MITSUBISHI ELECTRIC RESEARCH LABORATORIES
https://www.merl.com

DamageEst: Accurate Estimation of Damage for Repair using Additive Manufacturing

Gambill, Patrick; Jha, Devesh K.; Krishnamoorthy, Bala; Raghunathan, Arvind; Yerazunis, William S.

TR2025-158 November 05, 2025

Abstract

Repairing damages in high-value parts using additive processes can be more efficient than using state-of-the-art high-skilled manual processes. We describe DamageEst, an efficient computational geometry framework for detecting and estimating the damage volume (DV) and the inner damage surface (IDS) using point cloud data (PCD) of damaged parts and their original 3D models. DamageEst identifies points in PCD on the IDS to reconstruct the IDS. It then encloses the reconstructed IDS and original part in a slightly scaled up background mesh, from which the DV is reconstructed using Boolean operations. DamageEst also enables targeted overestimation of damage for repair using additive manufacturing followed by milling to guarantee high surface quality. Prior methods scale exponentially in both time and memory, while DamageEst scales in polynomial time and memory. DamageEst enables precise identification and representation of damages with minimal human intervention.

Solid Freeform Fabrication Symposium 2025

Mitsubishi Electric Research Laboratories, Inc.
201 Broadway, Cambridge, Massachusetts 02139

DamageEst: Accurate Estimation of Damage for Repair using Additive Manufacturing

Patrick Gambill[1,2], Devesh K. Jha[1], Bala Krishnamoorthy[2],
Arvind Raghunathan[1], and William S. Yerazunis[1 *]

Abstract

Repairing damages in high-value parts using additive processes can be more efficient than using state-of-the-art high-skilled manual processes. We describe DamageEst, an efficient computational geometry framework for detecting and estimating the damage volume (DV) and the inner damage surface (IDS) using point cloud data (PCD) of damaged parts and their original 3D models. DamageEst identifies points in PCD on the IDS to reconstruct the IDS. It then encloses the reconstructed IDS and original part in a slightly scaled up background mesh, from which the DV is reconstructed using Boolean operations. DamageEst also enables targeted overestimation of damage for repair using additive manufacturing followed by milling to guarantee high surface quality. Prior methods scale exponentially in both time and memory, while DamageEst scales in polynomial time and memory. DamageEst enables precise identification and representation of damages with minimal human intervention.

keywords: 3D metal printing, automatic part repair

1 Introduction

Automatic repair of lost-material damages in high value parts could be cost-effective compared to the current practice of using highly skilled laborers who carefully inspect and add metal to the abraded, corroded, or missing-fragment damage sites. However, performing accurate damage repair of lost-volume parts requires an accurate estimate of the damage volume followed by possibly an additive process to add metal in the damage volume. In most of these applications, achieving high accuracy while maintaining practical compute costs and use of machine time are critical. We propose a method that can perform high accuracy, lost-volume damage estimation while achieving cheaper computational costs. The proposed method makes use of a point cloud scan of the damaged part and the mesh model of the undamaged part to estimate the damage volume. We describe DamageEst, an efficient computational geometry framework for detecting and estimating the damage volume (DV) and the inner damage surface (IDS) using point cloud data (PCD) of damaged parts and their original 3D models.

The proposed estimation method makes use of a 3D solid model of the original, undamaged part and a point cloud scan of the damaged part to construct an inner damage surface, together

*: Correspondence author, 1: Mitsubishi Electric Research Laboratories, 2: Washington State University

with its boundary. We then make use of a background mesh that fully encloses the mesh of the original undamaged part. The boundary vertices of the IDS is then projected on this background mesh which allows us to split the background mesh into a part containing the damage volume and another part containing the undamaged part. Then, a more accurate estimate of the damage volume is obtained by Boolean intersection between the original solid model and part of the background mesh that contains the damage. Since we make use of the solid model of the original model, we can preserve the sharp features (like edges and corners) on the outer surface of the damage volume. The sharpness and accuracy of the inner damage surface will depend on the resolution of the point cloud scan obtained for the damaged part. The proposed method is tested on objects and damages of different shapes and size. We also compare the proposed method against an octree-based decomposition method and show that our method can achieve better computational results. We provide complexity analysis of each step of the proposed method to show that we can achieve polynomial memory and time complexity.

The problem of damage volume calculation is one of efficiency rather than one of possibility. The point cloud of even a relatively small part such as a plastic toy dragon may have over a million points; doing an $n * n$ brute-force comparison is not a realtime possibility and some thought is therefore required.

Over 30 years ago, Atallah et al approached the problem of combinatorial explosion in comparison of convex polygons [2], and Attene approached fast meshing. [3]. A similar issue applied to fast alignment of 3D shapes, as considered by Besl and McKay [4]. A second presumption is that the CAD model can be trivially converted to an accurate point cloud. Indeed, the reality is that an accurate, detail-preserving conversion is not quite trivial, but it can be done. [6], [7].

Although the CAD model may generate a ”perfect” point cloud (every vertex and edge is exactly represented by points in the cloud and no point in the cloud is not mappable to a point on the surface of the CAD model), the damaged part's point cloud will inevitably contain noise points, either points significantly displaced from the actual measurable surface, or simply artifacts of the scanning process. Fortunately, the DBSCAN algorithm provides not only density-based denoising, but also clustering of the point cloud results enabling a part with multiple missing sections to be properly processed [9].

In related work, Kanishka and Acherjee presented a systematic review of current AM-based repair and remanufacturing techniques [11].

But only limited details of frameworks for damage estimation were presented. Lingling Li et al. [13] developed a reverse engineering based approach that combined 3D surface data collection of damaged part with a reconstruction of nominal model (if model for the original part is not available) followed by the extraction of the patch to be repaired by AM using Boolean operations. The appropriate toolpath was then generated depending on whether the repair is to performed using additive or subtractive approaches. While details of implementation in practice are presented, computational efficiency and scaling behavior of the framework were not investigated.

Lan Li et al. [12] presented results on repairing damaged gear tooth using AM and quality of the repaired part was evaluated using microstructure analysis and Vickers hardness test. Models for both damaged and nominal parts were constructed using reverse engineering and are aligned using a two-step process using surface alignment followed by convex hull centroid alignment. While one could potentially generalize this framework for the repair of other types of objects or parts, details were presented only for the case of gear teeth. Furthermore, complexity of the computational steps in the framework were not investigated.

The work that is arguably the closest to ours is the framework of DUOADD presented by Perini et al. [14] for damage identification and modeling followed by toolpath generation for repairing the damage volume using the AM process of direct laser deposition (DLD). This framework models the 3D volume of the part (original and damaged) using a voxel representation that is handled using an octree data structure. Unfortunately, the computing time for building the octree grows exponentially with the depth level, a measure of the resolution of the model. Storing and handling the octree data structure for the entire volume of the part can also be memory intensive.

