GEOSpectralSimulator

Kevin Martín Fernández

Abstract– Resum

Keywords– Simulator, Terrain, Satelite, Multiespectre, Unreal Engine

1 INTRODUCTION

The simulation of the real world for the generation of synthetic images is a scope in which you are looking to represent the real world more accurately possible, in this way can generate synthetic information that represents real environments that can represent a danger (for the natural environment, people, etc) or a high economic cost.

On this project you want to get that is able to by means of elevation maps and aerial images of real terrains generates a simulated environment that it can generate its image datasets, which can be used in multiple areas of the computation learning. Also, it can simulate flying, see a geographical area or expand this project in other environments that uses a geographic information. That geographical information it can provided from multiple sources this allows to work with a lot of external data.

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 (Ureal 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
     - 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. Testejar
   - 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 METHODOLOGY

In this project its decide uses a methodology of Agile[1] style, this allows to identify more efficiently the little parts that compose the project, besides adapt to the changes. More concretely has used a technique called Kanban[2] which consists in organizing this backlog (tasks of short duration) on cards that will be placed on a board according to the point of the life cycle on the tasks founds it. For this to be used the Trello[3] software that can see the boards on web browser, create cards and move it between different lists.

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

3.1 Gant Diagram

In order to manage a project you must also include a Gant diagram done with excel. In this diagram several tasks and subtasks are contemplated in order to make a prediction of the work done, the approximate time is what it will take for each task, this will allow an approximation of the time that the project behaves in this way.

4 State of the art

Currently exists diverse applications for the generation of images in simulated environments with the finality of generating data used in machine learning algorithms.
In this scope one of the most import is AirSim[4] developed by Microsoft and Carla SIMULATOR[5] developed by "Centre de Visio per Computador (CVC)", also it's analyse others, how can they be LESS[6], DIRSIG dirsig or Google Earth Engine[8] of more specific scope.

4.1 AirSim

AirSim is a graphic simulator made with Unreal Engine, this simulator has the purpose of generating synthetic images on fake environments, it incorporates multiple modules that are offered the next functionalities (You can see an example in the figure 1):

- Simulation of cars.
- Simulation of drones.
- Compatibility with real drone controllers.
- Recording.
- Depth view.
- Segmentation view.
- Rain effects.
- Control of illumination according to the daily hours.
- Control of vehicles through a python script.

Fig. 1: AirSim simulator

4.2 Carla SIMULATOR

The Carla is a graphical simulator made on the Unreal Engine, this simulator has the finality of generating images on a fake environment with the maximum realism so that possible to generate images that they can serve to a made learning the neural networks able to drive safely a autonomous car taking into the unlikely cases that are difficult to generate in the real world. Its environment has the next functionalities:

- Simulation of cars.
- Depth view.
- Segmentation view.
- Simulation of traffic, pedestrians, etc.
- Control of the actors (traffic, pedestrian, cameras, etc) with a python script.

4.3 LESS

LESS is a model of the radiation (You can see an example, in the figure 2) generated on a three-dimensional object/terrain for different rays, generating from a technique of ray-tracing its able to simulate data and images over realistic three-dimensional scenes. This model implements a method of follow weighted photons for simulating the effect of multi-spectral bidirectional reflectance.

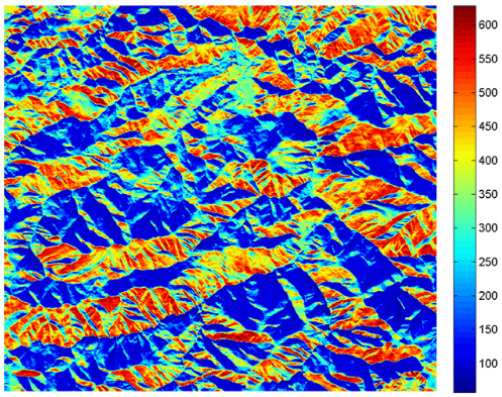


Fig. 2: Example of results of the radiation of a terrain

4.4 DIRSIG

The model of Digital Imaging and Remote Sensing Image Generation (DIRSIG) is a model of generation of synthetic images developed by the lab of Digital Imaging and Remote Sensing of the Rochester Institute of Technology. The model can produce one band images, multispectral or hiperspectral from visible trough of the infrared thermal region of the electromagnetic spectral. You can see an example generated by DIRSIG on the figure 3.

4.5 Google Earth Engine

Google Earth Engine is a project by Google, dedicated to offer the necessary tools to be able to analyse and visualize geospacial data, intended for an academy studies, non-profit institutions, companies and governments. The principal features of the Google Earth Engine are:

- We can work with datasets from different satellites such as LANDSAT, MODIS, SENTINEL, etc.

