Capítol 1: Introducció

Avui en dia el modelatge de ciutats és un problema obert pel que no existeixen solucions estàndards ni de codi lliure. Aquest és un problema força important per a indústries com la del cinema, la realitat virtual o els videojocs. Dins d'aquest problema es poden presentar sub-problemes interessants, dels quals el modelatge d'edificis i altres estructures arquitectòniques pren un rol fonamental. La complexitat geomètrica que té una ciutat fa que el modelatge d'aquest tipus d'estructures sigui una tasca gens menyspreable.

Qualsevol persona, amb més o menys relació amb el món dels ordinadors, haurà jugat a algun joc d'aventures gràfiques on l'escenari escollit és una gran ciutat, o haurà vist alguna pel·lícula relativament actual on el protagonista viu la seva aventura envoltat de grans edificis. Cada vegada més es fan servir eines informàtiques per a crear aquests escenaris monumentals. Molts d'aquests escenaris han estat creats manualment, o sigui, muntant els edificis un per un fins aconseguir una ciutat.

El modelatge procedural pretén solucionar bona part d'aquest problema utilitzant les seves funcionalitats per a generar de forma sistemàtica la geometria necessària en cada cas. A grans trets, el procediment consisteix en deixar per la màquina la feina de crear la ciutat pròpiament dita i així l'usuari només s'ha de preocupar d'introduir els paràmetres necessaris per aconseguir el seu propòsit.

Existeixen eines de desenvolupament procedural, però el seu ús es troba restringit a unes quantes aplicacions que, generalment, no inclouen edificis.

1.1 Objectius

L'objectiu d'aquest PFC és el desenvolupament d'una eina pel modelatge procedural d'edificis i altres estructures arquitectòniques. El modelatge d'edificis és, per si sol, un bon tema on aplicar-hi la programació procedural. Un edifici normal compte sempre amb elements que es repeteixen en altura i amplada. El fet de "repetir" una tasca suggereix sempre l'aplicació d'algun tipus de procediment per tal de simplificar i reduir la feina de l'usuari a l'hora de desenvolupar aquesta feina.

L'objectiu principal d'aquest projecte es centra en el desenvolupament d'una eina de fàcil ús que permeti a l'usuari obtenir de manera ràpida l'estructura bàsica d'un edifici.

1.2 Abast

Aquest projecte s'ha desenvolupat sobre Houdini, una plataforma genèrica pel modelatge procedural d'objectes. Per aconseguir això s'ha utilitzat com a base la feina descrita a l'article "Procedural Modeling of Buildings", Müller et al, Siggraph 2006. Es pot organitzar la feina seguint una estructura modular. El primer pas és la definició d'un sistema de derivació per a generar edificis: a partir d'una estructura de regles i una llavor, s'apliquen iterativament aquestes regles per aconseguir una estructura donada.

Això implica:

1. Definició d'un conjunt inicial d'estructures de base que es combinaran per crear la forma bàsica de cada edifici.
2. Aplicació d'un conjunt de regles sobre aquesta forma bàsica que generi iterativament les diferents parts de la façana: portes, finestres, xemeneies, balcons, etc.
3. Aplicacions d'aquest sistema per un edifici senzill.
4. Aplicacions per edificis d'oficines.
5. Aplicacions pel modelatge de cases unifamiliars.

1.3 Temporalització

Aquest projecte es va iniciar durant els primers dies de Febrer del 2008, un cop passats els exàmens del primer semestre, i s'ha acabat a finals d'agost.

Inicialment es va projectar la temporalització pensant en l'entrega de Juny, però la dificultat del projecte ha fet allargar el desenvolupament fins a Setembre. La falta de documentació sobre el tema i sobre les eines ha sigut, sens dubte, el retardant més important.

El projecte es pot dividir en tres fases principals. Els primers dos mesos van servir sobretot per a temptejar diferents opcions o camins a seguir: des de la creació de ciutats o carrers fins al modelatge d'edificis independents. Cal dir que abans de fer aquesta elecció calia conèixer bé les eines que s'havien de fer servir, per això no va ser fins a mitjanats de març que no es va definir concretament totes les línies a seguir. A partir d'abril, un cop marcat el camí, calia entrar a fons en el funcionament de les eines. La segona fase del projecte està compresa entre abril i juliol del 2008. Durant aquest període la feina es va centrar, majoritàriament, en la investigació i el desenvolupament del codi font. Cal dir que en tot moment calia indagar de manera exhaustiva ja que contínuament sorgien problemes. Tot i que no eren problemes importants, la desconeixença del funcionament i interacció de les eines retardava molt el ritme de treball. Finalment, la tercera fase del projecte engloba la documentació i els retocs finals del codi. Durant el mes d'agost del 2008 s'ha dut a terme la redacció de la memòria explicativa del projecte així com el refinament del codi per presentar.

La Figura 1 mostra un diagrama de *Gantt* amb la temporalització del projecte.

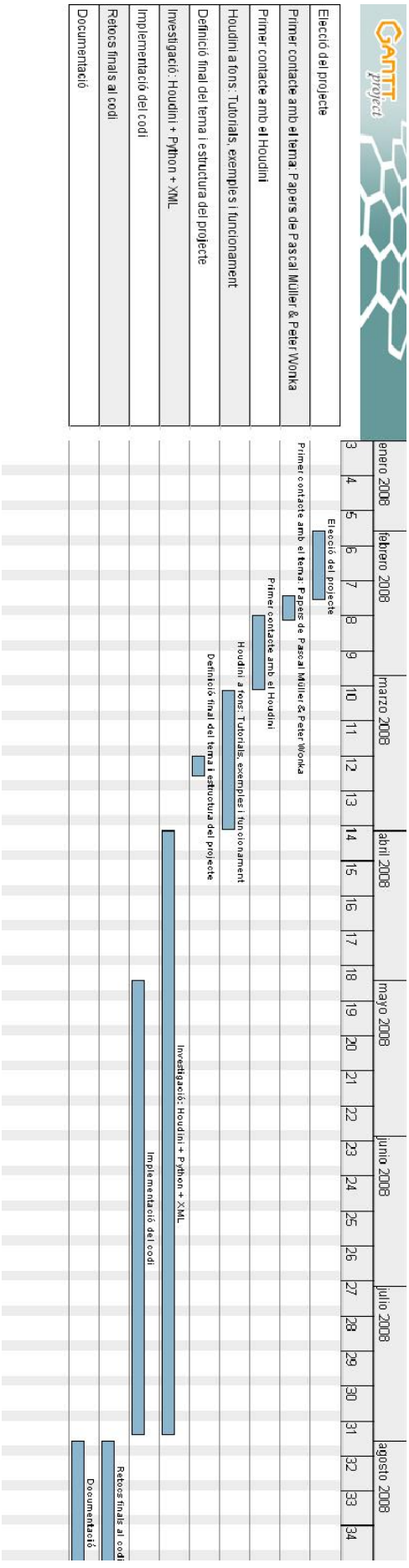


Figura 1: Diagrama de Gantt del projecte

1.4 Estructura del document

Aquesta memòria està estructurada en cinc capítols fonamentals que recullen tota la documentació i explicació necessària del projecte.

El primer capítol conté la introducció al tema central del projecte així com els objectius que es van marcar al inici del mateix i la temporalització estructurada.

Al segon capítol s'expliquen detalladament les eines escollides. Es mostren les funcionalitats de cadascuna d'elles i la seva interacció amb la resta.

El tercer capítol mostra els antecedents en què s'ha basat el projecte. Es fa èmfasi en el treball desenvolupat per Pascal Müller i Peter Wonka dins el camp de la generació de gràfics procedurals.

El capítol 4 conté tota la documentació referent al disseny del projecte. Es mostra el concepte inicial, el desenvolupament i la metodologia seguida a l'hora de crear les estructures de dades, les regles i demés.

El cinquè capítol és complementari al quart, ja que s'hi mostra tota la implementació que s'ha definit al capítol anterior. S'especifiquen tots els conceptes explicats anteriorment i es mostra com s'ha creat el codi del projecte.

El sisè capítol recull els resultats obtinguts amb exemples d'edificis construïts amb el programa.

El setè capítol mostra les conclusions extretes dels resultats obtinguts.

Finalment, el capítol 8 proposa el treball futur que es pot desenvolupar a partir del projecte actual.

Capítol 2: Eines

En la elaboració d'aquest projecte s'han utilitzat bàsicament tres eines diferents: Un modelador 3D, un llenguatge d'scripting i un llenguatge flexible per a definir els edificis.

Inicialment, es va proposar fer servir només el Houdini com a eina de treball per a modelar la geometria. Tot i això, aquest programa sol no donava la possibilitat de programar l'eina de manera sistemàtica, ja que està orientat a donar servei a través de la interfície gràfica d'usuari.

Per tal de poder implementar un mòdul recuperable i editable externament, es va optar per fer servir el Python que ja està incorporat dins el mateix Houdini. D'aquesta manera obtenim una perfecte connexió entre les dues eines. Primer es crea un script que construeixi la geometria, ja sigui directament amb la consola o usant un editor extern, i automàticament podem visualitzar el resultat dins de Houdini.

Finalment, s'ha optat per dotar al sistema d'un procediment de modelatge basat en el llenguatge XML. Aquesta tecnologia permet a l'usuari treballar amb una sintaxi coneguda i relativament senzilla.

2.1 Modelador 3D: Houdini

El primer pas a fer era escollir un modelador 3D adient i preparat per a suportar el tipus de treball que s'havia planificat. Existeixen diferents programes pensats per a desenvolupar aquest tipus de tasca, però sempre amb lleugeres diferències entre ells.

Maya, 3D Studio o Blender són algunes de les opcions que s'havien estudiat.

Maya és un dels softwares més potents del mercat. Destaca sobretot per la seves possibilitats d'expansió i personalització. Utilitza un llenguatge propi anomenat *MEL* que es fa servir per a crear scripts i executar-los sobre el programa.

El 3D Studio és una altra opció dins el món de l'animació 3d. Pertany a la mateixa empresa que Maya però, a diferència d'aquest, 3DStudio és només per Windows. Els dos programes formen part de la mateixa empresa degut a les continues fusions i adquisicions d'empreses més petites propietàries d'aquest software.

Blender és l'opció de software lliure més potent que hi ha al mercat. El programa no té res a envejar als seus germans de pagament, tot i que no disposa de tantes extensions i suplements com ells.

Finalment, i un cop valorat el funcionament dels programes i la feina a desenvolupar es va optar per utilitzar un programa d'animació 3D anomenat Houdini, ja que està especialment pensat per a crear sistemes de modelatge procedural.

A diferència dels altres programes esmentats abans, 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 2 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 és l'espai on es poden visualitzar els resultats usant diferents vistes i es pot seleccionar i modificar objectes. L'arbre de nodes és l'espai on es representen tots els objectes de l'escena en forma de caixetes independents enllaçades les unes a les altres. Finalment, un cop seleccionat un node, podem consultar i modificar les seves propietats a través del sistema de pestanyes que ens facilita la interfície.

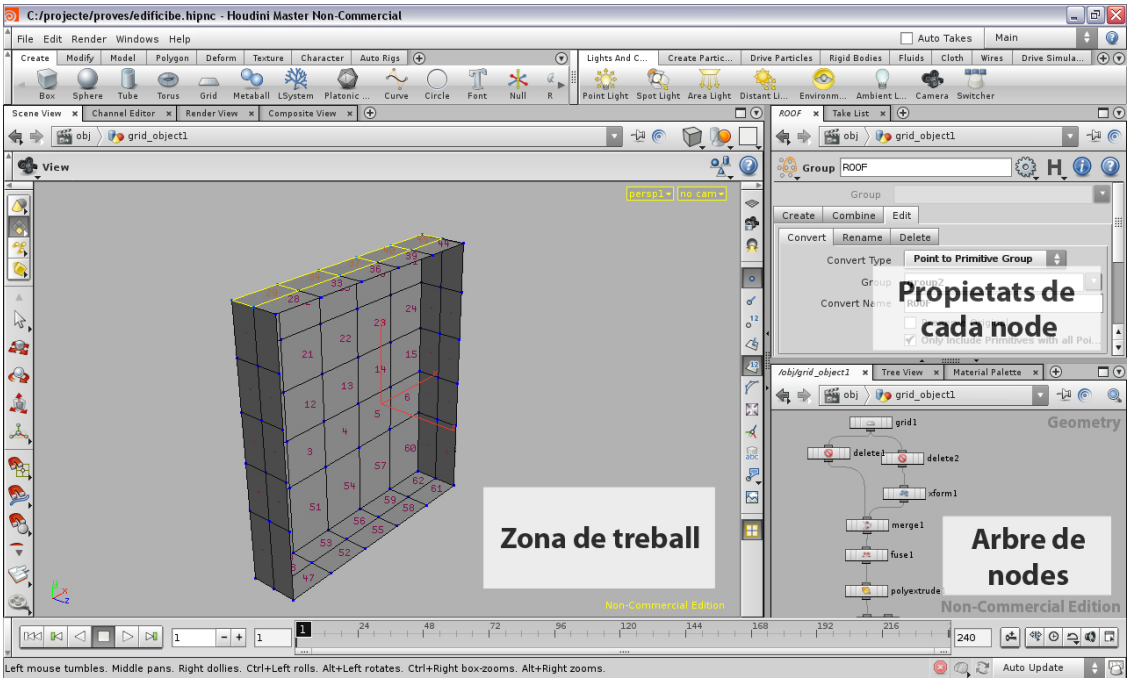


Figura 2: Captura de l'entorn de treball del Houdini.

2.1.1 Exemple de funcionament

A continuació, per il·lustrar el funcionament del Houdini, es crea com a exemple una figura senzilla: un cub, que després es modificarà.

➔ Primer pas: es crea un BOX

Per a crear un cub, senzillament es clica sobre el botó corresponent i el Houdini genera la figura. Simultàniament, s'obtenen dues representacions del mateix objecte. Per una banda, a la zona de treball s'hi genera la geometria en 3D i, per l'altra, es pot visualitzar l'escena en forma d'arbre de nodes, cadascun dels quals representa un objecte o acció.

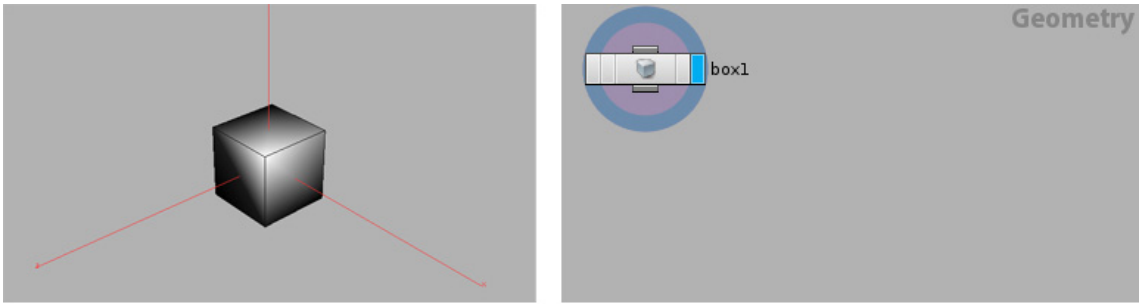


Figura 3: La geometria del cub i el seu respectiu node a l'arbre.

➔ Segon pas: s'aplica al BOX una transformació creant un node POLYEXTRUDE

A continuació, s'aplica a la figura una extrusió geomètrica creant, a partir de les 6 cares del cub, 6 cubs suplementaris. Aquest cop, usant l'arbre de nodes, es crea un POLYEXTRUDE i s'enllaça directament amb el BOX creat anteriorment. La sortida de dades del BOX es connecta amb l'entrada del POLYEXTRUDE.

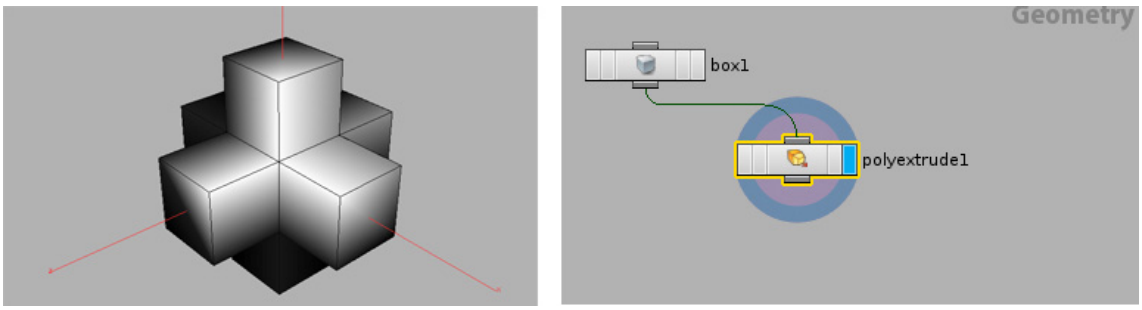


Figura 4: Per a crear una extrusió, cal afegir un node POLYEXTRUDE a l'arbre de nodes i enllaçar-lo amb el BOX.

➔ Tercer pas: s'aplica al sistema un rotació en tots els eixos creant un node TRANSFORM

S'aplica una rotació a la figura usant un node TRANSFORM. Com s'ha fet abans, es connecta la sortida del POLYEXTRUDE amb l'entrada del TRANSFORM.

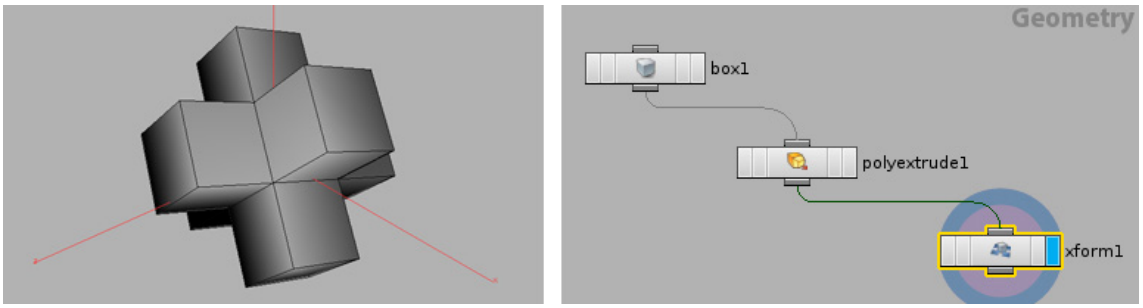


Figura 5: De la mateixa manera que al punt anterior, per a rotar l'objecte, s'aplica un node TRANSFORM i s'enllaça amb el POLYEXTRUDE, formant una cadena.

En general, cadascun dels nodes representa una funció que té una entrada de dades, manipula aquestes dades i retorna un resultat. Amb aquest sistema, s'obté un conjunt d'objectes independents enllaçats entre ells a través d'un flux de dades.

➔ Un exemple: es canvia el BOX per un TORUS

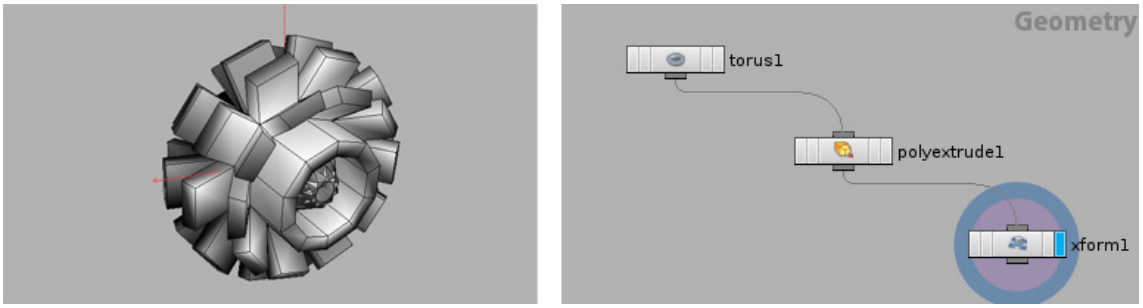


Figura 6: El canvi de l'objecte de base no altera el funcionament dels altres nodes. Senzillament s'obté un resultat diferent.

En qualsevol moment del procés, es pot fer un canvi en un node inicial i aquest s'anirà propagant a través dels seus successors. De la mateixa manera, es pot aplicar una transformació en qualsevol punt i s'obté un nou flux procedural usant els mateixos nodes de base.

2.1.2 Qüestions importants

Per entendre algunes parts importants de la implementació del projecte cal explicar detalladament el funcionament d'alguns dels nodes que es faran servir.

2.1.2.1 Enumeració de primitives - divisions

Molts objectes de Houdini poden ser dividits en primitives enumerades, quelcom significatiu dins aquest projecte. Per tal de poder crear portes, finestres i altres elements, cal poder dividir les parets dels edificis en espais que després ocuparan aquests elements. Houdini fa servir sempre el mateix sistema per enumerar les divisions dels elements: En el cas d'un poliedre de 6 cares, comença per 0 fins a N (sent N el nombre màxim de primitives) i segueix el següent ordre: frontal, posterior, superior, inferior, lateral dret i lateral esquerre. La Figura 7 mostra les divisions que genera el Houdini.

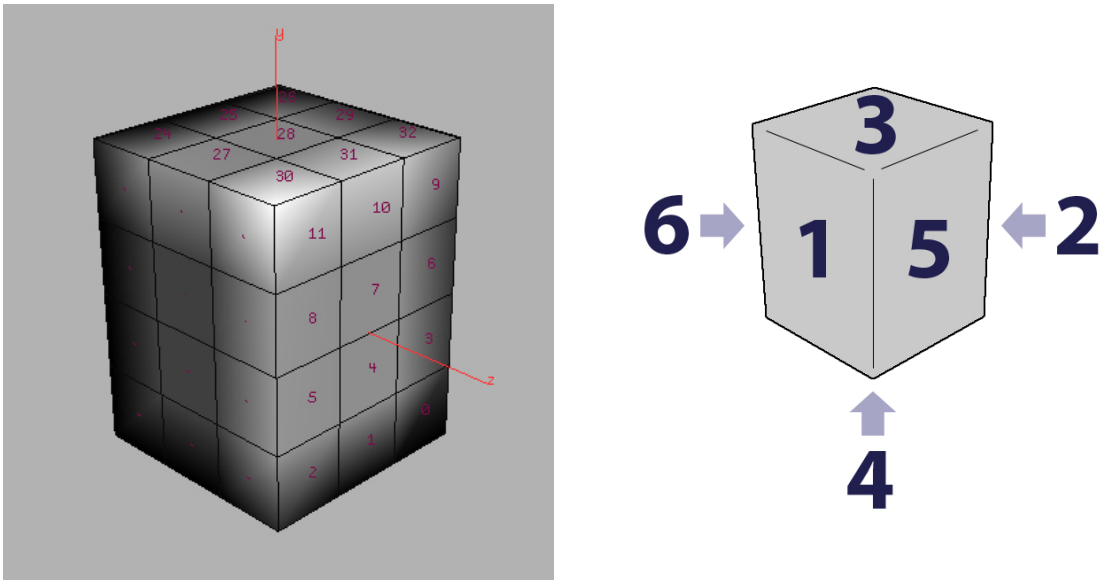


Figura 7: Totes les divisions són enumerades i referenciables.

2.1.2.2 Node DELETE

Aquest node juga un paper molt important dins la implementació del programa. La seva funció bàsica és eliminar geometria. La Figura 8 mostra un node de tipus GRID amb N divisions. Aquestes divisions estan enumerades i, per tant, poden ser referenciades. Tenint en compte això, a la Figura 9 s'ha aplicat un node DELETE que elimina una de cada dues divisions de 0 a N. La Figura 10 mostra el tauler de propietats del node DELETE. Es pot veure les variables modificades per tal que s'eliminin una de cada dues primitives de 0 fins a $N. La variable $N és la referència que fa servir Houdini per a representar el nombre màxim de primitives d'un objecte.

