

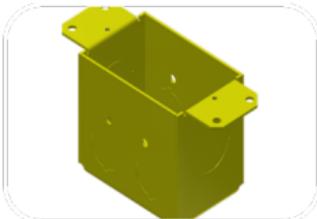
# COMPUTING MIDCURVE OF A THIN POLYGON FOR MECHANICAL ENGINEERING

Yogesh Haribhau Kulkarni

# Introduction To Midcurve



Aerospace



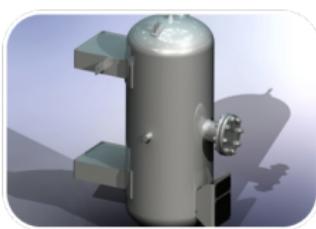
Machinery

Consumer  
Products

Energy

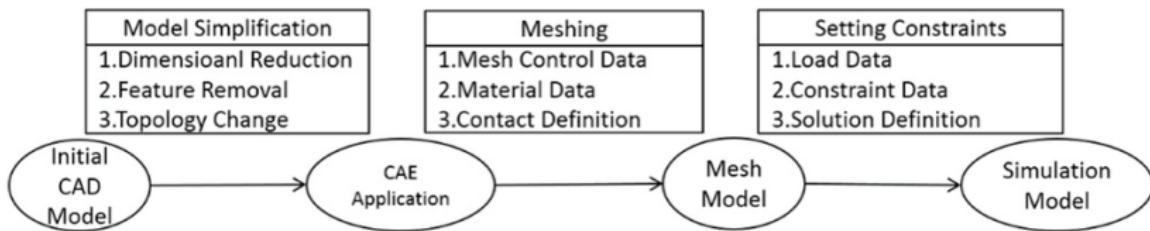


Construction

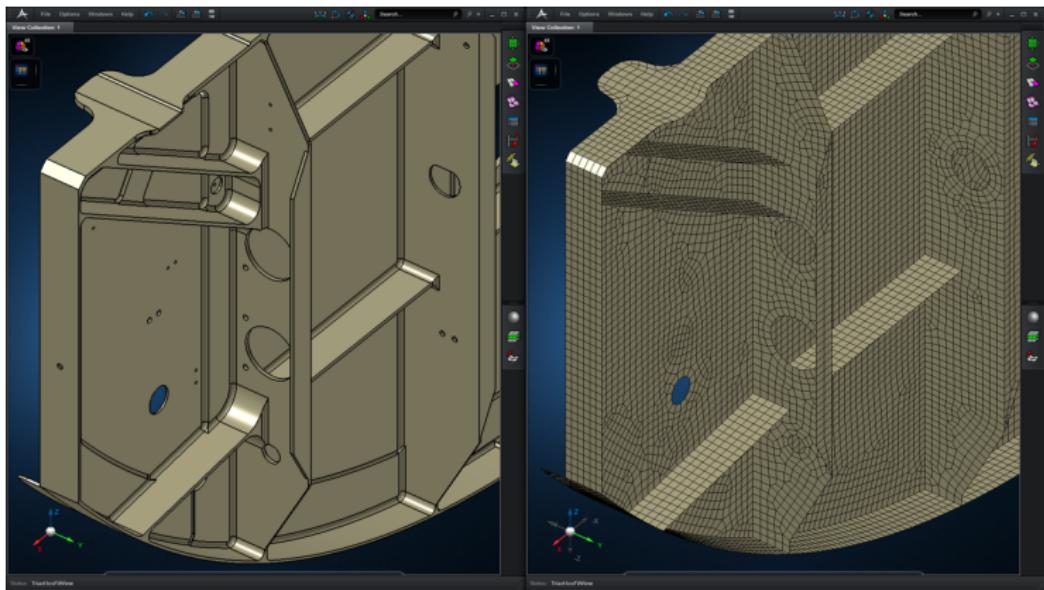
Industrial  
equipment

## Can we use shapes directly?

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



# CAD-CAE

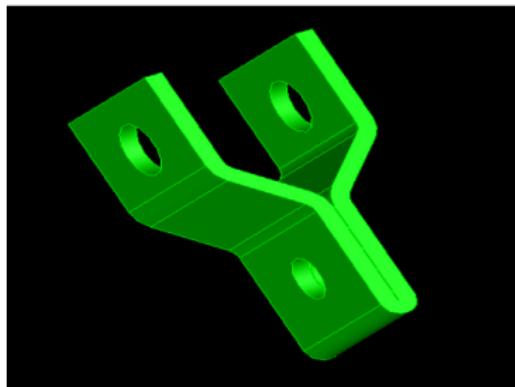


## For Shapes like Sheet Metal ...

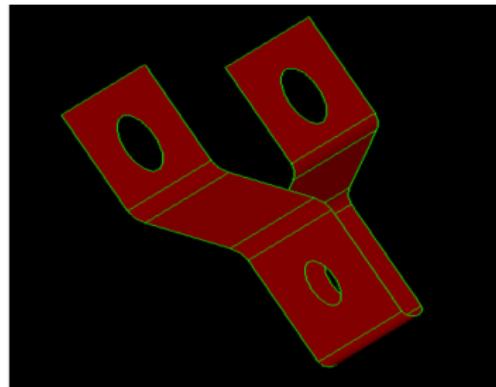
	Solid mesh	Shell+Solid mesh	Difference (%)
Element number	344,330	143,063	-58%
Node Number	694,516	75,941	-89%
Total Degrees of freedom	2,083,548	455,646	-78%
Maximum Von. Mises Stress	<b>418.4 MPa</b>	<b>430 MPa</b>	+3%
Meshing + Solving time	Out of memory	22 mins	N/A ( <b>4G RAM</b> )
Meshing + Solving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

