

Three-Dimensional Extended Mean Shift Edge Bundling

Daniel-Bálint Topor  

Aarhus University, Natural Sciences,
Computer Science,
Denmark

1 Abstract

2 Geometrical graph visualization is a nontrivial task when the graph is highly connected. To stop the
3 clutter and to see the details, we can deploy a technique called edge bundling, which is used to bundle
4 edges that are similar in length, direction, and orientation in space. 3D graph visualization is used
5 in many fields, but mostly in neuron science, where researchers are looking at neuron connectivity.
6 To accurately model brain activity, these edges can be ranked and the edges can have edge weights,
7 which helps to describe the brain in more detail.

8 Our solution for these highly complex tasks is to extend Mean Shift Edge Bundling (MSEB), a
9 force-directed edge bundling, and adding features—like edge weight integration, directed bundling,
10 minimizing edge whiplash—and make this algorithm, called Extended Mean Shift Edge Bundling
11 (EMSEB), state-of-the-art.

12 To verify our findings, we came up with new 3D quantifiable metrics, inspired by 2D metrics.
13 We tested our solutions on 10 different datasets, from social graphs to true brain activity. We have
14 found that we improved the original MSEB algorithm while having the same time complexity and
15 even better runtime with some clever optimization tricks.

2012 ACM Subject Classification Human-centered computing → Graph drawings

Keywords and phrases Graph visualization, edge bundling, 3D edge bundling, directed edge bundling,
aggregation, node-link diagrams, physical simulation

Acknowledgements I want to thank Hans-Jörg Schulz for mentoring me, Ágoston Róth for helping
me to port 2D directed edge bundling solution to 3D.

16 1 Introduction

17 In our day-to-day life, we often come across graphs, even if we do not realize them upfront
18 (for example, social connectivities can be interpreted as graphs). It is human behavior to look
19 into the details and try to explain why things are as they are. One way to understand graphs
20 is to visualize them. Visualizing graphs is a difficult problem where there are non-trivial
21 approaches.

22 One way of visualizing graphs is by straight edge node-link diagrams. The problem with
23 such solutions come when we increase in the number of edges. The visualization will suffer
24 from occlusion issues. Edge bundling is one method for decreasing occlusion in large or highly
25 connected datasets. One definition of edge bundling comes from Danny Holten [5] who said:
26 “[Hierarchical] edge bundling is based on the principle of visually bundling adjacency edges
27 together analogous to the way electrical wires and network cables are merged into bundles
28 along their joint paths and fanned out again at the end, in order to make an otherwise
29 tangled web of wires and cables more manageable”

30 With the rise of VR glasses, we can finally support immersive 3D visualizations which
31 was previously not feasible. This opens up a whole new dimension, compared to the 2D
32 solution [5]. To visualize graphs in 3D space, we need not only new hardware but new
33 software as well.



© Intellectual property of Daniel-Bálint Topor;
licensed under Creative Commons License CC-BY 4.0
Leibniz International Proceedings in Informatics



LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

34 Edge bundling is a reduction of clutters of a graph, which takes away from the truth and
 35 delivers a more comprehensible representation of the graph's connection. For this reason, it
 36 is quite a subjective field. We came up with new metrics, extending 2D metrics into 3D, to
 37 quantify each bundling output and have a quantifiable number next to the rendered bundled
 38 graph.

39 Disclaimer: This master's thesis was partially written by AI. AI was used to speed up
 40 implementation process and to improve the thesis text quality. The prompts are available in
 41 the Appendix A section.

42 2 Related works

43 Our solution is based on the family of algorithms known as 3D force-directed edge bundling,
 44 with the first successful and widely used implementation developed by Danny Holten and
 45 Jarke J. van Wijk [6].

46 Three-Dimensional Mean-Shift Edge Bundling (MSEB) [1] builds upon the foundations
 47 of Force-Directed Edge Bundling (FDEB) [6] and Kernel Density Estimation Edge Bundling
 48 (KDEEB) [7], combining the 3D applicability of FDEB with the numerical stability offered
 49 by KDEEB. In our work, we extend the MSEB algorithm by introducing more natural and
 50 intuitive edge paths, resulting in a solution that is expressive, effective, and computationally
 51 efficient. The original MSEB implementation focused on binary graphs, where edges are
 52 treated as either present or absent. While this simplifies the representation and allows for the
 53 selection of only the most relevant edges, it restricts the use of richer datasets that contain
 54 additional edge attributes, like edge weights. The authors noted that using binary data helps
 55 focus on the most important structural connections. However, we argue that discarding
 56 edge attributes or reducing the graph to binary form risks omitting valuable insights. To
 57 address this, we extend MSEB to support weighted edges, enabling the algorithm to account
 58 for the varying influence of connections. This allows for a more informative and nuanced
 59 representation of the underlying data.

60 Other edge bundling approaches have been proposed in the past. One of the state-of-
 61 the-start solutions up until the writing of this paper is Voxel-based edge bundling through
 62 direction-aware kernel smoothing (VBEB) [15] where they are using voxels instead and
 63 aggregate the edges instead of manipulating individual edges.

64 In the VBEB paper, it mentions the properties a good edge bundling algorithm should
 65 have:

- 66 ■ **Traceability (trace)**: individual edges can be traced when wanted, even if they bundled
 67 together (explicit bundles) and also represented as a bundle when needed,
- 68 ■ **Degree of freedom (dof)**: how many hyperparameters can you tune or need to be
 69 configured for a specific algorithm,
- 70 ■ **Directional bundling (dir)**: a bundling strategy that groups edges based on their
 71 directionality—specifically, edges that point toward the same target direction are bundled
 72 together, while edges originating from those target nodes (i.e., in the opposite direction)
 73 are excluded from the same bundle,
- 74 ■ **Dimensionality (dim)**: can bundle in 2D or even in 3D.
- 75 ■ **Scale (scale)**: VBEB is already known to be $O(n)$, where n is the number of edges, but
 76 the algorithm is based on voxel resolution and not purely on the mathematically defined
 77 geometric attributes of the given graph. Our algorithm has a complexity of approximately
 78 $O(n^2)$ which is acceptable.

79 ■ Adding to this list **Weighted edges (weighted)**: Having a way of considering weighted
80 edges in the calculation.

81 Extended Mean-Shift Edge Bundling (EMSEB) is capable of delivering all the above-
82 mentioned attributes: edges are traceable, relatively small number of hyperparameters (3),
83 directed bundling is supported, works for both 2D and 3D and the edge bundles are following
84 not just the mean position, but also the mean of the edge weights.

85 Our work and code are based on MSEB which is publicly available on GitHub¹. MSEB
86 lacks directed bundling (**dir**), weighted edges (**weighted**), edge traceability (**trace**) while
87 having the same number of hyperparameters (**dim**).

89 **3 Prior works**

90 In this section, we review previous work and identify the key challenges that motivated our
91 improvements. EMSEB's foundation can be traced back to MSEB and V. Singh's master's
92 thesis [11], which introduced several important enhancements to the original algorithm.

93 One of the most significant advancements was the refinement of the weighting mechanism.
94 The original MSEB algorithm was designed for unweighted (binary) graphs. In cases where
95 edge weights were present, it would handle them by duplicating edges according to their
96 integer weight, adding the same edge weight w times. This approach not only introduced
97 redundancy but also made the algorithm inefficient or even impractical for graphs with large
98 or continuous edge weight values.

99 V. Singh addressed this limitation by directly incorporating edge weights into the computation
100 of intermediate point positions. This improvement allowed for a more accurate and
101 scalable handling of weighted graphs, significantly enhancing the algorithm's efficiency and
102 applicability.

103 **3.1 Compatibility Measure and Compatibility Weighting**

104 The first paper to introduce these measures were Danny Holten and Jarke J. van Wijk [6].
105 In their work, they used four criteria to calculate a compatibility score between two edges, a
106 number between 0 and 1. Each geometric criterion is a number between 0 and 1 and is a
107 factor of the overall score. If the compatibility score is above the given threshold, we can
108 bundle the two edges together. The criteria are the following:

- 109 ■ **Angle compatibility (c_a)**: edges that are more or less parallel to each other should be
110 bundled together,
- 111 ■ **Scale compatibility (c_s)**: edges that are length wise similar should be bundled together,
- 112 ■ **Position compatibility (c_p)**: edges that are close to each other in space should be
113 bundled together,
- 114 ■ **Visibility compatibility (c_v)**: some edge pairs that have the desired attributes above-
115 mentioned still should not be bundled together. If they are parallel, have the same size,
116 and are relatively close to each other but are shifted apart, for example, opposite sides of
117 a rhombus, the edges should not be bundled. For more explanation, please read Danny
118 Holten's and Jarke J. van Wijk's paper [6].

119 The compatibility score is therefore calculated by combining these mentioned criteria into
120 one score: $c = c_a * c_s * c_p * c_v$. In addition, using the compatibility measure for thresholding,

85 ¹ <https://github.com/NeuroanatomyAndConnectivity/brainbundler>

XX:4 EMSEB

121 by the work of V.Singh, he postulates that this value can also introduce an improvement
122 factor to the mean calculation. Specifically, points on edges with higher compatibility scores
123 should have a greater influence on the mean than those barely meeting the threshold. While
124 Singh's findings indicate that this approach can lead to visual enhancements and improved
125 output quality, the observed benefits are relatively modest.

126 3.2 Weighted MSEB Algorithm

127 From V.Singh's master's thesis, we are introducing two new terms that accommodate the
128 improvements: 'inner weighting' and 'outer weighting'.

129 Compared to the original MSEB algorithm, which relies only on geometric proximity
130 and assumes that all edge weights are identical, the Extended MSEB algorithm (EMSEB)
131 introduces a weighted mean calculation when determining the mean location of the points.
132 This means that the new mean location is weighted both by proximity ('inner weighting')
133 (MSEB's implementation) and also by weight of the edges being compared ('outer weighting').

134 The forces between a pair of points $p_i \in P$ and $q_j \in Q$ for edges P and Q , respectively,
135 are determined by a Gaussian kernel and factoring the edge weight of Q . The formula for
136 the weighted contribution is:

$$137 \quad mean_w_{ij} = \frac{e^{-\frac{\|p_i - q_j\|^2}{2*bell}}}{weight(Q)} * comp(P, Q)^2 \quad (1)$$

138 where

- 139 ■ $\|p_i - q_j\|^2$ is the Euclidean distance between the two points.
- 140 ■ $bell$ is a Gaussian kernel parameter (typically set to 5 [1, 11]).
- 141 ■ $weight(Q)$ is the edge weight of Q .
- 142 ■ $comp(P, Q)$ is the compatibility score between edges P and Q , used as compatibility
143 weighting 3.1.

