# Never 'Drop the Ball' in the Operating Room: An Efficient Hand-Based VR HMD Controller Interpolation Algorithm, for Collaborative, Networked Virtual Environments

Manos Kamarianakis[1,2](✉), Nick Lydatakis[1,2], and George Papagiannakis[1,2]

[1] University of Crete, Heraklion, Greece
[2] ORamaVR, Heraklion, Greece
{manos,nick,george.papagiannakis}@oramavr.com
http://www.oramavr.com

**Abstract.** In this work, we propose two algorithms that can be applied in the context of a networked virtual environment to efficiently handle the interpolation of displacement data for hand-based VR HMDs. Our algorithms, based on the use of dual-quaternions and multivectors respectively, impact the network consumption rate and are highly effective in scenarios involving multiple users. We illustrate convincing results in a modern game engine and a medical VR collaborative training scenario.

**Keywords:** Interpolation · Keyframe generation · Geometric algebra

## 1 Introduction

Collaborative, shared virtual environments (CVEs) are among the most researched and developed areas of the last decades [1,13,16,18]. The growing need of remote networked communication, further accelerated by the ongoing pandemic, have resulted in great leaps in technological advancements. Head-mounted displays (HMDs) are now capable of supporting intensive resource-demanding Virtual Reality (VR) applications. To further facilitate this support, powerful algorithms are being developed and optimized by VR specialists (Fig. 1).

Part of this research revolves around the efficient relay of synchronized, networked information from the HMD to the VR engine that is responsible for the rendering of the scene [19]. This information typically involves user interactions through the HMD controllers such as *displacement data* (e.g., translation and rotation of the controller) within specific time intervals and button-press events.

Specifically, when the user moves the hand-based controllers of his HMD, the hardware initially detects the movement type and logs it, in various time


N. Magnenat-Thalmann et al. (Eds.): CGI 2021, LNCS 13002, pp. 694–704, 2021.
https://doi.org/10.1007/978-3-030-89029-2_52

(a) (b) (c)

**Fig. 1.** Catching a tool in a VR collaborative scenario. (a) A user throws a tool (in our case a medical drill) at another. (b) The object's keyframes, sent by the user that threw it, are interpolated using multivector LERP (see Sect. 4.2) on the receiver's VR engine. (c) The receiver manages to catch the tool, as a result of the effective frame generation that is visualized in his/her HMD. This example is just to illustrate extreme hand-based interpolation in collaborative, networked virtual environments and is provided for illustration only. To better understand the significance of this figure, please watch the paper's presentation found in [9]. DO NOT TRY THIS AT HOME.

intervals based on the user's or developer's preferences. This logged movement, that is either a translation and/or a rotation, is constantly transcoded into a suitable format and relayed to the VR application and rendered as a corresponding action, e.g., hand movement, object transformation or some action. The controller's data format to be transmitted to the rendering engine affects the overall performance and quality of experience (QoE) and poses challenges that must be addressed. These challenges involve keeping the latency between the movement of the controller and its respective visualization in the HMD below a certain threshold that will not break the user's immersiveness. Furthermore, the information must be relayed efficiently such that a continuous movement of the controller results in a smooth jitter-less outcome in the VR environment. Such challenges heavily depend on the implementation details regarding the communication channel that handles the way that position and rotation of the controller is relayed, as well as the choice of a suitable interpolation technique. The displacement data are transmitted at discrete time intervals, approximately 20–40 times per second. To maintain a high frame-per-second scenery in the VR, multiple in-between frames must be created on-the-fly by the appropriate tweening algorithm. An efficient algorithm will allow the generation of natural flow frames while requiring fewer intermediate keyframes. Such algorithms will help reduce a) bandwidth usage between the HMD and the rendering engine and b) CPU-strain, resulting in lower energy consumption as well as lower latency issues in bandwidth-restricted networks. Moreover, HMDs with controllers of limited frequency will still be able to deliver the same results as more expensive HMDs.