Fig. 3: Frame from the Tacoma port

- Incorporate a work environment to manipulate the data and wide API that allows us to combine images from different spectral, see it on a world map, export data to Google Drive and more.

5 Structure of project

In order to determine the project structure, it's study the diverse alternatives viewed on section 4. Will be analysed the structure from AirSim and Carla with the objective of deciding the ownership structure and the external libraries to incorporate them into the project.

5.1 AirSim

AirSim is composed of multiple modules written in various languages, as can be seen below:

- **AirLib (C++)**: Module for Unreal Engine that provides the base classes to communicate through the RPC protocol and control the simulated vehicles.
- **DroneServer (C++)**: Server to receive orders from RPC client.
- **DroneShell (C++)**: Shell client to send orders to the Server.
- **PythonClient (Python)**: Client to send orders trough RPC, also includes code to the manipulation of images.
- **SGM (C++)**: Code to manipulate images and generate the segmentation view.
- **Unity (C# i C++)**: Unity version, includes a modules to see the AirSim information.
- **Unreal Engine (C++)**: Unreal version, includes a modules to see the AirSim information.

5.2 Carla SIMULATOR

Carla SIMULATOR is composed for multiple modules as you can see in the figure 4 written in multiple languages. The modules of Carla are:

- **LibCarla (C++)**: The main library of Carla, is in charge of the logic of the simulation.
- **Unreal (C++)**: Graphic engine with the Carla plug-in, this includes all the functionalities added to Unreal.

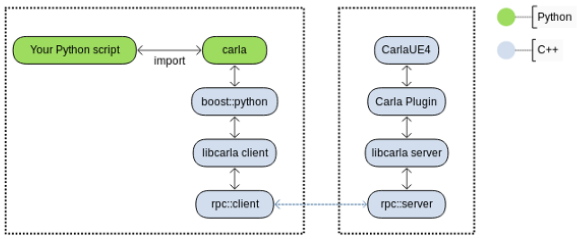


Fig. 4: Relation between the modules of Carla

- **PythonAPI (Python)**: The API allows sending orders to the Carla module that works like server, this API it's useful to make own scripts.

5.3 The structure chosen

Analysing multiple projects with similar features, it's decide for a own structure as we can see in the figure 5 being able to use something modules with other open-source projects. Its make this decision due to the fact that the other projects are based on the creation of terrain predefined on Unreal Editor, the opposite to the finality of this project in which its make terrain automatically provided by real data.

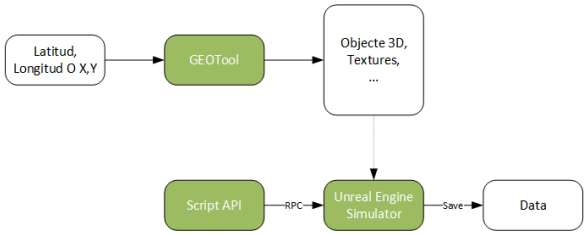


Fig. 5: Structure of GEOSpectralSimulator

Our project will consist of these modules:

- **GEOTool (Python)**: This module has the finality of provides the needed tools for obtaining and adapt the data from some standards for geographical data webservice with the objective of can import this data on any graphic motor, it can generate diverse layers of data for the downloaded terrains as textures (RGB, infrared, etc.), geojson, etc.
- **ScriptAPI (Python)**: This module provides the interface to control the simulator trough scripts and trajectories files, its allow to control the vehicle, cameras, when generate images for the dataset, etc. Also can implements extra functionality as generation of noise to the vehicle and more.
- **Unreal Engine Simulator (C++)**: Based on the Unreal Engine includes the plugins to implement an RPC Server, interface necessary to read files generated by GEOTool and the generation of images.

6 Module GEOTool

In this section can see how works the module GEOTool and which is the functionality implemented, more concretely can see the origin of data, the configuration file, how to prepare data for graphical engines like as Unreal, Unity,

OpenGL, etc. Finally it will be analysed the performance of the generation from wavefront object files (.obj).

6.1 Obtaining data

With the objective of getting geoghrapical data it has been decided the communication with standards proposed by Open Geospatial Consortium[10]: Web Map Service[11] in charge of make available image data how can it be orthophoto from a geographical region selected and Web Coverage Service[11] in charge of returns data concerning at the elevation of terrains in a concrete geographical region. In order to get it data they are made HTTP requests a the web services offered by multiple institutions that follows the standards called, in this case will be made tests with the Institut Cartogràfic I Geològic de Catalunya[13].

