Computer graphics representation and transformation of geometric entities using dual unit vectors and line transformations

Phillip Azariadis, Nikos Aspragathos*

Robotics Group, Department of Mechanical and Aeronautical Engineering, University of Patras, 26500 Patra, Greece

Abstract

In this paper, a representational model is proposed for the description and transformation of three-dimensional geometric entities in computer graphics. The structure of the proposed representation is based on dual unit vectors, while the corresponding transformations are carried out through dual unit quaternions or dual orthogonal matrices. The main advantage of this representation is its compactness since the additional useful geometric characteristics of a represented curve or surface such as a tangent or normal vector are incorporated within the actual representational structure itself. Rotations, translations and view transformations are naturally expressed using the concept of screw displacement, while scaling is accomplished utilizing the moment vector of each dual line. Furthermore, an analysis of the transform operator based on dual unit quaternions is presented in order to ascertain an efficient formula to be used in the implementation of a computational algorithm for computer animation. Finally, an analytical comparison between the proposed representational model and the usual homogeneous model in computer animation is presented showing the merits of our method.

Keywords: Representational model; Dual points; Homogeneous points; Spatial transformations; Screw displacement; Spatial motion; Animation

1. Introduction

Several representational forms for three-dimensional objects have been developed in computer graphics [1,2]. In the majority of these techniques, 3-vector representations of points have been adopted because it seems convenient to consider three-dimensional space to be composed of an aggregate of $\mathbb{R}^3$ points and to describe the space and its properties in terms of these points. Therefore, implementations of polygon mesh representations based on a 3-vector representation of point-vertices are ubiquitous in computer graphics systems. However, this representational structure does not offer further geometric data concerning the object represented, other than the coordinates of each point-vertex. Thus, the corresponding database is usually expanded to include further geometric data such as the normal or the tangent vector at each point-vertex of the object surface. These geometric data have to be recomputed or transformed after each transformation of the represented object.

In the area of kinematics several techniques based on 6-vector screws and 8-vector quaternions [3,4] for the representation and transformation of the manipulators have been devised [5–8]. In particular, a configuration of a chain of links is defined via dual angles, while the position and orientation of the end-effector of a robot is calculated by screws and line transformations. This representational model seems to offer a compact and efficient structure in robot kinematics [9–11]. In general, the design of rigid-body motion using dual quaternion curves interpolating a given set of positions and orientations has recently received considerable attention in the
fields of robotics, computer graphics (key frame animation) and computer-aided geometric design (interactive interpolation schemes for design) [12,13]. However, these techniques do not rely on the three-dimensional Euclidean space $E_3$ but on the eight-dimensional real-vector space $A_o(9)$. Since no structures exist for visualizing objects in this second vector space, the dual quaternion curves are mapped to 3 objects in this second vector space, the dual quaternion space $A$. Another possibility is to use the simple quaternion representation of displacements in the group $SO(4,9)$ [14–17]. Shoemake [14] proposed a technique for interpolating rotational motions using quaternions. He suggested that quaternions express more naturally and more efficiently such transformations than transformations based on Euler angles and homogeneous coordinates. The superior performance of the quaternion operator compared to the homogeneous transformation has been verified subsequently by Funda et al. [18] and has been extended by many researchers [15–17]. Nowadays, point transformations based on 4-vector quaternions have been adopted by the computer graphics community and have been implemented (in software and hardware) by modern computer graphics APIs [19].

A first attempt to describe the surface of a three-dimensional object using dual vectors in computer graphics was given by Parkin [20]. He suggested a surface-traversing algorithm using the mutual moment of screw vectors as a navigational control quantity. The object surface was expressed using a network of polygonal facets defined by screws. Then using the properties of the mutual moment vector, he defined the points of intersection between these screws and a plane in space. Although a new representational model for polygonal surfaces is indirectly proposed, the author did not describe a particular mathematical structure that could be generally used for the representation of geometric entities.

In this paper, an alternative representational model for three-dimensional geometric entities is proposed, which is based on dual unit vectors and dual unit quaternions. A new structure with sufficient degrees of freedom to hold three real point-vectors is described, which is called a dual point. Dual point transformations are based on dual unit quaternions, which can simultaneously translate and rotate each dual point (and thus the three point-vectors) about an arbitrary spatial axis. The last transformation is usually called screw displacement. An analysis is presented in this paper to ascertain an efficient form of screw displacement with respect to computational and memory costs. Dual points are used to describe line segments, curves and surfaces in $E_3$ as well as geometric-invariant properties such as normal vectors or curvature vectors. Furthermore, we describe how scaling and view transformations can be implemented using dual unit quaternions. The computational cost and the minimal memory demands of the proposed structures and transformations are also investigated and presented.

In the first part of this paper, we briefly describe the necessary mathematical background associated with dual unit vectors and line transformations. The reader who is not familiar with the dual numbers and dual vectors is referred to Appendix A, where the mathematical formulations and the necessary notational conventions are given. In the third section, we present an analysis of line transformations deriving a structure known in the kinematics as the $3 \times 3$ dual orthogonal matrix. In the fourth section, we define the proposed structure of a dual point, which is used to represent line segments, curves and surfaces in $E_3$. At the end of this paper, view transformations based on dual unit quaternions are presented and an indicative application of the proposed representational model in computer graphics animation is illustrated.

2. Line representation and transformation

2.1. The six Plücker coordinates of a line

For the representation of a line in space, the six Plücker coordinates [21] are used which are analogous to the four homogeneous coordinates of a point. These six coordinates are the components of two real vectors $\mathbf{L}$ and $\mathbf{L}_0$ (Fig. 1). The first unit vector $\mathbf{L}$ is determined by the three direction cosines $L$, $M$ and $N$ of the given line. The second vector $\mathbf{L}_0$ with components $L_0$, $M_0$ and $N_0$ is the moment of the line about the origin. Thus,

$$\mathbf{L}_0 = \mathbf{r} \times \mathbf{L},$$

where $\mathbf{r}$ is the position vector of any point on the line. In addition, the six Plücker coordinates satisfy the following two relationships:

$$\mathbf{L} \cdot \mathbf{L}_0 = L L_0 + M M_0 + N N_0 = 0$$

and

$$\mathbf{L} \cdot \mathbf{L} = L^2 + M^2 + N^2 = 1.$$  

From Eqs. (2) and (3) it is clear that a line has four degrees of freedom and that 8 such lines are required to fill the space. Moreover, it is useful to relate these six coordinates with the three real direction cosines of the