Half the computation time, but similar accuracy

## Midsurface is?



Input: Solid

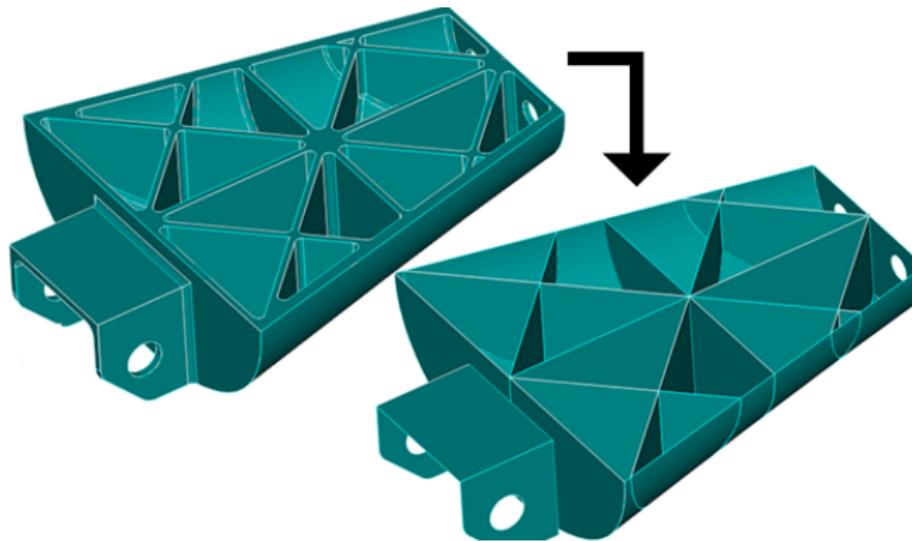


Output: Midsurface

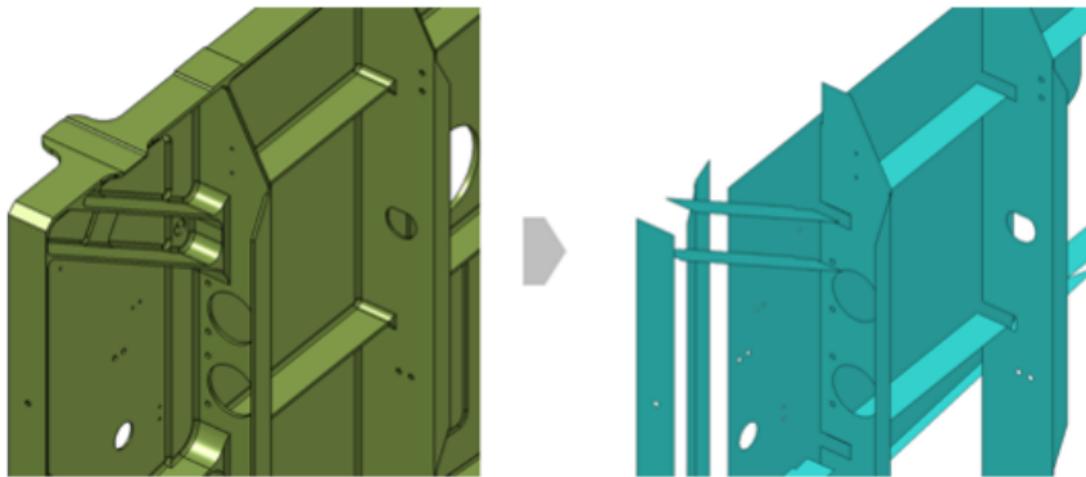
- ▶ Widely used for CAE of Thin-Walled parts
- ▶ Computation is challenging and still unsolved

## Getting Midsurface

- ▶ Going on for decades ...
- ▶ Manually by offsetting and stitching, initially
- ▶ Many CAD-CAE packages give automatic option, but ...



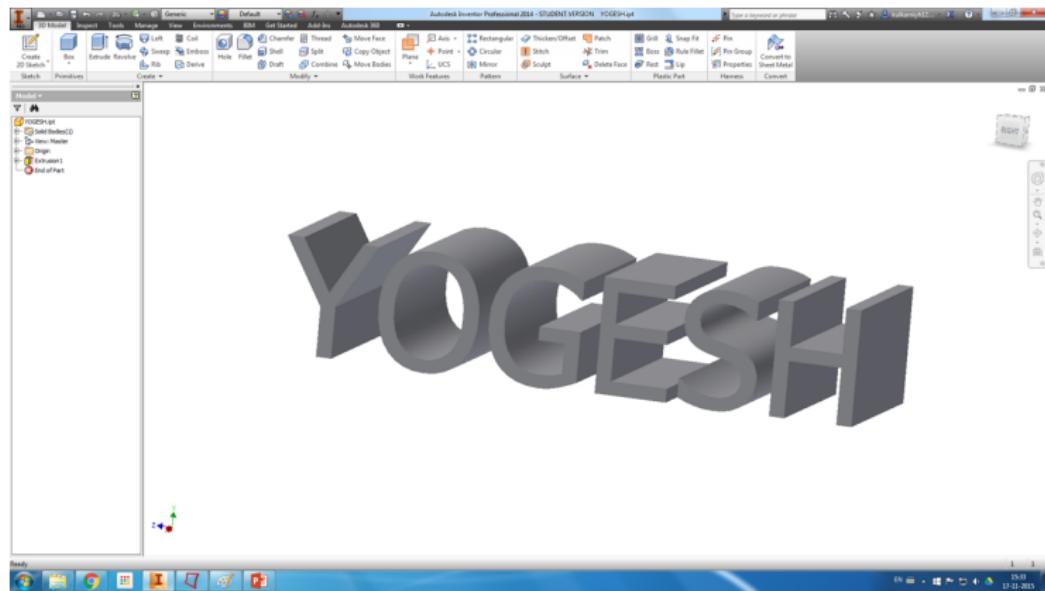
## Look at the output



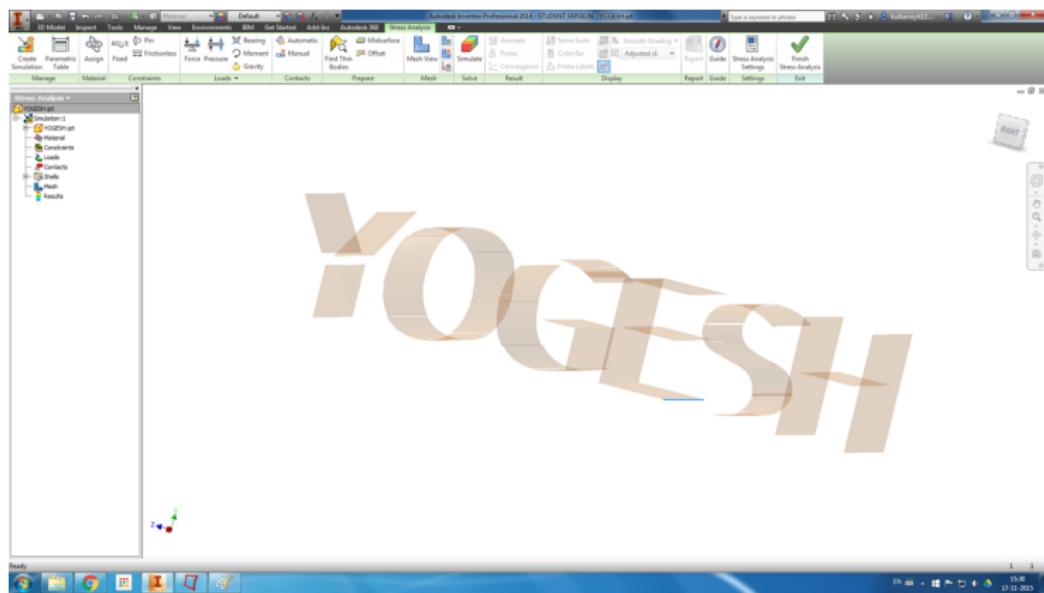
## Can't tolerate gaps

- ▶ We have thickness sampling,
- ▶ To recreate-represent the original shape
- ▶ Input and output difference not desirable

