

PROYECTO FIN DE CARRERA

“EDICIÓN E IMPRESIÓN DE MODELOS 3D”

Titulación: Ingeniería Informática (2º ciclo)

Departamento de Ingeniería Matemática e Informática

Alumno: Gonzalo Andrade Benavente

Teléfono: 677562988

Tutor: Oscar Ardaiz

Pamplona, XX de Abril de 2015

# Índice

[Índice 4](#_Toc416860084)

[Introducción 6](#_Toc416860085)

[Antecedentes 6](#_Toc416860086)

[Objetivos 6](#_Toc416860087)

[Tecnologías básicas 6](#_Toc416860088)

[Qué es una impresión 3D 6](#_Toc416860089)

[JavaScript 6](#_Toc416860090)

[Three.js 6](#_Toc416860091)

[Shapeways 6](#_Toc416860092)

[Desarrollo del trabajo 7](#_Toc416860093)

[Ficheros HGT 7](#_Toc416860094)

[Mapbox 7](#_Toc416860095)

[Uso Cesium 7](#_Toc416860096)

[Cesium 7](#_Toc416860097)

[Carga GPX 7](#_Toc416860098)

[Selección Mapa Mapbox 7](#_Toc416860099)

[Mapa y coordenadas 7](#_Toc416860100)

[Información tiles 7](#_Toc416860101)

[Mapa y recorrido 7](#_Toc416860102)

[Texturas 7](#_Toc416860103)

[Creación de Mesh 8](#_Toc416860104)

[Obtención de ruta 8](#_Toc416860105)

[Creación de escena y controles gráficos 8](#_Toc416860106)

[Información tiles 8](#_Toc416860107)

[Información del terreno 8](#_Toc416860108)

[Creación del terreno 8](#_Toc416860109)

[Creación x3d y envío a impresora 3D 9](#_Toc416860110)

[Creación del modelo 9](#_Toc416860111)

[Subir el modelo 9](#_Toc416860112)

[Resultados 10](#_Toc416860113)

[Aplicación Web 10](#_Toc416860114)

[Impresiones 3D 10](#_Toc416860115)

[Problemas y soluciones 10](#_Toc416860116)

[Conclusiones y líneas futuras 11](#_Toc416860117)

[Bibliografía 12](#_Toc416860118)

# Introducción

Actualmente uno de los temas más recurrentes en las nuevas tecnologías es todo lo relacionado con el diseño en tres dimensiones más conocido como 3D, desde la creación de modelos en el ordenador hasta la creación de impresoras que nos permitan crear esos modelos y que sean reales. La librería *threejs* nos permite crear modelos 3D en el navegador, podemos crear objetos básicos como cubos, planos, textos, cargar objetos prediseñados, crear nuestras propias geometrías y un sinfín de objetos en 3D.

## Antecedentes

Para cumplir el objeto de este proyecto desarrollare una aplicación que cree un modelo físico en 3D a partir de una ruta *gps*. Podemos encontrar en la web una idea que resume lo que queremos hacer en el artículo*[[1]](#footnote-1)* “*3D Print your trek, in color!”* el cual sigue los siguientes pasos:

* Descargar las elevaciones de los terrenos[[2]](#footnote-2).
* Unir varias regiones en un solo mapa con el programa 3DEM[[3]](#footnote-3).
* Seleccionar la región a imprimir y guardar el archivo en formato *dem*.
* Convertir el fichero *dem* a *x3d[[4]](#footnote-4)* con el programa AccuTrans 3D[[5]](#footnote-5).
* Crear el modelo 3D en Blender[[6]](#footnote-6) añadiendo efectos con la herramienta.
* Preparar la información del recorrido *gps* usando el programa Qstarz para exportar un fichero en formato *km[[7]](#footnote-7)l*.
* Generar la textura del terreno con *Google Earth*.
* Poner la textura del terreno sobre el modelo 3D con Blender además de editar la textura para que quede perfecta.
* Añadir grosor al modelo con Blender, escalar y exportar el modelo terminado.
* Imprimir el modelo 3D con *Shapeways*.

Como podemos ver se necesitan muchas herramientas y conocimientos para lograr a cabo un modelo 3D imprimible. Utilizare una serie de librerías en JavaScript que me simplificarán las cosas, la idea es simple:

* Con la ruta *gps* obtener los datos de las elevaciones de los terrenos con *Cesium*.
* Textura para el modelado tridimensional con *Mapbox*.
* Creación del objeto tridimensional en el navegador con *ThreeJS.*
* Unir objeto y textura enviando la información a *Shapeways* para su creación física.

En los siguientes apartados haré una breve descripción de cada librería y explicaré cómo funciona la aplicación web. Para lograr un código más legible y manejable he realizado durante todo el proyecto una programación modular, así cada módulo resuelve un problema particular.

## Objetivos

El objeto del proyecto es la creación de una aplicación web para la edición de objetos 3D que vayan a ser impresos en una impresora 3D usando tecnología WebGL[[8]](#footnote-8) con la librería *threejs* desarrollada en JavaScript.

## Tecnologías básicas

### Qué es una impresión 3D

Es la creación de un objeto tridimensional mediante la impresión sucesiva de finas capas de un material, este proceso es denominado fabricación por adición.

Existe un gran número de tecnologías para la impresión 3D, sus principales diferencias se encuentran en la forma en que las diferentes capas son usadas para crear piezas. Cada método tiene sus propias ventajas e inconvenientes, por lo tanto, todo dependerá de las prioridades del cliente. Generalmente las consideraciones principales son:

* Velocidad
* Coste del prototipo impreso
* Coste de la impresora 3D
* Elección y coste de materiales
* Colores.

|  |  |  |
| --- | --- | --- |
| Tipos | Tecnologías | Materiales |
| Extrusión | Modelado por deposición fundida | Termoplásticos, HDPE, metales eutécticos, materiales comestibles |
| Hilado | Fabricación por haz de electrones | Casi cualquier aleación |
| Granulado | Sinterizado directo de metal por láser | Casi cualquier aleación |
| Fusión por haz de electrones | Aleaciones de titanio |
| Sinterizado selectivo por láser | Polvo termoplástico |
| Sinterizado selectivo por calor | Termoplásticos, polvos metálicos, polvos cerámicos |
| Proyección aglutinante | Yeso |
| Laminado | Laminado de capas | Papel, papel de aluminio, capa de plástico |
| Fotoquímicos | Estereolitografía | Fotopolímero |
| Fotopolimerización por luz ultravioleta | Fotopolímero |

### JavaScript

Es un lenguaje de programación interpretado[[9]](#footnote-9) al igual que otros vistos en la diplomatura como php, Lisp, programación en R y Prolog.

Su sintaxis es similar a C, aunque adopta nombres y convenciones del lenguaje de programación Java aunque JavaScript y Java no están relacionados y tienen semánticas y propósitos diferentes.

JavaScript se utiliza principalmente en el lado del cliente añadiendo una mejora al interfaz para el usuario y dando dinamismo a las páginas interactuando con el DOM[[10]](#footnote-10).Con la llegada de Ajax[[11]](#footnote-11) proporcionando llamadas asíncronas al servidor JavaScript se ha convertido en uno de los lenguajes más populares en internet.

Algunos ejemplos del uso de JavaScript son:

* Cargar nuevo contenido para la página o enviar datos al servidor a través de Ajax sin necesidad de recargar la página (por ejemplo, una red social puede permitir al usuario enviar actualizaciones de estado sin salir de la página).
* Animación de los elementos de página, hacerlos desaparecer, cambiar su tamaño, moverlos, etc.
* Contenido interactivo.
* Validación de los valores de entrada de un formulario web para asegurarse de que son aceptables antes de ser enviado al servidor.

La ventaja de usar JavaScript es que al ser ejecutado en el lado del cliente la respuesta es más rápida. Otra ventaja es que JavaScript permite detectar acciones del usuario como pulsaciones de teclas, hacer clic en un botón, pasar por encima de un elemento de la página.

Finalmente otra de las razones por la cual JavaScript es un lenguaje de programación popular es la compatibilidad con la mayoría de los navegadores debido a la estandarización del W3C[[12]](#footnote-12).

### Three.js

A grandes rasgos *threejs* es una librería que nos permite crear WebGL[[13]](#footnote-13), básicamente la diferencia cabe en que si queremos crear un cubo con *threejs* basta con algunas líneas en cambio con JavaScript la cantidad de líneas sería mayor.