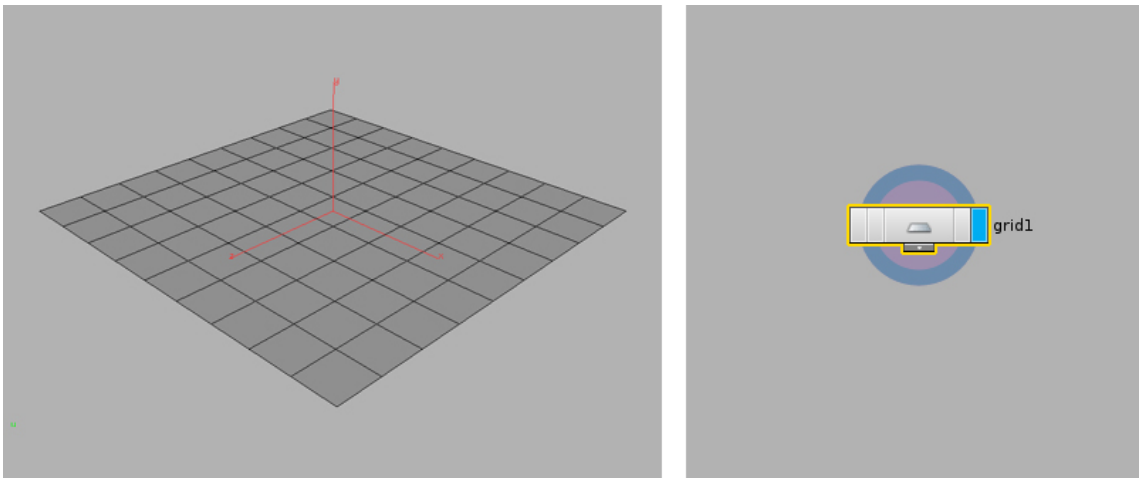


Figura 8: S'ha creat un node de tipus GRID.

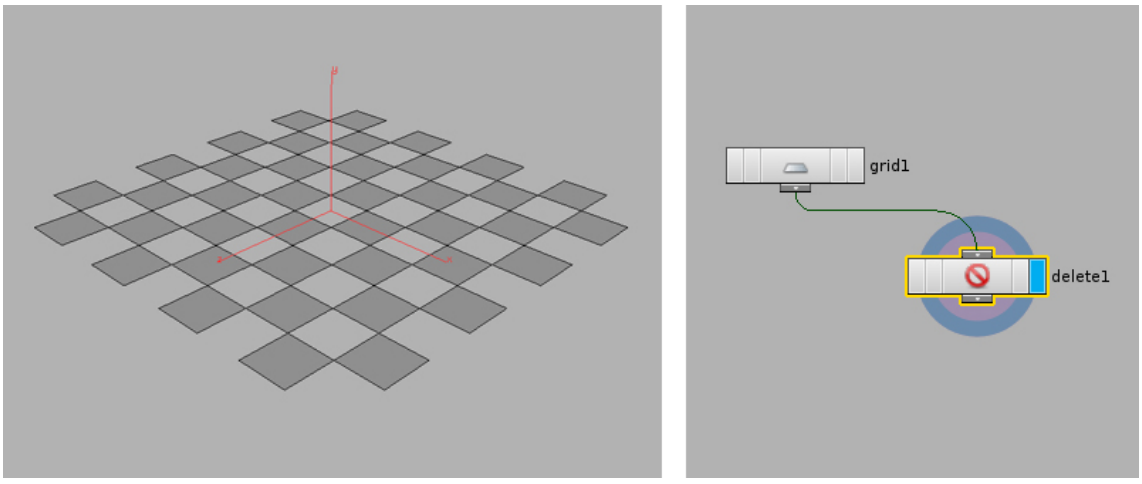


Figura 9: Un cop aplicat el DELETE, s'han eliminat una de cada dues primitives, de 0 a N.

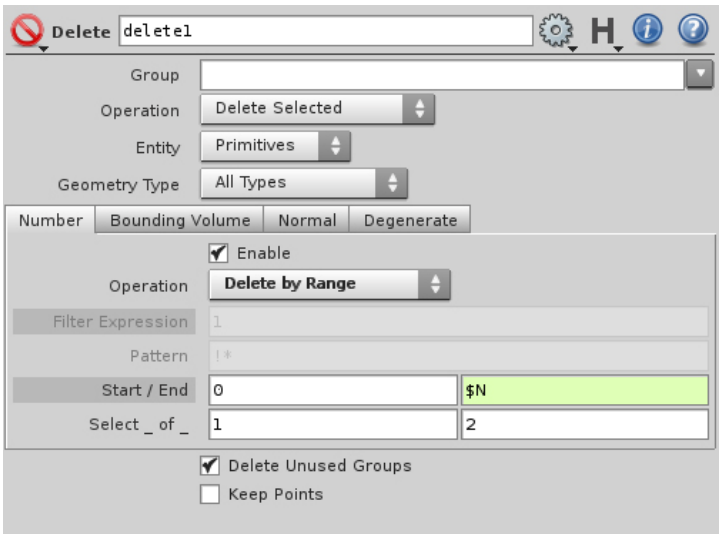


Figura 10: Propeitats del node DELETE. En aquest cas s'eliminen una de cada dues primitives seleccionades dintre del rang 0..N.

2.1.2.3 Node MERGE

Els nodes que genera Houdini, després de cada interacció, són independents i enllaçats per un flux de dades. Cadascun dels nodes pot ser visualitzat de manera separada com un objecte sol. Això vol dir que, després de totes les transformacions i accions fetes, s'obté un arbre de nodes amb moltes ramificacions diferents. Per aquesta raó, cal unir tots aquells nodes que formen part de la figura final utilitzant un *merge*, que no és res més que un altre node que serveix per a unir tots els objectes seleccionats per a tractar-los com un de sol. La Figura 11 mostra la visualització d'un sol objecte ja que és l'únic que està marcat com a visible. En canvi, a la Figura 12 es pot apreciar com connectant els objectes a un MERGE es pot visualitzar tota l'escena.

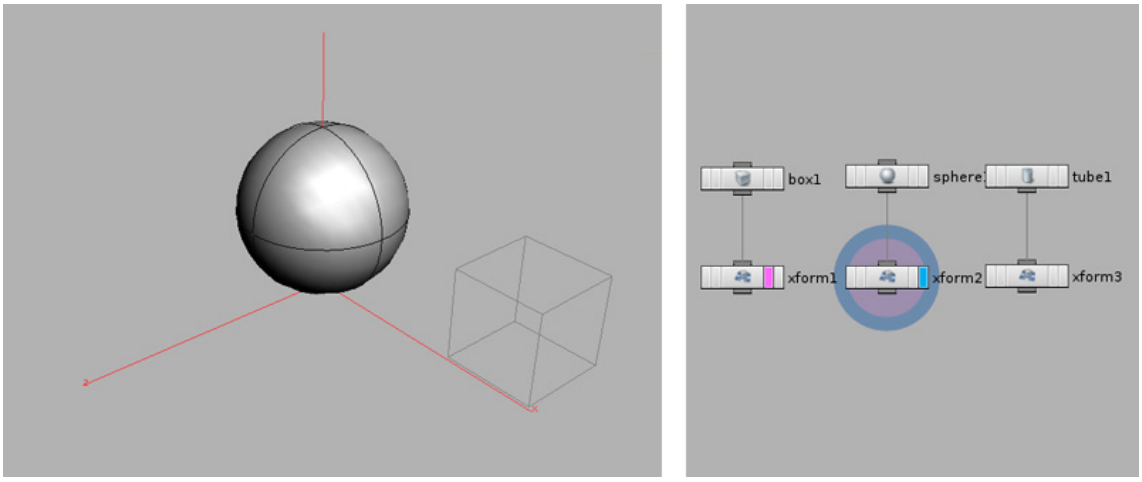


Figura 11: A l'escena hi ha tres objectes: un cub, una esfera i una cilindre. En aquest cas només es veu l'esfera que és la que està marcada i l'estructura del cub.

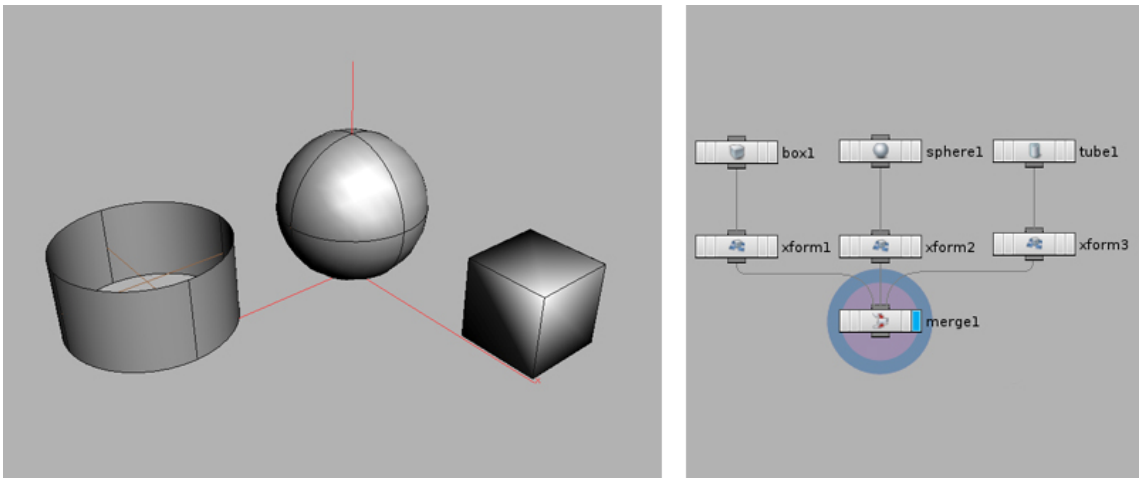


Figura 12: Afegint el node *merge* s'obté un nou objecte que agrupa a tots els anteriors. A partir d'ara els tres elements formen part d'un objecte independent que funciona com qualsevol altre.

2.2 Llenguatges d'scripting

Houdini incorpora per defecte dos llenguatges d'script dins el sistema. El primer d'ells, i més antic, és l'anomenat HSript. Aquest llenguatge és exclusiu de Houdini i existent des de les primeres versions. El segon és el Python, introduït a partir de la versió actual (9.0) amb l'objectiu d'estandarditzar el funcionament i dotar al programa d'un codi conegut.

S'ha decidit utilitzar el Python perquè està present a la versió més recent del Houdini, perquè és més potent que l'HScript i perquè posa a la disposició de l'usuari no només les funcions de Houdini, sinó totes les llibreries existents de 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. L'última versió estable del llenguatge és actualment la 2.5.2 (22 de febrer de 2008) (Es va anunciar l'arribada de la versió 3.0 per a agost de 2008).

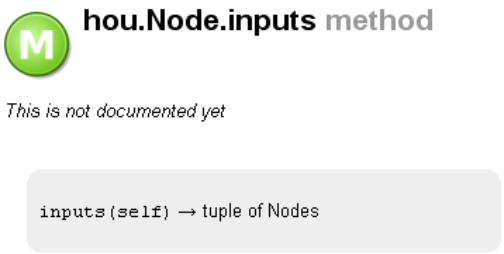
Python és considerat com la "oposició lleial" a Perl, llenguatge amb el qual manté una rivalitat amistosa. Els usuaris de Python consideren a aquest molt més net i elegant per a programar. Python permet dividir el programa en mòduls reutilitzables des d'uns altres programes Python. També hi ha mòduls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

El principal objectiu que persegueix aquest llenguatge és la facilitat, tant de lectura, com de disseny.

2.3 Python dins de Houdini

Actualment encara no està acabada tota la implementació de Python dins de Houdini. Això fa que la majoria de vegades sigui impossible trobar documentació sobre les funcions i procediments que disposa el programa.



2.4 Usant Python

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. 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 feria prement un botó del programa.

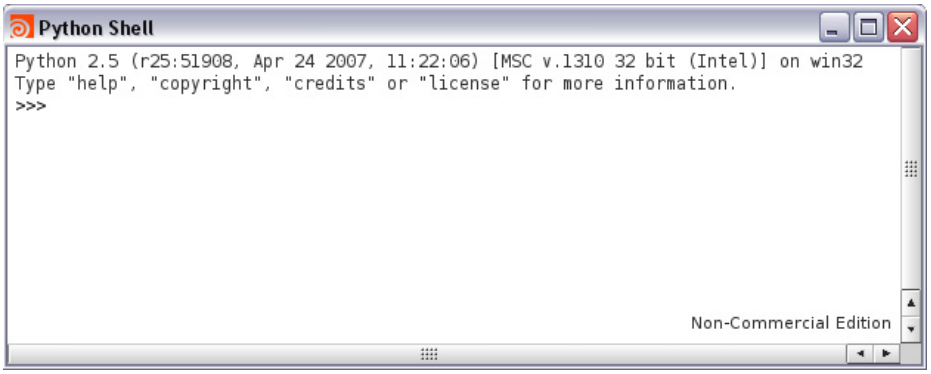


Figura 13: Captura de la consola de Python. Per accedir-hi: Windows > Python Shell.

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 conté totes les funcionalitats, objectes i propietats de Houdini. Aquesta API rep el nom de HOM (*Houdini Object Model*).

Un exemple creant un BOX amb la consola de Python. Primer de tot es s'importa el mòdul *hou*. Seguidament es crea un objecte de tipus *'geo'* que emmagatzemarà tots els objectes. Finalment es crea el box utilitzant al sentència *"createNode()"*.

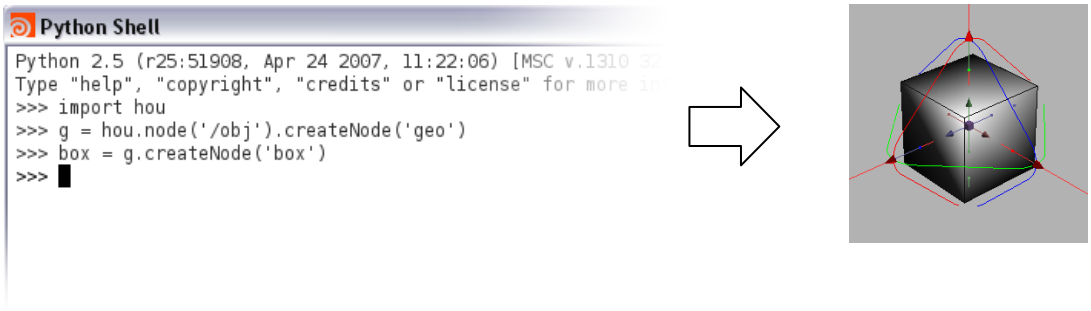


Figura 14: Creació d'un cub utilitzant el Python.

També es poden aplicar transformacions a través de la consola. En aquest cas es crea un node POLYEXTRUDE i s'enllaça amb el BOX utilitzant la sentència "*setFirstInput()*". Llavors es canvia el paràmetre *tz* del node POLYEXTRUDE.

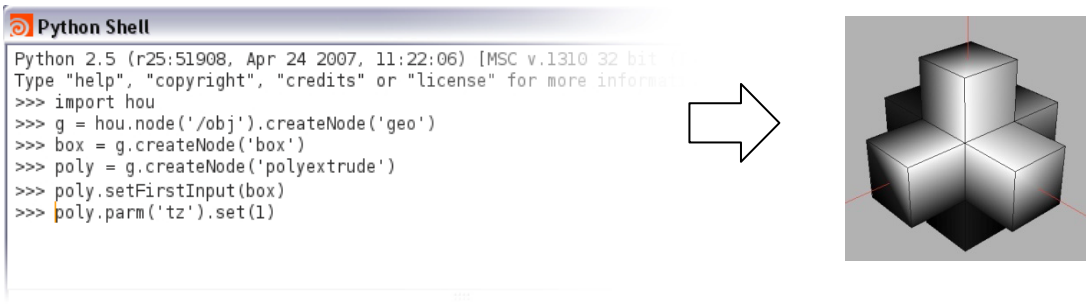


Figura 15: També es poden aplicar transformacions fent servir la consola.

També és possible carregar scripts des de fitxers externs. Aquesta és la manera que s'ha triat per a executar el projecte.

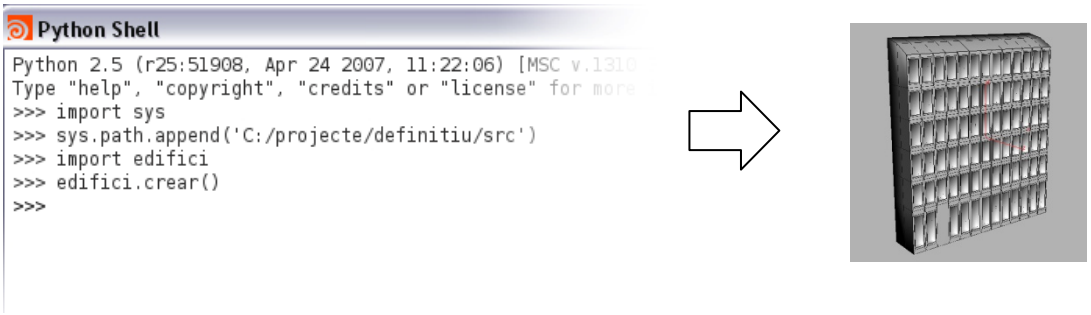


Figura 16: Des de la consola és des de on es carreguen els scripts externs.

Un cop s'ha importat un script aquest queda desat en memòria. Si es fa algun canvi al fitxer de l'script, només cal executar la sentència *reload(mòdul)* i s'actualitzarà la càrrega.

Cada vegada que es crea un objecte dins de Houdini aquest rep un nom que apunta a un espai de memòria on hi ha desades les dades de l'objecte. Quan aquesta creació es fa des de la interfície gràfica, aquest nom és invisible a l'usuari, que només pot veure l'etiqueta que rep l'objecte. En canvi, quan l'objecte és creat des de Python, la manera d'accedir a aquest és mitjançant el nom. En aquest cas, l'etiqueta es comporta com una propietat més, accessible a través dels mètodes apropiats. Aquesta etiqueta pot ser modificada, però amb la restricció que ha de ser única.

2.5 Llenguatge per a el disseny d'edificis

Arribats a aquest punt, l'únic problema que quedava per resoldre era trobar un llenguatge suficientment versàtil, senzill i conegut per a usar-lo com a llenguatge de disseny per a l'usuari final.

Tot i que es podria haver desenvolupat un llenguatge propi, es va optar per fer servir XML. Per a prendre aquesta decisió es van tenir en compte diferents factors, com ara que el Python incorpora per defecte un parser d'XML, el format standard del llenguatge o la possibilitat d'incorporar un sistema de validació (DTD, XML Schema) de manera relativament senzilla.

XML, de l'anglès eXtensible Markup Language, és un metallenguatge extensible, d'etiquetes, desenvolupat pel World Wide Web Consortium [W3C].

És una simplificació i adaptació de l'SGML, i permet definir la gramàtica de llenguatges específics (de la mateixa manera que HTML és, alhora, un llenguatge definit per SGML). Per tant, XML no és realment un llenguatge en particular, sinó una manera de definir llenguatges per a diferents necessitats. Alguns dels llenguatges que empren XML per a la seva definició són XHTML, SVG, MathML.

XML no ha nascut només per a la seva aplicació a Internet, sinó que es proposa com a un estàndard per a l'intercanvi d'informació estructurada entre diferents plataformes. Es pot utilitzar per a bases de dades, editors de text, fulls de càlcul i per moltes altres aplicacions diverses.

XML és una tecnologia relativament senzilla que té al seu voltant altres que la complementen i la fan notablement més extensa, a més de proporcionar-li unes possibilitats molt més grans. A l'actualitat té un paper molt important, ja que permet la compatibilitat entre sistemes, permetent de compartir informació d'una manera segura, fiable i fàcil.

En aquest cas, s'utilitzarà l'XML com una eina de desenvolupament que donarà a l'usuari la possibilitat de crear models d'edificis en 3D d'una manera ràpida i fàcil. El fet que aquest llenguatge sigui estàndard i conegut és un punt a favor a l'hora d'aprendre a fer servir el programa. La sintaxi senzilla i la quantitat d'informació existent sobre XML fa que requereixi un aprenentatge lleuger, fet que facilita la utilització del sistema.

El parser d'XML que incorpora el Python ha facilitat molt la feina a l'hora d'interpretar els fitxers XML's i transformar-los en una estructura de dades accessible des de Python.

Mitjançant aquest parser, s'obté una estructura de dades en forma d'arbre que segueix l'estàndard marcat per el DOM (*Document Object Model*)[W3C]. Així doncs, un cop recuperada l'estructura de l'XML i usant els mètodes que proporcionen les llibreries de Python, es pot accedir a tota la informació necessària per a construir la geometria.

Capítol 3: Treball previ

Dins aquest capítol s'introduiran tots aquells conceptes i referències que han servit d'inspiració per a realitzar el projecte.

El projecte s'ha basat sobretot en el treball realitzat per Pascal Müller et Al. Concretament a l'article *Procedural Modeling of Buildings* [Müller '06]. L'objectiu de Müller i els seus companys és crear un sistema per a construir ciutats de manera automàtica i amb uns resultats realistes. Usant una sintaxi anomenada *CGAshape* i seguint les regles introduïdes amb aquesta sintaxi, s'aconsegueix crear la geometria de qualsevol edifici d'una manera ràpida i fàcil.

3.1 CGA shape

El CGA shape és uns sintaxi basada en regles que serveix per a dissenyar l'estructura dels edificis.

Aquesta sintaxi està basada en els L-Systems [L-S], que s'utilitzen per a descriure el creixement de les plantes.

3.1.1 L-Systems

El funcionament bàsic dels L-Systems consisteix en substituir cadenes de caràcters per altres de més complexes de manera iterativa, tantes vegades com nivells defineix l'usuari.

$$\mathbf{G} = \{V, S, \omega, P\}$$

on

- **V** és un conjunt de símbols que conté elements que poden ser substituïts (variables)
- **S** és un conjunt de símbols que conté elements que es mantenen fixos (constants)
- **ω** és una cadena de símbols de **V** que defineixen l'estat inicial de sistema (inici o axioma)
- **P** és un conjunt de regles o produccions que defineixen la manera com les variables poden ser substituïdes per combinacions de constants i altres variables. Una producció està formada per dues cadenes: el predecessor i el successor.

Les regles gramaticals dels L-Systems s'apliquen de manera iterativa a partir de l'estat inicial.

Vegem-ne uns exemples:

3.1.1.1 Exemple: Creixement de les algues.

variables : A B
constants : ninguna
inici : A
regles : $(A \rightarrow AB), (B \rightarrow A)$

el qual produeix:

n=0 : $A \rightarrow AB$

n=1 : $AB \rightarrow ABA$

n=2 : $ABA \rightarrow ABAAB$

n=3 : $ABAAB \rightarrow ABAABABA$

3.1.1.2 Exemple: Sèrie de Fibonacci

variables : A B
constants : ninguna
inici : A
regles : $(A \rightarrow B), (B \rightarrow AB)$

el qual produeix:

n=0 : A

n=1 : B

n=2 : AB

n=3 : BAB

n=4 : ABBAB

n=5 : BABABBAB

n=6 : ABBABBABABBAB

n=7 : BABABBABABBABBABABBAB

Quan es pren la mida de la longitud de cada cadena, s'obté la famosa sèrie de números de Fibonacci

1 1 2 3 5 8 13 21 34 55 89 ...

3.2 Sintaxi del CGA shape

Com s'ha explicat abans, la gramàtica del CGA shape es basa en un seguit de regles consecutives construïdes seguint sempre un mateix patró:

Id: predecessor: condició → *successor: probabilitat*

on

- **Id** és un enter identificatiu de cadascuna de les regles.
- **predecessor** és aquell element que ha de ser substituït o sobre el qual s'aplica la regla.
- **condició** és el requeriment que s'ha de complir per a executar la regla.
- **successor** és el conjunt d'instruccions a executar.
- **probabilitat** és el coeficient de probabilitat que té la regla per ser aplicada.

