

MidcurveLLM: Geometric Dimension Reduction using Large language models

Abstract

Various applications necessitate lower-dimensional representations of shapes, with the midcurve serving as a one-dimensional (1D) depiction of a two-dimensional (2D) planar shape. Widely applied in animation, shape matching, retrieval, finite element analysis, and more, midcurves are integral for compactly conveying geometric information. Diverse methods for midcurve computation exist, contingent upon input shape types (e.g., images, sketches) and processing techniques (thinning, Medial Axis Transform (MAT), Chordal Axis Transform (CAT), Straight Skeletons, etc.).

This paper rigorously explores the integration of Large Language Models (LLMs) for computing midcurves in 2D geometric profiles, distilling complex shapes while preserving vital geometric details. Traditional methods, such as Medial Axis Transform and Chordal Axis Transform, face challenges with intricate shapes and diverse connections, prompting an in-depth evaluation of LLM efficacy.

The research dissects the problem, framing it as a Graph Summarization challenge akin to text summarization. Challenges in geometric representation, variable-length inputs, and branched midcurves are meticulously addressed. Neural network-based approaches, including Sequence-to-Sequence models and Image-to-Image transformations, are scrutinized. Despite limitations, Text-to-Text transformation using LLMs, illustrated through prompt-based evaluations, emerges as a promising avenue.

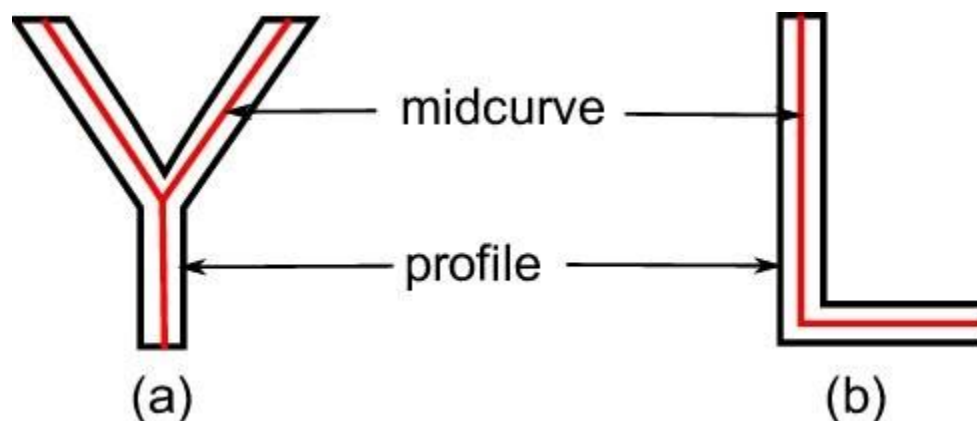
Fine-tuning, employing a Boundary Representation (Brep) structure, showcases potential in overcoming sequential point list limitations. Challenges in LLM fine-tuning, from model quality to dataset size, are outlined, calling for continued refinement. The conclusion emphasizes the dynamic nature of midcurve generation, signaling a need for ongoing exploration in tandem with LLM evolution. This research opens avenues for scholars and practitioners to delve deeper into geometric dimension reduction using LLMs.

Introduction

In the domain of Computer-Aided Design (CAD) for thin-walled solids, such as sheet metal and plastic components, a common practice is the dimensional reduction of models to their corresponding Mid-surfaces. This strategy expedites the Computer-Aided Engineering (CAE) analysis process. The Mid-surface is defined as a surface situated midway within the input shape, serving as an effective surrogate for analysis purposes. Despite its significance, the computation of Mid-surfaces remains a laborious and predominantly manual task, primarily due to the absence of robust automated methodologies.

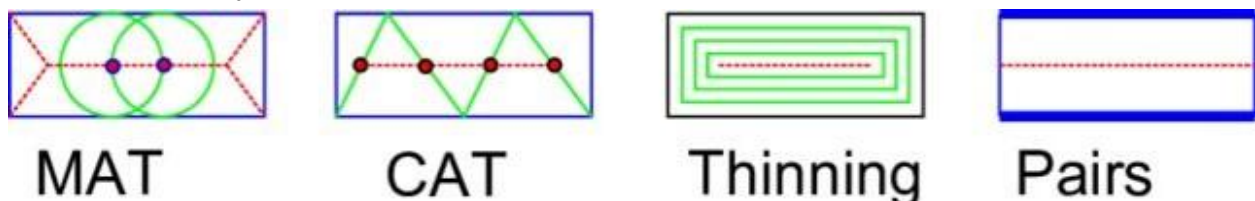
Existing automatic approaches for Midsurface generation often encounter pitfalls, resulting in issues such as gaps, missing patches, and overlapping surfaces. The manual correction of these errors demands significant time investment, extending to hours or even days. Therefore, the development of a dependable and efficient automated process for generating robust Midsurfaces emerges as a critical need in the field. As the Midsurface is with regards to 3D solid shapes, the corresponding equivalent in 2D planar shapes is called Midcurve.

The midcurve of a 2D geometric profile is a curve equidistant from the bounding curves, providing a simplified representation of the profile while retaining essential geometric information. Despite various traditional methods, the challenge persists due to shape complexity and connection variations. The MidcurveNN research project assesses the feasibility of employing Neural Networks, specifically LLMs, for midcurve generation.



(Example of Midcurves of 2D Geometric profiles [\[ref\]](#))

Although many approaches like Medial Axis Transform, Chordal Axis Transform, Thinning, Pairing, etc., have been tried for decades, the problem remains unsolved due to complexity of shapes and variety of connections.

