Automatic Processing and Cross Section Analysis of Topology Optimization Results

C. Gomes Alves, Y. Barthel
(German Aerospace Center, Germany)

Abstract

Finite element-based topology optimizations have become a vital tool in the design of lightweight-optimized components and structures. Used in the early stages of the product development process, they can help identify load paths and shape the fundamental design of a product. However, the interpretation of topology optimization results can be a challenging and time-consuming task. In order to properly analyze the topologies, both knowledge in the field of finite element computations as well as computer-aided engineering and design are necessary. This multi-disciplinary engineering expertise can be expensive and not feasible for small and medium-sized businesses. By automating this vital step in a product development process, the utilization of topology optimizations in all product development processes can be greatly promoted.

We propose a two-fold process for automatically processing density-based topology optimization results for future CAD design proposals: extracting a wireframe and deriving cross sections. The wireframe extraction supports mixed-element models featuring two-dimensional and three-dimensional finite elements and optional mirror symmetry consideration. The cross section extraction is currently available for three-dimensional finite elements.

The wireframe extraction is fundamentally based on voxelization and skeletonization of a finite element- and density-based topology optimization. The cross section extraction makes use of the element density as well as the stress distributions from the topology optimization result. At first, load conditions are identified for each beam of the previously extracted wireframe and an appropriate beam profile is chosen from a set of profiles (currently circle and rectangle, both filled or hollow, and I-beams). Then, using a least square loss optimization and image processing-based shape averaging, the geometric dimensions of each selected beam profile are determined. In the end, a model is available which can be converted into a parametric CAD model for further design work.

1. Introduction

Topology optimizations are becoming more and more present in product development processes as they enable the exploration of lightweight design potential in the early product development stages. As such, they can serve as one of the bases for the product's structural design. However, topology optimization results are inherently open for any structural design method. They can serve as a basis for a sheet metal design or a truss structure featuring beams of various cross sections, a welding-heavy design or one using screws and bolts. In the early design stage, these features may not have been defined yet. This leads to a variety of possible design solutions to be explored.

A topology optimization setup usually starts with a defined volume which is meshed with finite elements (FE) - the available design space. By applying loads and boundary conditions, defining design constraints (such as deformation or stress limits) and an optimization objective (such as minimizing the mass) the optimization algorithm calculates the optimal material distribution in the design space respecting the design constraints. The result can be used to identify load paths and shape the fundamental design of the product. However, properly interpreting the results and converting them into product designs is challenging. There are few and limited solutions available on the market. Existing commercial solutions do not work automatically and are usually designed for a specific manufacturing scenario, such as automotive sheet metal designs (Altair (2021)).

The engineering challenge of deriving a suitable design paired with the sparse availability of supporting algorithms may be one of the main hindrances for smaller businesses to implement topology optimizations into their product development processes. Providing automated (intermediate) steps for interpreting topology optimization results will lead to the reduction of (repetitive and error-prone) manual labor, essentially freeing up resources for other, more creative tasks. Ultimately, this will enable a quicker spread of topology optimizations in various product development processes which, in turn, can result in better lightweight-optimized structures and parts.

In this paper, we propose a process to automatically create truss-like structures with beam sections from density-based topology optimization results. The developed process is based on topology optimization results generated using the solid isotropic material with penalization method (SIMP) which yields density distributions for elements within the design space (see Bendsøe (2004)). In short, an element density-based topology optimization varies the relative element density for each element depending on the stresses and displacements each element is subject to and depending on the optimization objective and constraints. The relative element density is the scalar value of the current density divided by the original (full) density of the material, meaning its value varies between 0 and 1. A value of 1 for an element means that this

element is vital for the load distribution within the design space. Contrarily, an element with a density close to zero does not contribute to the stiffness of the structure. The elements with a high element density carry most of the load and, thus, depict load paths within the design space. By using a density threshold (Iso-value), elements with a lower relative element density than the chosen threshold will be masked from view, making the load paths visible. In case of the presented process elements below the threshold will be ignored from any evaluation.

One way of handling the complexity and freedom in the product development process which uses topology optimizations is to make the derived CAD (Computer Aided Design) model as parametric as possible. For this, a skeleton approach can be used as part of a top-down design approach. The skeleton can be derived from the topology optimization, basically by following the load paths within the design space. In top-down assemblies, one part serves as a geometrical reference for all subsequent parts. Changing dimensions in this part triggers a cascade of changes in the referenced parts. It thus serves as a parametric base for subsequent design steps where the connectivity and spatial information of the skeleton can be altered, e.g. as proposed by Ostrosi (2020). This approach enables the rapid exploration of possible designs and, thus, increase product development speed or the exploration depth.

The element density distribution of a topology optimization often creates a truss-like structure. As such, deriving models based on individual beams forming a truss structure is a viable path to choose. Most beams within such a structure are subject to tension-compression load, though, bending and torsion loads are possible as well. The stress data created by an FE calculation is made up of stress tensors for each element containing normal stresses ($\sigma_x, \sigma_y, \sigma_z$), shear stresses ($\tau_{zx}, \tau_{zy}$) and von Mises stresses. Principal stresses and main stress axes can be calculated from these results.

The proposed process (Figure 1) aims at automatically generating a skeleton and defining cross sections of the corresponding beams. It is comprised of two parts which will be discussed in detail in the following sections 3 and 4: wireframe creation and cross section extraction. The wireframe is the skeletal representation of the topology result. It consists of points and lines, each connecting two points. On its own, it can already be used as a skeleton for a CAD model. With the wireframe, each beam within the truss-like structure of a topology optimization can be identified. In the second part of the process, a cross section is allocated to each beam based on the topology optimization result’s density and stress distribution.

