

# SplatOverflow: Asynchronous Hardware Troubleshooting

AMRITANSH KWATRA, Cornell Tech, USA

TOBIAS WIENBERG, Cornell Tech, USA

ILAN MANDEL, Cornell Tech, USA

RITIK BATRA, Cornell Tech, USA

PETER HE, Cornell University, USA

FRANÇOIS GUIMBRETIÈRE, Cornell University, USA

THIJS ROUMEN, Cornell Tech, USA

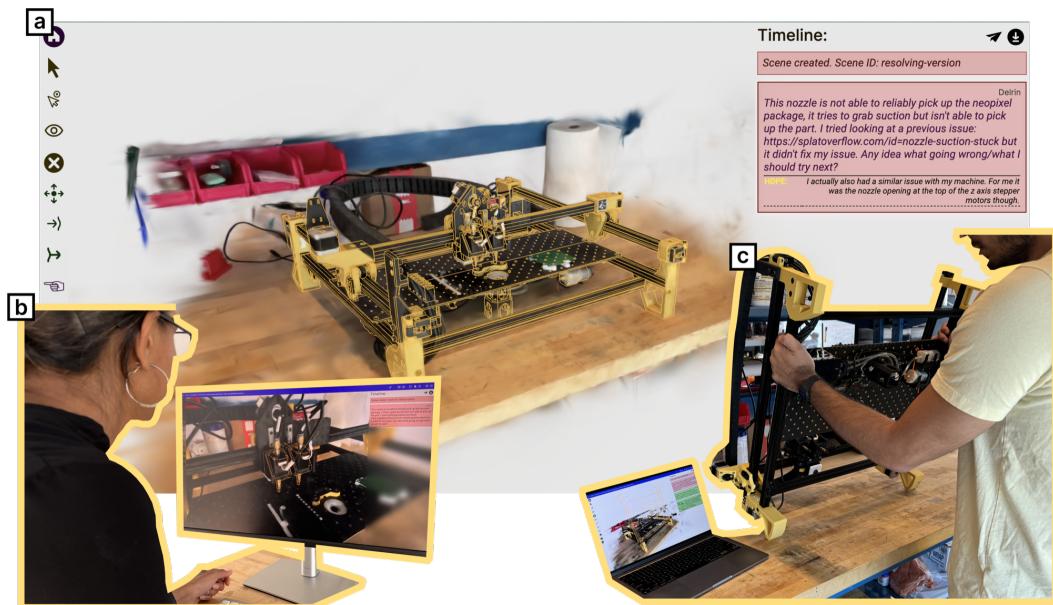


Fig. 1. (a) A SplatOverflow scene, comprising a 3D Gaussian Splat aligned and registered onto a digital CAD model, acting as a boundary object to facilitate asynchronous troubleshooting tasks for hardware. (b) A remote maintainer using SplatOverflow to explore a local user’s hardware and author instructions for them to follow. (c) A local user viewing the maintainer’s instructions rendered as an overlay onto their hardware in their workspace and performing the specified action.

As tools for designing and manufacturing hardware become more accessible, smaller producers can develop and distribute novel hardware. However, there aren’t established tools to support end-user hardware troubleshooting or routine maintenance. As a result, technical support for hardware remains ad-hoc and challenging to scale. Inspired by software troubleshooting workflows like StackOverflow, we propose a workflow for asynchronous hardware troubleshooting: SplatOverflow. SplatOverflow creates a novel boundary object, the SplatOverflow scene, that users reference to communicate about hardware. The scene comprises a 3D Gaussian Splat of the user’s hardware registered onto the hardware’s CAD model. The splat captures the current state of the hardware, and the registered CAD model acts as a referential anchor for troubleshooting

Authors’ addresses: Amritansh Kwattra, Cornell Tech, New York, USA, ak2244@cornell.edu; Tobias Wienberg, Cornell Tech, New York, USA, tmw88@cornell.edu; Ilan Mandel, Cornell Tech, New York, USA, im334@cornell.edu; Ritik Batra, Cornell Tech, New York, USA, rb887@cornell.edu; Peter He, Cornell University, New York, USA, ph475@cornell.edu; François Guimbretière, Cornell University, New York, USA, fvg3@cornell.edu; Thijs Roumen, Cornell Tech, New York, USA, tjr92@cornell.edu.

instructions. With SplatOverflow, maintainers can directly address issues and author instructions in the user’s workspace. The instructions define workflows that can easily be shared between users and recontextualized in new environments. In this paper, we describe the design of SplatOverflow, detail the workflows it enables, and illustrate its utility to different kinds of users. We also validate that non-experts can use SplatOverflow to troubleshoot common problems with a 3D printer in a user study.

## 1 INTRODUCTION

Hardware design and manufacturing tools have enabled small teams to develop and distribute novel machines and devices for niche applications. Examples of such hardware range from automation equipment for mid-scale manufacturing, such as the Lumen pick-and-place, to personal fabrication machines, such as the Prusa MK3S, to do-it-yourself gadgets like the Open Book e-reader. We refer to such products generally as *hardware*. For producers of this kind of hardware, turning a prototype into a viable product and supporting a growing user base with varying technical expertise remains challenging [31]. This paper focuses on one aspect of producing hardware: supporting end-user troubleshooting and maintenance.

Detailed and thorough documentation is an essential part of developing a hardware product[3, 10, 31]. However, documentation alone is insufficient to support the long tail of niche hardware issues users may encounter [54]. Instead, Subbaraman and Peek [66] argue that maintenance should be considered a core part of owning such hardware, and systems should be designed to support end-user troubleshooting and maintenance. We take inspiration from the infrastructure that supports software maintenance and troubleshooting workflows to examine how to create similar systems for hardware.

Platforms such as GitHub and StackOverflow have supported communities of software users by allowing them to help one another troubleshoot issues and by cataloging a history of past issues for anyone to reference. Crucially, these platforms rely on *asynchronous* communication between users. This allows users to seek out help or provide suggestions without coordinating availability or scheduling meetings with one another. Eliminating this barrier has allowed distributed communities of users to flourish and made asynchronous modes the norm in software development [2, 76]. This contrasts the workflows HCI researchers have proposed for remote expert guidance and hardware troubleshooting, which are primarily designed for synchronous modes of communication [26, 27, 52].

The utility of asynchronous workflows for troubleshooting is due, in part, to how they facilitate communication between users: via references to shared digital artifacts. These artifacts resemble boundary objects [2, 43, 65] that serve as mechanisms for communicating context and ideas online. In StackOverflow, for example, issues are accompanied by segments of code that the user is writing. Suggestions are then made as references to lines of code the user shared. The boundary object (in this case, lines of code) captures the user’s context and scaffolds the asynchronous communication with others. Importantly, this context is identical for those contributing to troubleshooting the issue and future users referencing the issue. As a result, a larger group of current and future users benefit from the knowledge gleaned in the interaction [76].

Hardware needs a robust boundary object to support the asynchronous modes of communication necessary to scale maintenance and troubleshooting infrastructure. We present SplatOverflow, a system that enables asynchronous hardware troubleshooting through a novel boundary object: a SplatOverflow scene. A SplatOverflow scene captures the as-built hardware through a scan (a 3D Gaussian Splat [39]) and aligns that scan to a shared CAD model of the hardware. The scan lets a remote user independently navigate a local user’s environment and inspect issues with the as-built hardware. The CAD model lets users interact with individual parts of the hardware and manipulate

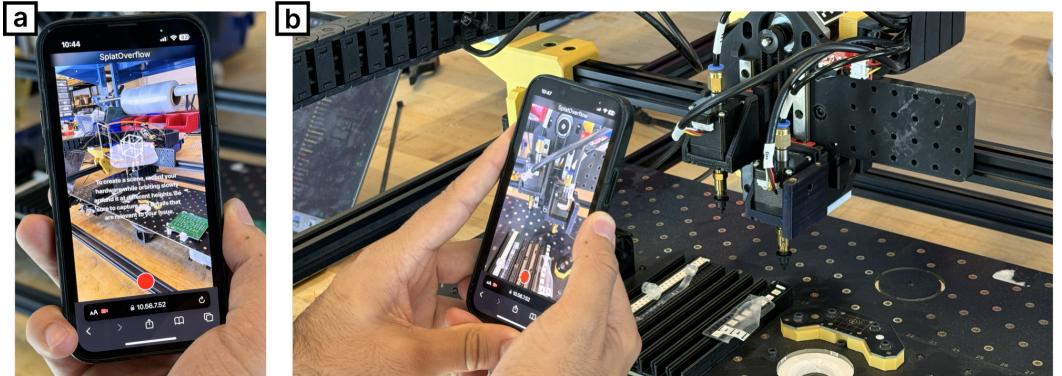


Fig. 2. (a) shows SplatOverflow’s mobile interface that captures a splat of the local user’s hardware. (b) shows a user scanning their hardware with the mobile web app. In our examples, an input video of roughly one minute from the web app produced a good splat, capturing details of the physical hardware in front of the user.

them to make actions. Interactions with the SplatOverflow scene are captured in a timeline that captures the history of instructions and discussions relating to a hardware issue.

The construction of a SplatOverflow scene presents distinct advantages to local users (who have the hardware in front of them), remote assistants, and future users who may encounter similar hardware issues. Local users benefit from access to technical documentation linked to the CAD model and the ability to describe issues on their hardware by pointing and clicking on parts rather than knowing hardware-specific vocabulary. Remote assistants benefit from seeing the local user’s issue registered onto the familiar CAD model and being able to freely move the view-port to inspect the hardware in the local user’s environment. Finally, future users benefit from being able to retrieve and replay solutions to past issues without seeking out synchronous support. Instead, SplatOverflow can re-contextualize instructions in a past issue by overlaying them onto the current user’s environment. This ability to accommodate multiple users and assistants is essential to how SplatOverflow can help support scaling the maintenance effort for hardware. In this paper, we present the design of SplatOverflow, demonstrate how it can be used to troubleshoot issues on different kinds of hardware, and validate that non-expert users can use SplatOverflow to troubleshoot hardware issues.

## 2 WALKTHROUGH

We demonstrate how a local user can troubleshoot a complex issue on the Lumen v3 Pick-and-Place machine using SplatOverflow. The Lumen is an automation machine that can assemble PCBs by picking up surface-mount electrical components with a vacuum nozzle and placing them on circuit boards. In this scenario, the local user’s machine can’t reliably pick up parts with one of the two vacuum nozzles. They use SplatOverflow to capture the issue they are facing and receive suggestions from a remote maintainer.

### 2.1 Creating a SplatOverflow Scene

They open SplatOverflow on their smartphone to capture the issue they are facing. Figure 2 shows SplatOverflow’s capture interface. The local user records a roughly one-minute-long video, filming the Lumen from multiple angles. After recording, they upload the video, and SplatOverflow generates a scene.

