Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Eletrotécnica
Mestrado em Eng.ª Eletrotécnica e de Computadores

SIMULAÇÃO DE FENÓMENOS ELETROMAGNÉTICOS E MECÂNICOS EM SISTEMAS DE PARTÍCULAS

RAÚL RODRIGUES FIGUEIRINHA

Leiria, Julho de 2025

Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Eletrotécnica
Mestrado em Eng.ª Eletrotécnica e de Computadores

SIMULAÇÃO DE FENÓMENOS ELETROMAGNÉTICOS E MECÂNICOS EM SISTEMAS DE PARTÍCULAS

Relatório final da Dissertação de Mestrado em Engenharia Eletrotécnica e de Computadores, ramo de Energia e Automação

RAÚL RODRIGUES FIGUEIRINHA
Número: 2222903

Trabalho realizado sob orientação
do Professor Doutor Alberto Negrão (alberto.negrao@ipleiria.pt) e
do Professor Doutor Telmo Fernandes (telmo.fernandes@ipleiria.pt).

Leiria, Julho de 2025

AGRADECIMENTOS

A jornada de desenvolvimento deste trabalho foi longa e desafiante, mas acima de tudo, um prazer. Para além de ser um projeto académico, é também pessoal. Quero aproveitar este pequeno parágrafo para agradecer a Deus, por me dar tudo, aos meus pais, por me darem uma educação privilegiada que me permitiu chegar aqui e também a toda a minha restante família e amigos próximos que sempre me motivaram. Agradeço também aos meus orientadores, Alberto Negrão e Telmo Fernandes por me darem a oportunidade de desenvolver este projeto em meio académico e me guiarem ao longo do seu desenvolvimento.

Gostaria de expressar o meu agradecimento ao Instituto de Telecomunicações, pelas excelentes condições de investigação e recursos computacionais, bem como à Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria pelos seus recursos, que foram cruciais para o sucesso desta investigação.

RESUMO

O uso extensivo da matemática na descrição rigorosa de fenómenos naturais tem sido praticado desde tempos imemoriais. Contudo, o domínio do formalismo matemático nem sempre resulta numa compreensão intuitiva dos fenómenos físicos subjacentes. Para superar essa dificuldade, métodos visuais e descritivos revelam-se ferramentas valiosas no processo de aprendizagem.

A presente dissertação descreve a jornada para implementar um simulador gráfico interativo que auxilie na aprendizagem do comportamento das forças fundamentais do universo, em particular das forças gravítica e eletromagnética, passando inclusive por uma pequena experiência com forças intermoleculares. Consiste portanto, num programa informático, desenvolvido para ser facilmente executado em computadores pessoais e permitir ao utilizador adquirir uma noção mais clara dos fenómenos em estudo.

Entre outras funcionalidades, esta peça de *software* permite a construção dos mais variados cenários, de onde se pode obter o desenho dos campos de forças gerados, interação de partículas móveis sujeitas aos campos em estudo, tutoriais incorporados, assim como a disponibilização de uma biblioteca com objetos configuráveis, passíveis de ser usados como componentes nos cenários físicos criados pelo utilizador.

ABSTRACT

The extensive use of mathematics for the rigorous description of natural phenomena has been employed since immemorial times. However, mastery of mathematical formalism does not always lead to an intuitive understanding of the underlying physical phenomena. To overcome this difficulty, visual and descriptive methods prove to be valuable tools in the learning process.

This thesis outlines the journey to implement an interactive graphical simulator designed to aid the learning of the behaviour of the fundamental forces of the universe, particularly gravitational and electromagnetic forces, including a brief exploration of intermolecular forces. It is, therefore, a computer program developed to be easily run on personal computers, enabling users to gain a clearer understanding of the phenomena under study.

Among other features, this software allows the creation of a wide variety of scenarios, from which force field diagrams can be generated, interactions of mobile particles subject to the fields under study can be observed, built-in tutorials accessed, and a library of customisable objects utilised. Such objects can be employed as components in the physical scenarios created by the user.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CPU	Central Processing Unit (Unidade de Processamento Central).
EDP	Equação Diferencial Parcial.
FDTD	Finite-Difference Time-Domain Method.
FOSS	Free and Open-Source Software.
GLSL	OpenGL Shading Language.
GPU	Graphics Processing Unit (Unidade de Processamento Gráfico).
PIC/FLIP	Particle-In-Cell/Fluid-Implicit Particles.
VRAM	Video random access memory.

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Os vários fenómenos observados no dia a dia, como por exemplo, mudanças de temperatura, de estado físico e até a própria luz, podem ser descritos com as leis físicas mais fundamentais de que dispomos.

De um conjunto de leis simples, ou talvez de uma só, emergem vários fenómenos intrínsecos à realidade, incluindo a própria vida. Numa escala mais alargada, para muitos fenómenos, a mecânica celeste pode ser simulada utilizando a lei da gravitação universal. Por outro lado, numa escala bem menor, a corrente elétrica, objetos magnéticos e a luz podem ser simulados utilizando as leis de Maxwell.

A aprendizagem destes conceitos adquire-se por meio da educação, que se caracteriza por ser um dos principais pilares de uma sociedade. Por esta razão, é importante torná-la o mais eficaz possível. Tendo isto presente, este projeto tem como principal motivação criar um produto informático que clarifique intuitivamente conceitos e fenómenos que por vezes se apresentam complexos no que diz respeito às forças que regem o universo, nomeadamente das forças eletromagnética e gravítica.

1.2 OBJETIVOS

Atualmente é possível explicar e prever vários fenómenos físicos baseando-se em modelos de partículas [1]. Os modelos de partículas reduzem os constituintes da matéria a pontos no espaço com características bem definidas que, em conjunto com um modelo simplificado definido por um conjunto de leis, cria métodos que permitem prever comportamentos reais da natureza. Desde galáxias, planetas, corpos rígidos, moléculas até chegar aos átomos, estes objetos podem ser reduzidos a um ponto no espaço e, utilizando um conjunto realista de características próprias e leis que as governem, é possível prever e simular o seu comportamento.

Assim sendo, a simulação de sistemas de partículas é um recurso valioso no que toca a áreas da engenharia e das ciências, como a física, a química e a biologia, permitindo analisar fenómenos complexos de forma detalhada e previsível.

Dando ênfase ao facto de a computação gráfica ser um recurso poderoso, altamente acessível nos dias de hoje, o objetivo deste projeto é desenvolver um *software*, intitulado ***Force Fields***, com capacidade de simular em tempo real, sistemas de partículas com características e leis baseadas na realidade, permitindo ao utilizador criar cenários físicos, com o propósito de serem simulados neste modelo simplificado da realidade. A presente dissertação visa documentar o trabalho

desenvolvido, desde o processo de pesquisa, até à interação com o programa e dele extrair resultados relevantes.

Esta ferramenta tem a capacidade de simular sistemas gravitacionais, eletromagnéticos e intermoleculares, utilizando leis que definem campos de forças no espaço tridimensional. Estes campos de forças são gerados pelas próprias partículas presentes no ambiente de simulação, mas, ao mesmo tempo, estas partículas têm o seu movimento condicionado pelo próprio campo, resultando numa interação contínua e dinâmica entre ambos. Este processo cria comportamentos complexos, que não foram pré-programados ou até previstos, sendo que de cada simulação podem resultar fenómenos e comportamentos que o próprio programador desconhece.

O desenvolvimento deste *software* pretende explorar uma área específica deste tipo de tecnologia, que tenha como prioridades a facilidade de uso, a interatividade e uma abundância de recursos disponíveis que permitam a criação dos mais variados cenários de simulação, dando prioridade à liberdade do utilizador.

Com ferramentas de análise e estatística incluídas no programa e uma alta capacidade de criação de cenários, o produto final deste trabalho permite simular comportamentos e fenómenos naturais e daí auxiliar potencialmente em futuros trabalhos científicos ligados à gravitação, ao eletromagnetismo ou até a forças intermoleculares.

1.3 ESTRUTURA DO DOCUMENTO

Este documento está estruturado da seguinte forma:

- O presente capítulo, a introdução (1), expõe os objetivos deste trabalho e o que dele se espera;
- A base teórica, estado da arte e revisão da literatura (2), apresenta todo o estudo realizado para desenvolver o *Force Fields*, especialmente no que toca à definição teórica dos campos de força, estudo geral de como são tipicamente desenvolvidos simuladores de partículas e ainda, de uma forma mais técnica, como implementar diversos integradores para calcular a evolução de um sistema de forma iterativa;
- Desenvolvimento do *Force Fields* (3) explora o funcionamento interno do simulador e as várias técnicas utilizadas na sua produção. Isto incluí a apresentação do *Godot*, um *software* de desenvolvimento de jogos e aplicações gráficas, que serviu de base à criação do projeto;
- Funcionalidades e capacidades do *Force Fields* (4) deixa de parte o trabalho de implementação do *software* e foca-se totalmente na explicação do funcionamento do *Force Fields*;
- Cenários de simulação, seus resultados e discussão (5) é um capítulo que coloca o *Force Fields* à prova e mostra diversos resultados e simulações utilizados para o testar;
- Por último, a conclusão (6) encerra a dissertação e indica possíveis trabalhos futuros para dar continuidade a este projeto.

2

BASE TEÓRICA, ESTADO DA ARTE E REVISÃO DA LITERATURA

2.1 BASE TEÓRICA - CAMPOS DE FORÇA SIMULADOS

No sentido de abordar uma das bases teóricas inerentes ao projeto, este subcapítulo visa clarificar o que são e como funcionam os campos de duas das quatro forças fundamentais do universo. As duas forças fundamentais abordadas, a força gravítica e a força eletromagnética, são implementadas num contexto onde a interação entre partículas dá-se instantaneamente sem a propagação de ondas no espaço. Devido à natureza quântica das outras duas forças fundamentais, as mesmas não são utilizadas neste projeto, no entanto, é implementada uma outra força, não fundamental, derivada do potencial de Lennard-Jones, no sentido de enriquecer os estudos feitos nesta dissertação.

2.1.1 *Força gravítica*

A gravidade é uma das forças fundamentais e descreve a tendência natural que a matéria tem de se aglomerar [2]. Considere-se a equação da atração universal (1), que descreve a força gravítica que um corpo 2 exerce num corpo 1,

$$\vec{F}_{12} = G\frac{m_1 m_2}{r^2}\hat{r}_{12}, \tag{1}$$

onde G é a constante gravitacional com o valor $6{,}674 \times 10^{-11}\,m^3\,kg^{-1}\,s^{-1}$, m_1 e m_2 correspondem às massas dos dois corpos (redutíveis a um ponto) dos quais se pretende obter a força de atração, r corresponde à distância entre os mesmos e $\hat{r}_{12}$ corresponde ao versor com origem no corpo *1* que aponta para o corpo 2.

A intensidade desta força decai com o quadrado da distância e é proporcional à massa de ambos os corpos ou partículas. Uma vez que a aceleração dos corpos é dada por $a = \frac{F}{m}$, a massa do próprio corpo não influencia a sua aceleração gravítica. Isto quer dizer que é irrelevante se um dado corpo tem uma massa extremamente elevada ou reduzida, a sua aceleração depende apenas da matéria envolvente.

A níveis cósmicos, a força gravítica prevalece sobre a força eletromagnética pelo facto de grandes corpos serem geralmente constituídos por números similares de cargas positivas e negativas, apresentando de forma global uma carga nula.

2.1.2 *Força eletromagnética*

A força eletromagnética define como interagem as partículas com carga elétrica não nula. Pode ser dividida entre força eletrostática e força magnética.

Para aumentar a flexibilidade em termos matemáticos e também simplificar cenários que de outra forma seriam mais complexos, ao invés de se definir uma força entre partículas, é comum definir-se o campo vetorial gerado por uma dada partícula no espaço, e de seguida, definir-se uma força que descreva como uma partícula interage com esse campo. Por esta razão serão definidos, em primeiro lugar, os campos elétrico e magnético e posteriormente a força que estes campos exercem numa partícula.

A lei de Coulomb define a força elétrica entre duas partículas tal como se pode observar na equação 2,

$$\vec{F}_{12} = \frac{1}{4\pi\epsilon}\frac{q_1 q_2}{r^2}\hat{r}_{12}, \tag{2}$$

onde $\vec{F}_{12}$ é a força que a partícula 2 gera na partícula 1, ϵ a permitividade elétrica do meio, sendo o seu valor no vácuo $8{,}854 \times 10^{-12}\,C^2\,N^{-1}\,m^{-2}$, $q1$ e q_2 a carga elétrica das partículas em interação, $\hat{r}_{12}$ o versor com origem na partícula 1 que aponta para a partícula 2 e r a distância entre as partículas. Adicionando a definição de campo elétrico presente na equação 3,

$$\vec{E}_1 = \frac{\vec{F}_{12}}{q_2}, \tag{3}$$

onde $\vec{E}_1$ é o campo elétrico gerado pela carga q_1 no ponto onde se encontra a carga de prova q_2. Destas duas fórmulas, é possível derivar o campo elétrico que uma partícula gera num ponto do espaço, expresso na equação 4,

$$\vec{E} = \frac{1}{4\pi\epsilon}\frac{q}{r^2}\hat{r}, \tag{4}$$

sendo q a carga elétrica da partícula geradora de campo elétrico, $\hat{r}$ o versor com origem na partícula e que aponta para o ponto onde se calcula o campo e r a distância entre a partícula e esse mesmo ponto. Uma vez que a carga pode ser positiva ou negativa (ou nula), este campo pode gerar forças tanto de atração, como de repulsão.

A força magnética, por outro lado, pressupõe a existência de cargas em movimento, uma vez que é uma força apenas evidente em partículas com velocidade não nula (relativamente à fonte de campo magnético). Para velocidades muito inferiores à da luz, no vazio e numa perspetiva clássica, é possível descrever o campo vetorial magnético gerado por uma partícula num ponto do espaço a partir da lei de Biot-Savart descrita na equação 5,

$$\vec{B} = \frac{\mu}{4\pi}\cdot\frac{q\left(\vec{v}\times\hat{r}\right)}{r^2} \tag{5}$$

onde μ é a permeabilidade magnética do meio, sendo o seu valor no vácuo $4\pi \times 10^{-7}\,H\,m^{-1}$, q a carga elétrica da partícula, $\vec{v}$ a velocidade da partícula, r a

distância entre a partícula e o ponto onde se está a calcular o campo magnético e $\hat{r}$ o versor com origem na partícula e que aponta para o ponto.

$\vec{E}$ e $\vec{B}$ são campos vetoriais, funções de ponto e do tempo, não sendo possível observá-los diretamente sem que interajam com algo. Sendo assim, é necessária uma lei que descreva a maneira como estes campos alteram o comportamento de uma dada partícula. Esta lei é dada pela força de Lorentz, descrita pela equação 6, que descreve a força que uma carga sofre na presença de um campo elétrico e magnético:

$$\vec{F} = q\vec{E} + q\left(\vec{v} \times \vec{B}\right) \tag{6}$$

sendo q a carga da partícula e $\vec{v}$ a velocidade da partícula que sofre a força. Esta fórmula define a força fundamental eletromagnética.

Com as leis de Coulomb, de Biot-Savart e de Lorentz já é possível criar um *software* que simule um sistema de partículas que interagem entre si através da força eletromagnética. Uma formulação matemática alternativa às leis de Coulomb e de Biot-Savart para definir o campo elétrico e magnético é dada pelas equações de Maxwell, as quais são elencadas, para o vazio, na tabela 1.

Tabela 1: Equações de Maxwell para o vazio na forma integral.

Lei de Gauss	$\int_S \vec{E} \cdot \hat{n}\, dS = \frac{Q}{\varepsilon_0}$
Lei de Gauss (magnetismo)	$\int_S \vec{B} \cdot \hat{n}\, dS = 0$
Lei de Faraday-Lenz	$\oint \vec{E} \cdot d\vec{r} = -\frac{d}{dt}\int_S \vec{B} \cdot \hat{n}\, dS$
Lei de Ampère-Maxwell	$\oint \vec{B} \cdot d\vec{r} = \mu_0 I + \mu_0 \varepsilon_0 \frac{d}{dt}\int_S \vec{E} \cdot \hat{n}\, dS$

Estas expressões matemáticas são equações diferenciais às derivadas parciais (EDP), tornando a sua resolução mais complexa que as equações diferenciais ordinárias. No entanto, é possível desenvolver as equações de Maxwell para campos discretos ao invés de campos contínuos, procedimento necessário para simular as equações num computador utilizando métodos numéricos [3].

As equações apresentadas na tabela 1, estão na sua forma integral, pelo que descrevem as características eletromagnéticas envolvendo volumes, planos ou linhas e não em relação a um ponto no espaço, como quando expressas na sua forma diferencial, apresentadas no sistema de equações 7:

$$\begin{cases} \vec{\nabla} \cdot \vec{E} = \dfrac{\rho}{\varepsilon_0}, \\ \vec{\nabla} \cdot \vec{B} = 0, \\ \vec{\nabla} \times \vec{E} = -\dfrac{\partial \vec{B}}{\partial t}, \\ \vec{\nabla} \times \vec{B} = \mu_0 \vec{J} + \mu_0 \varepsilon_0 \dfrac{\partial \vec{E}}{\partial t}. \end{cases} \tag{7}$$

Uma vez que o âmbito deste estudo incide em partículas, ou seja, pontos no espaço, a forma diferencial das equações de Maxwell é, neste caso, mais útil. Por essa razão será dada uma breve noção do significado físico destas equações na sua forma diferencial.

A Lei de Gauss mostra que a divergência do campo elétrico num dado ponto do espaço é proporcional à densidade de carga elétrica nesse ponto. Ou seja, a carga elétrica num dado ponto do espaço define o quão divergentes ou convergentes são as linhas de campo elétrico nesse ponto. Isto significa que para todos os pontos do espaço, a divergência do campo elétrico é nula, exceto para pontos onde existe uma fonte ou um sumidouro de campo elétrico, indicando assim a existência de uma carga positiva ou negativa nesse ponto como, por exemplo, um protão ou um eletrão.

A Lei de Gauss para o magnetismo mostra que a divergência do campo magnético num dado ponto do espaço é sempre nula. Tal facto implica que não existem pontos no espaço que sejam fontes ou sumidouros de campo magnético. Por esta razão, as linhas de campo magnético são fechadas, pelo que não existem monopolos magnéticos.

A Lei de Faraday-Lenz mostra que um campo magnético, variável no tempo, irá gerar um campo elétrico, cujo rotacional tem um sentido oposto ao do da variação do campo magnético. Este fenómeno pode ser observado quando um íman atravessa uma espira condutora fechada e, em consequência, é gerada uma corrente elétrica na própria espira. Este é o princípio da indução eletromagnética.

A Lei de Ampère-Maxwell mostra que um campo magnético pode ser formado tanto pela existência de uma densidade de corrente elétrica, como por uma densidade de corrente de deslocamento, sendo esta última dada pela variação do campo elétrico num ponto. Isto quer dizer que, por exemplo, qualquer condutor pelo qual circule uma corrente elétrica, gera um campo magnético girante ao longo do seu comprimento. Outro exemplo possível consiste no campo magnético gerado entre as placas de um condensador que esteja a ser carregado ou descarregado, mesmo não existindo qualquer corrente no dielétrico.

As equações de Maxwell não substituem a equação de Lorentz uma vez que apenas descrevem como se comportam o campo elétrico e magnético, mas não como partículas interagem com estes campos.

2.1.3 *Força intermolecular derivada do potencial de Lennard-Jones*

As forças gravítica e eletromagnética já fornecem motivação suficiente para a aprendizagem de um ambiente de desenvolvimento gráfico, criação de uma infra-estrutura para correr um simulador, criação de interfaces gráficas e implementação de cenários de simulação configuráveis. No entanto, no sentido de enriquecer o projeto e observar fenómenos eletromagnéticos a uma escala molecular, foi adicionado ao simulador, como já mencionado anteriormente, a força intermolecular derivada do potencial de Lennard-Jones.

Esta força tem carácter atrativo e repulsivo baseado na distância entre as partículas, pelo que existe uma distância de repouso onde o potencial é mínimo e a força é nula. Este é um comportamento que não se observa em sistemas gravitacionais, uma vez que nestes sistemas, não é possível ter aglomerações de partículas imóveis e estáveis. No entanto, sabemos da experiência do dia a dia, que os objetos normalmente não colapsam num ponto nem se desintegram por expansão, ou seja, a maior parte da matéria mantém a sua integridade e coesão graças a forças atrativas e repulsivas onde cada partícula ocupa a sua posição de equilíbrio.

O potencial de Lennard-Jones é dado pela fórmula 8,

$$V(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right], \tag{8}$$

onde r representa a distância entre duas partículas, ϵ, medido em Joules, é designado como a profundidade do poço de potencial e define a intensidade do potencial gerado, e σ é a distância na qual o potencial é nulo. É imprescindível que tanto r como σ tenham a mesma unidade de comprimento.

Sabendo que a força de um dado potencial que só dependa da distância r, pode ser obtida através da derivada simétrica do mesmo, obtém-se a força expressa na equação 9,

$$F(r) = -\frac{dV}{dr} = 24\epsilon \left[\frac{2\sigma^{12}}{r^{13}} - \frac{\sigma^{6}}{r^{7}} \right], \tag{9}$$

sendo esta a força intermolecular derivada do potencial de Lennard-Jones.

Na figura 1 [4] pode observar-se o potencial de Lennard-Jones.

Apesar de não ser uma força fundamental, a sua origem é eletromagnética, onde a componente atrativa é atribuída principalmente às forças de Van der Waals, ao passo que a componente repulsiva deve-se à sobreposição das orbitais eletrónicas. Este modelo surge com a necessidade de simular de forma eficiente comportamentos que surgem de uma ou mais forças fundamentais a partir de arranjos específicos entre partículas. Apesar disso, a componente repulsiva do potencial é escolhida por conveniência computacional, não possuindo uma justificação teórica rigorosa. Consequentemente, o potencial de Lennard-Jones é uma aproximação que se mostra mais eficaz para a modelação de sistemas com interações relativamente simples. No entanto, este modelo não deixa de ser a ferramenta de referência utilizada para simular e analisar fenómenos macroscópicos como a im-

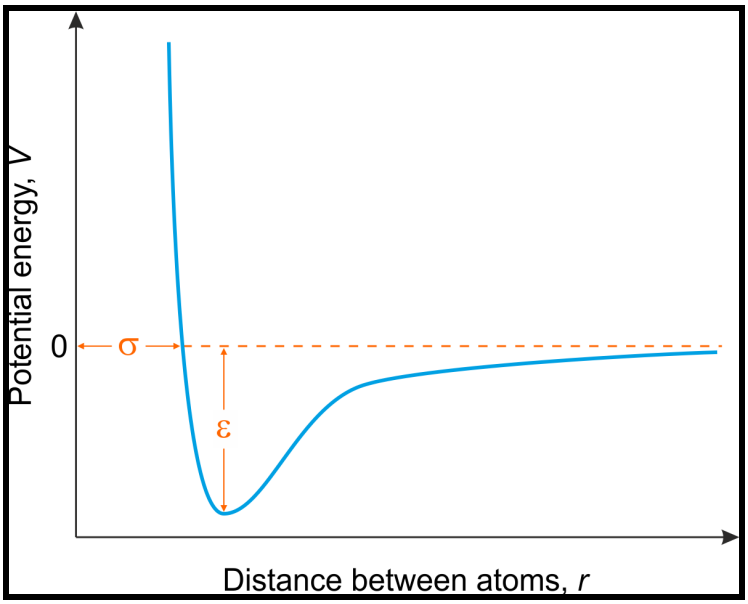

Figura 1: Potencial de Lennard-Jones consoante a distância entre partículas

pulsão (flutuabilidade), plasticidade/ductilidade de um material e até mudanças de estado físico com variações de pressão e temperatura [5].

2.2 SIMULADORES DE CAMPO E DE PARTÍCULAS

O presente subcapítulo procura expor e avaliar o estado da arte dos *softwares* de simulação de partículas mais acessíveis ao público, assim como as técnicas utilizadas para os criar. Com a reunião de todos os aspetos positivos destas ferramentas, fica estabelecida uma base sólida importante para o desenvolvimento deste projeto.

Os critérios levados em consideração para selecionar os simuladores a seguir apresentados, de entre todos os disponíveis, foram a acessibilidade dos mesmos. Simuladores pagos ou com necessidade de subscrição não foram tidos em conta. A seleção feita inclui apenas simuladores distintos entre si, ou seja, existem outros que não foram considerados devido à sua semelhança com os apresentados. Programas que sejam demasiado técnicos e com pouca funcionalidade gráfica foram também descartados uma vez que o objetivo é obter uma compreensão intuitiva e visual dos fenómenos simulados.

2.2.1 *Simuladores gravitacionais*

Com os nomes *Particle sandbox - Gravity Simulator* [6] e *Gravity by HermannBjorgvin* [7], estes simuladores de gravidade podem ser executados num navegador de *internet* comum. Cada partícula tem uma massa associada e altera o seu diâmetro conforme essa massa. Cada vez que uma colisão ocorre, é transferida massa da partícula menor para a maior de forma a simular a transferência de massa que ocorre, por exemplo, em colisões entre planetas. No entanto, para simular

fenómenos internos como a liquefação e destruição que ocorrem numa colisão planetária, será necessário simular planetas como sendo um aglomerado de partículas. Desta forma, as colisões podem gerar detritos, outros planetas, ou simplesmente unificar os dois corpos. O desenho das trajetórias dos corpos é uma característica muito importante para analisar um sistema gravitacional, pelo que é uma vantagem a ter em conta. As figuras 2 e 3 mostram o aspeto gráfico destas aplicações.

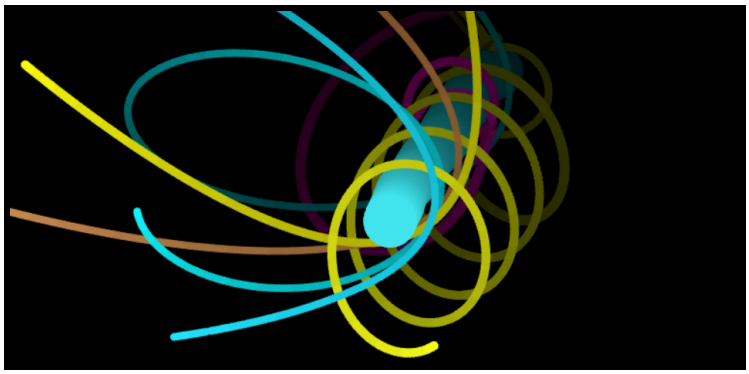

Figura 2: Aspeto gráfico do simulador *Particle Sandbox - Gravity Simulator*.

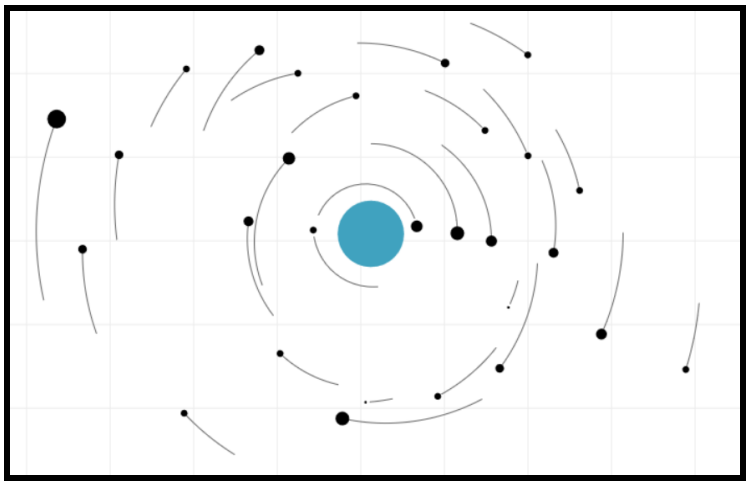

Figura 3: Aspeto gráfico do simulador *Gravity by HermannBjorgvin*.

O simulador *Lagrange Points* [8] é um exemplo de simuladores adequados à análise de aspetos específicos de uma força fundamental. Neste caso o objetivo é mostrar que existem 5 pontos específicos no espaço, chamados pontos de Lagrange, onde a combinação das forças gravitacionais exercidas por dois corpos de massa elevada e a força centrífuga no referencial em rotação permite que um terceiro corpo, de massa desprezável, permaneça em equilíbrio relativo. Nestes pontos, o terceiro corpo acompanha o movimento orbital dos dois corpos principais, mantendo o período orbital e a posição relativa no sistema. A figura 4 mostra os 5 pontos de *Lagrange* no sistema Terra-Lua.

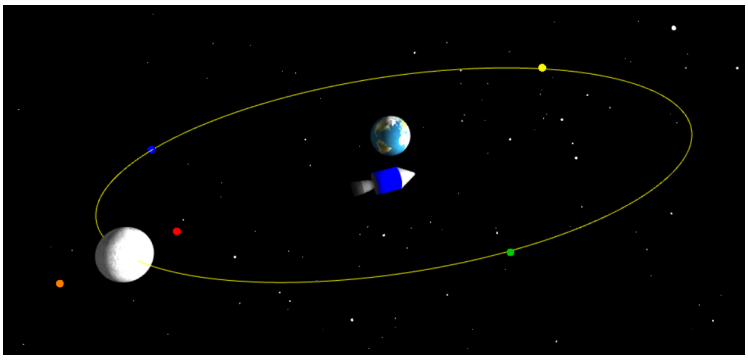

Figura 4: Pontos de Lagrange no sistema Terra-Lua - *Simulador Lagrange Points*.

2.2.2 *Partículas expostas a campos vetoriais*

O simulador com o nome *3-D Vector Fields* [9], simula o comportamento de partículas quando expostas a diferentes campos vetoriais. Nesta aplicação, os campos são gerados mediante equações que definem um potencial para quaisquer coordenadas espaciais. É ainda possível escolher diferentes campos e alterar vários parâmetros da simulação, sendo também possível observar os campos através das suas linhas de fluxo (figura 5) ou dos vetores de campo.

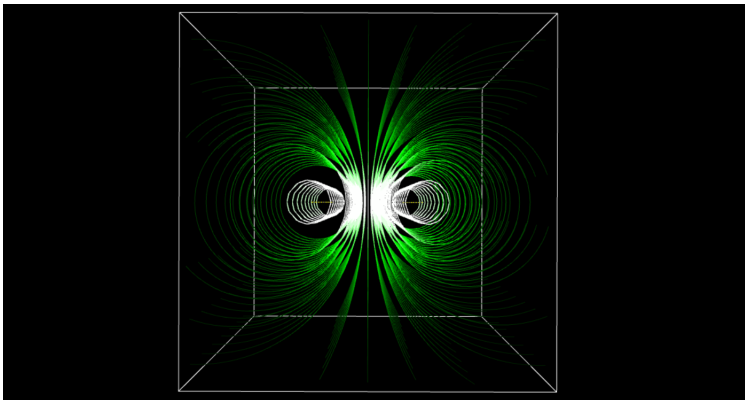

Figura 5: Linhas de fluxo num dipolo rotacional - *3-D Vector Fields*.

Existe ainda uma versão 2D [10] com funcionalidades semelhantes.

2.2.3 *Simulação gravítica segundo a teoria da relatividade geral usando a métrica de Schwarzschild*

Relativamente à teoria da relatividade geral, existe um simulador baseado na métrica de Schwarzschild para simular a gravidade dentro de certas condições (a carga elétrica do corpo é nula, o corpo não roda sobre si e não existe matéria escura). Esta métrica é a solução exata para as equações de campo de Einstein [11].

O programa chama-se *Schwarzschild Trajectories* [12] e permite ao utilizador interagir com o campo gravitacional adicionando partículas. Apesar de ser uma

simulação 2D, o criador aproveitou mais uma dimensão para representar a curva do espaço-tempo tal como pode ser observado na figura 6.

