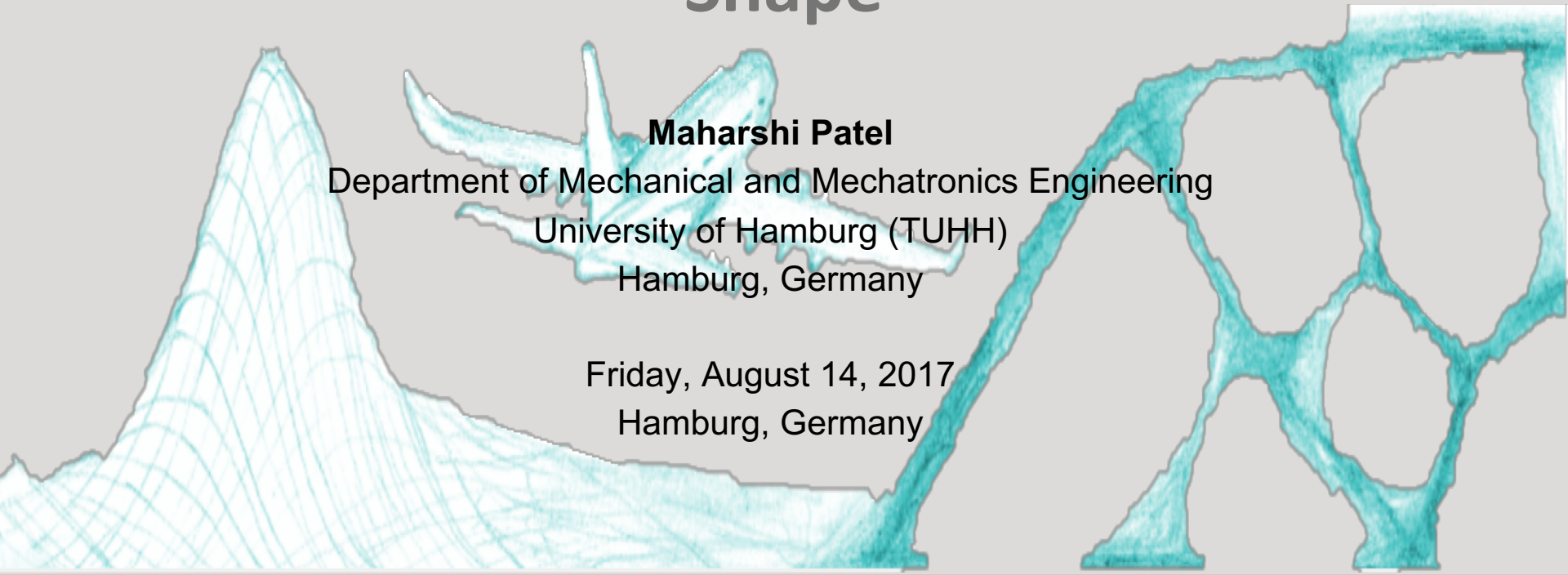
Structure and Beam Identification of 2D Bionic Shape



Maharshi Patel
Department of Mechanical and Mechatronics Engineering
University of Hamburg (TUHH)
Hamburg, Germany

Friday, August 14, 2017
Hamburg, Germany

TUHH
Technische Universität Hamburg

UNIVERSITY OF WATERLOO

Acknowledgements

Prof. Benedikt Kriegesmann
Julian Lüdeker (Phd. Candidate)
Olaf Ambrozkiewicz (Phd. Candidate)

TUHH
Technische Universität Hamburg

UNIVERSITY OF WATERLOO

Contents

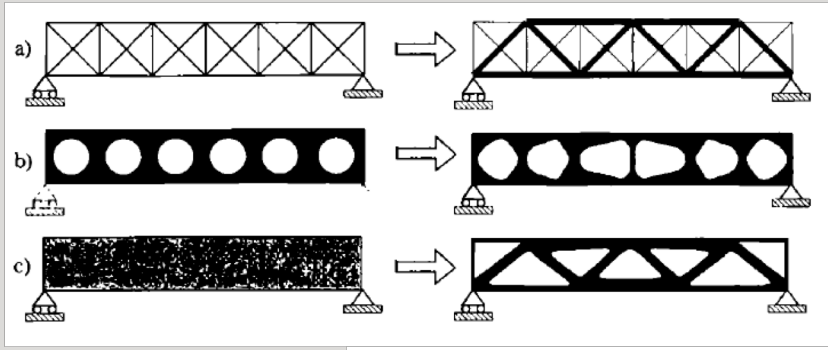
Technische Universität Hamburg

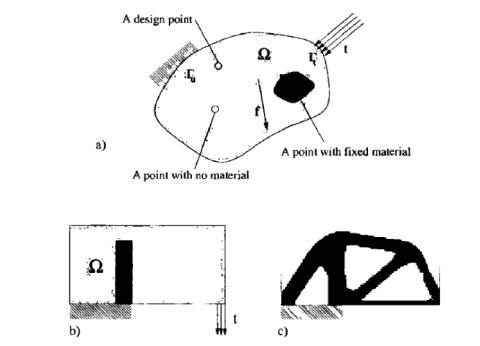
UNIVERSITY OF WATERLOO

Introduction

Topological Optimization (TO)

- Optimizes material layout within a given design space
  - Maximize performance with a given set of load and boundary conditions
- Application for additive manufacturing





TO Formulation

- $min_\rho\ F = F(u(\rho), \rho) = \int_\Omega f(u(\rho), \rho) dV$
  - » $\rho \in \{0,1\}$
  - » $G_0(\rho) = \int_\Omega \rho(u) dV - V_0 \leq 0$
  - » $G_j\big(u(\rho)\big) \leq 0\ with\ j = 1, \dots, m$

- Nomenclature
  - » Ω – Design Space
  - » $\rho(u)$ – Density per element 1/0
  - » m – constraints

Method Implementation

NURBS MESH Derivatives

TUHH
Technische Universität Hamburg

UNIVERSITY OF WATERLOO

NURBS Mesh Derivatives

- Main Functions
  - DensityToBionicShape
  - IdetifyBeams
    - Filterdata
    - Direction
    - BeamMatch
    - CenterPoint

```
function [ bionicShape ] = DensityToBionicShape( density )
    warning off;

    if nargin==0
        clc;clf('reset');
        density=load('OptStructEx5.mat');
        density=density.xe;
    end
    density=flip(density);
    [d,ang]=densityDerivations(density,0.1,0);
    [splines]=fitSplines(d,ang);
    bionicShape=BionicShape2d(splines(1),splines(2:end));
%     bionicShape.draw(1,'k');axis 'equal';
%     miny=min(nodes(:,2));
%     maxy=max(nodes(:,2));
%     midy=(maxy-miny)/2;
%     nodes(:,2)=midy-nodes(:,2);
%     model=FEModel([nodes,zeros(sizeN,1)],2);
%     model.addElementAssembly(elements,'TPN3',sizeN);
%     model.plot(0,1);
%     axis equal;
%     hold on
%     clf('reset')
    bionicShape.draw(0,'k');
    identifyBeams(bionicShape);
end

function [StoreCell, assignment, meshInline, meshOutline]=identifyBeams(bionicShape) ...

function [StoreCell] = filterdata(StoreCell) ...

function [DirectionPolyInlineStoreCell, DirectionDerivativePolyStoreCell]=direction(meshStoreCell) ...

function [] = BeamMatch(~, StoreCellDerPolyOut, StoreCell) ...

function [CenterStoreCell] = CenterPoints(StoreCell) ...

function [StoreCellPolyOut, StoreCellDerPolyOut] = BeamFittingDer(StoreCell)    ...

function [ splines ] = fitSplines(d,ang) ...

function [nodes,weights]=angleFilter(nodes,mindist,mindif) ...

function [ nodes,canidates ] = followLine(canidates,startNode,q,a) ...

function [ neighbors ]=getNeighbors(canidates,center) ...
```