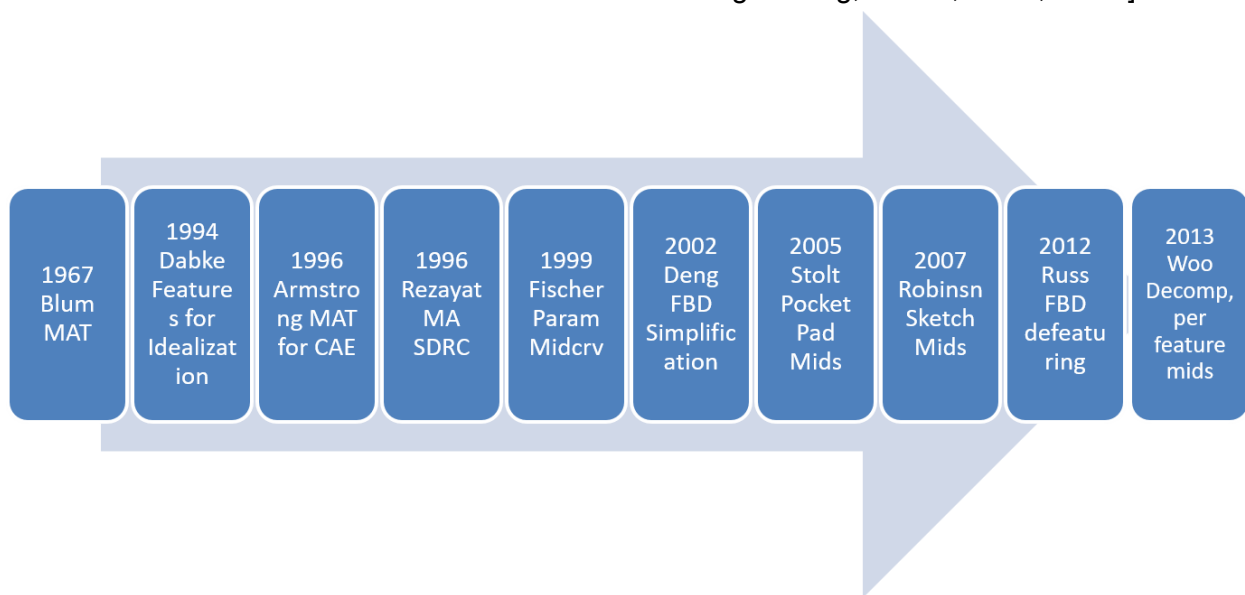


(Midcurve Approaches [\[ref\]](#))

Research project MidcurveNN attempts to evaluate if Neural Networks can be used to generate the midcurve of a 2D geometric profile.

Related Work

The exploration of computing Midsurface and Midcurve has spanned several decades, with noteworthy milestones depicted in Figure. A comprehensive analysis of this field is available in the survey paper [Kulkarni, Y.; Deshpande, S.: Medial object extraction - a state of the art. In International Conference on Advances in Mechanical Engineering, SVNIT, Surat, 2010.].

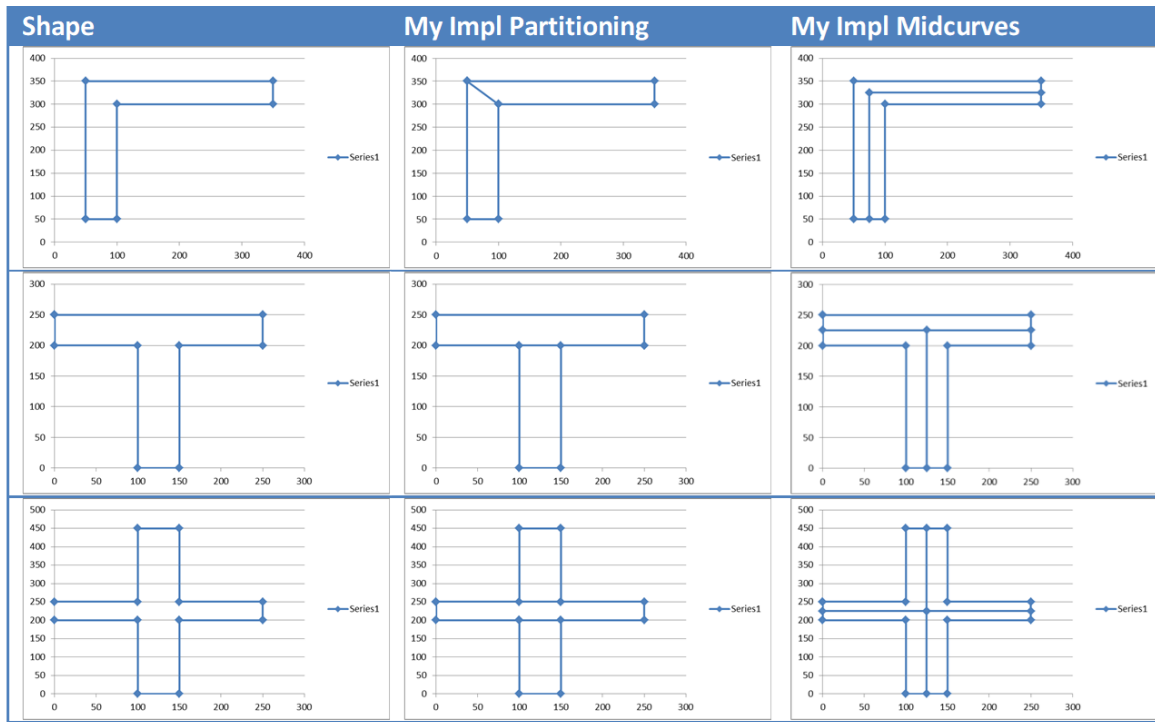


The following key observations emerge from classical approaches:

- Formal methods like MAT exhibit the strength of applicability to shapes of varying thickness, providing a reversible process for computing the original shape.
- MAT and Thinning methods, however, suffer from drawbacks such as the creation of unnecessary branches, a reduction in shape size, and a vulnerability to changes in base geometry.
- MAT-based approaches encounter robustness issues, necessitating recomputation with altered base geometries, leading to potentially divergent results.
- Despite the maturity of MAT approaches, their utilization in Midcurve generation remains complex, particularly regarding ensuring appropriate topology.
- CAT approaches face the limitation of requiring pre-generated meshes, which proves challenging for complex 2D profiles.
- Thinning approaches, relying on straight line skeleton split events, yield counterintuitive outcomes with sharp reflex vertices.

