
Il progetto **CAD Bridge** risponde alla richiesta della challenge di un'importazione "semplice e immediata" di modelli CAD in ambienti immersivi. Abbiamo definito un'architettura software prototipale che abilita la conversione e il caricamento dei modelli in Unity **a runtime** e **su richiesta**.

Il cuore della nostra soluzione risiede nel motore di conversione automatizzato. Questo componente si fa carico di tradurre i file CAD industriali (come **.step** e **.igs**) — che sono rappresentazioni matematiche precise (B-Rep) — in formati mesh leggeri (come **.obj**) necessari per il rendering in tempo reale. Il processo di *tassellazione* (la traduzione da superfici matematiche a poligoni) viene eseguito *on-demand* direttamente dall'Host (PC), eliminando la necessità di complesse ottimizzazioni manuali.

L'architettura è stata pensata per un contesto *PC-VR Tethered*, che risolve il principale ostacolo prestazionale permettendo la visualizzazione di modelli complessi senza alcun compromesso sulla **fedeltà visiva**. Abbiamo realizzato e validato un Proof-of-Concept (PoC) funzionante del solo motore di conversione, confermando la piena fattibilità tecnica della pipeline.

## 1. Analisi del Problema e Scelte Tecnologiche

Il workflow attuale per l'utilizzo di modelli CAD in VR è un collo di bottiglia. Richiede l'esportazione manuale da parte di un tecnico CAD, seguita da complessi passaggi di *data preparation* (ottimizzazione, decimazione, retopology) e conversione, solitamente eseguiti da uno specialista 3D. Questo processo è lento, costoso, richiede licenze software multiple e produce un asset statico: ogni modifica al progetto CAD richiede di ricominciare l'intero processo.

Il nostro obiettivo era progettare un'architettura che:

1. Automatizzasse il flusso di conversione da CAD a VR, rendendolo *on-demand*.
2. Preservasse la *massima fedeltà visiva* del modello, eliminando la necessità di decimazione manuale.
3. Utilizzasse strumenti flessibili e non proprietari per evitare *vendor lock-in*.

### Alternative Scartate

Abbiamo analizzato soluzioni *enterprise* come **Pixyz** e **Unity Reflect**. Sebbene siano potenti, presentano svantaggi chiave per il nostro contesto: sono soluzioni **proprietarie** e implicano un **costo di licenza elevato**, andando contro il nostro obiettivo di flessibilità e accessibilità. Altri convertitori presenti sull'Asset Store di Unity, invece, spesso non supportano la conversione *a runtime* o si affidano a servizi cloud esterni.

### Stack Tecnologico Scelto

Per superare questi ostacoli, abbiamo scelto una pipeline basata su strumenti open-source e scriptabili:



1. **Motore di Conversione: FreeCAD** È stato scelto come motore di conversione per i suoi vantaggi decisivi: è **open-source**, utilizza un robusto kernel CAD (OpenCASCADE) capace di leggere formati industriali (STEP, IGES, BREP), ed è **completamente scriptabile da riga di comando** (`freecadcmd`). Questo ci ha permesso di creare una pipeline automatizzata *headless* (senza interfaccia grafica) che gira sul PC Host.

2. **Importazione a Runtime: Runtime OBJ Importer** Per caricare il modello `.obj` generato all'interno della scena Unity a runtime, abbiamo previsto l'uso di un *importer* standard (come "As-is OBJ loader" o simili), che gestisce il parsing del file e la creazione delle mesh in C# al momento della richiesta.

**Scelta del Formato: .obj vs .fbx**
Abbiamo scelto **.obj** come formato mesh intermedio per la sua **semplicità** e **affidabilità**. È un formato di testo, facile da generare via script da FreeCAD (come dimostrato nel PoC) e da interpretare in Unity. Per la visualizzazione di modelli CAD, che sono tipicamente statici, è una scelta solida.

Il formato `.obj` è tuttavia limitato nella gestione di materiali PBR complessi e gerarchie di scena. Il formato **.fbx** sarebbe superiore sotto questi aspetti. Un'**implementazione futura** potrebbe evolvere la pipeline per generare file .fbx, ottenendo una gestione più ricca dei materiali e della struttura del modello. Questo richiederebbe l'integrazione di tool di conversione aggiuntivi (come Blender in modalità riga di comando) per tradurre l'output `.obj` in `.fbx` prima del caricamento.

## 2. Architettura della Soluzione Proposta

Proponiamo un'architettura software client-server eseguita interamente sul PC Host.

