CHAPTER 1

GAALOPScript

CONTENTS

GAALOPScript is a general language in order to describe Geometric Algebra algorithms. Here, we focus on one specific Geometric Algebra (Conformal Geometric Algebra) which is very important for many applications in computer graphics, computer vision and robotics.

1.1 Conformal Geometric Algebra

Conformal Geometric Algebra is very well suitable to realize engineering applications. This is primarily because of its easy handling of geometric objects such as spheres, planes and lines. For more details about Conformal Geometric Algebra see [2].

Conformal Geometric Algebra uses the three Euclidean **basis vectors** e_1, e_2, e_3 and two additional orthogonal basis vectors e_+, e_- with positive and negative signatures, respectively, which means that they square to $+1$ as usual (e_+) and to -1 (e_-), related to the geometric product.

$$e_+^2 = 1, \qquad e_-^2 = -1, \qquad e_+ \cdot e_- = 0. \tag{1.1}$$

Another basis e_0, e_∞, with the geometric meaning

e_0 represents the 3D origin,

e_∞ represents infinity,

can be defined by the transformation

$$e_0 := \frac{1}{2}(e_- - e_+), \qquad e_\infty := e_- + e_+. \tag{1.2}$$

TABLE 1.1 The two representations (IPNS and OPNS) of conformal geometric entities. The IPNS and OPNS representations are dual to each other, which is indicated by the asterisk symbol.

Entity	IPNS representation	OPNS representation
Point	$P = \mathbf{x} + \frac{1}{2}\mathbf{x}^2 e_\infty + e_0$	
Sphere	$S = P - \frac{1}{2}r^2 e_\infty$	$S^* = P_1 \wedge P_2 \wedge P_3 \wedge P_4$
Plane	$\pi = \mathbf{n} + d e_\infty$	$\pi^* = P_1 \wedge P_2 \wedge P_3 \wedge e_\infty$
Circle	$C = S_1 \wedge S_2$	$C^* = P_1 \wedge P_2 \wedge P_3$
Line	$L = \pi_1 \wedge \pi_2$	$L^* = P_1 \wedge P_2 \wedge e_\infty$
Point pair	$Pp = S_1 \wedge S_2 \wedge S_3$	$Pp^* = P_1 \wedge P_2$

Basis Blades are the elementary algebraic elements of Geometric Algebra. Conformal Geometric Algebra consists of basis blades with **grades** 0 to 5, where a scalar is a **0-blade** (a blade of grade 0) and the **1-blades** are the basis vectors. The **2-blades** $e_i \wedge e_j$[1] are basis blades spanned by two 1-blades, and so on. There exists only one element of the maximum grade 5. It is therefore also called the **pseudoscalar**. A linear combination of k-blades is called a k-vector (or a vector, bivector, trivector. ...). The sum $e_2 \wedge e_3 + e_1 \wedge e_2$, for instance, is a bivector. A linear combination of blades with different grades is called a **multivector**. Multivectors are the general elements of a Geometric Algebra. Table 10.1 of [2] shows the 32 basis blades of Conformal Geometric Algebra, consisting of the scalar, five (basis) vectors, ten bivectors, ten trivectors, 5 quadvectors and the pseudoscalar together with the indices used for GAALOP.

Conformal Geometric Algebra provides a great variety of basic geometric entities to compute with, namely points, spheres, planes, circles, lines, and point pairs, as listed in Table 1.1. These entities have two algebraic representations: the IPNS (inner product null space) and the OPNS (outer product null space). These representations are dual to each other (a superscript asterisk denotes the dualization operator). In Table 1.1, the vectors $\mathbf{x}$ and $\mathbf{n}$ are in bold type to indicate that they represent 3D entities obtained by linear combinations of the 3D basis vectors e_1, e_2, and e_3:

$$\mathbf{x} = x_1 e_1 + x_2 e_2 + x_3 e_3. \tag{1.3}$$

In the OPNS representation, the outer product $\wedge$ indicates the construction of a geometric object with the help of points $\{P_i\}$ that lie on it. A sphere, for instance, is defined by four points $(P_1 \wedge P_2 \wedge P_3 \wedge P_4)$ on its surface. In the IPNS representation, the meaning of the outer product is an intersection of geometric entities. A circle, for instance, is defined by the intersection of two spheres $S_1 \wedge S_2$.

