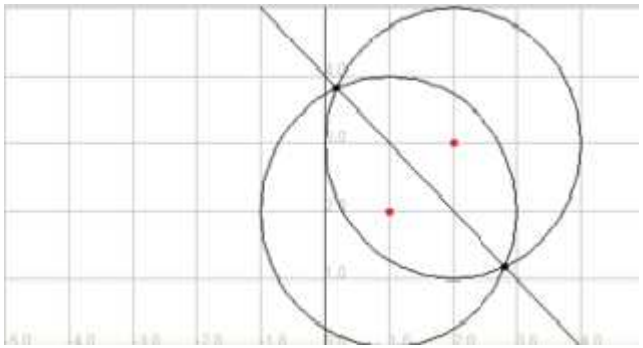
GAALOP for Different Programming Languages

May 08th., 2019

Dr.-Ing. Dietmar Hildenbrand

TU Darmstadt, Germany



GAALOP

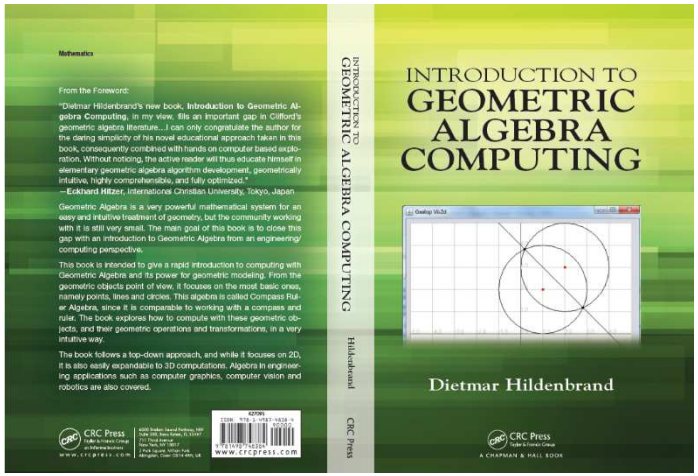
- Software to
  - visualize (2D) Geometric Algebra
  - compute with Geometric Algebra (of arbitrary dimension/signature)
  - generate optimized source code from Geometric Algebra

- GAALOP (**free download** from www.GAALOP.de)

Conformal Geometric Algebra in 2d

Compass Ruler Algebra

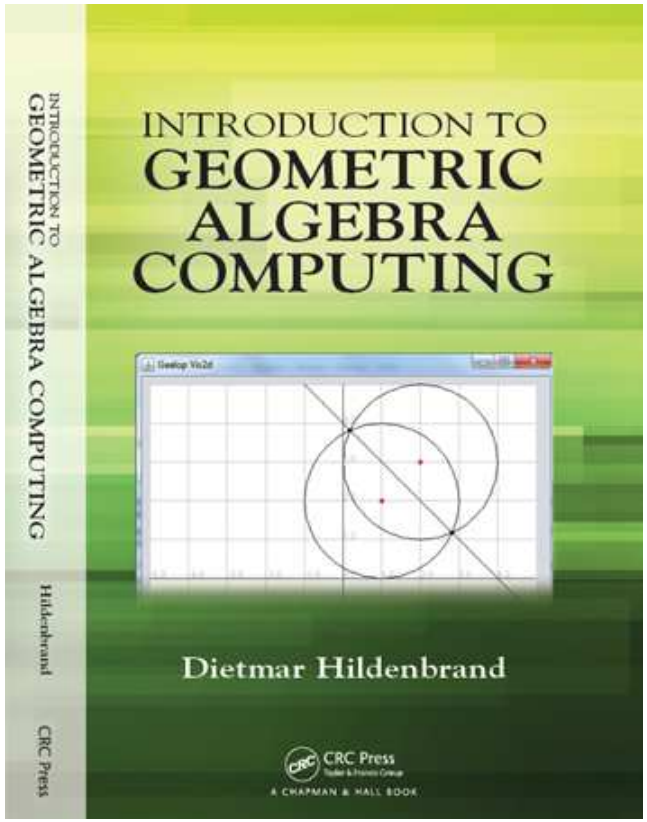
Entity	IPNS representation	OPNS representation
Point	$P = \mathbf{x} + \frac{1}{2}\mathbf{x}^2 e_\infty + e_0$	
Circle	$C = P - \frac{1}{2}r^2 e_\infty$	$C^* = P_1 \wedge P_2 \wedge P_3$
Line	$L = \mathbf{n} + d e_\infty$	$L^* = P_1 \wedge P_2 \wedge e_\infty$
Point pair	$P_p = C_1 \wedge C_2$	$P_p^* = P_1 \wedge P_2$



GAALOP reference

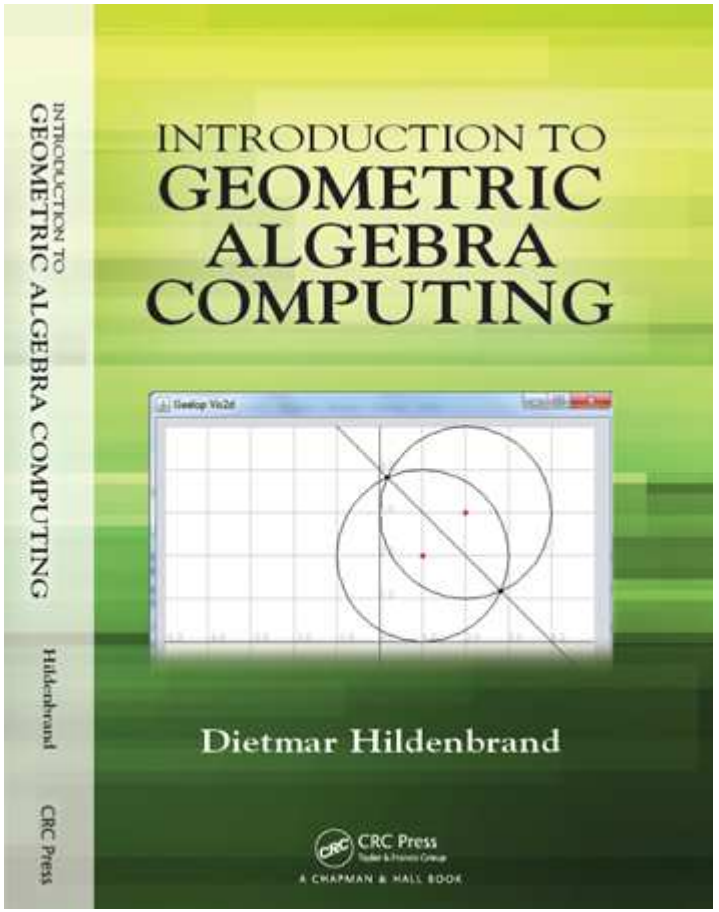
Focus on „Symbolic Geometric Algebra Calculator“

- „Introduction to Geometric Algebra Computing“
- Dietmar Hildenbrand
- CRC Press, 2019



