UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Visor 3D v2.0

Presentado por Jose Manuel Moral Garrido
en Universidad de Burgos — 13 de noviembre de 2017
Tutores: José Francisco Díez Pastor, Álvar Arnaiz González

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

Cabe mencionar que no he realizado una dedicación total al trabajo de final de grado, sino que ha sido una dedicación parcial ya que me encuentro trabajando al mismo tiempo, por lo tanto, en la mayoría de los *sprint* se han cerrado la mayor parte de las tareas el mismo día ya que mi horario laboral cambia cada semana y el día o los días que tengo libres los aprovecho al completo avanzando en el proyecto.

A.2. Planificación temporal

A continuación detallaremos los diferentes objetivos que se han establecido para cada *sprint* y, a su vez, el progreso obtenido en los mismos. Para ello hemos utilizado una metodología ágil denominada *SCRUM* [1]. Emplearemos a su vez el gestor de tareas provisto por GitHub y generaremos gráficos *burndown* para el seguimiento de los *sprint*, los cuales son provistos por la extensión *ZenHub*.

Sprint 1 - 18–92017/24-09-2017

En este *sprint* se pretenden instalar las herramientas necesarias para la realización del proyecto, así como la lectura y comprensión de la *memoria* y *anexos* del proyecto de partida [2]. A su vez, se pretende probar los métodos creados de la aplicación de partida para comprobar su correcto funcionamiento.

Podemos observar el progreso del sprint en la figura A.1

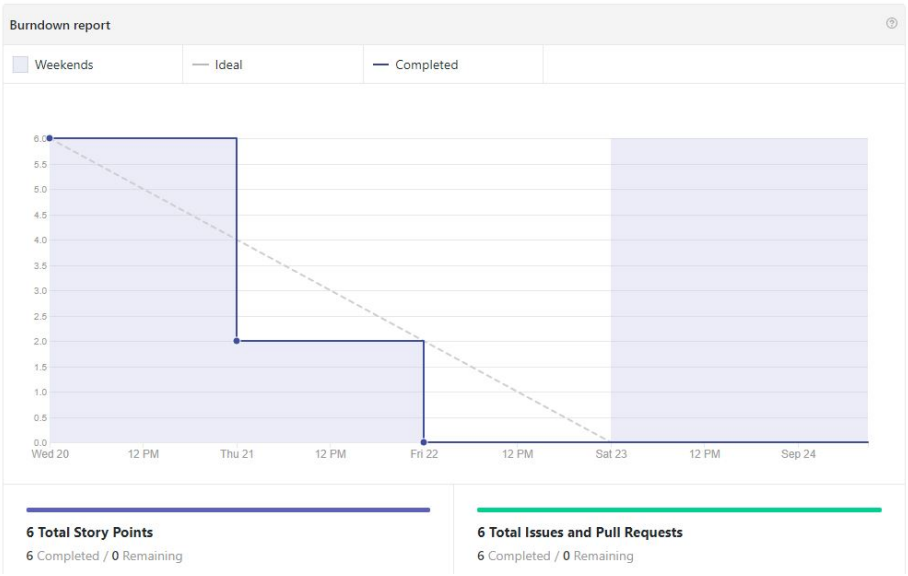

Figura A.1: Progreso en el sprint 1

Sprint 2 - 25-09-2017/01-10-2017

En este *sprint* se pretende subir a mi repositorio propio toda la información necesaria para continuar el proyecto y solucionar los fallos encontrados la semana anterior en los distintos métodos de la aplicación. Además se prentende conocer la estructura completa del proyecto con el fin de agilizar las tareas en el momento de la búsqueda de los puntos de la aplicación a corregir o modificar.

