Universidad del País Vasco
Euskal Herriko Unibertsitatea

Facultad de Informática

Grado de Ingeniería Informática

▪ Proyecto Fin de Grado ▪

Computación

Plataforma Comparativa de Sistemas de Resolución de Ecuaciones Diferenciales de N Cuerpos

Kevin García Santos

Julio 2017

Director:

Joseba Makazaga Odria

informatika fakultatea
facultad de informática

Agradecimientos

Me gustaría agradecerle principalmente a mi familia, a todos y cada uno de ellos, gracias a ellos estoy donde estoy y he llegado a ser lo que soy, sin ellos esto no hubiera sido posible, con una familia como la que me ha tocado a mí poco más se puede pedir, gracias por todo el apoyo y el cariño que siempre me habéis dado, sé que no lo digo mucho pero se os quiere muchísimo

En especial, a mi Aita que siempre me ha apoyado y confiado en mí y que sé que está muy orgulloso de mi, y aunque no se lo diga mucho, yo también lo estoy de él, por todo lo que ha hecho por mí y lo que hará, y que le quiero mucho, a mi hermana junto a la que he crecido y que al final siempre va ser mi hermana pequeña y siempre va a contar conmigo para todo, y a mi Ama también que aunque no siempre compaginemos tanto me ha ayudado muchísimo .

Tengo que agradecerle mucho también a mi director del proyecto, que me ha ayudado en todo momento cuando he tenido alguna duda o problema y siempre ha estado dispuesto a ayudar, muchísimas gracias de verdad, por si no nos volvemos a ver en el futuro, te deseo lo mejor a ti y a los tuyos, mucha suerte.

Por último, me gustaría también agradecer a mis amigos, a los que están ahí siempre que se les necesita, tener gente así alrededor no tiene precio, y a todos los compañeros con los que he compartido estos años de subidas y bajadas, en especial a unos pocos con los que he compartido muchas horas de mi tiempo tanto fuera como dentro, gracias, gracias a todos.

Resumen

Esta memoria contiene toda la información correspondiente a la realización del proyecto de fin de grado elaborado para la especialidad de computación del Grado en Ingeniería Informática de la Universidad del País Vasco impartido en la Facultad de Informática de Donostia-San Sebastián.

La cual tiene como tema del proyecto la realización de una plataforma de visualización de diferentes sistemas de ecuaciones diferenciales para un problema concreto, el problema de los n cuerpos, para la parte de visualización de este proyecto se utiliza HTML5, para crear la página web, que será la que contiene la aplicación, con elementos de JavaScript para el tratamiento de errores, actualización de elementos, y demás, pero también proporciona lo más importante, el entorno gráfico, que se enlaza a la página mediante el propio JavaScript.

Este entorno grafico se trata de WebGL, el cual está integrado en la mayoría de navegadores actuales para dotar a los navegadores web la capacidad de renderizar gráficos en 3D directamente desde el propio navegador, solamente con la propia API integrada, sin la necesidad de tener que instalar ningún tipo de *plug-in* externo para ejecutarse ni similares, del cual, utilizaremos la librería de Three.js, la cual es una herramienta muy completa que nos proporciona diferentes funcionalidades a la hora de desarrollar nuestra aplicación para este proyecto.

Índice

Resumen V
Índice VII
Lista de Figuras y Tablas X

1. Introducción 1
1.1. Objetivos y Alcance 2
1.2. Motivación Personal 3
1.3. Planificación 3
1.3.1. Tareas 3
1.3.2. Gestión del Tiempo 5
1.3.3. Riesgos 7
1.3.3.1. Almacenamiento del Proyecto y Documentación. 7
1.3.3.2. Cambios de Diseño. 8
1.3.3.3. Realización del Proyecto. 8
1.4. Herramientas 9
1.4.1. StopWatch 9
1.4.2. Atom 10
1.4.3. Notepad++ 11
1.4.4. Microsoft Word 2010 12
1.4.5. Chrome 13
1.4.6. Opera 14
1.4.7. Dropbox 17
1.4.8. Google Drive 18

2. Sistemas de Resolución de Ecuaciones Diferenciales 19
2.1. Métodos Numéricos 20
2.1.1. Problema de Valor Inicial 21
2.1.2. Métodos de Runge-Kutta 23
2.1.2.1. Tablero de Butcher 25
2.1.2.1.1. Explícitos 25
2.1.2.1.2. Implícitos 26
2.1.2.2. Método de Euler 35
2.1.2.3. Método de Runge-Kutta de cuarto orden (RK4) 36
2.1.2.4. Método de Gauss de tres etapas, orden seis 38

3. Problema de los n-cuerpos 41

4. Three.js 45

4.1. Características 46
4.2. Funcionamiento 50

5. Aplicación Web 61

5.1. Características 62
5.2. Interfaz 63
5.3. Funcionamiento 78
5.4 Errores 90
5.4.1. Web 90
5.4.2. Three.js 92

6. Conclusiones 97

6.1. Conclusiones 98
6.2. Posibles Mejoras 99

Glosario 101

Bibliografía 105

A Anexo. Enlaces Herramientas 108

Lista de Figuras y Tablas

FIGURAS

Figura 1.4.1.1. StopWatch 9

Figura 1.4.2.1. Interfaz de Atom 10

Figura 1.4.3.1. Interfaz de Notepad++ 11

Figura 1.4.4.1. Interfaz de Microsoft Word 2010 12

Figura 1.4.5.1. Interfaz de Google Chrome 13

Figura 1.4.6.1. Interfaz de Opera 14

Figura 1.4.6.2. Interfaz Modo Desarrollo Google Chrome y Opera 16

Figura 1.4.7.1. Interfaz Web de Dropbox 17

Figura 1.4.8..1 Interfaz Web de Drive 18

Figura 2.1.1.1. Formula de los problemas de valor inicial, ecuación 1 21

Figura 2.1.1.2. Formula de los problemas de valor inicial, ecuación 2 22

Figura 2.1.1.3. Formula de los problemas de valor inicial, ecuación 3 22

Figura 2.1.2.1. Formula de los Métodos de Runge-Kutta, parte 1 23

Figura 2.1.2.2. Formula de los Métodos de Runge-Kutta, parte 2 24

Figura 2.1.2.1.1. Tablero de Butcher Forma General 25

Figura 2.1.2.1.1.1. Tablero de Butcher RK4 Estándar 26

Figura 2.1.2.1.2.1. Tablero de Butcher SDIRK4 Implícito diagonal 27

Figura 2.1.2.1.2.2. Tablero de Butcher Lobatto3c Implícito completo 27

Figura 2.1.2.1.2.3. Ecuación que define al punto fijo 29

Figura 2.1.2.1.2.4. Ecuación de la iteración del punto fijo 29

Figura 2.1.2.1.2.5. Sistema de ecuaciones Runge-Kutta implícito 29

Figura 2.1.2.1.2.6. Formula del punto fijo 30

Figura 2.1.2.1.2.7. Método del punto fijo, inicialización 30

Figura 2.1.2.1.2.8. Ecuación ejemplo método del punto fijo 32

Figura 2.1.2.1.2.9. Mensaje de Error – Punto Fijo Diverge 34

Figura 2.1.2.2.1. Tablero de Butcher Euler Explicito 35

Figura 2.1.2.2.2. Tablero de Butcher Euler Implícito 35

Figura 2.1.2.2.3. Formula del Método de Euler 36

Figura 2.1.2.2.4. Formula del Método de Euler Implícita 36

Figura 2.1.2.3.1. Tablero de Butcher RK4 Explicito 37
Figura 2.1.2.3.2. Formula del método RK4 37
Figura 2.1.2.3.3. Formula de los pasos intermedios RK4 implícitos 38
Figura 2.1.2.4.1. Formula del método de Runge-Kutta, método de Gauss s=3 39
Figura 2.1.2.4.2. Formula de los pasos intermedios implícitos 39
Figura 3.1. Problema de los n-cuerpos, formula 1 42
Figura 3.2. Problema de los n-cuerpos, formula 2 43
Figura 4.1.1. Cubo Verde enThree.js 46
Figura 4.1.2. Líneas Azules enThree.js 47
Figura 4.1.3. Cubo iluminado por la parte frontal 48
Figura 4.1.4. Escena iluminada con luz focal 49
Figura 4.1.5. Esquema de la cámara 49
Figura 4.2.1. Página HTML en blanco 54
Figura 4.2.2. Visualización de la escena vacía 56
Figura 4.2.3. Visualización de esfera blanca 58
Figura 4.2.4. Visualización de esfera blanca wireframe 58
Figura 4.2.5. Rotación de esfera blanca 60
Figura 5.2.1. Página web principal completa 63
Figura 5.2.2. Interfaz web, tabla 1, parte superior 64
Figura 5.2.3. Interfaz web, incremento del título del elemento 65
Figura 5.2.4. Interfaz web, campos de posiciones 65
Figura 5.2.5. Interfaz web, campos de velocidades 66
Figura 5.2.6. Interfaz web, campo de masa 66
Figura 5.2.7. Interfaz web, campo de radio de la esfera 67
Figura 5.2.8. Interfaz web, botón añadir 67
Figura 5.2.9. Interfaz web, tabla 1, parte inferior 68
Figura 5.2.10. Interfaz web, métodos de resolución 69
Figura 5.2.11. Interfaz web, campo de tamaño de paso 69
Figura 5.2.12. Interfaz web, campo de numero de pasos 70
Figura 5.2.13. Interfaz web, tabla 2, información de elementos introducidos 70
Figura 5.2.14. Interfaz web, tabla 2, información de elementos introducidos 2 71
Figura 5.2.15. Interfaz web, botón iniciar 72
Figura 5.2.16. Interfaz entorno gráfico, entorno grafico completo 72
Figura 5.2.17. Interfaz entorno gráfico, menú, desplegado y plegado 74

Figura 5.2.18. Interfaz entorno gráfico, menú, directorio ubicación cámara 74
Figura 5.2.19. Interfaz entorno gráfico, menú, directorio LookAt cámara 75
Figura 5.2.20. Interfaz entorno gráfico, menú, directorio opciones de la cámara 75
Figura 5.2.21. Interfaz entorno gráfico, menú, directorio opciones de la gráfica de error 76
Figura 5.2.22. Interfaz entorno gráfico, grafica de error 77
Figura 5.3.1. Página principal con dos cuerpos introducidos 79
Figura 5.3.2. Opciones del Sistema 80
Figura 5.3.3. Mensaje de finalización del método. 82
Figura 5.3.4. Botón de iniciar el programa 82
Figura 5.3.5. Menú de la cámara. 84
Figura 5.3.6. Entorno gráfico, estado inicial 85
Figura 5.3.7. Entorno gráfico, estado objetos, numero de paso 206 85
Figura 5.3.8. Entorno gráfico, estado objetos, numero de paso 280 86
Figura 5.3.9. Entorno gráfico, estado objetos, numero de paso 403 86
Figura 5.3.10. Entorno gráfico, estado objetos, numero de paso 500, final 87
Figura 5.3.11. Entorno gráfico, cuerpos acerados mediante zoom 88
Figura 5.3.11. Entorno gráfico, cámara centrada en siguiente objeto 89
Figura 5.4.1.1. Mensaje de Error 01 90
Figura 5.4.1.2. Mensaje de Error 02 91
Figura 5.4.2.1. Mensaje de Error –Método del Punto Fijo 92
Figura 5.4.2.2. Grafica del Error, ejemplo de paso, imagen 1 94
Figura 5.4.2.3 Grafica del Error, ejemplo de paso, imagen 2 94
Figura 5.4.2.4 Grafica del Error, ejemplo de paso, imagen 3 95

TABLAS

Tabla 1.3.2.1. Tiempo Dedicación en Tareas de Gestión 5
Tabla 1.3.2.2. Tiempo Dedicación en Tareas de Formación y Estudio 6
Tabla 1.3.2.3. Tiempo Dedicación en Tareas de la Aplicación 6
Tabla 2.1.2.1.2.1 Resultado ejemplo método del punto fijo 33

Índice de Códigos

CODIGOS

Código 4.2.1. Página HTML con Three.js, librería en el proyecto 51
Código 4.2.2. Página HTML con Three.js, librería en URL.....51
Código 4.2.3. Etiquetas script.....52
Código 4.2.4. Definición de la escena, cámara y renderer 53
Código 4.2.5. Bucle de animación y renderización de la cámara con la escena.....55
Código4.2.6. Creación e introducción de una esfera en la escena.....56
Código4.2.7. Creación e introducción de una esfera en la escena.....59

1

Introducción

1.1. Objetivos y Alcance

El objetivo principal de este proyecto es desarrollar una aplicación para poder visualizar diferentes sistemas de resolución de ecuaciones diferenciales, los denominados como métodos numéricos de resolución de ecuaciones diferenciales, mediante la implementación de dichos métodos en un entorno grafico 3D, en el que se pueda ver el progreso que sigue el sistema de resolución seleccionado, en este caso el entorno grafico estará incluido en un navegador web. Este proyecto, más concretamente, tiene el objetivo de resolver las ecuaciones diferenciales que nos propone el problema de los n cuerpos para predecir los movimientos de los objetos que se rigen por las leyes de la gravitación universal de Newton, aunque haciendo algunas modificaciones en la aplicación podría resolver las ecuaciones diferenciales de otros problemas diferentes.

La aplicación final será una aplicación web ejecutable en cualquier navegador web compatible con WebGL, capaz de mostrar en el propio navegador web un entorno grafico en el cual se podrá observar el progreso que sigue el método de resolución seleccionado, en este caso se realizaran los métodos de Euler (explicito e implícito) y los métodos de Runge-Kutta de orden cuatro (explicito e implícito), sobre los diferentes cuerpos introducidos, además de una visualización del error cometido en el tiempo por el método que se está ejecutando en un intervalo de la ejecución a modo de muestra del error que se está cometiendo.

Con lo anteriormente comentado, conseguiríamos aunar varias materias que se han impartido a lo largo del grado en un mismo proyecto, por ejemplo, como Gráficos por Computador y Computación Científica.

1.2. Motivación Personal

Uno de los motivos por los que empecé en esto de la informática es debido a que desde pequeño me han gustado los ordenadores, la informática, los videojuegos, la tecnología y demás, y ya desde muy joven tenía en mente que mi objetivo era ser Ingeniero Informático.

Y uno de los temas o campos que me suscitan más atracción e interés es el relacionado con los gráficos 3D, entornos virtuales y demás, es un tema que siempre ha estado rondando por mi mente y en un futuro me gustaría tener la oportunidad de poder disfrutar de él más extensamente y realizar cosas interesantes.

Entonces, finalmente, se optó por realizar este proyecto ya que contiene tareas de desarrollo de objetos 3D y también debido a que en mi caso aparte de la informática, la física es uno de esos otros temas que me producen bastante interés y al realizar este proyecto el resultado final sería una mezcla de dos campos que me parecen muy interesantes.

1.3 Planificación

1.3.1. Tareas

A lo largo de todo el proyecto se ha tenido que realizar distintas tareas, tanto como de investigación, como de implementación, como de formación y demás, de los diferentes temas que engloban al desarrollo de todo el trabajo de fin de grado.

A continuación se listaran los diferentes tipos de tareas con sus respectivas tareas correspondientes que se han realizado para la finalización del proyecto

Gestión:

- Planificación
- Control de Horas
- Control de Riesgos
- Asistencia a tutorías

Formación y Estudio:

- Estudio de los sistemas de resolución de ecuaciones diferenciales
    - Métodos Numéricos
        - Problemas de Valor Inicial (PVI)
    - Tipos de Métodos
        - Métodos Explícitos
        - Métodos Implícitos
            - Métodos Iterativos
    - Métodos de Runge-Kutta
    - Tablero de Butcher
    - Errores
- Estudio de los cuerpos gravitatorios
    - Problema de los 2 cuerpos
    - Problema de los 3 cuerpos
    - Problema de los n cuerpos
- Estudio sobre Three.js
- Formación en Three.js
- Redacción de la Memoria

Aplicación:

- Diseño Web
- Implementación de la aplicación.
    - Programación de Código
    - Corrección de Errores
    - Realización de Pruebas
    - Depuración

1.3.2. Gestión del Tiempo

En este apartado se detallará una aproximación de las horas dedicadas a las diferentes tareas que se han tenido que llevar acabo para la elaboración del proyecto de fin de grado.

Tareas de Gestión	Horas
Planificación	5
Control de Horas	2.5
Control de Riesgos	2.5
Asistencia a tutorías	20

Tabla 1.3.2.1. Tiempo Dedicación en Tareas de Gestión

Tareas de Formación y Estudio	Horas
Estudio de los sistemas de resolución de ecuaciones diferenciales	40
Estudio de los cuerpos gravitatorios	40
Estudio sobre Three.js	10
Formación en Three.js	15
Realización de la Memoria	45

Tabla 1.3.2.2. Tiempo Dedicación en Tareas de Formación y Estudio

Tareas de la Aplicación	Horas
Diseño Web	25
Implementación de la Aplicación	150

Tabla 1.3.2.3. Tiempo Dedicación en Tareas de la Aplicación

Lo que suma aproximadamente un total de 355 horas invertidas para la realización del proyecto.

1.3.3. Riesgos

En el momento en que se empieza con un proyecto que hay que realizar para unas fechas determinadas con un tiempo determinado, siempre hay posibles riesgos que se pueden producir a la largo del mismo, por tanto, hay que realizar un estudio de los riesgos, que influencia pueden tener en el proyecto, que posibilidad hay de que se produzcan y que opciones tenemos para evitarlos o minimizarlos lo máximo posible.

1.3.3.1. Almacenamiento del Proyecto y Documentación

Cuando se trabaja con ordenadores u otros dispositivos que almacenan los datos en sí mismos, hay que tener un control sobre ellos, ya que, todo aparato electrónico está expuesto a una serie de riegos que pueden conllevar la perdida de todo lo realizado hasta el momento, lo que tendría un impacto muy alto en el proyecto.

Por tanto, como medida preventiva a lo largo del proyecto todos los días rigurosamente se ha ido haciendo una copia de seguridad de todos los archivos que engloban al proyecto de fin de grado al final del día en un almacenamiento en la nube, más concretamente en Dropbox y Drive, con lo que conseguimos tener nuestros datos a salvo de cualquier riesgo que se pueda producir en el ordenador en el que estamos trabajando, además de tenerle en dos sistemas de almacenamiento en la nube diferentes también nos protegemos por posibles fallos que pudiesen surgir en dichas plataformas.

1.3.3.2. Cambios de Diseño

En un proyecto de esta envergadura es posible que se produzcan cambios en el diseño de la aplicación en fases de implementación avanzada o intermedia, lo cual puede tener un gran impacto en el desarrollo del proyecto, al tener que cambiar dependencias en el código con gran envergadura.

Por tanto, para evitar este posible riesgo, en un principio se definió la aplicación que se iba a llevar a cabo y las funcionalidades que tenía. Posteriormente, se han ido haciendo reuniones periódicas con el tutor para ver que se conseguían los objetivos marcados o si por problemas encontrados en la realización de los objetivos marcados en las definiciones iniciales planteadas, reconducir el proyecto en la línea adecuada lo antes posible para minimizar el riesgo de tener que cambiar un volumen de implementación muy alto.

1.3.3.3. Realización del Proyecto

Desde un principio se organizó el proyecto de fin de grado con el fin de estar libre de otras responsabilidades que pudieran entorpecer la realización del mismo, ya que, existe el posible riesgo de tener diferentes actividades que consuman gran parte del tiempo y que por la realización de ambas actividades simultáneamente una pueda influir en el buen desempeño de la otra.

Por tanto, el primer cuatrimestre de este curso se hizo una dedicación para acabar los créditos pendientes del grado y poder elaborar el proyecto de fin de grado sin tener pendientes otras activades.

1.4. Herramientas

Para desarrollar con éxito la aplicación hace falta tener las herramientas adecuadas para poder avanzar de la mejor manera y completar las diferentes tareas. Posterior a elaborar el estudio de las tareas a desarrollar, se optó por utilizar las siguientes herramientas para realizar las correspondientes tareas:

1.4.1. StopWatch

Para llevar el control total de las horas invertidas en el proyecto, se contabilizaron las horas mediante este gadget de escritorio para Windows 7.

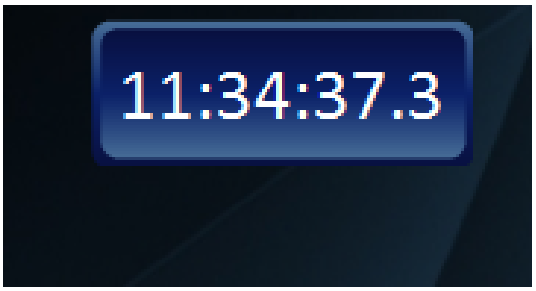

Figura 1.4.1.1. Gadget: StopWatch

1.4.2. Atom

Es un editor de textos, con una gran cantidad de lenguajes soportados, acepta JavaScript, HTML, CSS, PHP, Python, XML, C, C++, C#, Java, y demás. La mayoría del proyecto se ha realizado con esta aplicación, es un IDE bastante completo, con muchas opciones, atajos de teclado y funcionalidades. Además, es software libre, está desarrollado por GitHub.

En mi caso, se ha considerado buena opción, ya que, el proyecto engloba varios tipos diferentes de lenguajes programación, lenguajes de etiquetas como HTML, lenguajes orientados a objetos como JavaScript, hojas de estilos en cascada como CSS, y demás.

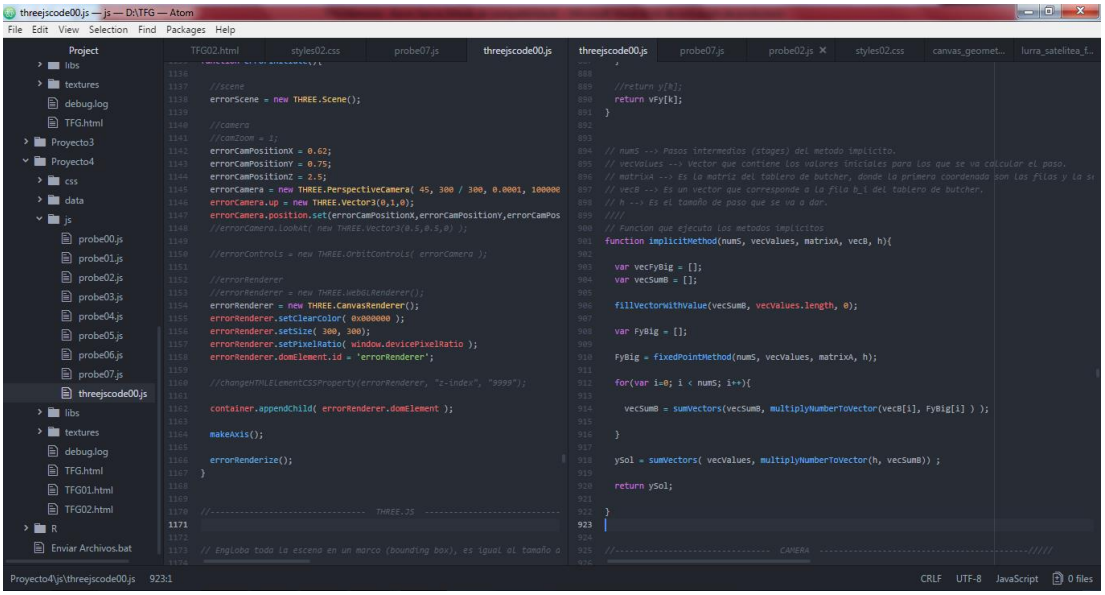

Figura 1.4.2.1. Interfaz de Atom

1.4.3. Notepad++

Notepad++ es otro editor de texto que también se ha utilizado en la realización del proyecto, ya que, es un editor muy ligero con una buena cantidad de lenguajes soportados. Es un IDE bastante inferior a Atom, debido a que no tiene tantos atajos ni funcionalidades, pero al ser tan ligero es muy manejable y puedes instalarlo en casi cualquier ordenador, ya que, exige muy pocos requisitos y espacio, por consiguiente, lo que permite poder trabajar casi en cualquier lugar. Además, también es software libre y si tienes poca memoria en el ordenador es una muy buena opción.

