A System for Near Real-Time Generation and Runtime Visualization of Generative 3D Models in XR Environments

SHI Yunze 25126879g
stcloudlake@ieee.org

***Abstract*—With the rapid advancement of generative artificial intelligence and extended reality (XR) technologies, integrating automatically generated 3D content into immersive environments has become increasingly promising yet technically challenging. In this work, we present an end-to-end system for near real-time generation and runtime visualization of generative 3D models in XR environments. The proposed system combines passthrough-based user input on an XR headset, a server-side orchestration layer, and a ComfyUI-based generative 3D pipeline to enable asynchronous, user-driven content creation. A key contribution of the system lies in its explicit runtime assembly strategy, which decouples geometry and texture generation to address material reconstruction and spatial consistency issues under XR runtime constraints. To ensure stable visualization and interaction, the system introduces a turntable anchor mechanism and automatic spatial normalization, enabling consistent presentation of generative assets with unpredictable internal structures. The system is implemented and evaluated on a standalone XR device, demonstrating reliable runtime visualization, responsive interaction, and robust workflow coordination. The results show that the proposed approach effectively bridges generative 3D pipelines and XR runtime environments, providing a practical foundation for immersive generative applications.**

***Index Terms*—Extended Reality (XR), Generative 3D Models, Runtime Asset Assembly, Asynchronous Communication, Passthrough Interaction, Near Real-Time Visualization.**

I. Introduction

A. Problem statement and motivation

Recent advances in generative artificial intelligence have enabled the automatic creation of 3D models from images, text, or multimodal inputs. Such generative 3D content has shown significant potential for immersive applications, including digital twins, virtual assets, and extended reality (XR) experiences. However, integrating automatically generated 3D models into XR systems remains a challenging problem, especially when generation and visualization are expected to occur in near real-time.

In practical XR applications, the generation and visualization pipeline is inherently distributed. User interactions are captured on XR head-mounted devices, computationally intensive generative models are executed on external servers, and the resulting assets must be transmitted back to the XR runtime for visualization. This multi-stage, multi-device communication process introduces non-trivial challenges in system coordination, latency management, and state synchronization. Existing XR pipelines are primarily designed for locally available, pre-authored assets, and thus provide limited support for asynchronous, network-driven generation workflows.

In addition to communication challenges, generative 3D models often exhibit unpredictable structural properties, including inconsistent coordinate origins, irregular scales, and non-standard material representations. When combined with modern XR rendering pipelines, which rely heavily on automated shader compilation and runtime abstraction, these factors frequently lead to unstable visualization behavior, incorrect spatial alignment, or loss of material fidelity during runtime asset loading. Naïvely importing generative assets into XR environments without explicit runtime handling often results in brittle or inconsistent user experiences.

These issues highlight that near real-time XR integration of generative 3D content is not solely a generation problem, but a system-level challenge involving communication architecture, runtime asset assembly, and spatial interaction design. Motivated

by this observation, this project aims to explore how a coordinated communication and visualization framework can effectively bridge XR devices, server-side generative pipelines, and runtime rendering constraints, enabling stable and interactive visualization of generative 3D models in XR environments.

B. Project objectives and questions

The primary objective of this project is to design and implement an end-to-end system that supports near real-time generation and stable runtime visualization of generative 3D models within XR environments. Rather than focusing solely on generative model quality, the project emphasizes system coordination across XR devices, server-side generation pipelines, and runtime visualization components.

Specifically, the project pursues the following objectives:

1) To design a distributed communication framework enabling interaction between XR head-mounted devices, a server-based control layer, and a generative 3D pipeline.
2) To implement an asynchronous generation workflow that supports user-initiated input, server-side processing, and result delivery with explicit state management.
3) To develop a runtime asset assembly strategy that explicitly handles geometry, textures, spatial placement, and scale for generative 3D models arriving over network communication.
4) To integrate spatial interaction and visualization mechanisms in XR environments, including stable placement, rotation, floating motion, and display mode switching.
5) To evaluate the robustness and responsiveness of the system through real-device deployment and interactive demonstrations on XR hardware.

Based on these objectives, the project seeks to answer the following research questions:

- How can communication between XR devices, server-side controllers, and generative 3D pipelines be coordinated to support near real-time user-driven content generation?
- What runtime design strategies are effective for assembling and visualizing generative 3D assets transmitted over asynchronous network channels?
- How does explicit system-level handling of communication, runtime assembly, and spatial interaction improve the stability and usability of generative 3D visualization in XR environments?

II. System Design and Methodology

A. Overall system architecture

Fig. 1 illustrates the overall architecture of the proposed system. The system is composed of three major components: an XR runtime environment deployed on a head-mounted device, a server-side control layer, and a generative 3D pipeline based on ComfyUI. User interactions are captured in the XR environment and transmitted to the server through a REST-based communication interface. The server coordinates the generation workflow and manages system states asynchronously. Generated geometry and texture assets are then delivered back to the XR runtime, where they are explicitly assembled and visualized at runtime.

