Differential 3D Scanning

Article *in* IEEE Computer Graphics and Applications · May 2017

DOI: 10.1109/MCG.2017.39

Ammar Hattab
Brown University
6 PUBLICATIONS **18** CITATIONS
SEE PROFILE

Gabriel Taubin
Brown University
231 PUBLICATIONS **11,417** CITATIONS
SEE PROFILE

Differential 3D Scanning

Ammar Hattab, Ian Gonsher, Daniel Moreno and Gabriel Taubin
Brown University

Abstract

In their creative process, designers employ techniques and strategies, moving from the abstract to the concrete through different physical and virtual means of representing form. Changes between the virtual and the physical are not always fluent. It would be a powerful tool for augmenting the design process if the designer was able to register and save each iteration of the physical and virtual models – a technique we call synchronization. In this article we propose a novel method called Differential 3D Scanning that allows designers to save and update their models throughout the entire design process. The key idea is to use 3D scanning to automatically detect changes in the physical model, and to reflect them into the virtual model. During this process, the use of digital fabrication and 3D scanning devices could accumulate reconstruction errors which could be narrowed to the changed regions by our method. This technique helps to speculate about unfinished great works of art.

Author Keywords: 3D Scanning; Change Detection; 3D Modeling; Fabrication; Physical Modifications.
ACM Keywords: H.5.2 User Interfaces. I.3.5.b Constructive solid geometry. I.4.8.l Surface fitting. I.2.10 Vision and Scene Understanding.

1. Introduction

Design is an iterative process. An essential feature of this process is the ability for the designer or designers to fluently translate abstract ideas into concrete form. In order to do this effectively, designers employ a variety of strategies for representing their ideas. These range from sketches to low-resolution models to CAD models to final prototypes. Each iteration is an opportunity to ask a question about a feature of the design, and these different modes of representations give the designers the tool set they need to understand, prototype, and critique their solution to any given design problem.

The rise of digital fabrication has revolutionized this process. And at the same time, many designers bemoan the loss of "hands on" craft based approaches to prototyping. Is something important lost when you cannot touch the thing you are making? Feeling something in your hands in order to give it form is a very different cognitive process than drawing something on a screen. It is necessary to make assumptions about form and function when the model is more abstract, as when it is designed on screen, than when it emerges from a direct engagement with materials. We still believe that we can get the best of the two worlds; digital and physical modeling. By combining them into different iterations.

Our intention is to suggest that new design tools can help designers more easily translate the abstract into the concrete, and more fluently move from the physical to the digital, and back to the physical. This is how ideas become real.

Moving from the digital to the physical can be done using a digital fabrication device like a 3D printer, while moving from the physical to the digital requires using a 3D scanning device. These devices work by making a copy of the whole 3D model. While the designer usually applies only a few changes to the 3D shape of the object in each design iteration. We argue that we don't need to operate on the whole 3D model, but only the relevant changes in each iteration. When one of the two models is modified, the changes only need to be transferred to the other model, a process we refer to as synchronization.

Previous work [4] showed how we can transfer changes made in the digital model to the physical model using a 3d printer and a milling machine. In this article, we show how to use a 3D scanner to synchronize the computer model to changes made in the physical model.

The main contributions of this work are:

* To propose using Differential 3D Scanning as tool for designers to update their digital model with any physical changes.
* Automatically detect physical changes in noisy 3D scans, and segment the reference model accordingly into changed and unchanged with smooth boundaries between the segments.

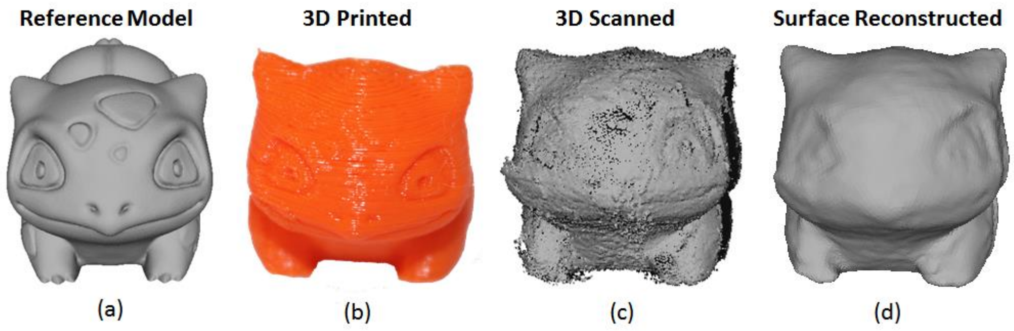

Figure 1: **Error sources in the proposed 3D design process**. (a) A high resolution reference 3d model. (b) 3D printed copy. All details that are smaller than the 3D printer resolution are lost. (c) 3D scanned copy with several errors because plastic is shiny and hard to scan. (d) 3D reconstructed model with important details smoothed

* To reflect the detected changes using 3D surface reconstruction while using the extracted boundary curves as constraints.
* To simplify the reverse engineering problem, by only considering the changed regions.

Errors are introduced and accumulated during the proposed 3D design process by digital fabrication, 3D scanning devices, and surface reconstruction algorithms [See **Figure 1**]. By only transferring the changes we could restrict those errors to a few regions of the 3D model that changed in the design iteration, and we reuse the rest of the digital model from the previous iteration.

There are many digital fabrication and 3D scanning technologies, each has its own problems and limitations. To use these devices for the design process –as proposed in this paper- we need them to provide a fast and user-friendly experience. So the designer can focus on his design and let the algorithm do the rest. And to encourage more designers to adopt this method we want this process to be more intuitive than digital modifications on a 3D modeling software. Thus it is important to select the right type of 3D scanning and digital fabrication devices and the right type of material that is easy to modify physically, and easy to scan. Nevertheless our method can handle the output of different types of devices with the right selection of parameters.