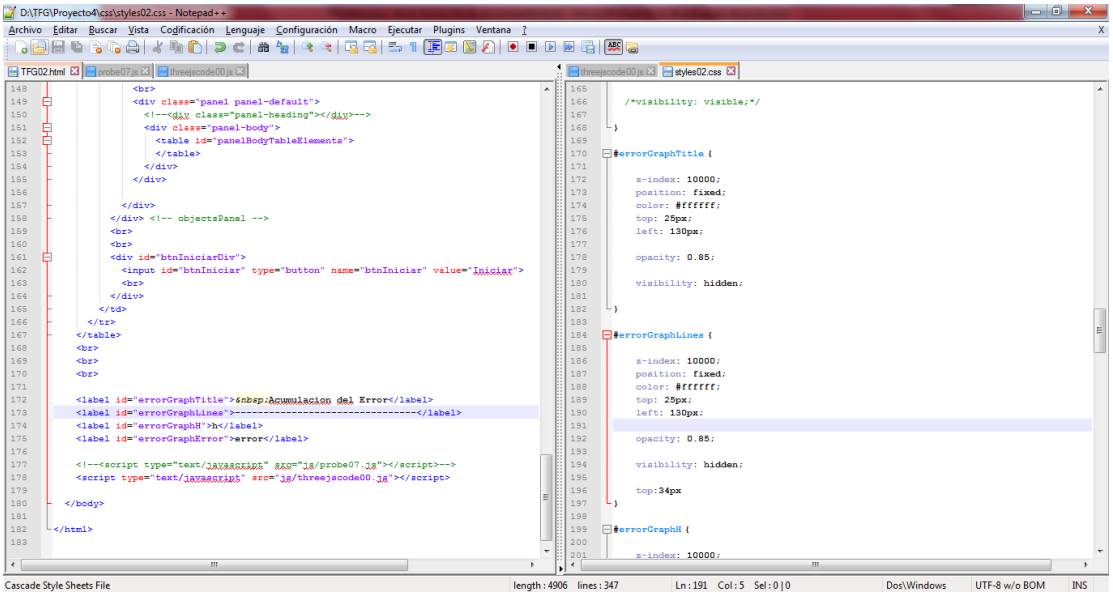

Figura 1.4.3.1. Interfaz de Notepad++

1.4.4. Microsoft Word 2010

Microsoft Word es un procesador de textos para crear documentos y es uno de los más populares del mundo, se encuentra incluido en el paquete ofimático de Microsoft Office, cuyo propietario del software es el mismo Microsoft.

Es una herramienta muy completa que permite realizar documentos con muchos elementos variados, y la cual tiene muchas funcionalidades, tanto de organización de texto, como de integración de imágenes, como de inserción de tablas, organigramas, temas y muchas más.

En mi caso se optó por esta herramienta, ya que, aparte de ser muy completa, se tenía un gran conocimiento previo del funcionamiento de la misma y resultaría más cómodo a la hora de redactar la memoria.

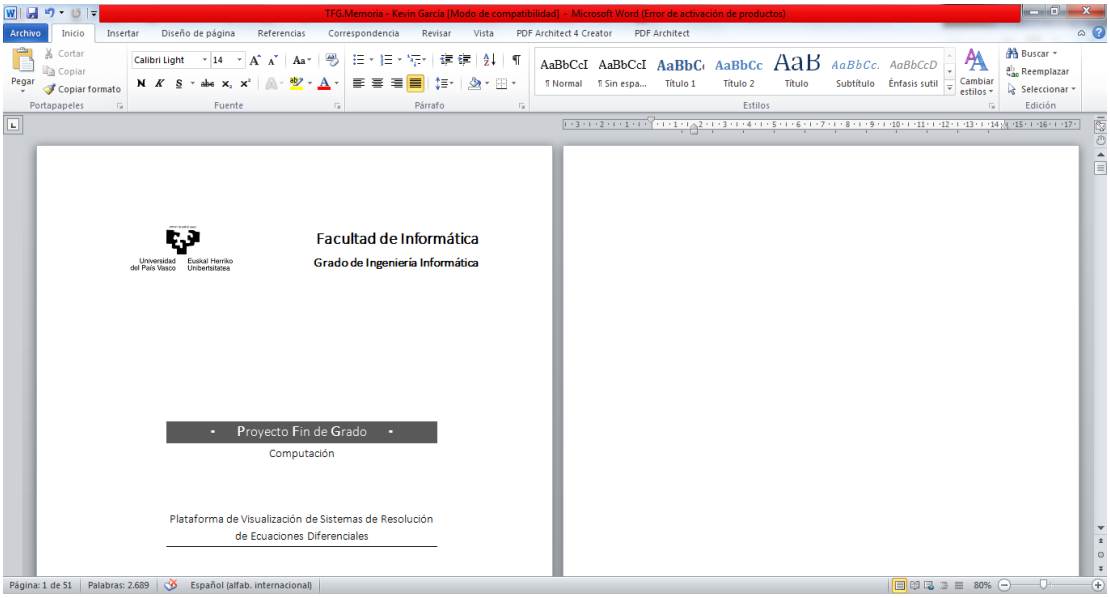

Figura 1.4.4.1. Interfaz de Microsoft Word 2010

1.4.5. Google Chrome

Google Chrome es un navegador web que está disponible gratuitamente para su instalación desde la propia página web de google, está desarrollado por la compañía Google, la cual da nombre a la misma aplicación, y hoy en día se mantiene entre los navegadores más usados a nivel mundial.

El uso de un navegador web era indispensable para efectuar con éxito este proyecto, ya que, la aplicación es una aplicación web y el entorno grafico se ejecutara en el navegador. Este fue uno de los navegadores utilizados en la realización del proyecto. Cuenta con un modo desarrollador que veremos más ampliamente al final de esta sección, que es muy útil a la hora de ver los resultados que se están obteniendo en la aplicación.

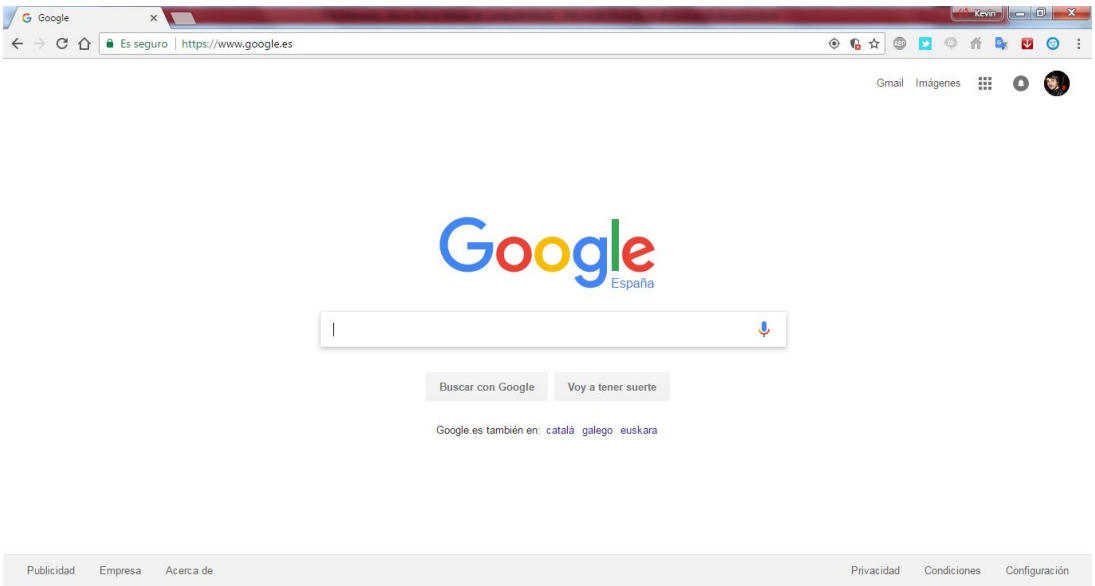

Figura 1.4.5.1. Interfaz de Google Chrome

1.4.6. Opera

Opera, es un navegador web desarrollado por Opera Software, al principio era un navegador de pago que después paso a ser freeware. Es compatible con Windows, Linux y Mac. Cuenta con un modo desarrollador integrado que es muy similar al de Google Chrome.

A modo de certificar el correcto funcionamiento de la aplicación también se ha utilizado este navegador en lo largo de la realización del proyecto.

Una de sus funcionalidades interesantes es que cuenta con una red privada virtual con la cual puedes navegar por internet como si lo estuvieras haciendo desde otra parte del mundo como Canadá, Alemania y demás.

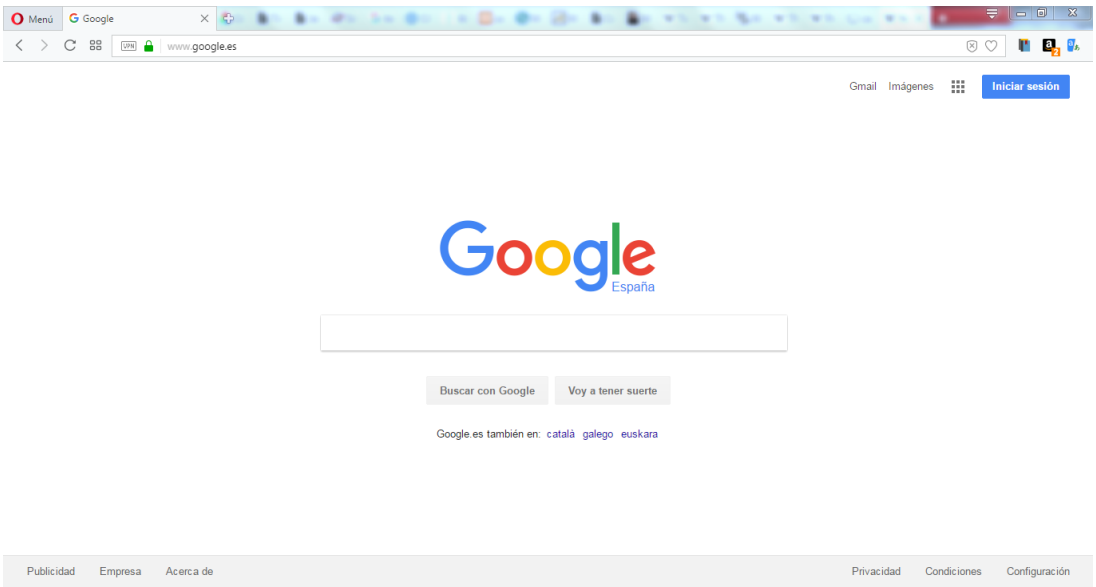

Figura 1.4.6.1. Interfaz de Opera

Tanto, Opera como Google Chrome tiene un modo desarrollador integrado en el navegador web que nos permite tener un control de lo que se está visualizando en el navegador.

Este modo desarrollador cuenta con:

- Un inspector de elementos, que nos muestra los diferentes elementos HTML que se están ejecutando en la página, además de los estilos que usan cada uno de los elementos y sus valores, y aparte estos estilos los podemos modificar a mano cambiando los valores, introduciendo otros nuevos o eliminándolos del elemento y poder visualizar el cambio que se produce en él, en directo.
- Una consola, donde se observan posibles errores, advertencias o información que se esté mostrando por la misma consola, que viene muy bien a la hora de estar probando el código que hemos realizado si tiene fallos o no es posible de ejecutar, esto se mostraría en la consola.
- Un visualizador de archivos fuente, esto nos permite observar los archivos web que se están ejecutando en el navegador, así como como su código original, la estructura de carpetas y los vínculos entre archivos.

A continuación, se muestra dos figuras con la interfaz de los dos navegadores utilizados en la realización de este proyecto.

Las interfaces son prácticamente iguales, por tanto, no se ha tenido ninguna dificultad en efectuar las pruebas en uno u otro navegador.

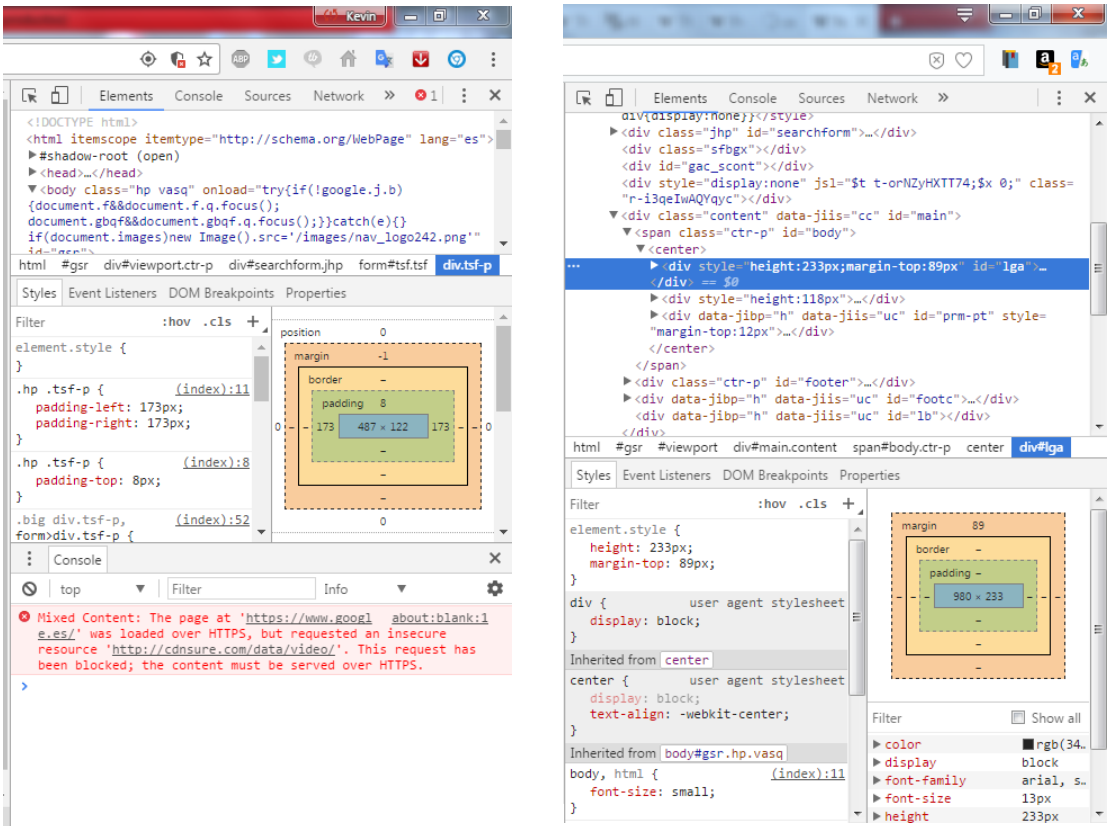

Figura 1.4.6.2. Interfaz Modo Desarrollo Google Chrome (Izquierda) y Opera (Derecha)

1.4.7. Dropbox

Dropbox es una de las plataformas para el almacenamiento de los archivos en la nube que se han utilizado en la realización del proyecto, que está desarrollada por Dropbox Inc., en 2007, y proporciona un espacio de almacenamiento en la nube gratuito de 2.5 GB para el usuario. Tiene paquetes adicionales que se pueden comprar para extender el almacenamiento y que también proporciona nuevos servicios, pero en este caso, se ha utilizado la versión gratuita.

Con esta aplicación puedes crear una carpeta en tu ordenador que estará enlazada con tu cuenta de almacenamiento en la nube y todos los archivos que se introduzcan en dicha carpeta, Dropbox , mediante una conexión de internet activa, los subirá a tu cuenta de forma automática, creando una copia de seguridad en la nube, en la que podrás acceder a todos tus archivos desde un navegador web. También puedes subir los archivos directamente desde el navegador para dejarlos almacenados en la nube.

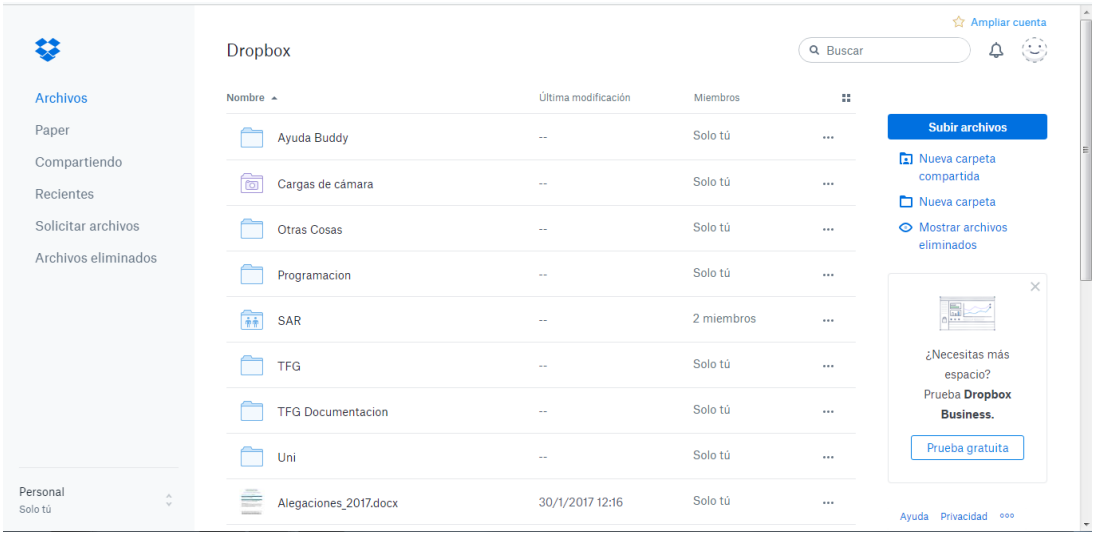

Figura 1.4.7.1. Interfaz Web de Dropbox

1.4.8. Google Drive

Google Drive, es la otra plataforma de almacenamiento de archivos en la nube que se ha utilizado para este proyecto, a modo de minimizar el riesgo de pérdida de datos.

El funcionamiento y estructura es similar a Dropbox, esta plataforma está desarrollada por Google, cuenta con una versión gratuita que proporciona 15 GB de almacenamiento en la nube gratuito a cada usuario, ampliables mediante planes de pago.

Tiene la opción de crear diferentes documentos al momento en la misma nube y también proporciona la posibilidad de hacer una copia de seguridad de los mensajes del WhatsApp del móvil o de las fotos del móvil, además de otras funcionalidades.

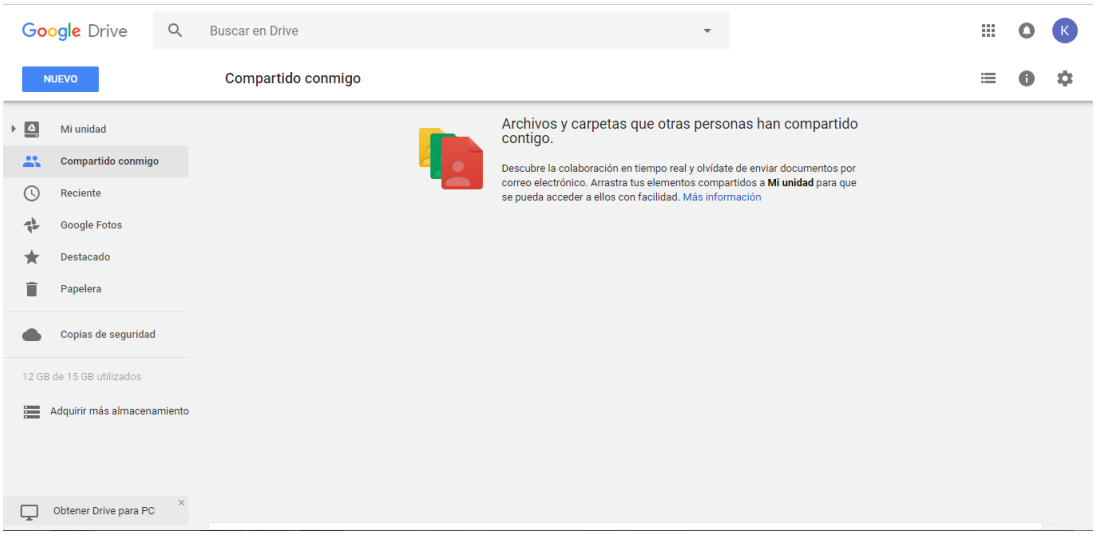

Figura 1.4.8.1. Interfaz Web de Drive

2

Sistemas de Resolución de Ecuaciones Diferenciales

2.1. Métodos Numéricos

Los ordenadores tienen gran capacidad de cómputo con lo que pueden hacer cálculos matemáticos complejos fácilmente, pero en realidad por debajo con lo que trabajan son con operaciones matemáticas simples, números binarios y con una representación de la información limitada, a pesar de ser tan potentes en el cálculo, tienen una cantidad finita de posiciones con las que pueden representar los números, es decir, para números que superan la longitud máxima del ordenador, se cometen pequeños errores de redondeo en el cálculo.

Por tanto, lo que hacen estos métodos numéricos es adecuar un problema, como el de resolución de ecuaciones diferenciales ordinarias, modificando la forma en la que se resuelve el problema a una forma en la que siguiendo un método obtendríamos una aproximación de la solución, en vez de resolverlo de forma analítica. Por ejemplo, en vez de resolver una ecuación diferencial ordinaria de manera que haya que identificar de qué tipo de ecuación es para luego resolverla de una forma o de otra según el tipo del que la hayamos identificado, en este caso seguiría un método que finalmente acabaría por devolver una aproximación de la solución que estábamos buscando, ya que, siempre tendremos cierto error debido al número finito de dígitos que soportan los ordenadores, con lo que conseguiríamos de esta manera que un ordenador si fuese capaz de resolver ese problema.

Estos métodos también se utilizan para resolver problemas que no tienen forma de resolverse de modo analítico, como las ecuaciones diferenciales ordinarias no lineales.

