Mind Reality

Abstract — 本專題旨在解決傳統 3D 建模流程繁瑣、耗時且門檻高的問題。隨著 AI 技術快速發展，AI 已能進行自然語言對話、程式撰寫與任務輔助。因此，本專題建置一套 AI 即時 3D 模型生成系統，讓 AI 依據使用者需求自動產生 3D 物件生成程式與動畫掛載程式，以快速、高自由度、低成本的方式取代傳統建模流程。本系統結合混合實境（MR, Mixed Reality）技術，並以 LARR（Localization-Assisted Remote Rendering）遠端算繪架構下的 MR 平台為基礎，支援使用者可透過多裝置同時連線至伺服器，並與 Realtime API 進行互動；此技術實現了「對話即建模」的自動化流程，系統會根據對話內容生成 3D 物件，回傳至 MR 環境中。使用者可進一步透過手勢觸碰、拖移與操作生成物，實現 AI 生成與 MR 互動整合的應用。

I. 簡介

這個專題的靈感來自科幻電影裡那種超酷的虛擬 3D 操作畫面：模型浮在空中，可以直接用手旋轉、調整、互動。

我們想，與其只能使用預先製作好的模型，不如像「言出法隨」一樣，說一句話就能生成 3D 物件，那不是更酷嗎？因此，我們希望結合 AI 與 MR 技術，打造一個如同變魔法般的系統，讓 3D 創作變得更直覺、更有趣。

1. 「言出法隨」的魔法體驗

想像一下，做 3D 模型像施魔法一樣，一句「我想要一隻活潑的機械鳥」或「生成一架帥氣的戰鬥機」，系統就能理解你的想法，並即時生成對應的 3D 物件與動畫，讓原本只存在腦中的畫面快速浮現眼前，相較傳統建模門檻降低。

透過 Real-time AI 與混合實境（MR）的結合，模型不只停留在螢幕上，更能被帶入真實空間，讓使用者以手勢觸碰、拖移與操作，體驗言出即現的互動感。未來這項技術可以應用在教育教材、遊戲角色與場景生成、產品設計原型，以及建築與室內預覽，甚至延伸到醫療、工程訓練和展覽導覽，讓創造變得更簡單即時，也更有想像力。

2. 會思考的 AI

不只是「你說什麼，它就做什麼」，我們希望打造的是一個「會思考的 AI」：能看懂環境、理解需求，並主動給出建議，幫助使用者完善想法。系統具備拍照功能，可讓 AI 觀察目前場景，理解空間中的物件並根據現場狀況推薦適合的生成內容。另外，AI 也能理解語言語意。當使用者提出具體物件，例如「一隻活潑的鳥」時，AI 不只會生成鳥本身，還能根據語意聯想到相關元素，如樹枝、鳥巢、花草或天空裝飾，進一步生成兩個搭配物件，讓畫面更自然，也更有情境感。



Figure 1. Mind Reality 未來應用圖

II. 系統說明

本專題的系統架構圖如 Fig. 2 所示，以下針對主要平台與硬體設備做說明：

1. LARR 多人混合實境遠端算繪平台

LARR（Localization-Assisted Remote Rendering）多人混合實境即時互動遠端算繪平台。使用者配戴的 HoloLens 為 Client 端，而 Server 端負責進行主要的運算與畫面處理，Server 根據 Client 端的位置與相機視角，計算虛擬物件與使用者視角之間的相對關係。接著，Server 會調整對應的物件相機角度，產生符合使用者目前視角的虛擬影像，最後再將影像即時回傳到 HoloLens 上顯示。在此架構中，我們也會使用到以下相關技術。

