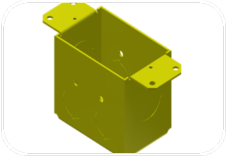
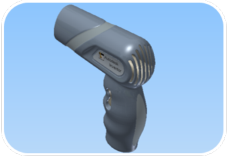
Use of Neural Networks for Geometric Problems

Yogesh Haribhau Kulkarni

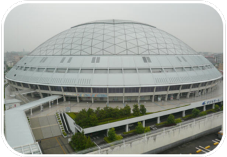
Introduction To Midcurve

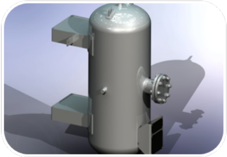
Aerospace


Machinery


Consumer
Products

Energy

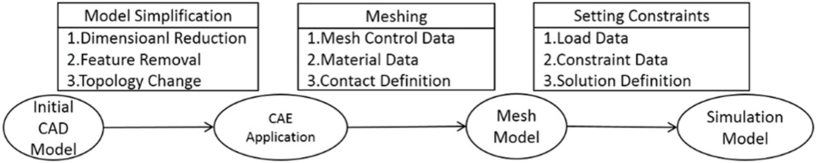

Construction


Industrial
equipment

YHK

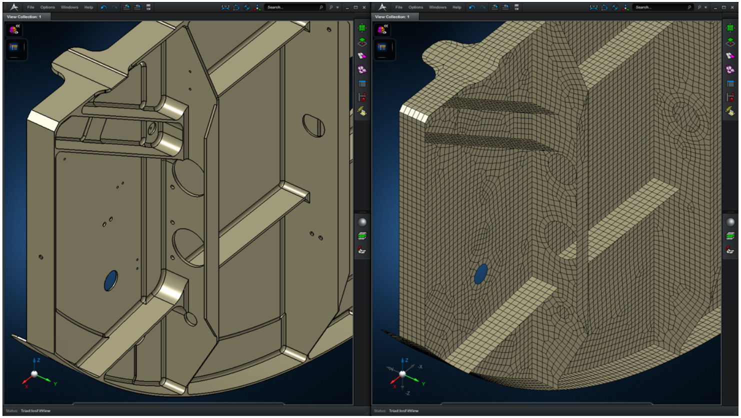
Can we use shapes directly?

- CAD : Designing Shapes
- CAE : Engineering Analysis
- CAD→CAE: Simplification for quicker results.



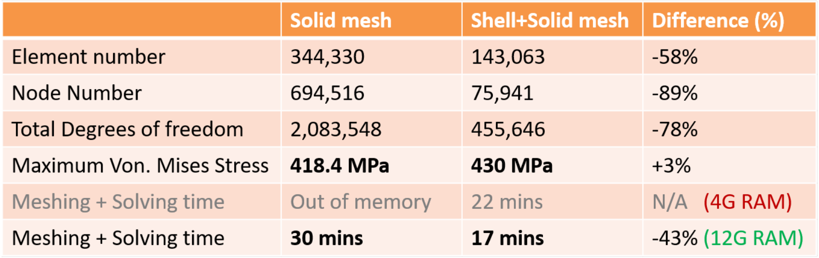
YHK

CAD-CAE


File Options Windows Help
View Collection: 1
Status: Triad:IsoFitView
File Options Windows Help
View Collection: 1
Status: Triad:IsoFitView

YHK

For Shapes like Sheet Metal . . .