2.1.1. Problema de Valor Inicial

Un problema de valor inicial en el campo de las ecuaciones diferenciales es una ecuación diferencial ordinaria, en la que se nos proporciona en un punto del domino de la solución de la función, un valor especifico, que se denomina condición inicial, con el cual tendremos que ser capaces de hallar la solución a la ecuación. También se puede encontrar con el nombre de problema de Cauchy.

Estas ecuaciones diferenciales lo que nos indican es la variación que sufren las diferentes variables de nuestro problema respecto del tiempo, como se relacionan esas variables con el tiempo, es decir, a medida que avanza el tiempo que cambios tendrían esas variables, esto viene determinado por lo siguiente:

$$y' = \mathrm{f}(\mathrm{t}, \mathrm{y})$$
$$y \in R^d$$
$$con\ f: R^{d+1} \rightarrow R^d$$

Figura 2.1.1.1. Formula de los problemas de valor inicial, ecuación 1

Por lo que, *f* recibe *d+1* valores reales y nos devuelve *d* valores reales.

Como ocurre en nuestro caso, en *f(t, y)* es posible que no necesitemos introducir la variable *t* del tiempo como parámetro de la función, por lo tanto, la ecuación pasaría a tener esta forma:

$$y' = \mathrm{f}(\mathrm{y})$$
$$y \in R^d$$
$$con\ f: R^d \to R^d$$

Figura 2.1.1.2. Formula de los problemas de valor inicial, ecuación 2

En caso de que si fuese necesario, para esta forma, denominada la forma autónoma, introducir la *t* para evaluar las ecuaciones diferenciales del problema, lo que haríamos sería introducir *t* entre las variables añadiendo la ecuación *t' = 1,* por lo que nos quedaría una formula como la siguiente:

$$z' = g(\mathrm{z})$$
$$z \in R^{d+1}$$
$$con\ g: R^{d+1} \to R^{d+1}$$

Figura 2.1.1.3. Formula de los problemas de valor inicial, ecuación 3

Lo que quiere decir que, *z* tendría un elemento más que los que tiene *y* de la ecuación anterior, el cual, sería *t,* lo podríamos poner de la forma que *z = (y,t),* donde indicaríamos que *z* aparte de las variables de *y* tendría la variable *t. g* recibe en este caso *d+1* valores y nos devolvería *g+1* valores, pero el ultimo valor que devolvería ese *g* siempre sería uno, ya que, corresponde a la ecuación *t' = 1*.

Con esto podríamos decir que la ecuación de la figura 2.1.1.2. es igual a la ecuación de la figura 2.1.1.3. pero introduciendo *d+1* variables en vez de *d,* por lo que podríamos referirnos a las ecuaciones de forma general directamente como aparece en la ecuación 2.

El problema del valor inicial indica que para *t* = t_0 tenemos un estado *y*, es decir, para *t* = t_0 tenemos *y* = y_0

Una vez tengamos esto, las ecuaciones diferenciales se pueden resolver de diferentes formas, en este caso utilizaremos los métodos numéricos de la familia de métodos de Runge-Kutta, que lo que hacen es dividir el tiempo en tamaño de pasos, que denominaremos *h,* la cual podría ser fija o variable, y va dando pasos de ese tamaño para un tiempo final, quedaría algo como esto:

$$t_0 < t_1 < t_2 < \cdots < t_f$$

En esos tiempos se obtendrían las soluciones numéricas para cada uno de esas t_i, es decir, partiendo de y_0 se obtiene y_1, en un segundo paso, partiendo de y_1 se obtiene y_2, y así sucesivamente hasta conseguir obtener el paso final de y_f.

2.1.2. Métodos de Runge-Kutta

Los métodos de Runge-Kutta son un grupo de métodos iterativos pertenecientes a la familia de los problemas de valor inicial en el área de los métodos numéricos, que se utilizan para resolver las ecuaciones diferenciales ordinarias mediante la obtención de soluciones aproximadas al problema que se nos indica. Hay dos tipos de métodos de Runge-Kutta, los explícitos y los implícitos.

Los métodos de Runge-Kutta se rigen mediante la siguiente expresión:

$$y_k = y_{k-1} + h \sum_{i=1}^{s} b_i f(Y_i)$$

Figura 2.1.2.1. Formula de los Métodos de Runge-Kutta, parte 1.

Donde, $\boldsymbol{y_k}$ es la solución al método en la etapa $\boldsymbol{k}$ que sería la actual, para la que se está ejecutando el método, $\boldsymbol{y_{k-1}}$ también es la solución al método pero en este caso en la etapa $\boldsymbol{k-1}$ que corresponde a la etapa anterior, o el valor inicial si estuviéramos en el inicio del método, $\boldsymbol{f}$ sería la función de la ecuación diferencial ordinaria, $\boldsymbol{h}$ es el tamaño del paso con el que se ejecutaría el método, $\boldsymbol{s}$ el número de etapas que tiene el método de Runge-Kutta que se esté utilizando, $\boldsymbol{b_i}$ uno de los coeficientes proporcionados por el tablero de butcher según el esquema de resolución seleccionado, que se verá en el siguiente sub-apartado, Y_i se refiere a los pasos intermedios del método, que varían en número según el esquema utilizado y el número de etapas del mismo, definido mediante la siguiente ecuación:

$$Y_i \;=\; y_{k-1} \;+\; h \sum_{j=1}^{s} a_i \, f\left(Y_j\right)$$

Figura 2.1.2.2. Formula de los Métodos de Runge-Kutta, parte 2.

Donde, $\boldsymbol{Y_i}$ es la solución del paso intermedio, $\boldsymbol{y_{k-1}}$ de nuevo es la solución al método en la etapa anterior, $\boldsymbol{h}$ es el mismo tamaño de paso que para la ecuación anterior, $\boldsymbol{s}$ el mismo número de etapas, $\boldsymbol{a_{ij}}$ es otro coeficiente perteneciente al tablero de butcher, $\boldsymbol{f}$ la misma función que la anterior, e $\boldsymbol{Y_j}$ sería el paso intermedio pero el de índice $\boldsymbol{j}$.

Para está aplicación los métodos de Runge-Kutta que se han implementado han sido, los métodos de Euler explicito e implícito, ambos de orden uno y una única etapa, el método de Runge-Kutta de orden cuatro y cuatro etapas, RK4, y uno de los métodos implícitos de Runge-Kutta, el denominado método de Gauss de tres etapas y orden 6 (los métodos Runge-Kutta implícitos de Gauss tienen un orden que es igual al doble del número de etapas)

2.1.2.1. Tablero de Butcher

Los coeficientes anteriormente mencionados son proporcionados por este tablero que caracteriza al método de Runge-Kutta utilizado.

El tablero de Butcher es un esquema el cual nos proporciona los coeficientes a utilizar para un método de Runge-Kutta u otro, por lo tanto, antes de realizar cualquier método de Runge-Kutta tendríamos que tener la tabla correspondiente al método para poder ejecutarlo.

El tablero de butcher tiene la siguiente forma:

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array}$$

Figura 2.1.2.1.1. Tablero de Butcher Forma General.

Donde $\boldsymbol{s}$ es el número de etapas del método.

La matriz A, formada por los coeficientes $\boldsymbol{a_{ij}}$, según la forma de la misma, determina el tipo de método de Runge-Kutta.

2.1.2.1.1. Métodos Explícitos

Para el caso de los explícitos, la matriz A todos sus coeficientes $\boldsymbol{a_{ij}}$ por encima de la diagonal y en la misma diagonal serían cero, por lo tanto,

$\boldsymbol{Y_i}$ solo necesitaría los valores correspondientes a $\boldsymbol{j < i}$, que corresponden a los obtenidos en los anteriores pasos intermedios, por lo que solo depende de los pasos intermedios anteriores.

A continuación se muestra en la siguiente figura, el tablero de butcher correspondiente al método de Runge-Kutta estándar de orden cuatro, también denominado RK4:

0	0	0	0	0
$\frac{1}{2}$	$\frac{1}{2}$	0	0	0
$\frac{1}{2}$	0	$\frac{1}{2}$	0	0
1	0	0	1	0
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Figura 2.1.2.1.1.1. Tablero de Butcher RK4 Estándar.

2.1.2.1.2. Métodos Implícitos

Los métodos implícitos tienen un tablero de butcher diferente, hay dos tipos, los implícitos diagonales, que cuenta con la matriz A triangular inferior, es decir, todos los coeficientes $\boldsymbol{a_{ij}}$ por encima de la diagonal son cero, pero en este caso los coeficientes de la diagonal si son distintos a cero. La siguiente figura muestra un ejemplo de tablero de butcher implícito diagonal:

$\frac{1}{4}$	$\frac{1}{4}$	0	0	0	0
$\frac{3}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	0	0	0
$\frac{11}{20}$	$\frac{17}{50}$	$\frac{-1}{25}$	$\frac{1}{4}$	0	0
$\frac{1}{2}$	$\frac{371}{1360}$	$\frac{-137}{2720}$	$\frac{15}{544}$	$\frac{1}{4}$	0
1	$\frac{25}{24}$	$\frac{-49}{48}$	$\frac{125}{16}$	$\frac{-85}{12}$	$\frac{1}{4}$
	$\frac{25}{24}$	$\frac{-49}{48}$	$\frac{125}{16}$	$\frac{-85}{12}$	$\frac{1}{4}$

Figura 2.1.2.1.2.1. Tablero de Butcher SDIRK4 Implícito diagonal.

Los implícitos completos son aquellos que la matriz A cuenta con todos sus correspondientes coeficientes $\boldsymbol{a_{ij}}$, es decir, los coeficientes de la matriz son distintos de cero, no es una matriz triangular. En la siguiente figura se observa un tablero de butcher correspondiente a un implícito completo:

0	$\frac{1}{6}$	$\frac{-1}{3}$	$\frac{1}{6}$
$\frac{1}{2}$	$\frac{1}{6}$	$\frac{5}{12}$	$\frac{-1}{12}$
1	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{6}$
	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{6}$

Figura 2.1.2.1.2.2. Tablero de Butcher Lobatto3c Implícito completo.

Los implícitos nos indican, que a diferencia de los métodos explícitos, estos no solo dependen de los pasos intermedios anteriores, para cada paso se necesitaría tener evaluada con anterioridad la función de los

demás pasos intermedios (implícitos completos) o como mínimo tener evaluada la función en el mismo paso intermedio el cual se está resolviendo (implícitos diagonales), esto nos crea un problema, debido a que no tenemos el paso intermedio resuelto y necesitamos tener evaluada la función de este mismo paso intermedio, por lo tanto, tenemos que recurrir a métodos iterativos, como el método iterativo del punto fijo, para resolver este problema.

El método del punto fijo, es un método iterativo que permite resolver sistemas de ecuaciones, el cual se ha utilizado en este proyecto para resolver el problema que se produce en los métodos implícitos a la hora de calcular los pasos intermedios.

Los métodos implícitos en el momento de calcular los pasos intermedios tienen el problema anteriormente mencionado, cada uno de los pasos intermedios necesita el valor de la función evaluada en los demás pasos intermedios y en sí mismo, por tanto, lo que nos queda en cada etapa es un sistema de ecuaciones que tenemos que resolver de alguna forma, entonces, ahí es donde entra el método del punto fijo.

El método del punto fijo, desde un primer momento se nos tiene que proporcionar un estado inicial, después a partir de ese estado inicial, lo que hace el método en cada iteración es lo siguiente, hallamos la solución a ese estado inicial, para el paso siguiente repetimos el mismo proceso que hemos realizado pero ejecutando el método con el estado que hemos obtenido como solución de la iteración anterior, lo que nos proporciona una nueva aproximación a la solución y para las siguientes iteraciones procedemos de la misma manera, y así sucesivamente, por lo tanto, en cada iteración nos deberíamos estar aproximando cada vez más a la solución, y cuando ya no pueda mejorar más, se detiene, ya que, habremos encontrado la solución al punto fijo y habremos solucionado el problema de los métodos implícitos.

El procedimiento del punto fijo sería el siguiente, partiendo de una ecuación que define al punto fijo que tiene la siguiente forma:

$$v = g(v)$$

Figura 2.1.2.1.2.3. Ecuación que define al punto fijo

Partimos de una aproximación a esa *v* que cumple esa ecuación, por lo que se va iterando de la siguiente manera:

$$v(i+1) = g(v_i)$$

Figura 2.1.2.1.2.4. Ecuación de la iteración del punto fijo

Con el fin de que la nueva aproximación a la solución, es decir, *v(i+1)* se acerque a *v*, que corresponde al punto fijo que buscamos.

En los métodos de Runge-Kutta implícitos, lo que tenemos que resolver es un sistema de ecuaciones implícito, como el propio nombre del método indica, por lo tanto, tendríamos algo parecido a esto:

$$\begin{aligned} Y_1 &= y_{k-1} + g_1(Y_1, Y_2, \dots, Y_s) \\ Y_2 &= y_{k-1} + g_2(Y_1, Y_2, \dots, Y_s) \\ Y_3 &= y_{k-1} + g_3(Y_1, Y_2, \dots, Y_s) \\ &\dots \\ Y_s &= y_{k-1} + g_s(Y_1, Y_2, \dots, Y_s) \end{aligned}$$

Figura 2.1.2.1.2.5. Sistema de ecuaciones Runge-Kutta implícito

El punto fijo nos tendrá que devolver todas las componentes correspondientes a las *Y* del sistema de ecuaciones, las componentes de $Y_1, Y_2, Y_3, \dots, Y_s$, el punto fijo que buscamos se podría indicar de la siguiente forma $v = (Y_1, Y_2, Y_3, \dots, Y_s)$, donde cada uno de los componentes de *v*, sería de dimensión igual a *d,* por lo que el punto fijo, *v*, tendría *s*d* componentes, siendo *s* el número de ecuaciones del sistema a resolver y *d* la dimensión de cada una de esas ecuaciones y podemos decir que la ecuación implícita a resolver es la siguiente:

$$\mathrm{v} = G(v)$$
$$v \in R^{s*d}$$
$$con\ G: R^{s*d} \rightarrow R^{s*d}$$

Figura 2.1.2.1.2.6. Formula del punto fijo

Entonces, lo primero que tenemos que hacer para resolver el sistema de ecuaciones, es decir, hallar los componentes $Y_1, Y_2, Y_3, \dots, Y_s$, es inicializar todas esas componentes $Y_1, Y_2, Y_3, \dots, Y_s$ con y_k en t_k, por lo tanto al principio tendríamos un vector de la siguiente forma:

$$v_0 = (y_{k-1}, y_{k-1}, y_{k-1}, \dots, y_{k-1})$$

Figura 2.1.2.1.2.7. Método del punto fijo, inicialización

Y a partir de esos valores obtendremos la solución a la siguiente iteración, $\mathrm{v}_1 = G(v_0)$ y así sucesivamente hasta que decidamos parar porque hemos llegado a que $v = (Y_1, Y_2, Y_3, \dots, Y_s)$, es ese momento podremos hallar la solución al paso del método implícito que estábamos calculando y_k, que tiene la forma $y_k = y_{k-1} + h \sum_{i=1}^{s} b_i f(Y_i)$.

La condición de parada que se ha implementado en este proyecto para el método del punto fijo, funciona de la siguiente manera, se obtiene la diferencia en valor absoluto entre los componentes de las soluciones obtenidas en la actual iteración, con los componentes de las soluciones obtenidas en la iteración anterior (lo que implica un mínimo de iteraciones antes para poder empezar a hacer la diferencia), tendremos un mínimo que guardará para cada componente el mínimo de ese componente que hemos conseguido en el método hasta la fecha, menos si el valor fuese cero, en ese caso el valor del mínimo no se actualiza.

Los resultados obtenidos en esa diferencia se comparan siguiendo las siguientes pautas:

- En cada iteración, si alguno de los componentes para los que se está ejecutando el método del punto fijo, mejora, seguimos iterando, es decir, si uno solo de los elementos que hemos obtenido en la diferencia es menor que el almacenado en el mínimo, seguimos, y se actualiza el mínimo para ese componente, menos en el caso de que sea cero, que el mínimo queda con el mismo valor.

El método pararía para los siguientes casos:

- Cuando todos los elementos de la diferencia sean cero, en ese caso habremos llegado a la solución del método del punto fijo.

- Cuando no se pueda mejorar más la solución obtenida, es decir, si al hacer la diferencia de los valores actuales con los de la anterior iteración, todos los valores son mayor que su valor correspondiente en el mínimo o iguales a cero, dejaríamos de buscar, no encontraremos solución mejor, debido a que el fin del

método en cada iteración es ir obteniendo valores más próximos a la solución, en el momento que no lo haga habremos llegado a la solución más próxima al punto fijo.

El punto flaco de este método podría ser la lentitud del mismo, el método suele necesitar de bastantes iteraciones para poder obtener la aproximación de la solución final, existen otros métodos para hallar la solución a este problema, como el método de Newton, el cual tiene una mejor convergencia, a la hora de hallar la solución es bastante más rápido, llega a la solución antes que el método del punto fijo, pero también exige unos cálculos más complejos, como el cálculo del Jacobiano, para hallar la aproximación de la solución.

A continuación veremos un ejemplo de cómo procedería el método del punto fijo, y que resultados se obtienen.

Para la siguiente ecuación:

$$\boldsymbol{x} = g(x) = \frac{3}{x-2}$$

Figura 2.1.2.1.2.8. Ecuación ejemplo método del punto fijo.

Y un estado inicial concreto, en este caso $\boldsymbol{x_0 = 4}$, el progreso que tendrá el método corresponde al que se muestra en la siguiente tabla:

	$x = g(x) = \frac{3}{x-2}$	Δx
x_0	4	0
x_1	1.5	2.5
x_2	-6	7.5
x_3	-0.375	5.625
x_4	-1.2632	0.8882
x_5	-0.9193	0.3438
x_6	-1.0276	0.1083
x_7	-0.9908	0.0367
x_8	-1.0030	0.0012
x_9	-0.9990	0.004
x_{10}	-1.0003	0.0013
x_{11}	-0.9999	0.0004
x_{12}	-1.0000	0.0001
x_{13}	-1	0

Tabla 2.1.2.1.2.1. Resultados ejemplo método del punto fijo.

Estos son los resultados que nos devolvería el método del punto fijo para una precisión de cuatro decimales, aunque como se puede observar a partir de la sexta o séptima iteración podríamos hacernos una idea de que el punto fijo que encontraríamos al final de la ejecución

del método sería el menos uno. A medida que el método avanza, sobre todo a un nivel de iteraciones alto o cercano a la solución, se aprecia que el incremento que se obtiene va decreciendo, debido a que cada iteración estaríamos más cerca de la solución, por tanto, los resultados que obtendremos serán muy parecidos entre sí, esto también nos indica que el método para ese estado inicial que se ha ejecutado, converge, es decir, al final de la ejecución habremos hallado la aproximación a la solución, lo que no sabemos es cuando encontrará dicha solución.

Se ha hecho también un control aparte para cuando el método del punto fijo diverge, no converge, en vez de que siga intentando hallar la solución obteniendo cada vez valores más alejados de la solución, cuando detecta ese tipo de comportamiento muestra un mensaje de error, y directamente para el método y acaba el programa, debido a que no tiene solución, y lo único que va hacer es hallar soluciones cada vez peores.

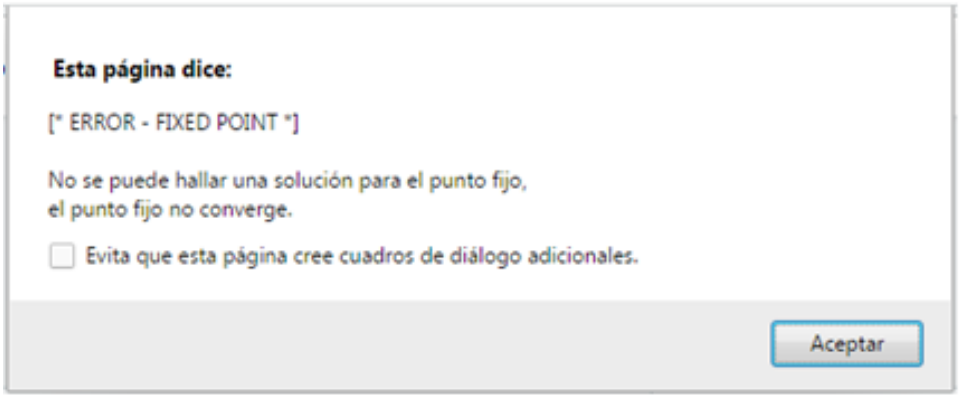

Figura 2.1.2.1.2.9. Mensaje de Error – Punto Fijo Diverge

2.1.2.2. Método de Euler

El método de Euler es el más sencillo de los métodos de Runge-Kutta, es un método de orden uno, lo que implica que el tamaño del paso determina el error, es decir, el error cometido es proporcional al tamaño del paso que hayamos elegido.

Los tableros de butcher correspondientes a su forma explícita e implícita, son los siguientes:

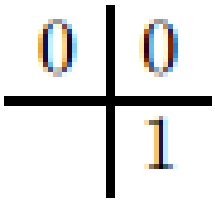

Figura 2.1.2.2.1. Tablero de Butcher Euler Explicito

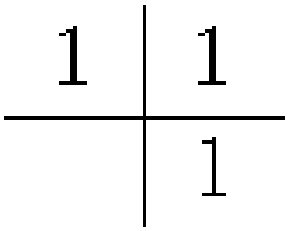

Figura 2.1.2.2.2. Tablero de Butcher Euler Implícito

En este caso, la formula general quedaría de la siguiente forma, para el caso de Euler explícito:

$$y_k = y_{k-1} + h\, f(Y_1)$$
$$Y_1 = y_k$$

$$y_k = y_{k-1} + h\, f(y_k)$$

Figura 2.1.2.2.3. Fórmula del método de Euler

Mientras tanto, para el método de Euler implcito quedaría de esta manera:

$$y_k = y_{k-1} + h\, f(Y_1)$$
$$Y_1 = y_k + h\, f(Y_1)$$

Figura 2.1.2.2.4. Fórmula del método de Euler implicita

2.1.2.3. Método de Runge-Kutta de cuarto orden (RK4)

El método de Runge-Kutta de orden cuatro, también conocido como RK4, es el método más conocido asociado a los métodos de Runge-Kutta, muchas veces cuando se hace referencia al método de Runge-Kutta se habla de este método.

Es un método de cuatro etapas y orden cuatro, lo que implica que el error cometido es igual al tamaño de paso elevado a la cuatro, por tanto, para tamaños de paso pequeños se obtiene un error menor que el que comete el método de Euler.