2. Completing Michelangelo's Sculpture

Before introducing our method details, we first show a design example using Michelangelo's statue.

Any design process is only as effective as the tools that allow the creator’s imagination to find form in material. Perhaps no better example exists in the history of Western Art than Michelangelo for translating the clarity of the artist’s vision into stone. In a famous quote attributed to the artist, he describes this process with a single sentence: “Every block of stone has a statue inside it and it is the task of the sculptor to discover it.”

It was this quote that was our inspiration for investigating how these design tools might be developed into creative strategies, which might be applied to the study of art history and the making of art more generally. We posed the question of whether it might be possible to finish an unfinished masterpiece by taking a scan of the original – one of Michelangelo’s “Slaves” (intended for the tomb of Julius II) - and to make a physical model, which could be completed by hand. In order to do this, we needed to be able to develop a process and the tools that would allow us to move fluently between the virtual model and the physical model. We needed to be able to both manipulate the model on the screen, and to manipulate the material by hand.

Michelangelo did not have the benefit of CNC mills, 3D printing, CAD, or scans. What he achieved, he did so with his own hands and eyes. The tools we are developing augment the creator’s hands and eyes, in order to speculate what could have been, and to give these speculations physical form. The process we’ve developed for this project allowed us to speculate about what these missing pieces could have looked like in the mind of the artist, and to modify the materials accordingly. **Figure 2** summarizes the experiment steps.

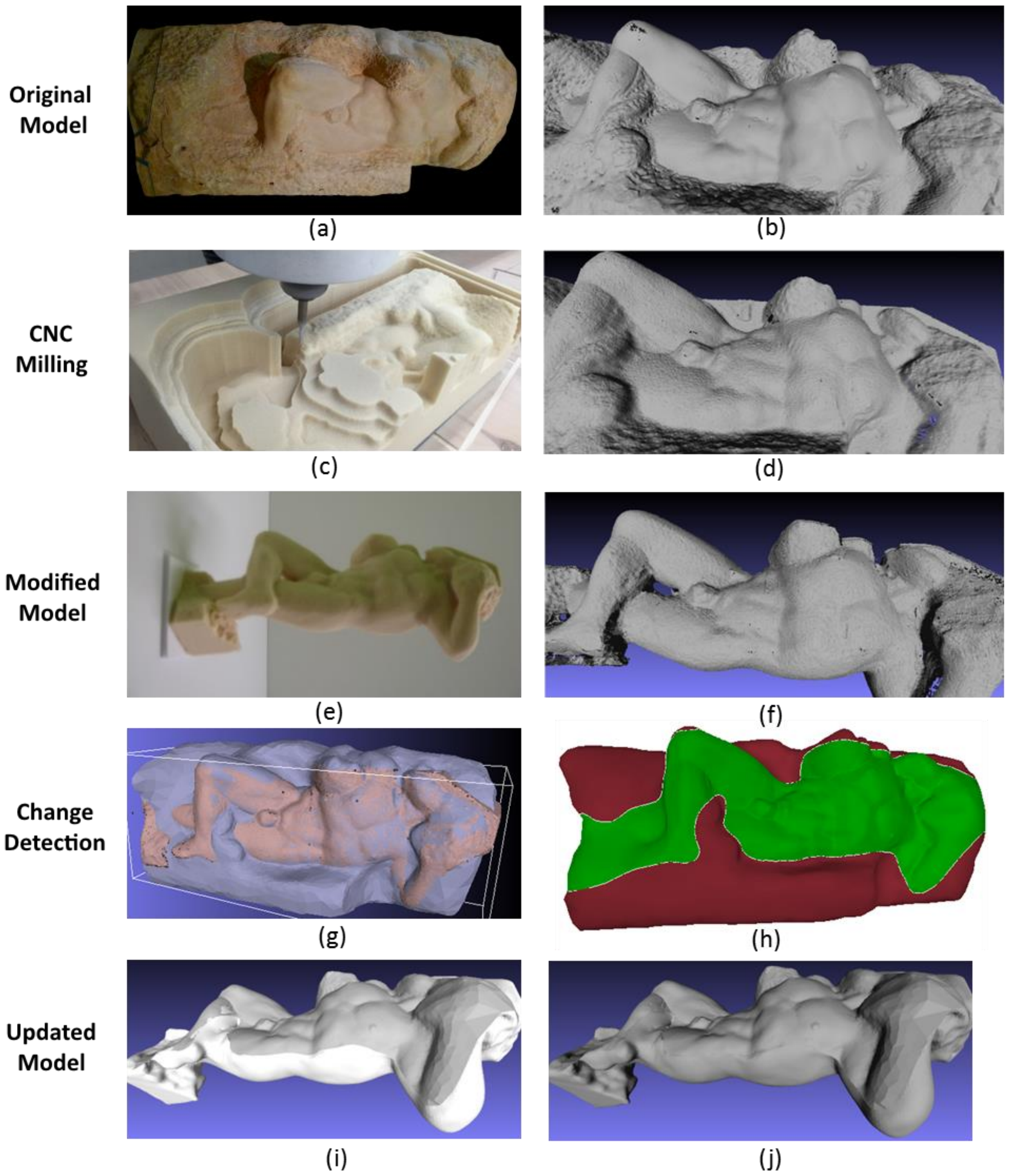

Figure 2: **Completing Michelangelo's unfinished work experiment steps**. (a) Original statue in Florence. (Source: Wikimedia Commons, 'Awakening Slave' by Michelangelo, 2011). (b) High resolution scan of the statue. (c) Using CNC carving machine to obtain a physical copy. (d) Scan after CNC carving. (e) Designer manual carving to finish Michelangelo's work. (f) Scan after manual carving. (g) Aligning the two models using ICP. (h) The reference model segmented into unchanged region in green, and changed region in red, separated by the smooth boundary curves. (i) Changed region deleted and replaced by the reconstructed changed region of the point cloud. (j) Regions merged.

