Research Article

BIOMEDICAL Journal of Scientific & Technical Research

ISSN: 2574 -1241

DOI: 10.26717/BJSTR.2021.38.006201

Fast GPU Interpolation for Medical Robotics Using the Conformal Geometric Algebra Framework

Gerardo Martinez Teran, Oswaldo Urena-Ponce, Gerardo Soria Garcia, S Ortega-Cisneros and Eduardo Bayro Corrochano*

Department of Electrical Engineering and Computer Science, Mexico

***Corresponding author:** Eduardo Bayro Corrochano, Department of Electrical Engineering and Computer Science, Mexico

ARTICLE INFO

Received: September 01, 2021

Published: September 15, 2021

Citation: Gerardo Martinez Teran, Oswaldo Urena-Ponce, Gerardo Soria Garcia, S Ortega-Cisneros, Eduardo Bayro Corrochano. Fast GPU Interpolation for Medical Robotics Using the Conformal Geometric Algebra Framework. Biomed J Sci & Tech Res 38(5)-2021. BJSTR. MS.ID.006201.

ABSTRACT

Interpolating trajectories of points and geometric entities is an im- portant problem for kinematics. To describe these trajectories, several algorithms have been proposed using matrices, quaternions, dual-quaternions, and the Study quadric; the last one allows the embedding of motors as 8D vectors into projective space P7, where the interpolation of rotations and translations becomes a linear problem. Furthermore, conformal geometric algebra (CGA) is an effective and intuitive framework for representing and manipulating geometric entities in Euclidean spaces, and it allows the use of quaternions and dual-quaternions for- mulated as Motors. In this paper, a new methodology for accelerating the Study quadric Interpolation based on Conformal Geometric Algebra is presented. This methodology uses General Purpose Graphics Processing Units (GPUs) and it is applied for medical robotics, but it can be also extended to other areas like aeronautics, robotics, and graphics processing.

Keywords: Motor Interpolation; Conformal Geometric Algebra; Motor Algebra

Introduction

Generating trajectories for 3D objects is a fundamental problem in areas of robotics, image processing, and others. There are few mathematical algorithms for the interpolation of points that use vector calculus, quaternions, dual-quaternions, and linear algebra. Geometric algebra is a powerful mathematical framework for solving problems using basic geometry entities (circles, points, spheres, planes, and lines), and can represent Euclidean geometry, quaternions, and dual quaternions. For this reason, it can be used for interpolating geometric entities for applications in medical robotics, graphic engineering, robotics, and aeronautics. In this work, a GPU implementation for interpolating geometric entities of conformal geometric algebra is proposed. The design is based on an interpolation algorithm that maps rotations and translations of motors as 8D vectors in the Study manifold.

The paper is organized as follows: section ?? reports on related work, section 2 gives a brief introduction to geometric algebra, with special emphasis on conformal geometric algebra. Section 3 presents the motor interpolation algorithm based on geometric algebra for interpolating entities in different dimensions, while section 4 describes the optimization and implementation of the interpolation algorithm in GPU and section 5 shows the experimental results. Finally, section 7 presents our conclusions and future work. Due to the easy representation of complex problems in robotics and image processing, several works have been proposed to accelerate geometric algebra operations. Most related techniques for accelerating geometric algebra algorithms are based on speeding up basic operations (inner, outer and geometric product, etc) with FPGA Co-processors [1-4] and GPU parallelization [5,6]. More complex architectures have been developed for applications on computational vision. In [7] the authors present an algorithm of Clifford convolution and Clifford Fourier transforms for color edge detection, and an alternative algorithm based on rotor edge detection is proposed in [8].

Gerardo Soria-Garc´ıa et al. introduce an FPGA implementation of Conformal Geometric Algebra Voting Scheme for Geometric Entities Extraction, such as lines, and circles on images of edges [9]. A GPU implementation for conformal geometric algebra interpolation application based on GPU is presented in [10]. The algorithm presented in this work is a modification using motors of the dual-quaternion method to interpolate rotation, translation, and dilation of the geometric entities like points, lines, circles, pair of points, planes, and spheres.

Geometric Algebra

In an n-dimensional real vector space R^n, we can introduce an orthonormal basis of vectors {}, i = 1, ..., n. This leads to a basis for the entire geometric algebra.

$$1,\{e_i\},\{e_i \wedge e_j\},\{e_i \wedge e_j \wedge e_k\},....,e_1 \wedge e_2 \wedge ..... \wedge e_n \qquad (1)$$

The Geometric Algebra (GA) of the real n-dimensional quadratic vector space (V, Q) is denoted as G_n. In the case $G_{p,q,r}$ (n = p + q + r) denotes the GA of (V, Q) where p, q, r correspond the number of the basis vectors which square in GA to 1,-1,and 0, respectively. The geometric product of two vectors a and b is written as ab and can be expressed as a sum of its symmetric and antisymmetric parts:

$$ab = a \cdot b + a \wedge b \qquad (2)$$

where the inner product a · b and the outer product a ∧ b is defined by $\frac{1}{2}(ab+ba)$ and $a \wedge b = \frac{1}{2}(ab-ba)$

$G_n = G_{p,q,r}$ when n = p + q + r with p-unit vectors, q-unit vectors and r-unit vectors which square to 1,-1 and zero respectively. G_n is a graded linear space

$$\Lambda v = \overset{0}{\Lambda} v \oplus \overset{1}{\Lambda} v \oplus \cdots \oplus \overset{n}{\Lambda} \qquad (3)$$

where the elements of Λ^n are referred to as (homogeneous) multivectors of grad k for k = 0,1, 2,.....,*n*. In the following, for short, elements of $\Lambda^1 V$ are called vectors. Thus, any $M \in G_n$ can be uniquely decomposed into a sum $\sum_k < M >_k$ where $<M>_k \in \Lambda^k V$. Furthermore, G_n is a Z_2-graded algebra in the following sense:

$$G_n = G_n^{(0)} \oplus G_n^{(1)}, G_n^{(i)} G_n^{(j)} \in G_n^{i+j \bmod 2}, i,j = 0,1 \qquad (4)$$

Where

$$G_n^{(0)} = A \in G_n : A = \sum_{k\,even} < A >_k G_n^{(1)} = A \in G_n : A = \sum_{k\,odd} < A >_k \qquad (5)$$

Then,

$G_n^{(0)}\left(res G_n^{(1)}\right)$ is called the even (resp. the odd) part pf G_n. Note that due to equation (4), $G_n^{(0)}$ is a subalgebra of G_n. Later in this paper, the even subalgebra of G_n will be denoted by .