For a simple model like

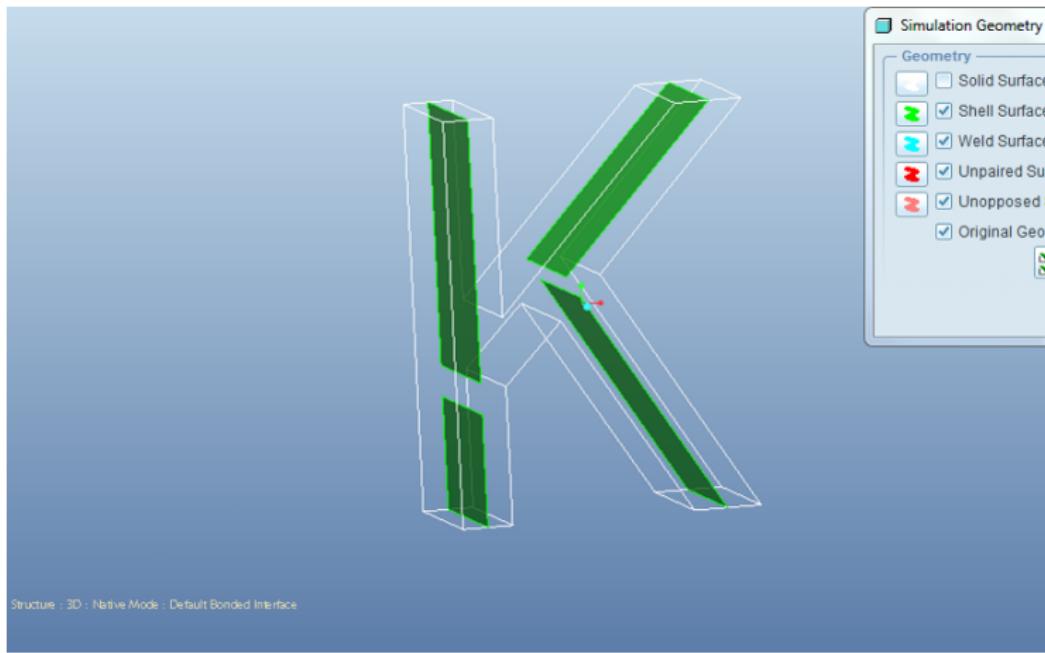


You get

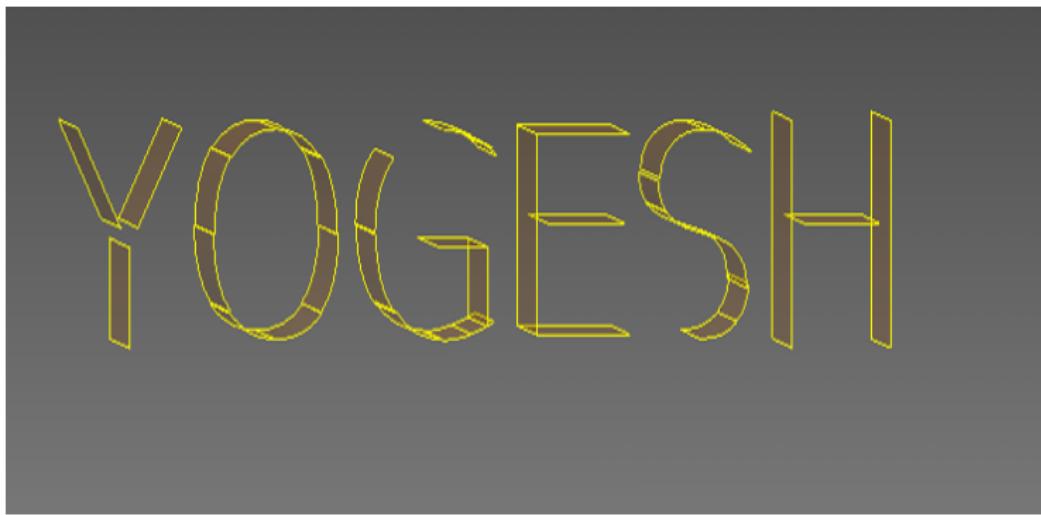


YHK

For a far simpler shape



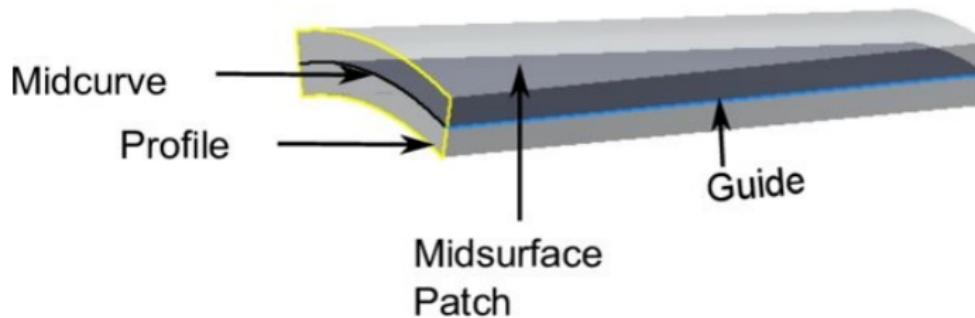
## Current Quality



- ▶ Errors take weeks to correct for complex parts.
- ▶ But still preferred, due to vast savings time
- ▶ From Days to hours ...

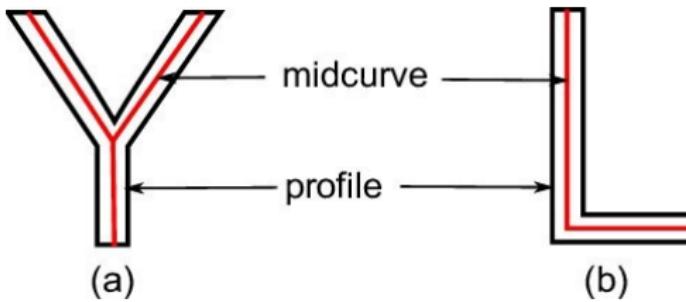
## Midsurface Computation

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



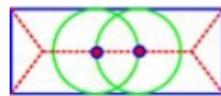
## What is a Midcurve?