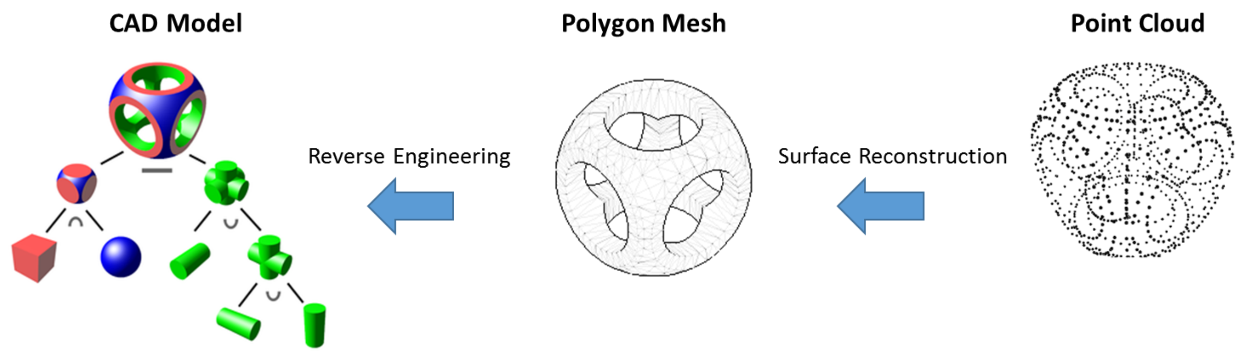

Figure 3: **3D Digital Representations**. Point Cloud is a collection of 3D points resulting from 3D scanning. Polygon Mesh is a collection of a vertices (3D points) connected by edges to form faces (Used for 3D printing). CAD models are based on ideal mathematical formulations, for example a tree of binary Boolean operators applied to simple geometry objects (Left Figure Source: Wikimedia Commons, Illustration of CSG tree, 2005).

At first we obtained a high quality scanned model of the statue from the digital Michelangelo project [10] which has around 2 million vertices. Then we used a milling machine to carve this model in foam. Then a designer used sculpting tools to carve and finish the unfinished parts of the model physically. At last we used a 3D scanner to scan the modified model, and we applied our method to get the final virtual model.

Here we notice that our method kept the original high resolution parts of the virtual model that were carved by Michelangelo. And only updated the parts that were modified by the designer and merged the two together. If we had tried to reconstruct the surface of the whole model, we would have lost many details from the original high resolution parts (Figure 2 part b), since 3D reconstruction tends to smooth out the result.

3. Method

Starting with a point cloud that contains the physical changes and a reference model. The designer could use some 3D modeling software to manually apply the changes to the reference model. For example MeshLab is a popular open source software that could be used for 3D alignment and surface reconstruction. Then the designer could use mesh editing software -like 3D Studio Max- to find the difference between the two models by applying a Boolean difference operation. The problem is that the two models match everywhere, except for a few regions. So many faces will be incident to each other, which is problematic when applying the Boolean operation. At the same time all small deviations due to fabrication and scanning errors will be detected by this method, and cannot be avoided.

Instead of this manual work, we propose an automated method that automatically aligns the two models, then finds and reflects the changes. The input to our method is a point cloud resulting from scanning the modified physical model and the 3D model of the previous iteration; which we refer to as “reference model”.

The digital representation of the reference 3D model varies between different applications. See [**Figure 3**]. It could be a polygon mesh, for example in some animations applications, or it could be a CAD model as used in mechanical and industrial applications. A polygon mesh is collection of a vertices (3D points) connected by edges to form faces. While CAD models are based on ideal mathematical formulations. There are several representations for CAD models, like parametric surface patches (NURBs) or Constructive Solid Geometry (CSG). CSG represents the model by using a tree where leafs are the simple geometry objects (sphere, cylinder, cube, etc), and the links are the binary Boolean operators applied to them (union, intersection, difference). We used CSG as the representation for CAD models for its simplicity.