The current state-of-the-art methods regarding the format used to transmit the displacement data mainly involves the use of 3D vectors for translation and quaternions for rotation data. These representation forms are dominant due to the fact that they involve very few bytes to be represented (3 and 4 respectively) and the fact that they support fast and efficient interpolations. Specifically, 3D vectors are usually linearly interpolated where as the SLERP method is usually used for quaternion blending. In some engines, such as Unity3D, rotations are sometimes provided in terms of Euler angles, but for interpolation needs, they are internally transformed to their quaternion equivalents.

**Our Contribution.** In this work, we propose the use of geometric algebra (GA) as a means to encapsulate the positional and rotational data of the hand-based VR HMD controllers and to generate the intermediate frames in the rendering engine. Our idea aims to take advantage of the fact that basic geometric entities used in VR, such as points, planes, lines, translations, rotations and dilations (uniform scalings), can be uniformly represented as *multivectors*, i.e., elements of a suitable geometric algebra such as 3D Projective (3D PGA) or 3D Conformal Geometric Algebra (3D CGA). Algebras such as 3D PGA and 3D CGA are showing rapid adaptation to VR implementations due to their ability to represent the commonly used vectors, quaternions and dual-quaternions natively as multivectors. In fact, quaternions and dual-quaternions are contained as a sub-algebra in both these algebras [7]. Therefore, they incorporate all benefits of quaternions and dual-quaternions representation such as artifact minimization in interpolated frames [11]. Furthermore, geometric algebras enable powerful geometric predicates and modules within an all-in-one framework [10], providing, if used with caution, performance which is on par with the current state-of-the-art frameworks [14]. We illustrate convincing results in a modern game engine and a medical VR collaborative training scenario (see the video presentation of this work [9] and Fig. 2).

**Fig. 2.** Images taken from a modern VR training application that incorporates our proposed interpolation methods for all rigid object transformations as well as hand and avatar movements. It is recommended to see the video presentation of this work [9], to better understand the significance of these figures.

## 2 State of the Art

The current state-of-the-art for representing the controller's displacement are 3D vectors for the positional data and quaternions for the rotational data. Regarding the position, the controllers log their current position $v = (v_x, v_y, v_z)$ at each time step with respect to a point they consider as the origin. Their rotation is stored as a *unit* quaternion $q = (q_x, q_y, q_z, q_w) = q_x\boldsymbol{i} + q_y\boldsymbol{j} + q_z\boldsymbol{k} + q_w$, i.e., it holds that $q_x^2 + q_y^2 + q_z^2 + q_w^2 = 1$. The use of unit quaternions revolutionized graphics as it provided a convenient, minimal way to represent rotations, while avoiding known problems (e.g., gimbal lock) of other representation forms such as Euler angles [11]. The ways to change between unit quaternions and other forms representing the same rotation, such as rotation matrices and Euler angles, are summarized in [2].

The interpolation of the 3D vectors containing the positional data is done linearly, i.e., given $v$ and $w$ vectors we may generate the intermediate vectors $(1 - a)v + aw$, for as many $a \in [0, 1]$ as needed. Given the unit quaternions $q$ and $r$ the intermediate quaternions are evaluated using the SLERP blending, i.e., we evaluate $q(q^{-1}r)^a$, for as many $a \in [0, 1]$ as needed, like before. If these intermediate quaternions are applied to a point $p$, the image of $p$, as $a$ goes from 0 to 1, has a uniform angular velocity around a fixed rotation axis, which results in a smooth rotation of objects and animated models.

## 3 Room for Improvements

The current state for representing and interpolating positional and rotational data is based on the use of 3D vectors and quaternions as the main VR rendering engines, Unity3D and Unreal Engine, have the respective frameworks already built in. Graphics courses worldwide mention quaternions as the next evolution step of Euler angles; a step that simplified things and amended interpolation problems without adding too much overhead in the process. Despite it being widespread, the combined use of vectors and quaternions does not come without limitations.