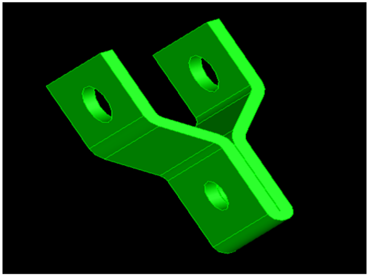


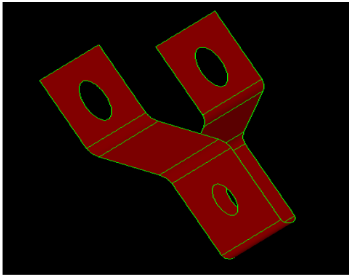
	Solid mesh	Shell+Solid mesh	Difference (%)
Element number	344,330	143,063	-58%
Node Number	694,516	75,941	-89%
Total Degrees of freedom	2,083,548	455,646	-78%
Maximum Von. Mises Stress	**418.4 MPa**	**430 MPa**	+3%
Meshing + Solving time	Out of memory	22 mins	N/A (4G RAM)
Meshing + Solving time	**30 mins**	**17 mins**	-43% (12G RAM)

Half the computation time, but similar accuracy

YHK

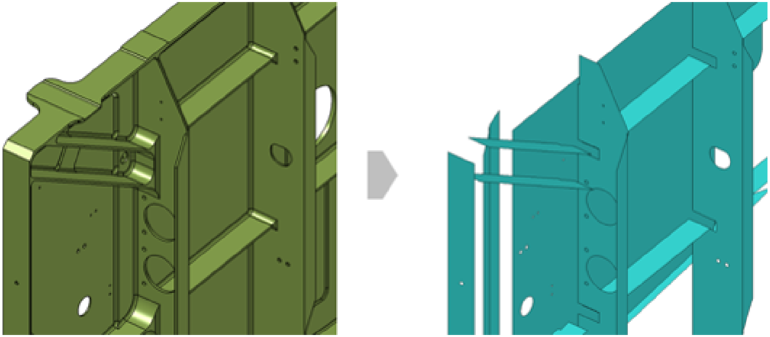
Midsurface is?


Input: Solid


Output: Midsurface

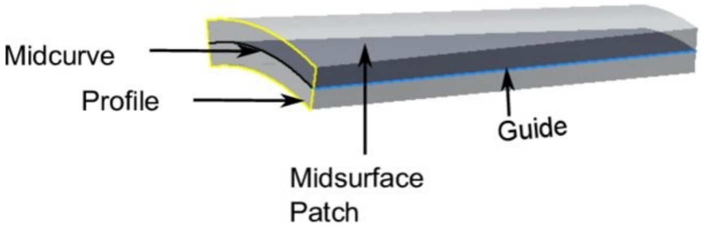
- Widely used for CAE of Thin-Walled parts
- Computation is challenging and still unsolved

Look at the output



Midsurface Computation

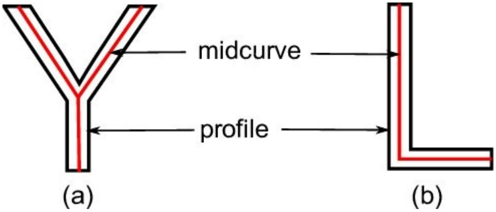
- Midsurface of a Patch is Midcurve of its profile extruded.
- So, it boils down to computing 1D midcurve of a 2D profile



YHK

What is a Midcurve?

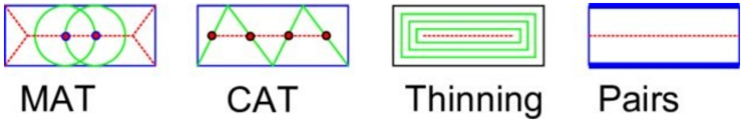
- Midsurface : From 3D thin Solid to 2D Surface
- Midcurve : From 2D Profile to 1D Curve