### Shapeways

Es una compañía cuya función principal es la impresión en 3D, nos podemos crear una cuenta usando un perfil de *Facebook*. Al disponer de una cuenta podemos subir nuestros modelos 3D creados con *threejs* y verificar si están bien construidos y si cumplen sus estándares para ser impresos. Se pueden elegir un sinfín de materiales lo cual implicaría un aumento o disminución de su precio, además de otros factores que alteran el precio como el tamaño del modelo.

# Desarrollo del trabajo

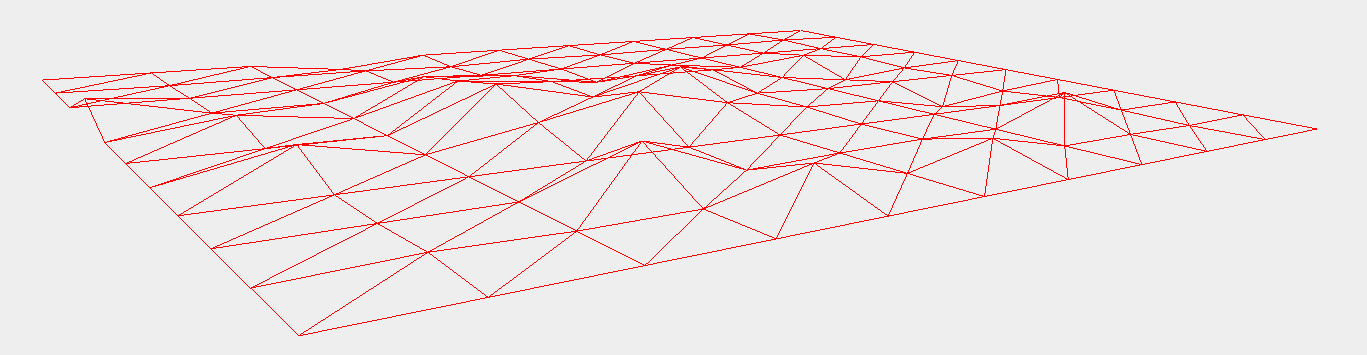
En este apartado se explicará el desarrollo y funcionamiento de la aplicación web.

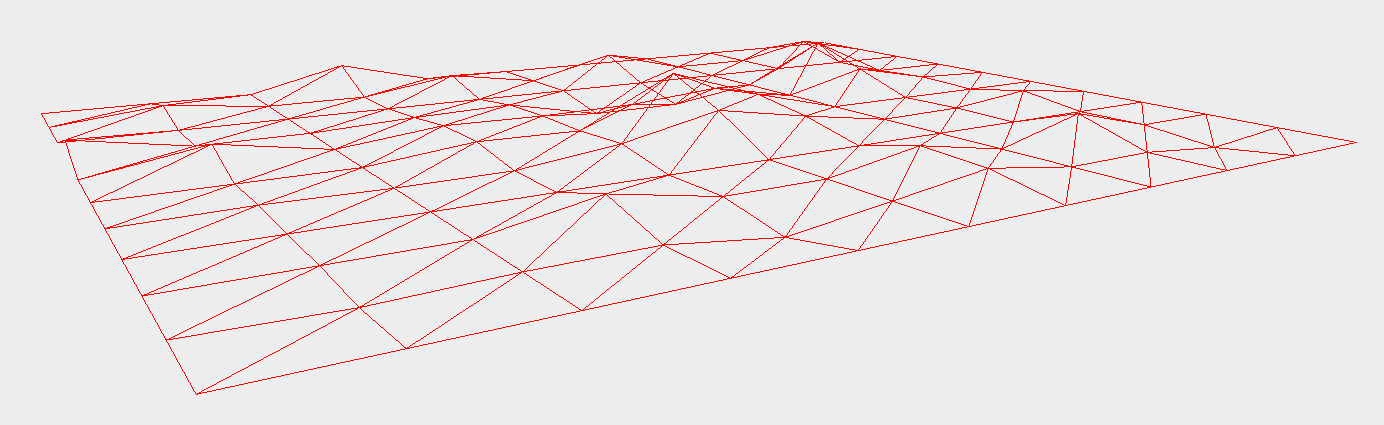
## Ficheros HGT

Estos ficheros contienen las elevaciones de la superficie de la tierra por regiones. Cada fichero describe un tile con la altura en grados de latitud y el ancho en grados de longitud, el nombre de cada fichero hace referencia a la latitud y longitud en la esquina inferior izquierda del tile, un ejemplo el fichero *N43W002.hgt*.

Realice un estudio de cómo interpretar la información de estos ficheros obteniendo los siguientes resultados basándome en el algoritmo *square-diamond* para la creación de terrenos aleatorios, este tipo de algoritmo se usa para la creación de terrenos en los videojuegos.

Junto a la memoria se adjunta los ficheros JavaScript para la creación de terrenos aleatorios sin base, con lo visto en el creador de terreno con *Cesium* se podría añadir fácilmente los laterales y el fondo creando una geometría completa. Imágenes de terrenos creados aleatoriamente:





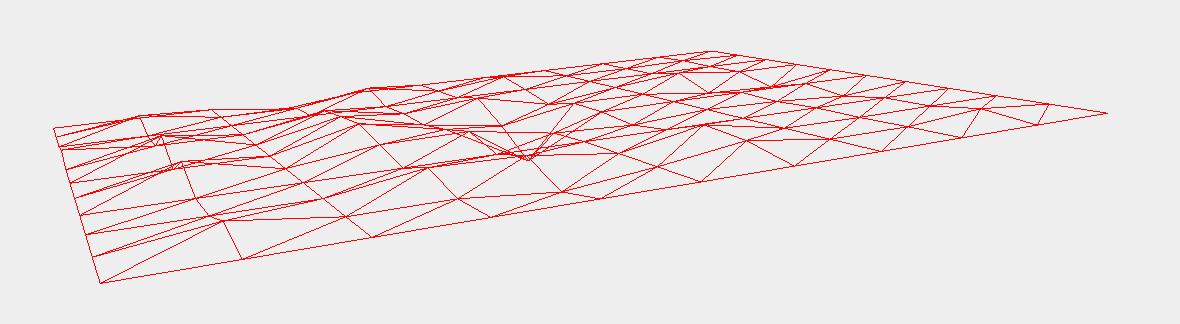
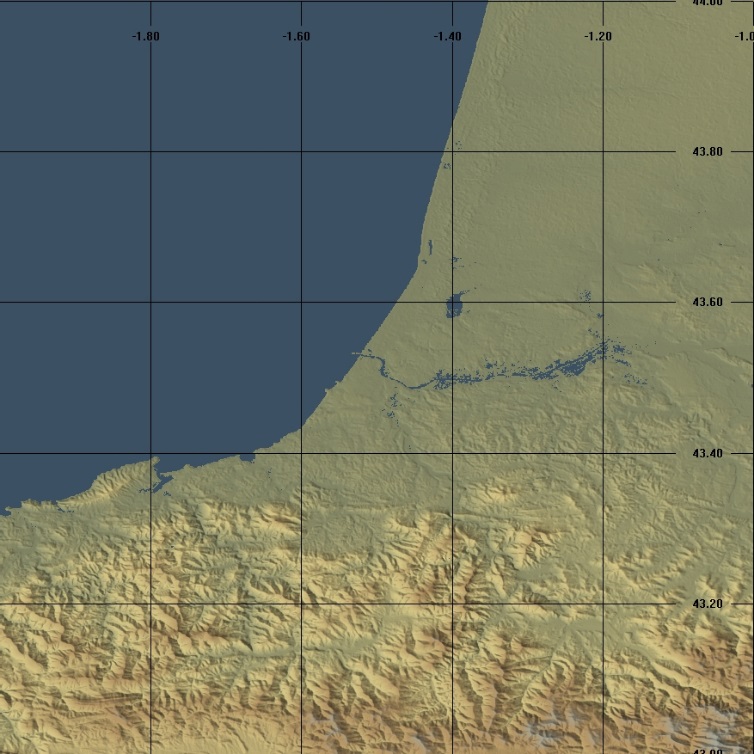
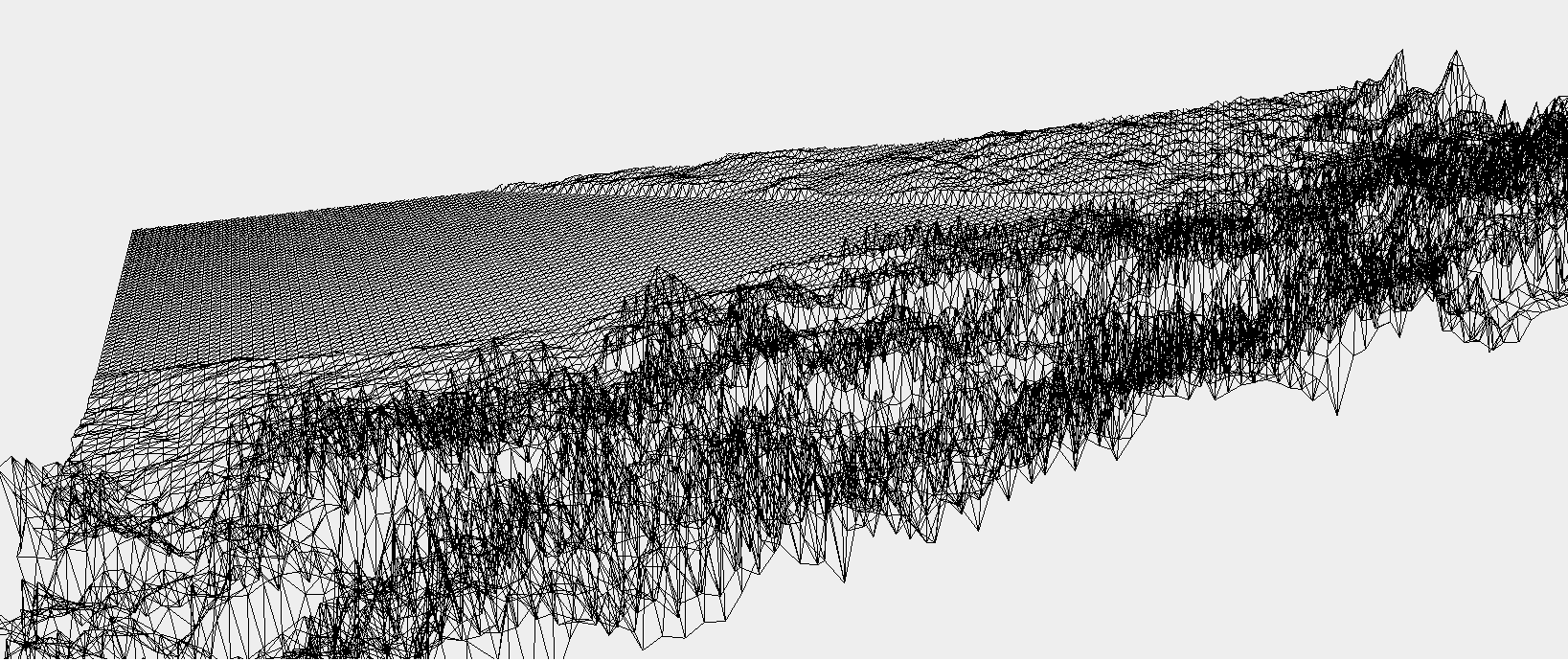