Un exemple: Dividir una façana en pisos:

1: fac → *subdiv("Y", 3.5,0.3,3,3,3){floor | ledge | floor | floor | floor}*

El primer paràmetre del successor defineix sobre quin eix (X, Y, Z) es farà la divisió i els següents representen les mides de cadascuna d'elles. Entre els {} s'hi posen les formes que es crearan com a resultat de la regla.

Tots els objectes creats tenen un nom al qual fer referència. Aquest nom pot ser repetit així es pot fer referència a diferents objectes a la vegada i aplicar-hi les regles a tots de cop. A la Figura 17 es pot veure com han quedat les divisions. En aquest cas s'ha creat una planta baixa (*floor*) de 3.5 metres. Com a separador s'ha creat un sortint (*ledge*) de 30 centímetres i, finament, les tres plantes superiors (*floor*) de 3 metres d'alt cadescuna.

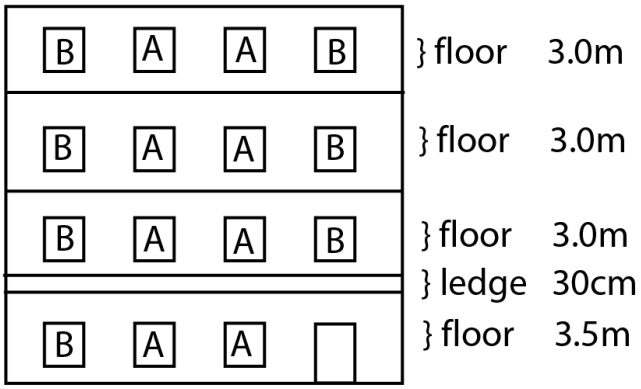


Figura 17: El resultat de l'execució de la regla. Font: "Procedural Modeling of Buildings". M&W (2006)

Capítol 4: Disseny

En aquest capítol es definiran tots aquells conceptes i idees que s'han usat en la elaboració d'aquest projecte.

4.1 Disseny d'un CGA shape propi: XML shape

Com s'ha dit al capítol anterior, el funcionament de projecte es basa en la sintaxi creada per Pascal Müller i Peter Wonka. Tot i això, aquesta gramàtica s'ha hagut d'adaptar a conseqüència de la utilització d'un programa de modelatge 3D ja existent, amb les seves característiques particulars, explicades als apartats 2.1 i 2.3.

Aquesta gramàtica resultant, que a partir d'ara anomenarem XML shape, funciona bàsicament igual que la seva antecessora, exceptuant algunes funcionalitats que no ha calgut dissenyar ja que el mateix Houdini les proporcionava.

Conceptualment, la sintaxi d'ambdós sistemes és similar[1]:

Id: predecessor → *successor*

D'aquesta manera s'obtenen el conjunt de regles necessàries per a dissenyar els edificis.

A continuació s'explicarà el funcionament de les regles de l'XML shape.

[1] Per motius pràctics, en aquesta versió de l'XML Shape, s'ha simplificat la gramática excloent les condicions i el coeficient de probabilitat.

4.2 Definició de les regles

A continuació es definirà el conjunt de regles que s'han dissenyat.

createBase és la regla més bàsica de totes les existents. Genera la geometria inicial en què es basen els edificis.

$$1: \emptyset \rightarrow createBase(type, x, y, z\ [, dx, dy, dz])\{geo\}$$

on

- **type** és el tipus de poliedre que es vol generar
- **x, y, z** són les dimensions del poliedre
- **dx, dy, dz** són paràmetres optatius que especifiquen les divisions que es volen fer en cada dimensió.
- **geo** és el nom del producte resultant de l'execució de la regla

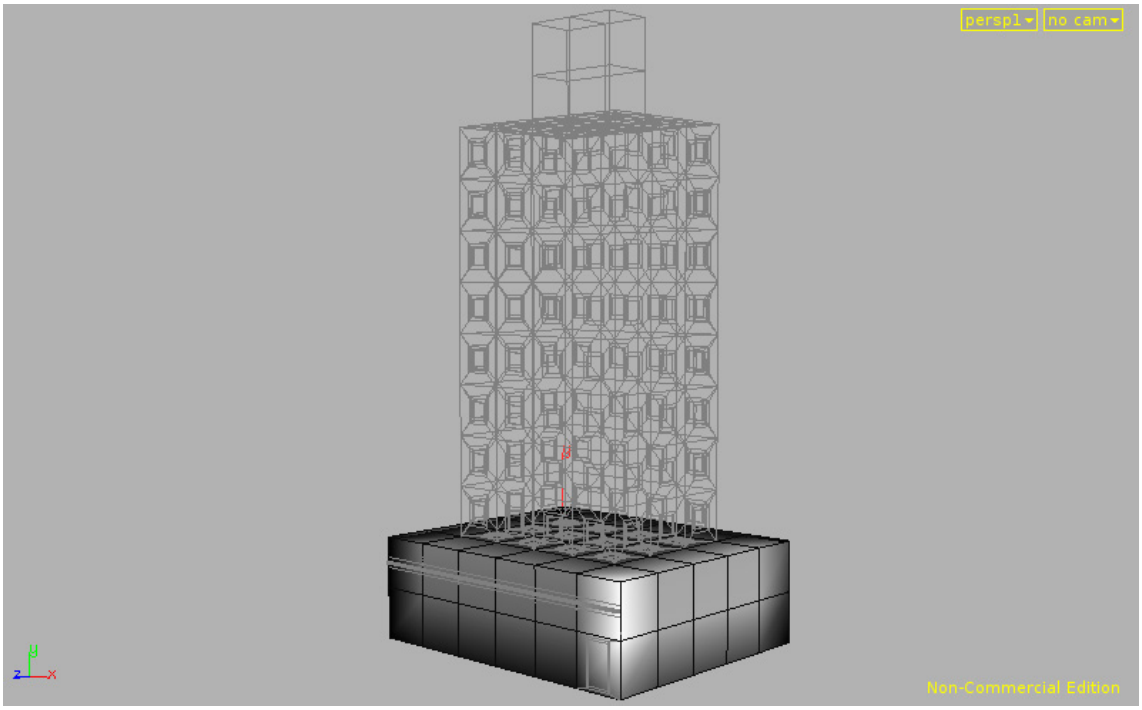


Figura 18: S'ha creat la base per un edifici. L'estructura de línies representa la resta de l'edifici, que es mostra perquè s'entengui la funció del poliedre creat dins el global de la geometria.

Cal observar que la regla createBase no té predecessor perquè és una regla que genera geometria i per tan no s'ha d'aplicar sobre cap altre objecte.

Els paràmetres opcionals *dx, dy, dz* serveixen per a especificar el nombre de divisions que es volen per cada eix. D'aquesta manera s'obtenen cel·les de diferents mides. La Figura 19 mostra un edifici amb el nombre de divisions igual a les mides, mentre que la Figura 20 mostra un edifici amb *dx* i *dz* sent el doble del valor de les mides.

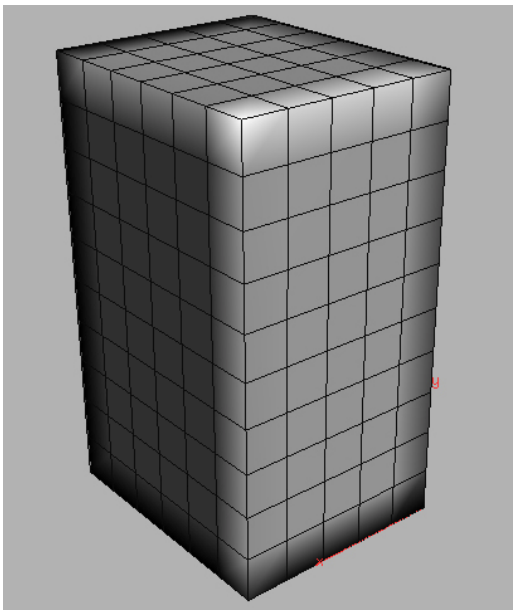


Figura 19: dx, dy i dz són iguals a x, y, z.

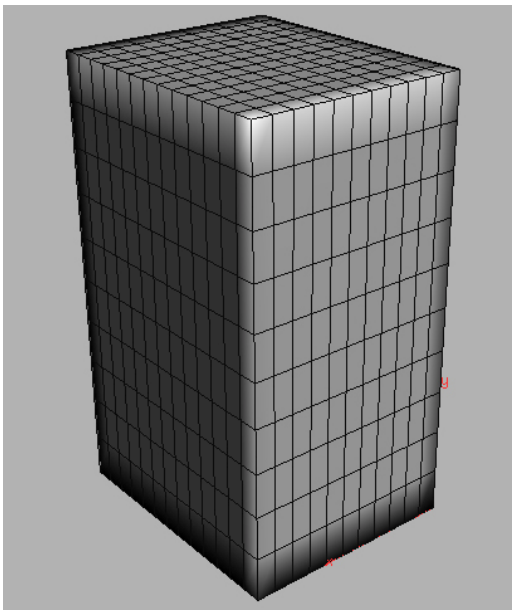


Figura 20: dx i dz són el doble que x i z, mentre que dy = y.

translate serveix per aplicar a un objecte un desplaçament des de la seva posició actual.

1: pred → translate(tx, ty, tz){geo}

on

- **pred** és el predecessor al qual apliquem la regla de translació.
- **tx, ty, tz** són les unitats que es vol desplaçar l'objecte, sobre cada eix.
- **geo** és el nom del producte resultant de l'execució de la regla.

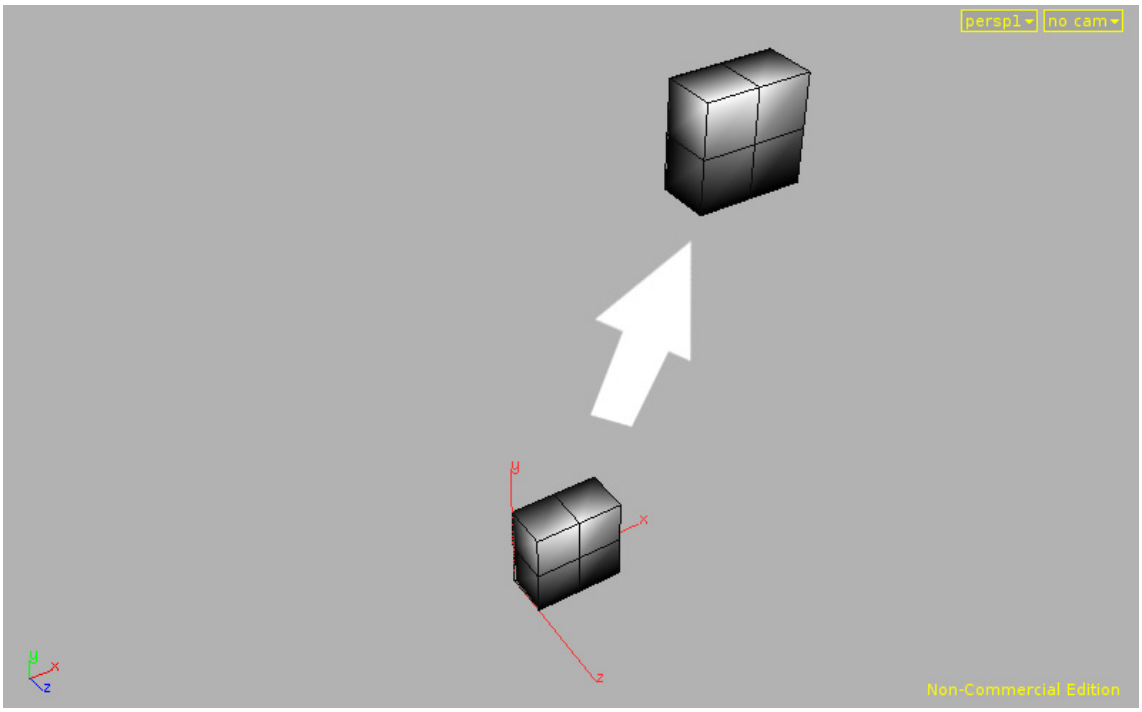


Figura 21: S'ha traslladat l'objecte una distancia x,y,z des de la seva posició actual.

Els desplaçaments de tots els objectes es fan sempre amb unitats de divisions, per exemple, un objecte es pot desplaçar N divisions positives en l'eix de les X. Cada objecte es desplaçarà segons la mida de les seves divisions.

Per exemple (Figura 22), es vol desplaçar dos objectes per l'eix de les X. L'objecte A té mida = 2 i 2 divisions i l'objecte B té mida = 2 i 4 divisions. Per a desplaçar-los la mateixa distància, *txB* haurà de ser 2 vegades *txA*.

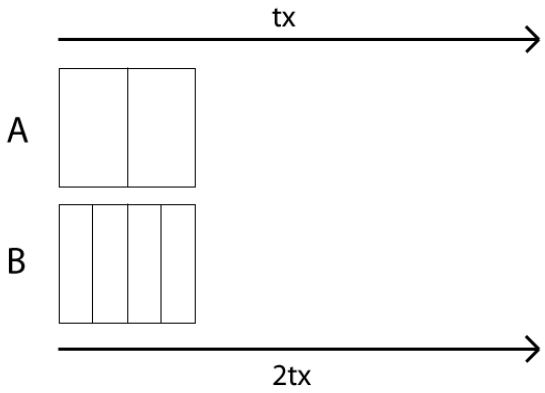


Figura 22: Els desplaçaments es mesuren en divisions, per això és diferent per a cadascun dels objectes.

comp és una acció que separa totes les cares d'un poliedre i les retorna per separat com objectes independents.

1: pred → *comp(){f1, f2, f3, f4, f5, f6}*

on

- **pred** és el predecessor al qual apliquem la regla de translació.
- **f1..f6** són totes les cares del poliedre, independents les unes de les altres.

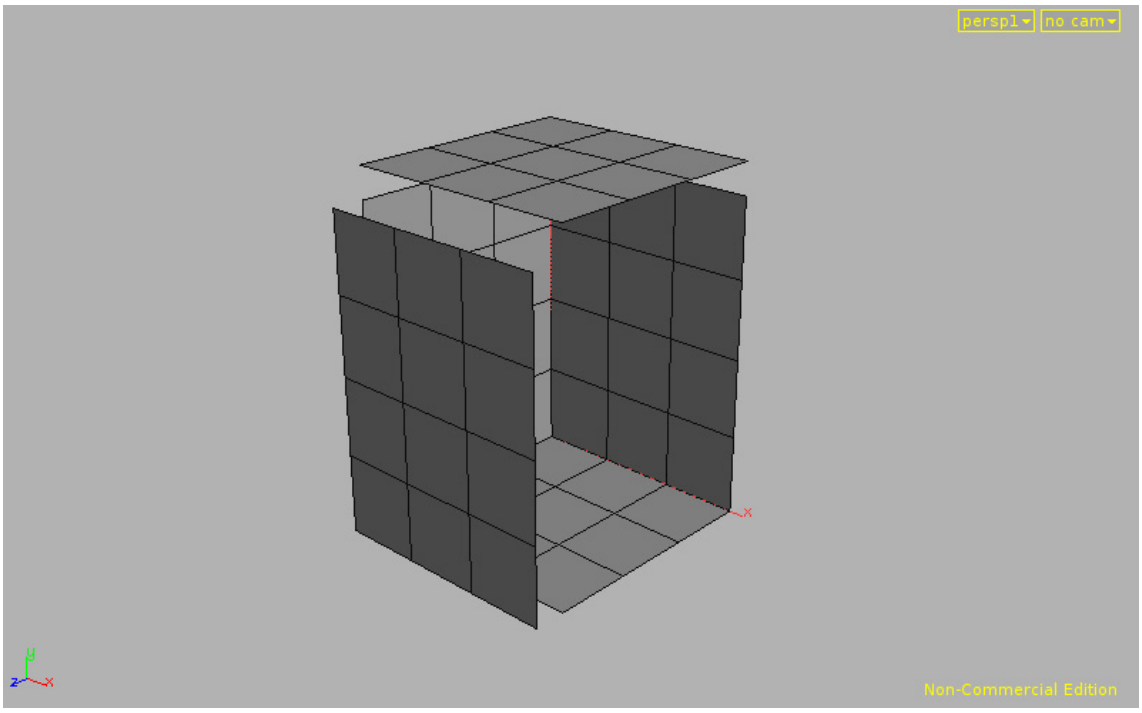


Figura 23: Aquesta figura representa la divisió dels components del poliedre. S'han separat manualment per a facilitar la comprensió del mètode ja que el resultat d'aquesta funció no es pot apreciar visualment perquè els components, tot i estar separats, segueixen al mateix lloc.

subdiv divideix una façana en parts. Tan es pot dividir en l'eix de les X com en l'eix de les Y. Es dividirà en Y quan es vulguin obtenir diferents plantes per separat, per distingir, per exemple, entre planta baixa i plantes superiors. En canvi, es dividirà en X quan es vulguin aconseguir columnes separades, com per exemple, per posar una porta a la part dreta de la planta baixa.

1: pred → subdiv(div1...divN){p1...pN}

on

- **pred** és el nom de la cara a la qual apliquem la divisió.
- **div1...divN** són les mides de les divisions desitjades.
- **p1...pN** són les divisions resultants.

Per a implementar aquesta acció s'ha de tenir en compte que es treballa sobre objectes 2D, per tan, només es fan servir dues components per executar les accions.

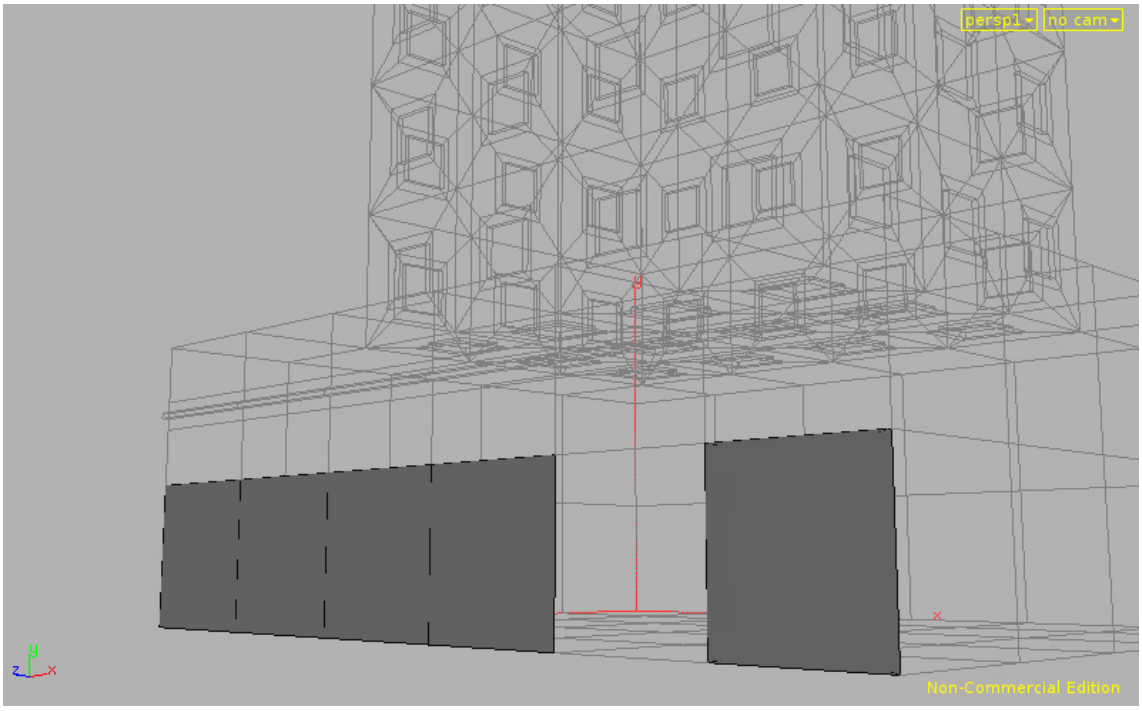


Figura 24: La divisió que s'aplica a aquesta peça és en l'eix de les X. S'obtenen vàries peces independents.

import és una regla que serveix per a afegir geometria detallada als objectes. En aquest cas, els elements disponibles són portes, finestres i sortints.

1: pred → *import(type [, params]){geo}*

on

- **pred** és el tipus d'element al qual volem aplicar-hi un import.
- **type** és el tipus de geometria que es vol importar (*door, window o ledge*).
- **params** són els paràmetres opcionals.
- **geo** és la geometria resultant.

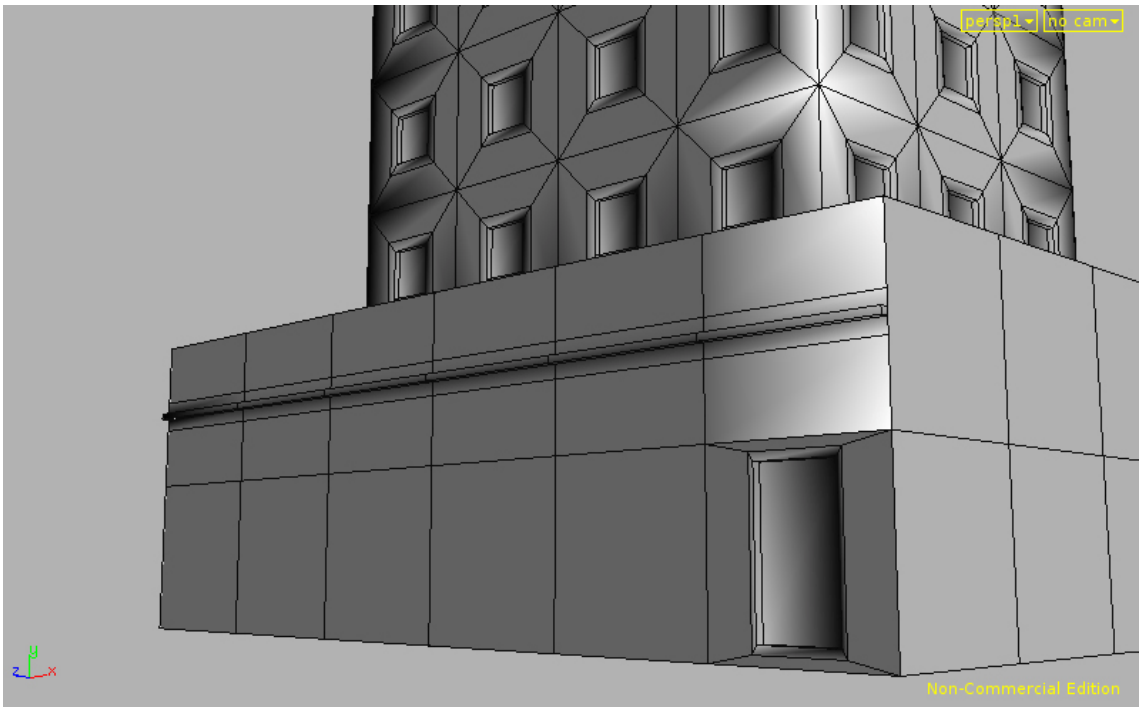


Figura 25: En aquesta imatge s'han aplicat diferents imports. Les finestres de la part superior, el sortint del primer pis i la porta de la planta baixa.

En el cas del *ledge*, es poden afegir paràmetres per a definir diferents opcions:

1: pred → *import(type [,orientation][,location][,noWall]){geo}*

on

- **orientation** és l'orientació del *ledge*. Pot ser horitzontal (H) o vertical (V).
- **location** és la posició del *ledge*. És un número entre -2 i 2 que col·loca el *ledge* més amunt o més avall en el cas horitzontal, o més a l'esquerra o a la dreta en el cas vertical.
- **noWall** és un flag que indica si el *ledge* es vol posar sol o amb una paret de fons.