El tablero de butcher correspondiente a su forma explícita es el siguiente:

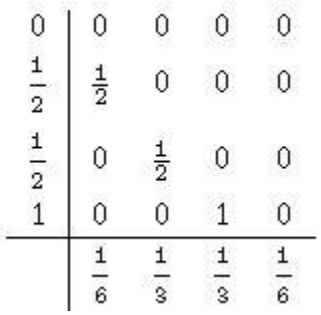

Figura 2.1.2.3.1. Tablero de Butcher RK4 Explicito

Por tanto la formula general de los métodos de Runge-Kutta se podría reescribir de esta forma:

$$y_k = y_{k-1} + h\left(\frac{1}{6} f(Y_1) + \frac{1}{3} f(Y_2) + \frac{1}{3} f(Y_3) + \frac{1}{6} f(Y_4)\right)$$

$$y_k = y_{k-1} + h\,\frac{1}{6}\left(f(Y_1) + 2f(Y_2) + 2f(Y_3) + f(Y_4)\right)$$

Figura 2.1.2.3.2. Fórmula del método RK4

Y para el caso de los explícitos, los pasos intermedios quedarían de la siguiente forma:

$$
\begin{aligned}
Y_1 &= y_{k-1} \\
Y_2 &= y_{k-1} + h\, a_{21}\, f(Y_1) \\
Y_3 &= y_{k-1} + h\, a_{31}\, f(Y_1) + h\, a_{32}\, f(Y_2) \\
Y_4 &= y_{k-1} + h\, a_{41}\, f(Y_1) + h\, a_{42}\, f(Y_2) + h\, a_{43}\, f(Y_3)
\end{aligned}
$$

$$
\begin{aligned}
Y_1 &= y_{k-1} \\
Y_2 &= y_{k-1} + h\, \frac{1}{2}\, f(Y_1) \\
Y_3 &= y_{k-1} + h\, \frac{1}{2}\, f(Y_2) \\
Y_4 &= y_{k-1} + h\, f(Y_3)
\end{aligned}
$$

Figura 2.1.2.3.3. Fórmula de los pasos intermedios de RK4 Explicito

2.1.2.4. Método de Gauss de tres etapas, orden seis

En este caso tenemos un método de Runge-Kutta implícito de tres etapas, y orden igual al doble de etapas, es decir, este método sería de orden seis, es conocido con el nombre de método de Gauss.

Por tanto, este método para tamaños de paso h pequeños el error cometido sería todavía más pequeño que para los otros métodos mencionados anteriormente, y sería el que mejor resultados nos mostraría en la aplicación, siempre que el tamaño de paso no sea muy alto.

Este método tiene el siguiente tablero de butcher asociado:

$$\begin{array}{c|ccc} \frac{1}{2}-\frac{\sqrt{15}}{10} & \frac{5}{36} & \frac{2}{9}-\frac{\sqrt{15}}{15} & \frac{5}{36}-\frac{\sqrt{15}}{30} \\ \frac{1}{2} & \frac{5}{36}+\frac{\sqrt{15}}{24} & \frac{2}{9} & \frac{5}{36}-\frac{\sqrt{15}}{24} \\ \frac{1}{2}+\frac{\sqrt{15}}{10} & \frac{5}{36}+\frac{\sqrt{15}}{30} & \frac{2}{9}+\frac{\sqrt{15}}{15} & \frac{5}{36} \\ \hline & \frac{5}{18} & \frac{4}{9} & \frac{5}{18} \end{array}$$

Figura 2.1.2.4.1. Fórmula del método de Runge-Kutta, método de Gauss s=3

Por lo que, para el caso de los implícitos la forma general de la fórmula de los métodos de Runge-Kutta se podría reescribir de esta manera:

$$\begin{aligned} Y_1 &= y_{k-1} + h\,a_{11}\,f(Y_1) + h\,a_{12}\,f(Y_2) + h\,a_{13}\,f(Y_3) \\ Y_2 &= y_{k-1} + h\,a_{21}\,f(Y_1) + h\,a_{22}\,f(Y_2) + h\,a_{23}\,f(Y_3) \\ Y_3 &= y_{k-1} + h\,a_{31}\,f(Y_1) + h\,a_{32}\,f(Y_2) + h\,a_{33}\,f(Y_3) \end{aligned}$$

Figura 2.1.2.4.2. Fórmula de los pasos intermedios Implícitos

3

Problema de los n-cuerpos

Para este proyecto, el problema que se ha integrado en la aplicación es el problema de los n-cuerpos, el cual, se define como el problema que hay que resolver para hallar el movimiento que se produce en los diferentes cuerpos debido a las leyes de gravitación de Newton,

Estos cuerpos por el simple hecho de existir y tener masa ejercen una fuerza gravitacional sobre los demás cuerpos al igual que los demás cuerpos ejercen una fuerza gravitacional sobre este.

El problema plantea que mediante el cálculo de las siguientes ecuaciones, podremos resolver el problema:

$$a_i = v'_i = \sum_{j=1, j\neq i}^{N} \frac{G\, m_j\, (q_i - q_j)}{||(q_i - q_j)||}$$

Figura 3.1. Problema de los n-cuerpos, formula 1

Donde, el índice *i* indica el objeto para el que se está calculando la aceleración, y el índice *j* el objeto que está influyendo gravitacionalmente sobre *i,* N es el numero de objetos que hay en el espacio, G es la constante de gravitación universal que es igual a $6.647 * 10^{-11}$, m_j es la masa del cuerpo que inflige la fuerza de gravitación sobre *i*, $(q_i - q_j)$ es la diferencia de la posición del cuerpo de índice *i* con el cuerpo de índice *j,* que en nuestro caso tendremos un resultado de la aceleración por cada coordenada en el espacio, que corresponden a X, Y, Z, y por último, $||(q_i - q_j)||$, es la norma del vector posición que hemos mencionado.

Gracias a esta fórmula podríamos hallar las ecuaciones del movimiento que corresponde a cada cuerpo, obteniendo así una ecuación por cada cuerpo en el espacio, lo que resulta en un sistema de ecuaciones.

Estas ecuaciones diferenciales también se pueden encontrar de la siguiente manera:

$$a_i = m_i P' = \sum_{j=1, j\neq i}^{N} \frac{G\, m_i m_j\, (q_i - q_j)}{||(q_i - q_j)||}$$

Figura 3.2. Problema de los n-cuerpos, formula 2

Donde $m_i P'$ se define como el momento angular del cuerpo de índice *i.* lo que nos daría que $m_i P' = v'$.

Entonces, lo que nuestra aplicación resuelve es este problema, realiza el cálculo de las ecuaciones diferenciales mediante los métodos numéricos anteriormente explicados, ya que, la función que utilizan en la ejecución del método es la correspondiente a este problema de los n-cuerpos.

En nuestro caso, para representar después los cuerpos en el entorno gráfico y poder observar la trayectoria que realizan influenciados por los diferentes cuerpos que haya a su alrededor, la función también calcula la derivada de la posición, con lo que devuelve los datos correspondientes a las nuevas aceleraciones y velocidades de todos los cuerpos, que son respectivamente las derivadas de las velocidades y las posiciones, en todas las coordenadas del espacio, es decir, en X, Y, Z, por lo que tendremos una ecuación de aceleración y velocidad por cada eje de coordenadas del cuerpo.

Estos datos obtenidos mediante estas ecuaciones diferenciales, los utilizamos con los métodos de resolución numéricos que se han implementado, lo que al ejecutar el método, nos proporciona el estado siguiente en el que se encontrará cada cuerpo gracias a las soluciones proporcionadas por este sistema de ecuaciones.

4

Three.js

4.1. Características

Es una librería JavaScript que utiliza WebGL para crear y mostrar animaciones 3D graficas en un ordenador utilizando un navegador web, fue desarrollada por Ricardo Cabello, más conocido como Mr.doob, y el código está alojado en GItHub.

Cuenta con diferentes tipos de objetos, como pueden ser huesos, líneas, partículas, mallas, y demás.

En el caso de las mallas pueden estar formadas por distintas geometrías que incluye Three.js, como las esferas, cubos, planos y demás,

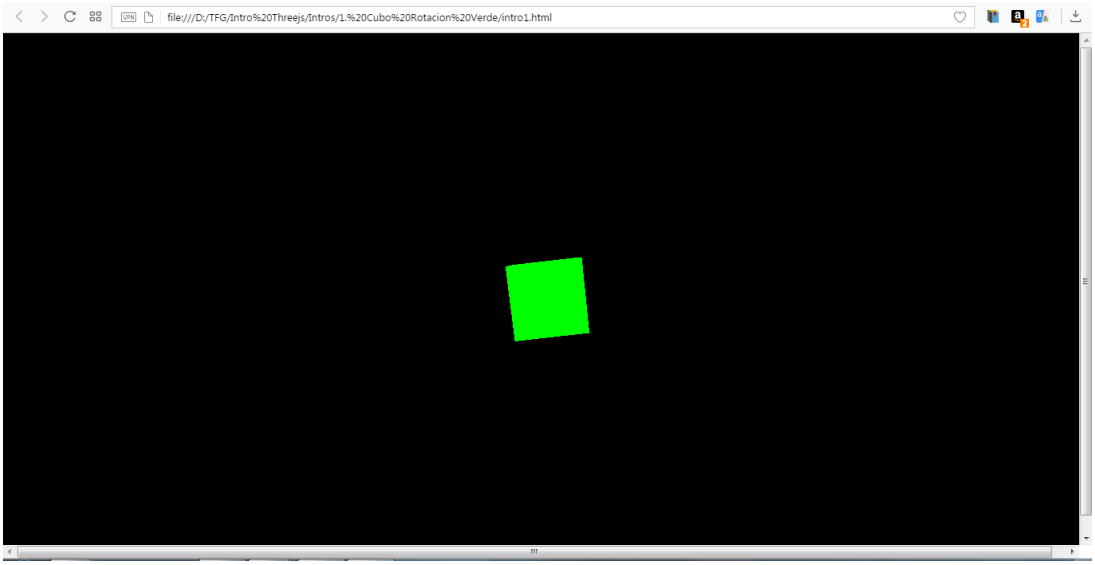

Figura 4.1.1. Cubo Verde en Three.js

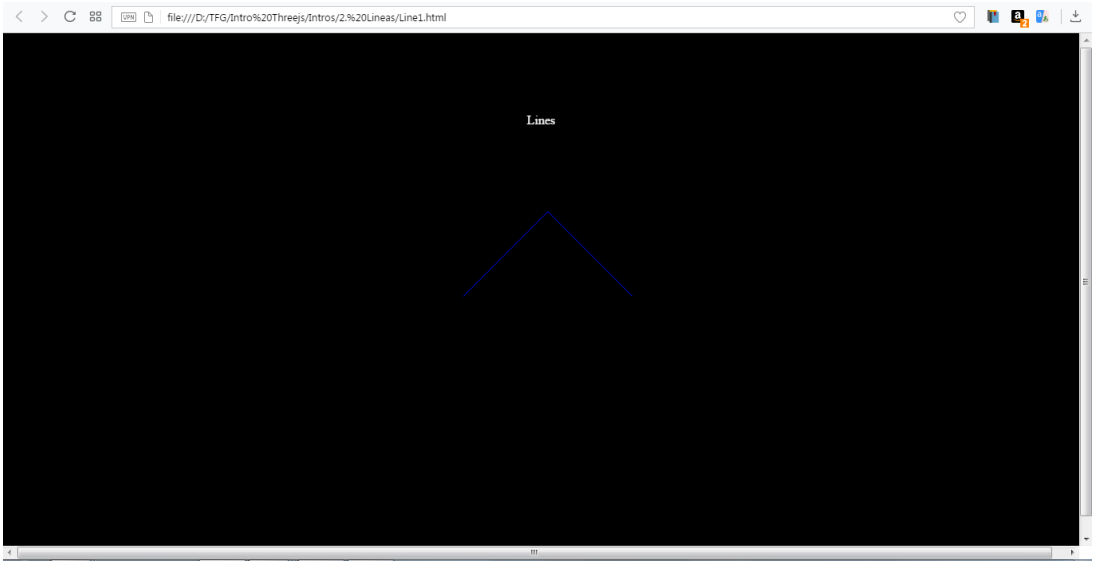

Figura 4.1.2. Líneas Azules en Three.js

Contiene un elemento *Scene,* que hace referencia a la escena donde se visualizaran todos los objetos que hayamos introducido, es decir, si creamos objetos pero no los introducimos en la escena nunca se llegaran a visualizar.

Al igual que ocurre con la escena, sino introducimos el elemento *Renderer* de Three.js con el que poder renderizar la escena, no podremos ver los objetos que tengamos en la escena. Three.js soporta tres tipos de renderizadores que son:

- CanvasRenderer
- SVGRenderer
- WebGLRenderer

Three.js también dispone de varios tipos diferentes de iluminación, como la luz ambiente, la cual ilumina toda las caras de todos los objetos que haya en la escena, la luz direccional, que incidirá en la escena según la dirección que le marquemos, la luz de foco, que ilumina simulando el haz de luz de un foco, y al igual que ilumina también es capaz de generar sombras.

Figura 4.1.3. Cubo iluminado por la parte frontal

Como se puede observar en la imagen anterior también cuenta con la opción de cargar texturas.

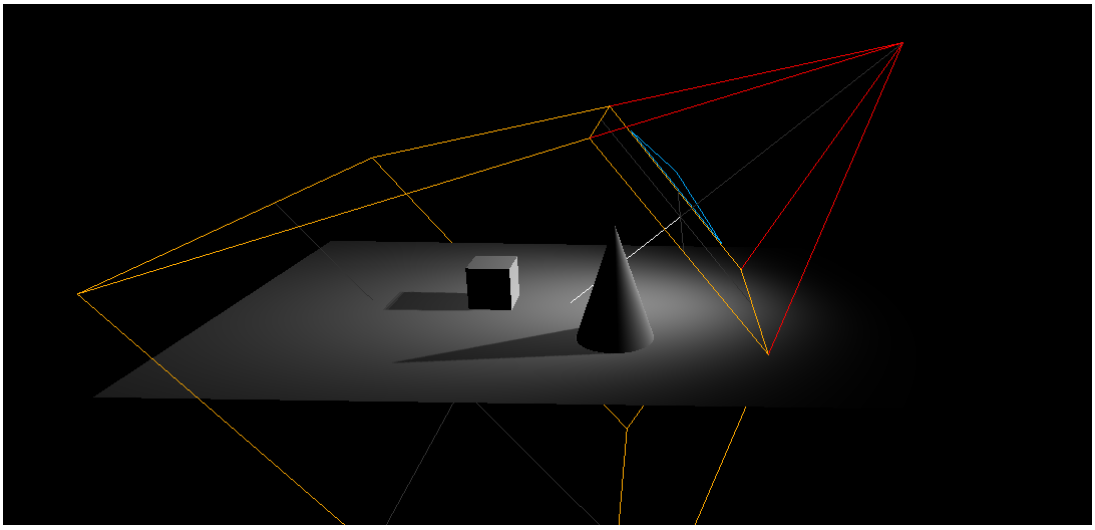

Figura 4.1.4. Escena iluminada con luz focal

Otro de los elementos importantes de Three.js es la cámara, hay varios tipos, como la cámara perspectiva, *PerspectiveCamera*, está diseñada para simular la forma humana de visión, es la más utilizada comúnmente, y la cámara ortográfica, *OrthographicCamera*, la que produce que el tamaño de un objeto no varié independientemente de a la distancia que esté de la cámara, y otros tipos como la cámara cubica, *CubeCamera,* y demás.

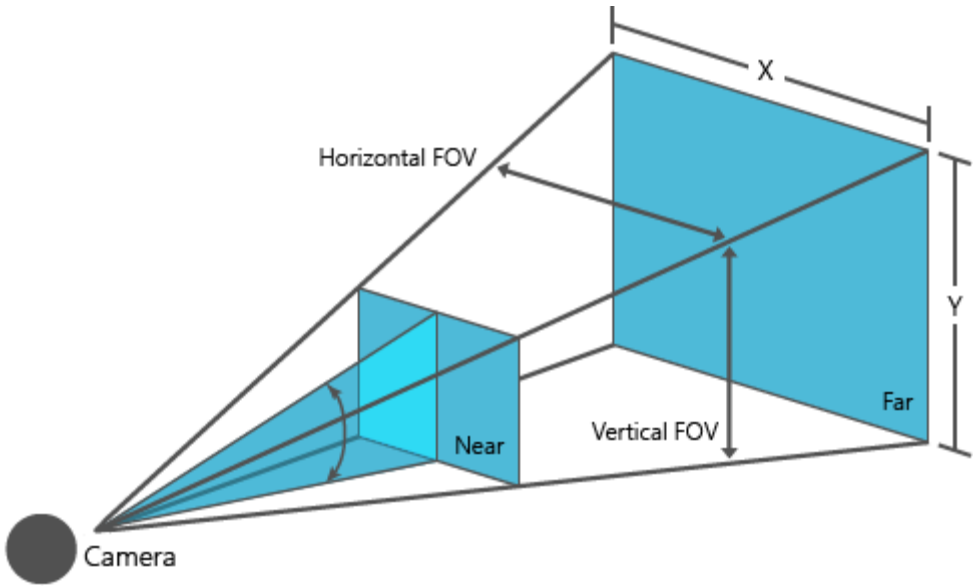

Figura 4.1.5. Esquema de la cámara

Cuenta con muchas más características, como animaciones, efectos, shaders, materiales y demás, pero los tres pilares básicos para poder vislumbrar algo en el navegador web que estemos utilizando son, la cámara, la escena y el renderer, después veremos cómo utilizarlos para poder tener una primera visión de nuestra escena.

4.2. Funcionamiento

En este apartado aprenderemos el funcionamiento de Three.js y como utilizar la herramienta.

Primero lo que tenemos que tener claro es, que la aplicación desarrollada será una aplicación web, es decir, se ejecutara en un navegador web, entonces, lo primero que tenemos que hacer es definir un documento HTML, ya que es con lo que trabaja Three.js, para crear la página en la que vamos a tener nuestra escena gráfica.

Lo siguiente será enlazar la página web con la librería de Three.js, ya sea, descargando el archivo *three.js* para introducirlo en el proyecto y enlazarlo, como se muestra en la siguiente figura (Figura 4.1.6.), o enlazando directamente con la ubicación de esa librería en internet, es decir, mediante la URL del código de Three.js donde esté alojado (Figura 4.1.7.), ambas formas son igualmente validas, lo único, esta última opción requiere de una conexión a internet para poder ejecutar la librería.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset=utf-8>
        <title>My first three.js app</title>
        <style>
            body { margin: 0; }
            canvas { width: 100%; height: 100% }
        </style>
    </head>
    <body>
        <script src="js/three.js"></script>
        <script>
            // Our Javascript will go here.
        </script>
    </body>
</html>
```

Código 4.2.1. Página HTML con Three.js, librería en el proyecto

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset=utf-8>
        <title>My first three.js app</title>
        <style>
            body { margin: 0; }
            canvas { width: 100%; height: 100% }
        </style>
    </head>
    <body>
        <script src="http://threejs.org/build/three.min.js"></script>
        <script>
            // Our Javascript will go here.
        </script>
    </body>
</html>
```

Código 4.2.2. Página HTML con Three.js, librería en URL

Una vez que tenemos este paso terminado ya podemos empezar a escribir nuestro código JavaScript de Three.js.

El código JavaScript debe ir entre las etiquetas denominadas script, lo que nuestro código tendríamos que tenerlo escrito entre estas etiquetas de la siguiente forma:

```
<script>
     // Aquí iría nuestro código
</script>
```

Código 4.2.3. Etiquetas script

Entonces para hacer nuestra primera escena, esta vez haremos una esfera y le daremos algo de movimiento con una animación de rotación, simulando la rotación de un planeta, necesitaremos como mínimo los tres elementos básicos para poder visualizar una escena en nuestro navegador web:

- Scene
- Camera
- Renderer

El código para definir dichos elementos sería el siguiente:

```
// Crea la escena
var scene = new THREE.Scene();

// Crea la cámara
var camera = new THREE.PerspectiveCamera( 40, window.innerWidth /
window.innerHeight, 0.1, 1000 );
camera.position.set(0,0,10);

// Crea el renderer
var renderer = new THREE.WebGLRenderer();

// Añadir el tamaño del área que va a controlar el
renderer, en este caso el ancho y alto de la ventana del
navegador.
renderer.setSize( window.innerWidth, window.innerHeight
);

//Se añade el elemento al documento HTML
document.body.appendChild( renderer.domElement );
```

Código 4.2.4. Definición de la escena, cámara y renderer

Vamos explicar un poco que contiene cada elemento, se ha definido una *PerspectiveCamera*, la cual sus argumentos son los siguientes, el primer argumento indica el campo de visión de la cámara, en nuestro caso sería 40, el segundo argumento es el *aspect*, que es el aspecto que tendrán los objetos vistos con la cámara, en nuestro caso el aspecto será el mismo que el tamaño del renderizado, para ver los objetos sin deformaciones, con un tamaño distinto al del renderizado, el aspecto de los objetos aparecería aplastados, deformados, etc. Como tercer argumento tenemos la componente *near*, esto nos señala como de cerca podemos ver los objetos, es decir, en nuestro caso, en caso de que nos acerquemos a un objeto demasiado y superemos la distancia mínima de 0.1, entonces el objeto se dejaría de ver, y el ultimo parámetro sería el *far,* que es parecido al *near*, pero en este caso lo que se controla es la distancia máxima a la que puede estar un objeto para que se puede visualizar, en este caso, si un objeto se encuentra a una distancia mayor de 1000 de la cámara, el objeto se dejaría de ver. La siguiente instrucción coloca la posición de la cámara en la posición

(0,0,10) de coordenadas x,y,z, ya que, por defecto, todos los elementos se introducen en el mismo punto el (0,0,0). También cuenta con otros atributos como el *lookAt* que indica a que coordenadas estaría mirando la cámara, en este caso no lo hemos modificado, ya que, por defecto son las coordenadas del origen, es decir, las coordenadas (0,0,0), en las que estará nuestro objeto.

Después, se ha definido el renderer, que en este caso se va a utilizar el *WebGLRenderer*, y a continuación se define el tamaño que se renderizará de la escena, en este caso será el tamaño de la ventana del navegador. Y por último, añadiremos el renderer que hemos creado al documento.