144 The new position for each intermediate point p_i is computed as the weighted average of
145 all potential influences:

$$146 \quad f_i = \frac{1}{\sum_j mean_w_{ij}} * \sum_j mean_w_{ij} * q_j \quad (2)$$

147 This equation ensures that edges with higher compatibility, bigger edge weights and closer
148 spatial proximity contribute more to the resulting mean.

149 Moreover, we are not only updating the position of the point of an edge this simply.
150 V.Singh's work also added a notion of 'inertia' based on the edge's own weight. The mean-
151 shifting operation is also influenced by the weight of its own edge. This approach draws
152 inspiration from Newtonian conceptualization of gravity, where the force exerted on object
153 A due to object B is directly proportional to the product of their respective masses. We
154 call this 'outer weight'. While the 'inner weight' gives general direction of the point, the
155 'outer weight' incorporates the edge weight's significance, shaping both the local and global
156 structure of the resulting bundle.

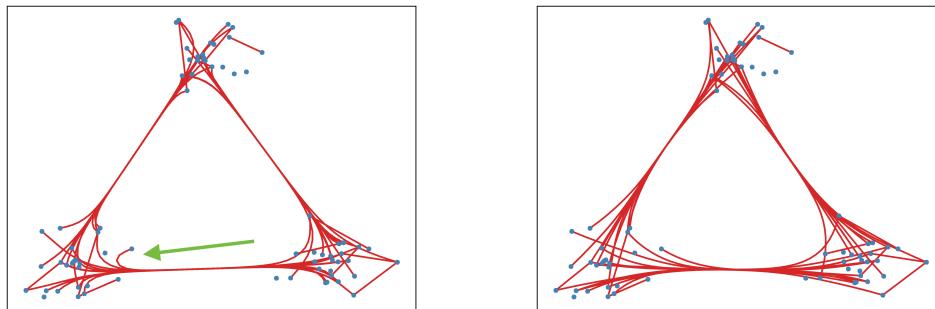
157 3.3 Edge Depth Weighting

158 Edge whiplash is a visual artifact which affects the visual appearance of the edge. The visual
159 artifact is characterized as edges curving backward to meet their origin or destination points,
160 resulting unnatural look, additional displacement and sharp bends which does not help to
161 reduce the visual clutter. This effect can be seen in figure 1, pointed out by the green pointer.

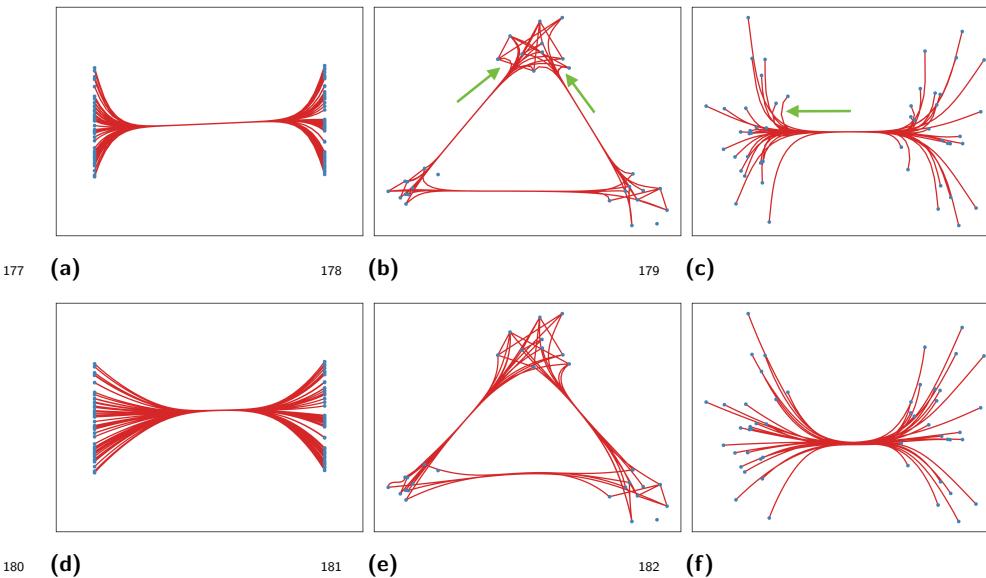
¹⁶² We observed these edge whiplashes in the output of both the original MSEB and the
¹⁶³ EMSEB algorithm. Adding the ‘inner’ and ‘outer’ weights helped to reduce it but did not
¹⁶⁴ eliminate the phenomenon completely.

¹⁶⁵ To correct the effect, V.Singh[11] introduced weighted correction based on edge ‘depth’.
¹⁶⁶ It is a new ‘weight’ where it is maximized at the center points of an edge and progressively
¹⁶⁷ diminished symmetrically in both directions away from it. This new edge depth weight
¹⁶⁸ enhances flexibility of the edge’s central region, allowing more adaptive deformation, while
¹⁶⁹ reinforcing the rigidity of segments which are closer to the start and end nodes.

¹⁷⁰ By applying ‘inner’, ‘outer’ and edge depth ‘weight’, we achieved a smoother and more
¹⁷¹ natural edge paths which can be seen in figure 2, where we compare the outputs of the
¹⁷² original MSEB algorithm and our solution.



¹⁷³ **Figure 1** Left: MSEB algorithm’s output, right EMSEB algorithm’s output with ‘inner’ and
¹⁷⁴ ‘outer’ weights. We can observe that how much the better ‘outer weight’ helps to maintain a pleasing
¹⁷⁵ solution, a solution where edge whiplash is mostly non-existent. (Figures taken from V.Singh’s
¹⁷⁶ master’s thesis [11])



¹⁸³ **Figure 2** Comparison of original MSEB (top row) and EMSEB (bottom row) visual outputs on
¹⁸⁴ the same datasets from V.Singh’s master’s thesis [11].

185 **3.4 Fibviewer**

187 The GitHub solution of MSEB also includes a program (written in QT²) to visualize these
 188 graphs. It is called Fibviewer, and it is a lightweight program which generates a 3D model
 189 of the graph. For input it uses the standard legacy VTK file format, which our solution
 190 outputs.

191 **3.5 The much needed improvements**

192 V.Singh[11] work brought improvements to MSEB[1], but did not reach the state-of-the-art
 193 solution status. It still lacked the following things:

- 194 ■ **Directed Bundling:** The ability to bundle edges based on their directionality. This is a
 195 crucial feature which can help the user to distinguish and discover connection, general
 196 trends and information or connectivity patterns.
- 197 ■ **Proper Benchmarking:** While V.Singh's[11] work had benchmarking, it was lacking
 198 quality metrics.
- 199 ■ **Providing illustrative results for each introduced weight:** V.Singh's[11] work
 200 introduced lots of new weights to improve the algorithm. Some weights are used but it is
 201 unclear if the selected weights were selected purely on empirical measures or they were
 202 properly tuned.
- 203 ■ **Open Sourcing the codebase:** To facilitate global adoption, one of our core goals was
 204 to release an open-source implementation. The value of accessible software is evident in
 205 the impact of MSEB, which, at the time of writing, has been cited 47 times with many
 206 practical deployments. In contrast, the implementation of V. Singh lacked reliability, as
 207 it did not include proper input validation, public availability, runtime optimizations, or
 208 implementation-level enhancements necessary for robust and efficient usage.

209 Having reviewed the relevant background and existing methods, we now focus on the
 210 novel enhancements and findings introduced in this work.

211 **4 Method**

212 **4.1 Overview**

213 In this section, we present the algorithm and the methodology used to extend V.Singh's
 214 algorithm to accommodate these improvements. Our method builds upon the core structure
 215 of MSEB, maintaining its fundamental procedures. We focus on improvements made to the
 216 'Attract' function, which is executed iteratively across k cycles, each consisting of at least i
 217 iterations. Between cycles, we perform an edge subdivision process by inserting subdivision
 218 points p_1 to p_n . Unlike linear subdivision schemes, our approach adopts the non-integer
 219 growth strategy, originating from MSEB, increasing the number of segments by a factor of
 220 1.3^c , where c denotes the current cycle.

221 **4.2 Preprocessing**

223 To produce high-quality visualizations, it is essential to properly prepare the input data.
 224 The algorithm supports two primary input formats. The first is the legacy VTK³ format,

186 ² <https://www.qt.io/>

187 ³ VTK file format: (https://docs.vtk.org/en/latest/design_documents/VTKFileFormats.html)

which does not allow for explicit specification of edge weights, thus all edges are assigned a default weight of 1. The second option involves providing separate files for nodes and edges. In the nodes file each line defines a node with its corresponding x , y , z coordinates, with numbering of the nodes starting from 0. In the edges file each connection/edge is marked by a row of three values: from i node to j node with an associated weight w . If no weight w is given, we use a default weight of 1.

After reading the input, we process the weights in the following way:

1. **Negative weights:** if there are negative weights, we flip the direction of the edge ('from node' with the 'to node') and take the absolute value of the weight.
2. **Adding 1:** We add 1 to the existing edge weights to assure that all the weights are in the range of $[1, \infty)$. This will avoid unwanted side effects when we use the edge weights in our calculations (dividing with tiny values).

4.3 Completing the compatibility Measure function

We are going to save the angle compatibility score if directed bundling is chosen, which indicates if two edges are parallel or anti-parallel (parallel but opposite direction) with each other. This is going to be used only when we are bundling with directionality.

4.4 Edge Momentum

To accelerate the convergence of our results, we introduced the point-based momentum, which we call edge momentum. By taking a fraction of the last asserted force on a specific point and adding it up with the newly calculated force, we 'award' the point's direction if it goes in the right direction by amplifying it even more, and we reduce point oscillations if the point moves back and forth.

To select the best β (fraction value) we ran a grid-based hyper-parameter search algorithm on 5 different datasets. We ran the edge bundling until the total force used to move the points was less than λ , our convergence constant. We ran the algorithm for each predefined β value and tracked which β value the bundles converged the fastest. To determine the best β , we summed up the total runs per β parameter and selected the β that had the least total runs.

From the conducted test, the value that converged in the least steps was $\beta = 0.8$. The plotted affirmation can be seen in figure 3.

