

**Title:****Software Tool for Interactive Design of Customized Hand Splints****Authors:**

Adam Gorski, agorski3@uwo.ca, Western University  
 O. Remus Tutunea-Fatan, rtutunea@eng.uwo.ca, Western University  
 Louis M. Ferreira, lferreir@uwo.ca, Western University

**Keywords:**

Customized Hand Splint, Interactive Design, Software Tool

DOI: 10.14733/cadconfP.2023.265-270

**Introduction:**

In physiotherapy, hand splints are frequently used to immobilize and support certain parts of a patient's hand that have been injured. At present, these splints are made from thermoplastics and require the physical presence of the patient for the molding process. One possible splint fabrication alternative could involve 3D scanning and printing. In this approach, the splint design process would start with a scan of patient's hand and would end with splint fabrication by means of additive manufacturing approaches. These solutions can help promote the growth of telemedicine, where in-person visits are no longer necessary or at least they are no longer mandatory [1]. However, physiotherapists may not have experience with computer-aided design (CAD) software, thus making it challenging for them to create the required customized splints. While automated splint generation has been attempted, this approach has typically relied in the past on commercial CAD software, a tool that might be difficult to use for non-technical users [2].

Building on this, the main objective of this study was to create a user-friendly, linear, and standalone custom parametric CAD solution that can automatically generate hand splints based on a few user-specified parameters and without the need for external CAD software libraries. The software developed for this purpose can generate hand splints with different parameters and with minimal user involvement.

**Interactive Design Process:**

The software was designed with a straightforward, step-by-step process that guides the user through the splint creation process. To begin, the user chooses an STL file, which is a 3D scan of the hand. These scans can be obtained using high-end systems or lower-cost scanners, which have been shown to provide accurate results [3]. After selecting the STL file, the user must verify the model's scale and align it with the coordinate system floor, a step that can be completed either automatically or manually. After that, the user has to specify three cutoff planes for the model. These planes rely on thumb, wrist, and finger positions. User also needs to choose the splint's offset tolerance and thickness, as well as the edge smoothing mode. In the trimming stage, the user must specify at least four points for the splint's outline and can add additional points and control the tangency of the Bezier curves applied to the splint model. The radius of the splint ventilation holes, their vertical or horizontal number and position can also be controlled by the user. Finally, the model is thickened and customization options such as stamping or extruding text can be added to the splint (Fig. 1).

**Design Tool Workflow:**

The entire application was developed from scratch. Its overall functionality is presented in Fig. 2. The user interface and interaction code were implemented using C#, while high-performance code was

written in C++. Platform invokes and direct pointers were used to improve the interoperability process in the C# code while OpenMP was used for parallelization on the C++ side. A custom-built OpenGL library was used for 3D rendering and Windows Forms were used. Advanced coding techniques were employed to enhance performance on the C++ side, particularly for fast model rebuilds. The interaction was achieved through either ray-casting or a spare framebuffer in which each pixel represented the ID of the rendered object and could be accessed through a mouse click. The program consisted of only an executable and two dynamic link library files, and could be run independently without the need for any additional software.