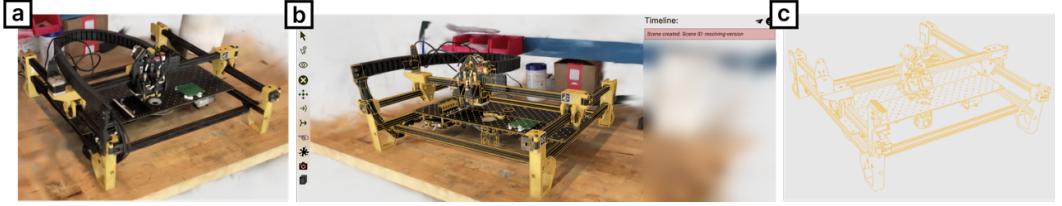


Fig. 3. (a) A 3D Gaussian Splat [39] of the local user’s Lumen v3. (b) A SplatOverflow Scene, comprising a 3D Gaussian Splat aligned onto a CAD model. (c) The CAD model is rendered as a yellow wireframe to preserve the rendering of physical details in the local user’s hardware present in the splat.

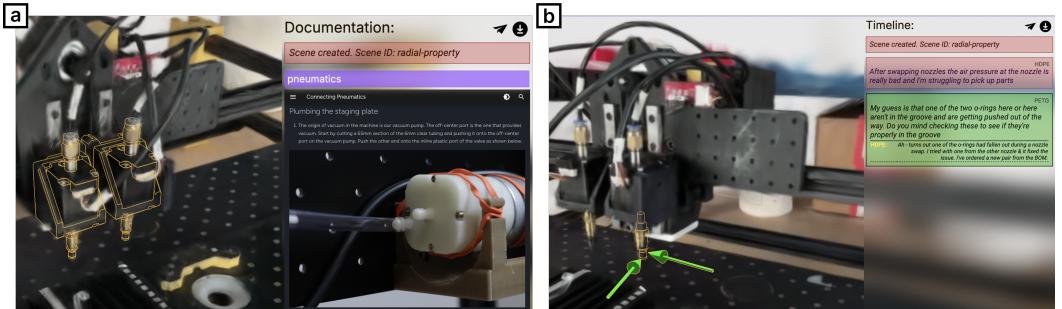


Fig. 4. (a) Using the CAD model to visually query and navigate technical documentation. When the user clicks on the nozzle, SplatOverflow retrieves sections of the assembly documentation referencing the nozzle. These sections are rendered in purple. The user can then click one of these sections and have its content rendered as an iframe within SplatOverflow. (b) The local user reprojects a previous issue referencing the nozzle’s poor suction. They view the past users’ suggestions to one another overlaid onto their own machine. In the previous issue, the maintainer indicated poor seating of two o-rings on the nozzle may be the issue. The local user tries this suggestion, but the o-rings are seated well, so they ask for more help.

After two minutes, SplatOverflow completes generating a *low resolution* scene. This scene is generated using down-sampled video, which yields a lower-quality splat. However, this lower-resolution scene allows the user to quickly begin using SplatOverflow. After roughly four minutes from the original upload, SplatOverflow finishes generating the *full-resolution* scene and presents the user with a rendering of their Lumen registered onto its CAD model. SplatOverflow removes the background regions of the splat by default to preserve the local user’s privacy. What’s left is a splat of the user’s hardware and parts of the immediate work surface on which it is placed. Figure 3 shows the scene as it is rendered in the browser.

## 2.2 Visually Querying Documentation and Past Issues

First, the local user examines any technical documentation for Lumen to find a solution. They select SplatOverflow’s documentation tool and click on the nozzle in the scene. Figure 4(a) shows how SplatOverflow then displays sections of the technical documentation and past SplatOverflow issues referencing the nozzle assembly in the CAD model. The user scrolls through the retrieved documentation but doesn’t find any suggestions about troubleshooting poor suction.

Next, the user sees a past issue that references a faulty nozzle. They open the issue, and SplatOverflow re-project the instructions from that scene onto their hardware. Figure 4(b) shows the solution to a prior issue re-projected into the local user’s SplatOverflow scene. The user walks

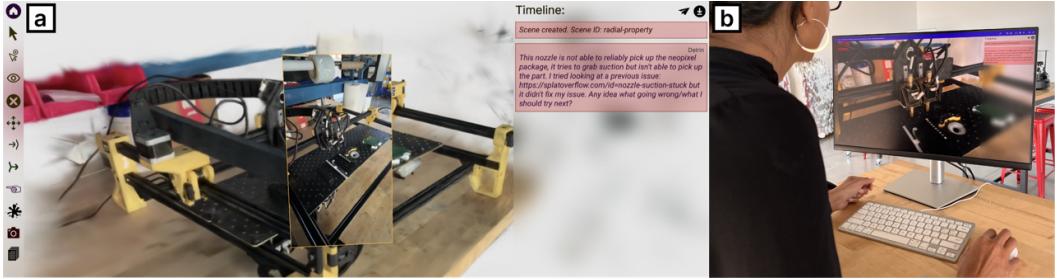


Fig. 5. (a) The local user’s video is placed as a floating screen in the SplatOverflow scene and aligned to the same perspective it was filmed in. (b) The remote maintainer reviews the local user’s issue by inspecting the vacuum nozzle in the splat and CAD model.

through the solution, which had to do with fixing the O-rings that seal the vacuum nozzle. They check that their O-rings aren’t dislodged and confirm this is not the source of their problem. The user then decides to capture more details about the issue they are experiencing and ask for help.

### 2.3 Asking for Help

To share their issue with a remote maintainer, the local user adds a description of the issue they are facing. They also use SplatOverflow’s video annotation gesture to add a video of the nozzle failing to pick up a component. SplatOverflow places the video into the scene to match the filming perspective, as shown in Figure 5a. The video describes the dynamic aspects of the issue that cannot be captured in the scan of the workspace. The local user posts the issue and waits for a remote maintainer to view it and offer suggestions.

### 2.4 Providing Guidance

After a few hours, the remote maintainer reviews the SplatOverflow issue and offers their feedback. Figure 5b shows the maintainer reviewing the issue by and inspecting the nozzle assembly in the scene. The maintainer cannot see any issues with the nozzle assembly and suspects that the source of the issue may be the pneumatics under Lumen’s staging plate. However, the underside of the staging plate is not visible in the machine’s current orientation.

The maintainer directs the local user to reorient the machine using SplatOverflow’s transform gesture and requests that they update the captured splat after moving it. The move instruction is visualized by animating the CAD model to move from the original aligned position to a new target position. The request to update the splat includes a QR code pointing the user to SplatOverflow’s mobile capture interface for adding a splat to an existing scene. Figure 6 shows the two instructions authored by the maintainer in the timeline and how each is rendered in SplatOverflow.

### 2.5 Understanding Suggestions

The local user reviews the suggestions made by the remote maintainer. They see that the maintainer has asked them to move their machine. Clicking the transform event in the SplatOverflow timeline shows the local user that they must stand the machine on the back legs. Figure 6 shows the user following the maintainer’s instructions by moving the machine and updating their splat. The user scans the QR code in the timeline event and records a video capturing the underside of the staging plate. Once the splat is updated, the scene contains two splats, with the machine in different orientations. Figure 7(a) shows the updated scene with the Lumen standing upright and

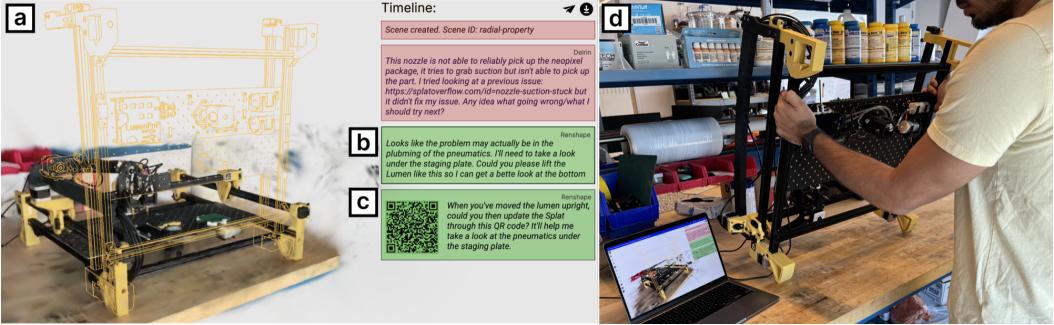


Fig. 6. A reply from the remote maintainer asking the user to re-orient the machine to a new position and then update the scene with footage of the underside of the machine. (a) shows the wireframe corresponding to the final position the machine should be in. (b) is the timeline element explaining why the movement needs to be made. When the local user clicks (b), the CAD model will animate to show how the machine should be moved. (c) shows how the request for a new splat is rendered for the local user. The QR code links them to the mobile capture interface, where they can update the scene the QR code links to. (d) shows the local user performing the action specified by the maintainer and preparing to update the splat.



Fig. 7. (a) shows the updated splat rendered in the browser. The machine is now standing upright as the maintainer requested, and the pneumatics in the staging plate are visible. (b) shows that the timeline is updated to indicate that a new splat has been added, replacing the QR code with a success message. (c) A close-up of the vacuum pumps dislodged with their positions in the CAD model overlaid. (d) A close-up of the pneumatic seal that the maintainer is concerned about.

the pneumatics easily visible for the remote maintainer to inspect. The splat captures the state of the wiring and routed tubes that are not included in the machine's CAD model.

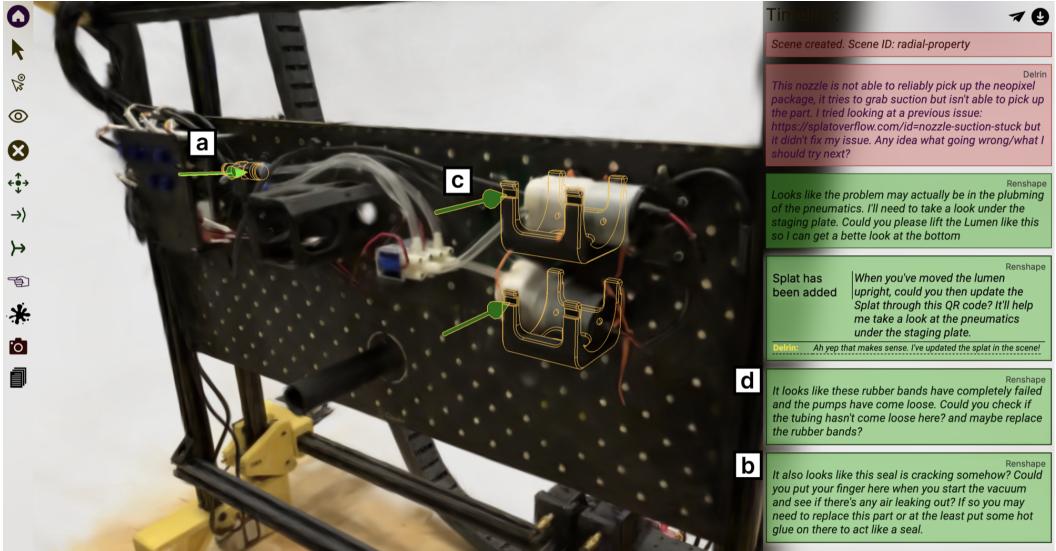


Fig. 8. The remote maintainer makes two sets of suggestions. First, they indicate that based on the splat, one of the fittings appears to be failing. They instruct the user on how to test the fitting and offer a stop-gap solution if it has failed. (a) the visualization indicates which part the remote maintainer references, and (b) shows the timeline element with the relevant instruction. Next, the remote maintainer notices that the pumps have come loose and asks the local user to check if the fitting is still secure. They also indicate that the local user should replace the deteriorated rubber bands. (c) shows the visualization overlaid onto the pump mounts that no longer hold anything, and (d) shows the timeline element with the relevant instruction.