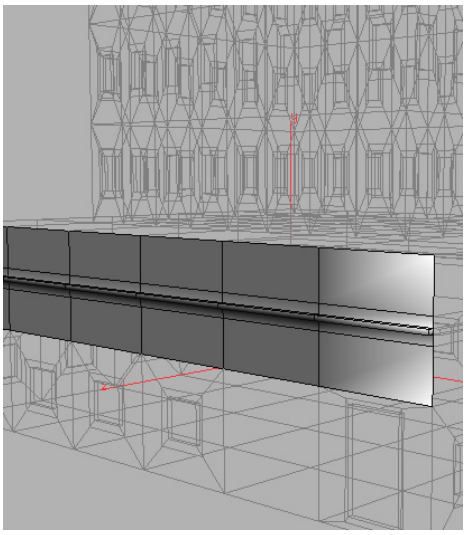


Figura 26: Aquest *ledge* està col·locat en horitzonal (*orientation = 'H'*), al centre (*location = '0'*) i amb *noWall = False*.

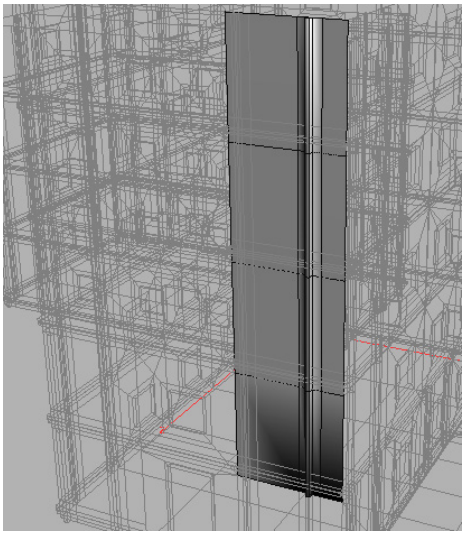


Figura 26: Aquest *ledge* està col·locat en vertical (*orientation = 'V'*), a la dreta (*location = '1.2'*) i amb *noWall = False*

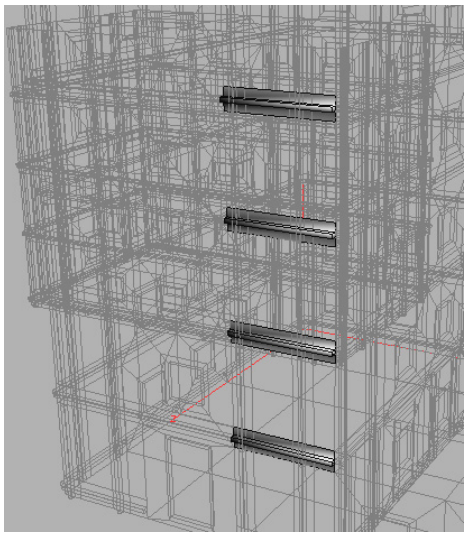


Figura 27: Aquest *ledge* està col·locat en horitzontal (*orientation = 'H'*), a baix (*location = '-2'*) i amb *noWall = True*

4.3 Formes geomètriques bàsiques

Per a començar a generar els edificis és necessari basar-se en algun tipus de forma i, a partir d'aquí, anar modelant la geometria segons es vulgui.

Inicialment s'han partit des de la figura més bàsica, el cub. A partir d'aquesta forma, es generen tots els objectes necessaris. Un cop es té un cub, és fàcil canviar-li la mida o traslladar-lo.

L'any 1985 l'arquitecte suïs Le Corbusier [LC] va definir les bases del que, anys després, concretament al 1993, es convertiria, de la mà d'Schmitt [SCH], en els models de l'arquitectura moderna. Aquest models (veure Figura 27) es basen en la redimensió, translació i unió de caixes formant conjunts geomètrics més complexos, seguint les formes d'algunes de les lletres de l'abecedari com ara la L, la H, la U i la T.

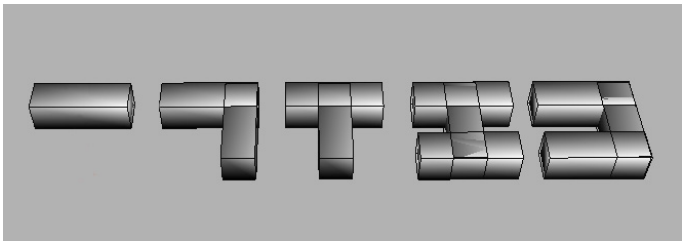


Figura 27: Formes bàsiques de l'arquitectura moderna[2].

4.4 Occlusion & Snapping

La ocultació d'elements darrera d'altres (occlusion) i la superposició d'objectes com finestres o portes al moment d'ajuntar dos parts d'un mateix edifici (snapping), és un problema important dins el modelatge d'edificis. Com que l'usuari executa una funció automàtica, aquests fenòmens poden escapar al seu control.

En aquest projecte s'ha optat per a solucionar aquest problema utilitzant sempre nombres enters per a tots els tipus de definicions i pels canvis a la geometria. Per exemple, la longitud d'un poliedre només es pot especificar en nombres enters. O quan es desplaça un objecte només es pot moure N unitats enteres. Fent servir aquest sistema s'eviten els problemes de *snapping* i *occlusion* d'una manera senzilla i eficaç. També cal afegir que tots els objectes tenen divisions de mida unitària, fet que facilita la creació de les finestres o la translació del mateix objecte.

[2] En aquest exemple hi falta el cilindre, que hem obviat en aquest projecte.

4.5 Sintaxi XML

Com s'ha comentat al capítol 2, el llenguatge escollit per a definir els edificis és XML. S'ha creat una sintaxi basada conceptualment amb el CGA shape de P.M.&P.W. i s'ha adaptat seguint l'estàndard que marca XML.

Un esquema conceptual del codi:

```
<XMLshape>
      <rule id="1">
            <predecessor name="pred"></predecessor>
            <successor>
                  <action name="act">
                        <param name="p1" value="x"/>
                        <param name="p2" value="y"/>
                        ...

                        <product name="prod"/>
                        ...
                  </action>
            </successor>
      </rule>
      ...
</XMLshape>
```

Les regles en general queden escrites de la següent manera:

Id: predecessor → successor

Un exemple per a identificar els elements:

1: pred → act(p1, p2){prod}

Seguint aquesta especificació, s'han de crear totes les regles necessàries per a generar la geometria que calgui.

4.5.1 Tags XML

<XMLshape> és el tag contenidor de tots els objectes. Serveix per a marcar l'inici i el final del fitxer XML, com també per identificar el tipus de fitxer que és.

```
<XMLshape>
      ...
</XMLshape>
```

<rule> defineix cadascuna de les regles del fitxer. Totes les regles estan numerades i són executades de manera seqüencial.

```
<rule id="n">
```

```
        ...
</rule>
```

<predecessor> és el tag que representa l'objecte al qual es vol aplicar l'acció descrita dins la regla. A part de la primera regla, que no s'aplica sobre cap objecte, ja que es crea geometria de nou, totes les altres s'han d'aplicar a un objecte predecessor determinat.

```
<predecessor name="nom"></predecessor>
```

S'ha definit aquest tag amb un inici i un tancament pensant en la possibilitat, en un futur, d'afegir-hi paràmetres a dins del tag.

<successor> empaqueta totes les accions i paràmetres que es faran servir per a aplicar l'acció sobre el predecessor.

```
<successor>
        ...
</successor>
```

<action> és l'acció que s'ha d'aplicar sobre el predecessor. Segons el tipus d'acció descrit, s'executarà un mètode o un altre.

```
<action name="nom">


        ...
</action>
```

Es pot aplicar més d'una *action* sobre un mateix predecessor al definir-les totes dins d'un únic *successor*.

<param> són els tags que representen els paràmetres necessaris per a dur a terme l'acció requerida. Poden ser mides, coordenades, definicions de tipus....

```
<param name="nom" value="valor"/>
```

<product> és el resultat d'aplicar l'acció a sobre el predecessor. Representa l'objecte resultant de les transformacions fetes per l'acció.

```
<product name="nom"/>
```

4.5.2 Parser d'XML i DOM

Per a poder crear una estructura de dades a partir del fitxer XML cal utilitzar el parser que porta incorporat el Python. Exitsteix un mòdul anomenat *dom* que conté totes les eines necessàries per a tractar els fitxers XML i recollir-ne les dades. En aquest cas s'ha utilitzat una simplificació d'aquest mòdul, anomenat *minidom*.

Un cop parsejat el fitxer, s'obté una estructura de dades en forma d'arbre, composta per nodes que segueixen l'especificació marcada per l'estàndard del DOM (*Document Object Model*).

Cadascun d'aquests nodes conté tota la informació relativa a un sol tag XML del fitxer. Així doncs, si s'agafa com a base l'esquema mostrat abans, s'obtindria un arbre com el mostrat a la Figura 28.

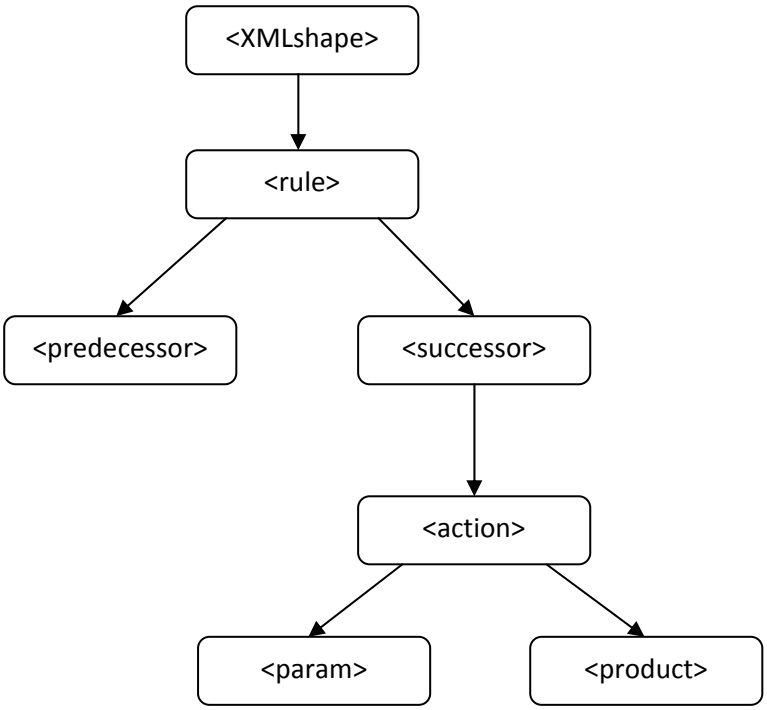


Figura 28: Representació conceptual de l'arbre de nodes del DOM, corresponent a l'apartat 4.5.

Utilitzant els mètodes que ens proporciona el *minidom* es pot navegar a través de tots els nodes i desar les dades que es considerin necessàries.

4.6 Un exemple senzill

A continuació es mostrarà, pas per pas, la construcció d'un edifici de negocis utilitzant com a base tres poliedres i aplicant-hi les transformacions pertinents.

Primer de tot cal especificar que estem utilitzant la sintaxi XMLshape:

```
<XMLshape>
    ...
</XMLshape>
```

A continuació, dins d'aquests marcadors, ja es poden començar a especificar les regles. Es comença definint la primera regla per a generar els tres poliedres que es faran servir com a base. Es farà servir l'acció *createBase*, dins de la qual es defineixen les mides del poliedre. Cadascuna de les *actions* retorna un *product*, que és la geometria resultant d'aplicar l'acció (veure Figura 29).

```
<rule id="1">
      <predecessor name="base"></predecessor>
      <successor>
            <action name="createBase" >
                  <param name="type" value="box" />
                  <param name="sx" value="6" />
                  <param name="sy" value="2" />
                  <param name="sz" value="5" />

                  <product name="b1" />
            </action>
            <action name="createBase" >
                  <param name="type" value="box" />
                  <param name="sx" value="4" />
                  <param name="sy" value="8" />
                  <param name="sz" value="3" />

                  <product name="b2" />
            </action>
            <action name="createBase" >
                  <param name="type" value="box" />
                  <param name="sx" value="2" />
                  <param name="sy" value="2" />
                  <param name="sz" value="1" />

                  <product name="b3" />
            </action>
      </successor>
</rule>
```

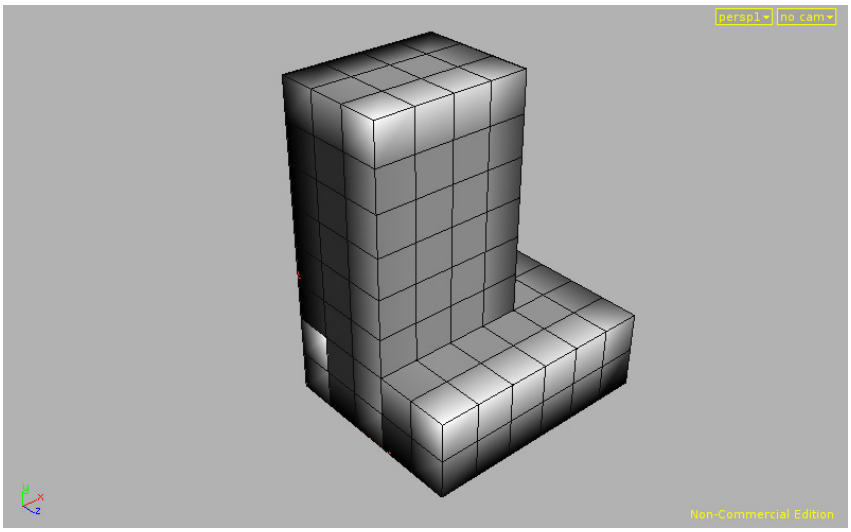


Figura 29: Els tres prismes s'ha creat, però estan superposats a l'origen i, per tant, amaguen al més petit

El següent pas és traslladar els objectes a la posició desitjada. S'utilitza l'acció *translate* i s'hi especifica la distància que es vol moure. Es desplaça l'objecte a partir de la seva posició actual. S'ha de tenir en compte sempre sobre quin objecte s'aplica l'acció (*predecessors*), com es mostra a la Figura 30.

```
<rule id="2">
      <predecessor name="b2"></predecessor>
      <successor>
            <action name="translate" >
                  <param name="tx" value="1"/>
                  <param name="ty" value="2"/>
                  <param name="tz" value="1"/>

                  <product name="b2_t"/>
            </action>
      </successor>
</rule>
<rule id="3">
      <predecessor name="b3"></predecessor>
      <successor>
            <action name="translate">
                  <param name="tx" value="2"/>
                  <param name="ty" value="10"/>
                  <param name="tz" value="2"/>

                  <product name="b3_t"/>
            </action>
       </successor>
</rule>
```

Cal observar que els paràmetres del *translate*, en aquest cas, són equivalents a les mides de l'objecte, ja que la mida és igual al nombre de divisions.

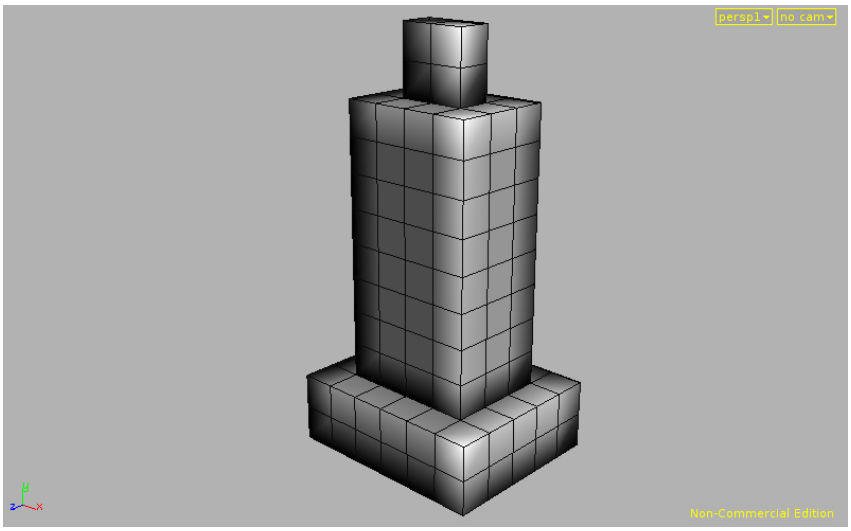


Figura 30: Un cop aplicat el *translate* ja es pot començar a entreveure la forma de l'edifici.

Un cop posicionada tota la geometria, és el moment d'aplicar els detalls. Primer de tot cal separar les cares del poliedre per a poder aplicar les accions a una sola cara en concret. Si no es fes així, els detalls s'aplicarien a totes les cares de la figura, el qual no és molt adient si per exemple, es vol posar una porta. Així doncs, i centrats a la peça que representa la planta baixa, es procedeix primer a separar les cares del poliedre (veure Figura 31).

```
<rule id="4">
      <predecessor name="b1"></predecessor>
      <successor>
            <action name="comp" >

                  <product name="facade"/>
                  <product name="sidewing"/>
                  <product name="floor"/>
                  <product name=”roof"/>
                  <product name=”sidewing"/>
                  <product name="sidewing”/>
            </action>
      </successor>
</rule>
```

En aquest exemple, el poliedre s'ha separat en 6 cares, que reben els noms "*facade*", "*sidewing*", "*floor*", "*roof*", "*sidewing*", "*sidewing*". Cal observar que el nom "*sidewing*" s'ha repetit per després poder aplicar-hi les mateixes transformacions de cop.

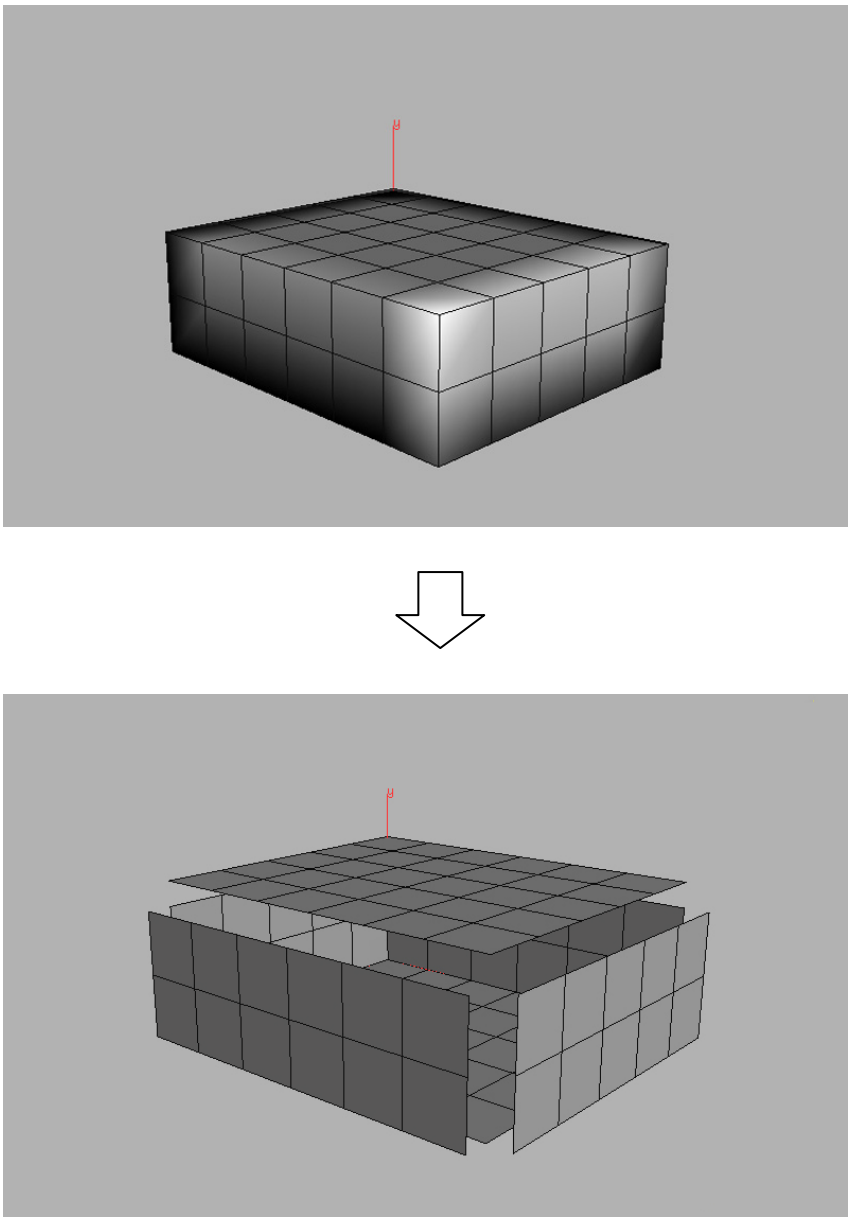


Figura 31: Per una millor comprensió del resultat, s'han separat les cares ja que l'acció *comp* no té resultats visualment apreciables, ja que aquesta acció només separa els components.

En aquest cas, el que es necessita és la façana de davant, per tan, s'aplica a *facade* l'acció *subdiv* en l'eix de les Y per tal de separar la planta baixa de la primera planta.

```
<rule id="5">
      <predecessor name="facade "></predecessor>
      <successor>
            <action name="subdiv" >
                  <param name="axis" value="Y"/>
                  <param name="div1" value="1"/>
                  <param name="div2" value="1"/>

                  <product name="groundFloor"/>
                  <product name="ledgeFloor"/>
            </action>
      </successor>
</rule>
```

Les plantes reben el nom "*groundFloor*" per a la planta baixa, i "*ledgeFloor*" per la primera planta (veure Figura 32).

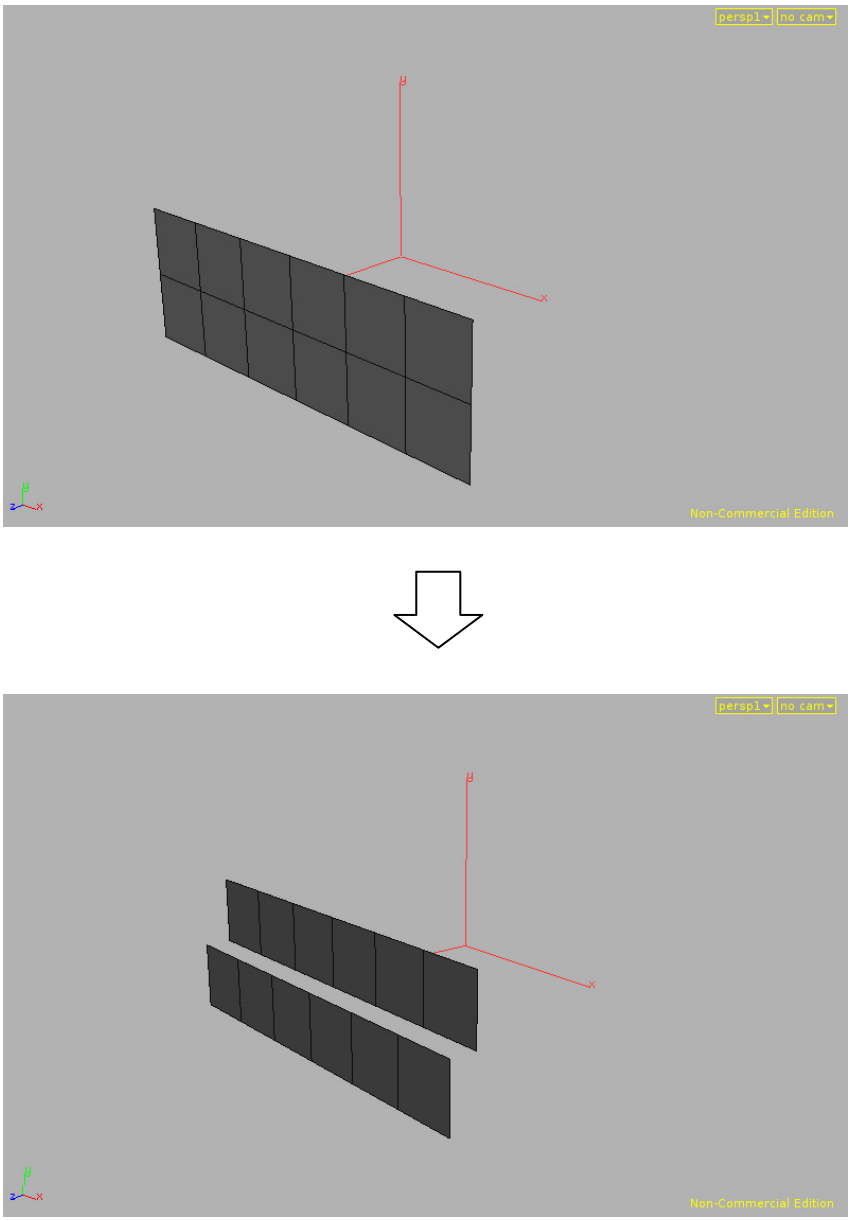


Figura 32: En aquest dibuix, les dues files són objectes diferents (*groundFloor* i *ledgeFloor*). Igual que amb el *comp*, els resultats no són apreciables visualment perquè el que es fa és únicament és separar un objecte en dos o més.