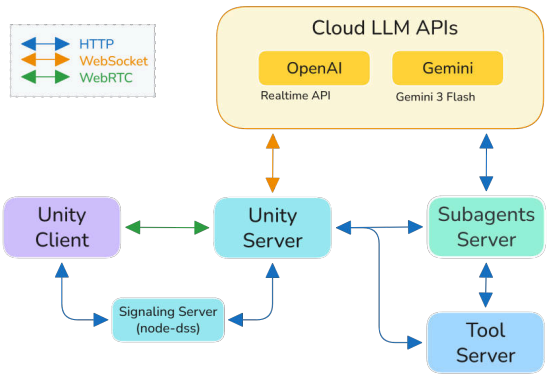


Figure 2. 系統架構

(a) WebRTC

WebRTC 是一種用來實現即時通訊的開放標準，讓不同裝置可以即時交換多媒體資料。在我們專題中，WebRTC 負責 Server 端、Client 端直接的資料傳輸，以低延遲傳送影像、聲音、3D 物件或控制指令。

(b) NodeDSS

NodeDSS 是一個作為 Signaling Server 的開源程式，可讓 Server 與 Client 在建立 WebRTC 連線時交換雙方所需的網路位址與通道參數，以建立 P2P 連線。

(c) MRTK

MRTK（Mixed Reality Toolkit）是由 Microsoft 推出的混合實境開發工具組，用來讓 HoloLens / MR / XR 更容易做出手勢、注視、互動抓取功能。

2. Unity

Unity 是 Unity Technologies 開發的跨平台引擎，廣泛應用於遊戲與延展實境 (XR) 開發。本專題使用 Unity 建構混合實境場景並實作互動流程。

3. Microsoft HoloLens 2

HoloLens 2 是 Microsoft 推出的混合實境頭戴裝置，使用者配戴後可在真實環境中看見虛擬 3D 物件，透過手勢、眼球追蹤與語音進行互動。本專題以 HoloLens 2 作為 Client 端裝置，提供使用者直接互動。

4. LangChain 與 LangGraph

LangChain 是一套用於建構 LLM 應用與 AI 工作流的開源框架。LangGraph 則是其延伸框架，將應用邏輯抽象為圖 (graph) 結構，以節點 (node) 代表函式或 LLM 呼叫、邊 (edge) 定義節點間的路由邏輯，支援循環、分支與狀態持久化，能更靈活地控制執行流程，適合多代理協作等複雜場景。本專題搭配 LangSmith 進行監控與除錯，以追蹤執行軌跡並快速定位問題。

5. Realtime API

Realtime API 是 OpenAI 推出的一項即時互動介面，主要用來建立低延遲、多模態 AI 應用，同時支援函式呼叫（function calling）。Realtime API 是讓 AI 可以「即時聽、即時想、即時說或回傳指令」的介面，很適合語音助理、MR 控制、即時互動系統。

6. LLMs

Large Language Models (LLMs) 擅長語言理解與程式生成。本專題依任務特性選用合適的模型驅動系統運作。具體任務所使用的模型如下：

- 使用者互動介面：GPT-Realtime-1.5
- 生成物件與動畫：Gemini 3 Flash Preview

GPT-Realtime-1.5 之低延遲與多模態能力，能提供流暢自然的互動體驗；而 Gemini 3 Flash Preview 兼顧推理速度與語義理解，適合用於即時程式碼生成。

III. 方法與設計

本章節將說明本專題之整體方法與系統設計，涵蓋從使用者輸入到 3D 物件生成與動畫生成之完整流程，其整體流程如 Fig. 3 所示，實現互動式對話建模。

在物件生成部分（如 Fig. 5 所示），系統透過 Three.js 作為中介表示，並結合即時算繪與多視角評估機制；在動畫生成部分，我們提出 Planning-based Architecture，使 LLM 能受控生成動畫腳本（如 Fig. 7 所示）。

