Design of an Alternative Method to Create Custom Ocular Prosthetics

A Technical Report submitted to the Department of Biomedical Engineering

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Raina Danielle Mourad

Spring, 2022

Technical Project Team Member

Jaden G. Stanford

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

William Guilford, Department of Biomedical Engineering

Design of an Alternative Method to Create Custom Ocular Prosthetics

Authors: Jaden Stanford and Raina Mourad

Word count: 5877
Figures: 8
Equations: 2
Supplementary Figures: 10
References: 33

William H. Guilford
May 11, 2022

Design of an Alternative Method to Create Custom Ocular Prosthetics

Jaden G. Stanford. Raina D. Mourad.

Abstract

Patients wear ocular prosthetics following the removal of an eye. While prosthetics do not restore vision, they protect the socket from infection and improve quality of life by restoring the individual's natural appearance. Custom prosthetics are preferred because they have superior fit, comfort, and appearance; however, they are much less affordable than generically-produced stock prosthetics. In addition, an ocularist must take a mold impression of the socket for a custom prosthesis, which is uncomfortable and sometimes painful for patients. This report details the design of an alternative method to create custom ocular prosthetics to replace traditional ocularist practices. By using photogrammetry, this technique eliminates the need for the mold impression step and reduces resource use. The steps in the method include photogrammetry to image the face, 3D mesh generation, mesh modification, and Boolean subtraction to deduce the shape of the prosthesis. The method relies on the assumption that the shapes of the natural and lost eye can be compared to deduce the required prosthesis dimensions. The researchers validated this assumption by comparing an individual's two eyes, finding that their average difference in thickness was 0.50 mm. The researchers optimized each step of the aforementioned method and created a written manual of the process so that it could be used by others. This method was developed primarily through testing with a computer-generated (CGI) model of a human female head and a real model of a human male head; they were both artificially enucleated using a Gaussian function with known dimensions. To measure accuracy and precision, the method was repeated for the CGI model five times. The average root mean square error between the thickness of the 5 prosthetics and the known dimensions was 0.43 +/- 0.05 mm, which is promising for initial tests.

Keywords: ocular prosthetics, ocularist, 3D technology, 3D modeling, photogrammetry, Boolean subtraction

Introduction

Significance

An estimated 5 million people worldwide wear prosthetic eyes following surgical removal (enucleation) of an eye[1]. Eye enucleation is often unexpected and distressing; the most frequent cause is traumatic injury (commonly work-related), followed by ocular diseases, tumors, and malformations[2]. Once the eye is removed, a surgeon will place a ball implant into the socket to prevent drooping and maintain facial symmetry. Although the globe of the eye is removed and the optic nerve is severed, the muscles that control eye movement are left intact[3]. Eventually, when the implant and subsequent prosthesis rest on top of the eye muscles, they will move in the appropriate direction with the contralateral eye, but the movement appears slower and slightly out of sync. A cross-sectional image of the implant and prosthesis is shown in Figure 1.

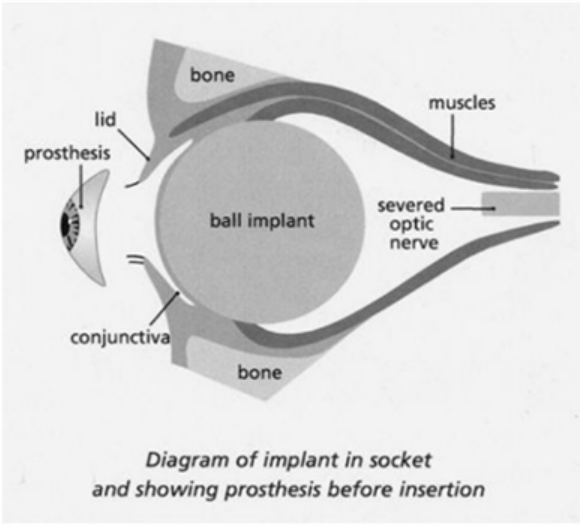


Fig. 1. Cross-sectional diagram of the placement of an ocular prosthesis and orbital implant[4]

Six to eight weeks post-surgery, a patient can be fitted for an ocular prosthesis, which will lie slightly under the lids, similar to a contact lens. Unlike the ball implant, the ocular prosthesis can be removed and reinserted at will. Although ocular prosthetics do not function to restore vision, they help to restore natural appearance. This is important for the well-being of anopthalmic patients, as the loss of an eye is a life-changing event and can lead to depression, anxiety, and overall reduced quality of life[5]. In addition, eyes are the first part of a face that humans are drawn to. In studies on infants' gazing patterns, the results show a clear preference for human eyes, emphasizing the significance of the loss of such a vital part of the face[6]. While there are mass-produced, generic (stock) ocular prosthetics, custom-made ocular prosthetics are preferable due to their improved aesthetic appearance and comfort[7]. Stock prostheses also have an increased risk of medical complications due to the collection of socket secretion behind the prosthesis and in front of the enucleated socket, which results from the imperfect fit between the concave backing of the prosthesis and the shape of the socket[7].

Although they are preferred to stock prostheses, current custom ocular prosthetics have limited accessibility for both patients and ocularists who create them. The price of a custom prosthesis ranges from $2,500 - $8,300, excluding the cost of surgery for eye removal[8]. Conversely, stock prosthetics cost around $15 and do not require an extensive fitting process. The low number of practicing ocularists also limits the number

of prostheses that can be made, as the creation of a custom ocular prosthesis requires significant resources and time. For reference, there are only 5 practicing ocularists in Virginia[9].

The current custom ocular prosthesis fabrication technique is based on dental procedures, which presents a number of challenges.[10] The initial step in custom prosthesis creation involves injecting a silicone-based material into the enucleated socket to make an impression, which is then left to harden and later extracted.[11] This process is invasive, uncomfortable for patients, and the impression material can be traumatic for the tissue; in most cases patients have undergone enucleation surgery as soon as 1 ½ months prior[8]. Ocularists use the impression to create a wax mold, which they then use to create the acrylic prosthesis. Next, they modify the prosthesis through smoothing and polishing, and hand paint it to make its appearance more realistic. Along with the process being painful for patients and inefficient, the impression technique can also be inaccurate. Because the material has to be pressed firmly into the socket, the folds and wrinkles of the tissue may be flattened and the socket may be overfilled, losing detail[12]. Also, since the impression material is only in the socket for a short period of time, the orbicularis muscles do not have time to relax after their initial contraction in response to the foreign body, which could result in an inaccurate shape. Finally, the impression material does not provide any information about the anterior shape or size of the eyeball.[12]

Furthermore, adult prosthetic eyes should be replaced every 5 years, while children who are growing need to have their prosthesis examined and potentially remade every 6 months[5]. According to a New Zealand study, the most common age group for eye loss is 1-9 years old, meaning a large number of the users of prosthetic eyes have theirs replaced on a regular and frequent basis[1]. There are also many other reasons why a prosthesis would need premature replacement, such as complications that can arise from allergic reactions, conjunctivitis, bacterial infections, post-enucleation socket syndrome, and other conditions.[5] Consequently, this presents a huge financial and psychological burden for the patient.

The researchers aim to address the problems with the current design approach by developing a more efficient, cheaper, and painless technique to create an ocular prosthesis that would completely eliminate the need for hydrocolloid impression materials. Rather, topographical information about the empty socket such as the volume of the cavity and fornices will be obtained using photogrammetry. Photogrammetry is a non-invasive scanning approach that generates 3D mesh models from a set of 2D images. Unlike the current process which requires specialized skills and is inaccessible to the average physician, photogrammetry is simple to learn, available in free software packages, and can be used with any digital camera. In addition to reducing discomfort, the method will also require fewer clinical visits, making the process of obtaining a custom prosthesis much faster. The novel process will reduce the costs associated with creating the prosthetic by requiring minimal resources, making it more accessible for individuals unable to afford custom prostheses. Once the desired mesh model is generated using photogrammetry and modified, the shape of the prosthesis will be determined by taking a Boolean difference between the enucleated and natural eye. The consequence of continuing to employ current methods is a drastic loss of time and money, and more importantly, enduring pain and discomfort for patients.

Innovation & Prior Art

There has been limited success in previous attempts to design a suitable method for creating an ocular prosthesis using noninvasive, inexpensive, and efficient techniques. However, several groups have attempted to advance the process digitally printing the iris or by 3D printing a prosthesis based on a variety of imaging methods. Although both of these methods offered improvements to current methods of ocular prosthesis creation, they have significant limitations.

Printing the iris

Although there is no available method for digitally modeling a complete ocular prosthesis, digital imaging technology has been used to improve the aesthetic quality of ocular prosthetics, mainly through better replication of the iris and sclera[13]. In the traditional method of creating an ocular prosthesis, the intricate details of the iris and veins of the sclera must be hand-painted. This is non-ideal because the aesthetic quality of the prosthesis is highly dependent on the ocularist's painting ability; this is also a time-consuming process. Digital imaging eliminates human error that could impact the appearance of the iris[14]. In addition, while hand-painting the prosthetics produces good results, digital photography offers greater color calibration technology and a standardized process for obtaining accurate images[13].

Multiple groups have explored techniques to digitally print the iris. In one technique, a digital camera with a macro lens and ring flash attachment was used to photograph images of a patient's contralateral iris[13]. Using graphics software, differences in color hue or contrast that occurred during the image process were adjusted to match the true color. After, the iris was printed on white paper using a laser printer, and attached to an ocular prosthesis using monopoly syrup and a coat of sealant painter's spray[15].

Although this innovation was successful, it only addresses aesthetics of the prosthesis, and does not significantly improve accessibility and price issues. It also does not reduce patient discomfort by adjusting the mold cast process of obtaining eye shape. The proposed technique in this paper will use the advantages of digital image processing in similar ways, but focusing on the actual modeling of the shape of the prosthetic. It is possible that printing the iris could eventually be used in conjunction with the proposed photogrammetry process, to further streamline the process of ocular prosthesis creation using digital imaging in every step.

3D modeling and printing an ocular prosthesis

3D modeling in ocular prosthesis creation has been explored previously. One method called cone beam computed tomography (CBCT)[16] was successful in generating a 3D computer model of the anophthalmic cavity in an elderly patient. The cavity was digitally delineated with segmentation and dilation of the outline, and the anterior curve of the model was calculated from standard values of normal eyes. Afterwards, the design was 3D printed using resin[16]. Although this method was successful in generating an ocular prosthesis using digital models and 3D printing, the major limitation is that CBCT uses X-ray exposure. Although CBCT uses a lower dose of radiation than conventional computed tomography (CT), it still cannot be used safely in children and pregnant women[16]. Considering the most common age for eye loss is 1-9[1], CBCT would not be a viable option to use consistently and safely in the creation of ocular prosthetics.

Other imaging techniques have been used to obtain 3D model data, such as CT or magnetic resonance (MR) imaging; however, since these methods are intended for clinical use, special software is needed to obtain the data and use the images for 3D modeling. Another imaging technique using light intensity 3D scanners reflects laser beams from the surface of objects to obtain 3D model data[17]; however, this is also inaccessible due to the necessity of a light intensity 3D scanner and

other specialized technology. Previous innovation in the use of 3D printing in ocular prosthetics has led to important developments, the most important being the recognition of the mold cast process as unnecessary, uncomfortable, and time consuming since it can lead to inaccurate impressions[16]. 3D printing has been validated through these studies as a valid non-contact way of creating ocular prosthetics; however, all of the aforementioned methods require highly specialized technology, and would require skilled experts. They are also expensive, as CBCT scanning requires costly equipment and is priced around $300 per scan[18]. There is still a need for a technique in creating ocular prosthetics that uses 3D printing, but also an accessible and affordable method of digital imaging. The 3D modeling method proposed in this paper plans to meet this need.

