The Journal of Open Source Software

A Modularised Java library for 3D Euclidean geometry

Andy Turner[1¶]

1 CCG, School of Geography, University of Leeds ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Reviewers:

- @abhishekvp

Submitted: 14 March 2022
Published: unpublished

Summary

A three-dimensional (3D) Euclidean geometry Java library for modelling 3D changes in geometries over time. The library is modularised, based on openJDK 17, and dependends on ccg-math which in turn depends on both big-math and ccg-io.

Point locations in space are defined using 3D cartesian coordinates with orthogonal X, Y and Z axes that meet at the origin - a point <x,y,z> where x=y=z=0. The coordinate system is "right handed": so if X increases to the right of this page, and Y increases towards the top of this page, then Z increases out from the page, (see orientation for details of handedness, essentially there are two choices - a "left handed" system would have the Z direction of increase reversed). Right handed is thought to be more commonly used in both physics and geography.

Coordinates are either stored as Math_BigRational numbers - a subset of rational numbers with java.math.BigDecimal numerators and denominators, or Math_BigRationalSqrt - numbers that also represent square roots (calculated to a given Order of Magnitude precision as required).

More light weight geometries are in the uk.ac.leeds.ccg.v3d.geometry.light package. These use V3D_V which have three Math_BigRational components (nominally x, y and z) and extend V3D_VGeometry - an abstract class that holds a V3D_V "offset" which can be thought of as the translation of the geometry from the origin. - V3D_VPoint - is for representing points. A V3D_V instance gives the location of the point relative to the offset. - V3D_VLine - is for representing finite lines. V3D_V instances p and q give the location of the ends of the line relative to the offset (and can be the same). - V3D_VTriangle - is for representing triangles. V3D_V instances p, q and r give the location of each corner of the triangle relative to the offset (two or more of them can be the same). - V3D_VTetrahedron - is for representing tetrahedron. V3D_V instances p, q, r and s give the location of each corner of the tetrahedron relative to the offset (two or more can be the same).

Heavier geometries use offsets and relative locations stored as Math_BigRationalSqrt numbers. This allows for example a point to be located at x=y=z=sqrt(2). For these geometries there are methods which test for intersection, that return the geometry of the intersection, and that calculate the minimum distance between any two geometries. The methods that test if two geometries intersect involve no rounding. Calculating geometries of intersection may involve rounding (as coordinates are only a subset of algebraic numbers).

For any finite ccg-v3d geometry, a V3D_Envelope - a rectangular cuboid aligned with the axes can be computed. Such envelopes (also commonly referred to as Axis Aligned Bounding Boxes - AABB) assist in the computation of intersections or tests for intersection by quickly ruling out intersection between geometries that are in different X, Y, Z domains.

V3D_Geometry is an abstract class with a V3D_Vector offset, and an Order of Magnitude. Similar to V3D_V, this allows geometries to be readily translated and rotated. V3D_Geometry is extended to define the following finite and infinite geometries: - V3D_Point - a point geometry where a V3D_Vector gives the position relative to the offset. - V3D_Line - an infinite line geometry that passes through two defined points. - V3D_Ray - an infinite line

geometry that extends from a point in one direction through another point. - V3D_Plane - an infinite plane geometry. The plane is defined either by 3 points (p, q and r) that are not collinear or coincident, or by a normal vector and a point (where all points orthogonal to the normal vector at the point are on the plane). - V3D_LineSegment - represents a single continuous finite part of a line with non-zero length. Line segments are considered equal irrespective of the order of the end points. The centroid of a line segment can be computed and stored. - V3D_Triangle - a triangular part of a plane with a non-zero area. The three coplanar points {p, q and r} are the corners of the triangle. Triangle sides can be stored as line segments. The centroid of a triangle can be computed and stored. - V3D_Tetrahedron - represents a tetrahedron. Any selection of three points are not coplanar with the fourth, so this always defines a volume. A tetrahedron surface can be thought of as compising of four different triangles.

The following geometry collections are supported: - V3D_Points - for collections of points. - V3D_LineSegments - for collections of collinear line segments. - V3D_TriangleCoplanar - for collections of coplanar triangles. These do not have the constraint of all the component triangles having to intersect. - V3D_Tetrahedrons - for collections of tetrahedrons. These do not have the constraint of all the component tratrahedra being interconnecting.

Statement of need

Whilst open source 3D spatial tools exist, those which are known about rely on floating point arithmetic for which precision varies and results in uncertainties. Utilising more heavyweight numerical representations for coordinates allows for greater precision. Shapes can be approximated using points and straight line segments which connect these. Real spatial objects can be represented in this way, particularly around human scales. For generating models of what is happening on or near the surface and within the Earth, such representation offers a way to link data together for a Digital Twin Earth.

Everything is moving and changing at rates that vary with scale. Exploring patterns of change spatially is common to much scientific activity. 3D models and maps are important in telecommunications, transportation, architecture and engineering. This library could be applied to represent objects in 3D and model things physically involving motion, energy transfer and the dissolution and formation of objects. An important next steps is to build some basic models and visualise them. Some preliminary work has been done to develop 2D pictures of 3D spatial objects. My hope is to develop a basic model of the solar system which inputs data that is known about the masses, orientation and location of the Sun, the Moon and the rest of the solar system. I also plan to create a basic static model of Earth using available data, and on a smaller scale, I would like to develop a model of a glaciated region.

Future development

Ideally, a collective effort is wanted to develop the library sustainably. Issue reporting and tracking, feature requests and a more detailed roadmap for developing the functionality is wanted. Perhaps rather than creating a new community, what might best happen is to migrate the functionality to Apache Commons Geometry (see also: Apache Commons Geometry GitHub Repository). This parallel development appears to be developing similar arbitrary precision geometrical functionality.

Acknowledgements

The University of Leeds supported the development of the software. Externally funded research grants supported the development of some of the dependencies. Thank you Eric for the

(Obermühlner, 2020) library, in particular BigRational upon which Math_BigRational is almost entirely based.

References

Obermühlner, E. (2020). Big-math: Advanced java BigDecimal math functions (pow, sqrt, log, sin, ...) Using arbitrary precision. In *GitHub repository*. GitHub. https://github.com/eobermuhlner/big-math