Università degli Studi di Bologna

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Informatica Triennale

Tecniche di modellazione procedurale per la ricostruzione di ambienti urbani: il caso di Bologna medievale, dall'acquisizione delle fonti storiche al rendering per un cortometraggio 3D

Relatore:
Chiar.mo Prof. Fabio Panzieri
Corelatore:
Ing. Antonella Guidazzoli

Tesi di Laurea di:
Marco Mantoanelli

III Sessione

Anno Accademico 2012–2013

. . . a chi mi pare :)

Indice

Elenco delle figure

ï»¿

Capitolo 1

Introduzione

Il presente lavoro di tesi illustra il processo di produzione di scenari urbani tridimensionali filologicamente accurati (ovvero supportati da fonti e valorizzati da un pool di esperti tra archeologi e storici), realizzati tramite modellazione procedurale con il software City Engine, unico strumento proprietario all'interno della pipeline Open definita dal Cineca per il progetto Apa, Museo della Città. Finalità del progetto era la realizzazione di un cortometraggio animato stereoscopico che raccontasse la storia di Bologna attraverso i secoli incorporando i numerosi modelli, frutto di studi e ricerche di dottorato di cui il Cineca (insieme al CNR Itabc e ad altri partner) era già in parte in possesso.

Ho avuto occasione di partecipare al progetto inizialmente come tirocinante, modellando in Blender, il principale software impiegato per la realizzazione del cortometraggio. In seguito, il mio personale interesse per la computer graphics mi ha condotto a prolungare il mio contributo come tesista e mi sono stati affidati due set: la Bologna attuale e la Bologna medievale, realizzati con City Engine.

City Engine consente di creare modelli geometrici di dimensioni considerevoli (i.e un'intera città) evitando di partire dai singoli edifici ma andando a definire regole che possono essere gestite da script in un linguaggio (procedurale) atto allo scopo. Modelli così costruiti possono essere particolarmente accurati dal punto di vista filologico, ovvero a partire dalle fonti (georeferenziazione e/o fotopiani), pur mantenendo un processo di realizzazione relativamente semplice.

Questo tipo di ricostruzione è particolarmente indicato per ricreare l'aspetto che un determinato territorio aveva in epoche passate a partire dai dati supportati dalle fonti e di fatto entra a far parte di quella che oggi prende in nome di Archeologia Virtuale (Virtual Archaeology).

L'impiego di ricostruzioni 3D filologicamente accurate per la navigazione di

ambientazioni storiche a fini divulgativi e/o di studio è pratica investigata in numerosi progetti di Virtual Heritage. Operazione originale del progetto Apa è stato rendere tali scenari set di un cortometraggio di animazione e quindi innalzare le location a pari se non maggiore importanza rispetto al protagonista dell'azione, l'etrusco Apa. Il filologico incontra l'emozionale per trasmettere cultura attraverso la partecipazione alle vicende del protagonista.

Il mio contributo, all'interno della pipeline, si inserisce nel momento in cui dovevano essere ultimati i modelli per poter realizzare i mockup, ovvero le immagini che avrebbero rappresentato il punto di riferimento per il mood grafico che avrebbe caratterizzato il corto e che insieme dovevano essere validate dagli esperti in quanto modelli filologicamente accurati. Ogni più piccola richiesta di cambiamento avrebbe potuto comportare una serie di modifiche che si sarebbero ripercosse a cascata sulla geometria di uno o più modelli. Si trattava di scenari urbani di Bologna in varie epoche: attuale, etrusca, romana, medievale e rinascimentale.

Figura 1.1: Mock-up di Enrico Valenza della Bologna turrita del 1200

Di fondamentale importanza era considerato il set medievale, in quanto si proponeva di illustrare un aspetto di Bologna inedito, con le oltre novanta torri che rendevano la vista della città molto suggestiva (Fig. 1.1), andando a incentivare l'aspetto emozionale, canale privilegiato dal progetto per divulgare cultura.

La modellazione procedurale, come strumento di Virtual Archaeology, rappresenta non solo un valido strumento per la ricostruzione di scenari ampi come per esempio una città, ma anche un'occasione di studio relativamente ai metodi stessi di ricostruzione. In particolare, l'intento del Cineca era definire una mo-

dalità di produzione degli asset (qualsiasi oggetto 3D all'interno di una scena, dal set dressing all'intero set), che ne favorisse il riutilizzo. La modellazione, infatti, coinvolge un processo lungo e costoso in termini di tempo e risorse umane. D'altro canto a un oggetto d'arte, in quanto segno, possono essere associate più *interfacce* per la sua esplorazione.

Parte di questo lavoro è pertanto da intendersi come documentazione di una modellazione procedurale inserita in una pipeline di produzione che ha integrato aspetti cinematografici tradizionali (scrittura di un soggetto, produzione di un breakdown e del successivo storyboard e parallelamente realizzazione dei concept 2D e dei mockup) con i vincoli filologici e la peculiarità di una modellazione volta al realizzo.

City Engine ha consentito, attraverso l'elaborazione procedurale di mappe reali e dati gis, di ricostruire interi spazi urbani a contorno dei modelli filologici provenienti da progetti di ricerca (per esempio NuMe), interagendo a stretto contatto con Blender per integrare le ricostruzioni con i livelli di dettaglio richiesti. In particolare, la tecnica adottata per la ricostruzione del modello della città di Bologna medievale ha dato modo di realizzare una pipeline originale e interessante, avendo consentito di produrre la scena a partire dai fotopiani di un modello filologico in gesso conservato nel museo Civico Medievale della città.

Pur trattandosi di software proprietario, City Engine è potuto entrare a far parte della pipeline di produzione del progetto perché è stato possibile esportarne i risultati in formato Wavefont OBJ, a sua volta importabile dentro a Blender.

Questa tesi si divide in due parti: la prima, ovvero il Capitolo 2, presenta un breve sunto dello stato dell'arte relativo all'impiego di software di modellazione 3D a supporto della Virtual Heritage e Virtual Archaeology. In tale contesto, descriveremo anche la pipeline integrata adottata dal Cineca, illustrandone principali software e accorgimenti. La seconda parte, composta dai capitoli 3 e 4, scenderà nel dettaglio del lavoro svolto a partire dal mio tirocinio, illustrando la realizzazione dei due set a me affidati. Verranno evidenziati i vari passaggi e il metodo utilizzato contestualmente alle finalità del progetto. Il Capitolo 5 fornirà, infine, le conclusioni, evidenziando alcuni aspetti inerenti la modellazione procedurale a supporto della realizzazione di modelli 3D filologicamente corretti che possano essere impiegati in altri progetti di Virtual Heritage. Si accennerà all'impegno del Cineca per il Virtual Heritage che procede all'interno del progetto europeo Vmust[1], per cui Apa è un caso di studio. Gli scenari modellati in maniera procedurale, infatti, rappresentano un ampio archivio per

[1] `http://www.v-must.net/`

estrapolare modelli singoli riutilizzabili in differenti contesti. La Bologna romana, per esempio, e' attualmente oggetto di un progetto patrocinato dal Museo Civico Archeologico di Bologna e che prevede la ricostruzione di parti del set all'interno di 3DMetaversity, un open simulator alternativo a Second Life, per la produzione di un cortometraggio storico con tecnica machinima.

Capitolo 2

Stato dell'arte

2.1 Virtual Heritage e modellazione 3D

Virtual Heritage (VH) è un termine usato per descrivere prodotti dell'ingegno umano che abbiano a che fare con la realtá virtuale (Virtual Reality) e insieme con la cultural-heritage (CH)[Rou]. In generale, CH e VH hanno significati distinti: CH fa riferimento a *Oggetti e siti con valore archeologico, estetico e storico* mentre la VH riguarda una ricostruzione di questi oggetti e siti tramite un dominio tecnologico (ICT), generalmente ricostruzioni 3D e tecnologie di computer graphics.

La realtá virtuale come mezzo per la visualizzazione e insieme come sistema interattivo di navigazione all'interno di uno scenario che riproduca il contesto storico originario, apre nuovi metodi percettivi per la fruizione dell'opera d'arte non solo in termini di ricerca ma anche e soprattutto di divulgazione. Determina, cioé, un contesto immersivo per l'utente che diventa sempre piú protagonista della propria esperienza culturale.

2.1.1 Virtual Archaeology

Il termine *Archeologia Virtuale* nasce con Paul Reilly [Rei90] che nel 1990 lo propone in riferimento all'uso di modelli tridimensionali di monumenti e oggetti, tracciando di fatto i contorni di una vera e propria disciplina. Egli ne definí, infatti, un ambito di ricerca ben preciso articolando la registrazione degli scavi con le possibilitá di ricrearli virtualmente attraverso l'impiego di ipertesti, multimedia e modellazione tridimensionale.

Questa metodologia, inizialmente pensata al solo fine della ricerca archeologica, introduce uno strumento estremamente potente poiché basasto sulla vista,

il nostro organo percettivo più importante, per l'accesso a dati talvolta di difficile consultazione e/o comprensione se non visti nella loro totalitá o nel contesto in cui sono nati (ricostruzioni storiche del territorio).

Ci si é pertanto ben presto resi conto del valore che un simile tipo di accesso alle informazioni poteva offrire non solo alla ricerca ma anche alla divulgazione, incorporando il dato in ricostruzioni 'virtuali' filologiche (ovvero supportate dalle fonti e validate dagli eseprti) che trasformassero il percorso archeologico in uno spazio immersivo, assumendo le caratteristiche di una visita, più che di uno studio nel significato tradizionale del termine, ovvero connesso alla metafora testuale.

La parola *virtuale* è stata nel tempo utilizzata in maniera allargata, includendo in generale ricostruzioni 3D, comprese le immagini (o rendering) dei modelli e perdendo quel fondamentale legame con il campo delle visualizzazioni interattive [Gil99].

Negli ultimi anni, l'Archeologia Virtuale sta guadagnando sempre più interesse nella comunità internazionale, dando origine a un dibattito su una sua eventuale integrazione nei curricula accademici e recuperando il suo valore interattivo, ovvero di strumento di navigazione di luoghi non raggiungibili per questioni di spazio e tempo permettendo di osservare l'ambiente in modo completo grazie ad una ricostruzione filologica digitale basata sulle fonti. La modalità visiva intrinseca al mezzo favorisce la formulazione di nuove domande, secondo il valido principio espresso dal prof. Antinucci per cui *Il visivo si spiega da solo*, ovvero entità per loro natura legate alla vista sono meglio indagabili e acquisibili per mezzo di strumenti che a loro volta coinvolgano tale senso [Ant10].

L'archeologia virtuale consente di interpretare, comprendere e comunicare il patrimonio culturale attraverso un processo di acquisizione, ricostruzione e verifica continua. Non si tratta, infatti, di semplice ricostruzione di scenari archeologici ma è innanzi tutto un processo cibernetico di simulazione attraverso cui è possibile ricombinare oggetti ed ecosistemi per delineare differenti potenziali gradi di informazione (si veda, per esempio, il progetto Aquae Patavinae VR a Montegrotto [SPC11]).

Un tale approccio richiede la definizione di una modalità multidisciplinare, aperta e trasparente di gestione dei progetti. Il percorso passa attraverso l'utilizzo di diversi strumenti e tecniche, a partire dall'acquisizione dei dati sul campo (DGPS, Scanner Laser, fotogrammetria), dall'elaborazione 3D, dalla realizzazione di sistemi spaziali, fino a giungere alla trasformazione di tali dati in *informazione*.

Differenti possono essere gli strumenti e gli stili di comunicazione: dalle

applicazioni filmiche e multimediali alle applicazioni immersive di Realtà Aumentata; dalla narrazione virtuale fino ad ambiti condivisi di multiutenza; dalle simulazioni complesse alla vita artificiale.

2.2 Pipeline integrata di produzione del progetto Apa

Avvalendosi della massima che "Il visivo si spiega da solo", il Cineca VisIT Lab ha condotto la giá consolidata esperienza in ambito di visualizzazione scientifica (nel quale questa massima è sovrana) nell'ambito del CH, avvalendosi dei numerosi modelli di cui era giá in possesso, frutto di precedenti collaborazioni con l'Università, ma anche con il CNR Itabc, il settore Cultura del comune e i Musei civici.

L'idea é stata quella di tessere intorno a queste ricostruzioni una narrazione che fungesse da espediente per trasmettere allo spettatore il contenuto culturale con il minimo sforzo cognitivo ma il massimo risultato in termini di apprendimento significativo nell'accezione costruttivista del termine [1].

In questo contesto si inserisce il progetto "Apa, Museo della Città" che fa capo all'iniziativa Genus Bononiae Musei nella Cittá[2], patrocinata dalla Fondazione Cassa di Risparmio di Bologna. L'intento condiviso è quello di riunire in un percorso interculturale comune (artistico e museale) alcuni edifici del centro storico di Bologna, restaurati e recuperati all'uso pubblico.

2.2.1 Il progetto Apa, Museo della Città

Nel 2010 viene chiesto al Cineca di realizzare un filmato 3D stereoscopico che racconti la storia di Bologna nelle sue tappe fondamentali e che fin da subito si decide di porre al centro del percorso museale del palazzo che sarà dedicato appunto alla storia della città: palazzo Pepoli vecchio. Il tema del filmato viene fissato nella frase: *Il big bang della storia di Bologna* e l'importanza comunicativa che viene ad esso attribuita è fin da subito sottolineata dalla commissione in parallelo (sempre al Cineca) della sala immersiva che sarà interamente dedicata alla sua proiezione (Fig. 2.1).

Il progetto ha impegnato per due anni un team di circa venti persone eterogenei per gruppo di appartenenza e professionalità. Lo staf del Cineca si è,

[1] `coopera.francibb.it/2011/costrutt-mo-e-appr-significativo`
[2] `http://www.genusbononiae.it/`

Figura 2.1: Progetto della sala immersiva realizzaro dal Cineca

infatti, avvalso della collaborazione dell'artista Enrico Valenza[3], Art Director del cortometraggio, Spark D.E.[4] per l'animazione e Lilliwood[5] per la stereoscopia, a cui si sono aggiunti i professionisti Gianpaolo Fragale[6] per l'animazione di Apa e Riccardo Covino[7] per il Lighting; CNR Itabc ha contribuito alla realizzazione di parte dei set. Il cortometraggio stereoscopico 3D così prodotto è attualmente proiettato a ciclo continuo a palazzo Pepoli, il museo della città di Bologna.

Si è trattato di una sfida implementativa, per altro giá altre volte raccolta dal Cineca [Be10], la ricostruzione di ambientazioni 3D accurate per il Virtual Heritage, a cui si è aggiunta la sfida comunicativa di impiegare tali ricostruzioni non solo per fini accademici o di ricerca ma anche come set per un cortometraggio, abbracciando il motto "dal filologico all'emozionale". Il film, cioè, sfrutta la capacità della narrazione per immagini di coinvolgere lo spettatore nelle vicende del protagonista, l'etrusco Apa fuoriuscito da una situla nel museo Archeologico e catapultato alla scoperta di Bologna attraverso i secoli, per passare cultura divertendo con un prodotto di edutainment (Fig. 2.2).

Proprio l'esperienza del Cineca nella ricostruzione archeologica ha consentito di selezionare gli strumenti piú adatti per affrontare quella che si è ben presto

[3] `www.enricovalenza.com`
[4] `www.sparkde.com/intro.html`
[5] `www.lilliwood.eu`
[6] `www.gianpaolofragale.com`
[7] `www.riccardocovino.it`

rivelata una duplice sfida: l'individuazione del software più adatto non solo per la realizzazione dei modelli a partire dalle fonti (software di modellazione 3D e di animazione vero e proprio ma anche di scansione laser, di modellazione dei terreni, di gestione di dati GIS, etc.) ma anche per gestire la convivenza di modelli provenienti da contesti differenti che dovevano confluire in un unico set.

