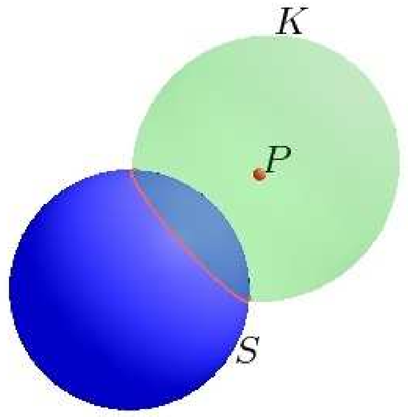
- How to describe the sphere *K*?

$$K = P - \frac{1}{2}a^2 e_\infty = P + (S \cdot P)e_\infty$$

- The formula for the horizon circle *C*?

$$C = S \wedge K$$

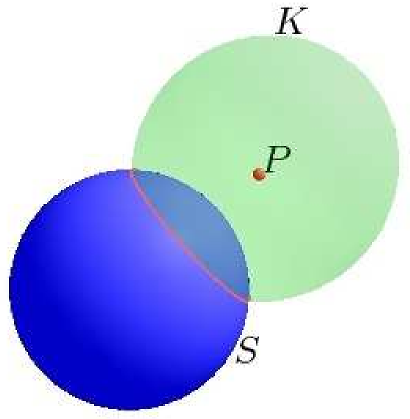
$$C = S \wedge (P + (S \cdot P)e_\infty).$$



Properties of Geometric Algebra

- geometrically intuitive?
- simple?
- compact?
- role of coordinates?

$$C = S \wedge (P + (S \cdot P)e_\infty).$$



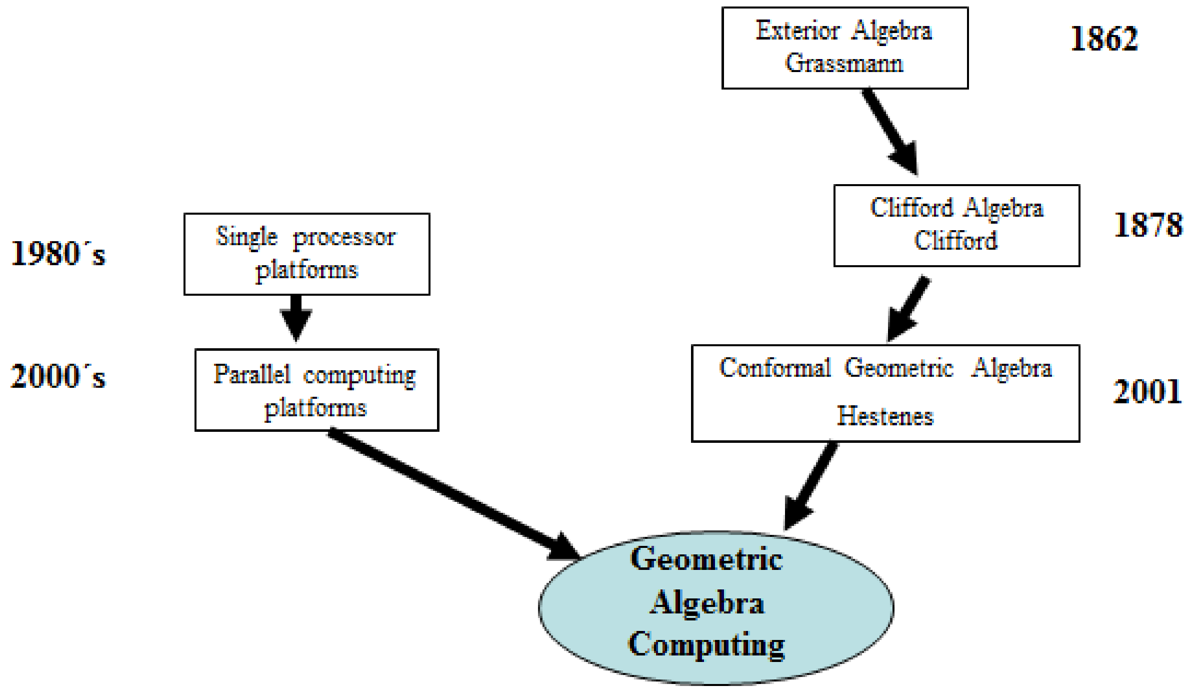
How to support the widespread use of geometric algebra?

- What form of geometric algebra do we need today?

"there will come a time when these ideas, perhaps in a new form, will arise anew and will enter into a living communication with contemporary developments.“ [Grassmann 1862]

- Making it accessible for as many people as possible and their applications
- -> provide a suitable computing technology

Geometric Algebra Computing



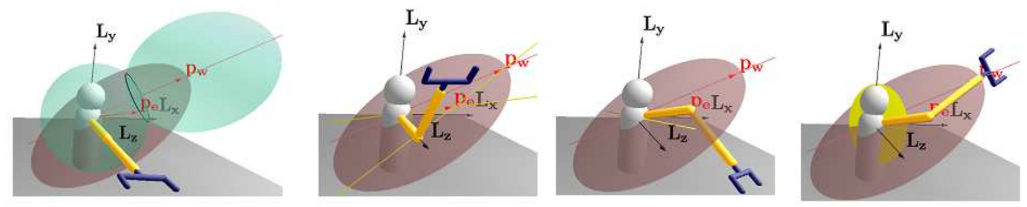
Exterior Algebra Grassmann
1862
Clifford Algebra Clifford
1878
1980´s
Single processor platforms
2000´s
Parallel computing platforms
Conformal Geometric Algebra Hestenes
2001
Geometric Algebra Computing

Goals of Geometric Algebra Computing

- **G**eometric **a**lgebra **al**gorithms ***op***timizer (www.gaalop.de)

- Easy development process
  - integration in standard programming languages
  - Intuitive and compact descriptions based on geometric algebra
  - leading to
    - reduction in development time
    - better maintainability
- High performance and robustness of the implementation

Proof-of-Concept Application



- Visual and interactive development
- Compact algorithms (about 30%)
- First implementation:
  - 50 times slower than conventional implementation
- Eurographics 2006:
  - Paper " Competitive runtime performance for inverse kinematics algorithms using conformal geometric algebra " by Dietmar Hildenbrand, Daniel Fontijne, Yusheng Wang, Marc Alexa and Leo Dorst
  - First geometric algebra implementation that was faster than the conventional implementation

