A Conformal Geometric Algebra code generator comparison for Virtual Character Simulation in Mixed Reality

Margarita Papaefthymiou, Dietmar Hildenbrand and George Papagiannakis

Abstract. Over the last few years, recent advances in user interface and mobile computing, introduce the ability to create new experiences that enhance the way we acquire, interact and display information within the world that surrounds us with virtual characters. Virtual Reality (VR) is a 3D computer simulated environment that gives to user the experience of being physically present in real or computer-generated worlds; on the other hand, Augmented Reality (AR) is a live direct or indirect view of a physical environment whose elements are augmented (or supplemented) by computer-generated sensory inputs. Both technologies use interactive devices to achieve the optimum adaptation of the user in the immersive world achieving enhanced presence, harnessing latest advances in computer vision, glasses or head-mounted-displays featuring embedded mobile devices. A common issue in all of them is interpolation errors while using different linear and Quaternion algebraic methods when a) tracking the user's position and orientation (translation and rotation) using computer vision b) tracking using mobile sensors c) tracking using gesture input methods to allow the user to interactively edit the augmented scene (translation, rotation and scale) d) having animation blending of the virtual characters that augmented the mixed reality scenes (translation and rotation). In this work, we propose an efficient method for robust authoring (rotation) of Augmented reality scene using Euclidean Geometric Algebra (EGA) rotors and we propose two fast animation blending methods using GA and CGA. We also compare the efficiency of different GA code generators: a) Gaigen library, b) libvsr and c) Gaalop using our animation blending methods and compare them with other alternative animation blending techniques: a) Quaternions and b) Dual-Quaternions, so that a future user of GA libraries can choose the most appropriate one that will give the most optimal and faster results.

Mathematics Subject Classification (2010). Primary 97R60.

Keywords. Geometric Algebra, Conformal model, Augmented Reality, Animation Blending, Animation.

1. Introduction

In this work, we aim to enhance the Conformal Geometric Algebra (CGA)[2, 3, 11] as the mathematical background for display and character animation control [12] in immersive and virtual technology [1], such as head-mounted displays (e.g. Google CardboardTM) or modern smartphones; a framework that offers a smooth and stable calibration/control can be used in real-time mobile mixed reality systems that featured realistic, animated virtual human actors who augmented real environments. GA is a mathematical framework that provides a convenient mathematical notation for representing orientations and rotations of objects in three dimensions, a compact and geometrically intuitive formulation of algorithms, and an easy and immediate computation of rotors; CGA extends the usefulness of the 3D GA by expanding the class of rotors to include translations, dilations and inversions. Rotors are simpler to manipulate than Euler angles, more numerically stable and more efficient than rotation matrices, avoiding the problem of Gimbal lock. This work allows us to a) propose a method that handles rotations with GA rotors that avoids the problem of Gimbal Lock, b) blend rotations and translations between character animations using CGA under a single geometric algebraic framework for Mixed Reality applications c) unify previously separated linear and Quaternion Algebra for rotation interpolation with fast CGA rotors, d) compare the performance of different GA code generators so that a future user of GA libraries can choose the most appropriate one that will give the most optimal and faster results and e) compare the performance of GA and CGA animation blending methods with Quaternions and Dual-Quaternions interpolation.

In computer graphics are used many alternative methods to represent rotations. The most common way is by using linear algebra. The angle of rotation is defined on each axis and converted to transformation matrices, each one represents rotation around x, y and z axis and then are multiplied together to extract the final rotation. A way for representing rotation combined with translation is Dual-Quaternions.

The most important properties when developing applications especially on mobile devices are robustness and efficiency. The transformations must not contain any discontinuities and be smooth. Also, they should consume the smallest necessary amount of memory and be computationally fast. Gimbal lock is a common problem arising when using euler angles and can be solved using GA rotors. Gimbal lock is the loss of one degree of freedom in 3D space which occurs when the axes of two of the three gimbals are driven into a parallel configuration and this leads to unexpected behaviour.

In this work, we propose a method for robust authoring (rotation) of Augmented reality scene using EGA rotors. Moreover, we present two fast animation blending methods using 3D EGA and CGA. We implemented these methods using Gaalop Precompiler (CLUCalc scripting language), GA Gaigen library and libvsr and we compare their efficiency with Dual-Quaternions and Quaternions methods. On Figure 1 is shown our AR application.

