INTERNATIONAL DESIGN CONFERENCE – DESIGN 2024
https://doi.org/10.1017/pds.2024.61

Learnings from developing a custom virtual assembly environment for mountability issues of cooling cabinets

Georg Hackenberg ✉ and Christian Zehetner

School of Engineering, University of Applied Sciences Upper Austria, Austria

✉ georg.hackenberg@fh-wels.at

Abstract

Various industries use computer simulation for verifying product properties in early phases of development. Traditionally, such properties include the stability of mechanical structures or the efficiency of aircraft turbines. More recently, research also focuses on the mountability of industrial products using virtual assembly. While research on virtual assembly already started in the mid-1990s, the applicability in different industries remains largely unclear today. To advance the state-of-the-art, in this paper we present learnings from developing a virtual assembly environment for cooling cabinets.

Keywords: case study, simulation-based design, virtual reality (VR)

1. Introduction

Since many years, companies from various industries such as automotive and aerospace (D'Ambrosio und Soremekun 2017) try improving the efficiency of their product development processes to increase their profit margins and stay competitive on the global market. A major lever for improving efficiency of development processes is to avoid poor management and engineering decisions that cause undesirable product properties and, hence, entail unnecessary follow-up costs. It is known for long time that the earlier such mistakes can be detected and corrected, the less follow-up costs they will entail and, consequently, the more efficient development processes will perform. Ideally, the corrective measures are taking place in the ideation, specification, and conception phases of the product life cycle (see Figure 1), while the realization, operation, and elimination phases can rely on stable grounds.

Figure 1. Generic model of the product life cycle

Different kinds of undesirable product properties can emerge due to poor management and engineering decisions along the product life cycle: For example, the mechanical stability of the product might be compromised, or the thermal isolation of the product might be unsatisfactory. Such undesirable properties still emerge regularly today from development processes in different industries because predicting the impact of management and engineering decisions on critical product properties (see Figure 2) remains a major challenge. Traditionally, managers and engineers predict such impacts with pen and paper models and calculations, which work fine for relatively simple relations between the decisions (e.g., the size of support beams in building construction) and the emerging product properties (e.g., the mechanical stability of the building). Later, many industries switched to computer models and calculations (Maciej A Bossak 1998) for handling more complex relations between the decisions (e.g.,

the geometry of turbines in aerospace engineering) and the constrained product properties (e.g., the energetic efficiency of the turbine under development).

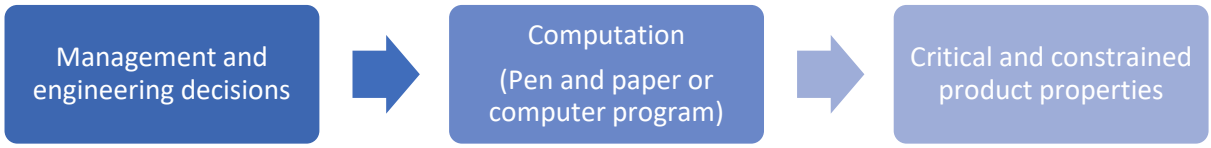

Figure 2. Generic model of the computation-based verification procedure

In this paper we focus on the mountability of physical products, which can be hindered due to limitation of the working space for example. As we know from our industrial partners, in practice the mountability of products cannot always be verified from manual reviews of CAD drawing only. While our partners typically have such review procedures in place, it happens from time to time that a product design passes the review gate and goes into the physical prototyping stage. The prototyping stage generally consumes substantial resources, such as material and labour, as well as time for logistics, manufacturing, and assembly. If mountability issues are detected, the product design must be revised and might run through the same verification procedure again. Consequently, unnecessary costs for product development accumulate, and the time to market of the product is delayed. In addition to manual reviews of CAD drawings, one can use virtual assembly (Florian Dyck et al. 2023) to simulate the assembly process and detect potential mountability issues. However, virtual assembly doesn't come at zero costs. Among others, the costs largely depend on how much software functionality can be purchased of the shelve today, and how much custom software functionality must be developed to cover a specific use case. To understand these costs better, in this paper we present the results from an industrial case study.

1.1. Research context

In this work we focus on a specific class of industrial products, namely cooling cabinets, which can be found, for example, in supermarkets. Typical issues, which our industrial partners have faced recurringly in the past with new product designs during physical prototyping, include unreachable target positions with specific assembly tools due to limitations of working spaces, or unergonomic working poses of human workers during assembly. Note that the most common assembly tools for this use case are screw drivers and silicon guns, but in principle the space problem applies to other assembly tools as well. Furthermore, due to the size of one physical part the assembly process under consideration includes a collaborative step, during which two workers need to carry that part together and threaten one end of the part through a narrow hole. Finally, the assembly process includes the bending of copper tubes as one of the final steps and, hence, the simulation of dynamic part geometries.

1.2. Research question

To understand the costs of developing a custom virtual assembly environment, we drive the following research questions: **(RQ1)** Which free and open source software components can serve as a foundation for a custom virtual assembly environment? **(RQ2)** Which basic features must be implemented into the environment and how much effort does the implementation require? **(RQ3)** How can specific assembly steps such as siliconizing, collaborative part carrying, and tube bending be represented in the virtual assembly environment and how much does effort does their implementation require?