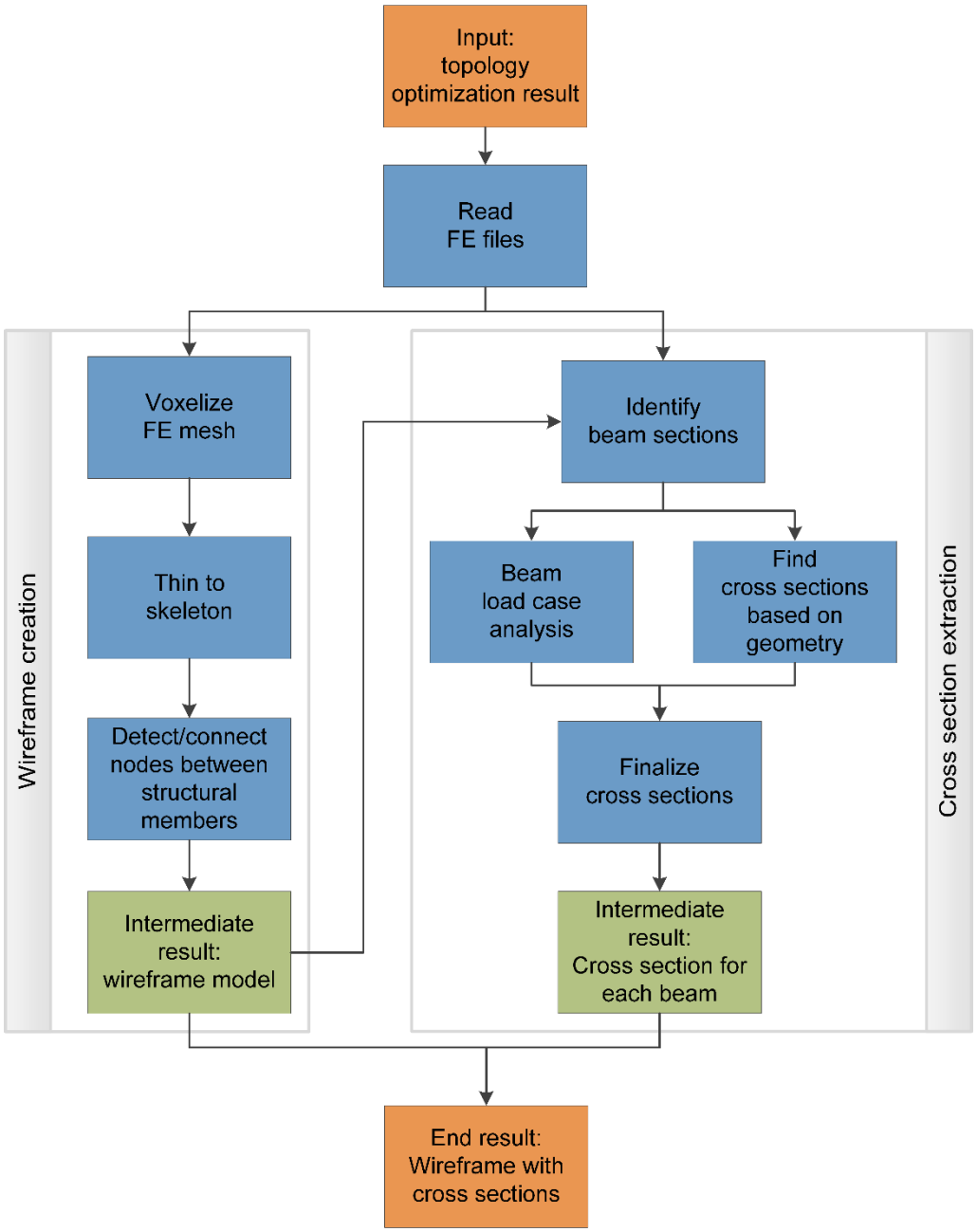

Figure 1: *Overall process flow chart*

In the following section, an overview of similar scientific work dealing with automatic processing of topology optimization results is given. Afterwards, two separate sections discuss the two main process steps, the wireframe extraction and the cross section approximation. They are accompanied by a simple example model. In section 5, accompanying data on this example model as well as a more complex example model are presented and algorithm performance is discussed. Lastly, a conclusion presents the most important aspects and provides an outlook for future developments.

2. Related work

A complete process for reconstructing beam structures from 3D topology optimization results is presented by Nana et al. (2017). They propose a process which directly uses the FE mesh to extract a wireframe. After filtering the topology optimization result, a rough mesh surface shape remains which needs to be smoothed. The curve skeletonization step only uses the FE nodes of the mesh as a point cloud. For each beam of the skeleton, a circular beam cross section is approximated using the average Euclidian distance between the skeleton segment and the point cloud of the FE mesh surface. No other cross sections are supported. Last but not least, they normalize the curve skeleton, straightening bent sections so that uniform circular beam sections remain. Results are validated by performing FE calculations and comparing volumes and compliance values to the initial topology optimization results.

Another process for generating parametric CAD models is proposed by Ramsaier et al. (2019). Here, clustering algorithms are used to identify beam sections and node clusters in topology optimization results via tension/compression stress results. For this, the principal stress direction for each element is examined and the data gathered in a histogram of element stress directions. By identifying peaks in the histogram, individual beams are identified. A limitation is that all cross sections are again assumed to be circles with a constant diameter along the beam axes, though they do not explain how exactly they are approximated.

Other approaches to automate the derivation of geometry from topology optimization results do not use a wireframe or are not based on beam sections and, instead, directly convert the FE mesh into surfaces (e.g. Swierstra et al. (2020), Hsu and Hsu (2005), Koguchi and Kikuchi (2006), Lin and Chao (2000) or Tang and Chang (2001)). However, due to their high dependency on the FE quality many of these approaches have strict requirements for the FE mesh and also the quality of the topology optimization result. Furthermore, some are limited to two-dimensional meshes and, thus, have limited use in many engineering tasks. Lastly, while yielding correct representations of the initial topology optimization result, these approaches usually generate complex surface models which are not well-suited for further processing, e.g. for further manual design work in a CAD program or for manufacturing preparations.

The process for generating the parametric CAD model proposed in this work is based on a voxelization approach, offering easy expansion support for all kinds of finite element types while still providing accurate results. The generated truss-like structures feature individual beam-sections offering advantages with respect to usability in real-life engineering tasks, such as change of dimensions or beam types. Depending on the design challenge and constraints, such as manufacturing constraints, many more beam types can be supported and the method could also be adapted to using extrusion profiles.

3. Wireframe extraction

In this section, a process for creating a wireframe from a topology optimization result is proposed which consists of three steps (see left box ‘wireframe creation’ in Figure 1). The steps will be explained in the following sections.

The wireframe will serve as the basis for a parametric CAD model. Since many topology optimizations use symmetry conditions to reduce the calculation time, it is important to preserve symmetries if applicable and create symmetric wireframes if desired. Currently, individually toggleable symmetries along the three main planes are supported. As a further benefit, the processing time is greatly reduced if symmetry conditions are enabled (almost halved for each active symmetry plane, see run time examples in section 5).

A model of a gantry crane will be used as an example throughout this paper to show the intermediate results for each process step. Figure 2 shows the finite element model setup (left), including applied forces and constraints, and the result of the topology optimization (right), viewed with a relative element density threshold of 50%. The FE model consists of 17,082 nodes and 75,288 tetra elements. The average element size is 50 mm. The bottom of the cranes supports are point-constrained (red) while loads in all three cardinal directions are applied on a non-design space (pink) in the middle of the bridge girder using an RBE2 element. This will create moments along the x- and y-axes in the bridge girder. There are three load cases in total. For the topology optimization, the two gantries form the design space whose compliance is to be minimized while restricting the remaining design space volume to 20%. Symmetry along the xz- and yz-planes is enabled.

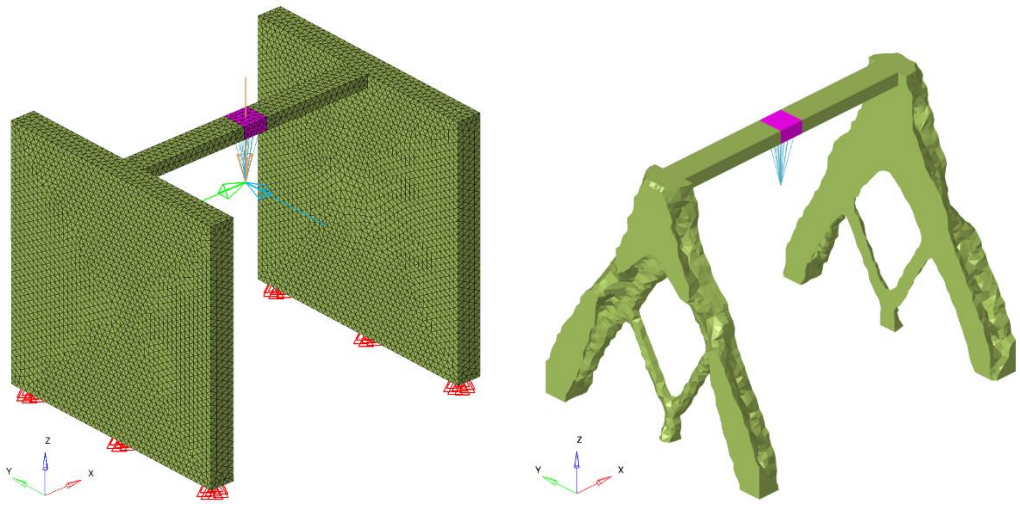

Figure 2: *Gantry crane finite element model (left) and topology optimization (right)*

3.1. Voxelization

In the first step called voxelization, the finite element-based topology model is converted into a voxel representation. A voxel is a cube arranged in an equidistant three-dimensional grid (analogous to a pixel in a 2D image). Each voxel has a binary value stating whether it is active or not, indicating that there is material at this volume in the model space.