The dimension of G_n is 2^n. The multivector basis of G_n has 2^n bases for scalars, bivectors, trivectors and *k*-vectors. A *k*-blade is either the identity element 1 of G_n (when k = 0 or, when k > 0), it is defined as the wedge product $e_{i1} \wedge e_{i2} \wedge .... \wedge e_{ik} \left(i_1 < i_2 < ... < i_k\right)$ A linear combination of k-vectors is called a homogeneous multivector.

Consider two homogeneous multivectors A_r and $B_s \in G_n$ of grades r and s, respectively. The geometric product of A_r and B_s can be written as

$$A_r B_s = \begin{cases} \sum_{k\,even} < A_r B_s >_k \; when \; r+s = 0 \bmod 2 \\ \sum_{k\,odd} < A_r B_s >_k when \; r+s = 1 \bmod 2 \end{cases} \qquad (6)$$

Conformal Geometric Algebra

In Conformal Geometric Algebra $G_{4,1}$ (CGA), the Euclidean vector space R^3 is represented in $R_{4,1}$. The space for $G_{4,1}$ has an orthonormal vector basis given by $\{e_1, e_2, e_2, e_4, e_5\}$ with the properties $e_i^2 = 1, i = 1,2,3,4, e_4^2 = 1, e_5^2 = -1, e_i.e_4 = e_i.e_5 = e_4.e_5 = 0, i = 1,2,3$

The null basis $\{e_0, e_\infty\}$ (origin and point at infinity) is defined as

$$e_0 = \frac{e_5 - e_4}{2}, \qquad e_\infty = e_4 + e_5 \qquad (7)$$

These null vectors satisfy the relations $e_0^2 = e_\infty^2 = 0$ and $e_\infty . e_0 = -1$

Let be $E = e_\infty \wedge e_0$ the Minkowski plane. The unit Euclidean pseudo-scalar is $I_e := e_1 \wedge e_2 \wedge e_3$, and the conformal pseudoscalar $I_c = I_e E$ is used for computing the inverse and duals of multivectors. Given a nonsingular k-vector $A_k \in G_{4,1} \equiv G_{4,1,0,}$ the dual and its inverse are respectively.

$$A^* = AI_c \quad and \quad A^{-1} = (-1)^q \frac{A^\dagger}{|A|^2} \qquad (8)$$

where A^+ stands for the reversion of A and |A| its magnitude. (Table 1) where IPNS and OPNS stand for Inner Product Null Space and Outer Product Null Space respectively.

Table 1: Representation of entities in conformal geometric algebra.

Entity	**IPNS**	**OPNS**
Point	$x_c = x + \frac{1}{2}x^2 e_\infty + e_0$	$x^* = s_1 \wedge s_2 \wedge s_3 \wedge s_4$
Sphere	$S = c + \frac{c_e^2 - p^2}{2} e_\infty + e_0$	$s^* = x_{c1} \wedge x_{c2} \wedge x_{c3} \wedge x_{c4}$
Plane	$\pi = n - e_\infty d$ $n = (x_1 - x_2) \wedge (x_1 - x_3)$ $d = (x_1 \wedge x_2 \wedge x_3) I_e$	$\pi^* = e_\infty x_{c1} \wedge x_{c2} \wedge x_{c3}$
Line	$L = \pi_1 \wedge \pi_2$ $L = nI_e - e_\infty dmI_e$ $n = x_1 - x_2$ $m = x_1 \wedge x_2$	$I^* = e_\infty \wedge x_{c1} \wedge x_{c2}$
Circle	Z= $s_1 \wedge s_2$ Z= $\pi \wedge s$	$z^* = x_{c1} \wedge x_{c2} \wedge x_{c3}$

Points, Lines, Planes, and Spheres

The representation of a 3D Euclidean point x = [x, y, z]^T in R^3 in the geometric algebra G$_{4,1}$ is given by

$$x_c = x + \frac{1}{2}x^2 e_\infty + e_0 \tag{9}$$

Given two conformal points x$_c$ and y$_c$, their difference in Euclidean space can be defined as

$$x - y = (y_c \wedge x_c).e_\infty \tag{10}$$

and, consequently, the following equality

$$(x_c \wedge y_c + y_c \wedge z_c).e_\infty = (x_c \wedge z_c).e_\infty \tag{11}$$

is fulfilled as well. The line is formulated as a form in the Inner Product Null Space (IPNS) as follows

$$L = nI_E - e_\infty m = -(n + e_\infty m) \tag{12}$$

where n (bivector) is the orientation and the vector m the moment of the line. The plane is formulated as a form in IPNS

$$\pi = nI_E - e_\infty d = n - e_\infty d \tag{13}$$

where the n (bivector) for the plane orientation and d is the distance from origin orthogonal to the plane.

The sphere is formulated as a form in IPNS

$$s = p_c - \frac{1}{2}\rho^2 e_\infty = p + (\frac{p^2 - \rho^2}{2})e_\infty + e_0 \tag{14}$$

A poin is a sphere with zero radius. Considering the dual equation for the sphere, we can write the constraint for a point lying on a sphere

$$x_c \wedge s^* = x_c \wedge (s \cdot I_c) = 0 \tag{15}$$

The sphere can be directly computed by the wedge of four conformal points in the Outer Product Null Space (OPNS)

$$s^* = x_{c1} \wedge x_{c2} \wedge x_{c3} \wedge x_{c4} \tag{16}$$

Replacing any of these points with the point at infinity, we obtain the OPNS plane equation

$$\pi^* = x_{c3} \wedge x_{c1} \wedge x_{c2} \wedge e_\infty = x_3 \wedge x_1 \wedge x_2 \wedge e_\infty + ((x_3 - x_1) \wedge (x_2 - x_1))E \tag{17}$$

Similar to Eq. (17), the OPNS line equation can be formulated as a circle passing through the point at infinity

$$l^* = x_{c1} \wedge x_{c2} \wedge e_\infty \tag{18}$$

Rigid Transformations

In CGA many of the transformations can be formulated in terms of successive reflections between planes.