Con esto ya tendríamos todas las piezas para empezar a renderizar, pero por el momento seguiríamos sin ver nada, nos aparecería algo como esto:

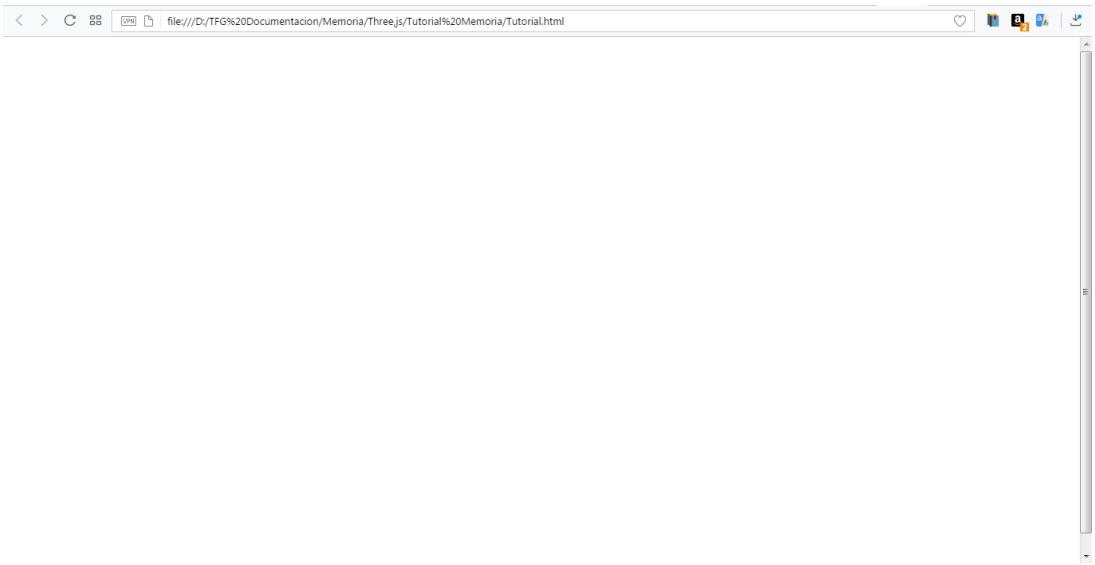

Figura 4.2.1. Página HTML en blanco

Esto ocurre, porque aparte de definir los elementos, como hemos hecho previamente, tendremos que renderizar la escena, para esto tenemos que introducir el bucle de animación y el renderizado de la cámara con la escena, el bucle de animación, conocido como r*equestAnimationFrame* se ejecuta 60 veces por segundo y todo lo que

se introduzca después de él y antes del renderizado de la cámara con la escena (`renderer.render( scene, camera )`), se ejecutara continuamente, a no ser de que se interrumpa el bucle de alguna forma.

Entonces lo que tendríamos que hacer sería definir una función como la que se verá en el siguiente recuadro de código (Código 4.2.3.), en la que al *requestAnimationFrame* se le está indicando que la función que va a ejecutarse en bucle es la propia *render()*, por lo que cuando hagamos una llamada a esa función, en nuestro caso denominada *render(),* entraremos en el bucle de animación, en el que estará continuamente renderizado todos los objetos que estén en la escena o otros nuevos que se vayan introduciendo con el transcurso de la ejecución.

```
function render() {
   requestAnimationFrame( render );

   // Aquí iría todo lo que quisiéramos animar
   // Esto se ejecutara cada frame (60 veces por segundo)

   renderer.render( scene, camera );
}
render();
```

Código 4.2.5. Bucle de animación y renderización de la cámara con la escena

Y ahora sí, después de haber definido la función de esta manera y hacer la llamada a dicha función, ya podremos visualizar el contenido de la escena, nos quedaría algo así en el navegador web:

Figura 4.2.2. Visualización de la escena vacía

En estos instantes ya estaríamos visualizando la escena, lo que pasa, es que no tenemos nada dentro de la escena, por eso se ve todo el espacio en negro, a continuación introduciremos la esfera que habíamos comentado anteriormente.

```
var sphereGeometry = new THREE.SphereGeometry( 1, 30, 30 );
var sphereMaterial = new THREE.MeshBasicMaterial( { color:
"#FFFFFF" } );
sphere = new THREE.Mesh( sphereGeometry, sphereMaterial );

scene.add( sphere );
```

Código 4.2.6. Creación e introducción de una esfera en la escena

THREE.SphereGeometry, crea la geometría de la esfera, donde, en este caso tiene tres argumentos, los cuales son, el primero es el radio de la esfera, que por defecto sería cincuenta, después, el segundo parámetro indica el número de segmentos horizontales que tendrá la esfera, que debe tener un mínimo de tres y por defecto su valor es ocho, y el tercer argumento es el número de segmentos verticales, que el mínimo son dos y por defecto son seis.

Después, se ha definido el *THREE.MeshBasicMaterial,* que es el material básico que se utiliza con muchos de los objetos, y en este caso al atributo propio *color* del material le hemos dado un color blanco, tiene otros tipos de atributos que se le pueden modificar, como el *wireframe*, que si ponemos este valor a *true* nos mostrará solamente las aristas de la malla, en nuestro caso las aristas de nuestra esfera, se puede observar un ejemplo en la figura 4.2.6.

Finalmente, aplicamos a la geometría que hemos definido, el material que hemos creado mediante *THREE.Mesh,* donde el primer argumento es la geometría, y el segundo el material a aplicar a esa geometría, y esto se guarda en la variable *sphere* que al no definirla como *var* pasa a ser una variable global, que podemos utilizar en otras funciones y demás.

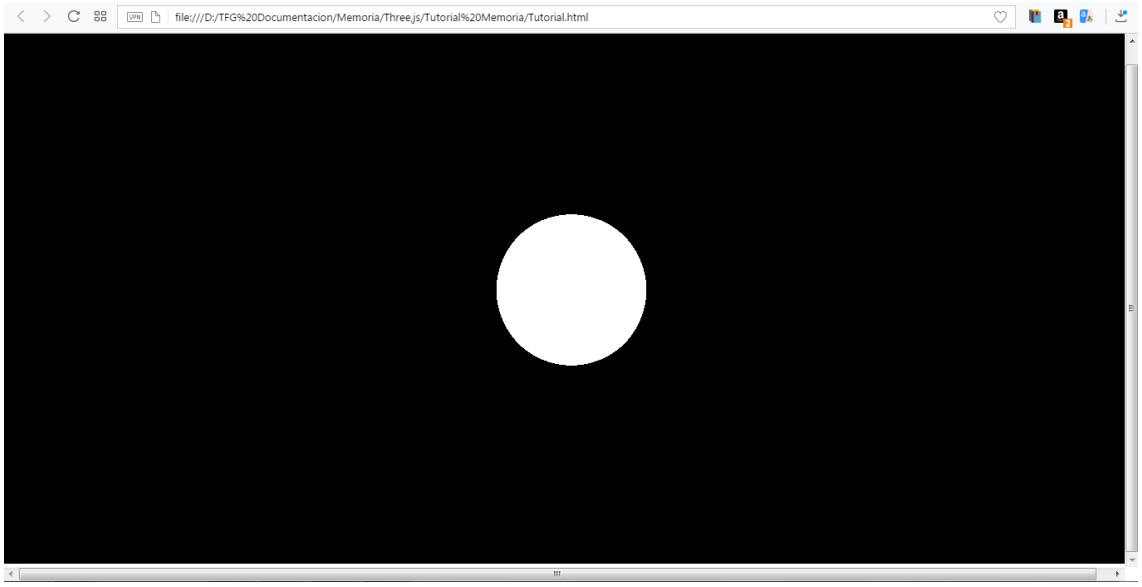

Figura 4.2.3. Visualización de esfera blanca

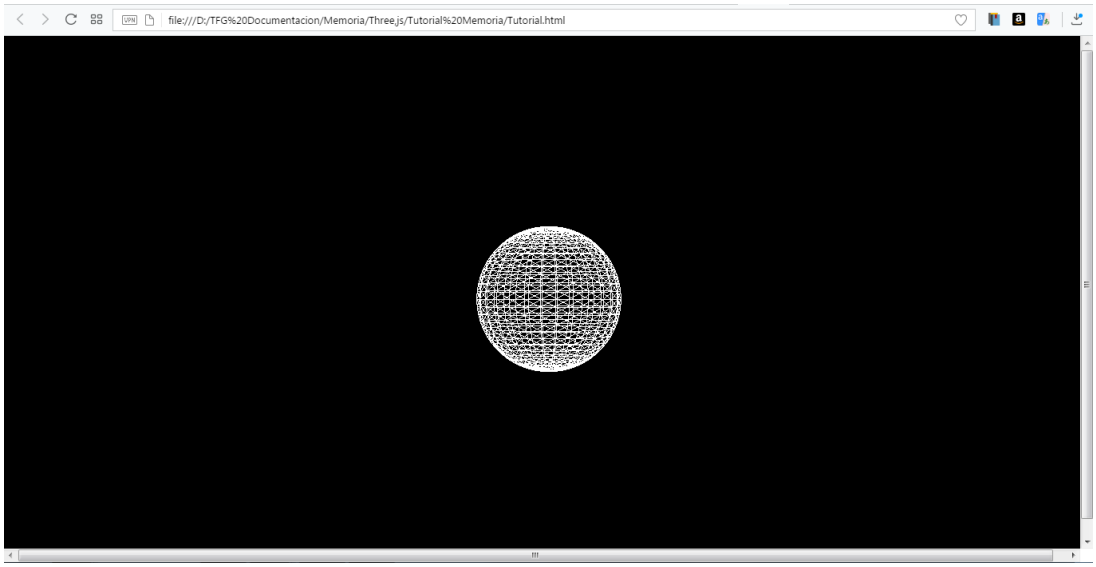

Figura 4.2.4. Visualización de esfera blanca wireframe

Y con esto ya tendríamos nuestra primera escena, y si queremos darle algo de movimiento para observar un pequeño ejemplo de animación lo que podemos hacer es, darle un movimiento de rotación en el eje 'y', simulando el paso del día en un planeta. Añadiendo el siguiente código

dentro del bucle de animación lo que haremos será darle ese movimiento de rotación a la esfera:

```
function render() {
   requestAnimationFrame( render );

            sphere.rotation.y += 0.002356;

   renderer.render( scene, camera );
}
```

Código 4.2.7. Creación e introducción de una esfera en la escena

La sphere al ser una variable global la podemos utilizar dentro de nuestro bucle de animación, por lo tanto, le sumamos un valor al atributo rotación en un eje y podemos ver como en estos momentos la esfera, en vez de estar parada, va girando continuamente sobre sí misma como si de un planeta se tratase, para observar bien el giro recomiendo en este caso poner el *wireframe* a *true*, ya que sin el cuesta más diferenciar el giro que realiza el objeto.

Se ha añadido una línea roja con el fin de hacer más visible el giro que hace el objeto en las imágenes siguientes:

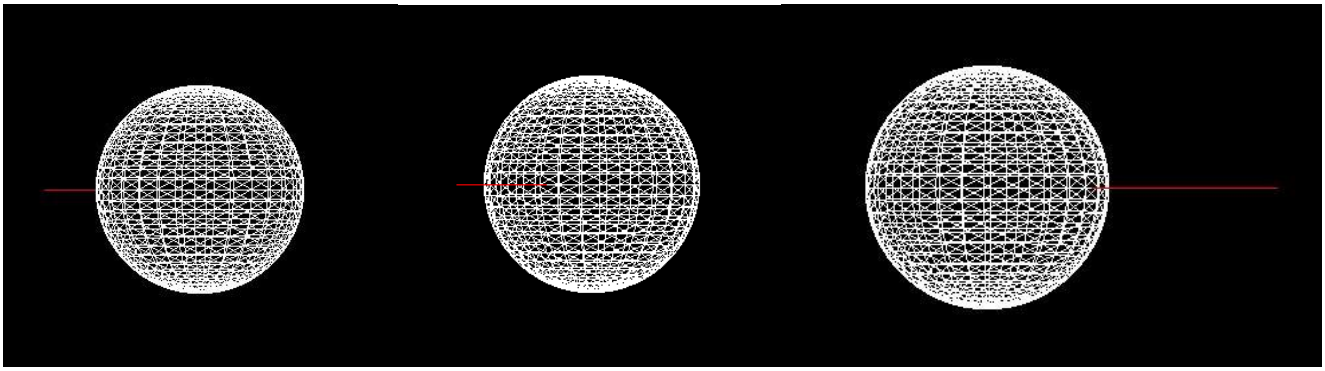

Figura 4.2.5. Rotación de esfera blanca

Three.js tiene muchas más funcionalidades y herramientas para desarrollar modelos 3D, animaciones y demás, como la carga de texturas en los objetos, la definición de diferentes tipos de controles de cámara, y un largo etcétera de posibilidades para nuestro navegador web.

5

Aplicación Web

5.1. Características

Esta aplicación cuenta con una interfaz gráfica web, que tendremos que saber utilizar para ejecutarla correctamente.

Está dividida en dos partes, la primera, es la página web inicial que está desarrollada en HTML, y contiene diferentes tipos de tecnologías web, como hojas de estilos en cascada, denominadas CSS, para definir el estilo de los elementos web, como tablas con clases de *Bootstrap*, como jQuery, el cual se ha utilizado para, por ejemplo, al seleccionar algún *radiobutton* o el botón de añadir la página se desplaza hacia arriba, para cambiar el tamaño de tablas, la visibilidad de botones en ciertas condiciones y demás, y para enlazar Three.js contiene JavaScript, además de para otras cosas, como recoger los valores de los campos a rellenar, o hacer control de errores por si no se han rellenado todos los campos y demás.

La segunda parte, una vez introducidos todos los datos y demos al botón de iniciar, se nos cambiará la página y entraremos en el entorno gráfico de Three.js, en el cual podremos ver los objetos introducidos que hayamos definido en la página anterior y el progreso que seguirán según el método seleccionado, aparte cuenta con algunas funcionalidades, y un gráfico donde se hace una pequeña muestra del error que se comete con el método de resolución seleccionado.

5.2. Interfaz

La interfaz que se ha definido para este proyecto tiene sus funcionalidades, pero mayormente se ha buscado realizar una interfaz simple, clara y limpia, la cual fuese lo más comprensible posible, donde se pudieran ver claramente los campos a rellenar, los datos introducidos y la forma de ejecutar la aplicación.

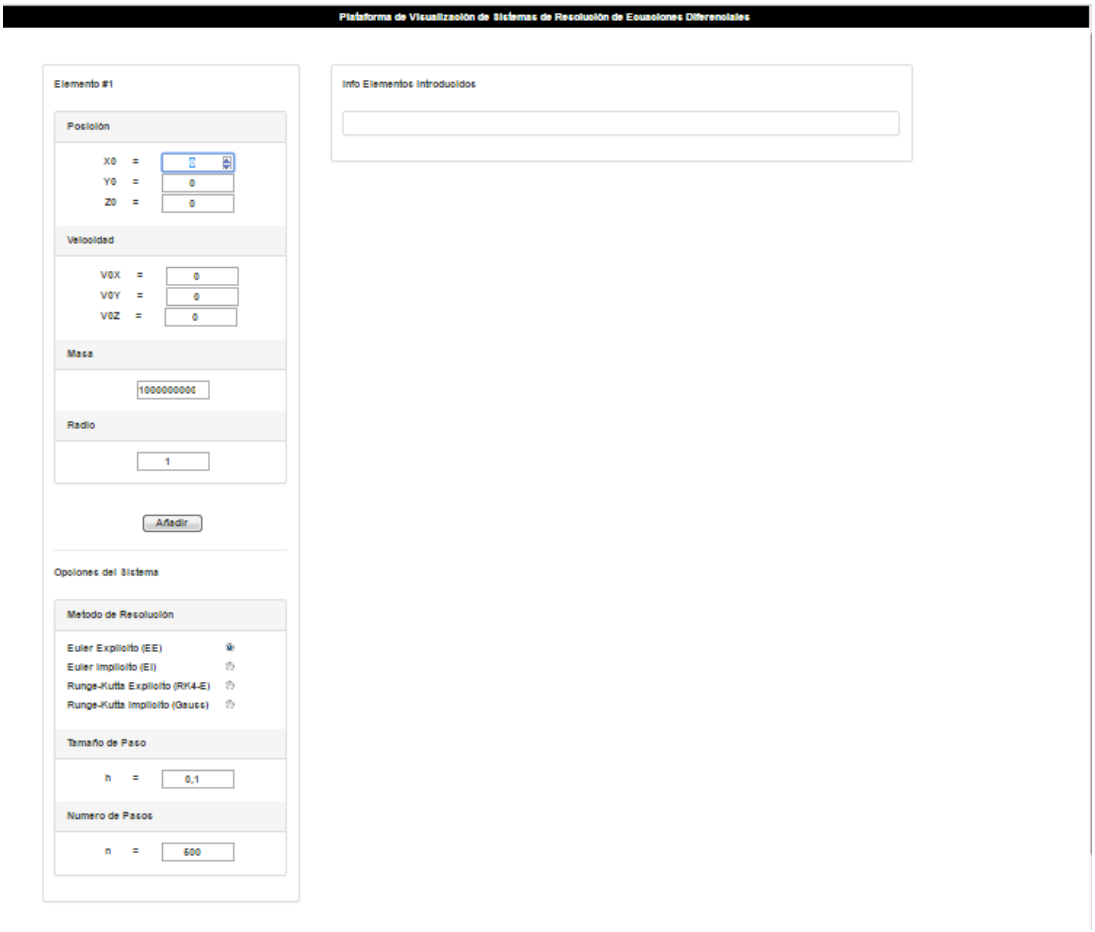

Figura 5.2.1. Página web principal completa

La primera parte de la página consta de una tabla que está dividida en dos partes, primero en la parte superior están los datos que hay que

introducir para el elemento que queramos introducir en el entorno, podemos introducir tantos elementos, en este caso cuerpos, como queramos, pero tenemos que tener en cuenta que cuantos más elementos introduzcamos más requisitos de computo se nos exigirán, por lo tanto, si tenemos un número elevado de cuerpos para los que ejecutar, el procedimiento puede que fuera más lento de lo normal.

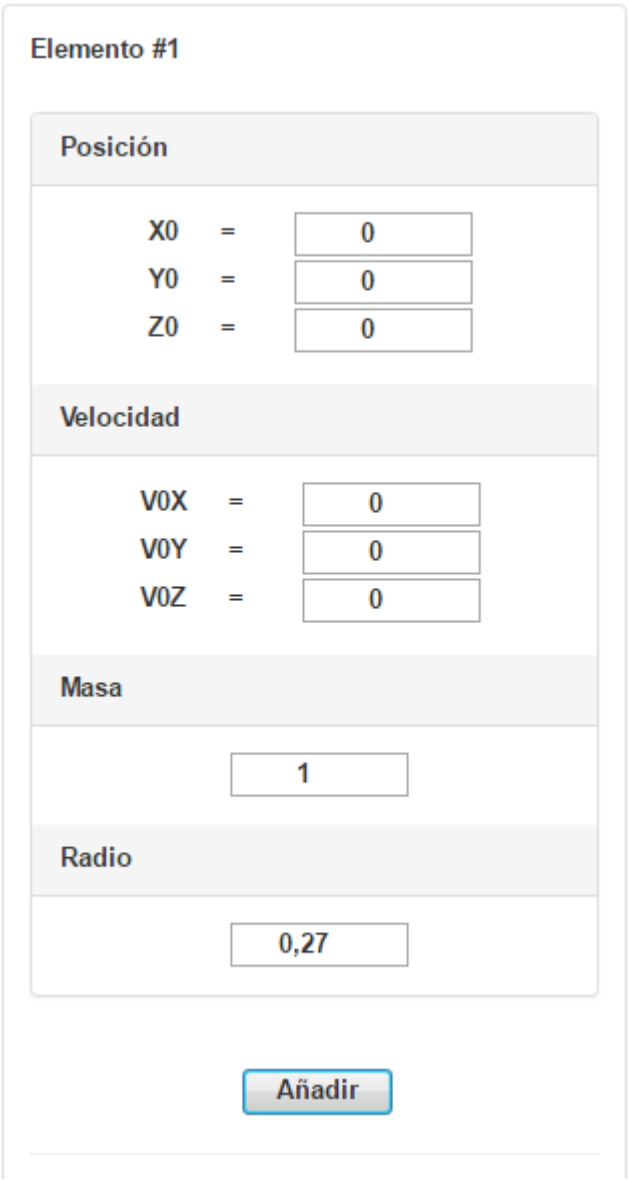

Figura 5.2.2. Interfaz web, tabla 1, parte superior

En esta sección de la página web, será donde introduzcamos los valores iniciales de todos los cuerpos que queramos introducir en el entorno grafico para ejecutar la aplicación.

Tenemos un título que enumera el elemento que se está introduciendo en este instante, que se incrementa según vayamos introduciendo elementos, el cual utiliza JavaScript para realizar esta acción, como se muestra en la siguiente figura.

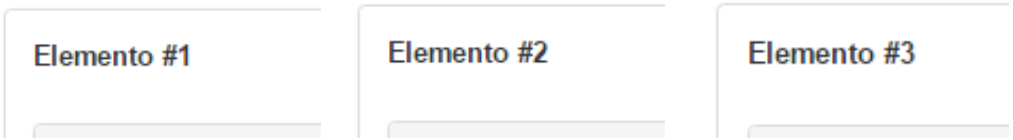

Figura 5.2.3. Interfaz web, incremento del título del elemento

A continuación, tendríamos los valores de posición que le queramos dar al objeto a introducir, que corresponde con las coordenadas X, Y, Z en el momento cero, es decir, en el valor inicial:

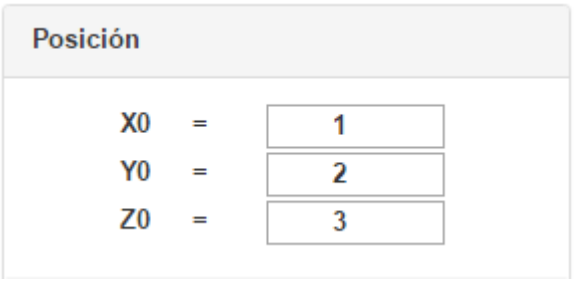

Figura 5.2.4. Interfaz web, campos de posiciones

Donde X0 hace referencia a la coordenada X en el instante cero, Y0 a la coordenada Y en el instante cero y Z0 a la coordenada Z en el instante cero, en este caso, tendríamos un objeto en la posición (1, 2, 3) del espacio.

Los siguientes campos a rellenar son los de la velocidad del cuerpo en el momento inicial, que corresponden a las velocidades en los ejes de coordenadas, X, Y, Z:

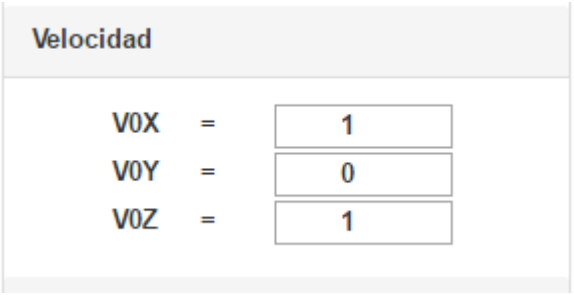

Figura 5.2.5. Interfaz web, campos de velocidades

Donde V0X hace referencia a la coordenada X de la velocidad en el instante cero, V0Y a la coordenada Y de la velocidad en el instante cero y V0Z a la coordenada Z de la velocidad en el instante cero, en este caso, tendríamos un objeto con un vector velocidad (1, 0, 1) en el momento inicial.

