Bezier surface reconstruction from scanned point cloud of a forearm stump for commercial CAD tool

Alon Spinner, Yam Ben-Natan

Supervisors:
Anath Fischer, Ronit Schneor

Alon Wolf, Yair Herbst

Faculty of Mechanical Engineering, Technion

Laboratory for CAD & Lifecycle Engineering
Faculty of Mechanical Engineering
Technion – Israel Institute of Technology
Haifa 32000, Israel

Email:
AlonSpinner@campus.technion.ac.il
Yam.B@campus.technion.ac.il

Contents

Table of figures

1 Introduction

To date, the design and production of a prosthesis is an expensive process which requires manual procedures such as measurements of the stump performed by a technician.

The project takes parts with the E-Nable nonprofit organization that manufactures fitted prostheses for people of limited means. Most of the patients in the association are children, which is because the need for a new prosthesis comes very often, a frequency like shoe replacement in children. Due to the high frequency the need for a cheap product is tremendous among this population

This project aims to improve the measuring process by suggesting a new way to procure these measurements - scanning the stump with a 3D camera. The motivation for this kind of solution have many faces:

- By transforming the scan to a CAD model, the scan will optimize prosthesis modeling and help to match the product to the specific patient.
- the technician will be allowed to measure more relevant distances, faster and with greater ease.
- This process will also open doors for advanced prosthesis planning methods using the negative of the stump surface.

2 Literature Review

The purpose of the project is to perform a hand scan and turn the scan into parametric surfaces designed for commercial modeling software. Therefore, throughout the survey, we will deal with these areas, in order respectively.

2.1 3D scanning technologies

We focused our research on active optical non-contact distance measurement methods which are rising in popularity in the recent years. The two leading scanning technologies are:

1. Structured Light:
   The method involves projecting a structured light pattern, usually alternating stripes, onto an object, and then filming it with one or more cameras to capture the ways in which the object deforms the light patterns. By triangulating multiple images of the scan one can calculate the dimensions of the object.

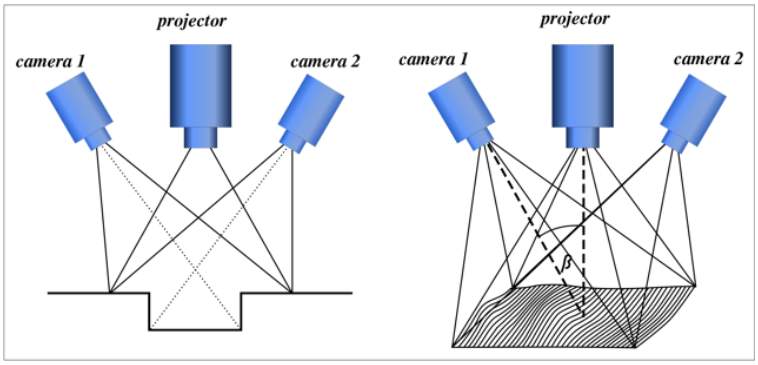


Figure 1 - Structred light concept [1]

2. Time of flight:
   A point-wise ToF sensor estimates its radial distance from a scene point by the time of flight principle. In simple words, since light travels in the air with constant velocity c, the distance ρ covered at time τ by an optical radiation is $\rho = c\tau$ [1]. More sophisticated methods involve measuring the phase difference between the light output and surface reflected light.
   Intel's RealSense camera, which our scanner is equipped with uses the ToF technology.

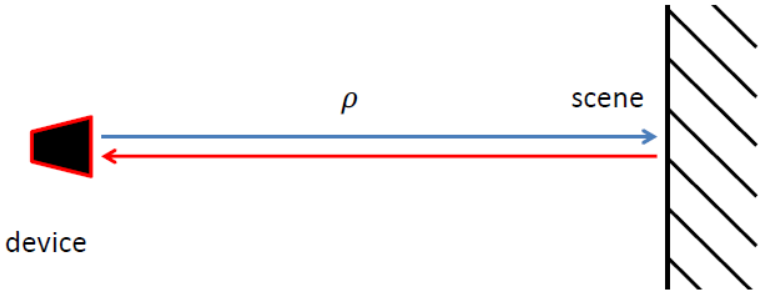


Figure 2 - Time of flight concept

2.2 Smoothness and Continuity

Parametric continuity can be described as follows:

C^0 Curves are continuous

C^1 First derivatives are continuous

C^n First through nth derivatives are continuous

Geometric continuity can be defined for vector functions and allows for greater flexibility in curve composition. It is defined as such:

G^0 The curves touch at the joint point

G^1 The curves also share a common tangent direction at the joint point

G^2 The curves also share a common center of curvature at the joint point

Two vector functions $f(t), g(t)$ have G^n continuity if $f^{(n)}(t) \neq 0$ and $f^{(n)}(t) = kg^{(n)}(t)$ for a scalar $k > 0$.

G^n Continuity exists if the curves can be reparametrized to have C^n continuity. A reparameterization of the curve is geometrically identical to the original; only the parameter is affected.

2.3 Bezier Curves

The Bernstein-Bezier polynomial curve defined by control points $\mathbf{b}_i \in \mathbb{R}^{mx1}$ and running parameter $t \in [0,1]$ is defined as such:

$$\mathbf{x}(t) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t) = \begin{bmatrix} B_0^n(t) & \cdots & B_n^n(t) \end{bmatrix} \begin{bmatrix} \mathbf{b}_0 \\ \vdots \\ \mathbf{b}_n \end{bmatrix}$$

$$B_i^n(t) = \frac{n!}{(n-i)!i!} t^i (1-t)^{n-i}$$

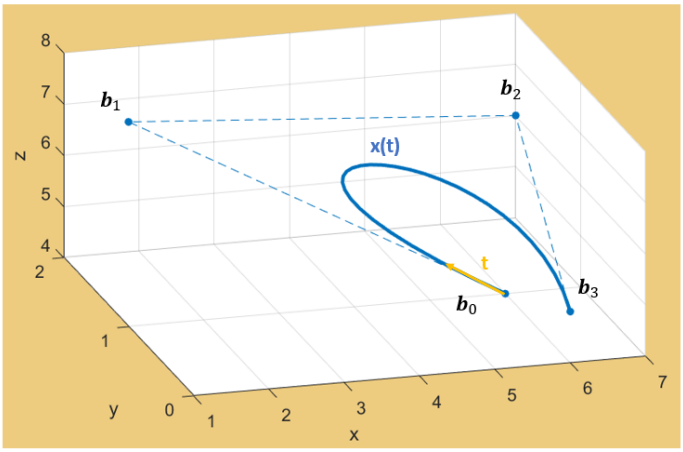


Figure 3 - Bezier curves control points and properties

The properties of the Bezier curves include the following:

- *Endpoint interpolation:* the curve passes through the polygon endpoints $\mathbf{x}(0)=\mathbf{b}_0;\quad \mathbf{x}(1)=\mathbf{b}_n$
- *Symmetry*: the two polygons $\mathbf{b}_0...\mathbf{b}_n$ and $\mathbf{b}_n...\mathbf{b}_0$ describe the same curve. Only the direction of the traversal parameter is changed.
- *Convex hull*: A point $\mathbf{x}(t)$ on the curve for $t\in[0,1]$ is in the convex hull of the control polygon defined by $\mathbf{b}_i$
- *Variation diminishing*: If a straight line intersects a planar Bezier polygon m times, then the line can intersect the curve at most m times. In other words, the curve doesn't wiggle more than the polygon.
- *Pseudo-local control*: Suppose we move the i^{th} control point. The curve changes most near $t=\frac{i}{n}$. In fact, all points on the curve move in a direction parallel to the vector formed by the difference of the old and new control point.
- *Affine invariance*: if an affine map Φ is applied to the control polygon, then the curve is mapped by the same map.

$$\sum_{i=0}^{n}\Phi(\mathbf{b}_i)B_i^n(t)=\Phi\left(\sum_{i=0}^{n}\mathbf{b}_iB_i^n(t)\right)$$

2.3.1 Derivatives of Bezier curves and implication for their composition

Differentiating $\mathbf{x}(t)$ with respect to t produces a tangent vector of the curve.

$$\frac{d\mathbf{x}(t)}{dt}=\dot{\mathbf{x}}(t)=n\sum_{i=0}^{n-1}\Delta\mathbf{b}_iB_i^{n-1}(t)$$
$$\frac{d^2\mathbf{x}(t)}{dt^2}=\ddot{\mathbf{x}}(t)=n(n-1)\sum_{i=0}^{n-2}\Delta^2\mathbf{b}_iB_i^{n-2}(t)$$

$$where:\qquad \Delta\mathbf{b}_i=\mathbf{b}_{i+1}-\mathbf{b}_i$$
$$\Delta^2\mathbf{b}_i=\Delta(\Delta\mathbf{b}_i)=\mathbf{b}_{i+2}-2\mathbf{b}_{i+1}+\mathbf{b}_i$$

Calculating the first and second derivatives at the end points:

$$\dot{\mathbf{x}}(0)=n\Delta\mathbf{b}_0;\quad \dot{\mathbf{x}}(1)=n\Delta\mathbf{b}_{n-1}$$
$$\ddot{\mathbf{x}}(0)=n(n-1)\Delta^2\mathbf{b}_0;\quad \ddot{\mathbf{x}}(t)=n(n-1)\Delta^2\mathbf{b}_{n-2}$$

Compositing two curves $\mathbf{x}_1,\mathbf{x}_2$ of similar degree n at $\{\mathbf{x}^A(t=1),\mathbf{x}^B(t=0),u=0.5\}$, where $u\in[0,1]$ is the parameter of the new composed curve, in a smooth joint will require the following conditions.

$$C^0: \quad \mathbf{b}_n^{\,A} = \mathbf{b}_0^{\,B}$$

$$C^1: \quad \Delta\mathbf{b}_{n-1}^{\,A} = \Delta\mathbf{b}_0^{\,B}$$

$$C^2: \quad \Delta^2\mathbf{b}_{n-2}^{\,A} = \Delta^2\mathbf{b}_0^{\,B}$$

It is also possible to to join the two curves at $\left\{\mathbf{x}^A(t=1), \mathbf{x}^B(t=0)\right\}$ with a different u_{joint} value. In that case, to ensure C^1 one demands:

$$\frac{1}{u_{joint}}\Delta\mathbf{b}_{n-1}^{\,A} = \frac{1}{1-u_{joint}}\Delta\mathbf{b}_0^{\,B}$$