1.1. Problem statement and main novelty

Our work, presents an efficiency comparison between three different GA code generators which are a) GA Gaigen library, b) libvsr and c) Gaalop precompiler, in order to provide to the audience that is interested in using GA code generators information about their performance. We also propose an efficient method for authoring of rotation of the Augmented reality scene using EGA rotors and we show that GA gives smooth results compared to euler angles that cause Gimbal Lock. Moreover, we propose two animation blending methods using EGA and CGA and we show that CGA approach implemented with Gaalop precompiler can give very fast results similar to Dual-Quaternions. However, the benefit of using CGA is that supports rotation, translation and dilation in a single representation compared to Dual-Quaternions that support only rotation and translation.

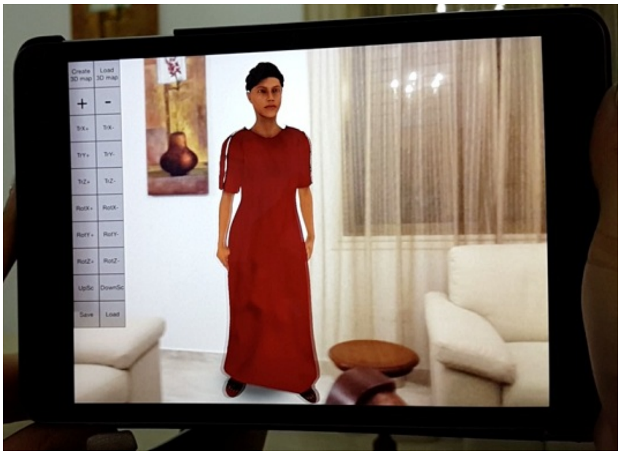

Figure 1. The AR application.

2. Previous work

In recent years, Geometric Algebra and Conformal Geometric Algebra mathematical tools are used in many areas of Computer Science and Computer Engineering such as Computer Vision, Computer Graphics and Robotics.

Dorst et al. [4] presents applications of GA and CGA in the field of Computer Graphics and provides useful examples written in C++ using GA Gaigen library. Some of these applications are interpolating rotations, recursive ray-tracing for illumination, constructing Binary Space partition (BSP) trees, handling rotations with rotors and handling intersections for collision detection and shadows.

In [12, 13] propose two alternative methodologies for implementing real-time Animation Interpolation for skinned characters using GA rotors. They compare their methodology with alternative animation blending techniques such as Quaternion Linear Blending and Dual-Quaternion slerp interpolation and show that they achieve smaller computation time, lower memory usage and more visual quality results. Moreover, Wareham et al. [9] proposes a method for pose and position interpolation using CGA which can also be extended to higher-dimension spaces.

[3, 14] applied Conformal Geometric Algebra for inverse kinematics solvers. Aristidou et al. [3] proposed a fast methodology called FABRIK, for solving the IK problem of a 3D human hand, which can also be used for other IK applications. This algorithm uses a forward and backward iterative methodology, to extract each joint position by locating a point on a line, and integrates rotation and orientation constraints. Hildenbrand et al. [14] presents an algorithm for inverse kinematics by generating code using two different optimizations, the first one is based on Maple and the second one on Gaigen2. Wareham et al. [10] propose an algorithm for interpolating between two or more displacements which combine both translation and rotation to a unique representation which produces smooth results.

3. AR Scene Authoring

We handle rotations of the AR scene objects by replacing euler angles with EGA rotors. GA rotors are simpler to manipulate than euler angles, more numerically stable and more efficient than rotation matrices. Moreover, GA rotors do not produce discontinuities to the rotation, by avoiding the problem of Gimbal Lock. Our main novelty, lies in the replacement of euler angles with fast and robust GA rotors while the user rotates the objects of the AR scene. This algorithm was developed using Gaigen2 C++ code generator and GA Gaigen library.

3.1. Algorithm description

As a first step, we set the initial rotation of the scene on each axis in euler angles representation and we compute the current GA rotor. We convert euler angles to Quaternion representation and then compute the angle and axis of the Quaternion in order to calculate the current GA rotor using the exponential Formula:

$$R = e^{-I_3 u \frac{\phi}{2}} \tag{3.1}$$

where I_3 is the pseudoscalar, ϕ is the angle of rotation and u is the axis of rotation.

