Deomar Santos da Silva Junior

Modelo tridimensional de célula biológica por sistema de multipartículas ativas

Porto Alegre

2021

Deomar Santos da Silva Junior

Modelo tridimensional de célula biológica por sistema de multipartículas ativas

Trabalho de Conclusão de Curso submetido à Universidade Federal do Rio Grande do Sul, como requisito parcial para obtenção do grau de Bacharel em Engenharia Física.

Universidade Federal do Rio Grande do Sul (UFRGS)

Instituto de Física

Orientador: Prof. Dr. Leonardo Gregory Brunnet

Coorientador: Me. Paulo Casagrande Godolphim

Porto Alegre

2021

Deomar Santos da Silva Junior
Modelo tridimensional de célula biológica por sistema de multipartículas ativas/ Deomar Santos da Silva Junior. – Porto Alegre, 2021-
100 p. : il. ; 30 cm.

Orientador: Prof. Dr. Leonardo Gregory Brunnet

Trabalho de Conclusão de Curso – Universidade Federal do Rio Grande do Sul (UFRGS)
Instituto de Física
, 2021.

I. Células 3D. II. GPU. III. Simulação. IV. Computação paralela V. OpenCL.
CDU 02:141:005.7

Deomar Santos da Silva Junior

Modelo tridimensional de célula biológica por sistema de multipartículas ativas

Trabalho de Conclusão de Curso submetido à Universidade Federal do Rio Grande do Sul, como requisito parcial para obtenção do grau de Bacharel em Engenharia Física.

Porto Alegre, 01 de dezembro de 2021:

Prof. Dr. Leonardo Gregory Brunnet
Orientador

Prof. Dr. Cristiano Krug
Professor convidado

Prof. Dr. Pedro Luis Grande
Professor convidado

Porto Alegre
2021

Agradecimentos

Primeiramente agradeço a minha minha família que sempre depositou confiança em mim, forneceu as estruturas e o apoio para todo trabalho e estudo feitos por mim e me apoiou nos momentos mais estressantes. Agradeço também aos amigos que sempre estiveram ao meu lado para descontrair e ajudar quando eu mais precisei e estiveram presentes ao longo dessa jornada. Eu não teria feito nada disso sem vocês.

Agradeço à UFRGS, seu corpo docente e técnico-administrativo e todos que trabalharam para manter a universidade funcional, limpa e segura. Agradeço imensamente também a sua estrutura de atividades extracurriculares , onde depositei as minhas maiores motivações e sonhos.

Por último, um carinho e agradecimento especial aos meus orientadores Leonardo e Paulo que estiveram presentes em todos os momento do meu trabalho para fornecer todo o apoio e conhecimento. Obrigado Paulo por sempre encontrar um tempo para se reunir comigo, tirar dúvidas, discutir e por ser tão prestativo e cuidadoso.

Se eu vi mais longe,
foi por estar sobre ombros de gigantes. (Isaac Newton)

Resumo

O objetivo deste trabalho é modelar e simular uma célula tridimensional composta por uma membrana celular constituída por multipartículas ativas interligadas por um potencial de mola, um potencial de flexão, um potencial de conservação da área total e outro de área local das faces do sólido e um potencial de conservação de volume para estabilização física. Para isso, foi construído um modelo de célula 3D que funciona a partir da ligação entre partículas que formam faces triangulares para compor a membrana celular. Após a simulação foram feitas medidas reológicas para entender o comportamento da célula frente a tensões externas aplicadas e interações com o seu ambiente. Para isso, foram revisados os principais *softwares* e *frameworks* utilizados para a simulação de sistemas multicelulares interagentes. Após, foi feita revisão dos principais modelos físicos e biológicos de sistemas multicelulares diretamente relacionados e este trabalho com aplicação em sistemas imune, movimento coletivo, câncer, cicatrização, transmissão e troca de proteínas em geral. Foi dada ênfase em trabalhos que exploram o uso de subelementos celulares para a criação de células. Na parte final estudamos em detalhe o comportamento global da célula como função dos diferentes potenciais envolvidos na sua construção e fizemos medidas reológicas para caracterizar e validar o sistema em que resultados qualitativos experimentais foram reproduzidos. Por fim, aproximamos o tempo de simulação real para um sistema multicelular com 11.340 partículas. No apêndice foi feita revisão dos métodos de aceleração por *hardware* em que foram exploradas e comparadas as principais vantagens entre as plataformas de computação paralela utilizadas para paralelizar códigos de simulação em placas de processamento gráfico (GPUs), unidades central de processamento (CPUs) e métodos para a criação de *clusters* entre esses dispositivos em uma rede distribuída para simulações futuras de sistemas multicelulares e células compostas por milhares de partículas.

Palavras-chaves: Células 3D. Simulação. Computação paralela. GPU. OpenCL.

Abstract

The aim of this work is to model and simulate a three-dimensional cell composed of a membrane that consists of active multiparticles interconnected by a bond potential, a bending potential, a total area conservation potential and another local area conservation potential of the faces and a volume conservation potential for physical stabilization. For that, a 3D cell model was built that works based on the particles bonding that form triangular faces to compose the cell membrane. After the simulation rheological measures were made to understand the cell behavior in the face of external stresses applied and interactions with the environment. For this, the main software and frameworks used for interacting multicellular systems were reviewed. After that the main physical and biological multicellular systems models related to this work with applications in immune systems, collective movement, cancer, wound healing, transmission and protein exchange in general were reviewed where emphasis was given in works that explore the use of subcellular element to create cells. In the last part of the work we studied in detail the general behavior of the cell as a function of the potentials involved in the three-dimensional model and we made rheological measurements to characterize and validate the cell in which experimental qualitative results were reproduced. Finally, we approximate the real time simulation for a multicellular system with 11,340 particles. In the appendix, a review was made in hardware acceleration methods where the main advantages were explored and compared among the parallel computing platforms that run codes in parallel on graphical processing units (GPUs), central processing units (CPUs) and methods to create clusters among those devices in a distributed network for future simulations of multicellular systems and cells composed of thousands of particles.

Key-words: 3D Cells. Simulation. Parallel programming. GPU. OpenCL.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	*Application Programming Interface*
AWS	*Amazon Web Services*
CUDA	*Compute Unified Device Architecture*
ESPResSo	*Extensible Simulation Package for Research on Soft Matter*
FLOPS	*Floating Point Operations Per Second*
GPGPU	*General-Purpose computing on Graphics Processing Units*
GPU	*Graphics Processing Unit*
MPI	*Message Passing Interface*
OpenACC	*for Open Accelerators*
OpenCL	*Open Computing Language*
ScEM	*Subcellular Element Model*

Sumário

Parte I

Introdução

1 INTRODUÇÃO

Processos celulares como migração, morte, divisão e diferenciação celular estão intimamente ligados a processos em tecidos, como a manutenção das funções normais do corpo humano adulto (homeostase), com o desenvolvimento embrionário (embriogênese) e com o desenvolvimento de tumores (Drasdo, Hoehme e Block (2007)). Sabe-se que a física da dinâmica celular tem papel importante nesses processos uma vez que as células precisam migrar e crescer através das diferentes estruturas tridimensionais do corpo humano (vídeo de células 3D *in vivo* se movimentando - Liu et al. (2018)). Tais processos podem ser realizados pelas células graças às características plásticas/elásticas de suas membranas e graças à reconstrução ativa dos seus citoesqueletos (Drasdo, Hoehme e Block (2007)). Além disso, as interações mecânicas sofridas pelas células podem induzir o destino de movimento e desenvolvimento celular, processo conhecido como mecanotransdução (Wang, Butler e Ingber (1993)).

Trabalho recente publicado na *Nature* por Saw et al. (2017) mostrou que pontos de alta deformação e de alto *stress* mecânico em monocamadas de células (estrutura *quasi*-2D de células) (ver figura 1) induzem morte celular por apoptose. Anos antes, Vedula et al. (2012) mostraram que diferentes restrições geométricas induzem diferentes modos coletivos de migração também em monocamadas de células. Trabalhos como esses permitem entender, em aproximações *quasi*-2D, a complexidade e a importância da física e das interações mecânicas nos sistemas biológicos reais em 3D. Leis de interações e equações físicas que ainda são pouco entendidas.

Abordagens de experimentos *in silico* têm sido usadas como complementos às medidas experimentais e teorias celulares nas últimas décadas possibilitando que sejam identificados mecanismos físicos por trás das ações celulares (Camley e Rappel (2017)). Entre as vantagens da utilização de simulações em relação aos experimentos *in vitro* e *in vivo* é que são menos custosas laboralmente e temporalmente, além de permitir que sejam feitas mudanças em qualquer instante da simulação como adição, remoção de células e tecidos e a investigação dessas mudanças na dinâmica do sistema. Também permitem maior controle sobre as condições do sistema através dos parâmetros do modelo.

Modelos de matéria ativa em 2D são muito utilizados para o entendimento da dinâmica coletiva de células (Vicsek et al. (1995) e Szabo et al. (2006)), entretanto falham na representação de células individuais e limitam o movimento das células em apenas duas direções, mudando assim seu comportamento e número de interações, como ocorre no caso tridimensional. Recentemente modelos 3D foram utilizados para entender o funcionamento do sistema imune (Cappuccio,

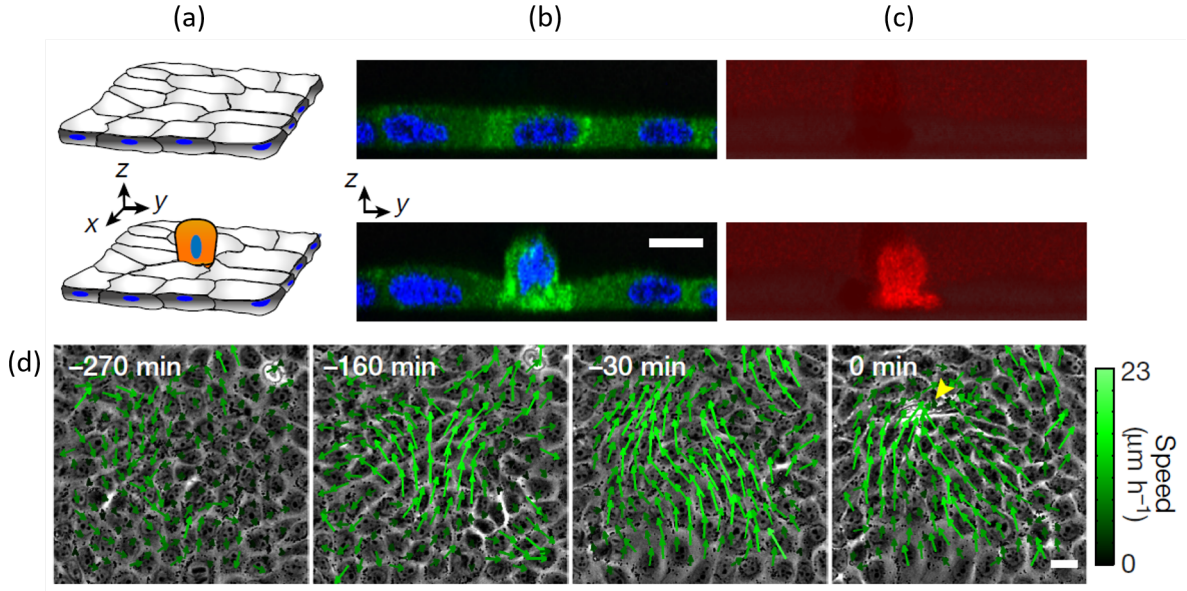


Figura 1 – Esquemático (a) e experimental (b-c-d) mostrando deformações e *stress* induzindo apoptose. Em (a) a célula na cor laranja sofre extrusão e apoptose após as células fluírem na direção dela. O mesmo ocorre experimentalmente em (b), onde é possível ver o núcleo em azul e em (c) em que a célula que sofre apoptose está em ênfase em vermelho mais claro. Em (d) é mostrado ao longo do tempo o campo de velocidades do grupo de células que têm suas direções no sentido das células brancas que sofrem apoptose. Adaptado de Saw et al. (2017).

Tieri e Castiglione (2016)) através de mecanismos de trocas de proteínas e comunicação celular, mas não englobam o caráter adaptativo do formato da célula. Modelos que consideram a célula como um aglomerado de subelementos (Newman (2005)), partículas (Gardiner et al. (2015)), elementos finitos Zhao et al. (2017) foram utilizados para entendimento da formação de tecidos e comportamentos reológicos como comportamentos plásticos e elásticos, entretanto, do ponto de vista do movimento coletivo em 3D, são custosos computacionalmente, pois consideram as células preenchidas com esses componentes, o que limita a expansão do modelo para um sistema multicelular, pois levaria em conta todos os componentes internos da célula para os cálculos de evolução temporal.

Uma alternativa que também permite deformações no formato das células e que acredita-se ser computacionalmente mais vantajoso é o modelo que utiliza potenciais de curvatura (*bending*) desenvolvido por nosso grupo de pesquisa (Teixeira, Fernandes e Brunnet (2021)). Nesta abordagem, uma célula é representada apenas por um anel de partículas ativas (vídeo de uma célula livre se movendo) presas entre si por um potencial derivado da física de polímeros (Mousavi, Gompper e Winkler (2019)). O modelo é bastante robusto frente às interações mecânicas (como podem ser vistos nos vídeos 1 e 2) sem a necessidade do *bulk* preenchido por partículas. Uma adaptação do modelo está sendo desenvolvida por Ourique, Teixeira e Brunnet (Physica A, Dez. 2021), também do nosso grupo, onde um núcleo é adicionado ao modelo e está sendo utilizado para complementar o experimento *in vitro* (vídeo do experimento) de Tlili

(2015). Como pode ser visto no vídeo da simulação, os movimentos e deformações das células são semelhantes ao experimentos.

Apesar de reproduzirem os comportamentos experimentais, ambos os modelos enfrentam a limitação de serem bidimensionais, o que impossibilita a exploração de importantes aspectos pertencentes ao caso tridimensional em que as células têm maior grau de liberdade para realizar diferentes modos de movimento.

Dadas essas limitações, este trabalho propõe a implementação de um modelo para o caso 3D de forma a incorporar maiores graus de liberdade para o movimento celular e deformações nas três dimensões mantendo a economia do custo computacional em relação a modelos já existentes (Newman (2005), Gardiner et al. (2015) e Zhao et al. (2017)). A simplicidade do modelo, por não incorporar estruturas internas da célula, permite maior rapidez e eficiência nos cálculos visando escalabilidade da simulação para maiores números de partículas e visando uma projeção futura de simulação para um sistema multicelular.

1.1 Motivação

A motivação do trabalho se dá pela necessidade de incorporar aspectos tridimensionais na modelagem computacional de células que possibilitem o estudo das deformações e a reprodução dos resultados experimentais. A incorporação desses aspectos são ainda mais importantes para um futuro cenário de modelo multicelular, pois se sabe que os comportamentos derivados deles estão intimamente conectados a processos metabólicos como embriogênese, migração, divisão e morte celular e o desenvolvimento de tumores.

1.2 Objetivos

O objetivo do trabalho é criar uma ferramenta que possibilite o estudo da dinâmica de células tridimensionais em um cenário multicelular. Para isso foi criada uma célula tridimensional como uma esfera flexível composta de uma membrana externa de multipartículas ativas (a superfície da esfera) (ver Figura 2). Para criar a membrana celular foi utilizado modelo de célula 3D composta por ligações entre partículas em que foram realizadas as adaptações necessárias para que o formato da célula seja estável e possibilite interações com estruturas externas. Ademais, foram feitas medidas reológicas virtuais (dependência das características mecânicas da célula com comportamento mais gerais como deformações, viscosidade etc) como tensão (*stress*) e deformação (*strain*) de forma que o modelo proposto possa ser comparado a modelos usuais e a dados experimentais da reologia celular, o que permite definir a proximidade do modelo com células reais, assim como as suas limitações. As caracterizações reológicas foram feitas em função dos parâmetros do modelo, o que gerou o espaço de parâmetros da célula, ponto principal que caracteriza o alcance do objetivo proposto.

Em relação ao cálculo computacional, foram utilizados códigos em *python* (80% do

código) e *Fortran 95* (20% do código) com o objetivo de ter a maior eficiência e controle do algoritmo, além da maior rapidez dos cálculos, aumentando a *performance* da simulação. Por fim, no apêndice, foi feita uma revisão de métodos de computação paralela para um futuro cenário de simulação de interações multicelulares com milhares de partículas.

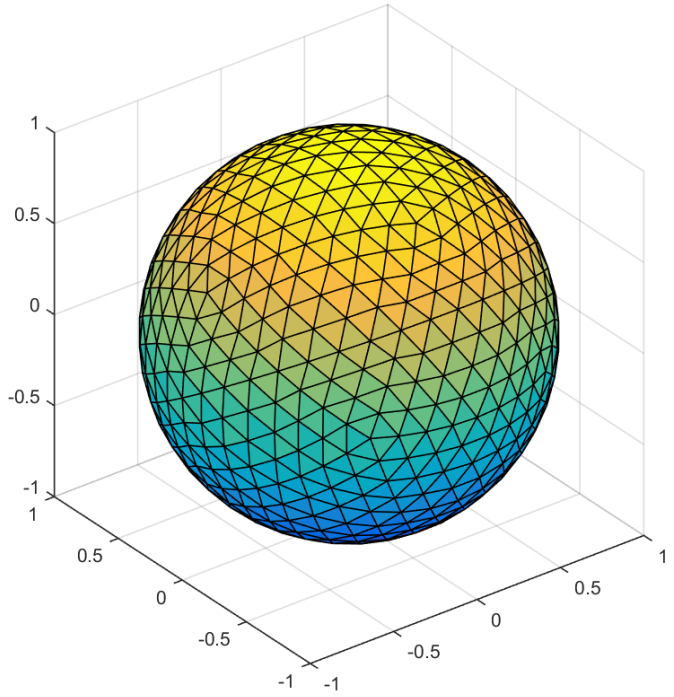


Figura 2 – Esfera composta por tecelagem de triângulos (Comunidade Matlab (2015)). A membrana celular é formada pela esfera de forma que cada vértice dos triângulos é representado por uma partícula.

Parte II

Revisão bibliográfica

2 Revisão bibliográfica

Uma vez que a proposta deste trabalho é modelar a membrana celular de uma célula 3D, torna-se necessária uma metodologia para a criação dessa estrutura composta por partículas, além de medidas para a validação das estruturas criadas com os comportamentos reológicos de células reais. Também, deve-se ter a possibilidade para a implementação de equações diferenciais para a modelagem do citoesqueleto e outras estruturas para futuros trabalhos.

O maior desafio da proposta é desenvolver uma programação ou adaptar uma metodologia, *software* ou *framework* que seja rápido para realizar a simulação da dinâmica dessas estruturas da célula e possibilite escalabilidade para sistemas multicelulares a longo prazo.

Por isso, como primeira abordagem, buscou-se na literatura por *softwares* e *frameworks* utilizados para a simulação 2D e 3D de células e por metodologias para a criação de estruturas 3D a partir de modelos utilizados em simulações físicas. Ao final foi feita uma revisão por ferramentas de aceleração de códigos por *hardware* que podem complementar a implementação das metodologias escolhidas. Na revisão em geral foi dada ênfase na pesquisa por *softwares* e *frameworks* abertos e modulares que permitem o posterior uso e modificação por futuros trabalhos.

2.1 Revisão de *softwares* para simulação 3D de células

Através de uma revisão inicial na literatura, foi observado que no campo de pesquisa da biologia são utilizados programas de simulação para o estudo *in-silico* das interações celulares 3D. Entre os principais *softwares* utilizados estão CellStudio (Liberman et al. (2018)), Compucell3D (Izaguirre et al. (2004)), FLAME-GPU (Richmond e Chimeh (2017)) entre outros (Liberman et al. (2018)). Cada software possui as suas vantagens e desvantagens, como a capacidade de ser interativo, possuir implementação do meio biológico, escalabilidade do modelo utilizado etc.

Foi levantado o questionamento de que talvez esses *softwares* pudessem ter as ferramentas necessárias para a modelagem proposta por esse trabalho. Então, foi feita revisão dos *softwares* mais alinhados com o objetivo desse trabalho para responder esse questionamento.

Iniciando a pesquisa por um dos softwares mais atuais na simulação biológica 3D que utiliza tecnologia de jogos para as interações: *CellStudio* (Izaguirre et al. (2004)). Neste *software* são realizadas simulações entre células 3D através de um modelo de agentes (Fachada, Lopes e Rosa (2007)) para a evolução temporal. O programa possui uma interface gráfica que

permite que sejam definidos parâmetros de inicialização e a visualização da simulação, onde é possível observar as células interagindo e realizando processos como mitose, diferenciação etc. O principal diferencial em relação aos outros *softwares* é a possibilidade do usuário fazer alterações no ambiente em tempo real de simulação, como a inserção e exclusão de células e observar as mudanças ocasionadas pelas alterações. Neste link é possível observar um vídeo da simulação feita pelo *software*. Apesar dessas características, o *software* não se mostrou útil para este trabalho uma vez que as células são representadas por agentes individuais e, para a simulação gráfica, são consideradas como esferas perfeitas. Para adaptar o *software*, seria necessário alterar o código fonte, que foi desenvolvido na ferramenta de criação de jogos *Unity 3D*.

Outro *software* muito conhecido em simulações 3D é o *CompuCell3D* (Izaguirre et al. (2004)). O programa utiliza um modelo de Potts (Wu (1982)) para a evolução temporal. Atualmente, o *software* possui técnicas de aceleração por GPU e também permite que a simulação seja feita em nuvem no *Amazon Web Services* (AWS). O *software* possui diversas técnicas implementadas para aceleração da simulação, entretanto, por ser utilizado um modelo predefinido para a simulação, também seria necessário editar o código fonte para o objetivo requerido.

Identificou-se, na revisão, que *frameworks* também são utilizados nas pesquisas *in-silico*. A vantagem dos *frameworks* é que eles permitem que outros *softwares* sejam criados a partir das suas ferramentas, assim gerando maior flexibilidade na criação de novos modelos, fato que direcionou as revisões seguintes.

2.2 Revisão de *frameworks* para simulação 3D de células

Um *framework* muito utilizado é o FLAME (*Flexible Large-scale Agent-based Modelling Environment*) que, em uma de suas aplicações, foi utilizado para simular o crescimento de uma epiderme 3D (Adra et al. (2010)) através da interação de células que podem realizar diferentes processos como diferenciação, migração etc. Entretanto o FLAME tem foco nas ferramentas para simulações de modelos de agentes, assim facilitando que outros *softwares* que utilizem esse modelo sejam criados a partir dele. Caso se quisesse utilizar um modelo diferente, a programação se tornaria complexa, pois seriam necessárias alterações no código fonte em relação à modelagem e também seria necessário adaptar as outras ferramentas do *framework* para o novo modelo, por isso não se mostrou interessante para o objetivo proposto.

Após uma pesquisa na literatura por *frameworks* para a simulação de partículas, o *Blender* (Community (1998)) destacou-se pelo uso da tecnologia de jogos em que são utilizados polígonos, que seriam análogos às partículas, para a criação de quaisquer tipos de superfícies. É possível aplicar diferentes elementos físicos às superfícies criadas, tais como: colisões, gravidade, deformações plásticas, entre outras. É possível implementar conceitos físicos e novos objetos a partir da adição de novos módulos de programação conhecidos pela comunidade de colaboradores como *addons*. Os módulos são criados em *Python*. Alguns *addons* criados recentemente modelam objetos e as suas deformações por partículas. Também seria possível realizar os cálculos e a

renderização a partir de GPU com a ativação de uma opção nativa do programa. Dentre tantas possibilidades oferecidas pelo *Blender*, utilizamos as ferramentas através da interface gráfica e criamos estruturas semelhantes a células (figura 3).

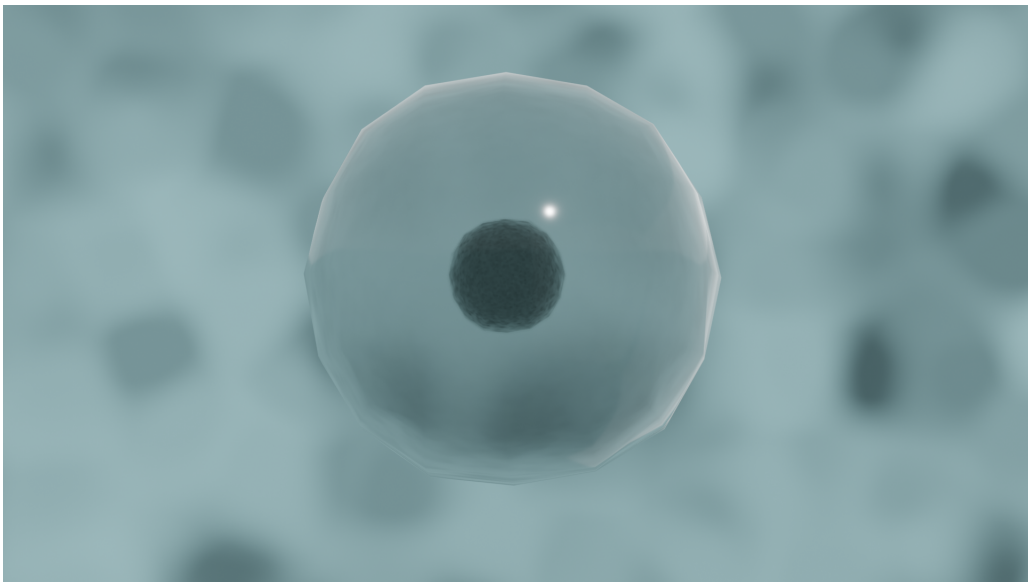


Figura 3 – Célula 3D criada no *framework* Blender.

Após, adicionamos propriedades físicas como deformações plásticas, ação da gravidade e colisões. Assim, é possível ter uma ideia prática do potencial de uso do *framework* para a proposta. No vídeo 1 é possível observar a célula única interagindo com o meio através de colisões, o que abre possibilidades para a simulação. Então, buscando interações mais complexas, adicionamos mais células, entretanto agora sem núcleo, apenas para visualizar as interações de superfície da membrana. No vídeo 2 é possível ver que as membranas interagem e, após as colisões, preenchem o espaço da superfície. Isso abre possibilidades para físicas mais complexas de interação, algo que será necessário no modelo proposto. Por último, buscando verificar interações entre as membranas e os núcleos, fizemos a simulação do vídeo 3, onde se tem resultados semelhantes, entretanto, dessa vez, é possível perceber, no segundo *02* do vídeo 3, que o núcleo sai da membrana, assim se tornando necessários maiores entendimentos sobre a física do programa e, principalmente, das equações envolvidas para dar o próximo passo na definição de uso da ferramenta.

Dentro desse contexto, notou-se que seria necessário avaliar o código fonte do programa para entendimento desses detalhes e que novos ingredientes físicos requeridas pelo modelo poderiam ser adicionados pela criação de *addons*. Seria necessário também otimizar o máximo possível os outros objetos já inseridos no ambiente de simulação com o objetivo de tornar o código o mais rápido possível. Entretanto, em um primeiro momento essa opção não se torna atrativa frente à programação criada do zero em um código próprio com a linguagem mais apropriada para cálculos numéricos como, por exemplo, a linguagem Fortran e linguagens com orientação a objetos como Python e C++, que dão a possibilidade de manipulação dos objetos em

níveis mais primitivos, ou seja, permite que o código seja otimizado em nível de instruções de programação e ter controle sobre a física do sistema, algo que com o *Blender* seria mais difícil.

Outro *framework* muito interessante é o ESPResSo (Weik et al. (2019)), que é mantido pelo *Institute for Computational Physics* da *University of Stuttgart* e tem como funcionalidade principal simular modelos de partículas para *soft matter*. É de código aberto, com diversos modelos prontos para serem utilizados, possui uma interface programável na linguagem de *script* TCL ou Python que roda o código em C permitindo a paralelização em CPU e GPU[1]. O *framework* teria as ferramentas para construir o modelo proposto, entretanto permite menor controle sobre a escolha de paralelização do código em relação ao *hardware* e, por isso, acreditamos que essa característica impedirá a máxima otimização de uso do *hardware* para a aceleração da simulação em cenários multicelulares futuros.

2.3 Revisão de modelos de células

A maioria dos modelos baseados em indivíduos podem ser caracterizados dentro de duas categorias: os baseados em *lattice* e os não baseados em *lattice* (Byrne e Drasdo (2009)). Os baseados em *lattice* utilizam uma matriz que descreve o espaço e, a partir das interações definidas pelo modelo, os indivíduos, que podem representar células individuais ou partes da célula, se movem entre as posições da matriz, sendo então um modelo discreto no espaço. Baseado nisso, foi dado foco na pesquisa de modelos não baseados em *lattice*, pois são mais próximos da proposta desse trabalho.

Os principais modelos físicos de movimento coletivo celular ao longo do tempo podem ser observados na figura 4. Os primeiros, bem conhecidos, são os modelos isotrópicos, a exemplo do modelo de Szabo et al. (2006), que modela as células por partículas individuais isotrópicas e autopropulsoras submetidas a ruído. A evolução temporal desse modelo reproduz movimentos coletivos, transições de fase e outros comportamentos de sistema celulares. Existem também os modelos de partículas deformáveis (Menzel e Ohta (2012)). Nesse modelo cada célula tem uma geometria que, em geral, é uma área circular ou em forma de elipse com eixos bem definidos. As interações mecânicas e forças entre diferentes células são distribuídas nesses eixos que, por sua vez, influenciam na dinâmica do movimento coletivo das células. Nos modelos de interações de *Voronoi* (Li e Sun (2014)), as células são criadas por uma tecelagem de *Voronoi* onde cada polígono representa uma célula. A dinâmica de movimento é aplicada ao centro de massa do polígono. Por ser um modelo sem espaços vazios entre as células, é mais indicado para a modelagem de tecidos confluentes. O modelo de Potts (Wu (1982)), o qual é um modelo baseado em *lattice*, considera uma matriz 2D ou 3D com posições fixas no espaço que incorporam as células. Cada elemento da matriz pode ser considerado uma célula ou parte de uma célula que pode se movimentar na matriz a partir das interações com outras células que são definidas por regras de interação. Esses modelos, em geral, fazem uso de um Hamiltoniano que governa a

[1] http://espressomd.org/wordpress/about/summary/

dinâmica de movimento celular.

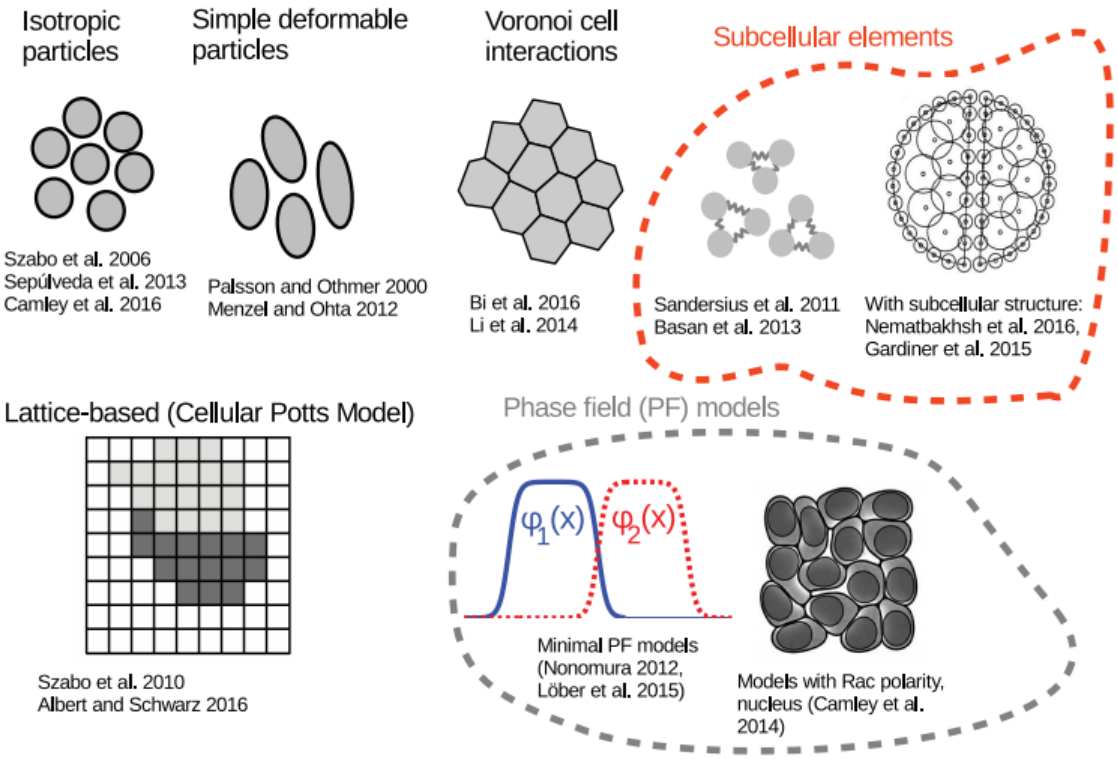


Figura 4 – Modelos físicos de movimento coletivo celular (Camley e Rappel (2017).

Existem também modelos de campo de fase (Ziebert, Swaminathan e Aranson (2012), Moreira-Soares et al. (2020)) que implementam uma equação de campo que varia entre 1 (dentro da célula) e 0 (fora da célula) de forma suave entre as bordas. Por último, e bem recentes, são os modelos que consideram uma célula composta por subelementos - *Subcellular Element Model (ScEM)* - (Newman (2005)) que são representados por vértices interconectados em uma rede. A coesão dos subelementos é dada por um potencial intracelular para os subelementos da mesma célula e potenciais intercelulares para interação com subelementos de outras células. Na figura 5 pode-se ver um agregado desses subelementos. Nesse modelo toda célula é preenchida por uma rede de subelementos interconectados. O *bulk* da célula também é formado por subelementos. Esse é o modelo mais próximo da proposta deste trabalho, uma vez que considera uma célula composta por subelementos, que é uma ideia análoga à célula composta por partículas.