1.3. Research methodology

To answer the previous questions, we first reviewed the state-of-the-art in virtual assembly (see Section 2), before developing a custom virtual assembly environment (see Section 3), and extracting our findings (see Section 4). Among the available product designs a case was selected, in which our industrial partners in the past overlooked mountability issues during manual reviews of the CAD models but detected these issues later during physical prototyping. For development of the custom virtual assembly environment, we decided to carry out an agile software development project with one product owner and one software developer from our research institute, as well as three stakeholders (one project manager, one quality manager, and one mechanical engineer) from the industrial partners. The research

https://doi.org/10.1017/pds.2024.61 Published online by Cambridge University Press

institute committed resources of one full-time equivalent (FTE) over a duration of six months to this project, while the industrial partners provided the necessary CAD models, compiled explanations of the current assembly procedures, and reviewed the project results after achieving certain milestones. For managing the source code, the development tasks, and the sprint plans, we used the free and open-source platform GitLab[1]. During the project, in total 105 tasks have been reported and 97 tasks have been solved across 16 sprints, while eight tasks remained open at the time this article was written.

2. Related work

Work on virtual assembly started in the mid-1990s, when the VR technology became mature enough to implement and test this type of application. The Virtual Assembly Design Environment (VADE) was one of the first solutions presented to the scientific community focusing on assembly process design and evaluation including part and tool trajectories and collision detection (Sankar Jayaram et al. 1999). Later, the attention shifted to implementing physical behaviour and constrained motion of objects in the virtual environment, which was found not to improve the evaluation of assembly processes (Wang et al. 2001). Then, first works were published quantifying the achievable economic benefits of virtual assembly, which can occur, e.g., in combination with product design for assembly due to early feedback on the performance of assembly processes and subsequent product design improvements (Choi et al. 2002) Thereafter, researchers worked on improving the usability of virtual assembly environments including data gloves and force feedback as well as acoustic feedback, which can help improving the overall effectiveness of virtual assembly (Wan et al. 2004). Subsequently, the use of lower resolution models for virtual assembly as well as the connection of the lower resolution models to the original CAD models was investigated, to reduce the overall complexity and improve the performance of virtual assembly environments (Q.-H. Wang und H.-Q. Gong 2006). Then, the readiness of virtual assembly for industrial application was assessed across several case studies showing the maturity of the technology, but also highlighting the need for further research with different industrial partners (Jayaram et al. 2007). Later, the first works on collaborative virtual assembly environments were published focusing on the system architecture rather than concrete applications (Ma, Dengzhe and Zhen, Xijin and Hu, Yong and Wu, Dianliang and Fan, Xiumin and Zhu, Hongmin 2011) Thereafter, a first review of virtual assembly technologies was conducted identifying important gaps for future work were including haptic feedback as well as more collaborative virtual assembly environments (Ming C. Leu et al. 2013). Consequently, the research on haptic feedback was intensified explaining technological requirements and showing improved usability (Mustufa H. Abidi et al. 2015). Around the same time, other researchers studied the integration of virtual and physical components during assembly simulation, which adds additional realism to virtual assembly simulation but requires augmented reality equipment (X. Wang et al. 2016). Afterwards, the positive effects of virtual assembly with haptic feedback on assembly training was studied showing reduced assembly times of pre-trained workers (Abidi et al. 2019). Later, virtual assembly was tested successfully in the aeronautics industry for evaluating the ergonomics for human workers as well as the performance of semi-automated solutions including different numbers of robots (Kiara Ottogalli, Daniel Rosquete, Javier Rojo, Aiert Amundarain, José María Rodríguez und Diego Borro 2021). Finally, a recent summary of virtual assembly was published highlighting remaining challenges such as the improvement of data integration with CAD applications and product lifecycle management (PLM) solutions, the design and implementation of advanced and task-specific interaction techniques, and the improvement of simulation accuracy (Florian Dyck et al. 2023).

While most research focused on the feasibility, effectiveness, usability, and benefits of virtual assembly, little research has focused on the costs of implementing virtual assembly environments including custom software development efforts. In particular, it remains unclear what role free and open source software can play today in the implementation of such environments. Furthermore, it remains unclear how much effort the implementation of advanced and task-specific interaction techniques requires (e.g. for representing assembly steps such as siliconizing, collaborative part carrying, and tube bending).

[1] https://gitlab.com

https://doi.org/10.1017/pds.2024.61 Published online by Cambridge University Press

3. Virtual assembly environment development

In this section, we describe the development of a virtual assembly environment. In Section 3.1 we introduce a software technology stack, which mostly includes free and open source software components to minimize license costs. Then in Section 3.2, we explain basic features, which need to be included into any virtual assembly environment independent of the product (e.g., cooling cabinets). Finally in Section 3.3, we describe the implementation of specific assembly steps such as siliconizing, collaborative part carrying, and tube bending.

3.1. Software technologies

Before fixing the software technology stack (see Figure 3), we looked at different options that exist on the market today. Ultimately, you need an appropriate VR development environment and application framework (VR platform), which can handle the CAD models, and which provides a solid basis for implementing the necessary features. Note that in this study we used the standard for the exchange of product model data (STEP) for exchanging the CAD models due to its wide support across CAD application vendors and industry adoption.

