Integració d'Informació Geogràfica al motor Unity3D

Marc Garcia Puig

Resum– Cada cop es fa servir més el *deep learning* per la conducció autònoma, però per utilitzar-lo es necessiten moltes dades amb anotacions especials anomenades *ground truth* (com la profunditat de la imatge o el *optical flow*). Anotar aquestes dades a mà és molt difícil, potser impossible. La forma més ràpida d'aconseguir gran quantitat d'aquestes dades, és generar aproximacions en motors gràfics com fa el dataset Synthia (que funciona en Unity3D). Però per generar una ciutat sintètica dins d'un d'aquests motors es necessita molta feina en el modelatge, texturitzat i arquitectura de la ciutat i terreny i això és molt lent. Aquest treball proposa fer un sistema per al projecte de Synthia que, a partir de diferents tipus d'informació geogràfica obtinguda d'Internet sigui capaç de generar, a temps real, una aproximació 3D del món real, i d'aquí extreure dades de la mateixa qualitat que les que es generen actualment.

Paraules clau– Unity3D, Generació procedural, Synthia-dataset, Sistemes d'informació geogràfica

Abstract– Nowadays deep learning is been used more and more for many autonomous driving applications, however for effectively using deep learning we need a lot of special annotations called ground truth. Annotate this data manually is very hard and maybe impossible. That's why the easiest way to get this huge quantity of data is to use graphical engines like Synthia does (using Unity3D). However, to generate a synthetic city like this you need also a lot of manual work, modeling, texturing and architecture of the city and the terrain and that requires a lot of labor. For this work we propose to do a system that is included inside the Synthia project that from different kinds of geographic information that is obtained from Internet, we are capable of generating in real time a 3D approximation of the world and from here we can extract data from the same quality as those that are currently generated.

Keywords– Unity3D, Procedural generation, Synthia-dataset, Geographic information system

1 Introducció

El *Deep learning* (DL), sorgit als anys 80, és un tipus de *machine learning* i una tecnologia molt popularitzada en els últims cinc anys. És emprada cada cop més per un gran ventall d'usos diferents; des de reconeixement de veu, fins a visió per computador.

El DL basat en l'aprenentatge supervisat es centra en la utilització d'una gran quantitat de dades a causa de que té una capacitat d'aprenentatge *end-to-end*, és a dir, que s'aplica el mínim coneixement humà entre l'algoritme i els resultats que s'obtenen. És per això que les dades han de tenir tanta variabilitat i informació com sigui possible i així tractar el número més gran de casos diferents perquè els algoritmes puguin entendre bé el problema i afrontar tot tipus de casos.

Si ens centrem concretament en les dades que utilitza el DL aplicat a la visió per computador, podem trobar-nos amb problemes com, per exemple, la dificultat (o impossibilitat en alguns casos) d'anotar el *ground truth* (GT). El GT és el terme que utilitzem per referir-nos a la informació que no podem percebre només amb el color RGB d'una imatge, però que sabem del cert que és així mitjançant altres mètodes, com ara la profunditat, el *optical flow* o l'anotació de l'equivalent real que defineix cada píxel de la imatge (ja sigui una casa, una persona ...). En tenim un exemple a la Figura 1. Anotar aquesta informació (i fer-ho bé) a mà és

• E-mail de contacte: marcgpuig@gmail.com
• Menció realitzada: Enginyeria del Software
• Treball tutoritzat per: Antonio Lopez Penar (Ciències de la Computació)
• Curs 2016/17

pràcticament impossible tenint en compte, a més, el temps que es tardaria a fer-ho.

D'aquí sorgeix la necessita de crear ràpidament grans volums de dades amb el seu GT correctament anotat. Treballs com el d'en Ros et al. [1], demostren que es poden utilitzar dades de caràcter similar, generades a partir de mons sintètics integrats dins de motors gràfics (com Unity3D o Unreal Engine 4). És per això que existeixen sets de dades (*dataset*) com el de Synthia (Fig. 1), integrat dins d'Unity3D i directament preparat per la conducció autònoma. El problema d'aquests tipus de datasets és què requereixen el disseny humà per generar les ciutats i zones d'on recollir aquestes dades. Per això és necessària una forma d'automatitzar aquest procés.

Una opció seria la generació procedural d'un món virtual pseudoaleatori, però té una molt elevada complexitat. Una altra opció és la d'utilitzar dades del món real que no poden ser directament útils tal com estan, però que un cop tractades poden servir per recrear zones del món (ex: mapes d'alçades, informació urbana, clima ...). Per contra aquesta informació sòl ser poc precisa i no ens aporta un nivell suficient de realisme.

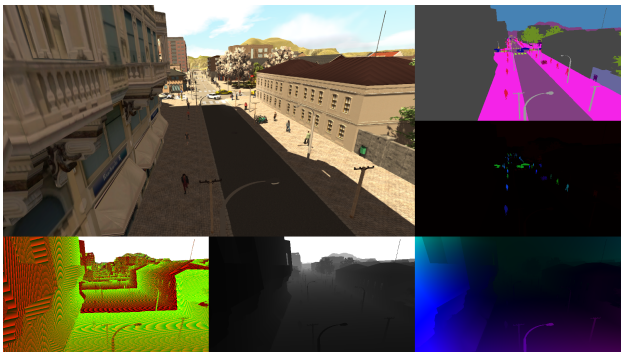

Fig. 1: *Diferents ground truth de Synthia: es poden veure diferents representacions de la profunditat, la segmentació semàntica i el optical flow.*

Per tant aquest treball pretén ser un estudi de viabilitat i un primer pas en la implementació d'un projecte de grans dimensions que utilitzi diferents tecnologies de les dues opcions mencionades anteriorment, intentant aconseguir un equilibri tot aprofitant els punts forts de cadascuna d'elles.

