Robotics: Science and Systems 2026
Sydney, Australia, July 13–17, 2026

Viser: Imperative, Web-based 3D Visualization in Python

Brent Yi[1], Chung Min Kim[1], Justin Kerr[1], Gina Wu[1], Rebecca Feng[1], Anthony Zhang[1], Jonas Kulhanek[2,3], Hongsuk Choi[1], Yi Ma[1,4], Matthew Tancik[5], Angjoo Kanazawa[1]

[1]UC Berkeley [2]CTU in Prague [3]ETH Zurich [4]HKU [5]Luma AI

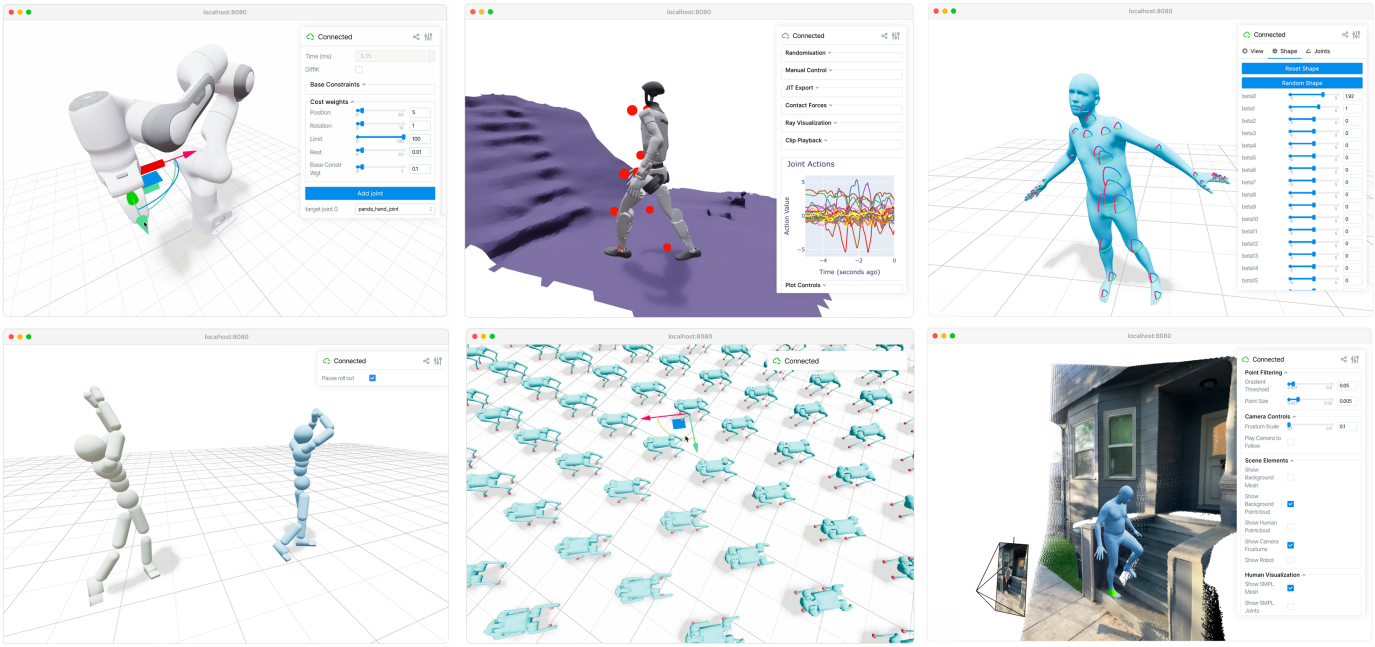

Fig. 1: **Visualization for robotics and computer vision.** Viser provides a web-based viewer, scene primitives, and GUI primitives for visualization. These can be composed for a broad set of applications. *From top-left:* (i) Interactive inverse kinematics in PyRoki [33]. (ii) Policy rollout visualization for reinforcement learning in VideoMimic [4]. (iii) Interactive SMPL [40] viewer with pose and shape controls. (iv) Humanoid control visualization from policies trained in IsaacGym [44]. (v) Batched rendering for parallel simulation in MuJoCo Playground [106]. (vi) Scene visualization from VideoMimic [4].

***Abstract*—We present Viser, a toolkit for 3D visualization in robotics and computer vision. Viser aims to bring easy and extensible 3D visualization to Python: we provide comprehensive 3D scene and 2D GUI primitives, which can be used independently with minimal setup or composed to build specialized interfaces. This paper details features and key design choices—an imperative-style API and a web-based viewer—which improve compatibility with modern programming patterns. We then discuss system architecture, adoption, and limitations.**

I. Introduction

Visual feedback enables fast, confident iteration in robotics and computer vision research. By letting researchers see, navigate, and interact with 3D data, visualization tools [1, 7, 9–11, 15, 18, 23, 41, 44, 47, 53, 59, 61, 65, 72, 75, 111] accelerate diagnosis of issues—consider incorrect coordinate conventions or self-colliding robot joints—while delivering immediate insight into both algorithmic behavior and data.

Choosing a visualization tool, however, is like many life choices. It requires weighing tradeoffs. Most existing tools fall into one of two categories, which each have distinct advantages. First, lightweight libraries can be imported and excel for simple visualization tasks [23, 47, 111]. They enable quick visualization with minimal setup, which is ideal for rapid debugging and prototyping. Meanwhile, domain-specific packages offer richer features for more specific settings. These tools are less general-purpose, but enable important workflow improvements: mobile robotics benefits from interfaces for setting 2D pose estimates [61], simulators benefit from contact and force visualization [53, 75], and radiance fields benefit from training control and rendering interfaces [74, 77]. In this work, we propose a system that aims to bridge these two categories of tools.

This paper presents Viser, an open-source toolkit for 3D

Fig. 2: **Real-time visualization for physical robots.** Viser enables live debugging of perception and control systems on physical robots. *Left:* A humanoid robot executing a learned locomotion policy. *Right:* Visualizing real-time state estimation and mapping in Viser. Viser's web-based architecture simplifies remote monitoring and debugging.

visualization in robotics and computer vision (Figure 1). Viser is designed to be as easy as possible for simple visualization tasks, while extending naturally to more specialized needs. This is achieved through a comprehensive set of 3D scene and 2D GUI primitives, which can be used either independently with minimal setup or composed to build more complex, task-specific interfaces. Design choices that improve usability include an imperative-style API and a web-based viewer, which make Viser easy to integrate with modern, Python-based programming patterns and workflows.

In this paper, we begin by describing the core features of Viser (Section II). We detail Viser's underlying design: its API (Section III) and layered system architecture (Section IV). We then present benchmarking and adoption statistics (Section V). We conclude with discussion (Section VI) and limitations (Section VII).

II. Features

Viser proposes a unified system for 3D visualization, which is built in three parts: a web-based viewer, a library of scene primitives, and GUI primitives. We begin by describing these features, which are designed to simplify visualization across both robotics and computer vision tasks.

