

# Augmented Reality UNIX C++ Engine for Enhanced Visual Guidance in Digital Fabrication

Andrea Settimi<sup>1</sup>, Hong-Bin Yang<sup>1</sup>, Julien Gamarro<sup>2</sup>, and Yves Weinand<sup>1</sup>

<sup>1</sup> IBOIS EPFL, Switzerland <sup>2</sup> Independent Researcher, Switzerland ¶ Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

## Software

- [Review](#)
- [Repository](#)
- [Archive](#)

Editor: [Open Journals](#)

## Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

## Summary

Augmented Carpentry is a lightweight and fast-developing UNIX C++ engine for prototyping AR applications leveraging bleeding-edge robotic vision research for digital fabrication. It features a modular layer-stack flow, a geometry framework for managing 3D objects, a computed feedback system for visual guidance, and an AR rendering system for synthesizing digital instructions into a simple monocular camera feed.

## Statement of need

Augmented Carpentry (AC) addresses critical limitations in existing augmented reality (AR) tools for digital fabrication. CompasXR ([Kenny et al., 2024](#)), the only open-source AR tool available in the digital fabrication field, provides a valuable common platform, particularly for assembly tasks. However, it currently lacks a streamlined integration pipeline for advanced robotic vision technologies due to its reliance on Unity ([Unity Technologies, 2023](#)) and the Windows operating system (OS). In the field of AR fabrication, developers from the current Incon.ai ([Furrer et al., 2024](#)) represent the peak of AR engine innovation with robotic vision algorithm integration for digital fabrication in research ([Mitterberger et al., 2020](#); [Sandy et al., 2016](#); [Sandy & Buchli, 2018](#)), nevertheless, its codebase remains unavailable to the public.

AC aims to fill this gap by providing a lightweight, open-source, and UNIX-compatible C++ engine for AR applications in digital fabrication. Its software architecture is similar to existing free engines ([T. ezEngine Contributors, 2024](#); [T. T. 3D. Contributors, 2024](#); [Linietsky et al., 2024](#)), yet it prioritizes rapid prototyping, flexibility, and customization for extended reality (XR) manufacturing using accessible sensors and hardware. Unlike feature-rich game engines with excessive functionalities or proprietary constraints ([Epic Games, 2019](#); [Unity Technologies, 2023](#)), AC is lightweight, aided by the adoption of a bloat-free UI system ([T. D. Contributors, 2024](#)), and maintains full compatibility with Linux systems—crucial for integrating the latest open-source robotic vision technologies in AR manufacturing.

## Layer-stack flow

The main AR engine is managed by a layer-stack flow. Designed as a modular system, each layer encapsulates the code for a specific domain of the AR application, such as camera processing, sensor's self-localization, object tracking, UI, and rendering. The general order and expansion of these layers can be configured in the top-level main file `ACApp.cpp`. This architecture provides flexibility to customize key AR features as needed, including integrating new sensors, modifying the rendering pipeline, or adapting camera pose estimation methods. For instance, users can implement pose estimation based on tags ([Muñoz-Salinas et al., 2019](#)), features ([Campos et al., 2021](#)), or hybrid ([Settimi et al., 2024](#)) approaches as supported by

the software out of the box.

Each layer in the stack inherits from a superclass interface defined in `Layer.h`, which includes event-like methods triggered at various points during frame processing (e.g., `OnFrameAwake()`, `OnFrameStart()`, etc). These methods are invoked by the main `Run()` function in the singleton application loop from `Application.h`. This design allows application tasks to be containerized and executed sequentially while facilitating data exchange between specific layers through the `AIAC_APP` macro, enabling the retrieval of any particular layer data. Exchange between layers can also take place in a more structured way with the integrated event system (`ApplicationEvent.h`), which is capable of queuing events from layers and trigger them in the next main loop.