Após sua primeira formulação para a modelagem de células 2D (Newman (2005)), o modelo foi estendido para o caso 3D e, como primeiro passo, foram realizados experimentos virtuais (vídeo de um experimento virtual com 1024 subelementos) em que forças forças de compressão e tensão foram aplicadas em uma célula única e medidas reológicas como diagrama de tensão-deformação, deformações ao longo do tempo para tensões constantes, entre outras, foram feitas para validação com os dados experimentais. Nessa formulação foi implementada uma dinâmica de *Langevin* para os subelementos. Ademais, não foi implementado nenhum tipo de gradiente ou polarização para dinâmica da célula, o que caracteriza uma modelagem sem

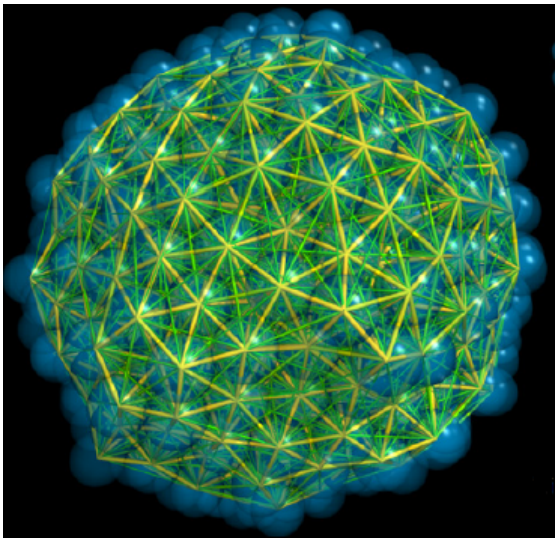


Figura 5 – Agregado de subelementos. Círculos foram desenhados nos vértices (Sandersius e Newman (2008)).

mecanismo de migração ordenado. O modelo reproduziu os resultados da literatura para média escala de tempo e tensões moderadas.

O modelo ScEM foi expandido para uma formulação ativa dos subelementos em Sandersius, Weijer e Newman (2011). O caráter ativo dos subelementos permite que a célula realize movimentos mais próximos ao movimento amebóide. O modelo reproduziu resultados experimentais relativos ao comportamento viscoso de tecidos, resposta a pequenos alongamentos e padrões de fluxo celular em resposta a gradiente quimiotáticos, diferente do modelo anterior que é mais limitado por ter natureza estática.

Uma abordagem similar ao ScEM emprega dinâmica de partículas dissipativas (DPD, na sigla em inglês) para simular uma célula 3D com apenas um núcleo conectado por molas a diversas partículas que modelam a membrana celular (Moreira-Soares et al. (2020)). O trabalho utiliza o *framework* ESPResSo para simular o movimento de uma célula que interage com estruturas de proteínas. O trabalho reproduz resultados experimentais de velocidade de migração de células dentro de um meio com alta densidade de proteínas. No entanto, as partículas da membrana interagem entre si somente com potencial de exclusão de volume, não apresentando uma ordem estrutural, limitando assim a exploração de comportamentos reológicos.

Analogamente existe o modelo recente de Gardiner et al. (2015), em que uma coleção de partículas representa uma célula onde cada partícula é uma pequena porção da célula. Também, grupos de partículas podem representar as diferentes organelas. As partículas interagem via um potencial de mola não linear e o modelo possui parâmetros que definem os comportamentos elásticos e dinâmicas reológicas das células, o que permite que as células os parâmetros sejam calibrados via dados experimentais.

Outra abordagem utilizada na simulação de células é a abordagem de elementos finitos. Nessa abordagem, a célula é composta por um conjunto de polígonos cujos vértices externos

estão localizados na membrana. O trabalho de Zhao et al. (2017) apresenta uma simulação 2D (DyCelFEM) onde os polígonos formam uma rede interconectada no interior da célula que garante a sua estabilidade. Utiliza-se uma lei matemática para a criação dos polígonos de tal forma que, dependendo das deformações da célula, novos polígonos sejam criados para adaptar a célula ao novo estado. Outra abordagem interessante foi feita por Tóthová, Jančigová e Bušík (2015), que utiliza o *software* chamado *Object-in-fluid* do *scientific software ESPResSo*. Nessa abordagem, uma hemácia 3D é criada a partir do método *tringular mesh* para estudar a elasticidade da membrana celular. No entanto, as considerações energéticas e de fluídica utilizadas deixam o modelo mais complexo do que o necessário para o propósito deste trabalho.

Sabe-se, a partir dessa revisão, que células com deformação adaptativa criadas por modelos ativos de subelementos reproduzem resultados experimentais relativos à reologia celular para um modelo 3D e viscosidade de tecidos para o modelo 2D, fatores importantes para o entendimento do movimento coletivo. Entretanto, por serem modelos que utilizam subelementos no *bulk* das células (Newman (2005), Zhao et al. (2017), Gardiner et al. (2015)), aumentam muito o custo computacional para maiores números de células no sistema. Tais abordagens provavelmente apresentariam problemas de escalabilidade em simulações 3D e multicelulares, com um custo alto de processamento para um tempo real correspondente pequeno.

O modelo do trabalho de Moreira-Soares et al. (2020), apesar de utilizar partículas, possui uma estrutura de membrana que limita a análises das respostas mecânicas e reológicas. O modelo de Tóthová, Jančigová e Bušík (2015), que utiliza *triangular mesh* para a criação de hemácias 3D, por outro lado, permite o estudo de elasticidade e deformação da membrana celular, porém apresenta complexidades no modelo que tornam mais custosa a implementação do algoritmo, o que prejudica a escalabilidade para um cenário multicelular.

Modelos mais simples que utilizem partículas que permitam o estudo de deformações e que tenham baixo custo computacional e de implementação são interessantes do ponto de vista do movimento coletivo, pois possibilitam estudar fatores importantes da reologia celular. Ao mesmo tempo, permitem que o modelo seja utilizado no regime multicelular no caso tridimensional por não demandarem tantos cálculos.

Recentemente foi proposto um modelo por nosso grupo de pesquisa para a dinâmica de células 2D que atende aos requisitos acima. O artigo de Teixeira, Fernandes e Brunnet (2021) trata de um modelo de matéria ativa que considera a célula como uma membrana circular (anel) composta por N partículas ativas interligadas por $N-1$ ligações e sujeitas a um potencial de curvatura que impede o colapso do anel (figura 6). Além dos potenciais de ligação (*bond*) e curvatura (*bend*), existe o potencial de exclusão de volume (*EV*), que impede que as partículas se sobreponham. O sistema é governado por uma dinâmica de *Langevin* superamortecida dada por:

$$\dot{\mathbf{r}}_\mathbf{i}(\mathbf{t}) = v_0 \hat{n}_i - \mu \sum_{i \sim j} \nabla U(\mathbf{r}_\mathbf{i}) + \sqrt{2D_T}\,\boldsymbol{\chi}_\mathbf{i}(\mathbf{t}) \tag{2.1}$$

$$\dot{\theta}_i(t) = \frac{1}{\tau'} \arcsin(\hat{n} \times \frac{\vec{v}}{|v|} \cdot e_z) + \sqrt{2D_R}\xi_i(t) \tag{2.2}$$

Onde $\mathbf{r_i(t)} = (x_i(t), y_i(t))$ é a posição da partícula i no tempo t, μ a mobilidade, v_0 o módulo da velocidade autopropulsora, $\hat{n}_i = (\cos\theta_i(t), \sin\theta_i(t))$ a orientação da velocidade autopropulsora descrita pelo ângulo θ_i que relaxa em direção a v_i com um tempo característico τ', enquanto é sujeito ao ruído angular gaussiano branco $\xi_i(t)$ de média zero e segundo momento $\langle \xi_i(t_1)\xi_i(t_2)\rangle = \delta_{ii}\delta(t_1 - t_2)$ independente para cada partícula a cada passo de tempo. D_R é o coeficiente de difusão rotacional que define uma escala de tempo $\tau_R \equiv 1/D_R$. Quando o ruído térmico está presente, cada partícula é sujeita ao ruído branco gaussiano de média zero e variância $\langle \chi_\mathbf{i}(\mathbf{t_1}) \cdot \chi_\mathbf{i}(\mathbf{t_2})\rangle = 2\delta_{ii}\delta(t_1 - t_2)$. D_T é o coeficiente de difusão térmico que define a escala de tempo $\tau_T \equiv \sigma^2/D_T$ para uma partícula que difunde em uma largura da ordem de seu tamanho σ.

As forças resultantes são derivadas da soma dos potenciais (U), que são:

$$U = U_{bond} + U_{bend} + U_{EV} \ . \tag{2.3}$$

De forma que o potencial de *bond* é dado por:

$$U_{bond} = \frac{k}{2} \sum_{j=1}^{i+1} (|\mathbf{d_j}| - r_0)^2 \ , \tag{2.4}$$

$\mathbf{d_j} = \mathbf{r_j} - \mathbf{r_{j-1}}$ é o vetor que conecta consecutivamente as partículas do anel (figura 6), k é a constante da mola, r_0 a distância de equilíbrio da ligação. O potencial de *bend* é dado por:

$$U_{bend} = \frac{k_b}{2} \sum_{j=1}^{i+2} \frac{\mathbf{d_j} \cdot \mathbf{d_{j-1}}}{|\mathbf{d_j}||\mathbf{d_{j-1}}|} \ , \tag{2.5}$$

onde, k_b é a rigidez de *bending*. E o potencial U_{EV} de exclusão é dado por:

$$U_{EV} = \frac{\varepsilon}{12} \begin{cases} (\sigma/r_{ij})^{12} - (\sigma/r_{ij})^6, & \text{se } r_{ij} < 2^{1/6}\sigma \\ 0, & \text{se } r_{ij} \geq 2^{1/6}\sigma \end{cases} \tag{2.6}$$

$r_{ij} = |\mathbf{r_i(t)} - \mathbf{r_j(t)}|$, ε, σ são a distância entre as partículas i e j, a energia característica do volume de exclusão e o diâmetro efetivo de uma dada partícula, respectivamente.

Como resultados, o trabalho caracteriza a dinâmica e a morfologia das células nos diferentes regimes de movimento e estabelece as condições para a emergência de movimentos coletivos no sistema, que podem ser translacionais ou rotacionais. Estabelece as condições de deformação da célula através de parâmetros do modelo e ruídos térmicos e angulares, e encontra soluções analíticas em certos limites de parâmetros.

O modelo foi expandido por Ourique, Teixeira e Brunnet (Physica A, Dez. 2021) onde

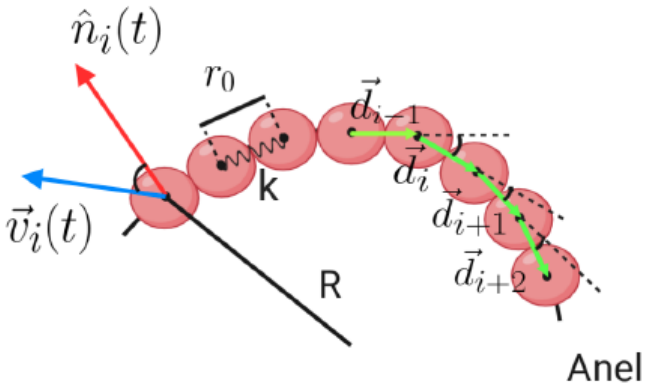


Figura 6 – Representação esquemática de parte da célula 2D (Teixeira, Fernandes e Brunnet (2021)).

um núcleo foi adicionado dentro do anel de partículas e uma força de conservação de área foi implementada com a aspiração de simular agregados celulares. Movimentos confluentes qualitativos semelhantes aos experimentais foram alcançados até o momento. Os vídeos da simulação e experimento podem ser vistos no capítulo 1.

Para desenvolver a proposta deste trabalho, foi feita uma adaptação do modelo de Tóthová, Jančigová e Bušík (2015), que implementa cinco potenciais para manter a estrutura estável sob aplicações de forças externas. O modelo proposto se encontra no capítulo 3. De forma complementar, torna-se necessário desenvolver o *software* da forma mais otimizada e acelerada possível uma vez que o caso tridimensional engloba mais partículas e graus de liberdade que o caso bidimensional. Por isso, considerando um futuro cenário de simulações multicelulares e milhares de partículas, foi feita uma revisão dos *softwares* e *frameworks* (apêndice A) que permitem paralelizar os cálculos das forças entre as partículas nas unidades de processamento de placas de processamento gráfico (GPU).

Parte III

Metodologia

3 Modelo de célula 3D

3.1 Célula 3D composta por multipartículas ativas

O modelo utilizado é uma adaptação do modelo de Tóthová, Jančigová e Bušík (2015), em que a célula é composta de uma membrana celular sujeita a cinco potenciais que, entre outras funções, estabilizam a estrutura celular e transferem as forças aplicadas à sua estrutura, funcionando como um pseudo-citoesqueleto. A membrana celular é uma esfera flexível composta por um conjunto $\{P_m\}$ de N partículas p_j ativas, ou seja, que possuem dinâmica própria sem a necessidade de forças externas. Todas as p_j partículas da membrana têm a mesma massa m. Na figura 7 é possível ver a ilustração da célula composta por multipartículas na membrana e uma partícula maior que representa o núcleo da célula.

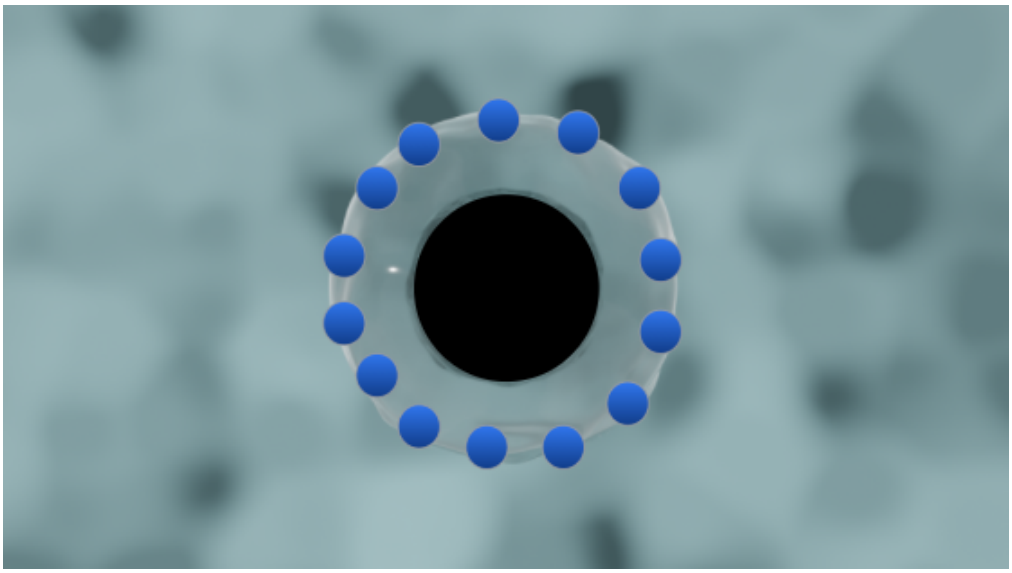


Figura 7 – Ilustração do modelo de célula 3D composta por partículas (corte bidimensional).

Foi implementada a dinâmica de Langevin superamortecido para as partículas que compõem a membrana da célula (equação 3.1) :

$$\lambda \dot{\mathbf{r}}_j(t) = \eta \xi_j(t) - \nabla U_{membrana} \tag{3.1}$$

em que $\dot{\mathbf{r}}_j$, $\mathbf{r}_j$ são, respectivamente, a velocidade e a posição da partícula p_j da membrana celular. λ, η são o coeficiente de viscosidade relacionados às partículas da membrana e a intensidade do ruído, que pode ser interpretado como uma temperatura efetiva interna da célula. $\boldsymbol{\xi}_\mathbf{j}(\mathbf{t})$ é o

ruído branco gaussiano, de média zero ($\langle \xi_{\mathbf{j}}(\mathbf{t}) \rangle = 0$) e delta correlacionado ($\langle \xi_{\mathbf{j}}(\mathbf{t_1}) \cdot \xi_{\mathbf{j}}(\mathbf{t_2}) \rangle = 2\delta_j(t_1 - t_2)$). $\nabla U_{membrana}$ é o potencial a que a partícula p_j da membrana é submetida (equação 3.2).

O potencial $U_{membrana}$, que sente uma partícula p_j pertencente à membrana $\{P_m\}$, é dado por:

$$U_{membrana} = U_{bond} + U_{bend} + U_{vol} + U_{sup} + U_{face} \tag{3.2}$$

onde $\Delta r_{j,k} = |\mathbf{r_j}(t) - \mathbf{r_k}(t)|$ é a distância entre as partículas p_j e p_k, U_{bond} é o potencial harmônico que conecta as partículas da membrana, U_{bend} o potencial de flexão (*bending*) que sustenta a estrutura da membrana, U_{vol} o potencial de conservação do volume, U_{sup} potencial de conservação da área de superfície e U_{face} o potencial de conservação da área das faces triangulares, em que a soma considera a contribuição dos potenciais de todas as partículas p_k pertencentes ao grupo P_m onde $k \neq j$.

Os potenciais são descritos a partir das suas respectivas forças, em que $F = \nabla U$. Para a definição, primeiro é necessário entender a estrutura de distribuição das partículas na membrana. Para construir a membrana foi utilizado um algorítimo recursivo de tecelagem triangular que constrói uma esfera a partir de um icosaedro regular (figura 8). A posição de cada vértice do icosaedro é associado à posição de cada partícula. Essa conformação resulta em que a maioria das partículas terá 6 primeiros vizinhos, a não ser pelas 20 partículas associadas aos vértices originais do icosaedro, que terão 5 primeiros vizinhos.

A vista superior da composição geral da célula por partículas pode ser vista na figura 9 em que, no centro, na parte mais próxima da vista do observador, tem-se a partícula de referência em vermelho e, conforme as cores ficam mais frias, próximas do azul, as partículas vão ficando mais longe da partícula referência.

A força de ligação (F_{bond}) mantém a distância média entre as partículas da membrana estáveis. A força de ligação que age na partícula j é dada por:

$$\mathbf{F}_{bond,j} = k_s \sum_{i=1}^{m=5,6} \frac{(|\mathbf{r_j} - \mathbf{r_i}| - r_0)}{r_0} \mathbf{n_{ji}} \tag{3.3}$$

Em que m é o de partículas vizinhas da partícula (j), que será igual a 5 ou 6, k_s é a "constante da mola", $|\mathbf{r_j} - \mathbf{r_i}|$ é a distância entre a partícula j e sua vizinha i (figura 10 (a)), r_0 é a distância de equilíbrio entre as partículas e $\mathbf{n_{ji}}$ é o vetor unitário na direção da partícula i a j.

A força de flexão (F_{bend}) impede que a membrana da célula recaia sobre si mesma, ou seja, impede que as faces da membrana se sobreponham, tornando a membrana mais estável frente à ação de forças externas.

A força de flexão que age na partícula j, vértice A do triângulo ABC (figura 10 (b)) é

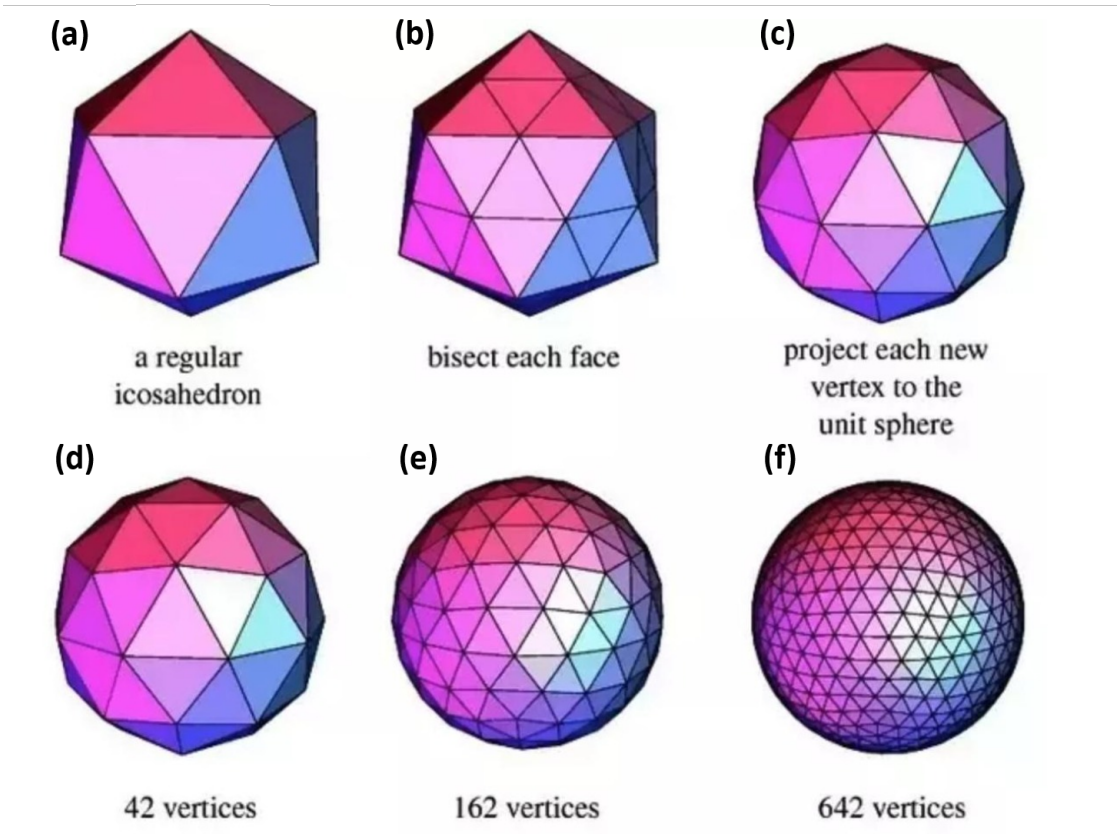


Figura 8 – Inicia-se com um icosaedro regular (a), divide-se as faces triangulares do icosaedro em 4 novos triângulos (b) e projeta-se (normaliza-se) os novos vértices em uma esfera de tamanho unitário (c). Após se faz o processo novamente de forma que se tem maior aproximação de uma esfera perfeita (d,e,f) (Comunidade Stack Overflow (2018)).

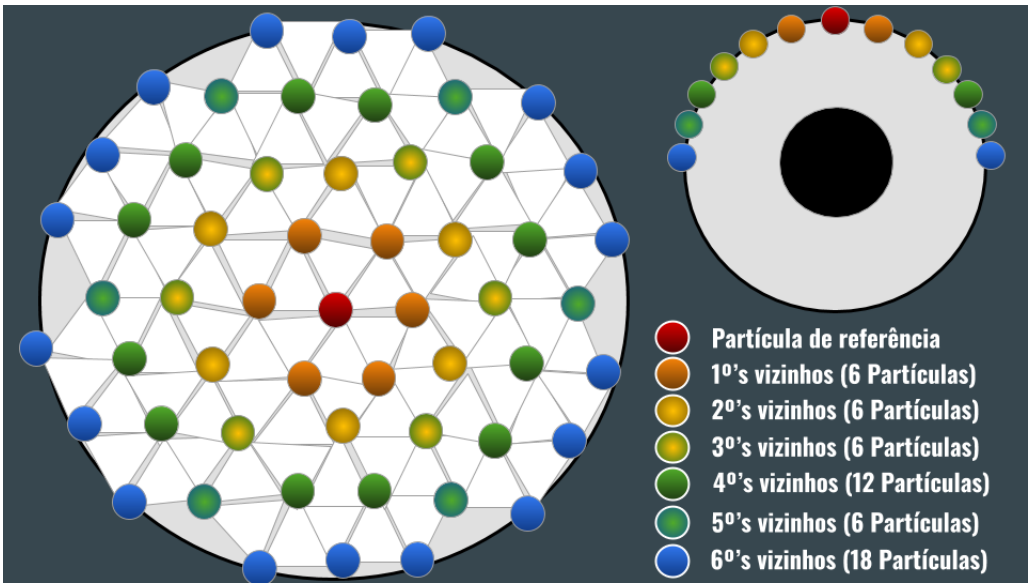


Figura 9 – Ilustração da vista superior do modelo de célula. Crédito da imagem ao Paulo C.G.

dada por:

$$\mathbf{F}_{bend,j}(ABC) = k_b \frac{(\theta_k - \theta_0)}{\theta_0} \mathbf{n}_{\mathbf{ABC}} \tag{3.4}$$

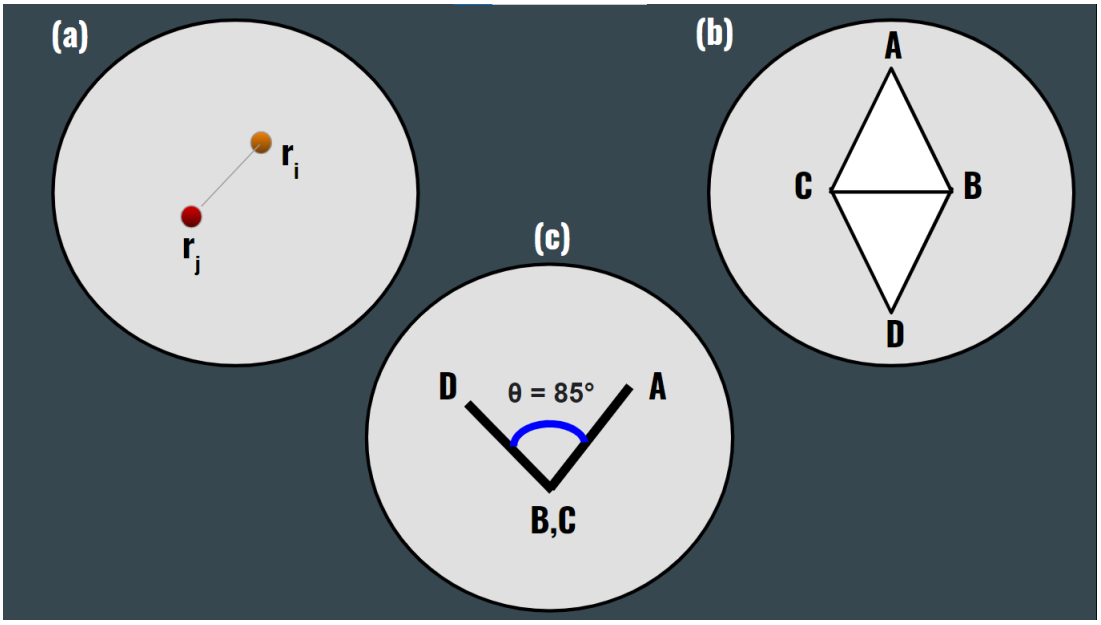


Figura 10 – Em (a) ilustração de duas partículas vizinhas na membrana. Em (b) dois triângulos (faces) da membrana adjacentes e em (c) o ângulo de calculado para a força de flexão.

Em que k_b é o coeficiente de flexão, θ_k o ângulo entre dois triângulos adjacentes que possuem os vértices AB em comum (figura 10 (c)), θ_0 o ângulo de equilíbrio e $\mathbf{n_{ABC}}$ é o vetor normal ao triângulo ABC. A força age no vértice não comum à aresta dos triângulos (partícula j - vértice A - do triângulo ABC). Metade da força age individualmente nas partículas dos vértices comuns (B e C), na direção oposta ao vetor normal.

Para conservar a área superficial do sólido são utilizadas duas forças de restauração de área. A primeira é a força de conservação da área superficial total do sólido (F_{sup}). A magnitude da força que atua individualmente em cada partícula j depende da área total do sólido e é dada por:

$$\mathbf{F}_{sup,j}(ABC) = -k_{ag}\,\frac{(S-S_0)}{S_0}\mathbf{w_A} \tag{3.5}$$

Em que k_{ag} é o coeficiente de conservação da área da superfície, S a área total da superfície, S_0 a área total de equilíbrio da superfície e $\mathbf{w_A}$ é o vetor unitário que tem direção do centroide do triângulo ABC ao vértice A. Forças análogas agem nos vértices B e C do triângulo ABC.

Além da conservação da área superficial total, é necessário existir a força de conservação da área das faces (F_{face}), pois, caso não existisse, permitiria que parte das faces estivessem, por exemplo, com a área dobrada e a outra metade das faces com a área dividida por 2. Nesse cenário, a área de equilíbrio superficial estaria conservada e a força de conservação da área total não agiria para restaurar o sólido a sua conformação original.

A força que atua nas partículas j dos triângulos ABC é dada por:

$$\mathbf{F}_{face,j}(A) = -k_{al}\,\frac{(S_{ABC} - S^0_{ABC})}{\sqrt{S_{ABC}}^0}\mathbf{w_A} \tag{3.6}$$

Em que k_al é o coeficiente de conservação da área da face, S_{ABC} a área da face, S^0_{ABC} a área de equilíbrio e $\mathbf{w_A}$ é o vetor unitário que tem direção do centroide do triângulo ABC ao vértice A. A força atua em igual intensidade e direção nas outras partículas da face.

Por último, existe a força de conservação do volume do sólido (F_{vol}). A magnitude e direção da força nas partículas j também depende da diferença entre o volume medido e o volume de equilíbrio. A força é dada por:

$$\mathbf{F}_{vol,j}(ABC) = -k_v\,\frac{(V - V_0)}{V_0}S_{ABC}\mathbf{n_{ABC}} \tag{3.7}$$

Em que k_v é o coeficiente de conservação do volume, V é o volume do sólido, V_0 é o volume de equilíbrio do sólido, S_{ABC} a área do triângulo ABC e $\mathbf{n_{ABC}}$ é o vetor normal ao triângulo ABC.

As cinco forças implementadas juntas possibilitam que a membrana seja estável frente a diversos tipos de forças externas, como compressão, tração, cisalhamento etc; e que seus comportamentos reológicos sejam mapeados e delimitados, além de oferecer diversos parâmetros que podem ser ajustadas com comportamentos reológicos experimentais.

3.2 Experimentos virtuais

Após criar uma célula estável, o primeiro passo é estudar o comportamento da célula frente a interações com estruturas externas. Do ponto de vista da engenharia, foram feitos ensaios mecânicos virtuais, ou seja, medidas reológicas virtuais para mapear a resposta da célula, considerada como um corpo de prova, frente a tensões.

O ponto de partida é o trabalho de Sandersius e Newman (2008), onde foi criado, virtualmente, um experimento em que a célula é colocada entre duas placas paralelas em que uma das placas aplica uma força na célula e a outra se mantém fixa, após são medidas as deformações da célula ao longo do tempo, medidas que permitem determinar o perfil de deformação da célula que engloba os regimes elástico e plástico.

A primeira medida realizada é a deformação (*strain*) linear celular ao longo do tempo pela aplicação de uma força de tração constante. Para determinar a deformação de um corpo de prova, neste caso a célula, aplica-se uma tensão constante e se mede as deformações totais em relação ao estado inicial do corpo. Para medir a deformação linear da célula (ε) (Desprat et al. (2005)), utiliza-se:

$$\varepsilon = \frac{\Delta l}{l_0} \tag{3.8}$$

Em que Δl é a variação na deformação linear da célula e l_0 é o comprimento original antes da deformação. Pretende-se variar a força constante aplicada para estudar os regimes de comportamento da deformação frente a diferentes forças.

A segunda medida é a obtenção da curva de tensão x deformação do corpo. A relação entre a tensão aplicada (σ) e a deformação (ε) é dada pelo módulo de Young (E):

$$E = \frac{\sigma}{\varepsilon} \tag{3.9}$$

O módulo de Young identifica a rigidez do material uma vez que mede quanta força é necessária aplicar para se ter uma determinada deformação. Para medir a deformação celular, cria-se uma máquina de ensaio virtual composta de duas placas paralelas entre o corpo de prova (figura 11). Então, aplica-se uma tensão constante ou variável, dependendo da curva que se deseja obter, em uma das placas paralelas enquanto a outra placa é mantida fixa. Nesse experimento, após a aplicação da tensão de tração, as partículas da membrana se rearranjam conforme as forças derivadas dos potenciais se distribuem no corpo. O cálculo da tensão é feito através das forças que estão sendo aplicadas nas partículas em contato com a placa superior, de forma que a força aplicada em cada partícula f_{ext} é igual a força total (F_{ext}) aplicada na placa dividida pelo número de partículas em que a força está sendo aplicada (N_f), na área de contato (A_f). Tem-se:

$$f_{ext} = \frac{F_{ext}}{N_f} \tag{3.10}$$

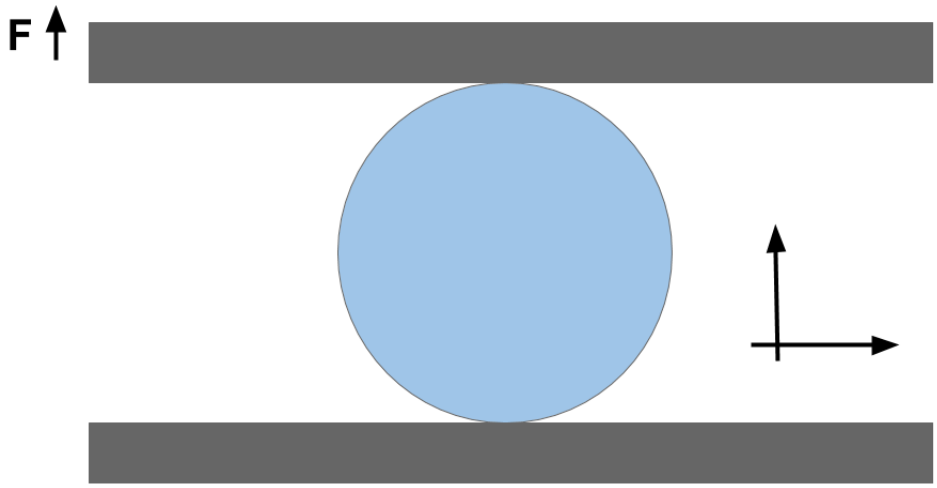


Figura 11 – Ilustração em vista lateral do experimento virtual de tração.