1 Veldkamp [6] uses the term dual point to express a point of a dual curve $\mathbf{s} = x(u) + x_0(u)$ or of a dual surface $\mathbf{s} = x(u, v) + x_0(u, v)$. In this paper, we adopted the term dual point in a different sense in order to simplify the presentation of the techniques.
given line. It can be proved [5] that
\[
\cos(x + \varepsilon d) = \cos x - \varepsilon d \sin x = L + \varepsilon L_0,
\]
\[
\cos(y + \varepsilon d_y) = \cos y - \varepsilon d_y \sin y = M + \varepsilon M_0,
\]
\[
\cos(z + \varepsilon d_z) = \cos z - \varepsilon d_z \sin z = N + \varepsilon N_0,
\]
where \(\varepsilon\) is the dual unit (see Appendix A). The dual numbers \(L + \varepsilon L_0, M + \varepsilon M_0\) and \(N + \varepsilon N_0\) are called dual direction cosines of the line and constitute the three components of the dual unit vector \(\hat{L}\) written as
\[
\hat{L} = L + \varepsilon L_0 = (L, M, N) + \varepsilon(L_0, M_0, N_0)
= (L + \varepsilon L_0, M + \varepsilon M_0, N + \varepsilon N_0).
\]
From Eqs. (2), (3) and (7), it is clear that at least four memory locations of double precision are needed for the storage of each dual unit vector.

2.2. Line transformations

Dual unit quaternions can be used to transform dual unit vectors in space. When a dual unit quaternion operates on a dual unit vector, this vector suffers a screw displacement about a dual axis in space. This screw displacement is expressed by a dual number of the form \(\hat{\theta} = \theta + \varepsilon S\), where \(\theta\) is the rotation angle about a line and \(S\) is the translation along the same line. In computer graphics, where the axis of the transformation is usually not intersecting the axis of the unit line vector at right angles, such a dual unit quaternion \(\hat{q}\) is expressed as
\[
\hat{q}_d(\theta, S) = \frac{\hat{\theta}}{2} + \sin \frac{\hat{\theta}}{2} \hat{n},
\]
where \(\hat{n}\) is a unit line vector such that
\[
\hat{n} = (\ell + \varepsilon')i + (m + \varepsilon n_0)j + (n + \varepsilon n_0)k,
\]
where, \(i, j, k\) satisfy the following multiplication rules of the quaternion algebra:
\[
i^2 = j^2 = k^2 = ijk = -1 \quad \text{and} \quad \]
\[
ij = k = -ji, \quad jk = i = -kj, \quad ki = j = -ik
\]
and
\[
(\ell + \varepsilon')^2 + (m + \varepsilon n_0)^2 + (n + \varepsilon n_0)^2 = 1.
\]

Dual quaternions form an eight-dimensional real vector space \(A_8(8)\). With the above multiplication rules they determine a module denoted by \(H(D)\) (\(D\) is defined in Appendix A). The quaternion of Eq. (8) operates on any dual unit vector \(\hat{L}\) by screw displacing its axis, without stretching, by an angle \(\theta\) about \(\hat{n}\) and by a distance \(S\) along \(\hat{n}\). This transformation is performed by [5]
\[
\hat{L}' = \hat{q}_d(\theta, S)\hat{L}\hat{q}_d^{-1}(\theta, S).
\]

The main advantage of a dual quaternion representation for a screw displacement lies in its simplicity and economy. These characteristics are particularly useful when several successive transformations are performed as in the derivation of the kinematics equations of a spatial linkage. From Eq. (8), it is derived that only six memory locations are needed to store a dual unit quaternion, while the well-known \(4 \times 4\) general matrix for homogeneous transformations about an arbitrary spatial axis requires at least nine memory locations (see Appendix B).
3. Analysis of the line transformations

In this section, we analyze further the line transformation expressed by Eq. (12) to ascertain which form of line transformations based on dual unit quaternions is efficient with respect to the computational and memory cost, since the form of Eq. (12) is general and requires further refinement in order to be suitable for implementation into a computational algorithm for use in applications such as computer animation. During the following analysis we derive an efficient formula for line transformations using dual unit quaternions. Based on this formula we obtain the equivalent $3 \times 3$ dual orthogonal matrix (as it is known in kinematics). Then these two transform operators are compared with respect to the computational and memory cost. Let us rewrite the dual unit vector defined by Eq. (7) in the form

$$\mathbf{L} = \hat{L}_1 \mathbf{i} + \hat{L}_2 \mathbf{j} + \hat{L}_3 \mathbf{k}.$$  

(13)

Let $\hat{q}$ be a dual unit quaternion and $\hat{q}^{-1}$ its inverse (or conjugate), that is,

$$\hat{q} = \hat{q}_1 + \hat{q}_2 \mathbf{i} + \hat{q}_3 \mathbf{j} + \hat{q}_4 \mathbf{k} = \hat{q}_1 + \hat{q}_2 \text{,}$$

(14)

$$\hat{q}^{-1} = \hat{q}_1 - \hat{q}_2 \mathbf{i} - \hat{q}_3 \mathbf{j} - \hat{q}_4 \mathbf{k} = \hat{q}_1 - \hat{q}_2 \text{,}$$

(15)

with

$$\hat{q}_1^2 + \hat{q}_2^2 + \hat{q}_3^2 + \hat{q}_4^2 = 1 \text{ or } \hat{q}_1^2 + \hat{q}_2^2 = 1. \text{ (16)}$$

The result of a general screw displacement of $\mathbf{L}$ is denoted by $\hat{\mathbf{L}}$. Taking into consideration that the quaternion product of two dual unit vectors, e.g. $\hat{\mathbf{A}} = \hat{A}_1 \mathbf{i} + \hat{A}_2 \mathbf{j} + \hat{A}_3 \mathbf{k}$ and $\hat{\mathbf{B}} = \hat{B}_1 \mathbf{i} + \hat{B}_2 \mathbf{j} + \hat{B}_3 \mathbf{k}$ is defined as

$$\hat{\mathbf{A}} \hat{\mathbf{B}} = -\hat{\mathbf{A}} \cdot \hat{\mathbf{B}} + \hat{\mathbf{A}} \times \hat{\mathbf{B}}, \text{ (17)}$$

we have,

$$\hat{\mathbf{L}}' = q \hat{\mathbf{L}} q^{-1} = (\hat{q}_1 + \hat{q}) \hat{q}_1 - \hat{q}$$

$$= \hat{q}^2 \hat{\mathbf{L}} + \hat{q}_1 (\hat{\mathbf{L}} \cdot \hat{q}) - \hat{q}_1 (\hat{\mathbf{L}} \times \hat{q}) + \hat{q}_1 (\hat{\mathbf{L}} \cdot \hat{q})$$

$$+ (\hat{q} \hat{\mathbf{L}}) - \hat{q} (\hat{\mathbf{L}} \times \hat{q})$$

$$= \hat{q}^2 \hat{\mathbf{L}} + \hat{q}_1 (\hat{\mathbf{L}} \cdot \hat{q}) - \hat{q}_1 (\hat{\mathbf{L}} \times \hat{q}) + \hat{q}_1 \hat{\mathbf{L}} - \hat{q} \hat{\mathbf{L}}$$