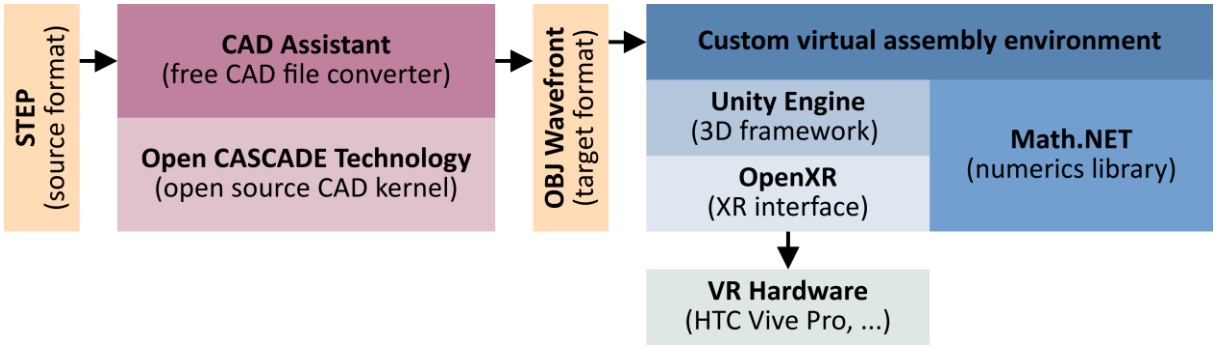

Figure 3. Overview of the software technologies for our custom virtual assembly environment

Two major VR development environments and application frameworks (or VR platforms) dominate the market today, namely the Unreal Engine[2] (written in C++) and the Unity Engine[3] (written in C#). Note that both options provide vendor- and device-specific integrations with VR devices (i.e., headsets, controllers, and trackers) of different vendors (e.g., Meta or HTC). These integrations provide access to the latest device features such as eye and hand tracking. Alternatively, both platforms support the OpenXR standard[4] for interfacing with a broad range of VR devices from different vendors, but which supports only the most common features across all devices and vendors. Besides the classical VR platforms, today also WebXR-compatible[5] browsers with JavaScript or WebAssembly can serve as the basis for implementation. However, the performance of JavaScript-based frameworks (e.g., Three.js[6]) does not suffice currently, and WebAssembly-based frameworks (e.g., the Wonderland Engine[7]) are not mature enough yet. Therefore, we limited our search to the two major VR platforms, i.e., the Unreal Engine and the Unity Engine.

In general, VR platforms do not support STEP files directly but support other data formats from the gaming industry such as OBJ Wavefront. Consequently, the CAD models must be converted from the STEP format into those supported data formats. With Open CASCADE Technology[8] (written in C++) a free and open-source library exists, which can read STEP data and convert it into various other data

[2] https://www.unrealengine.com/
[3] https://unity.com/
[4] https://www.khronos.org/openxr/
[5] https://www.w3.org/TR/webxr/
[6] https://threejs.org/
[7] https://wonderlandengine.com/
[8] https://dev.opencascade.org/

https://doi.org/10.1017/pds.2024.61 Published online by Cambridge University Press

formats including OBJ Wavefront. Furthermore, with CAD Assistant[9] a free, simple, and intuitive graphical user interface (GUI) exists, which builds on top of Open CASCADE Technology and with which end-users can perform the data conversion easily. Note that when using the Unreal Engine, the free and open-source Open CASCADE Technology library could be integrated into the VR application directly, thus, preventing the need for an external data conversion step. When using the Unity Engine, a .NET wrapper around the Open CASCADE Technology library exists, which also can be used for integrating the data conversion directly into the VR environment, but which is not free and not open source. Alternatively, you can write your own .NET wrapper around the Open CASCADE Technology library, which exposes only the data conversion functionality needed.
Finally, due to the lower entry barrier for software developers, we suggest using the Unity Engine as the underlying VR development environment and application framework. Furthermore, we suggest using OBJ Wavefront as the data format for the CAD models. Then, we suggest performing the conversion between the STEP and OBJ Wavefront externally using CAD Assistant. And lastly, we suggest adding the free and open-source Math.NET Numerics[10] library for performing mathematical computations, e.g., for implementing the collaborative carrying and threatening step as explained in Section 3.3.2.

3.2. Basic features

Based on the selected software technology stack, any virtual assembly environment must provide some basic features, which are independent of the concrete use case (e.g., cooling cabinets). These features include geometry tessellation (see Section 3.2.1), part selection and annotation (see Section 3.2.2), grabbing, moving, and snapping (see Section 3.2.3), collision detection and physical simulation (see Section 3.2.4), as well as online collaboration (see Section 3.2.5).

3.2.1. Geometry tessellation

Tessellation of the original CAD geometry data is necessary before the CAD models can be used in gaming engines such as the Unity Engine. Note that tessellation refers to the process of turning complex geometrical surface descriptions into simple triangle meshes. In general, during this process the original surface cannot be reconstructed exactly, but a small approximation error is introduced instead. The approximation error typically decreases with an increasing number of triangles, while the number of triangles can be controlled with the parameters of the tessellation algorithms (see Figure 4). On the other hand, the computational effort during virtual assembly simulation also increases with an increasing number of triangles. Hence, in practice an appropriate balance must be found between approximation error and computational effort.