B. Passthrough ROI Capture

Passthrough ROI Capture serves as the primary interface between the physical environment and the generative 3D pipeline in the proposed system. Mathematically, the capture process extracts a well-defined region of interest (ROI) from the two-dimensional passthrough image stream, where the ROI is represented as a bounded subset of image pixels aligned with the user's viewing direction. In XR environments, this region is not arbitrarily defined in image space, but is anchored through a virtual framing interface that is aligned with the XR camera's view direction. The center of the framing window corresponds to the camera's forward axis, while the ROI dimensions are determined by the mapping between the virtual user interface and the camera intrinsics.

From the perspective of projection geometry, this process can be viewed as a standard mapping from a three-dimensional view frustum to a two-dimensional image plane, expressed as

$$\Pi : (X, Y, Z) \rightarrow (u, v),$$

where the ROI effectively corresponds to a sub-frustum selected in the projection space. This formulation ensures geometric consistency between what the user observes in the XR environment and the

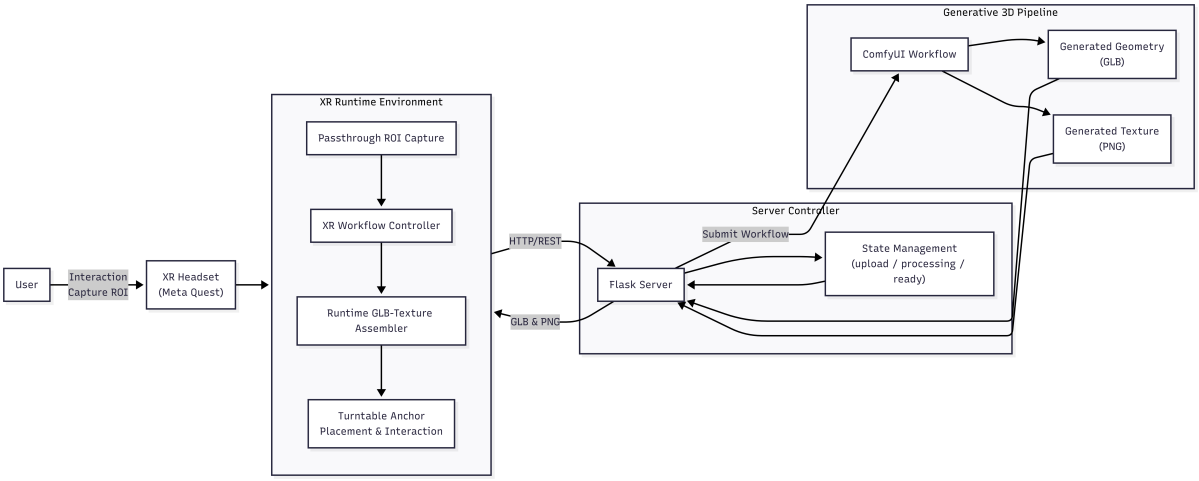


Fig. 1: Overall architecture of the proposed XR system. The system consists of an XR runtime environment, a server-side orchestration layer, and a ComfyUI-based generative 3D pipeline. User input is captured on the XR headset, transmitted to the server for workflow coordination, and processed by the generative pipeline. Generated geometry and texture assets are then returned to the XR runtime for explicit assembly and visualization.

visual data processed by the generative model, supporting deterministic and reproducible input selection. The design directly leverages fundamental concepts of camera projection and spatial mapping covered in this course, demonstrating their practical applicability in XR system development.

From an engineering perspective, ROI-based capture functions as an explicit input constraint that mitigates several practical challenges associated with directly using full passthrough video frames. Raw camera frames often contain excessive background information, incur high network transmission costs due to their resolution, and introduce input instability caused by continuous viewpoint changes. By constraining user input to a focused visual region, the system encodes spatial user intent into a compact and transferable visual representation. This significantly reduces communication overhead, shortens end-to-end latency, and stabilizes the input distribution for downstream generation.

Furthermore, ROI Capture integrates naturally with system-level mechanisms such as input blocking, frame buffering and discarding strategies, and asynchronous state-driven communication. This coordination enables reliable generation triggering under network latency and runtime uncertainty, without introducing frame-level ambiguity or excessive computational load. These properties make Passthrough ROI Capture a foundational component that supports robust, efficient, and user-intuitive generative interaction in XR runtime environments.

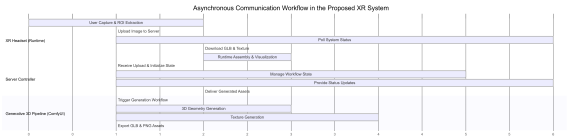


Fig. 2: Asynchronous execution timeline of the proposed system illustrated as a Gantt diagram. The diagram highlights the non-blocking interaction between the XR runtime, the server controller, and the generative pipeline, where geometry and texture generation proceed independently while the XR runtime remains responsive through state-driven polling.

C. Communication and Workflow Coordination

Fig. 2 presents the interaction sequence among the XR runtime, server controller, and generative 3D pipeline. The XR device initiates the workflow through user-driven capture and communicates with the server using a non-blocking, asynchronous protocol. Generation tasks are executed independently within the generative pipeline, while the XR runtime periodically polls the server for status updates. Once generation is completed, geometry and texture assets are transmitted separately and explicitly assembled at runtime, enabling stable visualization without interrupting user interaction.