Reflection: The equation of a point x reflected with respect to a plane π is

$$x' = -\pi x \pi^{-1} \tag{19}$$

Translation: The transformation of geometric entity O formulated as two successive reflections w.r.t. the parallel planes π$_1$ and π$_2$ is given by

$$O' = \underbrace{(\pi_2\pi_1)}_{T_a} O \underbrace{(\pi_1^{-1}\pi_2^{-2})}_{T_a}, \quad T_a = 1 + \frac{1}{2}ae_\infty = e^{\frac{a}{2}e_\infty} \tag{20}$$

where a = 2*dn*, d is the distance of translation, and n is the direction of translation

Rotation: Similarly, we can formulate a rotation as the product of two reflections between non-parallel planes π$_1$ and π$_2$ which cross the origin.

$$O' = \underbrace{(\pi_2\pi_1)}_{R_\theta} O \underbrace{(\pi_1^{-1}\pi_2^{-2})}_{\tilde{R}_\theta} \tag{21}$$

The geometric product of the unit normal vectors of these planes n$_1$ and n$_2$ yields the equation for the rotor

$$R = n_2 n_1 = \cos(\theta/2) - \sin(\theta/2)n = e^{-\theta n/2} \tag{22}$$

where the unit bivector n = n$_1$ ∧n$_2$, and teh angle θ is twice the angle between π$_1$ and π$_2$

Screw Motion: The operator for screw motion or motor *M* is a composition of a translator *T* and a rotor *R*, both w.r.t. to an arbitrary axis *L*. The equation for a motor is as follows

$$\begin{aligned} M &= TR = \cos(\frac{\hat{\theta}}{2}) + L\sin(\frac{\hat{\theta}}{2}) = \cos(\frac{\theta + e_\infty d}{2}) + L\sin(\frac{\theta + e_\infty d}{2}) \\ &= R + e_\infty R' = \exp^{-\theta L} \end{aligned} \tag{23}$$

where the screw line is n and m stand for the orientation and momentum of the screw line and the dual angle angle . Finally, the motor transformation for any object O ∈ G$_{4',1}$ reads

$$O' = TRO\tilde{R}\tilde{T} = MO\tilde{M} \tag{24}$$

Interpolation Algorithm Using the Study Quadric Manifold

According Gfrerrer [11], one can formulate an interpolation algorithm using the Study's kinematics mapping. This maps Euclidean rigid transformations into homogeneous points which in turn belong to the Study quadric M6 (projective space P^7).

$$\vartheta = \begin{Bmatrix} SE(3) \to P^7 \\ \alpha \to X = (x_0, \ldots, x_7) \end{Bmatrix} \tag{25}$$

where SE (3) is the special Lie group for 3D Euclidean rigid transformations, α represents any Euclidean rigid transformation and the vector X containing homogeneous coordinates X ∈ $\mathcal{M}_6$. Note that the motor algebra is a sub-algebra of the 3D conformal geometric algebra G$_{4,1,}$ thus the interpolation uses motors

DOI: 10.26717/BJSTR.2021.38.006201

depending upon which algebra is used one can use I for motors of or e_∞ for motors of $G_{4,1}$ which both square to zero and are acting similarly as the isotropic operator of the dual quaternions which use ε which squares to zero as well. Given the set $\mathcal{X} \in \mathcal{M}_6$ containing three homogeneous points $X_1, X_2, X_3 \in P^7$, the interpolation curve X $\in \mathcal{M}6$ is generated by interpolating the given homogeneous points $X_1, X_2, X_n \in P^7$which satisfy the set $\mathcal{X}$ computed as follows

$$x = f_0(t) X_1^T 2 X_2 X_0 + f_1(t) X_0^T 2 X_2 X_1 + f_2(t) X_0^T 2 X_1 X_2 \quad (26)$$

The interpolation polynomials $f_0(t)$, $f_1(t)$, and $f_2(t)$ are formulated as follows

$$\begin{aligned} f_0 &= (t_0 - t_1)(t_0 - t_2)(t - t_1)(t - t_2) \\ f_1 &= (t_1 - t_0)(t_1 - t_2)(t - t_0)(t - t_2) \\ f_2 &= (t_2 - t_0)(t_2 - t_1)(t - t_0)(t - t_1) \end{aligned} \quad (27)$$

where t, t_0, t_1, and t_2 are interpolation values between 0 and 1. t represents the segment of curve where the point is calculated, t_0, t_1 and t_2 are values that represent the section of curve where interpolated and control points meeting. The Study-quadric bilinear form is given by the following matrix G,

$$S = \begin{bmatrix} 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \end{bmatrix} \quad (28)$$

If one wishes to interpolate more points, the above-explained interpolation equation can be extended and adapted as De Casteljou algorithm [12]. This is shown on Algorithm 1. This algorithm calculates a homogeneous point for the given set X that contains the control points $(X_1,\cdots, X_N)$ and the interpolation values (t, t_0,t_1,t_2) and the the Study quadric bilinear matrix Q. Here t, t_0, t_1, t_2 are interpolation variables. t is the position interval variable and allows to determine the position within the curve of the interpolation point, the value t goes from 0 to 1. To calculate all the points of the curve, all the values of t are traversed. (t_0, t_1,t_2) are values that indicate at which point the interpolation points, in the functions that came in Klawiter library [12], the values $t_0 = 0, t_1 = 0.5, t_2 = 1$ are proposed so that the interpolated points correspond to the initial control point, the mid-control point and the endpoint respectively. These are fixed for each curve, only t is what can change when calculating a point within a specific curve.

```
Function SQI(X, t, t_0, t_1, t_2):
Input: A list X that contains the control homogeneous points $(x_1,\cdots,x_N)$
       and the interpolation values $t$, $t_0$, $t_1$ and $t_2$
Output: An homogeneous point that lies in the inerpolation curve $\mathscr{X}$
Result: Conjunto de puntos que indican la posicin durante el movimiento
        del brazo robtico.

begin
    if Points in X are pair then
        return Error
    else if Points in X = 1 then
        return X
    else
        $f_0 \leftarrow (t_0-t_1)(t_0-t_2)(t-t_1)(t-t_2)$
        $f_1 \leftarrow (t_1-t_0)(t_1-t_2)(t-t_0)(t-t_2)$
        $f_2 \leftarrow (t_2-t_0)(t_2-t_1)(t-t_0)(t-t_1)$
        $\mathscr{X}_t \leftarrow \emptyset$
        foreach $x_i \in X$ except $\{x_1, x_N\}$ do
            $\mathscr{X}_t \leftarrow f_0 x_i^T \mathscr{Q} x_N x_1 + f_1 x_1^T \mathscr{Q} x_N x_i + f_2 x_1^T \mathscr{Q} x_i x_N$
            $\mathscr{X} \leftarrow \mathscr{X} \cup \mathscr{X}_t$
        end
        return SQI($\mathscr{X}$, t, t_0, t_1, t_2)
end
```