A. Web-based Viewer

Viser automatically hosts a local visualization server when used, which provides a viewer that can be accessed from any modern web browser. This web-based approach has several advantages. *(1) Low setup effort:* a web-based viewer makes Viser simple to install and run across platforms, including on headless servers and mobile clients. *(2) Sharing:* Viser lets researchers embed offline visualizations into static webpages or share active sessions using simple URLs. Embedding examples can be found on webpages for [30, 83, 84, 101, 108]. *(3) Development:* web infrastructure accelerates development velocity. Viser benefits from mature libraries like React [50] and `three.js` for UI development and 3D rendering.

B. Scene Primitives

We implement a comprehensive range of 3D visualization features, which aim to be generally useful across applications:

Visualizing diverse 3D data. Users can display various data types—point clouds, meshes, images, Gaussian splats—along with primitives like coordinate frames, frustums, grids, splines, and line segments, all with simple function calls. Viser can directly load GLB and glTF formats, which enables integration with existing 3D assets. Batched rendering and level-of-detail optimizations maintain performance for large scenes.

Organizing complex scenes. Viser uses a hierarchical scene graph that simplifies coordinate transformations and kinematic relationships. This generalizes nested coordinate systems like robots and camera rigs.

Creating high-quality visuals. Viser's `three.js`-based rendering pipeline supports physically-based materials, environment maps, comprehensive lighting (ambient, directional, point, area, spot), and shadow mapping. These features enable high-quality visuals without specialized rendering software.

Handling real-time data streams. Viser automatically synchronizes state changes between Python and web clients. Updates are optimized to enable smooth visualization of dynamic data from neural network training, physics simulations, and robot sensors (Figure 2). The transport and rendering layers are architected around real-time data streams, which we benchmark in Section V-B.

Building interactive applications. To move beyond passive visualization, users can make objects clickable to trigger events, add transform gizmos to specify poses, and subscribe

```python
scene.add_point_cloud(
  "/points",
  points=np.array(...),
  colors=np.array(...),
)

scene.add_camera_frustum(
  "/camera",
  fov=np.pi / 2.0,
  aspect=h / w,
  wxyz=(qw, qx, qy, qz),
  position=(x, y, z),
)

# Add other camera frustums, meshes
```

(a) *Python API for scene control*

```python
next_btn = gui.add_button("Next
↪  Frame")
prev_btn = gui.add_button("Prev
↪  Frame")
playing_cb =
↪  gui.add_checkbox("Playing")

@next_btn.on_click
def _(_) -> None:
    ...
@prev_btn.on_click
def _(_) -> None:
    ...
@playing_cb.on_update
def _(_) -> None:
    ...

# Add other UI elements
```

(b) *Python API for GUI configuration*

Fig. 3: **Populating Viser.** Viser is used to display inputs and outputs from a multiview reconstruction pipeline [52]. We show (a) example code for adding 3D scene elements, and (b) example code for populating the graphical interface.

to general scene pointer events. A camera API enables programmatic access to and control over viewpoint information. These primitives directly support common robotics workflows: transform gizmos can specify 6D inverse kinematics targets (Figure 1), clickable scene objects can drive interactive grasp or contact selection, and the camera API supports human-in-the-loop teleoperation and data collection.

C. GUI Primitives

Viser also provides 2D GUI features. The 2D GUI enables domain-specific controls and custom information displays:

Capturing user input. Users can create standard interface elements—buttons, sliders, checkboxes, text inputs, dropdowns—with single function calls like `gui.add_button()`. Specialized controls like color pickers, vector inputs, upload buttons, and progress bars can be used for more advanced applications.

```python
# Create box.
box = scene.add_box("/box")

# Update box property.
box.color = (255, 0, 0)

# Attach click callback.
@box.on_click
def _(event): print("Box clicked")

# Remove box.
box.remove()
```

(a) 3D Scene API

```python
# Create button.
button = gui.add_button("Click")

# Update button property.
button.color = (255, 0, 0)

# Attach click callback.
@button.on_click
def _(event): print("Button clicked")

# Remove button.
button.remove()
```

(b) 2D GUI API

Fig. 4: **Imperative, side effect-driven component lifecycle management.** The same programming patterns apply to both 3D scene and 2D GUI primitives: creating objects with single function calls, updating properties through handle attributes, registering event callbacks with Python decorators, and explicit removal when no longer needed.

Displaying rich information. Beyond inputs, users can render text, markdown, and HTML content inline with visualizations. 2D images can be both displayed and streamed, while Plotly [24] and uPlot [70] integration enables 2D plotting. A notification system provides status updates and user feedback.

Organizing complex interfaces. As applications grow, folders and tab groups help structure complicated control panels, while modal dialogs handle focused interactions. GUI containers can also be placed directly in 3D scenes, creating spatially integrated inputs. This enables specialized use cases like per-object control panels and annotation tooling.

III. API Design

Viser proposes a Python API that is centered on ease-of-use and flexibility. We achieve this by designing Viser's scene and GUI features to integrate seamlessly with standard Python programming patterns: arbitrary scripts, notebooks [34], REPLs [56, 78], and step-through debuggers. To achieve this, we

```python
# Create elements.
slider = gui.add_slider("Value", 0, 100)
out = gui.add_text("0")

# Direct state update on event.
@slider.on_update
def _(_):
    out.value = str(slider.value * 2)

# Continue with other Python code.
while True:
    time.sleep(1.0)
```

(a) Imperative input/output relationships

```python
# Explicit state management.
counter = 0
label = gui.add_text(f"Count: {counter}")
button = gui.add_button("Increment")

# Update state on click.
@button.on_click
def _(_):
    global counter
    counter += 1
    label.value = f"Count: {counter}"

# Continue with other Python code.
while True:
    time.sleep(1.0)
```

(b) Imperative state management

```python
# Define UI as input-output transform.
def double_value(x):
    return str(x * 2)

demo = gr.Interface(
    fn=double_value,
    inputs=gr.Slider(0, 100),
    outputs=gr.Textbox(),
)

# Framework takes control.
demo.launch()
```

(c) Declarative input/output relationships

```python
# Framework manages state.
def increment(count):
    return (
        count + 1,
        f"Count: {count + 1}",
    )

demo = gr.Interface(
    fn=increment,
    inputs=[gr.State(0), gr.Button()],
    outputs=[gr.State(), gr.Textbox()],
)

# Framework takes control.
demo.launch()
```

(d) Declarative state management

Fig. 5: **Comparing imperative and declarative abstractions.** Viser's imperative-style API provides low-level control through side effects, which are replaced with higher-level abstractions in declarative APIs like Gradio [1], the dominant library for building user interfaces for machine learning. Viser's imperative-style API prioritizes user control of program flow, which simplifies integration into complex programs.

propose an *imperative-style* API characterized by explicit side effects in user-controlled program flow.