此外，本系統亦結合上下文工程與視覺場景理解，整合多模態資訊以強化模型決策能力，提升整體系統之穩定性、可控性與互動性。

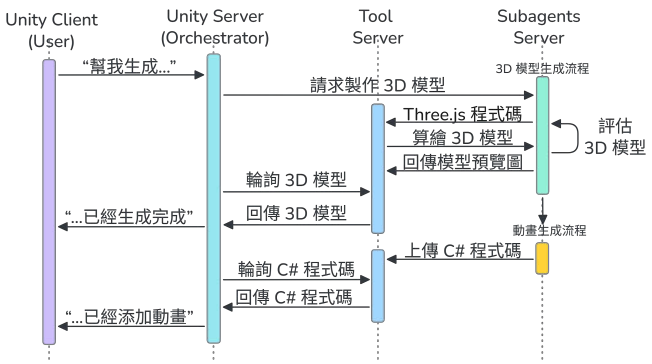


Figure 3. 物件與動畫生成流程

1. Multi-Agent 架構

本系統採用 Multi-Agent 架構，將不同職責的任務拆分由多個 Agent 分工協作。本架構採用 Supervisor 模式，由 Orchestrator 作為中央協調者，透過 Realtime API 與使用者進行即時語音互動，負責理解意

圖並將任務委派給對應子代理，待子代理 (Subagent) 回傳結果後整合回覆。對使用者而言，僅需面對單一互動入口，無需感知複雜的代理分工。

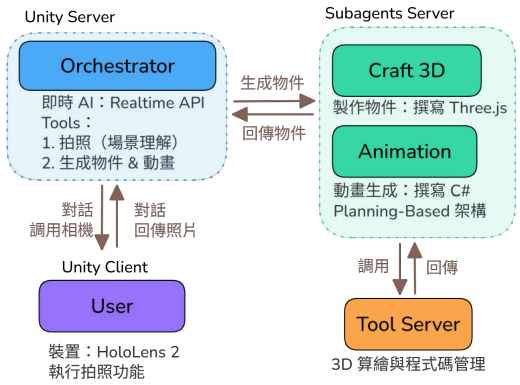


Figure 4. 本系統的 Multi-Agent 架構

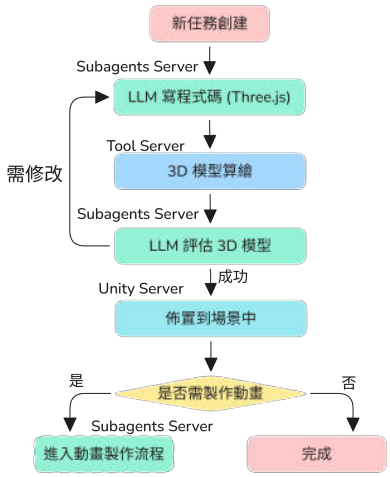


Figure 5. LLM 生成 3D 物件流程

各子代理皆以 LangGraph 框架實作，透過其圖式工作流與狀態管理機制定義代理內部的推理流程、工具呼叫與條件分支邏輯。子代理的關鍵作用在於上下文隔離 (Context Isolation)，每個子代理擁有獨立的系統提示與對話脈絡，僅接收與自身任務相關的資訊，看不到 Orchestrator 與使用者間的完整對話，也看不到其他子代理的工作內容。此設計使各子代理的提示內容單純、更專注於該領域任務，避免不同代理間的角色混淆與異常內容的污染，提升系統穩定性。

2. 生成物件

當系統接收到生成物件的任務指令後，會進入「生成與評估」的迭代迴圈，並在通過評估後將成品載入 Unity 場景中。在迭代迴圈中，LLM 首先根據需求撰寫 Three.js 程式碼，並在 Tool Server 中的無頭瀏覽器中執行、算繪出 3D 模型；產出的成果會經由多視角評估機制檢驗品質，若發現缺陷，則將評估結果回饋至程式碼撰寫階段重新生成，直到通過評估為止。最後，通過評估的模型以 glTF 格式輸出，並由 Unity 端的 glTFast 套件載入到 MR 場景中，完成整個生成流程。

(a) 程式碼生成

任務建立後，LLM 根據使用者描述產生 Three.js 程式碼，作為 3D 模型的中介表示。程式碼本質上是文字，能充分發揮 LLM 的文字生成能力；同時程式碼具有可讀性與可修改性，後續若評估發現缺陷，LLM 能針對程式碼進行局部調整，而非從零重新生成整個模型。

Three.js 是一個基於 WebGL 的 JavaScript 3D 函式庫，提供相機、光源、材質、幾何體、動畫等高階 API。它讓開發者能以較少的程式碼在瀏覽器中建立和算繪 3D 場景，大幅簡化原生 WebGL 的繁瑣流程。

