Project Topic Proposal: VR First Prompt to Walkable World System (Holodeck)

Leslie

February 12, 2026

Project Topic and Overview

This project builds a VR-first system that converts a user's natural language prompt into a walkable, explorable 3D environment. The user enters VR immediately into a stable "Creation Room" (the room of origin), where they can speak what they want to create. If voice is unavailable or fails, the system provides a text fallback so generation is always possible. After the prompt is submitted, the requested environment is compiled off to the side as a separate destination world. Once a minimum playability threshold is met, the system spawns a portal that the user can step through to enter the generated world.

The core design goal is to prioritize comfort, stability, and reliability. Instead of generating arbitrary 3D geometry from scratch, the system compiles worlds from a controlled library of pre-ingested assets and templates, and applies "StyleKits" to rapidly match the intended theme. The generated world first appears as a playable greybox, then upgrades visually in phases as assets stream in. This approach is meant to reduce perceived latency and prevent disorienting "world formation around the body" effects.

Significance and Related Work

A relevant direction is scalable scene generation for embodied AI and interactive simulation. Procedural systems can produce large collections of varied environments by sampling layouts, placing interactive objects, and randomizing materials and lighting, while enforcing physical validity and interaction consistency. ProcTHOR demonstrates large-scale procedural generation of interactive indoor environments for embodied AI and reports generalization benefits when training on procedurally generated scenes [1]. This project adopts a similar emphasis on validity and repeatability, treating a "prompt to world" request as a compilation problem with constraints, budgets, and a deterministic intermediate representation.

Prompt-driven environment generation is an increasingly important direction because it reduces the manual effort required to build interactive 3D spaces. Prior work demonstrates that language models can produce structured scene descriptions and constraints that can be compiled into interactive environments, rather than treating generation as a purely visual task. For example, Holodeck uses language to specify objects and relationships and

then compiles those constraints into an interactive 3D environment by selecting assets and optimizing object placement [3]. This project builds on that direction and emphasizes a VR-first user experience with a compiler-style pipeline that guarantees walkability and safety.

Recent work also shows that a text prompt can be converted into a room scale textured 3D mesh by lifting outputs from 2D text to image models into a consistent 3D representation [2].

The motivation for this project is to combine these insights into a practical, testable system: a user can request many different environment concepts, and the system consistently produces a playable world that matches the prompt as closely as possible within the limits of its asset library. The project emphasizes measurable outcomes, including time to first walkable world, compilation success rates, and runtime performance constraints appropriate for VR.

Goals of the Work

The main goals are:

1. VR-first generation loop: a user can enter VR, submit a prompt by voice or text, observe compilation progress, and walk through the generated world via a portal transition.

2. Reliable world compilation: prompts are converted into a structured intermediate representation called `WorldSpec`. The system validates `WorldSpec` against a schema and runtime constraints before generating the destination world. The compiler is deterministic so results can be reproduced from the same spec and seed.

3. Fast time to walkable: the destination world becomes playable quickly by loading a walkable greybox layout first, then progressively streaming improvements in phases.

4. Graceful missing asset handling: if a prompt requests objects not present in the library, the system substitutes similar assets or uses placeholders while preserving theme and functionality.

Software to Be Created

This project will create a Unity (Quest-native) OpenXR VR application (C# runtime scripts) plus backend services that together implement a planner-to-compiler pipeline.

Client (VR application)

- A Creation Room that always loads first and contains a minimal assistant UI.
- Voice input with transcript display and editing, plus a text fallback input path.
- Generation status and error messaging in VR.
- A portal system that transitions the user from the Creation Room into the compiled destination world, and supports returning home.

Backend (planning, compilation, and validation)

- A Pack Registry that stores asset packs, templates, tags, and performance metadata.
- A StyleKit system that defines theme presets (lighting, material palette, ambience, decals, post effects).
- A planner endpoint that maps prompt text to a schema-locked `WorldSpec` using only valid pack, template, and StyleKit IDs from the registries.
- A validator that checks schema correctness, walkability constraints, asset availability, and performance budgets.
- A compiler that converts `WorldSpec` into a destination world bundle and a progressive streaming manifest.