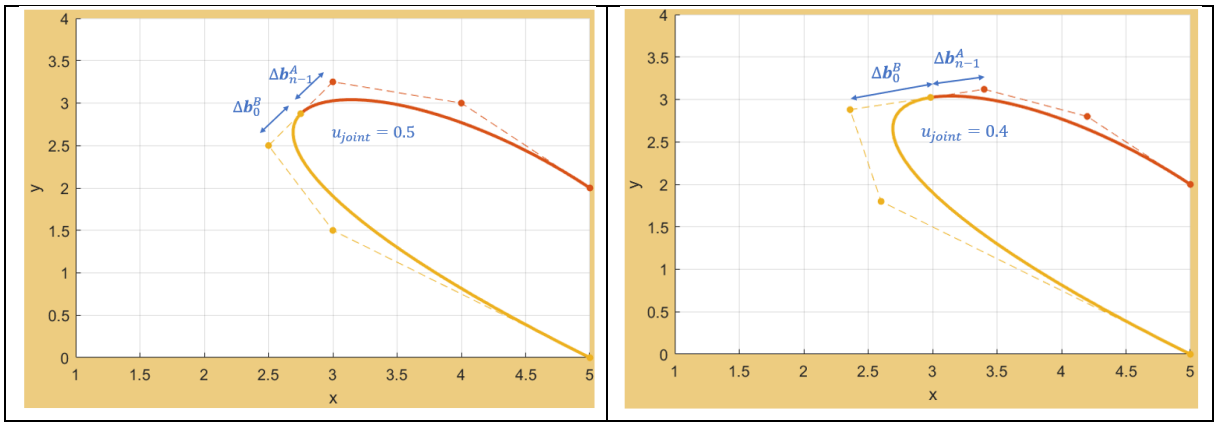


Figure 4 - Composite Bezier curves

2.3.2 De Casteljau Algorithm

The de Castljau algorithm provides a mean for evaluating Bezier curves. Given Bezier control points $\mathbf{b}_i$ and parameter $t \in [0,1]$, we are to find $\mathbf{x}(t_0)$.

The algorithm does so by computing new sets of control points $\mathbf{b}_i^{\,j}$ where $\mathbf{b}_i^{\,0}$ are the original control points, and $\mathbf{b}_i^{\,n}$ is the desired point on the curve.

The sets are computed iteratively as follows:

$$\mathbf{b}_i^{\,r} = (1-t_0)\mathbf{b}_i^{\,r-1} + t_0\mathbf{b}_{i+1}^{\,r-1} \quad \begin{cases} r = 0,\ldots,n \\ i = 0,\ldots,n-r \end{cases}$$

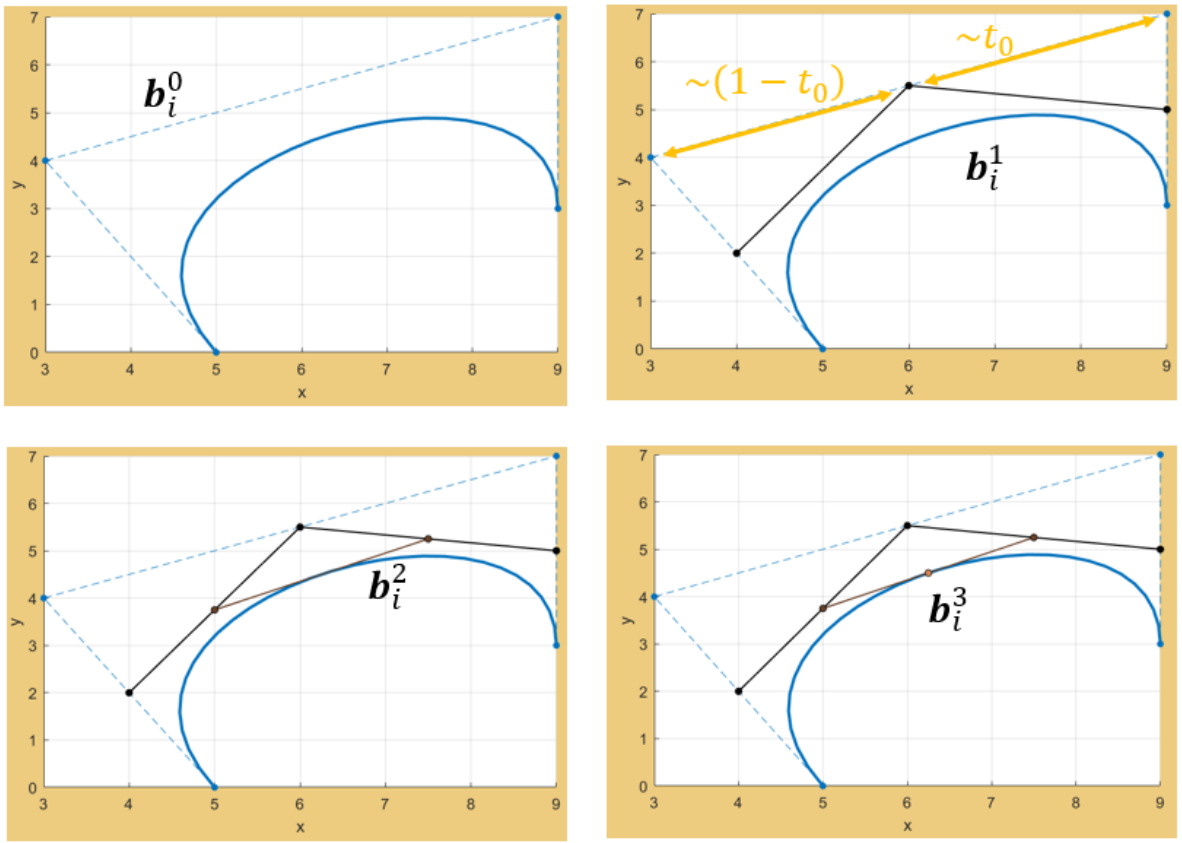


Figure 5 - Evaluating a point on a Bezier curve

It is also possible to reduce the problem of computing derivatives of the curve through the de Castljau algorithm. For example, the first derivative is a scaling of the difference vector $\mathbf{b}_1^{n-1} - \mathbf{b}_0^{n-1}$.

2.3.3 Bezier curve degree elevation

A degree n polynomial $\mathbf{x}(t)$ is also one of degree $n+1$. To obtain the higher degree form from the lower one, one must equate coefficients for the same powers of t .

$$\mathbf{x}(t) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t) = \sum_{i=0}^{n+1} \mathbf{c}_i B_i^n(t)$$

Solving for $\mathbf{c}_i$ produces:

$$\mathbf{c}_i = \frac{i}{n+1}\mathbf{b}_{i-1} + \left(1 - \frac{i}{n+1}\right)\mathbf{b}_i$$

Higher degree polynomials enable us to ensure higher degrees of continuity without over-constraining the composite curves.

Repeated degree elevation results in a polygon which converges to the curve. It is important to note that this is not a practical method to render the curve itself.

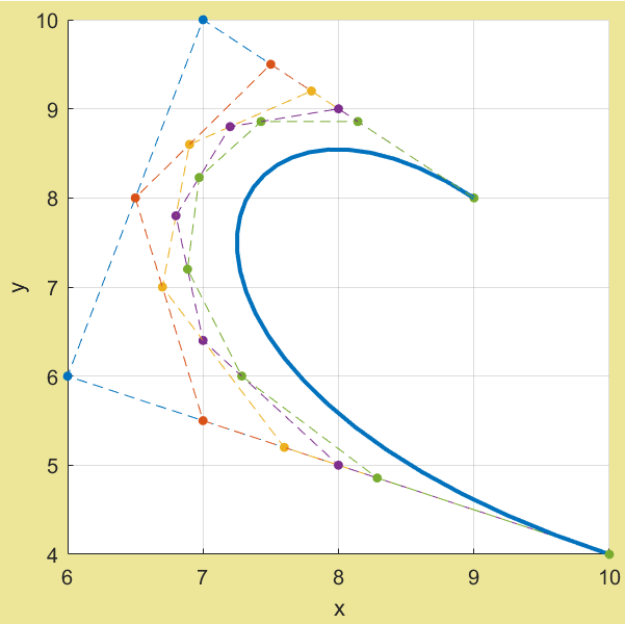


Figure 6 - Elevating a Bezier curve

2.4 Bezier Patches

Rectangular Bezier patches of degree (m,n) take form by the following formula:

$$\mathbf{x}(u,v)=\sum_{i=0}^{m}\sum_{j=0}^{n}\mathbf{b}_{i,j}B_i^m(u)B_j^n(v)=\begin{bmatrix}B_0^m(u) & \cdots & B_m^m(u)\end{bmatrix}\begin{bmatrix}\mathbf{b}_{0,0} & \cdots & \mathbf{b}_{0,n}\\ \vdots & & \vdots\\ \mathbf{b}_{m,0} & \cdots & \mathbf{b}_{m,n}\end{bmatrix}\begin{bmatrix}B_0^n(v)\\ \vdots\\ B_0^n(v)\end{bmatrix}$$

Where $\mathbf{b}_{i,j}$ represent the rectangular control net, or the “polyhedra”.