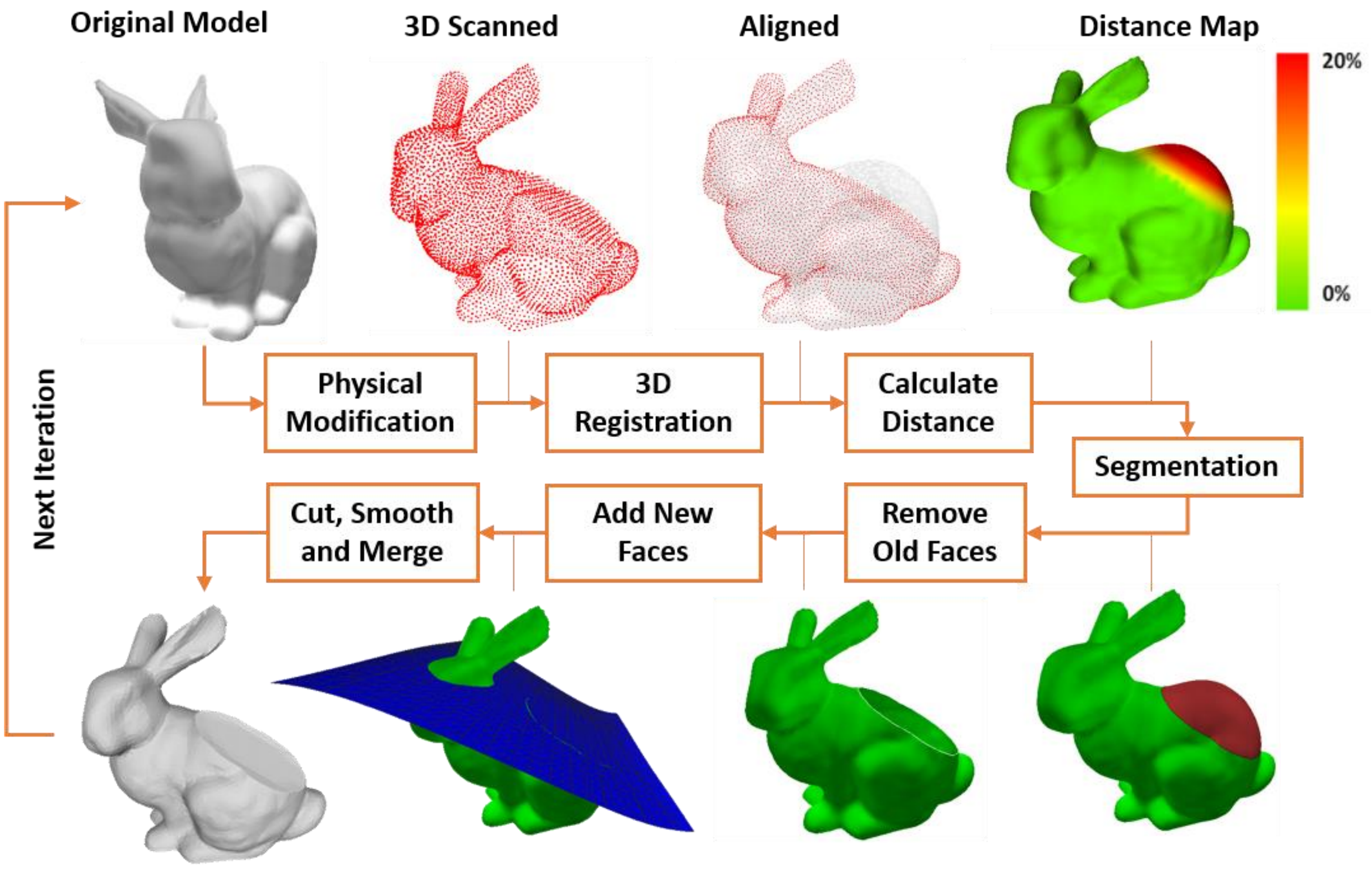

Figure 4: **Method steps for Polygon Mesh**. (a) **Physical Modification**: an example of design iteration, where the original model is fabricated, physically modified (cut) and scanned back resulting in a point cloud with changes. (b) **3D Registration**: Align the two models. (c) **Calculate Distance**: Change detection by measuring the distance between the two models, the color map shows the distance as a percentage of the diameter of the bounding box of the model. (d) **Segmentation**: Model segmented into changed region (red) and unchanged region (green). (e) **Remove old faces**: Changed region faces are removed. (f) **Add new faces**: The scanned Point Cloud is segmented as well into changed and unchanged regions, here we reconstructed the changed region only (blue) while using the boundary curves as strong constraints. (g) Then we **cut** the reconstructed surface using the boundary curves, and we **smooth** the result. The updated merged model could be used as an input for the next iteration.

The method for transferring the physical changes to the reference model consists of three steps:

- Aligning the two models
- Finding the changes
- Reflecting the changes

The first two steps are similar for polygon mesh and CAD models, while the third step requires surface reconstruction of the changes in the case of polygon mesh, and reverse engineering of the changes in the case of CAD models.

4. Polygon Mesh

For polygon meshes we propose an algorithm that segments the model into changed and unchanged regions with a smooth boundary curves that separates them. Then it uses these boundary curves in the surface reconstruction of the changed regions in the point cloud and merge them to the mesh. See [**Figure 4**].

4.1. 3D Registration

The physical object could be placed in an arbitrary position and orientation after modifications, and it needs to be aligned with the original virtual model in order to perform accurate comparison to find the changes, this process is called 3D registration. The inputs are a point cloud that resulted from scanning, and polygon mesh that represents the reference model, in our method we used the popular Iterative Closest Point (ICP) method [5]. Our method determines the initial alignment by aligning the principal axes of the two models.

4.2. Finding the Changes

The translation from the virtual to the physical and back from the physical to the virtual is not for free. Each device has a specific resolution, all details in the original model smaller than this resolution will be lost [See **Figure 1**]. For example, in milling machines, the drilling bit thickness determines the resulting resolution. While in FDM 3D printers, the extruder nozzle diameter controls the resolution. The same thing happens in 3D scanning devices.

Beside this systematic deviation from the reference model, noise errors will be introduced and accumulated by the digital fabrication and 3D scanning devices. And in some cases whole regions of the model might be deviated for example due to gaps in 3D scanning.

It is important for our method to distinguish between those deviations and the actual physical changes as applied by the designer. To do so, the applied physical changes must be larger than the largest resolution of the used devices so it can be detected.

Then when we detect changes we must use a threshold that is larger than the largest resolution of the used devices with some tolerance for the noise level and smaller than the smallest physical change. Knowing the specifications of the used devices could help in automating the selection of the threshold distance, which should be the same for the same settings.

The virtual and physical models match everywhere except in few regions, where they deviate from each other. Each of the changed regions has a part in one model and another corresponding part in the second model. And the size of those two parts might be

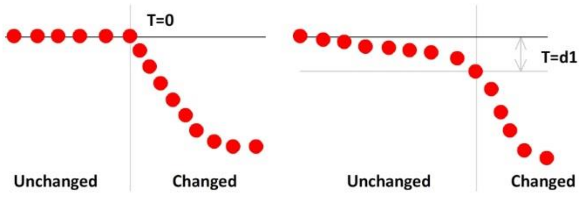

Figure 5: User specifies a threshold distance T after which we consider the new model (red point cloud) changed from the original surface (black line). Left: Clear change, we can use threshold T=0. Right: Smooth gradual change, the user decides a specific threshold distance for T.

different between the two models. For example, a small region in the reference model could be elongated in the point cloud. Establishing the correspondence between the changed regions in the two models might be hard. See [**Figure 6**]. So we decided to separate the process in two steps:

- Find the changed regions in the point cloud. Here we project each point in the point cloud to the reference model (polygon mesh) and calculate the Euclidean distance between the point and the projection point, and if that distance is larger than a threshold we mark this point as a change.
- Find the changed regions in the reference model (polygon mesh). To do that, we operate on the reference model vertices. And for each vertex we estimate the signed distance from this vertex to the point cloud using the non-convex hull surface (NCH) algorithm [6], and if this signed distance is larger than a threshold we mark this vertex as a change.