Finally,

$$(\hat{q} \hat{\mathbf{L}} \hat{q}) = \hat{\mathbf{L}} (\hat{\mathbf{q}} \cdot \hat{\mathbf{q}}) - \hat{\mathbf{q}} (\hat{\mathbf{q}} \cdot \hat{\mathbf{L}}) = (1 - \hat{q}^2) \hat{\mathbf{L}} - \hat{\mathbf{q}} (\hat{\mathbf{q}} \cdot \hat{\mathbf{L}}). \text{ (19)}$$

From Eq. (20), it is clear that $\hat{\mathbf{L}}'$ is a dual unit vector. Furthermore, Eq. (20) can be expanded and rewritten in dual matrix form:

$$\hat{\mathbf{L}}' \Rightarrow \hat{\mathbf{L}}' = \begin{bmatrix}
2\hat{q}_1^2 - 1 & 0 & 0 \\
0 & 2\hat{q}_2^2 - 1 & 0 \\
0 & 0 & 2\hat{q}_3^2 - 1
\end{bmatrix}
= \begin{bmatrix}
L_1 \\
L_2 \\
L_3
\end{bmatrix}
+ 2\hat{q}_1 \begin{bmatrix}
0 & -\hat{q}_4 & \hat{q}_3 \\
\hat{q}_4 & 0 & -\hat{q}_2 \\
-\hat{q}_3 & \hat{q}_2 & 0
\end{bmatrix}
+ \begin{bmatrix}
2\hat{q}_2^2 - 1 & -2\hat{q}_1 \hat{q}_4 + 2\hat{q}_2 \hat{q}_3 & 2\hat{q}_1 \hat{q}_3 + 2\hat{q}_2 \hat{q}_4 \\
2\hat{q}_1 \hat{q}_4 + 2\hat{q}_2 \hat{q}_3 & 2\hat{q}_2^2 - 1 & -2\hat{q}_1 \hat{q}_2 + 2\hat{q}_3 \hat{q}_4 \\
-2\hat{q}_1 \hat{q}_3 + 2\hat{q}_2 \hat{q}_4 & 2\hat{q}_1 \hat{q}_2 + 2\hat{q}_3 \hat{q}_4 & 2\hat{q}_3^2 - 1
\end{bmatrix}
\Rightarrow \hat{\mathbf{L}}' = \begin{bmatrix}
1 - 2(\hat{q}_3^2 + \hat{q}_4^2) & 2(\hat{q}_2 \hat{q}_3 - \hat{q}_1 \hat{q}_4) & 2(\hat{q}_2 \hat{q}_4 + \hat{q}_1 \hat{q}_3) \\
2(\hat{q}_2 \hat{q}_3 + \hat{q}_1 \hat{q}_4) & 1 - 2(\hat{q}_2^3 + \hat{q}_1^3) & 2(\hat{q}_2 \hat{q}_3 + \hat{q}_1 \hat{q}_4) \\
2(\hat{q}_2 \hat{q}_4 - \hat{q}_1 \hat{q}_3) & 2(\hat{q}_3 \hat{q}_4 + \hat{q}_1 \hat{q}_2) & 1 - 2(\hat{q}_2^3 + \hat{q}_3^3)
\end{bmatrix}
\Rightarrow \hat{\mathbf{L}}' = \begin{bmatrix}
L_1 \\
L_2 \\
L_3
\end{bmatrix}
\text{or}
\hat{\mathbf{L}}' = [\hat{T}] \hat{\mathbf{L}}. \text{ (21)}
\[ \mathbf{R}_d(\theta, S) = \mathbf{R}(\theta, S) + \epsilon \mathbf{R}_0(\theta, S), \]  
where \( \mathbf{R}(\theta, S) \) and \( \mathbf{R}_0(\theta, S) \) are the real and dual matrix components of \( \mathbf{R}_d(\theta, S) \) given by

\[
\mathbf{R}(\theta, S) = \begin{bmatrix}
l^2(1 - \cos \theta) + \cos \theta & ml(1 - \cos \theta) - n \sin \theta & n(1 - \cos \theta) + m \sin \theta \\
ml(1 - \cos \theta) + n \sin \theta & m^2(1 - \cos \theta) + \cos \theta & mn(1 - \cos \theta) - l \sin \theta \\
nl(1 - \cos \theta) - m \sin \theta & mn(1 - \cos \theta) + l \sin \theta & n^2(1 - \cos \theta) + \cos \theta
\end{bmatrix}
\]

and

\[
\mathbf{R}_0(\theta, S) = \begin{bmatrix}
2ll_0(1 - \cos \theta) + (l^2 - 1)S \sin \theta & (lm_0 + ml_0)(1 - \cos \theta) - nS \cos \theta & (nl_0 + ln_0)(1 - \cos \theta) + mS \cos \theta \\
(lm_0 + ml_0)(1 - \cos \theta) + nS \cos \theta + (mlS - m_0) \sin \theta & 2mn_0(1 - \cos \theta) + (m^2 - 1)S \sin \theta & (mn_0 + nm_0)(1 - \cos \theta) - lS \cos \theta + (nmS - l_0) \sin \theta \\
(nl_0 + ln_0)(1 - \cos \theta) - mS \cos \theta + (nlS - m_0) \sin \theta & (mn_0 + nm_0)(1 - \cos \theta) + lS \cos \theta + (nmS + l_0) \sin \theta & 2lm_0(1 - \cos \theta) + (n^2 - 1)S \sin \theta
\end{bmatrix}
\]

\[ [\mathbf{R}(\theta, S)] \] is the well-known orthogonal matrix, which is used to rotate three-dimensional points about an axis that passes through the origin of the system of reference. For such rotations, the second component \( [\mathbf{R}_0(\theta, S)] \) of \( [\mathbf{R}_d(\theta, S)] \) is zero, which implies that there is no distinction between a point transformation and a line transformation.

Comparing the screw displacement expressed by Eqs. (20) and (21) we get the following:

The screw displacement expressed by Eq. (20) requires:

- thirteen multiplications/divisions and 4 additions/subtractions for construction,
- forty-four multiplications/divisions and 34 additions/subtractions for the transformation (Appendix B),
- six memory locations for storage.