A drawback that often arises lies in the fact that the simultaneous linear interpolation of the vectors with the SLERP interpolation of the quaternions applied to rigid objects does not always yield smooth, natural looking results in VR. This is empirically observed on various objects, depending on the movement the user *expects* to see when moving the controllers. Such *artifacts* usually require the developer's intervention to be amended, usually by demanding more intermediate displacements from the controller to be sent, i.e., by introducing more non-interpolated keyframes. This results mainly in the increase of bandwidth required as more information must be sent back and forth between the rendering engine and the input device, causing a hindrance in the networking layer. Multiplayer VR applications, that heavily rely on the input of multiple users on the same rendering engine for multiple objects, are influenced even more, when such a need arises. Furthermore, the problem is intensified if the rendering application resides on a cloud or edge node; such scenarios are becoming increasingly more common as they are accelerated by the advancements of 5G networks and the relative functionalities they provide.

# 4 Proposing New Approaches

## 4.1 Proposed Method Based on Dual Quaternions

In the past few years, graphics specialists have shown that dual quaternions can be a viable alternative and improvement over quaternions, as they allow us to unify the translation and rotation data into a single entity. Dual quaternions are created by quaternions if dual numbers are used instead of real numbers as coefficients, i.e., they are of the form $d := A + \epsilon B$, where $A$ and $B$ are ordinary quaternions and $\epsilon$ is the *dual unit*, an element that commutes with every element and satisfies $\epsilon^2 = 0$ [12]. A subset of these entities, called *unit dual quaternions*, are indeed isomorphic to the transformation of a rigid body. A clear advantage of using dual quaternions is the fact that we only need one framework to maintain and that applying the encapsulated information to a single point requires a simple sandwich operator. Moreover, the rotation stored in the unit dual quaternion $A + \epsilon B$ can be easily extracted as the quaternion $r := A$ is the unit quaternion that amounts to the same rotation. Furthermore, if $B^\star$ denotes the conjugate quaternion of $B$, then $t := 2AB^\star$ is a pure quaternion whose coefficients form the translation vector [12].

Taking advantage of the above, we propose the replacement of the current state-of-the-art sequence (see Fig. 3, Top) with the following (see Fig. 3, Middle). The displacement data of an object is again represented as a vector and a quaternion; in this way, only a total of 7 float values (3 and 4 respectively) need to be transmitted. The VR engine combines them in a dual quaternion [12] and interpolates with the previous state of the object, also stored as a dual quaternion. Depending on the engine's and the user's preferences, a number of in-between frames are generated via SLERP interpolation [11] of the original and final data. For each dual-quaternion received or generated, we decompose it to a vector and a quaternion and apply them to the object. This step is necessary to take advantage of the built-in optimized mechanisms and GPU implementations of the VR engine.

A major advantage of the proposed method is that we can obtain similar results with the state-of-the-art method by sending less keyframes per second. As an empirical law, we may send 20 displacement data per second with our method to obtain the same quality of generated frames as if we had sent 30 data per second with the current state-of-the-art method. This 33% reduction of required data applies for each user of the VR application, greatly lowering the bandwidth required as more users join. As an example, if $n$ users participate, the total displacement data required for our method would be $1120n$ bytes per second (20 messages per second X 7 floats per message X 8 bytes per float, assuming a classic implementation) as opposed to $1680n$ bytes per second (20 messages per second X 7 floats per message X 8 bytes per float) with the default method. The numbers of updates per second mentioned above relate to the case of unrestricted-bandwidth network; for the respective results regarding constrained networks Sect. 5 and Table 1. The performance boost of our method