Figura 2.2: Le varie fasi di creazione di Apa

Le ambientazioni assumono, dunque, importanza pari al personaggio protagonista, l'etrusco Apa, che funge da filo conduttore del racconto.

2.2.2 La pipeline di produzione integrata

L'impegno implementativo si é declinato in due forme: da una parte, la gestione della pipeline tradizionale di progetto che ha richiesto non solo di selezionare il software piú adatto ma di farlo anche dialogare nei vari formati in uscita.

I programmi, per esempio, per la modellazione del terreno (Grass GIS, [8]) e della cittá (City Engine, [9]) nelle varie epoche storiche cosí come dati provenienti da sistemi informativi o acquisiti tramite scansione laser (per esempio, il Nettuno) dovevano essere integrati nei set insieme ai numerosi modelli di cui era giá in possesso il Cineca.

[8] `grass.osgeo.org`
[9] `www.esri.com/software/cityengine`

Strumenti di dialogo e collaborazione sono stati inoltre necessari per coordinare le professionalità che si sono unite al Cineca nell'obiettivo comune. Si voleva, cioé, consentire a ciascun gruppo di produrre internamente gli asset (qualsaisi elemento costituente uno scenario 3D) con il software piú consono alle esigenze della specifica IT interna astraendo, invece, a piú alto livello le fasi della pipeline di produzione.

2.2.3 La scelta Open Source

Per il progetto Apa, Cineca definisce una pipeline di produzione interamente Open Source, in cui City Engine compare come unico prodotto proprietario per cui si è reso necessario acquistare un pool di licenze. Come già accennato, la pipeline di produzione dei singoli elementi che sono poi confluiti a formare scene più complesse sono stati realizzati con un processo che ha richiesto la cooperazione di software differenti ed eterogeni tra di loro, per la maggior parte dei quali è stato possibile individuare un'alternativa Open altrettanto valida. Strumenti come Blender[10] innanzi tutto, ma anche The Gimp[11], Grass Gis e molti altri hanno consentito di affrontare in maniera appropriata le problematiche specifiche di progetto quali integrazione della gestione dei vincoli filologici all'interno della pipeline di produzione tradizionale e una realizzazione dei modelli volta al riutilizzo.

L'impiego dell'Open Source a inoltre aggiunto a peculiaritá quali:

- Possibilitá di far fronte alle difficoltá aggiungendo le nuove funzionalitá necessarie ottima reperibilitá e buona compatibilitá
- Possibilitá di fare affidamento su di una ricca community di sviluppatori;

funzionalitá quali:

- La possibilitá di esportare in numerosi formati insieme a una buona compatibilitá mantenuta dall'open source ha consentito di affrontare il probelma della comunicazione tra software.
- La possibilitá di esportare in formati compatibili ha inoltre consentito di integrare nella catena di produzione open anche software proprietario come City Engine, di cui non si é potuto fare a meno poiché non esiste ancora alternativa open altrettanto valida.

[10] `www.blender.org`
[11] `http://gimp.linux.it`

L'impiego di formati aperti garantisce di essere sempre in grado di riutilizzare il modello.

La scelta Open si è, inoltre, rivelata valido supporto alla collaborazione nel processo produttivo. Tramite Google docs, è stato possibile utilizzare la formula Cloud per gestire il ciclo di vita degli asset indipendentemente dai singoli strumenti di produzione. Un Blog ha favorito la condivisione di idee ed esperienza[12].

2.2.4 Blender

Blender é il software al centro della pipeline di produzione del cortometraggio. Si tratta di un prodotto open-source di modellazione 3D che consente anche rigging, animazione, rendering, compositing e sculpting 3D. Offre, inoltre, vari strumenti per il mapping UV, la simulazione di particelle e fluidi, e la possibilitá di creare applicazioni/videogames 3D con il suo game engine integrato. È disponibile per i più diffusi sistemi operativi con prestazioni che lo pongono al pari di programmi proprietari piú famosi quali Cinema4D, 3DStudioMax o Maya.

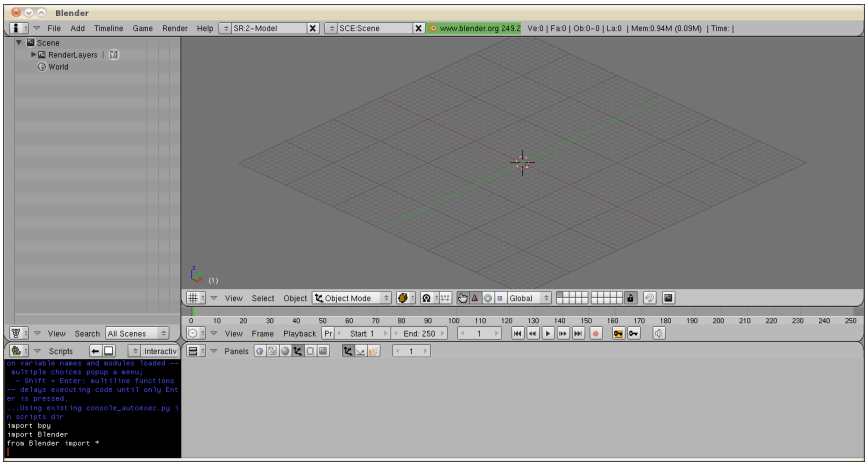

Figura 2.3: La vecchia interfaccia di Blender, cambiata dopo la release 2.5

La versione di Blender impiegata per il progetto è stata la 2.49b, la più stabile ai tempi in cui è partita la lavorazione degli asset e che si è deciso di mantenere fino alla fine, nonostante un'interfaccia grafica considerevolmente

[12] `https://rvn05.plx.cineca.it:12001/php/MDC/portal/wordpress/?page_id=6`

ostica dal punto di vista dell'usabilitá e che sarebbe stata completamente ridisegnata a partire dalla versione 2.5 del prodotto, in uscite in prossimità della terminazione del corto.

Oggi, arrivati alla versione 2.66, Blender offre un motore di rendering path-tracing, unbiased interattivo chiamato Cycles che consente di visualizzare il risultato finale di modelli e materiali durante la creazione di un asset.

2.3 La modellazione procedurale

Durante la produzione di questo lavoro, si é reso necessario introdurre parti di modellazione procedurale che consentissero, a monte di un considerevole lavoro di pianificazione, dovuto anche al vincolo filolofico delle scene, di automatizzare la realizzazione di geometrie significativamente complesse come parti della città di Bologna in epoche ben precise. Il software individuato come più adatto per questo scopo è stato City Engine. Al momento della produzione (e credo si possa affermare anche ora) non esiste software Open che possa confrontarsi con le capacità di questo strumento.

Le motivazioni per l'utilizzo di questo tipo di modellazione vanno valutate tenendo conto degli aspetti che la caratterizzano: questo genere di modellazione, infatti, introduce un vincolo indissolubile con i parametri forniti inizialmente e la modellazione attraverso script. Questo impone una progettazione lunga e complessa. D'altro canto i vantaggi rispetto ad una modellazione manuale delle geometrie sono la possibilità di realizzare scenari complessi e di grandi dimensioni, controllati da un numero esiguo di input, la predisposizione per l'utilizzo di oggetti che si ripetono nello spazio e la facilità nella creazione di oggetti simili, o che presentano proprietà comuni.

Tuttavia, per quanto l'impiego di modellazione procedurale sembri naturale in ambito di creazione di ambienti urbani, essa non è nata per soddisfare l'esigenza di rappresentare delle città.

2.3.1 Origini

La modellazione procedurale, nel senso più stretto del termine, indica un procedimento attraverso il quale creare un oggetto virtuale tramite l'utilizzo di una serie di regole che fungono da guida e demandando il compito della generazione al calcolatore stesso che, seguendo le indicazioni fornite, si occupa di produrre il risultato voluto.

Forte di questa definizione, risulta naturale individuare come primo ideatore della modellazione procedurale, Benoit Mandelbrot, che con la geometria frattale ha dato l'idea di utilizzare istruzioni matematiche per generare oggetti grafici complessi.

Dalla rappresentazione di immagini astratte a quella di oggetti realistici attraverso modelli matematici, il passo é breve, e a questo ha contribuito Aristid Lindermayer.

Da Lindermayer a Müller

Il biologo Aristid Lindermayer, nel 1968, ideò per primo una teoria biologica attraverso la quale descrivere la struttura dei vegetali utilizzando delle grammatiche che, fornendo alcune regole di riscrittura e un assioma di partenza, con la ricorsione, permettessero di creare alberi di discendenza profondi a piacere su una stringa composta da una serie di caratteri di tipo finito ripetuti senza limiti. Questo sistema viene chiamato D0L-system (*Deterministic 0-context Lindermayer-system*) e rappresenta l'esempio più semplice di caso di Lindermayer-system.

Essendo stati concepiti come una teoria matematica, gli aspetti geometrici legati agli L-System andavano al di là degli obiettivi per cui erano stati creati e inizialmente non se ne compresero le potenzialità. Ciononostante, studi successivi dimostrarono come questa teoria, applicata alla geometria, potesse essere impiegata per ottenere dei frattali.

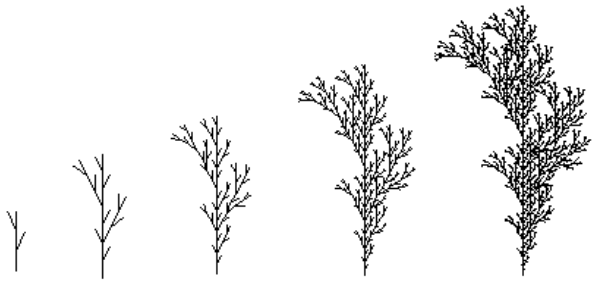

Figura 2.4: Lo sviluppo geometrico di un L-System

Queste idee, applicate alla grafica, consentivano di creare un'approssimazione dei frattali voluti con sorprendenti risultati nella realizzazione di figure del tutto simili a vegetali, confermando il lavoro svolto da Lindermayer (Fig.2.4).

Da questo presupposto, Pascal Müller, all'inizio degli anni 2000, ha rivisitato e fatto suoi questi concetti, applicandoli alla creazione di reticolati stradali e alla generazione delle geometrie da sfruttare in una pipeline per la costruzione virtuale di ambienti urbani in modo automatico e pilotato dai dati.

Su queste sue intuizioni, egli ha creato il software di modellazione procedurale per ambienti urbani deominato CityEngine, tuttora punto di riferimento nel mondo dei software di modellazione automatica.

2.3.2 City Engine

City Engine (CE) é un potente software proprietario sviluppato dalla Procedural Inc. [13], azienda fondata dallo stesso Müller. Il software è utilizzato in ambiti professionali di altissimo livello per la ricostruzione di ambienti urbani attraverso la modellazione procedurale. Le sue applicazioni trovano riscontri in numerosi ambiti, dalla visualizzazione scientifica all'entertainment, come dimostra la lunga lista di clienti che fa uso di questo prodotto e mostrati in Fig. 2.5

Figura 2.5: Elenco di alcuni tra i più importanti utilizzatori di CE

CE è disponibile su tutte le principali piattaforme: Linux, Microsoft Windows e Apple OSX.

Questo software nasce per rispondere all'esigenza di creare, in modo rapido e poco dispendioso, interi agglomerati urbani evitando la modellazione di ogni

[13] `www.procedural.com`

singolo edificio pur mantenendo un buon grado di controllo sull'aspetto e sul comportamento del software nella creaizone dello scenario.

I suoi principali ambiti applicativi sono la visualizzazione virtuale di ambienti urbani in svariati contesti: partendo dalla visualizzazione scientifica, passando per l'architettura, l'urbanistica, l'entertainment, la simulazione computerizzata, e molto altro ancora. Qualunque ambito richieda la realizzazione di uno scenario urbano virtuale, probabilmente necessita della modellazione e di conseguenza, data l'effettiva mancanza di scelta sia in ambiente opensource che commerciale, di City Engine. Si pensi, per esempio, all'impiego di tale software da parte della Pixar per la realizzazione del movie 'Cars 2'[14].

Fondamenti

Per generare l'ambiente urbano desiderato, City Engine assume che una città sia caratterizzata da tre elementi principali: il reticolato stradale (streetmap), i volumi degli edifici e lo stile delle facciate degli edifici stessi.

Per la creazione della streetmap, City Engine si basa su un Lindermayer System esteso, in grado di creare un reticolato realistico che si adatti ad una mappa (ad esempio, si può istruire il software in modo da non generare strade che attraversino corsi d'acqua segnati su una mappa utilizzata come parametro in input). La roadmap non deve presentare incongruenze come per esempio strade sovrapposte senza incroci o interrotte.

Pipeline di generazione La creazione di un ambiente urbano coerente e realistico, secondo l'algoritmo di Müller[YIHP01], prevede quattro diversi passaggi:

1. Creazione di una mappa stradale attraverso un L-System migliorato.

2. Divisione in lotti delle aree chiuse all'interno della mappa stradale mediante una suddivisione geometrica dei poligoni.

3. Generazione di stringhe descrittive la geometria mediante un algoritmo di tipo L-System applicando regole di riscrittura definite all'interno degli script CGA.

4. Creazione delle geometrie, con un parser sull'output del passaggio precedente, il software traduce le stringhe in geometria 3D completa di texture e materiali applicati.

[14] `http://www.escapestudios.com/cityengine-mapping-out-the-locations-on-pixar-s-cars-2/`

GUI L'interfaccia grafica di CE (Fig. 2.6), alla tradizionale barra degli strumenti, affianca un navigatore del workspace, che presenta i comandi principali con cui controllare e interagire all'interno del Viewport sottostante.

Per una migliore navigabilità all'interno delle strutture dati, vi sono due finestre di esplorazione, una (del workspace intero) che visualizza il filesystem localizzato alla sola cartella di lavoro, e un'altra che permette di accedere ai vari elementi del progetto corrente.

A supporto del Viewport, sulla destra, una finestra chiamata Inspector si occupa di visualizzare i parametri e i dati legati alla selezione corrente nella scena.

Ultima, ma non meno importante, la finestra degli script permette di effettuare modifiche ai file di script che contengono le regole create.

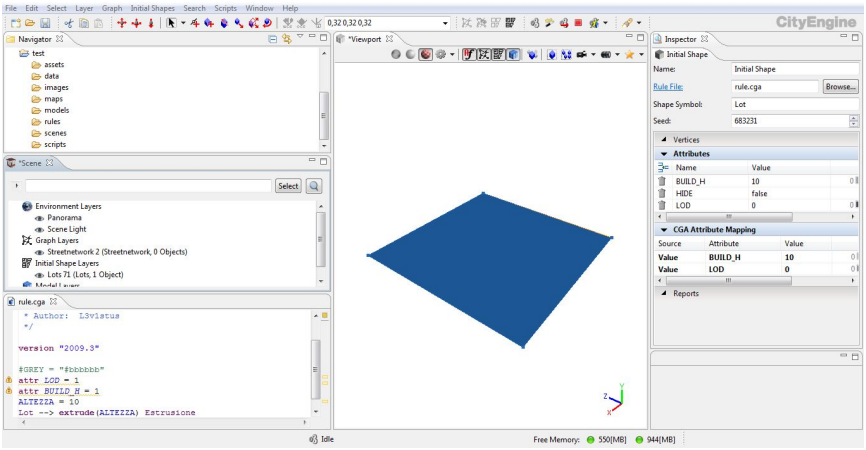

Figura 2.6: Interfaccia di City Engine

Concetti di base di CityEngine Per poter lavorare con CityEngine, occorre innanzitutto conoscere e capire quali sono i concetti astratti su cui si fonda l'intero software per lavorare.

Layer Ogni cosa, rappresentata all'interno del software è contenuta in un layer, o livello, il quale può essere di diversi tipi in base a cosa esso rappresenti. Ogni livello è nascondibile attraverso il navigatore del progetto in modo da poter eliminare dal Viewport informazioni inutili o che non si vogliano modificare (Fig. 2.7).