## 2.6 Back and Forth Communication

The maintainer sees the updated splat and looks closely at the pneumatics under the staging plate. The remote maintainer inspects the underside of the pneumatics. They can use the splat to inspect the routing of tubes not modeled in the CAD and the state of components that may be degrading. After orbiting around the model, the maintainer feels confident the wiring is correct; however, the maintainer notices two things. First, the rubber bands holding the pumps have wholly degraded, as shown in 7(c). Second, the blue fitting on one of the pneumatic fittings has come loose and will likely not provide an adequate seal, as shown in 7(d). Both are potential sources of the issue, so the remote maintainer instructs the user to fix both. They use SplatOverflow's point-to gesture and attach a comment to the pneumatic fitting as well as the vacuum pumps. They instruct the local user to test if the tubes leaving the air pump are still securely attached and to replace the rubber bands when they have the chance. Next, the maintainer guides the user on testing if air is escaping the fitting when the pump is turned. The maintainer asks the user to place their finger on the seal and feel for air when the pump is on. In case the seal leaks air, the maintainer suggests that the local user apply some hot glue as a stop-gap solution while waiting for a replacement. Figure 8 shows these gestures rendered in the timeline and within the scene.

The local user reviews this suggestion and performs the test as described by the maintainer. When they turn on the pump, they feel air escaping the fitting. They apply some hot glue, and the vacuum pressure becomes more substantial. The local user orders a replacement fitting and replies to the suggestion indicating that a poor fitting was the culprit. Once the issue is resolved, the timeline is indexed to all the parts referenced in the back-and-forth exchange. This way, if a

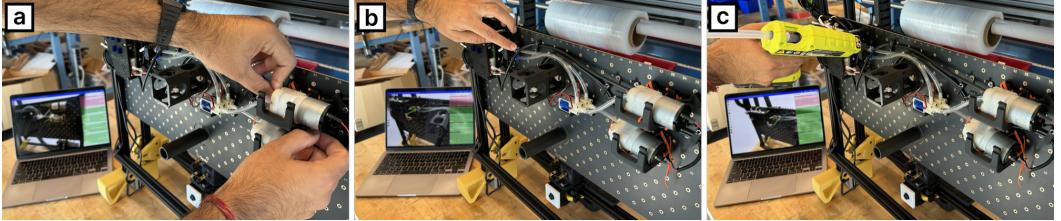


Fig. 9. The local user examines the instructions left by the remote maintainer. They start by (a) testing whether the tubing is still firmly attached to the vacuum pump. They feel that it is, so they move to the next suggestion. (b) shows them testing whether they can feel any air escaping the fitting that the maintainer referenced in their suggestion. They can feel air leaking out. (c) shows the local user trying the stop-gap solution of applying some hot glue to the fitting to improve the seal.

future user has a similar issue, the timeline for how to test this fitting will be available for them to reference.

### 3 UTILITY OF SPLATOVERFLOW

We contribute a technique to construct a boundary object comprising a 3D Gaussian Splat aligned to a CAD model and a timeline that can record interactions with the fused artifact [39]. We implement a system, SplatOverflow, that uses this technique to enable asynchronous troubleshooting workflows for hardware. This allows for clear communication between parties and enables troubleshooting and maintenance workflows to be shared between users. We argue that this workflow offers unique value to local users experiencing hardware issues and remote maintainers helping fix issues.

For the local user, the SplatOverflow scene introduces two kinds of interactions: retrieving digital information indexed to the CAD model and rendering instructions, physical actions, and references to hardware parts in situ. In SplatOverflow, users can click on the part they are interested in to retrieve information about it, such as assembly documentation. To do this, SplatOverflow uses its mapping from scan points to CAD parts to identify what the user has selected and search for references to that part in the available documentation. This search method can also surface past issues that other users have troubleshooted in SplatOverflow. This form of search allows users to query for information without knowing machine-specific vocabulary. For users who aren't familiar with the design of the hardware, this can simplify the process of navigating the corpus of documentation accompanying the hardware. SplatOverflow can also reduce the effort of mapping instructions across different contexts by rendering actions and references in the same environment as the user's hardware. Rather than translating a step from an online guide into action on their hardware, users can view the instruction as an overlay in their SplatOverflow scene. This can help clarify instructions that require physical actions to be carried out on the hardware. Moreover, since SplatOverflow anchors interactions to the CAD model of the hardware, instructions authored in one SplatOverflow issue can be reprojected onto any scan of the same hardware. In practice, this means users can replay instructions from past issues overlaid onto their own hardware.

SplatOverflow allows remote maintainers to independently navigate another user's hardware and rapidly author instructions. The registered 3D scan in SplatOverflow allows the remote maintainer to view the user's hardware from novel viewpoints. This lets the maintainer diagnose the *actual* state of the hardware, and the aligned CAD model allows them to quickly examine how the hardware may differ from the original design. Once a maintainer is ready to make suggestions, they can use SplatOverflow's gestures to author them directly in the scene. These gestures can communicate how the user should manipulate parts, where they should look, or what additional information the

remote maintainer needs. Each gesture is attached to the SplatOverflow timeline and rendered with an accompanying visual overlaid onto the scene. By enabling these interactions, SplatOverflow allows the remote maintainer to provide support without coordinating with the end user or other maintainers.

Moreover, while we have referred to the maintainer as a singular person, in practice, multiple people could carry out this role. A SplatOverflow scene can capture the history of the user's issue, suggestions made by multiple maintainers, and the back-and-forth about what worked and what did not. This ability to accommodate multiple users and maintainers is essential to how SplatOverflow can help support scaling the maintenance effort for hardware.

## 4 RELATED WORK

SplatOverflow builds on studies of hardware maintenance practices, hardware documentation support, telepresence for expert support, and spatial rendering techniques.

### 4.1 Hardware Maintenance Practice

While studying the practices of Xerox repair technicians, Julian Orr details how communication between expert technicians, end-users, and the machine plays an essential role in making sense of hardware issues [54]. Orr underscores how hardware maintenance cannot exhaustively be preempted in the hardware's documentation, instead describing the improvisational nature of diagnosing hardware issues. Rather than focusing on design and innovation when considering technological progress, Jackson [34] introduces "broken world thinking" and argues for foregrounding repair and maintenance practices. Similar ethnographic accounts of repair and "fixer" communities have examined commonalities and differences in these groups' norms, goals, and processes [32, 35]. More recently, Subbaraman and Peek [67] argue that maintenance is a core component of using digital fabrication workflows in 3D printing communities. Like Orr, they caution against addressing maintenance needs with fully automated procedures and instead advocate for systems to be designed with maintenance in mind.

### 4.2 Hardware Documentation Support

Documentation for assembly instructions, usage guides and maintenance tasks play an integral role in how users interact with and navigate hardware. Such documentation is distinct from the files that define the hardware, but researchers have argued that they contribute just as significantly to the hardware [10, 55]. In HCI, researchers have examined how such documentation may be made easier to produce. Mariscal-Melgar et al. [45] propose a semi-automated method for generating and updating assembly documentation from CAD software. Milara et al. [48] propose software tools for makers to create documentation as they are designing. Agrawala et al. [4] describe techniques for designing assembly instructions that are easy for end-users to understand. Similarly, researchers have explored how to make it easier for end-users to explore documentation. MagicNeRFLens[41] allows users to navigate a NeRF scene in virtual reality, with additional documentation overlaid onto the virtual scene. ARDW [15] is an augmented reality projection workbench that overlays documentation onto a PCB to support debugging.

### 4.3 Telepresence for Expert Support

In software development [5] and document editing [57], all collaborators typically access the same source code instead of compiled or high-level artifacts. Direct access to the collaborative artifact enables asynchronous support across the web [5]. HCI researchers examining synchronous support have proposed multiple theories and developed speculative systems for how remote experts can virtually occupy a shared task space with non-expert users to provide guidance [12, 24, 26, 52].

Telepresence solutions often try to establish a shared reference space, in which participants can communicate with one another with deictic expressions and gestures as they would if they were in person [23, 29, 36]. Such a reference space depends on both participants being able to communicate within a common context. In a survey on augmented reality systems for collocated experiences, Radu et al. [58] found that collaborators need a shared environment to ground their communication. Moreover, they find that collaborators need tools to guide attention and give instructions by referencing objects in the shared task space. Chastine et al. [14] develop a formal model of such inter-referential awareness, providing a detailed framework for how designers can develop referential systems.

Advances in augmented and virtual reality have led to work studying how experts can remotely collaborate on physical tasks [74]. Researchers have proposed systems for ‘mentoring’ tasks, where one less experienced user seeks out guidance from an expert user who views the scene remotely using mobile AR to bring the physical environment of users to remote collaboration [25, 26, 36]. Systems may have experts view the perspective of remote users in VR and communicate guidance through overlaid annotations [16, 71, 78].

Alternatively, some systems capture 3D reconstructions [33] and allow remote collaborators to interact by combining with teleoperated robots as in VRoxy [61] or by integrating live 360°video as Teo et al. [69] demonstrated.

In these systems, the interaction is synchronous, and the guidance provided is ephemeral, making it difficult for third-parties to replay and observe at a later time. *Heimdall* [37] is an alternative approach to remote collaboration for prototyping electrical circuits featuring a bespoke scanning system. Instructors can navigate around the prototype circuit and inspect its schematics through an instrumented breadboard. This enables instructors to remotely examine and debug student circuits without relying on a student to navigate around their breadboard.

#### 4.4 Spatial Rendering

Neural Radiance Fields (NeRFs) introduced a novel representation for 3D scenes that captured high-quality geometry estimations and view-dependent textures from collections of static images [49]. This work builds on earlier research in capturing real-world objects using images from multiple viewpoints [17, 46, 56, 75] and scenes from images captured in-the-wild [64]. Müller et al. [51] speed up training and rendering times for NeRFs and enable rapid capture and real-time navigation through the 3D scenes. More recently, researchers used collections of 3D Gaussian distributions to describe scenes in a process called 3D Gaussian Splatting [39]. A splat represents the scene statically and can be rendered in real-time using traditional graphics pipelines. These approaches build on COLMAP, a structure-from-motion (SfM) tool that constructs a point cloud and estimates camera pose from image frames [62, 63]. SplatOverflow is generally agnostic to scanning technology but leverages 3D Gaussian Splatting to capture the end-user’s workspace due to its fast training time and real-time rendering capabilities.

### 5 SPLATOVERFLOW

This section describes the design of SplatOverflow, shown in figure 10. We first discuss the constituent artifacts that comprise a SplatOverflow scene, including the benefits and shortcomings of each artifact. Then, we explain the construction of the SplatOverflow scenes, discuss how it facilitates asynchronous communication using SplatOverflow *gestures*, and finally describe how data from issues can be indexed and retrieved using a CAD model.