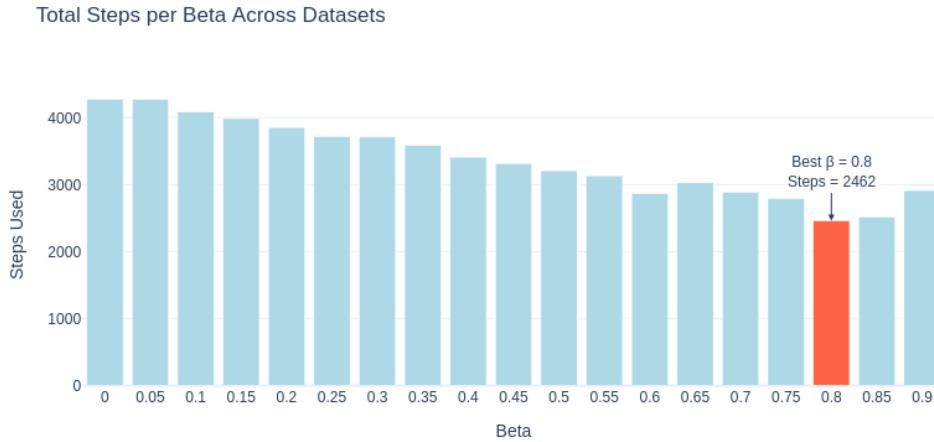
4.5 Tracking Point Movements

The algorithm subdivides the edges into segments by a non-linear factor of 1.3^c where c is the current cycle and floor this value to get an integer value.

In early cycles, it is possible that the number of segments stays the same from cycle to cycle. When that happens, the layout might already be visually stable, meaning that the points on the edges do not need to move any further. Running more iterations in this case would waste time, as the forces acting on the points are either very small or zero.

To handle this, we introduce a threshold value λ (convergence constant, default: 0.0001). After each iteration, we check the total amount of movement applied to the points. If it is below λ , we consider the layout stable and skip the rest of the iterations in that cycle, continuing with the next one.

This optimization is not present in the original version of the algorithm or in V. Singh's implementation. In those, the algorithm keeps running all iterations regardless of whether



255 **Figure 3** Hyperparameter search for β : convergence speed for three datasets.

269 anything is actually moving. By skipping unnecessary work, we can greatly reduce runtime,
270 even though the overall time complexity stays the same.

271 4.6 Directionality

272 Inspired by the work of David Selassie, Brandon Heller and Jeffrey Heer from the paper:
273 ‘Divided Edge Bundling for Directional Network Data’ [10], where they assert an opposite
274 force (pushing force) on edges which travel in anti-parallel direction, to create ‘highway’-like
275 bundling in 2D, we tried to do the same but in 3D.

276 The following steps can be used to add directed bundling feature to existing 3D force-
277 directed edge bundling algorithms.

278 4.6.1 Modifications

279 If directed bundling is selected, several key modifications are done to the algorithm:

- 280 1. **Compute and store the cross product:** mentioned in subsection 3.1, we are going to
281 save the angle compatibility (or in other names: cross product) value to determine which
282 edge pairs are parallel or anti-parallel with each other.
- 283 2. **Separate bundling for parallel and anti-parallel edges:** at the bundling stage we
284 are going to bundle the parallel and anti-parallel compatible edges separately based on
285 positive or negative angle compatibility measure. A positive direction factor indicates
286 parallel edges while a negative value indicates anti-parallels. This step does not insure
287 separated ‘lanes’⁴, but helps us to create two different semantic bundles which will help
288 us in the next step.
- 289 3. **Adding lateral/‘repelling’ force:** After the bundling process, if two anti-parallel edge’s
290 points are farther than a specified distance, then there is no repelling force added. Other
291 hand, the repelling force is calculated in the following way for points $p_i \in P$ and $q_j \in Q$:

283 ⁴ If we have node A and node B and two edges, one going from A to B and the other B to A , we are
284 not going to bundle these edges together but if they are not bundled with other edges neither, the two
285 paths are going to be overplotted.

- 295 a. **Tangent vector:** We calculate \vec{T}_j which is just the direction between q_{j+1} and q_{j-1}
 296 ($\vec{T}_j = q_{j+1} - q_{j-1}$) and normalize it,
 297 b. **Perpendicular Normal Vector:** using the vector $(0, 1, 0)$ we create the perpendicular
 298 vector $\vec{N}_j = \vec{T}_j \times (0, 1, 0)$ and normalize it,
 299 c. **Scaling the normal vector:** We scale \vec{N}_j by 2% of the mean of the two edges
 300 compared (P and Q) and added this lateral force to the attracted point q_j .
 301 The new attracted point is equal to:

302
$$\text{repelled_direction} = q_j + N_j * ((p.length + q.length)/2)/50$$

303 The 2% of the length helps us to automatically adjust the lateral force based on the
 304 geometrical attributes of the graph. This repelling force is only added to one of the edge
 305 pairs, only for one of the bundle directions, making it asymmetrical, but closer resembling
 306 the undirected bundling.

307 4. **Final bundling:** Finally, we run the bundling algorithm one more time, to smoothen
 308 any bumps and detachments that we introduced with the repelling force.

309 .
 310 We choose to bundle parallel and anti-parallel edges for the reason to make the visualization
 311 resemble the normal EMSEB. If the anti-parallel lines are bundling on a different track, far
 312 away (2% from the average length) from the parallel lines, we don't even need to separate
 313 them apart, thus keeping the force directed looks and nature.

314 In the implementation, there is a variable called `lane_width` which is set to 1. This
 315 variable is there for flexibility purposes. It was there in the Brandon Heller and Jeffrey Heer
 316 paper [10] as a constant l . In their paper, this constant was set to 5, but because of different
 317 datasets, in our case it 'blew' up the visualization with unnecessary and disturbing high
 318 forces when the node's coordinates were normalized in the $[0, 1]$ intervals. We adapted the
 319 self-regulating average length method to scale this force.

320 4.7 Edge Traceability

321 Edge traceability refers to the ability to identify and track which individual edges participate
 322 in which bundled structures. This is essential for post-bundling analysis, including filtering,
 323 interaction, and interpretation of connectivity patterns.

324 While the original implementation of the MSEB algorithm did not provide explicit support
 325 for edge traceability, we extended the system with a custom mechanism for recording per-edge
 326 bundle assignments.

327 We implemented a 'writeBundles()' function that traverses all edge pairs using a compatibility
 328 threshold and, when directionality is enabled, a consistent directionality check using
 329 the dot product of edge directions. Each edge is assigned to at most one bundle, preventing
 330 duplication across multiple bundles. For each bundle, we store:

- 331 ■ A unique bundle identifier.
- 332 ■ A list of all edge indices participating in that bundle.
- 333 ■ The direction label (e.g., forward or reverse) based on orientation consistency.
- 334 ■ The total weight of the bundle, computed as the sum of compatibility values among all
 335 its member edges.

336 This solution not only guarantees that each edge is uniquely associated with a single
 337 bundle, but also facilitates downstream traceability by logging these assignments in a human-
 338 readable text format. As a result, our extended bundling framework supports richer analysis
 339 of edge-level behavior within the bundled visualization.

340 **4.8 Edge Depth Factor justification**

341 V.Singh's[11] work introduced the concept of edge depth factor. This helped to reduce
 342 edge whiplash. The weight is maximized at the midpoint(s) of the edge and diminished
 343 symmetrically in both directions. For this, V.Singh used a quadratic function, seen in
 344 equation 3, where p is the number of points of an edge and i is the point's index on the edge.

345

$$346 \quad \text{edge_depth_factor} = \frac{4i(p - i)}{p^2} \quad (3)$$

347 This function is symmetric around the midpoint $\frac{p}{2}$ and it is concave. V.Singh did not justify
 348 why they used this specific function, so we wondered what other functions can be used to
 349 have these properties. We came up with:

350 ■ **Exponential function:** the function

$$351 \quad e^{-\alpha(i-p/2)^2}$$

352 ■ **Other Polynomial function:** we can use the same function as V.Singh and 'tune' it,
 353 making it any degree polynomial:

$$354 \quad \left(\frac{4i(p - i)}{p^2}\right)^{\text{poly_deg}}$$

355 , where poly_deg is a tunable parameter.

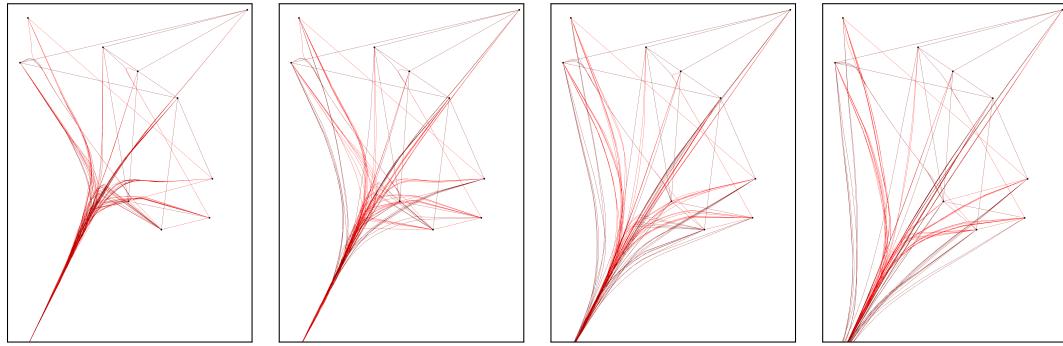
356 We tested the polynomial solution, where we ran multiple simulations with multiple
 357 poly_deg values (see figures 4). This sparked a general disagreement about which edge depth
 358 function is the best and why. While higher values of poly_deg stay closer to the original
 359 node-link representation, preserving the structure of the input data, they can work against
 360 the main goal of edge bundling, which is to reduce visual clutter by bending and grouping
 361 edges together. If we are looking for representative results we would want to bend the edges
 362 as minimally as possible in order to achieve the bundling. To truly determine the optimal
 363 value for poly_deg , new evaluation metrics would be needed. These could include measures
 364 of edge path distance (with straight lines as shortest paths), bundling quality, curvature or
 365 "bendiness," and possibly the presence of edge whiplash. Developing and validating such
 366 metrics is outside the scope of this work and would require further study and for these
 367 reasons, we set $\text{poly_deg} = 1$.

