

Entorn de simulació per la captura d'imatges des de dron

Kevin Martín Fernández

Resum– Resum

Paraules clau– Simulador, Drons, Terreny, Satelite, Multiespectre, Unreal Engine

1 INTRODUCCIÓ

La simulació d'espais per a la generació d'imatges artificials és un àmbit en el qual es busca representar el món real de la forma més realista possible per tal de poder generar informació que en entorns reals ens representaria un perill o un cost econòmic elevat.

En aquest treball es vol aconseguir que mitjançant dades de mapes d'elevacions i imatges aèries de terrenys reals generar-ho en un entorn simulat per tal de poder generar imatges fictícies amb la màxima realitat possible per tal de generar datasets d'imatges per la utilització en diversos camps de l'aprenentatge computacional.

2 OBJECTIUS

En aquest apartat determinarem els diferents objectius del projecte en format de jerarquia per tal de veure la dependència entre els diferents objectius:

1. Analitzar
2. Definir
 - 2.1. Definir mòduls pel projecte
 - 2.2. Definir l'estructura del software
 - 2.3. Definir plataformes utilitzades
 - 2.3.1. Definir els mòduls a desenvolupar
 - 2.3.2. Definir la comunicació entre els mòduls
 - 2.3.3. Definir estructura de les dades que rebrà Unreal Engine
3. Desenvolupar
 - 3.1. Desenvolupar mòdul de transformació i obtenció de dades
 - 3.2. Desenvolupar mòdul gràfic (Unreal Engine)
 - 3.2.1. Desenvolupar la interfície del menú
 - 3.2.2. Desenvolupar la lògica del vehicle
 - 3.2.3. Desenvolupar codi per la carrega de terreny y material

- E-mail de contacte: kevinmf94@gmail.com
- Menció realizada: Enginyeria de Computació
- Treball tutoritzat per: Felipe Lumbreras Ruiz
- Curs 2018/19

- 3.2.4. Desenvolupar llibreria RPC per el control del entorn
- 3.3. Desenvolupar mòdul de scripting
 - 3.3.1. Desenvolupament client que controlarà el vehicle
- 3.4. Integrar els mòduls d'AirSim en Unreal Engine
 - 3.4.1. Integrar mòdul de Segmentació en el projecte
- 3.5. Desenvolupar altres capes d'informació
4. Testear
 - 4.1. Fer provés del mòdul de transformació de dades
 - 4.2. Fer provés del mòdul gràfic
 - 4.3. Fer provés del mòdul de control per scripting
 - 4.3.1. Elaborar script d'exemple
 - 4.3.2. Provar script d'exemple
5. Documentar
 - 5.1. Redactar informe inicial
 - 5.2. Redactar informe de seguiment I
 - 5.3. Redactar informe de seguiment II
 - 5.4. Redactar l'informe final
 - 5.5. Elaborar proposta de presentació
 - 5.6. Elaborar pòster
 - 5.7. Gestionar la documentació del dossier

3 METODOLOGIA

En aquest projecte s'ha decidit utilitzar una metodologia de tipus Agile[1], ja que això ens permetrà identificar d'una forma millor les petites parts de les quals es compon aquest projecte, a més a més d'adaptar-se als canvis imprevistos. En concret s'ha escollit la tècnica Kanban[2] que consisteix en organitzar el nostre backlog (tasques de curta duració) en targetes que ficarem en un taulell segons en quin punt del cicle de vida de la tasca es trobi (pendent, començada, ...) per això s'ha decidit utilitzar l'eina Trello[3] que permet de forma visual crear targetes i moure-les entre les diferents llistes.

3.1 Diagrama de Gant

Per tal de gestionar el projecte també s'ha empleat un diagrama de Gant elaborat amb excel. En aquest diagrama contemplarem les diverses tasques i subtasques per tal de fer una previsió del treball a realitzar, el temps aproximat que trigarem per cada tasca, això ens permetrà fer una aproximació del temps que ens comportarà el projecte d'aquesta forma podrem determinar la viabilitat.

4 ESTAT DE L'ART

Actualment existeixen diverses aplicacions per la generació d'imatges en entorns que simulen la realitat amb la finalitat de generar dades per algoritmes d'aprenentatge. En aquest àmbit unes de les més importants és AirSim[4] desenvolupat per Microsoft i Carla SIMULATOR[5] desenvolupat pel centre de visió per computador, també s'analitzarà d'altres com poden ser LESS[6], DIRSIG[7] o Google Earth Engine[8] d'àmbit més específic.

4.1 AirSim

AirSim és un simulador gràfic elaborat en Unreal Engine[9] aquest simulador té la finalitat de generar imatges en un entorn totalment fictici, incorpora diversos mòduls que ens ofereix les següents funcionalitats (podem veure un exemple a la figura 1):

- Simulació de cotxes
- Simulació de drons
- Compatibilitat amb controladors reals de drons
- Gravació
- Vista de mapa de profunditats
- Vista segmentada
- Efectes de pluja
- Control d'il·luminació segons l'hora diària
- Control dels vehicles mitjançant scripts en python



Fig. 1: Simulador Airsim

4.2 Carla SIMULATOR

Carla és un simulador gràfic elaborat en Unreal Engine aquest simulador té la finalitat de generar imatges en entorn fictici amb la màxima realitat possible per generar imatges

que serveixin per a l'aprenentatge de xarxes neuronals capaces de conduir un cotxe autònom de forma segura tenint en compte els casos poc probables que no es podrien generar en un entorn físic. Aquest entorn compte amb les següents funcionalitats:

- Simulació de cotxes
- Vista de mapa de profunditats
- Vista segmentada
- Simulació de tràfic
- Control dels actors amb scripts de python

4.3 LESS

LESS és un model de la redacció (com podem veure en la figura 2) efectuada en un objecte/terreny tridimensional per diferents raigs, generat a partir de tècniques de ray-tracing ens permet simular dades i imatges sobre escenes 3D realistes. Aquest model implementa un mètode de seguiment de fotons ponderats per simular el factor de reflectància bidireccional multi-espectral.