Geometric Algebra Computing Architecture

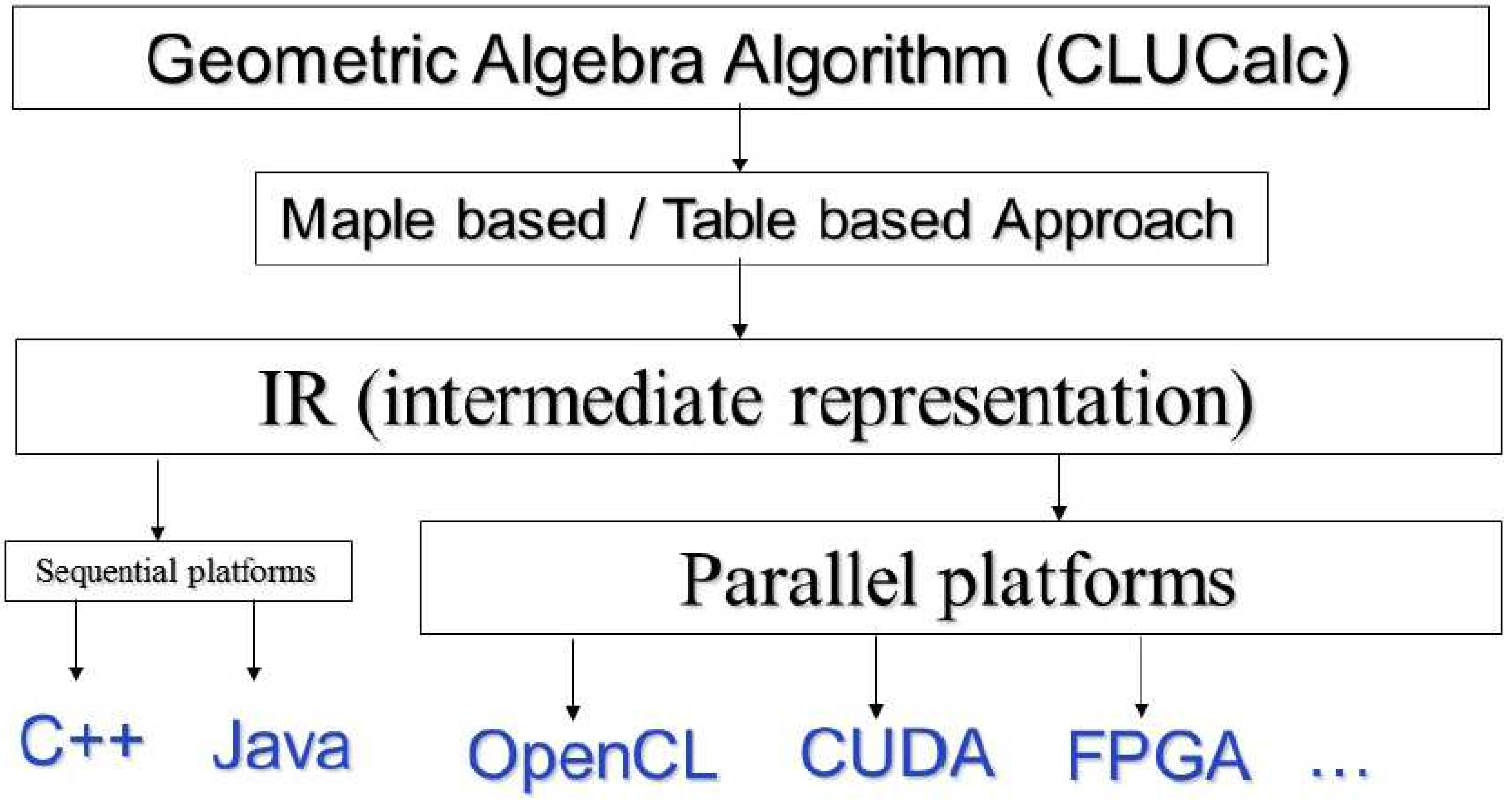

Geometric Algebra Algorithm (CLUCalc)
Maple based / Table based Approach
IR (intermediate representation)
Sequential platforms
Parallel platforms
C++
Java
OpenCL
CUDA
FPGA
...

Table based Compilation Example

- Geometric product multiplication table in 3D GA:

		b		**b**$_1$	**b**$_2$	**b**$_3$				
			E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
a			1	e_1	e_2	e_3	e_{12}	e_{23}	e_{13}	e_{123}
	E_1	1	0	0	0	0	0	0	0	0
a$_1$	E_2	e_1	0	E_1	E_5	E_7	0	0	0	0
a$_2$	E_3	e_2	0	$-E_5$	E_1	E_6	0	0	0	0
a$_3$	E_4	e_3	0	$-E_7$	$-E_6$	E_1	0	0	0	0
	E_5	e_{12}	0	0	0	0	0	0	0	0
	E_6	e_{23}	0	0	0	0	0	0	0	0
	E_7	e_{13}	0	0	0	0	0	0	0	0
	E_8	e_{123}	0	0	0	0	0	0	0	0

- GA algorithm:

```
a=a1*e1+a2*e2+a3*e3;
b=b1*e1+b2*e2+b3*e3;
?c=a*b;
```

$\longrightarrow$

resulting C code:

```
c[1]=a1*b1+a2*b2+a3*b3;
c[5]=a1*b2-a2*b1;
c[6]=a2*b3-a3*b2;
c[7]=a1*b3-a3*b1;
```

Table based Compilation Example

- Geometric product multiplication table in 3D GA:

		b		$\mathbf{b_1}$	$\mathbf{b_2}$	$\mathbf{b_3}$				
			E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
a			1	e_1	e_2	e_3	e_{12}	e_{23}	e_{13}	e_{123}
	E_1	1	0	0	0	0	0	0	0	0
$\mathbf{a_1}$	E_2	e_1	0	E_1	E_5	E_7	0	0	0	0
$\mathbf{a_2}$	E_3	e_2	0	$-E_5$	E_1	E_6	0	0	0	0
$\mathbf{a_3}$	E_4	e_3	0	$-E_7$	$-E_6$	E_1	0	0	0	0
	E_5	e_{12}	0	0	0	0	0	0	0	0
	E_6	e_{23}	0	0	0	0	0	0	0	0
	E_7	e_{13}	0	0	0	0	0	0	0	0
	E_8	e_{123}	0	0	0	0	0	0	0	0

- GA algorithm:

```
a=a1*e1+a2*e2+a3*e3;
b=b1*e1+b2*e2+b3*e3;
?c=a*b;
```

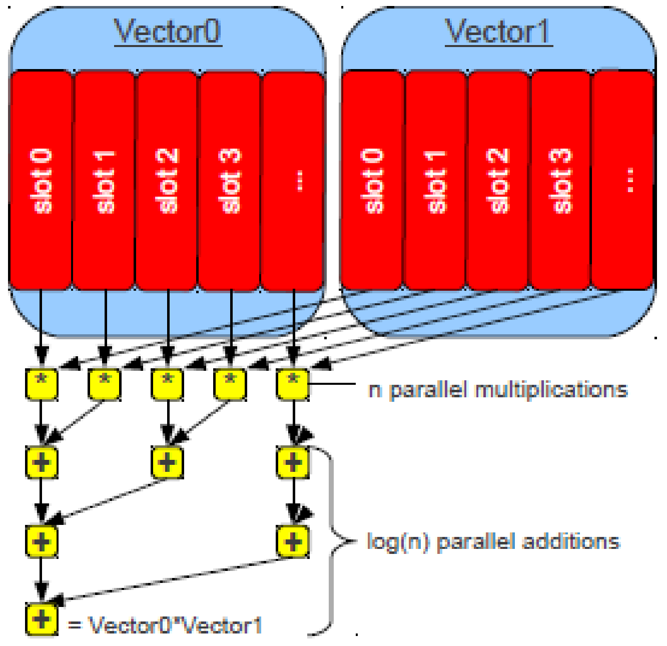
$\longrightarrow$

resulting C code:

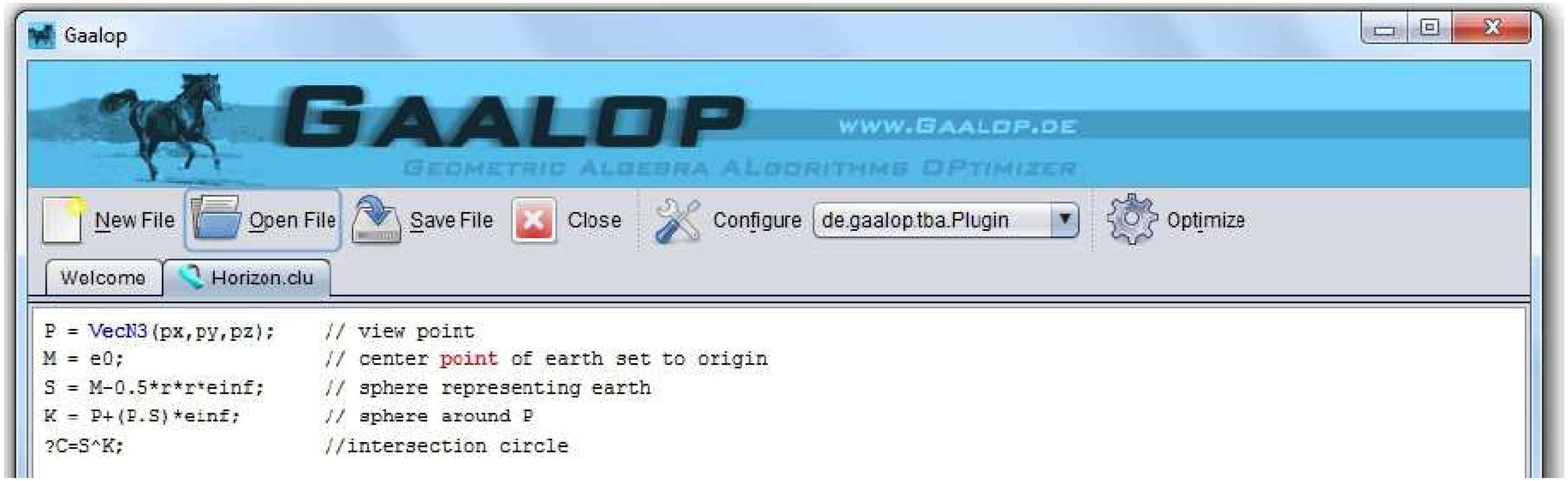
```
c[1]=a1*b1+a2*b2+a3*b3;
c[5]=a1*b2-a2*b1;
c[6]=a2*b3-a3*b2;
c[7]=a1*b3-a3*b1;
```

Advantage

- Sum of products -> SIMD



Horizon Example in Gaalop



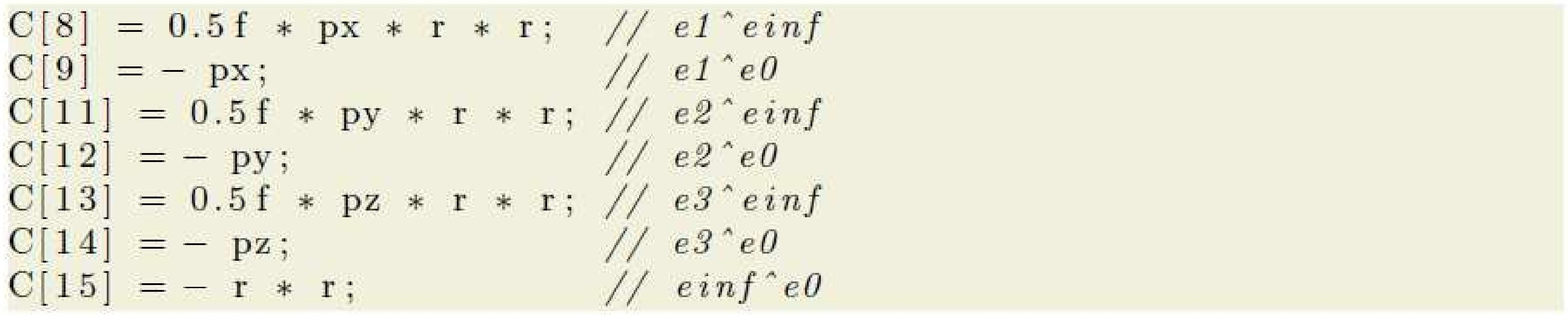
```
P = VecN3(px,py,pz);     // view point
M = e0;                  // center point of earth set to origin
S = M-0.5*r*r*einf;      // sphere representing earth
K = P+(P.S)*einf;        // sphere around P
?C=S^K;                  //intersection circle
```

- based on the CLUCalc language (www.clucalc.info)

```
P = VecN3(px,py,pz);        // view point
M = e0;                     // center point of earth set to origin
S = M-0.5*r*r*einf;         // sphere representing earth
K = P+(P.S)*einf;           // sphere around P
?C=S^K;                     // intersection circle
```

Horizon Example in Gaalop

- 32-dim multivector array with optimized coefficient computations



```
C[8]  = 0.5f * px * r * r;  // e1^einf
C[9]  = - px;               // e1^e0
C[11] = 0.5f * py * r * r;  // e2^einf
C[12] = - py;               // e2^e0
C[13] = 0.5f * pz * r * r;  // e3^einf
C[14] = - pz;               // e3^e0
C[15] = - r * r;            // einf^e0
```

- … to be further optimized from the storage point of view

Horizon Example in Gaalop

- 32-dim multivector array with optimized coefficient computations

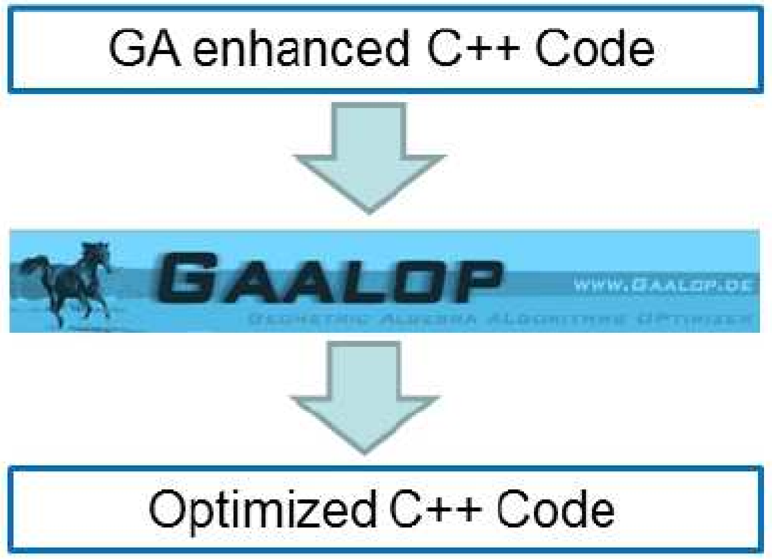
```
C[8]  = 0.5f * px * r * r;  // e1^einf
C[9]  = - px;               // e1^e0
C[11] = 0.5f * py * r * r;  // e2^einf
C[12] = - py;               // e2^e0
C[13] = 0.5f * pz * r * r;  // e3^einf
C[14] = - pz;               // e3^e0
C[15] = - r * r;            // einf^e0
```