Primitive lifecycle. Scene and GUI elements in Viser are created through single function calls, which return handles for subsequent interaction (Figure 3, 4). These handles encapsulate both lifecycle control and automatic state synchronization: users manage when primitives are removed, and can use assignment-style syntax to update properties. Property assignments immediately update visualizations, without manual synchronization or render calls. Similarly, user interactions in the browser—clicks, slider movements, text input—automatically flow back to Python, where they are made available for both polling-style reads and interrupt-style callbacks. This design combines explicit control over object lifecycles with implicit handling of the logic required for real-time, bidirectional synchronization between Python and web clients.

Why imperative? Viser's imperative-style design deviates from the declarative patterns that are more common for layout and visualization in Python [1, 24, 67, 68, 71, 79]. In declarative programming, users pass objectives like visual properties, data relationships, and scene layouts to a runtime, which is then responsible for the details of program flow. This simplifies many use cases: Gradio [1], for example, excels at managing independent state for multiple parallel clients. This comes at a cost: handing off control over program flow can limit integration flexibility. In contrast, Viser's imperative programming model is designed to be easier to integrate into interactive Python programming patterns. We provide a concrete comparison in Figure 5.

IV. System Architecture

To support the imperative-style API described in Section III, Viser employs an architecture that separates user-facing interfaces from the underlying web-based implementation (Figure 6). This design shields users from networking and state synchronization complexities. It is implemented in four layers.

Core API. Viser's core API provides high-level methods like `scene.add_mesh()` and `gui.add_button()` for

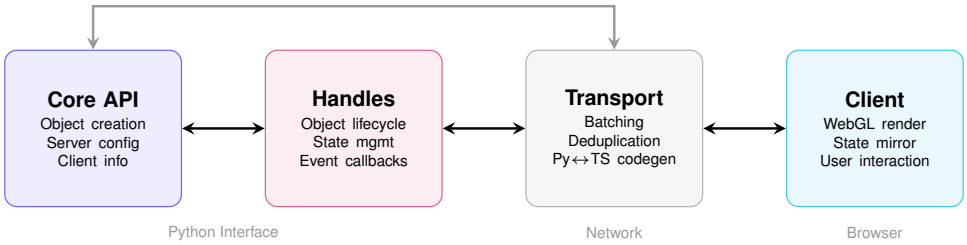

Fig. 6: **Implementation.** Viser is implemented in four layers. Users interact programmatically with a Python interface (Core API, Handles), which communicate (Transport) with a web-based frontend (Client).

creating objects, as well as methods for configuring servers and accessing client information. These methods can be called either globally to share objects between all clients or on a per-client basis for more targeted updates.

Handles. Each primitive creation method call returns a handle that maintains the created element's lifecycle and state. Handles provide property setters and getters, where operations like `box.color = (255, 0, 0)` immediately update the visualization. They also offer event callback registration for building interactive applications.

Transport. Commands from Viser's Python-based API are sent to clients through Viser's transport layer, which manages communication between Python and web clients via WebSocket connections. The transport layer automatically buffers, batches, and deduplicates state updates in a thread before serializing them via msgpack [16]. Python message definitions are used to generate TypeScript declarations for static verification, ensuring type safety across the Python-JavaScript API boundary. Deduplicated messages are efficiently persisted to maintain state consistency for new clients. Latency-aware buffering enables smooth framerates, even on unreliable network connections.

Client. Clients receive updates and render them in a web browser. Viser clients maintain a mirror of server-side state and capture user interactions—clicks, camera motion, and GUI input changes—which are relayed back to the Python server through the same transport layer. Performance optimizations include threading via WebWorkers, Gaussian splat sorting compiled via WebAssembly, and shadows rendered with cascaded shadow maps [12].

V. Evaluation and Adoption

We evaluate Viser along complementary axes: (i) a feature comparison against widely used alternatives, (ii) quantitative benchmarks of its API and transport layers, and (iii) adoption statistics.

A. Feature Comparison

Table I compares Viser's feature set against three widely used alternatives—MeshCat [11], Rerun [65], and Foxglove [15]—that target overlapping use cases in robotics visualization. All four tools share support for web-based visualization, scene primitives like points and meshes, and Jupyter embedding. Viser is differentiated by interactive controls that are designed to be composable from Python: customizable GUI inputs, 3D transform gizmos, and click/pointer callbacks. These primitives support the interactive applications described in Section III, such as setting inverse kinematics targets and controlling simulator state. Limitations are discussed in Section VII.

	Viser	MeshCat	Rerun	Foxglove
Robot visualization				
URDF	✓		✓	✓
Batched / instanced robots	✓			
Coordinate frames	✓	✓	✓	✓
Scene primitives				
Points, meshes, images	✓	✓	✓	✓
Gaussian splats	✓			
Camera frustums	✓		✓	✓
glTF / GLB models	✓		✓	✓
Lighting, environment maps	✓			
Interactive controls				
Customizable GUI inputs	✓			
3D transform gizmos	✓			
Click / pointer callbacks	✓			
Architecture				
Web-based viewer	✓	✓	✓	✓
Jupyter embedding	✓	✓	✓	✓
Open-source	✓	✓	✓	

TABLE I: **Feature comparison.** Viser's imperative API is uniquely designed around interactive visualization for robotics.

B. Benchmarks

We benchmark Viser against open-source alternatives—MeshCat [11] and Rerun [65]—on three common robotics visualization workloads: point cloud streaming, triangle mesh streaming, and 2D image streaming. We use the Playwright browser automation framework to measure end-to-end latency from a Python API call to the corresponding pixel change in the viewer. Figure 7 reports Python-side API call overhead (top row) and end-to-end render latency (bottom row) across these workloads. Viser's Python-native API and thin transport layer (Section IV) achieve latencies that are competitive with other tools. Viser scales particularly well on large point clouds, with roughly 10× lower API overhead than MeshCat at 1M points. These characteristics support the real-time use cases shown in Figures 1, 2, and 9.