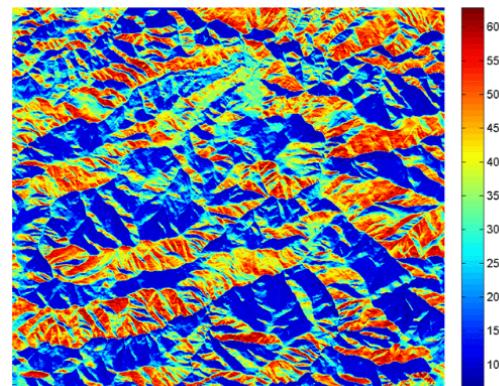


Fig. 2: Exemple de resultats de radiació en un terreny

4.4 DIRSIG

El model Digital Imaging i Remote Sensing Image Generation (DIRSIG) és un model de generació d'imatges sintètiques desenvolupat pel laboratori de Digital Imaging i Remote Sensing del Rochester Institute of Technology. El model pot produir imatges d'una sola banda, multiespectrals o hiperespectrals des del visible a través de la regió tèrmica d'infrarojos de l'espectre electromagnètic. Podem veure un exemple generat per DIRSIG en la figura 3.

4.5 Google Earth Engine

Google Earth Engine és un projecte de Google dedicat a oferir les eines necessàries per tal de poder analitzar i visualitzar dades geospatials destinat a estudis acadèmics, institucions sense ànim de lucre, empreses i governs. Les característiques principals Google Earth Engine són:

- Es pot treballar amb datasets de diferents satèl·lits com són LANDSAT, MODIS, SENTINEL, etc.



Fig. 3: Frame del port de tacoma

- Incorpora un entorn de treball per manipular la informació amb una extensa API que ens permet combinar imatges de diferents espectres, visualitzar-ho en el mapa del món, exportar-ho a Google Drive entre altres.

5 ESTRUCTURA DEL PROJECTE

Per tal de determinar l'estructura del nostre projecte s'han estudiat vàries alternatives vistes a l'apartat 4 en les quals es pot veure com estan organitzats altres projectes similars. En aquest apartat veurem l'estructura de diferents projectes amb l'objectiu de decidir l'estructura del projecte i diferents llibreries.

5.1 AirSim

AirSim està compost per múltiples mòduls escrits en diversos llenguatges com es pot veure a continuació:

- **AirLib (C++)**: Mòdul per a Unreal Engine que proporciona les classes bàsiques per comunicar-se mitjançant el protocol RCP i control dels vehicles simulats.
- **DroneServer (C++)**: Servidor per rebre ordres de Dron mitjançant RCP.
- **DroneShell (C++)**: Client de consola per enviar ordres al mòdul de dron.
- **PythonClient (Python)**: Client que envia ordres mitjançant RCP, també incorpora codi per al tractament, segmentació d'imatges, etc.
- **SGM (C++)**: Codi per tractar imatges i generar les vistes segmentada i stereo.
- **Unity (C# i C++)**: Demostració en el motor gràfic Unity, incorpora una sèrie de mòduls per Unity per visualitzar la informació d'AirSim.
- **Unreal Engine (C++)**: Demostració en el motor gràfic Unreal Engine, incorpora una sèrie de mòduls per Unreal Engine per visualitzar la informació d'AirSim.

5.2 Carla SIMULATOR

Carla SIMULATOR està compost per múltiples mòduls com podem veure en la figura 4 escrits en diversos llenyuatges com es pot veure a continuació:

- **LibCarla (C++)**: Llibreria principal de Carla

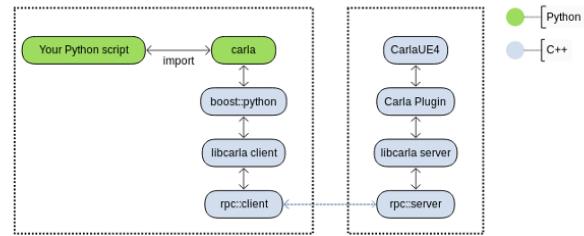


Fig. 4: Relació entre els mòduls de Carla

- **Unreal (C++)**: Motor gràfic amb el plugin de carla, que incorpora totes les funcionalitats afegides a Unreal.
- **PythonAPI (Python)**: API que ens permetrà enviar comandes al mòdul de Carla que fa de servidor, aquest api serveix per crear scripts propis.

5.3 Estructura escollida

Analitzant diversos projectes de característiques similars s'ha decidit per una estructura pròpia com podrem veure en la figura 5 podent reutilitzar alguns dels petits mòduls open-source d'altres projectes. S'ha pres aquesta decisió a causa del fet que els altres projectes es basen en la creació terrenys predefinits mitjançant l'entorn d'Unreal, contrari a la finalitat d'aquest projecte en el qual es volen elaborar de forma automàtica terrenys a partir de dades reals.

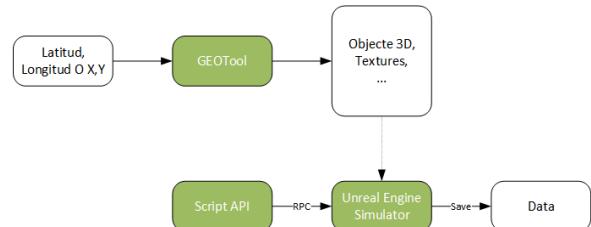


Fig. 5: Estructura DronSimulator

En el nostre projecte constarà d'aquests mòduls:

- **GEOTool (Python)**: Aquest mòdul té la finalitat de donar les eines necessàries per a l'obtenció i adaptació de les dades proveïdes per webservices estàndard en l'àmbit de l'obtenció de dades geogràfiques amb l'objectiu de poder importar-les en qualsevol motor gràfic, podent generar diferents capes d'informació per als terrenys descarregats.
- **ScriptAPI (Python)**: API que ens permetrà fer scripts en Python per tal de controlar la càmera en l'entorn gràfic. D'aquesta forma també ens permetrà l'obtenció d'imatges en l'entorn Unreal Engine.
- **Unreal Engine Simulator (C++)**: Incorporarà tot el motor gràfic i connectors necessaris per a la comunicació amb els fitxers generats amb el mòdul GEOTool, visualització de dades, obtenció d'imatge, etc.