The code that computes the initial rotor of the AR scene given the euler angles is provided on Section A.3. Our implementation gives the ability to the user to rotate the scene on global axis or on object's local axis.

When we rotate on local axis we rotate the GA basis vectors with the current GA rotor (R_{cur}) in order to define the new local coordinate system. We rotate the coordinate system by sandwiching the 3D EGA basis vectors

(e_1, e_2, e_3) between the current GA rotor and its inverse rotor. For example, to rotate the basis vector e_1 we use the Formula below:

$$rot\_e_1 = R_{cur} e_1 R_{cur}^{-1}$$

as deduced from:

$$RAR^{-1} \tag{3.2}$$

where A is a multivector and R is a rotor.

In this way, we compute the new rotated basis vectors rot_e_1, rot_e_2, rot_e_3 which are used to define the new planes of rotation (bivectors) by constructing the outer product $\wedge$ between the new rotated basis vectors which are: $rot\_e_1 \wedge rot\_e_2$, $rot\_e_2 \wedge rot\_e_3$ and $rot\_e_3 \wedge rot\_e_1$ on x, y, z axis respectively. When we change the axis of rotation we need to define the new local coordinate system. When rotating on the same axis we need to change the angle of rotation, recompute the current rotor and multiply it with the previous rotor. We compute the current rotor using Equation 3.3 as deducted from Equation 3.1.

$$R = e^{v\frac{\phi}{2}} \tag{3.3}$$

where v is the plane of rotation and ϕ is the angle of rotation. Section A.5 provides the code to rotate on X local axis and Section A.4 shows how to convert GA rotor to matrix representation. On Figure 2 is shown the rotation of a character on local X axis in our AR application.

In contrast, when we rotate in global coordinate system, the planes of rotation are defined by constructing the outer product between the GA basis vectors (e_1, e_2, e_3) i.e. $e_1 \wedge e_2$, $e_2 \wedge e_3$ and $e_3 \wedge e_1$.

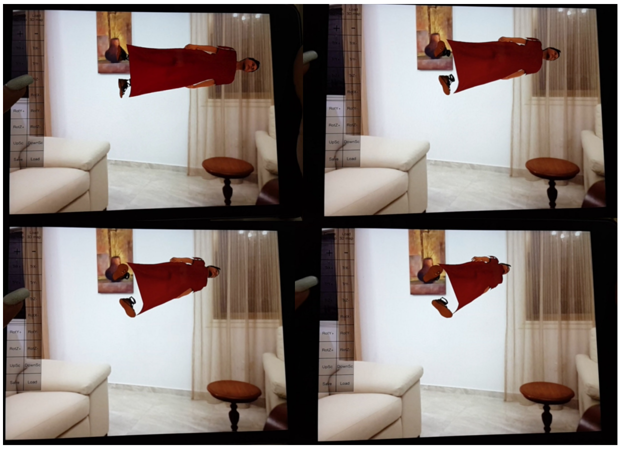

FIGURE 2. Rotating a character on X local axis with GA rotors.

4. Animation Blending

We have developed Animation Blending using two fast alternative methods. The first one is by using EGA rotors and the second one CGA motors. On the first method, we fully replace Quaternions for rotation interpolation with EGA rotors and on the second one, we blend rotations and translations for character animation using fast CGA motors. For optimization purposes we precompute Quaternions to rotors representation. We have developed GA and CGA approach with Gaigen2 C++ code generator (GA Gaigen library), CLUCalc (Gaalop precompiler) and libvsr.

4.1. Gaalop Precompiler

We use Gaalop Precompiler [5] in order to have optimized Geometric Algebra code and more efficient results. We have used the visualization tool for GA CLUCalc v4.3 [8] in order to produce our algorithms and then Gaalop standalone application to generate the C++ optimized code. Gaalop precompiler provides three different approaches of optimization: GAPP (Geometric Algebra Parallelism Program), Maple and Table-Based Approach. For generating the optimized C++ code we have chosen Table-Based Approach with the symbolic-computation tool Maxima [7] support and we have enabled optOneExpressionRemoval, optConstantPropagation, optUnusedAssignments and optInserting in the Configuration tab of the standalone application. The Table-Based approach uses precomputed multiplication tables to generate code that does not include any Geometric Algebra operations.