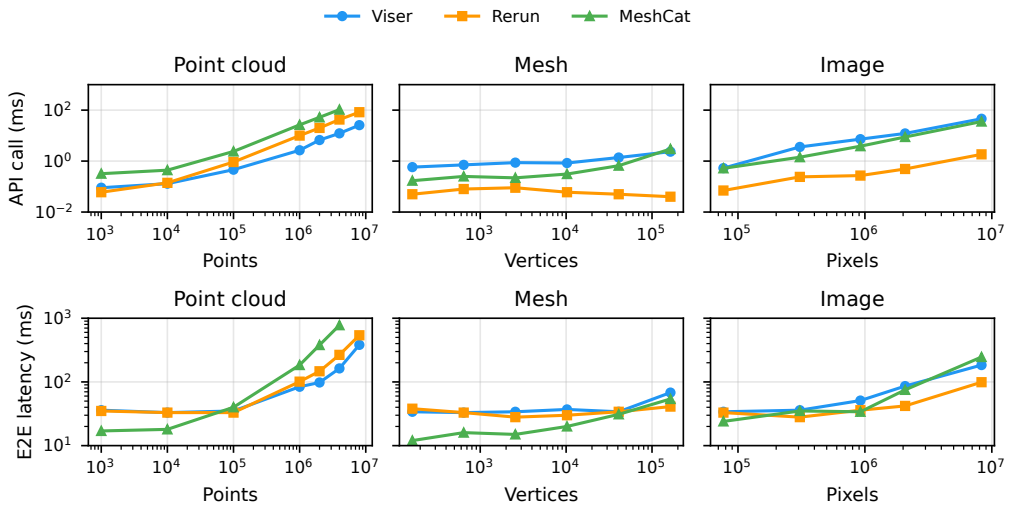

Fig. 7: **Visualization benchmarks.** Across three robotics streaming workloads—point cloud, triangle mesh, and 2D image—we measure (top row) Python-side API call overhead and (bottom row) end-to-end latency from Python call to visible pixel change in the browser, measured via Playwright with a requestAnimationFrame pixel watcher. Lower is better. Viser is competitive with or faster than the open-source alternatives on every workload, with particularly large gains in API throughput at high point counts.

C. Adoption

Evaluating software systems poses distinct challenges compared to evaluating algorithmic contributions. While user studies can assess learnability and benchmarks can measure performance, these approaches capture only narrow aspects of a tool's utility. Adoption captures factors difficult to isolate in controlled studies: compatibility with diverse workflows, reliability across edge cases, documentation quality, and long-term maintainability.

We measure Viser's adoption using (i) public installation statistics and (ii) by manually indexing research code releases that include Viser as a dependency. Viser has grown organically since being first made available in 2023: it has since been installed over 4.4 million times, and received 2,500+ GitHub stars. Published work that uses Viser spans neural rendering [26, 35, 38, 39, 64, 69, 72, 76, 82, 87, 88, 92, 94, 99, 100], generative models [21, 25, 39, 42, 45, 89, 110], scene understanding [3, 6, 17, 19, 26, 32, 39, 52], reconstruction and tracking [3, 14, 20, 22, 25, 28, 43, 80, 81, 83–85, 91, 93, 95–98, 101, 108, 109], and robotics [2, 4, 8, 13, 27, 30, 31, 33, 36, 37, 48, 49, 54, 57, 58, 60, 62, 63, 86, 90, 102–105, 107]. We highlight two projects, Nerfstudio [72] and mjlab [107], that make heavy use of Viser's extensible design in Figures 8 and 9, as well as 3D vision applications in Figure 10. We view the quantity, breadth, and diversity of projects that have relied on Viser as validation of its design choices.

VI. Conclusion

We presented the design and implementation of Viser, a toolkit for 3D visualization. Viser provides 3D scene and 2D GUI visualization primitives that aim to be easy to use independently, but are powerful when composed. Its adoption in robotics and computer vision (Section V-C) suggests this design addresses a real need in the research community.

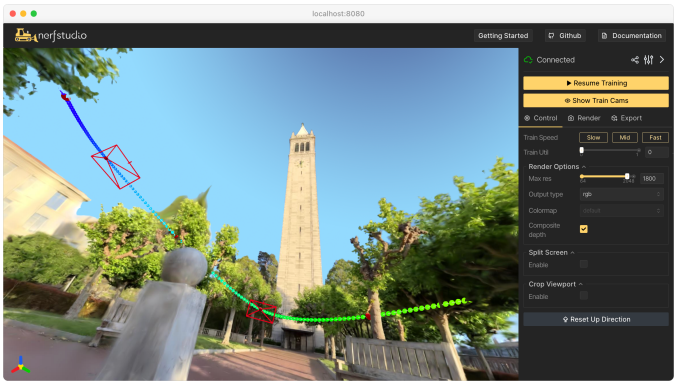

Fig. 8: **Nerfstudio [72].** Nerfstudio uses a Viser-based viewer for real-time visualization of neural radiance fields [51] and 3D Gaussian splats [29], training visualization, and camera path creation.

Viser is in part inspired by the history and impact of software contributions in machine learning and robotics research. This includes PyTorch [55] and einops [66] in machine learning, as well as MuJoCo [75], SymForce [46], and MuJoCo Playground [106] in robotics. We hope that Viser can play a similar role as these tools in assisting research efforts in computer vision, robotics, and related fields.

VII. Limitations

While Viser's design and implementation provides advantages in simplicity and flexibility, achieving these benefits requires tradeoffs. Current limitations include:

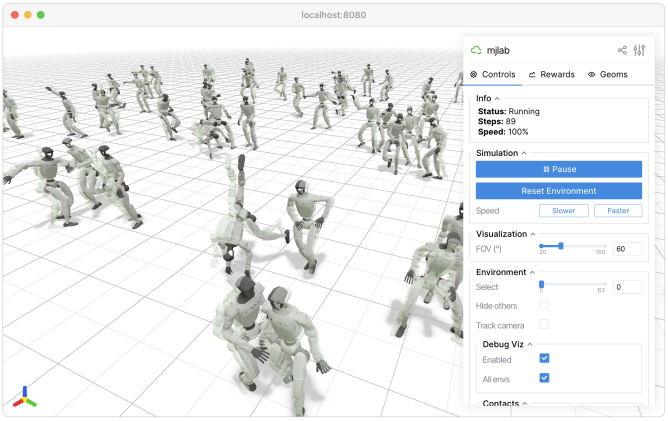

Fig. 9: **Real-time visualization for MuJoCo simulation.** The Viser visualizer developed by the mjlab [107] team enables web-based visualization for policy rollouts and reward curves.

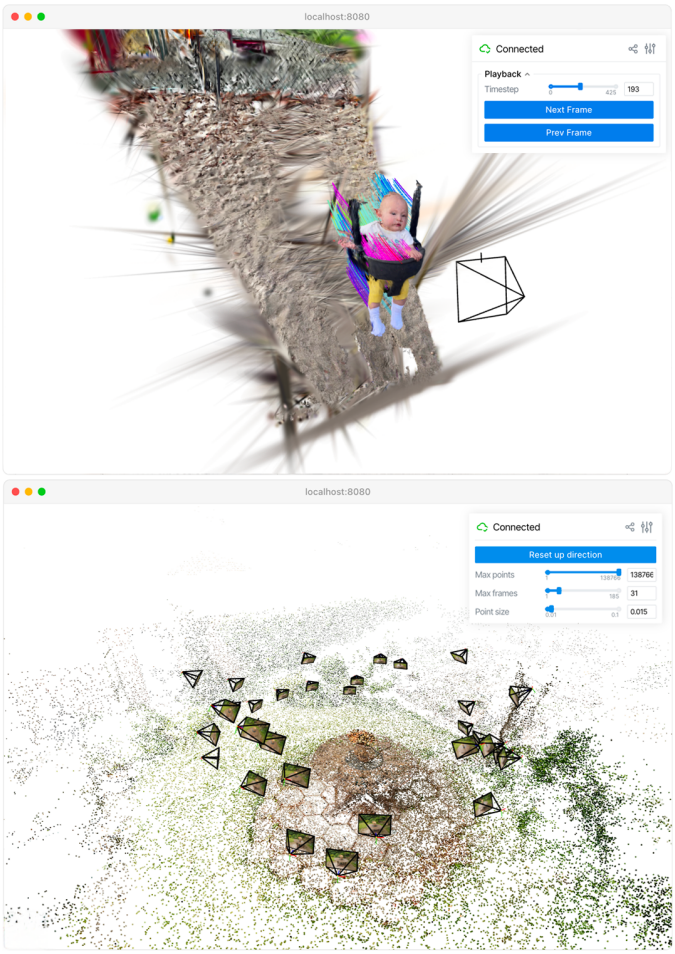

Fig. 10: **Visualization for 3D vision tasks.** We visualize 3D data from Shape of Motion [83] and Mip-NeRF 360 [5].

