EPS
Escola Politècnica Superior
UdG

Escola Politècnica Superior
Universitat de Girona

Copy&Paste façanes procedurals per skylineEngine

Projecte/Treball Fi de Carrera

Enginyeria Superior en Informàtica. Pla 1997

Document: Memòria

Autor: Alejandro Arangua Rovira

Director: Gustavo Ariel Patow

Departament: Informàtica, Matemàtica Aplicada i Estadística

Àrea: LSI

Convocatoria: Setembre 2013

Índex

Capítol 1

Introducció

Avui en dia el modelatge de ciutats (actuals i històriques) és un problema obert pel que no existeixen solucions estàndards ni de codi lliure. Aquest és un problema força important per indústries com la del cinema, la realitat virtual i els videojocs, que sovint necessiten crear grans escenaris. Degut a això, cada vegada es fan servir més eines informàtiques per a la creació d'aquests entorns urbans.

Segurament tothom ha tingut l'oportunitat de veure algun videojoc (com per exemple Assasins Creed o GTA IV) o alguna pel·lícula (com per exemple Cars 2) on l'escenari escollit és una gran ciutat. En el cas de l'Assasins Creed o el GTA IV, els escenaris han estat creats manualment, és a dir, muntant els edificis un per un fins aconseguir una gran ciutat. Això és una feina que ha requerit milers d'artistes i una despesa de temps i de diners més que considerable.

El modelatge procedural pretén solucionar gran part d'aquest problema utilitzant algoritmes per tal de generar de forma automàtica les ciutats o edificis que es necessitin. El procés consisteix en deixar que l'ordinador faci la feina complicada i d'aquesta forma l'usuari simplement ha d'especificar els diferents paràmetres per tal d'aconseguir el resultat desitjat.

Figura 1.1: Modelat de ciutat 3D per a Assasins Creed.

1.1 Motivacions personals

Ja abans de començar l'enginyeria, i també durant aquesta, m'ha sigut de gran interes tot el món dels gràfics per computador. He fet servir eines de modelat tals com el 3dStudio, Maya, Brice, RealFlow i d'altres. És un tema que m'encanta i em realitza. He cursat optatives de la carrera com Multimèdia i Informàtica gràfica. I al finalitzar l'enginyeria tècnica, vaig presentar com a projecte una aplicació sobre la il·luminació per radiositat d'escenes 3D amb algorismes de MonteCarlo, que també és de l'àmbit gràfic.

Ara finalitzant la l'enginyeria Informàtica, se m'ha obert la possibilitat d'aprendre sobre Houdini, **skyline**Engine, **building**Engine i programar amb Python. Aquestes eines, com es veurà més endavant, conformen un paradigma de modelat 3D que per a mi era desconegut i, per tant, de gran interès.

1.2 Propòsits

L'objectiu d'aquest projecte és el desenvolupament d'una eina d'alt nivell pel modelatge d'edificis procedurals que permeti copiar i enganxar parts arbitràries d'un edifici en un altre. Els edificis procedurals es basen en l'execució iterativa d'un conjunt de regles, que es poden representar per un graf d'operacions. Per tant, l'operació de copiar i enganxar es centra en la reescriptura dels grafs de regles amb l'objectiu de modificar els edificis per tal de dur aquesta tasca. Donat que es treballa sobre la plataforma de recerca anomenada **skyline**Engine, que s'executa sobre el programari Houdini 3D, l'aplicació també estarà implementada a sobre d'aquesta plataforma.

Aquesta eina és un gran avanç respecte les eines que existeixen pels dissenyadors, ja que encapsula un gran ventall d'operacions de re-escriptura de grafs de forma senzilla i transparent per l'usuari.

Un objectiu secundari és integrar aquest mecanisme a dins del **building**Engine, que és part de **skyline**Engine, una llibreria pel desenvolupament procedural d'edificis i ciutats desenvolupada pel Grup de Geometria i Gràfics de la UdG.

1.3 Objectius

Malgrat la seva importància, fins ara no existeixen gaires opcions de codi lliure en quant a eines de creació de models urbans, a excepció del **skyline**Engine, una llibreria pel modelatge de ciutats i edificis desenvolupada al Grup de Geometria i Gràfics. El **building**Engine és un dels mòduls del **skyline**Engine que s'executa sobre Houdini, essent una plataforma genèrica pel modelatge procedural. Podem organitzar les tasques de la següent manera:

- Aprendre i manejar el llenguatge de programació Python.
- Estudi de la plataforma 3D "Houdini" i de les llibreries per la incorporació de scripts Python.
- Estudi del **building**Engine, que és el mòdul inclòs al **skyline**Engine encarregat de la definició d'edificis.
- Desenvolupament de les llibreries bàsiques per fer recorreguts configurables a l'estructura de dades generada que representa els edificis.
- Desenvolupament d'un sistema de selecció de la part de façana a copiar.

- Algorisme de reconeixement de les primitives involucrades en la generació de la part seleccionada.
- Emmagatzemament dels nodes (operacions geomètriques) participants.
- Copia dels nodes.
- Selecció de la part corresponent del graf de destí.
- Reescriptura del graf de destí per incorporar-se les parts copiades.
- Desenvolupament d'optimitzacions per a la interactivitat.
- Acceleracions en la interacció de l'usuari.
- Documentació del treball realitzat.

1.4 Estructura del document

Aquesta memòria està estructurada en quinze capítols que recullen tot el procés des del naixement de la idea del projecte fins als resultats i conclusions finals.

Capítol 1 - Introducció:
En aquest capítol s'explica el perquè del desenvolupament d'aquest projecte, quins són els objectius proposats i com s'ha organitzat el desenvolupament.

Capítol 2 - Estudi de viabilitat:
Aquest capítol és on es justifiquen els paràmetres que fan possible el desenvolupament del projecte.

Capítol 3 - Metodologia:
S'explica la metodologia utilitzada justificant les raons per les quals s'ha acabat adoptant aquesta metodologia i no d'altres.

Capítol 4 - Planificació:
Aquest apartat defineix l'estratègia seguida per arribar als objectius plantejats i la temporització del que s'ha desenvolupat.

Capítol 5 - Marc de treball:
Conté la descripció els diversos aspectes relacionats amb el desenvolupament general del projecte, així com les principals accions dutes a terme als inicis del projecte.

Capítol 6 - Requisits del sistema:
En aquest capítol es defineixen els requeriments del software, és a dir, sobre quins elements s'ha desenvolupat el projecte.

Capítol 7- Estudis i decisions:
Aquest apartat explica les eines que han estat utilitzades, les seves característiques i el paper que han tingut al projecte.

Capítol 8 - Anàlisi i disseny del sistema:
Aquest capítol compren la investigació del problema a resoldre. Mitjançant enginyeria del software es concreten les especificacions, esquemes d'implementació, classes i mètodes del

projecte.

Capítol 9 - Implementació i proves:
En aquest capítol s'explica com s'han implementat els diferents mòduls del projecte.

Capítol 10 - Resultats:
Conté proves d'execució de l'aplicació, i resultats d'exemple per mostrar com s'ha acabat implementant la nova funcionalitat.

Capítol 11 - Conclusions:
Aquest capítol exposa les conclusions obtingudes un cop finalitzat el projecte.

Capítol 12 - Treball futur:
S'expliquen possibles millores, innovacions i noves funcionalitats sobre la línia d'aquest treball per a versions posteriors.

Bibliografia:
Conté les referències utilitzades pel desenvolupament del projecte.

Capítol 13 - Annex:
Per finalitzar s'inclouen un seguit d'ampliacions per a un millor enteniment del projecte realitzat.

Capítol 14 - Manual d'usuari:
Es tanca el projecte amb un manual que explica el funcionament de la funcionalitat implementada per tal que s'entengui com fer-la servir correctament.

Capítol 2

Estudi de viabilitat

2.1 Recursos tècnics per poder desenvolupar el projecte

Aquest projecte inicialment no es preveu que tingui grans requeriments, tret d'un ordinador amb un mínim de potència gràfica que és fàcilment assolible per a la majoria d'ordinadors de l'actualitat.
A continuació es pot veure una taula de les especificacions tècniques de l'equip utilitzat:

Característiques tècniques de l'equip utilitzat:	
CPU	AMD PhenomII 955 x4 BlackEdition
Placa Mare	Asus M4A77TDPro
Ram	Kingstom 4Gb 1600Mhz (2x2 DC)
Gràfica	ATI AMD Radeon HD6850 1Gb
Disc Dur	WesterDigital 500Gb (Magnètic)
Pantalla	SyncMaster 940BW 19" (16:10)
Sistema operatiu	Windows 7 Enterprise x64

Figura 2.1: Especificacions tècniques de l'equip utilitzat.

2.2 Recursos humans

El projecte requeria d'un analista, un cap de projecte i un programador, tant per definir i organitzar els passos a seguir, decidir quins son els camins més viables a seguir, com per programar en funció de les decisions preses. Aquests rols han sigut desenvolupats per mi, tenint com a guia en tot moment el tutor del projecte.

2.3 Viabilitat tecnològica i econòmica

Per a realitzar un estudi sobre la viabilitat primer de tot s'ha d'esmentar que aquest treball és de recerca. Per la mateixa raó no hi ha d'haver impediments legals i burocràtics que impedeixin la realització d'aquest projecte.
Per altra banda, si que s'ha de comprovar que es tinguin els recursos tècnics necessaris, i per això s'ha donat a conèixer el hardware que s'ha fet servir. Aquest hardware és suficient per a cobrir les necessitats d'aquest projecte.
El software que s'ha requerit com a principal eina és el Houdini 3D, amb llicència de tipus "Apprentice" que és de franc, tot i que inclou una sèrie de limitacions, però aquestes no són

limitadores per al projecte. Les limitacions afecten al renderitzat d'escenes, on hi solen inclou-re una marca d'aigua, i a més es limiten les resolucions. Aquest software es pot trobar a la web de SideFX, empresa creadora de Houdini 3D.

Vist això, no es requereix de cap pressupost inicial per a poder començar a dissenyar i implementar aquest projecte, ja que es disposa del hardware, software i material necessari, com s'ha esmentat.

2.4 Cost econòmic

Com s'ha dit anteriorment, aquest projecte és de recerca, per tant no ha de ser forçosament viable de manera econòmica. Tot i això s'ha fet un estudi segons les expectatives i els recursos, tant tecnològics com humans, que s'han fet servir.

En el cas dels recursos tècnics en quant a software, les especificacions descrites anteriorment no suposen cap cost pel fet de treballar sobre llicències d'aprenent.

El cost del maquinari, però, s'ha de tenir en compte ja que l'adquisició d'aquest ha sigut clau per el desenvolupament del projecte. Per tant l'amortització de l'equip durant els 14 mesos de desenvolupament és la següent:

- **Cost Total Maquinari:** 1000€
- **Nombre mesos treballats:** 14 Mesos (Juny 2012 - Setembre 2013)
- **Mesos d'amortització totals d'equips informàtics:** 36 Mesos (3anys)

Per tant:
Cost amortització= $\frac{1000€}{36Mesos} * 14Mesos = 389€$

Els recursos humans tenen diferent cost en funció de la feina que es desenvolupa, per exemple l'analista té un cost superior al del programador.

- Analista: 20€/h
- Programador: 10€/h

Les feines a desenvolupar son:

- Estudi de les eines
- Disseny del pipeline de la nova funcionalitat
- Disseny d'estructura de classes i algorismes principals
- Implementació
- Proves
- Memòria

Per tant el cost en funció del perfil de les tasques és el següent:

Tasques	Perfil	Hores	Cost€/hora	Cost total
Estudi de les eines	Analista	100	20€/h	2.000€
Disseny del pipeline de la nova funcionalitat	Analista	150	20€/h	3.000€
Disseny d'estructura de classes i algorismes principals	Analista	200	20€/h	4.000€
Implementació	Programador	600	10€/h	6.000€
Proves	Programador	100	10€/h	1.000€
Memòria	Analista	200	20€/h	4.000€
Total Programador:		350	10€/h	7.000€
Total Analista		325	20€/h	13.000 €
Total Amortització Hardware				389€
Total				20.389€

2.5 Conclusions

Pel que s'ha vist en els apartats anteriors, la viabilitat del projecte de recerca està garantida a nivell tècnic. Aixo és degut a que ja es disposa dels requeriments tècnics tant bon punt es comença el projecte, el programari que s'utilitza permet el seu us per l'aprenentatge o és codi lliure.

En canvi, a nivell econòmic, si s'hagués de remunerar a cadascuna de les tasques en funció del perfil que més s'adequa, suposaria un cost de 20.000€.

Capítol 3

Metodologia

3.1 Metodologia

Actualment hi ha moltes metodologies de desenvolupament eficients i diferenciables, des de les més antigues com la metodologia Waterfall fins a les més modernes tècniques Agile. Després d'estudiar diferents metodologies tals com Spiral, Scrum o Iterative, i les esmentades anteriorment, s'ha decidit no fer-ne servir cap en concret.

El que s'ha fet és seguir la metodologi **skyline**Engine, ideada especialment per als projectes derivats d'aquesta plataforma. Aquesta metodologia ja es va fer servir anteriorment i ha donat molt bons resultats als anteriors projectes sobre **skyline**Engine. A continuació es poden observar els passos dels quals consta en el diagrama de flux de la Figura 3.1:

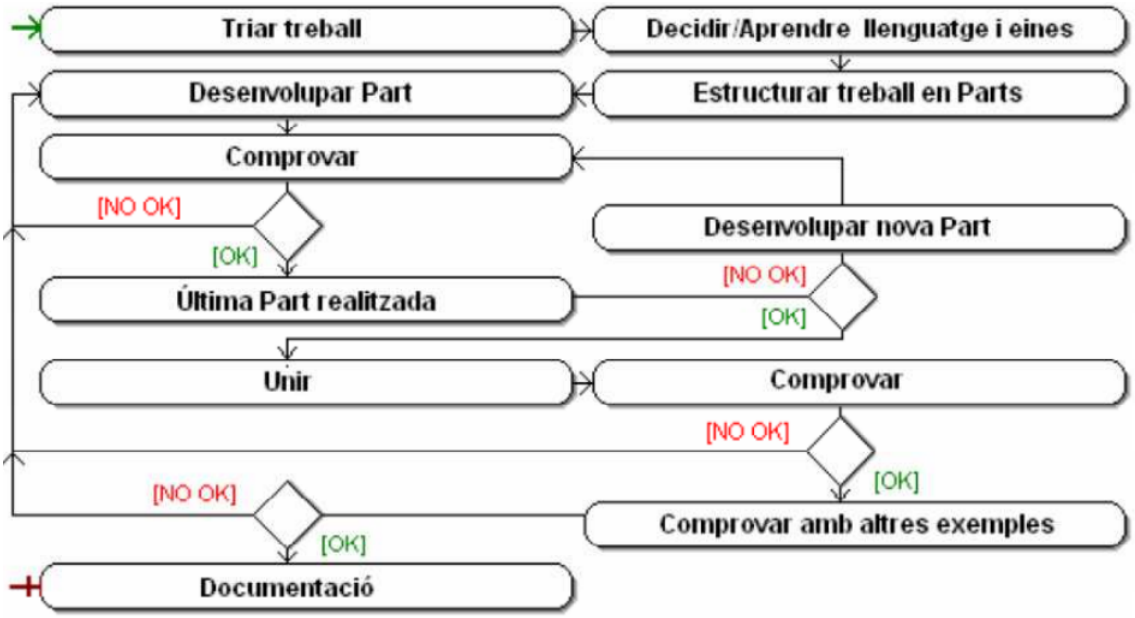


Figura 3.1: Diagrama de flux de la metodologia **skyline**Engine.

1. Triar el treball a desenvolupar.
2. Decidir el llenguatge de programació i les eines a utilitzar.
3. Aprendre el llenguatge de programació i el funcionament de les eines escollides.
4. Estructurar el treball en parts segons les funcions que ha de realitzar.
5. Desenvolupar la part corresponent seguint l'ordre de l'estructura del treball.
6. Fer comprovacions per tal de confirmar que el funcionament és correcte al finalitzar la part.

   6.1. Si al fer les comprovacions el resultat no és l'esperat, es torna al punt 5 per a realitzar els canvis oportuns en la última part desenvolupada o en les anteriors, si és convenient.

   6.2. Si al fer les comprovacions el resultat és l'esperat, es desenvolupa la part següent tornant al punt 5, en cas que s'hagin finalitzat les parts amb les seves respectives comprovacions s'inicia el punt 7.

7. Unir totes les parts desenvolupades i comprovar que el funcionament és correcte.

   7.1. Si al fer les comprovacions el resultat no és l'esperat, es torna al punt 5 per a realitzar els canvis oportuns en l'última part desenvolupada o en les anteriors, si és convenient.

   7.2. Si al fer les comprovacions el resultat és l'esperat s'inicia el punt 8.

8. Generar diferents models d'exemple per a comprovar que el funcionament és el correcte.

   8.1. Si al fer les comprovacions el resultat no és l'esperat, es torna al punt 5 per a realitzar els canvis oportuns en l'última part desenvolupada o en les anteriors, si és convenient.

   8.2. Si al fer les comprovacions el resultat és l'esperat s'inicia el punt 9.

9. Acabar la documentació.

Tal i com es pot veure consisteix en dividir el projecte en mòduls i organitzar en el temps el seu desenvolupament segons el temps de verificació i de correcció. Tant durant el temps de desenvolupament com de verificació, es fa un seguiment mitjançant tutories setmanals o bisetmanals depenent de l'etapa, ja que en els inicis són més lents que no pas les etapes finals i no sempre es necessari fer tutories setmanalment.

Quan s'acaba un mòdul, sempre que es pugui, es finalitza totalment de manera que no es torna a tocar. D'aquesta manera es garanteix que els errors que puguin sorgir en la resta de mòduls són únicament d'aquests i no de cap dels anteriors. I si es dona el cas que ho és, al menys s'han minimitzat al màxim els efectes que s'hagin pogut propagar.

Pel procés de disseny utilitzarem el llenguatge de modelat estàndard dins el camp de l'enginyeria del programari, l'UML (Unified Modeling Language). L'UML s'utilitza per definir un sistema, per detallar els seus elements, per documentar i construir. Per aconseguir això, l'UML disposa de nombrosos tipus de diagrames que mostren diversos aspectes dels elements representats.

Capítol 4

Planificació

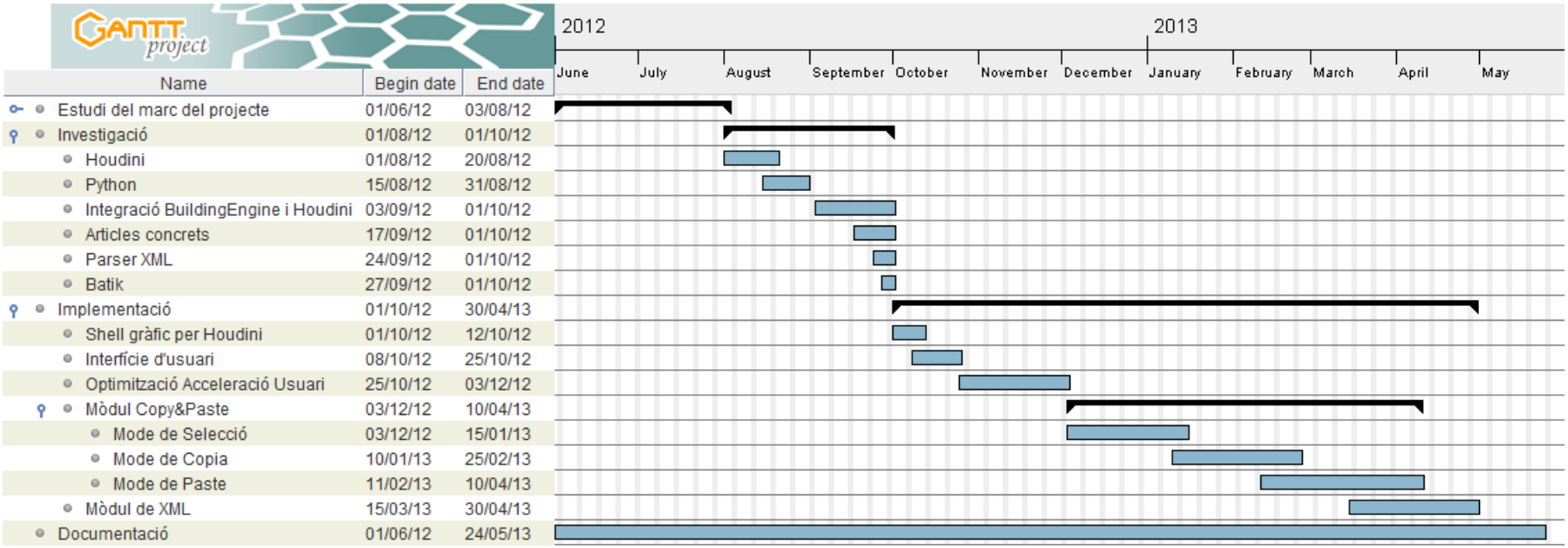
4.1 Planificació

Aquest projecte es va iniciar en el mes de Juny de 2012, els primers mesos es van dedicar a estudiar la proposta de projecte i definir el marc en que s'acotaria. No va ser fins a l'Agost quan es va començar amb l'aprenentatge de Houdini i Python, que fins a les hores eren desconeguts per mi. Un cop dominat el nou llenguatge i la plataforma, es va estudiar la integració de **building**Engine (mòdul de **skyline**Engine) sobre Houdini, per tal de veure les seves funcionalitats i comportament. Aquest enllaç permet el desenvolupament 3D a Houdini d'edificis procedurals, i consta d'un conjunt de llibreries per a fer modificacions sobre grafs generats per Houdini que són la base del projecte. Les classes que intervenen en el pipeline principal de la interacció de l'usuari a l'hora de realitzar una copia i un enganxat de geometria es valen d'aquestes llibreries. Seguidament es van estudiar els articles relacionats amb aquest projecte i es van també buscar eines per a poder implementar el mòdul de XML.

Al mes d'Octubre, es va començar la implementació, on primerament es va fer un estudi de com integrar shells i botons en Houdini per tal d'elaborar el menú que farà servir l'usuari per a fer servir les noves funcionalitats. Més tard, a mitjans de Novembre, es va començar a implementar la part d'optimització en la visualització i interacció de l'usuari. Es va fer d'aquesta manera ja que és un mòdul que no està tant fortament lligat a la plataforma de **building**Engine, com si és el cas del mòdul de "Copy&Paste". Per tant va semblar que era una bona manera de començar la implementació i poder veure els primers resultats d'aquesta fase, per més tard endinsar-se més profundament en el **building**Engine. Un cop acabat el mòdul d'acceleració, es va començar a implementar tot el procés del "Copy&Paste" i els diferents sub-mòduls dels quals consta: Selecció, Còpia i Enganxat.

La documentació ha estat un tasca contínua des del començament del projecte fins a l'últim mes, durant el qual s'ha acabat de formalitzar donant com a resultat l'actual memòria del projecte.

Per veure més clarament la planificació del projecte es pot observar a la Figura 4.1 on es pot veure el diagrama de Gantt de la planificació.



Name	Begin date	End date
Estudi del marc del projecte	01/06/12	03/08/12
Investigació	01/08/12	01/10/12
Houdini	01/08/12	20/08/12
Python	15/08/12	31/08/12
Integració BuildingEngine i Houdini	03/09/12	01/10/12
Articles concrets	17/09/12	01/10/12
Parser XML	24/09/12	01/10/12
Batik	27/09/12	01/10/12
Implementació	01/10/12	30/04/13
Shell gràfic per Houdini	01/10/12	12/10/12
Interfície d'usuari	08/10/12	25/10/12
Optimització Acceleració Usuari	25/10/12	03/12/12
Mòdul Copy&Paste	03/12/12	10/04/13
Mode de Selecció	03/12/12	15/01/13
Mode de Copia	10/01/13	25/02/13
Mode de Paste	11/02/13	10/04/13
Mòdul de XML	15/03/13	30/04/13
Documentació	01/06/12	24/05/13

Figura 4.1: Diagrama de Gantt de la planificació inicial.

Malauradament a finals de la fase d'implementació del mòdul de "Paste", a finals d'Abril, Houdini va començar a presentar símptomes d'inestabilitat a arrel d'una fallada en el codi del nucli del **building**Engine degut a un desfasament del mòdul amb les últimes versions de Houdini. Aquesta fallada junt amb el període d'exàmens del segon semestre va fer prorrogar l'entrega del projecte del Juny al Setembre. Trobar la solució al problema va suposar un elevat cost en temps, i mentre es trobava, es va optar per avançar el mòdul de XML. Finalment es va poder resoldre el problema i es va poder acabar el mòdul de "Paste" i arrodonir la documentació.

A continuació, en la Figura 4.2 es mostra el diagrama de Gantt real on hi consten els canvis en la planificació explicats anteriorment:

Name	Begin date	End date
Estudi del marc del projecte	01/06/12	03/08/12
Investigació	01/08/12	01/10/12
Houdini	01/08/12	20/08/12
Python	15/08/12	31/08/12
Integració BuildingEngine i Houdini	03/09/12	01/10/12
Articles concrets	17/09/12	01/10/12
Parser XML	24/09/12	01/10/12
Batik	27/09/12	01/10/12
Implementació	01/10/12	01/08/13
Shell gràfic per Houdini	01/10/12	12/10/12
Interfície d'usuari	08/10/12	25/10/12
Optimització Acceleració Usuari	25/10/12	03/12/12
Mòdul Copy&Paste	03/12/12	01/08/13
Mode de Selecció	03/12/12	15/01/13
Mode de Copia	10/01/13	25/02/13
Mode de Paste	11/02/13	01/08/13
Mòdul de XML	15/05/13	28/06/13
Documentació	01/06/12	23/08/13

Figura 4.2: Diagrama de Gantt de la planificació final.

Capítol 5

Marc del treball

5.1 Modelatge procedural

Normalment la creació de models 3D i la seva texturització es duu a terme per l'usuari de forma manual a partir de primitives geomètriques, com podrien ser cubs o esferes. Això és una tasca costosa i poc eficient en la majoria dels casos.

Per aquesta raó, en l'àmbit dels gràfics per computador, s'introdueix el concepte de modelatge procedural. Aquest consisteix en un seguit de tècniques que permeten generar models 3D i textures a partir d'un conjunt de regles per tal de suplir aquestes necessitats de manera menys costosa.

Així doncs el modelatge procedural es centra en la generació d'un model 3D basat en un conjunt de regles, enlloc de l'edició del model a través de l'entrada de l'usuari. Aquest conjunt de regles que conformen el modelatge procedural poden estar incorporades de forma parametritzada, o bé poden ser independents del motor d'avaluació. Aquest paradigma de modelatge és utilitzat sobretot quan la feina de crear un model 3D utilitzant modeladors és massa feixuga, o quan es requereixen eines més especialitzades. Aquest sol ser el cas de la generació de plantes i arbres, l'arquitectura o els paisatges. En la Figura 5.1 es pot veure un exemple de modelatge procedural.

Figura 5.1: Exemple de modelatge procedural.

5.2 **skyline**Engine

skylineEngine és una eina de modelatge procedural urbà desenvolupat com a banc de proves per a nous algoritmes i tècniques de modelatge urbà. Aquest sistema presenta algunes característiques noves que el converteixen en un sistema únic, com un paradigma basat en grafs que permet a l'usuari crear tant contingut ric en ciutats amb districtes diferents, carreteres principals i secundàries, els blocs, lots i edificis, com també altres elements urbans com carrers, voreres, parcs, ponts i monuments.

skylineEngine és un sistema desenvolupat al Grup de Geometria i Gràfics de la Universitat de Girona, que està basat de diferents mòduls. Alguns d'ells en codi Python i d'altres en forma de Houdini Digital Assets (HDA's), es pot veure un exemple a la Figura 5.2.

Figura 5.2: Exemple de modelatge procedural mitjançant **skyline**Engine.

5.3 **building**Engine

Per a la creació d'edificis virtuals s'ha utilitzat la llibreria **building**Engine, que és un mòdul de **skyline**Engine. El **building**Engine permet la definició i la construcció procedural d'edificis i altres estructures arquitectòniques. Veure Figura 5.3.

Figura 5.3: Model 3D de la ciutat de Girona generada pel **building**Engine.

A la Figura 5.4 es pot observar un altre exemple dels resultats que pot donar el **building**Engine en en acció.

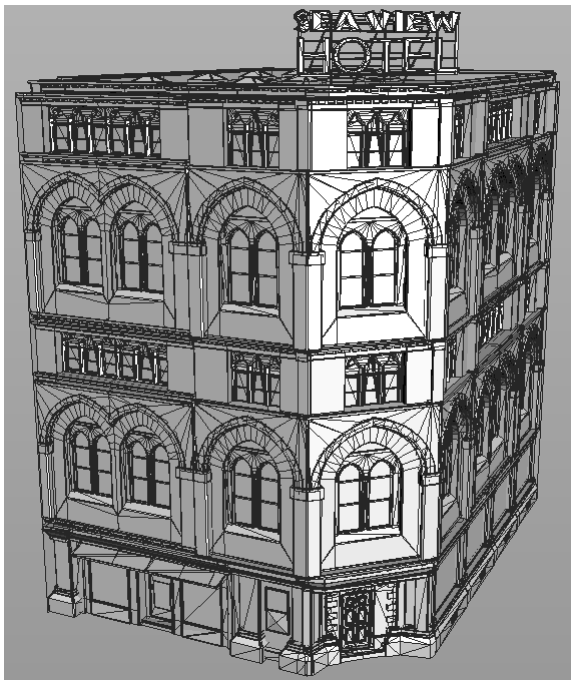


Figura 5.4: Exemple del **building**Engine.

Capítol 6

Requisits del sistema

A continuació es dediquen un seguit de seccions amb la voluntat de definir i explicar clarament la problemàtica que ataca aquest treball, i com s'ha plantejat resoldre-la, així com també explicar els requisits funcionals i no funcionals.

6.1 Plantejament de la problemàtica

Actualment, cada cop que l'usuari vol replicar geometria mitjançant les eines existents, se l'hi planteja un seguit d'accions llargues i tedioses que molts cops condueixen a resultats indesitjats. Moltes plataformes per a la creació de modelats 3D tals com 3DStudio Max, Blender, Maya, Lightwave, Cinema4D, SoftImage, etc, permeten replicar geometria, però ho fan de manera tradicional. Els passos serien els següents (comptant que poden variar en alguns aspectes depenent de la plataforma):

- Primer de tot l'usuari ha de seleccionar la geometria que vol replicar. Cal dir que generalment, no existeix cap mena de jerarquia en la geometria pel que la selecció tendeix a ser una mica artística, ja que amb el ratolí s'ha d'afinar ben bé allò que es vol seleccionar. Molts cops aquestes seleccions no es fan de cop, si no que passen per un procés de selecció primer, i mitjançant combinacions de tecles l'usuari ha d'anar afegint i treien la geometria que l'hi manca o sobra.

- Un cop seleccionada la geometria cal copiar-la, aquest pas sol ser relativament transparent a l'usuari.

- Per enganxar la geometria la majoria de cops no cal seleccionar on, si no que al enganxar-la, aquesta queda "flotant" i l'usuari s'ha d'encarregar de tractar-la de tal manera que la pugui incrustar i/o posicionar, transformar, escalar, allà on l'hi interessa. Aquest procés sol ser poc eficient i els resultats no sempre són els desitjats.

Es pot fer una analogia amb els editors de text. Quan l'usuari vol copiar i enganxar un tros de text, les regles que predominen estan implícites en l'escriptura. És a dir, l'eina de selecció (generalment amb el ratolí) permet acotar exactament allò que es vol, i en cap moment es fa una selecció desordenada o saltejada del text que es vol copiar. Un cop es té seleccionat i copiat, abans d'enganxar-lo s'ha de situar el cursor allà on es vol enganxar. Queda implícit que un cop s'hagi seleccionat el lloc, el text de la part dreta del punt seleccionat s'haurà de desplaçar a la dreta per a fer lloc al text que es vol enganxar. Si el lloc seleccionat per enganxar no és un punt si no una altra selecció d'un tros de text, queda implícit que al enganxar, el text seleccionat s'esborrara i el text que segueixi a la dreta es desplaçarà per fer lloc. Això, a grans trets, poden interpretar-se com a regles implícites en la edició de text.

En el cas de la construcció d'edificis procedurals es necessiten també unes regles. El **building**Engine permet crear els edificis mitjançant regles que poden ser interpretades com d'un graf d'operacions. Amb aquest paradigma, si es vol realitzar la replicació de geometria amb una eina de "Copy&Paste", aquesta eina cal que pugui permetre seleccionar inequívocament allò que l'usuari vol replicar. Aquesta acotació és possible gràcies a la jerarquia de geometria en diferents primitives. Un cop seleccionat allò que l'usuari vol replicar, ho ha de copiar. Aquesta operació és una selecció del subconjunt de regles que fan possible la generació de geometria seleccionada. Finalment quan l'usuari desitja enganxar la geometria, cal que seleccioni allà on ho vol fer i finalment enganxar el subconjunt de regles a la part que s'ha seleccionat. Aquest últim pas però, en l'analogia de l'editor de text, és molt senzilla de solucionar. Però en el cas de geometria presenta un seguit de deficiències i problemàtiques molt més complexes, les quals s'expliquen extensament en la publicació de Barroso et al. [3], on es proposa una solució mitjançant excepcions i s'exemplifiquen diversos resultats així com problemàtiques que encara s'han de resoldre.

Finalment l'usuari ha de tenir una interfície gràfica mitjançant la qual pugui realitzar la tasca definida anteriorment de manera intuïtiva i usable.

Quan es replica geometria en una escena 3D, es fa més densa en polígons i això fa que el nivell d'interactivitat descendeixi ja que la tarja gràfica ha de realitzar un major nombre de càlculs. Per a solucionar això, s'ha de dotar d'alguna eina a l'usuari que permeti realitzar operacions amb una major velocitat d'interacció.

6.2 Plantejament de la solució

Seguint la proposta de Barroso et al. [3], la solució que es presenta és la d'implementar un sistema de selecció tal que l'usuari pugui marcar, amb el ratolí, la geometria que vol. De manera automàtica, l'eina ha de cercar aquell subconjunt de regles que generen la geometria mitjançant l'arbre de primitives. En concret, la primitiva que es seleccionarà és aquella que sigui ancestre comuna a totes les primitives seleccionades. Per a trobar aquesta primitiva s'ha de recorre l'arbre de primitives fins que es trobi aquella regla que generi la geometria seleccionada

Un cop es sap quin és la primitiva ancestre comuna a les seleccionades, s'ha de seleccionar totes aquelles regles que són descendents d'aquesta primitiva. És aquest sub-arbre de regles el que s'acaba seleccionant finalment i el que fa que la selecció sigui neta i coherent. Per tant s'obtindrà tota aquella geometria que sigui el resultat del sub-arbre de regles i es realitzarà la còpia sobre aquest conjunt.

Arribat aquest punt l'usuari podrà, si vol, desar aquesta selecció en format XML per a una futura reutilització. Per tant, abans d'haver efectuat la copia, l'usuari haurà de facilitar la ruta en la que vol desar el fitxer XML amb la geometria seleccionada. Com és lògic, també es permetrà carregar aquests fitxers XML amb continguts d'altres còpies per tal de fer-los servir. La possibilitat de bolcar tota una selecció en un fitxer XML, obre tot un plegat de possibilitats que poden anar des del simple re-aprofitament de còpies fetes anteriorment, fins a la possibilitat de poder crear un estàndard que permeti la compartició d'art entre modeladors 3D, de manera que a més de la pròpia còpia es puguin definir algunes etiquetes segons la part de la façana que l'usuari hagi seleccionat per a la seva posterior replicació i adaptació. Això però, escapa al marc d'aquest projecte i no és l'objectiu principal d'aquest projecte.

Per a poder enganxar la selecció copiada, primer l'usuari ha d'explicitar en quin lloc vol fer-ho. El mètode per poder dir on vol enganxar la geometria copiada és el mateix que el

de seleccionar la geometria. L'algorisme és el mateix i el procediment de cercar la primitiva ancestre comuna que sustenta el sub-arbre de regles és el mateix.

Finalment l'usuari pot realitzar l'enganxat. Aquesta tasca és una de les més complexes del procés ja que s'han d'afegir un conjunt de nodes que creen noves regles. Aquestes regles són un conjunt d'excepcions i etiquetats que fan possible integrar la geometria copiada en el nou edifici. Com es tracta de regles, es poden fer adaptacions d'escales i posicionaments per a que el resultat sigui coherent i net. Aquest procés es pot veure més detalladament en el següent capítol (Capítol 8)

D'aquesta manera es dota d'una eina a la plataforma de **building**Engine que permet fer un proces de "Copy&Paste" procedural que fins ara no era possible, i que cap altre plataforma resol de manera transparent amb resultats nets i coherents.

6.3 Requisits funcionals

Els requeriments funcionals expliquen què ha de fer l'aplicació, és a dir, totes les funcionalitats que tindrà sense especificar com es farà.

Per a poder desenvolupar aquesta nova eina s'ha hagut d'especificar de bon principi i de forma clara, què ha de poder fer. Aquestes són les funcionalitats que ha de tenir:

- L'usuari no té perquè conèixer aspectes tècnics per a poder editar i efectuar canvis a l'edifici.
  - No ha preocupar-se pels canvis en el graf de nodes
  - No cal que conegui quines primitives contenen la geometria que vol copiar
  - Ni tampoc ha de saber com navegar per l'arbre de primitives equivalent per trobar els nodes que generen aquestes primitives.
- L'eina ha de ser intuïtiva i ha de permetre treballar de manera còmode.
  - L'usuari pot seleccionar diferents trossos de geometria quan manté el botó "shift" premut mentre selecciona el que vol amb el botó esquerra del ratolí.
  - Si és vol seleccionar un tros de geometria extens, en comptes de seleccionar com en el punt anterior, s'ha de permetre que l'usuari pugui crear un rectangle de selecció i es processi tot allò que el rectangle cobreixi.

6.4 Requisits no funcionals

En aquest apartat es defineix com ha de ser l'aplicació si no què és el que ha de fer. Aquests requisits fan referència al que s'ha de tenir en compte a l'hora de dissenyar el sistema, com per exemple els recursos necessaris.

Els requisits no funcionals són els següents:

- La nova eina ha de ser eficient de manera que s'optimitzin al màxim els càlculs necessaris per a dur a terme la tasca que l'hi pertany, ja que en molts casos el nombre de nodes pot ser elevat i també el de primitives.
  - A més de l'elevada quantitat d'informació que s'ha de tractar, s'ha de tenir en compte que no tots els usuaris disposen d'un equip potent d'última generació, com més eficient sigui més públic podrà fer servir l'eina.

- Aquesta eina ha de poder fer-se servir en diferents plataformes i sistemes operatius. Com a mínim en tants com es pugui fer servir el Houdini.
- Com fa ús del teclat i el ratolí, es farà servir els botons més bàsics d'aquests perifèrics per tal que no s'hagi de disposar de hardware especialitzat per a dur a terme la tasca.

La plataforma on s'executarà l'eina serà mitjançant Python sobre Houdini, per tant, els requeriments de hardware són els requeriments mínims que demana SideFX:

Títol	Descripció
Sistema operatiu:	Linux, Windows o MacOS.
Memòria Ram:	4 Gb
Processador:	AMD o Intel d'arquitectures tant x32 com x64
Memòria HDD:	1Gb lliure en el disc dur
Perifèrics:	Ratolí de 3 botons

Capítol 7

Estudis i decisions

7.1 Houdini

El Houdini Master és una eina per a la creació avançada d'efectes Visuals 3D, que permet controlar l'animació des de la mateixa interfície gràfica o fent servir un del dos llenguatges de script que incorpora. El primer d'ells, i més antic, és l'anomenat HSript. Aquest llenguatges és exclusiu de Houdini i existent des de les primeres versions. El segon és el Python, introduït a partir de la versió 9.0 amb l'objectiu d'estandarditzar el funcionament i dotar el programa d'un codi conegut. Per a la realització del projecte s'ha utilitzat la versió 12.0.572 de Houdini tant per Windows, com per MacOS i Linux Debian ja que són la base on funciona Houdini i per tant on ha de funcionar també el projecte.

Houdini cobreix totes les principals regions de producció en 3D incloent:

- Modelatge: totes les entitats de geometria estàndard, incloent polígons.
- Animació: animació de fotogrames, manipulació del canal i captures de moviment.
- Partícules
- Dynamics: Dinàmica de cossos rígids, de fluids computacionals, corbes dinàmiques...
- Il·luminació
- Rendering
- Volumètrica: generació, població, d'escalars i vectors.
- Composició: compositor complet de punt flotant en profunditat d'imatges (capes)
- Plugins per al desenvolupament: biblioteques per a la extensibilitat de l'usuari.

El Houdini treballa usant un sistema de nodes en forma de graf acíclic per a representar els objectes de l'escena. D'aquesta manera s'obté un sistema d'objectes independents, units entre ells per un flux de dades.

La Figura 7.1 mostra l'entorn de treball del Houdini. Aquest entorn està compost per una barra d'eines (superior), des d'on es generen totes les figures i accions. La zona de treball (marcat en vermell) és l'espai on es poden visualitzar els resultats usant diferents vistes i es poden seleccionar i modificar objectes. El graf de nodes (marcat en blau) és l'espai on es representen tots els objectes de l'escena en forma de caixetes independents, enllaçades les unes

a les altres.

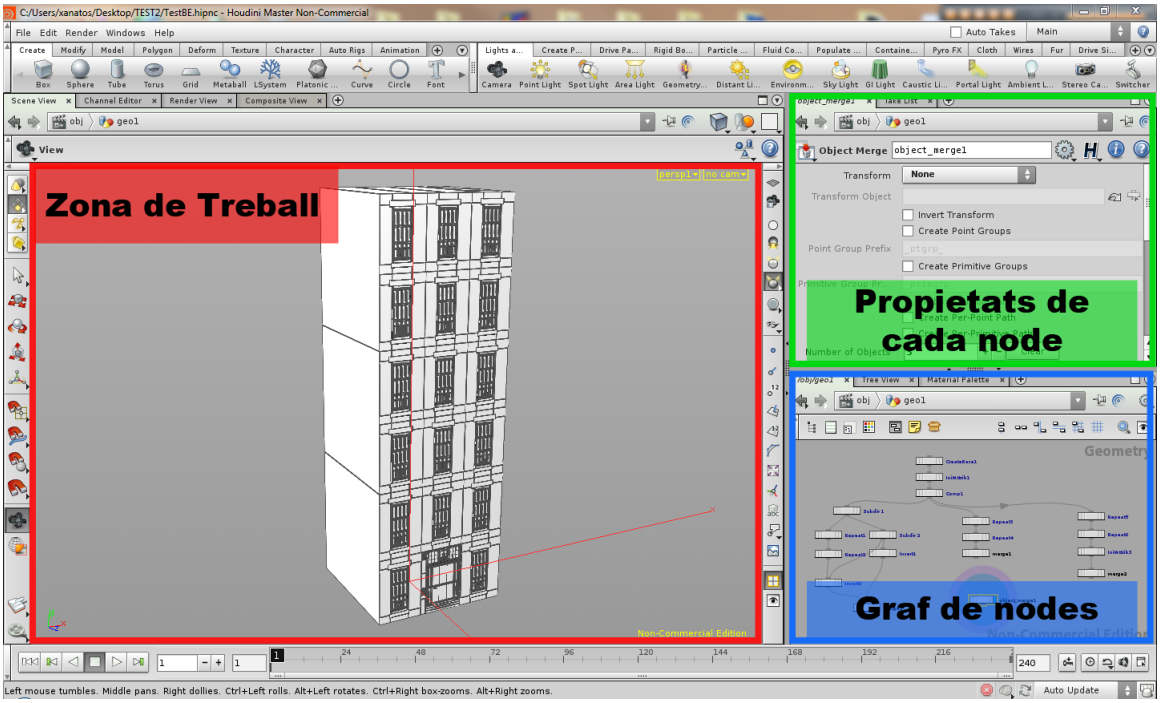


Figura 7.1: L'entorn de treball de Houdini

Finalment, un cop seleccionat un node, es pot consultar i modificar les seves propietats a través del sistema de pestanyes del quadre de propietats (marcat en verd) que permet efectuar aquestes tasques mitjançant la interfície gràfica.

El Houdini divideix les xarxes de nodes en diferents categories:

- **VEX Builder (VOP)** – Xarxes per al llenguatge del Houdini anomenat VEX.
- **Objects (OBJ)** – Xarxes d'objectes (geometria, llum, càmera, etc).
- **Geometry (SOP)** – Xarxes per construir geometries.
- **Particle (POP)** – Xarxes per efectes de partícules.
- **Compositing (COP2)** – Xarxes per composició 2D.
- **Dynamic (DOP)** – Xarxes per a efectes dinàmics.
- **Channel (CHOP)** – Xarxes per a operacions de canals. Els canals controlen com canvien els valors a través del temps.
- **Shaders (SHOP)** – Xarxes per a crear diferents shaders.
- **Output (ROP)** – Xarxes per especificar les opcions de renderització.

7.1.1 Nodes de Houdini

Per crear qualsevol node Houdini, cal prémer la tecla tabulador amb el cursor a la zona d'arbre de nodes. Seguidament apareix una finestra amb tots els nodes disponibles organitzats per famílies. Si se'n busca un de concret i se'n coneix el nom, es pot teclejar i el programa proposa el node. A la Figura 7.2 es pot veure el llistat de tots els nodes per defecte de Houdini i el cercador per a posar-hi el nom del node que es vol buscar.

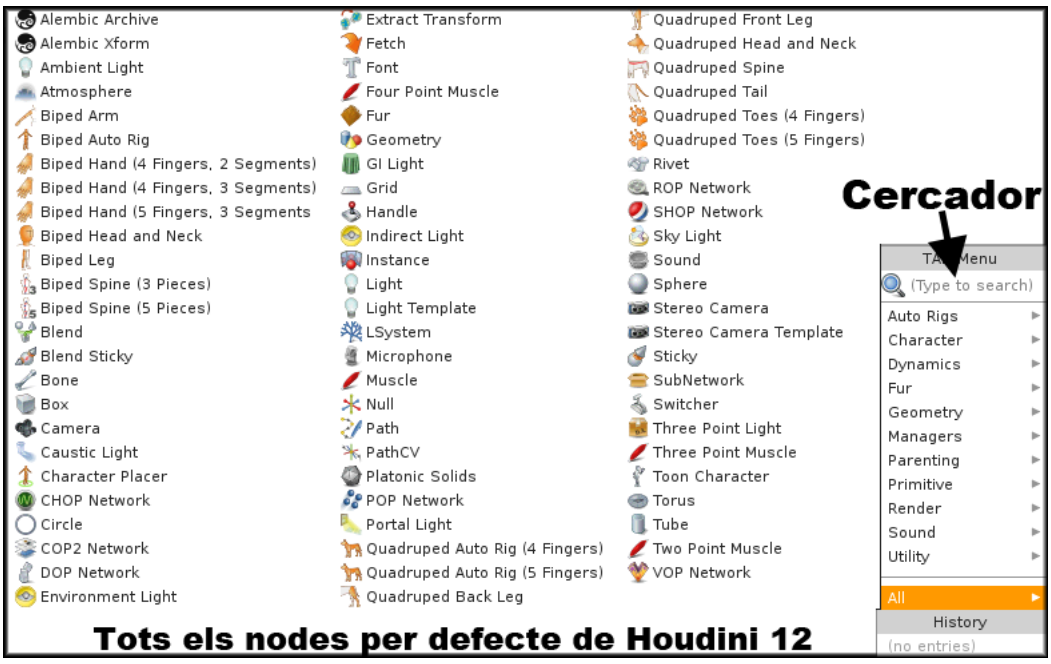


Figura 7.2: Llistat dels nodes mitjançant la GUI de Houdini.

7.1.2 Graf de nodes

Com s'ha vist en l'apartat anterior, en l'entorn de treball de Houdini s'hi troba el graf de nodes, en el qual es veu com s'enllacen i formen un graf del modelat.

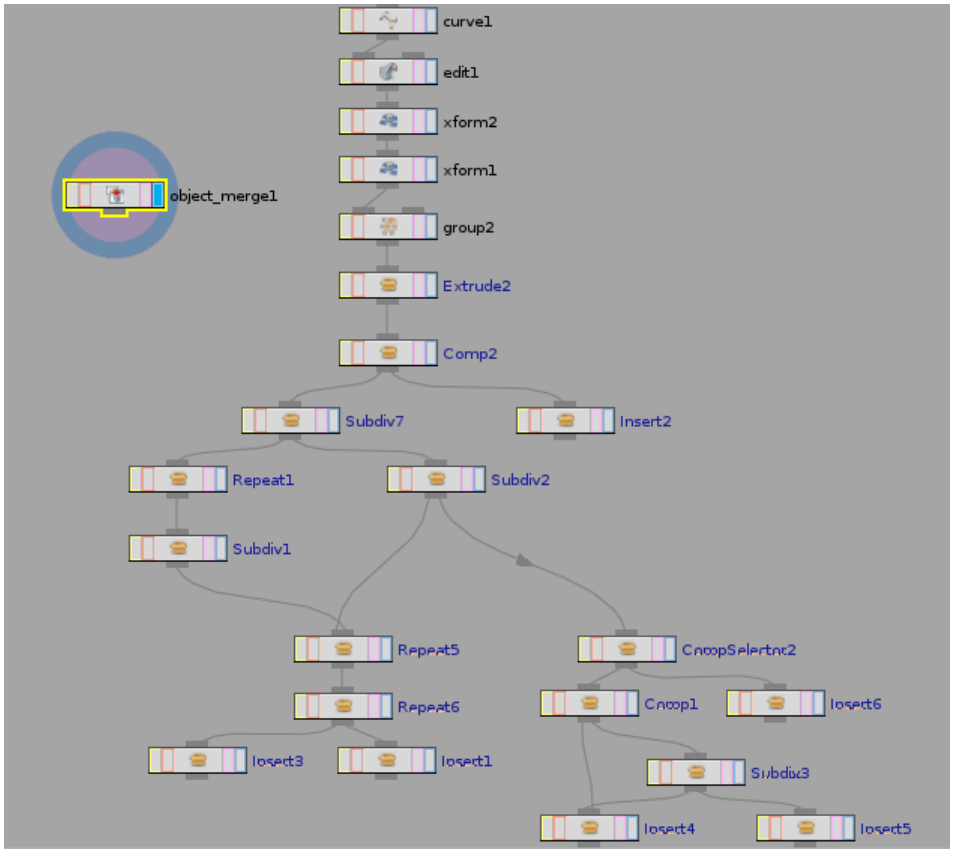


Figura 7.3: Exemple de graf de nodes de Houdini i **building**Engine.

Com es pot observar en la Figura 9.8 hi ha un seguit de nodes amb una tipologia concreta

que estan enllaçats per formar un graf. Concretament a la Figura 9.8 es poden observar tipus de nodes propis del **building**Engine tals com els que s'explicaran a continuació, i d'altres que son propis de Houdini, com ho són el *object_merge*, *curve*, *transform*, *group*, *extrude*, etc.

7.1.3 Tipus de nodes a Houdini

Com s'ha vist, Houdini classifica els nodes per contenidors de xarxes de nodes, tot i això, per cada xarxa de nodes té molts tipus de nodes per a crear diferents objectes, efectes, modificar o simplement realitzar funcions.

Aquest projecte centra l'atenció en la xarxa de nodes "Objects" i a continuació explicarem els diferents tipus de nodes que conté:

- **Auto Rigs:** nodes per afegir articulacions als models.
- **Character:** nodes per desenvolupar personatges.
- **Dynamics:** nodes per desenvolupar animacions dinàmiques com fluids o roba.
- **Fur:** nodes per afegir pell als personatges.
- **Geometry:** nodes per desenvolupar geometria.
- **Managers:** nodes per tractar els Digital assets i tenir control sobre les rutes d'aquests.
- **Parenting:** nodes per enllaçar altres nodes com ara Blend.
- **Primitive:** nodes per crear primitives com ara caixes o esferes.
- **Render:** nodes per fer diferents renders del que hem modelat com ara llums, càmeres, etc.
- **Sound:** nodes per afegir sons a l'entorn.
- **Utility:** nodes per fer funcionalitats útils a l'entorn.

Com es pot veure en la Figura 7.4 des de l'interior d'un objecte "Object" es poden crear els tipus esmentats anteriorment.

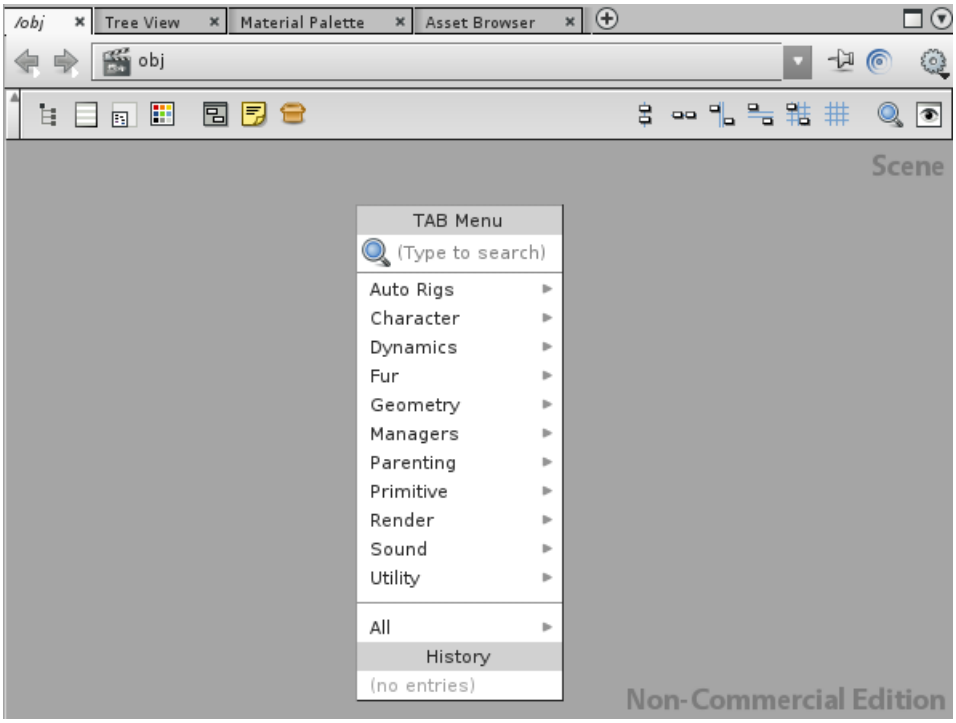


Figura 7.4: Tipus de contenidors dins de ”Object”.

Dels tipus esmentats anteriorment, el que té més interès per aquest treball és el tipus “Geometry”, ja que s'utilitza en el **building**Engine com a base del contingut de tot l'edifici.

A continuació es poden veure les diferents categories en les que es classifiquen els nodes “Geometry”. Els nodes de Houdini de la xarxa de nodes “Geometry” es classifiquen en divuit categories de les quals cadascuna té un número determinat de nodes:

Atribute	Hair	Particle
Character	Import	Polygon
Digital Assets	Managers	Primitive
Edge	Manipulate	Utility
Export	Material	VDB
Fluid	NURBS	Volume

Per veure cadascun dels nodes dinds de les divuit categories veure l'apartat 1 de l'annex.

7.1.4 Propietats dels nodes

Hi ha molts tipus de nodes a Houdini i cadascun té unes propietats intrínseques que es poden consultar i modificar, tant des de l'entorn de Houdini com mitjançant funcions del mòdul **hou** (que s'explicarà més endavant) mitjançant Python. En la Figura 7.5 es pot observar una captura de l'entorn de Houdini on es poden visualitzar i modificar com s'ha explicat, les propietats d'un node de Houdini (cas particular del tipus transform).

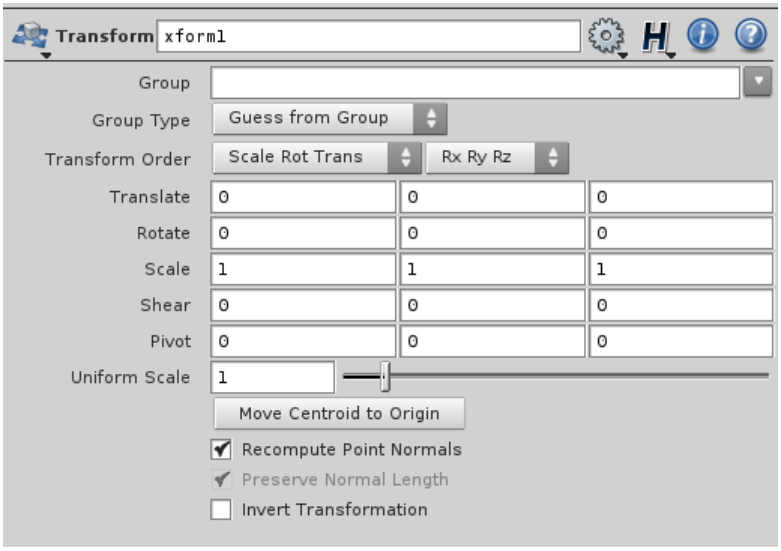


Figura 7.5: Exemple de Node.

7.1.5 Paràmetres dels nodes a Houdini

Cadascun dels nodes que s'han explicat anteriorment, tenen un seguit de paràmetres que en l'apartat anterior s'esmentaven com a propietats. El motiu pel qual es canvia el nom de "paràmetres" a "propietats" és la manera d'accedir a la informació del node. Quan s'accedeix a la informació a través de l'entorn gràfic de Houdini a les hores es parla de propietats del node. Quan s'accedeix via codi Python executat des de el Shell, a les hores s'anomenen paràmetres del node. Tot i que la informació sigui la mateixa fent servir un mètode o un altre, és evident que per a realitzar aquest treball és d'interès accedir via codi Python per tal de poder assolir els objectius plantejats.

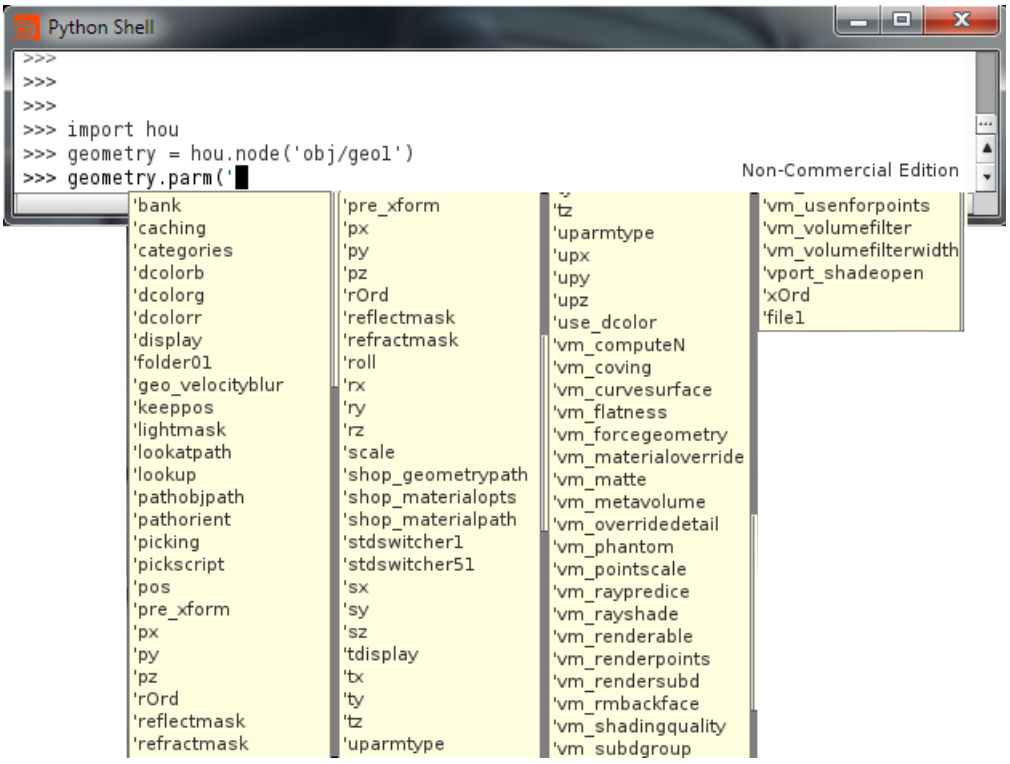


Figura 7.6: Exemple dels paràmetres que conté el node 'Geometry'.