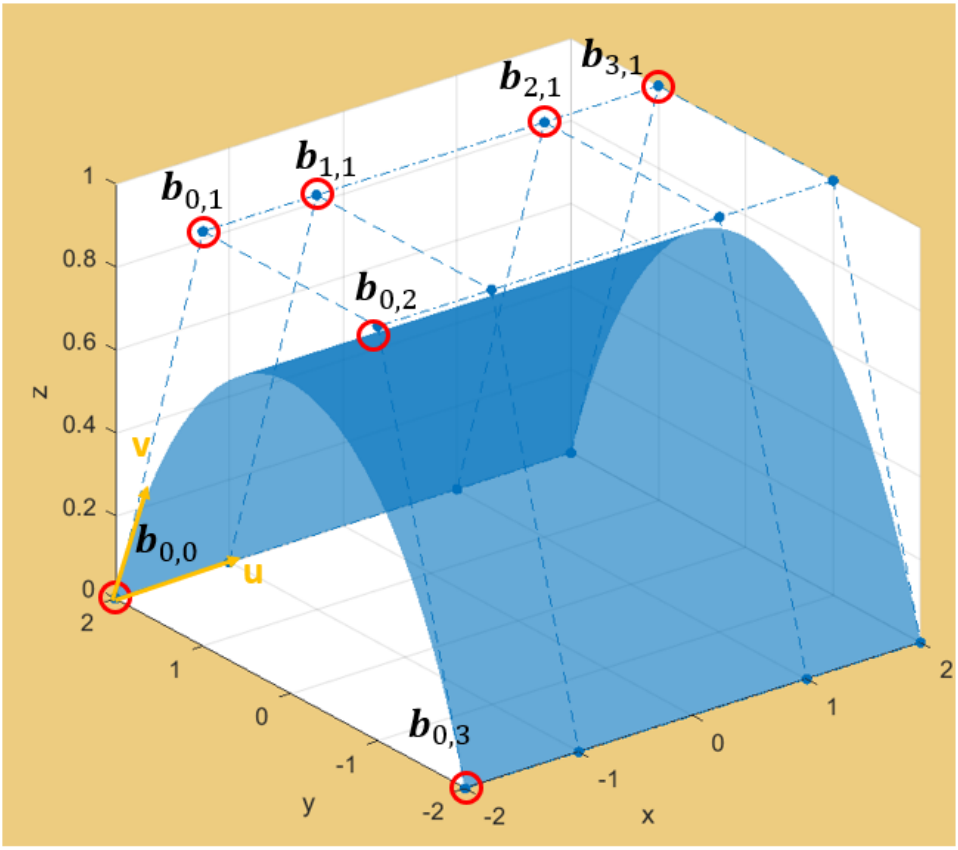


Figure 7 - Rectaunglar Bezier patch control points net

The properties of the Bezier patches include the following:

- *Endpoint interpolation*: The patch passes through the four corner control points. That is:

  $$\mathbf{x}(0,0)=\mathbf{b}_{0,0} \quad \mathbf{x}(1,0)=\mathbf{b}_{m,0}$$
  $$\mathbf{x}(0,1)=\mathbf{b}_{0,n} \quad \mathbf{x}(1,1)=\mathbf{b}_{m,n}$$

  Also, the control net's boundary control points are the control points of the patch boundary curves. For example: the curve $\mathbf{x}(u,0)$ has the control polygon $\mathbf{b}_{i,0} \quad i=0,...,m$.
- *Symmetry*: We could re-index the control net so that any corners corresponds to $\mathbf{b}_{0,0}$ and evaluation would result in a patch with the same shape as the original one.
- *Convex hull*: A Bezier surface lies within in the convex hull defined by its control net.
- *Affine invariance*: Apply an affine map to the control net, and then evaluate the patch. This surface will be identical to the surface created by applying the same affine map to the original patch.
- *Tensor product*: Bezier patches are in the class of tensor product surfaces. This property allows Bezier patches to be dealt with in terms of isoperimetric curves, which in turn simplifies evaluation and other operations. This also implies that the total degree of

(m,n) patch is $2mn$. The total degree is the maximum number of intersections of the patch with a straight line.

2.4.1 Evaluation of Bezier patches

Rewriting the equation for Bezier patch, we could show how the De-Casteljau algorithm (2.3.2) can be used to evaluate a Bezier patch from its control net.

$$\mathbf{x}(u,v)=\sum_{i=0}^{m}\sum_{j=0}^{n}\mathbf{b}_{i,j}B_i^m(u)B_j^n(v)=\sum_{i=0}^{m}B_i^m(u)\left(\sum_{j=0}^{n}\mathbf{b}_{i,j}B_j^n(v)\right)$$

We define a Bezier curve with control points $\mathbf{q}_i$ as follows: $\mathbf{q}_i(v)=\sum_{j=0}^{n}\mathbf{b}_{i,j}B_j^n(v)$.

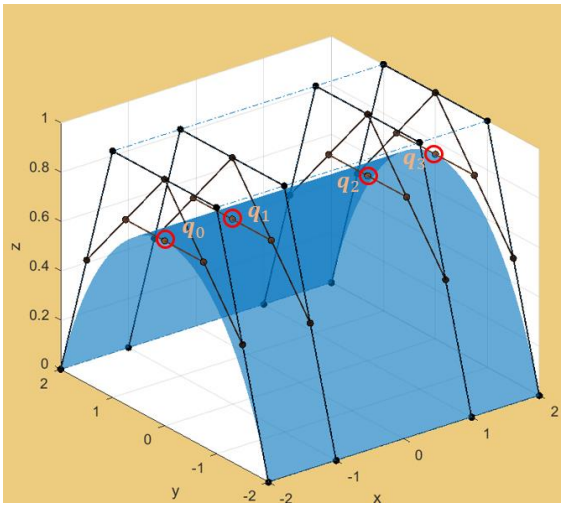
For a fixed $v=v_0$ we have $m+1$ points $\mathbf{q}_0(v_0),...\mathbf{q}_m(v_0)$. Each $\mathbf{q}_i(v_0)$ is a point on the Bezier curve defined by control points $\mathbf{b}_{i,0},\mathbf{b}_{i,1}...\mathbf{b}_{i,n}$. Plugging the defined expression to the patch equation yields:

$$\mathbf{x}(u,v_0)=\sum_{i=0}^{m}\mathbf{q}_i(v_0)B_i^m(u)$$

$\mathbf{x}(u_0,v_0)$ is a point on the Bezier curve $\mathbf{x}(u,v_0)$ defined by the $m+1$ control points $\mathbf{q}_i(v_0)$.

In total, the two-step process for calculating $\mathbf{x}(u_0,v_0)$ requires $m+2$ calls to the De-Casteljau algorithm. The first $m+1$ to calculate $\mathbf{q}_i(v_0)$, and then another to calculate $\mathbf{x}(u_0,v_0)$

First step



Second step

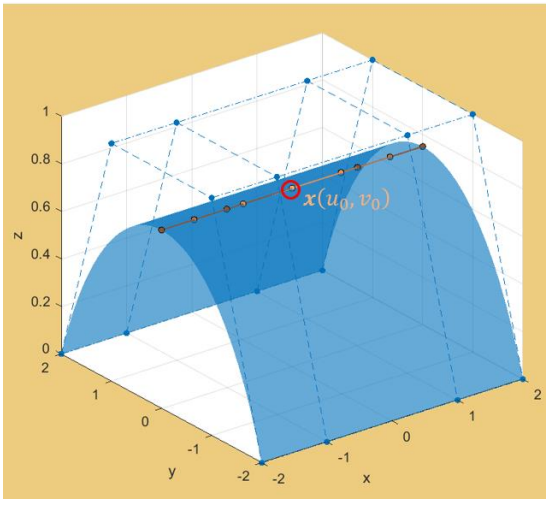


Figure 8 - Evaluating a point on a Bezier patch

2.4.2 Degree Elevation

Degree elevation of Bezier patches is a straight forward generalization of the degree elevation of Bezier curves, and it can be applied separately on both dimensions of the control point network.

$$\mathbf{x}(u,v)=\sum_{i=0}^{m}\sum_{j=0}^{n}\mathbf{b}_{i,j}B_i^{\,m}(u)B_j^{\,n}(v)=\sum_{i=0}^{m+1}\sum_{j=0}^{n}\mathbf{c}_{i,j}B_i^{\,m+1}(u)B_j^{\,n}(v)$$

$$\mathbf{c}_{i,j}=\frac{i}{n+1}\mathbf{b}_{i-1,j}+\left(1-\frac{i}{n+1}\right)\mathbf{b}_{i,j}$$

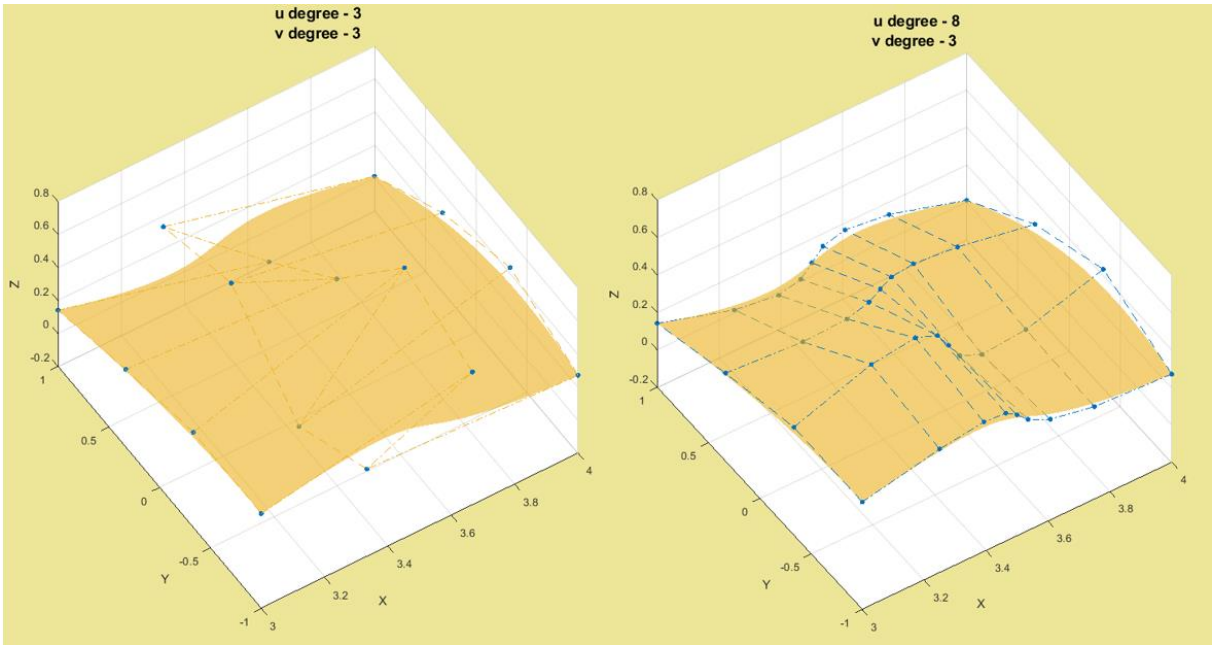


Figure 9 - Bezier patch degree elevation

2.4.3 Bezier patch C_1 continuity

Assume we have two adjacent Bezier patches $\mathbf{x}^A(u,v),\mathbf{x}^B(u,v)$ of the same degree (m,n), with corresponding control points $\mathbf{b}^A{}_{i,j},\mathbf{b}^B{}_{i,j}$ which we were to compose to a C_1 surface through the u direction. Positional continuity across the boundary will require that $\mathbf{x}^A(1,v)=\mathbf{x}^B(0,v)$ for all $v\in[0,1]$, which leads to $\mathbf{b}^A{}_{m,j}=\mathbf{b}^B{}_{0,j}$. The common boundary curve between the two patches requires a common boundary polygon between the two characteristics polyhedra.