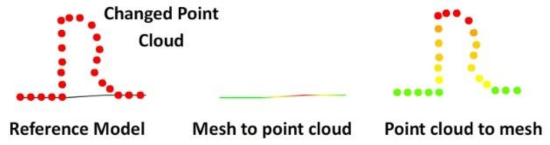

Figure 6: Two directions distance. Middle: distance from mesh to point cloud. Right: distance from point cloud to mesh.

We give the user the control to decide the threshold distance T after which the difference is considered a change:

$$|d| > T$$

The user can specify this threshold as a percentage of the diameter of the 3D model, or as an absolute value in the units of his 3D model whatever it be. See [**Figure 5**]. We also give the user the ability to ignore specific regions that might result from noise.

4.3. Finding the Boundary Curves

In the previous step, we calculated a signed distance field on the vertices of the reference model that represents the distance to the point cloud. For the unchanged regions part, the distance between the two models should be close to zero. While in the changed regions this distance is positive or negative (depending whether the change is to the inside or outside of the model). We need to find the iso-contour where this distance field goes from zero to negative or positive.

To extract those boundary contours we follow these steps:

1- Classify each vertex in the polygon mesh as changed or unchanged by using the distance threshold.
2- Find the edges of the mesh that has one vertex marked as change and the other marked as unchanged.
3- For each such edge extract the iso-vertex, where the boundary curve intersect the edge. We do that by using a sort of binary search. Where in each recursive step we evaluate the signed distance from the center of the edge to the surface and depending on the result we discard half of the edge and call the function again using the other half. We do that recursively until the two vertices of the edge approach each other.
4- We connect those iso-vertices to form one connected contour for each changed region that separates that region from the rest.

4.4. Fixing the Curve Displacement

Depending on the noise and the specified threshold distance, the resulting contour might not be smooth, and might be displaced from the actual real boundary by some distance [See **Figure 7**]. To fix this problem we run an optimization algorithm where we restrict the iso-contour vertices to stay on the surface while we minimize the distance between those vertices and the nearest K points of the changed region in the point cloud. At the same time we minimize the distance between the two vertices of every iso-contour edge to keep it smooth. We run gradient decent to minimize the following energy function for every contour vertex x until we get the desired result:

$$E(x) = \lambda_1 \sum_{(ij)} \left\| x_i - x_j \right\|^2$$

$$+ \lambda_2 \sum_{i} \sum_{j}^{K} \left\| x_i - p_j \right\|^2$$

$$p_j \in \{ NearestPoints(x_i, K) \}$$

The first term is a sum over the iso-contour edges that contain x, while the second term is a sum over K nearest changed points to x. We restrict the iso-contour vertices to stay on the surface by projecting them to the mesh after each optimization step. Although the contour vertices are restricted to stay on the surface, the contour edges are not. To fix this problem we project the contour edges onto the mesh.

The result is a smooth contour that approximate the real boundary between the changed and the unchanged regions. We cut the mesh along this contour, as we need to get rid of the old changed regions of the mesh, and replace them with the new reconstructed regions from the point cloud.

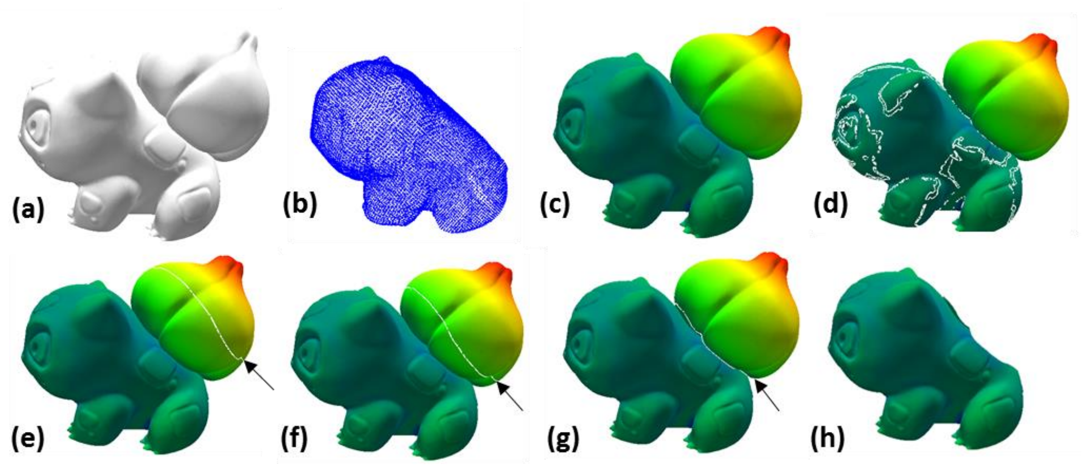

Figure 7: **Optimization procedure to fix the displacement of the boundary curve**. (a) Original reference model. (b) Scanned model of the 3D printed part that was cut by a cutting tool. (c) Color map of the signed distance field between the two models. (d) Using zero threshold (zero level set) will result in detecting many small changed regions (due to noise and errors). (e) Using high threshold will result in one displaced boundary curve. (f) Running the optimization algorithm will drag the boundary curve toward the right position. (g) The corrected position of the boundary curve. (h) The final model cut by the boundary curve.