In preparation for the voxelization, an element density threshold has to be chosen. Only elements whose density is above this threshold will be converted into the voxel grid. This choice is crucial for the resulting wireframe and beams cross sections. However, this is standard procedure for any interpretation of a density-based topology optimization, be it automatic or manual. Finding the "correct" choice is not a focus of this paper.

This intermediate voxelization step is unusual in comparison to other approaches for extracting a wireframe in the literature. For example, the above-mentioned approaches by Nana et al. (2017) and Ramsaier et al. (2019) both directly generate skeletons from FE meshes. A voxel grid, however, offers several advantages over using the FE mesh. For one, the FE mesh does not need to be smoothed in order to yield good wireframes. Since the finite element sizes and their geometrical form can vary greatly within one model, the smoothing can be a costly process. Another advantage is data efficiency. To define an FE mesh at least two data structures are needed: FE nodes, which are numbered and have three coordinates, and finite elements, which are also numbered and then defined by several nodes (e.g. a hexahedral cuboid element is defined by eight nodes). The voxel grid, on the other side, is stored in a single three-dimensional matrix where each voxel can be addressed via its position which makes processing it very efficient. The corresponding spatial coordinates can be derived by the (configurable) grid distance. The global coordinate system of the voxel matrix is always centered in the voxel matrix volume. If the FE model is not centered along the global coordinate system, an additional displacement vector is generated. It is, however, only needed to correctly align the derived model with the initial FE model and is not used in the calculations to receive a wireframe.

Alternatively, skeleton extractions from FE meshes often only use the outer surface of the mesh and treat the FE mesh as a point cloud, such as by Nana et al. (2017), Ju et al. (2007) or Cao et al. (2015). This can result in very rough skeletons which need heavy pruning or smoothing. Furthermore, this method does not produce good results for two-dimensional finite element meshes forming concave surfaces. Figure 3 (top) shows the topology optimization of the head of a train. It consists of a hull of 2D tria finite elements. The nose at the front forms a tight curvature with some holes in the topology. The skeleton made with the point cloud-based algorithm by Cao et al. (2015) (which is available on github) does not follow the hull well and contracts inwards

(Figure 3 bottom left, compare red lines to orange FE mesh). The voxelized train head (Figure 3 bottom right) follows the contour closely so that the wireframe can be extracted accurately.

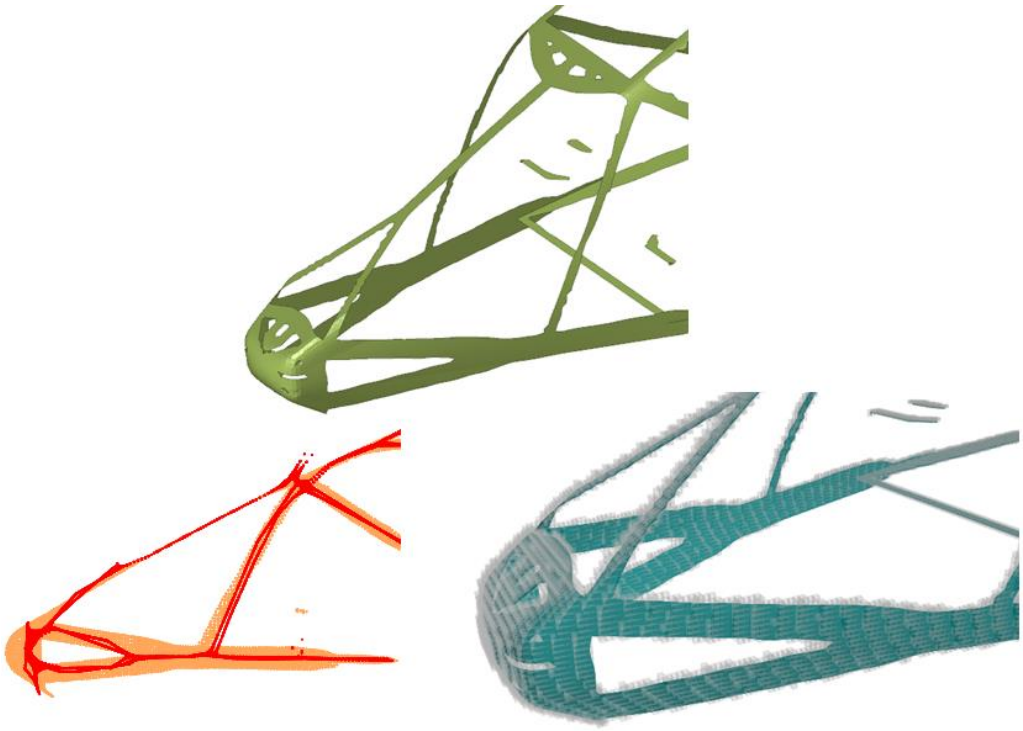

Figure 3: *Comparison of geometric accuracy of FE mesh (top) representations generated by point cloud-based (bottom left) and voxel-based (bottom right) algorithms*

Ultimately, by first voxelizing the FE mesh, an abstract homogenized representation is added to the process, decoupling all further processing steps from the FE mesh. From a programming perspective, this means that support for additional FE solvers and FE types can be implemented easily without affecting the following processing steps. To an extent, the process also becomes independent from the initial FE mesh quality where elements can have varying sizes and deformations which can result in poor performance or results during processing.

The method for converting an FE mesh into a voxel grid uses the method presented in Huang et al. (1998). It has been formulated for 2D elements in a 3D space and consists of three steps: voxelizing the vertices, the edges and the volume of an element. This method is directly applied to all currently supported two-dimensional finite elements (tria and quad).

For currently supported three-dimensional elements (tetrahedrons and hexahedrons), it is first checked if any voxel is contained in the volume enclosed by the finite element. Additionally, the above-mentioned method by

Huang et al. (1998) is used as a fall back if volume-voxelization fails. In Figure 4, the voxelized gantry crane is depicted in gray (compare to topology optimization from Figure 2, right). The voxel grid size is the same as the average FE size, 50 mm. Black voxels are part of the skeleton which will be explained in the next subsection.

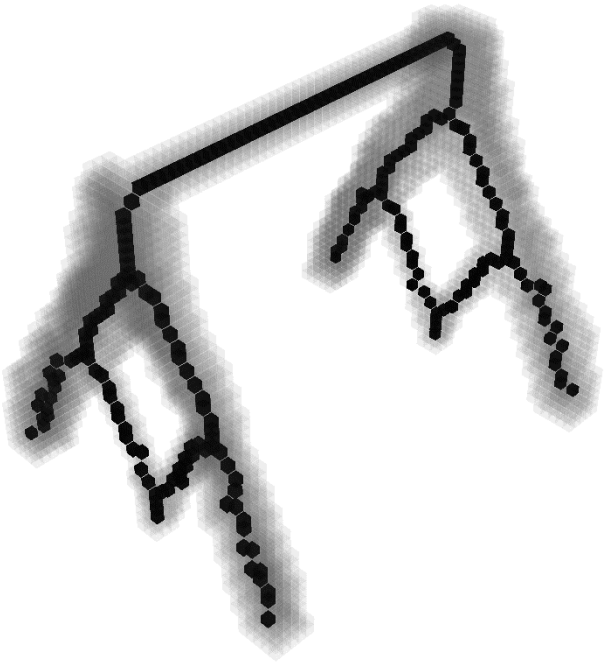

Figure 4: *Voxelized gantry crane topology optimization (gray) and the one-voxel thick voxel skeleton (black)*

3.2. Thinning

The voxelized model is a direct representation of the initial topology optimization which still represents a material distribution within the model space. In order to identify the model's topology (i.e. which parts of the model are interconnected and how), a skeleton is extracted. This skeleton will only feature one-voxel thick branches, essentially omitting any information of material distribution. The voxel skeleton is highlighted with dark voxels in Figure 4.