Com es pot veure en la Figura 7.6, el node "Geometry" té un seguit de paràmetres que poden ser obtinguts mitjançant les següents operacions del mòdul hou:

- **Node.parm('nom_paràmetre'):** per obtenir el paràmetre o propietat en qüestió.
- **Node.setParms()/Node.parm('parametre').set(valor):** per modificar el paràmetre o propietat en qüestió. (Es pot fer servir per emplenar un diccionari de paràmetres i passar-los de cop)
- Una altra opció pot ser **Node.parm('nom_paràmetre').set(valor)** per modificar el paràmetre en qüestió. (Per modificar un sol paràmetre concret)

7.1.6 Nodes buildingEngine

En aquest apartat s'expliquen els nodes que pertanyen al **building**Engine, que són els que s'han hagut de tenir en compte a l'hora de desenvolupar el projecte.
Primer s'ha de considerar que la geometria a **building**Engine es descriu mitjançant etiquetes semàntiques. Un exemple seria l'etiqueta 'facade', que representa la part de la façana, també podria ser un altre exemple 'door', que representaria una porta.
Els nodes del **building**Engine generen els edificis a través d'aquestes etiquetes utilitzant dos atributs: 'filter' i 'product', que es poden trobar en el panell de propietats dels nodes. L'atribut 'filter' indica la geometria que processarà el node, mentre que el 'product' indica l'etiqueta de la geometria que es generarà com a sortida resultant del node.
A la Figura 7.7 es pot observar com queda representat de manera gràfica un node de **building**Engine a dins de Houdini.

Figura 7.7: Exemple de Node.

7.1.6.1 Node CreateBase

Aquest node crea un model de volum inicial de sis cares rectangulars que representa l'edifici. Aquest node és molt semblant al node tradicional *Box* de Houdini. En realitat, internament es fa servir el *Box* de Houdini per a generar el model.
A la Figura 7.8 es pot observar un exemple d'aquest node i observar com afecten les propietats del node al resultat 3D i com es representa en el graf de nodes de Houdini.

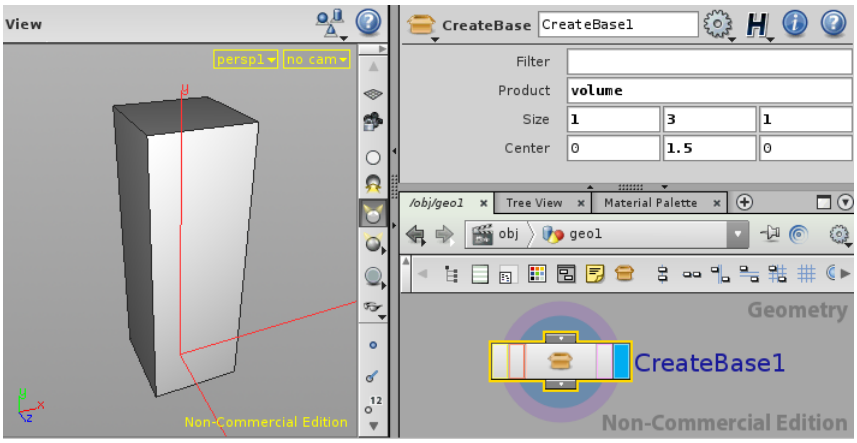


Figura 7.8: Exemple del node CreateBase.

7.1.6.2 Node Comp

S'encarrega de dividir un model de volum per les components. Aquest node classifica la geometria entrant d'acord amb els vectors normals dels polígons, com podria ser la façana, el sostre i les altres cares.
La Figura 7.9 il·lustra un exemple d'aquest node aplicat sobre la Figura 7.8 de l'apartat anterior. En aquest exemple, el bloc creat es classifica en 3 components.

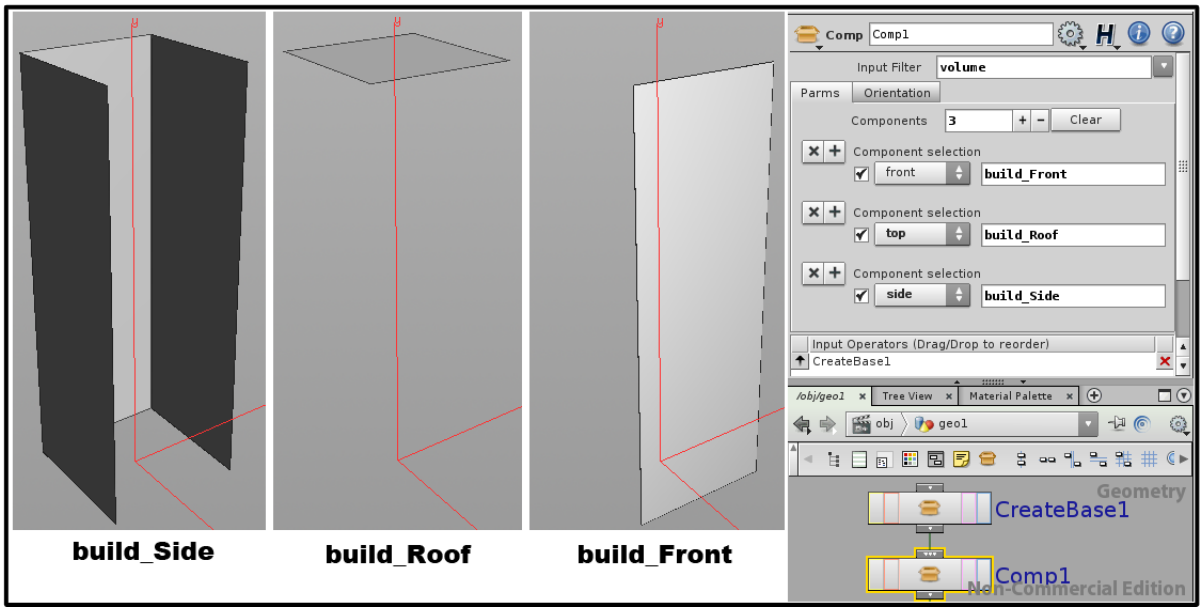


Figura 7.9: Exemple del node Comp.

7.1.6.3 Node Subdiv

Realitza una subdivisió de l'element que rep en elements més petits. Té tres modes de funcionament segons cada un dels eixos de coordenades, X, Y i Z, que permet fer divisions en qualsevol d'elles.
A la Figura 7.10 es pot veure, a la part de l'esquerra, com funciona el node en el mode Y; en aquest exemple s'ha dividit la façana de la Figura 7.9 de l'apartat anterior en la planta baixa, i la resta de plantes. Al costat dret es pot tornar a veure el funcionament del node Subdiv, però aquest cop en el mode X, on s'ha dividit la planta baixa de la imatge de l'esquerra en tres seccions més.

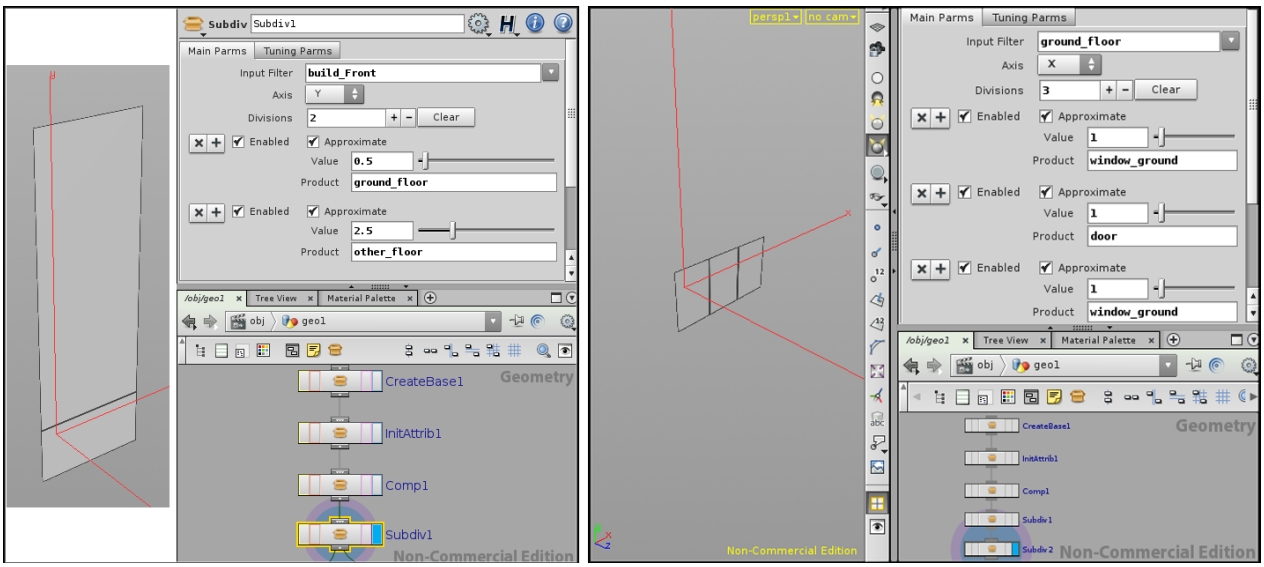


Figura 7.10: Exemple del node Subdiv en mode Y a l'esquerra, i en mode X a la dreta.

7.1.6.4 Node Repeat

Permet realitzar repeticions de divisions de l'element d'entrada en elements més petits. De la mateixa manera que el node Subdiv, té tres modes de funcionament en X, Y i Z.
A la Figura 7.11 es pot veure com funciona el mode en Y. En aquest exemple s'ha dividit la part superior de la façana de la Figura 7.9 de l'apartat anterior en plantes.

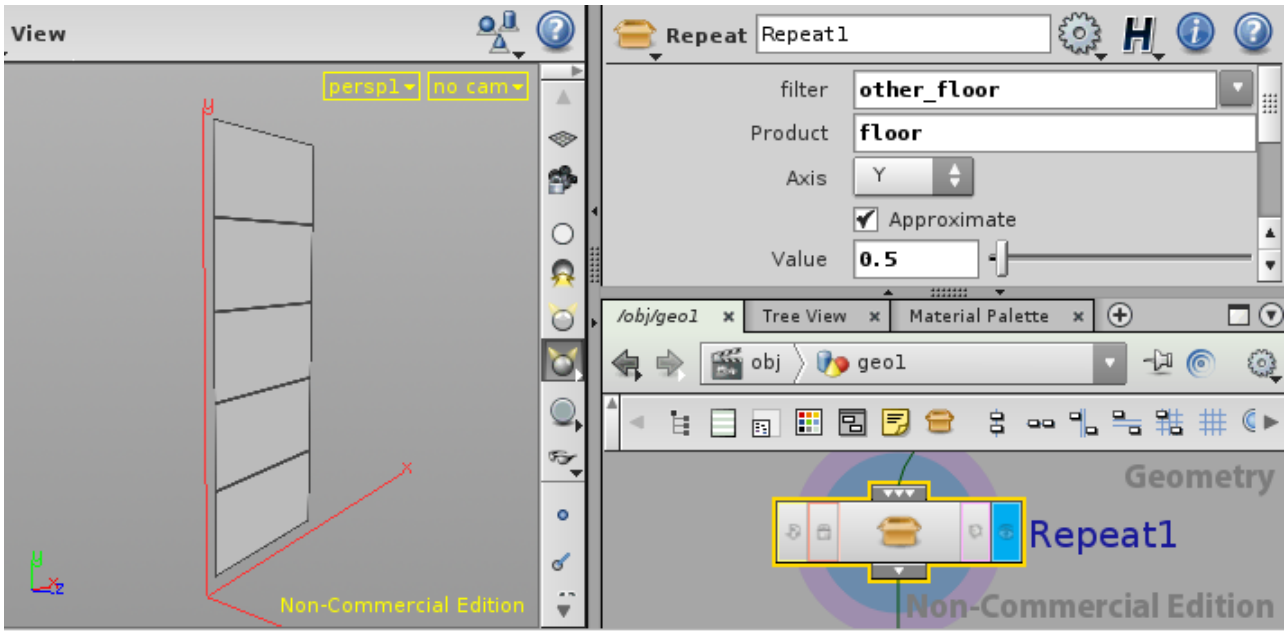


Figura 7.11: Exemple de funcionament del node Repeat en mode Y.

A la Figura 7.12 es pot veure com funciona el node en el mode Z. En aquest exemple s'ha dividit cadascuna de les plantes generades pel Repeat de la Figura 7.11 en tres parts.

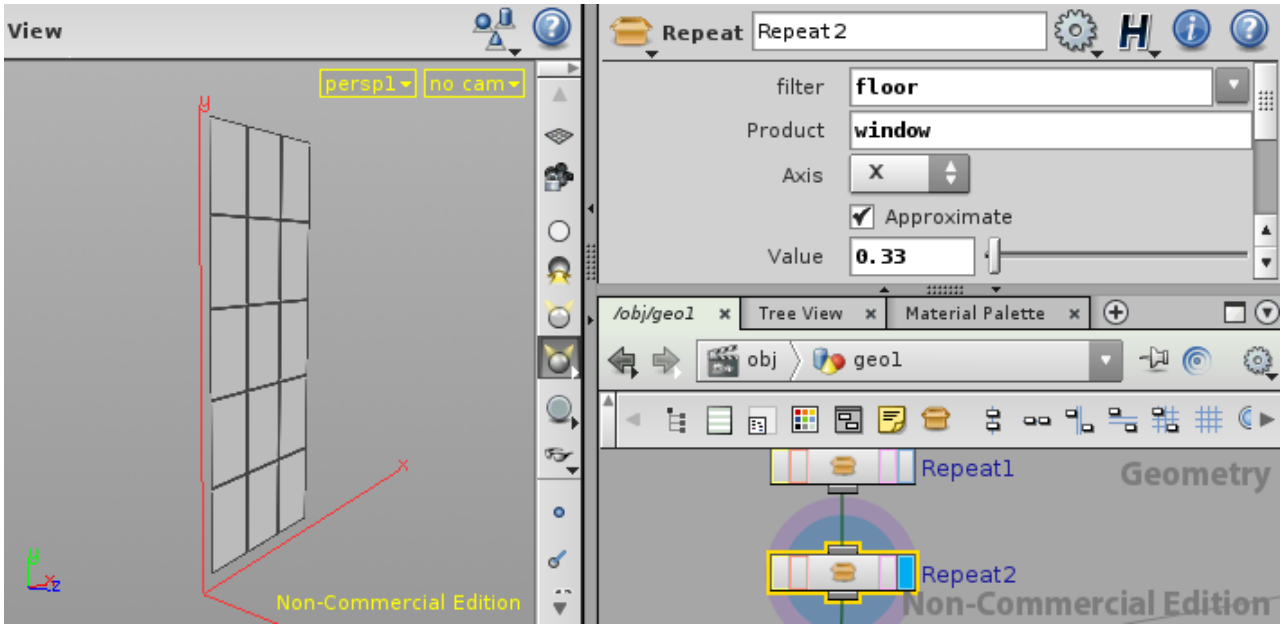


Figura 7.12: Exemple de funcionament del node Repeat en mode Y.

7.1.6.5 Node Insert

Insereix una nova geometria a l'espai definit per l'element base que rep. Aquesta geometria nova s'anomenarà *asset* en el context d'aquest projecte. Un asset és un model creat per l'artista o el modelador que l'ha encapsulat de tal manera que es pot treballar amb ell de manera planera. En alguns casos, l'artista pot permetre modificar alguns paràmetres de forma procedural, però no es farà servir en el marc d'aquest projecte.
De manera més intuïtiva espot observar un asset en la Figura 7.13, un d'una d'una finestra.

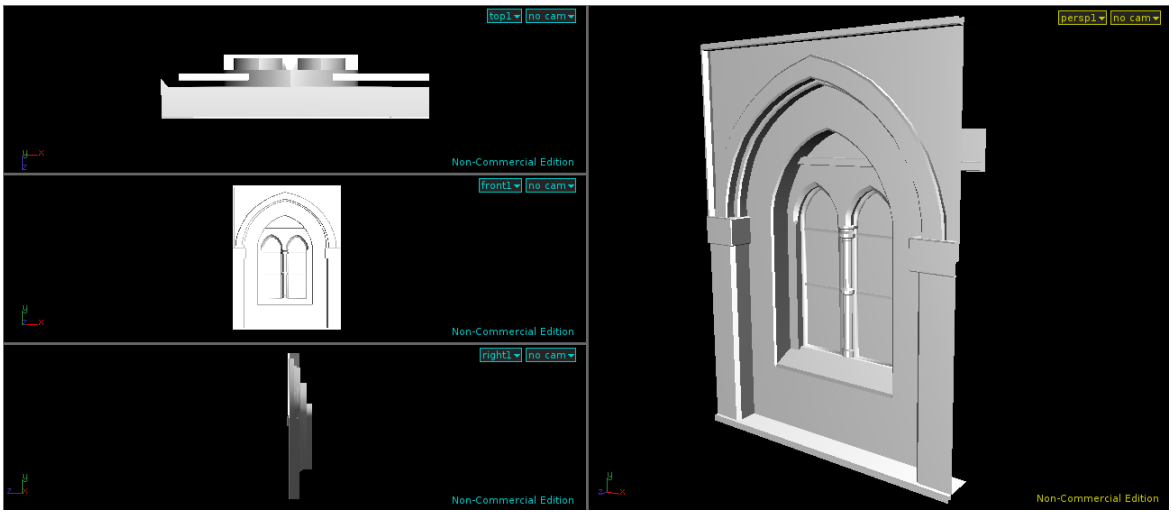


Figura 7.13: Exemple de funcionament dels assets.

El concepte d'asset és propi dels nodes de tipus 'Insert'. Aquests nodes permeten inserir geometria carregant-la d'un fitxer extern de manera que només cal introduir la ruta d'aquest fitxer i el node, mitjançant el seu paràmetre 'filter' descrit anteriorment, és capaç de reproduir la geometria d'aquest fitxer en la primitiva indicada segons el paràmetre 'filter' i els seus inputs. En la Figura 7.14 es pot observar els paràmetres d'un node de tipus 'Insert' que carrega un fitxer anomenat *TwoWindow.obj*.

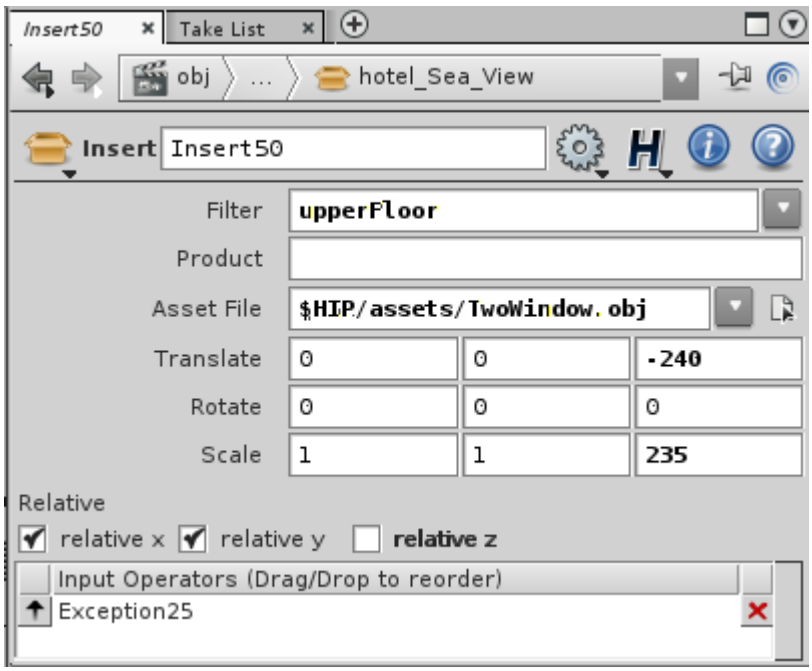


Figura 7.14: Exemple d'Inputs i Outputs de un node.

En la Figura 7.15 es pot veure com el node Insert treballa sobre el que rep del node Subdiv de la Figura 7.10, afegint-hi la porta "door" mitjançant un asset. Aquest es un exemple on es poden observar tant els paràmetres del node 'Insert' com el resultat en la escena.

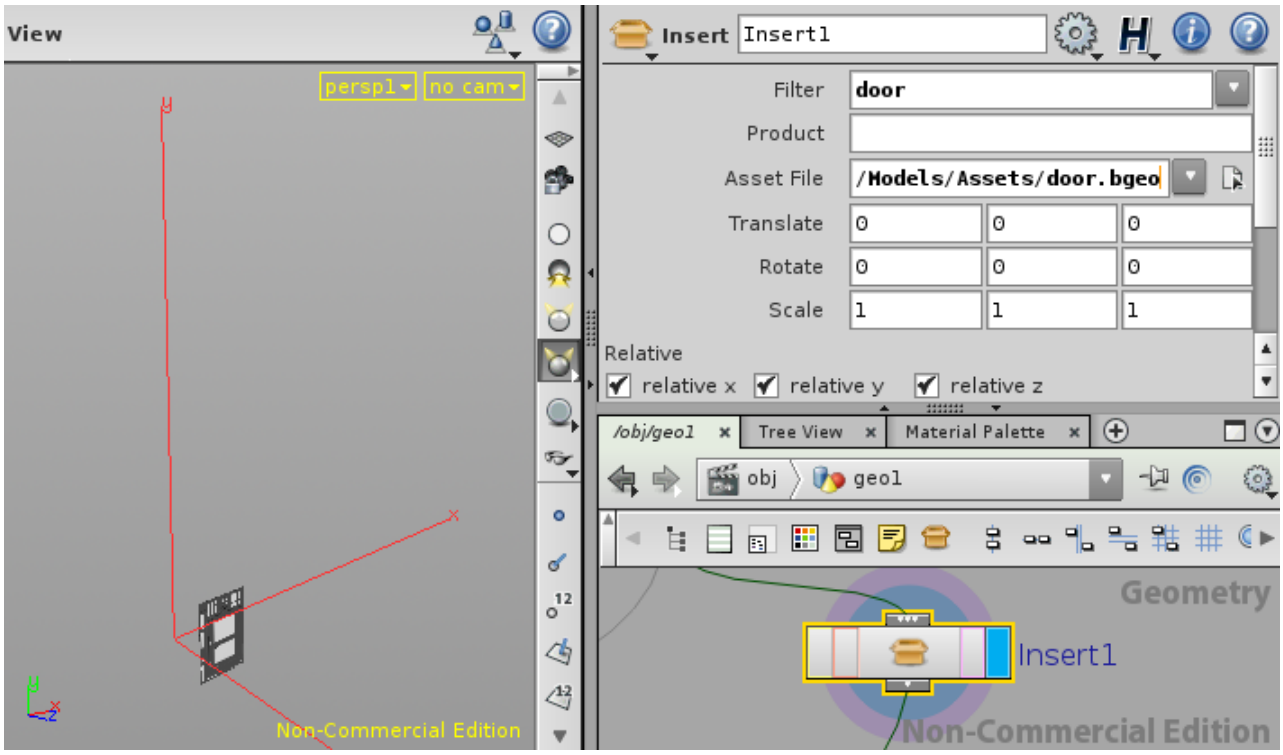


Figura 7.15: Exemple de funcionament del node Insert amb una porta.

De la mateixa manera, en la Figura 7.16 es pot veure com actua el node Insert sobre el node de la Figura 7.12, inserint l'Asset d'una finestra mitjançant els TAG(etiqueta) 'window' i 'window_ground'.

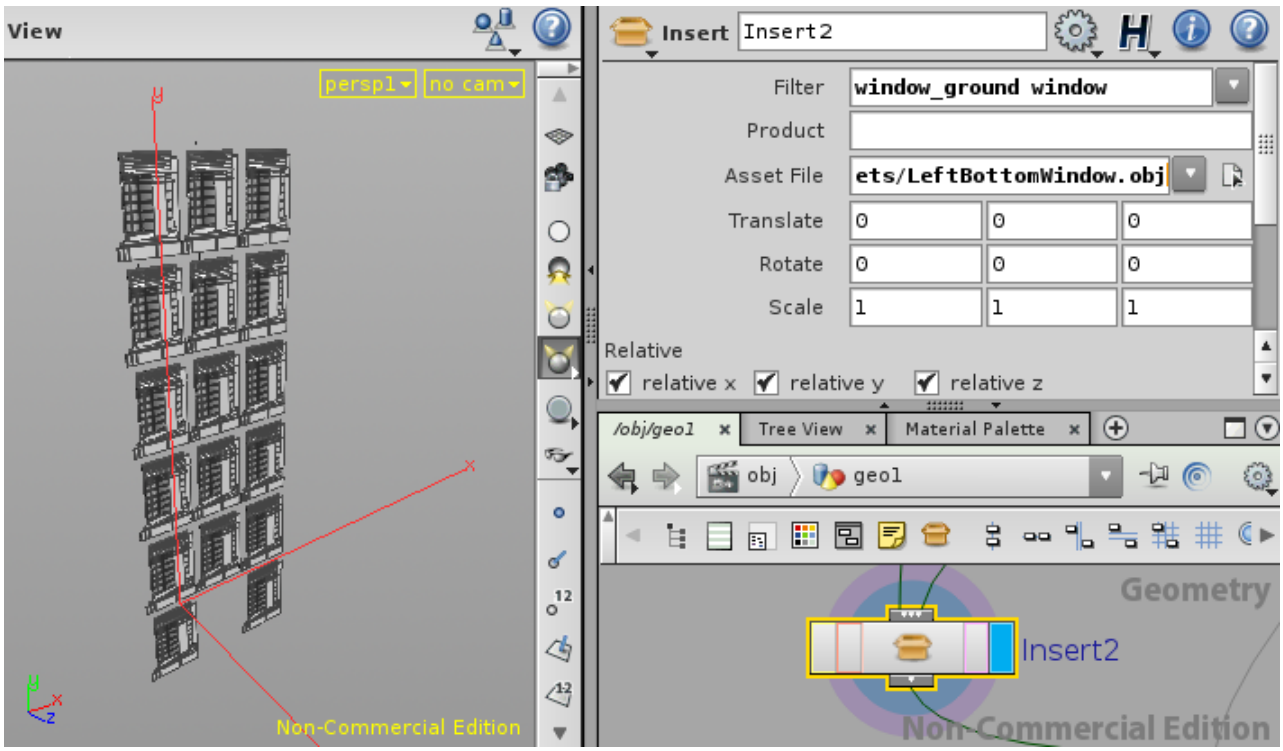


Figura 7.16: Exemple de funcionament del node Insert amb les finestres.

Es pot observar com, mitjançant el filtrat dels nodes, és possible aplicar una operació a un conjunt sense necessitat d'explicitar-ho primitiva a primitiva, sino fent servir les seves etiquetes.

7.1.7 Resum d'un edifici

Després de fer totes aquestes operacions amb els nodes, es poden ajuntar els nodes finals per poder veure el resultat final de l'edifici al complet (veure Figura 7.17).

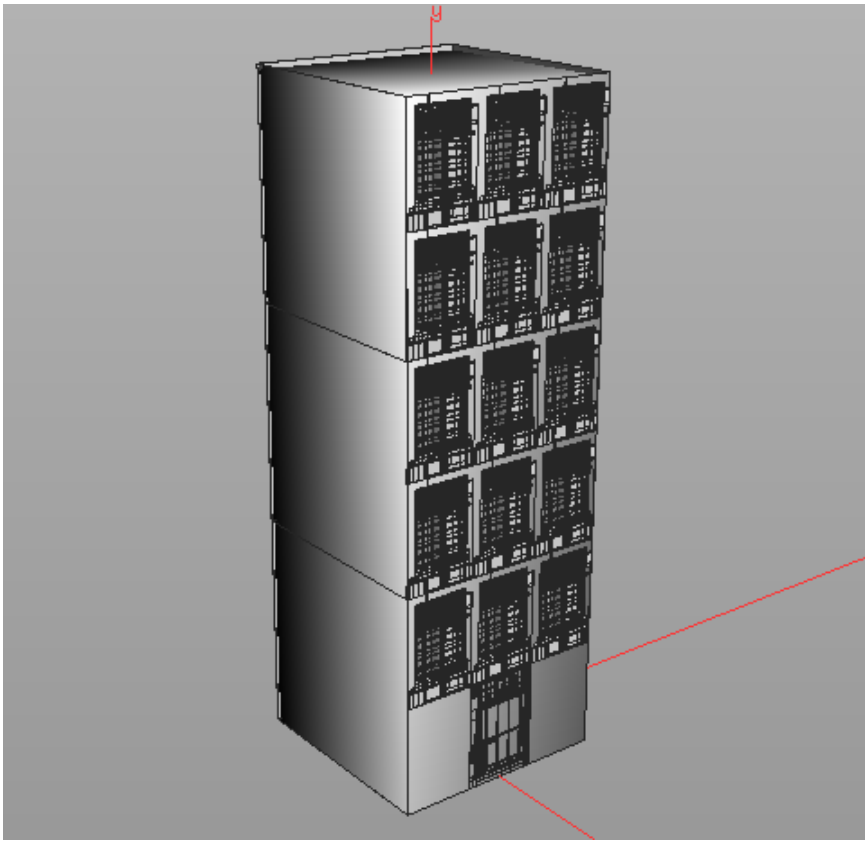


Figura 7.17: Exemple de visualització del resultat d'un edifici complet.

Per poder fer aquesta visualització, s'ha utilitzat el node *object_merge* i s'ha mostrat els nodes *Insert* i el node *Comp*. D'aquesta manera es pot mostrar les parets laterals, el sostre i les finestres, tot alhora.

7.1.8 Paràmetres dels nodes de **building**Engine

Els nodes de **building**Engine generen els edificis a través d'etiquetes, com s'ha comentat anteriorment. Per fer servir etiquetes, els nodes fan ús d'un seguit de paràmetres intrinsecs dels nodes, el filter i els productes. Aquests paràmetres son:

- **filter:** El grup dels nodes d'entrada sobre els que es vol que aquest node operi.
- **product n:** L'etiqueta product que aquest node produeix. Per alguns nodes simbolitza un únic product, per altres pot ser qualsevol nombre de products resultants.

Per poder consultar aquests paràmetres es fa mitjançant els mètodes explicats a l'apartat anterior.

7.1.9 Nomenclatura del graf de nodes

En aquest apartat s'introduiran conceptes relacionats o pertinents al **building**Engine que serviran per fer més entenedors els apartats següents de la memòria.
El primer concepte que s'explicarà és el dels inputs i outputs dels nodes. Si s'observa la Figura 7.18, es pot veure com els inputs d'un node qualsevol (anomenat node actual), són els nodes enllaçats d'un nivell superior del graf, i els outputs són els nodes que reben la sortida de l'actual, i que es troben en un nivell inferior del graf.

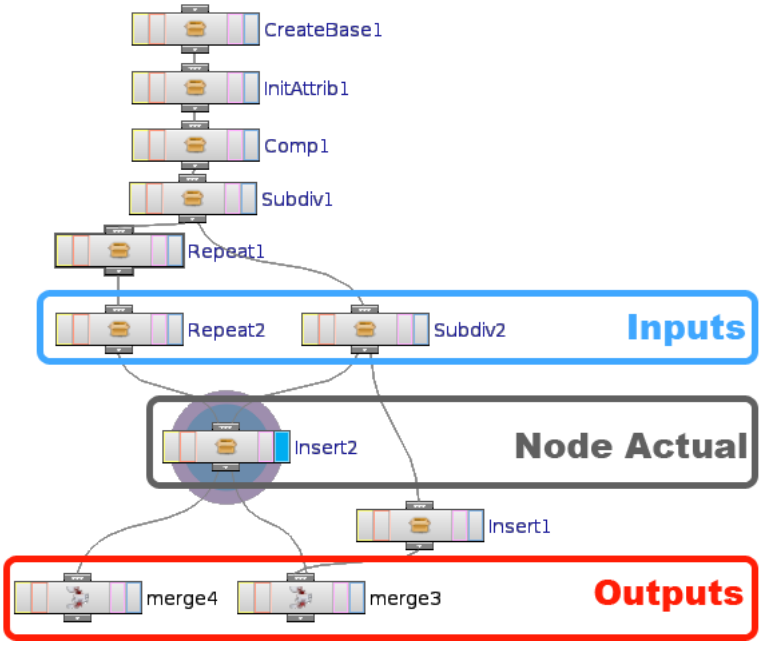


Figura 7.18: Exemple d'Inputs i Outputs de un node.

Com es pot observar, el node actual és l'anomenat (en gris) 'Insert2'. Els seus inputs són els nodes 'Repeat2' i 'Subdiv2', i els seus Outputs són els nodes 'merge4' i 'merge3'.
Un altre concepte que s'ha de tenir clar és el dels nodes fulla, arrel i nodes independents.

- Nodes fulla: no tenen outputs
- Nodes arrel: no tenen inputs
- Nodes independents: no tenen ni inputs ni outputs

Es poden observar aquests casos en la Figura 7.19:

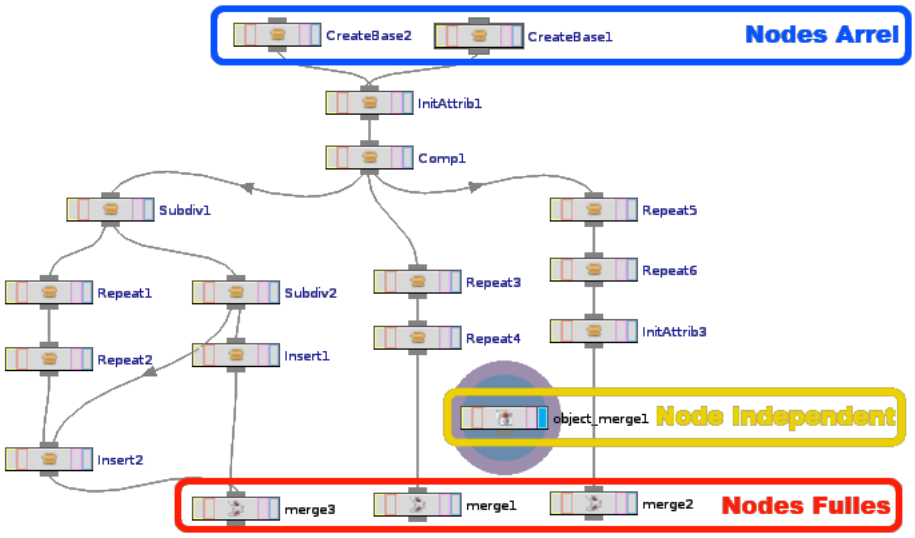


Figura 7.19: Exemple de nodes arrel, independent i fulla.

- Nodes fulla: merge3, merge1 i merge2
- Nodes arrel: CreateBase2 i CreateBase1
- Nodes independents: object_merge1

7.1.10 Arbre de primitives

Cal dir que el concepte d'arbre de primitives no és un concepte propi del **building**Engine, sinó que és propi del Houdini. Per poder explicar-lo millor, i com que nosaltres únicament treballarem amb el **building**Engine, ens basarem en l'exemple de la Figura 7.20

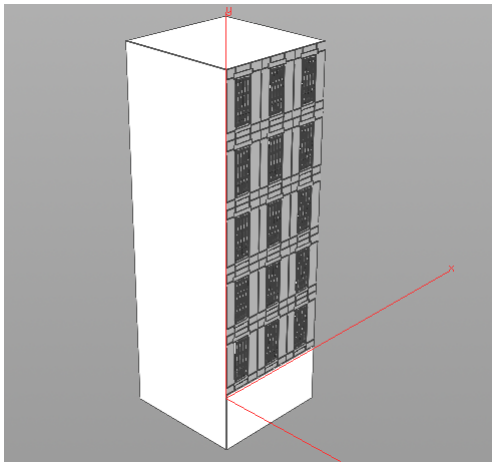


Figura 7.20: Exemple d'edifici senzill creat en l'entorn de Houdini.

El graf de la Figura 7.21 és el que genera l'edifici, on es pot veure els nodes que s'han explicat anteriorment.

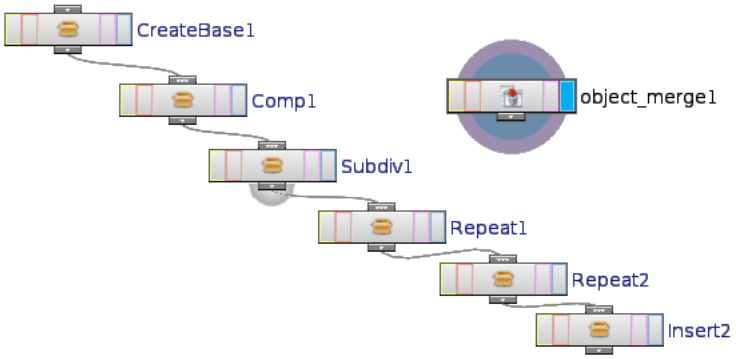


Figura 7.21: Graf de l'edifici senzill.

- CreateBase1: genera el volum
- Comp1: Divideix la base en façana, costats i teulat
- Subdiv1: Divideix planta baixa, i altres plantes
- Repeat1: Fa les divisions en Y de les plantes
- Repeat2: Fa les divisions en X de la planta
- Insert2: col·loca un asset en la posició determinada. En aquest cas, l'asset de la finestra.
- object_merge1: l'utilitzem simplement per visualitzar tot l'edifici

El que es pot observar d'aquest exemple és que es tracta d'un graf d'operacions. Per tant, si es volgués, es podria interpretar com a operacions que actuen sobre geometria, produint un arbre de resultats com el de la Figura 7.22:

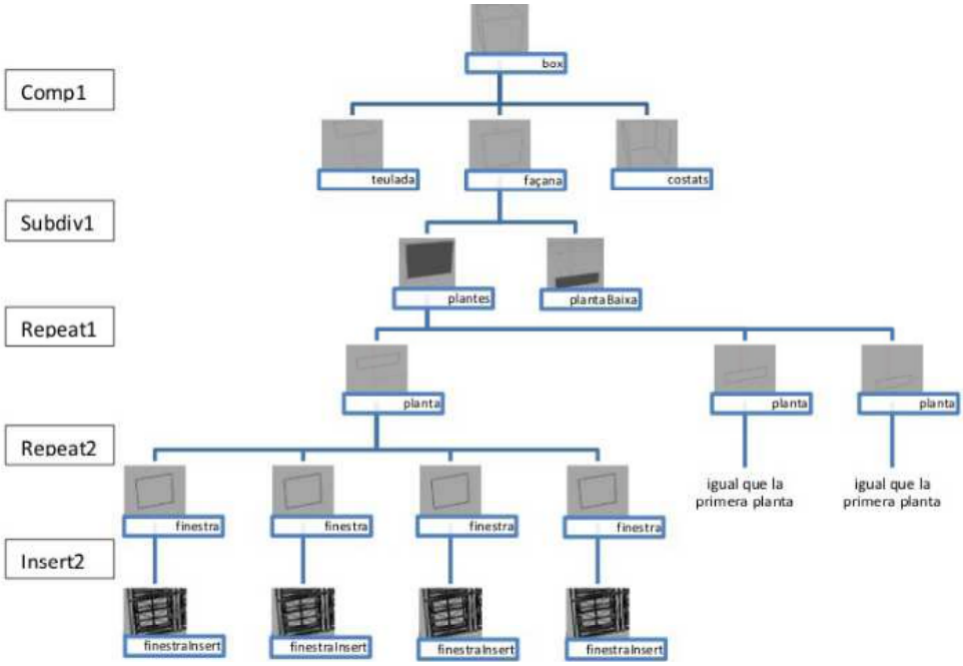


Figura 7.22: Representació de l'edifici senzill en forma d'arbre de primitives.

Tal i com es pot veure, el graf d'operacions del Houdini ha quedat representat per un arbre de primitives, on cada una té el seu TAG. Per exemple, la primera planta de l'edifici té el TAG o etiqueta "planta".

Aquí és on es pot introduir el concepte de primitives. Les primitives són les parts que genera cada operació. És important remarcar que una operació no té perquè crear només una primitiva, com el cas del comp1, que crea 3 primitives amb el tag "costats".

El que si que cal que quedi clar és que les operacions s'apliquen a cada primitiva, i gràcies a això es pot passar del graf a l'arbre i viceversa sense cap problema. Més endavant es veurà com es pot fer de manera pràctica mitjançant la llibreria d'atributs. Abans però, s'introduirà més nomenclatura sobre l'arbre de primitives.

7.1.11 Nomenclatura sobre l'arbre de primitives

A continuació es defineixen les paraules que s'utilitzaran quan es faci referència a les primitives amb l'ajuda d'una explicació i la Figura 7.23.

- Primitiva ascendent/pare: S'utilitzarà primitiva ascendent quan es faci referència a la primitiva de la que prové la primitiva actual.
- Primitiva descendent/fill: S'utilitzarà primitiva descendent quan es faci referència a la primitiva generada a partir de la primitiva actual.
- Primitiva fulla: Són les primitives a les quals se'ls hi podria aplicar un Insert. Per tant, es dirà que una primitiva fulla és una primitiva de la qual no en surt cap altre primitiva, que no siguin les generades per un Insert.
- Primitiva arrel: En aquest cas es te més d'una en un mateix arbre i són les primitives generades pel node Comp. La intuïció ens diu que hauria de ser el createBase, però aquest node només crea el volum i fins que no passa pel Comp, que és un pas indispensable

en qualsevol edifici creat utilitzant el **building**Engine, que converteix un volum 3D en superfícies 2D, no es poden considerar primitives.

- Primitiva actual: Es parlarà de primitiva actual quan es refereixi a la primitiva referenciada o seleccionada en aquell moment.
- Primitiva següent: S'utilitzarà primitiva següent quan es faci referència a la primitiva d'índex superior a la primitiva actual. Es consideren només les primitives germanes. (Descendents d'un mateix pare)
- Primitiva anterior: S'utilitzarà primitiva anterior quan es faci referència a la primitiva d'índex inferior a la primitiva actual. Es consideren només les primitives germanes.

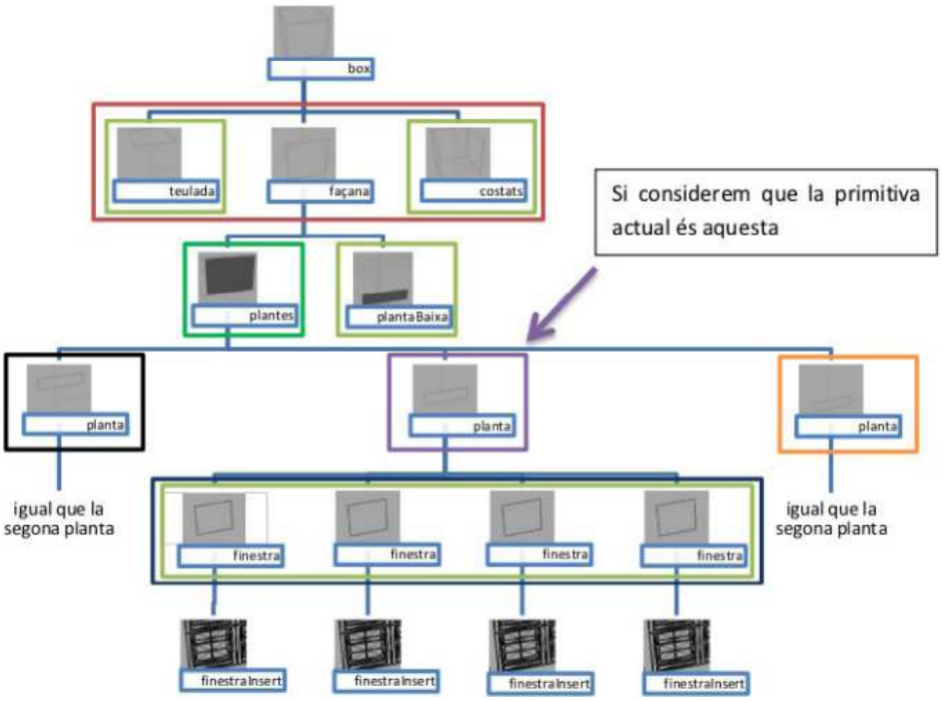


Figura 7.23: Exemple on es mostren les definicions de primitives que s'utilitzaran en aquest projecte.

7.2 Llibreria d'atributs del **skyline**Engine

A continuació s'explicarà uns altres nodes utilitzats al projecte que pertanyen a **skyline**Engine però que no són pròpiament part de **building**Engine. Es tracta d'una llibreria de nodes anomenada "atribs.otl".

7.2.1 Instal·lació de la llibreria Attribs.otl

Primer de tot, per a poder treballar amb llibreries externes (OTLs) i que els nodes d'aquestes s'incorporin al repertori de nodes de Houdini, s'ha de carregar mitjançant la interfície gràfica de Houdini, tal i com es pot observar en la Figura 7.24.

Un cop fet això, es pot observar com a l'hora de crear nodes en la secció del graf de nodes de Houdini, dins de la secció 'Digital Assets', s'hi troben els nodes que s'han carregat al instal·lar

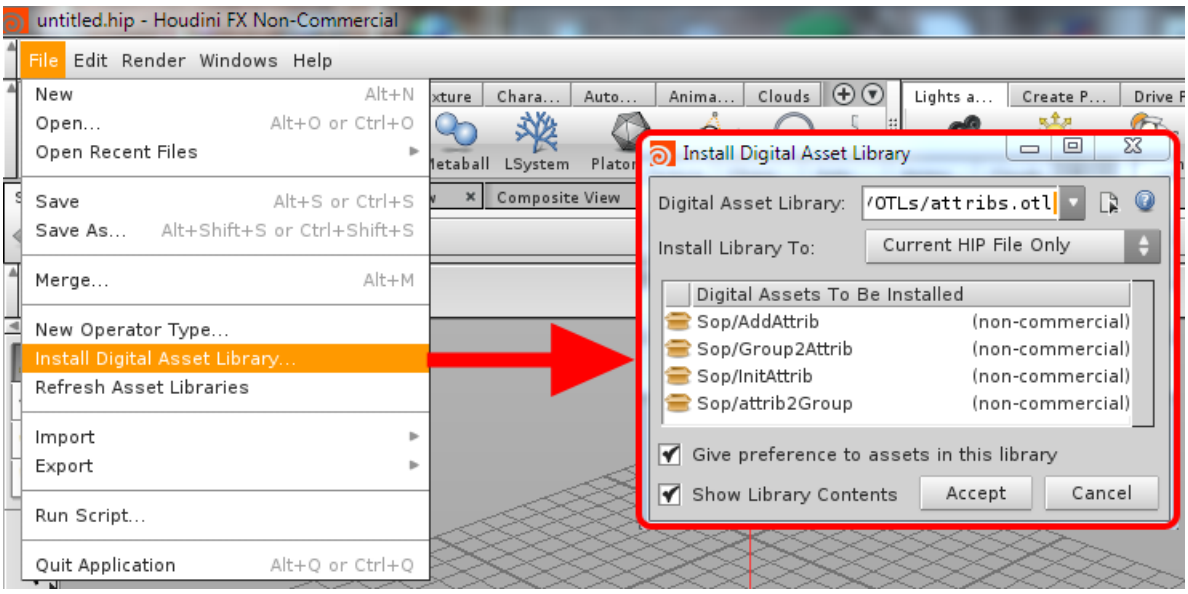


Figura 7.24: Exemple d'instal·lació de la llibreria 'attribs.otl'.

aquesta llibreria. Es pot accedir al llistat de la Figura 7.25, situant el ratolí en la secció del graf de nodes de Houdini i prement la tecla tabulador del teclat.

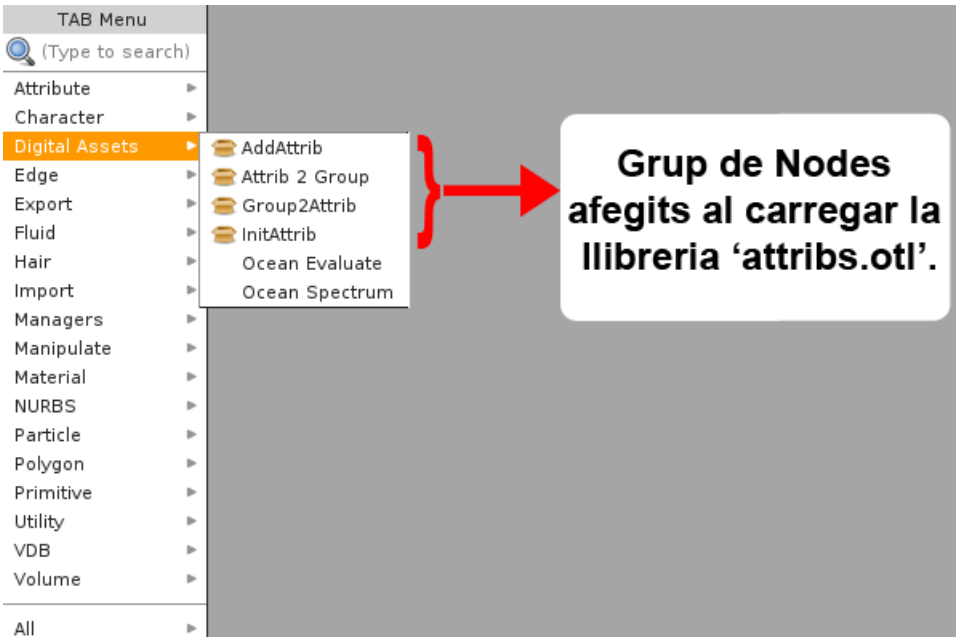


Figura 7.25: Secció i nodes afegits al instal·lar la llibreria 'attribs.otl'.

7.2.2 Funcionament de la llibreria Attribs.otl

Per explicar-ho es pot fer una analogia: en un llenguatge de programació imperatiu les variables permeten assignar a un identificador, diferents tipus de dades que es necessitaran més endavant durant l'execució dels programes que s'implementin amb tal llenguatge. La llibreria "atribs.otl" es pot equiparar a aquestes variables de programació.

Com en la majoria de llenguatges de programació imperativa, les variables necessiten ser declarades abans de fer-les servir. Aquesta inicialització indica quin tipus de dada s'assignarà o de quin tipus serà la dada que es pot trobar dins la variable.

Una vegada declarada, ja es pot fer servir la variable mitjançant una instrucció d'assignació, que segons el llenguatge de programació es farà d'una manera diferent. A la Figura 7.26 es pot veure un exemple d'inicialització de variables i d'assignació en llenguatge Java.

Llenguatge Java:
String c; //Declaració
c = "Hola Món"//Assignació

Figura 7.26: Exemple de declaració i assignació de variables en Java.

El node *iniAttrib* és el procés que es fa per declarar una variable, i el node *addAttrib* és la variable on s'assigna un valor diferent segons el moment d'execució del programa.

7.2.3 Nodes de la llibreria d'atributs del **skyline**Engine

7.2.3.1 Node initAttrib

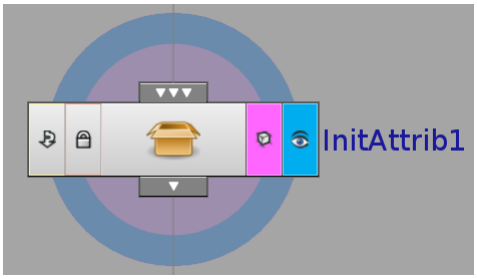


Figura 7.27: Node initAttrib a Houdini.

El node *initAttrib* és imprescindible per poder inicialitzar tot el procés de selecció de les primitives que s'han de visualitzar en cada moment.

Com ja s'ha dit anteriorment, es necessita declarar una variable. En aquesta declaració es dona un nom a la variable per identificar-la i poder assignar-hi les dades en el programa o funció que es faci servir.

El nom de la variable que es vol declarar el defineix el paràmetre "name" del node initAttrib, que a la Figura 7.28 es pot veure com a "Attrib Name".

Figura 7.28: Detall del node initAttrib i dels seus paràmetres.

A efectes pràctics, el node initAttrib és "transparent" i deixa passar tota la geometria. Per això és útil, ja que no interfereix a l'hora de tractar les primitives del node a on es connecta.

Està estretament lligat al node addAttrib. A grans trets, aquest node equival al procés d'assignació de valors a la variable que s'ha declarat inicialment. Tot seguit s'explicarà la relació amb el node addAttrib d'una manera més extensa.

Finalment, el "Attrib Name" lliga el node initAttrib amb l'attrib2Group, que permet "convertir" les primitives seleccionades en un grup. El funcionament del node attrib2Group s'explicarà més endavant.

7.2.3.2 Node addAttrib

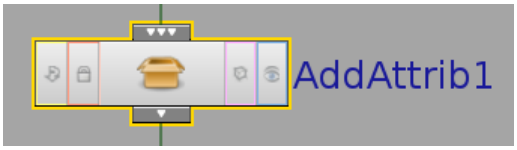


Figura 7.29: Node addAttrib a Houdini.

Seguint amb l'analogia amb els llenguatges de programació, el node addAttrib és el procés que es fa a l'hora d'assignar un o més valors a la variable que s'ha declarat prèviament amb el node initAttrib. Es necessita especificar quina és la variable (el seu nom) on es vol fer l'assignació, per això al paràmetre "Name" s'hi escriu el nom que s'ha assignat al node initAttrib, ja explicat en l'apartat anterior. Es pot veure enquadrat en vermell a la Figura 7.30, que el nom coincideix amb el "Attrib Name" del node initAttrib.

El node addAttrib permet seleccionar un o varis números de primitiva i desar aquest valors com si es tractés d'una variable. Aquest valor es desa en el paràmetre "Pattern", com es pot observar a la Figura 7.30 enquadrat en blau.

La funció d'aquest paràmetre és marcar la primitiva que s'ha desat amb una etiqueta. Aquesta etiqueta és necessària més endavant en el node attrib2Group, com es veurà en el següent apartat. Aquest paràmetre es pot observar enquadrat en verd en la Figura 7.30.

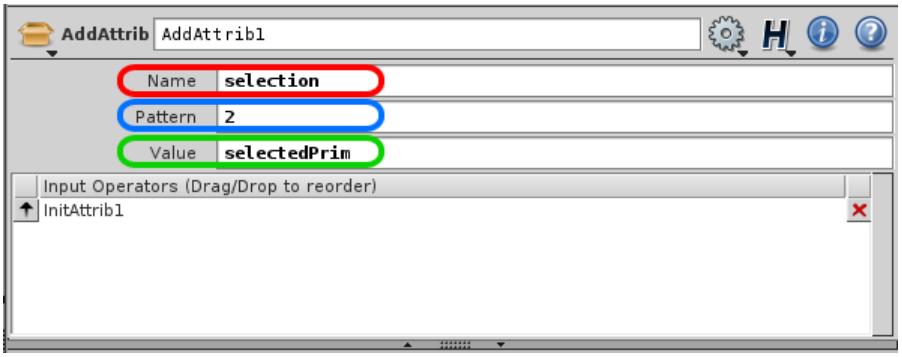


Figura 7.30: Detall del node addAttrib i dels seus paràmetres.

7.2.3.3 Node attrib2Group

Figura 7.31: Node Attrib2Group a Houdini.

Per poder visualitzar els resultats es necessita la intervenció del node attrib2Group.

El node attrib2Group, tal com indica el nom, permet fer la "traducció" d'un atribut a un grup. Com s'ha vist anteriorment:

- S'ha assignat un nom a la variable, el paràmetre "Attrib Name" del node initAttrib.
- S'han assignat valors a aquesta variable que s'ha ceat, paràmetres "Name" i "Pattern" del node addAttrib.
- S'ha marcat els diferents valors que anava agafant la variable "addAttrib" amb una etiqueta, definida pel paràmetre "value" del addAttrib.

Com s'ha introduït en els apartats anteriors, els nodes initAttrib i addAttrib s'han de lligar d'alguna manera amb el node attrib2Group. El lligam es fa a través dels valors dels paràmetres del node attrib2Group.

El node té un paràmetre anomenat "Atrib Name", aquest paràmetre permet saber a quina variable s'ha de lligar l'attrib2Group. En aquest cas, s'escriu el nom de la variable que s'ha introduït al crear el node initAttrib. En la Figura 7.32 es pot veure en color blau.

Un cop lligat amb la variable que s'ha declarat inicialment, s'indica quina serà la "traducció" que s'ha de fer per a que el node delete entregui amb quin grup o subgrup ha d'operar.

S'utilitza l'únic atribut que té el node. Aquest atribut consta d'un "value" i un "group" que van lligats. En aquest cas, dins del "value" s'introduirà el nom de l'etiqueta amb que s'han marcat els diferents valors que el node addAttrib ha anat prenent. Així, com es veu a la Figura 7.32 enquadrat en color vermell, s'escriu "selectedPrim", que és l'etiqueta que s'ha escrit en el paràmetre 'value' de l'addattrib creat anteriorment. En el "group" s'escriurà el nom del grup que es vol que s'associï.

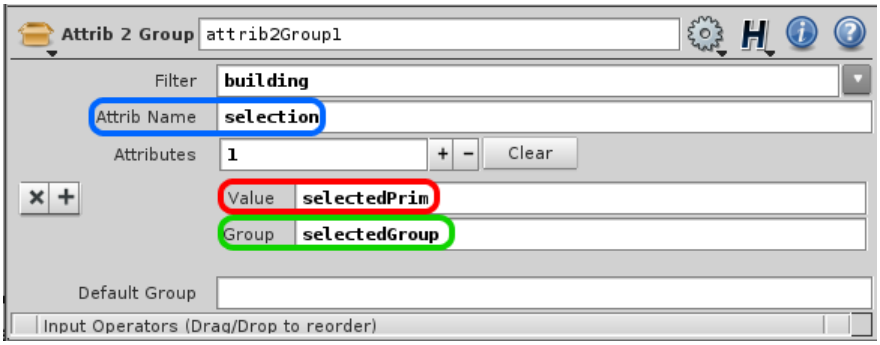


Figura 7.32: Analogia del node attrib2Group

Aquest node va estretament lligat amb initAttrib i addAttrib, com s'ha pogut veure en l'explicació anterior. Si es segueix amb l'analogia amb els llenguatges de programació, l'attrib2Group és com l'estructura "switch":

```
switch(selection):
        selectedPrim;
                return "selectedGroup";
        ...
        default:
                ...;
```

No es mostraran exemples de funcionament, ja que l'objectiu és explicar l'existència i el seu ús a alt nivell. Aquest mòdul és necessari per a aquest projecte però el seu ús no és directe si no que es fa servir com a eina en sub-processos i no és l'objectiu aprofundir més del que s'ha explicat en aquest treball.