Now we have every vertex classified as change or not-change and also the contour boundary that separates the changed and the unchanged regions. To get a complete segmentation of the mesh we need also to segment the faces. To do that we find incident faces for each vertex -except for the vertices along the contour boundary- and we mark the faces as changed if the vertex is changed, and vice versa. After the segmentation, we delete the faces marked as changed from the mesh in order to replace them with the new reconstructed regions from the point cloud.

4.5. Reflecting the Changes

To reflect the changes we need to apply some sort of Boolean operations (intersection, union, subtraction and difference) to remove and add new the changes to the original model.

There are several previous works [1] that applied Boolean operations to 3D polygon meshes. In our algorithm the changes happen on a point cloud instead. So we need to reconstruct the surface to get a polygon mesh. All previous studies reconstruct the surface of the whole point cloud before applying Boolean operations, without taking the advantage of the reference model.

We propose a surface reconstruction method that only need to reconstruct the changed regions of the point cloud, and takes into consideration the boundary curves that segments the model into changed and unchanged regions. So our surface reconstruction algorithm takes as an input a group of points (the unchanged regions) and a boundary curve that surrounds them.

There are a few studies that handled surface reconstruction for open surfaces with boundaries. For example Lin et al [2] used the shape boundaries for composition of multiple shapes. Also the peel algorithm [3] builds a Delaunay mesh using the sampled points while handling the boundary curves. But it suffers from robustness problems.

The input to this step is a list of contour lines that were extracted in the last step, and a list of changed points from the point cloud. And we need to reconstruct those changed points while using contour lines as boundary constraints to the surface reconstruction problem. There are several methods for surface reconstruction, but implicit volumetric methods work well especially in the presence of noise and holes in the model, for this reason we tried two implicit methods; Poisson [7] and SSD [8], while adding the boundary curves as constraints. Constraints could be added in a hard or soft way, we choose to apply soft constraints, by sampling the contour curves and adding those sample points to changed regions of the point cloud. In details we follow the following steps:

1- Construct a volumetric grid (we could use a regular grid or an octree) that contains both the changed points and the contour lines, and for each cell it must have at most one contour vertex.
2- Reconstruct the surface while using samples of the contour lines as point constraints (by simply adding those samples to the changed regions of the point cloud, the more samples we add, the more we pull the surface toward them.), now the reconstructed surface will pass nearby the boundary curves, but it will not stop at them, so we need to cut it.
3- Cut the reconstructed surface by the planes defined by the normals of the contour lines, and get rid of the remaining part.
4- Snap the vertices along the cut to the contour line vertices, and consolidate the vertices into ones.
5- Smoothing: minimize an energy function that smooths the reconstructed surface (first three terms) while better approximate the changed points (last term).

$$E(x,n) = \lambda_1 \sum_{(ij)} \left\|x_i - x_j\right\|^2 + \lambda_2 \sum_{(fg)} \left\|n_f - n_g\right\|^2 + \lambda_3 \sum_{(ij\_f)} \left(n_f^t(x_i - x_j)\right)^2 + \lambda_4 \sum_{f} \sum_{i\epsilon f} \sum_{k\epsilon \phi_f} \left(n_f^t(p_f - x_i)\right)^2$$

If we let vertices positions be constant, then the energy function becomes quadratic in the normals, and vice versa. So we need to run a few iterations of gradient decent with constant vertex position, then a few iterations with constant normal. Vertices on the contour are projected back to the contour after each smoothing step to keep them on the contour.

5. CAD Model and Reverse Engineering

For some applications the changes might need to be reflected to a CAD model. See [**Figure 8**]. Retrieving the CAD model from a point cloud or a polygon mesh is a complex operation that is called reverse engineering. The advantage of our approach is that we only need to reverse engineer a few regions and not the whole object.

The goal of reverse engineering is retrieve the designer original intent, which are represented usually using ideal shapes (ex: cylinder, sphere, etc). To do that we can search for those shapes in the point cloud using RANSAC for example [9]. But the result will be much better if we can segment the point cloud and fit these shapes directly to different segments. In general most reverse engineering algorithms have two main components:

1- **Segmentation**: to segment the model into surface patches (planes, spheres, quadratic surfaces, etc. usually based on surface curvature).
2- **Fitting**: where we directly fit geometric objects (planes, quadratic surfaces, free-form surfaces) to different surface patches.

Segmentation is the hard step and the result depends on the sensitivity parameter. In most cases the user has to select some regions to merge them manually. RANSAC-based segmentation could result in many false detected shapes and it could miss existing shapes.

In our case since we do our own segmentation and extract boundary Iso-curves, that makes things simpler. and if we restrict the problem such that the change is always of one type (one change at a time, so the user can for example only cut the object one cut, or drill one hole at a time...etc.). Then we don't need any further segmentation at all. This solves most of the problems that prevents automatic reverse engineering, but also restricts the application of our method to only one change at a time.

The problem then reduces to finding the best fitting plane, quadratic surface, or a free-form surface to the changed points. To do that we used the same algorithm as [9] and applied it to the group of changed points and the sampled points from the boundary Iso-curves, we found that adding these sampled points greatly enhances the result of fitting.

The result of the fitting step is a group of shapes (spheres, cylinders, cones…etc), to simplify the operation we used Constructive Solid Geometry (CSG) to design the reference CAD models, so we simply add the shapes resulted from fitting to the CSG tree by either a subtraction or addition operation. See [**Figure 8**].