- Pairs approach encounters challenges in maintaining continuity when the input curves or surfaces are not in one-to-one form.
- The quality of Midcurve generated by the Pairs approach is contingent on the sampled input shape points.
- Midcurve by profile decomposition, while not widely adopted, faces issues of generating redundant sub-shapes, rendering it ineffective and unstable for further processing.

The survey paper discourages the use of formal methods, such as MAT, CAD, Thinning, and Parametric, for Midcurve computation due to their reliance on heavy post-processing to eliminate unwanted curves. Heuristic methods, particularly decomposition, have been error-prone and inefficient. The author's prior work (Figure below) demonstrates success on simpler shapes but has scalability limitations due to its rule-based nature.



Recent advances in the field have seen the integration of Neural Network-based techniques for Skeleton Midcurve computation, primarily through image processing. Rodas employed Deep Convolutional Neural Networks to simultaneously compute object boundaries and skeletons. Shen et al. developed a Convolutional neural network-based skeleton extractor capturing both local and non-local image context. Zhao introduced a hierarchical feature learning mechanism. Wang et al. focused on content flux in skeletons as a learning objective. Panichev et al. approached the task using U-net to extract skeletons, although not for Midcurve generation.

A review of the state-of-the-art emphasizes the need for Midcurve computation approaches that consider application context, accuracy, and the characteristics and aspect ratio of sub-polygons. The current research demands Midcurve generation for primitive-shaped, thin sub-polygons, as non-primitive, skewed shapes could yield inappropriate Midcurves.

The current paper proposes a novel method of computing Midcurve using Neural Networks i.e. Deep Learning approach with Large Language Models.

Problem Statement

- Goal: Given a 2D closed shape (closed polygon) find its midcurve (polyline, closed or open)
- Input: set of points or set of connected lines, non-intersecting, simple, convex, closed polygon
- Output: another set of points or set of connected lines, open/branched polygons possible

Essentially, if we consider vertices as nodes and lines as arcs then the polygon/polyline profile is nothing but a Graph and thus the midcurve generation is a Graph

Summarization/Dimension-Reduction/Compression issue, i.e. reducing a large graph to a smaller graph preserving its underlying structure, similar to text summarization, which attempts to keep the essence.

Geometry Representation

Summarization/Dimension-Reduction/Compression can be viewed as an encoder-decoder problem where larger input is fed to the encoder side whereas reduced output is generated by the decoder side.

To be able to use Neural Network based encoder decoders, both input and output need to be in a fixed size vector form. So, can variable shape polygons/polylines be fed to the neural networks? Issues are:

- Shapes can not be modeled as sequences. Although polygon shape L may appear as a sequence of points, it is not.
- All shapes can not be drawn without lifting a pencil, which is possible for sequences. Say, Shapes like Y or concentric O, cannot be modeled as sequences. So, Midcurve transformation cannot be modeled as Sequence 2 Sequence network.
- How to represent a geometric figure to feed to any Machine/Deep Learning problem, as they need numeric data in vector form?
- How to convert geometric shapes (even after restricting the domain to 2D linear profile shapes) to vectors?
- Closest data structure is the graph, but that's predominantly topological and not geometrical. Meaning, graphs represent only connectivity and not spatial positions. Here,

nodes would have coordinates and arcs would have curved-linear shape. So, even Graph Neural Networks, which convolute neighbors around a node and pool the output to generate vectors, are not suitable.

- Need RnD to come up with a way to generate geometric-graph embedding that will convolute at nodes (having coordinates) around arcs (having geometry, say, a set of point coordinates), then pool them to form a meaningful representation. Real crux would be how to formulate pooling to aggregate all incident curves info plus node coordinates info into a single number!!