2 Problem Description

In this section, we briefly describe the problem we study in this paper (see Figure 1 for an illustration). We assume that we have access to a point cloud scan of the damaged part and a 3D model of the original part. Our objective is to propose a method that can accurately recover an estimate of the damage volume for arbitrary damage sites on the original part. In some applications, the parts of interest could have sharp edges and corners. Several state-of-the-art methods for volume/surface reconstruction from point cloud scan of damage sites assume smoothness such that these sharp geometric features could be lost during the reconstruction process. One objective of the proposed method is to preserve these sharp features in the damage estimate so that the damaged parts could be repaired properly. Furthermore, we also assume that we have the knowledge of the geometry of the original object via the CAD model of the object. Since the proposed damage estimation is based on the point cloud scan of the damaged part, we assume that the scanned point cloud does not have any gaps larger than the threshold λ for the desired accuracy. For generality, we also consider the scenario where there could be multiple damage sites in the part.

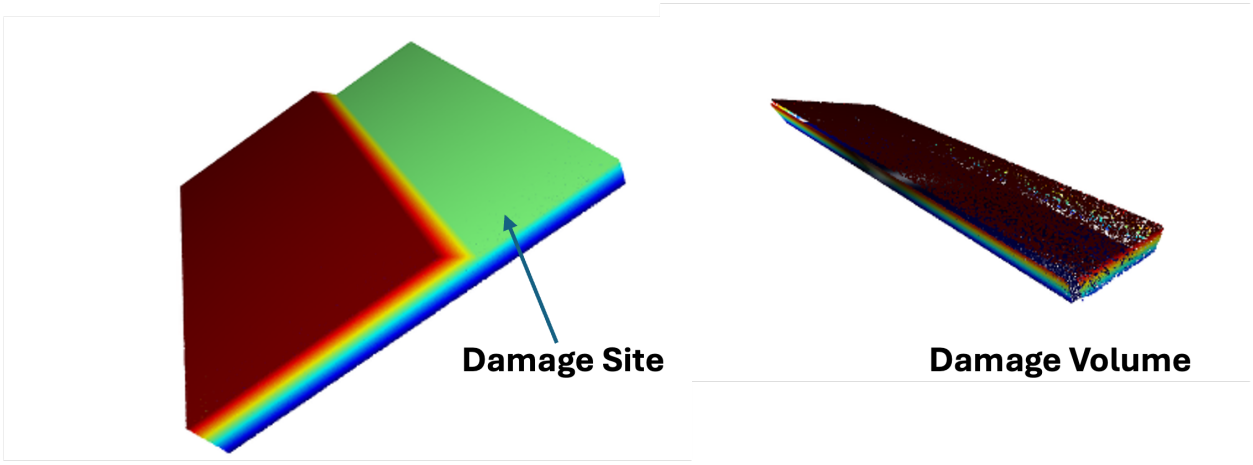


Figure 1: A part with the damage site in the form of a wedge (left) and the estimated damage volume (right).

3 Proposed Method

The key steps of our framework termed DamageEst are illustrated in Figure 2. We first list the sequence of all steps in Section 3.1 and then describe the details in Section 3.2.

3.1 DamageEst: Steps in the Algorithm

The sequence of steps involved in the proposed method are listed below:

1. Create a point cloud from the original CAD model of the undamaged object.
2. Scan the damaged workpiece with a 3D scanner, generating another point cloud.
3. Align the two point clouds.
4. Find the inner damage surface (IDS) point cloud using the Hausdorff distance.
5. Reconstruct the inner damage surface (IDS).
6. Find the boundary of this inner damage surface (IDS).
7. Create a background mesh with spacing distance λ.
8. Map the inner damage surface boundary onto the background mesh.
9. Connect the pieces of the mapped boundary to form a cycle.
10. Split the background mesh at the mapped boundary cycle.
11. Connect the undamaged component to the cycle of the inner damage surface to form an accurate and watertight triangular mesh.
12. Take the Boolean intersection of the repaired damage component with the original CAD model to estimate the damage volume.

3.2 DamageEst: Detailed Description

Consider the damaged part shown in Figure 1. The core idea of the proposed method is to first estimate the inner damage surface of a damage site (see Figure 2). This surface can then be used to extract the damage volume for the part using the CAD model of the original part using a background mesh that contains the original model of the object and Boolean operations (see the steps shown in Figure 2).

The first step for our proposed method is to estimate the point cloud corresponding to the inner damage surface (IDS) of the damaged part. This is estimated using a set difference method between the point cloud of the original workpiece and the damaged piece. We assume we have access to the solid model of the original workpiece. We sample a point cloud from the surface of the original CAD model. The vertices in this sampling are all exact and are precisely on the surface of the CAD model. It is important that the density of points sampled from the correct CAD model is adequately high. Ideally no gap should exist larger than a threshold of accuracy λ. Some such gaps may exist; but if these gaps are rare enough, the error points generated subsequently will be discarded as noise. Additionally, it is desirable to have sample points exactly on the sharp edges and corners, again with spacing not larger than λ. It is noted that the accuracy of the estimated damage volume will depend on the density of the point cloud sampled in this step.

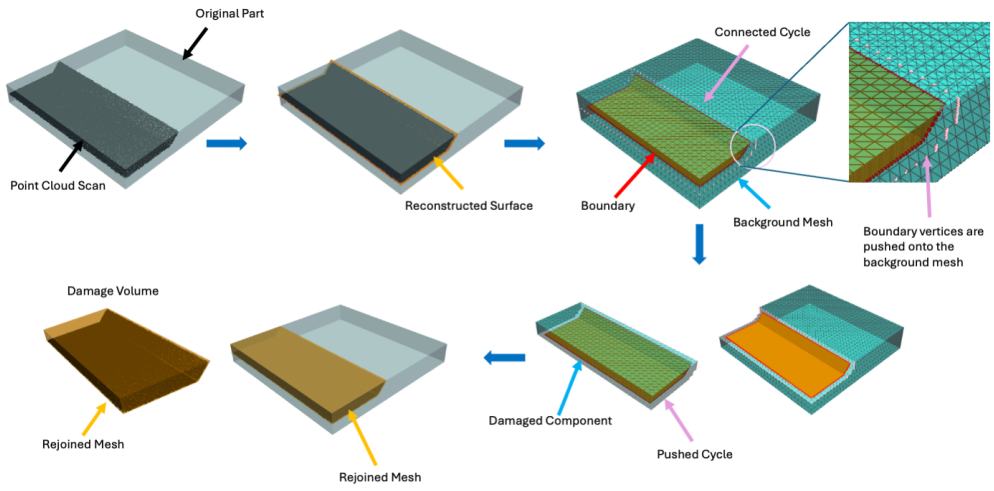


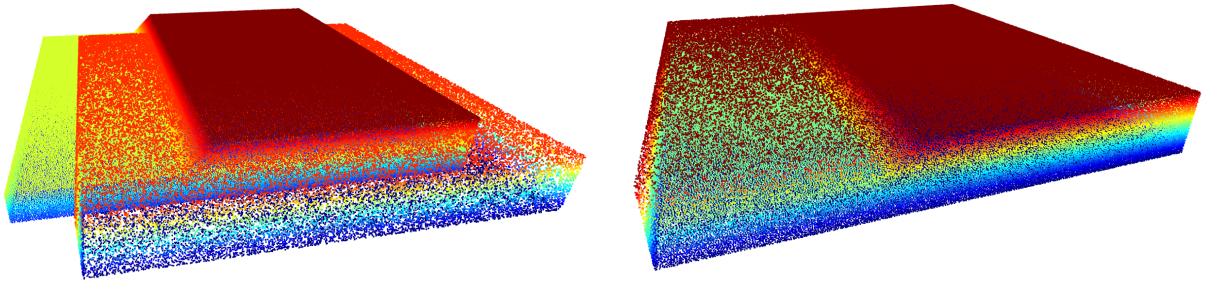
Figure 2: Key steps in the DamageEst framework. DamageEst first reconstructs the inner damage surface, and makes use of a background mesh to split the original model into two halves: one containing the damage volume and the other one containing the undamaged part. The final damage estimate is then obtained by taking Boolean intersection of the damaged mesh with the original solid model of the object.