I layer si distinguono tra quelli che vengono creati automaticamente dal sistema per contenere shape e street network, e quelli che invece contengono informazioni utili alla generazione procedurale o alle elaborazioni sulle strade, questi ultimi vengono chiamati *Attribute Layer*.

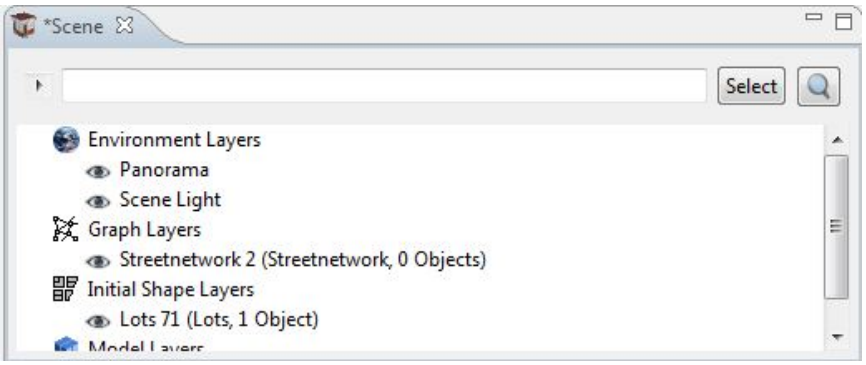

Figura 2.7: Diversi layer, rappresentati nel navigatore della scena di CE

Maps Un layer può essere "riempito" con una mappa, ovvero un'immagine generica che potrà successivamente essere utilizzata per pilotare in modo più accurato la generazione della geometria. Si può, ad esempio, definire che sopra alle aree di un determinato colore, gli edifici debbano essere tutti alti più di 100 unitá, oppure vincolare le strade a non potersi sovrapporre a certi colori (utile in fase di ricostruzione di uno scenario reale, per definire corsi d'acqua o ostacoli naturali).

Shape Geometricamente parlando, una shape altro non è che un poligono rappresentato nello spazio tridimensionale. Le shape e che identifica un'area "edificabile". Le shape sono il punto di partenza per qualunque generazione procedurale. Le regole, infatti, operano su di esse e non è possibile applicarle se la scena non possiede almeno una shape. Attraverso le operazioni definite tramite la grammatica CGA, che illutreremo nel prossimo paragrafo, le shape vengono elaborate e si ottiene la geometria finale. Un layer che rappresentanti delle shape viene definito *Shape Layer*.

Street networks Le street networks rappresentano, a tutti gli effetti, le arterie di una città, sia questa virtuale o reale, in quanto ne definiscono la forma

descrivendone porzioni chiuse, poligoni degli edifici e spazi tra essi. Ogni street network è formata da diverse strade ed incroci, identificati da segmenti e punti. Un layer rappresenti una serie di strade prende il nome di *Graph Layer*.

Comandi Viewport All'interno del Viewport, dove è visualizzata la scena su cui si sta lavorando, sono rese disponibili (per qualsiasi elemento tranne gli assi e lo sfondo) le tre trasformazioni geometriche tradizionali per i software di grafica: Traslazione, Rotazione e Scala (Fig.2.8).

Oltre a questo, il software fornisce metodi per creare e suddividere manualmente shape e strade, nel caso risultasse necessario effettuare modifiche puntuali.

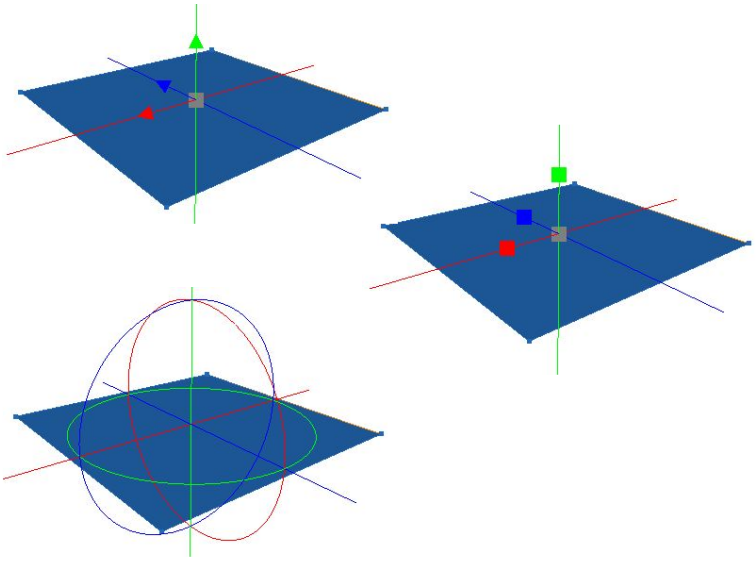

Figura 2.8: Le tre funzioni di trasformazione basilari della Viewport

La grammatica CGA Shape

Il vero cuore di CityEngine risiede nella sua grammatica CGA shape, dove CGA è l'acronimo di *Computer Generated Architecture*. Essa permette, attraverso la programmazione, di istruire il software in modo da applicare i risultati desiderati su larga scala, in poco tempo ed esercitando un controllo diretto su di essi. Attraverso la CGA si può definire una nuova grammatica formale, basata su un

alfabeto (o serie di simboli) e un insieme di regole di produzione che descrivono la geometria e le sue possibili varianti.

Shape Negli elementi della grammatica si trova il concetto di Shape che non deve essere confuso con le shape descritte nel precedente paragrafo. In questo caso, infatti, si tratta di una struttura dati differente e più articolata. Le due entità vengono, generalmente, distinte chiamando le prime *Initial Shape* (Fig. 2.9).

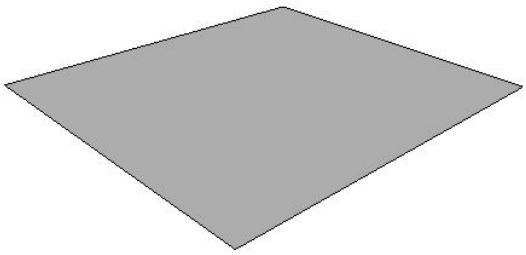

Figura 2.9: Una *InitialShape*, a cui farà riferimento una struttura dati *shape*

Una shape, intesa come struttura dati, è l'elemento base su cui si fonda la grammatica CGA shape. Possiede un nome (chiamato anche *Shape Symbol*) e consiste in una geometria vuota definita da un bounding box orientato, chiamato anche *scope*. Una shape comprende:

- **Shape symbol** Anche chiamato *Rule Name*. Si tratta del nome di una shape. È di fondamentale importanza in quanto, attraverso di esso, la regola definita con il nome corrispondente viene utilizzato per generare il successore della shape.
- **Parametri** Ogni shape può avere una lista di parametri ad esso associata. Ognuno di questi parametri può essere di tre tipi:
  - boolean
  - numerico (rappresentato da un double-precision float)
  - stringa
- **Attributi** Gli attributi contengono le informazioni numeriche e spaziali (inclusa la geometria) della shape. Durante l'applicazione delle regole,

questi attributi possono essere utilizzati e modificati dalle operazioni sulle shape. Possono essere definiti degli attributi personalizzati. Descriviamo di seguito quelli definiti automaticamente:

- *Geometria* La geometria di una shape è una mesh poligonale a piacere. Oltre a questo possono essere inclusi anche colore, matariale e texture utilizzate.
- *Scope* Lo scope è il bounding box che circoscrive la geometria. È rappresentato, nello spazio relativo, dal *Pivot* ed è definito da tre vettori: il vettore transizione, il vettore rotazione e il vettore scala.
- *Pivot* Questo attributo definisce il sistema di coordinate in cui esiste la shape. È definito da un vettore transizione e da un vettore orientamento.

Ogni shape, come già visto, ha un attributo Start Rule, che la associa ad un *Simbolo* della grammatica che si andrà a definire. Nel file delle regole, per ottenere un risultato, dovrà quindi essere presente una regola che richiami quel Simbolo ad un'azione tra quelle che la grammatica stessa mette a disposizione.

Una regola, in generale, è definita da un Simbolo predecessore, un operatore fisso `-->`, e una serie di funzioni e Simboli (i quali prendono il nome di successori), che a loro volta possono avere delle regole associate. Va precisato che, a differenza dei L-System da cui derivano, le regole della grammatica CGA non prevedono la ricorsione ma si basano sull'iterazione. Le regole possono essere di tre tipi:

- **Standard**

  ```
  PredecessorShape --> [operator] Successor
  ```

  Dove `PredecessorShape` e `Successor` sono entrambi simboli della grammatica.

- **Condizionale**

  ```
  PredecessorShape --> case condition1: Successor1

                       case condition2: Successor2
                       ...
                       else: SuccessorN
  ```

Dove, come accade per il case dei linguaggi di programmazione C-based, alla shape viene assegnato il valore che corrisponde al case di cui i requisiti sono soddisfatti.

- **Stocastica**

```
PredecessorShape --> percentage%: Successor1

                     percentage%: Successor2
                     ...
                     else: SuccessorN
```

Similmente alle regole condizionali, questo tipo di shape viene valutata al momento dell'esecuzione e viene scelto uno dei casi in base ad un'estrazione random generata attraverso un generatore di numeri casuali il quale utilizza come seed un valore differente per ogni shape.

Si fornirà ora una breve descrizione dei vari comandi della grammatica CGA shape utilizzati durante il progetto:

const

```
const id = expression
```

Definisce un valore invariabile che viene determinato prima dell'esecuzione della regola iniziale sulla shape corrente. Questo significa che una costante è valida ed immutabile solo durante l'esecuzione su una singola shape, ma può variare per le altre presenti nella scena. La grammatica, non prevedendo una tipizzazione forte, si occupa di riconoscere il tipo di dato della costante. La costante non è accessibile in nessun modo dall'Inspector o da una mappa.

attr

```
attr id = expression
```

Permette di mappare un attributo o di crearlo ex-novo. Ogni accesso agli attributi all'interno delle regole sottende una dichiarazione di questo genere. Anch'esso contiene un tipo di dato riconosciuto in fase di run-time e ne viene definito il valore prima dell'inizio dell'esecuzione delle regole. La differenza con le costanti è, appunto, l'interazione diretta con gli attributi della shape su cui si sta operando.

Nella definizione di attributi e costanti va segnalata una peculiarità, ovvero possono essere definite delle percentuali di "probabilita" ai diversi valori possibili in questo modo:

```
variabile = %10 : valore1
   %10 : valore2
   %30 : valore3
   else : valore4
```

Quando viene utilizzata la variabile (o quando viene dichiarata, nel caso degli attributi e delle costanti), il software si occuperà di selezionare uno dei valori a seconda del risultato di un generatore di numeri casuali, il cui seme cambia di shape in shape.

s, t, r

```
s(float xSize, float ySize, float zSize)
t(float xTranslate, float yTranslate, float zTranslate)
r(float xRotation, float yRotation, float zRotation)
```

Le operazioni *s, t* ed *r*, per chi ha familiarità con il funzionamento di OpenGL, si comportano nello stesso modo delle funzioni scale, translate e rotate. I vettori corrispondenti dello scope vengono, infatti, scalati, traslati e ruotati dei tre valori passati come parametri.

color

```
color(value)
```

Assegna alla geometria corrente il colore definito da ***value*** sotto forma di stringa RGB esadecimale (e.g. `''#ff0000''`)

extrude

```
extrude(height)
```

```
extrude(axisWorld, height)
```

Questa operazione si occupa di creare una geometria solida a partire da quella piana della shape su cui si sta applicando la regola. L'estrusione richiede due parametri: un vettore direzione lungo il quale estrudere e un valore numerico che indica la distanza dalla superficie di partenza. Si può omettere il vettore direzione. In questo caso, City Engine gli assegnerà la normale della shape come default.

comp

```
comp(compSelector) { selector operator operations |
                     selector operator operations ... }
```

La funzione component si distingue per la sua duttilità d'uso e ad una più attenta analisi, risulta essere una delle funzioni probabilmente più utilizzate durante il progetto assieme a `split`. Il suo ruolo è quello di selezionare e suddividere varie parti della geometria presa in esame ed assegnare le varie parti ad altrettante regole o funzioni. Generalmente viene usata per selezionare differenti facce di una mesh appena generata attraverso extrude. Prevede un parametro letterale che indichi quali parti della geometria si vuole prendere in considerazione. I valori ammessi sono: `f` (faces), `e` (edges), `v` (vertices). Successivamente si deve indicare, a seconda del tipo di parametro, almeno un selettore, un operatore e un'operazione. I selettori usati sono:

- `front, back, left, right, top, bottom` selezionano, in base al sistema di coordinate locali, le componenti corrispondenti.
- `side` seleziona tutti i componenti tranne quelli orizzontali.
- `all` seleziona tutte le componenti che formano la geometria.

Gli operatori sono due:

- `:` Ogni componente selezionato definisce una nuova shape.
- `=` Tutti i componenti selezionati vengono fusi in un'unica nuova shape.

Le operazioni non sono altro che una serie di istruzioni CGA da eseguire.

split

```
split(splitAxis) { size1 : operations1 |
                   size2 : operations2 |
                   ... |
                   sizen-1 : operationsn-1 }
```

L'altra operazione di fondamentale importanza all'interno della grammatica CGA shape è, come già sottolineato, *split*. La funzione effettua, lungo l'asse definita dal parametro `splitAxis`, una suddivisione in varie shape delle dimensioni indicate dai valori *`sizeN`* (Fig. 2.10). Questi valori possono essere preceduti da due prefissi:

- `'` *(relativo)* Questo prefisso indica che la nuova dimensione della shape sarà determinata in modo relativo dall'operazione `size` * dimensione dello scope corrente.
- (floating) tutto lo spazio rimanente dalle defnizioni assolute di dimensione viene distribuito tra i valori *floating* che diventano, nella pratica, dei valori relativi per determinare la dimensione finale.

i

`i(geometryPath)`

Questa operazione serve, in modo semplice, ad inserire all'interno dello scope della shape corrente un oggetto di tipo mesh. L'asset inserito viene modificato in modo da far coincidere il suo bounding box con lo scope.

roofHip, roofGable

```
roofHip (height, offset)
roofGable (height, offset)
```

Le operazioni `roofHip` e `roofGable` creano entrambe un tetto in corrispondenza della shape corrente. Il parametro `height` definisce la pendenza delle falde dei tetti espressa in gradi, il secondo parametro `offset` indica di quanto le falde del tetto dovranno sporgere dai bordi della shape. roofHip e roofGable sono le operazioni per creare, rispettivamente, tetti a due falde e tetti a quattro falde.

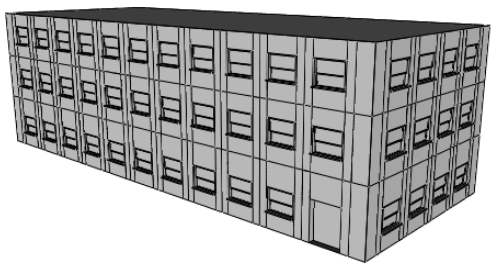

Figura 2.10: Risultato di multipli split partendo da un'unica geometria

Capitolo 3

Ricostruzioni 3D filologicamente accurate

Il progetto Apa, Museo della Città ha visto la ricostruzione di diversi scenari in epoche differenti di uno stesso ambiente urbano, Bologna, dalla sua nascita fino ai giorni nostri; questo ha dato modo di individuare alcune fasi comuni durante la produzione, nonostante la diversitá intrinseca tra i vari set, sia dal punto di vista dei dati di partenza che dell'obiettivo da raggiungere.

Ricordiamo infatti che L'ICT applicato ai beni culturali favorisce un'esperienza culturale che pone l'utente al centro del processo. Obiettivo della produzione del Cineca è stato utilizzare l'impatto emotivo di un prodotto di edutainment per favorire il trapasso di contenuti culturali.

Si é reso, pertanto, necessario individuare quali e quanti set si sarebbero dovuti elaborare, con che grado di dettaglio insieme alle modalitá di realizzazione degli stessi.