- **WebSocket transfer.** Viser's web-based client receives all visualized assets through a WebSocket connection. This introduces overhead, which is exacerbated by the fact that program state is managed by the Python server: updates in response to user interaction require round-trip messages and cannot be exported to static webpages. Websocket-based message passing also prevents CPU-to-GPU transfer optimizations like the ones implemented in Teed et al. [73].
- **Stateful API.** Viser's API is inherently stateful. This is an intentional design choice, but can also result in more duplicated or error-prone state management than declarative [1] or immediate-mode [9] visualization APIs.
- **Python.** Viser is designed for Python users. We do not provide bindings for languages like C++ or Rust, which are common in performance-critical systems.
- **Single process.** Viser launches per-script visualization servers. This may require effort to adapt to systems with concurrent processes, which are common in robotics [61].
- **Timestamps.** Viser does not assume temporal structure or provide standard ways to timestamp data. This can increase boilerplate for visualizing sequence data.
- **Serialization.** Many tools can load data from formats like rosbag [61], MCAP [15], and rrd [65]. Serialized data is useful for logging and offline playback, which Viser has limited built-in features for.

References

[1] Abubakar Abid, Ali Abdalla, Ali Abid, Dawood Khan, Abdulrahman Alfozan, and James Zou. Gradio: Hassle-free sharing and testing of ml models in the wild. *arXiv preprint arXiv:1906.02569*, 2019.

[2] Simeon Adebola, Chung Min Kim, Justin Kerr, Shuangyu Xie, Prithvi Akella, Jose Luis Susa Rincon, Eugen Solowjow, and Ken Goldberg. BotanyBot: Digital twin monitoring of occluded and underleaf plant structures with Gaussian splats. *arXiv preprint arXiv:2510.17783*, 2025.

[3] Simeon Adebola, Shuangyu Xie, Chung Min Kim, Justin Kerr, Bart M. van Marrewijk, Mieke van Vlaardingen, Tim van Daalen, Robert van Loo, Jose-Luis Susa Rincon, Eugen Solowjow, Rick van de Zedde, and Ken Goldberg. Growsplat: Constructing temporal digital twins of plants with gaussian splats, 2025.

[4] Arthur Allshire, Hongsuk Choi, Junyi Zhang, David McAllister, Anthony Zhang, Chung Min Kim, Trevor Darrell, Pieter Abbeel, Jitendra Malik, and Angjoo Kanazawa. Videomimic: Visual imitation enables contextual humanoid control, 2025.

[5] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[6] Thales Berriel and Javier Civera. Feature splatting for better novel view synthesis with low overlap, 2024.

[7] Sebastien Bonopera, Peter Hedman, Jerome Esnault, Siddhant Prakash, Simon Rodriguez, Theo Thonat,

Mehdi Benadel, Gaurav Chaurasia, Julien Philip, and George Drettakis. SIBR: A system for image-based rendering. https://sibr.gitlabpages.inria.fr/, 2020. Accessed 2025-05-05.

[8] Jan Brüdigam, Ali-Adeeb Abbas, Maks Sorokin, Kuan Fang, Brandon Hung, Maya Guru, Stefan Georg Sosnowski, Jiuguang Wang, Sandra Hirche, and Simon Le Cleac'h. Jacta: A versatile planner for learning dexterous and whole-body manipulation. In *Proceedings of the 8th Conference on Robot Learning (CoRL)*, volume 270 of *Proceedings of Machine Learning Research*, Munich, Germany, 2024. PMLR. URL https://openreview.net/forum?id=vobaOY0qDl. Preprint available at arXiv:2408.01258.

[9] Omar Cornut. Dear imgui: Bloat-free immediate mode graphical user interface for c++ with minimal dependencies. https://github.com/ocornut/imgui, 2014.

[10] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2023.

[11] Robin Deits, Jeremy Nimmer, Russ Tedrake, and the MeshCat Development Team. MeshCat: Web-based 3-d visualization, 2021. URL https://github.com/meshcat-dev/meshcat. MIT License. Accessed 21 May 2025.

[12] Rouslan Dimitrov. Cascaded shadow maps. Technical white paper, NVIDIA Corporation, August 2007. URL https://developer.download.nvidia.com/SDK/10.5/opengl/src/cascaded_shadow_maps/doc/cascaded_shadow_maps.pdf. Version 1.1, NVIDIA OpenGL SDK.

[13] Alejandro Escontrela, Justin Kerr, Arthur Allshire, Jonas Frey, Rocky Duan, Carmelo Sferrazza, and Pieter Abbeel. GaussGym: An open-source real-to-sim framework for learning locomotion from pixels. *arXiv preprint arXiv:2510.15352*, 2025.

[14] Haiwen Feng, Junyi Zhang, Qianqian Wang, Yufei Ye, Pengcheng Yu, Michael J. Black, Trevor Darrell, and Angjoo Kanazawa. St4rtrack: Simultaneous 4d reconstruction and tracking in the world, 2025.

[15] Foxglove Technologies Inc. Foxglove studio, 2025. URL https://foxglove.dev. A visualization and observability platform for robotics developers.

[16] Sadayuki Furuhashi. Messagepack: It's like json, but fast and small, 2008. URL https://msgpack.org/.

[17] Qiao Gu, Zhaoyang Lv, Duncan Frost, Simon Green, Julian Straub, and Chris Sweeney. Egolifter: Open-world 3d segmentation for egocentric perception, 2024.

[18] Vladimir Guzov, Ilya A. Petrov, and Gerard Pons-Moll. Blendify – python rendering framework for blender. *arXiv preprint arXiv:2410.17858*, 2024.

[19] Siming He, Zachary Osman, Fernando Cladera, Dexter Ong, Nitant Rai, Patrick Corey Green, Vijay Kumar, and Pratik Chaudhari. Estimating the diameter at breast height of trees in a forest with a single 360 camera. *arXiv preprint arXiv:2505.03093*, 2025.