Indices of blades of compass ruler algebra for GAALOP

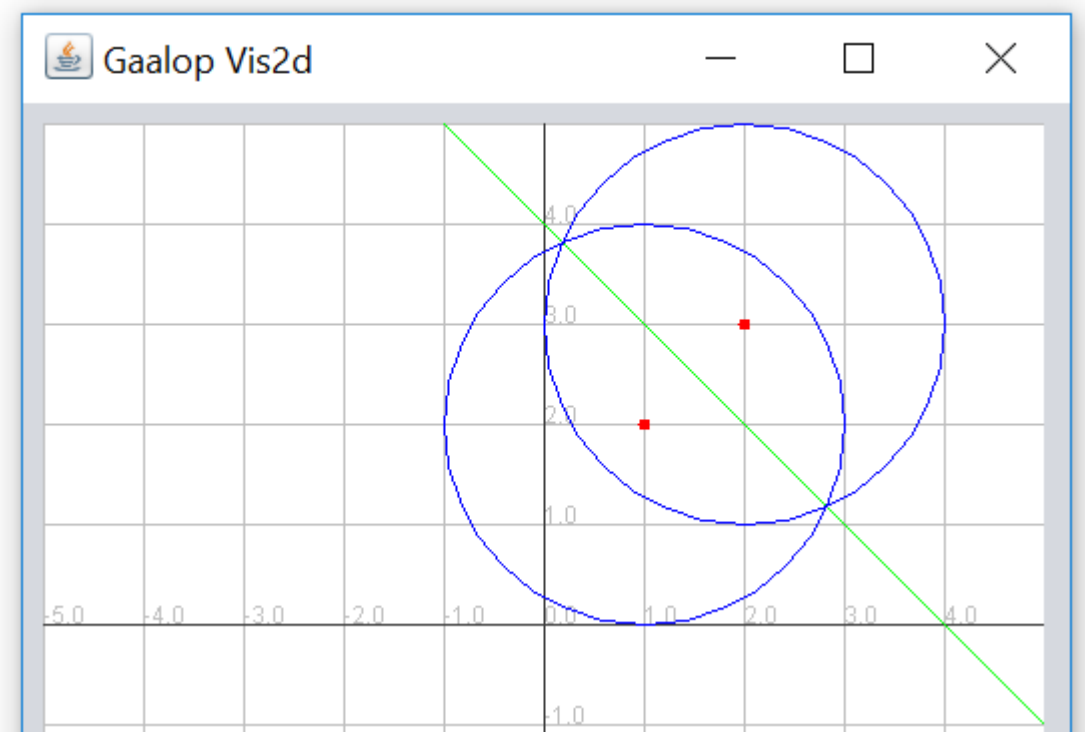
Index	Blade
0	1
1	e_1
2	e_2
3	e_∞
4	e_0
5	$e_1 \wedge e_2$
6	$e_1 \wedge e_\infty$
7	$e_1 \wedge e_0$
8	$e_2 \wedge e_\infty$
9	$e_2 \wedge e_0$
10	$e_\infty \wedge e_0$
11	$e_1 \wedge e_2 \wedge e_\infty$
12	$e_1 \wedge e_2 \wedge e_0$
13	$e_1 \wedge e_\infty \wedge e_0$
14	$e_2 \wedge e_\infty \wedge e_0$
15	$e_1 \wedge e_2 \wedge e_\infty \wedge e_0$



Bisector Example

GAALOP Code

```
P1 = createPoint(x1,y1);
P2 = createPoint(x2,y2);
S1 = P1 - 0.5*r*r*einf;
S2 = P2 - 0.5*r*r*einf;
PP = S1^S2;
?L = *(*PP^einf);
```

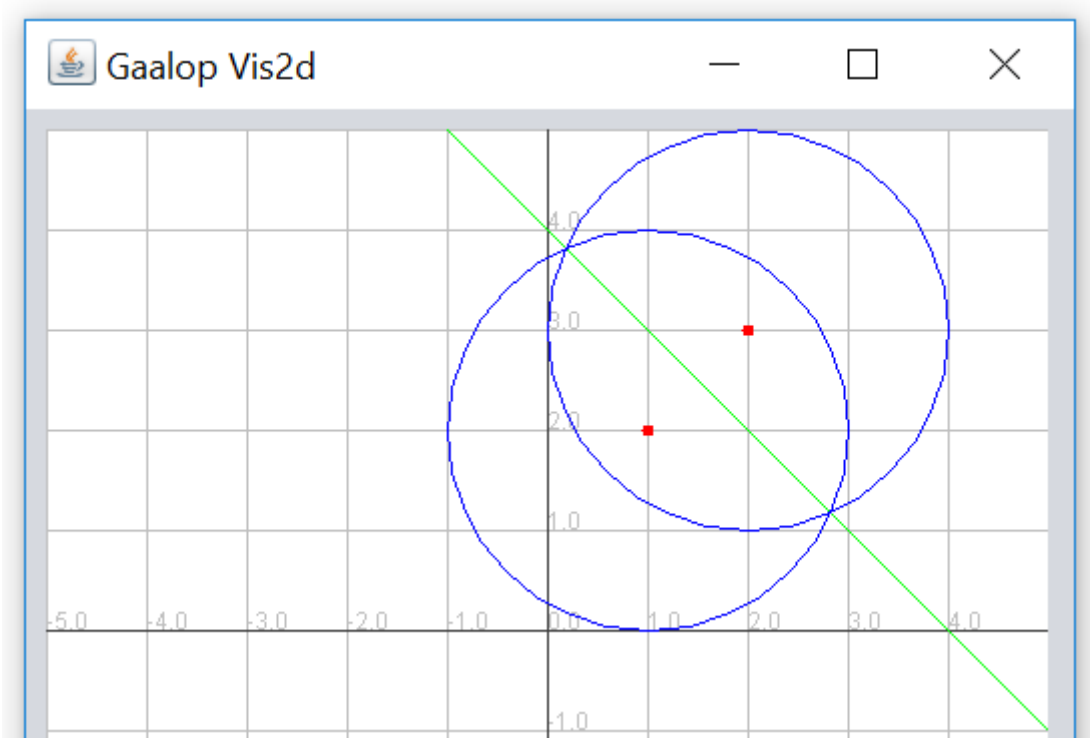


Bisector Example

Additional GAALOP Code for Visualizations

```
x1=1;
y1=2;
x2=2;
y2=3;

----
```

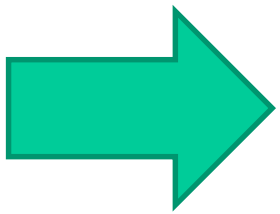



```
:Red;
:P1;
:P2;
:Blue;
:S1;
:S2;
:Green;
:L;
```

Bisector Example

Generated Latex Code

```
\begin{align*}
L_{1}&= x2-x1\\
L_{2}&= y2-y1\\
L_{3}&= \cfrac{y2*y2}{2}-\cfrac{y1*y1}{2}+\cfrac{x2*x2}{2}-\cfrac{x1*x1}{2}\\
\end{align*}
```



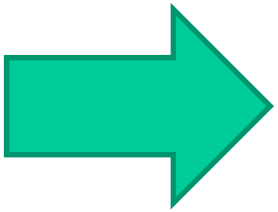
$$L_1 = x2 - x1$$
$$L_2 = y2 - y1$$
$$L_3 = \frac{y2 * y2}{2} - \frac{y1 * y1}{2} + \frac{x2 * x2}{2} - \frac{x1 * x1}{2}$$

Bisector Example

Generated Latex Code

```
\begin{align*}
L_{1}&= x2-x1\\
L_{2}&= y2-y1\\
L_{3}&= \cfrac{y2*y2}{2}-\cfrac{y1*y1}{2}+\cfrac{x2*x2}{2}-\cfrac{x1*x1}{2}\\
\end{align*}
```

What can we show with this result?



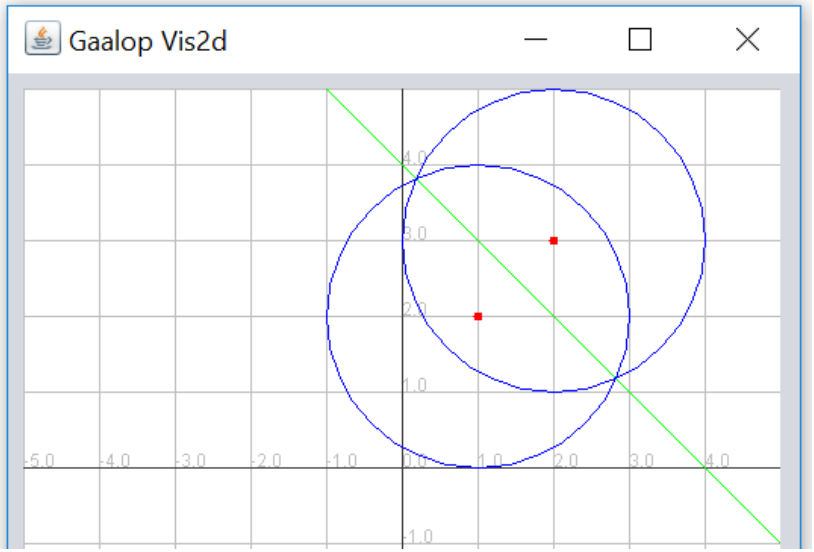
$$L_1 = x2 - x1$$
$$L_2 = y2 - y1$$
$$L_3 = \frac{y2 * y2}{2} - \frac{y1 * y1}{2} + \frac{x2 * x2}{2} - \frac{x1 * x1}{2}$$

Bisector Example

GAALOP code can be shorter

```
P1 = createPoint(x1,y1);
P2 = createPoint(x2,y2);
S1 = P1 - 2*einf;
S2 = P2 - 2*einf;
PP = S1^S2;
?L = *(*PP^einf);
```

```
P1 = createPoint(x1,y1);
P2 = createPoint(x2,y2);
?L= P2-P1;
```

