Snappets: A VR user interface for animation with bendable vectors

HAMISH TODD

Current animation software is extremely complicated and difficult for newcomers, having hundreds of buttons and menus to manipulate varied data structures. This project aims to create a virtual reality computer program, Snappets, with an intuitive graphical user interface while also offering the customizability of modern procedural animation. The design involves giving users direct control, with their hands, over structures in Projective Geometric Algebra; this creates a visualization of mathematically sophisticated tools including Poisson brackets, exterior algebra, geometric *flags*, and the Dual Quaternion exponential and logarithm. This has the potential to introduce even young children, not just to animation, but also to linear algebra.

ACM Reference Format:
Hamish Todd. 2024. Snappets: A VR user interface for animation with bendable vectors. 1, 1 (March 2024), 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

"Coding" consists primarily of working with symbols; this is currently true even in animation, where the data being worked with is usually quite visualizable. There is an almost-universal division of labour within computer animation based on this: there are *animators* (prefer visual interface) and *animation programmers* (comfortable with symbolic interface). Some systems, such as Unreal's blueprints, attempt to create less-symbolic programming for the benefit of animators, but these still require the data to be manipulated in the same way that ordinary code does, for example by splitting quaternions into "x", "y", "z", "w" scalars. This is despite quaternions being, ultimately, just rotations, so in some sense any child able to use a knife and fork, or lego technic, already has some experience, and internal picture, of them.

Inspired by the thinking of Kay and Papert [Kestenbaum 2005], the Snappets project starts from the premise that the user interface to animation and geometry should, as far as possible, capitalize on the intuitions that users have already developed for 3D euclidean space. To this end, it exposes mathematical objects visually, with the hope that a newcomer can learn to use them in a matter of minutes.

2 PROJECTIVE GEOMETRIC ALGEBRA

This project is based on the premise that *versors* of *Projective Geometric Algebra*, Cl(3,0,1), are fundamentally connected with the way humans (including laypersons) interact with the world - moreso than for any other mathematical object, including matrices, vectors, and in a pinch, even scalars.

Author's address: Hamish Todd, hamish.todd1@gmail.com.

ACM XXXX-XXXX/2024/3-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Fig. 1. The interface of a computer animation program, showing visual elements but also a huge amount of buttons (a tiny minority of the whole)

A good introduction to PGA is provided by [Sudgylacmoe [n. d.]], but briefly: PGA versors comprise planes, lines, points, rotations, translations, screw motions (handedness-preserving rigid motions), rotoreflections, and transflections (handedness-reversing rigid motions). Every *geometric object* in PGA (the points, lines and planes in 3D space) is simultaneously a transformation - a plane is a planar reflection, a point is a point reflection, and a line is a 180 rotation. Planar reflections are atomic within PGA, and everything else can be seen as a composition of them; a rotation is a composition of two reflections; a point reflection (which is the same thing as a point) is a composition of three reflections; a translation is also two reflections, etc.

The versors of PGA can be divided into two 8-float types, which are the core math data structures that Snappets uses under the hood: even versors (grade 0, 2 and 4) and odd versors (grades 1 and 3), essentially 2k-reflections, and 2k+1 reflections up to normalization. Another almost-equivalent term is "handedness-preserving" or "proper" isometry, or "Dual Quaternion" for even versors, and "handedness-reversing" or "improper" isometry, or "Flector" for odd versors. This is memory-efficient and a useful way to break problems down: to the author's knowledge there is no use, in computer graphics, for a sum of an odd versor and an even versor.

With regards to animation, the most obviously important PGA objects are even versors - these allow objects to be moved around ("rigidly") in a virtual environment. Transflections and rotoreflections (there is no such thing as a "screw-flection" in 3D) are relevant too, because it is common for objects to be mirror- symmetric - an example might be a bird flapping its wings. Planes and points are both also very useful: it is common to want to constrain an animated object by projecting its position (a point) onto a the floor (a plane). Planes, lines and points (blades) are useful for programming constraints and inverse kinematics.