(b) 模型算繪

LLM 產出的 Three.js 程式碼必須在瀏覽器環境中執行才能實際算繪為 3D 模型，因此本系統採用 Playwright 在伺服器端建立無頭瀏覽器實例，執行程式碼並擷取算繪結果。算繪完成後，系統會匯出 glTF 格式的模型檔，並從多個角度擷取截圖，作為下一階段評估的輸入。

Playwright 是一個瀏覽器自動化框架，支援無頭模式執行，適合在伺服器端自動操作網頁並擷取結果。本系統將 LLM 生成的程式碼透過 Playwright 在瀏覽器中執行並取得結果。

(c) 多視角評估

為確保生成的 3D 模型符合需求，本系統導入多模態 LLM 進行外觀評估。考量單一視角難以揭露背面、頂部或特定角度下的缺陷（如穿模、比例失衡、貼圖錯誤），系統以雙螺旋路徑在球面上均勻取樣 16 個相機位置進行拍攝，並將所有截圖交由 LLM 整體評估；若未通過，則回饋至程式碼生成階段觸發修正，至多評估進行 2 次。

(d) 3D 模型載入

通過多視角評估的 glTF 模型從 web 端傳送至 Unity 端，由 glTFast 套件載入後佈置到 MR 場景中，完成整個生成流程。

glTF（GL Transmission Format）是由 Khronos Group 制定的開放 3D 模型傳輸格式，被稱為「3D 界的 JPEG」，設計目標是體積小、載入快，適合即時算繪與網路傳輸。glTFast 則是 Unity 中高效能的 glTF 匯入套件，能在執行時動態載入模型。

3. 生成動畫

本專題之生成動畫流程建立於 3D 物件生成之後，作為模型完成後的後續處理機制。當模型生成完

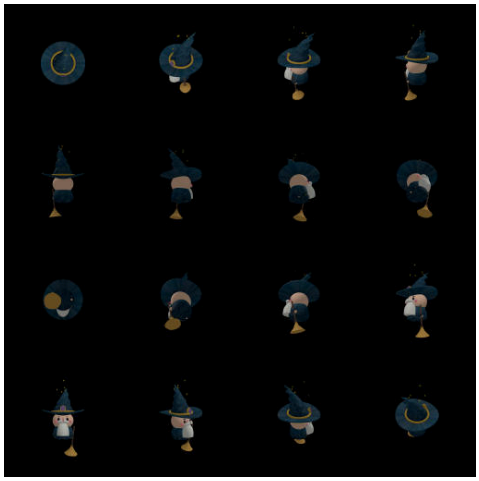


Figure 6. 多視角評估截圖範例（魔法師模型）



Figure 7. Planning-based Architecture 示意圖

後，系統除輸出 GLB 模型外，亦保留該任務之相關資訊，作為後續動畫生成之依據。

為確保大型語言模型生成之動畫腳本具備穩定性與可執行性，本專題提出 Planning-based Architecture 作為動畫生成之核心機制於 Unity 端建置一套動畫生成基礎架構，其整體流程如 Fig. 7 所示，包含以下三個核心元件：

- **Planner 基底類別**：所有動畫腳本皆需繼承統一基底類別，以限制操作範圍於目標物件及其語意子節點，確保執行流程受控。
- **動作函式庫**：提供預定義動畫操作（如位移、旋轉與特效），AI 僅能透過既定 API 呼叫，避免產生非預期行為。
- **流程控制模組**：負責動畫時序（順序、平行與循環），以維持行為之可預測性與一致性。

相較於傳統需預先製作 animation clip 或撰寫固定腳本之方式，本專題採用語意驅動（semantic-driven）的動畫生成方法，使新生成之 3D 物件於匯入 Unity 後即可具備對應之動畫行為與互動能力。