7.3 Python

Python és un llenguatge de programació creat per Guido van Rossum en l'any 1990. Es compara habitualment amb TCL, Perl, Scheme, Java i Ruby. En l'actualitat Python es desenvolupa com un projecte de codi obert, administrat per la Python Programari Foundation. Aquest projecte s'ha desenvolupat amb la versió 2.6.

Python és considerat com la "oposició lleial"a Perl, llenguatge amb el qual manté una rivalitat amistosa. Els usuaris de Python consideren a aquest molt més net i elegant per a programar. Python permet dividir el programa en mòduls reutilitzables des d'uns altres programes Python. També hi ha mòduls inclosos que proporcionen E/S de fitxers, crides al sistema, sockets i fins a interfícies a GUI (interfície gràfica amb l'usuari) GTK, Qt, WxWidgets(WxPython), PMW entre altres.

Python és un llenguatge interpretat, el que estalvia un temps considerable en el desenvolupament del programa, perquè no és necessari compilar ni enllaçar. L'intèrpret es pot utilitzar de manera interactiva, el que facilita experimentar amb característiques del llenguatge, escriure programes d'un sol ús o provar funcions durant el desenvolupament del programa. El principal objectiu que persegueix aquest llenguatge és la facilitat, tant de lectura, com de disseny.

Hem utilitzat Python com a llenguatge per desenvolupar el projecte perquè com hem explicat, Houdini té com a llenguatge incorporat Python, és a dir, per complir els requisits esmentats

7.3.1 Python dins de Houdini

Primer de tot cal aclarir les dues maneres de fer servir Houdini. La primera, i més habitual, és utilitzant la interfície d'usuari del programa. La segona és fent servir la consola de Python que es pot observar en la Figura 7.33. Usant la consola es poden fer totes les accions disponibles a la interfície gràfica. Com que Python és un llenguatge interpretat, es pot executar funcions en temps real, tal i com es faria prement un botó del programa.

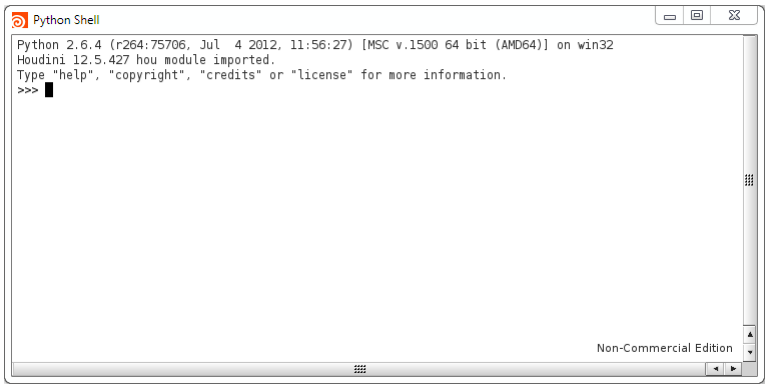


Figura 7.33: Captura de la consola de Python.

El sistema Python que té Houdini és exactament el mateix que es pot descarregar des de la web oficial de Python. L'única diferència és la incorporació d'un mòdul addicional anomenat hou (que explicarem a l'apartat següent) que conté totes les funcionalitats, objectes i propietats de Houdini. Aquesta API rep el nom de HOM (Houdini Object Model).

7.3.2 Mòdul HOU

Aquest mòdul és la base per interactuar amb Houdini a través de Python, és a dir, des del Python a dins de Houdini podem interactuar amb els nodes creats per l'entorn gràfic o afegir-ne de nous a trav s d'aquest mòdul.

A la Figura 7.34 podem veure un exemple creant un CURVE amb la consola de Python. Primer de tot s'importa el mòdul hou. Seguidament es crea un objecte de tipus 'geo' que emmagatzemarà tots els objectes. Es pot veure com crea un nou node de tipus "File" a l'interior:

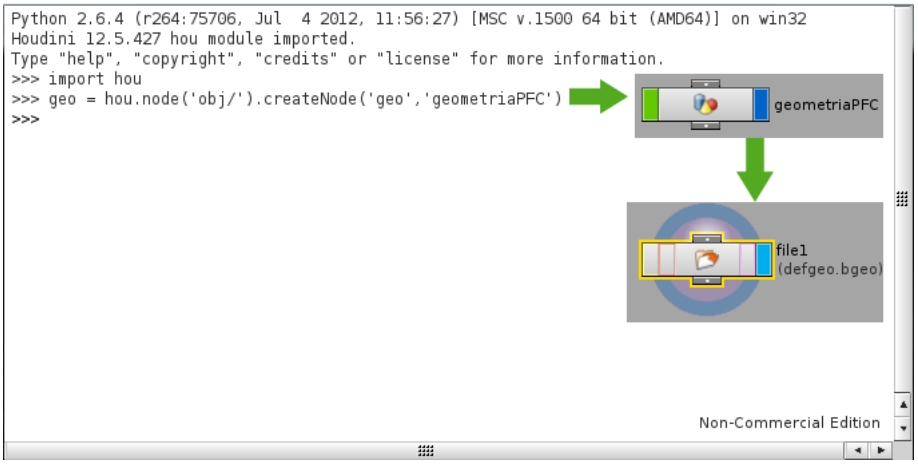


Figura 7.34: Captura de la consola de Python amb l'arbre de nodes que es va construint.

Tot seguit eliminem els elements que Houdini crea automàticament en construir un objecte "geo", es a dir, els tipus "file". Finalment es crea el "curve" utilitzant al sentència "createNode()".

Finalment la Figura 7.35 afegim les coordenades a la corba per tal de visualitzar el contingut a la zona de treball:

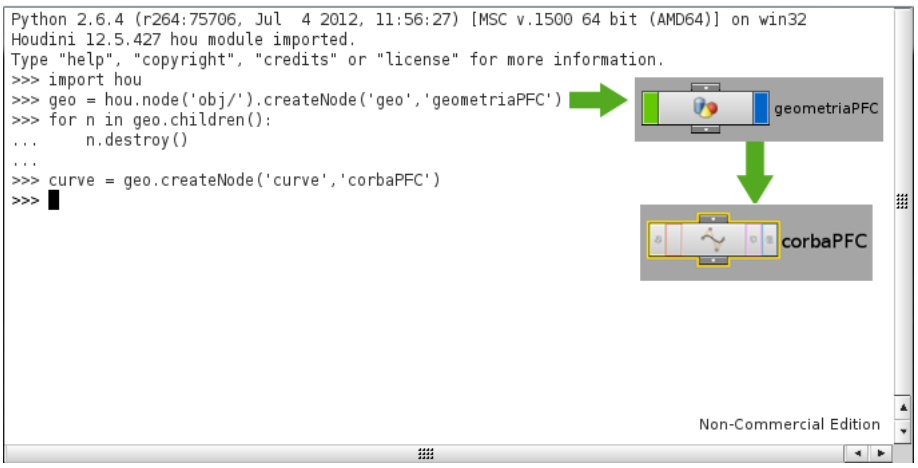


Figura 7.35: Captura de la consola de Python amb l'arbre de nodes que es va construint.

7.4 Editor de Text

Per a poder desenvolupar codi en Python es necessita un editor de text. Hi ha molts editors de text, des de el notepad de Windows, passant per worpad, Emacs, Gedit, jEdit, Kate, etc. però en aquest cas com s'han volgut fer proves tant a MacOS com a Windows, es volia que fossin coneguts o dels més populars.

A diferència de les altres eines, aquest aspecte no estava acotat per cap restricció, i per tant es van escollir els que es coneixien fins aquell moment. Principalment per Windows el Notepad++ i el bloc de notas ja que en alguns casos interessa obrir els fitxers en text pla, per temes de indentacions.

7.4.1 Notepad++

Notepad++ és una eina gratuïta, amb llicencia GPL per crear i editar text que va sorgir amb la intenció de substituir el Notepad de Windows i millorar les funcionalitats d'aquest.

Esta basat en el component d'edició Scintilla, tot i que Notepad++ està escrit en C++ i fa ús de la API Win32 i STL els quals asseguren una ràpida execució i una mida de programa petita.

Com a funcionalitats més importants té:

- Resaltar la sintaxi segons el llenguatge de programació.
- PCRE, Perl Compatible Regular Expression per cerca i reemplaçament.
- Entorn Gràfic minimalista
- Multi-document
- Zoom.
- Multi-llenguatge

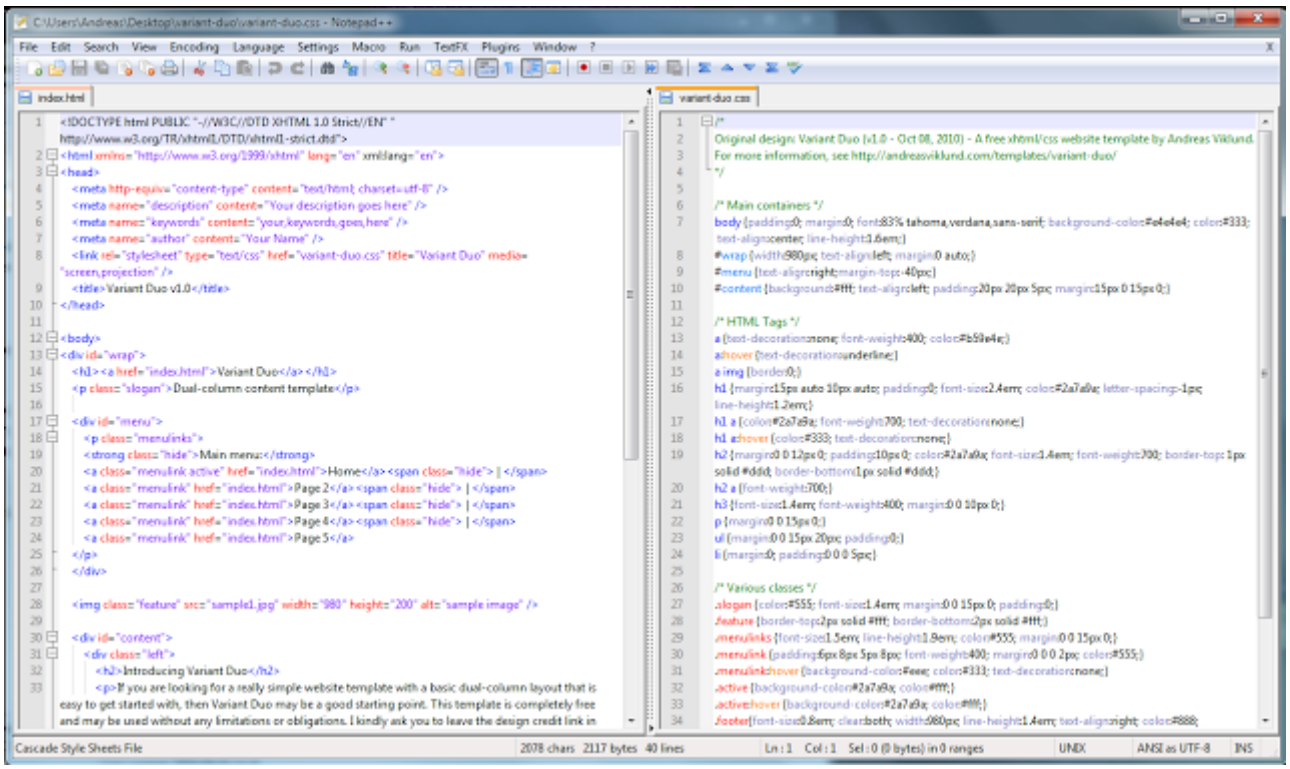


Figura 7.36: Exemple d'execució del Notepad++.

7.5 LaTeX

LaTeX és un sistema de composició de textos d'alta qualitat tipogràfica, orientat especialment a la creació de llibres, documents científics i tècnics que continguin fórmules matemàtiques, figures, citacions, etc.

Està format per un gran conjunt de macros de TeX, escrit per Leslie Lamport al 1984, amb la intenció de facilitar l'ús del llenguatge de composició tipogràfica TeX, creat per Donald Knuth. Aquest llenguatge o sistema de composició de textos està subjecte a la llicència *LaTeX project public license 1.3c (LPPL 1.3c)*, tal que ha permès que s'estengués a nivell mundial molt ràpidament i s'assolís gairebé com un estàndard a l'àmbit tècnic i científic. A més a més és un sistema multi-plataforma, per tant es ideal per a gestionar textos en qualsevol sistema operatiu.

El llenguatge d'instruccions per a poder composar textos és de baix nivell, entenent que el sentit de les accions resultants d'aquestes instruccions són molt elementals. Això fa que sigui un llenguatge molt versàtil i fàcil d'ampliar, podent fins i tot generar un conjunt de regles i instruccions pròpies per als propòsits personals de cadascú. A més té l'avantatge de permetre re-aprofitar estructures de documents, un cop creades.

Per totes aquestes raons s'ha fet servir per elaborar aquesta documentació del projecte de final de carrera, fent servir TeXWorks com a plataforma de suport i compilació per LaTeX.

7.5.1 TeXWorks

TeXWorks és un programa de software lliure distribuït sota la llicència GPL per a l'elaboració de documents basats en el llenguatge de composició tipogràfic TeX. TeXWorks també és multi-plataforma, pel que es pot trobar tant per Windows, Linux i MacOS X. En el cas d'aquest treball ha sigut d'interès per a poder composar amb LaTeX. La tasca de TeXWorks és la d'aportar un front-end (un entorn de desenvolupament) per tal de escriure, interpretar, compilar, donar un feedback a l'usuari si s'han produït errors i on, i finalment generar el PDF resultant.

TeXWorks és un conglomerat d'un conjunt important de "petites" eines que en el seu conjunt permeten fer les tasques anteriorment descrites. D'entre aquestes eines es pot trobar el bibTeX que s'explicarà més detalladament a continuació.

7.5.2 BibTeX

BibTeX ha sigut àmpliament utilitzat des de la seva introducció gràcies a Oren Patashnik fa uns 20 anys. Tal i com el seu nom indica, va ser una aportació destinada a ser feta servir en consonància amb el llenguatge de composició tipogràfica LaTeX, però avui en dia ha estat incorporat en molts altres llocs mitjançant eines third-party. En la Figura 7.37 es pot veure la seva incorporació a TeXWorks.

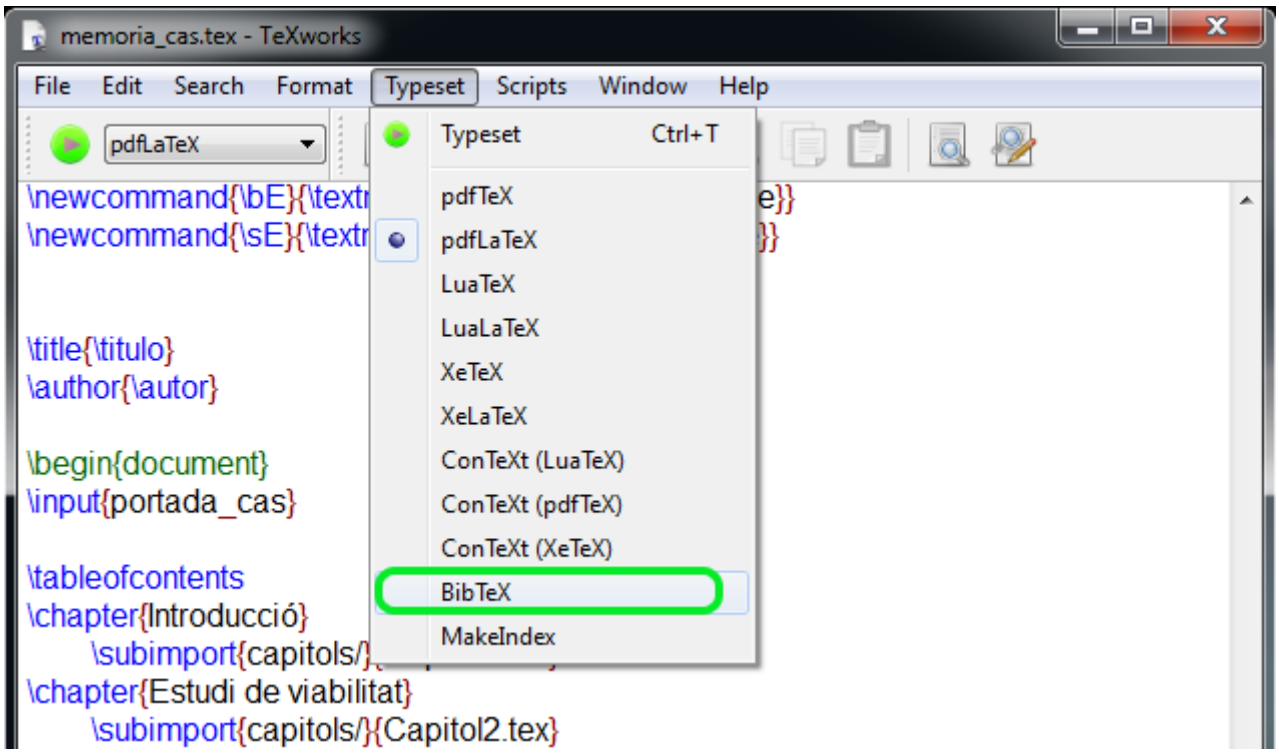


Figura 7.37: BibTeX integrat en TeXWorks.

BibTeX fa servir un format de text pla, el qual pot ser creat i modificat fent servir qualsevol editor de text per l'usuari. Existeixen eines per a un tractament més còmode de les entrades de BibTeX però en aquest projecte s'ha fet servir el mateix Notepad++ per a fer la edició de les entrades de la bibliografia.

Aquest format té l'avantatge que avui en dia, els propis autors, editorials i revistes aporten les entrades en aquest format per a que puguin ser citades sense necessitat d'haver d'introduir des de zero cadascun dels camps necessaris de cada entrada. A la Figura 7.38 es pot veure un exemple d'una entrada d'un dels papers citats en aquesta memòria.

```
@article{ barroso2013visual,
    title={Visual<i> copy & paste</i> for procedurally modeled buildings by ruleset rewriting},
    author={Barroso, Santiago and Besuievsky, Gonzalo and Patow, Gustavo},
    journal={Computers & Graphics},
    year={2013},
    publisher={Elsevier}
}
```

Figura 7.38: Exemple d'entrada de BibTeX.

7.6 Apache Batik SVG Toolkit 1.7

Batik és un conjunt d'eines basat en Java per aplicacions o applets que necessitin utilitzar imatges SVG (Scalable Vector Graphics). Aquest mòdul ha sigut de gran interès per al projecte ja que permet, a més de generar imatges en format SVG, transcodificar aquestes imatges a altres formats d'imatge. En concret ha sigut d'interès per transcodificar al format PNG per tal d'accelerar la interacció de l'usuari.

Aquesta eina permet l'execució de les operacions esmentades anteriorment mitjançant la línia de comandes i fins i tot es pot integrar com una crida des de altres llenguatges de programació. Això ha facilitat la integració i transparència com a llibreria d'aquest projecte.

7.7 GANTT Project

Gantt Project és una aplicació de programari lliure que permet realitzar diagrames de Gantt. Un diagrama de Gantt és una eina gràfica que te com a objectiu veure fàcilment la planificació d'un projecte mostrant el temps de dedicació per a cadascuna de les tasques.

El diagrama de Gantt no indica les relacions existents entre activitats, però la posició de cada tasca al llarg del temps fa que es puguin identificar aquestes relacions i dependències.

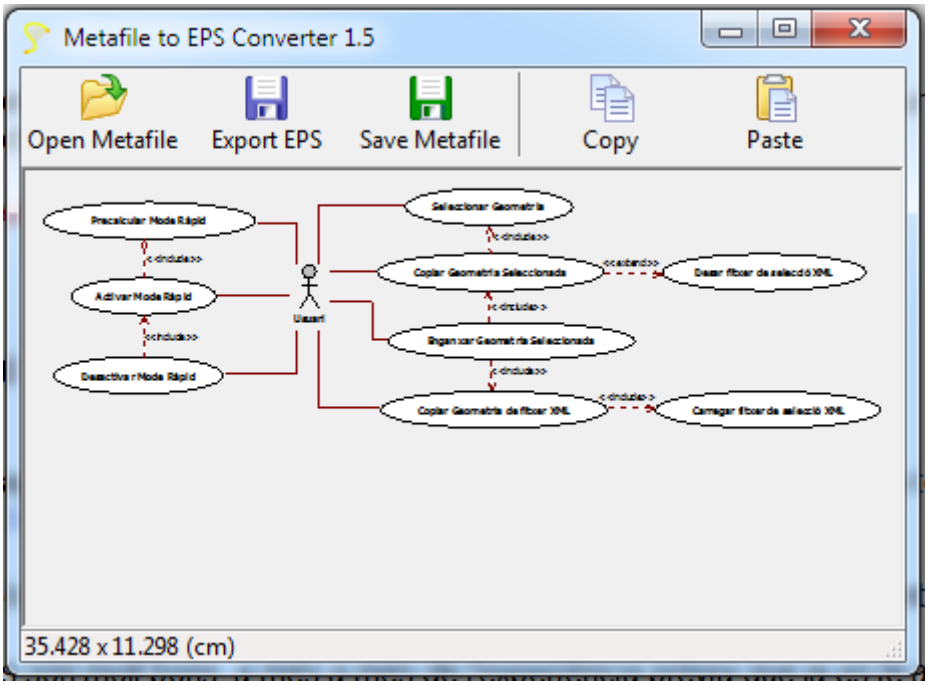
S'ha escollit aquest programa perquè és de programari lliure i els resultats són molt bons, a més a més de l'experiència prèvia que ja es té amb aquesta eina.

7.8 StarUML

StarUML és un projecte de codi obert per al desenvolupament flexible i ràpid de diagrames UML/MDA per a Windows. L'objectiu d'aquesta plataforma és construir una eina de modelatge de software com a alternativa al software comercial de UML tals com Rational Rose, Together i molts d'altres.

S'ha escollit aquesta aplicació ja que permetia desenvolupar tots els diagrames necessaris pel projecte, tals com: diagrames de classe, de cas d'ús, d'activitat, etc.

7.9 Metafile to EPS Converter



Metafile to EPS Converter és, tal i com el nom indica, un convertidor entre el format *.WMF* (Windows Meta File) al format *.EPS* (Encapsulated PostScript). Tots dos formats són formats d'imatge vectorials, és a dir, que poden ser escalables sense tenir pèrdues. Aquest convertidor de codi obert i multiplatforma permet realitzar la conversió dels dos formats explicats anteriorment de manera que es puguin incorporar les imatges en format *.EPS* en LaTeX. Així es pot gaudir de la funcionalitat de la lupa, i tot i tenir el document PDF les imatges no tenen pèrdues i es poden ampliar tant com es vulguin.

En el cas d'aquest projecte, amb la eina de la secció anterior 'StarUML' es desen els diagrames de l'anàlisi i disseny, i d'altres apartats que es puguin fer amb aquest programa, són exportats a format *.WMF* i després convertits amb aquesta eina a format *.EPS*. Integrant d'aquesta manera els gràfics a LaTeX.

Capítol 8

Anàlisis i disseny

En aquest capítol s'estudiaran i definiran els requeriments del sistema. Amb aquest objectiu, es farà servir el Unified Modeling Language (UML), que és un llenguatge de modelatge estàndard per l'enginyeria del programari.

Abans de començar, cal dir que es va plantejar el sistema des del punt de vista de l'usuari, ja que el sistema havia de ser tan còmode i usable com fos possible. Per tant, primer es va dissenyar la interfície gràfica, tal i com es pot veure a la temporalització, i després es va implementar les funcionalitats de Copy&Paste, acceleració i XML.

Per a realitzar el disseny d'aquest projecte es va agafar de referència l'article Barroso et al. [3]. En aquest article proposa una solució a una eina per a realitzar replicació de geometria mitjançant modelatge procedural, i acota de quina manera es copien i re-escriuen les regles, mitjançant excepcions, per a poder dur a terme aquesta tasca.

8.1 Descripció de la solució

La solució que es vol aportar és la de proporcionar una eina a l'usuari per tal que el procés de replicar geometria sigui transparent. Per això es va decidir que l'eina de selecció habia de ser amb el ratolí, igual que funciona en els programes d'edició gràfica. Per tal de començar la selecció, l'usuari te un botó amb el qual pot habilitar el mode de selecció i començar a seleccionar.

El procès de selecció comença per l'acció de l'usuari de seleccionar la part de la façana que vol copiar. Això ho pot fer polígon a polígon, mantenint premut la tecla *Shift* i seleccionant cadascun d'aquests amb el ratolí, o bé prement el botó esquerra del ratolí sobre l'edifici i mantenint-lo premut arrossegar-lo per tal de crear una finestra de selecció. Un cop ha seleccionat quelcom i ha premut la tecla "Enter", comença el procés de selecció.

D'aquesta selecció Houdini retorna un llistat de les primitives seleccionades. Aquest llistat, però, és numèric i és consecutiu per tots els triangles de l'escena. Per aquesta raó, no es té cap informació sobre a quin node genera cada primitiva i quin és el seu número respecte el seu node. Per saber-ho, s'ha de fer un recorregut per l'arbre de primitives i cercar-los. Aquesta operació és molt costosa per aquesta raó es fa una optimització prèvia. Quan Houdini retorna el llistat de primitives, en ocasions retorna rangs de la forma "75-100". Això significa que s'han seleccionat un conjunt de primitives que pertanyen al mateix conjunt la qual cosa significa que no aporten nova informació sobre nous nodes. Per això, el que es fa és quedar-se només amb el primer

número de primitiva del rang i és aquesta primitiva la que es fa servir per a cercar el node. D'aquesta manera s'optimitza la cerca dels números de primitiva respecte el node que les genera.

Un cop es té tots els números de primitiva respecte el node que les genera, es pot començar la següent fase. Aquesta tracta de trobar l'ancestre comú de totes les primitives seleccionades. Per fer-ho es recorre l'arbre de primitives des de cadascuna de les seleccionades fins l'arrel. Un cop es tenen tots els camins es va seleccionant el primer de cada camí i es compara, si són iguals a les hores son ancestre comú. Arribat al punt on un dels camins canvia, ja deixa de ser comú. Per tant la solució radica en l'últim comú trobat. Es pot veure un exemple de l'explicació en la Figura 8.1 Aquesta primitiva és la que sustenta el conjunt de regles que generen tota la geometria seleccionada. Al seleccionar-se aquesta primitiva es du a terme una selecció neta, ja que no es selecciona mai mitja finestra, o part de la geometria, si no que es selecciona parts atòmiques de la geometria (una finestra sencera, una porta sencera, etc).

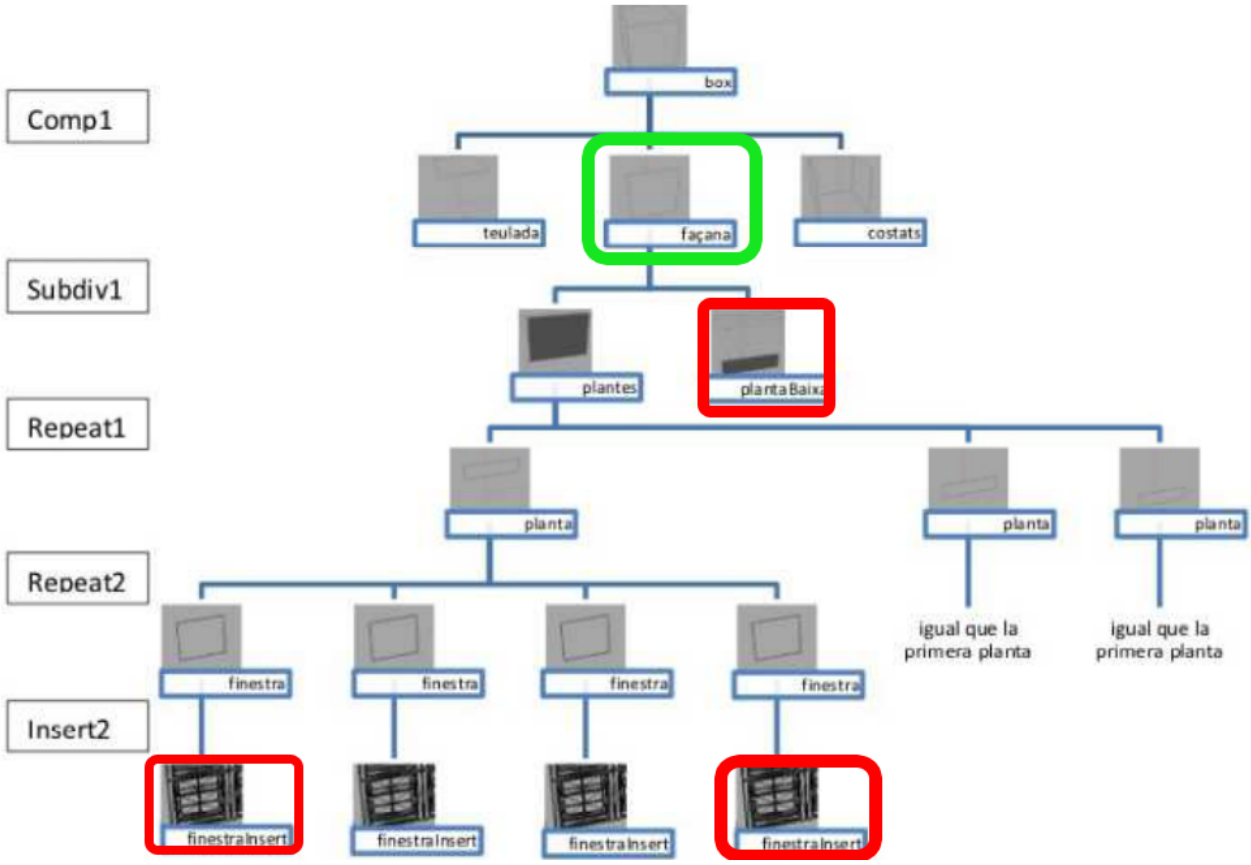


Figura 8.1: Exemple d'ancestre comú (verd) seleccionant les primitives marcades (vermel).

Per tal de realitzar la còpia de la selecció, el que es fa, un cop es sap la primitiva ancestre comú de tota la selecció, és agafar com a arrel aquesta primitiva. Es recorre tot el sub-arbre que genera i es copia en un vector en ordre topològic i en acabar es realitza una còpia d'aquesta estructura en el buffer de Houdini. A més de la còpia geomètrica, es desa també informació sobre:

- Mides de la primitiva ancestre comú (per poder proporcionar-la més endavant)
- Llistat de tags que hereten les primitives de la selecció en aquest edifici, ja que si s'enganxa això en un altre de diferent, aquestes etiquetes es perdrien i no es visualitzaria correctament.

- El número de la primitiva respecte el node que es genera que s'ha trobat.
- Nom i ruta del node que genera aquesta primitiva.

Aquesta tasca es desencadena per l'acció de l'usuari al prémer el botó de copiar la selecció. Si abans de prémer aquest botó, s'ha definit una ruta per desar un fitxer XML amb la còpia, al prémer el botó a més de fer-se la còpia al buffer de Houdini, també es fa en el fitxer XML.

Un cop copiada la geometria arriba l'hora d'enganxar-la. Per fer-ho l'usuari ha de tornar a fer el procés de selecció que s'ha explicat anteriorment i de tal acció se n'extreu la primitiva ancestre comú de les seleccionades, com ja s'ha explicat. Un cop feta la selecció l'usuari prem el botó d'enganxar i comença el procés. Aquesta funcionalitat comença per localitzar el node que crea la primitiva ancestre comú seleccionada. De fet, per ser més clars, es desa la mateixa informació que la selecció anterior però aquesta informació és respecte a la primitiva "destí", o sigui on s'enganxarà la geometria.

El procés d'enganxat s'explicarà amb ajuda de la Figura 8.2 extreta de l'article de Barroso et al. [3].

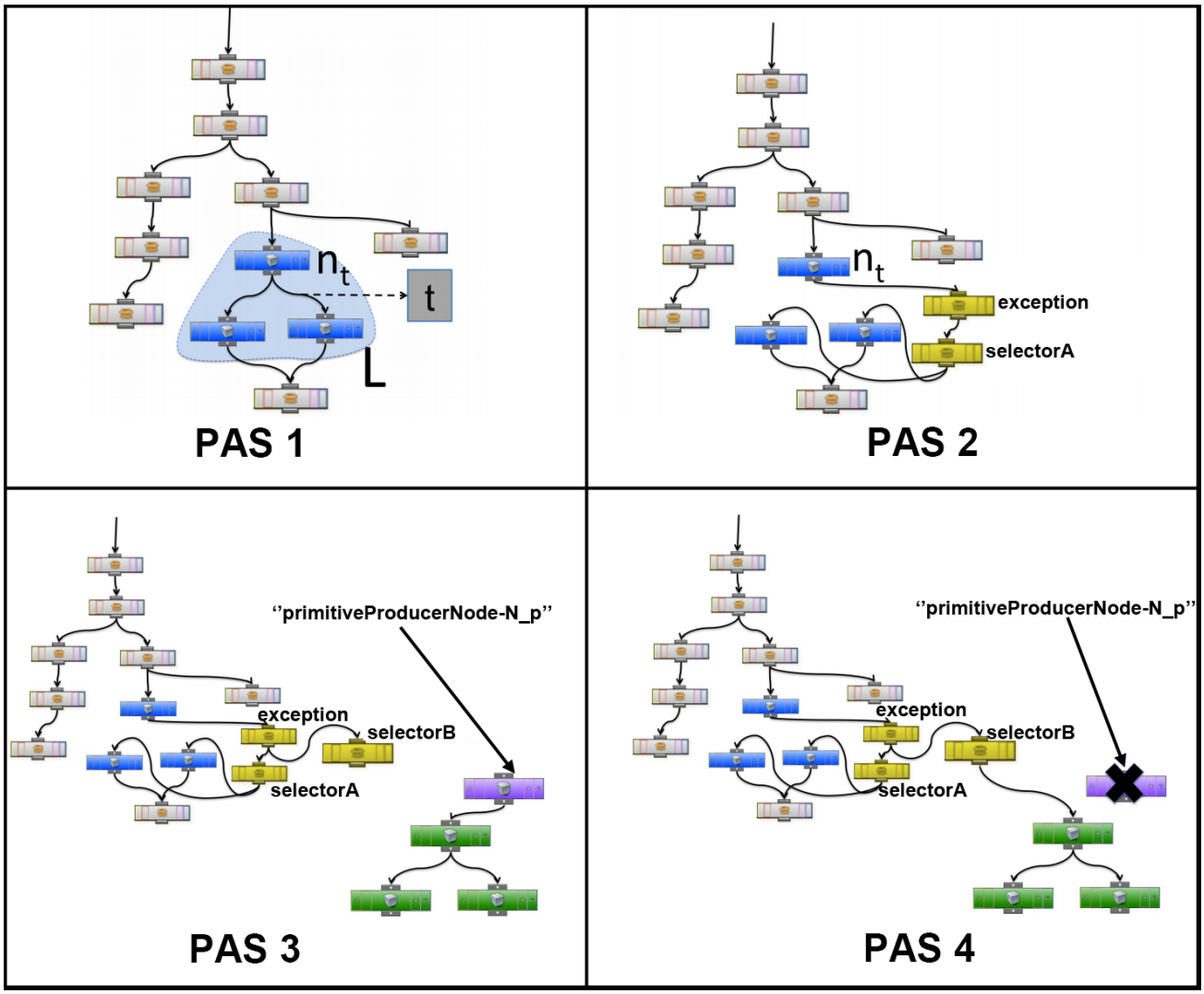


Figura 8.2: Procés conceptual d'enganxat de geometria.

El node que genera la primitiva ancestre comú trobada mitjançant la selecció del destí, es

pot veure denotada en la Figura 8.2 pel node n_t **(PAS 1)**. Per tal de dur a terme l'enganxat el que es fa és crear un node excepció just després del node n_t. A continuació, es crea un node selecció **(selectorA)** en el qual l'hi connecta el sub-graf que ja tenia com a descendent el node n_t **(PAS 2)**. Seguidament es crea un altre selector **(selectorB)**. Un cop fet això, només cal enganxar el graf copiat en el buffer de Houdini o de fitxer si és el cas que s'ha carregat des de fitxer (l'origen de les dades és transparent) **(PAS 3)**. Es pot veure el sub-graf enganxat marcat en verd. Un cop enganxat es cerca el node arrel del sub-graf enganxat (verd), el qual ha sigut alterat mitjançant el nom de **"primitiveProducerNode-N_p"** per a ser distingit en aquest punt. Aquest node només s'ha copiat ja que és el que sustenta el sub-graf, però s'ha de substituir pel selector **(selectorB)** creat anteriorment. D'aquesta manera, un cop copiats els tags, afegits els nodes excepció i selectors i eliminat el node **"primitiveProducerNode-N_p"** arrel del sub-graf copiat, s'ha efectuat l'enganxat **(PAS 4)**. Només cal refrescar l'edifici per visualitzar els canvis.

8.2 Disseny general

A continuació s'explicaran les diferents parts del disseny d'aquest programa i es mostrarà, mitjançant diferents diagrames, els casos que es contemplen. Per a començar, s'exposaran els diagrames de cas d'ús, on es podrà veure com l'usuari es relaciona amb l'aplicació i viceversa.

8.3 Diagrama de cas d'ús

En enginyeria del programari, un cas d'ús és una tècnica per a la captura de requisits potencials d'un nou sistema o una actualització de programari. Cada cas d'ús proporciona un o més escenaris que indiquen com hauria de interactuar el sistema amb l'usuari o amb un altre sistema per aconseguir un objectiu específic. Normalment, en els casos d'usos s'evita l'ús d'argots tècnics, preferint en el seu lloc un llenguatge més proper a l'usuari final.

En altres paraules, un cas d'ús és una seqüència d'interaccions que es desenvoluparan entre un sistema i els seus actors en resposta a un esdeveniment que inicia un actor principal sobre el sistema. Els diagrames de casos d'ús serveixen per especificar la comunicació i el comportament d'un sistema mitjançant la interacció amb els usuaris i/o altres sistemes.

S'utilitzen sobretot per il·lustrar els requeriments del sistema en mostrar com reacciona a esdeveniments que es produeixen en el seu àmbit o en ell mateix.

8.3.1 Diagrama de cas d'ús general

En la Figura 8.3 i Figura 8.4 es mostren els diagrames de cas d'us de context general d'aquest projecte.

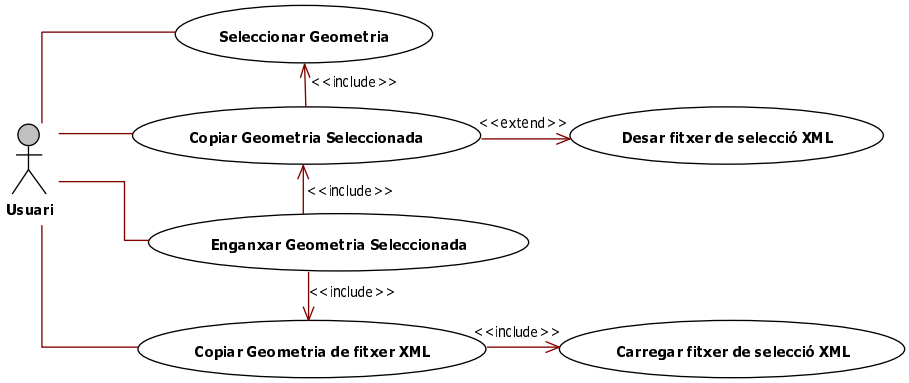


Figura 8.3: Diagrama de cas d'us de context general.

El diagrama de la Figura 8.3 es correspon a la branca principal d'aquest projecte, on s'encapsulen totes les funcionalitats relacionades amb el procés de Copy&Paste.

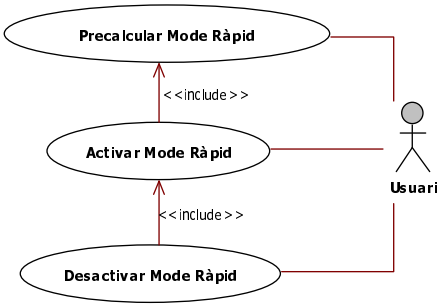


Figura 8.4: Diagrama de cas d'us de context general.

En la Figura 8.4 es poden veure els casos d'ús relacionats amb la branca on s'encapsulen totes les funcionalitats de l'acceleració de la interacció.

8.3.2 Fitxes dels casos d'ús i diagrames d'activitat dels casos d'ús

Un cas d'ús és una tècnica per a la captura de requisits potencials d'un nou sistema. Cada cas d'ús proporciona un o més escenaris que indiquen com hauria d'interactuar el sistema amb l'usuari i viceversa, per aconseguir l'objectiu que es pretén.

En el cas d'aquest projecte, l'actor és l'usuari final que pot be ser un dissenyador, arquitecte, modelador o artista. Tots aquests actors tenen el mateix rol, per tant els casos d'ús s'enfocaran a aquest tipus d'usuari.

8.3.2.1 Diagrama i fitxa de Precalcular Mode Ràpid

Aquesta funcionalitat permet pre-calcular l'edifici que es té obert per tal de poder activar i desactivar el mode d'acceleració d'interacció amb l'usuari. D'aquesta manera si l'edifici conté molta geometria i ralentitza la plataforma, l'usuari té una alternativa per a poder continuar treballant.

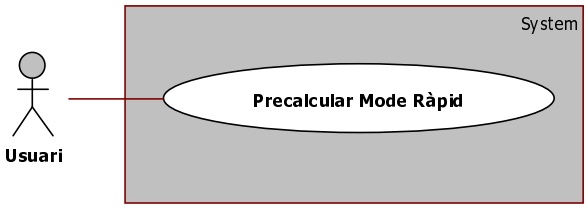


Figura 8.5: Diagrama de cas d'us de Precalcular Mode Ràpid.

Fitxa de cas d'ús:	Precalcular Mode Ràpid
Descripció:	Es pre-calcula tot l'edifici i es generen les imatges per a poder activar i desactivar el mode d'acceleració d'interacció de l'usuari.
Actor:	Usuari.
Precondició:	1. Digital Assets Libraries instal·lats: (a) **building**Engine (b) attribs 2. L'usuari ha de tenir un edifici obert en Houdini i sense errors.
Flux Principal:	1. Clicar la pestanya "Interaction" del shell d'eines 2. Clicar sobre el botó de "Preprocess"
Flux Alternatiu	Cap.
Postcondició	S'han generat totes les imatges transcodificades en format PNG i s'han desat en una carpeta auxiliar dins del la carpeta del projecte.

8.3.2.2 Diagrama i fitxa de Activar Mode Ràpid

Aquest cas mostra l'activació del mode ràpid per tal d'intercanviar geometria per imatges pre-calculades. D'aquesta manera l'usuari pot interactuar més ràpida i fluidament amb l'edifici amb el que està treballant.

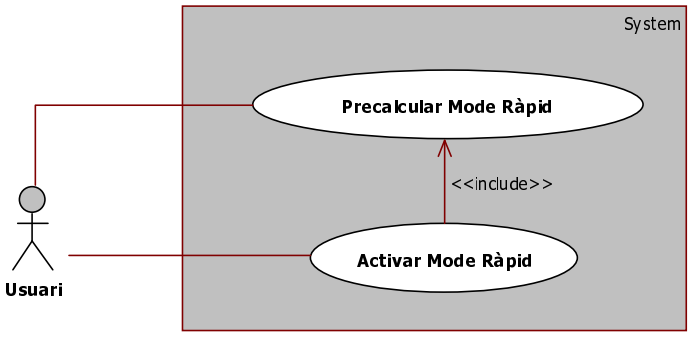


Figura 8.6: Diagrama de cas d'us d'Activar Mode Ràpid.

Fitxa de cas d'ús:	Activar Mode Ràpid
Descripció:	Intercanvia tota la geometria de l'edifici per textures pre-calculades que contenen el dibuix bidimensional de la mateixa geometria
Actor:	Usuari.
Precondició:	1. Digital Assets Libraries instal·lats: (a) **building**Engine (b) attribs 2. L'usuari ha de tenir un edifici obert en Houdini i sense errors. 3. L'usuari ha d'haver pre-calculat l'edifici. (Cas d'us de la Figura 8.5)
Flux Principal:	1. Clicar la pestanya "Interaction" del shell d'eines 2. Clicar sobre el botó de "Fast ON"
Flux Alternatiu	Cap.
Postcondició	L'edifici es mostra amb tota la geometria intercanviada per textures que reemplacen cadascun dels Inserts que contenien geometria.

8.3.2.3 Diagrama i fitxa de Desactivar Mode Ràpid

Aquesta funcionalitat permet desactivar el mode d'acceleració de la interactivitat per tal que l'usuari torni a visualitzar l'edifici amb la seva geometria original. D'aquesta manera pot tornar a realitzar les operacions que requereixin tenir la geometria original mitjançant Houdini, tot i que es perd la interactivitat accelerada.

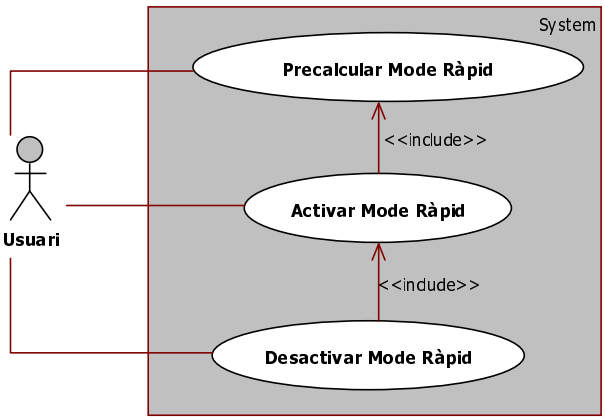


Figura 8.7: Diagrama de cas d'us de Desactivar Mode Ràpid.

Fitxa de cas d'ús:	Desactivar Mode Ràpid
Descripció:	Desactiva el mode ràpid tornant a intercanviar totes les textures que reemplacen la geometria per la geometria original perdent l'acceleració del mode ràpid fins a tornar a ser activat.
Actor:	Usuari.
Precondició:	1. Digital Assets Libraries instal·lats: (a) **building**Engine (b) attribs 2. L'usuari ha de tenir un edifici obert en Houdini i sense errors. 3. L'usuari ha d'haver activat prèviament el mode ràpid. (Cas d'us de la Figura 8.6)
Flux Principal:	1. Clicar la pestanya "Interaction" del shell d'eines 2. Clicar sobre el botó de "Fast Off"
Flux Alternatiu	Cap.
Postcondició	Es mostra l'edifici tornant a substituir les textures per la geometria corresponent i amb la interactivitat normal.

8.3.2.4 Diagrama i fitxa de Seleccionar Geometria

Aquest cas mostra com l'usuari realitza la selecció del conjunt de geometria que desitja replicar mitjançant la interfície creada per a realitzar aquesta tasca.

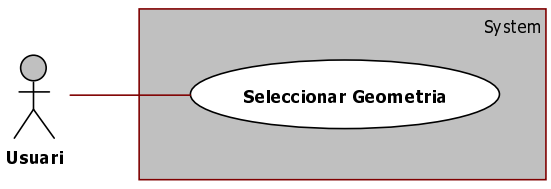


Figura 8.8: Diagrama de cas d'us de Seleccionar Geometria.

Fitxa de cas d'ús:	Seleccionar Geometria
Descripció:	L'usuari selecciona la geometria que desitja copiar.
Actor:	Usuari.
Precondició:	1. Digital Assets Libraries instal·lats: (a) **building**Engine (b) attribs 2. L'usuari ha de tenir un edifici obert en Houdini i sense errors.
Flux Principal:	1. Clicar la pestanya "Interaction" del shell d'eines. 2. Clicar el botó de "Copy&Paste". 3. En el menú de "GUI" clicar el botó "Select Geometry". 4. Mitjançant el ratolí, clicar per seleccionar una primitiva o clicar i arrossegar sense deixar el botó del ratolí per a fer una selecció múltiple. 5. Amb el ratolí situat encara a dins de l'àrea de renderització de l'edifici, prémer la tecla "Enter".
Flux Alternatiu	Si es prem "Enter" i no s'ha seleccionat geometria, emet un error.
Postcondició	L'usuari ha seleccionat la geometria que desitja i s'ha desat internament la primitiva ancestre comú de totes les primitives que l'usuari ha seleccionat.

8.3.2.5 Diagrama i fitxa de Copiar Geometria Seleccionada

Aquesta funcionalitat permet a l'usuari copiar tota aquella geometria, i les regles corresponents, que depengui de la selecció efectuada prèviament i desar-la temporalment en el buffer de Houdini, o en un fitxer en format XML per a la seva futura utilització i re-utilització.

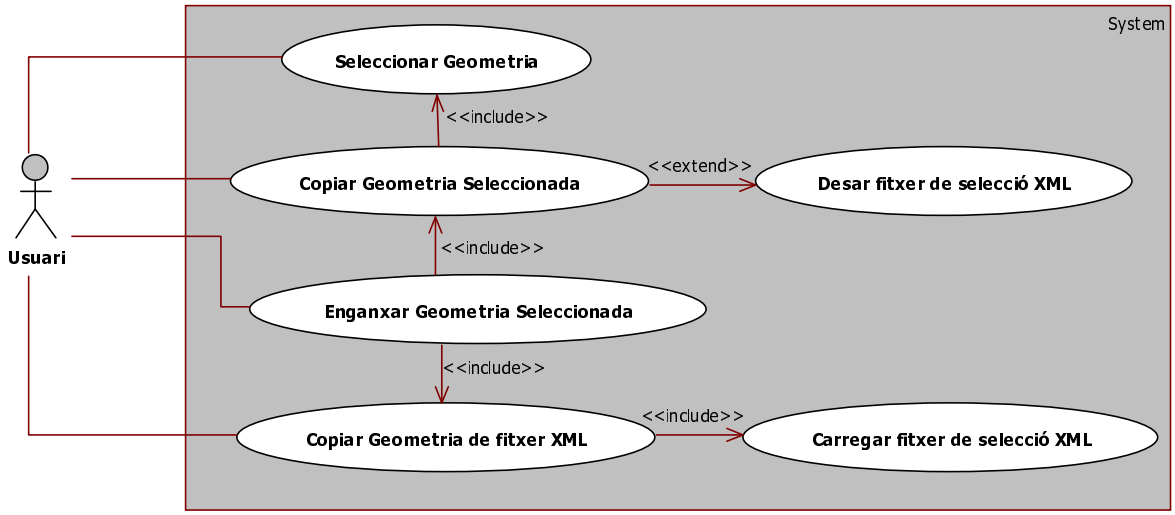


Figura 8.9: Diagrama de cas d'us de Copiar Geometria Seleccionada.

Fitxa de cas d'ús:	Copiar Geometria Seleccionada
Descripció:	L'usuari realitza una còpia de la geometria que depèn de la selecció prèvia amb la opció de desar-ho en format XML.
Actor:	Usuari.
Precondició:	1. Digital Assets Libraries instal·lats: (a) **building**Engine (b) attribs 2. L'usuari ha de tenir un edifici obert en Houdini i sense errors. 3. L'usuari ha d'haver seleccionat prèviament geometria amb el cas d'us de la Figura 8.8
Flux Principal:	1. Clicar la pestanya "Interaction" del shell d'eines. 2. Clicar el botó de "Copy&Paste". 3. En el menú de "GUI" clicar el botó "Copy Selection".
Flux Alternatiu	3. Clicar en el botó de selecció de fitxer del camp "Save Copy to File". 4. Navegar per l'arbre de directoris del sistema operatiu per trobar allà on es vol desar la còpia. 5. Introduir el nom del nou fitxer (o seleccionar-ne un d'existent per re-escriure'l). 6. Acceptar i desar els canvis o cancel·lar i tornar al pas 1.
Postcondició	L'usuari té la geometria seleccionada desada en el buffer de Houdini i opcionalment també en un fitxer amb format XML.

8.3.2.6 Diagrama i fitxa de Copiar Geometria des d'un fitxer XML

Aquest cas d'us il·lustra com l'usuari efectua l'acció de copiar la geometria a partir d'un fitxer en format XML d'una còpia prèvia. D'aquesta manera port re-aprofitar diverses vegades una mateixa còpia tot i treballant en dies diferents i tot i havent tancat Houdini.

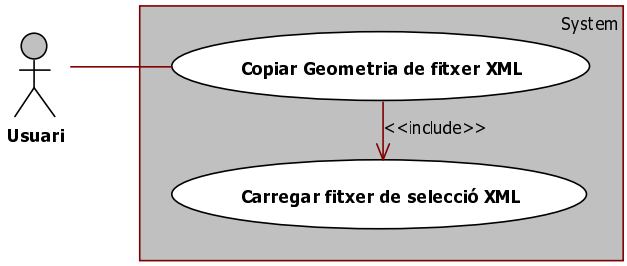


Figura 8.10: Diagrama de cas d'us de Copiar Geometria des d'un fitxer XML.

Fitxa de cas d'ús:	Copiar Geometria des d'un fitxer XML
Descripció:	Carrega en el buffer de Houdini la geometria d'una còpia anterior desada en un fitxer en format XML.
Actor:	Usuari.
Precondició:	1. Digital Assets Libraries instal·lats: (a) **building**Engine (b) attribs 2. L'usuari ha de tenir un edifici obert en Houdini i sense errors.
Flux Principal:	1. Clicar la pestanya "Interaction" del shell d'eines. 2. Clicar el botó de "Copy&Paste". 3. Clicar en el botó de selecció de fitxer del camp "Load Copy File". 4. Navegar per l'arbre de directoris del sistema operatiu per trobar el fitxer de còpia que es vol carregar. 5. Acceptar i desar els canvis o cancel·lar i tornar al pas 1.
Flux Alternatiu	Cap.
Postcondició	L'usuari té la geometria seleccionada del fitxer, desada en el buffer de Houdini a punt per ser enganxada.

8.3.2.7 Diagrama i fitxa d'Enganxar Geometria Seleccionada

Aquesta funcionalitat permet a l'usuari enganxar la geometria copiada prèviament en l'edifici. Per tal de poder escollir allà on vol enganxar-la, abans ha de fer una selecció. D'aquesta manera la plataforma sap on ha d'enganxar la geometria i les regles corresponents.

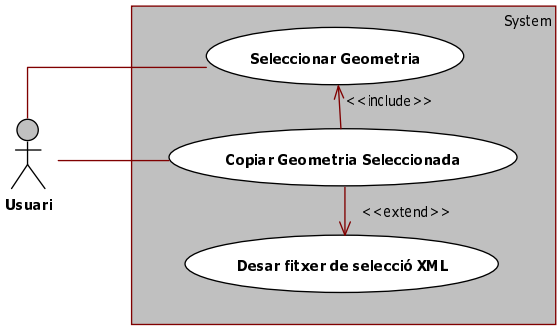


Figura 8.11: Diagrama de cas d'us d'Enganxar Geometria Seleccionada.

Fitxa de cas d'ús:	Enganxar Geometria Seleccionada
Descripció:	Permet enganxar la geometria copiada anteriorment allà on l'usuari desitja.
Actor:	Usuari.
Precondició:	1. Digital Assets Libraries instal·lats: (a) **building**Engine (b) attribs 2. L'usuari ha de tenir un edifici obert en Houdini i sense errors. 3. Realitzar una selecció de geometria, amb tots els seus passos, tal i com en el cas d'ús de la Figura 8.8
Flux Principal:	1. Clicar la pestanya "Interaction" del shell d'eines. 2. Clicar el botó de "Copy&Paste". 3. Clicar en el botó de selecció de fitxer del camp "Paste Selection".
Flux Alternatiu	Cap.
Postcondició	La geometria carregada s'ha enganxat allà on ha seleccionat l'usuari adaptant les seves proporcions a les del nou edifici.

8.4 Diagrama de classes

Un diagrama de classes és un tipus de diagrama d'estructura estàtica que descriu l'estructura d'un sistema, mostrant les classes del sistema, els atributs i les relacions entre elles.

8.4.1 Arquitectura del disseny

S'ha optat per fer una arquitectura Model-View-Controller ja que d'aquesta manera el codi queda més elegant i entenedor, per si en un futur es segueix amb el projecte.

L'arquitectura MVC (Model-Vista-Controlador), es tracta el Model, les Vistes i els Controladors com a entitats separades. Això aconsegueix reduir l'esforç de programació i el manteniment de l'aplicació.

La definició dels diferents components del model MVC és la següent:

- **Model:** és l'objecte que representa les dades del programa, encarregat de manipular-les i controlar tots els canvis.

- **Vista:** és l'objecte encarregat de la presentació visual de les dades representades pel Model, per tant que l'usuari pot interactuar-hi.

- **Controlador:** és l'objecte que actua sobre les dades representades del Model segons les ordres donades per l'usuari.

El model que té aquest projecte és el conjunt de nodes, primitives i regles que generen la geometria que l'usuari desitja modificar. Per tant, les classes que intervenen amb el model són principalment tres: transform2textures (per a l'acceleració), copyPasteOps (per operacions de Copy&Paste) i CopyPaste_XMLMannager (Per lectures i escriptures de còpies a XML).

8.4.2 Diagrama de classes modular

El diagrama de la Figura 8.12 es poden observar quatre seccions generals ben diferenciades. Per una banda es pot observar el mòdul de la interfície d'usuari (verd) que fa servir tant el mòdul d'acceleració d'interacció de l'usuari (taronja) com el mòdul de Copy&Paste (Blau). Aquest últim mòdul fa servir el mòdul de XML (groc) per a poder desar les copies realitzades en fitxers d'aquest format i poder-los recuperar.

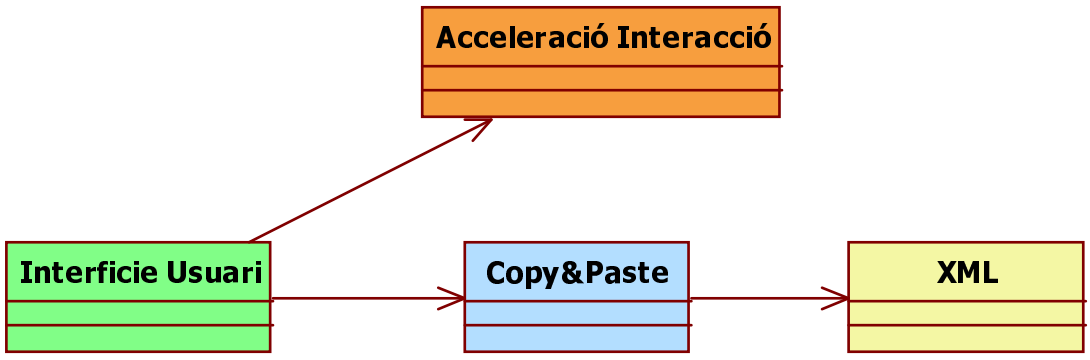


Figura 8.12: Diagrama de classes modular.

Cadascun dels mòduls esta compost per un conjunt de classes, les quals es llisten a continuació:

- Mòdul Interfície Usuari:
  - Shelf
  - GUI
- Mòdul Copy&Paste:
  - controller
  - copyPasteOps
  - commonPrimAncestor
  - commonNodeAncestor
  - selectPrim
  - houTopologicalSort
  - HouPrimOps
  - bE_sistematicNodeVisitors
  - productTree
  - bE_general
  - houPrimGroups
- Mòdul Acceleració Interacció:
  - transform2textures
  - graphRewriting
  - SVGFileWriter
- Mòdul XML:
  - CopyPaste_XMLMannager
  - XMLShapeFileReader
  - XMLShapeDump
  - DirectDumper
  - XMLShapeDumper
  - XMLShapeFileWriter

Com s'ha explicat anteriorment, el disseny segueix la filosofia Model-View-Controller. El mòdul d'Interfície d'usuari (verd) junt amb el d'Acceleració d'Interacció (taronja) pertanyen a la part de View de la filosofia. El pes més gran de la part de Controller la té la classe Copy&Paste (Blau) i finalment la part de Model s'encarrega sobretot el mòdul de XML (groc).