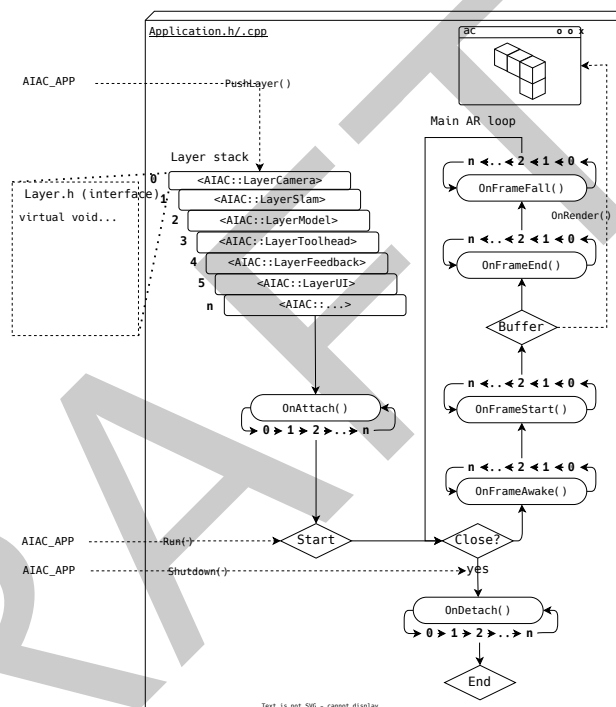


Figure 1: Illustration of the layer-stack design and the main loop for the AR engine.

## Geometry framework

The geometry framework provides a uniform infrastructure to handle all 3D objects present in the scene, including the CAD model, scanned models, and the fabrication instructions. This framework not only allows application layers to interact with the 3D object easily but is also tightly integrated with the rendering system and manages the OpenGL resources implicitly to ease the work for application layers.

The geometry is classified by the following primitive shapes: point, line, circle, cylinder, polyline, triangle, mesh, and text. Each primitive shape is a class (e.g. `G0Point`, `G0Line`, `G0Circle`, etc) inheriting from the base class `G0Primitive`, where `GO` stands for Geometry Object. The system also maintains a global table `G0Registry` to keep track of all the geometry objects. When a `GO` initializes, it registers itself in a global table with a unique `UUID`. As the table is exposed to the entire system, application layers can acquire specific objects through their `UUIDs` or iterate through all objects to perform operations.

## 63 Computed Feedback System

64 The LayerFeedback.h module manages the computation of all essential data required to  
65 provide visual guidance to users during the fabrication process. Feedback computation primarily  
66 relies on data retrieved from two preceding layers:

- 67 1. LayerModel.h: contains the execution model and geometries associated with the currently  
68 active hole or cut.
- 69 2. LayerToolhead.h: provides similar information, but specific to the toolhead currently  
70 attached to the tool.

71 Feedback is categorized based on similar operations, such as drilling (HoleFeedback.h), circular  
72 cutting (CutCircularSawFeedback.h), and chainsaw cutting (CutChainSawFeedback.h). Each  
73 feedback category inherits from an interface class (AIAC/Feedback/FabFeedback.h), which  
74 defines high-level control functions like Update(), Activate(), and Deactivate().

75 The visual guidance for each tool may consist of multiple visual cues, most of which are  
76 implemented using the template FeedbackVisualizer.h. These internal components (e.g.,  
77 CutBladeThicknessVisualizer.h or CutPlaneVisualizer.h) handle their own geometric  
78 visual cue calculations and store representations as G0 instances in a member vector of the  
79 corresponding superclass. Visualization of these G0 elements, and thus the feedback itself,  
80 can be selectively enabled or entirely toggled on/off using the Activate() and Deactivate()  
81 functions.

