Berne University of Applied Sciences (BFH)

Department Engineering and information technology

BZG1310 - Object-oriented geometry

Documentation Project 2

Subject: Spinning cube with spheres on its edges –
An approach using geometric algebra

Student: Sven Osterwalder (ostes2@bfh.ch)

Date: December 23, 2013

Professor: M. Stampfli

1 Management summary

This dissertation is part of a project in the context of the subject *BZG1310: Object-oriented geometry* at Bern University of Applied Sciences.

The goal of this project is the implementation of a rotating cube which has a sphere on each edge using methods of Geometric Algebra (GA) within the field of computer graphics.

The goal could be reached and the task was implemented using C++ and Versor (libvsr).

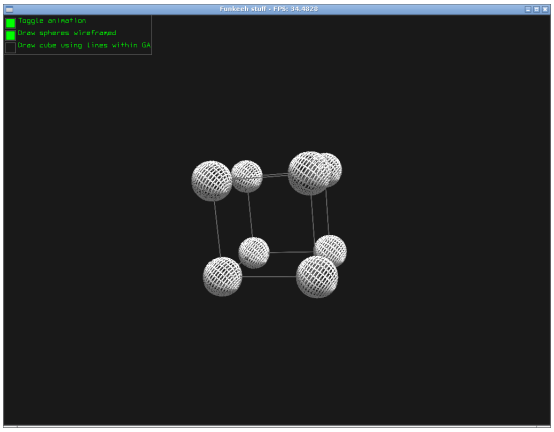

Figure 1: Screenshot of the running application

2 Implementation

The implementation was done using C++ and the Versor library.

2.1 Versor library

To cover the main tasks such as providing minimal application logic or handling input the Versor library was used as it provides drawing routines using an external library called GFX based upon OpenGL as well as a graphical user interface using another external library called GLV.

2.2 Geometric operations

As the geometric operations are the main aspect of the course BZG1310, this section shows how this aspect was realized within the application.

2.2.1 Implementation of the cube

For drawing the cube two methods were implemented:

- One using the *NCube* class built-in in Versor
- One using lines

The method using the *NCube* class has the advantage that the lines are finite, therefore they begin and end at certain, defined points in space.
When using lines there remains the problem that the lines do not end, given the implementation in Versor:

$$a \wedge b \wedge Inf(1)$$

2.2.2 Implementation of the spheres

The spheres were implemented using the *sphere* class from Versor:

$$Ro :: sphere(PositionOfSphere, SphereRadius);$$

2.2.3 Rotations

Versor provides two methods to implement rotations:

- Using the *rot()* method on objects
- Using generators

In this project only the first method was used to apply rotations. The rotations were realized using a Bivector whose coordinates are based on a timer[1] object:

$$bivRotation = Biv(sinf(fTime * 0.001f), 0, cosf(fTime * 0.001f));$$

This bivector is then applied to each vector which defines the coordinates of

[1] which is the current time in milliseconds minus the time when the application was started in milliseconds

a sphere respectively the edges of the cube:

$$Ro :: sphere(Vec(-0.5, 0.5, -0.5).rot(bivRotation), fSphereRadius);$$

$$Fl :: line(topLeftBack.rot(bivRotation), topRightBack.rot(bivRotation));$$

$$\begin{aligned} &gfx :: Glyph :: Line(\\ &\quad vecCube[edge.a].rot(bivRotation), \\ &\quad vecCube[edge.b].rot(bivRotation) \\ &); \end{aligned}$$

To provide a rotation for the spheres along their own axis resp. axes quite a lot of effort was made to realize this with operations within geometric algebra - sadly without any good results.
Versor always seems to need an offset for applying rotations, therefore a rotation with the basis $Vec(0, 0, 0)$ will not work. Neither when a translation is applied before or afterwards.
To achieve a rotation of the spheres along their own axes, a similar approach as used in the *GFX* library was used: Direct usage of the OpenGL command *glRotatef*.
This is not very sophisticated nor are operations from geometric algebra used but there seems currently no other way to achieve the task.

3 Conclusion

The *Versor* library seems to provide a quick way for making use of geometric algebra to realize graphical applications.
Nevertheless I personally think that the implementation or maybe even the subject is in a rather early stage for being a replacement for the conventional and yet used methods.

The library seems not to be up-to-date concerning the rendering as it makes use of conventional render methods such als *glVertex3f* and even GLUT.
Furthermore I think it would rather be hard or even impossible (at least at the moment) to implement things as for example Cube mapping.

At the end all gets realized using conventional methods within OpenGL. In my humble opinion it will provide an advantage to the currently used methods if the hardware supports it as well.
All in all it was an interesting experience on a not so explored subject.

4 Glossary

Bivector is a quantity in exterior algebra or geometric algebra that extends the idea of scalars and vectors[2]. 3

Cube mapping is a method of environment mapping that uses the six faces of a cube as the map shape within the field of computer graphics[3]. 4

Geometric Algebra (GA) is the Clifford algebra of a vector space over the field of real numbers endowed with a quadratic form [4]. 2

GFX is a generic graphics library (using OpenGL and OpenGL ES 2.0)[5]. 2

GLUT stands for OpenGL Utility Toolkit and is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system[6]. 4

GLV is a GUI building toolkit written in C++ for Linux, OSX, and Win32[7]. 2

OpenGL stands for Open Graphics Library and is a cross-language, multi-platform application programming interface (API) for rendering 2D and 3D computer graphics[8]. 2

Versor (libvsr) is a C++ Library for Geometric Algebra with built-in draw routines[9]. 2

[2] http://en.wikipedia.org/wiki/Bivector
[3] http://en.wikipedia.org/wiki/Cube_mapping
[4] http://en.wikipedia.org/wiki/Geometric_algebra
[5] https://github.com/wolftype/gfx
[6] http://en.wikipedia.org/wiki/OpenGL_Utility_Toolkit
[7] https://github.com/AlloSphere-Research-Group/GLV
[8] http://en.wikipedia.org/wiki/OpenGL
[9] https://github.com/wolftype/vsr2.0/