8.4.3 Diagrama de classes modular estès

En aquest diagrama es pot observar més clarament la distribució de les classes abans esmentades per cada mòdul principal. En la Figura 8.13 es pot observar com s'ha seguit el disseny Model-View-Controller per a realitzar aquest projecte. La classe *Shelf* s'ha d'esmentar particularment, ja que no existeix com a classe de per si, si no que es tracta d'una eina de Houdini per a realitzar interfícies gràfiques, de tal manera que pren alhora el paper tant de "controller" com "view" al usar la classe *transform2textures*.

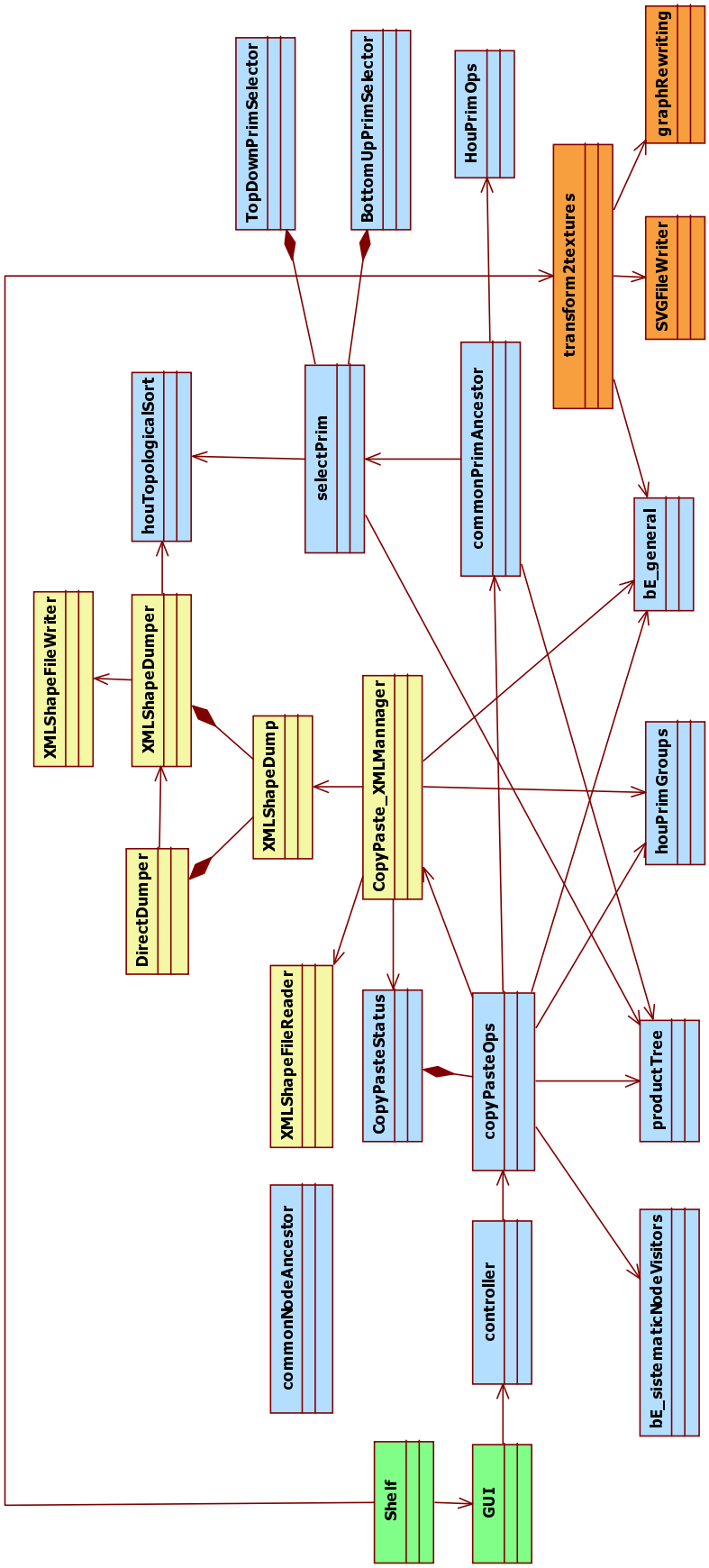


Figura 8.13: Diagrama de classes modular estès.

8.4.4 Diagrama de classes complet

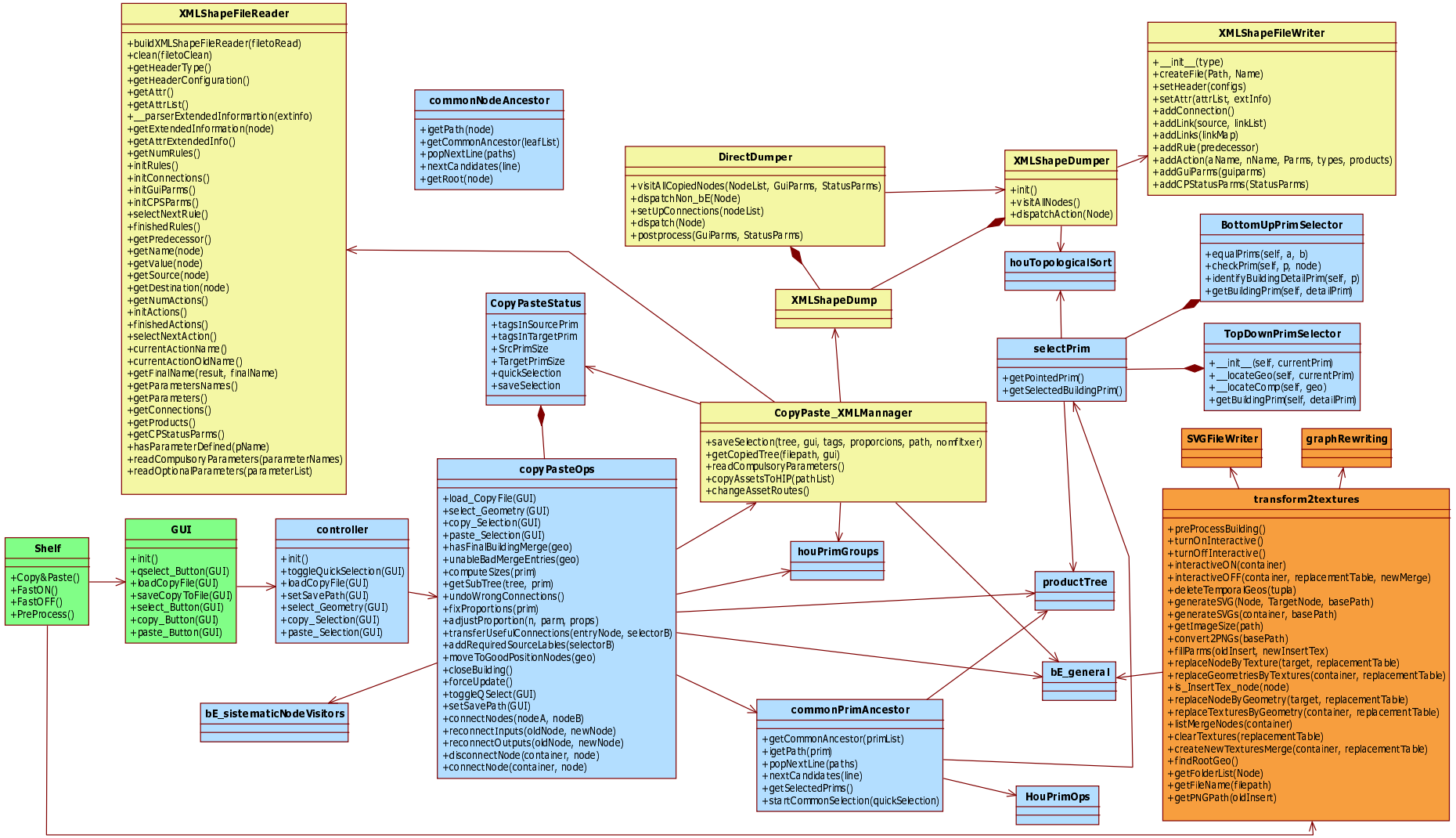


Figura 8.14: Diagrama de classes complet.

Com es pot veure en la Figura 8.14 del diagrama de classes complet, només s'han esmentat els mètodes implementats per a aquest projecte de manera que es pugui distingir entre les classes i funcionalitats existents i les pròpies d'aquest projecte. Les classes que no contenen mètodes són classes realitzades en altres projectes i que ara són part del conjunt de classes de **building**Engine. Tot i això, s'hi han afegit al diagrama ja que són importants en el desenvolupament de les funcionalitats creades en aquest projecte.

A continuació s'anirà explicant cadascun dels mòduls, classe per classe, descrivint tots els mètodes i funcionalitats.

8.4.4.1 Mòdul de View

Aquest mòdul està format principalment per dues classes: la "GUI" i la "Shelf", que com s'ha dit abans, aquesta última es representa com a una classe però no existeix com a tal, si no que és part de la configuració de Houdini. Aquesta configuració es desa en un fitxer i es pot carregar sense necessitat de fer-ho a mà.

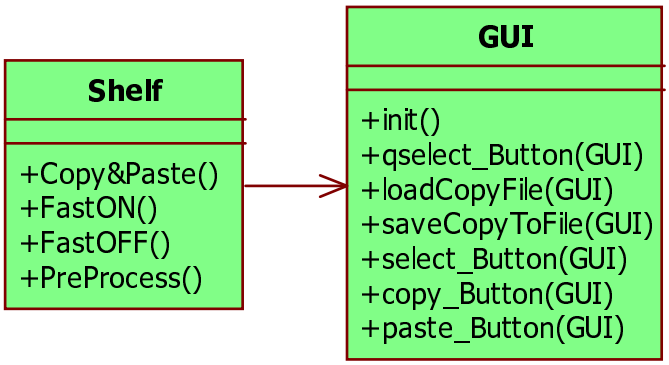


Figura 8.15: Diagrama de classes del mòdul del view.

Classe Shelf:

Aquesta classe realment no és una classe tot i que se la representi com a tal. Representa la interfície d'usuari que conté els botons per a poder realitzar les accions que es descriuen a continuació. Aquesta classe recull els events de ratolí i s'encarrega de cridar a les funcions pertinents de la classe GUI que s'explicarà més endavant. Es pot observar el seu diagrama en la Figura 8.16.

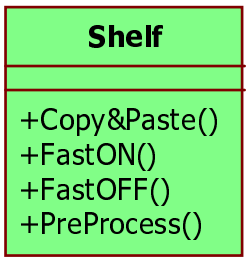


Figura 8.16: Diagrama de la classe Shelf.

- **Copy&Paste:** Mètode que crea el node "GUI" instanciant a la classe GUI.py per tal que l'usuari tingui accés a les opcions de copiar i enganxar geometria.
- **FastON:** Mètode que permet activar el mode d'acceleració de la interacció d'usuari.
- **FastOFF:** Mètode que permet desactivar el mode d'acceleració de la interacció d'usuari.
- **PreProcess:** Funció que realitza els càlculs necessaris per a poder activar i desactivar el mode d'acceleració de la interacció d'usuari.

Classe GUI:

Aquesta classe permet a l'usuari efectuar, mitjançant la interfície gràfica de Houdini, les operacions que intervenen en el procés del Copy&Paste. La interfície d'usuari la sustenta un nou node, el qual es crea al clicar sobre el botó de Copy&Paste de la classe "Shelf", i que conté com a paràmetres els botons per a facilitar la interacció del projecte a l'usuari. Es pot observar el seu diagrama en la Figura 8.17.

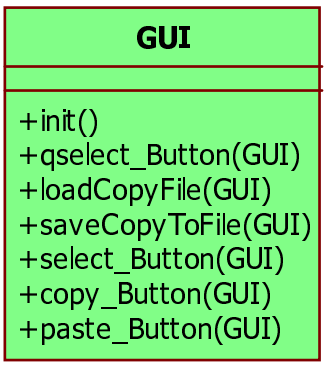


Figura 8.17: Diagrama de la classe GUI.

- **init:** S'encarrega de propagar la petició d'inicialització a la capa del controlador.
- **qselect_Button(GUI):** Canvia l'estat de la variable booleana del CopyPasteStatus que permet activar i desactivar la selecció ràpida (Per defecte està sempre activada).
- **loadCopyFile(GUI):** S'encarrega de propagar la petició la capa de controlador per tal que carregui un fitxer XML d'una còpia anteriorment desada.
- **saveCopyToFile(GUI):** S'encarrega de propagar la petició a la capa de controlador per tal que desi el path al fitxer on es vol desar la selecció actual.
- **select_Button(GUI):** S'encarrega de propagar la petició a la capa de controlador per tal que comenci a executar-se el "pipeline" de seleccionar geometria.
- **copy_Button(GUI):** S'encarrega de propagar la petició a la capa de controlador per tal que copi en el buffer la selecció i opcionalment si s'ha escollit desar la còpia en un fitxer XML.
- **paste_Button(GUI):** S'encarrega de propagar la petició a la capa de controlador per tal que enganxi, allà on s'ha seleccionat, la geometria copiada.

8.4.4.2 Mòdul de Copy&Paste

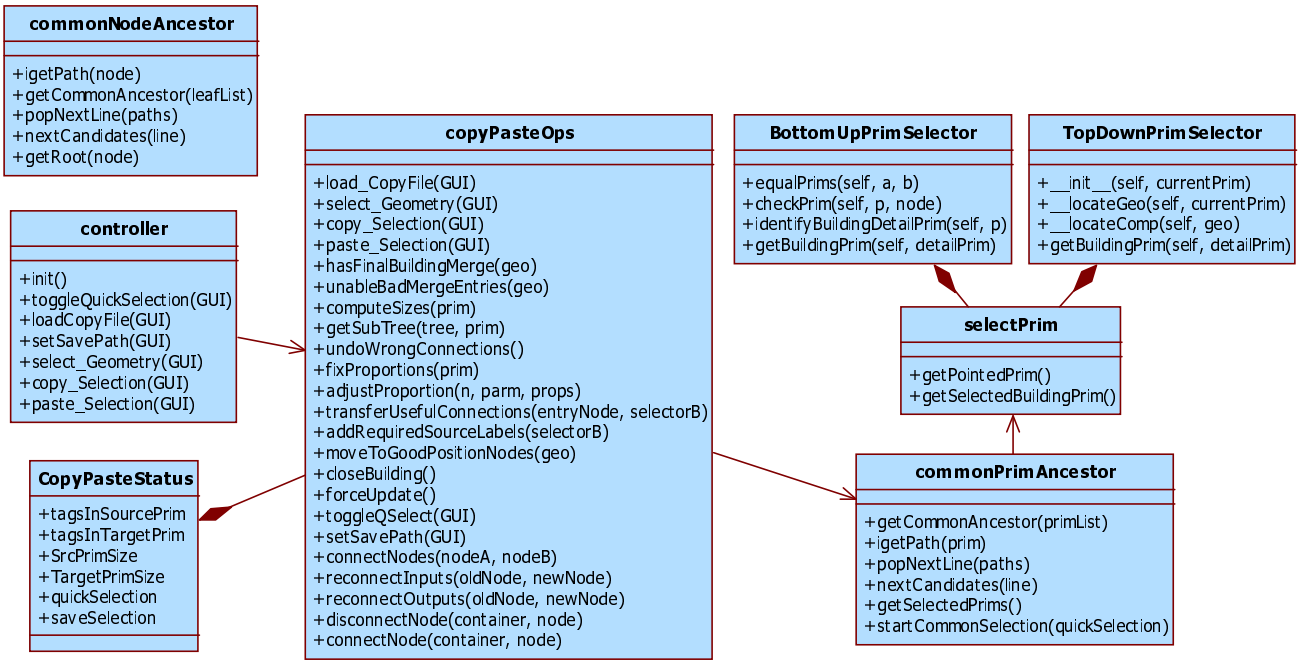


Figura 8.18: Diagrama de classes del mòdul del Copy&Paste.

Controller:

Aquesta classe és la que controla el transit de les peticions de Copy&Paste i s'encarrega de cridar a les classes corresponents segons les peticions que l'hi arriben. Es pot observar el seu diagrama en la Figura 8.19.

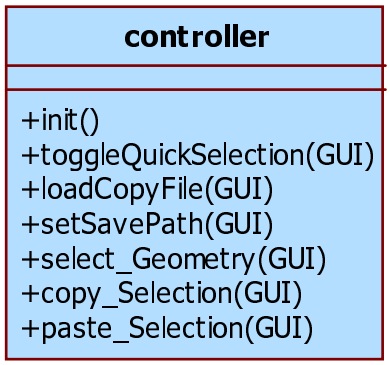


Figura 8.19: Diagrama de la classe Controller.

- **init():** S'encarrega de crear i inicialitzar el node "GUI" auxiliar creat en el path del projecte de Houdini *obj/Interface/GUI*.

- **toggleQuickSelection(GUI):** Canvia l'estat de la variable booleana que permet saber si la selecció ràpida està o no activada.
- **loadCopyFile(GUI):** S'encarrega de cridar al mètode per carregar un fitxer XML amb una selecció copiada anteriorment, a la classe copyPasteOps.
- **setSavePath(GUI):** S'encarrega de cridar al mètode per desar la ruta d'on es desarà el fitxer en format XML de la selecció realitzada, a la classe copyPasteOps.
- **select_Geometry(GUI):** S'encarrega de cridar al mètode per començar el "pipeline" de seleccionar geometria, a la classe copyPasteOps.
- **copy_Selection(GUI):** S'encarrega de cridar al mètode per copiar en el buffer (i opcionalment desar en fitxer XML) la selecció realitzada, a la classe copyPasteOps.
- **paste_Selection(GUI):** S'encarrega de cridar al mètode per enganxar la geometria i les regles copiades anteriorment, allà on desitgi l'usuari, a la classe copyPasteOps.

copyPasteOps:

Aquesta classe és l'encarregada d'efectuar totes aquelles operacions de seleccionar, copiar i enganxar del mòdul de Copy&Paste. Es pot observar el seu diagrama en la Figura 8.20.

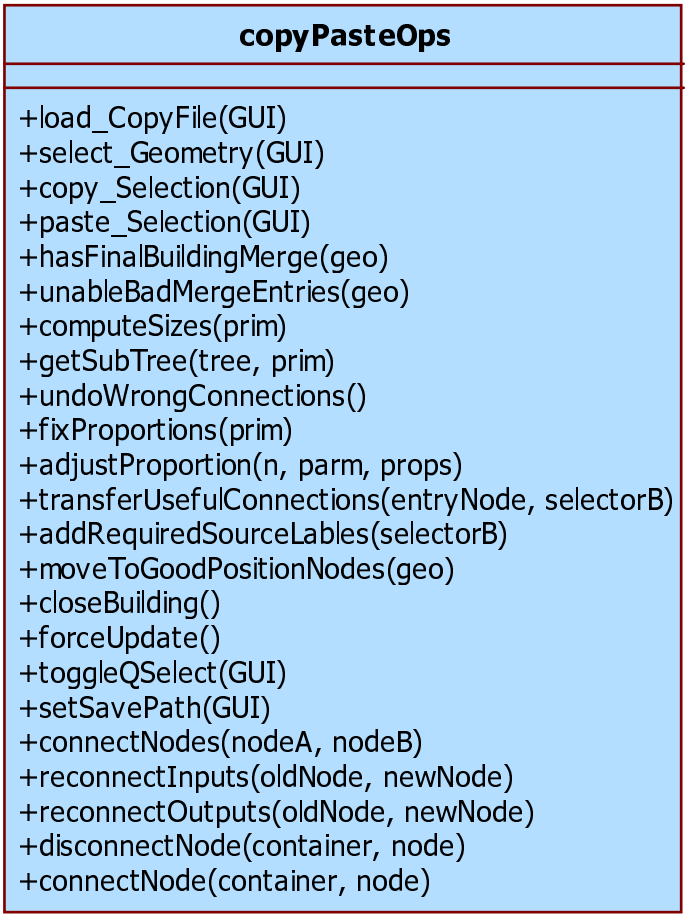


Figura 8.20: Diagrama de la classe CopyPasteOps.

- **load_CopyFile(GUI):** S'encarrega de comprovar que el fitxer XML escollit existeixi, pre-suposant que el format és correcte, i el carrega al buffer de Houdini amb l'estructura geomètrica que hi conté. També s'encarrega de copiar tots els *assests* dels models geomètrics que contenia l'antic edifici del qual prové la còpia, i re-nombrar tots els paths dels nodes *Insert* amb aquests nous assests copiats.

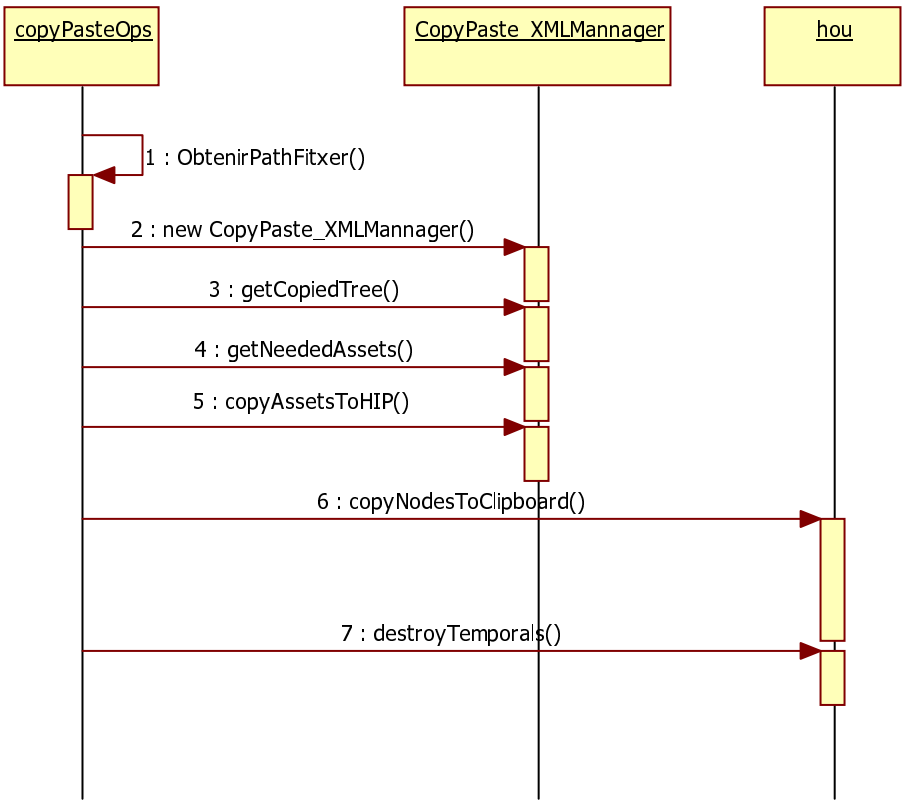


Figura 8.21: Exemple d'ús de la funció load_CopyFile(GUI).

- **select_Geometry(GUI):** Permet posar a Houdini en mode de selecció de geometria i queda a l'espera que l'usuari efectuï la selecció. Un cop ha fet la selecció, s'encarrega de cercar la primitiva ancestre comuna que genera a totes les primitives seleccionades, i desa aquesta informació i d'altre, en la classe CopyPasteStatus. Es pot veure el pseudocodi en la Figura 8.22.

```
METODE select_Geometry(Node:GUI) RETORNA Res
  obj_childs = ObtenirTotsElsNodes('/obj')
  //Cerca el node geo, només n'hi pot haver un
  PER_CADA node DINS obj_childs FER
    SI node.tipus() == 'geo' FER
      geo = node
      ACABAR
    FSI
  FPER
  #Cerca el primer node merge a dins del geo
  PER_CADA node DINS ObtenirFills(geo) FER
    SI node.tipus() == 'merge' o
      node.tipus() == 'object_merge' FER
      merge = node
      ACABAR
    FSI
  FPER
  SI merge<>None FER
    ActivarVisualització(merge)
    ancestre_comú = commonPrimAncestor.ComençarPipelineSelecció()
    SI ancestre_comú <> None FER
      //S'omplen els paràmetres del GUI amb la informació
      //que s'extreu de l'ancestre comú Tals com:
      //número de primitiva, node actual i el geo que sustenta
      //l'ancestre comú.
      GUI.setParametres(ancestre_comú)
    ALTRAMENT
      Emetre Error
    FSI
  FSI
FMETODE
```

Figura 8.22: Pseudocodi de la funció select_Geometry(GUI).

- **copy_Selection(GUI):** S'encarrega de seleccionar tot el sub-arbre de primitives de la primitiva ancestre comuna seleccionada anteriorment i de realitzar les marques en el node arrel, indispensables per a la seva posterior localització. El mètode ho deixa tot carregat en el buffer a punt per ser enganxat.

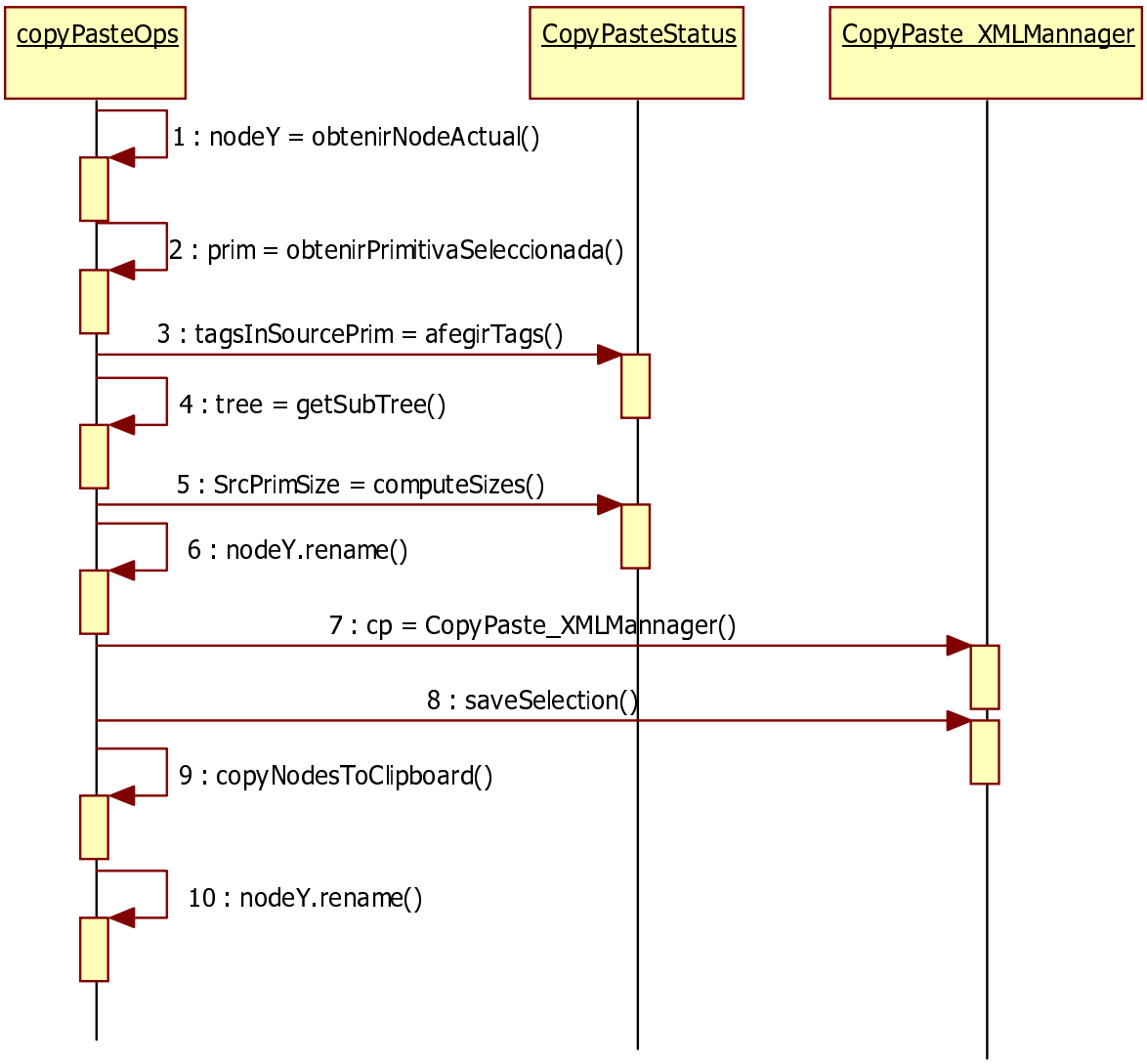


Figura 8.23: Exemple d'ús de la funció copy_Selection(GUI).

- **paste_Selection(GUI):** S'encarrega de realitzar l'enganxat de la geometria copiada. Per a dur a terme aquesta tasca primer de tot s'ha d'obtenir la primitiva seleccionada. Aquesta primitiva indica on s'ha de realitzar l'enganxat. Un cop s'ha obtingut es comença el procés creant tres nodes, un d'excepció i dos selectors, que serveixen per dirigir la part de l'edifici que s'ha de modificar de la resta de l'edifici. Un cop feta aquesta separació lògica, s'han d'incorporar a la part de l'arbre de primitives que s'ha de modificar, el conjunt de tags per tal que el sub-arbre que s'hi enganxi els hereti, i d'aquesta manera processi les regles correctament. Un cop fet això es realitza la connexió de tots aquests nodes, en el selectorA s'hi enganxa el sub-arbre de les regles successores de la primitiva allà on l'usuari vol efectuar l'enganxat. En el selectorB s'hi connecta el sub-arbre de les regles que són pròpies de la selecció que l'usuari desitja enganxar. Un cop efectuada aquesta tasca s'ha de comprovar que no hi hagin males connexions. Això és degut a que al realitzar operacions sobre nodes, si un node té el mateix nom que un altre que ja existeix, Houdini connecta els seus inputs amb els inputs del node que té el mateix nom i que ja existia. Això pot provocar errors que no es desitgen i per tant s'ha de verificar que no existeixin connexions produïdes per efectes laterals de les operacions realitzades en els nodes. Ja que el sub-arbre copiat ha estat bolcat primerament en una carpeta temporal, arribat aquest punt s'elimina per tal de mantenir net el projecte. Ara que es té l'edifici connectat amb la geometria enganxada i tot en ordre, s'ha de proporcionar geomètricament la geometria que s'ha enganxat, ja que pot ser l'edifici origen no té les mateixes dimensions que l'edifici destí. Un cop proporcionat, es reordenen els nodes visualment i es genera un nou node 'Merge' per tal de visualitzar tots els canvis. Finalment es recorre l'arbre de primitives per tal de refrescar cadascun dels nodes, ja que Houdini no sempre mostra el resultat actualitzat, de manera que se l'hi obliga a fer-ho. Per tal que l'usuari pugui efectuar una nova còpia o càrrega i torni a efectuar tot el procés, es netegen els caps del node 'GUI' per tal que es pugui tornar a fer servir sense les dades antigues.

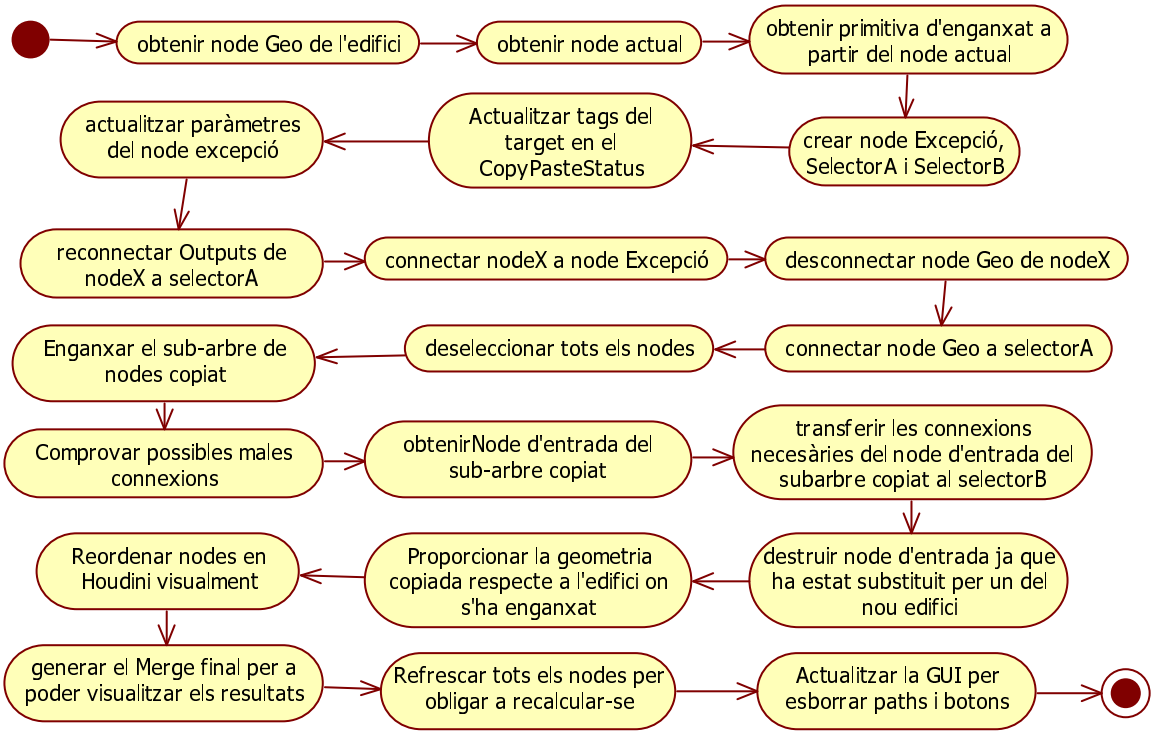


Figura 8.24: Exemple d'ús de la funció paste_Selection(GUI).

- **computeSizes(prim):** Realitza i retorna el càlcul de les mides de la primitiva 'prim' per a tenir més tard noció de la seva proporció.
- **getSubTree(tree, prim):** Donada una primitiva 'prim', desa a 'tree' tots els nodes que processa a algun dels seus descendents.
- **undoWrongConnections():** Desconnecta totes les possibles connexions errònies que poden resultar de la operació de còpia, ja que Houdini reacciona automàticament sota algunes operacions afegint connexions que no són desitjades.

```
METODE undoWrongConnections() RETORNA Res
  PER n DINS hou.selectedNodes() FER
    PER connexió DINS n.inputConnections() FER
      NodeFill := connexió.inputNode()
      SI NodeFill no està seleccionat LLAVORS
        n.setInput(connexió.inputIndex(),None)
      FSI
    FPER
  FPER
FMETODE
```

Figura 8.25: Pseudocodi de la funció undoWrongConnections().

- **fixProportions(prim):** Proporciona la primitiva 'prim' respecte la primitiva on es vol realitzar l'enganxat, de manera que la proporció estètica de l'origen es correspongui amb el destí.
- **adjustProportion(n,parm,props):** Efectua l'ajust de les proporcions dels paràmetres 'parm' del node 'n' amb les proporcions calculades prèviament 'props'.
- **transferUsefulConnections(entryNode, selectorB):** Per tots els descendents de 'entryNode', desa tots aquells 'filter' que també es troben en els tags desats en el CopyPasteStatus i que per tant requereixen ser desats per al correcte funcionament.
- **addRequiredSourceLabels(selectorB):** Actualiza les etiquetes dels nodes amb les del sub-arbre copiat per tal que al enganxar-lo tots els nodes heretin aquestes etiquetes d'entrada i funcioni correctament. Aquestes etiquetes estan compostes per 'filters' (l'entrada que reben els nodes) i 'products' (la sortida que generen). Interessa el conjunt d'aquelles etiquetes de tots els nodes que es trobin seleccionats que siguin tant 'products' com 'filters' i que no es trobin ja en l'edifici on es vulgui enganxar el conjunt de regles.
- **moveToGoodPositionNodes(geo):** Mou els nodes un cop enganxats per tal que es re-col·loquin de manera una mica més ordenada.
- **closeBuilding():** S'encarrega de recorre tot l'arbre de nodes final per crear un nou node 'Object_Merge' i que es pugui visualitzar el resultat final de l'edifici. Això es fa ja que el merge que visualitzava abans l'escena, no conté els nous nodes del sub-arbre enganxat. Per això es crea un nou node merge, que agrupa tot allò que es veia abans afegint-hi els nodes enganxats per a que tot es visualitzi correctament.
- **forceUpdate():** Actualitza tots els nodes per tal que es re-calculin segons els seus inputs, outputs i tags i aixi assegurar que s'han efectuat tots els canvis.

- **toggleQSelect():** Permet alterar l'estat de la variable booleana que desa si la selecció ràpida està o no activada.
- **setSavePath():** Desa en el CopyPasteStatus el path de la carpeta temporal on es farà la càrrega del sub-arbre a partir d'un fitxer XML.
- **connectNodes(nodeA, nodeB):** Connecta dos nodes de manera que queda "nodeA" com a pare i "nodeB" com a fill. Es pot veure com funciona a la Figura 8.26

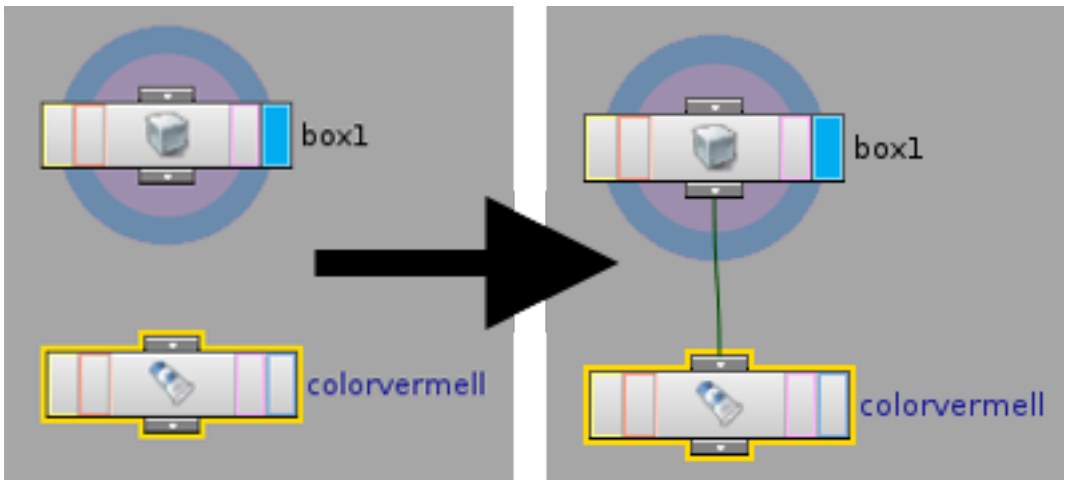


Figura 8.26: Exemple d'ús de la funció connectNodes.

- **reconnectInputs(oldNode, newNode):** Reconnecta els inputs del node "oldNode" al node "newNode". Si s'observa la Figura 8.27 es pot veure el funcionament del mètode sobre el node "colorvermell" i "colorblau" i la repercussió sobre l'objecte "box".

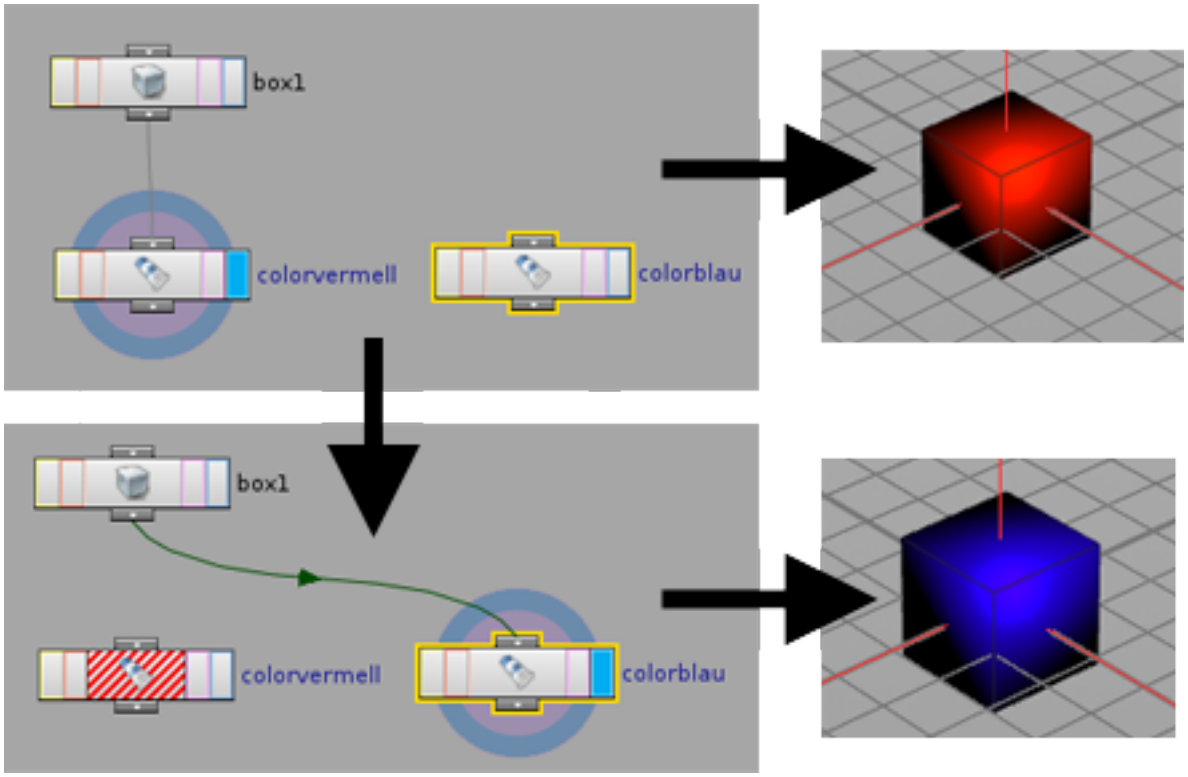


Figura 8.27: Exemple d'ús de la funció reconnectInputs.

- **reconnectOutputs(oldNode,newNode):** Reconecta els outputs del node "oldNode" al node "newNode"
- **disconnectNode(container,node):** Per a tots els fills de "container", si són del tipus "object_merge" i el paràmetre "objpath" és el path del node "node", a les hores és desconnectat.
- **connectNode(container,node):** Per a tots els fills de "container" afegeix el node "node" com a les seves entrades per tal que sigui visualitzat.

commonPrimAncestor:

Aquesta classe és l'encarregada de cercar, donat un conjunt de primitives, aquella primitiva que és comuna a totes les donades. És a dir, aquella que les seves regles successores sustentin a totes les primitives que s'han donat d'entrada. La primitiva que es troba és la més propera, és a dir, la més allunyada de l'arrel possible, sempre que compleixi la condició explicada anteriorment.

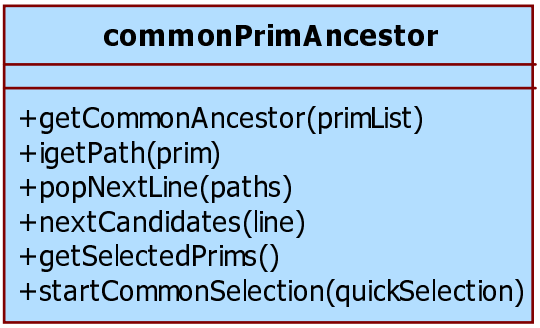


Figura 8.28: Diagrama de la classe commonPrimAncestor.

- **getCommonAncestor(primList):** Cerca dins de totes les llistes contingudes en "primList" quin és l'ancestre comú més proper. En el pitjor cas, l'ancestre comú serà el node arrel. Retorna aquest ancestre comú trobat.
- **igetPath(prim):** Retorna recursivament la llista de nodes que tracen el camí des de "prim" fins al node arrel, dins de l'arbre de primitives.
- **popNextLine(paths):** Retorna tots els tops de la llista de paths com si fos una pila per a poder comprovar si és ancestre comú o no.
- **nextCandidates(line):** Retorna el següent candidat a ser ancestre comú en forma de llista.
- **getSelectedPrims():** Posa l'entorn de Houdini en mode de selecció de geometria i es queda a l'espera a que l'usuari seleccioni geometria. Si no selecciona res, simplement emet un error, en cas contrari realitza una optimització en el nombre de primitives seleccionades per tal de fer la selecció més ràpida i retorna la llista resultant. Aquesta optimització es basa en que al realitzar la selecció moltes vegades es seleccionen primitives consecutives. La qual cosa vol dir que pertanyen a un mateix 'Insert' i que per tant no aporten informació nova en quant a la selecció. Per tal de minimitzar el cost del càlcul amb les primitives,

d'aquests conjunts consecutius només se n'extreu una d'elles, ja que per a qualsevol altre del conjunt s'obtindria el mateix node 'Insert'. Aquesta cerca del node 'Insert' es fa en la funció TopDownPrimSelector explicada més endavant. En la Figura 8.29 es pot veure el diagrama d'activitat d'aquesta funció.

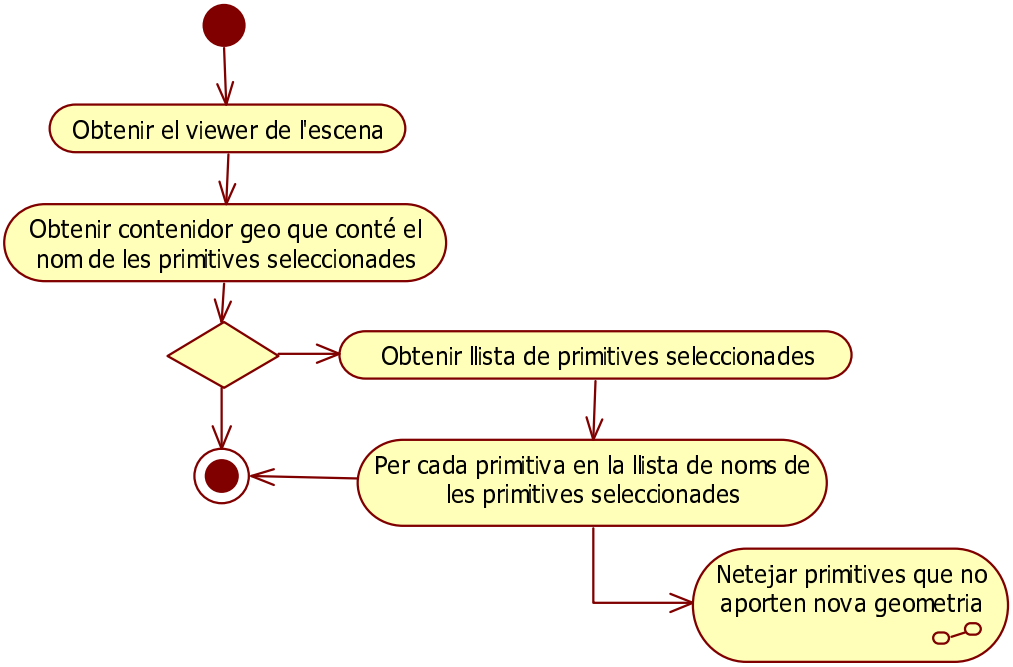


Figura 8.29: Exemple d'ús de la funció getSelectedPrims().

- **startCommonSelection(quickSelection):** Crida a la funció "getSelectedPrims" i un cop té la llista de primitives seleccionades, realitza una cerca per tal de trobar tots els identificadors de primitives respecte el node que les genera. Un cop trobats tos els identificadors, es cerca aquella primitiva que és comuna a totes. Aquesta serà la primitiva arrel del sub-arbre de primitives que representa la copia que pot realitza l'usuari més endavant. Es pot veure el seu diagrama de seqüència en la Figura 8.30

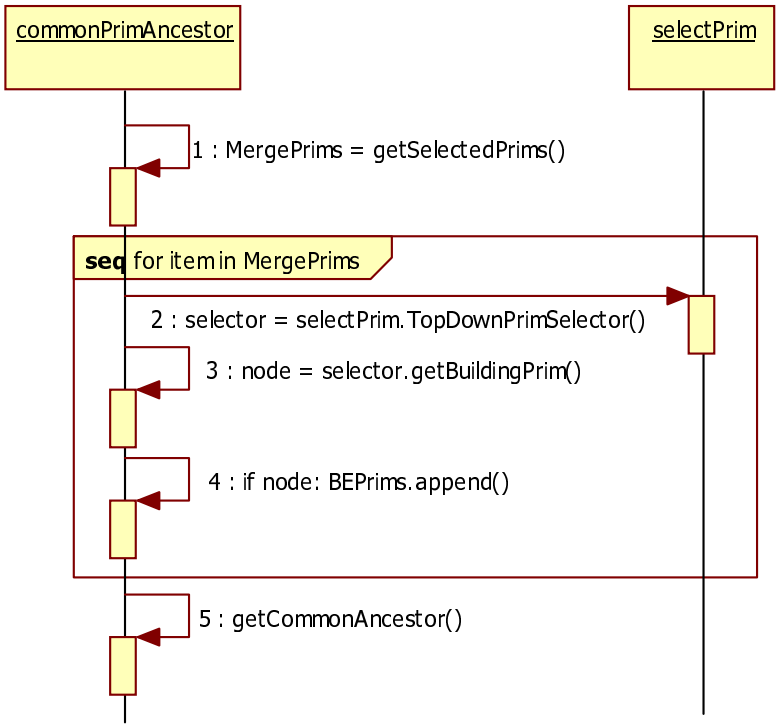


Figura 8.30: Exemple d'ús de la funció startCommonSelection(quickSelection).

Per tal d'il·lustrar millor aquest mètode, a la Figura 8.31 es pot veure com, si l'usuari selecciona dues finestres i la planta baixa (color vermell), l'algorisme retornaria la primitiva de la façana (color verd) com a resultat.

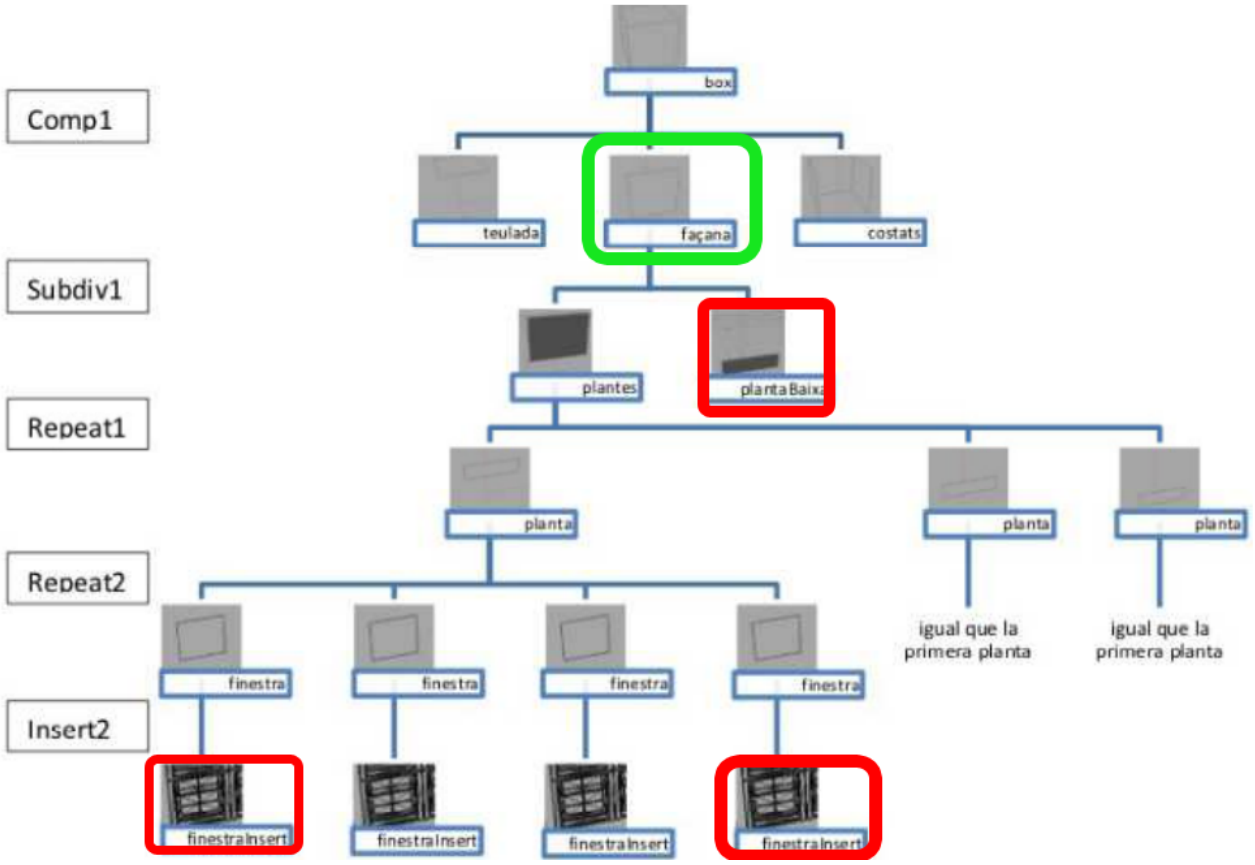


Figura 8.31: Exemple del funcionament de mètode startCommonSelection(quickSelection).

selectPrim

Aquesta classe conté els mètodes principals per a poder cridar les funcionalitats de les classes BottomUpPrimSelector i TopDownPrimSelector explicades més endavant.

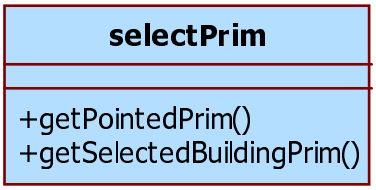


Figura 8.32: Diagrama de la classe selectPrim.

- **getPointedPrim():** Retorna una llista amb totes les primitives seleccionades per l'usuari.
- **getSelectedBuildPrim():** Crida al mètode getPointedPrim() i retorna la primitiva seleccionada de **building**Engine corresponent.

BottomUpPrimSelector:

Aquesta classe conté els mètodes per a poder cercar els nodes que generen les primitives que se l'hi passin. L'algorisme fa una cerca des dels nodes fulla fins a trobar el node que sustenta la

primitiva, la qual cosa el fa més costós que no pas el mètode de la classe TopDownPrimSelector explicada més endavant. Tot i això existeixen casos en que pot ser interessant i s'ha conservat com a una eina més del projecte. Es pot observar el seu diagrama en la Figura 8.33.

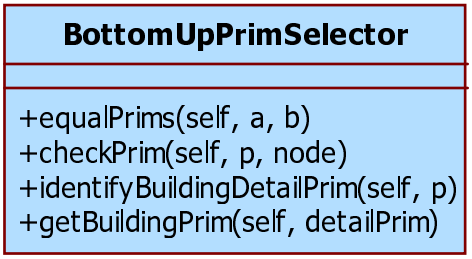


Figura 8.33: Diagrama de la classe BottomUpPrimSelector.

- **equalPrims(self, a, b):** Retorna cert si la primitiva a és igual a la primitiva "b" en termes de vèrtexs.

- **checkPrim(self, p, node):** Retorna el número de la primitiva respecte el node 'node' en cas que el contingui, altrament retorna None.

- **identifyBuildingDetailPrim(self, p):** Cerca i retorna la primitiva "p" però respecte al node que la sustenta i crea, no respecte el merge que la mostra. Houdini etiqueta les primitives de manera que si en selecciones una, es pot saber quin número de primitiva és. Aquest número, però, és consecutiu respecte totes les primitives de l'edifici, la qual cosa la fa difícil de tractar. Per a poder millorar-ho, es cerca el número de primitiva però en comptes de ser respecte tota l'escena, només és respecte el node que la genera. Això fa més fàcil realitzar el tractament de les primitives ja que es redueix el nombre dràsticament a l'hora de cercar i a més es té informació de qui la està sustentant i generant.

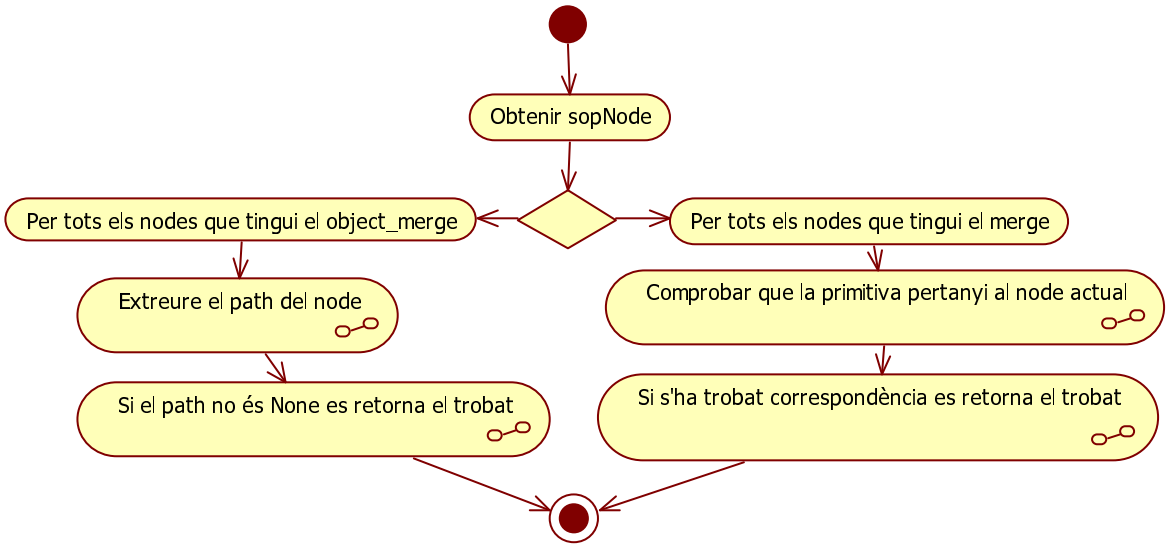


Figura 8.34: Exemple d'ús de la funció identifyBuildingDetailPrim(self,p).

- **getBuildingPrim(self, detailPrim):** Crida a la funció identifyBuildingDetailPrim i un cop té la primitiva respecte al node que la sustenta, obté l'ancestre i el retorna.

TopDownPrimSelector

Aquesta classe conté els mètodes per a poder cercar els nodes que generen les primitives que se l'hi passin.L'algorisme fa una cerca des del node arrel fins a les fulles la qual cosa el fa més eficient que no pas el mètode de la classe BottomUpPrimSelector explicada anteriorment. Es pot observar el seu diagrama en la Figura 8.35.

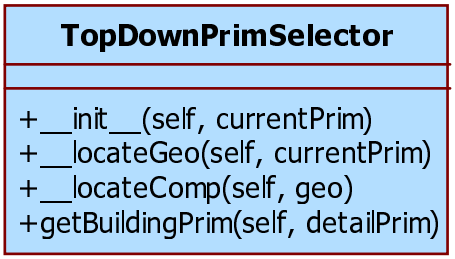


Figura 8.35: Diagrama de la classe TopDownPrimSelector.