4.2. Algorithm description

Our main novelty, lies in the employment of CGA motors as fast, drop-in replacements for Quaternion Algebra, during animation blending for skinned characters. On the first approach, we represent rotation with GA rotors and on the second approach, we represent rotation combined with translation with CGA motors. Then we use GA and CGA exponential formulas to interpolate between the two keyframes of the character animation. For the CGA approach, we also provide an alternative way for interpolation using linear interpolation of motors. On Figure 3, is illustrated the animation interpolation with CGA logarithm interpolation approach using Gaigen library on a character with walk animation.

4.2.1. GA rotors approach. Firstly, we convert the rotation of the two keyframes to GA rotors. We compute the angle and the axis of the Quaternions and extract the GA rotors using the Equation 3.1. Section A.3 shows how to convert Quaternion to GA rotor representation.

After expressing the source and destination rotation to GA rotors we compute the interpolated rotor base on a factor number that defines the animation interpolation step. We compute the rotor R from source (R_{src}) to destination (R_{dst}) rotor using the Equation 4.1. To compute the final rotor (R_{final}), we use the logarithm formula to interpolate the rotor R in N steps as given by the Equation 4.2.

On Table 1 and Section A.1 we provide the implementation of GA approach with CLUCalc and libvsr respectively. For more details for this approach refer to [13] which also, provides the implementation using GA Gaigen library.

$$R = R_{src}^{-1} R_{dst} \tag{4.1}$$

$$R_{final} = R_{src} e^{log(R)*N} \tag{4.2}$$

```
axis = srcX*e1 + srcY*e2 + srcz*e3;
axis = axis/sqrt(axis.axis);
p = *axis;
R=exp(-0.5*angle*p);

axis2 = dstX*e1 + dstY*e2 + dstZ*e3;
axis2 = axis2/sqrt(axis2.axis2);
p2 = *axis2;

R2=exp(-0.5*angle2*p2);

RtotQ = ~R*R2;
rotInterpolated = exp(factor*RtotQ);
finalR= R*rotInterpolated;
```

Table 1. CLUCalc implementation for Animation Blending with GA rotors approach.

4.2.2. CGA motors approach. A a first step we convert Quaternion and translation of the two keyframes to CGA motors as shown on Section A.6 The translator part of the CGA motor is computed using the Equation:

$$T = e^{-\frac{1}{2} t e_{\infty}} \tag{4.3}$$

where t is the vector that represents translation:

$$t = t_1 e_1 + t_2 e_2 + t_3 e_3 \tag{4.4}$$

We compute the rotor with the same way as the GA approach as described in Section 4.2.1. Motor is computed by multiplying Rotor (R) and Translator (T) as given by the following Equation:

$$M = RT \tag{4.5}$$

After expressing the source and destination rotation-translation to CGA motors we compute the interpolated motor base on a factor (f) number that defines the animation interpolation step.

In CLUCalc implementation we compute the interpolated motor from source (M_{src}) to destination (M_{dst}) motor using linear interpolation (Equation 4.6). In order to convert the CGA motor to matrix representation we compute the images of basis blades $e_1 \wedge n_\infty$, $e_2 \wedge n_\infty$, $e_3 \wedge n_\infty$ and $n_o \wedge e_\infty$ as proposed in [4].

$$M = M_{src} * (1 - f) + M_{dst} * f \tag{4.6}$$

Concerning GA Gaigen library and libvsr we interpolate using logarithms. We compute the motor from source to destination motor using the Equation 4.7. Then, we use the logarithm formula to interpolate between the motors in N steps as described in Equation 4.8. Table 2 shows how to compute the interpolated CGA motor. In order to convert the CGA motor to matrix representation we compute images of basis blades $e1 \wedge n_\infty$,$e2 \wedge n_\infty$,$e3 \wedge n_\infty$ and $no \wedge e_\infty$ as proposed in [4].

$$M = M_{src}^{-1} M_{dst} \tag{4.7}$$

$$M_{final} = M_{src} e^{log(M)*N} \tag{4.8}$$

On Table 2 and on section A.2 we provide the implementation for CGA motors using CLUCalc and libvsr respectively. Section A.7 shows how to compute the interpolated CGA motor.

