Competitive runtime performance for inverse kinematics algorithms using conformal geometric algebra

Dietmar Hildenbrand[1] , Daniel Fontijne[2] , Yusheng Wang[1] , Marc Alexa[3] and Leo Dorst[2]

[1]Darmstadt University of Technology, Germany
[2]University of Amsterdam, The Netherlands
[3]TU Berlin, Germany

Abstract

Conformal geometric algebra is a powerful tool to find geometrically intuitive solutions. We present an approach for the combination of compact and elegant algorithms with the generation of very efficient code based on two different optimization approaches with different advantages, one is based on Maple, the other one is based on the code generator Gaigen 2. With these results, we are convinced that conformal geometric algebra will be able to become fruitful in a great variety of applications in Computer Graphics.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Three-Dimensional Graphics and Realism]: Animation I.3.5 [Computational Geometry and Object Modelling]: Geometric algorithms, languages, and systems

1. Introduction

For the animation of humanoid models, inverse kinematics (IK) solutions are important as a basic building block for path planning. The standard model for arms (and also legs) is a seven 7 DOF kinematic chain, with 3 degrees of freedom $(\theta_1, \theta_2, \theta_3)$ at the shoulder, 1 degree of freedom at the elbow θ_4 and 3 degrees of freedom at the wrist $(\theta_5, \theta_6, \theta_7)$. The current standard tool for solving the inverse problem of mapping from a given end effector state to the configuration space $\{\theta_i\}$ is due to Tolani, Goswami, and Badler [TGB00]. They also discuss in detail less favorable, optimization-based solutions. The importance of their algorithm in computer graphics and animation can be seen from the large number of uses and citations of their work (just to give a few recent examples [SLGS01, ST03, BB04, SHP04]). Our analytic solution to the inverse problem in conformal geometric algebra is an improvement of [HBCZE05] (please refer also to [BCZE04], [Hil05], [RS05] and [WCL05]), and its derivation is considerably simpler than in affine or projective geometry. Perhaps more importantly for the prospective user, our approaches also turns out to be more than 3 times faster.

The goal of our inverse kinematics algorithm is to compute the output parameters q_s and q_e based on the input parameters (see Table 1). In our improved algorithm we benefit from the fact that quaternions are part of conformal geometric algebra. We present two different optimization approaches with different advantages, one is based on Maple, the other one is based on the code generator Gaigen 2.

Table 1: *input/output parameters of the inverse kinematics algorithm*

parameter	Maple / Gaigen 2	meaning
p_w	pw(pwx, pwy, pwz)	target point of wrist
θ	sangle	swivel angle
L_1, L_2	L1, L2	length of the arm
q_s	qs	shoulder quaternion
q_e	qe	elbow quaternion

2. Optimization with Maple

We use Maple in order to get the most elementary relationship between the input and output parameters of our inverse kinematics algorithm (see Table 1).

The most important feature of Maple is the symbolic calculation [MAP]. In order to deal with the computation of conformal geometric algebra, we use a library called Cliffordlib, developed by Rafal Ablamowicz and Bertfried Fauser. For download and installation hints please refer to [CLI05, Wan06]. For information about conformal geometric algebra please refer to the tutorial [HFPD04]. The most important operations of the Clifford package are presented in table 2. For the inner product we use the left contraction (LC) operation. Besides these main operations we also need

Notation	Meaning
`a &c b`	geometric product
`a &w b`	outer product
`LC(a,b)`	inner product
`-(a) &c e12345`	dualization
`reversion()`	reversion

Table 2: *Notation of GA-Operations*

some methods like `scalarpart()` or `vectorpart()` for extracting the scalar or the vector part of a multivector. In order to use conformal geometric algebra computation we have to load the Clifford package, set the metric of Clifford algebra, set aliases to basic blades (optional) and define e_0 and e_∞.

```
> with(Clifford);
> B:=linalg[diag](1, 1, 1, 1, -1);
> eval(makealiases(5, "ordered"));
> e0:=-0.5*e4+0.5*e5;
> einf:=e4+e5;
```

Here, we present an improvement of the algorithm of [HBCZE05] for the first 4 DOF as well as the corresponding Maple code :