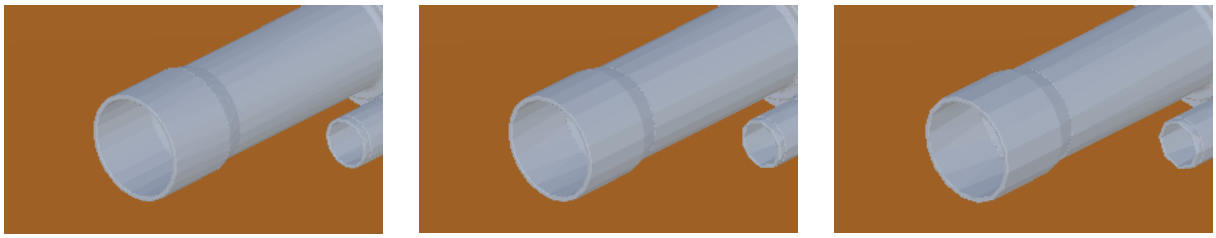

Figure 4. Tessellation results with high (left), medium (middle), and low (right) quality

3.2.2. Part selection and annotation

Then, part selection and annotation are required for interacting with the CAD model and linking design feedback (see Figure 5). For selection you can use basic ray casting, so that the parts of the product can be selected even from a distance, combined with visual highlighting of the selected part. Additionally, we tested a text display on top of the main hand controller showing the names of the selected parts. Note that OBJ Wavefront cannot represent the assembly hierarchy explicitly, but the hierarchy information can be reconstructed from the object names. In contrast, for CAD model annotation we tested two

[9] https://www.opencascade.com/products/cad-assistant/
[10] https://numerics.mathdotnet.com/

https://doi.org/10.1017/pds.2024.61 Published online by Cambridge University Press

options: a visual painting tool and an audio recording tool. The painting tool shall help users to highlight important shape characteristics, while the recoding tool shall help users to explain the issue that they have found. Both tools together can provide a rich documentation of the review results, that can inform follow-up project activities such as redesign efficiently and effectively. In the future, we plan to upload the annotations directly to (Hackenberg et al. 2023).

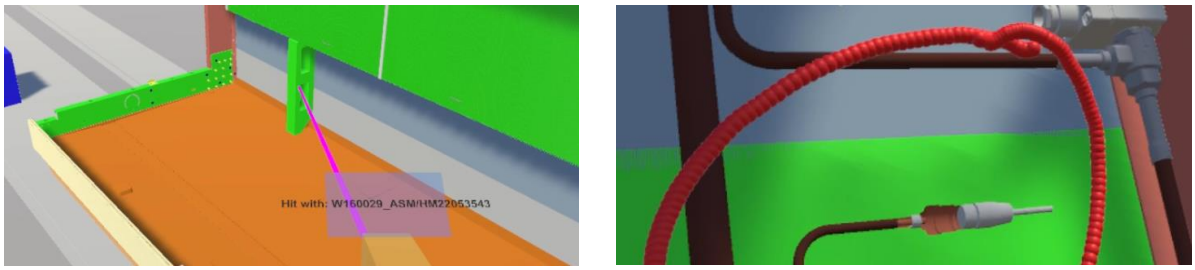

Figure 5. Screenshot of the selection (left) and the annotation (right) feature

3.2.3. Part grabbing, moving, and snapping

Based on the selection feature from the previous section, a general grabbing, moving, and snapping technique can be implemented (see Figure 6). The technique should allow the user to pick any part of the product and place it anywhere in the virtual assembly environment. For each such part, the system should remember the original position and orientation (or pose) in the CAD model. Consequently, during part movement a transparent version of each part can be displayed at its original pose to inform the user about where the part needs to be placed ultimately. Finally, when the user comes close to the original pose while moving a part, the part should snap to this pose to overcome inherent accuracies in VR simulation.