Here, another advantage of using a voxel grid for the wireframe extraction becomes apparent. Due to its ordered nature, a voxel grid can be processed by common image processing algorithms. These are usually very fast even for large models and some are available in standard code libraries, such as the thinning algorithm by Lee et al. (1994), which is implemented in pythons scikit-image library (see van der Walt et al. (2014)). It is currently being used to thin the voxel model to one-voxel thick branches. The algorithm examines

the 3x3x3-neighborhood of each voxel (i.e. a box of 27 voxels) and iteratively deactivates voxels until there is only one-voxel wide branches left.

A disadvantage of the used algorithm is that it does not guarantee symmetric skeletons even if the input data is perfectly symmetric. Since support for symmetry planes is desired, the resulting skeleton is trimmed and mirrored according to the active symmetry constraints.

3.3. Identify and connect junction voxels

The wireframe model is supposed to consist of points and lines. The points can be extracted from the skeletonized voxel model. They are exactly those voxels where three or more beams meet. To identify these junction voxels, an algorithm presented in Klette (2006) is used which divides the voxel skeleton into branches and clusters at junctions where three or more branches meet. The method was initially developed to detect brain cells from two-dimensional density images. In our proposed process, it is used to identify junction voxels. A junction cluster (gray voxels in Figure 5) consists of several voxels at the end of branches which are in the vicinity of a junction. The voxel closest to the centroid of the junction cluster is declared as the junction voxel (green voxel in Figure 5) and will later serve as a point in the wireframe.

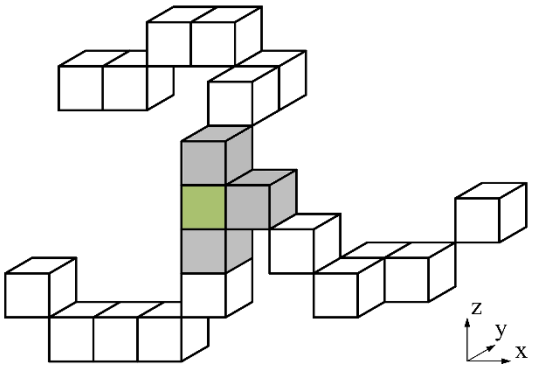

Figure 5: *Three branches meeting at a junction cluster of four voxels (gray + green) and the junction voxel (green)*

This step also respects symmetry constraints if symmetry planes are activated. For example, this entails that, for junction clusters touching a symmetry plane, a voxel at the symmetry plane will be chosen as the junction voxel.

Having identified junction voxels, i.e. points of the future wireframe, they also need to be connected with each other via lines. This is done using a flood filling algorithm. Starting from a junction voxel, the algorithm walks along the voxel skeleton voxel by voxel until it hits another junction voxel. The starting and the end voxel are then flagged as connected with each other. This is done

until all junction voxels, skeleton voxels and branches have been touched at least once. The result is a list of interconnected junction voxels.

However, this cannot yet be used as a wireframe because each connection is a single straight line from junction voxel to junction voxel. Beams in a topology optimization are often not straight but can curve freely through the 3D space. To account for this, the distance is calculated between each skeleton voxel and the straight-line connection between the junction voxels of the branch the skeleton voxel is part of. If the distance exceeds a certain threshold, the skeleton voxel is turned into a junction voxel, dividing the branch into two branches. This process is repeated until the distance for all skeleton voxels is below the threshold, ensuring that the line connections follow the voxel skeleton, and thus, the topology optimization closely. This parameter needs to be set with caution as geometrical accuracy (low threshold) can quickly lead to a high number of segmented beams, effectively over-separating the wireframe model, making it less usable for the following cross section extraction and real-life engineering application. The result is the final wireframe model (Figure 6).

Figure 6: *Wireframe (over voxel skeleton in light gray) of the gantry crane model*

4. Cross section extraction

In this section, the box ‘cross section extraction’ from Figure 1 will be presented. To explain the approach, the two process steps ‘beam load case analysis’ and ‘find cross sections based on geometry’ are broken down into more detailed sub-processes (see gray boxes in Figure 7). They will be discussed in detail in the following sections.

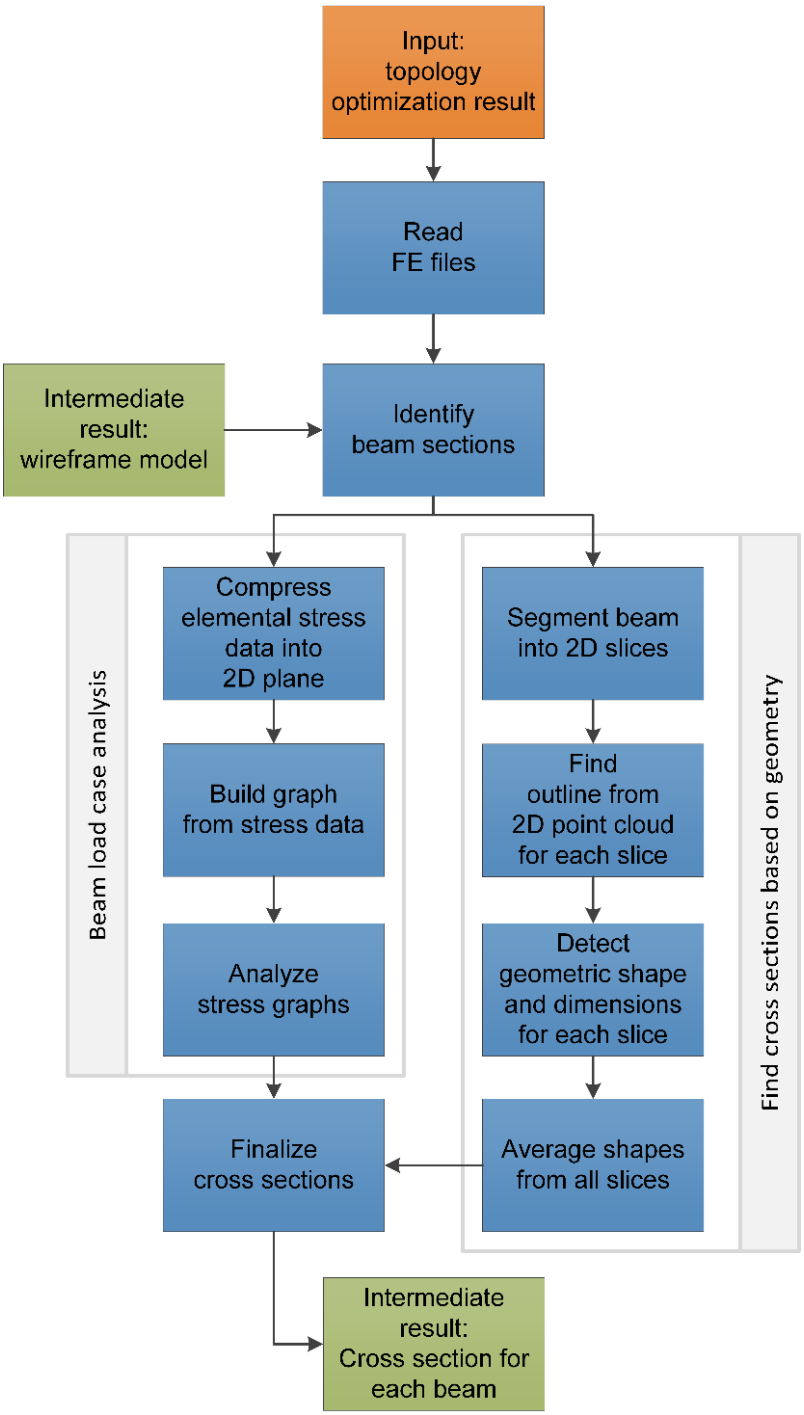

Figure 7: *Detailed flow chart of cross section extraction*

4.1. Beam load case analysis

For beam-like structures, three types of basic loads can be differentiated: tension/compression, bending and torsion. Each load is characterized by a unique stress pattern inside the material and requires a different type of cross

section. For compression and tension, the mechanical stress is evenly distributed across the whole cross section and the overall profile area matters most to handle the loads. The shape of the cross section becomes more important in case of bending load. To counteract large stress values along the outer faces of the beam, profiles with more material at the edges are usually chosen. Profiles like a standard EN 10365 I-beam have significantly lower bending stress and deflection than a circular cross section with the same profile area under similar loading conditions. For torsion, hollow circular profiles are preferred, and it is crucial to avoid any kind of cross sections with openings such as C-Profiles or I-beams due to their low torsional stiffness.