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

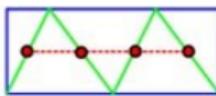


## Many Approaches

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



MAT



CAT

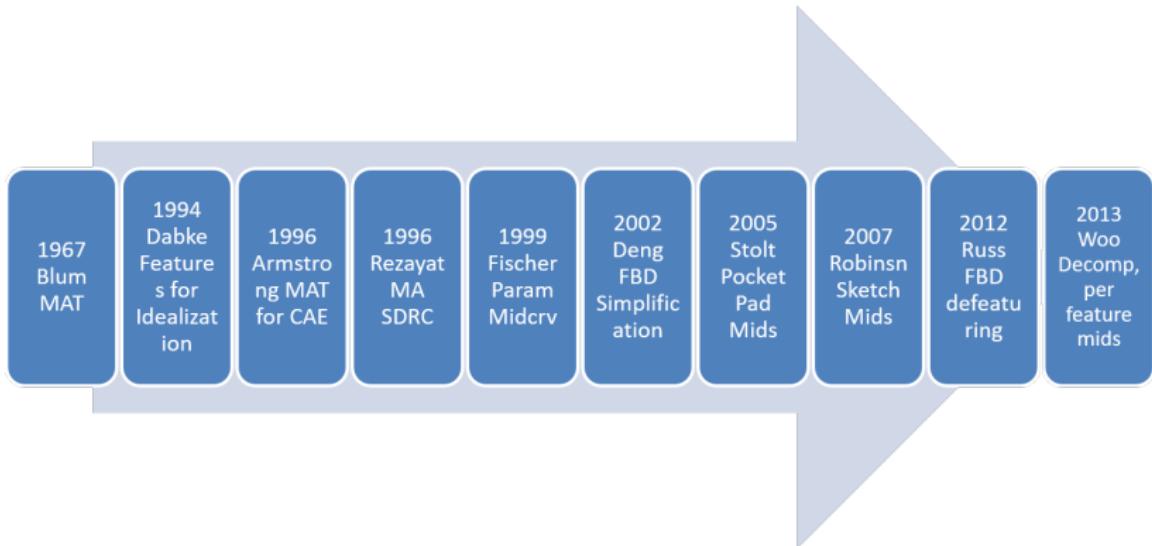


Thinning

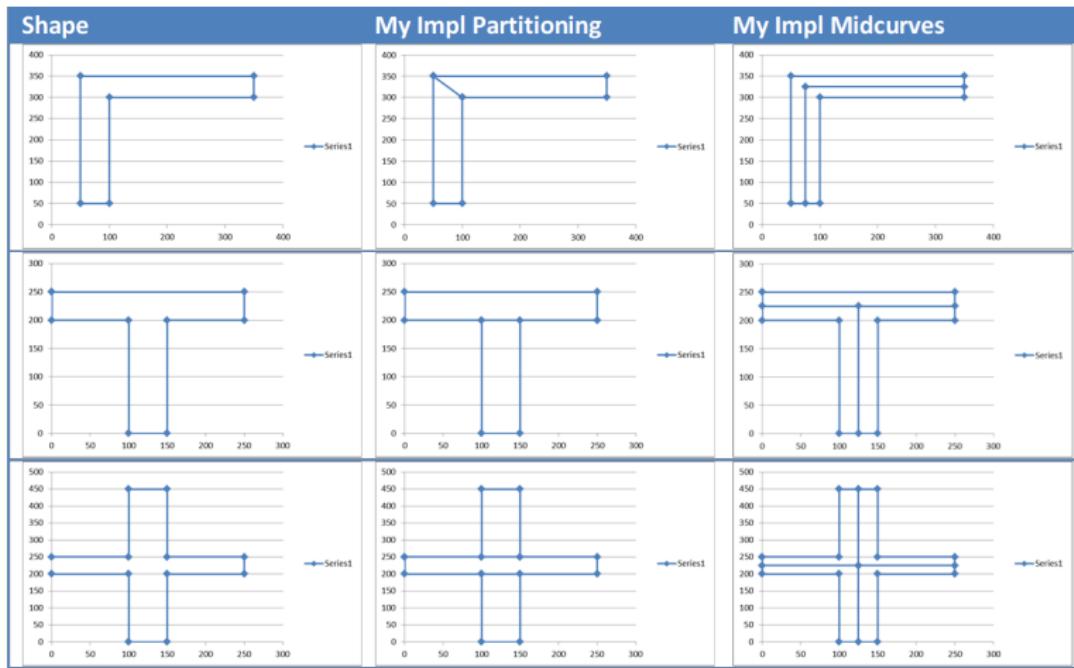


Pairs

# When-What?



# 2017: My PhD Work: Rule-based



## Limitations

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



# Idea



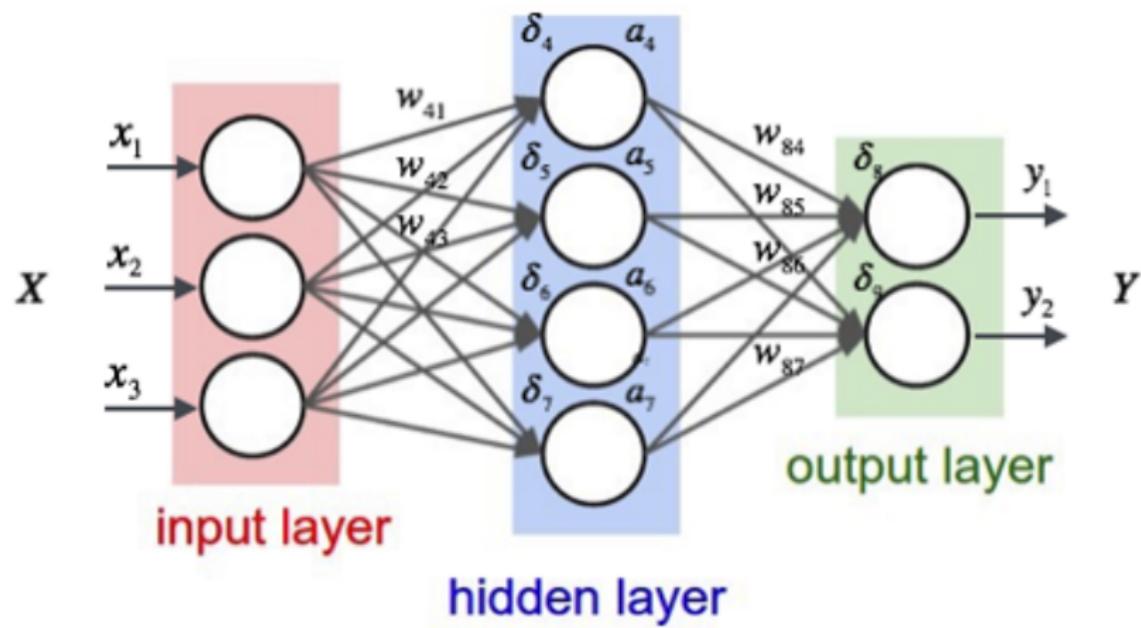
Can Neural Networks “learn” the dimension reduction transformation?

## How?

- ▶ Supply lots of training data of profiles and their corresponding midcurves and train.
- ▶ Then given an unseen profile, can Neural Network compute a midcurve, mimicking the original profile shape?