Fig. 3 illustrates the complete temporal sequence of the proposed workflow. First, the user initiates the capture process by pressing the trigger button on the controller. During this stage, the XR headset captures real-world imagery through its camera and

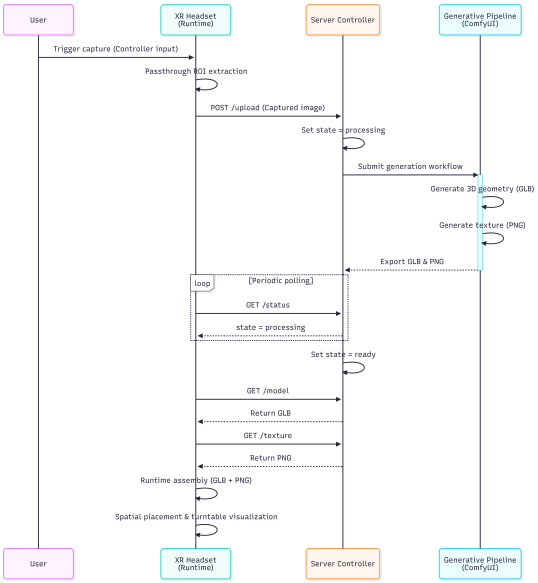


Fig. 3: Interaction sequence among the XR runtime, server controller, and ComfyUI-based generative pipeline. The sequence illustrates the asynchronous communication pattern, including user-triggered capture, server-side workflow orchestration, periodic status polling, asset retrieval, and runtime assembly of generated geometry and textures.

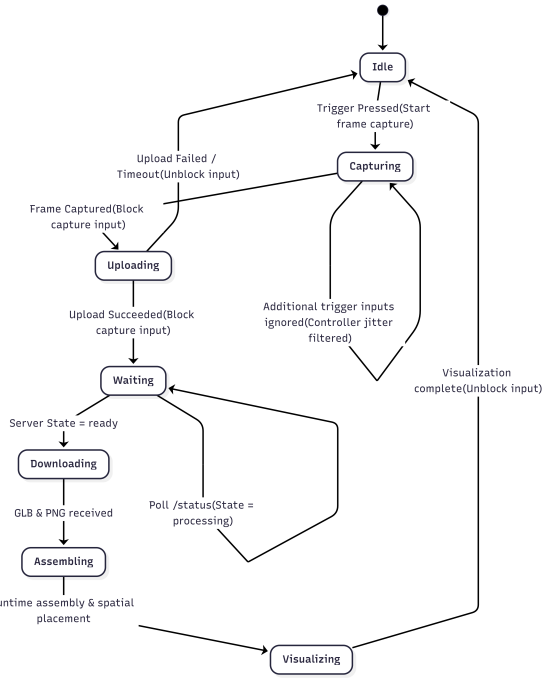


Fig. 4: State transition diagram of the XR workflow controller. The diagram shows how user input is gated through explicit states to prevent duplicate requests caused by controller jitter, while coordinating asynchronous upload, generation, asset retrieval, and visualization without blocking overall XR interaction.

crops video frames based on a predefined framing region in the virtual scene. It should be noted that at this point, subsequent user inputs are not blocked.

Once image processing is completed, the captured image is uploaded to the server via network communication. Based on the upload result, the XR headset determines whether to block subsequent capture and upload operations, while other interactions remain unaffected. If the upload fails, no further action is taken; if the upload succeeds, the XR runtime enters a blocking state and begins polling the server for processing status.

Upon receiving the image, the server immediately transitions into a processing state and triggers a 3D model generation workflow hosted on an external service, namely ComfyUI, through an API call. During this stage, the server continuously monitors the execution status of the ComfyUI workflow by tracking task identifiers and generated output files.

After the ComfyUI workflow completes, the generated geometry (GLB file) and texture assets are saved to designated directories. The server then detects the completion signal, retrieves the generated assets, and updates the system state to ready.

The XR headset receives this state update through subsequent status polling requests and, in the next retrieval phase, downloads the generated geometry and texture assets from the server. These assets are explicitly assembled at runtime for visualization. Finally, the assembled model is spatially placed and rendered in the XR environment, after which the temporary blocking on user input is released.

There are several engineering details in the system that warrant careful consideration. One important aspect is the management of user input blocking in the XR Workflow Controller. A key design question concerns when user input should be blocked. If blocking is applied only after receiving a response from the server, multiple image uploads may occur within a short time window, since the user remains able to trigger new captures before the previous request completes. This behavior can cause the server to issue excessive requests to the ComfyUI service, potentially overloading its task queue.