- Compute the elbow point p_e

```
> pw:=pwx*e1+pwy*e2+pwz*e3+0.5*
       (pwx^2+pwy^2+pwz^2)*einf+e0;
> S1:=pw-0.5*L2*L2*einf;
> S2:=e0-0.5*L1*L1*einf;
> C_e:=S1 &w S2;
> l_sw:=-(e0 &w pw &w einf)&c e12345;
> pi_swivel:=-(pe2 &w pw &w e0 &w einf)
             &c e12345;
> norm_l_sw:=sqrt(l_sw &c reversion(l_sw));
> q_swivel:=cos(sangle/2)+sin(sangle/2)
            *(l_sw / norm_l_sw);
> pi_swivel:=q_swivel &c pi_swivel
             &c reversion(q_swivel);
> PP:=-(C_e &w pi_swivel) &c e12345;
> PP:=vectorpart(PP,2);
> einf_PP:=LC(einf,PP);
> norm_einf_PP:=einf_PP &c
                  reversion(einf_PP);
> inv_einf_PP:=einf_PP/norm_einf_PP;
> p_e:=-(-sqrt(scalarpart(LC(PP,PP)))
       +PP) &c inv_einf_PP;
> p_e:=vectorpart(p_e,1);
```

The main difference to the [HBCZE05] algorithm equations (10) to (20) is the interpretation of the rotor of equation (11) as a quaternion. It can be shown that in conformal geometric algebra lines through the origin represent pure quaternions with the imaginary units $i = \mathtt{e}_3 \wedge \mathtt{e}_2, j = \mathtt{e}_1 \wedge \mathtt{e}_3, k = \mathtt{e}_2 \wedge \mathtt{e}_1$. The quaternion q_{swivel} describing the rotation of the swivel plane can be expressed as $q_{swivel} = \cos(\frac{\theta}{2}) + \sin(\frac{\theta}{2})\frac{l_{sw}}{|l_{sw}|}$ with the normalized line l_{sw}.

- Compute the quaternion q_e at the elbow joint
  For efficiency reasons we do not compute the angle Θ_4 explicitly based on the *arccos* function (see equations (21) and (22) of [HBCZE05]) but immediately compute the elbow quaternion q_e based on the formulae for $cos(\frac{\theta}{2})$ and $sin(\frac{\theta}{2})$.

```
> l_se:=-(e0 &w p_e &w einf)&c e12345;
> l_ew:=-(p_e &w pw &w einf)&c e12345;
> c4:=-LC(l_se,l_ew)/(L1*L2)/Id;
> qe:=sqrt((1+c4)/2)+sqrt((1-c4)/2)*(-qi);
```

- Rotate until the elbow position matches
  For efficiency reasons we combine the steps 3 and 4 of [HBCZE05] in one improved step as follows :

```
> p_ze:=L1*e3+0.5*L1^2*einf+e0;
> pi_m:= p_ze-p_e;
> pi_e:=-(p_ze &w p_e &w e0 &w einf)&c e12345;
> l_m:=pi_e &w pi_m;
> q12:=l_m/(sqrt(l_m &c reversion(l_m)));
```

We directly compute a quaternion that rotates an object from one point $p_1 = p_{ze}$ to another point $p_2 = p_e$, both points having the same distance to the origin (see Figure 1). First, we calculate the middle line l_m between the two points through the origin. In conformal geometric algebra, a middle plane of two points is described by their difference (see [PH04]). We calculate l_m with the help of the intersection of this plane and the plane through the origin and the points p_1 and p_2. Second, in order to rotate from

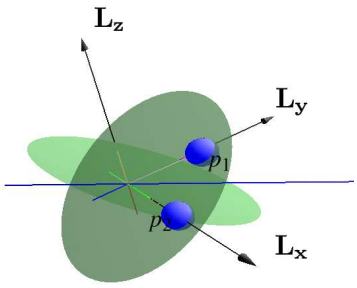

Figure 1: *Rotation based on the line between two points through the origin*

p_1 to p_2 we have to rotate about the middle line with radius π. If the line is normalized, this results in a quaternion identical to the normalized line. In Figure 1 the two points are indicated by two spheres. We can see the line between

two points through the origin and the two planes for its computation. The quaternion q_{12} describes the rotation at shoulder joint which lets the robot elbow reach its target p_e:

- Compute the quaternion q_s at the shoulder joint
  The quaternion q_s will let the robot wrist reach the given target pw:

```
> pi_yz2:=q_12 &c e1 &c reversion(q_12);
> _sign:=scalarpart(LC(pw,pi_yz2));
> _sign:=_sign/abs(_sign);
> norm_pi_swivel:=sqrt(pi_swivel &c
                  reversion(pi_swivel));
> c3:=scalarpart(-LC(pi_yz2,pi_swivel))
      /(norm_pi_swivel);
> q3:=sqrt((1+c3)/2)+sqrt((1-c4)/2)
      *_sign*qk;
> qs:=q12 &c q3;
```

  The quaternions q_s and q_e are the required results of our algorithm.