371 **4.9 Pseudo-code**

372 We provide the pseudocode for the EMSEB algorithm and a high-level explanation of it. Our
 373 algorithm is based on MSEB algorithm with the addition of weighted edge bundling, edge
 374 depth weighting, compatibility weighting and directionality. No changes were done to the
 375 helper functions found in MSEB.

376 For undirected bundling, the high level algorithm follows these steps (see pseudo-code 1):

- 377 1. The core procedure, **Attract**, takes three arguments: `edges` (a collection of edges), `b11`
 (the parameter for the Gaussian function), and `cthr` (a compatibility threshold).
- 379 2. It iterates over each edge in the `edges` collection.
- 380 3. For each edge, it retrieves its weight and number of subdivision points.
- 381 4. It then iterates over all interior points of the edge (excluding the endpoints).
- 382 5. For each point, an edge-depth factor is computed based on its relative position along the
 edge.



368 ■ **Figure 4** Same number of cycles, iterations, and threshold. Only the polynomial used on the
 369 edge depth factor varies. Smaller edge depth factor results in greater curvature. Polynomial values
 370 used in these figures are: 0.0, 1.2, 2.0, 2.8

- 384 6. Variables are initialized to accumulate the sum of weighted forces and the total weight.
 385 7. The algorithm then iterates over all other edges to compute pairwise compatibility with
 386 the current edge.
 387 8. If the compatibility exceeds the threshold, the corresponding point on the other edge is
 388 identified.
 389 9. The spatial distance between the two points is calculated.
 390 10. A weight is assigned based on the distance, the other edge's weight, and the compatibility
 391 function.
 392 11. The weighted contribution of the other point is added to the force accumulator.
 393 12. After processing all other edges, the accumulated force vector is normalized by the total
 394 weight.
 395 13. The resulting force is scaled using the current edge's weight and the edge-depth factor.
 396 14. The force acting on the current point is updated accordingly.
 397 15. Once all points of the current edge are processed, the algorithm moves to the next edge.
 398 16. After all edges have been traversed, the `applyForces` procedure is invoked for each edge
 399 to update the point positions.
- 400 For directed bundling, the high-level algorithm follows these steps (see pseudo-code 2):
 401 1. In the `Attract` function, when we check for the compatibility threshold, we also check if
 402 the edges are parallel or anti-parallel. We bundle edges only if they are parallel.
 403 2. After running the `Attract` function for the specified cycles and iterations, we apply
 404 a new step, `addLateralForces` which is really similar to `Attract` function, only this
 405 time we check if the two edges P and Q are anti-parallel and p_i and q_j are ‘close’ (see
 406 subsection 4.6) to each other.
 407 3. if yes, we apply the new repelling force and after accumulating all the forces call
 408 `applyForces`.
 409 4. Applying the `addLateralForces` introduces some visual disruption in bundled paths, a
 410 final `Attract` function is called, for i iterations, to smooth out and bundle again the
 411 edges which were affected with the lateral force, ensuring visual coherence.

412 **Algorithm 1** Attract function

```

413   1: for  $ie = 0$  to  $\text{length(edges)} - 1$  do
414     2:    $p \leftarrow \text{edges}[ie]$ 
415     3:   pointCount  $\leftarrow \text{length}(p.\text{points})$ 
416     4:   for  $i = 1$  to  $\text{pointCount} - 1$  do
417       5:      $p_i \leftarrow p.\text{points}[i]$  // Determining Edge Depth Factor for the specific point
418       6:     edgeDepthFactor  $\leftarrow \left(\frac{4i(p-i)}{p^2}\right)^{\text{poly\_deg}}$ 
419       7:     forceSum  $\leftarrow 0$ 
420       8:     force  $\leftarrow (0, 0, 0)$ 
421       9:     for  $ef = 0$  to  $\text{length(edges)} - 1$  do
422         10:        $c \leftarrow \text{compatibility}(ie, ef)$ 
423         11:       if  $c \leq c_{\text{thr}}$  or ( $\text{directed}$  and  $\text{directions}[ie][ef] < 0$ ) then
424           12:             continue
425           13:         end if
426           14:          $q \leftarrow \text{edges}[ef]$ 
427           15:          $q_j \leftarrow \begin{cases} q.\text{points}[i], & \text{if } \text{checkFlip}(q, p) \\ q.\text{points}[\text{length}(q.\text{points}) - 1 - i], & \text{otherwise} \end{cases}$ 
428           // Inner weight and Compatibility Weight
429           16:         weight  $\leftarrow \frac{\exp\left(-\frac{\|q_j - p_i\|^2}{2 \cdot \text{bell}^2}\right)}{q.\text{weight}} \cdot c^2$ 
430           17:         forceSum  $\leftarrow \text{forceSum} + \text{weight}$ 
431           18:         force  $\leftarrow \text{force} + \text{weight} \cdot q_j$ 
432           19:         end for
433           20:         if  $\text{forceSum} > 0$  then
434             21:           force  $\leftarrow \text{force}/\text{forceSum}$ 
435             // Edge depth Factor and Outer weight
436             22:           finalForce  $\leftarrow \text{edgeDepthFactor}^2 \cdot \frac{(\text{force} - p_i)}{p.\text{weight}}$ 
437             // Momentum
438             23:           finalForce  $\leftarrow \text{finalForce} + \beta \cdot p_i.\text{force}$ 
439             24:            $p_i.\text{force} \leftarrow \text{finalForce}$ 
440           25:         end if
441           26:         end for
442         27:       end for
443         28:     for all edge  $\in \text{edges}$  do
444       29:       applyForces(edge)
445     30:   end for

```

446 █ **Algorithm 2** addLateralForces

```

447 1: for  $ie = 0$  to  $\text{length}(\text{edges}) - 1$  do
448 2:    $p \leftarrow \text{edges}[ie]$ 
449 3:    $\text{pointCount} \leftarrow \text{length}(p.\text{points})$ 
450 4:   for  $i = 1$  to  $\text{pointCount} - 1$  do
451 5:      $p_i \leftarrow p.\text{points}[i]$ 
452 6:      $\text{edgeDepthFactor} \leftarrow (\frac{4i(p-i)}{p^2})^{\text{poly\_deg}}$ 
453 7:      $\text{forceSum} \leftarrow 0$ 
454 8:      $\text{force} \leftarrow (0, 0, 0)$ 
455 9:     for  $ef = 0$  to  $\text{length}(\text{edges}) - 1$  do
456 10:     $c \leftarrow \text{compatibility}(ie, ef)$ 
457 11:    if  $c \leq c_{\text{thr}}$  or  $\text{directions}[ie][ef] \geq 0$  then
458 12:      continue
459 13:    end if
460 14:     $q \leftarrow \text{edges}[ef]$ 
461 15:     $j \leftarrow \text{length}(q.\text{points}) - 1 - i$ 
462 16:     $q_j \leftarrow q.\text{points}[j]$ 
463 17:    if  $\|p_i - q_j\| > \frac{p.\text{length}() + q.\text{length}()}{2.50}$  then
464 18:       $\text{weight} \leftarrow 0$ 
465 19:       $\text{potential} \leftarrow q_j$ 
466 20:    else
467 21:       $q_{\text{prev}} \leftarrow q.\text{points}[j + 1]$ 
468 22:       $q_{\text{next}} \leftarrow q.\text{points}[j - 1]$ 
469 23:       $T_j \leftarrow \text{normalize}(q_{\text{next}} - q_{\text{prev}})$ 
470 24:       $N_j \leftarrow \text{normalize}(\text{crossProduct}(T_j, (0, 1, 0)))$ 
471 25:       $\text{potential} \leftarrow q_j + N_j \cdot \frac{p.\text{length}() + q.\text{length}()}{2.50}$ 
472 26:       $\text{weight} \leftarrow \frac{\exp\left(-\frac{\|\text{potential} - p_i\|^2}{2 \cdot \text{bell}^2}\right)}{q.\text{weight}} \cdot c^2$ 
473 27:    end if
474 28:     $\text{forceSum} \leftarrow \text{forceSum} + \text{weight}$ 
475 29:     $\text{force} \leftarrow \text{force} + \text{weight} \cdot \text{potential}$ 
476 30:  end for
477 31:  if  $\text{forceSum} > 0$  then
478 32:     $\text{force} \leftarrow \text{force}/\text{forceSum}$ 
479 33:     $\text{finalForce} \leftarrow \text{edgeDepthFactor}^2 \cdot \frac{(\text{force} - p_i)}{p.\text{weight}}$ 
480 34:     $\text{finalForce} \leftarrow \text{finalForce} + \beta \cdot p.\text{forces}[i]$ 
481 35:     $p_i.\text{force} \leftarrow \text{finalForce}$ 
482 36:  end if
483 37:  end for
484 38: end for
485 39: for all  $\text{edge} \in \text{edges}$  do
486 40:    $\text{applyForces}(\text{edge})$ 
487 41: end for
```

488 **Algorithm 3** checkFlip

```

489 1:  $ff \leftarrow q.\text{from\_node} - p.\text{from\_node}$ 
490 2:  $tt \leftarrow q.\text{to\_node} - p.\text{to\_node}$ 
491 3:  $ft \leftarrow q.\text{from\_node} - p.\text{to\_node}$ 
492 4:  $tf \leftarrow q.\text{to\_node} - p.\text{from\_node}$ 
493 5: if  $\|ff\| + \|tt\| < \|ft\| + \|tf\|$  then
494 6:     return true
495 7: else
496 8:     return false
497 9: end if

```

498 **Algorithm 4** applyForces

```

499 1: for all point  $\in edge.\text{points}$  do
500 2:     point  $\leftarrow$  point  $\cdot$  point.force
501 3: end for

```

502 **5 Metrics and Results**

503 For evaluation and results, we used qualitative and new quantitative metrics. We used
 504 multiple datasets: (Table 1) and performed visual comparisons. (For an easier explanation,
 505 the terms nodes and points are used interchangeably, referring to nodes and internal points.)
 506 The code base will be available in a public repository once the paper reaches a visualization
 507 conference for reproducibility purposes.