In addition, hardware limitations of the controller may introduce input jitter, where a single physical button press inadvertently generates multiple input events. This phenomenon can further result in redundant generation tasks being submitted to the server,

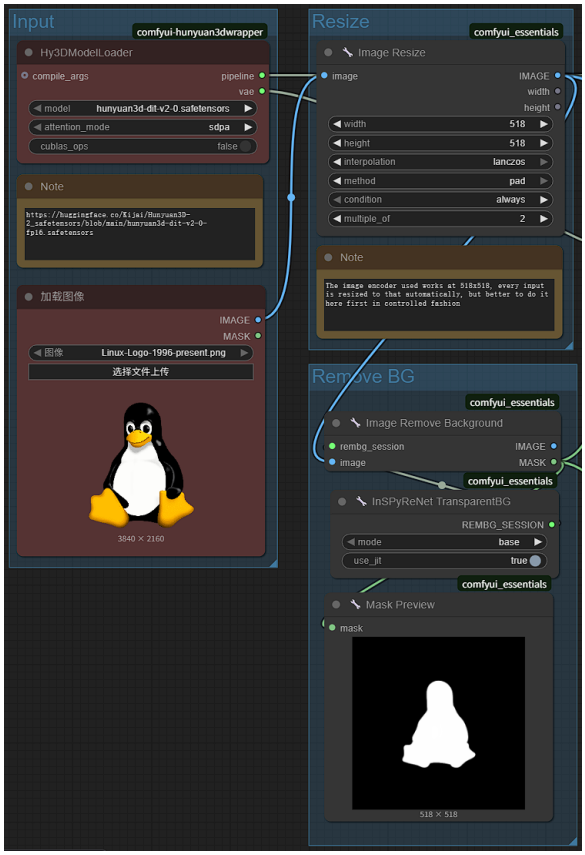


Fig. 5: Input preprocessing and background removal stage of the ComfyUI-based generative 3D workflow.

consuming unnecessary computational resources. To mitigate these issues, the adopted solution is to immediately block user input as soon as the first video frame capture command is triggered, thereby preventing jitter-induced duplicate requests.

User input blocking is released once the upload process completes and a server response is obtained, including cases of failure or timeout. Given the short duration of this blocking interval, users typically do not perceive any noticeable interruption under normal conditions. When generation proceeds successfully, the blocking state is maintained until the entire workflow completes. Furthermore, an additional frame buffering and discarding mechanism is implemented to avoid incorrect decision-making caused by network latency, ensuring robust interaction control under asynchronous conditions.

D. ComfyUI-Based Generative 3D Model Generation Workflow

The generative creation of 3D content in the proposed system is implemented through a modular workflow constructed in ComfyUI, with the Hunyuan3D family of models serving as the core generative backbone. The workflow begins by loading a pre-trained image-conditioned 3D generation model (Hunyuan3D-dit-v2.0) through a dedicated model loader node. In parallel, the captured passthrough ROI image is ingested and resized to a fixed square resolution to ensure compatibility and stable conditioning across subsequent inference stages. This preprocessed image is then distributed to multiple downstream branches in the workflow, enabling concurrent geometry and appearance processing.

Geometry generation is performed in the GenerateMesh stage, where the input image is transformed into an intermediate latent representation from which a detailed three-dimensional surface is decoded. Rather than exposing the internal architectural details of the generative model, this process can be abstracted as an image-conditioned mapping from visual input to polygonal geometry. The resulting mesh is subsequently refined through post-processing operations, including topology simplification and structural cleanup, to produce a representation suitable for real-time visualization and runtime loading in XR environments.

In parallel with geometry synthesis, a background removal branch is executed to isolate the image region corresponding to meaningful foreground content. This process produces a binary mask that identifies valid visual regions and filters out irrelevant background information. The generated mask is later reused in the appearance processing pipeline to ensure consistent treatment of foreground content across multiple views.

To decouple material appearance from scene-dependent lighting, the workflow incorporates a dedicated de-lighting stage based on the Hunyuan3D-delight-v2.0 model. Guided by the previously generated mask, this stage removes illumination effects from the input image to produce an albedo-like representation suitable for texture generation. This step is particularly important for XR runtime visualization, where lighting conditions may differ substantially from those present in the original captured scene.

Following geometry and appearance preprocess-

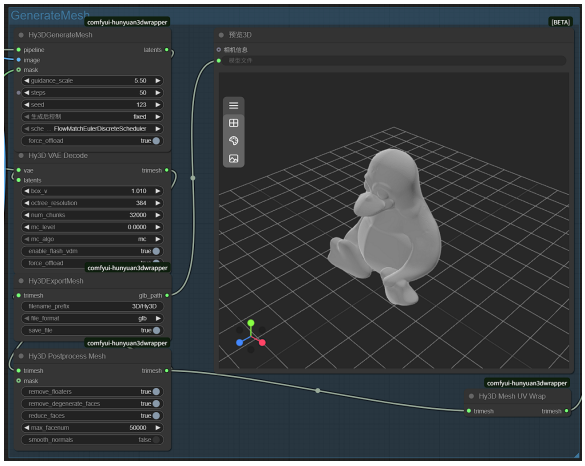


Fig. 6: Geometry generation and mesh post-processing stage in the ComfyUI-based workflow.