YHK

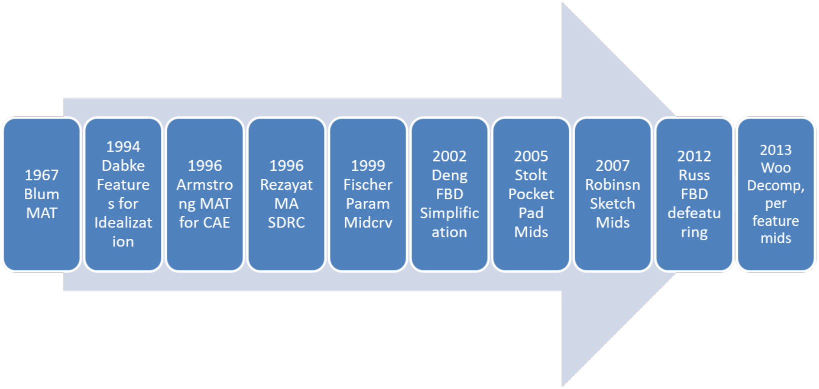
Many Approaches

- More than 6 decades of research. . .
- Most CAD-CAE packages. . .
- Rule-based!! Heuristic!! Case-by-case basis!!

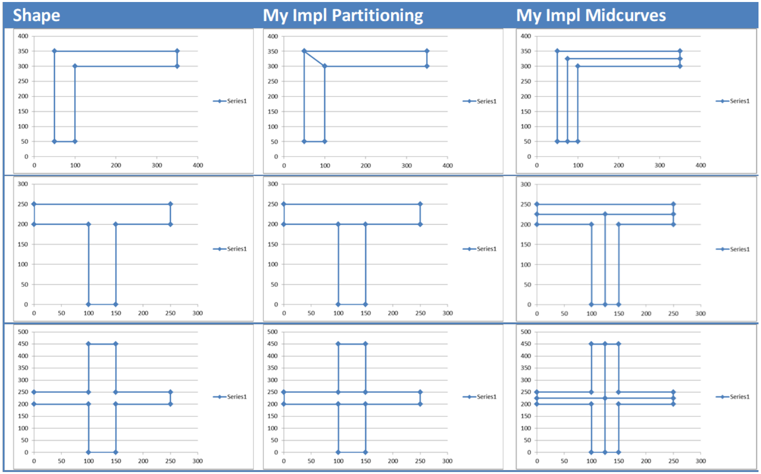


YHK

When-What?


1967 Blum MAT
1994 Dabke Features for Idealization
1996 Armstrong MAT for CAE
1996 Rezayat MA SDRC
1999 Fischer Param Midcrv
2002 Deng FBD Simplification
2005 Stolt Pocket Pad Mids
2007 Robinsn Sketch Mids
2012 Russ FBD defeaturing
2013 Woo Decomp, per feature mids

2017: My PhD Work: Rule-based



Limitations

- Fully rule-based
- Need to adjust for new shapes
- So, not scalable

Midcurve : The Problem

- **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

Midcurve : Graph 2 Graph

- **Input**: Graph of Input profile with vertices at nodes and lines/curves as edges
- **Output**: another Graph of Output profile with vertices at nodes and lines/curves as edges, open/branched polygons possible
- Both, input and output shapes have different topologies (number of nodes and edges are different) but geometry also, nodes and edges have different positions and shapes. So its network 2 network problem.
- Exiting Graph algorithms like node prediction and link prediction are not useful here as, there, topology of input and output is more or less similar.
- Graph to Graph translation does not seem to evolved enough to do the expected transformation.

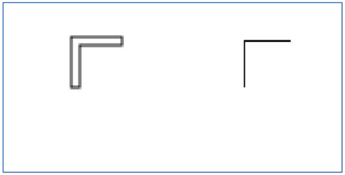
Any ideas?

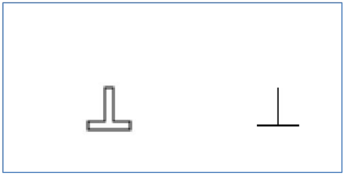
Variable Size Encoder Decoder

- OK for NLP, say Machine Translations, where padding values like “-1” can be added along with other words (vectors or indices)
- But in Geometry, its not OK.
- Because any value can represent a Valid Input, even though we don’t want it to be the input.