6.1.1 Configuration file

For determining which data will wish be obtained and where webservices the app accept by command line parameter a file with the configuration in JSON format. This allows determining the properties of the data they want to request as I can see on the appendix A.1. On this file can configure the next parameters:

- **Type**: It refers to the type of coordinates that we will pass, can be latlong or x-y, in the first case it will make the corresponding transformation to the UTM format (x-y).
- **Coordinates**: Coordinates on which we want to make the request in the format indicated in the type field. If ”xyïs selected, the attributes x,y will be defined, and in case of choosing l·latlong”we will define the lat, long attributes.
- **Dimensions**: The dimension field can choose the dimensions that you want to request in pixels, it is composed by:
  - Bbox: Bbox is the dimensions in pixels of geographical area, this is used by all requests to identify the area in a unique way.
  - Texture: Texture is a resolution in pixels request to the texture image.
- **chunks**: The fragments that want to be downloaded, the application calculates the displacement and generates n pieces of width * height.
  - width.
  - height.
- **cellsize**: Size of each pixel in meters.
- **meshStep**: The number indicates the quantity of points that wish to skip on the mesh (Default: 1). More information on the section 6.4.
- **Wcsurl**: The URL to the web service that will give us the height data.
- **Outputwcs**: Base name of the output files.
- **Formatwcs**: Format of file generated by heights. Available: raw, obj (Object 3D).
- **Wmsrequests**: Array with each request that will be for obtaining textures, each item are composed by:
  - Url: URL to the webservice WMS.
  - Layers: The layers we want to get from these webservice.
  - Name: The name of texture, used by the generated files.
  - Format: Output format. Available: JPG.

6.2 Generation of terrains from a elevations maps

In this section explain the multiple forms realized to generate terrains that can be interpreted with multiple graphic engines. More concretely can see the RAW format and Wavefront object (.obj).

6.2.1 Format RAW

The RAW format is a format plain based on values of 16 Bytes, where the value of sea is 128. All the heights are saved in binary format put in the file of form consecutive. This format is accepted by the Unreal and Unity land builder, but has dimension limitations; You must meet several specified conditions in the Unreal Editor, this causes you to lose control of the generated mesh, the texture coordinates do not match and the texture that is applied to the mesh will have to adapt. Motivation by which decide to add the generation of 3d objects in the standard format, generating own objects as you can see in section 6.2.2.

6.2.2 Generation of mesh 3D

To import land in the graphics engine, it decides to generate a 3D mesh in obj format compatible with any 3D editor, graphics engine, etc. This device gives the freedom to control the distance between the vertex, where the texture is applied and which is the normal vector of each vertex to correctly apply the algorithms of illumination.

As the treatment with loops is slow, it’s made all the operations with the library NumPy take advantage of efficiency incorporates this library with the calculus of matrices. The problem has been adapted to operations of type matricial, you can see the code available in the annex A.2.

For the generation of objects must be defined four types of objects:

- **Vertex**: Are each point in the world. To say what vertex are referenced are defined for the index according to read the elevation grid (1, 2, 3,..., H*W). Each point is multiplied by a K (K represents the distance between two vertex according to the distance that indicate the elevation map obtained).

- **Vertex of texture**: Vertex with two components x and y compressed between 0 and 1 that indicates the correspondence between the points of the texture and location in the mesh. This property is calculated with the equations 1 and 2.

$$u = f(column) = column/(width - 1) \quad (1)$$

$$v = f(row) = 1 - (row/(height - 1)) \quad (2)$$

- **Normal of vertex**: Are vectors indicates the direction in which reflects the light for each vertex of the object. In order to calculate these normals is necessary, calculate the normal for each face, these are not included in the final file because the engines generate it by default according to the order in which indicates the faces you can see later.
  - Generation of normal faces: In order to generate the normals of a face once you know it the relation between the faces will be following the pattern you can see in the figure 6 where will be follow the equation 1 for the calculation of the normal face. $\vec{A}$, it makes cross product $\vec{C} = \vec{B} * \vec{A}$ and finally is normalize the vector $\vec{Normal} = \frac{\vec{C}}{|\vec{C}|}$.
  - Generation of the normal for the vertex: In order to generate the vector normal for each vertex, it's using the structure that can see in figure 7 applying the next equation for each vertex where V correspond to the vertices and F correspond to the faces on the figure 7:

$$\vec{NormalV} = \vec{F1} + \vec{F2} + \vec{F3} + \vec{F4} + \vec{F5} + \vec{F6} \quad (3)$$

$$\vec{NormalV} = \frac{\vec{NormalV}}{|\ \vec{NormalV}\ |} \quad (4)$$

- **Faces**: In this step it's determined which is the union of vertices to generate the different faces of the mesh, in this implementation it's decided to make a triangulation, in other words, that is for each square of the own mesh will be generated two triangular faces. It's important generate the faces in correct order writing the vertices in opposite clockwise, in this way the graphic motors determine that the normal face will point up by displaying the 3D mesh correctly.