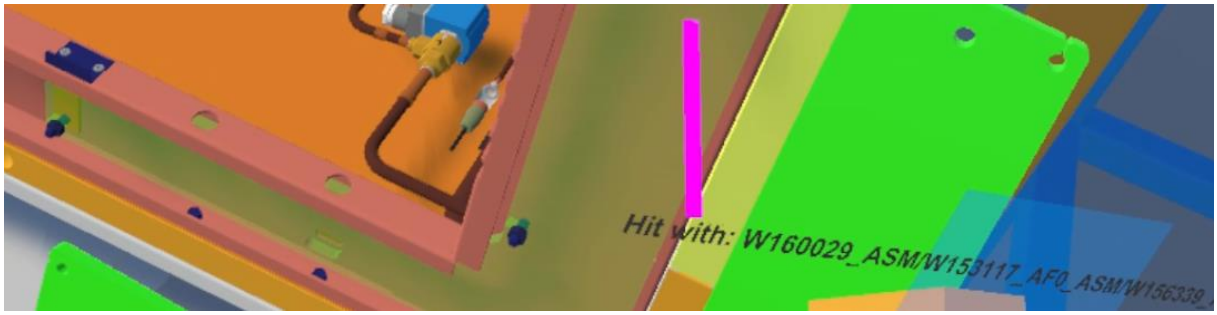

Figure 6. Screenshot of the grabbing, moving, and snapping feature

3.2.4. Collision detection and physical simulation

Then, a virtual assembly environment should support collision detection and physical simulation (see Figure 7). Collision detection helps detecting mountability issues due to limitations of the working space. Physical simulation helps making the environment behave more naturally, e.g., when tools are placed anywhere in the scene. For both features collision meshes need to be added relevant objects. Note that due to performance constraints in gaming engines typically not the original shapes of 3D objects are used for collision detection and physical simulation. Instead, the Unity Engine, e.g., provides an out-of-the-box algorithm for computing reduced collision meshes from any given source mesh. The algorithm can be configured such that it produces either convex or non-convex collision meshes. However, the collision detection algorithm of the Unity Engine can only detect collisions between pairs of collision meshes, if at least one of the meshes is convex. This limitation has been introduced because convex meshes can be processed more efficiently than non-convex ones. Note that this limitation introduces an inaccuracy in the results of collision detection (i.e., collisions between objects can be detected, which do not collide with each other). Consequently, these results must be taken with a grain of salt and mountability issues due to limitations of the working space cannot be derived directly.

https://doi.org/10.1017/pds.2024.61 Published online by Cambridge University Press

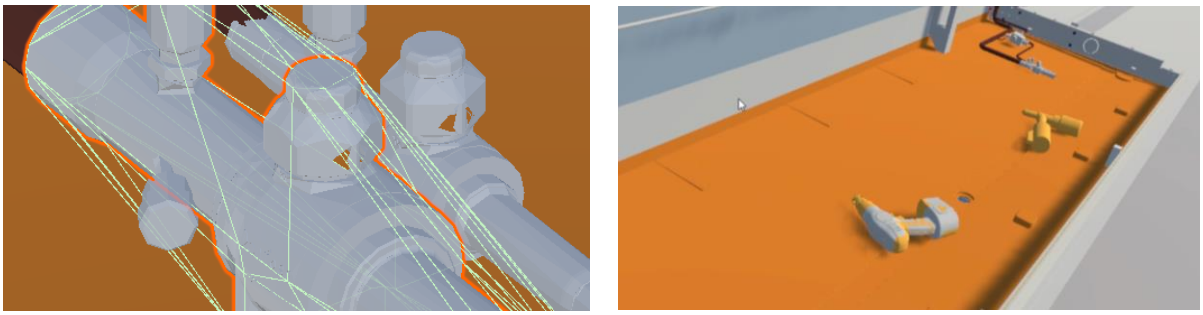

Figure 7. Screenshot of the collision detection (left) and physical simulation (right) feature

3.2.5. Online collaboration

Finally, basic online multi-player collaboration (see Figure 8) should be included in the virtual assembly environment for several reasons: First, different stakeholders such as customers, product managers, quality managers, product designers, and assembly workers might want to meet in the VR environment and review the product design collaboratively. And second, you might want to simulate collaborative assembly steps (such as collaborative carrying and threatening in our case) with two or more test persons. Consequently, the virtual assembly environment should support at least basic avatars for the different users connected to the same virtual assembly session. Note that you should provide visual cues, which help distinguishing the different users of the virtual assembly session from each other. We decided to add name signs for this purpose, which suffices as basic means. Then, for each user the state of the headset, the controllers, and additional body trackers must be tracked and synchronized with the other clients. Note that the state of the headset and the body trackers is given by their position and orientation, while the state of the controllers additionally includes the button interactions. The button interactions are required for replicating the interaction states on all the clients such as whether a user has grabbed a specific part of the product or not.

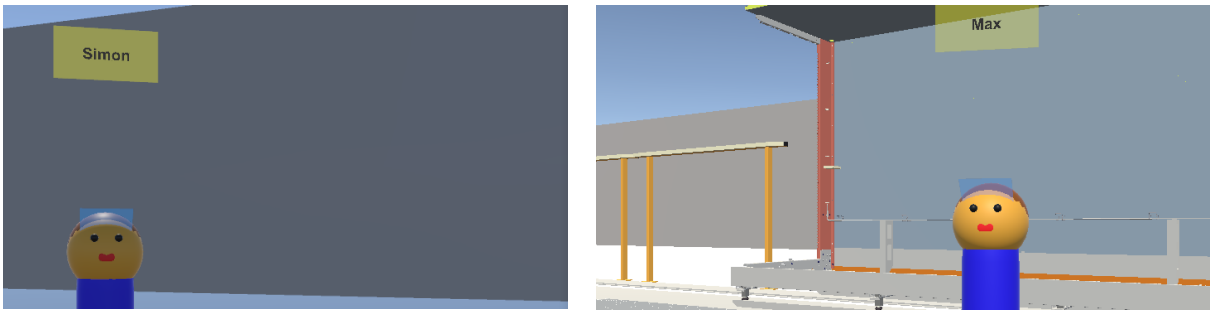

Figure 8. Screenshot of the online collaboration feature

Technically, we implemented the online collaboration feature based on the Netcode framework of the Unity Engine. Consequently, we defined network objects, behaviours, and variables, which enable the automatic synchronization of selected state information between the different clients. Note that the underlying protocols are optimized for large online multi-player gaming scenarios. Hence, we expect the technology to work for the use case of collaborative virtual assembly.