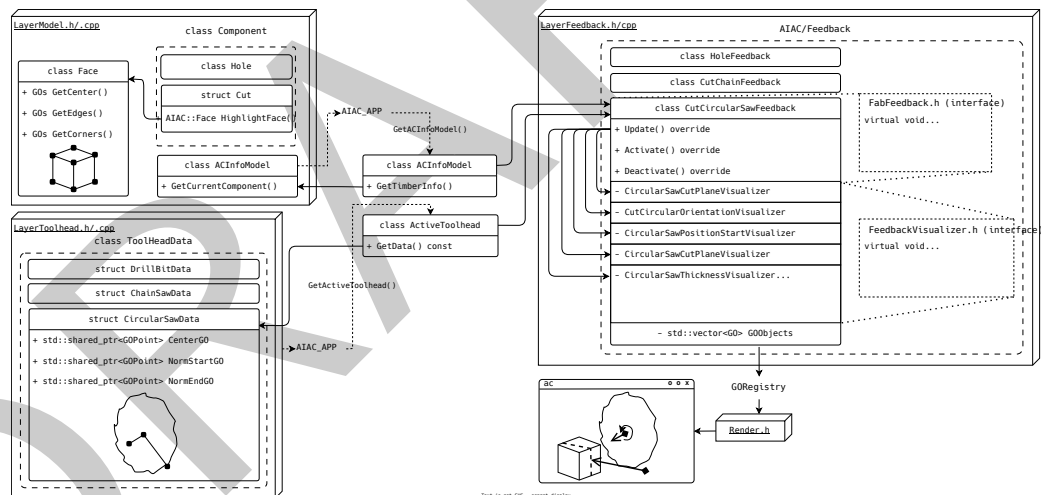


Figure 2: Dataflow for the functioning of the Augmented Carpentry's feedback system.

## 82 AR rendering

83 We would like to thank all the contributors to the Augmented Carpentry project, including  
84 the developers, researchers, and users who have provided valuable feedback and suggestions.  
85 Special thanks to the GIS and the Center for Imaging EPFL groups, for their support throughout  
86 the development process.

## 87 References

- 88 Campos, C., Elvira, R., Rodriguez, J. J. G., Montiel, J. M. M., & Tardos, J. D. (2021).  
89 ORB-SLAM3: An accurate open-source library for visual, visual inertial, and multimap  
90 SLAM. *IEEE Transactions on Robotics*, 37(6), 1874–1890. <https://doi.org/10.1109/tro.2021.3075644>  
91

- Contributors, T. D. (2024). *Dear ImGui: Bloat-free Graphical User interface for C++ with minimal dependencies* (Version 1.91.5). <https://github.com/ocornut/imgui>
- Contributors, T. ezEngine. (2024). *EzEngine engine* (Version 4.0.3). <https://ezengine.net/>
- Contributors, T. T. 3D. (2024). *Torque 3D engine* (Version 24.9). <https://github.com/TorqueGameEngines/Torque3D>
- Epic Games. (2019). *Unreal engine* (Version 4.22.1). <https://www.unrealengine.com>
- Furrer, F., Stich, M., Regenass, F., & Mansfield, B. (2024). *Instructive Construction*. <https://incon.ai/>.
- Kenny, J., Mitterberger, D., Casas, G., Alexi, E., Gramazio, F., & Kohler, M. (2024). *COMPAS XR: Extended reality workflows for the COMPAS framework*. [https://github.com/compas-dev/compas\\_xr/](https://github.com/compas-dev/compas_xr/). <https://doi.org/10.5281/zenodo.12514526>
- Linietzky, M., Manzur, A., Verschelde, R., & others, many. (2024). *Godot Engine – Multi-platform 2D and 3D engine* (Version 4.3). <https://github.com/godotengine/godot?tab=coc-ov-file>
- Mitterberger, D., Dörfler, K., Sandy, T., Salveridou, F., Hutter, M., Gramazio, F., & Kohler, M. (2020). Augmented bricklaying. *Construction Robotics*, 4(3-4), 151–161. <https://doi.org/10.1007/s41693-020-00035-8>
- Muñoz-Salinas, R., Marín-Jimenez, M. J., & Medina-Carnicer, R. (2019). SPM-SLAM: Simultaneous localization and mapping with squared planar markers. *Pattern Recognition*, 86, 156–171. <https://doi.org/10.1016/j.patcog.2018.09.003>
- Sandy, T., & Buchli, J. (2018). Object-based visual-inertial tracking for additive fabrication. *IEEE Robotics and Automation Letters*, 3(3), 1370–1377. <https://doi.org/10.1109/lra.2018.2798700>
- Sandy, T., Giftthaler, M., Dorfler, K., Kohler, M., & Buchli, J. (2016, May). Autonomous repositioning and localization of an in situ fabricator. *2016 IEEE International Conference on Robotics and Automation (ICRA)*. <https://doi.org/10.1109/icra.2016.7487449>
- Settimi, A., Yang, H.-B., Gamero, J., & Weinand, Y. (2024). TSLAM: A tag-based object-centered monocular navigation system for augmented manual woodworking. *Construction Robotics*, 8(1). <https://doi.org/10.1007/s41693-024-00118-w>
- Unity Technologies. (2023). *Unity* (Version 2023.2.3). <https://unity.com/>