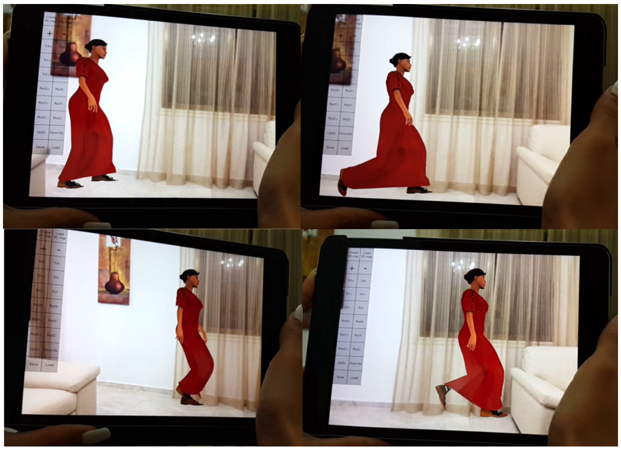

FIGURE 3. Animation Interpolation with CGA motors using GA Gaigen library.

5. Implementation details

The main framework used for our AR application is the open source OpenGL Geometric Application (glGA) [15, 16] framework. glGA is a lightweight,

```
tr=tr1X*e1+tr1Y*e2+tr1Z*e3;
trNew=1-0.5*tr*einf;
a=VecN3(axis1X,axis1Y,axis1Z);
axiss = *(a^VecN3(0,0,0)^einf);
axis=axiss/abs(axiss);
R = Exp_approx(-angle/2*axis);

tr2=tr2X*e1+tr2Y*e2+tr2Z*e3;
trNew2=1-0.5*tr2*einf;
a2=VecN3(axis2X,axis2Y,axis2Z);
axiss2 = *(a2^VecN3(0,0,0)^einf);
axis2=axiss2/abs(axiss2);
R2 = Exp_approx(-angle2/2*axis2);

motor1=trNew*R;
motor2=trNew2*R2;

m = motor1*(1-alpha) + motor2*alpha;

x=m*e1^einf/m;
y=m*e2^einf/m;
z=m*e3^einf/m;
t=m*e0^einf/m;
```

Table 2. CLUCalc implementation for Animation Blending with CGA motors approach.

shader based C++ Computer Graphics (CG) framework which is developed for educational as well as research purposes. glGA is a cross platform application development framework and supports many mobile and desktop platforms. glGA contains many operations like compiling and loading shaders, textures, sounds, animations, Image Based Lighting, loading 3D static meshes as well as rig meshes. In order to help the students visualize the externally rigged virtual characters (e.g. Collada or MD5 models) glGA provides the functionality required to parse the bone tree in real-time and retrieve the transformation matrix from each one of the joints. These matrices are then passed as uniform and vertex attribute parameters to the vertex shader.

In glGA framework is integrated the MetaioSDK [15] framework. This framework is responsible for the AR functionalities and can perform markerless SLAM-based 3D camera tracking.

We have developed glGA in such a way so that all of its examples and sample assignments can run in any of the standard desktop and mobile platforms: Windows, Linux, OSX and iOS. In order for all of those (10 in total) applications to be supported, we had to create a short Platform-Wrapper component that handles the platform specific functionality.

In addition to the desktop platforms of Windows, Linux and OSX, the glGA examples are also supported in the mobile iOS platform. Here is where the Platform-Wrapper is also employed not only due to the header files but also due to the different OpenGLES methods and calls (instead of standard OpenGL). An additional difference between desktop and mobile platforms is the way that external assets (e.g. textures, 3D models etc.) are loaded by the application. E.g. iOS uses bundles, while Windows, Linux and Mac retrieve the assets directly from the disk with either relative or absolute paths. Other than these, the current time retrieval is also different from desktop to mobile. E.g. it is essential during character animation, where we have to recalculate the matrix transformations based on the time passed since the animation started. As we have already, mentioned the code of the examples and assignments is in portable, standard C++, thus a standard C++ compiler (e.g. gcc, LLVM, Intel or Microsoft) is mandatory to be employed. In glGA, we have also included some IDE project files for certain platforms already set up and ready to be built and executed. The project files that exist in glGA are for Visual Studio 2010 for Windows and Xcode 7.2 for Mac and iOS, while we also provide the basic gcc/g++ makefiles for Linux. The only modification required is to define the specific platform on top of the Platform-Wrapper header file. Of course, other IDEs can also be used as long as they support standard C++.