6 MÒDUL GEOTOOLS

Mòdul generat en Python amb l'objectiu d'obtenir informació de mapes d'elevacions, ortofotos, etc. Amb l'objectiu posterior de fer un tractament d'aquestes dades per la generació de terrenys tridimensional i informació en format visible per tal de poder visualitzar-ho en un motor gràfic de tipus Unreal, Unity, OpenGL, etc.

6.1 Obtenció de dades

Amb l'objectiu d'obtenir dades geogràfiques s'ha optat per la comunicació amb els estàndards proposats per l'Open Geospatial Consortium[10]: Web Map Service[11] encarregat de posar a disposibilitat dades d'imatge com poden ser ortofotos d'una zona geogràfica seleccionada i Web Coverage Service[12] encarregat de retornar informació referent a les elevacions de terreny en una zona geogràfica concreta. Per tal d'obtenir dades es fan peticions HTTP a les direccions web oferides per distintes institucions que segueixen els estàndard anomenats, en aquest cas s'ha realitzat proves amb l'Institut Cartogràfic i Geològic de Catalunya[13].

6.1.1 Fitxer de configuració

Per tal de determinar quines dades volem obtenir i de quins webservices l'aplicació accepta per paràmetre un fitxer de configuració en format JSON que ens permet determinar diferents propietats de les dades que demanarem com es pot veure en l'apèndix A.1. En aquest fitxer es pot configurar els següents paràmetres:

- **Type:** Fa referència al tipus de coordenades que li passarem, pot ser latlong o xy en el primer cas farà la corresponent transformació al format UTM (xy)
- **Coordinates:** Coordenades sobre les quals volem fer la petició en el format indicat en el camp type. Si s'escau "xy" es definirà els atributs x, y en cas d'escollir "latlong" definirem els atributs lat, long.
- **Dimensions:** En aquesta secció escollirem les dimensions que volem que es demanin en les peticions per tal de mantenir la mateixa zona geogràfica.
 - **Bbox:** Aquestes seran les dimensions de la zona que volem obtenir amb les que es calcularà els límits que delimitaran la zona.
 - **Texture:** Aquesta serà la resolució que obtindrem per les diferents textures.
- **cellsize:** Mida de cada pixel en metres.
- **meshStep:** Número que indica la quantitat de punts que desitgem saltar a la malla (Per defecte 1). Més informació a la secció 6.5.
- **Wcsurl:** URL al webservice que ens donarà les dades d'altures.
- **Outputwcs:** Nom de sortida del fitxer generat per les altures
- **Formatwcs:** Format del fitxer generat per les altures. Disponibles: raw, obj (Objecte 3D)

- **Wmsrequests:** Array amb cada una de les peticions que farem a diferents WMS per tal d'obtenir varíes imatges

- **Url:** URL al webservice WMS.
- **Layers:** capa o capes que volem obtenir d'aquests webservice.
- **Output:** Nom del fitxer de sortida.
- **OutputFormat:** Format del fitxer de sortida. Disponibles: JPG

- **offset:** Objecte de dos components que ens indica el desplaçament en píxels aplicat a les coordenades.

6.2 Estructura de GEOTool

TODO

6.3 Generació de Terrenys a partir dun mapa d'altures

En aquesta secció s'explicarà les diverses formes que s'han optat per generar terrenys que puguin ser interpretats en diferents motors gràfics.

6.3.1 Format RAW

El format RAW és un format pla que es basa en guarda en valors de 16 Bytes totes les altures en format binari tenim com a referència del mar el valor 128, i afegits en el fitxer un darrere de l'altre. Aquest format és acceptat per al creador de terrenys d'Unreal i Unity, però té certes limitacions de dimensions que s'han de complir especificades a l'editor en crear el terreny que fa que es perdi el control de la malla generada, les coordenades de textura no coincideixen amb la textura que es vol aplicar al terreny. Motius pels quals s'ha optat per afegir la generació de l'objecte 3D en format estàndard definit nosaltres l'objecte com es podrà veure en l'apartat 6.3.2, ja que aquestes limitacions fan que es facin varíes adaptacions no estàndards perquè es pugui visualitzar correctament.

6.3.2 Generació de malla 3D

Per tal d'importar terrenys en els motors gràfics s'ha optat per generar una malla 3D en format Wavefront obj[14] format compatible amb qualsevol editor 3D, motor gràfic, etc. Aquest format dóna la llibertat per tal de controla la distància entre els vèrtexs, on s'aplicarà la textura i quines seran les normals dels vèrtexs fent que el terreny sigui suavitzat.

Com que el tractament amb bucles és lent s'ha realitzat tots els càlculs amb la llibreria NumPy aproveitant l'eficiència que incorpora aquesta llibreria amb el càlcul de matrius pel qual s'ha adaptat el problema com podem veure en el codi disponible en l'annex A.2.

Per la generació d'objectes cal definir 4 tipus d'objectes:

- **Vèrtex:** Són cada un dels punts en el món, van definits pels índexs segons llegui'm la graella d'elevacions, es

multipliquen per un K (Distància entre els vèrtexs segons la distància que ens indiqui el mapa d'elevacions obtingut).

- **Vèrtex de textura:** Vèrtex amb dos components x, y compresos entre el 0 i 1 que indiquen la correspondència entre els punts d'una textura i la malla en la qual es vol aplicar aquella textura. Aquestes propietats es calcularien amb les equacions 1 i 2.

$$u = f(\text{columna}) = \text{columna}/(\text{ample} - 1) \quad (1)$$

$$v = f(\text{fila}) = 1 - (\text{fila}/(\text{altura} - 1)) \quad (2)$$

- **Normal del vèrtex:** Vectors que ens indica la direcció en la qual es reflecteix la llum per a cada vèrtex de l'objecte. Per tal de calcular aquestes normals cal el pas previ de calcular les normals de cada cara, aquestes no seran introduïdes al fitxer final, ja que aquestes les genera'n els motors per defecte segons l'ordre en el qual indiquem els vèrtexs de les cares com es veurà més endavant.