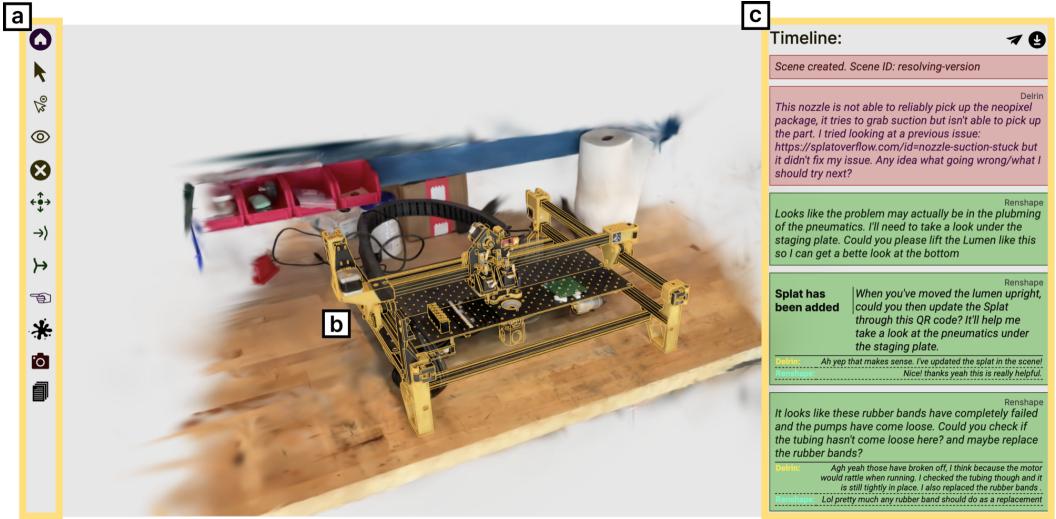


Fig. 10. The components of SplatOverflow’s interface. (a) is the palette of gestures SplatOverflow offers for selecting parts, guiding attention, and communicating actions. (b) the 3D Gaussian Splat of the hardware is aligned and registered onto the CAD model. (c) the timeline that captures the troubleshooting interaction as a sequence of instructions and responses between users.

## 5.1 Constituent Artifacts

SplatOverflow draws upon the benefits of a few existing artifacts that can be used to describe hardware. At a minimum, SplatOverflow relies on a scan of the hardware and a CAD model that describes the design of the hardware. Both artifacts have benefits and drawbacks as boundary objects for communicating the details and instructions to troubleshoot hardware. SplatOverflow combines these artifacts to construct a single boundary object that amplifies the benefits and minimizes the drawbacks of each constituent artifact.

**5.1.1 CAD Models.** CAD models can precisely and accurately describe the design of hardware. They also offer various viewing, selection, and manipulation tools to help make sense of hardware issues. For example, CAD models can let designers inspect internal mechanisms that aren’t visible in the assembled hardware. This ability to peer through parts can help designers reason about mechanical issues without taking apart the hardware. These models can also be the basis of generating visual assembly instructions that are easy for users to follow [4, 45].

The primary drawback of CAD models is that they do not reflect the host of ways hardware can err. This is because CAD models capture the hardware in an idealized state. In practice, however, hardware (especially malfunctioning hardware) does not perfectly resemble the CAD model. This is additionally problematic as some deviations from the CAD model are perfectly acceptable, while others can introduce errors. CAD models offer the tools and interactions to visualize, explore, and reason about a hardware design but do not capture the multitude of ways that each hardware instance is unique.

**5.1.2 Scans.** Scans, however, can capture the unique details of hardware instances. Dedicated mobile scanners using stereo vision, structured light, or novel-view synthesis techniques can accurately capture the geometry of hardware at different scales. This data can be used to measure

and inspect hardware as it has been built and assembled, which is essential when diagnosing what may be going *wrong* with the hardware.

The drawbacks of relying on scans are that they only capture visible geometry and do not know the semantic meaning of what they are capturing. Most scanning methods cannot safely penetrate materials to scan internal components. As a result, scans capture only an outer shell of the hardware geometry. Moreover, as scanning methods often do not know what they are scanning, it is challenging to segment out different parts. These two drawbacks are related in that they highlight how references are hard to anchor to scan data. An asynchronous workflow using scans must overcome this referential ambiguity.

## 5.2 A SplatOverflow Scene

SplatOverflow scenes comprise two essential artifacts: a scan of the user’s workspace aligned and registered onto a CAD model of the hardware. Our implementation generates the scans using 3D Gaussian Splatting [39]. We chose Gaussian Splatting as it can generate high-quality scans rapidly using only user-captured video data as input. We use open-source CAD models saved as .glb files with additional data to capture the assembly constraints in the model. The 3D Gaussian Splat (referred to here as *splat*) is aligned and registered to the CAD model using aruco markers that are placed on the hardware at known locations [50, 53]. The aligned artifacts support selecting and manipulating individual components on the hardware segmented using the CAD model and inspecting the physical state of the hardware as captured in the *splat*. SplatOverflow scenes are implemented in WebGL and run in the browser. As a result, they can be shared with anyone online, requiring no installation or OS-specific set-up. Scenes can be viewed on personal computers, mobile devices, and AR/VR headsets that support WebXR.

**5.2.1 Scanning a Workspace.** Scanning the hardware and its workspace captures the as-built state of the hardware. This includes modifications, job configurations, or parts not traditionally included in CAD, such as wires or cables. It can also capture the context in which the hardware is used. Scanning the physical hardware should ideally be fast and produce a high-fidelity rendering without requiring specialty hardware. To this end, SplatOverflow uses 3D Gaussian Splatting [39] to scan the hardware and workspace.

The input to the scan is a video of the user’s workspace. This video should adequately capture the hardware from multiple angles and include footage of any areas the user wants to troubleshoot. We implement a mobile phone interface for SplatOverflow to guide the user in creating a high-quality *splat*. Figure 2 shows this mobile interface in use.

The video is then sampled in two passes. First, at a fixed frame rate (4 FPS in our examples) and then again to more densely sample frames containing aruco tags. These two passes generate a set of images,  $I$ . SplatOverflow then feeds this set of images to COLMAP [62, 63] to generate an intermediate Structure-from-Motion model of the local users’ workspace. This model,  $M$ , includes a coarse point cloud representation of the workspace and camera pose estimates for each registered frame from set  $I$ . SplatOverflow uses model  $M$  to generate a *splat* of the recorded scene.

SplatOverflow takes roughly four minutes to generate a 3D Gaussian Splat from an input video filmed in 1080p resolution. This allows the local user to access SplatOverflow’s features quickly. SplatOverflow generates a lower-quality 3D Gaussian Splat from a downsampled version of the input video (at 720p resolution) to further accelerate processing. This *splat* is slightly worse at capturing details but still lets the local user query documentation and inspect past issues. The lower-quality *splat* is swapped out for a higher-quality one when processing is completed on the server. This low-resolution preview speeds up the feedback loop when creating a SplatOverflow scene.

**5.2.2 Localizing Video Feeds.** While 3D Gaussian Splats can capture the static details of a workspace with high resolution, they do not capture dynamic details in a scene. This is important for communicating the nature of many hardware issues, such as play or slop in an assembly. SplatOverflow allows users to capture these dynamic details as short video clips and aligns these clips in the SplatOverflow scene using an estimate of the camera pose at each frame. This allows remote maintainers to review the perspective from which a video was shot as well as the content of the video. Figure 5 shows the video feed in the SplatOverflow scene.

SplatOverflow estimates the pose of a video feed using COLMAP’s utility to localize frames within an existing model [62, 63], in our case  $M$ . Given an input video,  $V$ , SplatOverflow samples frames at a fixed frame rate (e.g., 2 FPS) and uses COLMAP to localize each frame within the SfM scene [62, 63]. The poses returned from COLMAP are used to place the video into the SplatOverflow scene.

**5.2.3 Registering a Gaussian Splat to a 3D CAD Model.** Our contribution is unlocked by creating a novel boundary object, a SplatOverflow scene, that can capture the physical state of the local user’s hardware and anchor instructions made by remote maintainers. At the core of a SplatOverflow scene is a 3D Gaussian Splat registered onto a CAD model of the hardware. Fusing these two artifacts allows references to be linked to common digital artifacts and render instructions in the local user’s environment.

SplatOverflow aligns and registers a splat of the local user’s hardware to the 3D CAD model in two steps using aruco tags placed in known locations on the hardware [50]. SplatOverflow first uses the SfM model  $M$  to compute a three-dimensional coordinate of every corner of each aruco tag [47]. These corners are used to re-scale the splat to be the same size as the CAD model. Next, SplatOverflow uses Arun’s method [6] to compute a rigid transformation going from corners in the SfM model’s coordinate space to CAD coordinate space. If there were no moving parts in the CAD assembly, this transformation yields an aligned and registered splat. However, hardware will often have moving parts. These degrees of freedom mean that the hardware can be in one of many states, which are unlikely to match the static state of the CAD model. To address this, SplatOverflow recommends at least one *constraint tag* be placed on each degree of freedom, and one *grounding tag* be placed on a rigid part of the hardware. When a CAD model is being prepared, the model’s maintainer indicates which tag ID maps to which degree of freedom. With this precondition, SplatOverflow uses the same detection method to find the centers of each *constraint tag* and computes an offset from the *grounding tag* to the *constraint tag*. This offset is then used to calculate a transformation that automatically aligns the CAD component to the position of the part in real life. Following these two steps, the hardware parts are aligned to their corresponding parts in the CAD model.

The tags used for alignment, registration, and constraint satisfaction can be applied manually onto existing hardware, placed during manufacturing, or designed into the hardware [20, 22]. In all of our examples, tags were manually placed onto existing hardware.

**5.2.4 Post Processing a Splat Using Features of the CAD Model.** Once the splat has been aligned and registered, SplatOverflow leverages information from the CAD model to post-process the splat. Firstly, SplatOverflow can separate captured points on the hardware from background points. SplatOverflow uses a signed distance function from points in the splat to CAD meshes. SplatOverflow not only allows users to toggle the subject and background on and off, but it enables the post-processing of a splat to preserve the local user’s privacy. By default, SplatOverflow removes the background details from a hardware scene and shares only parts of the splat that correspond to the hardware or the workspace the hardware rests on. Figure 11 shows a splat as it is generated and the same splat after SplatOverflow’s privacy filter. This has the benefit of protecting the privacy of local users while also cleaning up floating artifacts that can be generated when training a splat.



Fig. 11. (a) an unaltered 3D Gaussian Splat as returned by our system. (b) A pruned 3D Gaussian splat is used to preserve the user’s privacy and not share background details with a community of users.

### 5.3 Interacting with a SplatOverflow Scene

SplatOverflow offers tools to explore a scene, communicate actions, and facilitate discussion. This section describes each of these tools and the interactions they support.



Fig. 12. (a) a sub-assembly on the Lumen v3 with occluded components visible to the user. (b) a gesture indicating that the user tightens a bolt on the sub-assembly. (c) querying technical documentation referencing a specific sub-assembly.

**5.3.1 Selection and View Control.** When a part from the CAD model is selected, it is rendered to the user with a yellow wireframe. This allows the user to view details from the splat while examining the geometry of an individual CAD component. This allows the CAD model to act as a highlight on the splat, indicating the region of the hardware that corresponds to a distinct part.

SplatOverflow allows users to select parts and navigate subassemblies of the CAD model to make sense of occluded parts of the hardware not visible in the splat. Given that the CAD model is aligned to the splat, users can "peer through" sub-assemblies and view internal components that aren't visible. This is demonstrated in Figure 12(a).