A l'haver estat col·laborant íntegrament dins del projecte de Synthia durant força temps, he pogut veure de primera mà els problemes que suposa la creació de l'entorn. Per generar imatges en una ciutat nova, s'han de crear tots els models 3D, animacions i textures, repartir-los per la ciutat de forma realista i redimensionar-los en cas que sigui necessari. També cal amagar zones que no s'han modelat o que, directament, no es volen incloure per augmentar el rendiment del motor gràfic, etc. Tot aquest esforç no garanteix una semblança directa amb la realitat.

Si ens fixem en algunes d'aquestes tasques veurem que es poden automatitzar perquè segueixen un conjunt de patrons com ara la distribució de les carreteres, senyals de tràfic o el disseny i la posició de cases i edificis. Hi ha informació que no requereix gairebé cap tipus de control, o és molt fàcil de controlar com poden ser els models dels cotxes o els colors d'aquests o les posicions dels arbres dins i fora de la ciutat entre altres elements decoratius com fanals i boques d'incendi.

Hi ha altres elements més crítics com podria ser el mateix disseny de les ciutats que, tot i que es pot automatitzar proceduralment amb algoritmes força complicats, sol quedar millor quan una persona amb experiència en aquesta tasca ho fa, degut a la quantitat d'elements a tenir en compte entre ells la creativitat. És important tenir en compte que les ciutats s'han construït al llarg dels anys i hi ha una història darrere cada carrer, plaça o edifici, coses que els algoritmes actuals no tenen en compte i per tant tenen risc d'acabar generant informació poc variant o repetida. És per això que s'ha decidit utilitzar informació real i només generar allò del que no es té informació. Així disminuïm el risc de generar informació redundant. Tot i que les dades no tenen per què assemblar-se directament a la realitat física, al final, que millor que entrenar amb unes dades provinents d'aquest món on s'utilitzaran en un futur dits algoritmes?

Tot això és una tasca de grans dimensions, i per això s'ha decidit ser realista des d'un bon principi. Per tant aquest treball se centra en la part més tediosa d'un projecte així: fer una bona base (el *core*) per al projecte de forma que pugui ser expandit i millorat el més fàcil i ràpid possible.

1.1 Objectius

En particular, l'objectiu del projecte és integrar un software al motor gràfic Unity3D que permeti la generació dinàmica de contingut obtingut a través d'informació *GIS* (*Geographic Information system*) obtinguda a temps real de diferents fonts d'Internet. El sistema també ha de ser capaç de generar proceduralment certs continguts dels quals no disposem de forma total o parcial (Figura 2).

Per aconseguir això, s'han hagut de completar les següents etapes:

1. Descàrrega d'informació GIS.
2. *Parseig* i estructuració de les dades.
3. Generació de la malla 3D.
4. Optimització multithreading dels passos un i dos.

S'han utilitzat les millors pràctiques per al correcte desenvolupament del software que consisteixen en l'estudi de l'estat de l'art, fer una anàlisis dels requisits i una planificació de les tasques. Només després es pot fer el disseny del software i finalment la implementació i el test.

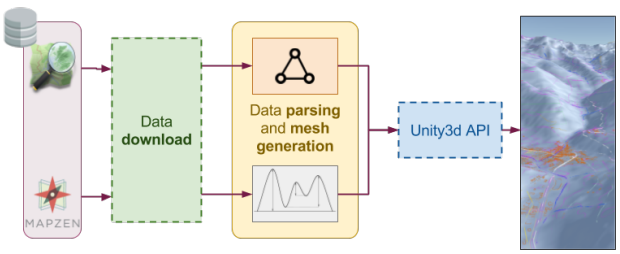

Fig. 2: *Esquema bàsic del funcionament del projecte.*

1.2 Organització del document

En aquest document quedarà reflectida la feina realitzada al llarg del treball; analitzarem l'estat de l'art, les tecnologies utilitzades, la metodologia seguida del desenvolupament amb els punts més importants del projecte, els resultats i finalment una breu conclusió.

1.3 Estat de l'Art

A causa de la necessitat de generar informació sintètica al llarg dels últims anys, la generació de contingut procedural ha estat una branca força treballada per diversos sectors de tot tipus d'indústries. Hi ha moltes publicacions que afronten i resolen problemes que podrien sorgir en fases més avançades d'aquest projecte. Ara en veurem uns exemples.

Ja s'ha abordat el tema de la generació de malles de carreteres a partir d'informació GIS: un exemple ens el dona H.H. Nguyen et al. en la seva publicació *Realistic Urban Road Network Modelling From GIS Data* [2] en la que es podrien basar les etapes intermèdies d'aquest projecte a l'hora d'implementar certes funcionalitats.

Les ciutats poden tenir arquitectures realment complicades, per tant és necessari un model molt sòlid per la generació d'edificis, siguin cases, naus industrials o gratacels. En Müller et al. ha fet grans aportacions a la matèria mitjançant publicacions dedicades a la generació d'edificis [3] i de ciutats [4], és per això que el seu mètode per generar edificis podria ser implementat més endavant. Hi ha altres molt bones contribucions en aquestes tècniques com la proposada per en T.Kelly [5] on es genera un edifici a partir de la seva silueta lateral, això permetria generar diferents tipus d'edificis representant diferents cultures.

Existeixen també algoritmes que a partir de trossos d'imatges reals en poden extreure textures reutilitzables per donar detall a edificis o a altres models generats [6]. Aquestes tècniques anteriors es podrien utilitzar juntament amb models més complets com l'elaborat per en G.Kelly i la seva tesis [7].