While the screw displacement expressed by Eq. (21) requires:

- twenty-four multiplications/divisions and 30 additions/subtractions for construction (Appendix B),
- twenty-seven multiplications/divisions and 21 additions/subtractions for the transformation,
- eighteen real memory locations for storage, from which only six are independent (three for rotation and three for translation).

The above operations illustrate the computational cost for a sequential execution of the corresponding operators (Table 1). The reader who wishes to have more information concerning the comparative costs of the two transformations is referred to the work of Funda and Paul [9], where an extensive computational analysis of the screw transformations is presented for both sequential and parallel implementations. Although the dual 3 × 3 matrix provides a relatively efficient tool for screw displacing lines, it introduces a representational redundancy. Taking into consideration that in most existing hardware, the cost of fetching an operand from memory exceeds the cost of performing a basic arithmetic operation, the use of the dual orthogonal matrix should theoretically lead to inferior performance. However, there are applications (such as computer animation) where the dual matrix operator is several times faster than the corresponding dual quaternion (see Section 7 for details). In the following sections, we define the structure of a dual point, which is used to represent spatial line segments, curves and surfaces.

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Construction</th>
<th>Seq. Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \hat{L} ' = \hat{q}(\theta, S) \hat{L} \hat{q}^{*}(\theta, S) )</td>
<td>13 4 44 34</td>
<td>( \hat{L} ' = [\mathbf{R}(\theta, S)] \hat{L} )</td>
</tr>
</tbody>
</table>
4. Dual point definition

It is well known that a point in space can be determined by the intersection of two lines. Thus using Plucker coordinates a dual point is defined by a set of two dual unit vectors intersecting at the point of interest. In general, dual unit vectors intersect when the dual component of their dual scalar product vanishes. In the rest of this paper, dual points are constructed in such a way that the above intersection property is satisfied. Taken this into account a dual point is obtained as follows.

In Fig. 2, two intersecting unit line vectors \( \hat{u} \) and \( \hat{v} \) are shown \((v_0 \cdot u_0 = 0)\), which represent a dual point \( P^* = (\hat{u}, \hat{v}) \). Then \( r_0(OM) \) is the position vector from the origin to a point on the line defined by the dual vector \( \hat{u} \) such that \( r_0 \) is perpendicular to \( u \), while \( r_v(ON) \) is a position vector to a point on \( \hat{v} \) defined in a similar way with the vector \( r_v \). Using the definition of the dual part of a dual vector (Eq. (2)), we obtain

\[
\begin{align*}
\hat{r}_u &= u \times u_0 \quad \text{and} \quad \hat{r}_v = v \times v_0. \quad (25)
\end{align*}
\]

If \( r(OP) \) is the position vector from the origin to the point of intersection, then there exist scalars \( \alpha \) and \( \beta \) such that

\[
\begin{align*}
r &= r_v + \alpha v = r_u + \beta u. \quad (26)
\end{align*}
\]

To determine \( \alpha \) and \( \beta \) we can rewrite Eq. (26) as

\[
\begin{align*}
\alpha v &= (r_v - r_u) + \beta u \Leftrightarrow \alpha = (r_v - r_u) \cdot v + \beta u \cdot v \quad (27)
\end{align*}
\]

or

\[
\begin{align*}
\beta u &= (r_v - r_u) + \alpha v \Leftrightarrow \beta = (r_v - r_u) \cdot u + \alpha v \cdot u. \quad (28)
\end{align*}
\]

We define the scalar \( \mu = u \cdot v \). If \( \mu = 0 \) the axis of the two unit line vectors are intersecting at right angles, while if \( \mu = \pm 1 \) the two lines are parallel or coincide. Thus, by substituting \( u \cdot v \) by \( \mu \) in Eq. (27) and since \( r_u \cdot u = r_v \cdot v = 0 \) we get

\[
\alpha = r_u \cdot v + \mu \beta. \quad (29)
\]

Substituting Eq. (29) into Eq. (28), we get the following expression for \( \beta \):

\[
\beta = \frac{r_v \cdot u + \mu r_u \cdot v}{1 - \mu^2}. \quad (30)
\]

From Eqs. (29) and (30), \( \alpha \) is determined by

\[
\alpha = \frac{r_v \cdot v + \mu r_u \cdot u}{1 - \mu^2}. \quad (31)
\]

Finally, substituting \( \alpha \) and \( \beta \) in Eq. (25), the point of intersection of two dual unit vectors is given by

\[
\begin{align*}
r &= v \times v_0 + \frac{r_u \cdot v + \mu r_v \cdot u}{1 - \mu^2} v = u \times u_0 + \frac{r_v \cdot u + \mu r_u \cdot v}{1 - \mu^2} u. \quad (32)
\end{align*}
\]

From the above analysis, it is concluded that we can express any point-vector by the intersection of two dual unit vectors. This implies that a dual point has sufficient freedom to hold three point-vectors simultaneously. Thus we can take advantage of this extra freedom in order to store within the main structure further data such as a tangent vector, a normal vector, a curvature vector, etc. Furthermore, when we construct a dual point we take extra attention to satisfy the intersecting condition \((v_0 \cdot u_0 = 0)\). For example, let us suppose we want to define a dual point given a position vector \( r \) and one direction vector \( w \). The required dual point \( P^* \) can be defined using two dual unit vectors \( \hat{u} \) and \( \hat{v} \) which intersect at the point of interest at right angles \( (\hat{u} \hat{w} = 0) \). Thus, taking into consideration, the above analysis we set \( u = w \) and \( u_0 = r \times u \). It follows that \( v = u \times u_0 \) and \( v_0 = r \times v \). Finally, for the storage of a dual point eight memory locations are required.

5. Representation of geometric entities using dual points

5.1. Dual line segment definition

A line segment can be defined in several ways. However, a graphics system database stores the two endpoints of the line, as these two points are enough for the definition of any other characteristic of the line such as its slope or direction. By analogy, we can define these two endpoints by two dual endpoints. Using the position vectors of the two endpoints \( r_1 = (x_1, y_1, z_1) \) and \( r_2 = (x_2, y_2, z_2) \) of a line segment, the dual line segment can be represented by the corresponding two dual endpoints \( P_1^* = (\hat{u}_1, \hat{v}_1) \) and \( P_2^* = (\hat{u}_2, \hat{v}_2) \). Each of the two dual
endpoints is defined by two dual unit vectors, which intersect at right angles. The storage requirements of a dual line segment varies depending on the choice of these two dual endpoints. For example, the dual line segment shown in Fig. 3 defined using two dual endpoints, each one defined by two orthogonal intersecting dual unit vectors, requires 10 memory locations while the dual line segment shown in Fig. 4 defined using two dual endpoints that share a common dual unit vector requires 9.