Design Constraints and Criteria

By considering the issues that arise from current ocularist methods in combination with prior art, a set of design constraints and criteria was developed for this method. The criteria given highest priority were dimensional accuracy, biocompatibility/safety, comfortable fit of the prosthesis, and patient comfort during the process (Table S1). In order to achieve dimensional accuracy, the researchers wanted to ensure that the thickness of prosthetics created using the novel method were within acceptable limits of the thickness of currently used prosthetics. The prosthesis' horizontal length and vertical height will vary based on patient differences in face size and shape; however, the average size of a prosthesis was used as a baseline to confirm the prostheses created using the new method are similar to that of the current process (Supplemental Table 2). To address the biocompatibility criteria, the custom ocular prosthesis was planned to be 3D printed in biocompatible acrylic resin from Formlabs[19]. However, future research is necessary to thoroughly evaluate the safety of the material on a mucosal membrane such as the eye socket. Since the focus of the current research is to validate that the method successfully produces an accurate prosthesis model, working with patients and testing biocompatibility is outside the scope of this paper, but will be considered in the future. Furthermore, the researchers tested the accuracy of the prosthetics created using the new method, as this would be important for comfort. A prosthesis that effectively conforms to the crevices and shape of the socket will be a comfortable fit for an anophthalmic patient. In the novel method, the prosthesis is lightly smoothened before 3D printing, which removes any jagged edges or sharp corners that may have resulted from Boolean subtraction. Next, patient comfort during the process was considered. Photogrammetry was chosen as the 3D scanning tool since it is a non-contact imaging method and does not require the use of radiation-based scanning devices. The benefit of this is two-fold. First, photogrammetry ensures that the method is painless for patients and ocularists can forgo the use of impression materials. Secondly, the method is safe since there is no need for harmful exposure to radiation. Since growing children may need to replace their prosthesis multiple times a year, avoiding radiation and reducing the cost was a priority for the researchers.

The secondary criteria the researchers considered are the efficiency, accessibility, and financial cost of the method. Because the current method requires multiple clinical visits and the use of disposable single-use materials, the researchers sought to improve upon this by minimizing the computational time and using as little disposable resources as possible. Although the prosthesis generated using photogrammetry and 3D modeling will still need to be hand-painted by an ocularist, there is no need for impression materials. The method can also be performed in 1-2 clinical visits and computational times for the mesh generation are only about 1 hour. After mesh generation, the model editing and 3D printing may take up to 2 days. Additionally, in order to be implemented into ocularists' clinics, the researchers attempted to design the method to be accessible to anyone, without needing background knowledge about 3D modeling. In comparison to previous work that incorporated 3D imaging and modeling into ocularistry, the method discussed here does not require large, expensive machinery that may be difficult to operate. Rather, photogrammetry can be performed using an iPhone camera and the modeling software is highly user-friendly. Minimizing the financial cost of the method was another important criterion. Since the current method costs patients up to $8,300 at each fitting, patients unable to afford these prices may be forced to purchase a stock prosthesis, which has more health risks and is less aesthetically pleasing[20]. In the new method, the chosen photogrammetry software, RealityCapture, costs $3,750 for an unlimited license to create 3D mesh models, or approximately 50 cents per model without a subscription[21]. The 3D modeling software, Meshmixer, is free and can be downloaded on any Windows operating system[22].

Results

The flowchart in Figure 2 shows an overview of the steps in the proposed alternative method devised by the researchers. For each step, the researchers optimized the process by testing different settings. Included in the supplements is the manual (supplementary material 9) detailing the specific steps and instructions for this method, with detailed explanations and images for each part.

Photogrammetry

The first step in the method is photogrammetry. This involves taking images of the face to generate a 3D mesh rendering. The photogrammetric measurement principle is to acquire many images of an object from different viewpoints and identify common physical object points among multiple photos[23]. From these corresponding points, 3D coordinates of objects can be reconstructed through triangulation. By taking a picture from at least 2 different locations and measuring the same point in each picture, a line of sight is identified from each camera location to the target[24]. If

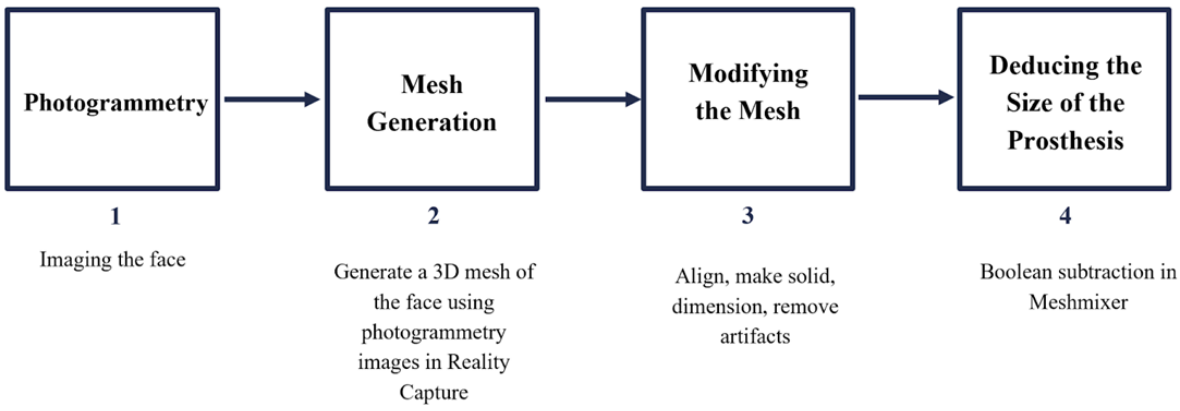


Fig. 2. Flowchart showing the steps in the proposed alternative method

the camera location and line of sight direction are known, the lines can be mathematically intersected to produce the XYZ coordinates of each target point[20]. By taking many photos, photogrammetry is able to identify XYZ coordinates of the 3D object and generate a mesh. Photogrammetry is governed by the coplanarity constraint: for two cameras, the viewing rays through corresponding image points must be coplanar, because they intersect at the 3D point[23]. This means that while taking photos of human eyes, it is important to rotate on a constant horizontal plane around the head. Other factors like lighting and camera resolution are also important to consider, as good lighting and higher camera resolution makes it easier for the photogrammetry software to identify common points in images, and will result in a higher resolution mesh.

Photogrammetry was selected over other imaging techniques to model an enucleated socket because of its ability to generate effective 3D models using a non-contact method and because it does not require expensive resources. Compared to similar imaging techniques such as LiDAR or structured light that require hardware that may amount to thousands of dollars, photogrammetry can be performed with a modern cell phone and freeware, making it easiest to implement. More importantly, photogrammetry is non-invasive and does not expose the patient to any harmful radiation. It is important to note that LiDAR could potentially be used with a smartphone in this application instead of photogrammetry; however, it requires a 3D scanner that only the latest iPhone models (12/13 Pro) have, and which was unavailable to the researchers at this time. Many of the apps are also new and require payment to create high-quality scans or to export them[27] . This form of LiDAR is not as widely accessible as photogrammetry, although it would be relevant to investigate in the future when it becomes available and is tested on more phone models. Because photogrammetry has never been used to image eye sockets, the researchers performed various tests to optimize the imaging conditions to generate the most accurate mesh models.

Lens type

First, a macro lens was chosen to image the faces of several non-enucleated subjects with their eyes open. However, the macro lens' short depth of field proved to be incompatible with photogrammetry. The images taken using the macro lens failed to capture the details of the eye socket and the mesh model was insufficient for Boolean subtraction (Figure S1). An iPhone 11 Pro camera was then used to image non-enucleated subjects with their eyes open. The mesh models generated using these images were significantly better (Figure S2).

Lighting

Proper lighting is an important factor in photogrammetry. Images were taken of the same non-enucleated subjects using artificial and outdoor lighting, and mesh models were created. After qualitative evaluation, it was determined that there was only a slight difference between the two lightings, with the natural outdoor lighting being superior. One other adjustment had to be made due to the glare that was created from light on the eyeball, which was preventing proper mesh reconstruction. The researchers compared the models of closed eyes to open eyes and found that closed eyes significantly improved resolution of the mesh, and that the shape of the eye could still be determined by looking at the shape of the eyelid. After this, researchers imaged exclusively closed eyes. To further test lighting, outdoor lighting and a ring light with three settings–white, warm white, and warm yellow– was used to generate meshes (Figure 3). There was only a slight difference in the quality of the models generated using different ring light settings, and outdoor lighting was deemed superior due to slightly better observed rendering of skin texture, as well as better rendering of eyelid curvature. Because of this advantage in quality, outdoor lighting was determined to be the optimal setting, with any of the ring light setting as a viable alternative if outdoor lighting is inaccessible. It was noted that artificial indoor lighting produced the worst results.

Mesh Generation

The next step in the method is mesh generation, which involves using the images from photogrammetry and using them to generate a 3D mesh rendering of the face. The software selected to use for this step is Reality Capture[21], because it is free and relatively easy to use. After importing the image set, Reality Capture aligns the images, and uses the alignment to generate a 3D mesh rendering of the object. The main optimization problem for this step was computational time, as generation of the 3D mesh takes time and computational power, and varies depending on the number of images used. The efficiency of the method is dependent on the computational time for mesh generation from the 2D image set. Other factors such as 3D printing and mesh model editing will also affect the efficiency of the method, but the number of photos used to generate an accurate mesh model within a reasonable timeframe is an easily adjustable factor. Therefore, photogrammetry was tested on the same object six times using a different number of images each time. The computational time was recorded with each iteration and plotted against the number of images in the set (Supplemental Figure 3). Researchers concluded that 60-80 images was optimal, as this had a computation time of ~60-90 minutes and produced a high-quality mesh with few observable artifacts. Increasing the number of images beyond this did not appear to improve the mesh quality or reduce artifacts, but significantly increased computational time.

Modifying the Mesh

The next step in the method involves modifying the mesh obtained in the previous step in a software called Meshmixer. Like Reality Capture, Meshmixer was selected because it is free and the interface is user friendly. Upon importing the mesh, any artifacts are removed. Then, the mesh is properly aligned in the Meshmixer plane. It must then be made solid to allow for later Boolean subtraction. It is also scaled to match its real-world dimensions, which is imperative for later accuracy in prosthesis shape.

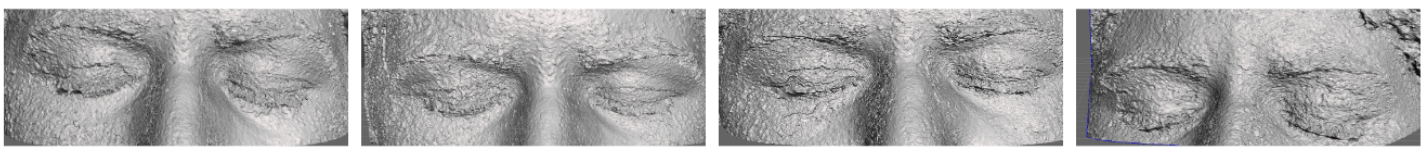
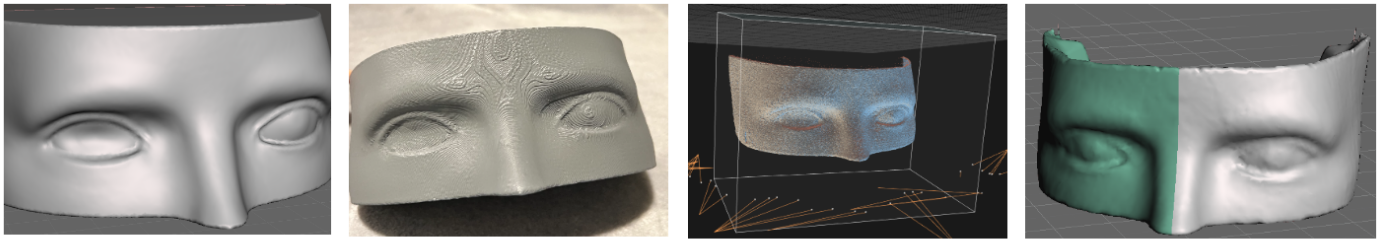


Fig. 3. 3D mesh models created with different ring light settings including a) warm yellow, b) warm white, c) white and d) outdoor lighting



a) Enucleated CGI using a Gaussian function | b) Printed enucleated CGI model | c) Regenerated enucleated CGI model | d) Mirrored enucleated CGI model

Fig. 4. CGI head model with left socket enucleated using a Gaussian function in a) Meshmixer, b) 3D printed, c) re-imaged in RealityCapture, d) modified in Meshmixer

Deducing the Size of the Prosthesis

The final step involves Boolean subtraction in Meshmixer. This relies on the assumption that the normal eye and enucleated eye can be superimposed, and the prosthesis can be determined by comparing the difference in shape. First, the enucleated face model is duplicated and offset vertically from the original model. The duplicated model is then mirrored across the center vertical line, so that it has two normal eyes. The duplicated model and enucleated model are then aligned, so that the enucleated eye of the original model is superimposed with the normal eye of the duplicated model. The areas of the models other than the superimposed eyes are then removed to reduce computational time. Then the Boolean difference tool is used, which is a mathematical operation that outputs the areas where the two models do not intersect. The prosthesis shape is obtained after removing noise artifacts from this output and smoothing it to get rid of any sharp edges.