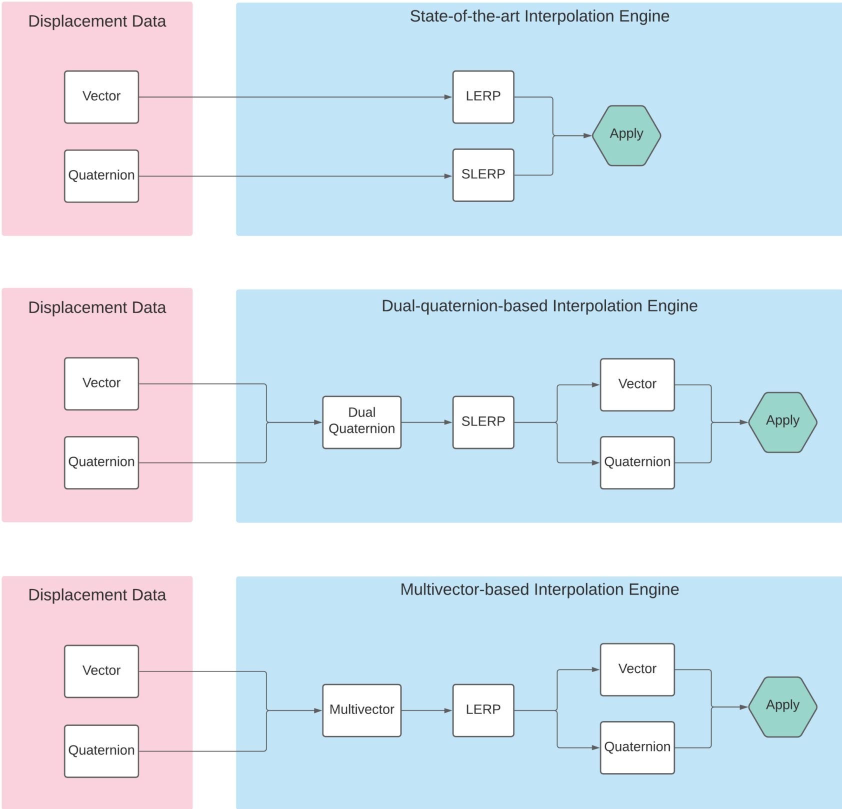


**Fig. 3.** Algorithm layout of the different interpolation engines used to generate intermediate frames.

is further validated as it is used in the MAGES SDK [17] for cooperative VR medical operations.

The drawbacks of this method is the need to constantly transform dual-quaternions to vector and rotation data after every interpolation step but this performance overhead is tolerable as the extraction of the displacement data is accomplished in a straight-forward way. Also, performing SLERP on a dual quaternion (proposed method) instead of a quaternion (state-of-the-art method) demands more operations per step. The trade-offs between the two methods seem to favor our method, especially in the case of collaborative VR applications.

### 4.2 Proposed Method Based on Multivectors

The proposed method described in Sect. 4.1 was based on the use of dual quaternions and the fact that interpolating them (using SLERP) produced smooth intermediate frames. In this section, we go one step further and suggest the use of multivectors instead of dual-quaternions (see Fig. 3, Bottom). This transition can be done in a straight-forward way if we use multivectors of 3D Conformal (see [7]) or 3D Projective Algebra (see [4] and its updated Chap. 11 in [3]). The interpolation of the resulting multivectors can be accomplished via LERP [6]; if $M_1$ and $M_2$ correspond to two consecutive displacement data, then we can generate the in-between multivectors $(1-a)M_1 + aM_2$, for as many $a \in [0,1]$ as needed (and normalize them if needed). Notice that since we are only applying these displacements to rigid bodies, we may use LERP instead of SLERP (see Fig. 4). For every (normalized) multivector $M$ received or interpolated, we may now extract the translation vector and rotation quaternion. Every multivector received or generated has to be decomposed to a vector and a quaternion in order to be applied to the object, as modern VR Engines natively support only the latter two formats. Assume that $M = T * R$ where $T$ and $R$ are the multivectors encapsulating the translation and rotation, we may extract them depending on the Geometric Algebra used (all products below are geometric unless stated otherwise):

- **3D PGA**: Given $M$, we evaluate $e_0 M$. Since in this algebra $T = 1 - 0.5e_0(t_1e_1 + t_2e_2 + t_3e_3)$, represents the translation by $(t_1, t_2, t_3)$ and $e_0e_0 = 0$, it holds that $e_0M = e_0TR = e_0R$. Therefore, if $e_0Q = e_0R = ae_0 + be_{012} + ce_{013} + de_{023}$, we obtain the multivector $R = a + be_{12} + ce_{13} + de_{23}$ which corresponds to the quaternion $q = a - di + cj - bk$. We can now evaluate $T$ as it equals $MR^{-1} = M(a - be_{12} - ce_{13} - de_{23}) = 1 + xe_{01} + ye_{02} + ze_{03}$ and extract the translation vector $(-2x, -2y, -2z)$.
- **3D CGA**: Given $M$, we obtain $R$ by adding the terms of $M$ that contain only the basis vectors $\{1, e_1, e_2, e_3, e_{12}, e_{23}, e_{13}\}$. This derives from the fact that $T = 1 - 0.5 * (t_1e_1 + t_2e_2 + t_3e_3)(e_4 + e_5)$ (which corresponds to the translation by $(t_1, t_2, t_3)$) and therefore $TR = R + m$ where $m$ necessarily contains basis elements containing $e_4$ and $e_5$ (or their geometric product) that cannot be canceled out. After the obtaining of $R$, we simply evaluate $T = MR^{-1}$, normalize it and extract the translation vector $(t_1, t_2, t_3)$ from the quantity $t = T \cdot (e_5 - e_4) = t_1e_1 + t_2e_2 + t_3e_3$. The conversion of $R$ to quaternions and the evaluation of $R^{-1}$ is identical with the case of 3D PGA above.