- **___init___(self,currentPrim:** Inicialitza les variables d'instància.
- **___locateGeo(self, currentPrim):** Cerca tots els nodes de tipus "geo" que hi hagi en l'edifici, pot tenir més d'un. Retorna la llista amb els que ha trobat.
- **___locateComp(self, geo):** Retorna una llista de tots els nodes de tipus "comp" que es trobin dins de "geo".
- **getBuildingPrim(self, detailPrim):** Retorna el node que genera la primitiva "detailPrim" en una llista.

commonNodeAncestor:

Aquesta classe permet cercar donats **n** nodes, el seu node ancestre comú. En el pitjor dels casos retorna l'arrel del graf de nodes. Es pot observar el diagrama d'aquesta classe en la Figura 8.36.

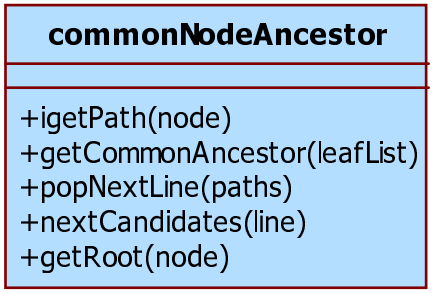


Figura 8.36: Diagrama de la classe commonNodeAncestor.

- **getCommonAncestor(leafList):** Retorna l'ancestre comú més pròxim a la llista de nodes.

- **igetPath(node):** Retorna tots els camins possibles des del node "node" fins a l'arrel en forma d'una llista de llistes [[camí1][camí2]...[camíN]]. El mètode és recursiu i permet anar empilant tots els possibles camins des del node que es passa com a paràmetre "node" fins a l'arrel del graf de nodes. Com hi ha punts on el mètode pot retornar un sol ancestre, o una llista d'ancestres, s'ha d'anar actualitzant tots els camins amb la informació recopilada fins al moment. Per a millor claretat de l'algorisme, es pot veure el pseudocodi en la Figura 8.37.

```
METODE igetPath(node) RETORNA llistaCamins
  inputs := node.inputs()
  multipleParents := []
  llistaCamins := []
  ret := []
  SI hiHaInputs(inputs) LLAVORS
    SI només té un ancestre LLAVORS
          ret := igetPath(inputs[0])
      SI ret és una llista LLAVORS
        //S'han retornat múltiples camins i s'han d'afegir
        PER cada item DINS ret FER
          llistaCamins.append([node]+item)
        FPER
      ALTRAMENT
        llistaCamins.append([node]+ret)
      FSI
    ALTRAMENT
      //Hi ha més d'un ancestre. Es crea 1 llista/ancestre.
      PER cada item DINS inputs FER
        multipleParents.append(igetPath(item))
      //En aquest punt es tenen tots els camins resultants
      //des de cada item fins al node arrel.
      PER cada item DINS multipleParents FER
        SI s'ha rebut múltiples camins LLAVORS
          //S'ha d'afegir el camí que els hi falta des de l'origen
          //fins al node que s'ha tractat
          PER cada llista DINS multipleParents[item] FER
            llistaCamins.append([node]+multipleParents[item])
          FPER
        ALTRAMENT
          llistaCamins.append([node]+multipleParents[item])
        FSI
    FSI
  ALTRAMENT
    llistaCamins.append([node])
  FSI
FMETODE
```

Figura 8.37: Pseudocodi de la funcióigetPath(node).

- **popNextLine(paths):** Retorna tots els tops de les llistes de camins de "paths" en forma d'una pila, per tal de ser tractada posteriorment. Per a més informació detallada sobre a que es refereix amb els "tops" de les llistes i en quant a l'algorísmica, es pot trobar en la secció 9.3.

- **nextCandidates(line):** Retorna una llista amb els candidats a ser ancestres comuns a partir de la llista de nodes "line".

- **getRoot(node):** Cerca recursivament el node arrel a partir d'un node "node".

CopyPasteStatus:

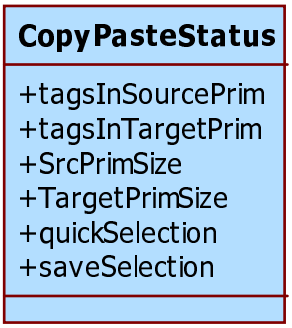


Figura 8.38: Diagrama de la classe CopyPasteStatus.

- **tagsInSourcePrim:** Llistat de tags de les primitives d'origen.

- **tagsInTargetPrim:** Llistat de tags de les primitives de destí.

- **SrcPrimSize:** Mida de la primitiva origen per dur a terme l'escalat per tal de proporcionar-ho. Aquest escalat es realitza en la funció paste_Selection de la classe CopyPasteOps explicada anteriorment, que és qui crida a aquest atribut.

- **TargetPrimSize:** Mida de la primitiva destí per a dur a terme transformacions a la primitiva d'origen.

- **quickSelection:** Booleà que permet saber l'estat del mode de selecció ràpid.

- **saveSelection:** Path en el que es desarà el fitxer XML triat per l'usuari.

8.4.4.3 Mòdul de XML

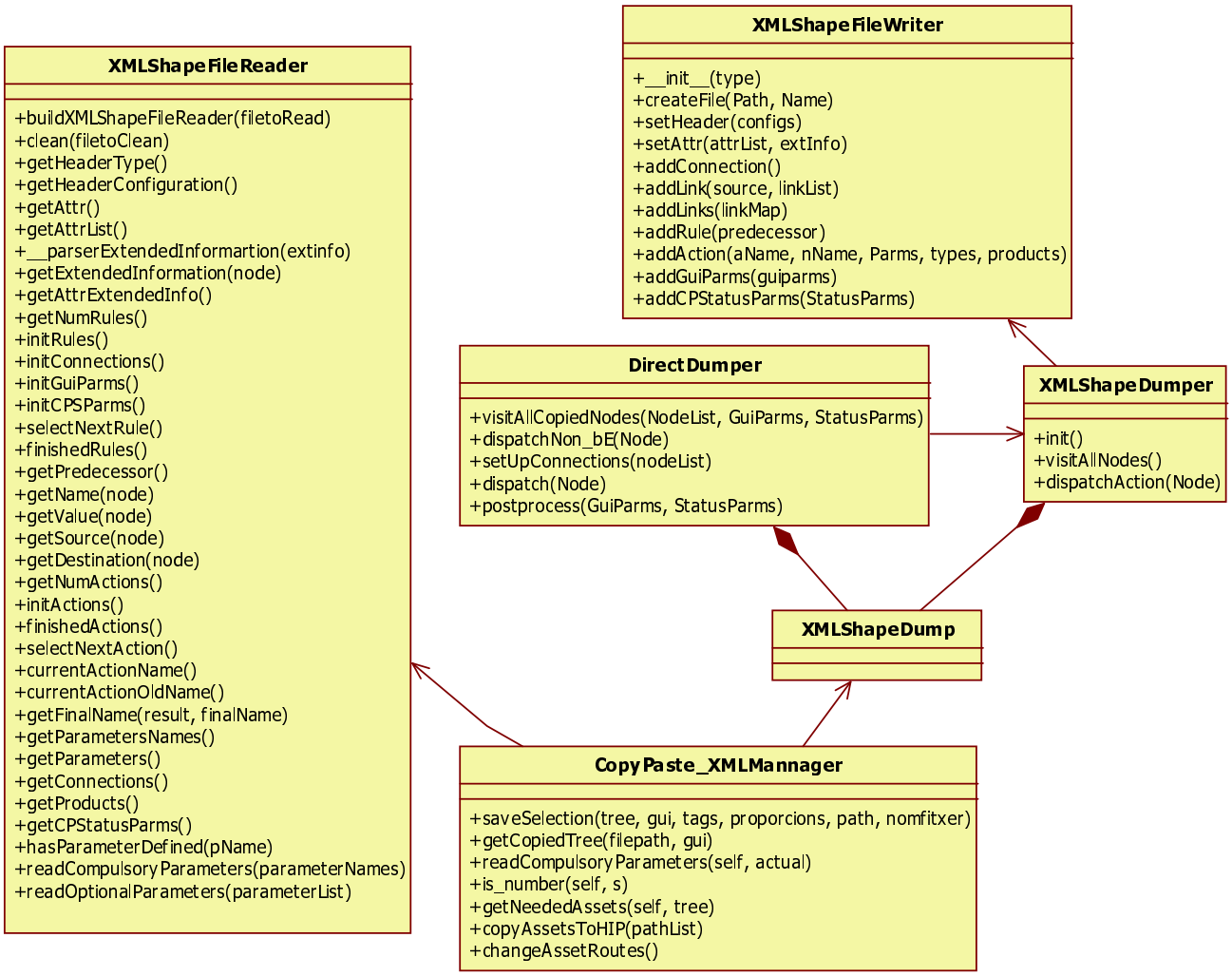


Figura 8.39: Diagrama de classes del mòdul XML.

CopyPaste_XMLMannager:

Aquesta classe és l'encarregada de gravar i carregar fitxers en format XML que contenen còpies efectuades per l'usuari. Es pot veure el diagrama d'aquesta classe en la Figura 8.40.

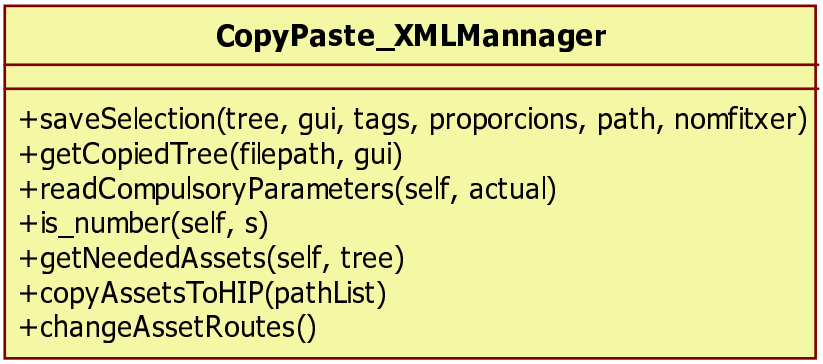


Figura 8.40: Diagrama de la classe CopyPaste_XMLMannager.

- **saveSelection(tree, gui, tags, proporcions, path, nomfitxer):** S'encarrega de cridar i passar els paràmetres a les classes necessàries per a poder generar un fitxer XML de nom 'nomfitxer' amb el sub-arbre de primitives copiat per l'usuari amb tota la informació necessària que es troba en 'tree', 'gui', 'tags' i 'proporcions'. El fitxer es desa en la ruta definida per 'path'. Es pot veure la funció en pseudocodi en la Figura 8.41.

```
METODE saveSelection(self, tree, gui, tagsInSourcePrim,
                     SrcPrimSize, path, filename) RETORNA Res
  guiparms := {}
  cpStatusparms := {}
  guiparms['guiparms'] = gui.obtenirTotsElsParametres()
  cpStatusparms['cpStatusparms'] =
        [tagsInSourcePrim.obtenirParms, SrcPrimSize.obtenirParms()]
  x := crear XMLShapeDump.DirectDumper(path, filename)
  x.visitarTotsElsNodesCopiats(arbre, guiparms, cpStatusparms)
FMETODE
```

Figura 8.41: Pseudocodi de la funció saveSelection(tree, gui, tags, proporcions, path, nomfitxer).

- **getCopiedTree(filepath, gui):** Aquest mètode carrega el fitxer XML que es troba en la ruta 'path', desant el contingut al buffer tal i com si l'usuari hagués acabat de realitzar la còpia del sub-arbre que conté el fitxer i emmagatzema al node 'gui' la informació necessària d'igual manera. També s'encarrega de canviar les rutes dels assets dels nodes de tipus 'Insert', delegant aquesta funcionalitat a la classe pertinent. Es pot veure el diagrama d'activitat d'aquest mètode en la Figura 8.42

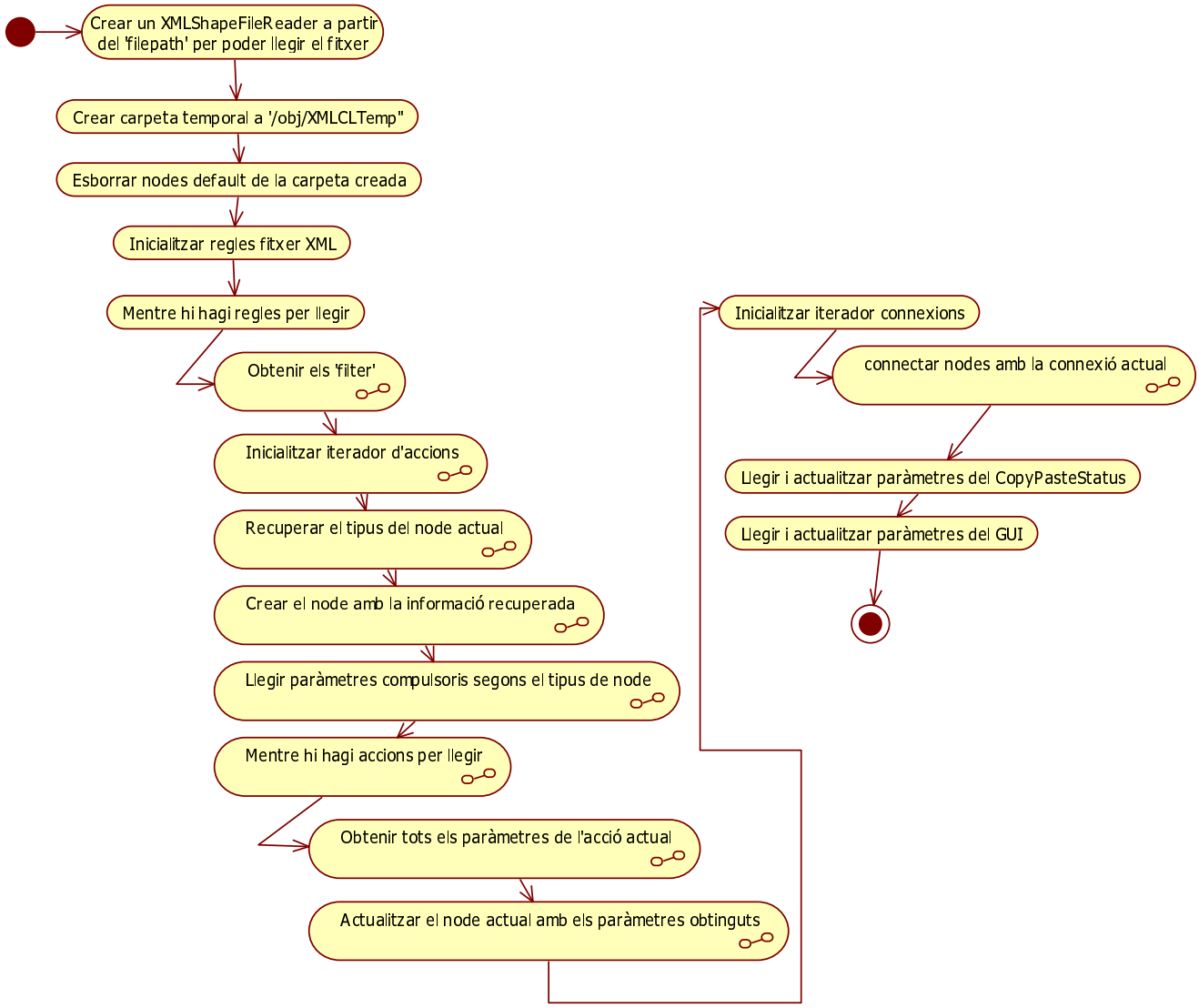


Figura 8.42: Exemple d'ús de la funció getCopiedTree(filepath, gui).

- **readCompulsoryParameters(self, actual):** S'encarrega d'extreure del node 'actual', dades que requereixen ser especialment llegides prèviament al intentar inserir tots els paràmetres d'aquest node. Per exemple, els nodes de tipus 'Subdiv' requereixen abans de res definir-lis el nombre de subdivisions que tindran, abans de definir cadascuna de les divisions. Retorna el paràmetre obligatori segons el tipus de node que sigui 'actual'.

- **is_number(self, s):** A partir de l'String 's' determina si el contingut llegit en forma de nombre és vàlid o no. Retorna cert si ho és i altrament retorna fals.

- **getNeededAssets(self, tree):** Retorna una llista amb tots els paths absoluts dels assets dels nodes de tipus 'Insert' que trobi fent el recorregut de l'arbre 'tree'. És a dir, crea una llista amb totes les rutes a fitxers de geometria que es troben en els nodes de tipus 'Insert' que s'han trobat continguts en el paràmetre 'tree'.

- **copyAssetsToHIP(self, pathList):** S'encarrega de realitzar una còpia de tots els assets que es troben en el llistat de rutes absolutes 'pathList' i les desa en un directori anomenat '/CopiedAssets' en l'arrel del directori que sustenta el projecta obert en Houdini.

- **changeAssetsRoutes(self, tree):** S'encarrega de canviar les antigues rutes dels nodes de tipus 'Insert' de l'arbre 'tree' per rutes relatives al directori '/CopiedAssets' de forma que trobin la geometria que referenciaven en l'edifici original.

DirectDumper:

Aquesta classe s'encarrega de recorre la còpia efectuada i gestionar la seva gravació en fitxer amb format XML de manera transparent. Es pot veure el diagrama de la classe en la Figura 8.43.

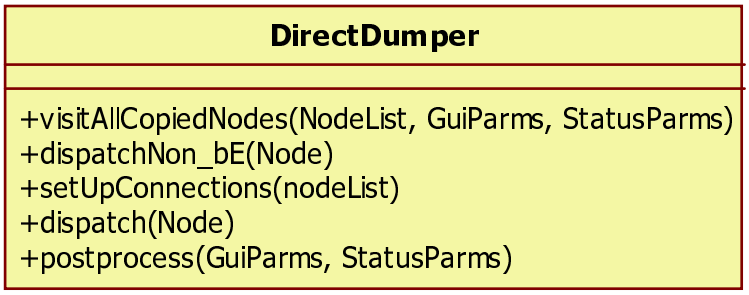


Figura 8.43: Diagrama de la classe DirectDumper.

- **visitAllCopiedNodes(self, NodeList, guiParms, StatusParms):** Aquest mètode és l'encarregat de dirigir i delegar totes les tasques del procés de copiar tota la informació de la llista de nodes 'NodeList' en un fitxer en format XML de tal manera que al tornar a llegir aquest fitxer es pugui reproduir el sub-arbre copiat per l'usuari per al seu posterior enganxat. Es pot veure el pseudocodi d'aquesta funció en la Figura 8.44, on ***w*** és una instància de la classe XMLShapeFileWriter.

```
METODE visitAllCopiedNodes(self, nodelist, guiparms, cpstatusparms)
RETORNA Res
  PER tots els nodes DINS nodelist:
    tractarNode()
  FPER
  setUpConnections(nodelist)
  postprocess(guiparms, cpstatusparms)
  self.w.crearFitxer(self.filePath, self.fileName)
FMETODE
```

Figura 8.44: Pseudocodi de la funció visitAllCopiedNodes(self, NodeList, guiParms, StatusParms).

- **dispatchNon_bE(self, Node):** Tracta la informació de nodes especials que no pertanyen pròpiament a **building**Engine, en cas de ser de tipus 'AsianRoof' desa els paràmetres especials d'aquest tipus de node, altrament s'omet el node ja que aquest projecte es centre en nodes pertinents a **building**Engine.

- **setUpConnections(self, nodeList):** S'encarrega de bolcar en el fitxer XML totes les connexions entre els nodes de la llista de nodes ' nodeList' per tal que al llegir el fitxer es sàpiguen les connexions que fan possible el sub-arbre copiat.

- **dispatch(Node):** S'encarrega de tractar el node 'Node' i de copiar el seu nom, tipus, paràmetres i valors en el fitxer en format XML.

- **postprocess(GuiParms, StatusParms):** S'encarrega d'acabar de bolcar en el fitxer XML, els paràmetres que no tenen pròpiament a veure amb el sub-graf si no amb l'estat de la còpia i variables auxiliars necessàries per a la consistència de la còpia.

XMLShapeDumper:

Aquesta classe s'especialitza en gravar a disc cadascun dels paràmetres relacionats amb un node. Es pot observar el seu diagrama en la Figura 8.45.

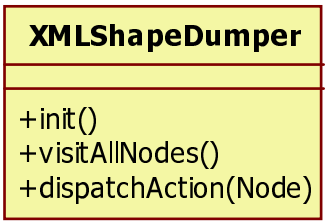


Figura 8.45: Diagrama de la classe XMLShapeDumper.

- **init(self, filePath, fileName, network):** S'encarrega d'inicialitzar la instància de la classe per tal que estigui preparada per a començar el procés de gravar nodes en un fitxer de format XML.

- **dispatchAction(self, aNode):** S'encarrega de gravar en el fitxer de format XML tots els paràmetres, un a un, del node 'aNode'. Aquesta tasca la du a terme en forma de tags XML tal i com s'ha explicat anteriorment.

XMLShapeFileWriter:

Aquesta classe és l'encarregada d'escriure a disc tot allò que l'hi mana la classe XMLShapeDumper. Conté la implementació de com escriure cadascun dels paràmetres, connexions, noms etc, necessaris per a la classe XMLShapeDumper. Es pot veure el seu diagrama en la Figura 8.46

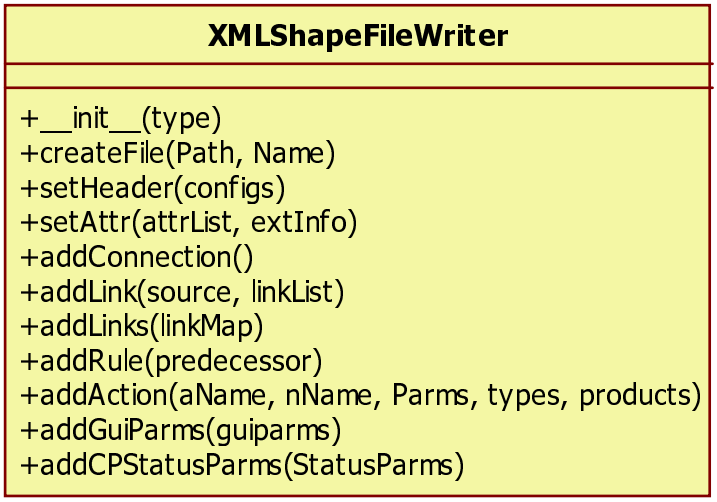


Figura 8.46: Diagrama de la classe XMLShapeFileWriter.

- **___init___(type):** S'encarrega d'inicialitzar les variables de classe necessàries per a realitzar l'escriptura a disc en format XML de tota la informació que conté el sub-graf copiat per l'usuari.

- **createFile(self, Path, Name):** Crea l'estructura interna en un fitxer XMLShape i el desa en el path 'Path' amb el nom 'Name', si el fitxer existeix el sobreescriu.

- **setHeader(configs):** Escriu l'encapçalament del fitxer XML amb paràmetres de configuració facilitats per 'configs'.

- **setAttr(self, attrList, extInfo):** Crea seqüencialment un element 'attr' (atribut) en el fitxer XML i hi desa el nom, tipus i valor a més d'informació extra si és necessari.

- **addConnection(self):** Crea seqüencialment un nou element 'connections' per tal que es puguin anar afegint enllaços en aquest apartat en el fitxer XML.

- **addLink(self, source,linkList):** Escriu de manera seqüencial, en el fitxer XML, tots els enllaços que es generen entre 'source' i els nodes de la llista 'linkList'. És a dir, es genera una línia en el fitxer prenent el paràmetre 'source' com a origen, i cadascun dels nodes que es trobin en 'linkList' com a destinació.

- **addLinks(self, linkMap):** S'encarrega de delegar l'escriptura a disc al mètode 'addLink' anterior, del mapa d'enllaços desat en 'linkMap'.

- **addRule(self ,predecesor):** Crea una nova regla 'rule' en el fitxer XML i l'inicialitza per tal que s'hi puguin afegir paràmetres més endavant.

- **addAction(self, aName, nName,Parms, types, products):** Crea una nova acció 'action' en el fitxer XML i l'inicialitza per tal que s'hi puguin afegir paràmetres més endavant.

- **addGuiParms(self, guiparms):** Crea una nova categoria 'guiparms' en el fitxer XML i hi desa els paràmetres del node GUI que es troben en 'guiparms'.

- **addCPStatusParms(self, StatusParms):** Crea una nova categoria 'cpStatusparms' en el fitxer XML i hi desa els paràmetres de l'estat de la còpia necessaris per a poder realitzar una reproducció del sub-graf de nodes correctament. I finalment desa els paràmetres que es troben en 'StatusParms' dins aquesta categoria en el fitxer XML.

XMLShapeFileReader:

Aquesta classe s'encarrega de, a partir d'un fitxer en format XML gravat per XMLShapeDump, reconstruir el sub-graf de nodes i regles contingut en el fitxer. Es pot observar el seu diagrama en la Figura 8.47.

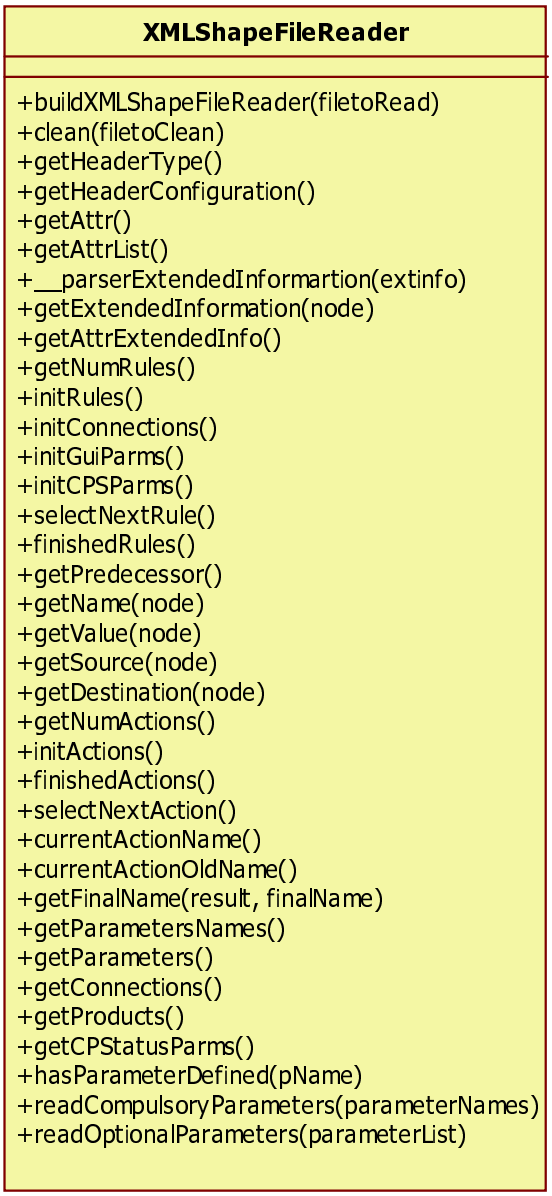


Figura 8.47: Diagrama de la classe XMLShapeFileReader.

- **___init___(self, fileToRead):** S'encarrega d'obrir el fitxer en el path 'fileToRead' i inicialitzar aquells paràmetres d'instància necessaris per iterar dins del fitxer XML.

- **buildXMLShapeFileReader(self, filetoRead):** Mètode principal que s'encarrega de crear una instància de 'XMLShapeFileReader' i retornar-la per a que es pugui iterar sobre el contingut del fitxer externament (retorna un iterador).
- **clean(self, filetoClean):** Mètode encarregat d'esborrar tots els possibles separadors introduïts per casos particulars en la generació del fitxer XML per a poder-hi iterar correctament.
- **getHeaderType(self):** Retorna el tipus de codificació desat en la capçalera llegida del fitxer XML.
- **getHeaderConfiguration(self):** Retorna tota la informació desada en la capçalera del fitxer XML.
- **getAttr(self):** Retorna una Map de tots els atributs de l'apartat XML on es troba llegint actualment.
- **getAttrList(self):** Retorna una Llista de tots els atributs de l'apartat XML on es troba llegint actualment.
- **___parserExtendedInformation(self,extinfo):** Retorna aquella informació estesa que es demani mitjançant 'extinfo', que contingui el fitxer XML.
- **getExtendedInformation(self,node):** Retorna tota la informació estesa que contingui el fitxer XML sobre el node 'node' en forma de Map.
- **getAttrExtendedInfor(self):** Retorna en forma de Map tota la informació estesa que tingui l'apartat de 'attr' actual que s'estigui llegint en el fitxer XML.
- **getNumRules(self):** Retorna el nombre de regles 'rule' que conté el fitxer XML.
- **initRules(self):** Inicialitza l'iterador al principi de tot del fitxer per tal que es pugui començar a llegir elements i iterar.
- **initConnections(self):** Inicialitza l'iterador al principi de l'apartat de 'connections' del fitxer XML.
- **initGuiParms(self):** Inicialitza l'iterador al principi de l'apartat de 'guiparms' del fitxer XML.
- **initCPSParms(self):** Inicialitza l'iterador al principi de l'apartat de 'cpstatusparms' del fitxer XML.
- **selectNextRule(self):** Selecciona de manera interna la següent regla (Itera a la següent regla) sense retornar res.
- **finishedRules(self):** Retorna cert si s'han acabat de llegir totes les regles del fitxer XML, altrament retorna fals.
- **getPredecessor(self):** Retorna el nom del predecessor que pugui tenir l'apartat 'predecessor' dins d'un 'rule' en cas que el tingui.
- **getName(self, node):** Retorna el nom del node 'node' escrit dins del camp 'name' del fitxer XML.
- **getValue(self, node):** Retorna el valor del node 'node' escrit dins del camp 'value' del fitxer XML.

- **getSource(self, node):** Retorna el valor del node 'node' escrit dins del camp 'src' del fitxer XML.
- **getDestination(self, node):** Retorna el valor del node 'node' escrit dins del camp 'dest' del fitxer XML.
- **getNumActions(self):** Retorna el nombre d'accions de la regla actual escollida per l'iterador intern de regles.
- **initActions(self):** Reinicialitza l'iterador per tal que apunti a la primera acció de la regla actual.
- **finishedActions(self):** Retorna cert si no hi ha més accions per iterar, altrament retorna fals.
- **selectNextAction(self):** Itera internament a la següent acció de la regla actual de manera que no retorna res.
- **currentActionName(self):** Retorna el nom de l'acció actual a la que apunta l'iterador.
- **currentActionOldName(self):** Retorna el nom original que tenia en l'edifici d'on s'ha copiat, del node que s'està tractant al que està apuntant l'iterador. D'aquesta manera es pot refer el graf original del fitxer (ja que l'edifici on es vol enganxar pot contenir nodes anomenats de la mateixa manera que en l'edifici antic.
- **___getFinalName(self, result, finalName):** Retorna el nom de 'finalName' segons la següent numeració no feta servir en l'edifici actual per tal de no crear conflictes.
- **getParametersNames(self):** Retorna una llista de String amb tots els noms dels paràmetres que conté l'acció actual dins de la regla actual dels dos iteradors respectivament.
- **getParameters(self):** Retorna una llista de tuples de String de tipus "nom:valor" que conté l'acció actual dins de la regla actual dels dos iteradors respectivament.
- **getConnections(self):** Retorna una llista de tuples de parelles de tipus String en format "destí:origen" que representa un enllaç entre els dos nodes llegits.
- **getProducts(self):** Retorna una llista de String amb els noms dels productes que genera el node de la regla actual i l'acció actual dels dos iteradors respectivament.
- **getGuiParms(self):** Obté una llista dels paràmetres del node GUI que s'hagin desat en el fitxer XML.
- **getCPStatusParms(self):** Obté una llista de tots els paràmetres de l'estat de la còpia desats en el fitxer XML.
- **hasParameterDefinded(self, pName):** Retorna cert si la regla actual té definits paràmetres amb nom 'pName' altrament retorna fals.
- **readCompulsoryParameters(self, parameterNames):** Retorna un Map amb tots aquells paràmetres que s'hagin volgut obtenir continguts en la llista 'parameterNames' del fitxer XML que s'està llegint.
- **readOptionalParameters(self, parameterList):** Retorna un Map amb tots aquells paràmetres opcionals que s'hagin volgut obtenir continguts en la llista 'parameterList' del fitxer XML que s'està llegint.

8.4.4.4 Mòdul d'acceleració de la interacció d'usuari

Aquest mòdul conté totes les operacions necessàries per a realitzar una acceleració en la interacció d'usuari cada cop que aquest ho desitgi. Es pot veure el diagrama en la Figura 8.48.

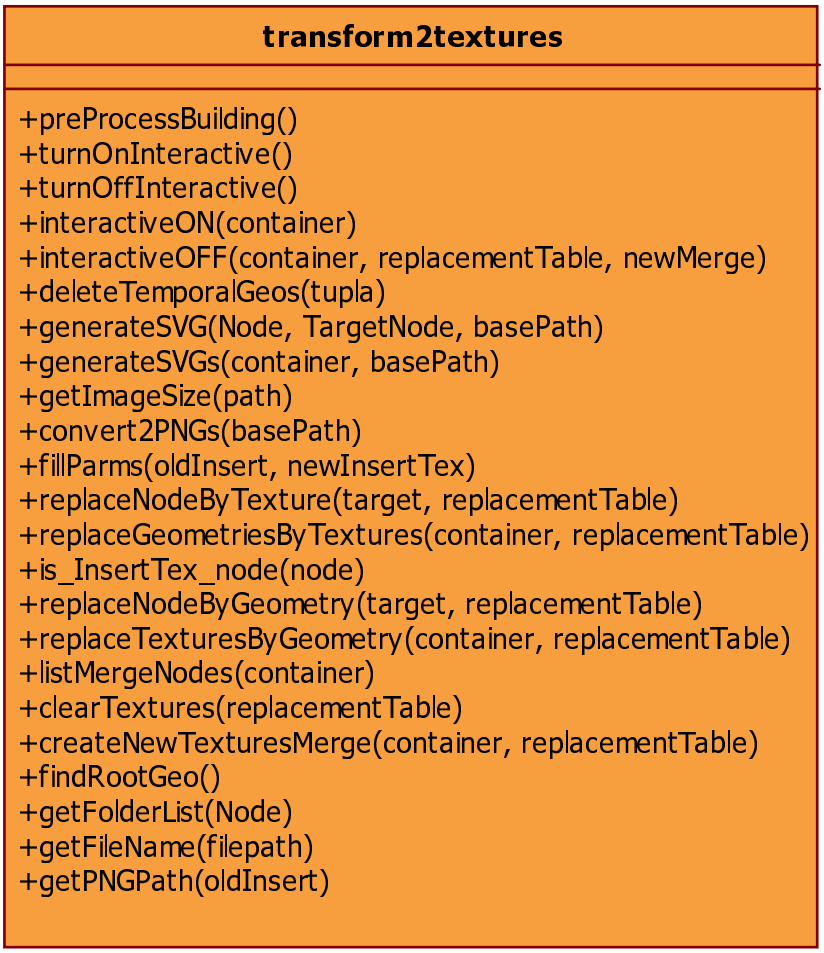


Figura 8.48: Diagrama de classes del mòdul d'acceleració de la interacció d'usuari.

transform2textures:

- **preProcessBuilding():** Processa tota la geometria de l'edifici carregat de manera que genera, de cadascun dels assets inserits, un fitxer SVG i un altre transcodificat a format PNG de manera automàtica per l'usuari. D'aquesta manera deixa a punt l'entorn per a poder activar i desactivar el mode d'interacció accelerada de l'usuari.

- **turnOnInteractive():** Aquest mètode permet activar el mode ràpid d'interactivitat d'usuari intercanviant tota la geometria per les imatges generades pel preProcessBuilding(). Retorna una llista amb els canvis que ha efectuat per a poder revertir-los. Es pot veure el pseudocodi d'aquesta funció en la Figura 8.49.

```
METODE turnOnInteractive() RETORNA switchInfo
  switchInfo = {}
  #S'obtenen tots els nodes de tipus 'subnet' o 'geo' que
  #poden contenir nodes que han de ser tractats
  containerList = obtenirGeoArrel()
  PER tots els nodes DINS containerList FER
    TaulaDeReemplaçament = interactiveON(node)
    newMerge = createNewTexturesMerge(node, TaulaReemplaçament)
    switchInfo.afegir((node, TaulaReemplaçament, newMerge))
  FPER
FMETODE
```

Figura 8.49: Pseudocodi de la funció turnOnInteractive().

Com es pot observar en el pseudocodi, el primer que es fa és obtenir els nodes de tipus 'geo' que siguin l'arrel d'un graf. Aquests nodes són passats com a paràmetre de la funció interactiveON, que s'explica a continuació d'aquest apartat. D'aquesta operació s'obté la taula de reemplaçaments efectuats. A continuació es crea un nou node Merge per tal de visualitzar els canvis en cada objecte 'geo' tractat. La informació dels canvis es desa en la variable 'switchInfo' que és retornada per tal de poder desfer aquests canvis posteriorment.

- **turnOffInteractive(switchInfo):** Mitjançant la llista de canvis efectuats prèviament 'erase', torna a substituir les imatges per la geometria inserida originalment, revertint així el procés del mètode turnOnInteractive(). Es pot veure el pseudocodi d'aquesta funció en la Figura 8.50.

```
METODE turnOffInteractive(switchInfo) RETORNA Res
  #switchInfo conté el que retorna la funció turnOnInteractive amb
  #les substitucions que ha realitzat per a poder desfer-les
  #S'obtenen tots els nodes de tipus 'subnet' o 'geo' que
  #poden contenir nodes que han de ser tractats
  containerList = obtenirGeoArrel()
  PER tots els nodes DINS switchInfo FER
                interactiveOFF(node[0], node[1], node[2])
  FPER
FMETODE
```

Figura 8.50: Pseudocodi de la funció turnOffInteractive(swithcInfo).

- **interactiveON(container):** S'encarrega de delegar l'intercanvi de geometria per textures a la funció explicada prèviament 'turnOnInteractive', i un cop fet, retorna la taula de reemplaçaments.

- **interactiveOFF(container, replacementTable, newMerge):** Reverteix el procés del mètode interactiveON, tornant a substituïr les textures per geometries gràcies a la taula de reemplaçaments 'replacementTable'. També s'encarrega d'esborrar el node Merge 'newMerge' creat per a visualitzar el mode ràpid i d'activar la visibilitat de tots els nodes per a que torni a quedar l'escena com abans d'activar el mode ràpid.

- **deleteTemporalGeos(tupla):** Elimina nodes temporals creats en el procés de activar i desactivar el mode ràpid.

- **generateSVG(Node, TargetNode, basePath):** És el mètode encarregat de delegar la generació del fitxer en format 'SVG' per al node 'Node' que es troben en el graf de nodes de l'edifici carregat. El paràmetre basePath és opcional, en cas de no estar definit, el mètode pren un path genèric com a predeterminat (relatiu a la carpeta on es desa el projecte obert), on es suposa que es troba la ruta absoluta de les imatges generades.

- **generateSVGs(container, basepath):** Mètode encarregat de, per a cada node del contenidor 'container', delegar la seva generació en format 'SVG' i d'esborrar els nodes residuals de tipus 'geo' creats eventualment per a dur a terme la tasca.

- **getImageSize(path):** Obté el fitxer que indica el paràmetre 'path' de tipus 'SVG' per tal de llegir-lo mitjançant un parser XML i poder extreure les dimensions de la imatge per a poder ser retallada més endavant pel Batik.

- **convert2PNGs(basePath):** Aquest mètode s'encarrega de delegar al mòdul Batik de transcodificar totes les imatges de format 'SVG' a format 'PNG' per tal que puguin ser importades més endavant com a textura. Es pot veure el diagrama de seqüència d'aquesta funció en la Figura 8.51.

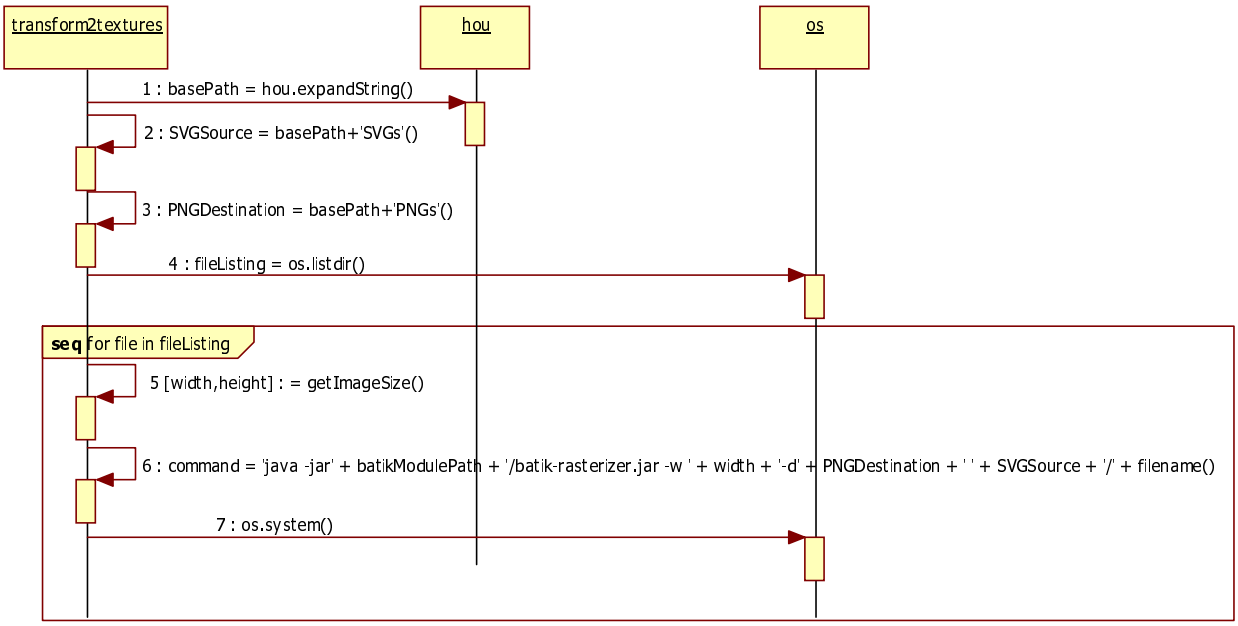


Figura 8.51: Exemple d'ús de la funcióconvert2PNGs(basePath).

- **fillParms(oldInsert, newInsertTex):** Aquest mètode copia tots aquells paràmetres del node 'oldInsert' al nou node 'newInsertTex' tals com:

  - Valors de translació o posició
  - Valors de rotació
  - Valors d'escalat

D'aquesta manera la textura substitutiva de la geometria adopta els mateixos valors i es situa i proporciona exactament en el lloc de la geometria.

- **replaceNodeByTexture(target, replacementTable):** Mètode que s'encarrega de la creació d'un node de tipus 'InsertTex' per tal que sustenti la textura per la qual es vol reemplaçar la geometria existent. S'encarrega de desconnectar, connectar i configurar els paràmetres dels nodes de tal manera que la operació d'intercanvi de geometria per textures quedi acabada. Els canvis efectuats queden registrats en el 'replacementTable' per tal de poder revertir els canvis més endavant.

- **replaceGeometriesByTextures(container, replacementTable):** Mètode encarregat de delegar el reemplaçament tots els nodes de tipus 'Insert' per nodes de tipus 'InsertTex' i esborrar els nodes de tipus 'geo' temporals que s'ha creat per a la tasca anterior.

- **is_InsertTex_node(node):** Retorna cert si el node 'node' és de tipus 'InsertTex', altrament retorna fals.

- **replaceNodeByGeometry(target, replacementTable):** Mètode que s'encarrega de delegar la reconnexió dels nodes de tipus 'Insert' ja que ara la connexió la tenen els nodes de tipus 'InsertTex'.

- **replaceTexturesByGeometry(container, replacementTable):** Mètode que s'encarrega de delegar la substitució les textures per geometria i també delegar la neteja dels nodes 'geo' creats de forma temporal anteriorment.

- **listMergeNodes(container):** Retorna una llista amb tots els nodes trobats de tipus 'merge' o 'object_merge'.

- **clearTextures(replacementTable):** Destrueix tots les nodes de tipus 'InsertTex' que contenien les imatges substitutòries de la geometria per tal de netejar el graf de nodes, això ho fa a partir de la taula de reemplaçament 'replacementTable'.

- **createNewTexturesMerge(container, replacementTable):** Crea un nou node de tipus 'Merge' per a poder visualitzar les textures incorporades. Es pot veure el pseudocodi de la funció en la Figura 8.52.

```
METODE createNewTexturesMerge(container, replacementTable)
        RETORNA ObjectMerge
  m = listMergeNodes(container)
  #Crea el nou ObjectMerge
  ObjectMerge = container.crearNode('object_merge')
  #Initializar transform: En un objecte especific
  ObjectMerge.parm('xformtype').set(2)
  #Activa el mode de view per a que es visualitzi el node
  ObjectMerge.activarDisplayFlag()
  #Per tots els nodes merge trobats, llegeix l'input
  PER n DINS m FER
    parms = n.obtenirTotsElsParametres()
    actual = parms.seguentParametre()
    #Tots els paràmetres prèvis a 'numobj' s'ometen
    MENTRE actual.name()!='numobj' FER
          actual = parms.seguentParametre()
    #Inicialitzar els inputs del merge
    ObjectMerge.init()
        #Es comparen els nodes visibles que no son Inserts
        #i s'afegeixen al nou Merge
  MENTRE hi hagi paràmetres en el Merge FER
      actual = parms.seguentParametre()
      #Es mira si està activat, si no està activat no cal afegir-lo
      SI actual.activat() FER
        ObjectMerge.afegir_I_ActualitzarParametres()
      FSI
    FMENTRE
  FPER
FMETODE
```

Figura 8.52: Pseudocodi de la funció createNewTexturesMerge(container, replacementTable).

Com es pot observar en el codi de la Figura 8.52, primer de tot es crea el nou node de tipus 'Merge' que serà l'encarregat de visualitzar els nous canvis efectuats. Un cop creat, s'indica el tipus de transformació que es vol realitzar. Seguidament s'activa la visualització del node, ja que Houdini només permet visualitzar un node alhora. Si es vol tenir la representació de varis, aquests varis s'han d'encapsular en un node de tipus 'Merge' o 'Object_Merge' i a les hores activar aquest per a visualitzar-los tots. Aquesta és la raó de ser d'aquest mètode. Per tant, un cop activat, cal dir-li quins nodes volem encapsular-hi. Per fer-ho l'algorisme recorre tots les nodes de tipus merge, i introdueix la informació per a visualitzar tots aquells nodes que no siguin de tipus 'Insert' ja que s'han substituït per textures. Com els nodes de tipus 'InsertTex' ja s'han creat prèviament a la crida d'aquest mètode, amb la operació anterior també han sigut afegits a la visualització. Per tant un cop finalitza el mètode, es pot observar que l'edifici ha passat de tenir figures geomètriques a imatges substitutives.

- **findRootGeo():** Cerca el primer node de tipus 'geo' de l'arbre amb arrel "/obj", aquest node ha de ser únic en aquest nivell de l'edifici, i és el que sustenta tota la informació geomètrica i de les regles que defineixen l'edifici.

- **getFolderList(Node):** Retorna una llista de nodes de tipus 'subnet' o 'geo' que han de

ser processats de manera separada, si no se'n troben retorna el node "Node" en forma de llista.

- **getFileName(filepath):** Retorna el nom del fitxer a partir del path 'filepath'.
- **getPNGPath(oldInsert):** Retorna el path amb el nom del fitxer i format inclòs d'allà on s'hauria de trobar la imatge que carrega el node 'oldInsert'.

Capítol 9

Implementació i proves

9.1 Mòdul d'Acceleració de la interacció d'usuari

En el capítol anterior s'han pogut veure les funcions d'aquests mòduls i com es relacionen amb la GUI de Houdini. A continuació es remarcaran alguns aspectes tècnics que fan possible el funcionament d'aquest mòdul. La clau de l'acceleració en la interacció, com ja s'ha explicat en el capítol 8, és la substitució de la geometria per imatges d'aquesta. Això fa que es redueixi dràsticament el nombre de polígons en l'escena i d'aquesta manera Houdini pugui processar més ràpidament l'edifici.

Per a dur a terme aquesta tasca, primer es fa el pre-processat de l'edifici. Aquest pre-processat és l'encarregat d'agafar tots els nodes de tipus 'Insert' i cercar el seu paràmetre 'asset'. Com s'ha explicat en el capítol 7, un asset és un fitxer de geometria extern que es carrega mitjançant un node de tipus 'Insert'. Un cop obtinguts tots els assets, el mòdul s'encarrega de fotografiar la geometria des de la seva vista frontal i en genera una imatge en format *.SVG*, desant-la en una nova carpeat */SVGs* creada en l'arrel del projecte.

El format d'imatge *SVG* (Scalable Vector Graphics) és, com indiquen les seves sigles, un format vectorial. Això vol dir que pot ser escalat sense que hi hagin pèrdues en el seu contingut. Malauradament Houdini no està preparat per a carregar imatges en aquest format. Per aquesta raó, s'ha fet servir el transcodificador Batik per tal de convertir les imatges de format *SVG* a format *PNG*. En la Figura 9.1 es pot observar la comanda que fa possible realitzar la conversió. Com es pot observar és una comanda executada sobre la màquina virtual de Java.

```
java -jar batik-rasterizer.jar -w <width> -h <height> -d <path>/PNGs/filename.png
<path>/SVGs/filename.svg
```

Figura 9.1: Comanda que permet convertir l'arxiu 'filename' de SVG a PNG.

Python permet fer crides a programes externs, i és així com s'aconsegueix executar el Batik fora de l'entorn de Houdini de manera integrada a Python. Aquesta operació es fa per a tots els fitxers *SVG* generats, convertint-los tots i desant-los en una nova carpeta anomenada */PNGs*. Un cop generats tots els fitxers en format *PNG* només cal substituir la geometria per aquestes imatges.

Aquesta última fase de la tasca es realitza creant un node de tipus 'InsertTex' per cada node de tipus 'Insert' que es trobi en la escena i connectant a aquest nou node els *input* del node 'Insert'. En el node 'InsertTex' la ruta del fitxer que es carrega és la del fitxer en format *PNG*.

Es poden veure resultats de la implementació d'aquest mòdul amb imatges de com queda el graf de nodes i l'escena a més de proves de rendiment, en el capítol 10 de resultats.

A continuació, en la Figura 9.2 es pot veure el codi que s'ha fet servir per a efectuar el test de rendiment que s'explicarà més endavant en el capítol 10.

```
def startTestTex(rangvalue):
  import time
  import hou
  f = open(hou.expandString('$HIP/textimes.txt'),'w')
  for n in range(rangvalue):
    now=time.time()
    hou.parm('.../Extrude1/height').set(5357.85+n*100);
    hou.node('.../object_merge1').cook();
    f.write(str(time.time() - now)+'\n')
    f.close()

def startTestIns(rangvalue):
  import time
  import hou
  f = open(hou.expandString('$HIP/instimes.txt'),'w')
  for n in range(rangvalue):
    now=time.time()
    hou.parm('.../Extrude1/height').set(5357.85+n*100);
    hou.node('.../object_merge5').cook();
    f.write(str(time.time() - now)+'\n')
  f.close()
```

Figura 9.2: Codi per efectuar els test de rendiment del mode d'acceleració.

Aquestes dues funcions generen dos fitxers anomenats "textimes.txt" i "instimes.txt" que contenen els temps de cada iteració de l'algorisme, amb els que s'ha construït la gràfica del capítol 10.

9.2 Mòdul de XML

Com s'ha vist anteriorment, el mòdul de XML consta de diverses classes tal i com es recorda en la Figura 9.3, principalment com és d'esperar es divideixen en dues funcionalitats: la de llegir fitxers XML i la d'escriure fitxers XML.

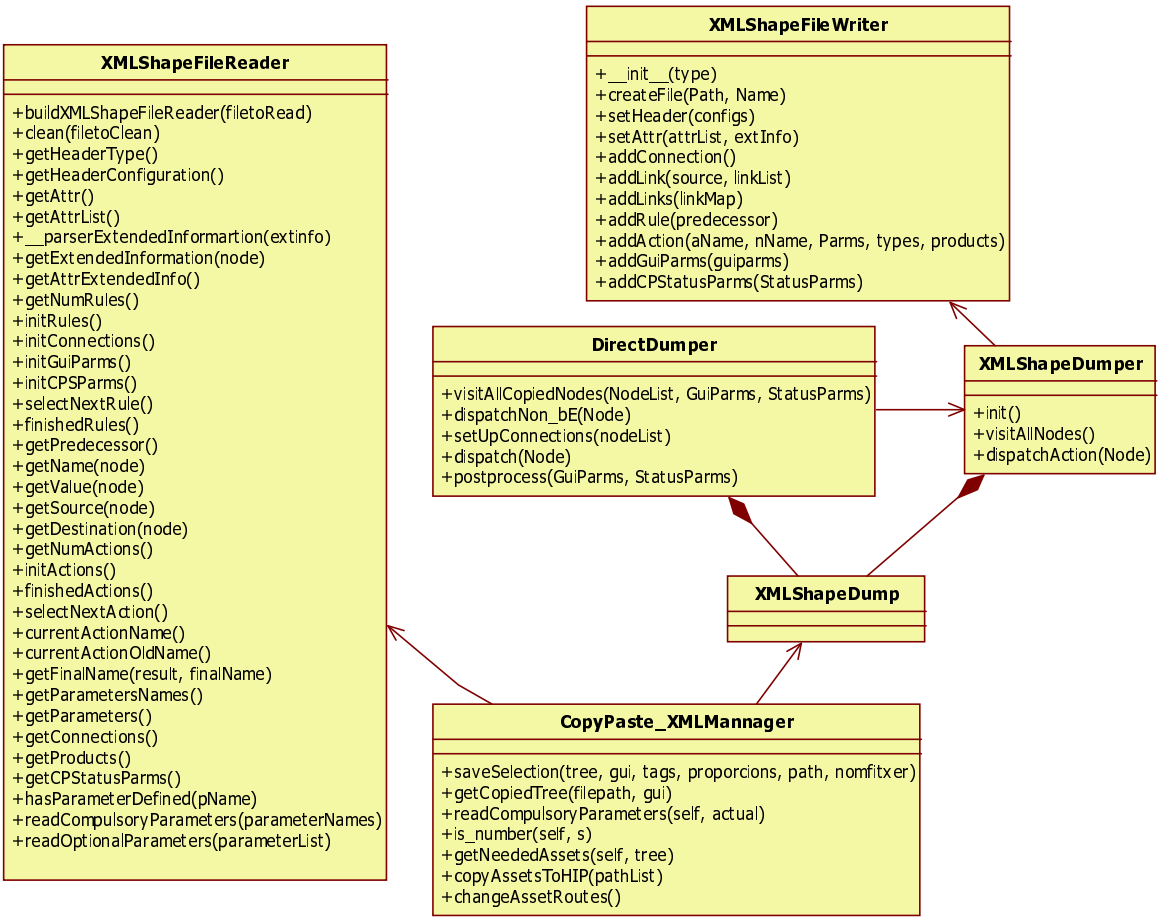


Figura 9.3: Diagrama de classes del mòdul XML.

Per tal de tenir ordenades aquestes dues funcionalitats, s'ha encapsulat en la classe "XMLShapeFileReader" les funcionalitats de lectura i en la de "XMLShapeFileWriter" les d'escriptura. Totes dues funcionalitats s'implementen independentment del recorregut dels nodes, ja que així es poden fer servir, implementant nous iteradors, per a altres estructures.

Es començarà explicant l'escriptura de fitxers. Per dur a terme aquesta tasca s'ha d'obtenir l'estructura copiada per l'usuari i passar-l'hi al gestor "CopyPaste_XMLMannager". Aquest s'encarrega de delegar la feina a la classe "XMLShapeDump". En la Figura 9.4 es pot veure un exemple on l'usuari selecciona una finestra de la casa Racolette.

Figura 9.4: Exemple de copiar la finestra marcada en vermell en un fitxer xml.

Les regles que generen la finestra seleccionada, les sustenten els nodes que es poden veure en la Figura 9.5. En aquest cas és molt senzill ja que la selecció només consta d'una finestra. D'aquesta manera es veurà quin fitxer XML es genera a partir d'aquesta selecció.

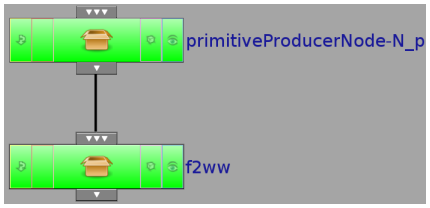


Figura 9.5: Nodes resultants de la còpia de la finestra de la Figura 9.4.

Com es pot observar només consta de dos nodes. El node 'primitiveProducerNode-N_p' és l'ancestre comú trobat de la selecció de la finestra. Per aquesta raó se l'hi ha canviat el nom, per tal de poder seguir-lo un cop s'enganxi en un nou edifici. Com es pot veure, el node 'f2ww' que correspón al node de tipus 'Insert', és el que sustenta la geometria de la finestra.

L'algorisme el que fa és recorre el sub-graf representat en la Figura 9.5 i volcar-lo en un fitxer XML. Per fer-ho s'ha triat una sintaxis específica per a la tasca d'aquest projecte.

9.2.1 Seccions d'un fitxer XML

A continuació es llisten les diferents seccions que formen un fitxer XML.

- Versió de XML
- CGAShape
  - Capçalera del fitxer on s'hi troben les metadades.

- Un seguit de regles que contenen:
  - Zona on es poden desar els predecessors
  - Zona on es poden desar els successors
    - Llista d'accions: cada acció conté una llista de paràmetres pròpia de cada tipus de node.
- Un apartat de connexions que conté en forma de paràmetres totes les relacions entre nodes.
- Un apartat de paràmetres del GUI clau.
- Un apartat de paràmetres del CopyPasteStatus que entre d'altres hi ha les proporcions de la còpia.

A continuació es pot veure en la Figura 9.6 i la Figura 9.7 una mostra del fitxer resultant de la selecció de la Figura 9.4 on s'observa l'estructura acabada d'explicar.

```
<?xml version="1.0" ?>
<CGAShape>
 <header type="XMLShape:Building">
  <config name="useFinalMerge" value="bE_sistematicNodeVisitors"/>
 </header>
 <rule>
   <predecessor name="f2windows">
   </predecessor>
   <successor>
     <action name="bE:Subdiv" nodename="primitiveProducerNode-N_p">
       <param name="Divisions" type="int" value="3"/>
       <param name="tol3d" type="float" value="0.00600000005"/>
       <param name="sumKs" type="float" value="2.44000006"/>
       <param name="MainParms1" type="int" value="0"/>
       <param name="rel2" type="int" value="1"/>
       <param name="rel1" type="int" value="1"/>
       <param name="rel0" type="int" value="1"/>
       <param name="dojitter" type="int" value="1"/>
     <param name="jitteramount" type="float" value="0.00100000005"/>
       <param name="seed" type="int" value="1"/>
       <param name="value2" type="float" value="0.200000003"/>
       <param name="value1" type="float" value="1.12"/>
       <param name="value0" type="float" value="1.12"/>
       <param name="enabled1" type="int" value="1"/>
       <param name="enabled0" type="int" value="1"/>
       <param name="enabled2" type="int" value="1"/>
       <param name="sumBs" type="float" value="0.0"/>
       <param name="axis" type="int" value="0"/>
       <product name="f2ww"/>
       <product name="f2ww"/>
       <product name="f2wr"/>
     </action>
   </successor>
 </rule>
```

Figura 9.6: Primera part de XML generat al efectuar la còpia de la finestra de la Figura 9.4).

```
<rule>
  <predecessor name="f2ww">
  </predecessor>
  <successor>
    <action name="bE:Insert" nodename="f2ww">
      <param name="sz" type="float" value="133.235992"/>
      <param name="sy" type="float" value="1.0"/>
      <param name="sx" type="float" value="1.0"/>
      <param name="tz" type="float" value="-47.5999985"/>
      <param name="tx" type="float" value="0.0"/>
      <param name="ty" type="float" value="0.0"/>
      <param name="rx" type="float" value="0.0"/>
      <param name="ry" type="float" value="0.0"/>
      <param name="rz" type="float" value="0.0"/>
      <param name="relative" type="str" value=""/>
      <param name="relx" type="int" value="1"/>
      <param name="asset" type="str"
     value="C:/Users/Xanatos/Desktop/Models/Raco/Assets/f2ww.bgeo"/>
      <param name="rely" type="int" value="1"/>
      <param name="relz" type="int" value="0"/>
    <product name=""/>
    </action>
  </successor>
</rule>
  <connections>
    <param dest="f2ww" src="primitiveProducerNode-N_p"/>
  </connections>
  <guiparms>
    <param name="actualNode"
           value="/obj/Raccolet/rightWing/primitiveProducerNode-N_p"/>
    <param name="primitiveNum" value="1"/>
  </guiparms>
  <cpstatusparms>
    <param name="tagsInSourcePrim"
           value="['mass2','facade2','__Comp2__2','__Subdiv37__1',
           'f2windows','f2ww','__Subdiv39__1']"/>
    <param name="SrcPrimSize"
           value="(324.9837646484375,309.00009155273437,0.0)"/>
  </cpstatusparms>
</CGAShape>
```

Figura 9.7: Segona part de XML generat al efectuar la còpia de la finestra de la Figura 9.4).