[20] Siming He, Zachary Osman, Fernando Cladera, Dexter Ong, Nitant Rai, Patrick Corey Green, Vijay Kumar, and Pratik Chaudhari. Estimating the diameter at breast height of trees in a forest with a single 360 camera, 2025. URL https://arxiv.org/abs/2505.03093.

[21] Wenbo Hu, Xiangjun Gao, Xiaoyu Li, Sijie Zhao, Xiaodong Cun, Yong Zhang, Long Quan, and Ying Shan. Depthcrafter: Generating consistent long depth sequences for open-world videos, 2024.

[22] Finlay G. C. Hudson, James A. D. Gardner, and William A. P. Smith. TAPVid-360: Tracking any point in 360 from narrow field of view video. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[23] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[24] Plotly Technologies Inc. Collaborative data science, 2015. URL https://plot.ly.

[25] Zeren Jiang, Chuanxia Zheng, Iro Laina, Diane Larlus, and Andrea Vedaldi. Geo4d: Leveraging video generators for geometric 4d scene reconstruction, 2025.

[26] Xin Jin, Pengyi Jiao, Zheng-Peng Duan, Xingchao Yang, Chun-Le Guo, Bo Ren, and Chongyi Li. Lighting every darkness with 3dgs: Fast training and real-time rendering for hdr view synthesis, 2024.

[27] Dvij Kalaria, Sudarshan S. Harithas, Pushkal Katara, Sangkyung Kwak, Sarthak Bhagat, S. Shankar Sastry, Srinath Sridhar, Sai Vemprala, Ashish Kapoor, and Jonathan Chung-Kuan Huang. DreamControl: Human-inspired whole-body humanoid control for scene interaction via guided diffusion. *arXiv preprint arXiv:2509.14353*, 2025.

[28] Gyeongjin Kang, Seungkwon Yang, Seungtae Nam, Younggeun Lee, Jungwoo Kim, and Eunbyung Park. Multi-view pyramid transformer: Look coarser to see broader. *arXiv preprint arXiv:2512.07806*, 2025.

[29] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (TOG)*, 42(4), 2023.

[30] Justin Kerr, Chung Min Kim, Mingxuan Wu, Brent Yi, Qianqian Wang, Ken Goldberg, and Angjoo Kanazawa. Robot see robot do: Imitating articulated object manipulation with monocular 4d reconstruction. In *CoRL*, 2024.

[31] Justin Kerr, Kush Hari, Ethan Weber, Chung Min Kim, Brent Yi, Tyler Bonnen, Ken Goldberg, and Angjoo Kanazawa. Eye, robot: Learning to look to act with a BC-RL perception-action loop. In *9th Annual Conference on Robot Learning*, 2025.

[32] Chung Min Kim, Mingxuan Wu, Justin Kerr, Ken Goldberg, Matthew Tancik, and Angjoo Kanazawa. GARField: Group anything with radiance fields. In *CVPR*, pages 21530–21539, 2024.

[33] Chung Min Kim, Brent Yi, Hongsuk Choi, Yi Ma, Ken Goldberg, and Angjoo Kanazawa. Pyroki: A modular

toolkit for robot kinematic optimization, 2025.

[34] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter notebooks–a publishing format for reproducible computational workflows. In *Positioning and power in academic publishing: Players, agents and agendas*, pages 87–90. IOS press, 2016.

[35] Jonas Kulhanek and Torsten Sattler. NerfBaselines: Consistent and reproducible evaluation of novel view synthesis methods, 2024.

[36] Albert H. Li, Brandon Hung, Aaron D. Ames, Jiuguang Wang, Simon Le Cleac'h, and Preston Culbertson. Judo: A user-friendly open-source package for sampling-based model predictive control. In *Proceedings of the Workshop on Fast Motion Planning and Control in the Era of Parallelism at Robotics: Science and Systems (RSS)*, 2025.

[37] Hongyu Li, Lingfeng Sun, Yafei Hu, Duy Ta, Jennifer Barry, George Konidaris, and Jiahui Fu. NovaFlow: Zero-shot manipulation via actionable flow from generated videos. *arXiv preprint arXiv:2510.08568*, 2025.

[38] Ruilong Li, Brent Yi, Junchen Liu, Hang Gao, Yi Ma, and Angjoo Kanazawa. Cameras as relative positional encoding. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[39] Kunhao Liu, Ling Shao, and Shijian Lu. Novel view extrapolation with video diffusion priors, 2024.

[40] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. Smpl: A skinned multi-person linear model. volume 34, pages 1–16. ACM, 2015.

[41] Steven Lovegrove. Pangolin: A lightweight portable rapid development library for managing opengl display and interaction. https://github.com/stevenlovegrove/Pangolin, 2013.

[42] Yifan Lu, Xuanchi Ren, Jiawei Yang, Tianchang Shen, Zhangjie Wu, Jun Gao, Yue Wang, Siheng Chen, Mike Chen, Sanja Fidler, and Jiahui Huang. Infinicube: Unbounded and controllable dynamic 3d driving scene generation with world-guided video models, 2024.

[43] Dominic Maggio, Hyungtae Lim, and Luca Carlone. VGGT-SLAM: Dense rgb slam optimized on the SL(4) manifold, 2025. URL https://arxiv.org/abs/2505.12549.

[44] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[45] Vongani H. Maluleke, Kie Horiuchi, Lea Wilken, Evonne Ng, Jitendra Malik, and Angjoo Kanazawa. Diffusion forcing for multi-agent interaction sequence modeling. *arXiv preprint arXiv:2512.17900*, 2025.

[46] Hayk Martiros, Aaron Miller, Nathan Bucki, Bradley Solliday, Ryan Kennedy, Jack Zhu, Tung Dang, Dominic Pattison, Harrison Zheng, Teo Tomic, Peter Henry, Gareth Cross, Josiah VanderMey, Alvin Sun, Samuel Wang, and Kristen Holtz. SymForce: Symbolic Computation and Code Generation for Robotics. In *Proceedings of Robotics: Science and Systems*, New York City, NY, USA, June 2022. doi: 10.15607/RSS.2022.XVIII.041.

[47] Matthew Matl. pyrender. https://github.com/mmatl/pyrender, 2021. URL https://github.com/mmatl/pyrender. Version 0.1.45.

[48] David McAllister, Songwei Ge, Brent Yi, Chung Min Kim, Ethan Weber, Hongsuk Choi, Haiwen Feng, and Angjoo Kanazawa. Flow matching policy gradients. *arXiv preprint arXiv:2507.21053*, 2025.

[49] Raphael Memmesheimer, Jan Nogga, Bastian Pätzold, Evgenii Kruzhkov, Simon Bultmann, Michael Schreiber, Jonas Bode, Bertan Karacora, Juhui Park, Alena Savinykh, and Sven Behnke. Robocup@home 2024 opl winner nimbro: Anthropomorphic service robots using foundation models for perception and planning, 2024. URL https://arxiv.org/abs/2412.14989.

