Università degli Studi di Padova
Dipartimento di Ingegneria Civile, Edile e Ambientale
Department of Civil, Environmental and Architectural Engineering
Corso di laurea in Ingegneria Edile-Architettura
Anno accademico 2021-22

Tesi di laurea

Sviluppo di una piattaforma di Hypermodeling per modelli BIM di edifici storici

Allievo: **Francesco Catania**

Relatore: **prof. Andrea Giordano**

Correlatori: **Paolo Borin,**
Rachele Angela Bernardello

ABSTRACT

L'adozione della metodologia BIM rappresenta ormai uno standard per le nuove costruzioni e si sta progressivamente affermando anche per la modellazione e analisi di edifici esistenti. Risulta invece ancora poco applicata quando si tratta di edifici storici e culturalmente rilevanti. Le ragioni di questa limitata applicazione sono da ricercarsi nella mancanza di strumenti che si adattino e soddisfino le richieste che questo flusso di lavoro poco esplorato comporterebbe, ma queste lacune non dovrebbero costituire un ostacolo tale da privarsi degli enormi vantaggi che la modellazione BIM comporta. Un'ulteriore limitazione è costituita dalla difficile interazione che le figure professionali di formazione storico e artistica possono incontrare quando devono interfacciarsi con un modello BIM: le attuali piattaforme di elaborazione sono eccessivamente tecniche per un utente non esperto, e quelle di visualizzazione non forniscono gli strumenti adatti ad un'analisi e interazione approfondita. Scopo di questo progetto è lo sviluppo di una piattaforma rivolta a storici dell'architettura e dell'arte, con la programmazione di strumenti e funzionalità personalizzate per supplire a queste mancanze. Il prodotto finale è un applicativo autonomo, che si interfaccia con i programmi di modellazione BIM, ne importa il modello e lo rende accessibile ed elaborabile con strumenti creati ad-hoc per venire incontro alle esigenze degli utenti di professione alternativa come gli storici dell'architettura.

Sommario

Indice delle figure

INTRODUZIONE

Con "Building Information Modeling" (BIM) si indica uno specifico metodo di sviluppo utilizzato nella progettazione e costruzione ingegneristica, architettonica e tecnica *(Architecture, Engineering and Construction, AEC)*, che consente di raccogliere, gestire, combinare e collegare digitalmente vari livelli e ambiti di conoscenza.
Questo tipo di metodologia, nato per essere usato nella progettazione e analisi dei nuovi edificati e nella loro gestione, è ormai largamente utilizzato anche nello studio di costruzioni preesistenti, ad esempio per abilitare la gestione di costi e manutenzioni. Tuttavia, nonostante l'adozione ormai sempre più diffusa di questo metodo, l'applicazione dello stesso all'analisi storica risulta tuttora limitata.

Nonostante ciò, l'utilizzo in questo specifico campo di applicazione può dimostrarsi un valido strumento, con le dovute modifiche. Allo stato attuale, la più grande difficoltà risiede nella difficile collaborazione con figure professionali esterne all'ambito AEC; nello specifico non sono presenti strumenti che consentano a professionisti storici e artistici di collaborare ad un modello BIM senza dover sottostare ad un corso che illustri l'utilizzo di programmi dedicati invece a ingegneri e progettisti.
A questo proposito dunque è necessario integrare ed analizzare all'interno del flusso di lavoro quali sono i requisiti necessari per l'analisi storico – architettonica, che possano essere impiegati e integrati in un modello BIM o in piattaforme di gestione, ed abilitare quindi una collaborazione efficace con le discipline della rappresentazione. Da questa considerazione nasce questo progetto, che a partire dall'analisi delle piattaforme e delle metodologie di lavoro esistente mira ad elaborare uno strumento di facile interfaccia e utilizzo che consenta anche a figure esterne all'ambito della rappresentazione informativa di collaborare alla produzione di modelli BIM per il patrimonio storico-culturale ed implementare quelle che sono le attività di ricerca proprie della loro disciplina, con la costruzione di funzionalità personalizzate.

La programmazione di queste nuove funzionalità risulta sotto molti punti di vista essenziale per migliorare lo scambio di informazioni tra le figure professionali di cui sopra: si identificano infatti due macrogruppi con funzioni distinte, un primo, composto da professionisti AEC, che si occuperanno di gestire la parte di progettazione e modellazione della componente BIM, ed un secondo gruppo, di tutti quei professionisti esterni all'ambito AEC, siano questi storici dell'architettura, storici dell'arte o anche semplicemente dei committenti, che hanno bisogno di interagire col modello, visionarlo e interrogarlo, senza tutte le conoscenze pregresse solitamente necessarie a maneggiare un programma di settore.

DEFINIZIONE DEL PROBLEMA

I programmi professionali per la produzione di modelli BIM possono risultare molto complessi agli occhi di utenti che non li conoscono approfonditamente, e questo può accadere quando figure professionali esterne all'ambito AEC devono collaborare ad un progetto di questo tipo.
L'interazione con il modello e la navigazione all'interno dell'interfaccia del programma possono non essere user-friendly e la loro fruizione può diventare elaborata e portare a inutili complicazioni o perdite di tempo, se non anche ad errori e fraintendimenti tra i vari componenti del team.
I professionisti dell'ambito storico e artistico, ad esempio, la cui presenza è fondamentale quando si lavora con strutture storiche e beni culturali, non hanno a disposizione uno strumento integrato e di agile utilizzo quando si tratta di dover mettere mano al progetto cui devono lavorare.
Alcune caratteristiche delle costruzioni in questione, inoltre, possono rendere ulteriormente elaborata la creazione del modello BIM, come ad esempio irregolarità geometriche più o meno accentuate, che vanno in contrasto con la modellazione parametrica tipica del BIM, o modifiche, ricostruzioni e demolizioni che hanno avuto luogo nel corso dei secoli e che si desidera rappresentare ed analizzare.

In questo studio si andrà a proporre uno strumento orientato all'utilizzo da parte di figure esterne al campo AEC, professionali e non, che permetta di interagire ed analizzare un modello professionale; nello specifico è stato preso come caso studio la chiesa di San Nicolò a Carpi (MO).

PIATTAFORME A CONFRONTO

Come punto di partenza si è scelto di considerare piattaforme già orientate alla modellazione BIM, che potessero quindi offrire nativamente degli strumenti utili al raggiungimento dello scopo prefissato. La mera presenza di questi strumenti, tuttavia, non è sufficiente a saggiare la bontà della piattaforma per l'utilizzo richiesto; come si vedrà infatti, piattaforme di base molto potenti possono essere inadatte in quanto difficilmente ampliabili, e al contrario piattaforme che nascono con tutt'altro scopo possono risultare talmente versatili da soddisfare tutte le richieste del caso. Le piattaforme che sono state prese in esame come punto di partenza per lo sviluppo sono state:

- **Autodesk Forge**
- **Blender**
- **Unity Reflect**

Per ciascuna di esse sono state valutate e analizzate le caratteristiche più significative, confrontate secondo parametri omogenei per procedere con la scelta della piattaforma stessa.

Requisiti delle piattaforme

Ricordando che scopo principale di questo progetto è avvicinare i modelli BIM ingegneristici a figure professionali di stampo storico e artistico, si rende evidente la necessità di offrire agli storici tutta una serie di strumenti che non sono attualmente integrati nelle piattaforme. Da un confronto diretto con professionisti che si devono interfacciare con modelli BIM, sono emerse principalmente le necessità di **poter visionare rappresentazioni bidimensionali direttamente all'interno del modello**, come ad esempio piante storiche o fotografie, per poter fare un confronto diretto con la rappresentazione tridimensionale elaborata, per fornire feedback o richiedere modifiche direttamente agli autori del modello stesso. Un ulteriore bisogno riscontrato è quello di poter visionare la rappresentazione delle varie **fasi storiche**, ovvero visualizzare in maniera distinta dei modelli che illustrino le varie modifiche subite dalla struttura nel corso della sua storia.
A queste richieste dovranno aggiungersi una serie di caratteristiche che la piattaforma deve presentare per essere effettivamente modificabile; deve infatti consentire un'agile elaborazione di componenti personalizzati, e questi devono avere a loro volta la possibilità di dialogare con il modello. In sintesi, l'estensione che andremo a compilare deve essere in grado di interagire con tutti gli elementi del modello importato, deve poterne leggere e modificarne le proprietà, questione che, come si vedrà in seguito, non risulta affatto triviale.
Considerati questi vari aspetti, i requisiti che sono stati ritenuti imprescindibili per la buona riuscita del progetto sono stati i seguenti:

- **Compatibilità con IFC**
  Possiamo definire come *formato proprietario* qualsiasi formato di file che non concede l'accesso a tutte le specifiche tecniche del file stesso, accessibili a pieno solo tramite software della stessa casa produttrice. Questo comporta una grossa limitazione nei casi di collaborazione e condivisione dei file, in quanto è necessario che tutti gli utenti dispongano di una licenza, ed in alcuni casi anche di versioni compatibili, per il software in questione nonché dispongano delle competenze necessarie per poterlo utilizzare. Nell'ambito della interoperabilità della modellazione informativa l'alternativa al formato proprietario è il *formato aperto*; nel caso dei modelli BIM, ci si riferisce principalmente al formato noto come *Industry Foundation Classes* (IFC). La maggior parte dei software AEC permette l'importazione e l'esportazione in formato aperto, anche se spesso questo comporta la perdita di alcune funzionalità. Nel caso di Revit, ad esempio, esportando in formato IFC si perde la parametricità degli elementi, in quanto tutte le geometrie vengono esportate come mesh; la perdita di queste utili funzionalità è dovuta alla necessità del formato IFC di mantenere la compatibilità con gli altri programmi AEC, e allo stesso tempo le varie case produttrici

incentivano all'utilizzo dei propri prodotti, rendendo queste possibilità aggiuntive una propria prerogativa.
In questo progetto si è deciso di lavorare concentrandosi sul formato aperto, in modo da rendere lo strumento il più orizzontale possibile, permettendo un utilizzo più diffuso; la possibilità di importare ed elaborare un file IFC risulta quindi una caratteristica imprescindibile per la piattaforma che andremo a scegliere.

- **Modello interrogabile**
  Di pari importanza alla necessità di importare modelli IFC possiamo considerare anche la possibilità di poter *interrogare* il modello BIM e ogni suo componente. Ogni elemento che costituisce il modello considerato (muri, porte, pilastri, travi ecc.) è dotato di una serie di proprietà, che possono essere comuni nell'intero progetto (nome, codice GUID, collocazione spaziale, ecc.) o specifiche della categoria (come la percentuale di inclinazione per un tetto piuttosto che la lunghezza di una trave). Poter accedere alle informazioni dei vari elementi è una delle caratteristiche principali degli elaborati prodotti con il modello BIM, ed in quanto tale, la piattaforma che andremo a scegliere deve supportare a pieno questa capacità. Senza di questa, il modello BIM si riduce ad una semplice rappresentazione 3D dell'edificio.

- **Facilità di ampliamento**
  Essendo obiettivo di questa ricerca la possibilità di ampliare la piattaforma, includendo quindi funzionalità aggiuntive e personalizzate, si palesa di enorme importanza la possibilità e l'agilità con cui la piattaforma consente di accedere, modificare ed estendere il suo codice sorgente.
  Va inoltre considerato che le piattaforme in analisi sono scritte in vari linguaggi di programmazione, e la loro complessità può variare molto. La conoscenza di questi risulta una limitazione più o meno considerevole, soprattutto quando si tratta di lavorare su codice scritto da terzi, di cui non si conosce né la semantica né i contenuti, sarebbe quindi auspicabile identificare una piattaforma il cui codice sia non solo accessibile, ma anche comprensibile.

- **Interazione modello-estensioni**
  Anche ammesso che il codice sia comprensibile e si riescano quindi a compilare con successo le estensioni, è ad ogni modo necessario che gli strumenti prodotti possano visualizzare e modificare ogni elemento del modello, così da poterlo elaborare. Di enorme importanza sarà, ad esempio, poter modificare i materiali degli elementi del modello per renderli trasparenti e produrre delle sezioni, cosa che si rende impossibile se non si ha la possibilità di accedere e modificare le proprietà degli oggetti.

Le piattaforme

Autodesk Forge

Forge è la piattaforma di sviluppo di Autodesk, casa produttrice di programmi largamente usati per la progettazione in campo ingegneristico e architettonico come AutoCAD e Revit.
La piattaforma si basa su tecnologia cloud, quindi tutte le elaborazioni grafiche (geometrie, texture, luci e simili) vengono effettuate in remoto su server Autodesk, che si occupano quindi di gestire la parte onerosa del processo e consentendo al dispositivo che viene utilizzato di preoccuparsi solo di visualizzare i dati che riceve dai server stessi.[1]
Essendo un servizio di una società specializzata in AEC, l'integrazione con gli altri programmi della stessa casa è estremamente facilitata grazie a processi e *plugin* nativi all'interno delle stesse, il che riduce drasticamente il quantitativo di lavoro necessario da questo punto di vista, infatti la compatibilità con IFC e l'interrogabilità del modello sono nativamente supportate.
Gli strumenti già presenti sono tanti e si integrano facilmente tra loro, ma allo stesso tempo risulta molto difficile creare ulteriori estensioni, soprattutto se queste richiedono una grossa interazione con il modello come nel caso di un progetto in divenire. In aggiunta, la documentazione in supporto all'API *(Application Programming Interface)* risulta completa per quanto riguarda l'utilizzo degli strumenti già presenti, ma piuttosto scarna quando si tratta della creazione di nuove estensioni.
Il tutto è inoltre complicato dalla gestione manuale delle autenticazioni ai server Autodesk: come detto precedentemente, la piattaforma si basa su tecnologia cloud, bisognerà quindi trasmettere i dati al server per far sì che questi vengano elaborati. Questa operazione purtroppo è resa molto laboriosa; è ad esempio necessario trasmettere il modello perché sia convertito in formato SVF per una corretta visualizzazione, e tutti questi passaggi intermedi di conversione, autenticazione e trasmissione sono da ripetersi ogni volta, gravando molto sull'esperienza d'uso.

Blender

Blender non è propriamente una piattaforma di sviluppo BIM, si tratta di un programma gratuito che si occupa principalmente di creazione e modellazione di oggetti 3D e di animazione degli stessi. Il motivo per cui è stato considerato è che Blender, essendo appunto gratuito e molto diffuso, dispone di una serie di plugin creati da utenti e società terze che estendono il programma, conferendogli nuove potenzialità e consentendo di raggiungere obiettivi che nativamente non sarebbero accessibili.
Con l'utilizzo di alcuni di questi plugin, infatti, risulta possibile importare file IFC in Blender e interrogare le caratteristiche del modello BIM senza difficoltà. Considerando poi la possibilità di creare

[1] La potenza di calcolo necessaria ad un'elaborazione grafica di alta qualità non è da ignorare: maggiore è il numero di elementi che compongono il modello e maggiore sarà il carico che il processore dovrà sopportare per portare a termine tutti i calcoli legati ad esempio ai rimbalzi della luce per calcolare le ombre. Poter affidare questo onere a dei server esterni riduce di molto il tempo di elaborazione e lo stress sul dispositivo locale, che dovrà solo ricevere i dati già elaborati e limitarsi a visualizzarli, consentendo ad esempio di elaborare il modello con buone prestazioni anche su dispositivi mobili.

ulteriori plugin personalizzati e le grandi capacità in termini di visualizzazione e rendering del programma, si è deciso di testarne l'utilizzo.
Anche l'interazione tra estensioni e modello non risulta un problema: il file IFC importato viene elaborato e caricato all'interno del programma come se fosse stato creato nel medesimo, tutti gli elementi, quindi, si rivelano accessibili ed editabili senza alcuna difficoltà.
Risulta tuttavia limitante il fatto che Blender permetta di lavorare solo all'interno della propria interfaccia, e non consenta perciò di creare ed esportare un eseguibile con tutti i contenuti al suo interno. Nell'ottica del progetto, risulterebbe invece ideale il poter produrre un programma *standalone*, ovvero un eseguibile senza dipendenze che sia dotato di una propria interfaccia e di un ambiente di lavoro autonomo, cosa che Blender, purtroppo, non permette.

Unity con Unity Reflect

La soluzione analizzata riguarda l'utilizzo della piattaforma Unity in associazione con il suo modulo aggiuntivo Unity Reflect. Come Blender, nemmeno Unity risulta nativamente una piattaforma BIM. Si tratta invece di una nota piattaforma di sviluppo per videogiochi, che mette a disposizione strumenti già molto collaudati sia in termini di gestione di modelli 3D anche molto complessi, sia in termini di creazione ed integrazione di componenti aggiuntivi. Essendo volta alla produzione di videogiochi infatti, la piattaforma mette a disposizione moltissimi strumenti per la creazione di contenuti personalizzati, e tramite codice è possibile interagire con tutti i componenti presenti all'interno del programma, siano essi nativi o importati.
Unity Reflect, invece, è un prodotto della famiglia di Unity orientato proprio alla gestione e allo sviluppo di modelli BIM: sfrutta la potenza degli strumenti nativi di Unity e li integra con le necessità che costellano la gestione di un progetto BIM. Nasce come rivolto principalmente alla visualizzazione dei modelli, anche tramite l'utilizzo di tecnologie AR e VR, ma grazie alla sua versione Develop consente di ampliare il proprio codice sorgente in modo da consentire una personalizzazione il più completa possibile dell'applicativo. Reflect si interfaccia tramite plugin con vari programmi e piattaforme di modellazione BIM, tra cui Revit e ArchiCAD (ma non solo), dai quali può esportare i modelli per caricarli nel cloud associandoli all'account dell'utente, da dove verranno poi trasferiti in Unity. Tramite questi plugin si raggiunge compatibilità completa sia con i formati proprietari delle varie case produttrici (quelle per cui è disponibile il plugin) sia con il formato IFC.
Il risultato è una piattaforma estremamente versatile, in grado di collaborare con altri programmi professionali del settore AEC, che consente un facile sviluppo di nuovi contenuti e con strumenti nativi che facilitano molto l'esperienza d'uso sia nella programmazione che nella fruizione.

Confronto

	Compatibilità IFC	Modello interrogabile	Facilità di ampliamento	Interazione modello-estensioni
Autodesk Forge	●●○	●●●	○○○	○○○
Blender	●●○	●●○	●●○	●●○
Unity Reflect	●●●	●●●	●●○	●●○

TAB. 1: Valutazione dei requisiti delle piattaforme analizzate.

Esito

Analizzando per ciascuna piattaforma le caratteristiche principali richieste per lo svolgimento del progetto, è possibile notare come la **compatibilità con i formati IFC** sia tutto sommato buona per ognuna di esse, sebbene alcune debbano appoggiarsi a plugin (di terze parti per quanto riguarda Blender, interni per quanto riguarda Unity Reflect).
Analogamente, anche la possibilità di **interrogare il modello** risulta soddisfacente per ogni programma preso in esame, prosegue infatti spesso di pari passo con la compatibilità dei formati BIM.

La situazione risulta diversa invece quando si considera la praticità nello **scrivere estensioni personalizzate** per la piattaforma, e come queste possono interagire con il modello caricato: Forge, per quanto sia nativamente orientato alla visualizzazione di modelli BIM, risulta molto complesso sia nella scrittura delle

estensioni (che devono dialogare con i server Autodesk e usano NodeJS come linguaggio), sia nell'**interazione col modello**, che una volta caricato risulta essere più un'entità a parte e singola, piuttosto che un insieme di elementi interagibili e modificabili, come sarebbe invece auspicabile.

Per Blender la situazione è leggermente migliore, le estensioni sono più facili da scrivere, non richiedono autenticazioni di sorta e vengono scritte in Python. Purtroppo la documentazione risulta molto scarsa poiché la compatibilità con IFC deriva da un plugin di terze parti, senza contare che Blender non consente l'esportazione di un eseguibile; tutto il lavoro sarebbe quindi utilizzabile solo all'interno di Blender stesso.

Con Unity la situazione risulta ancora più agile: le estensioni sono di facile scrittura (in C#) e tutto sommato si integrano bene anche con il modello, una volta capito come accedere agli elementi importati. La documentazione non è molto dettagliata, ma le comodità che la piattaforma fornisce consentono di non soffrire troppo questa mancanza. Considerando poi che Unity nasce per lo sviluppo di applicativi fortemente interconnessi, risulta molto più comodo intraprendere uno sviluppo così fortemente orientato alla programmazione su questo tipo di piattaforma.