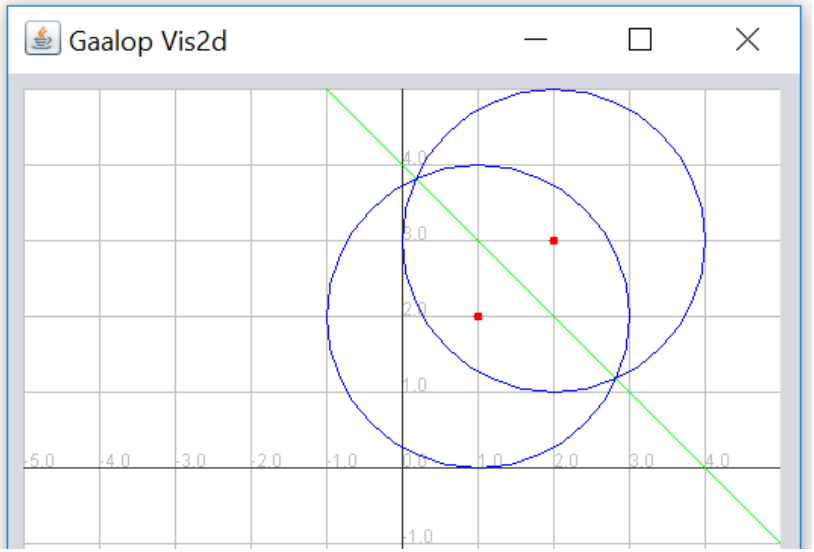


Bisector Example

GAALOP code can be shorter

```
P1 = createPoint(x1,y1);
P2 = createPoint(x2,y2);
S1 = P1 - 2*einf;
S2 = P2 - 2*einf;
PP = S1^S2;
?L = *(*PP^einf);
```

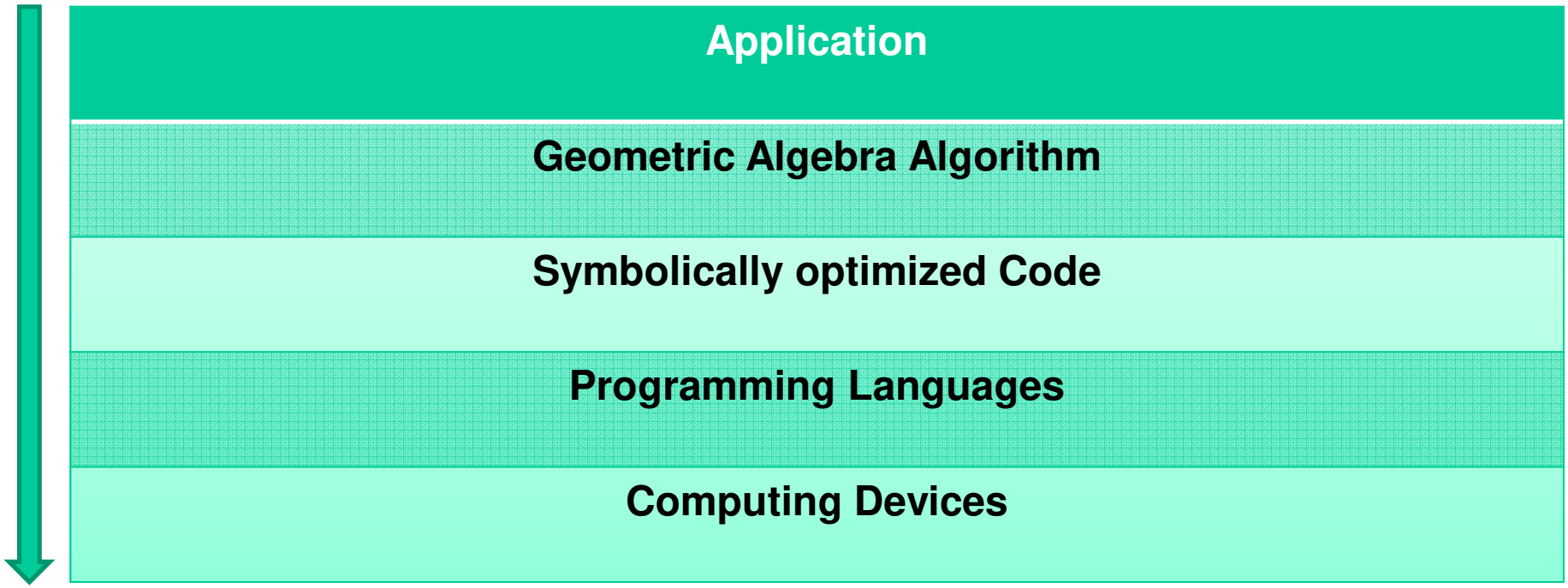
```
P1 = createPoint(x1,y1);
P2 = createPoint(x2,y2);
?L= P2-P1;
```



Whatever approach we use,
at the end
GAALOP is optimizing for some kind
of minimum of needed operations

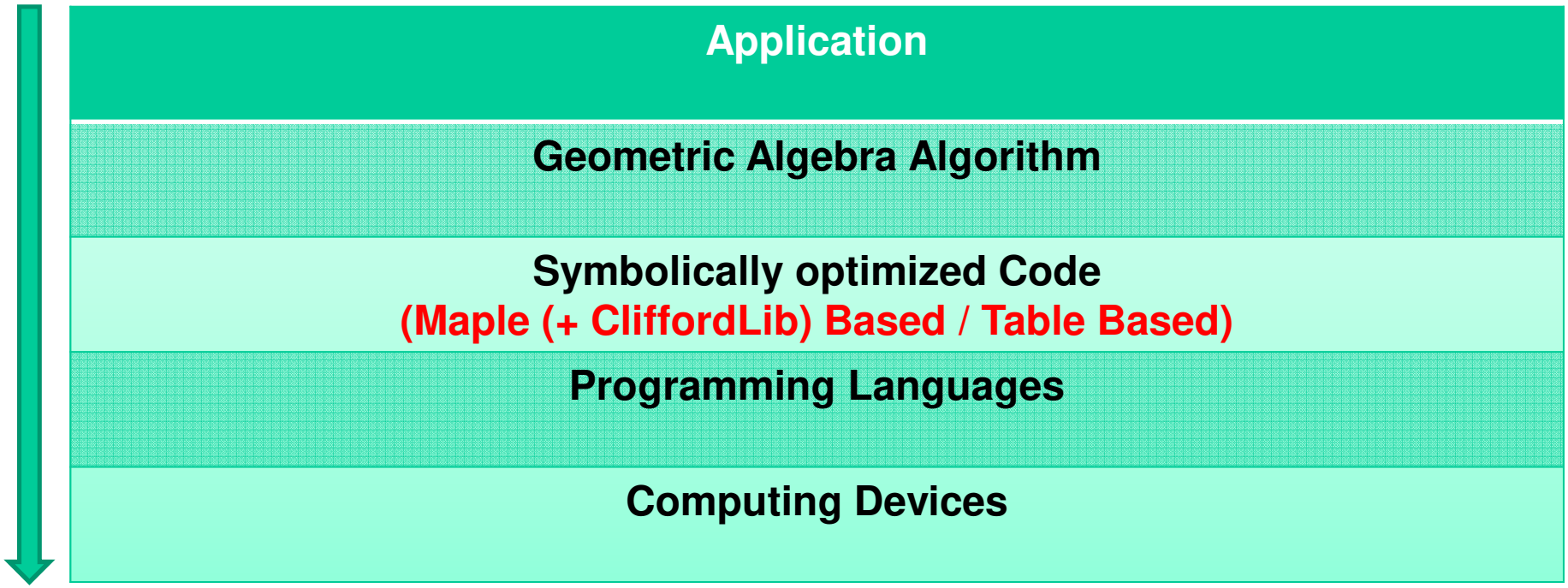
Fast Geometric Algebra Computing for wide range of computing devices

GAALOP Architecture



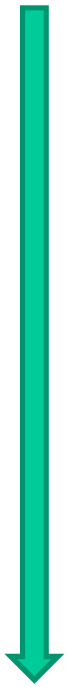
Fast Geometric Algebra Computing for wide range of computing devices