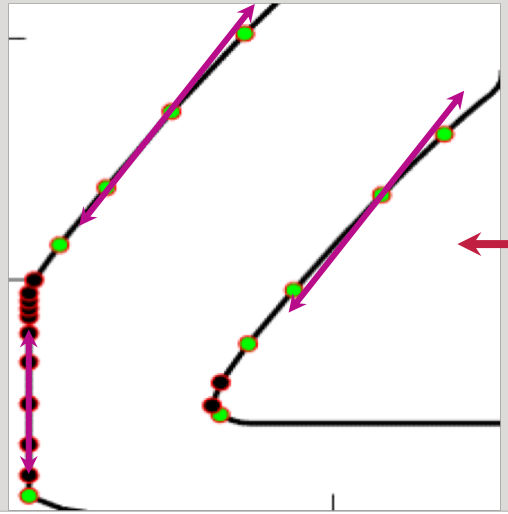
identifyBeams

- Gathering points from mesh and storing it for later use

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Gathering all the points for the bionic shape
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Outlines
%%%%%%%%%%
    meshTemp=bionicShape.outline.getMesh;
    meshOutline = meshTemp(:,1:2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Inlines
%%%%%%%%%%
    meshTempInline = cell(length(lines),1);
    for i=1:length(lines)
        line=lines(i);
        mesh=line.getMesh;
        meshTempInline{i} = [meshTempInline{i}; mesh(:,1:2)];
    end

    meshOutline(:,3) = 1;
    meshInline = meshTempInline;

    for O = 1:length(meshInline)
        meshInline{O}(:,3) = O + 1;
    end

    meshStoreCell = [meshOutline; meshInline]; % storing
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

idenfifyBeams → direction

- Calculates the derivative of each mesh point



```
%   Running direction for getting derivatives
    [~, DirectionDerivativePolyStoreCell]=direction(meshStoreCell);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
DirectionPolyInlineStoreCell = cell(length(Temp2),1);
DirectionDerivativePolyStoreCell = cell(length(Temp2),1);
for Z = 1:length(Temp2) % curve fitting each beam to know beam
    dirarray=zeros(length(Temp2{Z}),1);
    for B = 1:length(Temp2{Z})
        x = Temp2{Z}{B}(:,1);
        y = Temp2{Z}{B}(:,2);
        dof = 1;
        DirectionPolyInlineStoreCell = polyfit( x, y, dof);
        dirarray(B)=polyder(DirectionPolyInlineStoreCell);
    end
    DirectionDerivativePolyStoreCell{Z} = dirarray;
end

for i = 1:length(DirectionDerivativePolyStoreCell)
    Q = DirectionDerivativePolyStoreCell{i}(1,:);
    DirectionDerivativePolyStoreCell{i} = [DirectionDerivativePo
end
```

Cont…

- Determining the close points to compare the derivatives

```
%   Getting and Storing the lines

    Der_Tol = 0.3;
    meshInline = meshStoreCell;
    DirectionDerivativePolyInline = DirectionDerivativePolyStoreCell;
    DirectionDerivativePolyInlineTemp = DirectionDerivativePolyInline;
    meshInlineTemp = meshInline;
    onBeam = 1;
    i = 1;
    R = 1;
    for Q = 1:length(meshInline)
        inlineQ=meshInline{Q};
        meshInlineTemp{Q} = [];
        meshInlineTemp = meshInlineTemp(~cellfun('isempty',meshInlineTemp));
        dirderQ=DirectionDerivativePolyInlineTemp{Q};
        DirectionDerivativePolyInlineTemp{Q} = [];
        DirectionDerivativePolyInlineTemp = DirectionDerivativePolyInlineTemp(~cellfun('isem
        for p = 1:length(inlineQ)    % Selects 1 Lint to compare with various other inlines
            A = inlineQ(1,1:2);
            A_Temp = inlineQ(1,:);
            inlineQ(1,:)=[];
            ader=dirderQ(1);
            dirderQ(1)=[];
            for U = 1:length(meshInlineTemp)
                for V = 1:length(meshInlineTemp{U})
                    B = meshInlineTemp{U}(:,1:2);
                    distances = sqrt(sum(bsxfun(@minus, B, A).^2,2));      % compute distanc
                    [Min,ClosestBeamIndex] = min(distances);      %get index min value
                    inlineTemp(R,:) = U;
                    CloseIndexTemp(R,:) = ClosestBeamIndex;
                    MinDistanceTemp(R,:) = Min;
                end
                R = R + 1;
            end

            % Selects the min distance and index from all the inlines so it
            % make sures that it picks the closest one
            [~,inline] = min(MinDistanceTemp);
            CloseIndex = CloseIndexTemp(inline);
```

Cont…

- Storing of lines

```
        if abs(ader - (DirectionDerivativePolyInlineTemp{inline}(CloseIndex))) <= Der_Tol
            OuterPoint = A_Temp;
            InnerPoint = meshInlineTemp{inline}(CloseIndex,:);
            assignment(c) = 1;
            pj_1 = OuterPoint;
            pj_2 = InnerPoint;
            onBeam = 1;
            plot(pj_1(1),pj_1(2),'or','MarkerFaceColor','g');
            plot(pj_2(1),pj_2(2),'or','MarkerFaceColor','g');
            StoreCell{i*2-1} = [StoreCell{i*2-1}; pj_1 ];
            StoreCell{i*2} = [StoreCell{i*2}; pj_2 ];
        else
            OuterPoint = A;
            InnerPoint = meshInlineTemp{inline}(CloseIndex,:);
            assignment(c) = 0;
            pj_1 = OuterPoint;
            pj_2 = InnerPoint;
            plot(pj_1(1),pj_1(2),'or','MarkerFaceColor','k')
            plot(pj_2(1),pj_2(2),'or','MarkerFaceColor','k')
            if onBeam==1
                i = i + 1;
                onBeam = 0;
            end
        end
        R = 1;
    end
    i = i + 1;
    onBeam = 0;
    meshInlineTemp = meshInline;
    DirectionDerivativePolyInlineTemp = DirectionDerivativePolyInline;
end
StoreCell = StoreCell(~cellfun('isempty',StoreCell));
```