A tensão, por sua vez, é medida através da força total dividida pela área ou número de partículas que estão submetidas às forças:

$$\sigma = \frac{F_{ext}}{A_f} \tag{3.11}$$

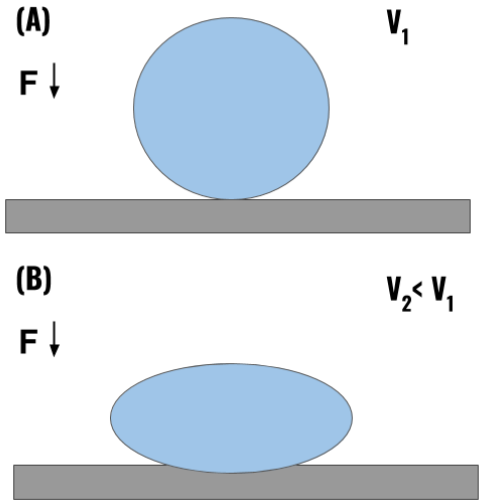
Para a célula real se movimentar, ela precisa aderir ao substrato (solo) e tracioná-lo, assim se "arrastando" de forma que grande parte da superfície celular está em contato com o substrato. Para caracterizar o quanto a célula consegue se aderir ao substrato, criou-se um novo parâmetro, o parâmetro de superfície de contato parcial, (S) em que é medida a proporção da superfície da membrana que está em contato com o substrato em função do potencial de *bending* que pode representar a rigidez da membrana celular frente a deformações. Essa medida é semelhante a uma medida de molhabilidade de superfícies.

O parâmetro de superfície de contato parcial é definido como:

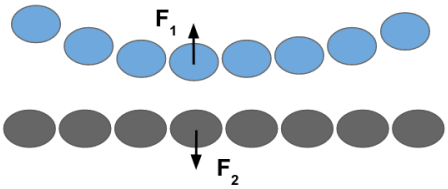
$$S = \frac{N_s}{N_f} \tag{3.12}$$

Em que N_s é o número de partículas da superfície em contato direto com o substrato e N_f é o número de partículas totais da superfície da célula. Imagina-se que quanto menos intensas as forças da membrana, maior a área de superfície que estará em contato direto com o substrato (figura 12a). Nesse experimento o substrato é composto de partículas imóveis não ativas e a interação entre as partículas da membrana e do substrato se dará por um potencial de exclusão de volume que impede que a membrana atravesse o substrato (figura 12b).

As medidas realizadas foram feitas variando-se o potencial de flexão a fim de obter o espaço de parâmetros que define os diferentes regimes de resposta reológica celular. Ao final



(a) Célula para dois potenciais de *bending*.



(b) Forças derivadas do potencial de exclusão entre as partículas do substrato (em cinza) e as partículas da membrana (em azul).

Figura 12 – Em (A) se observa a célula mais próxima do formato esférico e em (B) com formato não esférico e superfície de contato maior para um potencial de *bending* $V_2 < V_1$. Em (b) os detalhes de interação das partículas da membrana com as partículas do substrato.

da caracterização também foi comparado o comportamento reológico de tensão-deformação com comportamentos experimentais. Por fim, espera-se que as medidas sejam dependentes da temperatura efetiva do corpo de prova (intensidade do ruído, i.e. atividade da membrana) e, do ponto de vista virtual, pode-se variar o ruído a que as partículas estão submetidas para estudar esses comportamentos. O ponto que caracteriza o fim das medidas reológicas propostas é a caracterização da célula em relação a tensão-deformação em função dos potenciais de *bending*, o que gerará o espaço de parâmetros da célula.

Parte IV

Desenvolvimento da célula 3D

4 Desenvolvimento da estrutura geométrica

Para a criação da estrutura da célula 3D, implementou-se o algoritmo abordado no capítulo 3 que se baseia na criação de uma esfera a partir de um icosaedro de forma que as arestas do sólido são considerados partículas.

4.0.1 Criação do icosaedro

O icosaedro é basicamente a interconexão de duas pirâmides pentagonais de mesma dimensão deslocadas por um ângulo. A base da pirâmide é um pentágono circunscrito em um círculo de raio R. Partindo do círculo (figura 13), selecionamos 5 pontos espaçados por ângulos iguais de 72° (figura 14) para o primeiro pentágono. Os números sem unidades a seguir se referem a unidades arbitrárias (u.a.). Para formar a pirâmide, cria-se o vértice inferior com

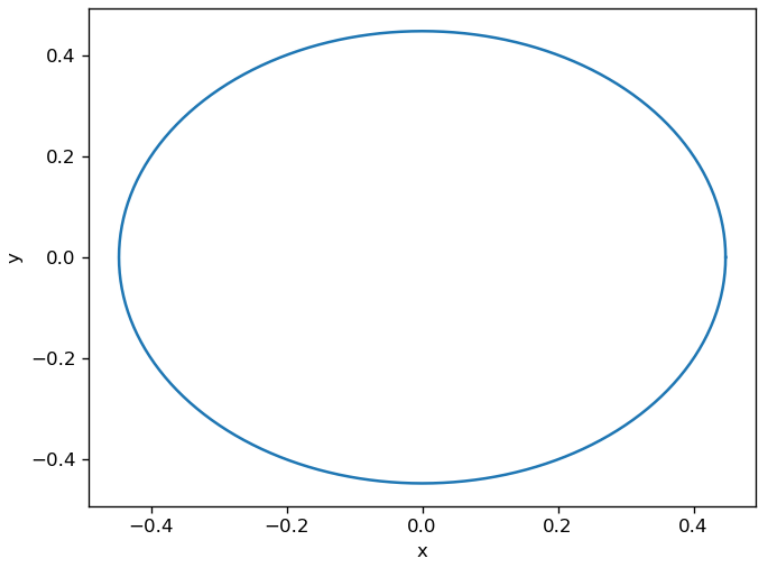


Figura 13 – Círculo que forma a base do icosaedro.

uma distância de $\frac{1}{2}(1-R)$ da base do pentágono (figura 15). Para criar a segunda pirâmide, cria-se outro pentágono deslocado em 36° do primeiro pentágono (figura 16). Cria-se o ponto superior da última pirâmide (figura 17). Separam-se as bases das pirâmides por uma distância de $(1-R)\cos(36°)$ (figura 18). Por fim, conectam-se os pontos das duas pirâmides para formar as faces do icosaedro (figura 19). Para criar a esfera composta por partículas, foi implementado o algoritmo (capítulo 3) que segmenta as arestas do icosaedro para a formação de novos vértices.

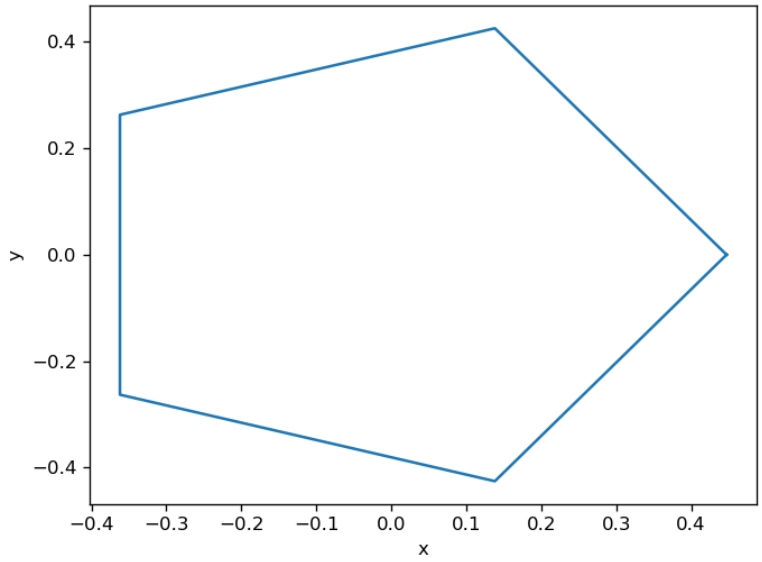


Figura 14 – Base da pirâmide inferior.

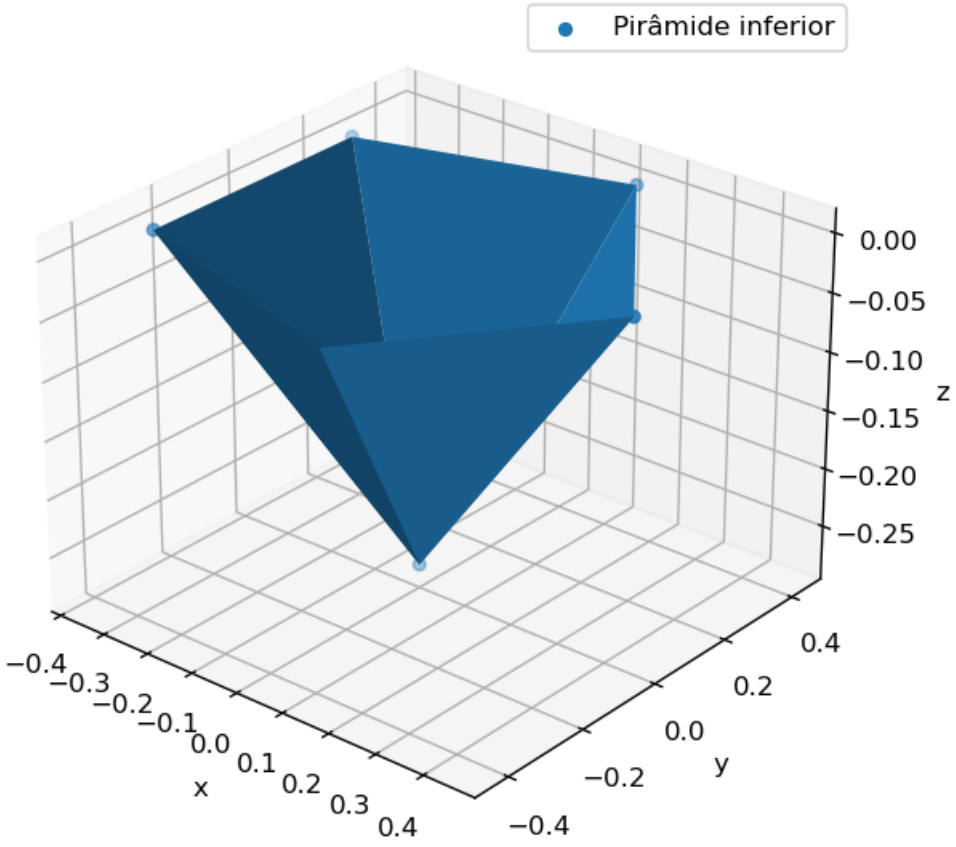


Figura 15 – Pirâmide inferior.

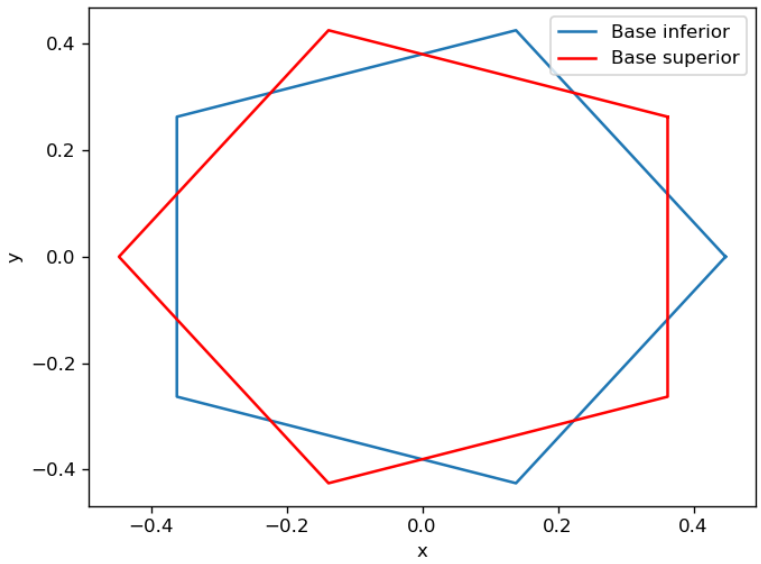


Figura 16 – Bases das pirâmides inferior e superior.

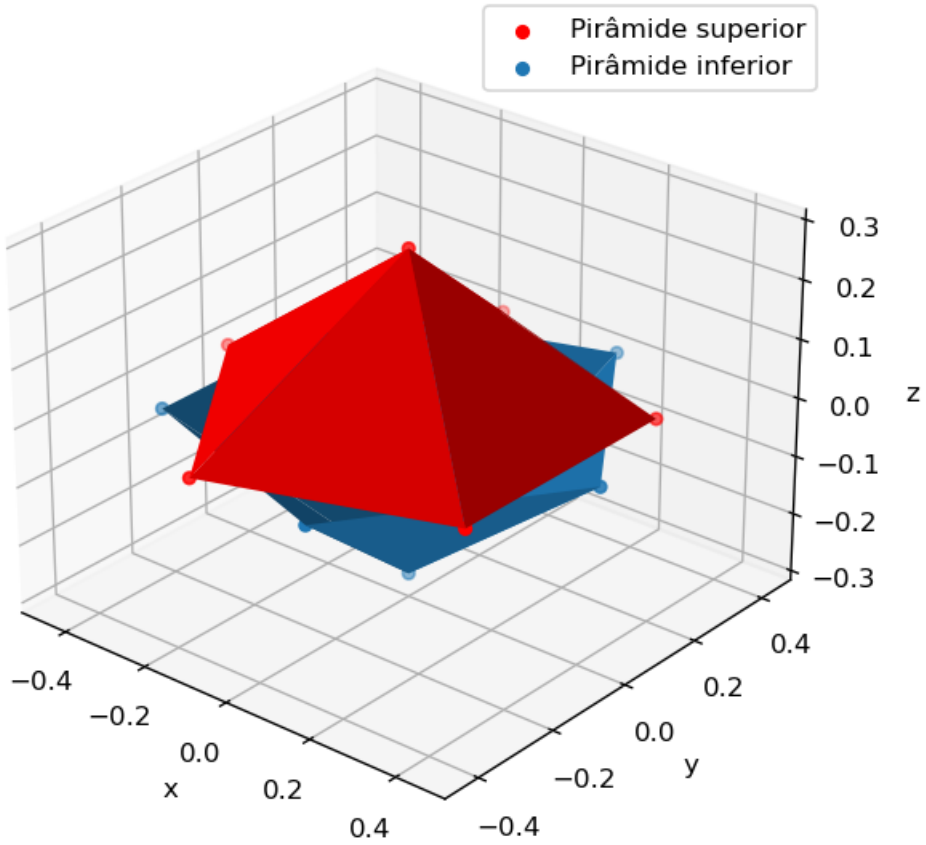


Figura 17 – Pirâmides superior e inferior.

4.0.2 Segmentação do icosaedro e aproximação esférica

Para segmentar o icosaedro, escolhe-se uma face do sólido (figura 20) e um novo vértice é inserido no centro de cada aresta de forma que serão criadas novas faces triangulares (figura

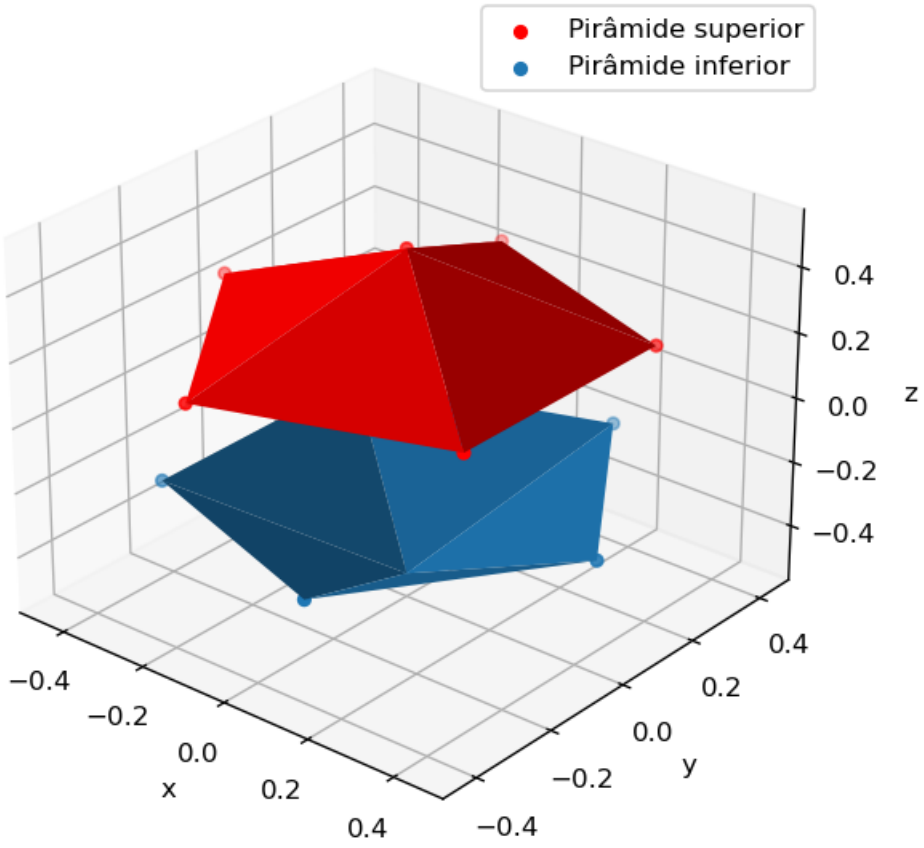


Figura 18 – Pirâmides superior e inferior.

21). O algoritmo é aplicada em todas as faces do icosaedro gerando o icosaedro de primeira segmentação (figura 22). Por fim, para aproximar o icosaedro segmentado para a esfera de primeira segmentação, projeta-se a distância de cada vértice em uma esfera de raio R (figura 23). Aplicando mais uma vez o algoritmo, obtemos a esfera de segunda segmentação (figura 24) Aplicando de forma recorrente o algoritmo no icosaedro, obtemos as esferas com maior grau de segmentação (figura 25).

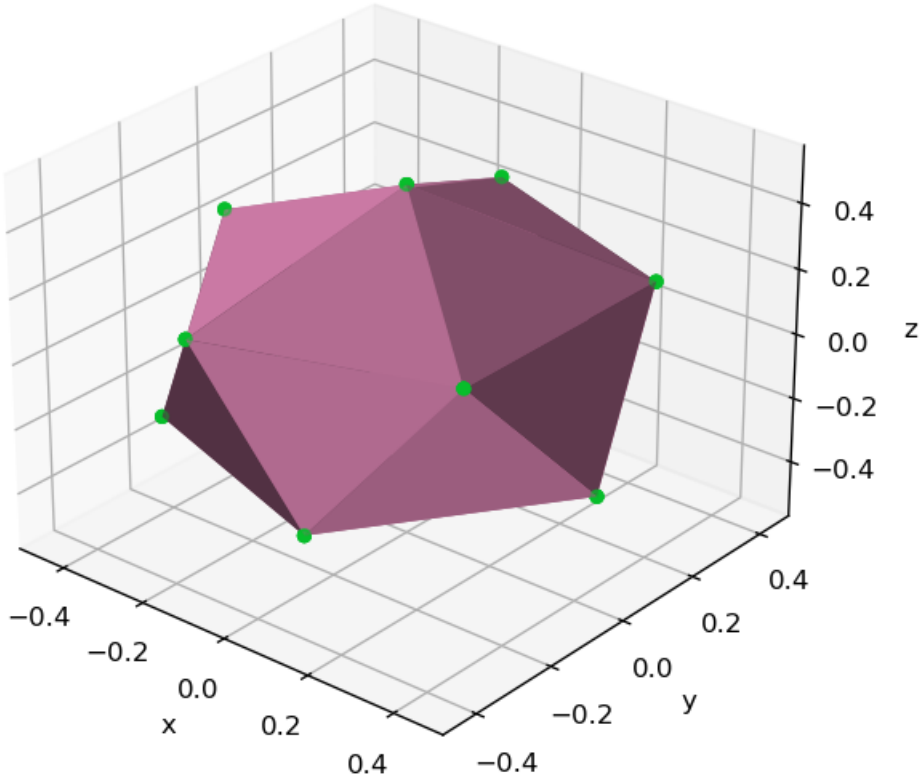


Figura 19 – Icosaedro.

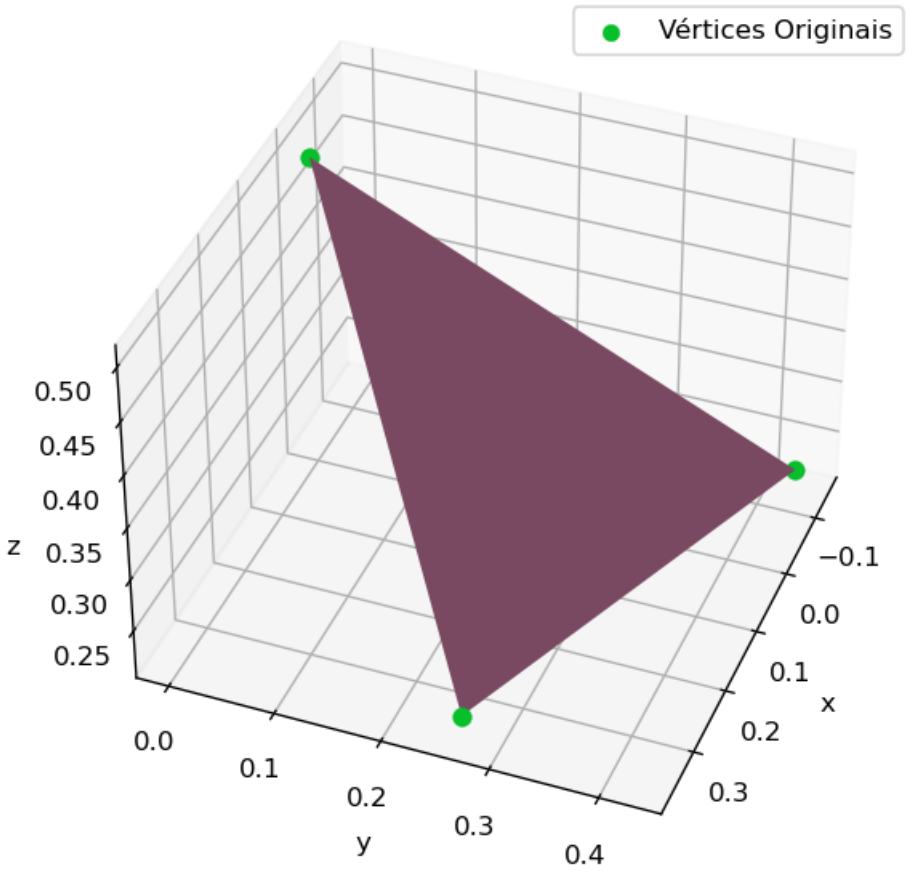


Figura 20 – Face triangular do icosaedro.

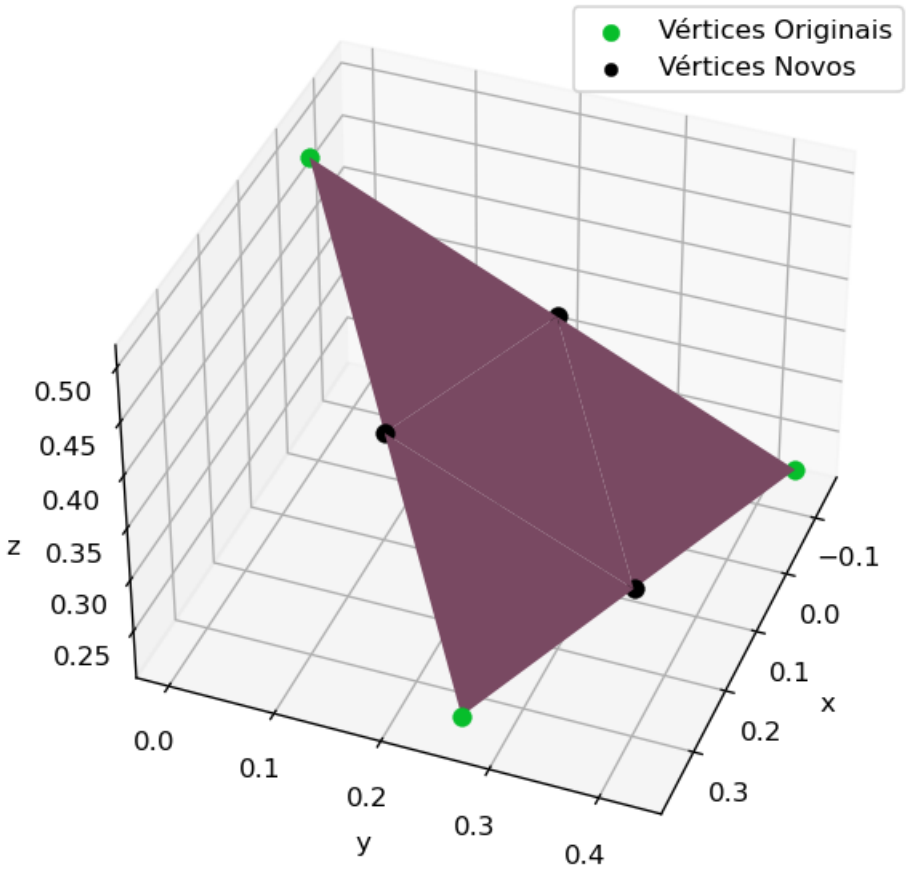


Figura 21 – Face triangular segmentada do icosaedro.

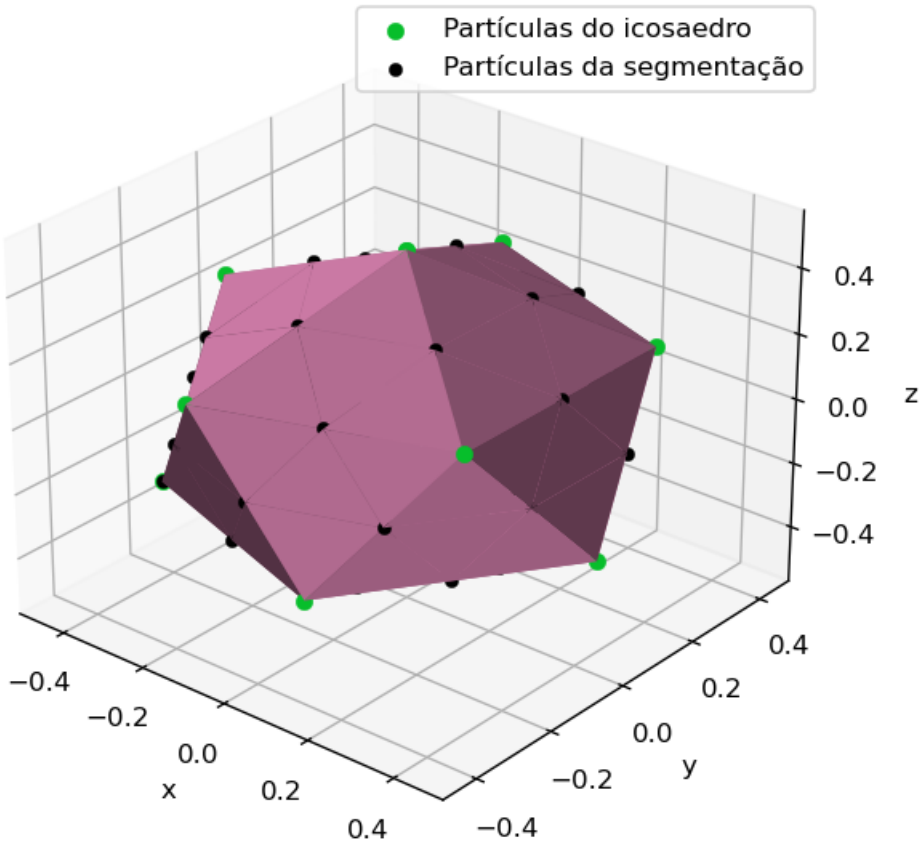


Figura 22 – Icosaedro de primeira segmentação.

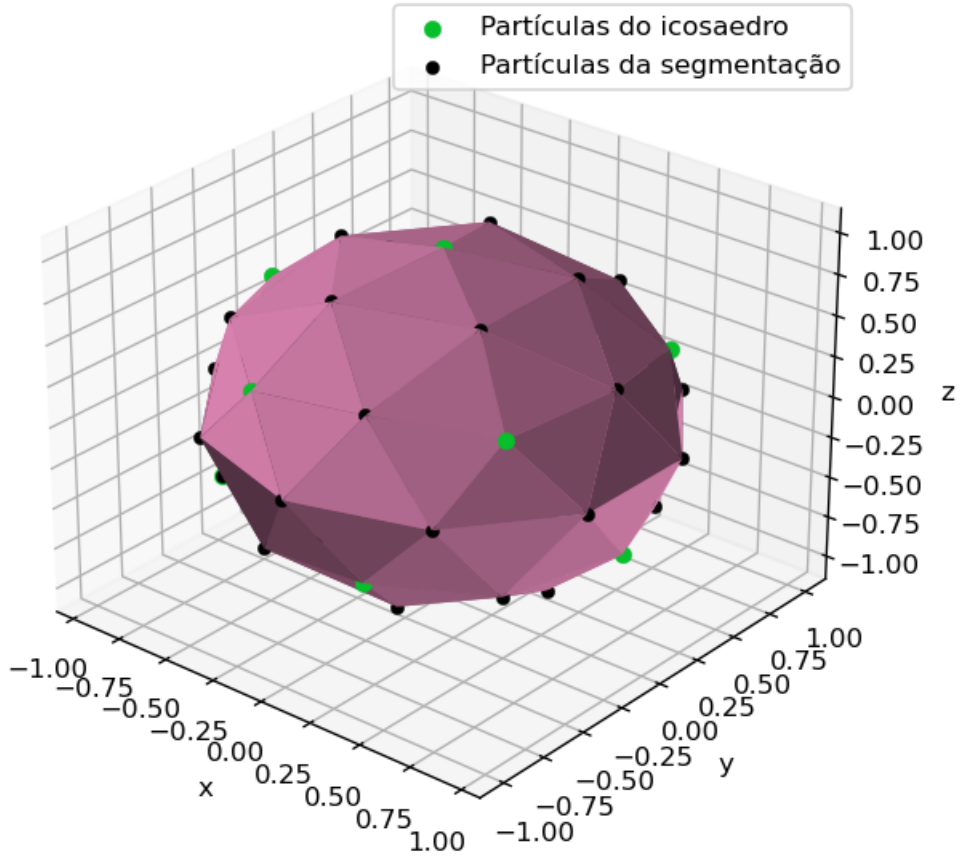


Figura 23 – Esfera de primeira segmentação.

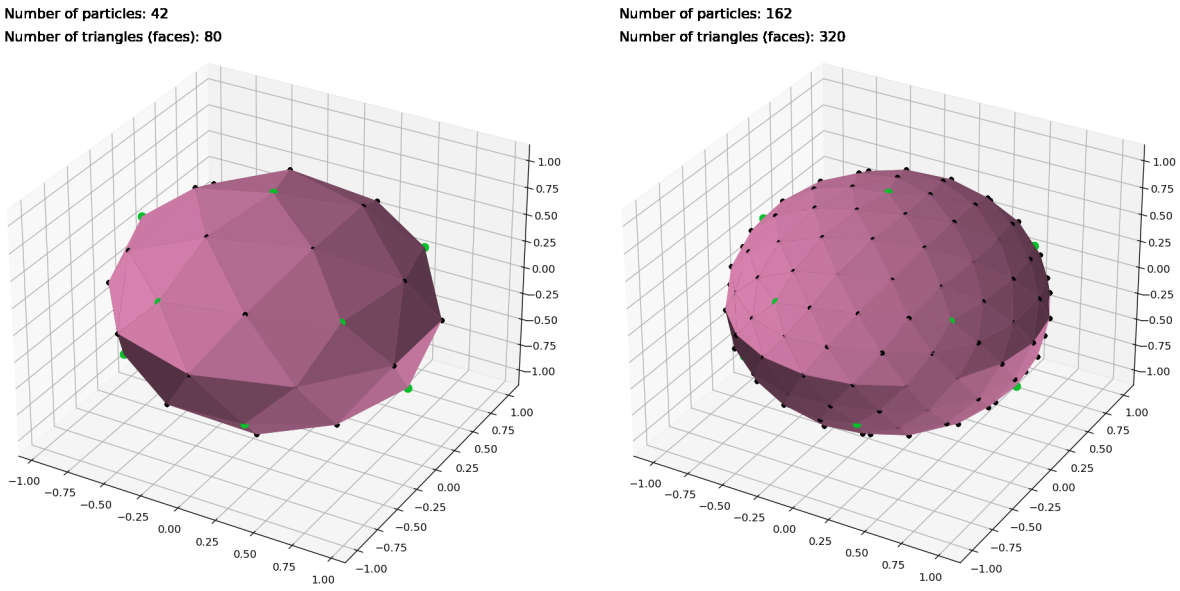


Figura 24 – Esferas de primeira e segunda segmentação.