[1] the symbol $\wedge$ means the outer product of Geometric Algebra.

1.2 THE GAALOPScript LANGUAGE

GAALOPScript" is derived from CLUScript [5] and somewhat similar to the C programming language. Just as in C, every program line has to be ended by a semicolon. The advantage of this convention is that a program line can be extended over a number of text lines as well as one program line can consist of several short commands. Comments can be included in the script in the same way as in C.

1.2.1 The main notations

Table 1.2 summarizes the most important notations of GAALOPScript and Table 1.3 shows some notations specific for Conformal Geometric Algebra.

TABLE 1.2 The main notations of GAALOPScript

GAALOPScript	**Meaning**	**Geometric Algebra**
A = ...	assignment to a multivector	
A*B	geometric product	AB
A∧B	outer product of A and B	$A \wedge B$
A.B	inner product of A and B	$A \cdot B$
~A	reverse of A	$\tilde{A}$
1/A	inverse of A	A^{-1}
*A	dual of A	A^*
e1, e2 ...	basis vectors	$e_1, e_2 ... e_n$
//	comment	
?A	explicitely compute multivector A	
:A	visualize multivector A	

TABLE 1.3 Notations of GAALOPScript for Conformal Geometric Algebra

GAALOPScript	**Meaning**	**Conformal Geometric Algebra**
e1, e2, e3	3D basis vectors	e_1, e_2, e_3
e0	origin	e_0
einf	infinity	e_∞

One essential part of GAALOPScripts are assignments of Geometric Algebra computations to multivectors. Please notice that while for the geometric

product no specific symbol is used, in GAALOPScript "*" is needed as a symbol. The operators for the dual and the reverse of multivector A are written in front of A. The inverse of A is expressed as "1/A". As an example, the following listing shows a simple GAALOPScript for the computation of the geometric product of two vectors.

Listing 1.1 GAALOPScript for the geometric product of two vectors.

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

On the left side of an assignment, there is either a scalar or a multivector variable[2] while unknown variables are always assumed to be scalars. In our example, *a*, *b* and *c* are multivectors while *a1*, *a2* and *a3* as well as *b1*, *b2* and *b3* are scalar values.

First, the two multivectors *a* and *b* are computed as a linear combination of the 3D basis vectors. In the last line, the multivector *c* is computed as the geometric product of them. The question mark in front of a multivector indicates that we are interested in the explicit computation of this multivector. This means, that we are here interested in the result of *c*, while *a* and *b* are only intermediate results.

Additionally, the colon in front of a multivector means that this multivector should be visualized. Please refer to Section 1.4 for details about visualizations based on GAALOPScript.

TABLE 1.4 Macros of GAALOPScript for Conformal Geometric Algebra

Macro	**Meaning**
VecN3(x,y,z)	Creates a conformal point with the coordinates x, y, z
createPoint(x,y,z)	Creates a conformal point with the coordinates x, y, z
SphereN3(centre, radius)	Creates a sphere from a given centre and a given radius
SphereN3(cx, cy, cz, radius)	Creates a sphere from given centre coordinates and a given radius
RotorN3(x,y,z,angle)	Creates a rotor, which rotates along an axis (defined by x,y,z) with an angle
TranslatorN3(x,y,z)	Creates a translator, which translates along a vector (defined by x,y,z)
ExtractFirstPoint(pp)	Extracts the first point of a given point pair
ExtractSecondPoint(pp)	Extracts the second point of a given point pair
Dual(mv)	Dualizes a given multivector
Normalize(mv)	Normalizes a given multivector mv

[2] multiple assignments to variables are not allowed

1.2.2 Macros and Pragmas

There are predefined macros in order to simplify the development of GAALOPScripts. Table 1.4 shows a list of GAALOPScript macros for Conformal Geometric Algebra.

TABLE 1.5 Pragmas of GAALOPScript

Pragma	**Meaning**
output	If a question mark is set at the specified variable, only the specified blades are calculated, Ex.; //#pragma output mv 1.0 e3 e1 ∧ e2
onlyEvaluate	Computes a component of a specified multivector only if it is needed for later use. Ex.: //#pragma onlyEvaluate mv1 mv2;

There are macros in order to control the process of the generation of optimized code according to Table 1.5. Please find an example for the use of the output macro in Section 1.5.