Algorithm 1: Study quadric interpolation.

First, the algorithm checks the number of elements in X. If there is only one point, this is returned as the solution. Otherwise, X is used to get intermediate control points via equation (26) and saved in $\mathcal{M}.\mathcal{M}$ is used to call the algorithm recurrently until we get only one interpolation point and returned it. Figure 1 shows the algorithm interpolation for five control points. See that in each interpolation step, for N control points, we get N−2 interpolation points and N −1 repeats are required. Since in the last step, equation (26) required 3 points, an odd number of control points are needed. where X is the set of control homogeneous points $(X_1, \cdot, X_N)$ that describe the curve trajectory, G is the quadric matrix, t_0, t_1, and t_2 are global constant values, in this case,0, 0.5, and 1 respectively. where G is the quadric matrix, and X the array of homogeneous points that contains the points X_1 to X_n, t_0, t_1, and t_2 are constant values, in this case, 0.5, and 1 respectively. As is seen int the algorithm, is required that the array X have an odd number of homogeneous points to get an interpolation curve. Note that the solution is in fact a rational motion utilizing a interpolating spline which in turn involves rational sub-spline motions, see [13].

In general, the Study quadric interpolation algorithm uses homogeneous points represented by dual-quaternions or by homogenous matrices. As we know, motors are isomorphic to dual-quaternions and a motor can be represented as a homogeneous point as well. This motor represented as a vector $\vec{M} = \{M_0, \ldots, M_7\} \in R^8$ Lies on the Study Manifold, see Table 2. In this regard, we propose a modified interpolation algorithm. For that, the homogeneous points are replaced by motors and substitute matrix operations with GA operations. Since the matrix multiplications in (26) with form , which result is a constant value expressed as:

$$x_a^T 2x_b = \frac{x_{a0}x_{b7} + x_{a1}x_{b6} + x_{a2}x_{b5} + x_{a3}x_{b4} + x_{a2}x_{b3} + x_{a1}x_{b2}}{2} \quad (29)$$

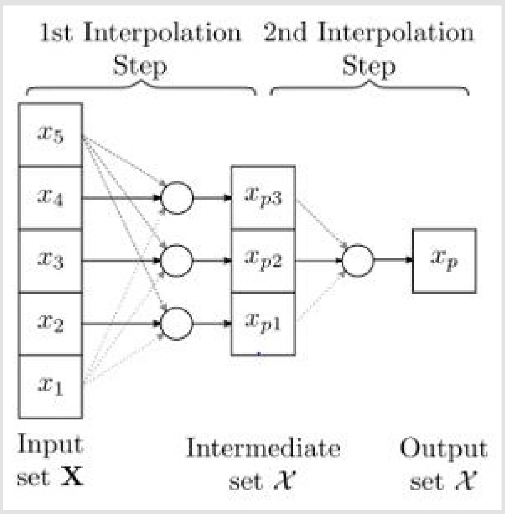

Figure 1: Interpolation of five control points with algorithm 1.

Table 2: Study quadric homogeneous point represented in dual-quaternion and its equivalences as motor in $G^+_{3,0,1}$ and as motor in $G_{4,1}$.

Homogeneous point	Dual-quaternion	$G^+_{3,0,1}$	$G_{4,1}$
x_0	α_0	α_0	α_0
x_1	$\alpha_1 i$	$\alpha_1 e_{12}$	$\alpha_1 e_{12}$
x_2	$\alpha_2 j$	$\alpha_2 e_{31}$	$\alpha_2 e_{31}$
x_3	$\alpha_3 k$	$\alpha_3 e_{23}$	$\alpha_3 e_{23}$
x_4	$\alpha_4 \varepsilon$	$\alpha_4 e_{1234}$	$\alpha_4 e_{123\infty}$
x_5	$\alpha_5 \varepsilon i$	$\alpha_5 e_{43}$	$\alpha_5 e_{\infty 3}$
x_6	$\alpha_6 \varepsilon i$	$\alpha_6 e_{42}$	$\alpha_6 e_{\infty 2}$
x_7	$\alpha_7 \varepsilon i$	$\alpha_7 e_{41}$	$\alpha_7 e_{\infty 1}$

Changing the homogeneous points with their respective motor representation in CGA and analyzing the motor multiplication, is shown that the coefficient of the blade $e_{123\infty}$ (α_3) from the motor multiplication is the double value of (29). This coefficient (α3) can be extracted from any motor using the partial derivative:

$$\alpha_3(M) = \frac{\partial M}{\partial e_{123\infty}} \quad (30)$$

or via inner product with the I_e and e_0:

$$\alpha_3(M) = (I_e \cdot M) \cdot e_0 \quad (31)$$

Using GA, the equation (26) is reformulated for a motor based interpolation algorithm. It uses 3 control motors as follows

$$M = \frac{f_0(t)\alpha_3(M_1\tilde{M}_2)M_0 + f_1(t)\alpha_3(M_0\tilde{M}_2)M_1 + f_2(t)\alpha_3(M_0\tilde{M}_1)M_2}{2} \quad (32)$$