## Midcurve by Neural network



## 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

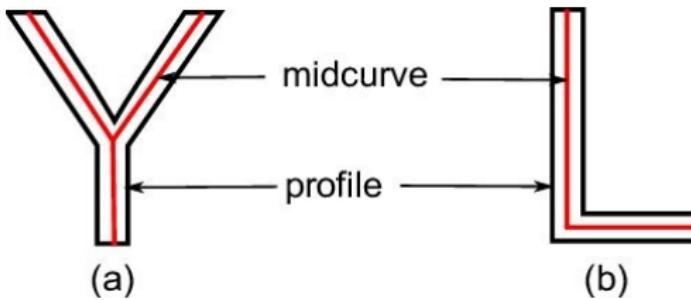
## Midcurve : Graph 2 Graph

- ▶ **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.
- ▶ Existing 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?

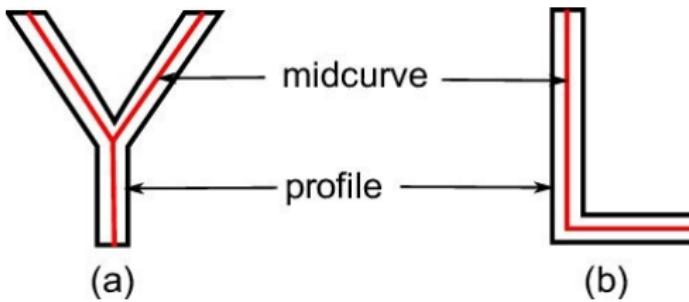
## Midcurve == Dimension Reduction

- ▶ Like PCA (Principal Component Analysis), wish to find Principal curve
- ▶ That 'represents' the original profile shape



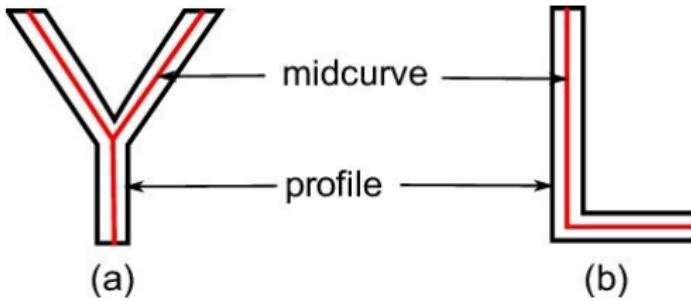
## Midcurve == Translation

- ▶ Left side (input): 2D Sketch Profile
- ▶ Right Side (output): 1D Midcurve
- ▶ Sequence 2 Sequence problem



## Midcurve $\neq$ Auto-Encoder Decoder

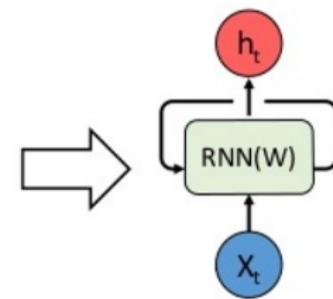
- ▶ Its not Auto-Encoder as Input and Output are different
- ▶ Its not fixed size i/o as Input and Output sizes are different



## Variable Size Encoder Decoder

- ▶ Batches need fixed lengths
- ▶ Made fixed size by Padding.

Friendly	against	Scotland	at	Murray	.
Nadim	Ladki	<PAD>	<PAD>	<PAD>	<PAD>
AL-AIN	United	Arab	Emirates	<PAD>	<PAD>
ROME	1996-12	<PAD>	<PAD>	<PAD>	<PAD>
Two	goals	in	the	last	minutes

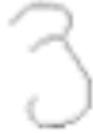


## Variable Size Encoder Decoder

- ▶ 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.



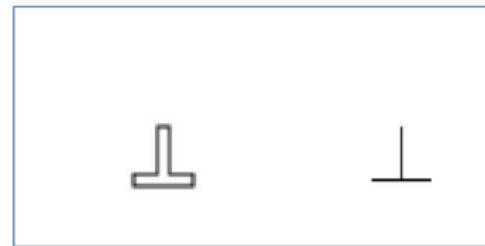
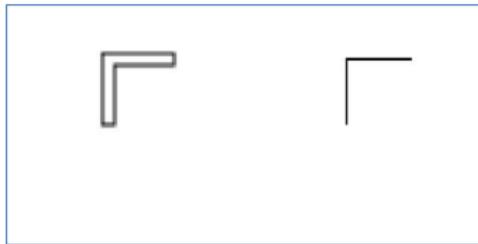
## A Twist to the problem



- ▶ Till we get good variable size encoder decoder network for geometry...
- ▶ Decided to convert this Sequence 2 Sequence problem as Image 2 Image problem.

## A Twist to the problem

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



## Solves ...

Problems of Geometric sequences

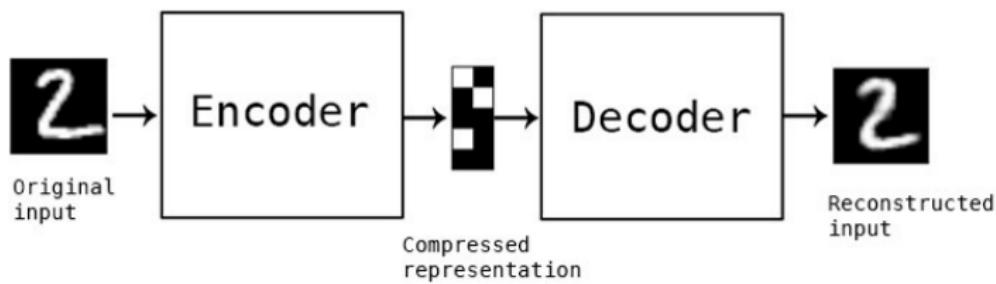
- ▶ Variable input/output sizes
- ▶ Loops need to be crossed
- ▶ Branches

I L T X K O Y

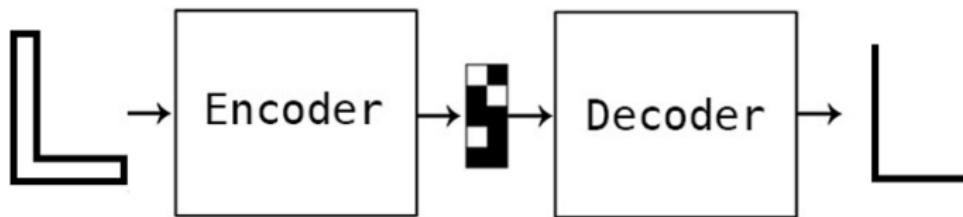
H U S D Q W

YHK

## Reuse Image Encoder Decoder



## For Dimension Reduction



## For Deep Learning

- ▶ Need lots of data
- ▶ Had just few input output image pairs
- ▶ How to augment/populate large variations ...