Imagen correspondiente a un fichero *hgt* y representación gráfica con *threejs*





### Mapbox

Es una librería JavaScript que nos permite crear mapas personalizados, sitios web como Foursquare, Pinteres, Evernote usan está librería. La información que nos ofrece Mapbox proviene de fuentes de información de libre acceso (open data) tales como OpenStreetMap y Nasa. Su tecnología está basada en Node.js, CouchDB, Mapnik, Gdal y Leafletjs.

Podemos usar algunos mapas que nos ofrece Mapbox por defecto sin la necesidad de crear una cuenta, para crear mapas y cambiar su diseño necesitamos crear una cuenta en la página oficial[[14]](#footnote-14).

## Uso Cesium

A la hora de usar los ficheros *hgt* me encontré con los siguientes problemas:

* Habría que descargar todos los ficheros correspondientes al plantea completo, gigabytes de información.
* Dificultad a la hora de trabajar con los datos debido a la gran cantidad de datos que son contenidos en ellos.
* Ubicación de coordenadas dentro del fichero.
* Gran cantidad de datos que ralentizaba la visualización del modelo en el navegador, muchos datos para el motor gráfico del navegador.

Debido a los problemas mencionados anteriormente en los ficheros *hgt* se optó por utilizar la librería *Cesium JS.*

### Cesium

Librería basada en JavaScript para la creación de globos terrestres en 3D y mapas en 2D en el navegador sin la necesidad de algún plugin, es de libre acceso[[15]](#footnote-15) bajo licencia Apache 2.0.

A diferencia de los otros frameworks basados en JavaScript este no posee *cdn*, por lo tanto, para su uso debemos descargarnos el framework desde la página oficial[[16]](#footnote-16).

Esencialmente utilizaremos la librería para acceder a los datos de las elevaciones del terreno. Resumiendo el uso de cesium:

* Con las coordenadas de la ruta accedemos a los tiles que nos ofrece cesium. Los tiles son rectángulos que guardan información acerca de los datos del terreno.
* La información obtenida en lo tiles la graficaremos usando *threejs*.

## Carga GPX

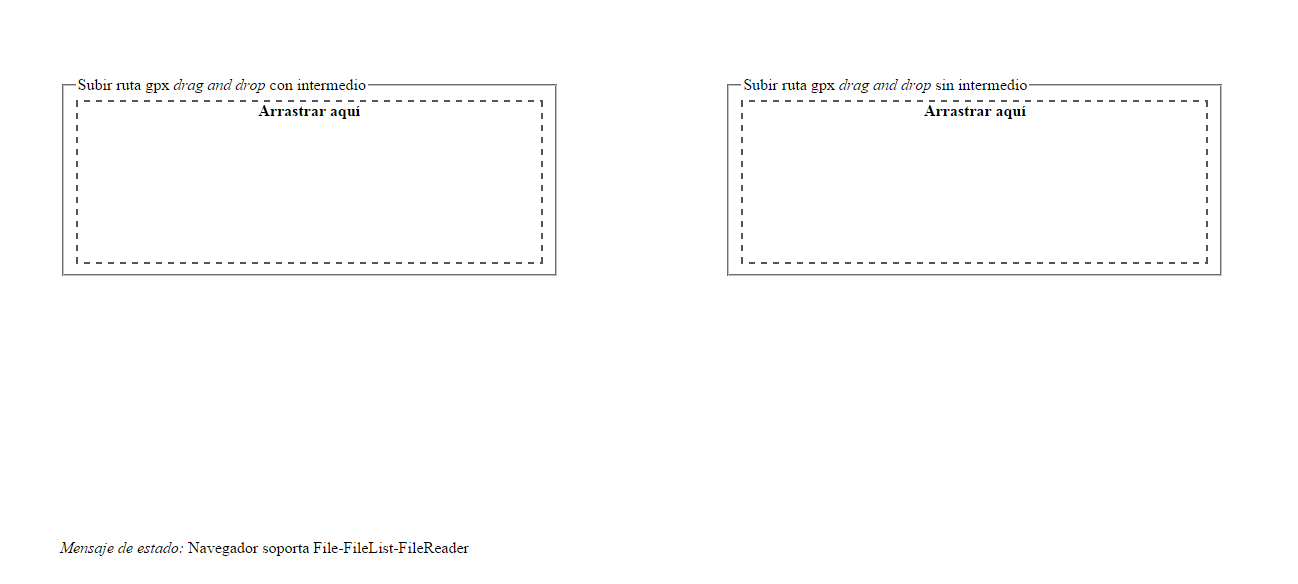
A partir de este apartado se explicará el funcionamiento de los distintos interfaces de la aplicación Web.

Se carga la página *PFCIndex.html* donde se subirá el fichero *gpx* al servidor mediante *ajax* y *php*, se ha controlado que solo se puedan subir ficheros *gpx* y cada vez que se cargue la página inicial se borraran todos los datos en el servidor.

El nombre del fichero subido al servidor quedará almacenado en una variable de sesión de JavaScript.



Para acceder a la variable almacenada en sesión solo debemos indicar el nombre de la variable.



Podremos elegir dos formas de subir el fichero, la primera nos permite elegir el tipo de mapa y visualizar la ruta antes de obtener el modelado 3D y la segunda cargaría el modelado 3D por defecto.

## Selección Mapa Mapbox

Si hemos elegido la primera opción que nos permite elegir el tipo de mapa y visualizar la ruta antes de crear el modelado 3D podremos interactuar en la página *PFCMyRute.html*, en caso contrario se cargará la página y se bloqueará hasta obtener todas las imágenes correspondiente a la ruta y se abrirá la página *PFCMyMesh.html* donde se creará el modelado 3D.