Ara s'aplica a la planta baixa un altre *subdiv*, aquest cop en l'eix de les X, per tal d'aconseguir un element per separat que servirà de porta (veure Figura 33).

```
<rule id="6">
      <predecessor name="groundFloor"></predecessor>
      <successor>
            <action name="subdiv" >
                  <param name="axis" value="X"/>
                  <param name="div1" value="1"/>
                  <param name="div2" value="5"/>

                  <product name="door"/>
                  <product name="wind_GF"/>
            </action>
      </successor>
</rule>
```

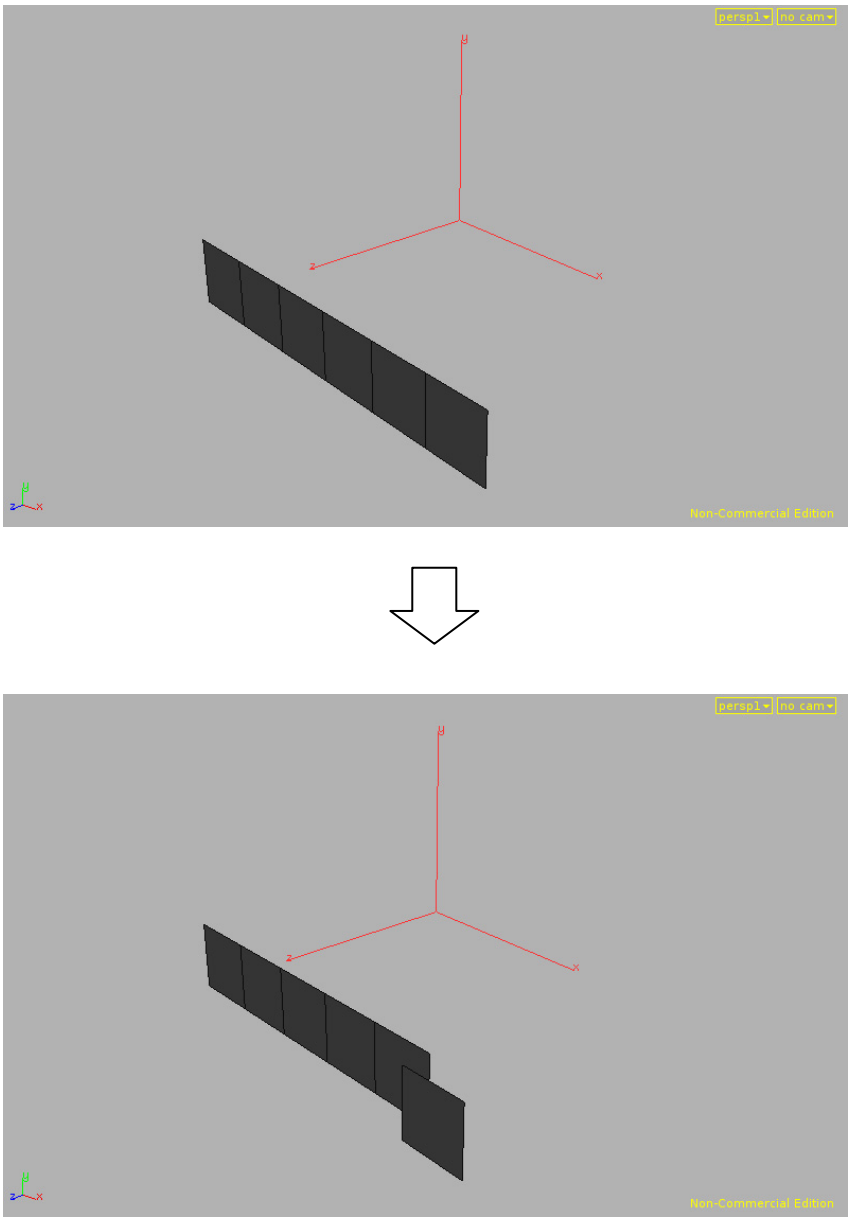


Figura 33: És un dels objectes resultants del *subdiv* en X, el *door*.

Cal dir que s'hauria obtingut el mateix resultat si primer s'hagués fet el *subdiv* en X i després en Y, però en aquest cas, després s'haurien hagut de fer més passos a l'hora de posar-hi geometria ornamental (portes, finestres), ja els objectes estarien tots separats, un per un.

Ara s'aplica a l'objecte resultant l'acció *import*, definint el tipus de geometria que es vol crear. En aquest cas, una porta. El resultat es pot veure a la Figura 34.

```
<rule id="9">
      <predecessor name="door"></predecessor>
      <successor>
            <action name="import" >
                  <param name="type" value="door"/>

                  <product name="doorOk"/>
            </action>
      </successor>
</rule>
```

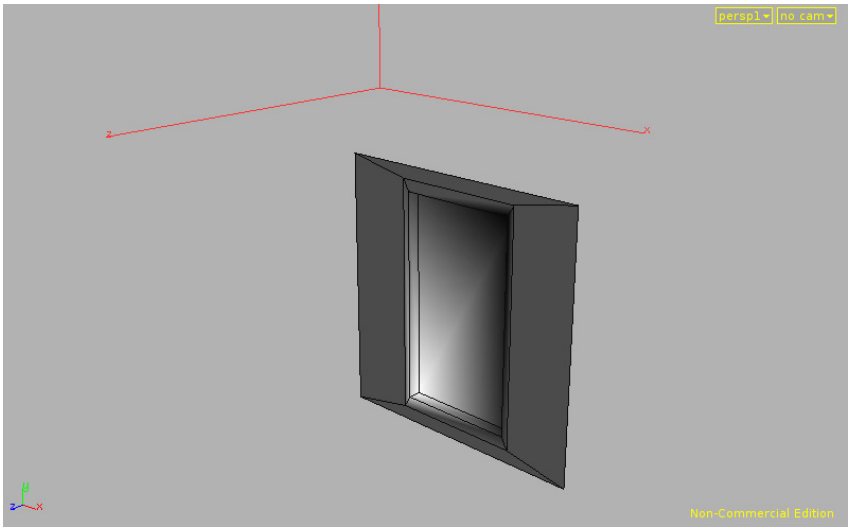


Figura 34: S'aplica al predecessor *door*, la geometria de la porta. Un marc rectangular amb un relleu cap endins.

Aquesta porta s'ha anomenat *doorOk*, però aquest nom, en aquest exemple, no es farà servir per cap altra transformació.

Seguint encara amb el poliedre que representa la part baixa de l'edifici, s'agafa la primera planta de la façana (*ledgeFloor*) i s'hi aplica l'acció *import*, aquesta vegada definint el tipus com a *ledge*. Amb aquesta acció s'obté un sortint o trencaaigües (veure Figura 35).

```
<rule id="7">
      <predecessor name="ledgeFloor"></predecessor>
      <successor>
            <action name="import" >
                  <param name="type" value="ledge"/>

                  <product name="ledge"/>
            </action>
      </successor>
 </rule>
```

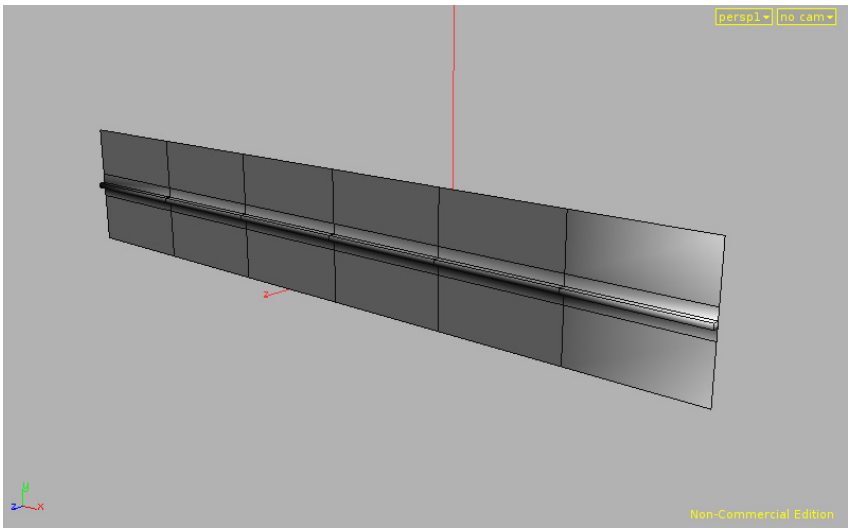


Figura 35: En aquest cas el tipus d'import es *ledge*, per tan es crea una línia amb relleu per a representar un sortint de l'edifici.

En aquest cas és quan el concepte "procedural" associat al Houdini es fa més notori. Quan apliquem l'*import* de tipus *ledge*, el mateix programa ho aplica en totes les divisions de l'objecte de manera automàtica.

Per acabar, s'aplica al volum que representa el cos de l'edifici l'acció *import*, definint el tipus com a *window*. D'aquesta manera s'obté l'edifici amb la façana plena de finestres (veure Figura 36).

```
<rule id="8">
      <predecessor name="b2_t"></predecessor>
      <successor>
            <action name="import">
                  <param name="type" value="window"/>

                  <product name="floors"/>
            </action>
      </successor>
 </rule>
```

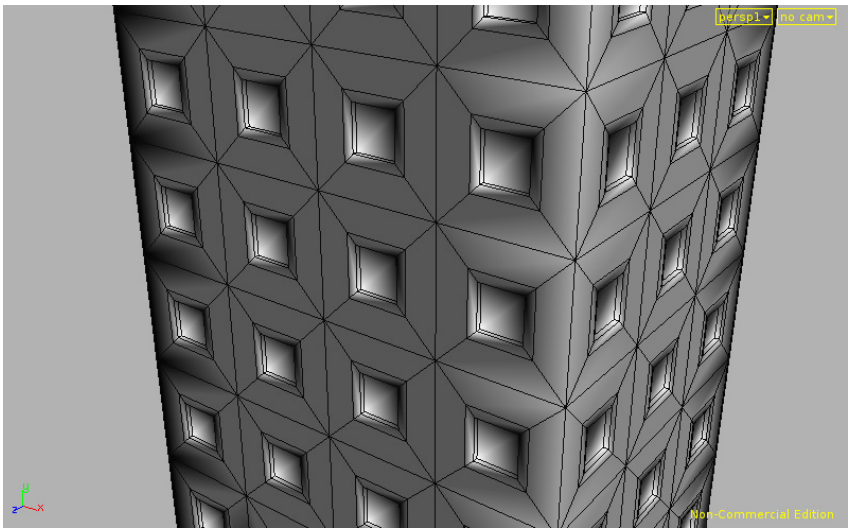


Figura 36: L'import *window* genera les finestres per a tots els seus predecessors. Aquesta vegada es fa servir un marc quadrat amb un petit relleu cap endins.

Finalment, s'obté l'edifici acabat i preparat per a ser exportat en qualsevol format.

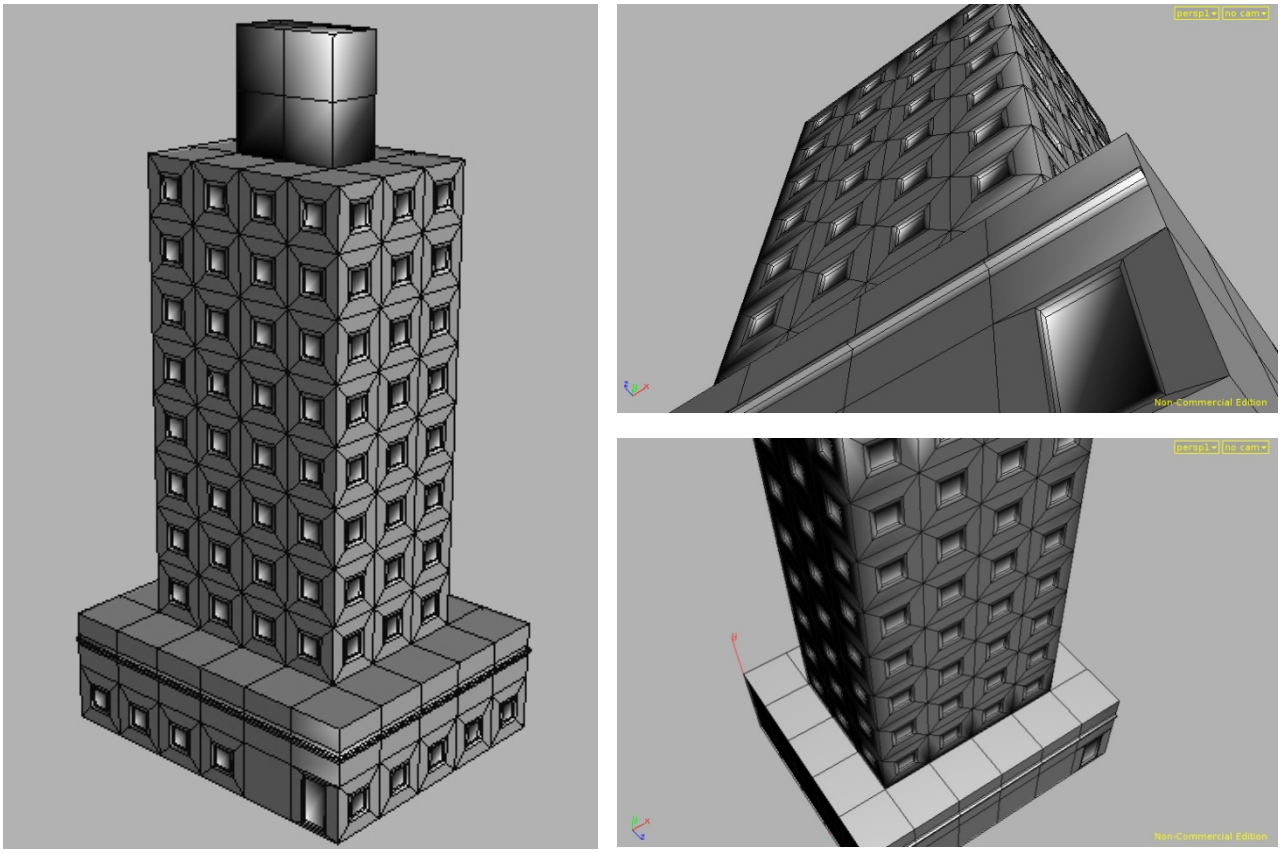


Figura 37: L'edifici acabat vist des de diferents angles.

Capítol 5: Implementació

Dins aquest capítol es mostrarà com s'ha implementat el disseny vist al capítol anterior i com s'ha creat el codi necessari per a interpretar les regles del XMLshape. També es veurà la manera com les tres eines que conformen el projecte (Houdini, Python i XML) s'entrellacen per tal d'aconseguir el resultat desitjat.

5.1 Estructures de dades

Un dels primers problemes que van aparèixer va ser l'emmagatzematge de les dades que s'anaven generant. Per solucionar aquest problema, s'ha optat per emmagatzemar els nodes utilitzant una llista de tuples anomenada *objects*. Cada tupla conté el nom de l'objecte com *string*, que és el que es fa servir al fitxer XML, una referència a l'objecte físic i una tupla anomenada *scope* que conté les mides de l'objecte.

Nom	Referència	Scope
Base	*&base*	[x,y,z]
Window	*&window1*	[x,y,z]
Facade	*&facade*	[x,y,z]
Window	*&window2*	[x,y,z]

Figura 38: Representació de l'estructura de dades que emmagatzema els objectes de l'escena.

Cada vegada que es crea un node nou, es substitueix en els nodes identificats com a *predecessor* de la llista, i s'hi afegeix els nodesidentificats com a *product*. D'aquesta manera s'arriba al final de l'execució amb la llista actualitzada amb els nodes que han de formar part de l'element final. Un cop processades totes les regles, es recorre la llista i es van afegint els nodes a l'entrada d'un *merge* creat especialment per aquesta finalitat.

Cal posar un especial èmfasi a la manera d'anomenar els objectes (Figura 38). Com es pot observar en aquest exemple, existeixen dos objectes amb el *nom* "Window". Els dos noms són iguals, però les referències són diferents. Aquest mètode ens dóna una avantatge significatiu a l'hora d'aplicar les regles als objectes. Per exemple, quan s'aplica la regla *import* de tipus *window*, aquesta es produeix sobre els predecessors anomenats "Window". D'aquesta manera aquesta regla s'aplica sobre totes els objectes anomenats així i per tan sobre tots els elements que han d'acabar sent finestres.

Tot i esborrar el predecessor de la llista *objects*, aquest només s'esborra de l'estructura de dades. Els nodes referenciats es conserven al Houdini.

5.2 Càrrega de l'XML

El primer pas a seguir és llegir el fitxer XML creat per l'usuari i carregar-lo a la memòria del Houdini per tal de poder generar la geometria seguint les regles marcades al fitxer. Per això es fa servir el mòdul *minidom* de Python, del que ja a l'apartat 4.5.2.

Es fan servir les següents sentències:

```
fitxer = open('fitxer.xml')
doc = minidom.parse(fitxer)
```

Un cop executat, la variable *doc* conté un arbre de nodes amb totes les dades del fitxer (Figura 28), com s'ha vist al capítol anterior. A partir d'aquí es van tractant les regles de manera seqüencial, accedint a elles amb els mètodes que proporciona el mateix DOM.

```
per totes les regles{
      identificar i processar regla;
}
```

5.3 Identificar les regles

Primer de tot, cal identificar i obtenir tots els elements que prenen part en cadascuna de les regles, processant totes les accions seqüencialment.

```
predecessor = obtPredecessor(regla);
successor = obtSuccessor(regla);
accions = obtAccions(successor);
per cada accio fer{
      cas(accio.nom){
            opcio "createBase": [...]
            opcio "translate": [...]
            opcio "comp": [...]
            opcio "subdiv": [...]
            opcio "import": [...]
      }
}
```

5.4 Processar les regles

Un cop s'ha identificat una regla i s'han obtingut totes els seus elements, és el moment d'aplicar-la. Per això cal identificar els predecessors dins la taula d'objectes.

Es fa servir la funció *find()* per a trobar els predecessors que ens indica la regla.

```
funció find(list, name){
      seleccionats = crearLlista()
      per tots els elements de list fer{
            si element.nom = name llavors{
                  seleccionats.afegir(element)
            }
      }
      return seleccionats;
}
```

El paràmetre *list* representa una llista d'objectes, que en aquest cas és l'estructura de dades *objects*. El paràmetre *name* és el nom de l'objecte que volem trobar.

Llavors, s'ha d'aplicar cadascuna de les accions als predecessors seleccionats que, depenent de la seva tipologia, seran un o varis. Per exemple, si el predecessor rep el nom de "window", la funció trobarà a l'estructura de dades varis objectes amb aquest nom i, per tant, s'aplicarà l'acció sobre tots aquests objectes.

Les accions que es presenten a continuació són la implementació del disseny presentat a l'apartat 4.2 on es defineixen les regles.

5.4.1 Acció createBase

Aquesta acció crea la base dels edificis. A partir d'un cub de mida 1 unitat, s'obté un poliedre amb les mides definides per l'usuari.

```
createBase(){
      b = crearCub();
      sx, sy, sz = obtMidesXML();
      dx, dy, dz = obtDivisionsXML();
      b.parm('dx').set(dx);
      b.parm('dy').set(dy);
      b.parm('dz').set(dz);
      b.parm('sx').set(sx);
      b.parm('sy').set(sy);
      b.parm('sz').set(sz);
      b.centrar();
```

```
            objects.afegir(b);
        }
```

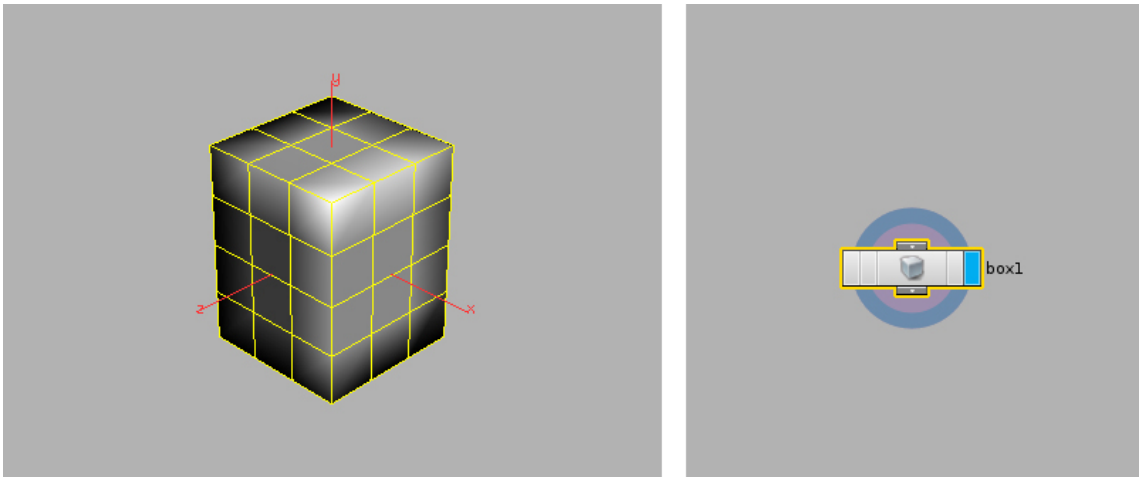


Figura 39: Poliedre creat amb sx=3, sy=4, sz=3.

La Figura 39 ens mostra el resultat del *createBase()*. S'obté un poliedre amb les mides indicades.