Testing Methods of Enucleation

Researchers decided to create an artificially enucleated model to test on, which was imperative to be able to test the method's accuracy and precision. By indenting the eye artificially, the exact dimensions of the indentation could be recorded and then compared to the dimensions of the prosthesis obtained through the method. A few different methods of enucleation were tested. A CGI head from Turbosquid was used to create the artificially enucleated models[28].

Manual indentation

The model was first enucleated by manually indenting the coordinates of one eye socket (Figure S4). However, this was determined to be ineffective since the socket appeared 'blocky' and measuring the size and shape of the indentation was imprecise and thus difficult to analyze.

Boolean subtraction with prosthesis

Next, an ocular prosthesis was designed in AutoCAD software using the average dimensions of a prosthesis (Figure S5). Boolean subtraction was performed between this ocular prosthesis with known dimensions and the CGI head model to create an enucleated socket (Figure S6). However, the size and shape of the enucleation was imprecisely quantifiable since accuracy was lost during Boolean subtraction.

Indentation with Gaussian function

Finally, a Gaussian function was used to artificially enucleate the right eye socket of the CGI model as shown in Figure 4a (See Materials and Methods). This was determined to be the most accurate and measurable method to artificially enucleate a socket, since the subtracted thickness could easily be calculated at any given point using the function.

Testing the Method on the Artificially Enucleated CGI Model

The enucleated model created by indenting the socket using a Gaussian function was then printed in polylactic acid (PLA) using fused deposition modeling (FDM) and spray-painted in matte gray to reduce the shine from the PLA (Figure 4b). The model was then imaged outdoors in midday, non-direct sunlight using photogrammetry. Then, a mesh model of the printed, enucleated CGI was generated using Reality Capture (Figure 4c). The model was imported to Meshmixer, duplicated, and mirrored across the vertical plane through the nose (Figure 4d) (see Materials and Methods). Afterwards, the two models were aligned and a Boolean difference was performed to obtain the size of the ocular prosthesis.

Assessment of Method Accuracy and Precision

After optimization of the method, it was important to test its accuracy and precision. Once the size and shape of the prosthesis was determined using Boolean difference, the thickness of the prosthesis at every point was calculated using a Python script. These reconstructed thickness values were plotted against the known dimensions from the Gaussian indentation and compared (See Fig S10 for code). The method was repeated four more times on the same CGI model to test precision. Root mean square error (RMSE) was calculated between the reconstructed and known points. The average RMSE for the five tests was 0.43 +/- 0.05 mm. A full table of results for the 5 iterations is shown in Supplemental Table 3. An example of a prosthesis and its subsequent plot from the fifth iteration of the method is shown in Figure 5.

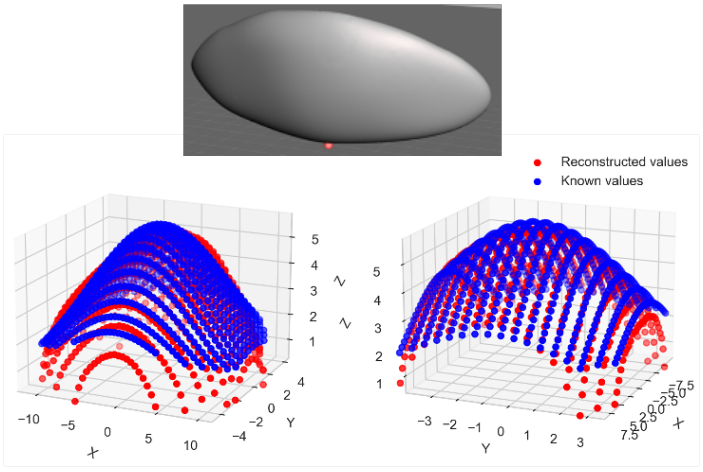


Fig. 5. An ocular prosthesis obtained from using the novel method on an artificially enucleated CGI head model and plots of the prosthesis thickness values compared to known Gaussian indentation dimensions

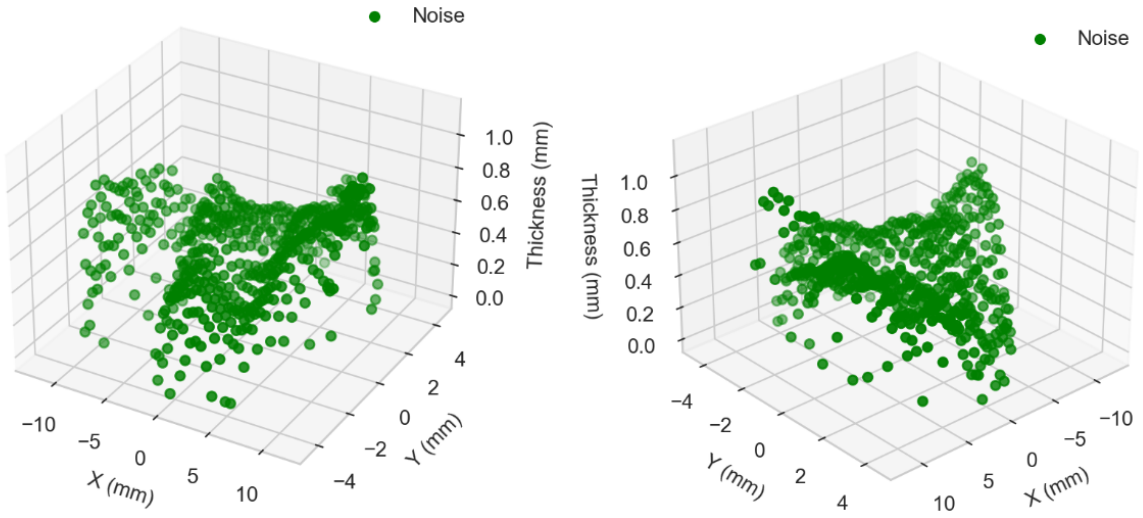


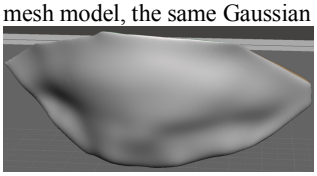
Fig. 6. Thickness values for each point where a difference existed between two non-enucleated, natural eyes

Boolean Subtraction of Non-Enucleated Real Model

This method is based on the theory that the contralateral eye can be used as an accurate replica of the enucleated eye. However, real faces may not be perfectly symmetrical. To test the validity of this assumption, researchers performed Boolean subtraction between two non-enucleated eyes of a mesh model created using the photogrammetry technique described in this paper on a male model. The average thickness of the noise artifacts was found to be 0.50 mm. These results are plotted in Figure 6, which also shows that there were many points with no detectable difference between the two eyes. However, this test was only performed a single time on one model. More data is needed to thoroughly assess the accuracy of using the contralateral eye as a model for the enucleated eye.

Boolean Subtraction of Enucleated Real Model

The method was then tested on an artificially enucleated real face model. After imaging a subject using photogrammetry and obtaining a mesh model, the same Gaussian function used to create the enucleation in the CGI model was applied. This created an artificially enucleated real face model (Figure 8). The method was used to modify the mesh and perform Boolean subtraction to obtain the ocular prosthesis shape. The thickness values at every point of this ocular prosthesis were measured and compared to the Gaussian indentation in the same way the CGI head model was tested. The resulting prosthesis and plot of reconstructed and known values is shown in Figure 7. The average RMSE between the prosthesis and Gaussian indentation was 0.29 mm, which was superior to the CGI model tests; however, more tests are needed to confirm this trend.



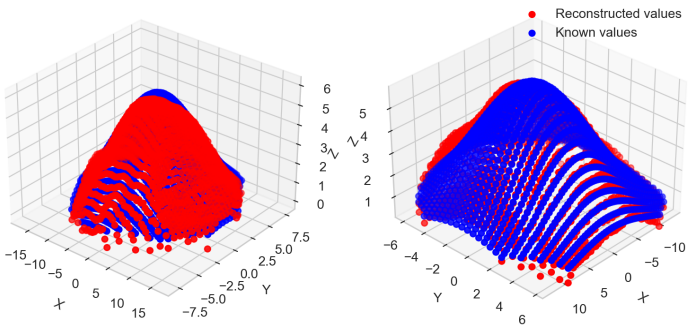


Fig. 7. Results after testing the method on an enucleated socket from a real face mesh model, showing the reconstructed prosthesis and comparison of thickness values and known dimensions from Gaussian indentation

Discussion

The results of this paper provide the foundation for a novel, alternative method to create custom ocular prosthetics. This alternative method also meets the design criteria of being efficient, accurate, accessible, and comfortable for patients. By shifting modeling and production digitally, this method eliminates the need for the uncomfortable mold impression step traditionally used by ocularists to determine the shape of a custom prosthesis for a patient. Additionally, the elimination of the impression mold and the following steps reduces the amount of physical resources needed to create the prosthesis, which saves time and potentially reduces cost. This would also reduce the number of appointments needed to create a prosthesis for a patient; instead of the current 3-4 appointments[29], this method would likely only require 1. Overall, a full iteration of the method costs less than $1 per case, and it takes around ~2.5 hours to obtain the shape prosthesis from start to finish. All of the software is free to download and is compatible with Windows OS. Through assessing the method accuracy and precision using the CGI model, it was found that the average RMSE was 0.43 +/- 0.05 mm, which is a promising start for both accuracy and precision. The RMSE for the real enucleated model was even less at 0.29 mm. This method also obtains accurate information about the anterior portion of the prosthesis and prevents measurement error due to overfilling during the impression step, which is advantageous compared to current methods. There were no available values in literature to determine how much error is physiologically significant or acceptable in ocular prosthesis design, but this is something that would be investigated in future work, perhaps in combination with patient trials. For perspective, the average prosthesis is about 9 mm thick[30], and the average RMSE was less than half a mm.

Next steps

The proposed method is limited as there is still work to be done to finalize it. First, there needs to be more tests and iterations done with real faces. Although certain steps in this method (photogrammetry, mesh generation) were optimized for use with real faces, much of the testing in the mesh editing and Boolean subtraction steps was done using the CGI model. Testing on scans of more real faces that have been artificially enucleated would improve the optimization of the method and perhaps reveal how to better align different face shapes, as in reality most faces are not symmetrical, and only one real face was tested in this project.

Additionally, it would be helpful to test the use of different Gaussian functions and other potential indentation techniques with this method. It is possible that a different indentation technique or shape could lead to different resulting errors or different imaging requirements during photogrammetry, which would need to be accounted for. The Gaussian function, though useful for testing in this project because it was a known value, is not a perfect approximation in shape for an enucleated socket. It could also be useful in this sense to perhaps take a facial scan of a patient with a lost eye and then 3D print that model to use in testing. Testing on more faces and with different shapes of enucleated sockets would also give the method more realism, as ocularists report that all enucleated sockets are unique and therefore each prosthesis has slightly different shape[29].