Il cortometraggio comprende scene che coinvolgono per intero la città e sue ricostruzioni nel tempo per un totale di cinque epoche principali che mostrano Bologna proprio come gli storici suppongono che fosse. Come ho già accennato nell'Introduzione, io mi sono occupato del set della Bologna contemporanea in low-poly (ovvero a livello di dettaglio basso) e di quello della Bologna in epoca medievale, sia low-poly che hi-poly (a livello più alto di dettaglio).

Per dovere di completezza e per poter meglio valutare il contributo significativo di City Engine nella pipeline integrata introdotta dal cineca, diamo di seguito una breve descrizione delle epoche in cui Bologna è stata ricostruita, così come compaiono nel film, sottolineando le difficoltà che hanno condotto all'impiego o meno di City Engine. Ci soffermeremo maggiormente sulle due location in oggetto a questa tesi: la Bologna attuale e quella medioevale, evidenziando

il ruolo di tale software nel processo produttivo.

Bologna attuale

Figura 3.1: Mock-up di Enrico Valenza della Bologna attuale

Lo scenario che presenta Bologna nel suo aspetto contemporaneo, il primo che compare nella pellicola, è stato anche il primo ad essere realizzato per via della realizzazione dei mockup, fig. 3.1.

I modelli del centro storico attorno alle due torri e alla basilica di San Petronio erano già in possesso del Cineca per via dell'antecedente progetto NuME[1]. Attorno ad essi dovevano essere integrati gli altri edifici che compongono la città.

Le fonti per la realizzazione di tale modello 3D sono state fornite dal Comune di Bologna in termini di dati georeferenziati, insieme alle planimetrie degli edifici del comprensorio bolognese e ai dati della vegetazione (interamente censita dal comune). Ogni singolo arbusto censito è stato riprodotto nel set. Per fornire un'idea della mole di dati in gioco, si consideri il fatto che il solo centro storico, per esempio, conta più di cinquemila piante.

Se i dati forniti consentivano di definire la dimensione precisa dei perimetri degli edifici e delle zone verdi, non comprendevano informazioni sulla geometria da riprodurre nel modello. In tal senso, è venuto in aiuto City Engine che ha consentito di ricreare per intero la città definendo regole di modellazione procedurale, ovvero in maniera algoritmica automatica o semi-automatica. Questo ha rappresentato il giusto compromesso tra aderenza con la realtà (e filologicità del modello) e complessità del processo di produzione sia in termini di tempi uomo che di agilità della geometria.

[1] `www.storiaeinformatica.it/nume/italiano/ntitolo.html`

Bologna etrusca Lo scenario etrusco, insieme a quello rinascimentale che vedremo dopo, non ha richiesto l'intervento della modellazione procedurale. stato infatti possibile avvalersi dei contenuti open rilasciati da uno degli open project di Blender, i.e. Big Buck Bunny (`www.bigbuckbunny.org`) che hanno consentito di ricreare la vegetazione e l'habitat per lo scenario della necropoli, composto dalle stele, ricostruzione filologica degli originali conservati presso il Museo Civico e Archeologico di Bologna. Anche i pochi edifici costruiti per il centro abitato sono stati frutto di modellazione 3D tradizionale in Blender.

Bologna romana La produzione di questo scenario si è avvalsa della collaborazione con il CNR Itabc di Roma, che ha ricostruito l'intera città a partire dalle fonti georeferenziate per i terreni, la centuriazioni, i templi e le singole domus che compongono il set.

Bologna medievale La produzione di questo scenario è quella che ha maggiormente usufruito delle potenzialità offerte dalla modellazione procedurale. Pur potendosi avvalere, infatti, delle due torri e degli edifici limitrofi realizzati per il già citato progetto NuME e migliorati per soddisfare le esigenze cinematografiche del progetto, questo set è dovuto nascere letteralmente dal nulla. La fonte che ha rappresentato la base di partenza è stato il plastico filologico conservato al Museo civico Medioevale che mostra nel dettaglio la geometria della città in epoca medievale. Il lavoro di ricostruzione è, pertanto, partito dai fotopiani del modello che sono stati importati in City Engine ed elaborati fino ad ottenere lo scenario virtuale della Bologna turrita che è mostrato nel cortometraggio.

Bologna rinascimentale Questo scenario lo si è potuto ricostruire per intero assemblando i vari modelli prodotti per il progetto NuME, congiuntamente a una ricostruzione basata sulla mappa vaticana di Bologna conservata in Città del Vaticano a Roma, nella stanza denominata 'la Bologna' nel palazzo di Sisto V. Grazie alla sua accuratezza di rappresentazione sia urbanistica che iconografica, questa mappa può essere considerata una sorta di Google Earth dell'epoca e ha consentito di condurre il racconto ad espolorare la Bologna città della seta e di entrare nel modello di un filatoio, frutto di un lavoro di dottorato.

La modellazione procedurale, come strumento di Virtual Archaeology, ha rappresentato anche un'occasione di studio relativamente ai metodi di ricostruzione rivolti al riutilizzo. Essa è stata, tuttavia, soggetta, come il resto della

pipeline, ai vincoli introdotti nel processo di produzione dall'intento filologico. Conformente anche alle linee guida indicate dal London Charter[2], il processo di pianificazione per la produzione procedurale ha compreso i passaggi descritti di seguito.

3.1 Fonti

L'acquisizione delle fonti e la loro validazione ha comportato la parte probabilmente piú lunga e rilevante dell'intero iter di ricostruzione degli scenari. Ogni epoca storica, infatti ha dovuto fare affidamento su tipi di dati differenti, e non sempre questi sono risultati sufficienti a fornire le risposte cercate.

La consultazione di diversi specialisti ed esperti, d'altronde, ha posto un problema di conflitto tra quelle che potevano essere scuole di pensiero differenti, rischiando di allungare tempi giá per loro natura, stretti.

A questo si aggiunga che, in ultima istanza, il filmato doveva risultare sia corretto dal punto di vista delle informazioni trasmesse, sia accattivante e capace di esprimere quell'aspetto emozionale verso il quale il cultural heritage sta virando per migliorare l'esperienza da parte dei suoi fruitori. Quindi tutto quanto è stato vagliato assieme al regista, il quale ha trovato il punto di incontro tra le informazioni e l'evolversi della storyline definita nel breakdown.

Sebben non sempre le fonti siano state di difficile reperimento o in forma tale da richiedere lavoro aggiuntivo per renderle utilizzabili, come si potrá vedere ad esempio per il caso della Bologna Contemporanea, il percorso di verifica a stretto contatto con storici e archeologi deve rappresentare una costante per progetti di virtual heritage in quanto i modelli devono attraversare numerose fasi di validazione che li accompagnano dalla progettazione allo sviluppo.

3.2 Analisi e progettazione del modello

Un modello realizzato come modello filologico ma da utilizzare come set di un cortometraggio deve sottostare innanzi tutto alle fonti ma anche a una serie di canoni estetici. In fase di progettazione si devono, cioè, individuare gli elementi caratteristici che contraddistinguono uno specifico scenario: il genere di architettura, i colori e i materiali piú diffusi sono, da questo punto di vista, gli aspetti che maggiormente permettono di riconoscere un'epoca storica al primo sguardo.

[2] `http://www.londoncharter.org/`

Le forme da distinguere sono in primo luogo quelle relative alle dimensioni dei basamenti degli edifici, in quanto questo dato serve poi per definire la posizione dei volumi sulla scena. Successivamente le domande da porsi e a cui trovare risposta sono, innanzi tutto: quanti piani conta mediamente un'abitazione? Di che tipo sono gli infissi che é possibile vedere? C'e presenza di aree verdi, elementi architettonici di rilievo quali portici, colonnati, terrazzi, tetti dall'aspetto particolare o non a spiovente? Ci sono nell'ambiente degli edifici storici che é possibile sostituire con modelli giá creati in precedenza?

Tutte le nozioni raccolte in questa fase, nel caso di utilizzo della modellazione procedurale, vanno poi tradotte sotto forma di regole CGA, ovvero regole di una grammatica che consente di controllare massa, geometria, proporzioni o texturing di edifici e strade.

Tali regole, una volta applicate alle shape che delimitano le fondamenta degli edifici, provvederanno a creare la geometria. Se la fase di analisi delle fonti é stata fatta correttamente, il modello ottenuto non sará troppo dissimile dalla realtá (Fig. 3.2).

Figura 3.2: Mock-up di Enrico Valenza della Bologna contemporanea

3.2.1 Assets

Si definisce asset qualsiasi elemnto 3D che faccia parte di una scena. La maggior parte degli asset utilizzati durante il progetto MDC provengono da una modellazione 3D tradizionale, eseguita con Blender ed in seguito importata all'interno delle scene, a questi, ovviamente si aggiungono gli asset creati con City Engine che, logicamente, hanno dimensioni di gran lunga maggiori a quelle delle comuni geometrie realizzate.

La suddivisione in assets delle varie scene risulta, comodo dal punto di vista della distribuzione del lavoro: ognuno può crearne in modo autonomo senza intralciare il lavoro di altri membri del team e in questo modo si velocizzano i tempi rimandando quello che diventa un vero e proprio "assemblaggio" della scenografia, al termine della realizzazione di tutti gli elementi che la comopongono.

Ogni scena richiede un numero variabile di assets, che può aumentare o diminuire a seconda dalla complessità della stessa, tuttavia uno dei vantaggi principali della progettazione 3D in generale è la possibilità di riutilizzo dei modelli, per cui una volta creato un asset, esso può essere inserito in più scene per mantenere, oltretutto, una coerenza visiva.

Nel caso del progetto Apa, avendo a che fare con scene praticamente dissimili tra loro in ogni dettaglio, per via della differente collocazione temporale, gli unici assets che è stato possibile riutilizzare per gran parte delle scene sono stati, per primo, ovviamente il modello del protagonista Apa, oltre a questo anche la geometria rappresentante il territorio, gli skydome[3] e l'intera geometria degli alberi.

Lo studio degli assets necessari in ogni scena dipende dal grado di libertà concesso dalle fonti. Meno fonti attendibili sono disponibili, più il lavoro tende a permettere alla creatività di entrare in gioco, d'altro canto, se le fonti sono consistenti e ben definite, l'unico lavoro da svolgere è quello di ricostruire tridimensionalmente un oggetto, edificio o personaggio seguendo la documentazione fornita.

3.2.2 Palette dei colori

Per ottenere una ricostruzione attendibile di un ambiente urbano, oltre allo studio sugli assets necessari, è fondamentale riconoscerne la palette di colori che lo caratterizza. Questo tipo di analisi può essere condotta in prima persona con sopralluoghi di un'area-campione o, alternativamente, attraverso fonti fotografiche, per non parlare delle documentazioni scritte e illustrate. Nel caso di ambienti urbani di epoche passate, escludendo a priori l'analisi diretta, le uniche informazioni reperibili sono di tipo documentario e illustrato, mentre le fonti di tipo fotografico sono presenti solo da un certo periodo storico in poi. In questi casi di scarsa attendibilità delle fonti o di mancanza totale di esse, il

[3]Sfere poligonali con le normali rivolte verso il centro, alle quali viene applicata una texture che simuli il cielo

metodo migliore, come è accaduto per Bologna Medievale, è quello di affidarsi alla palette offerta dagli edifici storici.

3.3 Scripting procedurale

La regola CGA serve a creare un modello che rispecchi quanto piú fedelmente l'aspetto originale di un ambiente urbano, e non a replicarlo in ogni suo dettaglio. La visione di insieme non ne risente, ad esempio, se il colore di due edifici adiacenti risulta invertito nel modello rispetto la realtà.

Per semplificare il piú possibile, generalmente, le regole scritte seguono il modello di un edificio e ne applicano alcune varianti per creare discontinuitá all'interno della scena.

La struttura della grammatica, inoltre, permette di inserire queste variazioni in modo semplice, come si vedrá nel caso di Bologna Medievale.

3.4 Esportazione e modifiche puntuali

Una volta applicate le regole, la geometria generata va selezionata ed esportata in formato Wavefont OBJ avendo cura di diminuire tutti i parametri di ottimizzazione a 0.00001 e spuntare la casella per triangolare le mesh, presenti nella maschera di esportazione di CityEngine; questi ultimi parametri, sono necessari per evitare di perdere delle informazioni sulla geometria nel processo di interscambio. Il file obj cosí formato, viene quindi importato all'interno di una scena vuota di Blender, che diventerá di fatto la base di dati da cui attingere per creare le scenografie per le scene e gli shot necessari.

La possibilità di esportare nel formato obj è stato il motivo che ha consentito a City Engine, software proprietario, di entrare a far parte della pipeline Open di produzione del progetto Apa. Risultati in formato obj sono potuti essere inseriti in ulteriori passi di lavorazione in Blender per assemblare in un unico set modelli pre-esistenti con il terriotrio urbano mancante o per definire il necessario livello di dettaglio difficilmente ottenibile con il solo procedurale.

3.5 Creazione delle scene

Per ogni scena, sul server del progetto, è stato creato un file Blender nel quale gli unici asset fisicamente salvati dentro di essi erano gli oggetti camera e i personaggi animati. Tutte le parti di scenografia necessaria si trovavano, invece,

negli stessi file dove erano stati modellati e salvati (o importati, nel caso dei modelli procedurali). Nel file della scena, per l'appunto, è stato creato un link per ogni file di assets, ed il rendering di ogni scena proviene da uno di questi file (Fig. 3.3)

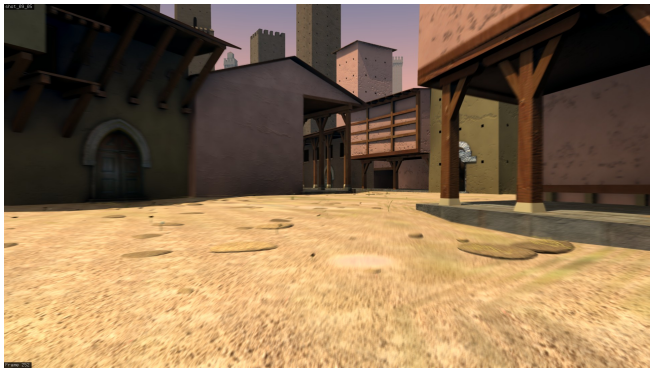

Figura 3.3: Un frame della sequenza di corsa sotto i portici medievali

Capitolo 4

City Engine per il progetto Apa

I casi studiati durante il periodo di tirocinio e tesi, ovvero gli scenari di Bologna contemporanea e Bologna in epoca medievale, hanno richiesto un processo di creazione particolare, rispetto a quanto descritto in termini generali nel precedente capitolo. Di seguito, porcediamo a scendere nel dettaglio.

4.1 Bologna contemporanea

Il lavoro che é stato necessario svolgere per la realizzazione di Bologna contemporanea, o Bologna attuale, si è avvalso di fonti accurate ed esatte come non é stato possibile fare per nessun'altra scena del progetto Apa. La sua realizzazione era, peraltro, necessaria in un tempo minore rispetto a tutti gli altri set, in quanto vincolante per la creazione dei mockup che sono stati soggetti alla validazione che ha dato l'ok per l'inizio dei lavori (Fig. 4.1).

Figura 4.1: La presentazione dei mockup per la loro validazione

Al contrario di quanto generalmente avviene nella pipeline tradizionale di produzione di un cortometraggio, infatti, dove prima si produce quella che viene definita *concept art*, ovvero forma di illustrazione che ha lo scopo di rappresentare il mood del film, il progetto Apa ha richiesto in prima istanza la terminazione

dei set, perlomeno nelle geometrie base (low-poly) al fine di consentire all'Art Director (l'artista Enrico Valenza) di creare i mockup che sono stati validati dalla commissione scientifica del progetto. Sono stati prodotti sette color layout che hanno consentito di dare al progetto l'ok per la produzione vera e propria. Si Ã¨ trattato di scenari urbani di Bologna in varie epoche: attuale, etrusca, romana, medievale e rinascimentale.

