University of Kent

Kent Academic Repository

Hibbin, Matthew and Rodgers, Peter (2026) *3D Euler diagram generation using a game engine: Work in progress*. In: Diagrammatic Representation and Inference. Lecture Notes in Computer Science . pp. 279-283. Springer E-ISBN 978-3-032-34178-5.

Downloaded from
https://kar.kent.ac.uk/115888/ The University of Kent's Academic Repository KAR

The version of record is available from
https://doi.org/10.1007/978-3-032-34178-5_29

This document version
Author's Accepted Manuscript

DOI for this version

Licence for this version
CC BY (Attribution)

Additional information
For the purpose of open access, the author(s) has applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising.

Versions of research works

Versions of Record
If this version is the version of record, it is the same as the published version available on the publisher's web site. Cite as the published version.

Author Accepted Manuscripts
If this document is identified as the Author Accepted Manuscript it is the version after peer review but before type setting, copy editing or publisher branding. Cite as Surname, Initial. (Year) 'Title of article'. To be published in ***Title of Journal*** , Volume and issue numbers [peer-reviewed accepted version]. Available at: DOI or URL (Accessed: date).

Enquiries
If you have questions about this document contact ResearchSupport@kent.ac.uk. Please include the URL of the record in KAR. If you believe that your, or a third party's rights have been compromised through this document please see our Take Down policy (available from https://www.kent.ac.uk/guides/kar-the-kent-academic-repository#policies).

3D Euler Diagram Generation Using a Game Engine: Work In Progress

Matthew Hibbin[0009-0002-2434-5313] and
Peter Rodgers[0000-0002-4100-3596]

University of Kent, Canterbury CT2 7NZ, UK

Abstract. We describe work embedding general 3D Euler diagrams using a game engine. We create closed surfaces using 3D meshes to represent sets and so indicate set intersections by mesh overlap. Our goal is to represent the underlying data accurately with all required set intersections present and without any additional intersections. This work in progress report details dual graph production, mesh creation, accuracy measurements, and subsequent improvements to the meshes that are intended to fix incorrect intersections. We also discuss future work to further improve the generation process.

1 Introduction

We explore the research question: Can game engines aid in the automatic generation of accurate 3D Euler Diagrams? We describe our work in progress to generate 3D Euler diagrams using the Godot game engine.

Research into 3D Euler diagrams is an area not fully explored. Our research intends to further showcase the utility of 3D Euler diagrams by developing a general generation tool. We intend to demonstrate how diagrams with poor readability in 2D space can be better visualised in 3D. We will produce techniques to allow easy interaction with 3D diagrams so that all their elements can be clearly observed. Furthermore, the utilisation of a game engine will demonstrate its applicability to diagrammatic visualization and interaction. Compared to traditional graphing software, a game engine gives us more interaction methods, a physics engine and in-depth manipulation control. Godot was chosen as it is a lightweight, open-source game engine with suitable functionality for our needs.

Regarding related work, 3D Euler diagrams were first explored in [1], where they were formally defined and compared to a 2D approach. The drawability of these diagrams was further explored in [2] where a graph representation was used as the basis for wellformed 3D Euler diagrams. This followed a 2D approach [3] where, from an abstract description describing the required set intersections, a dual graph is generated. This dual is then used to generate a 2D Euler diagram, with a general 2D generation algorithm given in [4]. The work of implementing similar methods and algorithms for 3D Euler diagrams is the purpose of our research project. We aim to introduce new generation methods for 3D representation.

In the remainder of this paper, we first outline related Euler diagram research. We then detail the current method implemented to generate 3D shapes from an abstract description, how we evaluate the diagram and make adjustments to improve accuracy. We include example images and a discussion of future work.

2 Diagram Generation Method

To implement the generation of 3D Euler diagrams, we chose the Godot game engine [5] due to its 3D capabilities. It provides easy surface mesh generation, including convex hull generation with the help of an external library [6] and camera control.