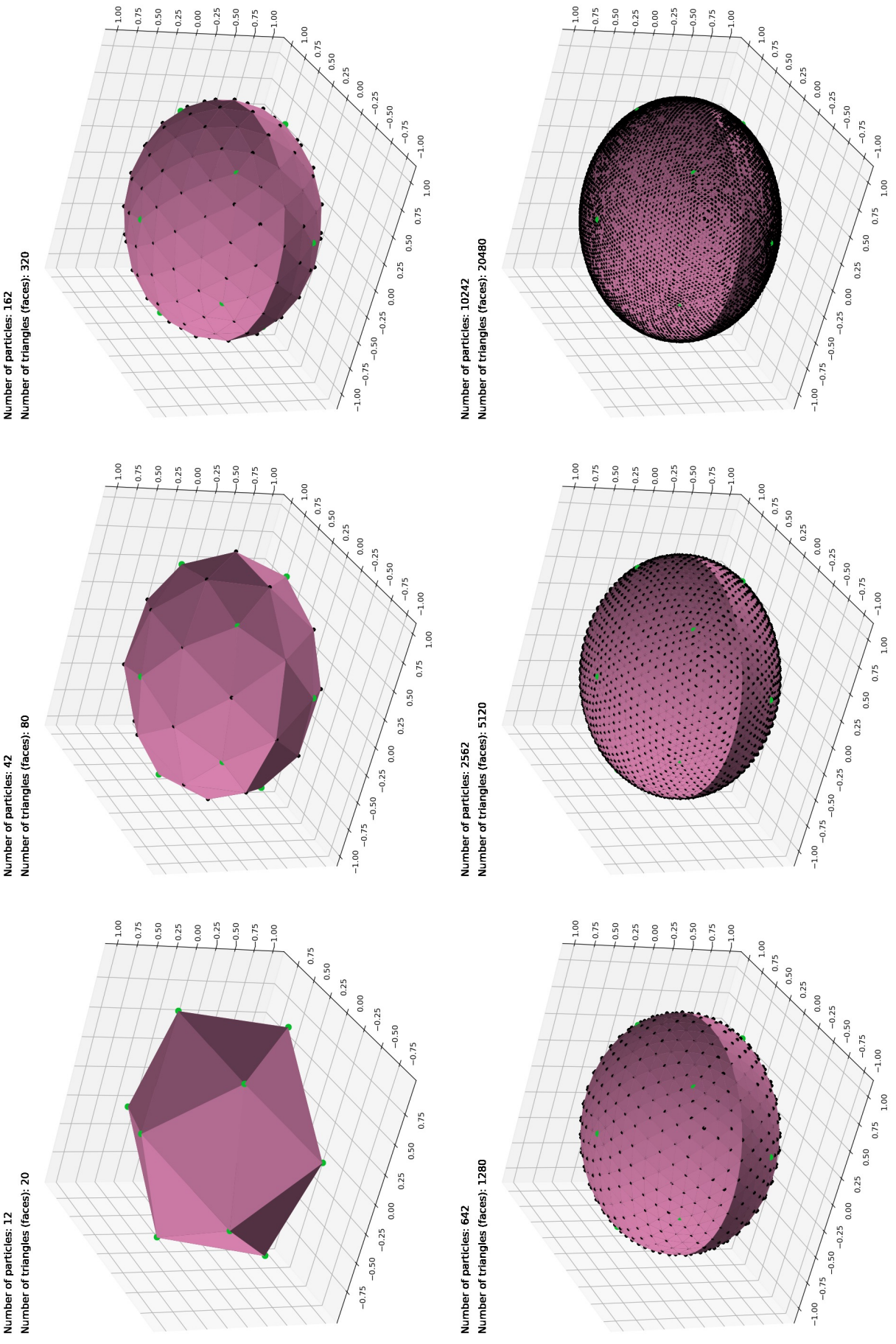


Figura 25 – Primeira até a quinta esfera segmentada.

5 Implementação do modelo físico

Uma vez que a estrutura geométrica da célula 3D foi construída, iniciamos a implementação do modelo físico apresentado no capítulo 3. O modelo envolve 5 forças que têm o papel de conservar, na ausência de forças externas, a estrutura física esférica da célula, permitir que interaja com corpos externos e tenha comportamentos físicos coerentes. A primeira força fundamental é a força de ligação que é basicamente uma força de mola que interconecta as partículas que dão o aspecto básico de um corpo sólido.

5.0.1 Força de ligação (*bonding*)

A força de ligação faz com que as partículas mantenham a mesma distância que tinham na esfera inicial. A força é dada pela equação 3.3. Para calcular a força de ligação entre as partículas, é necessário mapear todas as primeiras partículas vizinhas de cada partícula da esfera de forma a calcular todos os pares e eliminar cálculos duplicados. Foram construídas rotinas que identificam os pares de partículas vizinhas sem repetição e salvam em arquivo. Neste GIF é possível ver a identificação visual dos pares de partículas para o icosaedro e um *frame* do gif na figura 26. Foi, então, implementada a força de ligação entre as partículas com ruído para a esfera de primeira segmentação. As demais forças foram zeradas e a distância média entre as partículas foram medidas no tempo para diferentes valores do coeficiente de *bonding* (k_s) - figura 27. Com o ruído aplicado, as partículas tendem a se dispersar e a não manter suas distâncias médias, por isso há um claro aumento na média para $k_s = 0$. Quanto maior o valor de k_s, menor a dispersão da média no tempo, de forma que a curva mais estável é para $k_s = 100$. Este resultado indica que a força implementada está funcionando como esperado.

5.0.2 Força de flexão (*bending*)

A força de flexão age para que os ângulos entre os pares de faces triangulares do sólido se mantenham no seu valor de equilíbrio, isso impede que a célula se sobreponha e dificulta movimentos de invaginação, conforme esperado para uma estrutura estável. A função que define a força de flexão do modelo de Tóthová, Jančigová e Bušík (2015) é dada pela equação 3.4 em que o ângulo medido é o ângulo entre as faces triangulares. A força age no sentido do ângulo normal às faces.

Para medir o ângulo normal a uma face triangular, utiliza-se o produto vetorial entre dois vetores no plano da face, de forma que, dependendo da ordem do produto vetorial, o vetor aponta

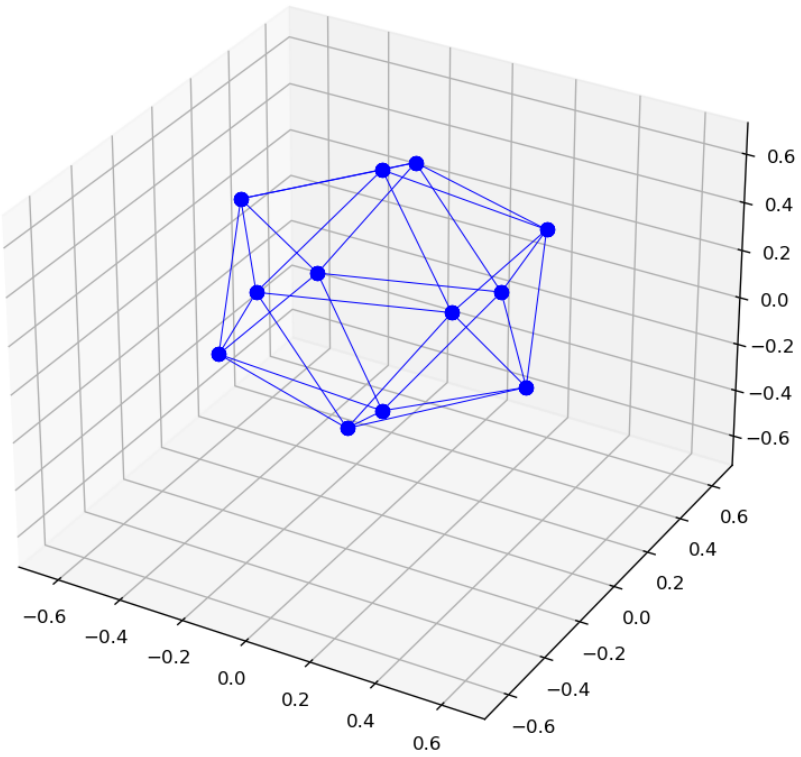


Figura 26 – Imagem dos pares de partículas do icosaedro identificados pelo algoritmo desenvolvido.

em sentidos diferentes. Para garantir que os cálculos estarão corretos, é necessário verificar se todos os vetores normais calculados apontam para a mesma direção radial da esfera, ou seja, o cálculo deve independer do quadrante em que os vértices do triângulo estão. Então, para validar os cálculos, foi construída uma face triangular e rotacionada ao longo dos 3 eixos do espaço e, ao mesmo tempo, foi calculado o vetor normal. Neste vídeo se encontra a rotação em torno do eixo X, neste vídeo a rotação em torno do eixo Y e neste vídeo a rotação em torno do eixo Z. Como se pode constatar, o ângulo calculado aponta sempre na mesma direção independente do quadrante, o que indica que a forma de cálculo pode ser utilizada. Como a força de flexão atua em pares de triângulos, foram construídos dois pares de triângulos e, para testar todos os cenários possíveis de ângulos da força de *bending*, fez-se simulações para diferentes ângulos iniciais. Na figura 28 podemos ver o *frame* de uma simulação entre um par de triângulos com ângulo inicial de 150 graus.

Começamos a simulação com um ângulo inicial de 180° (com rúido) e ângulo de

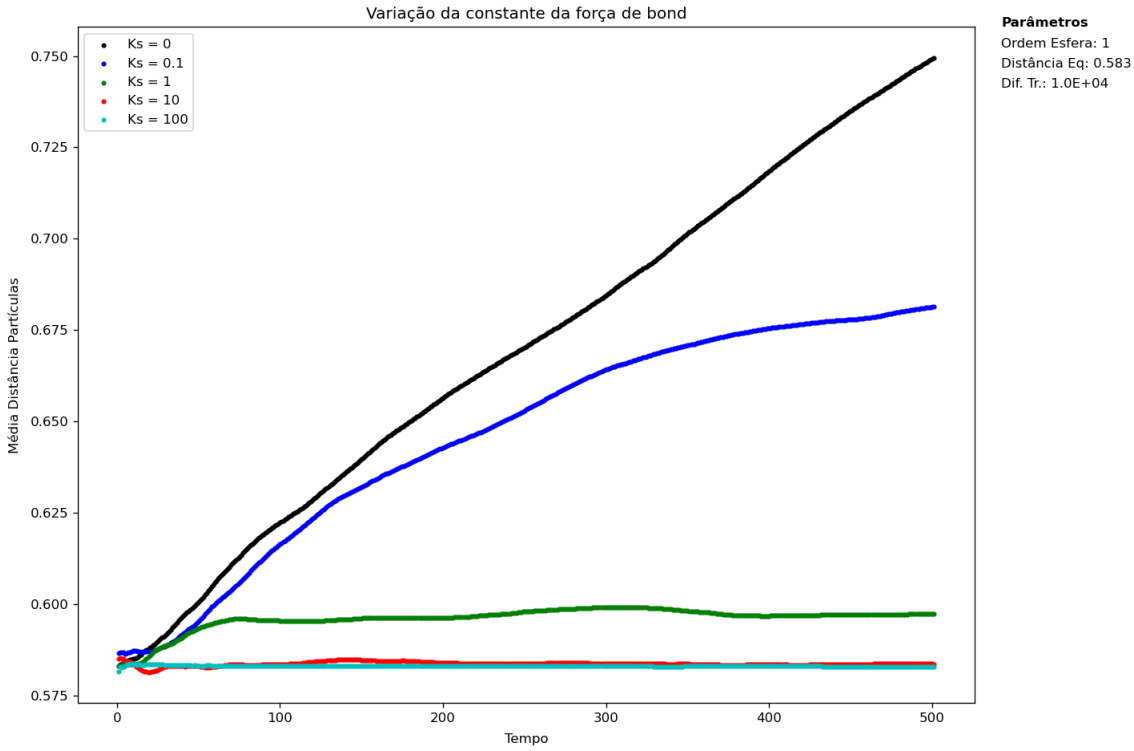


Figura 27 – Distância média entre as partículas da primeira esfera ao longo do tempo para diferentes coeficientes de força de *bonding*.

equilíbrio de 90° (vídeo) e notamos que a força age e retorna o par para o ângulo de equilíbrio. Mantendo constante o ângulo de equilíbrio para as outras simulações, o mesmo acontece para o ângulo inicial de 150° (vídeo), de 90° (com ruído) (vídeo), de 60° (vídeo), de 45° (vídeo), de 15° (vídeo) e de 0° (vídeo). Entretanto, quando entramos no quarto quadrante, em que o ângulo entre os vetores normais é igual ou maior do de 270°, a força age nas faces de forma que as faces se sobrepõem (vídeo da simulação). Isso ocorre porque o método de cálculo do ângulo - utilizando-se o produto escalar - não consegue diferenciar o ângulo 270° do ângulo de 90° no espaço tridimensional. Se aplicarmos essa equação à dinâmica da esfera, em uma situação em que a esfera se invagina, as faces podem se sobrepor, então a esfera deixará de se comportar como um corpo sólido. Essa simulação prova que precisamos modificar a forma de cálculo do ângulo de flexão.

Para corrigir esse problema, definimos um eixo de rotação com sentido fixo para cada par de triângulos (figura 29). Para descobrir em qual quadrante o ângulo medido se encontra, calcula-se também o vetor normal (vetor direção) aos vetores normais às faces dos triângulos. Esse vetor apontará em um dos sentidos do eixo de rotação. Então, para saber em qual quadrante o ângulo comparado se encontra, basta comparar o sentido do vetor direção em relação ao vetor que define o eixo de rotação. Exemplo: se o ângulo calculado é igual a 90° e os vetores eixo de rotação e vetor direção estão paralelos, o ângulo medido é de 90° em relação ao eixo X da origem (figura 29). Se o ângulo calculado é igual a 90° e os vetores eixo de rotação e vetor direção estão antiparalelos, o ângulo medido, na verdade, é de 270° em relação ao eixo X da

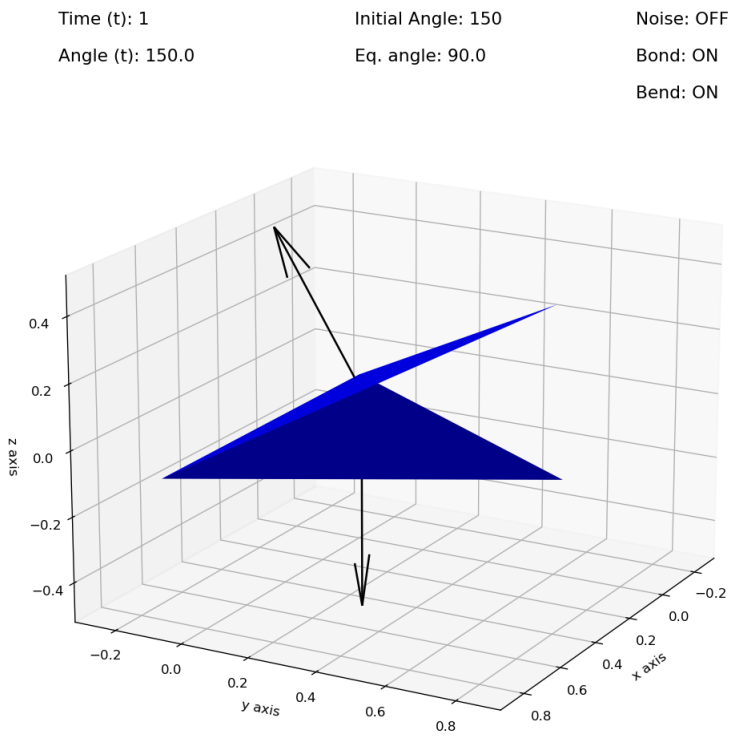


Figura 28 – Par de triângulos interagindo com ângulo inicial de 150 graus.

origem (figura 29).

Em resumo, se os vetores estão paralelos, o ângulo calculado (0 a 180 graus) estará, por definição, no 1° ou 2° quadrante (ângulo medido de 0 a 180 graus). Caso os vetores estejam antiparalelos, o ângulo calculado (0 a 180 graus) está no 3° ou 4° quadrante (ângulo medido de 180 a 360 graus). Nesse caso, então, é necessário somar 180° para descobrir o ângulo medido. Por fim, utilizamos o ângulo entre as faces dos triângulos na equação de força de flexão ao invés do ângulo entre os vetores normais às faces (como era utilizado no modelo original). Para isso, basta somar 180° ao ângulo entre os vetores normais calculados. Aplicando a mudança no algoritmo, temos o resultado esperado para a rotação dos ângulos iniciais ou acima de 270° (vídeo da simulação). Os resultados para ângulos menores também seguem o mesmo comportamento. Então, foram feitas simulações para a primeira esfera segmentada. Foi plotada a média dos ângulos entre os pares de triângulos no tempo para diferentes coeficientes de *bending* (K_b), enquanto as outras forças foram zeradas, e apenas o ruído ativo. Na figura 31 se percebe uma grande dispersão no ângulo médio para valores do coeficiente baixos (*0* e *0,1*) enquanto que, quanto maior o valor do coeficiente, mais o sólido mantém os seus ângulos próximos ao ângulo de equilíbrio.

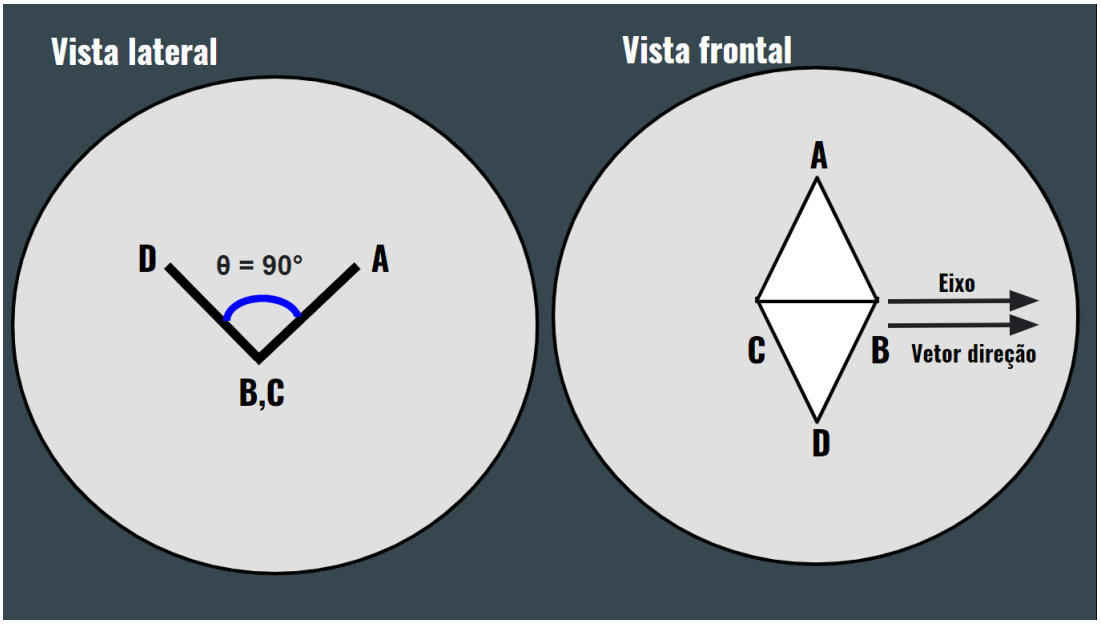


Figura 29 – Vistas lateral e frontal de um par de faces triangulares com ângulo de 90° (no 1°e 2° quadrantes) com a ilustração dos vetores eixo de rotação e vetor direção.

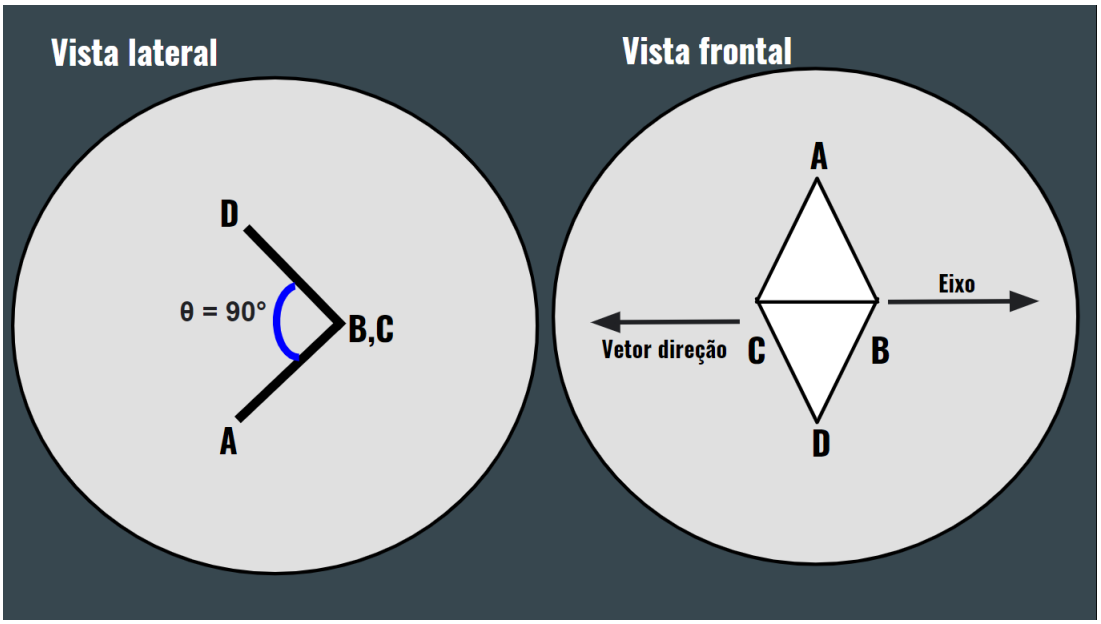


Figura 30 – Vistas lateral e frontal de um par de faces triangulares com ângulo de 90° (no 3°e 4° quadrantes) com a ilustração dos vetores eixo de rotação e vetor direção.

5.0.3 Força de volume

A força de volume tem a função de manter o volume da célula próximo do volume de equilíbrio de forma que aplica forças radiais nas partículas para que o volume aumente ou diminua. Foi medido o volume da célula ao longo do tempo para diferentes coeficientes de força de volume com ruído e com as outras forças zeradas (figura 32). Percebe-se que a oscilação em torno do valor de equilíbrio diminui conforme o coeficiente de volume aumenta, o que mostra que a força de volume realiza a função de manter a estabilidade volumétrica celular.

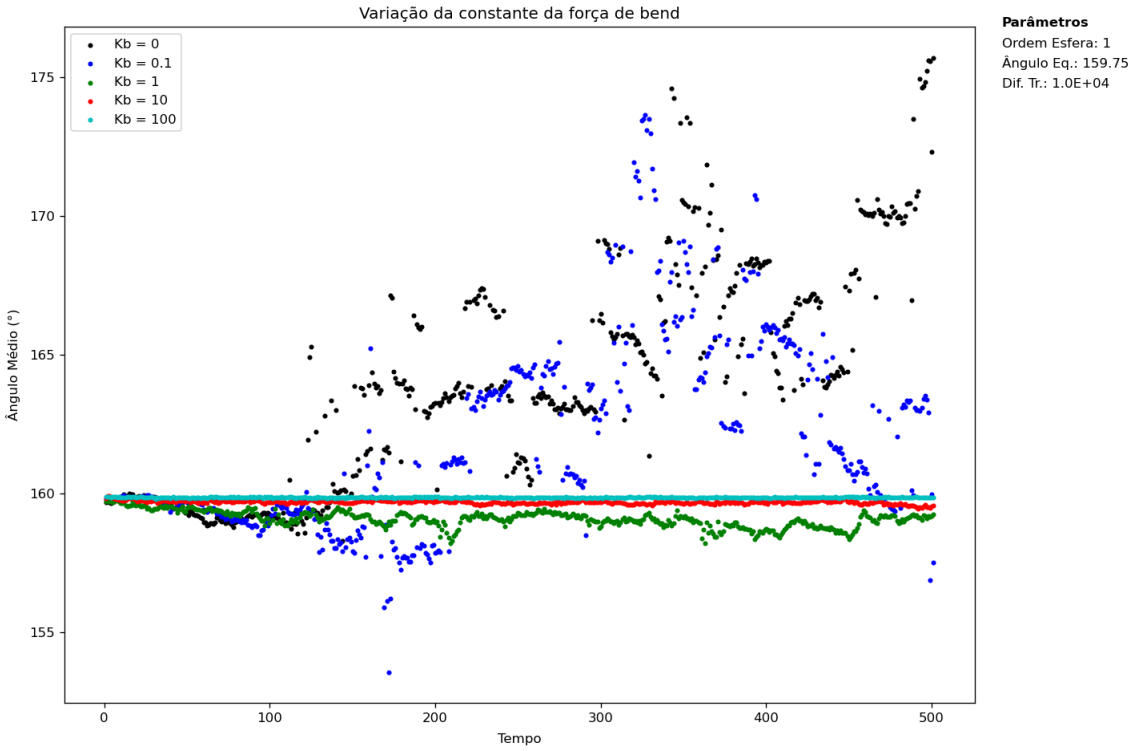


Figura 31 – Ângulo médio entre os pares de triângulos da primeira esfera ao longo do tempo para diferentes coeficientes de força de *bending*.

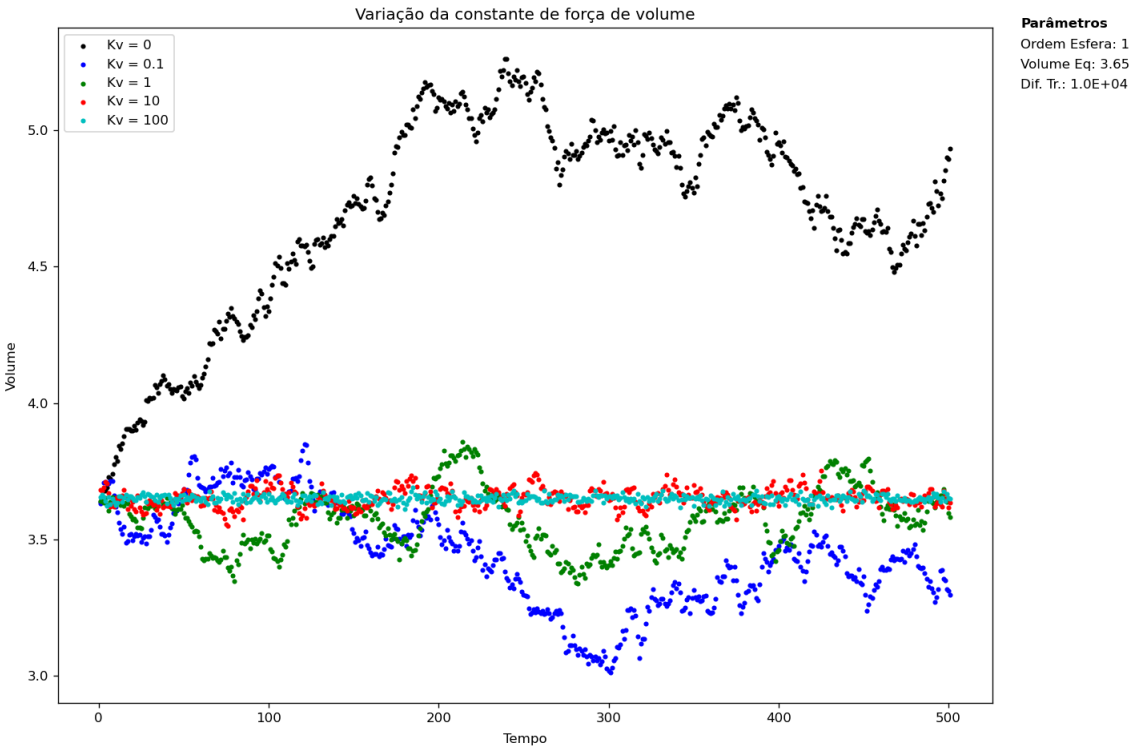


Figura 32 – Volume da primeira esfera ao longo do tempo para diferentes coeficientes de força de volume.

5.0.4 Força de área global

A força de área global atua mantendo a área superficial da célula próxima à área de equilíbrio e atua diminuindo a área das faces dos triângulos em função da área total medida. Foi medida a área total no tempo com ruído e outras forças zeradas para diferentes coeficientes de área global (figura 33). Mais uma vez, como esperado, a área superficial do sólido se mantém próxima à

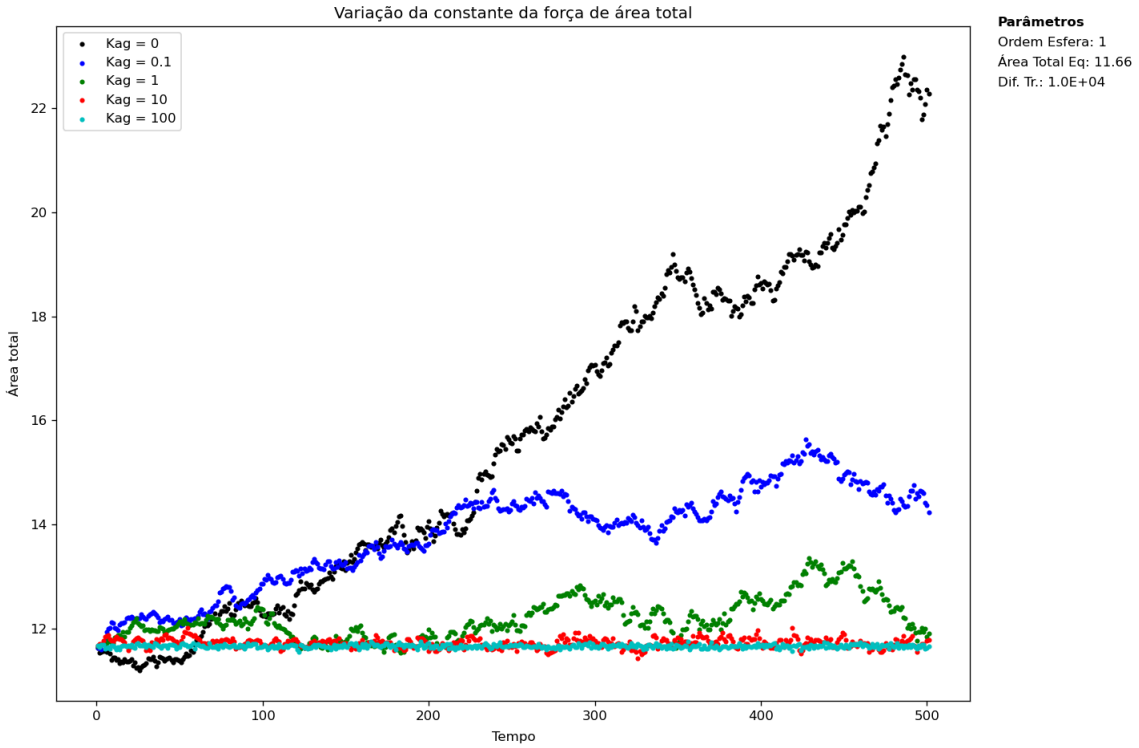


Figura 33 – Área superficial da primeira esfera ao longo do tempo para diferentes coeficientes de força de área global.

área de equilíbrio conforme maior o coeficiente da força.

5.0.5 Força de área local

Diferente da força de área global, a área local atua diminuindo ou aumentando a área das faces triângulos com base na diferença entre a área da face medida e a área de equilíbrio da face, ou seja, a área superficial não necessariamente precisa estar maior do que a área de equilíbrio para que a força tende a diminuir a área local de uma face. É uma força que atua localmente. A força de área local atua independentemente em cada face triangular. Para a implementação no sólido, foram feitas simulações em uma face triangular com forças externa aplicadas no sentido de expandir a face (vídeo da simulação).

A mesma simulação foi feita com a força de área local ativada e coeficiente de área local ($K_{al} = 10$) - vídeo da simulação. Percebe-se que a velocidade de expansão da área é muito mais baixa. Para um coeficiente 10 vezes maior ($K_{al} = 100$), a força consegue expandir apenas 10% da áre inicial, até que a força da área local seja de mesma magnitude da força externa

aplicada (vídeo da simulação). Com isso, confirmamos que a força de área local consegue manter a estabilidade das áreas das faces triangulares.

O próximo passo foi implementar a força para todas as faces do sólido e, então, foram medidas as médias das áreas das faces triangulares da esfera de primeira segmentação no tempo com ruído e com as outras forças zeradas (figura 34). Por fim, como esperado, o módulo do

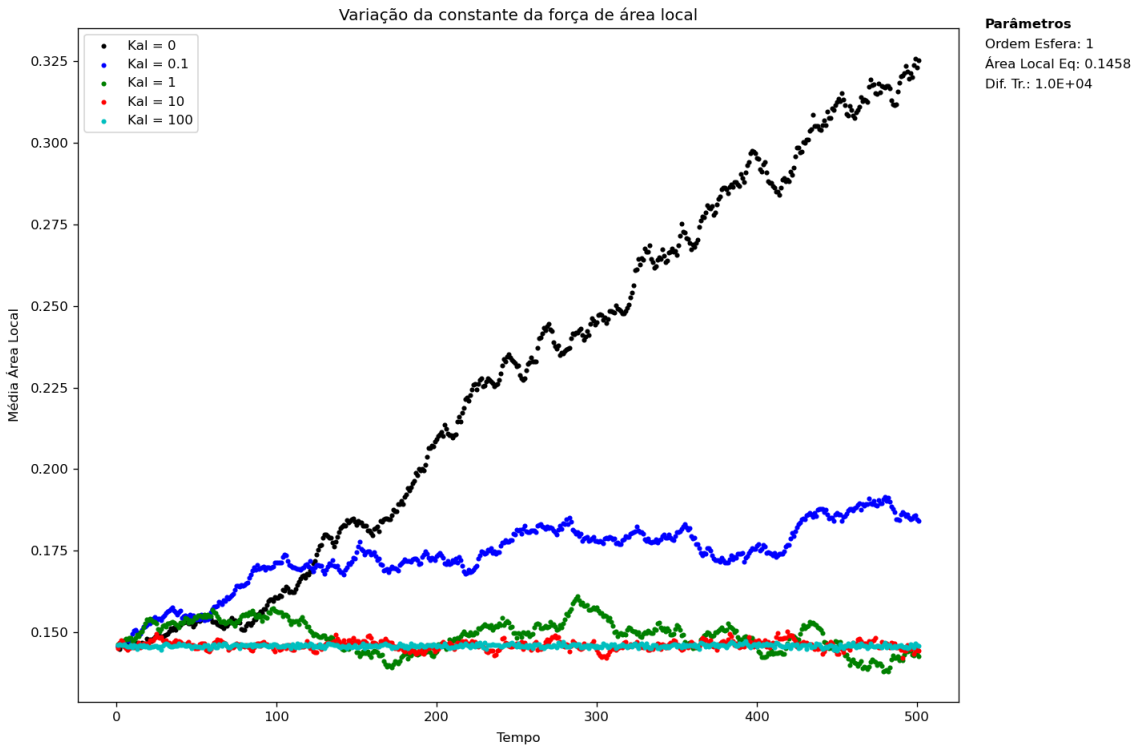


Figura 34 – Área média das faces triangulares da primeira esfera ao longo do tempo para diferentes coeficientes de força de área local.

coeficiente é proporcional ao quanto a média das áreas se aproxima do equilíbrio ao longo do tempo. Em uma primeira análise, conclui-se que o maior coeficiente ($K_{al} = 100$) mantém a esfera com as áreas locais mais estáveis e, por isso, é o melhor valor de coeficiente em termos de estabilidade geométrica. Entretanto, ao realizar a simulação da esfera sem ruído e com uma força na direção radial nas partículas da esfera para simular situações de pressão, para este coeficiente, percebe-se que a esfera se torna um icosaedro perfeito ao longo do tempo (vídeo da simulação). Isto acontece porque a forma geométrica desse sólido com as áreas das faces iguais é um icosaedro. Para corrigir esse problema, manteve-se constante a área local de cada triângulo individualmente no estado inicial, diferente de definir uma mesma área de equilíbrio para todos.