In the same manner as Algorithm 1, for more control points, equation (32) can be rewritten using the De Casteljou algorithm as described on Algorithm 2. This algorithm calculates one motor gave the set M of control motors ($m_1, \cdots, m_N$) and the interpolation constants (t, t_0, t_1, t_2). Similar to Algorithm 1, the interpolation function is called recursively with the calculated motors from the last step as input until we get only one motor. Since the last step needs three motors to calculate the last one, **M** must contain an odd number of motors. (Algorithm 2)

```
Function CGAMI($\mathbf{M}, t, t_0, t_1, t_2$):
Input: A set $\mathbf{M}$ that contains the control motors $(m_1, \cdots, m_N)$ and the
       interpolation values $t$, $t_0$, $t_1$ and $t_2$
Output: A motor that lies in the inerpolation curve $\mathscr{M}$
begin
    if Motors in $\mathbf{M}$ are pair then
        return Error
    else if Motors in $\mathbf{M}$ = 1 then
        return $M$
    else
        $f_0 \leftarrow (t_0 - t_1)(t_0 - t_2)(t - t_1)(t - t_2)$
        $f_1 \leftarrow (t_1 - t_0)(t_1 - t_2)(t - t_0)(t - t_2)$
        $f_2 \leftarrow (t_2 - t_0)(t_2 - t_1)(t - t_0)(t - t_1)$
        $\mathscr{M} \leftarrow \emptyset$
        foreach $m_i \in \mathbf{M}$ except $\{m_1, m_N\}$ do
            $\mathscr{M}_t \leftarrow \frac{f_0\alpha_0(m_i\widetilde{m_N})m_1 + f_1\alpha_0(m_1\widetilde{m_N})m_i + f_2\alpha_0(m_1\tilde{m}_i)m_N}{2}$
            $\mathscr{M} \leftarrow \mathscr{M} \cup \mathscr{M}_t$
        end
        return CGAMI($\mathscr{M}, t, t_0, t_1, t_2$)
end
```

Algorithm 2: Motor Interpolation on CGA.

Speeding up the Algorithms Using a GPU Accelerator

The algorithmic complexity using the framework $G_n \in R^{2n}$ is: for binary operations $O(2^{2N})$; for unary operations $O(2^N)$, and for M interpolated motors using K motors O(MK). To accelerate the motor interpolation algorithm for real-time applications, one can utilize arithmetic accelerators based on FPGA, ASIC, and GPU. We utilize an accelerator based on CPU-GPU co-design, where the GPU is the GTX-970 by Nvidia, which has a computing capability of 5.2 and it can be implemented on any GPU using CUDA. The algorithm was developed to compute in parallel single or multiple interpolation curves. For the computing of a single interpolation curve, the default GPU was implemented, where each motor is computed in one thread launched in a Stream. For the case of multiple curves, the multi-streaming implementation was selected to overlap the streams. GTX-970 GPU does not support recursive algorithms, due to hardware limitations. To implement the motor interpolation, we translated the Algorithm 2 into an equivalent-iterative version given by the Algorithm 3.

```
Function CGAMI($M, t, t_0, t_1, t_2$):
Input: A set M which contains the control motors $(m_1, \cdots, m_N)$ and the
        interpolation values $t$, $t_0$, $t_1$ and $t_2$
Output: A motor that lies in the inerpolation curve $\mathscr{M}$
begin
    if Motors in M are pair then
        return Error
    else if Motors in M = 1 then
        return M
    else
        $f_0 \leftarrow (t_0 - t_1)(t_0 - t_2)(t - t_1)(t - t_2)$
        $f_1 \leftarrow (t_1 - t_0)(t_1 - t_2)(t - t_0)(t - t_2)$
        $f_2 \leftarrow (t_2 - t_0)(t_2 - t_1)(t - t_0)(t - t_1)$
        repeat
            $\mathscr{M} \leftarrow \emptyset$
            foreach $m_i \in$ M except $\{m_1, m_N\}$ do
                $\mathscr{M}_t \leftarrow \frac{f_0 \alpha_0 (m_i \widetilde{m_N}) m_1 + f_1 \alpha_0 (m_1 \widetilde{m_N}) m_i + f_2 \alpha_0 (m_1 \widetilde{m_i}) m_N}{2}$
                $\mathscr{M} \leftarrow \mathscr{M} \cup \mathscr{M}_t$
                M $\leftarrow \mathscr{M}$
            end
        until Motors in M = 1
        return $\mathscr{M}$
end
```

Algorithm 3: Motor Interpolation on CGA (iterative).

The versatile Compute Unified Device Architecture (CUDA), is a platform for parallel computing, which was developed by NVIDIA for highly parallel implementations and uses its GPUs. CUDA is based on C/C++ programming languages, however other languages like Python or Fortran can be used as well. Our accelerator was written with CUDA C++ for the GPU acceleration and C++ for the CPU interface. The GTX-970 GPU does not support recursive algorithms; thus the Algorithm 2 has to be translated into an equivalent iterative version as presented in Algorithm 3. In this regard, the algorithm uses one stream for one curve of interpolation and multi-stream for multiple curves. Here, the GPU global memory is the largest in GPU, it has the highest latency, and it can be accessed from any stream for multi-stream applications. This is very helpful, if the number of motors to be computed is large, or if we compute multiple curves, or launch many algorithms employing the same GPU via multi-streaming. The GPU and CPU have different architectures.

The CPU has a few sets of cores for sequential applications. In contrast, the GPU has many cores for highly parallel computing. Figure 2 compares the CPU and GPU architectures. Note that GPU cores run asynchronously using threads. Each core is limited to 32, 64, 128, 256, or 512 threads. If a parallel GPU implementation is launched, it is required a certain number of cores. These are figured out as a direct function of the selected threads per block. In the case of the motor interpolation, each thread launches a copy of the algorithm.

All GPU operations run explicitly or implicitly on a stream. If a kernel is launched in GPU, it runs explicitly or implicitly on a stream. A stream is a sequence of asynchronous operations which are executed on a device following an order ruled by the host code. A stream involves these operations to maintain order, to permit operations to be queued in the stream, and to be executed after all preceding operations. It cares for querying the status of queued operations. These operations include data transfer and kernel launches. Recent GPUs allow a parallel technique known as multi-streaming, where multiple streams are launched, and the transference data and kernel launch can be overlapped as shown in Figure 3. For the case of parallel calculation of multiple curves, the multi-streaming implementation is used, and each curve is computed in an independent stream.