La página *PFCMyRute.html* a su vez trabaja con varios ficheros externos en JavaScript ubicados en la carpeta *js*, */js/my\_cesium\_rute.js*, */js/clases.js, /js/functions.js* y */js/Cesium.js* este último es el fichero ofrecido por Cesium.

En este apartado se explicará cómo se obtienen todos datos para la creación del terreno e imágenes del mismo.

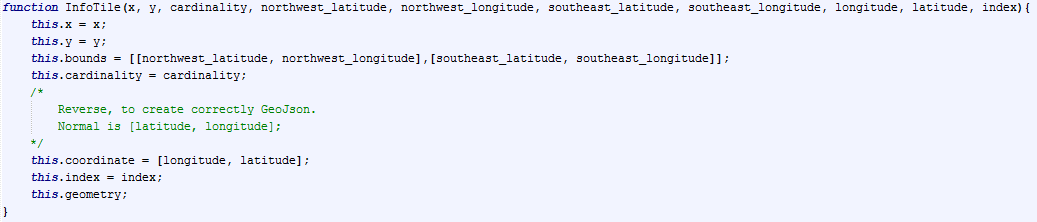
### Mapa y coordenadas

Cuando la conexión a Cesium esté lista llamamos a la función *getRute()* con la cual obtendremos las coordenadas de la ruta en el arreglo *coordinates* y se llamará a la función *createMap*() que creará el mapa y asignará los valores de las variables globales que usaremos en la aplicación.

### Información tiles

Cesium nos ofrece los datos mediante tiles, estos contiene una gran cantidad de información, entre esos datos tenemos las elevaciones del terreno. Un tile corresponde a un área rectangular del mapa. Usando la función *checkTile(coordinates, level)* creamos el arreglo *info\_tiles*,la función lleva como parámetros de entrada las coordenadas y el nivel de Mapbox. El arreglo guarda objetos *InfoTile* donde cada uno de ellos contiene la siguiente información:

* *bounds:* Límites del tile.
* *cardinality:* En un principio se usó para la creación del terreno, luego por modificaciones se ha usado para guardar la referencia de donde fue creado la variable, en que función del código.
* *coordinate:* coordenadas iniciales de la ruta.
* *Index:* posición inicial de la ruta correspondiente al arreglo de coordenadas *coordinates.*
* *x:* coordenada x de Cesium para solicitar la información de un tile.
* *y:* coordenada Y de Cesium para solicitar la información de un tile.



En este punto tenemos los tiles que corresponden a la ruta y como se mencionó antes la idea es crear un terreno rectangular, por lo tanto, llamamos a la función *createRectangle(info\_tiles, level)*. Está función nos devuelve un arreglo que contiene objetos *InfoTile* correspondientes a la ruta y añade nuevos valores completando un rectángulo, como parámetros de entrada la función recibe los tiles del recorrido y el nivel de Mapbox. El resultado de la función lo guardaremos en la arreglo *rectangle\_tiles*.

### Mapa y recorrido

Con el mapa creado y con toda la información necesaria para la creación de nuestro modelado en el arreglo *rectangle\_tiles* llamaremos a la función *loadGpx()* que se encargará de centrar el mapa con el primer y último tile del arreglo *rectangle\_tiles*, es posible que no se visualice toda la ruta en el mapa, para ello se puede acercar o alejar la vista del mapa usando el ratón.

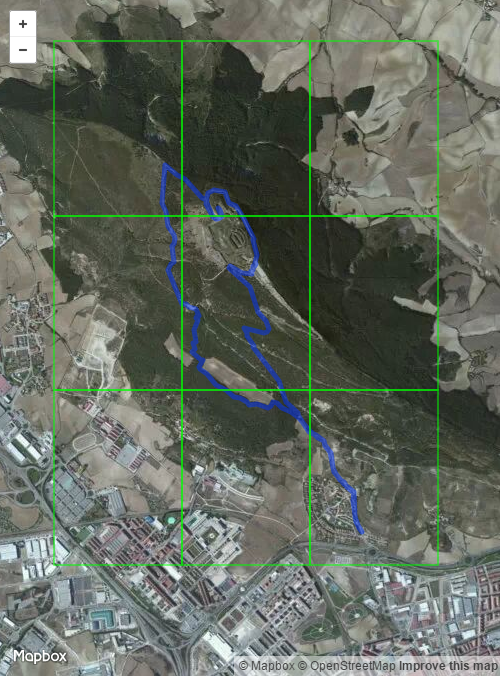
Además de centrar el mapa la función *loadGPX()* dibujará la ruta sobre el mapa para verificar que los datos cargados sean correctos. En principio se usó el método *omnivore* para cargar rutas *gpx* en los mapas pero en algunos casos no se dibujaba bien la ruta.

Por cada coordenada verifico al tile que pertenece obteniendo de esta manera todos los tiles correspondientes a la ruta. Además almaceno los límites Sur-Oeste, Sur-Este, Nor-Oeste, Nor-Este de cada tile, así en *Mapbox* puedo dibujar un rectángulo para identificar cada tile de la ruta.



### 

La idea es obtener un modelado rectangular del recorrido, por lo tanto, a partir de la ruta he añadido más tiles identificando los máximos y mínimos laterales de la ruta. Para obtener la información de cada tile debemos indicar sus coordenadas y nivel. En este proyecto se usó el nivel 14 ya que es el mayor nivel soportado para la zona Europea, en zonas de Estados Unidos soporta un nivel de 15, mientras mayor es el nivel más detallado son los datos obtenidos.

Mapas del recorrido en nivel 14 y 15.

### Texturas

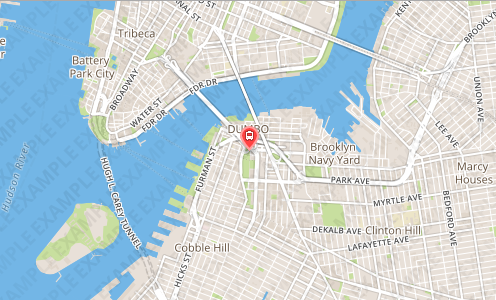
En este apartado se explicará cómo se crea la imagen de nuestro recorrido.

Para obtener la textura final debemos crear la textura por cada tile y luego unir todas las texturas creadas. Para extraer las imágenes usamos el API *Static maps* de Mapbox, nos permite obtener distintos tipos de imágenes:

* Imágenes de mapas simples.



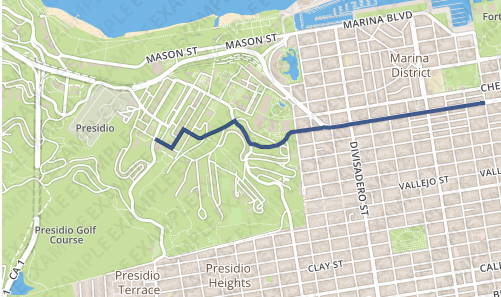
* Imágenes de mapas con marcadores.



* Imágenes de mapas con rutas mediantes GeoJSON.



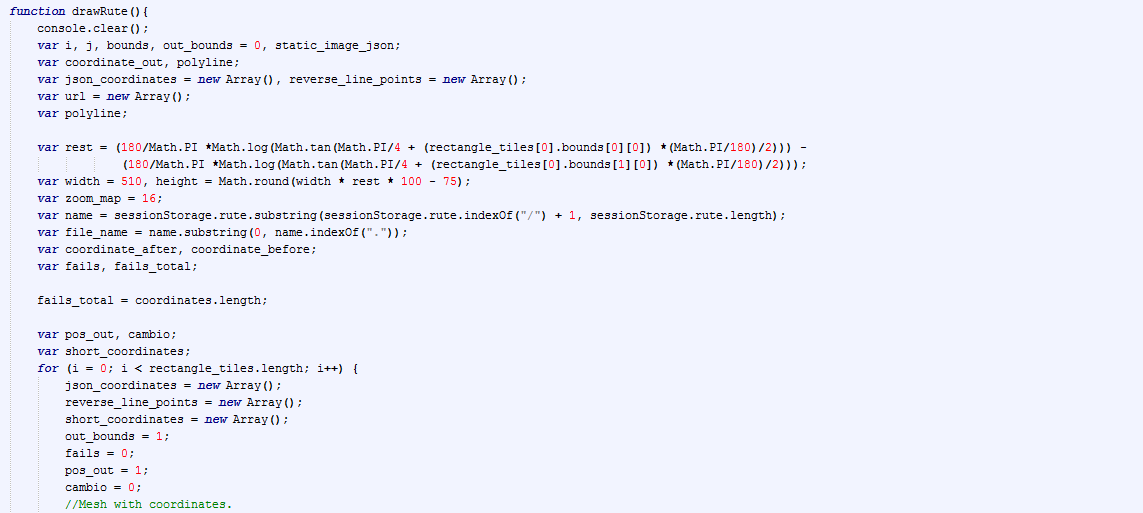
* Imágenes de mapas con ruta mediante codificación de líneas *Encoded polylines*.