5.2. Dual curve representation

A curve $C$ is defined as the set of points $x$ which satisfy a parametric equation $x = x(s)$, where $s \in \mathbb{R}$. If $s$ is the arc length then $x(s)$ is called the natural representation of $C$ [23]. The orthonormal triple $(t, n, b)$ is called the geodesic trihedron of the indicatrix $x = x(s)$ and defines all the geometric characteristics at any point $x(s)$ of $C$ (Fig. 5). $t$ is the unit tangent vector, $n$ is the first normal vector and $b$ is the second normal vector of $C$ at $x(s)$.

A dual curve is defined as the set of dual points $x^* = x^*(s) = x^*(\hat{u}(s), \hat{v}(s))$ derived by

$$\hat{u}(s) = u(s) + zu_0(s) = t(s) + e x(s) \times t(s),$$

$$\hat{v}(s) = v(s) + ev_0(s) = n(s) + e x(s) \times n(s).$$

(33)

Thus, taking into consideration that $b(s) = u(s) \times v(s)$, we have stored into the main representation all the geometric characteristics which are uniquely defined along the curve $C$. It is clear that the above representation of a curve using dual points is not unique, which means that one could utilize dual points to define a curve using alternative data.

5.3. Dual surface representation

A surface $S$ in space is defined by a unique parametric equation $x = x(u, v)$, where $u, v \in \mathbb{R}$ [23]. The tangent vector along the $u$-parametric curve is defined by the partial derivative $x_u(u, v) = dx/du$, while $x_v(u, v) = dx/dv$ defines the tangent vector along the $v$-parametric curve (Fig. 6). The tangent plane at any point $x(u, v)$ of $S$ is defined by the directions of these two tangent vectors, while the unit normal vector of $S$ at $x(u, v)$ is given by

$$N = \frac{x_u \times x_v}{|x_u \times x_v|}.$$ (34)

We define a dual surface as the set of dual points $x^* = x^*(\hat{u}(u, v), \hat{v}(u, v))$, $u, v \in \mathbb{R}$, where

$$\hat{u}(u, v) = u(u, v) + eu_0(u, v) = \frac{x_u(u, v)}{|x_u(u, v)|} + e x(u, v) \times x_u(u, v),$$

$$\hat{v}(u, v) = v(u, v) + ev_0(u, v) = \frac{x_v(u, v)}{|x_v(u, v)|} + e x(u, v) \times x_v(u, v).$$

(35)

Thus, taking into consideration that $N = u(u, v) \times v(u, v)$ we have actually stored within the representation of $S$ its first fundamental form.
6. Scaling and view transformations

In this section, we describe scaling and view transformations of dual points. Our aim is to show that transformations which are frequently used with homogeneous points can be implemented with dual points too.

6.1. Scaling

Scaling is used to change, increase or decrease, the size of an entity or a model. In the case of point-vectors, uniform scaling about the origin of the system of reference is expressed by

\[ \mathbf{P}' = s\mathbf{P} = (sx, sy, sz), \]

where \( \mathbf{P} = (x, y, z) \) is a point-vector and the positive scalar \( s \) is the scaling factor. The resultant point-vector \( \mathbf{P}' \) is closer to or further from the origin depending on whether the scaling factor \( s \) is smaller or greater than one, respectively (Fig. 7). Thus, the size of the model is decreased in the former case and increased in the latter case. Uniform scaling can be applied in dual points too. If \( \mathbf{P}_H = (u_L, v_L) \) is a dual point, it is easy to prove that uniform scaling occurs when the moment components \( u_L \) and \( v_L \) of both \( u \) and \( v \) are scaled by \( s \) in the \( X \) and \( Z \) directions. Let, \( u_L' = u + s(u_0) \) and \( v_L' = v + s(v_0) \) then the new point \( r' \) of intersection between \( u_L \) and \( v_L \) is calculated using Eq. (32) as

\[
r' = v \times (sv_0) + \left[ \frac{u \times (su_0)}{1 - z^2} \cdot v + \frac{v \times (sv_0)}{1 - z^2} \cdot u \right], \]

where \( r \) is the point of intersection between \( u \) and \( v \).

6.2. View transformations

View transformations (or projections) are classified into two major categories: orthographic and perspective [24]. Orthographic (or parallel) projections are obtained when the distance of the center of projection from the projection plane is infinite. These projections preserve parallelism and actual dimensions, and they are very popular among engineers. Perspective projections are obtained when the center of projection lies in finite distance from the projection plane. Perspective projections add more realism to views but they do not preserve
parallelism, actual dimensions and shapes of the transformed objects.

In this section, we present the corresponding quaternions, which can be used to realize an orthographic or perspective projection when the representation of a geometric object is based on dual points.

### 6.2.1. Orthographic projections

A few indicative cases of orthographic projections are illustrated below.

**Right view:** The object is rotated by $-90^\circ$ about the $Y$-axis, then by substituting in Eq. (8) the corresponding dual unit quaternion is

$$Q_{\text{Right}} = dq(-90, 0) = \frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}j.$$  \hspace{1cm} (38)

Taking a closer look at the above quaternion, we notice that it expresses a rotation of $45^\circ$ about the $Y$-axis of the system of reference. According to Eq. (27), the remaining rotation by $45^\circ$ is provided by the conjugate dual quaternion $Q_{\text{Right}}$. This also holds for all the following orthographic transformations.

**Dimetric view:** The model is rotated by $\theta = 22.08^\circ$ about the $Y$-axis and by $\phi = 20.705^\circ$ about the $X$-axis. Substituting in Eq. (8), the corresponding dual unit quaternion is given by $Q_{\text{Dimetric}} = Q_{x}Q_{y}$, where

$$Q_{x} = \cos \frac{\phi}{2} + \sin \frac{\phi}{2}i,$$

$$Q_{y} = \cos \frac{\theta}{2} + \sin \frac{\theta}{2}j.$$  \hspace{1cm} (39)

Then $Q_{\text{Dimetric}} = 0.965516 + 0.176378i + 0.188377j$

$$+ 0.034412k.$$

**Isometric view:** The model is rotated by $\theta = \pm 45^\circ$ about the $Y$-axis and by $\phi = \pm 35.26^\circ$ about the $X$-axis. Substituting into Eq. (8), the corresponding dual unit quaternion is given by

$$Q_{\text{Isometric}} = Q_{x}Q_{y} = \cos \frac{\phi}{2}\cos \frac{\theta}{2} + \sin \frac{\phi}{2}\cos \frac{\theta}{2}i$$

$$+ \cos \frac{\phi}{2}\sin \frac{\theta}{2}j + \sin \frac{\phi}{2}\sin \frac{\theta}{2}k.$$  \hspace{1cm} (40)

Substituting a pair of angles $(\theta, \phi)$ into Eq. (40) the corresponding dual unit quaternion for isometric view transformations is derived.