Después de la velocidad, tendríamos el campo asociado a la masa que tendrá ese objeto:

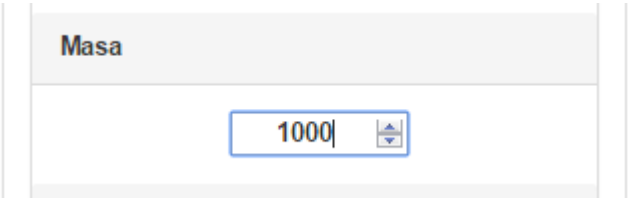

Figura 5.2.6. Interfaz web, campo de masa

Para el ejemplo que se puede observar en la figura, tendríamos un objeto de masa 1000, el cual esté no variará a lo largo de la ejecución del programa.

También, contamos con la opción de asignarle al objeto que queremos introducir, en nuestro caso una esfera, el radio con el que contará. La cual, se utilizará el siguiente campo para rellenar dicho dato:

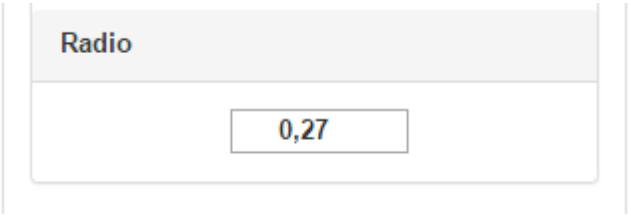

Figura 5.2.7. Interfaz web, campo de radio de la esfera

El cual, en este caso, se estaría introduciendo una esfera de radio 0.27.

Y por último, tenemos el botón de añadir, el cual lo que hará es introducirnos el elemento para el que hayamos introducido todos los datos, y así una vez ejecutemos el programa poder visualizar sobre el el método utilizado.

Figura 5.2.8. Interfaz web, botón añadir

Este botón tiene una funcionalidad extra, la cual, en el momento en que es pulsado, ejecuta una función de jQuery por la cual se hace *scroll* en la página para ubicarse arriba del todo de la misma.

En la segunda parte de la primera tabla se encuentran las opciones del sistema, donde tendremos los métodos de resolución de ecuaciones diferenciales que podremos ejecutar y el campo a rellenar con el valor de *h*, que corresponde al tamaño de paso.

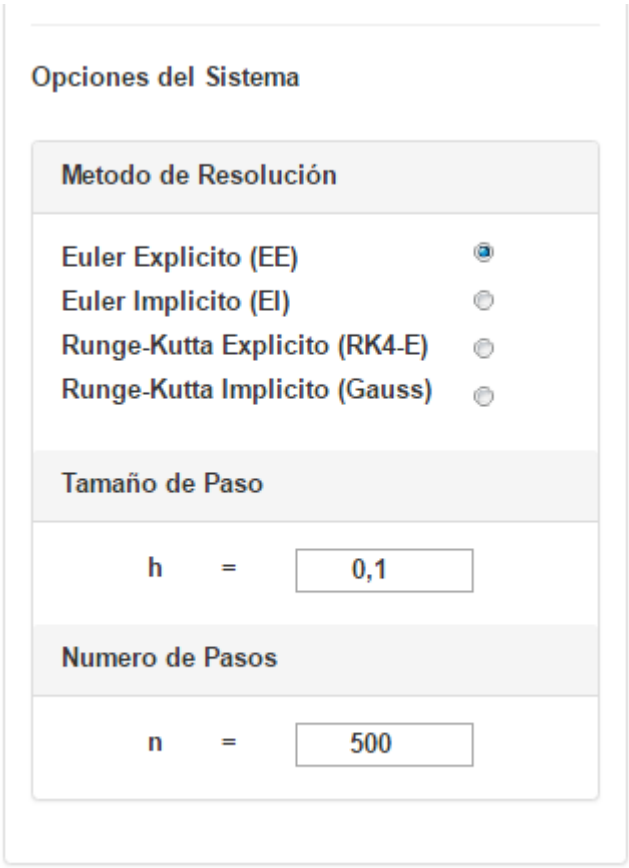

Figura 5.2.9. Interfaz web, tabla 1, parte inferior

Aquí encontremos primero los métodos de resolución disponibles para ejecutarse en el programa, que son el método de Euler explicito, el método de Euler implícito, el método de Runge-Kutta de cuarto orden explícito y el método de Runge-Kutta de cuarto orden implícito, podremos seleccionar cualquiera de ellos para resolver el problema, cada uno tiene su tablero de Butcher correspondiente asociado.

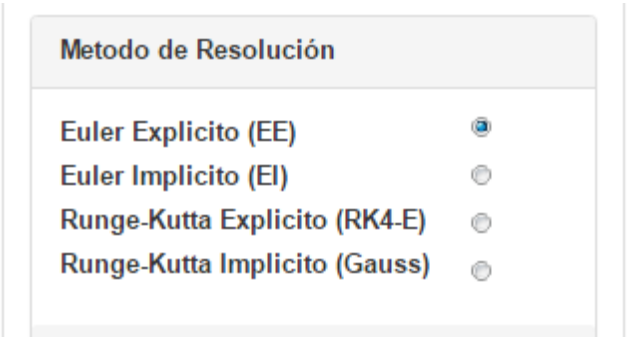

Figura 5.2.10. Interfaz web, métodos de resolución

Estos *radiobuttons* tienen la misma funcionalidad que el botón añadir mencionado anteriormente, al seleccionar uno de los métodos utilizará la misma función de jQuery, la cual, desplazará la ubicación de la página hasta la parte de arriba de la misma.

A continuación, en esta primera tabla tenemos el campo para definir el tamaño de paso que se utilizará a la hora de resolver el problema, el tamaño de *h*.

Figura 5.2.11. Interfaz web, campo de tamaño de paso

Por último, contamos con un campo asociado al número de pasos que ejecutaría el método.

Figura 5.2.12. Interfaz web, campo de numero de pasos

En este caso contaríamos con un número de pasos igual a quinientos, lo que quiere decir, que cuando alcanzase ese pasó, se detendría y habríamos llegado a dicha situación, y también se ha implementado la opción de que si introducimos un numero negativo en ese campo, lo que obtendríamos sería la ejecución del programa de forma infinita, sin un determinado numero de paso final.

Esto es todo para la primera tabla que tenemos en la página web principal, ahora vamos con la siguiente tabla y el botón de iniciar el programa.

En esta segunda tabla de primeras tenemos una tabla vacía con cero elementos en ella, como se puede observar en la siguiente figura:

Figura 5.2.13. Interfaz web, tabla 2, información de elementos introducidos

En esta tabla, a medida que se van introduciendo objetos al programa, irán apareciendo aquí en modo lista, cada una de las filas contendrá al número del objeto, la posición que ocupa en las coordenadas X, Y, Z, los valores de velocidad para las correspondientes coordenadas X, Y, Z, la masa del mismo y el radio de la esfera. En la siguiente figura veremos un ejemplo de tres objetos introducidos hasta el momento.

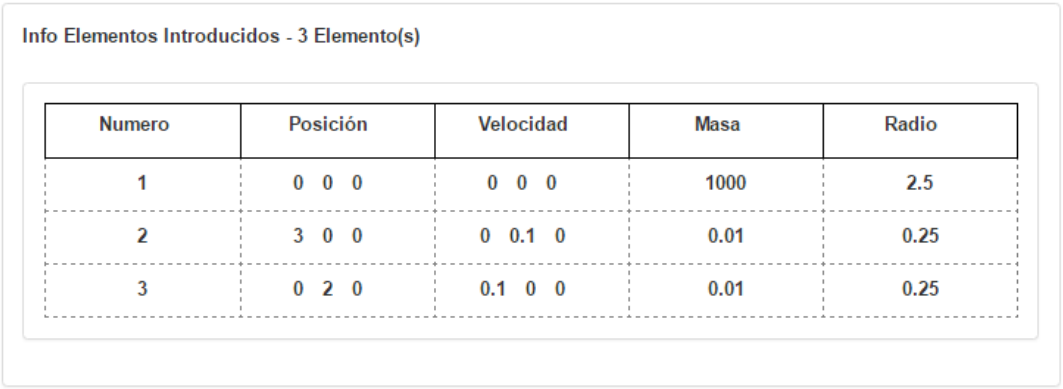

Figura 5.2.14. Interfaz web, tabla 2, información de elementos introducidos 2

Además, está tabla incorpora JavaScript para la actualización de los elementos en la lista y del título de la tabla que indica el número de elementos introducidos en la misma, funciona de la misma forma que el título de la primera tabla anteriormente mencionado, y jQuery para el caso de tener demasiados elementos listados, define un tamaño de tabla y se convierte en una tabla con barra de desplazamiento, para ser capaz de ver los demás elementos que hay introducidos sin incrementar la altura máxima de la tabla.

Y para terminar, en el momento que introduzcamos el primer objeto en esta lista, se hará visible el elemento HTML que contiene al botón iniciar, para poder iniciar el programa.

Figura 5.2.15. Interfaz web, botón iniciar

Este botón también incorpora funciones de jQuery, que es el que hace que el elemento HTML que contiene al botón se vuelva visible.

Ahora vamos con la interfaz del entorno Three.js, la apariencia inicial del entorno grafico es la siguiente:

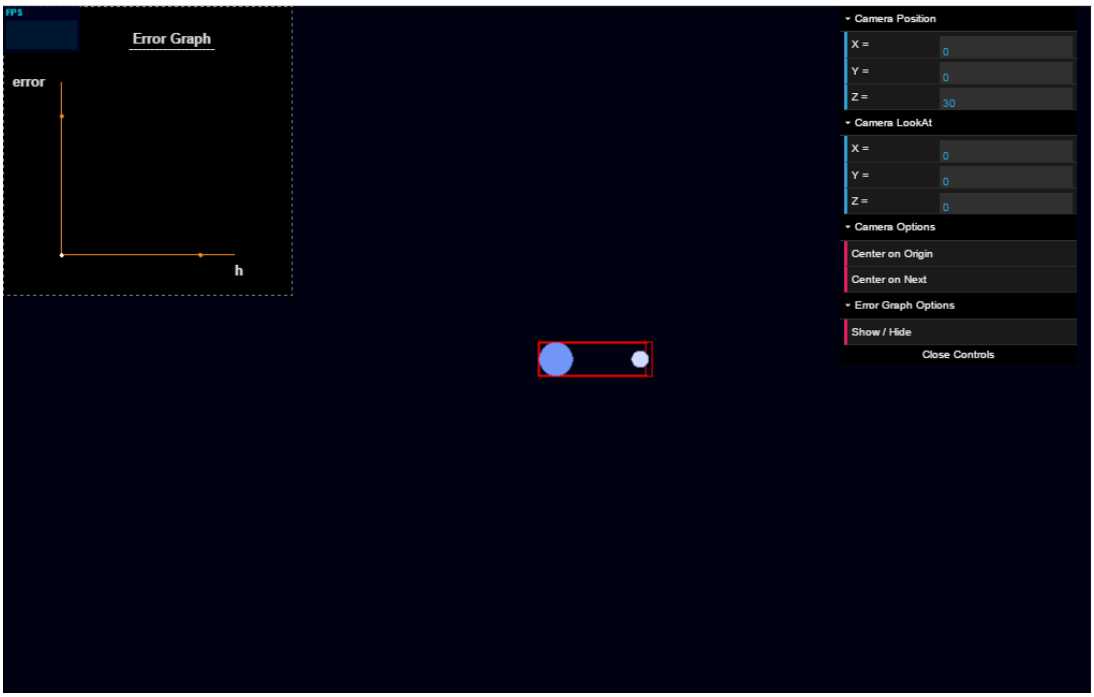

Figura 5.2.16. Interfaz entorno gráfico, entorno grafico completo

El entorno grafico tiene tres partes principales, la primera es la escena renderizada con la cámara, como hemos comentado anteriormente, Three.js utiliza una escena, un renderer y una cámara para visualizar el contenido de la escena, esa escena es en la que se verá la ejecución del programa y en la que podremos interactuar de alguna forma, con la cámara más concretamente.

A continuación, tenemos un menú desplegable, en el cual tenemos las siguientes opciones para realizar en nuestra aplicación, que son:

- Activar o Desactivar Grafica de Error
- Cambiar Posición Cámara
- Cambiar Dirección de Visión de Cámara
- Centrar cámara en el origen, dirección y ubicación
- Centrar la cámara en el siguiente objeto

Este menú cuenta con directorios, los cuales, tienen la funcionalidad de poderse plegar y desplegar en función de lo que desee el usuario, también se podría plegar el menú entero si se quisiese, como se muestra en la figura siguiente:

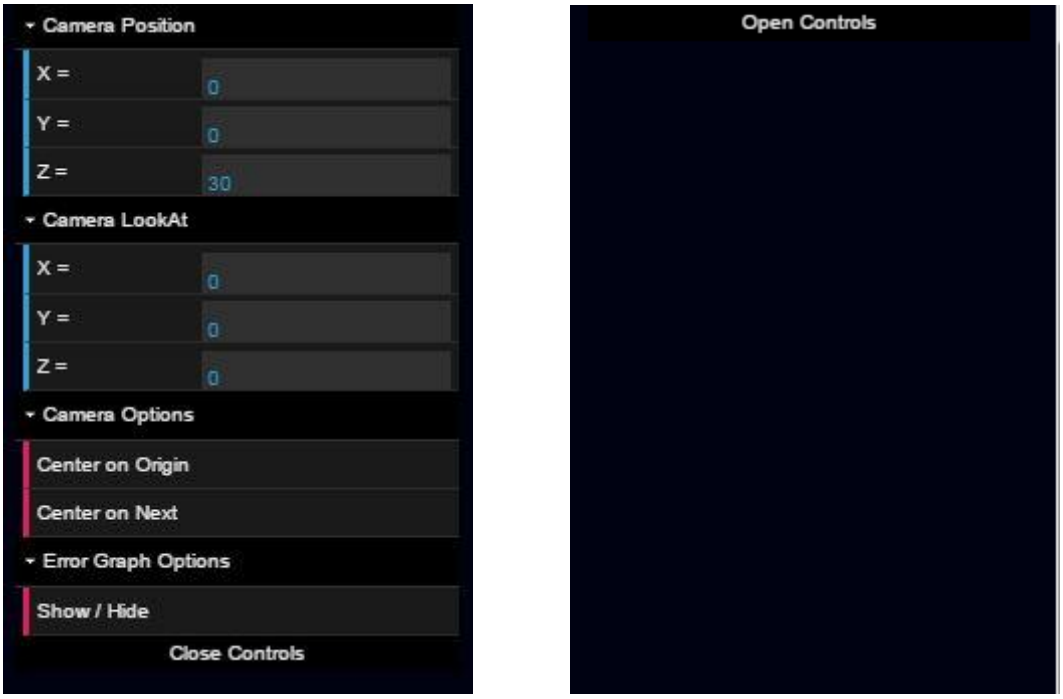

Figura 5.2.17. Interfaz entorno gráfico, menú, desplegado (izquierda) y plegado (derecha)

El menú está organizado en función de esos directorios, el primer directorio corresponde a la posición de la cámara, en la que se mostrarán los valores correspondientes a las coordenadas X, Y, Z en las que se encuentra ubicada la cámara, estos campos son modificables y repercutirán en la cámara, esto lo veremos más ampliamente en el siguiente apartado.

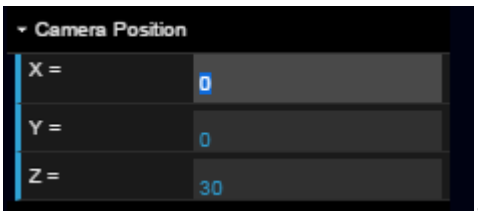

Figura 5.2.18. Interfaz entorno gráfico, menú, directorio ubicación cámara

El siguiente directorio del menú es el correspondiente al *LookAt* de la cámara, aquí se puede cambiar la dirección de la visión de la cámara, es decir, podremos ubicar el objetivo de la cámara.

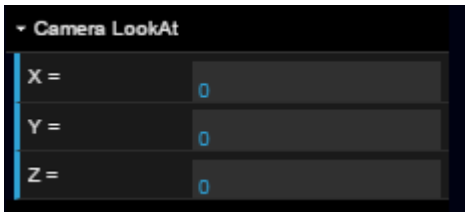

Figura 5.2.19. Interfaz entorno gráfico, menú, directorio LookAt cámara

Debajo de este directorio, iría el de las opciones de la cámara, que cuenta con dos botones, el primero de nombre *Center on Origin* lo que hace es centrar la cámara en el origen de coordenadas, tanto el LookAt como la ubicación de la cámara en sí, excepto para el eje Z de la ubicación de la misma, que se modifica con el valor diez, y segundo tendremos un botón llamado *Center on Next*, el cual centrará la cámara y el LookAt de la misma, en el siguiente objeto que tengamos en la escena, en el caso de que estuviéramos centrados con la cámara en el último objeto de la escena pasaríamos a centrar la cámara en el primer objeto directamente.

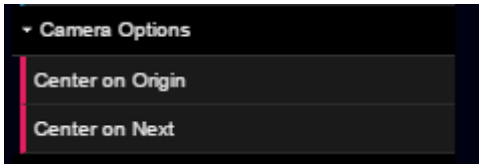

Figura 5.2.20. Interfaz entorno gráfico, menú, directorio opciones de la camara

Por ultimo tendremos el directorio correspondiente a las opciones de la gráfica de error que se muestra en el entorno gráfico.

Figura 5.2.21. Interfaz entorno gráfico, menú, directorio opciones de la gráfica de error

Este directorio cuenta con un botón que tiene la funcionalidad de poder mostrar y ocultar el grafico de error que tenemos integrado en el entorno gráfico, y el *Close Controls* que aparece al final del mismo menú es el botón que pliega el menú completo cuando está abierto, cambia de nombre y se ubica en la parte de arriba de la ventana y cuando esta cerrado también se modifica el valor del nombre y se despliega el menú completo.

Eso es todo sobre el menú, la interfaz del entorno también cuenta con la gráfica de error anteriormente citada, que sería la siguiente:

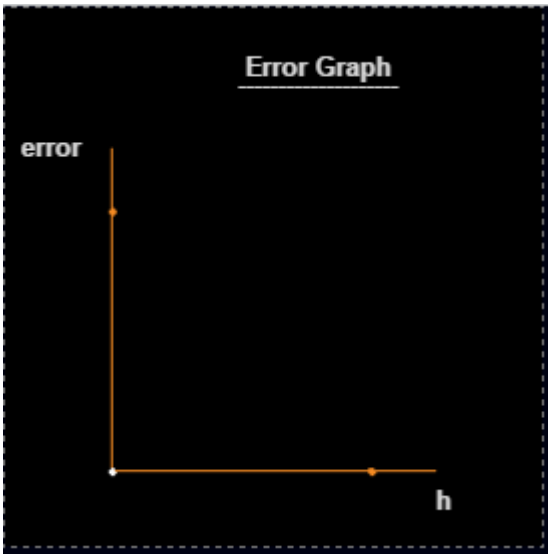

Figura 5.2.22. Interfaz entorno gráfico, grafica de error

Esta grafica de errores estaría ubicada en la parte superior izquierda de nuestra ventana, la cual, nos permitirá observar el error que está cometiendo el método en cada paso de tamaño *h*, el eje horizontal define los pasos de *h,* y el eje vertical define el error cometido.

5.3. Funcionamiento

En esta sección explicaremos como se ha realizado el proyecto y el funcionamiento que tiene el mismo.

El proyecto se ha realizado siguiendo las pautas explicadas anteriormente, descritas en la sección para el funcionamiento de Three.js, el proyecto está divido en dos partes la página web principal y el entorno grafico creado con WebGL y Three.js.

Por lo tanto, lo primero con lo que cuenta es con una página web desarrollada en HTML5, con sus respectivos estilos mediante una hoja de estilos en cascada CSS, JavaScript para enlazar las diferentes librerías que utilizaremos, como Three.js, y para control de errores, actualización de datos y otras funcionalidades, como las de la librería jQuery, eso a modo de resumen de la página web principal.

Para utilizar correctamente esta página web tendremos que seguir los pasos que comentaremos a continuación.

Primero, tenemos que introducir el número de objetos, con sus estados iniciales, para los que queramos que se ejecute el programa, a medida que vamos introduciendo cuerpos a nuestra plataforma, se observa en la tabla de la misma página el número de objetos, y los datos que tienen cada uno.

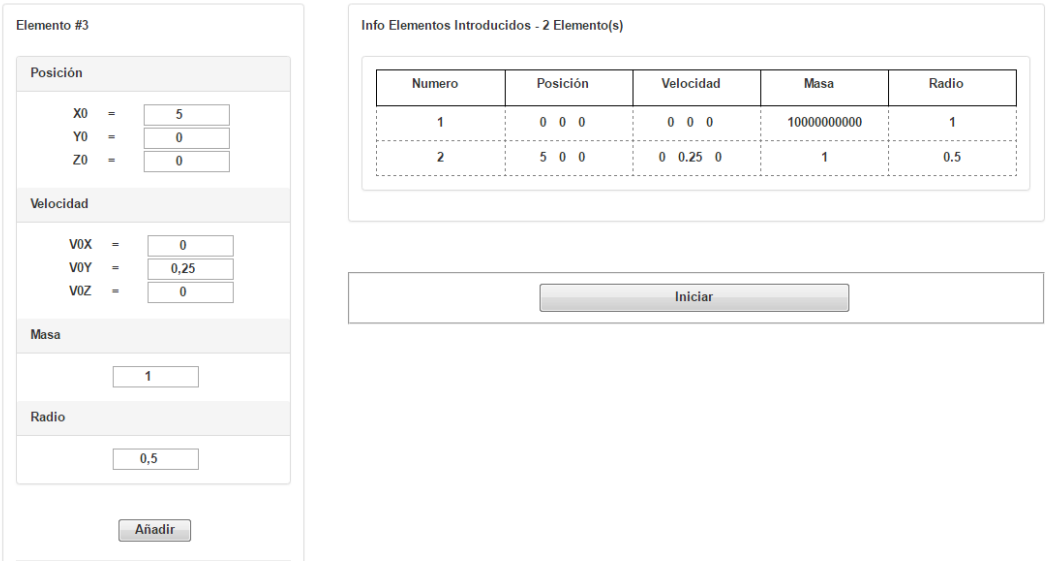

Figura 5.3.1. Página principal con dos cuerpos introducidos.

En la anterior figura tenemos dos cuerpos introducidos en la tabla, los cuales, ya estarían a disposición de introducirse en la escena.