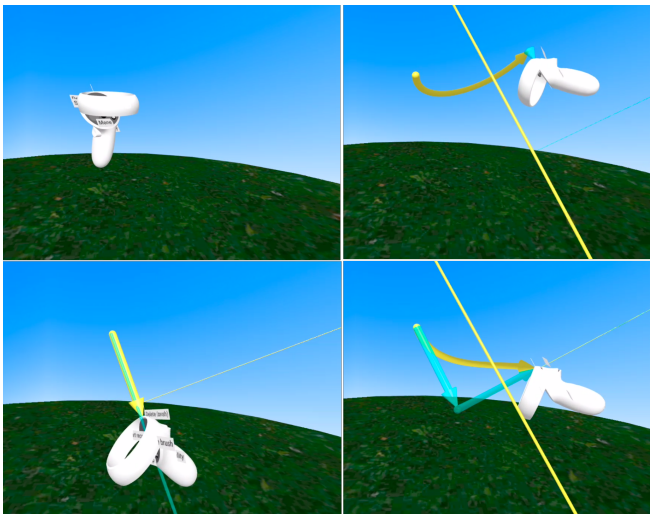

Fig. 2. Creating even versors with a single gesture

3 VERSOR VISUALIZATION AND SPECIFICATION FROM GESTURAL INPUT

To create an even versor in Snappets is a single-handed gesture: a particular button is pressed; the user moves their hand through the air; and the screw motion/rotation/translation is made which their hand has undergone. The most likely outcome is a screw motion; screw motions are in fact so likely that thresholding is added to make it so that a screw motion that is approximately a rotation is turned into a rotation, same with a translation. To find the "nearest" rotation and translation, the invariant decomposition [Roelfs and Keninck 2021] is employed.

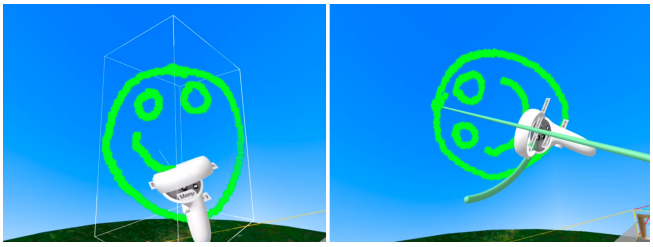

Fig. 3. Creating a 3D mesh and then changing its transform

Note that with rotations in place, one obtains lines "for free", since a line in PGA is the same object as a 180 degree rotation around the line.

The user can also create rotoreflections - this may not seem especially useful, until one becomes aware that points and planes are special cases of rotoreflections (a line is a special case of a rotation). To make one, the user grabs with both hands: the plane-point-arrow combination seen in figure 4 is created (possibly replacing an even versor). Instead of being defined using an initial and final hand position, the odd versor is instead calculated as the transformation that takes the user's left hand to their right hand (by definition a handedness-reversing transformation).

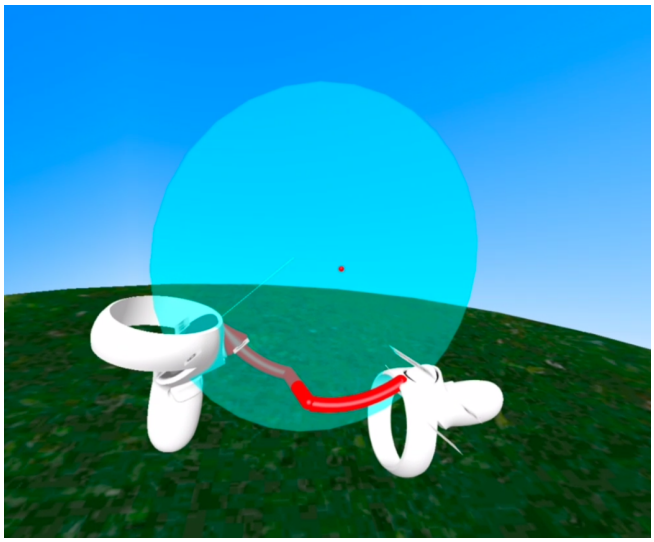

Fig. 4. Creating a rotoreflection

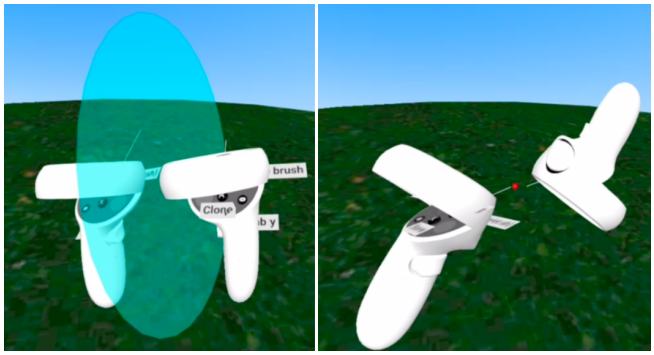

Fig. 5. Left: making a pure planar reflection (eg a plane); Right: making a pure point reflection (eg a point). Both are special cases of rotoreflections; a planar reflection has 0 grade-3 part, a point reflection has 0 grade-1 part

