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 representa 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
   - 1.1. Investigar l'estat de l'art
     - 1.1.1. Estudiar AirSim
     - 1.1.2. Estudiar Carla SIMULATOR
     - 1.1.3. Estudiar proposta pròpia
   - 1.2. Definir els objectius
   - 1.3. Investigar motors gràfics
     - 1.3.1. Investigar Unreal Engine
     - 1.3.2. Investigar altres motors gràfics
   - 1.4. Investigar els mapes d'altura
   - 1.5. Investigar les imatges tant RGB com multiespectrals
   - 1.6. Investigar webservices WMS
   - 1.7. Investigar llibreries RPC
2. Definir
   - 2.1. Definir mòduls interessants 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
     - 3.1.1. Desenvolupar el mòdul de transformació de dades d'altura i textures
     - 3.1.2. Desenvolupar l'importador de dades en Unreal Engine
   - 3.2. Desenvolupar mòdul gràfic (Ureal Engine)
     - 3.2.1. Desenvolupar la interfície del menú
     - 3.2.2. Desenvolupar el mòdul per importar fitxers personalitzats en Unreal Engine
     - 3.2.3. Desenvolupar codi per la generació del terreny
     - 3.2.4. Desenvolupar codi per la generació dels materials
       - 3.2.4.1. Visualitzar diferents materials (RGB, Multi-espectre, ...)
     - 3.2.5. Desenvolupar codi de servidor per control del vehicle
   - 3.3. Desenvolupar mòdul de scripting
     - 3.3.1. Desenvolupament client que controlarà el vehicle
   - 3.4. Integrar els mòduls de AirSim en Unreal Engine
     - 3.4.1. Integrar mòdul de Drons en el projecte
     - 3.4.2. Integrar mòdul de Segmentació en el projecte
   - 3.5. Desenvolupar altres capes d'informació
4. Testejar
   - 4.1. Fer provàs del mòdul de transformació de dades
     - 4.1.1. Provar amb dades locals
     - 4.1.2. Provar amb dades externes
   - 4.2. Fer provàs del mòdul gràfic
   - 4.3. Fer provàs 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

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

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 organitza 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.

4.1 AirSim

AirSim és un simulador gràfic elaborat en Unreal Engine[6] 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

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 2 escrits en diversos llenguatges com es pot veure a continuació:

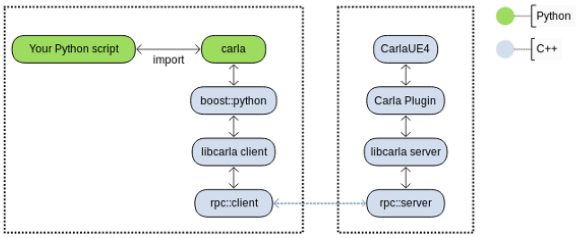

Fig. 2: Relació entre els mòduls de Carla

- **LibCarla (C++)**: Llibreria principal 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 3 podent reutilitzar alguns dels petits mòduls opensource 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.

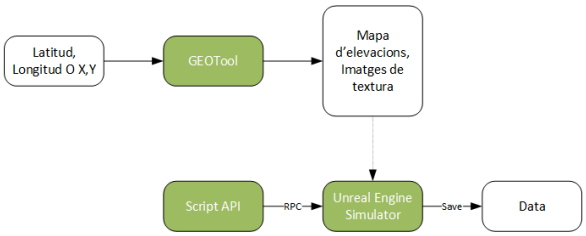

Fig. 3: Estructura DronSimulator

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[7]: Web Map Service[8] encarregat de posar a disponibilitat dades d'imatge com poden ser ortofotos d'una zona geogràfica seleccionada i Web Coverage Service[9] 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[10].

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'escull "xy"es definirà els atributs x, y en cas d'escollir l'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.
- **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 vàries 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

6.2 Estructura de GEOTool

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 vàries 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[11] 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 aprofitant 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 (Distancia 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 indican 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.

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

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

- **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 4 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 $\vec{Normal} = \frac{\vec{C}}{|\vec{C}|}$.
  - 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 5 aplicant la següent formula a cada vèrtex on V correspon als vèrtexs i F a les cares de la figura 5:

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

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

- **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.

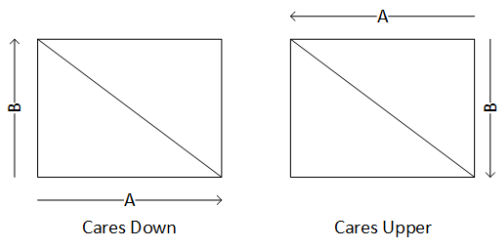

Fig. 4: Patró per càlcul de normals en les cares

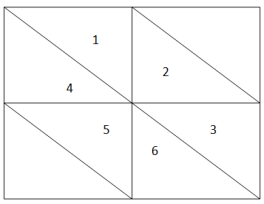

Fig. 5: Patró per càlcul de normals en un vèrtex

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 6

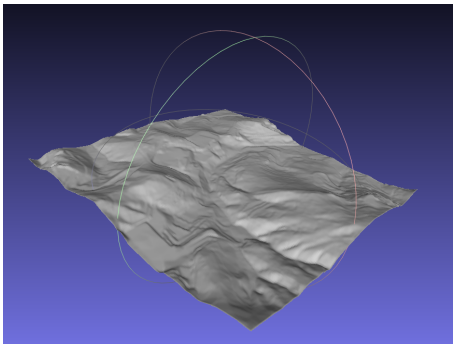

Fig. 6: Malla d'un terreny visualitzada en l'aplicació MeshLab

7 Mòdul: Simulador

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

8 Mòdul Scripting

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] Unreal Engine - `https://www.unrealengine.com/en-US/what-is-unreal-engine-4` [09/03/2019]

[7] Open Geospatial Consortium (OGC) - `http://www.opengeospatial.org/` [08/04/2019]

[8] Web Map Service (WMS) - `https://www.opengeospatial.org/standards/wms` [08/04/2019]

[9] Web Coverage Service (WCS) - `https://www.opengeospatial.org/standards/wcs` [08/04/2019]

[10] Institut Cartogràfic i Geològic de Catalunya (ICGC) - `http://www.icgc.cat/ca/` [08/04/2019]

[11] Wavefront .obj file - `https://en.wikipedia.org/wiki/Wavefront_.obj_file` [09/04/2019]

APÈNDIX

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

```
{
  "type": "latlong",
  "coordinates" : {
    "lat": 41.4677021,
    "long": 2.2859777
  },
  "dimensions": {
    "bbox":{
      "height": 225,
      "width": 225
    },
    "texture":{
      "height": 1500,
      "width": 1500
    }
  },
  "wcsUrl": "http://geoserveis.icc.cat/
     icc_mdt/wcs/service",
  "outputWcs": "outputwcs",
  "formatWcs": "obj",
  "wmsRequests": [
    {
      "url": "http://geoserveis.icc.cat
         /icc_ortohistorica/wms/service
         ",
      "layers": "orto25c2016",
      "output": "outputwms",
      "outputFormat": "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)
    normalVertex[:, 0] = normalVertex
        [:, 0] / module
    normalVertex[:, 1] = normalVertex
        [:, 1] / module
    normalVertex[:, 2] = normalVertex
        [:, 2] / module

    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 Secció d'Apèndix