Remote users can use SplatOverflow scenes to independently explore novel viewpoints within the local user's environment. This is enabled by the splat component of a SplatOverflow scene. From approximately one minute of video footage, SplatOverflow constructs a 3D Gaussian Splat [39] of the local user's scene. After the splat is aligned onto the CAD model, it occupies the same coordinate space and can be navigated with the same controls as a CAD environment. As a result, remote maintainers do not need to actively guide users to capture parts on their hardware. Instead, users can capture rough videos, and the Gaussian Splat can recover geometry and texture details.

**5.3.2 Gestures.** Users communicate in SplatOverflow via *gestures*. Gestures can be used to request information from the user and instruct actions to be applied to the hardware. Gestures are organized chronologically in an interactive timeline. The timeline elements trigger visualizations of the gesture in the SplatOverflow scene and house user discussions about the gesture.

Gestures can be used to request more information about the scene from the user. This can be done by requesting video annotations or updated splats using the *request* gesture. When a request

gesture is authored, a timeline element appears with a QR code. The QR code lets the user populate the request via SplatOverflow’s mobile app. When the request is populated, the timeline element is linked to the new media submitted by the user. This enables SplatOverflow issues to capture the changing state of a local user’s hardware throughout troubleshooting. An example of the request gesture is shown in Figures 6 and 7.

Gestures can also communicate instructions and actions a user should take on their hardware. Currently, SplatOverflow supports a few gestures meant to convey different actions. These *action* gestures are all anchored to parts in the CAD model, which are highlighted whenever a gesture is selected in the timeline alongside each gesture’s animation. Each gesture creates a timeline element that contains an additional text description explaining the gesture.

SplatOverflow’s *pointing* gestures indicate where on a part in CAD a user should inspect. These gestures are rendered with large green arrows pointing to or away from the CAD component. This lets them communicate different meanings depending on the context, such as tightening or loosening bolts, as shown in Figure 12(b). SplatOverflow’s *look-here* gesture prompts the user to inspect a specific CAD part from a prescribed perspective. When selected, the CAD part is highlighted, and the viewport shifts to re-orient the local user. SplatOverflow’s *move* gesture indicates to local users how parts should be manipulated. When selected, the referenced CAD part animates to tween between its original position and the final position defined by the remote maintainer. Finally, SplatOverflow supports a probe gesture for use in workflows involving PCBs. The probe gesture indicates where to place multimeter probes and the reading a user should expect to see.

**5.3.3 Timeline Elements and Discussions.** Timeline elements refer to *action* or *request* gestures authored by different users. The timeline element contains additional text to clarify what an action or request entails. Users can clarify or follow up on a gesture by replying to the corresponding element in the timeline. This facilitates the asynchronous back-and-forth to help make sense of a given gesture.

#### 5.4 Indexing Data to a CAD Model

SplatOverflow uses the CAD model to index and query data. This includes technical documentation referencing parts in the CAD model and past SplatOverflow issues. In the case of existing technical documentation, SplatOverflow renders web pages that reference CAD components as illustrated in Figure 12(c). Similarly, SplatOverflow can query the history of past issues to retrieve troubleshooting instructions that reference specific CAD components. This ability to feed structured data from troubleshooting issues into a database that can be queried allows SplatOverflow to become more valuable to hardware communities over time.

## 6 IMPLEMENTATION

As shown in Figure 13, SplatOverflow is divided into four sections: Scene Capture, Scene Generation, the SplatOverflow front-end, and the back-end Coordination Server.

The mobile capture interface runs in a browser and is written in JavaScript using the MediaStream Recording API. We typically use it from a smartphone browser. On the server, SplatOverflow uses a C++ 3D Gaussian Splatting implementation by MrNeRF [39] and the COLMAP package for Structure-from-Motion (SfM) to generate the scene. To align a CAD model to the scan, SplatOverflow uses the method from ArUco SfM Scale Adjustment [47] and an implementation of Arun’s method [6].

The back-end coordination server is a Flask application and SQL database that manages the data associated with a SplatOverflow scene. The SfM and Gaussian Splatting pipelines are run on a PC i7-14700K with 32GB RAM DDR5 and Nvidia 4080S GPU with 24GB of VRAM. On average, a

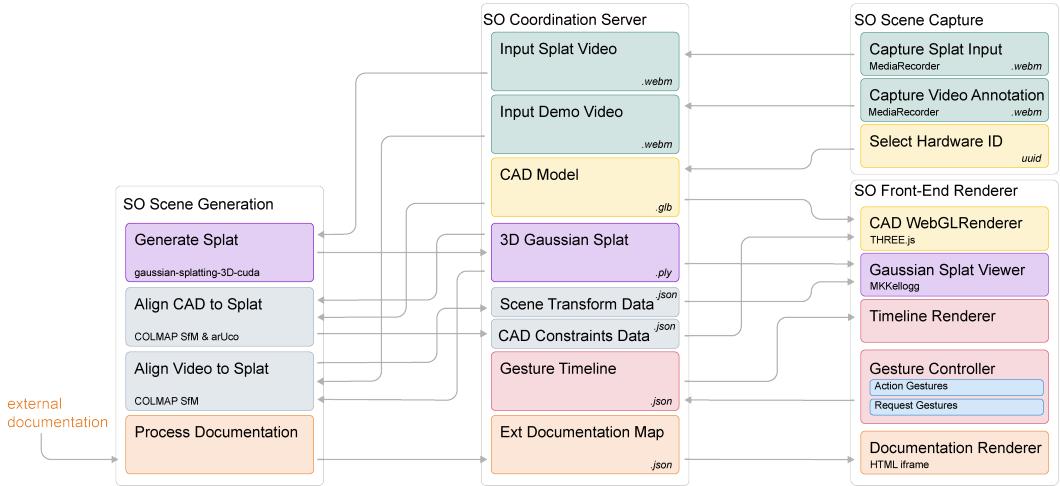


Fig. 13. A workflow illustrating our SplatOverflow implementation and the technologies it builds on.

60-second video takes 122 seconds to generate a scene trained on 360p footage and 250 seconds to generate a scene trained on 1080p footage.

The SplatOverflow front-end to interact with a scene is written in JavaScript using Three.js [19], and Mark Kellogg’s 3D Gaussian Splatting Renderer [38] is used to visualize the scene. The timeline and gesture visualizations are written in Javascript and leverage Three.js.

## 7 LIMITATIONS

This section details some limitations we have observed over the course of designing and implementing SplatOverflow.

### 7.1 Acquiring CAD Models

SplatOverflow requires access to the hardware’s CAD model to populate the scene. Moreover, for SplatOverflow scenes to be useful, the CAD models must be *complete*. CAD models like this can be challenging to come by outside of open-source hardware projects, which can limit the adoption of SplatOverflow. Moreover, even when CAD files are available, the structure of sub-assemblies, parts, and components is not standardized. As a result, preparing CAD files for use in SplatOverflow can require coercing the structure of the CAD assembly into a compatible shape.

### 7.2 Structure References in Technical Documentation

Another limitation of SplatOverflow lies in its ability to reference a corpus of existing technical documentation. With the Lumen v3, each page of technical documentation contained links referencing the related parts. This greatly simplified populating the documentation interface with instructions to search through. However, not all hardware will share technical documentation in the same structured fields as Opulo does with Lumen v3. As a result, existing technical documentation may have to undergo post-processing to add the relevant structured data indicating which CAD components are referenced.

### 7.3 Errors in Tag Placement

In all our examples, *grounding* and *constraint* tags were manually placed on the hardware. In doing so, we realized that while getting precisely aligned CAD models using manually placed tags is possible, the process is also prone to errors. Specifically, if a tag is misplaced, the amount SplatOverflow's alignment deviates increases proportionate to the distance from the tag. As a result, placing SplatOverflow tags manually is a challenging, error-prone task and may not be approachable for novice users.

## 8 DEMONSTRATIVE EXAMPLES

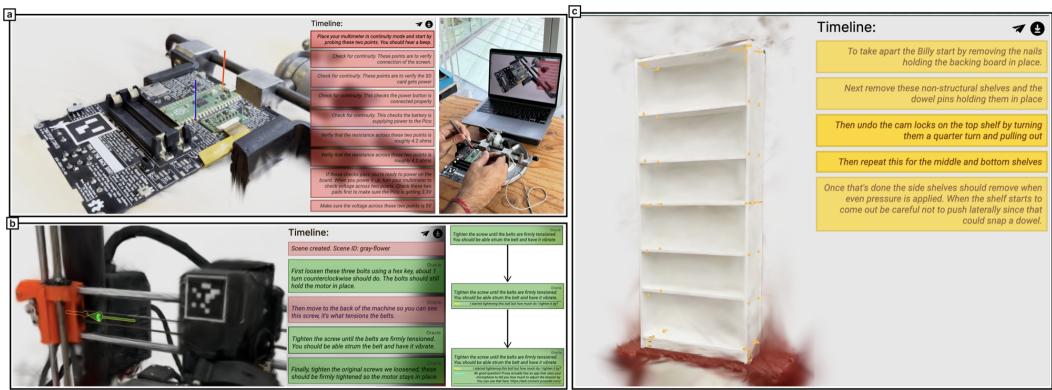


Fig. 14. Other workflows that extend the utility of SplatOverflow (a) non-mechanical CAD models (b) asynchronous workflows to create feedback loops on instruction from the community (c) enabling workflows on existing furniture with a simple sticker.

Figure 14 demonstrates the utility of SplatOverflow in three workflows that extend beyond troubleshooting of machines: (a) verifying assembly of the PCB in the *Open Book* e-reader by Oddly Specific Objects, (b) performing routine maintenance on the *Prusa MK3S* 3D printer, and (c) following disassembly instructions for a flat-pack bookshelf using the *Billy* bookshelf by IKEA. These workflows intend to capture a variety of scales, applications, and user roles that SplatOverflow can support.

### 8.1 Verifying the Correct Assembly of the *Open Book*

SplatOverflow is compatible with non-mechanical CAD designs. The *Open Book* [13] e-reader is an open source e-reader that consumers assemble on their own to demystify consumer electronics. Successful assembly is a function of correct and precise soldering. In this example, we show how the maintainer of the *Open Book* can help users verify correct assembly and track down potential errors that inhibit successful booting using a workflow authored and shared in SplatOverflow. Specifically, the maintainer defines a set of probe points to be checked with a multimeter that can indicate the exact probe points on the PCB the user must test, along with the expected outcome of each test.

The local user starts by creating a SplatOverflow scene to follow the assembly verification guide created by a maintainer. The *Open Book* has a 20mm tag that is used to align the CAD model of the PCB generated from KiCAD [68]. In this example, we applied the tag to the PCB, but in practice, such a tag could be printed directly onto the PCB's silkscreen. Once the SplatOverflow scene is generated, the local user loads the instructions authored by the maintainer into their scene and

sees the instructions projected onto their hardware. Figure 14(a) shows the instructions rendered onto their PCB.

The local user references verification instructions loaded into their SplatOverflow scene. As shown in Figure 14(a), instructions are overlaid onto the pins the user must probe. The user follows probing instructions to check for continuity, voltage and resistance at points across the PCB, as shown in Figure 14(b). At each step, the timeline event indicates the expected result of the probing for the user to compare against. The guide authored in a SplatOverflow scene in this application can be contextualized to each local user’s hardware. This contrasts most online guides, which are rendered using virtual objects or images and videos of a different hardware instance.