Podemos observar el progreso del sprint en la figura A.2

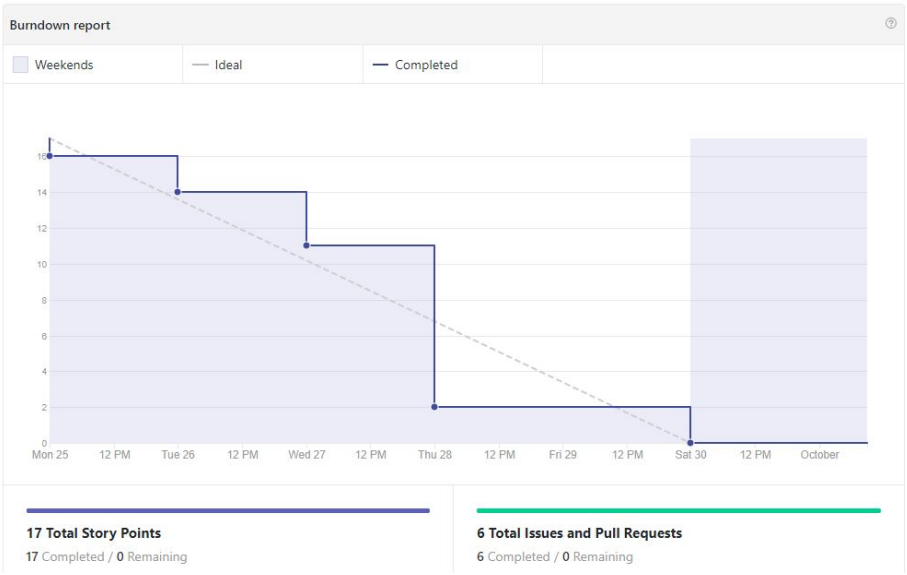

Figura A.2: Progreso en el sprint 2

Sprint 3 - 02-10-2017/08-10-2017

En este *sprint* se pretende cambiar la manera que tiene la aplicación de cotejar los roles de los usuarios (mediante base de datos) a ser cotejados y asignados mediante la *API* de UBUVirtual y e intentar interar la aplicación en un servidor público como es *Heroku* [3]

Podemos observar el progreso del sprint en la figura A.3

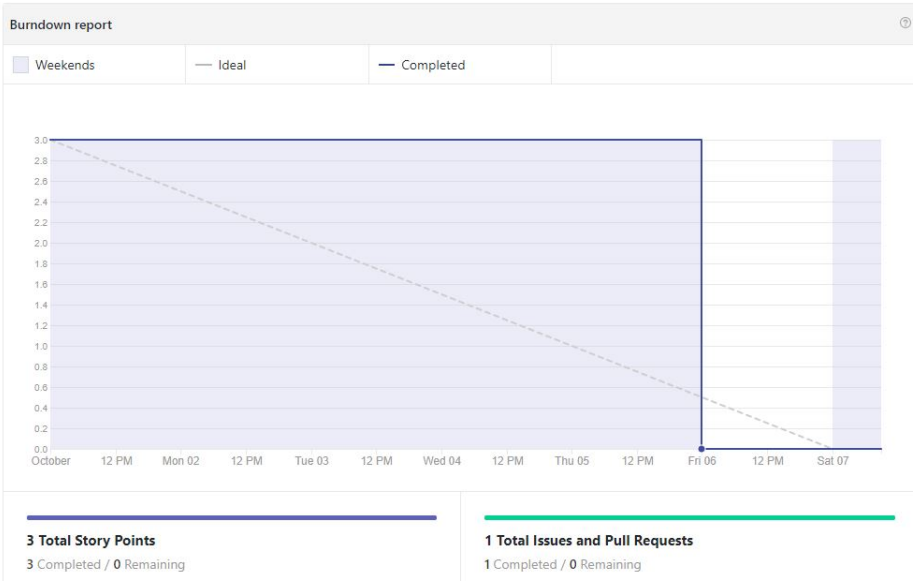

Figura A.3: Progreso en el sprint 3

Sprint 4 - 09-10-2017/15-10-2017

En este *sprint* se pretende continuar con el intento de integración de la aplicación en *Heroku*. A su vez, se pretende albergar los modelos de manera privada para que los alumnos no puedan acceder a ellos nada más que mediante la plataforma o el medio optado. También instalaremos *Moodle* de manera local para poder realizar pruebas sin depender de un tutor que pueda facilitarnos asignaturas, asignación de roles, subida de recursos, etc.