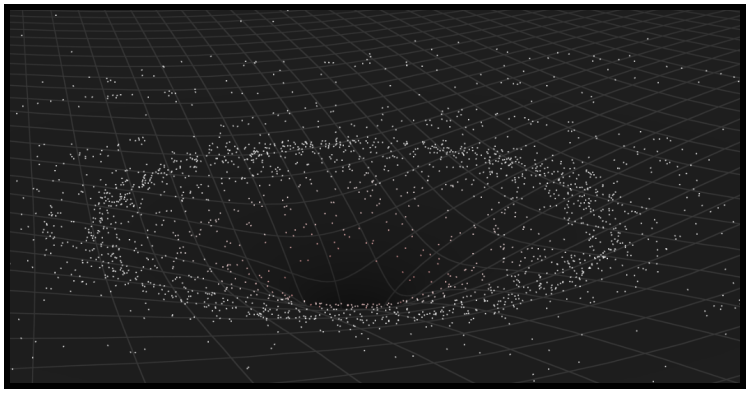

Figura 6: Simulação gravítica 2D com representação da curva do espaço-tempo.

Do mesmo criador, existe também um *software* com o nome *Continuum Gravity* que simula a gravidade para uma quantidade extremamente elevada de partículas com dimensões semelhantes, que pretende mostrar que um espaço com matéria uniformemente distribuída tende a criar galáxias de forma natural tal como mostrado na figura 7.

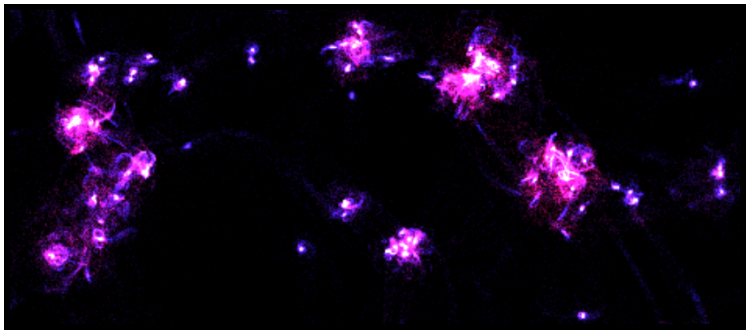

Figura 7: Simulação de um número elevado de partículas geradoras de campo gravitacional - Formação de galáxias.

2.2.4 *Simulação de um número elevado de partículas*

Existem certos simuladores que merecem ser estudados devido à forma como foram implementados do ponto de vista computacional. Estes simuladores têm a capacidade de simular quantidades elevadas de partículas em tempo real, tal como mostram os *websites Fluid Particles* [13] com capacidade para simular até 470 mil partículas e *The Spirit* [14] com capacidade de simular até 4 milhões de partículas.

A figura 8 mostra a simulação *Fluid Particles* que se baseia em *WebGL*, com simulação implementada na Graphics Processing Unit (Unidade de processamento gráfico) (GPU). Vale a pena mencionar que parte deste projeto foi desenvolvido para permitir a execução dos cálculos na GPU, uma vez que a mesma tem

um poder de processamento superior à Central Processing Unit (Unidade de processamento central) (CPU).

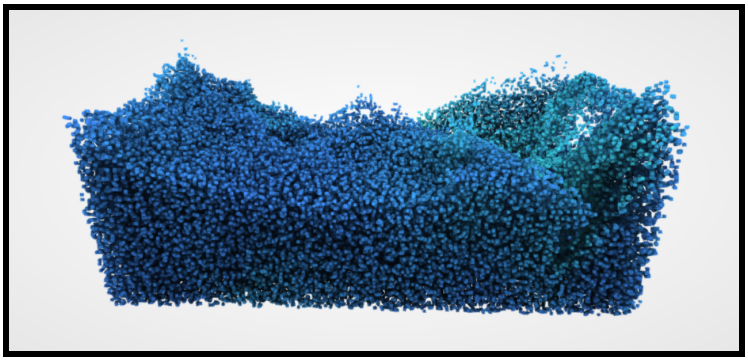

Figura 8: Simulação de um número elevado de partículas - Meio líquido.

O método utilizado nesta simulação é denominado Particle-In-Cell/Fluid-Implicit Particles (PIC/FLIP) e consiste em resolver equações parciais diferenciais por meio de duas perspetivas: como um sistema de partículas e como um volume dividido em pequenas células (discretização do espaço). Neste caso o sistema tem em conta a gravidade e usa partículas que interagem de forma a produzir matéria de alta densidade e incompressível, simulando assim o típico comportamento de líquidos.

Já a figura 9 mostra outra simulação que despreza a gravidade e utiliza partículas de baixa densidade, simulando então o comportamento típico de um gás. Esta aplicação foi também implementada em *WebGL* com a simulação processada na GPU, mas utiliza como métodos a derivação de ruído (*noise derivatives*) para o movimento aleatório das partículas, assim como também ruído de ondulação (*curl noise*) para formar vórtices. Estas técnicas não estão, neste caso, implementadas para simular forças fundamentais específicas. No entanto são excelentes formas de adicionar perturbações intencionais num sistema. Por exemplo, uma vez que a temperatura de um objeto é definida pela magnitude da vibração dos seus átomos e/ou moléculas, estas técnicas podem ser utilizadas para simular a temperatura em objetos.

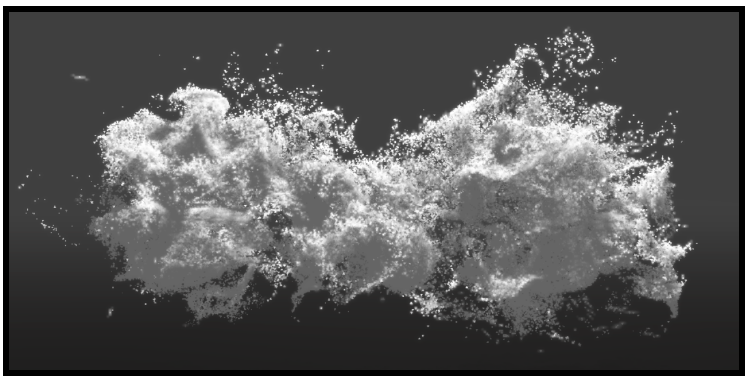

Figura 9: Simulação de um número elevado de partículas - Meio gasoso.

2.2.5 *Simulação da propagação de ondas eletromagnéticas*

Um campo elétrico variável no tempo gera um campo magnético, também variável no tempo, e vice-versa. Esta interação cíclica entre campos gera uma onda eletromagnética. Se a frequência desta onda estiver compreendida entre as frequências a que o olho humano é sensível, chamamos a estas ondas, luz. Simular a propagação destas ondas não é trivial, mas é possível, mesmo em tempo real. Um exemplo disto é o *software Ripple Tank Simulation* [15] que permite simular ondas eletromagnéticas ou sonoras e desta forma, mostrar vários fenómenos como a reflexão, refração ou padrões de interferências, mediante experiências como a fenda dupla, filtros passa-baixo e passa-alto, estrondo sónico para ondas sonoras (*sonic boom*), entre outros. Na figura 10 é possível observar a interferência ondulatória causada por duas fontes idênticas a uma certa distância.

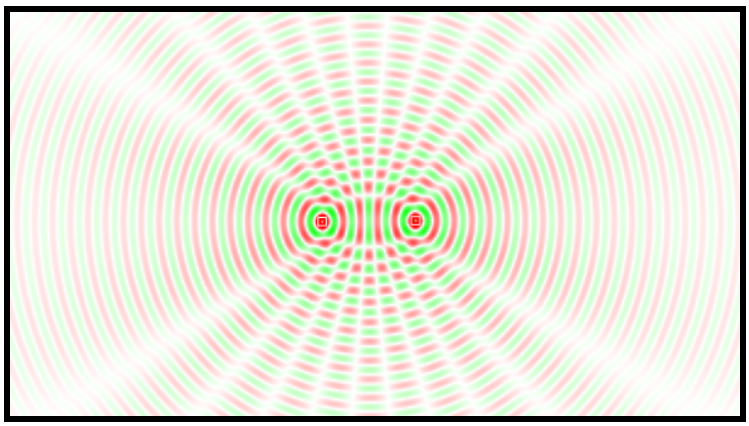

Figura 10: Interferência ondulatória causada por duas fontes idênticas - *Ripple Tank Simulation*.

Esta aplicação está programada em *Java* e apesar de não serem dados detalhes de como foi desenvolvida, provavelmente esta utilizou técnicas como o finite-difference time-domain method (FDTD), que é um método extremamente poderoso por ser desenvolvido para evoluir em tempo real no domínio do tempo, podendo lidar com várias frequências diferentes simultaneamente, fornecendo resultados confiáveis, mesmo em ambientes com materiais com índices de refração variável, temperatura variável e reflexão parcial variável.

2.2.6 *Considerações finais*

A lista de possibilidades para simular um sistema de partículas é muito abrangente, uma vez que existem inúmeros métodos dos quais se pode obter um resultado em tempo real. É importante encontrar um equilíbrio entre *performance* computacional e precisão dos resultados. Abaixo são elencadas as técnicas e métodos de simulação mais promissoras de entre as referidas neste capítulo:

- Simulação através da dinâmica de partículas com *N-body simulation*
- Método *Particle-in-cell (PIC)* para resolver equações parciais diferenciais discretizando o espaço;

- Método *Fluid-Implicit Particles (FLIP)* que calcula o movimento de partículas com base nas características de um volume discretizado;
- Método híbrido PIC/FLIP que combina elementos da simulação de fluidos com a simulação de partículas;
- Técnicas *noise derivatives*, *curl noise* e *perlin noise* para simular perturbações intencionais num sistema, que se baseiam na geração de números pseudoaleatórios que podem alterar certos parâmetros físicos em todos os pontos do espaço;
- Método *Finite-difference time-domain method (FDTD)* para simular ondas em ambientes com relativa complexidade;

Vários dos métodos aqui apresentados implicam a discretização do espaço para criar um campo vetorial nos vários pontos deste mesmo espaço. No caso deste projeto, o objetivo é apenas simular fenómenos gravitacionais, eletromagnéticos e intermoleculares para partículas bem abaixo da velocidade da luz. Por esta razão, o campo de força pode ser calculado em cada partícula para melhorar a *performance* do programa, pelo que o método *N-body simulation*, que se baseia apenas na dinâmica de partículas, é a melhor opção. Além disso, este método é exato para calcular a força aplicada a cada partícula uma vez que não recorre a valores interpolados como é típico de simulações com o espaço discretizado. A vantagem dos restantes métodos está na sua viabilidade em simular a propagação de ondas no espaço, uma característica que não é necessária no âmbito deste projeto.

Todos estes métodos têm como única função definir a força aplicada em cada partícula. No entanto, para daí derivar a sua velocidade e posição é necessária mais uma ferramenta matemática que será abordada no capítulo seguinte.

2.3 INTEGRADORES APLICADOS A SISTEMAS DE PARTÍCULAS

A revisão da literatura presente neste subcapítulo aborda um dos pontos críticos da simulação de sistemas de partículas: os métodos de integração numérica utilizados na resolução das equações do movimento das partículas.

Como exposto no subcapítulo anterior, para simular a dinâmica de um sistema de partículas é necessário considerar as forças que nele atuam, as quais podem depender de vários fatores. Uma vez determinadas as forças aplicadas em cada partícula, extrai-se a aceleração, a velocidade e por fim a posição. No entanto, a solução geral para este problema comporta equações diferenciais, das quais é em geral difícil ou até impossível obter soluções analíticas. A solução para este problema está nos integradores numéricos, uma vez que permitem calcular iterativamente a integração da aceleração para obter a velocidade, que por sua vez integrada permite obter a posição das partículas, tudo isto por meio de equações algébricas.

Um dos métodos mais utilizados é o método de Euler de 1.ª ordem. No entanto, embora simples, gera erros incrementais e acaba por violar leis fundamentais da física como, por exemplo, a conservação da energia. Existem modelos mais

sofisticados, como os de Runge–Kutta, Leapfrog, Yoshida e até modelos de 12.ª ordem que garantem uma propagação mínima de erros.

Desde o século XVIII que se têm criado métodos iterativos cada vez mais sofisticados e robustos. O propósito deste subcapítulo é selecionar os integradores adequados à simulação de sistemas de partículas para assim comparar os métodos com características mais interessantes em termos de desempenho, simplicidade e confiabilidade.

A principal motivação deste tópico é investigar os diferentes métodos iterativos existentes e destes recolher as principais características, assim como expor o seu funcionamento intrínseco.

O estudo dos integradores está organizado em 4 subsubcapítulos. O subsubcapítulo *Contextualização* (2.3.1) visa introduzir o propósito dos integradores e explicar o seu funcionamento. O subsubcapítulo *Estado da arte e investigação de integradores* (2.3.2) apresenta cada integrador, definindo as principais características, justificando o grupo de integradores escolhidos para o presente trabalho e explicando o funcionamento interno de cada um deles através das suas fórmulas matemáticas, representações gráficas e analogias. Por fim, o subsubcapítulo *Implementação computacional* (2.3.3) apresenta, em linguagem de programação simples, a lógica de implementação dos integradores em meio informático.

2.3.1 *Contextualização*

Para definir a necessidade e importância dos métodos iterativos aplicados a sistemas de partículas, é importante introduzir como se prevê o estado futuro do próprio sistema, de forma ideal, sem aproximações.

Primeiro, é necessário calcular a força aplicada a cada partícula, para assim prever a posição futura da mesma. O campo de forças é definido com base em várias propriedades das partículas como, por exemplo, a sua posição, velocidade, carga elétrica e massa. Para obter o valor exato destas interações à distância, utilizam-se as leis que definem o campo de força a ser calculado, como a Lei da Gravitação Universal, a força intermolecular derivada do potencial de Lennard-Jones, ou as leis de Coulomb, de Biot-Savart e de Lorentz.

Estando a força definida, considera-se a cinemática aplicada a partículas, da qual se obtém a evolução temporal da posição das partículas do sistema. Esta relação pode ser descrita a partir da 2.ª lei de Newton e de duas equações da cinemática, descritas no grupo de equações 10:

$$\vec{a} = \frac{\vec{F}}{m}, \tag{10a}$$

$$\vec{v} = \int_{t_0}^{t} \vec{a}\, dt, \tag{10b}$$

$$\vec{r} = \int_{t_0}^{t} \vec{v}\, dt, \tag{10c}$$

sendo $\vec{r}$ a posição, $\vec{v}$ a velocidade, $\vec{a}$ a aceleração, $\vec{F}$ a força resultante e m a massa da partícula. Estas 3 equações traduzem-se numa equação diferencial de segunda ordem (equação 11):

$$\vec{r} = \int_{t_0}^{t} \int_{t_0}^{t} \frac{\vec{F}(\vec{r})}{m} \, dt \, dt. \tag{11}$$

Equações diferencias nem sempre conduzem a uma solução analítica e, neste caso, cenários com 3 partículas apenas, já se mostram suficientemente complexos para que seja possível obter uma solução analítica geral, pelo que o seu estudo se baseia nos métodos iterativos. Além disso, sistemas de 3 corpos são caracterizados por serem caóticos dada a sua evolução temporal depender enormemente das condições iniciais (problema dos três corpos).

Como alternativa às equações diferenciais, os computadores têm a grande vantagem de resolver centenas de milhares de milhões de operações matemáticas por segundo, sendo a ferramenta ideal para aproveitar o potencial da integração numérica, em alternativa às soluções analíticas que nem sempre se conseguem obter das equações diferenciais. A integração, porém, pressupõe a discretização do sistema, pelo que a solução é aproximada e pode até resultar em erros elevados se não for utilizada uma técnica iterativa apropriada para o problema. A principal vantagem advém do facto destes métodos envolverem apenas operações matemáticas básicas, permitindo descrever a cinemática de um sistema de partículas sem recorrer a integrais ou derivadas.

Os métodos iterativos preveem o estado futuro do sistema dadas as condições iniciais (*initial value problem - IVP*) e um campo de forças dinâmico definido pelas próprias partículas. A partir desta informação, são feitos cálculos de forma cíclica, uma vez que o campo de forças depende, pelo menos, das posições das partículas, que por sua vez dependem das suas velocidades, que dependem da aceleração imposta pelo campo de forças. Cada um destes ciclos é chamado iteração, sendo que cada uma fornece o estado seguinte do sistema permitindo assim a análise da evolução temporal dadas as condições iniciais.

Qualquer método iterativo consegue prever a evolução de um movimento retilíneo uniforme com máxima exatidão, uma vez que a aceleração e velocidade são constantes. Numa situação em que a velocidade e a aceleração variam (como muitas vezes é o caso num sistema de partículas), é necessário abordar o problema de outra perspetiva. Para manter a exatidão da simulação, cada iteração deve introduzir uma alteração mínima nas trajetórias das partículas (quer direção, quer aceleração linear). Caso contrário, as trajetórias calculadas serão irregulares e irreais, levando a resultados inexatos. Tal como mostra a figura 11, obtida a partir da simulação gravítica num cenário simples de órbita elíptica, quanto maior o número de iterações realizadas, para o mesmo tempo de simulação, mais definidas serão as trajetórias das partículas e mais exato será o resultado. As leis de Kepler mostram que a velocidade é máxima quando os corpos se encontram mais próximos (periélio). É possível observar na imagem que a trajetória é menos definida nesses pontos, uma vez que a variação da aceleração entre iterações é superior.

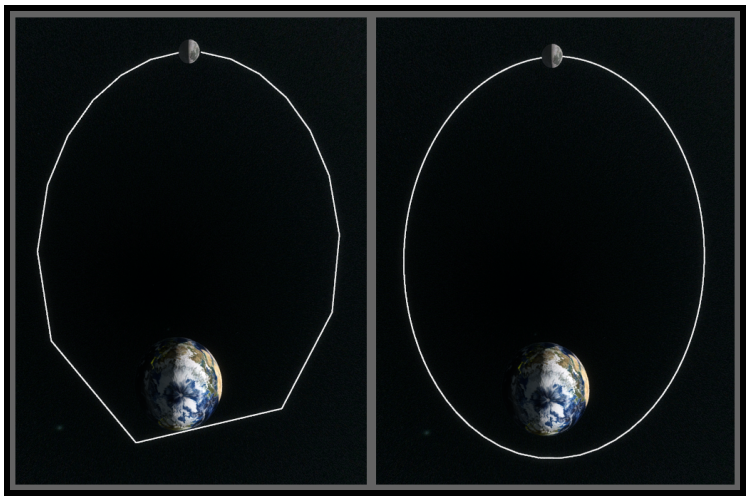

Figura 11: Comparação de uma simulação gravitacional para uma trajetória elíptica considerando um diferente número de iterações (17 à esquerda, 168 à direita).

Por esta razão, um integrador será tanto mais exato quanto mais iterações forem calculadas para o mesmo intervalo de tempo. A única exceção ocorre quanto são efetuadas tantas iterações que os erros de precisão dos números decimais se propagam de forma significativa. Para além disto, é também importante obter resultados o mais exatos possível por cada iteração. No entanto, métodos com iterações mais exatas tendem a ser aqueles com maior demanda por poder computacional, ou seja, cada iteração tem um tempo de cálculo superior. Os integradores que conseguem os resultados mais exatos por iteração, em conjunto com os cálculos computacionalmente mais rápidos, serão aqueles que melhor exercem a sua função.

Tendo isto em conta, esta investigação irá dar prioridade aos métodos que, com os mesmos recursos computacionais, consigam fornecer os resultados mais exatos, não importando se o número de iterações é elevado ou não. Existe uma nuance que será esclarecida no subsubcapítulo 2.3.2 relativamente a métodos com alta exatidão, mas com um tempo elevado de cálculo por cada iteração.

2.3.2 *Estado da arte e investigação de integradores*

Apesar de um computador ser uma ferramenta excelente para resolver problemas de integração numérica em tempo real, é ainda assim limitado. Desta forma a discretização do problema irá inevitavelmente introduzir erros no sistema, devido ao facto de um integrador não ser mais do que um método matemático para prever a evolução de uma dada função com base na informação atual e anterior. É desta premissa que surgem os vários métodos iterativos atuais, com capacidade de minimizar os erros de cálculo por meio de técnicas de refinamento sucessivo, algoritmos de correção de erros e também métodos de cálculo otimizados para o desempenho computacional.

Estas técnicas são explicadas ao longo deste capítulo. Relativamente às técnicas de desempenho computacional, que não são o foco principal, vale apenas deixar a

seguinte nota: os processadores utilizam memórias de *cache* de alta velocidade, mas baixa capacidade, para armazenar a informação utilizada com mais frequência e assim evitar o acesso repetido à *RAM (random acess memory)*, uma memória com menor velocidade (ainda que volátil). Uma implementação em código que aceda aos mesmos blocos de memória de forma sequencial, em termos de endereço, têm um melhor desempenho em geral, por utilizar a *cache* de memória repetidamente sem ter de a reescrever. Este é um fator muito relevante quando se pretende aceder às mesmas informações repetidamente e de forma intensiva, como é o caso do *Force Fields*, que no seu pico de desempenho (testado num computador pessoal), acede a posições, velocidades, acelerações, massas e outras características de partículas, dezenas de milhares de milhões de vezes por segundo (ver subcapítulo 4.6). Por esta razão, estruturar os dados para que o código aceda a blocos de memória que sejam próximos em termos de endereço, é de alta importância para o desempenho do programa.

Existem inúmeros métodos iterativos para simular computacionalmente um sistema de partículas. É importante encontrar um equilíbrio entre desempenho computacional e exatidão dos resultados. Abaixo são elencados os integradores mais promissores no âmbito deste projeto:

Método de Euler

O método de Euler, proposto por Leonhard Euler, tem como vantagem a sua simplicidade e pode ser derivado do cálculo da primeira ordem do polinómio de Taylor das equações diferenciais ordinárias da cinemática das partículas. É um método explícito, ou seja, calcula o estado futuro do sistema baseando-se apenas no estado atual.[16].

Este método é caracterizado por fornecer a posição e velocidade futuras ($\vec{r}_{t_{n+1}}$ e $\vec{v}_{t_{n+1}}$) de uma dada partícula que sofre uma aceleração $\vec{a}_{t_n}$ através das equações 12a e 12b:

$$\vec{v}_{t_{n+1}} = \vec{v}_{t_n} + \vec{a}_{t_n} \cdot \Delta t \tag{12a}$$

$$\vec{r}_{t_{n+1}} = \vec{r}_{t_n} + \vec{v}_{t_n} \cdot \Delta t \tag{12b}$$

É importante notar que este método apenas gera resultados exatos caso a velocidade e a aceleração sejam constantes para Δt. No entanto, quanto maior for o número de iterações realizadas (Δt menor), menor será a variação destes valores e mais exatos serão os resultados obtidos. Esta é uma das principais razões pela qual vários métodos iterativos não fornecerem soluções exatas.

A figura 12 mostra a evolução da posição de uma partícula num sistema bidimensional com o método de Euler para três iterações.

O método de Euler pode ser obtido diretamente a partir do polinómio de Taylor, como se mostra no grupo de equações 13:

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k \tag{13a}$$

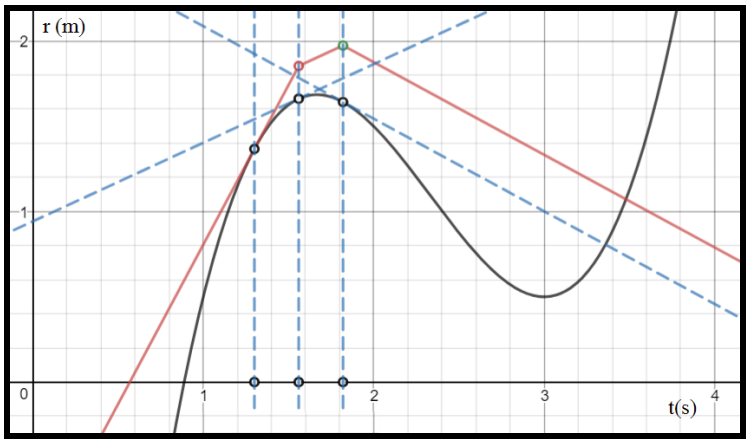

Figura 12: Representação gráfica da posição com o método iterativo de Euler para três iterações. A trajetória prevista é baseada no declive no início do intervalo de tempo. A curva a preto é a solução exata, a curva a vermelho é a prevista pelo método e as linhas azuis servem de guia para os diferentes declives em cada iteração.

$$\vec{r}(t) = \sum_{k=0}^{\infty} \frac{\vec{r}^{\,(k)}(t_0)}{k!}(t-t_0)^k \tag{13b}$$

$$\vec{r}(t) = \vec{r}(t_0) + \vec{r}\,'(t_0)\cdot(t-t_0) + O\left((t-t_0)^2\right) \tag{13c}$$

Alterando a formatação da equação de $\vec{r}(t)$ para $\vec{r}_{t_{n+1}}$, de $\vec{r}(t_0)$ para $\vec{r}_{t_n}$, de $\vec{r}\,'(t_0)$ para $\vec{v}_{t_n}$, de $(t-t_0)$ para Δt e considerando $O((t-t_0)^2)$ um erro desprezável considerando que t é próximo de t_0, obtemos a equação 12b, relativa ao cálculo posição. O mesmo se aplica à equação para a velocidade (12a). É importante notar que $O(\Delta t^2)$ é o erro causado por cada iteração e é chamado de erro local de truncamento.

Método de Euler semi-implícito

O método de Euler semi-implícito é uma modificação ao método de Euler direcionado à resolução de sistemas hamiltonianos (baseados na lei da conservação da energia), o que significa que é um integrador simplético, por definição. Por esta razão este método produz melhores resultados que o método clássico, uma vez que dispõe de uma melhor capacidade de conservar a energia mecânica do sistema.

Este método é semi-implícito porque estima a posição futura $\vec{r}_{t_{n+1}}$ com base na velocidade futura $\vec{v}_{t_{n+1}}$, ou seja, o cálculo da velocidade é uma operação explícita, mas o cálculo da posição é uma operação implícita, efetuada segundo as equações do grupo 14:

$$\vec{v}_{t_{n+1}} = \vec{v}_{t_n} + \vec{a}_{t_n}\cdot\Delta t \tag{14a}$$

$$\vec{r}_{t_{n+1}} = \vec{r}_{t_n} + \vec{v}_{t_{n+1}}\cdot\Delta t \tag{14b}$$

A figura 13 mostra a evolução da posição de uma partícula num sistema bidimensional com o método semi-implícito de Euler para três iterações.

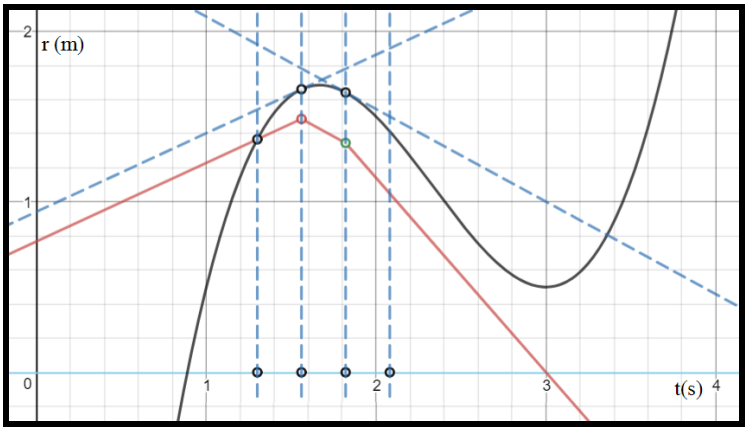

Figura 13: Representação gráfica da posição com o método semi-implícito de Euler para três iterações. A trajetória prevista é baseada no declive ao fim do intervalo de tempo. A curva a preto é a solução exata, a curva a vermelho é a prevista pelo método e as linhas azuis servem de guia para os diferentes declives em cada iteração.

Método Velocidade de *Verlet*

O método de Verlet [17], ou método de Störmer, foi criado e redescoberto por vários matemáticos para prever o movimento de partículas sujeitas a forças que dependem da sua posição, tendo sido utilizado inicialmente para simular a dinâmica molecular e prever a trajetória do cometa Halley em 1909. É um integrador simplético e não tem uma complexidade muito superior à do método de Euler, tendo um desempenho computacional semelhante. A sua grande vantagem vem do facto de possuir simetria temporal, ou seja, ser reversível no tempo. Esta característica permite utilizar intervalos de tempo Δt negativos para chegar a um estado anterior do sistema, ou até mesmo ao estado inicial. Outros métodos, como o de Euler, não têm esta característica.

Neste caso, será avaliada uma variação deste método, chamado método de Velocidade de *Verlet* [18]. O método clássico é constituído por uma única equação que determina a posição futura através da posição atual e da posição anterior. Já esta variante, é constituída por duas equações, sendo que uma delas determina a velocidade (15b) e outra a posição (15a), sendo por isso mais conveniente, uma vez que expõe diretamente a velocidade da partícula.

$$\vec{r}_{t_{n+1}} = \vec{r}_{t_n} + \vec{v}_{t_n} \cdot \Delta t + \vec{a}_{t_n} \cdot \frac{1}{2}\Delta t^2 \tag{15a}$$

$$\vec{v}_{t_{n+1}} = \vec{v}_{t_n} + \frac{\vec{a}_{t_n} + \vec{a}_{t_{n+1}}}{2} \cdot \Delta t \tag{15b}$$

Método Leapfrog

O método *Leapfrog* caracteriza-se pelo facto do cálculo da posição e da velocidade serem feitos em momentos temporais diferentes ao longo da simulação. Na prática, esta condição faz com que a velocidade futura prevista de uma partícula seja definida pela média das acelerações do intervalo de tempo atual e do intervalo de tempo seguinte. A figura 14 mostra um esquema visual do método *Leapfrog* onde a posição, velocidade e tempo são simbolizados por x, v e t.

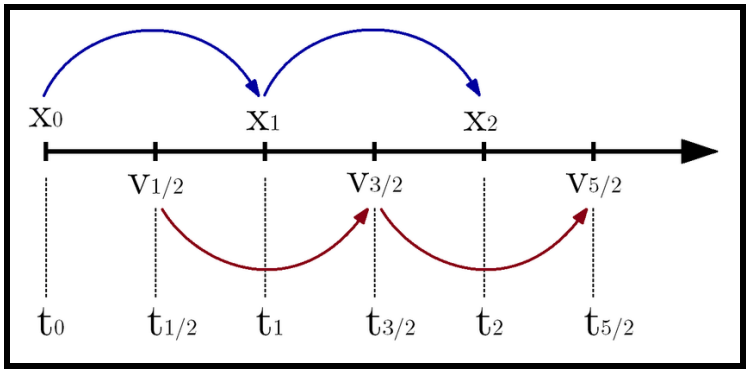

Figura 14: Esquema representativo do funcionamento do algoritmo Leapfrog. O cálculo da posição e da velocidade são feitos em momentos temporais diferentes ao longo da simulação. A posição, velocidade e tempo são simbolizados por x, v e t sendo que os seus índices representam o intervalo de tempo [19].

A formulação do método Leapfrog apresentada no grupo de equações 16 é chamada de *kick-drift-kick*:

$$\vec{v}_{t+1/2} = \vec{v}_{t_n} + \frac{1}{2}\vec{a}_{t_n} \cdot \Delta t \tag{16a}$$

$$\vec{r}_{t_{n+1}} = \vec{r}_{t_n} + \vec{v}_{t+1/2} \cdot \Delta t \tag{16b}$$

$$\vec{v}_{t_{n+1}} = \vec{v}_{t+1/2} + \frac{1}{2}\vec{a}_{t_{n+1}} \cdot \Delta t \tag{16c}$$

Tanto este método como o método de velocidade de *Verlet*, são de segunda ordem, simpléticos e também temporalmente simétricos. Na verdade, ambos os métodos são exatamente iguais em termos matemáticos, a lógica por detrás da sua criação é que se originou em raciocínios diferentes.