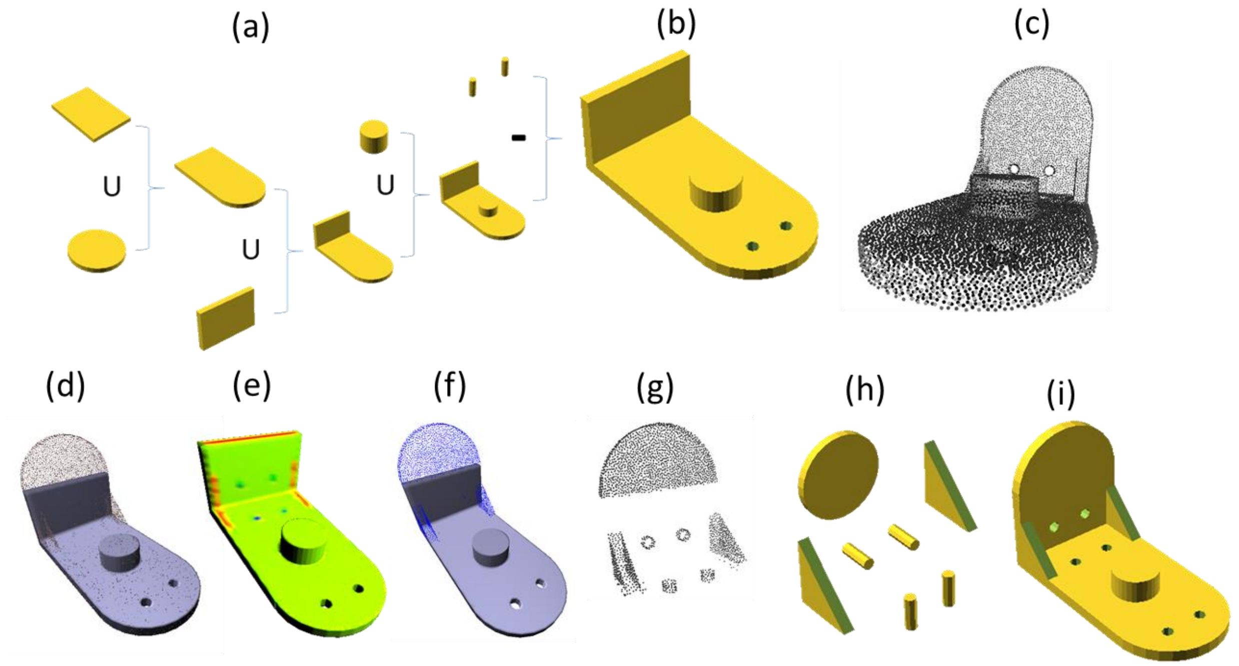

Figure 8: Method steps for reverse engineering CAD model changes. (a) CSG tree where different Boolean operations are applied to simple geometric shapes. (b) Resulting model. (c) A 3D scan of the modified object (holes drilled and support added). (d) The two models aligned. (e) A color map shows the signed distance between the two models. (f) Changed regions in the point cloud extracted. (g) Changed regions separated. (h) The result of fitting planes and cylinders to each changed region. These resulting shapes are added to the original CSG tree with simple union or difference operators. (i) The updated CAD model.

6. Limitations

Physical modifications are more intuitive for many designers when compared to digital modeling. For thousands of years, people have used various tools to modify physical objects. Lots of hand and power tools exist to perform operations such as cutting, sculpting, and carving.

Furthermore, many designers already integrate 3D scanning devices with their physical sculpting work. For example, in the car industry; they still use clay modeling to build a full-size clay models of the car. Then they use a 3D scanner to digitize the designed model. Oftentimes, designers need to design around or fit their design to existing objects, for example in the medical field, they use a 3D scanner to create digital models of body parts and teeth in order to make a perfectly fitting prosthetics and dentures. They also could be used to design fixes or extensions to existing broken and old parts. Or to scan parts of a larger model, for example Automobile customizers use 3D scanning to scan the existing part of a car that they want to customize, ensuring that the customization piece will fit seamlessly.

More recently, 3D printers are also being employed as the starting point for the physical sculpting process with the invention of Cx5 sculptable 3D printing material [11]. Cx5 is perfect for this approach, since it can be easily modified, and easily scanned (since it is fully opaque). Designers start with digital modeling (or 3D scanning), then they 3D print their model using the Cx5 sculptable material. Then they have the ability to hand-sculpt the finest details onto their 3D prints.

In this paper we build on this existing approach of combining physical design with digital fabrication and 3D scanning devices. We expect our approach to be more easily adopted by designers who already employ 3D scanning and digital fabrication devices in their work. So the real condition for it to be adopted is how easily the designer could employ these devices in his work compared to the use of a digital software. With wider spread of 3D scanning and printing technologies, more designers will become familiar with them, and adopt our approach.

Another limitation is to 3D scan shiny and translucent objects, like most of 3D printed plastic parts. On the other hand, there are opaque 3d printing material that could be used to overcome this problem.

Also 3D scanning and digital fabrication (for example: 3D printers) are slow in general, which may limit the number of design iterations using them. For 3D scanners to get an accurate model of the object, we need to scan it from different viewpoints to cover all occluded parts, which take even a longer time. Otherwise, important details could be lost. For instance, in our experiments, several drilled holes were missed by the scanner. One way to address this is by using a hierarchal approach to scanning, where we make a faster low resolution scan of the object, and once we find the changed regions, we can use a slower high resolution scanning technology to accurately scan them.

7. Next Steps

The fluent translation from the virtual to the physical and from the physical to the virtual will have many implications in the near future. By giving artists, art historians, designers, engineers, and many other creative professionals better tools to navigate between these spaces, new possibilities will emerge.

Moving forward, we intend to build upon the work outlined in this paper. We believe we can further refine these processes while working with partners in these fields to design the tools that can help them in their work.

8. Acknowledgment

The work described herein was partially supported by a Brown Fellowship, and by NSF grant IIP-1500249.

9. Conclusion

We presented a method to detect the 3D differences between a scanned model (point cloud), and a reference model (polygon mesh or CAD model), and then to reflect those changes to the reference model.

The method reuses the reference model with its higher accuracy, and saves the designer time by reconstructing only the small changed regions.