(a) Animation Agent

流程主要分為生成與儲存兩個階段。生成階段負責呼叫大型語言模型，根據 Tool Server 提供之資料產生 Unity C# 動畫規劃腳本；儲存階段則將生成結果回傳，作為該物件對應之動畫產物。模型會綜合物件結構與視覺特徵推論語意部位（如頭部、翅膀或特效發射點），並生成對齊之動畫行為。

在實作上，Animation Agent 嚴格限制輸出格式，要求僅生成一份符合規範之 planner script，且僅能調用既有動畫函式庫與時間控制模組，以避免不安全或與場景無關之操作。

(b) Planning-based Architecture

本專題提出之 Planning-based Architecture 建立於動畫資料庫之上，其核心概念為：在給定目標與場景狀態下，由系統自既有動作集合中選取並組合出一組動作序列（Action Sequence），以實現語意一致且結構連貫之動畫行為。

本方法受 Goal-Oriented Action Planning（GOAP）之設計理念啟發，但並未實作完整搜尋式規劃機制，而是透過大型語言模型進行高階動作編排，並結合受限之動作空間以確保生成結果之穩定性。

(c) Roslyn Runtime Compile

當 Tool Server 完成 planner 之生成與驗證後，Unity Server 於載入 GLB 物件後，向後端取得對應之動畫腳本，並利用 Roslyn 平台於執行期間將該 C# 程式即時編譯為 assembly，進而解析出其中之 planner 類別，並將其以 component 形式掛載至對應生成物件，使其立即執行動畫；若物件已存在舊版本 planner，則依據 metadata 進行替換以避免多重動畫衝突。此機制提供動態行為注入（dynamic behavior injection）能力，使系統無需重新建置 Unity 專案，即可於執行期間為新生成物件賦予對應之動畫行為。

(d) 方法優點

本專題提出之生成動畫方法具有三項主要優點。首先，其建立了從生成模型到生成動畫之完整流程，使 AI 所產生之 3D 內容不再僅為靜態模型，而能轉化為具備動態表現與互動能力之實體。其次，透過 Planner 架構與動畫資料庫之限制，系統在保留大型語言模型動作編排彈性之同時，有效避免生成不穩定或不可執行之程式碼。最後，結合 Roslyn runtime compile 與 Unity runtime attach 技術，使新生成動畫可即時套用於新生成物件，大幅提升系統靈活性與開發效率。

4. 上下文工程（Context Engineering）

本系統高度依賴 LLM 進行對話互動、程式碼生成與場景判斷。為提升輸出結果之穩定性與可控性，

本專題採用上下文工程（Context Engineering）方法，對 LLM 的輸入環境進行系統性設計。具體而言，系統將系統提示、對話歷史、場景截圖、現有程式碼及可用工具描述等多源資訊整合為統一上下文，作為模型輸入，使其能在完整語意脈絡下生成符合預期之結果。

在與使用者直接互動的 Realtime API 端，本系統透過 system prompt 規範 AI 的對話方式與行為界線，使其能在不同情境下引導使用者完成創作流程，例如協助釐清模糊的需求或適時提出建議，並避免做出超出系統能力範圍的回應。除對話規範外，系統亦透過提示詞描述各個 function call 的用途與觸發條件。