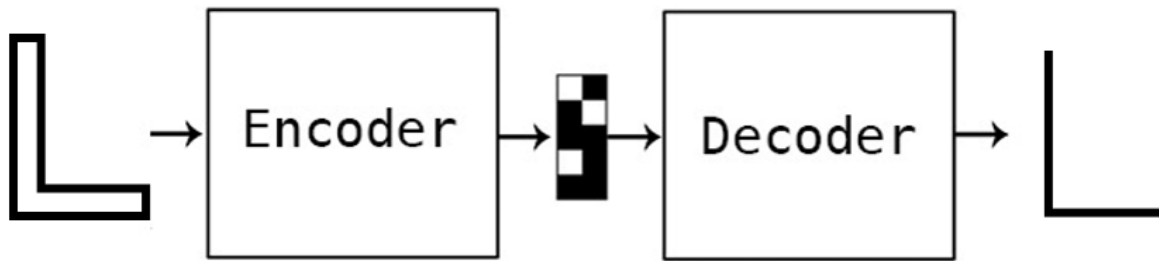
Variable Lengths Issue

- Midcurve Generation is a dimension-reduction problem. In 2D, input is the sketch profile (parametrically 2D), whereas the output is the midcurve (parametrically 1D). Input points are ordered (mostly forming closed loop, manifold). Output points may not be ordered, and can have branches (non-manifold)
- It is a variable input and variable output problem as the number of points and lines are different in input and output.
- It is a network 2 network problem (not Sequence to Sequence) with variable size inputs and outputs.
- For an Encoder-Decoder like network, libraries like Tensorflow need fixed length inputs. If input has variable lengths then padding is done with some unused value. But the padding will be a big issue here as the padding value cannot be like 0,0 as it itself would be a valid point.
- The problem of using seq2seq is that both input polygons and output branched midcurves are not linearly connected, they may have loops, or branches. Need to think more.
- Instead of going to a point list as i/o let's look at a well worked format of images. Images are of constant size, say 64x64 pixels. Let's color profile in the bitmap (b&w only) similarly midcurve in output bitmap. With this as an i/o LSTM encoder decoder seq2seq can be applied. Variety in training data can be populated by shifting/rotating/scaling both i/o. Only 2D sketch profile for now. Only linear segments. Only a single simple polygon with no holes.
- How to vectorise? Each point has 2 floats/ints. So the total input vector polygon of m points is 2m floats/ints. Closed polygon with repeat the first point as last. Output is a vector of 2n points. In case of a closed figure, repeat the last point. Prepare training data using data files used in MIDAS. To make 100s, 1000s of input profiles, one can scale both input and output with different factors, and then randomly shuffle the entries. Find max num points of a profile, make that as fixed length for both input and output. Fill with 0,0??? As origin 0,0 could be valid part of profile...any other filler? NULL? Run simple feed forward NN, later RNN, LSTM, Seq2seq

Dilution to Images

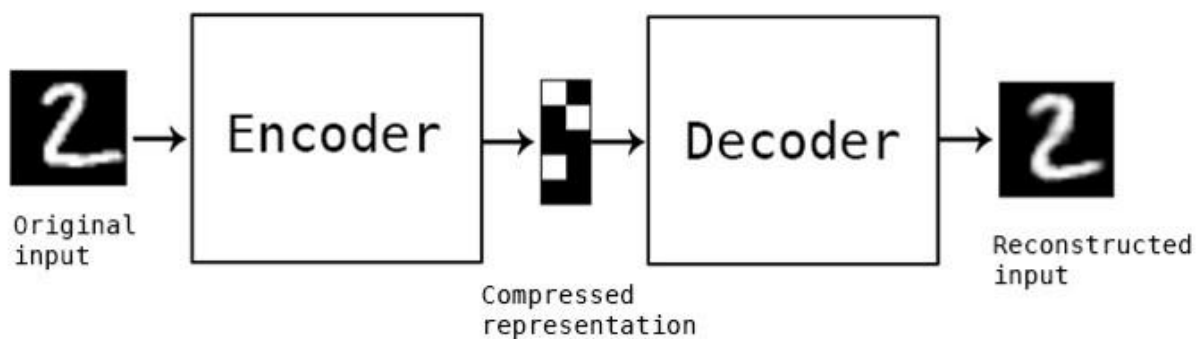
- Images of geometric shapes address both, representation as well as variable-size issues. Big dilution is that true geometric shapes are like Vector images, whereas images used here would be of Raster type. Approximation has crept in.
- Even after modeling, the predicted output needs to be post-processed to bring it to geometric form. Challenging again.
- Thus, this project is divided into three phases:
 - Phase I: Image to Image transformation learning
 - Img2Img: i/o fixed size 100x100 bitmaps
 - Populate many by scaling/rotating/translating both io shapes within the fixed size
 - Use Encoder Decoder like Semantic Segmentation or Pix2Pix of Images to learn dimension reduction
 - Phase II: Text to Text transformation learning
 - Look for a good representation to store geometry/graph/network as text so that NLP (Natural Language Techniques) can be applied. [Paper: "Talk like a graph: encoding graphs for large language models"](<https://arxiv.org/pdf/2310.04560.pdf>) surveys many such representations and benchmarks them, but none of them look appropriate for geometry.
 - So, here we leverage a geometry representation similar to that found in 3D B-rep (Boundary representation), but in 2D.
 - Phase III: Geometry to Geometry transformation learning
 - Build both, input and output polyline graphs with (x,y) coordinates as node features and edges with node id pairs mentioned. For poly-lines, edges being lines, no need to store geometric intermediate points as features, else for curves, store say, sampled fixed 'n' points.
 - Build Image-Segmentation like Encoder-Decoder network, given Graph Convolution Layers from DGL in place of the usual Image-based 2D convolution layer, in the usual pytorch encoder-decoder model.
 - Generate a variety of input-output polyline pairs, by using geometric transformations (and not image transformations as done in Phase I).
 - See if Variational Graph Auto-Encoders <https://github.com/dmlc/dgl/tree/master/examples/pytorch/vgae> can help.

Image-to-Image based Approach



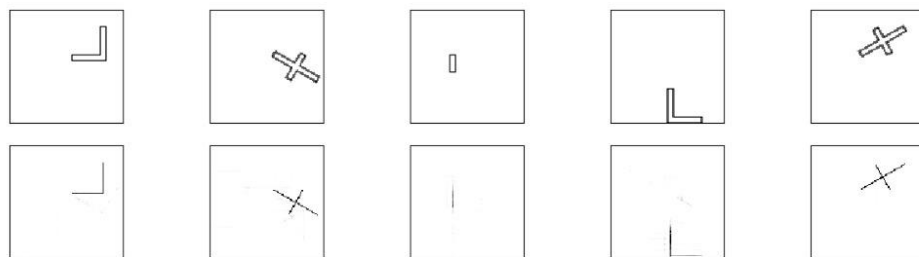
(Midcurve by Encoder Decoder [\[ref\]](#))

Such Transformers taking variable length input and output graphs have not been established. So, would taking this problem to image domain help?