Considerate tutte le caratteristiche riassunte sopra, si è optato per l'utilizzo di Unity Reflect per lo sviluppo di questo progetto. L'ampia personalizzazione che la piattaforma mette a disposizione risulta essere senza dubbio il suo punto di forza, e l'interazione nativa con le piattaforme AEC grazie al plugin messo a disposizione, si prepone di essere un ottimo punto di partenza.
Nonostante gli strumenti presenti consentano solo di visualizzare e interrogare superficialmente il modello, la possibilità di scrivere e integrare nuovi componenti che permettano di raggiungere gli obiettivi sopra esposti sembra promettente.

OBIETTIVI ED IMPLEMENTAZIONI IN UNITY REFLECT

Considerata la presenza dei vari plugin sviluppati per i principali programmi AEC, si è considerato di lavorare partendo direttamente dall'ambiente di lavoro (in questo caso su Revit) piuttosto che partire direttamente con un file completo ed esportato in IFC: questa scelta è stata adottata per permettere di ottimizzare un flusso di lavoro che coinvolga gli storici anche nella fase dello sviluppo del progetto, e non solo una volta che questo abbia raggiunto le fasi finali.

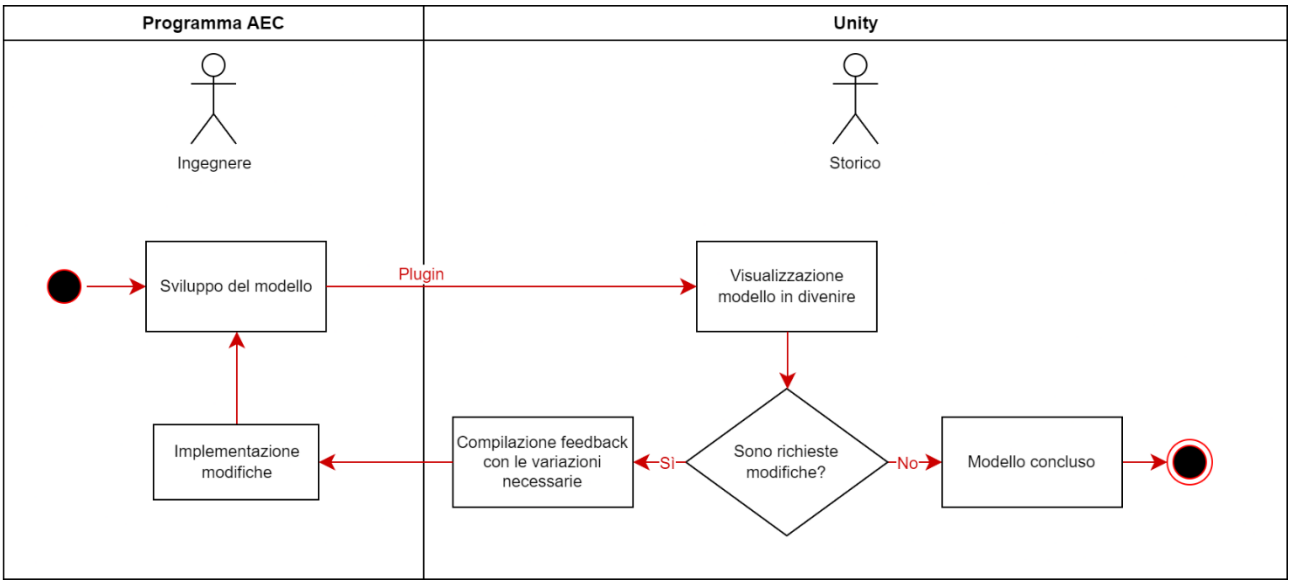

FIG. 1: Presentazione del workflow per lo sviluppo del modello in collaborazione.

Lo sviluppo del modello in collaborazione con gli storici comincia dal programma scelto per la virtualizzazione della struttura, per questo progetto si è lavorato con Revit; una volta installato il plugin relativo, sarà sufficiente fare l'accesso al proprio account di Unity per poter trasferire il modello.
Una volta importato, l'interfaccia di default consente una completa visualizzazione 3D del modello, più tutta una serie di strumenti che consentono un'analisi iniziale del progetto.
La piattaforma, tuttavia, presenta mancanze e limitazioni rispetto agli obiettivi prefissati, che andranno quindi integrati con aggiunte ad hoc.

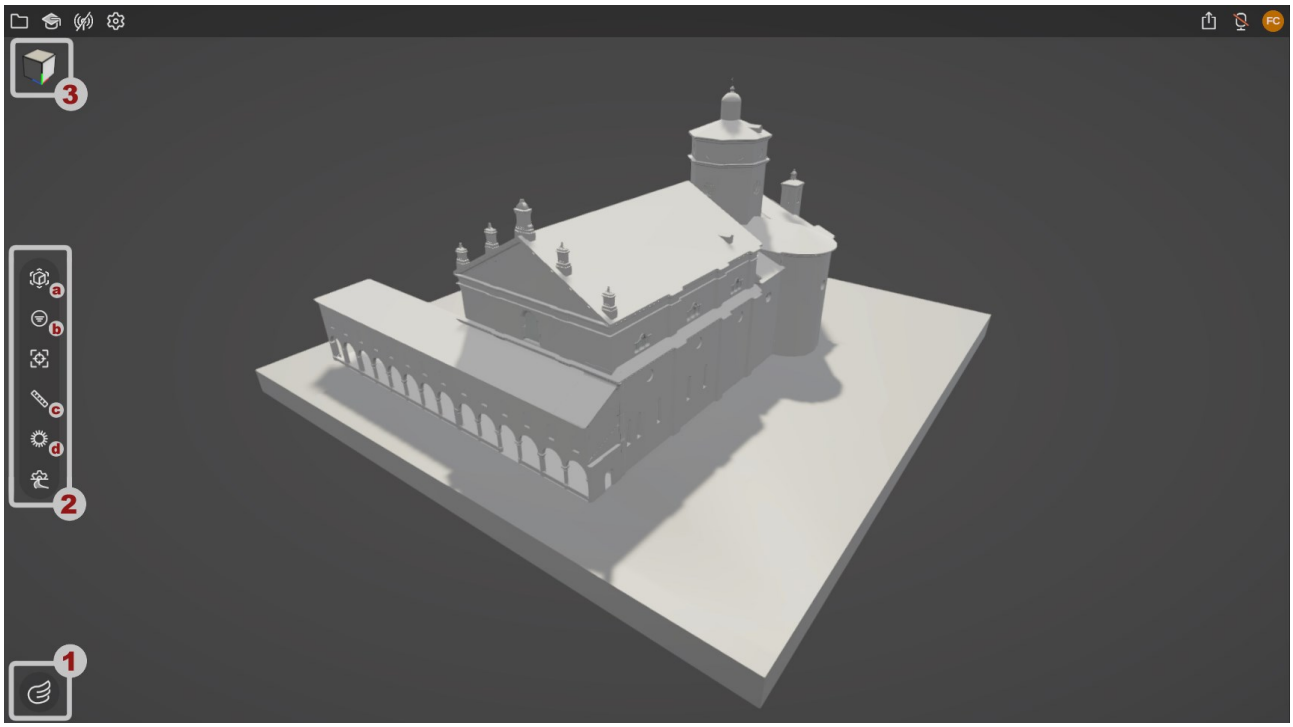

FIG. 2: Interfaccia originaria di Unity Reflect

Facendo riferimento alla Fig. 2, i principali strumenti nativamente presenti in Unity Reflect sono:

1. Modalità di movimento della telecamera
2. Barra degli strumenti
    a. Interrogazione del modello
    b. Filtro degli elementi
    c. Misura delle distanze
    d. Studio solare
3. Gizmo della telecamera

Integrazioni necessarie

Sezioni, Piante e Fasi storiche

Sebbene la visualizzazione del modello tridimensionale sia il punto di partenza all'interno di Unity Reflect, non è da trascurare la grande utilità che le viste bidimensionali portano in fase di analisi di un progetto: la consultazione di piante e sezioni, soprattutto se considerata la necessità di fare dei confronti tra lo stato di fatto e le rappresentazioni storiche che ci sono giunte, è fondamentale.
L'unico modo che Unity Reflect presenta per importare queste viste all'interno del programma però, è l'esportazione di un modello debitamente sezionato: se si desidera ad esempio vedere una sezione trasversale del modello, sarà necessario regolare come desiderato la *regione di taglio* in Revit, ed esportare la vista 3D che ne risulta, tramite il solito plugin; questo è principalmente dovuto al fatto che è possibile esportare tra i due programmi solo elementi che è possibile convertire in *mesh*, elementi dotati quindi di una geometria spaziale tridimensionale. Il diretto risultato è che per ogni sezione o pianta ritenuta

rilevante per l'analisi del progetto andrà esportata una vista tridimensionale, che a sua volta genererà un modello a parte: questa catena di operazioni, oltre che lunga e laboriosa, risulta molto pesante in quanto in fase di caricamento del modello, bisognerà caricare tutti i modelli così generati, cosa che allunga di molto i tempi di attesa, nonché le prestazioni del programma così appesantito, senza contare che Unity visualizzerà tutti i modelli così esportati contemporaneamente, rendendoli praticamente indistinguibili (vedi FIG. 3). Sarà necessario predisporre uno strumento atto a riprodurre piante e sezioni in Unity senza dover per forza esportarle direttamente dall'ambiente di sviluppo in Revit, in modo da soprassedere alla creazione e caricamento di tutti i modelli di cui sopra.

Parallelamente è invece necessario poter visualizzare in Unity, separatamente, i modelli connessi alle fasi storiche, ovvero dei modelli che rappresentano sempre la struttura in analisi, ma che rappresentano lo stato di fatto in un preciso momento della storia: questo strumento così importante per lo studio delle modifiche nel corso del tempo non è previsto nella versione di rilascio, e sarà quindi obiettivo di questo progetto.

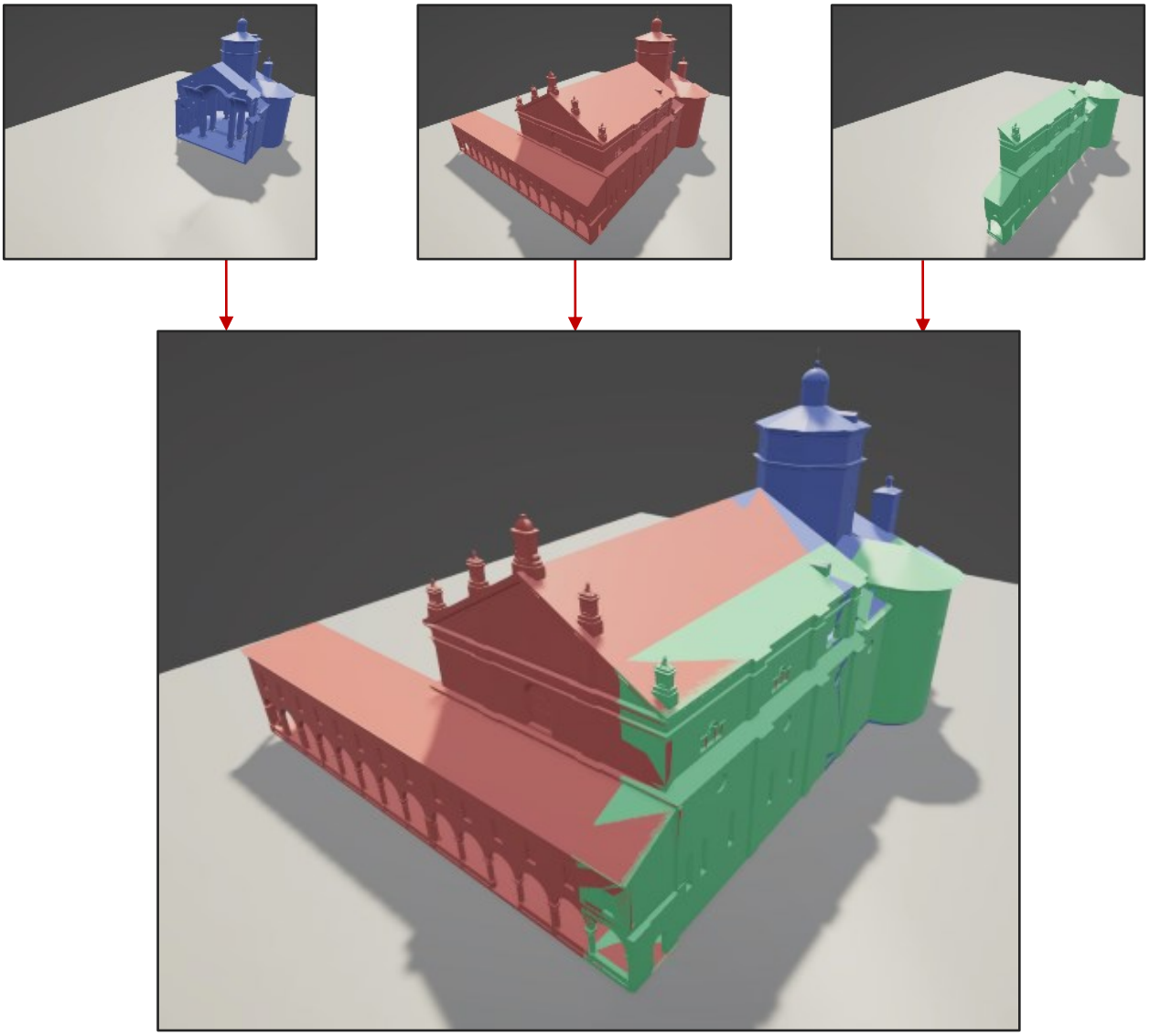

FIG. 3: La visualizzazione di default di Unity rappresenta più viste contemporaneamente

Livelli

Come specificato in precedenza, tutti quegli elementi che non sono dotati di una geometria tridimensionale vengono ignorati in fase di esportazione verso Unity: questa chiaramente risulta essere una limitazione, perché alcuni di questi dati sono fondamentali anche in fase di visualizzazione del modello e di analisi dello stesso, primi fra tutti gli elementi della scomposizione spaziale, considerato inoltre che non è possibile modificare in alcun modo i plugin di esportazione, sarà necessario trovare una via alternativa per importare tutti questi oggetti all'interno di Unity.
I livelli, ad esempio, risultano particolarmente interessanti quando si tratta di valutare le quote degli elementi o come riferimenti per spaccati e sezioni.

Immagini

Considerato l'intento di facilitare l'utilizzo da parte dei professionisti in campo storico e artistico, si è deciso di implementare la possibilità di visualizzare immagini all'interno del modello 3D: questo strumento risulta infatti molto utile quando si tratta di fare comparazioni tra il modello sviluppato e ad esempio delle piante storiche, o anche per simulare la presenza di quadri o altri elementi decorativi all'interno dell'ambiente virtuale.
Per procedere con questo tipo di implementazione, bisognerà innanzitutto inserire nel programma di modellazione il riferimento all'immagine desiderata: nel caso di Revit, si utilizzeranno quindi le decalcomanie. Analogamente a quanto si verifica per i livelli, anche le rappresentazioni bidimensionali vengono ignorate in fase di esportazione, ma costituiscono una problematica aggiuntiva in quanto contengono riferimenti al file che rappresentano, solitamente una foto o una pagina PDF. Quando queste verranno quindi importate in Unity bisogna assicurarsi che i riferimenti a questi file siano mantenuti o in altro modo recuperati.
A differenza dei livelli, che sono posizionati nello spazio in riferimento alla loro quota altimetrica, nell'esportazione delle immagini bisogna prestare attenzione anche alla posizione del piano su cui sono riferite (verticale o orizzontale) e alla dimensione per mantenere la coerenza di scala: se l'immagine in questione è una pianta, sarà infatti fondamentale assicurarsi che tutti i vari allineamenti siano mantenuti anche in Unity.

Annotazioni

Una delle interessanti funzionalità di Unity Reflect è anche quella di poter gestire commenti e annotazioni da parte di tutti gli utenti che hanno accesso al progetto per ogni singolo elemento presente. È infatti possibile in fase di interrogazione del modello inserire commenti o revisioni per far sapere agli altri membri se ci sono modifiche necessarie all'elemento in questione. Queste annotazioni sono visibili ai vari utenti ma è anche possibile direzionarle verso una persona specifica. È possibile infine visualizzare il progetto in più membri, ciascuno dal proprio dispositivo, e allinearsi alla visuale di un utente specifico in modo da seguire facilmente eventuali presentazioni.

Soluzioni proposte

Section box

Il primo degli strumenti proposti come soluzione alle problematiche di cui sopra, consiste in un cubo circoscritto al modello, le cui facce, muovendosi indipendentemente l'una dall'altra, fungano da piani di sezione per il modello stesso.
L'idea è quella di un parallelepipedo di taglio (*section box)* che consenta di riprodurre le viste che, come riscontrato sopra, non conviene esportare ma che risultano sempre uno strumento molto utile.

Step 1: Esportare i livelli di Revit

Con questo stesso strumento sarà possibile ricreare le piante dei vari piani, risulta quindi molto importante poter avere a disposizione tutte le varie informazioni che riguardano i livelli presenti all'interno del modello.
Prendendo come punto di partenza il modello in Revit, tramite l'utilizzo di Dynamo (un plugin di programmazione visiva che permette di elaborarne i contenuti) si è proceduto a creare delle piccole piattaforme in corrispondenza di tutti i livelli ritenuti rilevanti (contrassegnati dalla proprietà "piano edificio")[2]. A questi nuovi elementi tridimensionali è stato poi inserito all'interno dei parametri dell'oggetto stesso l'altezza in centimetri del livello associato: i nuovi oggetti hanno quindi tutte le caratteristiche necessarie per essere esportati in Unity portando con sé tutte le informazioni importanti[3].

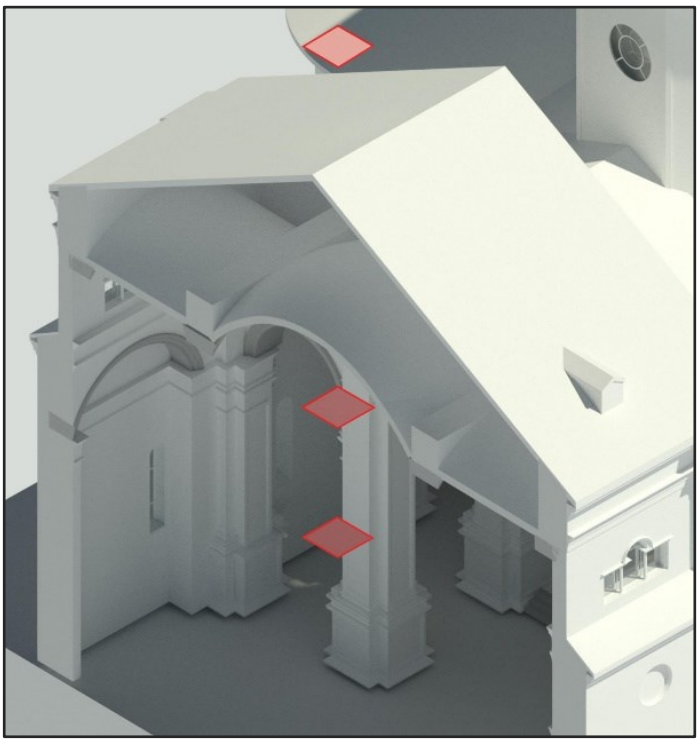

FIG. 4: Le piattaforme create come riferimento dei livelli in Revit

[2] Vedi paragrafo A1.1 per il codice relativo.
[3] Vedi paragrafo A1.2 e A1.3 per il codice relativo.