Cont → Other function calls

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Identifing Inlines for the bionic structure
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [StoreCell] = filterdata(StoreCell);
    [StoreCellPolyOut, StoreCellDerPolyOut] = BeamFittingDer(StoreCell);
    BeamMatch(StoreCellPolyOut, StoreCellDerPolyOut, StoreCell);
```

identifyBeams →filterdata

- Deleting and rearranging StoreCell for later use

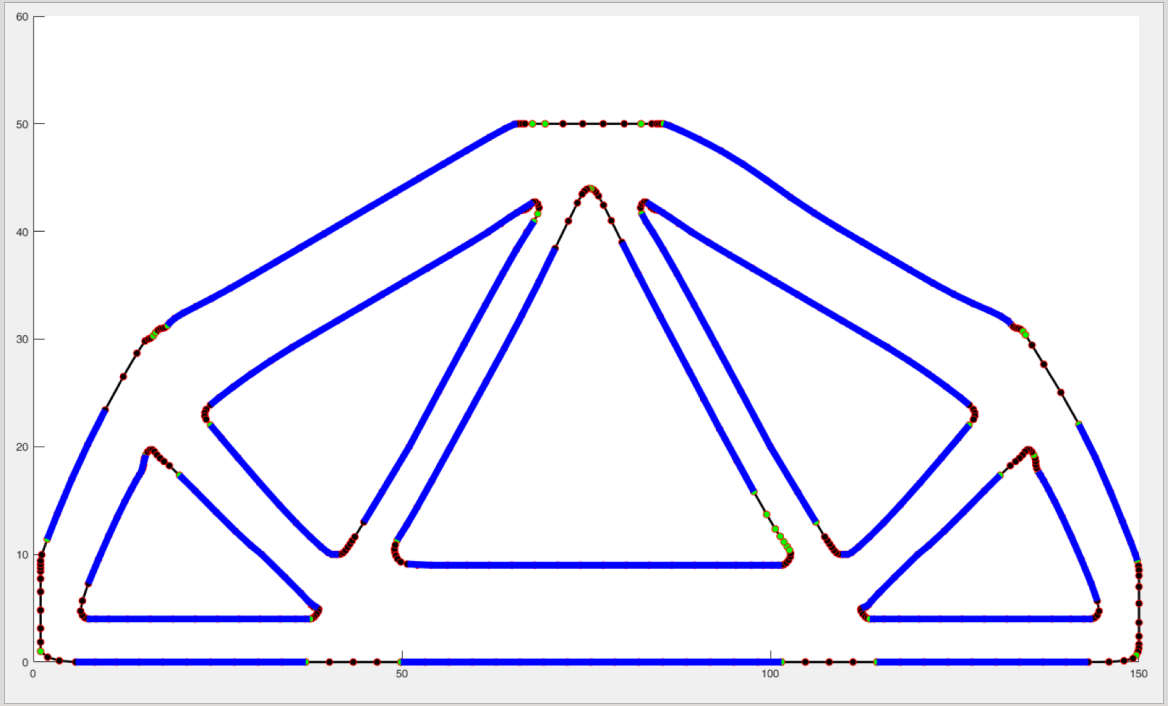
```
function [StoreCell] = filterdata(StoreCell)

%   deleting reoccuring points
    for I = 1:length(StoreCell)
        StoreCell{I} = unique(StoreCell{I},'rows');
    end

%   Deleting non valid lines for the structure
    for B = 1:length(StoreCell)
        if size(StoreCell{B},1) <= 3
            StoreCell{B} = [];
        end
    end
    StoreCell = StoreCell(~cellfun('isempty',StoreCell));

    Temp_Ind = [];
    P = 1;
    for I = 1:length(StoreCell)
        MODE_3 = mode(StoreCell{I}(:,3));
        for U = 1:length(StoreCell{I})
            if StoreCell{I}(U,3) ~= MODE_3
              Temp_Ind(P,1) = U;
              P = P + 1;
            end
        end
        StoreCell{I}(Temp_Ind(1:end),:) = [];
        Temp_Ind = [];
        P = 1;
    end
    StoreCell = StoreCell(~cellfun('isempty',StoreCell));
```

Cont…

- Rearranging and removing the duplicate lines

```
%   Deleting the duplicate line and merging the lines
    StoreCellInlinesTEMP = StoreCell;
    STORE_INDEX_COMP1 = [];
    STORE_INDEX_COMP2 = [];
    P = 1;
    N = 1;
    for U = 1:length(StoreCellInlinesTEMP)
        for Y = 1:length(StoreCellInlinesTEMP)
            if Y ~= U
                if length(StoreCellInlinesTEMP{U}) >= length(StoreCellInlinesTEMP{Y})
                    COMP1 = ismember(StoreCellInlinesTEMP{Y}(:,1:2),StoreCellInlinesTEMP{U}
                    COMP1 = sum(COMP1,2);
                    if mode(COMP1,1) == 2
                        O = Y;
                        STORE_INDEX_COMP1(P,:) = [U,O];
                        P = P + 1;
                    end
                    else
                        COMP2 = ismember(StoreCellInlinesTEMP{U}(:,1:2),StoreCellInlinesTEM
                        COMP2 = sum(COMP2,2);
                    if mode(COMP2,1) == 2
                        O = Y;
                        STORE_INDEX_COMP2(N,:) = [U,O];
                        N = N + 1;
                    end
                end
            end
        end
    end
    STORE_INDEX = [STORE_INDEX_COMP1; STORE_INDEX_COMP2];

    StoreCell_Temp = cell(length(StoreCell),1);
    for L = 1:length(STORE_INDEX)...

    StoreCell = StoreCell_Temp;

    StoreCell = StoreCell(~cellfun('isempty',StoreCell));
    for I = 1:length(StoreCell)       ...
```

Cont…



cont

- Other function calls

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Identifing Inlines for the bionic structure
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [StoreCell] = filterdata(StoreCell);
    [StoreCellPolyOut, StoreCellDerPolyOut] = BeamFittingDer(StoreCell);
    BeamMatch(StoreCellPolyOut, StoreCellDerPolyOut, StoreCell);
```