6. Results

In this section, we compare our GA and CGA animation interpolation approaches with Quaternions and Dual-Quaternions and provide efficiency comparison of the different GA code generators. We obtained our results on a MacBook Pro with processor 2.5 GHz Intel Core i7 and an NVIDIA Geforce GT 750M 2048 MB graphics card. We applied all the methodologies on 3 different characters/animations. Our characters are of dae format and consist of 43-54 joints and 4881-135976 triangles.

Libvsr headers generate optimized code at compile-time through template metaprogramming and Gaalop precompiler generates C++ optimized code that does not contain any GA operations. For that reason, Libvsr and Gaalop give faster results than Gaigen library. Furthermore, concerning GA approaches, Gaalop with CGA gives the most fast results because linear interpolation is faster than interpolating using logarithms. Table 3 provides the average time needed to execute animation interpolation on all joints of each of the 3 characters illustrated on Figure 4. Our results show that Dual-Quaternions are the most efficient method for animation blending. Moreover, CGA (especially CGA Gaalop) is almost as efficient in terms of performance with Dual-Quaternions but superior in compact, efficient, inclusive representation of composition of transformations (rotation, translation, dilation).

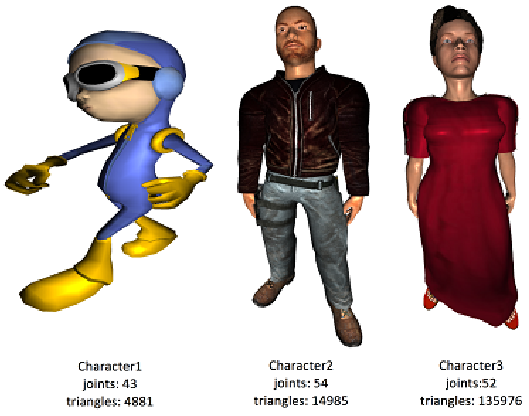

FIGURE 4. The three characters used to compare the animation blending methods.

Method	Character1	Character2	Character3
Quaternions	0.00044	0.00179	0.00108
Dual-Quaternions	0.00024	0.0016	0.00092
GA Gaigen	0.00131	0.00250	0.00162
CGA Gaigen	0.00182	0.00289	0.00229
GA Gaalop	0.00155	0.00282	0.00201
CGA Gaalop	0.00034	0.00177	0.00115
GA Versor	0.00076	0.00201	0.00157
CGA Versor	0.00085	0.00209	0.00174

TABLE 3. Average time in milliseconds (msecs) for each animation blending method for the three characters of Figure 3.

7. Conclusions and Future work

In this work, we achieved more efficient and robust AR scene authoring that avoids Gimbal lock by handling rotations of the AR objects with GA rotors. Moreover, we focused on implementing two fast methods for animation blending for skinned characters using EGA and CGA models. On the first approach we represent rotations with GA rotors and on the second approach we express translations and rotations with CGA motors. We interpolate using only the logarithm of rotors/motors respectively. For CGA motors we also interpolate using linear interpolation. We implemented our animation blending methods using GA Gaigen library, generating C++ code using Gaalop precompiler for more optimized code and libvsr. We compared our animation blending approaches with Quaternions and Dual-Quaternions techniques. Our results show a) that Dual-Quaternions and CGA approach implemented with Gaalop have similar efficiency and b) libvsr and Gaalop generate more efficient results compared to GA Gaigen library. We also compared the efficiency of different GA code generators so that a future user of GA libraries can choose the most appropriate one that will give the most optimal and faster results.

In the future, we aim to avoid conversions between different data types in order to achieve more efficient results. We will use rotors directly in shader programs to avoid converting from rotor to matrix representation. Moreover, we aim to extend our CGA framework by applying GA for global illumination and specifically, for rotating spherical harmonics for Precomputed Radiance Transfer for real-time rendering.