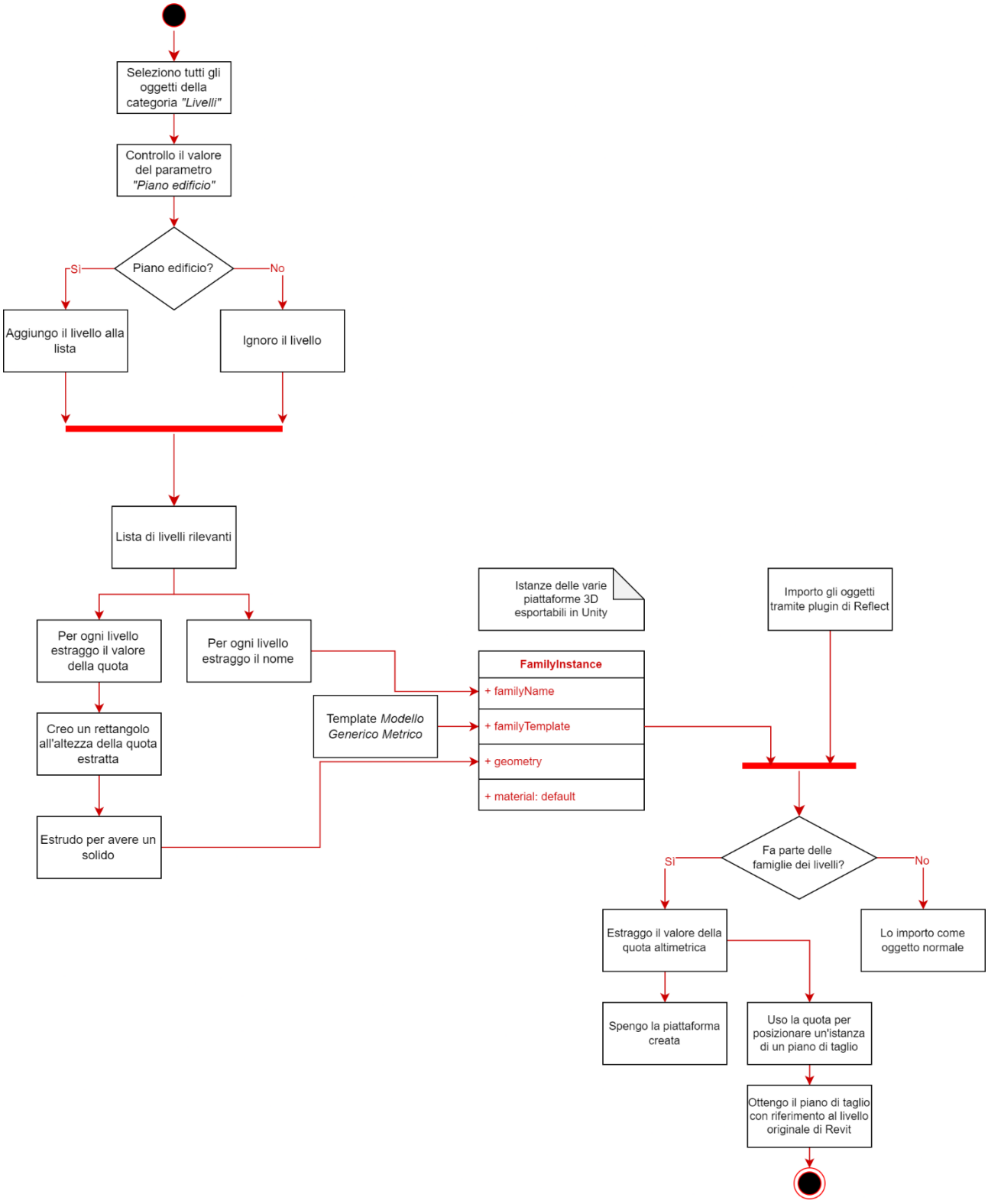

FIG. 5: Esportazione dei livelli da Revit a Unity

Step 2: Modificare una mesh in runtime

Quando un oggetto viene creato o importato in Unity, questo viene considerato come entità complessa di cui la mesh (il reticolo che ne definisce la geometria) è soltanto una delle proprietà che lo caratterizzano. Inoltre, non essendo la modellazione 3D la caratteristica principale di Unity, solitamente non è necessario modificare la struttura delle mesh, tantomeno in fase di esecuzione dell'applicativo (*runtime*).

Tuttavia, per poter sezionare il modello in corrispondenza delle varie facce della *section box* così come ci si è prefissato, sarà necessario dover ricostruire la posizione delle facce stesse in maniera indipendente l'una dalle altre: nativamente, essendo il cubo un unico oggetto, Unity riconoscerà come posizione solamente il centro geometrico dello stesso, dovremmo quindi andare a lavorare non tanto sul cubo in qualità di oggetto, quanto sulla sua proprietà mesh[4].

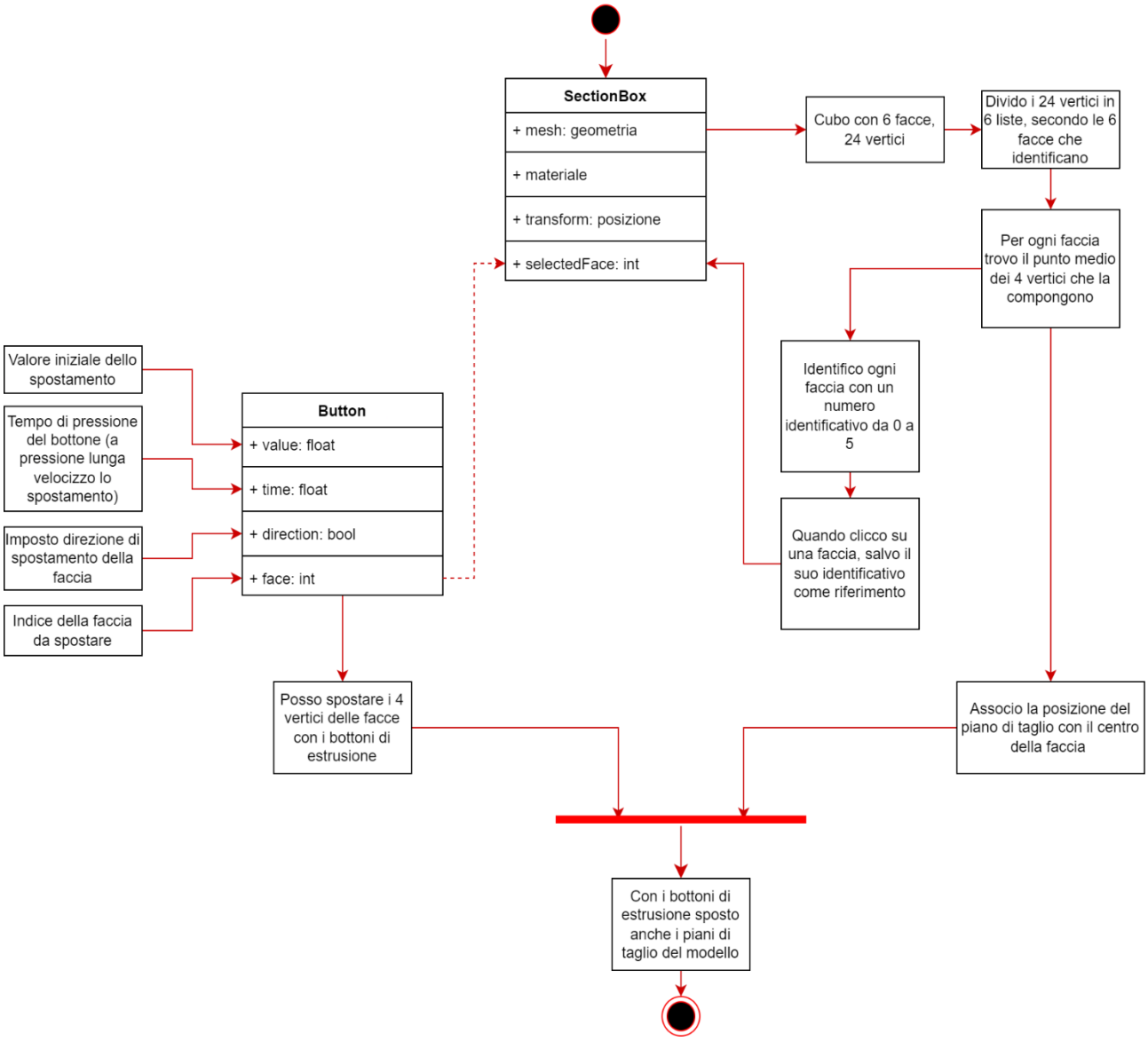

FIG. 6: Costruzione della Section Box

[4] Vedi paragrafo A2.1, A2.2 e A2.3 per il codice relativo.

Step 3: Creare uno shader personalizzato

Una volta che sono stati creati i riferimenti alle posizioni delle facce della Section Box, è necessario far sì che il modello importato reagisca ai rispettivi piani di taglio e diventi quindi invisibile quando si trova all'esterno della scatola. Per raggiungere questo risultato occorre necessariamente creare uno *shader* (un algoritmo che conferisca al materiale delle proprietà che possono reagire con altri componenti), nello specifico lo shader dovrà supportare come input le posizioni dei vari piani di sezione a impostare l'opacità del materiale a trasparente quando i vari elementi sono esterni o anche solo tagliati dai piani[5].

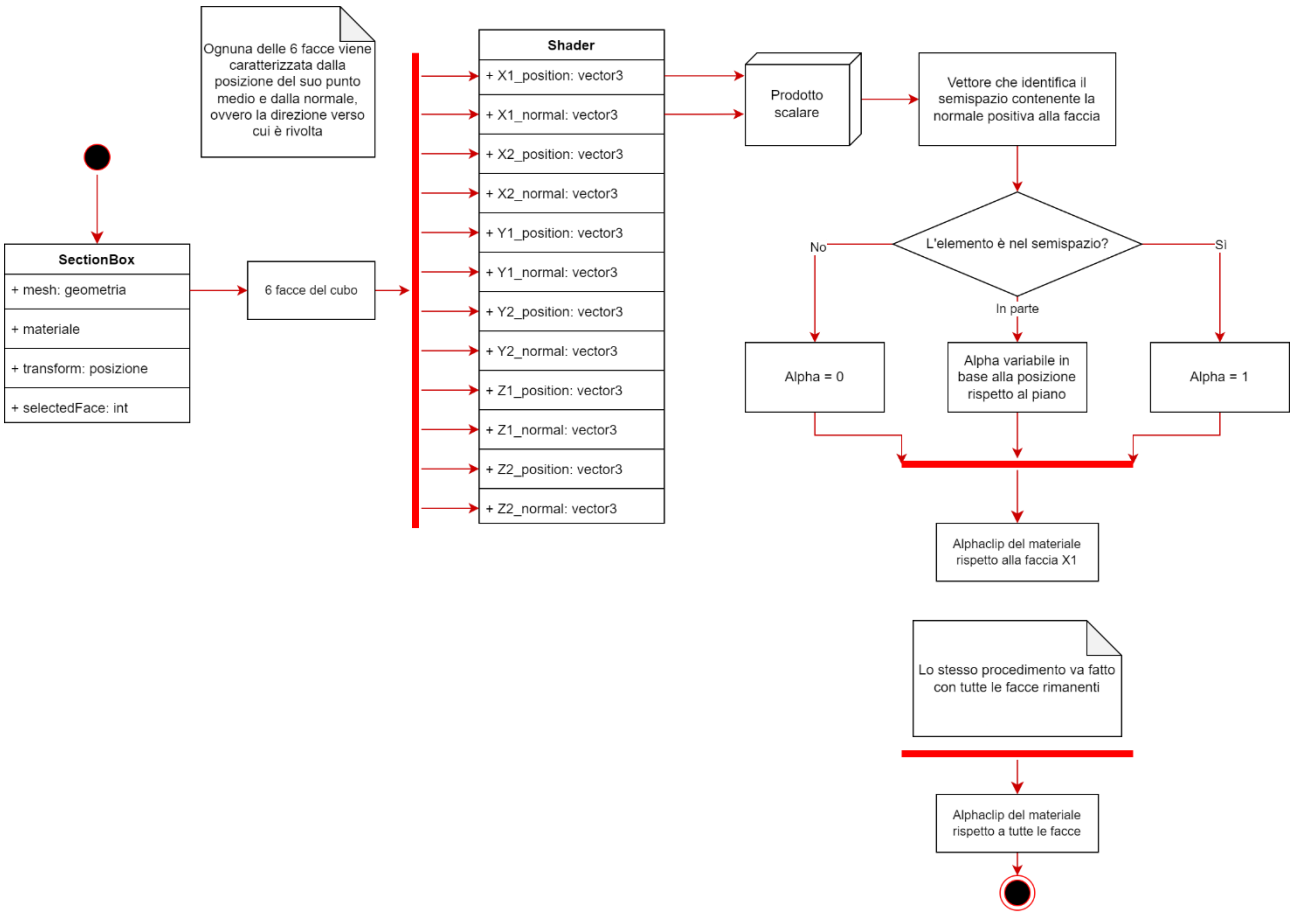

FIG. 7: Funzionamento dello shader personalizzato con riferimenti alle facce della Section Box

Step 4: Coordinare i modelli con la Section Box

Lo shader così costruito tuttavia non è sufficiente a raggiungere l'obiettivo per cui è stato creato: lo shader, infatti, serve a conferire determinate proprietà ad un materiale, bisognerà quindi associare ad ogni elemento del modello un materiale cui sia collegato lo shader appena creato.

Questa operazione sarebbe di per sé diretta e triviale, se non fosse per il fatto che il modello non esiste fino a quando non viene effettivamente importato nel programma, solo allora infatti si creano in Unity tutti gli oggetti e gli elementi del modello: è quindi necessario scrivere del codice che si occupi di assegnare

[5] Vedi paragrafo A2.4 per il codice relativo.

a tutti gli elementi importati il nuovo materiale associato allo shader, più tutta una serie di script che consentono la corretta interazione con i piani di taglio[6].

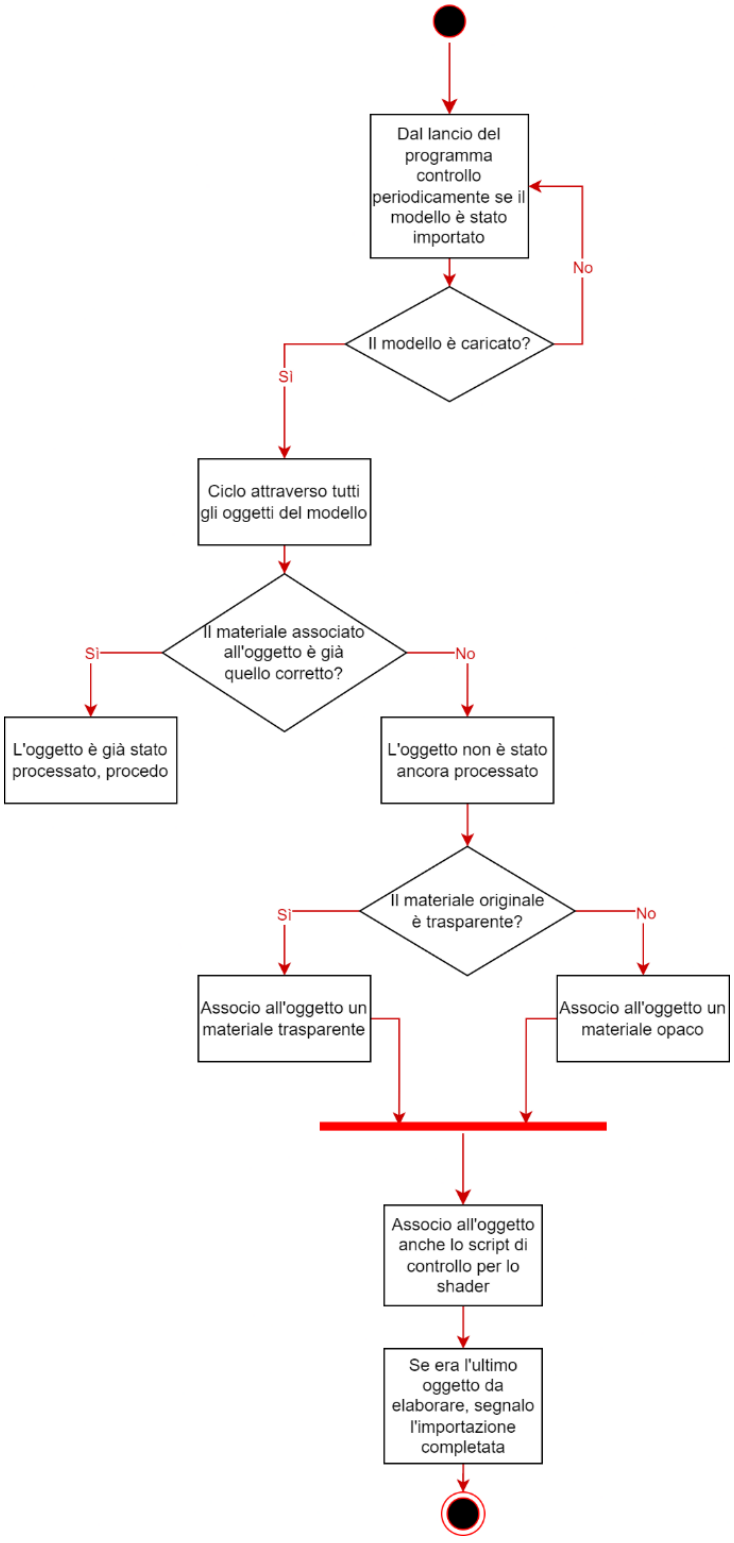

FIG. 8: Associazione del materiale e dello script di controllo

[6] Vedi paragrafo A2.5 e A2.6 per il codice di riferimento.

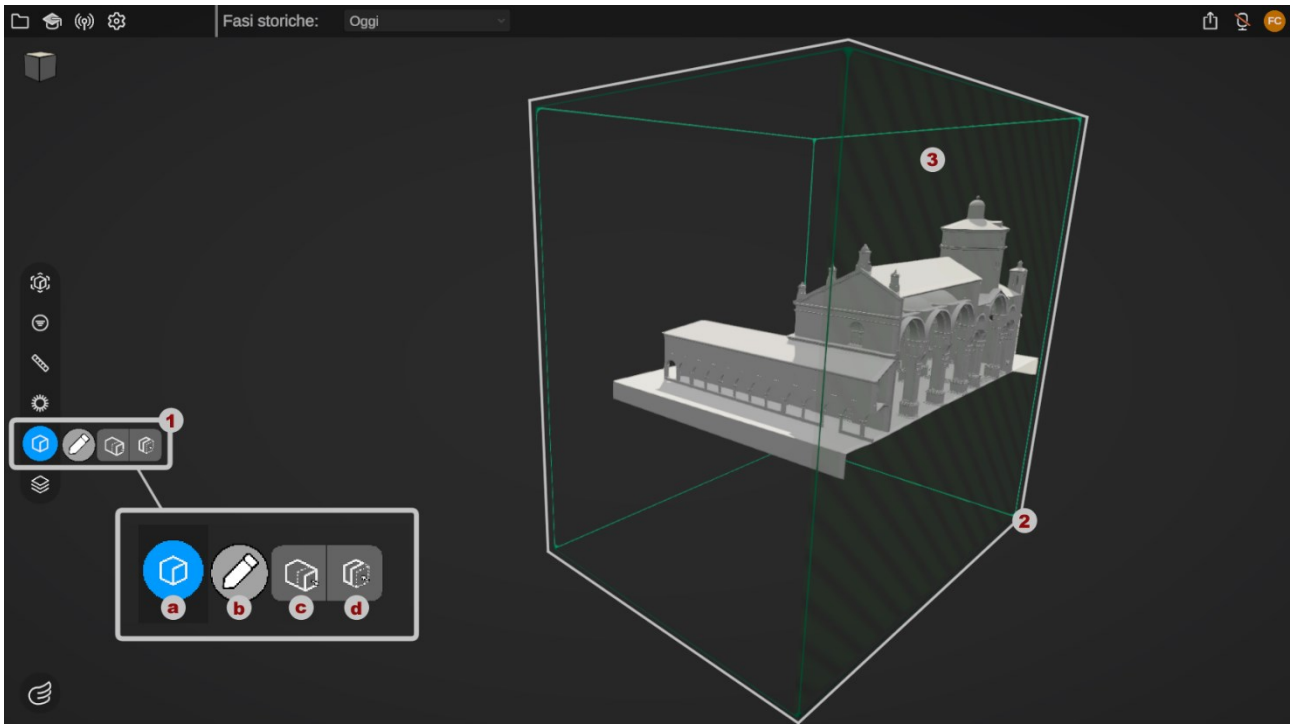

FIG. 9: Interfaccia di controllo della Section Box

La FIG. 9 mostra l'interfaccia di controllo della Section Box (1) e come questa si presenta nella scena (2): una volta attivata la modalità sezione con l'apposito pulsante (a), sarà possibile visionare il modello con le sezioni applicate. Per modificare la posizione dei piani di sezione sarà necessario abilitare la modifica (b) e selezionare il piano che si desidera spostare cliccando la relativa faccia del cubo di sezione: questa presenterà un retino in trasparenza (3) per indicare il piano attivo, che sarà possibile spostare tramite gli appositi pulsanti (c, d) verso l'interno o l'esterno, secondo quanto desiderato.