Once the various steps in the proposed method are improved, it would be ideal to begin exploring 3D printing of the prosthetics. The ideal outcome would be to take the 3D model of the prosthesis generated using the proposed steps of this method and 3D print it, after which it would be usable by a patient.3D printing the prosthetics would also require extensive research into printing material biocompatibility, as the prosthesis is in contact with the eye socket and cannot leach or degrade over time in a way that could cause harm. There are a number of biocompatible printing materials currently available, an example is the Formlabs BioMed resin, which is USP Class VI certified and supported by an FDA Master File[19]. This resin would be printed using stereolithography (SLA) which is a higher resolution than the fused deposition modeling (FDM) printing used for the CGI model; the prosthetics would need to be printed in SLA so that they have a smoother finish. Notably, the incorporation of 3D printing at the end of the method would increase the cost of the method and decrease accessibility. The BioMed resin filament costs $349 per liter (1L of resin could be used to print several prosthetics), while a SLA printer starts at $3,750[31]. However, this is still comparable to the current cost of even one custom prosthesis, which is ~$2000 - $8000[20].

Future work

In the future, it would be ideal to begin testing the finalized method with patients. This would be necessary before the method could be incorporated into any ocularist clinics. Working with patients could involve having them try the prosthetics on and report levels of comfort, or how similar the prosthetics are to ones they have worn in the past. This would also help determine how small the RMSE needs to be so it is undetectable when the prosthetic is worn.

In its current state, this method does not include any steps to replace or replicate the aesthetic decoration part of prosthetic design. Currently, ocularists hand-paint the iris and sclera onto the prosthetic to match the patient's contralateral eye. This requires significant time, and also means that the creation of the prosthetic is highly dependent on the skill of the ocularist. One way that this method could improve on this process would be to incorporate other digital imaging methods, like printing the iris. In prior art, there was a method discussed where the iris and sclera were printed onto the prosthetic after it was made using images of the patient's eye[13]. If this method were combined with the proposed method, the whole process of prosthetic creation would be digital, which would further streamline creation and minimize patient contact.

Limitations

One of the main limitations of this alternative method is that patients who have lost both of their eyes could not benefit, because the foundation of this method is using the contralateral eye as a base of comparison. This is also a limitation of current practice, as ocularists also use the contralateral eye as standard. There is the potential that with further research, this alternative method could be extended to patients who have lost both eyes by using other eyes as standard, but it would require additional work to determine the necessary shape and desired appearance. In a similar sense, this method would become more complex with more asymmetrical faces, and for cases where the enucleated socket has a more unique shape due to conditions like fat loss around the eye[29]. Using the shape of the eyelid to determine prosthesis shape is an additional limitation, as the smoothness of the eyelid can obscure small features within the socket. This could potentially be accounted for by using images of the patient's enucleated socket to sculpt these features into the prosthesis at the end. Another limitation of this method is that it is not fully automated, and relies on the use and upkeep of Reality Capture and Meshmixer software. In the future, it would be ideal to design one piece of software that could perform all the functions of the method from the original photogrammetry images (mesh generation, editing, and Boolean subtraction).

Alternative applications

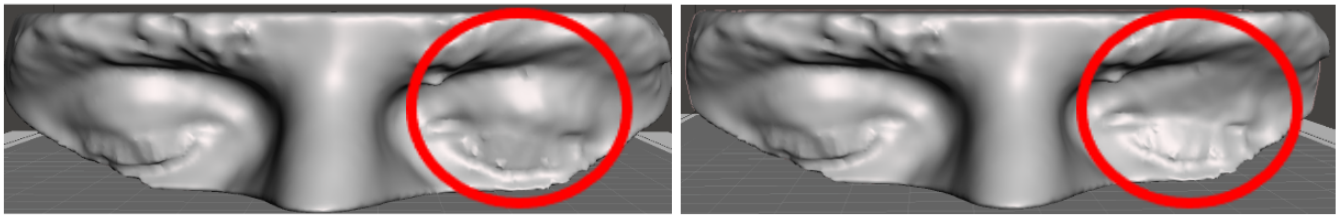
While the focus of this project was on an alternative process to create custom ocular prosthetics, the technology used and developed has potential alternative applications. 3D modeling could be used in many other aspects of prosthetic creation to streamline creation. For example, scanning and modeling the mold trays or old prosthetics could allow ocularists to have the exact shape of old prosthetic designs. This could be helpful when designing new prosthetics for patients, or to offer an alternative between a stock prosthesis and a fully custom one. If ocularists kept a repository of old designs they have made for previous patients, they could offer them to patients who are not able to pay for a fully custom one. This method could also be relevant in the creation of other prosthetic devices, especially in any case where one side of the body needs to match the other.

Materials and Methods

Materials

Software and Hardware

The two softwares used in this project are RealityCapture and Autodesk Meshmixer, which both have minimum system requirements. RealityCapture requires a 64-bit machine with at least 8 GB of RAM, 64-bit Windows version 7+, as well as a NVIDIA graphics card with CUDA 3.0+ capabilities and 1GB VRAM[32]. RealityCapture can still be used without a NVIDIA graphics card to register images, but a NVIDIA GPU is needed to create a textured mesh. Meshmixer requires Windows version 7+, 4GB of system RAM, and an integrated or discrete graphics card with updated drivers[22]. The Meshmixer and RealityCapture Support FAQs and support pages provide more information on installation instructions and troubleshooting. Three computers were used for the entirety of the project: (1) A desktop with an AMD Athlon(™) X4 880K Quad Core Processor (4.00 GHz) with 16 GB of RAM and a NVIDIA GeForce GTX 1050 Ti graphics card; (2) A desktop with an Intel(R)



a) 3D mesh model of a real, non-enucleated face b) 3D mesh model of a real, enucleated face

Fig. 8. The real face 3D mesh model, which was created using the novel method, originally has no enucleated sockets a) and is later artificially enucleated using the Gaussian function in equation 1 shown in b) and highlighted by the red circles

Xeon(R) W-2133 CPU @ 3.60 GHz, with 32 GB RAM and a NVIDIA Quadro P2000 GPU; and (3) a Surface Book 2 laptop with an Intel(R) Core(™) i7-8650U CPU @ 1.90 GHz processor with 16 GB of RAM and a NVIDIA GeForce GTX 1050 graphics card. All three computers had Windows 10 Pro installed and 64-bit operating systems. Meshmixer version 2.5 and RealityCapture version 1.2 were used for the duration of this project.

Two phone cameras were used for photogrammetry to image faces, (1) an iPhone 11 Pro and (2) an iPhone 12. The default back camera (1x) was used in each instance, and images were taken outdoors during the daytime approximately 6-10 inches from the face. The macro lens that was tested is the 25 mm M-Series macro lens from Moment for the iPhone 11 Pro. The ring light that was used to test artificial lighting was a 10-inch LED ring light purchased from Amazon[33]. The three lighting levels were white, warm white, and warm yellow. All settings were used at the max brightness level. More details on the imaging method are discussed in Methods.

Enucleated Models

The female CGI head model was obtained online as a .obj file from Turbosquid.com[28]. First, parts of the model were cut off and discarded so that it resembled a slab of the face including just the eyes. Then, it was dimensioned to be 150 mm wide (similar to that of a real head based on measurements researchers took). An image of this model with dimensions is shown in Supplemental Figure 7.

The CGI head model was artificially enucleated using a code program that directly edits the obj file containing the mesh points. The code for indenting the CGI head is shown in supplemental material 10. A 3D Gaussian function was used to indent the right eye, by subtracting the function from every point in the mesh. The Gaussian function used for this indentation is shown in Eq 1.

$$y = 6 \cdot e^{-\frac{(x-x1)^2}{2(8^2)}} \cdot e^{-\frac{(z-z1)^2}{2(4^2)}} \quad , x1 = -35, z1 = 24 \qquad [1]$$

X1 and Z1 indicate the center point of the enucleated eye, which was found by using the 'analysis' -> 'measure' tools in Meshmixer to look at xyz coordinates of individual points on the surface of the mesh. The final enucleated CGI model is shown in Figure 8b. This model was 3D printed using fused deposition modeling (FDM) on a LulzBot printer with polylactic acid (PLA) filament. The printed model is shown in Figure 4b.

The male enucleated model was created using photogrammetry by imaging a real face. This initial scan is shown in Supplemental Figure 8. Researchers followed the steps described in Methods up to Boolean subtraction. Once the 3D mesh of the face was obtained and edited, the model was dimensioned to 150 mm wide, similar to the CGI model. The eyelashes were also removed and smoothed using the 'select' and 'discard' tools in Meshmixer, followed by the 'select', 'deform', and 'smooth' tools. The eyelashes were highlighted and discarded with brush size 15. The final model is shown in Figure 8a.

This model was enucleated on the left eye using similar code, but with different values for the center point of the Gaussian function. This code is shown in supplemental material 10. The Gaussian equation is shown in Eq 2 below.

$$y = 6 \cdot e^{-\frac{(x-x1)^2}{2(8^2)}} \cdot e^{-\frac{(z-z1)^2}{2(4^2)}} \quad , x1 = 37, z1 = 12 \qquad [2]$$

The final enucleated model using a real face is shown in Figure 8b.

Methods

See supplement 9 for the detailed instruction manual for ocular prosthesis creation.

End Matter

Author Contributions and Notes

The authors declare no conflict of interest.

Acknowledgements

The authors would like to thank Dr. William Guilford, our biomedical engineering faculty advisor, for providing guidance and feedback throughout the research. We would also like to thank Dr. Allen and Dr. Barker, our Capstone instructors for the past year.

References

1. Pine, K. R., Sloan, B. H. & Jacobs, R. J. Clinical Ocular Prosthetics. (Springer International Publishing, 2015). doi:10.1007/978-3-319-19057-0.
2. Modugno, A. et al. Ocular prostheses in the last century: a retrospective analysis of 8018 patients. Eye 27, 865–870 (2013).
3. Moshfeghi, D. M., Moshfeghi, A. A. & Finger, P. T. Enucleation. Surv. Ophthalmol. 44, 277–301 (2000).
4. LA Custom Prosthetic Eye. https://ocularpro.com/ https://ocularpro.com/the-difference-between-a-stock-and-custom-prosthetic-eye/.
5. Rokohl, A. C. et al. Socket discomfort in anophthalmic patients—reasons and therapy options. Ann. Eye Sci. 5, 36–36 (2020).
6. Dupierrix, E. et al. Preference for human eyes in human infants. J. Exp. Child Psychol. 123, 138–146 (2014).