For gradient continuity across the boundary, the tangent plane of $\mathbf{x}^A$ on $u=1$ must coincide with that of $\mathbf{x}^B$ on u=0, for all $v\in[0,1]$. In other words, the direction of the surface normal must be continuous across the boundary:

$$\mathbf{x}^A{}_u(0,v)\times\mathbf{x}^B{}_v(0,v)=\lambda(v)\mathbf{x}^A{}_u(1,v)\times\mathbf{x}^B{}_v(1,v)$$

Faux and Pratt [1] suggest two ways of solving this equation.

First Solution:

The simplest solution is to take $\mathbf{x}^2{}_u(0,v)=\lambda(v)\mathbf{x}^1{}_u(1,v)$ which then solves for:

$$\mathbf{b}^{B}_{1,j} - \mathbf{b}^{B}_{0,j} = \lambda\left(\mathbf{b}^{A}_{m,j} - \mathbf{b}^{A}_{m-1,j}\right)$$

In terms of polyhedra of the two patches, the requirement is that the pairs of polyhedron edges which meet at the boundary much be collinear, and the cross-boundary tangent magnitude ratio λ must be constant along the common boundary. λ is also the distance ratio between the pairs of polyhedron edge's control points.

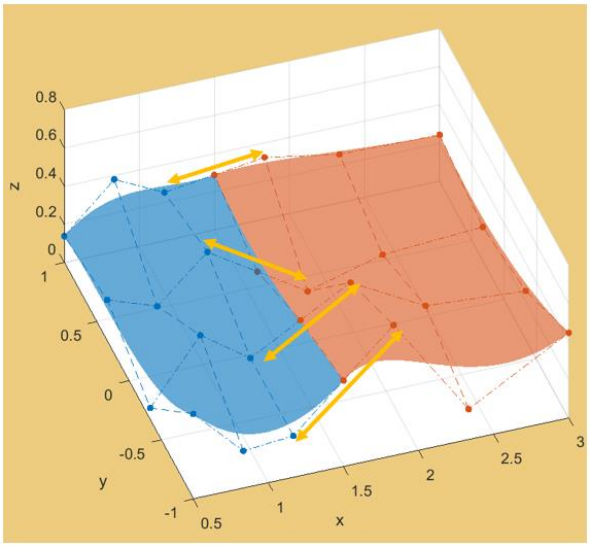


Figure 10 - Bezier patch continuity first solution

Second Solution:

To obtain more freedom in constructing surfaces, it is suggested to take

$$\mathbf{x}^{B}_{u}(0,v) = \lambda(v)\mathbf{x}_{u}{}^{A}(1,v) + \mu(v)\mathbf{x}^{A}_{v}(1,v)$$

Solving the equation shows that λ must be a constant, and μ may be linear in v .The resulting is a stretch in the v direction.

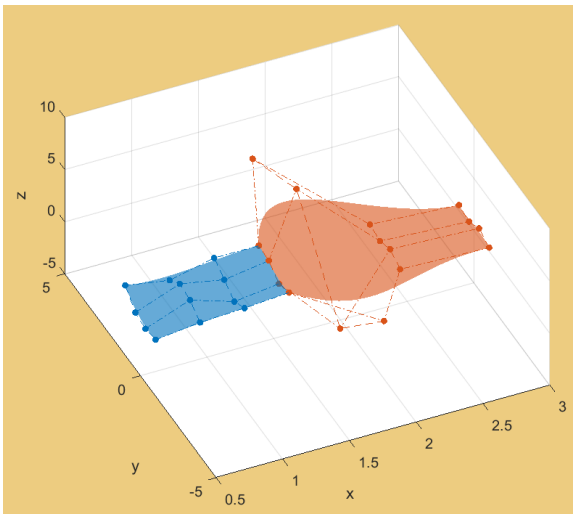


Figure 11 - Bezier patch continuity second solution

2.5 Reconstructing Free-Form Objects

2.5.1 Cubic Bezier curve least square fitting

In [1] Khan suggests a method for fitting a cubic Bezier curve to a data set p_i using the least squares equation for a cubic Bezier curve $q(t_i)$ of control points P_k and running parameter t. The least squares equation can be written as such:

$$S=\sum_{i=1}^{n}\left(p_i-q(t_i)\right)^2=\sum_{i=1}^{n}\left(p_i-\left(\left(1-t_i\right)^3P_0+3t_i\left(1-t_i\right)^2P_1+3t_i^{\,2}\left(1-t_i\right)P_2+t_i^{\,3}P_3\right)\right)^2$$

Determining $P_0=p_{i=1}$ and $P_3=p_{i=n}$ the following equations can produce P_1,P_2 :

$$\frac{\partial S}{\partial P_1}=0\quad\frac{\partial S}{\partial P_2}=0$$

If one wants to expand the system for composite Bezier curves, Khan suggests using break and fit strategy. The data is split iteratively into segments, for which the fitting method above is used. The number of iterations is decided by the following criteria: If the fitting method above fails to uphold the determined threshold for S in a certain segment, that segment data point is then split in two to create two segments for which to fit.

The method which Khan suggests can't be applied with ease to surface fitting as the number of segments and control points is undetermined a-priori. Furthermore, it would require a pre-processing down sampling algorithm.

2.5.2 Evaluating surface fitting

In [2, p. 656], a method is given for reconstructing free form objects as a tool for reverse engineering. It is assumed, that the data given is triangulated, and represents a composite free-form object, which is decomposable into meaningful surface portions, each formed by one or more patches of an original CAD design. This is not the case in our project, alas, the methodology used is still with insight.

Surface fitting by least mean squares is suggested. That is, assuming that for each data point $\mathbf{p}_i$ there is an associated point on the surface defined by the parameter pair (u_i,v_i), and a scalar weight w_i, surface fitting can be formulated as finding the surface which minimizes the following weighted least squares expression:

$$F(\mathbf{S})=\sum_i\omega_i^{\,2}\left|\mathbf{S}(u_i,v_i)-\mathbf{p}_i\right|^2$$

There are multiple issues with this approach:

- An appropriate parameter values u_i,v_i and weights w_i must be chosen
- Smoothness and accuracy may be inherently opposed.

Still, we would use the least squares method to evaluate our surface against its origins.

2.6 Pseudo Inverse Matrix

The pseudo-inverse matrix provides the least mean squares error solution to a system of linear equations. It is also known as the Moore-Penrose matrix inverse. A system of equations $Ax = y$ can be solved for **x** by using the inverse where A is invertible A^{-1}. This is done by $x = A^{-1}y$. This requires that for matrix A to be invertible the number of rows m must be equal to the number of columns n, or the number of equations is equal to the number of unknowns. If the number of equations m is greater than the number of unknowns n or m>n it is considered to be overdetermined. An overdetermined system is usually, but not always inconsistent. An example where an overdetermined system would be consistent or having an exact solution would be three straight lines which happen to pass through the same point. Since this is unlikely to happen every time a system is overdetermined, the pseudo-inverse of a matrix A is used in order to find the least mean squares solution and is denoted by A^+ This is described as $A^+ = \left(A^T A\right)^{-1} A^T$. The matrix A has a pseudo-inverse if $\left(A^T A\right)^{-1}$ exists. This is true since multiplying A^+ by A and rearranging the equation yields $\left(\left(A^T A\right)^{-1} A^T\right)A = \left(A^T A\right)^{-1} A^T A = 1$.

The least squares solution analogous to the determined inverse solution described earlier is shown below in the equations below. Multiplying both sides of the overdetermined system by the pseudo-inverse conveniently yields the least mean square solution x_m.

$$Ax = y \Rightarrow x = \left[\left(A^T A\right)^{-1} A^T\right] y \Rightarrow \boxed{x = A^+ y}$$

2.7 The Pseudo Inverse Technique

The process described in section 2.4 is the given forward process, which describes finding a parameterized surface $x(u,v)$ with the given control points. The reverse process is to use the Bezier surface to describe the shapes of a set of given graphic data points. This is done by setting the parameters in a matrix U and V, the pseudo inverse of a matrix could be applied to the forward process given in section 2.4 to find the control points P. This is derived in the equations below where P_s are the original graphic data points, U^+ and V^+ are the pseudo inverse matrices, B is the Bezier basis matrix coefficients. P is the resulting control point matrix found in the process below:

$$U=\begin{bmatrix} 0 & 0 & 0 & 1 \\ u_1^3 & u_1^2 & u_1 & 1 \\ u_2^3 & u_2^2 & u_2 & 1 \\ u_3^3 & u_3^2 & u_3 & 1 \\ u_4^3 & u_4^2 & u_4 & 1 \\ u_5^3 & u_5^2 & u_5 & 1 \end{bmatrix}, V=\begin{bmatrix} 0 & 0 & 0 & 1 \\ v_1^3 & v_1^2 & v_1 & 1 \\ v_2^3 & v_2^2 & v_2 & 1 \\ v_3^3 & v_3^2 & v_3 & 1 \\ v_4^3 & v_4^2 & v_4 & 1 \\ v_5^3 & v_5^2 & v_5 & 1 \end{bmatrix}$$

$$P_s = UBPB^TV^T$$

$$U^TP_sV = U^TUBPB^TV^TV$$

$$\left[\left(U^TU\right)^{-1}U^T\right]P_s\left[V\left(V^TV\right)^{-1}\right]^T = \left[\left(U^TU\right)^{-1}\left(U^TU\right)\right]BPB^T\left[\left(V^TV\right)\left(V^TV\right)^{-1}\right]$$

$$U^+P_s\left(V^+\right)^T = BPB^T$$

$$P = \left[\left(B^{-1}\right)\left(U^+\right)\right]P_s\left[\left(V^+\right)^T\left(B^{-1}\right)^T\right]$$

2.8 Principal Component Analysis for Rigid Body Transformations

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components.