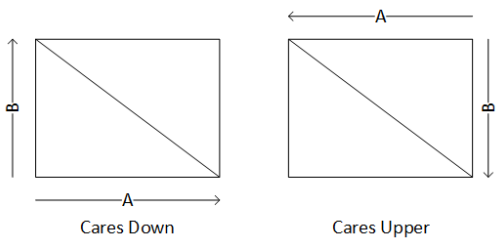


Fig. 6: Pattern for the calculation of normal on the faces

Once the generation process has been completed, the application will generate a mesh that can be opened in any editor. As it can see in the figure 8 it's open in software MeshLab.

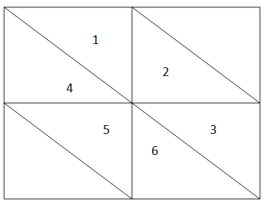


Fig. 7: Pattern for calculation of normal at a vertex

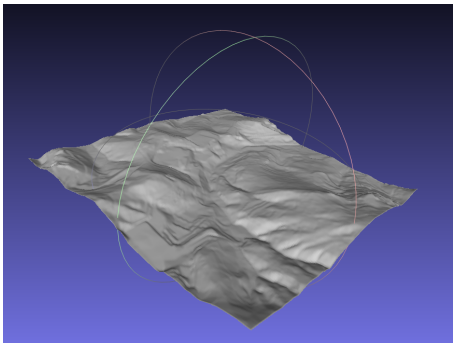


Fig. 8: Terrain mesh visualized in software MeshLab

6.2.3 GEOJson associated at the terrain

In order to can localize the terrain generated in other software that working with geospatial data or in the same simulator Unreal, it's decided to include in the generation a GEOJson that indicates the area which belongs the files generated. It's can see an example of this in the appendix A.3.

6.3 Time of generation

In this section it's analyse the two versions implemented and can see the difference in time of generation. This is an important point for future implementations in which will wish implement a viewer in real time that loading new data according to the user moves for the world.
On the figure 9 can see the time spent from the version with loops for and the version implemented with NumPy, can see when realised the implementation using the library NumPy are reducing the time in 24x for a size of 500*500, this is due to the fact that NumPy is optimized for a parallelise the processing of data vectorial way.

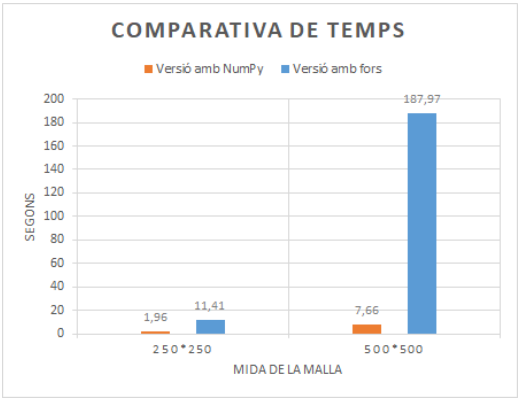


Fig. 9: Graph with the time of generation of a mesh according to size

6.4 Quality

In this section is looking the form in which reduce the quality of terrain and can see the results obtained both qualitatively (visual difference) as quantitatively (size in bytes). In order to make this reduction are made skips of size ”meshStep”set in the JSON configuration. It’s made tests with powers of 2 (2, 4, 8, 16,...) on the own matrix of points and using the terrain of size 300x300 pixels.

As we can see in the figure 10 when reducing the amount of points, we reduce the size of the disk by exponentially, making the file load faster on the Unreal environment. As it’s can see in the figure 11 is visible the loss of quality taking as a reference the mountain in the background on this see the loss of definition at the top of the mountain. It’s can consider visually that when it is configured ”meshStep” on 1 or 2 the loss qualitative, not are appreciable, from 4 are starting to appreciate slightly, finally on the levels 8 and 16 its can see the most loss, where the top of the mountain can see simplified the apposite that it can see with the ”meshStep” configured to 1 where can see a mountain range with a good definition.

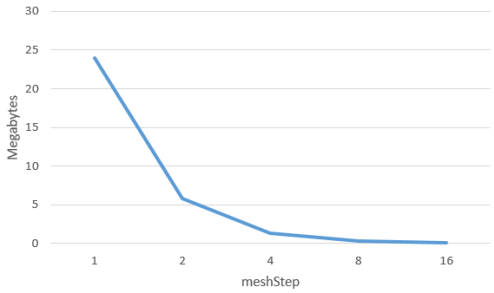


Fig. 10: Graphic with the size occupied on hard disk varying the ”meshStep” parameter for a terrain of size 300x300.