- … to be further optimized from the storage point of view

This kind of very easy arithmetic expressions lead to high runtime performance and robustness

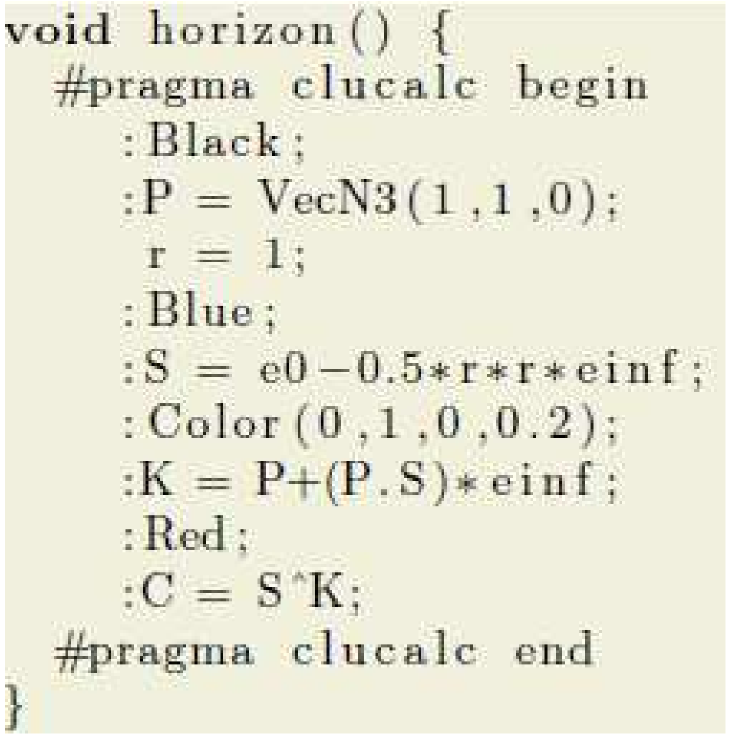
Gaalop Precompiler

```
#pragma gpc begin
  ...
  Import of multivectors (if needed)
  ...
  #pragma clucalc begin
    ...
    Geometric Algebra code based on CLUCalc
    ...
  #pragma clucalc end
  ...
  Export of multivectors (if needed)
  ...
#pragma gpc end
```

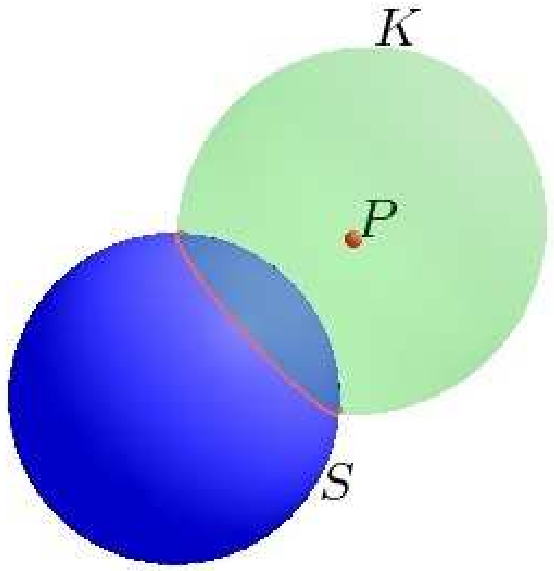


Gaalop Precompiler

- Visualization of the horizon example



```
void horizon() {
  #pragma clucalc begin
     :Black;
     :P = VecN3(1,1,0);
      r = 1;
     :Blue;
     :S = e0-0.5*r*r*einf;
     :Color(0,1,0,0.2);
     :K = P+(P.S)*einf;
     :Red;
     :C = S^K;
  #pragma clucalc end
}
```



Gaalop Precompiler

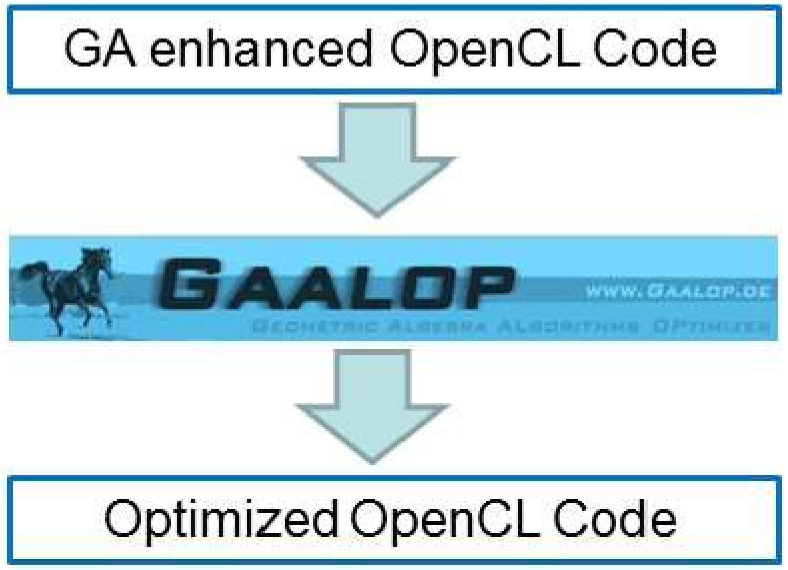
- Horizon computations in C++

```
#include <iostream>
#include <gp.h>

int main() {
  #pragma gpc begin
    #pragma clucalc begin
      P = VecN3(1,1,0);
      r = 1;
      S = e0-0.5*r*r*einf;
      C = S^(P+(P.S)*einf);
      ?homogeneousCenter = C*einf*C;
      ?scale = -homogeneousCenter.einf;
      ?EuclideanCenter = homogeneousCenter / scale;
    #pragma clucalc end
    std::cout << mv_get_bladecoeff(homogeneousCenter,e0) << std::endl;
    std::cout << mv_get_bladecoeff(EuclideanCenter,e0) << std::endl;
    std::cout << mv_get_bladecoeff(EuclideanCenter,e1)
    << "," << mv_get_bladecoeff(EuclideanCenter,e2)
    << "," << mv_get_bladecoeff(EuclideanCenter,e3);
  #pragma gpc end
  return 0;
}
```

