The Journal of Open Source Software

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

Andrea Settimi [1¶], **Hong-Bin Yang** [1], **Julien Gamerro** [2], and **Yves Weinand** [1]

1 IBOIS EPFL, Switzerland **2** Independent Researcher, Switzerland ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

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

test (Linietsky et al., n.d.)

Augmented Carpentry (AC) addresses critical limitations in existing augmented reality (AR) tools for digital fabrication. For example, CompasXR, the only open-source AR tool available, focuses on assembly tasks but lacks integration with advanced robotic vision technologies due to its reliance on Unity and C# wrapping. AC provides a lightweight, scalable C++ engine tailored for UNIX platforms, enabling rapid prototyping while remaining open-source to facilitate customization and exploration of niche research scenarios. Unlike cumbersome game engines with excessive features or proprietary constraints, AC is streamlined to support bleeding-edge robotics applications and maintain full compatibility with Linux systems, which is crucial for integrating custom vision pipelines in AR.

Layer-stack flow

The main AR engin flow is managed by a modular layer-stack system. Designed as a modular system, each layer encapsulates the code for a specific domain of the AR application, such as camera processing, object tracking, UI, and rendering. The general order and expansion of these layers can be configured in the top-level main file `ACApp.cpp`.

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.

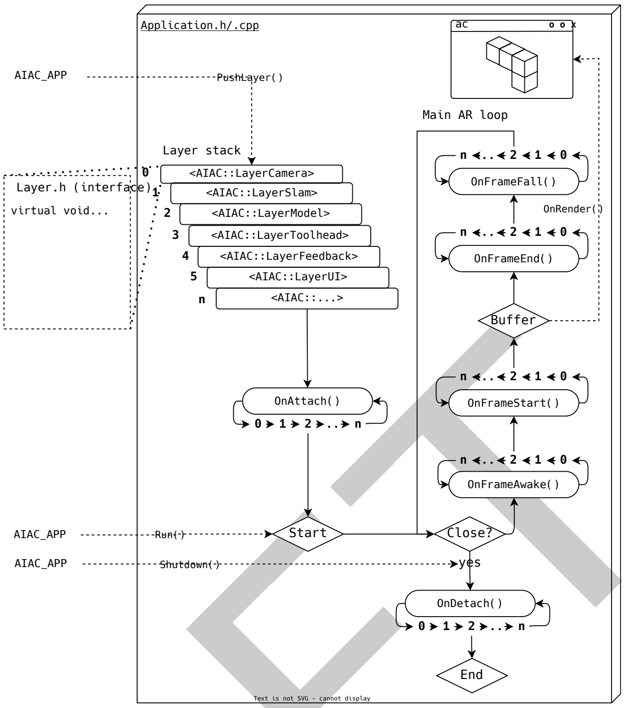

Figure 1: Illustration of the layer-stack design and the main loop for the AR engine.

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. `GOPoint`, `GOLine`, `GOCircle`, etc) inheriting from the base class `GOPrimitive`, where GO stands for Geometry Object. The system also maintains a global table `GORegistry` 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 `GO` instances in a member vector of the corresponding superclass. Visualization of these `GO` elements, and thus the feedback itself, can be selectively enabled or entirely toggled on/off using the `Activate()` and `Deactivate()` functions.

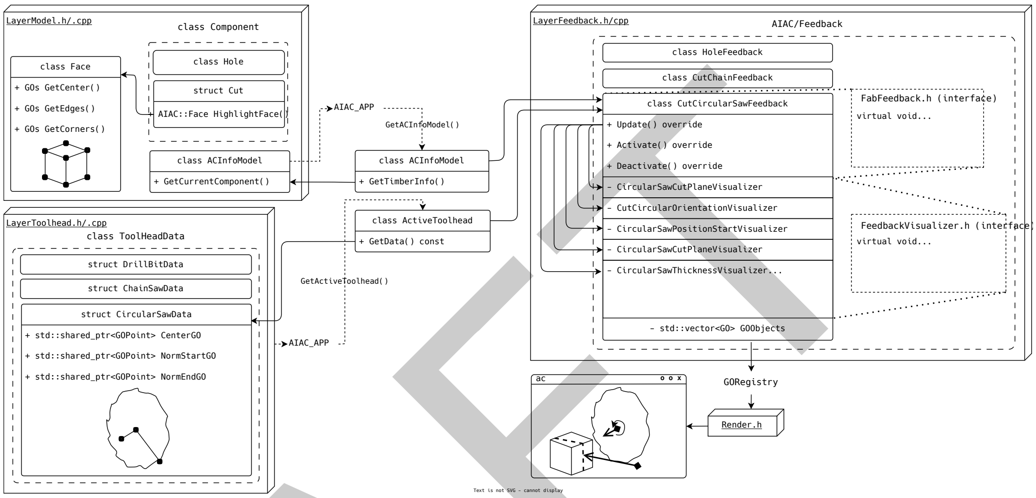

Figure 2: Dataflow for the functioning of the Augmented Carpentry's feedback system.

AR rendering

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

References

Linietsky, M., Manzur, A., Verschelde, R., & others, many. (n.d.). *Godot Engine – Multi-platform 2D and 3D game engine*. https://github.com/godotengine/godot?tab=coc-ov-file