Dual Graph Production: Following the 2D approach of [4], we first generate a dual graph from an abstract description (such as "$\emptyset, A, B, AB$"). Each letter is a set and each element is a zone or required set intersection. This uses a 3D representation of a dual graph using Godot's 3D nodes [5]. The edges were created by drawing 3D lines [7]. We utilise the spring embedder algorithm to lay out the dual graph [8].

Array Mesh for Shape Generation: Godot allows the creation of custom 3D shapes via its array mesh type which draws polygons between a set of defined points. A 3D mesh is used for each set where positions of the nodes in the dual graph that contain the set letter are used for the points of the array mesh. However, there were some immediate problems as some meshes contained a very small or very large number of nodes. The small sets have minimal nodes and therefore have an insufficient number of points for array mesh generation. This was solved simply by adding some artificial points for the array mesh closely surrounding the nodes to give the shapes a better definition in the 3D scene.

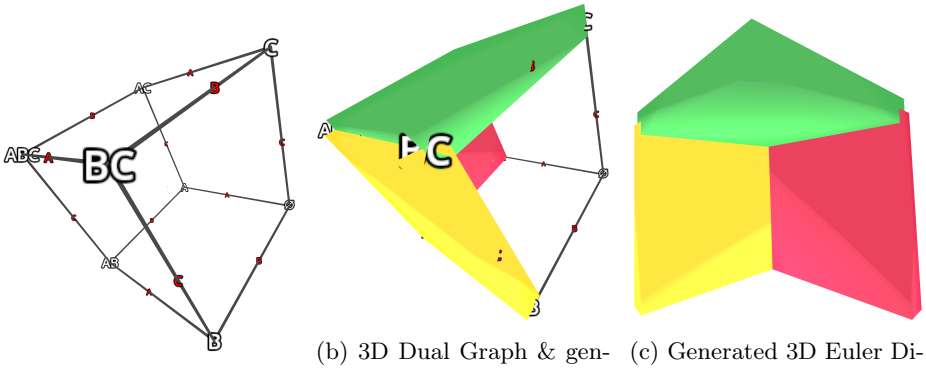
For the sets contained in a larger number of nodes, the dual graph has many nodes used for each array mesh where polygons would often overlap, not forming complete 3D shapes. This was solved by utilising a convex hull algorithm [9] which exposes the outermost points within a group of coordinates for a shape to be drawn around them [6]. However, the current implementation does not yet have support for non-convex meshes. This is something we plan to add in future (see section 3).

See Figure 1 for an example of output. It shows the dual graph in 1(a), the dual graph and the diagram in 1(b) and the diagram rotated compared to 1(b) in 1(c).

Correctness Detection and Improvement: After initial generation, the accuracy of the generated diagram is assessed. Inaccurate diagrams refer to those with missing zones in the abstract description or representing some zones that are not present in the abstract description. We build an abstract description of the generated diagram for comparison with the input abstract description. Our correctness test uses the points that were isolated by the convex hull algorithm for each array mesh. We find the meshes that contain a point, so indicating the zone which the point is contained within. As these are the points where the intersection of the array meshes are most likely to occur, the generated zones will

be detected. However some refinement is required, as the are rare cases where the intersection of meshes may not be on these points.

As well as being able to evaluate the success of the embedding method, the correctness assessment can also be applied to adjust the generated graph so that it better fits the original abstract description. This work is in its infancy but is showing some promising initial results. The design observes the results from the correctness assessment that specifies the missing and gained zones. For each gained zone, it locates each point used in the correctness assessment where this zone was observed and "shrinks" the involved array meshes by finding the centroid of the array mesh's convex hull points and then moving the violating point towards that centroid with the aim that it will remove the zone which is not required. Meanwhile, for each missing zone, the associated array meshes representing the sets that are a part of that zone are queried for each of the closest convex hull points. These points are then "expanded", with their positions moved towards each other. Whilst this system improves accuracy, it does not correct inaccuracies entirely. We are working on further additions and fine tuning.