We assume that we do not have access to a solid model of the damaged work piece (in general,it will be difficult to create a 3D model of a damaged workpiece as that would need accurate information describing the damage). However, we can scan the damaged workpiece to generate a second point cloud that includes the damaged region of the workpiece. Because the scanner is a physical device that has a nonzero noise floor, this sampling will be somewhat inexact and may not capture every edge and vertex exactly.

At this point it is important to note that the two point clouds should be aligned so as to avoid any miscomputation of distances between the original and the damaged point cloud. This is a common procedure in computer vision and could be done using standard techniques. An example of the misaligned and aligned point clouds for the objects before and after the damage in shown in Figures 3a and 3b, respectively.

In the case we use the exact triangulated CAD model directly below, we still may desire to use the point cloud version for this alignment step because algorithms for aligning point clouds are usually simpler and faster.

Once we have both the point clouds aligned, we estimate the point cloud for the IDS of the damaged work piece using a set difference method. We take the Hausdorff distance of each point in the damaged scan to the undamaged point cloud, and classify the points according to whether their Hausdorff distance is greater or lesser than the threshold of accuracy λ as mentioned in the first step. Alternatively, one could find the Hausdorff distance with the exact triangulated CAD model directly and classify each points based on the threshold of accuracy λ, as above. In either case, if the Hausdorff distance is below the threshold λ, we presume that the distance from the correct shape is due solely to noise in the scanning sensor system and the workpiece is undamaged at that position. In contrast, points in the damaged workpiece point cloud with a Hausdorff distance greater than λ away from the points in the undamaged point cloud are considered part of the damaged surface.



(a) Point clouds before alignment

(b) Aligned point clouds

Figure 3: The damaged and original point clouds before and after alignment; the two point clouds are individually colorized in Z to assist visualization and color is not otherwise meaningful in this figure.

We present the following discussion assuming a single damage site. An example scenario with multiple damage sites is presented later for completeness.

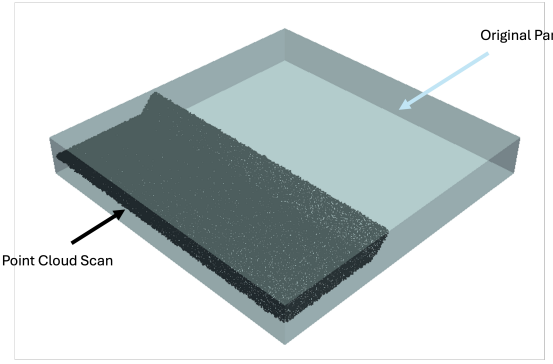


Figure 4: The target damage site and original part.

After alignment of the point clouds, the next step is to reconstruct the inner damage surface from the inner damage surface point cloud. This could be done with an existing algorithm such as ”reconstruct_surface()” from the open source ”pyvista” library in Python. Some reconstruction algorithms work better for certain damage types, but the above algorithm has done well with our tests. The reconstructed surface for the damage site considered above is shown in Figure 5.

The ”Ball Pivoting” algorithm (from the ”Open3d” python library) can work well for uniform density surfaces, but if the scan is non-uniform, it usually will not work as well as pyvista’s ”reconstruct_surface()”. The ”Poisson Reconstruction” algorithm (also from the ”Open3d” python library) cannot be used since the reconstructed surface must have a distinct boundary as described in the next steps. The ”Poisson Reconstruction” algorithm always returns a watertight surface (a closed volume, which has no boundary),

In the next step, we estimate the boundary of the inner damage surface mesh. We estimate the boundary by finding all the boundary vertices (see Figure 6). We find the boundary vertices by finding all the boundary edges. To do this, we consider each edge in the inner damage surface mesh and count the number of triangles associated with the edge. If an edge is a member of exactly one triangle, it is classified as a boundary edge. (edges having membership in zero triangles can be

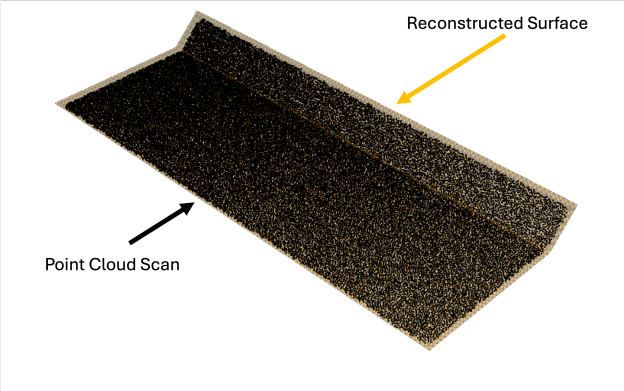


Figure 5: The reconstructed inner damage surface (IDS) using the point cloud scan of the damage part.

considered to be noise artifacts and discarded; edges with membership in two triangles are typical interior edges, and membership in three or more triangles indicates a bug in the mesh algorithm.) Given the boundary edge data, we can now classify any vertex in a boundary edge as a boundary vertex. The idea of boundary vertices is also shown in Figure 6. The boundary for the damage workpiece example is shown in Figure 7. The fit of the reconstructed IDS with its boundary vertices on the original solid model is shown in Figure 8. This also shows that the reconstructed IDS may have some noise protrusions compared to the original solid model– however, we will correct them in later steps of the method.

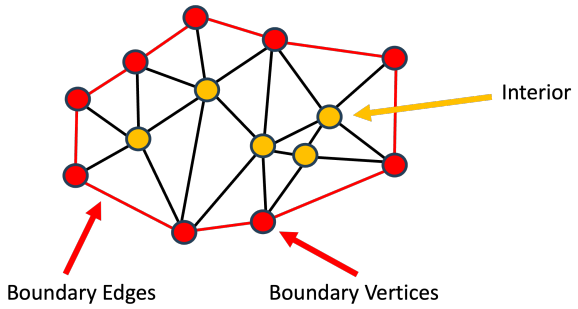


Figure 6: We find the boundary of the surface by finding edges in the mesh which are bound only by a single triangle. A simplified scenario is shown here.