Sense abandonar el tema de la generació de contingut, les carreteres són una part crítica. Hi ha softwares (p. ex. EasyRoads3D [8]) integrats per tercers a Unity3D. Aquests generen la malla a través del terreny i el modifiquen perquè s'hi adhereixi. A Synthia ja hem utilitzat aquesta eina i no hem quedat del tot satisfets a causa de la inestabilitat en les versions actuals. És important que les carreteres no siguin repetitives, que les línies dels carrils segueixin les normes de tràfic, que hi hagi cert nivell d'imperfecció (com sotracs i taques d'oli)...

Un altre aspecte a mencionar és la necessitat de tenir vianants i cotxes movent-se per la ciutat de forma automàtica i realista. És una àrea amb bones publicacions com *Interactive Hybrid Simulation of Large-Scale Traffic* [9] on ens explica com desplaçar grans quantitats de cotxes, o *Transforming GIS Data into Functional Road Models for Large-Scale Traffic Simulation* [10] que ens seria molt útil amb la nostra informació GIS, i es podria aprofitar per adaptar-ho als vianants juntament amb sistemes de moviment d'aquests tals com el proposat per Yunchao Qu et al. [11].

També existeixen algoritmes per l'optimització del renderitzat del terreny com ara [12] que, tot i que Unity3D ja té el seu propi sistema funcional i optimitzat per ell, està bé tenir alternatives per si hi hagués problemes en el futur, ja que, teòricament, aquest nou model hauria de millorar el rendiment respecte al qual ja hi ha.

2 TECNOLOGIES

2.1 Sistema Operatiu

S'ha utilitza Windows 10 (64 bits) ja que les *builds* de Unity3D per Linux són noves i no són gaire estables. A més windows és l'únic sistema que pot utilitzar *DirectX* a més de *OpenGL* i això ens permet comprovar que el codi funciona correctament per les dues plataformes.

2.2 Motor gràfic

S'ha decidit utilitzar Unity3D (concretament les versions 5.4.3f1 i 5.6) principalment perquè és l'entorn on s'ha creat Synthia i ja hi tenim tot el sistema per la correcta captura d'informació testejada i funcional. Aquest sistema de captura d'imatges no ha d'interferir amb rés d'aquest projecte, s'ha d'aconseguir que siguin eines separades i que funcionin independentment entre elles, ja que no sempre es voldrà fer-les servir juntes.

Unity3D pot fer servir *C#* (.Net 2.0) o *Javascript* per ser programat. S'ha utilitzat C#, ja que és el mateix llenguatge que utilitza Synthia.

A més degut a la nostra experiència en Unity3D, el rendiment ha sigut major i més fluid.

2.3 Programari

Unity3D aporta la interfície gràfica per al motor, però a l'hora de programar has d'utilitzar programari extern. Unity3D ofereix diferents opcions; per defecte l'editor de scripts de Unity3D és *MonoDevelop*, però s'ha decidit utilitzar principalment *Visual Studio* (versions 2015 i 2017). Aquest aporta el mateix que MonoDevelop però amb la comoditat de ser un programa més conegut, amb més suport i ofereix eines per la integració completa amb Unity3D.

També s'han utilitzat altres softwares com *Sublime Text 3* com a lector d'arxius de text, *Python* (versió 2.7.11) per provar funcions matemàtiques de forma ràpida abans d'implementar-les dins del projecte principal i *Git* (versió 2.13) per al control de versions.

3 METODOLOGIA

El projecte s'ha dut a terme amb una adaptació de la metodologia SCRUM, i tot i que no és una metodologia pensada per a un projecte d'una sola persona, és àgil, adaptable i funciona per tasques i això s'ajusta a les necessitats del projecte, ja que és ampli, canviant i ha sigut necessària l'obtenció d'entregues regulars. Es pot veure un diagrama de Gantt de l'organització del projecte a l'Apèndix B.1.

Per gestionar errors i millores s'ha fet servir *Github* i aprofitant aquest s'ha decidit fer un projecte de codi obert com s'explicarà en els punts següents.

3.1 Control de versions

El control de versions és pràcticament un element indispensable per qualsevol projecte participatiu, o amb caràcter *open source*. En aquest projecte s'ha cregut necessari l'ús del control de versions sobretot per poder treballar des de diferents llocs i veure diferències entre versions, però també i no menys important, per seguretat, integritat de dades i evitar la pèrdua d'aquestes. Per tant s'ha decidit des d'un bon principi utilitzar *Git*.

L'elecció ha sigut principalment per la compatibilitat amb Unity3D [13], tot i la seva dificultat en la integració en certs projectes [14].

Per distribuir el control de versions s'ha utilitzat Github, una plataforma de hosting de repositoris Git. S'ha triat aquest principalment perquè és gratuït, simple i potent. Per tant no cal perdre el temps en configuracions innecessàries. Github aporta eines per la gestió de resolució de problemes en el codi anomenat *Issues* i ajuda a l'organització de les tasques a realitzar gràcies als *Projects*, que són taules dinàmiques amb els continguts que vols afegir, arreglar, testejar...

3.2 Open source

S'ha cregut convenient fer el projecte *open source*, és a dir, que és un codi públic i tothom pot accedir-hi. D'aquesta manera gent de tot arreu pot contribuir-hi a través de Github, no només codi, sinó alertes de *bugs*, idees i fins i tot sol·licituds de noves funcionalitats.

Està sota llicència MIT [15], una llicència molt permissiva que deixa fer a la gent el que vulgui sempre i quan s'inclogui el *copyright* i la llicència original a totes les còpies del software o el codi.

4 DESENVOLUPAMENT

El desenvolupament s'ha realitzat en les etapes de disseny, implementació i testeig. A continuació es detallaran els punts que es creuen més rellevants d'aquest treball.