1.3 LINE-SPHERE-EXAMPLE

The papers [4] and [1] present an example of a ray tracing application. One part of the corresponding algorithm is the computation whether a ray is intersectiong a (bounding) sphere or not. This can be expressed by the following GAALOPScript[3]

Listing 1.2 *RaySphereIntersection.clu*: GAALOPScript for the computation whether there is an intersection of a ray with a sphere or not.

```
1 S = VecN3(Cx, Cy, Cz) - 0.5*r*r*einf;
2 O = VecN3(Ox, Oy, Oz);
3 L = VecN3(Lx, Ly, Lz);
4 R = *(O ^ L ^ einf);
5 PP = R ^ S;
6 ?hasIntersection = PP.PP;
```

This example computes the sphere S and the ray R through the points O and L (VecN3[4] computes a conformal point based on its 3D coordinates). The intersection of the sphere S and the ray R can easily be expressed with the help of the outer product of these two geometric entities (Fig. 1.1). The result is the point pair PP. In the script we call its norm $hasIntersection$,

[3] here, all geometric objects are described in the IPNS representation according to Table 1.1.

[4] VecN3(x,y,z) is a macro according to Table 1.4

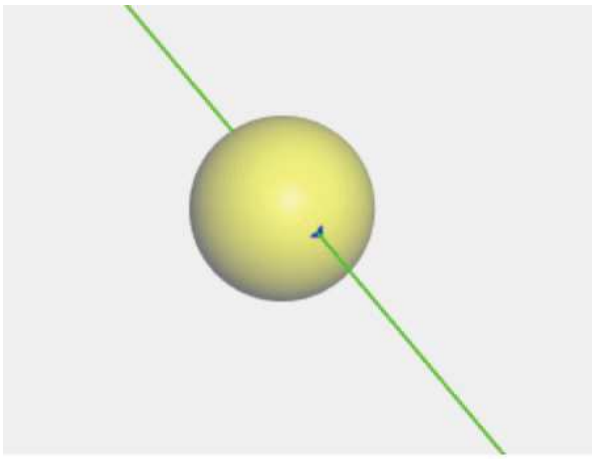

FIGURE 1.1 Spheres and lines are basic entities of Geometric Algebra that one can compute with. Intersection of these objects are easily expressed with the help of their outer product. Here, one of the two points of the intersection of the sphere and the line is shown in blue.

since its sign indicates whether the ray and the sphere are really intersecting each other or not.

A question mark at the beginning of a line indicates a multivector variable that has to be explicitly computed[5].

1.4 VISUALIZATIONS BASED ON GAALOPScript

A GAALOPScript including visualization is defined by three parts

1. Variable assignments
2. Code to optimize
3. Multivectors to be visualized

Listing 1.3 *VisRaySphereIntersection.clu*: GAALOPScript for the visualization of an intersection of a ray with a sphere.

```
1 // Variable assignments
```

[5]GAALOP is able to optimize not only single statements, but a number of Geometric Algebra statements. In the above script, the expressions for S, R, O, L and PP are used only by GAALOP, in order to compute an optimized result for the intersection indicator (see the question mark in the last line of the script)

```
2  Cx =1;  Cy=1; Cz=1;
3  r=0.4;
4  Ox =1;  Oy=1.1; Oz=1;
5  Lx =1;  Ly=0.5; Lz=0.5;
6
7  // Code to optimize
8  S =  VecN3(Cx, Cy, Cz) - 0.5*r*r*einf;
9  O = VecN3(Ox, Oy, Oz);
10 L = VecN3(Lx, Ly, Lz);
11 R = *(O ^ L ^ einf);
12 PP = R ^ S;
13 ?hasIntersection = PP.PP;
14
15 // Multivectors to be visualized
16 :Green;
17 :R_OPNS =*R;
18 :Yellow;
19 :S_OPNS =*S;
20 :Blue;
21 :PP_OPNS =*PP;
```

In the first part, all input variables needed for the general algorithm (code to optimize) are defined. In the last part, all multivectors to be visualized are defined together with the corresponding colours.