- Generació de normals de cares: Per tal de generar les normals d'una cara un cop sapiguem la relació entre les cares se seguirà el patró vist en la figura 6 on seguirem l'equació 1 per al càlcul de la normal de la cara. \vec{A} i, realitzarem el producte vectorial $\vec{C} = \vec{B} * \vec{A}$ i per últim normalitzarem el vector $\vec{\text{Normal}} = \frac{\vec{C}}{|\vec{C}|}$.
- Generació de normals en els vèrtexs: Per tal de generar el vector normal per a cada vèrtex utilitzarem l'estructura que es pot veure en la figura 7 aplicant la següent formula a cada vèrtex on V correspon als vèrtexs i F a les cares de la figura 7:

$$\vec{\text{Normal}}_V = \vec{F}_1 + \vec{F}_2 + \vec{F}_3 + \vec{F}_4 + \vec{F}_5 + \vec{F}_6 \quad (3)$$

$$\vec{\text{Normal}}_V = \frac{\vec{\text{Normal}}_V}{|\vec{\text{Normal}}_V|} \quad (4)$$

- **Cares:** En aquest punt determinarem quina és la unió dels vèrtexs per tal de generar les diferents cares de la malla, en aquesta implementació s'ha decidit per fer triangulació, és a dir, per cada quadrat de la nostra malla generarem 2 cares triangulars. És important generar les cares mantenint l'ordre dels vèrtexs contrari a les agulles del rellotge, d'aquesta forma els motors gràfics determina'n que la normal de la cara apuntarà cap dalt visualitzant correctament la malla 3D.

Un cop realitzat el procés de generació l'aplicació haurà generat una malla que podem obrir en qualsevol editor com podem veure en la figura 8

6.3.3 GEOJson associat al terreny

Per tal de poder localitzar el terreny que hem generat en altres aplicatius que treballin amb dades geoespaciais o en el mateix simulador d'Unreal s'hi ha incorporat la generació d'un GEOJson que ens indica la zona al qual pertany el fitxer generat. Podem veure un exemple de GEOJson a l'annexa A.3.

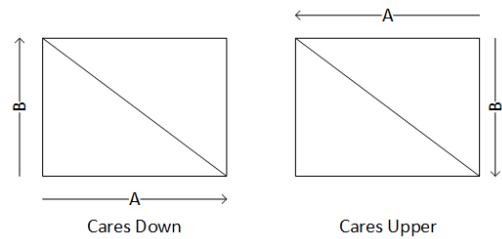


Fig. 6: Patró per càlcul de normals en les cares

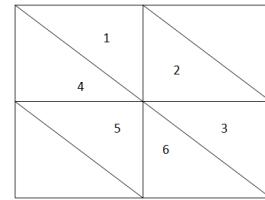


Fig. 7: Patró per càlcul de normals en un vèrtex

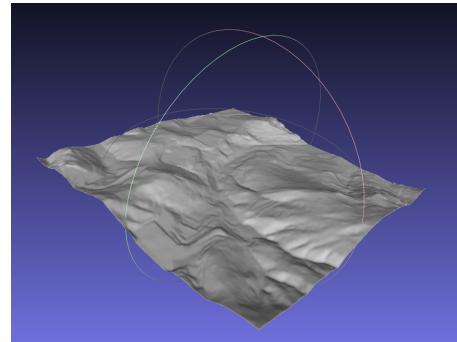


Fig. 8: Malla d'un terreny visualitzada en l'aplicació MeshLab

6.4 Temps de generació

En aquest apartat s'analitzaran les dues versions implementades i es podrà veure les diferencies en temps de generació. Aquest és un punt important per futures implementacions en les quals es desitjaria implementar un visor en temps real del món carregant nova informació segons l'usuari es mogui pel món.

En la figura 9 veiem els temps per la versió amb bucles for i la versió implementada amb NumPy, com es pot veure quant realitzem la implementació utilitzant la llibreria NumPy s'aconsegueix reduir el temps en 24x per la mida 500*500 gran això és degut al fet que NumPy està optimitzada per a paral·lelitzar el tractament de dades de forma vectorial.

6.5 Qualitat

En aquest apartat es mirarà la forma en la qual reduïm la qualitat del terreny i veurem els resultats obtinguts tant qualitativament (diferència visual) com quantitativament (mida en bytes).

Per tal de fer aquesta reducció fem salts de la mida que volem reduir (2,4,8,16,...) en la nostra matriu de punts. Les proves s'han fet amb un terreny de 300x300 pixels.

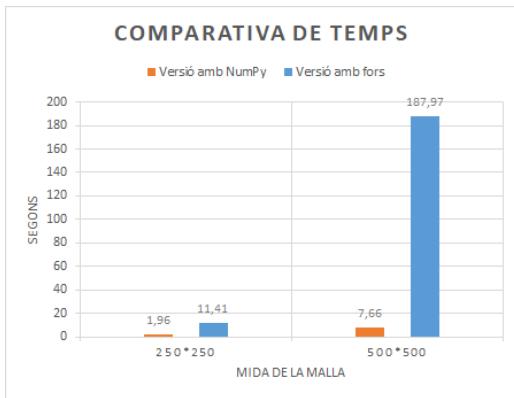


Fig. 9: Gràfic amb el temps de generació d'una malla segons la mida

Com podem veure en la figura 10 al reduir la quantitat de punts reduïm de forma exponencial la mida que ocupa en disc dur fent més lleugera la càrrega dels fitxers per l'entorn d'Unreal.

Com podem veure en la figura 11 es pot visualitzar la pèrdua de qualitat agafant com a referència la muntanya del fons en la que veiem com es perd la definició en el pic. Podem considerar visualment que quant configurem "meshStep" en 1 i 2 la pèrdua qualitativa no és apreciable, a partir de 4 es comença a apreciar lleugerament, finalment en els nivells 8 i 16 es pot veure la major pèrdua en la que es pot apreciar els pics en forma de punxa en compte de la serralada que es veu amb el step més baix.