## Phase I

### Image to Image Transformation Learning:

- ▶ Img2Img: This phase focuses on learning image-to-image transformation using fixed-size  $100 \times 100$  bitmaps.
- ▶ Data Augmentation: The training data will be augmented by scaling, rotating, and translating both input and output shapes within the fixed size.
- ▶ Network Architecture: An Encoder-Decoder network, specifically Semantic Segmentation or Pix2Pix, will be employed for image-based dimension reduction.

# Data Preparation

## Data

Original input and output are in the form of polylines, meaning a list of points, each having x,y coordinates

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			

# Data

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		

- ▶ For each shape, we have this pair of input and output. That's it.
- ▶ We need to start with these few samples only

## Augmentation

- ▶ Such few profile shapes, are just not enough for Neural Networks to train.
- ▶ Need more with as much diversity as possible.
- ▶ Will need to artificially augment data with transformations, like pan, rotate, mirror, etc.
- ▶ All needs to be automatically, programmatically

## Geometry to Image

- ▶ Raw input data is in the Vector format
- ▶ Converted it to fixed size (100x100) image by rasterization of drawSVG library.



**Vector format**

.svg

6KB

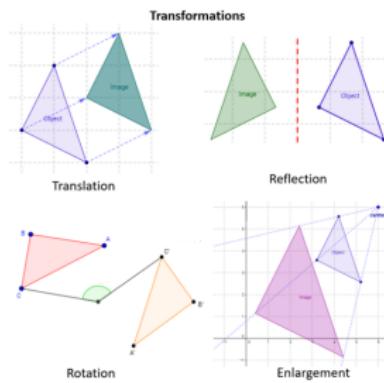


**Raster format**

.jpeg .gif .png

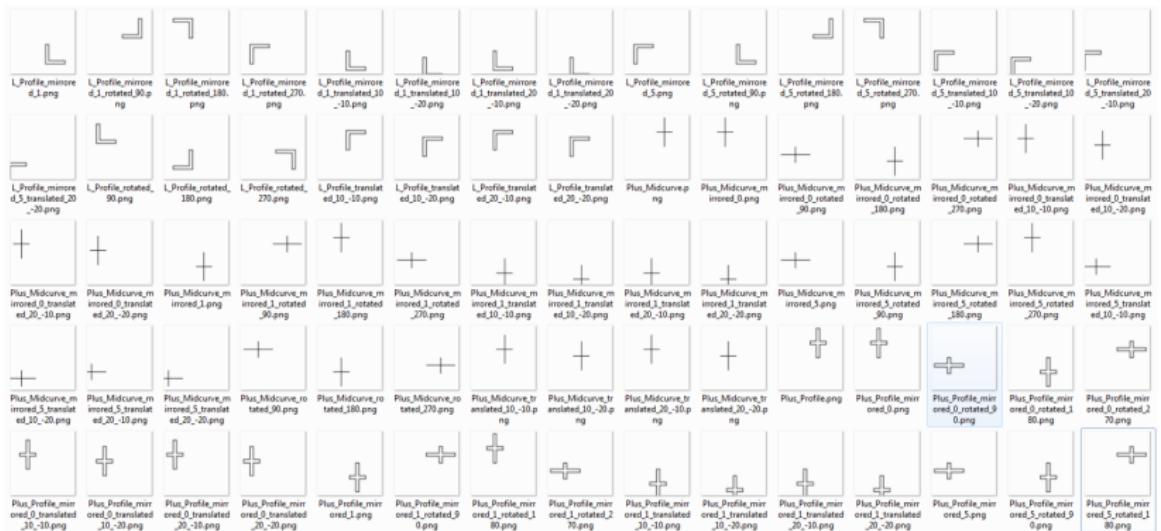
12KB

# Variations



- ▶ Inputs: I, L, Plus, T
- ▶ Operations:
  - ▶ Translated
  - ▶ Rotated
  - ▶ Mirrored
  - ▶ Mirrored Translated
  - ▶ Mirrored Rotated
- ▶ Total: 896 images (still less, but not bad)

# Training Data Samples



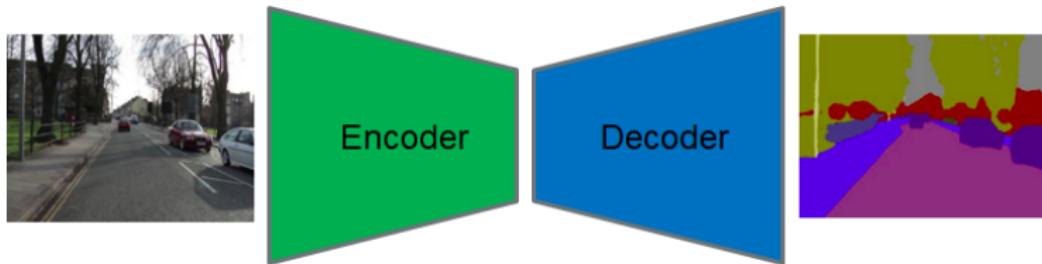
# Midcurve By Neural Network

## Options For Architectures

- ▶ Simple Encoder Decoder (one layer each)
- ▶ Dense Encoder Decoder
- ▶ Convolutional Encoder Decoder
- ▶ Pix2Pix
- ▶ ...

# Simple Encoder Decoder

# Simple Encoder Decoder



## Keras Implementation

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

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

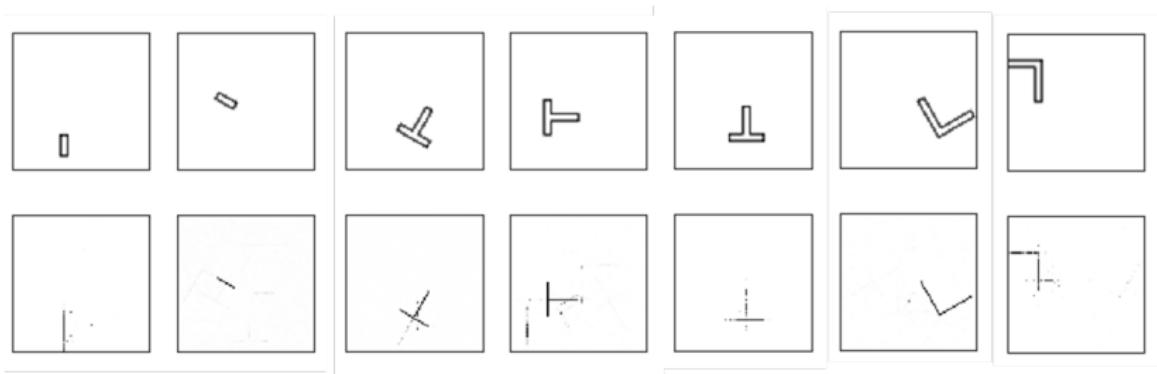
5 autoencoder = Model(input_img, decoded)