## 8.2 Sharing and Updating Routine Maintenance Workflows on the Prusa MK3S 3D Printer

Asynchronous sharing enables feedback loops, leading to improved workflows. The *Prusa MK3S* 3D Printer is a popular personal fabrication machine targeting hobbyists. Like all 3D printers and fabrication machines, the printer requires routine maintenance to ensure proper function. A typical maintenance task on the machine is tightening the belts that drive two axes of motion. Poorly tightened belts can lead to failed prints or unintended artifacts in otherwise successful prints. Learning to carry out such tasks is essential to owning and operating personal fabrication machines [66]. Here, we illustrate how SplatOverflow can be used to share maintenance workflows and how end-user feedback can improve the quality of such workflows.

A local user sees ghosting artifacts on their print and is pointed to a SplatOverflow issue explaining how to adjust the machine’s belt tension. Figure 14(b) shows the workflow rendered in the local user’s workspace. The user can follow the instructions but is unsure how much to tighten the belts. They leave a comment asking the maintainer how they know when to stop tightening the belt. The maintainer realizes that the instruction is underspecified and shares an online resource that uses a microphone to validate belt tension. Here, we show a simple example of how the open and asynchronous nature of SplatOverflow issues allows communities of users to iterate on workflows together.

## 8.3 Guided Disassembly of the *Billy* Bookshelf

SplatOverflow enhances static assemblies like furniture by adding a sticker. The *Billy* Bookshelf is a mass-produced piece of flat-pack furniture. Ikea supports DIY assembly through easy-to-understand assembly instructions. We demonstrate how SplatOverflow can guide users through step-by-step disassembly.

The *Billy* is roughly 80 inches tall, making it difficult to scan from all angles; particularly from the top down. The local user places an arUco marker on the back-right corner of the bookshelf and scans the *Billy*. As the bookshelf is upright during the scan, there is a noticeable degradation of quality above the bookshelf. However, since SplatOverflow aligns the CAD model in the scene, users can infer geometry while still receiving context cues from the scan.

The disassembly workflow is reasonably straightforward. The maintainer (in this case, IKEA) provided a SplatOverflow scene with the correct sequence of opening the cam locks that hold this bookshelf together. Users follow these instructions and sequentially disassemble the shelf. For more complex disassembly, it is possible to have the user update the splats to receive new instructions. This would resemble the scene update we described in the walkthrough. This example shows relevance beyond hobbyist hardware and how adding a sticker to ordinary furniture can enable new collaborative workflows for everyday objects.

## 9 EVALUATION

We conducted a usability study with twelve participants to validate whether end-users can use SplatOverflow to troubleshoot hardware issues. The hardware used in the study was a popular 3D printer: the Prusa MK3S.

### 9.1 Study Design

Our study is divided into two parts. First, we examine whether non-expert users can capture their hardware to create a SplatOverflow scene. We examine whether participants were able to generate a scene successfully. Second, we present them with two common issues experienced by users of this hardware. These issues were sourced based on the prevalence of online guides related to fixing them. We evaluate whether users could successfully follow the instructions in their SplatOverflow scene, and ask them to assess the usability of our system.

### 9.2 Part 1: Generating a SplatOverflow Scene

a				b	ID	Time (s)	Splat Generated	Splat Aligned	Issue #1	Issue #2
				p1	16.70		True	False	True	True
				p2	23.61		True	True	True	True
				p3	50.07		True	True	True	True
				p4	48.93		False	False	True	True
				p5	52.84		True	True	True	True
				p6	75.87		True	True	True	True
				p7	49.66		True	True	False	True
				p8	69.61		True	True	True	True
				p9	9.11		True	False	True	True
				p10	45.44		True	True	True	False
				p11	70.87		True	True	True	True
				p12	46.59		False	False	True	True

Fig. 15. We conducted a usability study with 12 participants to evaluate whether they were able to (a) generate a SplatOverflow scene and (b) successfully fix the issue.

Each participant was provided a smartphone running SplatOverflow's capture interface as a web app. The participants were asked to create a splat of the 3D printer in front of them by recording a video of the hardware from a variety of angles. All participants were able to capture a splat, and 83.3% of participants could successfully generate a SplatOverflow scene. For 33.3% participants, we were able to generate a SplatOverflow scene but did not have enough information in the user-captured video to solve for hardware constraints, i.e. align the degrees of freedom in the hardware. Figure 15 shows how many participants could generate a complete SplatOverflow scene with examples of high and low-quality scenes. We then showed participants the generated scene and explained how they could navigate our interface. Following this the participants filled out a survey evaluating the usability of the scanning interface.

We used their generated SplatOverflow scene for each participant in part 2 of our study. If the user's input failed to generate a scene (p1, p4, p9, p12), we assigned them to a placeholder scene that captured the same hardware.

### 9.3 Part 2: Following Instructions in SplatOverflow

In part 2, we presented the participant with two hardware issues in random order and asked them to follow instructions rendered in SplatOverflow to remedy the issues. The issues were: correcting

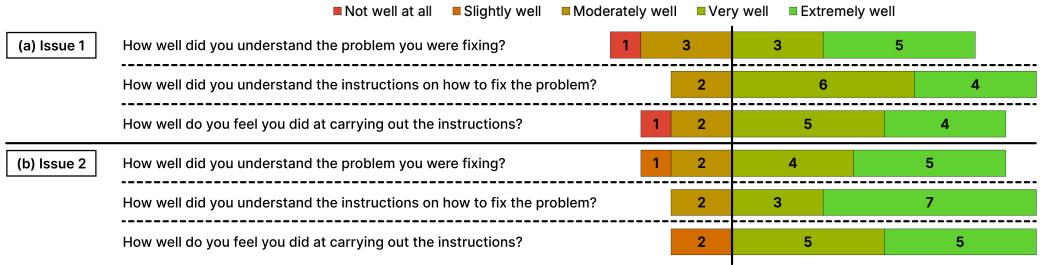


Fig. 16. Study participants responded to these questions after remedying a hardware issue with (a) Issue 1 (belt tensioning) and (b) Issue 2 (extruder jam).

the belt tension in the printer’s x-axis (issue 1) and clearing a jam in the extruder (issue 2). These issues were selected as they are commonly occurring issues with this printer and have a variety of online tutorials dedicated to them [1, 60]. We randomize the order of these issues, and asked the participants to follow the instructions to the best of their ability. 91.7% of participants were able to follow the instructions to fix the belt-tension issue and 91.7% of participants could fix the extruder jam issue. Following each issue we asked participants to answer some questions about their experience using SplatOverflow to troubleshoot the issue, as shown in Figure 16. Overall, users found that instructions rendered in SplatOverflow were easy to follow and well-aligned with the scanned hardware.

#### 9.4 System Usability

After both parts of the study we asked participants to evaluate the overall usability of the system in a survey. We use the System Usability Scale [11] and our system scored 86.25 / 100.

In addition participants reflected on how this system could be extended to help them in their lives. Participants shared that the benefits of SplatOverflow when facing issues with unfamiliar hardware (p1: *It is very helpful for me to use and repair, especially when I’m not familiar with the machine itself*) and they also compared our system to existing methods of troubleshooting (p3: *I was working with robots (Turtlebot 4 setup). There documentation was too wordy and it is hard to locate the solution from that. So, we have to post GitHub issues and wait for their response. It was sometime hard to explain them my specific problem. Of course, if there was a system that I could have scanned my problem and send them, it would have saved a lot of time*).

### 10 DISCUSSION

We discuss the implementation and application of SplatOverflow. We first highlight aspects of the experience for different users and then discuss technical features and future implementation.

#### 10.1 SplatOverflow Users

**10.1.1 Hardware Support for End-Users.** SplatOverflow offers a new way to collaborate on hardware. For workflows such as assembly, routine maintenance, operating instructions, and troubleshooting, SplatOverflow provides a novel way for users to receive guidance from collaborators or retrieve associated documentation. By connecting physical instances of hardware to their CAD model and associated documentation, end-users can inspect the ‘source’ of the hardware they are working with. This direct connection between the physical part and documentation is a more direct way to navigate documentation that can exist in disparate locations.

Moreover, users can use SplatOverflow to seek help, analogous to asking for help in a forum. In software, platforms such as GitHub [18] and StackOverflow [5] have been essential to help users learn about new software, troubleshoot issues and scale maintenance efforts needed to support projects. By supporting asynchronous communication about hardware, SplatOverflow presents a similar workflow for hardware.

**10.1.2 Hardware Support for Maintainers.** Through SplatOverflow, remote maintainers gain insight into the workspace and physical instance of the hardware a local user is working on. This was already possible with video conferencing and AR/VR workflows, but with SplatOverflow, maintainers can navigate a local user’s space independently. In SplatOverflow, maintainers do not rely on the local user to move a camera to inspect hardware. Maintainers can examine novel viewpoints independently and use gestures to guide the local user to move to the exact location.

This is a longstanding problem in collaborative workflows previously addressed with teleoperated robotic systems [37, 61]. SplatOverflow achieves a similar end without the need for bespoke hardware or additional instrumentation of a space. This, paired with SplatOverflow’s web-based interface, expands the number of users who can access remote support.

**10.1.3 Reducing Barriers to Distribute Hardware.** Tech support is a core component of a viable hardware product but requires significant effort for hardware companies to scale. Asynchronous support infrastructure could reduce this effort by (a) providing communities of users the means of querying past issues and examining solutions and (b) allowing maintainers to provide support in an open format without coordinating with end-users. It would thus support maintainers, producers, and hardware communities in scaling technical support for users. To better evaluate this, we plan to engage with Open-source hardware producers and community maker spaces to study longer-term deployments of SplatOverflow.

## 10.2 Technical Features

After implementing and testing SplatOverflow in a variety of contexts, we distilled a set of features to support it as we move towards a full deployment of the system.

**10.2.1 Guided Splat Capture.** In our evaluation, some user captures failed due to insufficient data to align the model. Given that proper alignment and registration are essential to a successful SplatOverflow scene, we plan to improve the feedback in SplatOverflow’s mobile interface. The capture process could be improved by specifying the hardware the user intends to scan. With this information, the interface can guide the user through capturing enough frames to appropriately align the splat to the CAD model and indicate which *grounding* and *constraint* tags require more footage.

**10.2.2 Automatically Preparing CAD Models.** To make a CAD model compatible with SplatOverflow, the producers of the hardware need to determine optimal locations to place the *grounding* and *constraint* tags. The choice of location for these tags can affect how difficult it is to align the captured splat to the CAD model. To support onboarding hardware into SplatOverflow, we plan to automate the tag placement process to optimize for visibility using a process similar to BrightMarker [21].

**10.2.3 IP of CAD models.** In some cases, CAD models are highly protected by hardware maintainers, who may not want to share these files with users. Currently, SplatOverflow uses a mesh representation of the CAD model, and only needs to know where the alignment tags are placed. For maintainers who are concerned about sharing mesh models, we plan to build an import pipeline that strips only the relevant data for our workflows without saving the actual CAD model on our server.

**10.2.4 Mixed Reality.** SplatOverflow is built in the browser and could be extended to fully immersive Augmented and Virtual Reality contexts by leveraging the WebXR Devices API [44]. While our implementation focuses on screen-based interaction, future work may examine how authoring SplatOverflow gestures could be extended to mixed reality.

### 10.3 Extending Multi-media workflows

Many tasks start with the assumption that the user is seeking guidance and has a screen available in front of them. Tools such as Interactive HTML BOM [30] allow users to visually correlate and search for components and their placements. The tool *assumes* users will be using a screen to aid in assembly and debugging.