3.3. Assembly steps

After having defined the software technology stack as well as basic features of a virtual assembly environment, we turn our attention to the implementation of advanced and task-specific interaction techniques. In the following, we first explain siliconizing in Section 3.3.1, which might be necessary to close small gaps between physical parts. Then, we explain collaborative carrying of large physical parts in Section 3.3.2, which in our case must be realized with two human workers and which requires the online collaboration feature introduced previously. Finally, we summarize our experience in adding tube bending capabilities to the virtual assembly environment in Section 3.3.3, which required additional preprocessing of the CAD models to work properly.

https://doi.org/10.1017/pds.2024.61 Published online by Cambridge University Press

3.3.1. Gap siliconizing

The starting point for the implementation of the gap siliconizing feature were the CAD model of the cooling cabinet, which included a path geometry for the silicon trace, as well as the CAD model of the silicon gun. Note that tool vendors might provide CAD models of their tools. Alternatively, one can prepare custom CAD models of the tools. In general, the models of the tools must be accurate enough so that mountability issues due to limitations of the working space can be detected reliably. Furthermore, we needed to perform some manual preparation for using the silicon gun during VR simulation (see Figure 9). First, we added a sphere at the tip of the gun to mark the effective operating position. And second, we added a box sphere to mark the trigger area of the gun. Note that the user must hold and guide the gun with the first hand/controller, while using the second hand/controller for triggering.

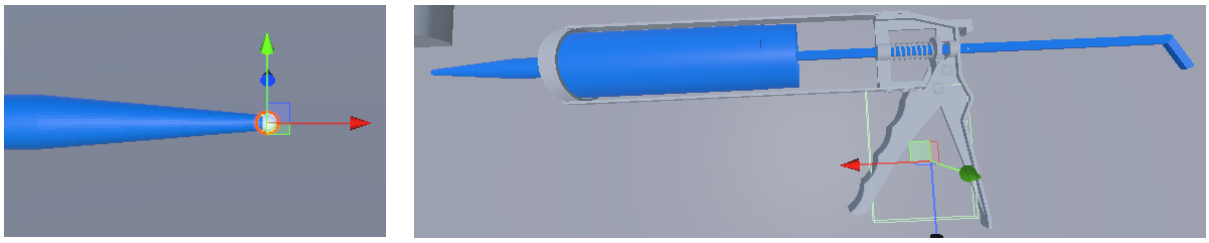

Figure 9. Model of silicon gun with a collision sphere at tip and a collision box at trigger

Then, we found out that the path geometry of the silicon trace is lost during conversion between the STEP format and the OBJ Wavefront format. The reason for this behaviour is, that the OBJ Wavefront format does not support two-dimensional geometries such as lines. For this reason, we decided to convert the original path geometry of the silicon trace into a volume geometry with minimal spatial extent. Consequently, you can detect hits of the silicon trace through simple collision detection with the tip of the silicon gun (see Figure 10).

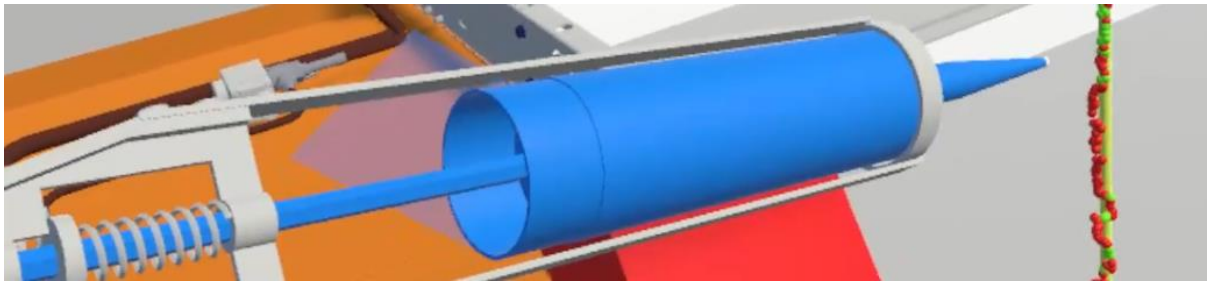

Figure 10. Detection of silicon trace hits (green spheres) with the tip of the silicon gun

Finally, you might want to detect automatically, whether a silicone trace has been covered completely or not. One option would be, to define additional collision geometries at the start and the end of the silicone trace. Consequently, you can check whether a continuous sequence of silicone trace hits from start to end has been recorded. Note that the collision geometries at the start and the end of the silicone trace can be added manually with to the CAD model with minimal effort.