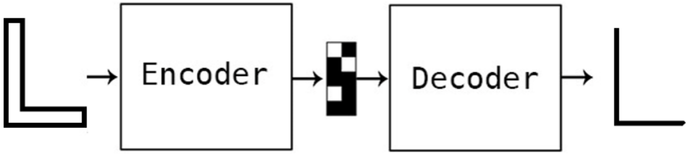
A Twist to the problem

- Input: Black & White Image of 2D profile
- Output: Black & White Image of 1D midcurve

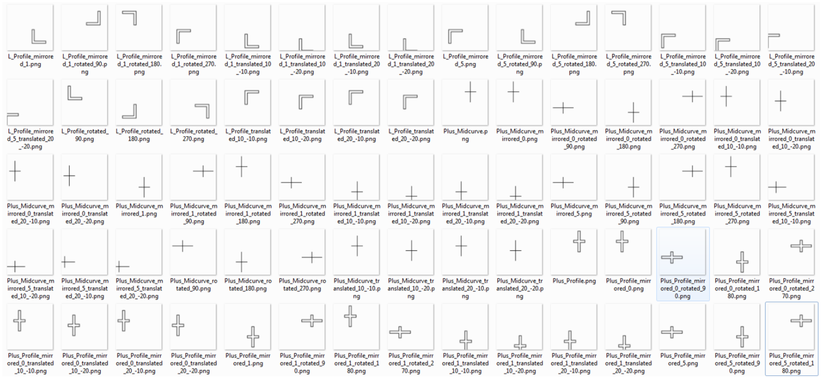




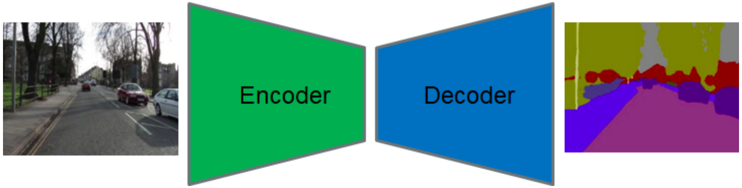
For Dimension Reduction



Training Data Samples


L_Profile_mirrored_1.png
L_Profile_mirrored_1_rotated_90.png
L_Profile_mirrored_1_rotated_180.png
L_Profile_mirrored_1_rotated_270.png
L_Profile_mirrored_1_translated_10_-10.png
L_Profile_mirrored_1_translated_10_-20.png
L_Profile_mirrored_1_translated_20_-10.png
L_Profile_mirrored_1_translated_20_-20.png
L_Profile_mirrored_5.png
L_Profile_mirrored_5_rotated_90.png
L_Profile_mirrored_5_rotated_180.png
L_Profile_mirrored_5_rotated_270.png
L_Profile_mirrored_5_translated_10_-10.png
L_Profile_mirrored_5_translated_10_-20.png
L_Profile_mirrored_5_translated_20_-10.png
L_Profile_mirrored_5_translated_20_-20.png
L_Profile_rotated_90.png
L_Profile_rotated_180.png
L_Profile_rotated_270.png
L_Profile_translated_10_-10.png
L_Profile_translated_10_-20.png
L_Profile_translated_20_-10.png
L_Profile_translated_20_-20.png
Plus_Midcurve.png
Plus_Midcurve_mirrored_0.png
Plus_Midcurve_mirrored_0_rotated_90.png
Plus_Midcurve_mirrored_0_rotated_180.png
Plus_Midcurve_mirrored_0_rotated_270.png
Plus_Midcurve_mirrored_0_translated_10_-10.png
Plus_Midcurve_mirrored_0_translated_10_-20.png
Plus_Midcurve_mirrored_0_translated_20_-10.png
Plus_Midcurve_mirrored_0_translated_20_-20.png
Plus_Midcurve_mirrored_1.png
Plus_Midcurve_mirrored_1_rotated_90.png
Plus_Midcurve_mirrored_1_rotated_180.png
Plus_Midcurve_mirrored_1_rotated_270.png
Plus_Midcurve_mirrored_1_translated_10_-10.png
Plus_Midcurve_mirrored_1_translated_10_-20.png
Plus_Midcurve_mirrored_1_translated_20_-10.png
Plus_Midcurve_mirrored_1_translated_20_-20.png
Plus_Midcurve_mirrored_5.png
Plus_Midcurve_mirrored_5_rotated_90.png
Plus_Midcurve_mirrored_5_rotated_180.png