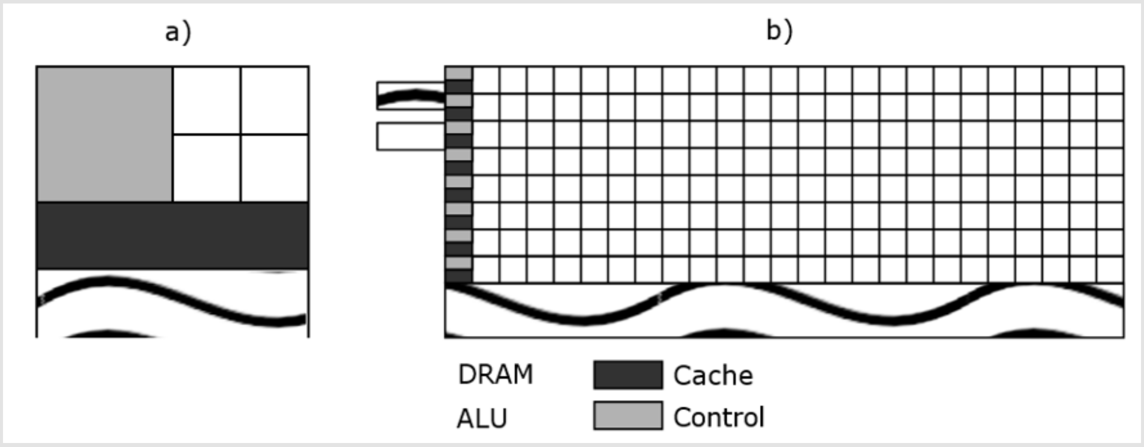

Figure 2: Architecture comparison of
a) CPU and
b) GPU.

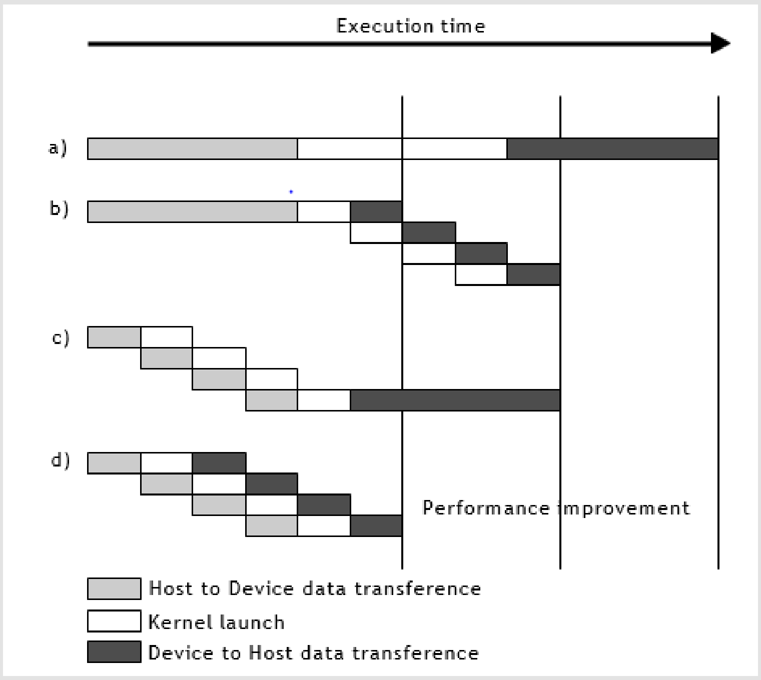
Execution time
a)
b)
c)
d)
Performance improvement
Host to Device data transference
Kernel launch
Device to Host data transference

Figure 3: Different streams used for a concurrent implementa- tion

a) 1-stream,

b) Concurrent approach for device-to-host asynchronous memory copy,

c) Concurrent approach for host-to-device asynchronous memory copy,

d) Concurrent approach for host-to- device and device-to-host asynchronous memory copy.

Experiments

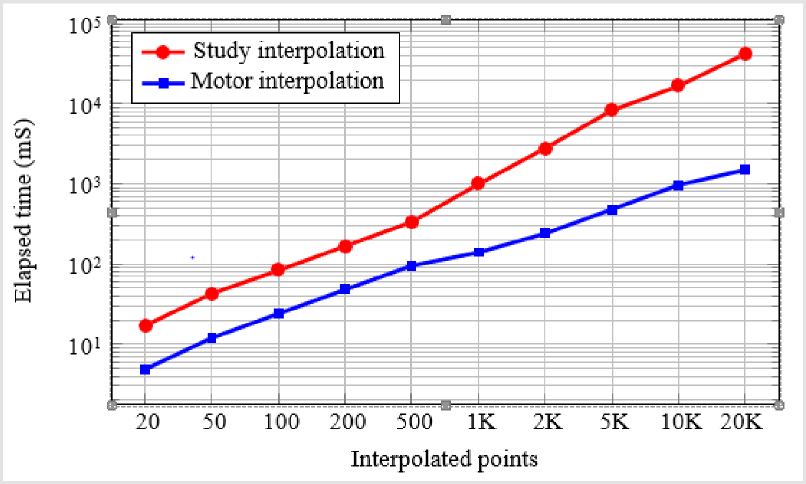

Figure 4: Comparison of Study's quadric and Motor interpola tion speed-up.

The Study's quadric and the motor interpolation are compared to analyze which of the algorithms has the best speed performance. Many motors were calculated using 15 test groups of 5 motors considering following amount of points 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000, and 20000. The test was programmed using MATLAB on a Desktop with Intel core I7 microprocessor at 2.4 GHz, 32 GB of RAM, and a GTX-970 GPU by NVIDIA. The results are presented in Figure 4. The execution time of both algorithms grew linearly while the number of points increases, thus the algorithm complexity is O(n). We see that the motor interpolation version is faster than the Study quadric interpolation, namely for 20000 points, it is 28× times faster. Figure 4 shows that the motor interpolation algorithm selected for sequential and parallel implementation utilizes same hardware parameters. The sequential implementation was programmed using C++ language and CUDA C++ for the parallel approach using 32 threads per block in the kernel. The interpolated motors were computed using single and double-precision floating-point representation. All the measures consider only the algorithm execution, but data transference, file writing, and reading are excluded. Results are shown in Figure 5.