O método de *Leapfrog* calcula $\vec{v}_{t+1/2}$ como cálculo auxiliar (equação 16a). No entanto, substituindo $\vec{v}_{t+1/2}$ nas equações para calcular a posição (16b) e velocidade (16c), constata-se que ambos os métodos são, na verdade, o mesmo.

Nem todos os métodos são diretamente provenientes do polinómio de Taylor, especialmente métodos de ordem superior a 2, até porque existem bastantes métodos diferentes com a mesma ordem. No entanto, tal como para o método de Euler, o método de velocidade de *Verlet* e de *Leapfrog* podem também ser derivados do polinómio de Taylor. Partindo da equação 13b:

$$\vec{r}(t) = \vec{r}(t_0) + \vec{r}\,'(t_0) \cdot (t - t_0) + \frac{\vec{r}\,''(t_0)}{2} \cdot (t - t_0)^2 + O\left((t - t_0)^3\right) \tag{17}$$

A equação 17 é equivalente à equação 15a, desde que se despreze o erro $O\left(\Delta t^3\right)$ considerando que Δt tende para zero.

Estes métodos permitem reversão no tempo, o que significa que, caso se queira saber o estado anterior do sistema, colocando Δt negativo, é possível retornar até ao estado inicial de simulação sem desvios.

A figura 15 mostra a progressão temporal da velocidade calculada pelos métodos referidos para três iterações. Como é possível observar, estes métodos são mais exatos que os métodos de Euler explícito e semi-implícito. Além disso, são análogos à integração de ponto médio uma vez que preveem a velocidade futura tendo em conta a média do declive (aceleração) do estado atual e do estado futuro estimado.

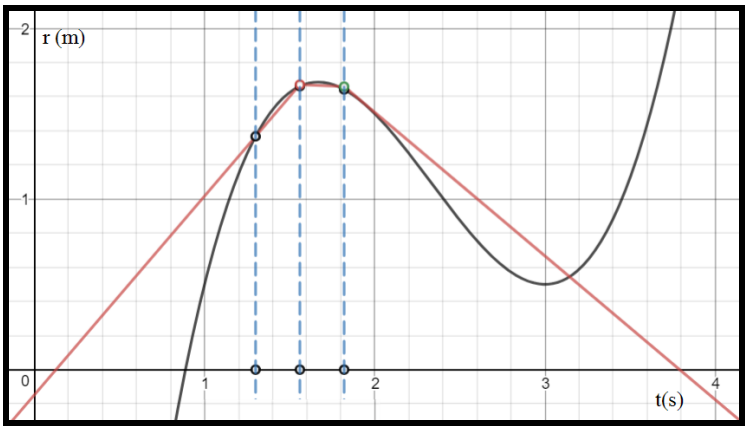

Figura 15: Representação gráfica do método *Leapfrog / Velocidade de Verlet* para duas iterações. A trajetória prevista é baseada na média do declive ao início do intervalo de tempo e ao fim do intervalo de tempo. A curva a preto é a solução exata, a curva a vermelho é a prevista pelo método.

Método de *Runge-Kutta* de 4.ª ordem (*RK-4*)

Desenvolvido por Carl Runge [20] e Wilhelm Kutta [21], o método de *Runge-Kutta* é na realidade uma família de métodos de múltipla ordem e inclui o próprio método de Euler, que neste contexto pode ser chamado de método Runge-Kutta de 1.ª ordem. O método mais conhecido da família é de 4.ª ordem, o que significa que o erro local de truncamento é de ordem $O(\Delta t^5)$, ao contrário de métodos de 1.ª ordem, onde o erro local de truncamento é proporcional a Δt^2. O erro deste método de 4.ª ordem é o mesmo que se obtém quando se utiliza a expansão de Taylor até à quarta derivada. As fórmulas deste método incluem coeficientes pré-calculados, sendo que existem muitas variações do método com coeficientes diferentes para obter características específicas [22].

O método de Runge-Kutta realiza 4 passos sequenciais por iteração, para assim obter 4 cenários distintos prováveis e posteriormente fazer uma média ponderada para obter o valor mais provável de evolução do sistema. A velocidade e posição futuras são obtidas das equações do grupo 18, que como se pode observar, são implementações do método de Euler explícito, com a diferença de que os declives da evolução do sistema (aceleração para a velocidade e velocidade para a posição) são gerados a partir da média ponderada de 4 declives/coeficientes:

$$\vec{v}_{n+1} = \vec{v}_n + \Delta t \left(\frac{\vec{k}_1^a}{6} + \frac{\vec{k}_2^a}{3} + \frac{\vec{k}_3^a}{3} + \frac{\vec{k}_4^a}{6} \right) \tag{18a}$$

$$\vec{r}_{n+1} = \vec{r}_n + \Delta t \left(\frac{\vec{k}_1^v}{6} + \frac{\vec{k}_2^v}{3} + \frac{\vec{k}_3^v}{3} + \frac{\vec{k}_4^v}{6} \right) \tag{18b}$$

Estes coeficientes também são calculados a partir do método de Euler, mas utilizam condições iniciais e intervalos de tempo distintos.

Por exemplo, os primeiros coeficientes, mostrados no grupo de equações 19, geram $\vec{k}_1^a$, que representa o declive da velocidade e $\vec{k}_1^v$, o declive da posição. O coeficiente relativo à posição $\vec{k}_1^r$ é apenas um cálculo auxiliar para obter $\vec{k}_1^a$, sendo este valor determinado pela função aceleração $Acc(posição)$ que a partícula sofre na posição $\vec{k}_1^r$. Estes coeficientes, aplicados nas equações 18a e 18b, têm um peso de $1/6$ no resultado final e representam o declive no início do intervalo (tal como no método clássico de Euler).

$$\vec{k}_1^r = \vec{r}_{t_n} \tag{19a}$$

$$\vec{k}_1^v = \vec{v}_{t_n} \tag{19b}$$

$$\vec{k}_1^a = Acc(\vec{k}_1^r) \tag{19c}$$

Já dos segundos coeficientes, visíveis no grupo de equações 20, obtém-se o declive no ponto médio do intervalo (daí $\frac{\Delta t}{2}$). Este ponto médio é estimado a partir dos primeiros coeficientes (grupo de equações 19), no sentido análogo à extensão de uma reta com dado declive (primeiros coeficientes) até metade do intervalo de tempo.

$$\vec{k}_2^r = \vec{r}_{t_n} + \vec{k}_1^v \cdot \frac{\Delta t}{2} \tag{20a}$$

$$\vec{k}_2^v = \vec{v}_{t_n} + \vec{k}_1^a \cdot \frac{\Delta t}{2} \tag{20b}$$

$$\vec{k}_2^a = Acc(\vec{k}_2^r) \tag{20c}$$

Dos terceiros coeficientes (grupo de equações 21), também se obtém o declive no ponto médio do intervalo. No entanto, este ponto médio é estimado a partir do declive dos segundos coeficientes. Os coeficientes estimados no ponto médio (2^{os} e 3^{os}) têm um peso superior, de $\frac{1}{3}$.

$$\vec{k}_3^r = \vec{r}_{t_n} + \vec{k}_2^v \cdot \frac{\Delta t}{2} \tag{21a}$$

$$\vec{k}_3^v = \vec{v}_{t_n} + \vec{k}_2^a \cdot \frac{\Delta t}{2} \tag{21b}$$

$$\vec{k}_3^a = Acc(\vec{k}_3^r) \tag{21c}$$

Os quartos coeficientes, representados no grupo de equações 22), têm um peso de $\frac{1}{6}$ tal como os primeiros. Destes coeficientes obtém-se o declive no fim do intervalo, sendo que o fim do intervalo é estimado a partir dos terceiros coeficientes.

$$\vec{k}_4^r = \vec{r}_{t_n} + \vec{k}_3^v \cdot \Delta t \tag{22a}$$

$$\vec{k}_4^v = \vec{v}_{t_n} + \vec{k}_3^a \cdot \Delta t \tag{22b}$$

$$\vec{k}_4^a = Acc(\vec{k}_4^r) \tag{22c}$$

Tendo os quatro coeficientes da aceleração para estimar a velocidade futura e os quatro coeficientes da velocidade para estimar a posição futura, basta calcular 18a e 18b ficando assim uma iteração finalizada utilizando o método de *Runge-Kutta* de 4.ª ordem.

Método de *Yoshida* de 4.ª e 6.ª ordem

Haruo Yoshida criou, em 1990, algoritmos que possibilitaram o aumento da ordem do método Leapfrog. Desta forma, o método Leapfrog é aplicado a diversos intervalos de tempo intermédios. A aceleração da partícula é calculada várias vezes a partir de diferentes posições, constituindo assim uma sucessão de sub-iterações, possibilitando uma maior exatidão. Consequentemente o custo computacional aumenta, uma vez são realizados mais cálculos por iteração. A grande vantagem é que quando são utilizados os intervalos de tempo intermédios corretos (definidos por coeficientes pré-calculados), como o método de Yoshida, os erros de cada cálculo tendem a cancelar-se entre si, levando assim a uma maior exatidão para simulações longas.

O método de Yoshida de 4.ª ordem [23] toma os coeficientes que se mostram nas equações do grupo 23:

$$c_1 = c_4 = \frac{1}{2(2 - \sqrt[3]{2})} \tag{23a}$$

$$c_2 = c_3 = \frac{1 - \sqrt[3]{2}}{2(2 - \sqrt[3]{2})} \tag{23b}$$

$$d_1 = d_3 = 2c_1 \tag{23c}$$

$$d_2 = -\frac{\sqrt[3]{2}}{2 - \sqrt[3]{2}} \tag{23d}$$

Estes coeficientes são calculados a partir de um método criado por *Yoshida* no qual $c_1 + c_2 + c_3 + c_4 = 1$ e $d_1 + d_2 + d_3 = 1$, sendo esta uma condição necessária para evitar aumentos ou decréscimos da energia total do sistema.

As equações do grupo 24 são feitas de forma sequencial até obter $\vec{r}_{n+1}$ e $\vec{v}_{n+1}$:

$$\vec{r}_1 = \vec{r}_{t_n} + c_1 \cdot \vec{v}_{t_n} \cdot \Delta t \tag{24a}$$

$$\vec{v}_1 = \vec{v}_{t_n} + d_1 \cdot \mathrm{Acc}(\vec{r}_1) \cdot \Delta t \tag{24b}$$

$$\vec{r}_2 = \vec{r}_1 + c_2 \cdot \vec{v}_1 \cdot \Delta t \tag{24c}$$

$$\vec{v}_2 = \vec{v}_1 + d_2 \cdot \mathrm{Acc}(\vec{r}_2) \cdot \Delta t \tag{24d}$$

$$\vec{r}_3 = \vec{r}_2 + c_3 \cdot \vec{v}_2 \cdot \Delta t \tag{24e}$$

$$\vec{v}_{n+1} = \vec{v}_2 + d_3 \cdot \mathrm{Acc}(\vec{r}_3) \cdot \Delta t \tag{24f}$$

$$\vec{r}_{n+1} = \vec{r}_3 + c_4 \cdot \vec{v}_3 \cdot \Delta t \tag{24g}$$

Para concluir a presente análise, foram extraídas as várias características que definem os integradores estudados reconhecendo as similaridades e diferenças de cada um [24]:

- Tipo - Um método pode ser explícito ou implícito. Um método explícito realiza cálculos para prever o estado futuro do sistema unicamente utilizando informação do estado atual. Já um método implícito, por outro lado, prevê o estado futuro do sistema utilizando valores do estado futuro previsto e do estado atual. Um método implícito é normalmente mais exato.
- S - Métodos simpléticos são, por definição, métodos próprios para a resolução de sistemas hamiltonianos, ou seja, tendem a conservar a energia do sistema.
- ST - Métodos com simetria temporal, permitem reversão no tempo, ou seja, é possível voltar ao estado inicial do sistema sem desvios, desde que se utilize sempre o mesmo método e intervalo de tempo absoluto.
- CTI - Métodos com cálculo temporal intercalado, integram a posição e a velocidade em momentos temporais diferentes (de forma intercalada), tendo a vantagem de manterem estável um sistema com trajetórias oscilatórias/repetitivas.

- O - A ordem de um método está ligada à sua exatidão, mas também à sua complexidade. Para exemplificar o significado da ordem de um método, considere-se como exemplo simples, uma função $f(x)$ qualquer. Um método de 1.ª ordem tem em conta apenas o declive (1.ª derivada) da função no ponto em análise, pelo que a previsão seguinte será a extensão da reta tangente à curva no próximo instante de tempo. Um método de 2.ª ordem, tem em conta a 2.ª derivada da função naquele ponto. Quanto maior for a ordem de um método, mais informação se obtém por iteração para prever o valor futuro. A ordem do método numérico refere-se à ordem do polinómio de Taylor utilizado na aproximação de $f(x)$, onde uma ordem mais elevada, indica uma melhor previsão da evolução de uma dada curva.

A tabela 2 mostra estas mesmas características aplicadas a cada integrador abordado:

Tabela 2: Características de cada integrador.

Integrador	Tipo	S	ST	O	CTI
Euler	Exp	✗	✗	1.ª	✗
S.I. Euler	Imp	✓	✗	1.ª	✗
Verlet	Exp	✓	✓	2.ª	✓
Leapfrog	Exp	✓	✓	2.ª	✓
RK-4	Exp	✗	✗	4.ª	✗
Yoshida 4.ª	Exp	✓	✗	4.ª	✓
Yoshida 6.ª	Exp	✓	✗	6.ª	✓

Métodos de ordem superior

Foram ainda analisados métodos de ordem superior como o método de Runge–Kutta–Nyström de 12.ª ordem [25]. Métodos como este produzem resultados altamente fiáveis por iteração. No entanto, cada iteração (apesar de ser mais exata em relação a métodos de ordem inferior) requer mais recursos computacionais, o que, para uma aplicação em tempo real, não é desejável. Utilizando este tipo de métodos a simulação teria uma taxa de atualização baixa, devido ao longo tempo necessário para calcular uma iteração, comprometendo a fluidez do programa. Este problema pode ser resolvido utilizando interpolação de valores entre cálculos para mover as partículas de forma suave, ao atualizar as suas posições mais vezes. No entanto, a complexidade de implementação iria aumentar desnecessariamente, devido à necessidade de reservar um núcleo do processador que calculasse a interpolação, enquanto todos os outros dedicar-se-iam à computação do integrador. Além disso, estes integradores não garantem um resultado mais preciso em aplicações em tempo real pelo facto de o número de iterações ser menor para o mesmo intervalo de tempo. Foi esta a principal razão pela qual os integradores de ordem superior foram preteridos. Todos os outros integradores abordados estão

implementados no *Force Fields*. O próximo capítulo irá mostrar a implementação computacional aplicada no programa, relativamente aos 7 integradores anteriores.

2.3.3 *Implementação computacional*

A tradução de linguagem matemática para a programação, relativamente aos integradores referidos no capítulo 2.3.2, é um passo necessário no desenvolvimento do *Force Fields*.

Uma vez que a principal linguagem de programação utilizada foi o *C++*, que se caracteriza por ter uma sintaxe relativamente complexa, os exemplos serão descritos utilizando *Python*, uma linguagem mais recente, criada tendo em conta a legibilidade do código. Além disso, a força gravítica é tida como exemplo para explicar os blocos de código com facilidade, uma vez que é uma força simples no sentido em que a única variável ao longo do tempo é a posição das partículas, e não depende da velocidade, como no caso do campo eletromagnético.

Método de Euler

Para implementar o método de Euler computacionalmente, considera-se uma lista de partículas com origem numa iteração anterior ou simplesmente preenchida com as condições iniciais do sistema. O bloco de código abaixo gera a nova lista correspondente à iteração seguinte:

```
# old_particle_list was defined before

new_particle_list = []
for old_particle in old_particle_list:
    acceleration = Acc(old_particle.position)
    new_particle = Particle()
    new_particle.mass = old_particle.mass
    new_particle.velocity = old_particle.velocity + acceleration * delta
    new_particle.position = old_particle.position +
        old_particle.velocity * delta
```

As partículas serão submetidas a uma aceleração $\vec{a}_i$ (equação 25) causada exclusivamente pela força gravítica exercida entre elas, calculada a partir da constante gravitacional G, das suas posições $\vec{r}_i$ e $\vec{r}_j$ e das suas massas m_j:

$$\vec{a}_i = G \sum_{j=1}^{n} m_j \frac{\vec{r}_j - \vec{r}_i}{|\vec{r}_j - \vec{r}_i|^3}. \tag{25}$$

Sempre que apareça a função "Acc(position)" é retornado o resultado do cálculo da aceleração gravítica mostrado na equação 25, baseado numa dada posição, que se pode traduzir em *python* da seguinte forma:

```
Acc(pi_position):
    acc = Vector3(0, 0, 0)
    for pj in particle_list:
        displacement = pj.position - pi_position
        square_dist = abs(displacement)
```

```
6           acc += pj.mass * displacement / (square_dist * square_dist *
                square_dist)
7       return acc * G
```

Método de Euler semi-implícito

O código para este método toma a seguinte forma:

```
1  acceleration = Acc(old_particle.position)
2  new_particle.velocity = old_particle.velocity + acceleration * delta
3  new_particle.position = old_particle.position + new_particle.velocity *
       delta
```

Método Velocidade de Verlet / Método *Leapfrog*

O código para este método toma a seguinte forma:

```
1  new_particle.position = old_particle.position + old_particle.velocity *
       delta + old_particle.acceleration * 0.5 * delta * delta
2  new_particle.acceleration = Acc(new_particle.position)
3  new_particle.velocity = old_particle.velocity +
       (old_particle.acceleration + new_particle.acceleration) * 0.5 *
       delta
```

É importante ter em atenção que neste método, a aceleração é necessariamente calculada após o cálculo da nova posição, uma vez que depende desta. No entanto a aceleração tem de ser calculada antes do cálculo da velocidade futura, uma vez que a velocidade depende da aceleração. Apesar desta característica, o método também é utilizado para simular sistemas onde a força depende da posição e velocidade das partículas, como é o caso da força eletromagnética. A diferença é que, nesta situação, a aceleração iria depender da nova posição e da velocidade anterior: Acc(new_particle.position, old_particle.velocity).

Método de Runge-Kutta de 4.ª ordem (*RK-4*)

Uma possibilidade de implementação do método de Runge-Kutta é a seguinte:

```
1  # Compute k1
2  k1_r = old_particle.position
3  k1_v = old_particle.velocity
4  k1_a = Acc(k1_r)
5
6  # Compute k2
7  k2_r = k1_r + 0.5 * delta * k1_v
8  k2_v = k1_v + 0.5 * delta * k1_a
9  k2_a = Acc(k2_r)
10
11 # Compute k3
12 k3_r = k1_r + 0.5 * delta * k2_v
13 k3_v = k1_v + 0.5 * delta * k2_a
14 k3_a = Acc(k3_r)
15
16 # Compute k4
17 k4_r = k1_r + delta * k3_v
```

```
k4_v = k1_v + delta * k3_a
k4_a = Acc(k4_r)

# Update position and velocity
new_particle.position = old_particle.position + delta / 6.0 * (k1_v + 2
    * k2_v + 2 * k3_v + k4_v)
new_particle.velocity = old_particle.velocity + delta / 6.0 * (k1_a + 2
    * k2_a + 2 * k3_a + k4_a)
```

Ao contrário dos métodos anteriores onde a aceleração é calculada apenas uma vez para cada iteração, este método realiza este passo quatro vezes, tendo assim um impacto computacional significativo. Esta desvantagem, que existe também nos métodos seguintes, é no entanto mitigada devido ao aumento da exatidão do cálculo por iteração, um tópico que será abordado no capítulo 5.

Método de Yoshida de 4.ª e 6.ª ordem

O método de Yoshida de 4.ª ordem pode ser implementado da seguinte forma:

```
# Coefficient definition
c1 = 0.6756035959798289
c2 = -0.1756035959798289
c3 = c2
c4 = c1
d1 = 1.3512071919596578
d2 = -1.7024143839193153
d3 = d1

# 1st sub-iteration
newPos = old_particle.position + c1 * old_particle.velocity * delta;
newVelo = old_particle.velocity + d1 * Acc(newPos) * delta;

# 2st sub-iteration
newPos += c2 * newVelo * delta;
newVelo += d2 * Acc(newPos) * delta;

# 3rd sub-iteration
newPos += c3 * newVelo * delta;
newVelo += d3 * Acc(newPos) * delta;

# 4th sub-iteration
newPos += c4 * newVelo * delta;

new_particle.position = newPos
new_particle.velocity = newVelo
```

O método de *Yoshida* de 6.ª ordem testado tem 16 sub-iterações. A lógica deste método é em tudo semelhante ao de 4.ª ordem. Por essa razão (e uma vez que possui uma grande quantidade de equações), será apenas apresentado o código utilizado que tem a vantagem de usar um ciclo *for* para condensar as equações repetitivas:

```
# Coefficient definition
pCoeffs = [0.74409614601461,
```

```
-0.425227929490565,
0.2762016693478,
-0.0029155067911599275,
-1.4465510537023341,
1.4478689195210459,
1.4451324786403945,
-1.5386047235397915,
-1.5386047235397915,
1.4451324786403945,
1.4478689195210459,
-1.4465510537023341,
-0.0029155067911599275,
0.2762016693478,
-0.425227929490565,
0.74409614601461]

vCoeffs = [
1.48819229202922,
-2.33864815101035,
2.89105148970595,
-2.89688250328827,
0.00378039588360192,
2.89195744315849,
-0.00169248587770116,
-3.075516961201882,
-0.00169248587770116,
2.89195744315849,
0.00378039588360192,
-2.89688250328827,
2.89105148970595,
-2.33864815101035,
1.48819229202922]

# Sub-iterations
newPos = old_particle.position
newVelo = old_particle.velocity

for i in range(15):
    newPos += delta * pCoeffs[i] * newVelo
    newVelo += delta * vCoeffs[i] * Acc(newPos)

    newPos += delta * pCoeffs[15] * newVelo

    new_particle.position = newPos
    new_particle.velocity = newVelo
```

3

DESENVOLVIMENTO DO *FORCE FIELDS*

Após o estudo de toda a base teórica apresentada no capítulo anterior, o próximo passo é colocar a ideia em prática, que se inicia com o desenvolvimento do *Force Fields*, já apresentado nos objetivos (subcapítulo 1.2).

3.1 BASE DE DESENVOLVIMENTO DO PROJETO

O *Force Fields* não foi totalmente criado a partir de uma linguagem de programação. Isto deve-se ao facto de a aplicação necessitar de ferramentas como:

- Renderização tridimensional em tempo real, incluindo texturas, materiais, iluminação dinâmica e pós-processamento;
- interfaces gráficas interativas com botões, barras de progresso, menus e caixas de texto;
- otimizações para apresentar uma enorme quantidade de objetos no ecrã como o instanciamento de malhas;
- suporte de uma linguagem de programação de GPU para realizar operações vetoriais a alta velocidade.

Para tal, a melhor ferramenta para desenvolver este projeto foi um motor de jogos, comummente conhecido como *Game Engine*. Esta peça de *software* é composta por um conjunto de ferramentas e bibliotecas projetadas para facilitar o desenvolvimento de jogos digitais. No entanto, é uma ferramenta igualmente útil para a criação de qualquer conteúdo que envolta renderização gráfica digital como produção de animações e imagens, jogos, demonstrações interativas de modelos tridimensionais e até mesmo produção de aplicações gráficas como é o caso do *Force Fields*.

Existem atualmente vários motores de jogos no mercado. O mais famoso é o *Unreal Engine* [26], que se caracteriza por ter os melhores gráficos em tempo real do mercado uma vez que utiliza as técnicas de iluminação em tempo real e de renderização mais avançadas. No entanto, é focada precisamente em gráficos realistas, que não é o objetivo deste projeto. Além disso, o *software* é de tal maneira complexo que ocupa mais de *100 GB* em disco rígido e a criação de um novo projeto em branco demora cerca de 3 a 5 minutos num computador recente (2022) adaptado para a utilização de aplicações gráficas. É portanto um motor que exige extensos recursos computacionais e tempo de espera. É ainda um *software* licenciado, uma vez que contém restrições relativamente à publicação do produto com ele desenvolvido. Por estas razões, apesar de ter sido testado, foi descartado para uso como base deste projeto. A interface de utilizador do *Unreal Engine* pode ser observada na figura 16.

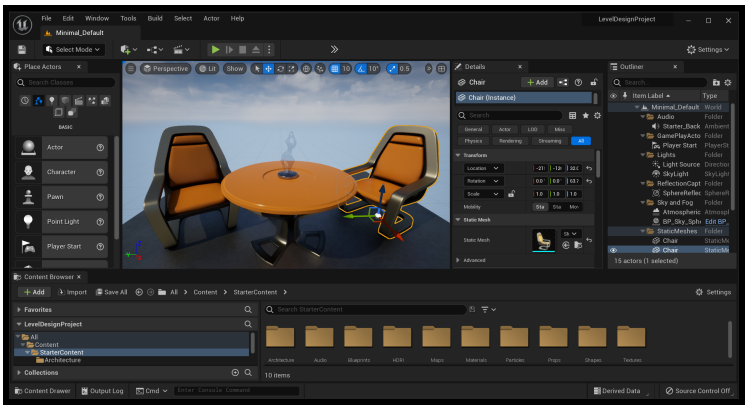

Figura 16: Interface gráfica da *Unreal Engine*, um dos motores de jogos testado para desenvolver o projeto, conhecido pela qualidade gráfica impressionante.

Outro motor de jogos também bastante famoso é o *Unity Real-Time Development Platform* [27]. Por ser altamente versátil e muito bem documentado, este *software* tem uma enorme comunidade de utilizadores, pelo que aprender a utilizar e criar sistemas é uma tarefa simplificada devido à ampla disponibilidade de informação. Pelo facto deste motor ser mais antigo e ter um longo período de desenvolvimento, comporta bastantes funcionalidades. No entanto, muitas delas estão mal mantidas ou desatualizadas, pelo que apesar de ser um programa apto à criação de aplicações leves e rápidas, é relativamente desorganizado e desnecessariamente complexo. Por estas razões, apesar de ser um *software* poderoso, foi também colocado de lado. A interface de utilizador do *Unity* pode ser observada na figura 17.

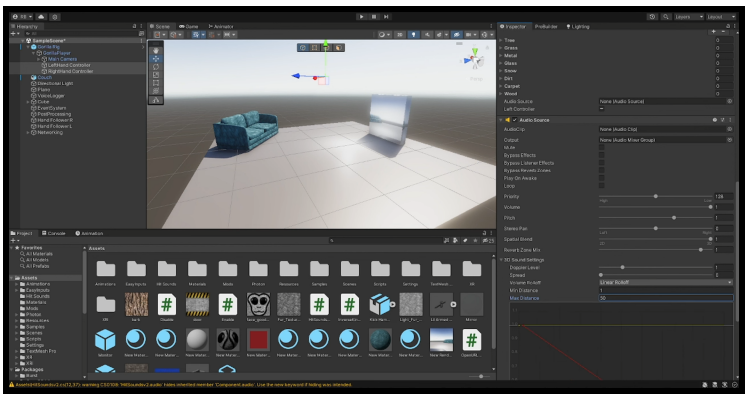

Figura 17: Interface de utilizador da *Unity*, um dos motor de jogos mais utilizados atualmente.

Foram ainda analisadas outras *game engines* como a *Cry Engine* [28], *Armory3D* [29], *UPBGE* [30] (utilizada no desenvolvimento inicial do projeto) e ainda outras como a *Torque 3D* [31] e a *Stride3D* [32], todas elas com as suas vantagens e desvantagens, sendo os seus pontos mais fracos, a falta de documentação, pouca manutenção/atualização recorrente do *software* e também políticas de licenciamento de uso pouco abertas.

Felizmente, existe um motor de jogos free and open-source software (FOSS) (*free and open-source*) que tem vindo a ganhar uma comunidade de utilizadores e

colaboradores enorme nos últimos anos. O seu nome é *Godot Game Engine* [33] e caracteriza-se por ter uma excelente documentação, capacidade para criar *software* 2D e 3D com excelente qualidade gráfica e acima de tudo, ser extremamente leve. Este *software* ocupa apenas 120 MB de memória após instalação levando menos de 15 segundos para criar um projeto em branco (na atual versão 4.4.1 e utilizando o mesmo computador anteriormente usado na criação de um projeto no *Unreal Engine*). É portanto um *software* poderoso, embora simples e compacto, sendo também a ferramenta de desenvolvimento ideal para quem gosta de ter uma experiência de trabalho fluida. Além disso, este motor de jogos tem funcionalidades completas o suficiente para que se despenda o mínimo de tempo a trabalhar fora do âmbito do projeto, uma vez que uma das principais filosofias de desenvolvimento do motor está na disponibilização de ferramentas que tomariam muito tempo do utilizador se este tivesse de as implementar de raiz. A interface de utilizador do *Godot* pode ser observada na figura 18.

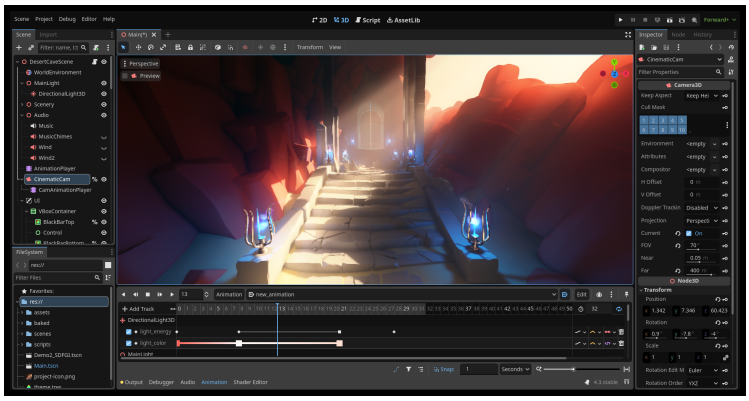

Figura 18: Interface gráfica do *Godot*, a *game engine* com as melhores características que se adaptam às necessidades do projeto.

3.2 ESTRUTURA E PROGRAMAÇÃO COM GODOT GAME ENGINE

Trabalhar com o *Godot* é relativamente simples para quem tiver experiência anterior com motores de jogos. O *software* caracteriza-se por ter uma interface intuitiva, oferecer uma vasta gama de ferramentas integradas, como um editor visual 3D, um sistema de programação através da linguagem *GDScript* e, acima de tudo, uma arquitetura modular que facilita a reutilização de componentes e criação de hierarquias, tornando o desenvolvimento mais eficiente e organizado.

Abaixo é descrita a filosofia de *design* do *Godot* [34]:

- *Design* orientado a objetos e composição uma vez que utiliza um sistema flexível de cenas e hierarquia de nós para estruturar jogos de forma intuitiva.
- Foco na independência de desenvolvimento, incluindo ferramentas integradas como editores de *script*, animação, *tilemap*, *shaders*, depurador e *profiler*, fornecendo uma experiência de desenvolvimento contínua sem a dependência de outros *softwares*.
- O projeto é de código aberto, permitindo que qualquer utilizador contribua para o seu desenvolvimento e beneficie das contribuições da comunidade.

- O desenvolvimento é guiado por esta mesma comunidade, que participa na evolução e melhoria constante do motor de jogos.
- O próprio editor do *Godot*, ou seja, o programa em si, é construído com o motor que executa as aplicações desenvolvidas, demonstrando a sua flexibilidade e capacidades.
- Disponibiliza motores dedicados para 2D e 3D, o que garante funcionalidades específicas e otimizadas, sendo esta uma vantagem que vários motores de jogos do mercado não disponibilizam uma vez que os seus motores 2D são, na verdade, o sistema tridimensional limitado numa das dimensões, resultando assim numa perda de desempenho e numa experiência de desenvolvimento desnecessariamente complexa.