7. Chao, J. Stock Eye vs. Custom Prosthesis. Prosthetics Advancement Lab http://www.prostheticslab.com/blog/2018/5/30/stock-eye-vs-custom-prosthesis-before-and-after.
8. Prosthetic Eye: Cost, Care, Surgery, and More. Healthline https://www.healthline.com/health/prosthetic-eye (2018).
9. American Society of Ocularists - Search by State/Province. https://www.ocularist.org/find_ocularist_search.asp?&tab=1.
10. Thakkar, P., Patel, J., Sethuraman, R. & Nirmal, N. Custom Ocular Prosthesis: A Palliative Approach. Indian J. Palliat. Care 18, 78–83 (2012).
11. Cevik, P., Dilber, E. & Eraslan, O. Different Techniques in Fabrication of Ocular Prosthesis: J. Craniofac. Surg. 23, 1779–1781 (2012).
12. Chinnery, H., Thompson, S. B. N., Noroozi, S. & Dyer, B. Scoping review of the development of artificial eyes throughout the years. Edorium J. Disabil. Rehabil. 3, 1–10 (2017).
13. Zoltie, T., Bartlett, P., Archer, T., Walshaw, E. & Gout, T. Digital photographic technique for the production of an artificial eye. J. Vis. Commun. Med. 44, 41–44 (2021).
14. Jain, S., Makkar, S., Gupta, S. & Bhargava, A. Prosthetic Rehabilitation of Ocular Defect Using Digital Photography: A Case Report. J. Indian Prosthodont. Soc. 10, 190–193 (2010).
15. Artopoulou, I.-I., Montgomery, P. C., Wesley, P. J. & Lemon, J. C. Digital imaging in the fabrication of ocular prostheses. J. Prosthet. Dent. 95, 327–330 (2006).
16. Ruiters, S., Sun, Y., de Jong, S., Politis, C. & Mombaerts, I. Computer-aided design and three-dimensional printing in the manufacturing of an ocular prosthesis. Br. J. Ophthalmol. 100, 879–881 (2016).
17. Ko, J., Kim, S. H., Baek, S. W., Chae, M. K. & Yoon, J. S. Semi-automated fabrication of customized ocular prosthesis with three–dimensional printing and sublimation transfer printing technology. Sci. Rep. 9, 2968 (2019).
18. Pallaver, A. & Honigmann, P. The Role of Cone-Beam Computed Tomography (CBCT) Scan for Detection and Follow-Up of Traumatic Wrist Pathologies. J. Hand Surg. 44, 1081–1087 (2019).
19. 3D Printing Materials For Healthcare. Formlabs https://formlabs.com/materials/medical/.
20. Prosthetic Eye: Cost, Care, Surgery, and More. Healthline https://www.healthline.com/health/prosthetic-eye (2018).
21. RealityCapture - CapturingReality.com. https://www.capturingreality.com/realitycapture.
22. System Requirements Meshmixer 3.5. https://www.meshmixer.com/download.html.
23. Schindler, K. Mathematical Foundations of Photogrammetry. in Handbook of Geomathematics (eds. Freeden, W., Nashed, M. Z. & Sonar, T.) 1–14 (Springer, 2020). doi:10.1007/978-3-642-27793-1_63-1.
24. What is Photogrammetry? – Geodetic Systems, Inc. https://www.geodetic.com/v-stars/what-is-photogrammetry/.
25. Hartley, R. I. & Sturm, P. Triangulation. Comput. Vis. Image Underst. 68, 146–157 (1997).
26. Deli, R. et al. Three-dimensional methodology for photogrammetric acquisition of the soft tissues of the face: a new clinical-instrumental protocol. Prog. Orthod. 14, 32 (2013).
27. Here are some hot apps to try on your iPhone Pro with LiDAR. iMore https://www.imore.com/best-apps-lidar (2020).
28. Free 3D Free Bust Head Base Mesh model - TurboSquid 1832518. https://www.turbosquid.com/3d-models/3d-free-bust-head-base-mesh-model-1832518.
29. American Society of Ocularists Conference. (2021).
30. Hoang, H. P. et al. Factors affecting dimensions of the 3D ocular prosthesis in patients rehabilitated at Mahidol University. 6.
31. 3D Printing Guide: Types of 3D Printers, Materials, and Applications. Formlabs https://formlabs.com/3d-printers/.
32. OS and hardware requirements. RealityCapture Support https://support.capturingreality.com/hc/en-us/articles/115001524071-OS-and-hardware-requirements.
33. Amazon.com : 10" Ring Light MACTREM LED Light Ring with Tripod, Clamp & Phone Holder for YouTube Video, Makeup, Selfie, Photography, Live Streaming, Tiktok, 3 Light Modes & 10 Brightness Level : Electronics. https://www.amazon.com/MACTREM-YouTube-Photography-Streaming-Brightness/dp/B08F36R7Y1.

Supplemental Materials

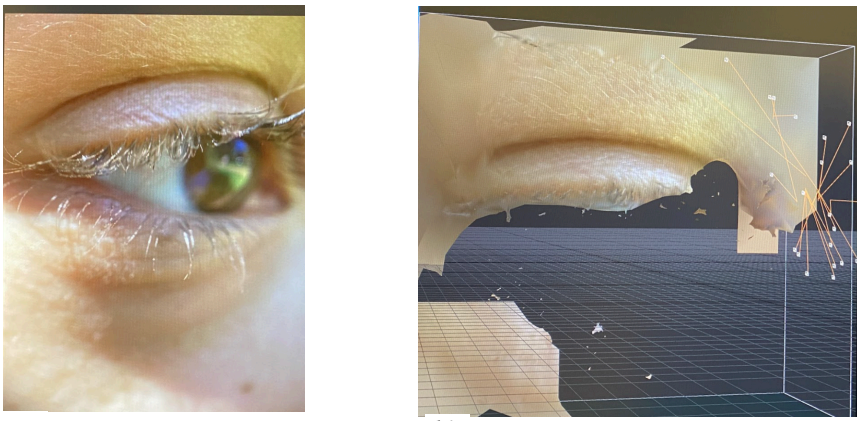
Supplemental Table 1: Pugh Chart

		Baseline	Alternative 1	Alternative 2	Alternative 3	Alternative 4
Design Evaluation Criteria	**Weight Factor**	**Current custom prosthesis**	**Custom prosthesis made using structured light scanning**	**Custom prosthesis made using Cone Beam Computed Tomography**	**Custom prosthesis with a digitally printed iris**	**Custom prosthesis made using photogrammetry**
Dimensional accuracy	3	0	S	S	S	S
Safety/biocompatibility	3	0	-	-	S	-
Comfortable fit	3	0	S	S	S	S
Patient comfort during prosthesis creation	3	0	+	+	S	+
Production cost	2	0	-	-	S	+
Production time	2	0	+	+	+	+
Resource use	2	0	+	+	+	+
Applicable to special cases	1	0	+	+	S	-
Aesthetically pleasing	1	0	S	S	-	S
Time required to learn method/expertise	1	0	-	-	-	+
Reusable model	1	0	+	+	+	+
Total +		0	5	5	3	6
Total -		0	3	3	2	2
Total S		0	3	3	6	3
Total		**0**	**3**	**3**	**3**	**8**

Supplemental Table 2: Design Constraints

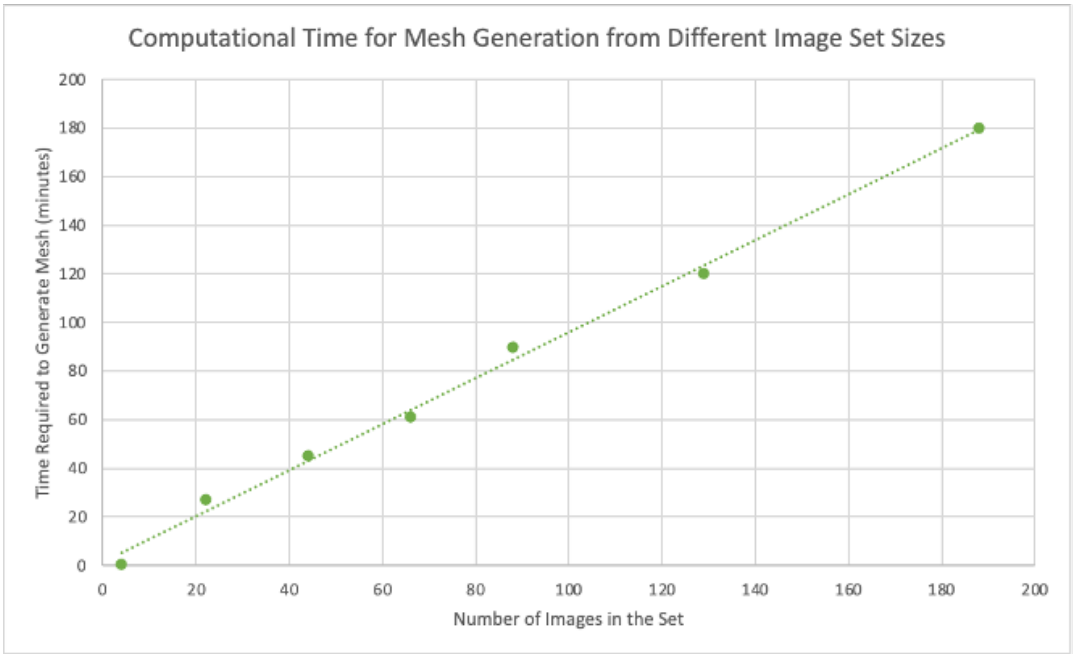
Need #	Design Constraint/Metric	Unit of Measure	Acceptable	Ideal Value
1a	Prosthetic horizontal width	mm	24 +/- 2	24+/- 0.5
1b	Prosthetic vertical height	mm	23 +/- 2	23 +/- 0.5
1c	Prosthetic thickness	mm	9 +/- 2	9 +/- 0.5
1d	Prosthetic volume	ml	2 - 3	2.2-2.5
1e	Prosthetic mass	g	2.36-3.54	2.60-2.95
2	Alignment metric	%	>90%	>95%
3a	Model computational time	Minutes	<180 minutes	<90 minutes
3b	Number of images used	# of images	50 - 100	50 - 70
4a	Software cost	Dollars	<$500	$0

4b	Time required to install/learn	Minutes	<360	<120
4c	Software compatibility	Yes/No	Windows OS	Windows, Linux, MacOS
5	Aesthetically pleasing/accurate	Yes/No	Yes	Yes

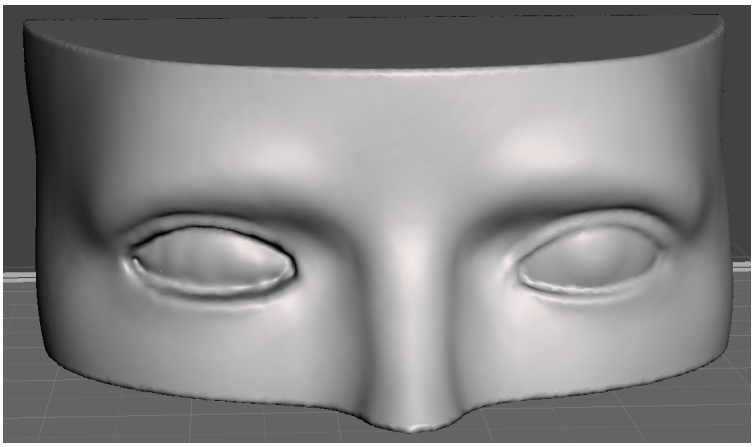


Supplemental Fig. 1. Testing photogrammetry using a macro lens. a) The non-enucleated eye being tested and b) 3D mesh model generated

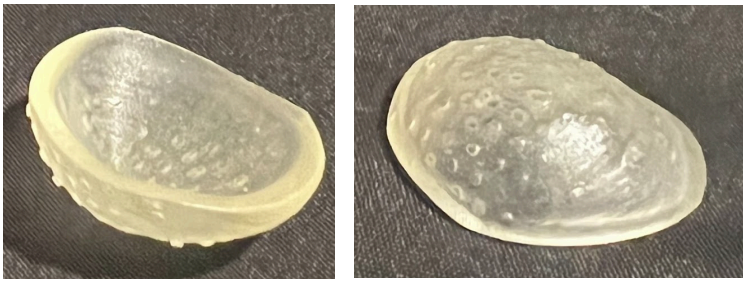
Supplemental Fig. 2. 3D mesh model generated using an iPhone 12 1x camera lens