Podemos observar el progreso del sprint en la figura A.4

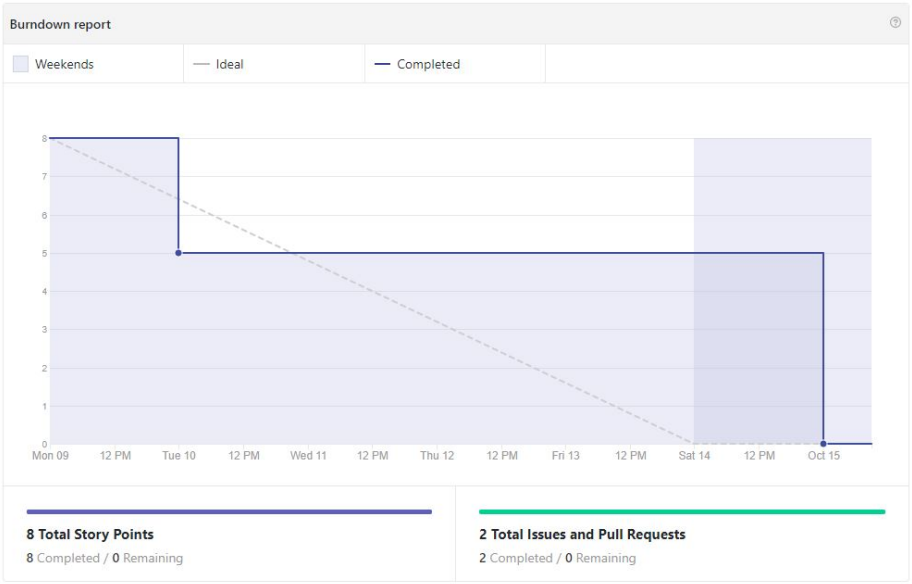

Figura A.4: Progreso en el sprint 4

Sprint 5 - 16-10-2016/22-10-2016

En este *sprint* se pretende instalar de nuevo *Moodle* ya que el instalado en el *sprint* anterior no funcionaba correctamente. A su vez continuamos con la integración de la aplicación en *Heroku* (tarea que se está alargando por dos motivos: errores en la estructura de la aplicación y que nos encontramos a la espera de que la UBU nos proporcione un servidor que cumpla ciertos requisitos)