Com base nessa simulação, notou-se que possivelmente seria necessário modificar os outros parâmetros de equilíbrio, os quais: distância entre as partículas e ângulos de equilíbrio entre os triângulos. Para validar essa hipótese, plotou-se a distribuição de distância de equilíbrio das partículas e dos ângulos entre as faces para as esferas com diferentes segmentações (figuras 35, 36, 37 e 38).

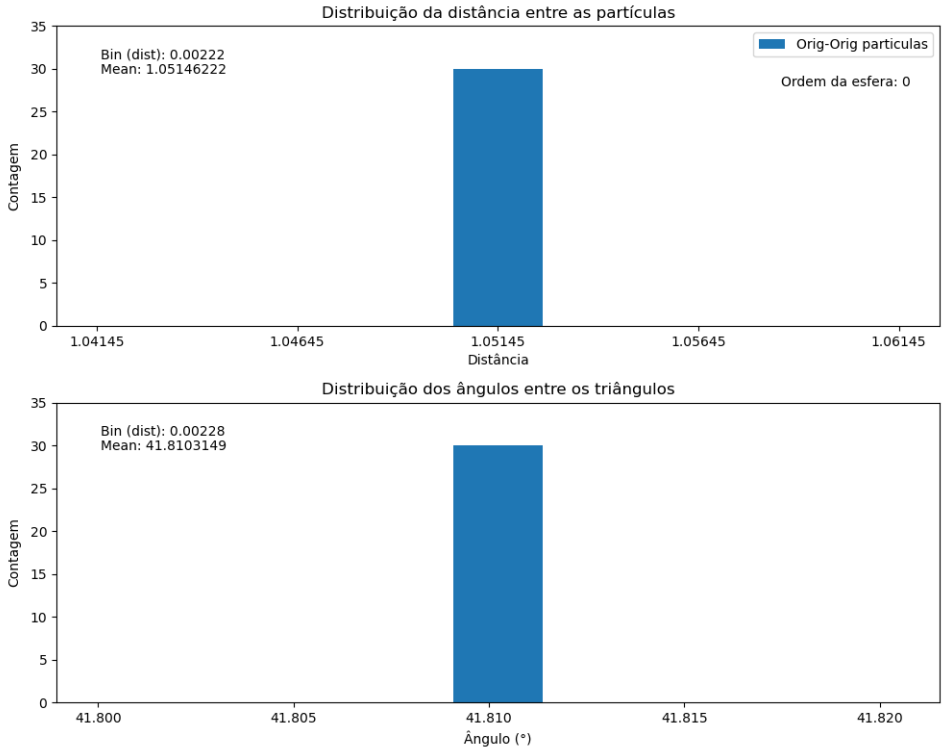


Figura 35 – Distribuição da distância entre as partículas e dos ângulos entre as fases dos triângulos para a esfera de ordem 0 (icosaedro).

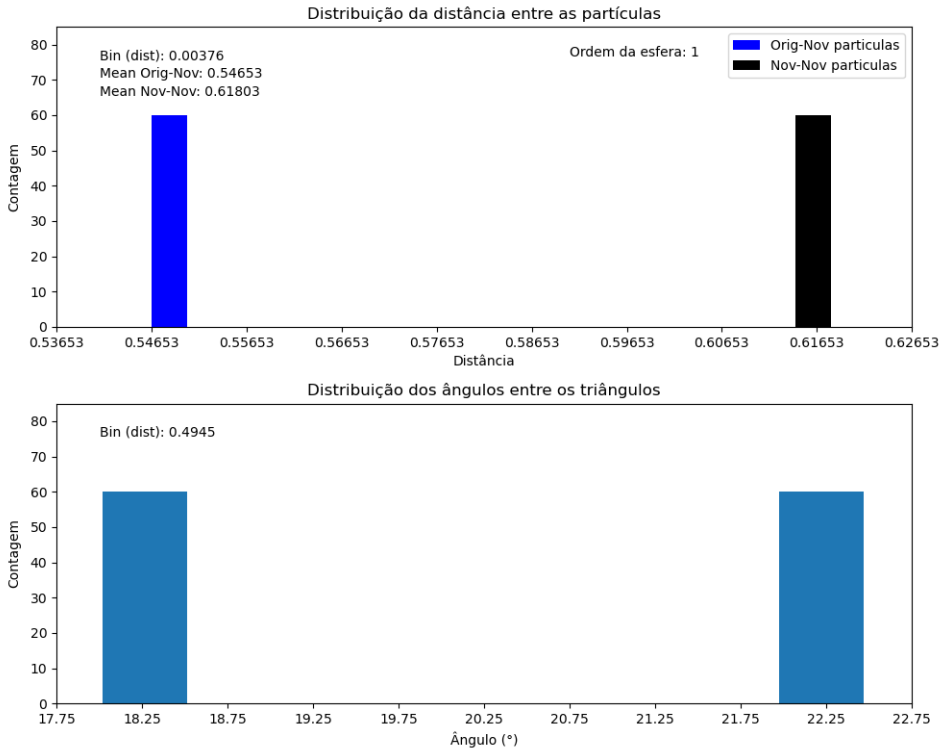


Figura 36 – Distribuição da distância entre as partículas e dos ângulos entre as fases dos triângulos para a esfera de ordem 1.

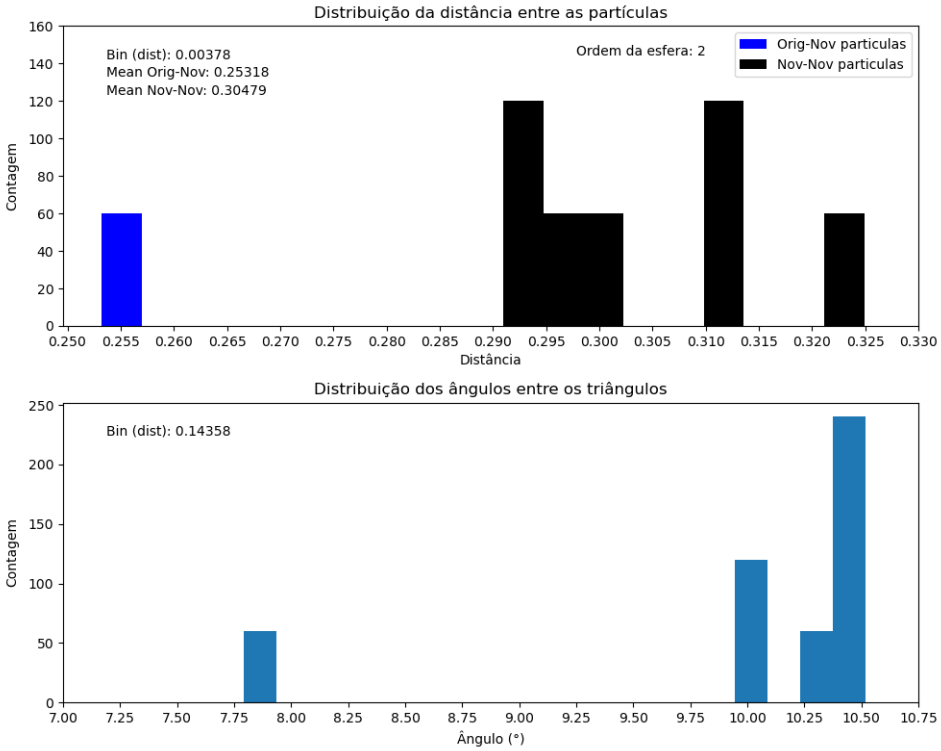


Figura 37 – Distribuição da distância entre as partículas e dos ângulos entre as fases dos triângulos para a esfera de ordem 2.

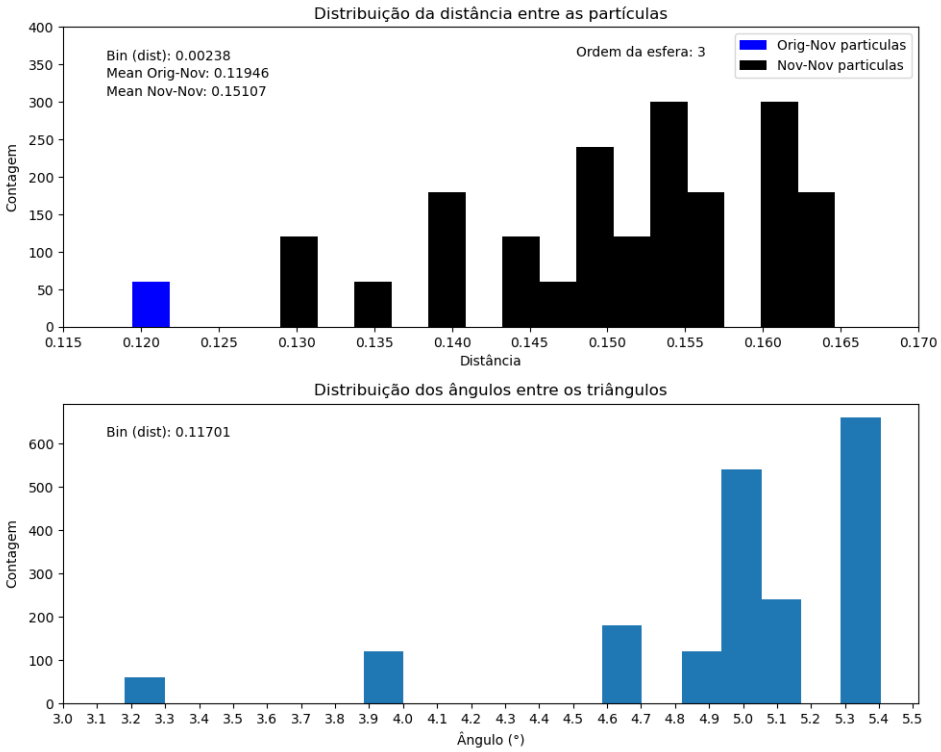


Figura 38 – Distribuição da distância entre as partículas e dos ângulos entre as fases dos triângulos para a esfera de ordem 3.

Notou-se que para o icosaedro (figura 35) temos apenas uma distância de equilíbrio inicial e um ângulo entre as faces. Conforme a ordem da esfera aumenta (figuras 36, 37 e 38), se inicia uma distribuição de distâncias e ângulos cada vez maior, o que indica que para aplicar corretamente as forças do modelo para as esferas de maior ordem (a partir da 2a esfera) e manter a estrutura da célula estável, é necessário preservar as distribuições iniciais ao invés de apenas uma distância/ângulo/área de equilíbrio igual para todas as partículas/ângulos/faces.

Parte V

Resultados

6 Comportamento mecânico da célula

Como primeiro passo torna-se necessário entender os comportamentos mecânicos da célula frente a forças externas aplicadas, assim como seus parâmetros de resistência mecânica serão variados para o compreendimento das limitações mecânicas do modelo físico.

6.1 Variação dos parâmetros de força sob aplicação de força externa

Para entender a influência de cada uma das 5 forças na conservação da estrutura física da célula, variamos de forma independente os 5 parâmetros das forças sob a aplicação de uma força externa de igual intensidade em todas as partícula da membrana em direção a uma placa fixa (figura 39). As forças foram aplicadas por um tempo longo contra a parede e, então, o tempo foi dobrado sem nenhuma força externa aplicada para entender o papel de restauração das forças sobre a estrutura celular.

6.1.1 Variação do coeficiente da força de volume

Mantiveram-se constantes os outros coeficientes de força de ligação ($k_s = 100$), flexão ($k_b = 10$), área local ($k_{al} = 10$) e área superficial ($k_{ag} = 10$), e variou-se o coeficiente da força de volume (Kv - o qual foi mantido fixo em $k_v = 30$ nas seções seguintes) (foram escolhidos os melhores valores dos coeficientes com base nos gráficos do capítulo 5). Então mediu-se o volume da célula ao longo do tempo (figura 40).

O volume da célula, após entrar em contato com a placa fixa (*t=44*) começa a diminuir do seu valor de equilíbrio (3,65). O valor máximo da diferença entre o volume instantâneo e o volume de equilíbrio diminui quanto maior o coeficiente de volume, indicando que a célula mantém o seu volume de forma mais resistente com coeficientes mais altos. Entretanto, todos os coeficientes utilizados foram capazes de recuperar o volume de equilíbrio da célula após determinado período.

6.1.2 Variação do coeficiente da força de área global

Analogamente ao experimento anterior, porém agora variando-se o coeficiente de área global (*Kag*) e medindo-se a área total da célula para entender a influência sob a área, obteve-se a figura 41. Percebemos que a área mínima da esfera é alcançada, para todos os coeficientes, em *t=85* e uma maior taxa de recuperação da área acima de *t=350* alcançando um máximo local

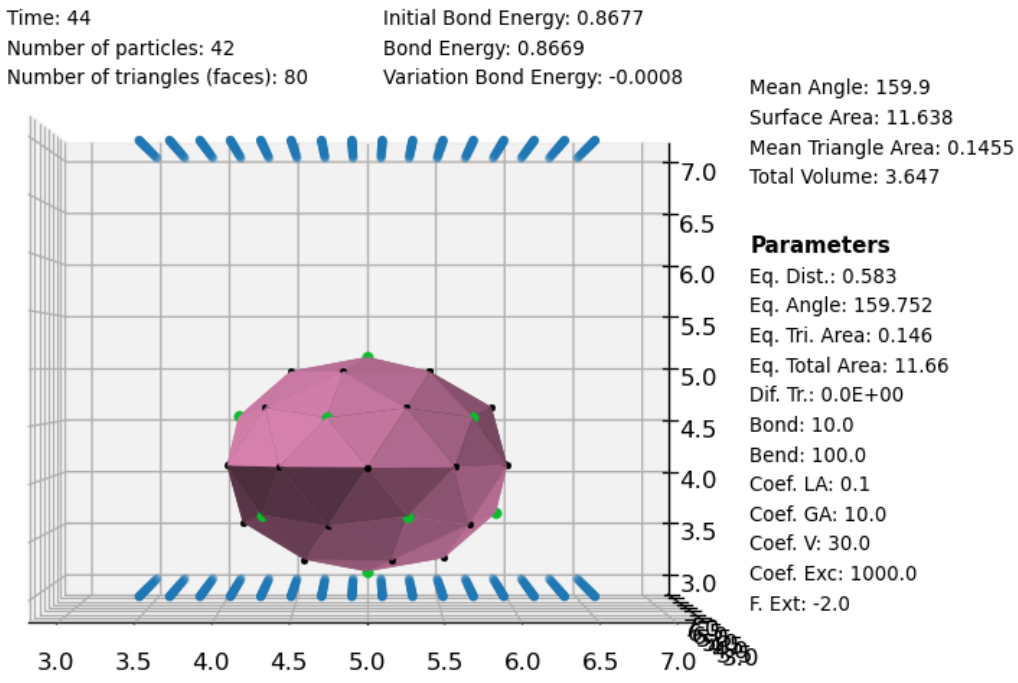


Figura 39 – Frame do experimento de aplicação de força externa em direção a uma placa fixa.

em *t=372*. Após, uma diminuição da taxa de recuperação acima de *t=372*. Para entender esse comportamento, plotamos todos os parâmetros medidos para $K_{ag} = 1$ e obtivemos a figura 42.

Observamos que, em *t=85*, a distância média entre as partículas alcança seu mínimo antes da força começar a recuperação da distância de equilíbrio. Quando as distâncias entre as partículas começam a aumentar, consequentemente a área superficial aumenta, então, por isso, temos o mínimo em *t=85*. Novamente, em *t=372*, a distância média é máxima, o que torna a área superficial maior. Após isso, a distância diminui em direção ao equilíbrio, o que faz com que a área superficial aumente em uma velocidade menor, pois as partículas estão mais próximas. Para comprovar essa conclusão, repetimos o experimento com a variação do coeficiente de força de área global e aumentamos o coeficiente de ligação (*bonding*) de $K_s = 10$ para $K_s = 30$ com o objetivo de impedir o aumento das distâncias entre as partículas (figura 43). e, consequentemente, diminuir a oscilação da área global nesse ponto. Os resultados obtidos na figura 44 diminuem drasticamente a oscilação, a tornando quase nula, o que corrobora a conclusão acima.

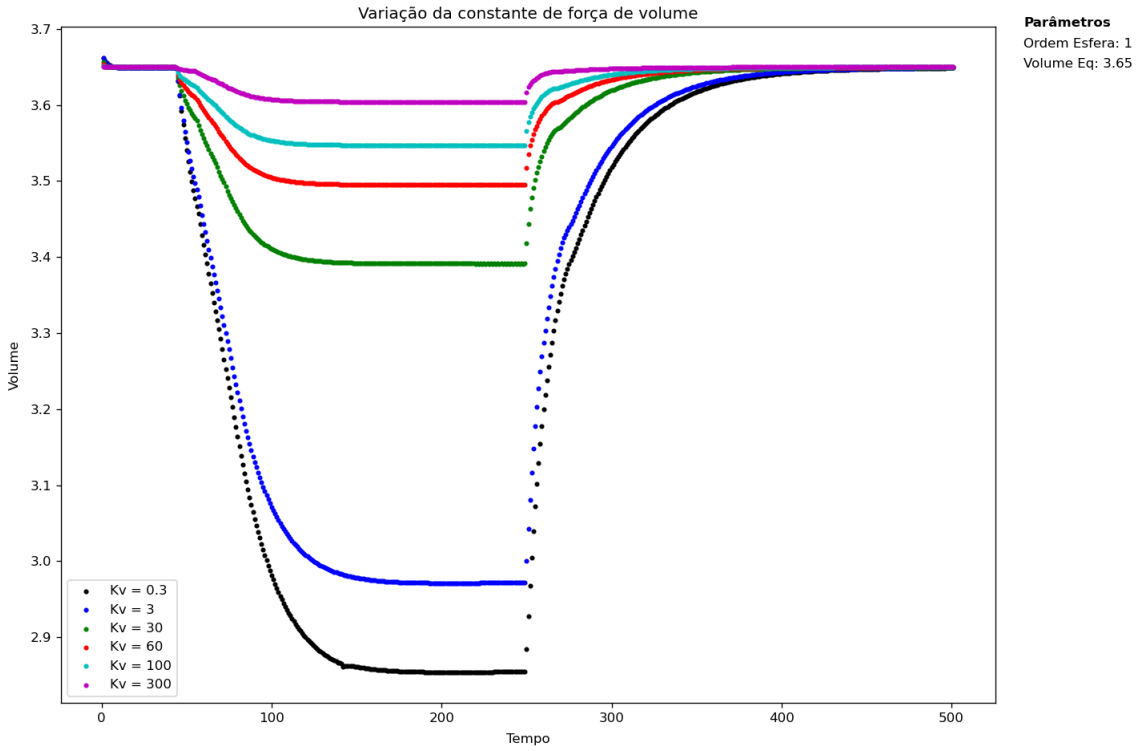


Figura 40 – Experimento com medida de volume ao longo do tempo para diferentes coeficiente de volume.

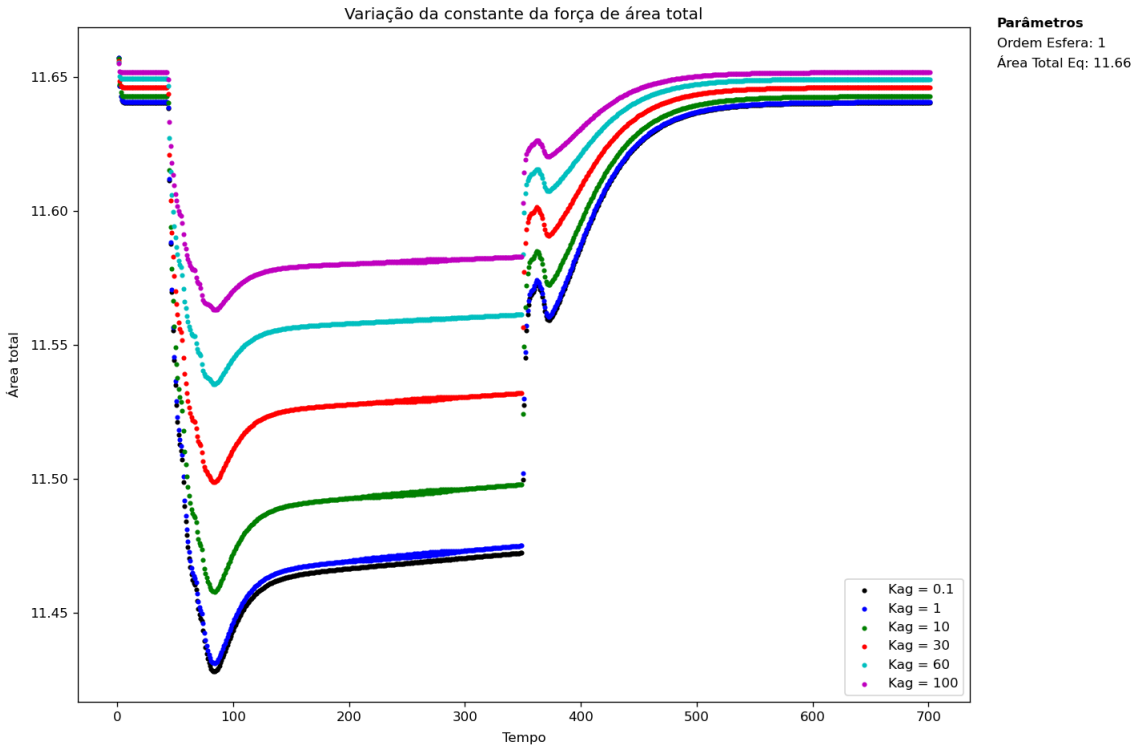


Figura 41 – Experimento com medida da área superficial ao longo do tempo para diferentes coeficiente de área global.

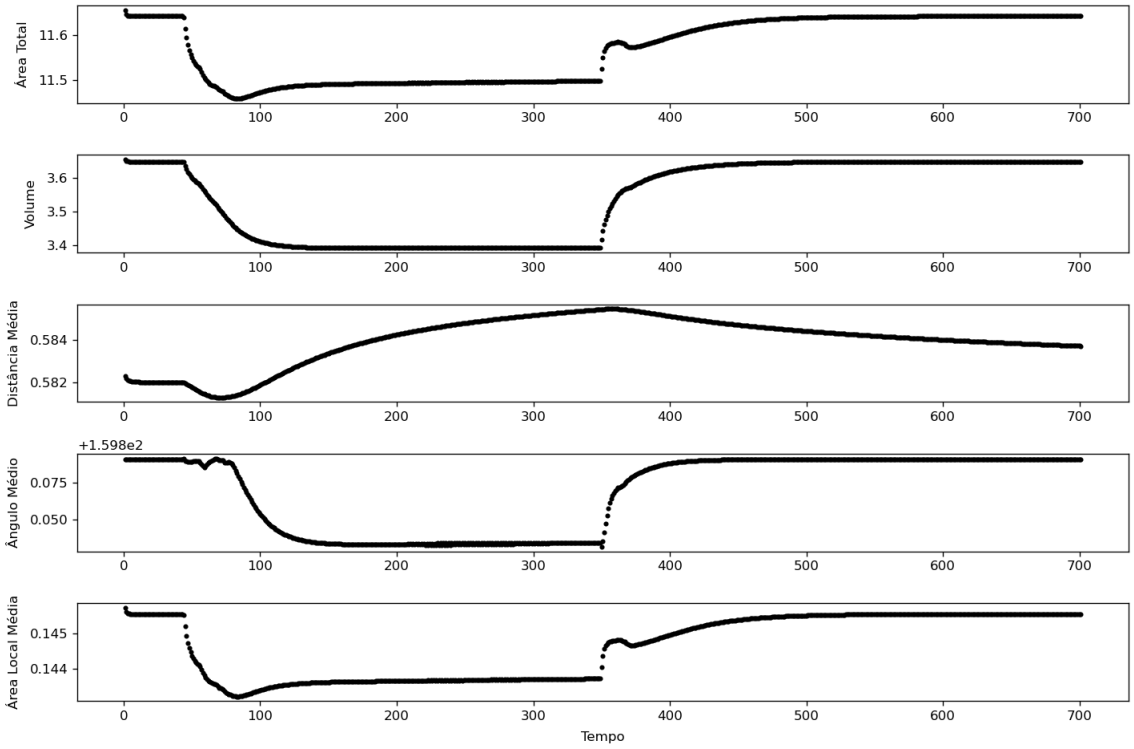


Figura 42 – Experimento com medida da área superficial, área local, volume, distância média das partículas e ângulo médio dos triângulos para $K_s = 10$.

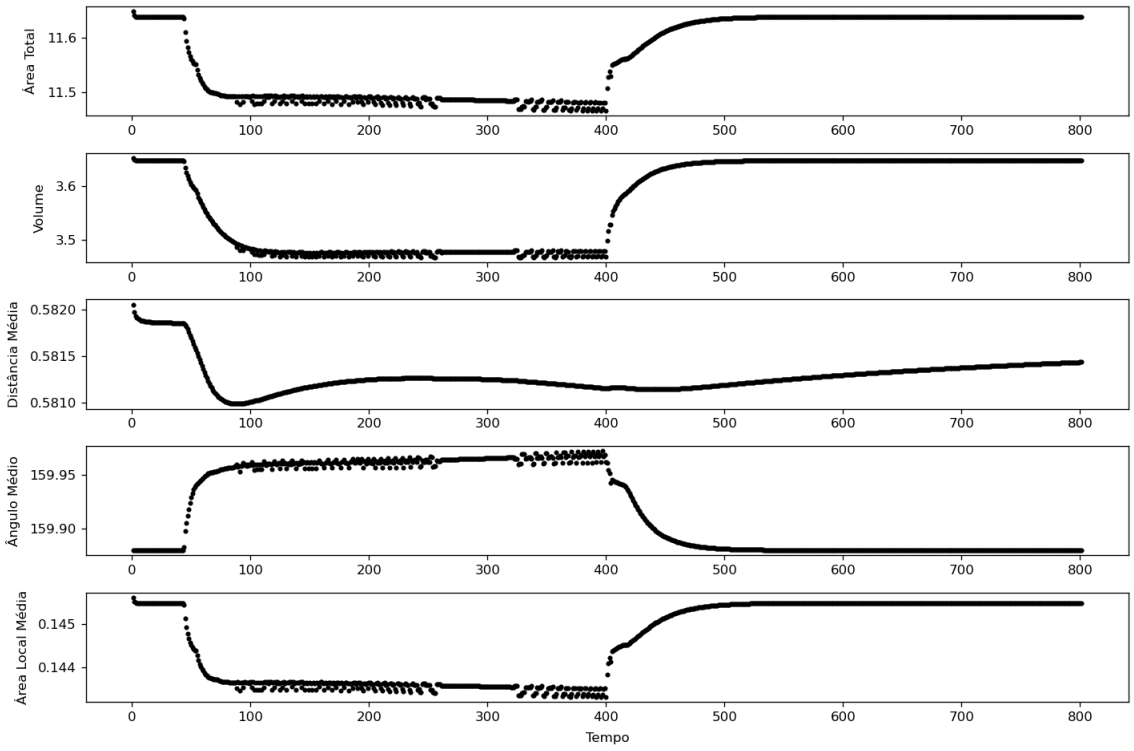


Figura 43 – Experimento com medida da área superficial, área local, volume, distância média das partículas e ângulo médio dos triângulos para $K_s = 30$.

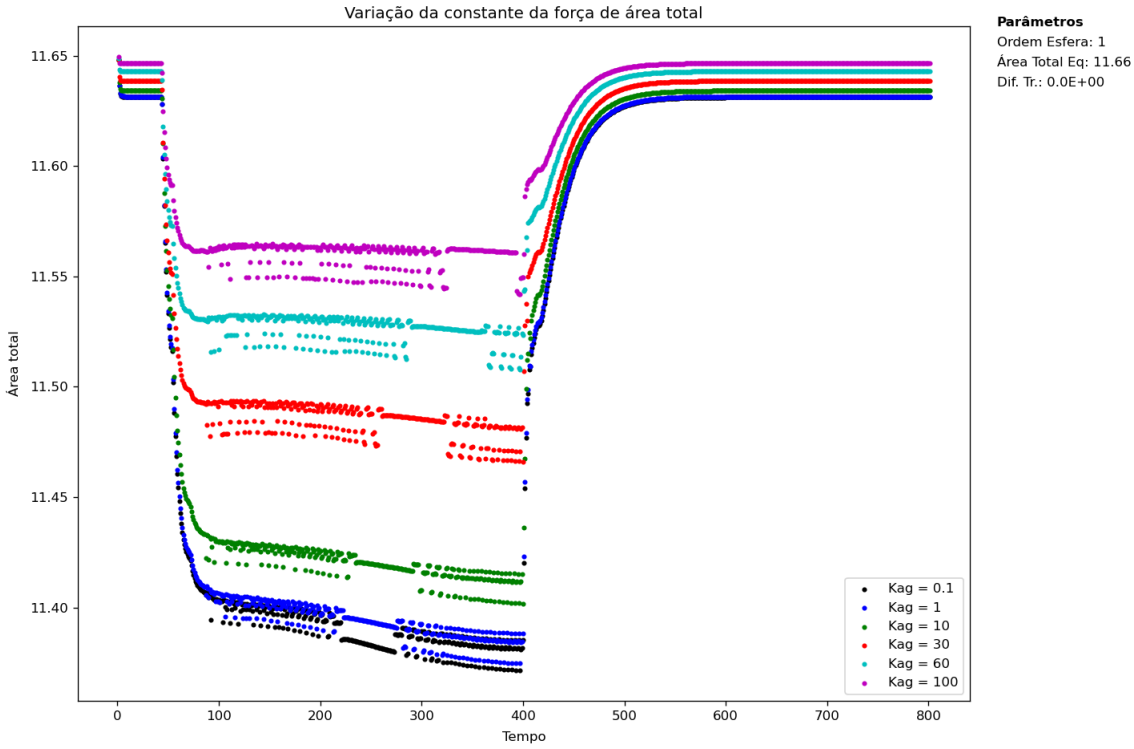


Figura 44 – Experimento com variação do coeficiente de força de área global (K_{ag}) para $K_s = 30$.

6.1.3 Variação do coeficiente da força de área local

Variando-se o coeficiente da força de área local e medindo-se a média das áreas dos triângulos do sólido no tempo, obtém-se a figura 45. Novamente, temos o mesmo comportamento e explicação para os tempos de *t=85* e *t=412*. O comportamento é diferente apenas para *Kal=0.1*. O que ocorre é que, como o coeficiente de força de área local é muito baixo, as áreas dos triângulos em contato com a superfície ficam livres para serem expandidas conforme a esfera é comprimida (vídeo 1 - para *Kal=0,1*), o que não acontece para coeficientes maiores (vídeo 2 - para *Kal=1*). Por isso, para coeficiente baixos, há uma aumento geral na área local.

Nota-se, também, que as curvas de *Kal=30, 60 e 100* oscilam entre três valores durante o estado de compressão. Isto acontece porque as partículas inferiores, a partir da interação com a parede, ficam oscilando conforme o vídeo 3, enquanto ficam fixas para outros valores (*kal=10*) conforme o vídeo 4.

