




1 Augmented Reality UNIX C++ Engine for Enhanced 2 Visual Guidance in Woodworking

3 **Andrea Settimi** ¹, **Hong-Bin Yang** ¹, **Julien Gamarro** ², and **Yves**
4 **Weinand** ¹

5 ¹ IBOIS EPFL, Switzerland ² 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Published: unpublished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](#))

6 Summary

7 Augmented Carpentry is a lightweight and fast-developing C++ engine for prototyping and
8 scaling AR applications. It features a modular layer-stack flow, a geometry framework for
9 managing 3D objects, a computed feedback system for visual guidance, and an AR rendering
10 system for synthesizing digital instructions with the real world.

11 Statement of need

12 Functionalities

13 Layer-stack flow

14 The layer stack is primarily responsible for managing the flow control of the AR engine.
15 Designed as a modular system, each layer encapsulates the code for a specific domain of the
16 AR application, such as camera processing, object tracking, UI, and rendering. The general
17 order and expansion of these layers can be configured in the top-level main file `ACApp.cpp`.

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

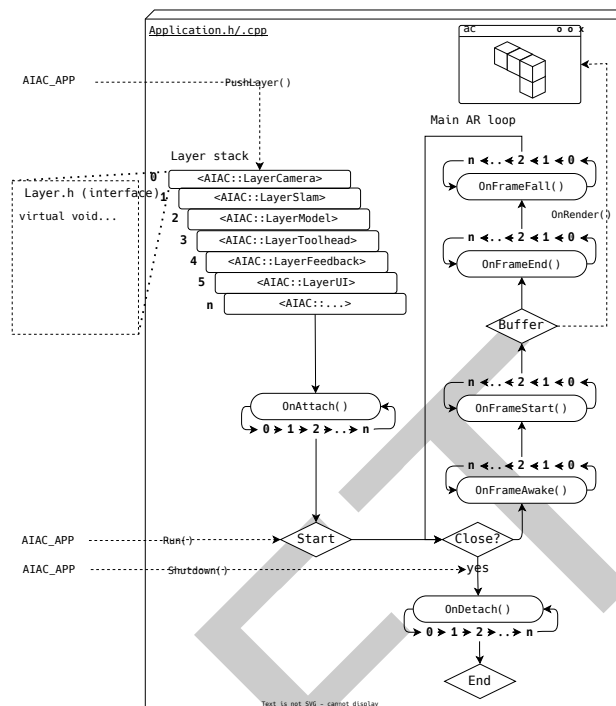


Figure 1: Illustration of the layer-stack design and the main loop for the AR engine.

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.

Computed Feedback System

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

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

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

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

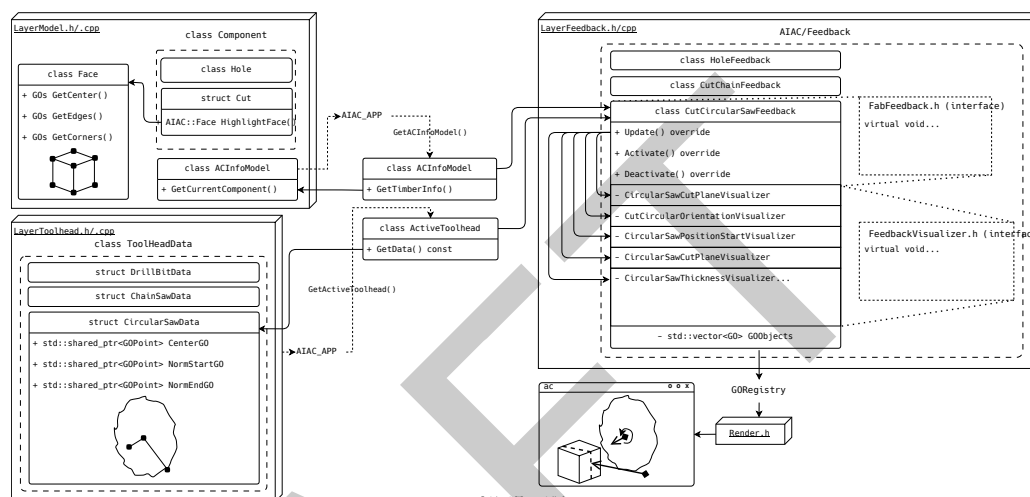


Figure 2: Dataflow for the functioning of the Augmented Carpentry's feedback system.

AR rendering

References