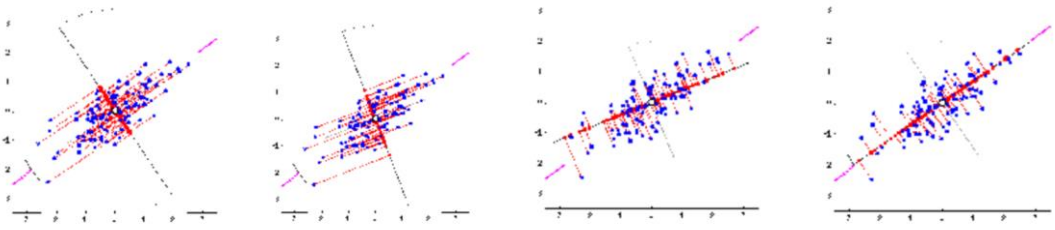


Figure 12 - Sample for PCA iteration

2.9 CAD Formats

The following is taken almost in its entirety from [4].

Any industrial part modeled in a commercial CAD software is, by default, its NURBS

representation. Non-Uniform Rational B-Spline (NURBS) is a standard mathematical representation of any parametric surface. Flexibility, affine transformations and ability to access local control are some of the properties that make NURBS a popular choice for representing industrial parts.

CAD formats are specific to CAD software. A CAD part modeled on a CAD software may not be compatible with the others.
Initial Graphics Exchange Specification (IGES) is an industry accepted neutral file format that is compatible on various CAD platforms. Smith et al. developed this format to enable convenient data exchange of CAD models. The format stores the mathematical, geometrical, topological and other nongeometric details of a product.

However, a simpler file format, namely, Stereolithography (STL) acts as an input to AM (additive manufacturing) machines.

A STL file can be exported directly from its CAD model in the CAD software. Tessellation of a CAD object into a series of planar triangles results in its corresponding STL file. STL files are industry accepted input formats to AM machines. The file can be represented in two formats – Binary and ASCII.

However, both the formats describe the CAD model with coordinates of vertices of facets and their corresponding normal.

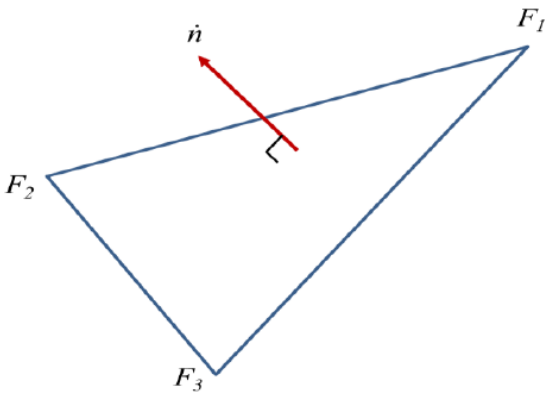


Figure 13 - A planar STL facet represented by vertices and outward normal

A sample STL tessellation method is shown in Figure 5 and Figure 6. The translation from CAD to STL format causes errors. The areas of STL file format errors and STL modification have received considerable amount of attention from researchers.

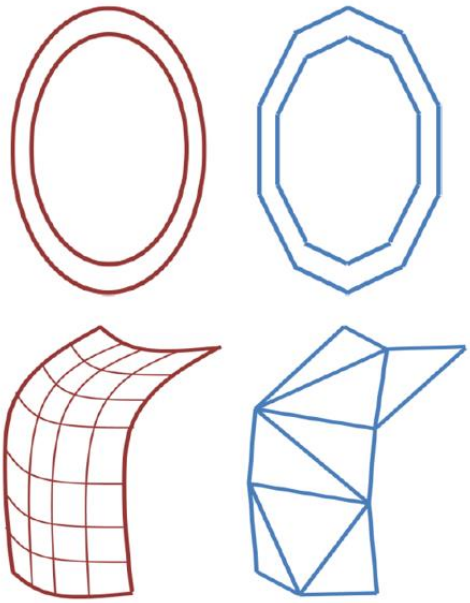


Figure 14 - NURBS curve and surface and their tessellated form

American Society for Testing and Material (ASTM) published an alternative format called the

Additive Manufacturing File (AMF) format. This format represents the design surface using curved triangles by utilizing second order Hermite curves. In addition, this format also includes information about the material properties, color and texture. However, these curved patches are converted back into planar triangles during slicing which would again lead to CAD to AMF translation errors.

2.10 Surface curvature

Curvature of surface is A non-intuitive term, because there is infinite different curvature for any point on a surface. As in fig 15, there are infinite direction where one can "Walk on the surface", and for each direction there is different curvature.
In order to quantify this term, there is an agreement on the definition of Surface curvature.

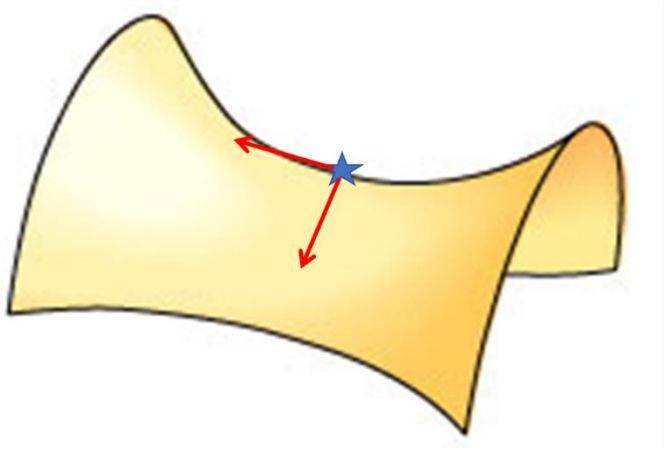


Figure 15 – example of major curvatures on surface

In order to determine the surface curvature, here some definitions:

$$\text{First fondamental form} = \begin{bmatrix} \underbrace{\frac{\partial^2 r}{\partial u^2}}_{E} & \underbrace{\frac{\partial r}{\partial u}\cdot\frac{\partial r}{\partial v}}_{F} \\ \frac{\partial r}{\partial u}\cdot\frac{\partial r}{\partial v} & \underbrace{\frac{\partial^2 r}{\partial v^2}}_{G} \end{bmatrix}$$

$$\text{Second fondamental form} = \begin{bmatrix} \underbrace{n\cdot\frac{\partial^2 r}{\partial u^2}}_{L} & \underbrace{n\cdot\frac{\partial r}{\partial u}\cdot\frac{\partial r}{\partial v}}_{M} \\ n\cdot\frac{\partial r}{\partial u}\cdot\frac{\partial r}{\partial v} & \underbrace{n\cdot\frac{\partial^2 r}{\partial v^2}}_{N} \end{bmatrix}$$

Explanation in detail about first and second fundamental form can be found in the relevant sources[1].

Another definition is for the two major curvatures k_1, k_2. The two are the maximal and minimal curvatures. The two are represented schematically in Fig 15.

k_1, k_2 are derived from the following equations:

$$k_1 \cdot k_2 = \frac{LN - M^2}{EG - F^2} \;;\; k_1 + k_2 = \frac{NE - 2MF + LG}{EG - F^2}$$

In differential geometry there are two options for surface curvature definition:

Gaussian Curvature: $\left[K = k_1 \cdot k_2\right]$

Mean Curvature: $\left[H = \frac{1}{2}\left(k_1 + k_2\right)\right]$

In the algorithm testing functions both options are available.

2.11 Hausdorff distance

Hausdorff distance from set A to set B is a maximin function, defined as:

$$h(A,B) = Max\left[Min\left(d\left(\underset{a\in A}{a}\ ,\ \underset{b\in B}{b}\right)\right)\right]$$

[1] https://en.wikipedia.org/wiki/First_fundamental_form ,
https://en.wikipedia.org/wiki/Second_fundamental_form

where a and b are points of sets A and B respectively, and $d(a,b)$ is any metric between these points.

for simplicity, we'll take $d(a,b)$ as the Euclidian distance between a and b.

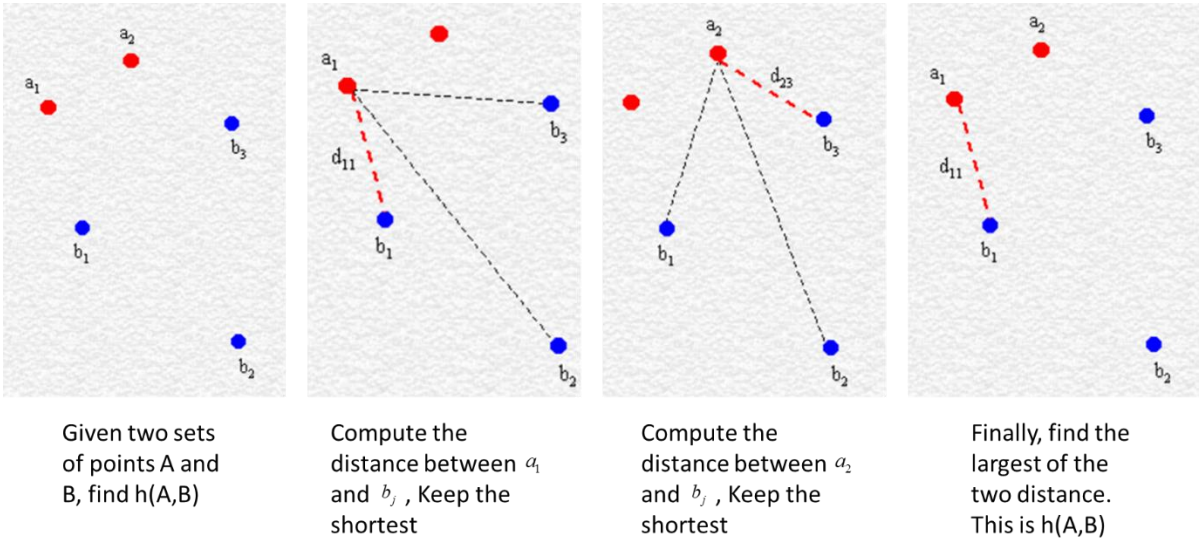


Figure 16 - Hausdorff distance on point sets.