Gaalop Precompiler

- Gaalop GPC for OpenCL



Gaalop Precompiler

- Horizon example
- in OpenCL

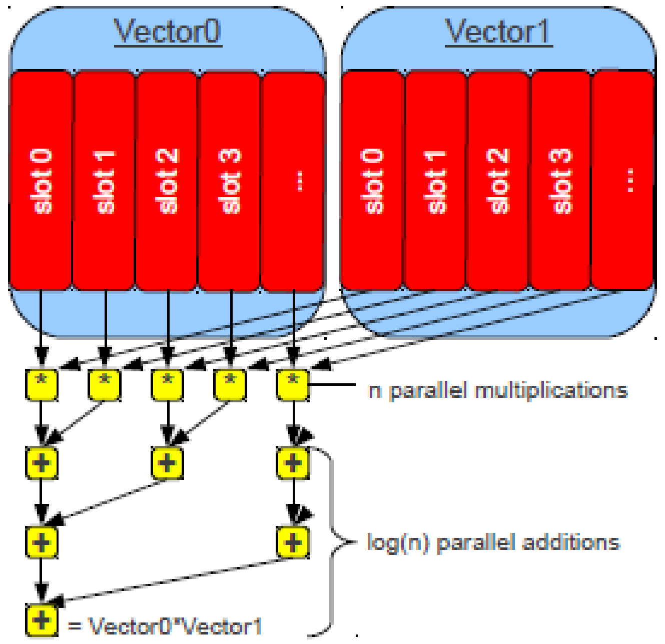
```
__kernel void horizonKernel(__global
float* circleCenters, __global const float* points,
const unsigned int num_points)
{
  const int id = get_global_id(0);
  #pragma gpc begin
    P = VecN3(points[id],
              points[id+num_points],
              points[id+2*num_points]);
    #pragma clucalc begin
      r = 1;
      S = e0-0.5*r*r*einf;
      C = S^(P+(P.S)*einf);

      ?homogeneousCenter = C*einf*C;
      ?scale = -homogeneousCenter.einf;
      ?EuclideanCenter = homogeneousCenter / scale;
    #pragma clucalc end
    circleCenters = mv_to_stridedarray(EuclideanCenter,
                                        id, num_points, e1,e2,e3);
  #pragma gpc end

}
```

Advantages for GPUs

- Sum of products -> SIMD



- Few special cases

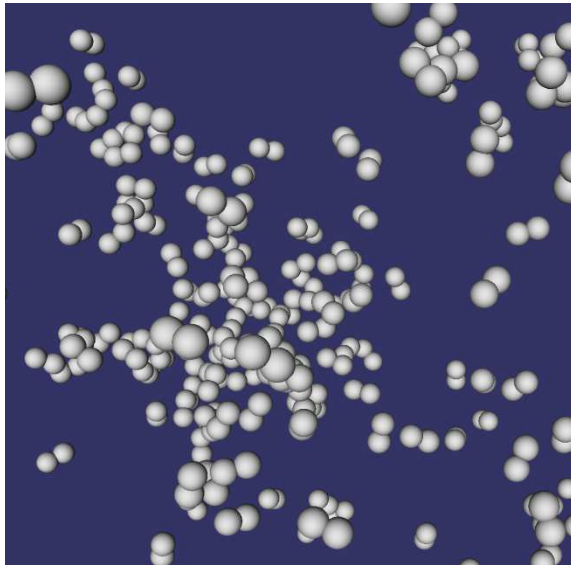
List of Precompiler Functions

coeff = mv_getbladecoeff(mv,blade);	Get the coefficient of blade **blade** of multivector **mv**.
array = mv_to_array(mv, blades,...);	Write the blades **blades ,...** of multivector **mv** to array **array**. Example **array = mv_to_array (mv,e1,e2,e3,e0,einf);**.
array = mv_to_stridedarray(mv,index,stride,blades ,...);	Write the blades **blades ,...** of multivector **mv** to array **array** with stride **stride**. Example **array = mv_to_array (mv,nummvs, e1,e2,e3,e0,einf);**.
vector = mv_to_vector(mv, blades,...);	Write the multivector **mv** to vector **vector**.
mv = mv_from_vector(vector,blades,..);	Construct multivector **mv** from vector **vector**.
mv = mv_from_array(array,blades,..);	Construct multivector **mv** from array **array**
mv = mv_from_stridedarray(array,index,stride,blades ,...);	Construct multivector **mv** from array **array** with stride **stride**.

Gaalop GPC for OpenCL …

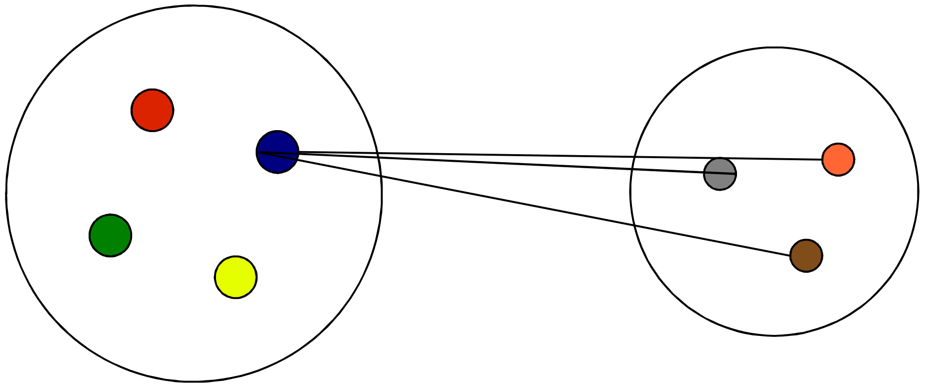
- … available for Windows and Linux (www.gaalop.de).

- Application example: Molecular dynamics …



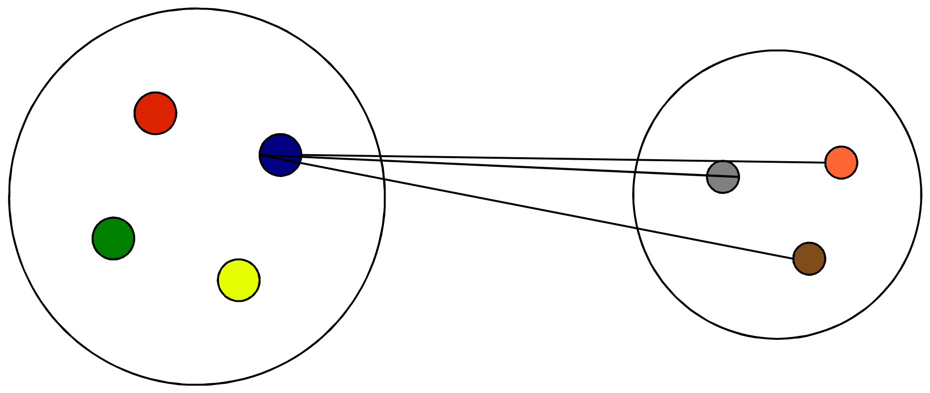
Molecular Dynamics in a Nutshell

- A Molecule is a compound of several atoms,
- which are assumed to be static inside the molecule.
- Every atom sends out attraction or repulsion forces to every other atom.
- These forces then result in a movement of the molecules according to Newton's and Euler's laws.
- This is simulated for 1000s of molecules in parallel.