GAALOP Architecture



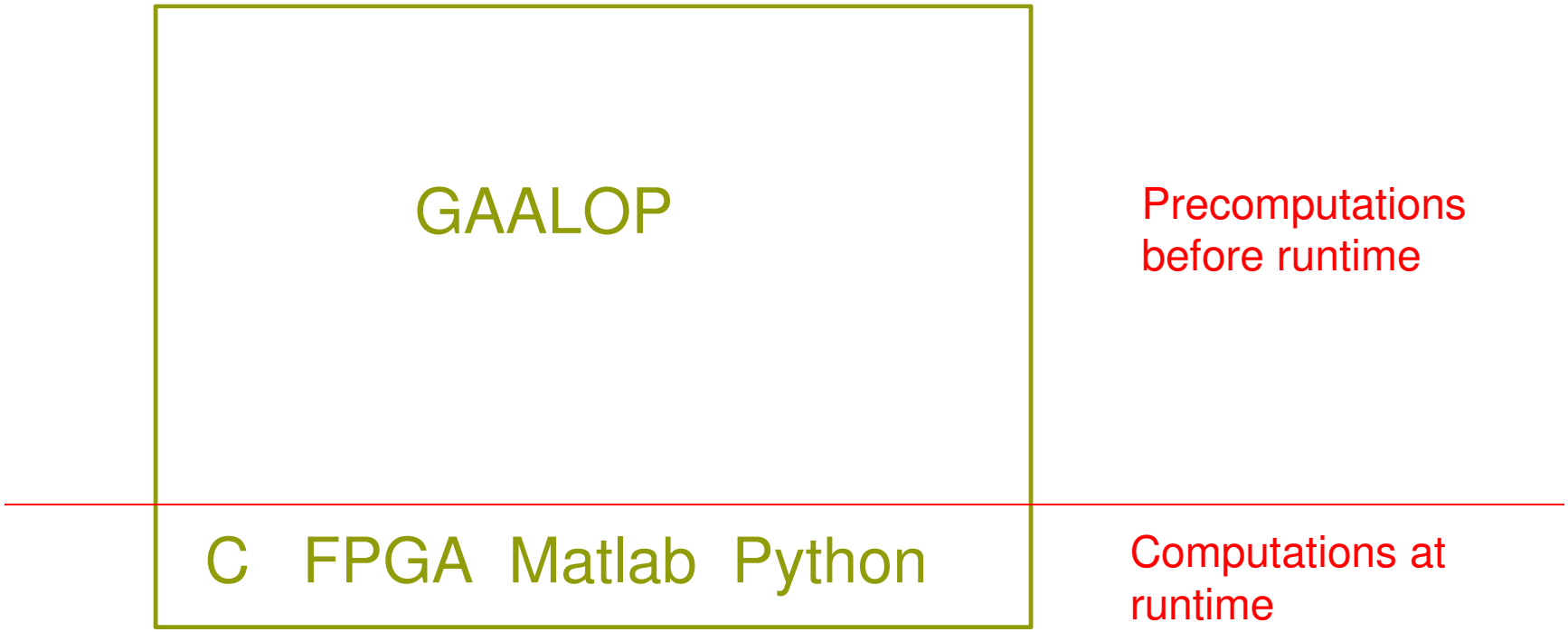
Fast Geometric Algebra Computing for wide range of computing devices

GAALOP Architecture



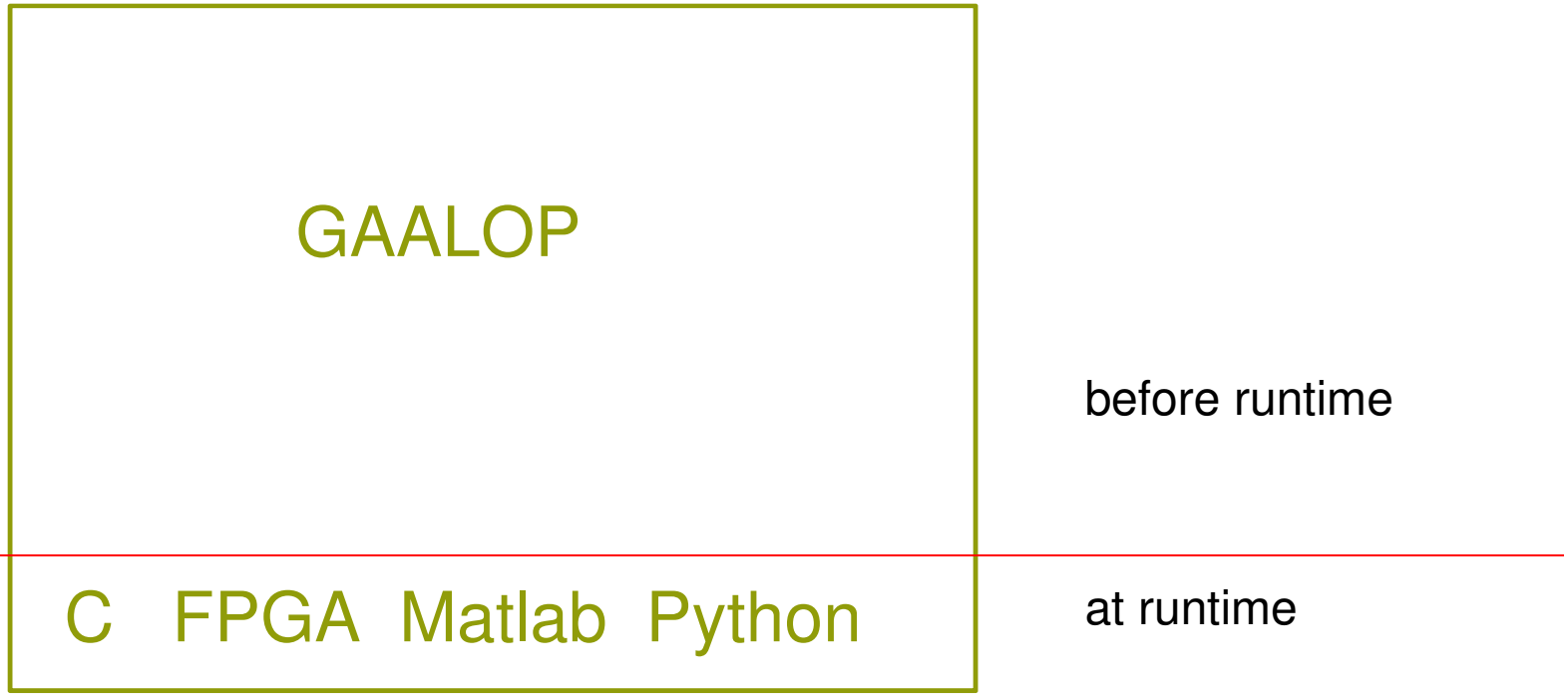
Application
Geometric Algebra Algorithm (CLUCalc notation)
Symbolically optimized Code (GAPP Geometric Algebra Parallelization Programs)
Programming Languages (C, C++, C++ AMP, Java, OpenCL, GAPPCO, Matlab, Python …)
Computing Devices (PC, Tablet, Smart Phone, GPU, DSP, FPGA, SoC, HPC …)

Is GAALOP just another Geometric Algebra tool?



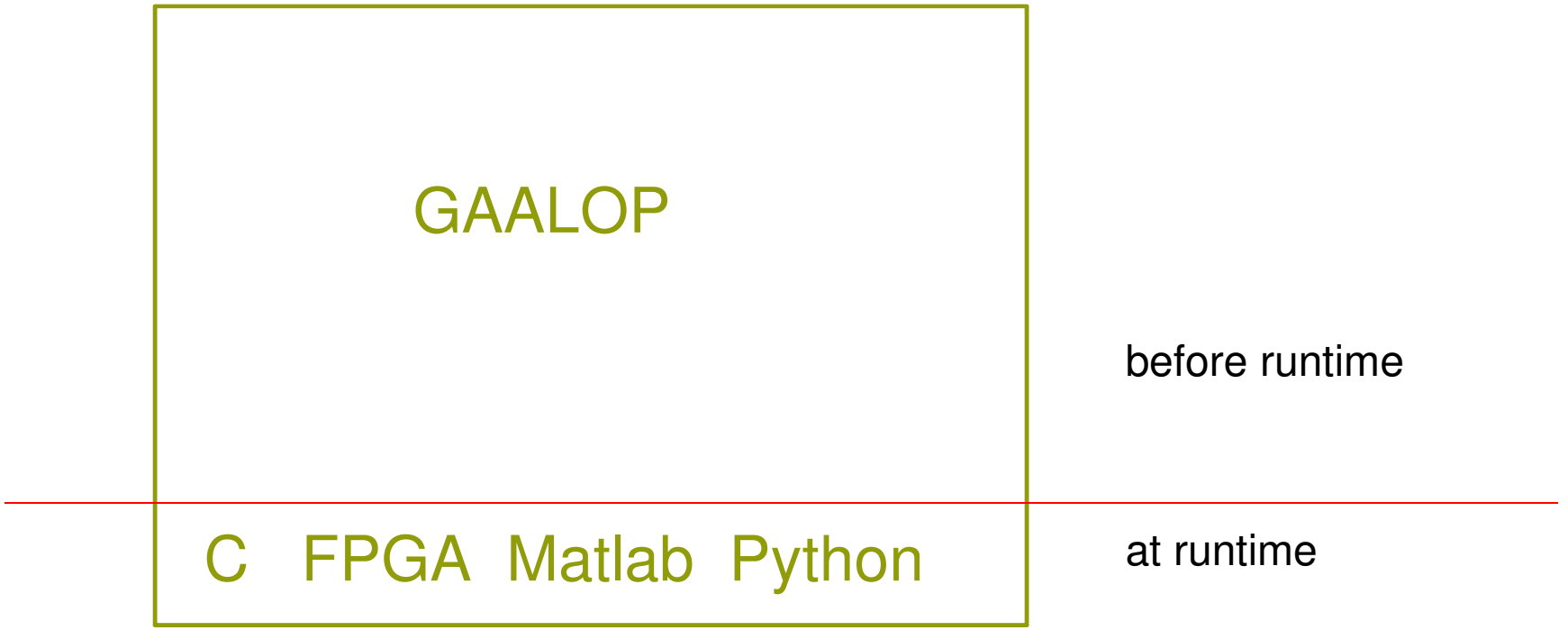
Is GAALOP just another Geometric Algebra tool?