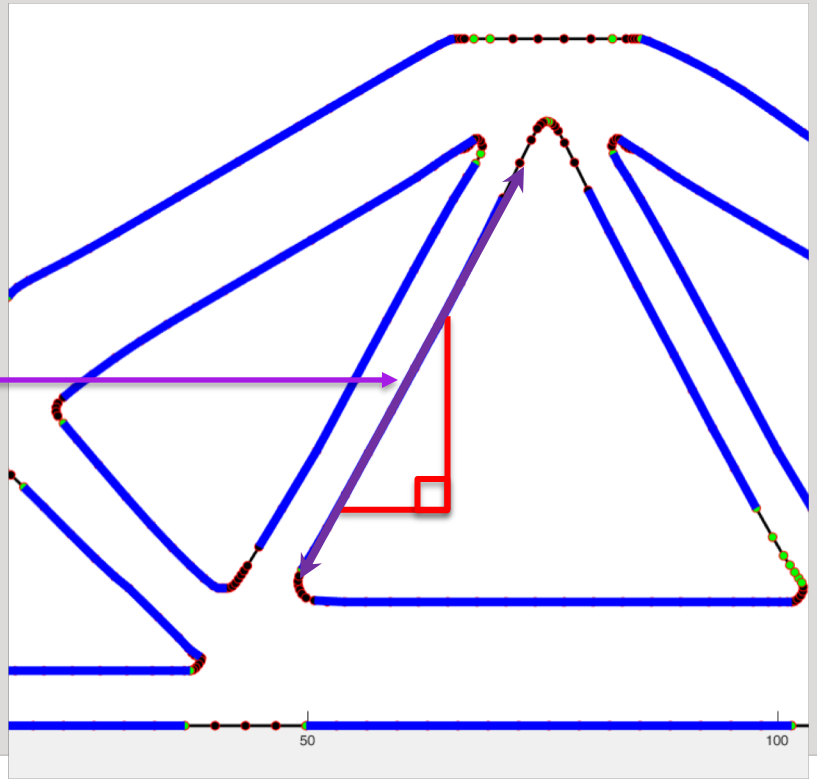
BeamFittingDer

- Getting the derivative for all the lines in the bionic shape

```
function [StoreCellPolyOut, StoreCellDerPolyOut] = BeamFittingDer(StoreCell)
    n = 1000;
    StoreCellPolyOut = cell(n,1);                                   % Cell Storing polyfit
%     PolyIn = cell(n,1);                                  % Cell Storing polyfit data o
    StoreCellDerPolyOut = cell(n,1);                                % Cell Storing polyfit
%     DerPolyIn = cell(n,1);                                % Cell Storing polyfit deriv
%      Identifying the outlines
    w = 1;
    for q = 1:length(StoreCell)        % curve fitting each beam to know beam length and
        x = StoreCell{q}(:,1);
        y = StoreCell{q}(:,2);
        if length(StoreCell{q}(:,1)) ==  1                          %dof = degree of f
            dof = 0;
        else
            dof = 1;                                                %length(StoreCellI
        end
        StoreCellPolyOut{w} = [StoreCellPolyOut{w}, polyfit( x, y, dof)];      %length(S
        StoreCellDerPolyOut{w} = [StoreCellDerPolyOut{w}, polyder(StoreCellPolyOut{w,1})
        w = w + 1;
    end

    StoreCellPolyOut = StoreCellPolyOut(~cellfun('isempty',StoreCellPolyOut));
    StoreCellDerPolyOut = StoreCellDerPolyOut(~cellfun('isempty',StoreCellDerPolyOut));

    clear dof n q StoreCell w x y;

end
```

Cont…

$$f'n = polyfit(StoreCell\{Line\}) = mx + b$$
$$m = slope\ of\ line$$



Cont…

- Other function calls

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Identifing Inlines for the bionic structure
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [StoreCell] = filterdata(StoreCell);
    [StoreCellPolyOut, StoreCellDerPolyOut] = BeamFittingDer(StoreCell);
    BeamMatch(StoreCellPolyOut, StoreCellDerPolyOut, StoreCell);
```

indentifyBeam →BeamMatch

- Matching each to its partner and creating a beam
- Plotting a patch to show the area of the beam

BeamStoreInline

11x2 cell

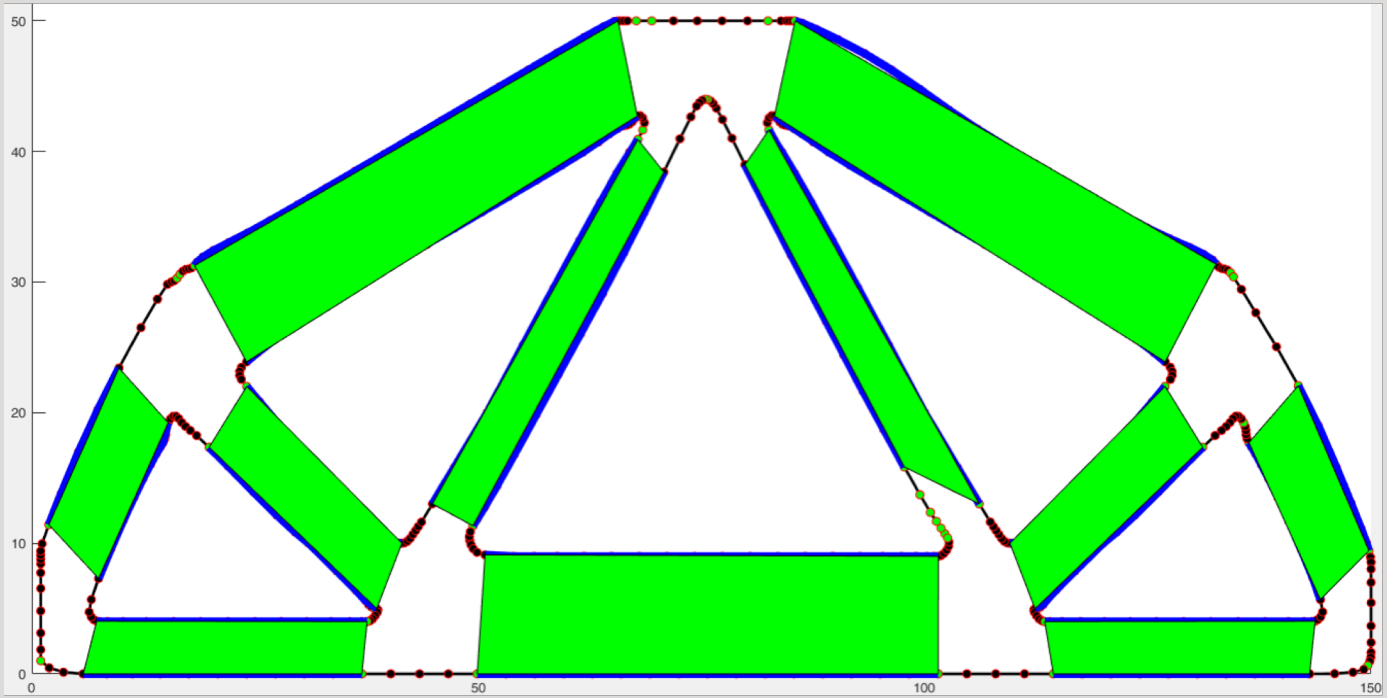
	1	2
1	5x3 double	10x3 double
2	21x3 double	17x3 double
3	20x3 double	18x3 double
4	7x3 double	10x3 double
5	10x3 double	15x3 double
6	17x3 double	19x3 double
7	11x3 double	15x3 double
8	12x3 double	9x3 double
9	13x3 double	10x3 double
10	11x3 double	6x3 double
11	13x3 double	11x3 double