Step 5: Scorciatoie per piante di livello

Si è ritenuto utile predisporre delle scorciatoie per replicare in maniera veloce le sezioni ai livelli precedentemente esportati da Revit: una volta attivati (vedi FIG. 10, elemento 1), sarà quindi possibile visualizzare graficamente tutti i livelli di cui sopra (2), cui è stato aggiunto un pulsante che ne riporti il nome relativo (3) e che una volta cliccato si occupi di posizionare la Section Box in maniera tale da riprodurre una sezione orizzontale all'altezza del livello scelto.

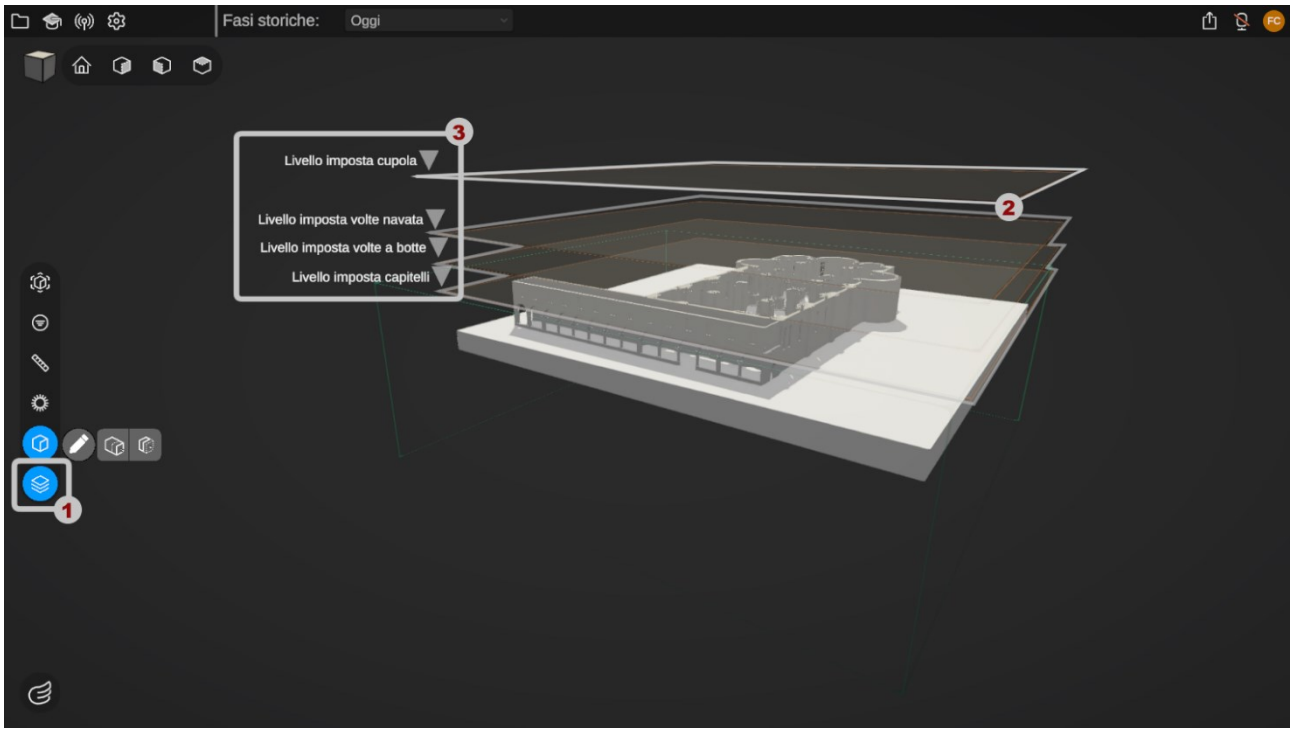

FIG. 10: Scorciatoie per sezioni ai livelli importati

Decalcomanie

Come da premessa, le immagini sono affette dallo stesso problema che è stato affrontato con i livelli: non sono infatti considerate in fase di esportazione dal plugin di Reflect, e non vengono di conseguenza importate in Unity. La soluzione proposta è in parte simile a quella adottata per i livelli, ma con la complicazione aggiunta della texture dell'immagine.
Nonostante Revit consenta in vari modi l'inserimento di immagini all'interno del progetto, si è usato quello tramite decalcomanie in quanto più comodo per i fini di questo progetto.

Intro: La necessità di un CDE

Il progetto è stato sviluppato con la visione di dover lavorare con membri da remoto, non collegati quindi ad una rete locale, ma che avessero comunque accesso a tutti gli strumenti e le risorse necessarie. Considerata inoltre la necessità di trasferire ora immagini tra Revit e Unity (non potendo lavorare con il percorso locale per quanto appena detto), si rende palese la necessità di lavorare con un *Common Data Environment* (CDE), ovvero un ambiente virtuale in cui tutti gli utenti abbiano accesso ai dati caricati in un server remoto. Si è proceduto quindi assumendo l'esistenza di questo CDE, in modo da poter avere riferimenti comuni ai file a prescindere da quale utente ne avesse bisogno.

Step 1: Ricavare le informazioni con Dynamo

A differenza dei livelli, per esportare le decalcomanie con le immagini in Unity, avremo bisogno di esportare più informazioni: quelle geometriche, nello specifico posizione e dimensione, e quelle visive, bisognerà infatti esportare l'immagine come texture in maniera che Unity possa poi visualizzarla nel

modello. Tuttavia, il plugin di Reflect non esporta informazioni come le immagini, ma solo la geometria 3D, è necessario quindi inserire un riferimento esplicito al file, e fare in modo che Unity afferisca direttamente al file stesso. Per limitazioni di Revit, non è possibile accedere da codice direttamente all'immagine, andrà quindi riportato manualmente il link del CDE dell'immagine considerata in uno dei parametri della decalcomania.[7]

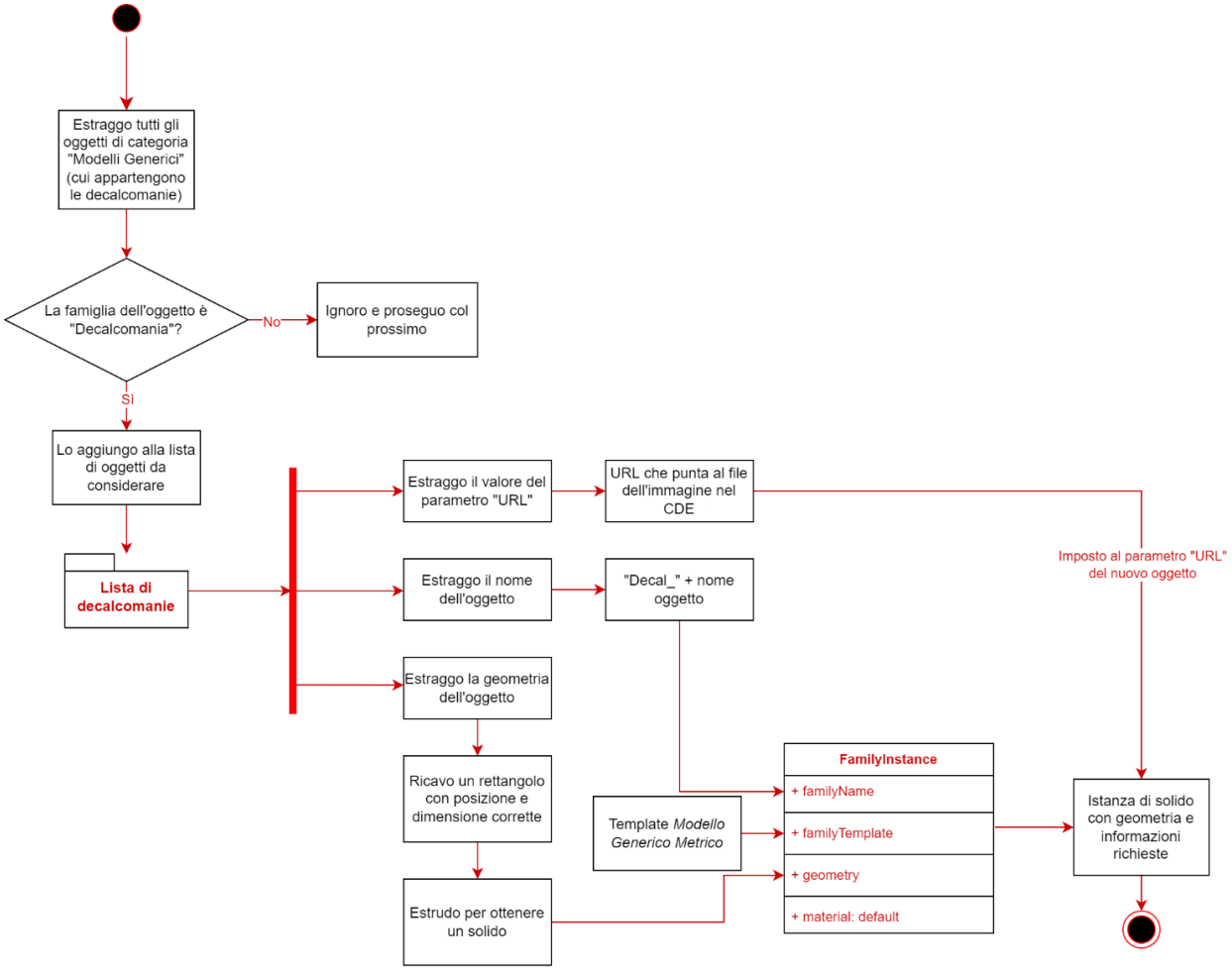

FIG. 11: Creazione delle geometrie 3D per l'esportazione delle decalcomanie

Step 2: Importare correttamente le immagini in Unity

Ora che ogni decalcomania è stata convertita in un oggetto 3D con tutte le informazioni necessarie, è necessario che questo venga importato correttamente in Unity, e che vi venga poi associata l'immagine relativa come texture del materiale[8]. Si fa presente subito il problema del posizionamento degli UV: si tratta in pratica delle coordinate che correlano la posizione della texture rispetto alla mesh su cui va applicata: le impostazioni di default non sono ottimali per il risultato desiderato, vanno infatti a posizionare ripetutamente l'immagine sulla superficie, fino a riempirla in maniera simile ad un mosaico.

[7] Vedi paragrafo A3.1 per il codice relativo.

[8] Vedi paragrafo A3.2 per il codice relativo.

In fase di importazione bisognerà quindi, una volta applicata l'immagine all'oggetto, impostare gli UV in maniera tale che l'immagine copra tutta la superficie disponibile, che sarà importata già delle dimensioni appropriate[9].

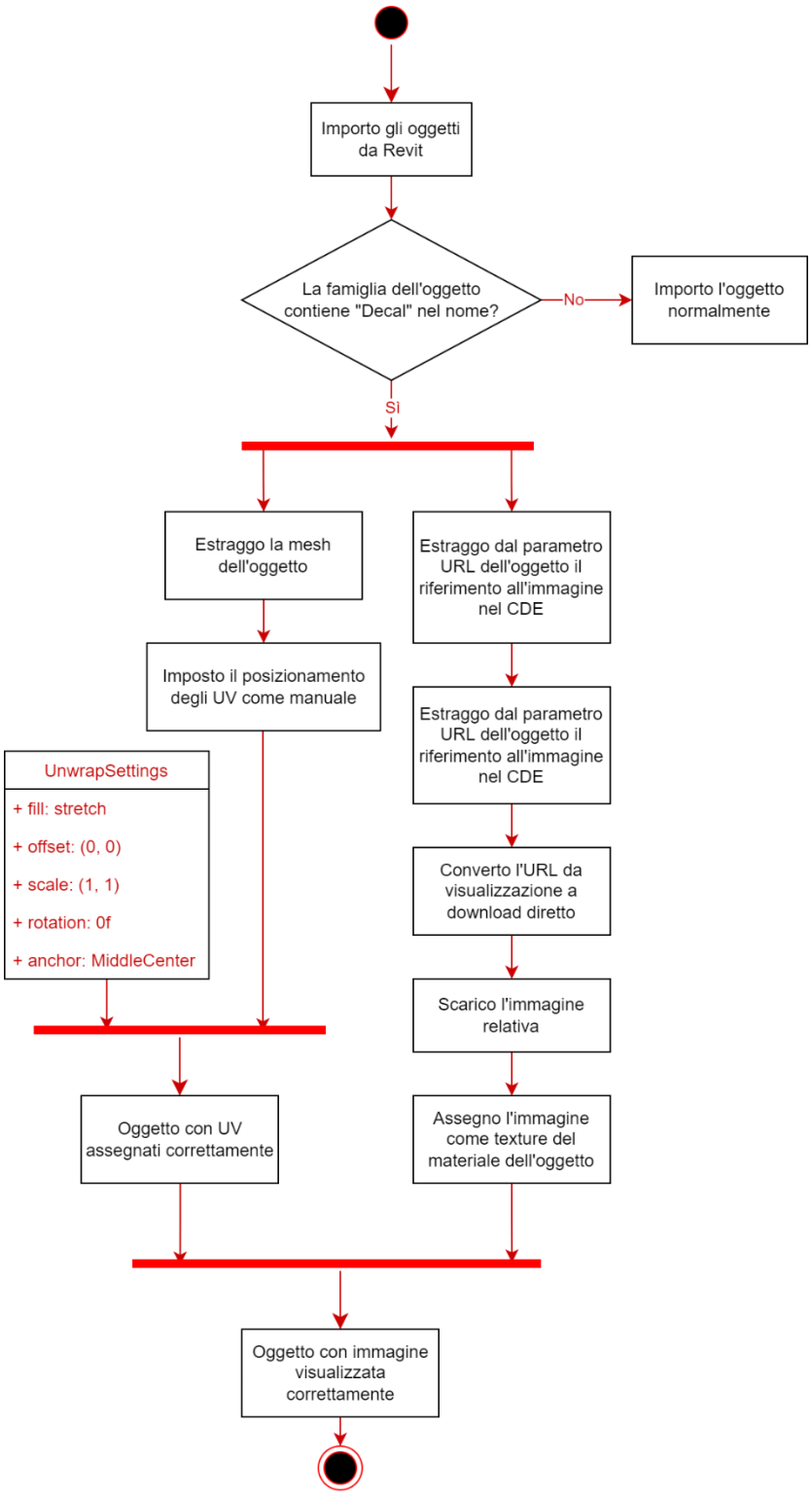

FIG. 12: Importazione corretta delle immagini in Unity

[9] Vedi paragrafo A3.3 per il codice relativo.

Funzioni aggiuntive

Sono state aggiunte o modificate alcune caratteristiche di importanza secondaria, che potessero tuttavia migliorare l'esperienza di utilizzo da parte degli utenti in maniera più o meno impattante.

Caricamenti

Considerati i lunghi tempi che il caricamento del modello può comportare (che si moltiplicano ulteriormente per ogni fase storica o vista esportata oltre a quella di base), si è ritenuto opportuno aggiungere un avviso per l'utente, da visualizzare durante le fasi di streaming del modello dai server remoti. Si è quindi provveduto alla creazione di una semplice schermata da visualizzare durante i caricamenti per dare all'utente un riscontro di ciò che il programma sta computando durante i tempi morti.

Viste ortogonali

L'interfaccia presenta il classico cubo di visualizzazione caratteristico dei programmi di visualizzazione 3D, interagendo con il quale è possibile accedere a delle scorciatoie che posizionano la videocamera perpendicolarmente alle varie facce del cubo: si simulano così la visualizzazione zenitale e i vari prospetti del modello, ma di default le impostazioni della videocamera raffigurano il modello con visuale prospettica, che non risulta adatta a questo tipo di inquadrature.
Andando quindi ad interagire con la videocamera stessa, si è proceduto a costruire un semplice script che cambi la tipologia di visualizzazione della videocamera, da prospettica a ortogonale, quando questa ha un asse di visualizzazione perfettamente verticale o orizzontale, quindi nei casi accessibili con le scorciatoie menzionate sopra.

Personalizzazioni aggiuntive

Essendo tutte le modifiche apportate create ad-hoc per il progetto, è possibile apportare varie modifiche sia agli elementi che alle interfacce.
Sarebbe possibile modificare il colore del materiale base del modello qualora questo venisse richiesto, o addirittura andare a differenziare il colore dei vari elementi o applicare dei pattern sulla base di un particolare tipo di filtro, ad esempio per distinguere elementi con un determinato tipo di finitura, o in base al periodo storico di costruzione.
Analogamente, è possibile apportare modifiche alla modalità di visualizzazione delle decalcomanie, e ad esempio creare un bottone che ne gestisca velocemente accensione e spegnimento, o visualizzarle in trasparenza.

Fasi storiche

Come menzionato precedentemente, è stato ritenuto opportuno aggiungere altri strumenti che potenziassero ulteriormente la piattaforma, fornendo un valore aggiunto che potesse facilitare ulteriormente il lavoro degli storici.
Si è considerato come, ad esempio, i software AEC dispongono di un sistema per la classificazione delle fasi costruttive e demolitive che risulta fondamentale per la ricostruzione del susseguirsi delle modifiche che l'edificio ha subito nel corso del tempo: si rende quindi necessario il poter visualizzare, all'interno dello stesso modello, le varie fasi costruttive individuate, e proporre un modo per poter passare dall'una all'altra in modo agevole.
Il modo migliore per esportare quelli che sono a tutti gli effetti altri modelli, prevede di caricare la vista 3D relativa alla fase storica ricercata ed esportarla con il plugin all'interno del progetto principale: verrà in questo modo importata in Unity come modello separato, che tuttavia, come visto analizzando il problema delle viste, verrà caricato contemporaneamente a quello già presente.[10]

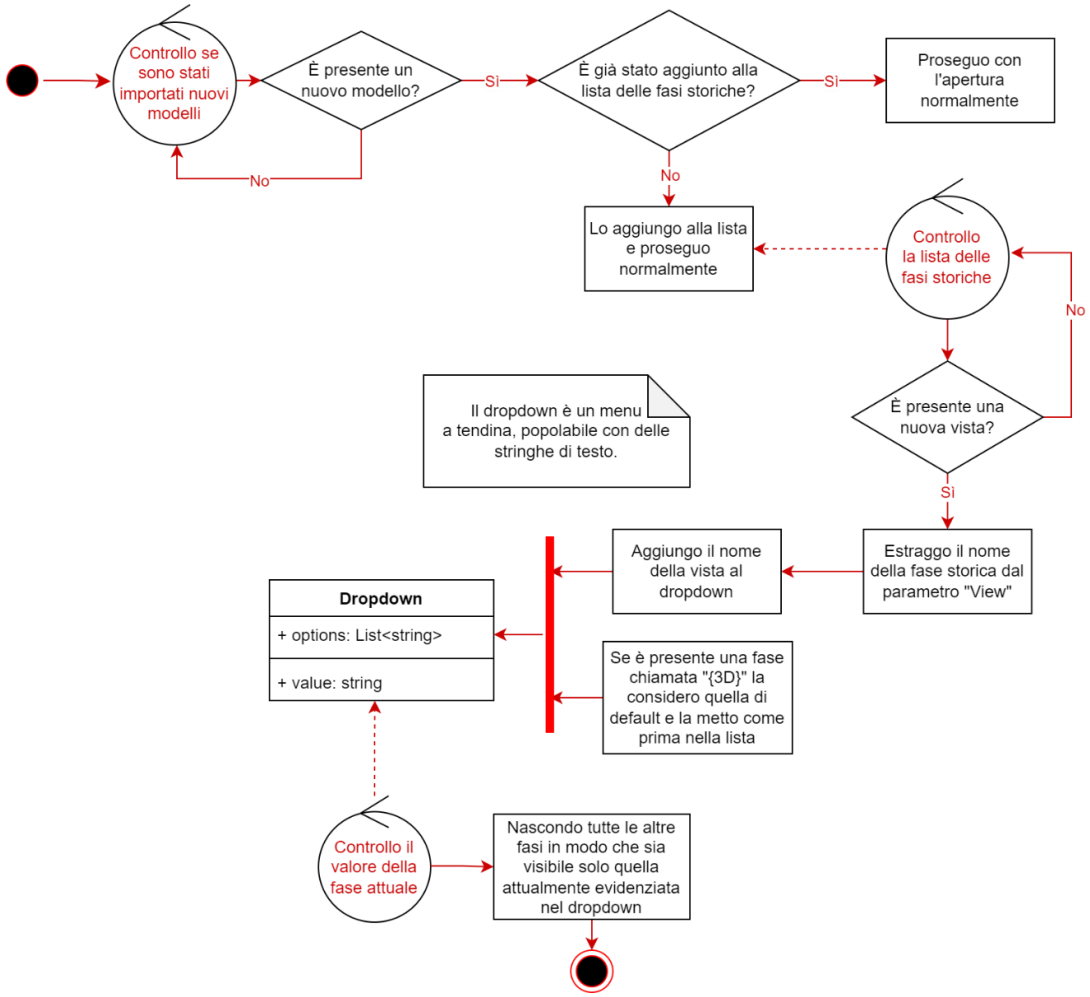

FIG. 13: Gestione delle fasi storiche e della loro visualizzazione

[10] Vedi paragrafo A4.1 per il codice relativo.