Development Plan

The work will proceed in phases aligned with the implementation roadmap in the appendix:

- Build the VR shell and Creation Room with locomotion and UI.
- Implement voice and text prompt input and the end-to-end request flow.
- Implement Pack Registry and StyleKit formats, loaders, and lookup endpoints.
- Define `WorldSpec` schema and validation.
- Implement greybox compilation, safe spawn placement, and portal transitions.
- Add progressive streaming phases for style, props, and ambience.
- Add missing asset substitution, budget enforcement, and automated tests.

Evaluation Plan

This project will be evaluated using metrics that reflect reliability and usability:

- Time to first playable world: time from prompt submission to a walkable destination with safe spawn and teleport surface.
- Compilation success rate: percentage of prompts that produce a valid `WorldSpec` and pass validation without manual intervention.
- Constraint satisfaction: collision validity, walkability checks, budget checks, and missing asset substitution frequency.
- Runtime performance: stability under target settings, including draw calls, texture tier limits, and frame time consistency.

Architecture Diagrams

Figure 1 presents a simplified end-to-end view of the prompt-to-walkable-world pipeline. Figure 2 expands this into a more detailed architecture view that separates planning, validation, compilation, and progressive streaming responsibilities.

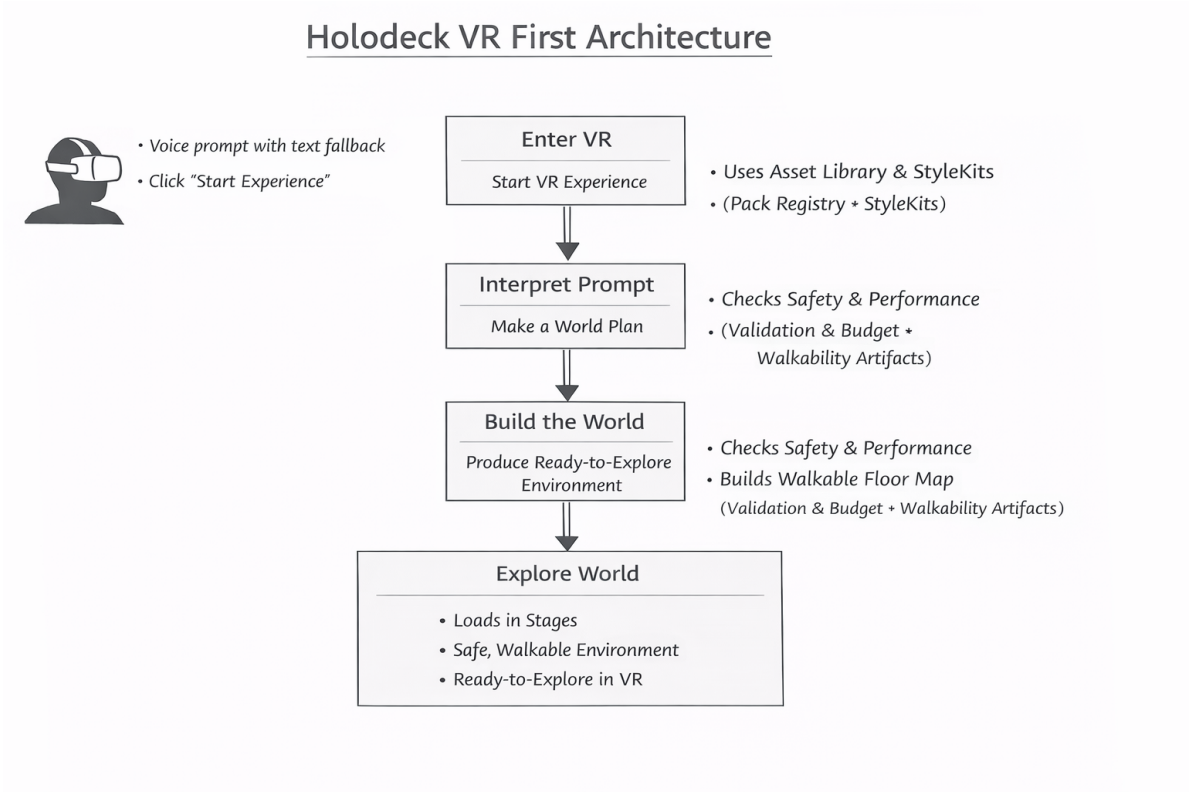

Figure 1: Simplified architecture for the VR-first prompt-to-walkable-world system.