Un cop que es vol carregar aquest fitxer, és la classe XMLShapeFileReader la que s'encarrega de transformar aquesta informació en el sub-arbre de nodes principal. Per fer-ho es crea un iterador que va obtinguent els valors de cadascun dels apartats i segons el tipus va creant els nodes. Aquests nodes es desen en una carpeta temporal creada a l'arrel del projecte en la ruta '/obj/XMLCLTemp/' de manera que es tinguin tots els nodes sense conflictes de possibles noms repetits. Un cop es tenen els nodes en aquesta carpeta es passarà llegir totes les connexions entre ells. Per aquesta raó és tant important preservar el nom dels nodes ja que aquest resulta

un identificador clau a l'hora de connectar-los. Un cop realitzada la connexió es llegeixen tots els paràmetres del node GUI i finalment els paràmetres pertinents al CopyPasteStatus per tal que tota l'estructura sencera es trobi en l'estat que estaria si l'usuari fes un copy sense desar la informació en el fitxer XML. Un cop fet tot això, sub-graf ja connectat en la carpeta temporal i amb totes les dades carregades es copia en el buffer de Houdini. En aquest punt, l'usuari ja pot seleccionar allà on vol enganxar la geometria i poder fer-ho.

9.3 Classe commonNodeAncestor

Aquesta classe ha sigut implementada al mateix temps que es va implementar la classe commonPrimAncestor però finalment no s'ha fet servir en el marc del projecte. Tot i això, pensant en el futur del projecte és una classe que resol un problema important i probablement es necessitarà en futures funcionalitats.

L'algorísmica de la classe commonPrimAncestor és relativament més senzilla que la de la classe commonNodeAncestor, ja que quan es treballa amb primitives, sempre s'ha parlat en el transcurs d'aquest document, ***d'arbre de primitives***. En canvi quan es parla de nodes, el seu àmbit normalment és el d'un ***graf de nodes***. Tot i que en el procés de selecció de de l'usuari, el tractament principal es fa sobre l'arbre de primitives, s'ha implementat una classe homònima però que en comptes de tractar primitives, tracta nodes. Aquesta és la classe commonNodeAncestor.

En el cas del paradigma d'un graf de nodes però és molt més complicat que el d'un arbre de primitives, ja que tot i que es tracta d'un graf dirigit, s'hereten les problemàtiques típiques dels grafs, com per exemple que un node pot tenir múltiples "pares" o ancestres.

Per a poder implementar aquesta classe s'ha creat una estructura de graf de nodes dins d'un nou projecte de Houdini que ha ajudat a debugar i finalitzar la implementació d'aquesta classe. En la Figura 9.8 es pot veure el graf de nodes del que s'està parlant.

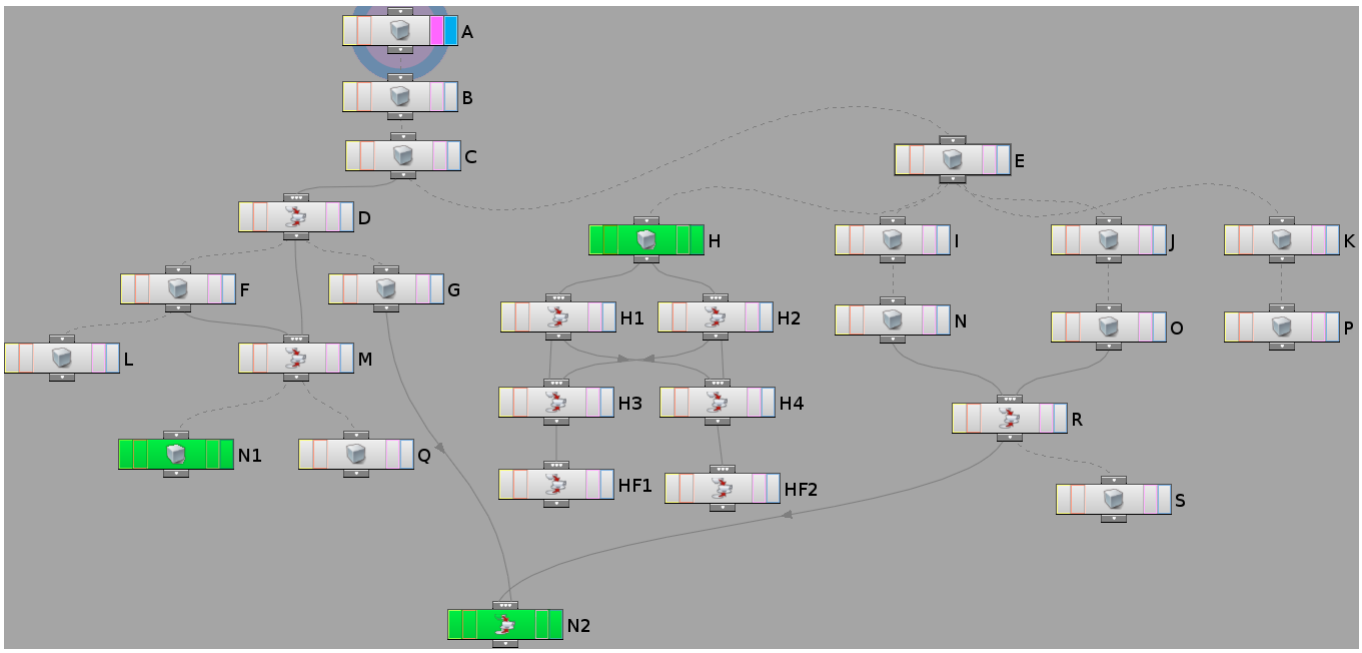


Figura 9.8: Captura de Houdini del graf de nodes de prova per debugar la classe commonNodeAncestor.

Un cop mostrada aquest graf de prova es passarà a explicar l'algorísmica de manera resumida, de l'algorisme que permet cercar l'ancestre comú de dos nodes. Per fer-ho ajudarà la imatge de la Figura 9.9 on es pot observar un exemple on es s'ha seleccionat el node "N1" i "N2" per a trobar el node ancestre comú d'aquests dos nodes.

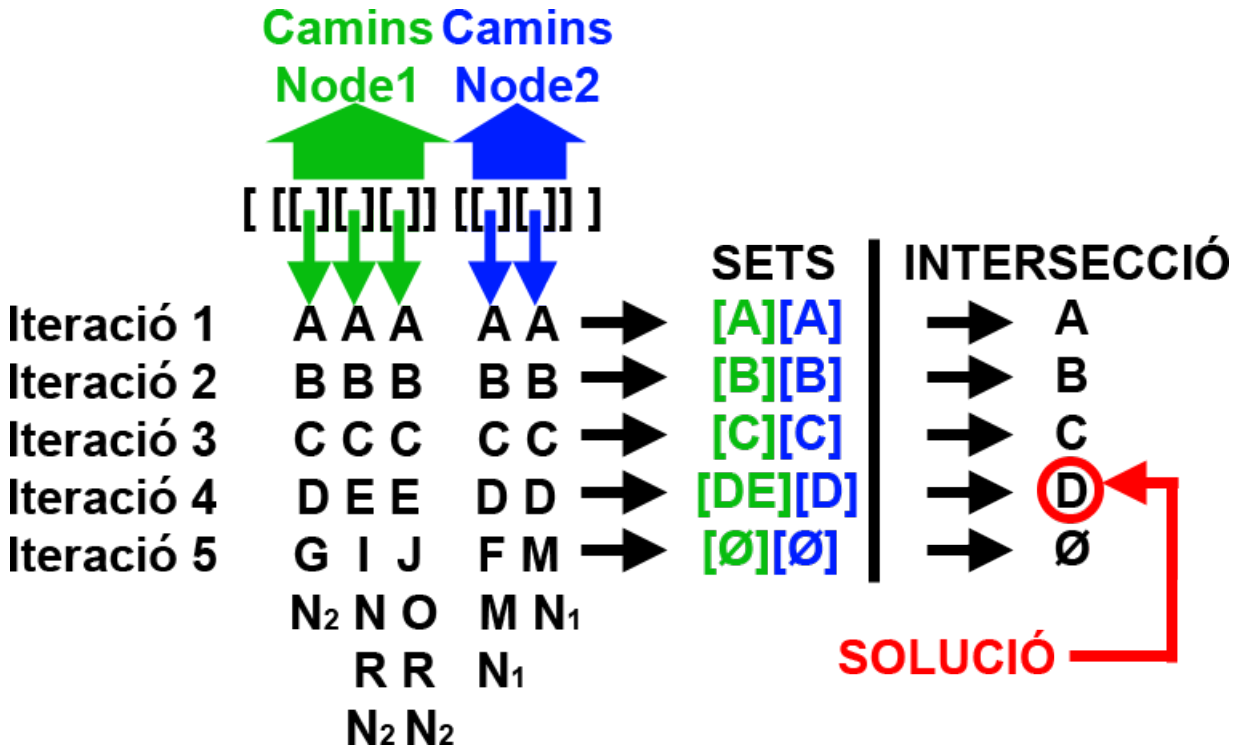


Figura 9.9: Exemple d'execució de l'algorisme del commonNodeAncestor seleccionant els nodes "N1" i "N2".

Com es pot observar, primer de tot es crea una llista amb tots els camins possibles des de "N1" al node arrel (Verd), i tots els camins possibles des de "N2" al node arrel (Blau).

Cadascun dels camins creats es desa en forma de pila, per tal que, en cada iteració, es vagi desempilant el top de cadascuna de les piles. Quan es té en una iteració tots els tops de les piles, dins de cada conjunt de camins del node "N1" i "N2" s'eliminen els repetits, creant un set en cada iteració que es pot observar a la dreta. Com es pot veure un set conté tots els nodes no repetits de cada conjunt de camins de cada un dels dos nodes. Un cop es tenen els sets de la iteració actual de "N1" i "N2", es fa la intersecció dels dos conjunts. S'ha de dir, que tot i que en l'exemple el resultat d'aquesta operació sempre dona un sol element, no té perquè ser sempre així, ja un node pot tenir diversos ancestres comuns al mateix nivell. Un cop s'arriba a una iteració on el resultat de tot aquest procés és un conjunt buit ($\emptyset$) a les hores s'agafa la intersecció de la iteració anterior. Aquest és el node ancestre comú més proper als nodes "N1" i "N2".

Per donar un exemple del que s'ha explicat en el paràgraf anterior, sobre que un node pot tenir més d'un ancestre comú simultàniament en el mateix nivell, s'ha creat, en el graf de la Figura 9.8, un conjunt de nodes que comencen per la lletra "H". Si amb aquest exemple de graf es demanés a l'algorisme de la classe commonNodeAncestor, que digués quin és l'ancestre comú dels nodes "HF1" i "HF2", la solució que dona és una llista que conté ["H1","H2"]. Tots dos nodes es troben en el mateix nivell i són ancestres comuns.

Com es pot veure, l'exemple de la Figura 9.8 intenta contemplar el màxim de casos possibles en que es poden trobar connectats els nodes per tal d'efectuar totes les proves que han fet possible la implementació de la classe commonNodeAncestor.

9.4 Problemàtiques trobades

Tal i com es comenta en el Capítol 4, a arrel d'una problemàtica es va haver de postergar el projecte fins a Setembre. Aquesta problemàtica va sorgir al final de la implementació del la funcionalitat d'enganxar geometria. Es va prendre consciència del problema quan al efectuar enganxades de geometria es desequilibrava l'estructura de l'edifici sense cap mena de lògica, arribant a veure resultats com els que es poden observar en la Figura 9.10.

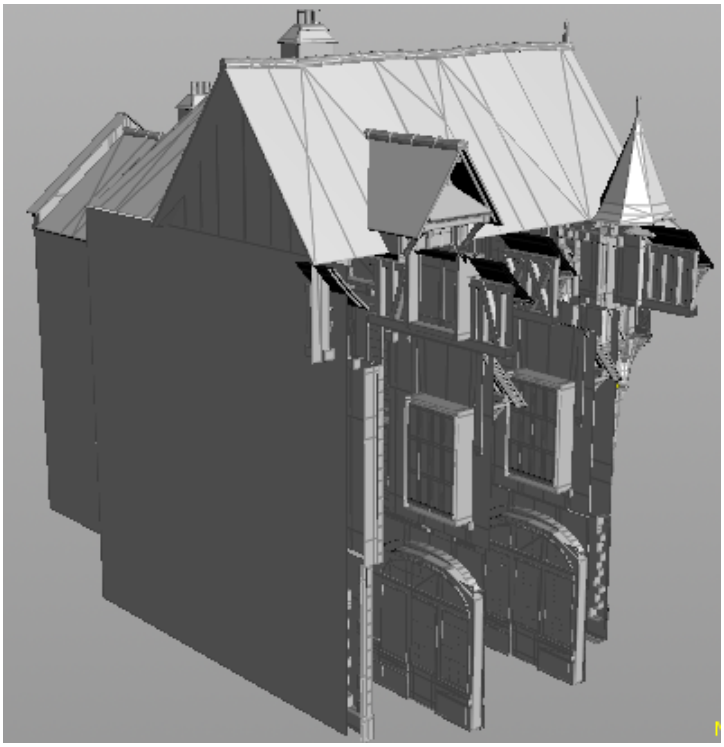


Figura 9.10: Exemple de façana de la casa "Racolette" a finals de la implementació de la funcionalitat d'enganxat.

A més d'aquesta reacció incontrolada, també es tenien problemes amb el buffer de Houdini, ja que en el procés de copiar i enganxar, al obrir l'edifici allà on es volia enganxar la geometria, Houdini no buidava el buffer amb l'edifici anterior. Curiosament sortien tots dos edificis solapats fent impossible el seguiment de quina geometria s'estava seleccionant i veure els resultats de l'enganxat.

A aquesta problemàtica no es va presentar quan s'executava sobre un portàtil Mac. Això va induir a pensar que el problema radicava o en el sistema operatiu, la versió de Houdini, o la tarja gràfica. Aquests tres components eren claus per trobar l'origen del problema. Per fer-ho es van prendre les següents decisions.

- Instal·lar diferents versions de Houdini
- Testejar entre processadors Intel i AMD.
- Testejar la versió de Houdini que hi havia en el Mac en diferents sistemes operatius.
- Testejar una targeta gràfica que no fos de la marca ATI.

9.4.1 Testejant diferents processadors

La majoria del projecte s'ha desenvolupat en l'ordinador de sobretaula que s'ha descrit en el Capítol 2. Per aquestes proves es va poder fer servir un nou ordinador, en aquest cas un portàtil amb les següents característiques:

CPU	Intel DualCore i5 540M 2800Mhz (21x133)
Ram	4Gb DDR3 SDRAM (7-7-7-20 533 MHz)
Gràfica	Intel HD Gràphics ATI Mobility Radeon HD 5470
Disc Dur	ST9500420AS 500Gb 7200rpm
Pantalla	Integrada AU Optronics B133XW02 13.3”
Sistema Operatiu	Windows 7 Enterprise x64

Tot i el canvi de processador malauradament no es va trobar l'origen de la fallada i es va descartar aquesta opció.

9.4.2 Testejant diferents sistemes operatius

Per tal de testejar diferents sistemes operatius es va fer servir el VirtualBox el qual va portar problemes ja que requeria de l'activació d'un paràmetre de la BIOS anomenat ”vt-x/amd-v” per a l'emulació en temps real que, per desgràcia, no tots els processadors tenen disponible. Finalment es va optar per fer servir el VMWare 9.0.2-1031768 i es van instal·lar els següents sistemes operatius:

- Windows 8 AIO x64 (emulat)
- Windows 8 Pro x64 (emulat)
- Windows 7 Enterprise x64
- Windows XP Pro SP2 x64 (emulat)
- Linux Debian Squeeze amd64

Després de testejar tots els sistemes operatius es va comprovar que no era l'arrel del problema.

9.4.3 Testejant una tarja gràfica diferent

Mentre es va dur a terme les proves amb diferents sistemes operatius, les quals van durar uns quants dies, es va poder obtenir prestada una tarja gràfica antiga però vàlida per a realitzar la prova, de la marca Nvidia model 7300GT. Malauradament després de testejar amb aquesta targeta gràfica sobre els diferents sistemes operatius un altre cop, es continuava sense arreglar el problema i sense trobar quin era el seu origen.

9.4.4 Testejant diferents versions de Houdini

Mentre es realitzaven les proves anteriors, es va descarregar la versió d'aprenent de Houdini 11.0.469-win64 per veure si hi havia diferència amb la versió 12.5.427-win64. El resultat va ser pitjor que fent servir la versió més nova. La quantitat d'errors i problemes va ser esgarrifosa. Això va donar a pensar que hi podia haver algun problema entre versions de Houdini, ja que no era normal tanta divergència de resultats entre versions.

9.4.5 Solució del problema

Finalment vista la última experiència amb les versions de Houdini, es va optar per repassar el nucli de **building**Engine. L'origen del problema va residir en el nucli de **building**Engine en la llibreria digital d'assets anomenada ”**building**Engine.otl” la qual es trobava desfasada respecte a la versió de Houdini que s'estava fent servir. Amb l'ajuda del director del projecte es

va poder actualitzar el nucli de **building**Engine i els errors van desaparèixer. Malauradament ja era tard per acabar la documentació i encara s'havia d'acabar d'implementar diverses coses. Per aquesta raó es va decidir postergar l'entrega del projecte al Setembre de 2013.

Capítol 10

Resultats

10.1 Resultats

En aquest capítol es mostren els resultats obtinguts amb aquest projecte. Els resultats que es mostren en la majoria de figures estan visualitzats directament des de l'àrea de treball de Houdini.

Per mantenir un ordre coherent en el document s'ha decidit estructurar els resultats en tres categories, una per cadascun dels mòduls principals del projecte (excepte el de XML explicat ja en profunditat al capítol 9): Interfície, acceleració de la interacció d'usuari i finalment el Copy&Paste.

10.1.1 Interfície d'usuari

La interfície d'usuari es divideix en dos mòduls, un és el Shelf que es pot observar en la Figura 3.1.

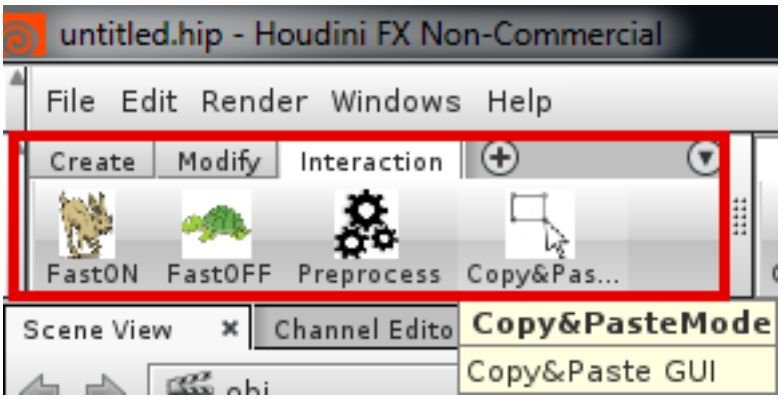


Figura 10.1: Exemple de la interfície d'usuari integrada a Houdini principal.

Aquesta part de la interfície permet a l'usuari realitzar les següents operacions:

- **Botó FastON:** Permet activar el mode ràpid d'interacció d'usuari realitzant un canvi de la geometria per textures precalculades.
- **Botó FastOFF:** Permet desactivar el mode ràpid d'interacció d'usuari realitzant un canvi de les textures precalculades a la geometria original.

- **Botó Preprocess:** Permet realitzar el processament previ de l'edifici per tal que es puguin fer servir els botons de FastON i FastOFF. D'altra manera aquests botons no realitzen cap funcionalitat.

- **Botó Copy&Paste:** Crea i activa la visualització del node *GUI* que s'explicarà a continuació, per tal que l'usuari pugui efectuar totes les tasques de seleccionar, copiar i enganxar geometria.

Un cop l'usuari prem el botó de *"Copy&Paste"* apareix el menú del *GUI* que permet a l'usuari seleccionar, copiar, enganxar, carregar i gravar fitxers XML de seleccions de geometria. Es pot observar aquest menú en la Figura 10.2.

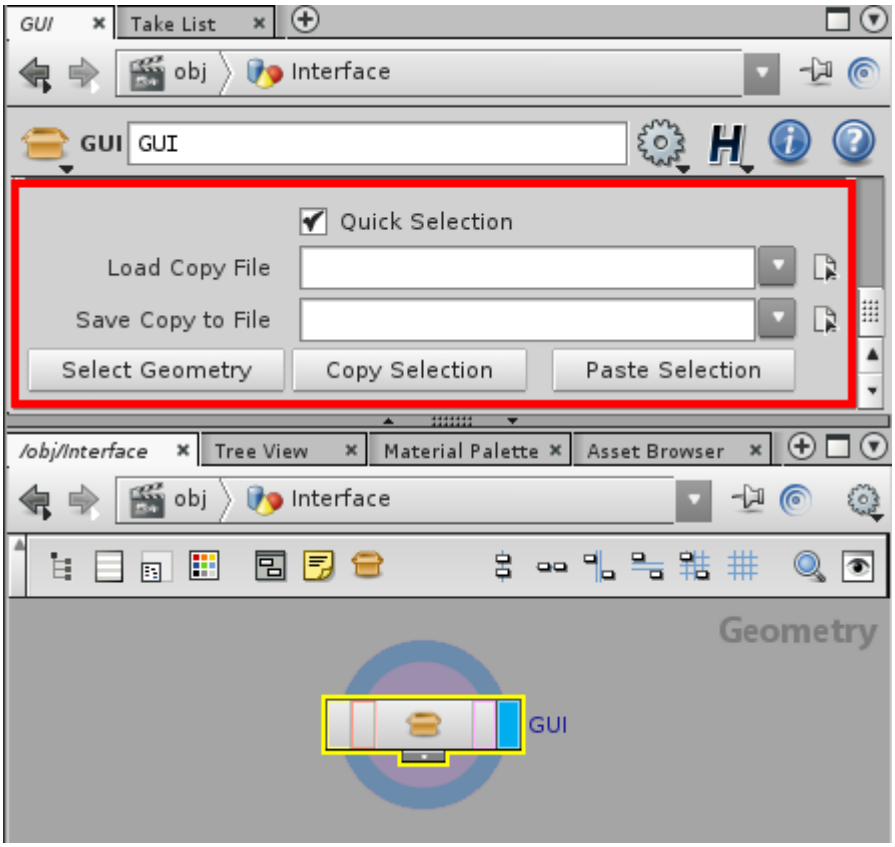


Figura 10.2: Exemple de la interfície d'usuari del node GUI per al Copy&Paste.

Quan l'usuari visualitza per primer cop aquest menú, només apareixen els camps de carregar i desar en fitxer, i el botó seleccionar. A mesura que l'usuari efectua operacions, van apareixent i desapareixent els botons segons les operacions que pugui efectuar en el moment. Això s'ha fet d'aquesta manera ja que no té sentit mostrar per exemple, el botó d'enganxar, quan encara no s'ha copiat res.

10.1.2 Acceleració de la interacció

Mitjançant els botons del Shelf mostrats anteriorment, l'usuari pot activar i desactivar el mode d'acceleració d'interacció, però abans ha de prémer el botó de precalcular l'edifici. En la Figura 10.3 es pot veure els canvis efectuats al realitzar els càlculs.

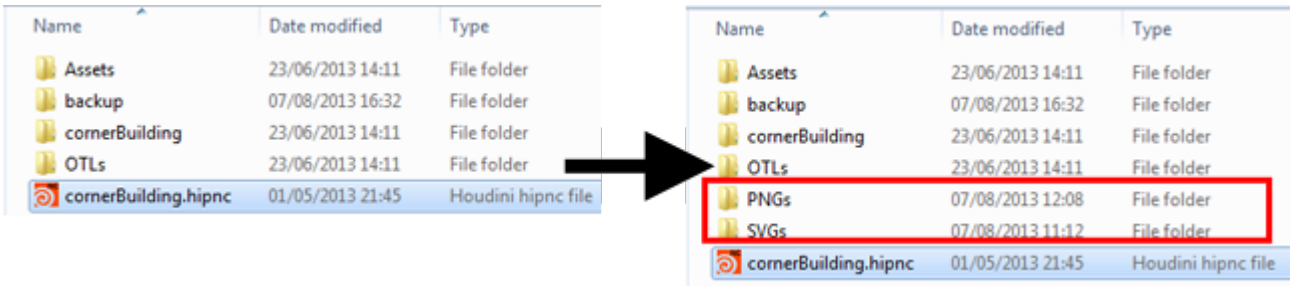


Figura 10.3: Exemple dels canvis en les carpetes del projecte quan es pre-processa l'edifici.

Com es pot veure en la Figura 10.3 al realitzar els càlculs es generen dues carpetes més dins de la carpeta del projecte sobre el que efectuem aquests càlculs. Una carpeta conté totes les imatges en format "SVG", el qual és un format d'imatge que no té pèrdues, això vol dir que es pot escalar tant com es vulgui sense perdre qualitat. I l'altre carpeta conté les mateixes imatges transcodificades en format "PNG", aquest format sí té pèrdues però requereix d'un còmput molt menor a l'hora de tenir-lo carregat en el viewport de Houdini.

Un cop es tenen totes les imatges de l'edifici precalculades, es pot activar i desactivar el mode ràpid d'interacció d'usuari. En la Figura 10.4 es pot veure com queda el graf de nodes de l'edifici en els dos estats.

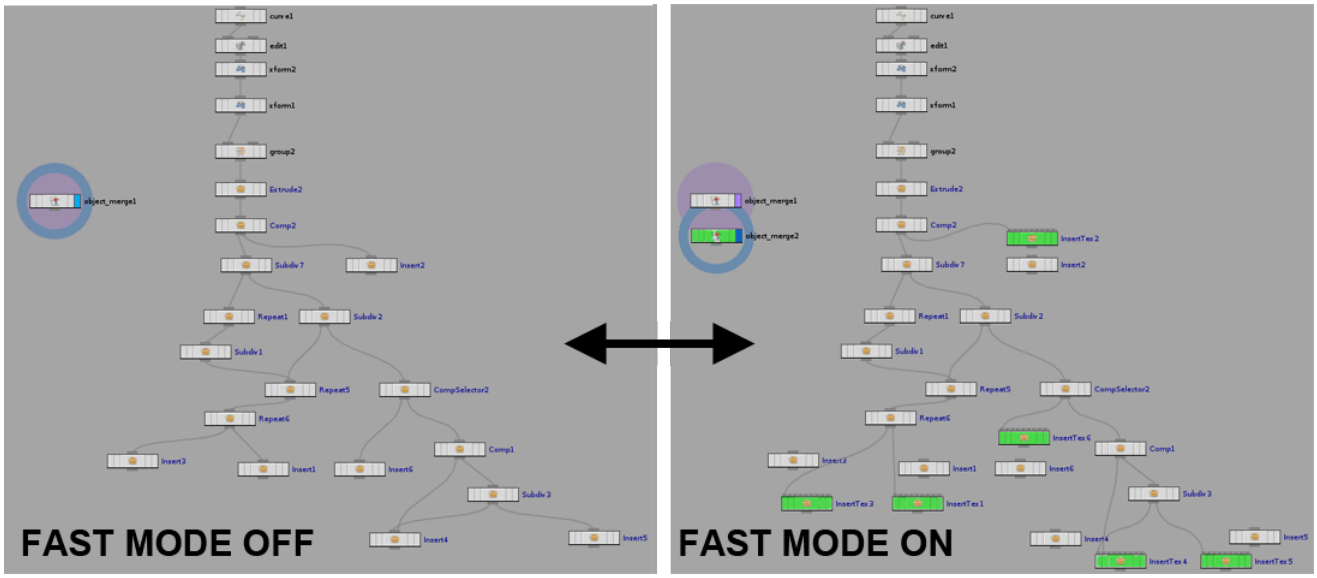


Figura 10.4: Exemple dels canvis en el graf de nodes al activar i desactivar el mode d'interacció ràpid (Fast mode).

Al activar el mode ràpid, es creen tants nodes "InsertTex" com nodes "Insert" hi hagi, i s'intercanvien les connexions. Al desactivar-lo tornen a connectar-se tots els nodes "Insert" i s'eliminen els nodes "InsertTex" així com el node "object_merge" que sustenta la visualització

dels nodes "InsetTex".
A continuació es pot observar el resultat en la Figura 10.5 de com queda l'edifici amb el mode activat i desactivat.

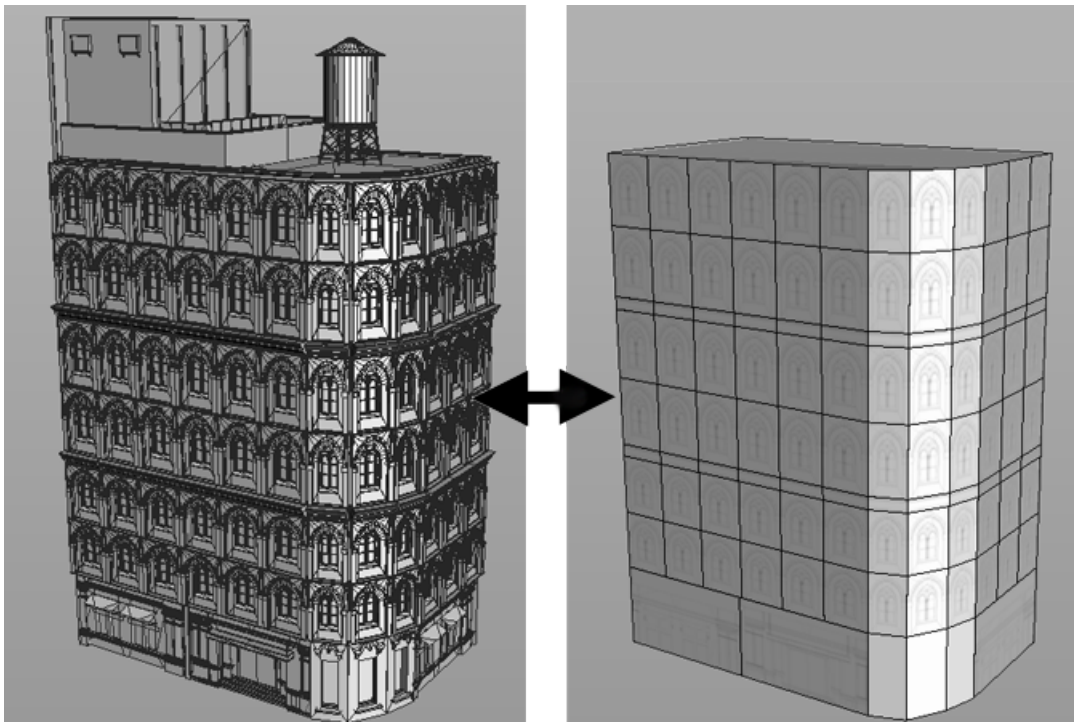


Figura 10.5: Exemple dels canvis en l'edifici al activar i desactivar el mode d'interacció ràpid (Fast mode).

10.1.3 Tests de rendiment

Per tal de testejar la millora en el rendiment amb aquest mètode d'acceleració, s'ha realitzat un codi mostrat en el capítol 9, on es va augmentat l'altura d'un edifici. A cada iteració s'augmenta una fracció aquesta altura i a causa d'aquest canvi, es van incorporant nous pisos i per tant més geometria. A cada iteració també se l'hi obliga a Houdini a re-calcular tot l'edifici sencer. És en aquest punt on es nota que com més geometria hi ha, més triga a processar-ho.

En la Figura 10.6 es pot observar el resultat visual del mateix test aplicat a un edifici sense acceleració (Geometry) i amb l'acceleració activada (Textures).

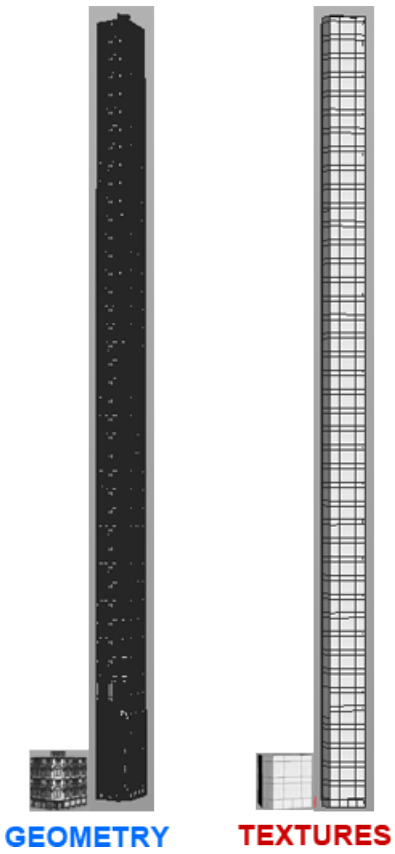


Figura 10.6: Exemple dels canvis visuals en l'edifici al realitzar el test de rendiment, sense i amb acceleració.

A continuació en la Figura 10.7 es pot veure representat gràficament, el temps que ha trigat Houdini en re-calcular tot l'edifici en cada una de les iteracions a mesura que s'anava augmentant l'alçada.

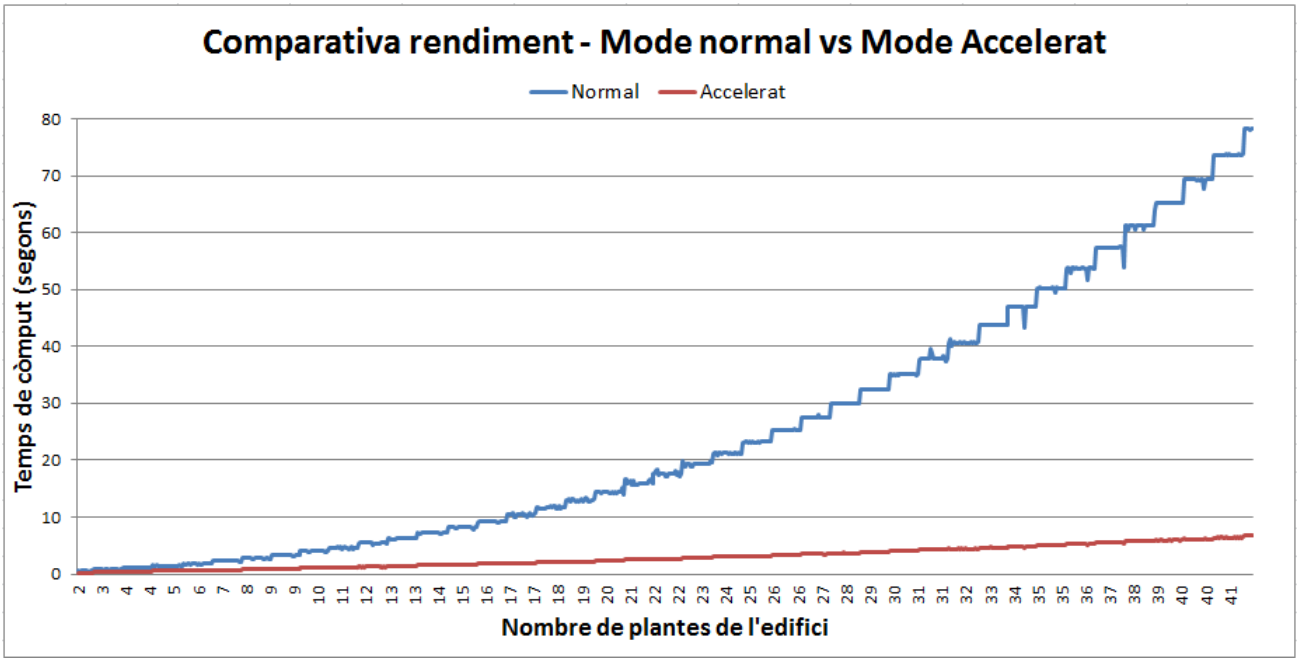


Figura 10.7: Resultat del test de rendiment on es representa el temps de càlcul vers el nombre de plantes.

Com es pot observar, la millora és considerable. A més, aparentment el creixement del gràfic sense acceleració (Blau) sembla quadràtic envers l'altre (Vermell) que sembla ser lineal. Així dons queda demostrada l'eficàcia d'aquest mètode a l'hora d'augmentar la interactivitat de l'usuari.

10.1.4 Copy&Paste

Seguidament es poden veure diversos exemples de façanes que se'ls hi ha realitzat modificacions amb l'eina de Copy&Paste entre edificis.
En la Figura 10.8 es pot observar l'objectiu de l'usuari a partir dels edificis originals i sense canvis.

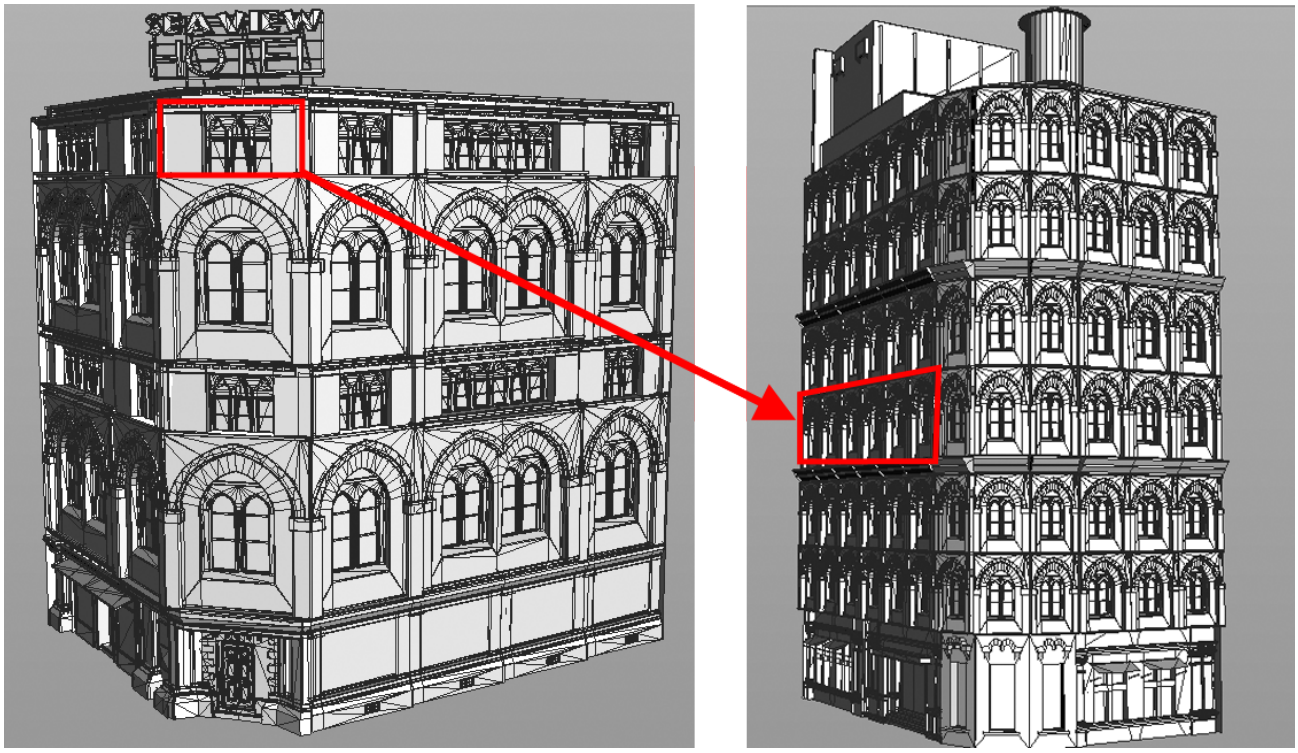


Figura 10.8: Exemple d'operació de Copy&Paste d'una finestra del hotel "SeaView" (Esquerra) i el "CornerBuilding" (Dreta).

En la Figura 10.9 es pot observar l'edifici "CornerBuilding" amb els canvis de la Figura 10.9 efectuats en la façana.

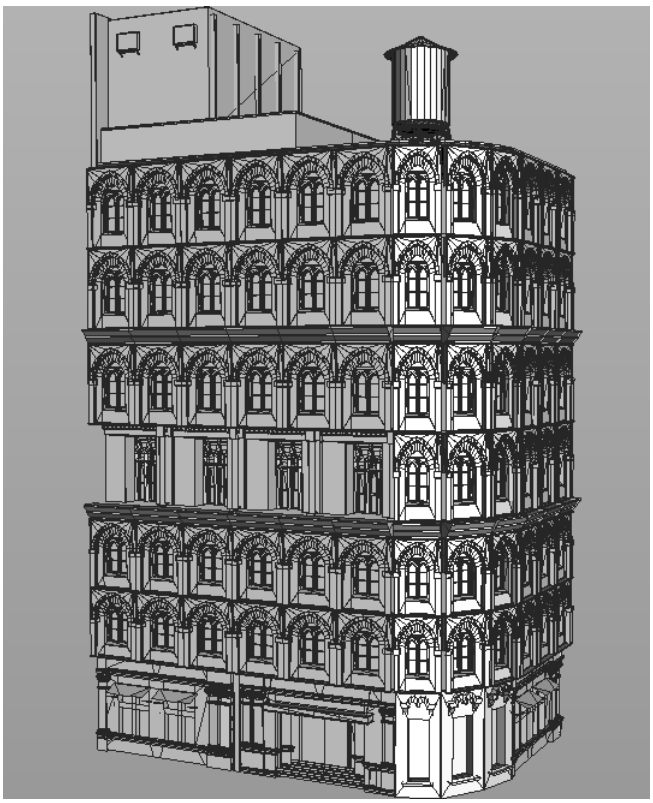


Figura 10.9: Exemple del resultat de la operació de Copy&Paste de la Figura 10.8.

Un altre exemple entre dos edificis, aquest cas l'edifici "SeaView" i la casa "Racolette" es pot observar en la Figura 10.10 quina és la intenció de l'usuari amb els edificis originals.

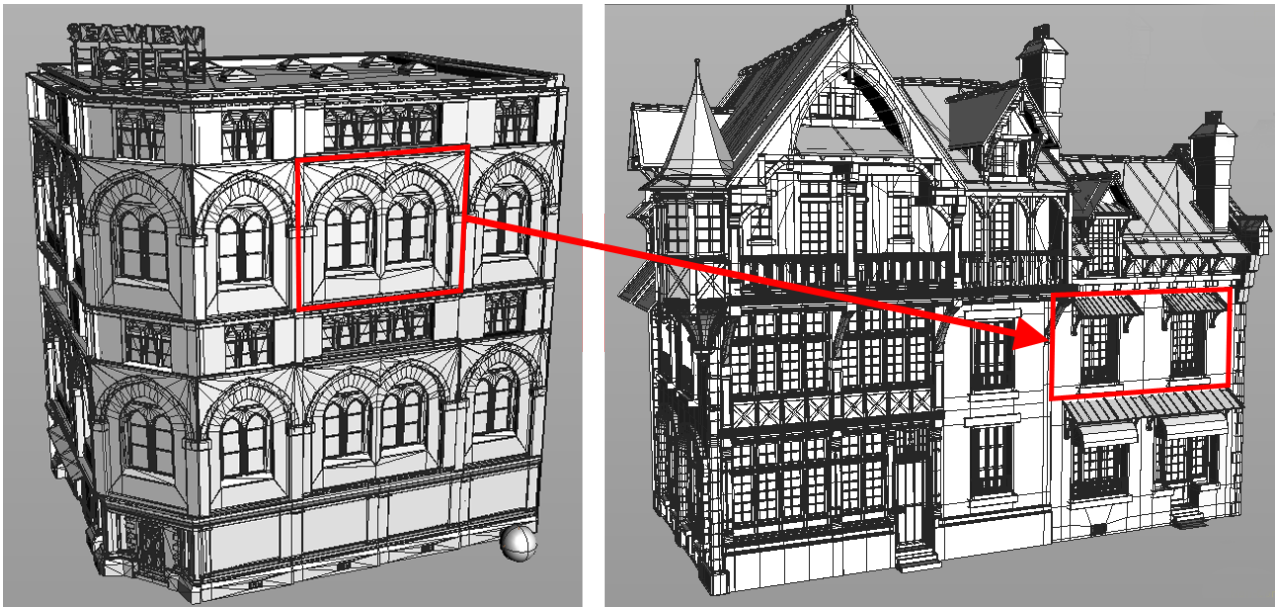


Figura 10.10: Exemple d'operació de Copy&Paste d'una finestra del hotel "SeaView" (Esquerra) i la casa "Racolette" (Dreta).

En la Figura 10.11 es poden veure els canvis efectuats en la façana després d'haver fet servir l'eina.

Figura 10.11: Exemple del resultat de la operació de Copy&Paste de la Figura 10.10.

Capítol 11

Conclusions

Els objectius d'aquest projecte, que s'han explicat en la secció 1.3, s'han pogut assolir completament.

S'ha aconseguit realitzar una interfície d'usuari integrada a Houdini amb la qual es poden realitzar totes les tasques que s'havien plantejat.

S'ha pogut fer una millora considerable en quant a rendiment en la interacció de l'usuari gràcies a l'intercanvi de la geometria per imatges en format 'PNG'. Aquest intercanvi és reversible i es fa automàticament de manera transparent a l'usuari. És a dir, l'usuari no necessita tenir nocions dels processos tècnics que es duen a terme en aquestes operacions.

També s'ha aconseguit dotar a **building**Engine de l'eina de copiar geometria i enganxar-la allà on l'usuari vulgui, i integrar-ho a Houdini. D'aquesta manera l'usuari no ha de tenir coneixements de com Houdini tracta l'arbre de nodes o l'equivalent en arbre de primitives, ni haver de navegar per elles. L'usuari pot seleccionar un tros de geometria després d'haver premut un botó, copiar prement-ne un altre, seleccionar amb el ratolí allà on vol enganxar-la i realitzar l'acció final d'enganxar amb un altre botó.

A més s'ha construït una nova eina que permet desar en fitxer en format XML aquelles còpies de geometria que realitza l'usuari, de manera que les pot aprofitar més endavant tot i tancar Houdini, tornant a carregar-les des de fitxer. Això encara fa més versàtil l'eina i obre tot un ventall de possibilitats que es comentaran més endavant en el capítol 12 de treball futur.

A continuació es recorden els objectius i es comentarà la feina assolida:

- Aprendre i manejar el llenguatge de programació Python.
  Com s'ha vist al llarg del projecte i després de la implementació de totes les eines descrites es pot donar per après el llenguatge de programació imperativa Python.
- Estudi de la plataforma 3D "Houdini" i de les llibreries per la incorporació de scripts Python.
  S'ha estudiat en profunditat la plataforma de Houdini i gràcies a això s'ha pogut elaborar la interfície d'usuari i s'han pogut dur a terme totes les proves directament sobre els nodes i les primitives, interactuant des del Shell de Houdini mitjançant python en la modificació del graf de nodes. A més per dur a terme aquesta tasca s'havien de conèixer en profunditat les llibreries per a la programació de scripts en python, per tant es pot donar aquesta tasca per assolida.
- Estudi del **building**Engine, que és el mòdul inclòs al **skyline**Engine encarregat de la definició d'edificis.

L'estudi d'aquest mòdul ha sigut clau per a la realització d'aquest projecte ja que proporciona tots els nodes involucrats en la selecció, còpia i enganxat de la geometria, ja que aquesta tasca es basa també en el conjunt de regles per a l'elaboració d'edificis sobre **building**Engine. Per tant s'ha complert amb aquesta tasca completament.

- Desenvolupament de les llibreries bàsiques per fer recorreguts configurables a l'estructura de dades generada que representa els edificis.
  Com s'ha explicat en el capítol 8, s'ha hagut de construir diverses llibreries relacionades amb el recorregut de nodes i primitives que generen un edifici per tal de poder realitzar la selecció de geometria així com el seu intercanvi per textures. Per tant s'ha realitzat amb èxit aquesta tasca.

- Desenvolupament d'un sistema de selecció de la part de façana a copiar.
  Gràcies a les classes *copyPasteOps* i *commonPrimAncestor*, explicades en la subsecció 8.4.4.2, s'ha aconseguit desenvolupar un sistema de selecció de geometria de la façana d'un edifici fet amb el mòdul de **building**Engine.

- Algorisme de reconeixement de les primitives involucrades en la generació de la part seleccionada.
  Com s'ha comentat en la subsecció 8.4.4.2, s'ha realitzat una classe anomenada *commonPrimAncestor* que cobreix l'algorisme de reconeixement de primitives involucrades en la generació de la part seleccionada de manera que aquesta tasca s'ha resolt amb èxit.

- Emmagatzemament dels nodes (operacions geomètriques) participants.
  S'ha construït tot un mòdul encarregat de l'emmagatzemament dels nodes participants bolcant-los en fitxers en format XML amb tota la informació necessària per a poder-los reproduir més endavant. Aquest mòdul s'ha explicat en profunditat en el capítol 9.

- Copia dels nodes.
  Mitjançant la funció de còpia explicada en la subsecció 8.4.4.2 de la classe *copyPasteOps*, s'ha aconseguit emmagatzemar els nodes seleccionats en el buffer de Houdini amb èxit.

- Selecció de la part corresponent del graf de destí.
  Per a la realització d'aquest mòdul s'ha re-aprofitat la funcionalitat de seleccionar geometria de la façana ja que l'algorísmica ha resultat ser molt semblant. Gràcies això, el codi per a poder realitzar seleccions és més robust i re-aprofitable, i per tant més fàcil de mantenir i integrar-hi noves funcionalitats en el futur. Per tant també s'ha realitzat aquesta tasca amb èxit.

- Reescriptura del graf de destí per incorporar-se les parts copiades.
  Ha sigut una de les funcionalitats clau del projecte ha estat implementada en la classe *copyPasteOps* i s'ha explicat en la subsecció 8.4.4.2. Aquesta funcionalitat s'ha basat en l'article de Barroso et al. [3] i s'ha pogut realitzar amb èxit l'algorísmica de la reescriptura del graf destí per incorporar la geometria copiada.

- Desenvolupament d'optimitzacions per a la interactivitat.
  Gràcies a la implementació de la selecció de geometria s'ha pogut dotar d'una eina que ha permès seleccionar molt més ràpidament la geometria. Ja que no cal que l'usuari conegui l'estructura tècnica que fa possible la visualització de l'edifici, si no que simplement seleccionant el que vol amb el ratolí es du a terme la selecció de forma transparent. Per tant aquesta tasca s'ha pogut aconseguir amb èxit.

- Acceleracions en la interacció de l'usuari.
  Aquest mòdul ha estat implementat amb èxit permetent a l'usuari obtenir una acceleració en la interactivitat amb els edificis realitzats amb **building**Engine. La classe principal

que sustenta aquesta funcionalitat és la *transform2textures* i s'ha explicat en profunditat en la subsecció 8.4.4.3.

- Documentació del treball realitzat.
  Aquesta documentació és la prova que s'ha aconseguit la tasca, a més de la realització d'un petit manual d'usuari per a l'ús de les eines construïdes.

Capítol 12

Treball Futur

El treball realitzat en aquest projecte permet un nombre de millores i ampliacions. Tant es poden dur a terme millores en les funcionalitats implementades en aquest projecte com d'altres de noves que es trobaven fora del marc del projecte o han sorgit en el transcurs del qual. Tot seguit es presente unes possibles idees que es poden dur a terme a partir del projecte.

En quant al mòdul de Copy&Paste es poden dur millores en l'eficiència de la selecció de la geometria, ja que el cost computacional de realitzar certes cerques en l'arbre de nodes i el seu equivalent en arbre de primitives, és molt elevat. Una possible solució seria que, a partir d'un llindar de primitives seleccionades, es pugui extreure un núvol de punts mitjançant els centroides de cada primitiva i amb aquest núvol poder efectuar una regressió d'un pla. Un cop es tingui el pla es pot fer una projecció ortogonal i seleccionar només les primitives que quedin en les 4 cantonades o les dues primitives que pertanyen a la diagonal teòrica del pla. D'aquesta manera es reduiria el nombre de primitives dràsticament i s'aconseguiria una millora significativa en la interacció.

Una altra millora que es podria realitzar seria la de poder seleccionar geometria d'un edifici realitzat amb **building**Engine i dur a terme la mateixa selecció que es fa ara, però en comptes de realitzar una còpia, permetre a l'usuari arrossegar la geometria entre plantes de l'edifici. D'aquesta manera, al arrossegar la geometria seleccionada, s'anirien observant els canvis en l'edifici a temps real de com aquell tros seleccionat quedaria incrustat en les demés plantes. Això seria una millora espectacular en quant a la interacció amb els edificis per part de l'usuari. Tot i que pot ser factible arribar a aquest punt de sofisticació, aquesta tasca té un cost computacional elevadíssim que probablement obligui a utilitzar llenguatges per a poder bolcar els càlculs a la GPU.

Per acabar, també s'ha de comentar el mòdul de XML ja que ara mateix augmenta versatilitat de la funció de copiar i enganxar respecte el temps, ja que es pot recuperar la còpia quan es vol. Aquesta tasca però, no funciona quan l'usuari canvia de carpeta un edifici del qual se n'ha extret una copia en format XML. Aixo es deu a que al carregar el fitxer XML, no es poden copiar els fitxers dels assets necessaris per a que el nou edifici visualitzi la geometria copiada, ja que aquests no es troben en la carpeta on apunten els paths del fitxer XML. Per això, s'hauria d'elaborar algun mecanisme d'empaquetament per tal que junt amb l'XML es desessin també la geometria necessària.
A banda d'aquesta petita problemàtica, el mòdul de XML té tot un futur per endavant. Es va escollir aquest format ja que es fa servir des de fa molts anys per a l'intercanvi d'informació entre plataformes completament disjuntes (on els llenguatges de cada plataforma poden ser completament diferents), i pràcticament tots els llenguatges de programació d'avui en dia

contenen llibreries natives per a la lectura i escriptura de fitxers en format XML amb pàrsers que permeten realitzar tractaments d'error. Això el fa un bon candidat a ampliacions i el seu futur us per a noves funcionalitats.
Entre aquestes noves funcionalitats, es podria arribar a implementar algun mètode que permetés no tant sols desar còpies de geometria, si no que podria arribar a realitzar un format més enriquit i amb més informació, que permetés compartir trossos d'edificis fets amb **building**Engine entre artistes i modeladors, amb un sistema de selecció tant simple de cares a l'usuari com el que hi ha actualment implementat. Això obriria les portes a tot un nou camp en la investigació en la edició d'edificis procedurals que permetria assolir noves fites en aquest àmbit.

Bibliografia

[1] Python Software Foundation. (2013). *Python Programming Language - Official Website.*, Recuperat el 22 de Juny 2013. URL http://www.python.org/.

[2] Side Effects Software Inc. (2013). *Houdini 3D ANIMATION TOOLS.*, Recuperat el 4 d'Abril 2013. URL http://www.sidefx.com/.

[3] Santiago Barroso, Gonzalo Besuievsky, and Gustavo Patow. Visual *copy & paste* for procedurally modeled buildings by ruleset rewriting. *Computers & Graphics*, 2013.

[4] BIBTEX. *The BibTEX resource*, Recuperat el 22 d'Agost 2013. URL http://www.bibtex.org/.

[5] Wikibooks Open books for an open world. *LaTeX/Bibliography Management*, Recuperat el 22 d'Agost 2013. URL http://en.wikibooks.org/wiki/LaTeX/Bibliography_Management.

[6] LATEX A document preparation system. *LATEX documentation*, Recuperat el 22 d'Agost 2013. URL http://latex-project.org/guides/.

[7] Wikipedia The Free Encyclopedia. *LATEX*, Recuperat el 22 d'Agost 2013. URL http://es.wikipedia.org/wiki/LaTeX.

[8] Gustavo Patow. User-friendly graph editing for procedural modeling of buildings. *Computer Graphics and Applications, IEEE*, 32(2):66–75, 2012.

[9] The Apache™ Batik Project. *Batik Java SVG TOOLKIT*, Recuperat el 10 de Novembre 2012. URL http://xmlgraphics.apache.org/batik/.

[10] Python v2.6.8 documentation. (2013). *Python documentation.*, Recuperat el 22 de Juny 2013. URL http://docs.python.org/2.6/.