It should be noted that Hausdorff distance is oriented (we could say asymmetric as well), which means that most of times $h(A,B)$ is not equal to $h(B,A)$. This general condition also holds for the example of Fig 16, as $h(A,B)=d(a_1,b_1)$, while $h(B,A)=d(b_2,a_1)$. This asymmetry is a property of maximin functions, while minimin functions are symmetric.

For conclusion, more general definition of Hausdorff distance would be:

$$H(A,B)=Max\left[h(A,B),h(B,A)\right]$$

3 Solution approach

We are to form the solution to the following: Reconstructing a manageable parametric surface from a point cloud of a stump obtained by a single RealSense camera for a commercial CAD program.

Our proposed solution framework consists of a seven-step process as illustrated in figure 17.

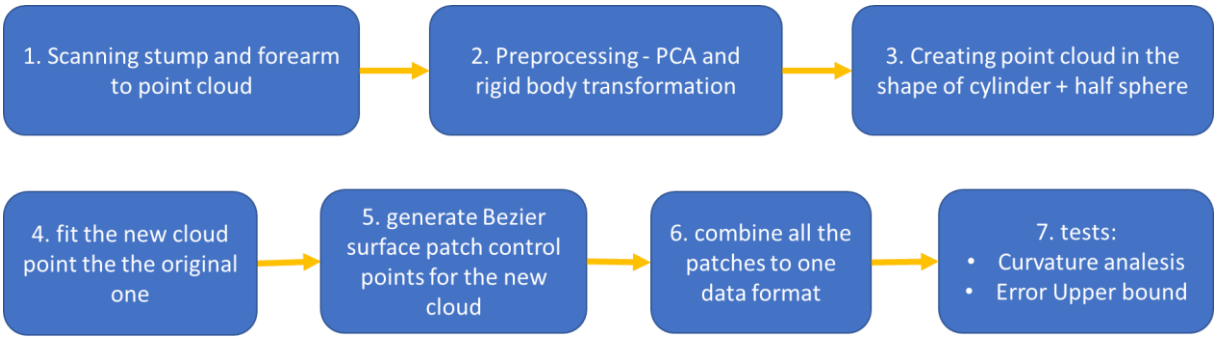


Figure 17 - Algorithm process flowchart

3.1 Scanning the stump:

We were given a 3D point cloud procured with a 3D RealSense camera mounted atop a dedicated scanner.

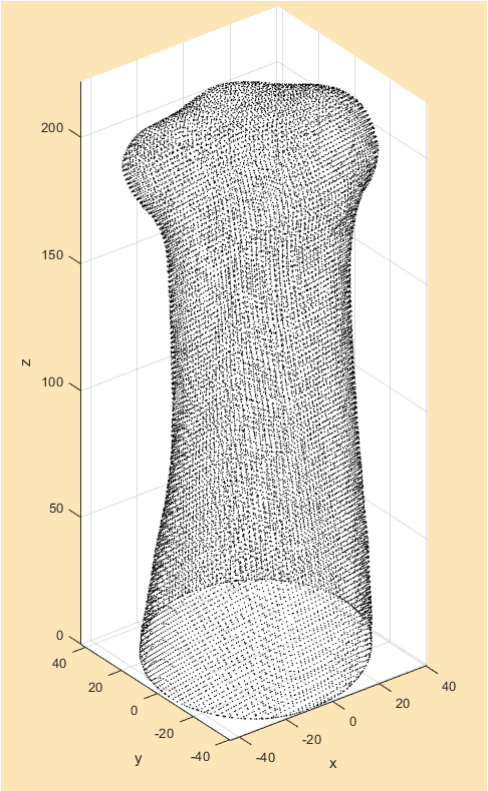


Figure 18 - Forearm stump cloud point

3.2 Rigid body transformations – "Z transformation"

As a limb has a primary axis, we decided to build an algorithm specific to its geometry. The algorithm assumes that the base of the limb is in z=0 and that its primary axis is the **z** axis. To enforce those conditions on the cloud point, we used PCA and rigid body transformations.

An example of our "Z transformation" in effect on a synthetic cylinder:

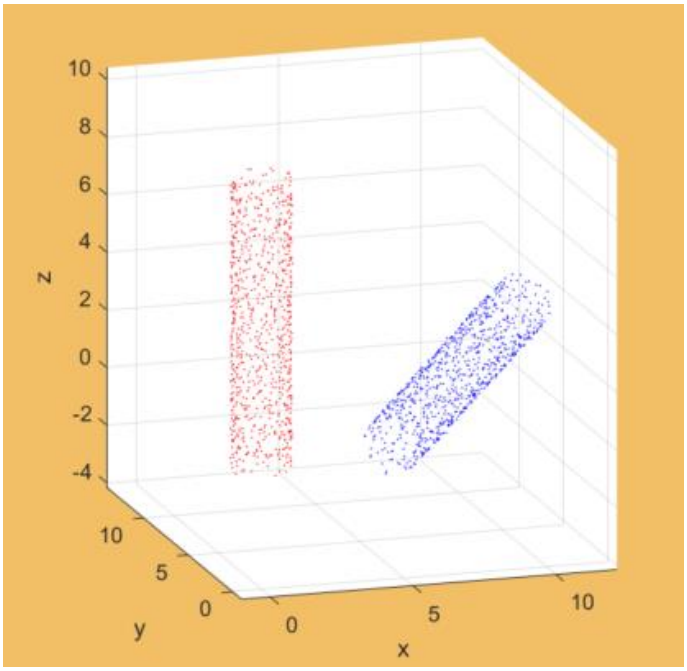


Figure 19 - Z Transformation for a Cylinder

3.3 Creating point cloud in the shape of cylinder + half sphere:

The physiology of the limb allows for division to two parts: forearm and stump – each with its own geometry characteristics.

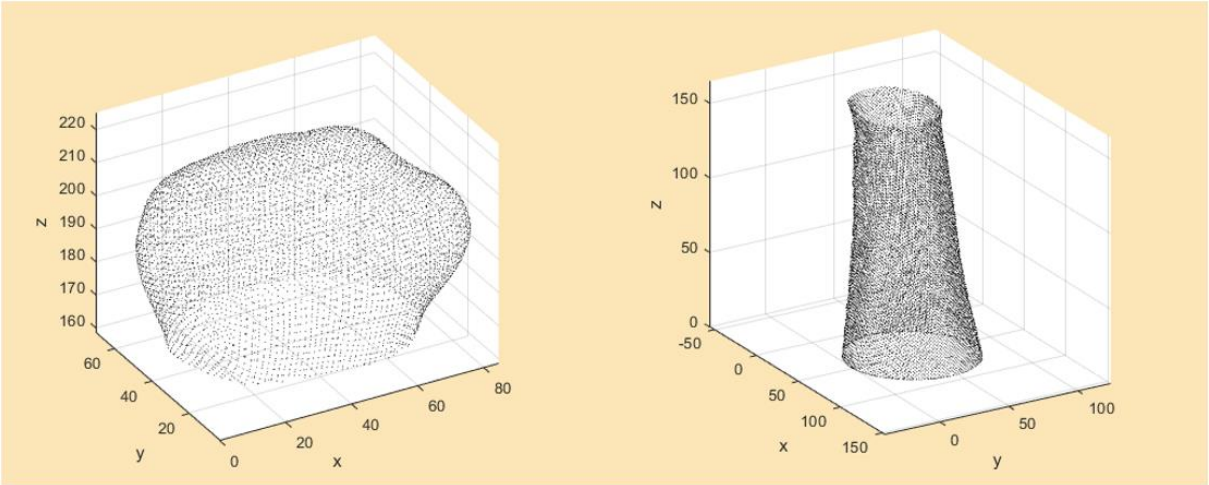


Figure 20 - division to two parts: forearm and stump

To simplify our model when building it, we used a cylinder and a sphere as stand-ins for the forearm and stump.

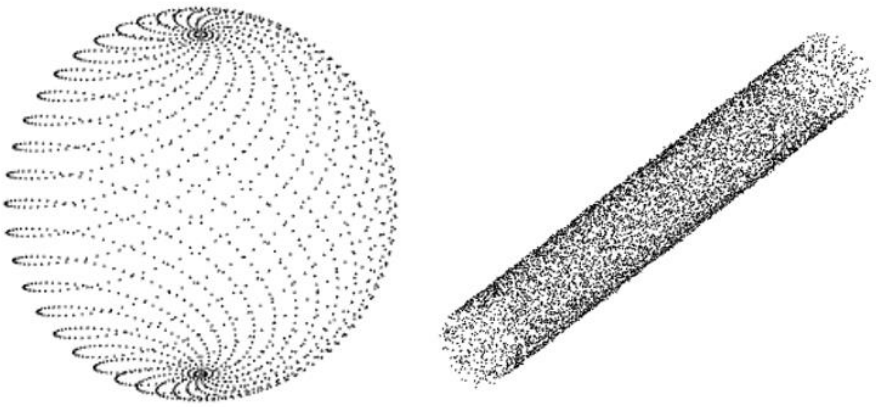


Figure 21 – Synthetic Cylinder and Sphere With built-in noise

In order to produce an initial cloud that reminiscent of the original shape, we extracted two parameters from the geometric structure –

- R – the radius of the shape from top view
- L – the length of the shape form side view

We create a shape of cylinder with half sphere on top:

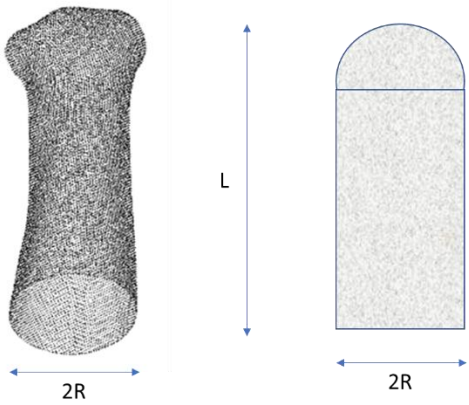


Figure 22 – demonstration of thecreation the new point cloud from the extracted R L

The amount and order of Bezier patches determines minimum the number of points in the new cloud.