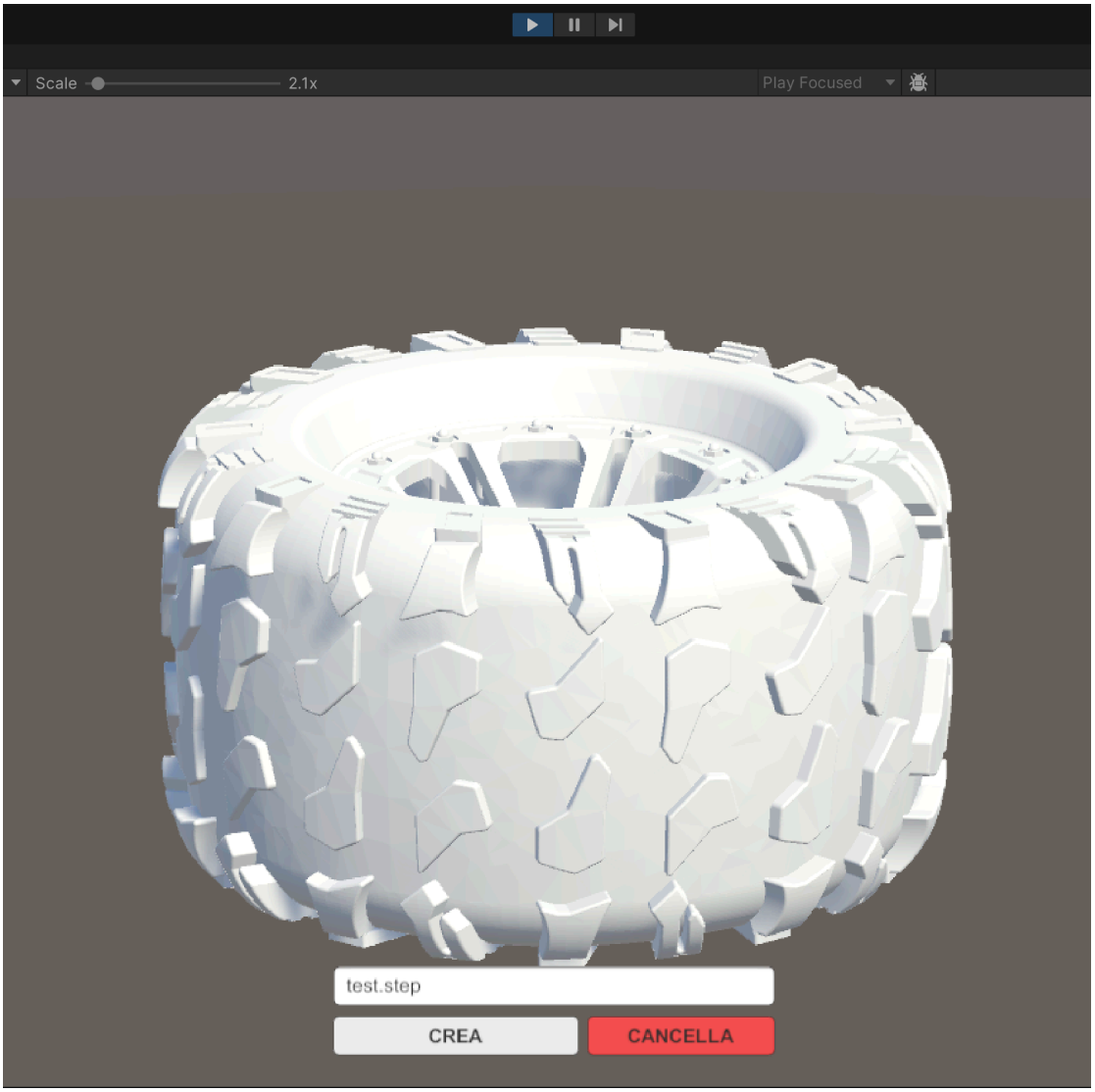
1. **Frontend (Client Unity):** L'utente interagisce con una semplice UI in Unity. Un campo di testo permette di inserire il nome del file CAD desiderato (es. `test.step`) e un bottone avvia la richiesta di conversione.
2. **Backend (Processo Esterno):** La pressione del bottone esegue uno script C# che avvia un processo esterno (`System.Diagnostics.Process`). Questo processo invoca l'eseguibile *headless* di FreeCAD (`freecadcmd`) passando come argomenti lo script Python di conversione e i parametri necessari.
3. **Conversione (Script Python):** Lo script Python si occupa di:
    - Costruire i percorsi di input (es. `cad_files/test.step`) e output (es. `converted_obj/test.obj`).
    - Leggere il file CAD tramite le librerie `Part` di FreeCAD.
    - Eseguire la *tassellazione* (conversione) in una mesh, usando la tolleranza specificata.
    - Salvare la mesh risultante come file `.obj`.
4. **Importazione (Ritorno a Unity):** Una volta che il processo esterno termina con successo, lo script C# in Unity carica il file `.obj` appena