4.1.1 Fonti ed elaborazione

Le fonti utilizzate come base per la creazione della scena con CityEngine provengono da informazioni del catasto comunale di Bologna, il quale ha fornito i dati georeferenziati di ogni edificio presente sul suolo comunale. La maggior parte delle informazioni legate ai singoli perimetri, a onor del vero, non sono state utilizzate in quanto inutili per i nostri scopi. Gli attributi che ci interessavano, infatti, si riguardavano unicamente l'altezza di gronda e la tipologia di costruzione. Quest'ultima può essere catalogata come `Chiesa`, `Campanile`, `Torre` o `Edificio generico` (Fig. 4.2).

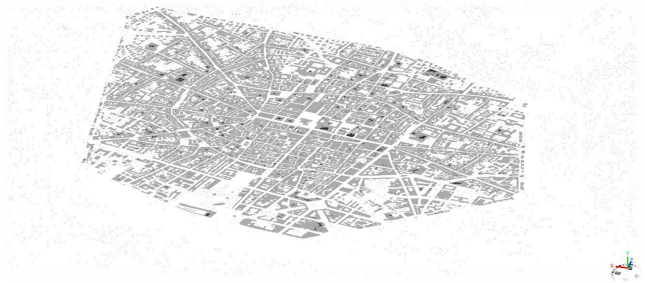

Figura 4.2: Una visualizzazione dei dati del catasto del centro cittadino in City Engine

Non è stata necessaria ulteriore elaborazione di questi dat in modo da renderli leggibili o compatibili per il rpogramma in quanto CityEngine gestisce i dati GIS.

Con l'integrazione della heightmap del terreno, ovvero una texture georeferenziata rappresentante in greyscale la mappa delle altitudini nel territorio, inoltre, si è potuto creare il panorama della città e dei colli circostanti in modo accurato, utilizzandola come base per creare il terreno.

Un problema riscontrato nelle fonti del catasto è stata la presenza dei perimetri interni agli edifici, ovvero la rappresentazione dei cortili interni dei palazzi, i quali erano identificati dallo stesso ID del poligono esterno, con la normale della faccia rivolta verso il basso. Questo problema è stato ovviato non rappresentando tale informazione in quanto la visuale della città non richiedeva un dettaglio tale da richiederne la realizzazione; si vedrà più approfonditamente come questo problema sia stato risolto direttamente all'interno delle regole di generazione.

A completamento delle fonti riguardanti l'aspetto contemporaneo della città di Bologna vi sono le informazioni relative alla flora anche'essa censita dal catasto. Soltanto all'interno dell'anello formato dai Viali, si contano più di 5000 unità.

Per ricavare la palette dei colori relativa agli edifici contrassegnati dalla tipologia `Chiesa` sono state utilizzate le texture e i colori presenti nel progetto accademico NuME [**?**]. I modelli provenienti da tale progetto sono inoltre stati utilizzati come base di partenza per la realizzazione dei modelli hi-poly.

Alberi

L'inserimento degli alberi, effettuato come prima prova di utilizzo delle grammatiche CGA durante la mia esperienza al Cineca, si è distinto in due passaggi fondamentali, il primo di apprendimento, per capire che potenzialità City Engine riusciva ad offrire, il secondo, invece ha visto l'adattamento delle esigenze di produzione, le quali richiedevano una dimensione file il più possibile limitata, con le richieste della ricostruzione verosimile di un paesaggio.

Va detto che, per funzionare in modo corretto, al momento dell'importazione della "nuvola" di punti rappresentanti la posizione della flora, City Engine ha automaticamente convertito all'interno della scena, ogni punto in una piccola mesh quadrata. Questo comportamento è dovuto al fatto che le grammatiche CGA necessitano di una shape almeno bidimensionale per poter lavorare correttamente.

4.1.2 Generazione delle texture

Ci si è presto resi conto che i dati georeferenziati non fornivano nessuna informazione circa i colori e i materiali che caratterizzano Bologna. Si è quindi effettuata una breve ricerca di risorse fotografiche della città che ne evidenziassero le tonalità.

Tenuto conto della dimensione considerevole dell'ambiente ricostruito, e non volendo appesantire lo scenario con geometrie complesse, si è pensato di realiz-

zare alcune texture ad hoc rappresentanti un campione fittizio di alcune facciate dei palazzi cittadini, da poter mappare adattivamente sui singoli modelli, seguendo un esempio di ricostruzione della città di Venezia fornito come guida all'apprendimento del software e della grammatica CGA [Pro08].

Questa scelta è stata dettata in buona parte, come già accennato, dalla necessità di non appesantire la geometria generata con CityEngine - già la creaizone di un solo parallelepipedo con un tetto spiovente per ogni edificio metteva in seria difficoltà i calcolatori più performanti a nostra disposizione, congiuntamente al fatto che mancavano quasi tutte le texture necessarie per coprire il modello. Per ovviare al problema dell'uniformità dello stile all'interno dello scenario, si è quindi deciso di creare le immagini attraverso alcuni passaggi effettuati con CityEngine e Blender in stretta sinergia.

Ipotizzando un limite massimo di 5 piani in altezza e di 6 "finestre" in larghezza per ogni edificio, è stato creato uno scenario con CE, da generare in hi-poly (importando alcune istanze di infissi di tipo OBJ modellate apposta con Blender). Esso rappresentava una matrice di edifici di ogni dimensione da 1x1 fino 6x5, conformi alle informazioni acquisite dalla ricerca sull'aspetto di Bologna, quindi sui colori delle facciate. SI è poi esportata la geometria generata come file OBJ e una volta importata all'interno di Blender, si è effettuato un rendering di grandi dimensioni con vista frontale e ortogonale rispetto agli edifici (Fig. 4.3).

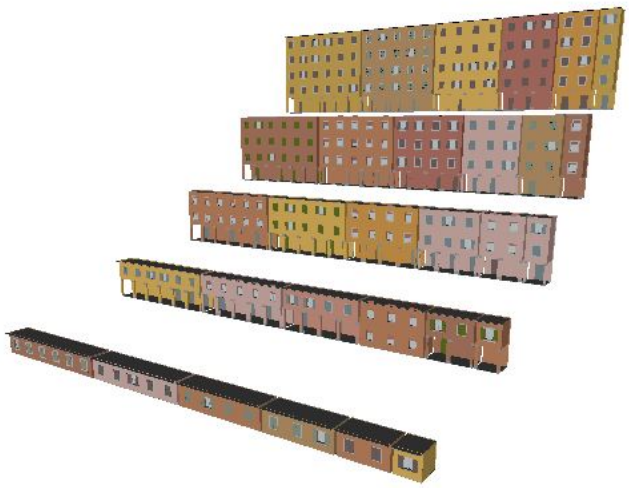

Figura 4.3: Lo scenario generato per la creazione delle texture ad hoc

L'immagine risultante, ovvero una matrice di facciate, è stata ritagliata per ricavarne le texture tramite un semplice file in linguaggio python (vedi Appendice A.2).

4.1.3 Scripting procedurale

Volendo considerare le differenze legate strettamente al tipo di risultato ottenuto, le rules create per generare la geometria sono tre, una per la realizzazione delle geometrie rappresentanti gli alberi, una ausiliaria utile per la creazione delle texture delle facciate degli edifici e una, suddivisa in due file, per la generazione delle geometrie degli edifici in modo distinto tra quelli all'interno dell'anello delimitato dai Viali, e quelli presenti sul resto del territorio comunale, in quanto richiedenti un dettaglio inferiore.

Alberi

Questa regola consiste nell'inserimento di una mesh semplice, modellata ad hoc con Blender ed esportata nel formato di interscambio wavefont OBJ, per ogni poligono rappresentante una pianta all'interno dei dati. Il risultato ottenuto, essendo stato il primo approccio con la modellazione procedurale, ha piacevolmente sorpreso sia per la semplicità di realizzazione che per la qualità raggiunta. Il comando che, nella pratica, esegue tutto il lavoro, una volta istanziato per ogni shape è questo:

```
Lot -->s(scala,scala,scala)
t(-7,0,-7)
i(alberello)
color(colore)
```

dove `scala, alberello` e `colore` sono dichiarati in precedenza (Fig. 4.4).

Successivamente, per limitare il numero di punti descritti all'interno dei file OBJ (e di conseguenza, la dimensione complessiva dell'asset) si è ripreso lo script di inserimento degli alberi e lo si è modificato in modo da creare un piano verticale a cui assegnare la texture di una pianta. Questa tecnica di rappresentazione che semplifica la complessità geometrica di un oggetto per demandarne la rappresentazione ad immagini applicate ad uno o più piani nella scena prende il nome di *billboard*. Il codice, questa volta, non fa uso dell'inserimento di geometria esterna, ma sfrutta i piani che è possibile generare con City Engine per creare i billboard e si divide in due parti. La prima regola crea la geometria necessaria

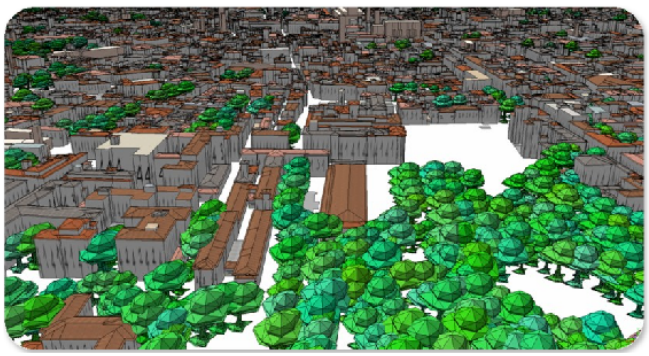

Figura 4.4: Il primo test sull'inserimento degli alberi nella scena

```
Lot --> extrude(world.y,1) Volume

Volume --> comp(f) {front: Albero | all: NIL}
```

La seconda, invece, applica l'immagine

```
Albero --> s(sz,sz,scope.sz)
   setupUV(0,scope.sx,scope.sy)
   set(material.colormap,alberino)
   bakeUV(0)
```

dove `sz` e `alberino` sono dichiarati in precedenza.

Facciate

Questo insieme di regole scritto per creare un gruppo di edifici generici a dettaglio alto, in seguito utilizzati per creare le texture necessarie al dressing del modello procedurale, ovvero il completamento dei set con gli oggetti di scena, consiste in una estrusione iniziale di ogni shape e in una serie di suddivisioni a cascata che ne determinano l'aspetto.

Lavorando sulla facciata frontale di ogni estrusione,

```
Piano --> comp(f) { front:Facciata |
side:Wall |
top: NIL |
bottom:Wall}
```

in seguito allo studio delle fonti, si sono poi suddivise le aree in modo sempre più raffinato, fino ad inserire nei punti desiderati la variante di una mesh poligonale rappresentante finestre o porte:

```
AreaFinestraSenzaAnte --> split(x) { ~1:Wall |
CORNICE: Cornice |
2*WIN_W:AreaFinestraSenzaAnte2 |
CORNICE: Cornice |
~1:Wall}
AreaFinestraSenzaAnte2  --> split(y) { 0.05:Soglia |
~1:AreaFinestraSenzaAnte3}
AreaFinestraSenzaAnte3 -->  t(0,0,-0.2)
extrude(world.z, 0.2)
comp(f) { front:Gray |
side:Wall |
top:NIL |
bottom:Finestra Tenda }

AreaFinestraAperta    --> split(x) { ~1:Wall |
WIN_W:Persiana |
2*WIN_W:AreaFinestra |
WIN_W:Persiana |
~1:Wall }
AreaFinestraChiusa    --> split(x) { ~1:Wall |
CORNICE: Cornice |
WIN_W:Persiana |
WIN_W:Persiana |
CORNICE: Cornice |
~1:Wall}

AreaFinestra  --> split(y) { 0.05:Soglia |
~1:AreaFinestra2}
AreaFinestra2 -->  t(0,0,-0.2)
‘extrude(world.z, 0.2)
comp(f) { front:Gray |
side:Wall |
top:NIL |
bottom:Finestra }
```

Edifici

La generazione degli edifici contemporanei ha preso spunto dal sistema proposto nei tutorial forniti da Procedural per la ricostruzione dell'aspetto di Venezia, ovvero l'assegnazione adattiva delle texture alle facciate degli edifici, in base alla loro dimensione.

Per prima cosa si sono mappati gli attributi della shape, poi si sono definite variabili e costanti ausiliarie utili per la generazione; oltre a questo, si è definita una funzione di controllo per verificare la direzione della normale per ogni shape, in modo da poterla riconoscere e, di conseguenza, ignorare nel caso fosse rivolta verso il basso.

```
const ny = cos(initialShape.origin.ox)
           * cos(initialShape.origin.oz)
           + sin(initialShape.origin.ox)
           * sin(initialShape.origin.oy)
           * sin(initialShape.origin.oz)
```

Una volta definite queste impostazioni, le regole di generazione consistono in una identificazione iniziale del tipo di shape, per evitare di creare geometrie inconsistenti su superfici troppo piccole o non desiderate

```
Lot  -->case (ALT_UV) <= 1:
NIL
 case HIDE:
  NIL
 case ny<-0.5:
  NIL
 case geometry.area < 20:
  NIL
         else:
extrude(world.y,ALT_UV)
Extr
```

per poi effettuare a cascata, con regole successive, tutte le operazioni necessarie per la generazione della geometria, l'inserimento degli asset delle torri nel centro cittadino e l'assegnazione automatica della texture più idonea a ricoprire le facciate dell'edificio, attraverso una semplice funzione:

```
getScopeY = case(nTextureFloors*floorHeight < scope.sy*0.85) :
```

```
scope.sy/2
else :
scope.sy


Side --> case scope.sx>maxTextureTiles*tileW*0.7:
split(x){~1.5: Side|~1: Side}
 else:
  alignScopeToAxes(y)
  setupUV(0,scope.sx,getScopeY)
bakeUV(0)
set (material.colormap,
getBuildingTexture(nTextureFloors,nTextureTiles))
```

Per quanto riguarda le regole di generazione degli edifici al di fuori del centro cittadino, invece, il procedimento è pressoché invariato, tranne per la semplificazione delle geometrie delle torri, le quali non sono più inserite come asset ma generate come parallelepipedi, considerata la distanza dalla telecamera nelle scene.

Il Concept Layout, realizzato dal direttore artistico, alla fine è risultato essere di forte impatto visivo, con grande soddisfazione da parte di tutto il team di lavoro (Fig. 4.5)

Figura 4.5: Una visuale del modello di Bologna attuale

4.2 Bologna medievale

La parte più rilevante del lavoro svolto come tesi è quella che mi ha visto impegnato nella realizzazione dei set relativi agli shot 07 e 09 del cortometraggio, ovvero Piazza Santo Stefano e la corsa sotto i portici che culmina con la visuale panoramica sullo skyline della città .

Gran parte di questo lavoro è stato realizzato con Blender, in quanto le inquadrature proposte imponevano un livello di dettaglio ben più alto rispetto a quello raggiungibile con CityEngine. Nonostante questo, la modellazione procedurale ha permesso una ricostruzione dell'intera città di Bologna entro le mura che fosse filologica ed accurata rispetto alle fonti, creando, di fatto, una delle scene più suggestive dell'intero filmato nella visione della cittá turrita ai piedi della torre degli Asinelli.

4.2.1 Acquisizione delle fonti

A differenza degli altri scenari realizzati per il progetto, quello di Bologna medievale non aveva nessun tipo di informazione georeferenziata da poter sfruttare come base per poterne ricostruire l'aspetto. La collaborazione con gli storici e gli urbanisti dell'università ha consentito di individuare, come fonte attendibile di informazione, un plastico dell'architetto Angelo Finelli intitolato Bologna al tempo di Dante, attualmente conservato al museo Civico Medievale. A partire da tale plastico sono stati realizzati dei fotopiani ad alta risoluzione, successivamente composti in un'unica immagine che ha fornito la base iniziale necessaria per la creazione dello streetgraph all'interno di City Engine (Fig.4.6).