For the cylindrical parametrization, the following holds true:

$$N = \underbrace{(O_{Bez}+1)}_{\text{first patch}} + \underbrace{(n_\theta - 2)\cdot O_{Bez}}_{\text{middle patches}} + \underbrace{(O_{Bez}-1)}_{\text{stitching patch}}$$

$$\rightarrow \boxed{N = n_\theta \cdot O_{Bez}}$$

$$M = \underbrace{(O_{Bez}+1)}_{\text{first patch}} + \underbrace{(n_z - 1)\cdot O_{Bez}}_{\text{middle patches}}$$

$$\rightarrow \boxed{M = n_z \cdot O_{Bez} + 1}$$

N –points in θ dimension; M –points in z dimension

n_θ - patches in θ dimension; n_z - patches in z dimension

O_{Bez} - Bezier surface order (both parameter directions)

Th last step in the process is to generate Bezier surface patch control points for the new cloud. After a few checks we got to conclusion that this amount of points is the required minimum and that the surfaces don't change so much for larger amount of points.

3.4 Fitting the new point cloud to the original one

in this step we optimize the new cloud to fit the old one. All points in the cloud can move along one line, the initial radius defining them, as illustrated below:

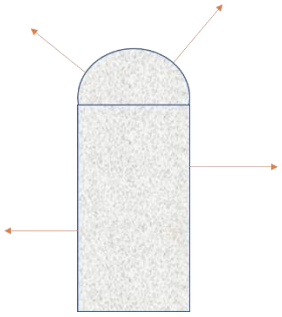


Figure 23 – illustration of the guideline of each point in the optimization

In this process we used the MATLAB function "fmincon". The optimization is about the minimum distance between the two clouds. we check the minimum distance for each point in the new cloud from the closest point in the old cloud. The min distance in the cost function is the optimal fit

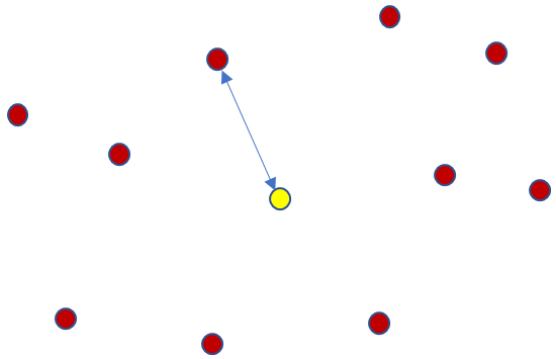


Figure 24 – minimum distance illustration

This new cloud represents the Bezier surface patches, and from this point we will use only the new cloud.

3.5 generate Bezier surface patch control points for the new cloud

The next step is to create the control points that's defines the Bezier surface patches. The problem we face is the inverse problem as presented in the theoretical background (section 2.6 and 2.7). Extraction of control points by the algorithm is done for each patch. At this point, the user have enough information for a injective definition of the patches.

3.6 Data format

Saving the control points and their connections will suffice as an input for a CAD program which can reconstruct the surfaces from them.

Inspired by the STL format which saves vertices, faces and normal vectors, we decided upon a struct format we named "Control Points" which goes as follows:

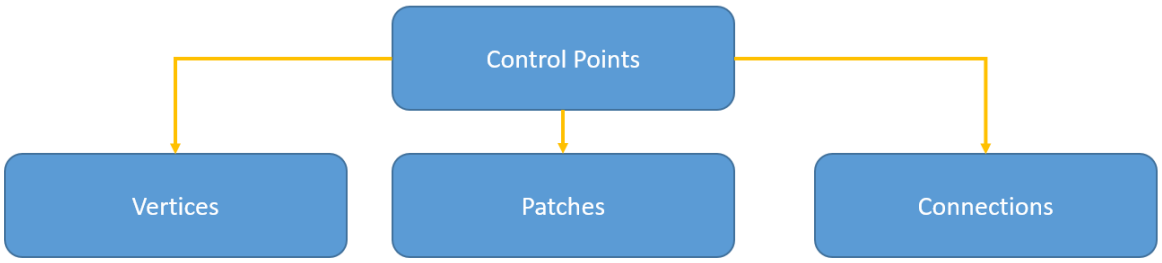


Figure 25 - Control point data format

Vertices – $m \times 3$ of real numeric matrix containing $[x, y, z]$ data for each point $i = 1, ..., m$

Patches - $(O_{Bez} + 1) \times (O_{Bez} + 1) \times$ Patch Amount of indices. Each patch is represented in a matrix which holds indices $i = 1, ..., m$ representing the vertices in the Vertices matrix.

Connections - Patch Amount $\times 4$ matrix of indices. Each row i in the matrix correlates to patch number i . The four columns of the matrix represent the neighbors of each surface - $[Left, Right, Up, Down]$ and hold patch index numbers.

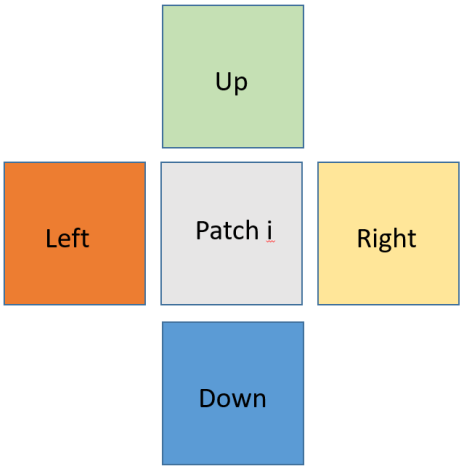


Figure 26 – Connections Matrix

If a surface has no neighbor in a certain direction, the value corresponding is zero (relevant for the edges).

3.7 Tests

In order to check the quality of the results obtained, two tests are performed. One is about the surface curvature and the other about the fitting error.

3.7.1 Curvature analysis

As described in section 2.10, the two definitions for surface curvature are "Mean" and "Gaussian", both have been taken from MATHWORKS library[2]. In the algorithm there is the option to calculate the curvature in both ways. in order to check the algorithm, it have been tested on a cylinder:

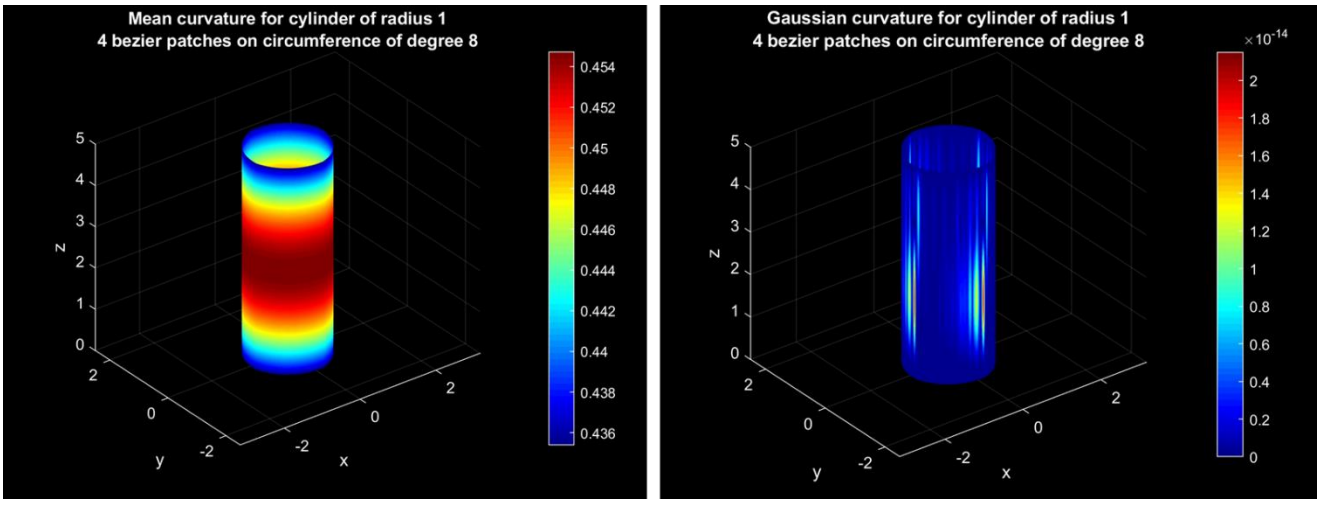


Figure 27 – test for the curvature calculation algorithm on a cylinder

[2] https://www.mathworks.com/matlabcentral/fileexchange/11168-surface-curvature

The results are satisfied. as expected, the curvature for the mean calculation needs to be $\frac{1+0}{2}=0.5$, and for the Gaussian needs to be $1\cdot 0=0$.

3.7.2 Error upper bond

In order to test the results, Quantifying the distance between the scanned cloud and the obtained surfaces will satisfy. The quantification is for an upper bond. The calculation for this bond is done by the Hausdorff distance, As described in section 2.11. in the algorithm, the user can have a look at a colormap of the error:

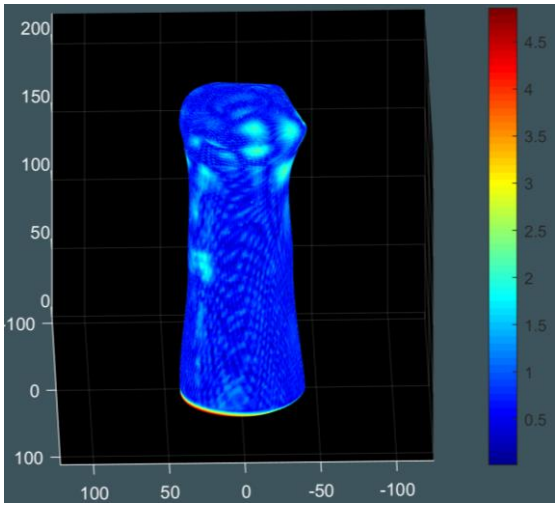


Figure 28 – colormap of the error, calculated as Hausdorff distance

4 Results

4.1 CAD model

We have extracted a point cloud from an STL file created from a scan of forearm stump and applied our algorithm to it. The STL file was given to us by a member of the BRML lab of the faculty of mechanical engineering – Technion.