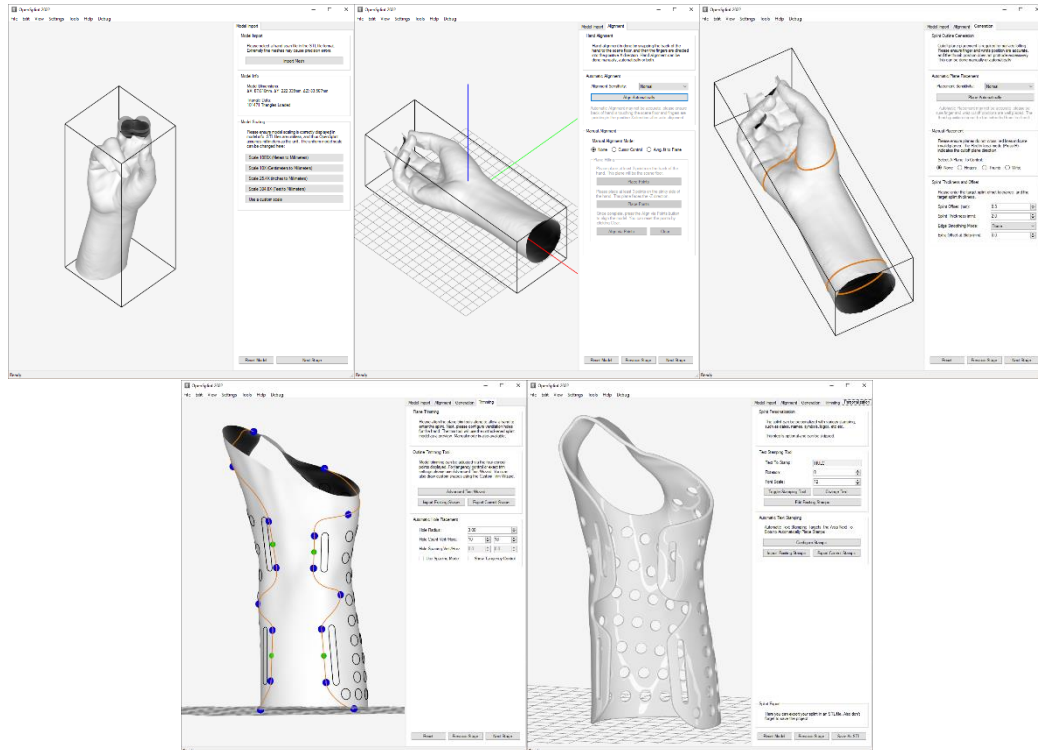


Fig. 1: Stages of the interactive splint design process: reverse engineering of the hand geometry, hand geometry alignment, splint surface trimming, splint geometry completion, volume addition.

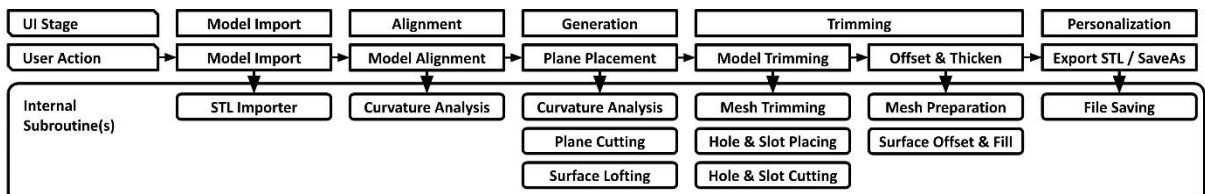


Fig. 2: Application workflow.

#### Curvature-Based Hand Feature Detection:

To streamline the splint generation process, an automated feature detection algorithm was developed to automatically detect and align features of a hand scan. This decreases the required user input to generate a splint, so that an entire splint will be generated by pressing one button on the interface. The algorithm

relies on angles between neighboring triangles to classify high and low curvature regions. The algorithm processes each triangle and flags all touching triangles that are within a specific angle as long as they touch the other flagged triangles. After that, the total area is computed and compared with a minimum area that is preset. If the computed area is bigger then all analyzed triangles are flagged on a global buffer. Finally, the resulting triangles are grouped into touching regions. The resulting groups are then used as reference points - such as average positions or surface normals - to build alignment matrices or to determine positions for plane cutoffs (Fig. 3). This algorithm is generally used to target large flat areas that can be used to determine the back of the hand. Side alignment can be achieved by using a filter on starting triangles to restrict the normal direction of the surface thus implying that only side-facing triangles are used as starting points for search.

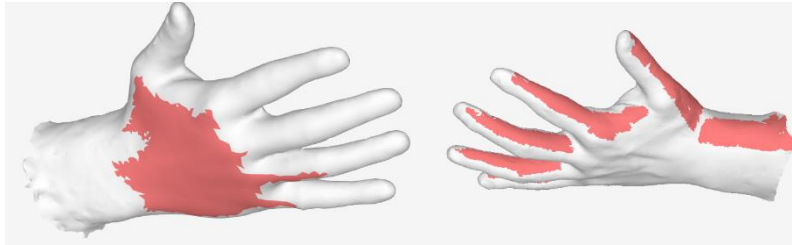


Fig. 3: Using high and low curvature detection for hand feature extraction.

#### Surface Lofting:

The thickening and offsetting of triangular meshes is restricted by geometry, so a surface loft is built between the wrist and finger cutoff planes. This lofting is also crucial for grid data powered automated hole positioning and trimming line creation. Intermediate planes are constructed by interpolating between start and end planes, with the thumb hole being closed off. The thumb cutting plane is slightly offset to provide a buffer between the grid data and the pre-thickened mesh in order to avoid floating point and smoothing errors. These planes are used to slice the model and generate outlines, which are aligned for consistent winding. A second centerline cutting plane perpendicular to all outlines is created, intersected with previously-generated outlines to align their starting vertices with the alignment path. Finally, all paths are divided into 50 segments, resulting in grid data that is smoothened to remove sharp edges. The entire process is depicted in Fig. 4.