A continuación, lo que tendríamos que hacer es completar las opciones del sistema que son las siguientes:

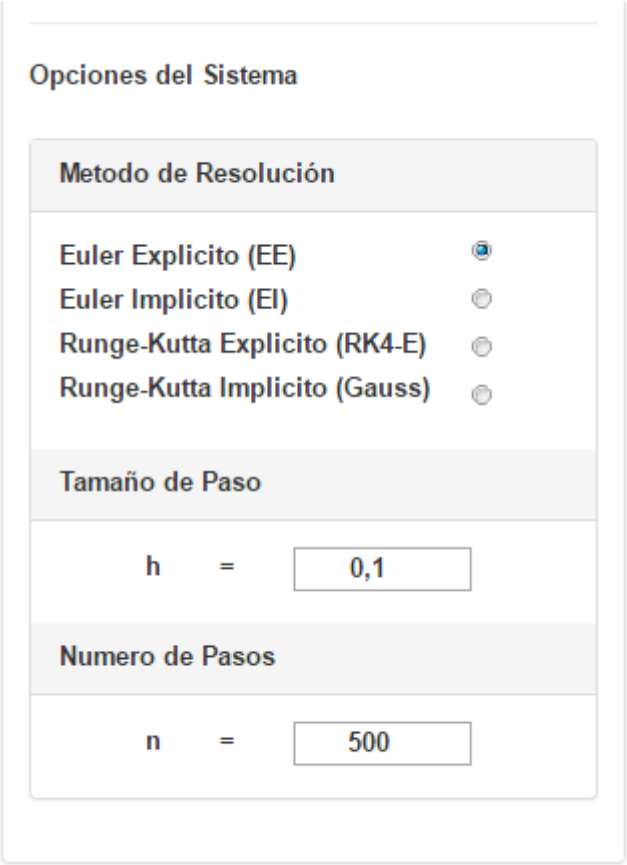

Figura 5.3.2. Opciones del Sistema

Esta parte contiene las opciones con las que ejecutaremos el programa, cuenta con una opción para elegir el método de resolución que queramos, los métodos que se han implementado ha sido los siguientes:

- Método de Euler Explicito: Es un método perteneciente a la familia de los métodos de Runge-Kutta, el cual, es el más sencillo de ellos, es de orden uno y de una única etapa, por lo tanto, el error es de orden *h*.
- Método de Euler Implícito: Es el mismo método que el anterior pero en su forma implícita, el cual, necesita de un cálculo previo de los puntos intermedios, en este caso un único paso intermedio mediante el método del punto fijo, para que se pueda resolver su forma implícita.

- Método de Runge-Kutta Explicito (RK4): Este método es de orden cuatro y cuatro etapas, en este caso está implementado para su forma explícita, por lo que el error cometido por el método es de orden *h* a la cuarta.
- Método de Runge-Kutta Implícito (Gauss): En este caso tendríamos un método de Runge-Kutta implícito, más concretamente el método de Gauss de tres etapas y orden seis.

Lo que hace la selección de uno de estos métodos es cargar todo lo necesario para que se ejecute el método que hemos seleccionado.

Cuando tengamos seleccionado el método procederíamos a introducir el tamaño de paso *h* para el que queramos ejecutar el programa, lo recomendable es introducir un tamaño de paso pequeño, ya que, el error que cometen los métodos está asociado al tamaño del paso, por lo tanto, si optamos por introducir uno grande no obtendríamos resultados satisfactorios.

Y por último, tendremos que introducir el número de pasos para los que queremos ejecutar la aplicación, el número de pasos determinará el número de veces que se ejecutan los métodos, y en el momento que se detenga el método habremos obtenido el estado para ese paso que hemos introducido, como se muestra en la siguiente figura:

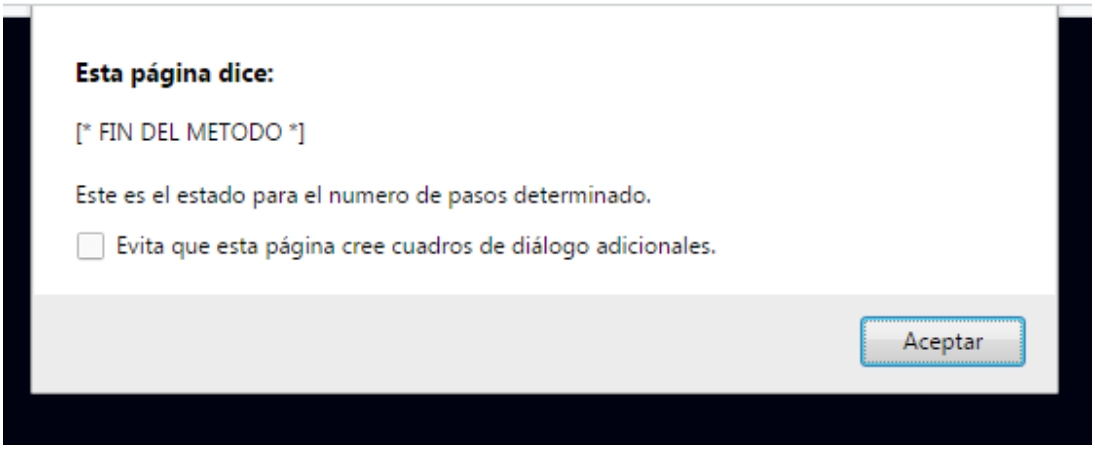

Figura 5.3.3. Mensaje de finalización del método.

En el caso de que no queramos visualizar un numero de pasos determinado y solamente queramos ver cómo se va ejecutando el método, se ha implementado la opción de que si introducimos un numero negativo como numero de pasos del método, el método pasará a ejecutarse continuamente hasta que cerremos la aplicación.

Para introducirse en el entorno gráfico, finalmente, con todos los valores anteriormente citados ya completados, nos habrá aparecido debajo de la tabla de información de los cuerpos introducidos, un botón con la palabra *Iniciar* como este:

Figura 5.3.4. Botón de iniciar el programa.

El cual en cuanto lo pulsemos entremos directamente en el entorno gráfico, en el que ya podremos visualizar los objetos que hayamos introducido en la escena.

La segunda parte del proyecto trata del entorno gráfico, lo primero que se ha definido ha sido la cámara, escena y renderer, este último se ha definido como *WebGLRenderer*, con el que utilizaríamos directamente la tecnología de WebGL, además de ser más rápido, que el CanvasRenderer, por ejemplo.

La cámara es una PersepectiveCamera, posicionada inicialmente en las coordenadas del primer objeto introducido y a una distancia treinta veces mayor a su radio, para así tener una cámara más dinámica que una que empiece siempre desde un punto fijo, por convención, en principio, el primer elemento que se introducirá será el sol, si representamos un sistema solar.

Las coordenadas de la cámara luego las podremos modificar mediante el menú que tenemos integrado en el entorno, es una cámara dinámica la cual utiliza la funcionalidad de OrbitControls que proporciona Three.js, con la cual, también la podremos manejar de forma que, con el botón izquierdo del ratón podamos rotar la escena, con el botón derecho podamos trasladar de ubicación la escena, y con la ruleta del *mouse* podremos hacer zoom, el zoom también lo podemos hacer dejando pulsado el botón central del ratón y deslizando el mismo para acercar y alejar.

A la cámara, se le han añadido dos formas más de establecer su ubicación y dirección visual, como hemos comentado anteriormente, el menú lleva integradas varias opciones para manejar la cámara en plena ejecución del programa, por tanto, podremos manejar la cámara a nuestro gusto.

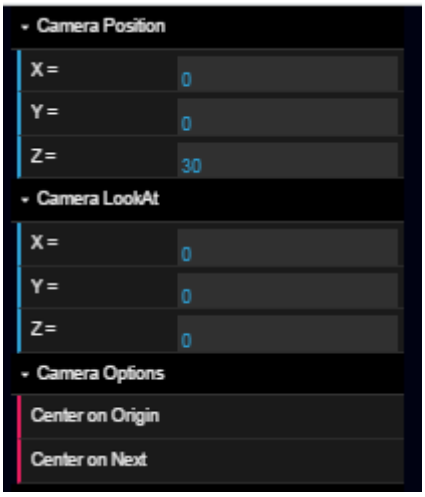

Figura 5.3.5. Menú de la cámara.

La escena cuenta con todos los elementos que le hayamos introducido en la página web previa que explicábamos al principio de este apartado, además del menú y de la gráfica de error que están integrados en la misma escena.

Ahora, una vez dentro del entorno gráfico, al principio tendremos un mensaje de bienvenida, que cuando estemos listos para que empiece a ejecutarse el método podemos cerrar o dar a aceptar.

En ese instante ya tendríamos la aplicación con nuestros datos introducidos ejecutándose, y podríamos ver cómo los cuerpos van siguiendo el método según el método seleccionado.

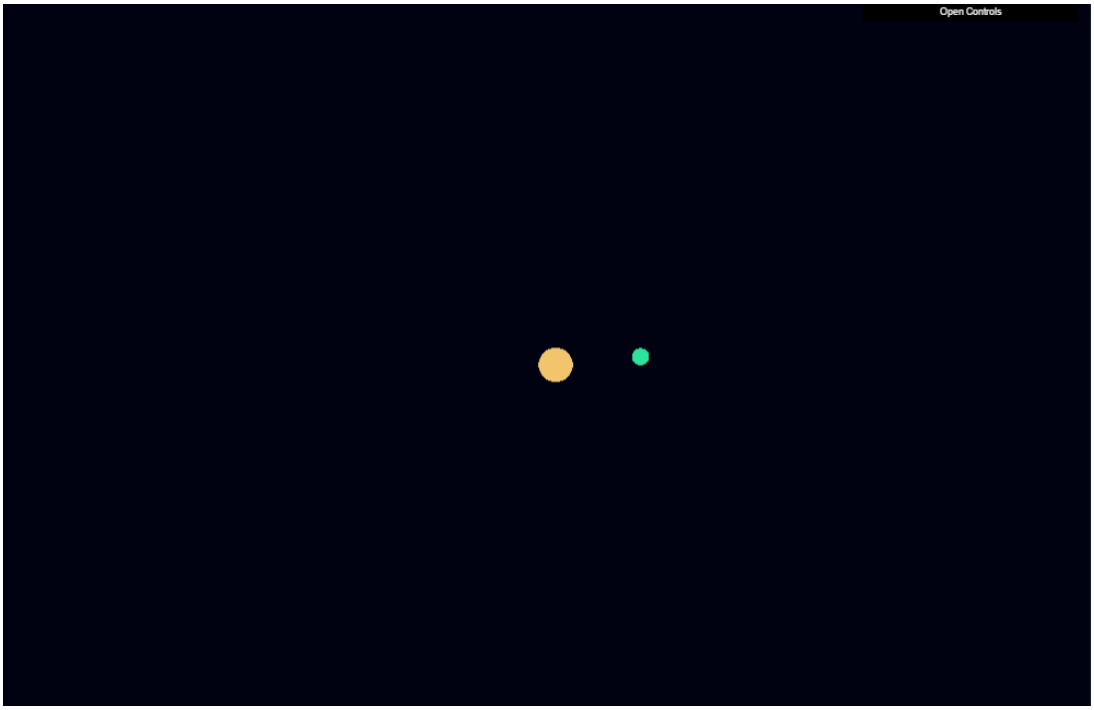

Figura 5.3.6. Entorno gráfico, estado inicial

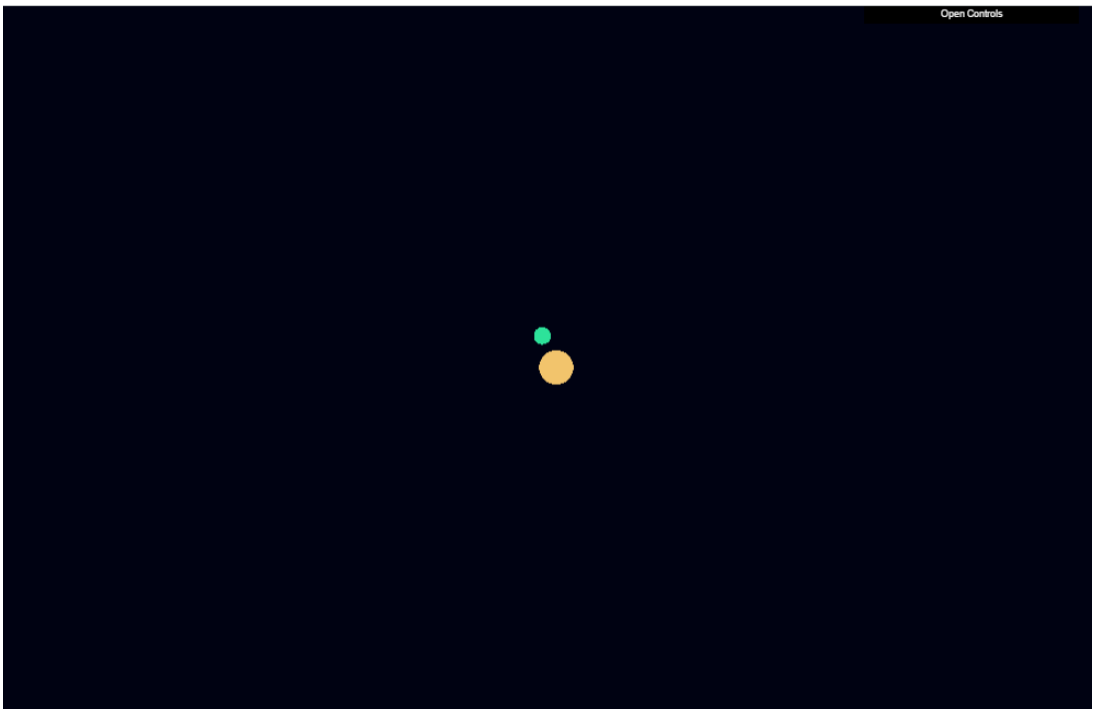

Figura 5.3.7. Entorno gráfico, estado objetos, numero de paso 206

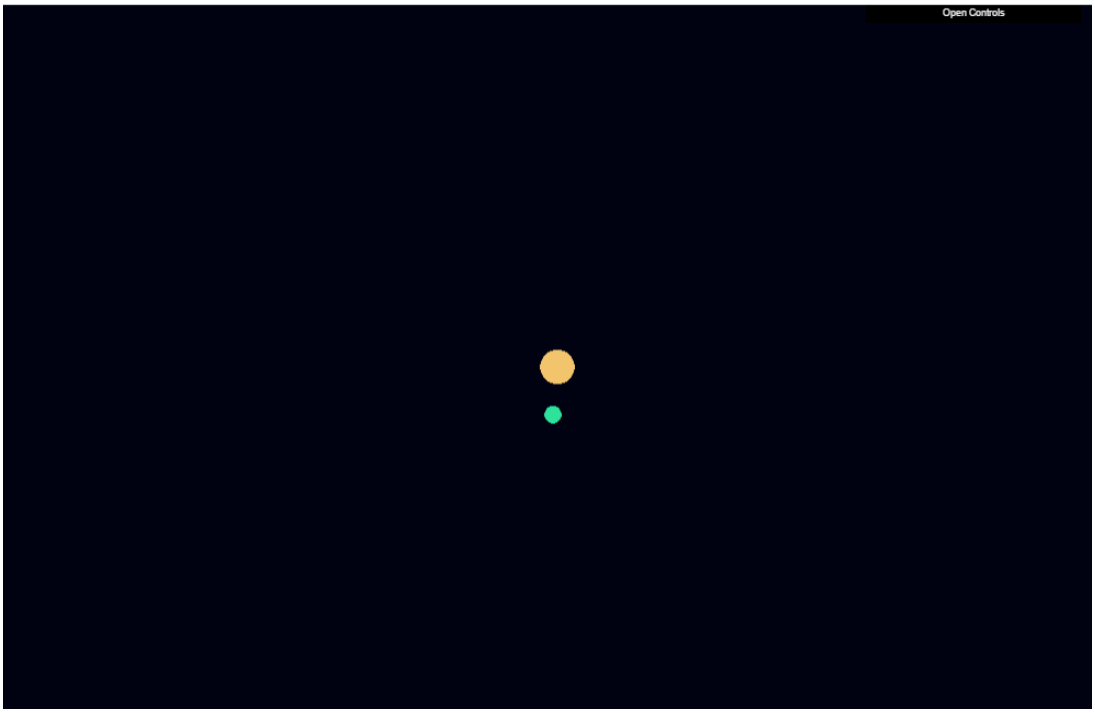

Figura 5.3.8. Entorno gráfico, estado objetos, numero de paso 280

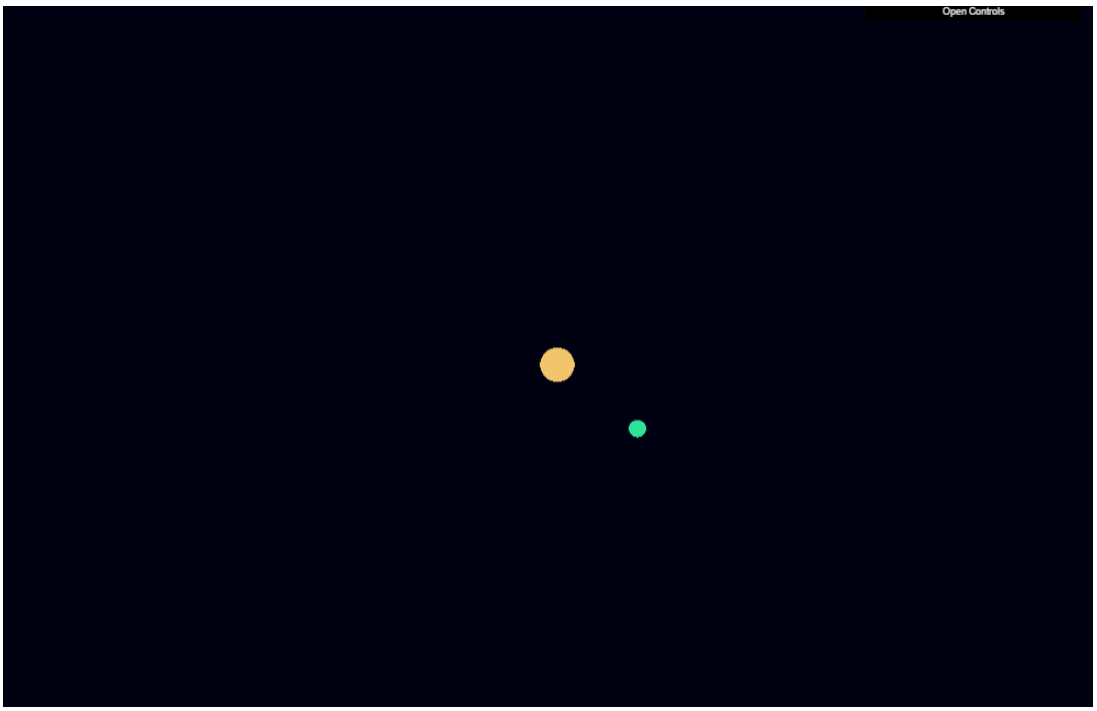

Figura 5.3.9. Entorno gráfico, estado objetos, numero de paso 403

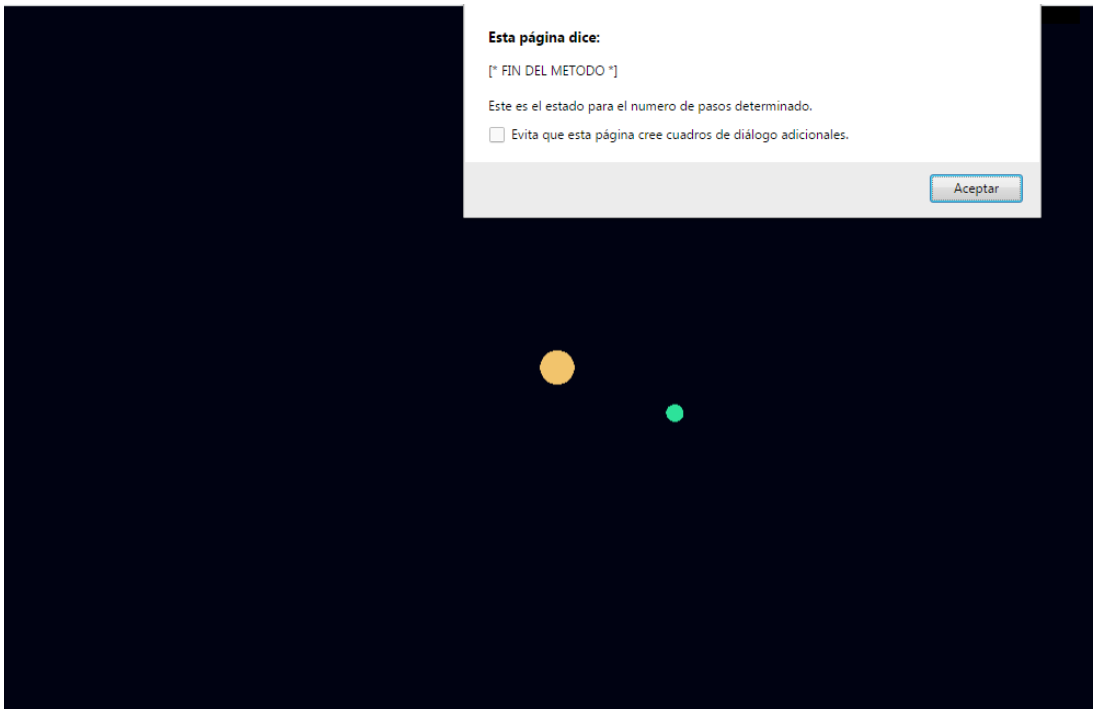

Figura 5.3.10. Entorno gráfico, estado objetos, numero de paso 500, final

En las anteriores imágenes se muestra la trayectoria que siguen dos cuerpos con un estado inicial y un método seleccionado, y en este caso para un numero de pasos igual a quinientos.

Finalmente, habríamos llegado al estado final y el programa habría acabado.

Podemos observar que la trayectoria que sigue es una órbita del cuerpo de tamaño menor alrededor del cuerpo de tamaño superior, esto se debe a los valores iniciales introducidos, y a las leyes de gravitación de Newton que se han implementado en el proyecto, el cuerpo de mayor tamaño al tener una masa muy elevada y el otro cuerpo tener una masa más pequeña y una velocidad solamente en el eje *y*, se ejerce una fuerza de atracción sobre el cuerpo de menor masa, por lo que esto repercute en el cambiando la trayectoria del mismo, y finalmente creando una órbita alrededor del cuerpo de mayor tamaño .

En este caso los datos iniciales han sido los siguientes:

- Cuerpo 1:
    - Posición (0,0,0)
    - Velocidad (0,0,0)
    - Masa = 10000000000 (1^10)
- Cuerpo 2:
    - Posición (5,0,0)
    - Velocidad (0, 0.25, 0)
    - Masa = 1

En este caso, no se ha movido la cámara pero podríamos realizar diferentes acciones como acercarnos al cuerpo que tenemos centrado, que como hemos mencionado anteriormente inicialmente es el primer objeto, para ver más de cerca la trayectoria.