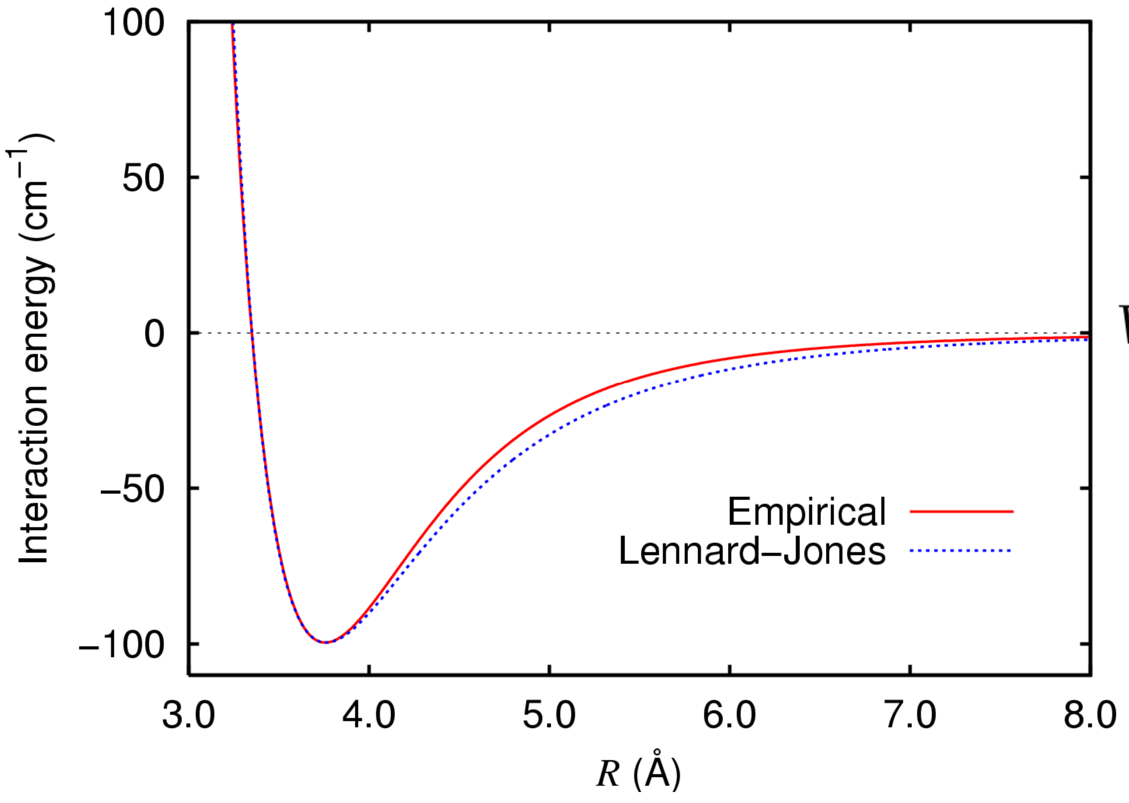
Molecular Dynamics in a Nutshell

- A Molecule is a compound of several atoms,
- which are assumed to be static inside the molecule.
- Every atom sends out attraction or repulsion forces to every other atom.
- These forces then result in a movement of the molecules according to Newton's and Euler's laws.
- This is simulated for 1000s of molecules in parallel.



Molecular dynamics simulation

- Lennard Jones potential



www.wikipedia.de

$$V(\mathbf{r}) = 4\epsilon\left[\left(\frac{\sigma}{\mathbf{r}}\right)^{12} - \left(\frac{\sigma}{\mathbf{r}}\right)^{6}\right]$$

$$\mathbf{r} = \mathbf{r}_i - \mathbf{r}_j \in \mathbb{R}^3$$

Molecular dynamics simulation

- Lennard Jones force
  - is the negative gradient of the Lennard Jones potential

$$F(\mathbf{r}) = -\nabla_{\mathbf{r}} V$$

  - and results in the following force vector acting upon a pair of atoms

$$F(\mathbf{r}) = 24\epsilon \frac{\mathbf{r}}{|\mathbf{r}|} \left[2\frac{\sigma^{12}}{|\mathbf{r}|^{13}} - \frac{\sigma^{6}}{|\mathbf{r}|^{7}} \right]$$

  - All forces (and torques) acting on a particular molecule can be summed up to a single net force.
  - With Newton's law *F=m*a* this force results in an accelaration of the molecule.

Conventional molecular dynamics solver

- Conventional Velocity Verlet timestep integration method

1. Calculate: $\vec{x}(t+\Delta t) = \vec{x}(t) + \vec{v}(t)\,\Delta t + \frac{1}{2}\vec{a}(t)(\Delta t)^2$
2. Calculate: $\vec{v}(t+\frac{\Delta t}{2}) = \vec{v}(t) + \frac{\vec{a}(t)\Delta t}{2}$
3. Derive $\vec{a}(t+\Delta t)$ from the interaction potential.
4. Calculate: $\vec{v}(t+\Delta t) = \vec{v}(t+\frac{\Delta t}{2}) + \frac{\vec{a}(t+\Delta t)\Delta t}{2}$

www.wikipedia.de

Note that those steps have to be computed separately

1. for the translation part
2. for the rotational part

CGA molecular dynamics solver

- Velocity Verlet in Conformal Geometric Algebra:

$$\begin{array}{lll}
\text{Displacement propagation:} & D(t+\Delta t) & = D(t) + \dot{D}(t)\Delta t + \frac{1}{2}\ddot{D}(t)(\Delta t)^2 \\
\text{Midpoint velocity:} & V_b\left(t+\frac{\Delta t}{2}\right) & = V_b(t) + \dot{V}_b(t)\frac{\Delta t}{2} \\
\text{Acceleration:} & \dot{V}_b(t+\Delta t) & = e_\infty \dot{v}_b(t+\Delta t) - e_{123}\dot{\omega}_b(t+\Delta t) \\
\text{Velocity propagation:} & V_b(t+\Delta t) & = V_b\left(t+\frac{\Delta t}{2}\right) + \frac{1}{2}\dot{V}_b(t+\Delta t)\Delta t
\end{array}$$

with

$$\dot{D} = \frac{1}{2}DV_b\left(= \frac{1}{2}V_b D = \frac{1}{2}DV_b D^{-1} D\right)$$

$$\ddot{D} = \frac{1}{2}\dot{D}V_b + \frac{1}{2}D\dot{V}_b = \frac{1}{4}DV_b^2 + \frac{1}{2}D\dot{V}_b$$

and with the Euclidean pseudoscalar

$$e_{123} = e_1 \wedge e_2 \wedge e_3$$

Florian Seybold (HLRS), based on David Hestenes's work

- CGA combines translational and rotational parts into one compact algorithm.

