# Digital Twin Modeling Framework for LCLS Motion

A CAD-driven simulation format for visualizing motorized stage motion, identifying mechanical interference, and supporting more interference-resistant assembly design.

Koa Shen (Intern) · Elon Goliger (Mentor) · LCLS Internship Program · SLAC National Accelerator Laboratory · LCLS Engineering Division · X-Ray Correlation Spectroscopy (XCS)

UC SANTA BARBARA

SLAC NATIONAL ACCELERATOR LABORATORY · U.S. DEPARTMENT OF ENERGY | Stanford University · LCLS

## Motivation and Challenge

Motorized stage stacks position optics and detectors with high precision throughout LCLS experimental systems. These assemblies combine many translating and rotating stages within tightly packed mechanical envelopes. Although conventional CAD captures component geometry and fixed assembly poses, it does not reliably describe motion axes, travel limits, or downstream movement for complicated systems with many degrees of freedom. This makes interference difficult to anticipate from a static model alone. This project, Twin Lab, converts reviewed CAD designs into interactive digital twins so engineers can examine the full range of an assembly's motion before manufacturing, modifying, or operating its hardware.

**Project Objective:**

Create a flexible modeling workflow that combines assembly geometry with kinematic information. The resulting simulation should reproduce stage-stack motion on command, expose potential collisions, and help engineers redesign adapters, enclosures, and neighboring assemblies to remain interference-resistant throughout their operating ranges.

**Key Requirements:**

The framework must utilize existing CAD as its source of geometry while allowing engineers to define motion separately. It must support linear and rotary stages, compound motion chains, OEM travel limits, repeatable home poses, and revision-aware updates. Most importantly, the displayed geometry and collision model must share the same kinematics.

## Framework and Architecture

Twin Lab separates automatically extracted geometry from human-reviewed engineering decisions. Open CASCADE imports the STEP assembly, traverses its hierarchy, and generates stable references for individual occurrences. A reusable stage catalog records manufacturer and model information, while an assembly-specific YAML inventory assigns stage instances, motion chains, joint types, axes, operating limits, and home positions. This setup requires no CAD modification while making the intended mechanical behavior of the system explicit, inspectable, and reusable.

The reviewed inventory is compiled into a Drake kinematic model and displayed via MeshCat, a browser-based visualizer. Users can move any of the assembly's joints with sliders, reset the system to its reviewed home pose, or animate coordinated motion across the entire stack. Because the viewer and interference checker are generated from the same inventory, simulated motion and collision queries can be accessed simultaneously through one MeshCat instance.

**Implementation Workflow**

1. Import the assembly STEP file, generate manifest, tessellate geometry with Open CASCADE, and generate hulls with CoACD.
2. Add DOF:  limits, homes, motion chains, and attachments.
3. Generate MeshCat geometry from source model.
4. Move individual joints or animate trajectories with controls.
5. Use detected interference to guide mechanical redesign.

## Results, Impact, and Future Work

The demonstration models assembly DSG-000040389, the XCS Polycapillary. It includes an ePix detector stage, a liquid jet stack, three crystal stacks, and three polycapillary stacks with 28 total DOF. Slider controls support inspection of individual stages, while a phase-staggered animation actuates the entire assembly. This provides a practical way to inspect stage interactions.

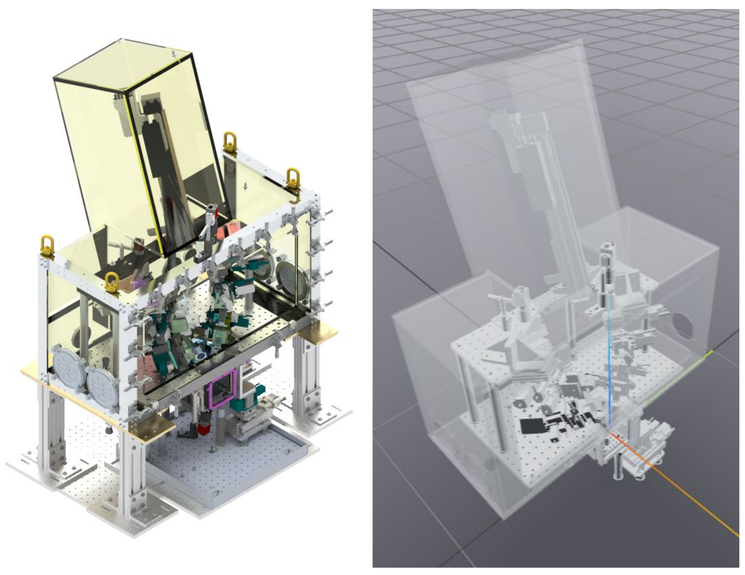
Collision analysis uses Drake proximity queries and CoACD convex decomposition. The demonstration model decomposes 215 components into 5,073 convex hulls, enabling meaningful clearance checks while preserving accurate geometry.