The advantage of such a method lies on the fact that we can use LERP blending of multivectors instead of SLERP. This saves as a lot of time and CPU-strain; SLERP interpolation requires the evaluation of a multivector's logarithm, which requires a lot of complex operations [5]. Notice that, LERP is efficient in our case since only rigid objects displacements are transfered via the network; if we wanted to animate skinned models via multivectors it is known that only SLERP can produce jitter-less intermediate frames [11]. Another gain of this proposed method is the ability to incorporate it in an all-in-one GA framework, that will use only multivectors to represent model, deformation and animation data. Such a framework is able to deliver efficient results and embeds powerful modules [10,14,15]. In such frameworks, decomposition of multivectors to vectors and quaternions will be redundant, as we can apply the displacement to the object's multivector form via a simple sandwich operation.

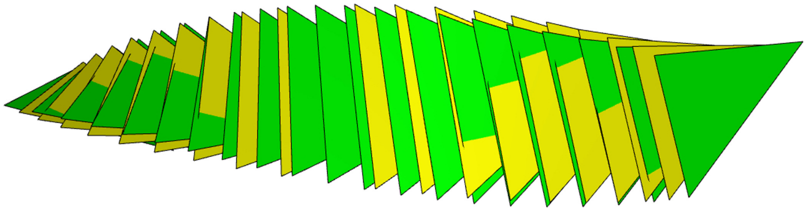

**Fig. 4.** A triangular object is interpolated via multivectors. A motor including both a translation and a rotation is applied to the triangle via its mass center. Between the extreme positions of the object, we generate 20 intermediate frames using LERP (yellow) and SLERP (green) interpolation of the multivector. Only minimal differences are spotted between the two outcomes. (Color figure online)

The trade-offs of such an implementation are based on the fact that modern VR engines do not natively support multivectors and therefore production-ready modules, with basic functions implemented, are almost non-existent. An exception is the Klein C++ module for 3D PGA, found in www.jeremyong.com/klein; for 3D CGA no such module is available the moment this paper is written. This makes it difficult for GA non-experts to adopt and implement such methods. Furthermore, multivectors require 16 (3D PGA) or 32 (3D CGA) float values to be represented and therefore even a simple addition between two amounts to 16 or 32 float operations respectively. Unoptimized modules, usually running in CPU and not in GPU, may result in slow rendering. On the contrary, optimized ones, such as GAALOP [8], can take advantage of the fact that very few of the multivector coordinates are indeed non-zero, as the multivectors involved are always motors, i.e., represent translations and/or rotations, and therefore have specific form.

## 5 Our Results

The methods proposed were implemented in Unity3D and applied to a VR collaborative training scenario. The video accompanying this work demonstrates the effectiveness of our methods compared with the current state of the art. Specifically, we compare the three methods under different input rates per second, i.e., the keyframes sent per second to the VR rendering engine. The input rates tested are 5, 10, 15 and 20 frames per second (fps), where the last option is an optimal value to avoid CPU/GPU strain in collaborative VR scenarios. These rates are indicative values of the maximum possible fps that would be sent in a network whose bandwidth rates from very-limited (5 fps) to unrestricted (more than 20 fps). In lower fps, our methods yield jitter-less interpolated frames compared to the state-of-the-art method, which would require 30 fps to replicate similar output. As mentioned before, this reduction of required data that must be transfered per second by 33%–58% (depending on the network quality, see Table 1) is multiplied by every active user, increasing the impact and the effectiveness of our methods in bandwidth-restricted environments.