General Solver concept

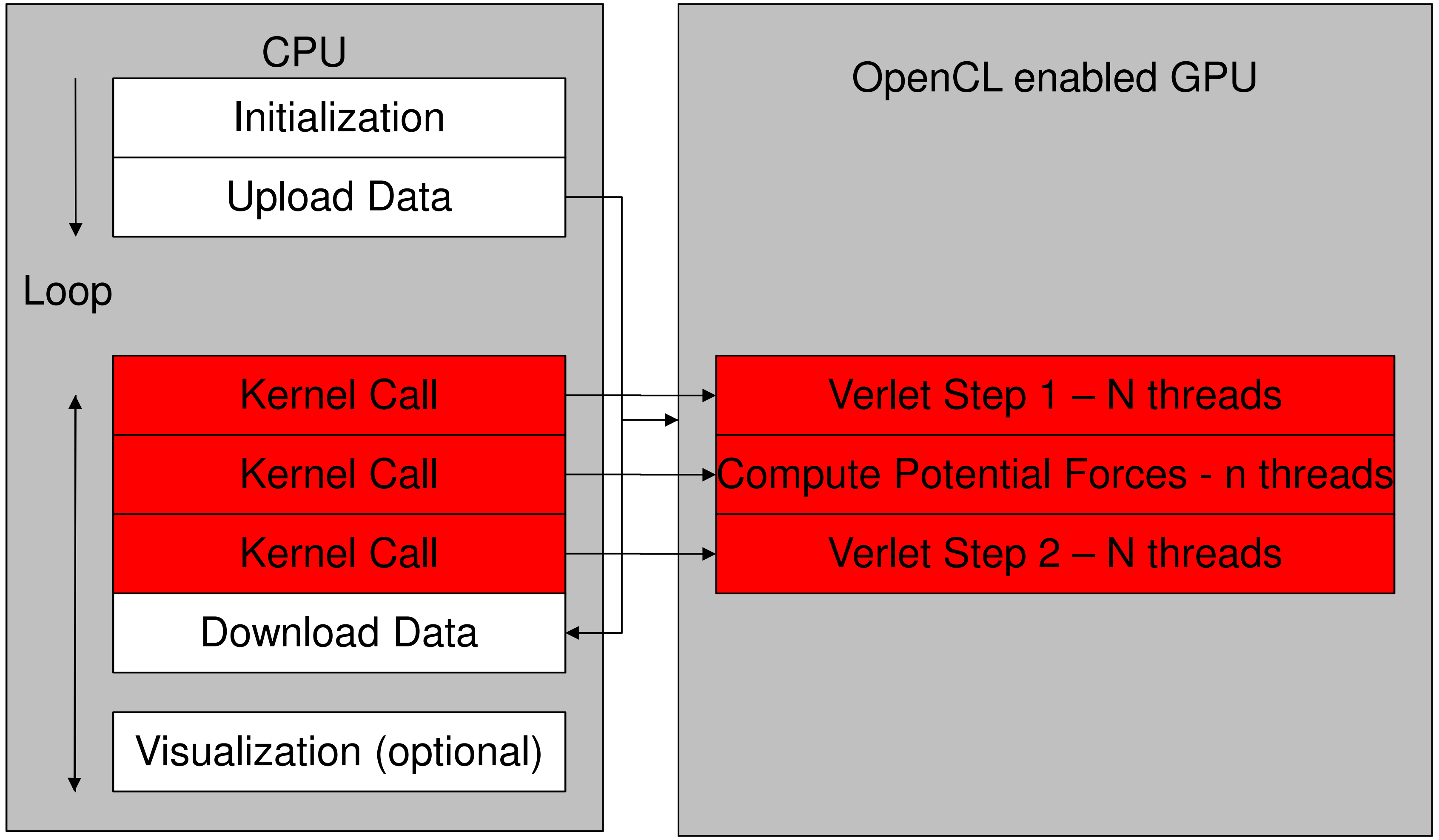
- solver is separated into 3 parts

1. molecule verlet time integration step 1
   - updates the molecule's position and orientation
   - N computations for N molecules

2. computation of potential forces
   - updates each molecule's force and torque
   - n x (n-1) computations for n atoms

3. molecule verlet time integration step 2
   - updates the molecule's linear and angular velocity
   - N computations for N molecules

OpenCL-Solver concept



Implementation

- Gaalop Precompiler for OpenCL code:

```
#pragma clucalc begin
        // calculation
   const floatmv D1_t = 0.5 * D0_t * V0_t;
   const floatmv D2_t = 0.5 * D1_t * V0_t + 0.5 * D0_t * V1_t;

   const floatmv D0_t_dt = D0_t + D1_t * dt + 0.5 * D2_t * dt * dt;
   const floatmv V0_t_05dt = V0_t + 0.5 * V1_t * dt;
#pragma clucalc end
```

$$\begin{array}{lll}
\text{Displacement propagation:} & D(t+\Delta t) & = D(t) + \dot{D}(t)\Delta t + \frac{1}{2}\ddot{D}(t)(\Delta t)^2 \\
\text{Midpoint velocity:} & V_b\left(t+\frac{\Delta t}{2}\right) & = V_b(t) + \dot{V}_b(t)\frac{\Delta t}{2} \\
\text{Acceleration:} & \dot{V}_b(t+\Delta t) & = e_\infty \dot{v}_b(t+\Delta t) - e_{123}\dot{\omega}_b(t+\Delta t) \\
\text{Velocity propagation:} & V_b(t+\Delta t) & = V_b\left(t+\frac{\Delta t}{2}\right) + \frac{1}{2}\dot{V}_b(t+\Delta t)\Delta t
\end{array}$$

with

$$\dot{D} = \frac{1}{2}DV_b\left(= \frac{1}{2}V_bD = \frac{1}{2}DV_bD^{-1}D\right)$$

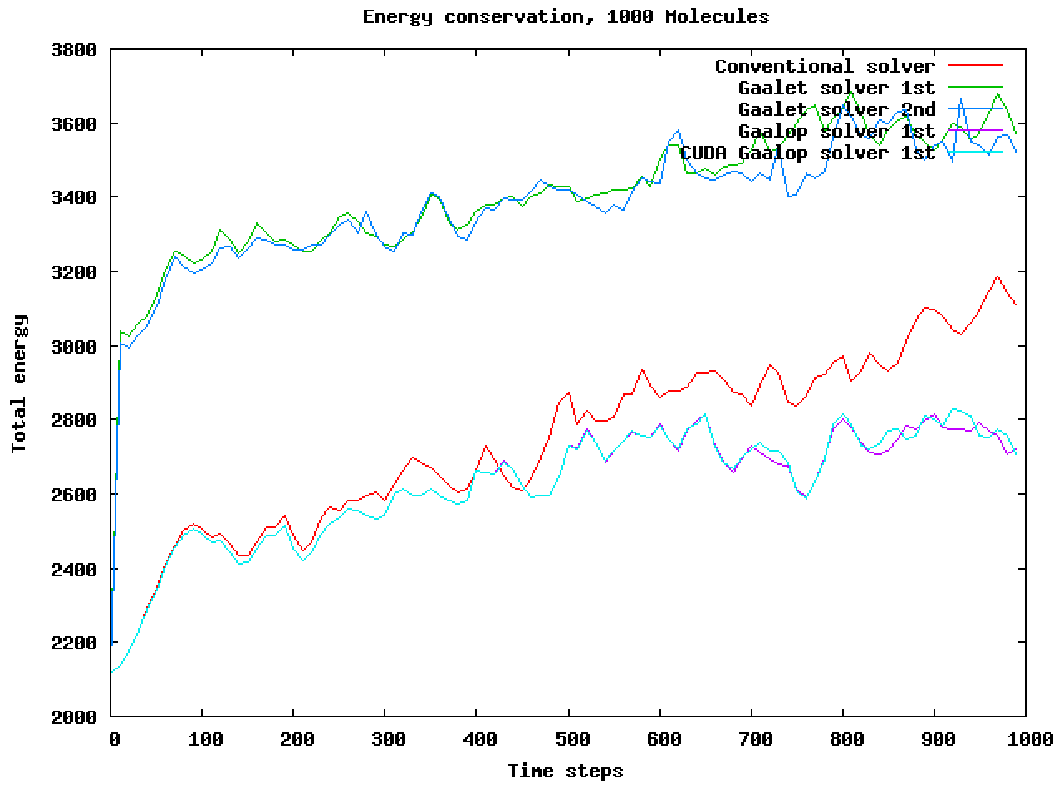
$$\ddot{D} = \frac{1}{2}\dot{D}V_b + \frac{1}{2}D\dot{V}_b = \frac{1}{4}DV_b^2 + \frac{1}{2}D\dot{V}_b$$