4.1 Disseny

En aquesta etapa, una vegada fixats els objectius i requisits del projecte, s'ha procedit a fer el disseny del software. Hi ha dos punts destacables que han definit el disseny final:

1. La *pipeline* de Unity3D. Com que és un software tancant, no ens facilita el seu codi font, per tant ens hem de centrar a entendre la seva estructura per tal de seguir-la. Per sort té una documentació força extensa i ens proporciona ajudes com aquest diagrama del seu funcionament general [16].

2. Modularitat en el codi. De bon principi i en tot moment al llarg del projecte s'ha intentat fer el codi el més modular i ampliable possible. Això facilita la cooperació d'altres programadors que volguessin, per exemple, afegir dades d'una plataforma completament nova o portar els algoritmes a un altre motor gràfic, gràcies al patró *adapter*. També s'ha procurat parametritzar tots els valors i fer així el codi més flexible per l'usuari final i adaptar-lo a les seves necessitats particulars.

4.2 Informació GIS

Per l'obtenció de dades d'aquest projecte s'han fet servir principalment els serveis que ens proporcionen dos APIs:

- OpenStreetMaps [17] (mapa urbà)
- Mapzen [18] (Mapa d'alçades)

A continuació s'explicarà com s'ha interactuat amb les dades de cada una d'aquestes APIs.

4.2.1 Mapa urbà

OpenStreetMap (també conegut com a OSM) és un projecte col·laboratiu per crear mapes de contingut lliure usant dades obtingudes mitjançant dispositius GPS mòbils, ortofotografies i altres fonts de dades. Les dades dels mapes (coordenades) i les imatges obtingudes amb elles es lliuren sota la llicència Open Database License [19]. Les seves coordenades es troben en el format WGS84 i són visualitzades normalment fent servir la projecció esfèrica de Mercator [20].

OSM retorna les dades en format *XML*, un format fàcilment interpretable. La informació de OSM està formada principalment pels següent *elements*:

- **Nodes**: Representa un punt específic a la superfície terrestre utilitzant la seva latitud i longitud. Cada node té com a mínim un identificador únic i pot tenir $0..n$ *tags*. Els nodes es fan servir per traçar la forma dels camins.

- **Ways** o camí: Són llistes ordenades formades per entre $2..2000$ nodes que defineixen una línia poligonal. Representen característiques geogràfiques com rius o carreteres. Els ways poden representar també àrees com edificis o boscos, en aquests casos el primer i l'últim node són els mateixos.

- **Relation**: És una estructura de dades útil per diversos propòsits que estableix una relació entre $2..n$ elements, com ara relacions entre camins.

- **Tag**: Tots els tipus d'elements anteriors poden tenir una col·lecció de tags i descriuen el significat de l'element al qual pertanyen. Els tags estan formats per una clau i un valor amb el format `key=value` dins de l'XML, i per tant un element no pot tenir més d'un tag amb la mateixa clau, ja que descriuen un valor únic.

4.2.2 Mapa d'alçades

Mapzen és un laboratori cartogràfic de codi obert que elabora eines i serveis de mapeig on OSM és un element central dels seus productes. S'ha utilitzat concretament el *Terrain Tiles Service* que ens ofereix mapes d'alçades representades en imatges *PNG* de 256x256 píxels.

Per comunicar-nos amb la API de Mapzen i demanar-li una zona concreta, s'han d'utilitzar els anomenats *Slippi Tiles* als que anomenarem *ST*. Per fer la conversió entre la longitud i latitud i aquest tipus de projecció es poden veure les fórmules de l'Apèndix B.2. Els ST representen una secció quadrada sobre la projecció Mercator, definida per dos índexs i un nivell de zoom en el rang $0..19$ on el 0 comprèn tot el globus terraqüi, el nivell 1 és la divisió entre dos en

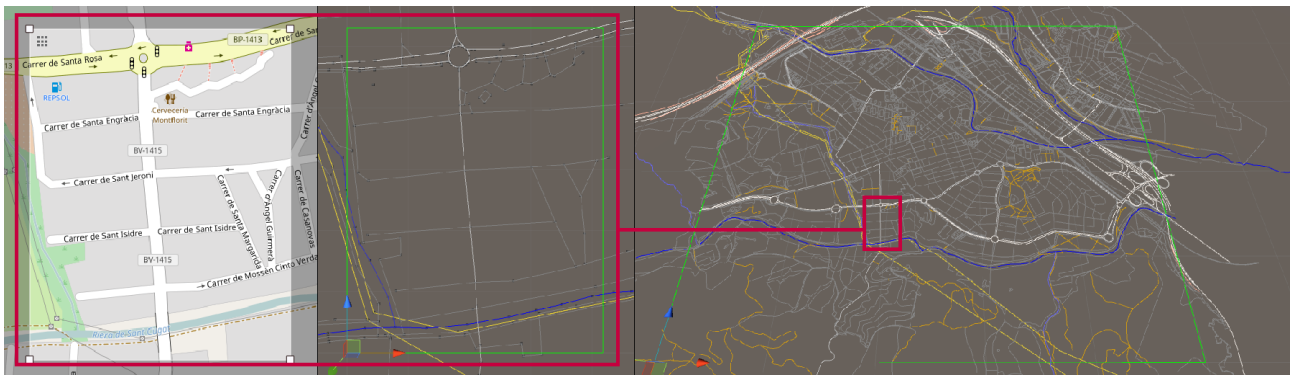

Fig. 3: *D'esquerra a dreta: informació urbana d'una part de Cerdanyola de OpenStreetMaps, la mateixa secció carregada a Unity3D i finalment tota Cerdanyola.*