Once we estimate the boundary of the inner damage surface, we would like to find a bounding mesh that contains the damage volume (see Figure 9). The main idea is to find an overapproximation of the damage volume which can help us estimate an accurate damage volume by performing Boolean operations with the 3D model of the original part. In order to ensure this, we create a background mesh. The background mesh will be a mesh that fully encloses the undamaged CAD model, the undamaged point cloud, and the inner damage surface. Each of these objects must be completely interior to the background mesh. For the example under consideration, we show the background mesh in Figure 9.

This background mesh supplies us with a properly filled 3D volume of appropriately spaced vertices and corresponding edges, with no holes or other irregularities. Since we control the cre-

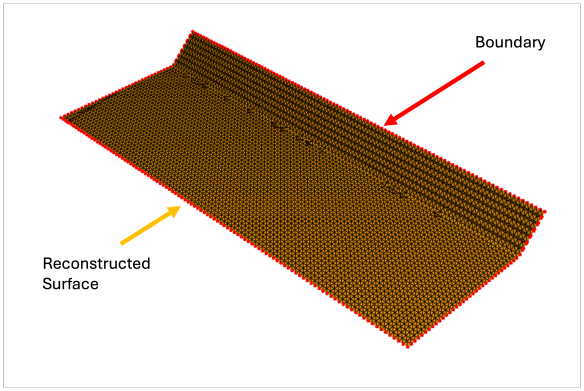


Figure 7: The boundary of the inner damage surface for the part considered in this example found by the definition provided above. The boundary points are marked in red.

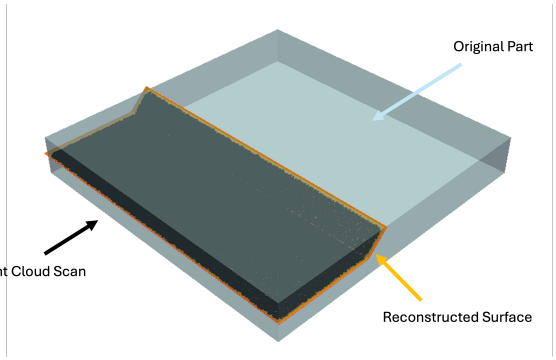


Figure 8: The inner damage surface may have noise-induced protrusions compared to the original part.

ation of the background mesh, we can make the node spacing anything we desire. In practice, having the background mesh node spacing be approximately equal to the threshold of accuracy λ is usually adequate.

The background mesh need not be based on the initial part shape. In principle, the background mesh might be a space-filling 3D mesh with a size just slightly larger than the working volume of the actual 3D printer, and with a maximum node spacing equal to the finest resolution available of the printer's deposition head. In this case, (and in principle) this is the only background mesh ever needed, so computing it once and saving it (perhaps on non-volatile media, and perhaps only once for a particular version of a mass-produced repair system) is not unreasonable.

The decision to use a precomputed maximum size mesh versus a smaller (bounding box) mesh is an interesting trade-off. Computationally, using the full maximal mesh is expensive unless careful programming is used, such as dividing the vertices and edges of the maximum mesh into subsections and only using those subsections that the CAD model and the damaged workpiece point cloud intersect can greatly decrease the time required versus the naive implementation that examines every vertex and edge in the maximum mesh. Given that the precomputed mesh can be invariant with respect to the CAD model of the workpiece, it is possible that the actual enumeration of the precomputed mesh is unnecessary and a functional "generator" can produce all of the desired precomputed mesh vertices and edges. This generator can either be a software function, or it could be implemented on a GPU or other array-style computational engine, or it could be implemented as a computational element on an FPGA or ASIC.

Another alternative exists for generating the background mesh. The previously described methods of generating the background mesh all create a 3D space filling mesh with nodes both inside and outside of the part volumes. Most of these nodes will never be accessed during the computation and are subsequently thrown away. In this sense they represent unnecessary computation. To minimize this wasted computation, it is possible to generate a background mesh that is a 2D watertight triangular mesh embedded in 3D space. In this case, the 2D geometry conforms to the correct part shape, but is spaced "outward". For parts with complicated geometries and concave sections, a scaled up version of the original mesh tends to do well, but can lead to larger run times. This scaling can be computed by expanded translating all of the vertices along the surface normals by a scaling factor or by computing the surface of the Minkowski sum of the original surface with a small scaling volume, and then repeatedly subdividing this mesh until no edge is longer than the threshold of accuracy λ desired.

For simpler geometries, it's often better to use a generic mesh generator that yields a mesh that better approximates the original object and the damaged object. The particulars depend on the original object, and some shapes work better than others. If the original CAD model is similar in geometry to a rectangular prism, a bounding box works well. If the original CAD model is rotationally symmetric, a capsule or cylinder tends to work well.

It is important the background mesh contains enough vertices to ensure the no edge is significantly longer than the threshold of accuracy, λ. If the number of vertices is too small, then a linear subdivision can be used to add intermediate vertices to the mesh. A few other subdivision methods, such as "butterfly" and "Loop" subdivision (both from the "pyvista" library in python) work well for the scaled background mesh, but tend to do poorly when the background mesh contains sharp corners. Additionally, adaptive subdivision methods, such as "subdivide_adaptive()" function in "pyvista" sometimes work better for meshes with non-uniform triangulations.

We now map each vertex from the boundary of the inner damage surface onto the background

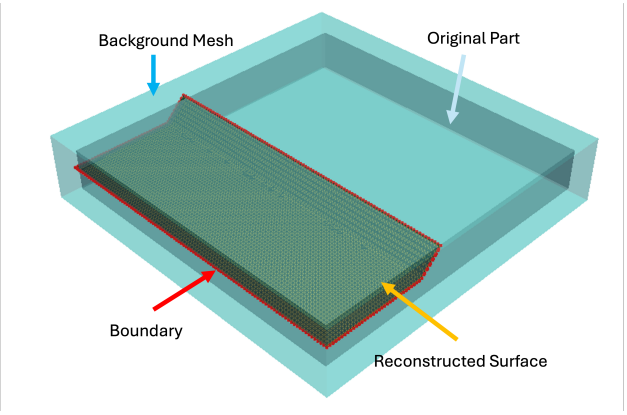


Figure 9: The background mesh encloses the damage surface and the original part.

mesh. Each boundary vertex is mapped to the vertex in the background mesh with the closest Euclidean distance. As described before, adaptive gridding methods can reduce the runtime of this step-and-compare greatly. This is shown for the damage we consider here in Figure 10.

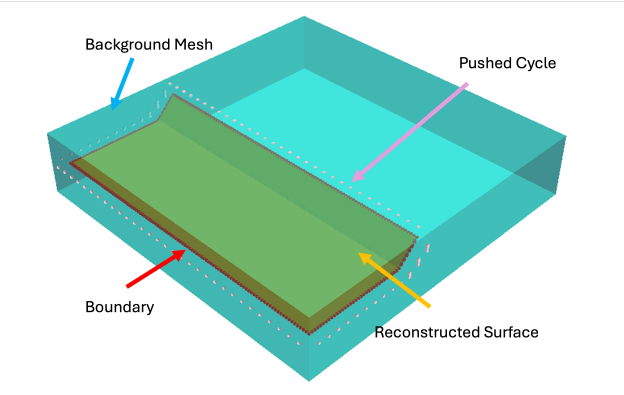


Figure 10: The boundary of the reconstructed IDS is pushed to the background mesh. This is done by creating a mapping from the boundary vertices to the mesh using Euclidean distances.