5.4.2 Acció translate

Aquesta acció serveix per desplaçar un objecte una distància determinada des del seu punt actual. Les unitats que es fan servir per mesurar les distàncies es basen en la mida de les divisions dels objectes. Per exemple, es pot desplaçar una figura N divisions seguint l'eix de les X.

```
translate(){
      per tots els predecessors fer{
            t = crearTransform();
            tx, ty, tz = obtPosXML();
            t.parm('tx').set(tx);
            t.parm('ty').set(ty);
            t.parm('tz').set(tz);
            t.enllaça(predecessor);
            objects.esborrar(predecessor);
            objects.afegir(t);
      }
}
```

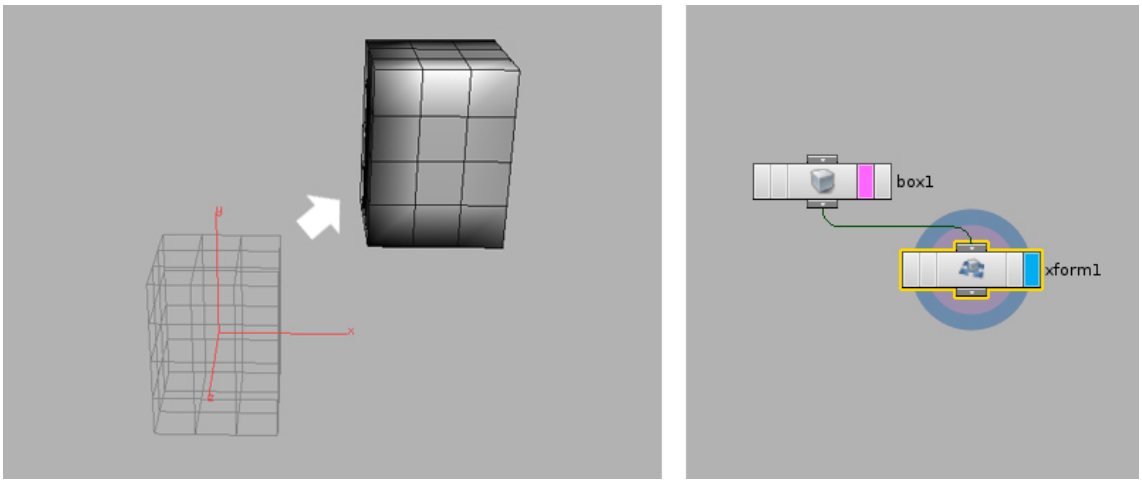


Figura 40: Es crea un node TRANSFORM i s'hi el desplaçament descrit al fitxer XML. S'enllaça la sortida del BOX amb l'entrada del TRANSFORM.

La Figura 40 ens mostra el desplaçament de l'objecte i com queda muntat l'arbre de nodes.

La funció *obtPosXML()* extreu del node actual els paràmetres del desplaçament (veure exemple apartat 4.6). Per tal d'obtenir les divisions i les mides, s'utilitza una funció recursiva que s'executa des del node TRANSFORM que es crea amb l'acció *translate*, fins l'objecte de base, en aquest cas un BOX. Això es fa perquè les mides i divisions només les coneixen els objectes de base.

El següent codi il·lustra la funció recursiva que s'utilitza per a obtenir les mides i les divisions.

```
FUNCIO obtDiviMides()
      SI (node.input != NULL) LLAVORS
            retorna obtDiviMides();
      ALTRAMENT
            dx = node.parm('div1');
            dy = node.parm('div2');
            dz = node.parm('div3');
            sx = node.parm('sx');
            sy = node.parm('sy');
            sz = node.parm('sz');
            retorna dx, dy, dz, sx, sy, sz;
      FSI
FFUNCIO
```

5.4.3 Acció comp

Com s'ha dit a la explicació sobre Hodini, a l'apartat 2.1, aquest programa enumera les divisions d'un objecte seguint sempre un mateix patró. D'aquesta manera es pot aconseguir una fórmula per a determinar quines divisions formen part de cada cara. Les dues funcions *calcularInici* i *calcularFinal* retornen respectivament un enter que determina la divisió d'inici i la de final de cadascuna de les cares. Un cop calculats *inici* i *final*, es crea un node DELETE per cadascuna de les cares resultants del poliedre.

```
comp(){
      per tots els predecessors fer{
            per i=1 fins 6 fer{
                  inici = calcularInici();
                  final = calcularFinal();
                  d = crearDelete()
                  d.parm('iniciDel').set(inici);
                  d.parm('finalDel').set(final);
                  d.enllaça(predecessor)
                  objects.afegir(d);
            }
            objects.esborra(predecessor)
      }
}
```

La Figura 41 il·lustra el resultat d'aplicar una acció *comp* a un BOX. El resultat però, no és apreciable visualment perquè les cares del poliedre, tot i estar separades, segueixen posicionades al mateix lloc i dóna la sensació que no s'ha aplicat cap acció. La Figura 42 mostra que les cares estan vertaderament separades ja que es poden moure individualment.

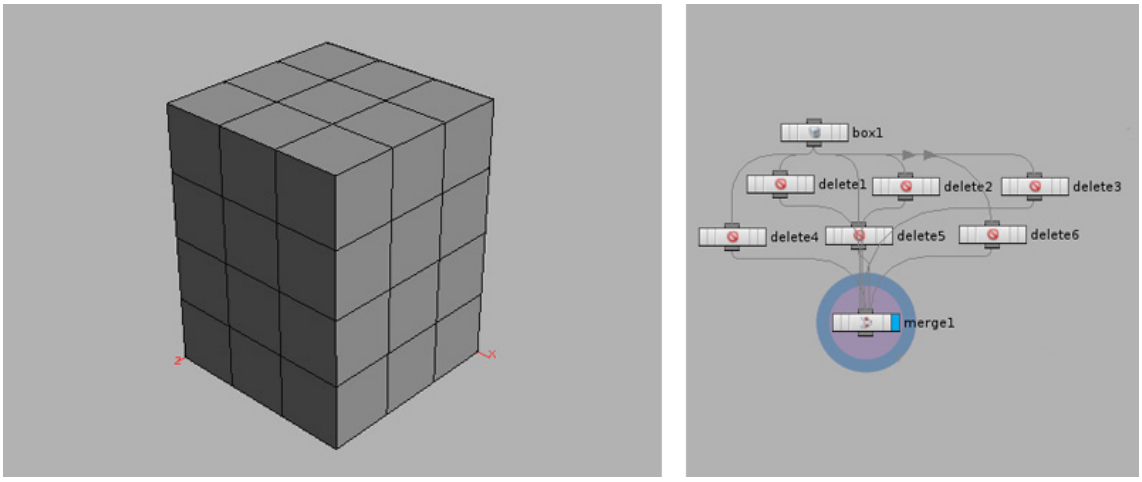


Figura 41: L'objecte ha estat separat amb totes les seves components.

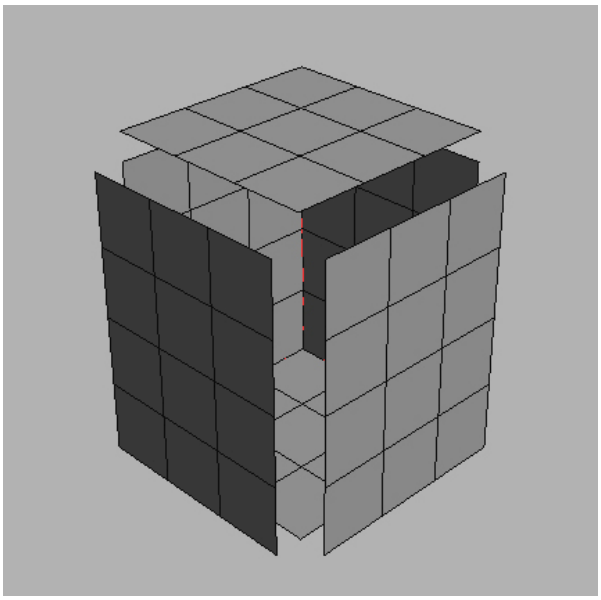


Figura 42: Resultat modificat per a poder-ho apreciar millor

Els mètodes *calcuarInici()* i *calcularFinal()*, fan servir una fòrmula matemàtica per a calcular els dos valors. Tenint en compte el sistema d'enumeració de primitives que fa servir el Houdini (veure apartat 2.1.2.1) s'ha desenvolupat aquesta fòrmula:

$$inici = r(x * y) + s(x * z) + t(y * z) \quad \rightarrow \quad per\ n - 1$$

$$final = r(x * y) + s(x * z) + t(y * z) - 1 \quad \rightarrow \quad per\ n$$

on

- **x, y, z** són les mides de la cara.
- **n** representa la cara que s'està processant.
- **r, s, t** segueixen la següent taula:

n	***r***	***s***	***t***
1	1	0	0
2	2	0	0
3	2	1	0
4	2	2	0
5	2	2	1
6	2	2	2

Aquesta taula s'obté seguint la manera d'enumerar les cares que utilitza el Houdini (veure Figura 7). Per exemple, quan s'està processant la cara 5, ja es porten acumulades les cares 1 i 2 (r) i les cares 3 i 4 (s). Per tant la funció queda:

$$inici = 2(x * y) + 2(x * z) + x * y$$

5.4.4 Acció subdiv

Seguint el mateix sistema basat en la numeració de les divisions que s'ha vist a l'acció *comp*, amb l'acció subdiv, es separen files o columnes d'una mateixa cara (veure Figura 43).

```
subdiv(){
	divisions = llista[]
	per tots els predecessors fer{
		si eix = 'Y' llavors{
			inici = calcularIniciY();
			final = calcularFinalY();
		}altrasi eix = 'X' llavors{
			inici = calcularIniciX();
			final = calcularFinalX();
		}
		d = crearDelete();
		d.parm('inici').set(inici);
		d.parm('final').set(final);
		d.enllaça(predecessor);
		objects.afegir(d);
		objects.esborrar(predecessor);
	}
}
```

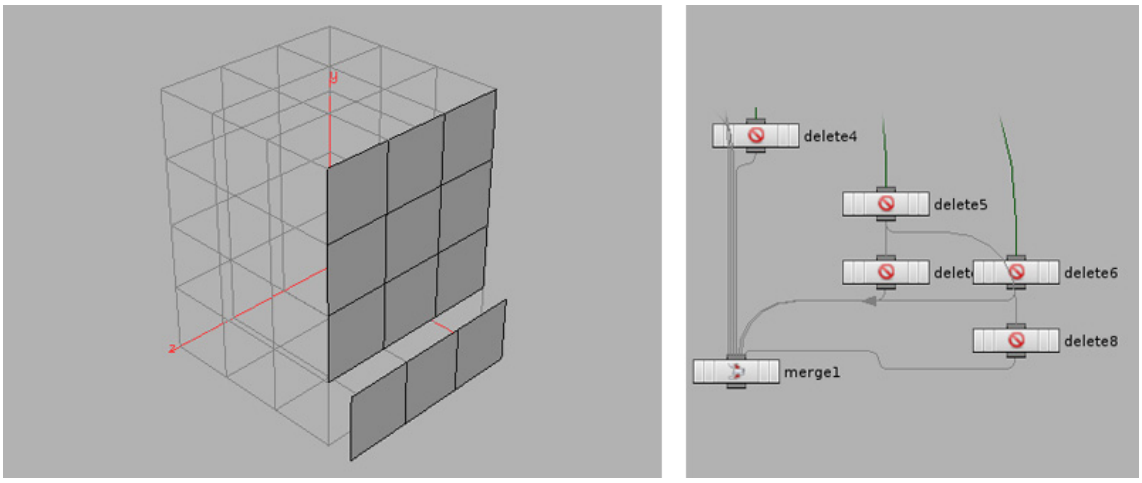


Figura 43: En aquest cas, la cara seleccionada és dividida en dos: la planta baixa (que s'ha desplaçat per mostrar-ho millor) i les plantes superiors.

Els mètodes *calcularInici()* i *calularFinal()* retornen les primitives d'inici i final de les divisions que fa el *subdiv*.

Càlcul a l'eix de les Y (per files):

$$inici = n * x$$

$$final = \big((m + 1) * x\big) - 1$$

on

- **n** és l'identificador de la columna actual. A la primera iteració es 0 i a les següents pren el valor de m+1.
- **m** és un valor que s'obté sumant *n* més el nombre de files que es volen agafar, menys 1. (m = (n + nFiles) -1).
- **x** i **y** són les mides de la cara que s'està dividint.

Càlcul a l'eix de les X (per columnes):

$$inici = n$$

$$final = y * x - 1$$

on

- **n** és l'identificador de la columna actual. A la primera iteració és 0 i a les següents pren el valor de *n* més el nombre de columnes que es volen agafar. (n = n + nFiles).
- **x** i **y** són les mides de la cara que s'està dividint.

A continuació es mostra una representació gràfica de la numeració de les files i columnes:

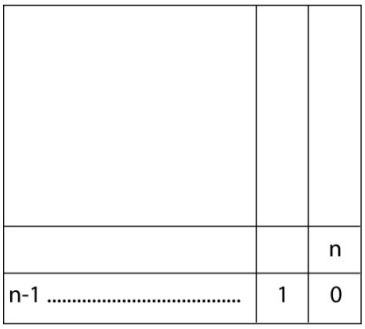


Figura 44: Representació de com Houdini enumera les primitives d'una paret.

Per exemple, en una paret de 5x4, si es vol agafar les fila 1, la fórmula quedaria:

$$inici = 0 * 5 = 0$$

$$final = \big((0 + 1) * 5\big) - 1 = 4$$

Per tan, s'agafaran les primitives enumerades del 0 al 4, o sigui, la primera fila de la paret.

5.4.5 Acció import

Aquest acció serveix per a generar la geometria dels detalls de l'edifici com portes, finestres i altres elements arquitectònics.

```
import(){
      per tots els predecessors fer{
            tipus = obtTipusXML();
            cas tipus{
                  opció "window":
                        nou = crearFinestra(predecessor);
                  opció "ledge":
                        nou = crearSortint(predecessor);
                  opció "door":
                        nou = crearPorta(predecessor);
            }
            nou.enllaça(predecessor);
            objects.esborrar(predecessor);
            objects.afegir(nou);
      }
}
```

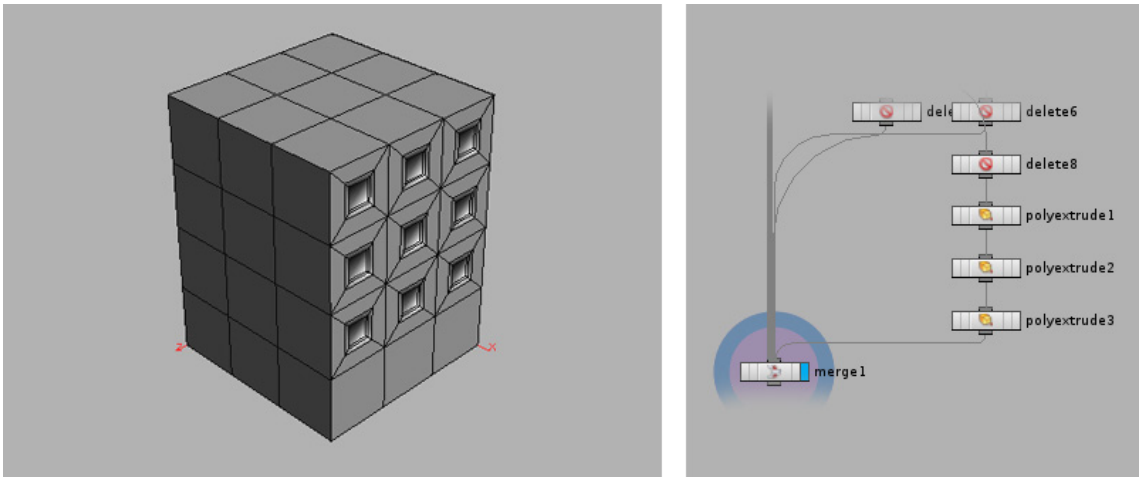


Figura 45: Es pot veure que només s'ha aplicat l'*import* "window" sobre el tros de façana que s'ha separat a la Figura 43

Els mètodes *crearFinestra()*, *crearSortint()* i *crearPorta()* generen la geometria de cada element. En aquest projecte, es crea a partir d'extrusions dels polígons de cada divisió. Tot i això, el sistema està preparat i pensat per a poder fer un import de qualsevol geometria. Si es vol, es pot crear per separat una finestra o una porta, utilitzant la interfície gràfica i després importar-la des del fitxer XML.

El mètode *crearSortint()* és segurament el més interessant dels tres. Per a crear el relleu del *ledge* es poden triar diferents opcions de creació (veure apartat 4.2). Segurament, la part més

complicada d'aquesta funció ha sigut fabricar només el sortint, eliminant el tros de paret que té a darrera (flag *noWall=True*). Per implementar aquesta funcionalitat s'ha optat pel mètode emprat durant tot el projecte: utilitzar nodes DELETE per a eliminar la geometria que no interessa. El problema, en aquest cas, torna a ser la identificació de les cares que es vol eliminar. Quan es genera l'extrusió de geometria aquesta funció crea diferents primitives, totes elles, com sempre, enumerades i referenciables. La Figura 46 mostra quin patró segueix el Houdini per a enumerar les cares.

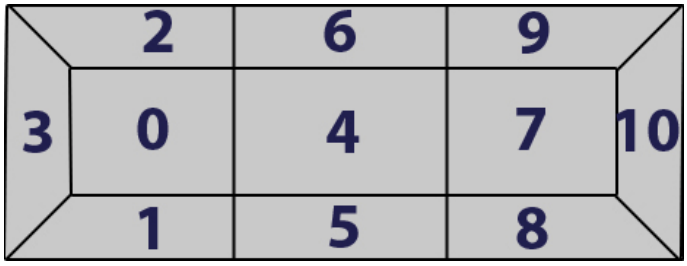


Figura 46: Enumeració de les primitives després d'aplicar un POLYEXTRUDE

En aquest exemple cal seleccionar les primitives 0, 4 i 7. Per això s'ha creat una funció que identifica la posició de les primitives i les selecciona automàticament. S'ha de tenir en compte quina és la posició de cadascuna dins la geometria. La funció que selecciona les primitives funciona calculant l'identificador de cada cara que es vol obtenir. Per exemple, en aquest cas el càlcul quedaria així:

0

0 + 4 = 4

4 + 3 = 7

Com es pot veure, sempre es suma 4 en el cas que la primitiva estigui situada a un dels dos laterals, ja que hi ha una cara més per sumar (cares 3 i 10). Quan la primitiva es troba entremig, es suma 3 per a obtenir la següent primitiva vàlida. Així doncs la funció que tria les primitives segueix aquest esquema, que es va repetint dins un bucle per totes les cares del poliedre:

```
SI x = 1 LLAVORS
      valor_a_sumar: = 5;
ALTRAMENT
      SI i = 1 LLAVORS
            valor_a_sumar := 4;
            i++;
      ALTRASI i=x LLAVORS
            valor_a_sumar := 4;
            i:=1;
      ALTRAMENT
            valor_a_sumar :=3;
```

```
                i++;
            FSI
        FSI
```

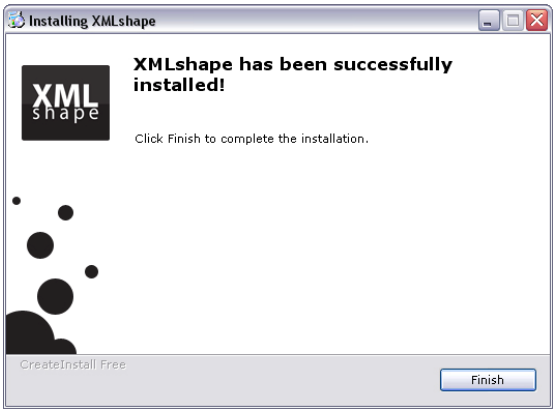
on

x és la mida en l'eix de les X del polígon.

i és l'identificador de la cara que s'està tractant.

5.5 Instal·lació i execució

Per a poder fer funcionar el programa cal, primer de tot, tenir el Houdini 9.0 o superior instal·lat. Després cal afegir-hi el mòdul XMLshape. Per això només cal obrir l'executable *XMLshapeSetup.exe* que ja s'encarregarà d'instal·lar-lo a la carpeta d'scripts de Python que té el Houdini. Una qüestió molt important és no tocar la ruta d'instal·lació que ja proporciona el programa i en el cas de fer-ho, assegurar-se que la ruta fins a la carpeta "HOUDINI" és correcte.



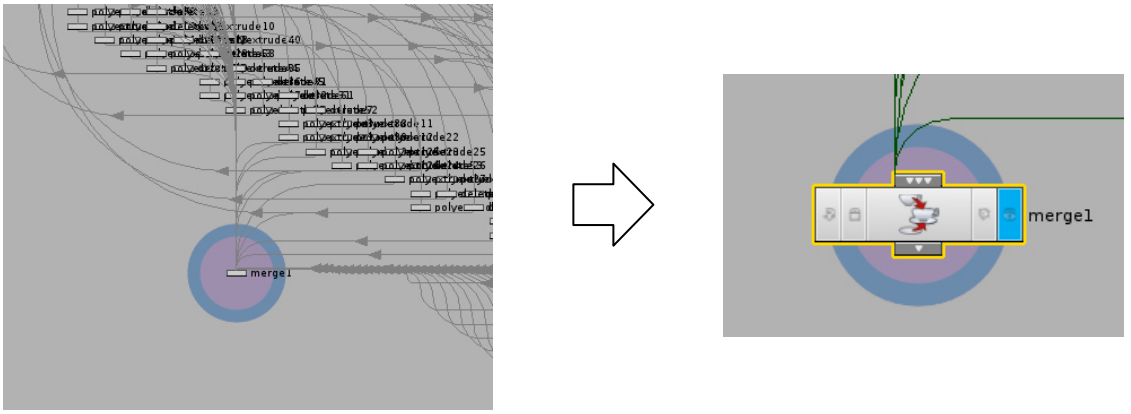
Un cop finalitzat l'assistent d'instal·lació cal engegar el Houdini, obrir la consola de Python (Windows > Python Shell) i escriure les següent sentència:

```
import xmlshape
```

Per a executar el mòdul cal escriure:

```
xmlshape.create('ruta_del_fitxer_xml')
```

Si tot és correcte ens apareixerà la figura a l'àrea de treball. Per a visualitzar l'edifici complet només cal navegar per l'arbre de nodes i seleccionar el node *merge* que uneix tot l'arbre en mode "visible".



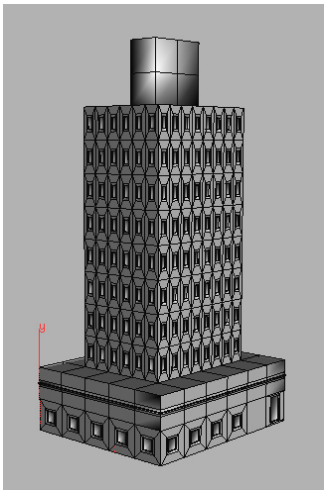
Capítol 6: Resultats

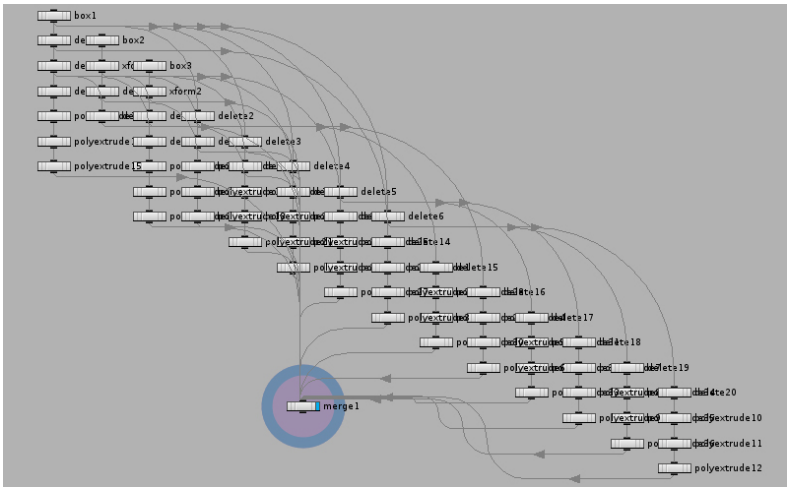
Dins d'aquest capítol es mostren els resultats obtinguts del projecte. Els resultats estan visualitzats directament des de l'àrea de treball de Houdini, sense ser renderitzats per els motors del programa ja que el que s'ha buscat amb la realització del projecte ha sigut la generació de la geometria, no pas el resultat visual "acurat" que dóna un motor de render.