Finding a good profile shape has also been investigated by Larsen and Jensen (2009). However, their method is limited to a CAD-tool which assists a user by matching predefined shapes with marked geometry. The user needs to select the axis and highlight an area of the topology optimization and the procedure matches a primitive shape by comparing the polar coordinate plots extreme values. For example, a triangle has 3 peaks so any geometry with a polar plot with 3 peaks at the right distance is classified as such. This method requires too much user interaction to be useful for the desired automated process.

We present a novel method to extract information about the type of load on beam-like structures created by topology optimizations. Based on the observations on single beams with varying loads, a strategy to analyze the stress patterns of the topology results was developed. Only three components of the stress tensor are found to be relevant for analysis: the normal stress σ_z along the beam axis (which will be called the z-axis) and the two perpendicular shear stresses τ_{zx} and τ_{zy}. The element stress tensors are transformed into the local coordinate system of the beam where the z-axis is oriented along the beam axis.

Tension/compression is characterized as a mostly constant normal stress along the beam axis and no shear stresses. Beams under bending loads show a normal stress distribution with layers parallel to the beam length. One side of the beam is stretched while the opposing side is compressed. In the middle, a layer without material stress can be observed, called the neutral filament. Viewed in the local xy-plane, this distribution forms a characteristic gradient as seen in Figure 8. Torsion produces no normal stresses but a layered distribution of shear stress.

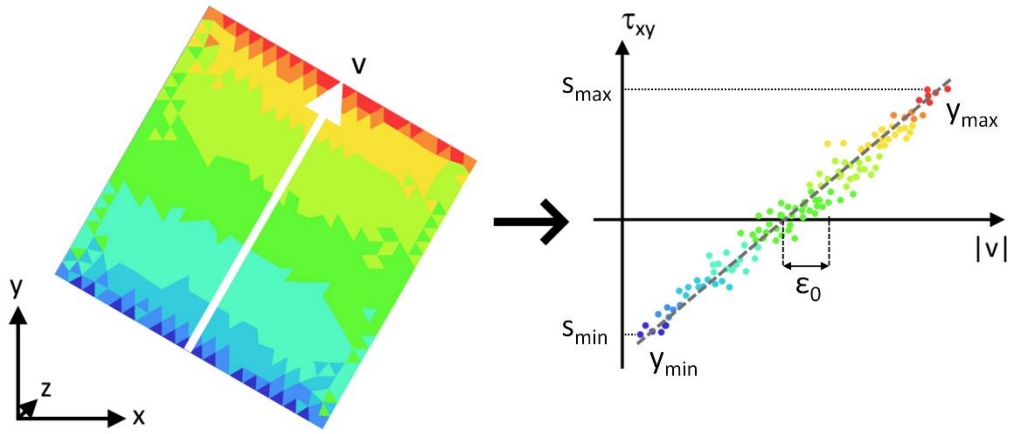

Figure 8: *Rectangular example beam under bending load transformed to a stress distribution graph y.*

The first step of the automated cross section detection is to assign each finite element to a beam of the previously extracted wireframe. This is done by calculating the Euclidian distance from an elements center of mass to the axis of each beam. The element is assigned to beam it has the smallest distance to. Then, for each beam of the wireframe model, the stress components are converted to a graph representation. All finite element stresses of a beam are projected into the beams local xy-plane and then sorted along a given direction vector v (see Figure 8). The stress values of the current component are graphed over their position along v so that the gradients become visible. To find the best graph representation for arbitrarily rotated beams, v vectors with different orientations are systematically tested. It is sufficient to consider orientation from 0 to π. For each graph, a linear function is fitted using the least squares regression and the orientation of v with the lowest regression error is chosen. Next, the fitted line can be analyzed for specific properties:

- linearity of the graph: compare the minimum and maximum of the stress component (s_{min}, s_{max}) to extreme values of the graph (y_{min}, y_{max}) and calculate a normed value $\varepsilon_l = \frac{|s_{min} - s_{max}|}{|y_{min} - y_{max}|}$. A value near 1 indicated a linear distribution, while 0 means it is most likely constant.
- x-intercept: ϵ_0 is 0 if the graph does not cross the $|v|$-axis, ϵ_0 is closer to 1 the closer the intercept is to the middle of the axis.
- relative component importance: the length $|s^{\alpha}_{min} - s^{\alpha}_{max}|$, $(\alpha \in \{\sigma_z, \tau_{zx}, \tau_{zy}\})$ between both stress extreme values is compared to both other components to calculate a value $\epsilon_c = \frac{|s^{\alpha}_{min} - s^{\alpha}_{max}|}{\max_{\alpha}|s^{\alpha}_{min} - s^{\alpha}_{max}|}$. It reflects how important this stress component is relative to the others.

To calculate a score for every load case, the relevant metrics for every stress component are added together. Their relevance is determined from the expected graph shape for a particular case. For example, for bending:

$$(1) \quad \varepsilon_{bending} = \frac{1}{5}(\varepsilon_l^{\sigma_z} + \varepsilon_0^{\sigma_z} + \varepsilon_c^{\sigma_z} + (1 - \varepsilon_c^{\tau_{xz}}) + (1 - \varepsilon_c^{\tau_{yz}}))$$

Properties can either be added normally or subtracted from 1, depending on if they are an indicator for or against the scored load case. Table 1 provides the rules for the scores of all three load cases.

Table 1: How to add the stress graph metrics to calculate each load case score

Compression/Tension

	ϵ_l	ϵ_0	ϵ_c
σ_z	-	-	+
τ_{xz}	0	0	-
τ_{yz}	0	0	-

Bending

	ϵ_l	ϵ_0	ϵ_c
σ_z	+	+	+
τ_{xz}	0	0	-
τ_{yz}	0	0	-

Torsion

	ϵ_l	ϵ_0	ϵ_c
σ_z	0	0	-
τ_{xz}	+	+	+
τ_{yz}	+	+	+

The result is normalized to values between 0 and 1 which indicates the confidence in the load case by the automated process. The highest score indicates the right load case for the beam.

4.2. Cross section analysis

The next step is the geometric analysis of the topology optimization results to find the right dimensions and shapes of the beam profiles. For further processing, all finite elements are converted to a single point representation by collapsing their nodes (e.g. four tetra element corners) into one average point. The single point retains all information like density, volume and stress tensor and can be processed with methods developed for point clouds.