6.1.4 Variação do coeficiente de flexão

Variando-se o coeficiente da força de flexão e medindo o ângulo médio entre os pares de triângulos obtemos a figura 46. Em *t=44* a esfera entra em contato com a placa, então a superfície inferior se torna reta, ou seja, os ângulos entre os pares de triângulos inferiores está menor do que o equilíbrio, por isso todas as curvas apresentam queda nesse tempo. Conforme a esfera vai sendo comprimida na parede, ela expande suas laterais, o que causa um aumento entre os ângulos dos pares de triângulos laterais, aumentando a média do ângulo para todas

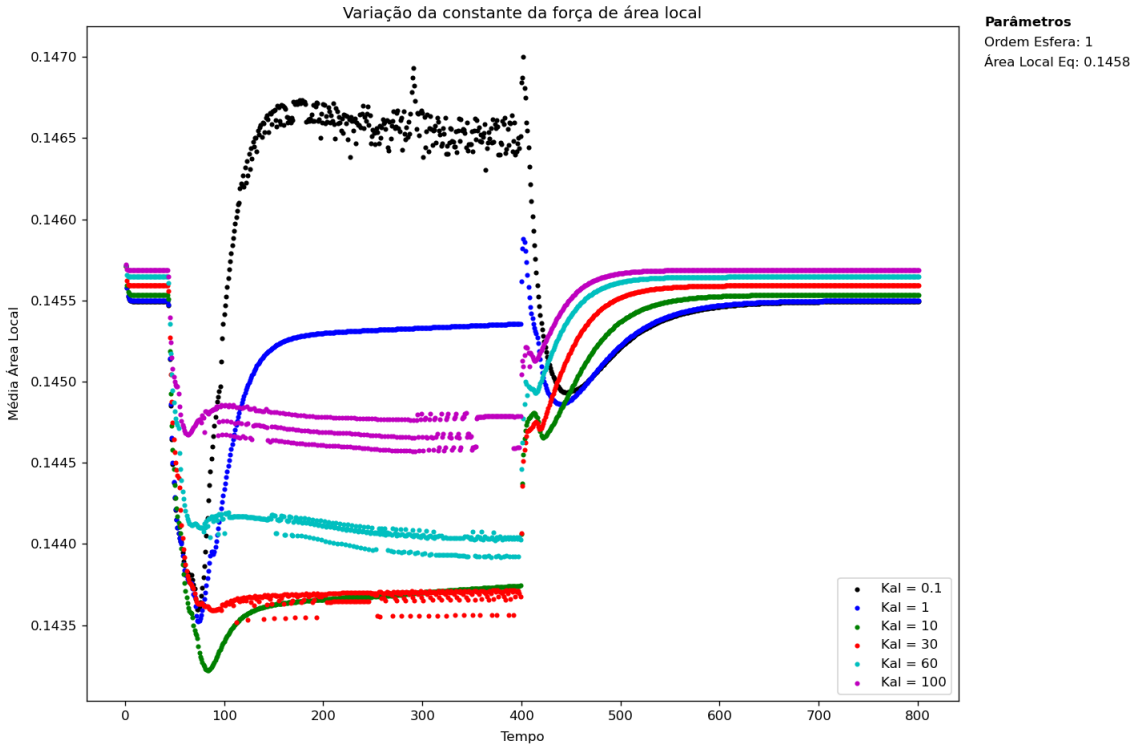


Figura 45 – Experimento com medida da área média dos triângulos ao longo do tempo para diferentes coeficiente de área local.

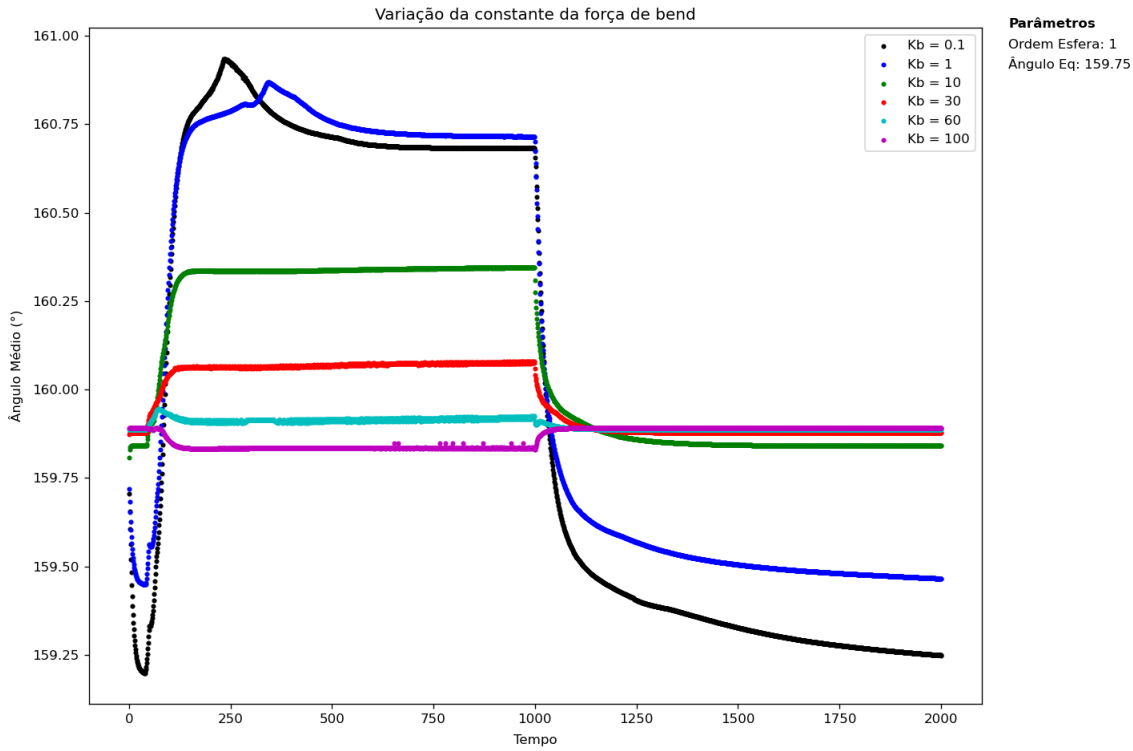


Figura 46 – Experimento com medida do ângulo médio entre os pares de triângulos ao longo do tempo para diferentes coeficiente da força de *bending*.

as curvas. A máxima diferença entre a média do ângulo instantâneo e o ângulo de equilíbrio depende do coeficiente. Quanto maior o coeficiente, menos os ângulos saem do equilíbrio e menor a diferença. Por fim, em *t=1000* a força externa é desligada e a força de *bending* tende a retornar a média para o ângulo padrão. Percebe-se que as curvas com maior coeficiente retornam com maior rapidez para o ângulo de equilíbrio.

6.1.5 Variação do coeficiente de ligação

Variando-se o coeficiente da força de ligação e medindo a média da distância entre as partículas, obteve-se a figura 47.

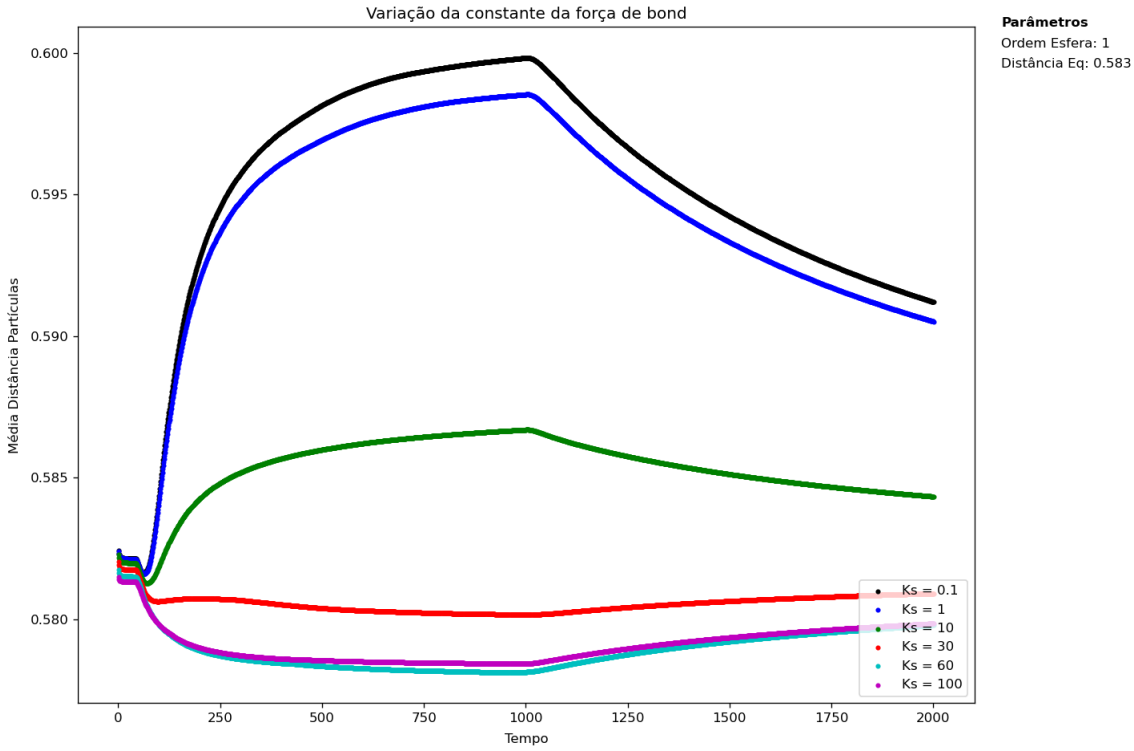


Figura 47 – Experimento com medida da distância média entre os pares de partículas ao longo do tempo para diferentes coeficientes da força de ligação.

Para valores pequenos do coeficiente de ligação (0,1, 1, 10), a distância média entre as partículas aumenta, pois o potencial permite que a esfera seja comprimida e aumente sua área superficial de contato com a placa (vídeo 5), o que aumenta a distância entre as partículas. Para valores maiores (aproximadamente 30 em diante), a esfera não é tão comprimida (vídeo 6) e, como visto no gráfico, a tendência é que a média da distância diminua enquanto em contato com a parede, pois as partículas em contato com a parede são empurradas para cima, ficando mais próximas as suas vizinhas e diminuindo a distância média.

6.2 Deformação linear por tensão constante no tempo

Para entender o comportamento da célula sob um estado de compressão de forças constantes no tempo, mediu-se a deformação linear (*strain*) da célula sob diferentes forças de compressão conforme o experimento citado na seção 3.2. Como pode-se observar na figura 48, a célula é colocada entre duas placas paralelas e partículas da superfície são fixadas em ambas as placas (partículas na cor branca).

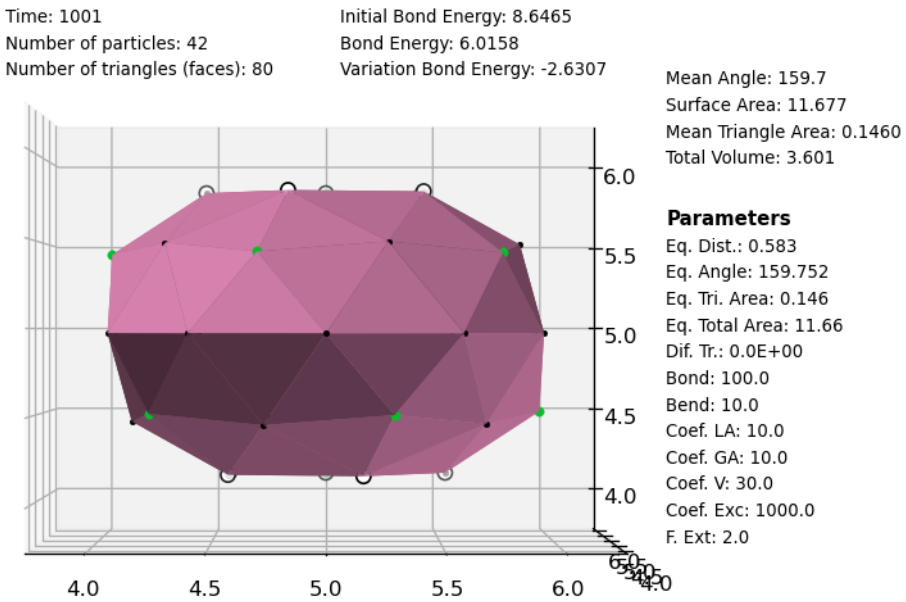


Figura 48 – Experimento virtual de deformação a tensões constantes. As partículas em branco são as partículas em contato com as placas paralelas.

A parede superior é livre para movimentação e a parede inferior é mantida fixa. Então uma tensão de tração constante é aplicada na placa superior e a deformação é medida ao longo do tempo. Após chegar na deformação máxima, a força de tração é zerada. Então a célula fica livre para retornar a sua deformação inicial como pode ser visto no vídeo 7.

São feitas curvas de deformação para diferentes tensões aplicadas (0,1, 1, 3, 5, 7 e 10 *N/partícula*). Como se pode ver na figura 49, quanto maior a força de tensão de tração, maior é a deformação linear da célula. Para uma força de tração de 10 *N/part*, a deformação linear alcança 23% da deformação inicial, enquanto que para uma força de 0,1 *N/part*, a deformação é inferior a 1%.

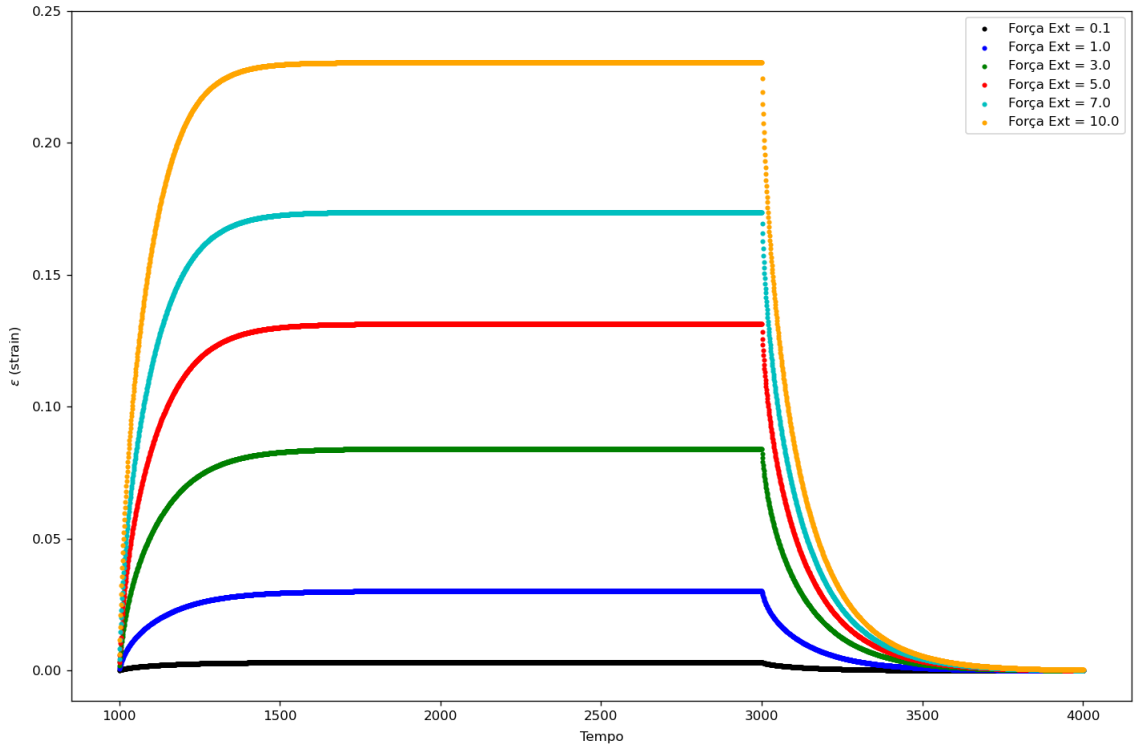


Figura 49 – Curvas de deformação linear da célula para diferentes tensões de tração constantes ao longo do tempo.

6.3 Comportamento de deformação x tensão da célula

Uma vez entendido o comportamento da célula ao longo do tempo para uma força de tração, é interessante entendermos a relação entre a deformação máxima com a força de tração aplicada na célula. Para isso, o mesmo experimento anterior foi feito para tensões de 0.1 a 50 *N/part* e a deformação máxima foi medida em função da tensão de tração aplicada.

Obtivemos então a figura 50, que mostra uma relação de lei de potência entre a tensão aplicada e a deformação da célula de forma que para uma tensão de 1 *N/part* tem-se uma deformação de aproximadamente 3% enquanto que para uma tensão de 80 *N/part* tem-se uma deformação de 100%. Nota-se que há dois regimes de comportamento, pois a inclinação da curva para deformações menores (abaixo de 10%) é diferente da inclinação para deformações maiores (acima de 10%). Realizando um ajuste logarítmico para os dois regimes, obtemos a figura 51. Para deformações baixas (região 1) a relação entre tensão e deformação possui um coeficiente angular de 3,74, enquanto que a região de maiores deformações (região 2), o coeficiente angular se torna maior (4,36). A transição entre os dois regimes indica que conforme a deformação aumenta, é necessário maior força para deformar o sólido. Uma vez que a magnitude das forças de restauração dependem do quanto a geometria do sólido é diferente da conformação de equilíbrio, as mesmas agem de forma diferente ao restaurar a estrutura.

O comportamento geral da curva tensão-deformação da figura 50 é similar ao comportamento experimental obtido por Micoulet, Spatz e Ott (2005) (figura 52) em experimento

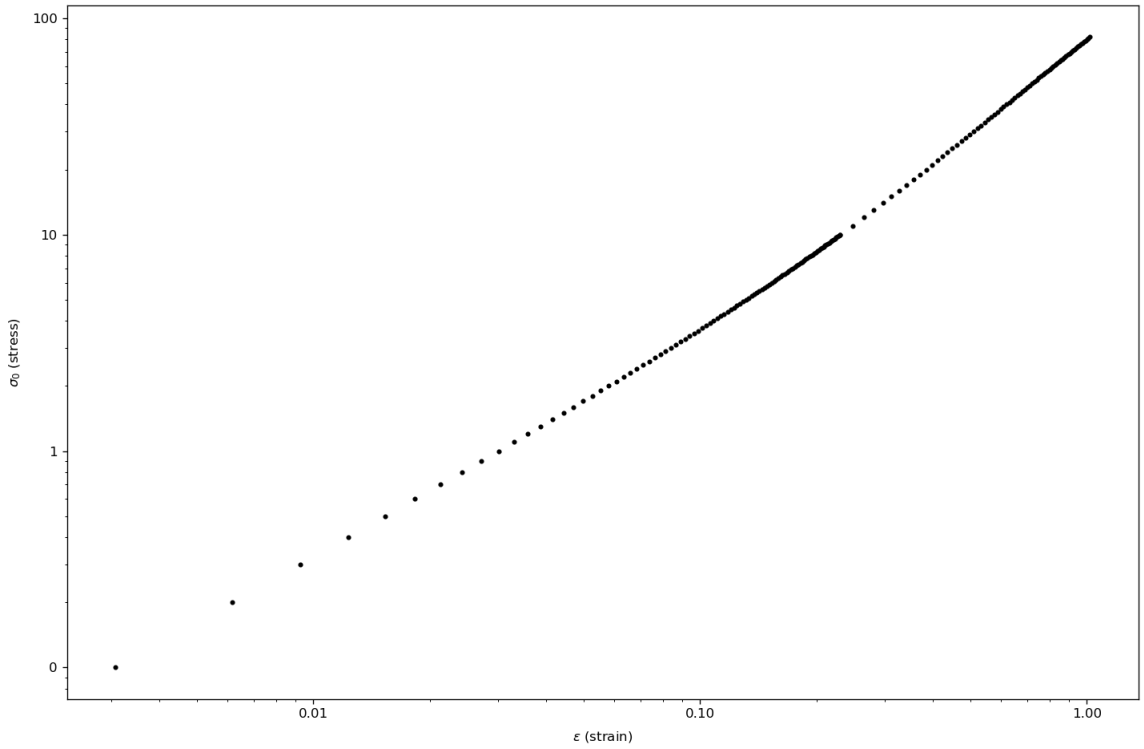


Figura 50 – Curva de deformação linear por tensão de tração da célula.

semelhante de tração por placas paralelas. Isso abre a possibilidade para que os parâmetros do modelo sejam ajustados para reproduzir a magnitude de outros resultados reológicos experimentais celulares encontrados na literatura, além de que os experimentos virtuais podem ser realizados diversas vezes para estudos estatísticas e de parâmetros, diferentes dos experimentos com células reais que dificilmente reproduzem os resultados ao serem realizados mais de uma vez na mesma amostra (células). Este fato pode ser observados na figura 52, em que os números indicam a ordem em que o experimento foi feito para a mesma célula. As curvas diferem entre si, pois é difícil reproduzir as mesmas curvas uma vez que as amostras (células) podem ser danificadas ou sofrerem alterações estruturais após a realização dos experimentos.

6.4 Superfície de Contato Parcial (*S*) da célula em relação ao coeficiente de flexão

Por último, para entender o quanto a célula "adere" à superfície, medimos a superfície de contato parcial (*S*), definida na seção 3.2. O coeficiente mede quantas partículas da superfície da célula estão em contato com a superfície. O coeficiente foi medido para diferentes coeficientes de *bending*, pois sabemos que o *bending* tem uma forte relação com o formato da célula, o que define o número de partícula em contato com a superfície. Para medir essa relação, foi aplicada uma força externa e constante nas partículas da célula em direção a uma parede para diferentes coeficientes de *bending* (vídeo 8 - para *Kb=100*).

Obtivemos a seguinte relação dada pela figura 53. Percebemos que quanto maior o coeficiente de *bending*, menor é a superfície de contato parcial (S), pois a força de *bending*

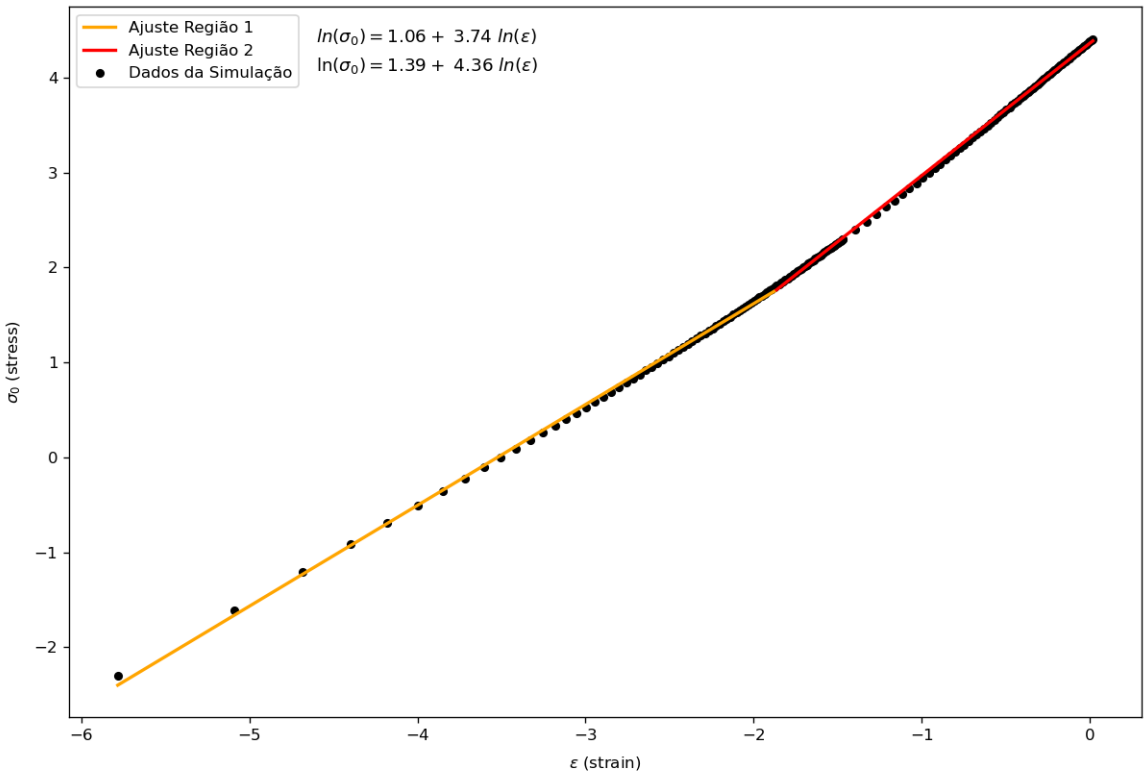


Figura 51 – Ajuste exponencial para dois regimes de deformação da célula.

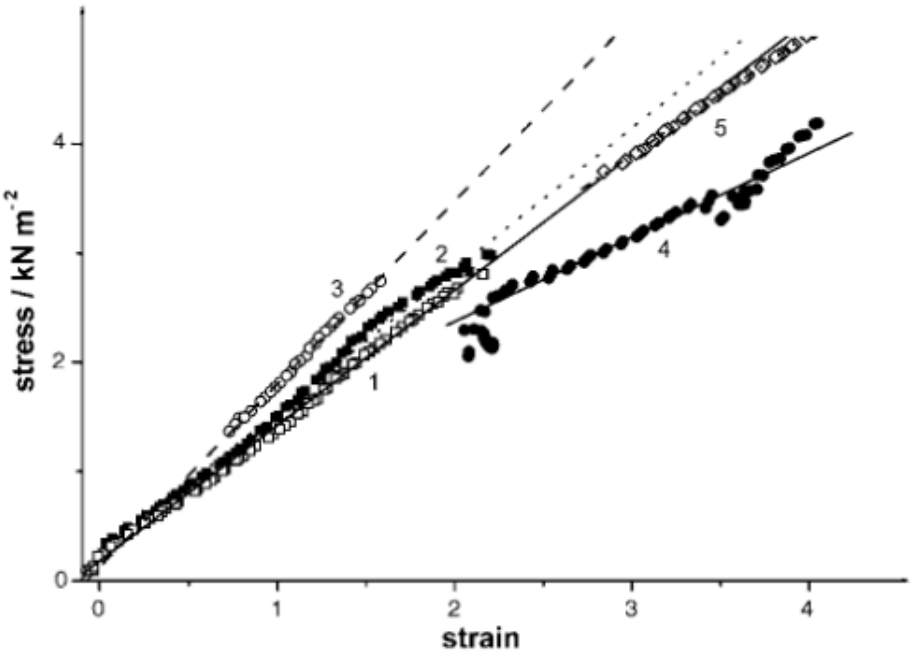


Figura 52 – Resultado experimental da curva tensão-deformação obtido por Micoulet, Spatz e Ott (2005).

mantém a estrutura da célula mais estável, o que diminui o número de partículas em contato com a superfície ao ser aplicada uma força de compressão.

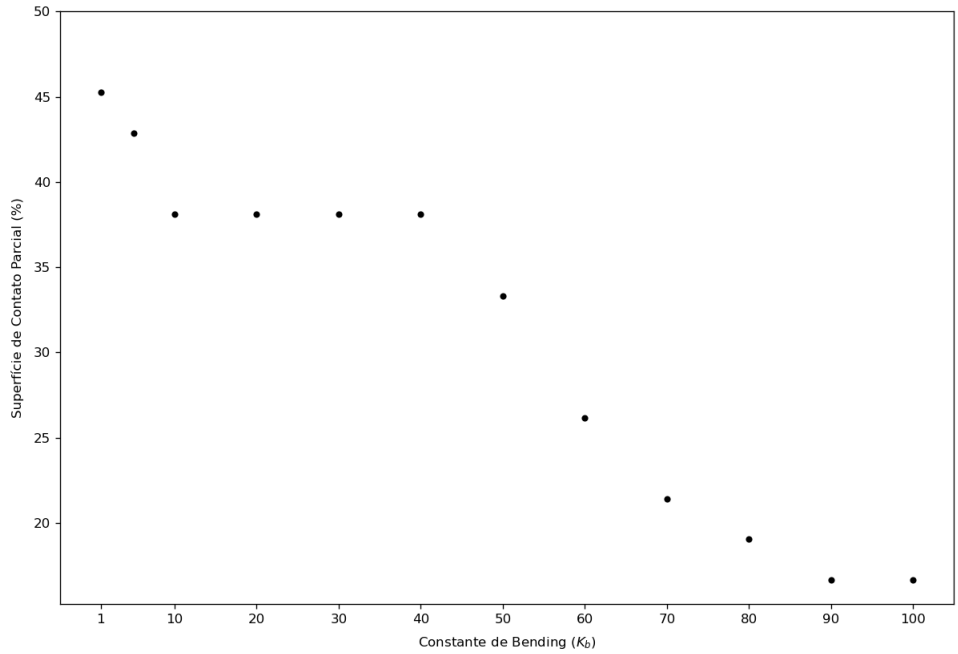


Figura 53 – Superfície Parcial de Contato (S) com a variação do coeficiente de *bending*.

6.5 Tempo de simulação e aproximação de tempo para simulação multicelular

Para a implementação de interações da célula com as paredes foi implementado o algoritmo das caixas, algoritmo que subdivide o sistema em espaços menores para que apenas as interações entre partículas próximas sejam calculadas, o que aumenta a eficiência do cálculo computacional.

Em relação a simulação, para cada esfera, o parâmetro que mais influencia no tempo de simulação da dinâmica da esfera é o tamanho da caixa, pois o algoritmo realiza os cálculos de interação entre as partículas a partir de quantas caixas estão ocupadas por partículas. Se o tamanho da caixa é muito pequeno (menor do que a distância entre as partículas), terá no máximo uma partícula por caixa, ou seja, o número de caixas calculadas será o maior possível, então o algoritmo calculará muitas interações entre diferentes caixas, o que aumenta o tempo da simulação. Por outro lado, se o tamanho da caixa for infinito (maior do que o tamanho da célula e paredes) e existirem muitas partículas no sistema, o algoritmo calculará interações entre todas as partículas existentes, o que também torna a execução do código extremamente lenta. Por isso, para encontrar um tamanho de caixa visando otimizar o tempo de simulação do código, foram feitas simulações para a primeira esfera com tempo de integração $dt = 0,001$ e número

de integrações $N_{ret} = 200$. O tamanho de caixa foi variado de 0,05 - 0,80. Os resultados são mostrados na figura 54.

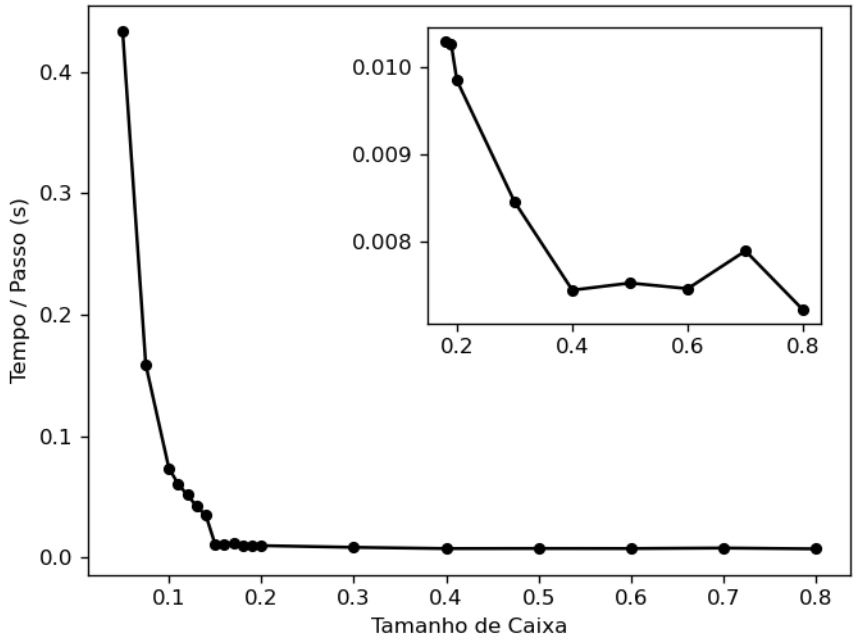


Figura 54 – Medida do tempo de execução da dinâmica da primeira esfera com ruído para diferentes tamanhos de caixa.

Percebe-se da figura 54 que, para tamanhos de caixa muito pequenos (abaixo de 0,1), o tempo de simulação é muito alto, pois o número máxima de caixas está sendo usado, então o algoritmo calcula muitas interações diferentes entre as caixas. Conforme o tamanho de caixa aumenta, o tempo diminui tendendo a ficar constante para tamanhos de caixa acima de de 0,4. Para otimizar o tempo desta simulação, é recomendado que sejam utilizados tamanho de caixa acima de 0,4.

Com o objetivo de se caracterizar o algoritmo em função da ordem da esfera (número de partículas) em termos de perspectivas para futuras simulações, escolheu-se o tamanho de caixa de 0,8 e variou-se a ordem do sólido para o mesmo tipo de simulação. Para 1000 iterações de $dt = 0,001$, obteve-se a figura 55. Da figura 55, concluímos que, para uma simulação de sólido de segunda ordem (162 partículas), para 1000 iterações, leva-se em torno de 27 segundos. A relação é exponencial para a ordem da esfera de forma que, para a terceira esfera (642 partículas), leva-se em torno de 400 segundos.

Além disso, fizemos uma aproximação do tempo que demoraria para realizar uma simulação multicelular, cenário em que ocorre a simulação da dinâmica e a interação simultânea de diversas células. A aproximação funcionou da seguinte forma: foram criadas 6 paredes ao redor da célula de forma que o espaçamento das partículas na parede é igual às distâncias entre as partículas da membrana (figura 56). O ruído foi ativado e foram feitas simulações onde as

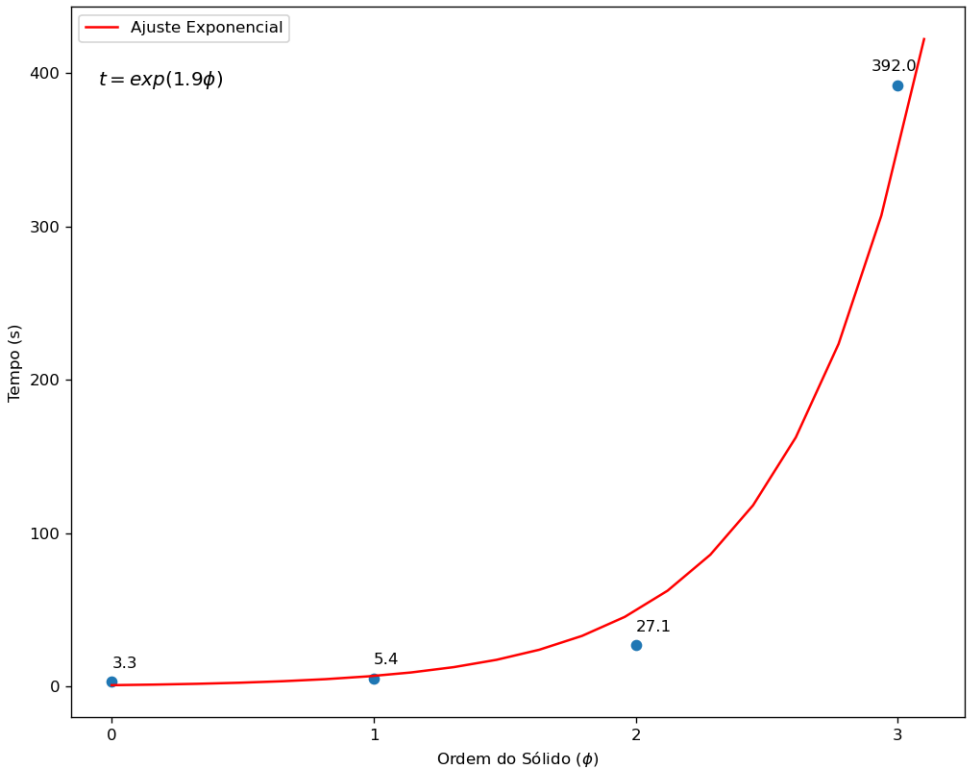


Figura 55 – Medida do tempo de execução da dinâmica para diferentes ordens do sólido com ajuste exponencial.

partículas da membrana interagem com as partículas da parede, simulando um cenário onde uma célula interage com outras 6 células na sua superfície (lados esquerdo, direito, cima, baixo, frente e traseira). No vídeo 9 é possível observar a simulação feita.