On the machine at runtime only a small amount of computation is needed (the most is already symbollically precomputed)



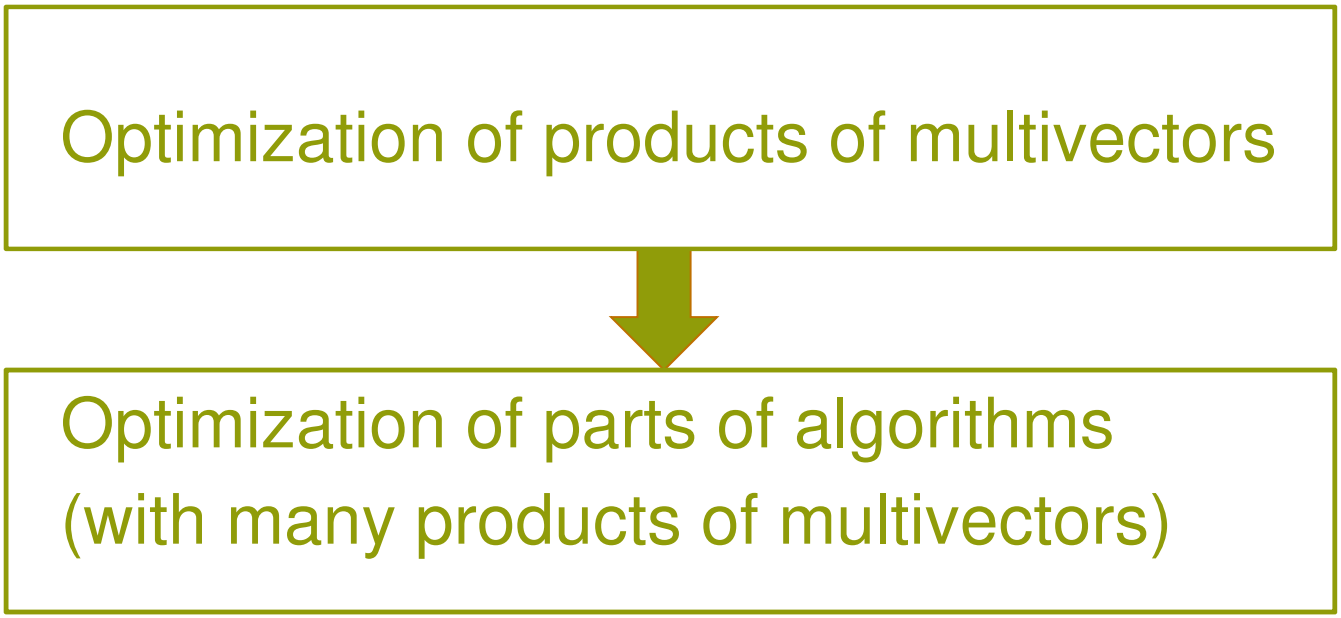
Advantage of precomputation?

- Fast implementations
- Robust implementations



Why is GAALOP so fast?

Two-Stages Optimization



Everything is precomputed before runtime (during compile time)

Symbolic optimization of products

I 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;
```

Symbolic optimization of products

I 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;
```

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;
```

Second Optimization step for algorithms

GAALOP Code for the Bisector Example

```
P1 = createPoint(x1,y1);
P2 = createPoint(x2,y2);
S1 = P1 - 0.5*r*r*einf;
S2 = P2 - 0.5*r*r*einf;
PP = S1^S2;
?L = *(*PP^einf);
```

Only the multivectors with a leading ? are really computed

Bisector Example

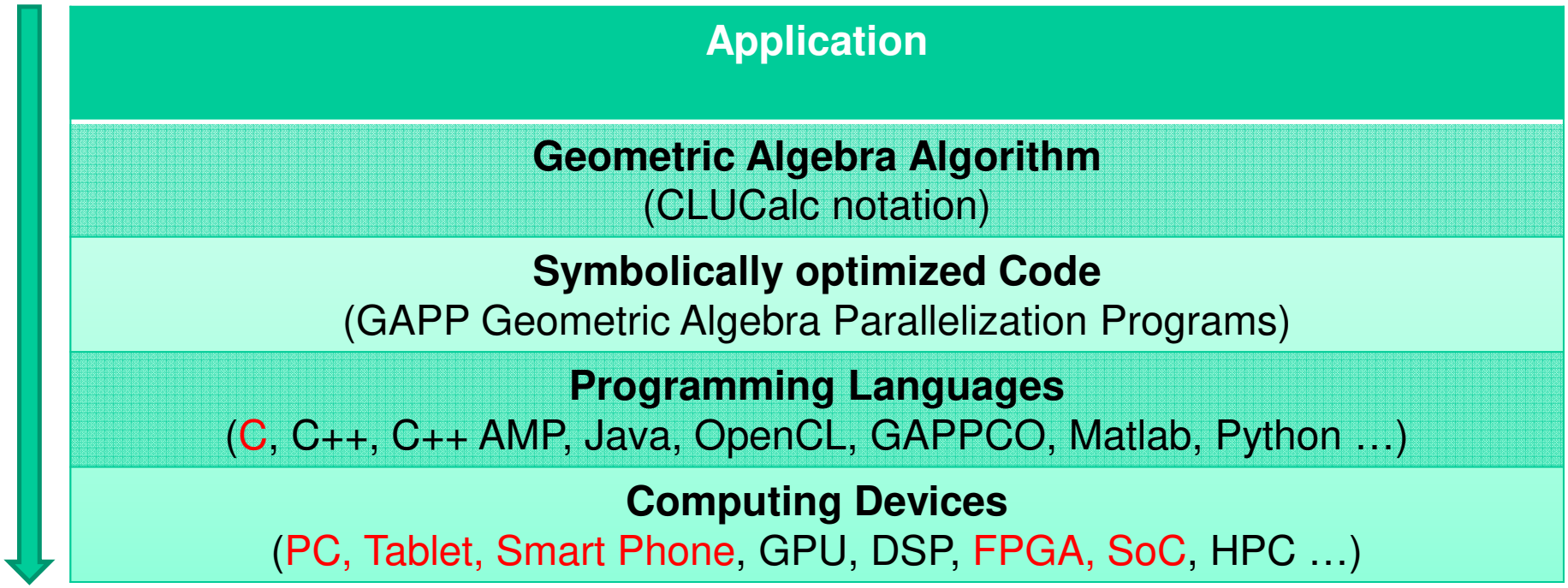
Generated C-Code