Each beam is cut into segments along its axis, which is given by the extracted wireframe (axis z in Figure 9). The segments are flattened into two-dimensional slices by removing the coordinate along the beam axis. For each of these 2D slices, a geometric shape, which represents the beam geometry in this area, is found. The segmentation is necessary because it is possible that the wireframe axis is angled towards the real beam axis. This could lead to distorted 2D slices and geometrically inaccurate shapes being detected. Hence, segmenting a beam ensures a geometrically accurate shape detection. Each segment has a small overlap with the next to ensure smooth transitions. The detected shapes will be averaged to one shape for each beam afterwards.

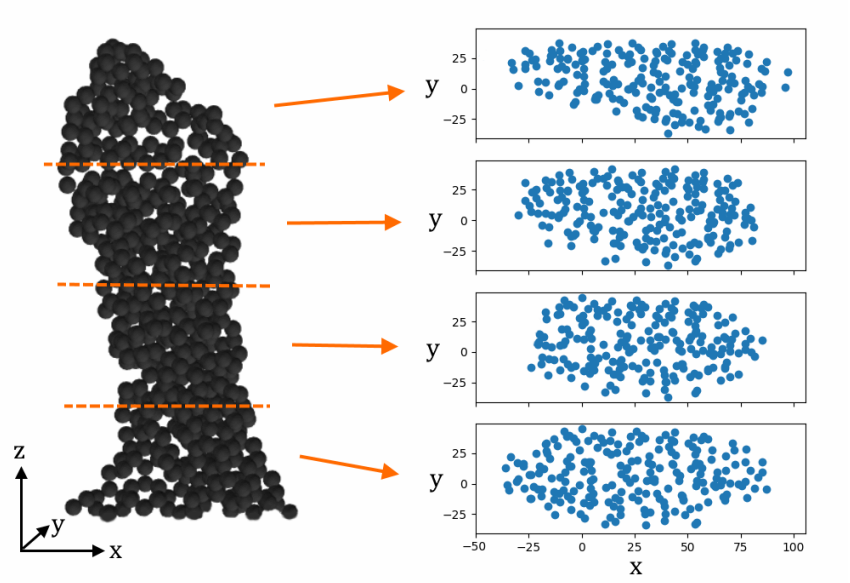

Figure 9: *Beam divided into four segments and 2D-flattened points of each segment*

For the shape detection of a 2D slice, the outline or hull points are extracted. This can be done using a convex hull algorithm. However, for all geometries of non-convex nature this leads to unsatisfactory hull detection (Figure 10, left). Therefore, the χ-shape algorithm is used as proposed by Duckham et al. (2008). It can calculate hull points for any given 2D point cloud and needs a single parameter in the range of 0 to 1. Figure 10 shows a comparison between a convex hull algorithm and the χ-shape algorithm with different parameter values for χ. The smaller the parameter χ is chosen, the closer the hull matches the point cloud outline.

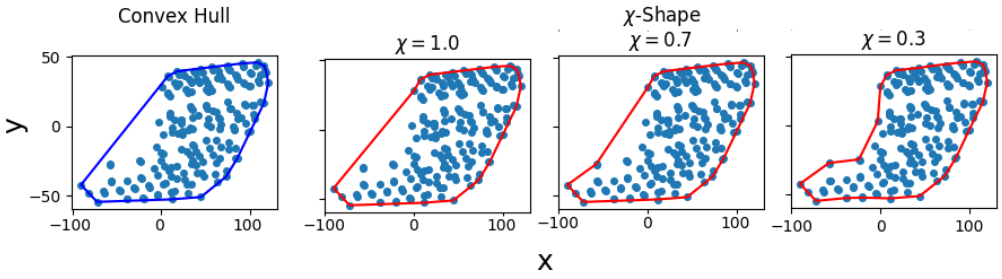

Figure 10: *Segment processed with the convex hull and Chi-Shape-algorithm*

For real-world applications, only certain geometric shapes are typically used as cross sections for beams. In this paper, the focus lies on rectangular and circular profiles as well as I-beams. The proposed workflow supports many different shapes which can be added with little effort. The supported shapes now need to be detected and fitted to the previously extracted outline points. Multiple methods were considered for this step, for example Hough-

Transformation on circles and extended to rectangles, as proposed by Jung et al. (2004). However, a different approach achieved better results: For each shape, a distance metric is employed which calculates the length from a given point to the shape border. Over several iterations, a least square loss function is minimized in order to find the optimal parameter vector for any shape. The geometric shape with the lowest least squares error is chosen to fit the segment.

At this stage, since one beam was divided into several segments, each beam has a number of shapes which can deviate from each other in small ways. The detected shapes need to be unified into one common profile. Shape averaging is a topic with only very few relevant contributions, mostly covering biology related shapes. The easiest form of shape average is the simple average of all shape parameter values. For circles, this is easy to compute and gives good results. However, rectangles are defined with rotation angles which cannot be averaged without problematic edge cases, e.g. two rectangles rotated by 180°. Furthermore, this simple average cannot combine two different types of shapes.

A more flexible approach was designed utilizing image processing. It can robustly find averages for all shapes. The first step is to generate an image buffer large enough for all shapes of a single beam. The buffer is a gray value image with a range of 255 values per pixel and can be controlled via a resolution parameter. All n shapes are additively drawn into the buffer with a value of $255/n$ so that a white region appears where all segment shapes overlap, as seen in Figure 11 (left). The image is blurred with a simple gauss filter to avoid hard edges (Figure 11, center).

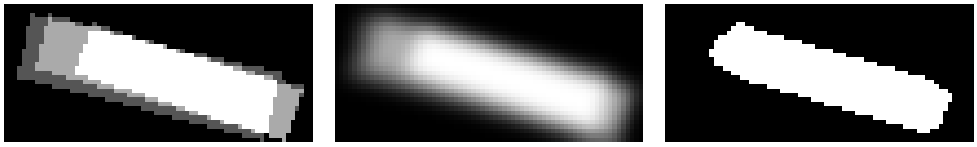

Figure 11: *Rectangular shapes averaged using the image processing method*

Next, a threshold value has to be calculated to extract the correct shape from the buffer. The maximum dimensions of every input shape are evaluated and averaged. For a circle, this would be the diameter; for a rectangle, the diagonal and its perpendicular length. All image pixels above the threshold are set to 255 and all below to 0, resulting in a shape with sharp edges, as seen in Figure 11 (right). The resulting white shape in the image buffer can now be analyzed to find its maximum dimensions. From the center point of all pixels, two perpendicular rays are cast, measuring the length to the shape border. This process is repeated in the possible range of all directions and the maximum dimensions of all rays cast is compared to the previously calculated average dimensions. If the average is lower, the threshold is increased and vice versa. This iteratively narrows down a threshold value where the difference in dimensions is minimized. Now the shape edges are extracted using a canny

edge detector. Knowing the image resolution, all remaining white pixels can be transformed back to world positions. This outline is the average outline of all segments of a beam. Now, the shape of this outline can be determined by reusing the previously described least square loss optimization and, ultimately, receive the overall average shape of the beam.

4.3. Finalize cross sections

In the previous steps, all beams were analyzed to find their respective load case (tension/compression, bending, torsion) and their average beam shape and dimensions. The information from these two steps need to be consolidated to receive a cross section proposal for each beam.

Beams under tension/compression and torsion loads are ultimately treated the same. The profile shape and dimensions are taken from the detected shape from the geometric topology information (e.g. circular or rectangular). Finally, the shapes are hollowed out to match the beams mass to the mass of the beam-like structure in the topology optimization result (considering the relative element densities from the topology optimization). Beams under bending load however are only allowed to fit rectangular shapes which are ultimately converted to standard EN 10365 I-beams. The resulting cross sections for the gantry crane model are shown in Figure 12.

So far, this method only supports three-dimensional finite elements. In future, it is planned to also take two-dimensional finite elements into account to generate cross sections for fully mixed-mesh topology models.