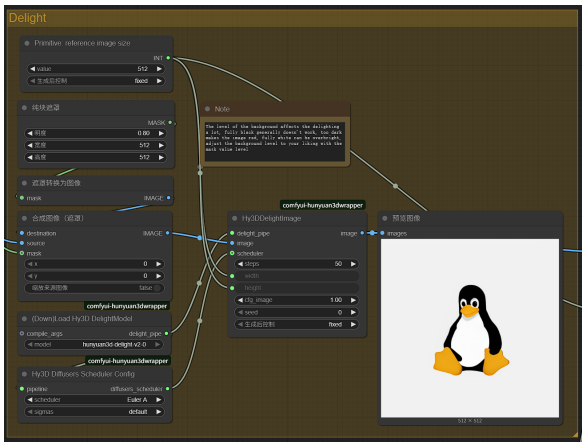


Fig. 7: Illumination removal (delighting) stage in the ComfyUI-based generative workflow.

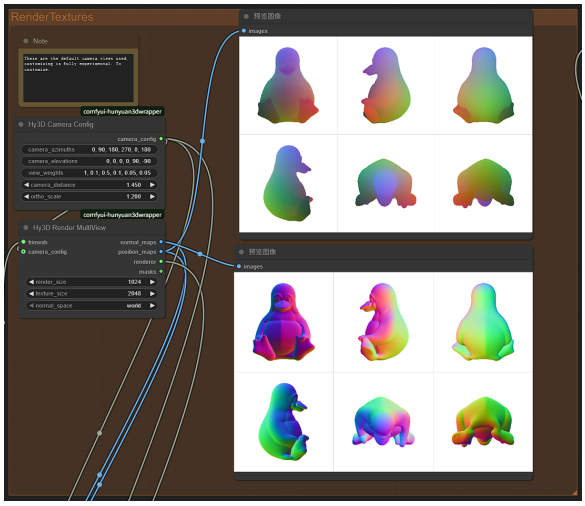


Fig. 8: Multi-view rendering stage for texture sampling in the ComfyUI-based workflow.

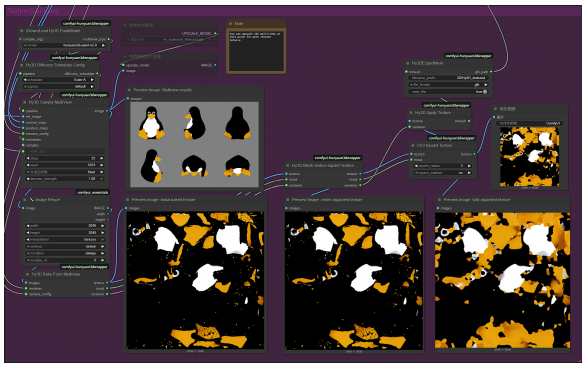


Fig. 9: Multi-view texture sampling and UV texture baking stage in the ComfyUI-based workflow.

ing, the workflow employs a multi-view rendering stage to derive image-to-surface correspondences. Using predefined virtual camera parameters, the generated mesh is rendered from multiple viewpoints, producing spatial mapping information that establishes consistent relationships between image pixels and mesh surface coordinates. These multi-view projections serve as geometric constraints for subsequent texture synthesis.

In the final TextureSampling stage, a texture painting model (Hunyuan3D-paint-v2.0) integrates the de-lighted appearance information with the multi-view geometric mappings to perform consistent texture sampling across viewpoints. The synthesized texture is then baked into a unified UV map associated with the generated mesh. Importantly, the workflow explicitly exports geometry and texture as separate assets—a GLB file for mesh representation and a standalone image file for texture data.

This explicit separation is a deliberate design choice motivated by XR runtime constraints. In practical XR rendering pipelines, materials generated through shader graphs cannot be reliably reconstructed during dynamic loading, particularly when assets are introduced at runtime. By decoupling geometry from texture representation at the generative stage, the system preserves full control over material binding and rendering behavior during runtime assembly. As a result, the ComfyUI-based workflow produces generation outputs that are directly compatible with the system's explicit runtime assembly strategy, providing a robust foundation for near real-time visualization of generative 3D models in XR environments.

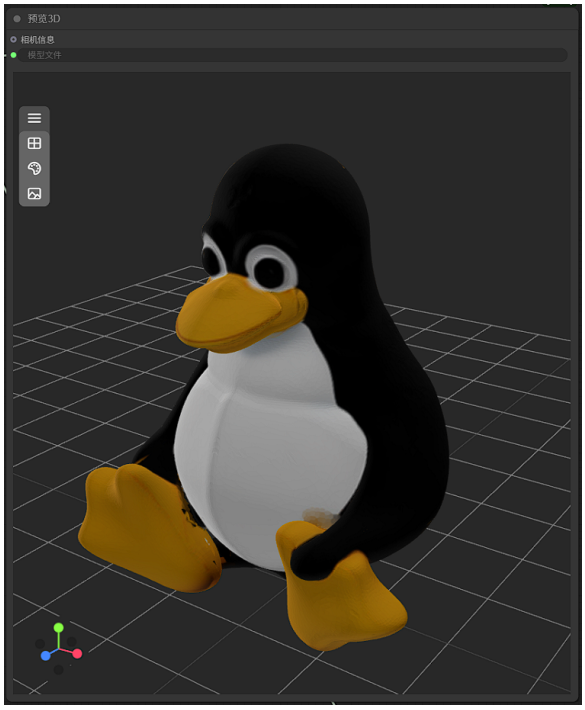


Fig. 10: Final generated 3D model visualized after texture baking in the ComfyUI environment.