[50] Meta Platforms, Inc. React: A javascript library for building user interfaces. https://react.dev, 2013. Accessed 2025-05-05.

[51] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 405–421, 2020.

[52] Lea Müller, Hongsuk Choi, Anthony Zhang, Brent Yi, Jitendra Malik, and Angjoo Kanazawa. Reconstructing people, places, and cameras, 2025.

[53] NVIDIA Corporation. Nvidia isaac sim. https://developer.nvidia.com/isaac-sim, 2024. Version 2024.1.0, accessed May 2025.

[54] Chaoyi Pan, Changhao Wang, Haozhi Qi, Zixi Liu, Homanga Bharadhwaj, Akash Sharma, Tingfan Wu, Guanya Shi, Jitendra Malik, and Francois Hogan. SPIDER: Scalable physics-informed dexterous retargeting. *arXiv preprint arXiv:2511.09484*, 2025.

[55] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pages 8024–8035, 2019.

[56] Fernando Pérez and Brian E. Granger. IPython: A system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007. ISSN 1521-9615. doi: 10.1109/MCSE.2007.53. URL https://ipython.org.

[57] Haozhi Qi, Brent Yi, Sudharshan Suresh, Mike Lambeta, Yi Ma, Roberto Calandra, and Jitendra Malik. General in-hand object rotation with vision and touch, 2023.

[58] Haozhi Qi, Brent Yi, Mike Lambeta, Yi Ma, Roberto Calandra, and Jitendra Malik. From simple to complex skills: The case of in-hand object reorientation, 2025.

[59] Yuzhe Qin, Wei Yang, Binghao Huang, Karl Van Wyk, Hao Su, Xiaolong Wang, Yu-Wei Chao, and Dieter Fox. Anyteleop: A general vision-based dexterous robot arm-hand teleoperation system. In *Robotics: Science and Systems*, 2023.

[60] Tianshuang Qiu, Zehan Ma, Karim El-Refai, Hiya Shah, Chung Min Kim, Justin Kerr, and Ken Goldberg. OmniScan: Creating visually-accurate digital twin object models using a bimanual robot with handover and Gaussian splat merging. *arXiv preprint arXiv:2508.00354*, 2025.

[61] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[62] Adam Rashid, Satvik Sharma, Chung Min Kim, Justin Kerr, Lawrence Chen, Angjoo Kanazawa, and Ken Goldberg. LERF-TOGO: Language embedded radiance fields for zero-shot task-oriented grasping. In *CoRL*, 2023.

[63] Davis Rempe, Mathis Petrovich, Ye Yuan, Haotian Zhang, Xue Bin Peng, Yifeng Jiang, Tingwu Wang, Umar Iqbal, David Minor, Michael de Ruyter, et al. Kimodo: Scaling controllable human motion generation. *arXiv preprint arXiv:2603.15546*, 2026.

[64] Xuanchi Ren, Yifan Lu, Hanxue Liang, Zhangjie Wu, Huan Ling, Mike Chen, Sanja Fidler, Francis Williams, and Jiahui Huang. Scube: Instant large-scale scene reconstruction using voxsplats, 2024.

[65] Rerun Team. Rerun: A visualization sdk for multimodal data, 2025. URL https://www.rerun.io. Available from https://www.rerun.io/ and https://github.com/rerun-io/rerun.

[66] Alex Rogozhnikov. Einops: Clear and reliable tensor manipulations with einstein-like notation. In *International Conference on Learning Representations*, 2022.

[67] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. Vega-Lite: A grammar of interactive graphics. *IEEE Transactions on Visualization & Computer Graphics*, 2017. doi: 10.1109/TVCG.2016.2599030. URL http://vis.csail.mit.edu/pubs/vega-lite.

[68] Falko Schindler and Rodja Trappe. NiceGUI: Web-based user interfaces with python. the nice way. URL https://doi.org/10.5281/zenodo.15346216.

[69] Saptarshi Neil Sinha, Julius Kühn, Holger Graf, and Michael Weinmann. Spectralsplatsviewer: An interactive web-based tool for visualizing cross-spectral gaussian splats. In *Web3D '24*, 2024.

[70] Leon Sorokin. uPlot: A small, fast charting library for time-series data. URL https://github.com/leeoniya/uPlot. Accessed 2025-06-24.

[71] Streamlit Inc. Streamlit: An app framework for machine learning and data science. https://streamlit.io, 2020. Accessed 2025-05-05.

[72] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David McAllister, and Angjoo Kanazawa. Nerfstudio: A modular framework for neural radiance field development. *arXiv preprint arXiv:2302.04264*, 2023.

[73] Zachary Teed, Lahav Lipson, and Jia Deng. Deep patch visual odometry. *Advances in Neural Information Processing Systems*, 36:39033–39051, 2023.

[74] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. doi: 10.1145/3528223.3530127. URL https://arxiv.org/abs/2201.05989.

[75] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 5026–5033. IEEE, 2012.

[76] Adam Tonderski, Carl Lindström, Georg Hess, William Ljungbergh, Lennart Svensson, and Christoffer Petersson. Neurad: Neural rendering for autonomous driving. *arXiv preprint arXiv:2311.15260*, 2023.

[77] Towaki Takikawa, Or Perel, Clement Fuji Tsang, Charles Loop, Joey Litalien, Jonathan Tremblay, Sanja Fidler, and Maria Shugrina. Kaolin wisp: A pytorch library and engine for neural fields research. https://github.com/NVIDIAGameWorks/kaolin-wisp, 2022.

[78] Guido Van Rossum and Fred L Drake Jr. *Python tutorial*, volume 620. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[79] Jacob VanderPlas, Brian E. Granger, Jeffrey Heer, Dominik Moritz, Kanit Wongsuphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh, and Scott Sievert. Altair: Interactive statistical visualizations for python. *Journal of Open Source Software*, 3(32):1057, 2018. doi: 10.21105/joss.01057.

[80] Kjetil Vasstein, Christian Le, Simon Lervåg Breivik, Trygve Maukon Myhr, Annette Stahl, and Edmund Førland Brekke. Pygemini: Unified software development towards maritime autonomy systems, 2025. URL https://arxiv.org/abs/2506.06262.

[81] Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. Vggt: Visual geometry grounded transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5294–5306, June 2025.

[82] Junjie Wang, Jiemin Fang, Xiaopeng Zhang, Lingxi Xie, and Qi Tian. Gaussianeditor: Editing 3d gaussians

delicately with text instructions. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 20902–20911, 2024.

[83] Qianqian Wang, Vickie Ye, Hang Gao, Jake Austin, Zhengqi Li, and Angjoo Kanazawa. Shape of motion: 4d reconstruction from a single video, 2024.

[84] Qianqian Wang, Yifei Zhang, Aleksander Hołyński, Alexei A. Efros, and Angjoo Kanazawa. Cut3r: Continuous 3d perception model with persistent state, 2025.