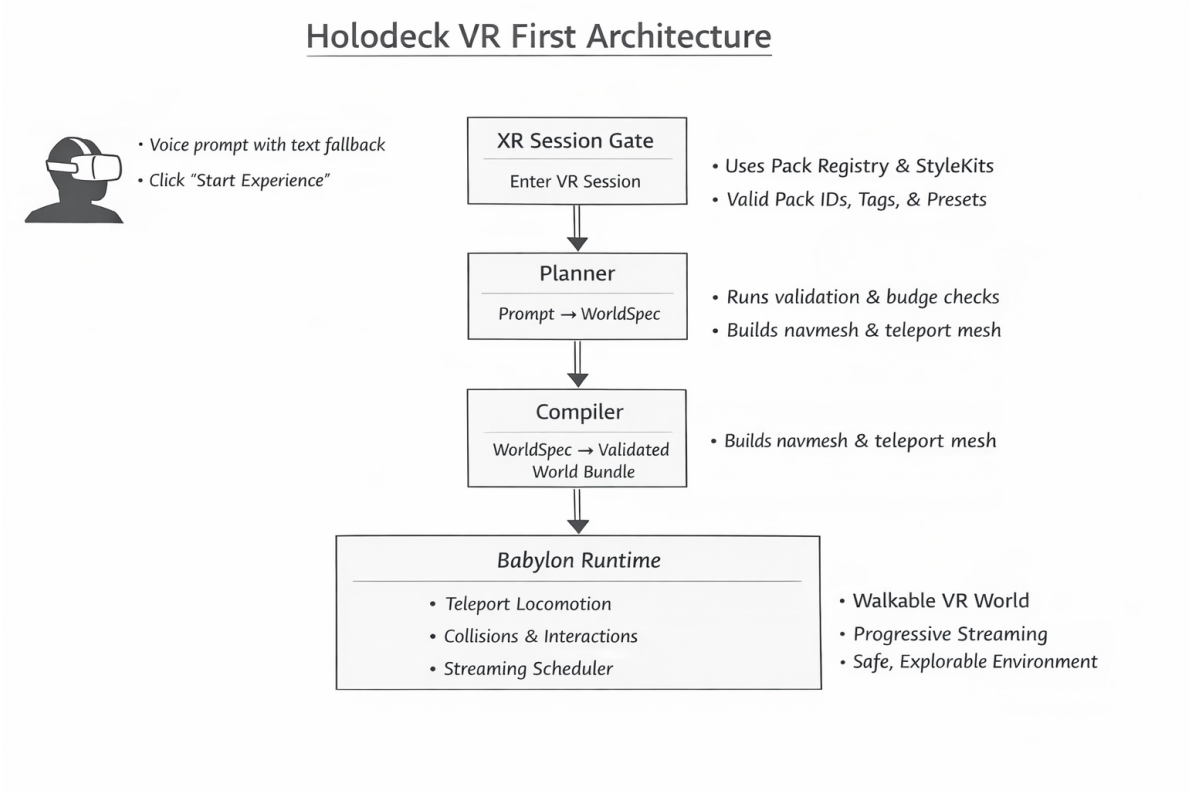

Figure 2: Detailed (complex) architecture for planning, validation, compilation, and streaming.

Appendix: Planned Feature List

Roadmap, implementation-level.