```
%   For All Lines!!

    for p = 1:length(StoreCell)
        if StoreCell{p}(1,1) > StoreCell{p}(end,1)
            StoreCell{p} = flip(StoreCell{p});
        end
    end

    [CenterStoreCell] = CenterPoints(StoreCell);
    count=1;
    while size(StoreCell,1)>1   % Selects 1 Outline to compare with various inline
        A = CenterStoreCell(1,:);
        OuterBeamInline = StoreCell{1};
        StoreCell{1}=[];
        StoreCell = StoreCell(~cellfun('isempty',StoreCell));
        ader=StoreCellDerPolyOut{1};
        StoreCellDerPolyOut{1}=[];
        StoreCellDerPolyOut = StoreCellDerPolyOut(~cellfun('isempty',StoreCellDerP
        CenterStoreCell(1,:)=[];
        B = CenterStoreCell();
        distancesInline = sqrt(sum(bsxfun(@minus, B, A).^2,2));            % compu
        [~,ClosestBeamIndex] = min(distancesInline);                %get index min va

        if abs(ader - (StoreCellDerPolyOut{ClosestBeamIndex})) <= Der_Tol   % check
            InnerBeamInline = StoreCell{ClosestBeamIndex};
            BeamStoreInline{count,1} = OuterBeamInline;
            BeamStoreInline{count,2} = InnerBeamInline;
            PatchCoorXInline(count,:) = [BeamStoreInline{count,1}(1,1), BeamStoreIn
            PatchCoorYInline(count,:) = [BeamStoreInline{count,1}(1,2), BeamStoreIn
            patch(PatchCoorXInline(count,:),PatchCoorYInline(count,:),'green');
            StoreCell{ClosestBeamIndex}=[];
            StoreCell = StoreCell(~cellfun('isempty',StoreCell));
            StoreCellDerPolyOut{ClosestBeamIndex}=[];
            StoreCellDerPolyOut = StoreCellDerPolyOut(~cellfun('isempty',StoreCell
            CenterStoreCell(ClosestBeamIndex,:)=[];
        end
        count=count+1;
    end