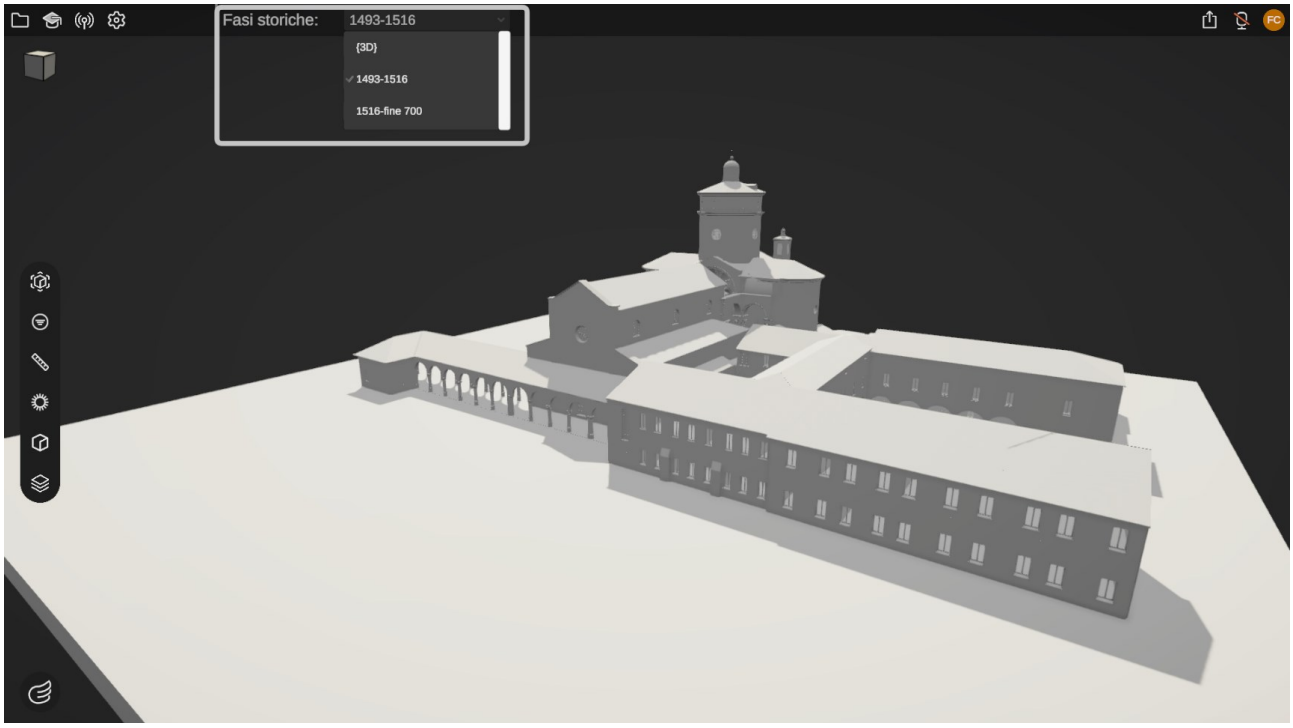

FIG. 14: Menu a tendina con le fasi storiche presenti nel progetto

Miglioramenti annunciati

Come dichiarato prima, la struttura base di Unity lo rende una piattaforma estremamente versatile e facilmente modificabile e migliorabile qualora si presentasse la necessità.
Con l'attuale versione in distribuzione di Reflect Develop, ad esempio, non è ancora stato aggiunto lo strumento che gestisce le annotazioni agli elementi, che purtroppo non è possibile sviluppare privatamente in questa sede, in quanto le suddette annotazioni vanno salvate obbligatoriamente sui server di Reflect, che provvedono a caricarle e a sincronizzarle sui modelli di tutti gli utenti del progetto.
Questa feature, già presente nella versione Reflect Review, sarebbe un importante strumento aggiuntivo di coordinamento tra gli utenti, ed è stato dichiarato che verrà inclusa nella prossima versione di Develop, sebbene il rilascio della stessa non abbia ancora una data prevista.
L'implementazione delle modifiche apportate per questo progetto con la nuova versione non dovrebbe comportare alcun problema, trattandosi principalmente di aggiunte al codice e non tanto di modifiche a quello di base, e con l'utilizzo di un ambiente di GIT non si dovrebbe incorrere in problemi di sorta.

SAN NICOLÒ

Per lo sviluppo del programma è stato necessario avvalersi dell'utilizzo di un caso studio, una struttura storica di cui si è sviluppato il modello tramite programmi AEC professionali (in questo caso si è utilizzato Revit) e che presentasse quindi tutte le complicazioni di cui si è discusso finora, e che potesse contestualmente essere poi visionata da figure professionali dell'ambito storico che potessero saggiare quindi l'effettiva utilità dello strumento in via di sviluppo.
A questo fine è stato scelto il complesso della Chiesa di San Nicolò e l'annesso convento dei Frati, a Carpi, di cui si desidera ora ripercorrere brevemente la storia.

Inquadramento storico

1123	Prima testimonianza: una bolla di Callisto II nomina la Cappella di San Nicola
XII-XIV sec	La cappella attira donazioni e lasciti, crescendo di importanza
1446	La famiglia Pio, signori di Carpi, ottengono il diritto di sepoltura presso la cappella
1477	Lionello Pio consacra definitivamente San Nicolò come chiesa dei signori di Carpi
1493	Progetto di Alberto Pio per la nuova Chiesa, che preoccupa i frati per le forme non consone alla Regola dell'Ordine, che presuppone una parziale demolizione del convento
1494-1511	Inizio edificazione della nuova struttura, con donazioni da famiglie notabili carpigiane per ornare e rinnovare le cappelle gentilizie
1516	Conclusione prima fase costruttiva
1516-18	Interruzione dei lavori per mancanza di fondi e materiali: Alberto Pio aveva infatti iniziato anche la costruzione della Collegiata e spostava beni e maestranze tra le due
1518	Alberto Pio dà ordine di destinare i materiali per terminare la chiesa di San Nicolò
1518-21	Edificazione della "seconda fabbrica", probabilmente le nuove navate
1810	Scioglimento dell'ordine dei Minori a Carpi, restano solo due frati come custodi
1812	Il convento viene venduto a privati, che causano una demolizione parziale
1818	Il duca di Modena restituisce il convento ai frati
1824-25	Ricostruzione di uno dei lati del chiostro grande
1830-50	Adeguamento ai nuovi dettami devozionali, con decorazione pittorica, inserimento di statue e nuova pavimentazione
1862-78	Interventi strutturali, con nuove capriate e copertura del coro
1866	Allontanamento dei frati, confisca del convento e trasferimento dei beni, demolizione della biblioteca
1872	Cessione da parte del Regno d'Italia del complesso di San Nicolò al Municipio di Carpi

Gli spazi

L'unione della chiesa di San Nicolò e del convento dei Minori Osservanti ad essa annesso forma un unico complesso monumentale, distinguibile ancora oggi nonostante i vari lavori e cambiamenti che hanno avuto luogo dal XIX secolo ad oggi.
La chiesa attuale è esito della ricostruzione innescata da Alberto Pio, in cui tribuna e navate, nonostante siano state costruite in diversi momenti storici, risultano concatenate in un sistema spaziale estremamente coerente. La cupola, che dall'esterno è mascherata dal tiburio, è sostenuta da quattro pilastri che si trovano all'incrocio tra quattro bracci voltati a botte, e che formano altre quattro cellule minori, anch'esse coperte da cupole, sulle diagonali, formando una struttura conosciuta come *quincunx*, caratteristica delle costruzioni bizantine e romaniche. I bracci di cui sopra terminano con delle absidi finestrate, mentre le cappelle laterali al presbiterio sono anch'esse absidate ma cieche, inoltre sono state ricavate due torri scalari nei tratti di muratura tre le esedre terminali, visibili all'esterno come campanili.

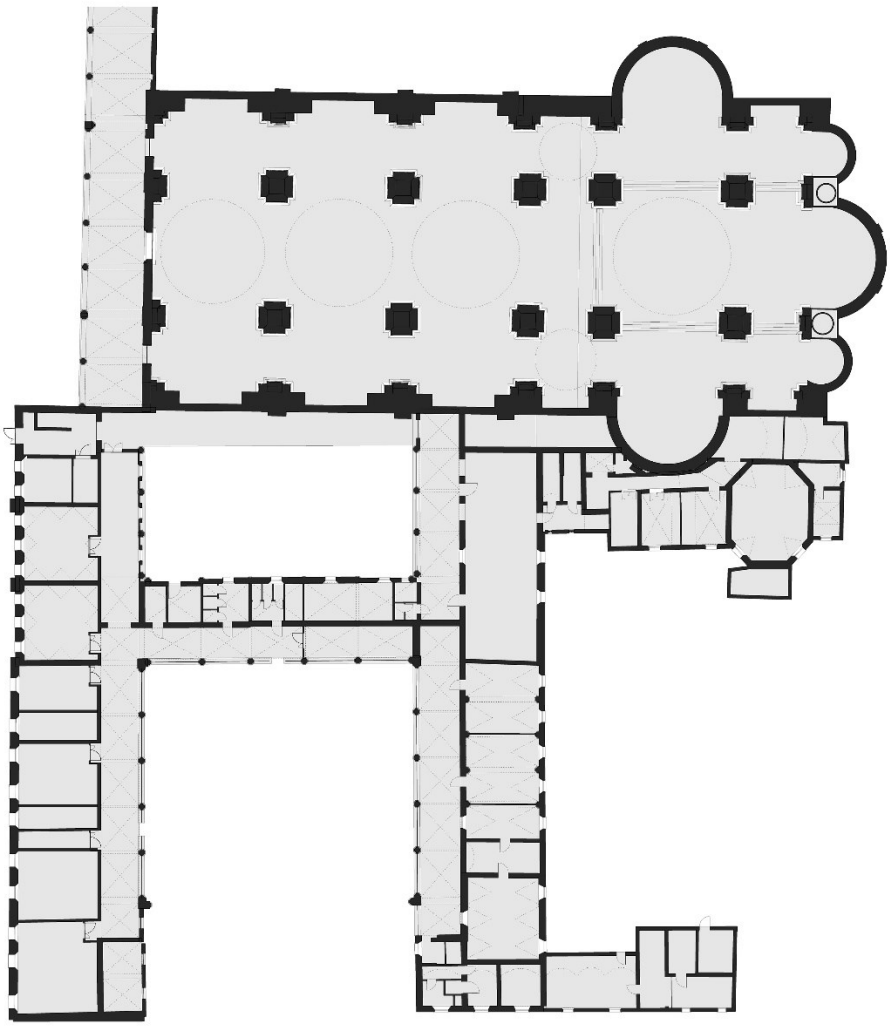

FIG. 15: Planimetria convento e chiesa, situazione odierna, quota attacco a terra. Elaborazione grafica di Paolo Borin

La parte basilicale della chiesa, frutto della seconda fase costruttiva (1518-1521), prevede una struttura a tre navate, ciascuna con tre campate: la cellula centrale è voltata a vela, mentre quelle laterali presentano una volta a botte come quella presente nel transetto. Questa gerarchia degli spazi si rispecchia anche sui pilastri che scandiscono il ritmo delle campate, i quattro spigoli presentano infatti altrettante lesene, che sono tuttavia di altezza e stili diversi: quelle rivolte alla navata centrale e al transetto sono di ordine corinzio maggiore rispetto a quelle doriche che si affacciano sulle navate laterali, che si intrecciano per evidenziare gli archi trasversali che dividono le varie campate, adattandosi infine anche alle cappelle periferiche.
Al sistema architettonico e decorativo della chiesa così complesso viene accostato un complesso di chiostri dal tenore significativamente più modesto: dei due chiostri presenti, il primo risulta addossato direttamente alla chiesa, e dei suoi tre lati porticati due sono tamponati, il secondo invece, più grande, è privo di un lato e anch'esso ha i portici quasi interamente tamponati. Un cortile di servizio era inoltre presente a est del dormitorio, anche questo a ridosso della chiesa.

Pittura e decorazioni

Mentre all'esterno la distinzione tra i due corpi di fabbrica risulta molto visibile, internamente lo spazio architettonico risulta molto coso, questo grazie anche all'apparato pittorico che trasmetta un forte senso di unitarietà di lettura. Ciò che è attualmente visibile all'interno è frutto di una serie di restauri e integrazioni che hanno portato ad una situazione differente rispetto a quella del primo Cinquecento.

L'approccio del progetto originale prevedeva elementi puramente decorativi per i muri interni, volti a sottolineare le figure architettoniche, mentre la zona degli altari presentava elementi e rappresentazioni connessi alla devozione. Questa ricostruzione è possibile grazie ad un dipinto del 1831 ad opera di Pietro Gualdi, che raffigura gli interni di San Nicolò allo stato di fatto prima degli interventi di restauro cui lui stesso ha preso parte. Dall'analisi del dipinto si notano che i diversi elementi decorativi sono distribuiti in base agli elementi architettonici: candelabre sui pilastri delle navate e del transetto a fondo blu e oro, e sugli arconi a sfondo rosso; le cornici presentano invece dei fregi a grottesche a tralcio abitato su sfondo blu e rosso; le volte e le calotte invece sono decorate con dei finti lacunari dipinti per simulare uno sfondato prospettico. Uno schema analogo è presente anche nella zona del transetto, così come sull'altare maggiore e nella cupola, decorata tramite delle fasce verticali a sfondo alternato oro e blu. A seguito dei lavori tardo ottocenteschi, i pennacchi della cupola raffigurano i quattro evangelisti, mentre sugli arconi sono stati inseriti medaglioni con i santi.
Le tonalità raffigurate nel dipinto di Gualdi mostrano una chiesa drasticamente diversa da quella attuale: in primo luogo, il pavimento in cotto si presta come ottimo elemento di legatura, offrendo al tempo stesso unitarietà di luce, nonché eleganza e solennità, mentre gli elementi geometrici risultano molto meno rigidi se paragonati alle rappresentazioni ottocentesche.
Questo programma decorativo non era inusuale nel contesto in cui si colloca: lo stesso sistema decorativo si trova a Palazzo dei Pio, nelle decorazioni che adornano l'interno della facciata occidentale, sia per quanto riguarda l'abbinamento di candelabre a fasce a grottesche, sia proprio per la tipologia di queste

ultime, cosiddette "a tralcio abitato", dove vengono raffigurati animali e figure mitologiche, tra rami ed elementi vegetali. In entrambi i casi, le decorazioni sono frutto del lavoro di Giovanni del Sega, presumibilmente nei primi trent'anni del Cinquecento, quantomeno per il progetto e la sua impostazione, sono state portate poi a compimento da altri artisti carpigiani, specialmente nelle navate.

FIG. 16: Pietro Gualdi, *Interno della chiesa di San Nicolò*, 1831, Musei di Palazzo dei Pio di Carpi, inv. A/221

Le cappelle

Le cappelle che è possibile visionare attualmente all'interno della chiesa presentano l'assetto e le decorazioni dovute alla Controriforma, è tuttavia possibile ipotizzare la struttura che esse presentavano del primo Cinquecento prendendo come guida la cappella di San Rocco e quella di San Bonaventura, che conservano le ancone lignee strutturalmente semplici, con colonne che incorniciano lo spazio dedicato alla pala.
Dei dipinti originali che adornavano la chiesa e le cappelle resta principalmente la descrizione a meno di Gasparo Pozzuoli, canonico, datata 1624, che riporta alcune delle modifiche apportate con la Controriforma. Permangono inoltre le tavole di Bernardino Loschi, dedicate all'annunciazione e a San Rocco, commissionate probabilmente dalla famiglia Pio, per la quale l'artista lavorava già come

sovrintendente dei cantieri. Un'altra tavola, sempre di sua fattura, raffigurante la *Natività*, era posizionata nella cappella della famiglia Priori, acquistata poi da privati ed ora conservata nel Museo di Palazzo dei Pio. Sappiamo inoltre che la tavola col *Martirio dei Francescani* era posizionata nella cappella della famiglia Muzzarini, sebbene sia stata poi spostata altrove, facendo poi perdere le proprie tracce.
Come in molte altre chiese dell'epoca, le immagini di devozione presenti sugli altari sono state nella maggior parte rimosse e sostituite con opere consone ai dettami della controriforma, più orientate quindi ad una semplificazione dell'evento e del personaggio sacro per facilitarne la comprensione e l'avvicinamento da parte dei fedeli.

L'area dell'altare invece sembra non fosse decorata nel primo Cinquecento, come si evince dal testamento di Alberto Pio stesso, che lascia indicazioni proprio per la decorazione della zona absidale: non sono presenti tracce di nessun genere dell'opera da lui descritta, probabilmente in quanto non allineata ai dettami in divenire. La prima opera collocata sull'altare maggiore di cui ci è giunta traccia è invece il dipinto raffigurante le *Stimmate di San Francesco*, sostituito poi con una tela con un soggetto analogo, che è quella che è ancora presente sul posto.

L'Ottocento

Contestualmente ai lavori strutturali, anche le decorazioni pittoriche subirono interventi che si riscontrano nello stato attuale.
Il primo di questi è stato attuato negli anni Trenta, contemporaneamente ai lavori necessari per via del terremoto che colpì la zona nel 1832. Le decorazioni rinascimentali vennero restaurate com'era costume dell'epoca, vennero quindi apportati pesanti integramenti per completare le parti distrutte, di fatto risultando in una ridipintura con i soggetti originali ma parzialmente aggiornati. L'esecuzione viene inizialmente affidata a Pietro Gualdi, e successivamente a Claudio Rossi, che, a differenza del suo predecessore, inserisce elementi caratteristici della scuola raffaellesca nell'ornato rinascimentale, incoerenti con gli affreschi già presenti. Le decorazioni a candelabre e i medaglioni con i santi sono conseguenza di questa serie di restauri, così come lo è il cambio di cromia: l'obiettivo di portare uniformità risulta invece in un effetto di rigidità e piattezza, in netto contrasto con la vivacità che si può vedere nel sopracitato dipinto di Gualdi dello stato precedente ai restauri.
Un secondo intervento venne iniziato con l'acquisizione del 1872 da parte del Municipio di Carpi, questa volta con l'intenzione di adeguare le decorazioni alla nuova idea di devozione, inserendo i temi della celebrazione dell'ordine francescano e la glorificazione della verità della fede. La sostituzione dei pavimenti venne bocciata dal Ministero, optando per il mantenimento del cotto, che venne tuttavia rimosso nel 1895, sostituito con delle mattonelle romboidali disposte a stella, come quelle presenti al giorno d'oggi, che mantengono il disegno di allora ma con l'utilizzo del marmo anziché del cotto dell'epoca, troppo fragile.

APPENDICE: IL CODICE

In questa appendice è stata raccolta la maggior parte del codice che è stato scritto nello sviluppo di questo progetto. Il codice è stato diviso in quattro raggruppamenti, in base alla funzionalità di cui va a far parte, tuttavia, essendo l'applicativo fortemente interconnesso, alcuni metodi sono utilizzati in più di un raggruppamento.