Once we have the boundary mapped to the background mesh, we would like to split the mesh into parts, one of which encloses the damage volume. To do this, we connect the pieces of the mapped boundary to get a cycle. For any two vertices in the boundary that share an edge, we find the shortest path between their mapped vertices in the background mesh (as the background mesh may be of finer resolution than the mesh of the inner damaged volume). The resulting cycle is shown in Figure 11.

Take each of the vertices contained in a shortest path, and add them to a list. We will call this list the pushed cycle, and it represents the edge of the repair deposit as described in the fine-resolution background mesh. In the pushed cycle, we occasionally get multiple disjoint components. This will occur if the boundary of the inner damage surface is not one contiguous cycle.

For example, consider a typical set of dress shoes after being worn a few times; the toe section of the sole will be somewhat abraded, then a long section under the instep will be undamaged, and finally the heel section will be abraded. This will result in two disjoint pushed cycles: one around the toe zone, and another, smaller one around the heel.

If multiple cycles exist, there are a few methods to deal with this. The first method is to consider each cycle individually for the steps below. We repeat the steps below for each cycle. In practice, this usually isn't helpful. In the event that the cycles are very small, the cycles tend to be artifacts of the mesh reconstruction algorithm. These small holes in the reconstructed mesh are detectable as the count of nodes and edges will be very small (vaguely, 10 or fewer nodes or edges). Such small holes will be repaired in a later step and should be ignored.

If a workpiece has a hole worn all the way through the material, then there will be multiple large cycles at the corresponding reconstructed surface. Again, the largest cycle is the important one and should be connected to the background mesh. The other large cycles will be closed properly in a later step. If all of the cycles are pushed to the background mesh, it will slow down the computation of later steps without a significant change to the final result.

In the event there are multiple larger cycles, only one needs to be connected to the background mesh for the algorithm to work properly. The other holes will be repaired in a later step, and joining multiple cycles will lead to many damage components in the background mesh. Sorting through the extra components requires more human intervention and can be avoided.

The second method is to throw out all cycles except the largest (by node count) cycle. Alternatively, the largest cycle could be found by considering the largest number of edges or the largest linear extent. Node count is quickest to compute, and usually gives the same result as the other methods.

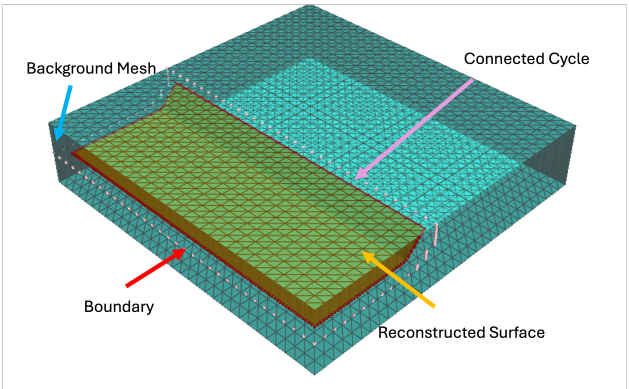


Figure 11: Consecutive points in the pushed cycle and connected by paths to create a cycle on the background mesh.

We use the projected cycle to split the background mesh to get two component sub-meshes (the mesh is split at the cycle). To do so, we remove all vertices contained in the pushed cycle as well as any edges and triangles in the background mesh containing these pushed-cycle vertices. One connected component will correspond to the damage site, and the other will correspond to the undamaged material of the workpiece. Usually the smaller component corresponds to the damage site, but this is highly dependent on the nature of the damage. The two components for the given example are shown in Figure 12 and 13.

Connect the component corresponding to the damage site to the inner damage surface. To do this, create an edge between each vertex in the boundary cycle to the vertex in the background mesh that is mapped to it. Then, create an edge from each vertex in the cycle to the image of the the successive neighbor in the background mesh. Finally, fill in all triangles that appear with these

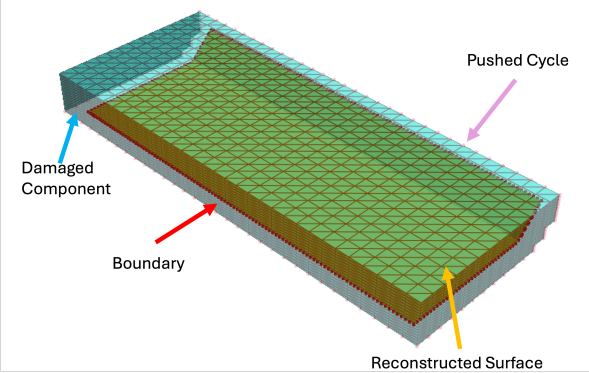


Figure 12: The damaged component of the split background mesh

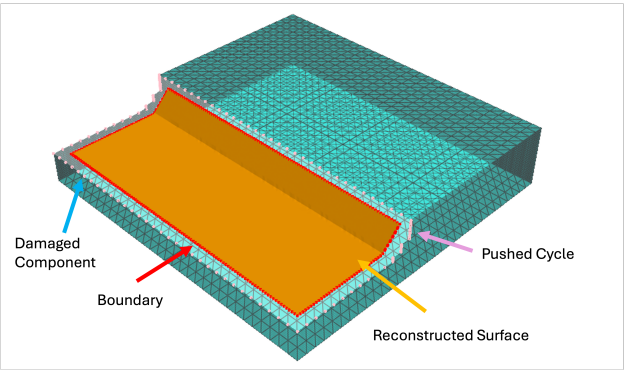


Figure 13: The undamaged component of the split background mesh

joined edges (see Figure 14 for the example damage volume).

Mathematically speaking, let $f : V \rightarrow W$ be the mapping function from the boundary vertices of the inner damage surface to the background mesh. Create an edge $\{v, f(v)\}$ and an edge $\{v, f(v+1)\}$.

For all vertices W_v created in the pushed cycle between $f(v)$ and $f(v+1)$, add the edges in

$$\{\{v, f(w)\}|w \in W_v\}.$$

To make sure the result is watertight and well defined, run a mesh repair algorithm, such as the algorithm "repair" from the open source "pymeshfix" library in Python. Any mesh fix algorithm should patch any holes, align any normal vectors, and fix any self intersection issues.

Regardless of what method is chosen, it is essential the chosen method results in a watertight mesh. If a mesh is not watertight, it is not possible to compute Boolean operations.