A figura 19 mostra a interface de utilizador do *Godot*, exibindo, neste caso, o projeto associado à presente dissertação. A interface do *Godot* está dividida em diversos separadores, sendo os mais utilizados:

- Um visualizador 3D / 2D, ao centro;
- a estrutura da cena com os diversos nós em hierarquia, no canto superior esquerdo;
- um gestor de ficheiros, na parte inferior esquerda;
- uma barra de propriedades para o nó selecionado, à direita;
- uma consola para auxiliar na programação e *debugging*, abaixo ao centro.

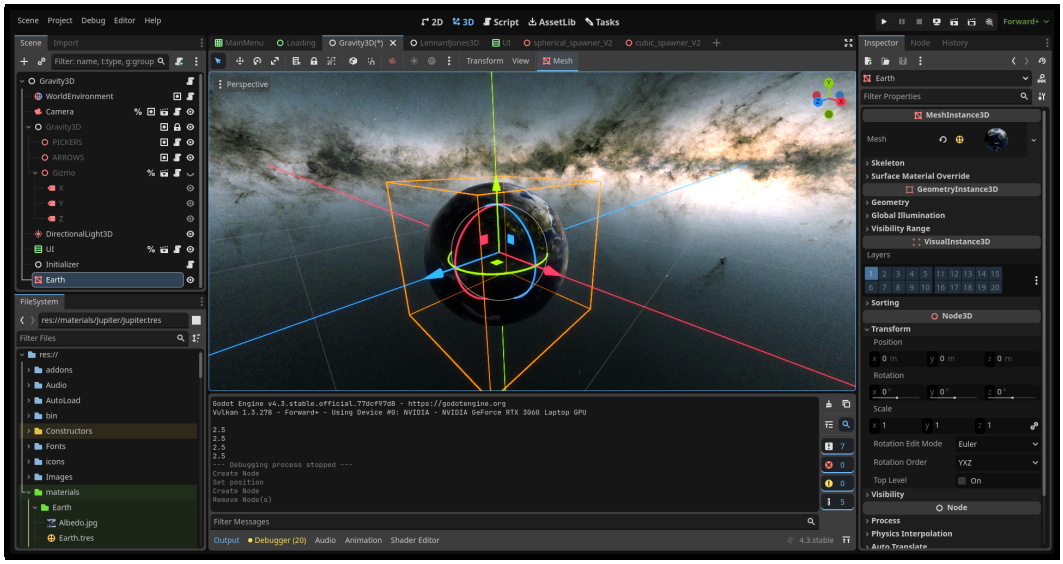

Figura 19: Interface gráfica base do *Godot*. Contém um visualizador 3D / 2D, a estrutura da cena com os diversos nós em hierarquia, um gestor de ficheiros, uma barra de propriedades do nó selecionado e ainda uma consola para auxiliar na programação.

A organização do *Godot* baseia-se numa arquitetura hierárquica de cenas e nós. Cada cena é uma unidade independente que contém um ou mais nós, permitindo uma construção modular e reutilizável pelo facto de ser possível duplicar cenas e variar os seus parâmetros para obter comportamentos diferentes. Os nós

representam elementos fundamentais, como malhas 3D, componentes de uma interface gráfica, uma câmara, ou até mesmo um reprodutor de áudio espacial e são organizados numa hierarquia totalmente livre onde existe um nó principal por cena, que por sua vez pode ter vários outros nós associados e cada um desses nós pode ter outros, formando uma estrutura em árvore que possibilita a organização eficiente e escalável de todos os elementos do projeto.

Os sinais, o principal método de comunicação deste sistema, consistem em mensagens que possuem nós emissores e recetores claramente definidos e que podem transportar qualquer tipo de informação. Este mecanismo elimina dependências complexas, promovendo um desenvolvimento mais intuitivo e escalável.

Em termos de programação, o *Godot* oferece suporte ao *GDscript* e à *GDExtension*, combinando produtividade e desempenho. O *GDscript*, uma linguagem interpretada e integrada no motor de jogos, é ideal para desenvolver rapidamente protótipos e funcionalidades menos exigentes. Por sua vez, a *GDExtension* permite programar diretamente em *C++*, sendo ideal para as partes do projeto que exijam alta performance, uma vez que é uma linguagem compilada. O sistema *GDExtension* facilita a expansão da funcionalidade do *Godot* e a integração de bibliotecas externas, o que permite aproveitar o melhor dos dois mundos: facilidade de programação em *GDscript* e alto desempenho com *C++*.

A facilidade de exportação no Godot é um dos seus maiores trunfos. A *engine* permite compilar projetos diretamente para uma ampla variedade de plataformas, incluindo Windows, macOS, Linux, Android, iOS e navegadores de *internet*. Esta flexibilidade é essencial para tornar as ferramentas desenvolvidas o mais acessíveis possível.

A documentação oficial [35] é uma excelente fonte de aprendizagem. Através de guias, tutoriais e exemplos práticos, é possível adquirir uma compreensão profunda das funcionalidades e ferramentas do *Godot*, desde os conceitos básicos até tópicos mais avançados.

3.3 INTRODUÇÃO À SIMULAÇÃO COMPUTACIONAL DE SISTEMAS DE PARTÍCULAS

Pretende-se agora explorar como montar vários sistemas simplificados que permitam a simulação das três forças abordadas neste projeto, iniciando com um sistema regido pela força gravítica, com apenas duas partículas. Utilizando o método iterativo semi-implícito de Euler, por questões de simplicidade, aplicaremos a força gravítica para modelar as interações entre partículas, permitindo observar com mais detalhes o método de trabalho com o *Godot*. A abordagem passo-a-passo a seguir, deixará claros os princípios básicos que sustentam as simulações de sistemas de partículas.

1. **Criar o projeto:** Abrir o Godot 4, criar um novo projeto (figura 20).

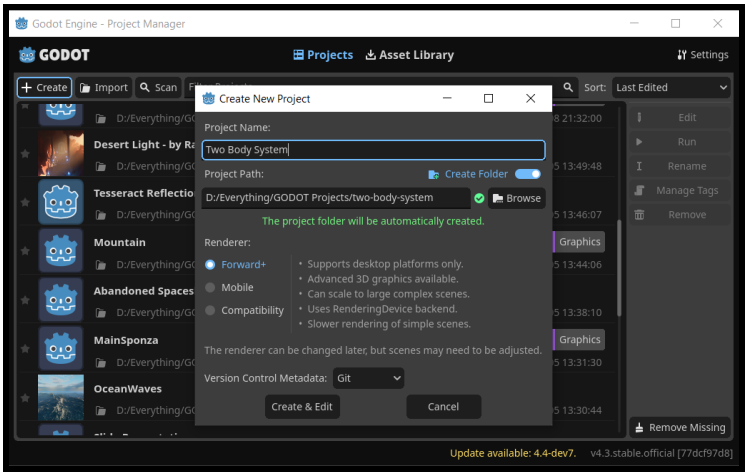

Figura 20: Criação de um projeto com o nome "Two Body System" no *Godot*. Quando um projeto é criado, é possível escolher de entre três métodos de renderização, sendo o "*Forward+*" o mais avançado e completo. O modo *"Compatibility"* é otimizado para aplicações que corram diretamente no browser e o modo *"Mobile"* é direcionado a aplicações móveis

2. **Montagem do cenário:** Existem alguns passos básicos para criar uma cena tridimensional. Para tal, são adicionados alguns nós à cena. O nó *Camera3D* que define o ponto de vista de renderização do cenário, possui várias propriedades como o ângulo de visão, o tipo de projeção (perspetiva ou ortográfica), a exposição da câmara, a abertura da lente, entre outras propriedades. O nó *DirectionalLight3D* irá criar um ponto de luz infinitamente distante, simulando uma fonte luminosa como o Sol. Este tipo de iluminação irá gerar sombras ortogonais. O nó *WorldEnvironment* tem muitas funcionalidades gráficas, sendo a principal, gerar iluminação ambiente para que zonas com sombra/penumbra não apareçam demasiadamente escuras. Estes nós podem ser observados no canto superior esquerdo na figura 21.

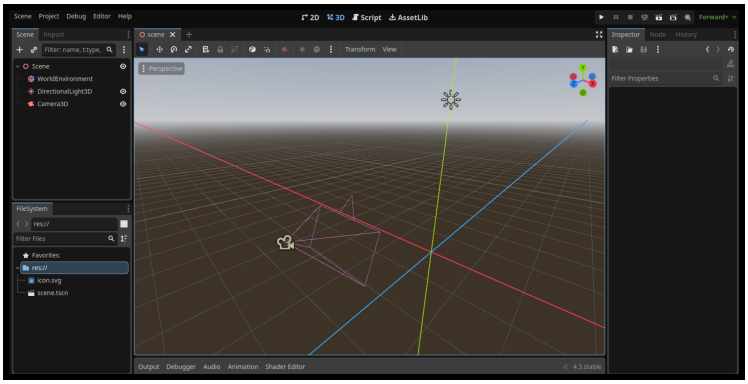

Figura 21: Montagem base de cenário típico tridimensional. Cada nó tem uma função puramente visual. O nó *Camera3D* define o ponto de vista de onde será visualizado o cenário, o nó *DirectionalLight3D* cria um ponto de luz para iluminar objetos e o nó *WorldEnvironment* cria iluminação ambiente.

3. **Adicionar duas partículas:** Adicionar dois nós do tipo *MeshInstance3D* para representar as partículas graficamente. Definir a malha do tipo *Sphere Mesh* para uma aparência esférica e registá-las com os nomes "Body1" e "Body2" (figura 22).

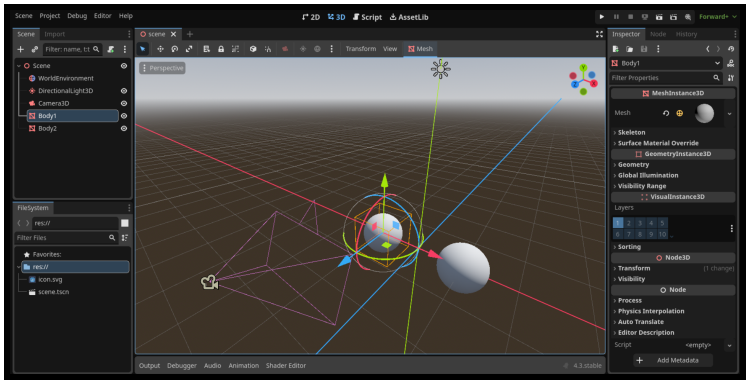

Figura 22: Projeto com duas malhas esféricas para representar duas partículas no espaço.

4. **Criação do *script*:** Criar um novo script no nó principal. A linguagem de programação padrão é o *GDscript*, semelhante a *python* mas adaptado para o *Godot*. A função *_ready()* executa apenas uma vez ao correr a aplicação. A função *_process(delta)* é executada a cada fotograma, normalmente 60 vezes por segundo e delta é o tempo em milissegundos que cada fotograma é mostrado, cerca de 17ms (figura 23).

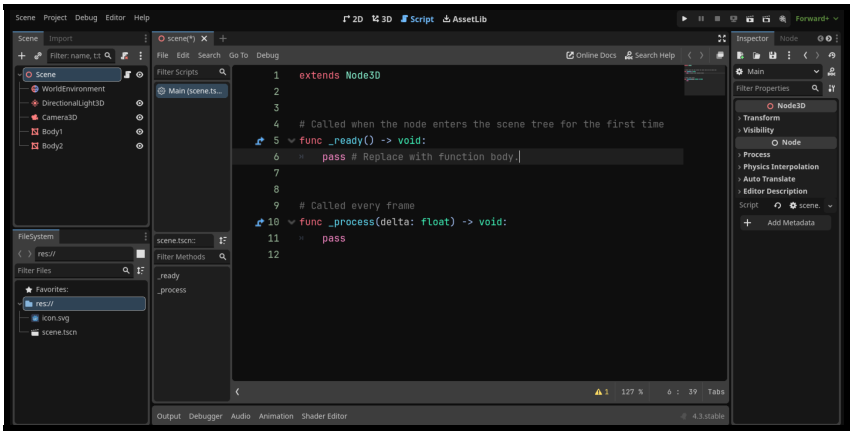

Figura 23: *Script* padrão de um nó do *Godot*. Utiliza a linguagem de programação *GDscript* que é especificamente desenvolvida para a engine.

5. **Inicialização de variáveis:** Um sistema de partículas simulado pressupõe a definição das condições iniciais, tal como mostrado na figura 24. Neste caso com duas partículas:

   - A massa de cada partícula é de 10×10^{10}kg;
   - as velocidades iniciais são de 1m/s, paralelas ao eixo z, com sentidos opostos;
   - as partículas estão posicionadas a 3m de distância entre si, ao longo do eixo x;
   - a constante gravitacional tem o valor $G = 6{,}6743 \times 10^{-11}\,\mathrm{N\,m^2\,kg^{-2}}$.

A definição das variáveis é normalmente feita no cabeçalho do ficheiro de texto. No entanto, a posição das partículas é diretamente aplicada aos nós "*Body1*" e "*Body2*". Isto obriga à definição das posições dentro da função *_ready()* uma vez que já se estão a aplicar transformações diretamente nos nós.

```
extends Node3D

const G : float = 6.6743e-11 # N . m^2 / kg^2

const mass1 : float = 10e10 # kg
const mass2 : float = 10e10 # kg

var velocity1 := Vector3(0, 0,  1) # m/s
var velocity2 := Vector3(0, 0, -1) # m/s

# Called after Body1 and Body2 are initialized
func _ready() -> void:
    # For circular orbit: r = G*m/(4*v^2)
    $Body1.position = Vector3( 1.5, 0, 0) #m
    $Body2.position = Vector3(-1.5, 0, 0) #m
```

Figura 24: Inicialização das variáveis em *GDscript* para a força gravítica.

6. **Lógica de execução cíclica:** A lógica executada a cada fotograma, que pode ser observada na figura 25, serve para mover os corpos para as posições calculadas a cada iteração. Esta lógica começa por calcular a força gravítica entre os corpos, utilizando a Lei da Gravitação Universal: $\vec{F} = G\frac{m_1 m_2}{r^2}\hat{r}$. Posteriormente, é calculada a aceleração com base na 2.ª lei de Newton ($\vec{F} = m\vec{a}$). Com a aceleração, é então aplicado o integrador semi-implícito de Euler para obter a posição segundo a aceleração ($\vec{v}_{t_{n+1}} = \vec{v}_{t_n} + \vec{a}_{t_n} \cdot \Delta t$ e $\vec{r}_{t_{n+1}} = \vec{r}_{t_n} + \vec{v}_{t_{n+1}} \cdot \Delta t$).

```gdscript
# Called every frame
func _process(delta: float) -> void:
    # Force calculation (same for both particles, but simetrical)
    var r_vector : Vector3 = $Body1.position - $Body2.position
    var r_versor : Vector3 = r_vector.normalized()
    var square_distance : float = r_vector.length_squared()
    var force : Vector3 = G * r_versor * mass1 * mass2 / square_distance # N

    # Acceleration calculation
    var acceleration1 : Vector3 = -force / mass1 # m/s^2
    var acceleration2 : Vector3 =  force / mass2 # m/s^2

    # Velocity Calculation
    velocity1 += acceleration1 * delta # m/s
    velocity2 += acceleration2 * delta # m/s

    # Position Calculation
    $Body1.position += velocity1 * delta # m
    $Body2.position += velocity2 * delta # m

```

Figura 25: Execução cíclica, realizada a cada fotograma, de forma a mover as partículas com base na força gravítica.

7. **Executar a aplicação:** Dentro do *Godot*, executar a aplicação e observar o movimento dos corpos. Por questões de visualização, foi ainda desenvolvido um *script* para desenhar um rasto a verde para indicar a trajetória, vetores a azul para indicar as velocidades e vetores a vermelho para indicar as acelerações, tal como se pode observar na figura 26.

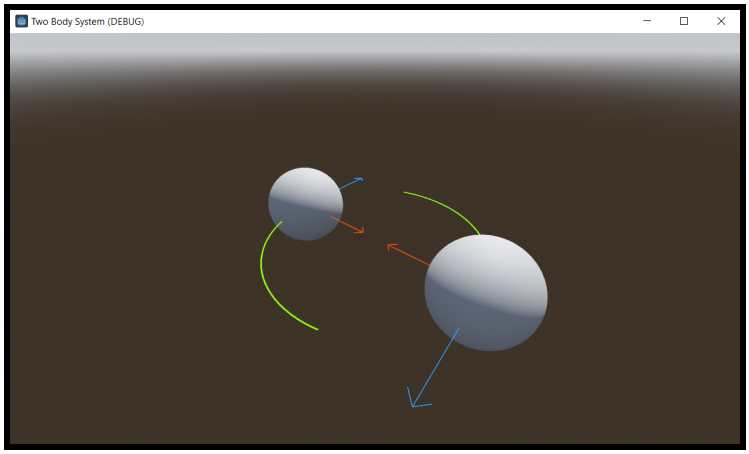

Figura 26: Aplicação criada em execução, mostrando dois corpos em órbita elíptica em torno do seu centro de massa, com um rasto a verde indicando a trajetória, vetores a azul indicando a velocidade e vetores a vermelho indicando a aceleração.

Este exemplo aborda apenas a base de simulação de um sistema de partículas tridimensional. É assim possível observar o quão facilitado fica o desenvolvimento deste tipo de aplicações utilizando motores de jogos, devido à simplicidade na renderização de objetos e ainda pelo facto de conterem poderosas bibliotecas de vetores multidimensionais que facilitam bastante a programação em termos de operações matemáticas geométricas.

É ainda pertinente mostrar mais dois exemplos que incluem a força intermolecular derivada do potencial de Lennard-Jonnes e a força eletromagnética. Relativamente ao exemplo anterior, os primeiros passos, até ao ponto 5, são idênticos. Partindo desta base, para implementar um sistema regido pela força intermolecular, tomam-se os seguintes passos:

1. **Definição da força:**

   Como já abordado no subsubcapítulo 2.1.3, a força intermolecular derivada do potencial de Lennard-Jonnes é dada pela equação 26,

   $$\vec{F} = 24\epsilon \left[\frac{2\sigma^{12}}{r^{13}} - \frac{\sigma^6}{r^7} \right] \hat{r}. \tag{26}$$

2. **Inicialização das variáveis:** Definição das constantes sigma e epsilon, assim como das massas e vetores de velocidade e posição, como se pode observar na figura 27.

```
extends Node3D

const epsilon : float = 20 # J
const sigma : float = 2 # m (equilibrium distance)

const mass1 : float = 1 # kg
const mass2 : float = 1 # kg

var velocity1 := Vector3.ZERO # m/s
var velocity2 := Vector3.ZERO # m/s

# Called after Body1 and Body2 are initialized
func _ready() -> void:
    $Body1.position = Vector3( 1.5, 0, 0) #m
    $Body2.position = Vector3(-1.5, 0, 0) #m
```

Figura 27: Inicialização das variáveis em *GDscript* para o potencial de Lennard-Jonnes.

3. **Lógica de execução cíclica:** A lógica de execução a cada fotograma é semelhante à da força gravítica, onde se utiliza a 2.ª lei de Newton e o integrador semi-implícito de Euler. No entanto a definição da força será calculada segundo a equação derivada do potencial de Lennard-Jonnes, tal como se pode observar na figura 28.

```gdscript
# Called every frame
func _process(delta: float) -> void:

    # Force calculation (same for both particles, but simetrical)
    var r_vector : Vector3 = $Body2.position - $Body1.position
    var r_versor : Vector3 = r_vector.normalized()
    var distance : float = r_vector.length()
    var force : Vector3 = r_versor * 12 * epsilon * (
                    (sigma**12 / distance**13)- (sigma**6 / distance**7))

    # Acceleration calculation
    var acceleration1 : Vector3 = -force / mass1 # m/s^2
    var acceleration2 : Vector3 =  force / mass2 # m/s^2

    # Velocity Calculation
    velocity1 += acceleration1 * delta # m/s
    velocity2 += acceleration2 * delta # m/s

    # Position Calculation
    $Body1.position += velocity1 * delta # m
    $Body2.position += velocity2 * delta # m

    $Extras.acc1 = acceleration1
    $Extras.acc2 = acceleration2
```

Figura 28: Execução cíclica, realizada a cada fotograma, de forma a mover as partículas com base na força de Lennard-Jonnes

4. **Executar a aplicação:** Dentro do *Godot*, executar a aplicação e observar o movimento dos corpos. Foi ainda criado um *script* que gera um rasto a verde para cada partícula. Para possibilitar a visualização da trajetória dos corpos ao longo do tempo, ambos têm uma velocidade constante no eixo y, para assim gerar a trajetória visível na figura 29.

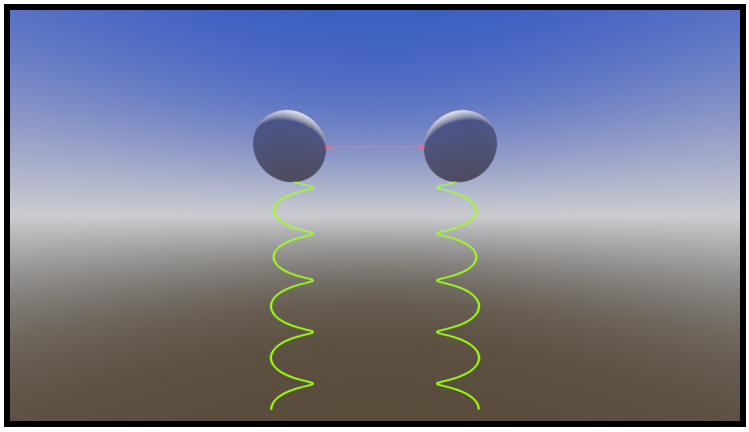

Figura 29: Aplicação criada em execução, mostrando dois corpos em oscilação devido ao potencial de Lennard-Jonnes, com um rasto a verde indicando a trajetória. Para deixar clara a trajetória de ambos ao longo do tempo, os corpos movem-se a uma velocidade constante no eixo y.

5. **Diminuição da energia do sistema:** Caso se deseje colocar as partículas em repouso no seu estado de equilíbrio, ou seja, numa situação onde a temperatura do sistema é nula, altera-se a equação da velocidade no integrador semi-implícito de Euler tal como se pode observar na equação 27,

$$v = v \cdot (1 - \gamma \cdot \Delta t) + a \cdot \Delta t. \tag{27}$$

O parâmetro γ é um fator de amortecimento de unidade s^{-1}, cujo propósito é definir a taxa de redução da velocidade das partículas a cada iteração. As alterações ao código podem ser observadas na figura 30. Com $\gamma = 2s^{-1}$, a trajetória das partículas será a representada na figura 31.

```
# Velocity Calculation (with damping γ = 2 s-1)
var gamma = 2 # s-1
velocity1 = velocity1 * (1 - gamma * delta) + acceleration1 * delta # m/s
velocity2 = velocity2 * (1 - gamma * delta) + acceleration2 * delta # m/s
```

Figura 30: Bloco de código da equação da velocidade com um fator de amortecimento $\gamma = 2s^{-1}$, para a força de Lennard-Jonnes.

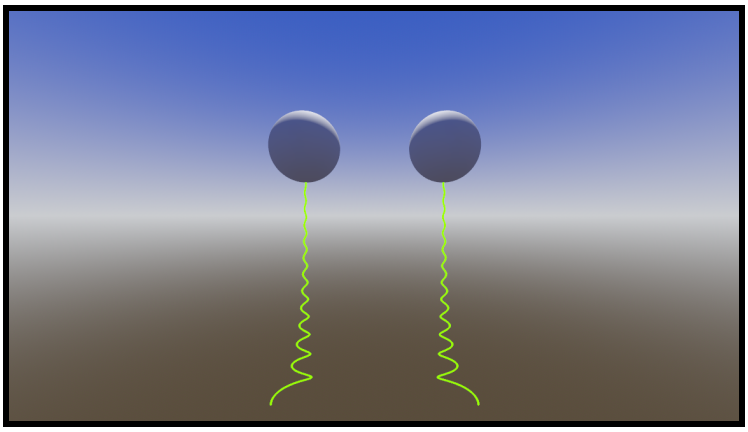

Figura 31: Aplicação criada em execução, mostrando dois corpos em oscilação devido ao potencial de Lennard-Jonnes, com um fator de amortecimento aplicado $\gamma = 2s^{-1}$.

Para simular a força eletromagnética e observar os seus efeitos, será interessante simular um fio percorrido por uma corrente elétrica para daí observar a força magnética a atuar sobre uma partícula de teste. Para tal, o código tem de ser reescrito para possibilitar a criação de múltiplas partículas de forma facilitada. A primeira demonstração irá focar-se exclusivamente na força elétrica.

1. **Inicialização das variáveis:** Definição das constantes ϵ_0 e μ_0, para daí obter as constantes utilizadas nas equações de Biôt-Savart e Coulomb como se pode observar na figura 32.
2. **Função para criar e armazenar partículas:** Para armazenar as informações de cada partícula, uma lista de dicionários como o nome "*particles*" é criada. Cada dicionário armazena as informações de cada partícula, nomeadamente, a sua posição, velocidade, aceleração, massa, carga e um valor booleano que indica se as partículas podem acelerar. Além disso, armazenam também um ponteiro para a malha da partícula, utilizado para atualizar a posição da malha. Para facilitar a criação de partículas e preenchimento desta base de dados utilizando apenas código, foi desenvolvida uma função com o nome "*add_particle*" que gera uma partícula com uma posição, carga e velocidade definidas. Além disso é possível

```
# Vacuum permittivity
const e0 : float = 8.8541878176e-12; # F/m
# Vacuum permeability
const u0 : float = 4 * PI * 1e-7; # H/m
# Coulomb constant
const ke : float = 1 / (4 * PI * e0); # N.m2/C2
# Biot-Savart Constant
const kb : float = u0 / (4 * PI); # H/m
```

Figura 32: Definição das constantes para o campo eletromagnético.

definir se a partícula pode ou não acelerar, o que é importante em determinadas simulações. Esta função começa por criar uma malha esférica, com uma cor definida pela carga da partícula, por questões de visualização, preenchendo de seguida um dicionário com as características da partícula e inserindo-o na lista. É de notar que todas as partículas têm uma massa de 0.005 kg por questões de simplicidade. Abaixo encontra-se a implementação em *GDscript*.

```
1     # List each particle information
2     var particles : Array[Dictionary]
3
4     # Function to easily create particles
5     func add_particle(pos := Vector3.ZERO, charge : float = 0,
6       velocity = Vector3.ZERO, can_accelerate : bool = true) -> void:
7
8         var sphere = SphereMesh.new()
9         sphere.radial_segments = 32
10        sphere.rings = 16
11        sphere.height = 0.8
12        sphere.radius = 0.4
13
14        var material = ORMMaterial3D.new()
15        if charge < 0:
16            material.albedo_color = Color(0.132, 0.222, 1)
17        elif charge > 0:
18            material.albedo_color = Color(1, 0.376, 0.223)
19
20        var mesh_instance = MeshInstance3D.new()
21        mesh_instance.material_override = material
22        mesh_instance.mesh = sphere
23        mesh_instance.position = pos
24        add_child(mesh_instance)
25
26        var particle_dic : Dictionary = {
27            Mesh = mesh_instance, Position = pos,
28            Velocity = velocity, Acceleration = Vector3.ZERO,
29            Charge = charge, CanAccelerate = can_accelerate, Mass =
                  0.0005 #kg
30        }
31
```

```
32          particles.append(particle_dic)
```

3. **Configurar partículas para simular o campo elétrico:** Neste primeiro exemplo foram simulados três pares de partículas, servindo cada par para mostrar a atração entre partículas com cargas de sinal oposto e a repulsão de partículas com cargas do mesmo sinal. O código para definir as partículas encontra-se na figura 33.

```
func _ready() -> void:
    add_particle(Vector3(-4, 8, 0),  1E-5) # P
    add_particle(Vector3( 4, 8, 0), -1E-5) # N

    add_particle(Vector3(-4, 4, 0),  1E-5) # P
    add_particle(Vector3( 4, 4, 0),  1E-5) # P

    add_particle(Vector3(-4, 0, 0), -1E-5) # N
    add_particle(Vector3( 4, 0, 0), -1E-5) # N
```

Figura 33: Definição de três pares de partículas para mostrar a simulação das interações segundo o campo elétrico. Os argumentos da função *add_particle* aqui definidos são, por ordem, a posição e a carga de cada partícula

4. **Lógica de execução cíclica:** Para calcular a força exercida em cada partícula, gerada por todas as outras, é necessário utilizar dois ciclos *for* encadeados. O ciclo externo irá primeiramente verificar se uma dada partícula pode acelerar, e se sim, irá então calcular a aceleração a partir do ciclo *for* interno, que é um somatório da aceleração gerada por todas as outras partículas. Com esta aceleração é calculada a velocidade e finalmente a posição utilizando o método semi-implícito de Euler. O cálculo da aceleração é obtido a partir da força de Lorenz, que por sua vez é obtida do cálculo dos campos elétrico e magnético. Toda esta lógica pode ser observada em *GDscript* no bloco de código abaixo.

```
1       # Called every frame
2       func _process(delta: float) -> void:
3           # Acceleration calculation
4           for p1 in particles:
5               if p1.CanAccelerate == false: continue
6               p1.Acceleration = Vector3.ZERO
7
8               for p2 in particles:
9                   if p1 == p2: continue
10
11                  var vector_r : Vector3 = p1.Position - p2.Position
12                  var versor_r : Vector3 = vector_r.normalized()
13                  var dist2 : float = vector_r.length_squared()
14
15                  var electric_field : Vector3 = ke * p2.Charge *
                        versor_r / dist2 # V/m
16                  var magnetic_field : Vector3 = kb * p2.Charge * (
17                  p2.Velocity.cross(versor_r)) / dist2 # T
18
19                  # Lorenz Force = qE + qv x B
20                  var Fe : Vector3 = p1.Charge * electric_field # N
```

```
21                  var Fb : Vector3 = p1.Charge *
                        (p1.Velocity.cross(magnetic_field)) # N
22
23                  # Velocity Integration step
24                  # Perfoming this step directly reduces float precison
                        errors
25                  var acceleration : Vector3 = (Fe+Fb) / p1.Mass # m/s2
26                  p1.Velocity += acceleration * delta # m/s
27
28          # Update Positions after all velocities are calculated
29          for p1 in particles:
30              p1.Position += p1.Velocity * delta # m
31              p1.Mesh.position = p1.Position # m
```

5. **Executar a aplicação:** Executando a aplicação dentro do *Godot*, observar o movimento dos corpos, cujas forças e velocidades calculadas na simulação estão representadas na figura 34. Vale a pena notar que cada par foi simulado individualmente para evitar interações entre os mesmos.

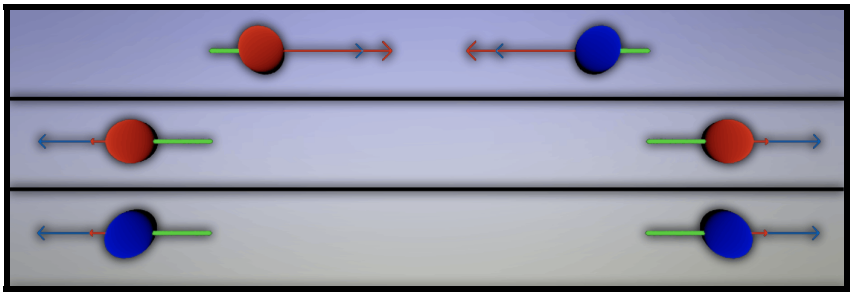

Figura 34: Simulação dos três pares de partículas. Cada par mostra a atração entre partículas com cargas de sinal oposto e a repulsão de partículas com cargas do mesmo sinal. Cada par foi simulado individualmente para evitar interações indesejáveis, sendo esta imagem uma montagem que une as três simulações.