3.3.2. Collaborative part carrying

Next, we implemented the collaborative part carrying feature allowing two users to grab and move a common part simultaneously (see Figure 11). Such behaviour can be implemented in multi-player mode using two computers connected over a local area network and each equipped with one VR headset and two VR controllers for the hands. In our implementation, the carrying process starts as soon as the avatars of all four VR controllers (i.e., standard box geometries for simplicity) collide with the part, that needs to be carried collaboratively, and on each controller a specific button is pressed. Afterwards, the position of the part can be computed by fitting a plane to the four controller positions, which can be done using the singular value decomposition (SVD) algorithm as implemented in the Math.NET Numerics library. Note that plane fitting is required, because the four hands of the two users practically never form a perfect working plane and, hence, small deviations must be compensated.

https://doi.org/10.1017/pds.2024.61 Published online by Cambridge University Press

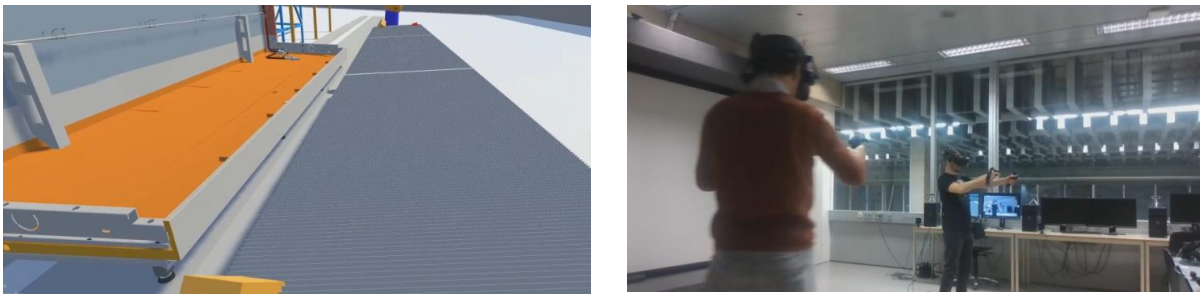

Figure 11. Collaborative carrying of a large physical part in a multi-player setting

Plane fitting essentially finds a plane determined by a point in the plane and a normal vector of the plane, which best describes the four hand positions. One challenge we faced here was that the normal vector of this plane might flip instantly by 180 degrees, because the SVD algorithm does not guarantee a certain orientation of the normal vector. The flipping can be compensated by comparing the current normal vector to the previous normal vector and avoiding rotations surpassing a certain limit angle.

3.3.3. Tube bending

Finally, we implemented tube bending into the virtual assembly environment (see Figure 12). It is worth noting that CAD models typically contain the part geometries in their target state, while source as well as intermediate states needed to be reconstructed manually during a preprocessing step. For this study, we decided to prepare a separate tube model in the free and open-source 3D application Blender[11]. We used a basic cylinder geometry for representing the tube and rigged the geometry with a simple skeleton model consisting of two bones and one joint at the bending point. Then, we imported this tube model into the Unity Engine and successfully tested bending state variation.

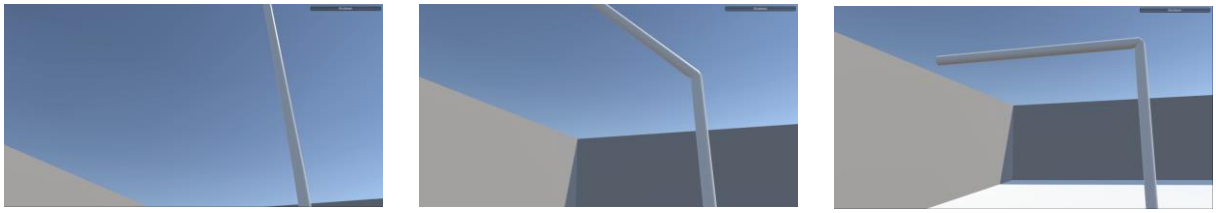

Figure 12. Status of the tube before (left), during (middle), and after (right) beding

Note that with this custom tube model we can detect mountability issues that might arise during specific intermediate states, which is not possible if you only have the geometry data for the target state. However, on the downside an additional manual preprocessing step of the original CAD models is required to make this feature work. And in this case, you need special knowledge of rigged geometry, which is more common among computer animation experts.

4. Conclusion

In this paper, we presented learnings from developing a custom virtual assembly environment for cooling cabinets. In **RQ1** we investigated open source software technologies, which can serve as a basis for virtual assembly environment implementation. We suggest using OpenXR as a vendor- and device-independent means for interfacing with VR equipment, the Unity Engine as the core development environment and application framework, and the Math.NET Numerics library for performing numerical computations such as SVD. Furthermore, we suggest using OpenCASCADE Technology and the CAD Assistant for converting CAD models from STEP into OBJ Wavefront format. The selection process roughly took one person month. In **RQ2** we investigated the implementation of basic features for virtual assembly environments and their implementation costs. The basic features comprise part selection and visual/auditive annotation, part grapping, moving, and snapping, collision detection and physical simulation, and online collaboration. The implementation of these features roughly took two person

[11] https://www.blender.org/

https://doi.org/10.1017/pds.2024.61 Published online by Cambridge University Press

months. In **RQ3** we investigated the implementation of specific assembly steps such as siliconizing, collaborative part carrying, and tube bending as well as their implementation costs. For siliconizing we relied on extrusion of the silicone path and collision detection to detect path hits and calculate path coverage. For collaborative carrying we relied on multi-player synchronization and plane fitting to compute the part position and orientation. For tube bending we relied on rigged mesh geometry to compute intermediate states. The implementation of these features roughly took three person months. In summary, we found that open source technology is well suited for implementing virtual assembly, but return on investment likely requires heavy reuse of feature implementations across assembly studies.