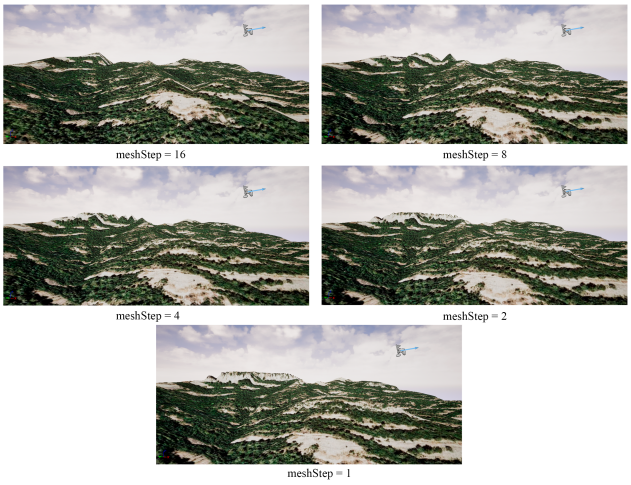


Fig. 11: Visual comparison of the effects produced by the level quality

7 Module: Simulator

In this section its explain the multiple parts of which is composed the module of simulation. The objective of this module is to make available a tool that allows simulate a travel with a generic vehicle above a terrain getting images from multiple cameras configured on board of the vehicle. A way to make this is providing the interface for control the vehicle with an external script sending commands with the RPC Protocol, in this case, it’s implemented the server as can see in section 7.1.1 and control it with a client implemented in a module as can you see in the section 8.

7.1 Visualization and control of the vehicle

This part is based on providing the interface for moving a generic vehicle to the simulated world, this vehicle is configured with several cameras that are displayed on the screen and that can generate images of each one of them, in this way the possibility is added to view the same area in multiple perspectives. As an example, you can see an zenith view in the figure 12.

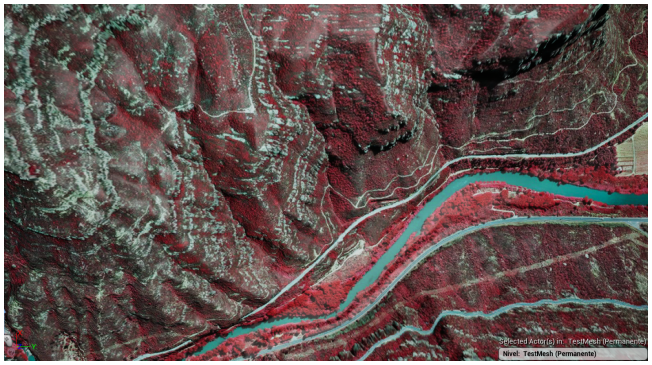


Fig. 12: Infrared view of Montserrat in zenith position

7.1.1 Remote control

In the order to controller remotely the vehicle, it’s implemented a communication using the RPC protocol (Remote Procedure Call) more concretely it’s using the library Rpclib[15] for C++. This server is implemented in the Unreal module, the RPC server management is done through a class (AVehiclePawn) that can be inherited so that it offers basically the interface necessary to control a vehicle, specifically the following actions is implemented:

- Initialize/stop the RPC server: Initialize the server by pressing the Y cloth to the simulator of Unreal and stop it with the U key.
- Change the position of the vehicle.
- Change rotation of the vehicle or cameras (Implemented with LookAt).
- Ask for images from one of the cameras to be generated.

The assignment of the functionalities is done in the function ”void BindFunctions(rpc::server* server)” which receives the instance of the server, this function can be overridden to later be added to legacy classes with more functionality

as can be seen in the annex ref appendix: extendrpc. By default the AVehiclePawn class allows us to call the following RPC functions:

- setLocation(double x, double y, double z): Enviem la localització en la qual volem que és situí el nostre vehicle.
- setLookAt(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.
- setLocationAndLookAt(double x, double y, double z, double lx, double ly, double lz): Funció que crida d'un a les dues anteriors.
- setCameraLookAt(int cameraId, double x, double y, double z): Funció que canvia la direcció en la que mira la càmera que l'indiquem.
- 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 sense ombres per tal que aquestes siguin generades per al motor d'Unreal. Podem veure un exemple de diferents hores del dia a la figura 13.

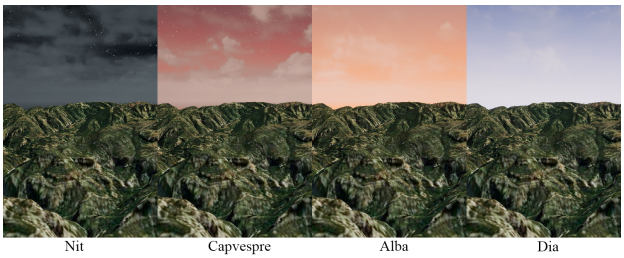


Fig. 13: Exemple de cels

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 Browser[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:

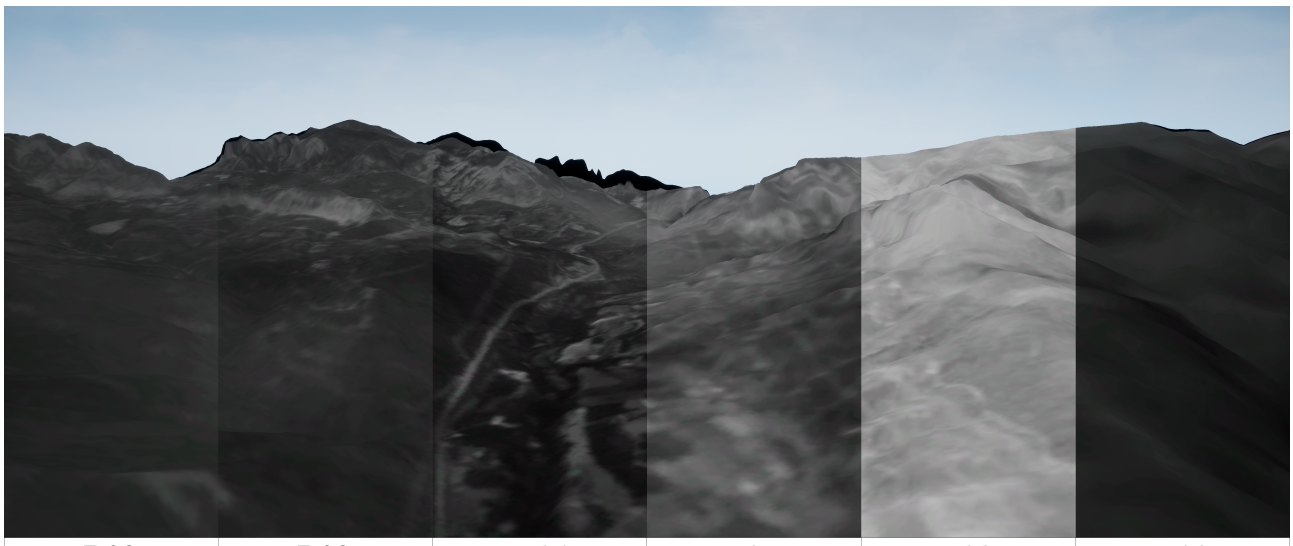
- NVDI[20]: Basat en la combinació de les bandes (B08 - B04)/(B08 + B04), índex utilitzat per veure l'estat del conreu.
- Moisture index[21]: Basat en la combinació de les bandes (B8A - B11)/(B8A + B11), indica la proporció de precipitació que es necessita per tal de satisfer les necessitats de la vegetació.
- NDWI[22]: Basat en la combinació de les bandes (B03 - B08)/(B03 + B08), índex que s'utilitza per determinar l'estrès 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 per algoritmes de reconstrucció 3D amb monocular stereo, prevenció de col·lisions, algoritmes de multiview stereo, etc.

7.4 Interacció amb el simulador

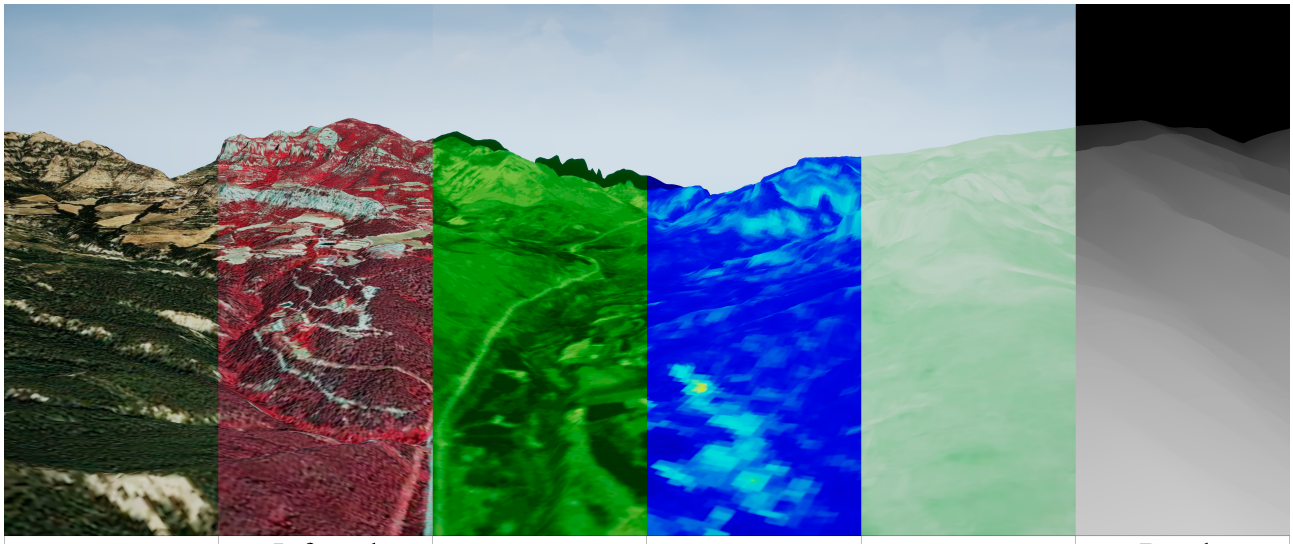
Per tal d'interaccionar amb el simulador s'han definit les següents tecles:

- A,W,S,D: Per moure'ns en el simulador.
- Q,E: Per rotar la vista.
- Y: Inicia el servidor RPC.
- U: Para el servidor RPC.