[85] Yufu Wang, Yu Sun, Priyanka Patel, Kostas Daniilidis, Michael J. Black, and Muhammed Kocabas. Prompthmr: Promptable human mesh recovery. arXiv preprint arXiv:2504.06397, April 2025. submitted Apr 8, 2025.

[86] Zihan Wang, Jiashun Wang, Jeff Tan, Yiwen Zhao, Jessica Hodgins, Shubham Tulsiani, and Deva Ramanan. CRISP: Contact-guided Real2Sim from monocular video with planar scene primitives. *arXiv preprint arXiv:2512.14696*, 2025.

[87] Ethan Weber, Aleksander Hołyński, Varun Jampani, Saurabh Saxena, Noah Snavely, Abhishek Kar, and Angjoo Kanazawa. NeRFiller: Completing scenes via generative 3d inpainting. In *CVPR*, pages 20731–20741, 2024.

[88] Ethan Weber, Riley Peterlinz, Rohan Mathur, Frederik Warburg, Alexei A. Efros, and Angjoo Kanazawa. Toon3d: Seeing cartoons from a new perspective, 2024.

[89] Ethan Weber, Norman Müller, Yash Kant, Vasu Agrawal, Michael Zollhöfer, Angjoo Kanazawa, and Christian Richardt. Fillerbuster: Multi-view scene completion for casual captures, 2025. URL https://arxiv.org/abs/2502.05175.

[90] Zhenyu Wei, Zhixuan Xu, Jingxiang Guo, Yiwen Hou, Chongkai Gao, Zhehao Cai, Jiayu Luo, and Lin Shao. D(r,o) grasp: A unified representation of robot and object interaction for cross-embodiment dexterous grasping. *arXiv preprint arXiv:2410.01702*, 2024.

[91] Mingxuan Wu, Huang Huang, Justin Kerr, Chung Min Kim, Anthony Zhang, Brent Yi, and Angjoo Kanazawa. Predict-optimize-distill: A self-improving cycle for 4d object understanding, 2025.

[92] Minye Wu, Haizhao Dai, Kaixin Yao, Tinne Tuytelaars, and Jingyi Yu. BG-Triangle: Bézier gaussian triangle for 3d vectorization and rendering, 2025.

[93] Hanyuan Xiao, Yingshu Chen, Huajian Huang, Haolin Xiong, Jing Yang, Pratusha Prasad, and Yajie Zhao. Localized Gaussian splatting editing with contextual awareness. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2025.

[94] Congrong Xu, Justin Kerr, and Angjoo Kanazawa. Splatfacto-w: Wide-baseline factorized gaussian splatting, 2024.

[95] Gengshan Yang, Andrea Bajcsy, Shunsuke Saito, and Angjoo Kanazawa. Agent-to-sim: Learning interactive behavior models from casual longitudinal videos, 2024.

[96] Jianing Yang, Alexander Sax, Kevin J. Liang, Mikael Henaff, Hao Tang, Ang Cao, Joyce Chai, Franziska Meier, and Matt Feiszli. Fast3r: Towards 3d reconstruction of 1000+ images in one forward pass. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21924–21935, June 2025.

[97] Jiawei Yang, Jiahui Huang, Yuxiao Chen, Yan Wang, Boyi Li, Yurong You, Apoorva Sharma, Maximilian Igl, Peter Karkus, Danfei Xu, Boris Ivanovic, Yue Wang, and Marco Pavone. Storm: Spatio-temporal reconstruction model for large-scale outdoor scenes, 2024.

[98] David Yifan Yao, Albert J Zhai, and Shenlong Wang. Uni4d: Unifying visual foundation models for 4d modeling from a single video. *arXiv preprint arXiv:2503.21761*, 2025.

[99] Vickie Ye, Ruilong Li, Justin Kerr, Matias Turkulainen, Brent Yi, Zhuoyang Pan, Otto Seiskari, Jianbo Ye, Jeffrey Hu, Matthew Tancik, and Angjoo Kanazawa. gsplat: An open-source library for gaussian splatting, 2024.

[100] Brent Yi, Weijia Zeng, Sam Buchanan, and Yi Ma. Canonical factors for hybrid neural fields. In *ICCV*, pages 3414–3426, 2023.

[101] Brent Yi, Vickie Ye, Maya Zheng, Yunqi Li, Lea Müller, Georgios Pavlakos, Yi Ma, Jitendra Malik, and Angjoo Kanazawa. Estimating body and hand motion in an ego-sensed world. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 7072–7084, 2025.

[102] Justin Yu, Kush Hari, Kishore Srinivas, Karim El-Refai, Adam Rashid, Chung Min Kim, Justin Kerr, Richard Cheng, Muhammad Zubair Irshad, Ashwin Balakrishna, Thomas Kollar, and Ken Goldberg. Language-embedded gaussian splats (legs): Incrementally building room-scale representations with a mobile robot, 2024.

[103] Justin Yu, Letian Fu, Huang Huang, Karim El-Refai, Rares Ambrus, Richard Cheng, Muhammad Zubair Irshad, and Ken Goldberg. Real2render2real: Scaling robot data without dynamics simulation or robot hardware, 2025.

[104] Justin Yu, Kush Hari, Karim El-Refai, Arnav Dalal, Justin Kerr, Chung Min Kim, Richard Cheng, Muhammad Zubair Irshad, and Ken Goldberg. Persistent object gaussian splat (pogs) for tracking human and robot manipulation of irregularly-shaped objects, 2025.

[105] Justin Yu, Yide Shentu, Di Wu, Pieter Abbeel, Ken Goldberg, and Philipp Wu. EgoMI: Learning active vision and whole-body manipulation from egocentric human demonstrations. *arXiv preprint arXiv:2511.00153*, 2025.

[106] Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A Kahrs, et al. Mujoco playground. *arXiv preprint arXiv:2502.08844*, 2025.

[107] Kevin Zakka, Qiayuan Liao, Brent Yi, Louis Le Lay, Koushil Sreenath, and Pieter Abbeel. mjlab: A lightweight framework for gpu-accelerated robot learning, 2026. URL https://arxiv.org/abs/2601.22074.

[108] Junyi Zhang, Charles Herrmann, Junhwa Hur, Varun Jampani, Trevor Darrell, Forrester Cole, Deqing Sun, and Ming-Hsuan Yang. MonST3R: A simple approach for estimating geometry in the presence of motion, 2025.

[109] Xinyu Zhang, Haonan Chang, Yuhan Liu, and Abdeslam Boularias. Motion blender Gaussian splatting for dynamic scene reconstruction. *arXiv preprint arXiv:2503.09040*, 2025.

[110] Jensen Zhou, Hang Gao, Vikram Voleti, Aaryaman Vasishta, Chun-Han Yao, Mark Boss, Philip Torr, Christian Rupprecht, and Varun Jampani. Stable virtual camera: Generative view synthesis with diffusion models, 2025.

[111] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv preprint arXiv:1801.09847*, 2018.