**Table 1.** Summary of the metrics of our methods (Ours) versus the state-of-the-art methods (SoA). The first column describes the possible network quality which correlates to the maximum number of updates per second that can be performed. The second column contains the update rate required to obtain the same QoE under the specific network quality limitations. The third column contains the comparison of the bandwidth and the running time difference by our algorithms compared with the SoA algorithm, when using the respective update rates of the second column.

| Network quality | How to achieve best QoE | Metrics on our methods |
|---|---|---|
| Excellent | SoA: 30 updates/sec | 33% less bandwidth |
| | Ours: 20 updates/sec | 16.5% lower running time |
| Good | SoA: 20 updates/sec | 50% less bandwidth |
| | Ours: 10 updates/sec | 16.5% lower running time |
| Mediocre | SoA: 15 updates/sec | 53% less bandwidth |
| | Ours: 7 updates/sec | 16.5% lower running time |
| Poor | SoA: 12 updates/sec | 58% less bandwidth |
| | Ours: 5 updates/sec | 16.5% lower running time |

The workflows of the two algorithms, compared with the current state of the art, are summarized in Fig. 3. In Fig. 5 we demonstrate the interpolation of the same object, at specific time intervals, for all methods; the intermediate frames feel natural for both methods proposed.

In Table 1, it is demonstrated that, under various network restrictions, both proposed methods required less data (in terms of updates per sec) to be transmitted via the network to achieve the same QoE. This decrease in data transfer leads to a lower energy consumption of the HMDs by 10% (on average, preliminary result) and therefore enhances the overall mobility of the devices relying on batteries. Our methods provide a performance boost, decrease the required time to perform the same operation, with fewer keyframes but the same number of total generated frames, by 16.5% (on average). The running times were produced in a PC with a 3,1 GHz 16-Core Intel Core i9 processor, with 32 GBs of DDR4 memory. The same percentage of performance boost is expected in less powerful CPUs; in this case, the overall impact, in terms of absolute running time, will be even more significant.

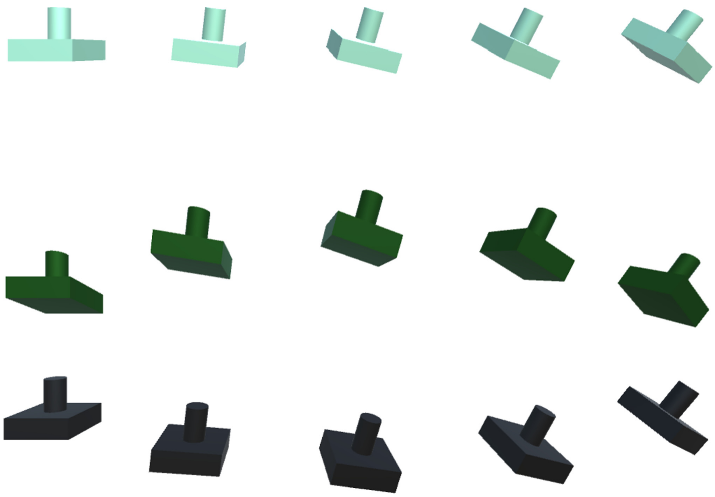

**Fig. 5.** Different interpolation algorithms yield different, yet jitter-less, intermediate frames. (Top): State of the art: Vector and quaternion separate interpolation. (Middle): Dual-quaternion based interpolation algorithm. (Bottom): Multivector based interpolation algorithm.

## 6 Conclusions and Future Work

In this work, we proposed two alternative interpolation algorithms based on dual-quaternions and multivectors respectively. These algorithms can be applied in the context of a networked virtual environment to efficiently handle the interpolation of displacement data for hand-based VR HMDs. The amount of displacement data per second that should be transmitted over the network to support a good QoE can be reduced using our methods instead of the state-of-the-art. This results in a performance boost and also lowers device energy consumption. The significance of our proposed methods are further highlighted in bandwidth-restricted networks and when multiple users are involved. Our results are illustrated in a modern game engine and a medical VR collaborative training scenario.