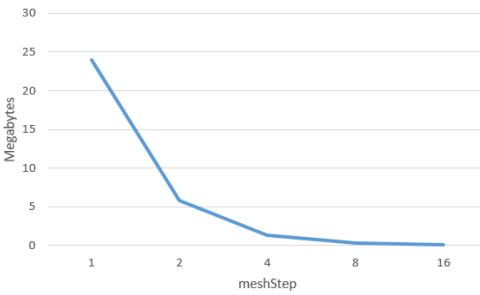


Fig. 10: Gràfic de mida en disc dur dels fitxers .obj per un terreny de 300x300

7 MÒDUL: SIMULADOR

En aquesta secció explicarem els diferents punts dels quals es compon el mòdul de simulació amb el que se cerca l'objectiu de posar a disposició una eina que ens permeti simular un viatge amb un vehicle genèric per sobre d'un terreny obtenint imatges des de diferents càmeres configurades anteriorment.

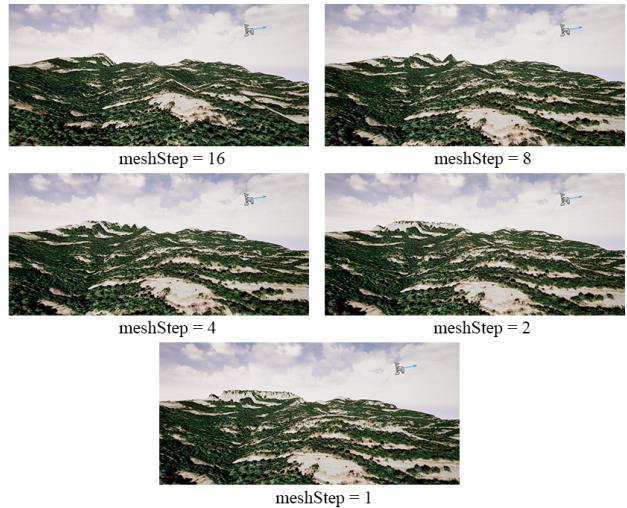


Fig. 11: Comparativa visual dels efectes produits pel nivell de qualitat

7.1 Visualització i control del Vehicle

Aquesta part serà l'encarregada de donar la interfície per moure un vehicle genèric pel nostre món simulat, aquest vehicle es configurarà amb una sèrie de càmeres que es podran visualitzar en la pantalla i generar imatges de cada una d'elles afegint la possibilitat de obtenir múltiples vistes d'una mateixa localització. Podem veure una vista zenital en la figura 12.

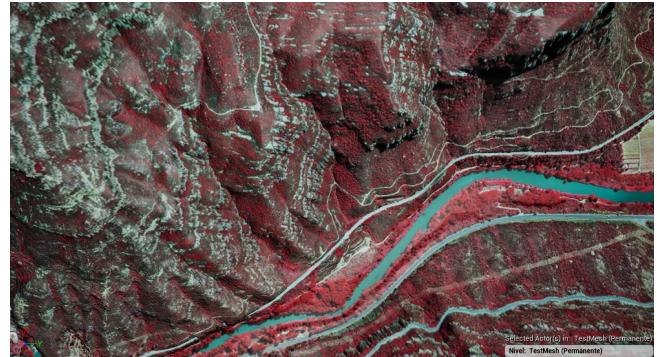


Fig. 12: Vista zenital infraroja de Montserrat

7.1.1 Control remot

Per tal de controlar remotament el vehicle s'implementa una comunicació utilitzant el protocol RPC (Remote Procedure Call) en concret s'utilitza la llibreria Rpclib[15] per a C++. Aquest servidor s'implementarà en el mòdul d'Unreal, la gestió del servidor RPC es fa mitjançant una classe (AVehiclePawn) que podem heretar de forma que permetrà donar la interfície necessària per a controlar un Vehicle en concret es podran realitzar les següents accions:

- Inicialitzar/parar el servidor RPC: Inicialitzem el servidor prement la tela Y al simulador d'Unreal i l'aturarem amb la tecla U.
- Canviar la posició del vehicle
- Canviar la rotació

- Demanar que es generin imatges d'una de les càmeres

L'assignació de les funcionalitats es fa en la funció "void BindFunctions(rpc::server* server)" la qual rep la instància del servidor, aquesta funció es pot sobreescrivir per afegir posteriorment en classes heretades més funcionalitat com es pot veure en l'annexa A.4. Per defecte la classe AVehiclePawn ens permet cridar a les següents funcions RPC:

- setLocation(double x, double y, double z): Enviem la localització en la qual volem que és situí el nostre vehicle.
- setRotation(double x, double y, double z): Enviem el vector amb la posició en el món que volem visualitzar, d'aquesta forma calcularem en quin punt del simulador es ha de mantenir la vista de la càmera.
- setLocationAndRotation(double x, double y, double z, double lx, double ly, double lz): Funció que crida d'un a les dues anteriors.
- getImage(int idCamera, std::string path): Ens permet indicar de quina càmera i on volem guardar la imatge.

7.1.2 Visualització del cel

Per tal d'implementar el cel hem utilitzat un modul natiu d'Unreal que ens permet generar una sphere en la que es renderitza un cel aquest cel té diversos paràmetres ajustables que ens permet determinar la posició del sol en aquell moment, el nivell de brillantor de les estrelles, la quantitat de núvols, etc.

Això dona la possibilitat de generar imatges sintètiques en diferents moments del dia amb diferents il·luminacions, el que permetrà generar datasets més complexos. D'altra banda això comportarà que haurem d'aplicar textures senseombres per tal que aquestes siguin generades per al motor d'Unreal. Podem veure un exemple de diferents hores del dia a la figura 13.