在後端 Agent 中，每一個環節皆針對其任務性質設計專屬提示詞。物件和動畫生成階段的提示詞會明確規範可使用的函式庫與其呼叫方式。

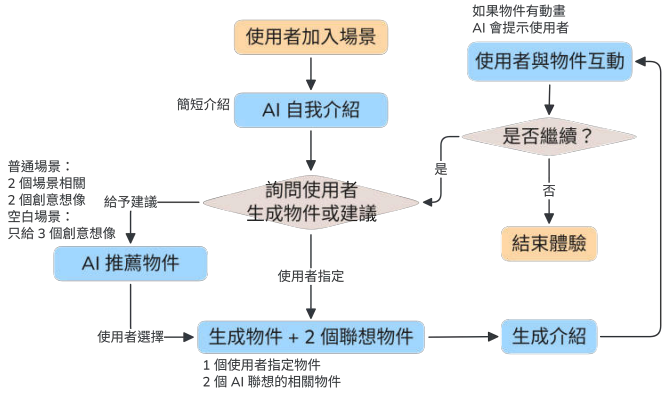


Figure 8. 在本專題中使用系統提示詞引導使用者

5. Visual Scene Understanding

為讓 AI 不只是被動執行指令，本系統賦予 AI 主動觀察真實環境的能力。Realtime API 可透過 function call 機制，主動呼叫 HoloLens 2 的前置攝影機進行截圖，將擷取的場景影像作為視覺輸入後，分析空間配置、既有物件與整體氛圍，並據此提出適合加入該場景的 3D 物件建議。使用者可從建議中選擇感興趣的項目，再由系統進行後續的 3D 物件生成。當截圖工具被調用時，Server 會發送指令給 Client，Client 取得截圖後回傳，Server 收集完成後再轉發給 Realtime API。工具資料透過 WebRTC 的 Data Channel 傳輸。

IV. 模擬與實驗成果與討論

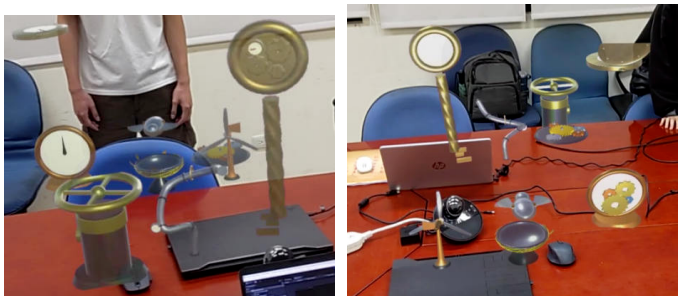
1. 模擬結果

完成應用程式建置後，我們成功在 Unity 中建立執行環境，並透過 MR 平台將多個 Client 端裝置連接至 Server 端。Fig. 10 為使用者視角下觀察 AI 生成物件的畫面，所有物件都具備動畫。

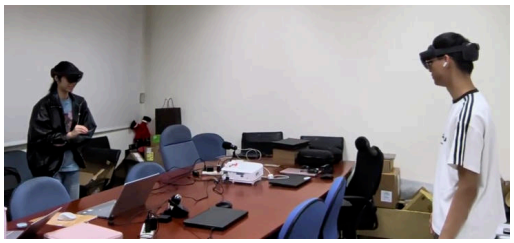
Fig. 11 為使用者在使用應用程式的過程中，使用 AI 生成的物件之展示，包含機械主題與金屬類物件，如齒輪組、機械鳥、閥門、風車等。



Figure 9. AI 理解場景並給予生成建議示意圖



(a) 使用者的視角　　**(b)** 另一個使用者的視角



(c) 使用者側拍畫面

Figure 10. 使用者生成以機械為主題的物件



Figure 11. 以機械主題生成的物件展示

2. 問題與討論

本章節將針對系統實作過程中所遭遇之關鍵問題進行分析，並探討對應之設計取捨與解決策略。說明本專題在穩定性、可控性與系統架構選擇上的實務考量。

(a) LLM 幻覺

在開發過程中，我們觀察到 LLM 的幻覺問題其一是 AI 在與使用者對話時宣稱自己具備系統未實作的能力，例如承諾超出流程範圍的功能；其二是 AI 在生成程式碼時調用不存在的 API、引用未匯入的函式庫，使生成結果無法執行。

為解決上述問題，我們將 AI 常見的錯誤類型在提示詞中明確規範。對話端的提示詞會限定 AI 可承諾的能力範圍；撰寫程式碼流程中的提示詞則會列舉可使用的函式庫與正確的呼叫方式，並標示常見錯誤寫法提醒 AI 迴避。透過反覆實測後的提示詞迭代，可有效降低幻覺發生的機率。

(b) 生成式程式碼之安全性設計

為避免 LLM 生成程式碼直接執行所帶來的系統汙染、資源濫用與流程阻塞風險，本專題採分層式安全設計，涵蓋 Prompt 約束、隔離執行、輸出驗證與失敗控制等機制。