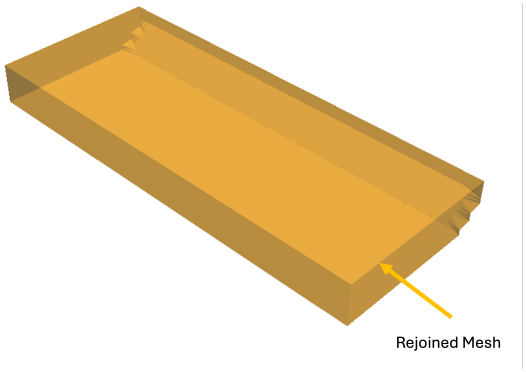


Figure 14: The rejoined mesh

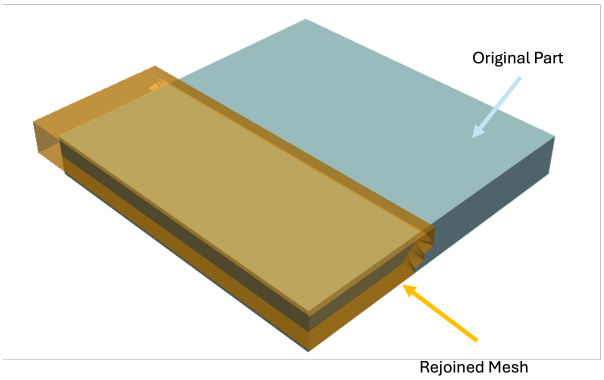


Figure 15: The rejoined mesh and original part

Take the Boolean intersection of the repaired damage component mesh with the original CAD model to get the damaged surface. This step pushes the extended size of the background mesh back to the correct surface of the workpiece, and preserves any complicated geometry in the original CAD model's surface (see Figure 15).

Unlike existing methods of reconstruction, this allows highly detailed surfaces to be preserved. Rather than the pushed surface or the noisy 3D sensor data surface reconstruction, the original

CAD-modeled detailing is restored exactly because those points are passed through without any modification or low-pass filtering.

This yields a highly accurate model for the damaged volume of the workpiece. Every point in the to-be-deposited repair is either the most accurate estimate of the damage surface obtainable from the 3D scanner, or is derived directly from the original CAD model without any modification.

If the original CAD model is not available, it is possible to estimate the original part, by reconstructing the surface of the undamaged scan using a reconstruction algorithm, such as those mentioned in step 5. If the reconstructed surface is not watertight, a mesh repair algorithm such as the aforementioned ”repair” function must be run. Subsequently, taking the Boolean intersection of the repaired damage component with the reconstructed surface to get the damage volume estimate.

To recover the undamaged volume of the workpiece, take the Boolean difference of the original CAD model (or the reconstructed original surface) with the damage volume estimate. Alternatively, apply step 10 to the component that doesn't correspond to the damage volume, then take the Boolean intersection with this second component and the original CAD model (or the reconstructed original surface). The estimated damage volume and its fit with the original solid model are shown in Figure 16 and 17 respectively. As could be seen, we can get rid of the protrusion noise that was not present in the damage volume in the final estimate. Another point to note is that we are able to retain the sharp geometric features on the outer surface (like the corners and the edges) in the reconstructed surfaces.

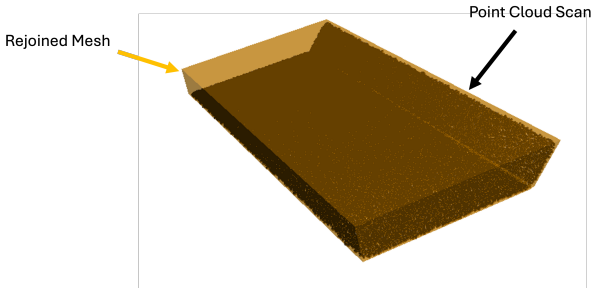


Figure 16: The estimated damage volume for the damage site considered in the example.

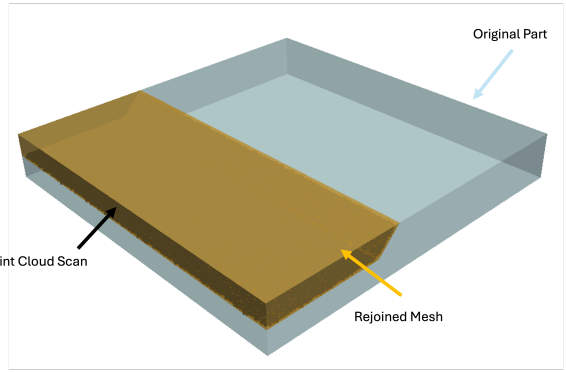


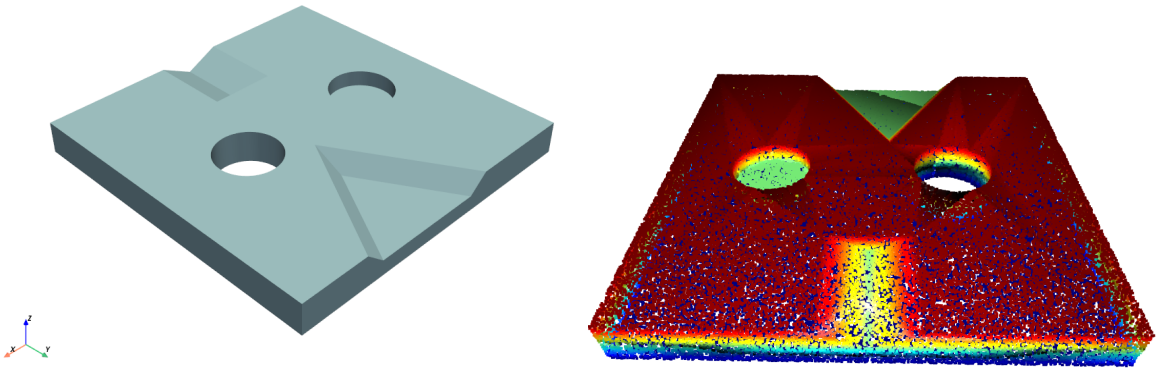
Figure 17: The estimated damage volume with the original part for the considered example.

Once we have the final damage mesh, we save the reconstructed surface as an STL file and

send it to slicing software and then the 3D printer.

3.3 Parts with Multiple Damage Sites

In cases, where we might have multiple damage sites, we first estimate the number of damage sites using a clustering algorithm and then, repeat the previously described damage estimation technique. An example part with multiple damage sites and the corresponding point cloud scan is shown in Figure 18a and 18b. At this point, it is important to note that the inner damage point



(a) Damaged part

(b) Point cloud scan of damaged part.

Figure 18: A part with four damage sites used for describing the proposed algorithm and a point cloud scan of the same.

cloud has been estimated without the knowledge of number of damage sites while using the set difference method. Thus, before we estimate the damage volume, we need to estimate number of damage sites, if there are more than one site. This is to make sure that we can estimate the damage volume at each of the sites independently. Once we have compared all points in the damaged workpiece point cloud and partitioned them into representing damaged and undamaged zones on the workpiece being repaired, we can discard the points representing undamaged surfaces. The union of all points classified as representing damage points forms the inner damage surface point cloud.