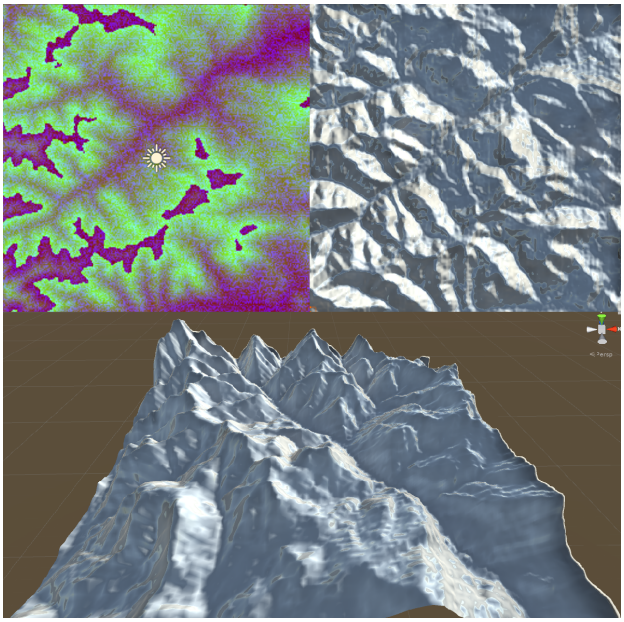

Fig. 4: *La imatge superior esquerra mostra la codificació que d'alçades que retorna Mapzen, mentre que la dreta la vista ortogràfica superior de l'alçada equivalent. Finalment la imatge inferior és la mateixa alçada representada amb una projecció perspectiva.*

alçada i en altura formant una graella de 2x2 i així fins a arribar al nivell 19 que és el zoom màxim que es té registrat a certes parts de la Terra amb uns (2^{2z} on z és el nivell de zoom) 274 milers de milions de cel·les.

L'alçada per cada píxel que ens retorna la API estan codificades utilitzant els diferents canals de color RGB. Cada canal representa una un valor més o menys significatiu de l'alçada, d'aquesta manera es pot codificar en tres bytes fent un total de 256^3 valors possibles per totes les alçades de la Terra. Aquesta alçada es pot extreure mitjançant la fórmula

$$H(r,g,b) = \left(\frac{r * 256 + g + \frac{b}{256}}{256}\right) - \left(\frac{256^2}{2}\right) \quad (1)$$

on r, g i b són els tres canals de color de la imatge. Es pot veure quin aspecte té aquesta codificació a la imatge a la Figura 4.

Per saber la distància representada per un píxel fem servir la fórmula

$$S = C \cdot \frac{cos(y)}{2^{(z+8)}} \quad (2)$$

(provinent de [21]) on C és la circumferència equatorial de la Terra (40075.016686 km [22]), z el nivell de zoom i y la latitud de la posició que volem saber.

4.3 Sistema de chunks

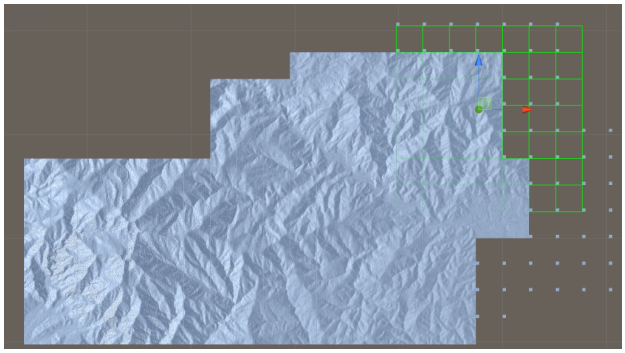

Fig. 5: *Sistema de chunks amb la informació d'alçades en funcionament.*

En aquest projecte entendrem com a *chunk* tota la informació que conté una secció quadrada de terreny de mida invariable. Aquesta informació pot ser qualsevol cosa, però seran principalment dades GIS i depenent de quines siguin es guardarà amb diferents estructures com veurem en els següents apartats.

Per tant un sistema de chunks ens permet administrar una gran quantitat d'informació de forma eficient, ja que només tindrem en compte els chunks propers a la posició del nostre punt de vista dins de la simulació.

Es divideix l'espai horitzontalment de Unity3D en una quadrícula on cada cel·la equivaldrà a un chunk com a la Figura 5. Ara es mira en quina d'aquestes caselles es troba el nostre punt de vista i s'utilitza la distància de Chebyshev per determinar quins chunks volem utilitzar. Aquesta distància segueix la fórmula

$$D(p,q) = \max\left(|q_x - p_x|, |q_y - p_y|\right) \quad (3)$$

on p i q són dos punts amb coordenades cartesianes (p_x, p_y) i (q_x, q_y) respectivament. Quan el sistema detecta que el punt de vista canvia de chunk, es recalculen les distàncies i

es descarreguen de la memòria els chunks que no necessitem i es carreguen els nous chunks adjacents. Es poden fer servir altres heurístiques més complicades per decidir quins chunks volem carregar però aquesta ja és funcional.

A la Figura 5 podem veure els chunks descarregats a l'esquerra i uns quadres verds que indiquen quins chunks volem mostrar. A la dreta podem veure uns quadrats blancs més petits que indiquen que encara no s'ha acabat de descarregar la informació d'alçades d'aquest chunk.