(Auto Encoder on images [\[ref\]](#))

Various Image2Image networks have been tried by the MidcurveNN project but with limited success. Here is a sample:



(Simple Encoder Decoder network in Tensorflow/Keras [\[ref\]](#))

Text-to-Text based Approach

Prompt-based Approach

Evaluating LLMs (Large Language Models), like can be employed to generate the midcurve. Idea is to give a prompt telling what needs to be done along with some examples, and see if LLMs can generate shape for the test example. Geometry has been serialized into text as a simple list of points.

A few shots prompt developed:

You are a geometric transformation program that transforms input 2D polygonal profile to output 1D polyline profile. Input 2D polygonal profile is defined by set of connected lines with the format as:
input : [line_1, line_2, line_3,...] where lines are defined by two points, where each point is defined by x and y coordinates. So
line_1 is defined as ((x_1, y_1), (x_2,y_2)) and similarly the other lines.
Output is also defined similar to the input as a set of connected lines where lines are defined by two points, where each point is defined by x and y coordinates. So,
output : [line_1, line_2, line_3,...]

Below are some example transformations, specified as pairs of 'input' and the corresponding 'output'. After learning from these examples, predict the 'output' of the last 'input' specified.
Do not write code or explain the logic but just give the list of lines with point coordinates as specified for the 'output' format.

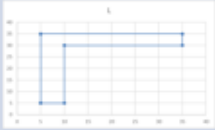
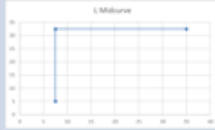
input:[((5.0,5.0), (10.0,5.0)), ((10.0,5.0), (10.0,30.0)), ((10.0,30.0), (35.0,30.0)), ((35.0,30.0), (35.0, 35.0)), ((35.0, 35.0), (5.0,35.0)), ((5.0,35.0), (5.0,5.0))]
output: [((7.5,5.0), (7.5, 32.5)), ((7.5, 32.5), (35.0, 32.5)), ((35.0, 32.5) (7.5, 32.5))]

input: [((5,5), (10, 5)), ((10, 5), (10, 20)), ((10, 20), (5, 20)), ((5, 20),(5,5))]
output: [((7.5, 5), (7.5, 20))]



input: [((0,25.0), (10.0,25.0)), ((10.0,25.0),(10.0, 45.0)), ((10.0, 45.0),(15.0,45.0)), ((15.0,45.0), (15.0,25.0)), ((15.0,25.0),(25.0,25.0)), ((25.0,25.0),(25.0,20.0)), ((25.0,20.0),(15.0,20.0)), ((15.0,20.0),(15.0,0)), ((15.0,0),(10.0,0)), ((10.0,0),(10.0,20.0)), ((10.0,20.0),(0,20.0)), ((0,20.0),(0,25.0))]
output: [((12.5,0), (12.5, 22.5)), ((12.5, 22.5),(12.5,45.0)), ((12.5, 22.5), (0,22.5)), ((12.5, 22.5), (25.0,22.5))]

input:[((0, 25.0), (25.0,25.0)),((25.0,25.0),(25.0,20.0)), ((25.0,20.0),(15.0, 20.0)), ((15.0, 20.0),(15.0,0)), ((15.0,0),(10.0,0)), ((10.0,0),(10.0,20.0)), ((10.0,20.0),(0,20.0)), ((0,20.0),(0, 25.0))]
output:

The first input example above represents an 'L' shape (shown below) and the second is an 'I', whereas the 3rd is a 'Plus' sign shape.

Profile Data		Profile Picture	Midcurve Data		Midcurve Picture
5.0	5.0		7.5	5.0	
10.0	5.0		7.5	32.5	
10.0	30.0		35.0	32.5	
35.0	30.0		7.5	32.5	
35.0	35.0				
5.0	35.0				

The last shape for which LLM has been asked for the answer is actually a 'T' shape. The picture below shows the correct/actual answer as well.

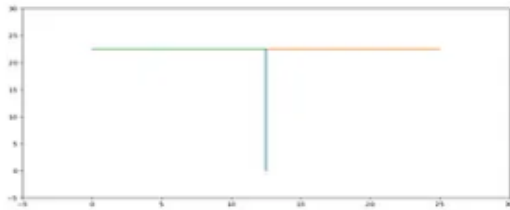
Profile Data		Profile Picture	Midcurve Data		Midcurve Picture
0	25.0		12.5	0	
25.0	25.0		12.5	22.5	
25.0	20.0		25.0	22.5	
15.0	20.0		0	22.5	
15.0	0				
10.0	0				
10.0	20.0				
0	20.0				

And the outputs computed by various LLMs (ChatGPT, Perplexity AI, Bard) , along with the real/actual answer:

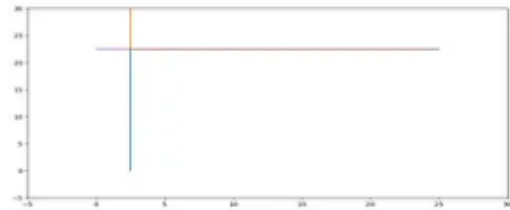
Actual: [((12.5,0), (12.5,22.5)), ((12.5,22.5),(25.0,22.5)), ((12.5,22.5),(0,22.5))]
ChatGPT: [((2.5, 0), (2.5, 22.5)), ((2.5, 22.5), (2.5, 45.0)), ((2.5, 22.5), (25.0, 22.5)), ((2.5, 22.5), (12.5, 22.5)), ((2.5, 22.5), (0, 22.5)), ((2.5, 22.5), (25.0, 22.5))]
Perplexity: [((12.5,0), (12.5, 22.5)), ((12.5, 22.5),(12.5,45.0)), ((12.5, 22.5), (0,22.5)), ((12.5, 22.5), (25.0,22.5))]
Bard: [((12.5, 0), (12.5, 25.0)), ((12.5, 25.0), (25.0, 25.0)), ((25.0, 25.0), (25.0, 0))]