508 **5.1 Voxel-based edge density metrics**

509 Inspired by the work of Geoffrey Ellis and Alan Dix: ‘The plot, the clutter, the sampling and
 510 its lens: occlusion measures for automatic clutter reduction’ [2], we ported the 2D metrics
 511 into 3D metrics, extending the area-based metrics to voxel-based ones. These metrics consists
 512 of:

513 **5.1.1 Definitions**

514 1. **Overplotted percentage:** % of all used voxels which contains at least one edge/node,
 515 formula:

$$516 \quad 100 * \frac{|overplotted_voxels|}{|used_voxels|}.$$

517 2. **Overcrowded percentage:** % of edges which are bundled together, formula:

$$518 \quad 100 * \frac{|bundled_edges|}{|edges|}.$$

519 3. **Ink-paper ratio:** % of how crowded the whole visualization is, formula, quantifying
 520 spatial efficiency:

$$521 \quad \frac{|used_voxels|}{|available_voxels|}.$$

522 .

523 The terms used are defined as follows:

- 524 └─ *overplotted_voxels*: The number of voxels where there are more than one point/node
 525 inside the voxel,
 526 └─ *used_voxels*: The number of voxels where there is at least one point/node inside the
 527 voxel,
 528 └─ *bundled_edges*: The number of edges where one edge has at least one internal (non-
 529 terminal) point shared with another edge, indicating bundling,
 530 └─ *available_voxels*: All voxels within the defined 3D bounding box grid.

531 5.1.2 Methodology

532 To compute these metrics, we first determine the bounding box that encapsulates all nodes
 533 in the graph. This box (bounding volume) is discretized into regular 3D grid composed of
 534 x^3 smaller voxels, where x denotes the resolution along each axis. This resolution can be
 535 changed according to the desired resolution.

536 Each node is assigned a corresponding voxel using its 3D coordinates and the derived
 537 voxel size.

538 For edges, we used a 3D extension of Bresenham's line algorithm to identify all voxels
 539 intersected by the edge paths, including intermediate voxels between connected nodes.

540 The following pipeline is used to compute the metrics:

- 541 1. **Bounding Box Computation**: Compute the minimum and maximum coordinates
 542 across all nodes.
- 543 2. **Discretization / Voxelization**: Divide the bounding box uniformly into $x \times x \times x$
 544 voxels and determine the size accordingly.
- 545 3. **Node Mapping**: Assign each node to a voxel based on its spatial coordinates.
- 546 4. **Edge Voxelization**: Trace edge path through voxels using Bresenham's algorithm,
 547 marking all touched voxels.
- 548 5. **Metrics Calculation**: We evaluate the overplotted percentage, overcrowded percentage,
 549 and ink-paper ratio metrics based on all the marked and unmarked voxels.

550 This method provides a consistent voxel-based framework for analyzing spatial density and
 551 bundling quality in a 3D graph visualization. All results were conducted using our Python
 552 implementation (see: codebase, metrics folder), ensuring reproducibility and adaptability
 553 across datasets, edge bundling algorithms, and resolutions.

554 5.2 Results

555 To showcase the improvements, we are going to compare our results to the original MSEB
 556 algorithm's output. MSEB will be used as the baseline both in output's visual quality,
 557 bundle quality, and runtime. The enhancements of all added features have not changed the
 558 underlying force-directed bundling principles. The work of V.Singh [11] clearly states that
 559 for unweighted binary graphs, the results are nearly the same: "we aimed to establish that
 560 MSEB can be considered as a special case of EMSEB, where edge weights are binary, and in
 561 such a case both algorithms produce near identical results. This analysis serves to highlight
 562 that EMSEB can be a viable alternative to MSEB, capable of handling the same input cases
 563 without deviations in output or computational complexity". Given that MSEB has a time
 564 complexity of $\mathcal{O}(N \cdot M^2 \cdot K)$ — where N is the number of iterations, M is the number of
 565 edges, and K is the number of subdivision points per edge — our method, EMSEB, retains
 566 the same computational complexity. Runtime comparisons are presented in Table 2. For
 567 MSEB to replicate the EMSEB's edge weight bundling, it should duplicate edges which have
 568 edge weights as many times as the weight dictates, meaning that the runtime of MSEB would

XX:16 EMSEB

569 be clearly higher than EMSEB (having more edges in the graph). For simplification, MSEB
570 is running with unweighted graph mode and EMSEB is running with edge weights in mind.

571 5.2.1 Datasets

572 We tested the algorithms on multiple datasets from different domains. The datasets contain
573 social graphs that were converted into 3D topology to fit our purpose (Jazz Musicians Network,
574 Email EU Core, C. Elegans Neural Network), 2D geometrical (US Airlines and Migration,
575 EuroSiS Web Mapping), and neuron connectivity graphs (from the IEEE SciVis Contest
576 2023 - Dataset of Neuronal Network Simulations of the Human Brain). The brain_bundle_1,
577 brain_bundle_2 graphs are logical areas of the brain, sets of neurons on the surface of the
578 brain, brain_bundle_8 and brain_bundle_9 are sampled versions of the ‘viz-no-network’
579 simulation’s last iteration. Table 1 summarizes the datasets and their attributes.

580 Dataset	581 Vertices	582 Edges	583 3D	584 Directed	585 Weighted
586 US Airlines ⁵	587 235	588 2101	589 –	590 –	591 –
592 US Migration ⁶	593 6517	594 9780	595 –	596 –	597 –
598 EuroSiS Web Mapping ⁷	599 1285	600 7586	601 –	602 –	603 –
604 Jazz Musicians Network [4]	605 198	606 2742	607 c	608 –	609 –
610 Email EU core [14, 8]	611 1005	612 25571	613 c	614 –	615 –
616 C. Elegans Neural Network [13, 12]	617 306	618 2148	619 c	620 x	621 –
622 brain_bundle_1 [3]	623 17	624 240	625 x	626 x	627 x
628 brain_bundle_2 [3]	629 47	630 1790	631 x	632 x	633 x
634 brain_bundle_8 [3]	635 39	636 1128	637 x	638 x	639 x
640 brain_bundle_9 [3]	641 5000	642 7222	643 x	644 x	645 x

594 **Table 1** Overview of datasets used in experiments. Check marks (x) indicate supported properties;
595 dashes (–) denote unsupported or unavailable data; c means converted with force directed layout
596 method.

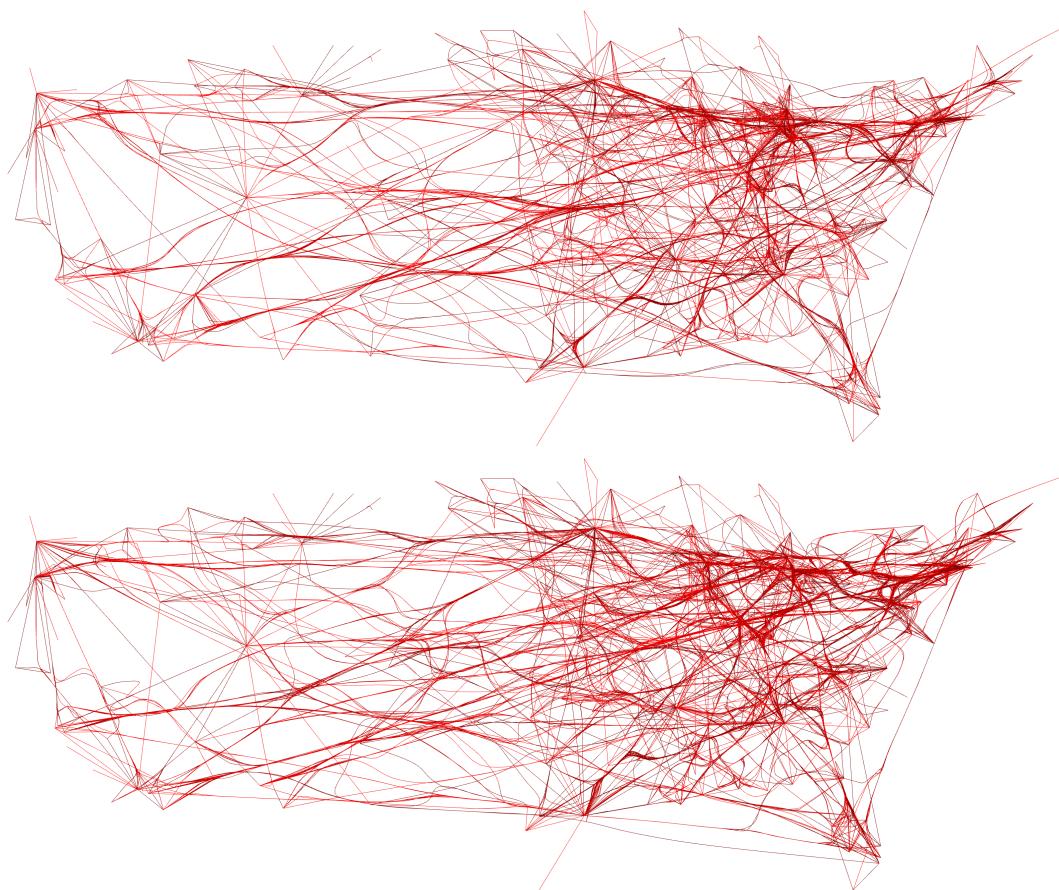
597 5.2.2 Comparison between MSEB and EMSEB

598 In this section, we compare side by side the outputs and metrics of these algorithms, seen in
599 table 2. The metrics were calculated with 50^3 voxel resolution. All datasets and results are
600 going to be available on the GitHub repository for reproducibility purposes. The results were
601 calculated on an Intel i7-7700HQ processor with 32 GB RAM, running Linux. The C++
602 code was built using Qt Creator 16.0.2, GCC 15.1.1. and compiled in Debug mode.

626 5.3 Visual confirmation

627 Although the table of comparisons (table 2) tells us how our algorithm performs compared
628 to MSEB, it does not tell the full picture. To verify the numbers, we are going to present
629 some quality improvements with side-by-side picture comparisons.

630 From the comparisons, especially from the brain_bundle_8 dataset (figure 7), we can see
631 that with all the added weights (‘inner’, ‘outer’, ‘edge depth’ and ‘compatibility’ weighting)
632 moving the subdivision points is taking longer times and does not bundle all the way together
633 for far apart but compatible edges. This can be an issue if we are not really interested in the
634 individual edge’s path.