Rotoreflections themselves have some uses - however, the user is more likely to be interested in making points and planes - which are the special cases of rotoreflections where the rotation is by an angle of 180 degrees (for a point reflections/points) or 0 degrees (a planar reflection/plane). Again Snappets has thresholding to make these cases easier for a user to create, since in general the transformation taking a person's left hand to their right is a rotoreflection with probability 1.

Two more important special cases of odd versors are transflections, which appear when the rotation is by 0 degrees but the hands are separated from one another by a translation; and points-at-infinity, which is the special case of a transflection where the translation is by an infinite distance. A reflection followed by a translation by an infinite distance may seem again a fairly unlikely transformation to want to consider, but points at infinity are of interest for other reasons: for example, the derivatives with respect to time of ordinary points, moving around in space, is a point at infinity.

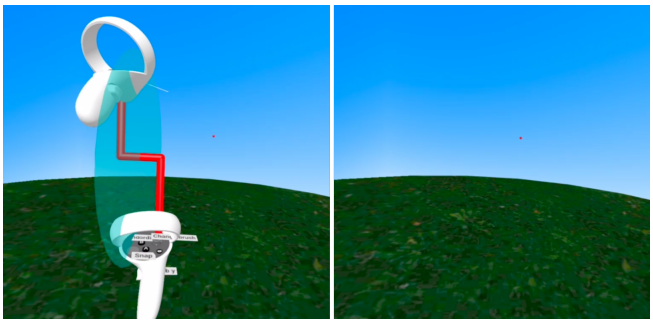

Fig. 6. Left: creating a transflections, eg a reflection followed by a translation; Right: a point at infinity (red, in the distance)

4 SNAPPING

An animated character is a set of rigid bodies ("bones") with relationships that are chosen by an animator and stored in memory using some representation. For example, they might choose that a hand's transformation is the arm's transformation followed by a translation (so the hand stays on the end of the arm); or they might enforce that a character's eyeballs always rotates to face where a particular object is currently placed.

The goal with snappets is to allow the user to choose any of these, i.e. to establish "declarative" relationships between bones. To enable them to do this, following Bret Victor [Victor [n. d.]], "snapping" is used; a specific example is shown in figure 7. Abstractly it means that when two versors are visible, and the user performs a gesture that creates another versor, they can press a button that will "snap" the versor so that it gains a geometric relationship (chosen from table 1) with the original two, a relationship that will be maintained thereafter.

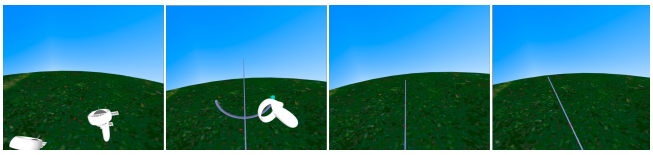

Fig. 7. An example of snapping: 1) two points A and B are visible 2) the user makes a gesture to create an even versor 3) after pressing the "snap" button, the even versor has been snapped to being the line joining the two points (it could also have been the translation taking one point to the other, but this one was closer) 4) when one of the points is moved, the line is moved, since it will now *always* be the join of the points - that is, a relationship has been "declared" using the PGA regressive product.

5 CONCLUSION

Aside from the fact that the program is based on snapping, the name "snappets" comes from the fact that the program is intended to be used to make puppet shows, eg live performances. These can be more engaging than "canned" animations such as Pixar movies, even in spite of the huge budgets of the latter. Performing with snappets also offers an advantages in comparison with ordinary puppet shows, and animation software, that it may well be engaging for audiences to watch animated characters being *created* - much of Instagram consists of watching people making things with their hands. Characters could be made in Snappets faster than traditional puppets, and more understandably than ordinary animated characters (because the controls are less complicated).