Visually here is how results from different LLMs look:

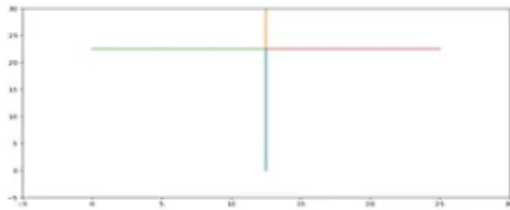
Actual/Expected



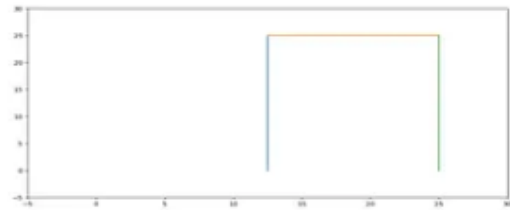
ChatGPT



Perplexity AI



Bard



All of the above have failed. Even latest (Oct 2023), the results are:

- llama 7B g4_0 ggml: (8, 17) & (64, 32): Wrong.
- Bard: [((8.33,5),(8.33, 22.5)), ((8.33, 22.5), (25,22.5)), ((8.33, 22.5), (0,25))]: Wrong.
- Hugging Chat: [((12.5, 0), (12.5, 22.5)), ((12.5, 22.5), (25.0, 22.5)), ((25.0, 22.5), (25.0, 25.0))]: a bit wrong on the last line.
- GPT-4: [((12.5,0), (12.5,22.5)), ((12.5,22.5),(0,22.5)), ((12.5,22.5),(25.0,22.5))] just change in sequence of lines, and that's inconsequential, so the answer is correct.
- Claude: [((12.5, 0.0), (12.5, 22.5)), ((12.5, 22.5), (12.5, 25.0)), ((12.5, 22.5), (0.0, 22.5)), ((12.5, 22.5), (25.0, 22.5))]

Although other proprietary and open-source models need to catch-up with GPT-4, even GPT-4 needs to be developed further to understand not just sequential lines but graphs/networks with different shapes, essentially, the geometry.

Fine-tuning based approach

One limitation of 2D geometry-shape, represented as a sequential list of points, is that we can not represent line intersections or concentric loops. To address this a more comprehensive structure has been proposed which is based on the corresponding 3D structure popular in Solid Modeling, called Brep (Boundary Representation).

```

{
  'ShapeName': 'I',
  'Profile': [(5.0, 5.0), (10.0, 5.0), (10.0, 20.0), (5.0, 20.0)],
  'Midcurve': [(7.5, 5.0), (7.5, 20.0)],
  'Profile_brep': {
    'Points': [(5.0, 5.0), (10.0, 5.0), (10.0, 20.0), (5.0, 20.0)], # list of (x,y) coordinates
    'Lines': [[0, 1], [1, 2], [2, 3], [3, 0]], # list of point ids (ie index in the Points list)
    'Segments': [[0, 1, 2, 3]] # list of line ids (ie index in Lines list)
  },
  'Midcurve_brep': {
    'Points': [(7.5, 5.0), (7.5, 20.0)],
    'Lines': [[0, 1]],
    'Segments': [[0]]
  }
}