end
```

indentifyBeam →BeamMatch → CenterPoints

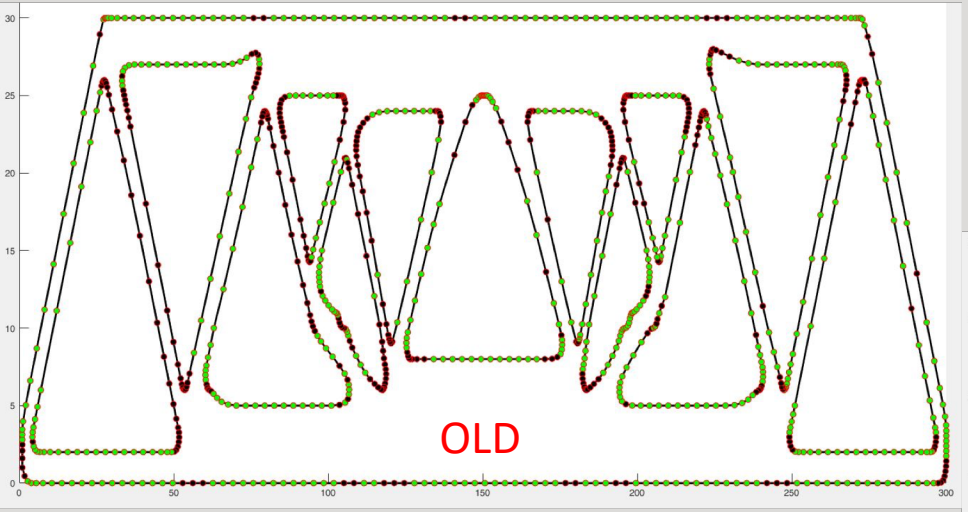
- Finds the center points for all the lines to match the closest line

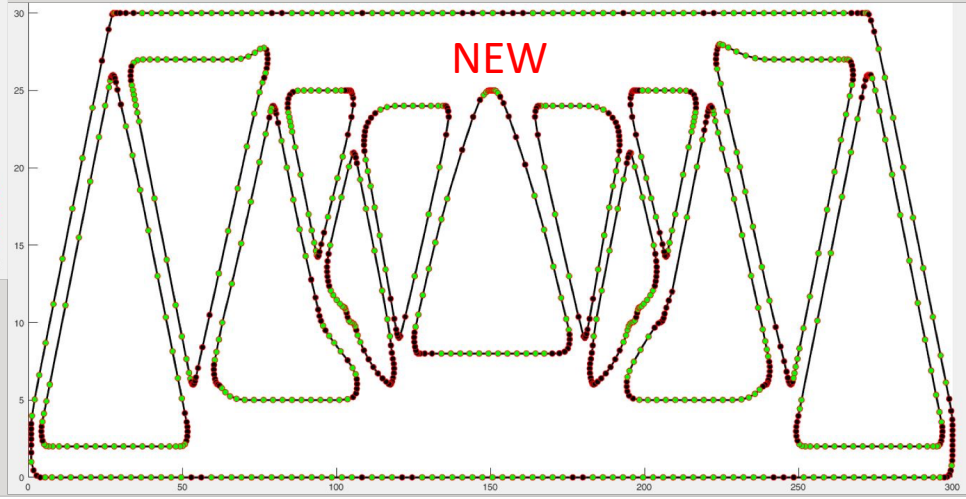
Cont…



Post Processing

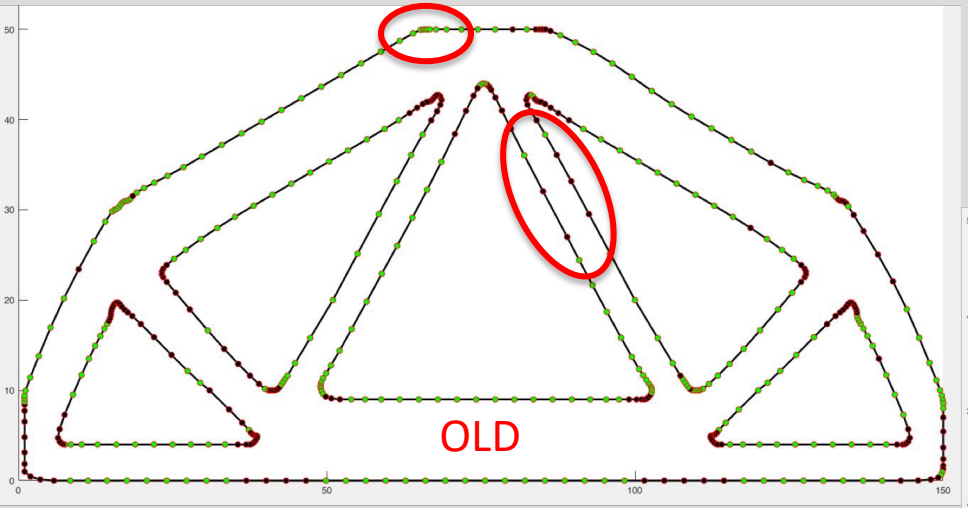
Method 1 vs Method 2 – Points Detection

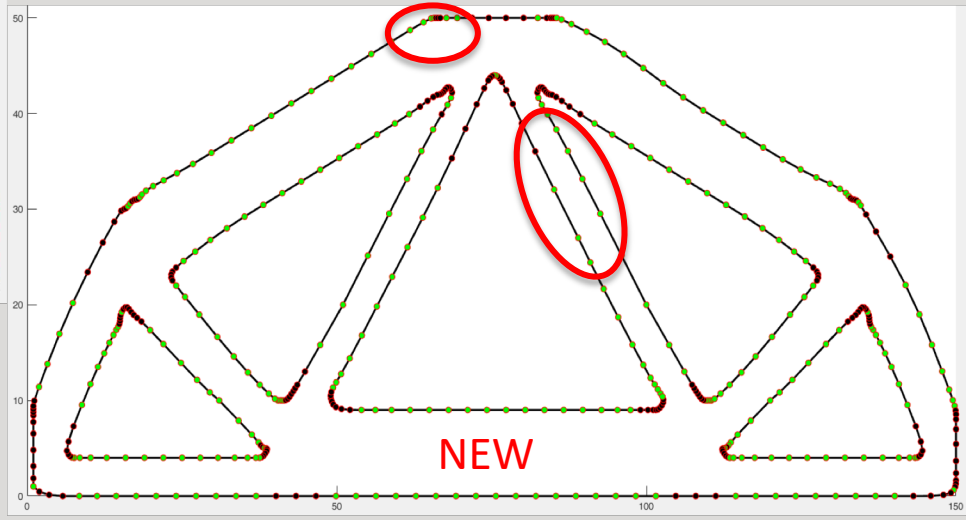




Example 1: Structure 1

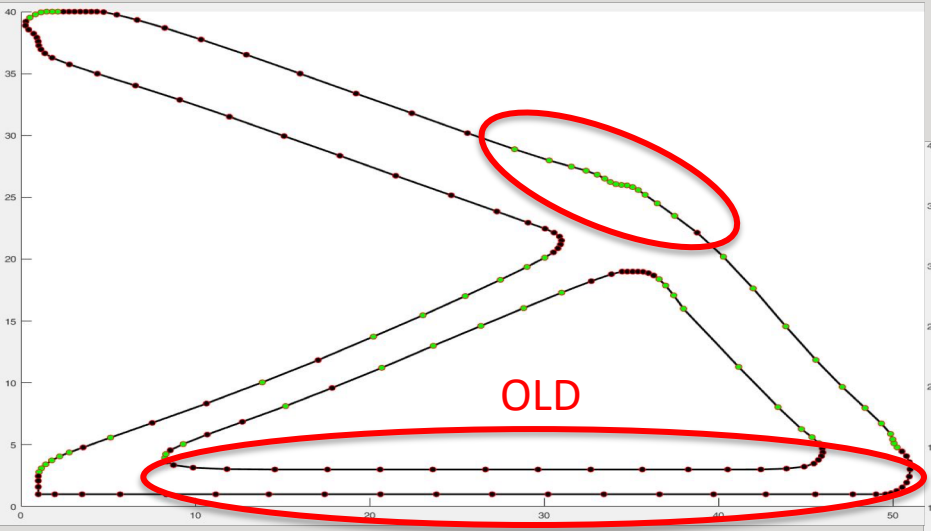
Method 1 vs Method 2 – Points Detection

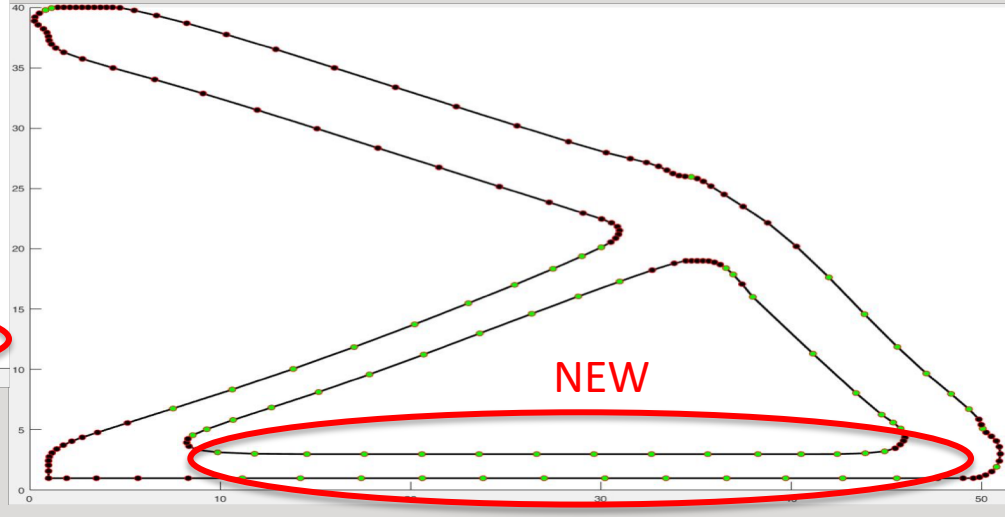




Example 2: Structure 2

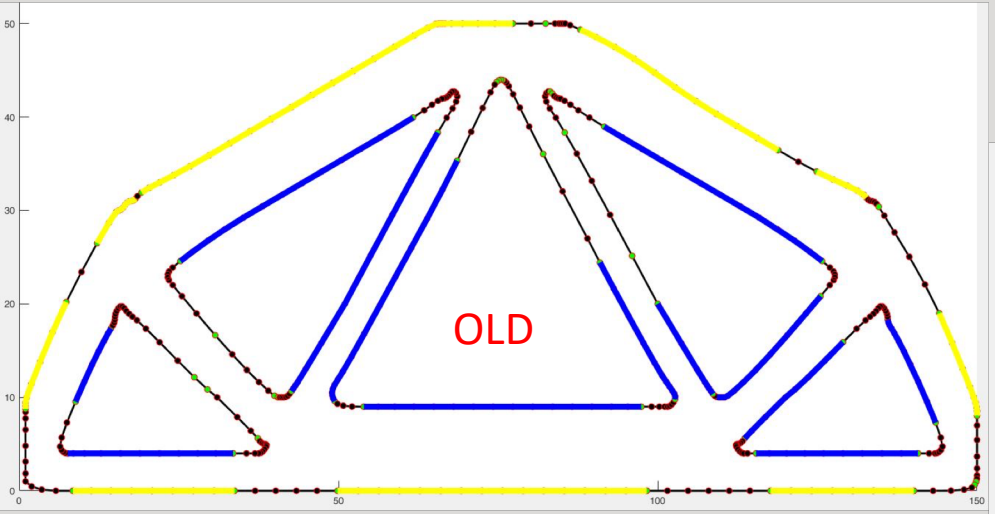
Method 1 vs Method 2 – Points Detection

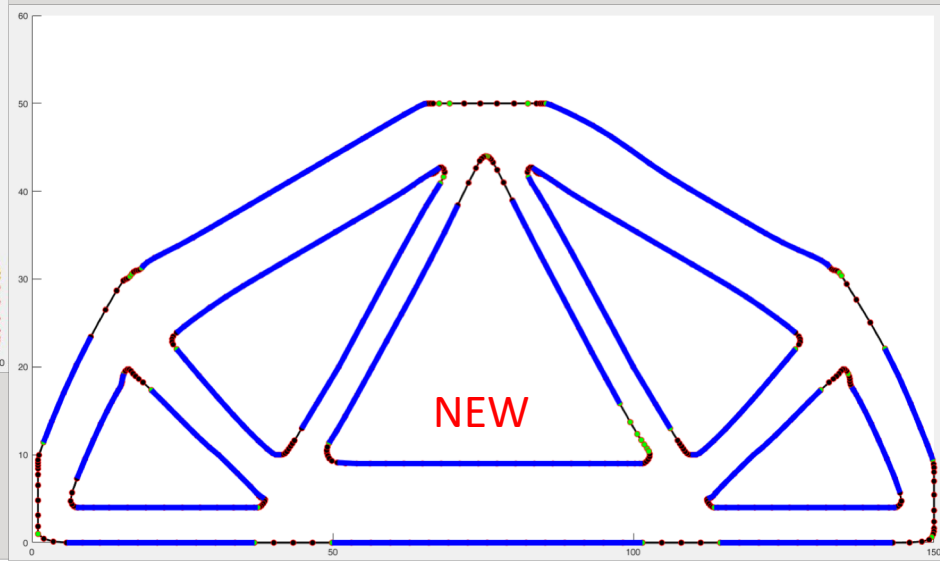




Example 4: Structure 4

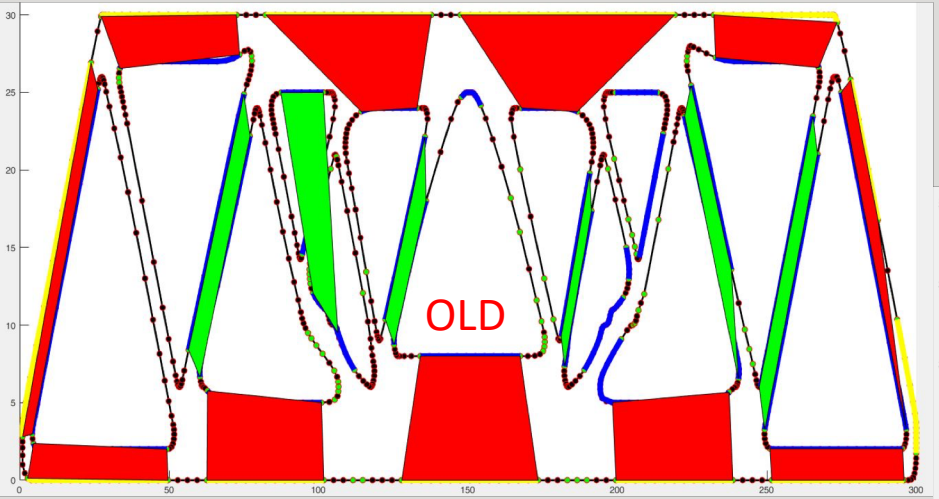
Method 1 vs Method 2 - Lines

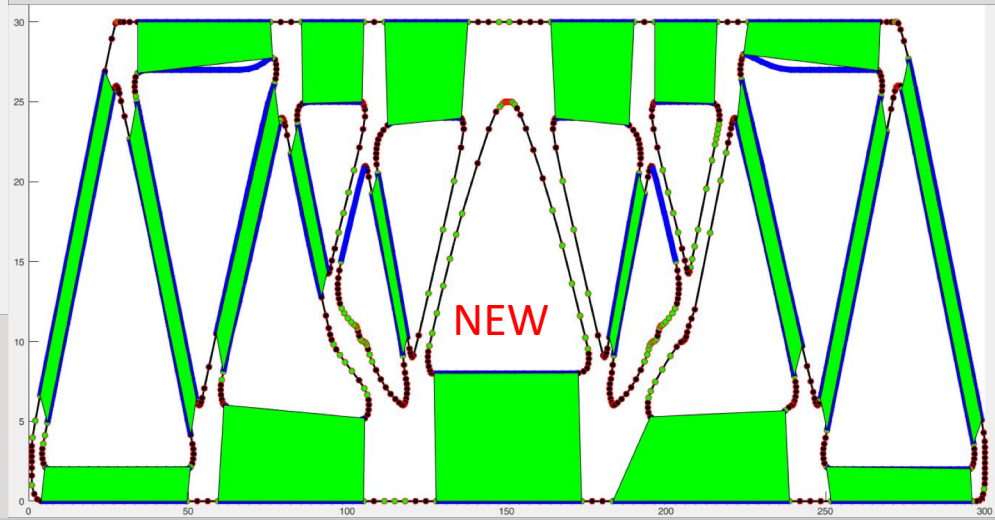




Example 2: Structure 2

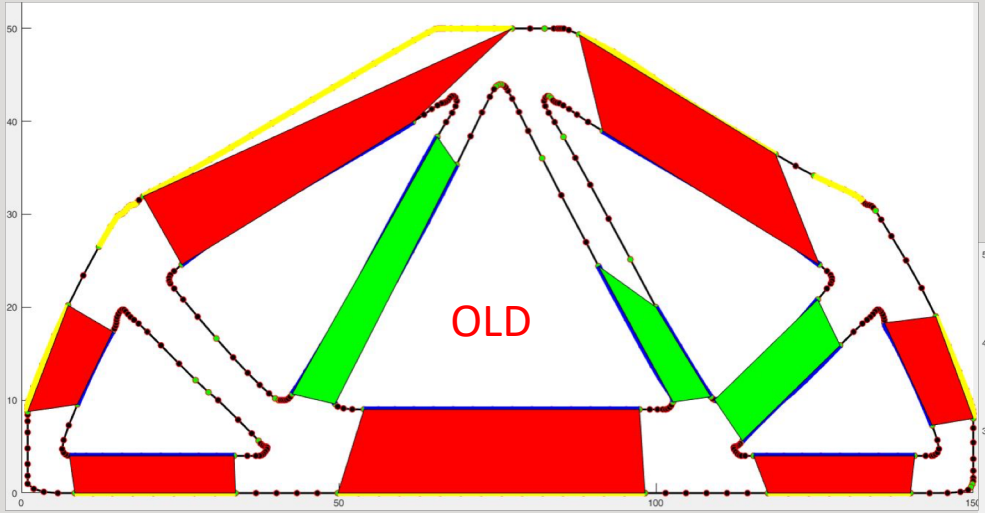
Method 1 vs Method 2 - Patch

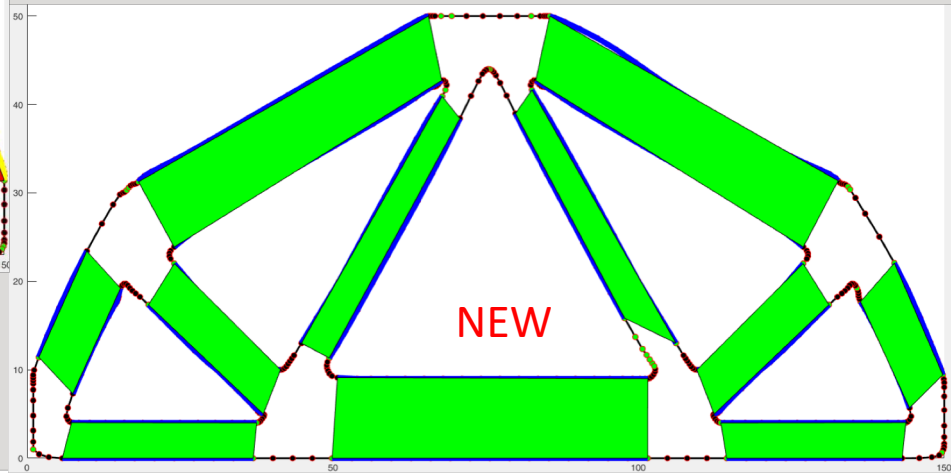




Example 1: Structure 1