A1. Importazione dei livelli di Revit in Unity

A1.1 Creazione delle geometrie 3D associate ai livelli "piano edificio" con Dynamo

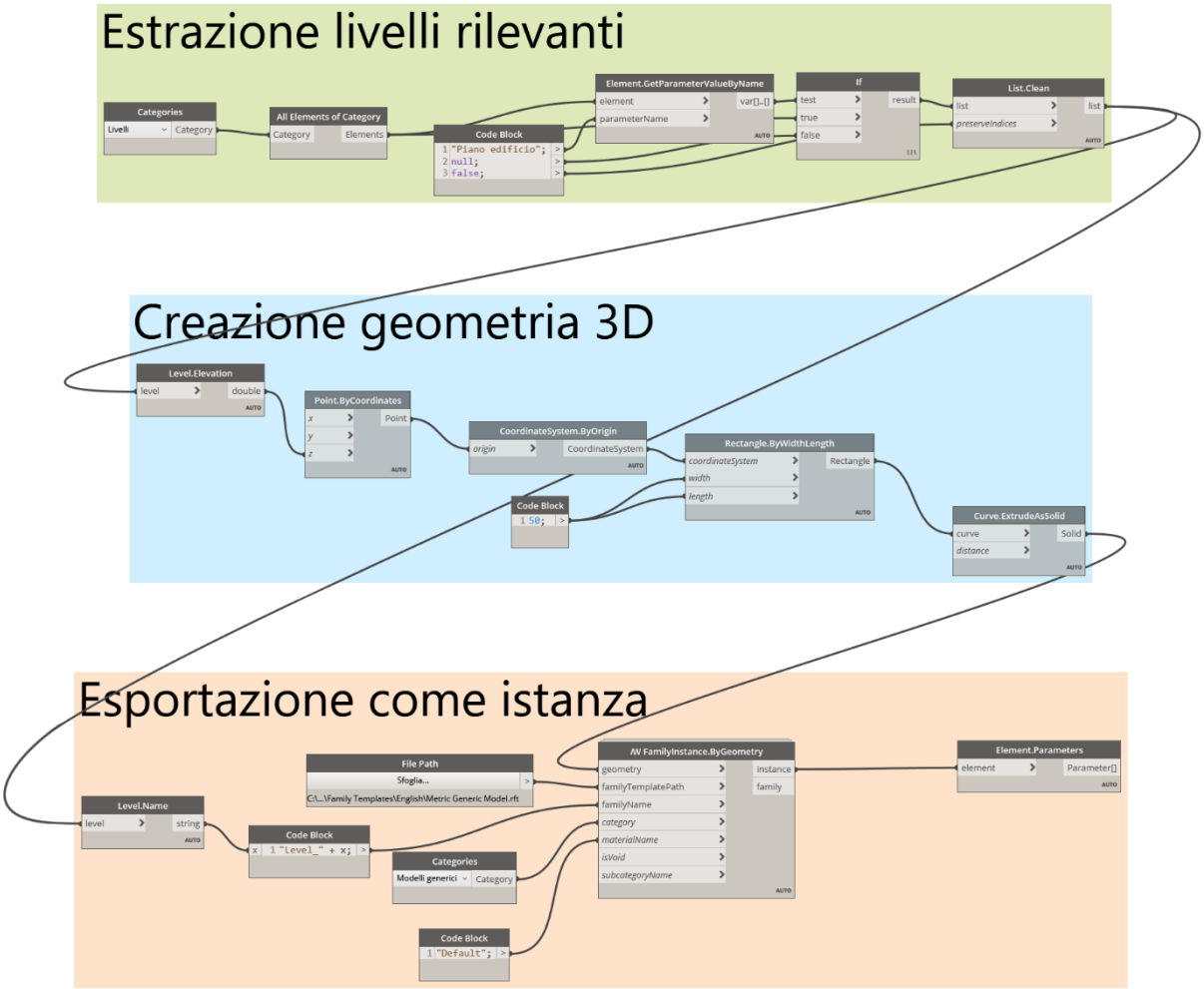

FIG. 17: Algoritmo in Dynamo per importare i livelli in Unity

A1.2 Importazione in Unity e creazione degli elementi connessi

```
public class LevelManager : MonoBehaviour
{

    public GameObject levelPlane, levelButton, refGroup;
    public List<GameObject> levels;
    public Toggle levelToggle, sectionsToggle;

    public void CheckLevels()
    {
        foreach(Transform child in refGroup.transform)
        {
            try
            {
                Metadata m = child.gameObject.GetComponent<Metadata>();
                if(m != null)
                {
                    if(m.GetParameter("Family").ToLower().Contains("level") ||
m.GetParameter("Family").ToLower().Contains("livell"))
                    {
                        if (child.transform.childCount == 0)
                        {
                            GameObject plane = GameObject.Instantiate(levelPlane,
child);
                            float y = 0;
                            foreach(var par in m.GetParameters())
                            {
                                if(par.Key.Contains("Quota altimetrica"))
                                {
                                    plane.GetComponent<LevelHeight>().height =
int.Parse(par.Value.value);
                                }
                            }
                            plane.transform.localPosition = new Vector3(0, 0, 0);
                            plane.SetActive(true);

                            GameObject levelBtn = GameObject.Instantiate(levelButton,
levelButton.transform.parent);
                            levelBtn.GetComponent<FaceCamera>().target =
plane.transform.Find("BtnPlaceholder").gameObject;
                            levelBtn.SetActive(true);

                            levels.Add(child.gameObject);
                            levels.Add(levelBtn);
                            child.GetComponent<MeshRenderer>().enabled = false;

                            string levelName =
child.GetComponent<Metadata>().GetParameter("Type");
                            levelBtn.GetComponentInChildren<TextMeshProUGUI>().text =
levelName;

plane.transform.SetParent(refGroup.transform.parent.Find("Livelli"), true);
                        }
                    }
                }
            }
            catch { }
        }
    }
```

```
    public void LevelStatus()
    {
        foreach(GameObject level in levels)
        {
            level.SetActive(levelToggle.isOn);
        }
    }

    // Update is called once per frame
    void Update()
    {
        if(levels.Count > 0)
        {
            LevelStatus();
        }
    }
}
```

A1.3 Lettura e associazione quota altimetrica ai nuovi oggetti

```
public class LevelHeight : MonoBehaviour
{

    public float height;
    public SectionCube sectionCube;
    // Start is called before the first frame update
    void Start()
    {
        try
        {
            foreach (var par in GetComponentInParent<Metadata>().GetParameters())
            {
                if (par.Key.Contains("Quota altimetrica"))
                {
                    height = float.Parse(par.Value.value);
                }
            }
        }
        catch { }
    }

    public void Section()
    {
        Debug.Log(height);
        sectionCube.SectionToLevel(height);
    }
}
```

A2. Creazione della Section Box

A2.1 Modificare la mesh della Section Box in runtime

```
public class SectionCube : MonoBehaviour
{
    public float fac = 1000;
    public Vector3[] vertices;
    public int[] originalTriangles;

    public Vector4[] FaceVertexes = new Vector4[6];

    public GameObject planeX1, planeX2, planeY1, planeY2, planeZ1, planeZ2, sampleLevel;

    MeshFilter meshFilter;
    Mesh originalMesh, clonedMesh;
    Vector3[] originalVerts, newVerts;
    int[] triangles;

    void Start()
    {
        InitMesh();
        InitializeFaces();

    }

    public void InitMesh()
    {
        meshFilter = GetComponent<MeshFilter>();
        originalMesh = meshFilter.sharedMesh; //1
        clonedMesh = new Mesh(); //2

        clonedMesh.name = "clone";
        clonedMesh.vertices = originalMesh.vertices;
        clonedMesh.triangles = originalMesh.triangles;
        clonedMesh.normals = originalMesh.normals;
        clonedMesh.uv = originalMesh.uv;
        meshFilter.mesh = clonedMesh;  //3

        newVerts = clonedMesh.vertices; //4
        triangles = clonedMesh.triangles;

        originalVerts = GetComponent<MeshFilter>().sharedMesh.vertices;
    }

    public void InitializeFaces()
    {

        FaceVertexes[0] = new Vector4(0, 1, 6, 7);
        FaceVertexes[1] = new Vector4(2, 3, 4, 5);
        FaceVertexes[2] = new Vector4(1, 3, 5, 7);
        FaceVertexes[3] = new Vector4(0, 2, 4, 6);
        FaceVertexes[4] = new Vector4(0, 1, 2, 3);
        FaceVertexes[5] = new Vector4(4, 5, 6, 7);

    }

    public void ExtrudeFace(float extrudeAmount, int faceId)
    {

        // Loop tra gli indici dei vertici nella faccia
```

```
        Vector4 face = FaceVertexes[faceId];

        // X1, X2, Y1, Y2, Z1, Z2

        int[] indexes = { ((int)face.x), ((int)face.y), ((int)face.z), ((int)face.w) };

        foreach(int ind in indexes)
        {
            // Loop tra tutti i vertici della mesh
            foreach(Vector3 v in GetComponent<MeshFilter>().sharedMesh.vertices)
            {
                // Trovo corrispondenza
                if(v == vertices[ind])
                {
                    Vector3[] meshVertices =
GetComponent<MeshFilter>().sharedMesh.vertices;
                    int n = System.Array.IndexOf(meshVertices, v);

                    // Muovo il vertice della mesh
                    if (faceId == 0)
                    {
                        // X2
                        meshVertices[n].y -= extrudeAmount;

                        // plane Y1
                        Vector3 y1_0 = new Vector3(
                            planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[0].x,
                            planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[0].y -
(extrudeAmount / 12),
                            planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[0].z
                        );

                        Vector3 y1_2 = new Vector3(
                            planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[2].x,
                            planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[2].y -
(extrudeAmount / 12),
                            planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[2].z
                        );

                        Vector3[] newVerts_y1 = new Vector3[]
                        {
                            y1_0,
                            planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[1],
                            y1_2,
                            planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[3],
                        };

                        Mesh m_y1 = planeY1.GetComponent<MeshCollider>().sharedMesh;
                        m_y1.vertices = newVerts_y1;
                        planeY1.GetComponent<MeshCollider>().sharedMesh = null;
                        planeY1.GetComponent<MeshCollider>().sharedMesh = m_y1;

                        // plane Y2
                        Vector3 y2_0 = new Vector3(
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[0].x,
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[0].y -
(extrudeAmount / 12),
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[0].z
                        );

                        Vector3 y2_2 = new Vector3(
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[2].x,
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[2].y -
(extrudeAmount / 12),
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[2].z
                        );
```

```
                        Vector3[] newVerts_y2 = new Vector3[]
                        {
                            y2_0,
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[1],
                            y2_2,
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[3],
                        };

                        Mesh m_y2 = planeY2.GetComponent<MeshCollider>().sharedMesh;
                        m_y2.vertices = newVerts_y2;
                        planeY2.GetComponent<MeshCollider>().sharedMesh = null;
                        planeY2.GetComponent<MeshCollider>().sharedMesh = m_y2;

                        // plane Z1
                        Vector3 z1_0 = new Vector3(
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[0].x,
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[0].y -
(extrudeAmount / 12),
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[0].z
                        );

                        Vector3 z1_2 = new Vector3(
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[2].x,
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[2].y -
(extrudeAmount / 12),
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[2].z
                        );

                        Vector3[] newVerts_z1 = new Vector3[]
                        {
                            z1_0,
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[1],
                            z1_2,
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[3],
                        };

                        Mesh m_z1 = planeZ1.GetComponent<MeshCollider>().sharedMesh;
                        m_z1.vertices = newVerts_z1;
                        planeZ1.GetComponent<MeshCollider>().sharedMesh = null;
                        planeZ1.GetComponent<MeshCollider>().sharedMesh = m_z1;

                        // plane Z2
                        Vector3 z2_0 = new Vector3(
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[0].x,
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[0].y -
(extrudeAmount / 12),
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[0].z
                        );

                        Vector3 z2_2 = new Vector3(
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[2].x,
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[2].y -
(extrudeAmount / 12),
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[2].z
                        );

                        Vector3[] newVerts_z2 = new Vector3[]
                        {
                            z2_0,
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[1],
                            z2_2,
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[3],
                        };

                        Mesh m_z2 = planeZ2.GetComponent<MeshCollider>().sharedMesh;
                        m_z2.vertices = newVerts_z2;
```

```
                    planeZ2.GetComponent<MeshCollider>().sharedMesh = null;
                    planeZ2.GetComponent<MeshCollider>().sharedMesh = m_z2;
                }
if (faceId == 1)
                {
                    // X1
                    meshVertices[n].y += extrudeAmount;

                    // plane Y1
                    Vector3 y1_1 = new Vector3(
                        planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[1].x,
                        planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[1].y +
(extrudeAmount/12),
                        planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[1].z
                    );

                    Vector3 y1_3 = new Vector3(
                        planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[3].x,
                        planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[3].y +
(extrudeAmount/ 12),
                        planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[3].z
                    );

                    Vector3[] newVerts_y1 = new Vector3[]
                    {
                        planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[0],
                        y1_1,
                        planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[2],
                        y1_3
                    };

                    Mesh m_y1 = planeY1.GetComponent<MeshCollider>().sharedMesh;
                    m_y1.vertices = newVerts_y1;
                    planeY1.GetComponent<MeshCollider>().sharedMesh = null;
                    planeY1.GetComponent<MeshCollider>().sharedMesh = m_y1;

                    // plane Y2
                    Vector3 y2_1 = new Vector3(
                        planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[1].x,
                        planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[1].y +
(extrudeAmount / 12),
                        planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[1].z
                    );

                    Vector3 y2_3 = new Vector3(
                        planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[3].x,
                        planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[3].y +
(extrudeAmount / 12),
                        planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[3].z
                    );

                    Vector3[] newVerts_y2 = new Vector3[]
                    {
                        planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[0],
                        y2_1,
                        planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[2],
                        y2_3
                    };

                    Mesh m_y2 = planeY2.GetComponent<MeshCollider>().sharedMesh;
                    m_y2.vertices = newVerts_y2;
                    planeY2.GetComponent<MeshCollider>().sharedMesh = null;
                    planeY2.GetComponent<MeshCollider>().sharedMesh = m_y2;
```

```
                        // plane Z1
                        Vector3 z1_1 = new Vector3(
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[1].x,
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[1].y +
(extrudeAmount / 12),
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[1].z
                        );

                        Vector3 z1_3 = new Vector3(
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[3].x,
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[3].y +
(extrudeAmount / 12),
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[3].z
                        );

                        Vector3[] newVerts_z1 = new Vector3[]
                        {
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[0],
                            z1_1,
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[2],
                            z1_3
                        };

                        Mesh m_z1 = planeZ1.GetComponent<MeshCollider>().sharedMesh;
                        m_z1.vertices = newVerts_z1;
                        planeZ1.GetComponent<MeshCollider>().sharedMesh = null;
                        planeZ1.GetComponent<MeshCollider>().sharedMesh = m_z1;

                        // plane Z2
                        Vector3 z2_1 = new Vector3(
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[1].x,
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[1].y +
(extrudeAmount / 12),
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[1].z
                        );

                        Vector3 z2_3 = new Vector3(
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[3].x,
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[3].y +
(extrudeAmount / 12),
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[3].z
                        );

                        Vector3[] newVerts_z2 = new Vector3[]
                        {
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[0],
                            z2_1,
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[2],
                            z2_3
                        };

                        Mesh m_z2 = planeZ2.GetComponent<MeshCollider>().sharedMesh;
                        m_z2.vertices = newVerts_z2;
                        planeZ2.GetComponent<MeshCollider>().sharedMesh = null;
                        planeZ2.GetComponent<MeshCollider>().sharedMesh = m_z2;
                    }

                    if (faceId == 2)
                    {
                        // Y1
                        meshVertices[n].x -= extrudeAmount;

                        // plane X1
                        Vector3 x1_0 = new Vector3(
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[0].x -
(extrudeAmount / 12),
```

```
                        planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[0].y,
                        planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[0].z
                    );

                    Vector3 x1_3 = new Vector3(
                        planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[3].x -
(extrudeAmount / 12),
                        planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[3].y,
                        planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[3].z
                    );

                    Vector3[] newVerts_x1 = new Vector3[]
                    {
                        x1_0,
                        planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[1],
                        planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[2],
                        x1_3
                    };

                    Mesh m_x1 = planeX1.GetComponent<MeshCollider>().sharedMesh;
                    m_x1.vertices = newVerts_x1;
                    planeX1.GetComponent<MeshCollider>().sharedMesh = null;
                    planeX1.GetComponent<MeshCollider>().sharedMesh = m_x1;

                    // plane X2
                    Vector3 x2_0 = new Vector3(
                        planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[0].x -
(extrudeAmount / 12),
                        planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[0].y,
                        planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[0].z
                    );

                    Vector3 x2_3 = new Vector3(
                        planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[3].x -
(extrudeAmount / 12),
                        planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[3].y,
                        planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[3].z
                    );

                    Vector3[] newVerts_x2 = new Vector3[]
                    {
                        x2_0,
                        planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[1],
                        planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[2],
                        x2_3
                    };

                    Mesh m_x2 = planeX2.GetComponent<MeshCollider>().sharedMesh;
                    m_x2.vertices = newVerts_x2;
                    planeX2.GetComponent<MeshCollider>().sharedMesh = null;
                    planeX2.GetComponent<MeshCollider>().sharedMesh = m_x2;

                    // plane Z1
                    Vector3 z1_1 = new Vector3(
                        planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[1].x +
(extrudeAmount / 12),
                        planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[1].y,
                        planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[1].z
                    );

                    Vector3 z1_2 = new Vector3(
                        planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[2].x +
(extrudeAmount / 12),
                        planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[2].y,
                        planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[2].z
                    );
```

```
                        Vector3[] newVerts_z1 = new Vector3[]
                        {
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[0],
                            z1_1,
                            z1_2,
                            planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[3],
                        };

                        Mesh m_z1 = planeZ1.GetComponent<MeshCollider>().sharedMesh;
                        m_z1.vertices = newVerts_z1;
                        planeZ1.GetComponent<MeshCollider>().sharedMesh = null;
                        planeZ1.GetComponent<MeshCollider>().sharedMesh = m_z1;

                        // plane Z2
                        Vector3 z2_0 = new Vector3(
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[0].x -
(extrudeAmount / 12),
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[0].y,
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[0].z
                        );

                        Vector3 z2_3 = new Vector3(
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[3].x -
(extrudeAmount / 12),
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[3].y,
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[3].z
                        );

                        Vector3[] newVerts_z2 = new Vector3[]
                        {
                            z2_0,
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[1],
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[2],
                            z2_3
                        };

                        Mesh m_z2 = planeZ2.GetComponent<MeshCollider>().sharedMesh;
                        m_z2.vertices = newVerts_z2;
                        planeZ2.GetComponent<MeshCollider>().sharedMesh = null;
                        planeZ2.GetComponent<MeshCollider>().sharedMesh = m_z2;
                    }

                    if (faceId == 3)
                    {
                        // Y2
                        meshVertices[n].x += extrudeAmount;

                        // plane X1
                        Vector3 x1_1 = new Vector3(
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[1].x +
(extrudeAmount / 12),
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[1].y,
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[1].z
                        );

                        Vector3 x1_2 = new Vector3(
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[2].x +
(extrudeAmount / 12),
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[2].y,
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[2].z
                        );

                        Vector3[] newVerts_x1 = new Vector3[]
                        {
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[0],
                            x1_1,
```

```
                        x1_2,
                        planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[3],
                    };

                    Mesh m_x1 = planeX1.GetComponent<MeshCollider>().sharedMesh;
                    m_x1.vertices = newVerts_x1;
                    planeX1.GetComponent<MeshCollider>().sharedMesh = null;
                    planeX1.GetComponent<MeshCollider>().sharedMesh = m_x1;

                    // plane X2
                    Vector3 x2_1 = new Vector3(
                        planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[1].x +
(extrudeAmount / 12),
                        planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[1].y,
                        planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[1].z
                    );

                    Vector3 x2_2 = new Vector3(
                        planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[2].x +
(extrudeAmount / 12),
                        planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[2].y,
                        planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[2].z
                    );

                    Vector3[] newVerts_x2 = new Vector3[]
                    {
                        planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[0],
                        x2_1,
                        x2_2,
                        planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[3],
                    };

                    Mesh m_x2 = planeX2.GetComponent<MeshCollider>().sharedMesh;
                    m_x2.vertices = newVerts_x2;
                    planeX2.GetComponent<MeshCollider>().sharedMesh = null;
                    planeX2.GetComponent<MeshCollider>().sharedMesh = m_x2;

                    // plane Z1
                    Vector3 z1_0 = new Vector3(
                        planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[0].x +
(extrudeAmount / 12),
                        planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[0].y,
                        planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[0].z
                    );

                    Vector3 z1_3 = new Vector3(
                        planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[3].x +
(extrudeAmount / 12),
                        planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[3].y,
                        planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[3].z
                    );

                    Vector3[] newVerts_z1 = new Vector3[]
                    {
                        z1_0,
                        planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[1],
                        planeZ1.GetComponent<MeshCollider>().sharedMesh.vertices[2],
                        z1_3,
                    };

                    Mesh m_z1 = planeZ1.GetComponent<MeshCollider>().sharedMesh;
                    m_z1.vertices = newVerts_z1;
                    planeZ1.GetComponent<MeshCollider>().sharedMesh = null;
                    planeZ1.GetComponent<MeshCollider>().sharedMesh = m_z1;
```

```
                        // plane Z2
                        Vector3 z2_1 = new Vector3(
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[1].x +
(extrudeAmount / 12),
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[1].y,
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[1].z
                        );

                        Vector3 z2_2 = new Vector3(
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[2].x +
(extrudeAmount / 12),
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[2].y,
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[2].z
                        );

                        Vector3[] newVerts_z2 = new Vector3[]
                        {
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[0],
                            z2_1,
                            z2_2,
                            planeZ2.GetComponent<MeshCollider>().sharedMesh.vertices[3],
                        };

                        Mesh m_z2 = planeZ2.GetComponent<MeshCollider>().sharedMesh;
                        m_z2.vertices = newVerts_z2;
                        planeZ2.GetComponent<MeshCollider>().sharedMesh = null;
                        planeZ2.GetComponent<MeshCollider>().sharedMesh = m_z2;
                    }

                    if (faceId == 4)
                    {
                        // Z1
                        meshVertices[n].z += extrudeAmount;

                        // plane X1
                        Vector3 x1_1 = new Vector3(
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[1].x,
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[1].y +
(extrudeAmount / 12),
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[1].z
                        );

                        Vector3 x1_3 = new Vector3(
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[3].x,
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[3].y +
(extrudeAmount / 12),
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[3].z
                        );

                        Vector3[] newVerts_x1 = new Vector3[]
                        {
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[0],
                            x1_1,
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[2],
                            x1_3
                        };

                        Mesh m_x1 = planeX1.GetComponent<MeshCollider>().sharedMesh;
                        m_x1.vertices = newVerts_x1;
                        planeX1.GetComponent<MeshCollider>().sharedMesh = null;
                        planeX1.GetComponent<MeshCollider>().sharedMesh = m_x1;

                        // plane X2
                        Vector3 x2_0 = new Vector3(
                            planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[0].x,
```

```
                    planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[0].y -
(extrudeAmount / 12),
                    planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[0].z
                );

                Vector3 x2_2 = new Vector3(
                    planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[2].x,
                    planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[2].y -
(extrudeAmount / 12),
                    planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[2].z
                );

                Vector3[] newVerts_x2 = new Vector3[]
                {
                    x2_0,
                    planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[1],
                    x2_2,
                    planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[3],
                };

                Mesh m_x2 = planeX2.GetComponent<MeshCollider>().sharedMesh;
                m_x2.vertices = newVerts_x2;
                planeX2.GetComponent<MeshCollider>().sharedMesh = null;
                planeX2.GetComponent<MeshCollider>().sharedMesh = m_x2;

                // plane Y1
                Vector3 y1_0 = new Vector3(
                    planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[0].x -
(extrudeAmount / 12),
                    planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[0].y,
                    planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[0].z
                );

                Vector3 y1_3 = new Vector3(
                    planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[3].x -
(extrudeAmount / 12),
                    planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[3].y,
                    planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[3].z
                );

                Vector3[] newVerts_y1 = new Vector3[]
                {
                    y1_0,
                    planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[1],
                    planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[2],
                    y1_3,
                };

                Mesh m_y1 = planeY1.GetComponent<MeshCollider>().sharedMesh;
                m_y1.vertices = newVerts_y1;
                planeY1.GetComponent<MeshCollider>().sharedMesh = null;
                planeY1.GetComponent<MeshCollider>().sharedMesh = m_y1;

                // plane Y2
                Vector3 y2_1 = new Vector3(
                    planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[1].x +
(extrudeAmount / 12),
                    planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[1].y,
                    planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[1].z
                );
                Vector3 y2_2 = new Vector3(
                    planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[2].x +
(extrudeAmount / 12),
                    planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[2].y,
                    planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[2].z
                );
```

```
                        Vector3[] newVerts_y2 = new Vector3[]
                        {
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[0],
                            y2_1,
                            y2_2,
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[3],
                        };

                        Mesh m_y2 = planeY2.GetComponent<MeshCollider>().sharedMesh;
                        m_y2.vertices = newVerts_y2;
                        planeY2.GetComponent<MeshCollider>().sharedMesh = null;
                        planeY2.GetComponent<MeshCollider>().sharedMesh = m_y2;
                    }
if (faceId == 5)
                    {
                        // Z2
                        meshVertices[n].z -= extrudeAmount;

                        // plane X1
                        Vector3 x1_0 = new Vector3(
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[0].x,
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[0].y -
(extrudeAmount / 12),
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[0].z
                        );

                        Vector3 x1_2 = new Vector3(
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[2].x,
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[2].y -
(extrudeAmount / 12),
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[2].z
                        );

                        Vector3[] newVerts_x1 = new Vector3[]
                        {
                            x1_0,
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[1],
                            x1_2,
                            planeX1.GetComponent<MeshCollider>().sharedMesh.vertices[3],
                        };
                        Mesh m_x1 = planeX1.GetComponent<MeshCollider>().sharedMesh;
                        m_x1.vertices = newVerts_x1;
                        planeX1.GetComponent<MeshCollider>().sharedMesh = null;
                        planeX1.GetComponent<MeshCollider>().sharedMesh = m_x1;

                        // plane X2
                        Vector3 x2_1 = new Vector3(
                            planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[1].x,
                            planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[1].y +
(extrudeAmount / 12),
                            planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[1].z
                        );
                        Vector3 x2_3 = new Vector3(
                            planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[3].x,
                            planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[3].y +
(extrudeAmount / 12),
                            planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[3].z
                        );

                        Vector3[] newVerts_x2 = new Vector3[]
                        {
                            planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[0],
                            x2_1,
                            planeX2.GetComponent<MeshCollider>().sharedMesh.vertices[2],
                            x2_3,
                        };
```

```
                        Mesh m_x2 = planeX2.GetComponent<MeshCollider>().sharedMesh;
                        m_x2.vertices = newVerts_x2;
                        planeX2.GetComponent<MeshCollider>().sharedMesh = null;
                        planeX2.GetComponent<MeshCollider>().sharedMesh = m_x2;
                        // plane Y1
                        Vector3 y1_1 = new Vector3(
                            planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[1].x +
(extrudeAmount / 12),
                            planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[1].y,
                            planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[1].z
                        );
                        Vector3 y1_2 = new Vector3(
                            planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[2].x +
(extrudeAmount / 12),
                            planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[2].y,
                            planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[2].z
                        );
                        Vector3[] newVerts_y1 = new Vector3[]
                        {
                            planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[0],
                            y1_1,
                            y1_2,
                            planeY1.GetComponent<MeshCollider>().sharedMesh.vertices[3],
                        };
                        Mesh m_y1 = planeY1.GetComponent<MeshCollider>().sharedMesh;
                        m_y1.vertices = newVerts_y1;
                        planeY1.GetComponent<MeshCollider>().sharedMesh = null;
                        planeY1.GetComponent<MeshCollider>().sharedMesh = m_y1;
                        // plane Y2
                        Vector3 y2_0 = new Vector3(
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[0].x -
(extrudeAmount / 12),
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[0].y,
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[0].z
                        );

                        Vector3 y2_3 = new Vector3(
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[3].x -
(extrudeAmount / 12),
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[3].y,
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[3].z
                        );

                        Vector3[] newVerts_y2 = new Vector3[]
                        {
                            y2_0,
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[1],
                            planeY2.GetComponent<MeshCollider>().sharedMesh.vertices[2],
                            y2_3,
                        };

                        Mesh m_y2 = planeY2.GetComponent<MeshCollider>().sharedMesh;
                        m_y2.vertices = newVerts_y2;
                        planeY2.GetComponent<MeshCollider>().sharedMesh = null;
                        planeY2.GetComponent<MeshCollider>().sharedMesh = m_y2;
                    }

                    GetComponent<MeshFilter>().sharedMesh.vertices = meshVertices;
                    GetComponent<MeshFilter>().sharedMesh.RecalculateNormals();
                    GetComponent<MeshFilter>().sharedMesh.RecalculateBounds();
                    GetComponent<MeshFilter>().sharedMesh.RecalculateTangents();
                }
            }
        }
    }
```

```
void Update()
    {

        if(GetComponent<MeshFilter>().sharedMesh.name != "clone")
        {
            InitMesh();
        }

        vertices = RetrieveVerts();
        //originalTriangles = GetComponent<MeshFilter>().sharedMesh.triangles;

        planeX1.transform.localPosition = new Vector3(planeX1.transform.localPosition.x,
vertices[(int)FaceVertexes[1][0]].y, planeX1.transform.localPosition.z);
        planeX2.transform.localPosition = new Vector3(planeX2.transform.localPosition.x,
vertices[(int)FaceVertexes[0][0]].y, planeX2.transform.localPosition.z);
        planeY1.transform.localPosition = new Vector3(vertices[(int)FaceVertexes[2][0]].x,
planeY1.transform.localPosition.y, planeY1.transform.localPosition.z);
        planeY2.transform.localPosition = new Vector3(vertices[(int)FaceVertexes[3][0]].x,
planeY2.transform.localPosition.y, planeY2.transform.localPosition.z);
        planeZ1.transform.localPosition = new Vector3(planeZ1.transform.localPosition.x,
planeZ1.transform.localPosition.y, vertices[(int)FaceVertexes[4][0]].z);
        planeZ2.transform.localPosition = new Vector3(planeZ2.transform.localPosition.x,
planeZ2.transform.localPosition.y, vertices[(int)FaceVertexes[5][0]].z);

    }

    public void SectionToLevel(float height)
    {
        float absHeight = sampleLevel.transform.TransformPoint(new Vector3(0, height / fac,
0)).y;
        //Debug.Log("HERE current abs height " + absHeight.ToString());
        float amount = absHeight - planeX1.transform.position.y;
        Vector3 test = transform.InverseTransformVector(new Vector3(0, amount, 0));
        //Debug.Log("HERE amount y " + test.y.ToString());
        ExtrudeFace(test.y, 1);
    }

    private Vector3[] RetrieveVerts()
    {
        List<Vector3> new_verts = new List<Vector3>();
        Vector3[] verts = GetComponent<MeshFilter>().sharedMesh.vertices;

        foreach(Vector3 v in verts)
        {
            if (!new_verts.Contains(v))
            {
                new_verts.Add(v);
            }
        }
        return new_verts.ToArray();

    }

}
```