Similarly, YouTube and other video hosting platforms [9, 40, 72] are used extensively for step-by-step instructions in cooking, assembly, and DIY Home repair. Documentation often has directly embedded video [28, p. 121] and manufacturers such as IKEA [73] have long produced videos to better demonstrate tasks. These videos can be valuable resources but are often expensive and challenging to create. As a result, video is not the ideal choice for end-users to articulate issues or seek guidance.

Our work extends multi-media hardware support by allowing users to rapidly author and share workflows for acting upon hardware amongst themselves. Communication can now more easily be made bidirectional rather than broadcast from experts or designers.

### 10.4 Cross-Referencing, Inspectability and Information Sharing

SplatOverflow enables the ability to link across hardware and data. This interaction mode is similar to *View Source* or Right Click to *Inspect Element* that is available in browsers [77]. These features were originally intended to help developers debug their own software [42, 59]. However, they rapidly became tools for education and exploration where “Every single web page you visited contained the code showing you how it was created. The entire internet became a library of how-to guides on programming” [70]. While we leave the specific application of SplatOverflow in education contexts to future work, we believe that the ability to inspect, discuss, and cross-reference physical and digital representations is vital to sharing hardware designs and know-how. SplatOverflow provides a set of primitives to support a known challenge for commons-based peer production [7, 8] of physical systems [66], primarily that of scalable instruction, support, and collaboration.

## 11 CONCLUSION

We have introduced SplatOverflow, a system for asynchronous hardware troubleshooting. SplatOverflow constructs a boundary object that can capture the physical details of a user’s hardware and communicate instructions as actions on the hardware instance. We demonstrate SplatOverflow through a series of workflows with different kinds of hardware. We have shown how these workflows can support complex troubleshooting workflows that require multiple exchanges between users and physical manipulation of the hardware. Moreover, we show how a SplatOverflow scene can be used in other application areas, such as querying data indexed to the CAD model and sharing maintenance and assembly workflows. Local users searching multiple disparate sources for relevant information can now access documentation and past solutions directly through the physical hardware. Maintainers relying on local users or telepresence robots to inspect remote spaces can now do so on their own, asynchronously.

We plan to distribute and deploy SplatOverflow with a broader user base through hardware maintainers and study our proposed forms of collaboration *in the wild*. Besides direct utility for hardware users and maintainers, we believe tools for communicating about physical hardware

can reduce the barrier to entry for smaller producers distributing niche forms of hardware and supporting community development amongst their users.

## ACKNOWLEDGMENTS

We would like to thank the Digital Life Initiative at Cornell Tech for supporting this work through a doctoral fellowship. We would also like to thank Joey Castillo, Frank Bu and Stephen Hawes for taking part in preliminary discussions that helped motivate this work.

## REFERENCES

- [1] Prusa 3D. 2023. Adjusting the belt tension on the Original Prusa MK4 - Belt Tuner App. [https://youtu.be/oeq2MVxE\\_H8?feature=shared&t=56](https://youtu.be/oeq2MVxE_H8?feature=shared&t=56)
- [2] Mark S. Ackerman, Juri Dachtera, Volkmar Pipek, and Volker Wulf. 2013. Sharing Knowledge and Expertise: The CSCW View of Knowledge Management. *Computer Supported Cooperative Work (CSCW)* 22, 4-6 (Aug. 2013), 531–573. <https://doi.org/10.1007/s10606-013-9192-8>
- [3] John R Ackermann. 2008. Toward Open Source Hardware. (2008).
- [4] Maneesh Agrawala, Doantam Phan, Julie Heiser, John Haymaker, Jeff Klingner, Pat Hanrahan, and Barbara Tversky. 2003. Designing Effective Step-by-Step Assembly Instructions. *ACM Trans. Graph.* 22, 3 (jul 2003), 828–837. <https://doi.org/10.1145/882262.882352>
- [5] Ashton Anderson, Daniel Huttenlocher, Jon Kleinberg, and Jure Leskovec. 2012. Discovering value from community activity on focused question answering sites: a case study of stack overflow. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, Beijing China, 850–858. <https://doi.org/10.1145/2339530.2339655>
- [6] K. S. Arun, T. S. Huang, and S. D. Blostein. 1987. Least-Squares Fitting of Two 3-D Point Sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-9*, 5 (Sept. 1987), 698–700. <https://doi.org/10.1109/TPAMI.1987.4767965>
- [7] Yochai Benkler. 2002. Coase's penguin, or, linux and" the nature of the firm". *Yale law journal* (2002), 369–446.
- [8] Yochai Benkler and Helen Nissenbaum. 2006. Commons-based peer production and virtue. *Journal of political philosophy* 14, 4 (2006).
- [9] Aditi Bhatia. 2018. Interdiscursive performance in digital professions: The case of YouTube tutorials. *Journal of Pragmatics* 124 (2018), 106–120.
- [10] Jérémie Bonvoisin, Robert Mies, Jean-François Boujut, and Rainer Stark. 2017. What is the “Source” of Open Source Hardware? *Journal of Open Hardware* 1, 1 (Sept. 2017), 5. <https://doi.org/10.5334/joh.7>
- [11] John Brooke. [n. d.]. SUS - A quick and dirty usability scale. ([n. d.]).
- [12] William Buxton. 1992. Telepresence: Integrating shared task and person spaces. In *Proceedings of graphics interface*, Vol. 92. Canadian Information Processing Society Toronto, Canada, 123–129.
- [13] Joey Castillo. [n. d.]. Open Book Project. <https://github.com/joeycastillo/The-Open-Book/tree/reboot#state-of-the-book>
- [14] Jeffrey W. Chastine, Ying Zhu, and Jon A. Preston. 2006. A Framework for Inter-referential Awareness in Collaborative Environments. In *2006 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 1–5. <https://doi.org/10.1109/COLCOM.2006.361859>
- [15] Ishan Chatterjee, Tadeusz Pforte, Aspen Tng, Farshid Salemi Parizi, Chaoran Chen, and Shwetak Patel. 2022. ARDW: An Augmented Reality Workbench for Printed Circuit Board Debugging. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. ACM, Bend OR USA, 1–16. <https://doi.org/10.1145/3526113.3545684>
- [16] Lei Chen, Yilin Liu, Yue Li, Lingyun Yu, BoYu Gao, Maurizio Caon, Yong Yue, and Hai-Ning Liang. 2021. Effect of Visual Cues on Pointing Tasks in Co-Located Augmented Reality Collaboration. In *Proceedings of the 2021 ACM Symposium on Spatial User Interaction (Virtual Event, USA) (SUI '21)*. Association for Computing Machinery, New York, NY, USA, Article 12, 12 pages. <https://doi.org/10.1145/3485279.3485297>
- [17] Shenchang Eric Chen and Lance Williams. 1993. View interpolation for image synthesis. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. ACM, Anaheim CA, 279–288. <https://doi.org/10.1145/166117.166153>
- [18] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*. 1277–1286.
- [19] Brian Danchilla. 2012. Three.js Framework. In *Beginning WebGL for HTML5*. Apress, Berkeley, CA, 173–203. [https://doi.org/10.1007/978-1-4302-3997-0\\_7](https://doi.org/10.1007/978-1-4302-3997-0_7)

- [20] Mustafa Doga Dogan, Vivian Hsinyueh Chan, Richard Qi, Grace Tang, Thijs Roumen, and Stefanie Mueller. 2023. StructCode: Leveraging Fabrication Artifacts to Store Data in Laser-Cut Objects. In *Proceedings of the 8th ACM Symposium on Computational Fabrication (SCF '23)*. Association for Computing Machinery, New York, NY, USA, 1–13. <https://doi.org/10.1145/3623263.3623353>
- [21] Mustafa Doga Dogan, Raul Garcia-Martin, Patrick William Haertel, Jamison John O'Keefe, Ahmad Taka, Akash Aurora, Raul Sanchez-Reillo, and Stefanie Mueller. 2023. BrightMarker: 3D Printed Fluorescent Markers for Object Tracking. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. ACM, San Francisco CA USA, 1–13. <https://doi.org/10.1145/3586183.3606758>
- [22] Mustafa Doga Dogan, Ahmad Taka, Michael Lu, Yunyi Zhu, Akshat Kumar, Aakar Gupta, and Stefanie Mueller. 2022. InfraredTags: Embedding Invisible AR Markers and Barcodes Using Low-Cost, Infrared-Based 3D Printing and Imaging Tools. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems (CHI '22)*. Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/3491102.3501951> event-place: New Orleans, LA, USA.
- [23] Paul Dourish and Victoria Bellotti. 1992. Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work - CSCW '92*. ACM Press, Toronto, Ontario, Canada, 107–114. <https://doi.org/10.1145/143457.143468>
- [24] Susan R. Fussell, Robert E. Kraut, and Jane Siegel. 2000. Coordination of Communication: Effects of Shared Visual Context on Collaborative Work. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work* (Philadelphia, Pennsylvania, USA) (CSCW '00). Association for Computing Machinery, New York, NY, USA, 21–30. <https://doi.org/10.1145/358916.358947>
- [25] Susan R. Fussell, Robert E. Kraut, and Jane Siegel. 2000. Coordination of communication: effects of shared visual context on collaborative work. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. ACM, Philadelphia Pennsylvania USA, 21–30. <https://doi.org/10.1145/358916.358947>
- [26] Steffen Gauglitz, Cha Lee, Matthew Turk, and Tobias Höllerer. 2012. Integrating the Physical Environment into Mobile Remote Collaboration. In *Proceedings of the 14th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '12)*. Association for Computing Machinery, New York, NY, USA, 241–250. <https://doi.org/10.1145/2371574.2371610> event-place: San Francisco, California, USA.
- [27] Steffen Gauglitz, Benjamin Nuernberger, Matthew Turk, and Tobias Höllerer. 2014. World-stabilized annotations and virtual scene navigation for remote collaboration. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM, Honolulu Hawaii USA, 449–459. <https://doi.org/10.1145/2642918.2647372>
- [28] Alicia Gibb. 2015. *Building open source hardware: DIY manufacturing for hackers and makers*. Pearson Education.
- [29] Carl Gutwin and Saul Greenberg. 2002. A Descriptive Framework of Workspace Awareness for Real-Time Groupware. *Computer Supported Cooperative Work (CSCW)* 11, 3-4 (Sept. 2002), 411–446. <https://doi.org/10.1023/A:1021271517844>
- [30] Hackaday. 2018. Interactive KiCAD BOMs Make Hand Assembly A Breeze. <https://hackaday.com/2018/09/04/interactive-kicad-boms-make-hand-assembly-a-breeze/>
- [31] Steve Hodges and Nicholas Chen. 2019. Long Tail Hardware: Turning Device Concepts Into Viable Low Volume Products. *IEEE Pervasive Computing* 18, 4 (Oct. 2019), 51–59. <https://doi.org/10.1109/MPRV.2019.2947966> Conference Name: IEEE Pervasive Computing.
- [32] Lara Houston, Steven J. Jackson, Daniela K. Rosner, Syed Ishtiaque Ahmed, Meg Young, and Laewoo Kang. 2016. Values in Repair. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, San Jose California USA, 1403–1414. <https://doi.org/10.1145/2858036.2858470>
- [33] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. 2011. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, Santa Barbara California USA, 559–568. <https://doi.org/10.1145/2047196.2047270>
- [34] Steven J. Jackson. 2014. Rethinking Repair. In *Media Technologies: Essays on Communication, Materiality, and Society*, Tarleton Gillespie, Pablo J. Boczkowski, and Kirsten A. Foot (Eds.). The MIT Press, 0. <https://doi.org/10.7551/mitpress/9780262525374.003.0011>
- [35] Steven J. Jackson and Laewoo Kang. 2014. Breakdown, obsolescence and reuse: HCI and the art of repair. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, Toronto Ontario Canada, 449–458. <https://doi.org/10.1145/2556288.2557332>
- [36] Janet G Johnson, Danilo Gasques, Tommy Sharkey, Evan Schmitz, and Nadir Weibel. 2021. Do You Really Need to Know Where “That” Is? Enhancing Support for Referencing in Collaborative Mixed Reality Environments. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) (CHI '21). Association for Computing Machinery, New York, NY, USA, Article 514, 14 pages. <https://doi.org/10.1145/3411764.3445246>
- [37] Mitchell Karchemsky, J.D. Zamfirescu-Pereira, Kuan-Ju Wu, François Guimbretière, and Bjoern Hartmann. 2019. Heimdall: A Remotely Controlled Inspection Workbench For Debugging Microcontroller Projects. In *Proceedings of*