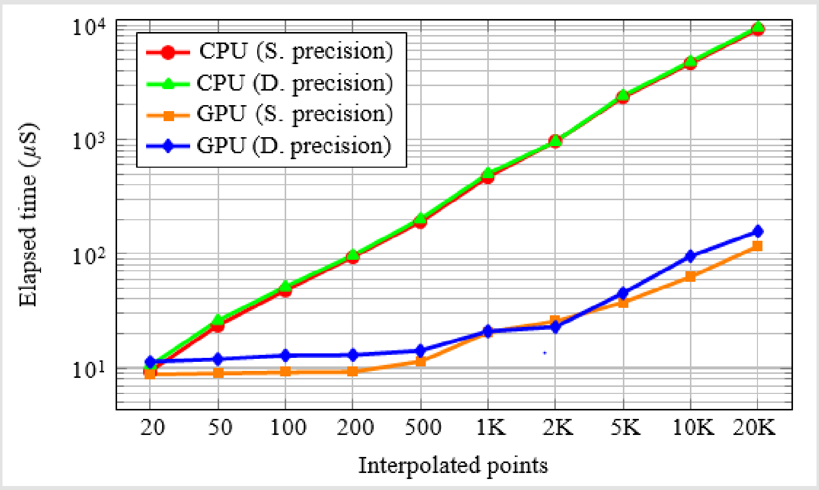

Figure 5: CPU and GPU speed-up analysis.

It was confirmed that the GPU has the best time performance for calculating the interpolation algorithm. When the time execution in CPU increases linearly, the time in GPU for 500 or fewer points remains almost constant. For the case of 20000 motors, the GPU implementation runs 79× and 60× times faster than the CPU for single and double floating-point respectively. The different multi-streaming implementations were analyzed; 16 different interpolation curves were calculated using the concurrent approach for host-to-device and device-to-host asynchronous memory copy with 4 configurations (1, 2, 4, and 8 streams) with 32 threads per block. The measure times include data transference. Figure 6 depicts that the use of multi-streaming reduces the execution time considerably, i.e. for 20000 points, 8 streams version is 2.7×, 1.9× and 1.6× times faster than 1, 2 and 4 streams respectively.

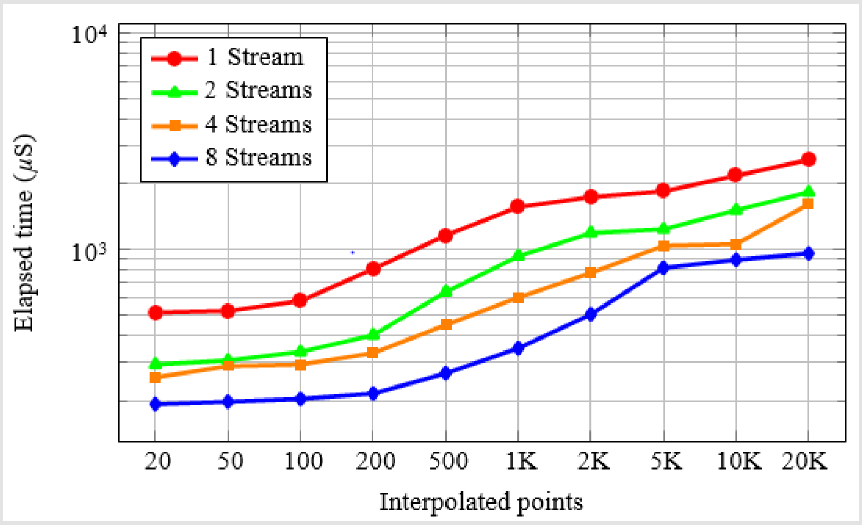

Figure 6: Comparison of the Multi-Stream speed-up.

DOI: 10.26717/BJSTR.2021.38.006201

Interpolation of Surgery Motion

This Study's quadric-based motor interpolation was used to interpolate trajectories in medical robotics for kidney surgery, see Figure 7. The robot manipulator hast to follow a path for suture around a certain region on the kidney phantom. We design to tasks follow an Ultrasound (US)-probe trajectory and the interpolation of a given 3D points sequence. One gives the point coordinates and their desired orientations, these are interpolated by the motor algorithm, see Figures 7b-7d. Afterward, we check the precision error using inverse kinematics. The inverse kinematics was also formulated in conformal geometric algebra, the algorithm was reported in [14 -17].