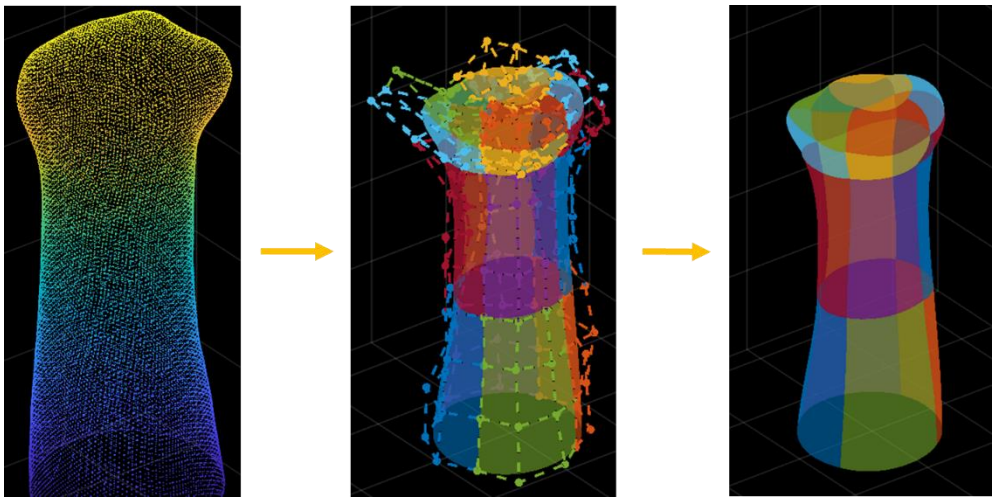


Figure 29 - Algorithm results with the control points

At this point, the missing step was to transform the control points data into a CAD format. the chosen format was IJS. In order to do so, we used the program of professor Gershon Elber called "IRIT". The program can take the control points and instantly transform it into IGS file, which most of the commercial cad programs can open easily.

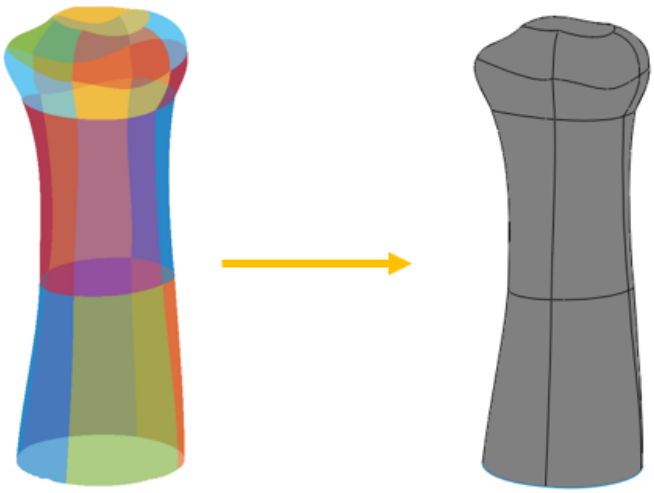


Figure 30 - illustration of the transition from MATLAB data to CAD file.

Finally, one can take the IGS file and obtain it's negative using most commercial CAD programs.

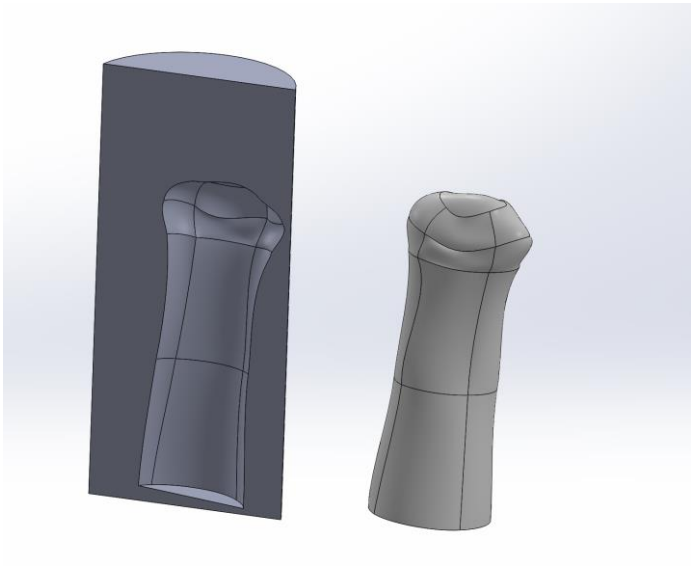


Figure 31 – illustration, one step in SOLIDWORKS can create the negative of the stump.

4.2 GUI

Another product of this project is the algorithm GUI. All the steps that have been described in detail in section 4 are concentrated in a few clicks. it was built in MATLAB:

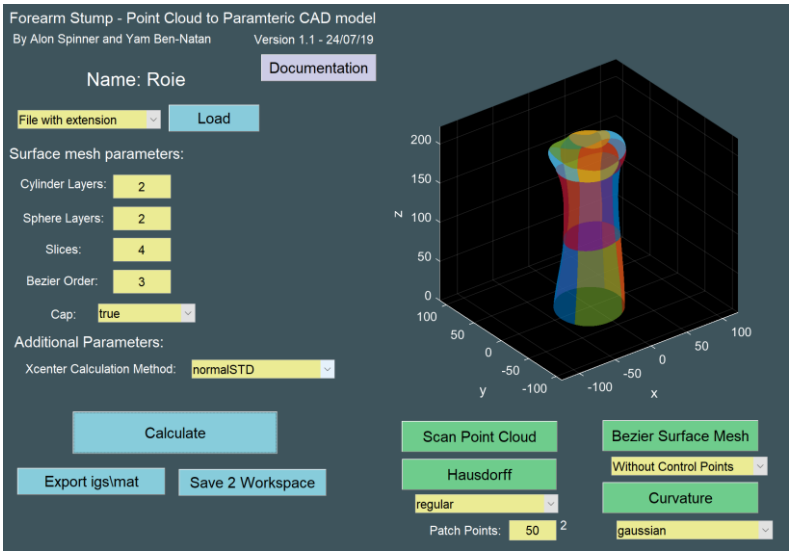


Figure 32 – GUI illustration

5 Summary

In our goal to represent a scanned point cloud of a forearm stump using parametric surfaces we chose to use a net of Bezier patches. The Bezier surfaces usually intended for design work were decided on because they are simple mathematically and can represent convex shapes accurately. To understand the properties and qualities of these surfaces we had gone through researching Bezier curves in 2D and 3D that are a directly connected to the Bezier surfaces.

After deciding on the parametric surface approach, it would turn out that fitting the Bezier surface mesh to the stump's point cloud was no simple task.
We had first used PCA and rigid body transformation to place the point cloud into a conventional frame of reference where the forearm's base is at $z = 0$.

We then found a specific plane inside the convex shape of the point cloud, which splits its geometry to a sphere and a cylinder. We had tried multiple ways of calculating the dividing plane's location and the most successful one would be that which uses the maximal change of the standard deviation of a trimesh's normal of the scan point cloud.

Establishing a sphere and a cylinder around the scanned point cloud, we applied an optimization process which minimizes the summation of the Euclidian distance between the sphere-cylinder and the point cloud, allowing for movement of sphere-cylinder points in a radial direction. This is in fact a non-rigid body transformation.

With points who have parametric representation now lying in the scan point cloud, we are now able to fit parametric surfaces to them. We used the pseudo-inverse technique to establish Bezier control points. We forced geometrical continuity G_0 between each two surfaces using the mean function on their same-edge control points.

To conclude our work, we mapped a visual one-sided Hausdorff distance measurements between the parametric surface net and the scan point cloud, and a curvature map of both gaussian and mean curvatures.

This proposed algorithm or a variation of it will be integrated into an automated pipeline that creates forearm prosthesis from a 3D scanned stump to be used for the E-Nable nonprofit organization.

Conclusions and Future Research:

We are pleased with the general algorithmic pipeline. The computation is quick (less than 5 minutes), the data structures are sound and malleable, and the results show promise.

Yet for the future, and before much else, it would be wise to procure multiple forearm stump scans to run tests on. We would suggest testing for curvature and errors of the generated surfaces with respect to their amount.
New error and distance functions should be thought of to allow good judgment of the surfaces net produced.
With new error functions, one could devise an optimization process determining the degree of the Bezier surfaces used and their amount with correspondence to the error.
The Bezier surface net procured by our algorithm does represent the stump's geometry with canny likeness and generally low Euclidian errors, yet it would not prove suitable for material-geometry analysis. Hence, if that is what one aims to suggest, we would suggest moving to more advanced method of interpolation using B-spline surfaces or NURBS.

6 Bibliography

[1] P. Z. G. M. C. Carlo Dal Mutto, Time of Flight Cameras and Microsoft Kinect, Springer.

[2] M. I.D.Faux, Computational Geometry for Design and Manufacture, Chichester, West Sussex: Chichester, 1987.

[3] M. Khan, Approximation of data using cubic-Bezier curve least square fitting.

[4] H. K. Farin, Handbook of Computer Aided Geometric Design, North-Holland, 2002.

[5] S. Allavarapu, A New Additive Manufacturing (AM) File Format Using, Cincinnati, 2013.

[6] M. M. S. M. P. Z. A. B. S. G. E. M. S. Zennaro, PERFORMANCE EVALUATION OF THE 1ST AND 2ND GENERATION KINECT FOR, University of Padova, Italy.