Podemos observar el progreso del sprint en la figura A.5

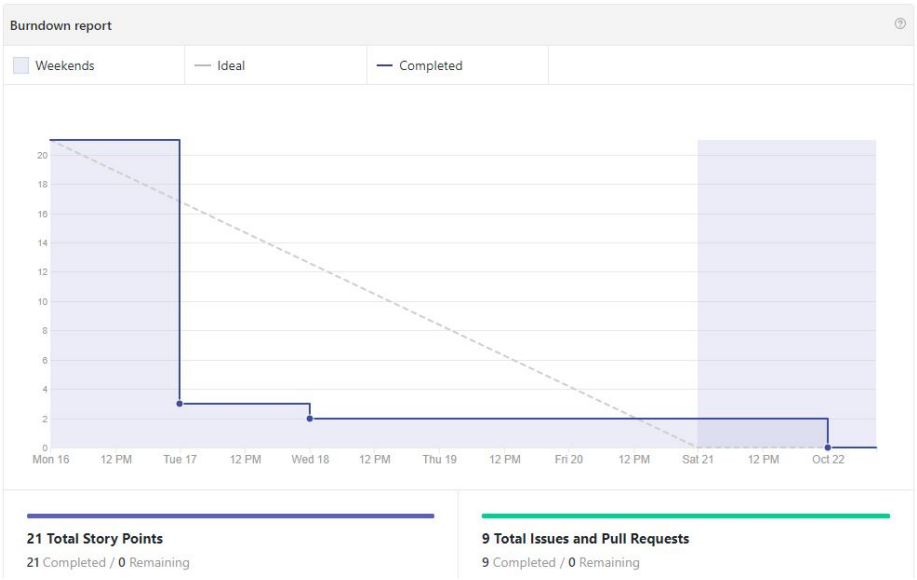

Figura A.5: Progreso en el sprint 5

Sprint 6 - 24-10-2016/29-10-2016

En este *sprint* se pretende revertir al aplicación a un punto anterior ya que podemos decir que el *sprint* anterior ha sido inservible. Esta vuelta atrás la realizaremos de manera manual ya que si la realizamos mediante los *commit*, perderemos unos cambios que no queremos perder. Este cambio manual también involucra cambiar las dependencias que eran necesarias para el lanzamiento de la aplicación en Heroku (*sprint 5*), ya que en este *sprint* hemos sustituido un servidor público como es Heroku por el proporcionado por la Universidad de Burgos. Por otra parte, tendremos que ejecutar la aplicación en el servidor provisto por la *UBU* (*arquimedes*), realizando a su vez una comparativa de los distintos servidores posibles para desplegar nuestra *API*

Podemos observar el progreso del sprint en la figura A.6

Figura A.6: Progreso en el sprint 6

Sprint 7 - 30-10-2016/06-11-2016

En este *sprint* se pretende trabajar los aspectos relacionados con la seguridad de los modelos subidos al servidor. Con esto queremos decir que en este *sprint* nos dedicaremos a realizar una encriptación de los modelos para que únicamente los usuarios autorizados sean capaces de visualizar el modelo tal y como es. Esto se realiza con el fin de conservar la privacidad de los modelos ya que estos son únicos. La encriptación se realizará en el momento de la subida del modelo a la aplicación alojada en el servidor, y justo en el momento de visualizar el modelo se realizará su desencriptación. La encriptación se hará para los modelos *PLY* tanto en formato **binario** como en **ascii**, mientras que la desencriptación se hará únicamente desde formato *ascii* para así ahorrar tiempo, complejidad y la programación de dos desencriptadores diferentes. A su vez, se realizará un estudio de los tiempos de carga de los modelos debido a las operaciones realizadas para proceder con su encriptación y desencriptación.

En este *sprint* no se ha conseguido integrar el *script* de desencriptación en el cargador de modelos *PLY*, por lo que será una tarea prioritaria en el siguiente *sprint*.