In terms of complexity-reduction, Snappets already has a head start over traditional computer animation software in that it is based in VR - when all the user has is a mouse, they cannot easily specify a rotation and translation simultaneously, whereas in VR this is trivial. Whether or not Snappets receives any widespread adoption, it should be expected that any future interface for animations incorporating virtual reality control ought to be made with a similar handling of screw motions.

Snappets' depictions of versors diverge somewhat from [Roelfs and Keninck 2021]. There, to visualize a translation, say, we are shown two (example) parallel mirror planes that the translation can be decomposed into. In Snappets' visualizations, a maximum of a single mirror plane is shown, which is for the case of rotoreflections, transflections, and planar reflections (not point reflections or points at infinity) where a plane is explicitly involved in the transformation. Instead of plane-pairs, there is the arrow - this induces a larger element of subjectivity/"gauging" than showing the planes, because we must make a choice of where the arrow should begin - from a mathematical standpoint this is utterly arbitrary, since all points in space are affected by transformations. For the sake of a program for presenting the mathematics of PGA and gauges, a fork of snappets is planned which will use the everything-is-mirrors visualization.

Together the translations, rotations, and screw motions are the "bendable vectors" referred to in the title. Mathematically they are dual quaternions/even versors. I propose to use the name "bendable vectors" because the unfortunate truth is that unless it says "vector" somewhere, computer graphics practitioners, and math teachers, are likely to feel like a system is completely unrecognizeable to them. Note that the special case of dual quaternions that model pure translations, eg when the arrow is straight, behave in most of the ways that engineers expect "vectors" in 3D space to behave, in the sense that their composition (geometric product) looks like vector addition. There has been debate over whether "vectors" in PGA should be planes or points, or indeed negative-square bivectors since the xyz part of a quaternion is sometimes called the "vector" part. Plausibly none of these is true - "vectors" should be identified with translations, eg scalar-plus-null-bivector.

ACKNOWLEDGMENTS

The author is indebted to Steven De Keninck, Pontus Granström, Lauren Davies, Alan Kay, and Jon Selig for productive and inspiring conversations.

REFERENCES

David Kestenbaum. 2005. The challenges of IDC: what have we learned from our past? *Commun. ACM* 48, 1 (2005), 35–38.

Martin Roelfs and Steven De Keninck. 2021. Graded Symmetry Groups: Plane and Simple. arXiv:2107.03771 [math-ph]

Sudgylacmoe. [n. d.]. *A Swift Introduction to Projective Geometric Algebra.* Youtube. https://www.youtube.com/watch?v=0i3ocLhbxJ4

Bret Victor. [n. d.]. *Stop Drawing Dead Fish.* Youtube. https://www.youtube.com/watch?v=ZfytHvgHybA

Table 1. The operations that can be "snapped" to, which are offered to the user. One goal, for now, has been to avoid norms and scalars; this is the reason for the somewhat awkward final row, where a scalar (distance or angle) is extracted from a 2-versor (translation or rotation) and immediately put into an exponential function

Intuitive description	Operation	Restriction on a = grade(A) and b = grade(B)	Output
Transform composition	AB	A and B both mixed grade	Versor
Transform application	$(-1)^{ab}ABA^{-1}$	No restriction	Versor
Projection	$(A \cdot B)B^{-1}$	No restriction	Versor
Midpoint / bisector / "equal mixture"	$\frac{A+B}{}$	a and b both even or both odd	Versor
Transform between	$1 + \overline{(1 + (A\tilde{B})^2)}$	$0 < a < 4; 0 < b < 4$	2-versor
Intersection/"Meet"	$A \wedge B$	$a + b < 4; 0 < a < 3; 0 < b < 3$	Blade grade $a + b$
Orthogonal-and-incident	$A \cdot B$	$\|a - b\| > 1$	Blade grade $\|a - b\|$
Join point	$A \vee B$	$1 < a < 4; b = 3$	Blade grade $a - 1$
Velocity	$A \times log(B)$	B even versor, $1 < a < 4$	Blade grade a
Rotation/translation from axis and angle/distance	$A^{\max(\text{distance(B)},\text{angle(B)})}$	$a = 2$, 2-versor B	2-versor