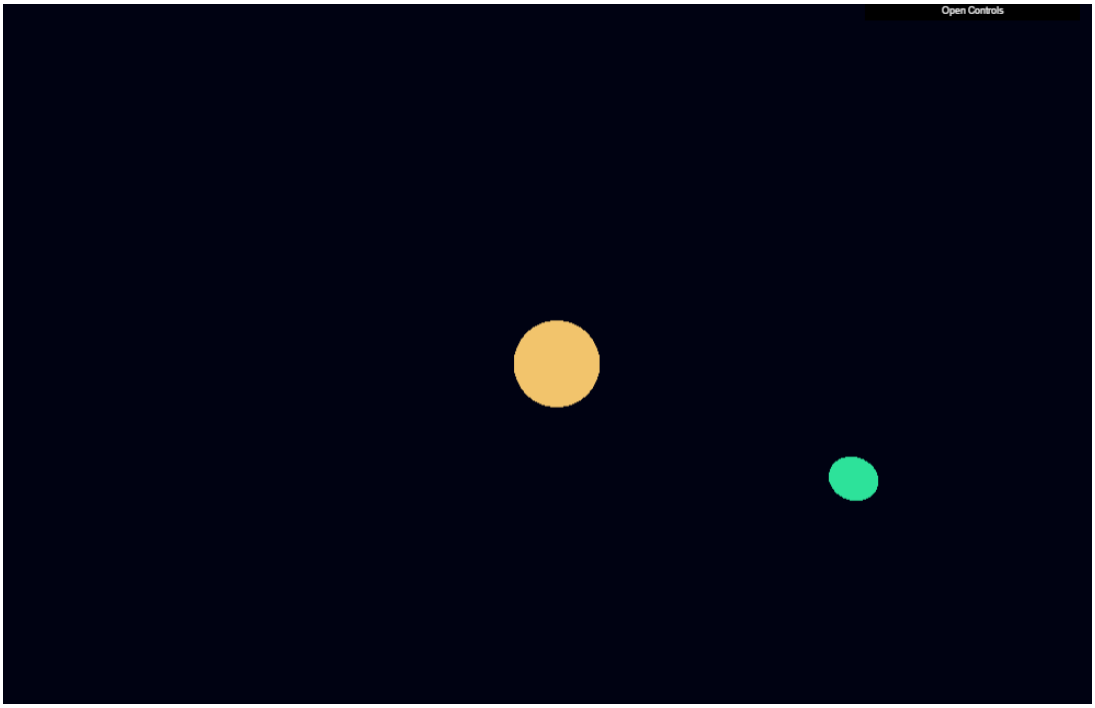

Figura 5.3.11. Entorno gráfico, cuerpos acerados mediante zoom

También podríamos centrar la cámara en el otro objeto para ver cómo sigue el método desde la perspectiva del otro cuerpo.

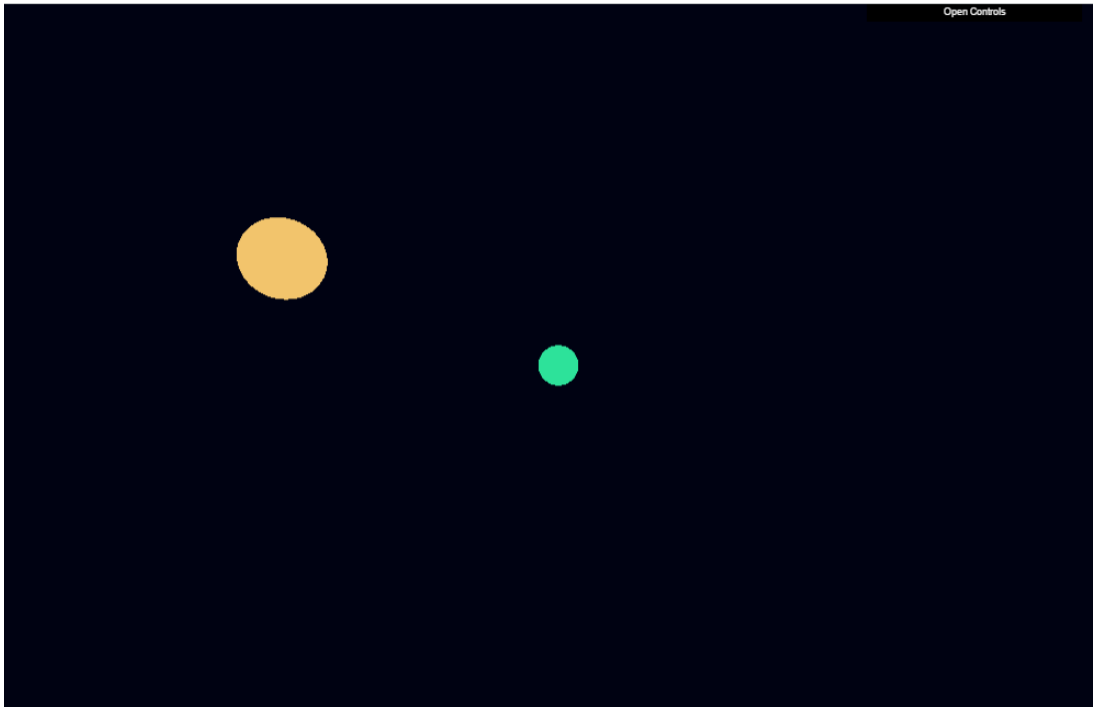

Figura 5.3.11. Entorno gráfico, cámara centrada en siguiente objeto

La cámara la podríamos manejar de muchas maneras, ya sea, mediante el ratón, rotando, trasladando, acercando y alejando la escena, como mediante el menú integrado en la aplicación, centrando en el origen, en un cuerpo u otro, moviendo la dirección de la visión de la cámara o trasladando la propia cámara a un punto de las coordenadas concreto que queramos.

Después, lo que podríamos hacer es probar con los diferentes métodos, en este caso se estaba utilizando el método de Euler explicito, para ver que trayectoria siguen o cual sería el estado final para ese mismo número de pasos, si hay desviación en la orbita según el método y demas.

5.4. Errores

Para este proyecto se ha realizado un control de errores tanto en la interfaz web como en el entorno gráfico de Three.js, aparte en este último se hace una muestra del error cometido por el método.

5.4.1. Web

El primer control de error que se implementó en la página web principal, fue el siguiente, si alguno de los campos de la parte superior de la primera tabla de la página ha quedado vacío, se ejecuta un *alert* que indica que ha habido un error, y que se han de rellenar correctamente todos los campos.

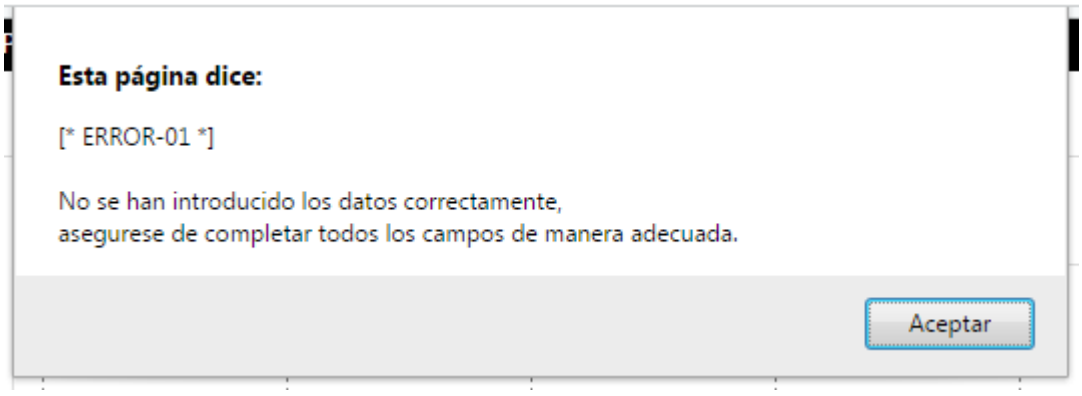

Figura 5.4.1.1. Mensaje de Error 01

A continuación, podríamos insertar el valor y continuar introduciendo los datos normalmente o iniciar el método.

El siguiente control de error que se implementó fue el control de que el campo correspondiente al valor *h* y al valor de *n*, hayan podido quedar vacíos cuando se pulsa el botón iniciar, por lo tanto, en tal caso, aparecería un *alert* con el siguiente mensaje de error:

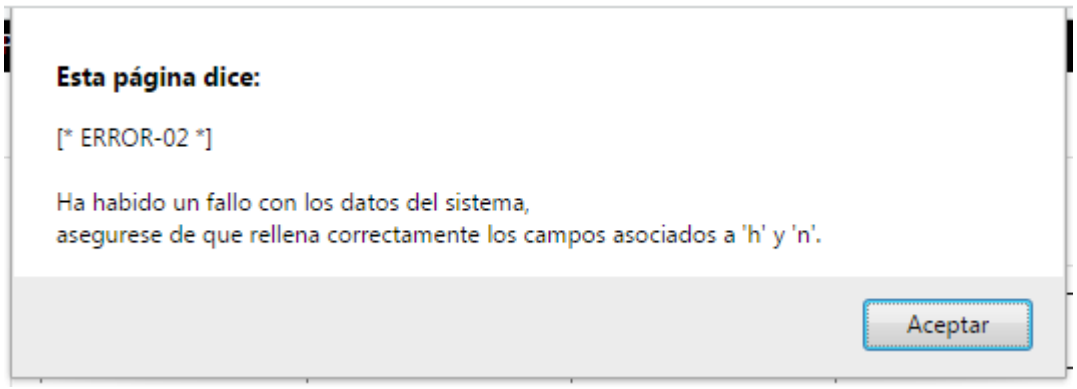

Figura 5.4.1.2. Mensaje de Error 02

Y debido a este error, aparte de mostrar dicho mensaje, se recargará la página entera.

Al principio, se implementó un tercer control de error en la página web, el cual si dabas al botón iniciar sin ningún objeto introducido aparecería el mensaje de error y se volvería a cargar la página, pero esto se cambió, y se optó por hacer invisible el botón iniciar en el momento inicial, y una vez añadido, al menos, un objeto se volvería visible para ejecutar el programa.

5.4.2. Three.js

Una vez dentro del entorno gráfico, debido al método del punto fijo que se ejecuta a la hora de resolver los problemas implícitos, puede dar lugar a que el punto fijo no converja, es decir, dado un estado inicial no consiga encontrar solución al problema, cada iteración estaría más lejos de la solución, por tanto, se ha hecho un control para cuando esto ocurra, la funcionalidad que tiene es que reinicia la página si se da este caso, y muestra un mensaje de error indicando el problema, como se ve en la figura siguiente:

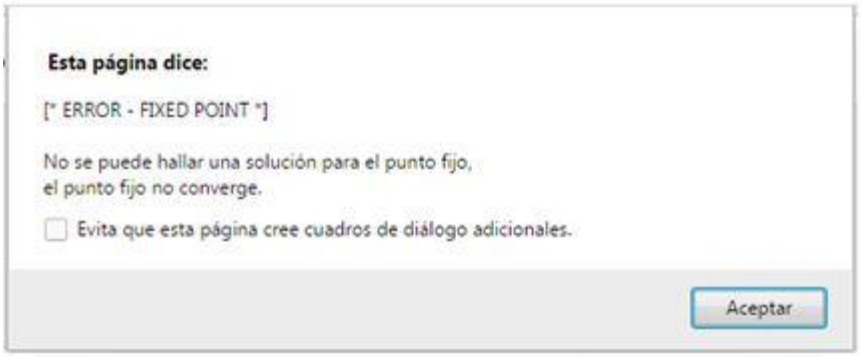

Figura 5.4.2.1. Mensaje de Error – Método del Punto Fijo

Para el error que se visualiza en la gráfica de error, el error cometido por el método que se esté ejecutando, se calcula de la siguiente forma, a medida que se va ejecutando el método, simultáneamente está el mismo método seleccionado ejecutándose continuamente en paralelo pero con una tamaño de paso de la mitad de tamaño que el que se ha introducido, por lo que consideraríamos como solución del método está ultima ejecución, este método con la mitad del tamaño de paso nos estaría devolviendo el doble de estados que para la ejecución normal, por lo tanto, a la hora de calcular el error cometido lo que se hace es comparar un resultado obtenido del método normal con uno de cada dos resultados obtenidos del método con tamaño de paso la mitad, es decir, si se está ejecutando el método normal con un tamaño de paso h = 0.1, la otra ejecución del método lo estaría haciendo con un tamaño de paso h = 0.05, por lo tanto, para poder diferenciar los valores correspondientes tendríamos que comparar los valores obtenidos del método normal con los valores obtenidos del método con la mitad del tamaño de paso pero para los que coincidan con la misma *h,* si el método normal se ejecuta en 0.1 en el siguiente paso se ejecutará en 0.2, mientras que el otro método que se ejecuta en paralelo a este, ejecutaría con la mitad de esa *h,* por lo que ejecutaría en 0.05, 0.1, 0.15, 0.2 y así sucesivamente, entonces lo que tenemos es que este último método hace el doble de pasos, por lo tanto, si queremos hacer la diferencia entre los dos métodos para considerarla como el error que se comete, tendremos que diferenciar para uno de cada dos pasos ejecutados en el método de *h* inferior, en este caso se haría la diferencia con los tamaños de paso h=0.1 y h=0.2.

Ese error se representará en la gráfica como el logaritmo del error que hemos obtenido de la diferencia, por lo que en el primer paso *h,* tendríamos una gráfica como esta:

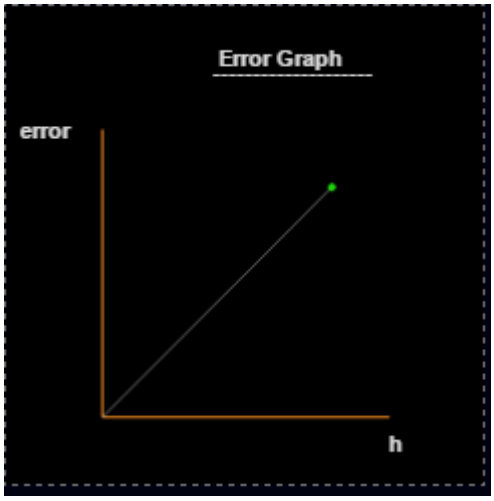

Figura 5.4.2.2. Grafica del Error, ejemplo de paso, imagen 1

Y a medida que se fueran completando los pasos obtendríamos nuevos valores que se irían representando en la gráfica como puntos, esto se debe a que la gráfica, es una gráfica que va haciendo escala, es decir, por cada diferencia que se realiza, la cual determina el error cometido para el paso actual, el resultado que se obtiene se representa en la gráfica mediante puntos escalados, en cada diferencia obtenida en los pasos correspondientes se hace la escala de la gráfica entonces se ubican los puntos en esa misma grafica pero para el nuevo paso de *h* y la diferencia obtenida, reajustando los introducidos anteriormente.

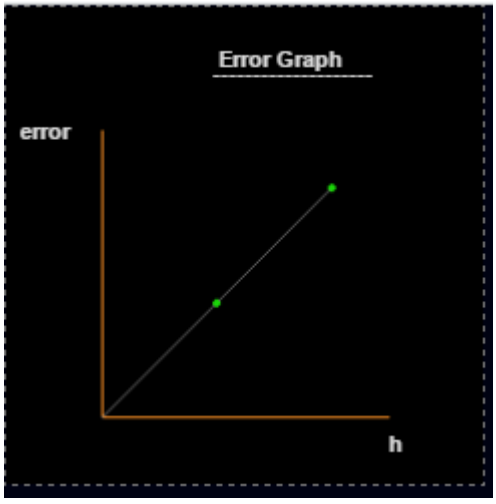

Figura 5.4.2.3 Grafica del Error, ejemplo de paso, imagen 2

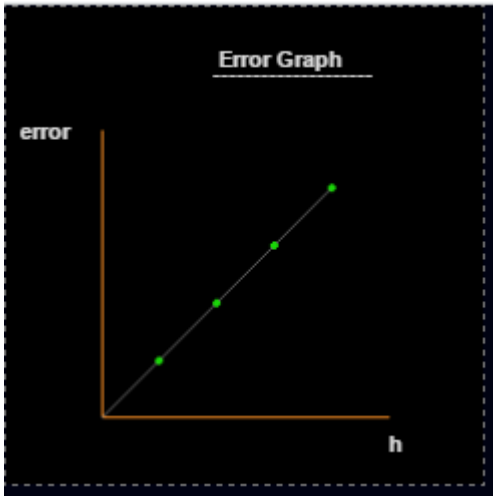

Figura 5.4.2.4 Grafica del Error, ejemplo de paso, imagen 3

Finalmente, si se están ejecutando muchos pasos o lo estamos ejecutando de forma infinita, a partir de un número determinado de puntos en la gráfica pararíamos de introducir más puntos que representaran los errores, debido a que ya tendríamos una muestra observable del error que comete el método, y los puntos estarían muy cerca unos de otros.

6

Conclusiones

6.1. Conclusiones

Una vez acabado el proyecto y echando la vista atrás, podemos concluir en que el proyecto cumple lo que se definía en el alcance.

Se ha generado una aplicación ejecutable en un navegador web compatible con WebGL, capaz de resolver los sistemas de ecuaciones diferenciales que se proponían en un principio, tanto los explícitos como los implícitos, y se muestra en un entorno gráfico, como lo es Three.js, el progreso que siguen estos métodos para el problema de los n cuerpos, también se ha hecho una pequeña grafica donde se muestra el error provocado por los métodos, para tener una pequeña muestra del error.

En lo que respecta a la realización del mismo, en un principio se estimaron menos horas de las que finalmente se han dedicado, debido a que este proyecto engloba varios temas complejos, por lo menos para mi persona, lo que dificultaba la implementación de los mismos, lo cual provocó algún cambio de dirección a la largo del proyecto que supuso tiempo extra de dedicación, debido a alguna confusión de algunos de los conceptos.

6.2. Posibles Mejoras

- Integrar en la aplicación el método de Runge-Kutta que ajusta el tamaño del paso para mantener el error acotado en todo momento, y así no incrementar el error que se produce por el tamaño del paso, el error se mantendría estable y el tamaño de paso sería variable, en cada paso se calcularía el tamaño de paso más adecuado.
- Ampliar la interfaz con más funcionalidades o métodos de resolución diferentes
- También se podría hacer una lista de texturas aplicables a los cuerpos que se van introduciendo de manera aleatoria, de modo que hubiese una variedad de cuerpos con texturas diferentes.
- La posibilidad de eliminar los objetos introducidos mediante la interfaz web, es decir, si introducimos un objeto por error con algún dato incorrecto, tener la opción de eliminarlo al momento, sin tener que volver a cargar la página.
- Añadir más ubicaciones diferentes de la cámara para ver la escena desde otras situaciones, por ejemplo, se podría colocar encima de uno de los objetos y mirando hacia el origen o hacia un eje fijo u otro objeto o similar.
- Volcar los datos que se van obteniendo debido a la resolución del método en un fichero externo, para tener la opción después de manejar esos datos con otras aplicaciones y demas.
- Proporcionar al usuario la posibilidad de ejecutar otro tipo de problemas o que defina el usuario el problema y pueda decir que variables hay que tener cuenta para dibujar los objetos.

Glosario

WebGL: API para renderización de gráficos 3D en navegadores web

Plug-in: Complemento que aporta una función nueva específica a una aplicación.

Freeware: El termino anglosajón para denominar al software libre.

Renderización: Proceso de generación de una imagen o animación a partir de un modelo 3D.

CanvasRenderer: Renderizador que no utiliza WebGL, dibuja utilizando la API Canvas 2D Context con una lentitud mayor que WebGL.

SVGRenderer: Renderizador de gráficos vectoriales bidimensionales, basado en XML.

WebGLRenderer: Renderizador que utiliza WebGL para renderizar.

Shaders: Tecnología para la realización de sombreado.

URL: Referencia a un recurso web que especifica su ubicación en la red.

Wireframe: Método de renderización que solo muestra las aristas de las mallas.

HTML: Lenguaje de etiquetas o marcas, utilizado para la creación de páginas web.

CSS: Lenguaje de diseño, que se utiliza en páginas HTML, para personalizar los elementos y crear el diseño visual de la página.

Bootstrap: es un *framework* que contiene diseños para páginas web.

Framework: Se denomina a un entorno de trabajo que proporciona diferentes herramientas para el desarrollo de software.

jQuery: Es una biblioteca de JavaScript para dotar a las páginas web de animación y demás, para hacerlas más dinámicas.

JavaScript: Lenguaje de programación orientado a objetos y que se ejecuta normalmente en la parte del cliente, proporciona dinamismo.

Alert: Mensaje enviado mediante JavaScript al cliente, suele utilizarse como advertencia.

Scroll: Referido a la acción de desplazar la barra de desplazamiento de la ventana.

Radiobutton: Elemento HTML, botones de una única selección dentro de un grupo de opciones.

LookAt: Referencia a la función de Three.js, la cual, mediante un vector dirección indica hacia donde está mirando la cámara.

Bibliografía

[1] 5.4. MÉTODO DE PUNTO FIJO
http://portales.puj.edu.co/objetosdeaprendizaje/Online/OA10/capitulo5/5.4.htm

[2] Fixed-point iteration - Wikipedia
https://en.wikipedia.org/wiki/Fixed-point_iteration

[3] Métodos de Runge-Kutta - Solución numérica de ecuaciones diferenciales - Mathstools
http://www.mathstools.com/section/main/Metodos_de_Runge_Kutta?lang=es

[4] Runge Kutta Calculator - Metodos Runge Kutta online
http://www.mathstools.com/dev.php/section/main/runge_kutta_calculator

[5] Euler method - Wikipedia
https://en.wikipedia.org/wiki/Euler_method

[6] Runge-Kutta methods - Wikipedia
https://en.wikipedia.org/wiki/Runge–Kutta_methods

[7] Runge-Kutta methods - Scholarpedia
http://www.scholarpedia.org/article/Runge-Kutta_methods

[8] LECCIÓN 3: MÉTODOS NUMÉRICOS PARA ECUACIONES DIFERENCIALES ORDINARIAS
http://www.matematicaaplicada2.es/data/pdf/1322432650_806622928.pdf

[9] Reducing and monitoring round-off error propagation for symplectic implicit Runge-Kutta schemes
https://link.springer.com/article/10.1007/s11075-017-0287-z?wt_mc=Internal.Event.1.SEM.ArticleAuthorOnlineFirst

[10] three.js - Javascript 3D library

https://threejs.org

[11] three.js docs

https://threejs.org/docs/

[12] Gráficos en 3D básicos con Three.js

https://msdn.microsoft.com/es-es/library/dn479430(v=vs.85).aspx

[13] Two-body problem - Wikipedia

https://en.wikipedia.org/wiki/Two-body_problem

[14] Three-body problem - Wikipedia

https://en.wikipedia.org/wiki/Three-body_problem

[15] n-body problem - Wikipedia

https://en.wikipedia.org/wiki/N-body_problem

[16] jQuery API Documentation

https://api.jquery.com

Anexo A: Enlaces Herramientas

A.1. Enlaces Herramientas

En esta sección se pueden ver los enlaces a las herramientas que se han utilizado para desarrollar el proyecto.

StopWatch:

http://win7gadgets.com/clock/blue-stopwatch.html

Atom:

https://atom.io

Notepad++:

https://notepad-plus-plus.org/download/v7.4.2.html

Microsoft Word 2010:

https://www.office.com

Google Chrome:

https://www.google.es/chrome/browser/desktop/index.html

Opera:

http://www.opera.com/es

Dropbox:

https://www.dropbox.com

Google Drive:

https://www.google.es/drive/apps.html