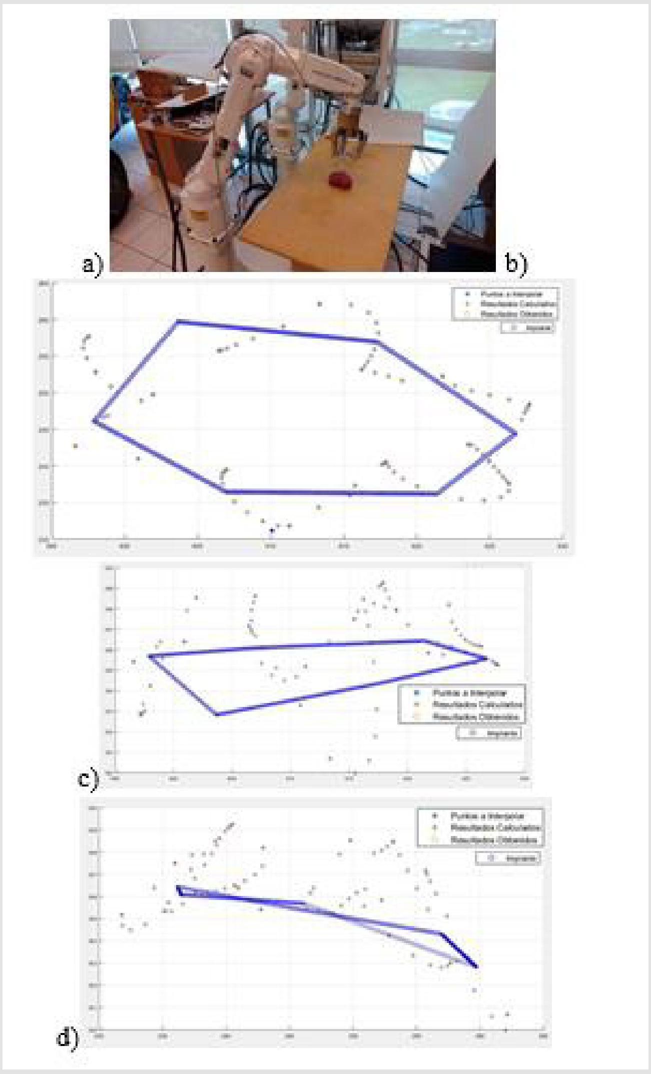

Figure 7:

a) Robot; views:

b) X-Y,

c) X-Z,

d) Y-Z.

Conclusion

In this work, an optimized version of the Study quadric interpolation algorithm is presented using the conformal geometric algebra framework. A Matlab implementation confirms that conformal geometric algebra interpolation is 28x times faster than a Study quadric interpolation for 20000 motors. The performance of CPU and GPU approaches of motor interpolation algorithm were contrasted for the cases of single and double floating-point precision. The comparison shows that the GPU is 79x and 60 times faster than the CPU implementation for single and double precision respectively. Furthermore, 16 interpolation curves were calculated via multi-streaming. These results confirm that multi streaming reduces the execution time for multiple curves interpolation. We show an application of the motor algorithm for suture by kidney surgery. In near future, we will pursue to implement the interpolation algorithm using multiple GPUs and reduce the data dependency. Also, we will build an OpenCL library of the algorithms using Python.

Competing Interests

There was no competing interest among the authors.

References

1. Franchini Silvia, Gentile Antonio, Sorbello Filippo, Vassallo Giorgio, Vitabile S (2008) An FPGA implementation of a quadruple-based multiplier for 4D Clifford algebra. 11th EUROMICRO Conference on Digital System Design: Architec- tures, Methods and Tools IEEE, pp. 743-751.
2. Franchini Silvia, Gentile Antonio, Sorbello Filippo, Vassallo Giorgio, Vitabile Salvatore, et al. (2011) A New Embedded Coprocessor for Clifford Algebra Based Software Intensive Systems. International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS), IEEE, pp. 335-342.
3. Franchini Silvia, Gentile Antonio, Sorbello Filippo, Vassallo Giorgio, Vitabile Salvatore (2013) Design and implementation of an embedded coprocessor with native support for 5D, quadruple-based Clifford algebra. IEEE transactions on Computers 62(2): 2366-2381.
4. Franchini Silvia, Gentile Antonio, Sorbello Filippo, Vassallo Giorgio, Vitabile Salvatore (2015) ConformalALU: a conformal geometric algebra coprocessor for medical image processing. IEEE Transactions on Computers 64(4): 955-970.
5. Lange Holger, Stock Florian, Koch Andreas, Hildenbrand Dietmar (2009) Acceleration and energy efficiency of a geometric algebra computation using reconfig- urable computers and GPUs. FCCM'09 17th IEEE Symposium on Field Programmable Custom Computing Machines, pp. 255-258.
6. (2015) Accelerating Clifford Algebra Operations Using GPUs and an OpenCL Code Generator, Franchini, Silvia and Gentile, Antonio and Vassallo, Giorgio and Vitabile, Salvatore, 2015 Euromicro Conference on Digital System Design, IEEE, pp. 57-64.
7. Franchini Silvia, Gentile Antonio, Vassallo Giorgio, Sorbello Filippo (2013) A specialized architecture for color image edge detection based on Clifford algebra. Complex, Intelligent, 2013 Seventh International Conference Com- plex, Intelligent, and Software Intensive Systems (CISIS) IEEE, pp. 128-135.
8. (2006) Color edge detection hardware based on geometric algebra, Mishra, Biswajit and Wilson, Peter, The 3rd European Conference on

Visual Media Production (CVMP 2006) Part of the 2nd Multimedia Conference 2006, IET, pp. 115–121,

9. (2007) Conformal Geometric Algebra Voting Scheme implemented in reconfigurable devices for geometric entities extraction, Soria-Garcia, Gerardo and Altamirano-Gomez, Gerardo Esteban and Ortega-Cisneros, Susana and BayroCorrochano, Eduardo, IEEE Transactions on Industrial Electronics, IEEE, pp. 8747-8755.
10. Papaefthymiou Margarita, Papagiannakis George (2016) An inclusive Conformal Geometric Algebra GPU animation interpolation and deformation algorithm. The Visual Computer 32(6-8): 751-759.
11. (2010) Geometric Computing for Wavelets Transforms, Robot Vision, Learning, Control and Action. Bayro-Corrochano, E Springer.
12. Li Hongbo, Hestenes David, Rockwood (2001) Generalized homogeneous coordinates for computational geometry. Alyn, Geometric Computing with Clifford Algebras, Springer, pp. 27-59.
13. Hestenes David, Sobczyk Garret (1984) Clifford algebra to geometric calculus: a unified language for mathematics and physics. D. Reidel Publishing Company.
14. BayroCorrochano, Eduardo Reyes Lozano Leo, Zamora Esquivel Julio (2006) Conformal geometric algebra for robotic vision. Journal of Mathematical Imaging and Vision 24(1): 55-81.
15. Klawitter D (2010) Documentation-Kinematic Toolbox. Technische Universit¨t Dresden.
16. Gfrerrer A (2010) Studys Kinematic Mapping. A Tool for Motion Design. Advances in Robot Kinematics, Springer, pp. 7-16.
17. Prautzsch Hartmut, Boehm, Wolfgang, Paluszny Marco (2013) Bezier and B-spline techniques. Springer Science & Business Media.

ISSN: 2574-1241

DOI: 10.26717/BJSTR.2021.38.006201

Eduardo Bayro Corrochano. Biomed J Sci & Tech Res