Acknowledgments

The research leading to these results has received funding from the European Union People Programme (FP7-PEOPLE-2013-ITN) under grant agreement n^o 608013.

References

[1] Egges, A., Papagiannakis, G., and Magnenat-Thalmann, N. (2007). Presence and interaction in mixed reality environments. Visual Computer 23, 5, 317333.

[2] Hestens, D., Sobczyk, G. (1984). Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics. Reidel, Dordrecht.

[3] Aristidou, A., Lasenby, J. (2011). Inverse Kinematics solutions using Conformal Geometric Algebra, In L. Dorst and J. Lasenby (Eds), Guide to Geometric Algebra in Practice, Springer Verlag.

[4] Dorst, L., Fontijne, D., and Mann, S. (2010). Geometric Algebra for Computer Science. Morgan Kaufmann.

[5] Hildenbrand, D. (2013). Foundations of Geometric Algebra Computing.

[6] Perass, C. (2009). Geometric Algebra with Applications in Engineering.

[7] Maxima Development Team. Maxima, a computer algebra system. version 5.36.1. Available at http://maxima.sourceforge.net/, 2015.

[8] Christian Perwass. The CLU home page. Available at http://www.clucalc.info,2010.

[9] Wareham, R., Cameron, J. and Lasenby, J. (2005). Applications of Conformal Geometric Algebra in Computer Vision and Graphics. In Proceedings of the 6th International Conference on Computer Algebra and Geometric Algebra with Applications, Shanghai, China 2005

[10] Wareham, R. and Lasenby, J. (2008). Mesh Vertex Pose and Position Interpolation Using Geometric Algebra, Articulated Motion and Deformable Objects, 5098, 122-131

[11] Kanatani, K. Understanding Geometric Algebra: Hamilton, Grassmann, and Clifford for Computer Vision and Graphics. A K Peters/CRC Press, 2015.

[12] Papagiannakis, G., Greasidou, E., Trahanias, P. and M. Tsioumas. A geometric algebra animation method for mobile augmented reality simulations in digital heritage sites. The Computer Journal, pages 258267, 2014.

[13] Papagiannakis, G. Geometric algebra rotors for skinned character animation blending. In SIGGRAPH Asia 2013 Technical Briefs, SA 13, pages 11:111:6, New York, NY, USA, 2013. ACM.

[14] Hildenbrand, D., Fontijne, D., Wang, Y., Alexa, M. and Dorst, L. Competitive runtime performance for inverse kinematics algorithms using conformal geometric algebra. In Eurographics 2006 - Short Presentations, Vienna, Austria, September 4-8, 2006, pages 58, 2006.

[15] Papagiannakis, G., Papanikolaou, P., Greasidou, E. and Trahanias, P. glga: an opengl geometric application framework for a modern, shader-based computer graphics curriculum. Eurographics 2014, pages 18, 2014.

[16] Papaefthymiou, M., Feng, A., Shapiro, A. and Papagiannakis, G. A fast and robust pipeline for populating mobile AR scenes with gamified virtual characters. In SIGGRAPH Asia 2015 Mobile Graphics and Interactive Applications, SA '15, pages 22:122:8, New York, NY, USA, 2015. ACM.

Appendix A. GA code for AR scene authoring and Animation Blending

A.1. GA animation blending approach using libvsr

```
glm::mat4 interpolateGAVersor(Rot srcR, Rot dstR, float Factor){
   Rot srcdst = !srcR * dstR;

    Rot log =  Gen::log(srcdst);

    Rot result = srcR * Gen::mot(log * Factor);

    Flp FlatPoint2 = result * (Vec::x^Inf(1)) * !result;
    Flp FlatPoint3 = result * (Vec::y^Inf(1)) * !result;
    Flp FlatPoint4 = result * (Vec::z^Inf(1)) * !result;

    glm::mat4 final(1.0);
    final[0][0]=FlatPoint2.val[0]; final[0][1]=FlatPoint2.val[1];
    final[0][2]=FlatPoint2.val[2]; final[0][3]=0;
    final[1][0]=FlatPoint3.val[0]; final[1][1]=FlatPoint3.val[1];
    final[1][2]=FlatPoint3.val[2]; final[1][3]=0;
    final[2][0]=FlatPoint4.val[0]; final[2][1]=FlatPoint4.val[1];
    final[2][2]=FlatPoint4.val[2]; final[2][3]=0;
    }
```