**One integrated model supports both motion visualization and interference analysis.** The viewer highlights offending parts and identifies the worst pairs. Close (yellow) and conflicting (red) members are displayed live as controls are manipulated.

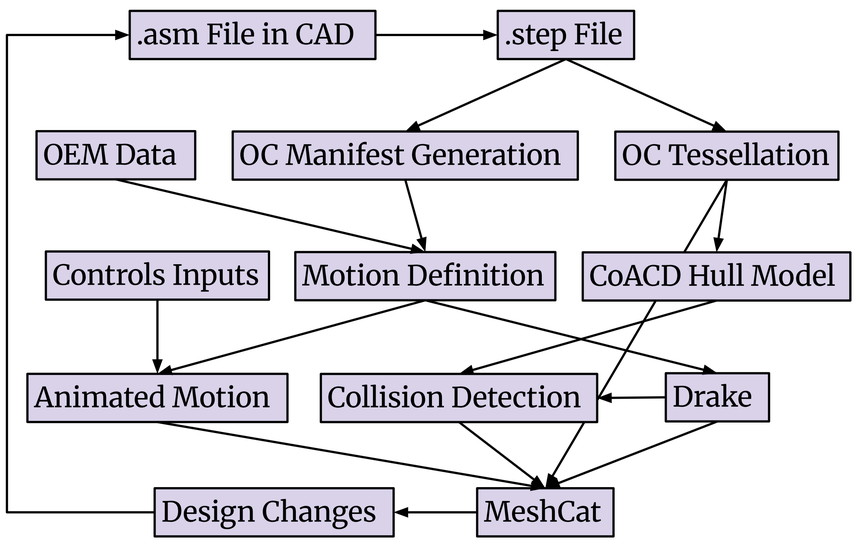
**Impact & Future Work**

Twin Lab shifts interference analysis earlier in the development process, when designs are easier to modify. Engineers can test candidate motion and flag interferences pre-manufacturing. The workflow also preserves motion definitions across CAD revisions, facilitating integration of rapid-iteration redesigns.

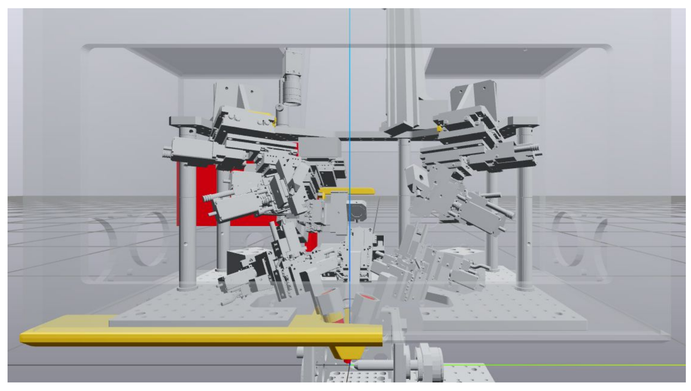
Future work will focus on two enhancements: live display of motor positions during experiments and path planning to optimize stage movements. Twin Lab is well-suited for both capabilities, leveraging its Python foundation and Drake's integrated path planning algorithms. Beyond the present application, the broader impact lies in extending these capabilities to other complex stage systems in LCLS where precise interference control is critical.



From static CAD to motion-aware analysis. A STEP assembly provides geometry and hierarchy, while a user-reviewed configuration adds the mechanical meaning required to simulate each movable stage.




Closed-loop digital twin workflow. Automated geometry extraction is combined with a reviewed configuration, allowing revised assemblies to be remapped without rebuilding their motion logic from scratch.



Interactive clearance feedback. The viewer evaluates each commanded pose live, highlights close or interfering components, and reports out the responsible part IDs for design review.

## ACKNOWLEDGEMENTS

This development work was completed thanks to the support of the LCLS Internship Program at SLAC National Accelerator Laboratory. I would like to thank my program mentor, Elon Goliger, as well as the XCS Team and the LCLS Engineering Division for their technical guidance, engineering resources, and feedback throughout the project. I'd also like to acknowledge the developers and maintainers of the several open-source Python libraries that made this project possible.

## REFERENCES & RESOURCES

1. Twin Lab Repository (linked at QR) github.com/slaclab/Twin-Lab
2. Drake https://drake.mit.edu/
3. MeshCat https://github.com/meshcat-dev/meshcat
4. Open CASCADE https://dev.opencascade.org/
5. CoACD https://github.com/SarahWeiii/CoACD