A Fast Mesh Deformation Tool for Blender (sap_0556)

James Verrill*
University of Cambridge

Joan Lasenby†
University of Cambridge

1 Introduction and Related Work

Standard mesh deformation techniques can sometimes provide unsatisfactory results when attempting to automatically deform natural objects, e.g. humans. Many such recently developed techniques attempt to preserve some quantity (e.g. volume) via optimisation, e.g. [Huang et al. 2006].

[Wareham 2006] developed techniques for deforming meshes using the mathematical language of geometric algebra (GA). These methods produce a completely smooth interpolation between a number of general displacements (rotations and translations) giving a natural looking motion. We build upon this work by developing it as a plug-in script written in Python for Blender (http://www.blender.org). The tool allows anyone to select key points on an object, move them independently of each other and deform the mesh smoothly and naturally to match these movements.

The method can be faster than alternative approaches as it is not iterative and the method always results in a smooth surface.

2 Concept Overview

In conformal GA (a 5D representation of 3D Euclidean space) we can describe a rotation and translation by the exponentiation of a 5D rotor to produce a bivector (bivectors are grade 2 objects analogous to planes). Interpolation is achieved by interpolation of these bivectors (this can be linear, quadratic, use of splines etc).

By considering the vertices of a mesh to exist in a conformal bivector space, we can interpolate the movements of each vertex. For each key vertex that the user is controlling, we can then apply a certain portion of this movement to its neighbouring vertices, such that the whole object has a plasticine effect. If we have several key points, proportions of each movement can be added to all the points so that deformation of the mesh by moving several points is possible. A brief overview of the key equations follows.

If each key vertex has an associated rotor, \( R \), formed from the rotation and translation of the key point, we must take the ‘log’ of this, \( \ell(R) \), to produce the bivector we use for interpolation: 

\[
\ell(R) = ab + c _\perp n + c \parallel n
\]

where

\[
|ab| = \sqrt{|(ab)_2|} = \cos^{-1}(\langle R \rangle_0) \\
ab = \frac{\langle (R)_2 \rangle n \cdot c}{\sin(c ||ab||)}
\]

\( c _\parallel n = -\frac{ab \langle R \rangle_4}{||ab||^2 \sin(c ||ab||)} \),

\( c \parallel n = -\frac{ab \langle R \rangle_6}{||ab||^2 \sin(c ||ab||)}\)

The amount of each movement that is performed on each vertex is determined by some function (which can be changed) of the distance of that vertex to the control point. Standard distance functions available in the tool include, gaussians and inverse functions.

3 Results

The results of the deformation algorithm are shown in Figures 1 and 2. First, a simple rotation of a vertex results in the twisting of the plane around it, as would naturally be expected. Then a cube is deformed by moving several key vertices, twisting them around, and then allowing the cube to move to a shape that matches them.

![Figure 1: Rotation of a point](image1.png)

![Figure 2: Movement of several points on a cube](image2.png)

4 Conclusion

We have developed a demonstration tool which shows the abilities of the interpolation techniques described here when applied to deforming meshes. The tool produces smooth deformations and could be useful for 3D artists working with Blender to both create simple objects and deform existing objects. The methods may be extendible for use in automatic techniques for otherwise complex areas, for example, skinning/rigging skeletons from motion capture data.

References


*e-mail: jrv23@cam.ac.uk
†e-mail: jl@eng.cam.ac.uk