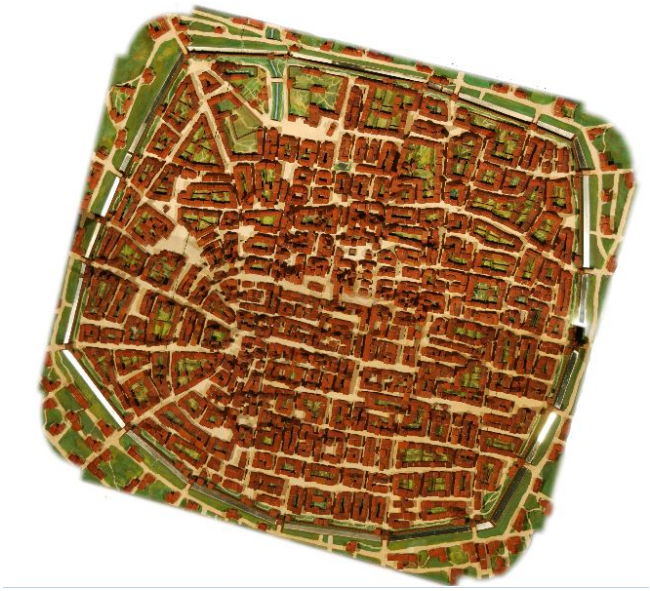

Figura 4.6: Fotopiano del plastico *Bologna al tempo di Dante*

4.2.2 Produzione del modello Low Poly

L'immagine composta dai fotopiani del plastico, inserita all'interno del software Layer Map, ha permesso, attraverso gli strumenti di creazione interattiva delle strade, di ricalcare il reticolato delle vie. Questo passaggio per City Engine costituisce il primo step della pipeline di generazione di una città procedurale.

Dallo streetgraph, attraverso gli strumenti del software, sono state ricavate tutte le aree chiuse delimitate dalle strade. Si è cosi' ottenuto un layer di "macroshape", rappresentante tutte le aree dell'interno delle mura di Bologna occupate da edifici.

Attraverso il tool di suddivisione delle shape, sono state ottenuti dei poligoni, prevalentemente quadrilateri, che hanno rappresentato, seppur in modo in modo approssimativo, i perimetri degli edifici su un layer differente da quello delle aree di partenza.

Ottenuti i perimetri, la preoccupazione maggiore è stata quella di impedire la generazione degli edifici corrispondenti a quelli già presenti all'interno del progetto NuME medievale, che sarebbe stato inserito successivamente nella scena. Oltre a questo, si e' dovuto individuare le posizioni delle oltre novanta torri che caratterizano l'aspetto di Bologna durante il medioevo. Non ultimo, si è dovuta evitare una generazione incoerente dei portici, i quali, come noto, sono la caratteristica distintiva della città.

Queste problematiche sono state risolte in modo manuale e non automatico, in quanto il tempo e le energie necessari al loro svolgimento non è apparso troppo oneroso.

Ad ogni shape presente nella scena è stato quindi assegnato un attributo boolean `HIDE` che indicasse la necessità di generare la geometria corrispondente. Dopodiché, si è assegnato selettivamente il valore `false` ai lotti corrispondenti alle aree che sarebbero state occupate, nella scenografia finale, dai modelli hi-poly.

La definizione delle posizioni delle torri è stata effettuata in modo analogo, assegnando un attributo `Tower` di valore `false` ad ogni shape. Similmente alla creazione dello streetgraph, si è andati a ricalcare quelle che sarebbero diventate le aree in cui ubicare le torri posizionandosi con la telecamera del software sopra alle aree shape sovrapposte alla mappa.

Lo stesso procedimento è stato utilizzato per i portici. Alcune shape contigue sono state contrassegnate con l'attributo boolean `LayoutVisible` di valore `true`.

Le fonti documentaristiche a disposizione per la ricostruzione visiva dei portici attraverso le regole [Boc97] sono state fondamentali per avere un'idea

dell'impostazione del lavoro da svolgere (Fig. 4.7)

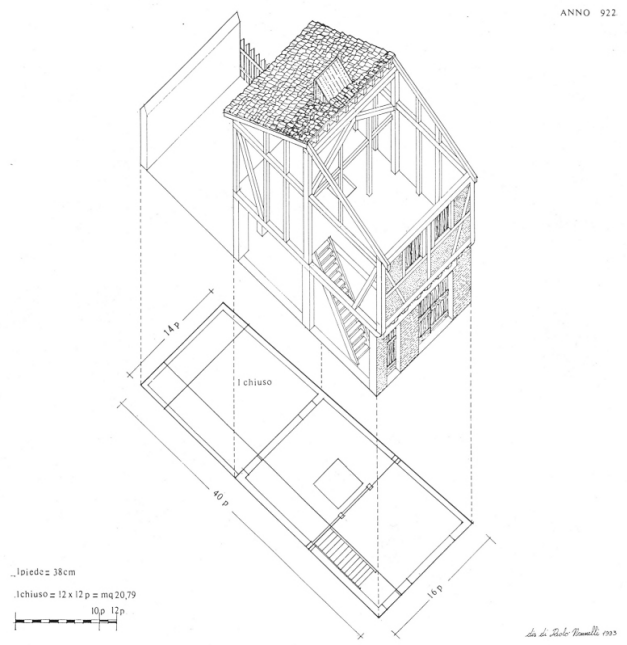

Figura 4.7: Una delle fonti documentarie sulla forma degli edifici

Le regole fornite tramite la grammatica CGA hanno, pertanto, seguito le indicazioni basate sulle fonti. Si è cercato di ricreare in modo accurato l'aspetto degli edifici del centro storico di Bologna in epoca medievale, gestendo al meglio il trade-off sempre presente tra la complessità della geometria generata (che ne definisce, su modelli procedurali estesi, le dimensioni in termini di memoria virtuale) e la qualità visiva.

A tal proposito, si è scelto di utilizzare un approccio misto, tra modellazione procedurale e manuale, inserendo piccole mesh descrittive di alcuni dettagli create con Blender ed esportate come file wavefont OBJ, di difficile realizzazione attraverso le grammatiche CGA.

Come per i piccoli dettagli a cui si è appena fatto riferimento, le torri inserite nello scenario, sono state modellate utilizzando Blender, utilizzando come base le torri Garisenda e Asinelli di Bologna, presenti all'interno di NuME.

Attraverso piccole ma significative modifiche quali il posizionamento della merlatura, l'inserimento di finestre, e l'aggiunta di guglie, si sono ottenute alcune varianti di torre, che, in modo randomico, sono state aggiunte, tramite script, all'interno della geometria.

Non avendo nessuna texture adatta allo scopo, l'ultimo passaggio per l'ultimazione del modello procedurale attraverso City Engine, è stato quello di progettare la geometria necessaria alla rappresentazione del reticolato stradale.

City Engine permette di farlo esattamente come avviene per la creazione delle regole per la modellazione degli edifici, assegnando un simbolo dell'alfabeto della grammatica ed un file di regole da utilizzare ad ogni segmento stradale e ad ogni incrocio tra essi, che vengono considerati come entità distinte.

Il risultato finale di questo lavoro, quindi, è una riproduzione del plastico di Bologna, chiaramente non fedele ed accurata dal punto di vista dei dettagli, ma in grado di rappresentarne l'aspetto al punto da renderla riconoscibile al pubblico che avrebbe visionato il filmato. Questa soluzione, per quanto possa sembrare in contrasto con la filologia di cui il progetto si fregia, non influisce sulla qualità dei contenuti proposti, in quanto la visualizzazione della città nel suo complesso risulta credibile e pertinente alle fonti.

Scripting procedurale

Le regole create in questa fase di progetto sono, complessivamente, tre:

1. Regola per la generazione delle geometrie relative agli edifici,
2. Regola necessaria alla modellazione procedurale delle strade,
3. Regola di utilità prettamente estetica, di inserimento nelle strade di alcuni asset premodellati in modo da "popolare" la scena (set dressing).

Edifici

Le regole per la generazione delle geometrie rappresentanti gli edifici hanno seguito le indicazioni fornite dalle fonti sulle strutture già citate in **??**. Per realizzarle, è stata sfruttata ripetutamente la funzione di split delle forme in modo da suddividere le varie aree ed assegnare loro regole differenti per applicare texture, eliminarle o continuare ad operare su di esse.

La prima regola, ad ogni modo, è del tutto simile a quella già utilizzata nel modello della città contemporanea e si occupa di selezionare, per la generazione, soltanto le shape idonee.

```
Lot  --> case (ALT_UV) <= 1:
      NIL
   case HIDE || enclosed:
      NIL
```

```
   case ny<-0.5:
      NIL
   case geometry.area < 20:
      NIL
   case DEC_TY=="Chiesa" || DEC_TY=="Campanile" :
      extrude(world.y,ALT_UV)
      MassChurch
   else:
      GenTower GenBuilding
```

Successivamente, le regole si occupano di creare la geometria degli edifici principali.

Inizialmente il codice utilizza una delle funzioni fornite da City Engine per estrarre un rettangolo ritagliato all'interno della shape alla base (questo nel caso in cui non abbia una forma regolare).

```
GenBuilding --> innerRect GenBuildProb
```

Dopodiché, viene indicato se l'edificio avrà o meno un cortile sul retro.

```
GenBuildProb -->case p(0.5):
      split(z) {scope.sz/100*4: Cinta |
      scope.sz/100*41: CortileCasa |
      scope.sz/100*55: Edificio}
   else:
      Edificio
```

Di qui, la costruzione della struttura vera e propria consiste in una serie di regole successive, le quali controllano se sia presente o meno un portico

```
PorticoNonPortico --> case !LayoutVisible:
   GroundF
else:
   GroundFPortico
```

per poi indirizzare la generazione verso il ramo di regole predefinito.

Strade

La generazione delle strade e degli incroci (che, ricordiamo, City Engine tratta come due entità separate) si occupa di applicare delle texture elaborate con il software The Gimp alle shape generate automaticamente dal software sullo streetgraph.

L'unico accorgimento usato è stato quello di creare una semplice funzione `calcLanes` che restituisse il numero di corsie di una strada, in modo da mappare la texture in maniera proporzionata ed evitare aberrazioni visive nel modello finale.

```
calcLanes(streetwidth) =
streetwidth/ceil(streetwidth/avgLanewidth)
```

Asset delle strade

Ultimo set di regole create è quello dell'inserimento degli asset, un lavoro più di abbellimento che di reale esigenza filologica. Tuttavia, la sua realizzazione risponde all'esigenza "emozionale" che il filmato usa come espediente per trasmettere cultura.

Le regole definite, per ovviare alla ridondanza delle informazioni e creare geometrie inutili, si occupa di selezionare le varie shape che compongono le strade, di cancellarle e inserire sopra allo spazio da loro occupato, il modello poligonale di un carretto. Si è scelto di non procedere con l'inserimento di ulteriori asset, quali modelli di personaggi e oggetti di vario genere che avrebbero potuto caratterizzare il periodo storico, sia per evitare, come già accaduto in altre occasioni, di appesantire la geometria dei modelli finali, sia perché la produzione del filmato non richiedeva un livello di dettaglio tale da giustificarne la presenza (Fig. 4.8).

4.2.3 Produzione del modello Hi-Poly

Se con la modellazione procedurale si è creato l'aspetto in senso generale della città, con quella tradizionale in Blender sono stati realizzati, invece, tutti quegli scenari che prevedevano un livello di dettaglio alto.

Questi scenari, che corrispondono nel cortometraggio alle scene 7 e 9, rappresentano una panoramica di piazza Santo Stefano e coinvolgono una corsa in soggettiva che Apa compie lungo un percorso definito dal regista attraverso i portici di Bologna.

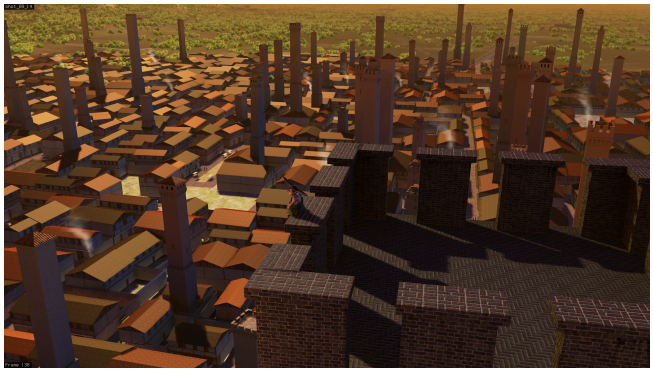

Figura 4.8: Un frame tratto dalla visuale panoramica della città medievale ricostruita

L'intera ricostruzione dell'aspetto è stata fatta a partire dai modelli di edificio e torre e dagli asset presenti nel progetto NuME.

Tali modelli, sufficientemente vari tra loro, una volta duplicati ed adattati, hanno realizzato una scenografia perfetta per ricreare l'aspetto di Bologna, mantenendosi in linea con il mood creato proceduralmente.

Va notato come, per la realizzazione di ognuno dei set e per conformarsi ad uno standard definito ad inizio progetto, ognuna delle scene create all'interno di Blender sia stata realizzata posizionando telecamere, oggetti e personaggi all'interno dello scenario in modo coerente con il modello georeferenziato generato con City Engine.

Piazza Santo Stefano

La scena 7 del cortometraggio, corrispondente a una visuale della piazza bolognese, è stata quella che ha richiesto il lavoro meno impegnativo dal punto di vista della preparazione. Difatti, l'edificio principale visibile, che caratterizza l'aspetto dell'area, è l'abbazia, il cui modello era già presente, per lavori precedenti, nell'archivio del Cineca. Unica modifica ad esso apportata, per mantenere la coerenza con le fonti storiche che ne datavano la realizzzione ad un periodo successivo il 1200, è stata la rimozione del pulpito e del rosone sulla facciata della chiesa.

Gli edifici circostanti alla piazza e il fondo stradale, non esistendo fonti particolari che ne attestassero l'aspetto preciso in epoca medievale, sono stati

ripresi da NuME ed opportunamente adattati nelle forme per entrare a far parte del set.

Corsa sotto i portici

La realizzazione della corsa sotto i portici, culminante con la scalata delle due torri nella città medievale, ha richiesto un lavoro più raffinato rispetto agli altri set.

I movimenti della telecamera, per la scena in soggettiva della corsa, non erano modificabili in modo semplice. Basandosi, infatti, sullo storyboard creato a partire dal modello procedurale, il breakdiwn prevedeva un percorso attraverso due porzioni distinte della città ben definite, con inquadrature e movimenti macchina che, per spettacolarizzare la scena, in più di un punto passano rasenti alla geometria.

Questo ha imposto innanzitutto di ricreare le geometrie procedurali esplorate durante la corsa, con due set "montati" attraverso i modelli ricavati da NuME. Successivamente è stato aggiuto un "dressing" con dei modelli ricavati sia da NuME che da alcuni asset dell'Open Project Durian della Blender Foundation (`www.sintel.org`). In particolare, sono state attinte bancarelle e oggettistica in generale quali scampoli di stoffa, frutti e suppellettili.

La scalata delle torri

I modelli delle torri Garisenda e Asinelli, sono stati rielaborati anch'essi a partire da quelli creati per il progetto NuME, il quale, tuttavia, ne forniva una rappresentazione ancora troppo poco dettagliata per le esigenze del filmato. Si è, quindi, realizzata una serie di modelli limitati a ciò che il cortometraggio doveva mostrare, come accade per esempio negli shot 9-11, 9-12 e 9-13, dove la parte di torre non inquadrata è stata cancellata dalla geometria per permettere di alzare il livello di dettaglio nella porzione visibile e non appesantire troppo la geometria con conseguenti rallentamenti durante il processo di rendering (Fig. 4.9).