6. **Configurar partículas para o campo magnético:** Toda a complexidade adicionada na implementação do campo eletromagnético é justificada com a simulação do campo magnético. A razão está na necessidade de criar um componente que simule um fio com uma corrente elétrica constante, que irá criar um campo magnético girante ao seu redor. Para tal foram posicionadas 1000 partículas positivas com 100 Coulomb ao longo do eixo z com velocidade nula e outras 1000 partículas com carga negativa com -100 Coulomb também ao longo do eixo z, mas a uma velocidade de 20 m/s. As cargas positivas simulam a matriz de protões num condutor metálico e as partículas negativas representam o movimento de eletrões no interior do condutor que irão gerar uma corrente elétrica. As partículas estão espaçadas 1 metro entre si. A corrente neste condutor é de: $I = C/s = 100C/m \cdot 20m/s = 2000$ Ampere.

   Todas as partículas que constituem o condutor foram criadas com o parâmetro "*can_accelerate*" a falso, caso contrário o condutor seria desintegrado no momento da simulação devido às forças internas. É criada ainda uma partícula fora deste condutor, que pode acelerar e cuja trajetória será objeto de estudo para observar o campo magnético. A definição desta e restantes partículas pode ser observada na figura 35.

```
func _ready() -> void:
    # Create wire with 1000 positive and 1000 negative charges
    for i in 1000:
        add_particle(Vector3(0, 0, -500+i),  100, Vector3(0,0,0),  false)
        add_particle(Vector3(0, 0, -500+i), -100, Vector3(0,0,20), false)

    # Create charge to interact with the wire
    add_particle(Vector3(5, 0, 0), -1, Vector3(0, 0, 10))
```

Figura 35: Criação de um condutor elétrico e partícula de teste. As partículas que definem o condutor são criadas através de um ciclo *for*. É ainda criada uma partícula individual para observar a sua trajetória. Os argumentos da função *add_particle* aqui definidos são, por ordem, a posição, a carga, a velocidade e a possibilidade de aceleração de cada partícula.

7. **Testar a aplicação para o campo magnético:** Para testar esta simulação o campo elétrico não foi considerado. Idealmente, uma vez que o condutor é eletricamente neutro, não deveria existir força elétrica nesta simulação, no entanto, uma vez que o condutor é constituído por cargas enormes (relativamente a protões e eletrões) e considerando ainda os erros de precisão, o campo elétrico é sempre significativo. Este efeito indesejável é especialmente agravado pelo facto do campo elétrico gerado por uma carga ser sempre superior em magnitude ao campo magnético gerado pela mesma carga, mesmo para velocidades elevadas. Segue-se a seguinte explicação.

   Para uma carga pontual, a magnitude do campo elétrico é dada pela equação 28

$$E = \frac{q}{4\pi\epsilon_0 r^2}, \tag{28}$$

   enquanto que a magnitude do campo magnético gerado pela mesma carga é dado pela equação 29, onde para garantir as condições que permitem o campo magnético máximo, considera-se o cenário impossível onde a partícula se moveria à velocidade da luz.

$$B = \frac{\mu_0 q c}{4\pi r^2}. \tag{29}$$

   Sabendo que a velocidade da luz $c = \frac{1}{\sqrt{\mu_0 \epsilon_0}}$, a proporção dos campos elétrico e magnético será dada pela na equação 30,

$$\frac{E}{B} = \frac{\frac{q}{4\pi\epsilon_0 r^2}}{\frac{\mu_0 q c}{4\pi r^2}} = \frac{1}{\mu_0 \epsilon_0 c} = c, \tag{30}$$

   demonstrando desta forma que mesmo para o cenário impossível que maximiza o campo magnético gerado por uma partícula, ainda assim o campo elétrico será c ($3 \times 10^8\,\mathrm{m\,s^{-1}}$) vezes superior ao campo magnético, uma diferença substancial.

   Por esta razão, cenários onde se pretende ver a influência do campo magnético, o campo elétrico deve ser desativado ou irá dominar o comportamento das partículas. A figura 36 mostra a atração e repulsão em dois cenários distintos gerados unicamente pelo campo magnético.

8. **Trajetórias helicoidais:** Uma vez que esta simulação é já bastante completa, é possível simular trajetórias ainda mais complexas. No exemplo a seguir, a carga

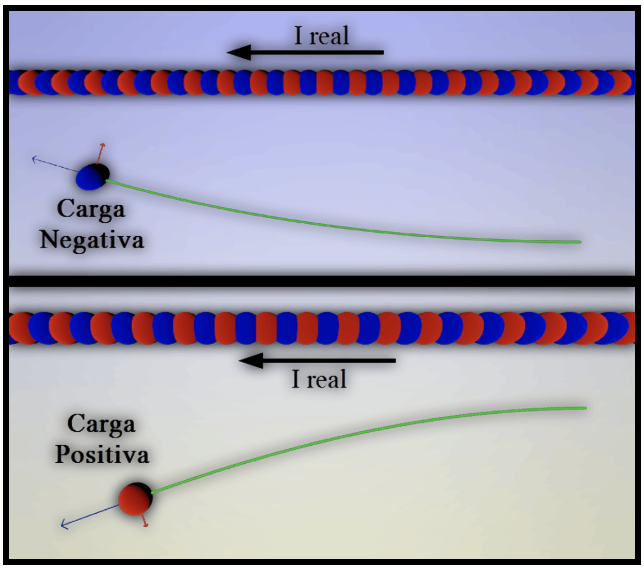

Figura 36: Trajetória de partículas sob um campo magnético. Aqui a carga da partícula e o sentido real da corrente elétrica definem se a força magnética será atrativa ou repulsiva.

da partícula de teste foi aumentada consideravelmente, de forma que a mesma fizesse uma pequena trajetória circular perto do condutor. Aplicando ainda uma força no eixo y para colocar a partícula a orbitar o condutor, é possível observar uma trajetória helicoidal dupla (como do filamento de tungsténio numa lâmpada incandescente), tal como se pode observar na figura 37.

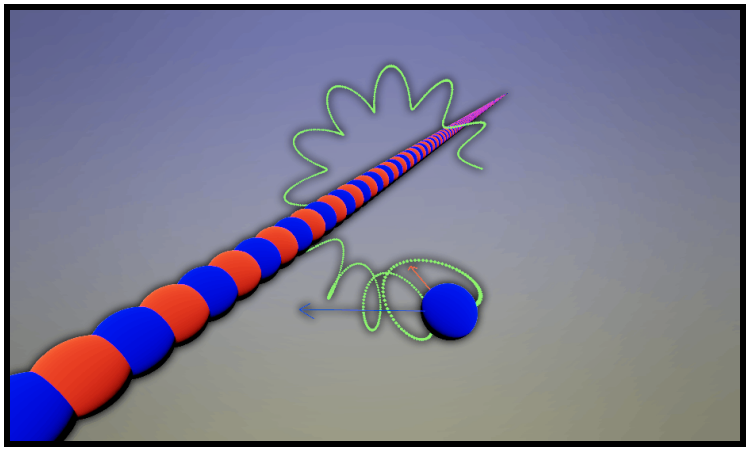

Figura 37: Trajetória de uma partícula em torno de um condutor com forte campo magnético. Aplicando uma velocidade inicial à partícula de teste, é possível observar uma trajetória helicoidal dupla.

Um campo magnético planar fará uma carga elétrica, com velocidade puramente perpendicular, rodar em torno de um ponto (chamado centro guia). Uma vez que o campo magnético gerado por um condutor com corrente elétrica é cilíndrico, a partícula irá seguir uma trajetória circular, tanto em torno do condutor, como de uma linha imaginária helicoidal com centro no condutor, desde que tenha uma componente perpendicular e paralela à direção da corrente.

Este fenómeno é a razão pela qual o campo magnético da terra blinda a superfície terrestre contra as partículas emitidas pelos ventos solares. Estas partículas ficam presas ao campo magnético da Terra (chamadas de *"trapped particles"*) e seguem as linhas de campo magnético até por fim atingirem os polos terrestres, originando as auroras boreais. Uma representação deste fenómeno pode ser observada na figura 38 [36].

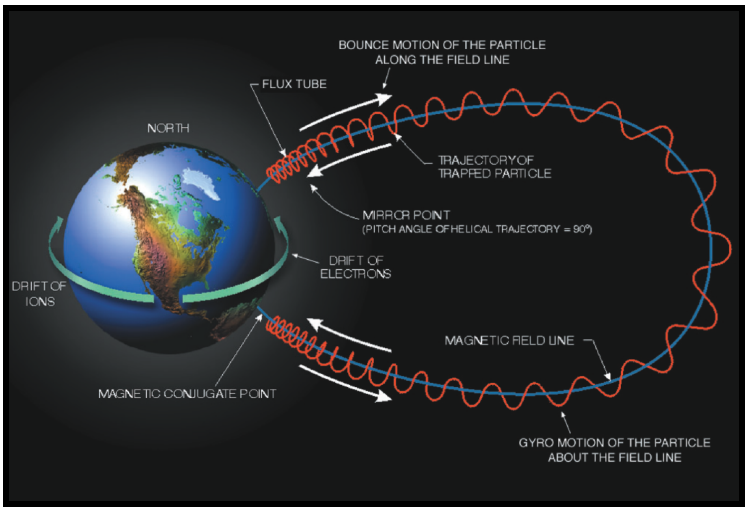

Figura 38: Imagem ilustrativa de partículas presas ao campo magnético da terra, gerando fenómenos como a blindagem natural da Terra contra os ventos solares e das auroras boreais.

Fica assim encerrado este capítulo, com a descrição destas simulações, as quais demonstram e clarificam o funcionamento intrínseco do *Force Fields*. O capítulo seguinte dedicar-se-á à descrição do uso, do ponto de vista do utilizador, do *Force Fields*, que como um todo permite realizar e analisar estas e outras simulações através de interfaces gráficas desenvolvidas para este propósito.

4

FUNCIONALIDADES E CAPACIDADES DO *FORCE FIELDS*

Este capítulo visa explicar detalhadamente a forma de utilizar o *Force Fields*, uma vez que este é o produto final da dissertação e deve, por isso, ser analisado em pormenor. Ao longo desta descrição são detalhadas as diversas funcionalidades e características que o compõem.

4.1 MENU PRINCIPAL

Após a execução do *software*, o utilizador será direcionado para o menu principal, o qual pode ser observado na figura 39, onde poderá selecionar uma de duas secções principais, "Scenarios" e "FreeStyle", para começar a interação com o simulador.

Figura 39: Menu principal do Force Fields. Disponibiliza a secção de cenários preparados "Scenarios" e a secção livre "FreeStyle" onde o utilizador poderá construir o seu próprio cenário.

A secção "Scenarios", cujo menu pode ser observado na figura 40, disponibiliza ao utilizador uma biblioteca de cenários preparados com o intuito de mostrar facilmente uma série de fenómenos físicos. Estes cenários poderão ser o sistema solar, um sistema de dois ou três corpos ou até mesmo um sistema de partículas eletricamente carregadas, que evidencie facilmente a lei de Coulomb ou a indução magnética gerada por uma corrente elétrica.

A secção "FreeStyle" é a mais importante, uma vez que dá ao utilizador total liberdade de criação, podendo construir sistemas de partículas por meio de várias ferramentas disponibilizadas pelo *Force Fields*. O utilizador deve, nesta janela, escolher qual o simulador a utilizar, com base na força gravitacional, na força intermolecular derivada do potencial de Lennard-Jones ou na força eletromagnética, tal como se pode observar na figura 41. Os nomes dos simuladores dão ênfase ao facto de serem tridimensionais (3D), porque um dos objetivos futuros

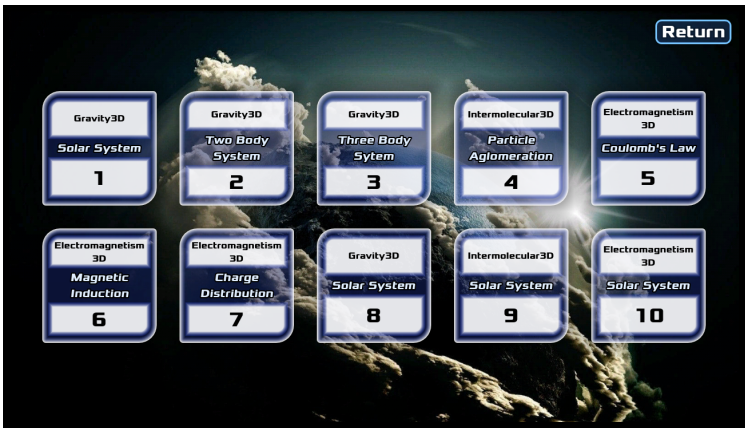

Figura 40: Secção "Scenarios", onde se poderá escolher de entre vários cenários preparados que permitem executar uma simulação específica rapidamente.

deste projeto será adicionar motores de simulação bidimensionais (2D) para um melhor desempenho e facilidade de utilização em cenários que não necessitem das três dimensões. A escolha de desenvolver os simuladores em 3D vem do simples facto de não fazer sentido simular a lei de Biot-Savart em 2D, uma vez que as forças magnéticas geradas por um sistema de partículas contido num plano serão sempre perpendiculares, ou nulas, ao próprio plano. Por esta razão, todo o desenvolvimento do projeto se baseou na simulação 3D.

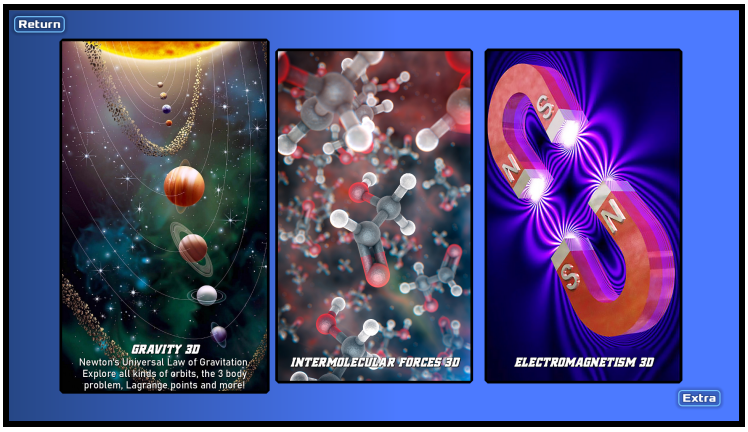

Figura 41: Secção *FreeStyle*, onde se pode escolher o simulador de partículas governado pela força gravítica, intermolecular ou eletromagnética.

4.2 INTERFACE GRÁFICA EM AMBIENTE DE SIMULAÇÃO

Após selecionar um dos três simuladores disponíveis, o *Force Fields* disponibiliza ao utilizador o principal cenário de criação de simulações, neste caso direcionado à força gravítica, tal como se pode observar na figura 42. O cenário é composto por um ambiente tridimensional, onde se podem facilmente visualizar as simulações calculadas em tempo real. Sobreposto a este está uma interface gráfica bidimensional semitransparente, com o propósito de não ocultar o ambiente de simulação em demasia. Além disso, as abas esquerda e direita podem ser ajustadas de

forma a ocuparem mais ou menos espaço do ecrã ou até mesmo serem totalmente ocultadas.

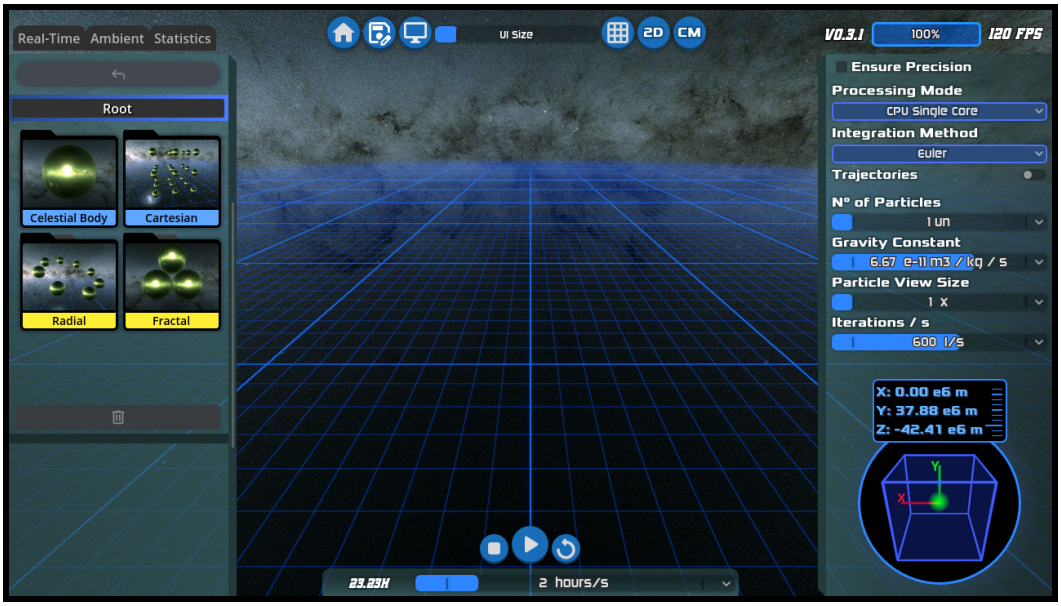

Figura 42: Interface gráfica do ambiente de simulação do Force Fields. Pode ser ajustada pelo utilizador e possui várias ferramentas para adicionar recursos ao ambiente 3D, alterar as suas propriedades, ajustar parâmetros de simulação e obter informações em tempo real.

Cada componente da interface gráfica e do ambiente de simulação foram desenvolvidos especificamente para as necessidades do *Force Fields*:

- O cenário tridimensional é completamente interativo, permitindo ao utilizador movimentar-se livremente e ainda interagir com partículas e objetos, alterando as suas posições e velocidades.
- O sistema de manipulação temporal, na parte inferior do ecrã, permite ao utilizador parar, reiniciar, iniciar a simulação e ainda alterar a escala de tempo, com uma precisão que vai dos picossegundos até às décadas.
- A biblioteca de recursos, à esquerda, possui ferramentas para adicionar elementos simuláveis ao cenário e alterar as suas propriedades.
- A aba de configurações, à direita, permite ao utilizador alterar parâmetros como constantes físicas, o tipo de integrador desejado, o modo de processamento, visualização de trajetórias, entre outras configurações.
- A barra de ferramentas, no topo, permite ao utilizador gravar em disco o cenário construído, esconder ou mostrar a grelha tridimensional, visualizar o centro de massa do sistema, a versão do *software*, as iterações calculadas por segundo e ainda outras funcionalidades úteis como alterar as dimensões da interface gráfica.
- No canto superior esquerdo do ecrã, existem ainda algumas abas ("Real-Time", "Ambient", "Statistics") que permitem mostrar janelas de configurações, para assim obter dados estatísticos em tempo real e ainda alterar o aspeto visual do simulador.

- Na aba estatísticas é possível ativar um gráfico capaz de mostrar a evolução temporal da energia potencial e cinética do sistema para um dado cenário tal como se pode observar na figura 43. Neste caso, as curvas mostradas foram obtidas de um sistema de dois corpos com trajetórias elípticas regidos pela força gravítica. O gráfico é interativo e é gerado em tempo real no decorrer da simulação.

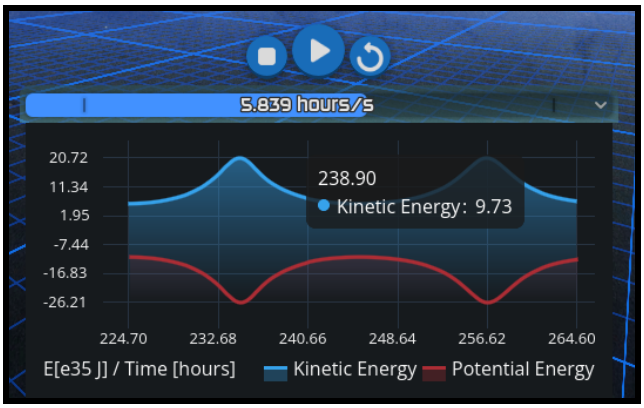

Figura 43: Gráfico integrado no *Force Fields* para a energia cinética e potencial ao longo do tempo.

4.3 SIMULAÇÃO EM TEMPO REAL COM ESCALA DE TEMPO VARIÁVEL

O ambiente 3D está inerentemente divido em dois modos principais: um modo de edição, para definir as condições iniciais do sistema por meio de várias ferramentas, e um modo de simulação, onde o utilizador observa o desenvolvimento do sistema de partículas criado. Um exemplo de simulação / edição pode ser observado na figura 44.

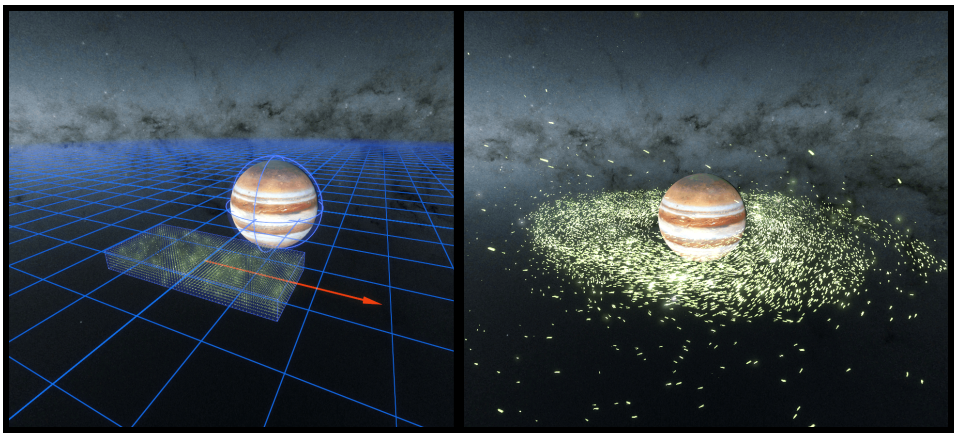

Figura 44: Comparação do modo de edição (à esquerda) com o modo de simulação (à direita) no *Force Fields*.

Para alternar entre modos, basta utilizar o painel de controlo temporal que se pode observar ao centro, no inferior da figura 42), onde o utilizador pode iniciar a simulação, saindo automaticamente do modo de edição. Para voltar a editar as

condições iniciais do sistema, basta parar a simulação. Enquanto a simulação é executada, existe um indicador que informa o tempo passado desde o início da simulação.

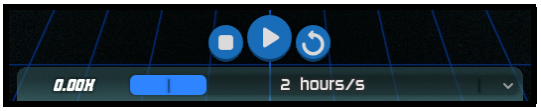

Figura 45: Painel de controlo temporal que permite iniciar, pausar, parar, reiniciar e até alterar a escala de tempo da simulação.

É possível também pausar a simulação para, de certa forma, congelar o sistema e daí tirar informações desse mesmo instante. Este estado, como todos os outros, permite movimentar a câmara livremente, sendo útil para observar o sistema de partículas num dado instante, através de diversos pontos de vista.

Além disso, é possível também reiniciar a simulação com um único clique. Esta função é útil quando o utilizador pretende modificar certas características do cenário e quer ver de forma rápida alterações na simulação. Estas características podem ser, por exemplo, o tipo de integrador utilizado, o número de iterações por segundo, entre outras.

Outra função muito relevante para um simulador versátil é a possibilidade da alteração da escala de tempo da simulação. Para tal existe um controlo deslizante, tipicamente chamado *slider*, que permite variar o parâmetro *delta* (intervalo de tempo entre iterações) utilizado nos integradores disponibilizados. O parâmetro *delta* é proporcional à taxa de variação do sistema, sendo que aumentar o seu valor irá necessariamente aumentar o tempo simulado relativamente ao tempo real. Por exemplo, um valor *delta* inalterado fará com que a simulação seja executada a 1 segundo (simulado) por segundo (real). Se o valor *delta* for multiplicado por 60, então a simulação será executada a 1 minuto (simulado) por segundo (real). Uma vez que o *Force Fields* lida com simulações que vão desde interações planetárias, a interações moleculares, este recurso é imprescindível. No entanto, vale a pena notar que se o valor *delta* for demasiado elevado e as partículas alterarem a sua trajetória de forma abrupta entre iterações, a exatidão da simulação fica comprometida. O *Force Fields*, por omissão, executa o máximo de iterações que o computador permite. Por essa razão, cabe ao utilizador escolher uma escala de tempo adequada à exatidão desejada nos resultados.

O controlo deslizante aqui presente é um bom exemplo de uma cena criada com um propósito básico, que por sua vez é replicada várias vezes dentro do *Force Fields* para a edição de valores. Esta grande vantagem das cenas instanciáveis, permite evitar a repetição de código, o que leva o programa a ocupar menos memória e se o programador desejar fazer uma alteração neste componente, todas as instâncias sofrerão as alterações imediatamente. Os controlos deslizantes são o principal componente para inserir valores no programa e uma vez que esta é uma ferramenta bastante utilizada, é desenvolvida para ser o mais dinâmica e útil possível. As principais características são:

1. Caso o utilizador exceda os valores mínimo ou máximo, os mesmos ajustam-se para permitir uma escala maior de valores.

2. As unidades de medida alteram-se dinamicamente conforme o valor escolhido e podem também ser escolhidas num seletor lateral, tal como mostra a figura 46.

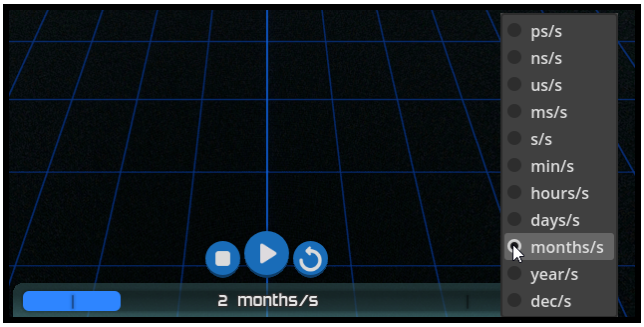

Figura 46: Os *sliders* ou controlos deslizantes desenvolvidos no *Force Fields* são a principal forma de definir os parâmetros dos sistemas de partículas e da simulação.

3. Para uma maior precisão, é também possível inserir o valor desejado manualmente. Para tal é necessário pressionar e largar o botão esquerdo do rato enquanto este se encontra sobre o *slider*. Por outro lado, se o botão esquerdo for pressionado e deslizado na horizontal, o *slider* irá variar o valor em tempo real.
4. A inserção de valores manuais interpreta e resolve expressões matemáticas escritas em texto, como por exemplo "$5 * \sin(\mathrm{PI}/6) + 3.5/4$", funcionando como uma calculadora integrada.

4.4 LOCOMOÇÃO E INTERAÇÃO NUM ESPAÇO TRIDIMENSIONAL

O ambiente tridimensional tem ferramentas de locomoção e interação para que o utilizador se possa deslocar livremente e ainda movimentar partículas e outros objetos. É importante referir que o eixo de coordenadas do *Godot* tem como cota o eixo Y, ao contrário da convenção utilizada em física, o eixo Z. Ou seja, o eixo Y define o movimento para cima e para baixo, o eixo X define o movimento para a esquerda e para a direita e o eixo Z define o movimento para a frente ou para trás. A documentação do Godot relativamente a vetores tridimensionais, apresentada na figura 47 deixa este aspeto claro e pode ser consultada na página oficial de documentação do software. [37]

A movimentação no espaço 3D implementada no *Force Fields* é relativamente simples. Para olhar livremente em todas a direções, ou seja, rodar a câmara, basta manter pressionada a tecla direita do rato e movê-lo livremente, sendo que a translação horizontal do rato é convertida numa rotação da câmara relativamente ao eixo global Y e a translação vertical é convertida numa rotação local no eixo X. Quando a tecla direita do rato não está pressionada, o utilizador pode interagir normalmente com a interface gráfica sem afetar o movimento tridimensional. Para se locomover no espaço, utilizam-se as teclas W para a frente, S para trás, A para a esquerda, D para a direita, Q para baixo e E para cima. Estes movimentos são locais, ou seja, dependem da orientação da câmara. Na prática, movimentos locais

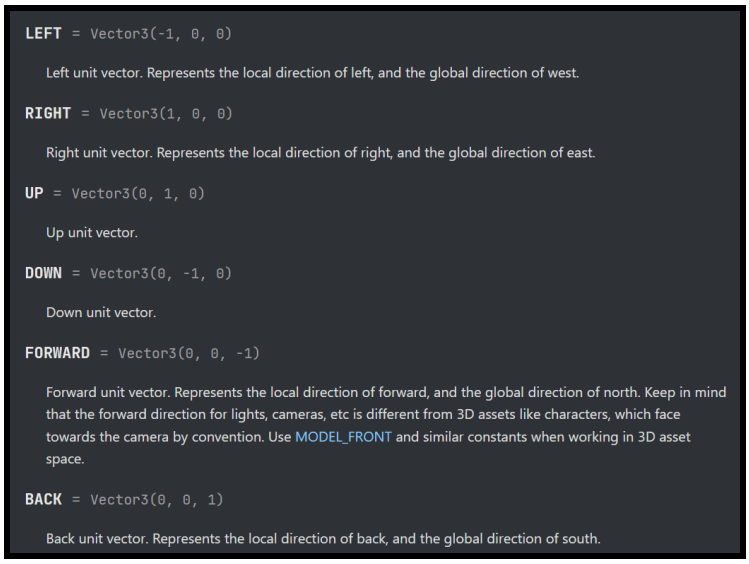

Figura 47: Página oficial de documentação do *Godot*, especificamente no documento que descreve a classe *Vector3*, utilizada para criar e manipular vetores tridimensionais.

são obtidos multiplicando a matriz de rotação 3x3 do objeto ($\mathbf{R}$), com o vetor global do qual se quer obter a transformação local ($\vec{v}_{\text{local}} = \mathbf{R}\vec{v}_{\text{global}}$).

Em certos cenários, onde as partículas são escassas e não existem pontos de referência suficientes no espaço, torna-se difícil a locomoção pelo cenário, pelo que foram implementadas algumas funções para auxiliar neste aspeto. Uma delas é a disponibilização das coordenadas globais do ponto de vista no canto inferior direito do ecrã, perto de um mini-mapa tridimensional, que ajuda o utilizador a orientar-se no espaço. Este mini-mapa, presente na figura 48, mostra a localização dos vários recursos adicionados ao ambiente 3D (representados por pontos brancos) e ainda a localização do próprio utilizador relativamente a esses pontos (ponto verde). Este mapa segue a rotação local da câmara para uma interpretação mais intuitiva e mostra ainda o eixo de coordenadas globais do cenário.

Para finalizar na ajuda à localização no espaço, existe ainda uma grelha (azul) definida no plano $y = 0$ do ambiente tridimensional, visível na figura 42. Esta grelha, já anteriormente mostrada, varia de forma dinâmica conforme a distância da câmara ao plano, ou seja, conforme a câmara se aproxima do plano, as quadrículas subdividem-se em outras 100 (matriz 10x10). Por outro lado, quando a câmara se afasta do plano, conjuntos de 100 quadrículas 10x10 desvanecem-se numa única. Por esta razão, o comprimento de cada quadrícula varia em múltiplos de 10. Este comportamento permite o afastamento e aproximação da câmara, mantendo sempre a grelha com quadrículas bem definidas, independentemente da escala espacial em que se está a trabalhar.