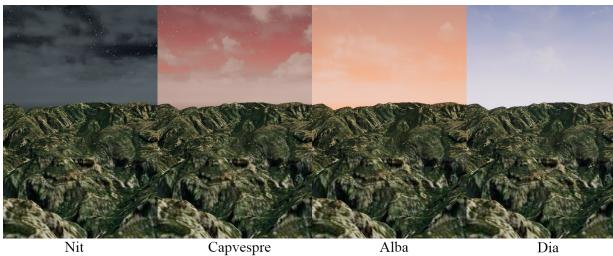


Fig. 13: Exemple de céls

7.2 Terreny

El terreny és carregat mitjançant codi obtenint el fitxer .obj i processant aquest fitxer per tal de generar una malla en temps d'execució generant un UProceduralMesh[16] d'Unreal, per tal de realitzar aquesta tasca es té en compte que Unreal treballa amb un sistema de coordenades invertit respecte al model UTM en el qual l'eix Y es troba invertit a conseqüència d'això es ha de realitzar la reflexió en l'eix Y els objectes fent la correspondència que

pertoqui en cada moment entre el món real i el món simulat.

Aquesta malla és carregada al món i representada per la classe "MapChunk" al qual se li assignarà el material dinàmic que contindrà les textures carregades de les diferents visualitzacions que desitgem podent visualitzar qualsevol de les imatges que preparem en format imatge per al terreny com poden ser textures RGB, Infraroig, multi espectral, etc.

7.3 Visualització del terreny

En aquest apartat veurem els resultats obtinguts a partir de terrenys 3D generats pel mòdul GEOTool i l'aplicarem diverses visualitzacions de fonts com són textures o shaders obtinguts per diversos medis, en concret les vistes RGB i infraroig són obtingudes de l'Institut Cartogràfic de Catalunya, les dades multiespectrals les obtenim de l'explorador EO Brower[17] i la informació de profunditat l'obtenim d'aplicar un shader a l'escena.

Com podem veure en la figura 14 hem aplicat a un terreny corresponent a "Sales de Pallars" diferents bandes multiespectrals obtingudes pel satèl·lit Sentinel 2[18]. Les tres primeres bandes B02, B03 i B04 corresponen a bandes de colors, la banda B05 correspon al canvi ràpid de reflectància que fa la vegetació en un rang proper a l'infraroig, la banda B08 obté informació NIR[19] i per últim veiem la banda B09 capaç de detectar vapors d'aigua. D'aquesta forma amb aquestes bandes podem obtenir informació composta per diversos espectres que podem aplicar a diversos àmbits. En la figura 15 es poden veure alguns exemples, entre ells podem veure índexs com són:

- NDVI[20]: Basat en la combinació de les bandes (B08 - B04)/(B08 + B04), índex utilitzat per veure l'estat del conreu.
- Moisture index[21]: Basat en la combinació de les bandes (B8A - B11)/(B8A + B11), indica la proporció de precipitació que es necessita per tal de satisfer les necessitats de la vegetació.
- NDWI[22]: Basat en la combinació de les bandes (B03 - B08)/(B03 + B08), índex que s'utilitza per determinar l'estrés hídric de la vegetació, saturació de la humitat en el terra o realitzar delimitacions de masses d'aigua com a llacs o embassaments.

En l'última vista de la figura 15 es pot veure la de profunditat o proximitat utilitzada en l'àmbit de la visió per computador per introduir aplicacions". TODO

7.4 Interacció amb el simulador TODO

8 MÒDUL D'SCRIPTING

En aquest apartat explicarem i veurem els principals punts del mòdul d'scripting en el qual podrem gestionar el simulador de forma que podrem recrear viatges llegint fitxers de trajectòries, generar imatges, generar soroll a les trajectòries, etc.