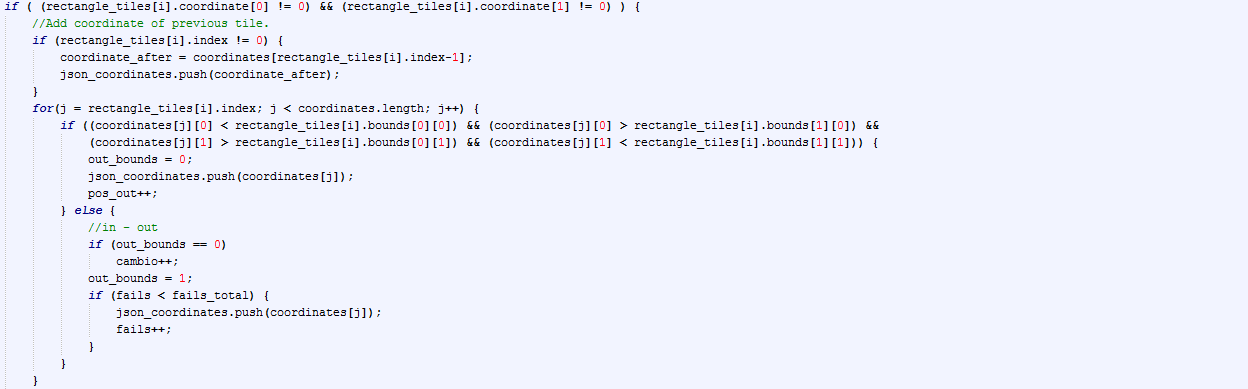


Para obtener la textura de cada tile con su ruta podemos usar uno de los dos últimos métodos. En un principio utilice el método de codificación de líneas pero surgían muchos problemas por este método ya que había que usar una librería de *google* y a veces no se dibujaba la ruta correctamente, al final decidí usar el método pasando la ruta en formato GeoJSON.

Con ambos métodos cargo una nueva página pasando en el *url* toda la información y en ambos si se supera una cantidad aproximada de 120 coordenadas no se devuelve correctamente la imagen del mapa, de este modo cuando un tile tiene un número muy grande de coordenadas aplico la función *fixCoordinates()* que reduce la cantidad de coordenadas sin perder el sentido de la ruta.

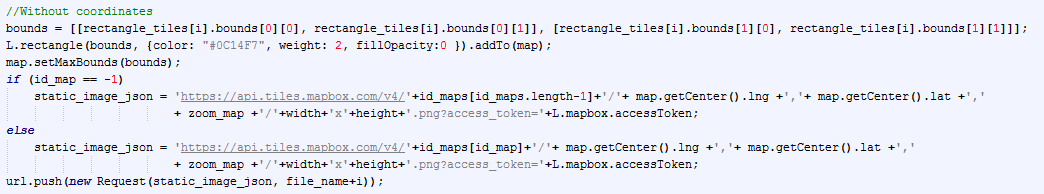
Para obtener las texturas de cada tile he creado la función *drawRute()*, esta función recorre todos los tiles almacenados en el arreglo *rectangle\_tiles* y por cada tile se realiza lo siguiente:

**

1. Compruebo que el tile corresponde a la ruta y no a los tiles que cree como relleno, por cada uno de ellos:  
   1. Si no es el primer tile donde se inicia la ruta añado la coordenada anterior para que la línea de la ruta no tenga cortes entre tiles, de este modo no perdemos el trazado original.
   2. En *rectangle\_tiles* tengo el índice donde se inicia la ruta en el tile con respecto al arreglo de coordenadas, a partir de esa posición compruebo las coordenadas que pertenecen a ese tile y las almaceno. Compruebo que las coordenadas estén dentro de los límites de cada tile. En el estudio para obtener la ruta pensando en todas las rutas posibles que puede realizar un usuario me encontré con los siguientes problemas:  
      1. Si la ruta en el tile no tiene cortes el algoritmo inicial funciona sin problemas. Cuando hablo de cortes me refiero a que la ruta sale del tile y vuelve a entrar, para ello establecí una tasa de fallo que sigue almacenando la ruta hasta que lleguemos a un número elevado de fallos. Este paso es necesario porque a la hora de dibujar la ruta en Mapbox no puede tener cortes ya que no dibujaría la ruta tal cual es.
      2. Limitación a la hora de crear texturas con Mapbox por cualquiera de los métodos descritos tenemos un límite de coordenadas.
      3.  Almacenar más coordenadas de las debidas. Cuento la cantidad de posiciones que se ha salido, de este modo luego puedo saber la cantidad de coordenadas que pertenecen a la ruta, esto lo hago ya que al principio almacenaba todas las coordenadas lo cual al final provocaba fallos.
   3. Si no han habido cambios significa que al ruta se ha mantenido dentro del tile, si el valor de la variable cambio es 1 significa que ha salido una sola vez, por lo tanto, acortamos las coordenadas hasta la última posición antes de salir.
   4. Invierto las coordenadas para que sean reconocidas por *Mapbox* para crear la ruta.
   5. Creo el objeto GeoJson para enviar por *url* para la creación de la textura del mapa con la ruta. Codifico el objeto como JSON con la función *JSON.stringify()* y codifico nuevamente el para poder ser pasado por *url* con la función *encodeURIComponent*.
   6. Creo el enlace con el cual luego descargaremos la imagen correspondiente para cada tile.
   7. Para lograr un mejor orden en el código almaceno el enlace y su índice en un arreglo para luego crear las texturas con una sola llamada *ajax*, al realizar cada llamada algunas texturas no se creaban y a la hora de unir todas las texturas quedaban espacios vacíos*.*



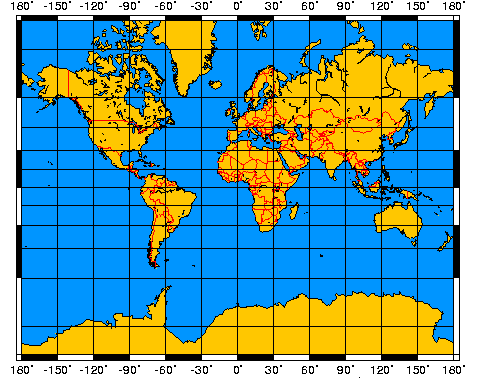
1. Compruebo que el tile que no corresponde a la ruta tenga o no coordenadas. Durante las pruebas encontré tiles de relleno que contienen coordenadas de la ruta.  
   1. Realizo los mismos pasos mencionados anteriormente en el punto 1 por si el tile tiene coordenadas.
2. Creo la imagen del tile sin coordenadas y la almaceno.



Con el arreglo envío todas las texturas mediante *ajax* y se crean las texturas en el servidor mediante *php*. Al principio las imágenes las cree con un tamaño definido para todos los lugares por igual dando errores dependiendo de la zona, para que no suceda eso se usó el algoritmo de *Proyección de Mercator* que hace una relación entre el ancho de la imagen junto con la diferencia de latitud del tile.Cuando el servidor termine de crear las texturas se cargará la página *PFCMyMesh.html* donde aparecerá el modelado en 3D con *threejs*.

Asumiendo que la tierra tiene forma esférica (elipsoide) se busca transformar del sistema longitud-latitud al sistema cartesiano, ese es el principio de la Proyección de Mercator.





## Creación de Mesh

Con las texturas en el servidor nos queda crear la geometría y añadir la ruta recorrida. En esta etapa del proyecto es donde mezclaremos el uso de la librería *Cesium* con los datos de las elevaciones y *threejs* para crear el terreno de la ruta recorrida.

La página *PFCMyMesh.html* a su vez trabaja con varios ficheros externos en JavaScript ubicados en la carpeta *js*, */js/my\_cesium\_mesh.js*, */js/clases.js, /js/functions.js* y */js/Cesium.js* este último es el fichero ofrecido por Cesium.