References

Abidi, Mustufa Haider; Al-Ahmari, Abdulrahman; Ahmad, Ali; Ameen, Wadea; Alkhalefah, Hisham (2019): Assessment of virtual reality-based manufacturing assembly training system. In: *The International Journal of Advanced Manufacturing Technology* 105 (9), S. 3743–3759. https://dx.doi.org/10.1007/s00170-019-03801-3.

Choi, A. C. K.; Chan, D. S. K.; Yuen, A. M. F. (2002): Application of Virtual Assembly Tools for Improving Product Design. In: *The International Journal of Advanced Manufacturing Technology* 19 (5), S. 377–383. https://dx.doi.org/10.1007/s001700200027.

D'Ambrosio, Joseph; Soremekun, Grant (2017): Systems engineering challenges and MBSE opportunities for automotive system design. In: 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), S. 2075–2080.

Florian Dyck; Harald Anacker; Roman Dumitrescu (2023): Virtual Assembly for Engineering – A Systematic Literature Review. In: *Procedia CIRP* 118, S. 912–917. https://dx.doi.org/10.1016/j.procir.2023.06.157.

Hackenberg, Georg; Zehetner, Christian; Frühwirth, Dominik (2023): GITHUB FOR PRODUCT DEVELOPMENT - HOW COULD THAT LOOK LIKE? In: *Proceedings of the Design Society* 3, S. 2055–2064. https://dx.doi.org/10.1017/pds.2023.206.

Jayaram, Sankar; Jayaram, Uma; Kim, Young Jun; DeChenne, Charles; Lyons, Kevin W.; Palmer, Craig; Mitsui, Tatsuki (2007): Industry case studies in the use of immersive virtual assembly. In: *Virtual Reality* 11 (4), S. 217–228. https://dx.doi.org/10.1007/s10055-007-0070-x.

Kiara Ottogalli, Daniel Rosquete, Javier Rojo, Aiert Amundarain, José María Rodríguez; Diego Borro (2021): Virtual reality simulation of human-robot coexistence for an aircraft final assembly line: process evaluation and ergonomics assessment. In: *International Journal of Computer Integrated Manufacturing* 34 (9), S. 975–995. https://dx.doi.org/10.1080/0951192X.2021.1946855.

Ma, Dengzhe and Zhen, Xijin and Hu, Yong and Wu, Dianliang and Fan, Xiumin and Zhu, Hongmin (2011): Collaborative Virtual Assembly Operation Simulation and Its Application. In: Ma, Dengzhe and Fan, Xiumin and Gausemeier, Jürgen and Grafe, Michael (Hg.): Virtual Reality & Augmented Reality in Industry. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 55–82.

Maciej A Bossak (1998): Simulation based design. In: *Journal of Materials Processing Technology* 76 (1), S. 8–11. https://dx.doi.org/10.1016/S0924-0136(97)00308-7.

Ming C. Leu; Hoda A. ElMaraghy; Andrew Y.C. Nee; Soh Khim Ong; Michele Lanzetta; Matthias Putz et al. (2013): CAD model based virtual assembly simulation, planning and training. In: *CIRP Annals* 62 (2), S. 799–822. https://dx.doi.org/10.1016/j.cirp.2013.05.005.

Mustufa H. Abidi; Ali Ahmad; Saber Darmoul; Abdulrahman M. Al-Ahmari (2015): Haptics Assisted Virtual Assembly. In: *IFAC-PapersOnLine* 48 (3), S. 100–105. https://dx.doi.org/10.1016/j.ifacol.2015.06.065.

Q.-H. Wang, J.-R. Li; H.-Q. Gong (2006): A CAD-linked virtual assembly environment. In: *International Journal of Production Research* 44 (3), S. 467–486. https://dx.doi.org/10.1080/00207540500319294.

Sankar Jayaram; Uma Jayaram; Yong Wang; Tirumali, H.; Lyons, K.; Hart, P. (1999): VADE: a Virtual Assembly Design Environment. In: *IEEE Computer Graphics and Applications* 19 (6), S. 44–50. https://dx.doi.org/10.1109/38.799739.

Wan, Huagen; Gao, Shuming; Peng, Qunsheng; Dai, Guozhong; Zhang, Fengjun (Hg.) (2004): MIVAS: A Multi-Modal Immersive Virtual Assembly System (International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Volume 4: 24th Computers and Information in Engineering Conference).

Wang, Yong; Jayaram, Uma; Jayaram, Sankar; Lyons, Kevin (Hg.) (2001): Physically Based Modeling in Virtual Assembly (International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Volume 1: 21st Computers and Information in Engineering Conference).

X. Wang; S.K. Ong; A.Y.C. Nee (2016): Real-virtual components interaction for assembly simulation and planning. In: *Robotics and Computer-Integrated Manufacturing* 41, S. 102–114. https://dx.doi.org/10.1016/j.rcim.2016.03.005.