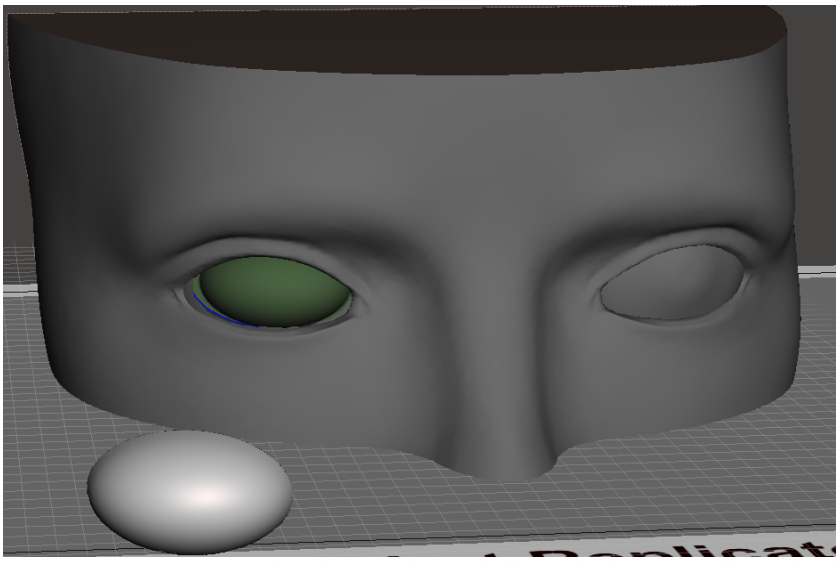
Supplemental Fig. 3. Computational time for mesh generation with different image set sizes



Supplemental Fig. 4. Enucleated socket model in Meshmixer created by manipulating the coordinates of the obj file to indent the eye between 0.1 and 0.4 centimeters



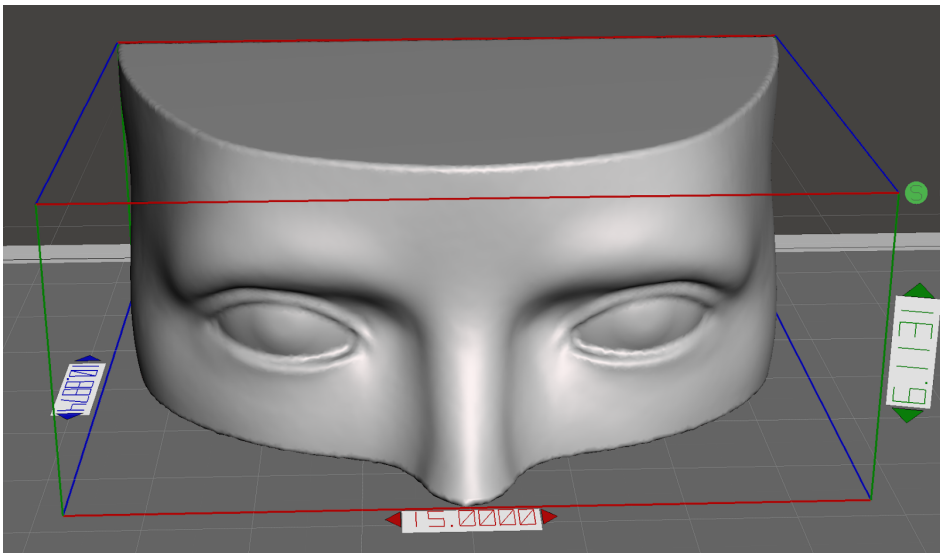
Supplemental Fig. 5. 3D printed ocular prosthesis created in AutoCAD and used for Boolean subtraction to create an enucleated socket model



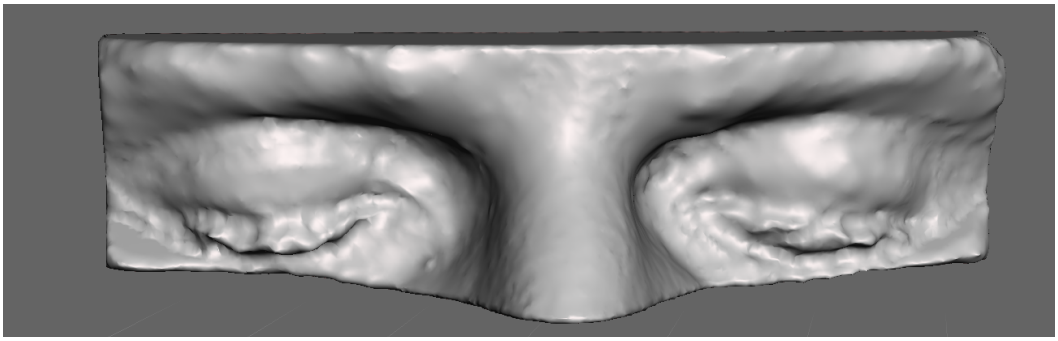
Supplemental Fig. 6. Enucleated CGI socket model in Meshmixer created by performing boolean subtraction between the prosthesis shown and the original non-enucleated socket model

Supplemental Table 3: RMSE values for each prosthesis created using the novel method

Prosthesis	RMSE
1	0.42 mm
2	0.54 mm
3	0.43 mm
4	0.41 mm
5	0.38 mm
Average RMSE: 0.43 +/- 0.05 mm	Range: 0.38–0.54 mm



Supplemental Fig. 7. Dimensions CGI head model in Meshmixer with two non-enucleated sockets



Supplemental Fig. 8. Original 3D mesh model of a real face with two non-enucleated sockets before model editing

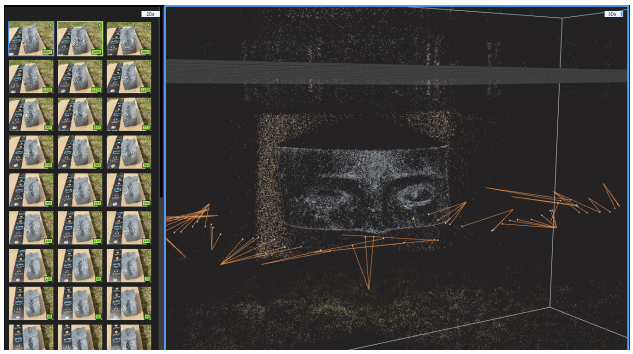
Supplemental material 9: Detailed manual of steps in ocular prosthesis creation

Instruction Manual for Custom Prosthesis Creation

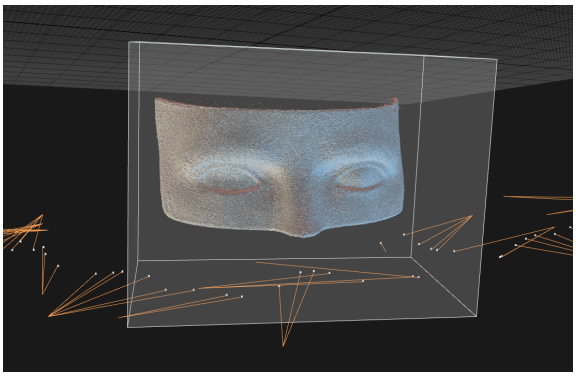
Photogrammetry

The first step involves imaging the face. A smart phone camera is suitable enough for these pictures, although a camera with higher resolution could be used. Approximately 60-80 pictures were taken for each model using an iPhone 11 Pro/ iPhone 12 camera, 6-10 inches away from the face. Pictures were taken while revolving around the individual, keeping the height of the camera constant, so that the images showed the face from all angles. The pictures were taken outdoors in the afternoon on a sunny day, though not in direct sunlight. The individual was positioned so that shadows on the face were minimized while taking pictures. The individual was instructed to keep their eyes closed and face relaxed, as movement/fidgeting could disrupt later image alignment.

Mesh Generation



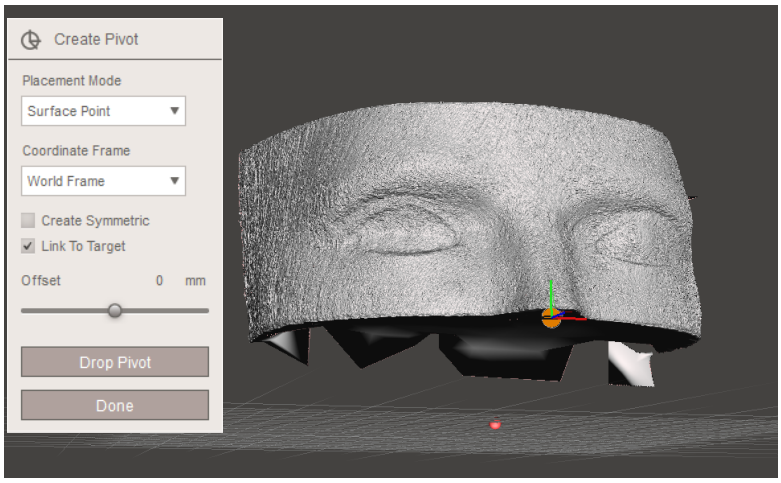
The next step involves generating a 3D mesh of the face, using the images from the previous step, which uses the RealityCapture software. This process is relatively straightforward and involves minimal input from the user after importing the images. Upon opening RealityCapture, researchers registered with a free account. First, the images were imported using the '*inputs*' tool in the workflow tab. Next, in the '*alignments*' tab, the '*align images*' function was used to generate a point cloud rendering of the object in 3D. A picture of what this step looks like is shown to the right. At this point, if some images can not be aligned, they will show up in a different color to indicate failures. These images can be selected from the image list on the left and deleted so they will not be used during the mesh generation. The reconstruction region can be specified by selecting the '*set reconstruction region tool*' within the '*mesh model*' menu, and editing the box dimensions. This will designate which parts of the point cloud will be generated into a full mesh, so certain parts of the face like the eyes can be selected, reducing computation time as well. Next, in the '*mesh model*' tab, a 3D mesh can be generated using the '*high detail*' setting in the create model section. An example of this is shown in to the right. After this, the .obj file containing the mesh can be generated using the '*export*' button in the main menu. It should be noted that users will have to pay for the licensing of their mesh to export it as a .obj, which costs around $0.40 - $0.70 per .obj file; alternatively, a lifetime subscription is available for purchase.



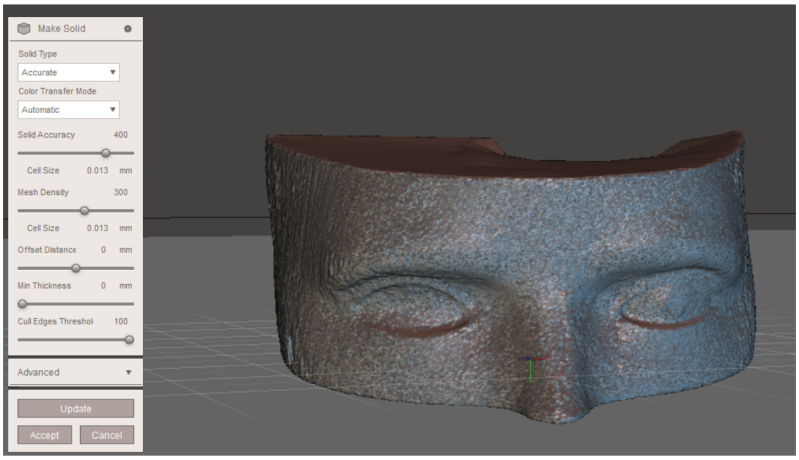
Mesh Modification

The next step involves importing the .obj file containing the mesh into Meshmixer, which is the software used for all further 3D modeling for this method. Several example images are included for these next two sections with settings visible, as the manipulation is a bit more complex. As a starting note, the object browser where mesh objects can be selected can be toggled visible by pressing ctrl + shift + o. Additionally, camera controls can be toggled by holding down the space bar. The '*import*' tool was first used to load in the .obj file. If the mesh is over 150,000 vertices, researchers reduced the mesh detail to shorten computation time by using the '*select*' tool, selecting the entire object, and then using '*edit*' and '*remesh*' to reduce the mesh between 20 - 40%. Upon importing the mesh, the face might be rendered in a variety of different directions. First, the '*edit'* and '*transform*' tools were used to spin the head so that it was oriented

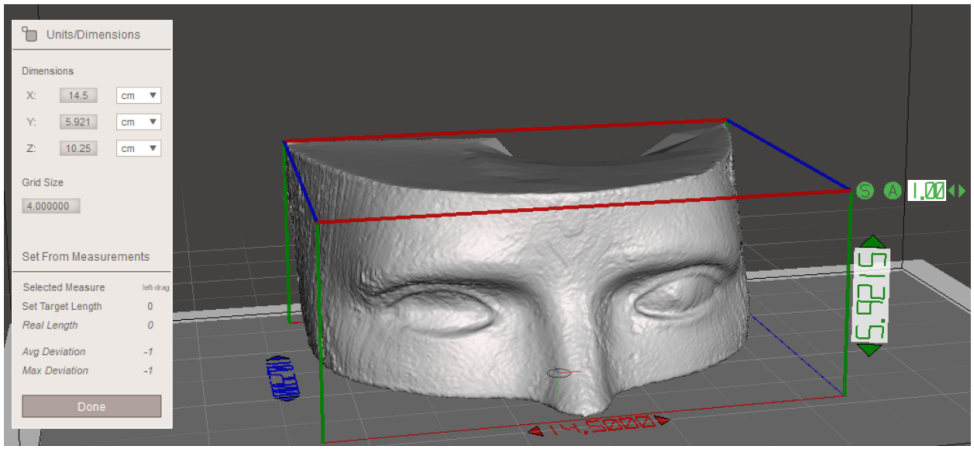
facing the proper direction. Next, to get the head flat to the bottom plane in Meshmixer, a pivot was created by using the '*edit*' and '*create pivot*' tools (right image). In the create pivot menu, the placement mode was set to surface point, the coordinate frame was set to world frame, and the link to target box was selected. The flat lower surface of the face was clicked, and '*drop pivot*' was selected to place the pivot. Next, the '*edit*' and '*align*' tools were used to align the face to the world origin. For source, pivot was selected (the pivot on the face must be clicked after this is selected to change the source), and the destination was set as 'World Origin / Y-up. For this example, translate and rotate and flip were selected, but it is possible that it will work better if only translate is selected. After this, the head was rotated so that it was parallel and perpendicular to the grid lines on the base plane.