```

ShapeName	Profile	Midcurve	Profile_brep	Midcurve_brep
I	[(5.0, 5.0), [10.0, 5.0], [10.0, 20.0], [5.0, 20.0]]	[[7.5, 5.0], [7.5, 20.0]]	("Points": [(5.0, 5.0), [10.0, 5.0], [10.0, 20.0], [5.0, 20.0]], "Lines": [[0, 1], [1, 2], [2, 3], [3, 0]], "Segments": [[0, 1, 2, 3]])	("Points": [(7.5, 5.0), [7.5, 20.0]], "Lines": [[0, 1]], "Segments": [[0]])
L	[(5.0, 5.0), [10.0, 5.0], [10.0, 30.0], [35.0, 30.0], [35.0, 35.0], [5.0, 35.0]]	[[7.5, 5.0], [7.5, 32.5], [35.0, 32.5]]	("Points": [(5.0, 5.0), [10.0, 5.0], [10.0, 30.0], [35.0, 30.0], [35.0, 35.0], [5.0, 35.0]], "Lines": [[0, 1], [1, 2], [2, 3], [3, 4], [4, 5], [5, 0]], "Segments": [[0, 1, 2, 3, 4, 5]])	("Points": [(7.5, 5.0), [7.5, 32.5], [35.0, 32.5]], "Lines": [[0, 1], [1, 2]], "Segments": [[0, 1]])
Plus	[[0.0, 25.0], [10.0, 25.0], [10.0, 45.0], [15.0, 45.0], [15.0, 25.0], [25.0, 25.0], [25.0, 20.0], [15.0, 20.0], [15.0, 0.0], [10.0, 0.0], [10.0, 20.0], [0.0, 20.0]]	[[12.5, 0.0], [12.5, 22.5], [12.5, 45.0], [0.0, 22.5], [25.0, 22.5]]	("Points": [[0.0, 25.0], [10.0, 25.0], [10.0, 45.0], [15.0, 45.0], [15.0, 25.0], [25.0, 25.0], [25.0, 20.0], [15.0, 20.0], [15.0, 0.0], [10.0, 0.0], [10.0, 20.0], [0.0, 20.0]], "Lines": [[0, 1], [1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [6, 7], [7, 8], [8, 9], [9, 10], [10, 11], [11, 0]], "Segments": [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]])	("Points": [[12.5, 0.0], [12.5, 22.5], [12.5, 45.0], [0.0, 22.5], [25.0, 22.5]], "Lines": [[0, 1], [4, 1], [2, 1], [3, 1]], "Segments": [[0], [1], [2], [3]])
T	[[0.0, 25.0], [25.0, 25.0], [25.0, 20.0], [15.0, 20.0], [15.0, 0.0], [10.0, 0.0], [10.0, 20.0], [0.0, 20.0]]	[[12.5, 0.0], [12.5, 22.5], [25.0, 22.5], [0.0, 22.5]]	("Points": [[0.0, 25.0], [25.0, 25.0], [25.0, 20.0], [15.0, 20.0], [15.0, 0.0], [10.0, 0.0], [10.0, 20.0], [0.0, 20.0]], "Lines": [[0, 1], [1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [6, 7], [7, 0]], "Segments": [[0, 1, 2, 3, 4, 5, 6, 7]])	("Points": [[12.5, 0.0], [12.5, 22.5], [25.0, 22.5], [0.0, 22.5]], "Lines": [[0, 1], [1, 2], [3, 1]], "Segments": [[0], [1], [2]])
I_scaled_2	[[10.0, 10.0], [20.0, 10.0], [20.0, 40.0], [10.0, 40.0]]	[[15.0, 10.0], [15.0, 40.0]]	("Points": [[10.0, 10.0], [20.0, 10.0], [20.0, 40.0], [10.0, 40.0]], "Lines": [[0, 1], [1, 2], [2, 3], [3, 0]], "Segments": [[0, 1, 2, 3]])	("Points": [[15.0, 10.0], [15.0, 40.0]], "Lines": [[0, 1]], "Segments": [[0]])
L_scaled_2	[[10.0, 10.0], [20.0, 10.0], [20.0, 60.0], [70.0, 60.0], [70.0, 70.0], [10.0, 70.0]]	[[15.0, 10.0], [15.0, 65.0], [70.0, 65.0]]	("Points": [[10.0, 10.0], [20.0, 10.0], [20.0, 60.0], [70.0, 60.0], [70.0, 70.0], [10.0, 70.0]], "Lines": [[0, 1], [1, 2], [2, 3], [3, 4], [4, 5], [5, 0]], "Segments": [[0, 1, 2, 3, 4, 5]])	("Points": [[15.0, 10.0], [15.0, 65.0], [70.0, 65.0]], "Lines": [[0, 1], [1, 2]], "Segments": [[0, 1]])

Column information is as below:

- ShapeName (text): name of the shape. Just for visualization/reports.
- Profile (text): List of Points coordinates (x,y) representing outer profile shape, typically closed.
- Midcurve (text): List of Points coordinates (x,y) representing inner midcurve shape, typically open.
- Profile_brep (text): Dictionary in Brep format representing outer profile shape, typically closed.
- Midcurve_brep (text): Dictionary in Brep format representing inner midcurve shape, typically open.

Each Segment is a continuous list of lines. In case of, say `Midcurve-T`, as there is an intersection, we can treat each line in a separate segment. In the case of 'Profile O', there will be two segments, one for outer lines and another for inner lines. Each line is a list of points, for now, linear. Points is a list of coordinates (x,y), later can be (x,y,z).

Once we have this Brep representations of both, profile and the corresponding midcurve, in the text form, then we can try various machine translation approaches or LLM based fine tuning or few-shots prompt engineering.

One major advantage of text based method over image based method is that image output still has stray pixels, cleaning which will be a complex task. But the text method has exact points. It may just give odd lines, which can be removed easily.

Dataset when imported looks like

```
[ ] from datasets import load_dataset, Dataset, DatasetDict

base_url = "/content/drive/MyDrive/ImpDocs/Work/AICoach/Notebooks/data/"
dataset = load_dataset("csv", data_files={"train": base_url + "midcurve_llm_train.csv",
                                          "test": base_url + "midcurve_llm_test.csv",
                                          "validation": base_url + "midcurve_llm_val.csv"})

print(dataset)

DatasetDict({
  train: Dataset({
    features: ['ShapeName', 'Profile', 'Midcurve', 'Profile_brep', 'Midcurve_brep'],
    num_rows: 793
  })
  test: Dataset({
    features: ['ShapeName', 'Profile', 'Midcurve', 'Profile_brep', 'Midcurve_brep'],
    num_rows: 99
  })
  validation: Dataset({
    features: ['ShapeName', 'Profile', 'Midcurve', 'Profile_brep', 'Midcurve_brep'],
    num_rows: 100
  })
})
```

Preprocessing is done to tokenize, generate ids and prepare attention.

```

▶ from transformers import RobertaTokenizer

tokenizer = RobertaTokenizer.from_pretrained("Salesforce/codet5-small")

prefix = "Skeletonize the Profile: "
max_input_length = 256
max_target_length = 128

def preprocess_examples(examples):
    # encode the code-docstring pairs
    profiles = examples['Profile_brep']
    midcurves = examples['Midcurve_brep']

    inputs = [prefix + profile for profile in profiles]
    model_inputs = tokenizer(inputs, max_length=max_input_length, padding="max_length", truncation=True)

    # encode the summaries
    labels = tokenizer(midcurves, max_length=max_target_length, padding="max_length", truncation=True).input_ids

    # important: we need to replace the index of the padding tokens by -100
    # such that they are not taken into account by the CrossEntropyLoss
    labels_with_ignore_index = []
    for labels_example in labels:
        labels_example = [label if label != 0 else -100 for label in labels_example]
        labels_with_ignore_index.append(labels_example)

    model_inputs["labels"] = labels_with_ignore_index

    return model_inputs