and with the Euclidean pseudoscalar

$$e_{123} = e_1 \wedge e_2 \wedge e_3$$

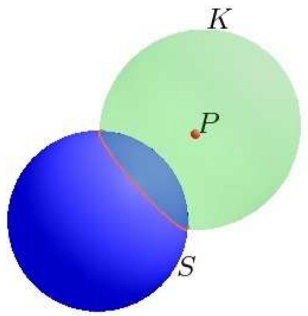
Results

- Better runtime performance
- Better Numerical Stability (Energy Conservation)

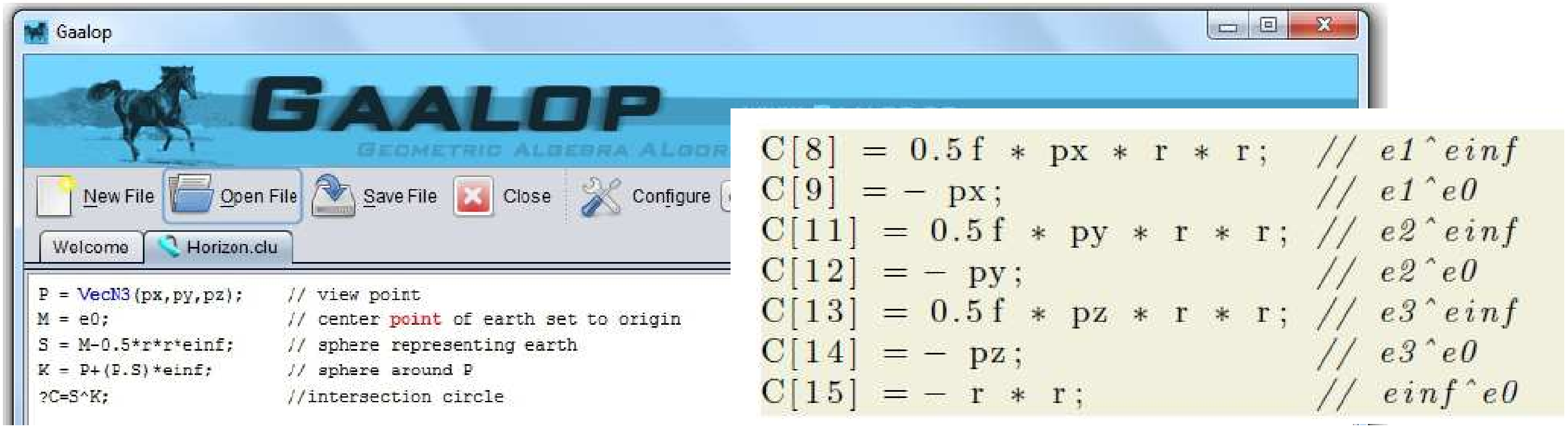


Conclusion

- Geometrically intuitive



- Fast and robust implementations

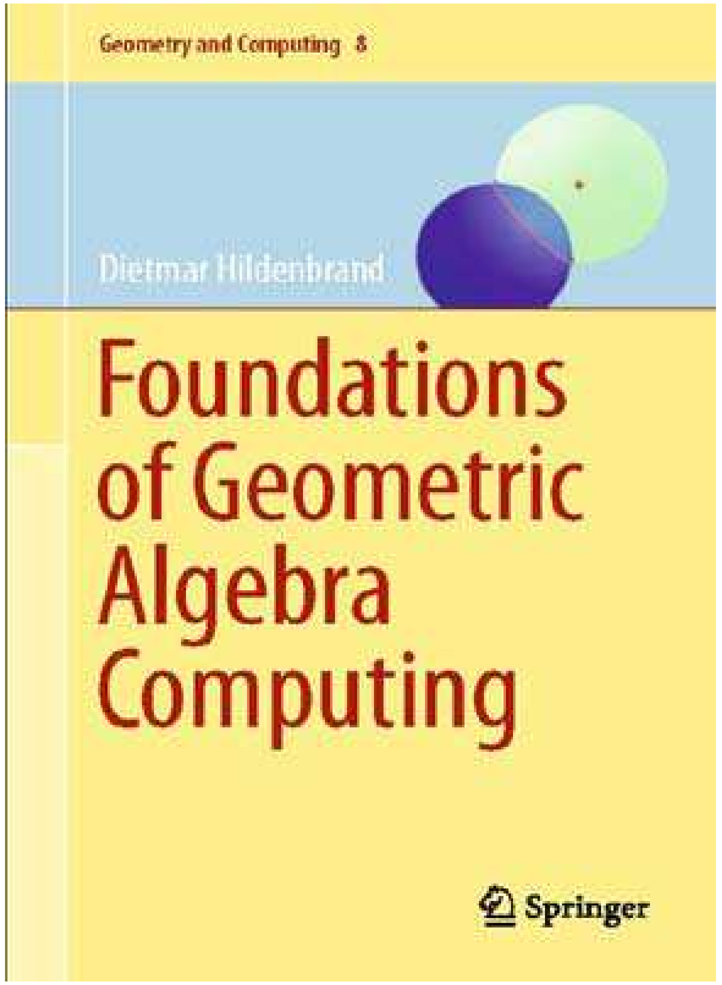


```
P = VecN3(px,py,pz);      // view point
M = e0;                   // center point of earth set to origin
S = M-0.5*r*r*einf;       // sphere representing earth
K = P+(P.S)*einf;         // sphere around P
?C=S^K;                   //intersection circle
```

```
C[8]  = 0.5f * px * r * r;  // e1^einf
C[9]  = - px;               // e1^e0
C[11] = 0.5f * py * r * r;  // e2^einf
C[12] = - py;               // e2^e0
C[13] = 0.5f * pz * r * r;  // e3^einf
C[14] = - pz;               // e3^e0
C[15] = - r * r;            // einf^e0
```

Reference

- „Foundations of Geometric Algebra Computing“
- Dietmar Hildenbrand
- Springer, 2013


Geometry and Computing 8

Dietmar Hildenbrand

Foundations of Geometric Algebra Computing

Springer

TECHNISCHE UNIVERSITÄT DARMSTADT

Thanks a lot

Google Dietmar Hildenbrand