Core Features (in Planned Completion Order)

1. Create the Creation Room base scene in Unity with floor, walls, ceiling, and a fixed safe spawn point.

2. Add teleport locomotion with a visible teleport arc and restrict targets to valid floor surfaces.

3. Implement an in-VR Assistant Panel UI with transcript area and buttons (Start Recording, Stop, Submit Prompt, Clear, Return Home).

4. Implement microphone permission handling on user gesture and graceful error states for denied permission.

5. Implement speech-to-text transcript capture, display transcript in VR, and allow manual transcript edits.

6. Implement text prompt fallback input in VR and route both voice and text into the same prompt submission path.

7. Define a versioned `WorldSpec` JSON Schema including `template_id`, `stylekit_id`, `pack_ids`, `seed`, `room_graph`, `placements`, `interactions`, and `budgets`.

8. Implement `POST /validate_worldspec` to validate `WorldSpec` against JSON Schema and return structured validation errors.

9. Define Pack Registry manifest format (`pack_id`, version, tags, asset list, template compatibility, performance metadata).

10. Implement Pack Registry loader that scans pack folders, loads manifests, validates them, and builds an in-memory index.

11. Implement Pack Registry endpoints `GET /packs/search` and `GET /packs/<id>` with tag-based filtering and ranking.

12. Define StyleKit manifest format (lighting preset, material palette, decals, ambience audio, postfx preset, budget overrides).

13. Implement StyleKit loader and endpoints `GET /stylekits` and `GET /stylekits/<id>`.

14. Implement `POST /plan` that converts prompt text into a schema-locked `WorldSpec` using only valid registry IDs.

15. Implement deterministic seed handling so repeated `WorldSpec` inputs compile to the same layout and placements.

16. Implement Compiler Phase 0: generate destination greybox geometry from template and `room_graph`, and attach colliders.

17. Implement teleport surface tagging in the destination world and ensure floors are valid landing targets.

18. Implement safe spawn placement in the destination world that guarantees no collisions at spawn.

19. Implement portal spawning in the Creation Room once the destination world reaches minimum playability.

20. Implement Return Home action that restores the Creation Room scene without reloading the page.

21. Implement Phase 1 style loading: apply StyleKit lighting and base material palette to the greybox.

22. Implement Phase 2 prop streaming: load and place large props first, then secondary props and clutter.

23. Implement Phase 3 polish: load ambience audio, decals, and post-processing presets after props load.

24. Implement performance budget checks (lights, texture tiers, prop counts) and automatic downgrades when budgets exceed limits.

25. Implement missing asset substitution rules: replace missing assets with closest tagged substitutes or fall back to placeholders and log substitutions.

26. Add unit tests for schema validation, registry loading, and planner outputs, plus an end-to-end smoke test that verifies walkability.

Stretch Goals (Only After the Core System Is Complete)

27. Add a missing asset quick options UI that offers 2–3 alternatives (substitutes or theme adjustments) and generates a new portal variant.

28. Add world remixing: save `WorldSpec` plus pack versions and enable safe regeneration with a modified StyleKit or adjusted prop density.

29. Add safe live edits via an allowlisted `StyleKitPatch` schema supporting lighting, fog, ambience, and material palette swaps, with no structural edits.

References

[1] Deitke, M., VanderBilt, E., Herrasti, A., Weihs, L., Ehsani, K., Salvador, J., Han, W., Kolve, E., Kembhavi, A., and Mottaghi, R. ProcTHOR: Large-Scale Embodied AI Using Procedural Generation. In *Advances in Neural Information Processing Systems* (2022), vol. 35.

[2] Höllein, L., Cao, A., Owens, A., Johnson, J., and Niessner, M. Text2Room: Extracting Textured 3D Meshes from 2D Text to Image Models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (2023), pp. 7909–7920.

[3] Yang, Y., Sun, F.-Y., Weihs, L., VanderBilt, E., Herrasti, A., Han, W., Wu, J., Haber, N., Krishna, R., Liu, L., Callison-Burch, C., Yatskar, M., Kembhavi, A., and Clark, C. Holodeck: Language Guided Generation of 3D Embodied AI Environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2024), pp. 16227–16237.