```

The dataset is used to fine-tune the base model called "Salesforce/codet5-small".
Trained model is saved and used for inference on test samples.

```

[ ] test_example = dataset_infr['test'][2]
    print("Profile_brep:", test_example['Profile_brep'])

Profile_brep: "{\\"Points\\": [[-6.96, 1.23], [-9.83, 5.32], [-22.12, -3.28], [-19.25, -7.38]], \\"Lines\\": [[0, 1], [1, 2], [2, 3], [3, 0]], \\"Segments\\": [[0, 1, 2, 3]]}"

[ ] # prepare for the model
    input_ids = tokenizer(test_example['Profile_brep'], return_tensors='pt').input_ids
    # generate
    outputs = model.generate(input_ids)
    print("Generated Midcurve:", tokenizer.decode(outputs[0], skip_special_tokens=True))

/usr/local/lib/python3.10/dist-packages/transformers/generation/utils.py:1273: UserWarning: Using the model-agnostic default `max_length` (=20)
  warnings.warn(
Generated Midcurve: "{\\"Points\\": [[-8.83, 4.32], [-21.23

```

The output predicted far away from the corresponding ground truth.

```

[ ] print("Ground truth:", test_example['Midcurve_brep'])

Ground truth: "{\\"Points\\": [[-8.4, 3.28], [-20.68, -5.33]], \\"Lines\\": [[0, 1]], \\"Segments\\": [[0]]}"

```

There could be many reasons, quality of LLM, training parameters, and above all, need for a far bigger dataset for fine-tuning.

Conclusions

This research not only advances our understanding of midcurve computation but also underscores the challenges and potential avenues for further exploration. The intricate interplay between traditional methodologies and cutting-edge approaches, particularly the integration of Large Language Models (LLMs) in midcurve generation, reveals the nuanced nature of geometric dimension reduction. The delineation of problems associated with variable-length inputs, the representation of complex shapes, and the limitations of existing models like Seq2Seq and Image2Image networks necessitate a paradigm shift.

The introduced MidcurveNN, utilizing Encoder-Decoder neural networks for supervised learning, offers a promising trajectory in overcoming the hurdles associated with midcurve computation. The innovative application of Brep structures in fine-tuning LLMs represents a commendable endeavor towards geometric precision. However, the observed dissonance between predicted outputs and ground truths highlights the imperative for more extensive datasets and refined training parameters.

In essence, this research serves as a catalyst for the evolution of midcurve computation methodologies. The seamless integration of LLMs, the exploration of novel geometric representations, and the introspection into fine-tuning mechanisms collectively contribute to the ongoing discourse in geometric dimension reduction. As we navigate the intricate landscape of midcurve generation, this work invites further scrutiny, refinement, and advancements, beckoning researchers and practitioners to delve deeper into the transformative intersection of geometry and advanced machine learning paradigms.

Yogesh H. Kulkarni <http://orcid.org/0000-0001-5431-5727>

References

- Github repo: <https://github.com/yogeshhk/MidcurveNN>
- MidcurveNN: Encoder-Decoder Neural Network for Computing Midcurve of a Thin Polygon, viXra.org e-Print archive, viXra:1904.0429 <http://vixra.org/abs/1904.0429>
- CAD Conference 2021, Barcelona, pages 223-225
http://www.cad-conference.net/files/CAD21/CAD21_223-225.pdf
- CAD & Applications 2022 Journal paper 19(6)
[http://www.cad-journal.net/files/vol_19/CAD_19\(6\)_2022_1154-1161.pdf](http://www.cad-journal.net/files/vol_19/CAD_19(6)_2022_1154-1161.pdf)
- Medium story Geometry, Graphs and GPT talks about using LLMs (Large Language Models) to see if geometry serialized as line-list can predict the midcurve.
- Github repository [yogeshhk/MidcurveNN](https://github.com/yogeshhk/MidcurveNN)
- Vixra paper MidcurveNN: Encoder-Decoder Neural Network for Computing Midcurve of a Thin Polygon, viXra.org e-Print archive, viXra:1904.0429 <http://vixra.org/abs/1904.0429>

- ODSC proposal

<https://confengine.com/odsc-india-2019/proposal/10090/midcurvenn-encoder-decoder-neural-network-for-computing-midcurve-of-a-thin-polygon>

- CAD Conference 2021, Barcelona, pages 223-225

http://www.cad-conference.net/files/CAD21/CAD21_223-225.pdf

- CAD & Applications 2022 Journal paper 19(6)

[http://www.cad-journal.net/files/vol_19/CAD_19\(6\)_2022_1154-1161.pdf](http://www.cad-journal.net/files/vol_19/CAD_19(6)_2022_1154-1161.pdf)

- Google Developers Dev Library

<https://devlibrary.withgoogle.com/products/ml/repos/yogeshhk-MidcurveNN>

- Medium story [Geometry, Graphs and

GPT](<https://medium.com/technology-hits/geometry-graphs-and-gpt-2862d6d24866>) talks about using LLMs (Large Language Models) to see if geometry serialized as line-list can predict the midcurve. An [extended

abstract](https://github.com/yogeshhk/MidcurveNN/blob/master/TalksPublications/MidcurveLLM/MidcurveLLM_content.md) on the same topic.