E. Runtime Assembly of Generative Assets in XR

The outputs produced by the ComfyUI-based generative pipeline are not directly visualized within the XR environment. Instead, the proposed system adopts an explicit runtime assembly strategy that reconstructs generative assets under the constraints of XR rendering pipelines. This design choice is motivated by the observation that automatically generated 3D assets often lack reliable runtime semantics when imported as monolithic objects, particularly when advanced shader graph systems are involved.

Upon completion of the generative workflow, geometry and appearance data are delivered to the XR runtime as two independent artifacts: a polygonal mesh encoded in a GLB file and a corresponding texture image. Rather than relying on embedded material definitions, the XR runtime explicitly instantiates a native rendering material and binds the received texture to the appropriate input channel. This approach bypasses implicit shader compilation mechanisms, ensuring that appearance information remains accessible and controllable at runtime. The separation of geometry and texture thus provides a clear and deterministic interface for asset reconstruction.

After loading the geometry and texture assets, the XR runtime performs assembly under a controlled spatial hierarchy. The generative model is attached to a dedicated Turntable Anchor, as described in the previous section, which serves as the sole reference frame for visualization-related transformations. By design, the internal transform of the generative model remains unchanged, while all spatial operations—including rotation, scaling, and animation—are applied at the anchor level. This ensures that inconsistencies in pivot placement or model coordinates do not propagate into the user-facing visualization.

To accommodate the wide variability in scale inherent to generative 3D models, the system further applies an automatic normalization process during runtime assembly. The spatial extent of the assembled model is estimated using its bounding volume, and a uniform scaling factor is computed such that the model fits within a predefined perceptual size range. This normalization step guarantees consistent visual presentation across different generative outputs and viewing contexts, without requiring manual asset adjustment or offline preprocessing.

Finally, the assembled asset is positioned relative to a higher-level spatial root within the XR scene, enabling predictable placement with respect to the user's viewpoint and interaction space. At this stage, the generative model becomes a fully interactive XR object, supporting stable rotation, floating animation, and display mode switching. Because all assembly steps occur dynamically at runtime, the system remains flexible to variations in generative output while preserving robust visualization behavior.

Through this explicit runtime assembly strategy, the system establishes a clear separation between generative content creation and XR presentation logic. Geometry synthesis and texture generation are treated as external processes, while spatial normalization, material binding, and interaction semantics are handled entirely within the XR runtime. This separation allows the system to integrate generative 3D assets reliably into immersive environments, providing a stable foundation for near real-time visualization despite the inherent unpredictability of generative pipelines.

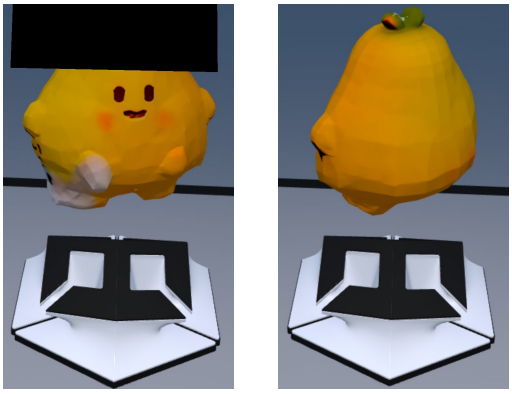
F. Turntable Anchor for Stable Spatial Visualization

A key challenge in visualizing generative 3D models within XR environments lies in achieving perceptually stable and spatially consistent object rotation and presentation. Unlike manually authored assets, generative 3D models often exhibit unpredictable pivot positions, irregular coordinate origins, and heterogeneous internal structures. Directly rotating such models around their native object pivots frequently leads to visually unstable behavior, such as unintended orbital motion or apparent drifting, which degrades the user experience.

To address this issue, the proposed system introduces a Turntable Anchor abstraction that explicitly decouples spatial presentation from the intrinsic coordinate properties of generated models. Instead of attempting to geometrically correct or re-center the model's internal pivot, a dedicated anchor object is created within the XR runtime as a stable spatial reference. The generative 3D model is attached as a child of this anchor without modifying its original local transform, while all presentation-related transformations—such as rotation, floating motion, and scaling—are applied exclusively to the anchor.

From a geometric perspective, this design establishes a clear separation between the model's local coordinate system and the visualization coordinate system used for XR interaction. Let T_m denote the local transform of the generative model and T_a denote the transform of the Turntable Anchor. The final world-space transformation of the rendered model is given by the composition $T = T_a \circ T_m$, where only T_a is dynamically updated during interaction. By constraining rotational transformations to the anchor, the rotation axis remains fixed and well-defined in world space, independent of the model's internal geometry.