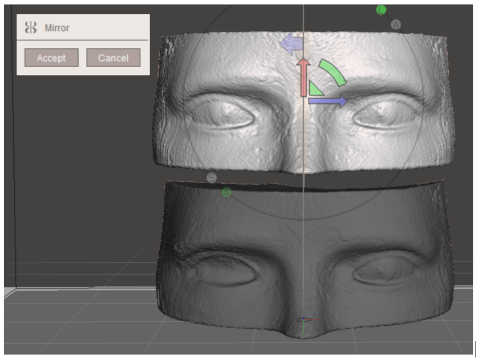
Next, the head was made solid using the '*edit*' and '*make solid*' tool (right image).



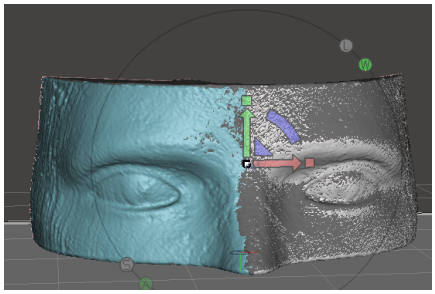
The solid type was selected as accurate, color transfer mode as automatic, solid accuracy at 400, and mesh density at 300. The model was then dimensioned to match the known dimensions using the '*edit*' and '*Units/Dimensions*' tool. It should be noted that adjusting the specific x, y, and z dimensions using the typing boxes here will change the other dimensions. Using the '*edit*' and '*transform*' tools, the squares at the ends of the colorful coordinate arrow box can be selected and dragged to adjust one coordinate dimension without changing the others; alternatively, the dimensions can be edited directly in the settings box within '*edit*' and '*transform*'. The dimensions should be set in mm (eg. 150 mm instead of 15 cm) because Meshmixer by default loads in obj files with mm; additionally, all of the Python code files included in supplements assume that the dimensions are in mm. The head should also be rotated in Meshmixer so that the X axis is the long width of the head, as shown in the image below. Artifacts from the head were removed from the edges of the model using '*edit*' and '*plane cut.*'

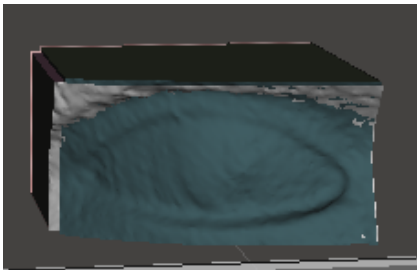


Boolean Subtraction

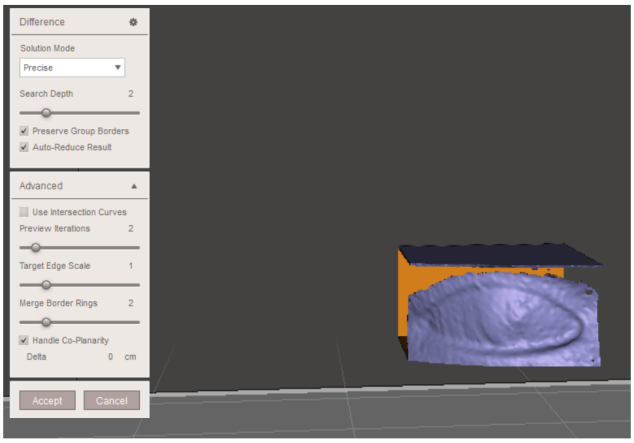


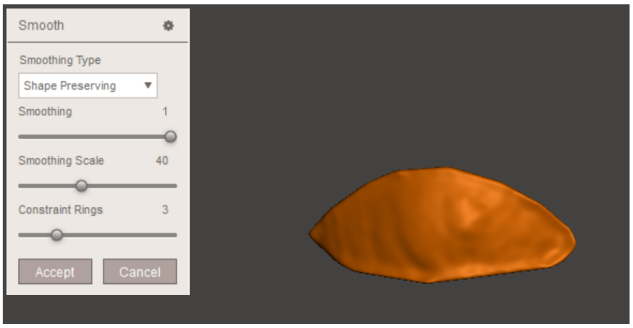
The final step involves using a tool called Boolean subtraction in Meshmixer to compare the shapes of the enucleated eye and contralateral eye to determine the shape of the prosthesis (i.e. the difference in shape). The current model was first duplicated using the '*edit*' and '*duplicate*' tool. Then, using '*edit*' and '*transform*', the duplicated model was moved vertically upwards from the current model. Next, the model on top was mirrored across the center point using the '*edit*' and '*mirror*' tools, so that the top model had 2 normal eyes, as shown in the image to the right. Next, the model on top was moved back on top of the original model using '*edit*' and '*transform'*, as shown below. Further accuracy in alignment was obtained by using the '*align to target*' tool. To use this tool, the original head was set as the target by toggling its magnet icon in the object browser. Next, the '*select*' tool was used to select the mirrored head, and the '*edit*' and '*align to target*' tools were used. The solve iterations were set to 70, and the error tolerance to 0.0005 cm. After this, all regions of the head except for the area of overlap between the normal eye of the mirrored model and the enucleated eye of the original model were removed using plane cuts, as shown to the right.





Next, using the object browser, the normal eye was first selected, followed by the enucleated eye, and the '*boolean difference*' tool was selected. Note that this tool subtracts the second selected object from the first, so selecting the non-enucleated eye first is important. Often, the first iteration of the solution failed and appeared in red. The solution mode was set to precise, and the handle coplanarity option was selected. If this still produces a failed result, researchers modified the delta value to increase incrementally from 0.001 cm up to 0.05 cm, and it should resolve. A passed result will be in purple and orange, as shown to the right. Next, the '*edit*' and '*separate shells*' tool was selected to separate some of the artifacts created by Boolean subtraction; all of those that were not the main object (which will be a lighter color when selected) were deleted. Then, the shape of the prosthetic was found by removing the areas surrounding the eye using plane cuts and the '*select*' and '*discard*' tools. Then, the prosthesis was made solid using the same settings as previously mentioned. It was then smoothed using the '*select*' tool, selecting the whole prosthesis, and using the '*deform*' and '*smooth*' tools, with the smoothing type set to shape preserving and the smoothing scale set to 40, as shown to the right. The smoothing scale was different on a case-to-case basis.





Supplemental material 9: a) Code used to enucleate CGI model, **b)** code used to enucleate real model, **c)** code used to measure thickness of prosthesis and graph against Gaussian values **d)** code to measure and graph thickness of two normal subtracted eyes

a) Code used to enucleate CGI model,

```
import math

#opening the write file for writing
f = open("cgi-solid-minus40density-write1.obj", "w")


#opening obj file and read line by line
with open('cgi-solid-minus40density-edit1.obj') as file:
    for line in file:
        content = line.split()
        #check if vertices line
        if content[0] != 'v':
            print(line, file = f)
        else:
            list = []
            list.append('v')
            for item in content:
                if item != 'v':
                    item2 = float(item)
                    list.append(item2)
                    #check  for desired values
                    if len(list) == 4:
                        x1 = -3.5
                        z1 = 2.4
                        x = list[1]
                        z = list[3]
                        val = list[2] - (.6 * math.exp(-((x-x1)**2)/(2*(.8**2)))
* math.exp(-((z-z1)**2)/(2*(.4**2))))
                        list.pop(2)
                        list.insert(2,val)
            for item in list:
                print(item, end=" ", file = f)
            print('\n', file=f)

#closing the write file
f.close()


with open("cgi-solid-minus40density-write1.obj", "r") as f:
    lines = f.readlines()
with open("cgi-solid-minus40density-write1.obj", "w") as f:
    for line in lines:
```

```
        if line.strip("\n") != "":
            f.write(line)
```

b) code used to enucleate real model

```
import math

#opening the write file for writing
f = open("anth-eyelashes-removed-enucleated.obj", "w")


#opening obj file and read line by line
with open('2eyes_anth_smooth.obj') as file:
    for line in file:
        content = line.split()
        #check if vertices line
        if content[0] != 'v':
            print(line, file = f)
        else:
            list = []
            list.append('v')
            for item in content:
                if item != 'v':
                    item2 = float(item)
                    list.append(item2)
                    #check  for desired values
                    if len(list) == 4:
                        x1 = 37
                        z1 = 12
                        x = list[2]
                        z = list[3]
                        val = list[1] - (6 * math.exp(-((x-x1)**2)/(2*(8**2))) *
math.exp(-((z-z1)**2)/(2*(4**2))))
                        list.pop(1)
                        list.insert(1,val)
            for item in list:
                print(item, end=" ", file = f)
            print('\n', file=f)

#closing the write file
f.close()


with open("anth-eyelashes-removed-enucleated.obj", "r") as f:
    lines = f.readlines()
```

```
with open("anth-eyelashes-removed-enucleated.obj", "w") as f:
    for line in lines:
        if line.strip("\n") != "":
            f.write(line)
```