在生成物件流程中，Three.js 程式碼不於 Subagents Server 執行，而是交由 Tool Server 進行獨立算繪，在受限的無頭瀏覽器環境中完成 GLB 生成與預覽圖擷取；Subagents Server 僅接收模型與影像結果，並透過 Timeout 避免異常程式長時間佔用資源。同時，系統僅擷取程式碼主體，以降低非程式內容進入執行流程的風險。

在 Prompt 約束層面，生成模板限制模型不得進行外部資源存取或任意 I/O 操作，並要求透過指定函式輸出結果，使可用 API 與資料流維持於封閉環境中。生成結果再經由多視角預覽圖進行檢查，並設置最大迭代次數以避免代理無限修正。

在生成動畫中，生成程式碼須繼承指定類別，且僅能調用預定義 API 操作目標物件。Tool Server 進一步進行靜態驗證，過濾危險 token 並檢查程式結構，僅將通過檢查之結果以文字形式保存，而不直接執行。

(c) 動畫生成技術之方法選擇

為實現即時且具語意一致性的動畫生成，本專題評估了兩種主要方法：基於模板之參數化動畫生成，以及完全由大型語言模型生成並即時編譯完整 C# 程式碼。

前者具備高穩定性與低延遲優勢，但在動作結構與整體連貫性上較為受限；後者則可生成高度自由且細緻之動畫行為，但在生成時間與執行穩定性方面存在顯著限制。

基於上述考量，本專題最終採用以 Planner 為核心之方法，並將其整合於前述之 Planning-based Architecture 中，以兼顧生成品質與系統效率。

(d) 文字轉 3D 模型 (Text-to-3D) 的實作方式

文字轉 3D 模型目前有兩種主流做法。其一為使用專門訓練的生成模型，多基於 Diffusion Model 或 NeRF 技術，直接由文字生成 3D 模型，優點是視覺真實感佳，但輸出為封閉的網格或點雲，難以局部修改。其二為透過 LLM 生成程式碼後算繪 3D 物件，雖然在擬真度上有所限制，但生成結果具備可讀性與可編輯性，並能透過多輪對話迭代修正。本專題選擇後者，以利後續整合 LLM 評估機制與互動式調整。

V. 結論

本專題成功建構出一套結合 AI 與 MR 的即時 3D 模型生成系統，讓使用者能透過語音與手勢直接與 AI 互動，將腦中構想快速轉化為可操作的 3D 內容。系統不僅能根據需求生成 3D 物件，更能進一步賦予其動畫與互動能力，使生成結果不再只是靜態模型，而是具備動態表現的實體。此外，AI 也能主動觀察真實環境並提出創作建議，讓互動體驗更貼近使用者情境。整體而言，本系統大幅降低了 3D 建模的門檻，使創作變得更直覺、更即時，也更具想像力。未來此技術可廣泛應用於教育、遊戲、產品設計與展覽導覽等領域，展現 AI 與 MR 整合應用的無限潛力。

參考資料

[1] H.-L. Zeng, Z.-T. Huang, and C.-W. Huang, “LARR: A Localization-Assisted Method to Conceal Latency Induced Position Errors in MR Remote Rendering,” in *2023 IEEE International Conference on Communications Workshops (ICC Workshops)*, Rome, Italy, 2023, pp. 1439–1444, doi: 10.1109/ICCWorkshops57953.2023.10283669.

[2] T. Guo et al., “Large Language Model based Multi-Agents: A Survey of Progress and Challenges,” in *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI)*, Jeju, Korea, 2024, pp. 8048–8057, doi: 10.24963/ijcai.2024/890.

[3] A. Madaan et al., “Self-Refine: Iterative Refinement with Self-Feedback,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023.

[4] M. Dalida and M. S. Lyan, “Multimodal agents tutorial: How to use Gemini, Langchain, and LangGraph to build agents for object detection,” *Google Cloud Blog*, Jun. 5, 2025. [Online]. Available: https://cloud.google.com/blog/products/ai-machine-learning/build-multimodal-agents-using-gemini-langchain-and-langgraph. Accessed: Apr. 2026.