Method 1 vs Method 2 - Patch





Example 2: Structure 2

Method 1 vs Method 2 – Run Time

Structure Number	Method 1 (OLD)	Method 2 (NEW)
1	15.912 s	14.568 s
2	6.959 s	5.041 s
3	Does Not Work (DNW)	5.168 s
4	DNW	1.074 s
5	DNW	12.234 s

Conclusion

Conclusion

- The NURBS mesh derivative method yields better results
- To get better results and better runtime
  - Tweaking the polynomial and having a better curve fitting parameter
  - Optimizing the tolerance for each structure

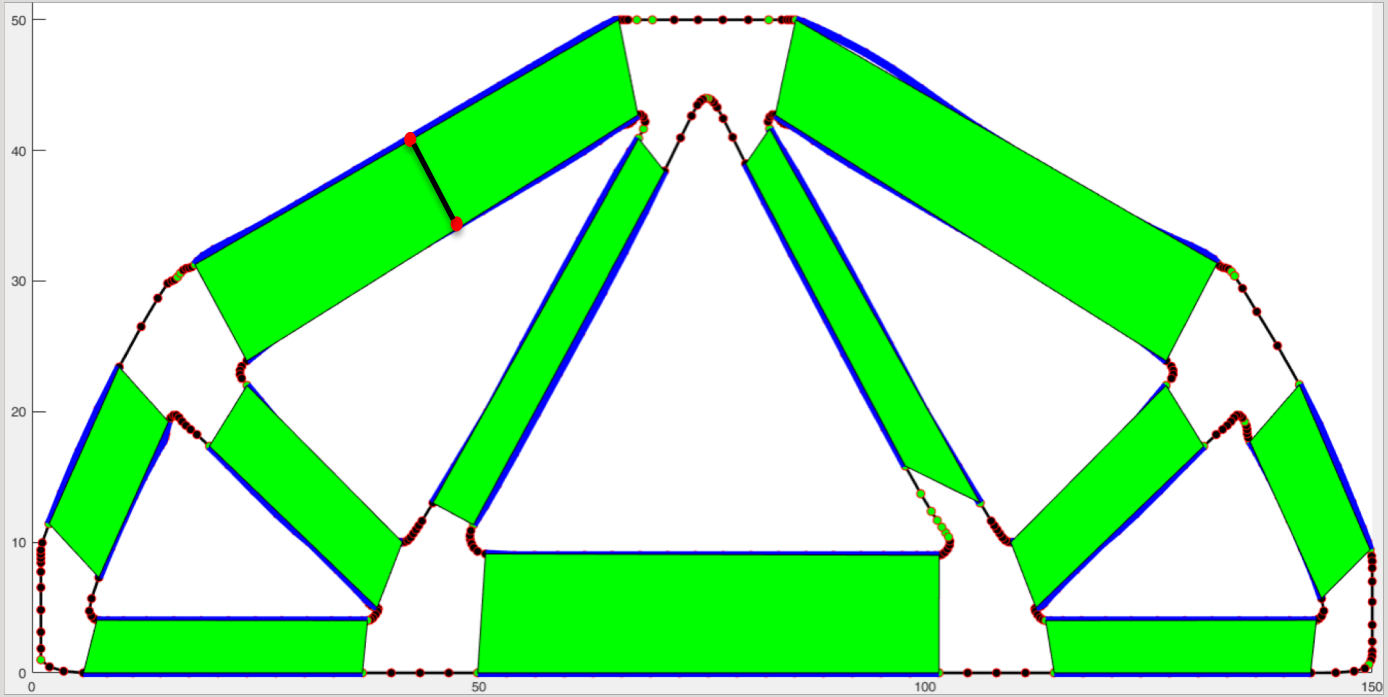
Future Plans

Future Plans

- Look for new storage array for faster process
  - Identifybeam function
- Figure out a new way to compare lines to itself
- New way to compare derivatives and construct polynomial better

- Start simulating the structure on the FEM code
- Start to delete elements
  - Compare the thickness and store elements to delete
  - Ham Code

Element Deletion



BeamStoreInline

11x2 cell

	1	2
1	5x3 double	10x3 double