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

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

13
```

# Results



## Results

- ▶ Not very perfect but encouraging
- ▶ NN is correct with
  - ▶ The location (bounding box)
  - ▶ Dimension Reduction is seen
- ▶ But, still some stray points and misses

## What can be done?

- ▶ For the noise, use bounding boxes
- ▶ Feedback into error term: differencing with the known output expected
- ▶ Classify single pixel image as the skeleton, and rest as noise.

## What Next?

- ▶ Add denoiser network after the current one
- ▶ More Network Architectures
- ▶ Sequence-to-Sequence based approaches, taking closed thin polygon as input and polyline as output
- ▶ Extending to 3D, ie Midsurface

# Idea



Can Large Language Models “learn” the dimension reduction transformation?

# LLMs for Midcurve Implementation

## Proposed Approach

### Text to Text Transformation Learning:

- ▶ Geometric Representation: A text-based representation of the geometry/graph/network will be explored to leverage Natural Language Processing (NLP) techniques.
- ▶ Existing Methods: The paper (Fatemi, Halcrow, and Bryan 2023) surveys several text-based representations for graph data, but none appear specifically suited for geometric shapes.
- ▶ Proposed Approach: A geometry representation similar to 2D Boundary Representation (B-rep) will be utilized, adapting the concept from 3D to 2D.

## Representation of Geometry

- ▶ 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).

## 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:

```
1  {
2      'ShapeName': 'I',
3      'Profile': [(5.0, 5.0), (10.0, 5.0), (10.0, 20.0), (5.0, 20.0)],
4      'Midcurve': [(7.5, 5.0), (7.5, 20.0)],
5      'Profile_brep': {
6          'Points': [(5.0, 5.0), (10.0, 5.0), (10.0, 20.0),(5.0, 20.0)], # list of
7              (x,y) coordinates
8          'Lines': [[0, 1], [1, 2], [2, 3], [3, 0]], # list of point ids (ie index
9              in the Points list)
10             'Segments': [[0, 1, 2, 3]] # list of line ids (ie index in
11             Lines list)
12     },
13     'Midcurve_brep': {
14         'Points': [(7.5, 5.0), (7.5, 20.0)],
15         'Lines': [[0, 1]],
16             'Segments': [[0]]
17     },
18 }
```

# Data

ShapeName	Profile	Midcurve	Profile_brep	Midcurve_brep
I	<code>[[5.0, 5.0], [10.0, 5.0], [10.0, 20.0], [5.0, 20.0]]</code>	<code>[[7.5, 5.0], [7.5, 20.0]]</code>	<code>{"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]]}</code>	<code>{"Points": [[7.5, 5.0], [7.5, 20.0]], "Lines": [[0, 1]], "Segments": [[0]]}</code>
L	<code>[[5.0, 5.0], [10.0, 5.0], [10.0, 30.0], [35.0, 30.0], [35.0, 35.0], [5.0, 35.0]]</code>	<code>[[7.5, 5.0], [7.5, 32.5], [35.0, 32.5]]</code>	<code>{"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]]}</code>	<code>{"Points": [[7.5, 5.0], [7.5, 32.5], [35.0, 32.5]], "Lines": [[0, 1], [1, 2]], "Segments": [[0, 1]]}</code>
Plus	<code>[[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], [10.0, 0.0], [10.0, 20.0], [0.0, 20.0]]</code>	<code>[[12.5, 0.0], [12.5, 22.5], [12.5, 45.0], [0.0, 22.5], [25.0, 22.5]]</code>	<code>{"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], [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]]}</code>	<code>{"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]]}</code>
T	<code>[[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]]</code>	<code>[[12.5, 0.0], [12.5, 22.5], [25.0, 22.5], [0.0, 22.5]]</code>	<code>{"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]]}</code>	<code>{"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]]}</code>
t_scaled_2	<code>[[10.0, 10.0], [20.0, 10.0], [20.0, 40.0], [10.0, 40.0]]</code>	<code>[[15.0, 10.0], [15.0, 40.0]]</code>	<code>{"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]]}</code>	<code>{"Points": [[15.0, 10.0], [15.0, 40.0]], "Lines": [[0, 1]], "Segments": [[0]]}</code>
t_scaled_2	<code>[[10.0, 10.0], [20.0, 10.0], [20.0, 60.0], [70.0, 60.0], [70.0, 70.0], [10.0, 70.0]]</code>	<code>[[15.0, 10.0], [15.0, 65.0], [70.0, 65.0]]</code>	<code>{"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]]}</code>	<code>{"Points": [[15.0, 10.0], [15.0, 65.0], [70.0, 65.0]], "Lines": [[0, 1], [1, 2]], "Segments": [[0, 1]]}</code>

## Data Explanation

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).

## Data Resolution

- ▶ 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.

# Prompt based techniques

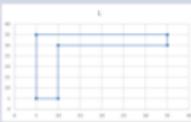
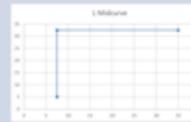
YHK

## Few Shots Prompt

- 1 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: ...
  - 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`.
- 5     :[((5.0,5.0), (10.0,5.0)), ... ((5.0,35.0), (5.0,5.0))]
- 7     output: [((7.5,5.0), (7.5, 32.5)), ... ((35.0, 32.5) (7.5, 32.5))]
- 9     : [((5,5), (10, 5)), ... (5, 20)), ((5, 20),(5,5))]  
      output: [((7.5, 5), (7.5, 20))]
- 11    :  
13    :[((0, 25.0), (25.0,25.0)),... ((0,20.0),(0, 25.0))]  
      output:

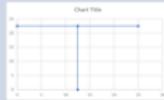
## Few Shot Examples Shapes

The first input example above represents '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			

## Few Shot Examples Shapes

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			

## Output

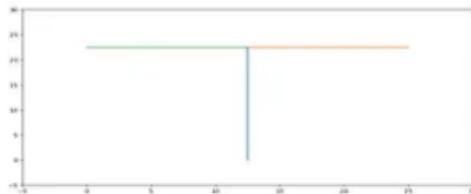
And the outputs computed by various LLMs (ChatGPT, Perplexity AI, Bard) , along with the real/actual answer:

```
Actual: [((12.5,0), (12.5,22.5)), ((12.5,22.5),(25.0,22.5)),  
        ((12.5,22.5),(0,22.5))]  
2 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))]  
4 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:

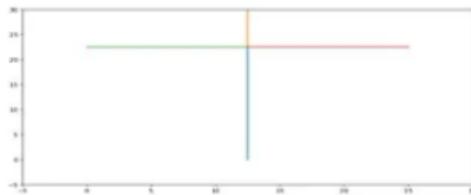
Actual/Expected



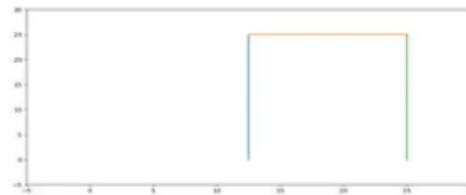
ChatGPT



Perplexity AI



Bard



## Interpretation

All of the above have failed. Even latest, 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.