1.5 EXAMPLE: CENTER OF A SPHERE

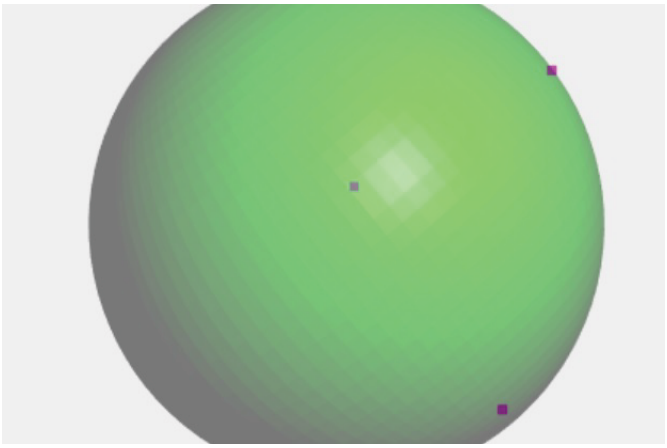

FIGURE 1.2 Visualization of the Listing 1.4.

The following Listing 1.4 computes the sphere S defined by the four points

$P1, P2, P3, P4$ and visualizes the points in magenta and the sphere in green (see Figure 1.2).

Listing 1.4 GAALOPScript for the visualization of a sphere defined by 4 points.

```
P1x=0; P1y=0; P1z=1;
P2x=0; P2y=1; P2z=0;
P3x=1; P3y=0; P3z=0;
P4x=0; P4y=0.701; P4z=0.701;

P1 =  createPoint(P1x, P1y, P1z);
P2 =  createPoint(P2x, P2y, P2z);
P3 =  createPoint(P3x, P3y, P3z);
P4 =  createPoint(P4x, P4y, P4z);

S= P1^P2^P3^P4;

:Magenta;
:P1;
:P2;
:P3;
:P4;
:Green;
:S;
```

The macro createPoint (see Table 1.4) computes a conformal point based on its 3D coordinates. The sphere S is computed based on the outer product of the 4 points according to Table 1.1. The GAALOPScript according to Listing 1.5 focuses on the computation of the sphere and its center point.

Listing 1.5 GAALOPScript for the computation of the center of a sphere defined by 4 points.

```
P1 =  P1x*e1 + P1y*e2 + P1z*e3 + P1i*einf +e0;
P2 =  P2x*e1 + P2y*e2 + P2z*e3 + P2i*einf +e0;
P3 =  P3x*e1 + P3y*e2 + P3z*e3 + P3i*einf +e0;
P4 =  P4x*e1 + P4y*e2 + P4z*e3 + P4i*einf +e0;

S= P1^P2^P3^P4;
//#pragma output C e1 e2 e3
?C= Normalize(S);
```

Here, the points $P1, P2, P3, P4$ are computed according to Table 1.1, assuming that its e_∞- component is already known. If a sphere is normalized[6], its

[6] this is computed based on the predefined macro Normalize according to Table 1.4

e_1, e_2, e_3-components describe the center point of the sphere [7], means the other coefficients do not have to be computed. This is defined by the macro output according to Table 1.5.

[7] see, for instance, Sect. 5.6 of "Introduction to Geometric Algebra Computing" [3]

Bibliography

[1] Hugo Hadfield, Dietmar Hildenbrand, and Alex Arsenovic. Gajit: Symbolic optimisation and jit compilation of geometric algebra in python with gaalop and numba. *Proceedings of CGI conference Calgary, Canada*, 2019.

[2] Dietmar Hildenbrand. *Foundations of Geometric Algebra Computing*. Springer, 2013.

[3] Dietmar Hildenbrand. *Introduction to Geometric Algebra Computing*. Taylor & Francis Group, 2019.

[4] Dietmar Hildenbrand, Justin Albert, Patrick Charrier, and Christian Steinmetz. Geometric algebra computing for heterogeneous systems. *Advances in Applied Clifford Algebras Journal*, 2016.

[5] Christian Perwass. *Geometric Algebra with Applications in Engineering*. Springer, 2009.