Capítol 1

Nodes de cadascuna de les categories dins de ”Geometry”

Els nodes de HOudini de la xarxa de nodes ”Geometry” es classifiquen en divuit categories les quals cadascuna te un número determinat de tipus de nodes:

- **Attribute**

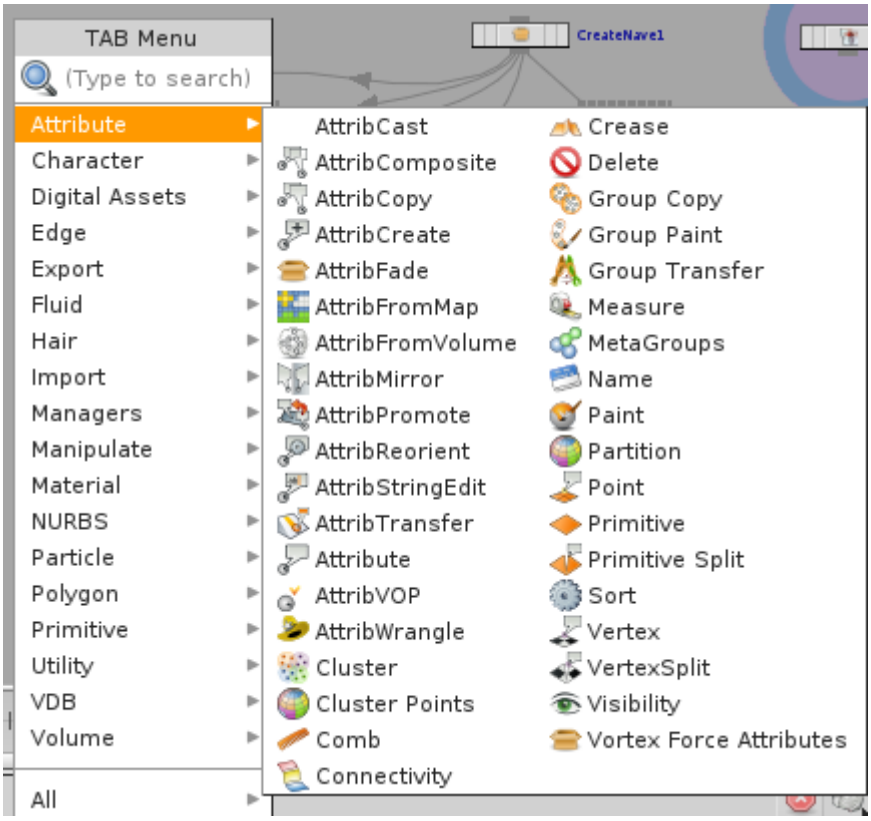


Figura 1.1: Tipus de nodes de la categoria Attribute

- **Character**

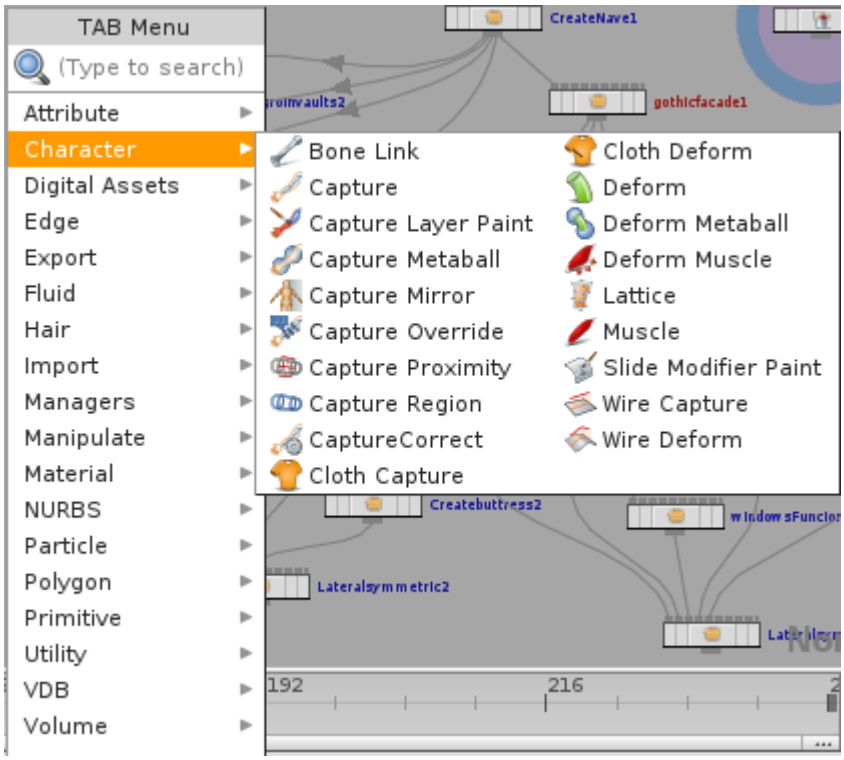


Figura 1.2: Tipus de nodes de la categoria Character

- **Digital Assets**

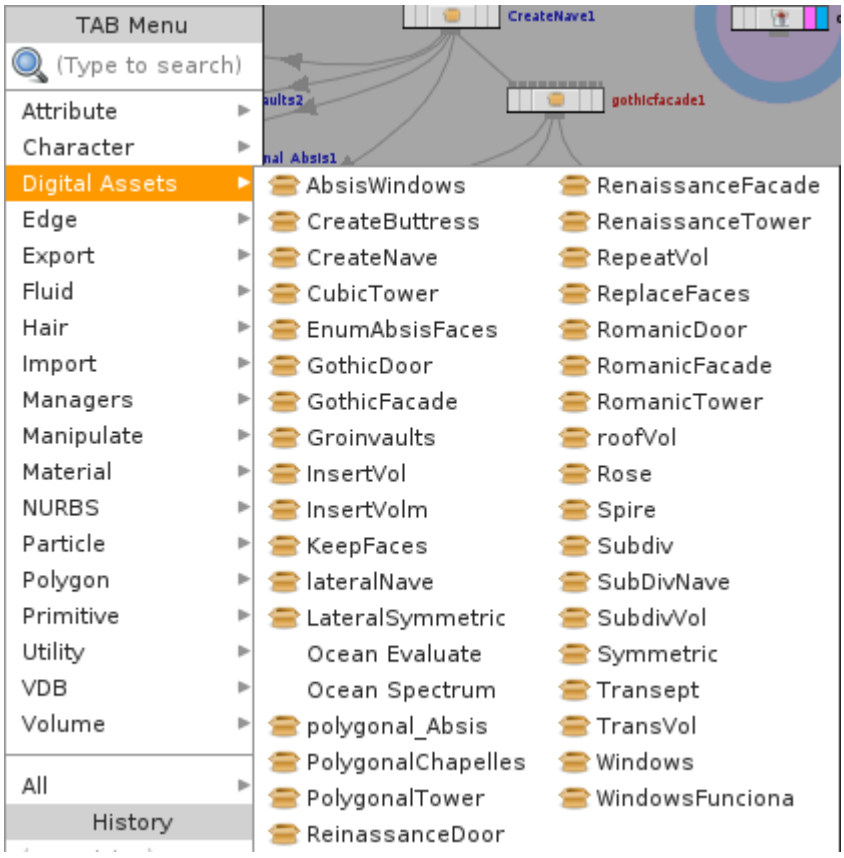


Figura 1.3: Tipus de nodes de la categoria Digital Assets

- **Edge**

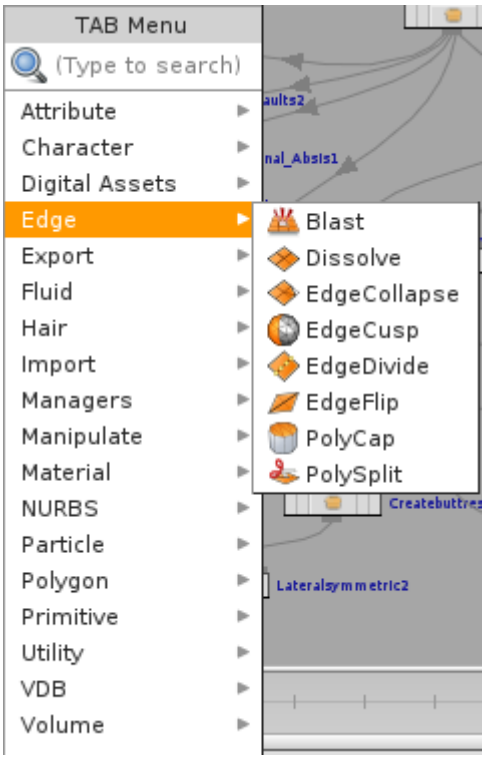


Figura 1.4: Tipus de nodes de la categoria Edge

- **Export**

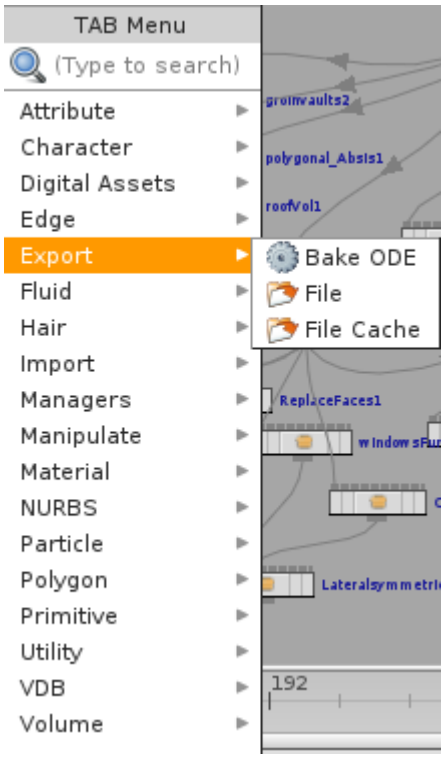


Figura 1.5: Tipus de nodes de la categoria Export

- **Fluid**

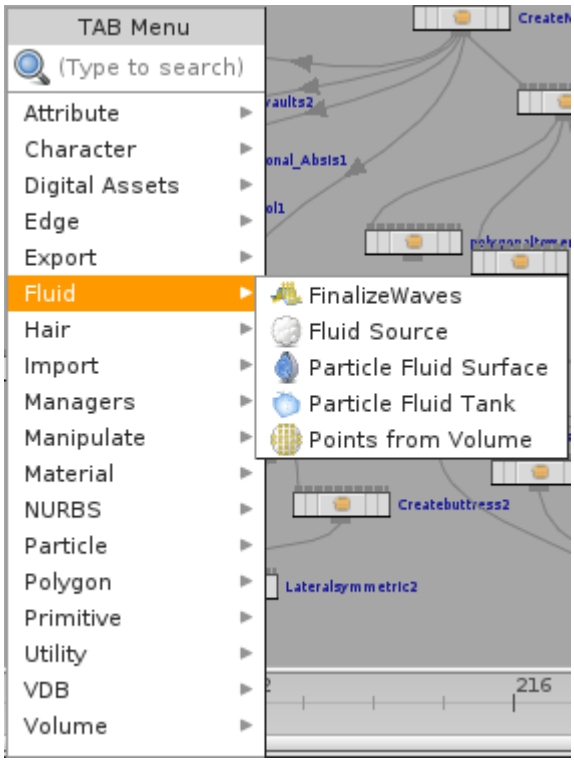


Figura 1.6: Tipus de nodes de la categoria Fluid

- **Hair**

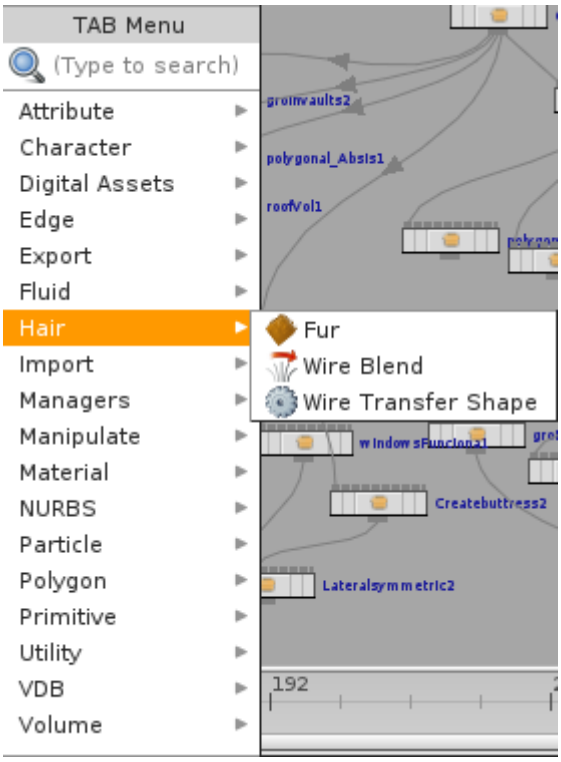


Figura 1.7: Tipus de nodes de la categoria Hair

- **Import**

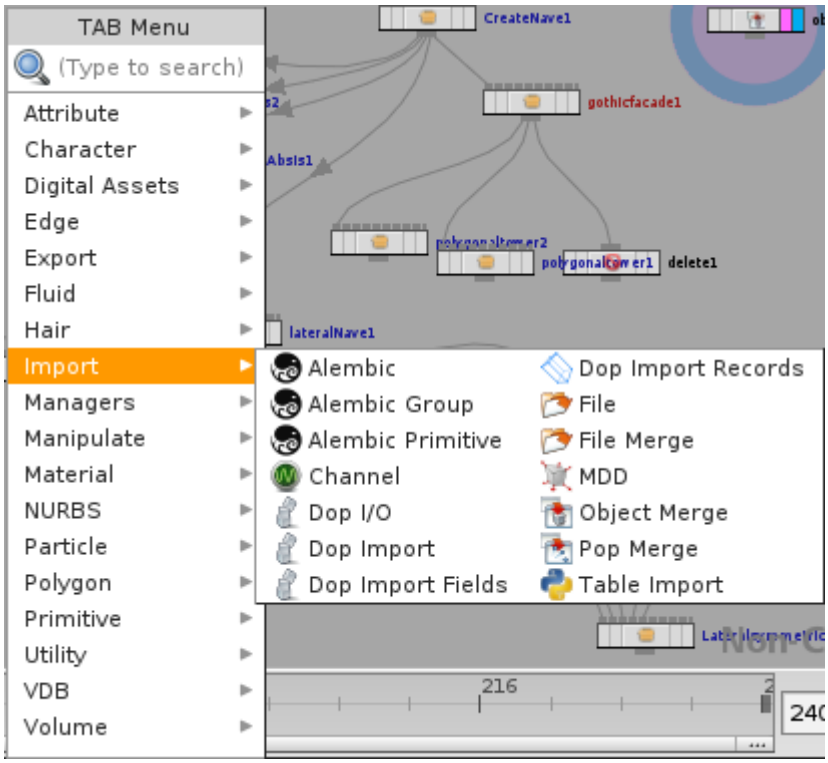


Figura 1.8: Tipus de nodes de la categoria Import

- **Managers**

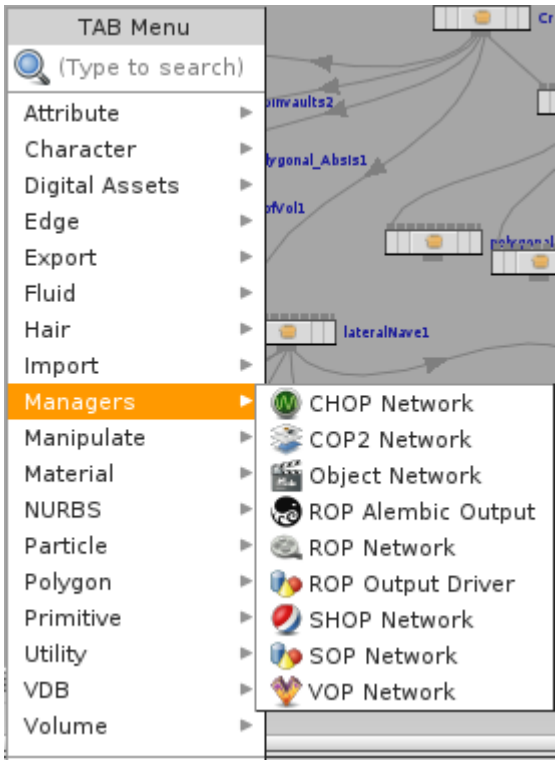


Figura 1.9: Tipus de nodes de la categoria Managers

- **Manipulate**

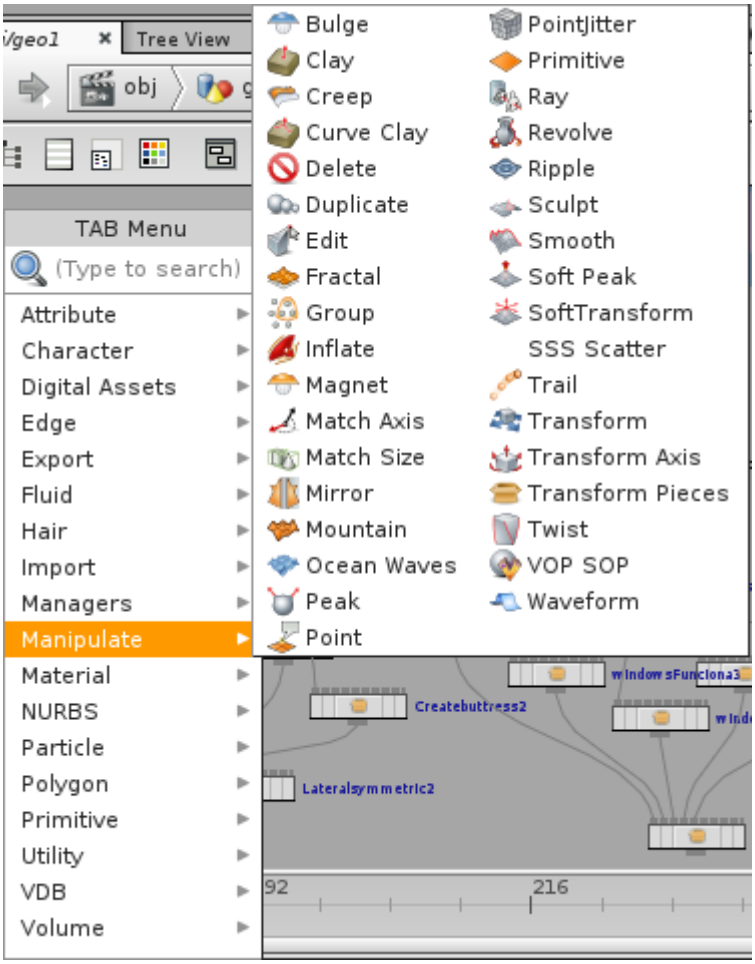


Figura 1.10: Tipus de nodes de la categoria Manipulate

- **Material**

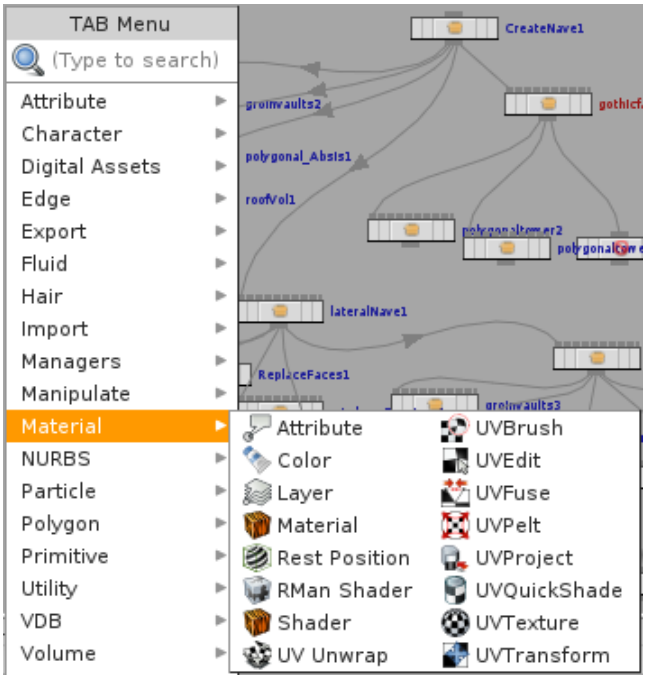


Figura 1.11: Tipus de nodes de la categoria Material

- **NURBS**

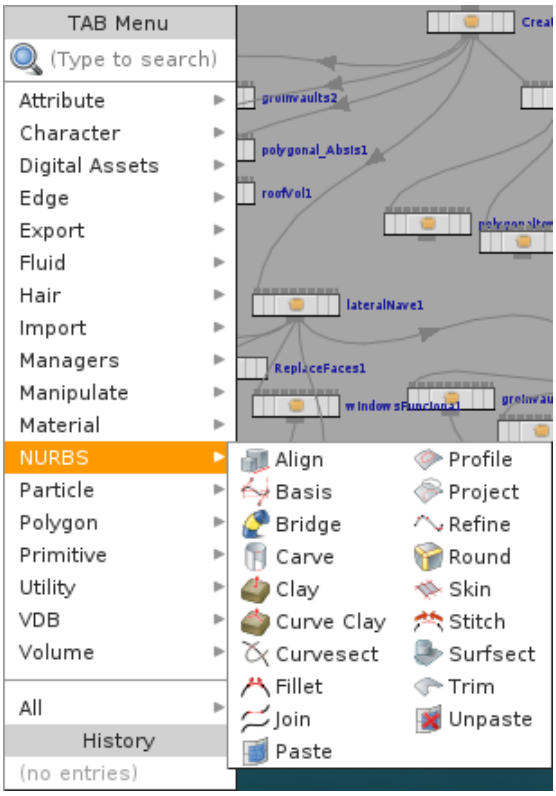


Figura 1.12: Tipus de nodes de la categoria NURBS

- **Particle**

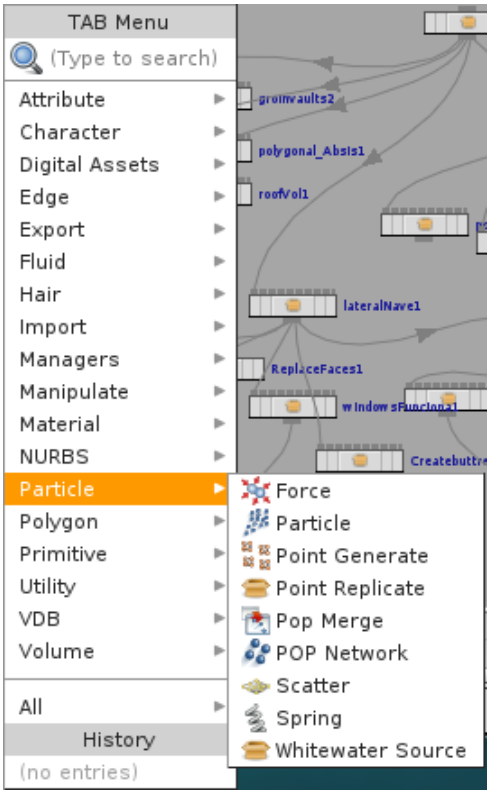


Figura 1.13: Tipus de nodes de la categoria Particle

- **Polygon**

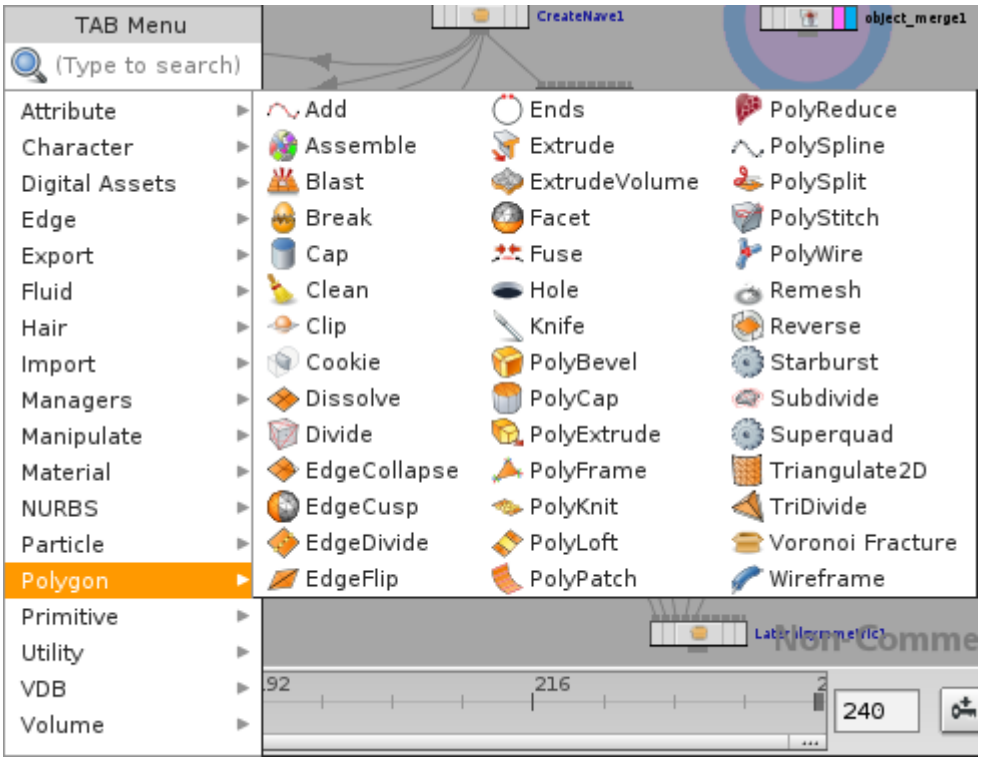


Figura 1.14: Tipus de nodes de la categoria Polygon

- **Primitive**

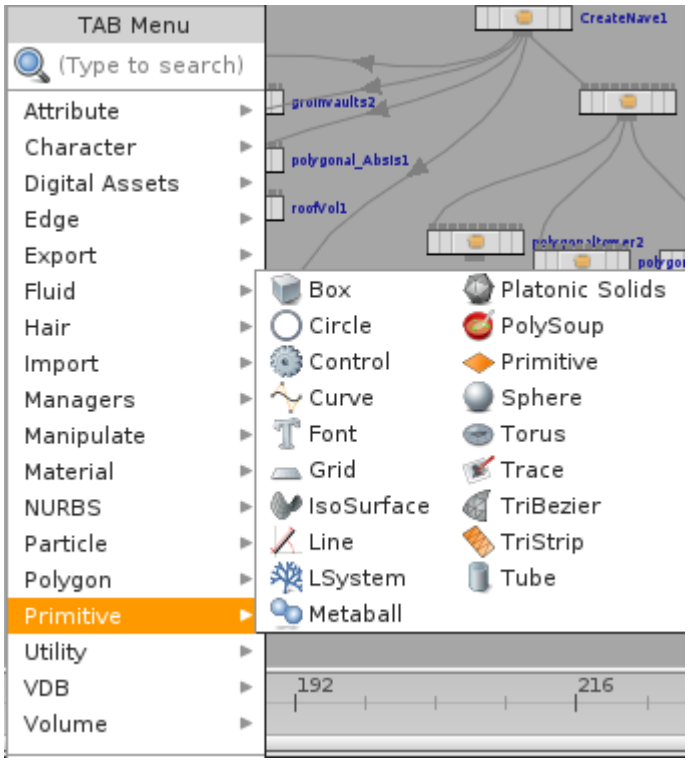


Figura 1.15: Tipus de nodes de la categoria Primitive

- **Utility**

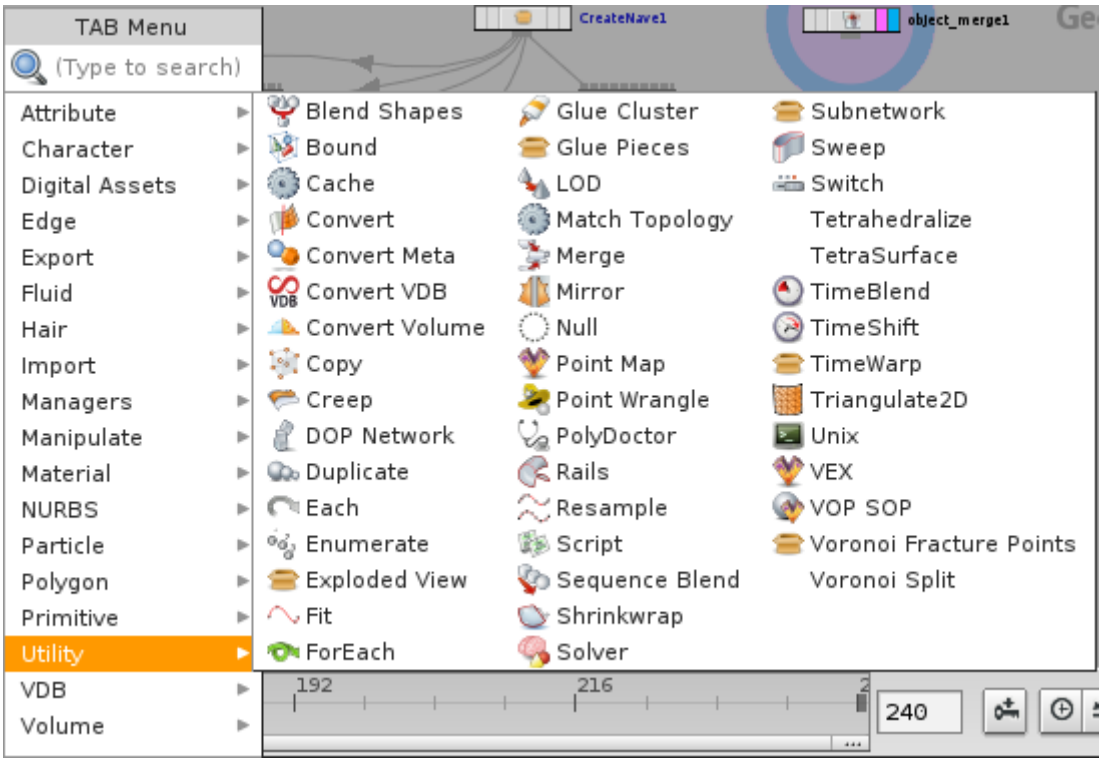


Figura 1.16: Tipus de nodes de la categoria Utility

- **VDB**

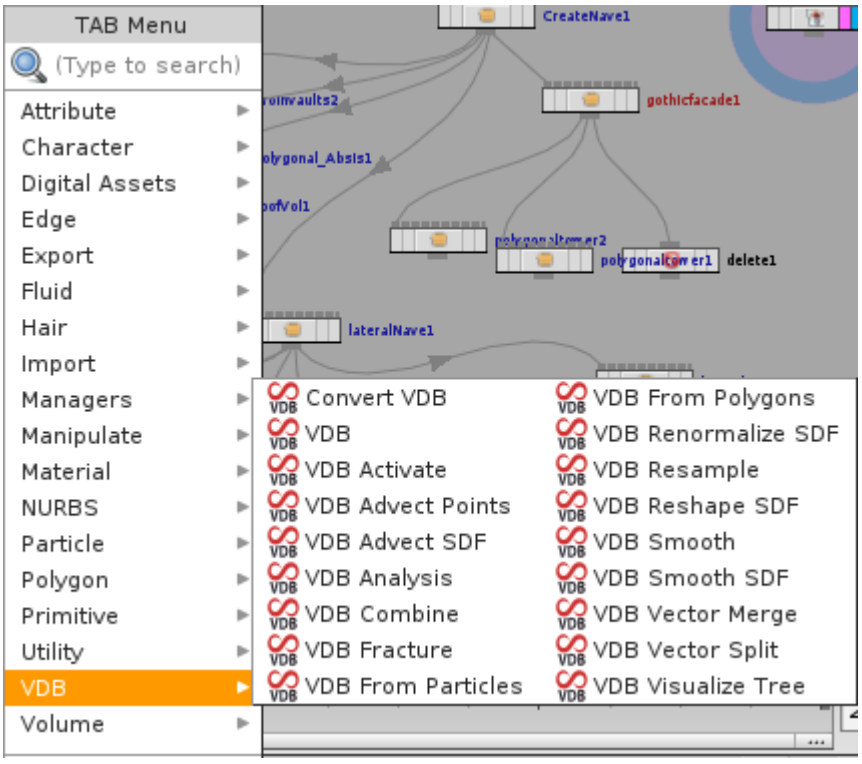


Figura 1.17: Tipus de nodes de la categoria VDB

- **Volume**

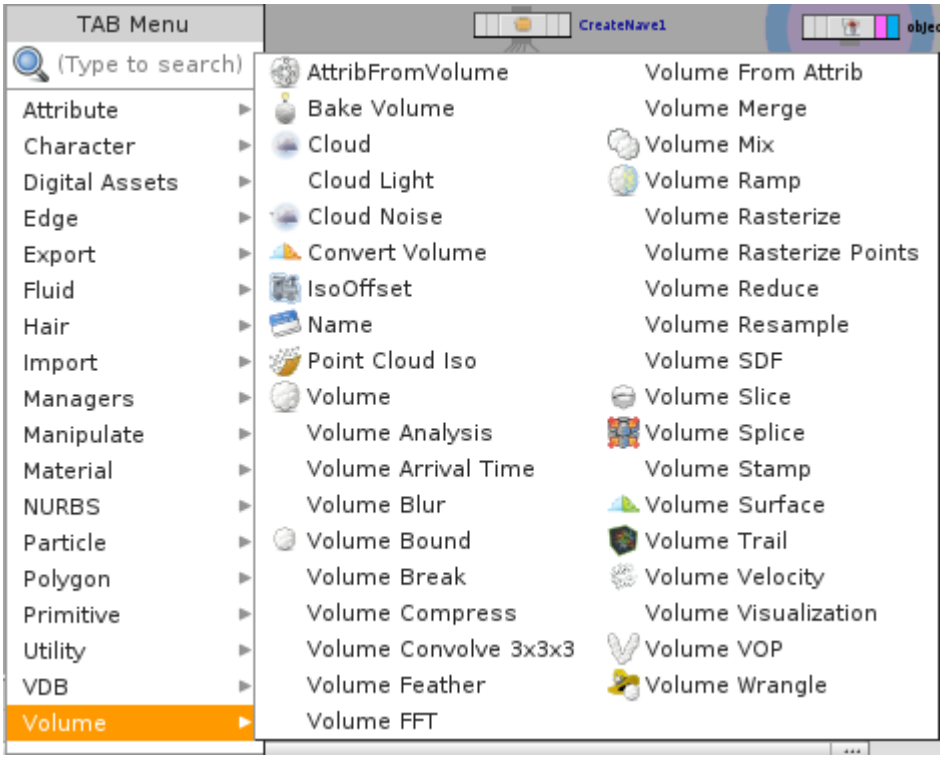
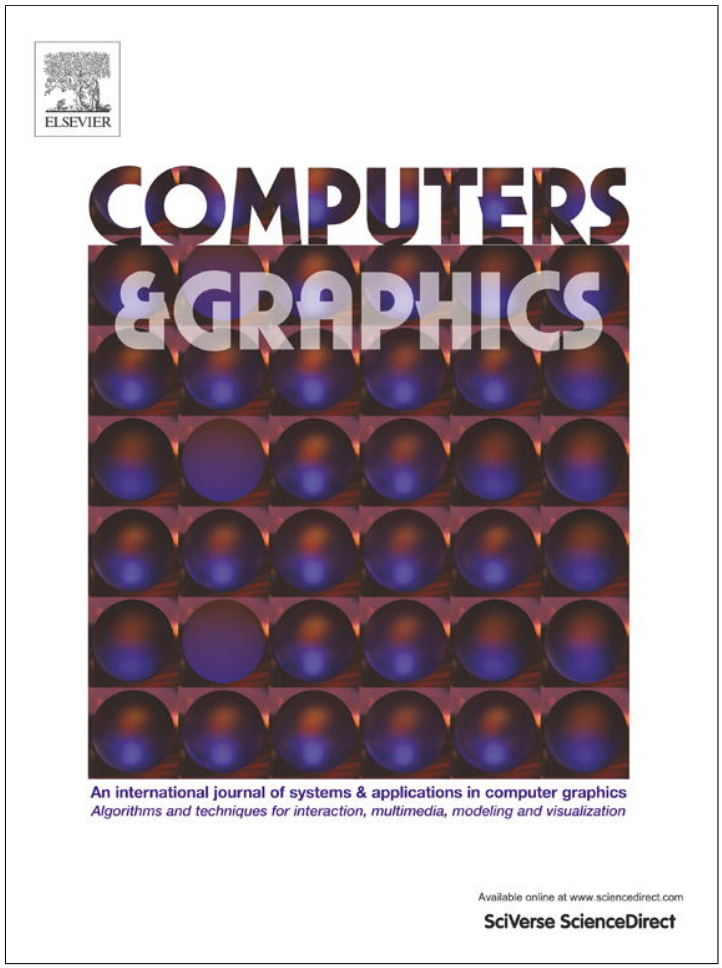


Figura 1.18: Tipus de nodes de la categoria Volume

Capítol 2

Article en el que es basa el document

A partir de la pàgina següent es troba l'article on s'explica a nivell teòric el procés de Copy&Paste procedural.



Computers & Graphics 37 (2013) 238–246

Contents lists available at SciVerse ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Special Section on Procedural Modeling

Visual copy & paste for procedurally modeled buildings by ruleset rewriting

Santiago Barroso, Gonzalo Besuievsky *, Gustavo Patow

Geometry and Graphics Group, Universitat de Girona, Spain

ARTICLE INFO

Article history:
Received 15 October 2012
Received in revised form
8 January 2013
Accepted 8 January 2013
Available online 26 January 2013

Keywords:
Procedural buildings
Interactive modeling
Graph rewriting

ABSTRACT

With the increase in popularity of procedural urban modeling for film, TV, and interactive entertainment, an urgent need for editing tools to support procedural content creation has become apparent. In this paper we present an end-to-end system for procedural copy & paste in a rule-based setting to address this need. As we show, no trivial extension exists to perform this action in a way such that the resulting ruleset is ready for production. For procedural copy & paste we need to handle the rulesets in both the source and target graphs to obtain a final consistent ruleset. As one of the main contributions of our system, we introduce a graph-rewriting procedure for seamlessly gluing both graphs and obtaining a consistent new procedural building ruleset. Hence, we focus on intuitive and minimal user interaction, and our editing operations perform interactively to provide immediate feedback.

1. Introduction

A broad range of areas, such as games, movies or urban simulation require virtual 3D city models with detailed geometry. Procedural modeling [1] has proven to be quite effective for this task [2], offering a potential alternative to the labor-intensive modeling tasks required by traditional 3D modeling techniques for building reconstruction. However, traditional procedural methods are not always a suitable alternative to manual modeling. With this, there is an increasing need for more advanced content creation and editing tools. However, it is not straightforward to extend existing tools, as for example sketch-based interfaces for modeling [3], drag-and-drop mesh tools [4] or the modeling-by-example approach [5] to procedural models because these tools operate on the mesh level, and are not able to preserve the procedural nature (i.e., the ruleset) of the input building.

In this paper we focus on a specific editing application: copy & paste for procedural building models. One of the main motivations for choosing this application is to provide a simple to use, intuitive editing metaphor that would allow non-experts to generate new content using pre-existing rulesets (see Fig. 1). We will present a complete, end-to-end system for procedural copy & paste consisting of the following components: selection, copying, composition (paste) and proportion adjustment. We will also describe our contributions for each component.

With our approach, artists can easily reuse already known building styles to create new content. As a benefit, the tools may shorten significantly the modeling time creation, avoiding designing new rules or reconfiguring old ones. Similar to [6], we used a visual programming paradigm, where the user can construct and modify the building by simply connecting its components on screen by interactive visual inspection. Actually, our approach is complementary to existing modeling techniques, as it produces a building ruleset ready to be used in any production environment.

Contributions To address these challenges, we have developed an end-to-end system for procedural copy & paste with the following additional contributions:

- We provide an interactive and intuitive visually-based method for editing models.
- We introduce a graph-rewriting procedure for seamlessly gluing source and target graphs.
- Our editing operations perform within interactive rates to provide immediate feedback.

2. Previous work

The idea of adding copy & paste operations to modeling tools is not new. The modeling-by-example approach, presented by Funkhouser et al. [5] allowed compositing of different parts from different input models to obtain a desired output. Later, Nealen et al. [3] presented sketch-based interfaces for preserving details

* Corresponding author. Tel.: +34 972418833; fax: +34 972418792.
E-mail address: gonzalo@ima.udg.edu (G. Besuievsky).

0097-8493/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cag.2013.01.003

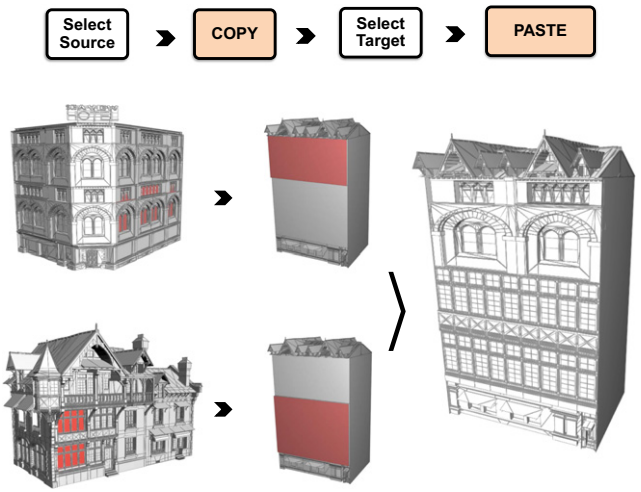


Fig. 1. Copy-Paste workflow system (top). From a procedural buildings selected sources (bottom, left) and a selected target (bottom, middle), a new building is composed (bottom, right).

while mesh editing. There are also drag-and-drop mesh tools, like Autodesk's Meshmixer [4] to manipulate geometry allowing copy & paste operations. However, none of these approaches is suitable to be applied to procedural models because they operate on the mesh level, and are not able to preserve the procedural nature (i.e., the ruleset) of the input building.

Procedural Modeling is a term that describes a family of techniques that generate geometry from a set of rules. These approaches [7,1] have emerged as an elegant solution for the generation of massive urban landscapes from a simple ruleset. Since the earliest works, great improvements have been done in terms of interactivity and ease of use, specially for urban layouts [8,9]. The interested reader is referred to the articles by Vanegas et al. [10] and Watson et al. [2] for a more in-depth review of the current state of the art in this topic.

Lipp et al. [6] presented one of the firsts attempts to improve editing operations for procedural buildings using an interactive visual system, which completely avoids text editing rules. Later, CityEngine [11], Epic's UDK [12] and Patow [13] independently developed visual representations for the rulesets, considerably easing the development process. More recently, some approaches tried to bring together both worlds: direct control and visual procedural languages, resulting in a simple visual traversal of the hierarchy tree plus direct visual assignment of the desired changes [14]. However, tool development has not gone much further in this direction, reducing the user options to only a few simple operations [13]. In this paper we provide a completely visual copy & paste application to simplify the user's editing tasks.

The idea of the copy & paste operation is intimately related with local control in procedural models. As mentioned, Lipp et al. [6] developed a local control mechanism, but it was external to the ruleset and required the re-evaluation of the local changes for every change in the building structure. Later, Patow [13] introduced an explicit *Exception* command, while Krecklau and Kobbelt [15] added specific rules selecting a given label with a given shape index to apply the different change. None of these approaches achieve the flexibility of the copy & paste editing operation proposed here. In the context of urban models, Lipp et al. [9] also presented a solution for interactive modeling of procedural city layouts that allows to manipulate them intuitively using drag-and-drop operations.

Our work is also based on graph-grammars and graph-rewriting techniques. In Computer Graphics this topic cannot considered to be new: L-systems [16] can be considered as a graph-rewriting system if we consider the processed string as a "language" that is executed to generate a tree, and the rules then are part of a graph-grammar that transforms one string into another. A graph-rewriting system is also presented in [17] for specific 3D furniture model fabrication, but with no user interaction. Even in the context of urban procedural modeling there have been some approaches that take advantage of the graph-like structures that arise in urban layouts [9] or the building rulesets [13]. Here we also perform graph-rewriting operations over the rulesets, but in contrast to [13], where only simple operations to fix some minor design issues in the procedural model where allowed, our editing tool goes far beyond current state of the art techniques, allowing complex editing operations to be performed in a way transparent to the user. There is a large bulk of literature on graph- rewriting techniques, so we refer the interested reader to the excellent work by Heckel [18] and the well-known handbook by Rozenberg [19].

3. Procedural modeling

The seminal works by Wonka et al. [7] and Müller et al. [1] introduced grammar-based procedural modeling for buildings. The main concept of this technique is a shape grammar, which is based on a ruleset: starting from an initial axiom primitive (e.g., a building outline), rules are iteratively applied, replacing shapes with other shapes. A rule has a labeled shape on the left hand side, called predecessor, and one or multiple shapes (also called primitives) and commands on the right hand side, called successor

$$predecessor \rightarrow CommandA, CommandB : labelB;$$
$$labelB \rightarrow CommandC : labelC;$$

The resulting geometry is formed by shapes that can be optionally assigned new labels with the purpose of being further processed. In our system, this geometry carries all labels that the shape or any ancestor has received during the production process. The main commands, the macros that create new shapes in the classic approach, are

- *Subdivision* that performs a subdivision of the current shape into multiple shapes,
- *Repeat* that performs a repeated subdivision of one shape multiple times,
- *Component split* that creates new components shapes (faces or edges) from initial volumes,
- *Insert* command that replaces a pre-made asset on a current predecessor.

Traditionally, during a rule application, a hierarchy of shapes is generated corresponding to a particular instance created by the grammar while inserting rule successor shapes as children of the rule predecessor shape. This production process is executed until only terminal shapes are left.

3.1. Graph-based Procedural modeling

The whole production process described above can be seen as a graph where each node represents an operation applied to its incoming geometry stream and the leaf nodes are the geometry assets [13] (see Fig. 2). A ruleset can be regarded as a *directed acyclic graph* $G=(N,E)$, where N is a set whose elements n_i are called vertices or nodes (i.e., these are commands of the ruleset), and E is a set of ordered pairs of vertices, called edges (i.e., the connections between the rules that represent the flux of geometry). Each command $n_i \in N$ processes its incoming flux of geometry, which is given by all its incoming primitives (i.e., the shapes)

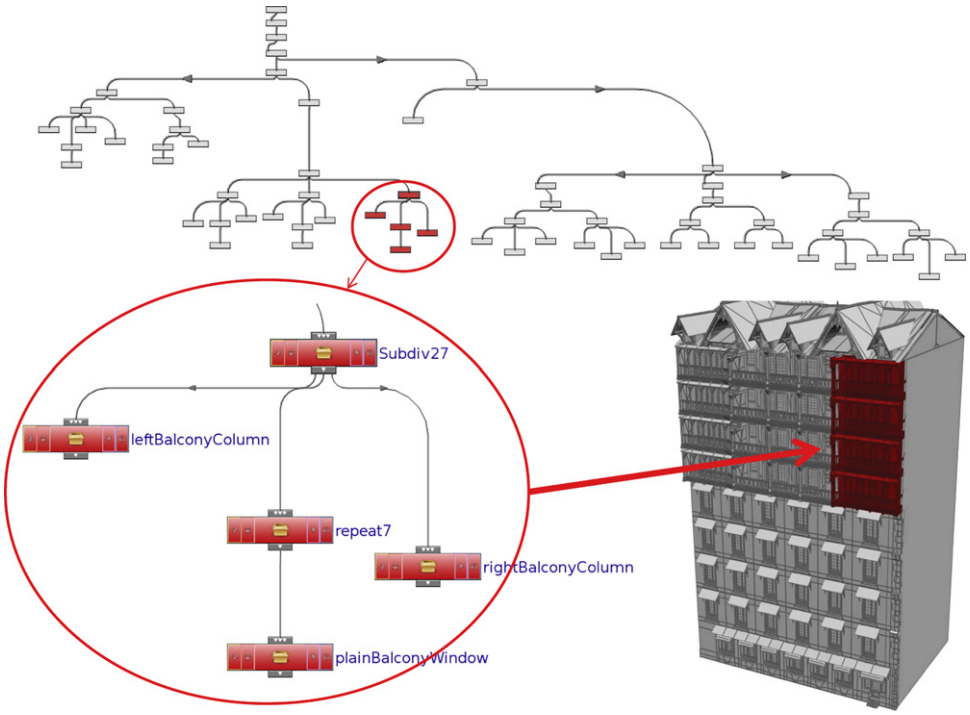


Fig. 2. A full graph-based representation (top) of a procedurally modeled building (bottom, right). The red subgraph (bottom, left) represents the red part of the facade. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

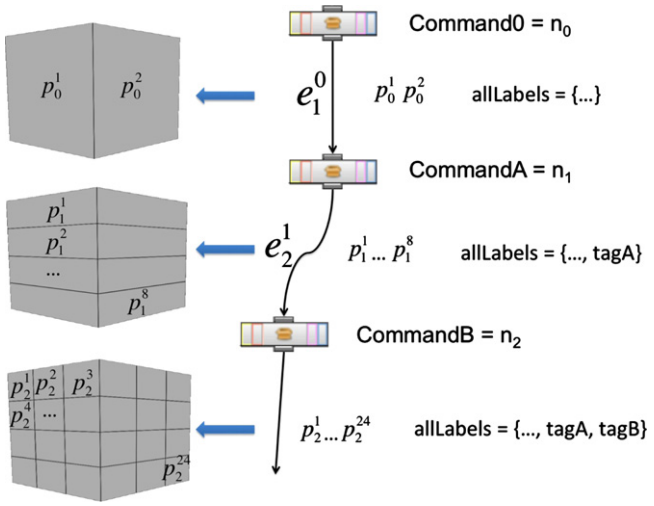


Fig. 3. Rules and products of the graph. Each node operates on the primitive inputs with correct label and outputs a set of tagged primitives.

that have the given predecessor label. Let $e_j^i \in E$ be the edge connecting the output of node n_i to the input of a *downstream* node n_j. Fig. 3 shows node n_1 wired to node n_2, and the wire that connects them is e_2^1. The left part of the figure shows the respective primitives that constitute the geometry flux in e_2^1.

Now, let P_i^{in} be the set of geometric primitives input by n_i through its input wires, and P_i^{out} the ones it produces after its processing. We denote the j-th primitive in the n_i output stream P_i^{out} by p_i^j. Given a node n_j, the set of primitives it receives is given by

$$P_j^{in} = \bigcup_{i:\exists e_j^i} P_i^{out}$$

Each incoming primitive has an associated label, which we denote by $label(p_i^j)$. A command selects the geometry to process by filtering the incoming primitives by their labels (the predecessors). The user must provide the filter for each node n_i as one of the node parameters, which can be recovered through the operation $filter(n_i)$. The commands operate primitive-wise: node n_i operates over each input primitive in P_i^{in} at a time, processing it only if the primitive has the required label, and discarding it otherwise. That is, an input primitive p will be processed if and only if $filter(n_i) = label(p)$. Thus, each node operates only on the primitives with the correct label, resulting in a set of zero, one or more output primitives. This renders the primitives into a tree-like relationship.

As every primitive p is the result of the operation of node n on an input primitive q, we can always track the relationship between p and n by using the operation $node(p) = n$. Also, for any output primitive p, we can track its parent primitive q with the operation $q = parent(p)$. Conversely, the set of children of any input primitive q is retrieved with the operation

$$children(q) = \{p : parent(p) = q\}$$

Any such primitive p is called *child* of q. Note we are assuming that each primitive q is processed by one and only one node n (resulting in zero, one or more primitives p), but it is easy to generalize the notation for the case where more than one node operate on the same primitive.

In our implementation, each primitive carries all labels that were assigned to any of its ancestors or itself. We implemented the operation $parent(p)$ for a primitive p from a node n_i to return the primitive q such that $q \in P_i^{in}$ and

$$\|allLabels(q)\| <= \|allLabels(p)\| + 1$$

where $allLabels(p)$ is an operation that returns the full set of labels attached to primitive p and $\|X\|$ gives the cardinality of a set X (see Fig. 3). The equality case comes from the fact that the user might decide not to add any new label to the generated primitives in some operations. The actual implementation of the operation to retrieve a primitive *most recent* label is

$$label(p) = allLabels(p) - allLabels(parent(p))$$

where the operator "−" is the difference of the label sets and *parent*(p) returns the ancestor primitive of p. Finally, we treat asset insertion separately from the primitive tree. The *Insert* nodes applied to any primitive p can be found with the operation *childInserts*(p).

4. Visual procedural copy & paste

Our visual copy & paste system for procedural buildings allows a user to select objects from several procedural source buildings and composite them into a desired procedural target building. The editing workflow for procedural copy & paste is shown in Fig. 1. Input to the system are procedural building rulesets for the source and target, which where previously created by an artist using tools like CityEngine [11] or Epic's UDK [12]. The system can be divided into four components:

- Selection
- Copying
- Composition (Paste)
- Proportion adjustment

Selection consists of the navigation through the building hierarchy created at rule evaluation, and selecting a location in the hierarchy. Then the users select one or more objects from the source buildings to be copied to any desired location selected in the target building.

The *Copying* operation can be decomposed into identifying the participating primitives, identifying their respective generating operations, and storing this information for later reuse.

Composition consists of consistently pasting the previously copied selection in the target building. It is performed interactively while the user is viewing the resulting composite. Finally, the *Proportion adjustment* component fixes the (possibly) wrong measures that could have been left in the pasted subgraph.

4.1. Selection

The first step is to provide a way to select part of a facade by a traversal of the primitive-tree. This clearly implies navigating through the primitive hierarchy, which means tracking it as described in Section 3.1. We provide two different starting points for this navigation: by selecting the roots of the hierarchy (the 2D primitives produced by the *Component split* operation), or by directly selecting of a leaf by a point-and-click interface. Then, a simple user interface lets the user visit any primitive children, browse through its siblings, and go back to the parent primitive as desired [14].

Once the user selected a primitive p, either as a result of this navigation through the primitive tree or by direct point and click, we can precisely locate it both at the tree level and at the graph level. This is done through the operations *children*(p), *parent*(p) and *node*(p) described in Section 3.1. Fig. 4 shows examples of navigation and selection of different parts of a building.

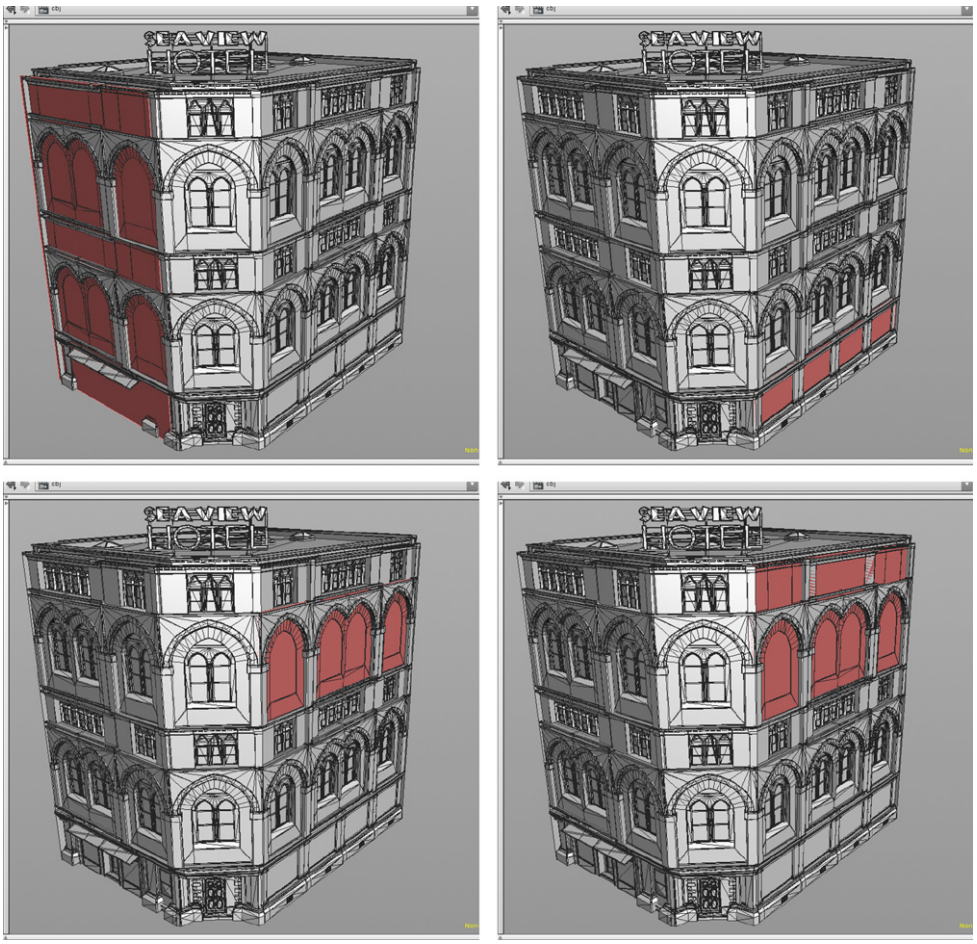


Fig. 4. The selection system with four examples. The visual interface allows to navigate through the primitives of the tree for interactive selection.

4.2. Copy

The next step, once the user has selected a given primitive p, is to collect all nodes in the source graph G^S that operate on p or any of its descendants (see Fig. 5). For that, we iteratively traverse the primitive tree in depth-first order, starting from p, until we reach the leaf primitives. For every descendant d of p, we collect the node $node(d)$ that produced it. This new set of nodes is $N' \subset N^S$, and the node inter-relations are the edges $E' \subset E^S$, forming a subgraph $G' = (N', E')$. We record the labels associated to p with $l = allLabels(p)$. We also keep track of the *Insert* commands associated with each primitive.

Once all primitives have been traversed, and the subset N' has been built, we copy all the nodes belonging to N' and all the edges that connect such nodes into a buffer, which will be later used during composition. Observe that we also select the node n_p that originates p, which should *not* be part of N'. However, keeping this node will be useful later on.

4.3. Composition

Once we have copied the subgraph N', we must select the target area in the composite building. This is done as before, by navigating the primitive hierarchy and selecting a primitive t. Once t is selected, we need to merge the graph $G^T = (N^T, E^T)$ of the target building with the subgraph G' we copied from the source building. This is done following the rules of graph rewriting [19,18]: first, we identify the actions performed on t by G^T (Fig. 6(a)), to then separate t from G^T, as it is going to be processed by G' (Fig. 6(b)). This is done by modifying the ruleset so t is excluded from its regular evaluation. Then, we need to make sure that *only* t is directed towards G' (Fig. 6(c)), and finally, G' must be connected to G^T (Fig. 6(d)). However, the sizes and distances in G' can be very different than those in G^T, so a final adjustment step is needed.

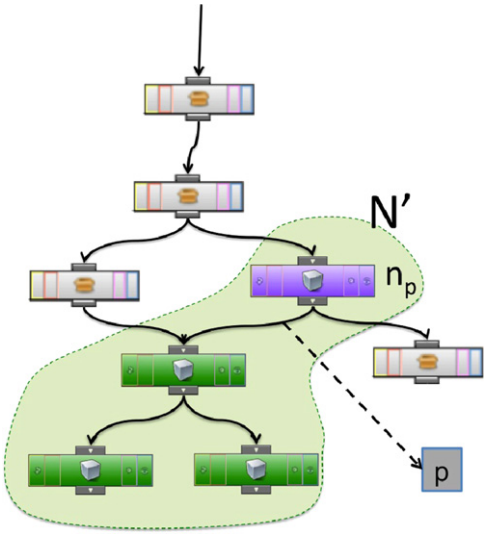


Fig. 5. Copy process. The processes node p (lilac node) and all its descendants (green nodes) are copied into a buffer. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

A *graph transformation* is defined by a tuple $(c,L,R,glue)$ consisting of a condition c that, if positively evaluated, will activate the rewriting transformation from the left term L into the right term R by means of the gluing mechanism *glue*. In our case, we do not need to evaluate c as the user directly gives us the information where to glue into the target graph by specifying primitive t. Here, $R \equiv G'$ and $L \equiv \{n_t = node(t)\} \cup \{n_i : \exists\, e_i^t \in E^T\}$, i.e., the node that generated t plus all nodes connected downstream to it. In Fig. 6(a) these nodes are shown in blue color. Now, it only remains to define *glue* to transfer the selection to the target building facade.

First, we need to exclude t from the regular processing in the target building. This can be done with an *Exception* node [13], which assigns a user-defined label to a primitive that fulfills a given condition; and a *selector* node, which deletes any primitive that does not have the adequate label. We connect $node(t)$ to the exception node, and then the selector to the exception output. Finally, we reconnect all nodes connected to the outputs of $node(t)$ to be the outputs of the selector node. The parameters for the *exception* node are set to select t. This way, all other primitives produced by $node(t)$ will continue being processed as before, with t set aside of the process. In Fig. 6(b) these new nodes are shown in yellow.

The next step is to redirect t to the new subgraph we are going to attach. Again, we do this with another *selector* node that is set to keep t and delete all other primitives. Now, we have to copy the graph G' into the new graph, attaching the nodes in N' to the outputs of the new *selector* node. For that, we instantiate G', including all nodes in N' and their connections. Remember that we have kept n_p, the node in G^S that produced the chosen primitive p, so it will be pasted along with the other nodes in G' (see Fig. 6(c)), where n_p is represented as a lilac node and the other nodes in G' are shown in green.