The proposed algorithms and results can be further improved by using optimized C# Geometric Algebra bindings (such as the ones provided in bivector.net). This would allow for efficient SLERP for the multivector interpolation engine and therefore unlock the potential to apply motors for rigged model animation in VR, as in [15]. It is our intention to integrate the algorithms proposed to an all-in-one GA framework that also enables features such as cut, tear and drill, as in [10].

**Acknowledgments.** This work was co-financed by European Regional Development Fund of the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH - CREATE - INNOVATE (project codes: T1EDK-01149 and T1EDK-01448). The project also received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871793.

## References


1. Churchill, E.F., Snowdon, D.: Collaborative virtual environments: an introductory review of issues and systems. Virtual Reality **3**(1), 3–15 (1998)
2. Diebel, J.: Representing attitude: euler angles, unit quaternions, and rotation vectors. Matrix **58**(15–16), 1–35 (2006)
3. Dorst, L.: A guided tour to the plane-based geometric algebra pga. https://bivector.net/PGA4CS.html
4. Dorst, L., Fontijne, D., Mann, S.: Geometric algebra for computer science - an object-oriented approach to geometry. The Morgan Kaufmann series in computer graphics (2007)
5. Dorst, L., Valkenburg, R.: Square root and logarithm of rotors in 3d conformal geometric algebra using polar decomposition. In: Guide to Geometric Algebra in Practice, pp. 81–104. Springer, London (2011). https://doi.org/10.1007/978-0-85729-811-9_5
6. Hadfield, H., Lasenby, J.: Direct linear interpolation of geometric objects in conformal geometric algebra. Adv. Appl. Clifford Algebras **29**(4), 1–25 (2019). https://doi.org/10.1007/s00006-019-1003-y
7. Hildenbrand, D.: Foundations of geometric algebra computing. Springer (2013)
8. Hildenbrand, D., Pitt, J., Koch, A.: Gaalop-high performance parallel computing based on conformal geometric algebra. In: Geometric Algebra Computing, pp. 477–494. Springer (2010)
9. Kamarianakis, M., Lydatakis, N., Papagiannakis, G.: Video presentation of the paper 'Never Drop the Ball' (2021). https://youtu.be/xoXrRU-2gLQ
10. Kamarianakis, M., Papagiannakis, G.: An all-in-one geometric algorithm for cutting, tearing, drilling deformable models. arXiv preprint arXiv:2102.07499 (2021)
11. Kavan, L., Collins, S., Žára, J., O'Sullivan, C.: Geometric skinning with approximate dual quaternion blending. ACM Trans. Graph. **27**(4), 105 (2008)
12. Kenwright, B.: A beginners guide to dual-quaternions: What they are, how they work, and how to use them for 3D character hierarchies. In: WSCG 2012 - Conference Proceedings, pp. 1–10. Newcastle University, United Kingdom, December 2012
13. Molet, T., et al.: Anyone for tennis? Presence: Teleoperators Virtual Environ. **8**(2), 140–156 (1999)
14. Papaefthymiou, M., Hildenbrand, D., Papagiannakis, G.: An inclusive Conformal Geometric Algebra GPU animation interpolation and deformation algorithm. Vis. Comput. **32**(6–8), 751–759 (2016)
15. Papagiannakis, G.: Geometric algebra rotors for skinned character animation blending. In: SIGGRAPH Asia 2013 Technical Briefs, SA 2013, December 2013
16. Papagiannakis, G., Singh, G., Magnenat-Thalmann, N.: A survey of mobile and wireless technologies for augmented reality systems. Comput. Animation Virtual Worlds **19**(1), 3–22 (2008)
17. Papagiannakis, G., et al.: Mages 3.0: Tying the knot of medical vr. In: ACM SIGGRAPH 2020 Immersive Pavilion. Association for Computing Machinery (2020)
18. Ruan, J., Xie, D.: Networked vr: State of the art, solutions, and challenges. Electronics **10**(2), 166 (2021)
19. Vilmi, O.: Real-time Multiplayer Software Architecture. Bachelor thesis, Metropolia University of Applied Sciences, March 2020