This anchoring strategy offers several practical advantages. First, it guarantees stable and predictable rotation behavior, ensuring that models visually rotate in place rather than exhibiting unintended translational artifacts. Second, it avoids reliance on geometric heuristics such as bounding-box-based pivot correction, which can be unreliable for complex or asymmetrical generative models. Finally, the Turntable Anchor integrates naturally with other spatial presentation mechanisms, including automatic scaling, floating animation, and passthrough or virtual scene switching, all of which operate at the anchor level.



(a) (b)

Fig. 11: Runtime visualization of a generated 3D model after texture binding in the XR environment.(a)front (b)back

By introducing the Turntable Anchor as an explicit runtime abstraction, the system provides a robust and perceptually coherent solution for presenting generative 3D models in XR environments. This approach prioritizes visual stability and user perception over strict geometric manipulation of model data, making it particularly well suited for real-time visualization of automatically generated assets with limited structural guarantees.

III. Implementation and Results

A. System Implementation Overview

The proposed system was implemented as a distributed XR application consisting of three main components: an XR runtime deployed on a standalone head-mounted display, a server-side orchestration module, and a ComfyUI-based generative 3D pipeline. The XR application was developed using Unity and executed on a Meta Quest headset, enabling real-time passthrough capture, interaction control, and runtime visualization. The server component was implemented as a lightweight controller responsible for workflow coordination, state management, and asset routing, while all computationally intensive generative tasks were delegated to the ComfyUI backend. Communication between the XR runtime and the server was realized through a REST-based interface, supporting asynchronous upload, status polling, and asset retrieval. The system was designed to operate entirely at runtime without requiring any offline preprocessing or editor-based asset preparation, reflecting



(a) (b)

Fig. 12: User input acquisition and blocking states in the XR runtime.(a) Passthrough ROI capture with visual framing prior to generation. (b) Visual indication of input blocking during active generation.

realistic deployment conditions for XR applications that integrate generative content.

Fig. 11 presents a representative example of a generated 3D model visualized in the XR environment after runtime assembly. The model geometry and texture were successfully reconstructed from assets generated by the ComfyUI workflow, demonstrating correct texture binding and stable spatial placement. The explicit runtime assembly strategy ensured that the generated model was rendered with consistent appearance despite being dynamically loaded at runtime.

In all tested cases, the Turntable Anchor mechanism enabled stable rotation of generated models around a fixed spatial reference, avoiding unintended orbital motion or visual drift. The rotation and floating animations remained smooth and perceptually stable, even when the internal coordinate structures of generative models varied across different runs.

B. Interaction and Workflow Behavior

The system's interaction workflow was evaluated by repeatedly triggering the generative pipeline through user input on the XR controller. The input blocking strategy implemented in the XR workflow controller effectively prevented redundant uploads caused by rapid or jittered trigger inputs. During active generation phases, capture and upload operations were temporarily gated, while other XR interactions such as view manipulation remained responsive. The asynchronous communication design ensured that the XR runtime was not blocked while waiting for server-side processing. Status polling allowed the XR application to remain interactive throughout the generation process, with visual feedback provided to indicate system state transitions. Upon generation completion, geometry and texture assets were retrieved and assembled without requiring user intervention, confirming the effectiveness of the state-driven coordination mechanism.

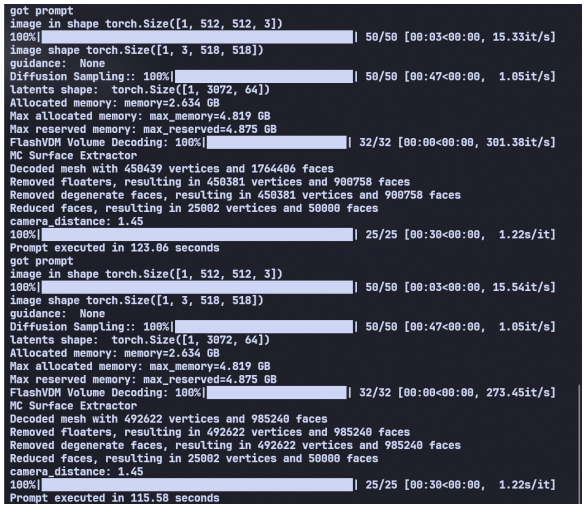


Fig. 13: Runtime execution log of the ComfyUI-based generative pipeline.

C. Performance Characteristics and Practical Observations

In practical usage, the overall generation latency from user-triggered capture to final visualization was on the order of several ten seconds, depending on the complexity of the generative pipeline and server-

side workload. While this latency does not meet strict real-time constraints, it is consistent with the design goal of near real-time interaction, where user engagement is preserved through asynchronous execution and visual feedback. The system demonstrated stable behavior across multiple consecutive generations. No persistent failures or unrecoverable error states were observed during testing. In cases of upload failure or generation timeout, the workflow controller correctly released input blocking and returned the system to an idle state, allowing subsequent attempts without requiring application restart. Overall, the results indicate that the proposed system successfully integrates generative 3D pipelines into an XR runtime environment, achieving reliable visualization and interaction under realistic constraints.

Use of Artificial Intelligence Tools

During the preparation of this paper, artificial intelligence tools were used to assist with language editing, structural refinement and layout guidance. No AI-generated content was used to perform original analysis, generate data, or determine conclusions. The author retains full responsibility for the accuracy and integrity of the work.