Podemos observar el progreso del sprint en la figura A.7

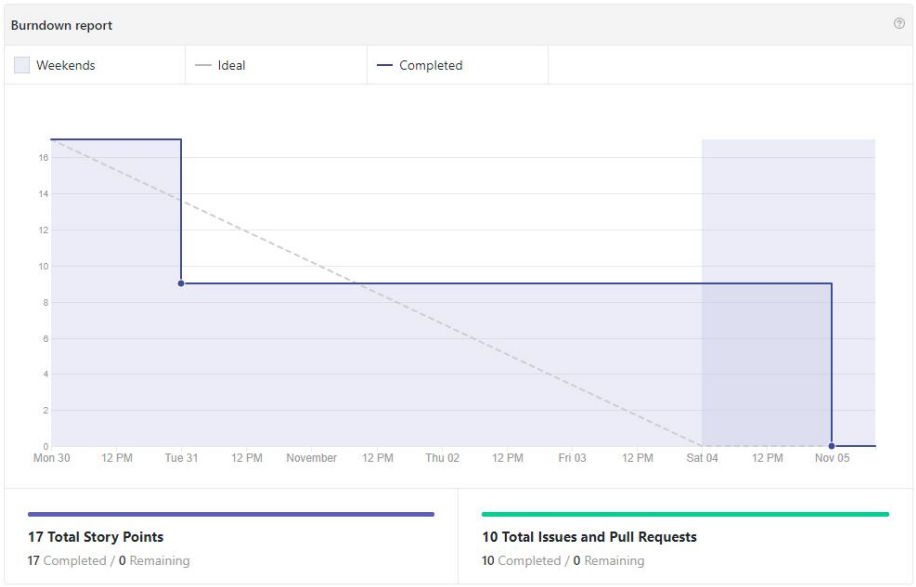

Figura A.7: Progreso en el sprint 7

Sprint 8 - 06-11-2016/12-11-2016

En este *sprint* trataremos de terminar de hacer funcionar la funcionalidad de encriptación y desencriptación de nuestro visor, ya que en el *sprint* anterior tuvimos problemas con el tema de los números en coma flotante, lo cual será mencionado en el *Manual del Programador*. A su vez, dedicaremos este *sprint* a documentar al completo los pasos avanzados hasta el momento, así como solucionar los errores pertinentes a la hora de compilar *LaTeX*. Con el fin de mejorar los tiempos de carga del visor así como la precisión de los modelos a la hora de encriptarlos y desencriptarlos, se estudiarán diferentes métodos de encriptación (vértices, caras, etc). También realizaremos la configuración *VPN* correspondiente para poder conectarnos al servidor proporcionado por la Universidad de Burgos desde otra red diferente a la de la misma.

A.3. Estudio de viabilidad

Viabilidad económica

Viabilidad legal

Apéndice B

Especificación de Requisitos

B.1. Introducción

B.2. Objetivos generales

B.3. Catalogo de requisitos

B.4. Especificación de requisitos

Apéndice C

Especificación de diseño

C.1. Introducción

C.2. Diseño de datos

C.3. Diseño procedimental

C.4. Diseño arquitectónico

Apéndice D

Documentación técnica de programación

D.1. Introducción

D.2. Estructura de directorios

D.3. Manual del programador

Instalación y configuración de Moodle como API Rest

Como se menciona en los aspectos relevantes de la memoria, es necesario configurar *Moodle* para poder utilizarlo como una *API Rest*. A continuación se detalla como configurar la *API*:

En primer lugar debemos tener un usuario con el rol de administrador de la plataforma para poder acceder a la *Administración del sitio* para poder activar los servicios web que por defecto vienen desactivados. Deberemos dirigirnos a *Administración del sitio–Características avanzadas* y habilitar los servicios web. Una vez hecho esto deberemos activar el protocolo *Rest*, que básicamente es el protocolo seguido por una *API Rest*, el cual se accede mediante *Administración del sitio – Extensiones – Servicios Web – Administrar protocolos* y habilitamos dicho protocolo.

A su vez, para que podamos acceder a dichas funcionalidades, además de tener que estar el servicio web y el protocolo activado, lo usuarios deben tener una ficha o *token* el cual los identifique de manera única. Para generar desde nuestra *API* dichos *tokens* nos dirigiremos a *Administración del sitio*

– *Extensiones* – *Servicios web* – *Administrar tokens* y ahí generaremos los tokens para los usuarios. De esta manera, cada usuario tendrá un identificador único para realizar las peticiones correspondientes [**?**].

Tratamiento de los roles de los usuarios

Como se menciona en los aspectos relevantes de la memoria, inicialmente se tenía una idea inicial de obtener los roles de los usuarios mediante la comprobación de la base de datos en la que estarían definidos dichos roles, pero posteriormente se decidió obtener dichos roles directamente de *UBUVirtual*.

Para ellos utilizamos las funciones proporcionadas por *Moodle* para realizar peticiones a nuestra *API Rest* (sección **??**), que en este caso es UBUVirtual. En dicho listado de funciones ([**?**]) encontramos la función *core enrol get enrolled users*, la cual nos permitirá conocer los usuarios de la asignatura, así como su rol en la misma y más información variada de cada uno de los participantes. Dicha función nos muestra esta información en forma de diccionario *JSON* desde el que buscaremos al usuario correspondiente para así conocer su rol en la asignatura correspondiente. Dicha información nos es presentada con la siguiente estructura D.1:

Figura D.1: Estructura de la información proporcionada por la API en formato JSON

Como se puede apreciar en el campo *roles* nos encontramos con el rol correspondiente del usuario en cuestión, que en este caso es *Profesor* y el id de dicho rol es *3*

Obtención de los modelos

Ya que la idea inicial era la de obtener los modelos a través de *UBUVirtual* mediante recursos, cabe mencionar como conseguimos obtenerlos aunque la idea no prosperase.