### 6.2.2. Perspective projections

As already mentioned, in perspective projections the center of projection is located in finite distance from the projection plane as it is illustrated in Fig. 8, where the center of projection $C$ lies on the $Z$-axis of the world coordinate system. For simplicity, we assume that the projection plane $XsYs$ coincides with the $XY$ and its distance from $C$ is equal to $d$. When the point $P(x, y, z)$ is represented using Cartesian coordinates, it is trivial to derive the coordinates of the projected point $P_s(x_s, y_s, 0)$ as [24]:

$$x_s = x\left(\frac{1}{1 - z/d}\right) \quad \text{and} \quad y_s = y\left(\frac{1}{1 - z/d}\right).$$  \hspace{1cm} (41)

On the other hand, if instead of the point-vector $P(x, y, z)$ we have the dual point $P^* = (\mathbf{u}, \mathbf{v})$ then the required coordinates of the projected point $P^*_s = (\mathbf{u}_s, \mathbf{v}_s)$ can be derived using a screw displacement (without rotation) expressed by $\hat{q}_d(0, s)$. The axis of displacement and the distance $s$ are calculated as follows.

From the geometry illustrated in Fig. 8 it is concluded that

$$\mathbf{n} = \mathbf{n} + z\mathbf{n}_0 = \frac{\mathbf{P} - \mathbf{C}}{|\mathbf{P} - \mathbf{C}|} + s\mathbf{P} \times (\mathbf{P} - \mathbf{C}).$$  \hspace{1cm} (42)

From the similarity of the triangles $CP_2P_1$ and $CP_1P$ it is obtained

$$\frac{s_1}{s_1 + s} = \frac{d_2}{d_1} = \frac{d - z}{d}$$

$$\Rightarrow s = s_1\left(\frac{1}{d/z - 1}\right).$$  \hspace{1cm} (43)

where

$$s_1 = \sqrt{(d - z)^2 + x^2 + y^2}.$$  \hspace{1cm} (44)
Thus, using Eqs. (42) and (43) and adopting the notation of Eq. (8) we derive the dual unit quaternion for perspective projections as
\[
\hat{q}_0(0, s) = 1 + \frac{Is}{2} + \frac{ms}{2} j + \frac{ns}{2} k. \tag{45}
\]

Obviously, the above dual unit quaternion for perspective projections is not constant but depends on the spatial position of the projected dual point. Thus, more efficient formalisms for perspective projections of dual points should be explored by future research to determine the proper representation of this kind of projections using dual unit quaternions.

7. Application in computer animation

This section is divided in three parts. In the first part, a theoretical estimation of the computational cost required to animate a three-dimensional object represented by both homogeneous points and dual points is derived. In the second part, we discuss some implementation issues concerning the corresponding computational algorithms, in order to facilitate the reproduction of the results. Finally, in the last part, we present the experimental data and we compare them with the theoretical predictions.

7.1. Theoretical estimation of the computational cost

First, we have to decide upon the screw displacement form, which shall be adopted for implementation. There are two candidates: (a) the dual unit quaternion, and (b) the dual orthogonal matrix. Due to the minimal storage requirements and the relatively low cost for construction, dual unit quaternions seem suitable for computer animation. However, this advantage is verified only when we repetitively transform one dual vector. In any other case, the performance of the dual orthogonal matrix operator is superior since it requires a significantly lower number of additions/multiplications for the same transformation (almost half). The latter statement is empirically verified, since it has been measured that the dual orthogonal matrix operator was usually several times faster than the equivalent quaternion operator. Therefore, for the requirements of computer animation, the dual orthogonal matrix is more efficient form of line transformations with respect to the computational cost. The computational costs of various operations concerning the dual vector algebra are listed in Table 2. In Table 3, the cost for the construction of the equivalent 4 × 4 homogeneous matrix and for a transform operation are illustrated. Finally, in order to measure equivalent quantities, we compare each dual point with a triplet of point-vectors (one for the position and two of unit magnitude for the directions).

In computer animation, the spatial motion of a three-dimensional object is expressed through several successive transformations, which are produced by a smooth interpolation of a given set of orientations and positions. If \( M \) is the number of the required transformations and \( N \) the number of the points-vertices of the facets which constitute the polygonal mesh of the object surface and \( S_H \) and \( S_D \) the overall costs for animating the object using point-vectors and dual points, respectively, then the cost \( S_H \) is computed by the form
\[
S_H = (33M + 48N)m + (24M + 36N)a, \tag{46}
\]
where \( m \) and \( a \) denote one multiplication and one addition, respectively.

By setting the ratio between the number of transformations and points as \( \lambda = M/N \), and the ratio of the CPU cycles needed to perform one addition versus the cycles for one multiplication as \( \kappa = a/m \), Eq. (46) can be rewritten as
\[
S_H = [(33\lambda + 48) + (24\lambda + 36)\kappa]Nm. \tag{47}
\]
order to facilitate the repetition of the experiment by another researcher. Both representational models are implemented using an object-oriented programming language. In addition, the implementation of both representational structures (homogeneous and dual points) requires the same amount of memory. More specifically, we implemented homogeneous points by storing three point-vectors (CPoint3D) within one class (CPointEx), while dual points are implemented using one class (CDualPoint), which holds two dual vectors (CDual Vector). This kind of implementation ensures that both structures require the same memory (96 byte). These structures are briefly described in the following list.

class 

\[
\text{CPoint3D} \{ \\
\text{double } x, y, z, w; \rightarrow \text{The } (x, y, z, w) \text{ homogeneous coordinates of a point in space.} \\
\} \text{ (32 byte of memory)}
\]

class 

\[
\text{CPointEx} \{ \\
\text{CPoint3D position; } \rightarrow \text{The position vector.} \\
\text{CPoint3D } u; \rightarrow \text{The } u\text{-direction} \\
\text{CPoint3D } v; \rightarrow \text{The } v\text{-direction.} \\
\} \text{ (96 byte of memory)}
\]

class 

\[
\text{CDualNumber} \{ \\
\text{double } x, x_0; \rightarrow \text{The two components of a dual number } (x + x_0). \\
\} \text{ (16 byte of memory)}
\]

class 

\[
\text{CDualVector} \{ \\
\text{CDualNumber } L, M, N; \rightarrow \text{The three dual direction cosines of a dual vector.} \\
\} \text{ (48 byte of memory)}
\]

class 

\[
\text{CDualPoint} \{ \\
\text{CDualVector } u, v; \rightarrow \text{The } (\hat{u}, \hat{v}) \text{ coordinates of a dual point.} \\
\} \text{ (96 byte of memory)}
\]