Plus_Midcurve_mirrored_5_rotated_270.png
Plus_Midcurve_mirrored_5_translated_10_-10.png
Plus_Midcurve_mirrored_5_translated_10_-20.png
Plus_Midcurve_mirrored_5_translated_20_-10.png
Plus_Midcurve_mirrored_5_translated_20_-20.png
Plus_Midcurve_rotated_90.png
Plus_Midcurve_rotated_180.png
Plus_Midcurve_rotated_270.png
Plus_Midcurve_translated_10_-10.png
Plus_Midcurve_translated_10_-20.png
Plus_Midcurve_translated_20_-10.png
Plus_Midcurve_translated_20_-20.png
Plus_Profile.png
Plus_Profile_mirrored_0.png
Plus_Profile_mirrored_0_rotated_90.png
Plus_Profile_mirrored_0_rotated_180.png
Plus_Profile_mirrored_0_rotated_270.png
Plus_Profile_mirrored_0_translated_10_-10.png
Plus_Profile_mirrored_0_translated_10_-20.png
Plus_Profile_mirrored_0_translated_20_-10.png
Plus_Profile_mirrored_0_translated_20_-20.png
Plus_Profile_mirrored_1.png
Plus_Profile_mirrored_1_rotated_90.png
Plus_Profile_mirrored_1_rotated_180.png
Plus_Profile_mirrored_1_rotated_270.png
Plus_Profile_mirrored_1_translated_10_-10.png
Plus_Profile_mirrored_1_translated_10_-20.png
Plus_Profile_mirrored_1_translated_20_-10.png
Plus_Profile_mirrored_1_translated_20_-20.png
Plus_Profile_mirrored_5.png
Plus_Profile_mirrored_5_rotated_90.png
Plus_Profile_mirrored_5_rotated_180.png

Simple Encoder Decoder

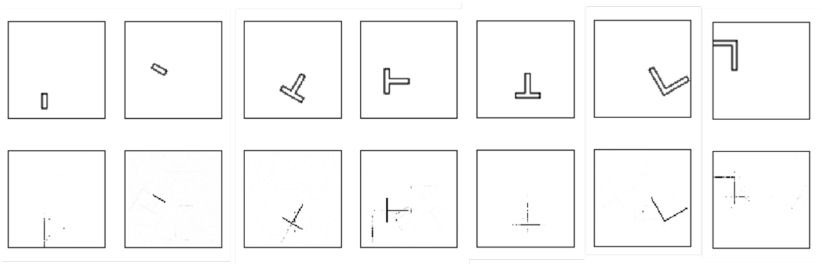


YHK

Keras Implementation

```python
input_img = Input(shape=(input_dim,))

encoded = Dense(encoding_dim,
      activation='relu',activity_regularizer=regularizers.l1(10e-5))(input_img)
decoded = Dense(input_dim, activation='sigmoid')(encoded)

autoencoder = Model(input_img, decoded)

encoder = Model(input_img, encoded)
encoded_input = Input(shape=(encoding_dim,))
decoder_layer = autoencoder.layers[-1]
decoder = Model(encoded_input, decoder_layer(encoded_input))

autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
```

Results



Idea

Can Large Language Models "learn" the dimension reduction transformation?