c) code used to measure thickness of prosthesis and graph against Gaussian values

```
# Python program to measure thickness of a prosthesis

# This program is fully automated, there are only 2 spots where you might have to
manually enter values towards the end
   # one is if the max thickness value is not the center point of the prosthetic
      # in that case you can manually find the center point and enter it in
   # the second is the trimming bounds for finding the RMS error

import csv
from pyexpat.errors import XML_ERROR_CANT_CHANGE_FEATURE_ONCE_PARSING
import numpy
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
from mpl_toolkits import mplot3d
import math

# field names
fields = ['X', 'Y', 'Z']

# name of csv file
filename = "measured-thickness.csv"

# writing to csv file
with open(filename, 'w', newline='') as csvfile:
    # creating a csv writer object
    csvwriter = csv.writer(csvfile)

    # writing the rownames
    csvwriter.writerow(fields)

    #opening obj file and read line by line
    with open('prosthesis-4-smooth.obj') as file:
        for line in file:
            content = line.split()
            #check if vertices line
            if content[0] == 'v':
                list = []
```

```
                for item in content:
                    if item != 'v':
                        list.append(item)
                    if len(list) == 3:
                        # writing the data rows
                        csvwriter.writerow(list)

#closing the write file
file.close()

#importing csv as data frame
df = pd.read_csv (r'measured-thickness.csv')

#converting data frame to array
matrix1 = df.to_numpy()

print('Vertices stored in an array: ')
print(matrix1)
print('\n')

#getting dimensions of matrix
dims = matrix1.shape
rows = dims[0]
columns = dims[1]
print('Dimensions of array: ')
print(rows)
print(columns)
print('\n')


#sort matrix by x values
matrix = matrix1[matrix1[:, 0].argsort()]
print('Array sorted by x value: ')
print(matrix)
print('\n')

#sort matrix by y values
matrix2 = matrix1[matrix1[:, 1].argsort()]
print('Array sorted by y value: ')
print(matrix2)
print('\n')

#find max values for x and y for bounds of scan box
max_x = numpy.max(matrix[:,0])
min_x = numpy.min(matrix[:,0])
```

```
max_y = numpy.max(matrix[:,2])
min_y = numpy.min(matrix[:,2])
print('Max and min values: ')
print(max_x)
print(min_x)
print(max_y)
print(min_y)
print('\n')

#finding thickness by 'box' method?
box_width = 0.5
box_height = 0.5

# calculate iterations across and down based on box dimensions and scan box
dimensions
iters_x = int((max_x - min_x) // box_width)
iters_y = int((max_y - min_y) // box_height) + 1


#starting from top left of bounding box
box_coord_y = max_y
box_coord_x = min_x


#initialize empty list to store points and thickness vals
points = []


#starting at min x and max y , top left of square
#scanning vertical rows and then moving horizontally to the right across the
square
for i in range (0, iters_x):
    for j in range(0, iters_y): #scanning down one column
        array1 = numpy.zeros((2000,3))
        count = 0
        for k in range(0, rows): # scanning through all vertices
            if matrix[k,0] >= box_coord_x and matrix[k,0] <= (box_coord_x +
box_width) and matrix[k,2] <= box_coord_y and matrix[k,2] >= (box_coord_y -
box_height):
                array1[count,0] = matrix[k,0] # x coord
                array1[count,1] = matrix[k,2] # y coord
                array1[count,2] = matrix[k,1] # z coord
                count = count + 1
        array1 = array1[~numpy.all(array1 == 0, axis=1)]
        if array1.size:
```

```
            max_z = numpy.max(array1[:,2])
            min_z = numpy.min(array1[:,2])
            thickness = max_z - min_z
            points.append(box_coord_x + (box_width / 2))
            points.append(box_coord_y -(box_width / 2))
            points.append(thickness)
        box_coord_y = box_coord_y - box_height
    box_coord_y = max_y
    # once you get out of j for loop
    box_coord_x = box_coord_x + box_width

cols = 3
points_fin = numpy.array([points[i:i+cols] for i in range(0, len(points), cols)])

# finding dims
dims_fin = points_fin.shape
rows_fin = dims_fin[0]

# print first 10 rows to check
print('Final array of x vals, y vals, and thickness: ')
print(points_fin)
print('\n')

#finding weighted average of x and y to approximate center point
countx = 0.0
county = 0.0
totalz = 0.0
for i in range(0, rows_fin):
    x_val = points_fin[i,0]
    y_val = points_fin[i,1]
    z_val = points_fin[i,2]
    countx = countx + (x_val * z_val)
    county = county + (y_val * z_val)
    totalz = totalz + z_val
mid_x = countx / float(totalz)
mid_y = county / float(totalz)
print(mid_x)
print(mid_y)


# plotting prosthesis thickness vs. gaussian
points3d = points_fin.copy()
for i in range(0, rows_fin):
    x_val = points3d[i,0]
    y_val = points3d[i,1]
```

```
    #gaussian function for indenting, x1 and y1 are center point
    x1 = mid_x
    y1 = mid_y
    z_val = (6 * math.exp(-((x_val-x1)**2)/(2*(8**2))) * math.exp(-((y_val-
y1)**2)/(2*(4**2))))
    points3d[i,2] = z_val

x1 = points3d[:,0]
y1 = points3d[:,1]
z1 = points3d[:,2]


#3D graph
fig = plt.figure(figsize = plt.figaspect(0.5))
ax = fig.add_subplot(1,2,1,projection='3d')
x = points_fin[:,0]
y = points_fin[:,1]
z = points_fin[:,2]
ax.scatter3D(x, y, z, c = 'r', marker = 'o', label = 'prosthesis')
ax.scatter3D(x1, y1, z1, c = 'b', label = 'gaussian')
ax.set_xlabel('X', rotation=150)
ax.set_ylabel('Y')
ax.set_zlabel('Z', rotation=60)


# finding RMS error
cut1 = box_width * 5.0
cut2 = box_width * 2
list_del = []
count = 0
for i in range(0,rows_fin):
    xs = points_fin[i,0]
    ys = points_fin[i,1]
    xs1 = points3d[i,0]
    ys1 = points3d[i,1]
    if (xs < (min_x + cut1)) or (xs1 < (min_x + cut1)) or (xs > (max_x - cut1))
or (xs1 > (max_x - cut1)) or (ys < (min_y + cut2)) or (ys1 < (min_y + cut2)) or
(ys1 > (max_y - cut2)) or (ys > (max_y - cut2)):
        list_del.append(count)
    count = count + 1


points_fin1 = numpy.delete(points_fin, list_del, axis=0)
points3d1 = numpy.delete(points3d, list_del, axis=0)
```

```
dims_del = points_fin1.shape
rows_del = dims_del[0]

#finding weighted average of x and y to approximate center point for updated set
of points
countx1 = 0.0
county1 = 0.0
totalz1 = 0.0
for i in range(0, rows_del):
    x_val = points_fin1[i,0]
    y_val = points_fin1[i,1]
    z_val = points_fin1[i,2]
    countx1 = countx1 + (x_val * z_val)
    county1 = county1 + (y_val * z_val)
    totalz1 = totalz1 + z_val
mid_x1fin = countx1 / totalz1
mid_y1fin = county1 / totalz1
print(mid_x1fin)
print(mid_y1fin)


# updated plotting prosthesis thickness vs. gaussian
for i in range(0, rows_del):
    x_val = points3d1[i,0]
    y_val = points3d1[i,1]
    #gaussian function for indenting, x1 and y1 are center point
    z_val = (6 * math.exp(-((x_val-mid_x1fin)**2)/(2*(8**2))) * math.exp(-
((y_val-mid_y1fin)**2)/(2*(4**2))))
    points3d1[i,2] = z_val

# 3D graph after trim
ax = fig.add_subplot(1,2,2,projection='3d')
x2 = points_fin1[:,0]
y2 = points_fin1[:,1]
z2 = points_fin1[:,2]
x3 = points3d1[:,0]
y3 = points3d1[:,1]
z3 = points3d1[:,2]
ax.scatter3D(x2, y2, z2, c = 'r', marker = 'o', label = 'Reconstructed values')
ax.scatter3D(x3, y3, z3, c = 'b', label = 'Known values')
ax.set_xlabel('X', rotation=150)
ax.set_ylabel('Y')
ax.set_zlabel('Z', rotation=60)
plt.legend(loc = 'upper right')
# plt.show()
```

```
# RMS Error calculation
dims_rms = points_fin1.shape
rms_count = dims_rms[0]
resids = 0.0
for i in range(0, rms_count):
    z4 = points_fin1[i,2]
    z5 = points3d1[i,2]
    val = z4 - z5
    val2 = val * val
    resids = resids + val2
num = resids / rms_count
rms = math.sqrt(num)
print('The RMS Error is: ', rms)

plt.show()
```

d) code to measure and graph thickness of two normal subtracted eyes

```
# Python program to measure thickness of two subtracted eyes

import csv
from pyexpat.errors import XML_ERROR_CANT_CHANGE_FEATURE_ONCE_PARSING
import numpy
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
from mpl_toolkits import mplot3d
import math

# field names
fields = ['X', 'Y', 'Z']

# name of csv file
filename = "measured-thickness2.csv"

# writing to csv file
with open(filename, 'w', newline='') as csvfile:
    # creating a csv writer object
    csvwriter = csv.writer(csvfile)

    # writing the rownames
    csvwriter.writerow(fields)
```

```
    #opening obj file and read line by line
    with open('anth-eyes-subtracted-solid.obj') as file:
        for line in file:
            content = line.split()
            #check if vertices line
            if content[0] == 'v':
                list = []
                for item in content:
                    if item != 'v':
                        list.append(item)
                    if len(list) == 3:
                        # writing the data rows
                        csvwriter.writerow(list)

#closing the write file
file.close()

#importing csv as data frame
df = pd.read_csv (r'measured-thickness2.csv')

#converting data frame to array
matrix1 = df.to_numpy()

print('Vertices stored in an array: ')
print(matrix1)
print('\n')

#getting dimensions of matrix
dims = matrix1.shape
rows = dims[0]
columns = dims[1]
print('Dimensions of array: ')
print(rows)
print(columns)
print('\n')


#sort matrix by x values
matrix = matrix1[matrix1[:, 0].argsort()]
print('Array sorted by x value: ')
print(matrix)
print('\n')

#find max values for x and y for bounds of scan box
max_x = numpy.max(matrix[:,0])
```

```
min_x = numpy.min(matrix[:,0])
max_y = numpy.max(matrix[:,2])
min_y = numpy.min(matrix[:,2])
print('Max and min values: ')
print(max_x)
print(min_x)
print(max_y)
print(min_y)
print('\n')

#finding thickness by 'box' method?
box_width = 0.5
box_height = 0.5

# calculate iterations across and down based on box dimensions and scan box
dimensions
iters_x = int((max_x - min_x) // box_width)
iters_y = int((max_y - min_y) // box_height)


#starting from top left of bounding box
box_coord_y = max_y
box_coord_x = min_x


#initialize empty list to store points and thickness vals
points = []


#starting at min x and max y , top left of square
#scanning vertical rows and then moving horizontally to the right across the
square
for i in range (0, iters_x):
    for j in range(0, iters_y): #scanning down one column
        array1 = numpy.zeros((500,3))
        count = 0
        for k in range(0, rows): # scanning through all vertices
            if matrix[k,0] >= box_coord_x and matrix[k,0] <= (box_coord_x +
box_width) and matrix[k,2] <= box_coord_y and matrix[k,2] >= (box_coord_y -
box_height):
                array1[count,0] = matrix[k,0] # x coord
                array1[count,1] = matrix[k,2] # y coord
                array1[count,2] = matrix[k,1] # z coord
                count = count + 1
        array1 = array1[~numpy.all(array1 == 0, axis=1)]
```

```
        if array1.size:
            max_z = numpy.max(array1[:,2])
            min_z = numpy.min(array1[:,2])
            thickness = max_z - min_z
            points.append(box_coord_x + (box_width / 2))
            points.append(box_coord_y -(box_width / 2))
            points.append(thickness)
        box_coord_y = box_coord_y - box_height
    box_coord_y = max_y
    # once you get out of j for loop
    box_coord_x = box_coord_x + box_width

cols = 3
points_fin = numpy.array([points[i:i+cols] for i in range(0, len(points), cols)])

# finding dims
dims_fin = points_fin.shape
rows_fin = dims_fin[0]

# print first 10 rows to check
print('Final array of x vals, y vals, and thickness: ')
print(points_fin)
print('\n')

# print average and max value of thickness
max_thick = numpy.max(points_fin[:,2])
mean_thick = numpy.mean(points_fin[:,2])
print('The maximum thickness is ', max_thick, 'mm')
print('The average thickness is ', mean_thick, 'mm')


#3D graph
fig = plt.figure(figsize = plt.figaspect(1))
ax = fig.add_subplot(1,2,1,projection='3d')
x = points_fin[:,0]
y = points_fin[:,1]
z = points_fin[:,2]
ax.scatter3D(x, y, z, c = 'g', marker = 'o', label = 'Noise')
ax.set_xlabel('X (mm)', rotation=150)
ax.set_ylabel('Y (mm)')
ax.set_zlabel('Thickness (mm)', rotation=60)

plt.legend(loc = 'upper right')
plt.show()
```