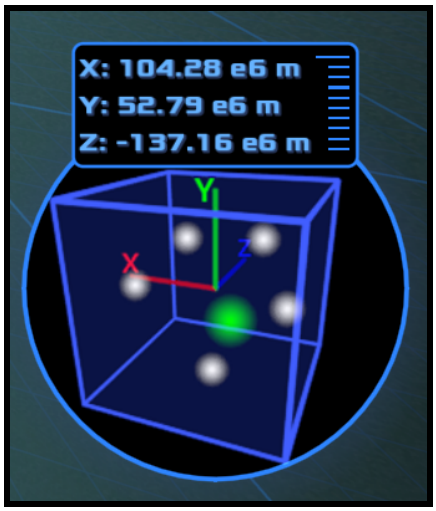

Figura 48: Coordenadas globais da câmara e um mini-mapa tridimensional que mostra a disposição de recursos do ambiente 3D (pontos brancos) e a posição da câmara relativamente a esses recursos (ponto verde).

4.5 SISTEMA DE RECURSOS CONFIGURÁVEIS

Quando se pretende estudar a evolução de um sistema de partículas, o primeiro passo será definir as condições iniciais do problema a ser estudado. Para tal, o *Force Fields* dispõe de uma biblioteca de recursos configuráveis cujo objetivo é construir o sistema de partículas de uma forma rápida e simples. A biblioteca é diretamente acessível a partir da aba esquerda na interface gráfica já mostrada na figura 42.

No *Force Fields*, um recurso define-se como sendo uma ferramenta de distribuição de partículas no espaço tridimensional por meio de um conjunto de regras e características, sendo estas últimas, configuráveis pelo utilizador. Por exemplo, alguns recursos distribuem as partículas num formato matricial, outros num formato radial, outros num formato helicoidal e assim por diante. Deixando clara a distinção entre regras e características, por exemplo numa disposição matricial, irão existir regras que garantem que as partículas realmente sejam dispostas em matriz, mas existem também características configuráveis, que permitem, por exemplo, alterar a separação entre as partículas, ou as suas massas.

A biblioteca de recursos está organizada hierarquicamente em três níveis, da qual um exemplo pode ser observado na figura 49.

- A amarelo, à esquerda da figura, é apresentada uma pasta de recursos cujo objetivo é agrupar recursos com características semelhantes. Neste caso, a pasta apresentada na figura agrupa todos os recursos cuja construção se baseia em coordenadas polares para obter formatos esféricos, circulares ou helicoidais.
- Abrindo uma pasta, revelam-se os recursos, tal como o que se apresenta ao meio a verde. Cada recurso tem uma lógica de implementação única, definida por uma função em linguagem C++ que irá, a partir de certos parâmetros, gerar a distribuição das partículas no espaço tridimensional e também definir

Figura 49: Representação hierárquica de recursos. A figura foi criada com o intuito de mostrar a hierarquia presente na aba de recursos. À esquerda, a amarelo, as pastas de recursos, ao meio, a verde, os recursos, e a azul, à direita, recursos pré-configurados

as características de cada partícula. Por exemplo, neste caso, um dos recursos da pasta "Radial" é o "*Circle*" que distribui de forma equidistante a um ponto central, uma certa quantidade de partículas uniformemente distribuídas num plano.

- Abrindo um recurso, revelam-se as pré-configurações criadas para obter rapidamente uma geometria específica. Por exemplo, neste caso, dentro do recurso "*Circle*", uma das pré-configurações possíveis será o "*Pentagon*" que distribui apenas 5 partículas seguindo as regras mencionadas anteriormente.

Note-se ainda que cada pasta de recursos pode ter um número ilimitado de recursos, assim como estes podem ter um número ilimitado de pré-configurações. Além disso, cada recurso dispõe de uma pré-configuração base, com o nome "*CUSTOM*" (referência à palavra inglesa "*costumizable*") (figura 50) na qual o utilizador poderá editar diretamente todas as características de um dado recurso para obter a distribuição de partículas no espaço desejada.

Figura 50: Todos os recursos têm a versão *"CUSTOM"*, que quando adicionada ao ambiente tridimensional, disponibiliza diretamente ao utilizador diversas configurações para o modificar.

Quando uma pré-configuração "*CUSTOM*" é arrastada com o rato do painel até ao ambiente tridimensional, a distribuição de partículas é imediatamente gerada e o painel esquerdo da interface gráfica é substituído por um painel de configurações. O exemplo presente na figura 51, mostra os parâmetros do recurso que gera uma distribuição helicoidal.

Uma vez que esta é uma das distribuições mais complexas, os seus parâmetros repetem-se para boa parte dos outros recursos, pelo que vale a pena explicar a função de cada um deles:

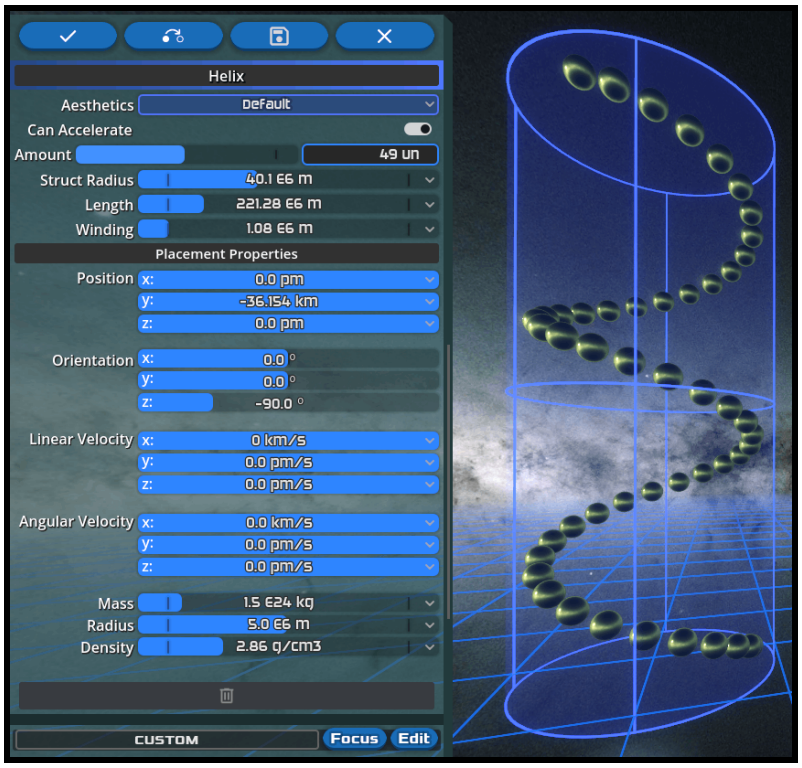

Figura 51: Painel de configurações para um recurso com distribuição helicoidal. Como exemplo, à direita é possível observar a distribuição de partículas gerada para estes parâmetros.

- *Aesthetics (lista)* - Permite alterar a aparência das partículas. Existem 12 possibilidades, sendo 11 destas a aplicação de texturas dos vários planetas do sistema solar (8 planetas), da Lua, de Plutão e do Sol, visíveis na figura 52.

  A última configuração possível, apesar de não ter qualquer textura, torna variável a aparência das partículas nesta configuração. Na simulação gravítica, esta configuração varia a luminosidade de cada partícula proporcionalmente à velocidade com que se movem (visível na figura 44). Na simulação eletromagnética, a partícula será (por convenção) azul tendo carga positiva, vermelha tendo carga negativa, e branca tendo carga nula. Na simulação intermolecular, a cor da partícula varia consoante a escala de pseudo-cores *jetmap* que pretende representar a temperatura (energia cinética) da matéria formada pelas partículas, sendo que partículas em repouso (temperatura absoluta nula) são azuis e com o aumento da temperatura, a cor passa por todas a frequências visíveis como o verde e amarelo, até chegar ao tom vermelho onde a temperatura será teoricamente infinita (a escala de cores é logarítmica). De notar que a aparência em nada irá afetar a simulação. No entanto, é uma ferramenta útil para visualizar certas características, ou apenas tornar uma simulação visualmente mais apelativa.
- *Can Accelerate (valor boleano)* - Desligar esta função inibirá as partículas de acelerarem ou, por outras palavras, de interagirem com outras partículas. Esta definição é útil quando se pretende gerar partículas estáticas ou com

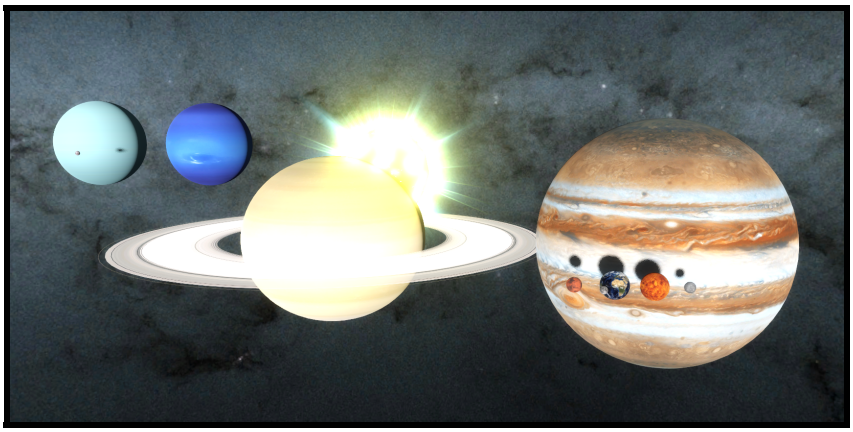

Figura 52: Planetas, Sol, Lua e Plutão em escala real. O Sol apresenta-se significativamente afastado da câmara devido ao seu enorme diâmetro relativamente aos restantes corpos celestes.

uma velocidade constante (definida pela velocidade inicial). Apesar de parecer ser uma função que remove a credibilidade da simulação quando desativada é, no entanto, credível fazer simulações onde se pretenda que um conjunto de partículas tenha uma massa quase infinita, ou considerar que essas partículas fazem parte de uma estrutura rígida.

- *Amount (valor inteiro)* - Este parâmetro vai de 1 a infinito e define a quantidade de partículas que vão formar a estrutura.
- *Struct Radius (valor decimal)* - Define o raio da hélice. Todos os valores decimais têm associada uma certa unidade de medida.
- *Length (valor decimal)* - Define o comprimento da hélice entre o ponto central de ambas as extremidades.
- *Winding (valor decimal)* - Define o encaraculamento/espiralização da hélice, mais especificamente, o número de voltas completas por cada 100 unidades de comprimento da hélice.
- *Position (vetor tridimensional)* - Define a posição global do recurso relativo ao centro de uma das extremidades da hélice. No entanto, para outros recursos a definição da sua posição é normalmente relativa ao seu centro geométrico (que normalmente corresponde também ao centro de massa).
- *Orientation (vetor tridimensional)* - Permite rodar nos três eixos, de -180º a +180º, o recurso como um todo em torno da sua posição de referência.
- *Linear Velocity (vetor tridimensional)* - Define a velocidade de todas as partículas, de forma global e linear. Alterar este valor não afeta a velocidade relativa entre partículas de um mesmo recurso.
- *Angular Velocity (vetor tridimensional)* - Define a velocidade instantânea angular $\vec{\omega}$ do recurso como um todo, ou seja, cada partícula terá uma velocidade $\vec{v}_{total}$ cujo módulo e direção depende do produto vetorial de $\vec{\omega}$ com o vetor deslocamento $\vec{r}$ ao centro geométrico do recurso. De notar que esta velocidade é adicionada à velocidade inicial $\vec{v}_i$ do parâmetro anterior tal como se pode observar na equação 31.

$$\vec{v}_{total} = \vec{v}_i + \vec{\omega} \times \vec{r} \tag{31}$$

- *Mass (valor decimal)* - Define a massa de cada uma das partículas, sendo que a massa total do recurso será este valor multiplicado pelo número total de partículas (*amount*).
- *Radius (valor decimal)* - Define o raio de cada partícula. Este é um parâmetro puramente estético uma vez que em nada influencia o resultado da simulação, quer seja ela gravítica, eletromagnética ou intermolecular.
- *Density (valor decimal)* - Define a densidade de cada partícula. Uma vez que a densidade, massa e raio da partícula são valores interdependentes, estes três controlos deslizantes estão intrinsecamente ligados de forma a que modificar um deles, irá consequentemente variar um dos outros para que todos os valores permaneçam verdadeiros. Vale notar que o valor que não varia será o último que foi editado, ou seja, se uma partícula for definida com uma massa de $10kg$, e uma densidade de $3g/cm^3$, então o raio da partícula será automaticamente ajustado para $9.3cm$. Estes valores são simplesmente calculados seguindo a fórmula do volume da esfera.

4.6 SIMULAÇÃO DE UM NÚMERO ELEVADO DE PARTÍCULAS COM RECURSO À GPU E A *multithreading* DA CPU

O método de execução responsável pela simulação no *Force Fields* foi reescrito de três formas diferentes para permitir alternativas ao modo como o computador processa as iterações. Apesar de tecnicamente serem bastante distintos, os três métodos geram exatamente o mesmo resultado, não tendo qualquer impacto na análise quantitativa das simulações. A diferença entre eles está na compatibilidade e desempenho de cálculo.

O primeiro método, com o nome "*CPU Single Core*" foi escrito de forma a ser o mais compatível e simples possível, onde as iterações da simulação são calculadas na CPU, utilizando apenas uma linha de execução. Esta é a forma padrão em que código escrito em C++ será executado, não sendo necessário qualquer cuidado especial.

O segundo método, com o nome "*CPU Multithreading*", também calcula as iterações na CPU, mas utiliza todos os núcleos (e linhas de execução) disponíveis que o computador tem a oferecer. O *Force Fields* utiliza a biblioteca "*thread*" em C++ para obter o número de linhas de execução (thread::hardware_concurrency()) e para executar blocos de código em várias linhas de processamento simultaneamente (vector<std::thread>.emplace_back()). Para tirar partido desta vantagem e executar a simulação com uma velocidade superior, o sistema de partículas é automaticamente dividido pelo número total de linhas de execução disponíveis. Cada linha de execução calcula a posição futura de um conjunto de partículas, sendo que no final o resultado de todas é unido num único vetor com as informações da simulação. A CPU do computador utilizado possui 12 linhas de execução simultânea, o que permite uma melhoria substancial no desempenho do programa.

O terceiro método, com o nome "*GPU Compute*" utiliza a placa gráfica (GPU) cuja arquitetura é desenhada para executar dezenas de milhares de operações similares em paralelo. Para tirar proveito da placa gráfica, os dados presentes na memória são convertidos para um formato reconhecível pela linguagem de programação OpenGL Shading Language (GLSL) e enviados para a memória da placa gráfica (VRAM). A partir deste passo, é necessário definir vários parâmetros dos quais a especificidade técnica foge do objetivo deste capítulo. No entanto, de forma simplificada, o código que se pretende executar na placa gráfica é escrito num ficheiro à parte em formato GLSL, que contém os integradores reescritos de forma a processarem as iterações de forma paralela. Após a ordem de execução dos cálculos, a CPU fica à espera que a GPU termine os cálculos e a informação é então reconvertida para ser utilizada pelo programa normalmente. Apesar deste método ser bem mais complexo que os outros dois, a execução de cálculos em paralelo da placa gráfica é de tal forma eficiente que faz deste método o mais rápido deles todos.

Para comparar o desempenho destes métodos de forma quantitativa, foram feitos três testes onde se simulam números elevados de partículas que interagem por meio de campos gravitacionais. Uma vez que estes testes dependem enormemente das características do computador, vale mencionar que os mesmos foram feitos num portátil com o nome 'Legion 5 15ACH6H' com uma CPU *'AMD Ryzen 5 5600H 3,30 GHz'* e uma GPU *'NVIDIA GeForce RTX 3060 Mobile'*.

O primeiro teste foi realizado para 27 000 partículas cujo resultado se pode observar na tabela 3.

Tabela 3: Comparação de métodos de execução para simulação gravitacional com 27000 partículas

Método de execução	Iterações por segundo	Interações calculadas por segundo	Desempenho relativo
CPU Single Core	0,43	156 729 195	+0%
CPU Multithreading	2,4	874 767 600	+458%
GPU Compute	29	10 570 108 500	+6 644%

A posição das partículas e os recursos utilizados para as gerar em nada afetam o desempenho da simulação, pelo que apenas o número de partículas é relevante. Nesta situação apenas o método que usa a placa gráfica teve resultados aceitáveis com 29 iterações por segundo, ao passo que os outros dois métodos não conseguiram ultrapassar sequer 3 iterações por segundo. A coluna "Interações calculadas por segundo"indica o número de interações entre partículas por cada segundo. Por exemplo, num sistema de 50 partículas é necessário calcular a interação entre cada partícula com todas as outras. Por questões de desempenho, ao invés de simplesmente se iterar sobre todas as partículas (o número total de interações seria $50^2 = 2500$), o *Force Fields* calcula o mínimo de interações possível descartando iterações previamente calculadas, pelo que o número total de interações é dado pelo coeficiente binomial mostrado na equação 32,

$$\binom{n}{2} = \frac{n!}{2!(n-2)!}. \tag{32}$$

Assim, ao invés de 2500 cálculos para um sistema de 50 partículas, são realizados apenas $\frac{50!}{2!(50-2)!} = 1225$ cálculos. A coluna "Interações calculadas por segundo" indica portanto o número de interações realizadas numa iteração multiplicado pelo número de iterações por segundo, o que representa efetivamente a eficiência computacional do método. Para clarificar os resultados, a coluna "Desempenho relativo" mostra em percentagem a quantidade de iterações por segundo que os métodos "CPU Multithreading" e "GPU Compute" realizam a mais comparativamente ao método "CPU Single Core" que para estas simulações é considerado o teste de controlo.

Os outros dois testes apresentados nas tabelas 4 e 5 foram simulados utilizando respetivamente 6859 e 3120 partículas. A quantidade de partículas em cada um destes testes foi determinado de forma a que fosse possível constatar quantas partículas conseguem os métodos "CPU Multithreading" e "CPU Single Core" simular a 29 iterações por segundo, tal como aconteceu para o método "GPU Compute" no primeiro teste.

Tabela 4: Comparação de métodos de execução para simulação gravitacional com 6859 partículas

Método de execução	Iterações por segundo	Interações calculadas por segundo	Desempenho relativo
CPU Single Core	6,3	148 172 919	+0%
CPU Multithreading	29	682 065 819	+360%
GPU Compute	110	2 587 146 210	+1 646%

Tabela 5: Comparação de métodos de execução para simulação gravitacional com 3120 partículas

Método de execução	Iterações por segundo	Interações calculadas por segundo	Desempenho relativo
CPU Single Core	29	141 103 560	+0%
CPU Multithreading	104	506 026 560	+259%
GPU Compute	181	880 680 840	+524%

Estes dois últimos testes apenas confirmam o primeiro, no sentido em que deixam claro que utilizar as várias linhas de execução do processador ou utilizar a placa gráfica, traz grandes vantagens em termos de desempenho computacional. No entanto é importante disponibilizar estes três métodos ao utilizador uma vez que poderão existir problemas de compatibilidade com os métodos mais eficientes. Por esta razão, a escolha do método a utilizar pode ser alterada pelo utilizador a qualquer instante na aba de configurações (já mostrada na figura 42 à direita), no seletor com o nome "Processing Mode" que disponibiliza os modos "CPU Single Core", "CPU Multithreading" e "GPU Computing".

4.7 PERSONALIZAÇÃO DO AMBIENTE, DESENHO DE TRAJETÓRIAS E LIGAÇÕES MOLECULARES

De forma a melhor captar certas características das simulações geradas, existem algumas ferramentas para auxiliar na visualização de trajetórias e, no caso do simulador intermolecular, de ligações químicas.

Na aba de configurações existe uma opção chamada "Trajectories" que quando habilitada, gera linhas que marcam as posições anteriores das partículas. Um exemplo, na simulação eletromagnética, pode ser observado na figura 53, que mostra a trajetória de duas partículas com cargas simétricas sob influência de um campo elétrico gerado por duas placas com cargas também simétricas.

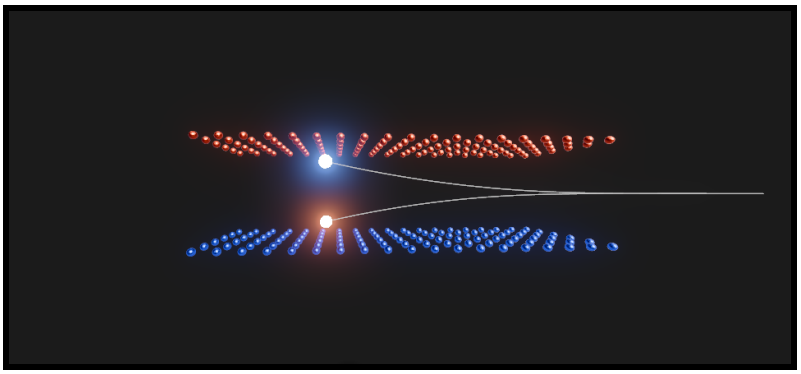

Figura 53: Exemplo de trajetória gerada no *Force Fields* no simulador de campos eletromagnéticos.

Outra possibilidade com um propósito um pouco diferente é utilizar uma técnica chamada "*motion blur*" onde corpos em movimentos são linearmente desfocados, o que se torna útil quando se pretende obter imagens do *software* como a da figura 54 e mesmo assim reter a noção de movimento. Esta opção pode também ser habilitada na aba de configurações, no menu "*Ambient*".

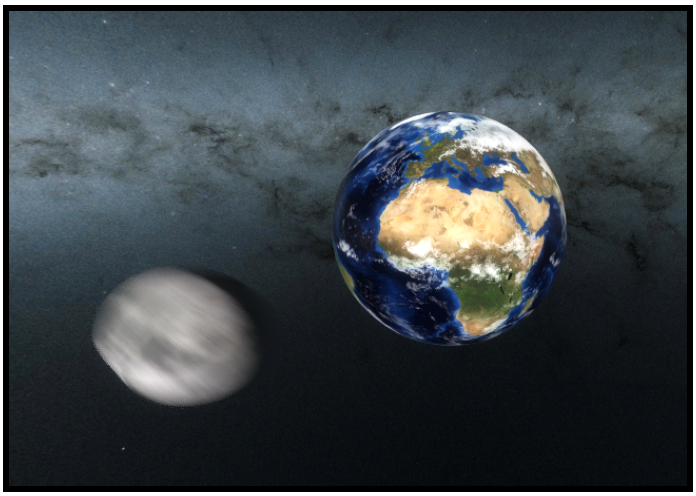

Figura 54: Simulação com o intuito de mostrar a técnica *motion blur* que mantém a noção de movimento em imagens estáticas.

Na simulação intermolecular de Lennard-Jonnes, é pertinente mostrar graficamente quando duas partículas estão próximas o suficiente de forma a que seja necessária alguma energia para as separar. Esta representação é normalmente feita através de linhas que conectam ambas as partículas e são tipicamente chamadas de ligações moleculares/químicas. Estas ligações começam a aparecer quando a distância entre partículas é inferior a 1,2σ sendo σ a distância onde o potencial de Lennard-Jonnes é nulo. A espessura das ligações é variável e máxima quando a distância é igual ou inferior a σ. Um exemplo pode ser observado na figura 55.

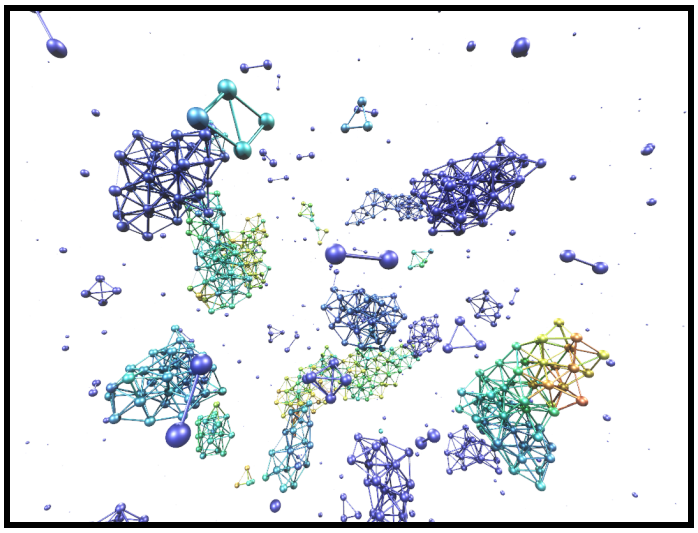

Figura 55: Exemplos de ligações moleculares/químicas entre partículas regidas pelo potencial de Lennard-Jonnes.

5

CENÁRIOS DE SIMULAÇÃO, SEUS RESULTADOS E DISCUSSÃO

Finalmente, para testar na prática o *Force Fields*, este capítulo apresenta a avaliação do seu desempenho em diferentes condições, por meio de vários parâmetros de desempenho bem definidos, como por exemplo a estabilidade energética do sistema. Para tal foram construidos cenários onde o resultado esperado é bem conhecido, de forma a que se faça uma avaliação tanto qualitativa como quantitativa do desempenho do simulador.

5.1 FORMULAÇÕES UTILIZADAS NA RECOLHA DE DADOS

O *Force Fields* está nativamente equipado com ferramentas de análise estatística. Para tal, o sistema recolhe variáveis ao longo do tempo como a velocidade do centro de massa e a energia mecânica total do sistema de partículas e, quando a simulação termina, realiza uma série de cálculos para fornecer dados estatísticos já tratados.

Os sistemas estudados são analisados num referencial inercial relativo às estrelas fixas e são isolados de influências externas pelo que a resultante das forças é nula. Por esta razão, a velocidade do centro de massa do sistema deverá necessariamente ser constante. Tendo isto em conta, medir este parâmetro é um bom indicador de que o integrador respeita a 1.ª lei de Newton. O momento de um sistema de partículas é dado pelo produto da velocidade do centro de massa com a massa total. Tendo isto em conta, a velocidade do centro de massa $\vec{v}_{CM}$ (equação 33) é calculada tendo em conta o momento $m_i\vec{v}_i$ e a massa m_i de um certo número n de partículas, que também pode ser interpretado como sendo a média ponderada das velocidades das partículas,

$$\vec{v}_{CM} = \frac{\vec{P}}{M} = \frac{\sum_{i=1}^{n} m_i\vec{v}_i}{\sum_{i=1}^{n} m_i}. \tag{33}$$

De forma análoga, o momento angular total de um sistema isolado também deve permanecer constante, sendo este um bom indicador da conservação do momento angular. O momento angular $\vec{L}$ de um sistema de partículas é dado pela soma dos momentos angulares individuais de cada partícula, calculados em relação a um ponto de referência, neste caso, o centro de massa do sistema. Para um sistema de n partículas, o momento angular total relativo ao centro de massa (equação 34) é calculado tendo em conta a posição relativa $\vec{r}_{i/CM} = \vec{r}_i - \vec{r}_{CM}$, a velocidade relativa $\vec{V}_{i/CM} = \vec{V}_i - \vec{V}_{CM}$ e a massa m_i de cada partícula,

$$\vec{L} = \sum_{i=1}^{n} \vec{r}_{i/CM} \times m_i \vec{V}_{i/CM}. \tag{34}$$

A conservação do momento angular é particularmente útil para validar a precisão do simulador em sistemas com velocidade angular inicial não nula, pois qualquer desvio significativo do valor inicial indicaria a presença de erros de integração.

Para além da conservação dos momentos linear e angular, a energia mecânica total de um sistema isolado também deve permanecer constante. Esta é uma consequência direta do teorema da conservação da energia mecânica, que estabelece que a soma das energias cinética e potencial de um sistema isolado é constante. A energia mecânica total do sistema Em_{total} é a soma das energias cinéticas Ec e das energias potenciais Ep de todas as partículas.

A energia cinética Ec total é calculada segundo a equação 35a tendo em conta a velocidade instantânea v_i e a massa m_i de cada partícula,

$$Ec = \frac{1}{2} \sum_{i=1}^{n} m_i v_i^2. \tag{35a}$$

A energia potencial gravítica total é calculada segundo a equação 35b tendo em conta a distância entre partículas r_{ij} e as suas massas m_i e m_j, incluindo a constante gravitacional G,

$$Ep = -G \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{m_i m_j}{r_{ij}}. \tag{35b}$$

Com estas duas fórmulas, a energia mecânica toma a forma completa que se pode observar na equação 35c,

$$Em_{total} = \sum_{i=1}^{n} \left(\frac{1}{2} m_i v_i^2 - G \sum_{j=i+1}^{n} \frac{m_i m_j}{r_{ij}} \right). \tag{35c}$$

A implementação computacional realizada no *Force Fields* utiliza uma lista que armazena uma variedade de informações de cada partícula, sendo pertinentes neste caso a posição e a velocidade (vetores de três dimensões) e a massa (escalar). Um ou mais ciclos *for* iteram sobre esta lista de forma a obter dados estatísticos como a média e desvio padrão dos dados obtidos. Estes dados permitem avaliar os vários integradores implementados e até a fidelidade do programa nas condições testadas.

5.2 AVALIAÇÃO DE INTEGRADORES - CENÁRIO DE SIMULAÇÃO GRAVITACIONAL COM DOIS CORPOS

Após a definição das formulações utilizadas para a recolha de dados, procede-se à avaliação dos diferentes integradores implementados no *Force Fields*. Para tal, foram selecionados dois cenários de teste: um sistema Terra-Sol, que representa um caso real e bem conhecido e um sistema hipotético de três corpos com massas iguais. Ambos os sistemas são unicamente governados pela força da gravidade. Este subcapítulo aborda o cenário Terra-Sol, representado pelo esquema da figura 56 e tem as seguintes condições iniciais:

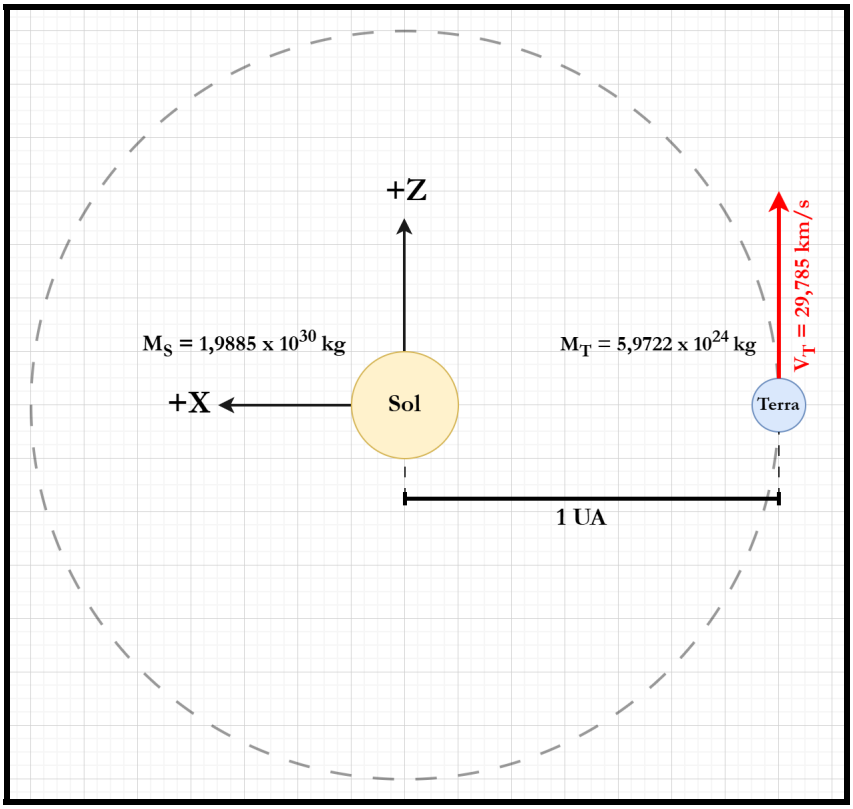

Figura 56: Esquemático do cenário criado para o sistema Sol-Terra.