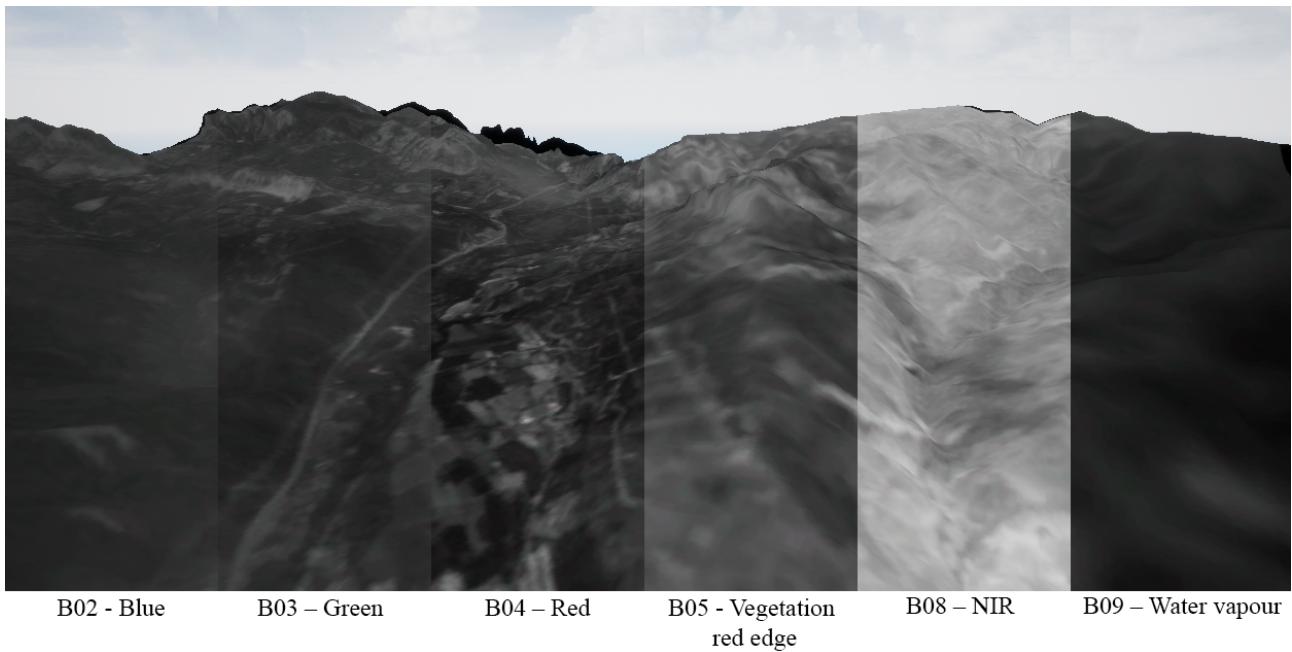


Fig. 14: Vista de Sales de Pallars en 6 bandes diferents

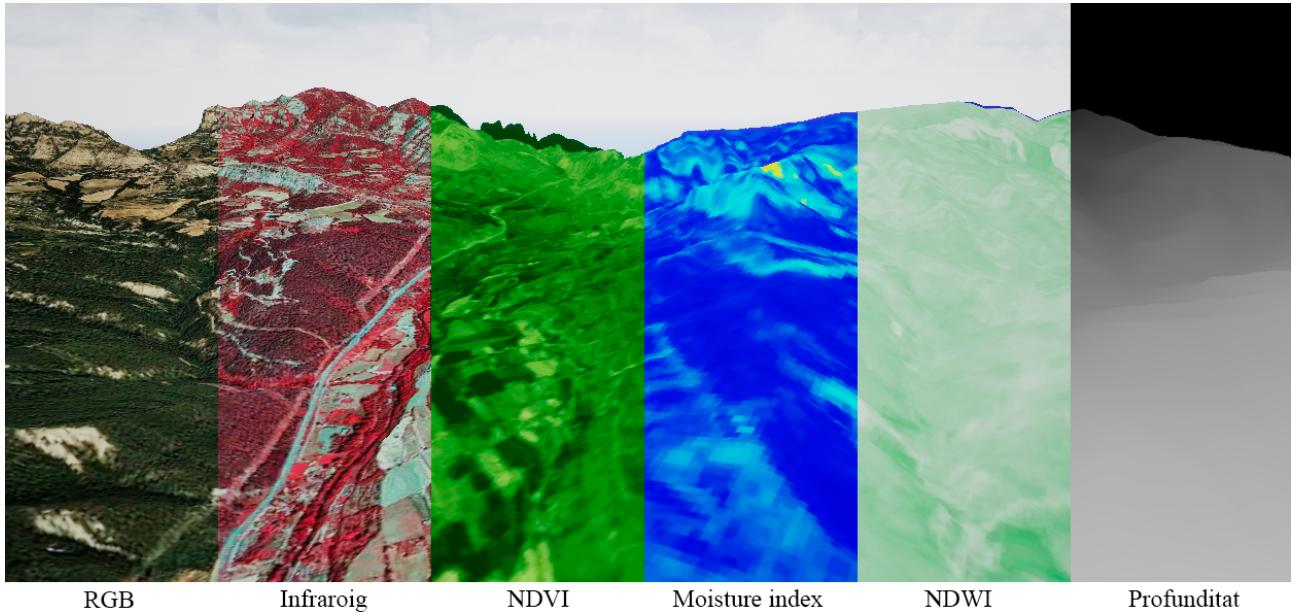


Fig. 15: Vista de Sales de Pallars en RGB, Infraroig, NDVI, Moisture, NDWI i profunditat

8.1 Trajectòries

8.1.1 Fitxer de trajectòries

8.2 Simulació de soroll TODO

8.3 Generació de Datasets TODO

9 CONCLUSIONS

AGRAÏMENTS

REFERÈNCIES

- [1] Agile software development
https://en.wikipedia.org/wiki/Agile_software_development [19/02/2019]
- [2] Kanban
<https://www.iebschool.com/blog/metodologia-kanban-agile-scrum/> [19/02/2019]
- [3] Trello - <https://trello.com/> [19/02/2019]
- [4] AirSim - <https://github.com/Microsoft/AirSim> [19/02/2019]
- [5] Carla SIMULATOR - <http://carla.org> [19/02/2019]

- [6] LESS - <http://lessrt.org/> [11/04/2019]
- [7] Digital Imaging and Remote Sensing Image Generation - <http://dirsig.org/> [11/04/2019]
- [8] Google Earth Engine - <https://earthengine.google.com/> [20/05/2019]
- [9] Unreal Engine - <https://www.unrealengine.com/en-US/what-is-unreal-engine-4> [09/03/2019]
- [10] Open Geospatial Consortium (OGC) - <http://www.opengeospatial.org/> [08/04/2019]
- [11] Web Map Service (WMS) - <https://www.opengeospatial.org/standards/wms> [08/04/2019]
- [12] Web Coverage Service (WCS) - <https://www.opengeospatial.org/standards/wcs> [08/04/2019]
- [13] Institut Cartogràfic i Geològic de Catalunya (ICGC) - <http://www.icgc.cat/ca/> [08/04/2019]
- [14] Wavefront .obj file - https://en.wikipedia.org/wiki/Wavefront_.obj_file [09/04/2019]
- [15] RPC Lib - Modern msgpack-rpc for C++ - <http://rpclib.net/> [10/04/2019]
- [16] Procedural Mesh Component - https://wiki.unrealengine.com/Procedural_Mesh_Component_in_C%2B%2B:Getting_Started [21/05/2019]
- [17] EO Browser - <https://apps.sentinel-hub.com/eo-browser/> [25/05/2019]
- [18] Sentinel 2 - https://www.esa.int/esl/ESA_in_your_country/Spain/SENTINEL_2 [25/05/2019]
- [19] Tecnología NIR, sus Usos y Aplicaciones - <https://www.engormix.com/balanceados/articulos/tecnologia-nir-sus-usos-t32534.htm> [25/05/2019]
- [20] El NDVI o Índice de vegetación de diferencia normalizada - <https://geoinnova.org/blog-territorio/ndvi-indice-vegetacion/> [25/05/2019]
- [21] Moisture index - http://glossary.ametsoc.org/wiki/Moisture_index [25/05/2019]
- [22] Cálculo del índice NDWI - <http://www.gisandbeers.com/calcular-del-indice-ndwi-diferencial-de-agua-y-no-paisaje/> [25/05/2019]