■ **Figure 5** First: MSEB algorithm's output, Second: EMSEB algorithm's output on the US Airplanes dataset. ($nr_{cycles} = 10$, $iteration = 40$, $c_thr = 0.8$)



■ **Figure 6** First: MSEB algorithm's output, Second: EMSEB algorithm's output on the US Migration dataset. ($nr_{cycles} = 10$, $iteration = 40$, $c_thr = 0.8$)

Graph	Alg.	RT (ms)	Overplt.	Overcrwd.	Ink-Paper R
US Airlines	MSEB	33556	96.391	77.153	0.0115
	EMSEB	26269	96.838	54.402	0.0116
US Migration	MSEB	913540	97.060	73.384	0.0117
	EMSEB	687992	97.377	61.646	0.0118
EuroSiS	MSEB	683065	92.908	37.740	0.0111
	EMSEB	437925	93.976	30.885	0.0110
Jazz Musicians Network	MSEB	49685	69.787	98.869	0.0188
	EMSEB	38638	69.054	98.285	0.0185
Email EU core	MSEB	11381598	66.599	99.593	0.0276
	EMSEB	623872	45.850	99.096	0.0570
C. Elegans	MSEB	42490	74.325	100.0	0.0172
	EMSEB	27538	74.373	100.0	0.0169
brain_bundle_1	MSEB	736	82.829	95.416	0.0116
	EMSEB	590	78.975	90.416	0.0118
brain_bundle_2	MSEB	23784	93.257	98.8268	0.0335
	EMSEB	18344	88.735	57.821	0.0371
brain_bundle_8	MSEB	10576	79.070	97.429	0.0686
	EMSEB	8375	75.853	90.780	0.0707
brain_bundle_9	MSEB	969760	74.184	6.549	0.1058
	EMSEB	1046768	5.289	90.780	0.1040

Table 2 The calculations were carried out with $c_thr = 0.8$, $start_i=40$, $numcycles=10$, $bell=5$, $directed=0$ and voxel resolution of $x=50$

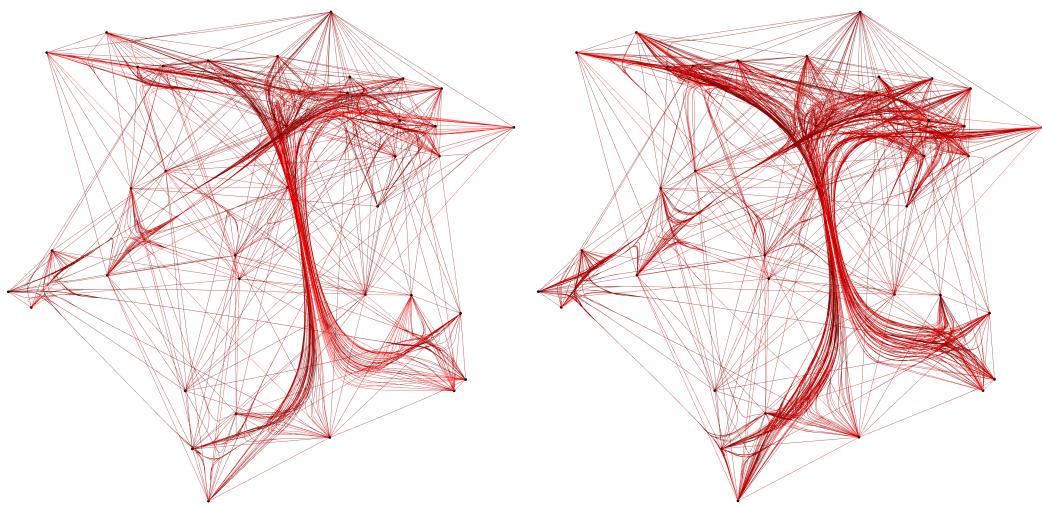
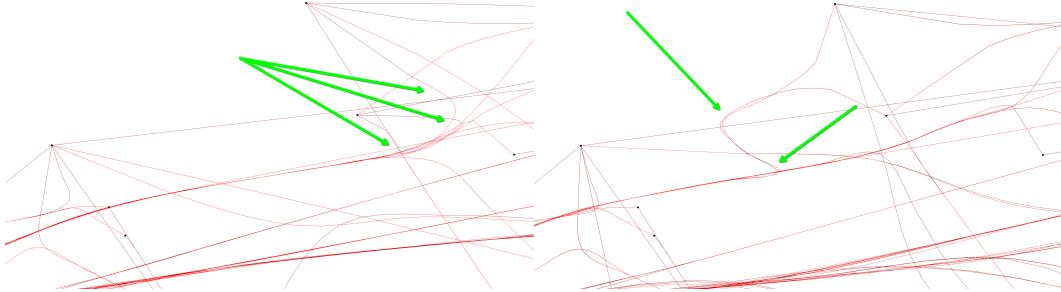


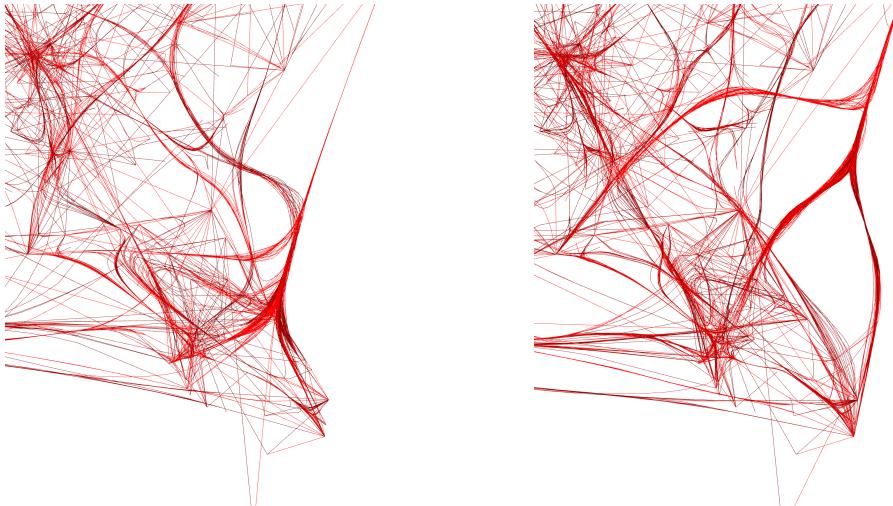
Figure 7 Left: MSEB algorithm's output, right EMSEB algorithm's output on the brain_bundle_8 dataset. ($nr_{cycles} = 10$, $iteration = 40$, $c_thr = 0.8$)

XX:20 EMSEB

635 Quality differences can be seen in the figures 8 and 9. We can see that the edge whiplashes
636 occur less time, making the edges easier to follow. This worsens the overplotting and ink-paper
637 ratio, but in a tolerable amount.



638 ■ **Figure 8** On the left MSEB making edge whiplash, over curved bends while on the right EMSEB
639 algorithm's output shows a less edge whiplash look on the US Airplanes dataset but with a noticeable
640 'S' bend.



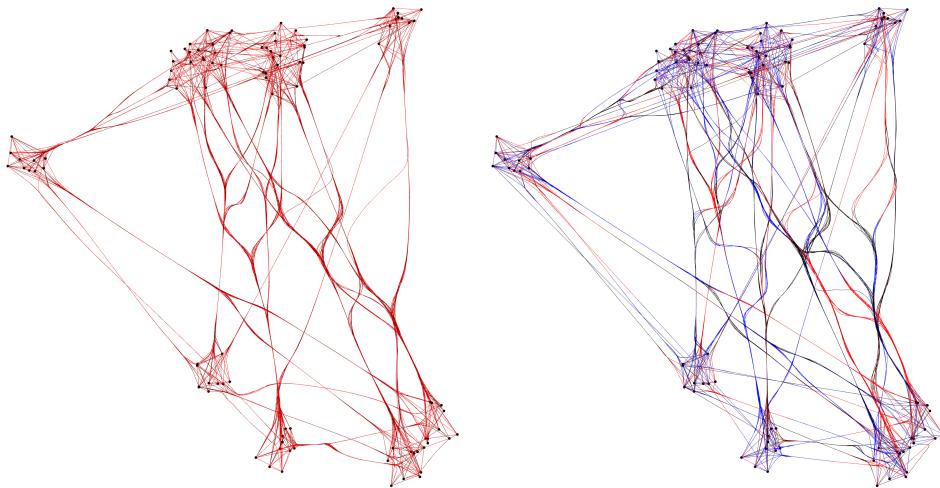
641 ■ **Figure 9** On the left MSEB making sharp, over curved bends while on the right EMSEB
642 algorithm's output shows a more natural look also on the US Migration dataset.

643 Also, to showcase the directed bundling of the algorithm, where the compatible edges are
644 bundled if they are parallel or anti-parallel and pushed apart into lanes, we used a random
645 weighted 3D graph (figure: 10, 11). We colored the edges in the following ways: at the start
646 node, the edge color is red, it switches gradient to black in the middle of the path and ends
647 in a blue color at the end node.

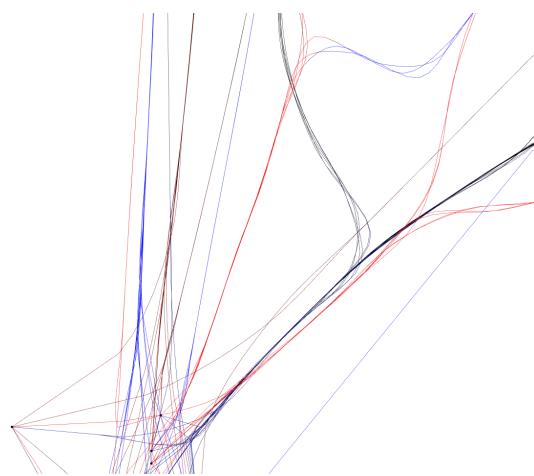
651 5.4 Improvement to Fibviewer

652 We improved this application to make the comparison of the algorithms easier in the following
653 ways:

- 654 ■ **View/Orientation saving/configuration:** To compare the same angle and position of
655 the graphs, we added a configurable text box field to input the matrix's values which
656 helps to align the two compared graphs together.



648 ■ **Figure 10** Undirected and directed bundling algorithms on a random graph, EMSEB output.
649 The edge starts out with red color and transitions over to blue at the endpoint.



650 ■ **Figure 11** Close-up view of separated lanes in directed bundling

- 657 ■ **Zoom level:** Added a zoom level text box field, to set the zoom level
 658 ■ **Catmull-Rom splines:** Added Catmull-Rom splines to connect the internal points on
 659 an edge's path. This improved the visualization, from having straight line segments to
 660 having smooth curves.
 661 ■ **Gradient coloring:** Added gradient coloring (from red to blue) to showcase the direc-
 662 tionality of the edges.