Creamos las variables locales que usaremos y esperamos hasta lograr la conexión a *Cesium.*

### Obtención de ruta

Cuando la conexión a Cesium esté lista llamamos a la función *getRute()* con la cual obtendremos las coordenadas de la ruta y llamaremos a la función *load()* que se encargará de crear la escena en 3D, crear algunos controles gráficos y crear el terreno de la ruta.

### Creación de escena y controles gráficos

Llamaremos a la función *loadThreeJS()* y a la función *createGUI()*. La función loadThreeJS creará la escena, posicionará la cámara y aplicará el renderizado al añadir el terreno. La función *createGUI()* se encargará de añadir los botones para volver al inicio, visualizar el mapa y exportar el modelo para ser impreso en *Shapeways*.

### Información tiles

Realizaremos los mismos pasos del apartado anterior para obtener los datos, usando la función *checkTile(coordinates, level)* creamos el arreglo *info\_tiles*,la función lleva como parámetros de entrada las coordenadas y el nivel de Mapbox. El arreglo guarda objetos *InfoTile* explicados anteriormente.

En este punto tenemos los tiles que corresponden a la ruta y como se mencionó antes la idea es crear un terreno rectangular, por lo tanto, llamamos a la función *createRectangle(info\_tiles, level)*. Está función nos devuelve un arreglo que contiene objetos *InfoTile* correspondientes a la ruta y añade nuevos valores completando un rectángulo, como parámetros de entrada la función recibe los tiles del recorrido y el nivel de Mapbox. El resultado de la función lo guardaremos en la variable *rectangle\_tiles*.

### Información del terreno

En el arreglo *rectangle\_tiles* tenemos toda la información necesaria para crear el terreno, con un simple bucle solicitaremos a *Cesium* los datos de cada tile con la función *requestTileGeometry()* mencionada anteriormente, en este punto explicaremos los datos que hemos utilizado de *Cesium* cuando realizamos la petición por cada tile. Además iremos almacenando cada petición en un arreglo para controlar el programa debido a que *requestTileGeometry()* es una función asíncrona, por lo tanto, solo se creará el terreno cuando hayamos procesado todos los tiles.

Por cada llamada de la función *requestTileGoemetry()*



### 

Para la creación del terreno usaremos los arreglos:

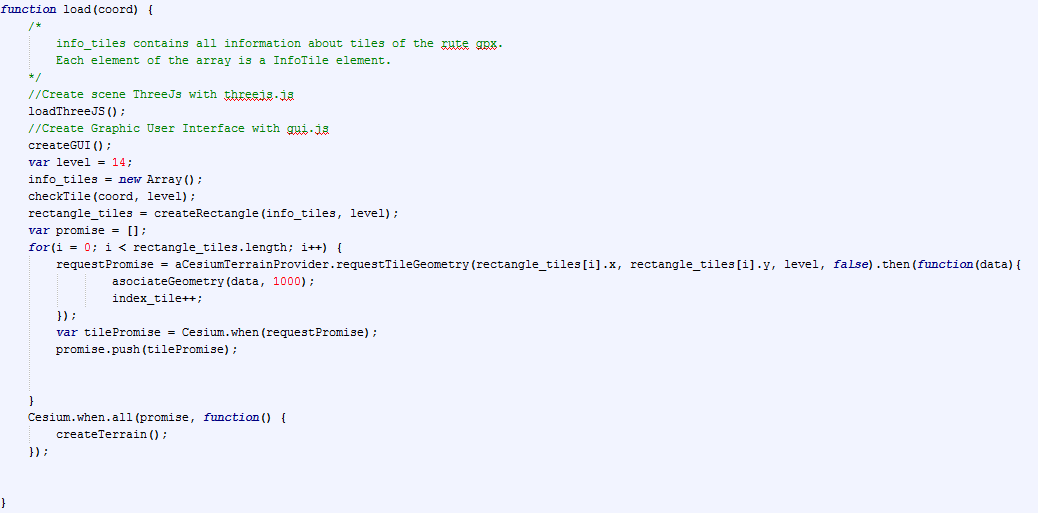
* *\_uValues*: contiene el valor de las coordenadas horizontales en el tile, si es 0 indica que es el borde Oeste del tile. Cuando el valor es 32767 nos indica el borde Este del tile. Para el resto de los valores este vector es una interpolación lineal entre las longitudes de los bordes del este y oeste del tile.
* *\_vValues*: contiene el valor de las coordenadas verticales en el tile, si es 0 indica que es el borde Sur del tile. Cuando el valor es 32767 nos indica el borde Norte del tile. Para el resto de los valores este vector es una interpolación lineal entre las longitudes de los bordes del este y oeste del tile.
* *\_heightValues*: contiene las alturas del tile, si es 0 indica que es la menor altura del tile. Cuando el valor es 32767 nos indica la mayor altura del tile. Para el resto de los valores este vector es una interpolación lineal entre la mínima y máxima altura. Para obtener las alturas debemos tener en cuenta los valores *\_minimumHeight* y *\_maximumHeight* y establecer una regla de tres para obtener las alturas reales.
* *\_indices:* específica como los vértices van unidos entre sí en triángulos, cada tres índices especifican un triángulo, este valor es el más importante a la hora de crear los terrenos.

Los tres primeros arreglos son almacenados conjuntamente en el arreglo *\_quantizedVertices* agrupados en vector de 3 elementos. Con los datos mencionados anteriormente ya tenemos todo lo que necesitamos para crear cada tile que formará el terreno. La creación de cada tile es de la siguiente manera:

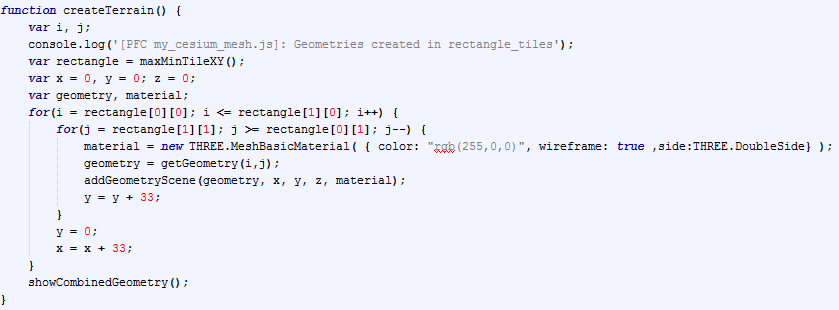
1. Creamos una geometría con *threejs*.
2. Añadimos los vértices a la geometría. Cada vértice será el valor de la coordenada horizontal *\_uValues*, coordenada vertical *\_vValues* y altura *\_heightValues* escalada usando los valores de altura máxima y mínima de cada tile.
3. Creamos las caras de la geometría usando la variable *\_indices* obteniendo el terreno.
4. Almacenaremos la geometría creada en el atributo *geometry* del arreglo *rectangle\_tiles*.

Por cada iteración del bucle llamaré a la función *asociateGeometry(data, 1000)*, la cual recibe como parámetros de entrada toda la información de *Cesium* y un valor para escalar los datos. Tendremos una variable global *index\_tile* la cual hace referencia a que tile estamos tratando en la función *asociateGeometry()*, se hace todo esto ya que el resultado de la función *requestTileGeometry()* es asíncrono.

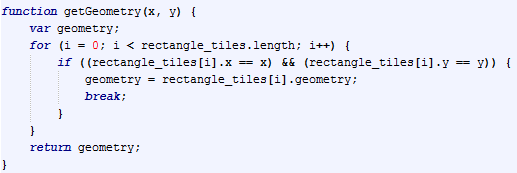
Al terminar el bucle en el arreglo *rectangle\_tiles* tendremos por cada tile su geometría, el bucle terminará cuando todas las peticiones hayan devuelto la información solicitada ya que un principio no todas las peticiones se completaban y algunas geometrías no se creaban.

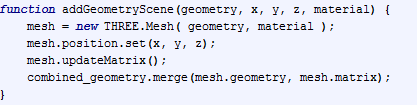


### Creación del terreno

Ahora debemos unir todas las geometrías para obtener una única geometría que será nuestra ruta recorrida. La función que se encargará de crear el terreno es *createTerrain()*.