APÈNDIX

A.1 JSON d'exemple per la configuració de GEOTool

```
{
  "type": "xy",
  "coordinates": {
    "x": 413894.13,
    "y": 4610649.22,
    "zone": "31T"
  },
  "dimensions": {
    "bbox": {
      "height": 300,
      "width": 300
    },
    "texture": {
      "height": 1500,
      "width": 1500
    }
  },
  "cellsize": 5,
  "meshQuality": 3,
  "wcsUrl": "http://geoserveis.icc.cat/icc_mdt/wcs/service",
  "outputWcs": "example9x9q3/outputwcs02",
  "formatWcs": "obj",
  "wmsRequests": [
    {
      "url": "http://geoserveis.icc.cat/icc_ortohistorica/wms/service",
      "layers": "orto25c2016",
      "output": "example9x9q3/outputwcs02",
      "outputFormat": "jpg"
    },
    {
      "url": "http://geoserveis.icc.cat/icc_ortohistorica/wms/service",
      "layers": "ortoi25c2016",
      "output": "example9x9q3/outputwmsi02",
      "outputFormat": "jpg"
    }
  ],
  "offset": {
    "x": 300,
    "y": 300
  }
}
```

A.2 Codi per la generació d'una malla 3D a partir d'un fitxer d'altures

```
def generate_obj(values, k=5):
    h, w = values.shape
    ij = np.meshgrid(np.arange(h), np.arange(w), indexing='ij')
```

```

i = ij[0].reshape(h*w)
j = ij[1].reshape(h*w)
values = values.reshape(h*w)

# Generate list of vertex
vertex = np.zeros((h*w, 3))
vertex[:, 0] = i*k
vertex[:, 1] = j*k
vertex[:, 2] = values

# Generate faces
mask = np.logical_and(i < (h-1), j
                      < (w-1))
uFaces = np.zeros((h*w, 3), dtype=
                  np.int32)
indexVertex = np.arange(h*w)+1

uFaces[mask, 0] = indexVertex[mask]
uFaces[mask, 1] = indexVertex[mask]
    + w
uFaces[mask, 2] = indexVertex[mask]
    + w + 1
uFaces = uFaces[mask]

dFaces = np.zeros((h * w, 3), dtype
                  =np.int32)
dFaces[mask, 0] = indexVertex[mask]
    + w + 1
dFaces[mask, 1] = indexVertex[mask]
    + 1
dFaces[mask, 2] = indexVertex[mask]
dFaces = dFaces[mask]

faces = np.concatenate((uFaces,
                       dFaces), 0)

# Generate UV Map
uvMap = np.zeros((h*w, 2))
uvMap[:, 0] = j / (w - 1)
uvMap[:, 1] = 1 - (i / (h - 1))

# Generate normal faces upperFaces
#mask = np.logical_and(i > 0, j < h
-1)
aUpper = (np.roll(vertex, h, axis
                   =0) - vertex)
bUpper = (np.roll(vertex, -1, axis
                   =0) - vertex)

normalUpperFaces = np.cross(bUpper,
                            aUpper)
module = np.linalg.norm(
            normalUpperFaces, axis=1)
normalUpperFaces[:, 0] =
    normalUpperFaces[:, 0] / module
normalUpperFaces[:, 1] =
    normalUpperFaces[:, 1] / module
normalUpperFaces[:, 2] =
    normalUpperFaces[:, 2] / module

# Generate normal faces downFaces
#mask = np.logical_and(i < w-1, j >
0)
aDown = (np.roll(vertex, -h, axis
                   =0) - vertex)
bDown = (np.roll(vertex, 1, axis=0)
                   - vertex)

normalDownFaces = np.cross(bDown,
                           aDown)
module = np.linalg.norm(
            normalDownFaces, axis=1)
normalDownFaces[:, 0] =
    normalDownFaces[:, 0] / module
normalDownFaces[:, 1] =
    normalDownFaces[:, 1] / module
normalDownFaces[:, 2] =
    normalDownFaces[:, 2] / module

# Generate vertex normals
normalVertex = np.zeros((h*w, 3))
mask = np.logical_and(np.
                      logical_and(i > 0, j > 0),
                      np.
                      logical_and(i < w-1, j < h-1))

normalVertex = np.roll(
    normalUpperFaces, 1, axis=0)
normalVertex += normalUpperFaces
normalVertex += np.roll(
    normalUpperFaces, -h, axis=0)
normalVertex += normalDownFaces
normalVertex += np.roll(
    normalDownFaces, h, axis=0)
normalVertex += np.roll(
    normalDownFaces, -1, axis=0)

module = np.linalg.norm(
            normalVertex, axis=1)
valid_modules = module != 0
normalVertex[valid_modules, 0] =
    normalVertex[valid_modules, 0] /
    module[valid_modules]
normalVertex[valid_modules, 1] =
    normalVertex[valid_modules, 1] /
    module[valid_modules]
normalVertex[valid_modules, 2] =
    normalVertex[valid_modules, 2] /
    module[valid_modules]

normalVertex[np.logical_not(mask),
            0] = 0
normalVertex[np.logical_not(mask),
            1] = 0
normalVertex[np.logical_not(mask),
            2] = 1

return vertex, faces, uvMap,
        normalVertex

```

A.3 GEOJson d'exemple

```
{"type": "FeatureCollection", "name": "example9x9q3/outputwcs00", "crs": {"type": "name", "properties": {"name": "urn:ogc:def:crs:EPSG:23031"}}, "features": [{"type": "Feature", "properties": {}, "geometry": {"type": "Polygon", "coordinates": [[[412394.118618708, 4612149.208618707], [412394.118618708, 4613649.2313812915], [413894.1413812921, 4613649.2313812915], [413894.1413812921, 4612149.208618707], [412394.118618708, 4612149.208618707]]}]}]}
```

A.4 Codi d'exemple per estendre la funcionalitat RPC

```
.h:
virtual void BindFunctions(rpc::server* server) override;
```



```
.cpp:
void MyClass::BindFunctions(rpc::server*
    * server)
{
    Super::BindFunctions(server);

    server->bind("nameOfFunction", [
        context_params](Variables ... ) {
            //MyCode
        });
}
```