Note. All figures in this report are original creations by the author. The referenced literature is utilized to provide engineering insights and validation support, and as such, is not explicitly cited within the main text.

References

[1] R. van Glabbeek, U. Goltz, and J.-W. Schicke, "On synchronous and asynchronous interaction in distributed systems," 2008.

[2] J. Shin, M. A. Hedderich, A. Lucero, and A. Oulasvirta, "Chatbots facilitating consensus-building in asynchronous co-design," in *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, ser. Uist '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3526113.3545671

[3] T. Kojic, M. Vergari, M. Warsinke, D. Ali, S. Möller, and J.-N. Voigt-Antons, "Multimodal user experience in extended reality: Exploring hand tracking, voice, and passthrough interactions," in *Proceedings of the 17th International Workshop on Immersive Mixed and Virtual Environment Systems*, ser. Mmve '25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 8–14. [Online]. Available: https://doi.org/10.1145/3712677.3720459

[4] J. Han, F. Kokkinos, and P. Torr, "VFusion3D: Learning scalable 3D generative models from video diffusion models," 2024.

[5] M. Behravan, M. Haghani, and D. Gracanin, "Transcending dimensions using generative AI: Real-time 3D model generation in augmented reality," 2025.

[6] "How Can I Generate 3D Models with ComfyUI? Complete Guide — 3daistudio.com," [Accessed 05-04-2026]. [Online]. Available: https://www.3daistudio.com/{M}odels/{C}omfy{U}{I}-3{D}-{M}odels

[7] Behance, "3D Turntable Audio Visualizer - After Efffects Template - Roland Hartmann — behance.net," [Accessed 05-04-2026]. [Online]. Available: https://www.behance.net/gallery/108873925/3{D}-{T}urntable-{A}udio-{V}isualizer-{A}fter-{E}fffects-{T}emplate

[8] "Free resource: The Best 3D Turntable | CG Channel — cgchannel.com," [Accessed 05-04-2026]. [Online]. Available: https://www.cgchannel.com/2024/04/free-resource-the-best-3d-turntable/

[9] D. (Lin), "Hunyuan 3D 3.0 in ComfyUI: State-of-the-Art 3D Asset Generation — blog.comfy.org," [Accessed 05-04-2026]. [Online]. Available: https://blog.comfy.org/p/hunyuan-3d-30-in-comfyui-state-of

[10] "GitHub - facebookresearch/vfusion3d: [ECCV 2024] Code for VFusion3D: Learning Scalable 3D Generative Models from Video Diffusion Models — github.com," [Accessed 05-04-2026]. [Online]. Available: https://github.com/facebookresearch/vfusion3d?tab=readme-ov-file

[11] "GitHub - MrForExample/ComfyUI-3D-Pack: An extensive node suite that enables ComfyUI to process 3D inputs (Mesh & UV Texture, etc) using cutting edge algorithms (3DGS, NeRF, etc.) — github.com," [Accessed 05-04-2026]. [Online]. Available: https://github.com/{M}r{F}or{E}xample/{C}omfy{U}{I}-3{D}-{P}ack

[12] "GitHub - rdeioris/glTFRuntime: Unreal Engine Plugin for loading glTF files at runtime — github.com," [Accessed 05-04-2026]. [Online]. Available: https://github.com/rdeioris/gl{T}{F}{R}untime

[13] "glTF - Runtime 3D Asset Delivery — khronos.org," [Accessed 05-04-2026]. [Online]. Available: https://www.khronos.org/gltf/

[14] E. Dritsas, M. Trigka, C. Troussas, and P. Mylonas, "Multimodal interaction, interfaces, and communication: A survey," *Multimodal Technologies and Interaction*, vol. 9, no. 6, 2025. [Online]. Available: https://www.mdpi.com/2414-4088/9/1/6

[15] "ComfyUI Trellis2 Workflow | Image-to-3D Mesh, Remeshing & Texturing — runcomfy.com," [Accessed 05-04-2026]. [Online]. Available: https://www.runcomfy.com/comfyui-workflows/comfyui-trellis2-workflow-advanced-structure-image-generation

[16] H. Kim, S. Lee, and C. Kang, "From controllers to multimodal input: A chronological review of XR interaction across device generations," *Sensors*, vol. 26, no. 196, 2026. [Online]. Available: https://www.mdpi.com/1424-8220/26/1/196

[17] "[PPT] - Synchronous and Asynchronous Interaction in Distributed Systems PowerPoint Presentation - 95933 — sambuz.com," [Accessed 05-04-2026]. [Online]. Available: https://www.sambuz.com/doc/synchronous-and-asynchronous-interaction-in-distributed-ppt-presentation-95933

[18] “[PPT] - Synchronous and Asynchronous Interaction in Distributed Systems PowerPoint Presentation - 95933 — sambuz.com,” [Accessed 05-04-2026]. [Online]. Available: https://www.sambuz.com/doc/synchronous-and-asynchronous-interaction-in-distributed-ppt-presentation-95933