References

[1] Pavić, Darko, Marcel Campen, and Leif Kobbelt. "Hybrid booleans."Computer Graphics Forum. Vol. 29. No. 1. Blackwell Publishing Ltd, 2010.

[2] Lin, Juncong, et al. "Mesh composition on models with arbitrary boundary topology." IEEE transactions on visualization and computer graphics 14.3 (2008): 653-665.

[3] Dey, Tamal K., et al. "Isotopic reconstruction of surfaces with boundaries." Computer Graphics Forum. Vol. 28. No. 5. Blackwell Publishing Ltd, 2009.

[4] Teibrich, Alexander, et al. "Patching Physical Objects." Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology. ACM, 2015.

[5] Arun, K. Somani, Thomas S. Huang, and Steven D. Blostein. "Least-squares fitting of two 3-D point sets." IEEE Transactions on pattern analysis and machine intelligence 5 (1987): 698-700.

[6] Taubin, Gabriel. "Non-convex hull surfaces." SIGGRAPH Asia 2013 Technical Briefs. ACM, 2013.

[7] Kazhdan, Michael, Matthew Bolitho, and Hugues Hoppe. "Poisson surface reconstruction." Proceedings of the fourth Eurographics symposium on Geometry processing. Vol. 7. 2006.

[8] Taubin, Gabriel. "Smooth signed distance surface reconstruction and applications." Iberoamerican Congress on Pattern Recognition. Springer Berlin Heidelberg, 2012.

[9] Schnabel, Ruwen, Roland Wahl, and Reinhard Klein. "Efficient RANSAC for point-cloud shape detection." Computer graphics forum. Vol. 26. No. 2. Blackwell Publishing Ltd, 2007.

[10] Levoy, Marc, et al. "The digital Michelangelo project: 3D scanning of large statues." Proceedings of the 27th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co., 2000.

[11] Adam Beane. “Cx5 Sculptable Filament Kickstarter Video.” Online video clip. YouTube. YouTube, 24 August 2016. Web. 9 October 2016.

10. Biographies

Ammar Hattab (armmar_hattab@brown.edu) is a PhD student in Brown University in his fourth year, working with Prof. Gabriel Taubin. He received a B.S. degree in Computer Engineering from Yarmouk University, Jordan, in 2008. He received a master's in Computer Engineering from Brown University in 2014. Ammar's research mainly focus on the integration of 3D scanning and 3D printing. And how to enhance existing 3D scanning techniques. While focusing on how to use that to enhance the design process. His research interests also include geometry processing and computer vision. Ammar is interested in making his own things. Among many things he made; he built a 3d milling machine from scratch during his undergraduate study. Prior to joining Brown; Ammar worked as a software developer around the world in California, Jordan, Malaysia and Senegal for leading companies including Adobe, eSense and T-FORCE.

Ian Gonsher (ian_gonsher@brown.edu) is an Assistant Professor of Practice in the School of Engineering and Department of Computer Science at Brown University. His current teaching and research focus on developing new strategies for design, and enriching the creative process more generally. This work takes a human centered and critical approach to projects that reach across disciplines and contexts, establishing a theoretical framework for the creative process, and applying theses insights to the design of projects that have social impact. This work is diverse and multidisciplinary, including but not limited to recent projects that have examined "situated robotics", design for the developing world, design for incarcerated populations, and design for early childhood education. He holds an MFA from the Rhode Island School of Design, and BFAs in Art History and Industrial Design from the University of Kansas. More information about his work can be viewed on his website: http://gonsherdesign.com/

Daniel Moreno (daniel_moreno@brown.edu) received a B.S. degree in Computer Science from Universidad Nacional de Rosario, Argentina, in 2011. He received his Ph.D. degree in Computer Engineering from Brown University, Providence, RI, USA, in 2016. His research interests are precision 3D scanning for metrology applications, structured light and phase shifting algorithms, and digital geometry processing. His work experience includes internships at Evolution Robotics, Inc., NVIDIA Corp., and Cognex Corp.

Gabriel Taubin (gabriel_taubin@brown.edu) is Professor of Engineering and Computer Science at Brown University. He earned a Licenciado en Ciencias Matematicas degree from the University of Buenos Aires, Argentina, and a Ph.D. degree in Electrical Engineering from Brown University. In 1990 he joined the IBM Research Division, where he held various positions, including Research Staff Member in the Exploratory Computer Vision Group, Research Staff Member in the Visualization, Interactions and Graphics Group, and Research Manager of the Visual and Geometric Computing Group. In 2003 he joined Brown University as a faculty member. Taubin has held various visiting positions, including: Visiting Professor of Electrical Engineering at the California Institute of Technology; Visiting Associate Professor of Media Arts and Sciences at MIT; Visiting Professor of Computer Science at the School of Exact and Natural Sciences, University of Buenos Aires, Argentina; and Visiting Professor of Engineering at Universidad Nacional del Sur, Buenos Aires, Argentina. Prof. Taubin serve two terms as Editor-in-Chief of the IEEE Computer Graphics and Applications Magazine starting in 2010, he is a current member of the Editorial Board of the Geometric Models journal, and has served as associate editor of the IEEE Transactions of Visualization and Computer Graphics. Prof. Taubin was named IEEE Fellow for his contributions to the development of three-dimensional geometry compression technology and multimedia standards, won the Eurographics 2002 Günter Enderle Best Paper Award, and was named IBM Master Inventor. Prof. Taubin is a current member of the Fulbright Specialist Roaster, and a Fulbraigh Specialist grantee. He has contributed to the field called Digital Geometry Processing with methods to capture 3D shape, for surface reconstruction, geometric modeling, geometry compression, progressive transmission, signal processing, and display of discrete surfaces. The 3D geometry compression technology that he developed at IBM was incorporated into the MPEG-4 standard in the late 90's, and became an integral part of IBM products.