Foram feitas simulações de 1000 iterações com tempo de integração $dt = 0,001$ para as esferas com diferentes ordens de segmentação. Os resultados obtidos encontram-se na figura 57. Nota-se que uma simulação de 1000 iterações para uma esfera de segunda ordem (162 partículas) interagindo com 6 paredes, aproximadas por 6 outras esferas, demoraria em torno de 500 segundos. Somando-se o tempo de simulação da dinâmica das outras 6 esferas da figura 55, teríamos um tempo de simulação de 6 dinâmicas de 162 segundos somados ao tempo de interação com dinâmica de 500 segundos totalizando 662 segundos para cada 7 esferas. Então, para um cenário de 1000 iterações com 70 células cada uma interagindo com 6 células na superfície, teríamos um tempo de simulação total de 110 minutos para um notebook com processador intel core i5 8250U de 8ª geração com velocidade de processamento de 1,6GHz até 3,2GHz (modelo do *hardware* com especificações).

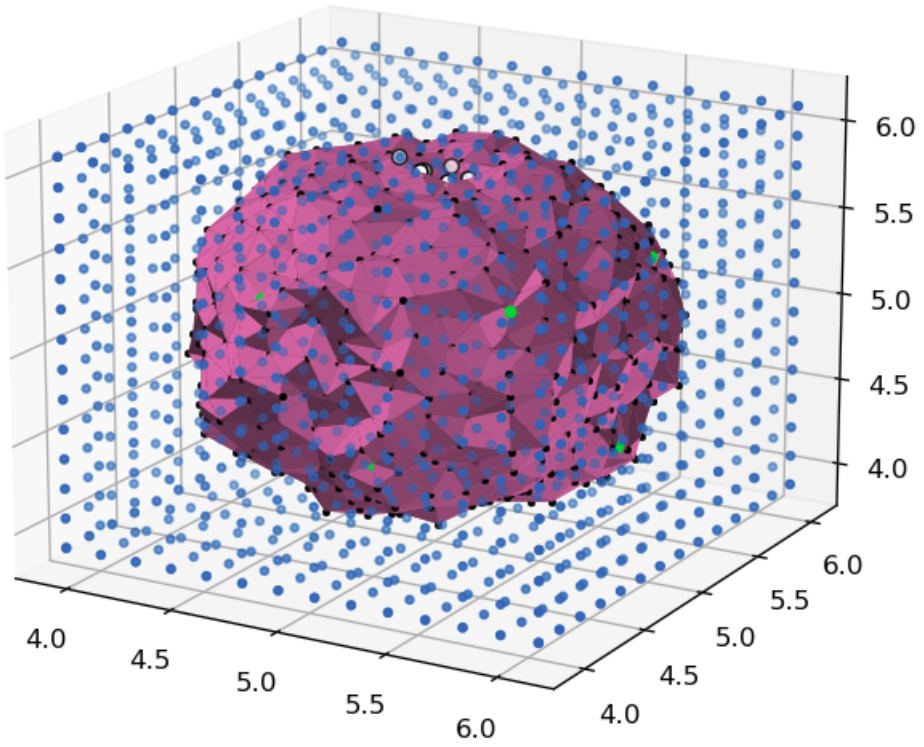


Figura 56 – *Frame* da simulação de uma célula de terceira ordem interagindo com 6 paredes laterais.

6.6 Tamanho dos dados de entrada em função da ordem da esfera

Um fator determinante para a definição do tempo da simulação são os arquivos de entrada que são transferidos do python para o fortran nas integrações de tempo. Por isso, foi feita uma análise preliminar da relação do tamanho dos arquivos de entrada em relação a ordem da esfera na figura 58. Percebe-se que o tamanho dos arquivos é menor do que 50 MBs para esferas de até ordem 3. Entretanto, para a 4ª esfera o tamanho dos dados alcança acima de 600 MBs, fato que deve ser levado em consideração para um futuro cenário de simulações para a 4ª esfera e/ou multicelular, pois, dependendo do número de passos da evolução temporal definido para o fortran, a transferência de dados entre python e fortran pode alcançar milhares de vezes para apenas uma simulação, de forma que se torna necessário modificar o algoritmo para diminuir o número de transferência de dados ou mudar a forma de armazenamento dos dados(por exemplo, eliminar o armazenamento de espaços vazios em vetores que não são usados na simulação para então diminuir o espaço em memória).

6.7 Espaço de parâmetro tensão-deformação por coeficiente de *bending* da primeira esfera

Para entender o comportamento da deformação sob tensão da esfera, medimos o módulo de *Young* (definido na seção 3.2) variando o coeficiente de *bending* da esfera, obtendo, então, o espaço de parâmetro da esfera em relação ao coeficiente que define a flexão do sólido. Como nos

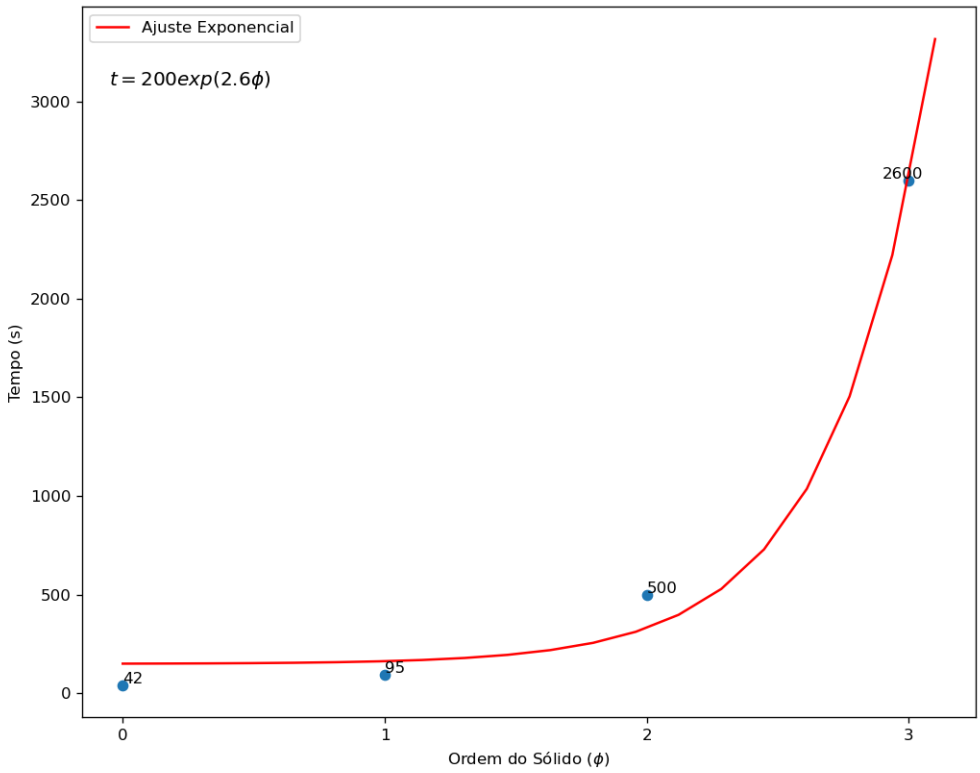


Figura 57 – Medida do tempo de execução da dinâmica para diferentes ordens do sólido interagindo com 6 paredes.

experimentos anteriores, foi feito o experimento de tração para diferentes coeficientes de flexão e o módulo de *Young* foi medido. Obteve-se a figura 59.

Percebem-se duas regiões de regimes diferentes: a primeira região para coeficiente de flexão menores de 5 e a segunda para maiores de 5. Na primeira região, o módulo de *Young* diminui conforme o aumento do coeficiente. Nesta região, o sólido tracionado não se aproxi a de uma esfera tracionada (figura 61). Por ter um coeficiente de *bending* pequeno, o sólido mantém sua estrutura próxima a de um icosaedro, mudando então suas respostas às deformações. De forma que, no experimento de tração, o volume de equilíbrio na tração máxima é *menor* do que o volume de equilíbrio (figur 60). Esse comportamento faz com que o módulo de *Young* diminua conforme o coeficiente de flexão aumenta, pois o sólido se torna cada vez mais parecido com uma esfera tracionada, aumentando seu volume sob tração em direção ao volume de equilíbrio. Para coeficientes de flexão maiores que 5, o volume sob tração aumenta acima do volume de equilíbrio (figura 60), e, então, o módulo de *Young* aumenta conforme o aumento do coeficiente de flexão.

Analisando o segundo regime, em que o sólido é uma esfera, como mostra a figura 59, quanto maior o coeficiente de *bending*, maior é o módulo de *Young*, indicando que a célula se

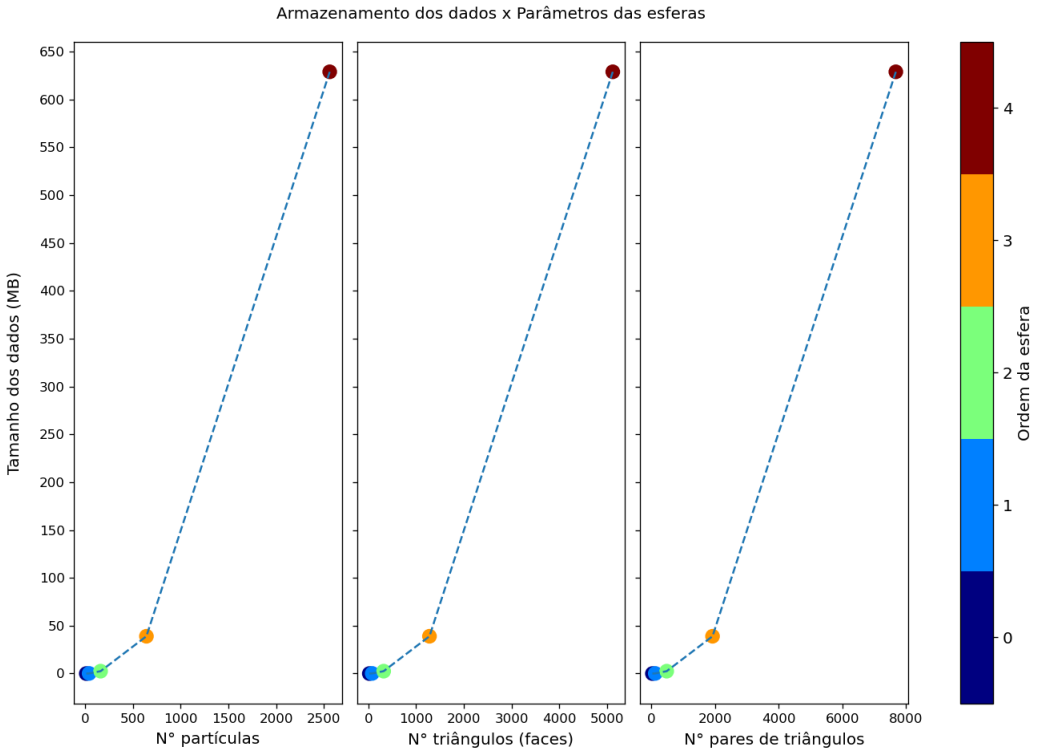


Figura 58 – Tamanho dos arquivos de entrada em função da ordem da esfera.

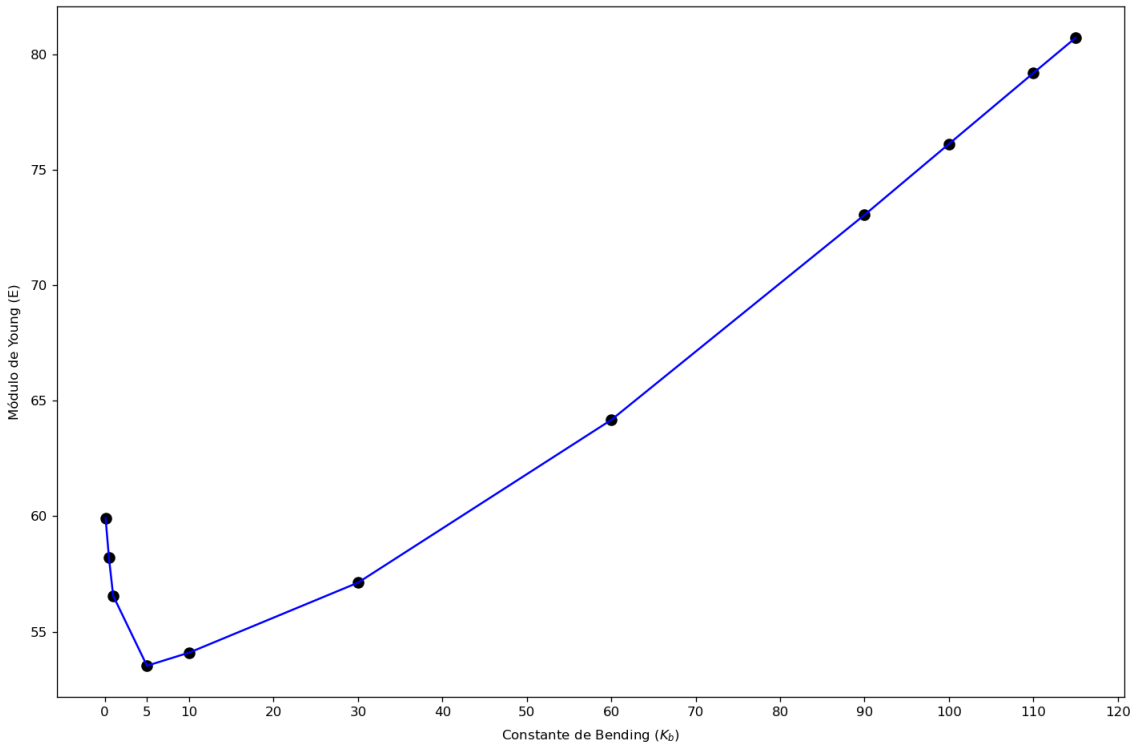


Figura 59 – Espaço de parâmetro da tensão-deformação em relação ao coeficiente de *bending* para a primeira esfera.

torna mais rígida exigindo forças maiores para deformá-la. Esse resultado mostra que deve-se dar atenção aos limites de coeficientes utilizados no algoritmo a depender dos experimentos que

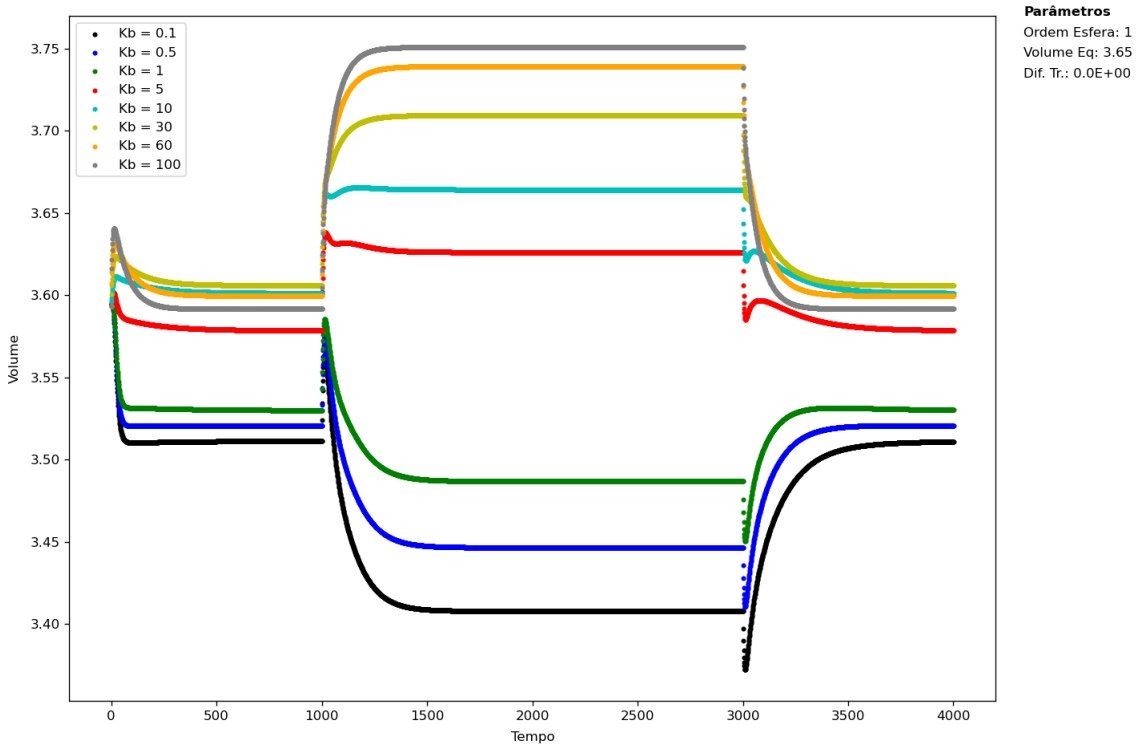


Figura 60 – Variação do volume da esfera sob tração nos dois regimes de comportamento do módulo de *Young* da primeira esfera.

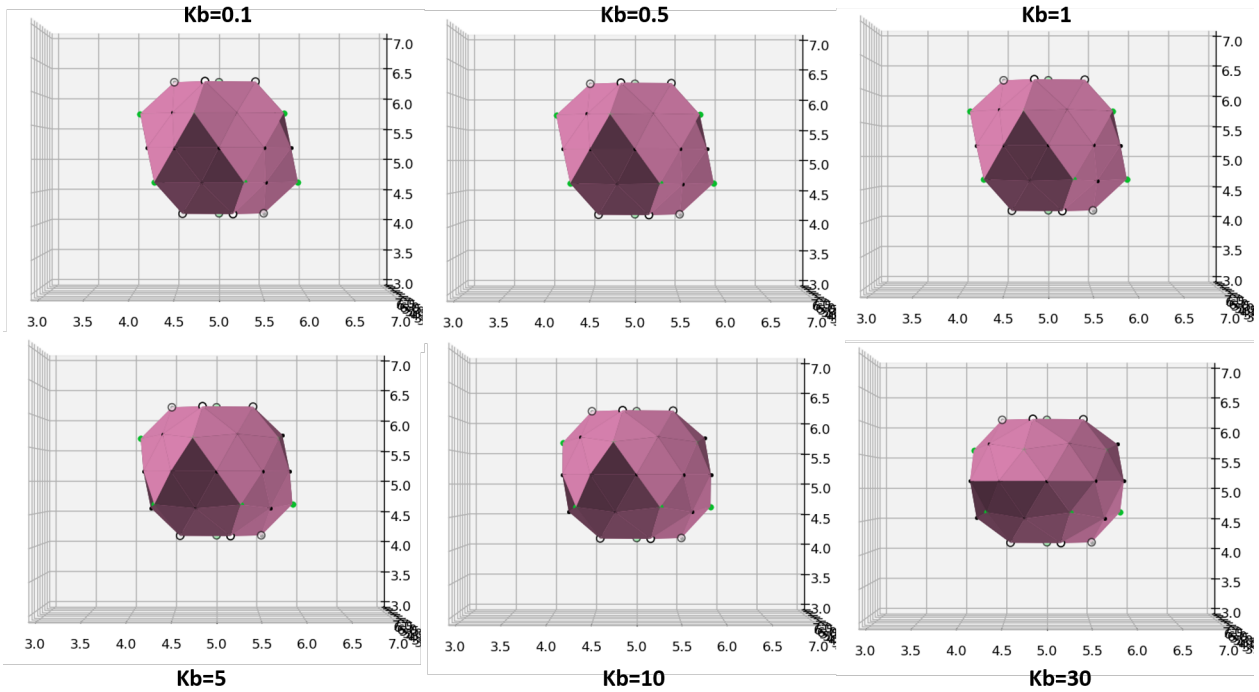


Figura 61 – Diferentes geometrias do sólido em estado de tração máxima para diferentes coeficientes de flexão.

se deseja realizar, pois sua magnitude pode influenciar na geometria e, consequentemente, no comportamento do sólido sob ação de forças externas e interações e na sua dinâmica.

Parte VI

Conclusão

7 CONCLUSÃO

Neste trabalho revisamos os principais modelos e *softwares* utilizados para a simulação de células 3D (capítulo 2). Concluímos que a melhor alternativa é criar um programa com a utilização de *python* e *fortran*, pois a linguagem *python* permite a criação do modelo via orientação a objetos que facilita a criação e gerenciamento do código enquanto o *fortran* é a linguagem que otimiza a velocidade dos cálculos necessários permitindo a escalabilidade do modelo para um futuro cenário de simulações multicelulares. Em relação ao modelo, escolhemos adaptar o modelo de Tóthová, Jančigová e Bušík (2015) que utiliza uma série de potenciais que estabilizam a estrutura geométrica do sólido não sendo necessárias estruturas internas na célula, assim diminuindo o custo computacional.

Para a construção da estrutura tridimensional, foi utilizado um algoritmo recursivo de tecelagem triangular que constrói uma esfera a partir da ligação de partículas no espaço e, então, potenciais foram aplicados na estrutura. A maior complexidade do trabalho foram os testes para validar a aplicação dos potenciais individuais na estrutura. Como resultados, obtivemos curvas, para cada potencial, que demonstraram que estabilizam a estrutura como proposto inicialmente (capítulo 5), assim como entender a contribuição de cada potencial na estrutura celular quando todos estão aplicados. Além das curvas de validação dos potenciais, foram feitos testes visuais, pois mostraram, por exemplo, que a esfera se transforma novamente em um icosaedro (seção 5.0.4) dependendo da magnitude do potencial, aspecto que não pode ser encontrado nas curvas de validação. Os testes visuais também motivaram o estudo da distribuição das distâncias e ângulos da esfera (5.0.4) no seu estado inicial (maior aproximação de uma esfera perfeita), onde pode-se verificar a necessidade de preservar os ângulos iniciais dos pares de faces bem como as áreas iniciais de cada face para que o sólido fique estável no tempo.

Em relação à *performance* do algoritmo, obtivemos a curva do tempo de simulação em função da ordem da esfera/número de partículas aproximada para um cenário multicelular, em que constatamos que para simular 70 células de segunda ordem (162 partículas por esfera) onde cada esfera interage com 6 vizinhas para 1000 iterações de $dt = 0,001$, levaríamos, aproximadamente, 110 minutos para um notebook com processador intel core i5 8250U de 8ª geração com velocidade de processamento de 1,6GHz até 3,2GHz. Nesse cenário, seria possível estudar dinâmicas de movimento coletivo, divisão celular, migração etc; do ponto de vista tridimensional.

Em relação à reologia e o comportamento celular, realizamos medidas do módulo de *Young* da célula e comparamos qualitativamente com resultados experimentais. Obtivemos, tam-

bém, a relação entre o parâmetro superfície de contato parcial (S) e a magnitude do coeficiente de flexão (*bending*) mostrada na figura 53. Resultado importante para simulações de tensão cortical (contração do citoesqueleto cortical da célula), pois, como mostrado, aumentar o coeficiente de flexão, equivalente a aumentar a tensão cortical no espaço tridimensional, diminui a adesão da célula ao substrato, o que pode alterar a dinâmica celular, sendo necessário considerar ambos efeitos para o estudo. Ademais, a geometria da célula durante a fase de divisão celular está intimamente relacionada ao sucesso ou falha do processo. Estudos recentes de segregação celular (Miroshnikova et al. (2018)) mostraram que a tensão cortical celular e a adesão celular influenciam diretamente no desenvolvimento e proliferação de células e tecidos. Tanto a tensão cortical quanto a adesão são definidos pelo potencial de flexão no algoritmo utilizado no presente trabalho. Portanto, o algoritmo estudado aqui apresenta parâmetros que podem ser estudados na aplicação desse tipo de mecanismo de segregação.

7.1 Perspectivas

A partir da revisão das plataformas de computação paralela, pode-se identificar metodologias que podem tornar os cálculos mais rápidos no futuro, como o protocolo de comunicação MPI que possibilita a criação de *clusters* de GPU que podem suportar a infraestrutura que possibilita a aceleração na simulação para um cenário multicelular.

A longo prazo, como feito no trabalho de Liberman et al. (2018), em que foi utilizado o *framework* Unity3D para a simulação e criação de uma interface gráfica de fácil uso e implementação de simulações, podem ser utilizadas ferramentas de *software* livre como, por exemplo, o *framework* Blender. Este *framework* permite que uma simulação seja divida em módulos, especificamente *addons* no contexto do Blender, que são códigos que adicionam novas interações e funcionalidades à simulação, os quais são desenvolvidos pela comunidade de colaboradores. O *framework* também possui uma interface gráfica de fácil uso que permitiria que outros usuários utilizassem o modelo sem a necessidade de conhecimentos específicos de programação e paralelização de código.

Ademais, futuramente, pode-se tornar o modelo mais complexo com a adição de migração celular, crescimento, diferenciação, divisão etc. Isso permitiria entender processos complexos do sistema celular.

7.1.1 Modo de trabalho deste TCC

Esse trabalho foi feito por meio da orientação supervisionada, modelo de trabalho em que o doutorando da área de desenvolvimento do trabalho orienta o aluno com a supervisão do professor sênior. Em relação a minha experiência como aluno, gostaria de salientar que foi uma experiência de aprendizado ímpar, pois o doutorando focou em instruir as metodologias de trabalho do seu dia a dia que eu poderia utilizar para encontrar as respostas para as perguntas que surgiam, diferente de um modo mais tradicional em que o professor oferece um proposta de

trabalho bem direcionada desde o início. Esse modo de trabalho me aproximou e oportunizou conhecer o trabalho que um pesquisador da pós graduação realiza diariamente. Além disso, a proposta de trabalho foi construída em conjunto a partir das revisões bibliográficas, não surgindo assim de uma demanda já pensada anteriormente, o que me permitiu a experiência de desenvolver um projeto do zero.

Referências

ADRA, S. et al. Development of a three dimensional multiscale computational model of the human epidermis. *PloS one*, Public Library of Science, v. 5, n. 1, p. e8511, 2010. Citado na página 23.

ASANO, S.; MARUYAMA, T.; YAMAGUCHI, Y. Performance comparison of fpga, gpu and cpu in image processing. In: IEEE. *2009 international conference on field programmable logic and applications*. [S.l.], 2009. p. 126–131. Citado na página 90.

BLACK-SCHAFFER, D. *Introduction to OpenCL*. 2016. (Acesso em: 28 de Nov. de 2020). Disponível em: <https://www.youtube.com/playlist?list=PLiwt1iVUib9s6vyEqdpcgAq7NBRlp9mAY>. Citado na página 96.

BYRNE, H.; DRASDO, D. Individual-based and continuum models of growing cell populations: a comparison. *Journal of mathematical biology*, Springer, v. 58, n. 4-5, p. 657, 2009. Citado na página 25.

CAMLEY, B. A.; RAPPEL, W.-J. Physical models of collective cell motility: from cell to tissue. *Journal of physics D: Applied physics*, IOP Publishing, v. 50, n. 11, p. 113002, 2017. Citado 3 vezes nas páginas 8, 17 e 26.

CAPPUCCIO, A.; TIERI, P.; CASTIGLIONE, F. Multiscale modelling in immunology: a review. *Briefings in bioinformatics*, Oxford University Press, v. 17, n. 3, p. 408–418, 2016. Citado na página 18.

COMMUNITY, B. O. *Blender - a 3D modelling and rendering package*. Stichting Blender Foundation, Amsterdam, 1998. (Acesso em: 11 de Nov. de 2020). Disponível em: <http://www.blender.org>. Citado na página 23.

COMUNIDADE MATLAB, c. M. *Generate unit geodesic sphere created by subdividing a regular icosahedron*. 2015. (Acesso em: 28 de Nov. de 2020). Disponível em: <https://www.mathworks.com/matlabcentral/fileexchange/50105-icosphere>. Citado 2 vezes nas páginas 8 e 20.

COMUNIDADE STACK OVERFLOW, C. S. O. *Snapping vector to a point from a grid on a sphere (icosahedron)*. 2018. (Acesso em: 28 de Nov. de 2020). Disponível em: <https://stackoverflow.com/questions/48553298/snapping-vector-to-a-point-from-a-grid-on-a-sphere-icosahedron>. Citado 2 vezes nas páginas 8 e 34.

DESPRAT, N. et al. Creep function of a single living cell. *Biophysical journal*, Elsevier, v. 88, n. 3, p. 2224–2233, 2005. Citado na página 37.

DRASDO, D.; HOEHME, S.; BLOCK, M. On the role of physics in the growth and pattern formation of multi-cellular systems: What can we learn from individual-cell based models? *Journal of Statistical Physics*, Springer, v. 128, n. 1-2, p. 287, 2007. Citado na página 17.

FACHADA, N.; LOPES, V.; ROSA, A. Agent-based modelling and simulation of the immune system: a review. In: *EPIA 2007-13th Portuguese Conference on Artificial Intelligence*. [S.l.: s.n.], 2007. Citado na página 22.

GARDINER, B. S. et al. Discrete element framework for modelling extracellular matrix, deformable cells and subcellular components. *PLoS Comput Biol*, Public Library of Science, v. 11, n. 10, p. e1004544, 2015. Citado 4 vezes nas páginas 18, 19, 27 e 28.

IZAGUIRRE, J. A. et al. Compucell, a multi-model framework for simulation of morphogenesis. *Bioinformatics*, Oxford University Press, v. 20, n. 7, p. 1129–1137, 2004. Citado 2 vezes nas páginas 22 e 23.

KARIMI, K.; DICKSON, N. G.; HAMZE, F. A performance comparison of cuda and opencl. *arXiv preprint arXiv:1005.2581*, 2010. Citado na página 98.

KHRONOS GROUP, K. *Khronos Group: Connecting Software to Silicon*. 2020. (Acesso em: 11 de Nov. de 2020). Disponível em: <https://www.khronos.org/>. Citado na página 93.

LI, B.; SUN, S. X. Coherent motions in confluent cell monolayer sheets. *Biophysical journal*, Elsevier, v. 107, n. 7, p. 1532–1541, 2014. Citado na página 25.

LIBERMAN, A. et al. Cell studio: A platform for interactive, 3d graphical simulation of immunological processes. *APL bioengineering*, AIP Publishing LLC, v. 2, n. 2, p. 026107, 2018. Citado 2 vezes nas páginas 22 e 83.

LIU, T.-L. et al. Observing the cell in its native state: Imaging subcellular dynamics in multicellular organisms. *Science*, American Association for the Advancement of Science, v. 360, n. 6386, 2018. Citado na página 17.

MEMETI, S. et al. Benchmarking opencl, openacc, openmp, and cuda: programming productivity, performance, and energy consumption. In: *Proceedings of the 2017 Workshop on Adaptive Resource Management and Scheduling for Cloud Computing*. [S.l.: s.n.], 2017. p. 1–6. Citado na página 98.

MENZEL, A. M.; OHTA, T. Soft deformable self-propelled particles. *EPL (Europhysics Letters)*, IOP Publishing, v. 99, n. 5, p. 58001, 2012. Citado na página 25.

MICOULET, A.; SPATZ, J. P.; OTT, A. Mechanical response analysis and power generation by single-cell stretching. *ChemPhysChem*, Wiley Online Library, v. 6, n. 4, p. 663–670, 2005. Citado 3 vezes nas páginas 10, 71 e 73.

MIROSHNIKOVA, Y. A. et al. Adhesion forces and cortical tension couple cell proliferation and differentiation to drive epidermal stratification. *Nature cell biology*, Nature Publishing Group, v. 20, n. 1, p. 69–80, 2018. Citado na página 83.

MOREIRA-SOARES, M. et al. Adhesion modulates cell morphology and migration within dense fibrous networks. *Journal of Physics: Condensed Matter*, IOP Publishing, v. 32, n. 31, p. 314001, 2020. Citado 3 vezes nas páginas 26, 27 e 28.

MOUSAVI, S. M.; GOMPPER, G.; WINKLER, R. G. Active brownian ring polymers. *The Journal of chemical physics*, AIP Publishing LLC, v. 150, n. 6, p. 064913, 2019. Citado na página 18.

NEWMAN, T. J. Modeling multi-cellular systems using sub-cellular elements. *arXiv preprint q-bio/0504028*, 2005. Citado 4 vezes nas páginas 18, 19, 26 e 28.

NICKOLLS, J. et al. Scalable parallel programming with cuda. *Queue*, ACM New York, NY, USA, v. 6, n. 2, p. 40–53, 2008. Citado na página 93.

NVIDIA CORPORATION, N. *CUDA C PROGRAMMING GUIDE*. 2018. (Acesso em: 11 de Nov. de 2020). Disponível em: <https://docs.nvidia.com/cuda/archive/9.1/pdf/CUDA_C_Programming_Guide.pdf>. Citado 2 vezes nas páginas 11 e 92.

OURIQUE, G.; TEIXEIRA, E. F.; BRUNNET, L. G. Single cell aspiration simulation. Physica A, Dez. 2021. Citado 2 vezes nas páginas 18 e 29.