Para ello, recurrimos de nuevo a las funciones *API Rest* siendo esta vez la función *mod resource get resources by courses* la elegida [**?**]. Dicha función nos ofrece la información de los recursos presentes en la *API* de UBUVirtual en los cursos correspondientes. En el caso de no seleccionar un curso en

concreto, nos devuelve cada uno de los recursos a los que dicho usuario puede acceder. La información resultante tiene la siguiente estructura D.2:

Figura D.2: Estructura de la información proporcionada por la API en formato JSON

De esta manera podemos acceder al nombre de recurso con su correspondiente extensión y comprobar que es del curso correspondiente mediante el campo *course*.

Pero posteriormente, la Universidad de Burgos nos proporcionó un servidor privado, de nombre .ªrquimedes.en el que podemos desplegar nuestra *API* sin necesitar por ello todo lo mencionado anteriormente acerca de los albergar los modelos como recursos de *UBUVirtual*, ya que podremos albergarlos en nuestro servidor.

D.4. Encriptado y desencriptado de los modelos

Para otorgar seguridad a los modelos, hemos tenido que encriptar los mismos de manera que el modelo que se alberga en el servidor esté modificado

de tal manera que alguien ajeno a la *API* que quiera obtener datos o modificar los datos guardados de los modelos sea incapaz.

Para poder realizar una modificación que posteriormente podamos deshacer, lo primero es la generación de números aleatorios de tal manera que, conociendo la semilla inicial, obtengamos los mismos números para encriptar y desencriptar los modelos. Para obtener dichos números aleatorios hemos introducido en el código una función la cual dada un número, nos devuelve otro, con lo cual realizando esta operación un cierto número de veces, obtendremos una secuencia de números .ªleatorios"(se encuentra entrecomillado porque los valores son aleatorios, pero si se conoce la semilla inicial siempre obtendremos la lista de valores en el mismo orden)[1].

Obtención de los números aleatorios

La manera de obtener números aleatorios en el caso de los vértices es la siguiente: *7 * número aleatorio anterior % 101*, mientras que en el caso de las caras obtendremos los valores aleatorios de la siguiente manera: *7 * número aleatorio anterior % 11*. Vemos que la diferencia reside en el valor máximo que puede alcanzar el número aleatorio.

Para la modificación de los valores de los vértices

Llegados a este punto tendremos que decidir cómo modificar los valores de los modelos, para lo cual hemos realizado un estudio de los tiempos que tarda el modelo en ser encriptado. La operación a realizar en cada caso será determinada por el tipo de los valores que se vayan a modificar.

Siendo la manera de generar los números aleatorios la mencionada en la sección D.4 y la manera de modificar los valores de los vértices: *valor del vértice * (2 * número aleatorio obtenido)*, estaríamos en la encrucijada de elegir la cantidad de operaciones a realizar debido al gran volumen de datos que queremos modificar, por lo tanto hemos obtenido:

Para los modelos ASCII:

- Si modificamos todos los vértices que componen al modelo obtenemos un tiempo de: **12.013121366500854 segundos**
- Si modificamos solamente los vértices que ocupan posiciones pares del modelo (la mitad de operaciones) obtenemos un tiempo de: **10.27306342124939 segundos**

[1] https://cdsmith.wordpress.com/2011/10/10/build-your-own-simple-random-numbers/

Para los modelos Binarios:

- Si modificamos todos los vértices que componen al modelo obtenemos un tiempo de: **2.7190418243408203 segundos**
- Si modificamos solamente los vértices que ocupan posiciones pares del modelo (la mitad de operaciones) obtenemos un tiempo de: **2.048550605773926 segundos**

Una vez conocidos estos datos, decidimos modificar únicamente los valores que ocupan posiciones pares ya que la encriptación del modelo es suficiente para conservar su seguridad como podemos observar a continuación y el tiempo de codificación es menor:

Teniendo un modelo inicial como el de la figura D.3:

Figura D.3: Modelo de partida

Obtendremos un modelo encriptado como el mostrado en la figura D.6:

Figura D.4: Modelo de encriptado

Como se puede apreciar, el modelo encriptado es lo suficientemente difuso como para poder obtener mediciones o datos del mismo en caso de que este fuera robado de la carpeta de almacenamiento de la aplicación. Pero a partir de aquí nos surge el problema relacionado con el redondeo de los decimales, así como de la cantidad de decimales que se devuelven al realizar un *casteo* a otra clase (por ejemplo de *str a float* en *Python*).