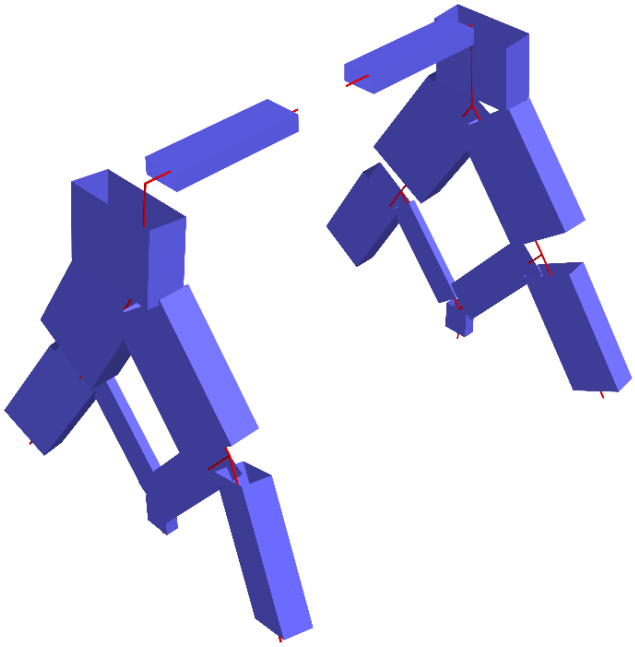

Figure 12: *Gantry crane wireframe with identified cross sections*

5. Exemplary application

During discussion of the proposed process in sections 3 and 4, a small-scale gantry crane model served as an example model to demonstrate the individual process steps (see all steps in figures 2, 4, 6 and 12). In this section, statistical information such as element counts and algorithm run times will be provided. Furthermore, an additional large-scale model of a train car body will be introduced.

All calculations were performed on a desktop computer featuring an Intel Xeon E5-1660 processor with a 3,2 GHz base clock speed. The algorithm runs as single-core. Table 2 provides the statistics on the gantry crane model as well as the car body, which will be introduced afterwards. For all runs, symmetry along the xz- and yz-plane were activated. Run times are median values of five runs, whereas multiple runs show consistent run times. Understandably, the voxelization takes the most time since in this step the - by far - largest amount of data is processed (each finite element needs to be touched at least once).

Table 2: Quantified facts and results for gantry crane and train car body

Model	**Gantry crane**	**Train car body**
Dimensions (x/y/z)	2.4 x 2 x 2 m	20 x 2.94 x 4.6 m
Element types	3D (tetra)	mixed 2D/3D (tria, quad, penta, hexa)
Number of FE nodes	17,082	260,890
Number of finite elements	75,288	240,164
Average mesh size	50 mm	60 mm
Element density threshold	0.5	0.2
Size of voxel grid element	50 mm	60 mm
Number of active voxels	6,283	34,312
Number of skeleton voxels	239	4,561
Number of wireframe nodes	17	249
Number of wireframe connections	18	296
Run time wireframe extraction	50 s	147 s
…of which is voxelization	48 s	129 s
Run time cross section extraction	14 s	11 s

The second application example is a train card body which is modeled in real-life size (Figure 13, top). The undercarriage (green) of the wagon is modeled with 3D elements; the sides and roof (blue), fronts (purple) and the intermediate floor (orange) are modeled hulls using 2D elements. All elements are part of the design space. The topology optimization setup is complex, involving 15 load steps and eight optimization constraints limiting displacements at strategic points. The optimization objective is to minimize the overall mass. The topology optimization result in Figure 13 (center) is filtered using an element density of 0.2. The extracted wireframe (overlaid over the

voxel skeleton) shown in Figure 13 (bottom) can already serve as a useful base for a parametric CAD model. However, it is also noteworthy that the process may be further improved in future work, specifically to increase the result quality for larger models.

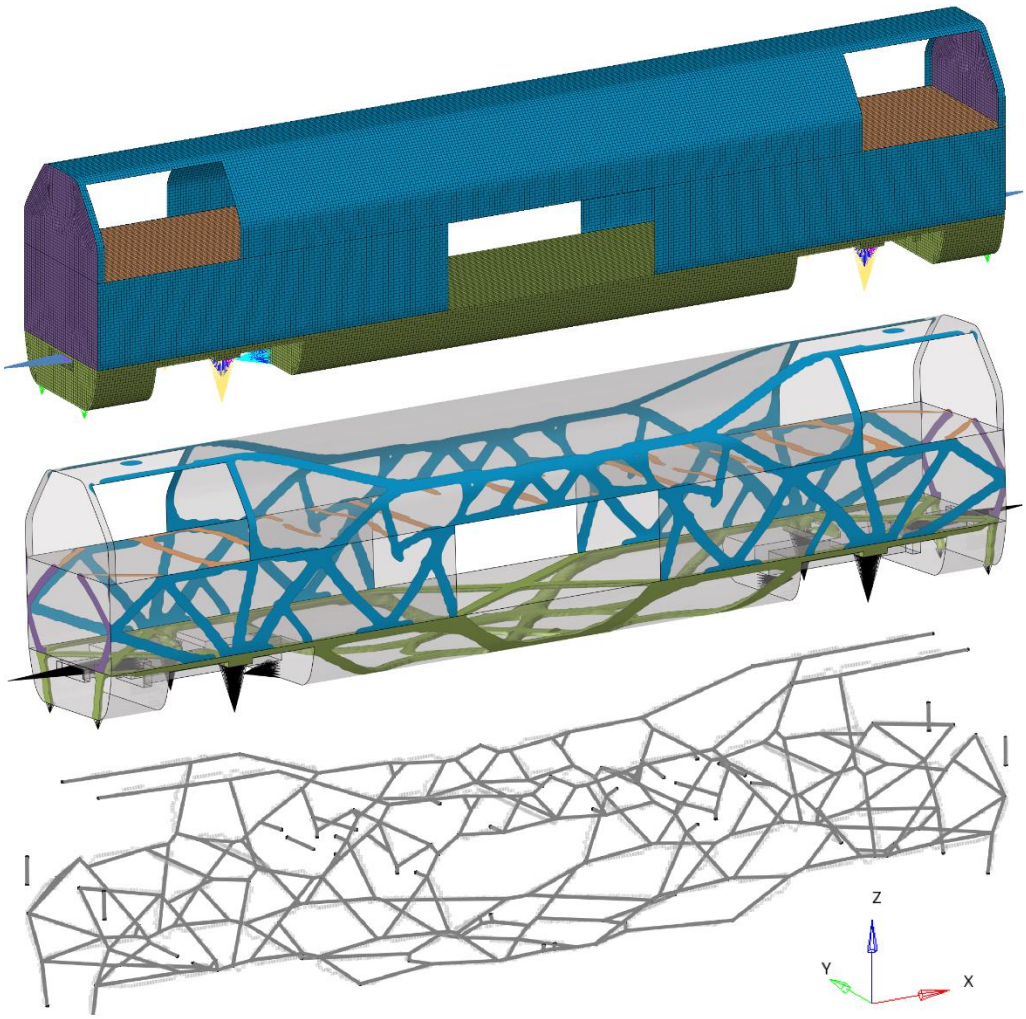

Figure 13: *Train car body model: FE (top), topology optimization results (center) and extracted wireframe (bottom)*

As described in section 4, the cross section extraction is currently only capable of handling three-dimensional data. Consequently, the calculations were run again with only the undercarriage which consists of penta and hexa elements. Figure 14 shows the topology optimization result of only the undercarriage (top), the extracted wireframe (center) and the extracted cross sections (bottom).