(a) 3D Dual Graph

(b) 3D Dual Graph & generated 3D Euler Diagram

(c) Generated 3D Euler Diagram at 90°

Fig. 1: Output with the abstract description: $\emptyset, A, B, C, AB, AC, BC, ABC$

3 Conclusions

We believe the initial results have proven promising, as we have successfully generated 3D Euler diagrams. The dual graph production, the spring embedder algorithm, and tools to observe the diagram all work well. Running our correctness evaluation on a sample of one hundred generated Euler diagrams with a max of 3 sets and 6 zones, we found that 82% were observed to be accurate by the system. We also believe we have so far proven the feasibility of game engines being suitable for 3D diagrams.

Future work includes improving the correctness evaluation. The current design does not allow the detection of zones when the intersections of meshes do

not happen on the evaluated points. The testing of points beyond those used to generate the mesh should allow us to detect currently missed zones.

We intend to introduce concave shapes to support dual graph layouts that require non-convex set meshes. We will add more points to the array mesh that cause a "dip" in its geometry, causing a concave shape to form, so removing incorrect intersections.

Wellformedness of 3D Euler diagrams [2] will be addressed. This will examine the aspects of triple points, concurrency and duplicate curve labels that allow all abstract descriptions to be drawn. There are particular aspects of these in 3D that will be examined [1].

Looking further into the future, shape smoothing will be introduced thereby improving the readability of the diagrams. The array meshes from the generated diagram could be evaluated for any simple 3D shapes they may closely resemble and seeing if we can retain accuracy while changing their shape to more closely represent the chosen simple 3D shape. Whether this will be by adjusting the array mesh itself or replacing the mesh with Godot's in-built simple 3D shapes will have to be tested. Outside of shape smoothing, the readability of the generated 3D Euler diagrams can also be improved by removing the sharp edges of the meshes generated.

We will also be adding better user interaction with diagrams. We already have the tools to move the virtual camera freely around the diagram, but need improved solutions to see the encapsulated sets in three dimensions. One idea is to implement an "x-ray" view slider that will move a 2D "window" plane across the diagram and expose the interior of the diagram.

References

1. Peter Rodgers, Jean Flower, and Gem Stapleton. Introducing 3d venn and Euler diagrams. In *Proc. 3rd Int. Workshop on Euler Diagrams 2012*, volume 854, pages 92–106. CEUR-WS. org, 2012.
2. Jean Flower, Gem Stapleton, and Peter Rodgers. On the drawability of 3d venn and Euler diagrams. *J. Visual Languages & Computing*, 25(3):186–209, 2014.
3. Jean Flower and John Howse. Generating Euler diagrams. In *Proc. Diagrams 2002*, pages 61–75. Springer, 2002.
4. Peter Rodgers, Leishi Zhang, and Andrew Fish. General Euler diagram generation. In *Proc. Diagrams 2008*, pages 13–27. Springer, 2008.
5. Juan Linietsky, Ariel Manzur, and the Godot community. Godot docs - 4.5 branch. https://docs.godotengine.org/en/4.5/index.html, 2026. Accessed: 2026-02-12.
6. Matt Campbell and David Sehnal. MIConvexHull: A .net fast convex hull library for 2d, 3d, and higher dimensions. https://designengrlab.github.io/MIConvexHull/, 2019. Accessed: 2026-02-15.
7. betalars. Godot asset library - simple line renderer. https://godotengine.org/asset-library/asset/2563, 2024. Accessed: 2026-02-14.
8. Frank Simon, Frank Steinbrückner, and Claus Lewerentz. *3d-spring embedder for complete graphs*. BTU, Inst. of Computer Science, 2000.
9. C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Mathematical Software (TOMS)*, 22(4):469–483, 1996.