Tras realizar las operaciones de codificación del modelo, procedimos a comprobar que los valores obtenidos dividido entre los valores originales nos devolvieran el multiplicando (nuestro número aleatorio D.4) y es aquí cunando nos damos cuenta de que no podemos utilizar los valores en coma flotante de los vértices ya que al multiplicar o dividir, los valores de los

mismo son corrompidos por los redondeos y el número de decimales. Por ejemplo, para un multiplicando de **0,7**, al dividir el valor obtenido entre el valor inicial obtenemos que el multiplicando es **0.712999999991845**, con lo que podemos concluir que esta no es una manera viable de encriptar los modelos. A continuación mostramos el modelo de la figura D.3 desencriptado tras modificar sus vértices en la figura D.5:

Figura D.5: Modelo desencriptado utilizando los valores de los vértices

Para la modificación de los valores de las caras

Tras el resultado nefasto de modificar los valores de los vértices, decidimos que podríamos alterar los valores de las caras formadas por los vértices, las cuales son enteras y no tendremos el problema de los decimales. En este caso, siendo la manera de generar los números aleatorios la mencionada en la sección D.4 y la manera de modificar los valores de las caras: *valor de*

*la cara * (número aleatorio obtenido * 10) + (número aleatorio obtenido * 100)* habiendo también realizado un estudio de los tiempos de codificación de los modelos, teniendo en cuanta que el número de caras en el modelo cogido de ejemplo es **248999** mientras que el número de vértices es de **126720**:

Para los modelos ASCII:

- Si modificamos todos los vértices que componen al modelo obtenemos un tiempo de: **14.124540567398071 segundos**
- Si modificamos solamente los vértices que ocupan posiciones múltiplos de cuatro del modelo (la cuarta parte de operaciones) obtenemos un tiempo de: **11.115345239639282 segundos**

Para los modelos Binarios:

- Si modificamos todos los vértices que componen al modelo obtenemos un tiempo de: **2.940922975540161 segundos**
- Si modificamos solamente los vértices que ocupan posiciones múltiplos de cuatro del modelo (la cuarta parte de operaciones) obtenemos un tiempo de: **1.9844660758972168 segundos**

Una vez conocidos estos datos, decidimos modificar únicamente los valores que ocupan posiciones múltiplos de cuatro ya que la encriptación del modelo es suficiente para conservar su seguridad (el modelo no se llega a mostrar en el navegador ya que no es capaz de dibujarlo) y el tiempo de codificación es menor. De este modo obtenemos tanto una mejor codificación en lo relacionado con la seguridad como de precisión de resultados tras la decodificación. Por lo tanto, concluiremos adjudicando a las **caras** la encriptación en lugar de a los **vértices**.

Obtendremos un modelo encriptado como el mostrado en la figura D.6:

Figura D.6: Modelo de encriptado

Ejecución de nuestra aplicación en arquimedes

Hay un script que ejecuta la aplicación. Desde Linux con el comando ssh (usuario y contraseña), en windows con putty y lo mismo. Una vez que se esté ejecutando, accederemos a la aplicación en la dirección url: https://arquimedes.ubu.es/visor3d/.

D.5. Compilación, instalación y ejecución del proyecto

D.6. Pruebas del sistema

Apéndice E

Documentación de usuario

E.1. Introducción

E.2. Requisitos de usuarios

E.3. Instalación

E.4. Manual del usuario

Subir Modelos

Bibliografía

[1] Ken Schwaber and Mike Beedle. *Agile software development with Scrum*, volume 1. Prentice Hall Upper Saddle River, 2002.

[2] Alberto Vivar. 3d viewe, 2016-2017.

[3] Wikipedia. Heroku – wikipedia, la enciclopedia libre.