We cluster the points in the inner damage point cloud using a clustering algorithm, preferably the open-source DBSCAN algorithm [9]. DBSCAN classifies each point into one of n possible clusters or into noise. The advantages of using DBSCAN over other clustering methods are twofold. First, the number of clusters does not need to be known in advance. For a damaged workpiece, such as a lattice or grill, there could be many isolated damage sites and counting them may be impractical; forcing the algorithm to join or split independent damage sites to conform with a predetermined but incorrect n leads to an incorrect repair. Second, DBSCAN can automatically detect outlier points (caused by sensor noise) and will not include them in any clusters. This acts as a noise filter without any additional steps. With a noisy 3D sensor signal, filtering is essential.

The DBSCAN algorithm starts by finding seed points. These are points that possess at least k neighbors within a 3D sphere of radius ϵ. Each of these seed points as well as their ϵ neighbors will be added to the same cluster. Denote the set of neighbors of point v as N_v.

For each point $w \in N_v$ find the ϵ neighborhood of w and add each of the points in N_w to the cluster. If a point is found that is within ϵ of k or more points in more than one cluster, then the two clusters are actually one cluster and we relabel both clusters as a single cluster (a speed optimization step is to not relabel all the points in this phase; instead keep a table of relabelings and after all points are labeled go through the point list linearly applying the relabel table).Repeat this seed-growing process until either all points are in a cluster or do not have at least k neighbors no more than ϵ distance away and therefore are considered to be noise.

Picking the optimal value of k and ϵ is nontrivial, but in practice, setting ϵ to the same value as the damage threshold λ and setting k to values between 5 and 20 have worked well in our testing.

A heuristic method for picking ϵ is to sort each of the points from the lowest average distance to its k-nearest neighbors to the highest average distance to its k-nearest neighbors. In this plot, the distance where the sharpest curvature appears is usually chosen for ϵ (more mathematically speaking, where the second derivative is maximized). One method for computing this is to find

$$\bar{m} = \text{argmax}_m \{f(m-1) - 2f(m) + f(m+1)\}$$

where $f(m)$ is the average distance of point m from its k-nearest neighbors, and where the points are sorted from least to greatest average distance. The value of ϵ can be set to $f(\bar{m})$.

This will remove any noise points and the clusters will give the individual damage sites for each component of the damage surface. If there are multiple damage sites, simply repeat the steps below for each damage site to get the full damage volume. An example case for the detected clusters (or damage sites) and the noise in the point cloud is shown in Figure 19. Once the clusters

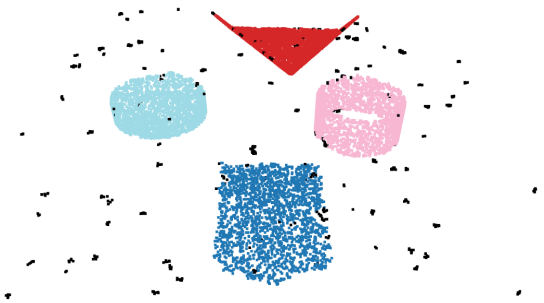


Figure 19: Each damage site forms a cluster. The black points are noise and will be ignored.

are detected, we estimate the damage volume for each site. The damage estimates superimposed over the damaged part model is shown in Figure 20.

4 Results

4.1 DamageEst: Computational Complexity

We present a complexity analysis of the key steps in our framework (as listed in Section 3.1), each of which runs in polynomial time and memory. Hence, we conclude that DamageEst runs in polynomial time using polynomial memory. Here we used the same step numbering as in Section 3.1.

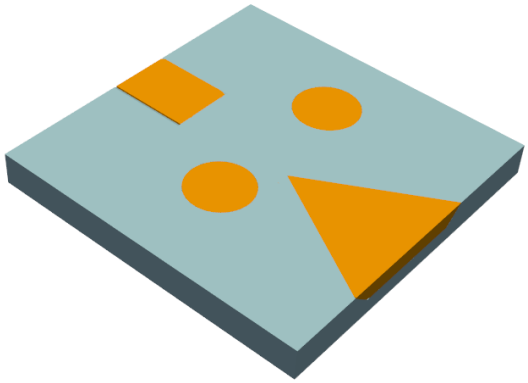


Figure 20: The final estimation for all damage sites

For clarity of presentation, we list the variables used (**Var**) and their descriptions.

Var	Description
N_d	Number of points in the damage scan
N_w	Number of points in the workpiece point cloud
N_i	Number of points in the IDS
T_i	Number of triangles in the IDS
λ	Distance threshold for background mesh
T_λ	Number of triangles in output mesh as determined by λ
N_b	Number of vertices on the boundary
N_{bm}	Number of vertices on the background mesh
E_{bm}	Number of edges in the workpiece mesh
T_r	Number of triangles in the repaired mesh
T_o	Number of triangles in the original part mesh

3. **Align the two point clouds:** The ICP method [4] for 3D registration runs in $O(N_d N_w)$ time.
4. **Find IDS using Hausdorff distance:** As the original mesh is a triangulated surface, this can be done in linear time [2].
5. **Reconstruct the IDS:** We used the method of Hoppe et al. [10], which runs in $O(N_i^2)$ time.
6. **Find the boundary of IDS:** This step can be implemented in $O(T_i)$ time. [8].
7. **Create a background mesh with distance threshold λ:** Delaunay mesh refinement algorithms [6] can be used for this step, which run in $O(T_\lambda \log T_\lambda)$ time where T_λ is the number of triangles in the output mesh as determined by the threshold λ. But this step usually runs in linear time in practice [6]. For our current implementation, we used the simple barycentric subdivision repeated a few times (each round runs in $O(T)$ time).
8. **Map the IDS boundary onto the background mesh:** This can be done naively in $O(N_b N_{wm})$ time

9. **Connect pieces of mapped boundary to form a cycle:** This can be done by running a shortest path algorithm for each vertex in the boundary to find nearest vertices to close the loop. While the shortest path algorithm can theoretically be run in almost-linear time [5], using Dijkstra's algorithm in practice is almost as efficient [1]. Hence, this step runs in $O(N_b(E_{bm} + N_{bm}\log(N_{bm}))$ time.
10. **Split background mesh at boundary cycle:** This step involves examining the neighborhood of each vertex in the boundary cycle and can be done in $O(N_b)$ time.
11. **Connect undamaged component to the boundary cycle to form watertight mesh:** Similar to the previous step, this step also involves examining the neighborhoods of vertices in the boundary cycle, and runs in $O(N_b)$ time. More generally, such repair operations can be implemented efficiently using lightweight or local approaches [3].
12. **Boolean intersection of the repaired damage component with original CAD model:** While the worst case complexity of Boolean intersection is $O(T_r T_o)$. This step typically runs much faster in practice, as most triangles in the two meshes will not intersect [7].

Note that DamageEst will always compute up to floating point precision. The accuracy of our method is affected only by the surface reconstruction method and the selected background mesh. On the other hand, the voxel method DUOADD [14] depends on the depth of the octree and will scale exponentially in memory as well as runtime.

4.2 Instances of DamageEst Applications

To illustrate the versatility of DamageEst, we present its damage estimation computed on parts with various geometries: a gear with damaged teeth in multiple pieces (Figure 21), a cylinder with damages from two sides (Figure 22), and a half ring with damage (Figure 23). In these examples, we see that DamageEst is able to recover the damage volume accurately.