Figura 4.9: Un frame del filmato, sono state modellate solo le porzioni visibili delle torri

Capitolo 5

Conclusioni

Il presente lavoro illustra il contributo fornito al progetto Cineca per la realizzazione del cortometraggio 3D stereoscopico intitolato 'Apa alla scoperta di Bologna', attualmente visionabile presso il Museo della storia di Bologna (Palazzo Pepoli [1], all'interno della stanza immersiva appositamente progettata e realizzata anch'essa dal Cineca.

Il mio contributo come stagista e in seguito come tesista ha condotto alla realizzazione della modellazione procedurale di due set: la Bologna attuale e i modelli high e low poly della Bologna medievale.

La realizzazione del cortometraggio Apa ha l'obiettivo di contribuire alla realizzazione di un museo moderno e di facile fruibilità che sfrutti tecnologie all'avanguardia per raccontare la storia della città da un punto di vista più vicino al visitatore che al curatore della collezione. Per questo la sala immersiva è stata fin da subito collocata al centro del percorso museale: la narrazione per immagini facilita la comprensione del messaggio culturale proposto, ovvero il passaggio di informazione culturale connessa alla filologicità dei set così come del suo protagonista, Apa, un etrusco furiuscito da una situal conservata nel Museo Civico Archeologico di Bologna.

Il progetto ha coinvolto professionalità eterogenee: a ingegneri e informatici si sono affiancati gli esperti storici e archeologi, un regista, un art director, un concept artist per la realizzazione dei bozzetti 2D, l'animatore, esperti di produzione cinematografica 3D e il direttore di stereoscopia.

Il mio lavoro si è svolto a partire dall'Aprile del 2010 e ha coinvolto attività di progettazione e insieme di realizzazione. Alla base del progetto, infatti, l'impegno a delienare una pipeline di produzione che integrasse i vincoli filologici andando a definire le regole di 'buona modellazione' per oggetti che fossero riu-

[1] `http://www.genusbononiae.it/index.php?pag=25`

tilizzabili nel tempo. E in effetti, le soluzioni da me elaborate si sono rivelate efficaci e utili per chi, in un secondo momento, ha dovuto modificare i modelli per amalgamarli alle esigenze artistiche del prodotto. Nella scena medioevale in high-poly, per esempio, sono stati aggiunti elementi di decoro utili alla resa visiva del filmato in quanto la modellazione procedurale permette un livello di dettaglio che mal si concilia con una resa grafica di tipo fotorealistico come, invece, è richiesto in un cortometraggio animato per l'edutainment.

Ai fini del progetto, CityEngine, unico tool non open-source dell'intera pipeline produttiva, si è dimostrato essere il solo software in grado di soddisfare i bisogni e risolvere le problematiche insite nella speciale pipeline che si stava delineando, mantenendo il giusto trade off tra tempo di realizzazione e accuratezza dei risultati. Non ultima, la semplicità di utilizzo per un utente con capacità di programmazione, lo ha reso uno strumento interessante che coniuga Virtual Heritage e grafica 3D alle competenze informatiche.

L'intero progetto mi ha, pertanto, consentito di perseguire l'obiettivo personale di alimentare la mia passione per la computer graphics, venendo a contatto con figure professionali proprie del settore in un contesto multidisciplinare stimolante. Ho imparato, attraverso la pratica, quali siano i passaggi necessari alla realizzazione di un filmato di animazione 3D. L'inserimento in un progetto con finalità di Cultural Heritage ha fatto crescere in me un interesse inaspettato verso la realizzazione di modelli filologicamente accurati e mi piacerebbe aver modo di proseguire il mio cammino professionale in questo ambito che a mio parere coniuga capiacità artistiche e insieme informatiche.

Appendice A

Codice Bologna Contemporanea

A.1 Alberi.cga

```
colore = "#"+ceil(rand(5))+ceil(rand(5))
           +"cc"+ceil(rand(9))+ceil(rand(9))

#Alberi
alberello = "albero1.obj"
const scala = rand (15,25)
const rotazione = rand(360)

Lot -->s(scala,scala,scala)
t(-7,0,-7)
i(alberello)
color(colore)
```

A.2 AlberiBillboard.cga

```
const alberino = "tex_alberi/albero"
                   +ceil(rand(0,4))+".png"

Lot --> extrude(world.y,1) Volume

Volume --> comp(f) {front: Albero | all: NIL}

attr sz = rand(10,14)
```

```
Albero --> s(sz,sz,scope.sz)
   setupUV(0,scope.sx,scope.sy)
   set(material.colormap,alberino)
   bakeUV(0)
```

A.3 Facciate.cga

```
# ok -- cornice
# ok --porte portici

# --- palazzi 6,7 piani
# tex dirt
# canala gronda
# cornici piani
# tende dentro finestre

attr ALTEZZA =3
attr WIN_H = 1.3 #1.5
attr WIN_W = 0.5 #0.6
attr COL_W = 0.3

attr CORNICE = 50% :  0
               25% :  0.1
               else: 0.2

attr FINESTRA = 60% : "F0.obj"
                else: "F1.obj"

attr PERSIANA = 30% : ""
                15% : "A1.obj"
                15% : "A2.obj"
                15% : "A3.obj"
                else: "A0.obj"

attr COLORE_PERSIANA =  25% : "#87979E"
                        25% : "#7b6b68"
                        25% : "#677924"
                        else: "#95A190"
```

```
attr VETRO = "#D8E6EE"

attr COLORE_MURO = 10% : "#DDA08C"
                   10% : "#CE8E67"
                   10% : "#F0C466"
                   10% : "#DDB459"
                   10% : "#BD836A"
                   10% : "#E3A652"
                   10% : "#CAA170"
                   10% : "#C27B6F"
                   else: "#E4BEB7"

attr COLORE_TENDA = 30% : "#d2776b"
                    30% : "#B97E6F"
                    else: "#AB7575"


Lot --> extrude(world.y, ALTEZZA) Mass

Mass--> split(y) { 0.05:Marciapiede |
                   ~3:PianoTerra |
                   {~3:Piano}* |
                   0.05:Tetto}
Marciapiede --> color("#AAAAAA")

Tetto --> s('1,'1,'1.2) color("#AAAAAA")

Piano --> comp(f) { front:Facciata |
                    side:Wall |
                    top: NIL |
                    bottom:Wall}

Wall --> color(COLORE_MURO)

Facciata --> split(x) { {3:PareteFinestra}* }

PareteFinestra --> split(y) { ~1:Wall |
```

```
                              CORNICE: AreaCorniceH |
                              WIN_H:PareteFinestraCenterH |
                              CORNICE: AreaCorniceH |
                              ~1:Wall}

PareteFinestraCenterH --> case PERSIANA == "":
                             AreaFinestraSenzaAnte
                          case p(0.3):
 AreaFinestraAperta
  else:
                             AreaFinestraChiusa

AreaFinestraSenzaAnte-->split(x) { ~1:Wall |
                                   CORNICE: Cornice|
                                   2*WIN_W:AreaFinestraSenzaAnte2 |
                                   CORNICE: Cornice |
                                   ~1:Wall}

AreaFinestraSenzaAnte2  --> split(y) { 0.05:Soglia |
                                     ~1:AreaFinestraSenzaAnte3}

AreaFinestraSenzaAnte3 --> t(0,0,-0.2)
                           extrude(world.z, 0.2)
                           comp(f) { front:Gray |
                                     side:Wall |
                                     top:NIL |
                                     bottom:Finestra Tenda }

AreaFinestraAperta --> split(x) { ~1:Wall |
                                  WIN_W:Persiana |
                                  2*WIN_W:AreaFinestra |
                                  WIN_W:Persiana |
                                  ~1:Wall}

AreaFinestraChiusa --> split(x) { ~1:Wall |
                                  CORNICE: Cornice |
                                  WIN_W:Persiana |
                                  WIN_W:Persiana |
```

```
                                    CORNICE: Cornice |
                                    ~1:Wall}

AreaFinestra  --> split(y) { 0.05:Soglia |
                             ~1:AreaFinestra2}

AreaFinestra2 --> t(0,0,-0.2)
                  extrude(world.z, 0.2)
                  comp(f) { front:Gray |
                            side:Wall |
                            top:NIL |
                            bottom:Finestra }

Finestra --> [split(x) { WIN_W:AntaFinestra |
                         WIN_W:AntaFinestra }] Vetro

AntaFinestra --> i(FINESTRA)
                 color("#DDDDDD")

Persiana --> i(PERSIANA)
             color(COLORE_PERSIANA)

Vetro --> color(VETRO)
          t(0,0,0.1)

Soglia --> extrude(world.z, 0.1)
           color("#AAAAAA")

Tenda  -->  color(COLORE_TENDA) Tenda1

Tenda1 -->case p(0.8):
              t(0,'0.8,-0.1)
            else:
              t(0,'0.5,-0.1)

AreaCorniceH --> split(x) { ~1:Wall |
                            2*WIN_W+2*CORNICE:Cornice |
                            ~1:Wall}
```

```
Cornice --> color("#CCCCCC")

PianoTerra --> case ALTEZZA <= 3:
                  Piano
               case p(0.9):
                  Portico
               else:
                  Piano

Portico --> split(x) { {3:PorticoElemento}* }

PorticoElemento --> split(z) { COL_W:AreaPorta |
                               ~1:NIL |
                               COL_W:PorticoFront }

PorticoFront --> split(x) { COL_W:Col |
                            ~1:NIL |
                            COL_W:Col }

AreaPorta  --> case p(0.7):
                  comp(f) { front:AreaPorta0 |
                            side:Wall }
               else:
                   Wall

AreaPorta0 --> split(y)  { 2.2:AreaPorta1 |
                           ~1:Wall }

AreaPorta1 --> split(x) { ~1:Wall |
                          1+rand(0.5):AreaPorta2 |
                          ~1:Wall }

AreaPorta2 --> t(0,0,-0.2)
               extrude(world.z, 0.2)
               comp(f) { front:Marciapiede |
                         side:Wall |
                         top:NIL |
```

```
                        bottom:Porta  }

Porta --> color(COLORE_PERSIANA)

Col --> split(y) { COL_W:Capitello |
                   ~1:Wall |
                   COL_W:Capitello }

Capitello --> s('1.1,'1.1,'1.1)
              color("#AAAAAA")

Gray --> color("#AAAAAA")

Y --> color("#FFFF00")

R --> color("#FF0000")

G --> color("#00FF00")

B --> color("#0000FF")
```

A.4 Crop_facciate.py

```
#! /usr/bin/python
# -*- coding: utf-8 -*-
import Image
im = Image.open("Facciate.jpg")
ascisseB = (1864,1677,1402,1035,576,28)
ascisseE = (1954,1856,1670,1390,1023,565)
ordinateB = (1386,1178,879,490,16)
ordinateE = (1474,1354,1144,848,459)
for x in range(6):
    for y in range(5):
        fox = (ascisseB[x],
               ordinateB[y]+14,
               ascisseE[x],
               ordinateE[y]+14)
        tex = im.crop(box)
```

```
        tex.save("f" + str(y+1) + "t" + str(x+1) + ".jpg")
```

A.5 EdificiCentro.cga

```
torre = "torre/torre.obj"
tex_torre = "torre/torre.png"

attr tetto_con_coppi = prob(90)
attr ALT_UV = 10  # altezza di gronda
attr DEC_TY = "" # stringa tipologia edificio"
attr VOL_UV = 10
attr portico= false
attr DEC_TY_E= ""

attr HIDE = false

attr wallColor = 15%: "#FEDCCE"
                 6%  : "#E3CBA1"
                 4%  : "#DBBC8B"
                 4%  : "#D4A896"
                 15% : "#F7C7A1"
                 4%  : "#D4D6C2"
                 4%  : "#DFDECB"
                 15% : "#F8BC76"
                 15% : "#F8CC76"
                 else: "#FBE5BB"

attr roofColor = 12% :"#5F4D30"
                  12% :"#735C36"
                  12% :"#836841"
                  12% :"#804B1E"
                  12% :"#8A4F1A"
                  12% :"#7B410D"
                  12% :"#8A4F1A"
                  else:"#A8725A"

attr flatRoofColor = 25% :"#AAAAAA"
                     25% :"#BBBBBB"
```

```
                    25% :"#CCCCCC"
                    else:"#999999"

attr churchColor = "#FEDCCE"

########################

camino_w = 1
camino_h = rand(1,1.5)

const Tiles = 3
#Larghezza di ogni finestra
const tileW = 6

#Altezza di ogni piano
const floorHeight = 6

const maxTextureTiles = 6
const maxFloors = 5
const nTextureFloors  = case (scope.sy<floorHeight*maxFloors):
                           ceil(scope.sy/floorHeight)
                        else:
                           maxFloors
nTextureTiles = case (scope.sx<tileW*maxTextureTiles):
                   ceil(scope.sx/tileW)
                else:
                   maxTextureTiles

########################

# componente y della normale del Lot
const ny = cos(initialShape.origin.ox)
              * cos(initialShape.origin.oz)
              + sin(initialShape.origin.ox)
              * sin(initialShape.origin.oy)
              * sin(initialShape.origin.oz)

########################
```

```
# general util functions
#
prob(percentage) = case percentage > rand(100): true else: false

const getRoofTexture = "tex_facciate/Coppi_0"
                          +ceil(rand(0,4))
                          +".jpg"
const getFlatRoofTexture = "tex_facciate/Concrete_0"
                              +ceil(rand(0,8))
                              +".jpg"
getBuildingTexture(floors,tiles) = "tex_facciate/f"
                                      +floors
                                      +"t"
                                      +tiles
                                      +".jpg"

########################
# Begin
#

Lot  --> case (ALT_UV) <= 1:
            NIL
         case HIDE:
            NIL
         case ny<-0.5:
            NIL
         case geometry.area < 20:
            NIL
         else:
            extrude(world.y,ALT_UV)
            Extr

Extr -->alignScopeToAxes(y)
        split(y){ALT_UV : Mass |
                 ~1 : NIL}

Mass --> case DEC_TY=="Chiesa" || DEC_TY=="Campanile" :
            comp(f) {bottom: NIL |
```

```
                    side: Church |
                    top: ChurchRoof0 }
         else:
            case ALT_UV > 37.2:
               Torre
            else:
               comp(f) {bottom: NIL |
                        side: Side |
                        top: Top }

Church --> case inside():
              NIL
           else:
              color (churchColor)
              setupProjection (0,scope.xy,20,20,1)
              projectUV(0)

ChurchRoof0 --> roofHip(15,0.5)
                comp(f) {top: ChurchRoof }

ChurchRoof --> setupProjection (0,scope.xy,6,6,1)
               projectUV(0)
               set (material.colormap,
                    "tex_facciate/Coppi_chiese.jpg" )

Torre --> s('1,ALT_UV,'1)
          set(material.colormap, tex_torre)
          i(torre)

Top --> case tetto_con_coppi && ALT_UV > 4.8 :
           roofHip(15,1)
           comp(f) {top: Roof }
        else :
           FlatRoof

Roof -->case p(geometry.area * 0.01):
           FaldaConCamino
        else:
```

```
           Falda

Falda --> setupProjection (0,scope.xy,6,6,1)
          projectUV(0)
          set (material.colormap, getRoofTexture() )

FaldaConCamino --> Texturing
                   split(x) { ~1: NIL |
                   camino_w: FaldaConCamino1 |
                   ~0.5: NIL }

FaldaConCamino1 --> split(y) { ~1: NIL |
                               camino_w:BaseCamino |
                               ~0.5: NIL }

BaseCamino --> extrude(world.y,camino_h *2)
               Camino

Camino --> alignScopeToAxes(y)
           split(y) {camino_h: comp(f) {top : Falda |
                                        bottom : NIL |
                                        side: LatoCamino} |
                     ~1: NIL}

Texturing --> setupProjection (0,scope.xy,6,6,1)
              projectUV(0)
              set (material.colormap, getRoofTexture() )

LatoCamino --> alignScopeToAxes(y)
               setupProjection (0,scope.xy,20,20,0)
               projectUV(0)
               color("#FEDCCE")

FlatRoof --> setupUV(0,scope.sx,scope.sy)
             bakeUV(0)
             set (material.colormap, getFlatRoofTexture )

getScopeY = case (nTextureFloors * floorHeight < scope.sy * 0.85):
```

```
            scope.sy/2
         else:
            scope.sy

Side --> case scope.sx>maxTextureTiles*tileW*0.7:
         split(x){~1.5: Side |
                  ~1: Side}
      else:
         alignScopeToAxes(y)
         setupUV(0,scope.sx,getScopeY)
         bakeUV(0)
         set (material.colormap,
              getBuildingTexture(nTextureFloors,nTextureTiles))
```