```
void calculate(float x1, float x2, float y1, float y2, float L[16]) {
  L[1] = x2 - x1; // e1
  L[2] = y2 - y1; // e2
  L[3] = (y2 * y2) / 2.0 - (y1 * y1) / 2.0 + (x2 * x2) / 2.0 - (x1 * x1) / 2.0; // einf
}
```

- Only the multivector L is computed
- x1, x2, y1, y2, r are variables
- P1, P2, S1, S2, PP are only intermediate results

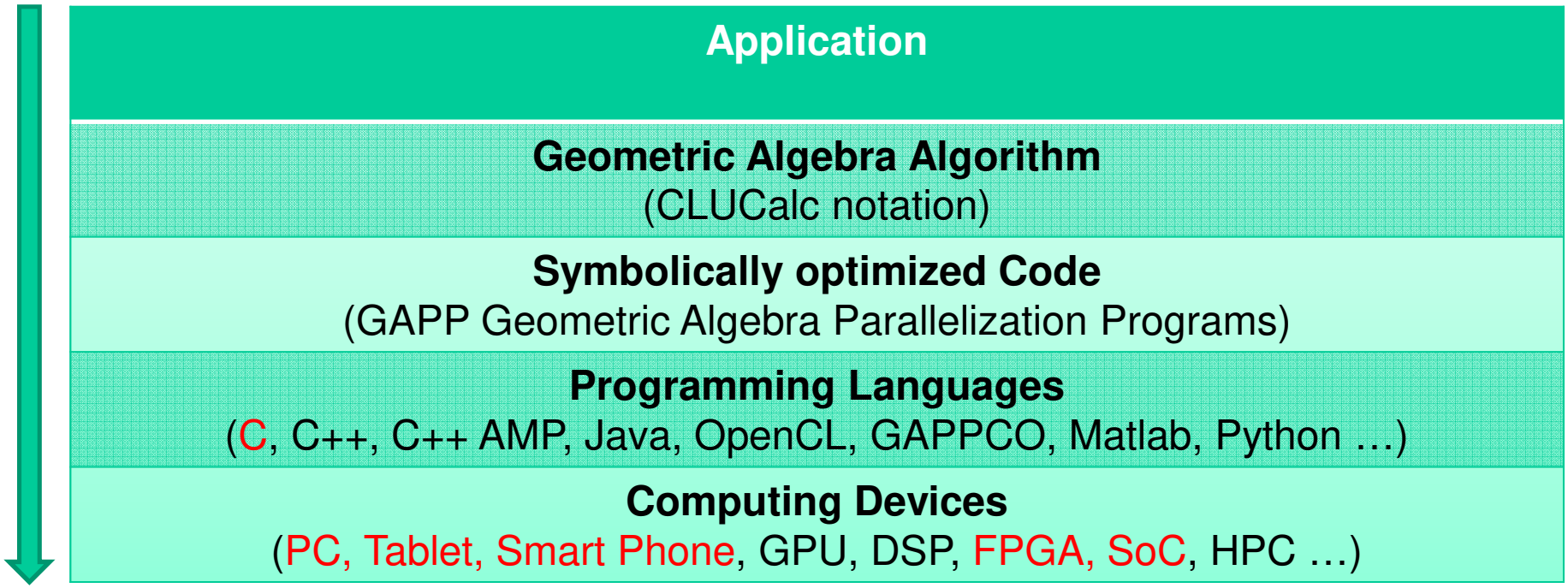
Fast Geometric Algebra Computing for wide range of computing devices

Where can we use this C code?

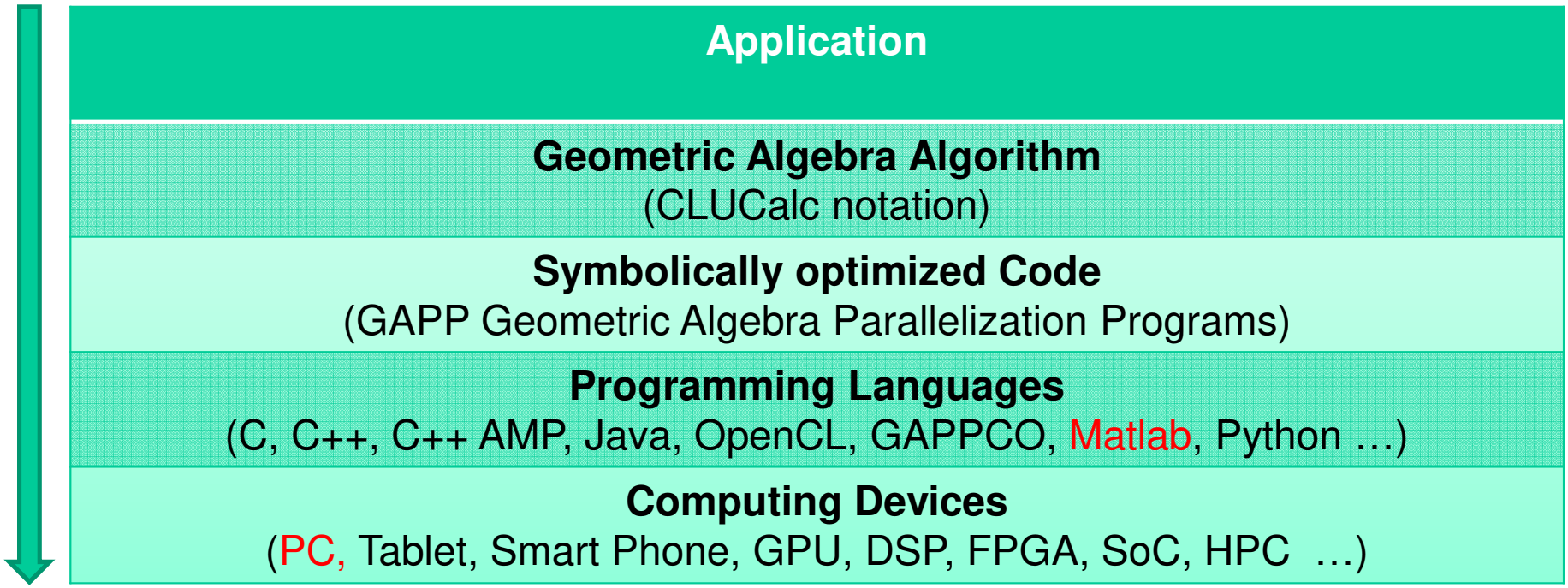


Fast Geometric Algebra Computing for wide range of computing devices

GAALOP -> C



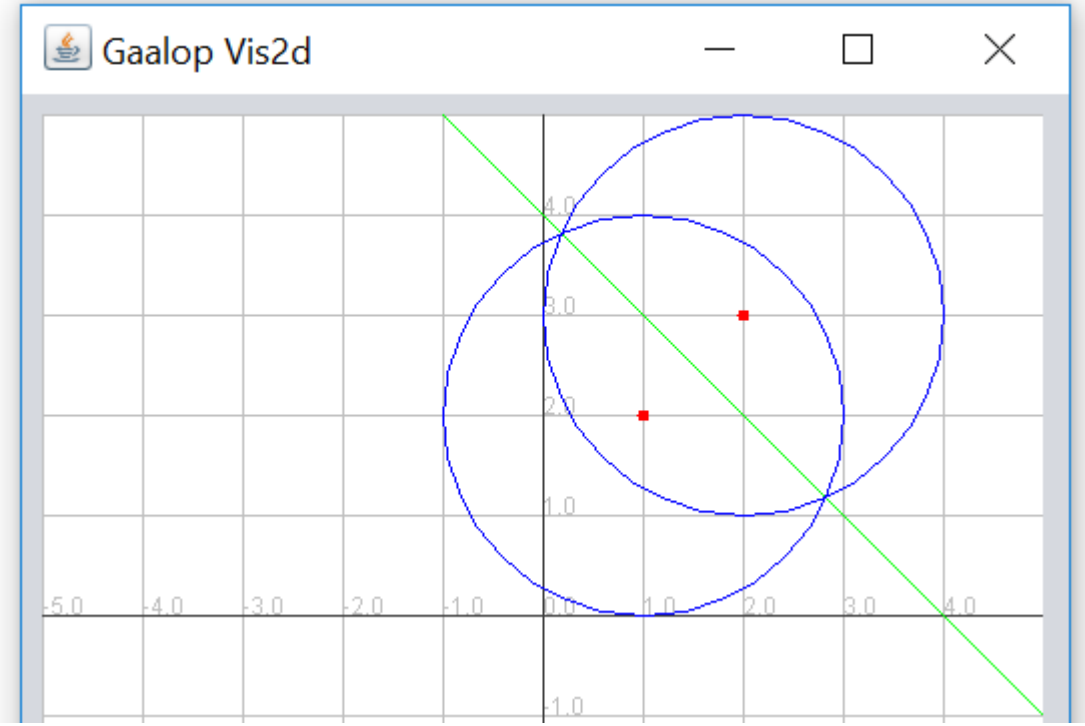
GAALOP -> Matlab



GAALOP -> Matlab

1. Generate C code and save it (Bisector.c)

```
P1 = createPoint(x1,y1);
P2 = createPoint(x2,y2);
S1 = P1 - 0.5*r*r*einf;
S2 = P2 - 0.5*r*r*einf;
PP = S1^S2;
?L = *(*PP^einf);
```



Bisector Example

2. Automatically import optimized Matlab Code

```
function [L] = Bisector (x1, x2, y1, y2)
    L(1) = x2 - x1;
    L(2) = y2 - y1;
    L(3) = (y2 * y2) / 2.0 - (y1 * y1) / 2.0 + (x2 * x2) / 2.0 - (x1 * x1) / 2.0;
end
```

via

importGAALOP('L', 'Bisector')

Bisector Example

Matlab Call