The dual orthogonal matrix (Eq. (47)) requires 18 real parameters for storage, where only six of them are independent. This implies that the three dual column vectors \(\hat{\mathbf{u}}, \hat{\mathbf{v}}, \hat{\mathbf{z}}\) of the matrix are mutually orthogonal and thus storage is required for only two of them (e.g. \(\hat{\mathbf{u}}\) and \(\hat{\mathbf{z}}\)). However, in this convention it is necessary to restore the normal dual vector \(\hat{\mathbf{u}}\) when the full matrix is needed for a particular operation, a task which requires one dual cross-product and thus an additional computational burden of 18 multiplications and 12 additions. Therefore, we have chosen to explicitly store all three dual unit vectors. Expanding Eq. (47) and adopting the notation used in Eq. (7), the screw displacement of the dual vector \(\hat{\mathbf{L}}\) is obtained by

\[
\hat{\mathbf{L}}' = \begin{bmatrix}
\begin{align*}
n_x L + o_x M + a_x N + \varepsilon(n_{o_x} L + n_x L_0 + o_x M + a_x M_0 + a_{o_x} N + a_x N_0) \\
n_L L + o_M + a_N + \varepsilon(n_{o_L} L + n_L L_0 + o_M + a_M M_0 + a_{o_L} N + a_N N_0) \\
n_L + o_M + a_N + \varepsilon(n_{o_L} L + n_L L_0 + o_M + a_M M_0 + a_{o_L} N + a_N N_0)
\end{align*}
\end{bmatrix}
\]
where,
\[
\mathbf{n} = (n_x + \epsilon n_{o_x}) \mathbf{i} + (n_y + \epsilon n_{o_y}) \mathbf{j} + (n_z + \epsilon n_{o_z}) \mathbf{k},
\]
\[
\mathbf{d} = (d_x + \epsilon d_{o_x}) \mathbf{i} + (d_y + \epsilon d_{o_y}) \mathbf{j} + (d_z + \epsilon d_{o_z}) \mathbf{k},
\]
\[
\mathbf{x} = (x_x + \epsilon x_{o_x}) \mathbf{i} + (x_y + \epsilon x_{o_y}) \mathbf{j} + (x_z + \epsilon x_{o_z}) \mathbf{k}.
\]

The elapsed time for the completion of a particular operation is considered as a norm of the corresponding computational cost. This elapsed time is expressed by the number of CPU cycles required to complete the task divided by the number of CPU cycles per second. Thus, the norm of cost is given in seconds by

\[
\text{cost(operation)} = \frac{\text{CPU cycles}}{\text{CPU cycles per second}}.
\]

(51)

Obviously, the above norm of computational cost is independent of the utilized hardware and gives a very
Table 4

The results of the described experiment

<table>
<thead>
<tr>
<th>Object points</th>
<th>Elapsed time (s)</th>
<th>Gain (%)</th>
<th>Deviation from theoretical (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dual points</td>
<td>homogeneous points</td>
<td></td>
</tr>
<tr>
<td>210</td>
<td>57.343</td>
<td>67.406</td>
<td>14.93</td>
</tr>
<tr>
<td>410</td>
<td>217.348</td>
<td>246.189</td>
<td>11.71</td>
</tr>
<tr>
<td>550</td>
<td>372.581</td>
<td>418.446</td>
<td>10.96</td>
</tr>
<tr>
<td>700</td>
<td>647.932</td>
<td>722.278</td>
<td>10.29</td>
</tr>
<tr>
<td>1400</td>
<td>3680.171</td>
<td>4326.531</td>
<td>14.94</td>
</tr>
</tbody>
</table>

High accuracy. Finally, the corresponding ratio between dual points and homogeneous points costs for the application under study is

\[ \rho' = \frac{\text{cost(homogeneous points animation)}}{\text{cost(dual points animation)}}. \]  

(52)

7.3. Results

The described animation has been applied to six different objects constituted of 210, 410, 550, 700 and 1400 vertices, using a sixth generation × 86 machine under a multitasking environment. Each object motion is composed by a set of translations and rotations. Translations are carried out along a circular path around the center of the system of reference. Furthermore, each object is rotated about the axis defined by the circle tangent at the current object position by a fixed angle.

After extensive experimentation the ratio \( \kappa \) was measured to be about half of the unit (\( \kappa \approx 1/2 \)). Note that the most trustworthy results are those derived from the objects with the lower number of points since in these cases the cost of memory fetching is minimal. This ensures that we measure the performance of the actual method and not the overall performance of the computing hardware. All the experimental data are illustrated in the charts (a)–(e) of Fig. 10. Each graph shows the theoretical predictions computed by Eq. (49) and the experimental data calculated by Eq. (52). It is clear, that the theoretical predictions match the experimental data in all cases, since the former approaches asymptotically the latter with high accuracy. In fact, when we reach a steady state, the relative error between the theoretical predictions and the experimental data is 1.3, 1.51, 2.26, 3.04 and 2.26% for each test case, respectively. Obviously, the described animation was completed faster using the proposed representational model. More specifically, the gain using the dual points model is 14.93, 11.71, 10.96, 10.29 and 14.94%, respectively. All experimental results are listed in Table 4.

The perturbations at the beginning of each graph are attenuated as the number of points is increasing. This happens because in the case of small data sets (when \( \lambda \) takes small values) the execution time of each operation tends to zero making the ratio \( \rho' \) sensitive to perturbations since it expresses a quotient of the form 0/0. Taking into account that under a multitasking environment it is not possible for the applications to have total control of the CPUs power, these deviations are inevitably entered into the experiment. Finally, taking into consideration both the theoretical and the experimental data, we may conclude that the dual points model is well suited for this kind of computer applications since it provides a computationally more efficient formalism than the homogeneous points model.

8. Conclusions

In this paper, a new model for the representation and transformation of geometric entities using dual unit vectors and dual unit quaternions has been presented. This model introduces a new structure called dual point, which has sufficient degrees of freedom to hold three point vectors simultaneously. Using dual points we describe line segments, curves and surfaces in \( E_3 \) utilizing invariant geometric properties such as normal or curvature vectors. Rotations, translations and view transformations are naturally expressed using the concept of screw displacement, while scaling is accomplished utilizing the moment vector of each dual line. An extensive analysis of the minimal memory requirements of the dual points representational model and of the transform operator shows that the proposed model is well suited for a particular set of computer graphics applications. This has been verified by theoretical predictions and the experimental results for the case of computer animation.