A.6 EdificiPeriferia.cga

```
version "2009.3"

attr tetto_con_coppi = prob(90)
attr ALT_UV = 10  # altezza di gronda
attr DEC_TY = "" # stringa tipologia edificio"
attr VOL_UV = 10
attr portico= false
attr DEC_TY_E= ""

attr HIDE = false

attr wallColor = 15% : "#FEDCCE"
                 6%  : "#E3CBA1"
                 4%  : "#DBBC8B"
                 4%  : "#D4A896"
                 15% : "#F7C7A1"
                 4%  : "#D4D6C2"
                 4%  : "#DFDECB"
                 15% : "#F8BC76"
                 15% : "#F8CC76"
                 else: "#FBE5BB"
```

```
attr roofColor = 12% : "#5F4D30"
                 12% : "#735C36"
                 12% : "#836841"
                 12% : "#804B1E"
                 12% : "#8A4F1A"
                 12% : "#7B410D"
                 12% : "#8A4F1A"
                 else: "#A8725A"

attr flatRoofColor = 25% : "#AAAAAA"
                     25% : "#BBBBB"
                     25% : "#CCCCCC"
                     else: "#999999"

attr churchColor = "#FEDCCE"

########################

#Larghezza di ogni finestra
const tileW = 6

#Altezza di ogni piano
const floorHeight = 6

const maxTextureTiles = 6
const maxFloors = 5

const nTextureFloors = case (scope.sy<floorHeight*maxFloors):
                          ceil(scope.sy/floorHeight)
                       else:
                          maxFloors

nTextureTiles = case (scope.sx<tileW*maxTextureTiles):
                   ceil(scope.sx/tileW)
                else:
                   maxTextureTiles
```

```
########################

# componente y della normale del Lot
const ny = cos(initialShape.origin.ox)
              * cos(initialShape.origin.oz)
              + sin(initialShape.origin.ox)
              * sin(initialShape.origin.oy)
              * sin(initialShape.origin.oz)

########################
# general util functions
#
prob(percentage) = case percentage > rand(100):
                      true
                   else:
                      false

const getRoofTexture = "tex_facciate/Coppi_0"
                          +ceil(rand(0,4))
                          +".jpg"
const getFlatRoofTexture = "tex_facciate/Concrete_0"
  +ceil(rand(0,8))
  +".jpg"
getBuildingTexture(floors,tiles) = "tex_facciate/f"
                                      +floors
                                      +"t"
                                      +tiles
                                      +".jpg"

########################
# Begin
#

Lot  --> case (ALT_UV) <= 1:
            NIL
         case HIDE:
            NIL
         case ny<-0.5:  #poligoni con la normale verso il basso
```

```
            NIL
         case geometry.area < 20: # too small
            NIL
         else:
            extrude(world.y,ALT_UV)
            Extr

Extr --> alignScopeToAxes(y)
         split(y){ ALT_UV : Mass |
                   ~1 : NIL}

Mass --> case ALT_UV>37.2||DEC_TY=="Chiesa"||DEC_TY=="Campanile" :
            comp(f) {bottom: NIL |
                     side: Church |
                     top: Church }
         else:
            comp(f) {bottom: NIL | side: Side | top: Top }

Church --> case inside():
              NIL
           else:
              color (churchColor)

Top --> case tetto_con_coppi && ALT_UV > 4.8 :
           comp(f) {all: Roof }
        else :
           FlatRoof

Roof --> setupProjection (0,scope.xy,6,6,1)
         projectUV(0)
         set (material.colormap, getRoofTexture() )

FlatRoof --> setupUV(0,5,5)
             bakeUV(0)
             set (material.colormap, getFlatRoofTexture )

getScopeY = case(nTextureFloors*floorHeight < scope.sy*0.85):
               scope.sy/2
```

```
            else:
               scope.sy

Side --> case inside():
            NIL
         case scope.sx>maxTextureTiles*tileW*0.7:
            split(x){~1.5: Side|~1: Side}
         else:
            alignScopeToAxes(y)
            setupUV(0,scope.sx,getScopeY)
            bakeUV(0)
            set (material.colormap,
                 getBuildingTexture(nTextureFloors,nTextureTiles))
```

Appendice B

Codici Bologna Medievale

B.1 Bologna1200.cga

```
attr LayoutVisible = false
attr Tower = false
attr HIDE = false
attr enclosed = false
attr DEC_TY = ""
attr ALT_UV = 10

attr churchColor = 25%: "#FEDCCE" #rosa
   15%  : "#E3CBA1" #marroncino
   10%  : "#DBBC8B" #marrone
   35% : "#F7C7A1" #beige
   else: "#FBE5BB" #panna

const numTower = ceil(rand(5))

const W_TRAVI = 0.2
const tower_w = rand(7,9)
const tower_h = tower_w*rand(3.5,10)
torre(n) = "torre"+n+".obj"

const texlegno = "legno_travi" + ceil(rand(3)) + ".jpg"
const texroof = "tegole_pulite" + ceil(rand(3)) + ".jpg"
const texdoor = "porta" + ceil(rand(3)) + ".jpg"
```

```
const texwall = "muro" + ceil(rand(3)) + ".jpg"

# componente y della normale del Lot
# eliminare il Lot se ny < -0.5
# cioe' e' il Lot di un buco
const ny = cos(initialShape.origin.ox) *
   cos(initialShape.origin.oz)
   + sin(initialShape.origin.ox)
   * sin(initialShape.origin.oy)
   * sin(initialShape.origin.oz)


//=========================================
//MATERIALS
//=========================================

Muro --> setupProjection(0,scope.xy, 9, 6,1)
   set(material.colormap, texwall)
   bakeUV(0)

TexLegno --> set(material.colormap, texlegno)
   bakeUV(0)

TexLegnoR --> r(0,0,90)
   set(material.colormap, texlegno)
   bakeUV(0)
   r(0,0,-90)

TexPorta --> setupUV(0,scope.sx,scope.sy)
   set(material.colormap, texdoor)
   bakeUV(0)

TexTrave --> color("#ffffff")

Coppi --> setupProjection (0,scope.xy,2,2,1)
   projectUV(0)
   set (material.colormap, texroof )
```

```
//============================================
//GEOMETRY
//============================================

Lot  --> case (ALT_UV) <= 1:
      NIL
   case HIDE || enclosed:
      NIL
   case ny<-0.5:
      NIL
   case geometry.area < 20:
      NIL
   case DEC_TY=="Chiesa" || DEC_TY=="Campanile" :
      extrude(world.y,ALT_UV)
      MassChurch
   else:
      GenTower GenBuilding


MassChurch --> comp(f) {bottom: NIL |
   side: Church |
   top: ChurchRoof0 }

Church -->case inside():
      NIL
   else:
      color (churchColor)

ChurchRoof0 --> roofHip(15,0.5)
   color("#992222")


GenTower --> case Tower: TorreSplitX
   else: NIL


GenBuilding --> innerRect GenBuildProb
```

```
GenBuildProb -->case p(0.5):
      split(z) {scope.sz/100*4: Cinta |
      scope.sz/100*41: CortileCasa |
      scope.sz/100*55: Edificio}
   else:
      Edificio

CortileCasa --> split(x) {0.3: Cinta |
   ~1 : Cortile |
   0.3: Cinta}

Cortile --> color("#339944")

Cinta --> extrude(world.y,5) BaseCuspideCinta

BaseCuspideCinta --> comp(f) {top: Cuspide |
   all: Muro}

Cuspide --> roofGable(45) TexCusp

TexCusp --> comp(f){all: Muro}

Edificio --> extrude(world.y,rand(7,11)) Casa

Casa --> split(y) {~1: PorticoNonPortico |
   ~1: FirstF}

PorticoNonPortico --> case !LayoutVisible:
   GroundF
else:
   GroundFPortico

GroundF --> split(z) {~1: GroundBlock |
   0.5: Prefacciata}

Prefacciata --> NIL

GroundBlock --> comp(f) {front : FacciataGF |
```

```
   back : RetroGF |
   side : Muro}

FacciataGF --> split(y) {~3: AltezzaPorte |
   0.5 : AltezzaArchitrave}

AltezzaArchitrave --> split(x){~1: Muro |
   scope.sx-2: Architrave |
   ~1 : Muro}

Architrave --> extrude(0.2) TexTrave

AltezzaPorte --> case p(50):
   split(x){~rand(1,2): Muro |
      1.5 : SpPorta |
      ~rand(1,3): Muro |
      1.5 : SpPorta |
      1.5 : SpPorta |
      ~1 : Muro}
   else:
      split(x){~rand(1,5): Muro |
      1.5 : SpPorta |
      1.5 : SpPorta |
      ~rand(1,3): Muro |
      1.5 : SpPorta |
      ~1 : Muro}

SpPorta --> extrude(-0.1) Porta

Porta --> comp(f) {bottom: NIL |
   side : Muro |
   top: TexPorta}

RetroGF --> split(x) {~rand(10): MuroBack |
   1.3: DoorBack |
   ~1: MuroBack}

DoorBack --> split(y){2.5: NIL |
```

```
   ~1: MuroBack}

MuroBack --> extrude(-0.5) TexMuroBack

TexMuroBack --> alignScopeToAxes(y)
   comp(f){front: Muro}

GroundFPortico --> split(z) {~1: GroundBlock |
   2.2: Portico |
   0.2: ColonnatoY |
   0.2 : Prefacciata}

Portico --> NIL

ColonnatoY --> split(y){~1: Colonnato | 0.2: TexLegnoR}

Colonnato --> split(x){W_TRAVI: TexLegno |
   {~2: Infracolonna | W_TRAVI: TexLegno}*}

Infracolonna --> split(y){0.7: MurettoPortico |
   ~1: NIL | 1: Sostegni}

MurettoPortico --> case p(0.6):
   i("murettoColonne.obj") TexLegnoR
 else:
   NIL

Sostegni --> i("sostegniColonne.obj") TexLegnoR

FirstF --> setupProjection(0,scope.xy, 5, 7,1)
   FirstFloorOne FirstFloorTwo

FirstFloorOne --> comp(f) {top: Tetto}

FirstFloorTwo --> split(x) {0.2 : SideFloor |
   ~1 : BlockFloor |
   0.2 : SideFloor}
```

```
SideFloor --> comp(f) {front: TexLegno |
   side: WallFloorY |
   back: TexLegno}

WallFloorY --> split(y){0.2: TexLegnoR |
   ~1: WallFloor |
   0.2: TexLegnoR}

WallFloor --> split(x) {0.2: TexLegno |
   ~1: Muro |
   0.2: TexLegno |
   ~1: Muro |
   0.2: TexLegno}

BlockFloor --> split(y){0.2: AltezzaTravi |
   ~1 : FirstBlock}

AltezzaTravi --> split(z) {~1 : TexLegnoR |
   0.5 : AreaTravi}

AreaTravi --> split(x){~0.4: NIL |
   {0.2: TexLegnoR | ~0.4: NIL}*}

FirstBlock --> comp(f) {front : FacciataFF |
   back : FacciataFF |
   side: Muro}

FacciataFF --> split(y) {0.2 : TexLegnoR |
   ~1 : FacciataFloor |
   0.2 : TexLegnoR }

FacciataFloor --> split(x){~rand(2,5): RoomTile |
   {0.2 : TexLegno | ~rand(2,5): RoomTile }*}

RoomTile --> split(y){~1: Muro |
   0.2: TexLegnoR |
   ~1.2 : SpaceTile |
   0.2: TexLegnoR |
```

```
   ~0.1: Muro }

SpaceTile --> case p(0.5):
   split(x){~1 : Muro | 0.2 : TexLegno | 1 : TexLegno}
 else:
   split(x){1 : TexLegno | 0.2 : TexLegno | ~1 : Muro}

Tetto --> roofGable(20,0.5,0) TettoComp

TettoComp --> comp(f) {top: Coppi | side: TexLegno}


TorreSplitX --> split(x){~1: NIL |
   tower_w: TorreSplitY |
   ~rand(0,2): NIL}

TorreSplitY --> split(z){~rand(0,1): NIL |
   tower_w: Torre | ~1: NIL}

Torre --> s(tower_w,tower_h,tower_w)
   i(torre(numTower))
```

B.2 streets_1200.cga

```
## Textures
const road_tex = "strade/road.png"
const cross_tex = "strade/crossing.png"

## User attributes

attr avgLanewidth = 1

## Functions

// Calcola il numero di corsie in base alla larghezza
calcLanes(streetwidth) =
```

```
streetwidth/ceil(streetwidth/avgLanewidth)


/********************
   Rules
*********************/

Street -->
setupProjection(0,scope.xz, 6, calcLanes(scope.sz))
projectUV(0)
scaleUV(0, 1, 0.98)
texture(road_tex)

Crossing -->
setupProjection(0,scope.xz, scope.sx, scope.sz)
projectUV(0)
texture(cross_tex)
```

B.3 application_assets.cga

```
##Assets
const carretto = "strade/donna.obj"
/*
const animale = "strade/animale.obj"
const uomo = "strade/uomo.obj"
const donna = "strade/donna.obj"
const bancarella = "strade/bancarella.obj"
*/

##Rules

Crossing --> InsertionQuery

Street --> split(z){~1: Carreggiata |
   ~1: Carreggiata}

Carreggiata --> split(z){0.15 : NIL |
```

```
   ~0.5 : Lane |
   0.15 : NIL}

Lane --> split(x) {~rand(0,20): NIL |
   {1 : InsertionQuery | ~rand(2,20) : NIL}* |
   ~rand(0,10): NIL}

InsertionQuery --> case (rand > 0.7): Insertion
   else: NIL

Insertion --> s(.5,1.8,1)
   center(xz)
   i(carretto)
```

Bibliografia

[Ant10] F. Antinucci. *Comunicare nel museo*. Roma e Bari: Laterza, 2010.

[Be10] F. Bocchi and R. Smurra (eds.). *La storia della città per il Museo Virtuale di Bologna*. Bologna: Bononia University Press.a, 2010.

[Boc97] F. Bocchi. *Bologna e i suoi portici*. Grafis Edizioni, Bologna, 1997.

[Gil99] M. Gillings. Engaging place: a framework for the integration and realisation of virtual-reality approaches in archaeology. *in "Archaeology in the age of the Internet", CAA 1997. Edited by Dingwall, L., Exon, S., Gaffney, V., Laflin, S., Van Leusen, M., Oxford: British Archaeological Reports (Int. Series, S750)*, 1999.

[Pro08] Procedural Inc. *City Engine Examples: Venice*, 2008.

[Rei90] P. Reilly. Towards a virtual archaeology. computer applications in archaeology. *Edited by K. Lockyear and S. Rahtz, Oxford: British Archae- ological reports (Int. Series 565), pp. 133-139*, 1990.

[Rou] M. Rousseau. *Virtual Heritage: From the Research Lab to the Broad Public*, pages 93–100.

[SPC11] G. Lucci Baldassarri D. Ferdani S. Pescarin, B. Fanini and L. Calori. Archeologia virtuale, realismo, interattività e performance: dalla ricostruzione alla fruizione on line. realism, interactivity and performance: a pipeline for large scale virtual heritage dataset on line. *in "Tecnologie per la comunicazione del patrimonio culturale", pp. 62-70*, 2011.

[YIHP01] Pascal Müller Yoav I H Parish. Procedural modeling of cities. In *Siggraph*, 2001.