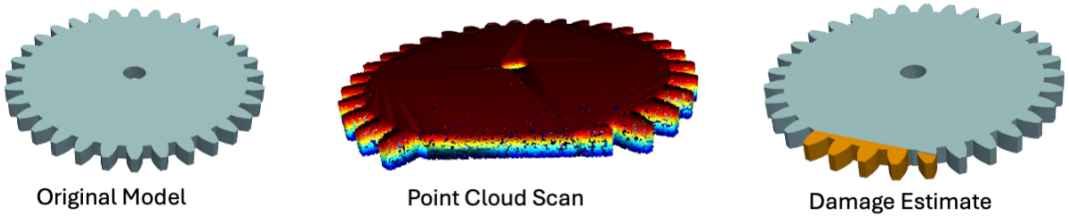


Figure 21: Application of DamageEst on the gear with missing teeth and the estimate obtained by DamageEst.

5 Discussion

Repairing damages in high-value parts using additive processes can be more efficient than using state-of-the-art high-skilled manual processes. We describe DamageEst, an efficient computational

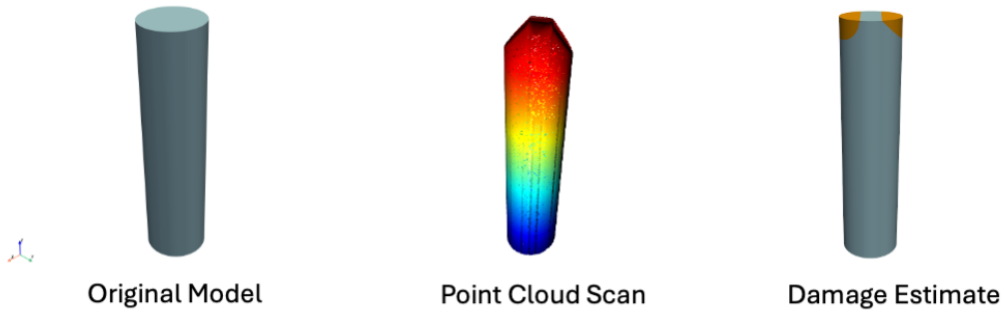


Figure 22: Another example where we show damage estimation at multiple sites where DamageEst can recover accurate damage estimates.

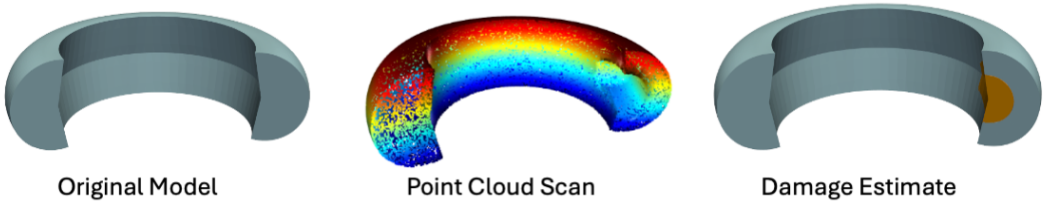


Figure 23: Result of DamageEst on a half torus which is a challenging work piece due to its non-convex geometry.

geometry framework for detecting and estimating the damage volume (DV) and the inner damage surface (IDS) using point cloud data (PCD) of damaged parts and their original 3D models. DamageEst identifies points in PCD on the IDS to reconstruct the IDS. It then encloses the reconstructed IDS and original part in a slightly scaled up background mesh, from which the DV is reconstructed using Boolean operations. DamageEst also enables targeted overestimation of damage for repair using additive manufacturing followed by milling to guarantee high surface quality. The proposed method is verified on various objects with different geometry for damage. Furthermore, we verify that we are able to also preserve sharp features like corners and sharp edges present in the original object in the reconstructed damage volume.

5.1 Future Work

The biggest computational bottleneck of our current implementation of DamageEst comes from the subdivision of the background mesh (Step 7 in Section 4.1). We use barycentric subdivision just for its simplicity and ease of implementation. Some possible speedups of the current implementation may come from "pushing" each vertex of the boundary to the closest point of the background mesh

rather than the closest vertex of the background mesh. If the subdivision of the background mesh occurs only at these closest points, i.e., in an adaptive fashion, there may be a speedup without significant increase in the background mesh complexity. The alternative is to implement more sophisticated Delaunay mesh refinement algorithms [6].

Path planning algorithms for 5D printers will also allow the damaged region to be filled with repair material. The path planning would require reasoning about collision with the damaged part and would be an important future research direction for automatic damage correction.

References

[1]

[2] M. J. Atallah, C. C. Ribeiro, and S Lifschitz. A linear time algorithm for the computation of some distance functions between convex polygons. *RAIRO - Operations Research - Recherche Opérationnelle*, 25(4):413–424, 1991.

[3] Marco Attene. A lightweight approach to repairing digitized polygon meshes. *The Visual Computer*, 26:1393–1406, 2010.

[4] Paul J. Besl and Neil D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992. doi:10.1109/34.121791.

[5] Li Chen, Rasmus Kyng, Yang P Maximum flow and minimum-cost flow In *63rd Annual IEEE Symposium on Foundations of Computer* $Science(FOCS), pages612 - -623, 2022.$

[6] Siu-Wing Cheng, Tamal K. Dey, and Jonathan R. Shewchuk. *Delaunay Mesh Generation*. CRC Press, 2012.

[7] Gianmarco Cherchi, Fabio Pellacini, Marco Attene, and Marco Livesu. Interactive and Robust Mesh Booleans. *ACM Transactions on Graphics*, 41(6), November 2022. doi:10.1145/3550454.3555460.

[8] Herbert Edelsbrunner and John L. Harer. *Computational Topology: An Introduction*. American Mathematical Society, December 2009.

[9] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996. URL: https://cdn.aaai.org/KDD/1996/KDD96-037.pdf.

[10] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. *ACM SIGGRAPH 1992 Proceedings*, 2:71–78, 1992.

[11] Kumar Kanishka and Bappa Acherjee. A systematic review of additive manufacturing-based remanufacturing techniques for component repair and restoration. *Journal of Manufacturing Processes*, 85:220–283, 2023. doi:10.1016/j.jmapro.2023.01.034.

[12] Lan Li, Xinchang Zhang, Tan Pan, and Frank Liou. Component repair using additive manufacturing: experiments and thermal modeling. *The International Journal of Advanced Manufacturing Technology*, 119:719–732, 2022. doi:10.1007/s00170-021-08265-y.

[13] Lingling Li, Congbo Li, Ying Tang, and Yanbin Du. An integrated approach of reverse engineering aided remanufacturing process for worn components. *Robotics and Computer-Integrated Manufacturing*, 48:39–50, 2017. doi:10.1016/j.rcim.2017.03.001.

[14] Matteo Perini, Paolo Bosetti, and Nicolae Balc. Additive manufacturing for repairing: from damage identification and modeling to DLD. *Rapid Prototyping Journal*, 26(5):929–940, 2020. doi:10.1108/RPJ-03-2019-0090.