With the help of Maple our conformal geometric algebra formulae are simplified and combined to very efficient expressions because of the symbolic computation feature of Maple. For instance, for the first lines of the algorithm we get a result as follows for the intersection circle

```
C_e = 0.5*(1-L1^2)*(pwx*e15+pwy*e25+pwz*e35)-

      0.5*(1+L1^2)*(pwx*e14+pwy*e24+pwz*e34)+
      0.5*e45*(pwx^2+pwy^2+pwz^2+L1^2-L2^2)
```

with only some simple multiplications and additions.

3. Optimizations with Gaigen 2

The philosophy behind Gaigen 2 is based on two ideas : generative programming and specializing for the structure of conformal geometric algebra. Gaigen 2 takes a succinct specification of a geometric algebra and transforms it into an implementation. The resulting implementation is very similar to what someone would program by hand and can be directly linked to an application.

In many types of programs, each variable has a fixed 'specialized' multivector type. I.e., the inverse kinematics algorithm uses variables of with multivector types like *line* and *sphere*. If the GA implementation could work directly with these leaner specialized multivector types, performance would be greatly increased.

As an implementation of this insight, Gaigen 2 allows the user to define specialized types along with the algebra specification, and generates classes for each of them. These specialized multivector classes require much less storage than the generic multivector, but as a result, they are of course unable to store an arbitrary multivector type. For example, a *line* variable can not be stored in a *sphere* variable.

The below description of the first part of the algorithm (see item Compute the elbow point p_e at page 2) is offered as an explicit example of how one can think in geometry, and directly program in geometric elements.

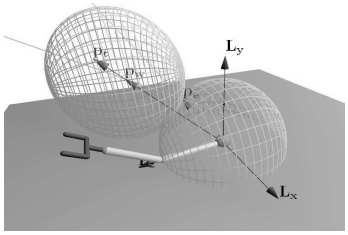

Figure 2: *compute the elbow point*

We implement the computation of the elbow point in Gaigen 2 as follows:

```
Sphere s1 = _Sphere(pw-
0.5f* L2* L2*einf); // einf means e∞

originSphere s2 = _originSphere(e0-
0.5f* L1* L1*einf);

Circle c_e = _Circle(s1^s2);
```

With the help of the two spheres s1, s2 with center points p_w (target point of the wrist) and e_o (shoulder located at the origin) and radii L_2, L_1 we are able to compute the circle determining all the possible locations of the elbow as the intersection of the spheres.

For the needed geometric objects we use the following specializations:
Both, the target point p_w as well as the sphere s_1 are assigned to a multivector type called `Sphere` (please remember that a point in conformal geometric algebra is simply a sphere with 0 radius). Type `Sphere` is defined as follows:

```
specialization:

blade Sphere(e0=1, e1, e2, e3, einf);
```

Type `Sphere` means a linear combination of basis blades with the coefficient of e_0 being 1.

The center of the sphere s_2 is the origin e_0. We use the type `originSphere`:

```
specialization:

blade originSphere(e0, einf);
```

since we can see that `e0-0.5f* L1* L1*einf` only needs the blades `e0` and `einf`. The result of the intersection of the spheres $C_e = s_1 \wedge s_2$ is of type `Circle` being a bivector.

4. Results

First we developed and simulated our algorithm visually based on CLUCalc (see [Per05]), developed by Christian Perwass. Then, we had to implement it on the target platform

Avalon ([ZGD]), a virtual reality system written in C++ using Visual Studio.NET 2003. Previously, inverse kinematics was implemented using IKAN [TGB00], a widely used C++ library.

Our Maple approach outperformed the IKAN implementation clearly. It turned out to be about 3.3 times faster. Based on this approach we are able to design and test our algorithms on a high level. When we are satisfied with our algorithm we are able to transfer it into the C/C++ language without the need of additional libraries.