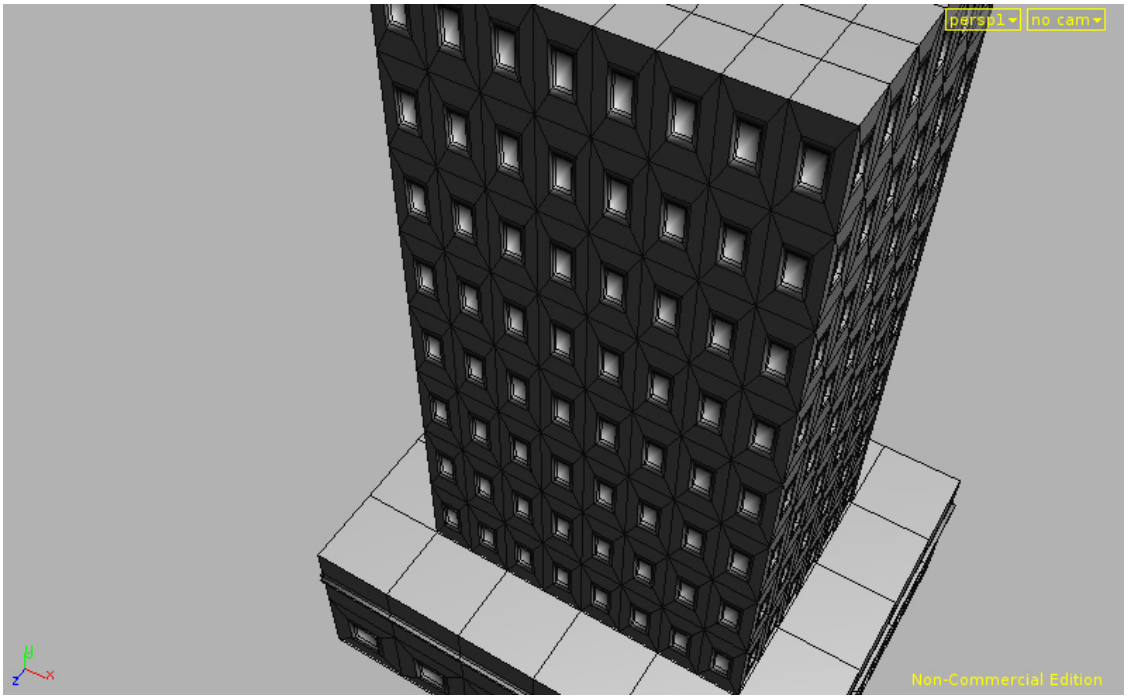
6.1 Resultats

6.1.1 Edifici d'oficines

Aquest edifici està compost per tres mòduls posats l'un sobre l'altre a mode de planta baixa, pisos i terrat. Aquest exemple és una variant de l'edifici que s'ha presentat a l'apartat 4.6. En aquest exemple s'ha mostrat com queda l'arbre de nodes després de l'execució de l'script. (veure annexos, apartat 11.2).

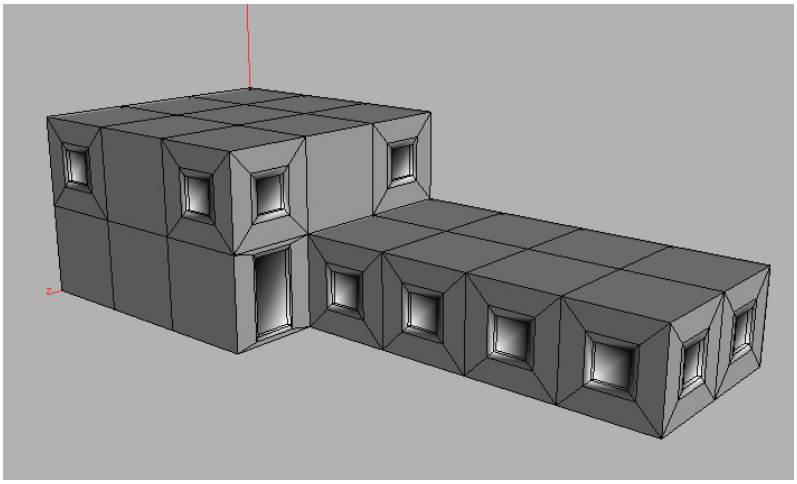


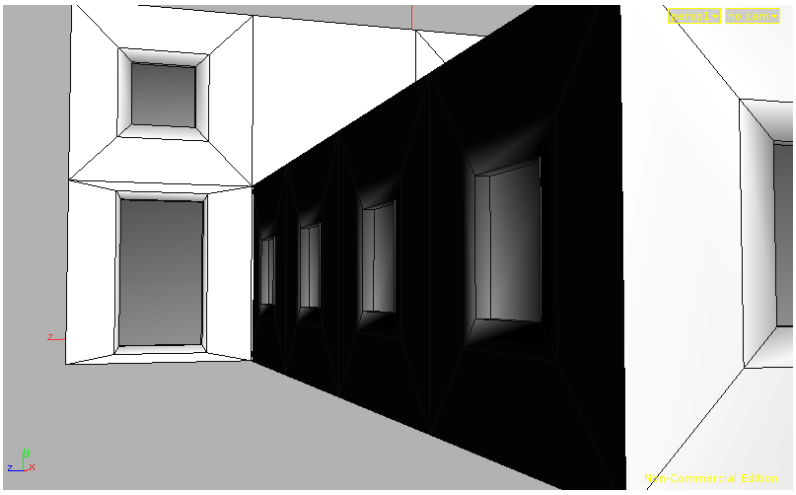


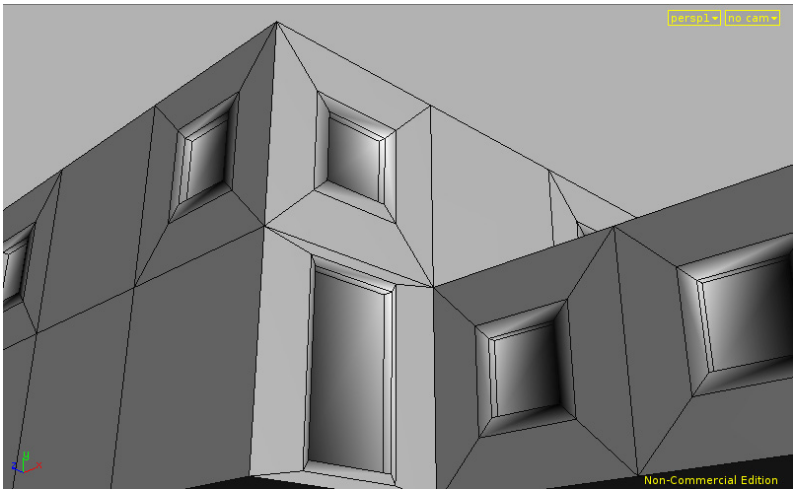


6.1.2 Casa unifamiliar

En aquest cas, s'ha creat una casa unifamiliar a partir de dos bloc bàsics, un de dues plantes i un amb planta baixa. Només han calgut 10 regles per a generar tota la geometria que es pot veure a continuació. (veure annexos, apartat 11.1).

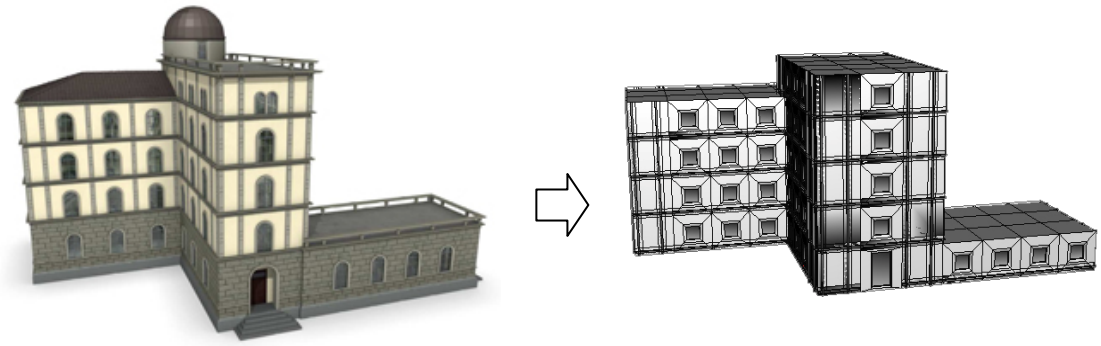


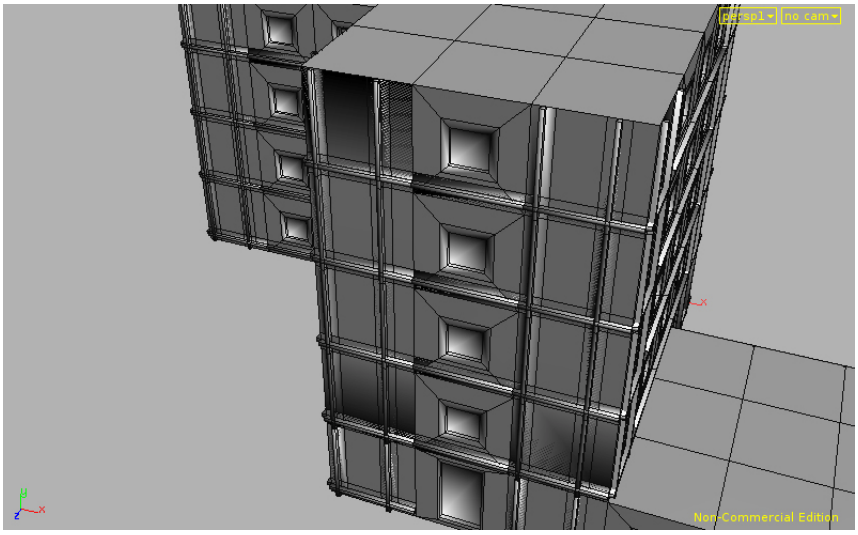


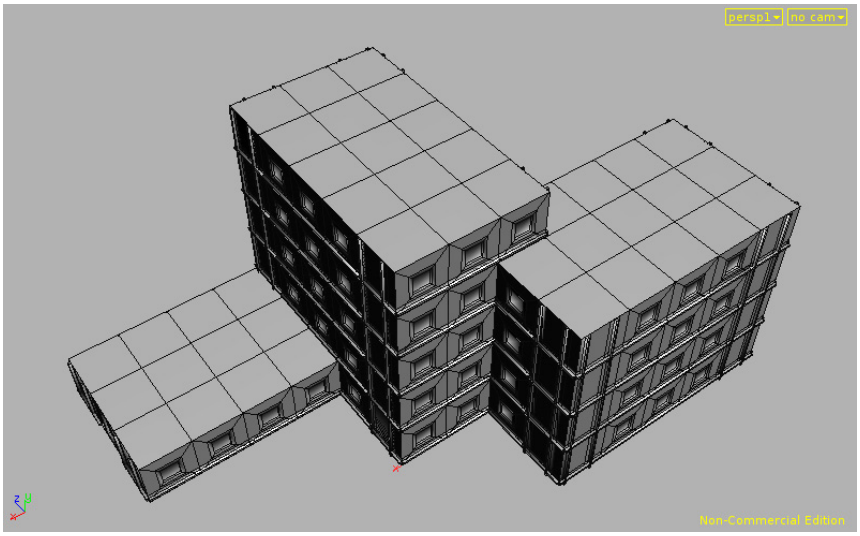


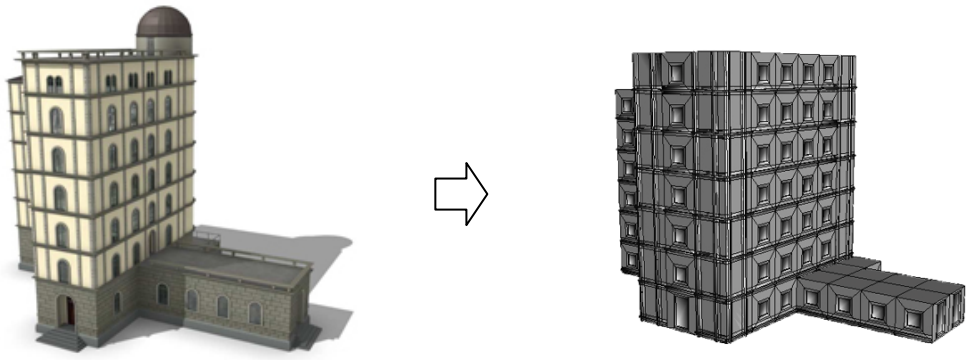
6.1.3 Recreant edificis

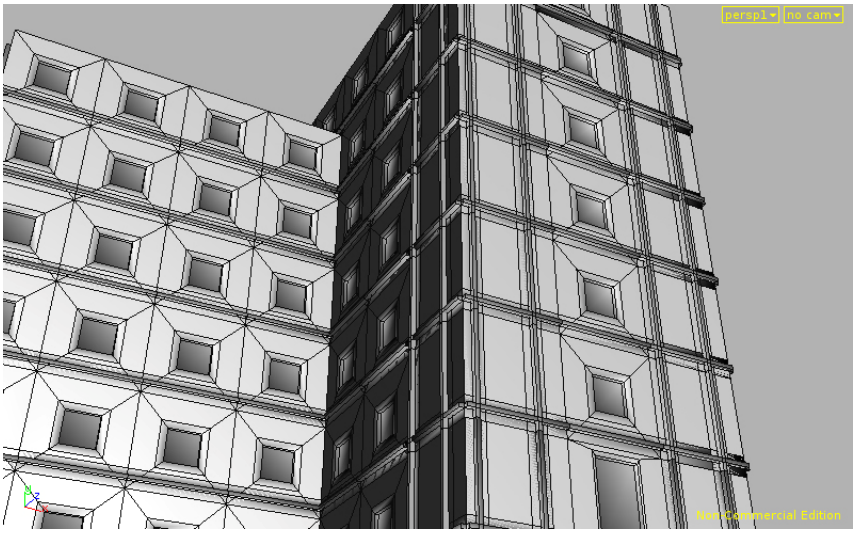
Per tal de provar l'eficàcia del programa, s'ha intentat reproduir amb el màxim de fidelitat possible els quatre edificis que es poden trobar a l'article "Procedural Modeling of Buildings" [Müller '06].

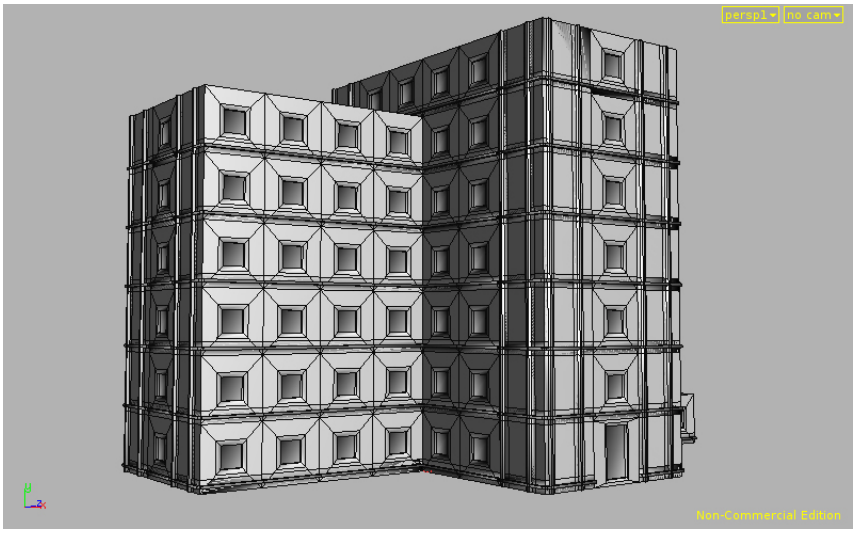


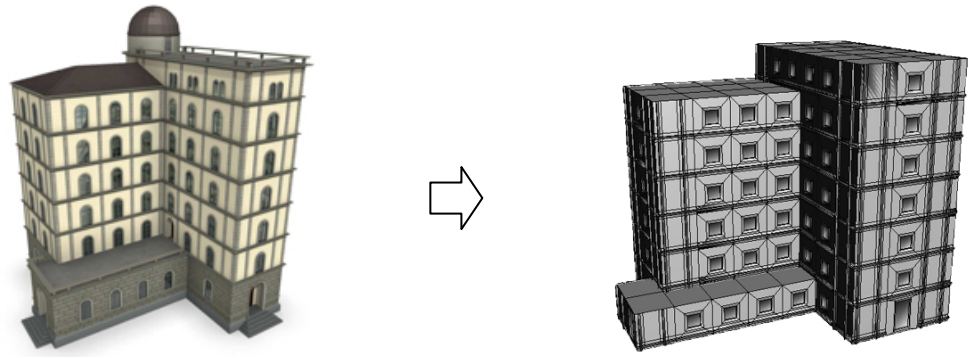




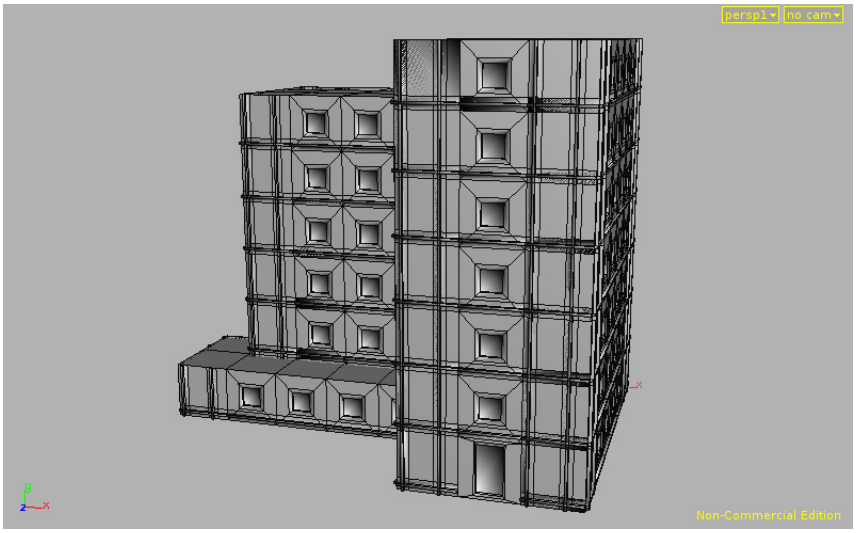


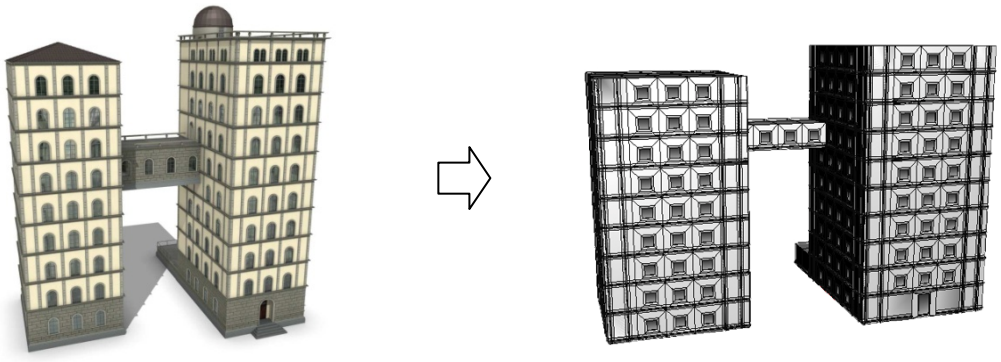

persp
no cam
Non-Commercial Edition


persp
no cam
Non-Commercial Edition

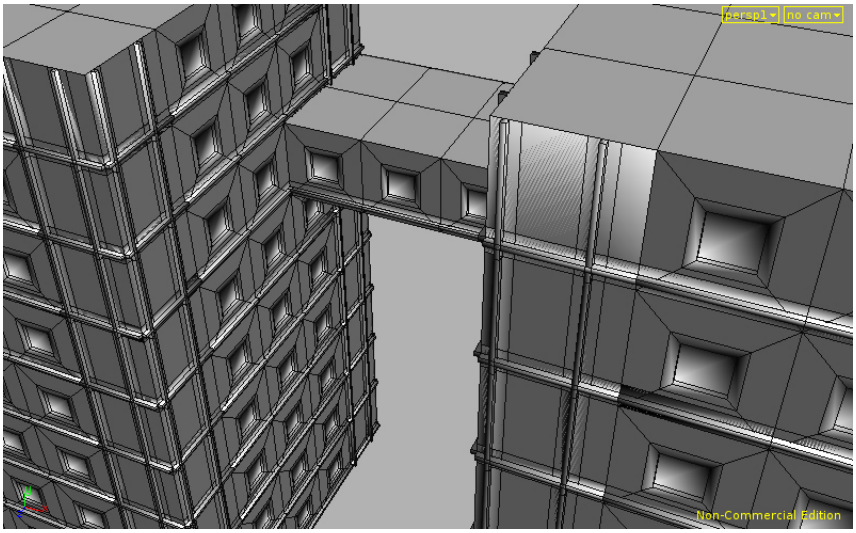


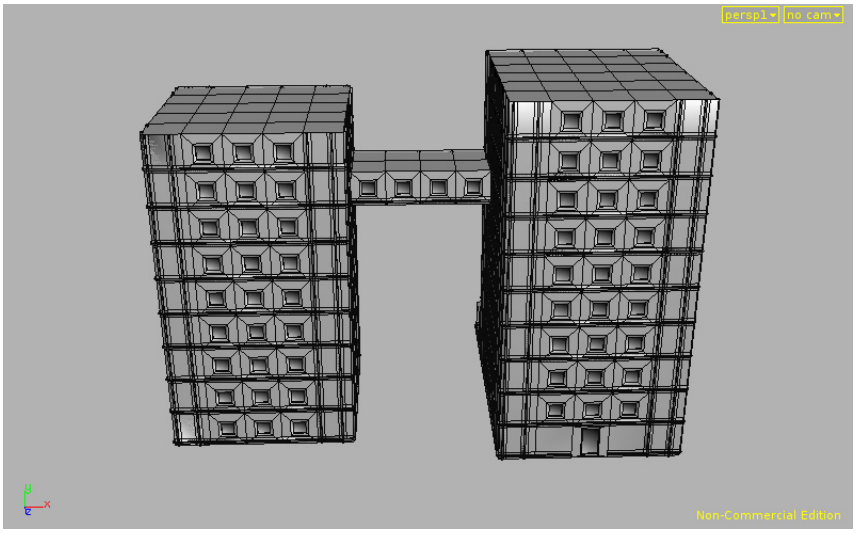

persp
no cam
Non-Commercial Edition


persp
no cam
Non-Commercial Edition



(veure annexos, apartat 11.3)





Capítol 7: Conclusions

Un cop enllestida tota la feina és moment de valorar el treball realitzat durant aquests mesos.

7.1 Conclusions

Un cop valorats els resultats obtinguts cal estudiar si s'ha aconseguit allò que es pretenia a l'inici. L'objectiu bàsic d'aquest projecte era el desenvolupament d'una eina de fàcil ús que permetés a l'usuari obtenir de manera ràpida l'estructura bàsica d'un edifici. A priori l'objectiu s'ha complert, tot i que encara queda molt per a afegir i retocar, com a tots els projectes. En un principi, el camí a seguir era el marcat per Peter Wonka i Pascal Müller als seus articles. S'ha aconseguit un sistema per a crear edificis bastant semblant al de Müller i Wonka. La generació de la geometria bàsica ha quedat bastant completa i el fet que, un cop executat l'algorisme, es pugui seguir manipulant l'objecte amb el Houdini fa que sigui una bona eina a l'hora de crear estructures arquitectòniques de manera ràpida i senzilla. S'ha aconseguit crear un conjunt de regles de fàcil comprensió, però a la vegada molt potents a l'hora de crear geometria. Els sistema tan és vàlid per un edifici senzill, d'un sol bloc, sense molts detalls, com per edificis de gran complexitat com poden ser, blocs d'oficines, amb moltes finestres i complements. També s'ha aconseguit crear un model de casa unifamiliar fent els blocs més baixos i senzills. En definitiva, s'ha creat una eina polivalent i útil que avarca multitud de solucions geomètriques.