4.4 Multithreading

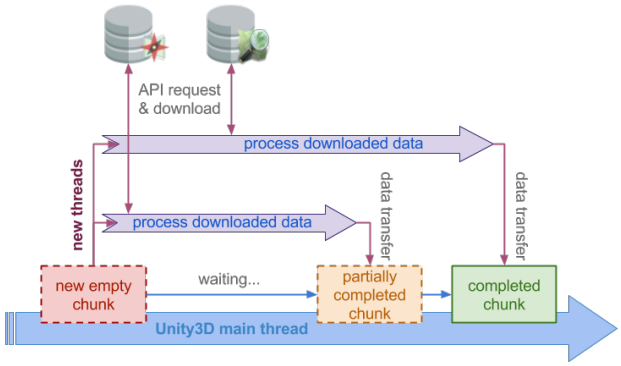

Fig. 6: Diagrama explicatiu del sistema de multithreading utilitzat en el projecte. En aquest cas descarregant dades de les APIs de Mapzen (a l'esquerra) i OpenStreetMaps (a la dreta).

És de vital importància per al projecte que la descàrrega, interpretació, selecció i modelat geomètric de la informació no afecti al rendiment del motor gràfic. Aquest és un punt crític, ja que no es pot permetre que una simulació s'executi a un *framerate* variant. El framerate és el nombre d'imatges per segon que el sistema és capaç de generar a través del motor gràfic. Si el nostre framerate és variable, el set d'imatges que obtindrem al final no seguirà un flux continu, sinó que veurem alteracions en la velocitat de la seqüència.

És per aquest motiu que s'ha decidit assignar un generós espai de temps per la resolució d'aquesta tasca, sacrificant així, altres punts que es volien haver implementat al llarg d'aquesta etapa del treball.

S'ha arribat a la conclusió que la millor manera de resoldre-ho és fer servir *multithreading* (o múltiples fils d'execució). Tot i que no hi ha cap impediment a l'hora d'utilitzar-lo amb Unity3D, aquest no té una API *thread-safe*. Això significa que, dins d'un thread, no podem utilitzar cap funció de la API de Unity3D ni accedir a cap dels seus objectes.

El funcionament general dels threads del projecte es pot veure reflectit a la Figura 6. La idea principal es resumeix en que quan el codi que tenim executant en el thread principal veu que s'ha de crear un nou chunk, el genera, i aquest últim crea i executa un thread per cada API a la que es vol accedir i s'espera mentre els threads treballen. En el moment en que un thread acaba, el chunk és informat i actualitza la seva informació amb la del thread i l'elimina. Farà això amb cada thread. D'aquesta manera cada chunk administra els seus threads i per tant la seva informació.

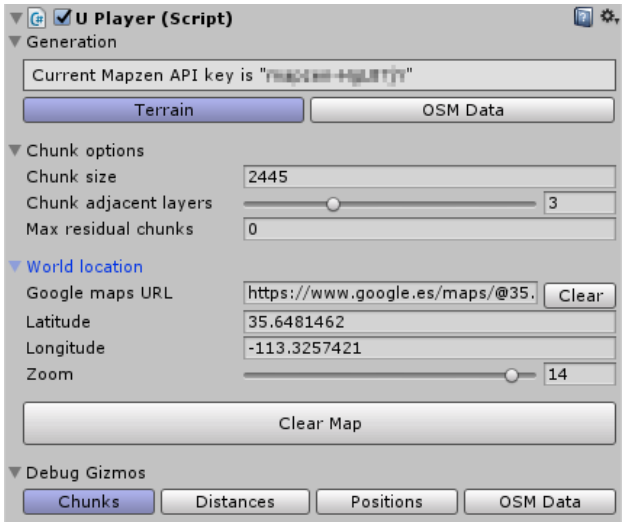

Fig. 7: Interfície gràfica personalitzada per al projecte dins del motor Unity3D del projecte.

S'han provat dos mètodes diferents a l'hora de gestionar els threads, però aquest és el que ha donat millors resultats.

4.5 Interfície gràfica

L'usuari final d'aquest software no ha de ser necessàriament cap expert en Unity3D i per tant s'ha d'intentar que tothom se senti còmode estilitzant-lo. Una de les millors maneres de fer l'eina *user-friendly* és la creació d'una interfície gràfica agradable, intuïtiva i sobretot fàcil d'utilitzar.

Unity3D permet crear interfícies personalitzades per als scripts. Aprofitant això s'ha fet una interfície per al projecte, de la que es pot destacar:

- Decidir quin tipus d'informació vols descarregar i utilitzar.
- Avisos de falta de configuració de l'eina p. ex. si falten claus per l'ús de certes APIs.
- Conversió de URLs de *Google Maps* per extreure'n la longitud i latitud ràpidament.
- Decidir quins aspectes vols debugar.
- Interfície molt senzilla i amb funcions preparades per expandir-la mantenint l'estil.

5 Resultats

La realització d'aquest projecte ha permès entendre diferents models geogràfics i així utilitzar aquests coneixements per unificar dades GIS de diferents fonts amb diferents sistemes de coordenades en el motor Unity, com es pot veure a la Figura 8. Això s'ha aconseguit de forma òptima gràcies a la utilització de multithreading.

S'ha probat el texturitzat

5.1 Testing

La naturalesa gràfica del projecte afavoreix el *exploratory testing*, ja que visualment entra molta més informació i es poden trobar molts més errors que no pas a tràbees d'una