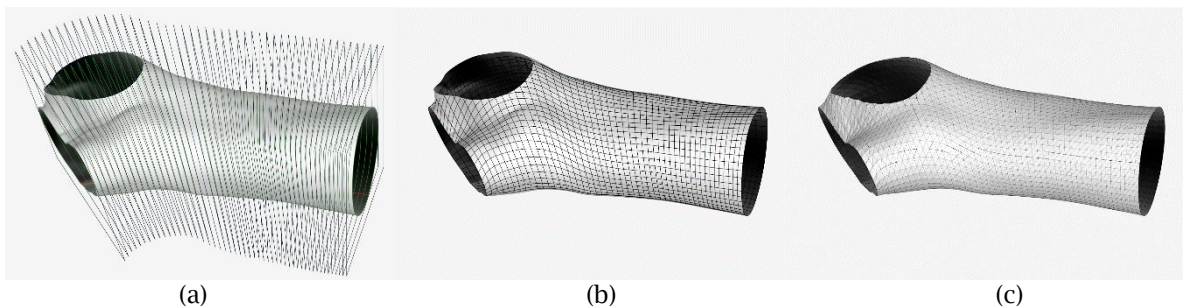


Fig. 4: Surface lofting: (a) interpolated slice lines, (b) resulting grid data and (c) raw mesh triangles.

#### Mesh Trimming:

All instances of mesh slicing use similar algorithms to perform cuts on a mesh and all cutting operations are completed before thickening. Planar slicing utilizes linear systems to determine the vertices of a triangle that lie on the undesirable side of the system. This information is then used in conjunction with XOR patterns to determine what edges of the triangle should be taken into consideration during line-

plane intersections. Non-planar cutting operations utilize custom algorithms to prevent triangle vertices from inadequately falling on other triangle edges, thus avoiding false edge detections.

The mesh cutting process involved two approaches: projected polygon cutting and quad cutting. The projected polygon cutting algorithm projected a cutline path onto the triangle by intersecting the triangle with the cutting segment and vice-versa. The resulting paths were then compared to the edges of the triangle and processed through a winding algorithm to identify the individual polygons that made up the cut. These polygons were then converted to triangles by means of a polygon to triangle converter. The triangles were examined to determine if they were located inside the cut polygon and if they were, the entire polygon was disregarded. This process is illustrated in Fig 5.

The mesh trimming algorithm operates similarly to the mesh cutting algorithm, with two main differences. Firstly, the planar slicing code employed slicing quads instead of projecting a cut polygon. Secondly, the triangle rejection method was based on area propagation, rather than on triangle's position within the polygon.

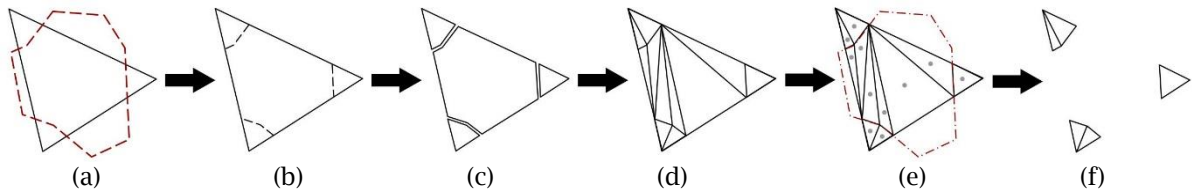


Fig. 5: Mesh cutting phases: (a) cut path projected against triangle, (b) paths placed on triangle, (c) paths intersected to build polygons, (d) polygons split into triangles, (e) triangles are filtered out based on centroids, and (f) end result.

#### Splint Geometry:

This phase involves removing a central piece from a splint and creating slots and breathability holes in the mesh. The process begins with mesh trimming, a process that involves wrapping a 2D Bezier curve onto 3D mesh grid data as specified by the user. The Bezier paths are constructed from user-defined points, with tangency direction determined by the positions of neighboring points, and tangency controlled by the user. The points are transformed into a 2D plane and divided by a triangle intersection grid that splits the Bezier paths along mesh triangle edges. The translated points are then interpolated onto the mesh, and intersection quads are generated to cut the back of the mesh (Fig 6). Cutting is performed after mesh trimming is completed.

The position of the breathing holes was determined by interpolating the  $X$  and  $Y$  coordinates of the hole center onto the splint mesh grid. The  $X$  and  $Y$  coordinates were calculated either by using a specified horizontal/vertical hole count or by means of a target hole spacing mode. Splint breathing holes were only placed if they fell within the boundaries of the unwrapped 2D Bezier curves. The position of the strapping slots was determined by means of a perpendicular plane thus allowing for the specification of an exact distance from the edge.