A2.2 Creazione bottoni per controllare gli spostamenti delle facce della Section Box

```
public class ButtonExtrude : MonoBehaviour, IPointerDownHandler, IPointerUpHandler
{
    public SectionCube sectionCube;
    public int faceToExtrude = 0;
    public bool ispressed = false;

    public static float startIntensity = 0.005f;
    public bool backwards = false;
    float startPress = 0f;
    bool x2 = false, x4 = false, x8 = false;
    float extrudeAmount = startIntensity;

    void Update()
    {
        if (!ispressed)
            return;

        if (Time.time - startPress >= 2.0f && x2 == false)
        {
            extrudeAmount *= 2;
            x2 = true;
        }
        if (Time.time - startPress >= 4.0f && x4 == false)
        {
            extrudeAmount *= 2;
            x4 = true;
        }
        if (Time.time - startPress >= 6.0f && x8 == false)
        {
            extrudeAmount *= 2;
            x8 = true;
        }

        if (backwards)
        {
            float negAmount = extrudeAmount * -1;
            sectionCube.ExtrudeFace(negAmount, faceToExtrude);
        }
        else
        {
            sectionCube.ExtrudeFace(extrudeAmount, faceToExtrude);
        }

    }

    public void OnPointerDown(PointerEventData eventData)
    {
        ispressed = true;
        startPress = Time.time;
    }

    public void OnPointerUp(PointerEventData eventData)
    {
        ispressed = false;
        x2 = false;
        x4 = false;
        x8 = false;
        startPress = 0f;
        extrudeAmount = startIntensity;
    }
}
```

A2.3 Rilevare la faccia selezionata e aggiornare la texture di conseguenza

```
public class SelectFace : MonoBehaviour, IPointerDownHandler
{
    public Button outwards, inwards;
    public Toggle toggle;
    public SectionCube sectionCube;
    public int faceId;

    public void OnPointerDown(PointerEventData eventData)
    {
        outwards.GetComponent<ButtonExtrude>().faceToExtrude = faceId;
        inwards.GetComponent<ButtonExtrude>().faceToExtrude = faceId;
    }

    MeshFilter meshFilter;
    Mesh originalMesh, clonedMesh;

    private void Start()
    {
        meshFilter = transform.GetChild(0).GetComponent<MeshFilter>();
        originalMesh = meshFilter.sharedMesh; //1

        if (originalMesh.name != "clone")
        {
            clonedMesh = new Mesh(); //2

            clonedMesh.name = "clone";
            clonedMesh.vertices = originalMesh.vertices;
            clonedMesh.triangles = originalMesh.triangles;
            clonedMesh.normals = originalMesh.normals;
            clonedMesh.uv = originalMesh.uv;
            meshFilter.mesh = clonedMesh;  //3
        }
    }

    public void TextureStatus()
    {
        if (toggle.isOn)
        {
            outwards.GetComponent<ButtonExtrude>().faceToExtrude = -1;
            inwards.GetComponent<ButtonExtrude>().faceToExtrude = -1;

            sectionCube.gameObject.GetComponent<MeshRenderer>().material.color = new
Color(0.1563545f, 0.6792453f, 0.5268528f, 0.2f);
            this.GetComponent<MeshCollider>().enabled = false;
        }
        // Editabile
        else
        {
            outwards.GetComponent<ButtonExtrude>().faceToExtrude = 0;
            inwards.GetComponent<ButtonExtrude>().faceToExtrude = 0;

            sectionCube.gameObject.GetComponent<MeshRenderer>().material.color = new
Color(0.1563545f, 0.6792453f, 0.5268528f, 1.0f);
            this.GetComponent<MeshCollider>().enabled = true;
        }
    }

    public Vector3 v1, v2, v3, v4;
```

```
private void Update()
    {
        if (faceId == outwards.GetComponent<ButtonExtrude>().faceToExtrude)
        {
            transform.GetChild(0).GetComponent<MeshRenderer>().enabled = true;
        }
        else
        {
            transform.GetChild(0).GetComponent<MeshRenderer>().enabled = false;
        }
    }
}
```

A2.4 Creazione dello shader personalizzato

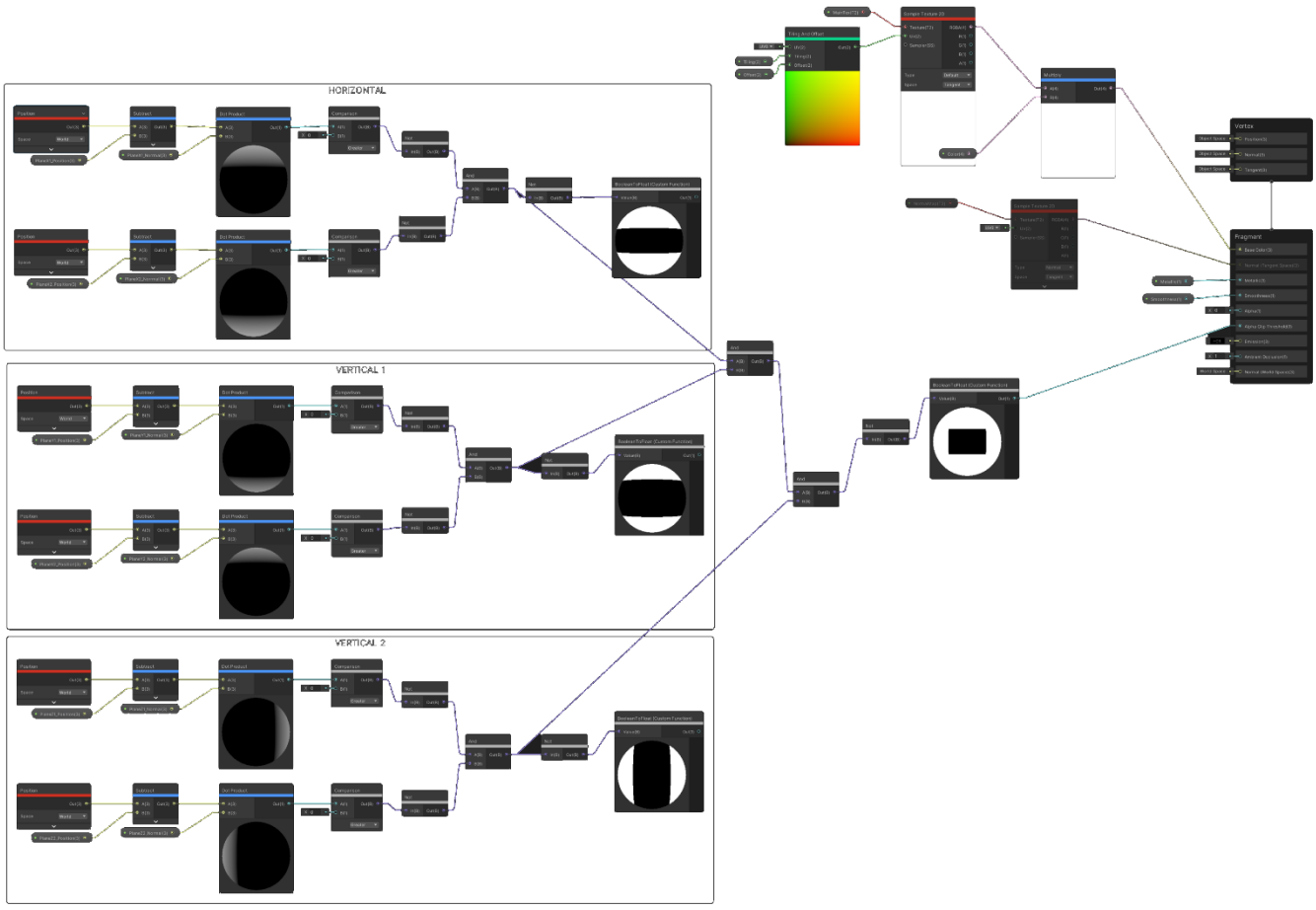

FIG. 18: Struttura dell'algoritmo dello shader in Unity

A2.5 Script di coordinamento tra i piani di taglio e lo shader

```
public class TwoPlanesCuttingController : MonoBehaviour {

    public GameObject planeX1, planeX2, planeY1, planeY2, planeZ1, planeZ2;
    public GameObject parent, clipMGR;
    public Vector3 normalX1, positionX1, normalX2, positionX2, normalY1, positionY1,
normalY2, positionY2, normalZ1, positionZ1, normalZ2, positionZ2;
    private MaterialPropertyBlock m_MaterialPropertyBlock;

    bool _sectionsOn;

    private void Awake()
    {
        m_MaterialPropertyBlock = new MaterialPropertyBlock();
    }

    private void Start()
    {
        if (parent)
        {
            UpdateShaderProperties();
            GetComponent<MeshFilter>().sharedMesh.RecalculateNormals();
            GetComponent<MeshFilter>().sharedMesh.RecalculateTangents();
        }
    }

#if UNITY_EDITOR
    private void Update()
    {
        _sectionsOn = clipMGR.GetComponent<ClipManager>()._sectionsOn;

        if (parent)
        {
            UpdateShaderProperties();
        }
    }
#endif

    public void UpdateShaderProperties()
    {
        if (planeX1 && _sectionsOn)
        {
            normalX1 = planeX1.transform.TransformVector(new Vector3(0, 0, -1));
            positionX1 = planeX1.transform.position;
        } else {
            normalX1 = Vector3.up;
            positionX1 = new Vector3(0f, 1000f, 0f);
        }

        if (planeX2 && _sectionsOn)
        {
            normalX2 = planeX2.transform.TransformVector(new Vector3(0, 0, -1));
            positionX2 = planeX2.transform.position;
        } else {
            normalX2 = -Vector3.up;
            positionX2 = new Vector3(0f, -1000f, 0f);
        }
        if (planeY1 && _sectionsOn)
        {
            normalY1 = planeY1.transform.TransformVector(new Vector3(0, 0, -1));
            positionY1 = planeY1.transform.position;
        }
```

```
        else
        {
            normalY1 = -Vector3.right;
            positionY1 = new Vector3(-1000f, 0f,0f);
        }
        if (planeY2 && _sectionsOn)
        {
            normalY2 = planeY2.transform.TransformVector(new Vector3(0, 0, -1));
            positionY2 = planeY2.transform.position;
        }
        else
        {
            normalY2 = Vector3.right;
            positionY2 = new Vector3(1000f, 0f, 0f);
        }
        if (planeZ1 && _sectionsOn)
        {
            normalZ1 = planeZ1.transform.TransformVector(new Vector3(0, 0, -1));
            positionZ1 = planeZ1.transform.position;
        }
        else
        {
            normalZ1 = Vector3.forward;
            positionZ1 = new Vector3(0f, 0f, 1000f);
        }
        if (planeZ2 && _sectionsOn)
        {
            normalZ2 = planeZ2.transform.TransformVector(new Vector3(0, 0, -1));
            positionZ2 = planeZ2.transform.position;
        }
        else
        {
            normalZ2 = -Vector3.forward;
            positionZ2 = new Vector3(0f, 0f, -1000f);
        }
        try
        {
            m_MaterialPropertyBlock.SetVector("Vector3_PlaneX1_Normal", normalX1);
            m_MaterialPropertyBlock.SetVector("Vector3_PlaneX1_Position", positionX1);
            m_MaterialPropertyBlock.SetVector("Vector3_PlaneX2_Normal", normalX2);
            m_MaterialPropertyBlock.SetVector("Vector3_PlaneX2_Position", positionX2);

            m_MaterialPropertyBlock.SetVector("Vector3_PlaneY1_Normal", normalY1);
            m_MaterialPropertyBlock.SetVector("Vector3_PlaneY1_Position", positionY1);
            m_MaterialPropertyBlock.SetVector("Vector3_PlaneY2_Normal", normalY2);
            m_MaterialPropertyBlock.SetVector("Vector3_PlaneY2_Position", positionY2);

            m_MaterialPropertyBlock.SetVector("Vector3_PlaneZ1_Normal", normalZ1);
            m_MaterialPropertyBlock.SetVector("Vector3_PlaneZ1_Position", positionZ1);
            m_MaterialPropertyBlock.SetVector("Vector3_PlaneZ2_Normal", normalZ2);
            m_MaterialPropertyBlock.SetVector("Vector3_PlaneZ2_Position", positionZ2);

            GetComponent<MeshRenderer>().SetPropertyBlock(m_MaterialPropertyBlock);
        }

        catch { }
    }
}
```