The final step in the gluing process is to transfer all the *useful* connections from n_p to the last *selector* node created. We say that a connection e_i^p from n_p to n_i is *useful* if $filter(n_i) \in l$ (remember that $l = allLabels(p)$). Finally, we simply delete n_p (see Fig. 6(d)). However, the process still requires a last stage: as mentioned, the whole procedural paradigm is based on the rules processing the primitives with the right labels. As we have pasted parts of a completely different ruleset, there might happen that an interior node in the pasted subgraph G' requires a label that was actually set before, in the part of the source graph G^S that does not contain the subgraph G', but whose nodes produce geometry that is latter processed in G'. This would result in a wrong processing when G' is pasted in the new graph. To prevent this, we use the collected

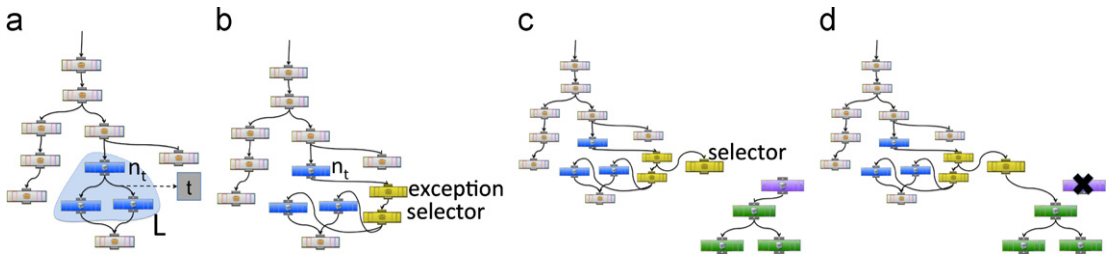


Fig. 6. Composition process. L nodes in blue, G' nodes in green, n_p in lilac, *glue* nodes in yellow. From left to right: (a) selection of the target primitive t and its corresponding nodes in G^T, (b) the isolation of the selected primitive from the regular processing, (c) the subgraph G' is pasted, and (d) the final merging of the copied subgraph G' into the target graph G^T. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

labels l and apply them to t. To avoid assigning unnecessary labels, before assigning them we check whether they are effectively used in G'. To add a label to t, we simply add a rule

$labelT \rightarrow nop : newLabel;$

where *nop* means *no-operation*.

4.4. Proportion adjustment

Although the above steps have glued the selected ruleset G' into the target graph G^T, special care must be taken to the parameters used in the rules being copied to the new building. As already explained, the method is intended for an automatic replacement of the building part, so, if the sizes of the source and target selected primitives are different, a readjustment must be performed.

If primitive p has a size of $W^p \times H^p$ and primitive t has a size of $W^t \times H^t$, then we change each parameter of the nodes in G' by the respective proportion. If a parameter is defined in a node that operates Y direction (e.g., a *subdiv*{Y} or a *repeat*{Y}) with a value v^p, then we modify it to be $v^p \cdot W^t/W^p$. Conversely, for parameters in nodes operating in the X direction, the new value would be $v^p \cdot H^t/H^p$.

Once this is done, the primitive t, and thus the corresponding part of the new facade, will have the appearance of the original building facade.

5. Results and discussion

Our system is implemented on top of SideFX's Houdini [20]. The implementation is done using embedded Python scripts and external Python methods.

Our visual interface allows to perform all operations iteratively, allowing the user to navigate through the hierarchy and select any primitive at any level (see Fig. 4). Please refer to the accompanying supplemental video showing the interactivity of all the results presented in this section.

Fig. 7 shows a new building with a facade created using three parts from already existing procedural buildings. A typical modeling session using our system (see Fig. 1) has the following sequential steps for each part of the building to be copied: navigate through the source building using the selection tool (see Fig. 4), use the copy button to store in a buffer the interesting building part, navigate through a target building to select the part to paste the copied structure to, and finally apply the paste operation to merge both structures. All these operations took only a couple of minutes for generating the whole model in Fig. 7.

The main contribution in usability of our approach is that it is an end-to-end process that runs without the need of writing rules or even neither changing parameters: everything is done visually.

Our system also allows modifying existing building rulesets (see Fig. 8). In this case we follow the same steps as in the previous example, but the paste operation has to modify the parts affected by the changes. This implies to extract the selected geometry from its previous regular processing, and its processing with the new ruleset. This is done automatically, without the need of user intervention to reconnect or modify the nodes.

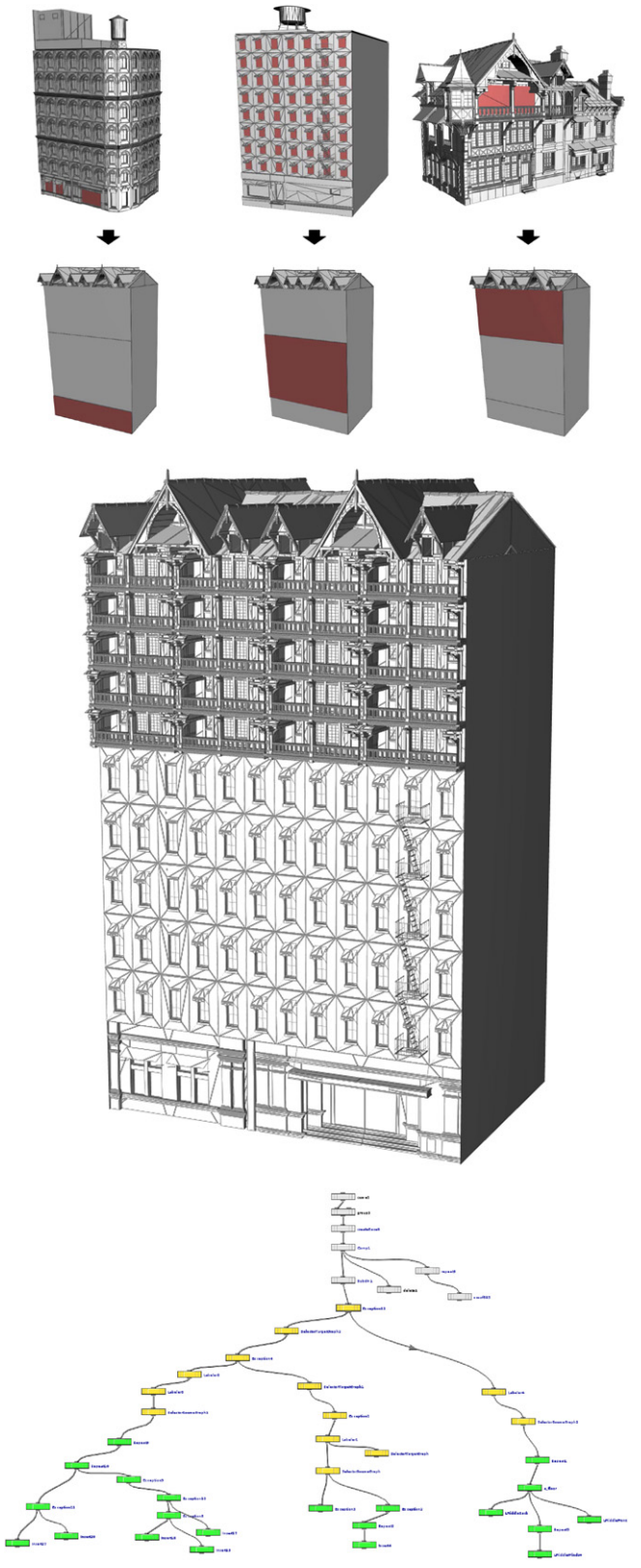


Fig. 7. From three selected parts of existing buildings (top), a new building is created (middle). In the resulting graph (bottom), the green nodes represent the subgraph G', whereas the *glue* ones are shown in yellow. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

5.1. Discussion

Comparing our system to existing solutions like the one proposed by Esri's CityEngine [11] or Epic's UDK [12], we observe that in their system the only way to copy & paste the ruleset operating on a building part is by manually selecting the needed rules, pasting them in the target ruleset, and then manually merging the rules to obtain the final ruleset.

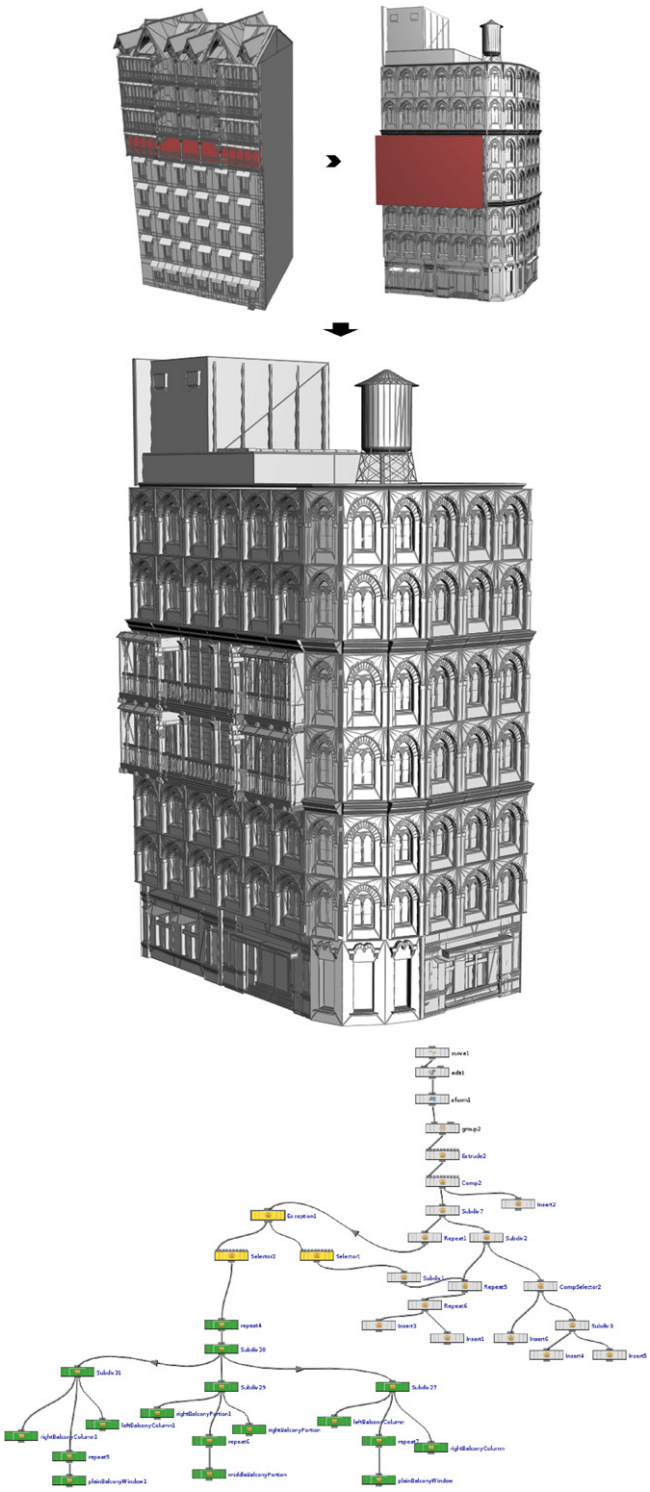


Fig. 8. From an existing building (top, right), a new building is created (middle) using a peace of facade as source of modification(top, left). The resulting graph (bottom) is shown with the same color representation of the previous example. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

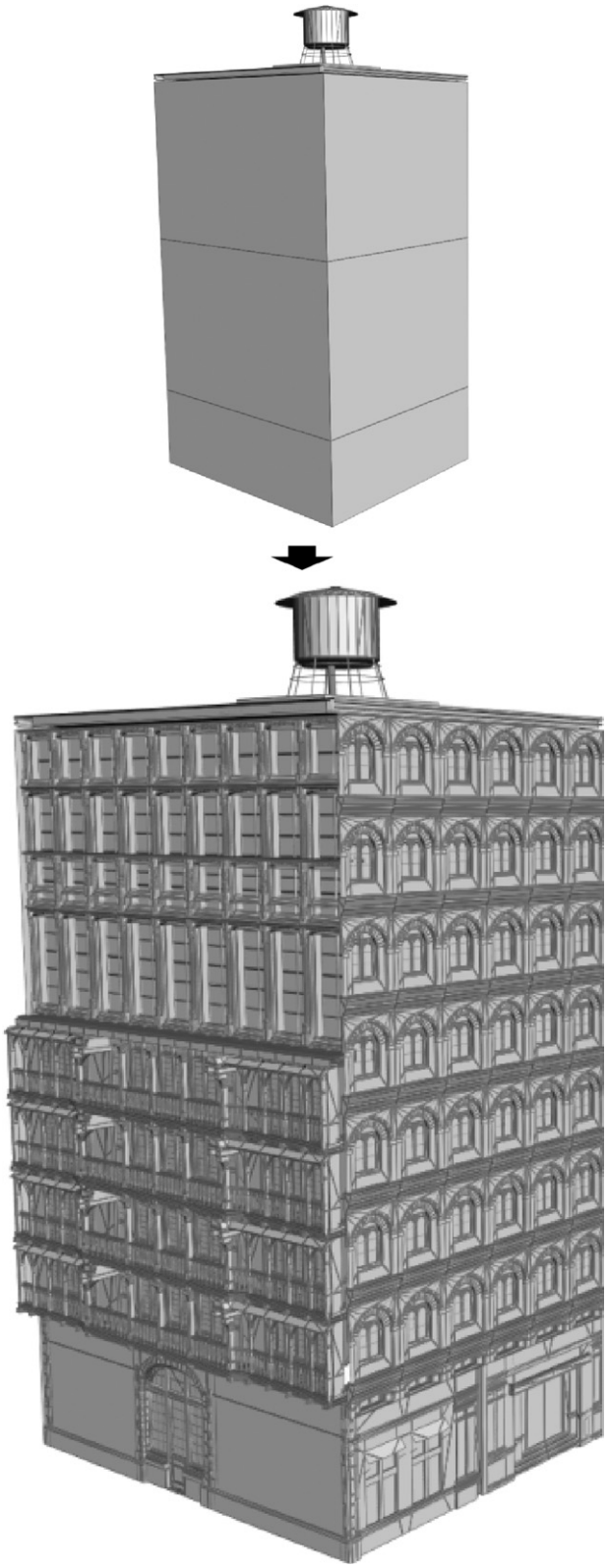


Fig. 9. Final building (bottom) used for counting the number of internal operations performed in Table 1. The model is generated from a naked building (left) using different parts of previous models.

As we stated before, our technique shortens modeling time. In order to evaluate the improvement with respect to a manual approach, we measured the number of operations we perform, which is equivalent to the number of manual operations required. For this test we used the model shown in Fig. 9, built from a naked building using selected parts from the four input models shown in Figs. 1,7 and 8. Table 1 shows the number of operations for creating and deleting nodes, connecting and disconnecting nodes, and changing parameters. A total of 226 operations were

Table 1
Number of paste operations for generating the building of Fig. 9. We measure the following number of operations: creating/deleting nodes, connecting/disconnecting nodes and changing parameters.

Operations	Node+	Node−	Connect+	Connect−	Ch.Param	Total
Top left	4	1	8	1	17	31
Middle left	4	1	8	1	27	41
Bottom left	4	1	8	1	20	34
Right facade	18	1	35	1	65	120

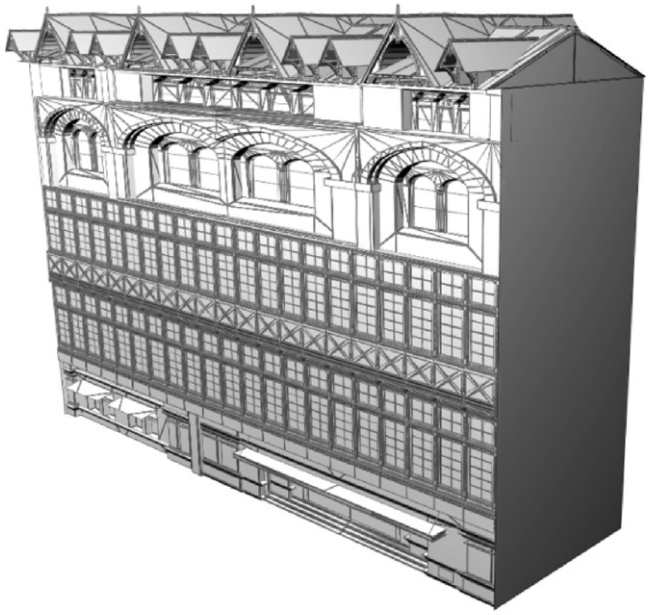


Fig. 10. Resulting model for applying an unwanted operation, the top inserted windows does not fit well with the rest of the building. The user can both manually readjust them changing rules or undo the operation.

needed for the whole copy & paste process (copying implies no changes in the source graph). A simple analysis leads to the conclusion that the computational time required by our algorithm is approximately linear in the number of pasted nodes. From the user perspective, we could compare this figures with the number of mouse clicks required in our system for the same operation. Of course, this will depend mainly on the number of model inputs. For this test model, a total of 16 mouse clicks were required, four for each input part. We can conclude that, for this test, we reduced work by a factor of 14 times, computed as the ratio of manual changes in the ruleset compared to the effort required with our technique (*num.op./num.clicks*). However, this comparison is somewhat incorrect in the sense that it is difficult to measure the time needed to *understand* a given ruleset in order to be able to select nodes, and then to *understand* the target ruleset to be able to marge the result. In our system, no understanding of the ruleset is required, just a simple visual tree traversal.

Although we designed our system with the idea of freeing the user from manual intervention at the ruleset level during the copy & paste operations, in some cases the user might want to manually adjust the resulting ruleset. In general, in our experiments we found that there was no need to further edit the resulting ruleset, as the implemented graph-rewriting steps already glued correctly the two graphs. However, the automatic proportion adjustment for the parameters described in the previous section may need some further parameter tweaking, which is to be expected: it can happen that the measures and distances designed for a building to look right might not fit well to a target building and could require some adjustment. For instance, an asset could require an offset to be positioned correctly in a given building, but the same offset, even if corrected as described in our system, might lead to a separation between parts of the building coming from different rulesets, resulting in a lower geometric quality as the model would not be watertight. In any case, these small adjustments are quite simple to perform. All the examples in this paper and the companion supplemental video were rendered *without* any manual user intervention.

Another related aspect that we would like to emphasize is that the copy & paste system does not free the user from any responsibility during the editing process. For instance, if the user copies a floor and pastes it into a whole facade, the result would be a very stretched floor, which probably is not what the user had in mind, but is the logical consequence of the choices made. As an example, Fig. 10 shows a resulting model where the top windows were copied from a *Subdiv* rule and pasted into a large part of the facade. For these cases two kinds of strategies can be used. One of them is, as explained before, to require manual intervention to change the rules and parameters (add a *Repeat* rule in this example). The other workaround that we came up with would be to implement both a generalized *undo* operation, plus adding the possibility for the system to add, upon user-confirmation, *repeat* nodes in either the horizontal direction, the vertical direction, or both. We leave this for further research, as guessing user intentions is beyond the scope of this paper.

6. Conclusions and future work

We have explored new alternatives for visual editing procedural buildings. In particular, we have developed a copy & paste mechanism that would allow non-technical users to reuse whole rulesets from existing ones, without the burden of any manual intervention. Up to now, users cannot seamlessly and transparently perform these actions. However, our copy & paste application can be smoothly integrated in any of these systems: Our current implementation is a few hundred lines long and can be immediately incorporated into any rule-based system like the commercial ones mentioned before. Our system only requires a small and intuitive input from the user: just to select the area to copy, the corresponding target area, and the system automatically produces a new ruleset that includes the selected modifications. We believe that this greatly enhances the artist toolbox, who is now able to obtain elaborated procedural models in only few minutes and without requiring to program rules. All complex manipulations are automatic and transparent to the end user.

As mentioned above, trying to infer the user's intention for any copy & paste operation is not a trivial task, but is worthwhile exploring, as it would increase the friendliness and usability of the whole system. Also, after the initial step done is here, it is possible to devise a ruleset-merging mechanism for buildings much in the same line as Lipp et al. [9] did for whole cities.

Acknowledgments

This work was partially funded by the TIN2010-20590-C02-02 project from Ministerio de Ciencia e Innovación, Spain. Our buildings are based on the Raccolet house and Urban Sprawl models from Daz3D (http://www.daz3d.com/).

Appendix A. Supplementary material

Supplementary data associated with this article can be found in the online version of http://dx.doi.org/10.1016/j.cag.2013.01.003.

Capítol 3

Manual d'usuari

3.1 Instalació de l'interfície d'usuari

Per dur a terme la instal·lació del Shelf s'ha de sobreescriure el fitxer que es troba en la carpeta d'instal·lació de Houdini, que en el cas de Windows es troba en:

"C:/Users/NomUsuari/Documents/houdini12.5/toolbar/default.shelf"

L'arxiu "default.shelf" s'ha de substituir, conservant el mateix nom, per l'arxiu "copypaste.shelf" adjunt amb el codi d'aquest projecte. D'aquesta manera quan l'usuari obri Houdini, l'hi apareixerà el Shelf que es mostra en la Figura 3.1 amb la pestanya "Interacció" i tota la botonera del projecte.

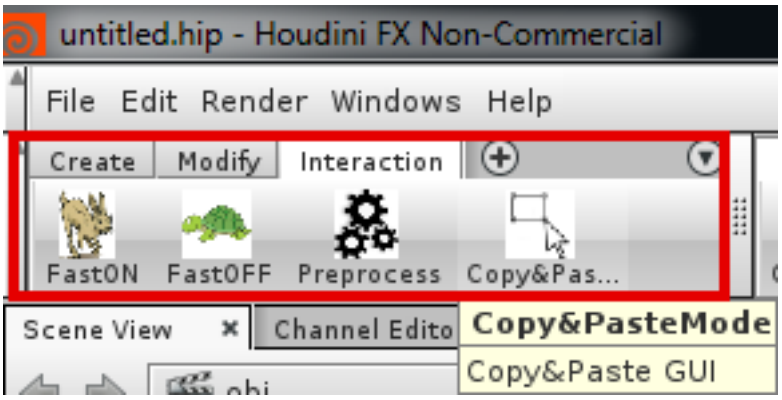


Figura 3.1: Exemple de la interfície d'usuari integrada a Houdini principal.

Un cop carregat el shelf ja es té tot el necessari per a poder començar a fer servir les funcionalitats del projecte.

3.2 Funcionament del mòdul d'interacció ràpida d'usuari

Com els requisits per a fer funcionar el mòdul d'interacció ràpida d'usuari és tenir instal·lat el Shelf del projecte i estar treballant sobre un edifici construït mitjançant **building**Engine, a les hores es suposa que l'edifici porta instal·lats també els Digital Assets de "**building**Engine.otl" i el "attribs.otl".

L'únic que fa falta és utilitzar els tres botons facilitats pel Shelf per a dur a terme el preprocessament de l'edifici, activació i desactivació del mode ràpid d'interacció d'usuari. Es poden observar els passos en la Figura 3.2.

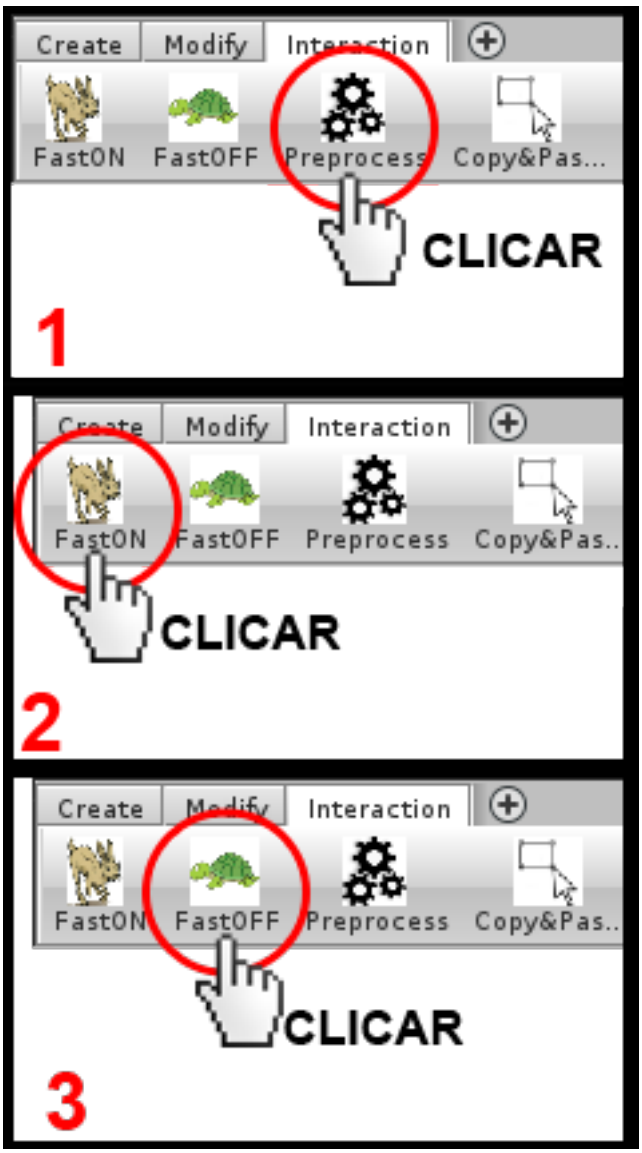


Figura 3.2: Exemple d'ús del mòdul d'acceleració d'interacció d'usuari.

- **Pas 1:** Prémer el botó "Preprocess " per a realitzar el precalcular l'edifici.
- **Pas 2:** Prémer el botó de "FastON" per a activar el mode accelerat d'interacció d'usuari.
- **Pas 3:** Prémer el botó de "FastOFF" per desactivar el mode accelerat d'interacció d'usuari.

3.3 Funcionament del Copy&Paste

Per a explicar el funcionament del procés de Copy&Paste, es farà un exemple pas a pas en el que a més de copiar i enganxar també es desa a fitxer. Els passos on es crea el fitxer o es carrega, són opcionals, però s'ha fet totes les possibles operacions per a que amb un sol tutorial quedin cobertes totes les branques.

Primer de tot s'ha de prémer el botó de Copy&Paste de la botonera localitzada en el Shelf tal i com es veu a la Figura 3.3.

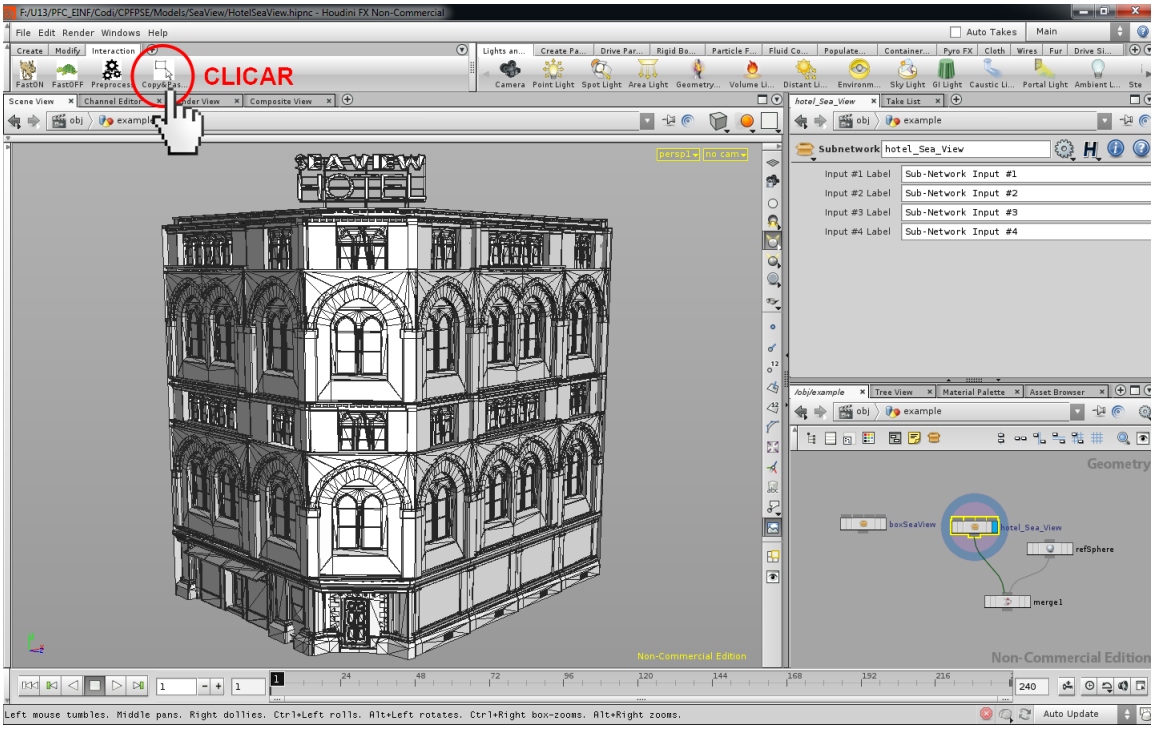


Figura 3.3: Exemple del Pas 1 en el funcionament del Copy&Paste.

Un cop premut, apareix el node "GUI" amb la botonera per copiar i enganxar. Seguidament s'ha de prémer el botó de "Select Geometry" tal i com es veu a la Figura 3.4.

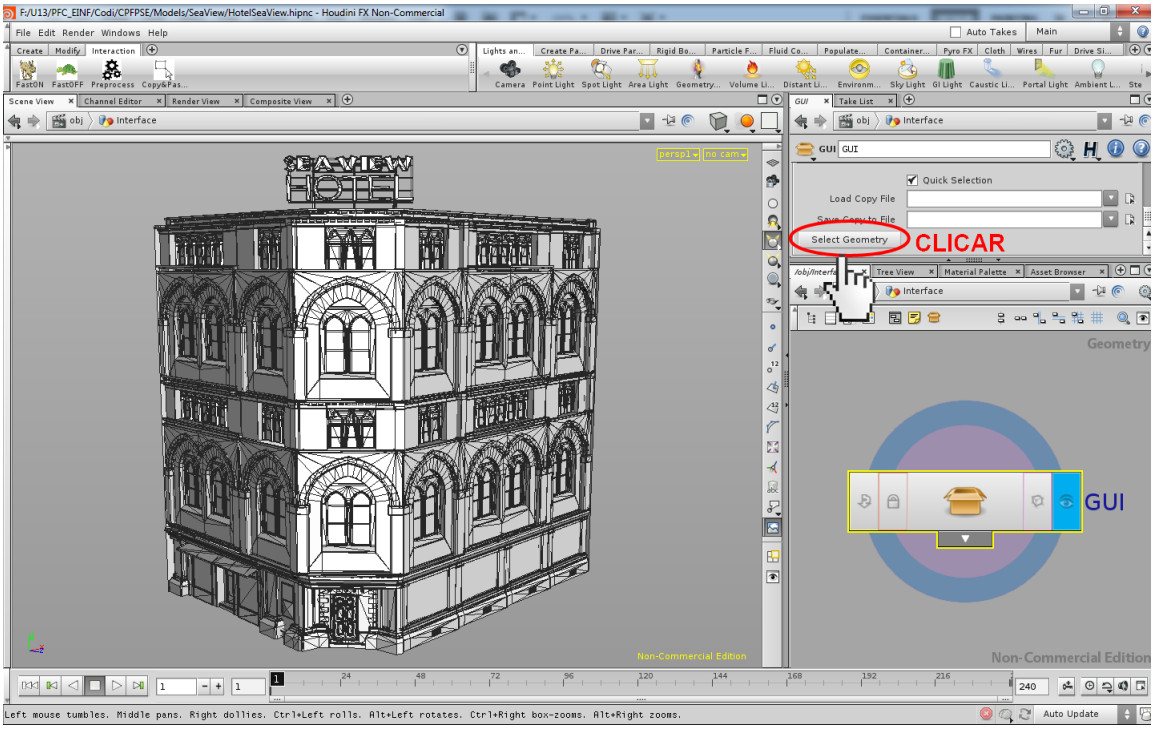


Figura 3.4: Exemple del Pas 2 en el funcionament del Copy&Paste.

Quan s'ha premut el botó, Houdini es posa en mode de selecció de geometria i permet a l'usuari efectuar seleccions, tal i com es pot observar en la Figura 3.5.

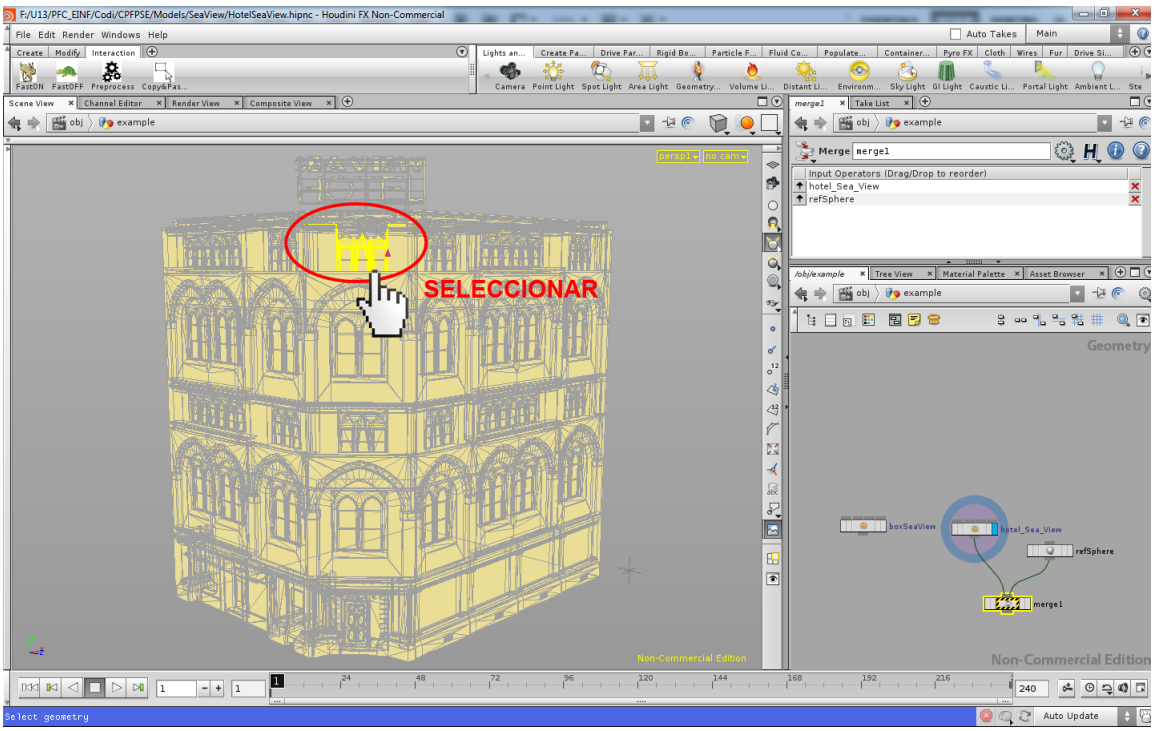


Figura 3.5: Exemple del Pas 3 en el funcionament del Copy&Paste.

Un cop l'usuari ha efectuat la selecció pertinent, ha de prémer la tecla **"ENTER"**. És important que l'usuari mantingui en el moment de prémer la tecla **"ENTER"**, el cursor dins de l'àrea del viewport de Houdini (on es veu l'edifici), ja que d'altra manera no s'efectua la selecció.

Un cop seleccionat, s'ha de triar la ruta on es desarà el fitxer XML amb el contingut de la selecció. Per fer-ho s'ha de fer els tres passos que es poden veure en la Figura 3.6.

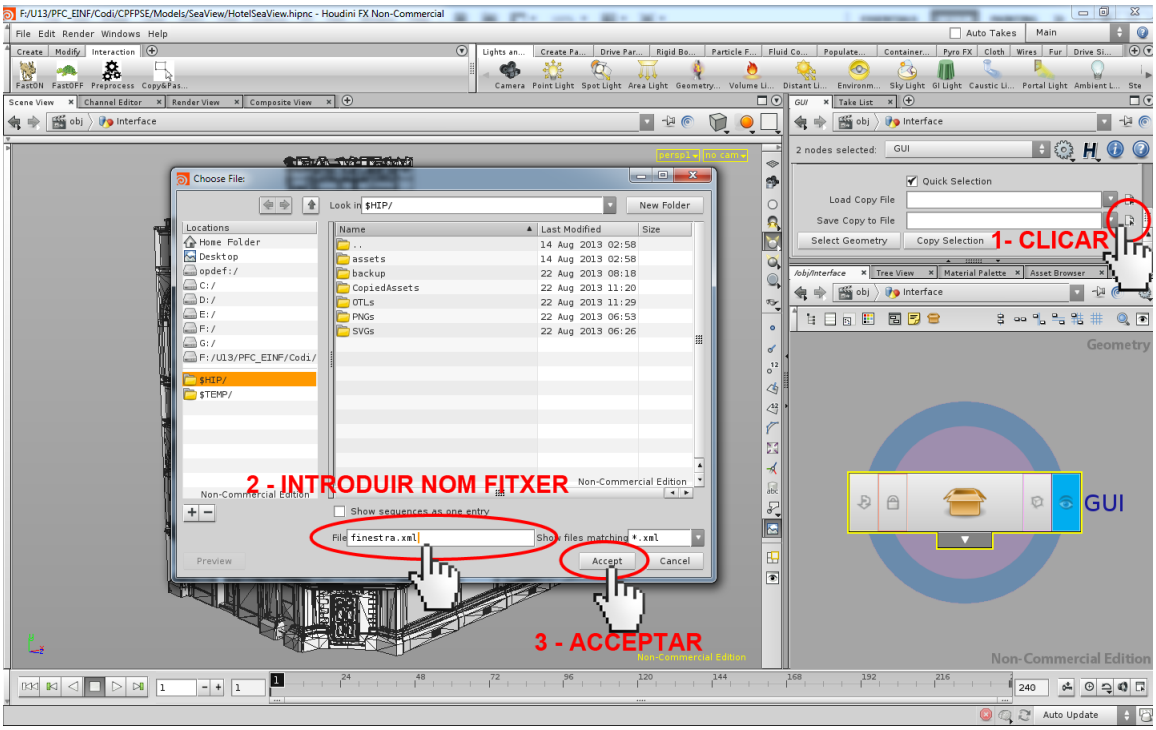


Figura 3.6: Exemple del Pas 4 en el funcionament del Copy&Paste.

Un cop definida la ruta del fitxer s'ha de clicar al botó "Copy Selection" per tal d'efectuar la còpia a buffer i a fitxer. Això es pot veure en la Figura 3.7.

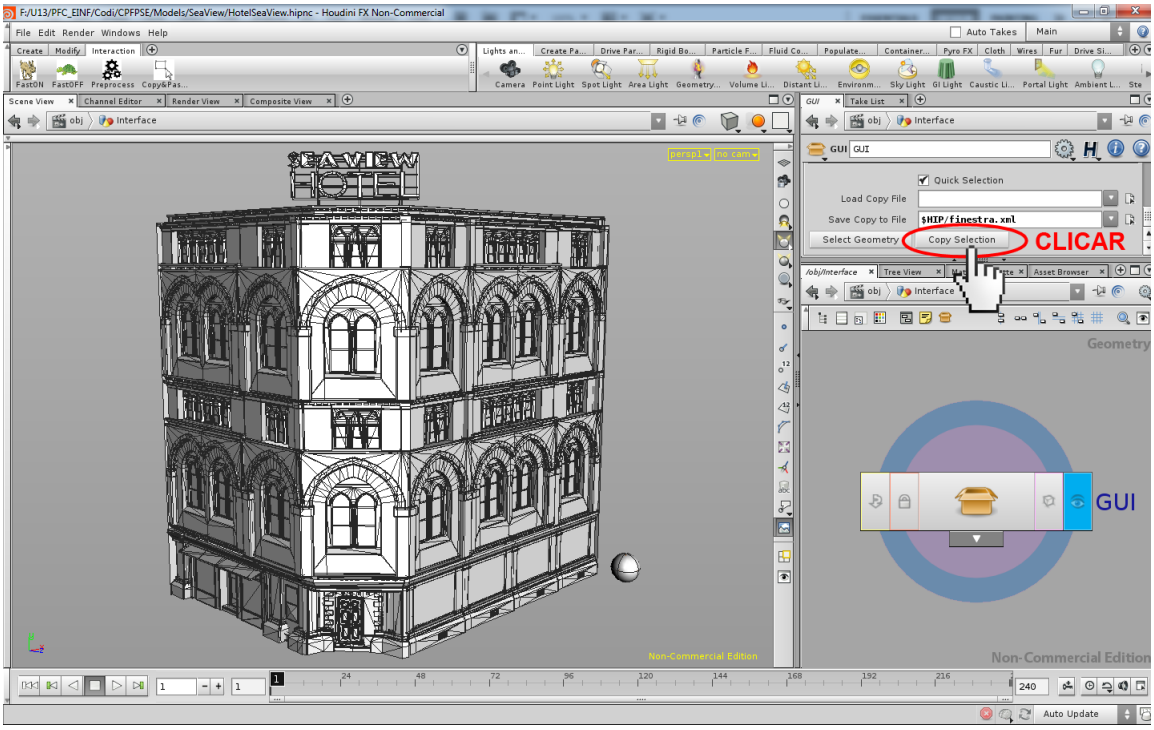


Figura 3.7: Exemple del Pas 5 en el funcionament del Copy&Paste.

A continuació obrim amb Houdini un nou edifici allà on es vol enganxar la selecció realitzada, en aquest cas s'obre l'edifici "CornerBuilding". Un cop obert, s'ha de clicar en la botonera del Shelf sobre el botó de "Copy&Paste" com mostra la Figura 3.8.

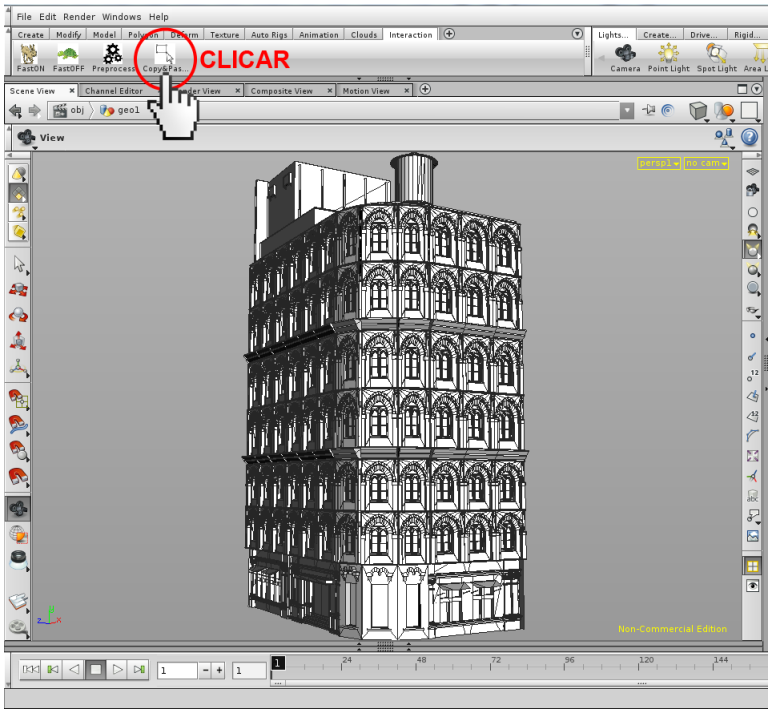


Figura 3.8: Exemple del Pas 6 en el funcionament del Copy&Paste.

Un cop activat el node de "GUI" es pot anar a escollir el fitxer que l'usuari ha desat

anteriorment, efectuant els passos que es poden observar en la Figura 3.9.

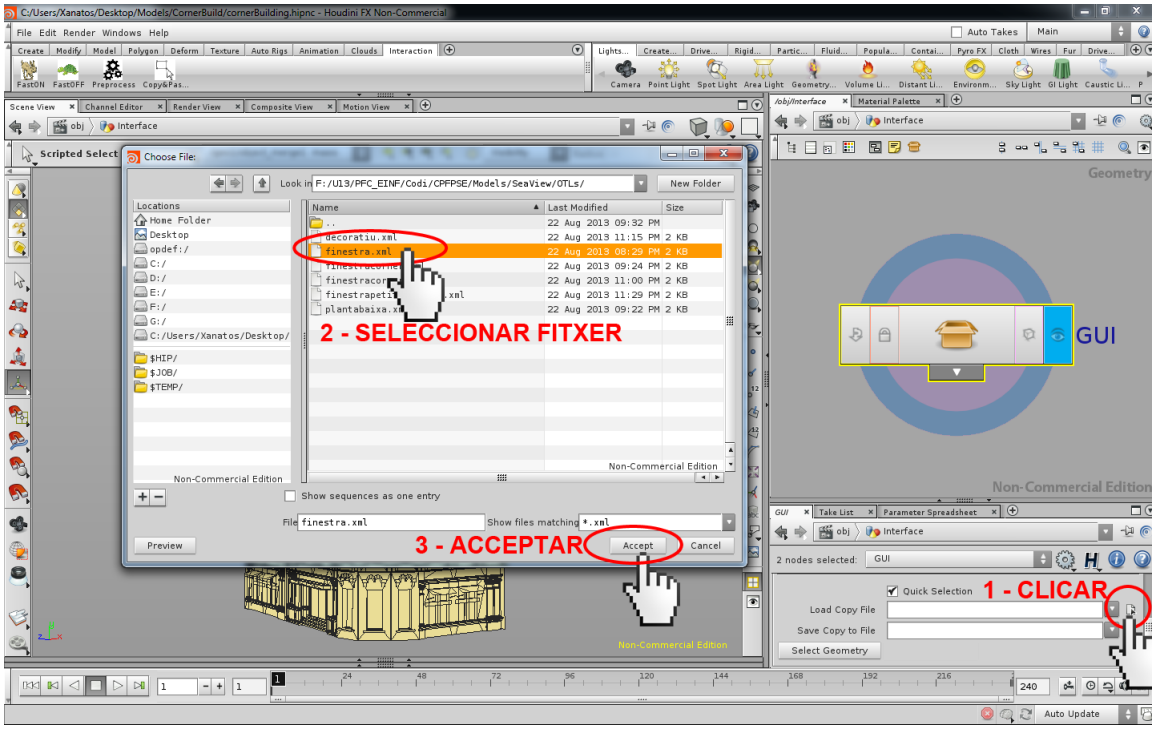


Figura 3.9: Exemple del Pas 7 en el funcionament del Copy&Paste.

Un cop carregat el fixer, s'ha de seleccionar allà on es vol efectuar l'enganxat. Per fer-ho primer s'ha de clicar en el botó "Select Geometry" com s'ha fet anteriorment. Es pot observar en la Figura 3.10 com es du a terme aquest pas.

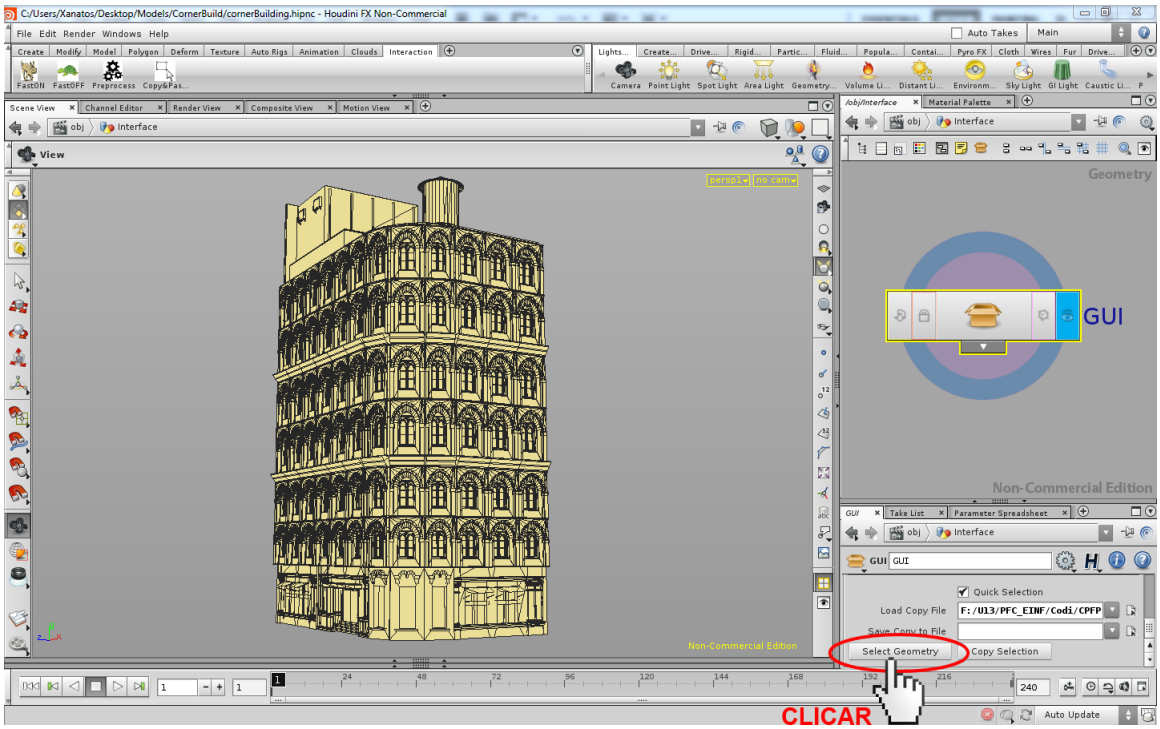


Figura 3.10: Exemple del Pas 8 en el funcionament del Copy&Paste.

Un cop Houdini s'ha posat en mode de selecció, l'usuari ha d'efectuar aquesta amb el ratolí tal i com es mostra en la Figura 3.11.

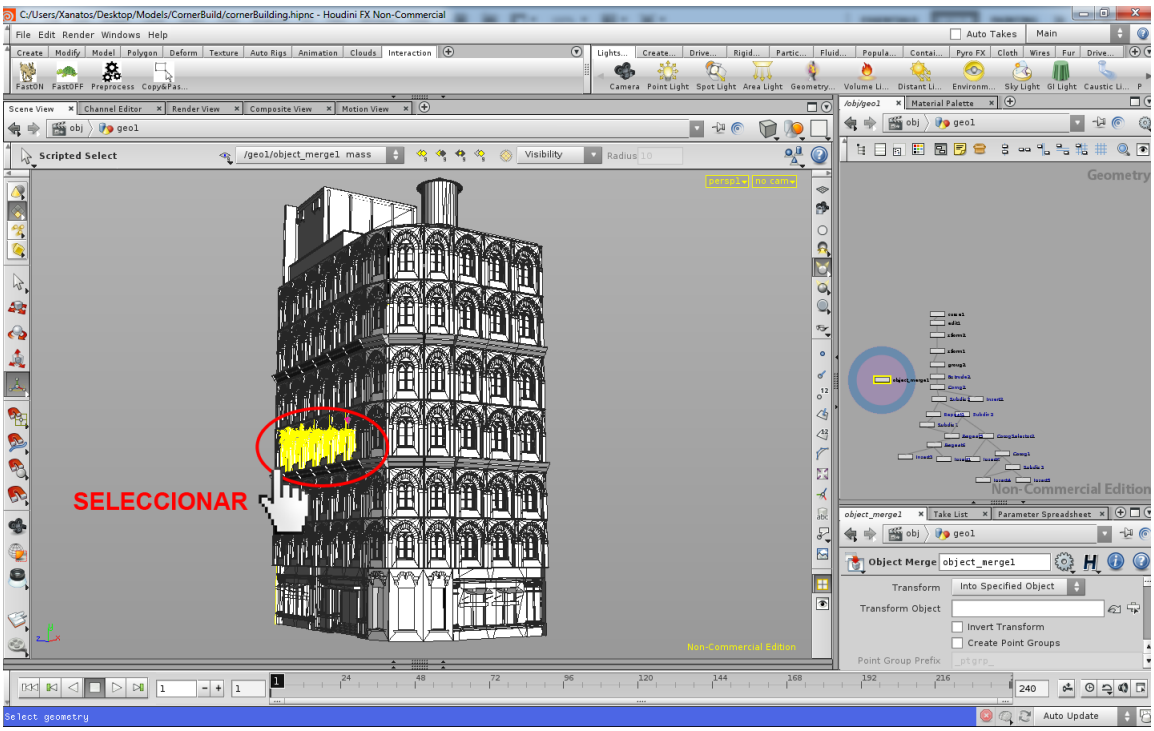


Figura 3.11: Exemple del Pas 9 en el funcionament del Copy&Paste.

Un cop efectuada la selecció només cal clicar en el botó de "Paste Selection" per realitzar l'enganxat. Es pot veure l'acció en la Figura 3.12.

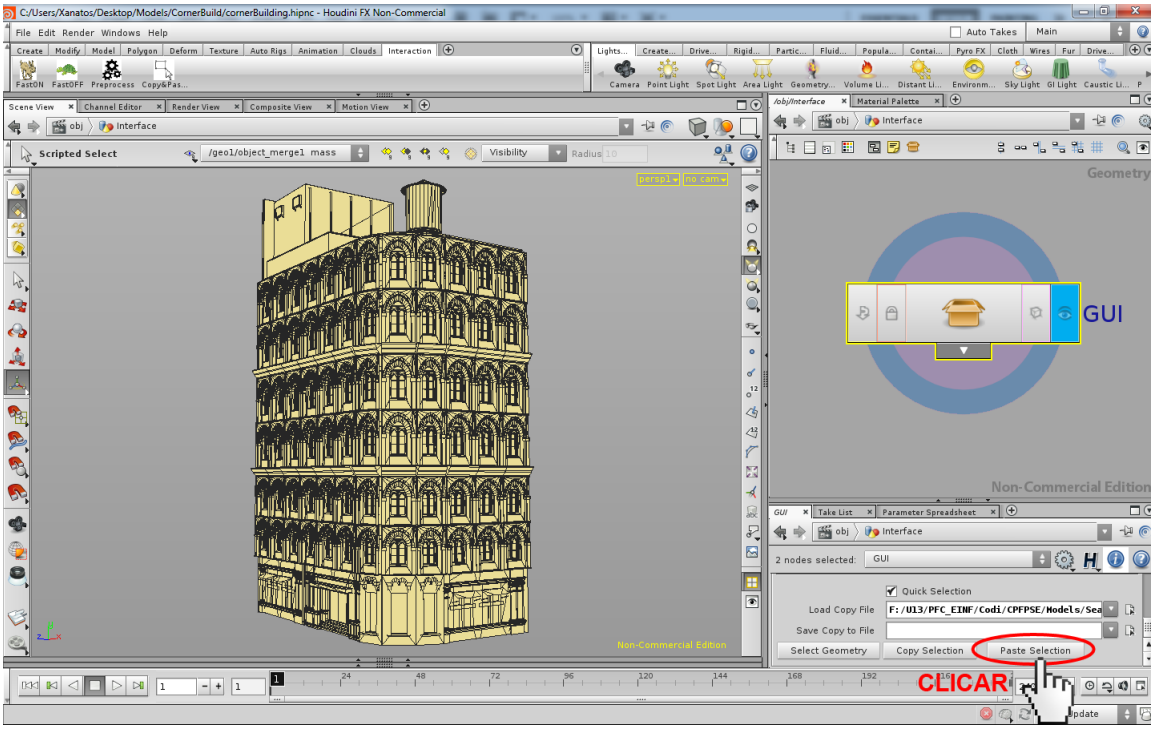


Figura 3.12: Exemple del Pas 10 en el funcionament del Copy&Paste.

Un cop s'ha realitzat l'enganxat es pot veure el resultat en la Figura 3.13.

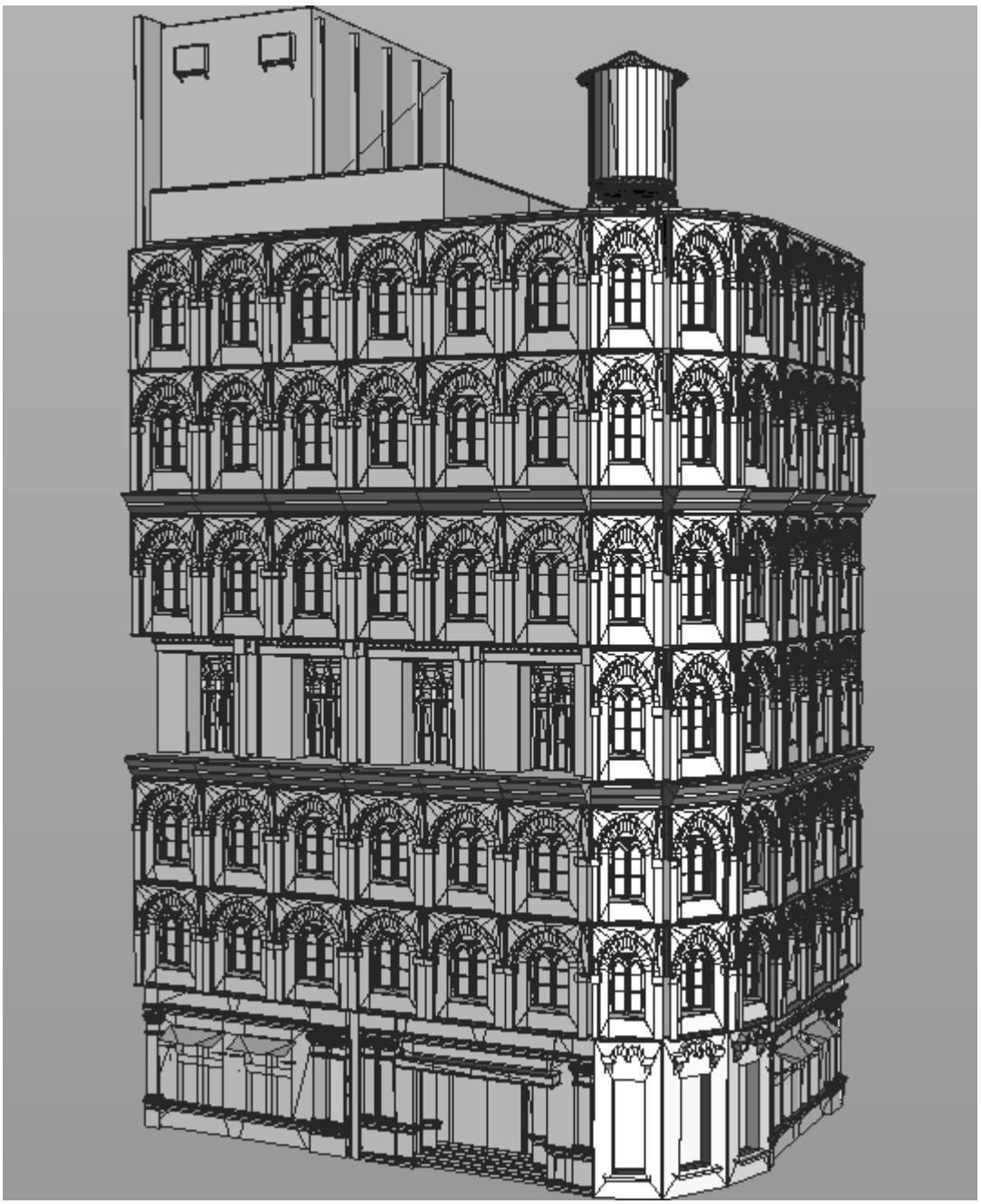


Figura 3.13: Exemple del resultat final de tots els passos del Copy&Paste.