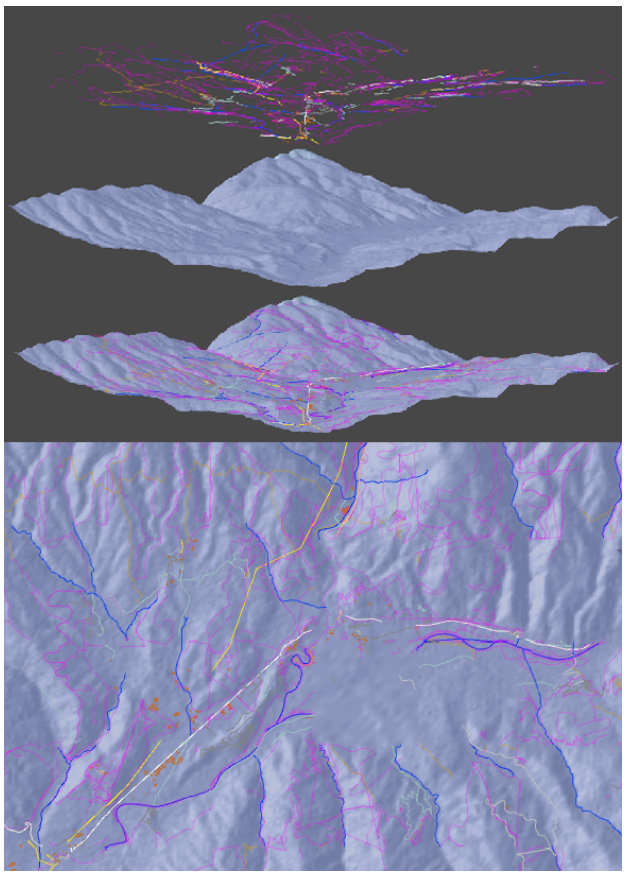

Fig. 8: *Diferents capes de dades GIS obtingudes de les APIs de OpenStreetMaps i Mapzen i representades en el motor Unity3D.*

consola. S'estan revisant activament els resultats i entenent per on fallen els mètodes, el que permet millorar cada cop més els tests.

S'ha utilitzat un repositori [23] que conté casos de testing preparats per la informació OSM. Aquests casos s'han fet servir per testejar el correcte funcionament de les classes encarregades de la representació d'informació urbana dins de Unity3D.

Tot i això en certes parts del codi (com les parts més matemàtiques) s'ha procurat fer probes amb tots els límits, com ara els límits del món i dels mapes. Gràcies a això s'han trobat errors que seran comentats més endavant. També s'ha creat codi per poder debugar separadament Unity3D de la nostra implementació amb un *log* separat especial per al test.

S'han utilitzat durant el desenvolupament zones del món conegudes i així conèixer de primera mà els resultats esperats. També s'han fet servir zones amb característiques geogràfiques molt particulars, com el Gran Canyó del Colorado, el Mont Fuji o els Pirineus.

5.2 Problemes coneguts

Hi ha diversos problemes que no es poden resoldre fàcilment:

- Problemes de render a grans distàncies de l'origen de coordenades de Unity3D (problema extern del motor).
- Artefactes en els mapes d'alçades en zones amb llacs o costes (problema extern).
- Actualment els ponts són ignorats i són carreteres normals, admès falta assegurar si es podria trobar l'alçada d'aquests (problema de disseny).
- Els servidors de les APIs poden fallar per causes externes i no donar servei (problema extern).
- Tot i poder saber quins nodes equivalen a un túnel, no podem perforar el terreny de Unity3D (problema extern).
- Són serveis gratuïts i oberts, per tant ningú assegura la veracitat ni la totalitat de les dades.

6 Conclusions

Tot i que inicialment aquest treball anava més encaminat a la generació procedural de carreteres, un cop analitzat amb profunditat s'ha decidit replantejar el tema i pensar en un projecte molt més gran i ambiciós, capaç de generar tot el que ens interessa que la informació GIS ens pogués oferir.

Podem concloure que hem aconseguit una base sòlida per al projecte, ja que s'ha aconseguit resoldre de forma eficient la descàrrega i representació d'informació GIS des de diferents plataformes d'Internet, de forma que no influeix directament en el rendiment del motor gràfic Unity3D. Com que la metodologia seguida ha sigut sempre intentar programar de la forma més modular possible, s'ha aconseguit un software ampliable, sempre utilitzant Git com a eina principal per mantenir el codi net i amb un caràcter open source.

Tot i que l'eina és bona, ara per ara no té una utilitat directa a causa de la falta de contingut real (com els objectes de l'entorn o les textures del terreny). Tot aquest contingut es pot afegir fàcilment gràcies a la modularitat del codi i posiciona-ho a través de tota la informació GIS de la que es disposa; des de la posició i informació dels senyals de tràfic, fins a quins carrers són o no transitables, tot passant per rius, vies de tren, tipus de vegetació...

6.1 Línies futures

Una funcionalitat fàcil d'implementar que es va tenir en compte en el disseny, però per falta de temps no s'ha pogut fer, és el guardat en local d'arxius per no haver d'accedir constantment a les APIs i així estalviar càrrega als servidors i permetre executar el codi *offline* o en cas que caiguin els servidors.

Aquest projecte té encara feina per endavant per ser funcional. Ara mateix es podria fer servir per a tot tipus d'aplicacions i és només qüestió d'imaginació trobar-li noves aplicacions i implementacions.

Per als nostres interessos particulars el proper pas seria la creació de malles 3D de carreteres procedurals, el texturitzat procedural del terreny i un sistema simple de tràfic dinàmic. Així es podrien començar a gravar els primers sets d'imatges, utilitzar-los en DL i obtenir els primers resultats reals del funcionament d'aquesta eina.

Seria molt interessant implementar qualsevol de les tecnologies esmentades en l'apartat de l'estat de l'art, ja que aportarien tecnologies recents per millorar la quantitat i la qualitat dels elements que apareixerien a les nostres dades finals.

AGRAÏMENTS

M'agradaria agrair a un seguit de persones que directament o indirectament han contribuït de diverses maneres al llarg del meu projecte.