```
>> L = Bisector(1,2,3,4)
L =
   1   1   5
>>
```

Only after a change of Bisector.clu in GAALOP, you have to call
>> importGAALOP(‘L’, ‘Bisector’)
again

Recall: Bisector Example

Generated C-Code

```
void calculate(float x1, float x2, float y1, float y2, float L[16]) {
  L[1] = x2 - x1; // e1
  L[2] = y2 - y1; // e2
  L[3] = (y2 * y2) / 2.0 - (y1 * y1) / 2.0 + (x2 * x2) / 2.0 - (x1 * x1) / 2.0; // einf
}
```

- Only the multivector L is computed
- x1, x2, y1, y2, r are variables

importGAALOP (multivector, GAFunctionName)

Handle the first line of the C-file

Line = strrep(Line, multivector, '');

delete the name of the multivector (L)

```
void calculate(float x1, float x2, float y1, float y2, float [16]) {

        L[1] = 2.0 * x2 - 2.0 * x1; // e1
        L[2] = 2.0 * y2 - 2.0 * y1; // e2
        L[3] = y2 * y2 - y1 * y1 + x2 * x2 - x1 * x1; // einf
}
```

importGAALOP (multivector, GAFunctionName)

Handle the first line of the C-file

```
Line = strrep(Line, 'float ', '');

delete all ‘float’s

void calculate( x1,  x2,  y1,  y2, [16]) {
        L[1] = 2.0 * x2 - 2.0 * x1; // e1
        L[2] = 2.0 * y2 - 2.0 * y1; // e2
        L[3] = y2 * y2 - y1 * y1 + x2 * x2 - x1 * x1; // einf
}
```

importGAALOP (multivector, GAFunctionName)

Handle the first line of the C-file

```
Line = strtrunc(Line,index(Line,', [') -1);

truncate starting with ‘, [‘

void calculate( x1,  x2,  y1,  y2
        L[1] = 2.0 * x2 - 2.0 * x1; // e1
        L[2] = 2.0 * y2 - 2.0 * y1; // e2
        L[3] = y2 * y2 - y1 * y1 + x2 * x2 - x1 * x1; // einf
}
```

importGAALOP (multivector, GAFunctionName)

Handle the first line of the C-file

```
MatlabString = strcat('function [', multivector, '] = ', GAFunctionName,' ');
Line = strrep(Line, 'void calculate', MatlabString );
```

Replace 'void calculate' by 'function [L] = Bisector '

```
function [L] = Bisector( x1,  x2,  y1,  y2
        L[1] = 2.0 * x2 - 2.0 * x1; // e1
        L[2] = 2.0 * y2 - 2.0 * y1; // e2
        L[3] = y2 * y2 - y1 * y1 + x2 * x2 - x1 * x1; // einf
}
```

importGAALOP (multivector, GAFunctionName)

Handle the first line of the C-file

Line = strcat(Line, ‘)’);

add ‘)’ at the end of the line

```
function [L] = Bisector( x1,  x2,  y1,  y2)
        L[1] = 2.0 * x2 - 2.0 * x1; // e1
        L[2] = 2.0 * y2 - 2.0 * y1; // e2
        L[3] = y2 * y2 - y1 * y1 + x2 * x2 - x1 * x1; // einf
}
```

first line completed

importGAALOP (multivector, GAFunctionName)

For all multivector coefficients

```
Line = strrep(Line, '//', '%');   % replace characters for comments
CoeffNo = CoeffNo+1;  % CoeffNo ={1, 2, ...}
Head = strcat( multivector, '(', int2str(CoeffNo), ')' ); % generate 'multivector(No)'
Line = substr(Line, index(Line, ']')+1); % substring after ']'
Line = strcat(Head, Line);
```

```
function [L] = Bisector( x1,  x2,  y1,  y2)
        L(1) = 2.0 * x2 - 2.0 * x1; % e1
        L(2) = 2.0 * y2 - 2.0 * y1; % e2
        L(3) = y2 * y2 - y1 * y1 + x2 * x2 - x1 * x1; % einf
}
```

importGAALOP (multivector, GAFunctionName)

Last line

```
fprintf(writeFile, 'end' ); % this is working for Octave and Matlab


function [L] = Bisector( x1,  x2,  y1,  y2)
      L(1) = 2.0 * x2 - 2.0 * x1; % e1
      L(2) = 2.0 * y2 - 2.0 * y1; % e2
      L(3) = y2 * y2 - y1 * y1 + x2 * x2 - x1 * x1; % einf
end
```

M-file completed

showGAALOPCoefficients

Matlab Call

```
 >> showGAALOPCoefficients('L', 'Bisector');
1 : e1
2 : e2
3 : einf
>>
```

Based on a loop with

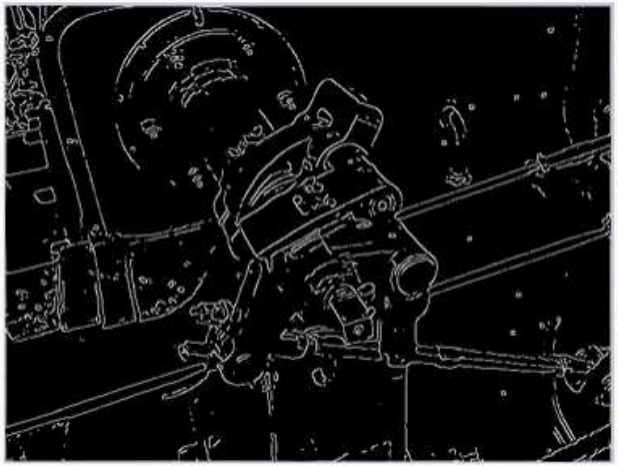
```
  Line = strtrim(Line); # remove leading blanks
  No = No+1;            # indices 1..
  myLine = strcat(No, " : ");
  Line = substr(Line, index(Line, "// ")+2);   # text after the comment
  Line = strcat(myLine, Line);     # concatenate index and its meaning
```

Detection of Circles and Lines in Images Using GAALOP

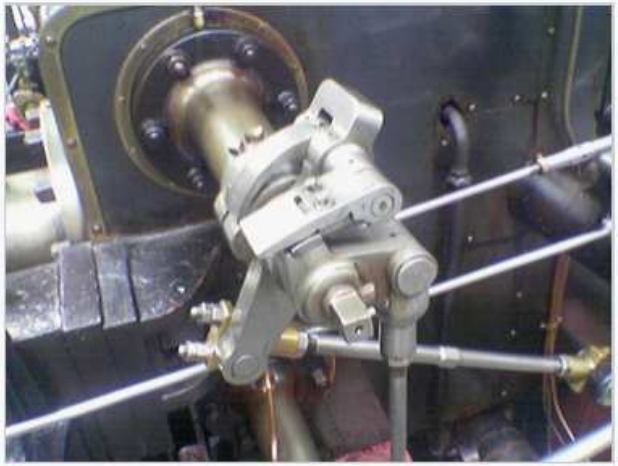
- CGAVS (Conformal Geometric Algebra Voting Scheme)

[65] G. Soria-Garcia, G. Altamirano-Gomez, S. Ortega-Cisneros, and Eduardo Bayro Corrochano. Fpga implementation of a geometric voting scheme for the extraction of geometric entities from images. In *Advances in Applied Clifford Algebras Journal*, Sept. 2016.

- Basis: edge image showing only the discontinuities of a photograph.
- https://en.wikipedia.org/wiki/Canny_edge_detector



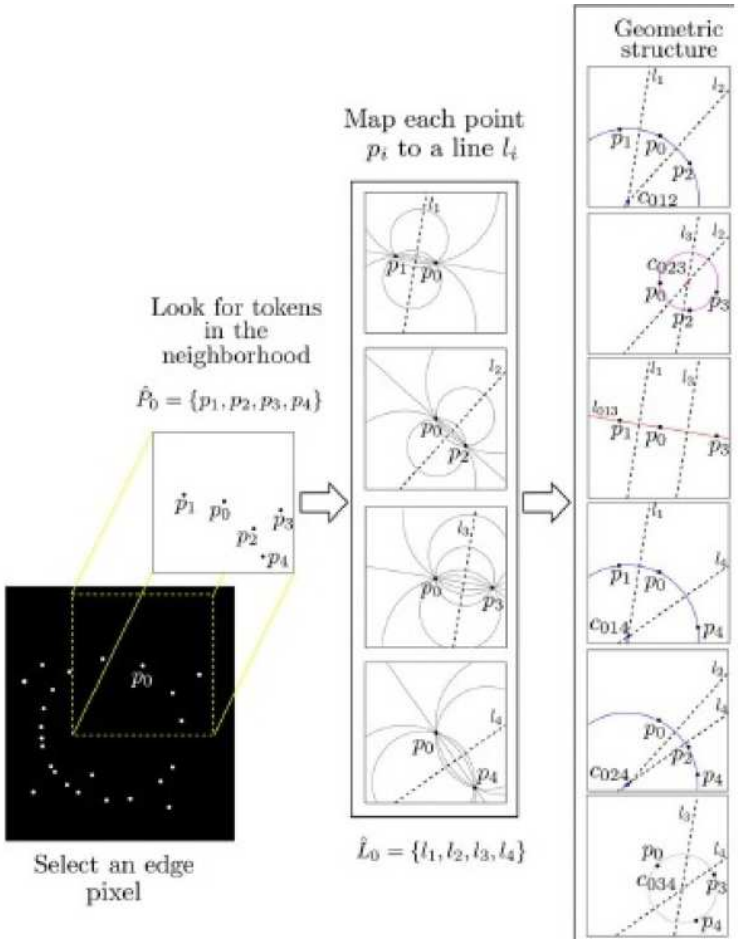
The Canny edge detector applied to a color photograph of a steam engine.



The original image.

CGAVS (Conformal Geometric Algebra Voting Scheme)

- Select an edge pixel
- Look for tokens in neighborhood
- Map each point to a line
- Intersect lines
- Circles/lines?



CGAVS (Conformal Geometric Algebra Voting Scheme)

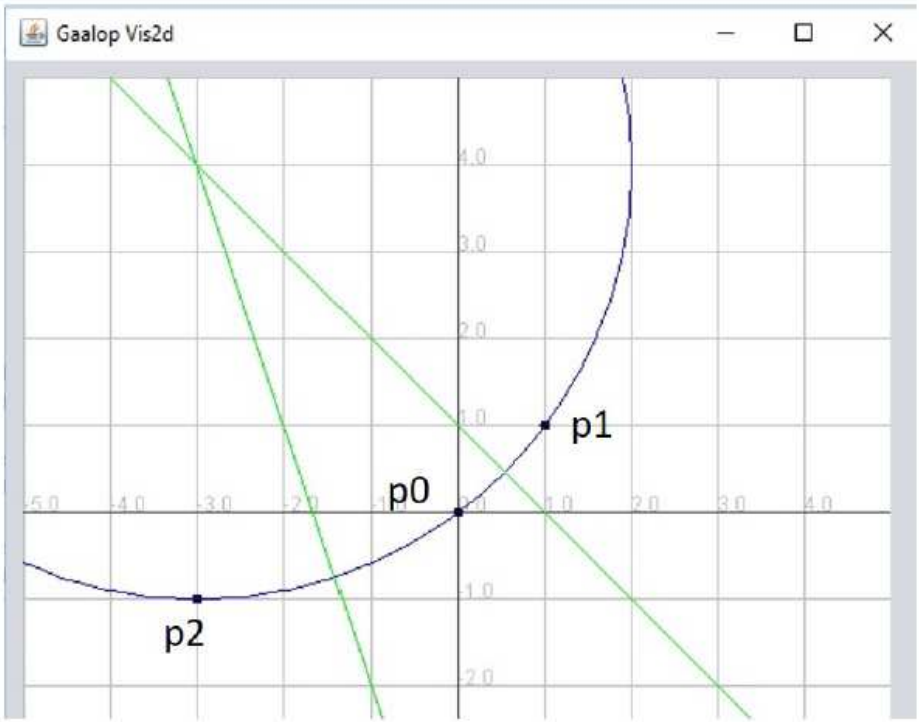


FIGURE 10.2 Visualization of *EntitiesExtraction.clu*: compute the circle through three points.

Python

How to import GAALOP?