B02	B03	B04	B05	B08	B09
Blue	Green	Red	Vegetation red edge	NIR	Water vapour

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



RGB	Infrared	NDVI	Moisture index	NDWI	Depth
0,25m/px	0,25m/px	10m/px	10m/px	10m/px	5m/px
Cartografic	Cartografic	Sentinel-2	Sentinel-2	Sentinel-2	Simulator

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

- H: Mostra/oculta la visualització del vehicle.

- K: Guarda en un fitxer la vista actual de la segona càmera.

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.

8.1 Trajectòries

El mòdul d'scripting permet fer vols llegint trajectòries creades en el món real (adaptant-les al nostre format) o generades sintèticament d'aquesta forma podem recrear en el simulador la trajectòria feta anteriorment podent reproduir tants cops com vulguem aquesta, això ens permetrà fer diverses proves amb diferents visualitzacions del terreny. Per aquesta finalitat s'ha generat el fitxer que expliquem a la secció 8.1.1.

8.1.1 Fitxer de trajectòries

En aquest apartat explicarem com està formatjat el fitxer de trajectòries i el fitxer per posicionar la càmera. Els fitxers estaran en format CSV i seran llegits des de mòdul GEO-

Control per a indicar al simulador les posicions del vehicle i cap a quina posició del món mira tant el vehicle com les càmeres.
El fitxer que controla al vehicle està compost pels següents camps:

- Time: Temps en mil·lisegons d'ençà que comença el script.
- x: Posició X en format UTM.
- y: Posició Y en format UTM.
- z: Posició Z en format UTM.
- LookX: Posició X a la que mirem en format UTM.
- LookY: Posició Y a la que mirem en format UTM.
- LookZ: Posició Z a la que mirem en format UTM.

Per tal de controlar a on miran les cameras tindrem un altre fitxer composat per els camps:

- Time: Time en milisegons des de que comença el script.
- cameraId: Id de la camera que volem modificar.
- LookX: Posició X a la que mirem en format UTM.
- LookY: Posició Y a la que mirem en format UTM.
- LookZ: Posició Z a la que mirem en format UTM.
- GetImage: Booleà de 0 o 1 en el que indiquem si volem generar una imatge d'aquesta càmera en aquell instant de temps.

Podem veure un exemple dels fitxers CSV a l'apèndix A.5 generats amb Excel.

8.2 Simulació de soroll

Ja que els vehicles tenen moviments inesperats a conseqüència de l'aire, l'asfalt entre d'altres s'ha implementat un generador de soroll aplicat a la trajectòria que volem reproduir d'aquesta forma podem generar imatges amb soroll i moviments inesperats. Aquest soroll s'ha implementat mitjançant un soroll gaussià en el qual utilitzem com a centre el punt al qual ens volem moure i una sigma petita (entre 0 i 1) que ens generarà un moviment lleuger.

8.3 Generació de Datasets

Un dels objectius d'aquest treball és la possibilitat de generar datasets d'imatges sintètiques amb dades generades per satèl·lits, per tal de poder generar datasets s'utilitza el camp GetImage vist a la secció 8.1.1 que ens generarà una imatge cada cop que trobi aquest camp.
En la figura 16 podem veure una petita mostra amb poques imatges d'un recorregut en el qual hem fixat que la càmera mires a un punt en concret que deixàvem enrere amb el transcurs del temps.