6 Discussion and Improvements

663 Our results show us that we improved the MSEB algorithm and made it quicker. With these
 664 changes we can see more natural lines, fewer whiplashes, more truthful edge paths when we
 665 use edge weights (thanks to the 'outer' weight and edge 'depth' concepts), but there are still
 666 some challenges to overcome to make this algorithm more usable. The following features are
 667 not implemented but can be done in a future work:

- 668 ■ **'S' bends:** As seen in the EMSEB's output in figures 8, we eliminated edge whiplash but
 669 introduced a kind of 'S' bends, where edges are trying to join a bundle in a very sharp
 670 curve. We suspect that this bend is caused by the incorrect depth factor of the edge,
 671 more likely an improper value `poly_deg`. In future work, we should try to eliminate these
 672 bends and make the bends look as natural as possible.
 673 ■ **Supporting more edge types:** The bundling algorithm only supports directed bundling,
 674 but one can easily use this directed edge bundling algorithm to separately bundle different
 675 kind of edges and adding different lateral forces based on the edge types.
 676 ■ **Showing self loops:** The Fibviewer does not show self loops. Support for self-loops
 677 needs to be added.
 678 ■ **Showing bundled edge weights by colors:** Currently Fibviewer is not showing the
 679 bundled path's total edge weights. It can only show directionality.
 680 ■ **More supported input/output formats:** While our solution works, it does not take
 681 standardized graph description file formats.
 682 ■ **Improved fibviewer GUI:** While the fibviewer is not the most powerful graph visual-
 683 izer, and be replaced with a better graph visualization software, it would still be usable as
 684 a lightweight and fast visualizer.

7 Conclusion

685 We presented our improved EMSEB algorithm, making it again a state-of-the-art solution
 686 within the family of force-directed edge bundling algorithms for 3D graphs. We improved
 687 MSEB's algorithm by incorporating new, even experimental, techniques such as weighted
 688 edge bundling, edge-depth weighting, compatibility weighting, edge traceability and directed
 689 bundles.

690 The incorporation of edge-depth and compatibility weighting improves the stability and
 691 coherence of edge paths, minimizing visual artifacts such as "edge whiplash." The result
 692 is a smoother, more natural-looking layout that better reflects the inherent structure of
 693 the underlying data. Importantly, EMSEB maintains the computational complexity of the
 694 original MSEB algorithm, ensuring its practicality for large-scale datasets. Despite the added
 695 enhancements, the performance overhead remains minimal. We think that incorporating the
 696 edge weights into the edge path's calculation leads to a more accurate representation of the
 697 bundles, and the underlying data itself. This helps us to show unknown associations between
 698 edge relationships in datasets where edge weights play a significant role.

To evaluate EMSEB's effectiveness, we applied it to several datasets. The results consistently demonstrate that EMSEB produces superior bundling outcomes, even in binary edge scenarios, which represent a special case of the algorithm. Looking ahead, we see strong potential for EMSEB in immersive environments such as VR and AR, where the spatial qualities of 3D bundling can be fully leveraged. This makes it particularly suitable for domains involving complex spatial data, such as neuroscience and connectivity research, where understanding edge relationships in three dimensions is crucial.

In summary, EMSEB advances the state of 3D edge bundling by combining technical rigor with practical flexibility, and we anticipate its adoption across a broad range of visual analytics applications.

References

- 1 Joachim Bottger, Alexander Schaefer, Gabriele Lohmann, Arno Villringer, and Daniel Margulies. Three-dimensional mean-shift edge bundling for the visualization of functional connectivity in the brain. *IEEE transactions on visualization and computer graphics*, 20, 08 2013. doi:10.1109/TVCG.2013.114.
- 2 Geoffrey Ellis and Alan Dix. The plot, the clutter, the sampling and its lens: occlusion measures for automatic clutter reduction. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '06, page 266–269, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1133265.1133318.
- 3 Tim Gerrits, Fabian Czappa, Divya Banesh, and Felix Wolf. Ieee scivis contest 2023 - dataset of neuronal network simulations of the human brain, October 2022. doi:10.5281/zenodo.10519411.
- 4 PABLO M. GLEISER and LEON DANON. Community structure in jazz. *Advances in Complex Systems*, 06(04):565–573, December 2003. URL: <http://dx.doi.org/10.1142/S0219525903001067>, doi:10.1142/s0219525903001067.
- 5 Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, 2006. doi:10.1109/TVCG.2006.147.
- 6 Danny Holten and Jarke J. Van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990, 2009. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2009.01450.x>, arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2009.01450.x>, doi:10.1111/j.1467-8659.2009.01450.x.
- 7 C. Hurter, O. Ersoy, and A. Telea. Graph bundling by kernel density estimation. *Computer Graphics Forum*, 31(3pt1):865–874, 2012. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2012.03079.x>, arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2012.03079.x>, doi:10.1111/j.1467-8659.2012.03079.x.
- 8 Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007. doi:10.1145/1217299.1217301.
- 9 OpenAI. Chatgpt (june 3 version), 2025. Large language model. URL: <https://chat.openai.com>.
- 10 David Selassie, Brandon Heller, and Jeffrey Heer. Divided edge bundling for directional network data. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2354–2363, 2011. doi:10.1109/TVCG.2011.190.
- 11 Vidur Singh. Extended mean shift edge bundling: A 3d weighted edge bundling algorithm. Master's thesis, Aarhus University, 2023.
- 12 Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998. doi:10.1038/30918.

- 749 13 J. G. White, E. Southgate, J. N. Thompson, and S. Brenner. The structure of the nervous
 750 system of the nematode *caenorhabditis elegans*. *Philosophical Transactions of the Royal*
 751 *Society of London. B, Biological Sciences*, 314:1–340, 1986. doi:10.1098/rstb.1986.0056.
- 752 14 Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. Local higher-order graph
 753 clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge*
 754 *Discovery and Data Mining (KDD)*, pages 555–564. ACM, 2017. doi:10.1145/3097983.
 755 3098050.
- 756 15 Daniel Zielasko, Xiaoqing Zhao, Ali Can Demiralp, Torsten W. Kuhlen, and Benjamin
 757 Weyers. Voxel-based edge bundling through direction-aware kernel smoothing. *Computers*
 758 & *Graphics*, 83:87–96, 2019. URL: <https://www.sciencedirect.com/science/article/pii/S0097849319301025>, doi:10.1016/j.cag.2019.06.008.

760 A ChatGPT Conversations

761 This appendix contains excerpts of conversations with ChatGPT[9] (OpenAI, 2025) that
 762 informed various parts of the thesis. All conversations were conducted between April and June
 763 2025 using ChatGPT (GPT-4.5-turbo, June 3 version), accessed via <https://chat.openai.com>.

764 A.1 Rephrase

765 **Date:** June 6, 2025

766 **Prompt:** ‘can you rephrase this? Three-Dimensional Mean-Shift Edge Bundling
 767 (MSEB) [1] builds on the foundation laid by Force Directed Edge Bundling (FEDB) [6] and
 768 Kernel Density Estimation Edge Bundling (KDEEB) [7] in order to combine 3D usability
 769 of FDEB and the numerical stability of KDEEB. We extend MSEB algorithm with more
 770 natural, intuitive edge paths, making our solution expressive, effective and efficient. MSEB[1]
 771 used binary graphs, where edges are represented as either present or absent. This limits the
 772 usage of more complex, richer datasets where we could actually use the unused attributes for
 773 our grain to accurately model the graph. The authors expressed that by using only binary
 774 data, it is actually benefits the representation because it has the advantage to select only
 775 relevant or important edges of the data. However, we argue that by not using all the data or
 776 reducing the edges we may lose significant insights. For this reason, we extended MSEB[1]
 777 to accommodate edge weights which are exerting to each other, making a more insightful
 778 solution. in latex form’

779 **Response:**

780 Certainly! Here’s your rephrased paragraph formatted in LaTeX:

781 Three-Dimensional Mean-Shift Edge Bundling (MSEB) [1] builds upon the foundations
 782 of Force-Directed Edge Bundling (FDEB) [6] and Kernel Density Estimation Edge Bundling
 783 (KDEEB) [7], combining the 3D applicability of FDEB with the numerical stability offered
 784 by KDEEB. In our work, we extend the MSEB algorithm by introducing more natural
 785 and intuitive edge trajectories, resulting in a solution that is expressive, effective, and
 786 computationally efficient. The original MSEB implementation focused on binary graphs,
 787 where edges are treated as either present or absent. While this simplifies the representation
 788 and allows for the selection of only the most relevant edges, it restricts the use of richer
 789 datasets that contain additional edge attributes. The authors noted that using binary data
 790 helps focus on the most important structural connections. However, we argue that discarding
 791 edge attributes or reducing the graph to binary form risks omitting valuable insights. To
 792 address this, we extend MSEB to support weighted edges, enabling the algorithm to account
 793 for the varying influence of connections. This allows for a more informative and nuanced

794 representation of the underlying data. Let me know if you'd like it integrated into a specific
795 section (e.g., 'Related Work' or 'Methodology') or need citation keys adjusted.

796 A.2 Rephrase

797 **Date:** June 6, 2025

798 **Prompt:** 'how would you say this a bit better? **Directional bundling (dir):** a way of
799 bundling edges which goes to nodes in the same directions, but does not bundle with edges
800 which are coming from those nodes'

801 **Response:**

802 Certainly! Here's a clearer and more polished version of your sentence:

803 **Directional bundling (dir):** a bundling strategy that groups edges based on their
804 directionality—specifically, edges that point toward the same target direction are bundled
805 together, while edges originating from those target nodes (i.e., in the opposite direction) are
806 excluded from the same bundle.

807 This version improves clarity and uses more precise terminology. Let me know if you'd
808 like an even more concise or technical variant.

809 A.3 Rephrase

810 **Date:** June 11, 2025

811 **Prompt:** 'okay rephrase this: We tested the polynomial solution, where we ran multiple
812 simulations with multiple *poly_deg* values (see figures 4). This sparked a general disagreement
813 about which edge depth function is the best and why. Although using a high *poly_deg* would
814 help to be more truthful to the original node-link graph representation, thus presenting the
815 data as is, it defeats the whole bundling idea. If we are looking for representative results
816 we would want to bend the edges as minimally as possible in order to achieve the bundling.
817 For properly find the best *polydeg* parameter, one should come up with new metrics which
818 are looking of edge path distance (straight lines would be the shortest paths), the bundling
819 quality or the bendy-ness of the curve, and/or if there is edge whiplash or not. This is beyond
820 our work, and needs more attention that firstly anticipated.'