Also our Gaigen 2 approach outperformed the IKAN in a similar way. The conformal geometric algebra based algorithm is 43 % faster than IKAN, and even 240 % faster when the conversion from matrices to quaternions is taken into account (this has to be done in our application, because the rotations must be represented as quaternions in order to perform SLERP operations on them, while the conformal geometric algebra based algorithm delivers quaternions by default). Based on this approach we are able to design and test our algorithms on a high level as well as implement them in a way that still reflects the elegant features of conformal geometric algebra.

5. Conclusion

Algorithms based on conformal geometric algebra are geometrically intuitive and easy to understand. But, often, better structure and greater elegance comes at the prize of lower run time performance. The approaches of this paper show that we are now able to get algorithms that can even be much faster than conventional ones. Because of their compactness conformal geometric algebra algorithms are easy to implement and process in Maple and Gaigen 2.

The two approaches have different advantages: Based on the Gaigen 2 approach we are able to implement our algorithms in a way that still reflects the elegant features of conformal geometric algebra. Based on the Maple approach we are able to implement them with the help of our standard compilers without the need of additional libraries.

With these results, we are convinced that conformal geometric algebra will be able to become fruitful in a great variety of applications in Computer Graphics.

References

[BB04] BAERLOCHER P., BOULIC R.: An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. *The Visual Computer 20*, 6 (2004), 402–417.

[BCZE04] BAYRO-CORROCHANO E., ZAMORA-ESQUIVEL J.: Inverse kinematics, fixation and grasping using conformal geometric algebra. In *IROS 2004, September 2004, Sendai, Japan* (2004).

[CLI05] The homepage of the package cliffordlib. HTML document http://math.tntech.edu/rafal/cliff9/, 2005. Last revised: September 17, 2005.

[HBCZE05] HILDENBRAND D., BAYRO-CORROCHANO E., ZAMORA-ESQUIVEL J.: Advanced geometric approach for graphics and visual guided robot object manipulation. In *proceedings of ICRA conference, Barcelona, Spain* (2005).

[HFPD04] HILDENBRAND D., FONTIJNE D., PERWASS C., DORST L.: Tutorial geometric algebra and its application to computer graphics. In *Eurographics conference Grenoble* (2004).

[Hil05] HILDENBRAND D.: Geometric computing in computer graphics using conformal geometric algebra. *Computers & Graphics 29*, 5 (2005), 802–810.

[MAP] The homepage of maple. http://www.maplesoft.com/products/maple. 615 Kumpf Drive, Waterloo, Ontario, Canada N2V 1K8.

[Per05] PERWASS C.: The CLU home page. HTML document http://www.clucalc.info, 2005.

[PH04] PERWASS C., HILDENBRAND D.: *Aspects of Geometric Algebra in Euclidean, Projective and Conformal Space*. Tech. rep., University of Kiel, 2004.

[RS05] ROSENHAHN B., SOMMER G.:. *Journal of Mathematical Imaging and Vision 22* (2005), 27–70.

[SHP04] SAFONOVA A., HODGINS J. K., POLLARD N. S.: Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 514–521.

[SLGS01] SHIN H. J., LEE J., GLEICHER M., SHIN S. Y.: Computer puppetry: An importance-based approach. *ACM Transactions on Graphics 20*, 2 (Apr. 2001), 67–94.

[ST03] SMINCHISESCU C., TRIGGS B.: Kinematic jump processes for monocular 3d human tracking. In *2003 Conference on Computer Vision and Pattern Recognition (CVPR 2003)* (June 2003), pp. 69–76.

[TGB00] TOLANI D., GOSWAMI A., BADLER N. I.: Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical Models 62*, 5 (Sept. 2000), 353–388.

[Wan06] WANG Y.: *Algorithm and performance of the grasping movement of a human-arm-like chain based on Conformal Geometric Algebra*. Master's thesis, TU Darmstadt, 2006.

[WCL05] WAREHAM R., CAMERON J., LASENBY J.: Applications of conformal geometric algebra in computer vision and graphics. *Lecture Notes in Computer Science 3519* (June 2005), 329–349.

[ZGD] ZGDV: The Avalon home page. HTML document http://www.zgdv.de/avalon/.