2	21x3 double	17x3 double
3	20x3 double	18x3 double
4	7x3 double	10x3 double
5	10x3 double	15x3 double
6	17x3 double	19x3 double
7	11x3 double	15x3 double
8	12x3 double	9x3 double
9	13x3 double	10x3 double
10	11x3 double	6x3 double
11	13x3 double	11x3 double

Questions?

References

- Hydrogen Fuel Cells – PEM

  - [1] S. G. Chalk and J. F. Miller, “Key challenges and recent progress in batteries, fuel cells, and hydrogen storage for clean energy systems,” *J. Power Sources*, vol. 159, no. 1 SPEC. ISS., pp. 73–80, 2006.

  - [2] L. M. Das, R. Gulati, and P. K. Gupta, “A comparative evaluation of the performance characteristics of a spark

Visual Aids

[1] F. Felix, “Oil output to hit 500,000 barrels daily.” 2014.

[2] Motertrend, “2013-tesla-model-s-front-1.” 2014.

[3] Electrovelocity, “2011-chevy-volt-front.” 2010.

[4] Newcars, “2012-Nissan-LEAF-Coupe-Hatchback-SV-4dr-Hatchback-Photo-9.” .

Thank You!

Technische Universität Hamburg

UNIVERSITY OF WATERLOO