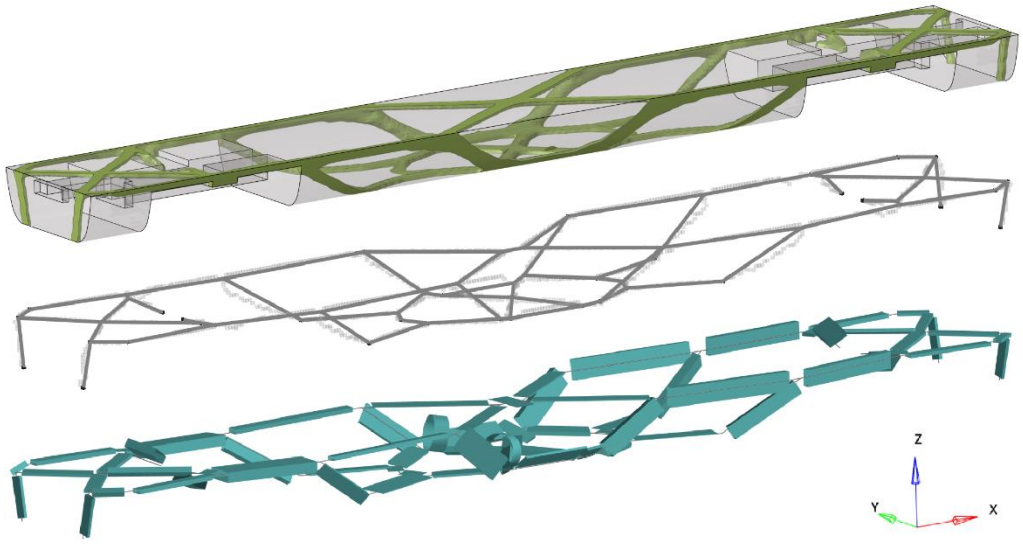

Figure 14: *Processed undercarriage: Topology optimization (top), wireframe (center) and beam sections (bottom)*

6. Conclusion

In this paper, a process is proposed for automatically processing density-based topology optimization results with mixed two- and three-dimensional finite elements. After manually selecting the desired element density threshold, the two-fold process first extracts a wireframe using the density distribution results. The finite element mesh is converted into an equidistant grid of voxels. This allows for a geometrically accurate derivation of information from the finite element mesh. A skeletonization algorithm reduces the branches to one-voxel-wide representations. Structural junction voxels are then detected within the set of skeleton voxels. The connections between the junction voxels are then smoothed to correctly follow the skeletal contour. Trimming and filtering operations finalize this first process step which produces a wireframe model. If desired, the wireframe can be axially symmetric on up to three planes.

After having extracted the wireframe, the second step of the proposed process is to suggest cross section geometries for each branch of the wireframe using the geometric mesh information and the elemental stress results from the topology optimization. For this, the local load case for every wireframe beam (tension, compression, bending or torsion) is detected using stress distribution graphs of the elemental stresses. Then, a geometric shape is derived by segmenting each beam and flattening the density, volume and elemental stress information into two-dimensional slices. From this, hulls are extracted and averaged to identify one best-fitting cross section from a selection of cross sections (filled/unfilled rectangle or circle and I-beam). The result is a geometrically accurate wireframe where each branch possesses cross section properties.

The process is planned to be refined and extended to enable the fully automatic conversion of topology optimization results into basic parametric CAD models, ready to be used in a typical product development process. To enable this, the wireframe geometry extraction can be refined to produce more continuously straight branches.

The suggestions for cross sections can also be refined, possibly by using machine learning algorithms to consider local and global stiffness or compliance, or to support more cross section geometries. Furthermore, the geometric design of the junction points, i.e. providing smooth transitions between beams can also be subject of further research.

To validate the proposed wireframe and cross sections, an FE analysis can be used to validate the models against the original topology optimization result. By applying the original load cases from the topology optimization, the model can be validated considering stresses, compliance or mass.

7. References

Altair Engineering Inc. (2021). *C123*. https://web.altair.com/c123. Accessed 22nd June 2021.

Bendsøe, Martin; Sigmund, Ole (2003). *Topology Optimization - Theory, Methods and Applications*: Springer.

Cao, Junjie; Tagliasacchi, Andrea; Olson, Matt; Zhang, Hao; Su, Zhinxun (2010). Point Cloud Skeletons via Laplacian-Based Contraction. In: *Proc. of IEEE Conf. on Shape Modeling and Applications*: IEEE. 187-197.

Duckham, Matt; Kulik, Lars; Worboys, Mike; Galton, Antony (2008). Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. In: *Pattern Recognition* 41 (10). 3224-3236.

Hsu, Ming-Hsiu; Hsu, Yeh-Liang (2005). Interpreting three-dimensional structural topology optimization results. In: *Computers & Structures* 83 (4-5). 327-337.

Huang, Jian; Yagel, Roni; Filippov, Vassily; Kurzion, Yair (1998). An accurate method for voxelizing polygon meshes. In: *IEEE Symposium on Volume Visualization*. 119-126.

Ju, Tao; Baker, Matthew L.; Chiu, Wah (2007). Computing a family of skeletons of volumetric models for shape description. In: *Computer Aided Design* 39 (5). 352-360.

Jung, Clàudio Rosi; Schramm, Rodrigo (2004). Rectangle detection based on a windowed Hough transform. In: *Proceedings. 17th Brazilian Symposium on Computer Graphics and Image Processing*. IEEE. 113-120.

Klette, Gisela (2006). Branch Voxels and Junctions in 3D Skeletons. In: *Combinatorial Image Analysis* 4040. Springer. 34-44.

Koguchi, Atsushi; Kikuchi, Noboru (2006). A surface reconstruction algorithm for topology optimization. In: *Engineering with Computers 22 (1).* 1-10.

Larsen, Shane; Jensen, C. Greg (2009). Converting Topology Optimization Results into Parametric CAD Models. In: *Computer-Aided Design and Applications* 6 (3). 407-418.

Lee, Ta-Chih; Kashyap, Rangasami L. (1994). Building Skeleton Models via 3-D Medial Surface Axis Thinning Algorithms. In: *CVGIP: Graphical Models and Image Processing* 56 (6). 462-478.

Lin, C.-Y.; Chao, L.-S. (2000). Automated image interpretation for integrated topology and shape optimization. In: *Structural and Multidisciplinary Optimization* 20 (2). 125-137.

Nana, Alexandre; Cuillière, Jean-Christophe; Francois, Vincent (2017). Automatic reconstruction of beam structures from 3D topology optimization results. In: Computers & Structures 189. 62-82.

Ostrosi, Egon; Bluntzer, Jean-Bernard; Stjepandic, Josip (2020). A CAD Material Skeleton-Based Approach for Sustainable Design. In: *Transdisciplinary Engineering for Complex Socio-technical Systems – Real-life Applications*. IOS Press. 700-709.

Ramsaier, Manuel; Till, Markus; Schumacher, Axel; Rudolph, Stephan (2019). On a Physics-based Reconstruction Algorithm for Generating Clean Parametric Native CAD-Models from Density-based Topology Optimization Results. In: *The World Congress of Structural and Multidisciplinary Optimization*.

Swierstra, Marco K.; Gupta, Deepak K.; Langelaar, Matthijs (2020). Automated and Accurate Geometry Extraction and Shape Optimization of 3D Topology Optimization Results. In: *NAFEMS European Conference on Simulation-Based Optimisation.*

Tang, Poh-Soong; Chang, Kuang-Hua (2001). Integration of topology and shape optimization for design of structural components. In: *Structural and Multidisciplinary Optimization* 22 (1). 65-82.

van der Walt, Stéfan; Schönberger, Johannes L.; Nunez-Iglesias, Juan; Boulogne, François; Warner, Joshua D.; Yager, Neil et al. (2014). *scikit-image: image processing in Python*. In: PeerJ 2, e453.