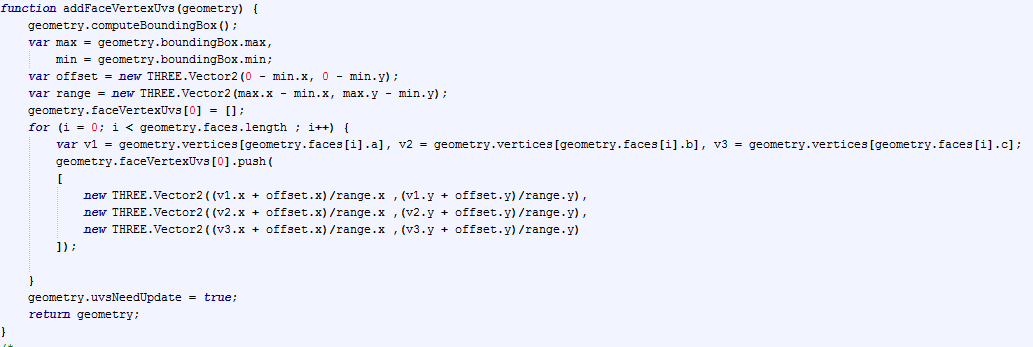
*createTerrain()* creará el terreno realizando las siguientes acciones:

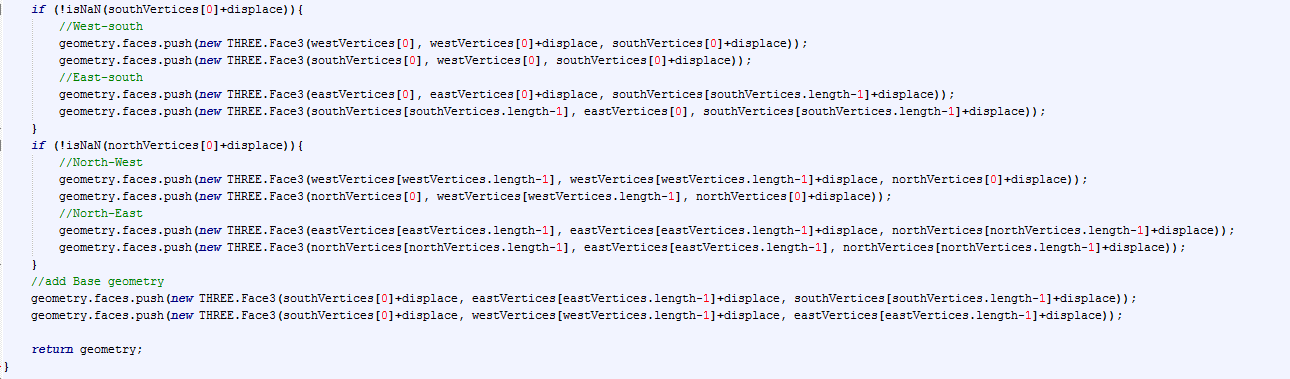
1. Obtener el máximo y menor tile con la función *maxMinTileXY()* para poder unir cada geometría de un tile con su vecina y recorrer todos los tiles de forma ordenada.
2. Con los valores obtenidos de la función *maxMinTileXY()* crearemos un bucle que recorra todos los tiles que pertenecen a la ruta.
3. Utilizare una serie de variables para la localización de cada geometría en la escena, para obtener la geometría usaremos la función *getGeometry()* que nos devuelve la geometría del tile que indiquemos. Usaremos la función *addGeometryScene()* para crear una geometría que ira uniendo cada geometría de cada tile hasta obtener una geometría final que corresponderá a la ruta. Para ello usaremos la función *merge()* de *threejs* que une mallas, una malla es la unión de una geometría con una textura que en este caso será una textura básica ya que al final añadiremos la textura de la ruta completa.  
     
   



1. Actualizaremos los valores de las variables por cada iteración del bucle.
2. Finalmente llamaremos a la función *showCombinedGeometry()* la cual creará la textura para la geometría y la malla que será visualizada en la escena.

A continuación explicaré la función *showCombinedGeometry()* :

1. Usamos la función maxMinTileXY() anteriormente mencionada, al tener los máximos y mínimos de la ruta recorremos todos los tiles para obtener cuantas filas y columnas tendrá nuestra ruta completa. Se hace este proceso para unir todas las imágenes de los tiles creando una sola imagen que usaremos como textura a nuestra geometría final.
2. Realizamos una petición asíncrona al servidor enviando como datos la cantidad de columnas, cantidad de filas, nombre del fichero en uso y la medida de alto que debería tener cada imagen algoritmo de *Proyección de Mercator*.
3. Cuando la imagen de toda la ruta ha sido creada el servidor nos devuelve su ubicación en el servidor.
4. Creamos la textura para nuestra geometría final.
5. Como es una geometría creada desde cero debemos indicar como la textura debe implantarse en cada cara, para ello usamos la función *addFaceVertexUvs()*.
6. Para crear el volumen del terreno debemos añadir una base, para ello usamos la función *addBase().*

**

1. Creamos el material indicando la textura que usará y que la textura aparezca en ambos lados de la geometría.
2. Creamos la malla que es la unión de la geometría y el material.
3. Especificamos su ubicación en la escena. Esta ubicación debe ser dinámica siempre para que el usuario pueda interaccionar correctamente con el terreno.
4. Rotamos la malla, ya que su creación ha sido modificando todos los vértices.
5. Añadimos la malla a la escena.

Código completo de la función *ShowCombinedGeometry.*

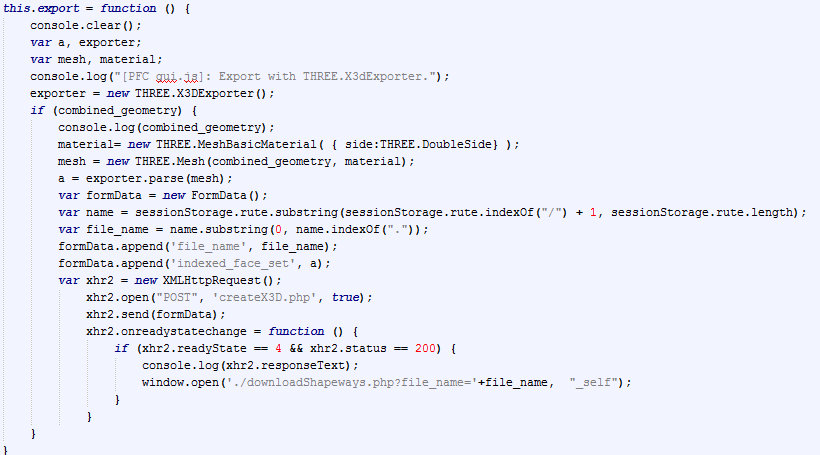
## Creación x3d y envío a impresora 3D

En este apartado se explicará cómo se crea el objeto y como debe ser enviado a *Shapeways.*

### Creación del modelo

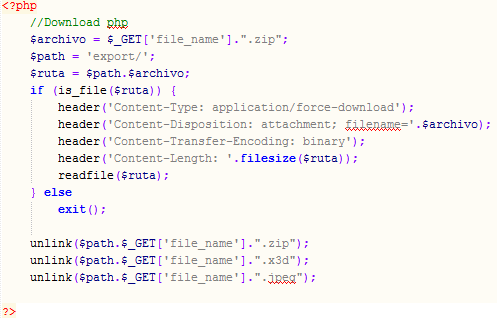
Para subir un objeto debemos crear un *zip* que incluya el objeto en formato *x3d* y su material en algún formato de imagen, para ello usaremos la librería *x3dExporter.js* con la cual transformamos nuestra malla y la añadimos a nuestro *x3d* de la siguiente manera:

* Debemos crear el objeto exporter.
* Aplicar la función *parse(malla)* pasando como argumento la malla que une nuestra geometría y material de la ruta.
* Ahora enviamos la información mediante *ajax* a la página *createX3D.php* que se encarga de crear el objeto *x3d*. La imagen ya la tenemos en el servidor ya que la usamos para crear el terreno en el navegador.
* El servidor envía la respuesta del servidor con el cual cargamos la página *downloadShapeways.php* que se encarga de generar el *zip* uniendo el *x3d* y la imagen el cual es descargado en nuestro ordenador.