generato utilizzando un *importer* a runtime (come Runtime OBJ Importer), istanziando il modello 3D direttamente nella scena.

**Input del Processo di Conversione**

Come dimostrato dal nostro PoC e dallo script, il processo avviato da Unity richiede i seguenti argomenti da riga di comando:

1. `freecadcmd`: Il percorso all'eseguibile di FreeCAD.
2. `<script.py>`: Il percorso al nostro script Python di conversione.
3. `<nome_file>`: Il nome del file CAD sorgente (es. `test.step`). Lo script si aspetta di trovarlo in una sottocartella predefinita (`cad_files`).
4. `[tolleranza]` (Opzionale): Un valore numerico (es. `0.05`) che definisce la precisione della mesh. Una tolleranza più bassa produce una mesh più dettagliata (più poligoni) ma più pesante. Se non specificato, viene usato un valore di default.




Importazione su Unity a runtime

## 3. Proof-of-Concept (PoC) e Validazione

Abbiamo implementato lo script di conversione e testato la pipeline:

1. *Dati di Test:* Abbiamo utilizzato modelli CAD industriali (formati .step e .igs) reperiti da *GrabCAD*.
2. *Script di Conversione:* Abbiamo sviluppato lo script che automatizza l'invocazione di FreeCAD per convertire i file di input in .obj.
3. *Test Parametrici:* Abbiamo eseguito test variando la *tolleranza di tassellazione* (es. 0.1 vs 0.001), confermando la possibilità di bilanciare velocità di conversione e fedeltà della mesh.
4. *Verifica in Unity:* I file .obj generati sono stati importati e renderizzati con successo. I nostri test hanno validato la pipeline con modelli complessi: ad esempio, **un file di test (`test.step`) convertito con una tolleranza di 0.05 mm ha prodotto una mesh da oltre 1.4 milioni di poligoni**, che è stata caricata senza problemi, confermando la validità dell'approccio PC-VR.

File .step su FreeCAD

File .obj generato

## 4. Analisi dei Limiti e Roadmap Futura

Grazie alla nostra architettura, il problema delle performance è già risolto. Il PoC presenta un unico, chiaro limite:

• *Limite Attuale: Perdita di Dati Semantici* Il formato .obj non supporta la *gerarchia dell'assieme* (l'albero dei componenti) né i *metadati* (nomi delle parti). È una "zuppa di poligoni" che impedisce interazioni complesse (es. "Seleziona la parte 'Bullone M12'"), fondamentali per le DWI.

**Roadmap di Sviluppo**

La nostra roadmap è focalizzata sull'abilitazione dell'interattività e sulla scalabilità:

1. *Fase 1: Transizione a glTF 2.0 (Abilitare l'Interattività)* Sostituire .obj con lo standard moderno .glTF (o .glb). Questo formato *preserva la gerarchia dell'assieme e i metadati*, risolvendo il nostro limite attuale e sbloccando la creazione di vere DWI interattive.
2. *Fase 2: Ottimizzazione Automatica (per la Scalabilità)* Introdurre un passaggio di *decimazione automatica* (es. tramite Blender/ MeshLab). Non per il rendering di un singolo modello (che già gestiamo), ma per *scalare* e permettere la visualizzazione di assiemi estremamente complessi (es. un'intera linea di produzione) mantenendo un framerate elevato.

## 5. Conclusione

Nelle 24 ore dell'hackathon, abbiamo progettato e validato un *Proof-of-Concept* per un'architettura "CAD-to-XR Bridge". Sebbene il prototipo attuale sia ben lontano da una soluzione robusta, risponde alla challenge dimostrando un approccio innovativo e ad alta fedeltà.

Automatizzando la conversione on-demand, il nostro PoC dimostra con successo la *fattibilità* di superare i lenti workflow tradizionali. La roadmap definita traccia un percorso chiaro per evolvere questa idea iniziale in una soluzione DWI industriale più matura e completa.