821 **Response:**

822 Sure! Here's a clearer and more polished version of your paragraph, keeping the same intent
823 but improving the flow, tone, and phrasing:

824 We tested the polynomial edge depth function by running multiple simulations with
825 different values of *poly_deg* (see Figure 4). This led to some disagreement about which
826 function produces the most effective results and why. While higher values of *poly_deg*
827 stay closer to the original node-link representation—preserving the structure of the input
828 data—they can work against the main goal of edge bundling, which is to reduce visual clutter
829 by bending and grouping edges together.

830 To truly determine the optimal value for *poly_deg*, new evaluation metrics would be
831 needed. These could include measures of edge path distance (with straight lines as shortest
832 paths), bundling quality, curvature or "bendiness," and possibly the presence of edge whiplash.
833 Developing and validating such metrics is outside the scope of this work and would require
834 further study.

835 A.4 Rephrase

836 **Date:** June 9

XX:26 EMSEB

837 **Prompt:** rephrase this: In this section we are discussing what has been done before and
838 what was needed to be solved. While the EMSEB is based on MSEB [1], V.Singh's master's
839 thesis [11] presents lots of improvements over the original algorithm. The most obvious and
840 important improvement to MSEB [1], was the upgraded weighting algorithm. The original
841 algorithm only operated on binary graphs, and if there were weights, the same edge be added
842 as many times as the edge weight (positive integer) would be. This would slow down the
843 algorithm and even be unusable for graphs, where the edge weight would have a large range
844 of values. V.Singh[11] solution solved this issue by integrating the calculations with the edge
845 weight itself to determine the new position of the intermediate point.

846 **Response:**

847 In this section, we review previous work and identify the key challenges that motivated our
848 improvements. While EMSEB builds upon the foundation of MSEB [1], V. Singh's master's
849 thesis [11] introduced several important enhancements to the original algorithm.

850 One of the most significant advancements was the refinement of the weighting mechanism.
851 The original MSEB algorithm was designed for unweighted (binary) graphs. In cases where
852 edge weights were present, it would handle them by duplicating edges according to their
853 integer weights—adding the same edge multiple times. This approach not only introduced
854 redundancy but also made the algorithm inefficient or even impractical for graphs with large
855 or continuous edge weight ranges.

856 V. Singh [11] addressed this limitation by directly incorporating edge weights into the
857 computation of intermediate point positions. This improvement allowed for a more accurate
858 and scalable handling of weighted graphs, significantly enhancing the algorithm's efficiency
859 and applicability.

860 A.5 Code

861 **Date:** May 26

862 **Prompt:** Hi! I have a graph visualization program written in C++, QT. This is my
863 paintGL function where I put the spheres and lines from each point. void Edge::paintGL(bool
864 intermediateNodes, bool startAndEndNodes, float alpha, bool dualGradient) { if (startAndEnd-
865 Nodes) { glPointSize(10); glColor4f(0.0, 0.0, 0.0, alpha); glBegin(GL_POINTS); glVertex(fn);
866 glVertex(tn); glEnd(); }
867 glLineWidth(1); glPointSize(5);
868 int n = points.length(); if (n < 2) return;
869 for (int i = 0; i < n - 1; ++i) { QVector3D p1 = points.at(i); QVector3D p2 = points.at(i
870 + 1);
871 QVector3D nor = p1 - p2;
872 QVector3D col1, col2;
873 if (dualGradient) { float t1 = float(i) / float(n - 1); float t2 = float(i + 1) / float(n - 1);
874 // Red → Black → Blue gradient auto gradient = [] (float t) -> QVector3D { if (t < 0.5f)
875 { float f = t / 0.5f; // 0 to 1 return QVector3D(1.0f * (1 - f), 0.0f, 0.0f); // Red to black }
876 else { float f = (t - 0.5f) / 0.5f; // 0 to 1 return QVector3D(0.0f, 0.0f, 1.0f * f); // Black to
877 blue } };
878 col1 = gradient(t1); col2 = gradient(t2); } else { // Solid red col1 = col2 = QVector3D(1.0f,
879 0.0f, 0.0f); }
880 glBegin(GL_LINES); glNormal3f(nor.x(), nor.y(), nor.z());
881 glColor4f(col1.x(), col1.y(), col1.z(), alpha); glVertex(p1);
882 glColor4f(col2.x(), col2.y(), col2.z(), alpha); glVertex(p2); glEnd();

883 if (intermediateNodes && i != 0) { glColor4f(col1.x(), col1.y(), col1.z(), alpha); glBegin(GL_POINTS); glVertex(p1); glEnd(); } } }. I want to use Carmull-Rom splines instead of straight lines. Please implement the functions and make the necessary changes to the paintGL. Also make it into separate files the impelemtnation, so header and the cpp file itself.

888 **Response:**

889 To render smooth curves instead of straight lines, you can use Catmull-Rom splines, which
890 interpolate between control points and are ideal for paths and trajectories. Below is a
891 step-by-step solution to:

892 Add a Catmull-Rom spline interpolation function.

893 Update the paintGL method to use the interpolated spline.

894 Split the logic into a header and source file. Step 1: Create Header File — SplineUtils.h

```
895 #ifndef SPLINEUTILS_H #define SPLINEUTILS_H
```

```
896 #include <QVector3D> #include <QVector>
```

```
897 namespace SplineUtils {
```

```
898     // Computes Catmull-Rom spline interpolated points QVector<QVector3D> interpolateCatmullRom(const QVector<QVector3D>& controlPoints, int samplesPerSegment);
```

```
899     } // namespace SplineUtils
```

```
900 #endif // SPLINEUTILS_H
```

902 Step 2: Create Source File — SplineUtils.cpp

```
903 #include "SplineUtils.h"
```

```
904 namespace SplineUtils {
```

```
905     // Interpolates between 4 points using Catmull-Rom spline static QVector3D catmull-  
906 Rom(const QVector3D& p0, const QVector3D& p1, const QVector3D& p2, const QVector3D&  
907 p3, float t) { float t2 = t * t; float t3 = t2 * t;
```

```
908         return 0.5f * ((2.0f * p1) + (-p0 + p2) * t + (2.0f * p0 - 5.0f * p1 + 4.0f * p2 - p3) * t2  
909 + (-p0 + 3.0f * p1 - 3.0f * p2 + p3) * t3); }
```

```
910         QVector<QVector3D> interpolateCatmullRom(const QVector<QVector3D>& control-  
911 Points, int samplesPerSegment) { QVector<QVector3D> result;
```

```
912         int n = controlPoints.size(); if (n < 4) return controlPoints; // Not enough points
```

```
913         for (int i = 1; i < n - 2; ++i) { for (int j = 0; j < samplesPerSegment; ++j) { float  
914 t = float(j) / samplesPerSegment; QVector3D point = catmullRom(controlPoints[i - 1],  
915 controlPoints[i], controlPoints[i + 1], controlPoints[i + 2], t); result.append(point); } }
```

```
916         // Add final point result.append(controlPoints[n - 2]);
```

```
917         return result; }
```

```
918 } // namespace SplineUtils
```

919 Step 3: Update Your Edge::paintGL to Use Splines

920 You'll need to include the new spline header and call the interpolation before rendering.

921 Modified Edge::paintGL:

```
922 #include "SplineUtils.h"
```

```
923 void Edge::paintGL(bool intermediateNodes, bool startAndEndNodes, float alpha, bool  
924 dualGradient) { if (startAndEndNodes) { glPointSize(10); glColor4f(0.0, 0.0, 0.0, alpha);  
925 glBegin(GL_POINTS); glVertex(fn); glVertex(tn); glEnd(); }
```

```
926     glLineWidth(1); glPointSize(5);
```

```
927     int n = points.length(); if (n < 4) return;
```

```
928         // Use spline interpolation QVector<QVector3D> interpolatedPoints = SplineUtils::interpolateCatmullRom(points,  
929 10); // 10 samples per segment
```

XX:28 EMSEB

```
930     int ipCount = interpolatedPoints.length(); for (int i = 0; i < ipCount - 1; ++i) {  
931         QVector3D p1 = interpolatedPoints[i]; QVector3D p2 = interpolatedPoints[i + 1];  
932         QVector3D nor = p1 - p2; QVector3D col1, col2;  
933         if (dualGradient) { float t1 = float(i) / float(ipCount - 1); float t2 = float(i + 1) /  
934             float(ipCount - 1);  
935             auto gradient = [] (float t) -> QVector3D { if (t < 0.5f) { float f = t / 0.5f; return  
936                 QVector3D(1.0f * (1 - f), 0.0f, 0.0f); } else { float f = (t - 0.5f) / 0.5f; return QVector3D(0.0f,  
937                 0.0f, 1.0f * f); } };  
938             col1 = gradient(t1); col2 = gradient(t2); } else { col1 = col2 = QVector3D(1.0f, 0.0f,  
939                 0.0f); }  
940             glBegin(GL_LINES); glNormal3f(nor.x(), nor.y(), nor.z());  
941             glColor4f(col1.x(), col1.y(), col1.z(), alpha); glVertex(p1);  
942             glColor4f(col2.x(), col2.y(), col2.z(), alpha); glVertex(p2); glEnd();  
943             if (intermediateNodes && i != 0) { glColor4f(col1.x(), col1.y(), col1.z(), alpha); gl-  
944                 Begin(GL_POINTS); glVertex(p1); glEnd(); } } }
```

945 A.6 Code

946 **Date:** May 15

947 **Prompt:** I have a 3D graph stored in a VTK ASCII file with nodes and edges. I want
948 to analyze edge density using a voxel-based approach. Can you write a Python script that:
949 Reads nodes and edges from a VTK ASCII file. Divides the 3D space into a voxel grid.
950 Maps edges to voxels using 3D Bresenham's algorithm. Computes three metrics: Percentage
951 of voxels that are overplotted (contain more than one edge), Percentage of edges that are
952 bundled (share interior nodes after quantization), Ink-to-paper ratio (used voxels / total
953 voxels). Optionally visualize the voxel grid, nodes, and edges in 3D using matplotlib. Include
954 functions for each step and make the code modular and reusable.

955 **Response:** It generated the metrics/voxel_based_edge_density.py file