#### Volume Addition:

The last stage involves two offset procedures. The first offset is used to create a gap between the inner surface of the splint and hand skin add thickness while the second one is used to add thickness/volume to the mesh/surface. The second offset is mandatory for additive manufacturing purposes. The edge finder tool is utilized to identify all edges of the 3D surface before any offset is made. After thickening the mesh, the contour gap present between the inner and outer surface is filled. If a smoother contour edge is desired, a Bezier dome can be used instead of flat surface filler (Fig. 7). The offset process involves computing the average normal of the triangles that share a vertex and shifting the vertex by a specified distance. This technique leads to smooth surfaces that do not have self-intersections because the mesh was already smooth prior to the supplementary smoothing step [4].

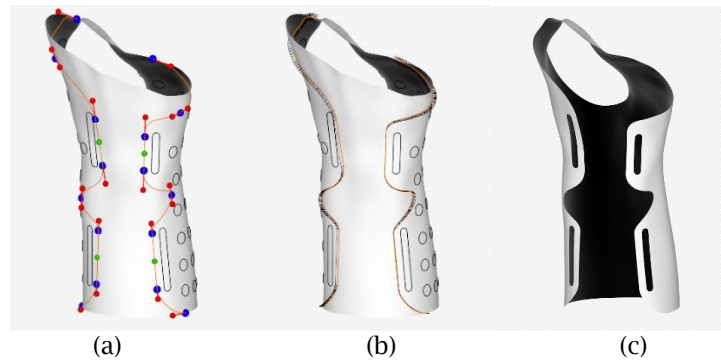


Fig. 6: Mesh trimming and cutting: (a) adjustable points, (b) cutting quads and (c) final mesh.

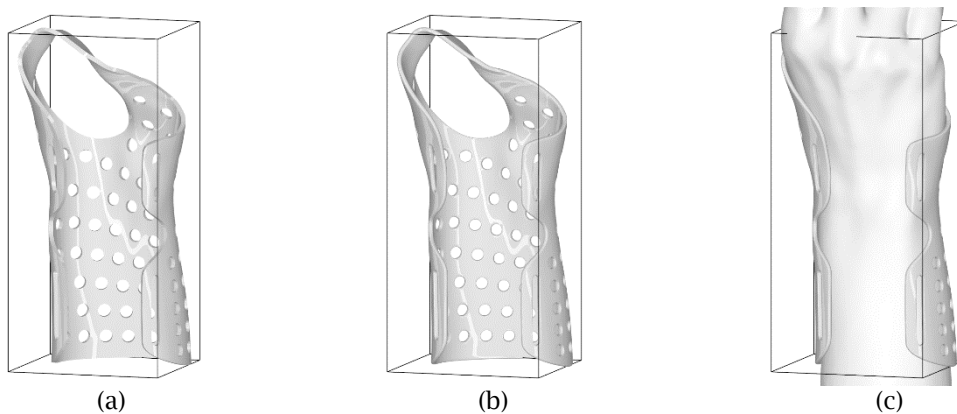


Fig. 7: Mesh thickening phase: (a) pre-smoothing edges, (b) post-smoothing edges and (c) virtual splint fitting test.

### Conclusions:

The application of a customized splint building software has proven to be capable of constructing tailor-made splints in a quick and efficient manner. With just a few clicks, the splint generator application can produce customized splints. The software can also be adjusted and enhanced to support a wider range of splint types while still remaining user-friendly. This eliminates the need for physiotherapists to acquire CAD expertise for splint modeling purposes. As a result, the developed software tool could significantly enhance patient's experience in a sense that - unlike the conventionally-fabricated thermoplastic molded splints - customized 3D printed splints can be fabricated to more precisely match the unique configuration of each patient's hand.

### Acknowledgement

This research was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) and Mitacs Canada.

### References:

- [1] Bashshur, R.; Doarn, C.-R.; Frenk, J.-M.; Kvedar, J.-C.; Woolliscroft, J.-O.: Telemedicine and the COVID-19 pandemic, Lessons for the future, *Telemedicine and e-Health*, 26(5), 2020, 571-573. <https://doi.org/10.1089/tmj.2020.29040.rb>

- [2] Li, J.; Tanaka, H.: Rapid customization system for 3D-printed splint using programmable modeling technique – a practical approach, 3D Printing in Medicine, 4, 2018. <https://doi.org/10.1186/s41205-018-0027-6>
- [3] Yang, Y.; Xu, J.; Elkhuisen, W.-S.; Song, Y.; The development of a low-cost photogrammetry-based 3D Hand Scanner, HardwareX, 10, 2021. <https://doi.org/10.1016/j.ohx.2021.e00212>
- [4] Qu, X.; Stucker, B.: A 3D surface offset method for STL-format models, Rapid Prototyping Journal, 9(3), 2003, 133–141. <https://doi.org/10.1108/13552540310477436>