A.2. CGA animation blending approach using libvsr

```
Mot interpolateCGAVersor(vec3 translation1, vec3 translation2,
                         Rot srcR, Rot dstR, float Factor){
    const Drv srcTr(translation1.x, translation1.y, translation1.z);
    Drv dstTr(translation2.x, translation2.y, translation2.z);

    Trs srcT=Gen::trs(srcTr);
    Trs dstT=Gen::trs(dstTr);

    Mot mot = srcT * srcR;
    Mot mot2 = dstT * dstR;

    Mot srcdst = !mot * mot2;
    Mot log =  Gen::log(srcdst);

    Mot motor = mot * Gen::mot(log * Factor);

    return motor;
}
```

A.3. Euler angles to GA rotor representation

```
  rotor EulerToRotor(vec3 euler){
    destQ = quat(euler);
    float angleDest=angle(destQ);
    vec3 axisDest=axis(destQ);
    mv v=unit_e(axisDest.x*e1+axisDest.y*e2+axisDest.z*e3);
    return _rotor(exp((angleDest)/2*(-I3*v)));
  }
```

A.4. Convert GA rotor to matrix representation

```
mat4 rotorToMat4(mv rotor){
    quat quat(_float(rotor),rotor.e1e2(),
             rotor.e3e1(), rotor.e2e3());
    return mat4_cast(quat);
}
```

A.5. Rotate on X local axis with GA rotors

```
void rotateXLocal(float angle,int& prevRot, mv &rotor,
                  mv& newe1,mv& newe2,mv& newe3,mat4 &finalRotation){
    //set current rotation and rotate coordinate system
    if (prevRot != ROTATIONX){
        prevRot = ROTATIONX;
        rotateCoordinateSystem(rotor, newe1, newe2, newe3);
    }
    mv plane=newe1^newe2; // define plane on rotation
    mv Rsrc=_rotor(exp(plane*(angle/2.0)));
    rotor=Rsrc*rotor;
    finalRotation = rotorToMat4(rotor);
}
```

A.6. Convert translation and rotation to CGA motor representation

```
  mat4 ToMotor(vec3 translation,quat quaternion){
    srcQ=normalize(srcQ);
    float angleSrc=angle(srcQ);
    vec3 axisSrc=axis(srcQ);
    destQ=normalize(destQ);
    float angleDest=angle(destQ);
    vec3 axisDest=axis(destQ);
    //angle in radians
    mv rotor =exp(angleSrc/2.0*(-I3 * (uC)));
    mv uC=unit_e(axisSrc.x*e1+axisSrc.y*e2+axisSrc.z*e3);
    vectorE3GA t=_vectorE3GA(translation.x*e1+ translation.y
                             *e2+translation.z*e3);
    normalizedTranslator TC=exp(_freeVector(-0.5f*(t^ni)));
    //construct motor(translation and rotation)
    TRversor TRC = _TRversor(TC*rotor);
    return TRC;
}
```

A.7. Compute the interpolated CGA motor for Animation blending

```
mat4 interpolateCGA(TRversor TRC,TRversor TRD,float Factor){
    TRSversor versor=_TRSversor(TRC*exp(_dualLine(Factor*
                    log(_TRSversor(inverse(TRC)*TRD)))));
    TRSversor iVersor = inverse(versor);
    return makeMat4(_flatPoint(versor*e1ni*iVersor),
                   _flatPoint(versor*e2ni*iVersor),
                   _flatPoint(versor*e3ni*iVersor),
                   _flatPoint(versor*noni*iVersor));
}
```

Margarita Papaefthymiou
Institute of Computer Science, Foundation for Research and Technology Hellas and Computer Science Department,University of Crete,
Heraklion, Greece
e-mail: margarita@csd.uoc.gr

Dietmar Hildenbrand
Hochschule RheinMain,
Wiesbaden, Germany
www.gaalop.de
e-mail: dietmar.hildenbrand@gmail.com

George Papagiannakis
Institute of Computer Science, Foundation for Research and Technology Hellas and Computer Science Department,University of Crete,
Heraklion, Greece
e-mail: papagian@ics.forth.gr