- the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) (*CHI '19*). Association for Computing Machinery, New York, NY, USA, 1–12. <https://doi.org/10.1145/3290605.3300728>
- [38] Mark Kellogg. 2024. 3D Gaussian splatting for Three.js. <https://github.com/mkkellogg/GaussianSplats3D>
- [39] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (Aug. 2023), 1–14. <https://doi.org/10.1145/3592433>
- [40] Juho Kim, Phu Tran Nguyen, Sarah Weir, Philip J Guo, Robert C Miller, and Krzysztof Z Gajos. 2014. Crowdsourcing step-by-step information extraction to enhance existing how-to videos. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 4017–4026.
- [41] Ke Li, Susanne Schmidt, Tim Rolff, Reinhard Bacher, Wim Leemans, and Frank Steinicke. 2023. Magic NeRF Lens: Interactive Fusion of Neural Radiance Fields for Virtual Facility Inspection. <https://doi.org/10.48550/arXiv.2307.09860> [cs].
- [42] Chandan Luthra and Deepak Mittal. 2010. *Firebug 1.5: Editing, Debugging, and Monitoring Web Pages*. Packt Publishing.
- [43] Wayne G. Lutters and Mark S. Ackerman. 2007. Beyond Boundary Objects: Collaborative Reuse in Aircraft Technical Support. *Computer Supported Cooperative Work (CSCW)* 16, 3 (June 2007), 341–372. <https://doi.org/10.1007/s10606-006-9036-x>
- [44] Blair MacIntyre and Trevor F Smith. 2018. Thoughts on the Future of WebXR and the Immersive Web. In *2018 IEEE international symposium on mixed and augmented reality adjunct (ISMAR-Adjunct)*. IEEE, 338–342.
- [45] J. C. Mariscal-Melgar, Pieter Hijma, Manuel Moritz, and Tobias Redlich. 2023. Semi-Automatic Generation of Assembly Instructions for Open Source Hardware. *Journal of Open Hardware* 7, 1 (Aug. 2023), 6. <https://doi.org/10.5334/joh.56>
- [46] Leonard McMillan and Gary Bishop. 1995. An Image-Based Rendering System. *Los Angeles* (1995).
- [47] Lukas Meyer. 2023. Aruco Scale factor Estimation for COLMAP. <https://pypi.org/project/aruco-estimator/>
- [48] Iván Sánchez Milara, Georgi V. Georgiev, Jani Ylioja, Onnur Özüdürür, and Jukka Riekki. 2019. "Document-while-doing": a documentation tool for Fab Lab environments. *The Design Journal* 22, sup1 (April 2019), 2019–2030. <https://doi.org/10.1080/14606925.2019.1594926>
- [49] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. <http://arxiv.org/abs/2003.08934> arXiv:2003.08934 [cs].
- [50] Rafael Munoz-Salinas. 2012. Aruco: a minimal library for augmented reality applications based on opencv. *Universidad de Córdoba* 386 (2012).
- [51] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant NGP. *ACM Transactions on Graphics* 41, 4 (July 2022), 1–15. <https://doi.org/10.1145/3528223.3530127>
- [52] Ohan Oda, Carmine Elvezio, Mengu Sukan, Steven Feiner, and Barbara Tversky. 2015. Virtual Replicas for Remote Assistance in Virtual and Augmented Reality. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (Charlotte, NC, USA) (*UIST '15*). Association for Computing Machinery, New York, NY, USA, 405–415. <https://doi.org/10.1145/2807442.2807497>
- [53] Edwin Olson. 2011. AprilTag: A robust and flexible visual fiducial system. In *2011 IEEE International Conference on Robotics and Automation*. IEEE, Shanghai, China, 3400–3407. <https://doi.org/10.1109/ICRA.2011.5979561>
- [54] JULIAN E. ORR. 1996. *Talking about Machines: An Ethnography of a Modern Job*. Cornell University Press. <http://www.jstor.org/stable/10.7591/j.ctt1hhfnkz>
- [55] JULIAN E. ORR. 1996. *Talking about Machines: An Ethnography of a Modern Job*. Cornell University Press. <http://www.jstor.org/stable/10.7591/j.ctt1hhfnkz>
- [56] Eric Penner and Li Zhang. 2017. Soft 3D Reconstruction for View Synthesis. *ACM Transactions on Graphics* 36, 6 (Dec. 2017), 1–11. <https://doi.org/10.1145/3130800.3130855>
- [57] Ilona R Posner, Ronald M Baecker, and M Mantei. 1993. How people write together. In *Proceedings of the Hawaii International Conference on System Sciences*, Vol. 25. IEEE INSTITUTE OF ELECTRICAL AND ELECTRONICS, 127–127.
- [58] Iulian Radu, Tugce Joy, Yiran Bowman, Ian Bott, and Bertrand Schneider. 2021. A Survey of Needs and Features for Augmented Reality Collaborations in Collocated Spaces. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW1, Article 169 (apr 2021), 21 pages. <https://doi.org/10.1145/3449243>
- [59] Mike Ratcliffe. 2013. The History of Firebug. <https://flailingmonkey.com/the-history-of-firebug>
- [60] LA 3D Printer Repair. 2019. Prusa MK3S fixing stuck filament or bad unload. <https://www.youtube.com/watch?v=i5xnAQ5dHVs>
- [61] Mose Sakashita, Hyunju Kim, Brandon Woodard, Ruidong Zhang, and François Guimbretière. 2023. VRoxy: Wide-Area Collaboration From an Office Using a VR-Driven Robotic Proxy. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. ACM, San Francisco CA USA, 1–13. <https://doi.org/10.1145/3586183.3606743>
- [62] Johannes Lutz Schönberger and Jan-Michael Frahm. 2016. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

- [63] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. 2016. Pixelwise View Selection for Unstructured Multi-View Stereo. In *European Conference on Computer Vision (ECCV)*.
- [64] Noah Snavely, Steven M. Seitz, and Richard Szeliski. 2006. Photo tourism: exploring photo collections in 3D. In *ACM SIGGRAPH 2006 Papers on - SIGGRAPH '06*. ACM Press, Boston, Massachusetts, 835. <https://doi.org/10.1145/1179352.1141964>
- [65] Susan Leigh Star and James R. Griesemer. [n. d.]. Institutional Ecology, ‘Translations’ and Boundary Objects: Amateurs and Professionals in Berkeley’s Museum of Vertebrate Zoology, 1907–39. <https://doi.org/10.1177/030631289019003001>
- [66] Blair Subbaraman and Nadya Peek. 2023. 3D Printers Don’t Fix Themselves: How Maintenance is Part of Digital Fabrication. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference*. ACM, Pittsburgh PA USA, 2050–2065. <https://doi.org/10.1145/3563657.3595991>
- [67] Blair Subbaraman and Nadya Peek. 2023. 3D Printers Don’t Fix Themselves: How Maintenance is Part of Digital Fabrication. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference* (Pittsburgh, PA, USA) (*DIS ’23*). Association for Computing Machinery, New York, NY, USA, 2050–2065. <https://doi.org/10.1145/3563657.3595991>
- [68] KiCad Development Team. [n. d.]. KiCad EDA - Schematic Capture & PCB Design Software. <https://www.kicad.org/>
- [69] Theophilus Teo, Louise Lawrence, Gun A. Lee, Mark Billinghurst, and Matt Adcock. 2019. Mixed Reality Remote Collaboration Combining 360 Video and 3D Reconstruction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) (*CHI ’19*). Association for Computing Machinery, New York, NY, USA, 1–14. <https://doi.org/10.1145/3290605.3300431>
- [70] Clive Thompson. 2020. *Coders: The making of a new tribe and the remaking of the world*. Penguin.
- [71] Balasaranavan Thoravi Kumaravel, Cuong Nguyen, Stephen DiVerdi, and Bjoern Hartmann. 2020. TransceivVR: Bridging Asymmetrical Communication Between VR Users and External Collaborators. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) (*UIST ’20*). Association for Computing Machinery, New York, NY, USA, 182–195. <https://doi.org/10.1145/3379337.3415827>
- [72] Sonja Utz and Lara N Wolfers. 2022. How-to videos on YouTube: The role of the instructor. *Information, Communication & Society* 25, 7 (2022), 959–974.
- [73] Keith Wagstaff. 2012. IKEA Starts ‘How to Build’ YouTube Channel to Help Frustrated Customers | TIME.com. <https://techland.time.com/2012/02/24/ikea-starts-how-to-build-youtube-channel-to-help-frustrated-customers/>
- [74] Peng Wang, Xiaoliang Bai, Mark Billinghurst, Shusheng Zhang, Xiangyu Zhang, Shuxia Wang, Weiping He, Yuxiang Yan, and Hongyu Ji. 2021. AR/MR remote collaboration on physical tasks: A review. *Robotics and Computer-Integrated Manufacturing* 72 (2021), 102071.
- [75] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. 2020. SynSin: End-to-End View Synthesis From a Single Image. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Seattle, WA, USA, 7465–7475. <https://doi.org/10.1109/CVPR42600.2020.00749>
- [76] Yutaka Yamauchi, Makoto Yokozawa, Takeshi Shinohara, and Toru Ishida. 2000. Collaboration with Lean Media: how open-source software succeeds. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. ACM, Philadelphia Pennsylvania USA, 329–338. <https://doi.org/10.1145/358916.359004>
- [77] Mykyta Yevstifeyev. 2011. *The ‘view-source’ URI Scheme*. Internet-Draft draft-yevstifeyev-view-source-uri-01. Internet Engineering Task Force. <https://datatracker.ietf.org/doc/draft-yevstifeyev-view-source-uri/01/> Work in Progress.
- [78] Kevin Yu, Ulrich Eck, Frieder Pankratz, Marc Lazarovici, Dirk Wilhelm, and Nassir Navab. 2022. Duplicated reality for co-located augmented reality collaboration. *IEEE Transactions on Visualization and Computer Graphics* 28, 5 (2022), 2190–2200.