RAJA, C. V.; BALASUBRAMANIAN, S.; RAGHAVENDRA, P. S. Heterogeneous highly parallel implementation of matrix exponentiation using gpu. *arXiv preprint arXiv:1204.3052*, 2012. Citado 2 vezes nas páginas 11 e 97.

RICHMOND, P.; CHIMEH, M. K. Flame gpu: Complex system simulation framework. In: IEEE. *2017 International Conference on High Performance Computing & Simulation (HPCS)*. [S.l.], 2017. p. 11–17. Citado na página 22.

SANDERSIUS, S.; WEIJER, C. J.; NEWMAN, T. J. Emergent cell and tissue dynamics from subcellular modeling of active biomechanical processes. *Physical biology*, IOP Publishing, v. 8, n. 4, p. 045007, 2011. Citado na página 27.

SANDERSIUS, S. A.; NEWMAN, T. J. Modeling cell rheology with the subcellular element model. *Physical biology*, IOP Publishing, v. 5, n. 1, p. 015002, 2008. Citado 3 vezes nas páginas 8, 27 e 37.

SAW, T. B. et al. Topological defects in epithelia govern cell death and extrusion. *Nature*, Nature Publishing Group, v. 544, n. 7649, p. 212–216, 2017. Citado 3 vezes nas páginas 8, 17 e 18.

SNIR, M. et al. *MPI–the Complete Reference: the MPI core*. [S.l.]: MIT press, 1998. v. 1. Citado na página 99.

STONE, J. E.; GOHARA, D.; SHI, G. Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, IEEE Computer Society, v. 12, n. 3, p. 66–73, 2010. Citado na página 93.

SZABO, B. et al. Phase transition in the collective migration of tissue cells: experiment and model. *Physical Review E*, APS, v. 74, n. 6, p. 061908, 2006. Citado 2 vezes nas páginas 17 e 25.

TEIXEIRA, E. F.; FERNANDES, H. C.; BRUNNET, L. G. A single active ring model with velocity self-alignment. *Soft Matter*, Royal Society of Chemistry, 2021. Citado 4 vezes nas páginas 8, 18, 28 e 30.

TLILI, S. *Biorhéologie in vitro: de la cellule au tissu*. Tese (Doutorado) — Sorbonne Paris Cité, 2015. Citado na página 19.

TÓTHOVÁ, R.; JANČIGOVÁ, I.; BUŠÍK, M. Calibration of elastic coefficients for spring-network model of red blood cell. In: IEEE. *2015 International Conference on Information and Digital Technologies*. [S.l.], 2015. p. 376–380. Citado 5 vezes nas páginas 28, 30, 32, 51 e 82.

VEDULA, S. R. K. et al. Emerging modes of collective cell migration induced by geometrical constraints. *Proceedings of the National Academy of Sciences*, National Acad Sciences, v. 109, n. 32, p. 12974–12979, 2012. Citado na página 17.

VICSEK, T. et al. Novel type of phase transition in a system of self-driven particles. *Physical review letters*, APS, v. 75, n. 6, p. 1226, 1995. Citado na página 17.

WANG, N.; BUTLER, J. P.; INGBER, D. E. Mechanotransduction across the cell surface and through the cytoskeleton. *Science*, American Association for the Advancement of Science, v. 260, n. 5111, p. 1124–1127, 1993. Citado na página 17.

WEIK, F. et al. Espresso 4.0–an extensible software package for simulating soft matter systems. *The European Physical Journal Special Topics*, Springer, v. 227, n. 14, p. 1789–1816, 2019. Citado na página 25.

WIENKE, S. et al. Openacc—first experiences with real-world applications. In: SPRINGER. *European Conference on Parallel Processing*. [S.l.], 2012. p. 859–870. Citado na página 93.

WU, F.-Y. The potts model. *Reviews of modern physics*, APS, v. 54, n. 1, p. 235, 1982. Citado 2 vezes nas páginas 23 e 25.

ZHAO, J. et al. Dynamic cellular finite-element method for modelling large-scale cell migration and proliferation under the control of mechanical and biochemical cues: a study of re-epithelialization. *Journal of The Royal Society Interface*, The Royal Society, v. 14, n. 129, p. 20160959, 2017. Citado 3 vezes nas páginas 18, 19 e 28.

ZIEBERT, F.; SWAMINATHAN, S.; ARANSON, I. S. Model for self-polarization and motility of keratocyte fragments. *Journal of The Royal Society Interface*, The Royal Society, v. 9, n. 70, p. 1084–1092, 2012. Citado na página 26.

Apêndices

A Anexo

Para um futuro cenário de simulações multicelulares ou de milhares de partículas, para desenvolver a proposta deste trabalho, uma abordagem interessante é desenvolver o *software* da forma mais otimizada e acelerada possível, uma vez que o caso tridimensional engloba mais partículas e cálculos que o caso bidimensional. Sabe-se, a partir da revisão dos *softwares* e *frameworks*, que é possível paralelizar os cálculos das forças entre as partículas nas unidades de processamento de placas de processamento gráfico (GPU). Por isso, realizou-se a essa revisão sobre os métodos disponíveis para a aceleração dos cálculos por GPU.

A.1 Revisão de paralelização de *software*

O tempo de processamento de uma simulação multicelular considerando células pontuais em uma CPU (Unidade Central de Processamento) pode ser da ordem de horas ou dias. Para otimizar esse cenário, uma das principais abordagens é o uso de técnicas de paralelização de *software*. Para isso, pode-se utilizar *clusters* de CPU, ou seja, uma rede distribuída de CPUs onde cada CPU realiza o cálculo de uma parte da simulação. Uma segunda abordagem é o uso de GPUs (Unidades de Processamento Gráfico), ou seja, utilizar o poder de processamento de placas de vídeo para a realização dos cálculos. Pode-se também utilizar um *cluster* de GPUs. E, por último e mais sofisticado, é a utilização de *clusters* de CPU + GPU, em que os cálculos podem ser paralelizados entre as CPUs e GPUs da rede.

Os cálculos feitos em CPU são realizados pelo processador central que, em geral, tem apenas de 2 a 4 núcleos eficientes para cálculos complexos. Deve-se mencionar aqui que atualmente já há processadores com até 32 cores e 64 threads. Devido a essa capacidade, tem grande vantagem em relação à GPUs nos casos em que o algoritmo não pode ser paralelizado.

Mesmo tendo desvantagens em relação à CPU, a GPU pode ter até milhares de núcleos (Asano, Maruyama e Yamaguchi (2009)), permitindo, então, milhares de cálculos paralelos. Um fator limitante da GPU é a quantidade de memória e velocidade de escrita/leitura se comparado à CPU, assim para alcançar a máxima *performance* em algoritmos em GPU, é necessário levar em consideração essas limitações na construção do código e escolha do algoritmo. É necessário que o código seja bem estruturado e distribuído entre os núcleos requerendo que se utilizem plataformas de computação paralela, linguagens e instruções de programação específicas para a paralelização.

Essas características tornam a GPU atrativa para a paralelização dos cálculos feitos nas

diversas partículas do sistema. Então, a seguir, é feita uma revisão bibliográfica com o objetivo de validar a possibilidade de paralelização do código.

A.1.1 Paralelização em GPUs

A taxa superior da evolução do poder de processamento das GPUs em relação às CPUs nos últimos anos chamou atenção da comunidade científica e de outras áreas do conhecimento não voltadas ao processamento gráfico. Na figura 62 pode-se ver essa evolução do ponto de vista dos GFLOPS (operações de ponto flutuante por segundo) que, em geral, representa bem esse quadro. Essa característica abriu a possibilidade da utilização de GPUs, antes dedicadas para o processamento gráfico, para outras aplicações que abrangem inteligência artificial, simulações físicas, visão computacional etc. A esse tipo de aplicação se dá o nome de computação de propósito geral em unidades gráficas de processamento (GPGPU).

Na figura 62, percebe-se que as GPUs e CPUs de precisão única tem uma taxa de crescimento maior do que as de precisão dupla, que oferecem maior precisão nos cálculos e são mais adequadas para GPGPUs, mas ao mesmo tempo consomem maior processamento. Com base no crescente interesse do mercado por GPGPUs, se iniciou a produção de uma nova série de placas dedicadas a GPGPU, a exemplo do modelo NVIDIA Tesla que, em comparação aos modelos comuns, possui maior poder de processamento para precisão de operações com ponto flutuante (dupla precisão), maior espaço e velocidade de acesso à memória, possui unidades de processamento voltadas para *deep learning* e outras aplicações.

Entretanto, nem sempre as GPUs aceleram os cálculos de *software*. Como lidam com paralelização de operações, é necessário que as operações realizadas sejam independentes, o que não é possível em muitas aplicações. Para exemplificar, abaixo tem-se uma função que realiza operações em duas entradas ***a*** e ***b***. percebe-se que o cálculo da linha 4 depende do cálculo da linha 3. Logo, o código abaixo é não paralelizável.

```
1    function dependence(a, b)
2    {
3        c = a + b
4        d = 5 * c
5    }
```

Código 1: Exemplo de código não paralelizável.

No código exemplo 2, todas as linhas da função são independentes, apenas dependem das entradas **a** e **b** da função, assim é possível paralelizar o código.

```
1    function no_dependene(a, b)
2    {
3        c = a + b
4        d = 5 * a
```

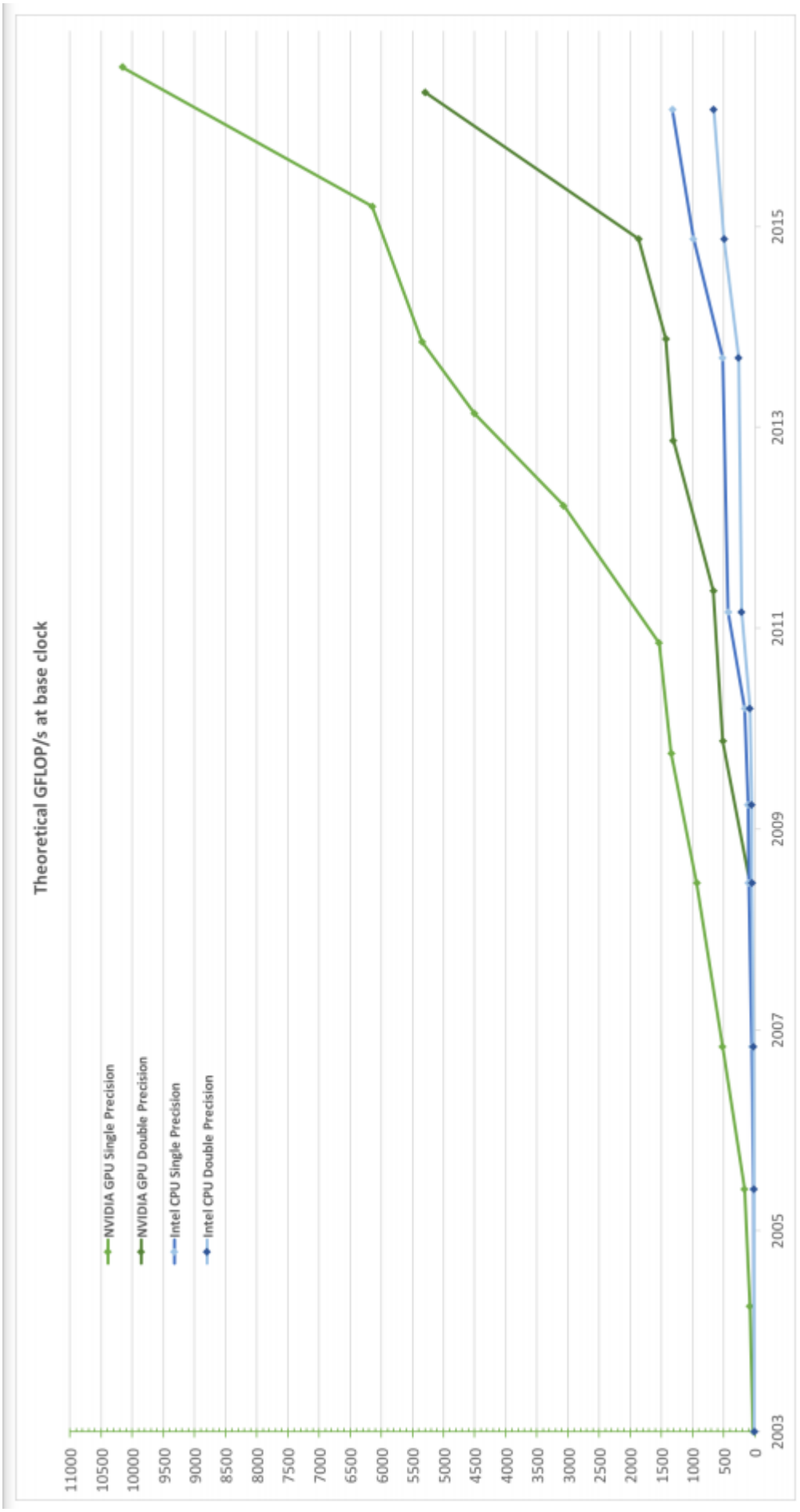


Figura 62 – Comparação da evolução do poder de processamento entre GPUs e CPUs ao longo do tempo (NVIDIA Corporation (2018)).

```
5           e = 8 * b
6           f = a * b
7       }
```

Código 2: Exemplo de código paralelizável.

A paralelização pode ser uma boa abordagem para simulações de dinâmica molecular, se as forças forem de curto alcance, pois a posição futura de cada partícula do sistema depende apenas da posição atual de cada partícula (que são as entradas da função) e da posição de suas vizinhas.

Existem diversas plataformas de computação paralela que possibilitam que códigos sejam paralelizados em GPU (neste trabalho essa nomenclatura é usada para englobar os métodos que variam de APIs a linguagens de programação dedicadas para a paralelização de códigos). As três plataformas mais conhecidos são **OpenCL** (*Open Computing Language*) - (Stone, Gohara e Shi (2010)), **CUDA** (*Computer Unified Device Architecture*) - (Nickolls et al. (2008)) e **OpenACC** (for Open Acceleratos) - (Wienke et al. (2012)).

A programação acelerada por GPU, diferente da programação serial em CPU, requer que o programador tenha conhecimentos da arquitetura do *hardware*, especificações, tempo de escrita e leitura de dados, da linguagem de programação e arquitetura da plataforma de computação paralela utilizada.

A.1.2 Paralelização em OpenCL

OpenCL é um padrão aberto de programação de baixo nível de alta performance para sistemas heterogêneos (Khronos Group (2020)). O OpenCL permite que o programa seja paralelizado e distribuído nos *cores* de uma placa de processamento gráfico e, por ser de baixo nível, permite a manipulação da escrita e leitura dos dados entre a GPU e a CPU e dentro da GPU. A heterogeneidade se refere ao suporte a diversas plataformas diferentes como, por exemplo, permite que os códigos sejam paralelizados em GPUs, CPUs, FPGAs, *mobiles* entre outras. A programação é feita em C e C++. O OpenCL oferece diferentes versões de funções nativas matemáticas que facilitam as operações em GPU. Algumas funções tem precisão única (mais rápidas, porém menos precisas) e de precisão dupla, de forma que o programador pode otimizar o código dependendo da sua aplicação.

Nem sempre a paralelização e o poder de processamento aumentam a velocidade de execução de um código. Por exemplo, sabe-se que o tempo para acesso de memória de um dado da CPU para a GPU é maior do que o tempo de execução de uma operação matemática simples (em torno de 30 vezes mais). Pode-se explicar esse comportamento a partir do código abaixo em que uma função recebe um *array* $\boldsymbol{A}$ onde cada operação no *array* é feito em uma unidade de processamento da GPU.

```
1   function prod(array A)
```

```
2    {
3        temp+= A[i] + A[i]
4        A[i] = exp(temp)*log(temp)
5    }
```

Código 3: Exemplo de código com escrita e leitura de memória.

Neste exemplo, é necessário ler a variável ***A[i]***, na linha 3, realizar a operação de soma e guardar o resultado na variável ***temp***. Na linha 4, aplicar operações matemáticas exponenciais e logarítmicas e escrever novamente o resultado na variável ***A[i]***. Ou seja, 2 operações de memória (lentas) e diversas operações matemáticas (rápidas). Como resultado, por mais que sejam adicionados mais *cores* para paralelizar os cálculos matemáticos, não será possível aumentar o tempo de escrita/leitura. Então, ao chegar na máxima *performance* de aceleração das operações, o gerenciamento de memória se tornará o gargalo de velocidade do código. Esse é o motivo de muitos códigos não terem maior velocidade em GPU se comparados a CPU. Por isso, torna-se tão necessário o conhecimento dos tempos de memória e especificações da GPU para gerenciamento dos dados e otimização do código.

Todas essas características devem ser levadas em consideração ao programar dentro da plataforma OpenCL, que possui a sua própria estrutura interna de organização, a qual deve-se entender em detalhe.

A.1.2.1 Arquitetura OpenCL

Todo conjunto de operações que é executado diversas vezes em diferentes *cores* GPU, paralelamente, é chamado de ***thread*** ou, especificamente em OpenCL, de **item de trabalho**. Exemplo: em dinâmica molecular é comum que seja calculado o módulo da distância de uma partícula ***i*** de todas as outras partículas ***j*** do sistema e sejam aplicadas operações matemáticas nesse resultado. Essa mesma operação é realizada para cada uma das partículas, ou seja, é o mesmo código sendo executado diversas vezes em entradas diferentes, então pode ser entendido como uma *thread*.

Em OpenCL as *threads* são organizadas em um espaço maior chamado de dimensão global que pode ter até 3 dimensões em que cada ponto da dimensão representa uma *thread*. Exemplo: Caso se queira renderizar um vídeo de resolução 1920x1080 *pixels* em uma GPU, será necessário uma *thread* para cada pixel, então, pode-se organizar as *threads* em uma dimensão global de 2 dimensões de 1920x1080.

As dimensões globais podem ser decompostas em **dimensões locais**, também conhecidas como **grupos de trabalho locais** quem tem até 3 dimensões. Essa organização é extremamente importante, pois cada grupo de trabalho é executado na mesma unidade de processamento. E isso permite que as *threads* dentro da mesma unidade de processamento sejam sincronizadas, então possibilitando que o programador force que partes do código que sejam executadas depois

de alguma outra parte no mesmo grupo de trabalho. Apenas *threads* do mesmo grupo de trabalho podem ser sincronizadas.

Exemplo: na figura 63 são criados 2 grupos de trabalho que agrupam 2 *threads* cada (o grupo **A** com as *threads* **I** e **II** e o grupo **B** com as *threads* **III** e **IV** - caixas cinzas). Para realizar a operação de soma, por exemplo, entre os dados de entrada **1** e **2** (caixas azuis), utiliza-se a *thread* **I** (caixa cinza) na linha 1 e assim por diante. Após computar os dados das *threads* **I** e **II** na linha 1, utiliza-se novamente a *thread* **I** na linha 2 para computar os resultados, entretanto, a *thread* **I** na linha 2 só deve ser executada após as threads **I** e **II** na linha 1, senão o cálculo será feito com um número qualquer alocado na memória da GPU. Para resolver esse problema, basta sincronizar o tempo de execução da thread **I** na linha 2 com as *threads* anteriores. Entretanto, não pode ser feito o mesmo com a *thread* **II** da linha 2 porque ela está em um grupo de trabalho diferente das *threads* **III** e **IV**.

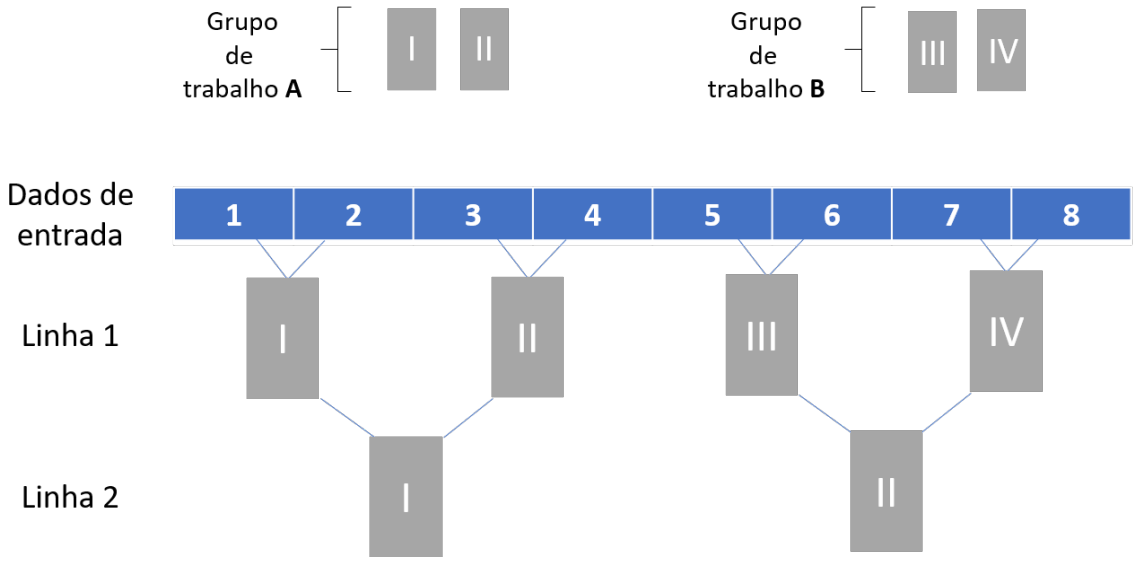


Figura 63 – Sincronização em grupos de trabalho no OpenCL.

Ademais, o dimensionamento correto dos grupos de trabalho permitem que o uso de processamento da GPU seja ótimo. Caso contrário, pode-se deixar *cores* ociosos durante os cálculos. Por exemplo, caso se tenha uma GPU com 16 *cores* por unidade de processamento e sejam dimensionados grupos de trabalho com 52 *threads*, parte dos *cores* não serão utilizados, pois como há 16 *cores* na unidade, os processos serão executados em grupos de 16 *threads* e, então, após 3 iterações, 48 *threads* serão executadas. Restando apenas 4 das 52 totais. Então, na quarta iteração, apenas 4 *cores* dos 16 disponíveis serão utilizados deixando 12 *cores* ociosos. Estarão sendo utilizados 81% dos cores da GPU, o que ocasionará um baixo desempenho da velocidade do código. Assim como o dimensionamento incorreto da dimensão global pode fazer com que apenas parte dos dados sejam executados.

Por último, e bem importante para ganho de velocidades em códigos executados em GPU é o gerenciamento da escrita/leitura de dados na memória. A transferência de dados da CPU para a GPU e entre as memórias internas da GPU para as unidades de processamento é feita

manualmente pela programação do OpenCL. Para se obter um código otimizado, é desejado o mínimo de transferência de dados entre esses componentes uma vez que a transferência de dados é muito mais lenta que os cálculos simples feitos pelas unidades de processamento. Entretanto, como não é possível eliminar as transferências, se deseja que sejam feitas as melhores escolhas de transferência na arquitetura interna. Para isso, é necessário conhecer os tipos de memórias existentes e a velocidade de transferência entre elas.

Em geral, existem 5 tipos de memórias para o gerenciamento dos dados com velocidade de escrita/leitura distintas (Black-Schaffer (2016)):
1. Memória global da CPU (RAM): é a memória que armazena dados de todos os processos e pode ser acessada por todos. A velocidade de acesso é de 5-10 GB/s, sendo a mais lenta .
2. Memória constante da GPU: armazena dados que são compartilhados por todos os itens de trabalho no modo de leitura. Tem velocidade de acesso de 10-50 GB/s.
3. Memória global da GPU: é semelhante à memória constante, mas tem também escrita. Velocidade de acesso de 50-200 GB/s.
4. Memória local: memória de leitura e escrita compartilhada entre os itens de trabalho de um mesmo grupo. Velocidade de acesso de 1000 GB/s.
5. Memória privada: são registradores que são acessíveis apenas pelos itens de trabalho. Com velocidades acima de 1000 GB/s.

A memória global da CPU (em vermelho, na caixa *Compute Device Memory* da figura 64) pode armazenar os dados de todos os processos do computador. As outras 4 memória estão localizadas dentro da GPU (caixa *Compute Device*). O *design* do *hardware* é feito disponibilizando-se diferentes velocidades e permissões de acesso dependendo da memória. A memória constante (em amarelo) armazena os dados que são acessados apenas por leitura por todos os grupos de trabalho, sendo, em relação às outras da GPU, a mais lenta. A memória global (em amarelo também) tem função semelhante à memória constante, mas permite também escrita. Os itens de trabalho de um mesmo grupo podem acessar a memória local (em roxo) que, por ser dedicada ao grupo, tem velocidade de acesso até 100x maior do que a constante. E, por último, existem as memórias que são acessadas apenas pelos itens de trabalho. São os tipos de memórias mais rápidas, entretanto, por serem, na prática, registradores, possuem muito pouca capacidade de armazenamento.

Pelas diferentes velocidades de acesso de memória, tipos de acesso e pelos acessos serem mais lentos que os cálculos feitos pelas unidades de processamento, torna-se necessário que o programador projete o seu algoritmo de forma a minimizar o tráfego entre os dados e utilize os tráfegos mais rápidos oferecidos, além de levar em consideração o tamanho das memórias usadas que dependem da GPU disponível.

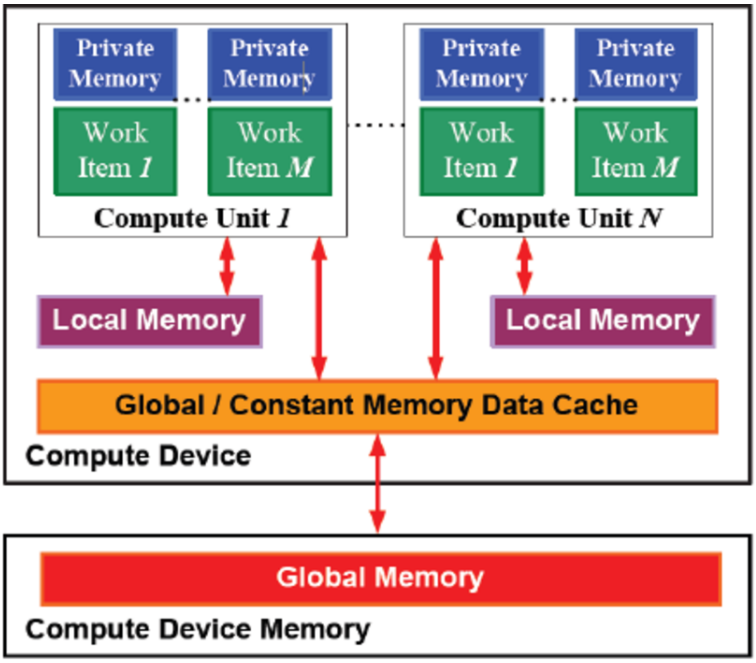


Figura 64 – Modelo de gerenciamento de memória pelo OpenCL (Raja, Balasubramanian e Raghavendra (2012)).

A.1.3 Paralelização em CUDA

Compute Unified Device Architecture (CUDA) é uma arquitetura de programação de computação paralela de propósito geral dedicado para placas NVIDIA. Apesar de não ser multiplataforma como o OpenCL, CUDA possui suporte a diversas linguagens de programação como C, C++, Fortran, DirectCompute e suporte de *wrappers* para Python e Java. Em geral, a tecnologia CUDA possui maior flexibilidade de programação em relação ao OpenCL e maior velocidade de transmissão de dados de memória, fator importante quando se trata de programação em GPUs.

A arquitetura de CUDA é bem semelhante ao OpenCL, então esses detalhes serão omitidos. Na tabela 1 é possível observar a equivalência dos termos em OpenCL para a arquitetura CUDA.

OpenCL	CUDA
Unidade de processamento	Streaming Multiprocessor (SM)
host	host
kernel	kernel
Item de trabalho	thread
Grupo de trabalho	Bloco
Dimensão global	*Grid*

Tabela 1 – Equivalência de termos entre OpenCL e CUDA.

Em relação à velocidade, em um teste de *performance* feito com uma simulação de Monte

Carlo conhecida como Algoritmo Quântico Adiabático (Karimi, Dickson e Hamze (2010)) a performance de OpenCL ficou entre 13% e 67% mais lento do que CUDA. Foram testados os tempos de transferência em memória onde CUDA também teve maior velocidade. Entretanto, dependendo do tipo de aplicação, OpenCL pode ter uma *performance* até melhor do que CUDA, apesar de CUDA, em geral, ter uma *performance* melhor Memeti et al. (2017).

A.1.4 Paralelização em OpenACC

OpenACC (*for Open Accelerators*) é um padrão de programação de alto nível para programação paralela. A principal diferença entre OpenACC e as outras plataformas é a facilidade com que OpenACC permite que códigos sejam paralelizados. Não é necessário especificar os processos que serão iniciados, dimensionar a arquitetura do código nem especificar as transferências de memória como nas plataformas de computação paralela citadas anteriormente. O OpenACC trabalha com diretivas, na mesma lógica de variáveis em linguagens de programação. Por exemplo, para definir um *inteiro* na linguagem C é necessário escrever a diretiva *int* antes do nome da variável para o compilador entender que será declarado um número inteiro. O OpenACC utiliza a mesma lógica para a paralelização, ou seja, basta digitar diretivas nas partes do código que se deseja paralelizar que o compilador se encarrega do gerenciamento e arquitetura de paralelização. Essa facilidade de programação paralela torna fácil o aprendizado e diminui bastante o número de linhas do código. Quantitativamente, um programa em OpenACC requer, em média, 6.7 vezes menos esforço do que programar em OpenCL, enquanto que OpenCL requer 2 vezes mais esforço do que programar em CUDA (Memeti et al. (2017)). Sendo o esforço uma relação entre o número de linhas relacionados à arquitetura da plataforma comparado ao número de linhas totais do programa.

Essa característica permite que programas seriais sejam facilmente paralelizáveis. Entretanto, apesar da maior facilidade de programação, o OpenACC abstrai a arquitetura, tendo menos controle dos recursos de otimização e, por isso, é a plataforma que tem o pior desempenho em relação à velocidade entre OpenCL e CUDA. Apesar disso, o OpenACC vem sendo atualizado e atualmente dispõe de ferramentas opcionais para o controle de memória. Para se ter uma ideia quantitativa, foi feito um teste de *performance* com 3 diferentes algoritmos em OpenCL e OpenACC, o OpenCL teve desempenho de 5 a 20 vezes maior nos primeiros dois algoritmos, enquanto que no terceiro algoritmo teve desempenho semelhante ao OpenACC Memeti et al. (2017). Outra desvantagem do OpenACC é que o seu compilador oficial é pago.

A *performance* das diferentes plataformas é muito dependente do algoritmo e das otimizações feitas na programação considerando a arquitetura da mesma, podendo em alguns casos a plataforma mais lenta ter um desempenho mais rápido que os outras que são mais rápidas na média. Entretanto, em geral, pode-se considerar que a performance média entre as três plataformas mencionadas é a seguinte: CUDA tendo a maior velocidade seguido do OpenCL e, por fim, o OpenACC.

Na tabela A.1.4 se tem o resumo da comparação entre as três plataformas pesquisadas. As colunas de *performance* e esforço de programação são descritas de forma comparativa entre as três.

Plataforma	Multiplataforma	Linguagens de Programação	Compilador	*Performance* Média	Esforço de Programação
OpenCL	Sim	C/C++	*Free*	Média	Maior
CUDA	Não	C/C++/ Fortran/ Python e Java *Wrappers*/ DirectCompute	*Free*	Maior	Médio
OpenACC	Sim	C/C++/Fortran	Pago	Menor	Menor

Tabela 2 – Comparação entre as plataformas de computação paralela OpenCL, Cuda e OpenACC.

Baseado nesse contexto, o OpenCL é uma boa recomendação para o desenvolvimento de trabalhos de simulação, pois é multiplataforma, o que possibilita que os cálculos sejam expandidos para outros sistemas mantendo o mesmo código, tem o compilador oficial não pago e uma performance média, o que possibilita um bom cenário para a proposta multicelular futura.

Por fim, abrindo possibilidades futuras para implementações que suportem o cenário multicelular como o uso de um *cluster* de GPUs ou GPUs + CPUs, fez-se uma breve revisão dos requisitos necessários. Para isso, necessita-se que o OpenCL se conecte entre os diversos dispositivos em uma rede distribuída e, para isso, se torna necessário um protocolo de comunicação que paraleliza os códigos na rede. Uma ferramenta muito utilizada para esse propósito é o *Message Passing Interface* (MPI) que faz a comunicação em redes distribuídas.

A.1.5 Message Passing Interface (MPI)

Message Passing Interface (MPI) é um padrão de comunicação para computação paralela em redes distribuídas (Snir et al. (1998)). Ele permite que dados sejam trocados entre dispositivos CPU de uma rede, conhecidos como nós, a partir das linguagens C, C++ e Fortran. É possível utilizar versões alternativas do MPI para *Python*, *R* e outras linguagens de programação.

Não apenas dados podem ser enviados como também programas podem ser executados nas CPUs da rede de forma paralela. Por exemplo, na mesma ideia de paralelização dos cálculos em dinâmica molecular mencionados anteriormente, onde é possível calcular as posições das partículas do sistema entre vários *cores* de uma GPU, é possível também, com o uso do MPI,

paralelizar os mesmos cálculos nas CPUs da rede. Uma vez que a ponte de comunicação é feita pelo MPI, pode-se utilizá-lo em conjunto com o OpenCL para a criação do *cluster*.

Praticamente, um programa MPI envia diversos processos ou comandos diferentes de uma máquina, considerada como pai, para as outras máquinas, que são consideradas os filhos. Os filhos, ao receber o processo, podem responder através do envio de outro processo para o pai. Cada processo tem uma ID de identificação que possibilita o mapeamento dos dados. O MPI possui algumas diretivas básicas que permitem que sejam enviados e recebidos dados, que sejam contados os números de processos, inicialização e saída do ambiente MPI.