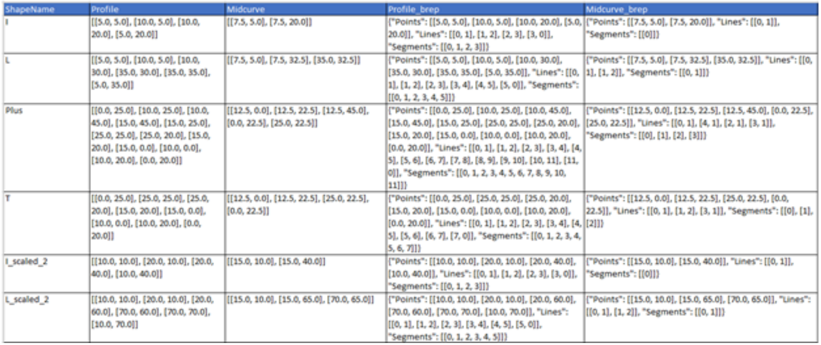
2D Brep Representation

Leverage a geometry representation similar to that found in 3D B-rep (Boundary representation), but in 2D. It can be shown as:

```
{
 '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]]
    },
}
```

YHK

Data



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]]}

YHK

Few Shots Prompt

```
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: ...

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)), ... ((5.0,35.0), (5.0,5.0))]
output: [((7.5,5.0), (7.5, 32.5)), ... ((35.0, 32.5) (7.5, 32.5))]

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

:
input:[((0, 25.0), (25.0,25.0)),...  ((0,20.0),(0, 25.0))]
output:
```

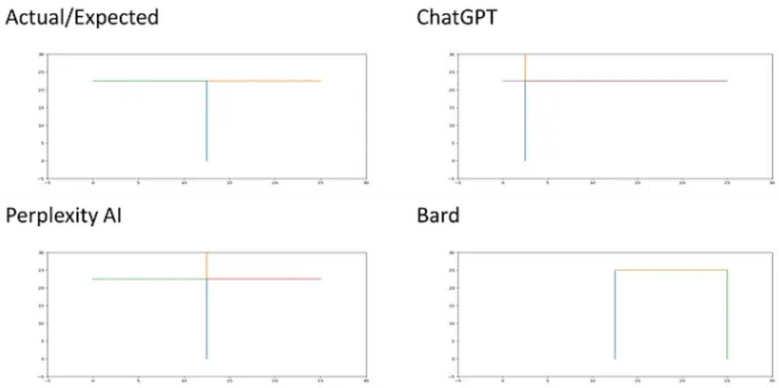
Output

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))]
```

Output

Visually here is how results from different LLMs look:



References

- Kulkarni, Y. H.; Deshpande, S. Medial Object Extraction - A State of the Art In International Conference on Advances in Mechanical Engineering, SVNIT, Surat, 2010.
- Kulkarni, Y. H.; Sahasrabudhe, A.D.; Kale, M.S Dimension-reduction technique for polygons In International Journal of Computer Aided Engineering and Technology, Vol. 9, No. 1, 2017.
- Chollet, F. Building Autoencoders in Keras In https://blog.keras.io/building-autoencoders-in-keras.html , 2019.
- Video: https://www.youtube.com/embed/ZY0nuykqgoE?featureōembed
- Presentation: https://drive.google.com/file/d/1Tx5JJK1_LUflMTW-B43HNN2GDMKJMOxR/preview
- Short paper: https://vixra.org/abs/1904.0429
- Github repo, source code: https://github.com/yogeshhk/MidcurveNN

Thanks . . .

- Search **"Yogesh Haribhau Kulkarni"** on Google and follow me on LinkedIn and Medium
- Office Hours: Saturdays, 2 to 5pm (IST); Free-Open to all; email for appointment.
- Email: yogeshkulkarni at yahoo dot com

(https://www.linkedin.com/in/yogeshkulkarni/, QR by Hugging Face QR-code-AI-art-generator, with prompt as "Follow me")

YHK