- O Sol encontra-se posicionado na origem do referencial.
- O Sol tem velocidade nula.
- O Sol tem uma massa de $1{,}9885 \times 10^{30}$ kg.
- A Terra encontra-se a uma UA (unidade astronómica) de distância do Sol ($1{,}496 \times 10^{11}$ m).
- A velocidade da Terra é perpendicular ao vetor que a liga ao Sol e tem magnitude 29,785km/s.
- A Terra tem uma massa de $5{,}9722 \times 10^{24}$ kg.

- O cenário é calculado para 2 anos, em tempo de simulação, mas com um fator de escala de 2 meses/s, sendo portanto o tempo real de simulação de 12s.
- No total são realizadas cerca de 13 000 iterações, sendo que este valor depende do método integrativo utilizado. Isto acontece porque neste teste o tempo de simulação é fixo, o que significa que se o método for computacionalmente mais intensivo, serão realizadas menos iterações, ao passo que um método simples realizará mais iterações para o mesmo intervalo de tempo. Esta é a forma mais justa de avaliar os métodos uma vez que desta forma, todos despendem do mesmo poder computacional.
- Uma vez que a Terra tem uma velocidade inicial, mas o Sol se encontra estático, é expectável que a velocidade do centro de massa do sistema seja não nula, mesmo que a massa da Terra seja várias ordens de grandeza menor que a do Sol.
- Não existem forças externas ao sistema, logo a energia mecânica deverá conservar-se.

A figura 57 mostra uma captura de ecrã do *software* com o cenário anteriormente descrito.

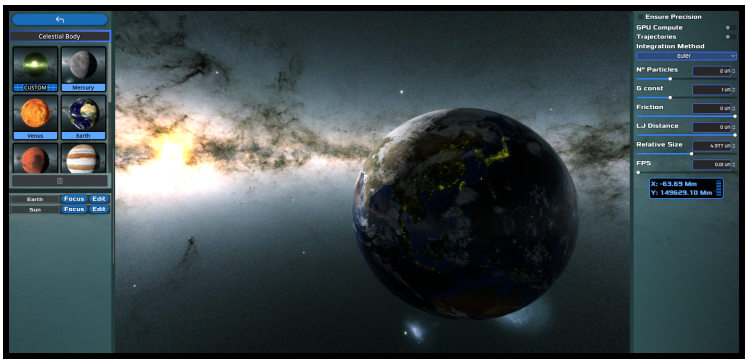

Figura 57: Captura de ecrã do *software Force Fields*, no momento da simulação do sistema Terra-Sol.

É pertinente calcular previamente algumas grandezas físicas de forma analítica, para daí testar a exatidão do *Force Fields* relativamente a estas medições.

A energia mecânica total do sistema Terra-Sol é dada pela expressão 36, resolvida para os valores descritos anteriormente:

$$\begin{aligned} E_m &= \frac{1}{2} M_T v^2 - G \frac{M_S M_T}{r_{ST}} \\ &= \frac{1}{2} (5{,}9722 \times 10^{24}) (2{,}9785 \times 10^{4})^2 - \frac{(6{,}6743 \times 10^{-11}) (1{,}9885 \times 10^{30}) (5{,}9722 \times 10^{24})}{1{,}4960 \times 10^{11}} \\ &= -2{,}6492 \times 10^{33}\,\mathrm{J}\,. \end{aligned} \tag{36}$$

A velocidade do centro de massa $\vec{V}_{CM}$ do sistema Terra-Sol, para estas condições, terá a mesma direção e sentido que a velocidade da Terra, e consequentemente, que o eixo z. O seu módulo é dado pela expressão 37:

$$\begin{aligned} V_{CM} &= \frac{m_S v_S + m_T v_T}{m_S + m_T} \\ &= \frac{(1{,}9885 \times 10^{30}) \cdot 0 + (5{,}9722 \times 10^{24}) \cdot (2{,}9785 \times 10^{4})}{1{,}9885 \times 10^{30} + 5{,}9722 \times 10^{24}} \\ &= 0{,}0895\,\mathrm{m/s}. \end{aligned} \tag{37}$$

Para calcular o momento angular total do sistema, é útil determinar primeiro a posição do centro de massa. Uma vez que $\vec{r}_{cm}$ está contido no segmento de reta que une o centro do Sol ao centro da Terra, o mesmo terá a mesma direção mas sentido contrário ao eixo x. O módulo de $\vec{r}_{cm}$ pode ser observado na expressão 38:

$$\begin{aligned} r_{cm} &= \frac{m_S \cdot r_S + m_T \cdot r_T}{m_S + m_T} \\ &= \frac{(1{,}9885 \times 10^{30}) \cdot 0 + (5{,}9722 \times 10^{24}) \cdot (1{,}496 \times 10^{11})}{1{,}9885 \times 10^{30} + 5{,}9722 \times 10^{24}} \\ &= 4{,}493 \times 10^{5}\,\mathrm{m} \end{aligned} \tag{38}$$

O momento angular total do sistema Terra-Sol é dado pela expressão 39a:

$$\begin{aligned} \vec{L}_{total} &= \vec{L}_S + \vec{L}_T \\ &= m_S(\vec{r}_S - \vec{r}_{CM}) \times (\vec{v}_S - \vec{v}_{CM}) + m_T(\vec{r}_T - \vec{r}_{CM}) \times (\vec{v}_T - \vec{v}_{CM}) \end{aligned} \tag{39a}$$

Para simplificar o cálculo de $\vec{L}$, constata-se que tanto $(\vec{r}_S - \vec{r}_{CM}) \times (\vec{v}_S - \vec{v}_{CM})$ como $(\vec{r}_T - \vec{r}_{CM}) \times (\vec{v}_T - \vec{v}_{CM})$ apontam no sentido positivo do eixo y. Por esta razão, L tem a mesma direção e sentido que o eixo y e pode ser calculado através da expressão 39b:

$$\begin{aligned} L_{total} &= m_S \cdot (-r_{CM}) \cdot (-v_{CM}) + m_T \cdot (r_T - r_{CM}) \cdot (v_T - v_{CM}) \\ &= (1{,}9885 \times 10^{30}) \cdot -(4{,}493 \times 10^{5}) \cdot -0{,}0895 \\ &+ (5{,}9722 \times 10^{24}) \cdot (1{,}496 \times 10^{11} - 4{,}493 \times 10^{5}) \cdot (29785 - 0{,}0895) \\ &= (7{,}996 \times 10^{34}) + (2{,}6611 \times 10^{40}) \\ &= 2{,}6611 \times 10^{40}\,\mathrm{kg \cdot m^2/s} \end{aligned} \tag{39b}$$

O desempenho de cada um dos métodos no *Force Fields* foi avaliado através da variação da velocidade do centro de massa ($\vec{V}_{CM}$), da variação do momento

angular calculado em relação ao ponto de centro de massa ($\vec{L}_{CM}$), da média da energia mecânica ($\bar{E}_m$) e do seu desvio padrão (σ_{E_m}). Os resultados obtidos foram os seguintes:

Método de Euler

- $\vec{V}_{CM} = (0{,}0;\ 0{,}0;\ 0{,}0895)$ m/s ±0,000%
- $\vec{L}_{CM} = (0{,}0;\ 2{,}6961;\ 0{,}0) \times 10^{40}$ kg · m²/s ±1,30%
- $\bar{E}_m = -2{,}61763 \times 10^{33}$ J ±1,20%
- $\sigma_{E_m} = 1{,}78851 \times 10^{31}$ J

Método de Euler semi-implícito

- $\vec{V}_{CM} = (0{,}0;\ 0{,}0;\ 0{,}0895)$ m/s ±0,000%
- $\vec{L}_{CM} = (0{,}0;\ 2{,}6611;\ 0{,}0) \times 10^{40}$ kg · m²/s ±0,003%
- $\bar{E}_m = -2{,}64914 \times 10^{33}$ J ±0,00%
- $\sigma_{E_m} = 8{,}6277941 \times 10^{27}$ J

Método Velocidade de Verlet / Método *Leapfrog*

- $\vec{V}_{CM} = (0{,}0;\ 0{,}0;\ 0{,}0895)$ m/s ±0,000%
- $\vec{L}_{CM} = (0{,}0;\ 2{,}6609;\ 0{,}0) \times 10^{40}$ kg · m²/s ±0,006%
- $\bar{E}_m = -2{,}64943 \times 10^{33}$ J ±1,01%
- $\sigma_{E_m} = 1{,}49880 \times 10^{29}$ J

Método de Runge-Kutta de 4.ª ordem (*RK-4*)

- $\vec{V}_{CM} = (0{,}0;\ 0{,}0;\ 0{,}0895)$ m/s ±0,000%
- $\vec{L}_{CM} = (0{,}0;\ 2{,}6611;\ 0{,}0) \times 10^{40}$ kg · m²/s ±0,003%
- $\bar{E}_m = -2{,}64916 \times 10^{33}$ J ±0,00%
- $\sigma_{E_m} = 8{,}12221 \times 10^{27}$ J

Método de Yoshida de 4.ª ordem

- $\vec{V}_{CM} = (0{,}0;\ 0{,}0;\ 0{,}0895)$ m/s ±0,000%
- $\vec{L}_{CM} = (0{,}0;\ 2{,}6611;\ 0{,}0) \times 10^{40}$ kg · m²/s ±0,003%
- $\bar{E}_m = -2{,}64919 \times 10^{33}$ J ±0,00%
- $\sigma_{E_m} = 1{,}29484 \times 10^{28}$ J

Método de Yoshida de 6.ª ordem

- $\vec{V}_{CM} = (0{,}0;\ 0{,}0;\ 0{,}0895)$ m/s ±0,000%
- $\vec{L}_{CM} = (0{,}0;\ 2{,}6610;\ 0{,}0) \times 10^{40}$ kg · m²/s ±0,002%
- $\bar{E}_m = -2{,}64918 \times 10^{33}$ J ±0,00%

- $\sigma_{E_m} = 2{,}80732 \times 10^{26}\,\mathrm{J}$

Para melhor visualizar a distribuição da energia mecânica para cada integrador, apresenta-se na figura 58 um diagrama de caixas que compara os diferentes métodos.

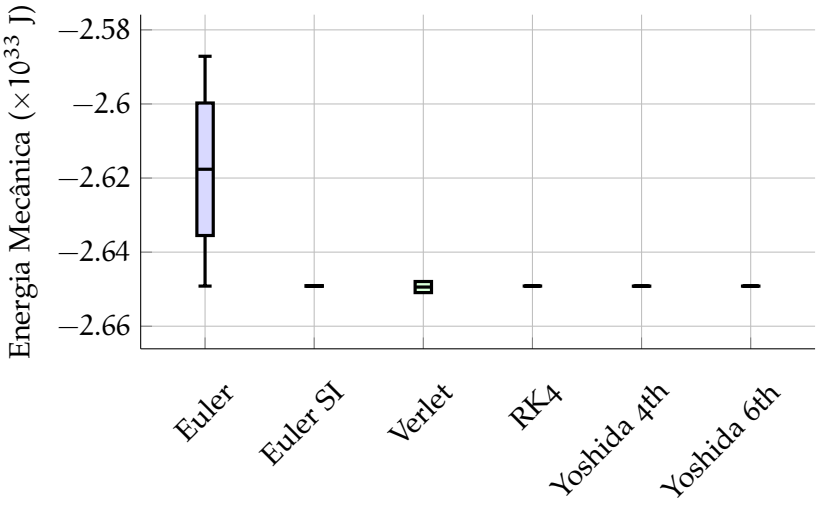

Figura 58: Diagrama de caixas da energia mecânica para cada integrador. As extremidades das caixas são definidas pelo desvio padrão e não pelos quartis. As extremidades das linhas indicam os valores mínimo e máximo da energia registados para cada método.

Embora o método de Euler seja útil para ilustrar limitações de integradores simples, o seu erro é tão elevado que dificulta a visualização das diferenças entre os métodos mais precisos. Por isso, apresenta-se na figura 59 um novo diagrama de caixas sem o método de Euler, permitindo comparar melhor os integradores de maior precisão no sistema Terra-Sol.

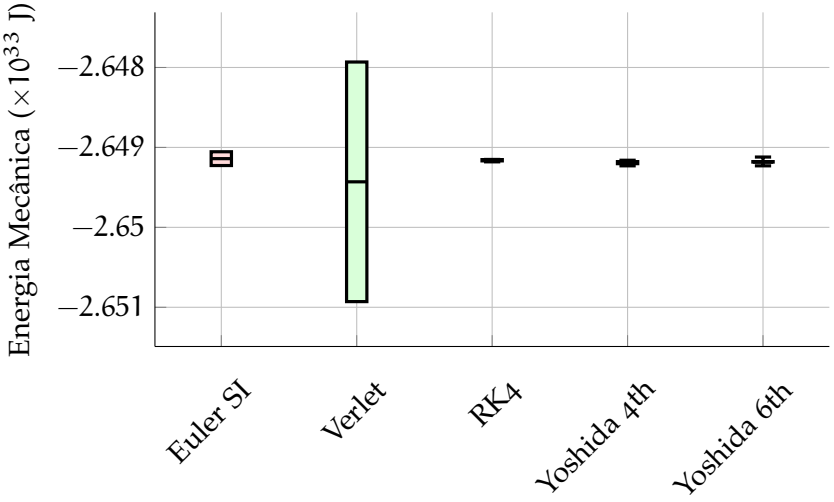

Figura 59: Diagrama de caixas da energia mecânica para cada integrador no sistema Terra-Sol, excluindo o método de Euler.

Conclusão

Os resultados obtidos são bons marcadores para avaliar o desempenho dos integradores, pelo menos nas condições em que são utilizados no *Force Fields*.

A velocidade do centro de massa para todos os integradores manteve-se constante, igual e de acordo com o resultado teórico calculado anteriormente ($8.95 cm/s$), mostrando a precisão do simulador uma vez que calcula sem problemas uma velocidade na ordem dos cm/s num sistema de proporções astronómicas.

É relevante comparar também a conservação do momento angular do sistema, uma vez que este é outro indicador fundamental da precisão dos integradores. A tabela a seguir apresenta os valores do momento angular total relativo ao centro de massa ($\vec{L}_{CM}$) para cada integrador, bem como a respetiva variação percentual ao longo da simulação.

Tabela 6: Momento angular total relativo ao centro de massa ($\vec{L}_{CM}$) e variação percentual para cada integrador no problema dos três corpos

Método	$\vec{L}_{CM}$ $(kg \cdot m^2/s)$	Variação (%)
Euler	$(0{,}0;\ 2{,}6961;\ 0{,}0) \times 10^{40}$	$\pm 1{,}30$
Euler semi-implícito	$(0{,}0;\ 2{,}6611;\ 0{,}0) \times 10^{40}$	$\pm 0{,}003$
Verlet/Leapfrog	$(0{,}0;\ 2{,}6609;\ 0{,}0) \times 10^{40}$	$\pm 0{,}006$
Runge-Kutta 4ª	$(0{,}0;\ 2{,}6611;\ 0{,}0) \times 10^{40}$	$\pm 0{,}003$
Yoshida 4ª	$(0{,}0;\ 2{,}6611;\ 0{,}0) \times 10^{40}$	$\pm 0{,}003$
Yoshida 6ª	$(0{,}0;\ 2{,}6610;\ 0{,}0) \times 10^{40}$	$\pm 0{,}002$

Como se pode observar, quase todos os integradores apresentam uma excelente concordância com a lei da conservação do momento angular, com variações percentuais muito reduzidas. O método de Euler, por outro lado, apresenta uma variação significativamente maior, evidenciando as suas limitações já conhecidas. Para além disso, os valores simulados do momento angular são muito próximos do valor teórico calculado de $2{,}6611 \times 10^{40}\ kg \cdot m^2/s$, com todos os integradores, exceto Euler, apresentando um erro inferior a 0,01%.

Relativamente à variação da energia mecânica, que deverá ser a menor possível, o valor mais importante para comparar os integradores é o desvio padrão relativamente ao valor médio. Na tabela 7, estes valores são apresentados por integrador.

Para facilitar a interpretação dos resultados, mostra-se, na última coluna, o desvio padrão relativo, o qual expressa o desvio padrão relativamente à média da energia do sistema.

O desvio padrão tomou valores distintos para cada integrador, tendo o método de Yoshida de 6.ª ordem tido o menor desvio padrão, seguido do de Runge-Kutta e do de Euler semi-implícito. No entanto, considerando que a energia do sistema Sol-Terra calculada anteriormente é cerca de $-2{,}6492 \times 10^{33} J$, os desvios padrão apresentados (com ordens de grandeza de 26 a 31), mostram uma grande precisão no cálculo da energia mecânica, ao passo que as médias mostram uma grande exatidão no resultado.

Tabela 7: Energia mecânica e desvio padrão absoluto e relativo do sistema Sol-Terra, para cada um dos integradores utilizados

Método	$\bar{E}_m$ (J)	σ da Energia (J)	σ (%)
Yoshida 6th	$-2{,}64918 \times 10^{33}$	$2{,}80732 \times 10^{26}$	0,0000106%
Runge-Kutta	$-2{,}64916 \times 10^{33}$	$8{,}12221 \times 10^{27}$	0,000307%
Euler SI	$-2{,}64914 \times 10^{33}$	$8{,}6277941 \times 10^{27}$	0,000326%
Yoshida 4th	$-2{,}64919 \times 10^{33}$	$1{,}29484 \times 10^{28}$	0,000489%
Verlet/Leapfrog	$-2{,}64943 \times 10^{33}$	$1{,}49880 \times 10^{29}$	0,00566%
Euler	$-2{,}61763 \times 10^{33}$	$1{,}78851 \times 10^{31}$	0,683%

5.3 AVALIAÇÃO DE INTEGRADORES - CENÁRIO DE SIMULAÇÃO GRAVITACIONAL COM TRÊS CORPOS

Após a análise do sistema Terra-Sol, que representa um caso de dois corpos, procede-se agora à avaliação dos integradores num cenário mais complexo: um sistema de três corpos com massas iguais. Este cenário é particularmente interessante, pois ao contrário do sistema Terra-Sol que tem uma solução analítica conhecida, os sistemas de três corpos são, em geral, caóticos. Para garantir a estabilidade do sistema, recorreu-se a um conjunto de condições iniciais onde os três corpos seguem uma trajetória em forma de lemniscata. Este é um caso particular cuidadosamente definido [38], pelo que os valores a seguir definidos, são importantes para evitar que os corpos colidam ou entrem em trajetória de escape, como é típico de sistemas de três corpos cujas condições iniciais não sejam cuidadosamente definidas para manterem estabilidade temporal:

- Cada um dos três corpos tem uma massa de $14{,}97 \times 10^{27}$ kg.
- As posições e velocidades dos três corpos tomam os valores mostrados no grupo de equações 40. Segundo estas condições iniciais, o movimento dos corpos está naturalmente contido no plano $Y = 0$.

$$\vec{r}_1 = -\vec{r}_2 = (97{,}0\hat{\imath} - 24{,}3\hat{k}) \times 10^6 \text{ m} \tag{40a}$$

$$\vec{r}_3 = \vec{0} \tag{40b}$$

$$\vec{v}_3 = (-93{,}24\hat{\imath} - 86{,}47\hat{k}) \text{ km/s} \tag{40c}$$

$$-2\vec{v}_1 = -2\vec{v}_2 = \vec{v}_3 \tag{40d}$$

- Uma vez que a soma vetorial das velocidades das partícula é nula (ver equação 40d) e considerando ainda que os três corpos têm a mesma massa, é expectável que a velocidade do centro de massa seja também nula.
- Não existem forças externas ao sistema, logo a energia mecânica deve conservar-se.
- O cenário é calculado para 6 horas, em tempo de simulação, mas com um fator de escala de 30 min/s pelo que o tempo real de simulação é de 12s.
- No total são realizadas cerca de 11 000 iterações.

A figura 60 mostra uma captura de ecrã do *software* durante a simulação do sistema de três corpos. Os valores de posição, velocidade [38] e massa criam um sistema de 3 corpos que seguem uma trajetória estável em forma de lemniscata.

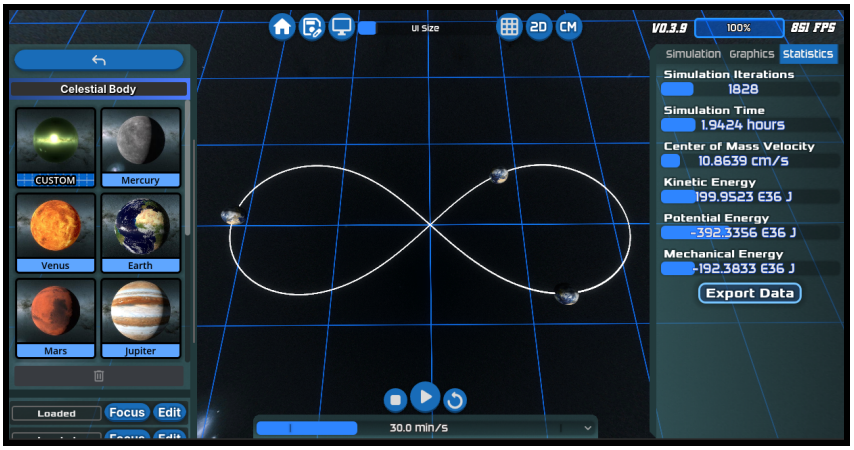

Figura 60: Captura de ecrã do *software Force Fields*, no momento da simulação do sistema de três corpos.

É pertinente calcular previamente qual a energia mecânica esperada e observar se o *Force Fields* a calcula corretamente. A energia mecânica total do sistema de três corpos é dada pelo grupo de equações 41:

$$E_m = Ec + Ep = \sum_{i=1}^{3} \frac{1}{2} m \|\mathbf{v}_i\|^2 - \sum_{1 \leqslant i < j \leqslant 3} G \frac{m^2}{\|\mathbf{r}_i - \mathbf{r}_j\|} \tag{41a}$$

A energia cinética Ec é calculada tendo em conta as velocidades das três partículas com velocidades em módulo previamente calculadas:

$$\begin{aligned} Ec &= \frac{1}{2} m \left(\|\mathbf{v}_1\|^2 + \|\mathbf{v}_2\|^2 + \|\mathbf{v}_3\|^2 \right) \\ &= \frac{1}{2} (14{,}97 \times 10^{27}) \left(2 (63{,}58 \times 10^3)^2 + (127{,}16 \times 10^3)^2 \right) \\ &= 1{,}8156 \times 10^{38}\,\text{J} \end{aligned} \tag{41b}$$

Para a energia potencial Ep, considera-se que a distância D entre as partículas 1 e 3 (e também entre 2 e 3) é $D = \sqrt{97^2 + 24{,}3^2} \times 10^6 \approx 9{,}9997 \times 10^7$ m, enquanto que a distância entre as partículas 1 e 2 é 2D.

$$\begin{aligned} Ep &= -G\, m^2 \left(\frac{1}{2D} + 2\,\frac{1}{D} \right) \\ &= -(6{,}6743 \times 10^{-11})(14{,}97 \times 10^{27})^2 \left(\frac{1}{2\,(9{,}9997 \times 10^7)} + 2\,\frac{1}{9{,}9997 \times 10^7} \right) \\ &= -3{,}7394 \times 10^{38}\,\text{J} \end{aligned} \tag{41c}$$

Portanto, a energia mecânica total do sistema é:

$$\begin{aligned} E_m = Ec + Ep &= 1{,}8156 \times 10^{38} \; - \; 3{,}7394 \times 10^{38} \\ &= -1{,}9238 \times 10^{38}\,\text{J} \end{aligned} \tag{41d}$$

O desempenho de cada um dos métodos no *Force Fields* foi avaliado através da velocidade do centro de massa ($\vec{V}_{CM}$), da média da energia mecânica ($\bar{E}_m$) e do seu desvio padrão (σ_{E_m}). Os resultados obtidos foram os seguintes:

Método de Euler

- $\vec{V}_{CM}$ = (0,000; 0,000; 0,000) m/s ±0,000%
- $\bar{E}_m = -1{,}765845 \times 10^{38}$ J ±8,94%
- $\sigma_{E_m} = 8{,}084402 \times 10^{36}$ J

Método de Euler semi-implícito

- $\vec{V}_{CM}$ = (0,000; 0,000; 0,000) m/s ±0,000%
- $\bar{E}_m = -1{,}923936 \times 10^{38}$ J ±0,034%
- $\sigma_{E_m} = 4{,}201333 \times 10^{34}$ J

Método Velocidade de Verlet / Método *Leapfrog*

- $\vec{V}_{CM}$ = (0,000; 0,000; 0,000) m/s ±0,000%
- $\bar{E}_m = -1{,}924243 \times 10^{38}$ J ±0,078%
- $\sigma_{E_m} = 8{,}642559 \times 10^{34}$ J

Método de Runge-Kutta de 4.ª ordem (*RK-4*)

- $\vec{V}_{CM}$ = (0,000; 0,000; 0,000) m/s ±0,000%
- $\bar{E}_m = -1{,}923815 \times 10^{38}$ J ±0,00064%
- $\sigma_{E_m} = 6{,}217429 \times 10^{32}$ J

Método de Yoshida de 4.ª ordem

- $\vec{V}_{CM}$ = (0,000; 0,000; 0,000) m/s ±0,000%
- $\bar{E}_m = -1{,}923812 \times 10^{38}$ J ±0,00070%
- $\sigma_{E_m} = 5{,}182608 \times 10^{32}$ J

Método de Yoshida de 6.ª ordem

- $\vec{V}_{CM}$ = (0,000; 0,000; 0,000) m/s ±0,000%
- $\bar{E}_m = -1{,}923810 \times 10^{38}$ J ±0,0012%
- $\sigma_{E_m} = 8{,}921184 \times 10^{32}$ J

A figura 61 mostra a distribuição estatística da energia mecânica através de um diagrama de caixas, permitindo uma análise comparativa entre os diversos métodos de integração.

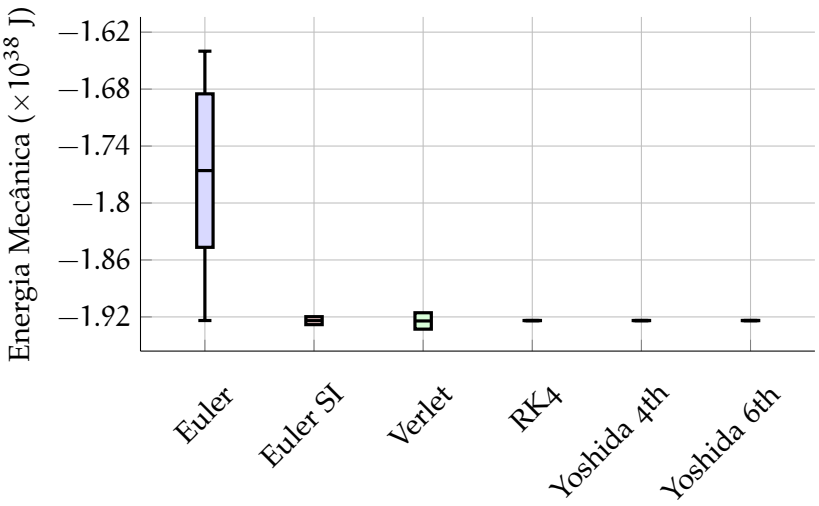

Figura 61: Diagrama de caixas da energia mecânica para cada integrador no problema dos três corpos. As extremidades das caixas são definidas pelo desvio padrão e não pelos quartis. As extremidades das linhas indicam os valores mínimo e máximo da energia registados para cada método.

Mais uma vez, a diferença de precisão entre o método de Euler e os restantes integradores é bastante acentuada, tanto é que o valor máximo/mínimo da energia mecânica para este método varia ±8,94% e o desvio padrão representa 0,42% do valor médio. A baixa precisão do método deve-se ao facto de ser um integrador de primeira ordem que utiliza apenas a derivada no início do intervalo de tempo para estimar a evolução do sistema. Esta aproximação linear ignora as variações da derivada ao longo do intervalo, o que leva a uma subestimação sistemática das mudanças no sistema. Em contraste, os outros integradores utilizam múltiplos pontos de avaliação da derivada, como é o caso para métodos de 2.ª ordem ou superior, permitindo uma melhor aproximação da evolução real do sistema.

Tendo em conta a baixa precisão do método de Euler, apresenta-se na figura 62 um novo diagrama de caixas sem o mesmo, facilitando a comparação detalhada entre os integradores mais precisos neste cenário.

Conclusão

Após a análise detalhada do sistema de três corpos, os resultados obtidos permitem retirar conclusões importantes sobre o desempenho dos diferentes integradores. Relativamente à velocidade do centro de massa, a sua variação ao longo da simulação é nula, mostrando a precisão dos integradores neste requisito. Para além da variação ser nula, a própria velocidade do centro de massa é nula, tal como esperado, uma vez que a soma das velocidades iniciais dos corpos é também nula.

A variação da energia mecânica, tal como apresentada no cenário anterior, é quantificada através do desvio padrão relativamente ao valor médio. Na tabela 8, estes valores são apresentados por integrador. É possível separar os integradores em três grupos distintos: o método de Euler é definitivamente o método mais inexato com um desvio padrão na ordem dos 10^{36} J. Os métodos de Euler semi-

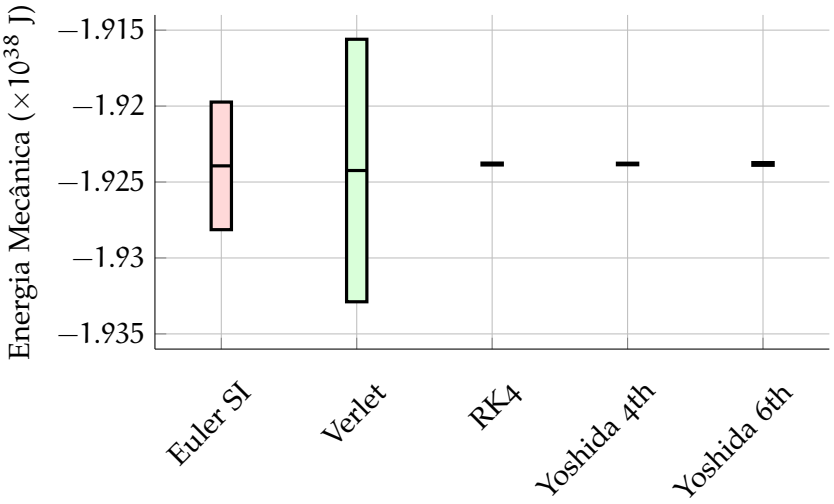

Figura 62: Diagrama de caixas da energia mecânica para cada integrador no problema dos três corpos, excluindo o método de Euler.

implícito e de Verlet/Leapfrog são já mais estáveis em termos energéticos com um desvio padrão na ordem dos 10^{34} J. Os métodos de Runge-Kutta e de Yoshida 4th/6th são os que menos variam a energia do sistema, com um desvio padrão na casa dos 10^{32} J, correspondendo isto a variações de energia inferiores a 0,002%.

Tabela 8: Energia mecânica e desvio padrão absoluto e relativo do sistema de três corpos, para cada um dos integradores utilizados

Método	$\bar{E}_m$ (J)	σ da Energia (J)	σ (%)
Yoshida 4th	$-1{,}923812 \times 10^{38}$	$5{,}182608 \times 10^{32}$	0,000027%
Runge-Kutta	$-1{,}923815 \times 10^{38}$	$6{,}217429 \times 10^{32}$	0,000032%
Yoshida 6th	$-1{,}923810 \times 10^{38}$	$8{,}921184 \times 10^{32}$	0,000046%
Euler SI	$-1{,}923936 \times 10^{38}$	$4{,}201333 \times 10^{34}$	0,0022%
Verlet/Leapfrog	$-1{,}924243 \times 10^{38}$	$8{,}642559 \times 10^{34}$	0,0045%
Euler	$-1{,}765845 \times 10^{38}$	$8{,}084402 \times 10^{36}$	0,42%

É muito provável que com este nível de precisão, o fator limitante sejam os erros propagados devido ao números de casas decimais dos números flutuantes, especialmente considerando que o método de Yoshida de 6ª ordem deveria ter um desempenho significativamente superior ao de Yoshida de 4.ª ordem tendo, no entanto, um desvio padrão superior. Este facto sugere que o alto número de cálculos por iteração deste método está a reduzir a sua precisão devido à propagação de erros causada por números flutuantes de *32 bits*.