However, further study is necessary in order to exploit all the advantages of the proposed representation in other fields of computer graphics such as constructive solid modeling, motion design using dual quaternion curves and the incorporation of the proposed model with rendering algorithms and the various illumination models. Furthermore, future research should exploit more efficient formalisms for scaling and perspective transformations. Another area of interest includes the application of the introduced structures in robotics and more specifically in the planning of trajectories for painting and welding.

Finally, it is not our aim to substitute the entire homogeneous model with the proposed dual points model.
Obviously, the homogeneous model is well suited for most applications concerning computer-aided design and computer graphics. Our purpose is to show that alternative representations can be more efficient for certain applications. Perhaps, even better representations could be realized utilizing the advantages of both representational structures. For example, since the dual points model is rather complicated to be a primary user’s choice, an interface between these two models could be implemented for converting homogeneous points into dual points and vice versa. Thus, depending on the required task, a potential system should be responsible for its execution using dual points, homogeneous points or both without the user’s interference.

Appendix A. Dual numbers and dual vectors

A dual number is an ordered pair of real numbers \(x\) and \(x_0\) combined in the form \(\hat{a} = a + \varepsilon a_0\), where \(\varepsilon\) is called the dual unit and has the properties

\[
\varepsilon 
eq 0, \quad 0\varepsilon = 0, \quad 1\varepsilon = 1 = \varepsilon, \quad \varepsilon^2 = 0.
\]

The operations of addition and multiplication are defined for any two dual numbers \(\hat{a} = a + \varepsilon a_0\) and \(\hat{b} = b + \varepsilon b_0\) as:

**Addition:** \(\hat{a} + \hat{b} = (x + x_0) + (b + b_0) = (x + b) + \varepsilon(x_0 + b_0)\).

**Multiplication:** \(\hat{a}\hat{b} = (x + x_0) (b + b_0) = zb + \varepsilon(xb_0 + a_0b)\).

Under the above addition and multiplications rules, dual numbers form a ring, which is denoted by \(D\).

If \(f(\hat{a})\) is a differentiable function of dual numbers, then by a Taylor series expansion we have,

\[
f(\hat{a}) = f(a + \varepsilon a_0) = f(a) + \varepsilon f'(a) + \frac{\varepsilon^2(\alpha_0)^2}{2}f''(a) + \cdots
\]

A dual number fully specifies the relative position of two lines in space when expressed in the form

\[
\hat{\theta} = \theta + \varepsilon d,
\]

where \(\theta\) is the projected angle between the two lines and \(d\) is the length of the common perpendicular to the two lines. The dual number \(\hat{\theta}\) is frequently called a dual angle. Thus, using the Taylor series expansion we can derive the trigonometric functions of dual angles as

\[
\sin \hat{\theta} = \sin \theta + \varepsilon d \cos \theta, \\
\cos \hat{\theta} = \cos \theta - \varepsilon d \sin \theta.
\]

A dual vector is an ordered triple of dual numbers \((\hat{a}_1, \hat{a}_2, \hat{a}_3)\). This defines an ordered pair of vectors \(\mathbf{A}\) and \(\mathbf{A}_0\) combined in the form \(\hat{\mathbf{A}} = \mathbf{A} + \varepsilon \mathbf{A}_0\), where \(\mathbf{A} = (x_1, x_2, x_3)\) and \(\mathbf{A}_0 = (x_01, x_02, x_03)\). The vector \(\mathbf{A}\) is called principal vector and the vector \(\mathbf{A}_0\) is called principal moment. For a depth study of the above quantities the reader is referred to \([3,4,6]\).

Appendix B. The calculation of spatial operations

The homogeneous transform operator effecting a rotation through an angle of \(\theta\) about an arbitrary spatial axis that passes through \(A(x_0, y_0, z_0)\) with direction cosines \((l, m, n)\) (which does not pass through the origin of the system of reference) and a translation through \(d\mathbf{P} = [dx, dy, dz]\) is given by the following matrix \([24]\):

\[
[M] = \begin{bmatrix}
1 & 0 & 0 & x_0 \\
0 & 1 & 0 & y_0 \\
0 & 0 & 1 & z_0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & -x_0 \\
0 & 1 & 0 & -y_0 \\
0 & 0 & 1 & -z_0 \\
0 & 0 & 0 & 1
\end{bmatrix},
\]

where

\[
[T] = \begin{bmatrix}
\hat{r}(1 - \cos \theta) + \cos \theta & m(1 - \cos \theta) - n \sin \theta & n(1 - \cos \theta) + m \sin \theta & dx \\
ml(1 - \cos \theta) + n \sin \theta & m^2(1 - \cos \theta) + \cos \theta & mn(1 - \cos \theta) - l \sin \theta & dy \\
nl(1 - \cos \theta) - m \sin \theta & mn(1 - \cos \theta) + l \sin \theta & n^2(1 - \cos \theta) + \cos \theta & dz \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Expanding Eq. (B1), the transformation matrix \([M]\) is written as

\[
[M] = \begin{bmatrix}
-R & -x_0[\hat{r}(1 - \cos \theta) + \cos \theta] - y_0[m(1 - \cos \theta) - n \sin \theta] - z_0[n(1 - \cos \theta) + m \sin \theta] + dx + x_0 \\
-R & -x_0[m(1 - \cos \theta) + n \sin \theta] - y_0[m^2(1 - \cos \theta) + \cos \theta] - z_0[nn(1 - \cos \theta) - l \sin \theta] + dy + y_0 \\
-R & -x_0[nl(1 - \cos \theta) - m \sin \theta] - y_0[nn(1 - \cos \theta) + l \sin \theta] - z_0[n^2(1 - \cos \theta) + \cos \theta] + dz + z_0 \\
0 & 1
\end{bmatrix}
\]
Thus, 24 multiplies and 10 adds are required to construct $[T]$. Overall — without duplicating any operation and ignoring multiplication by zero — 33 multiplications and 24 additions suffice to compose $[M]$. The storage requirements of $[M]$ are 16 memory locations, although only nine of them are independent: three for the position vector $(x_0, y_0, z_0)$, three for the translation $dP$, two for the axis of rotation $(l, m, n)$ and one for the angle of the rotation.

For the construction of the $3 \times 3$ dual orthogonal matrix $[\hat{T}]$ of Eq. (30), we need to compute three squares of dual numbers, six dual number products, nine dual number sums and another three additions for the diagonal elements. Thus, the total cost for the construction of $[\hat{T}]$ is 24 multiplications and 30 additions. A more detailed discussion of the spatial operators calculations can be found in [9,10,18].

References