A2.6 Script di gestione del modello in caricamento

```
public class ClipManager : MonoBehaviour
{
    public List<GameObject> startingChildren;

    public Material materialToAssign;
    public GameObject planeX1, planeX2, planeY1, planeY2, planeZ1, planeZ2, utils,
loadingScreen;

    public GameObject sectionsOn;
    public bool _sectionsOn, callComplete;

    public ViewManager viewManager;

    public List<GameObject> newChildren = new List<GameObject>();

    public TextMeshProUGUI statusText, currentlyLoading;
    public bool streamingFromServer = false;

    GameObject toBeChecked;
    public bool startedImporting = false;
    bool isChildCountStable = false;

    void Start()
    {
        foreach (Transform child in transform)
        {
            startingChildren.Add(child.gameObject);
        }

        InvokeRepeating("CheckStreamingStatus", 1f, 1f);
        //InvokeRepeating("ProcessModels", .5f, 2f);
    }

    public void CheckStreamingStatus()
    {
        string status = statusText.text;

        // non sto streammando
        if (!streamingFromServer)
        {
            if (status.Contains("Streaming") || status.Contains("Opening"))
            {
                streamingFromServer = true;
                currentlyLoading.text = "Streaming from server...";
            }
        }
        else
        {
            if (!statusText.transform.parent.GetComponent<Canvas>().enabled &&
!status.Contains("Opening"))
            {
                // Tutti i componenti sono stati importati correttamente
                Debug.LogWarning("Done with the streaming!");
                streamingFromServer=false;
                InvokeRepeating("ProcessModels", .5f, 3f);
                //CancelInvoke("CheckStreamingStatus");
            }
        }
    }
```

```
    public void ProcessModels()
    {
        // Per ogni figlio di questo oggetto
        foreach (Transform child in transform)
        {
            // controllo che non sia nelle lista permanente e non sia già stato aggiunto
alla lista dei nuovi
            if (!newChildren.Contains(child.gameObject) &&
!startingChildren.Contains(child.gameObject))
            {
                toBeChecked = child.gameObject;

                // controllo se è stabile
                CheckStable();
                if (isChildCountStable)
                {
                    // se lo è, lo aggiungo alla lista di quelli nuovi
                    newChildren.Add(child.gameObject);
                    currentlyLoading.text = "Processing material...\n(this may take a
while)";
                    utils.GetComponent<LevelManager>().refGroup = child.gameObject;
                    utils.GetComponent<LevelManager>().CheckLevels();
                    isChildCountStable = false;
                }
            }
        }

        // per ogni oggetto nuovo
        foreach (GameObject child in newChildren)
        {

            if (!viewManager.views.Contains(child))
            {
                viewManager.views.Add(child);
            }

            // inizio a contare il numero di figli
            int counter = 0;
            callComplete = false;
            if (child != null)
            {
                startedImporting = true;
            }
            //Debug.LogWarning("Processing " + child.name + ": " +
child.transform.childCount);

            // ciclo tra i suoi figli
            foreach (Transform elem in child.transform)
            {

                // se hanno un mesh renderer
                if (elem.gameObject.GetComponent<MeshRenderer>() != null)
                {
                    // assegno il materiale corretto
                    //if (elem.gameObject.GetComponent<MeshRenderer>().material.name !=
materialToAssign.name + " (Instance)")
                    //{
                    //    elem.gameObject.GetComponent<MeshRenderer>().materials = new
Material[] { };
                    //    elem.gameObject.GetComponent<MeshRenderer>().material =
materialToAssign;
                    //}
                    if
(!Array.Exists(elem.gameObject.GetComponent<MeshRenderer>().materials, x =>
x.name.Contains(materialToAssign.name)))
```

```
                    {
                        Material[] newMats = new
Material[elem.gameObject.GetComponent<MeshRenderer>().materials.Length];
                        int n = 0;
                        foreach (var mat in
elem.gameObject.GetComponent<MeshRenderer>().materials)
                        {
                            if (!mat.name.ToLower().Contains("glass"))
                            {
                                if (!mat.name.Contains(materialToAssign.name))
                                {
                                    newMats[n] = materialToAssign;
                                }
                            }
                            else
                            {
                                newMats[n] = mat;
                                newMats[n].color = Color.clear;
                            }
                            n++;
                        }
                        elem.gameObject.GetComponent<MeshRenderer>().materials = newMats;
                    }

                    // e lo script di controllo
                    if (elem.gameObject.GetComponent<TwoPlanesCuttingController>() == null)
                    {
                        elem.gameObject.AddComponent<TwoPlanesCuttingController>();
                        elem.GetComponent<TwoPlanesCuttingController>().planeX1 = planeX1;
                        elem.GetComponent<TwoPlanesCuttingController>().planeX2 = planeX2;

                        elem.GetComponent<TwoPlanesCuttingController>().planeY1 = planeY1;
                        elem.GetComponent<TwoPlanesCuttingController>().planeY2 = planeY2;

                        elem.GetComponent<TwoPlanesCuttingController>().planeZ1 = planeZ1;
                        elem.GetComponent<TwoPlanesCuttingController>().planeZ2 = planeZ2;

                        elem.GetComponent<TwoPlanesCuttingController>().parent = child;

                        elem.GetComponent<TwoPlanesCuttingController>().clipMGR =
this.gameObject;
                    }

                    // e lo probuilderizzo
                    if (elem.gameObject.GetComponent<MakePrimitiveEditable>() == null)
                    {
                        elem.gameObject.AddComponent<MakePrimitiveEditable>();
                    }

                    // e se è una decalcomania, scarico la relativa immagine
                    try
                    {
                        if
(elem.gameObject.GetComponent<Metadata>().GetParameter("Family").Contains("Decal"))
                        {
                            string url =
elem.gameObject.GetComponent<Metadata>().GetParameter("URL");
                            string convertedUrl =
utils.GetComponent<DownloadImg>().ConvertURL(url);
                            //Debug.LogWarning("URL: " + convertedUrl);

StartCoroutine(utils.GetComponent<DownloadImg>().DownloadImage(convertedUrl,
elem.gameObject));
                            if (elem.gameObject.GetComponent<FIxUV>() == null)
                            {
                                elem.gameObject.AddComponent<FIxUV>();
```

```
                        }
                    }
                }
                catch { }

            }
            counter++;
        }

        // se il contatore ha raggiunto il numero totale di figli, ho finito con questo
        if (counter == child.transform.childCount)
        {
            //Debug.LogWarning(child.name + " has " + counter.ToString() + " and will
be removed");

            // e lo sposto nell'altra lista
            startingChildren.Add(child);
            newChildren.Remove(child);
        }

    }

    if(newChildren.Count == 0 && startedImporting)
    {
        if (transform.childCount == startingChildren.Count)
        {
            callComplete = true;
            Debug.LogWarning("All done!");
            //CancelInvoke();
        }
    }
}


int latestChildCount = 0;
public void CheckStable()
{
    int cc = toBeChecked.transform.childCount;
    //Debug.LogWarning("----\nChecking " + toBeChecked.name);
    if (cc > 0)
    {
        if (cc > latestChildCount)
        {
            //Debug.LogWarning("Last checked: " + latestChildCount.ToString() + "\nNow
has: " + cc.ToString());
            latestChildCount = cc;
            isChildCountStable = false;
        }
        else
        {
            latestChildCount = 0;
            //Debug.LogWarning(toBeChecked.gameObject.name + " is stable: " +
cc.ToString());
            isChildCountStable = true;
        }
    }
}

// Update is called once per frame
void Update()
{

    if (sectionsOn)
    {
        _sectionsOn = sectionsOn.gameObject.activeSelf;
    }
```

```
        if (loadingScreen != null)
        {
            if (!callComplete && streamingFromServer)
            {
                loadingScreen.SetActive(true);
            }
            if (callComplete)
            {
                loadingScreen.SetActive(false);
            }
        }
    }
}
```

A3. Importazione delle immagini in Unity

A3.1 Creazione delle geometrie 3D per le decalcomanie in Dynamo

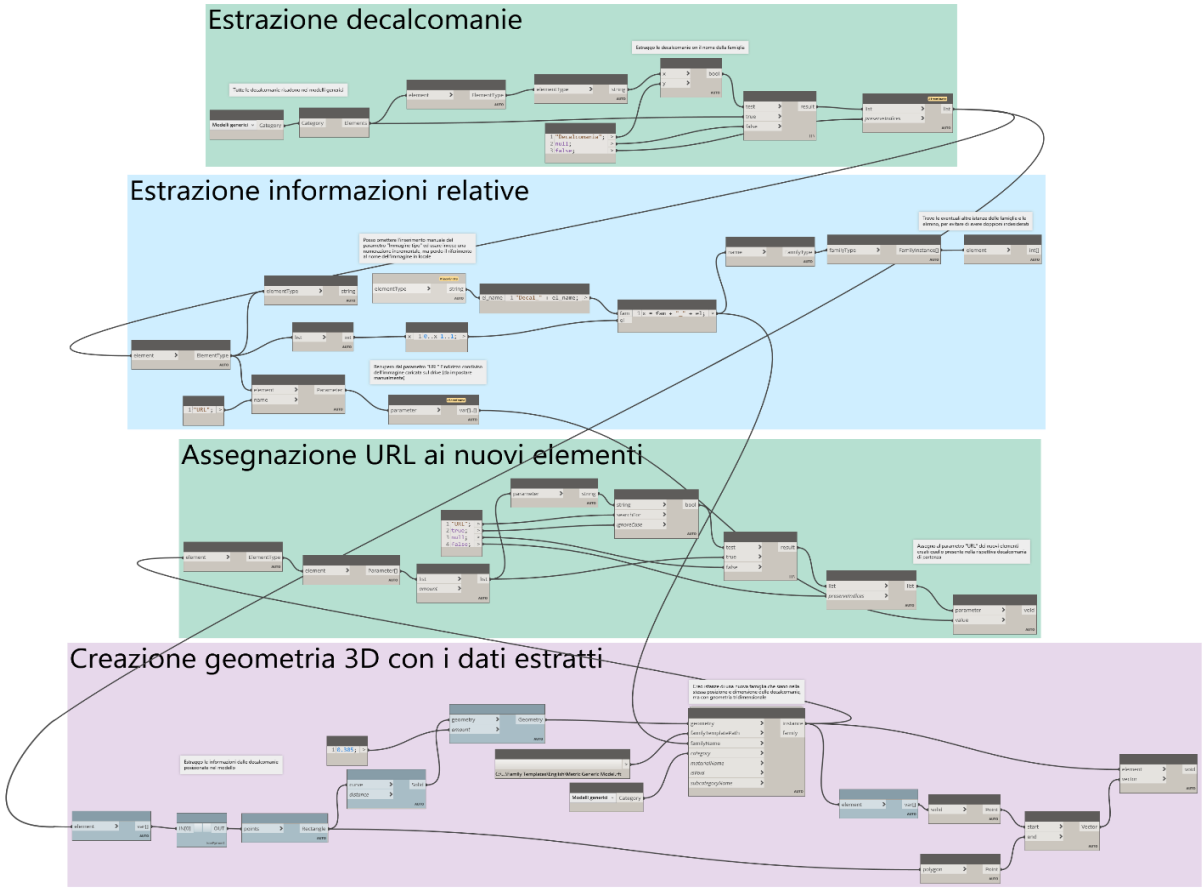

Figura 19: Algoritmo in Dynamo per l'esportazione delle decalcomanie

A3.2 Identificazione delle decalcomanie e download delle immagini

```
public class DownloadImg : MonoBehaviour
{
      if
      (elem.gameObject.GetComponent<Metadata>().GetParameter("Family").Contains("Decal"))
                              {
                                  string url =
      elem.gameObject.GetComponent<Metadata>().GetParameter("URL");
                                  string convertedUrl =
      utils.GetComponent<DownloadImg>().ConvertURL(url);
                                  //Debug.LogWarning("URL: " + convertedUrl);

      StartCoroutine(utils.GetComponent<DownloadImg>().DownloadImage(convertedUrl,
      elem.gameObject));
                                  if (elem.gameObject.GetComponent<FIxUV>() == null)
                                  {
                                      elem.gameObject.AddComponent<FIxUV>();
                                  }
                              }

    public string ConvertURL(string url)
    {
        string id;

        if(url == null) { return null; }
        else if(url != string.Empty)
        {
            id = url.Split(new string[] { "file/d/" },
StringSplitOptions.None)[1].Split('/')[0];

            url = "https://drive.google.com/uc?id=" + id;
        }

        return url;
    }

    public IEnumerator DownloadImage(string MediaUrl, GameObject gameObject)
    {
        UnityWebRequest request = UnityWebRequestTexture.GetTexture(MediaUrl);
        yield return request.SendWebRequest();
        Debug.LogWarning("URL: " + MediaUrl);
        Debug.LogWarning("Req: " + request.result);
        gameObject.GetComponent<Renderer>().material.mainTexture =
((DownloadHandlerTexture)request.downloadHandler).texture;
        gameObject.GetComponent<Renderer>().material.color = Color.white;
        gameObject.GetComponent<Renderer>().material.SetFloat("_Metallic", 0f);

    }

}
```

A3.3 Sistemazione UV dei piani una volta applicata la texture

```
public class FIxUV : MonoBehaviour
{
    void Start()
    {
        ProBuilderMesh pb = gameObject.GetComponent<ProBuilderMesh>();
        foreach (var face in pb.faces)
        {
            face.manualUV = false;
            Debug.LogWarning(face.uv.ToString());
            face.uv = new AutoUnwrapSettings()
            {
                fill = AutoUnwrapSettings.Fill.Stretch,
                offset = Vector2.zero,
                scale = Vector2.one,
                rotation = 0f,
                anchor = AutoUnwrapSettings.Anchor.MiddleCenter
            };
        }
        pb.Refresh();
    }
}
```

A4. Gestione delle Fasi Storiche

A4.1 Controllo del modello attivo in base alla fase storica attualmente selezionata

```
public class ViewManager : MonoBehaviour
{

    public List<GameObject> views;
    public TMPro.TMP_Dropdown dropDown;
    public string currentView;
    public List<string> alreadyAdded;
    int indexOfDefault = -1;
    TMPro.TMP_Dropdown.OptionData def;

    void Update()
    {
        foreach (GameObject view in views)
        {
            string name = view.GetComponentInChildren<Metadata>().GetParameter("View");
            view.name = name;

            if(!alreadyAdded.Contains(name))
            {
                dropDown.AddOptions(new List<string> { name });
                alreadyAdded.Add(name);
            }
            currentView = dropDown.options[dropDown.value].text;
```

```
            if(dropDown.options[dropDown.value].text.Contains(name))
            {
                view.SetActive(true);
            }
            else
            {
                view.SetActive(false);
            }
        }
        if (dropDown.options.Count > 0)
        {
            if (!dropDown.options[0].text.Contains("{3D}"))
            {
                foreach (var opt in dropDown.options)
                {
                    if (opt.text.Contains("{3D}"))
                    {
                        indexOfDefault = dropDown.options.IndexOf(opt);
                        def = opt;
                    }
                }
                if (indexOfDefault != -1)
                {
                    TMPro.TMP_Dropdown.OptionData old0 = dropDown.options[0];
                    dropDown.options[0] = def;
                    dropDown.options[indexOfDefault] = old0;

dropDown.transform.Find("Label").GetComponent<TMPro.TextMeshProUGUI>().text = def.text;
                }
            }
        }
    }
}
```

BIBLIOGRAFIA

Arletti, Natascia, Rachele Bernardello, Paolo Borin, Simone Fatuzzo, Alfonso Garuti, Andrea Giordano, Gianmario Guidarelli, et al. 2022. *Il Principe e La Sua Chiesa*. Edited by Paolo Bonacini, Francesca Portanova, Alice Previte, and Alessandro Vicenzi. Modena: Franco Cosimo Panini.

Biagini, Carlo, Pietro Capone, Vincenzo Donato, and Nora Facchini. 2016. "Towards the BIM Implementation for Historical Building Restoration Sites." *Automation in Construction* 71. https://doi.org/10.1016/j.autcon.2016.03.003.

Bruno, N., and R. Roncella. 2018. "A Restoration Oriented HBIM System for Cultural Heritage Documentation: The Case Study of Parma Cathedral." In *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*. Vol. 42. https://doi.org/10.5194/isprs-archives-XLII-2-171-2018.

Buhammood, A. H., Henry Abanda, Peter Garstecki, M. B. Manjia, Chrispin Pettang, and Abdulrasheed Madugu Abdullahi. 2020. "Coupling BIM and Game Engine Technologies for Construction Knowledge Enhancement." *International Journal of Gaming and Computer-Mediated Simulations* 12 (4). https://doi.org/10.4018/IJGCMS.2020100103.

Chen, Lin, Xiaolong Chen, and Lingyun Lang. 2022. "Building Information Protection Method of Urban Historical Features Based on BIM Technology." *Advances in Multimedia* 2022. https://doi.org/10.1155/2022/8998225.

Chernick, Adam, Christopher Morse, Steve London, Tim Li, David Ménard, John Cerone, and Gregg Pasquarelli. 2021. "On-Site BIM-Enabled Augmented Reality for Construction." In *Proceedings of the 2020 DigitalFUTURES*. https://doi.org/10.1007/978-981-33-4400-6_5.

Ma, Yu Pin. 2022. "Improved Interaction of BIM Models for Historic Buildings with a Game Engine Platform." *Applied Sciences (Switzerland)* 12 (3). https://doi.org/10.3390/app12030945.

Pour Rahimian, Farzad, Saleh Seyedzadeh, Stephen Oliver, Sergio Rodriguez, and Nashwan Dawood. 2020. "On-Demand Monitoring of Construction Projects through a Game-like Hybrid Application of BIM and Machine Learning." *Automation in Construction* 110. https://doi.org/10.1016/j.autcon.2019.103012.

Talaverano, Rafael Martín, José Ignacio Murillo Fragero, and María de los Ángeles Utrero Agudo. 2021. "Reflections and Criteria Related to the Creation of BIM Models of Historical Buildings." In *Arqueologia de La Arquitectura*. https://doi.org/10.3989/ARQ.ARQT.2021.005.