Com estes resultados, os três melhores integradores provaram que, para este cenário, o seu desempenho é bastante satisfatório e conforme a lei da conservação da energia mecânica.

5.4 AVALIAÇÃO DA PRECISÃO DE UMA SIMULAÇÃO GRAVÍTICA - SISTEMA SOLAR

Após a avaliação dos integradores em sistemas simples de dois e três corpos, procede-se agora à análise do desempenho do *Force Fields* num cenário mais complexo e realista: o Sistema Solar completo, incluindo os oito planetas e Plutão. Este cenário permite testar o comportamento do simulador num sistema com múltiplos corpos de massas muito diferentes a distâncias elevadas, representando assim um teste mais abrangente da capacidade do simulador em lidar com sistemas gravitacionais complexos.

Para esta simulação, foi utilizado o integrador de Yoshida de 6.ª ordem, por ser um dos integradores que demonstrou o melhor desempenho nos testes anteriores e ser o de maior ordem.

A figura 63 mostra uma captura de ecrã do *software* durante a simulação da evolução do Sistema Solar.

Figura 63: Visualização das órbitas do Sistema Solar na simulação. As órbitas simuladas formam círculos concêntricos em torno do Sol (centro).

As condições iniciais definidas para esta simulação não são exatamente realistas, no sentido em que não se considera a ligeira excentricidade das órbitas planetárias nem órbitas inclinadas relativamente ao plano da eclíptica. Por outras palavras, a simulação recebe como parâmetros iniciais os valores médios das distâncias dos planetas e Plutão ao Sol obtidos através de medições astronómicas e as suas velocidades orbitais são calculadas a partir da equação $v = \sqrt{\frac{GM}{r}}$, de forma a que as órbitas sejam perfeitamente circulares. Desta forma a inserção de valores é simplificada e não é necessário ter em conta a velocidade no periélio e afélio de cada órbita, assim como ângulos de inclinação orbital, de onde seria necessário adicionar uma componente no eixo y da velocidade inicial de cada corpo celeste.

A observação das órbitas circulares estáveis no Sistema Solar simulado é portanto um indicador significativo da precisão do *Force Fields*. O facto das órbitas simuladas serem de facto circulares demonstra que o simulador calcula corretamente as interações gravitacionais entre os corpos celestes. As interações entre

planetas são praticamente desprezáveis tendo em conta a distância entre os mesmos e ainda o facto do Sol possuir 99,86% da massa de todo o Sistema Solar.

Este resultado é particularmente positivo, pois apesar de na realidade os planetas e Plutão descreverem órbitas elípticas, ao utilizar valores médios das suas posições e velocidades, o simulador reproduz com sucesso as órbitas circulares esperadas nestas condições.

A tabela 9 apresenta os desvios padrão das distâncias de cada corpo celeste ao Sol, bem como a sua distância média e variação percentual. Para obter estes valores, o Sistema Solar foi simulado durante um período de 250 anos terrestres, tempo necessário para que todos os corpos celestes completassem pelo menos uma órbita completa, incluindo Plutão que possui o maior período orbital de 248,6 anos. Foram realizadas 22000 iterações para obter estes resultados.

Tabela 9: Desvio padrão das distâncias de cada planeta e Plutão ao Sol

Corpo Celestial	Distância média ao Sol (UA)	Desvio Padrão (UA)	Desvio padrão relativo (%)
Mercúrio	0,389	0,00102	± 0,263%
Vénus	0,724	0,00045	± 0,061%
Terra	1,001	0,00073	± 0,073%
Marte	1,526	0,00129	± 0,085%
Júpiter	5,209	0,00317	± 0,061%
Saturno	9,525	0,00616	± 0,065%
Urano	19,149	0,02201	± 0,115%
Neptuno	30,015	0,03263	± 0,109%
Plutão	39,427	0,03832	± 0,097%

Os resultados apresentados na tabela 9 demonstram a notável estabilidade das órbitas simuladas. Os desvios padrão relativos das distâncias ao Sol são todos inferiores a 0,3%, sendo que a maioria dos corpos celestes apresenta variações na ordem das centésimas de percentagem. O facto de Mercúrio possuir uma variação significativamente superior relativamente aos outros planetas, advém do facto de o próprio Sol se mover ligeiramente devido à influência gravitacional dos outros corpos celestes, sendo este efeito mais pronunciado em Mercúrio por ser o planeta mais próximo. Ainda assim, uma variação de apenas 0,263% na sua órbita demonstra a elevada precisão do simulador para o cenário estudado.

5.5 AVALIAÇÃO DA PRECISÃO DA FORÇA MAGNÉTICA - RAIO DE LARMOR

Para avaliar a precisão do *Force Fields* no cálculo da força magnética, foi simulado o movimento de uma partícula carregada num campo magnético uniforme. Este cenário resulta num movimento circular uniforme bem definido, cujo raio é conhecido como raio de Larmor, o qual pode ser calculado analiticamente e comparado com os resultados da simulação.

O raio de Larmor r_L em metros (m) é dado pela equação 42, onde m é a massa da partícula em quilogramas (kg), $v_\perp$ é a componente da velocidade perpendicular ao campo magnético em metros por segundo (m/s), q é a carga da partícula em Coulomb (C) e B é a magnitude do campo magnético em teslas (T).

$$r_L = \frac{m v_\perp}{|q|B} \tag{42}$$

Neste teste, foi utilizado o integrador de Runge-Kutta de 4.ª ordem (RK4), uma vez que este método é o mais apto para sistemas em que a força depende tanto da posição como da velocidade da partícula, como acontece com a força magnética. Outros integradores, como o método de Verlet e os métodos de Yoshida de 4.ª e 6.ª ordem dele derivados, não estão disponíveis no simulador eletromagnético, pois assumem que a força depende apenas da posição, tal como mencionado no capítulo 2.3.2.

No *Force Fields*, o raio de Larmor r_L é obtido a cada iteração através da equação 43, onde $\bar{v}$ é a velocidade média entre dois instantes consecutivos, Δt é o intervalo de tempo entre esses dois instantes e $\Delta\theta$ é o ângulo entre os vetores velocidade nesses dois instantes.

$$r_L = \frac{\bar{v}\Delta t}{\Delta\theta} \tag{43}$$

Este método é válido uma vez que o raio da curvatura de uma dada trajetória para um curto intervalo de tempo é dado pelo comprimento do segmento percorrido, dividido pelo deslocamento angular $\Delta\theta$ ($r = \frac{s}{\Delta\theta}$), onde s é o comprimento do arco percorrido. Considerando ainda que s é o produto da velocidade média com o intervalo de tempo ($\bar{v}\Delta t$), obtem-se a equação 43.

Este raio é calculado para cada instante e no fim da simulação é realizada a média e o desvio padrão de todos os dados recolhidos.

Foram realizados vários ensaios variando os parâmetros que influenciam o raio de Larmor, como a massa da partícula, a intensidade do campo magnético, a velocidade inicial perpendicular ao campo e a carga da partícula. Para cada conjunto de parâmetros, o raio de Larmor teórico foi calculado e comparado com o valor obtido na simulação, da qual se extraiu também o desvio padrão relativo.

A figura 64 mostra uma captura de ecrã do *software* enquanto simula uma partícula carregada num meio com campo magnético uniforme e constante.

A tabela 10 apresenta os resultados obtidos, comparando os valores teóricos e simulados.

Os resultados demonstram a excelente precisão do *Force Fields* no cálculo do raio de Larmor, com diferenças residuais entre os valores teóricos e simulados. Isto é um indício da boa capacidade de previsão de trajetórias regidas por campos magnéticos no *Force Fields* e ainda da robustez do integrador Runge-Kutta de 4.ª ordem para simulações que envolvam forças que dependem da velocidade.

É importante notar que o raio de Larmor permaneceu estável durante toda a simulação, tal como se pode observar nos desvios padrão relativos, sendo este um

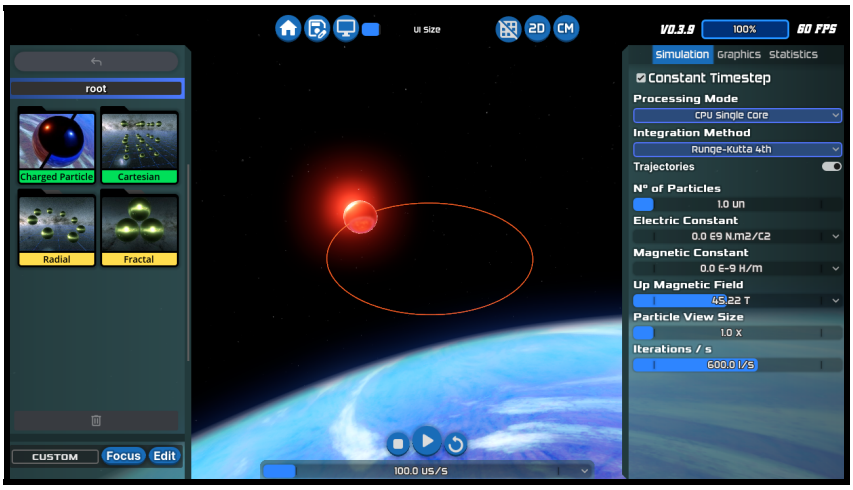

Figura 64: Captura de ecrã do *software Force Fields*, durante a simulação da trajetória de uma partícula carregada na presença de um campo magnético uniforme e constante.

Tabela 10: Comparação entre o raio de Larmor teórico e o simulado, para diferentes parâmetros utilizando o método de Runge-Kutta de 4.ª ordem

m (kg)	B (T)	$v_\perp$ (m/s)	q (C)	r_L (m) teórico	r_L (m) simulado	σ (%)
$1{,}00\times10^{-2}$	$1{,}00\times10^{-2}$	1,00	1,00	1,00	1,00	0,000089
$3{,}00\times10^{-2}$	$5{,}00\times10^{-3}$	$1{,}00\times10^{3}$	0,20	$3{,}00\times10^{4}$	$3{,}00\times10^{4}$	0,000052
3,00	5,00	$2{,}00\times10^{2}$	6,00	$2{,}00\times10^{1}$	$2{,}00\times10^{1}$	0,000150
$1{,}00\times10^{-27}$	$1{,}00\times10^{-14}$	$8{,}00\times10^{6}$	$1{,}00\times10^{-3}$	$8{,}00\times10^{-4}$	$8{,}00\times10^{-4}$	0,0002
$1{,}00\times10^{28}$	45,22	$2{,}00\times10^{-3}$	$5{,}00\times10^{18}$	$8{,}85\times10^{4}$	$8{,}85\times10^{4}$	0,00040

teste que valida tanto a precisão como a exatidão da simulação para o presente cenário.

5.6 AVALIAÇÃO DA PRECISÃO DA FORÇA MAGNÉTICA - TRAJETÓRIA HELICOIDAL GERADA POR UM CAMPO MAGNÉTICO UNIFORME E CONSTANTE

Para testar de forma mais extensiva e completa a trajetória de uma partícula carregada sob um campo magnético uniforme, poderá utilizar-se um sistema de equações paramétricas 44 para calcular de forma analítica a posição da partícula para certos instantes de tempo, comparando-os com a posição calculada pelo *Force Fields* [39].

$$\begin{cases} x(t) = \dfrac{v_{z0}m}{qB}\cos\left(\dfrac{qB}{m}t\right) + x_0 \\ y(t) = v_{y0}t + y_0 \\ z(t) = \dfrac{v_{z0}m}{qB}\sin\left(\dfrac{qB}{m}t\right) + z_0 \end{cases} \tag{44}$$

O sistema de equações toma como pressuposto um espaço cartesiano onde o eixo y tem a mesma direção e sentido que as linhas de campo magnético. Para além disso, o sistema considera que a velocidade inicial tem apenas uma componente inicial em y e em z. Um esquema do cenário pode ser observado na figura 65. Vale notar que a trajetória apresentada nesta figura é apenas válida para partículas com carga positiva.

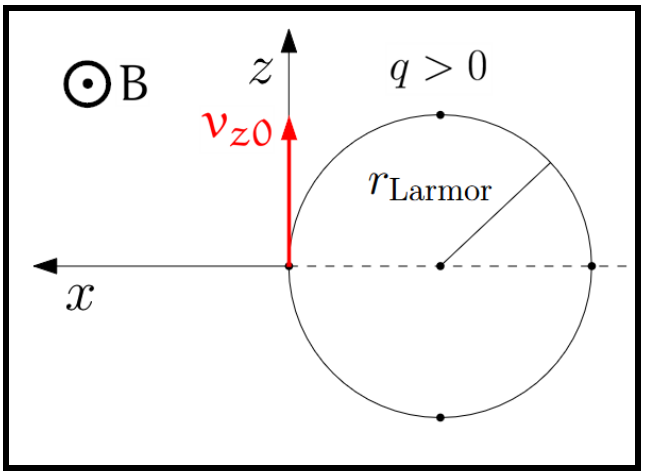

Figura 65: Esquemático do cenário utilizado para a determinação do raio de Larmor.

Se nenhum dos valores for nulo, a trajetória descrita será uma hélice com raio igual ao raio de Larmor. Um exemplo pode ser observada na figura 66.

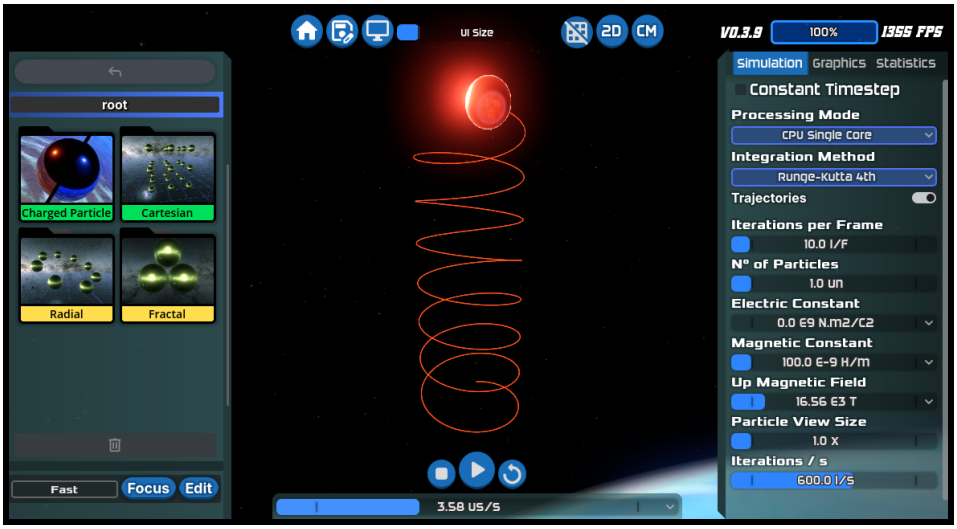

Figura 66: Trajetória de partícula carregada sob campo magnético uniforme e constante com velocidade inicial nos três eixos cartesianos.

Para garantir a coerência das equações paramétricas com os testes feitos no *Force Fields*, é necessário garantir que as constantes x_0, y_0 e z_0 tenham os valores apropriados que garantam o posicionamento da partícula na origem do referencial para o instante $t = 0$, tal como mostrado no sistema de equações 45.

$$\begin{cases} x(0) = \dfrac{v_{z0}m}{qB}\cos(0) + x_0 = 0 \\ y(0) = y_0 = 0 \\ z(0) = \dfrac{v_{z0}m}{qB}\sin(0) + z_0 = 0 \end{cases} = \begin{cases} x_0 = -\dfrac{v_{z0}m}{qB} \\ y_0 = 0 \\ z_0 = 0 \end{cases} \tag{45}$$

O termo x_0 compensa o deslocamento inicial causado pelo movimento circular, permitindo que a trajetória comece na origem. Substituindo os termos de deslocamento nas equações paramétricas 44, obtém-se o sistema de equações 46.

$$\begin{cases} x(t) = \dfrac{v_{z0}m}{qB}\cos\left(\dfrac{qB}{m}t\right) - \dfrac{m}{qB}v_{z0} \\ y(t) = v_{y0}t \\ z(t) = \dfrac{v_{z0}m}{qB}\sin\left(\dfrac{qB}{m}t\right) \end{cases} \tag{46}$$

Este sistema de equações descreve uma trajetória helicoidal que começa na origem e tem velocidade inicial com a mesma direção e sentido que o eixo z.

Para realizar este teste, foram utilizados os seguintes valores:

- $m = 3 \times 10^{-2}$ kg
- $B = 5 \times 10^{-3}$ T
- $v_{z0} = 1$ m/s
- $v_{y0} = 0{,}5$ m/s
- $q = 0{,}2$ C

Destes parâmetros obtêm-se a frequência angular do movimento circular e o raio de Larmor, tal como mostram as equações 47 e 48,

$$\omega = \frac{qB}{m} = \frac{0{,}2 \times 5 \times 10^{-3}}{3 \times 10^{-2}} = \frac{1}{30}\,\mathrm{rad/s}, \tag{47}$$

$$r_L = \frac{mv_{z0}}{qB} = \frac{3 \times 10^{-2} \times 1}{0{,}2 \times 5 \times 10^{-3}} = 30\,\mathrm{m}, \tag{48}$$

com os quais se obtém o sistema de equações 49.

$$\begin{cases} x(t) = r_L\cos(\omega t) - r_L \\ y(t) = v_{y0}t \\ z(t) = r_L\sin(\omega t) \end{cases} \Leftrightarrow \begin{cases} x(t) = 30\cos\left(\dfrac{t}{30}\right) - 30 \\ y(t) = 0{,}5\,t \\ z(t) = 30\sin\left(\dfrac{t}{30}\right) \end{cases} \tag{49}$$

Para analisar a trajetória completa, foram escolhidos instantes de tempo que correspondem a pontos-chave do movimento circular: $t = 0$ (início), $t = \frac{\pi}{2\omega}$ (quarto de volta), $t = \frac{\pi}{\omega}$ (meia volta), $t = \frac{3\pi}{2\omega}$ (três quartos de volta) e $t = \frac{2\pi}{\omega}$

(volta completa). É então possível calcular a posição teórica da partícula nestes instantes.

As posições teóricas foram determinadas através da resolução analítica do sistema de equações 49, cujos resultados podem ser observados na tabela 11. Para obter os valores simulados foram inseridos os parâmetros iniciais do cenário no *Force Fields* de forma a extrair a posição da partícula para diversos instantes de tempo, no decorrer de 70000 iterações.

Tabela 11: Comparação entre valores teóricos e simulados da posição da partícula

ωt	$\vec{r}_{\text{expressão}}$	$\vec{r}_{\text{teórico}}$ (m)	$\vec{r}_{\text{simulado}}$ (m)	ε_{max} (%)
0	$x : 0$ $y : 0$ $z : 0$	$x : 0{,}0000$ $y : 0{,}0000$ $z : 0{,}0000$	$x : 0{,}0000$ $y : 0{,}0000$ $z : 0{,}0000$	0,0000%
$\frac{\pi}{2}$	$x : -R$ $y : \frac{\pi v_{y0}}{2\omega}$ $z : R$	$x : -30{,}0000$ $y : 23{,}5619$ $z : 30{,}0000$	$x : -30{,}0031$ $y : 23{,}5603$ $z : 29{,}9999$	0,0105%
π	$x : -2R$ $y : \frac{\pi v_{y0}}{\omega}$ $z : 0$	$x : -60{,}0000$ $y : 47{,}1239$ $z : 0{,}0000$	$x : -60{,}0001$ $y : 47{,}1213$ $z : -0{,}0272$	0,0055%
$\frac{3\pi}{2}$	$x : -R$ $y : \frac{3\pi v_{y0}}{2\omega}$ $z : -R$	$x : -30{,}0000$ $y : 70{,}6858$ $z : -30{,}0000$	$x : -29{,}9787$ $y : 70{,}6843$ $z : -29{,}9999$	0,0711%
2π	$x : 0$ $y : \frac{2\pi v_{y0}}{\omega}$ $z : 0$	$x : 0{,}0000$ $y : 94{,}2478$ $z : 0{,}0000$	$x : -0{,}0000$ $y : 94{,}2452$ $z : -0{,}0745$	0,0027%

A análise dos resultados mostra uma excelente concordância entre os valores teóricos e simulados. Para quantificar esta concordância, foi calculado o erro relativo para cada componente (x, y, z), tomando como valor máximo entre eles a medida do desvio apresentado na tabela. O raio de Larmor (30m) é verificado na simulação, como se pode observar nos valores máximos das componentes x e z. O desvio máximo de 0,0711% é significativamente pequeno, o que indica uma elevada precisão na modelação de trajetórias de partículas carregadas em campos magnéticos uniformes.

6 CONCLUSÕES

Mostrar e estudar fenómenos naturais sempre foi a motivação deste projeto. O objetivo inicial de desenvolver uma aplicação que simule em tempo real sistemas de partículas com características e leis baseadas na realidade foi cumprido. O ***Force Fields*** é agora uma ferramenta de simulação de sistemas de partículas dedicado a campos gravitacionais, eletromagnéticos e intermoleculares, permitindo ao utilizador explorar os mais variados fenómenos e características de algumas das forças que regem o universo.

O estudo dos integradores e dos vários simuladores disponíveis foi um pilar importante no desenvolvimento do *Force Fields*, uma vez que proporcionou o conhecimento técnico e os conceitos necessários para a implementação desta ferramenta de simulação. A documentação detalhada dos componentes do programa, apresentada nesta dissertação, permitiu a identificação e resolução sistemática de problemas de implementação no *Force Fields*.

O *software* concretiza aquilo para que foi feito, na medida em que, por ser baseado em métodos de integração numérica, o resultado das simulações pode ser inesperado, mesmo para o programador. Aquando do teste de uma simulação eletromagnética, constituída por um condutor com corrente elétrica e uma partícula de teste, verificou-se um comportamento inesperado onde a partícula descrevia uma trajetória de dupla hélice em torno do condutor. Esta constatação desencadeou um pequeno estudo sobre o fenómeno, que veio mais tarde a confirmar-se ser a razão pela qual o campo magnético da Terra naturalmente nos protege da radiação emitida pelos ventos solares, sendo também o princípio que gera as auroras boreais. Este é um exemplo ideal daquilo que se pretende obter com o *Force Fields*: uma ferramenta que permita a exploração, teste e descoberta de fenómenos físicos sobre os quais é difícil fazer experiências num ambiente experimental convencional.

Este programa de simulação está, no entanto, longe de ser completo e perfeito. Por esta razão, o projeto não chegou ao fim. O *Force Fields* já pode ser utilizado ao que se propõe, mas ainda tem muito por onde melhorar e uma vez que a motivação para continuar este projeto não cessou, o mesmo terá continuidade, mesmo que fora do meio académico.

6.1 TRABALHO FUTURO

Existem várias frentes onde se pretende melhorar e incrementar funcionalidades na aplicação desenvolvida. Como base para o trabalho futuro, elencam-se os pontos principais:

- A forma de obter dados estatísticos da simulação poderá ser melhorada com a adição de ferramentas de medição de comprimentos, ângulos, áreas,

perímetros e volumes no espaço tridimensional. Ferramentas que permitam obter o campo vetorial num dado ponto, volume ou superfície tridimensional do espaço definido pelo utilizador poderão ser também bastante úteis na análise do cenário.

- O *Force Fields* poderá ser compilado numa base de *64 bits*. Esta modificação deverá ser sujeita a testes para verificar se esta alteração irá melhorar significativamente a exatidão dos resultados estatísticos obtidos, uma vez que esta alteração impacta negativamente o desempenho do programa.
- A presente dissertação contém os passos que devem ser seguidos para começar a interagir com o *software*. No entanto, o mesmo não possui um sistema de tutoriais integrado que guie novos utilizadores na aprendizagem do programa. Este é um ponto crucial que deve ser implementado para gerar uma maior adesão por parte de novos utilizadores.
- A biblioteca de cenários poderá ser expandida para se tornar mais rica. A possibilidade de salvar os próprios cenários criados é também uma funcionalidade importante que carece de implementação completa (o cenário pode ser salvo num ficheiro *Json*, mas o mesmo ainda não pode ser armazenado e carregado no disco). Será ainda pertinente, a longo prazo, criar uma biblioteca *online* onde vários utilizadores possam compartilhar os vários cenários criados.
- Novos campos de forças poderão ser implementados para enriquecer o projeto, como por exemplo simulações de mecânica de fluidos, simulações bidimensionais da propagação de ondas no espaço (reproduzindo assim a interferência, reflexão, refração e outros fenómenos relativos a ondas) e até mesmo simulação de interações químicas que consigam retratar a polaridade e outras características necessárias à criação de moléculas e estruturas atómicas.

BIBLIOGRAFIA

[1] Matthew Robinson, Karen Bland, Gerald Cleaver, and J. Dittmann. A simple introduction to particle physics part i - foundations and the standard model, May 2013.

[2] Isaac Newton, Daniel Bernoulli, Colin MacLaurin, and Leonhard Euler. *Philosophiae naturalis principia mathematica*, volume 1. excudit G. Brookman; impensis TT et J. Tegg, Londini, 1833.

[3] P B Visscher. Discrete formulation of maxwell equations. *Comput. Phys.*, 3(2):42, 1989.

[4] Eni Generalic. Lennard-jones potential. Croatian-English Chemistry Dictionary & Glossary, June 2022. Accessed on 15 July 2023. Available at https://glossary.periodni.com.

[5] W W Wood and F R Parker. Monte carlo equation of state of molecules interacting with the Lennard-Jones potential. i. a supercritical isotherm at about twice the critical temperature. *J. Chem. Phys.*, 27(3), September 1957.

[6] apexearth. Particle sandbox - gravity simulator, 2023. Accessed on 15 July 2023. Available at http://particlesandbox.com/.

[7] HermannBjorgvin. Gravity, 2023. Accessed on 16 July 2023. Available at https://hermann.is/gravity/.

[8] Paul Falstad. Lagrange points, 2023. Accessed on 17 July 2023. Available at http://falstad.com/lagrange/.

[9] Paul Falstad. 3-d vector fields, 2023. Accessed on 16 July 2023. Available at http://falstad.com/vector3d/vector3d.html.

[10] Paul Falstad. 2-d vector fields, 2023. Accessed on 16 July 2023. Available at http://www.falstad.com/vector/vector.html.

[11] Werner Israel. Event horizons in static vacuum space-times. *Phys. Rev.*, 164(5):1776–1779, December 1967.

[12] Ricky Reusser. Schwarzschild trajectories, 2023. Accessed on 16 July 2023. Available at https://rreusser.github.io/schwarzschild-spacetime/.

[13] David Li. Fluid particles, 2023. Accessed on 16 July 2023. Available at http://david.li/fluid/.

[14] Edan Kwan. The spirit, 2023. Accessed on 16 July 2023. Available at https://edankwan.com/experiments/the-spirit/#amount=252k&motionBlurQuality=medium.

[15] Paul Falstad. Ripple tank simulation, 2023. Accessed on 17 July 2023. Available at http://falstad.com/ripple/Ripple.html.

[16] L. Euler. *Institutiones calculi integralis*. Number vol. 1 in Institutiones calculi integralis. Academia Imperialis Scientiarum, 1792.

[17] Loup Verlet. Computer "experiments"on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Phys. Rev.*, 159:98–103, Jul 1967.

[18] William C. Swope, Hans C. Andersen, Peter H. Berens, and Kent R. Wilson. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *The Journal of Chemical Physics*, 76(1):637–649, 01 1982.

[19] Angel T. Gisbert and Nicola Piovella. Multimode collective atomic recoil lasing in free space. *Atoms*, 8:93, 12 2020.

[20] C. Runge. Ueber die numerische auflosung von differentialgleichungen. *Mathematische Annalen*, 46:167–178, 1895.

[21] O.X. Schlömilch, B. Witzschel, M. Cantor, E. Kahl, R. Mehmke, and C. Runge. *Zeitschrift für Mathematik und Physik*. Number v. 57. B. G. Teubner., 1909.

[22] J.C. Butcher. A history of runge-kutta methods. *Applied Numerical Mathematics*, 20(3):247–260, 1996.

[23] Haruo Yoshida. Construction of higher order symplectic integrators. *Physics Letters A*, 150(5):262–268, 1990.

[24] John C. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley and Sons, Hoboken, New Jersey, third edition, 2016.

[25] T.E. Simos, E. Dimas, and A.B. Sideridis. A runge—kutta—nyström method for the numerical integration of special second-order periodic initial-value problems. *Journal of Computational and Applied Mathematics*, 1994.

[26] Epic Games. Unreal engine: The most powerful real-time 3d creation tool, 2024. Accessed on 13 December 2024. Available at https://www.unrealengine.com/en-US.

[27] Unity Technologies. Unity real-time development platform 3d, 2d, vr and ar engine, 2024. Accessed on 18 December 2024. Available at https://unity.com/.

[28] Crytek GmbH. Cryengine, 2024. Accessed on 18 December 2024. Available at https://www.cryengine.com/.

[29] Armory3D. Armory3d game engine, 2024. Accessed on 18 December 2024. Available at https://armory3d.org/.

[30] UPBGE. Upbge game engine, 2024. Accessed on 18 December 2024. Available at https://upbge.org/.

[31] GarageGames. Torque 3d game engine, 2024. Accessed on 18 December 2024. Available at https://torque3d.org/.

[32] Stride. Stride 3d game engine, 2024. Accessed on 18 December 2024. Available at https://stride3d.net/.

[33] Ariel Manzur Juan Linietsky and contributors. Hosted by the Godot Foundation. Godot engine - free and open source 2d and 3d game engine, 2024. Accessed on 18 December 2024. Available at https://godotengine.org/.

[34] Ariel Manzur Juan Linietsky and the Godot Community. First look at godot's interface - godot engine 4.3 documentation in english. Accessed on

19 December 2024. Available at https://docs.godotengine.org/en/stable/getting_started/introduction/first_look_at_the_editor.html.

[35] Ariel Manzur Juan Linietsky and the Godot Community. Introduction to godot - godot engine documentation. Accessed on 19 December 2024. Available at https://docs.godotengine.org/en/stable/getting_started/introduction/introduction_to_godot.html.

[36] National Academies of Sciences, Engineering, and Medicine. Plasma physics of the local cosmos, chapter 6: Energetic particle acceleration. https://nap.nationalacademies.org/read/10993/chapter/8#69. Accessed on 16 March 2024.

[37] Ariel Manzur Juan Linietsky and the Godot community (CC BY 3.0). Godot engine 4.3 documentation. Accessed on 11 January 2025. Available at https://docs.godotengine.org/en/stable/classes/class_vector3.html#class-vector3-constant-left.

[38] Alain Chenciner and Richard Montgomery. A remarkable periodic solution of the three-body problem in the case of equal masses, 2000.

[39] Stephen T. Thornton and Jerry B. Marion. *Classical dynamics of particles and systems*. Cengage Learning, Andover, 2014.

DECLARAÇÃO

Declaro, sob compromisso de honra, que o trabalho apresentado nesta dissertação, com o título *"Simulação de fenómenos eletromagnéticos e mecânicos em sistemas de partículas"*, é original e foi realizado pelo Raúl Rodrigues Figueirinha (2222903), sob orientação do Professor Doutor Alberto Negrão (alberto.negrao@ipleiria.pt) e do Professor Doutor Telmo Fernandes (telmo.fernandes@ipleiria.pt).

Leiria, Julho de 2025

Raúl Rodrigues Figueirinha