Agrair primerament al meu tutor Antonio M. López per tot el material i llibertat de decisió que m'ha donat des del primer dia del treball. Al Felipe, al Néstor, al José i a tots els meus companys de feina del grup ADAS que tants consells m'han donat i m'han portat en bona direcció. A la meva família pels seus ànims incondicionals, no només en el projecte sinó durant tota la universitat, que tant m'han ajudat. Al David Fàbrega per no haver-me deixat mai que parés d'aprendre d'Informàtica.

També al Rubén, l'Enric, la Sílvia i tots els amics que porten amb mi des de l'escola, que mai han permès que em preocupés massa, sovint sense saber-ho, si les coses no m'anaven bé.

Finalment a tots els amics i coneguts de Cerdanyola, que sempre han tingut alguna frase que m'ha inspirat.

REFERÈNCIES

[1] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio Lopez. The SYNTHIA Dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *CVPR*, 2016.

[2] Hoang Ha Nguyen, Brett Desbenoit, and Marc Daniel. Realistic urban road network modelling from gis data. In *Eurographics Workshop on Urban Data Modelling and Visualisation*, pages 9–16. Eurographics Association Publisher, 2016.

[3] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. In *Acm Transactions On Graphics (Tog)*, pages 614–623. ACM, 2006.

[4] Yoav IH Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308. ACM, 2001.

[5] Tom Kelly and Peter Wonka. Interactive architectural modeling with procedural extrusions. *ACM Trans. Graph.*, 30(2):14:1–14:15, April 2011.

[6] Alexandre Kaspar, Boris Neubert, Dani Lischinski, Mark Pauly, and Johannes Kopf. Self tuning texture optimization. *Computer Graphics Forum*, 34(2):349–359, 2015.

[7] George Kelly. H.: Citygen: An interactive system for procedural city generation. In *In Proceedings of GDTW 2007: The 5th Annual International Conference in Computer Game Design and Technology*, pages 8–16, 2007.

[8] Unity3D assetstore easyroads3d free. `https://www.assetstore.unity3d.com/en/#!/content/987`. Accessed: 2017-06-10.

[9] Jason Sewall, David Wilkie, and Ming C Lin. Interactive hybrid simulation of large-scale traffic. In *ACM Transactions on Graphics (TOG)*, page 135. ACM, 2011.

[10] David Wilkie, Jason Sewall, and Ming C. Lin. Transforming gis data into functional road models for large-scale traffic simulation. *IEEE Transactions on Visualization and Computer Graphics*, 18(6):890–901, June 2012.

[11] Yunchao Qu, Ziyou Gao, Penina Orenstein, Jiancheng Long, and Xingang Li. An effective algorithm to simulate pedestrian flow using the heuristic force-based model. *Transportmetrica B: Transport Dynamics*, 3(1):1–26, 2015.

[12] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Bdam — batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum*, 22(3):505–514, 2003.

[13] Unity3D creating your first source control repository. `https://unity3d.com/es/learn/tutorials/topics/cloud-build/creating-your-first-source-control-repository` Accessed: 2017-06-11.

[14] Vianney Petit. Unity3D GitHub using git with unity. `https://gist.github.com/Ikalou/197c414d62f45a1193fd`. Accessed: 2017-06-11.

[15] MIT massachusetts institute of technology. `https://opensource.org/licenses/MIT`.

[16] Unity3D execution order of event functions. `https://docs.unity3d.com/Manual/ExecutionOrder.html`. Accessed: 2017-06-16.

[17] OpenStreetMaps API. `http://wiki.openstreetmap.org/wiki/API_v0.6`. Accessed: 2017-06-13.

[18] Mapzen API terrain tiles. `https://mapzen.com/documentation/terrain-tiles/`. Accessed: 2017-06-20.

[19] OpenStreetMap data license is ODbL. `https://blog.openstreetmap.org/2012/09/12/openstreetmap-data-license-is-odbl/`.

[20] Wikipedia. Mercator projection — wikipedia, the free encyclopedia, 2017. 22-05-2017.

[21] OpenStreetMap Wiki zoom levels. `http://wiki.openstreetmap.org/wiki/Zoom_levels`. Accessed: 2017-06-15.

[22] OpenStreetMap Wiki slippy map tilenames. `http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames#Resolution_and_Scale`. Accessed: 2017-06-15.

[23] Jochen Topf. Github syntactically valid OSM data files. 1-04-2017.

APÈNDIX

B.1 Diagrama de Gantt

B.2 Fórmules

Conversió entre sistemes de coordenades geogràfiques.

- De longitud i latitud (en graus) a Slippy tiles:

  1. Fórmules matemàtiques.

$$x(lon, zoom) = \frac{lon + 180}{360} \cdot 2^{zoom} \tag{4}$$

$$y(lat,zoom) = \left(1 - \frac{\ln\left(\tan\left(lat\frac{\pi}{180}\right) + \frac{1}{\cos\left(lat \cdot \frac{\pi}{180}\right)}\right)}{\pi} \right) \cdot 2^{zoom-1} \tag{5}$$

  2. Pseudocodi.

$$n = 2^zoom$$
$$x = n * ((lon_deg + 180)/360)$$
$$y = n * (1 - (log(tan(lat_rad) + sec(lat_rad))/\pi))/2$$

- De Slippy tiles a longitud i latitud (en graus):

  1. Fórmules matemàtiques.

$$lon = \frac{x}{2^z} \cdot 360 - 180 \tag{6}$$

$$lat = \arctan\left(\sinh\left(\pi - \frac{y}{2^z}\right)\right) \cdot \frac{180}{\pi} \tag{7}$$

  2. Pseudocodi.

$$n = 2^zoom$$
$$lon\_deg = xtile/n * 360.0 - 180.0$$
$$lat\_rad = arctan(sinh(\pi * (1 - 2 * ytile/n)))$$
$$lat\_deg = lat\_rad * 180.0/\pi$$