```
from clifford.g3c import *
def Bisector( x1,  x2,  y1,  y2 ):
    MV = ( 2.0 * x2 - 2.0 * x1)*( e1)
    MV += ( 2.0 * y2 - 2.0 * y1)*( e2)
    MV += ( y2 * y2 - y1 * y1 + x2 * x2 - x1 * x1)*( einf)
    return MV
```

importGAALOP (multivector, GAFunctionName)

Handle the first line of the C-file

line = line.replace(multivector, "")

delete the name of the multivector (L)

```
void calculate(float x1, float x2, float y1, float y2, float [16]) {

        L[1] = 2.0 * x2 - 2.0 * x1; // e1
        L[2] = 2.0 * y2 - 2.0 * y1; // e2
        L[3] = y2 * y2 - y1 * y1 + x2 * x2 - x1 * x1; // einf
}
```

importGAALOP (multivector, GAFunctionName)

Handle the first line of the C-file

```
line = line.replace('float', '')
```

delete all ‘float’s

```
void calculate( x1,  x2,  y1,  y2, [16]) {
        L[1] = 2.0 * x2 - 2.0 * x1; // e1
        L[2] = 2.0 * y2 - 2.0 * y1; // e2
        L[3] = y2 * y2 - y1 * y1 + x2 * x2 - x1 * x1; // einf
}
```

importGAALOP (multivector, GAFunctionName)

Handle the first line of the C-file

```
line = line.split(',  [',1)[0]

truncate starting with ‘,  [‘

void calculate( x1,  x2,  y1,  y2
        L[1] = 2.0 * x2 - 2.0 * x1; // e1
        L[2] = 2.0 * y2 - 2.0 * y1; // e2
        L[3] = y2 * y2 - y1 * y1 + x2 * x2 - x1 * x1; // einf
}
```

importGAALOP (multivector, GAFunctionName)

Handle the first line of the C-file

line = line.replace('void calculate', ‘def ' + GAFunctionName)

line = line + ‘):‘

Complete first line in Python style

```
Def Bisector( x1, x2, y1, y2 ):

    L[1] = 2.0 * x2 - 2.0 * x1; // e1
    L[2] = 2.0 * y2 - 2.0 * y1; // e2
    L[3] = y2 * y2 - y1 * y1 + x2 * x2 - x1 * x1; // einf
}
```

first line completed

importGAALOP (multivector, GAFunctionName)

Handle the second line

Initialize the multivector (according to the highest index of the multivector)

```
Def Bisector( x1, x2, y1, y2 ):
    L=[0,0,0,0]
    L[1] = 2.0 * x2 - 2.0 * x1; // e1
    L[2] = 2.0 * y2 - 2.0 * y1; // e2
    L[3] = y2 * y2 - y1 * y1 + x2 * x2 - x1 * x1; // einf
}
```

importGAALOP (multivector, GAFunctionName)

Python code for the second line

```
# compute string for multivector
line = Lines[length-2] # the line with the biggest index
Split_str = line.split('[',1)
line = Split_str[1]             # text after [
Split_str = line.split(']',1)
No = Split_str[0]               # the coefficient of the multivector
InitStr = '[0'
i=1
while i<= int(No):
    InitStr += ',0'
    i+=1
InitStr += ']'
# Complete second line
SecondLine = '    ' + multivector + '=' + InitStr + '\n'
```

importGAALOP (multivector, GAFunctionName)

For all multivector coefficients

line = line.replace('//', '#')

replace characters for comments

```
Def Bisector( x1, x2, y1, y2 ):

    L[1] = 2.0 * x2 - 2.0 * x1; # e1
    L[2] = 2.0 * y2 - 2.0 * y1; # e2
    L[3] = y2 * y2 - y1 * y1 + x2 * x2 - x1 * x1; # einf
}
```

importGAALOP (multivector, GAFunctionName)

Last line and definition of L

line = line + 'return ' + multivector

```
Def Bisector( x1, x2, y1, y2 ):
    L=[0,0,0,0]
    L[1] = 2.0 * x2 - 2.0 * x1; # e1
    L[2] = 2.0 * y2 - 2.0 * y1; # e2
    L[3] = y2 * y2 - y1 * y1 + x2 * x2 - x1 * x1; # einf
    return L
```

Python-file completed

showGAALOPCoefficients()

Python Call

```
from GAALOP import *
importGAALOP('L', 'Bisector')
showGAALOPCoefficients('L', 'Bisector')
```

showGAALOPCoefficients()

Python Code

Loop over:

```
Split_str = line.split('[',1)
line = Split_str[1]             # text after [
Split_str = line.split(']',1)
No = Split_str[0]               # the coefficient of the multivector
line = Split_str[1]
Split_str = line.split('//',1)  # split the comment
print(No, ' : ', Split_str[1])  # index of the multivector and its meaning
```

Thanks a lot

Google Dietmar Hildenbrand