## Interpretation

LLMs by an large seem to have failed for such simple shapes.

There could be two prominent reasons:

- ▶ The prompt design was not effective and could be improved upon.
- ▶ The LLM model itself is not able to learn the pattern and predict well.

The current geometry representation as a sequence of lines, has limitations.

Trying to look for a good representation to store geometry/graph/network as text so that NLP (Natural Language Techniques) can be applied.

## Fine-tuning based techniques

# Training Data

```
from datasets import load_dataset, DatasetDict
2
base_url = "/content/drive/MyDrive/ImpDocs/Work/AICoach/Notebooks/data/"
4
dataset = load_dataset("csv", data_files={"train": base_url + "midcurve_llm.train.csv",
5                               "test": base_url + "midcurve_llm.test.csv",
6                               "validation": base_url + "midcurve_llm.val.csv"})
7
DatasetDict({
8     train: Dataset({
9         features: ['ShapeName', 'Profile', 'Midcurve', 'Profile_brep', 'Midcurve_brep'],
10        num_rows: 793
11    })
12    test: Dataset({
13        features: ['ShapeName', 'Profile', 'Midcurve', 'Profile_brep', 'Midcurve_brep'],
14        num_rows: 99
15    })
16    validation: Dataset({
17        features: ['ShapeName', 'Profile', 'Midcurve', 'Profile_brep', 'Midcurve_brep'],
18        num_rows: 100
19    })
20})
21})
```

## Data Preprocessing

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.

```
1 from transformers import T5ForConditionalGeneration, AdamW, get_linear_schedule_with_warmup
2 import pytorch_lightning as pl
3
4 class CodeT5(pl.LightningModule):
5     def __init__(self, lr=5e-5, num_train_epochs=15, warmup_steps=1000):
6         super().__init__()
7         self.model_name = "Salesforce/codet5-small"
8         self.model = T5ForConditionalGeneration.from_pretrained(self.model_name)
9         self.save_hyperparameters()
10
11    def forward(self, input_ids, attention_mask, labels=None):
12        outputs = self.model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
13        return outputs
14    :
15    :
16
17 model = CodeT5()
18
19 trainer = Trainer(max_epochs=5, accelerator="auto", # gpus=1,
20                   default_root_dir="/content/drive/MyDrive/CodeT5/Notebooks/Checkpoints",
21                   logger=wandb_logger,
22                   callbacks=[early_stop_callback, lr_monitor])
23 trainer.fit(model)
```

## Inferencing

The output predicted far away from the corresponding ground truth.  
There could be many reasons for the mismatch such as the quality of LLM,  
training parameters, and above all, need for a far bigger dataset for fine-tuning.  
But overall, the approach and preliminary results look promising to warrant  
further investigations.

```
1 input_ids = tokenizer(test.example['Profile_brep'], return_tensors='pt').input_ids
  outputs = model.generate(input_ids)
3
4 print("Ground truth:", test.example['Midcurve_brep'])
5 print("Generated Midcurve:", tokenizer.decode(outputs[0], skip_special_tokens=True))
7 Ground truth: "{\"Points\": [[-8.4, 3.28], [-20.68, -5.33]], \"Lines\": [[0, 1]], \"Segments\": [[0]]}"
Generated Midcurve: "{\"Points\": [[-8.83, 4.32], [-21.23,
```

## Summary

- ▶ Traditional methods of computing midcurves are predominantly rules-based and thus, have limitation of not developing a generic model which will accept any input shape.
- ▶ A novel Large Language Model based approach attempts to build such a generic model.
- ▶ One such model, GPT-4, seems to be very effective. 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.

## Summary

- ▶ This research significantly advances midcurve computation by exploring the interplay between established methodologies and cutting-edge approaches, particularly integrating Large Language Models (LLMs).
- ▶ Emphasizing the nuanced nature of geometric dimension reduction, it identifies challenges in handling variable-length input data, representing intricate shapes, and addressing limitations in existing models.

## MidcurveLLM Architecture

- ▶ Utilizes an Encoder-Decoder architecture and B-rep structures.
- ▶ Showcases promise, despite discrepancies with ground truths.

## Implications and Future Directions

- ▶ Points to the need for more extensive datasets and refined training parameters.
- ▶ Serves as a crucial catalyst for advancing midcurve computation methodologies.
- ▶ Invites further scrutiny and advancements in the transformative intersection of geometry and advanced machine learning.

## References

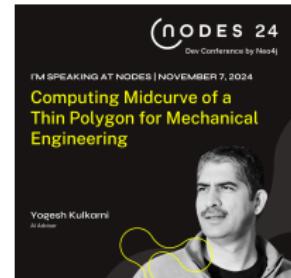
- ▶ Kulkarni, Y. H.; Deshpande, S. Medial Object Extraction - A State of the Art In International Conference on Advances in Mechanical Engineering, SVNIT, Surat, 2010.
- ▶ Kulkarni, Y. H.; Sahasrabudhe, A.D.; Kale, M.S Dimension-reduction technique for polygons In International Journal of Computer Aided 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?feature=oembed>
- ▶ Presentation:  
[https://drive.google.com/file/d/1Tx5JJK1\\_LUflMTW-B43HNN2GDMKJMOxR/view](https://drive.google.com/file/d/1Tx5JJK1_LUflMTW-B43HNN2GDMKJMOxR/view)
- ▶ Short paper: <https://vixra.org/abs/1904.0429>
- ▶ Github repo, source code: <https://github.com/yogeshhk/MidcurveNN>

# My Talk at Nodes 2024 ...

- ▶ Computing Midcurve of a Thin Polygon for Mechanical Engineering
- ▶ Session Track: Data Science
- ▶ 12:30 pm - 1:00 pm November 7
- ▶ *Calculating physical product design is a complicated problem. Existing models provide rough examples in reasonable times but still require weeks of manual correcting. This presentation explores how graphs can help mechanical engineering by modeling geometric polygonal figures into a graph representation. Neural networks can then train and learn from examples provided to compute the midcurve of a thin polygon, becoming a graph-to-graph operation akin to an encoder-decoder problem.*



(QR by <https://www.the-qrcode-generator.com/>)



## My TEDx Talk :

### Hit Refresh : A story of purposeful resets

*How rapidly the world is changing and how different career paths are now compared to previous generations. Yogesh shares his own journey of constant reinvention.*

(<https://www.youtube.com/watch?v=-VbWRs7BsPY>, QR by

<https://www.the-qrcode-generator.com/>)



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")