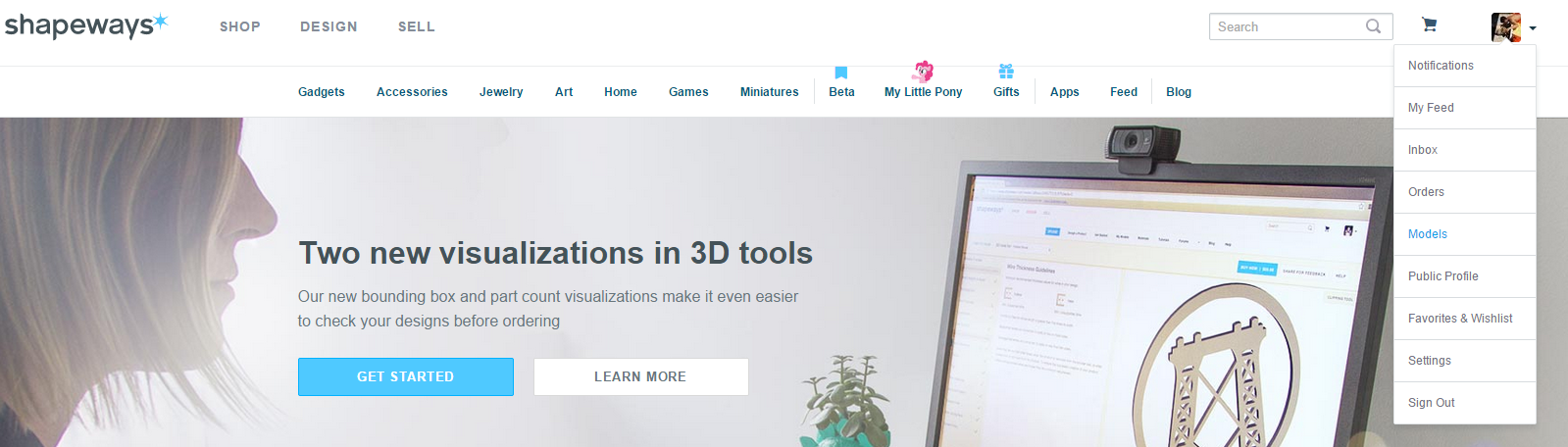
**Fichero *createX3D.php*

Fichero *downloadShapeways.php*

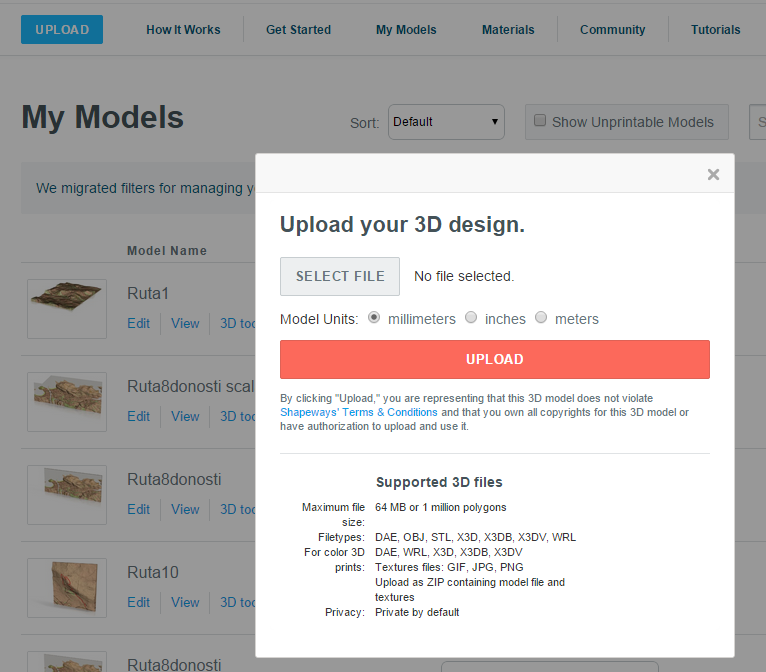


### Subir el modelo

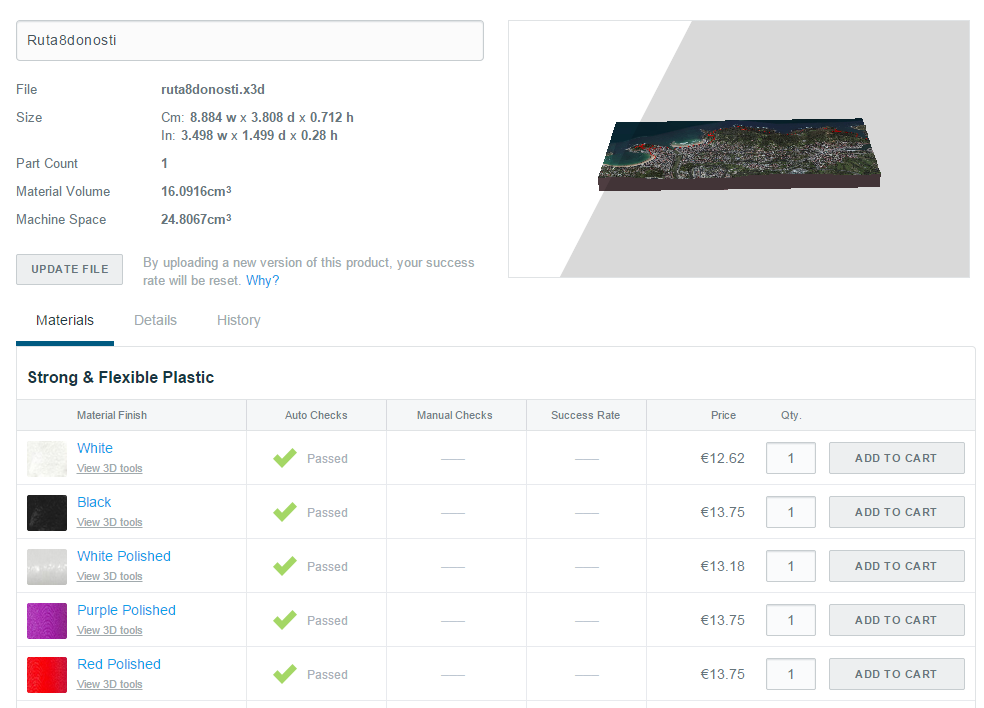
Ya creado el objeto solo nos queda subir el *zip* a *Shapeways*, logeados como usuarios nos dirigimos a nuestro perfil y seleccionamos la opción *Models*.



Ya en nuestros modelos en la esquina superior derecha nos encontramos con el botón *Upload* el cual nos permitirá subir nuestro modelo.



Mientras el fichero es procesado *Shapeways* nos indica sus características como el nombre del fichero, el tamaño, su volumen entre otras características.Cuando el objeto ha sido cargado por completo a la derecha de la descripción aparecerá una imagen con el modelo cargado y hacia abajo todas las opciones de material, referencia a si el objeto cumple los estándares de *Shapeways*.



# Resultados

## Aplicación Web

## Impresiones 3D

## Problemas y soluciones

# Conclusiones y líneas futuras

# Bibliografía

Impresión en 3D

<http://www.fierasdelaingenieria.com/fabricacion-por-adicion-el-camino-hacia-la-proxima-revolucion-de-la-tecnologia-aeroespacial/>

<http://es.wikipedia.org/wiki/Impresi%C3%B3n_3D>

Proyección de mercator.

<http://es.wikipedia.org/wiki/Proyecci%C3%B3n_de_Mercator>

1. <http://www.instructables.com/id/3D-Print-Your-Trek-in-color/> [↑](#footnote-ref-1)
2. [http://earthexplorer.usgs.gov](http://earthexplorer.usgs.gov/) [↑](#footnote-ref-2)
3. <http://www.visualizationsoftware.com/3dem.html> [↑](#footnote-ref-3)
4. x3d es una extensión para objetos 3D. [↑](#footnote-ref-4)
5. [http://www.micromouse.ca](http://www.micromouse.ca/) [↑](#footnote-ref-5)
6. [http://www.blender.org](http://www.blender.org/) [↑](#footnote-ref-6)
7. xml para representar datos geográficos en 3D. [↑](#footnote-ref-7)
8. 3D en el navegador. [↑](#footnote-ref-8)
9. Lenguaje interpretado: lenguaje de programación que esta diseñado para ser ejecutado por medio de un intérprete. [↑](#footnote-ref-9)
10. DOM: Document Object Model. [↑](#footnote-ref-10)
11. Ajax : Asynchronous JavaScript and Xml [↑](#footnote-ref-11)
12. WWWC World Wide Web Consortium, consorcio internacional que realiza recomendaciones a la red informática mundial conocida como World Wide Web. [↑](#footnote-ref-12)
13. 3D en el navegador. [↑](#footnote-ref-13)
14. <https://www.mapbox.com/> [↑](#footnote-ref-14)
15. Open Source [↑](#footnote-ref-15)
16. <http://cesiumjs.org/downloads.html> [↑](#footnote-ref-16)