7.2 Conclusions personals

D'entrada, el plantejament d'un projecte d'aquesta mida és quelcom que espanta fàcilment. Durant la carrera l'alumne està acostumat a fer pràctiques i treballs contínuament però no tenen res a veure amb el projecte de final de carrera. Aquest projecte és l'acabament d'un procés d'aprenentatge i és on, a priori, s'ha de demostrar tot allò que s'ha après durant els anys de classe a la universitat. Per això i sobretot perquè un està fent allò que ha triat i que li agrada, és normal passar hores i hores davant de l'ordinador cercant el millor resultat.

És indispensable veure tot el desenvolupament del projecte i seguir les diferents fases que s'han anat superant fins a arribar a l'objectiu final per a comprendre la dificultat de crear un sistema nou, sense casi documentació, i la il·lusió que s'ha posat en tot el procés. Cal dir doncs, que no ha estat una empresa fàcil, però tots els entrebancs queden amagats darrera de la satisfacció d'haver fet una feina ben feta.

Capítol 8: Treball Futur

Dins aquest capítol es repassarà breument les idees que han anat sorgint mentre es desenvolupava el projecte i que per una raó o altra no s'han implementat. Les idees que es mostraran són conceptes que es poden implementar en un futur per tal de dotar al sistema de més eines per a crear edificis més realistes.

8.1 Implementació de laterals arrodonits

Una dels temes que queda per tractar és la implementació de figures arrodonides. Seria bo poder afegir cilindres i esferes a l'eina per tal de poder construir torres, columnes, cúpules, etc. Aquesta feina però, no és gens trivial. El fet que el sistema es basi en divisions enteres fa que s'hagi de crear un sistema de mesura especial per les figures cilíndriques i esfèriques, que no disposen de les mides x,y,z. Tot i això, la dificultat més gran estaria al moment d'afegir elements com portes i finestres a una figura. Caldria convertir la geometria en una xarxa poligonal per tal de seguir el mateix sistema implementat per a les altres figures. A la Figura 47 és pot veure com quedaria una torre en un dels edificis creats.

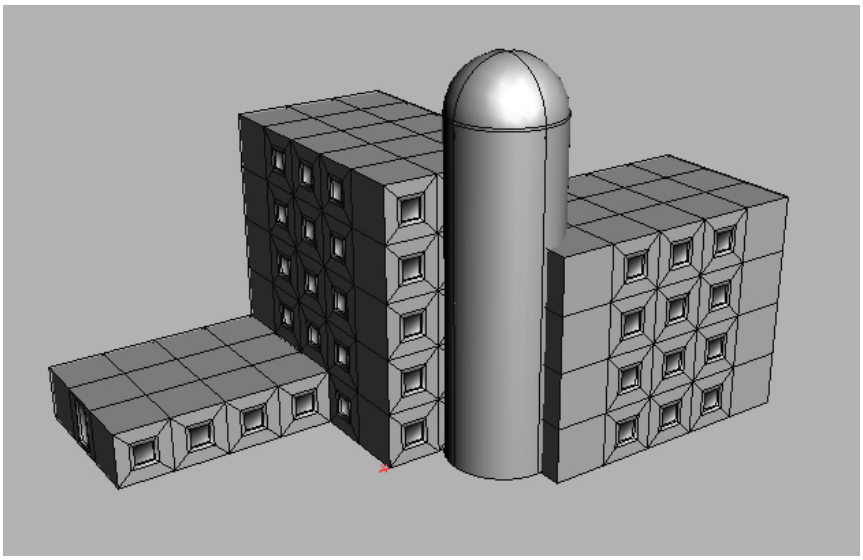


Figura 47: Torre afegida manualment a través del Houdini.

8.2 Aplicació de textures

L'aplicació de textures als edificis és una opció que representaria, segurament, el tema principal d'un altre projecte de final de carrera. La complexitat de les textures ja siguin planes o amb relleu, i la gran varietat d'opcions que ofereix el Houdini per a aplicar-les fa que sigui un feina no gens fàcil.

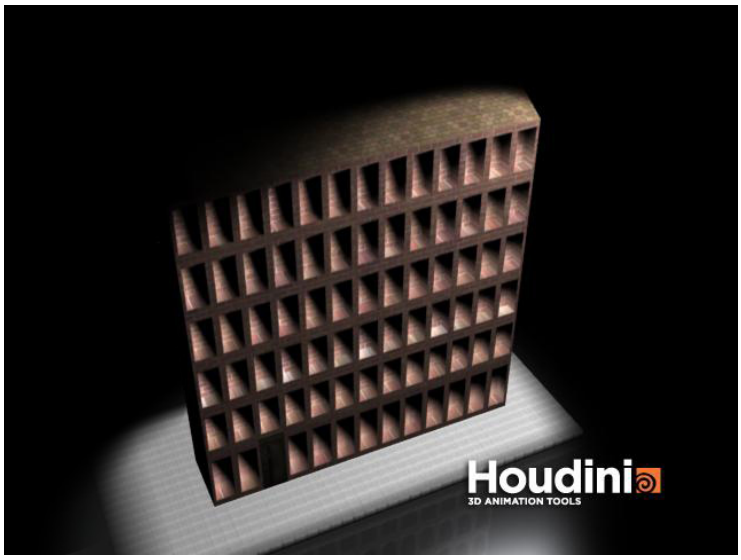


Figura 48: Demo d'un edifici creat amb XMLshape i posteriorment tractat manualment amb el Houdini. S'han afegit les textures de paret i teulada. S'ha fet servir el motor de render per a obtenir una millor visualització de l'escena.

8.3 Generació de teulades

Aquest és un altra aspecte que seria bo plantejar com a treball futur. Sobretot per a dissenyar cases és essencial disposar d'un mètode que permeti crear primitives amb una certa inclinació per a simular teulades.

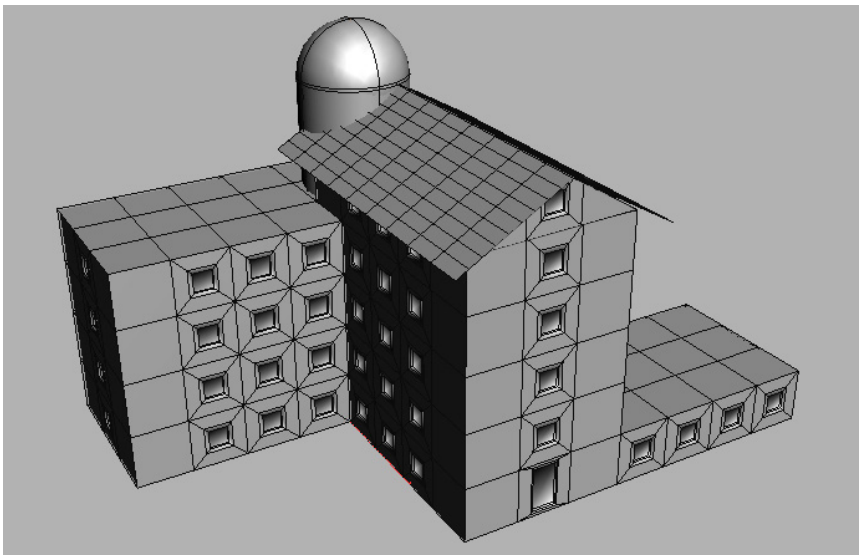


Figura 49: Edifici al qual se li ha afegit una teulada manualment utilitzant el Houdini.

Capítol 9: Agraïments

Molt especialment al meu tutor Gustavo Patow, per dedicació i il·lusió desenfrenada que ha posat en aquest projecte.

A la meva família, pel seu suport durant tots aquests mesos.

I a tots aquells que, en més o menys mesura, han patit i compartit les meves desesperacions i alegries tot aquest temps.

Capítol 10: Bibliografia

Referència	Títol complet
[Müller '06]	Pascal Müller, Peter Wonka Simon Haegler, Andreas Ulmer i Luc Van Gool. *Procedural Modeling of Buildings*. ACM SIGGRAPH (2006).
[W3C]	*World Wide Web Consortium*. 1999. W3C. http://www.w3c.com
[L-S]	Prusinkiewicz i Lindenmayer. *Sistemes de Lindenmayer*. (1991).
[LC]	Le Corbusier. *Towards a New Architecture*. Dover Publications. (1985).
[SCH]	Shmitt, G. *Architectura et machina*. Vieweg & Sohn.(1993).

Capítol 11: Annexos

11.1 Codi XML de la casa unifamiliar

```
<XMLshape>
	<rule id="1">
		<predecessor name="base"></predecessor>
		<successor>
			<action name="createBase">
				<param name="type" value="box" />
				<param name="sx" value="6" />
				<param name="sy" value="2" />
				<param name="sz" value="5" />

				<product name="b1" />
			</action>
			<action name="createBase">
				<param name="type" value="box" />
				<param name="sx" value="4" />
				<param name="sy" value="8" />
				<param name="sz" value="3" />
				<param name="dx" value="8" />
				<param name="dy" value="8" />
				<param name="dz" value="6" />

				<product name="b2" />
			</action>
			<action name="createBase">
				<param name="type" value="box" />
				<param name="sx" value="2" />
				<param name="sy" value="2" />
				<param name="sz" value="1" />

				<product name="b3" />
			</action>
		</successor>
	</rule>
	<rule id="2">
		<predecessor name="b2"></predecessor>
		<successor>
			<action name="translate">
				<param name="tx" value="2"/>
				<param name="ty" value="2"/>
				<param name="tz" value="2"/>

				<product name="b2_t"/>
			</action>
		 </successor>
	</rule>
	<rule id="3">
		<predecessor name="b3"></predecessor>
		<successor>
			<action name="translate">
				<param name="tx" value="2"/>
				<param name="ty" value="10"/>
				<param name="tz" value="2"/>

				<product name="b3_t"/>
			</action>
		 </successor>
```

```
</rule>
<rule id="4">
	<predecessor name="b1"></predecessor>
	<successor>
		<action name="comp">

			<product name="facade"/>
			<product name="sidewing"/>
			<product name="roof"/>
			<product name="floor"/>
			<product name="sidewing"/>
			<product name="sidewing"/>
		</action>
	</successor>
</rule>
<rule id="4">
	<predecessor name="b2_t"></predecessor>
	<successor>
		<action name="comp">

			<product name="windows"/>
			<product name="windows"/>
			<product name="roof"/>
			<product name="floor"/>
			<product name="windows"/>
			<product name="windows"/>
		</action>
	</successor>
</rule>
<rule id="5">
	<predecessor name="facade"></predecessor>
	<successor>
		<action name="subdiv">
			<param name="axis" value="Y"/>
			<param name="div1" value="1"/>
			<param name="div2" value="1"/>

			<product name="groundFloor"/>
			<product name="ledgeFloor"/>
		</action>
	</successor>
</rule>
<rule id="6">
	<predecessor name="sidewing"></predecessor>
	<successor>
		<action name="subdiv">
			<param name="axis" value="Y"/>
			<param name="div1" value="1"/>
			<param name="div2" value="1"/>

			<product name="windows"/>
			<product name="ledgeFloor"/>
		</action>
	</successor>
</rule>
<rule id="7">
	<predecessor name="ledgeFloor"></predecessor>
	<successor>
		<action name="import">
			<param name="type" value="ledge"/>

			<product name="ledge"/>
		</action>
	</successor>
</rule>
<rule id="8">
	<predecessor name="groundFloor"></predecessor>
```

```
            <successor>
                  <action name="subdiv">
                        <param name="axis" value="X"/>
                        <param name="div1" value="1"/>
                        <param name="div2" value="1"/>
                        <param name="div2" value="4"/>

                        <product name="door"/>
                        <product name="door2"/>
                        <product name="windows"/>
                  </action>
            </successor>
      </rule>
       <rule id="9">
            <predecessor name="b2_t"></predecessor>
            <successor>
                  <action name="import">
                        <param name="type" value="window"/>

                        <product name="floors"/>
                  </action>
            </successor>
       </rule>
       <rule id="10">
            <predecessor name="door"></predecessor>
            <successor>
                  <action name="import">
                        <param name="type" value="door"/>

                        <product name="doorOk"/>
                  </action>
            </successor>
       </rule>
       <rule id="11">
            <predecessor name="windows"></predecessor>
            <successor>
                  <action name="import">
                        <param name="type" value="window"/>

                        <product name="floors"/>
                  </action>
            </successor>
       </rule>
</XMLshape>
```

11.2 Codi XML per l'edifici de negocis

```
<XMLshape>
	<rule id="1">
		<predecessor name="base"></predecessor>
		<successor>
			<action name="createBase">
				<param name="type" value="box" />
				<param name="sx" value="6" />
				<param name="sy" value="2" />
				<param name="sz" value="5" />

				<product name="b1" />
			</action>
			<action name="createBase">
				<param name="type" value="box" />
				<param name="sx" value="4" />
				<param name="sy" value="8" />
				<param name="sz" value="3" />
				<param name="dx" value="8" />
				<param name="dy" value="8" />
				<param name="dz" value="6" />

				<product name="b2" />
			</action>
			<action name="createBase">
				<param name="type" value="box" />
				<param name="sx" value="2" />
				<param name="sy" value="2" />
				<param name="sz" value="1" />

				<product name="b3" />
			</action>
			</successor>
	</rule>
	<rule id="2">
		<predecessor name="b2"></predecessor>
		<successor>
			<action name="translate">
				<param name="tx" value="2"/>
				<param name="ty" value="2"/>
				<param name="tz" value="2"/>

				<product name="b2_t"/>
			</action>
		</successor>
	</rule>
	<rule id="3">
		<predecessor name="b3"></predecessor>
		<successor>
			<action name="translate">
				<param name="tx" value="2"/>
				<param name="ty" value="10"/>
				<param name="tz" value="2"/>

				<product name="b3_t"/>
			</action>
		</successor>
	</rule>
	<rule id="4">
		<predecessor name="b1"></predecessor>
		<successor>
			<action name="comp">

				<product name="facade"/>
```

```
                        <product name="sidewing"/>
                        <product name="roof"/>
                        <product name="floor"/>
                        <product name="sidewing"/>
                        <product name="sidewing"/>
                </action>
        </successor>
</rule>
<rule id="4">
        <predecessor name="b2_t"></predecessor>
        <successor>
                <action name="comp">

                        <product name="windows"/>
                        <product name="windows"/>
                        <product name="roof"/>
                        <product name="floor"/>
                        <product name="windows"/>
                        <product name="windows"/>
                </action>
        </successor>
</rule>
<rule id="5">
        <predecessor name="facade"></predecessor>
        <successor>
                <action name="subdiv">
                        <param name="axis" value="Y"/>
                        <param name="div1" value="1"/>
                        <param name="div2" value="1"/>

                        <product name="groundFloor"/>
                        <product name="ledgeFloor"/>
                </action>
        </successor>
</rule>
<rule id="6">
        <predecessor name="sidewing"></predecessor>
        <successor>
                <action name="subdiv">
                        <param name="axis" value="Y"/>
                        <param name="div1" value="1"/>
                        <param name="div2" value="1"/>

                        <product name="windows"/>
                        <product name="ledgeFloor"/>
                </action>
        </successor>
</rule>
<rule id="7">
        <predecessor name="ledgeFloor"></predecessor>
        <successor>
                <action name="import">
                        <param name="type" value="ledge"/>

                        <product name="ledge"/>
                </action>
        </successor>
</rule>
<rule id="8">
        <predecessor name="groundFloor"></predecessor>
        <successor>
                <action name="subdiv">
                        <param name="axis" value="X"/>
                        <param name="div1" value="1"/>
                        <param name="div2" value="1"/>
                        <param name="div2" value="4"/>
```

```
                        <product name="door"/>
                        <product name="door2"/>
                        <product name="windows"/>
                </action>
        </successor>
</rule>
<rule id="9">
        <predecessor name="b2_t"></predecessor>
        <successor>
                <action name="import">
                        <param name="type" value="window"/>

                        <product name="floors"/>
                </action>
        </successor>
</rule>
<rule id="10">
        <predecessor name="door"></predecessor>
        <successor>
                <action name="import">
                        <param name="type" value="door"/>

                        <product name="doorOk"/>
                </action>
        </successor>
</rule>
<rule id="11">
        <predecessor name="windows"></predecessor>
        <successor>
                <action name="import">
                        <param name="type" value="window"/>

                        <product name="floors"/>
                </action>
        </successor>
</rule>
</XMLshape>
```

11.3 Codi XML de l'edifici número 4 de Müller i Wonka

```
<XMLshape>
	<rule id="1">
		<predecessor name="base"></predecessor>
		<successor>
			<action name="createBase">
				<param name="type" value="box" />
				<param name="sx" value="5" />
				<param name="sy" value="10" />
				<param name="sz" value="6" />

				<product name="b1" />
			</action>
			<action name="createBase">
				<param name="type" value="box" />
				<param name="sx" value="5" />
				<param name="sy" value="9" />
				<param name="sz" value="5" />

				<product name="b2" />
			</action>
			<action name="createBase">
				<param name="type" value="box" />
				<param name="sx" value="4" />
				<param name="sy" value="1" />
				<param name="sz" value="2" />

				<product name="b3" />
			</action>
			<action name="createBase">
				<param name="type" value="box" />
				<param name="sx" value="5" />
				<param name="sy" value="1" />
				<param name="sz" value="2" />

				<product name="b4" />
			</action>
			</successor>
	</rule>
	<rule id="2">
		<predecessor name="b2"></predecessor>
		<successor>
			<action name="translate">
				<param name="tx" value="-9"/>
				<param name="ty" value="0"/>
				<param name="tz" value="0"/>

				<product name="b2_t"/>
			</action>
		</successor>
	</rule>
	<rule id="3">
		<predecessor name="b3"></predecessor>
		<successor>
			<action name="translate">
				<param name="tx" value="-4"/>
				<param name="ty" value="6"/>
				<param name="tz" value="0"/>

				<product name="b3_t"/>
			</action>
		</successor>
	</rule>
	<rule id="4">
```

```
        <predecessor name="b4"></predecessor>
        <successor>
                <action name="translate">
                        <param name="tx" value="0"/>
                        <param name="ty" value="0"/>
                        <param name="tz" value="-2"/>

                        <product name="b4_t"/>
                </action>
        </successor>
 </rule>
 <rule id="5">
        <predecessor name="b1"></predecessor>
        <successor>
                <action name="comp">

                        <product name="facade"/>
                        <product name="windows"/>
                        <product name="roof"/>
                        <product name="floor"/>
                        <product name="sidewing"/>
                        <product name="sidewing"/>
                </action>
        </successor>
</rule>
 <rule id="6">
        <predecessor name="b2_t"></predecessor>
        <successor>
                <action name="comp">

                        <product name="oneWindowfloor"/>
                        <product name="oneWindowfloor"/>
                        <product name="roof"/>
                        <product name="floor"/>
                        <product name="oneWindowfloor"/>
                        <product name="oneWindowfloor"/>
                </action>
        </successor>
 </rule>
 <rule id="7">
        <predecessor name="b3_t"></predecessor>
        <successor>
                <action name="comp">

                        <product name="windows"/>
                        <product name="windows"/>
                        <product name="roof"/>
                        <product name="floor"/>
                        <product name="wall"/>
                        <product name="wall"/>
                </action>
        </successor>
 </rule>
 <rule id="8">
        <predecessor name="b4_t"></predecessor>
        <successor>
                <action name="comp">

                        <product name="ledgeOnly"/>
                        <product name="ledgeOnly"/>
                        <product name="roof"/>
                        <product name="floor"/>
                        <product name="ledgeOnly"/>
                        <product name="ledgeOnly"/>
                </action>
        </successor>
 </rule>
```

```
<rule id="9">
	<predecessor name="facade"></predecessor>
	<successor>
		<action name="subdiv">
			<param name="axis" value="Y"/>
			<param name="div1" value="1"/>
			<param name="div2" value="9"/>

			<product name="groundFloor"/>
			<product name="oneWindowfloor"/>
		</action>
	</successor>
</rule>
<rule id="10">
	<predecessor name="oneWindowfloor"></predecessor>
	<successor>
		<action name="subdiv">
			<param name="axis" value="X"/>
			<param name="div1" value="1"/>
			<param name="div2" value="3"/>
			<param name="div3" value="1"/>

			<product name="wall"/>
			<product name="windows"/>
			<product name="wall"/>
		</action>
	</successor>
</rule>
<rule id="11">
	<predecessor name="DoorFloor"></predecessor>
	<successor>
		<action name="subdiv">
			<param name="axis" value="X"/>
			<param name="div2" value="1"/>
			<param name="div3" value="1"/>

			<product name="door"/>
			<product name="wall"/>
		</action>
	</successor>
</rule>
<rule id="12">
	<predecessor name="sidewing"></predecessor>
	<successor>
		<action name="subdiv">
			<param name="axis" value="X"/>
			<param name="div1" value="1"/>
			<param name="div2" value="4"/>
			<param name="div3" value="1"/>

			<product name="wall"/>
			<product name="windows"/>
			<product name="wall"/>
		</action>
	</successor>
</rule>
<rule id="13">
	<predecessor name="ledgeFloor"></predecessor>
	<successor>
		<action name="import">
			<param name="type" value="ledge"/>

			<product name="ledge"/>
		</action>
	</successor>
</rule>
<rule id="14">
```

```
        <predecessor name="groundFloor"></predecessor>
        <successor>
                <action name="subdiv">
                        <param name="axis" value="X"/>
                        <param name="div1" value="1"/>
                        <param name="div2" value="1"/>
                        <param name="div3" value="1"/>
                        <param name="div4" value="1"/>
                        <param name="div5" value="1"/>

                        <product name="wall"/>
                        <product name="ledgeOnly"/>
                        <product name="door"/>
                        <product name="ledgeOnly"/>
                        <product name="wall"/>
                </action>
        </successor>
</rule>
<rule id="15">
        <predecessor name="door"></predecessor>
        <successor>
                <action name="import">
                        <param name="type" value="door"/>

                        <product name="doorOk"/>
                </action>
        </successor>
</rule>
<rule id="16">
        <predecessor name="windows"></predecessor>
        <successor>
                <action name="import">
                        <param name="type" value="window"/>

                        <product name="floors"/>
                </action>
                <action name="import">
                        <param name="type" value="ledge"/>
                        <param name="location" value="-2"/>
                        <param name="orientation" value="H"/>
                        <param name="noWall" />

                        <product name="floors"/>
                </action>
        </successor>
</rule>
<rule id="17">
        <predecessor name="wall"></predecessor>
        <successor>
                <action name="import">
                        <param name="type" value="ledge"/>
                        <param name="location" value="-1"/>
                        <param name="orientation" value="V"/>
                        <param name="noWall" />

                        <product name="floors"/>
                </action>
                <action name="import">
                        <param name="type" value="ledge"/>
                        <param name="location" value="2"/>
                        <param name="orientation" value="V"/>
                        <param name="noWall" />

                        <product name="floors"/>
                </action>
                <action name="import">
                        <param name="type" value="ledge"/>
```

```
                        <param name="location" value="-2"/>
                        <param name="orientation" value="H"/>
                        <param name="noWall" />

                        <product name="floors"/>
                </action>
        </successor>
    </rule>
    <rule id="18">
        <predecessor name="ledgeOnly"></predecessor>
        <successor>
                <action name="import">
                        <param name="type" value="ledge"/>
                        <param name="location" value="-2"/>
                        <param name="orientation" value="H"/>

                        <product name="floors"/>
                </action>
        </successor>
    </rule>
</XMLshape>
```