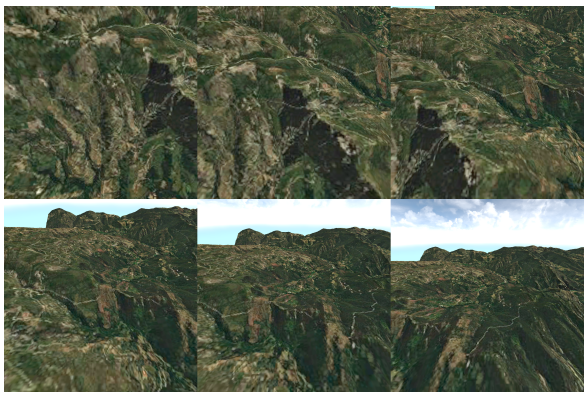


Fig. 16: Conjunt d'imatges generades per GEOControl

9 Conclusions

En aquest projecte hem vist com hem pogut obtenir textures i mapes d'elevacions de fonts procedents de satèl·lits i convertir-les en informació 3D que pugui ser afegida a un motor gràfic com és Unreal Engine. S'ha tingut en compte aspectes com la velocitat en generar el model 3D on s'ha pogut veure com afecta llibreries com NumPy en el tractament de matrius accelerant aquest procés gràcies a la paral·lelització, s'ha pogut veure l'efecte que té quantitativament i qualitativament en les malles en reduir la quantitat de punts amb els quals es genera el model 3D necessari per a la renderització de models de grans dimensions.
Per la part del mòdul simulador hem pogut veure com podem visualitzar informació en un entorn 3D generat amb Unreal Engine. Es poden afegir càmeres a un vehicle simulat i veure des de diferents perspectives una mateixa informació per això s'ha creat un altre modulo que envia comandes al servidor RPC que s'ha implementat en Unreal Engine el qual pot moure, dir cap a on mira el vehicle, dir cap a on mira les càmeres i obtenir imatges d'aquestes càmeres. S'han fet proves amb diverses visualitzacions provinents d'informació de satèl·lits com és el Sentinel 2 en el que s'ha pogut veure com és representa'n en el món simulat i quines utilitats té aquesta informació.
Per la part del mòdul d'scripting s'ha vist com es poden generar trajectòries reproduïbles en l'entorn simulat per tal de fer diversos viatges amb diferents informacions i extraure dades en forma de dataset. Aquestes trajectòries poden controlar diversos paràmetres del vehicle i càmeres indicant en quins punts es vol obtenir una imatge i de quina càmera. Per tal de simular el moviment a causa del vent o altres factors s'ha afegit un petit filtre com és la generació de soroll per tal de fer que se sacsegi el vehicle.

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 mgspack-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/calculo-del-indice-ndwi-diferencial-de-agua-normalizado/ [25/05/2019]

APÈNDIX

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

```
{
  "type": "xy",
  "coordinates" : {
    "x": 326737.23,
  "y": 4676971.49,
  "zone": "31T"
  },
  "dimensions": {
    "bbox":{
      "height": 1500,
      "width": 1500
    },
    "texture":{
      "height": 1500,
      "width": 1500
    }
  },
  "chunks": {
    "width": 3,
    "height": 3
  },
  "cellsize": 15,
  "meshStep": 4,
  "wcsUrl": "http://geoserveis.icc.cat/
     icc_mdt/wcs/service",
  "outputWcs": "example9x9big/pallars15
     ",
  "formatWcs": "obj",
  "wmsRequests": [
    {
      "url": "http://geoserveis.icc.cat
         /icc_ortohistorica/wms/service
         ",
      "layers": "orto25c2016",
          "name": "rgb",
      "format": "jpg"
    },
    {
      "url": "http://geoserveis.icc.cat
         /icc_ortohistorica/wms/service
         ",
      "layers": "ortoi25c2016",
          "name": "ir",
      "format": "jpg"
    }
  ]
}
```

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
  });
}
```

A.5 Fitxers CSV d'exemple per controlar el vehicle i les càmeres

Time	x	y	z	LookX	LookY	LookZ
0	10000	2000	3000	10000	2100	3000
50	10000	2020	3000	10000	2120	3000
100	10000	2040	3000	10000	2140	3000
150	10000	2060	3000	10000	2160	3000
200	10000	2080	3000	10000	2180	3000
250	10000	2100	3000	10000	2200	3000
300	10000	2120	3000	10000	2220	3000
350	10000	2140	3000	10000	2240	3000
400	10000	2160	3000	10000	2260	3000
450	10000	2180	3000	10000	2280	3000
500	10000	2200	3000	10000	2300	3000
550	10000	2220	3000	10000	2320	3000

Fitxer CSV per controlar el vehicle

Time	cameraId	LookX	LookY	LookZ	GetImage
0	1	10000	2000	0	1
50	1	10000	2020	0	0
100	1	10000	2040	0	0
150	1	10000	2060	0	0
200	1	10000	2080	0	0
250	1	10000	2100	0	0
300	1	10000	2120	0	0
350	1	10000	2140	0	0
400	1	10000	2160	0	0
450	1	10000	2180	0	0
500	1	10000	2200	0	1
550	1	10000	2220	0	0
600	1	10000	2240	0	0
650	1	10000	2260	0	0
700	1	10000	2280	0	0
750	1	10000	2300	0	0
800	1	10000	2320	0	0
850	1	10000	2340	0	0
900	1	10000	2360	0	0
950	1	10000	2380	0	0
1000	1	10000	2400	0	1

Fitxer CSV per controlar les càmeres